Computing the Medial Axis of a Polyhedron
Reliably and Efficiently

by

Tim Culver

A Dissertation submitted to the faculty of The University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill

2000

Approved by:

Dinesh Manocha, Advisor

Stephen Pizer, Reader

Guido Gerig, Reader
TIM CULVER. Computing the Medial Axis of a Polyhedron Reliably and Efficiently.
(Under the direction of Dinesh Manocha.)

ABSTRACT

The medial axis transform is a fundamental shape operation with applications in many fields. In solid modeling, the MAT has proven a useful tool for finite element meshing, model simplification, and feature recognition. The MAT is also a complete shape representation that could be used in place of a boundary representation. Yet the MAT is not widely used because computing it is difficult both in theory and in practice. For a three-dimensional polyhedral solid, the medial axis consists of quadric surfaces and degree-four algebraic space curves. Computing with high-degree curves and surfaces requires high numerical precision. Most previous methods attempt to avoid such computation by discretizing, or otherwise approximating, the medial axis. The few existing continuous methods are based exclusively on floating-point arithmetic, which leads to reliability problems.

I present a new reliable, continuous algorithm for accurately computing the medial axis of a polyhedron. It is the only continuous medial axis algorithm that is insensitive to roundoff error. Further, my algorithm handles the most common forms of degeneracy. The algorithm is also efficient in a practical sense. The foundation of my approach is exact computation. My MAT representation uses arbitrary-precision rational numbers to describe the medial geometry. My algorithm is based on a point-ordering predicate that is always evaluated correctly.

By its nature, exact computation requires high-precision arithmetic, which is significantly more expensive than hardware-supported floating-point arithmetic. However, my approach avoids the extra expense where feasible, using techniques such as floating-point filters and lazy evaluation. The result is an algorithm whose running time approaches that of floating-point methods when high precision is not required. I demonstrate this assertion by applying my implementation to several complex polyhedral solids.
To Kathleen
and my parents
Acknowledgments

This research was funded by the Army Research Office, the Department of Energy, Intel, the National Science Foundation, and the Office of Naval Research. Many of the models are courtesy of the University of Washington Computational Design and Manufacturing Lab and the NIST Design Repository.

Thanks to my advisor, Dinesh Manocha, for his guidance and his patience over the last four and a half years. Thanks also to my collaborators: Shankar Krishnan, Mark Foskey, and especially John Keyser, with whom I had many—possibly hundreds—of fruitful discussions in the course of this research. I appreciate the contributions of my committee: Pankaj Agarwal, Guido Gerig, Ming Lin, and Steve Pizer.

My deepest gratitude goes to the three people who have contributed the most to helping me reach this point. My parents have encouraged my interest in math and computers from the earliest stages. Thank you, Mom and Dad. Finally, my wife Kathleen has supported me with kindness and patience through my graduate studies. Thank you, Kathleen.
Contents

Acknowledgments ix

List of Tables xv

List of Figures xvii

1 Introduction 1
  1.1 Problem definition ............................................. 3
  1.2 Combinatorial complexity ........................................ 5
  1.3 Previous approaches ........................................... 8
  1.4 Robustness .................................................... 9
  1.5 Thesis ....................................................... 13
  1.6 New results ................................................... 14
  1.7 Overview of chapters ........................................... 19

2 Related Work 21
  2.1 Applications of the medial axis transform ..................... 21
  2.2 Problems related to polyhedral medial axis computation ....... 22
  2.3 Medial axis algorithms based on discretization ............... 25
  2.4 Continuous medial axis algorithms ............................ 27
  2.5 Robust geometric computation .................................. 32
  2.6 Summary ..................................................... 39

3 An Accurate Medial Axis Algorithm 41
  3.1 Terminology .................................................. 41
  3.2 Overview of the accurate tracing algorithm .................... 42
  3.3 Exact arithmetic .............................................. 45
  3.4 Exact geometric representations ................................ 48
  3.5 Technical assumptions ........................................ 49
## 4 Distance Functions and Bisectors

4.1 Distance functions ................................................. 51
4.2 Domain polytopes ................................................ 53
4.3 Bisectors .......................................................... 57
4.4 Summary .......................................................... 66

## 5 Junctions, Seams, and Sheets

5.1 Junctions .......................................................... 67
5.2 Seams and tracing ............................................... 71
5.3 Sheets ............................................................ 75

## 6 Efficiency Improvements

6.1 Alleviating the combinatorial penalty ....................... 79
6.2 Reducing the number of kernel operations ................. 87
6.3 Speeding up the kernel operations ......................... 87

## 7 Degeneracies

7.1 The degenerate junction ....................................... 93
7.2 The degenerate seam .......................................... 94
7.3 Incidental degeneracies ..................................... 95

## 8 Determinant Sign Evaluation

8.1 Introduction .................................................... 97
8.2 Determinant signs in non-linear computation ............ 99
8.3 Modular arithmetic ............................................ 101
8.4 A floating-point filter ....................................... 102
8.5 Multiprecision interval arithmetic ......................... 106
8.6 Empirical results .............................................. 108
8.7 Hybrid methods ................................................ 114
8.8 Conclusions ..................................................... 115

## 9 Implementation and Results

9.1 Implementation .................................................. 117
9.2 Results and examples ......................................... 119
9.3 Analysis ........................................................ 121
10 Conclusions and Future Work 129
  10.1 Summary ........................................ 129
  10.2 Discussion ...................................... 131
  10.3 Future work .................................... 133

A Algorithmic Algebra 135
  A.1 Resultants ...................................... 135
  A.2 Univariate Sturm sequences .................. 139
  A.3 Multivariate Sturm sequences ............... 140
  A.4 References .................................... 143

Bibliography 145
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Performance on some example polyhedra.</td>
<td>15</td>
</tr>
<tr>
<td>5.1</td>
<td>Some curves that arise as seams in the medial axis.</td>
<td>72</td>
</tr>
<tr>
<td>9.1</td>
<td>Performance on some example polyhedra.</td>
<td>120</td>
</tr>
<tr>
<td>9.2</td>
<td>Average timing for quadric-quadric-quadric 3-D Sturm operations in</td>
<td>121</td>
</tr>
<tr>
<td></td>
<td>the Venus model.</td>
<td></td>
</tr>
<tr>
<td>9.3</td>
<td>Breakdown of time spent in the most expensive routines.</td>
<td>126</td>
</tr>
<tr>
<td>9.4</td>
<td>The effect of two of the efficiency techniques on running times for</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>some of the example polyhedra.</td>
<td></td>
</tr>
</tbody>
</table>
List of Figures

1.1 A simple polyhedron and its medial axis. ........................................ 1
1.2 A schematic of the medial axis of a simple polyhedron. .................. 2
1.3 The medial axis of a polygon, illustrating a few maximal inscribed circles. 4
1.4 The medial axis (left) and Voronoi diagram (right) of a polygon. ..... 6
1.5 A polyhedron whose medial axis has $\Theta(n^2)$ combinatorial complexity. 6
1.6 Another polyhedron whose medial axis has $\Theta(n^2)$ complexity. More surprisingly, it has $\Theta(n^2)$ sheets governed by the same two faces. ... 7
1.7 Roundoff error causes a catastrophic failure of a polygon intersection algorithm. ................................................................. 10
1.8 A simple non-convex polyhedron: an L-shaped box. ......................... 16
1.9 A polyhedron with two grooves and its medial axis. ...................... 16
1.10 The “iron maiden pizza box” and a schematic of its medial axis. ...... 17
1.11 A CAD model of a bracket and a schematic of its medial axis. ........ 17
1.12 A model of the Venus de Milo sculpture. .................................. 18
3.1 The geometric elements of the medial axis. .................................. 42
3.2 A simple box with a degenerate medial axis. ................................ 43
3.3 An illustration of a step in the seam-tracing algorithm. .................. 44
3.4 (a) The lines $y = 0$ and $y = x$. (b) Their bisector is the pair of lines represented by $-x^2 + 2xy + y^2 = 0$, an equation that does not factor over the rationals. ............................................................. 46
3.5 The medial axis of a near-regular hexagon. The central region is highly magnified to reveal its combinatorial structure. .................... 47
4.1 Domain polytopes in 2-D. ....................................................... 55
4.2 Polytopes containing the Voronoi regions of a face (top row), an edge (middle row), and a vertex (bottom row). ............................. 56
4.3 The bisectors that arise in the medial axis. .................................. 59
4.4 The generic edge-edge bisector is a hyperbolic paraboloid. ............ 61
4.5 The generic vertex-face bisector is a circular paraboloid. ............... 62
4.6 The generic vertex-edge bisector is a parabolic cylinder. ............... 63
4.7 The generic face-face bisector is a plane. ................................... 65
4.8 The generic edge-face bisector is an elliptical cone. .......... 66

5.1 Testing seam validity at a generic junction (in two dimensions). ... 68
5.2 Testing seam validity at a degenerate junction (in two dimensions). 69
5.3 The output of the curve topology algorithm. .................. 74
5.4 A polyhedron with a protosheet containing $\Theta(n^2)$ sheets. .... 77

6.1 Criterion 3 cannot use $g_1$ to eliminate $g_2$ from the governor list of $C$ unless the cell lies entirely within the domain polytope of $g_1$. .... 84
6.2 The cells corresponding to a junction can be many and disconnected. 84
6.3 If the polytope exclusion rule is applied too early, cells outside the polyhedron are unnecessarily kept. .................. 86
6.4 The Gershgorin disks in the complex plane can be used to gauge the reliability of eigenvalue signs. .................. 92

7.1 The two common forms of medial axis degeneracy. .......... 94
7.2 A degree-four seam curve with a singular point. .......... 96

8.1 Curve-line intersection. .......................... 100
8.2 Curve-curve root projection. .......................... 100
8.3 Algebraic and numerical complexity of the geometric problems used to generate the test matrices. .......................... 109
8.4 Well-conditioned matrices. .......................... 111
8.5 Ill-conditioned, but nonsingular, matrices. .................. 112
8.6 Singular matrices. .......................... 112
8.7 Random matrices with uniformly-distributed 100-bit entries. .... 113

9.1 The major components of the medial axis algorithm. ........ 118
9.2 A simple non-convex polyhedron. .......................... 120
9.3 A simple non-convex polyhedron: an L-shaped box. ........ 121
9.4 A polyhedron with two grooves. .......................... 122
9.5 The "iron maiden pizza box" and a schematic of its medial axis. ... 122
9.6 A CAD model of a bracket and a schematic of its medial axis. .... 123
9.7 A CAD model and a schematic of its medial axis. ........ 123
9.8 An irregularly-faceted torus and a schematic of its medial axis showing the central seams. ........ 123
9.9 A model of the Venus de Milo sculpture. .................. 124
A.1 The two-dimensional volume function $V(u)$ associated with the point $(\alpha, \beta)$ at the crosshairs has roots at $u = -1, 8, \text{ and } 9$.  

A.2 Bivariate Sturm sequences via the volume function.
Chapter 1

Introduction

The medial axis transform is a fundamental shape operation with applications in many fields. It was originally proposed by Blum [7] in the 1960s for biological shape measurement. Since then, the medial axis and other medial surface techniques have proven useful in shape representation, shape analysis, and image segmentation and registration. In solid modeling, the medial axis transform has proven a useful tool for many different problems, such as finite element meshing, model simplification for engineering analysis, shape morphing, and feature recognition. The medial axis transform is a complete shape representation. As such it could be used as an alternative to the traditional boundary and constructive solid geometry (CSG) representations.

The *medial axis* of a shape in Euclidean space is the locus of centers of maximal inscribed spheres. For most three-dimensional shapes, the medial axis is a surface. Since the axis lies at the center of the shape and has the same topology, it is also known as the *skeleton* of the shape. Figures 1.1 and 1.2 show examples.

The *medial axis transform* (MAT) is defined by associating to each axis point $p$

![Figure 1.1: A simple polyhedron and its medial axis.](image-url)
the radius $r(p)$ of the maximal inscribed sphere. The word "transform" in the name is justified, since the original shape is reconstructed by taking the union of all the spheres. The MAT is thus a complete shape representation, describing an object by a lower-dimensional skeleton together with a local thickness. As such, it can be considered as an alternative to a boundary representation, which has no notion of "local thickness."

Yet the adoption of the 3-D medial axis transform in solid modeling systems and other applications of computational geometry has been slow. Use of the MAT has been hindered by the difficulty of designing and implementing a reliable algorithm for computing the MAT of a solid represented by its boundary surface.

Medial axis computation is difficult for two reasons. First, the medial axis is combinatorially complex: its structure can be $O(n^2)$, where $n$ is the boundary complexity. Even a theoretically optimal algorithm would have limited applicability.

The other difficulty is that the medial axis is also geometrically complex. The medial surface typically has higher algebraic degree than the boundary. Computing with curves and surfaces of high algebraic degree requires high numerical precision. In fixed-precision arithmetic, such as floating-point arithmetic, numerical error is magnified greatly by algebraic degree, making it difficult to design a reliable algorithm.

In this dissertation I present a new algorithm for accurately computing the medial axis of a polyhedron. I focus mainly on the geometric complexity of the problem, rather than on the combinatorial complexity.
1.1 Problem definition

For a closed polyhedron in $\mathbb{R}^d$, the medial axis is the locus of centers of maximal inscribed $d$-spheres. An inscribed sphere is maximal if no other inscribed sphere contains it. Another characterization of a maximal inscribed sphere is that it meets the boundary in at least two points, called its footprints. Figure 1.3 demonstrates the definition for $d = 2$.

The boundary of a polygon in $\mathbb{R}^2$ consists of edges and vertices, collectively called boundary elements. The medial axis of a polygon consists of line segments and parabolic arcs. Most medial axis points have two footprints, and locally, as a point $x$ moves along the medial axis, each of its footprints either moves along an edge or stays fixed at a vertex. The two closest boundary elements are called governors. If the governors are two edges or two vertices, $x$ traces out a line. If one governor is an edge and the other a vertex, $x$ moves along a parabola. The line or parabola, when treated as an algebraic curve, is called the bisector of the governors. In general, the bisector of two boundary elements is the set of points equidistant from the two elements. In summary, the medial axis of a polygon in the plane is a planar graph whose edges are line segments and parabolic arcs.

For a polyhedron in $\mathbb{R}^3$, the boundary elements are the faces, edges, and vertices. As in two dimensions, the generic medial axis point $x$ lies on the bisector of its two governors. In 3-D, the bisectors are algebraic surfaces, so the polyhedral medial axis is locally a surface.¹ Algebraically, the bisectors are quadrics or planes. These medial surface elements are called sheets. The sheets meet along curves called seams, which are algebraic space curves of degree at most four. The seams come together at junction points. Sheets, seams, and junctions are indicated on figure 1.2.

The Voronoi diagram. The medial axis of a polyhedron is closely related to the Voronoi diagram of the polyhedron. The Voronoi diagram is a fundamental construction in computational geometry. Informally, the Voronoi diagram captures the proximity structure of a collection of elements in space. The notion of "proximity" for the Voronoi diagram is related to the notion of "local thickness" for the medial axis.

More formally, consider a fixed, finite set of points $\{s_i\}$ in Euclidean space. The

¹The medial axis of a polyhedron would, therefore, more accurately be called the medial surface. Some writers have adopted this terminology.
Figure 1.3: The medial axis of a polygon, illustrating a few maximal inscribed circles. All such circles touch the boundary twice. A few, such as the rightmost circle, touch three times.

points $\{s_i\}$ are called sites. An arbitrary point $p$ is said to lie in the Voronoi region of $s_i$ if $p$ is closer to $s_i$ than to any other site $s_j$. The Voronoi regions tessellate space into convex cells. Some points $p$ may have two or more closest sites, in which case it lies on the Voronoi diagram. The Voronoi diagram divides space into regions according to the closest site.

In the foregoing definition, the point-sites $\{s_i\}$ may be replaced by line segments, polygons, or any other geometric elements $\{S_i\}$, yielding the generalized Voronoi diagram. Instead of measuring the distance from $p$ to $s_i$, one simply takes the distance from $p$ to the element $S_i$. The generalized Voronoi diagram is still a spatial tessellation, though the cells are no longer necessarily convex. The Voronoi diagram of a polyhedron is simply the generalized Voronoi diagram whose sites are the vertices, edges, and faces.

For a polyhedron, the medial axis is a subset of the Voronoi diagram of the polyhedron. A point on the medial axis has two or more closest sites, since the two footprints cannot lie on the same face, edge, or vertex. Therefore, a medial axis point is on the Voronoi diagram. A Voronoi diagram point is usually, but not always, on the medial axis. A Voronoi point $x$ on the Voronoi diagram may be equidistant from two boundary elements without having two distinct equidistant boundary points if the elements share a point. In this case, the two footprints coincide, and so the largest interior sphere centered at $x$ touches the boundary in only one point. Shared
points imply adjacent boundary elements—a face and its edge, for example. Two-dimensional analogues are shown in figure 1.4. The number of these extra surfaces in the Voronoi diagram is at most $O(n)$. Thus, the medial axis and Voronoi diagram can be computed from each other in linear time. It is preferable to compute the Voronoi diagram, since the extra structure is in fact integral to a representation of the medial axis by algebraic surfaces. In figure 1.4, the Voronoi junction marks the transition from a line segment to a parabolic arc. Analogously, in 3-D, the extra Voronoi surfaces can mark the transition from one quadric surface patch to the next. Accordingly, throughout most of this dissertation I describe the computation of the internal generalized Voronoi diagram. The medial axis proper is obtained by a simple, linear-time post-processing step.

The medial axis admits an alternate definition in terms of offset surfaces. The offset at distance $r > 0$ of a set $S$ is the set of points at Euclidean distance $r$ from $S$. When $S$ is a closed surface in $\mathbb{R}^3$, the offset consists of two offset surfaces, interior and exterior. Informally, the interior offset surfaces of the boundary of a polyhedron define the medial axis by their singular points (tangent discontinuities). Formally, a point $p$ on an interior offset surface is on the medial axis if and only if there is no well-defined tangent plane to the offset surface at $p$. Conversely, the set of such singular points on all interior offset surfaces for $r > 0$ constitute the medial axis. This medial axis definition is a formalization of the grassfire analogy, which describes the medial axis as the set of self-quenching points of a fire lit simultaneously everywhere on the boundary.

1.2 Combinatorial complexity

The complexity of the 3-D polyhedral medial axis is defined as the total number of sheets, seams, and junctions. Tight bounds on the worst-case complexity of the medial axis are not known. It is currently known to be $\Omega(n^2)$ and $O(n^{3.5+\epsilon})$, where $n$ is the total number of vertices, edges, and faces in the polyhedron.

The $\Omega(n^2)$ bound is realized by the polyhedron in figure 1.5. Each of the top faces governs a sheet with each of the bottom faces. Another interesting example is the "iron maiden pizza box" shown in figure 1.6. Each of the $\Theta(n^2)$ horizontal sheets is governed by the same two faces. This example also serves as a reminder that the sheets, seams, and junctions are not determined by their governors.

The upper bound follows from Sharir and Agarwal [85], theorem 7.17. Specialized
Figure 1.4: The medial axis (left) and Voronoi diagram (right) of a polygon. The only difference is in the Voronoi boundaries touching the reflex vertices. Circles such as the one shown are not maximal. The center is equidistant from two boundary elements, but not from two boundary points.

Figure 1.5: A polyhedron whose medial axis has $\Theta(n^2)$ combinatorial complexity. (Image courtesy of Jeff Erickson.)
Figure 1.6: Another polyhedron whose medial axis has $\Theta(n^2)$ complexity. More surprisingly, it has $\Theta(n^2)$ sheets governed by the same two faces. The top and bottom faces have been removed to show the spikes poking in from two sides. This particular instance has ten spikes, breaking the top-bottom bisector into sixteen pieces.

to the present problem, the theorem states that given any $\epsilon > 0$, the complexity of the 3-D Voronoi diagram of a collection of points, line segments, and polygons in general position is $O(n^{3+\epsilon})$. A polyhedron's vertices, edges, and faces are not a collection of such elements in general position. However, the authors argue that the general position assumption is not essential for the bound. The bound clearly applies to the medial axis as well as the Voronoi diagram.

In the same work, Sharir and Agarwal present an algorithm for computing the Voronoi diagram in $O(n^{3+\epsilon})$ randomized expected time, matching their complexity bound. (See [85], theorem 7.22.) The analysis assumes that certain operations, such as computing the location of a junction point, take constant time. It is not clear whether their algorithm is suitable for a practical implementation.

Thus, there is a gap between the upper and lower bounds on the complexity of the 3-D medial axis problem. This is to be contrasted with the polygonal medial axis in 2-D, which can be computed in $O(n \log n)$ time by several methods (summarized in chapter 2) or in linear time if the polygon is simple (Chin et al. [16]).

Agarwal and Sharir [1] conjecture that the worst-case complexity of the Voronoi diagram of points, line segments, and triangles in $\mathbb{R}^3$ is near-quadratic. In the cited article, the authors prove a near-quadratic bound on the complexity of a related structure: the boundary of the configuration space of a ball of radius $r$ among point, line, line-segment, and triangle obstacles. This boundary is the set of points at distance $r$ from the point, line, line-segment, and triangle sites. It may be viewed
as a cross-section of the Voronoi diagram of the sites. The features of the Voronoi diagram are traced out by the boundary as $r$ varies over the positive real numbers. The part of the Voronoi diagram corresponding to a fixed $r$ is the configuration space boundary.

1.3 Previous approaches

Researchers have devised many algorithms for computing the polygonal medial axis in the plane. Standard techniques in computational geometry, such as divide-and-conquer, sweepline, and incremental insertion of boundary elements have been applied successfully to the 2-D problem. Unfortunately, these techniques do not generalize readily to 3-D. Only a few conceptually distinct algorithms have been proposed for computing the 3-D polyhedral medial axis. Of those, most are discrete in nature. The discrete algorithms compute the medial axis only to a certain "resolution." The complete medial axis is obtained only as the resolution goes to infinity. (The meaning of the word "resolution" varies among the methods.) In practice, these algorithms stop at a finite resolution and return an approximate medial axis. The common deficiency of these methods is that the topology of the approximate medial axis is not guaranteed to be correct. That is, two boundary elements that govern a portion of the medial axis might not govern any part of the approximate medial axis. An incorrect topology makes the result unusable for some applications.

A medial axis algorithm is said to be continuous if it does not rely on the sufficiency of a resolution parameter. Very few continuous approaches have been proposed, and of those, only one appears to be practical. I refer to this approach as the tracing algorithm.

The tracing algorithm operates on the graph formed by the seam curves and junction points of the 3-D medial axis. The idea is to construct this graph by performing a breadth-first or depth-first search. The algorithm thus requires one seam-construction step for each seam in the medial axis. Constructing each seam curve requires, in the worst case, considering each element of the boundary. Several techniques exist for alleviating this combinatorial penalty, under the assumption that the input is not pathological.

At a lower level, the tracing algorithm requires ordering points along an algebraic curve. This operation inherently requires high precision when the points are close together. Any numerical implementation of the tracing algorithm is therefore sensitive
to roundoff error. As a result, continuous methods based on floating-point arithmetic have had limited application.

1.4 Robustness

The success of the medial axis tracing method depends on the reliability of the point-ordering subroutine. This subroutine may not be reliable if it has only approximate data about the junction locations. This reliability problem is not unique to the tracing method, or even to medial axis computation. Reliability is an issue in most continuous algorithms for geometric problems.

Two chronic reliability problems tend to recur in implementations of geometric algorithms: failure due to roundoff error, and the inability to handle degenerate configurations. Geometric algorithms and software systems often suffer from one or both of these problems. The problems are so common that the term robustness problems is widely used in computational geometry to refer specifically to roundoff error and degeneracies. This is the sense in which I will use the word.

1.4.1 Roundoff error

Some algorithms are not sensitive to roundoff error. For example, the Newton-Raphson algorithm for improving an estimate to a root of a smooth function has the property that in a neighborhood around a root, error decreases with every iteration. Even though roundoff error is propagated through the computation, this does not counteract the algorithm's tendency to converge to a root.

Many geometric algorithms, on the other hand, are quite sensitive to roundoff error. An algorithm for intersecting two polygons in the plane, for example, may require testing the position of the vertex of one polygon against the edge of the other polygon. The test is formulated as a predicate returning −1.0, or 1, according to whether the vertex is inside, on, or outside the edge. The algorithm is designed under the assumption that the test yields an error-free answer. However, if the test is implemented in floating-point arithmetic, roundoff error can cause an incorrect test result. One may hope that the result would be some sort of incorrect output, ideally only slightly different from the correct output. However, the result may be completely incorrect. Worse, the algorithm may fail completely or loop indefinitely. This could happen if, for instance, the inside-outside test incorrectly reported that a vertex of
one polygon is inside the other polygon, and the algorithm attempted to construct a non-existent intersection polygon (figure 1.7).

Two general approaches present themselves. One may design the algorithm to be stable in the presence of roundoff error, or one may use *exact computation* to assure that roundoff error does not affect the progress of the algorithm. The former approach has had limited success, and much of the work on robustness has focused on exact computation.

### 1.4.2 Degeneracies

Often a geometric algorithm assumes, for simplicity, that its input is in *general position*. This means that certain configurations are not allowed, and further, that the disallowed configurations are rare in the sense that almost any infinitesimal perturbation of the input will break the configuration. The configuration is called a *degeneracy*. For instance, a 2-D line segment intersection algorithm may assume that no two segment endpoints have the same $x$-coordinate. If two endpoints *are* vertically aligned, then any perturbation of those points which does not move them both to the same new $x$-coordinate—and most perturbations do not—will bring them under the assumption. Thus the algorithm is making a general position assumption. It may be possible in the implementation to perform a straightforward symbolic perturbation such as $(x, y) \rightarrow (x + \epsilon y, y)$, corresponding to lexicographic ordering.

For another example, consider an algorithm that intersects two spheres in 3-D. Two spheres in general position intersect in either a circle or the empty set. The algorithm may fail if it is given two spheres meeting at a point, or two coincident spheres. In fact, computing the intersection for these special cases may require special-purpose algorithms. In this case, the general position assumption can actually restrict

> ![Figure 1.7: Roundoff error causes a catastrophic failure of a polygon intersection algorithm. (a) The highlighted vertex is incorrectly classified as “inside” the highlighted edge. (b) In an attempt to compute the intersection polygon, the algorithm constructs a polygon which lies outside of both input polygons.](image)

10
the inputs that the implementation can handle. On the other hand, expanding the algorithm to handle all of the cases can lead to combinatorial explosion. A system component which takes a sphere-sphere-intersection as an input must now consider all four geometric types. So a general position assumption may serve to simplify an algorithm or an implementation (or both).

The problem is exacerbated by finite-precision arithmetic. In a floating-point implementation, non-degenerate data may become degenerate due to roundoff error. Also, degeneracies may be hard to detect, since the data are approximate. Finally, the domain of a limited-precision number system is fundamentally discrete, so it may be impractical to make a small enough perturbation to remove a degeneracy without introducing new degeneracies.

Moreover, real data are usually not in general position. Models of synthetic objects are often in special positions. A block-like object is likely to be modeled with faces parallel to the coordinate planes. Two physical balls in contact will be modeled by two spheres meeting at a point. Also, synthetic objects are often symmetric, so that their geometric elements are in special positions with respect to each other.

For some problems, it is possible to design an algorithm that is insensitive to the presence of degeneracies. Many others, especially three-dimensional problems, seem to require either systematic detection of degenerate cases or a systematic way of applying a small perturbation. Either solution—detection or perturbation—seems to require some form of exact computation.

### 1.4.3 Exact computation and exact arithmetic

Floating-point algorithms can be made robust against roundoff error and degeneracies. A number of examples are given in chapter 2. However, most of this work has been applied only to problems in the plane, such as the Voronoi diagram of line segments, or 3-D problems that are linear in nature, such as polyhedral solid modeling. Further, the techniques used in these approaches do not generalize easily to curved objects in 3-D. In particular, it is not clear that a purely floating-point implementation of a continuous medial axis algorithm can be made robust.

*Exact computation* is an alternative to finite-precision arithmetic. Exact computation means that discrete values are always evaluated correctly, even when they depend on continuous values. This allows the high-level algorithm to ignore the effects of roundoff error, and offers the opportunity to detect degenerate configurations correctly.
I distinguish *exact arithmetic*, meaning full-precision numerical computation, and *exact computation*, meaning any form of computation which does not return an approximate answer. The distinction is important in view of the wide variety of techniques for achieving exact computation without relying exclusively on exact arithmetic.

The typical means for achieving *exact arithmetic* is to represent integers by arbitrary-length arrays of machine integers. Rational numbers, if needed, are represented by pairs of these "big" integers. Alternately, a large integer may be represented by its images modulo a collection of relatively-prime numbers. Algorithms for integer, rational, and modular exact arithmetic are well-established, and can be found in Knuth [62]. For algebraic numbers, exact arithmetic is more difficult. General-purpose methods supporting addition, multiplication, and division on algebraic numbers are too slow to be practical.

The notion of exact computation includes exact arithmetic, along with a host of other techniques for ensuring a correct answer without necessarily using full-precision arithmetic. These other techniques apply to problems where the input is numerical while the output is discrete. Computing the sign of an algebraic expression is one example. Another is the problem of sorting points according to some criterion. Exact computation always implies some attention to the arithmetic performed by an algorithm.

