COMPUTATIONAL GEOMETRY WITH IMPRECISE DATA AND ARITHMETIC

C. Bradford Barber

CS-TR-377-92

October 1992

COMPUTATIONAL GEOMETRY WITH IMPRECISE DATA AND ARITHMETIC

C. Bradford Barber

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

October 1992

Abstract

This thesis presents an algorithm for computing the 3-d convex hull of imprecise points using floating point arithmetic. The algorithm produces a set of "thick" facets that contain all exact convex hulls of the data. It is based on Quickhull and Beneath-beyond. Parameters for the algorithm include the precision of the data and the maximum angle between adjacent facets. This allows the user to simplify the output and may reduce the exponential growth in output size as dimension increases. It is the first 3-d convex hull algorithm to work with fixed precision arithmetic. We derive a bound for the maximum width of a facet when certain restrictions are satisfied.

The algorithm produces a data structure called a *locally convex box complex.* Similar to a simplicial complex, a *box complex* is a graded DAG of sets called *boxes.* Each node of the DAG is a *face* of the box complex. A face represents a vertex, edge, facet, or other feature. Its box bounds the possible locations of the face. A box complex is *locally convex* when hyperplanes define the boxes for facets, and the hyperplanes for adjacent facets meet in a convex angle.

The thesis also presents an algorithm for point inclusion in box complexes and polyhedra. It identifies a subset of the vertices. The subset has odd parity if the point is inside and even parity if it is outside. It is the first point-in-polyhedron algorithm to work in general dimension on arbitrary inputs with fixed precision arithmetic. It is the first point-in-polyhedron algorithm to work when the only geometric information is bounding boxes.

Thesis Advisor: Professor David Dobkin

Acknowledgments

This section is a treat. No theorems to prove; no transitions to write; no program to debug. Just a chance to thank everyone who helped make these four years enjoyable and productive.

My thesis advisor, David Dobkin, kept with me through several changes of topic and many drafts. He never failed to understand what I was trying to say. He knew what a thesis should be, and kept me moving in the right direction. My other advisors, Hector Garcia-Molina and Raphael Alonso, got me going and helped me try out new areas. I was supported in part by the National Science Foundation under Grant Number CCR90-02352 and under a subcontract from the National Science and Technology Research Center for Computation and Visualization of Geometric Structures at the University of Minnesota.

My collaborator, Michael Hirsch, made box complexes real and academic work fun. He took an ad hoc creation called d-surfaces and turned it into a mathematical structure. Our many discussions have been a delight. I am indebted to Miller Maley who started me on this road. Steven Fortune's invitation for a talk at Bell Labs made it all worth well. Albert Marden's invitation for a postdoc at the Geometry Center promises an exciting future.

My Master's thesis advisor, Dennis Shasha (aka Jacob Ecco), guided my first efforts in academia. His seminar and continual help made my experience here at Princeton possible. My thanks to Dennis and all of the professors at the Courant Institute of New York University. Their courses repaired my atrophied "schooling" neurons and gave me a formal education in Computer Science. Special thanks to Michael Overton who introduced me to numeric methods.

My readers, Pat Hanrahan and Bernard Chazelle, performed a much needed

review of the thesis. Pat saved us from bounding boxes (now called "boxes") and showed me how great courses can be taught. Bernard introduced me to the Quickhull algorithm, computational geometry, and the subfield of numeric error handling. Bernard, with the assistance from Melissa, Ayellet, and David, saved us from boring videotapes. Jin-Yi Cai gave me a model example of mathematical proof. Eric Ristad gave me the opportunity to study Bernstein's work. Steve, Kennedy, Pat, Jim, and Ed saved me from death by UNIX. Melissa, Dot, Sharon, Ruth, Grace, Vernon, Gene, Trish, Ginny, Becci, Winnie, Claire, and Geree saved me from death by administration. The new building made day-to-day life comfortable and enjoyable (thanks Charlotte and all).

The students of Pat Hanrahan's and Andrew Appel's CS217 enriched my life. With luck, memories of marked up programs will remind them that the purpose of a program is communication, not code. With the help of Sally McKee, Norman Ramsey, John Danskin, Carrie Beam and Chris Williams, I survived as a preceptor.

Eleftherios Koutsofios wrote Cheyenne, the source of hundreds of pictures. Susan Dorward wrote hullio, a whole lot faster than Quick_hull. Jim Plank wrote jgraph (my very first graph was good, and my last graph was exactly what I wanted). Sigal Ar got me through Tarjan's course. Matt Blaze created pbj. Heather Booth taught gardening. Adam ("Schwarzenegger") Buschsbaum managed perfect summer evenings on the ballfield, and made our year one of the best.

My second education at Princeton expanded my understanding of the world. Saul Kripke taught philosophy of number. Stephen Cohen taught Soviet politics. Bastiaan von Fraassen taught metaphysics. Sara McLanahan taught poverty issues and public policy. James Higginbotham taught philosophy of language. Richard Curtis taught animal tracking. Michael Doyle taught international politics and introduced me to Thucydides. Hugo Sonnenschein, Paul Benacerraf, and Richard Spies taught practical budget making. My thanks to all.

I thank my housemates Trace Jordan, Ying Hu, Tamzin Anderson, Yi Hong Chen, Bill Dorland, Mark Kuo, and David Bluestein for making Magie 5H a fun place to live. I thank Bill, Judy, Blake, Hu Ying, Matt, Nurit, Trace, Dolares, Katrien, George, and Kuan for late nights on Entry 27. I thank Ksenia Bobylak for

the opera. I thank my fellow graduate students and postdocs for long dinner conversations, great parties, dancing, rowing, canoeing, hiking, and trips to Vermont, the Pine Barrens, Delaware Water Gap, New York City, and Quebec.

My parents, Charles and Lois Barber, have encouraged my adventures from the day I was born. My father showed me how to treat others and conveyed his enjoyment of the out-of-doors. My mother showed me how to throw a party and how to find the creative solution. My sisters, Ann, Robin and Betsy, are great friends of mine. My grandmother, Helen Cope, gave me the gumption to pursue life. My aunt, Esther Hymer, inspired me.

Thank you family. Thank you Princeton.

C. Bradford Barber
Princeton, New Jersey
June 1992

Love truth but pardon error
Chinese fortune cookie, 1991

Dedicated to Helen Gibson Cope

1889 — 1992

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In computer-aided design, a designer works in front of a computer terminal. The computer can display architectural drawings, a schematic model, or a solid model. It acts as an electronic drawing tablet or modeling medium. For example, it can display a model of a car and a mannequin sitting in the front seat. Besides replacing drafting tables and clay models, it can compute properties of the representation. For instance, "Does the mannequin fit between the front seat and the steering wheel?" With a real model, the question is easily answered. With a computer, the question must be transformed into a computation. Geometrically, the question is equivalent to asking if the mannequin and steering wheel intersect. The computer-aided design system can test if the intersection of the steering wheel and mannequin is empty.

The field of computational geometry studies the solution of geometric problems by computers. Computational geometry has traditionally dealt with precise sets of points, e.g., an edge, a line, or a polygon. But real world data is imprecise. One says the temperature of a room is 68 degrees; one does not say the temperature is the real number $68.34216431877\ldots$ or even say 68.32 ± 0.03. Most data is only known to some small number of digits. Data may also be imprecise because of representation or computation. For example, the number $1/3$ does not have a finite decimal representation. Also, roundoff error makes computed values imprecise, e.g., roundoff error can shift the intersection of two nearly parallel lines.

With imprecise data, simple questions may not have an answer. For example,

asking "Is point A above or below line L?" is different than asking "Is imprecise point A above or below line L?". In the former case, A is either above, below, or exactly on L. In the later, A has a region of possible locations, e.g., a ball. If the region for A is clearly above or clearly below L, then A is above or below L. Otherwise the relationship between A and L is ambiguous. The ambiguity occurs when the maximum error due to imprecision is greater than the distance from the point to the line.

Error analysis determines the maximum error due to imprecision. It is an important part of scientific experimentation and numeric methods. It is largely ignored in computational geometry. Most algorithms for computational geometry are defined for points with real or rational coordinates. Furthermore, to simplify the presentation of an algorithm, researchers usually assume, for example, that points are not coplanar, lines are not vertical, and four planes do not meet at a point. These relationships are called *singularities* or *violations of general position.* Theoretically, the assumptions are justified because the domain of geometry is point sets, singularities cause "inconsequential but lengthy details," singularities do not occur with random data, and singularities can be removed by symbolic perturbations [Edelsbrunner & Mücke 1988; Yap 1990; Emiris & Canny 1991].

These assumptions fail when a programmer implements a geometric algorithm with floating point arithmetic. Floating point hardware fixes the number of digits; floating point arithmetic causes roundoff error; and singularities may occur. Ad hoc solutions do not always work.

This thesis proposes imprecise data as an alternative domain for geometric algorithms. Imprecise data is represented by a set of possible locations, e.g., all edges from the origin to the point (1,0) that are within 0.1 of the x axis. Imprecise data has the useful property that an algorithm defined for imprecise data works correctly when implemented with floating point arithmetic.

1. Point-in-polyhedron and convex hull

The thesis develops two algorithms: point-in-polyhedron and convex hull. When defined on precise points, both problems have a long mathematical history.

Point-in-polyhedron reports whether a point is inside or outside a bounded polyhedron. Point inclusion goes back at least to Jordan [1887]. He initiated a proof that every simple closed curve separates the plane into two pieces: inside and outside. A corollary is the *parity test*: A point is inside a simple, closed curve if and only if a ray crosses the curve an odd number of times. Most algorithms for point-in-polyhedron use a generalization of the parity test.

The *convex hull* of a set of points is the smallest convex set containing the points. Informally, the convex hull is a flat surface that wraps around the points. In the 1720s, Newton used the convex hull to determine the behavior of a real algebraic plane curve near a singular point. In 1752, Euler proved his famous equation: $V - E + F = 2$. In 1970, McMullen proved the upper bound conjecture about the number of faces in a convex polytope. [Berger 1990; McMullen & Shephard 1971]

There are many algorithms for computing the convex hull of a set of points. The first 2-d algorithm was by Graham [1972]. In 3-d, the gift-wrapping algorithm [Chand & Kapur 1970] starts with a point know to be on the convex hull (e.g., the point with the minimum x-coordinate). The algorithm locates two other points that define a supporting plane (all points must be on one side). Then it repeatedly selects a free edge and locates a third point that defines a supporting plane. In this way, it wraps a set of facets around the point set.

The convex hull is unique if data is precise and points are in general position. In 3-d under these conditions, all facets of the convex hull are triangles.

2. Imprecision

This thesis concerns three problems of computational geometry: singularities, fixed precision arithmetic, and imprecise data. As shown in this section, imprecise data subsumes the other two. Imprecise geometric data consists of geometric features, e.g., points, edges, and polygons. Their topology is the same as the topology of precise geometric features. The geometry differs. The geometry of a precise or imprecise feature is the set of points assigned to the feature. With precise features, the geometric dimension is the same as the topological dimension. For example, an edge is a subset of a line and a polygon is a subset of a plane. With imprecise

features, the geometric dimension is no less than the topological dimension.

A ball or cube are examples of imprecise vertices. A cylinder or rectangular solid are examples of imprecise edges. Their geometries delimit the possible locations of a feature. So the actual vertex is anywhere within the ball or cube, and the actual edge is anywhere within the cylinder or rectangular solid. The geometry of an imprecise feature is an approximation. The only requirement is that the feature can not be outside of the specified region.

The geometry of an imprecise feature can always be widened as long as it is still contained in the geometry of incident, higher-dimension features. This is how we handle imprecision. We specify the geometry with an easily manipulated shape. The three shapes we use in this thesis are bounding boxes (maximum and minimum coordinates for the feature), δ-boxes (the δ-region around a precise feature), and polytopes (the bounded intersection of half-spaces). A *box* is the shape associated to a feature. A box is large enough to contain any imprecision due to the input. So if a point's coordinates are known to an accuracy of 0.5, the point's box is at least 1.0 wide.

A geometric algorithm uses computation to determine the relationship between features. Most implementations of geometric algorithms use fixed precision arithmetic that causes roundoff error. We compute the maximum roundoff error for a computation. We account for the roundoff error by widening the feature's box by the maximum roundoff error. In this way, imprecise data subsumes fixed precision arithmetic.

A geometric algorithm may use computation to create new features. As with input data and roundoff error, we make the new feature's box wide enough to account for imprecision errors. Note that a new feature's box contains the boxes of subordinate features. For example, the box for an edge contains the boxes for the edge's endpoints.

Imprecise data causes a fundamental change in geometric relationships. This is illustrated by the algorithms in this thesis. The underlying problem is that a geometric relationship may be ambiguous. For example in point-in-polyhedron, the point may be within roundoff error of one of the polyhedron's boxes. It could be

on the surface of the polyhedron, but it also it could be inside the polyhedron or outside the polyhedron. We call this ambiguous relationship "*can't tell*".

The possibility of a *can't tell* relationship effects the design of an algorithm as well as its output. For point-in-polyhedron, we shot a ray from the test point. If the test ray intersects the polyhedron near more than one facet, we can not tell which facet it crosses. Similarly with the gift-wrapping algorithm for convex hull, we can not always tell if adjacent facets form convex angles.

With precise features, the analogue of a *can't tell* relationship is a singularity. Examples of singularities are "coplanar", "equals", and "vertical". A *singularity* (or *degeneracy*) is a geometric relationship that is undone by an infinitesimal perturbation.

Existence of a singularity is often equivalent to a computation equaling zero. If so, the singular points for a relationship defines an implicit surface. With a random distribution, the probability of a point being on a surface is zero. So singularities are also called *measure zero events.*

If singularities do not occur for a set of points, then the points are in *general position.* General position is usually assumed to simplify geometric algorithms. As discussed above, a researcher is justified in ignoring singularities. Unfortunately, the theoretical assumptions may be violated when implementing a geometric algorithm. Numbers are represented by a fixed number of digits, floating point arithmetic causes roundoff error, and inputs come from non-random distributions. Yap presents the dilemma:

> Degeneracy in computational geometry is a general phenomenon. So in what sense can we justify its neglect in theoretical algorithms? One justification is that explicit handling of degeneracies obscures the centrality of the non-degenerate cases: degeneracies normally involve an overwhelming number of cases that are disproportionate to their likelihood of occurrence. But an implementor of these algorithms must handle the degeneracies when they do arise. [Yap 1990, p. 350]

Similar dilemmas occur wherever real numbers are used, e.g., numeric methods, computer graphics, computational geometry, and scientific programming. For

example, a ray tracing algorithm computes the reflection of a ray at a surface. With floating point arithmetic, the computed intersection of the ray with the surface could actually be below the surface. An implementor must detect this case, otherwise the ray will be reflected back to the original direction when it re-intersects the surface.

Imprecise data subsumes singularities when the boxes for imprecise features are open sets. Then any feature can be perturbed infinitesimally, and all potential singular relationships are *can't tell* relationships.

3. Previous work

The effects of fixed precision arithmetic and singularities are active topics in computational geometry. There are three main approaches: simulating infinite precision, error analysis, and epsilon values. Other approaches include symbolic reasoning and perturbations.

3.1. Infinite precision. There are several ways to simulate infinite precision: rational arithmetic, symbolic perturbations, and variable precision arithmetic. With rational arithmetic, the goal is to limit the maximum number of digits required [cf., Sugihara 1989]. For example, Milenkovic [1989b] computes an arrangement of pseudo-linear lines with $2n+1$ bit arithmetic where n is the input precision. With rational arithmetic, implementations can accurately detect singularities.

With rational arithmetic and symbolic perturbation, singularities of a computation can be consistently removed [Edelsbrunner & Mücke 1988; Yap 1990; Emiris & Canny 1991] The method is called *Simulation of Simplicity.* It evaluates a sequence of determinates with rational arithmetic. The first non-zero determinate determines the computed relationship. The sequence is defined so that a singularity can not occur.

With variable precision arithmetic, singular results can be resolved at higher precision [Dobkin & Silver 1988; Dobkin & Silver 1990; Karasick et al 1990]. Least-significant-bit-accuracy computation can also be used [Kulisch & Miranker 1983; Ottmann et al. 1988].

The problems with simulating infinite precision are:

- Input data is seldom precise, especially if produced by measurement or computation.

- Multiplication and other operations increase the required number of significant bits. For example, intersecting line segments with rational endpoints can increase the bit requirement by eight-fold [Fortune 1992a].

- Symbolic perturbation does not control the direction of perturbation nor does it limit the creation of very small features. For example, Simulation of Simplicity computes a non-degenerate convex hull from multiple copies of the same point.

- Even with least-significant-bit accuracy, singularities are not transitive. For example even if edges A and C are computed to be coplanar with edge B, edges A and C may be convex. The lose of transitivity allows inconsistent results that can invalidate an algorithm.

- Computer hardware usually operates on fixed precision numbers. Higher precision requires slower, software implementations.

3.2. Error analysis. In numeric methods, inverse error analysis is a successful approach to fixed precision arithmetic [e.g., Golub & van Loan 1983]. A computation using fixed precision arithmetic produces a somewhat erroneous result. That result may be the exact result of precise arithmetic on a different input. The computation is *stable* if the input yielding the exact result is a small perturbation of the actual input. Inverse error analysis bounds the perturbation with some function of the input data and precision of the arithmetic.

Fortune [1989] uses inverse error analysis for a 2-d convex hull algorithm. He proves that his algorithm computes a convex hull that is the exact convex hull for a small perturbation of the input. Milenkovic and Fortune [Milenkovic 1988; Fortune & Milenkovic 1991] produce an arrangement that is the exact arrangement

for pseudo-linear lines. Greene and Yao [1986] produces an arrangement for a perturbation of the input lines that only intersects at lattice points.

A problem with inverse error analysis is that it does not guarantee properties that are useful for later computations. As contrast, Li and Milenkovic [1990] and Guibas, Salesin and Stolfi [1990] guarantee that their 2-d convex hulls remain convex despite perturbations of the vertices.

Forward error analysis computes the maximum roundoff error for a computation. Guibas et al [1989] partitions the result of a computation into three regions: positive, negative, and within roundoff error of zero. They propose an *ϵ-geometry.* Interval arithmetic is an automatic method to bound roundoff error [Moore 1979]; each operation expands the interval to include roundoff error. If intervals overlap, the computation can be done at higher precision [Kao & Knott 1990], the problem can be sub-divided [Mudur & Koparkar 1984], or features can be merged [Segal & Sequin 1988]. More sophisticated methods include statistical inference of roundoff error [Vignes 1988], automatic computation of partial derivatives [Iri 1984], and automatic search for computational instabilities [Miller 1975].

A problem with forward error analysis is that the computed maximum error may be overly pessimistic. If the intervals become too wide, relationships are lost. For example Segal's algorithm for combining polyhedra can reduce the union of polyhedra to a single point [Segal 1990].

Hopcroft and Kahn [1989], and Hoffman et al [1989; Hoffman 1989a; Hoffman 1989b] use symbolic reasoning to determine relationships when fixed precision arithmetic yields ambiguous results. They call their approach the *reasoning paradigm.* A problem with the reasoning paradigm is that it may be intractable without additional assumptions. For example, their intersection algorithms require feature separation for the input polyhedra.