One paradigm for exact computation is approximate arithmetic with error management. An example is interval arithmetic. For example, an algebraic expression is evaluated in interval arithmetic, and if the resulting interval does not contain zero, then the sign is known reliably. If the interval does contain zero, the computation is repeated in higher-precision interval arithmetic or in exact arithmetic. This basic idea has been manifested in many different ways. One may bound the width of the interval at compile time, resulting in less computation at run time. Or one may represent relative error instead of absolute error, or the log of the relative error (which is equivalent to the number of significant digits in the result). In all of these forms, this idea amounts to *forward error analysis*, in which a bound on the error of a result is obtained by bounding the cumulative error at each step. Another option is *backward error analysis*, in which the net error of an entire computation is analyzed. The net error can sometimes be shown to be less than would be predicted by forward error analysis.
1.5 Thesis

My thesis is

The medial axis of a polyhedron can be computed accurately and efficiently using exact computation.

I demonstrate my thesis by designing an accurate algorithm, implementing it, and showing its practical efficiency on a variety of polyhedra.

Accuracy. By accuracy I mean that the medial axis is computed correctly and to any desired precision. The topology of the medial axis is correct and is independent of the output precision. Accuracy implies that roundoff error has neither cumulative effect on the result nor catastrophic effect on the algorithm’s progress. For the algorithm to exhibit robustness, it would need first to be accurate, and then to handle all forms of degeneracy. It is not my intention to settle the issue of degeneracy completely in the polyhedral medial axis problem. However, I identify the most common forms of degeneracy and address them in the algorithm.

Exact computation. Exact computation is the means by which I achieve accuracy. With approximate arithmetic, accuracy can only be verified empirically, by applying an implementation to a large class of inputs. With exact computation, one can design an algorithm which is inherently accurate.

Practical efficiency. I strive for efficiency in a practical sense rather than the standard sense of asymptotic dependence on input size. By practical efficiency I mean that the algorithm can be implemented so as to compute the continuous medial axis of a reasonably complex polyhedron in a reasonable amount of time.

My goal is to obtain running times within one to two orders of magnitude of similar floating-point based algorithms, while providing a much more reliable solution. Evaluating my work relative to this goal is difficult, because there are no publicly available continuous medial axis implementations, to my knowledge. I demonstrate that one can obtain an accurate medial axis in a reasonable amount of time, by applying my implementation to a number of complex examples.

My approach is to design an algorithm that is inherently accurate and then improve the efficiency until the algorithm becomes practical. This is in contrast to the common
strategy, which is to implement an algorithm in floating-point arithmetic for efficiency and then improve the behavior of the algorithm in an attempt to attain robustness. The main drawback to the common strategy is that robustness problems are often not revealed until late in development, when more complicated inputs are attempted. With my approach, accuracy is ensured at all times.

1.6 New results

The main result of this work is an accurate algorithm for computing the continuous medial axis of a polyhedron. To my knowledge it is the first such algorithm to be designed or implemented. In the course of this work, I have examined many subproblems. I describe some of the results below.

- **Three exact seam searching algorithms.** The fundamental step in my medial axis algorithm is based on ordering points along an algebraic seam curve. I present three different algorithms for this operation, corresponding to three representations for the seam curve. The choice of the most efficient algorithm depends on the simplest representation available for each seam.

- **Efficient intersection of three algebraic surfaces.** A medial axis junction point lies at the intersection of three quadric surfaces. I describe an efficient algorithm for isolating and refining solutions to a system of three low-degree polynomial equations in three variables, based on multivariate Sturm sequences and the Macaulay resultant.

- **Parametric forms for bisector surfaces of points, lines, planes.** The medial axis consists of quadric surfaces, which necessarily admit rational parametrizations. I present practical methods for computing parametrizations of all of the surface types that occur in the medial axis.

- **A complete differential-geometric characterization of the incidence structure to a generic or degenerate junction point.** Degenerate junction points are common in symmetric objects. I give a complete characterization of the structure of the incident curves and surfaces at a degenerate junction point. I also present algorithms to identify and distinguish these curves and surfaces.

- **Improvements to the overall running time.** No asymptotically efficient algorithm for this problem is known. I present two high-level strategies for improving the
running time on a class of inputs that is common in practice. One strategy is based on polytope intersection, and its effect on the overall running time is easily understood both in theory and in practice. The second strategy is based on recursive subdivision of space. It is an improvement upon the subdivision strategies suggested by other authors.

- A hybrid method for evaluating determinant signs for moderate-sized matrices. Algebraic computation for geometric problems often relies on Sturm sequences. An efficient Sturm sequence implementation based on the subresultant algorithm requires an efficient means to find the sign of a matrix determinant. This problem has received significant attention recently, but research has focused on small matrices (typically of order six or less) with machine-precision entries. My algorithm for intersecting three quadric surfaces requires determinant signs of order 15 with entries many times larger than machine precision. I present a hybrid algorithm for evaluating such determinants featuring a floating-point filter based on the singular value decomposition (SVD) of a matrix.

Some of these results will be useful to those computing or using polyhedral medial axes, in an exact setting or otherwise. Some are of general interest in geometric computing and solving polynomial systems.

Figures 1.8 through 1.12 show example polyhedra and their medial axes as computed by my algorithm. The running times are given in table 1.1. The box with grooves in figure 1.9 is particularly challenging for a non-robust algorithm due to the small, sharp features in the center. Other examples are challenging due simply to their complexity, which exceeds that of many of the polyhedra that have been used to demonstrate other continuous polyhedral medial axis algorithms.

<table>
<thead>
<tr>
<th>Polyhedron</th>
<th>Input complexity</th>
<th>Output complexity</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>faces</td>
<td>edges</td>
<td>vert.</td>
</tr>
<tr>
<td>L (1.8)</td>
<td>10</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>Groove (1.9)</td>
<td>16</td>
<td>38</td>
<td>24</td>
</tr>
<tr>
<td>Pizza (1.10)</td>
<td>56</td>
<td>124</td>
<td>70</td>
</tr>
<tr>
<td>Bracket (1.11)</td>
<td>252</td>
<td>378</td>
<td>124</td>
</tr>
<tr>
<td>Venus (1.12)</td>
<td>250</td>
<td>375</td>
<td>127</td>
</tr>
</tbody>
</table>

Table 1.1: Performance on some example polyhedra. Running times are on an SGI computer with an R12000 processor running at 300 MHz.
Figure 1.8: A simple non-convex polyhedron: an L-shaped box. (a) The polyhedron. (b) Its seam curves. (c) The central sheets.

Figure 1.9: A polyhedron with two grooves and its medial axis. (a) The polyhedron. (b) Its seam curves. (c) Its medial axis. (d) The part of the medial axis which does not meet the boundary. magnified 2×. (e) The central sheet is a hyperbolic paraboloid. the bisector of the two grooves. Magnified 200×.
Figure 1.10: The “iron maiden pizza box” and a schematic of its medial axis.

Figure 1.11: A CAD model of a bracket and a schematic of its medial axis. The medial axis has a few degenerate junctions, indicated by the isolated points. The implementation currently supports tracing only seams having at least one generic junction.
Figure 1.12: A model of the Venus de Milo sculpture. The medial axis of this asymmetrical polyhedron has no degenerate seams or junctions. It does have many small seams and sheets. (a) Venus. (b) Its seam curves as line segments. (c) The "central" seams, those that do not have an endpoint on the boundary.
1.7 Overview of chapters

Chapter 2 reviews work related to medial axis computation and robustness in geometric computing. Chapter 3 describes my medial axis algorithm at a high level. The following two chapters give the details of the algorithm. Chapter 4 describes distance functions, domain polytopes, and the computation of bisector surfaces. Chapter 5 covers the medial elements which require trimming and assembly: junctions, sheets, and seams.

Chapter 6 describes the techniques used at all levels to improve the efficiency of the exact algorithm without sacrificing accuracy. Chapter 7 discusses issues of degeneracies in the medial axis and in the algorithm. Chapter 8 addresses the important subproblem of determinant sign evaluation. For this problem I describe a new hybrid method based on fixed-precision, mixed-precision, and modular arithmetic.

Chapter 9 describes my implementation of the algorithm and highlights its performance on a few nontrivial polyhedra. I conclude in chapter 10 and discuss directions for future work. Appendix A summarizes a few standard results from algorithmic algebra.
Chapter 2

Related Work

In this chapter I review work related to the polyhedral medial axis problem and robust geometric computation. Section 2.1 mentions some of the applications of the medial axis, focusing on applications in solid modeling. Section 2.2 describes work on a few closely-related problems. Sections 2.3 and 2.4 give an overview of the various methods that have been used to compute the medial axis of a polyhedron. I focus on approaches that compute the continuous medial axis, and I evaluate the practical efficiency of these approaches. Lastly, section 2.5 summarizes some of the previous work on exact geometric computation.

2.1 Applications of the medial axis transform

The medial axis was initially conceived by Blum [7] for the measurement and analysis of biological shapes. The medial axis and other medial surface techniques have proven useful in computer vision, shape analysis, and image segmentation. For recent examples of these applications, see Ogniewicz and Kübler [74], Székely [93], and Fritsch et al. [41].

In solid modeling, the medial axis transform has proven a useful tool for many different problems. For finite-element meshing, the medial axis transform—the medial surface together with the radius function—provides local scale information. The mesh should adapt to the “local size” of the domain, becoming coarser or finer according to whether the domain is wide or narrow. Local size is not directly available from a boundary representation. The MAT, on the other hand, reveals object width explicitly in the radius function. Srinivasan et al. [90] present a MAT-based meshing algorithm and place it in the context of other meshing methods. Other MAT-
based meshing methods are presented by Gursoy and Patrikalakis [47, 48] and Storti et al. [91].

The MAT can also be used for model simplification for engineering analysis. Two fundamental simplification techniques are dimensional reduction and detail suppression. By providing a reduced-dimensional representation of an object, the MAT can assist in abstracting a complicated shape to a simpler one. For instance, a complicated 3-D shape might be more simply represented as a swept polygon or thickened sheet. Geometric detail may be suppressed with the assistance of the medial axis, thus improving the performance of the analysis without significantly affecting the results. See Armstrong [4], Armstrong et al. [3], and Donaghy et al. [28]. The medial axis has been used to construct offset curves of polygons in $\mathbb{R}^2$ (Held [53]) and may prove useful for computing offset surfaces in $\mathbb{R}^3$.

Further applications in solid modeling include shape morphing and shape synthesis (Storti et al. [91]) and feature recognition (Price et al. [79]). Several authors, noting that the medial axis transform is a complete shape description, have suggested the use of the MAT as an alternate representation in a geometric modeling system, alongside the traditional boundary and constructive solid geometry (CSG) representations. Thall et al. [94] propose sampled medial surfaces for the fundamental representation in a modeling system.

### 2.2 Problems related to polyhedral medial axis computation

This section mentions work on problems that are closely related to the polyhedral medial axis problem: computing the medial axis of other objects, computing Voronoi diagrams, generalizations of Voronoi diagrams, and computations with quadric surfaces.

**Medial axis of CSG objects.** Dutta and Hoffmann [29] and Hoffmann [55] have presented algorithms for computing the medial axis of constructive solid geometry (CSG) objects. Their algorithms compute the points of closest approach between pairs of boundary elements. The cited work also addresses the exact representation of bisectors arising in medial axis computation of CSG objects delineated by planes, quadrics, and tori. Elber and Kim [31] construct parametric bisector surfaces for
various CSG primitives, a fundamental operation for computing the medial axis of a CSG solid.

Voronoi diagrams. For polyhedra, the medial axis is closely related to the generalized Voronoi diagram. The simpler problem of computing the Voronoi diagram of a point set (the non-generalized Voronoi diagram) has received much attention in computational geometry. Point-bisectors are hyperplanes in \( \mathbb{R}^d \), so the point Voronoi diagram requires only linear computation. Also, a tight bound is known on the combinatorial complexity of the Voronoi diagram in \( \mathbb{R}^3 \) (\( \Theta(n^2) \) in the worst case). The Voronoi diagram of points is thus easier in terms of both combinatorial complexity and geometric complexity. For these reasons, several authors interested in the medial axis either settle for the Voronoi diagram as an approximation to the medial axis, or use the Voronoi diagram as the basis for a medial axis algorithm (as Sheehy et al. [86] do). The most common algorithms for the Voronoi diagram of points in 3-D are incremental.

A recent survey by Aurenhammer and Klein [5] summarizes the state of the art of Voronoi diagram theory and computation. Also see Okabe et al. [75].

While no bound tighter than \( O(n^{3+c}) \) is known for the generalized Voronoi diagram of lines or planar polygons in \( \mathbb{R}^3 \) (see section 1.2), other Voronoi diagrams do have quadratic complexity, encouraging speculation that the medial axis may actually be quadratic in the worst case. One such generalization is to replace the Euclidean metric with another \( L_p \) metric. The unit sphere of any metric on \( \mathbb{R}^d \) is the set of points at distance \( \leq 1 \) from the origin. In \( \mathbb{R}^3 \), the unit sphere of \( L_2 \) is a sphere: the unit sphere of \( L_1 \) is a cube: the unit sphere of \( L_\infty \) is an octahedron. In fact, for any compact, convex set \( C \) containing the origin, one may construct a distance function \( d_C(-,-) \) whose unit sphere is \( C \). The distance from \( p \) to \( q \) is defined by translating \( C \)'s origin to \( p \) and magnifying \( C \) to the point where its boundary touches \( q \). The magnification factor gives the distance. Such a distance function \( d_C \) is a metric only if \( C \) is symmetric about the origin. A distance function defined in this way is called a \textit{convex distance function}. A notion that clearly generalizes the \( L_p \) metric. Generalized Voronoi diagrams are well-defined with respect to a convex distance function in \( \mathbb{R}^d \), though some allowance must be made for bisectors which include \( d \)-dimensional regions of space (as in the \( L_1 \) or \( L_\infty \) metric on the plane). Considering convex distance functions on \( \mathbb{R}^3 \) where \( C \) is a polytope of constant complexity, Chew et al. [15] show that the Voronoi diagram of \( n \) lines based on \( d_C \) is near-quadratic—\( O(n^2\alpha(n)\log(n)) \). Unfortunately, complexity
results on Voronoi diagrams in non-Euclidean metrics do not apply to the Euclidean problem. Corbalán et al. [21] show that the Voronoi diagram with respect to $d_C$ may quite easily have a topology which is not realizable in $L_2$.

**Curved boundaries and symmetry sets.** For a shape with higher-degree boundary elements, the medial axis is no longer a subset of the generalized Voronoi diagram. For instance, consider a "polygon" in the plane with linear and parabolic edges. An inscribed circle can now be tangent to two points on the same edge; the center is on the medial axis, but not on the Voronoi diagram. In this context, and in the arbitrary Euclidean space $\mathbb{R}^d$, the medial axis and Voronoi diagram are both subsets of the symmetry set of the boundary—the set of centers of twice-tangent spheres, where the spheres are not restricted to lie within the shape.

The medial axis was conceived in this more general setting. Blum [7] defined the medial axis not just for polygons but for planar regions a with piecewise-smooth boundary. Some of his original results characterize local geometry of the medial axis. Nackman [73] generalizes Blum's results to three dimensions.

In recent work, Rieger [81] generalizes the symmetry set of a collection of hypersurfaces in $\mathbb{R}^d$ to the bifurcation sets of collections of semi-algebraic sets of arbitrary codimension. The bifurcation sets can be used to partition space for proximity queries.

**Quadric surfaces.** Effective computation of the polyhedral medial axis requires careful manipulation of the intersections of quadric surfaces. Quadrics are fundamental to computer-aided geometric design, and the literature in that field offers many methods for analyzing and computing their intersections. Especially notable is the work of Farouki et al. [36], who give a general method for finding the morphology of a quadric-quadric intersection curve. Many other researchers have limited their attention to a selection of so-called "natural" quadrics. Bisector surfaces are almost never natural quadrics.

The problem of finding the intersection points of three quadric surfaces is examined by Chionh et al. [17], who compute the Macaulay resultant of the three equations and locate the roots by examining their projections on the coordinate axes.

Having described some work on closely-related problems, I will now focus on the polyhedral medial axis problem itself. I divide the previous work into two categories: algorithms based on discretization and continuous algorithms. The major previous approaches are covered in the following two sections.
2.3 Medial axis algorithms based on discretization

In view of the apparent complexity of the polyhedral medial axis problem, many algorithms are based on a discretization, either of the solid's boundary or of the Euclidean space in which the solid is embedded. Most methods which have succeeded on large inputs (hundreds or thousands of boundary elements) are based on discretization.

**Boundary discretization.** Many authors have computed the medial axis using the Voronoi diagram of a set of regularly-spaced points on the polyhedron's surface. The Voronoi diagram of the points is piecewise linear and can be computed reliably by any of several techniques (see Aurenhammer and Klein [5] for a survey). A subset of the Voronoi cell boundaries is identified as an approximation to the medial axis. Such an approximation suffices for some applications.

Among the algorithms based on this idea, the method of Sheehy et al. [86] is notable in that it attempts to construct the complete medial surface topology, rather than the topology of an approximate medial surface. The algorithm samples the polyhedron's boundary by points and computes the Voronoi diagram of those points. The vertices of the Voronoi diagram are centers of maximal spheres that fit among the points. Many of these maximal spheres are close to maximal spheres inscribed in the original polyhedron. The centers of these spheres lie near the medial axis. Unfortunately, many of the spheres do not correspond to medial axis points. For instance, some lie outside the polyhedron, while others are “rogues” that do not reflect the medial surface topology. The primary challenge is to classify the Voronoi spheres relative to the medial axis. Sheehy et al.'s method is also capable of adaptively resampling the polyhedron’s boundary until the correct medial surface topology is revealed, although the required sampling may be so fine as to be impractical. Interestingly, their method for dealing with near-degenerate medial junctions is similar to the tracing methods described in section 2.4.

**Spatial subdivision.** Another popular approach is to subdivide space into rectilinear cells and compute an approximation to the medial axis at the resolution of the cells. The approximation can be considered an image of the medial axis. It does not contain a complete description of the adjacency relationships unless the resolution is high enough.

Vleugels and Overmars [95] describe a spatial subdivision algorithm for computing the generalized Voronoi diagram of arbitrary convex sites. Associated to each recti-
linear cell are eight sites, the closest site to each of the eight vertices. The algorithm relies on a black-box subroutine for computing the distance from a point to a site. A cell with more than one closest site contains part of the Voronoi boundary. A cell with only one closest site is assumed to lie completely inside that site's Voronoi region, though this may not always be the case. Except for this problem, the algorithm identifies a collection of cells whose union contains the Voronoi diagram.

Vleugels and Overmars make two important observations that improve the efficiency of the technique. The first is that by connectivity of the Voronoi diagram, the Voronoi boundaries may be found by moving along bisector surfaces. This avoids examining most cells lying far from the Voronoi diagram. In essence, the algorithm behaves like an isosurface extraction algorithm. The other observation is that a cell which touches a set S of Voronoi regions may be divided into sub-cells, each of which can only touch Voronoi regions included in S. This eliminates many Voronoi sites from consideration when evaluating a cell during recursive subdivision.

The principal drawback of the technique is that full topological knowledge of the Voronoi diagram is not revealed. The authors prove that the correct topology is available in the limit, as the resolution approaches infinity. At finite resolution, a particular problem is that a cell may appear to lie within a single Voronoi region, though it actually touches another. This occurs when a curved bisector intersects the box without separating two vertices.

The recent work of Etzion and Rappaport [34] is also based on spatial subdivision. An important difference is that Etzion and Rappaport restrict their view to the three-dimensional medial axis of a polyhedron. By specializing to the Voronoi diagram of points, line segments, and polygons, the authors can take advantage of the analytical representation of the bisectors. The bisectors, as quadric surfaces, are intersected with the cell's faces and edges. In this way, the marked cells contain all of the Voronoi diagram. This alleviates the inexactness in the algorithm of Vleugels and Overmars. However, the method retains the disadvantage that a near-degenerate or degenerate Voronoi diagram cannot be completely resolved without subdividing to extremely high resolution. Finally, the authors provide a method for extracting the actual Voronoi diagram from the cell structure. The algorithm gives a complete, continuous Voronoi diagram if the resolution is high enough and there are no degeneracies. Otherwise, an approximate Voronoi diagram is produced. In the approximate diagram, near-degeneracies are represented as degeneracies. The local cell size gives an estimate of the spatial error.
Etzion and Rappaport have implemented their algorithm. The implementation has been used to compute the medial axis of polyhedra up to about 350 faces. On such a polyhedron the implementation requires on the order of eight minutes. Thus the method is fairly efficient from a practical point of view.

Hoff et al. [56] compute discrete generalized Voronoi diagrams by spatial sampling of distance functions. Taking advantage of hardware designed for raster graphics, the algorithm produces high-resolution discrete Voronoi diagrams very efficiently. Since the algorithm relies entirely on sampling and does not analyze the distance function over a cell, it is more closely related to the technique of Vleugels and Overmars than that of Etzion and Rappaport. The algorithm has been applied to tessellated curves and surfaces in 2-D and 3-D. Its practical efficiency may prove good enough for on-line applications in 3-D, such as motion planning in a dynamic environment [57].

2.4 Continuous medial axis algorithms

While discrete methods can compute a complete and correct medial axis, in practice this is quite often prohibitively expensive. The required sampling resolution is arbitrarily high for degenerate and near-degenerate problems. Unfortunately, many practical problems are degenerate or near-degenerate. Though a discrete method may be capable of computing a topologically correct medial surface at a high but finite resolution, in practice it terminates at a somewhat lower resolution, returning an approximate medial axis.

A continuous algorithm is one whose correctness does not depend on the sufficiency of some sampling parameter. The field of computational geometry is mostly concerned with continuous algorithms for geometric problems. The literature of computational geometry offers diverse solutions for closely-related problems, but few continuous methods have been devised for the three-dimensional medial axis problem.

2-D continuous algorithms. The medial axis of a polygon in the plane is a well-studied problem. Several fundamentally different algorithms, all with favorable worst-case or average-case behavior, are known. Some of the works here address the 2-D medial axis problem, while others address the generalized Voronoi diagram of line segments. A solution to the Voronoi problem can be directly applied to the medial axis problem without affecting the combinatorial complexity.

The classic divide-and-conquer algorithm for the two-dimensional medial axis
problem was proposed by Lee [64]. Lee’s algorithm splits the polygon into two roughly equal-sized polygonal chains. A partial medial axis is computed recursively for the two chains. The partial results are then combined in a linear-time merge step. The overall running time is $O(n \log n)$ for a polygon with $n$ edges.

Fortune has presented a sweepline algorithm for the generalized Voronoi diagram of points and line segments [40]. The algorithm computes the elements of the diagram in order by their $x$-coordinates. The running time is $O(n \log n)$.

Imai has presented an incremental algorithm for the Voronoi diagram of line segments in 2-D [58]. The algorithm maintains topological consistency even in the presence of numerical error. The algorithm runs in $O(n^3)$ time in the worst case, but experiments on random inputs of thousands of line segments suggest that the running time is typically $O(n)$ in that range. Recently, Held has extended Imai’s technique [52].

Held describes wavefront-propagation algorithms for the medial axis of arbitrary 2-D polygons [49] and for convex 3-D polyhedra [50]. The wavefront-propagation algorithms compute the medial axis in a greedy way, constructing the smallest feature (in terms of the radius function) at each step when possible. For a convex polygon or polyhedron, this idea leads to an algorithm that correctly computes the medial axis in 2-D or 3-D. For a non-convex polygon or polyhedron, Held’s purely greedy algorithm does not compute the correct medial axis without backtracking. Held’s 2-D non-convex algorithm remedies this problem by constructing the medial axis greedily in the early stages and globally in the later stages. The global part of the algorithm uses the merge step from Lee’s algorithm [64] as a subroutine. The resulting hybrid algorithm has $O(n^2)$ complexity in the worst case. Empirically, however, its performance is nearly linear in the input size. Practically, Held’s algorithm compares favorably to Lee’s.

Chin et al. [16] give an algorithm for computing the medial axis of a simple polygon in linear time. Though theoretically optimal, it is not clear that the algorithm is suitable for a practical implementation. Finally, the 3-D tracing algorithm described below has been applied to the 2-D problem by Evans et al. [35]. Their algorithm is simple to implement. It takes $O(n^2)$ time in the worst case but less on average.

3-D continuous algorithms. Interestingly, most of the foregoing algorithms have not been extended to non-convex 3-D polyhedra. For instance, the 2-D divide-and-conquer algorithm has not been generalized to 3-D. The merging step in the 2-D al-
gorithm constructs a piecewise-parabolic polygonal chain. The segments of the chain are computed in order from one end to the other. In 3-D, a divide-and-conquer algorithm would presumably involve constructing a large piecewise-quadric surface. The computation of such a surface appears to be a very difficult problem, one that has not been attacked in the literature. Similarly, the sweepline algorithm and the wavefront propagation algorithm do not appear to generalize easily to three dimensions.

The 3-D problem may be amenable to an incremental algorithm. Milenkovic has proposed such an approach [69], but to my knowledge, the algorithm has not been fully described nor fully analyzed. The Voronoi cell of a single boundary element in 3-D has a complex structure that appears to be difficult to characterize.

3-D tracing algorithms. All practical, continuous algorithms for computing the 3-D medial axis are based on the tracing approach. Starting from a junction point, a seam emanating from the junction is followed. The seam terminates at another junction. Once this point is found, the algorithm recursively forks and follows all seams emanating from that junction.

The key step is the search for the terminator of a seam. In the two-dimensional medial axis problem, the search is limited to smaller and smaller sub-chains of the polygon. The search cannot be limited in the three-dimensional problem: each terminating junction is found at the expense of a search of (in the worst case) the entire polyhedron. Thus the complexity is proportional to both the input size \( n \) of the polyhedron and the output size \( m \) of the medial axis. The algorithm's worst-case behavior is necessarily at least a factor of \( n \) larger than optimal \( \Theta(m) \) time. (A tight upper bound on \( m \), the size of the medial axis, is currently not known: see section 1.2.)

Milenkovic [69] proposed the first 3-D tracing algorithm. His algorithm finds a sequence of points along each seam curve. At a point on the curve, a small tetrahedron is formed with its apex at the point and its base pointing in the direction of the seam tangent. The algorithm tests for the presence of the terminating junction inside the tetrahedron. This involves a pass through the entire input. If the tetrahedron does not contain the terminator, a new point on the seam is found by intersecting the seam with the walls of the tetrahedron. Milenkovic discusses the suboptimal complexity of this algorithm and proposes a heuristic for reducing the number of terminator candidates. Space is subdivided recursively into boxes. Associated with each box is a superset of the nearest boundary elements to the box. When examining a tetrahedron, only these sites are considered as possible terminators. The preprocess is similar to
the method of Etzion and Rappaport [34] described in section 2.3.

Sherbrooke et al. [87] have presented a variation on the same algorithm, together with further details and an implementation. Instead of using a tetrahedron, their algorithm takes its small steps along the seam curve by numerically integrating a system of differential equations that describe the seam.

The authors also describe a technique for identifying the valid seams incident to a degenerate junction. A generic junction has four governing boundary elements. From these four governors, each combination of three governs an incident seam. Thus at a generic junction, the incident seams are easily identified. At a junction with \( k > 4 \) governors, not all of the \( \binom{k}{3} \) combinations govern an incident seam. If a combination of three governors does govern a seam, the combination is called valid. The authors give a numerical method for identifying the valid combinations. The key observation is that a small enough step away from the degenerate junction does not move past the other end of the seam. The algorithm takes a single small step along the proposed seam and checks whether this new point is closer to the seam governors than to any other element. To define “small enough” at a degenerate junction, let \( \Delta r \) be the difference between the distance to the governors and the distance to the next closest element. Then any step along the proposed curve which is less than \( \Delta r/2 \) in arc length does not move past the end of the curve. This “safe distance” is also used during seam traversal.

Both of these algorithms have essentially the same asymptotic complexity. Let \( n \) be the number of boundary elements in the input polyhedron, and let \( m \) be the number of seams in the medial axis. For each of the \( m \) seams, the algorithms consider all \( n \) boundary elements as potential terminators. Thus a simple estimate of the running time is \( O(mn) \). An optimal algorithm would run in \( O(m) \) time, so it is natural to try to reduce the effect of the factor of \( n \). Milenkovic [69] claims that his spatial subdivision preprocess can reduce this factor essentially to a constant for practical inputs. The preprocess consumes some time, but Milenkovic recommends simply stopping the recursive subdivision when the time taken starts to become significant compared to the tracing algorithm.

Another factor in the running time is the number of steps taken along the seam. This number is difficult to estimate a priori, since it depends on the geometric structure of the medial axis. In particular, a near-degenerate or highly curved medial axis may require many small steps. Finally, there is a small amount of overhead in data structures. Both methods accumulate the junctions in a set, so searching may stop
at a junction that has already been discovered. This introduces another small factor into the asymptotic running time.

Sherbrooke et al. have implemented their algorithm and applied it to moderate-sized polyhedra, up to about 60 faces. For polyhedra of this size, the running time is on the order of ten minutes. The algorithm of Milenkovic has not been implemented, to my knowledge.

Reddy and Turkiyyah [80] have presented a variation of the tracing algorithm. Rather than tracing the curve in small steps, their algorithm constructs all candidate terminating junctions and their maximal spheres. Each of these spheres is then tested for containment in the polyhedron. The authors demonstrate that only one sphere satisfies this criterion, if it is assumed that each bisector is simply connected. This assumption rules out some polyhedra (such as the pizza box, figure 1.6), but the authors claim that the assumption holds on practical models. The asymptotic running time is $O(mn^2)$ since tracing a seam requires finding $O(n)$ candidate junctions and testing each candidate against $O(n)$ elements. Asymptotically, the algorithm is therefore worse than the tracing algorithms. As a heuristic, the authors suggest presorting the candidate junctions by Euclidean distance from the starting junction. Assuming this heuristic always works, the running time is reduced to $O(mn \log n)$.

Reddy and Turkiyyah also suggest an alternate method for finding a seam. The alternate method simply picks the candidate junction that is closest to the starting junction. This heuristic is effective when the seam is well-approximated by a straight line. Finally, the authors describe a boundary-sampling algorithm and contrast the two approaches.

Reddy and Turkiyyah have implemented their tracing algorithm and demonstrated it on polyhedra of about 50 faces. Their implementation computes the medial axis of such a polyhedron in about three minutes.

**Reliability.** A difficulty common to all of the cited tracing algorithms is reliability. At each stage, the algorithm assumes that all previous seam terminators have been found correctly. If numerical error causes an incorrect seam terminator to be chosen, the algorithm can fail. Based on reported implementation results, the practical efficiency of these continuous algorithms is acceptable. Yet the algorithms have not, to my knowledge, been applied to inputs of a significant size. This may be due to reliability problems.

Reliability is a major issue not just for medial axis algorithms, but for geomet-
ric algorithms in general. Recently, much attention has been paid to reliability, or robustness, issues in geometric computing. Some authors have focused on particular geometric algorithms. Others have attempted to find more general solutions to these problems. The remainder of this chapter is dedicated to issues of reliability in geometric computing. Since the reliability of 3-D medial axis algorithms has not been examined thoroughly, I describe work on closely-related problems, such as 3-D polyhedral modeling and 2-D Voronoi diagram computation. I also describe some of the general techniques.

2.5 Robust geometric computation

Geometric algorithms tend to be unstable in the presence of roundoff error. Designing algorithms that are stable when implemented purely in floating-point arithmetic is possible, but difficult. Hoffmann, in chapter 4 of his book on geometric modeling [54], gives an overview of the problem and some possible approaches. Many diverse methods have been found to increase the robustness of geometric software. In this section, I give a high-level overview of the most important techniques. I split the topics into two categories, according to whether the technique is based on exact computation. Finally I summarize the relevant work on degeneracies.

2.5.1 Improving robustness without computing exactly

In modern computers, floating-point arithmetic is provided in hardware, and the operations are highly optimized. An implementation based solely on floating-point arithmetic has a significant advantage over an algorithm using any sort of extended precision arithmetic not directly supported by the CPU. The efficiency gain is a strong incentive to make algorithms robust without using specialized arithmetic. In some situations, it is possible to design a reliable algorithm in the presence of floating-point error. Researchers have suggested several approaches.

One approach is to place a premium on the topological structure of the output, allowing the numerical part of the output to be in error. It may not be possible to determine the correct topology using only fixed-precision arithmetic. However, one can sometimes obtain an output which is topologically reasonable. An algorithm which behaves in this way could be called "topologically stable." For a few problems, researchers have designed such algorithms. Sugihara and Iri [92] compute the Voronoi
diagram of a set of points in the plane in single-precision arithmetic. Even in the presence of numerical error, the algorithm always produces a straight-line graph dividing the plane into cells, such that no two cells share more than one edge. The article cited is written in detail and is a good starting point for understanding topologically stable algorithms. Imai [58] extends this approach to line segments and polygons in the plane. Held employs similar techniques to triangulating planar polygons [51] and to the 2-D line segment Voronoi diagram problem [52].

Such careful algorithm design is not always feasible, especially in a large software system involving many algorithms. A common compromise in large systems is to employ tolerances in geometric computation. The fundamental idea is to assume that two points in space are considered equal if the distance is less than some carefully chosen constant $\delta$. Usually $\delta$ is chosen to be on the order of the desired output precision, which may be $10^{-6}$ but certainly larger than machine epsilon (around $10^{-8}$ for single precision or $10^{-16}$ for double precision). An advantage to this approach is that roundoff error is often subsumed into the tolerance. Difficulties associated with managing roundoff error and detecting degeneracy are combined into a conceptually simpler mechanism.

However, tolerance-based systems exhibit certain undesirable behavior. One such behavior is incidence intransitivity, which is exhibited by the tolerated test for point equality $|p - q| < \delta$. Incidence intransitivity means that one may have $p = q$ and $q = r$ while $p \neq q$. Incidence intransitivity may not cause problems if the geometric data has a minimum feature size many times larger than $\delta$, but enforcing or checking a minimum feature size is difficult. Even if incidence intransitivity does not in itself cause problems, the progress of an algorithm will depend upon which pairs of points are compared for equality. When two points $p, q$ compare equal with respect to $\delta$, the system notionally moves the points together. Other computations in the same algorithm may treat $p$ and $q$ as though they remain in their original position, and topological inconsistencies may result. For an overview of issues in tolerance-based systems, see section 4.2 of Hoffmann [54]. For a specific example of a tolerance-based system, see Segal [82]. Milenkovic [70] proposes a normalization process that preconditions a geometric model for fixed-precision geometric computation.

2.5.2 Exact geometric computation

While pure floating-point methods can be made reliable in some contexts, it appears that exact computation is required for many problems. By exact computation I mean
any form of computation which returns a result that is not an approximation. Exact computation allows the high-level algorithm to assume that numerical computation occurs without error.

A great range of techniques are available for exact geometric computation. Any algorithm that demands only addition, subtraction, multiplication, and division can be implemented over the exact rational numbers, but this is needlessly inefficient in most cases. Instead, expensive arithmetic is usually employed for only part of the algorithm. A common theme in many works is the identification of a predicate, taking numerical inputs and yielding a discrete result (often true/false or \{-1, 0, 1\}), which contains the numerically sensitive part of the algorithm. The predicate is then evaluated exactly, using a specialized numerical model.

**Fixed-precision exact arithmetic.** One may compute in exact rational arithmetic without resorting to arbitrary precision, working within the constraints of a bounded, discrete domain. Greene and Yao [44] and Guibas and Marimont [46] apply the principle of *snap rounding* for arrangements of line segments. In both methods, segment endpoints and intersection points are forced to lie on an integer grid. The algorithms introduce kinks into line segments where necessary. Snap rounding can be viewed as a formalization of tolerancing, ameliorating the consistency problems at the expense of an increase in representation complexity.

**Interval methods.** Many geometric algorithms can be based on a sign-valued predicate. When the predicate is evaluated in interval arithmetic, the sign of the result is known if the resulting interval does not contain zero. This straightforward idea is the basis for much of the recent work on robust geometric computation. The machine-precision interval arithmetic provided by the IEEE floating-point standard typically does not suffice—the interval too often contains zero. Extended precision is sometimes necessary, and a wide variety of ideas have been suggested for extended-precision numeric data types with built-in precision management.

The LEDA system [68] and the CORE system [60] provide extended-precision numeric data types with dynamic error management. In both systems, the data type for a number stores a pointer to a complete expression defining the number. The expression is used to recompute a number to higher precision on demand. The PRECISE system [63] is different mainly in that the expression is not stored. When PRECISE reports insufficient precision, the programmer must increase the precision
and restart the computation. Nevertheless, this is a benefit for computations with very large expressions, such as matrix determinants. Fortune and Van Wyk [38] derive static error bounds on the value of an expression. The bounds are looser, since only a bound on the input values is known at compile time, but the runtime efficiency is very close to pure floating point computation. All of these techniques amount to automatic propagation of forward error through a computation. While such methods can provide exactness at near-floating-point efficiency in some situations, their utility is limited on ill-conditioned problems.

2.5.3 Determinant sign evaluation

Many geometric predicates can be expressed as the sign of the determinant of a matrix. This holds not only for linear problems, such as the convex hull, but also in non-linear contexts where Sturm sequences are used to manipulate algebraic numbers. Many interesting techniques have been proposed for fast, exact determinant sign computation. Most of the techniques assume that the matrix has integer entries. (A matrix with rational or floating-point entries is readily converted to an integer matrix with the same determinant sign.)

The most powerful general technique computes the determinant of an integer matrix using modular arithmetic. The matrix determinant is computed modulo several primes, and the complete determinant is reconstructed from the residues. Brönnimann et al. [9] have recently improved the reconstruction step by avoiding the use of multiprecision arithmetic. The authors have also released an efficient implementation of their algorithm, which works for integer matrices with 53-bit entries. Wiedemann [96] computes the determinant of a sparse matrix in a finite field (modulo a prime number, for example) by computing the characteristic polynomial.

Avnaim et al. [6] give a specialized determinant sign algorithm for $2 \times 2$ and $3 \times 3$ matrices. The algorithm computes the determinant sign using arithmetic on $b$ or $b+1$ bits, where $b$ is a bound on the entry bit length. Brönnimann and Yvinec [10] extend this algorithm, obtaining the “lattice method” for $n \times n$ matrices using arithmetic on $O(b + n)$ bits. Clarkson [18] proposes a determinant sign algorithm based on Gram-Schmidt orthogonalization. The orthogonalization preconditions the matrix for safe determinant sign computation by Gaussian elimination.

Much of the recent research is focused on floating-point filters. For instance, one may compute the determinant in interval arithmetic. If the interval contains zero, the computation is repeated in higher precision or a slower exact method is used as
a fallback. Many floating-point filters are generalizations of this idea. Fortune and Van Wyk [38, 39] analyze the structure of the determinant computation at compile time to obtain static error bounds, minimizing the runtime penalty of error analysis. The LEDA system [68], the CORE system [60, 98], and the PRECISE system [63] all provide numeric data types supporting runtime error analysis and automating higher-precision recomputation. For determinant sign computation, LEDA improves this model by incorporating static error bounds (Burnikel et al. [12]). LEDA and CORE currently handle only small matrices (up to about 6 × 6). The PRECISE system [63] can handle larger matrices, since it does not store an expansion of the determinant expression. Shewchuk [88] uses an unusual form of multiprecision arithmetic to compute signs of matrices up to 4 × 4. In this model, the actual error in each arithmetic operation is carried along, rather than a bound on the error magnitude. All of these methods are based on analyzing the forward error in a numerical computation.

To my knowledge, few authors have attempted a backward error analysis of a determinant algorithm. In two related articles, Pan et al. [76, 77] compute the sign of the determinant by Gaussian elimination and then validate the computation by estimating the distance from the given matrix to the nearest singular matrix.

### 2.5.4 Symbolic computation for irrational numbers

Any rational number can be explicitly represented in a finite number of bits, by its numerator and denominator. Algebraic numbers have, in general, no such representation. An explicit representation of an algebraic number must be an approximation, such as a floating-point number. Exact representations are possible, but are implicit in nature. Research in the field of symbolic computation has produced many interesting methods for representing algebraic numbers.

This dissertation relies on a few standard techniques in symbolic computation. The classical resultants (Sylvester and Macaulay formulations) are used to eliminate variables from two- and three-dimensional systems. Sturm sequences are used to isolate roots of polynomials. These techniques are presented in appendix A as background material.

For three-dimensional systems, I apply a multivariate Sturm sequence technique due to Milne [71]. Milne's technique takes a system of \( n \) polynomial equations in \( n \) unknowns and can isolate the roots of the system within \( n \)-dimensional intervals. In this sense, it is a generalization of univariate Sturm theory. An overview to the
technique is given in appendix A. The details of my efficient implementation are given in section 6.3.1.

2.5.5 Degeneracies

Sections 2.5.1 through 2.5.4 have addressed issues of precision in geometric computation. But precision is only one of the challenges associated with robust computing. The other major challenge, the management of degeneracies, is the subject of this section.

A geometric algorithm typically makes decisions based on the sign of a predicate. In designing the algorithm, it is sometimes simpler to assume that the sign will evaluate to ±1 and never to zero. The zero case may correspond to a configuration that occurs with mathematical probability zero for a random input. The algorithm designer working in floating-point arithmetic is further encouraged to try to ignore the zero case, since a zero answer from the floating-point predicate evaluator is likely to be incorrect anyway. The two most general approaches to the problem are explicit handling and perturbation techniques.

Explicit handling of degeneracies. An algorithm designer may choose to detect and handle all possible degeneracies. The designer must

1. enumerate the possible degeneracies;

2. design a test for each degeneracy;

3. handle each degenerate configuration.

In a complex algorithm, none of these tasks is trivial. Hoffmann ([54], chapter 3) gives a complete algorithm for intersecting two polyhedral solids. His approach is to explicitly detect and handle degeneracies.

Locally, the algorithm requires six intersection routines: face-face, face-edge, face-vertex, edge-edge, edge-vertex, and vertex-vertex. Of these, only the first two occur generically. Hoffmann's algorithm checks for all six possibilities. Some of the routines must address further degeneracies. For instance, two faces either intersect transversely or are coplanar. In the coplanar case, the behavior of the algorithm depends on whether the faces are in the same or opposite orientation. In each degenerate case, specialized methods for resolving the adjacency relationships between created
structures (split faces and edges, introduced edges and vertices) and existing ones. Implementing these cases is likely to be tedious and error-prone.

**Perturbation techniques.** Rather than handle the degenerate cases explicitly, it is sometimes possible to convert an arbitrary input into a generic one by perturbing it symbolically. Several general techniques have been developed and applied to problems with linear predicates, such as determinant signs and polyhedral solid modeling.

Seidel [83] gives an overview to symbolic perturbation. He defines a perturbation formally as follows. Consider a geometric problem as a mapping \( F \) from an input space \( \mathcal{I} \) to an output space \( \mathcal{O} \). For a specific problem instance \( q \in \mathcal{I} \), a perturbation of \( q \) is a continuous curve \( \hat{q} : [0, \infty) \rightarrow \mathcal{I} \) with \( \hat{q}(0) = q \). A perturbation scheme is an assignment of a curve \( \hat{q} \) to each problem instance \( q \).

The fundamental idea of perturbation is to replace the mapping \( F : \mathcal{I} \rightarrow \mathcal{O} \) with a more easily computed mapping \( \hat{F} : \mathcal{I} \rightarrow \mathcal{O} \). The two functions are required to agree on all input instances where \( F \) is continuous. The perturbation allows \( \hat{F} \) to differ from \( F \) at points where \( F \) is discontinuous—the degenerate inputs. At such a degenerate input \( q \in \mathcal{I} \), \( \hat{F} \) may compute the limit of \( F \) at \( q \) from any direction. Any perturbation scheme, as defined above, induces such a function, satisfying

\[
\hat{F}(q) = \lim_{\epsilon \rightarrow 0^+} F(\hat{q}(\epsilon)).
\]

In the cited work, Seidel also describes some fundamental limitations to perturbation techniques. For instance, the convex hull of \( n \) identical points has a simple, special structure, and one might wish that a convex hull algorithm could compute it quickly. But if the algorithm uses a perturbation scheme, the output has the topological structure of the convex hull of \( n \) random points. In other words, the perturbing algorithm produces an output of size \( O(n) \) while the exact output has size \( O(1) \). Even without examining the algorithm, it seems that the price of this perturbation scheme may be quite high.

Emiris and Canny [32, 33] describe an efficient, general perturbation technique and demonstrate its effectiveness on an “InSphere” predicate. Denote by \( q_{i,j} \) the \( j \)th coordinate of the \( i \)th input point (so that \( 1 \leq i \leq n \) and \( 1 \leq j \leq d \), where \( d \) is the dimension). The perturbation adds to each input quantity \( q_{i,j} \) a term depending on a symbolic variable \( \epsilon \):

\[
\hat{q}_{i,j}(\epsilon) = q_{i,j} + \epsilon \cdot v^j.
\]

38
This scheme perturbs each coordinate by a different amount, as a scheme must in order to ensure that the perturbed problem is generic. The "InSphere" test is computed as a matrix determinant, which is expanded symbolically as a polynomial in $\epsilon$. In order to compute the sign of the limit of this polynomial as $\epsilon \to 0$, the algorithm simply computes the signs of the polynomial coefficients in order of increasing power of $\epsilon$. The first non-zero sign in this sequence is the sign of the perturbed predicate. Since the lowest-order term is $\epsilon^0$ times the original matrix determinant, no extra work is required for a non-degenerate instance of the predicate.

Fortune [37] describes a polyhedral solid modeler that uses symbolic face perturbation to remove the degeneracy of four planes meeting at a point.

Though perturbation techniques are effective in theory, they have not been applied to the polyhedral medial axis problem (nor to any closely-related problem). I believe that there are two chief obstacles. First, no simple medial axis algorithm has been devised. One way to achieve a "simple" algorithm would be to express all geometric decisions as signs of a few algebraic expressions or matrix determinants. I know of no such medial axis algorithm. The other obstacle is the non-linearity of the medial axis. The previous work on perturbation techniques has focused on linear geometric constructions such as the Voronoi diagram of points. To my knowledge, no perturbation scheme has been developed for an algorithm manipulating algebraic curves or surfaces.

2.6 Summary

Even though the medial axis is an important and fundamental construction, previous methods are mainly based on approximation. The few continuous approaches may be subject to failure due to roundoff error. The roundoff problem has been examined in the context of geometric algorithms, but solutions have been applied mainly to two-dimensional problems and to problems requiring only linear computations.

The remainder of this dissertation describes a continuous approach to computing the polyhedral medial axis. My approach defends against failure due to roundoff error by using exact computation.
Chapter 3

An Accurate Medial Axis Algorithm

This chapter gives an overview of my medial axis algorithm. Details are given in the following two chapters.

3.1 Terminology

My terminology for the parts of the medial axis is similar to that of Sherbrooke et al. [87]. The boundary of the input polyhedron consists of faces, edges, and vertices. The coordinates of the vertices are assumed to be rational numbers. The faces, edges, and vertices are collectively called boundary elements.

Figure 3.1 shows the anatomy of a polyhedral medial axis. The medial axis consists of sheets, seams, and junctions. They form the faces, edges, and vertices of a curved, non-manifold surface. (From now on I shall use the words “face,” “edge,” and “vertex” only in reference to the input polyhedron.) The sheets are trimmed quadric surfaces. The seams are algebraic curves of maximal degree four, with rational coefficients. The junctions are points whose coordinates are algebraic numbers of degree at most eight. (Since figure 3.1 is convex, the sheets are planar and the seams linear.)

A sheet is said to be governed by two boundary elements, meaning that the sheet is equidistant from those two elements. Since each governor is either a face, an edge, or a vertex, there are six basic sheet types. The types include parabolic cylinders, elliptical cones, and hyperbolic paraboloids. The shape is further affected by the configuration of the governors. The set of possible sheet morphologies, as determined by the governors and their configuration, is enumerated in chapter 4.
A seam is governed by three or more elements. Generically, there are exactly three governors, but the degenerate case of four or more governors is not uncommon. For example, the box in figure 3.2 has a central seam with four governors.

A junction is governed by four or more elements. Again, it is not unusual to have degenerate junctions with five or more governors. A cube, for example, has a single internal junction with six governors.

The medial axis is a subset of the generalized Voronoi diagram of the faces, edges, and vertices. The algorithm computes the portion of the generalized Voronoi diagram that lies within the polyhedron. As a post-process, it then removes certain sheets, leaving the medial axis. The sheets removed are the bisectors of a face with its edge, an edge with its vertex, a face with a parallel adjacent face, and an edge with a parallel adjacent edge.

### 3.2 Overview of the accurate tracing algorithm

At a high level, my accurate algorithm is similar to the tracing algorithms mentioned in section 2.4. This section describes my algorithm at this high level. The differences between my tracing algorithm and other tracing algorithms lie in the details of tracing: these details are given in chapter 5.

My algorithm starts by finding a single seam. If the polyhedron has a convex, trivalent vertex (having exactly three incident edges), its three incident faces govern a seam. For example, any of the vertices of the polyhedron in figure 3.1 qualify. The first step of the algorithm is to construct this seam and trace it to its terminus, an internal junction. From this junction, the algorithm forks, tracing the other seams

![Diagram](image.png)

Figure 3.1: The geometric elements of the medial axis. For convenience, the polyhedron's vertices are considered junctions.
Figure 3.2: A simple box with a degenerate medial axis. The central seam has four governors, instead of the generic three. The two internal junctions have five governors, instead of the generic four.

that commence at the junction. Tracing the initial seam is only slightly different from tracing any other seam. I shall describe the general technique first.

**Tracing a typical seam.** The seam tracing problem is illustrated in figure 3.3. The seam has three (or more) governors. Suppose the governors are the boundary elements $e_1, e_2, e_3$. One of the seam endpoints is known—a junction $J_{\text{start}}$, whose governors include the seam governors and one or more others. Suppose the governors of $J_{\text{start}}$ are $e_0, e_1, e_2, e_3$. The extra governor $e_0$ is called the *back element* since tracing will proceed away from it.

From the seam governors, the algorithm constructs an algebraic curve, consisting of points simultaneously equidistant from all of the governors. The tangent line at $J_{\text{start}}$ is readily computed, but the algorithm must choose between the two possible tangent directions. The correct direction is the one which takes the seam away from the back element, relative to the seam governors. More formally, the functions $d_{e_0}^2 - d_{e_1}^2$, $d_{e_0}^2 - d_{e_2}^2$, and $d_{e_0}^2 - d_{e_3}^2$ must all initially increase along the seam. This is a question of the local geometry at $J_{\text{start}}$. The situation is formalized in section 5.1.

The goal is to find the *terminating junction* $J_{\text{term}}$ and its single unknown governor $e_4$, the *terminating element*. The algorithm proceeds by forming a list of candidates for terminating element. In the worst case, this list may include nearly all $n$ boundary elements.

Each candidate $e$ determines up to eight candidate junctions. The candidate junctions are the points simultaneously equidistant from $e_1, e_2, e_3$, and $e$. There are at most eight, since they correspond to the roots of three quadratic equations in
Figure 3.3: An illustration of a step in the seam-tracing algorithm.
three unknowns. All of the candidate junctions—there may be approximately $8n$ in the worst case—are located on the algebraic seam curve. The terminating junction is the candidate that occurs first along the seam curve. Finding the correct terminus requires ordering the candidate junction points along the curve.

If the terminus is a new interior junction, three (or more) new seams are created. The new seams are inserted into a to-do list of unsearched seams. The terminus may instead be an internal junction that is already part of the medial axis, or it may be a vertex of the polyhedron. In these cases, no new seams need be created. When the to-do list is empty, the algorithm terminates.

**Tracing the initial seam.** The initial seam is special in that it does not have a starting junction $J_{\text{start}}$ in the usual sense. If the initial seam starts from a trivalent vertex, $J_{\text{start}}$ is the vertex itself. Moreover, this vertex is used as the back element $e_0$, even though convex vertices are not otherwise treated as governors.

If the polyhedron has no trivalent vertex, any other vertex can be used. However, the vertex is a degenerate junction with more than four governors. The algorithm in section 5.1 is used to find any one seam incident to the vertex, and this seam is then traced in the usual way.

**The medial axis transform.** The *medial axis transform* of an object consists of its medial axis together with a *radius function*, which assigns to each point on the axis the radius of the maximal ball at that point. The radius function is constructed by composing the parametric form of a sheet, quadratic in $s$ and $t$, with the distance function to either of its governors, a quadratic in $x, y$, and $z$. The radius function is in general the square root of a degree-four expression in $s$ and $t$.

### 3.3 Exact arithmetic

My algorithm assumes that the boundary of the polyhedron can be represented using rational numbers. However, the medial axis of such a polyhedron generally has no exact, explicit representation in rational numbers. There is a fundamental limitation to the use of exact rational arithmetic in Euclidean geometry: constructions on rational numbers often result in irrational algebraic numbers. For an example in the plane, consider the set of points equidistant from the line $y = 0$ and the line $y = x$ (figure 3.4). The bisector of these two lines is the union of the lines $y = (-1 + \sqrt{2})x$
and \( y = (-1 - \sqrt{2})x \). But only one of the bisector lines is relevant to the medial axis. Identifying the relevant line is straightforward, but representing it individually requires an irrational number. Describing the line by two points, instead of by an equation in \( x \) and \( y \), does not avoid the difficulty, since there is only one point on either line with rational \( x \) and \( y \) coordinates (the origin).

This situation poses a difficulty for any exact medial axis algorithm. Some bisectors have no convenient, exact representation. Worse, such bisectors are not rare (the precise conditions are given in chapter 4). Three options present themselves for bisectors such as the one in figure 3.4.

1. Use an approximate representation where necessary.

2. Use an exact representation such as \( y = (-1+\sqrt{2})x \). Perform exact computation with algebraic numbers.

3. Use an exact representation such as \(-x^2 + 2xy + y^2 = 0\). Do not distinguish between the two lines in the bisector representation.

Exact computation does not prohibit the first option. approximating the bisector. However, perturbing a bisector slightly can cause seams and junctions to appear and disappear from the medial axis. It is not clear how good the bisector approximation must be in order to get the correct medial axis topology. An exact representation for all bisectors is therefore preferable.

The second option requires exact computation with irrational numbers. The simplest technique is to represent expressions such as \( -1 + \sqrt{2} \) symbolically. This is, in fact, the approach taken by software systems such as Mathematica. As an experiment, I implemented a two-dimensional medial axis tracing algorithm in Mathematica, using that system’s facility for algebraic numbers. The algorithm encounters only numbers

Figure 3.4: (a) The lines \( y = 0 \) and \( y = x \). (b) Their bisector is the pair of lines represented by \(-x^2 + 2xy + y^2 = 0\), an equation that does not factor over the rationals.
of algebraic degree four or less, which can always be expressed by radicals. Mathematica represents these as symbolic expressions with radicals (no Root objects are necessary). Figure 3.5 shows the medial axis of a hexagon formed by taking the regular hexagon with vertices on the unit circle and perturbing its vertices by rounding to rational numbers, moving each vertex by no more than $10^{-16}$. Unfortunately, this simple example took approximately 30 minutes of running time. Several other small examples failed to complete within 24 hours. I hypothesize that the main difficulty is the complexity of the symbolic expressions that must be evaluated during tracing.

A deeper problem with option 2 is that symbolic expressions with radicals cannot always be used for algebraic numbers of degree five and higher. In two dimensions, the medial axis requires degree-four computations only. In three dimensions, degree-eight algebraic numbers appear, necessitating a more sophisticated representation. The representation should support the rational arithmetic operations. An algebraic number can be represented, for example, by its minimal polynomial together with an identifying integer. Unfortunately, arithmetic on such representations is very expensive.

I have chosen option 3 for my medial axis algorithm. I avoid arithmetic involving such algebraic numbers, choosing instead to design implicit geometric data structures that use only rational numbers. In the example of figure 3.4, I would represent the pair of lines together as the degenerate conic $-x^2 + 2xy + y^2 = 0$.

One chief disadvantage to option 3 is that the bisector's representation includes an irrelevant component. I will show that this presents no serious obstacle to the algorithm. Still, the choice complicates many of the details.

The other important disadvantage is that the representation has a higher algebraic degree than the geometry it represents. As a result, operations on these bisectors are

![Magnified 10^{15} times](image)

Figure 3.5: The medial axis of a near-regular hexagon. The central region is highly magnified to reveal its combinatorial structure.
more expensive than they might otherwise be. On the other hand, the chief advantage of option 3 is that operations on these bisectors should be no more expensive than similar operations performed on truly curved medial surfaces.

3.4 Exact geometric representations

This section describes exact representations for the sheets, seams, and junctions of the medial axis. The parts are illustrated in figure 3.1.

A sheet is the bisector of two boundary elements. Sheets are quadric surfaces, amenable to two different representations. The implicit form is an equation

\[ F(x, y, z) = 0 \]

where \( F \) is a polynomial of total degree 2. The rational parametric form consists of three rational functions

\[
\begin{pmatrix}
X(s, t) & Y(s, t) & Z(s, t) \\
W(s, t) & W'(s, t) & W''(s, t)
\end{pmatrix}
\]

where \( X, Y, Z, \) and \( W \) are polynomials of total degree 2. Because the coefficients are always rational, the implicit form is the primary bisector representation in my medial axis algorithm. The algorithm sometimes uses the parametric form as well. Chapter 4 discusses sheet morphology and the computation of the implicit and parametric forms. Section 5.3 describes a method for finding the boundaries of the sheets.

A seam curve lies on three different sheets. It is represented as the intersection of any two of these sheets. An explicit rational parameterization of these curves is, in general, impossible, so the curves are kept in implicit form. The implicit form \( G(s, t) = 0 \) is obtained by substituting the parametric form of one surface (when its coefficients are rational) into the implicit form of the other. Computations involving the curve are sometimes formulated in the \((s, t)\) plane. Section 5.2 describes the representation of seams and algorithms for searching them.

A junction is an algebraic point, represented jointly by

- a system of three algebraic equations in three variables with rational coefficients;
- a three-dimensional box with rational coordinates containing exactly one root of the system.

With 3-D Sturm sequences, the algorithm can find a box with a single root and refine it to an arbitrarily small positive volume. My approach is based on the volume
function of Milne [71]. Reasoning about the point typically involves reasoning about the box, refining the box until a query can be answered unambiguously. Junctions are also sometimes represented as a 2-D interval on a surface or as a 1-D interval on a curve. The representation is dictated by the morphology of the seam being traced.

3.5 Technical assumptions

I conclude this overview with some technical assumptions I make on the input polyhedron.

I assume that the polyhedron's boundary is a closed orientable surface. This assumption is required for the medial axis to be well-defined. The boundary may have arbitrary genus. I further assume that the polyhedron is bounded.

The faces of the input polyhedron are assumed to be convex polygons. Non-convex faces are subdivided into convex subfaces. In the worst case, this increases the input complexity by a constant factor. The subdivision may create parallel adjacent faces, a degenerate configuration which nonetheless causes no difficulties to the algorithm. Another need for subdivision arises when the polyhedron possesses faces that are not simply connected. In this situation the medial axis may be disconnected. Any approach based on tracing would require subdividing such faces.