Random perturbations of the input may reduce the chance of singularities. For example, Corkum and Wyllie [1990] randomly rotate a polyhedron before testing the crossing parity of a ray. With ill luck, the algorithm must be restarted on a new rotation.

3.3. Epsilon values. Many geometric algorithms do not work with fixed precision arithmetic. A common method of implementing these algorithms is to define a small constant that is typically named "EPS" or "EPSILON". The constant EPS determines when two numbers are considered equal or a result is considered zero. When a predicate is nearly zero, the predicate is treated is if it were exactly zero. This is the same as treating nearly singular relationships as if they were singular.

Using a constant EPS may delay the ill-effects of fixed precision arithmetic but does not solve the problem of imprecision. Some of the undesirable results are:

- Broken topology and delayed blowup. Two computations may report inconsistent topological relationships for geometric data. The inconsistency can propagate through the program until it causes a hard error such as a core dump. The hard error may occur long after the inconsistent computations were made.

- Micro-features. The output may contain undesired features that are smaller than the precision of the input data.

- Scale limitations. As the input size increases, the implementor may fail to find a successful value for the constant EPS.

3.4. Summary. Fixed precision arithmetic adds thickness to an implicit surface. If a point is too close to the surface, a computation may incorrectly report that it is above, below or on the surface. If several computations return inconsistent results, the algorithm can fail catastrophically. Proposed solutions include simulating infinite precision, inverse error analysis, forward error analysis, symbolic reasoning, and perturbations. These solutions may require software implementation of arithmetic, produce results with unknown or undesirable properties, assume preconditions of the input, or restart the algorithm on detecting a singularity. In practice, a constant EPS is picked and fingers crossed.

If singularities are rare events, inconsistencies are unlikely to occur. With moderate data sets and low dimensions, such an assumption is reasonable and the effect of an occasional singularity may be unimportant. Problems occur when singularities become common events. This can happen with large data sets or large outputs.

4. Contributions of this thesis

This thesis looks at two different kinds of algorithms. Point-in-polyhedron takes a geometric structure and returns an *inside/outside* answer. Convex hull takes a set of points and returns a geometric structure.

The algorithm for convex hull returns a data structure of imprecise facets and lower dimensional features. The facets are wide enough to contain the boundary of all possible exact convex hulls of the points. Adjacent facets form a convex angle and satisfy a maximum angle constraint. Another algorithm can use the data structure to answer queries about the convex hull of the point set. It can rely on convex angles between adjacent facets.

The maximum width of a facet of the convex hull depends on the input precision, roundoff error, and convexity constraints. Improving these parameters, reduces the maximum width. In the limit, the convex hull of imprecise points is the same as the convex hull of precise points.

Imprecise data has several advantages. The most important is that algorithms work on existing hardware for all inputs. Though several researchers have considered fixed precision arithmetic, few allow imprecise data [an exception is Hopcroft & Kahn 1992].

Imprecise data is an approximation to a precise structure. Often, the precise structure is not needed or undesirable. For example, the precise structure could contain small features that are irrelevant to the result. In high dimensions, the size of a precise structure can grow exponentially. For example the convex hull of 5000 points in $\mathbf{R}^5$ can use 15 megabytes of storage [Fortune 1992a]. With imprecise data, the number of features is bounded by the precision available instead of the input size. By reducing the precision, the output size is reduced.

There are two complementary approaches to imprecision. One is to minimize the effect of imprecision and the other is to bound its effect. We emphasize bounded error.

The contributions of this thesis are:

- A practical approach to computational geometry based on imprecise data. Imprecise data subsumes fixed precision arithmetic and singularities. Furthermore, it reduces the size of the output.

- Algorithms for point-in-polyhedron and convex hull on imprecise data. Both algorithms are implemented in 2-d and 3-d. The convex hull algorithm (Quick_hull) is the first 3-d, convex hull algorithm to work with fixed precision arithmetic. The point-in-polyhedron algorithm (Point_in_polyhedron) is the first one to work in general dimension with fixed precision arithmetic.

- A mathematical structure, the *box complex*, for modeling imprecise data. A box complex has a simple data structure. This data structure is the input of Point_in_polyhedron and the output of Quick_hull. Each facet and lower dimensional feature of a box complex is a full-dimensional set in R^d.

- A correctness proof for Point_in_polyhedron.

- Specializations of the box complex: the *locally convex box complex* and the *convex box complex*. A box complex is *locally convex* if all neighboring facets form *clearly convex* angles. It is a *convex box complex* for a set of imprecise points if it contains all exact convex hulls of the points.

- A correctness proof in general dimension that Quick_hull produces an convex box complex. If a set of balance conditions are satisfied, Quick_hull runs in $O(n \log h + h)$ where n is the size of the input and h is the size of the output.

- An error analysis for Quick_hull in 2-d that bounds the maximum width of a facet. An error analysis for Quick_hull in 3-d that bounds the maximum width of a facet when certain restrictions are satisfied.

5. Thesis outline

Chapter 2 discusses imprecision, convex hulls, and box complexes. A box complex is a graded DAG of sets of points called *boxes*. Its *trace* is the union of its boxes. An algorithm on box complexes works irrespective of the precision used for data or arithmetic. Chapter 2 concludes with a set of computations and operations that are useful for imprecise data and arithmetic.

Chapter 3 describes a point-in-polyhedron algorithm based on the parity test. It works independently of the precision of the arithmetic. In three and higher dimensions, the algorithm is the first to use comparisons alone when the point is away from the surface. It is also interesting because the algorithm never determines if a test ray crosses a facet. Instead it determines when a test ray can not cross a facet, and uses this information to reduce point inclusion to the parity of a subset of the vertices.

Chapter 4 introduces the convex hull problem and an algorithm for precise data and arithmetic. The algorithm combines the Quickhull algorithm [Eddy 1977 and others] with beneath-beyond [Kallay 1981]. We introduce a simple data structure for general dimension convex hulls, and discuss the limitations of this and other algorithms under imprecise data and arithmetic.

Chapter 5 adapts the precise convex hull algorithm to imprecise data and arithmetic. The resulting algorithm merges coplanar and concave facets.

Chapter 6 analyzes the maximum width of a facet in Quick_hull. The first half determines a 2-d bound while the second half determines a 3-d bound under an input restriction.

Chapter 7 discusses the 2-d and 3-d implementation of Quick_hull and shows pictures of its output. The implementation allows the user to explore all aspects of the algorithm. For example, the user can watch a slow movie of Quick_hull with one frame for each event in a window. With imprecise data and arithmetic, the maximum size of the output is bounded by the parameters and dimension. For fixed parameters and dimension, the empirical performance is $O(n)$.

Chapter 8 concludes the thesis with a discussion of Quick_hull and box complexes. Quick_hull is designed for general dimension. Using the correspondence

between convex hull and Delaunay triangulations [Brown 1979], Quick_hull can be adapted to Voronoi diagrams. Similar correspondences make it suitable for other applications such as power diagrams [Aurenhammer 1991]. Issues to be resolved include: defining imprecise Voronoi diagrams, building a geometry of locally convex box complexes, merging redundant ridges, constructing new hyperplanes for facets in 4-d, and visualizing 4-d structures.

Appendix A gives the proof of correctness for Point_in_polyhedron. We first define the "realization" of a box complex and use realizations and homology theory to prove a separation theorem for box complexes. Then we prove the reduction theorem that justifies Point_in_polyhedron.

Chapter 2

Imprecise structures and computations

This chapter defines imprecise data, imprecise predicates, box complexes, convex box complexes, and locally convex box complexes. The chapter concludes with the basic computations and operations that are used in this thesis.

1. Imprecise data

"The objects considered in Computational Geometry are normally sets of points in Euclidean space" [Preparata & Shamos 1985]. This is also the domain for Euclidean geometry. The introduction discussed reasons why sets of points may be inappropriate for geometric algorithms. The underlying reason is that programmers implement geometric algorithms on computers, and computers can only represent a measure zero subset of all possible point sets. This is one source of imprecision. Two other sources are roundoff error due to floating point arithmetic, and input error due to measurement.

Precise data is data consisting of real or rational numbers. *Imprecise data* is data consisting of approximations to geometric features. The approximations are due to finite representation, roundoff error, or measurement. Imprecise data is traditionally defined as a probability distribution with a mean value and confidence interval, e.g., 3.24 ± 0.07. Statistical analysis determines the confidence interval of

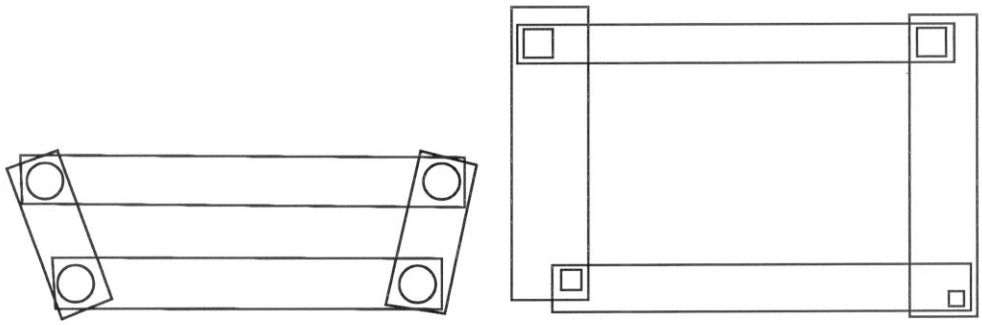


Figure 2.1: Two sets of boxes for an imprecise polygon. A box delimits the possible locations for a face. The left-hand side consists of δ-boxes and the right-hand side consists of bounding boxes. Note that a box contains all subordinate boxes.

measured data. The confidence interval of a derived value is computed from the original confidence intervals [e.g., Taylor 1982]. A simpler approach records an interval of possible values for each quantity. Interval arithmetic defines addition, subtraction, multiplication, and division on intervals [Moore 1979].

Imprecise geometric features are harder to define. The Cartesian product of intervals defines a bounding box in $\mathbf{R}^d$. For example in $\mathbf{R}^2$, a rectangle could delimit the maximum roundoff error for each coordinate of the intersection of two lines. If the lines are nearly parallel, roundoff error can be large. If both lines are almost 45% to the axes, the bounding box is much larger than the region of possible intersections.

A probability distribution for a geometric feature would be more accurate, but it is harder to compute. For example, if the location of a point has a normal distribution, the edge between two points would have a dumbbell shaped distribution around a line segment. The width of the dumbbell depends on the likelihood of extreme values for the points.

We adopt an intermediate representation. An imprecise feature is represented by an open set of points in Euclidean space. The set of points, the feature's *box*, delimits the location of the feature. A box can be any shape as long as the feature can not be outside of the box. Figure 2.1 shows some examples.

An imprecise object is a collection of features (called *faces*) with their boxes

and subordinate features. This collection is called a *box complex.* Informally, a box complex is like a polyhedron or simplicial complex with open sets instead of exactly defined vertices, edges, facets, etc. Its topology is the same as the topology of the corresponding polyhedron or simplicial complex. A later section gives a formal definition.

2. Imprecise predicates

A geometric predicate performs a calculation on point coordinates and returns a *yes/no* answer. For example, consider the line $\{x \in \mathbf{R}^d \mid \langle \hat{h}, x \rangle = h_{d+1}\}$ where $\hat{h}$ is the line's normal and h_{d+1} is the line's offset. A point P is above the line if $\langle \hat{h}, P \rangle > h_{d+1}$.

With imprecise data, a simple *yes/no* answer is not always possible. Consider asking if an imprecise point is above a precise line. Let the box for the imprecise point be a circle. The actual point is somewhere inside the circle. If the line intersects the circle, the actual point could be above, below, or on the line. Following Fortune [1992a], the predicate returns a *can't tell* answer.

Fixed precision arithmetic complicates the picture. *Fixed precision arithmetic* is arithmetic on fixed precision, floating point numbers. Each operation can cause roundoff error. If a point is too close to a line, roundoff error may change a *yes* into a *no* or vice versa. We account for roundoff error by expanding a box by the maximum roundoff error. Then if the expanded, imprecise point is above a line by computation, the imprecise point must be above the line. The last section of this chapter computes the maximum roundoff error for each predicate used in this thesis. In this way, fixed precision arithmetic is subsumed by imprecise data.

An *imprecise predicate* is a geometric predicate on imprecise data using fixed precision arithmetic. Its *maximum error* is the sum of the maximum error due to roundoff and the maximum error due to precision. Imprecise predicates are split into two forms: strong and weak predicates. A *strong predicate* is true for all possible exact features and roundoff errors. A *weak predicate* is true for at least one possible feature and roundoff error. The negation of a strong predicate is a weak

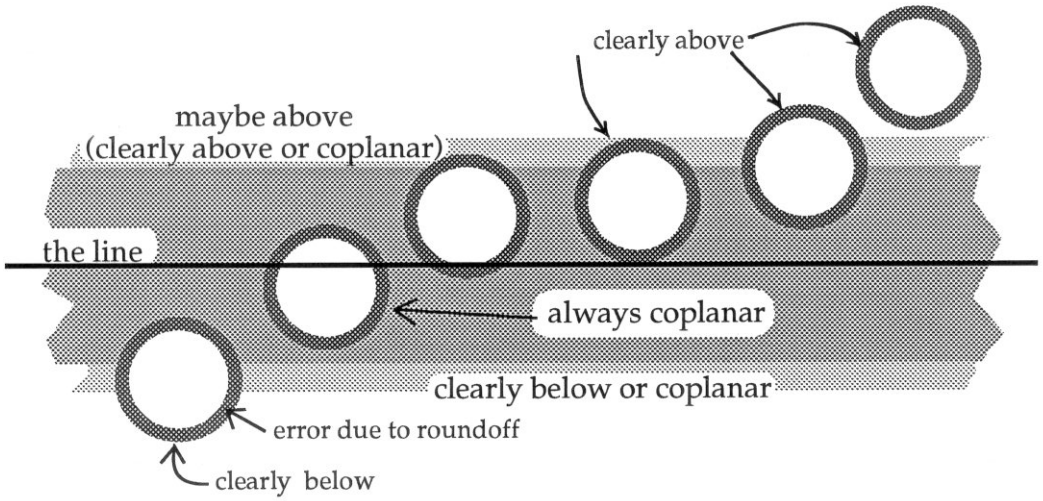


Figure 2.2: *Clearly above* and *clearly below* line for an imprecise point and a precise line. *Clearly above* and *clearly below* line are strong predicates. *Coplanar is* a weak predicate.

predicate and vice versa. To distinguish weak predicates from negated strong predicates, strong predicates are always preceded by *clearly* or *not* and the negation of a strong predicate is always preceded by *maybe*. So *clearly above* and *not infront* are strong predicates; their negations are *maybe below* and *maybe infront* respectively (Figure 2.2).

Weak and strong predicates are the same as Guibas et al's [1989] ϵ-predicates an $(-\epsilon)$-predicates respectively. We use different terminology because our boxes can be arbitrarily shaped, Also, Guibas et al. emphasize weak predicates while we emphasize strong predicates. Fortune [1988] makes extensive use of ϵ-predicates to construct approximate 2-d convex hulls and triangulations. Segal [1990] defines tolerance regions to determine weak predicates.

Li and Milenkovic [1990] and Guibas et al. [1990] use weak and strong predicates with pre-computed epsilons. They describe algorithms for a 2-d ϵ-strongly convex δ-hull. Their output remains convex despite ϵ perturbations in vertex coordinates, and points are never further than δ from the hull's interior. Our approach has similar properties.

Our approach is to:

- Assume imprecise data. Roundoff error is handled by increasing the imprecision of the data. The maximum error is computed.

- Design a mathematical construct for imprecise data, the box complex, that is suitable for implementation on a computer. The next section defines box complexes. Algorithms on box complexes work with whatever precision is available.

- Describe the output with strong predicates so that later programs can depend on the results. Li & Milenkovic and Guibas et al make similar claims.

- Seek practical solutions, since precision problems are practical problems.

The box complex is the key to this approach. It is a simple, flexible structure that capture the notions of arbitrarily imprecise data and arithmetic. Designing an algorithm for box complexes is more difficult than designing an algorithm for points in Euclidean space. But an algorithm using box complexes will work with arbitrarily imprecise data and arithmetic.

3. Box complexes

A box complex is an approximation to a surface in $\mathbf{R}^d$ [1]. A box complex can represent polyhedra as well as rather wild structures that self-intersect and overlap in all dimensional components. The topology of a box complex is a slight generalization of the topology of a simplicial complex or subdivision of a manifold. The main contribution of the box complex is its geometry. Instead of a precisely defined geometry, a box complex delimits the possible geometries with boxes.

Before defining box complexes, we review some of the mathematics used in this thesis.

[1] The box complex and related proofs are joint work with Michael Hirsch.

3.1. Mathematics of convex hulls. This section is a quick review of convexity, hyperplanes, convex hulls, and simplicial complexes. The next section defines the box complex.

We adapt the terminology of geometers and topologists [Brondsted 1983; Grünbaum 1967; McMullen & Shephard 1971; Munkres 1984]. A set of points $\{x_i\}$ is *linearly independent* if the points only satisfy trivial relations of the form $\sum_i \lambda_i x_i = 0$. The *affine combination* of a set of points $\{x_i\}$ is the set

$$\left\{ x \mid x = \sum_i \lambda_i x_i, \quad \sum_i \lambda_i = 1 \right\}$$

A *convex combination* is an affine combination with each $\lambda_i > 0$. A *convex (*resp. *affine) set* is a set of points that is closed under convex (resp. affine) combinations. An affine set of $d+1$ independent points is a *d-flat.* The *convex (*resp. *affine) hull* is the smallest convex (resp. affine) set containing a set of points.

A *hyperplane* of $\mathbf{R}^d$ is the affine hull of d independent points. A *halfspace* of $\mathbf{R}^d$ is an unbounded, closed set whose boundary is a hyperplane. An outward pointing normal equation defines a hyperplane and a halfspace. A point is *above* a hyperplane if the inner product of the point's coordinates and the hyperplane's normal is greater than the hyperplane's offset. A point is *coplanar* with the hyperplane if the inner product is zero. Otherwise it is *below* the hyperplane.

The convex hull of a finite set of points is a *polytope.* A polytope is the bounded, finite intersection of closed halfspaces. A vertex of the convex hull is called an *extreme point* of the point set.

A *supporting hyperplane* for a set of points intersects the boundary of the convex hull of a set but not its interior. A *face* of a convex hull is the intersection of a supporting hyperplane with the convex hull. All faces are convex polytopes.

A *d-simplex* is the convex hull of $d+1$ independent points. A 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron. A *face* of a d-simplex is the convex hull of a subset of its points. A simplex is also the convex hull of a $(d-1)$-face (its *base*) and a point (its *apex*). In $\mathbf{R}^d$, a *facet* is a $(d-1)$-face and a *ridge* is a $(d-2)$-face. If $d \leq n^{1/2+\xi} - 1 (\xi \approx 0)$, the maximum number of facets and ridges in a d-polytope of n vertices is $O(n^{\lfloor d/2 \rfloor})$ [Klee 1966].

A *d-dimensional simplicial complex*, or *d-complex*, is a set of i-simplices ($0 \leq i \leq d$) where every simplex is a face of some d-simplex. A d-complex includes all faces. The intersection of any two i-simplices of a d-complex is a common face (if any). An *abstract simplicial complex* is a finite collection of non-empty sets that includes all non-empty subsets [i.e., an abstract simplicial complex is a simplicial complex without geometric information].