Finally, I assume that the vertices of the input polyhedron are given in rational numbers. This is often satisfied in practice, since integers and floating-point numbers are rational. However, it excludes a polyhedron defined by, for instance, a rotation of $\pi/4$ around the z-axis of a polyhedron with integer vertices. For such an input, my algorithm computes a rational approximation to the input. The approximation process may inadvertently produce an invalid polyhedron, but apart from this difficulty, the approximation should not be troublesome. Of course, the medial axis itself cannot in general be described by exact rational numbers, and any usable output format would also be an approximation.
Chapter 4

Distance Functions and Bisectors

In this chapter and the following chapter, I give the details of the medial axis algorithm. This chapter constructs the distance functions and bisector surfaces. The material in this chapter is applicable to any continuous medial axis algorithm. The following chapter uses the tracing method to construct the junctions, seams, and sheets of the medial axis. The junctions, seams, and sheets are all defined algebraically in terms of bisector surfaces.

In section 4.1 I formulate the squared distance functions for a point, line, and plane in $\mathbb{R}^3$. The distance functions for a vertex, edge, and face of a polyhedron are obtained by restricting the point, line, and plane distance functions to certain convex regions, the domain polytopes, defined in section 4.2.

A bisector is the set of points equidistant from two geometric elements. The 3-D polyhedral medial axis problem requires bisectors among points, lines, and planes. Such bisectors are quadric surfaces. An implicit form for the bisector surface is obtained by setting two distance functions equal. A parametric form is possible as well, since the bisectors are quadric surfaces. Section 4.3 gives methods for computing representations of bisector surfaces in both implicit and parametric form.

4.1 Distance functions

The Euclidean distance from a point $p$ is a non-linear function on $\mathbb{R}^3$:

$$d_p(x) = \sqrt{(x_1 - p_1)^2 + (x_2 - p_2)^2 + (x_3 - p_3)^2}$$

The analogous distance functions for a line segment and a polygon are likewise non-linear. The line segment and polygon functions are further complicated by requiring
piecewise definitions. For instance, the distance from \( x \) to a line segment \( l \) is either the distance to the nearest endpoint or the distance to the projection of \( x \) onto \( l \), depending on the location of \( x \).

The distance functions can, however, be expressed in purely algebraic terms. The \textit{squared distance functions} for points, lines, and planes are degree-two polynomials in the coordinates \((x_1, x_2, x_3)\) of the point \( x \).

- Squared distance from \( x \) to a point \( p = (p_1, p_2, p_3) \):
  \[
d^2_p(x) = \sum_i (x_i - p_i)^2
  \]

- Squared distance from \( x \) to a line \( l \), parametrized by \( a + tv \):
  \[
d^2_l(x) = \sum_i \left( x_i - a_i - v_i \cdot \left( \frac{v \cdot x - v \cdot a}{v \cdot v} \right) \right)^2
  \]

- Squared distance from \( x \) to a plane \( h \), defined by \( a \cdot x + D = 0 \):
  \[
d^2_h(x) = \frac{(a \cdot x + D)^2}{a \cdot a}
  \]

These algebraic forms are more manageable than the actual Euclidean distance functions. Thus they are the preferred notion of distance used through most of this dissertation. However, I will occasionally have use for the actual Euclidean distance from a point, line segment, or planar convex polygon. I use the term \textit{global distance} for this concept. The global distance functions are square roots of piecewise-quadratic functions of \( x, y, z \). For line segments and convex polygons, the domains of the pieces are defined by the closest feature.

- Global distance from \( x \) to a point \( p \):
  \[
d_p(x) = \sqrt{d^2_p(x)}
  \]

- Global distance from \( x \) to a line segment \( L \), contained in the line \( l \) and having
endpoints \( p_1, p_2 \):

\[
d_L(x) = \begin{cases} 
  d_{p_1}(x) & \text{if } x \text{ is closest to } p_1, \\
  d_{p_2}(x) & \text{if } x \text{ is closest to } p_2, \\
  \sqrt{d_L^2(x)} & \text{if } x \text{ is closest to a point in the relative interior of } L.
\end{cases}
\]

- Global distance from \( x \) to a convex polygon \( P \), contained in the plane \( h \) and having vertices \( \{p_i\} \) and edges \( \{L_i\} \):

\[
d_L(x) = \begin{cases} 
  d_{p_i}(x) & \text{if } x \text{ is closest to } p_i, \\
  d_{L_i}(x) & \text{if } x \text{ is closest to } L_i, \\
  \sqrt{d_h^2(x)} & \text{if } x \text{ is closest to a point in the relative interior of } P.
\end{cases}
\]

The term *global distance function* is used to emphasize that the functions are defined over all of space.

The global distance is difficult to use in algebraic algorithms. Throughout most of this dissertation I will use the squared distance functions. Where the global distance is used, the distinction will be drawn carefully.

### 4.2 Domain polytopes

The squared distance from a point, line, or plane is a quadratic function. Measuring the distance from a vertex, edge, or face of a polyhedron is somewhat more complicated. Take, for example, the two-dimensional situation in figure 4.1. The point \( p \) is closer to the edge \( e' \) than to the edge \( e \). But measured from the *lines* containing the edges, \( p \) is closer to \( e \). Clearly, \( p \) is nowhere near \( e \) for the purpose of computing the medial axis of the polygon.

The problem can be resolved by limiting the domain of the distance function of \( e \) to the infinite region shaded in the figure. Any point to the left or right of the region cannot be the center of a circle tangent to \( e \), since such a point is closer to an endpoint of \( e \) than to \( e \) itself. Further, any point above the region cannot be the center of an inscribed sphere tangent to \( e \). The shaded region is called the *domain*
polytope of \( e \). Reflex vertices such as \( v \) have triangular domain polytopes. One can consider the squared distance function to be defined by the appropriate quadratic formula inside the domain polytope and by \(+\infty\) elsewhere.

This notion extends easily to 3-D. Formally, the domain polytope of a boundary element \( e \) is the closure of the set of points that

1. are closer to \( e \) than to the subfacets and superfacets of \( e \), and

2. appear to lie inside the polyhedron from the perspective of \( e \).

By “subfacets and superfacets” I mean a face’s edges and vertices, an edge’s two endpoints and two incident faces, and a vertex’s incident edges and faces. In 2-D, the constraints are half-planes formed by (1) lines through the vertices perpendicular to the incident edges and (2) lines containing the edges. In 3-D, the constraints are half-planes formed by (1a) planes through the edges perpendicular to the incident faces, (1b) planes through the vertices perpendicular to the incident edges, and (2) planes containing the faces. Domain polytopes in 3-D are illustrated in figure 4.2 and described explicitly as follows.

- The face polytope is formed by taking the half-space defined by the face itself, and a half-space orthogonal to each edge. It is the space swept out by the face as it moves orthogonally toward the interior of the polyhedron.

- The edge polytope is formed by four half-spaces. Two are crowns, planes orthogonal to the edge at the vertices, and two are skirts, planes containing the edge and orthogonal to the adjacent faces. The edge polytope is tent-shaped, with an interior angle less than \( \pi \).

- The vertex polytope is formed by a crown from each incident edge. The shape of the vertex polytope is pyramidal.

In summary, both types of constraint on the domain polytope are linear in nature. Therefore, the domain polytope is defined by the intersection of half-spaces, and the name “polytope” is justified.

The domain polytope contains the internal Voronoi region. This is easily seen by characterizing the internal Voronoi region of \( e \) as the set of points that

1. are closer to \( e \) than to any other boundary element, and

2. appear to lie inside the polyhedron from the perspective of \( e \).
Figure 4.1: Domain polytopes in 2-D. The edge \( e \) has influence only over the shaded region on the left. This region excludes \( p \), so the squared distance from \( p \) to the line containing \( e \) is taken to be \(+\infty\). The reflex vertex \( v \) has influence only over the shaded region on the right.

The internal Voronoi region is defined by introducing more constraints onto the domain polytope. (Of course, these constraints are non-linear.)

The main virtues of these polytopes are that they are simple, linear, and locally defined. A data structure for polytopes needs to support the following operations:

1. an interior/exterior test on a point
2. polytope intersection
3. an empty/nonempty test
4. a test for positive volume
5. list all vertices

Operations 1–2 can be supported by a simple list of plane equations. To support operations 3–5, I compute polytopes with a randomized incremental algorithm [72]. The algorithm runs in \( O(a \log a) \) expected time, where \( a \) is the number of plane equations. In many situations, \( a \) may safely be assumed to be a small constant. The fundamental geometric operation in the incremental algorithm is expressed as the sign of the ratio of a \( 4 \times 4 \) determinant and a \( 3 \times 3 \) determinant. The determinant-sign operation takes advantage of the floating-point filter described in section 6.3.2. For operations 4 and 5, the vertex locations are computed in exact rational arithmetic.

The domain polytope has a closer relationship to the Voronoi region than containment. The two regions also have the same dimension. Empty and zero-volume
Figure 4.2: Polytopes containing the Voronoi regions of a face (top row), an edge (middle row), and a vertex (bottom row).
Voronoi regions can be detected by examining domain polytopes. For example, there are three edge types in a polyhedron:

1. a reflex edge, whose domain polytope and internal Voronoi region both have positive volume;

2. a convex edge, whose domain polytope and internal Voronoi region both are empty; and

3. an edge whose incident faces are parallel.

An edge governs part of the medial axis if and only if it is in category (1). These edges are called active edges, and it is reasonable to identify the active edges before computing the medial axis. Categorizing an edge is straightforward. One can test its angle against π, or, in exact arithmetic, one can test the midpoint of one face against the plane equation of the other. Another way is to compute the edge's domain polytope and test it for positive volume.

Similarly, a vertex is called active if its domain polytope (and therefore its internal Voronoi region) has positive volume. The active vertices are exactly those that govern part of the medial axis. Unlike an edge, a vertex can be neither convex nor concave nor flat. and such a vertex may or may not be active. I determine whether a vertex is active by constructing its domain polytope and testing it for positive volume.

In summary, domain polytopes are an important consideration throughout the medial axis algorithm. The interior of a boundary element's polytope is exactly the region where its distance function is valid. This property is used to limit the parametric domain for some bisector surfaces (see sections 4.3.3.3 and 4.3.3.4). Another useful property is that the polytope contains the Voronoi region. This is useful for culling away potential seam terminators during the search step (see section 6.1.2). A third important property is that the polytope has positive volume if and only if the internal Voronoi region has positive volume. My algorithm classifies the active and inactive elements before computing the medial axis.

4.3 Bisectors

The bisector of two boundary elements is the algebraic surface formed by the locus of equidistant points. The medial axis is made up of bisectors, in the sense that
the generic medial axis point lies on the bisector of two boundary elements. Further, the seams and junctions are defined by bisector intersections. Bisectors are the fundamental algebraic object used in polyhedral medial axis computation.

The implicit and parametric forms of a bisector surface are computed from its two governors. This section describes the construction of both forms.

4.3.1 Bisector morphology

Bisectors fall into six basic categories defined by the governors: vertex-vertex, vertex-edge, vertex-face, edge-edge, edge-face, and face-face. These six basic categories are further divided into types according to the relative position of the governors. The types correspond to geometric shape. I define the morphology of a bisector to be its geometric type as determined by the implicit form. In figure 4.3, I describe the morphology of all of the bisectors that arise in the polyhedral medial axis.

Observe that some of the non-generic configurations arise in perfectly generic polyhedra. For instance, the bisector of a plane \( h \) and a line \( l \) lying in \( h \) occurs between the Voronoi region of a face and that of an adjacent edge.

The last column indicates whether the coefficients of the parametric form of the surface are rational numbers, assuming that the polyhedron's vertices (equivalently, its face equations) have rational coefficients. The implicit form always has rational coefficients under this assumption.

4.3.2 Implicit form

The implicit form of the bisector is the equation

\[
F(x, y, z) = 0
\]

where \( F \) is a polynomial of total degree 2. The polynomial \( F \) is the difference of two squared distance functions, as given in section 4.1.

A few governor configurations require a specialized bisector definition. For instance, the bisector of a plane \( h \) with a contained line \( l \) is the plane orthogonal to \( h \) containing \( l \). However, the difference of the distance functions yields a quadratic equation, not a linear equation. The quadratic is actually the bisector plane counted twice. Instead of attacking this algebraic issue directly, I observe that the bisector plane can be computed directly from \( h \) and \( l \) without recourse to their squared
<table>
<thead>
<tr>
<th>Governors</th>
<th>Configuration</th>
<th>Bisector</th>
<th>Coeffs in $\mathbb{Q}$?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-point</td>
<td>Generic</td>
<td>Plane</td>
<td>Yes</td>
</tr>
<tr>
<td>Point-line</td>
<td>Generic Incident</td>
<td>Parabolic cylinder $(z = x^2)$</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Incident Plane</td>
<td>Parabolic cylinder $(z = x^2 + y^2)$</td>
<td>Yes</td>
</tr>
<tr>
<td>Point-plane</td>
<td>Generic Incident</td>
<td>Circular paraboloid $(z = x^2 + y^2)$</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Incident Plane</td>
<td>Hyperbolic paraboloid $(z = x^2 - y^2)$</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Parallel Plane</td>
<td>Orthogonal plane pair $(z^2 + x^2 = 0)$</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Line lies on plane</td>
<td>Elliptical cone $(z^2 = x^2 + y^2)$</td>
<td>No</td>
</tr>
<tr>
<td>Plane-plane</td>
<td>Generic Parallel</td>
<td>Orthogonal plane pair</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Parallel Plane</td>
<td>Plane</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Perpendicular</td>
<td>Orthogonal plane pair</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Figure 4.3: The bisectors that arise in the medial axis.

distance functions.

The morphology (geometric type) of the bisector surface can be determined entirely from the implicit form. Methods are given by Spain [89] and Coolidge [20]. However, it is also possible to deduce the morphology from the configuration of the governors. This is more efficient than an analysis of $F$, and often reveals extra information. For an example of such extra information, consider a point-plane bisector. Classifying a point-plane bisector requires testing whether the point lies on the plane. This determines whether the bisector is a paraboloid or a degenerate line. But if the point lies behind the plane (that is, outside the polyhedron from the face's point of view), the bisector surface does not participate in the medial axis, and it can be skipped.

### 4.3.3 Parametric form

Every quadric surface has a rational parametrization (see an algebraic geometry text such as Shafarevich [84], pp. 6–7 and 38). That is, every bisector surface in the medial axis of a polyhedron can be represented as

$$\left( \frac{X(s,t)}{W(s,t)}, \frac{Y(s,t)}{W(s,t)}, \frac{Z(s,t)}{W(s,t)} \right)$$

where $X, Y, Z,$ and $W$ are polynomials of total degree 2.
A parametrization with rational coefficients exists if and only if a nonsingular point $P$ with rational coordinates lies on the surface. For the medial axis problem, it is possible to deduce the existence of a parametrization in rational coefficients from the governor configuration. For a general implicit quadric, Legendre's theorem gives a condition for the existence of a rational $P$. See Borevich and Shafarevich [8], chapter 1, theorem 7.2.

Instead of using a general surface parametrization method such as stereographic projection, I use specialized methods for the various bisector morphologies. These methods are described in the six sections below. In each case, the goal is to find rational expressions for $X, Y, Z$ in terms of $s, t$ and to find a rectangular region of the $(s, t)$-plane whose image under the parametrization covers the relevant portion of the bisector. The second part of the problem, finding a domain, is in some cases more difficult than the parametrization. The degenerate morphologies are mentioned briefly at the end of each section: the missing details for these constructions are straightforward.

### 4.3.3.1 Edge-edge

The generic edge-edge bisector is a hyperbolic paraboloid (figure 4.4). The parametric formulation is from Elber and Kim [30], who construct a parametrization for the bisector surface of two rational space curves. In particular, their construction finds the bisector of two lines and furthermore, it solves the problem of finding an appropriate domain for the parametrization.

Let $s$ vary linearly along one edge, so that $s = 0$ at one endpoint and $s = 1$ at the other endpoint. Let $t$ similarly parametrize the other edge. As $s$ and $t$ vary over the unit interval, denote the corresponding edge-points $S$ and $T$. For $s, t \in [0, 1]$, associate a unique point $P$ on the bisector as the intersection of the three planes:

- the plane orthogonal to the first edge and containing $S$;
- the plane orthogonal to the second edge and containing $T$;
- the planar bisector of the points $S$ and $T$.

The coefficients of the first two plane equations vary linearly with $s$ and $t$. The coefficients of the third are quadratic expressions in $s$ and $t$. By Cramer's rule, the solution to the linear equation is a rational expression in these coefficients. Thus, the resulting parametrization is a polynomial expression in $s$ and $t$. 

60
The idea is equivalent to mapping each point on the bisector surface to its orthogonal projection onto each of the two edges. The bisector point can participate in the medial axis only if its projection onto each edge-line lies within the edge, that is, when \( s.t \in [0, 1] \). The unit square contains the relevant portion of the bisector.

The procedure described here works when the edge-lines are skew. When they are parallel, the bisector is a plane, as in the vertex-vertex case. When the edge-lines intersect, the bisector is a plane-pair with irrational coefficients, as in the face-face case.

### 4.3.3.2 Vertex-face

The generic vertex-face bisector is a circular paraboloid (figure 4.5). By analogy with the edge-edge case, I let \( s.t \) parametrize the face-plane linearly. Let \( S \) be the point corresponding to \((s. t)\). A unique point \( P \) on the bisector is formed by the intersection of

- the line orthogonal to the face and through \( S \);
- the planar bisector of \( S \) and the vertex.

An argument similar to the one in the previous section (4.3.3.1) applies to show that this yields a polynomial parametrization. Further, any \((s.t)\)-domain that contains the entire face must contain the relevant part of the bisector. This solution is equivalent to projecting the bisector onto the face-plane.

If the vertex lies on the face-plane, the bisector is not a surface at all: it is a line, namely the line orthogonal to the plane at the vertex. The line is a redundant part of the medial axis and can be ignored.

![Figure 4.4](image.png)

**Figure 4.4:** The generic edge-edge bisector is a hyperbolic paraboloid. Isoparametric curves are shown.
4.3.3.3 Vertex-edge

The generic vertex-edge bisector is a parabolic cylinder (figure 4.6). Let $s$ be a parameter along the edge, varying linearly from 0 and 1 between the endpoints. Construct any line $l$ orthogonal to the plane containing the vertex and the edge, and let $t$ parametrize $l$. Define the points $S$ and $T$ as before. Construct a point on the bisector by the intersection of

- the plane orthogonal to the edge at $S$;
- the plane orthogonal to $l$ at $T$;
- the planar bisector of the points $S$ and $T$.

There remains the question of finding a bounded range for $t$. The algorithm makes use of the intersection of the domain polytopes of the two governors. The intersection is a polytope containing the relevant part of the bisector. The polytope’s vertices are projected onto $l$, and the extreme projections are used as the endpoints of the $t$-interval.

If the vertex is one of the edge’s endpoints, the bisector is the plane orthogonal to the edge and containing the vertex. The plane’s coefficients are rational.
Figure 4.6: The generic vertex-edge bisector is a parabolic cylinder. Isoparametric curves are shown.

4.3.3.4 Vertex-vertex

The bisector of two points is a plane. When the implicit form of the bisector is computed as in section 4.3.2, the quadratic terms cancel, leaving the plane equation. Finding a parametrization is equivalent to finding three non-collinear points $P, Q, R$ on the plane. The plane is parametrized by, say, $P + s(Q - P) + t(R - P).

In order to find a bounded domain, intersect the two domain polytopes with the bisector plane. This yields a convex polygon. Find the $(s, t)$-coordinates of the polygon’s vertices, and extend the $s$ and $t$ intervals to contain these points.

4.3.3.5 Face-face

The generic bisector of two signed planes is a plane. However, the implicit form’s quadratic terms do not cancel as they do in the point-point case. In fact, the implicit form gives not only the bisector plane, but also an orthogonal plane (figure 4.7). One plane is the desired bisector, lying inside the polyhedron from the point of view of the governors. The other plane is perpendicular, lying outside the polyhedron from the point of view of the governors, and is therefore not part of the medial axis.

The implicit form can be factored into two linear equations, one for each plane, but in general, the coefficients are irrational. Rather than compute directly with irrational numbers, or use an approximate representation of the plane, I structure the algorithm so as to rely only on the implicit form, and discard the effects of the unremovable second plane of the pair as they appear.

Let $l$ be the line at the intersection of the face-planes. Generally, $l$ contains all points with rational coordinates on the bisector plane-pair. Choose two points $P, Q$ on $l$. A third, non-collinear point $R$ is found by scaling the face normals to have the
same length, taking the average, and adding the result to \( P \). The coordinates of \( R \) are rational if and only if the scaling can be done with a rational number, that is, if the ratio of the lengths of the face normals is rational. This can be easily checked by testing whether the ratio of the squared lengths is a square number. Usually it is not, but when it is, the parametric form of the plane can be used, and the degree-two implicit form can be replaced with the degree-one plane equation. This happens when, for instance, the two faces are perpendicular. Otherwise, the bisector plane is computed by the same procedure in approximate arithmetic, for visualization and output.

The important case of coplanar adjacent faces is simpler. The bisector plane has rational coefficients and contains the two common vertices and a third point found by adding the common face-normal to a vertex. Coplanar adjacent faces occur whenever a non-convex face has been split into convex subfaces. Coplanar nonadjacent faces do not have a relevant bisector. Parallel, non-coplanar faces have a planar bisector with rational coefficients.

### 4.3.3.6 Edge-face

The generic bisector of a line and a plane is an elliptical cone (figure 4.8). The vertex of the cone has rational coordinates, and lies at the intersection of the line and the plane. In general, the vertex is the only point on the cone with rational coordinates, and no rational parametrization exists with rational coefficients. Only the implicit form is used by the medial axis algorithm.

An approximate rational parametrization is nonetheless useful for applications in solid modeling. The parametric form of the elliptical cone is computed in floating-point arithmetic and included in the output. A right circular cone in standard position is mapped by an affine transformation to the bisector. The cone \( z^2 = x^2 + y^2 \) is parametrized

\[
\left( \frac{1 - s^2}{1 + s^2} t, \frac{2s}{1 + s^2} t, t \right).
\]

There exists an affine transformation from this cone to the bisector, though its coefficients are irrational. In floating-point arithmetic, the origin is mapped to the vertex of the bisector cone and the circle at \( z = 1 \) to a cross-sectional ellipse of the bisector.

With this parametrization, one line in the cone corresponds to \( s = \infty \). Any finite \( s \)-interval excludes this line and a surrounding region on both sides. The algorithm chooses the position of the \( s = \infty \) line so that it does not lie in the relevant portion
Figure 4.7: The generic face-face bisector is a plane. The bisector plane is horizontal in this figure. The implicit form yields not one plane but an orthogonal plane pair, which cannot generally be separated without irrational numbers.

of the bisector. There are two cases. First, if the line-plane intersection lies outside the face or on the face’s boundary, the $s = \infty$ line is placed so that the projection of its “upper half” lies outside the face. The “upper half” is the portion of the line lying on the positive (interior) side of the face-plane. For example, in figure 4.8, the $s = \infty$ line might go from the upper left to the lower right. In the second case, the line-plane intersection lies inside the face. Here the $s = \infty$ line is placed anywhere outside the domain polytope of the edge.

If the edge and face are coplanar, the bisector is the plane through the edge orthogonal to the face, and has rational coefficients.

If the edge and face are non-coplanar but parallel, the bisector is a parabolic cylinder: it can be computed in the following way. Let $t$ run along the edge. Let $s$ run across the face in any direction that is not parallel to $t$’s direction, so that $(s,t)$ parametrizes the plane. Construct a point on the bisector by intersecting

- the line orthogonal to the face at $(s,t)$:

- the planar bisector of the face-point at $(s,t)$ and the edge-point at $t$. 

65
Figure 4.8: The generic edge-face bisector is an elliptical cone. The top half can participate in the medial axis. The parametric form generally involves irrational numbers, so the algorithm relies on the implicit form only.

4.4 Summary

The main result of this chapter is the formulation of implicit and parametric representations for all of the bisector surfaces that occur in the polyhedral medial axis. The parametric form always exists, but for a few surfaces, it cannot be expressed in rational numbers. These cases correspond to pairs of governors whose dimension sum to more than two.

When the governor dimensions sum to two or less, the surface can be parametrized by mapping arbitrary points on the governors to unique points on the bisector using the idea of Elber and Kim. When the dimensions sum to three or four, arbitrary points on the governors determine not one but two points on the bisector. These bisector points are found by solving a quadratic equation. Thus the Elber-Kim idea produces a parametrization which is not a rational function—it includes a square root. A rational parametrization can nonetheless be found by stereographic projection, but the coefficients may not be rational.
Chapter 5

Junctions, Seams, and Sheets

In this chapter I describe in detail the representations for the junctions, seams, and sheets of the medial axis, and the algorithms that compute them.

The high-level algorithm described in chapter 3 computes the junctions and seams. The details of this process are described in sections 5.1 and 5.2. The sheets are constructed separately in a post-process, described in section 5.3.

5.1 Junctions

Before tracing a seam, the medial axis algorithm must decide which seams to trace and in which direction. Both of these issues are questions of the local geometry of the medial axis at the seam's starting point. This section describes the local geometry at a junction and presents methods for finding valid seams and their initial directions. Sherbrooke et al. [87] describe a similar technique, although theirs is strictly a numerical scheme for finding valid seams. The cited work also lacks a definition of seam validity which is both formal and operative.

5.1.1 Preliminary examples

As an overview, I shall give motivating examples in two dimensions. Figure 5.1 shows a generic junction, governed by three elements $g_1, g_2, b_1$. Consider a “seam” governed by $g_1$ and $g_2$. The remaining non-governor $b_1$ is called a back element. A seam governed by $g_1, g_2$ is horizontal, and there are two possible seam directions, left and right. Intuitively, the correct direction is the one that moves “towards” $g_1$ and $g_2$ and “away from” $b_1$. However, the seam does not necessarily move “towards” its governors in the usual sense—the distance to $g_1$ does not necessarily decrease. Rather,
the *distance difference* between a governor and a non-governor must decrease. This must hold for each seam-governor/back-element pair at the junction. In geometric language, the correct direction is the one whose tangent vector has a positive dot product with *all* bisector normals pointing toward a governor and away from a back element. In figure 5.1, the two bisector tangents agree that a valid seam governed by $g_1$ and $g_2$ exists and that the initial tangent is to the right.

Figure 5.2 shows a degenerate junction with six governors (all points). Two governors, $g_1$ and $g_2$, are tested to decide whether they govern an incident seam. Again, the two possible seam tangents point to the left and right. Each proposed governor in \{$g_1, g_2$\} is tested against each back element in \{$b_1, b_2, b_3, b_4$\}, resulting in eight tests. Two of these tests are illustrated in the figure. Since one test precludes a seam to the left and another precludes a seam to the right, no incident seam is governed by $g_1$ and $g_2$.

### 5.1.2 Local geometry at a junction point

In this section I give a complete characterization of the local geometry at a junction point in the three-dimensional medial axis.

Consider a junction governed by elements \{$e_1, \ldots, e_k$\} where $k \geq 4$. In the generic case, $k = 4$ and each of the three-element subsets of \{$e_1, e_2, e_3, e_4$\} govern a seam. One problem is the choice of one of the two tangent directions along the seam. Also, a junction may be degenerate. I define a *degenerate junction* as a junction with more than four governors. When $k > 4$, not all three-element subsets actually govern seams, and the non-redundant triples must be identified. These two problems—the

![Figure 5.1: Testing seam validity at a generic junction (in two dimensions). Elements $g_1$ and $g_2$ govern a valid seam, since all bisector normals agree that the seam tangent should point to the right.](image-url)
Figure 5.2: Testing seam validity at a degenerate junction (in two dimensions). Elements $g_1$ and $g_2$ do not govern a valid seam. The bisector normal associated with $g_1, b_1$ rules out a seam pointing to the right, while the bisector normal associated with $g_1, b_1$ rules out a seam pointing to the left.

choice between two search directions and the identification of incident seams—are solved by comparing seam tangent vectors with bisector tangent planes.

It is advantageous to distinguish two types of junction point in the implementation: \textit{internal junctions} and \textit{vertex-junctions}. An internal junction $q$ is characterized by the following conditions:

- $q$ is equidistant from four or more boundary elements:
- this distance is positive, and $q$ lies in the interior of the polyhedron:
- $q$ being equidistant from five or more boundary elements is indicative of a truly degenerate situation:
- the coordinates of $q$ are algebraic numbers of degree at most eight.

A vertex-junction $v$ is characterized by the following conditions:

- $v$ coincides with a vertex of the polyhedron:
- $v$ is equidistant from several boundary elements, but this distance is zero;
- the number of equidistant boundary elements is quite commonly more than four, as it includes all incident faces and (reflex) edges;
- the coordinates of $v$ are rational by assumption.

It can be argued (see Sherbrooke et al. [87]) that the connectivity properties of the medial axis make it feasible to avoid searching out of vertex-junctions at all—they serve as “sinks” for the recursion. Doing so typically removes many degenerate
juncti ons from consideration. However, the problem of degenerate interior junctions remains. Degenerate interior junctions are not rare—they arise dependably in models of artificial objects possessing symmetry.

For any type of junction—interior or vertex, degenerate or non-degenerate—the following predicate decides whether a potential incident seam is a part of the medial axis. Let $E$ be a subset of the set of $k$ junction governors, and let $E'$ be its complement, so $E \cup E' = \{e_1, \ldots, e_k\}$. Let $F$ have at least three elements. For each pair $(e, e')$ where $e \in E$ and $e' \in E'$, let $h(e, e')$ be the tangent plane to the bisector surface of $e$ and $e'$ at the junction point. The tangent plane $h(e, e')$ separates vectors into those pointing toward $e$ and those pointing toward $e'$. Vectors lying in the tangent plane itself are ambiguous.

Now let us consider whether $E$ is the governor set of some seam. Assuming that it is, one may compute the curve's tangent direction at the junction point (for example, by crossing the normal vectors to two bisectors). Let $w$ and $-w$ be the tangent vectors. The seam defined by $E$ and $w$ participates in the medial axis if and only if for every pair $(e, e')$, the vector $w$ points toward $e$. The predicate "points toward $e$ and away from $e'$" is evaluated by comparing $w$ to the normal vector of the bisector tangent plane $h(e, e')$. The same predicate may also be applied to $-w$. Possibly neither $w$ nor $-w$ are true seam tangents. If one is a seam tangent, the other is not.

Checking all possible seams incident to a junction can be expensive if the junction has many governors. There are $\binom{k}{3}$ possible seams incident to our vertex. Ignoring (for the sake of combinatorial analysis) the possibility that a seam may be governed by more than three elements. Checking a seam requires constructing $3(k-3)$ tangent planes. Checking all possible seams, then, requires $\frac{1}{2}k^4 - 3k^3 + \frac{11}{2}k^2 - 3k$ vector-versus-plane checks. For a non-degenerate interior vertex, there is a seam for each $E$, and it is just a matter of choosing an initial tangent $w$ for each one. This requires only four tangent plane checks. For a very degenerate vertex, though, the exhaustive method can be expensive.

Comparison of the vector $w$ to the plane $h$ is performed in exact arithmetic as follows. The seam is represented implicitly in $\mathbb{R}^3$ as the intersection of two surfaces. A seam tangent $w$ may be found by crossing the gradients of the two surfaces. The plane $h$ is the orthogonal complement of the gradient $v$ of the bisector surface $d^2_e(x, y, z) - d^2(x, y, z) = 0$. When the surface is written this way (instead of in the opposite order), the gradient points "towards" $e$, so the predicate on $w$ can be evaluated as the sign of $w \cdot v$. The latter expression is the determinant of the three gradient
vectors. The determinant is expanded symbolically in terms of $x$, $y$, and $z$, and forms a cubic polynomial in those variables. If the rational box containing the junction point lies entirely on one side of the zero set of this cubic polynomial, then the sign can be evaluated at any of the eight corners.

### 5.2 Seams and tracing

The fundamental step in the medial axis algorithm is *seam tracing*. Given a seam curve and one of its junctions as a starting point, the goal is to identify and compute the *terminator*—the junction at the other end of the seam. The terminator's governors are the seam's governors, together with another governor to be identified. Tracing begins by forming a list of *candidate governors*. In the simplest case, all $n$ active boundary elements are candidates. Associated with each candidate are as many as eight points, the solutions to three quadratic equations. So the tracing algorithm may consider up to $8n$ *candidate terminators*. The terminator is the first of these points encountered when moving along the seam curve away from the known endpoint.

Seam tracing is essentially the problem of ordering points along an algebraic curve. I propose three algorithms for seam tracing. Each algorithm uses a different representation of the seam curve. The algorithm is chosen based on the availability of parametric bisectors, which is determined by the type and configuration of the seam governors.

In this section, I enumerate all possible morphologies for bisector surfaces and seam curves for all configurations of governors. Each is suited to a different configuration, and they may be thought of roughly as evaluation of curves in one-dimensional, a two-dimensional, and a three-dimensional space. The ten combinations of boundary elements are listed in table 5.1. with the most general seam curve type for each combination. The analysis method (third column) corresponds to a choice of three algorithms, described in sections 5.2.1–5.2.3 below.

#### 5.2.1 Rational parametric curves

Some seams are rational space curves. The parametric form for the curve is straightforward to derive, using techniques similar to those in section 4.3.3. When this form exists, and its coefficients are rational, the algorithm performs a one-dimensional
<table>
<thead>
<tr>
<th>Governors</th>
<th>Generic seam</th>
<th>Analysis method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point-point-point</td>
<td>Line</td>
<td>5.2.1</td>
</tr>
<tr>
<td>Point-point-line</td>
<td>Parabola</td>
<td>5.2.1</td>
</tr>
<tr>
<td>Point-point-plane</td>
<td>Conic</td>
<td>5.2.1</td>
</tr>
<tr>
<td>Point-line-line</td>
<td>Quartic</td>
<td>5.2.2</td>
</tr>
<tr>
<td>Point-line-plane</td>
<td>Quartic</td>
<td>5.2.2</td>
</tr>
<tr>
<td>Point-plane-plane</td>
<td>Conic</td>
<td>5.2.2</td>
</tr>
<tr>
<td>Line-line-line</td>
<td>Quartic</td>
<td>5.2.2</td>
</tr>
<tr>
<td>Line-line-plane</td>
<td>Quartic</td>
<td>5.2.2</td>
</tr>
<tr>
<td>Line-plane-plane</td>
<td>Conic</td>
<td>5.2.3</td>
</tr>
<tr>
<td>Plane-plane-plane</td>
<td>Line</td>
<td>5.2.3</td>
</tr>
</tbody>
</table>

Table 5.1: Some curves that arise as seams in the medial axis.

search along the curve parameter $t$ for the terminator. Each candidate terminator is represented as the root of a degree-four polynomial in a rational interval. Sorting points along a rational parametric curve is a matter of choosing the candidate interval that is closest, on one side, to the starting junction. The intervals are reduced as necessary until the minimum interval is distinct from the others.

5.2.2 Non-rational curves lying on rational surfaces

In most cases, there is no rational parametrization of the curve with rational coordinates. However, there is sometimes a parametrization in rational numbers of a surface that contains the curve. In such cases, I form an implicit representation of the algebraic curve in the parametric domain of that surface.

In the cases represented by the first eight rows of table 5.1, at least one surface $S_0$ from the three bisector surfaces admits a rational parametrization with rational coefficients. Using the parametric form of $S_0$, the seam curve is pulled back to the $(s,t)$ plane. The curve topology algorithm described below decomposes the curve into monotonic segments.

For each candidate boundary element $e$, the algorithm constructs the bisector surface $S_e$ associated with one of the seam governors. The intersection curve of $S_0$ and $S_e$ is pulled back to the same $(s,t)$ domain, forming a “candidate curve.” The candidate terminators are the intersection points of the candidate curve with the seam curve. Though both curves can be degree four, they still intersect in at most eight points, since these points correspond to the intersections of three quadric surfaces. Using the topological decomposition of the seam curve, it is then a simple matter to
order the candidate terminators and find the first one along the seam curve.

Now I shall describe the curve topology algorithm. This algorithm is described in full in Keyser et al. [61]. The algorithm divides an algebraic plane curve into segments which are monotonic in both \( s \) and \( t \). Such a division allows points on the curve to be easily ordered along the curve. Although general algebraic decomposition algorithms such as Johnstone's [59] can be applied to this problem, the algorithm of Keyser et al. is specialized for relatively low-degree curves, such as quartic seam curves. An example of the output of our algorithm is shown in figure 5.3.

The algorithm makes two key assumptions. First, it assumes that the algebraic plane curve contains no self-intersections or other singularities in the region of interest. Although this significantly restricts the class of curves the algorithm can handle, singular seam curves are rare in practice. This issue is discussed further in chapter 7. Second, it assumes, for simplicity, that all points on the curve are computed distinctly—that is, their bounding rational boxes do not overlap. This second requirement is met by on-demand subdivision of the boxes.

The algorithm begins compute all of the turning points of our algebraic plane curve. \( f(s, t) = 0 \). This is done by finding all common solutions between \( f = 0 \) and \( f_s = 0 \), and between \( f = 0 \) and \( f_t = 0 \). Subscripts denoting partial derivatives. This will isolate all of the turning points in \( s \) and \( t \) as well as the inflection points. Next, the algorithm finds all of the "edge points"—intersections of \( f = 0 \) with the boundaries of a rectangular domain containing the region of interest.

The next step is to determine the connections between the turning and edge points. The curve between any two of these connected points will be monotonic in both \( s \) and \( t \). To make these connections, the algorithm proceeds by recursively subdividing rectangular regions. A region falls into one of the following categories:

- Contains more than one turning point
- Contains one turning point and more than two edge points
- Contains one turning point and exactly two edge points
- Contains no turning points

Treating each of these cases separately, the algorithm recursively subdivides each region until it falls into one of the last two categories. At this point, the curve \( f = 0 \) has been divided into a number of segments monotonic in both \( s \) and \( t \) in the region
of interest. Finally, the algorithm subdivides further until the bounding boxes of the curve segments do not overlap.

![Diagram](image)

Figure 5.3: The output of the curve topology algorithm.

### 5.2.3 Non-rational curves lying on algebraic surfaces

In the last two seam combinations, line-plane-plane and plane-plane-plane, the bisector surfaces (cones and plane-pairs) may not have rational parametric forms with rational coefficients. Rather than performing arithmetic in an extension field, I observe that in these two cases (including degenerate configurations) the seam curve is always a conic or a line. I treat these cases with specialized three-dimensional methods. The candidate junctions are all isolated in \((x, y, z)\)-space using 3-D Sturm sequences. The isolation method is detailed in section 6.3.

For the generic plane-plane-plane case, the equidistant point set corresponds to four lines meeting at the intersection of the planes. The various slopes of these lines are all irrational. The seam is contained in one of the eight “branches” (half-lines) of this curve, and the plane equations of the three governors distinguish points on this branch from points on the others. The ordering among the candidate junctions is determined by their projections onto almost any fixed line. The projections are intervals that may overlap. When overlap leads to an ambiguity about the next junction, the candidates are refined until their projections are distinct. Though almost any line will serve as a projection line, work is minimized when a line is chosen that is nearly parallel to the seam.
For the generic line-plane-plane case, the curve consists of two non-intersecting conics lying in perpendicular planes. The two planes' slopes are generally irrational. One conic is the seam, while the other is irrelevant. Candidate junctions on the irrelevant conic can be culled away by testing against the plane equations of the two governing planes. Ordering points along a conic is not too difficult, since conics are simple curves. I present methods for each of the parabola, hyperbola, and ellipse cases.

If the seam is a parabola, it can be projected onto a line. The candidates are then ordered along the line. A convenient projection line is found in the intersection of the two governor planes. The parabola always projects one-to-one onto this line.

If the seam is a hyperbola, it can still be projected onto the same line. The only difficulty is that the hyperbola has two branches. In fact, candidate junctions on the "wrong" branch are easily rejected by comparing them to the governor planes.

Finally, if the seam is an ellipse, it cannot be projected one-to-one onto any line. Instead, the candidate junctions are ordered by angle about a point in the ellipse's interior. This operation does not actually require computing angles. Rather, a line \( l \) is chosen that passes near the center of the ellipse, roughly orthogonal to the ellipse's plane. Two candidate junctions are ordered by finding a separating plane that contains \( l \). These separating planes are used to order the candidates along the ellipse.

5.3 Sheets

The sheets are not needed during the main seam-tracing process. They are computed after all of the seams and junctions are constructed. This section describes the sheet construction process. Under a simplifying assumption, the running time is \( O(m \log m) \), where \( m \) is a bound on the complexity of the medial axis.

A protosheet is the portion of a bisector surface which participates in the medial axis. In detail, a protosheet is the locus of points \( p \) such that

- \( p \) is equidistant from two governors \( e_1, e_2, \)
- \( p \) lies inside the domain polytopes of \( e_1 \) and \( e_2, \)
- \( p \) is closer to those governors than to any other element \( e \) in whose domain polytope \( p \) lies, and
• $p$ is within the input polyhedron.

A *sheet* is a single connected component of a protosheet. Figure 5.4 shows a protosheet with $\Theta(n^2)$ components.

Finding the sheets based on these definitions would be very difficult. However, once the seams and junctions have been constructed, the protosheets and sheets are easily identified. A sheet exists wherever two seams

• meet at a junction, and

• have two governors in common.

This information—a junction and two incident seams sharing two governors—constitutes a *sheet-corner*. This holds for generic and degenerate seams and junctions.

A single pass through the seam graph finds the sheet-corners, placing them in a data structure $S$ supporting insertion and removal in logarithmic time. The sheet-corners form an implicit graph, consisting of a number of disjoint cycles, one for each sheet. The cycles of corners are found by depth-first search. Let $m$ be the total number of sheets, seams, and junctions in the medial axis. Then the total running time for sheet identification is proportional to the number of sheet-corners, which is $O(m)$, times a factor of $\log m$ for inserting and removing each corner once from $S$.

The running time is increased by two more factors. One is that at each junction, all incident seams are examined. Assume that the number of seams incident to a junction is bounded by $g_1$. For a generic medial axis, $g_1 = 4$. The other factor is that two seam-governor lists must be compared to decide whether they share two governors. So I shall further assume that the number of governors of a seam is bounded by $g_2$. For a generic medial axis, $g_2 = 3$. Letting $g = \max(g_1, g_2)$, the total running time is

$$O(g_1 \cdot g_2 \log g_2 \cdot m \log m) = O(g^2 \log g \cdot m \log m).$$

The sensitivity to $g$ can be reduced if one has an efficient representation for the local geometry at a highly degenerate junction. Essentially, my algorithm assumes that the junction has an unknown structure and always performs an exhaustive search.

The sheet is represented by a parametric surface, together with a loop of algebraic curves in its domain. The curves are found by substituting the parametric surface coordinates into the implicit form of one of the other surfaces of each seam. The preimages of the junction points are found by curve-curve intersection, taking care
Figure 5.4: A polyhedron with a protosheet containing $\Theta(n^2)$ sheets.

to identify the intersection point that matches the original 3-D junction. This representation is exact (except in the cases where the parametric form has irrational coefficients and must be approximated).

A more useful sheet representation is as a trimmed Bézier patch with polygonal, rather than algebraic, trimming curves. This representation is compatible with CAD systems and is used in my system for visualization. Conversion involves tessellating the algebraic curves. I use a "marching squares" algorithm that samples the curve by intersecting it with horizontal and vertical grid lines.
Chapter 6

Efficiency Improvements

The exact tracing algorithm presented in the foregoing chapters has two sources of inefficiency: it is asymptotically suboptimal, and it is based entirely on exact arithmetic. In this chapter, I present techniques that alleviate both sources of inefficiency. Section 6.1 reduces the combinatorial penalty of the tracing algorithm. Two methods are presented that reduce the number of candidate junctions considered during a typical seam trace. Sections 6.2 and 6.3 gives several techniques that significantly reduce the cost of exact computation without sacrificing accuracy. These techniques work by reducing both the number of exact operations and the cost of each operation.

6.1 Alleviating the combinatorial penalty

The tracing algorithm finds a terminating junction for each seam. The number of candidate junctions may be as many as $\Theta(n)$, where $n$ is the number of faces, edges, and vertices of the input polyhedron. In practice, the expense is in computing the candidate junctions, not comparing them. Any technique which rules out candidate elements without examining the corresponding candidate junctions is likely to speed up the tracing algorithm.

Some boundary elements can be rejected before the associated junctions are computed. In this section I describe two methods for rejecting candidate boundary elements. Either method can be used as a pre-process to the tracing algorithm. Neither method reduces the asymptotic worst-case complexity of the algorithm, though both reduce its running time significantly in practice.
6.1.1 Recursive subdivision of space

Several authors have proposed spatial subdivision algorithms for approximating the medial axis or computing it completely. Some of these approaches are discussed in section 2.3. I propose a method that is related to the methods of Milenkovic [69] and Etzion and Rappaport [34].

My algorithm begins by finding a bounding cube for the input polyhedron. The bounding cube is divided recursively into eight sub-cubes. The subdivision continues recursively until one of several termination conditions occurs. The termination conditions, and the reason for preferring a cube to a general bounding box, are described below. A cell is removed if it has fewer than four governors.

Each cube is called a cell. Associated with each cell is a list of boundary elements, called the governors of the cell. The defining property of a cell's governor list is that it contains all boundary elements whose internal Voronoi region intersects the cell. Put another way, the list is a superset of the set of closest boundary elements to all points in the cell. This property is the only invariant maintained during subdivision. The subdivision could consist entirely of cells governed by all boundary elements, in which case nothing is revealed about the medial axis. But for many polyhedra, it is possible to compute a cell division of higher quality, revealing most or all of the medial axis topology.

At the heart of the algorithm are three criteria for removing a governor from a cell's governor list. They can be applied in any order. The order of application does, however, affect the quality of the subdivision. Consider a single cell $C$ with the governor list $G$. Let $g \in G$ be any governor of $C$. Then $g$ can be removed from the list if any of the following conditions hold.

1. Polytope exclusion. The domain polytope of $g$ does not intersect $C$. (The domain polytope is defined in section 4.2.)

2. Distance exclusion. There exists another governor $g' \in G$ such that the center of $C$ is "globally closer" to $g'$ than to $g$, and the difference in distance is greater than the diameter of $C$.

3. Bisector exclusion. There exists another governor $g' \in G$ such that all points in the cell are closer to $g'$ than to $g$, and the cell lies entirely within the domain polytope of $g'$.

I shall discuss each of these criteria in turn. Criteria 1 and 2 were proposed by
Milenkovic [69]. His version of criterion 1 is slightly looser; it is equivalent to intersecting the domain polytope with the circumsphere of the cell. Criterion 3 was proposed by Gottschalk and Culver [43].

**Polytope exclusion.** The domain polytope of \( g \) contains the internal Voronoi region of \( g \). Clearly, if a cell is disjoint from the domain polytope, it is disjoint from the Voronoi region.

**Distance exclusion.** This criterion relies on the global distance functions defined in section 4.1.

The distance exclusion criterion states that a governor \( g \) may be removed if another governor \( g' \) is closer to the cell center by a margin of \(|C|\), the cell diameter. The proof of this criterion is straightforward, relying on two applications of a "generalized triangle inequality" relating the distances between two points and a boundary element:

\[
d_e(p) \leq |p - q| + d_e(q)
\]

This inequality holds when \( d_e \) represents the global distance. It does not hold for the purely algebraic squared distance functions (unless \( p \) and \( q \) are known to lie inside the domain polytope—too strict an assumption). Thus, the criterion states that the governor \( g' \) should be "globally closer" to the cell center, meaning that the global distance functions for \( g \) and \( g' \) are to be compared.

To implement the criterion, it is reasonable to compute the global distance from the cell center to all of the cell governors. The closest governor to the cell center takes the role of \( g' \), and its distance is compared to each other governor \( g \).

The criterion is more likely to succeed when the cell is closely approximated by a concentric sphere with diameter \(|C|\). Cubic cells are therefore preferred. In practice, the benefit of cubic cells is significant enough to warrant beginning the recursion with a cube instead of a tight bounding box.

**Bisector exclusion.** The third criterion is a sharper, but more expensive, variation on the idea in criterion 2. In criterion 2, the cell is approximated by a single point (the center). The error in this approximation is accounted for by introducing a margin of \(|C|\).

Criterion 3 is applied to any two governors \( g_1, g_2 \in G \) as follows. First, the
algorithm computes the difference of the squared algebraic distance functions

\[ D(x, y, z) = d_{g_1}^2(x, y, z) - d_{g_2}^2(x, y, z). \]

Since the algebraic distance is used, the difference is a quadratic function—the implicit form of the bisector of \( g_1 \) and \( g_2 \).

Next, the range of \( D(x, y, z) \) over the cell is computed. If the range contains zero, the test is inconclusive, and neither governor can be eliminated. If the range is entirely negative and the cell lies entirely within the domain polytope of \( g_1 \), then every point in the cell is closer to \( g_1 \) than to \( g_2 \), and \( g_2 \) is removed. The cell is required to lie in \( g_1 \)'s polytope, since any point outside the polytope is essentially at infinite distance from \( g_1 \). Figure 6.1 shows a two-dimensional example.

The argument is symmetric in \( g_1 \) and \( g_2 \). In either case, the closer governor must contain the cell in its domain polytope. The polytope of the farther governor is immaterial.

The range of \( D(x, y, z) \) is computed by evaluating \( D \) at its critical points on the boundary of the cell. The critical points are a finite set defined in terms of the gradient vector \( \nabla D = (\frac{\partial D}{\partial x}, \frac{\partial D}{\partial y}, \frac{\partial D}{\partial z}) \). The critical set is the union of the following four sets.

1. The corners of the cell.
2. The points on a cell edge where \( \nabla D \) is perpendicular to the edge.
3. The points on a cell face where \( \nabla D \) is perpendicular to the face.
4. The points in the cell interior where \( \nabla D \) is the zero vector.

Since the gradient \( \nabla D \) consists of three linear functions in \( x, y, \) and \( z \), the points in categories 2, 3, and 4 can be found by solving one-, two-, and three-dimensional linear systems respectively. Points in category 4 only exist when the bisector surface \( D(x, y, z) = 0 \) is bounded. The only bounded quadric is an ellipsoid, which is not a bisector occurring in the polyhedral medial axis, so category 4 may be ignored. Thus the test requires only symbolic differentiation and the solution of one- and two-dimensional linear systems.

Criterion 3 differs from criterion 2 in that there is no single choice of \( g' \) that is automatically sufficient for removing all other removable governors. For a cell with \( n \) governors, fully applying criterion 3 can require \( O(n^2) \) comparisons, whereas criterion 2 never requires more than \( O(n) \) comparisons. One way to avoid the penalty
associated with criterion 3 is simply to stop after $O(n)$ bisector-box tests. An effective alternative, and the one I use, is to ignore criterion 3 altogether until a cell has only a few governors.

If there are many governors, it is usually wasteful to apply criterion 3 to arbitrary pairs of governors. Criterion 3 is best applied to cells in the later stages of subdivision, when there are only a few governors and the cell is small enough to fit within the polytopes of some of its governors.

**Terminating the recursion.** When a cell's governor list is reduced to three or fewer, the cell is immediately removed. A cell having four governors is subdivided until it reaches a minimum tree depth $l_1$. A cell having five or more governors is subdivided to a deeper level $l_2$ in the hope that it will break up into cells with fewer governors.

The result of the spatial subdivision algorithm is a collection of governor lists. If the medial axis is generic and the subdivision proceeds far enough, each list has four governors. A four-governor list may correspond to a single junction, many junctions, or no junctions at all. Even when it does correspond to a single junction, the junction may not be contained in the cell or in any nearby cell. Figure 6.2 shows that a single generic junction may be represented by many disconnected cells.

The connection between junctions and junction-like cells is loose. The benefit of the subdivision algorithm is that it often rules out a vast number of boundary element quadruples that cannot govern any junction.

**Combining the criteria.** The criteria may be applied in any combination and in any order. The result is always a valid spatial subdivision that can be used to reduce the number of candidates during seam tracing. Naturally, some subdivisions are better than others, and the order in which the exclusion criteria are applied makes a significant difference.

Surprisingly, it is best *not* to apply the polytope exclusion rule until a cell has very few governors. I find that if the polytope rule is applied to every cell at every stage, many cells outside the polyhedron are ultimately kept as possible junctions. If instead the polytope rule is applied only after the number of cell governors has been reduced to four, the polytope rule tends to eliminate these exterior cells. The reason is that the subdivision algorithm does not know the interior of the polyhedron from exterior. In the interest of removing as many exterior cells as possible, it is beneficial
Figure 6.1: Criterion 3 cannot use $g_1$ to eliminate $g_2$ from the governor list of $C$ unless the cell lies entirely within the domain polytope of $g_1$. Every point in the cell is closer (in algebraic distance) to $g_1$ than to $g_2$. Yet the cell contains part of the Voronoi region of $g_2$, so $g_2$ must not be eliminated.

Figure 6.2: The cells corresponding to a junction can be many and disconnected.
to delay polytope exclusion. Figure 6.3 shows a 2-D example.

**Numerical error.** The subdivision algorithm can be implemented in exact arithmetic, since it requires only rational arithmetic. However, it is possible to implement the algorithm safely in floating-point arithmetic. Tolerances are introduced to outweigh the effects of roundoff error. For instance, in the bisector exclusion rule, the range of the quadric function is computed in floating-point and then expanded slightly. An interesting topic for future work is to compute a lower bound on each of the tolerances based on the input precision.

**Comparison to previous subdivision algorithms.** My subdivision method is similar to two previous approaches. Milenkovic's [69] is the most closely related. The main difference is that his tracing algorithm takes small steps through space, examining the cells that the seam passes through to check for possible junctions. Since my tracing algorithm does not take small steps through space, it does not need to use the locations of the cells; the subdivision is used only for its combinatorial information.

The subdivision method of Etzion and Rappaport [34] is different in many ways. The most significant is that their method retains cells with three or more governors. Cells with three governors are used in approximating the seams. (In fact, a postprocess traces out the seams from cell to cell, suggesting that at a high level of abstraction, the algorithm has a close relationship to Milenkovic's.) Keeping the cells with three governors may increase the memory requirements significantly.

Etzion and Rappaport also propose a different variation on the bisector exclusion criterion. They observe that a bisector intersects a cell if and only if it intersects the cell boundary. Their algorithm finds a point on the bisector-cell boundary intersection curve. The operation requires solving a quadratic equation in one variable. The version I present is preferable in that it requires solving only linear equations.

### 6.1.2 Polytope pairs

An alternate preprocess for speeding up the search algorithm is based on the domain polytopes defined in chapter 4. For each each pair of governors, the preprocess tests their polytopes for mutual intersection inside the bounding box of the polyhedron. If the polytopes do not intersect, the governors cannot be Voronoi neighbors. If the
Figure 6.3: If the polytope exclusion rule is applied too early, cells outside the polyhedron are unnecessarily kept. (a) The two boundary elements shown are removed from the cell's governor list by the polytope exclusion rule. (b) The cell is equidistant from three other boundary elements. As a result, the cell is kept. It could have been removed if the two closer elements had first been used to eliminate the three farther elements.

Polytopes do intersect, each governor is put on the other's sorted list of potential neighbors.

During the main algorithm, the lists are used as follows. When the seam governed by $e_1, e_2, e_3$ is traced, the desired fourth governor $e_4$ must be a Voronoi neighbor of all three seam governors. The tracing algorithm then limits its attention to the governors which are on all three lists.

The preprocessing involves $\binom{n}{4}$ steps, each of which performs a polytope intersection and a list insertion. The polytope intersection usually takes constant time in practice, since the polytopes are typically defined by fewer than ten planes. The list insertion takes $O(\log n)$ time. The overall preprocessing running time is thus $O(n^2 \log n)$. The memory required is $O(n^2)$ pointers. The asymptotic running time of the complete medial axis algorithm is not increased, since the list-intersection step takes at worst $O(n)$ time for each seam traced—the same amount of time that was already committed to examining all the other boundary elements.

The preprocessing time of $O(n^2 \log n)$ is not significantly different from the complexity of the medial axis algorithm, which takes at least $\Omega(n^2)$ time without such a preprocessing heuristic. Moreover, the constants in the preprocessing time are much smaller than those in the medial axis implementation, since the polytope intersection step involves only linear geometric computation (signs of $4 \times 4$ determinants).

The polytope pair test is often not as effective as spatial subdivision. The main
advantage to the polytope pair test is that its behavior is predictable in terms of time and resource consumption.

6.2 Reducing the number of kernel operations

In my system, roots of systems are computed only to the precision necessary. Root representations are exact, but implicit, and the geometric algorithms usually pay attention only to the rational box containing the root. At all times, the rational box is shrunk only as much as necessary to answer the query. This is a form of "lazy evaluation" of solutions to polynomial systems.

The efficiency of floating-point arithmetic can be leveraged against the problem of finding roots to systems. While isolating roots is based on bisection, an initial estimate to the root can be based on a floating-point approximation. Three-dimensional root finding proceeds in three steps:

1. Using 3-D Sturm sequences, count the number of roots in a bounding box.

2. Using multivariate Newton-Raphson iteration, locate the roots and construct small, disjoint rational boxes around them.

3. Using 3-D Sturm sequences, verify that each box contains a single root.

Usually the Newton-Raphson method finds the roots to very high precision in very little time. In such a case, the rational box is very small and further refining will usually not be necessary. In other cases, the floating-point method may not be able to distinguish closely-spaced roots. When this happens, the Sturm sequence computation will detect that a box contains more than one root. The roots are then isolated by bisection.

The fundamental 2-D kernel operation is isolating and refining curve-curve intersections. The bivariate Sturm sequence is an appropriate tool, but a more efficient method has been developed for the MAPC library [61]. This method, based on Sylvester resultants and univariate root solving, consistently outperforms bivariate Sturm. Unfortunately, the method does not generalize easily to three dimensions.

6.3 Speeding up the kernel operations

The implementation spends most of its time in the 3-D Sturm sequence routine. A multivariate Sturm sequence computation is executed in two main stages. In the
elimination stage, a system of four equations in four unknowns is reduced to a single polynomial, the volume function. The sequence evaluation stage counts the number of negative real roots of the volume function. The formulation is due to Milne [71]. The performance of the system depends primarily on the efficiency of 3-D Sturm sequences.

6.3.1 Elimination

The elimination stage turns a multivariate root-counting problem into a univariate root-counting problem. To solve a system of three equations in \(x, y,\) and \(z,\) a fourth variable \(u\) and a fourth equation \(u - (x - \alpha)(y - \beta)(z - \gamma)\) are adjoined. The variables \(\alpha, \beta, \gamma\) are constants, the coordinates of a corner of the rational box. The variables \(x, y,\) and \(z\) are eliminated from the system, leaving a single polynomial in \(u,\) the volume function \(V(u)\). The method is detailed in appendix A.