A *polyhedron* in $\mathbf{R}^3$ is a finite set of plane polygons such that each edge is shared by exactly two polygons, and no subset has the same property. We assume a polyhedron is simple, i.e., its polygons only intersect at shared edges. Higher dimensional polyhedron have the same twoness condition.

If X and Y are subsets of $\mathbf{R}^d$, two continuous maps $h, k : X \mapsto Y$ are *homotopic* if there is a continuous map $F : X \times I \mapsto Y$ such that $F(x, 0) = h$ and $F(x, 1) = k$ for all $x \in X$. A set of points, X is *contractible* to a point if there exists a constant function homotopic to the identity function on X. The unit ball is contractible to a point.

3.2. Definition of box complexes. A box complex represents an imprecise geometric object. Its topological structure is almost the same as the topological structure of a simplicial complex. Both are defined by a graded DAG of faces with edges between a face and its subordinates. Both are typically objects without a boundary. The two differences are that a box complex allows non-simplicial faces, and a boundaryless box complex satisfies an evenness condition instead of a twoness condition (see below).

The geometric structure of a box complex is quite different from a simplicial complex. An i-face of a simplicial complex is a convex subset of an i-flat. An i-box of a box complex can be any contractible subset of $\mathbf{R}^d$. It represents the possible locations of the corresponding i-face. The only requirement is that a box contains the boxes of its subordinate faces.

This section defines the box complex, related terms, and basic theorems. The next section contains illustrations and examples.

DEFINITION 2.1. *A* box complex $\mathcal{B}$ *of dimension* n *in* $\mathbf{R}^d$ *is a finite graded, directed acyclic graph (DAG) that is labeled with contractible sets in* $\mathbf{R}^d$. *The partial order defined by set containment is compatible with the partial order defined by the DAG. Edges may go from level* i *to level* $i-1$ *only. Only one edge is allowed between nodes. The levels are* $0 \leq i \leq n$. *We require that every node must be on a path from level 0 to level* n.

The DAG induces a partial order on the nodes. Every box complex will be assumed to have an implicit top node of level $n+1$ *and a bottom node of level* -1 *which are maximal and minimal, respectively, for this order.*

For box complexes, we use the terminology of simplicial complexes. By a slight abuse of notation we refer to both a node and its label by the same symbol, B_j^i. When we wish to be specific, a node at level i is a *face* B_j^i of the box complex , and the label of a node is a *box* $B_j^i \subset \mathbf{R}^d$. B_j^{i-1} is *subordinate* to B_j^i, $B_j^{i-1} \prec B_j^i$, if there is an edge between the respective nodes. Note that $B_j^{i-1} \prec B_j^i$ implies $B_j^{i-1} \subset B_j^i$, but not vice versa.

The *dimension* of face B_j^i is i. An *i-face* is a face of dimension i. A 0-face is called a *vertex*, an $(n-1)$-face is a *facet*, and an $(n-2)$-face is a *ridge*. The *i-skeleton* of a box complex is the set of i-faces in the complex. The set of *neighbors* of an $(i-1)$-face are the i-faces it is subordinate to. We also say, two i-faces are neighbors if they have the same subordinate face. Two faces are *incident* if connected by a path in the DAG.

The *trace* of a box complex, $trace(\mathcal{B})$, is the union of its boxes. The *width* of a face, $width(B)$, is the diameter of the largest open ball contained in its box. An *exact convex hull* of a set of 0-faces is the convex hull of one point per 0-face.

DEFINITION 2.2. *Let* $\mathcal{B}$ *be a box complex of dimension* n *in* $\mathbf{R}^d$ *and let* B^n *be a* n*-face of* $\mathcal{B}$. *The* boundary *of* B^n, ∂B^n, *is the sub-DAG* $\mathcal{B}$ *given by all faces and their boxes that are below* B^n *in the partial order.*

DEFINITION 2.3. *Let $\mathcal{B}$ be a box complex of dimension n in $\mathbf{R}^d$. Let $S = \{B_1^n, B_2^n, \ldots B_s^n\}$ be a collection of n-faces in $\mathcal{B}$. We define the* contour[2] *of S, ∂S, as follows.*

Consider the s DAGs $\partial B_1^n, \partial B_2^n, \ldots \partial B_s^n$. Each $(n-1)$-face occurs some number of times in these DAGs. Delete all $(n-1)$-faces that occur an even number of times, and delete all other faces which no longer have paths to the top node. Then ∂S is the union of the remaining DAGs.

The contour is found by taking the mod 2 union of ridges in S. It takes $O(1)$ time per ridge by marking facets in S. Visiting all ridges in S identifies those ridges with a neighbor not in S.

The simplest example of a box complex is a n-dimensional simplicial complex in $\mathbf{R}^d$. The box for each simplex is the simplex itself, which is a contractible set. The DAG is exactly the same. Note that conventional notation for simplicial complexes also blurs the distinction between the geometric simplex and the topological simplex.

In this thesis, we will want the faces to represent an uncertainty in the location of the vertices, edges, facets, etc. Thus, for the rest of this thesis, *all boxes are open sets.* In the context of the point-in-polyhedron algorithm, the boxes will be, in fact, convex open sets.

To satisfy this requirement in the example above, instead of labeling each node with the corresponding simplex we can label it with the open ϵ-neighborhood of the simplex. This is still a contractible set and the necessary containment relations are still satisfied.

In this thesis, we are interested in convex hulls and polyhedron. Both objects are boundaryless. We achieve boundarylessness by the evenness condition below. It is a slight generalization of the simplicial requirement that there exists exactly two paths between any $(i+1)$-face and $(i-1)$-face. We need evenness instead of twoness because the intermediate steps of Point_in_polyhedron preserve evenness but not twoness.

[2] "Contour" is used in place of "boundary" to avoid confusion with the boundary of an open set.

DEFINITION 2.4. *A box complex satisfies the* evenness condition *if for any i, j, k there is an even number of paths from B_j^{i+1} to B_k^{i-1}. One or both faces may be the implicit top or bottom element of the partial order.*

The following two lemmas verify the boundarylessness of the evenness condition and of polyhedra.

LEMMA 2.5. *If $\mathcal{B}$ is a n dimensional box complex satisfying the evenness condition, $\partial\mathcal{B} = 0$.*

PROOF. By the evenness condition, every ridge occurs an even number of times in $\mathcal{B}$. Since the ridges are the $(n-1)$-faces of $\mathcal{B}$, all DAGs are deleted from $\partial\mathcal{B}$. ∎

LEMMA 2.6. *A polyhedron is a box complex that satisfies the evenness condition.*

PROOF. The construction is similar to a simplex. Each i-box is a subset of an i-flat. The DAG and evenness condition is constructed from bottom up. A vertex occurs in two edges of a polygon. Each edge of a polygon is an edge of another polygon. Similarly, each $(i-1)$-face belongs to exactly two i-faces. ∎

The Point_in_polyhedron algorithm successively reduces the dimension of a box complex. In Appendix A, we prove that point inclusion in a box complex is equivalent to point inclusion in a contour of the box complex. The following theorems concern the preservation of box complexes and the evenness condition. The first theorem proves a contour is a box complex. The second theorem will be used to prove that a box complex (the contour) is preserved under projection to a hyperplane. Note that if the theorems were restated for simplicial complexes, the first theorem is false and the second theorem is vacuous.

LEMMA 2.7. *Let $\mathcal{B}$ be a box complex of dimension n in $\mathbf{R}^d$ which satisfies the evenness condition. Let B^n be a n-face of $\mathcal{B}$. Then ∂B^n is a box complex of dimension $n-1$ satisfying the evenness condition.*

PROOF. Clearly, ∂B^n is a box complex of dimension $n-1$. We need to check that there are an even number of paths from level $i+1$ to $i-1$ in the new DAG.

Accounting for the implicit top and bottom nodes, the new DAG is exactly the part of the old DAG below and including B^n. The new top node is the node that was B^n.

Given two nodes in ∂B^n, they must be below B^n, so all paths between them in $\mathcal{B}$ are as well. Hence the number of paths between the two nodes is unchanged in ∂B^n. ∎

THEOREM 2.8. *Let $\mathcal{B}$ be a box complex of dimension n in $\mathbf{R}^d$ that satisfies the evenness condition. Let $S = \{B_1^n, B_2^n, \ldots B_s^n\}$ be a collection of n-faces in $\mathcal{B}$. Then ∂S is a $n-1$ dimensional box complex that satisfies the evenness condition.*

PROOF. Clearly ∂S is a (possibly empty) $n-1$ dimensional box complex. What must be shown is that it satisfies the evenness condition. We first show the evenness condition for the ridges of ∂S, and then the evenness condition for lower dimensional faces.

Consider the number of paths in ∂S from B^{n-2} to the top node. The top node corresponds to one of the B_i^n in $\mathcal{B}$. In each ∂B_i^n there are an even number of paths from B^{n-2} to the top [Lemma 2.7], thus there is an even number of paths in all the DAGs $\{B_i^n\}$ together. Since each pair of nodes is connected by at most one edge, B^{n-2} is subordinate to an even number of nodes in all the DAGs together. We need to show this property is preserved in ∂S. We analyze the two steps in the construction of ∂S in turn.

If $B^{n-2} \prec B^{n-1}$ and B^{n-1} occurs in an even number of ∂B_i^n, it gets deleted from the DAGs. Thus, an even number of paths from B^{n-2} to the top node are eliminated, preserving the parity of the number of paths.

The second step deletes all nodes that have lost paths to the top node. After the second step, we are guaranteed an odd number of paths from B^{n-2} to the top node passes through any given B^{n-1} (unless the number is zero). In the union of the DAGs, this odd number of paths becomes one path, thus again preserving the parity of the number of paths from B^{n-2} to the top.

Now consider the number of paths from B^{i-1} to B^{i+1} $(i < n-1)$ in ∂S. Any i-face between the two is on a path from top to bottom, so by construction it is

retained in every ∂B_i^n containing B^{i+1}. Thus the number of paths from B^{i-1} to B^{i+1} in ∂S is the same as in $\mathcal{B}$ itself. ∎

The next theorem is immediate from the definition of box complex and evenness.

THEOREM 2.9. *Let $\mathcal{B}$ be a box complex of dimension n in $\mathbf{R}^d$. Let $V \subset \mathbf{R}^d$ be a m-flat which we may identify with $\mathbf{R}^m$. Suppose that for each $B_k^i \in \mathcal{B}$, $V \cap B_k^i \neq \emptyset$. Suppose also that $V \cap B_k^i$ is a contractible open set for every B_k^i. Let $\mathcal{B}'$ be the DAG given by the DAG for $\mathcal{B}$, but with each box changed from B_j^i to $B_j^i \cap V$.*

Then after identifying V with $\mathbf{R}^m$, $\mathcal{B}'$ is also a box complex of dimension n in $\mathbf{R}^m$. If $\mathcal{B}$ satisfies the evenness condition, so does $\mathcal{B}'$.

3.3. Examples and related structures. Given a simplicial complex C embedded in $\mathbf{R}^d$, a box complex can be used to represent nearby simplicial complexes. (In Appendix A, we show that the converse is true, i.e., that a simplicial complex can be used to approximate a box complex.) Let the simplices of C be the nodes of a graph. Then the containment relation for the simplices induces a graded DAG structure on the graph. Label each node with an δ-neighborhood of the corresponding simplex and it is easy to see that the labeled graded DAG is a box complex.

One can think of this box complex as representing all possible mappings of C which are δ-close to the original embedding. A vertex appears somewhere in its box B_i^0, i.e., the box represents the location of a vertex with uncertainty. The uncertainty may be due to round-off error, experimental error, computational error, or other causes.

A similar construction works equally well for other combinatorial structures such as polyhedra and smooth triangulations of submanifolds in $\mathbf{R}^d$.

The boxes of a box complex can be any shape without holes. In this thesis, we use two bounding boxes and δ-boxes. A bounding box is a polytope with axis parallel sides. It is widely used in computational geometry to improve the efficiency of an algorithm. The right hand side of Figure 2.1 is a box complex with bounding boxes.

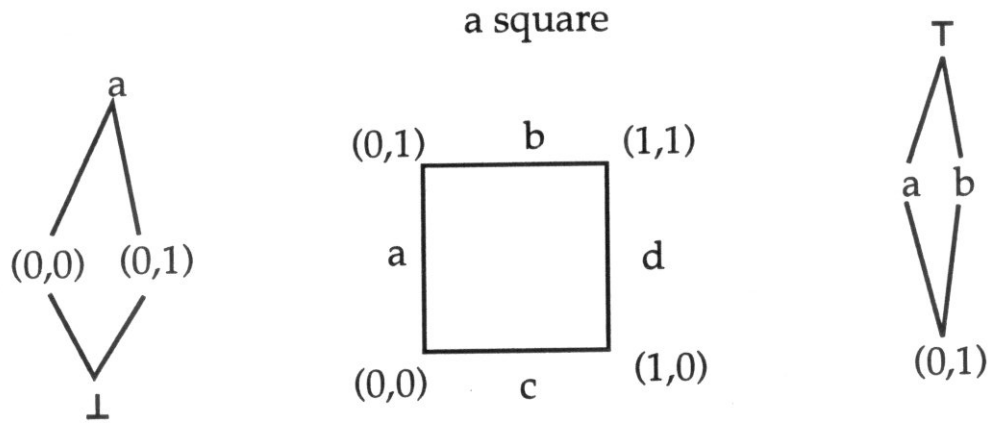


Figure 2.3: Subgraphs for the evenness condition of a square. Each subgraph contains two paths.

DEFINITION 2.10. *A* δ-box *is defined by an oriented hyperplane and a positive and negative offset from the hyperplane. The positive offset defines a parallel hyperplane called the facet's* outer plane, *and the negative offset defines its* inner plane. *The width of a* δ*-box is* δ*. The box is clipped at the projection of the facet's subordinate boxes to the hyperplane. The clipped box may be implicit.*

Note that the δ-neighborhood of a facet is contained in a (2δ)-box and the δ-box of a point is a $(\delta/2)$-ball. Figure 2.1 included an example of δ-boxes.

Box complexes allow a rich variety of structures in $\mathbf{R}^d$. Some examples are lines segments, squares, bow ties, all possible points, all possible curves between a pair of points, and all possible curves between all possible pairs of points. Figure 2.3 shows two subgraphs of a square for the evenness condition. Figure 2.4 shows a variety of box complexes.

The topology of box complexes is nearly the same as the topology of other representations for subdivisions of a manifold. Besides simplicial complexes, the principle representations are Baumgart's winged-edges, Guibas and Stolfi's quad-edges, Dobkin and Laszlo's facet-edges, and Brisson's cell-tuples. Each representation specifies topological information via a data structure and routines for traversing the data structure. Geometric information is attached to the data structure.

We defined the box complex instead of adapting an existing representation

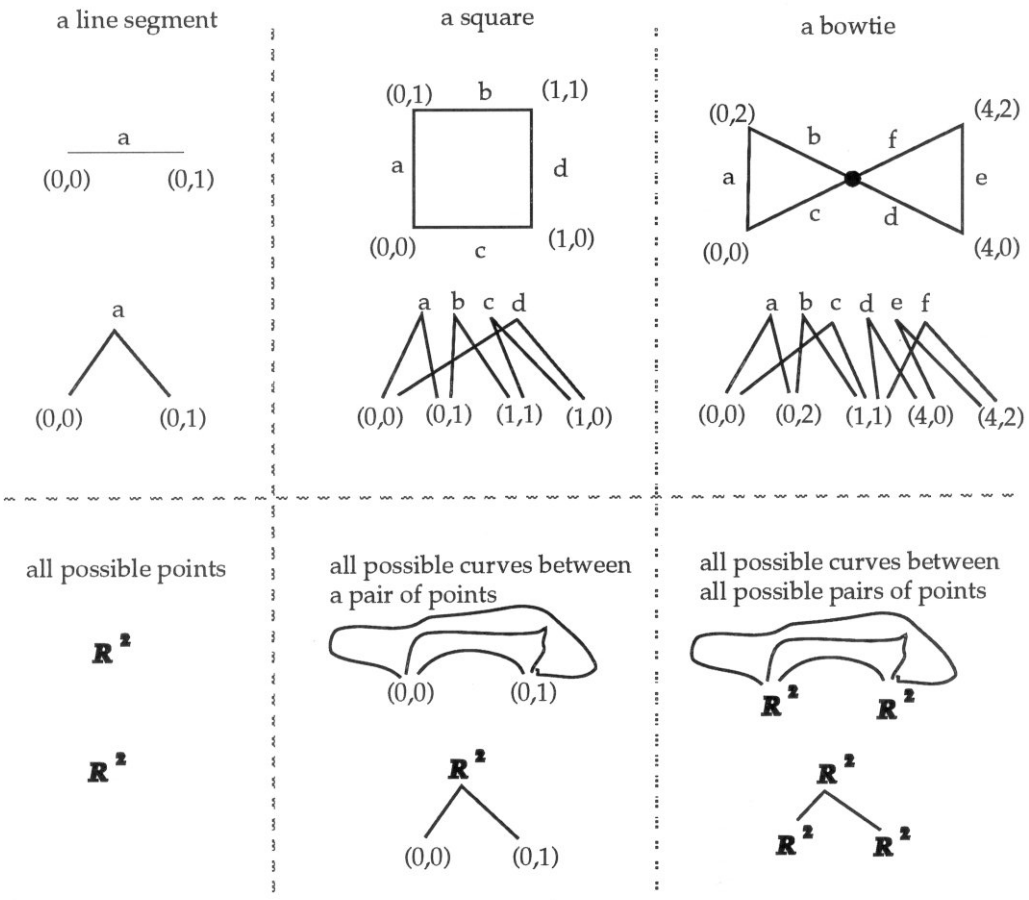


Figure 2.4: Examples of box complexes. If a box is not explicitly defined, it is the convex hull of its subordinates. The upper left examples are also represented by other models. The upper right example puts one vertex (a ridge) in four edges (facets). It is not valid in models that require a twoness condition for boundarylessness. The bottom three examples are instances of imprecise data.

for several reasons. The primary motivation was the intermediate structures constructed by Point_in_polyhedron. Point_in_polyhedron projects the contour of a subset of the facets to a hyperplane, and recursively solves point inclusion in the projected contour. Box complexes provide a representation that is easily reduced a dimension, allows for more than two uses of a ridge, and delimits imprecise data with boxes. Other representations are either specific to a dimension, limited to simplices, or require a twoness condition.

Like a simplicial complex, the box complex is a mathematical construct with a natural representation as sets of faces. For many applications (e.g., Quick_hull), the natural representation is sufficient. For 3-d and 4-d applications that satisfy a twoness condition, one of the existing representations can represent box complexes. We briefly discuss each representation below.

Baumgart's winged-edge structure was the first topologically complete, computer model of bounded polyhedra [Baumgart 1975]. Winged-edges represent a polyhedron by sets of edge nodes, face nodes, and vertex nodes. Each edge node references its neighboring faces, vertices, and edges. This specifies the corresponding DAG of a box complex by explicitly defining the edges to and from each 1-face.

Guibas and Stolfi [1985] define quad-edge structures for subdivisions of 2-d manifolds. A quad-edge links an edge into 4 rings of edges: the edges around its two neighboring faces, and the edges around its two incident vertices. Explicitly listing the vertex ring allows subordinate vertices of an edge to be the same. Operations on quad-edges corresponds to an algebra for traversing the primal, dual, and mirror image of a subdivision. Each operation takes unit time. If subordinate vertices are distinct, a quad-edge structure is equivalent to a box complex.

Dobkin and Laszlo [1989] define facet-edge structures to extend quad-edges to 3-d manifolds. They use "facet" for the 2-dimensional faces of a 3-dimensional polyhedron. A facet-edge links an edge of a facet into two rings of facet-edges: the ring of edges about a facet and the ring of facets about the edge. Facet-edges are also given for the dual representation. As the DAG for a box complex, the primal structure lists the edges between 1-faces, 2-faces, and 0-faces. The dual structure lists the edges between 1-faces, 2-faces, and 3-faces.

Brisson [1989] defines cell-tuple structures in general dimension. In the notation of box complexes, a cell-tuple is a tuple $(B_0^i, B_1^i, \ldots, B_{d-1}^i)$ such that $B_j^i \prec B_{j+1}^i$. In terms of the DAG, a cell-tuple is a path from the bottom to the top of the DAG. Brisson's cell-tuples satisfy the evenness condition with two paths for each subgraph.

Mathematically, a box complex is an open cover for its trace (a manifold). An *open cover* for a manifold is a set of open sets whose union is the manifold. Brisson proved that cell-tuple structures represent all subdivisions of a manifold [Brisson 1989]. Since box complexes include cell-tuple structures, they include subdivisions.

3.4. Convex box complexes. The Point_in_polyhedron algorithm will use the full flexibility of box complexes. The convex hull algorithm, Quick_hull, requires less flexibility and additional geometric constraints. We define two specializations of box complexes to represent the convex hull of imprecise points.

DEFINITION 2.11. *A* convex box complex *for a set of imprecise points P is a box complex such that:*

- *Its trace includes the boundaries of all exact convex hulls of P.*
- *The union of its trace with its interior includes all points in P.*

A convex box complex allows rather weak approximations. For example, any box complex is convex if one of its boxes is $\mathbf{R}^d$. The definition is strengthened by bounding the maximum facet width. As the maximum width goes to zero, a convex box complex becomes a closer and closer approximation to a precise convex hull. A *convex δ-wide box complex* is a box complex whose maximum facet width is δ.

The Quick_hull algorithm creates a convex box complex as its last step. Prior iterations of the algorithm create a locally convex box complex of δ-boxes.

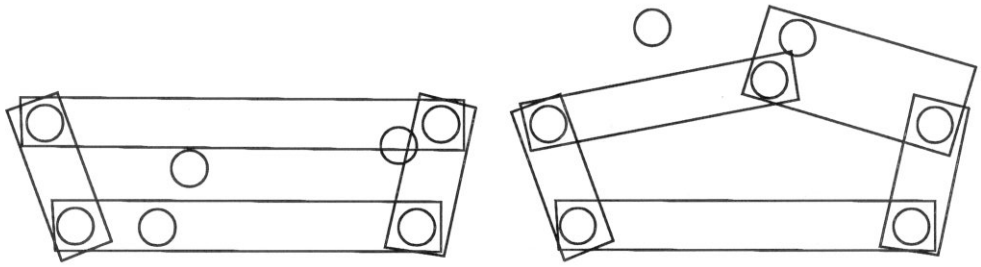


Figure 2.5: Convex and locally convex box complexes. The left-hand figure is convex and locally convex. The right-hand figure is locally convex but not convex.

DEFINITION 2.12. *A* locally convex box complex *for a set P of imprecise points is a box complex of δ-boxes such that*

- *Its 0-skeleton is a subset of P.*
- *Each δ-box contains a point (the* centrum*) that is below the hyperplanes of the δ-boxes of neighboring facets.*

As shown in "if R is clearly convex†"[3], the second condition makes each ridge clearly convex. Quick_hull always satisfies the evenness condition with two paths. The *orientation* of a facet is the orientation of its hyperplane. A facet's orientation induces a coherent orientation of its ridges and neighboring facets. In Quick_hull, we compute the centrum to be near the center of the facet (see "centrum for a facet†" below for details). Figure 2.5 shows some examples.

The next lemma shows how to turn a locally convex box complex into a convex one.

LEMMA 2.13. *Let C be a locally convex box complex for a set of imprecise points P. If the set P is contained in all outer halfspaces of C, C is a convex box complex for P.*

PROOF. The trace and interior of C contains the intersection of halfspaces defined by the outer planes. So the trace and interior of C contains the trace of P. The

[3]The dagger symbol "†" indicates a computation that is defined at the end of this chapter.

vertices of a facet are above its inner plane. So the boundary of an exact convex hull of the vertices is above the inner planes and below the outer planes. Since the convex hull of a set of points includes the convex hull of any subset, its boundary is in the trace of C. ∎

4. Basic computations and operations

This section gives the basic computations and operations used in this thesis. They may be useful in other algorithms. When mentioned in the thesis, they are flagged by "†". Operations specific to Quick_hull are flagged by "‡" and defined in Chapter 5. The basic computations determine angles and hyperplanes. The basic predicates determine the orientation of a point to a hyperplane and the orientation of a simplex. Other basic operations are defined in terms of these predicates and computations. Point_in_polyhedron uses the orientation of a point to a hyperplane. Quick_hull uses everything in this section.

The predicates return one of three results: *yes*, *no*, or *can't tell*. A predicate returns *can't tell* when its the magnitude of its computation is smaller than the maximum error due to roundoff and imprecision.

Roundoff error effects addition, subtraction, multiplication, division, and square root. It does not effect comparison. Following Wilkinson [1965], fixed precision arithmetic is same as exact arithmetic followed by rounding the result to the nearest floating point number. The maximum, relative roundoff error for a single operation is the *machine roundoff* β. Fixed precision arithmetic is modeled by the float operator $fl()$ from the reals R to the floating point numbers $R_f \subset R$:

$$fl: \ R \to R_f : fl(x) = x(1+\delta) \qquad |\delta| < \beta$$

$$fl(x \ \square \ y) = (x \ \square \ y)(1+\delta) \qquad |\delta| < \beta \qquad (\square \in \{+, -, *, /, \sqrt{}\} \qquad x, y \in R_f$$

The IEEE standards for floating point arithmetic (IEEE 754 and 854) support this model [cf. Goldberg 1991].

For example in 2-d, the distance of point $P = (p_1, p_2)$ to the line with normalized coefficients $h = (h_1, h_2, h_3)$ is $\langle h, (P, 1)\rangle$. When computed, each operation causes

roundoff. The exact, computed result is:

$$(((h_1p_1)(1+\delta_1)+(h_2p_2)(1+\delta_2))(1+\delta_3)+h_3)(1+\delta_4)$$

where $\delta_k \leq \beta$ is the roundoff error for the k'th operation. The point P is *clearly above* the line if its distance from the line is greater than the maximum error due to roundoff.

We use a symbolic math package such as Maple [Char et al 1988] to determine the partial derivatives of each error term. The sum of the magnitudes of the partial derivatives for round-off error terms is the *sensitivity* (σ) of the computation [Miller 1975]. The sensitivity times β gives the maximum roundoff error for the computation.

For example, P is above the line if

$$\langle h,(P,1)\rangle > \sigma\beta$$

The sensitivity, σ, is the sum of the magnitudes of the partial derivatives for δ_k:

$$\sigma = \sum_{i=1..2} |h_ip_i| + |\sum_{i=1..2} h_ip_i| + |\langle h,(P,1)\rangle| + 2nd\ order\ terms$$

We ignore second order effects because machine epsilon is small, round-off errors tend to cancel, and the probability of maximum precision errors is nearly zero.

If the point is imprecise, its δ-box is a $(\delta/2)$-ball. The point is above a line only if the ball is above the line. This holds if the distance of the ball's center to the line is greater than the maximum roundoff error plus δ.

A more complicated case occurs when the precision of a point cannot be separated from the roundoff error of a computation. In this case, we use a bounding box for the point's box. For example in 2-d, the area of a triangle P_1, P_2, P_3 is proportional to

$$\begin{vmatrix} p_{21}-p_{11} & p_{22}-p_{12} \\ p_{31}-p_{11} & p_{32}-p_{12} \end{vmatrix}$$

where p_{ij} is the j'th coordinate of P_i. With explicit terms for roundoff error and precision, the computation becomes:

$$\begin{aligned}&((p_{21}+\rho_{21}-p_{11}-\rho_{11})(1+\delta_1)(p_{32}+\rho_{32}-p_{12}-\rho 12)(1+\delta_2)(1+\delta_3) \; - \\ &\quad (p_{22}+\rho_{22}-p_{12}-\rho_{12})(1+\delta_4)(p_{31}+\rho_{31}-p_{11}-\rho_{11})(1+\delta_5)(1+\delta_6))(1+\delta_7)\end{aligned}$$

where $\delta_k < \beta$ is the roundoff error in operation k, p_{ij} is the j'th coordinate of P_i, and ρ_{ij} is its actual precision error. This expression ignores second order roundoff errors due to adding and subtracting the precision terms. The area of a triangle is positive if the expression is greater than the maximum error due to roundoff and imprecision (see "orientation of a simplex†" for details).

The rest of this chapter lists the basic computations and operations. We first list primitive computations and predicates. We then discuss convexity and list some basic operations that are defined in terms of the primitives. Each section discusses the operation in general dimension and gives its computational complexity as a function of the dimension. Calculations are given for 2-d and 3-d. "F" refers to a facet, "P" refers to an imprecise point, "R" refers to a ridge, and "S" refers to a set.

4.1. Primitives.

angle of a ridge

The cosine of the angle formed by a ridge's neighboring facets is the inner product of their normalized hyperplane coefficients: $\langle \hat{h}_1, \hat{h}_2 \rangle$. The interior angle for the facets is the supplementary angle: $\cos^{-1}(-\langle \hat{h}_1, \hat{h}_2 \rangle)$. For this thesis, the interior angle is always used.

The parameter θ for Quick_hull sets the maximum interior angle. Two facets are *θ-coplanar* if the interior angle is in the range $[\theta, 2\pi - \theta]$. The parameter θ is used to determine if neighboring facets of a ridge form a clearly convex angle (see "if R is clearly convex†"). It is entered as the maximum, computed cosine that includes roundoff error and normalization error.

The inner product uses $O(d)$ operations.

LEMMA 2.14. *Let d be the dimension and $\beta < 0.01(d+1)$ be machine epsilon. Given normalized hyperplane equations, the roundoff error in computing their angle cosine is $1.01\beta d$.*

PROOF. The cosine is equivalent to the inner product of two normalized equations. For vectors x and y of length d, the inner product has an error bound of

$1.01\beta d\langle |x|, |y| \rangle$ [e.g., Golub & van Loan 1983, 3.2-5]. The inner product is at most 1 because $\langle x, y \rangle \leq \|x\|\|y\|$. ∎

In Quick_hull, the hyperplane equations are not exactly normalized because of roundoff error. In 2-d (resp. 3-d) the absolute error in a coefficient is at most 3.5β (resp. 4.5β), see "hyperplane through a simplex†". So the maximum roundoff error is 4.51β (resp. 5.51β). If the user specifies a bound for θ, we assume that the bound includes the maximum roundoff error.

hyperplane through a simplex

In $\mathbf{R}^d$, exactly d independent points defines a hyperplane. These points also define a $(d-1)$-simplex. To create an oriented hyperplane for a facet, we use a simplex consisting of an apex (a point) and an oriented base (a ridge of $d-1$ points). We compute the coefficients of the hyperplane through the simplex by standard methods [e.g., Bowyer & Woodwark 1983]. It takes $O(d^3)$ operations. The normal equation's offset places the apex exactly on the hyperplane. Orientation of the hyperplane is set by the topological orientation of the ridge.

The hyperplane's coefficients are normalized. If normalization causes overflow, the largest magnitude coordinate becomes the unit normal. Roundoff error effects normalization. By partial differentiation, the sensitivity of 2-d (resp. 3-d) normalization is less than 3.5 (resp. 4.5) [inner product error terms contribute 1/2 apiece].

The following is a subtle point: *A computed hyperplane is the exact hyperplane with the given floating point coefficients.* So a computed hyperplane through a simplex is not an imprecise object. Then we can use the hyperplane's coefficients as precise values instead of a range of possible values. A side effect of this decision is that the simplex's vertices may be above or below the hyperplane. We compute the maximum deviation of a vertex from a hyperplane.

Note that our use of precise hyperplanes differs significantly from Guibas et al [1989]. Their "ϵ-butterfly" is the set of all possible hyperplanes through a simplex (In 2-d, all possible lines through two points). We have found that a single precise hyperplane is easier to manipulate than the set of all possible hyperplanes.

if P is clearly above (below) a hyperplane

An imprecise point P is *clearly above* (resp. *clearly below*) a precise hyperplane H if the signed distance of P to H is clearly positive (resp. negative). This is equivalent to evaluating the hyperplane's normal equation at the point. It takes $O(d)$ operations.

Let $(h_1, h_2, \ldots, h_{d+1})$ be the hyperplane's coefficients and $(p_1, p_2, \ldots, p_d)$ be the point's coefficients. The distance of P to H is clearly positive (resp. negative) if it is positive (resp. negative) and

$$|\sum_{i=1..d} h_i p_i + h_{d+1}| > \rho + \sigma\beta$$

where ρ is the point's precision, σ is the computation's sensitivity, and β is the machine epsilon.

In three and higher dimensions, we use balanced addition (e.g., (1+2)+(3+4)) instead of left-to-right addition. By partial differentiation, the sensitivity in 2-d is:

$$\sigma = \sum_{i=1,2} |h_i p_i| + |\sum_{i=1,2} h_i p_i| + |\sum_{i=1,2} h_i p_i + h_3| + 2nd\ order\ terms.$$

In 3-d the sensitivity is:

$$\sigma = \sum_{i=1..3} |h_i p_i| + |\sum_{i=1,2} h_i p_i| + |\sum_{i=3,4} h_i p_i| + |\sum_{i=1,3} h_i p_i + h_4| + 2nd\ order\ terms.$$

In both cases, the last term is the distance computation. Since we are only concerned about values that are nearly zero, the last term has a second order effect and can be ignored.

If $\rho > 0$, we must account for roundoff error in normalizing the hyperplane equation (see "hyperplane through a simplex†"). If the maximum coordinate is Δ, the maximum error due to normalization in 2-d (resp. 3-d) is $7\Delta\beta$ (resp. $13.5\Delta\beta$).

Note that the sensitivity depends on the magnitude of the point's coordinates. Let P' be the arithmetic center of the point set. Translating the point set by $-P'$ would reduce the average coordinate but would introduce an additional roundoff error of up to:

$$(\sum |p_i| + \sum |p'_i|)\beta.$$

This would cause a larger total error for many of the points.

In 3-d, if addition were performed left to right instead of balanced, the 5th error term would be $|h_1p_1 + h_2p_2 + h_3p_3|$. When the distance is nearly zero, the current 5th term is approximately $|h_1p_1 + h_2p_2|$. So balanced addition reduces σ. Another potential improvement is the Kahan summation formula [Kahan 72], but it only bounds the error to twice the absolute value of each summand. While not helpful in 3-d, it would reduce roundoff error in higher dimensions.

orientation of a simplex

The orientation of a simplex is positive (resp. negative) if its signed volume is positive (resp. negative). The volume of a simplex is proportional to the magnitude of a $d \times d$ determinant[4]. In general dimension this is equivalent to finding the L-U decomposition of a matrix and multiplying the diagonal elements. Gaussian elimination with partial pivoting solves this in $O(d^3)$ operations [e.g., Golub & van Loan 1983]

The volume of $d+1$ points $P_0, P_1, \ldots, P_d$ is clearly negative or clearly positive if

$$\left| \begin{array}{c} P_1 - P_0 \\ P_2 - P_0 \\ P_3 - P_0 \end{array} \right| > \mathcal{E}_{\text{prec}} + \sigma\beta$$

where $\mathcal{E}_{\text{prec}}$ is the maximum error due to a point's precision, and $\sigma\beta$ is the maximum roundoff error. Symbolic differentiation determines $\mathcal{E}_{\text{prec}}$ and σ. Let ρ be the maximum precision error of a point and Δ_i be the maximum magnitude of the i-th coordinate. Then the maximum error due to a point's precision is bounded by:

$$\mathcal{E}_{\text{prec}} \leq \begin{cases} 2\rho, & d = 1 \\ 3\rho(\Delta_1 + \Delta_2) + 12\rho^2 + 2nd\ order\ terms, & d = 2 \\ 9\rho(\Delta_1\Delta_2 + \Delta_2\Delta_3 + \Delta_1\Delta_3) + 36\rho^2(\Delta_1 + \Delta_2 + \Delta_3) + 2nd\ order\ terms, & d = 3 \end{cases}$$

[4]This computation is not used in Point_in_polyhedron or in Quick_hull. It is used for the counter-clockwise test and its higher dimensional equivalents (see "if R is clearly convex†").

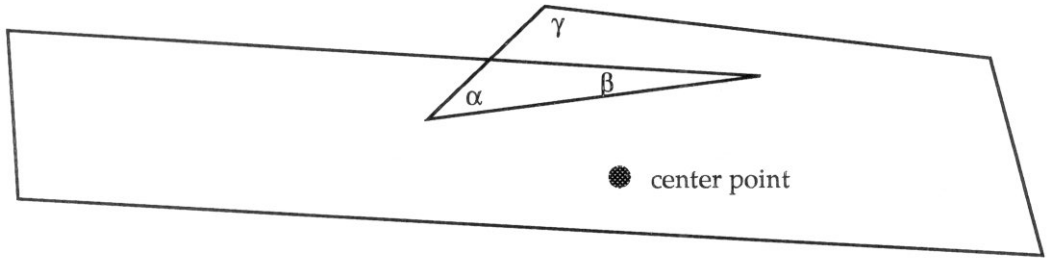


Figure 2.6: Clearly convex angles may wind more than once around an interior point

and when the computation is nearly 0, its sensitivity is bounded by:

$$\sigma \leq \begin{cases} 2\Delta_1, & d = 1 \\ 6\Delta_1\Delta_2 + 2nd\ order\ terms, & d = 2 \\ 40\Delta_1\Delta_2\Delta_3 + 2nd\ order\ terms, & d = 3 \end{cases}$$

4.2. Other operations.

if R is clearly convex

A ridge is *clearly convex* if its angle is less than θ (see "angle of a ridge†") and if its neighboring centrums are both *clearly below* the opposite hyperplane (see "if P is clearly above†"). If all ridges of a box complex are clearly convex, the box complex is locally convex.

A locally convex box complex may wind its facets more than once around an interior point. The problem is easily illustrated in 2-d where a sequence of convex angles can form a loop in the surface (Figure 2.6). In 2-d, Quick_hull partitioning prevents this situation from arising. In 3-d, facets could wind. An example is shown in Chapter 7. To prevent winding in Quick_hull, we perform an additional test when the angle angle is less than $\pi/2$. For acute angles, the arithmetic center of the initial hull must be *clearly below* both hyperplanes.

All computations use $O(d)$ operations in $\mathbf{R}^d$.