The elimination is accomplished with Macaulay's resultant [67], which represents the resultant as the quotient of the determinant of a matrix \(M\) and the determinant of a sub-matrix \(M'\). When the system consists of three quadrics in \(x, y,\) and \(z,\) as it does in the worst case for the medial axis problem. \(M\) is an 84 \(\times\) 84 matrix whose entries are constants and linear polynomials in the variable \(u.\)

The resultant could also be applied directly to the three quadric equations in \(x, y, z\) (rather than the three quadrics and a fourth cubic equation in \(x, y, z, u\)). The Macaulay resultant can eliminate, for example, \(y\) and \(z,\) leaving a degree-8 polynomial in \(x\) whose roots are the \(x\)-coordinates of the intersection points. Two more resultants give the \(y\)- and \(z\)-coordinates. This technique is explored in Chionh et al. [17]. The more expensive multivariate Sturm sequence approach has two main advantages: roots which share a coordinate are separated, and roots with complex coordinates are ignored.

The system can be arranged so that the Macaulay denominator \(M'\) depends only on the coefficients of the system, and does not contain \(\alpha, \beta, \gamma,\) or \(u.\) The denominator \(\text{det}(M')\) is then a constant, so no polynomial division is necessary. One only needs to check that the denominator matrix is nonsingular. The denominator is also independent of the point \((\alpha, \beta, \gamma)\) in space at which the Sturm sequence is evaluated, so it needs to be checked only once per system.
The Macaulay numerator is arranged into block form:

\[
M = \begin{pmatrix}
A & B \\
C(\alpha, \beta, \gamma) & D(\alpha, \beta, \gamma) + uI
\end{pmatrix}
\]

where \(A\) and \(B\) contain coefficients of the system, and \(C\) and \(D\) depend on the point \((\alpha, \beta, \gamma)\). \(A\) is a 76 \(\times\) 76 matrix and \(D\) is 8 \(\times\) 8. By performing block Gaussian elimination on this matrix, the bulk of the work of taking the determinant of the Macaulay numerator is performed before \(\alpha, \beta, \gamma\) are known. The precomputation consists of finding the \(LU\)-decomposition of \(A\) and computing \(\text{det}(A)\) and \(A^{-1}B\). Then, for each \((\alpha, \beta, \gamma)\), the method computes \(H = D - C \cdot (A^{-1}B)\) and the 8 \(\times\) 8 symbolic determinant \(\text{det}(H + uI)\). An exact method for computing the polynomial \(\text{det}(H + uI)\) is to substitute nine different numeric values for \(u\), compute the nine determinants in exact arithmetic, and solve for the polynomial coefficients with Vandermonde interpolation. The determinant \(\text{det}(M)\) is then given by \(\text{det}(A) \cdot \text{det}(H + uI)\). The volume function is then \(V(u) = \text{det}(A) \cdot \text{det}(H + uI) / \text{det}(M')\). Constant factors are irrelevant for sequence evaluation, though, so \(\text{det}(H + uI)\) is used as the volume function, after \(A\) and \(M'\) are checked for singularity.

### 6.3.2 Sequence evaluation

Sturm sequences can be evaluated by constructing the actual sequence of polynomials \([f(u), f'(u), f_3(u), \ldots]\) where \(f_i = \text{rem}(f_{i-1}, f_{i-2})\), and counting the sign changes in the sequence of constant terms. Here \(\text{rem}(f, g)\) gives the remainder on polynomial division of \(f\) by \(g\). This evaluation algorithm is similar to Euclid's algorithm for computing the GCD of two polynomials. For low-degree polynomials, such as the degree-8 volume function \(V(u)\) arising in isolating roots of three quadric surfaces, this approach can be practical. An implementation based on adaptive-precision arithmetic has been developed as part of the PRECISE library [63].

For higher-degree polynomials, the subresultant polynomial remainder sequence algorithm [11] can improve efficiency by avoiding the exponential coefficient growth in the polynomial \(\text{GCD}\) operations. The subresultant algorithm computes the coefficients of the sequence's polynomials as determinants of matrices. For root counting, one needs only the signs of the constant terms of the polynomials in the sequence. The problem is thus reduced to computing the signs of determinants of matrices with rational entries, or equivalently, with integer entries. The determinant sign evaluation
problem is discussed in chapter 8.

6.3.3 Using numerical eigenvalues

The most expensive part of sequence evaluation is computing the symbolic determinant $V(u) = \det(H + ul)$. Though it is a fairly small matrix, the entries usually have bit lengths many times larger than the bit lengths in the input. It is often possible to circumvent this symbolic determinant by computing the numerical eigenvalues of a floating-point approximation to the matrix $-H$. The resulting technique is a floating-point filter for computing the signs of the roots of the volume function without computing the volume function's coefficients explicitly.

The expensive degree-8 univariate polynomial $V(u)$ is used only to evaluate the Sturm sequence of $V(u)$ at $u = 0$. The evaluation produces a small integer $\text{var}_V(0)$, the number of variations in the Sturm sequence of $V(u)$ at 0. (The variation operator $\text{var}_V$ is defined in the appendix, section A.2.) The number of variations is equal to the number of negative roots of $V(u)$ plus the number of complex conjugate root pairs. Since $V(u)$ is given by the matrix determinant $\det(H + ul)$, it is equal to the characteristic polynomial of the matrix $-H$. Therefore the roots of $V(u)$ are the eigenvalues of $-H$. The eigenvalues can be computed quite efficiently by a numerical routine, such as the DGEEV routine of LAPACK [24]. The more subtle issue is the reliability of this operation.

The accuracy of numerical eigenvalues is an important and well-studied problem. For a treatment, see Golub and van Loan [42]. The Gershgorin Circle Theorem is one of the main results. Informally, it states that if a matrix is perturbed slightly, the eigenvalues move within certain disks in the complex plane. The theorem applies to eigenvalue algorithms such as the one implemented in the LAPACK library. LAPACK’s algorithm applies a series of similarity transformations to the matrix to convert it to a “upper quasi-triangular” matrix, that is, with $1 \times 1$ and $2 \times 2$ blocks on the diagonal. The diagonal elements $\tilde{\lambda}_i$ of the transformed matrix are the eigenvalues. The net perturbation by the similarity transformations is small, and Gershgorin’s theorem can be applied to the results of LAPACK’s algorithm. The result is that the true eigenvalues $\lambda_i$ lie within the union of certain disks centered at the computed eigenvalues $\tilde{\lambda}_i$. Further, if $k$ of the disks form a connected domain that is isolated from the other disks, then the domain contains exactly $k$ true eigenvalues. The details of this result are given in Demmel et al. [24].

The circle theorem gives a sufficient condition for the reliability of the numerical
eigenvalues. The number of negative real eigenvalues of $-H$ is correct unless either of the following conditions occurs:

- A disk contains the complex number 0.
- Two or more disks overlap, and any of them touches the real line.

The conditions are illustrated in figure 6.4.

Recall that each real root of $V'(u)$ corresponds to the signed volume of the 3-D box with one corner at a root of the 3-D system and the opposite corner at the point of evaluation $(\alpha, \beta, \gamma)$ (see appendix A). This implies that whenever the evaluation point is at a small distance $\delta$ from a root, the volume function $V'(u)$ has a root whose magnitude is on the order of $\delta^3$. Such a root can be expected to produce a Gershgorin disk that contains the complex number 0.

In fact, this situation occurs frequently in practice. Usually, the Newton iteration described in section 6.2 produces a high-precision approximation to each 3-D root. A small rational box, say with side $\delta$, is drawn about the approximate root, and the 3-D Sturm sequence is evaluated at each of the eight corners. Every corner is close to a root of the system, so for all eight evaluations, the volume function has a tiny root on the order of $\delta^3$, causing the eigenvalue filter to fail. The sign variation in this tiny root is exactly what the 3-D Sturm sequence is attempting to measure.

The problem can be circumvented at some cost. Let $\lambda_1$ be a real eigenvalue whose sign is in doubt. Suppose that the rest of the eigenvalues $\lambda_2, \ldots, \lambda_n$ are each known to be positive, negative, or imaginary. Since

$$\text{sign}(\lambda_1) \cdot \text{sign}(\lambda_2) \cdots \text{sign}(\lambda_n) = \text{sign}(\det(-H)).$$

the sign of the tiny root can be derived from the sign of the determinant of $-H$. Finding $\text{sign}(\det(-H))$ is expensive, but the exact alternative involves computing the nine exact determinants $\{\det(H + uI) : u = 0, \ldots, 8\}$ and exactly solving a $9 \times 9$ Vandermonde system, so the filter is still a win. This case—one tiny real eigenvalue, the others well-separated—is the common one for this algorithm. The filter may fail if the 3-D system has roots that are close together. Also, it may fail if a root lies very close to the evaluation point in one or more coordinates. that is, close to one or more of the planes $x = \alpha, y = \beta, z = \gamma$.

In summary, the roots of $V'(u)$ are the eigenvalues of $-H$. The construction of $V(u)$ can be reliably circumvented when its roots are well-separated, in the complex
Figure 6.4: The Gershgorin disks in the complex plane can be used to gauge the reliability of eigenvalue signs. (a) Even with some types of overlap, the signs can be reliably determined. This example has one negative eigenvalue, one positive eigenvalue, and three complex conjugate pairs. (b) The eigenvalue signs are ambiguous. This example illustrates all four basic types of sign ambiguity.

plane, from each other and from 0. The separation condition can be evaluated reliably by examining the eigenvalues of $-H$ and their Gershgorin disks. In a particularly common case, all of the disks are well-separated except for one near 0. In this case, only the sign of the constant term of $V'(u)$, or equivalently the determinant of $-H$, needs to be computed exactly.
Chapter 7
Degeneracies

The medial axis algorithm presented in the foregoing chapters assumes a non-degenerate medial axis. For example, when ordering candidate junctions along a seam curve, it is assumed that there is not a tie for the first junction point along the curve. This chapter describes the degeneracies that arise in the medial axis problem. These degeneracies are of only a few types, and many can be detected and dealt with explicitly during the course of the algorithm. Only three forms of degeneracy pose a problem to the medial axis seam-tracing algorithm:

- the junction with more than four governors,
- the seam with more than three governors, and
- the degree-four seam with a singular point.

The first two forms are common in solids possessing symmetry. They are illustrated in figure 7.1. Accounting for the excess governors requires minor provisions in the seam tracing algorithm. I describe these provisions in this chapter. I also illustrate and discuss the third form of degeneracy, the singular seam, which is an open problem. Finally I mention the types of degeneracy which arise incidentally during the algorithm and are not indicative of a degenerate medial axis.

7.1 The degenerate junction

A junction is said to be degenerate when it is equidistant from five or more boundary elements. Our algorithm detects degenerate junction points by a three-stage process.
Figure 7.1: The two common forms of medial axis degeneracy. (a) A cube has one degenerate junction. The junction has six governors instead of four. (b) A rectangular box with dimensions (1,1,2) has two degenerate junctions (five governors instead of four) and one degenerate seam (four governors instead of three).

1. During the searching step, the box containing the starting point overlaps with the box containing a candidate end point, even after the intervals or boxes have been reduced to a reasonably small size. This signals that the junction may be degenerate.

2. Check whether the four bisector surfaces involved actually meet at some point. using the Dixon resultant [27].

3. If so, either further refine the root to the precision recommended by Canny’s gap theorem [13], or check for a root of the system

$$\{f_1^2 + f_2^2 + f_3^2 + f_4^2, f_1, f_2\}$$

inside the overlap of the boxes.

The first two steps may be regarded as filters for avoiding step 3 where possible. If the junction is found to be degenerate, the new governor is adjoined to the list of junction governors, and the collection of incident seams is re-evaluated as described in section 5.1.

### 7.2 The degenerate seam

A seam is degenerate when it is governed by four or more boundary elements. Our algorithm does not need to do extra work to discover degenerate seams. Such a
seam is analyzed using three of its governors, while a fourth governor is necessarily discovered during the search for the ending junction. While attempting to isolate a zero-dimensional solution to this system, the algorithm encounters a Sturm volume function that is identically zero, meaning that the solutions set is infinite. When this occurs, the extra governor is simply added to the list of seam governors.

Another form of seam degeneracy is the singular seam curve. A seam curve of degree higher than two can have a singular point, that is, a point at which there is no unique seam tangent. An example is shown in figure 7.2. The curve is the set of points equidistant from three lines. The lines, each defined by a pair of points, are

\[ l_1 : \{(1.0.0),(1.0,1)\} \]
\[ l_2 : \{(0.1,0),(1.1,0)\} \]
\[ l_3 : \{(1.1,0),(0.1,1)\} \]

The three lines are pairwise skew, so each bisector is a hyperbolic paraboloid. The curve is the union of two parabolas intersecting tangentially. The singular point does not lie on any of the lines. This example is due to Little [65].

Singular seam curves pose a problem for any tracing-based algorithm. All such algorithms assume that each seam curve has two ends, so that given one end, the other can be computed. If a tracing algorithm were to come across the seam curve given in figure 7.2, it may need to find the singular point and find both parabolas. The singular point occurs where the seam tangent is the zero vector. The seam tangent can be defined as the cross product of two bisector normals. The singular point can be defined by the simultaneous vanishing of the three coordinates of the seam tangent vector, each of which is a quadratic equation in \(x, y, z\). So in principle, the singular point can be found by intersecting three quadric surfaces. This does not, however, solve the problem of “tracing” the singular seam curve. A general solution may involve an analysis of the singularity by algebraic techniques such as the “blowup” transformation [84].

### 7.3 Incidental degeneracies

Though the medial axis is inherently degenerate in only a few ways, the algorithm itself introduces further degeneracies. Such incidental degeneracies are the result of arbitrary choices made in the course of the algorithm. In practice, these can often be
Figure 7.2: A degree-four seam curve with a singular point. The curve is the union of two parabolas intersecting tangentially.

alleviated by introducing a random variable into the arbitrary choice. This section identifies the most important incidental degeneracies in the medial axis algorithm.

**Resultants.** Macaulay's formulation of the resultant is only well-defined for a generic choice of coefficients. Specifically, there is a small subset of polynomial systems for which the Macaulay formulation takes the form $0/0$. A random affine transformation on the variables $x, y, z$ usually removes this degeneracy. For this reason, my implementation of 3-D Sturm sequences manipulates not rectangular boxes, but parallelepipeds. Even with such a transformation, some systems may still fail due to special or missing coefficients. In medial axis computation, this can usually be alleviated by switching to an equivalent problem: simply pick different surfaces. A junction is defined by the intersection of three surfaces, but for any particular junction, there are always at least six surfaces to choose from (the pairwise bisectors of the four governors). However, to be totally rid of this problem, a symbolic perturbation may be necessary.

**Parametric surfaces.** The parametrization of the sheets is arbitrary. Once the parametrization is chosen, degeneracies can occur in the plane which pose a challenge to the curve topology algorithm. For instance, a turning point of the seam curve may lie at the corner of the $(s, t)$-domain. The implementation must be prepared to detect such problems, and expand or recompute the parametric domain as necessary.
Chapter 8

Determinant Sign Evaluation

Many geometric predicates can be expressed as the sign of the determinant of a matrix. In a convex hull algorithm, for instance, all of the algorithm's decisions can be encoded in terms of determinant signs. Determinant sign evaluation is also important in non-linear geometric computation, since the subresultant algorithm expresses the coefficients of the Sturm sequence of a polynomial as matrix determinants.

Previous work on fast, accurate determinant sign evaluation has focused on small matrices with small entries. The subresultant algorithm, on the other hand, often requires the signs of determinants of large matrices with large entries. In this chapter I give a hybrid method for efficiently and accurately finding the sign of the determinant of a moderate-sized matrix (up to order 55) with exact integer entries of more than 64 bits. Since the results are useful outside the context of medial axis computation, this chapter addresses the problem in the more general context of non-linear geometric computation.

This chapter represents joint work with Shankar Krishnan, John Keyser, and Dinesh Manocha.

8.1 Introduction

Determinant sign evaluation is an important problem for geometric applications. For some applications, a common technique for alleviating robustness problems is to express algorithmic decisions as determinant signs and then focus on a fast, exact implementation of this operation. Researchers have developed many specialized techniques for computing determinant signs of small matrices, often focusing on $6 \times 6$ matrices and smaller. A review of previous work on determinant sign evaluation is presented in section 2.5.3.
In non-linear geometric computation, robustness requires effective handling of algebraic numbers. The key tool is often the Sturm sequence of a polynomial. One of the most effective ways to evaluate the Sturm sequence is with the subresultant algorithm, which expresses the terms of the sequence as matrix determinants. Often only the signs are needed. However, the matrix size grows with the degree of the polynomials. Evaluating a degree-\( n \) polynomial's Sturm sequence using the subresultant algorithm requires a determinant of order \( 2n - 1 \) (and also the determinants of smaller matrices). For example, a three-dimensional algorithm requiring intersections of quadric surfaces involves algebraic numbers of degree eight, leading to subresultant matrices of order 15. Polynomials of much higher degree may be used as well. Of the known specialized methods for fast, exact determinant signs, most are ineffective for matrices of this size.

A further difficulty is the bit length of the matrix entries. In some linear problems, the matrix entries are simple formulae in the input data. But in non-linear computation, the matrix entries tend to have a bit length significantly greater than the input bitlength, simply because the matrix entries are further removed from the input data. Specialized determinant sign algorithms often have a bit length restriction which is not satisfied in practice.

**Chapter overview.** In section 8.2, I describe several applications for determinant sign evaluation. The examples come from non-linear geometric computation in two and three dimensions.

Sections 8.3 through 8.5 present determinant sign algorithms. In section 8.3, I review the technique of modular arithmetic, including recent advances. In section 8.4, I present a new floating-point filter based on singular value decomposition. In section 8.5, I present a determinant sign method based on multiprecision interval arithmetic.

In section 8.6, I examine the performance of several algorithms on a variety of matrices encountered in non-linear geometric computation. In section 8.7, I weigh the strengths and weaknesses of the different methods and demonstrate that the methods may be combined into a powerful, efficient multistage method. I conclude in section 8.8.
8.2 Determinant signs in non-linear computation

The Sturm sequence is a basic tool for isolating the real roots of a univariate polynomial. An efficient means of computing the Sturm sequence is by the subresultant algorithm of Collins [19], which expresses the coefficients of the sequence as matrix determinants. Univariate Sturm sequences and the subresultant algorithm are summarized in appendix A.

For systems of polynomial equations, my approach is to construct a univariate polynomial whose roots are related to the roots of the system. Several techniques can reduce the dimension of the system. In this section, I describe three multivariate algebraic problems with geometric applications, and I give a method for reducing each problem to a univariate root isolation problem amenable to the subresultant algorithm.

8.2.1 Curve-line intersection

In the \((s,t)\) plane, a line can be parametrized by \(u \text{ as } (a + bu, c + du)\). If the line is intersected with the algebraic curve \(f(s,t) = 0\), the intersection points correspond to the roots of \(r(u) = f(a + bu, c + du)\). In this case, the substitution operation reduces a two-dimensional problem to a univariate problem.

This approach for curve-line intersection is used in the curve-curve intersection algorithm presented in Keyser et al. [61]. Each of two curves \(f(s,t) = 0\), \(g(s,t) = 0\) is intersected with each of the four walls of an axis-aligned box. The algorithm examines the configuration of the curve-line intersections and infers the presence or absence of a root.

8.2.2 Curve-curve root projection

The resultant of a pair of polynomials is a projection operation. Given two curves \(f(s,t) = 0\), \(g(s,t) = 0\), the Sylvester resultant with respect to \(s\) is a univariate polynomial \(r(s) = \text{Res}_s(f,g)\) whose roots are the \(s\)-coordinates of the intersections of \(f\) and \(g\). The roots are effectively projected onto the \(s\)-axis. This operation may also be viewed as eliminating the variable \(t\) from the system. "Elimination" here carries the same meaning as it does in Gaussian elimination.

Projection is used in the curve-curve intersection algorithm (Keyser et al. [61]). The \(x\)-resultant and the \(y\)-resultant are used together to compute possible locations.
Figure 8.1: Curve-line intersection. The curve $f(s, t) = 0$ is a torus-ellipsoid intersection curve, pulled back to the domain of one of the surfaces. The line is $t = \text{const.}$

Figure 8.2: Curve-curve root projection. The closed curve $f(s, t) = 0$ is a torus-torus intersection curve, pulled back to the domain of one of the surfaces. The other curve is $\frac{\partial f}{\partial s} = 0$. The roots are projected onto the $s$-axis by taking the resultant with respect to $t$. 

100
for roots.

8.2.3 Surface-surface-surface intersection

To intersect three algebraic surfaces \( f(x, y, z) = 0 \), \( g(x, y, z) = 0 \), \( h(x, y, z) = 0 \), one may use multivariate Sturm sequences to isolate the roots of the system. Multivariate Sturm theory was developed by Pedersen [78] and Milne [71]. Milne's method introduces a fourth variable \( u \) and a carefully-chosen fourth equation in \( x, y, z \), and \( u \). The variables \( x, y, z \) are then eliminated using a higher-order resultant. The remaining equation is a univariate polynomial \( V(u) \), called the volume function, whose roots are in one-to-one correspondence with the roots of the three-dimensional system. In effect, the system is promoted to four dimensions, and then the roots are projected in such a way as to ensure that distinct roots have distinct projections.

8.3 Modular arithmetic

Modular arithmetic has two main advantages over arbitrary-length integer arithmetic for determinant computation. First, the modular computation can be carried out in a fixed-precision arithmetic, chosen for its efficiency on the target platform. Second, the intermediate coefficients in Gaussian elimination grow exponentially in length, typically exceeding Hadamard's bound on the ultimate result. As a consequence, a "bigint" implementation suffers from coefficient explosion. Modular arithmetic avoids this explosion. In this section I briefly review several methods for computing matrix determinants in modular arithmetic.

The modular arithmetic approach computes \( \det(A) \) in modular arithmetic over a collection of relatively prime numbers \( \{p_i\}_{i=1}^m \). The determinant residues \( d_i = \det(A) \mod p_i \) can be used to reconstruct the determinant residue over the much larger finite field of order \( P = \prod p_i \). That is, \( D = \det(A) \mod P \) is computed from the \( \{d_i\} \). If \( D \) is then reinterpreted as an integer in the interval \( (-P/2, P/2) \) and the actual determinant \( \det(A) \) is known to be within the same range, then \( D = \det(A) \).

The \( \{p_i\} \) are chosen so that arithmetic modulo \( p_i \) can be implemented efficiently in a hardware-supported data type. For instance, a machine with fast IEEE double-precision floating point operations can use that data type for 53-bit integer arithmetic. In this system, one may use a modulus as large as \( 2^{27} \) while avoiding lost precision on multiplication. One may choose \( p_1 \) to be the largest prime less than \( 2^{27} \), \( p_2 \) the
largest prime less than \( p_1 \), and so on. Further, the \( \{ p_i \} \) must be chosen so that 
\[
\prod p_i > |\text{det}(A)|.
\]
This requires an a priori bound on the size of the determinant. Hadamard’s bound states that
\[
|\text{det}(A)| \leq \prod_{i=1}^n \left( \sum_{j=1}^n A_{ij}^2 \right)^{1/2}
\]
so one uses just as many primes as is necessary so that their product exceeds twice this bound. See Knuth [62].

The usual choice for computing the modular determinant is fraction-free Gaussian elimination, although Wiedemann’s algorithm [96] may be more efficient for large, sparse matrices. For the reconstruction step, the implementor has several options. An algorithm attributed to Lagrange appears to be the fastest in the typical case. An iterative algorithm attributed to Newton has the advantage that it allows early termination. The determinant may have significantly fewer bits than Hadamard’s bound predicts. If \( |\text{det}(A)| < \prod_{i=1}^{m'} p_i \) with \( m' < m \), then only \( d_1, \ldots, d_{m'+1} \) need be computed. The case of a small determinant can be detected, albeit without absolute certainty, during reconstruction by noticing that \( d_1, \ldots, d_{m'} \) yield the same answer as \( d_1, \ldots, d_{m'+1} \). If this happens, reconstruction can be halted early. There remains the possibility that the answer is incorrect, with a probability of \( 1/p_{m'+1} \). Newton’s algorithm is more efficient than Lagrange’s when the determinant magnitude is significantly less than Hadamard’s bound. See Knuth [62] for the details of these reconstruction algorithms. A third reconstruction algorithm is given by Brönnimann et al. [9], based on Lagrange’s algorithm. Theirs is different from all other known methods in that it requires only single-precision operations (assuming single-precision input), yet computes the sign correctly.

8.4 A floating-point filter

Modular techniques are an effective alternative to arbitrary-precision arithmetic for determinant sign computation. Yet a modular routine can be slower than a purely floating-point routine by orders of magnitude. Further, the floating-point method may be right most of the time. It is sometimes possible to combine two such routines into a single system that has the reliability of the exact method and efficiency approaching that of the floating-point method. The key tool is a floating-point filter: a floating-point algorithm together with a conservative criterion for deciding whether
the algorithm's result is reliable. Only when a result is found to be unreliable does the system fall back on the more expensive exact method.

I propose a floating-point determinant sign filter based on the singular value decomposition of a matrix. Like most useful floating-point matrix decompositions, SVD leads to an estimate of the determinant and its sign. However, the SVD has two additional advantages. First, it can be computed by a backward-stable algorithm (Golub and van Loan [42]). The error in the decomposition can be measured without assuming that the maximum error occurs at each arithmetic operation. Second, a tight bound is known on the error in the smallest singular value, which is the distance to the nearest singular matrix. The SVD assists in the computation of this bound. These properties allow one to use the SVD both to find the determinant sign and to decide whether the computed sign is reliable. As I will show, the computed sign is reliable whenever the matrix is well-conditioned—a criterion that is decided by a threshold on the condition number $\kappa$ of the matrix.

The singular value decomposition of a square matrix $A_{n \times n}$ is a factorization $A = P\Sigma Q$, where $P$ and $Q$ are orthogonal, and $\Sigma$ is a diagonal matrix with positive entries $\sigma_i$. By convention, the $\sigma_i$ are placed in nonincreasing order $\sigma_1 \geq \cdots \geq \sigma_n$. The largest singular value $\sigma_1$ is equal to the Euclidean matrix norm $\|A\|_2$. The smallest singular value $\sigma_n$ is equal to the Euclidean distance to the nearest singular matrix to $A$. The SVD has many useful properties and many applications. For details and an algorithm based on QR iteration, see Golub and van Loan [42].

The SVD can be computed in floating-point arithmetic with the following backward error bound. The computed matrices $P\Sigma Q$ are the exact singular value decomposition of a perturbed matrix $A + E_{SV,D}$, where

$$\|E_{SV,D}\|_2 \leq f_1(n) \cdot \varepsilon \cdot \|A\|_2. \quad (8.1)$$

The value $\varepsilon$ is machine epsilon. The function $f_1(n)$ is less than $100n^3$ (Demmel [25]) but experience suggests that $f_1(n) = n$ is a safe assumption for the problem at hand: the appropriate value for $f_1(n)$ is discussed further at the end of this section.

Demmel and Kahan [26] improve the traditional SVD algorithm. Their modified QR algorithm is forward stable as well as backward stable. The authors prove fairly tight error bounds on all of the singular values computed by their algorithm, whereas the traditional QR algorithm computes the smaller singular values with less accuracy. In the modified QR algorithm, the relative error in each singular value is on the order
of \( n \epsilon \). This can be expressed as

\[ |\sigma_i - \hat{\sigma}_i| \leq f_2(n) \cdot \epsilon \cdot \sigma_i \]

(8.2)

where \( \sigma_i \) is a singular value of \( A \), \( \hat{\sigma}_i \) is its computed value, and \( f_2(n) \) is a modestly-grown function of \( n \): the precise meaning of \( f_2(n) \) is discussed at the end of this section. The modified QR algorithm is implemented in the LAPACK system [2].

By combining the forward and backward error bounds, I obtain conditions for the correctness of the determinant sign as computed by the SVD. Let \( A \) be an exact integer matrix, and \( A + E_A \) its floating-point representation, with \( \|E_A\| \leq \epsilon \|A\| \).

Backward stability (8.1) states that the SVD algorithm computes the sign of the determinant of a matrix \( A + E_A + E_{SVD} \) for a small perturbation \( E_{SVD} \). On the other hand, forward stability (8.2) states that the distance from \( A + E_A \) to the nearest singular matrix is computed accurately as \( \sigma_n(A + E_A) \). If the net perturbation of \( A \) is less than the distance to the nearest singular matrix, the computed sign is correct. That is, the sign is correct if

\[ \|E_A + E_{SVD}\| < \sigma_n \]

(8.3)

Neither of these quantities is available exactly. However, the following inequality can be tested, and if it holds, it is sufficient to establish equation (8.3) and the correctness of the sign.

\[ (f_1(n) + 1) \cdot \epsilon \cdot \sigma_i < \sigma_n \]

(8.4)

The proof that (8.4) implies (8.3) is established using the backward stability of the SVD algorithm (8.1), which ensures that \( \|E_{SVD}\| \) is small.

\[
\begin{align*}
\|E_A + E_{SVD}\| &\leq \|E_A\| + \|E_{SVD}\| \\
&\leq \epsilon \cdot \|A\| + f_1(n) \cdot \epsilon \cdot \|A\| \\
&= (f_1(n) + 1) \cdot \epsilon \cdot \|A\| \\
&= (f_1(n) + 1) \cdot \epsilon \cdot \sigma_i
\end{align*}
\]

Rearranging the test in equation (8.4), it can be seen to express a static bound on
the condition number $\kappa = \sigma_1/\sigma_n$:

$$\sigma_1/\sigma_n < \frac{1}{(f_1(n) + 1) \cdot \varepsilon}.$$  \hfill (8.5)

In other words, the SVD algorithm dependably computes the sign of the determinant whenever the matrix is well-conditioned. Moreover, the notion of "well-conditioned" is explicitly defined in terms of a largest allowable condition number, determined by the matrix size and the working precision.