The remainder of this subsection concerns the convexity test. We prove that it is correct and that an angle test is insufficient. The subsection concludes with a

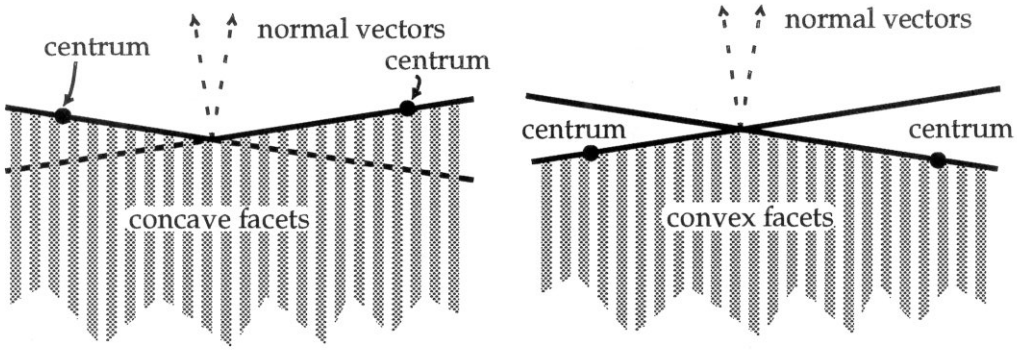


Figure 2.7: Facet centrums are needed to distinguish convex from concave angles

discussion of Fortune's 3-d convexity test and the counter-clockwise test.

THEOREM 2.15. *Two facets form a convex angle if each centrum is below the other facet's hyperplane.*

PROOF. The line segment between the centrums is interior to both half spaces. Similarly all line segments for points between the centrums and the hyperplanes' intersection are interior to both half spaces. ∎

LEMMA 2.16. *Testing the angle between hyperplanes is insufficient for distinguishing concave angles from convex ones.*

PROOF. Without loss of generality, consider hyperplanes that are parallel to all but the x and y axis, symmetric about the y axis, with the origin at their intersection (Figure 2.7). The concave and convex cases are only distinguished by marking the facets. ∎

In other words, facets may form a convex angle but not hyperplanes. For this reason, Fortune's test of 3-d convexity [Fortune 1992] does not extend to non-simplicial facets. His test compares two simplicial facets. If the outer product of the hyperplane normals has the same direction as the common edge, the facets are convex, otherwise they are concave (Figure 2.8). With simplicial facets, the

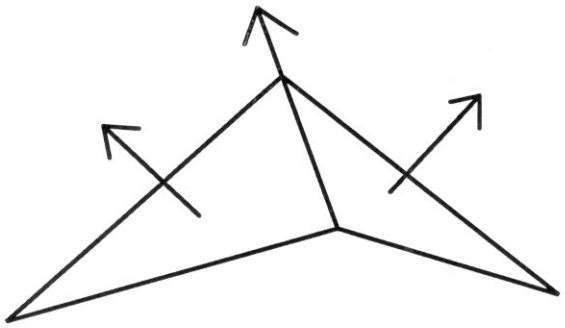


Figure 2.8: Fortune's [1992b] test for 3-d convexity.

common edge lies on both hyperplanes. With non-simplicial facets, the common edge is unrelated to the intersection of the hyperplanes and the test is not valid. In 2-d, by using the XY plane in $\mathbf{R}^3$, Fortune's test is valid since all ridges are "parallel" to the Z axis.

In 2-d, another way to distinguish concave from convex angles is the counter-clockwise test. Given a sequence of three points, the counter-clockwise test determines if the sequence turns counter-clockwise. It identifies convex angles between successive edges of a polygon. If the exterior of the polygon is to the right, edges are convex if their vertices are counter-clockwise.

The counter-clockwise test consists of determining the sign of the area formed by the three points. In 3-d and higher dimensions, the counter-clockwise test generalizes to the sign of the volume of a simplex (see "orientation of a simplex†"). It is equivalent to testing a point against the ϵ-butterfly of hyperplanes through a $(d-1)$-simplex [Guibas et al 1989].

The counter-clockwise test is used for precise convex hull algorithms [e.g., Graham 1972] and for imprecise, 2-d convex hull algorithms [Fortune 1989; Guibas et al 1990; Li & Milenkovic 1990]. As others probably have done, we attempted to use the counter-clockwise test for fixed precision arithmetic in 3-d. A problem is that 3-d facets may have more than 3 vertices. Each simplex of a facet's vertices can define different hyperplanes. Repeated merges can then arbitrarily perturb the original hyperplane. As done in Quick_hull, a simplex can be assigned to each facet, but there does not appear to be a consistent way to pick the other vertex for the

counter-clockwise test.

if P is maybe above (below) a hyperplane

A point is *maybe above* (resp. *maybe below*) a hyperplane if it is not *clearly below* (resp. *clearly above*) the hyperplane (see "if P is clearly above†").

Both computations take $O(d)$ operations.

if R is clearly concave

A ridge is *clearly concave* if its angle is less than θ (see "angle of a ridge†") and if its neighboring centrums are both *clearly above* the opposite hyperplane (see "if P is clearly above†").

Both computations use $O(d)$ operations in $\mathbf{R}^d$.

if R is coplanar

A ridge is coplanar if it is not *clearly convex* nor *clearly concave* (see "if R is clearly convex†" and "if R is clearly concave†").

Either test takes $O(d)$ operations in $\mathbf{R}^d$.

Chapter 3

Point_in_polyhedron

Is a point inside or outside a polyhedron? This is a basic geometric operation for point membership classification. For example in constructive solid geometry, objects are composed by set operations on primitive and previously constructed objects. Point inclusion is the first step in adding an object to a representation. Point classification is also important for clipping, geometric intersection, and ray tracing [Tilove 1980; Hanrahan 1989].

A standard algorithm for Point_in_polyhedron shoots a test ray from the point. One counts the number of facets crossed by the test ray. If an even number are crossed, the point is outside of the polyhedron; if odd, the point is inside. This is called the *parity test.*

As often happens with geometric algorithms, there are a number of special cases to worry about. For example, an edge could be exactly coincident with a ray, or the ray could intersect a vertex. In 2-d, an infinitesimally rotated, horizontal ray avoids these cases. Since the ray is horizontal, incidence is equivalent to comparing y coordinates. Since y coordinates have a finite representation, an infinitesimal rotation avoids all incidences [Preparata & Shamos 1985].

The problem is more complex in three and higher dimensions. The parity test remains valid but a fixed rotation does not avoid all incidences. Consider a ray that intersects an edge. If the rotation is fixed, it may be in the same direction as the edge. If so, the rotated ray still intersects the edge. Furthermore, with fixed precision arithmetic, a computation can not always tell whether or not a ray

intersects an edge.

This chapter presents a point-in-polyhedron algorithm that avoids such problems. It is the first point inclusion algorithm to work in general dimension for arbitrary inputs and fixed precision arithmetic. In addition, it works even if the only geometric information is bounding boxes. The algorithm uses the parity test but it never determines that a ray crosses a facet. In brief, it locates facets whose boxes intersect the test ray, finds the contour of these facets, and recursively runs the parity test on the contour. It ends when the contour is a subset of the polyhedron's vertices.

The parity test on simple polyhedra is an application of the Jordan-Brouwer separation theorem and its generalizations [cf., Munkres 1984, Th. 36.3]. Jordan in 1887 initiated a proof that every simple closed curve separates the plane into two pieces. This was proved in 1905 by Veblen and generalized to arbitrary dimension by Brouwer. The Jordan-Brouwer separation theorem states that a set homeomorphic to a sphere separates $\mathbf{R}^d$ into two components. One of the components is bounded while the other is unbounded. The bounded component is the sphere's *inside* while the other is its *outside*. A corollary states that a ray from the inside crosses a sphere an odd number of times. If a polyhedron, this is equivalent to counting the number of facets crossed by a ray. The even/odd parity of the number of crossing is the *crossing parity* for the point.

Several solutions have been published for point inclusion in 3-d. The simplest is to randomly rotate the polyhedron before counting the facet crossings [Corkum & Wyllie 1990]. If the ray intersects an edge or vertex, the polyhedron is rotated again. The second solution is to merge faces when a singularity is detected [Kalay 1982]. The third solution is to locate the closest feature and then test the orientation of neighboring facets and edges [Horn & Taylor 1989].

The fourth solution is quite different. Instead of casting a single ray, it casts all rays from a point. It indirectly counts the crossings of each ray by projecting each oriented facet to the unit sphere, and integrating the signed, projected areas. If a ray crosses two facets, their surface patches have opposite sign and cancel. So if a ray crosses an even number of times, the corresponding surface patches contribute

zero to the surface integral. If a ray crosses an odd number of facets, it contributes one surface patch to the surface integral. If the sum is the sphere's area (resp. zero, area/2), the point is inside (resp. outside, on) [Lane et al 1984]. This approach is practical and robust when arithmetic is sufficiently precise to distinguish three possible values.

We seek a general dimension algorithm that is independent of the arithmetic's precision. The other algorithms are either dimension specific or may fail under fixed precision arithmetic. Kalay's and Horn and Taylor's algorithms are specifically for 3-d. They also include many special cases. Simulation of Simplicity uses precise arithmetic. With fixed precision arithmetic, Corkum & Wyllie's algorithm may never find a rotation that avoids singularities. If the arithmetic is sufficiently imprecise, Lane et al's algorithm can not distinguish the three cases.

The algorithm presented here works in general dimension with imprecise data and arithmetic. The algorithm is intriguing because it reduces point inclusion to the parity of a subset of the vertices. It never identifies a facet crossing. Of the four solutions, it is closest to Kalay's.

1. Point_in_polyhedron algorithm

The following algorithm works with precise and imprecise data. Point_in_polyhedron($\mathcal{B}$, P, d) returns *clearly inside*, *clearly outside*, or *can't tell*. The first two are strong predicates for P inside or outside of a polyhedron $\mathcal{B}$ in $\mathbf{R}^d$. All sources of imprecision are captured by the boxes of $\mathcal{B}$.

After presenting the algorithm, we will discuss precise data, bounding boxes, and δ-boxes. These effect the classification subroutine but do not effect the algorithm itself. Point_in_polyhedron uses d orthogonal test rays (we use the d positive axes originating at P).

The classification subroutine, Classify(F,P,d), classifies facet F for the d'th test ray from point P. It returns *in front* if the test ray intersects F's box. It returns *not infront* if the test ray does not intersect. It returns *can't tell* if the test ray and its opposite ray intersects F's box. The last case occurs if P is inside the box, or if the box wraps around P. With precise data and arithmetic, *can't tell* only occurs

```
let S_f be a set of facets (the front half), initially empty
for each facet F of B
    Class= Classify (F, P, d)
    if Class = can't tell
        return can't tell
    if Class = in front
        append F to S_f
if d > 1
    return Point_in_polyhedron(S_f's projected contour, P, d-1)
else
    if S_f's parity is odd
        return clearly inside
    else
        return clearly outside
```

Table 3.1: Algorithm for the recursive function Point_in_polyhedron($\mathcal{B}$, P, d). The algorithm tests if point P is inside polyhedron $\mathcal{B}$ in $\mathbf{R}^d$. It returns *clearly inside*, *clearly outside*, or *can't tell*. Each recursive call projects the contour of S_f to the perpendicular hyperplane to the test ray. Instead of $d = 1$, the recursion can terminate at $d = 2$ with the point inclusion algorithm of Preparata and Shamos [1985].

when P is on the facet. Figure 3.1 illustrates the possible cases.

Point_in_polyhedron is a recursive algorithm that reduces the dimension of the problem. At dimension d, it divides the facets ($(d-1)$-faces) into the front half and the back half. The *front half* consists of those facets *in front* of the point. The *back half* consists of those *not infront* of the point. Their intersection is the contour. See Table 3.1 for the algorithm.

Point_in_polyhedron is simplest when each test ray misses all lower dimensional features. For example, in $\mathbf{R}^3$ the facets are polygons. Let the point be inside the polyhedron. Let the test ray go through exactly one polygon. Point_in_polyhedron projects the edges of the polygon (its contour) to the perpendicular plane through the test point. Point-in-polyhedron becomes point-in-polygon. Let the second test ray go through exactly one edge (note that the test point is inside the polygon).

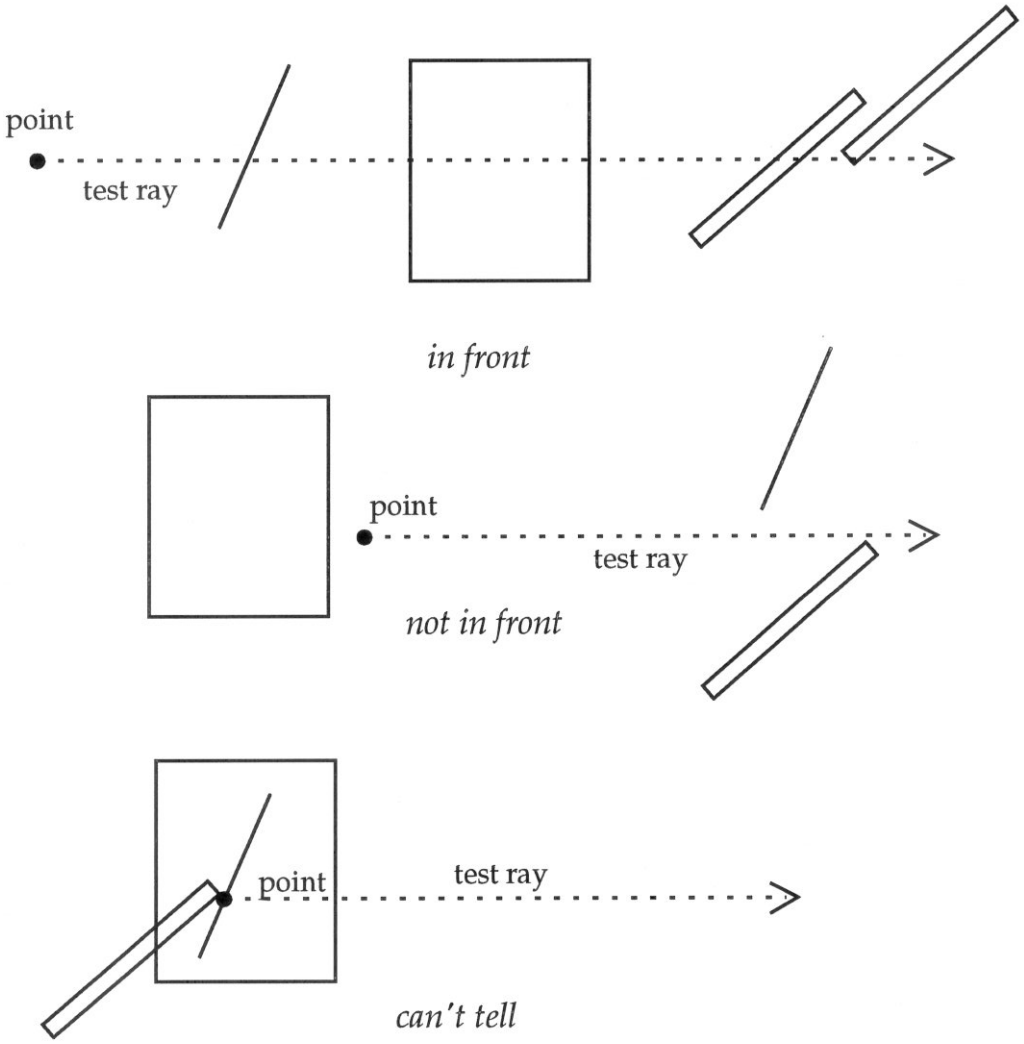


Figure 3.1: The orientation subroutine returns *in front*, *not infront*, or *can't tell.*

Point_in_polyhedron projects the vertices of the edge to a perpendicular line through the test point. Now there's two vertices with the test point between them. The third test ray (there's only two possibilities) goes through one or the other vertex. Since the size of S_f is odd, Point_in_polyhedron reports that the point is clearly inside the original polyhedron.

The following subsections discuss what happens when data is precise, boxes are bounding boxes, or boxes are δ-boxes. These change the orientation subroutine but do not change Point_in_polyhedron. Then we sketch the correctness proof for Point_in_polyhedron (proved in Appendix A).

1.1. Precise data and arithmetic. If data and arithmetic are precise, a test ray either crosses or does not cross a facet. If the test ray crosses a lower dimensional feature, then it crosses all incident facets. There are two cases in Point_in_polyhedron. If $d = 1$, then the only faces are vertices and the point is inside if an odd number of vertices are crossed. If $d > 1$, then at the recursive call to Point_in_polyhedron, S_f consists of all facets crossed by the test ray. Taking the contour of S_f is the same as merging these facets together and returning their boundary ridges. This reduces the singular case to the non-singular case above.

Figure 3.2 shows the steps for $\mathbf{R}^2$. The test ray crosses three edges. Their contour consists of two vertices. Point_in_polyhedron projects the vertices to the perpendicular through the test point. This is the same as stretching the figure at the vertices and intersecting it with the perpendicular. The reduced problem is the same as the original because the stretched figure does not change the insideness of the point.

1.2. Bounding boxes. If data and arithmetic are imprecise, the boxes for the polyhedron are open sets. First consider what happens when the boxes are bounding boxes. It is easy to tell if a test ray intersects a box (just compare coordinates), but it may not cross the facet itself. Point_in_polyhedron still works because it only needs to determine those facets that could be crossed by the test ray.

Point_in_polyhedron is exactly the same as in the precise case. Figure 3.3 illustrates the process. Again there are two cases. If $d = 1$, the bounding box for a vertex is either crossed or not crossed. So parity determines point inclusion. If

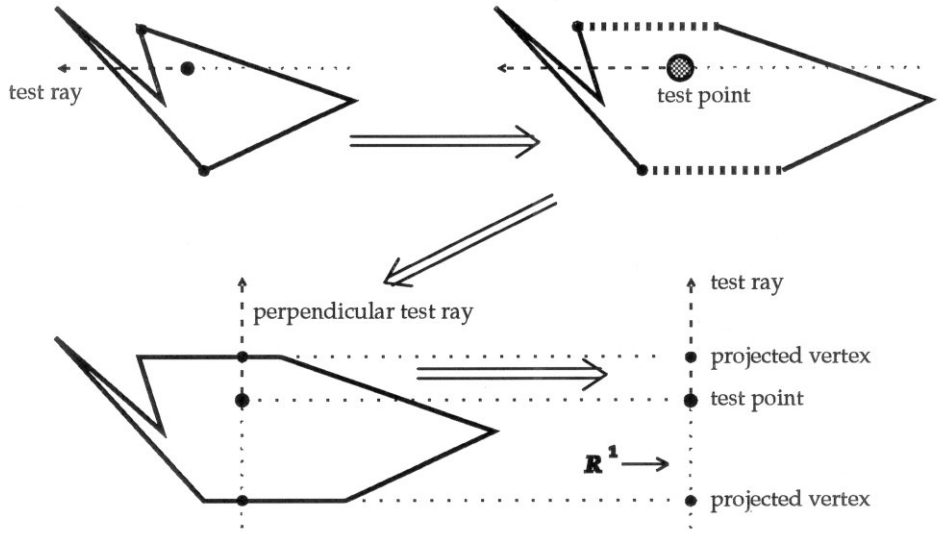


Figure 3.2: Point_in_polyhedron reduces point inclusion to a lower dimensional problem. It identifies the crossed facets, stretches the figure at their contour, and intersects the stretched figure with the perpendicular hyperplane.