To evaluate the test (8.5), one uses the computed singular values $\tilde{\sigma}_1, \tilde{\sigma}_n$. These are not the exact singular values, but they are close. The test needs to be weakened slightly to account for the error present in the singular values. Taking advantage of forward stability (8.2), one obtains absolute error bounds on the actual singular values $\sigma_1, \sigma_n$ in terms of the computed singular values $\tilde{\sigma}_1, \tilde{\sigma}_n$. I use the upper bound on $\sigma_1$ and the lower bound on $\sigma_n$. The computed version of (8.5) is

$$\tilde{\kappa} = \tilde{\sigma}_1/\tilde{\sigma}_n < \frac{1}{(f_1(n) + 1) \cdot \varepsilon} \cdot \frac{1 - f_2(n) \cdot \varepsilon}{1 + f_2(n) \cdot \varepsilon}.$$  \hfill (8.6)

The procedure for computing the determinant sign is straightforward:

1. Approximate the integer matrix $A$ with a double-precision floating-point matrix and compute the singular value decomposition $P\Sigma Q$.

2. Compute the condition number $\tilde{\kappa}$ from $\Sigma$ and the determinant sign from $P$ and $Q$.

3. If $\kappa$ is small (it satisfies (8.6)), accept the determinant sign.

If $\tilde{\kappa}$ is large, the filter rejects the matrix, and another method must be used to find the sign.

Many systems offer some form of extended precision arithmetic implemented in software, with a precision of $\varepsilon'$. If extended precision is available, the same value of $\kappa$ can be compared against the looser static bound involving $\varepsilon'$. That is, the filter reveals whether the extended-precision SVD will give a dependable answer before the extended-precision factorization is computed. Lastly, the determinant magnitude can be estimated by $\prod \tilde{\sigma}_i$, so even when the filter rejects the determinant sign, the SVD reveals a good guess as to whether Lagrange-style or Newton-style reconstruction will be more efficient if modular computation follows. This idea is developed in section 8.7, in which I propose a multistage strategy.
Reliability of filters based on backward error estimation. Backward error bounds in numerical analysis are usually viewed as “approximate error bounds.” In other words, the error bounds are an accurate estimate of the actual net error committed. The bounds are not mathematically proven to be strict upper bounds. Often such bounds are known only up to a constant factor.

Therefore, the possibility remains that a determinant sign filter based on backward error estimation, like the SVD filter presented here, could return an incorrect sign and certify it as correct. Using the most conservative error bounds, such as $f_1(n) = 100n^3$ in the backward error in LAPACK’s SVD algorithm, does not remove this possibility.

Yet in practice, the error bounds are accurate. In my tests, the SVD filter always returned either the correct sign or a warning that the sign was not to be trusted. Experimentation showed that the less conservative bounds $f_1(n) = n$ and $f_2(n) = n$ were still conservative enough for all of the matrices in my tests.

8.5 Multiprecision interval arithmetic

The two determinant sign methods presented so far, modular arithmetic and the floating-point filter based on SVD, can be combined into a powerful two-stage evaluation system. However, before I discuss hybrid systems, I will introduce a third method based on forward error propagation. In this section, I describe a means for computing the determinant sign using multiprecision floating-point arithmetic with explicit maintenance of lost precision. The algorithm in this section is due to Shankar Krishnan. The arithmetic system is described in detail in Krishnan et al. [63]. The arithmetic system forms the foundation of the PRECISE library for matrix operations and polynomial root finding.

The numerical data structure consists of a sign, a mantissa, an exponent, and a bound $\delta$ on the absolute error. The mantissa is represented as an arbitrary-length array of digits. Only significant digits are stored, so as the error increases through the computation, the mantissa length decreases. This strategy has two principal advantages. One is that arithmetic speeds up as precision is lost. The other advantage is that the error always lies entirely in the last digit, allowing $\delta$ to be represented by a single-precision quantity. The number type behaves like interval arithmetic over a “bigfloat” type (such as LEDA’s), though it requires only one multiprecision computation per operation and a single precision computation to maintain $\delta$, rather than two multiprecision operations.
To perform a computation over this numerical type, one first specifies a \textit{fixed}
maximum precision $P$ for the computation. The input data are rounded off to $P$ bits,
and then as the computation progresses, numbers are computed only to a precision
of $P$ bits. The bounds $\delta$ keep track of the forward error in the computation. If
the result has enough precision, the computation terminates. For instance, if one is
interested in the sign of an expression, the result's interval is checked to see whether
it contains zero. If the result is too imprecise, $P$ is increased and the computation
restarts. By controlling $P$, the system adapts the precision to the forward stability
of the computation. The performance of any algorithm in this framework clearly
depends on the number of times the computation has to be repeated.

The possibility remains that the result is exactly zero. In general, this situation is
difficult to resolve. But if the result is known to be an integer, as it is in this context,
the sign is established whenever the interval length is less than one.

The method computes the determinant sign by performing interval Gaussian elim-
ination with partial pivoting. The algorithm starts by fixing the initial precision $P$
for the computation. Ideally, $P$ should be set to an unknown quantity $P_{\text{ideal}}$: the
smallest value of $P$ such that the sign of the diagonal entries after Gaussian elimi-
nation can be determined unambiguously. A larger value for $P$ results in extra work
at each arithmetic operation. A smaller value also results in extra work, in the form
of restarts. The ideal value $P_{\text{ideal}}$ is difficult to compute, so I describe an estimation
method which has been successful in practice. The algorithm begins by setting

$$P = \alpha \cdot n \cdot \log \|A\|_F ,$$

where $\alpha$ is a constant less than one (0.1 suffices). It is easy to see that if $\alpha = 1,$
this bound is similar to the determinant bitlength predicted by Hadamard's bound.
During the course of Gaussian elimination, if any diagonal element loses enough
precision so that its sign is indeterminate, $P$ is increased and the entire computation
is redone. Suppose that the pivot after $i$ steps of Gaussian elimination ($0 \leq i \leq n - 1$)
is found to have an ambiguous sign. The method then increases $P$. The closer $i$ is to
$n$, the closer $P$ is to $P_{\text{ideal}}$. I use the following formula to decide how much $P$ should
be increased for the next attempt at Gaussian elimination.

$$\Delta P = \alpha \cdot i \cdot \log \|A_i\|_F$$

where $\|A_i\|_F$ is the absolute value of the largest element in the lower right-hand
submatrix of \( A \) before column \( i \) is eliminated. (\( A_i \) is therefore a matrix of size \((n - i) \times (n - i)\).) In my experiments, this estimate is good especially when the determinant size predicted by Hadamard’s bound is pessimistic.

The principal difference between PRECISE’s number type and adaptive precision schemes like LEDA’s \texttt{real} type and CORE’s \texttt{Expr} type is that PRECISE does not store the structure of the determinant expression at run time. The disadvantage to this approach is that PRECISE’s numbers are not drop-in replacements for built-in numerical datatypes. The outer loop for repeating the computation at higher precision must be explicitly programmed. The advantage is that the structure is not stored. As most determinant algorithms perform \( O(n^3) \) arithmetic operations, the cost of maintaining the expression structure is appreciable for even a moderate-sized matrix.

8.6 Empirical results

My determinant sign algorithm is a multi-stage system built from the three methods described in sections 8.3 through 8.5. There is no single best way to combine the methods into a multi-stage system. The system design depends on the sort of matrices expected for a given application. In this section, I evaluate the effectiveness of each of the three methods on various matrix types. The following section will discuss ways to combine the three methods into a hybrid system.

My main battery of test matrices comes from a few representative algebraic problems of the type described in section 8.2. The table in figure 8.3 describes the problems in terms of their algebraic degree and bit complexity, and characterizes the matrix determinants in terms of their size and bit complexity. In all cases, the curves and surfaces are derived from inputs with reasonable bit lengths. The algebraic curves are intersection curves of various ellipsoids and tori. An ellipsoid is defined in terms of a center point \( p \) and three axis vectors \( \{v_i\} \), while a torus is defined in terms of a vector frame \( \{v_i\} \) and two radii \( r_1, r_2 \). The surface-surface-surface problem intersects three quadric surfaces. The surfaces are line-line bisectors, where each line is defined by two points \( p_{1,2} \). For the examples considered here, all of the quantities \( p, p_i, v_i, r_i \) are given as 32-bit integers, and most require fewer than eight bits.

The SVD algorithm of James Demmel is implemented in the LAPACK library [2]. For the modular algorithm, I use Patrick Theobald’s implementation in the LiDIA library [45], which uses Lagrange’s interpolation algorithm. I have modified the li-
<table>
<thead>
<tr>
<th>Problem</th>
<th>Degree of problem</th>
<th>Degree of $f(u)$</th>
<th>Coef bits in $f(u)$</th>
<th>Matrix max $n$</th>
<th>Matrix entry bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve-line intersection</td>
<td>4</td>
<td>4</td>
<td>32</td>
<td>7</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
<td>56</td>
<td>15</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>16</td>
<td>54</td>
<td>31</td>
<td>58</td>
</tr>
<tr>
<td>Curve-curve root projection</td>
<td>4.4</td>
<td>8</td>
<td>105</td>
<td>15</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>8.7</td>
<td>24</td>
<td>156</td>
<td>47</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td>8.7</td>
<td>28</td>
<td>157</td>
<td>55</td>
<td>161</td>
</tr>
<tr>
<td>Surface-surface-surface root isolation</td>
<td>2.2.2</td>
<td>8</td>
<td>2820</td>
<td>15</td>
<td>2822</td>
</tr>
</tbody>
</table>

Figure 8.3: Algebraic and numerical complexity of the geometric problems used to generate the test matrices.

Library to support Newton’s algorithm as well. Sylvain Pion’s implementation of the Brönnimann et al. algorithm [9] is very efficient for matrices with 53-bit entries, but unfortunately my matrices do not fit this restriction. The multiprecision interval arithmetic is implemented in the PRECISE library [63].

Figures 8.4, 8.5, and 8.6 show running times of the determinant-sign algorithms on the matrices generated by my examples. The matrices from the various problems are pooled, then collated into three categories: well-conditioned, ill-conditioned but nonsingular, and singular.

One important observation from the experiments is that small determinants are rare. When I examined the determinant bitlength, I found that it was always at least 80% of the length predicted by Hadamard’s bound, and usually above 95%. Newton reconstruction takes about the same amount of time as Lagrange reconstruction when the determinant bitlength is large. Figures 8.4 and 8.5 omit the Newton algorithm for this reason. As the determinant magnitude grows, the running time is dominated by Gaussian elimination, and the choice of reconstruction algorithm matters less.

Singular matrices, on the other hand, do occur, and Newton’s algorithm far outperforms Lagrange’s algorithm on these matrices. The matrices are ill-conditioned by definition, so the SVD filter fails to be useful. Yet the SVD does reveal an estimate of the determinant magnitude which can be used to choose between Newton and Lagrange reconstruction. Since full Lagrange reconstruction is unnecessary on these matrices. Lagrange time is absent from figure 8.6. PRECISE requires significantly more time on singular matrices—so much more that it goes from an order of magnitude faster than modular arithmetic to an order of magnitude slower. Singular matrices are typically a worst-case input for methods based on forward error.
In all cases, the floating-point SVD algorithm takes a fraction of the time of the modular algorithms. In the data in figures 8.4 and 8.5, the ratio of SVD time to modular time never exceeds 0.28, and is less than 0.06 for larger matrices ($n > 10$). In figure 8.6, the SVD algorithm runs in about one-fourth the time of the Newton algorithm.

Unfortunately, the SVD filter fails to be useful on all matrices above $n = 15$ in my experiment. One might think that this due to overly pessimistic bounds on the error in the SVD algorithm, but in fact, the SVD computes an incorrect sign for about one-third of the ill-conditioned matrices in the experiment. Large matrices are not necessarily ill-conditioned in general, but all of the large matrices generated by my test cases are ill-conditioned.

In a separate experiment, I ran the algorithms on matrices with $5 \leq n \leq 53$ with uniformly-distributed 100-bit random entries (figure 8.7). Such matrices tend to be well-conditioned, and the SVD filter succeeded on all of the matrices I tried. Such a distribution might occur in a high-dimensional convex hull or Delaunay triangulation algorithm.

Since the subresultant matrices have a sparse structure, I implemented Wiedemann's algorithm [96]. The algorithm computes the characteristic polynomial over a finite field by iteratively multiplying the matrix by random vectors. Since the input matrix is used only for matrix-vector multiplication, the running time depends on the number of nonzero entries. I found that it ran significantly slower than the others I examined. This may reflect a flaw in my implementation strategy. It is also possible that my matrices are not large enough for the better asymptotic complexity of the sparse algorithm to take over.

The performance of the PRECISE-based algorithm depends on the determinant magnitude. From my data, PRECISE's performance appears to be independent of the condition number.
Figure 8.4: Well-conditioned matrices. Each symbol represents a matrix for which the SVD filter succeeded. SVD is many times faster than PRECISE. PRECISE is also many times faster than modular arithmetic, though this is not expressed in the graph. The SVD filter fails to be useful on all matrices above $n = 15$. The symbols fall into curves, since the matrices are not evenly distributed with respect to entry bitlength.
Figure 8.5: Ill-conditioned, but nonsingular, matrices. Each symbol represents a matrix for which the SVD failed to be useful but which turned out to be nonsingular. Since the SVD's sign is not dependable, I rely on the next fastest method, which is PRECISE in all cases. The SVD time is included on this graph to show that not much time is wasted on computing the SVD.

Figure 8.6: Singular matrices. The SVD filter fails to be useful. For these matrices, my Newton implementation (based on the LiDIA library) ceases after discovering that the determinant is zero modulo a single prime. As a consequence, modular arithmetic performs very well. PRECISE, on the other hand, performs poorly.
Figure 8.7: Random matrices with uniformly-distributed 100-bit entries. The SVD filter succeeds on all of the matrices in this test.
8.7 Hybrid methods

The algorithms presented here perform differently on different matrix classes. The following table indicates the most efficient algorithm for each class. Note that in the geometric literature the term "near-singular" usually refers to a matrix with a small determinant magnitude, while in the numerical computational literature, the same term refers to an ill-conditioned matrix. Both concepts are important, so I avoid the term "near-singular."

<table>
<thead>
<tr>
<th></th>
<th>Well-conditioned</th>
<th>Ill-conditioned</th>
<th>Singular</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>SVD</td>
<td>PRECISE</td>
<td>--</td>
</tr>
<tr>
<td>Small</td>
<td></td>
<td></td>
<td>Modular-Newton</td>
</tr>
</tbody>
</table>

Two modular reconstruction algorithms are not mentioned in the table. Lagrange reconstruction is useful when the tiny probability of failure inherent in early-exit Newton reconstruction is not tolerable, and the algorithm of Brönnimann et al. is useful for machine-precision matrices.

I propose a hybrid method, based on combining these algorithms in a flowchart. The design of the flowchart depends on the efficiency of the implementation of the various algorithms. Just as importantly, the design depends on the distribution of the matrices in the application at hand. For example, if singular matrices are extremely rare in practice, modular arithmetic could be omitted. One should also consider whether the unlikely possibility that Newton reconstruction fails is tolerable. One fact that seems constant is that the SVD is by far the cheapest algorithm, and even when it rejects the determinant sign, it provides enough information to suggest which of the other algorithms may be most efficient.

A flowchart for my application follows. It is based on several observations about my matrix distribution: singular matrices occur with some regularity, and nonsingular matrices with small determinants do not occur. Further, I am willing to tolerate the possibility that Newton reconstruction may fail.

1. Compute the SVD in machine-precision floating point arithmetic, with relative precision $\epsilon$.

2. If the matrix is well-conditioned with respect to $\epsilon$, stop.

3. If the matrix is ill-conditioned with respect to $\epsilon$, but well-conditioned with
respect to a system-supported extended arithmetic with precision \( \epsilon' \). Compute the SVD in this arithmetic and stop.

4. If the matrix is very ill-conditioned:

(a) Guess whether the matrix is singular by comparing the smallest singular value to a threshold. Is \( \sigma_n < T' \)? (The value of \( T \) affects only efficiency, not correctness.)

(b) If the matrix appears nonsingular, evaluate the sign using adaptive-precision arithmetic.

(c) If the matrix appears singular, compute the determinant in modular arithmetic with Newton reconstruction. (If the determinant is nonzero modulo the first prime, the matrix is nonsingular; switch to adaptive precision.)

8.8 Conclusions

I have presented a hybrid method for exactly computing the sign of the determinant of a moderate-sized integer matrix. The method begins by computing the singular value decomposition of the matrix. If the matrix is well-conditioned, the sign is known. Otherwise, information from the SVD is used to choose between a multiprecision interval algorithm and a modular arithmetic algorithm.

The determinant-sign filter based on the singular value decomposition is effective for well-conditioned matrices. For such matrices, the SVD reveals the sign at a fraction of the cost of an exact algorithm.

For ill-conditioned matrices, the SVD filter rejects the matrix, but the cost of the filter remains small compared to the fallback algorithm. Further, the SVD reveals useful information about the matrix that can aid in choosing a reconstruction algorithm. An efficient multiprecision interval arithmetic system is effective on ill-conditioned but nonsingular matrices. Modular arithmetic is effective at verifying that a matrix is singular.
Chapter 9

Implementation and Results

In this chapter, I describe my implementation of the algorithm. I highlight its performance on a few polyhedra. Finally, I analyze the performance of the implementation and the efficacy of the filters and efficiency techniques.

9.1 Implementation

The algorithm has been fully implemented in C++. Routines for root isolation in one, two, and three dimensions take advantage of the efficiency techniques described in chapter 6. My implementation uses the bigint and bigrational exact arithmetic types in the LiDIA library [45]. The MAPC library [61] contains polynomial data structures and algebraic routines used in 1-D and 2-D seam tracing. The LAPACK [24] library is used for its floating-point LU-decomposition, eigenvalue, and SVD routines. The PRECISE library [63] is used for its adaptive-precision Sturm sequence and determinant sign routines. The implementation consists of about 50,000 lines of C++ code, including MAPC (to which I contributed in the course of this research) but excluding the other libraries. Figure 9.1 is a block diagram illustrating the major components of the algorithm and their relationships.

As a postprocessing step, the seam curves are tessellated into polygonal chains in the respective parametric domains of their sheets. The chains are then used as trimming boundaries for Bézier patches, one patch per sheet. For the images shown here, the trimmed patches are then triangulated and rendered by the OpenGL graphics library [97]. The medial axis transform (MAT) is represented by the parametric surface patches together with a radius function \( r(s, t) \) for each patch. To compute the radius function, the parametric surface expressions \((X(s, t), Y(s, t), Z(s, t))\) are substituted.
Figure 9.1: The major components of the medial axis algorithm. The shaded areas denote libraries whose methods are not discussed in this dissertation.
into the distance function $\sqrt{d_z^2(x, y, z)}$ for either of the sheet governors. Since the squared distance function is quadratic in $x, y, z$ (chapter 4), the radius function is the square root of a degree-four rational function in $s$ and $t$.

Though the entire implementation is in C++. I found the algebraic facilities of Mathematica invaluable for prototyping the low-level parts of the system. The C++ expressions for computing the parametric bisectors were generated by Mathematica. For example, the bisector of two skew line segments is a hyperbolic paraboloid. The coefficients for the surface are rational expressions in the endpoints of the two line segments. These expressions could be computed at runtime by MAPC's polynomial arithmetic operations, but precomputed (and pre-simplified) expressions result in a significant reduction in the running time.

The techniques in chapter 7 for handling degeneracies in the medial axis are not yet fully implemented. Currently the implementation recognizes degenerate junctions and uses them as seam terminators but not as seam originators. All seams are found except those whose endpoints are both internal degenerate junctions. This includes degenerate seams. Since the algorithm relies on the connectivity of the medial axis, it must be restarted one or more times if deleting these seams disconnects the seam-junction graph. For example, the medial axis in figure 9.6 was found in three runs.

9.2 Results and examples

In this section I highlight several examples of polyhedra and their medial axes as computed by my implementation. Table 9.1 summarizes the running time on the examples shown. Times are reported for an SGI computer with an R12000 processor running at 300 MHz. The times reported include the spatial subdivision preprocess, seam tracing, and sheet identification.

I have omitted the seam polygonization (sheet-trimming) post-process from the running time. The tessellation of algebraic curves is a challenging problem, and I have implemented only a simple solution for the purpose of visualizing small examples.

Figure 9.2 shows a simple non-convex polyhedron (two tetrahedra attached at a face) and its medial axis. The medial surfaces are parts of planes and cones, and the seams are segments of lines and ellipses.\footnote{For color images, see http://www.cs.unc.edu/~geom/MAT.}

A more complex example is given in figure 9.4. This polyhedron is an octagonal
<table>
<thead>
<tr>
<th>Polyhedron</th>
<th>Input complexity</th>
<th>Output complexity</th>
<th>Running time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>faces</td>
<td>edges</td>
<td>vert.</td>
</tr>
<tr>
<td>Tiny (9.2)</td>
<td>6</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>L (9.3)</td>
<td>10</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>Groove (9.4)</td>
<td>16</td>
<td>38</td>
<td>24</td>
</tr>
<tr>
<td>Pizza (9.5)</td>
<td>56</td>
<td>124</td>
<td>70</td>
</tr>
<tr>
<td>Bracket (9.6)</td>
<td>252</td>
<td>373</td>
<td>121</td>
</tr>
<tr>
<td>Igloo (9.7)</td>
<td>148</td>
<td>222</td>
<td>76</td>
</tr>
<tr>
<td>Bagel (9.8)</td>
<td>104</td>
<td>156</td>
<td>52</td>
</tr>
<tr>
<td>Venus (9.9)</td>
<td>250</td>
<td>375</td>
<td>127</td>
</tr>
</tbody>
</table>

Table 9.1: Performance on some example polyhedra. Running times are on an SGI computer with an R12000 processor running at 300 MHz.

Figure 9.2: A simple non-convex polyhedron. (a) The polyhedron. (b) Its seam curves. (c,d) Its medial axis rendered as trimmed Bézier surfaces.
Figure 9.3: A simple non-convex polyhedron: an L-shaped box. (a) The polyhedron. (b) Its seam curves. (c) The central sheets.

box with two opposing grooves. The two non-convex edges are separated by a very small gap. Visualizing the entire medial axis is difficult. Figure 9.4(d) shows the medial axis with the outer parts removed. The remaining sheets form a surface similar to a hyperbolic paraboloid, but formed mainly of cylinders, cones, and planes. The small hyperbolic paraboloid that is the bisector of the two reflex edges is shown in figure 9.4(e). Each of the paraboloid's four edges is a single branch of a (planar) hyperbola.

Table 9.2 shows the time taken by an average 3-D Sturm computation. The computation proceeds in two stages. The first stage computes the Macaulay denominator and the portion of the Macaulay numerator that is not dependent on the evaluation point. The second stage computes the number of permanencies in the volume function at a given point in space. In order to compute the number of roots in a box, the second stage is repeated once for each corner.

### 9.3 Analysis

The overall running times in the previous section demonstrate that the algorithm runs in a reasonable amount of time on polyhedra with hundreds of faces. However,

<table>
<thead>
<tr>
<th>Task</th>
<th>Average time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set up system</td>
<td>38.9</td>
</tr>
<tr>
<td>Evaluate at one point</td>
<td>1.73</td>
</tr>
<tr>
<td>Count roots in one box</td>
<td>$38.9 + 8 \times 1.73 = 52.7$</td>
</tr>
</tbody>
</table>

Table 9.2: Average timing for quadric-quadric-quadric 3-D Sturm operations in the Venus model. Only the systems of full degree (2, 2, 2) are included in this average.
Figure 9.4: A polyhedron with two grooves. (a) The polyhedron. (b) Its seam curves. (c) Its medial axis. (d) The part of the medial axis which does not meet the boundary, magnified $2\times$. (e) The central sheet is a hyperbolic paraboloid, the bisector of the two grooves. Magnified $200\times$.  

Figure 9.5: The "iron maiden pizza box" and a schematic of its medial axis.
Figure 9.6: A CAD model of a bracket and a schematic of its medial axis. The medial axis has a few degenerate junctions, indicated by the isolated points. The implementation currently supports tracing only seams having at least one generic junction.

Figure 9.7: A CAD model and a schematic of its medial axis.

Figure 9.8: An irregularly-faceted torus and a schematic of its medial axis showing the central seams.
Figure 9.9: A model of the Venus de Milo sculpture. The medial axis of this asymmetrical polyhedron has no degenerate seams or junctions. It does have many small seams and sheets. (a) Venus. (b) Its seam curves as line segments. (c) The "central" seams, those that do not have an endpoint on the boundary.
the reported times vary widely among the examples. In this section, I account for the variation by identifying the major contributing factors to the overall running time. I also discuss the efficacy of the filters and other efficiency improvements.

To understand the efficiency of an algorithm, the first consideration is combinatorial complexity. The tracing algorithm is sensitive to both the input complexity (number of faces, edges, and vertices) and the output complexity (number of sheets, seams, and junctions). The output complexity can be approximately the same as the input complexity, or it can be much larger. For example, the Venus model has many more faces, edges, and vertices than the pizza box model, but their medial axes are of similar complexity.

In this section, the emphasis is not on combinatorial complexity, but rather on practical efficiency as defined in chapter 1: the ability of an implementation to produce accurate results in a reasonable amount of time.

### 9.3.1 Execution profile

An execution profile for the Venus model is shown in table 9.3. It is typical for the polyhedra I have tested. Most of the time is spent in 3-D Sturm evaluation. The next significant contributor is the spatial subdivision stage.

### 9.3.2 Factors affecting running time

This section identifies the factors that largely determine the running time. The list is not exhaustive; rather, it focuses on the factors that have shown to be most important in practice.

**Combinatorial complexity of input and output.** As I established in chapter 3, the algorithm's running time depends on both the input complexity $n$ and the output complexity $m$. It is also clear that $m$ depends not only on $n$ but also the polyhedron. For example, the pizza box model has fewer faces, edges, and vertices than the Venus model, but their medial axes have similar complexity.

**Input bit complexity.** For any expression evaluated in exact arithmetic, the execution time is sensitive to the input bit length. Since the medial axis running time is dominated by such exact computations, the input bit complexity is important. For example, the Venus model is the output of a simplification algorithm working
<table>
<thead>
<tr>
<th>Polyhedron</th>
<th>Time spent in the most significant routines (seconds)</th>
<th>Total time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Spatial subdiv</td>
<td>3-D Sturm setup</td>
</tr>
<tr>
<td>Tiny (9.2)</td>
<td>2.1</td>
<td>6.9</td>
</tr>
<tr>
<td>L (9.3)</td>
<td>2.3</td>
<td>5.5</td>
</tr>
<tr>
<td>Groove (9.4)</td>
<td>10.3</td>
<td>131.</td>
</tr>
<tr>
<td>Pizza (9.5)</td>
<td>215.</td>
<td>822.</td>
</tr>
<tr>
<td>Bracket (9.6)</td>
<td>2355.</td>
<td>10235.</td>
</tr>
<tr>
<td>Igloo (9.7)</td>
<td>163.</td>
<td>10502.</td>
</tr>
<tr>
<td>Bagel (9.8)</td>
<td>722.</td>
<td>830.</td>
</tr>
<tr>
<td>Venus (9.9)</td>
<td>2125.</td>
<td>15382.</td>
</tr>
</tbody>
</table>

Table 9.3: Breakdown of time spent in the most expensive routines. The routines are mutually exclusive. "2-D tracing" includes bisector parametrization, curve topology, 2-D junction location, and junction ordering.

in floating-point arithmetic. Its vertices are input as single-precision floating-point numbers and converted to exact rational numbers of approximately 24 bits in length. By contrast, the pizza box was modeled by hand using exact rational numbers such as $1/3$ and $4/5$. This is one of the main reasons that Venus takes so much longer than the pizza box.

**Algebraic degree.** A convex polyhedron has a completely linear medial axis. For a general polyhedron, curved bisectors occur only near concave features. The number of high-degree medial axis features depends in some way on the number of concave boundary features.

A more subtle dependence on algebraic degree arises from the use of the implicit face-face bisector. Usually a face-face bisector is a plane with irrational slope. The algorithm presented here represents this bisector by its implicit form with rational coefficients, which is a degree-two equation representing a pair of planes. Computing with these bisectors is just as expensive as computing with curved bisectors. However, the faces may happen to have a bisector plane with rational coefficients. This occurs when, for instance, the faces are parallel or perpendicular. In such cases, the algebraic degree drops, speeding up every operation involving the bisector. A junction defined by three rational bisector planes can be located by intersecting the planes directly, in a tiny fraction of the time it takes to isolate the solution with Sturm sequences.

Often, face-face bisectors make up a significant portion of the medial axis. Therefore, when many of the faces of a polyhedron are pairwise parallel or perpendicular,
the algebraic degree drops for many junctions. The total running time decreases significantly. This is the other reason that the pizza box example takes so much less time than the Venus model.