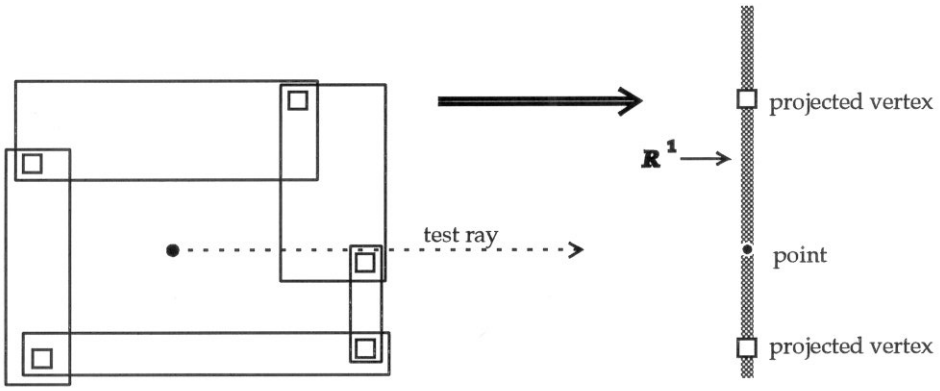


Figure 3.3: Point_in_polyhedron with bounding boxes. As in the precise case, Point_in_polyhedron identifies the facets whose boxes intersect the test ray, stretches the figure at their contour, and intersects the stretched figure with the perpendicular hyperplane.

```
Class= ClassifyBoundingBox (F, P, d)
if d > 1 and Class = can't tell
    Class= ClassifyHyperplane (F, P, d)
    if d > 2 and Class = can't tell
        Contour= project F's contour to a hyperplane along the test ray
        if ∀_{R∈Contour} ClassifyRidge (R, P, d-1) = not infront
            Class= not infront
        else if ∀_{R∈Contour} ClassifyRidge (R, P, d-1) = in front
            Class= in front
        else
            Contour= project F's contour to the test ray's perpendicular hyperplane
            if Point_in_polyhedron(Contour, P, d-1) = clearly outside
                Class= not infront
            else
                Class= can't tell
return Class
```

Table 3.2: Classify (F, P, d). Classification subroutine for facet F relative to point P's test ray in $\mathbf{R}^d$.

$d > 1$, then Theorems 2.8 and 2.9 show that the reduced structure is still a box complex. The argument that point inclusion is the same is more complicated. The problem is that the actual polyhedron could be anywhere in the boxes. We sketch the proof in a later section.

Note that we need very little geometric information about the location of a facet and its faces. Point_in_polyhedron is the first point inclusion algorithm to work with bounding boxes alone.

1.3. δ-boxes. The problem with bounding boxes is that they can sweep out a much larger volume than necessary. We use bounding boxes for the initial classification of a facet, and δ-boxes if the initial classification is *can't tell.* In an application, other box descriptions could also be used. The pseudocode for Classify with δ-boxes is in Table 3.2.

Each step of Classify(F, P, d) is discussed below.

1.4. Pseudocode steps for Classify.

Class= ClassifyBoundingBox (F, P, d)

See Section 1.2 for an explanation.

if $d > 1$ and Class = *can't tell*

If $d = 1$, the bounding box and δ-box of a facet are the same. If $d > 1$, the δ-box can be much smaller than the bounding box.

Class= ClassifyHyperplane (F, P, d)

The test ray is the positive d'th axis originating at P. This step determines whether or not the test ray crosses F's hyperplane H. If P is or could be between the inner and outer plane for H, ClassifyHyperplane returns *can't tell.* If the test ray crosses H, ClassifyHyperplane returns *in front.* Otherwise, ClassifyHyperplane returns *not infront.* See "if P is clearly above $H^{\dagger}$" for computing the distance of P to H. Crossing is detected by multiplying the signed distance to H by H's d'th coefficient.

if $d > 2$ and Class = *can't tell*

If $d = 2$, F's hyperplane and bounding box determines its box. If $d > 2$, P can be near F's hyperplane but outside of F's box. The following steps determine if this is the case.

Contour= project F's contour to a hyperplane along the test ray

First we exactly project F's contour to a hyperplane along the test ray. If either the test ray or a ray in the opposite direction entirely misses the projected contour, then it entirely misses the original contour. The current implementation of Classify(F, P, d) just uses the recursive call to Point_in_polyhedron described below. So the details for these steps are incomplete.

if $\forall_{R \in \text{Contour}}$ ClassifyRidge (R, P, d-1) = *not infront*

ClassifyRidge(R, P, d-1) is the same as Classify(R, P, d-1) except that the test rays differ.

else if $\forall_{R \in \text{Contour}}$ ClassifyRidge (R, P, d-1) = *not behind*

This condition can be determined in the previous step. A ridge is *not behind* the point if the ray opposite to the test ray does not intersect the ridge's box.

Contour= project F's contour to the test ray's perpendicular hyperplane

The previous projection fails if the facet lies in the perpendicular hyperplane or if the contour wraps around the test point. We resolve these cases by exactly projecting the contour to the perpendicular hyperplane and running Point_in_polyhedron.

if Point_in_polyhedron(Contour, P, d-1) = *clearly outside*

If the test point is outside of the projected contour, then the test ray must miss the facet. Otherwise, the test point is inside the facet's bounding box or the test ray lies in the facet's hyperplane and the facet's contour wraps around the test point. Note that a different test ray could resolve the last case.

2. Correctness proof for Point_in_polyhedron (sketch)

LEMMA 3.1. *All ridges in the contour of a front half are not infront and not behind the test point.*

PROOF. Point_in_polyhedron partitions the facets into a front half and a back half. The front half contains all facets that are *in front* of the test point. These facets and their subordinates are *not behind* the point. Similar all facets and subordinates in the back half are *not infront* of the point. The contour is in both the front half and the back half. ∎

COROLLARY 3.2. *Let $\mathcal{B}$ be the projection of the contour's boxes to the perpendicular hyperplane. Then $P \notin \text{trace}(\mathcal{B})$.*

THEOREM 3.3. *Let $\mathcal{B}$ be a 0-box complex and P be a point in $\mathbf{R}^1 - \text{trace}(\mathcal{B})$. Inclusion of P in $\mathcal{B}$ is equivalent to the odd parity of a multiset of vertices of $\mathcal{B}$.*

PROOF. A 0-face complex is embedded in $\mathbf{R}^1$, so a test ray and its opposite intersect every 0-face of the complex. The algorithm partitions the facets (i.e., vertices) into those which the test ray does *not* cross and those which the opposite ray does *not* cross. Vertices in the front half are crossed by the test ray but not by its opposite. The reverse holds for vertices in the back half. By the evenness condition, there is an even number of vertices in a 0-face complex. So an odd number of vertices are crossed if and only if either half contains an odd number of vertices. If $\mathcal{B}$ is a simplex, it consists of two vertices. Consider points in neither vertice's box. If a point is between the vertices, it is *clearly inside* $\mathcal{B}$, otherwise it is *clearly outside* $\mathcal{B}$. ∎

The proof of correctness for higher dimensions is in Appendix A. We first construct a *realization* for $\mathcal{B}$, i.e., a simplicial complex that has the same DAG as $\mathcal{B}$. The boxes of $\mathcal{B}$ contain the realization in each dimension. We prove the point inclusion theorems for a realization, and then prove that all realizations give the same results. We use homology theory to prove these theorems.

The correctness proof comes in three parts. The first part proves that box complexes separate $\mathbf{R}^d$ into multiple components. The second part proves that the parity test distinguishes inside points from outside points. The third part proves that point inclusion in a box complex is equivalent to point inclusion inside the projected contour of the front half.

The key theorem follows:

THEOREM 3.4. *Let H be a plane perpendicular to a test ray through a point p, and C be the contour of p front half in a box complex $\mathcal{B}$. Let C' be the projection of C to H. If the boxes of C' are convex, the crossing parity of p relative to $\mathcal{B}$ is the same as the crossing parity of p relative to C'.*

PROOF. [proof sketch] Translate the front half along the test ray until it is on the positive side of H. Similarly, translate the back half until it is on the negative side of H.

The contour still connects the front half to the back half. Its boxes were stretched rectilinearly. The boxes of the neighboring facets were also stretched. Since no topological changes were made and all boxes are contractible to a point, the stretched structure is still a box complex. It looks like the old box complex with a "fat middle".

The realization is similarly stretched. Since the intersection of the test ray with the stretched realization is unchanged, the crossing parity of the point is unchanged. Now intersect the stretched boxes with H. The intersection is the same as the projection of the contour's boxes to H. A test ray in H must have the same crossing parity as the original test ray.

By Theorem 2.9, we still have a box complex. Retriangulating the intersection of H with the stretched realization gives a realization for the intersected boxes. The intersected realization has the same crossing parity as before. ∎

Point_in_polyhedron is an implementation of this theorem with Theorem 3.3 ending the recursion. With Appendix A, this completes the proof of correctness:

THEOREM 3.5. *Let $\mathcal{B}$ be a box complex and p be a point in $\mathbf{R}^d - \text{trace}(\mathcal{B})$. If the boxes of projected contours are convex, Point_in_polyhedron computes the crossing parity of p.*

If the projected contours are non-convex, Point_in_polyhedron usually computes the crossing parity of p. If it doesn't, it reports *can't tell.* This occurs when both the test ray and its opposite intersect a box.

3. Complexity analysis of Point_in_polyhedron

The *tolerance region* of a facet's hyperplane is the intersection of its bounding box with the region between its outer plane and inner plane.

LEMMA 3.6. *Let n be the number of faces in a box complex.*[1] *If the point is outside of all but a constant number of tolerance regions, the space and time complexity for Point_in_polyhedron is $O(n)$.*

PROOF. Classify($\mathcal{B}$, P, d) is non-recursive if points are outside the tolerance regions. It processes each face no more than once with $O(1)$ work per face. Point_in_polyhedron performs $O(1)$ work per face. A recursive call to the orientation subroutine has the same complexity as Point_in_polyhedron with points outside of tolerance regions. The size of the maximum contour is n which bounds the space complexity. ∎

4. Summary

Point_in_polyhedron is a point inclusion algorithm that allows approximate geographic information. Its input is a point and a polyhedron represented by a box complex. Its output is *clearly inside*, *clearly outside*, or *can't tell*. Unless a point is inside many tolerance regions, Point_in_polyhedron takes $O(n)$ space and $O(n)$ time where n is the number of faces.

Point_in_polyhedron is the first point inclusion algorithm to work in any dimension, on data that is arbitrarily imprecise, and with arithmetic that is arbitrarily imprecise. It works even if the only geometric information is bounding boxes. Its boxes may be useful for other applications. Point_in_polyhedron is unique in that it tests point inclusion by determining the parity of a subset of the vertices.

[1] Note that n is $O(m^{\lfloor d/2 \rfloor})$ where m is the number of vertices and d is the dimension.

Chapter 4

Convex hull

Consider the convex hull in arbitrary dimensions. This is a well understood problem with many applications and algorithms. It is used in pattern recognition, statistics, polyhedron intersection, and image processing. In 3-d and higher dimensions, it is used for Delaunay triangulation, Voronoi diagrams, and power diagrams. Aurenhammer [1991] gives a thorough review of the literature. He describes applications in file searching, cluster analysis, collision detection, crystallography, metallurgy, urban planning, cartography, image processing, numerical integration, statistics, sphere packing, point location, and others.

Chapter 2 reviewed the mathematics for convex hulls. This chapter presents a general dimension algorithm with good average case performance. The algorithm uses precise data and arithmetic. Like other algorithms designed for precise arithmetic, it does not work with imprecise data or arithmetic. At the end of this chapter, we present some of the problems caused by imprecision. In the next chapter, we extend the algorithm to handle imprecision.

1. Quickhull and beneath-beyond algorithms

The 2-d Quickhull algorithm [Eddy 1977; Bykat 1978; Green & Silverman 1979; Floyd 1976] incrementally constructs a convex hull. Like its namesake, Quicksort, Quickhull repeatedly partitions the point set. At the beginning of each iteration, Quickhull associates an *outside list* of points to each edge. Each point on an outside

```
create an initial hull from a simplex of points
partition points into the edges' outside lists.
for each edge with a non-empty outside list
    pick the leftmost, furthest point on the list
    form a triangle of the point and edge
    remove any points below the two new edges
    partition the remaining points into the new edges' outside lists
    replace the old edge with the two new edges
```

Table 4.1: 2-d Quickhull algorithm for precise data and arithmetic

list is above the corresponding edge (see Table 4.1).

A *furthest point* is the point on an outside list that is furthest from the facet's hyperplane. A *processing level* of Quickhull consists of processing all old facets before processing a newly created facet. Consider adding the furthest point to the convex hull. Quickhull partitions the remaining points on the outside list into the two newly created edges. Consider those points that are outside one of the edges. Unless the input is designed to be bad for Quickhull, about half of the points will go to one edge and the other half to the other edge. If the partition is always bad then Quickhull runs in $O(n^2)$ time. But the normal case is the length of long outside lists decreasing geometrically per processing level.

LEMMA 4.1. *If the length of long outside lists decreases geometrically per processing level, Quickhull runs in $O(n \log h + h)$ time where n is the size of the input and h is the size of the output.*

PROOF. The number of processing levels is $O(\log h)$. A point is partitioned no more than once per processing level. The total number of edges created is $O(h)$. Each step occurs either a constant number of times per edge or per partitioning of a point. ∎

In 2-d, the furthest point above an edge is a vertex of the convex hull, so points can only be above one of the new edges. This is not true in 3-d and higher. Consider the convex hull of many points on a sphere. The initial hull would be a tetrahedron

```
create an initial hull from a simplex of points
for each unprocessed point
    delete facets that are below the point
    add the point to coplanar facets
    add the point to horizon faces
```

Table 4.2: Beneath-beyond algorithm for computing the convex hull with precise data and arithmetic.

of four of those points. Points above one edge of the tetrahedron would be above two of its facets.

The Beneath-beyond algorithm [Kallay 1981; Preparata & Shamos 1985] constructs a convex hull for any dimension. It is named after terminology in [Grünbaum 1967]. A point is *beneath* (resp. *beyond*) a facet if the point is below (resp. above) the facet[1]. A *coplanar facet* is a facet that is coplanar with the point. A *horizon face* is a face that has an incident facet below the point and another incident facet above the point. A point is added to a face or facet by taking the convex hull of their union. Table 4.2 gives the algorithm.

Beneath-beyond is a direct implementation of the following theorem [Grünbaum 1967, Th. 5.2.1]:

THEOREM 4.2. *Let H be a convex hull in $\mathbf{R}^d$, and let p be a point in $\mathbf{R}^d - H$. Then G is the convex hull of $p \cup H$ where the facets of G are defined by:*

1. *A face f of H is a face of G iff there is a facet F of H such that $f \prec F$ and p is below F.*

2. *If f is a face of H, then the convex hull of $p \cup f$ is a face of G iff either (a) p is in the affine hull of f, or (b) p is above one facet of H containing f and below another.*

Beneath-beyond explicitly modifies the facial structure of a convex hull. A simpler version suffices if faces may be simplicial complexes. Consider the simplicial

[1]To avoid confusion, we use "above" and "below" in this thesis instead of "beneath" and "beyond".

```
create an initial hull from a simplex of points
for each unprocessed point
    find a facet below the point
    find the horizon of facets above or coplanar with the point
    make a cone of new facets from the point to the horizon
    merge coplanar facets
    replace facets within the horizon by the cone
```

Table 4.3: Algorithm for modified Beneath-beyond on facets with precise data and arithmetic.

complex for a face. Call its simplices, *cofaces* of the face. All cofaces of a face belong to the same affine hull. Similarly let a *cofacet* be a $(d-1)$-dimensional coface. Cofacets can be merged after each iteration or as a post-processing step. Lower dimensional cofaces can be merged at the same time.

THEOREM 4.3. *Let H be a convex hull in $\mathbf{R}^d$ and let p be a point in $\mathbf{R}^d - H$. Then G is the convex hull of $p \cup H$ where the facets of G are defined by:*

1. *A facet F of H is a facet of G iff F is above p*
2. *A facet not of H is a facet of G iff it has a simplicial decomposition of (a) cofacets of a facet F of H coplanar with p, and/or (b) simplices whose apex is p and whose base is a ridge with one incident facet below p and the other incident facet above or coplanar with p.*

PROOF. Let each face of H be a simplicial complex of cofaces. If a facet is in H (resp. G) then all of its faces are in H (resp. G). By Theorem 4.2, if p is below a facet F in H, F is also in G. Also from Theorem 4.2, if a ridge r has one neighbor above p and the other below p, the simplex of r with p is a cofacet of G. By the convexity of a facet, all ridges in the affine hull of a cofacet are bases of cofacets in the same facet. If p is coplanar with a facet F in H, it is in its affine hull. Since F's ridges are also in the affine hull, all cofacets belong to the same facet. ∎

By Theorem 4.3 a simpler version of beneath-beyond suffices (see Table 4.3).

```
create an initial hull
partition points into the facets' outside lists.
for each facet with a non-empty outside list
    pick the furthest point on the list
    find the horizon of facets above or coplanar with the point
    make a cone of new facets from the point to the horizon
    merge coplanar facets
    remove any points below the cone
    partition the remaining points so that each point is on one outside list
    replace facets within the horizon by the cone
```

Table 4.4: Quick_hull algorithm in $\mathbf{R}^d$ with precise data and arithmetic.

Each step of this algorithm is subsumed by steps in the Beneath-beyond algorithm, so the analysis of Preparata & Shamos [1985] carries through:

LEMMA 4.4. *The worst-case complexity of modified Beneath-beyond is $O(nM)$ where n is the number of points and M is the maximum number of faces.*

2. Quick_hull with precise data and arithmetic

The 2-d Quickhull algorithm is extended to $\mathbf{R}^d$ by replacing its triangulation step with one iteration of Beneath-beyond (see Table 4.4). Unlike 2-d Quickhull, the furthest point for a facet is not necessarily an extreme point of the entire point set. If an extreme point is above multiple facets, the partitioning step assigns it to only one facet. The furthest points for neighboring facets may be interior to this extreme point[2].

Guibas et al [1990] and Clarkson et al [1992; cf. Fortune 1992a] give a convex hull algorithm which performs the same steps as Quick_hull but in a different

[2]The partitioning step could assign a point to multiple facets if a heap stored the furthest point for each facet. This would complicate the data structure and add a $O(n \log n)$ factor to the average-case complexity analysis. Empirically in 3-d, the furthest point is almost always an extreme point.

order. They use beneath-beyond on a random permutation of a point set in general position. They retain the old triangulations to speedup selection of a visible facet. Chazelle [1991] derandomizes the algorithm to get an asymptotically optimal deterministic algorithm ($O(h_{\max} + n \log h_{\max})$ where $h_{\max}$ is the maximum size of the output). Boissonnat and Devillers-Teillaud [1989] use a similar method for Delaunay triangulations.