**Clearance between junctions.** Usually the Newton-Raphson approximation to a junction is fairly accurate, and the 3-D Sturm operation simply serves to verify its accuracy. But whenever two candidate junctions on a seam are sufficiently close together, the rational boxes must be reduced to determine their ordering. The impact of this on the total running time is surprisingly small, though, since refining a root is inexpensive compared to finding it in the first place (as evidenced by table 9.2).

### 9.3.3 Efficacy of filters

My accurate approach to the medial axis would not be practical without the efficiency techniques in chapter 6. In this section I examine the benefits of these techniques. Table 9.4 gives a numerical summary of the effect of the filters on a few of the examples.

**Spatial subdivision.** By far the most important efficiency technique is the spatial subdivision preprocess. In essence, this numerical preprocess alleviates the poor asymptotic behavior of the tracing algorithm. The subdivision step significantly narrows down the possibilities for the topology of the medial axis. In fact, many seams have only one possible terminating element: the rest are eliminated during subdivision. For these seams, the cost of tracing a single seam is just the cost of locating a single junction point. On the whole, few extra candidate junctions are created.

Subdivision provides a close approximation to the medial axis in both a topological and a geometric sense. Yet it should be emphasized that the result is always a

<table>
<thead>
<tr>
<th>Polyhedron</th>
<th>Running time</th>
<th>Running time without eigenvalue filter</th>
<th>Running time without eigenvalue filter or subdivision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tiny (9.2)</td>
<td>10.2 seconds</td>
<td>16.8 seconds</td>
<td>723 seconds</td>
</tr>
<tr>
<td>Groove (9.4)</td>
<td>152 seconds</td>
<td>221 seconds</td>
<td>8876 seconds</td>
</tr>
<tr>
<td>Pizza (9.5)</td>
<td>22.9 minutes</td>
<td>40.9 minutes</td>
<td>66 hours (estimated)</td>
</tr>
<tr>
<td>Bagel (9.8)</td>
<td>18.5 minutes</td>
<td>36.1 minutes</td>
<td>107 hours (estimated)</td>
</tr>
</tbody>
</table>

Table 9.4: The effect of two of the efficiency techniques on running times for some of the example polyhedra.
conservative estimate of the medial axis topology. It reveals a superset of the proximity relationships between boundary elements. For instance, it is possible for the subdivision to retain cells that lie entirely outside the polyhedron. It is the tracing process that ensures that such cells are discarded by finding a single connected component of the seam graph.

**Efficient 3-D Sturm sequences.** The two important efficiency techniques are partial evaluation and the eigenvalue-based floating-point filter. In section 6.3.3 I argued that the eigenvalue filter could speed up sequence evaluation by nearly a factor of ten. Testing the filter on two small polyhedra ("tiny" and "groove") supports this claim. Comparing eigenvalue-filtered sequence evaluation to the purely exact Vandermonde method, the filtered method outperformed the exact method by a factor of 9. Since sequence evaluation is a bottleneck in the overall algorithm, the speedup of 9 improves the total running time significantly. As reflected in the running times in table 9.4, the overall running time is improved by as much as a factor of 2. With the eigenvalue filter, sequence evaluation takes a small fraction of the total running time, effectively shifting the bottleneck entirely to the setup stage of the 3-D Sturm operation.
Chapter 10

Conclusions and Future Work

10.1 Summary

I have presented a reliable, accurate algorithm to compute the medial axis of a polyhedron using exact computation. The practicality of my approach has been demonstrated by an implementation, which has been applied to several nontrivial polyhedra. I have also given efficient solutions to several subproblems, including the parametrization of bisector surfaces, the intersection of three algebraic surfaces, and the sign of the determinant of a matrix. Techniques based on modular arithmetic, floating-point filters, adaptive precision, and lazy evaluation combine to improve the efficiency of the system by orders of magnitude.

My thesis is

The medial axis of a polyhedron can be computed accurately and efficiently using exact computation.

In chapters 3 through 5, I have presented an accurate algorithm for the polyhedral medial axis. The algorithm's accuracy derives from its foundation of exact computation. Exact computation prevents the inconsistencies that can arise when making decisions based on floating-point quantities. In my algorithm, exact computation is enabled by exact representations of algebraic points, curves, and surfaces, based on arbitrary-precision rational numbers.

The exact algorithm, as described in chapters 3 through 5, is not efficient enough to be practical. In chapter 6, I have given several techniques which together accelerate the algorithm by many orders of magnitude. The improvements are diverse, including pre-processing steps to reduce the number of candidate junctions, careful rearrangement of algebraic expressions, lazy evaluation techniques, and floating-point filters.
(One important subproblem, determinant sign evaluation, is treated separately in chapter 8.) The improved algorithm is efficient enough to be practical.

The goal of this research was to explore one type of solution to the geometric robustness problem in the context of a particular geometric construction, the polyhedral medial axis. The robustness problem has two aspects: roundoff error and degeneracies. My exact algorithm solves the problem of roundoff error by ensuring that it affects no decisions. The degeneracy issue has not been completely solved, so it would be misleading to call my algorithm robust in its current form. However, in chapter 7 I have described in detail the common forms of degeneracy and given algorithmic means to identify and handle them.

In the course of developing my accurate algorithm, I have arrived at new solutions to several subproblems. In many cases, the new solutions rely on results from algorithmic algebra, numerical analysis, and geometric computing.

- **Three exact seam searching algorithms.** The fundamental step in my medial axis algorithm is based on ordering points along an algebraic seam curve. I have presented three different algorithms for this operation, corresponding to three representations for the seam curve.

- **Efficient intersection of three algebraic surfaces.** A medial axis junction point lies at the intersection of three quadric surfaces. I have described an efficient method for isolating and refining solutions to a system of three low-degree polynomial equations in three variables, based on multivariate Sturm sequences and the Macaulay resultant. The isolation method provides a form of lazy evaluation for junction points. The method's efficiency is due to three particular techniques:
  - The separation of the operation into two parts to avoid repeated work.
  - A floating-point filter based on the eigenvalues of a matrix.
  - The use of multivariate Newton's method to guess the locations of roots.

- **Parametric forms for bisector surfaces of points, lines, planes.** I have presented practical methods for computing rational parametrizations of all of the quadric surfaces that occur in the medial axis.

- **A complete differential-geometric characterization of the incidence structure to a generic or degenerate junction point.** I have given a complete characterization
of the structure of the incident curves and surfaces at a degenerate junction point. I have also given algorithms to identify and distinguish these curves and surfaces.

- *Two pre-processes for improving the overall running time.* No asymptotically efficient algorithm for this problem is known. I have presented two high-level strategies for improving the running time on a class of inputs that is common in practice. One strategy is based on polytope intersection, and its effect on the overall running time is easily understood both in theory and in practice. The second strategy is based on recursive subdivision of space. It is an improvement upon the subdivision strategies suggested by other authors.

- *A hybrid method for evaluating determinant signs for moderate-sized matrices.* Algebraic computation for geometric problems often relies on Sturm sequences. The subresultant algorithm for evaluating Sturm sequences requires an efficient means to find the sign of a matrix determinant. My algorithm for intersecting three quadric surfaces commonly requires determinant signs of order 15 with entries many times larger than machine precision. I present a three-stage hybrid algorithm for evaluating such determinants. The stages are

  - A floating-point filter based on the singular value decomposition (SVD) of a matrix.

  - Gaussian elimination implemented over a carefully designed arbitrary-precision numerical data type.

  - Exact determinant evaluation with modular arithmetic.

10.2 Discussion

Exact computation may seem a heavy-handed approach. Arbitrary-precision arithmetic is expensive. Efficient, filtered methods are difficult to implement because they require two or more different algorithms to solve the same problem under different conditions. It is reasonable to hope that this effort could be avoided, that one could design a continuous polyhedral medial axis algorithm without recourse to such complex and expensive operations. Indeed, for several years researchers have attempted to find such algorithms. Unfortunately, very few practical algorithms have been found, and these are subject to reliability problems that appear to limit their
applicability. I have shown here that reliability can be attained at the cost of some
efficiency at running time and some complexity of implementation. Although my al-
gorithm is slower than others based on floating-point arithmetic, it is efficient enough
for practical applications where either reliability or accuracy is important.

The medial axis transform is a useful shape transformation in many contexts. 
However, the shape in question is not always a polyhedron. Objects designed with
geometric solid modeling software often have curved boundary elements in addition to
flat facets. Medial axis computation also has applications to computer vision, in which
objects are usually described on a discrete image raster. Polyhedra are nevertheless
important, because they are widely used to approximate three-dimensional objects of
all kinds.

If polyhedra are often approximations to non-polyhedral objects, one may wonder
whether an exact algorithm is warranted. For instance, a polyhedron approximating
a curved object will have a discretization error that is orders of magnitude larger
than floating-point roundoff error. In fact, an exact medial axis is usually not re-
quired. Solid modeling operations, for instance, may only require six decimal digits
of precision. In this dissertation, exactness itself is not the goal. Rather, I have used
exact computation as the means to reliability. By assuming that the input is known
to infinite precision, my algorithm is able to avoid drawing inconsistent conclusions
from uncertain data.

For the general problem of shape understanding, a reasonable approach based on
the medial axis is as follows.

1. Approximate the shape by a polyhedron.

2. Extract the medial axis of this polyhedron.

3. Employ this medial structure in shape analysis.

Unfortunately, this approach suffers from a subtle difficulty. The structure of the
medial axis is highly sensitive to the choice of polyhedral approximation. Given
two different tessellations of the same shape, the polyhedral medial axes will almost
certainly have no one-to-one correspondence of sheets. In order to perform shape
analysis, it is necessary to process the polyhedral medial axis to extract the salient,
stable features. Such processing is an active area of current research, and it appears
that this problem is at least as difficult as the computation of the medial axis.

Many shape analysis researchers eschew the continuous medial axis for this reason.
Their challenge is to compute an approximate medial structure without assuming an
exact boundary representation. The resulting algorithms essentially accomplish medial surface extraction and simplification at the same time. Such approaches still face the problem of algorithm stability: the potential for drawing inconsistent conclusions from approximate data. Robustness problems can occur at any level: at the floating-point precision of sixteen decimal digits; at engineering precision, six digits; or at image precision, three.

Despite its efficiency, my algorithm is not practical for polyhedra with many thousands or millions of faces. Inputs of this complexity can occur when, for instance, a polyhedral boundary is extracted from a three-dimensional digital image. For such applications, a continuous approach may be inappropriate. Designing a reliable approximate medial axis algorithm for shapes of very high description complexity remains a challenging problem.

I have shown here that careful use of symbolic computation (algorithmic algebra) can lead to efficient geometric algorithms. Yet symbolic techniques are not widely used for such applications. I believe that symbolic techniques are avoided due to their perceived inefficiency. Symbolic techniques usually assume exact arithmetic, which is expensive. However, the perception may also be due to the poor asymptotic complexity of known algorithms for computing Gröbner bases of systems of polynomial equations. Gröbner bases are very general tools for solving such systems, but they can be prohibitively expensive for applications such as the present one. Another reason that symbolic techniques are perceived to be slow is that they are most familiar in software systems such as Mathematica and Maple. These systems typically sacrifice some efficiency for generality and flexibility. By avoiding expensive Gröbner basis techniques and implementing algebraic algorithms carefully, one can design efficient, reliable geometric software.

10.3 Future work

One important area for future work is the treatment of degeneracies. At a very degenerate junction (one with \( k \gg 4 \) governors), the exhaustive checking of all governor triples becomes very expensive, since the complexity grows with \( k^4 \). A similar inefficiency occurs during the sheet identification stage, in which an incoming seam at a junction is matched with each other seam to find one that shares two governors. Though I give a complete geometric characterization of a junction point, a useful further result would be a complete understanding of the combinatorial structure of
the seams incident to a junction point, to allow for more efficient searching among these seams.

The algorithm as a whole would benefit from a uniform treatment of degenerate configurations. Symbolic perturbation techniques may prove an effective alternative to the enumeration approach taken here. Perturbation techniques developed for this medial axis algorithm would likely be useful in many other non-linear geometric constructions.

The problem of degree-four seam curves with singular points is unsolved. It remains to be seen whether the most effective method would involve detecting and analyzing the singular points or removing them with symbolic perturbation.

A global pre-process that modifies the face equations slightly could decrease running times significantly. In a typical polyhedral medial axis, many sheets are face-face bisectors. Unless a face-face bisector is a rational plane, it is treated as a degenerate quadric. A sufficient (but not necessary) condition for all face-face bisectors to have rational coefficients is that each face's unit normal vector have rational coefficients. Guaranteeing this property by a modification of the polyhedron would be a certain win. An interesting problem is to adjust the vertices so that the property holds without moving the vertices very far and without producing self-intersections.

Finally, some CAD models, such as the bracket in chapter 9, involve spherical and cylindrical surfaces. Such surfaces are highly symmetrical. When tessellated, they tend to produce degenerate and near-degenerate polyhedral medial axes. It seems a waste to treat such models as degenerate, as they are in fact approximations to simple objects with medial surfaces that are possibly non-degenerate in some sense. While handling general curved boundaries is a very difficult problem, spheres and cylinders may be amenable to a treatment similar to the one given here, as their bisectors are still quadric loci.
Appendix A

Algorithmic Algebra

The algorithms in this dissertation rely on several previous results in algorithmic algebra. In this appendix I summarize these results. Section A.1 describes two formulations of the resultant of a collection of polynomials. Resultants are the basis of elimination theory. Section A.2 summarizes the technique of univariate Sturm sequences, also known as polynomial remainder sequences. Section A.3 describes a generalization of Sturm theory to multidimensional polynomial systems. Finally, section A.4 gives references for the various techniques.

A.1 Resultants

Resultants are an important tool in solving systems of polynomial equations. By way of motivation, consider the following uses of the matrix determinant in systems of linear equations.

A homogeneous linear system of \( n \) equations in \( n \) unknowns has a solution if and only if the determinant of the coefficient matrix is zero. For example, when \( n = 2 \), the condition

\[
\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = 0
\]

is a necessary and sufficient condition for the existence of a solution to the system

\[
\begin{align*}
ax + by &= 0 \\
(cx + dy) &= 0.
\end{align*}
\]

Resultant theory is an extension of this mechanism to systems of polynomial equations. The goal of resultant theory is to find, for a system of polynomial equations, a necessary and sufficient condition for the existence of a solution. The condition should be an expression in the coefficients of the polynomials. Several different formulations have been devised over the last century. I will focus on two formulations: Sylvester's and Macaulay's.
The Sylvester resultant formulation is often called “the resultant” in computer algebra text books and software systems. Sylvester’s formulation gives a necessary and sufficient condition for two polynomials in one variable to have a common root. The condition is expressed as a matrix determinant. For the system

\[
\begin{align*}
  f(x) &= a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0 = 0 \\
g(x) &= b_m x^m + b_{m-1} x^{m-1} + \cdots + b_0 = 0
\end{align*}
\]

the Sylvester resultant is the \((m + n) \times (m + n)\) determinant

\[
\text{Res}_x(f, g) = \det \begin{pmatrix}
a_n & a_{n-1} & \cdots & a_0 \\
a_n & a_{n-1} & \cdots & a_0 \\
\vdots & \vdots & \ddots & \vdots \\
a_n & a_{n-1} & \cdots & a_0 \\
b_m & b_{m-1} & \cdots & b_0 \\
\vdots & \vdots & \ddots & \vdots \\
b_m & b_{m-1} & \cdots & b_0
\end{pmatrix} = 0
\]

where the blank entries are zeros.\(^1\) In brief, the coefficients of the degree-\(n\) polynomial is repeated \(m\) times and the degree-\(m\) polynomial is repeated \(n\) times.

A useful way of viewing this matrix is to imagine the columns as labeled with the monomials \(x^{n+m-1}, x^{n+m-2}, \ldots, x^0\). With this labeling, the last \(a\)-row simply puts the coefficients of \(f(x)\) in their proper column. The row immediately above is \(xf(x)\), then \(x^2f(x)\), and so on to \(x^{m-1}f(x)\). Similarly, the \(b\)-rows are monomial multiples of \(g(x)\). One may view the Sylvester resultant as the construction of \(m + n\) polynomials of degree \(m + n - 1\) whose coefficients may be put into a square matrix. The number \(d = m + n - 1\) can be considered the “Sylvester degree” of the system (my terminology).

The most important application of the polynomial resultant is elimination. Given a system of two polynomials in two variables, Sylvester’s resultant can be applied by absorbing one of the variables into the coefficients. In the system (A.2), the coefficients \(a_i\) and \(b_i\) are replaced by polynomials in \(y\). The resultant condition (A.3) then becomes a single polynomial equation \(R(y) = 0\). The roots of \(R(y) = 0\) are the values of \(y\) such that the system (A.2) has a solution in \(x\). The resultant has

---

\(^1\)Many sources have this matrix typeset incorrectly. There are no zeros on the main diagonal.
eliminated the variable \( x \) from a system in \( x \) and \( y \). Geometrically, if the two original polynomials are the equations of planar curves, then the resultant is a polynomial whose roots are the \( y \)-coordinates of the curve-curve intersection points.

For more than two equations, the most general resultant formulation is Macaulay's (Macaulay [66], Canny [14]). It applies to a system of \( n \) homogeneous polynomials in \( n \) variables. It is a generalization of the Sylvester resultant in the sense that the Macaulay resultant with \( n = 2 \) applied to the homogenization of the system (A.2) is the homogenization of the Sylvester resultant (A.3).

Let \( f_1, \ldots, f_n \) be homogenous polynomials in \( x_1, \ldots, x_n \). Denote the degree of \( f_i \) by \( d_i \). The size of the Macaulay matrix is determined by a certain number \( d \), which I shall call the "Macaulay degree" of the system:

\[
d = 1 + \sum_{i=1}^{n} (d_i - 1)
\]

For \( n = 2 \), the Macaulay degree is the same as the Sylvester degree. Each polynomial \( f_i \) whose degree is strictly less than \( d \) is "promoted" to a polynomial of degree \( d \) by multiplying by a homogeneous monomial of degree \( d - d_i \).

The Macaulay resultant is formulated as the quotient \( \det(M)/\det(M') \) of two matrix determinants. First I shall construct the numerator matrix \( M \). Let \( X^d \) be the set of homogeneous monomials of degree \( d \) in \( x_1, \ldots, x_n \):

\[
X^d = \{ x_1^{\alpha_1} \cdots x_n^{\alpha_n} : \alpha_1 + \cdots + \alpha_n = d \}
\]

The columns of \( M \) are indexed by the monomials in \( X^d \). The number of monomials is \( N = \binom{n + d - 1}{d} \). Each row of \( M \) contains the coefficients of a polynomial of the form \( x_1^{\alpha_1} \cdots x_n^{\alpha_n} \cdot f_i \), where \( f_i \) is one of the original polynomials. and \( \alpha_1 + \cdots + \alpha_n + d_i = d \). In other words, each row of \( M \) is one of the \( f_i \), promoted to the Macaulay degree \( d \) by multiplication by some monomial.

The promotions themselves are not particularly intuitive. The coefficients of the \( f_i \) are placed in \( M \) according to the following algorithm. First, the rows of \( M \) are put into correspondence with the elements of \( X^d \). (The particular correspondence is not important. and an example is given below.) Consider a row \( k \), corresponding to the monomial with exponents \( (\alpha_1, \ldots, \alpha_n) \). If \( \alpha_1 \geq d_1 \), then this row will contain the coefficients of \( f_1 \). Otherwise, if \( \alpha_2 \geq d_2 \), then the row will contain the coefficients of \( f_2 \), and so on. In general, the coefficients of \( f_i \) are put into all rows \( (\alpha_1, \ldots, \alpha_n) \) such
that

- \( \alpha_i \geq d_i \)

- \( \alpha_j < d_j \) for all \( j < i \)

To promote \( f_i \) to degree \( d \), it must be multiplied by some monomial. The monomial is the row's monomial divided by \( x_i^{d_i} \), so that row \( k \) contains the coefficients of

\[
F_k = \frac{x_1^{\alpha_1} \cdots x_n^{\alpha_n}}{x_i^{d_i}} f_i
\]

The denominator matrix \( M' \) is the submatrix of \( M \) defined by

- deleting the columns of \( M \) corresponding to monomials \( (\alpha_j) \) where \( \alpha_j < d_j \) for all but one value of \( j \)

- deleting the rows of \( M \) whose original polynomial \( f_i \) is multiplied by a monomial \( (\alpha_j) \) where \( \alpha_j < d_j \) for all \( j > i \)

A useful consequence of this deletion rule is that all rows corresponding to the last polynomial \( f_n \) are deleted. This property is useful for 3-D Sturm sequence evaluation, where the resultant of \( \{f_1, f_2, f_3, f_4\} \) is repeated for eight or more slightly different values of \( f_4 \). As long as the changing polynomial is listed last, the Macaulay denominator \( M' \) does not change.

The ordering of the monomials in \( X^d \) is arbitrary. (Thus the ordering of the rows and columns of \( M \) is arbitrary, and I have defined the resultant only up to a sign. This suffices for elimination, since the resultant is a polynomial whose importance lies in its roots.) To construct the Macaulay matrix efficiently, it is useful to have a fixed ordering on \( X^d \), a two-way mapping between the monomials in \( X^d \) and the integers \( 0, \ldots, N \). One such mapping is given by lexicographic ordering on the monomials \( (\alpha_1, \ldots, \alpha_n) \). This mapping may be efficiently computed in the following manner.

First define \( P(a, b) = \binom{a-b}{b} \). Construct a small table of the values of \( P(a, b) \) for \( a = 0, \ldots, d \) and \( b = 0, \ldots, n - 1 \). The table is part of Pascal’s triangle. It can be constructed by the recurrence

\[
P(a, 0) = 1, \\
P(0, b) = 1, \\
P(a, b) = P(a-1, b) + P(a, b-1).
\]

138
The final entry in the table is $P(d, n - 1) = N$.

Now, given the monomial $(\alpha_1, \ldots, \alpha_n)$, the corresponding row/column index $k$ is given by

$$
k = P(\alpha_n, 0) - P(0, 0)
+ P(\alpha_n + \alpha_{n-1}, 1) - P(\alpha_n, 1)
+ P(\alpha_n + \alpha_{n-1} + \alpha_{n-2}, 2) - P(\alpha_n + \alpha_{n-1}, 2)
+ \ldots
+ P(\alpha_n + \ldots + \alpha_1, n - 1) - P(\alpha_n + \ldots + \alpha_2, n - 1)$$

This method for computing $k$ takes optimal $\Theta(n)$ time.

To compute the exponents $\alpha_1, \ldots, \alpha_n$ from the index $k$, let $D_1, \ldots, D_n$ stand for the lines of the expression above (each $D$ is a difference of two values of $P$). Choose $\alpha_1$ as large as possible so that $k - D_n \geq 0$. Then choose $\alpha_2$ as large as possible so that $k - D_n - D_{n-1} \geq 0$, and so on. The largest allowable $\alpha_j$ is found by searching through one row of the table of values of $P(a, b)$. This method takes $O(n + d)$ time, slightly worse than the optimal $\Theta(n)$.

### A.2 Univariate Sturm sequences

The Sturm sequence is a basic tool for isolating the real roots of a univariate polynomial. Given a square-free polynomial $f(u)$ with integer coefficients, the Sturm sequence of $f(u)$ is the polynomial remainder sequence \{f_1(u), \ldots, f_m(u)\} where

\[

t_1(u) = f(u),
\]

\[

t_2(u) = f'(u),
\]

\[
\hat{t}_i(u) = -\text{rem}(f_{i-1}(u), f_{i-2}(u)). \quad i = 3, \ldots, m
\]

Here $\text{rem}(f, g)$ gives the remainder on polynomial division of $f$ by $g$. The sequence is finite with length at most the degree of $f(u)$.

Evaluating \{f_i\} at a real number $a$ and counting the number of sign changes in the sequence gives the variation operator $\text{var}_f(a)$. The variation operator has the useful property that $\text{var}_f(b) - \text{var}_f(a)$ computes the number of real roots of $f$ in the interval $[a, b]$. Multiple roots are counted only once and complex roots are ignored. Using $\text{var}_f()$, one may easily construct disjoint intervals, each containing one of the real roots of $f$. Such an interval, together with $f$, is a complete specification of a real algebraic number.
The Sturm sequence can be computed by symbolic polynomial division. The process is similar to Euclid's GCD algorithm applied to \( f(u) \) and \( f'(u) \). Explicitly computing the Sturm sequence is an effective approach in some situations. But the length of the coefficients of the polynomials \( f_i \) can grow exponentially with \( i \) in this algorithm. In fact, the coefficient growth is avoidable. The integer coefficients of \( f_i \) share a large common factor. The subresultant algorithm of Collins [19] computes the coefficients of \( f_i/c_i \), where \( c_i \) is a positive integer dividing the coefficients of \( f_i \). The coefficient length of the modified sequence grows linearly. Further, the algorithm expresses the coefficients as matrix determinants. The matrices are submatrices of the Sylvester resultant matrix of \( f \) and \( f' \). Each of the coefficients may be computed individually, so one may compute only the constant terms of the \( \{f_i\} \) and obtain \( \text{var}_f(0) \). In fact, we need only the signs of the constant terms. Thus, \( \text{var}_f(0) \) can be computed by evaluating determinant signs. The number of variations at an arbitrary rational number \( a \) is computed by expanding \( g(x) = f(x - a) \) symbolically and computing \( \text{var}_g(0) \).

A.3 Multivariate Sturm sequences

Sturm sequences can be generalized to root isolation problems in dimensions higher than one. The goal is to count the number of real, zero-dimensional solutions to a system of \( n \) polynomials in \( n \) variables within an \( n \)-dimensional interval (box). Multivariate Sturm techniques are given by Pedersen [78] and Milne [71]. I shall summarize Milne's technique, which is used in the medial axis algorithm.

The two-dimensional root isolation problem asks for the number of real solutions to the system

\[
\begin{align*}
  f(x, y) &= 0 \\
  g(x, y) &= 0
\end{align*}
\]  

(A.4)

inside a two-dimensional interval \([x_0, x_1] \times [y_0, y_1]\). Milne’s technique constructs a volume function \( V(u) \) at each of the four box corners. The volume function is a univariate polynomial with the property that its real roots are in one-to-one correspondence with the real roots of the system (A.4).

The volume function is constructed with respect to an arbitrary point \((\alpha, \beta)\) in the plane. A root \((x, y)\) of the system (A.4) is said to have the signed volume \((x - \alpha)(y - \beta)\).
Geometrically, the signed volume is positive if \((x, y)\) is above and to the right, or below and to the left, of \((\alpha, \beta)\). See figure A.1. The volume function \(V(u)\) has the property that each root \(u\) is the signed volume of one of the roots. To construct \(V(u)\) at the point \((\alpha, \beta)\), adjoin an equation to the system (A.4):

\[
\begin{align*}
  f(x, y) &= 0 \\
  g(x, y) &= 0 \\
  u - (x - \alpha)(y - \beta) &= 0
\end{align*}
\] (A.5)

From this system, \(x\) and \(y\) are eliminated, leaving a single polynomial in \(u\), the volume function. The elimination can be accomplished using resultants. This elimination problem is discussed in Keyser et al. [61].

To compute the number of roots in the interval \([x_0, x_1] \times [y_0, y_1]\), four volume functions are constructed, with \((\alpha, \beta)\) taking on the values at the four box corners. Associated with each corner is the number \(N(\alpha, \beta)\) of negative roots of that corner’s volume function. The number of negative roots of \(V(u)\) is found with univariate Sturm sequences or by other means. A certain linear combination of the four values of \(N(\alpha, \beta)\) gives the number of roots inside the box. See figure A.2. The numbers \(N(\alpha, \beta)\) at the box corners are the multivariate analogues of the number of sign variations at the interval endpoints in univariate Sturm theory.

In three dimensions, the volume function \(V(u)\) is obtained by eliminating \(x, y,\) and \(z\) from the system

\[
\begin{align*}
  f(x, y, z) &= 0 \\
  g(x, y, z) &= 0 \\
  h(x, y, z) &= 0 \\
  u - (x - \alpha)(y - \beta)(z - \gamma) &= 0
\end{align*}
\] (A.6)

The volume function is computed at the eight corners of a 3-D interval. The elimination can be accomplished with the Macaulay resultant.

Almost always, distinct roots of the multivariate system map to distinct roots of the volume function. Possibly, though, two distinct multivariate roots have the same signed volume. In this case, the volume function has multiple roots. The Sturm sequence algorithms do not work in the presence of multiple roots, but the situation is easily detected. The solution is to move the point \((\alpha, \beta)\), which is always somewhat arbitrary in the context of this dissertation. (The two typical cases are that the
Figure A.1: The two-dimensional volume function $V(u)$ associated with the point $(\alpha, \beta)$ at the crosshairs has roots at $u = -1, 8,$ and $9$. It may have complex roots as well.

\[ \frac{1}{2}(\text{pattern} + \text{pattern} - \text{pattern} - \text{pattern}) = \text{pattern} \]

Figure A.2: Bivariate Sturm sequences via the volume function.
multidimensional interval is either a bounding box for the entire polyhedron or a small containing interval for a floating-point root estimate.)

A.4 References

Univariate Sturm sequences, the subresultant algorithm, and Sylvester resultants are described in texts on algorithmic algebra such as Davenport et al. [23]. The Macaulay resultant is often mentioned, but the only published modern source describing it in detail is the article of Canny [14]. For this reason, I have attempted to give enough detail here to allow the reader to implement the algorithm. General techniques for multivariate root isolation are given by Pedersen [78] and Milne [71]. Keyser et al. [61] give an efficient implementation in two dimensions.
Bibliography