These algorithms locate a facet that is below the point (if any), by traversing the DAG of old triangulations. Quick_hull tests exactly the same set of hyperplanes for a point, but it organizes the tests differently. Instead of testing all hyperplanes for a given point, it tests all points for a given hyperplane. In Quick_hull the partitioning happens after adding each new point. In the other algorithms, "partitioning" happens before adding a new point. We prefer the former because:

- Quick_hull maintains a list of outside points for each facet while the others maintain a DAG of old triangulations. The lists are often smaller than the DAG. The size of the DAG is $O(n^{\lfloor d/2 \rfloor})$, or about d times the size as the final output [Fortune 1992a].

- Empirically, the average time complexity of Quick_hull is $O(n \log h + h)$ where h is the size of the output. Consider the convex hull of a sphere of points inscribed in a cube. Quick_hull runs in $O(n)$, while the other algorithms run on average in $O(n \log n)$. Output sensitivity becomes increasingly significant as the dimension increases.

- Empirically, Quick_hull inserts fewer interior points into the convex hull than the other algorithms. As shown above, Quick_hull only processes points that are extreme points of the points assigned to a facet. If a vertex of the convex hull is above just one facet of the preceding hull, Quick_hull removes all interior points to its cone. The other algorithms may insert some of the interior points before the vertex. As shown in Chapter 6, selecting extreme points is also important for bounding the error due to fixed precision arithmetic.

LEMMA 4.5. *With precise data and arithmetic, if an extreme point of the convex hull is above two or more facets at a partition step in Quick_hull, it must be processed irrespective of which facet it is assigned to.*

PROOF. Assume the contrary and consider the furthest point whose cone is above or coplanar with the extreme point. The visible facets for the furthest point include all facets that the extreme point was above. The extreme point is not above the cone so it must be inside the convex hull. This contradicts its membership in the convex hull. ∎

THEOREM 4.6. *With precise arithmetic and data, the Quick_hull algorithm produces the convex hull of a set of points in* $\mathbf{R}^d$.

PROOF. The Quick_hull algorithm is a specialization of the simplified, beneath-beyond algorithm for convex hulls. In particular, Quick_hull partitions the points into new facets and picks furthest points for processing. After a visible facet is located in beneath-beyond, the processing is the same for both algorithms. By Lemma 4.5, partitioning can not prevent an extreme point from being processed. The termination conditions are the same, so the correctness of beneath-beyond proves the correctness of Quick_hull. ∎

LEMMA 4.7. *The worst case complexity of Quick_hull is* $O(n^2M)$ *where* n *is the number of points and* M *is the maximum number of faces.*

PROOF. Quick_hull is the same as Beneath-beyond except for the partitioning steps. At worst, n iterations can occur and each partitioning involves every facet. ∎

This worst-case complexity is pessimistic because partitioning is usually balanced and processed points are likely to be extreme points. In Chapter 6, we prove that if some balance conditions hold, Quick_hull runs in $O(n \log h + h)$ time where n is the size of the input and h the size of the output.

LEMMA 4.8. *The Quick_hull algorithm on point sets in* $\mathbf{R}^d$ *adds a point to the hull only if it is an extreme point of the points assigned to a facet.*

PROOF. A Quick_hull iteration processes the point furthest from the facet's hyperplane. Since points are in general position, all points assigned to a facet are inside the affine hyperplane through the furthest point. ∎

When implementing the algorithm in $\mathbf{R}^d$, the following is useful:

- If possible, build the initial hull from d maximum points, i.e., those with minimum or maximum coordinates.
- Process the remaining maximum points first. This quickly builds an approximation to the hull. If points are randomly distributed in a square then diagonal maximum points will eliminate all but $O(\sqrt{n})$ points [Golin and Sedgewick 1988].
- Remove and partition points in a single step.
- During the partition step, identify the next furthest point for each new facet.
- When identifying the horizon, merge all facets below the point. This leaves a single old facet for the replacement step. (This should not be done with imprecise data).

3. Imprecise predicates and convex hull algorithms

The next chapter shows how to adapt Quick_hull for imprecise data and arithmetic. Most of the changes are in the merge step. As it stands, Quick_hull, like many other convex hull algorithms, does not work for imprecise arithmetic. This section discusses some of the problems that may arise.

Consider the convex hull of cospherical points. If all but one point is restricted to a ring, the convex hull looks like an ice cream cone with a bite taken out. If the diameter of the disk is narrowed down to $2 \cdot 10^{-6}$, the disk is nearly flat. If the disk contains 20,000 points, most adjacent simplices do not form clearly convex

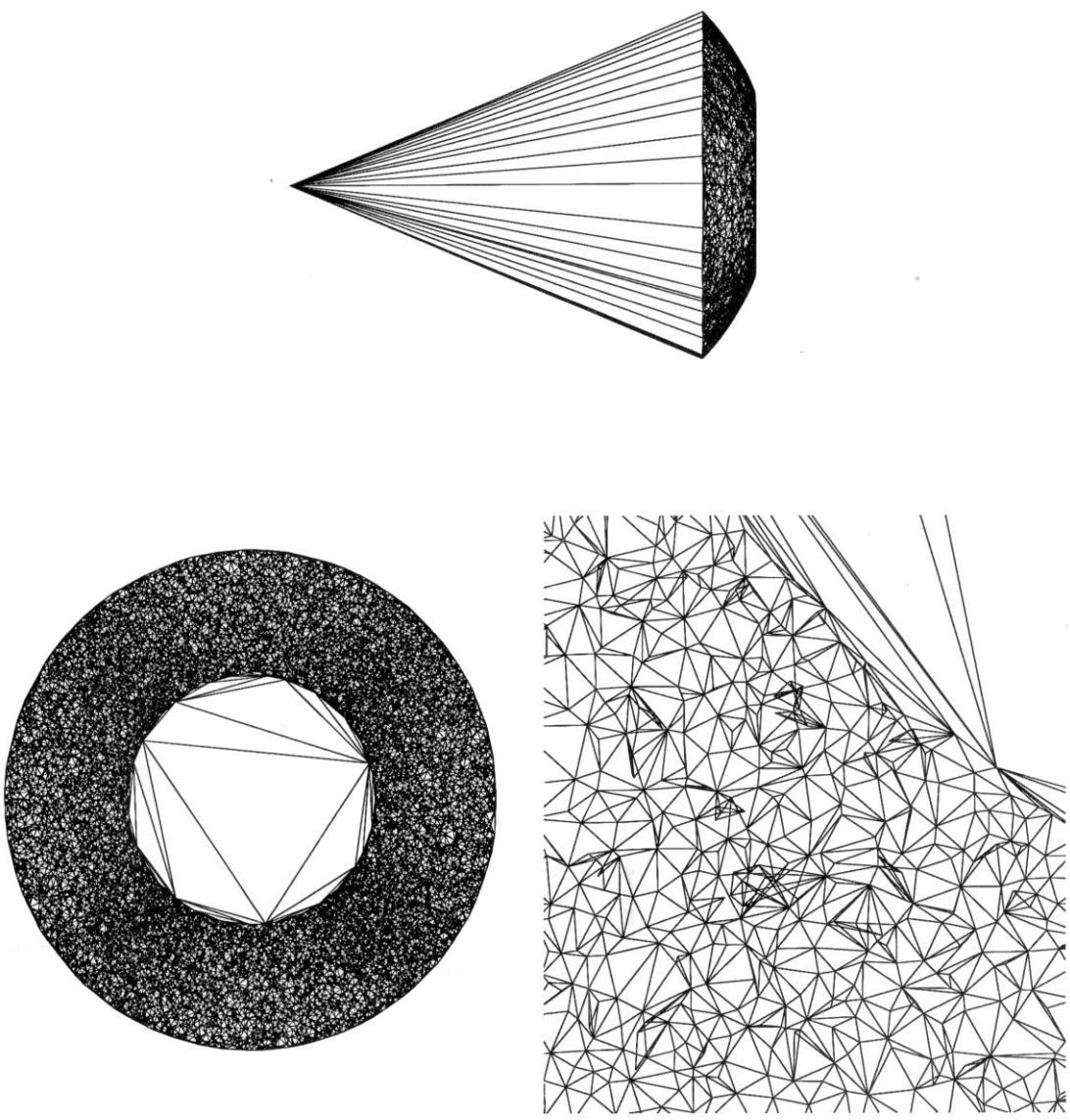


Figure 4.1: Erroneous convex hull of 20,001 cospherical points on the unit sphere. A $2 \cdot 10^{-6}$ disk contains 20,000 points. The right hand figure is a 10^{-7} closeup. At this scale, the other point is over 800 miles away.

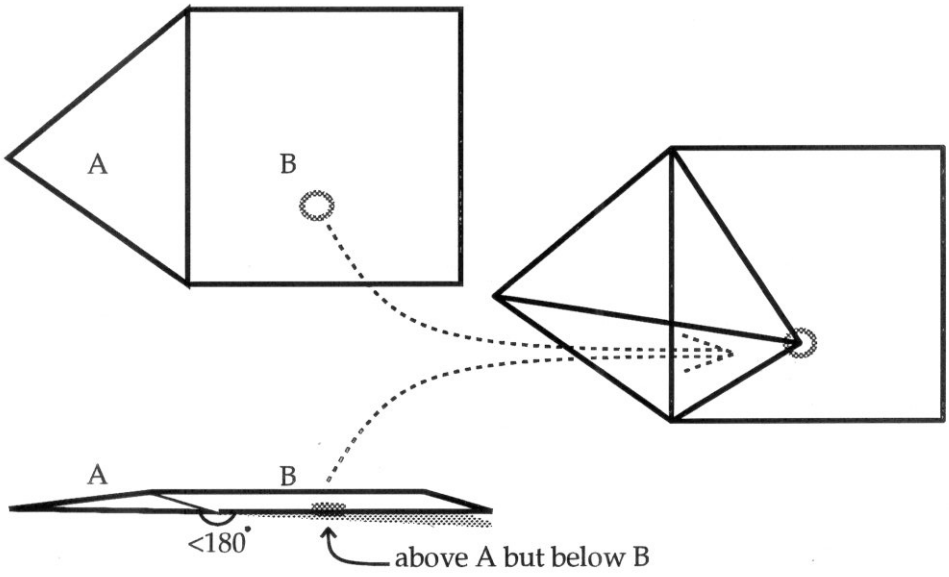


Figure 4.2: Roundoff error may incorrectly report that a point is below a facet that it should be above. If classification is inconsistent, a spike can result.

angles. The precise algorithm for Quick_hull fails (Figure 4.1). One reason is that points above two facets may be computed as above one facet and below the other. Triangulating such a point leads to a facet oriented in the reverse direction (Figure 4.2). Exactly the same problem will happen in Beneath_beyond and related algorithms [e.g., Guibas et al. 1990; Clarkson et al. 1992].

The Graham Scan algorithm for 2-d convex hull [Graham 1972] compares polar angles to sort points radially around an inside point. It tests for convex angles to produce the convex hull. Both computations are equivalent to testing the orientation of a triangle by the sign of a determinate (see "orientation of a simplex†"). Sorting is by the angle. For collinear points, sorting is by distance from origin. When building the hull, collinear points are treated the same way as concave points. If collinearity is determined with precise arithmetic, Graham scan produces a convex hull. With imprecise arithmetic, it may fail.

Consider a large number of uniformly placed points on one turn of a clockwise spiral (Figure 4.3). They could be sorted for Graham Scan from innermost point to outermost. If so the algorithm produces a degenerate hull of the first and last

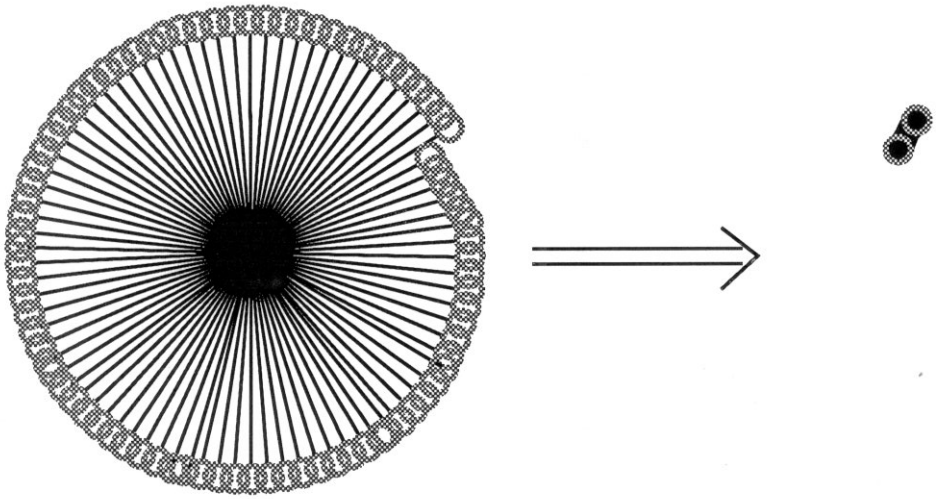


Figure 4.3: If the Graham scan algorithm ignores roundoff error, sorting could be in reverse order. If so, a degenerate hull is built.

points.

Chand and Kapur's [1970] gift-wrapping method adds a facet to a contiguous set of facets by rotating a hyperplane around an outside ridge. The maximum rotation defines the next facet. Consider a large number of points on a sphere. In the precise case, each point becomes a vertex of the convex hull. Consider what happens if gift-wrapping treats near-zero computations as zero. Then nearly coplanar points combine into a single facet. This can cause problems when gift wrapping comes close to an old facet. The old facet's vertices may be below the current facet, but a new facet may undercut the old facet. Figuratively, imprecision causes wrinkles in the surface where adjacent facets do not match up. (Figure 4.4).

Other convex hull algorithms suffer similar faults under fixed precision arithmetic. The options are to ignore nearly coplanar points and get concave facets, or to treat near-coplanarity as exact coplanarity and get inconsistent results.

Several 2-d algorithms work under fixed precise arithmetic. Fortune uses weak predicates for testing orientation of 3 points [Fortune 1989]. His analysis guarantees that the convex hull is the exact convex hull for a slightly perturbed set of points. Li and Milenkovic [1990] make two passes over the points to guarantee that convexity is preserved despite perturbations of the extreme points and all points are inside

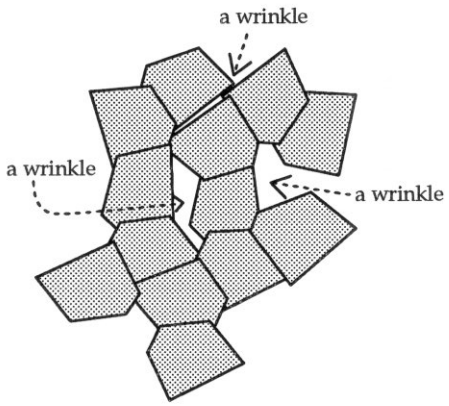


Figure 4.4: Gift-wrapping with nearly coplanar points treated as coplanar points. Non-triangular facets may not meet in *clearly convex* angles.

or nearly inside their convex hull. Guibas et al [1990] produce similar results with tighter error bounds by considering the hull as a whole.

Beichl & Sullivan [1992] have implemented a 3-d Delaunay triangulation program. They first scale the data and apply a small random perturbation to get rid of degeneracies in the input. This may introduce new degeneracies but for many point sets, the probability is low. They then project the point set to a paraboloid and find its convex hull. They use a variation of the gift-wrapping algorithm [Chand & Kapur 1970]. They pick a "shelling" order for adding tetrahedron to the triangulation. The key to their algorithm is using QR factorization for the InSphere test and shelling order. QR factorization is a stable numeric method for solving the least squares problem. Their code works for many cases but they do not have an error analysis.

A review of the literature did not find a 3-d convex hull algorithm that worked with fixed precision arithmetic.

Chapter 5

Quick_hull with imprecision

As discussed above, imprecise data and arithmetic can invalidate the assumptions of convex hull algorithms. This chapter presents a convex hull algorithm that produces a box complex. Like Point_in_polyhedron, the algorithm allows for imprecise data, fixed precision arithmetic, and singularities. It is built on the precise arithmetic version of Quick_hull.

If points are in general position, Quick_hull produces the convex hull of the points. In this case, the steps performed by Quick_hull are the same as those performed by the precise arithmetic version of Quick_hull. The only difference is that Quick_hull tests every ridge for convexity. With precise arithmetic, these tests are superfluous since every pair of adjacent facets must be convex. With imprecise data and fixed precision arithmetic, convexity can not be guaranteed.

If points are not in general position, Quick_hull produces a convex box complex that approximates the convex hull. It uses δ-boxes. Informally, it is a polytope with "thick" facets. The trace of its facets contain all possible convex hulls of the points.

At the end of each iteration, Quick_hull produces a locally convex box complex. The iteration is as before: pick a furthest point, find its horizon, make a cone of new facets to the horizon, and merge non-convex facets. This chapter discusses the changes that must be made to the last step. The next chapter gives an error analysis that bounds the maximum width of a facet when certain conditions are meet. Chapter 7 discusses the implementation of Quick_hull and gives examples of

its output.

The key to Quick_hull is how it merges non-convex facets. If this is done indiscriminately, the width of a facet can become as wide as the diameter of the point set. The steps used by Quick_hull are: retriangulate concave ridges, merge and redefine coplanar, newly created facets, merge remaining non-convex new facets, merge non-convex facets across the horizon, and incorporate the new facets if they are better than the old facets.

1. Introduction

To handle imprecise data and arithmetic either the input or output domain is restricted. Hopcroft and Kahn [1989] restrict the input domain. They intersect a convex polyhedron with a half space using imprecise coordinates and arithmetic. Their input is a polyhedron in general position, i.e., no four vertices are coplanar nor three vertices collinear.

Bentley and Faust [1982] produce an approximate convex hull by subdividing the plane into strips, finding the outlying points, and computing the convex hull of these points. In 3-d, they subdivide space into cubes.

Sugihara and Iri [1989] restrict the output of their Voronoi diagram algorithm. While they guarantee topological correctness (each site has one Voronoi region and neighboring regions share one edge), geometric correctness is only guaranteed for precise arithmetic. If arithmetic is imprecise, they make a best effort at correctness. With single-precision floating point and careful implementation of the incircle test, they produced a Voronoi diagram that appears correct for one million, uniformly distributed points in the unit square.

We prefer the results of Li and Milenkovic's [1990] and Guibas et al's [1990] 2-d convex hull algorithms: guaranteed topological correctness and guaranteed bounds for geometric error. Users of their algorithms can depend on strong convexity constraints and bounded errors for points. We develop a convex hull algorithm that satisfies similar constraints. It differs from theirs by using boxes to define an approximate convex hull. It also works in $\mathbf{R}^d$ as well as $\mathbf{R}^2$.

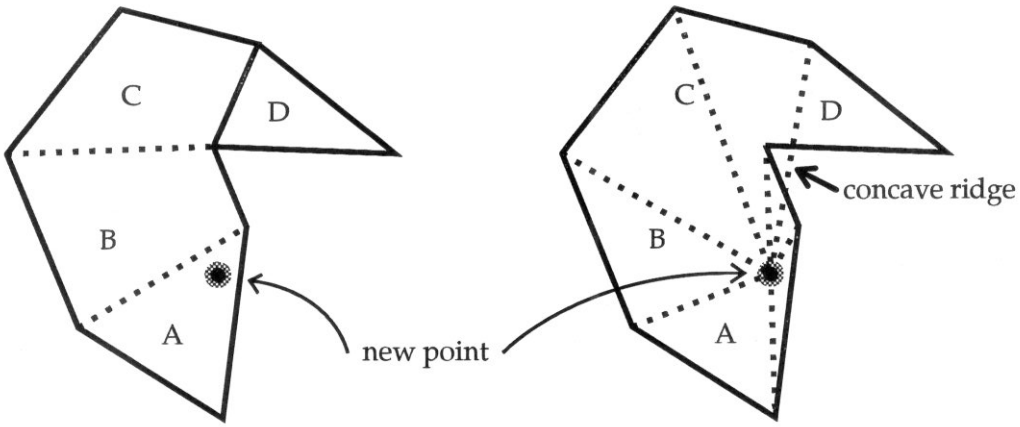


Figure 5.1: A non-convex facet may cause a concave ridge.

In the precise version of Quick_hull, imprecision can cause the following problems:

1. The horizon may not be convex. It may contain concavities and folds. For example Figure 5.1 shows three facets (A, B, C) that are below a point. Triangulating the horizon causes a concave ridge.

2. Coplanarity is not an exact relationship. For example, Figure 5.2 shows one end of a cylinder of points. If a point is added above the end, the ridges may be too flat to be *clearly convex*.

3. New facets may be concave or coplanar with old facets across the horizon.

None of these cases could happen with precise data and arithmetic. For example in Figure 5.1, the new point is above facet C and coplanar with D. With exact data, it would have to be above D as well.

This chapter resolves these problems by expanding the step *merge coplanar facets*.

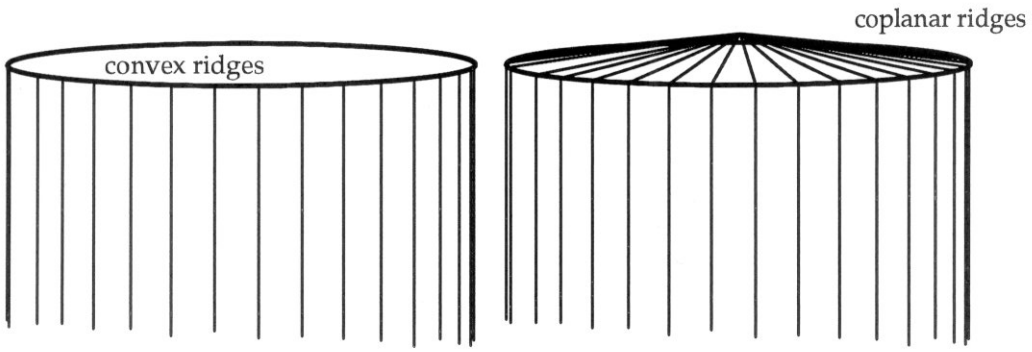


Figure 5.2: Adding a point to a cylinder of facets may cause many coplanar facets.

2. The Quick_hull algorithm

Figure 5.3 shows Quick_hull at the start of one iteration. It has selected the furthest point above one of the facets. The furthest point is called $P_{furthest}$. Facets below $P_{furthest}$ are *interior facets*; the others are *exterior facets*. The contour of the interior facets is the *horizon*. Its ridges are *horizon ridges* and its vertices are *horizon vertices*. The exterior neighbor of an horizon ridge is an *horizon facet*. The other exterior facets incident to an horizon are *near-horizon facets*.

Quick_hull builds a *cone* of new facets to replace the interior facets (Figure 5.4). Each new facet has a hyperplane defined by d points called the facet's *simplex*. The simplex has an *apex* (usually $P_{furthest}$), and a *base* (usually an horizon ridge). The simplex's vertices are the *simplicial vertices* of a facet. The facet's other vertices are its *coplanar vertices*. A *cone facet* uses $P_{furthest}$ as its apex and horizon vertices as its base. A *chord facet* is a facet of horizon vertices. A *flipped facet* includes $P_{furthest}$ as a coplanar vertex.

For each facet, Quick_hull classifies non-vertex points into two lists: the outside and coplanar list. Points on an *outside list* are *clearly above* the corresponding facet's hyperplane and *maybe above* its outer plane. $P_{furthest}$ is the furthest point on an outside list. Points on a *coplanar list* are *clearly below* the facet's outer plane and *maybe above* the facet's inner plane. The remaining points are *clearly below* the all outer planes. Points can be on multiple coplanar lists but only on one outside list. If a point is above multiple facets, it was placed on the outside list of the facet that it is furthest above. See Table 5.1 for the algorithm.

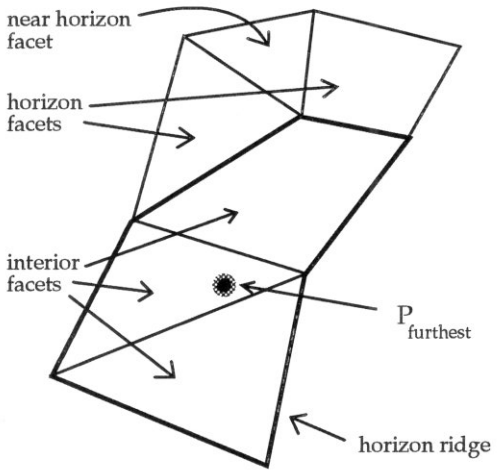


Figure 5.3: Quick_hull after finding the horizon for $P_{furthest}$. There are three interior facets below $P_{furthest}$ and six horizon facets that are not below $P_{furthest}$.

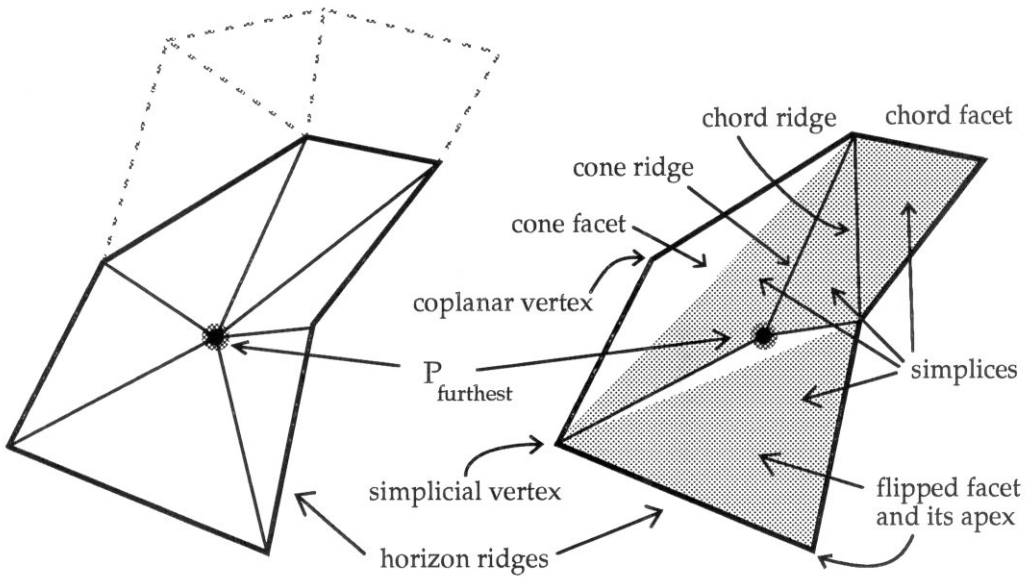


Figure 5.4: Quick_hull after creating a cone of new facets and after completing an iteration. In between, it retriangulates a concave ridge and merges two pairs of coplanar new facets. The simplex for one pair was flipped. All horizon ridges are convex.

create an initial hull	(*initial hull*)
partition points into outside and coplanar lists	(*partition all*)
while there exists a furthest point $P_{furthest}$ for any facet F	*repeat*
find the horizon for $P_{furthest}$ starting with F	(*find horizon*)
make a cone of new facets from $P_{furthest}$ to the horizon	(*make cone*)
merge concave and coplanar ridges	(*merge cone*)
merge concave and coplanar new facets	(*fix cone*)
if cone is better than the old, interior facets[†a]	*if better*
partition points into outside and coplanar lists	(*partition points*)
attach the cone to the horizon[‡]	
merge coplanar horizon facets	(*fix horizon*)
else	*else*
raise outer planes to clearly include $P_{furthest}$[‡]	
partition outside lists of modified facets[‡]	(*partition outside*)
	when done
raise outer planes to include coplanar points and vertices	(*raise outers*)

[a]The symbol "‡" is the flag for operations defined in the "Operations" section at the end of this chapter.

Table 5.1: Quick_hull algorithm using imprecise data and arithmetic. The corresponding procedure names are in parenthesis.

The rest of this chapter expands each step of Quick_hull, proves that Quick_hull produces a convex box complex, and determines its worst-case computational complexity. We ignore the cost of combinatorial operations. This chapter uses operations and computations defined in "Basic computations and operations" of Chapter 2, and other operations defined at the end of this chapter.

We first give the conditions satisfied by Quick_hull and then list its parameters.

LEMMA 5.1. *Quick_hull produces a convex box complex of the set of points if the following conditions are satisfied:*

1. *initial hull produces a convex box complex,*
2. *outside lists contain all points maybe above an outer plane,*
3. *no operation creates new outside points,*
4. *each iteration produces a locally convex box complex, and*
5. *raise outers produces a convex box complex.*

PROOF. At this level of detail, what matters is the outside points. Each iteration removes at least one point from the outside lists. Since new outside points are not created, Quick_hull terminates when no point is outside. ∎

Input parameters		Output parameters	
n	input size	h	hull size
d	dimension	δ	max facet width
Δ	input diameter	m	max facet size
β	machine roundoff	ϵ_β	max roundoff error
ρ	point precision	ϵ_ρ	max centrum precision
θ	max angle	M	max interior size, etc.
ζ	max twist	V	max embedded, concave ridges

Table 5.2: Input and output parameters for Quick_hull. See text for explanation.

3. Quick_hull parameters

The complexity analysis in this chapter and the error analysis in the next chapter uses the input and output parameters in Table 5.2.

The input to Quick_hull is a set of n points in $\mathbf{R}^d$. In order to bound the maximum output size to $O(n^{\lfloor d/2 \rfloor})$, $d \leq n^{1/2+\xi} - 1$, $(\xi \approx 0)$. The diameter of the point set is Δ. We assume that a Δ ball around the origin includes all of the points. The machine roundoff is β. The maximum precision of an input point is ρ. The user may specify the maximum convex angle θ and maximum twist ζ. If a maximum convex angle is given, it is at least $3\pi/2$ (for error analysis). Twist measures how twisted two facets are relative to their common ridge. We use the maximum twist to bound the maximum facet width. See "twist of a merge‡" for more details.

The output of Quick_hull is a convex box complex. The total number of facets and ridges is h. The maximum value of h is $O(n^{\lfloor d/2 \rfloor})$ [Klee 1966]. The maximum facet width is δ. The maximum number of ridges in a facet during Quick_hull is m. The maximum roundoff error in testing distances for coplanarity is ϵ_β. Using the sensitivity analysis in Chapter 2, it can either be recorded for each computation, or bounded in terms of Δ. The maximum computed size of a centrum is ϵ_ρ. This includes ρ, the roundoff error in computing the centrum, and the roundoff error in testing if the centrum is below a hyperplane. The maximum number of embedded

concave ridges in *merge cone* is V.

The parameter M is the maximum of several parameters: M_{interior} the maximum number of interior facets, M_{merge} $1/m$'th of the maximum number of merges in *merge cone* or *fix horizon*, and M_{raise} the maximum number of facets coplanar with a point in *raise outers*. An upper bound for m and M is h; empirically m and M are small. In 2-d, $m = 2$.

LEMMA 5.2. $\epsilon_\rho = \rho + 2\epsilon_\beta$.

PROOF. Since the centrum is computed, its center point can be up to ϵ_β from the hyperplane. Its radius is ρ. Testing a neighboring hyperplane can introduce an additional error of ϵ_β. ∎

LEMMA 5.3. *In 2-d,* $\epsilon_\beta = 3\Delta\beta$+*2nd order terms. In 3-d,* $\epsilon_\beta = 7\Delta\beta$+*2nd order terms.*

PROOF. From "if P is clearly above†"[1], the sensitivity of the distance computation in 2-d is

$$\sigma = \sum_{i=1,2} |h_i p_i| + |\sum_{i=1,2} h_i p_i| + |\sum_{i=1,2} h_i p_i + h_3| + 2nd \text{ order terms.}$$

ϵ_β bounds the roundoff error when the distance computation is nearly zero. So the third sum is nearly zero and the second sum is nearly h_3. The normal equation H is normalized so $h_1, h_2 \leq 1$. Expanding the first sum gives three 1st-order terms, each less than Δ. In 3-d the sensitivity is:

$$\sigma = \sum_{i=1..3} |h_i p_i| + |\sum_{i=1,2} h_i p_i| + |\sum_{i=3,4} h_i p_i| + |\sum_{i=1,3} h_i p_i + h_4| + 2nd \text{ order terms.}$$

Similarly, the last sum is nearly zero and the first three sums are less than 7Δ. ∎

[1]The symbol "†" is the flag for computations and operations defined in the "Basic Computations and Operations" section at the end of Chapter 2.

LEMMA 5.4. *In 2-d, $\zeta = 2$ and $M = 3$.*

PROOF. Each edge has exactly two vertices and its centrum is the arithmetic mean of the vertices, yielding $\zeta = 2$. Since $P_{furthest}$ is the furthest point above an edge, outside points are above exactly one edge with precise arithmetic. With imprecise arithmetic, $P_{furthest}$ is is above at most two edges because each centrum is *clearly below* its neighboring edges. Since each edge has exactly two vertices, the horizon is exactly two vertices. A coplanar point or edge is coplanar with at most one edge in the clockwise or in the counter-clockwise direction. A point is one side or the other of a vertex so the number of non-convex horizon edges is at most three, and the number of coplanar edges for a point is at most three. ∎

We now discuss each step of Quick_hull. See Table 5.1 for the overall algorithm.

4. *Initial hull* **and** *partition all*

Initial hull starts with the subset of points that have a maximum or minimum coordinate in some dimension (called the *maximum points*). In the case of ties, one point is selected arbitrarily. A point set in $\mathbf{R}^d$ has between 1 and $2d$ maximum points. Using this subset as a starting point, Quick_hull produces a convex box complex. Table 5.3 gives the algorithm.

Up to $d - 1$ maximum points may be unused after creating the initial hull. These may be added before executing *partition all* by the same Quick_hull iteration as normal points. In 3-d this can quadruple the volume of the initial hull while doubling the number of initial facets. If done, the number of facets for *partition all* is $O((2d)^{\lfloor d/2 \rfloor})$ instead of $O((d+1)^{\lfloor d/2 \rfloor})$ so the computational complexity of *partition all* is not changed. The total computational complexity for the other steps of Quick_hull remains the same.

The explicit test for convexity gives us:

LEMMA 5.5. *Initial hull terminates. If it succeeds, it creates a locally convex box complex.*

```
let S be a most distant pair of maximum points.‡
until maximum points exhausted or S contains d + 1 points
    determine a maximum point P furthest from the flat through S‡
    add P to S
until all points exhausted or S contains d + 1 points
    determine any point P furthest from the flat through S‡
    add P to S
if S contains less than d + 1 points
    fail Quick_hull
create an initial hull from S‡
if the initial hull contains non-convex ridges‡
    fail Quick_hull
partition the remaining points into outside and coplanar lists‡
```

Table 5.3: Algorithm for *initial hull* and *partition all.*

The most expensive operations are determining the maximum points and, if needed, determining the furthest of any point to a flat through S. These operations have a computational complexity of $O(dn)$. The expensive part of creating the initial hull is creating d hyperplanes at $O(d^3)$ per plane. The last section of this chapter gives the computational complexity for each operation flagged with "‡". The other operations are less expensive, yielding:

LEMMA 5.6. *The worst case computational complexity for initial hull independent of partition all is* $O(dn + d^4)$.

Partition all removes interior points from further consideration and initializes the outside and coplanar lists. If a point is *clearly above* a hyperplane it appears on one outside list. Otherwise the point appears on all appropriate coplanar lists. *Partition all* also identifies the point on each outside list that is furthest above its hyperplane. For more details and a complexity analysis, see *partition points* below.

```
let P_furthest be the furthest point on facet F's outside list
let S_i be a list of interior facets initialized to F
let S_h be a list of horizon ridges, initially empty
for each facet F in S_i
    for each unprocessed ridge R in F‡
        let F_n be R's neighboring facet
        if P_furthest is clearly above F_n's hyperplane†
            append F_n to S_i
        else
            append R to S_h
```

Table 5.4: Algorithm for *find horizon*.

Other initial sets of points can be used in place of maximum points[2]. For example, Golin & Sedgewick [1988] use the diagonal maximum points; these are particularly effective for axis-aligned point sets. Another possibility is the set of maximum +1, -1, and 0 combinations of coordinates. A larger set of initial points usually creates an initial hull with more facets and a larger interior. This makes *partition all* more expensive while increasing the probability that a point is immediately rejected. Careful design of *partition all* can reduce the number of computations, especially in 2-d [Golin & Sedgewick 1988; Kao & Knott 1990]. The optimal initial set and *partition all* depends on the point distribution and is not explored here.

5. *Find horizon*

After initialization, every point that is outside the hull is on an outside list. Each outside list includes a point, $P_{furthest}$, that is furthest above the corresponding hyperplane. *Find horizon* takes $P_{furthest}$ for a facet and returns a horizon of ridges. Table 5.4 gives the algorithm.

[2]A poor choice for the initial set of points is the first $d+1$ non-degenerate points. If used, *partition all* would locate the points furthest from each facet. Like maximum points, these points would be on the convex hull but the cost of finding them is relatively high.

LEMMA 5.7. *Find horizon terminates and returns a non-empty horizon. The horizon is the contour of the facets that are clearly below* $P_{furthest}$.

PROOF. *Find horizon* terminates when the hyperplanes of all neighboring facets are *maybe above* $P_{furthest}$. The hyperplanes of all the facets define a convex polytope, so $P_{furthest}$ can not be clearly outside all of them. There is at least one hyperplane it is *maybe below*. At worst, that facet's ridges will be the horizon. The horizon is a contour by construction. ∎

LEMMA 5.8. *The total worst-case computational complexity for find horizon is* $O(ndmM)$.

PROOF. *Find horizon* is called at most n times. Testing hyperplane distance takes $O(d)$ operations. Processing occurs for interior facets and horizon facets. Each interior facet has at most m ridges. There are no more than M interior facets and one horizon facet per ridge. ∎

for each horizon ridge R make a new facet F with $P_{furthest}$ as the apex and R as the base orient the new ridges of F to be coherent with R pair up new ridges‡ construct an oriented hyperplane through F's vertices†

Table 5.5: Algorithm for *make cone*.

6. *Make cone*

Make cone takes $P_{furthest}$ and a horizon from *find horizon*, and returns a cone of new facets based on horizon ridges. Table 5.5 gives the algorithm.

LEMMA 5.9. *The new ridges of make cone are coherently oriented with the horizon ridges.*

PROOF. The interior facets are coherently oriented by precondition. This induces an orientation of the ridges and $(d-3)$-faces of the horizon. Since the ridges have the same orientation, each $(d-3)$-face has an opposite orientation in its two ridges. Each new facet is a simplex with a ridge as its base and $P_{furthest}$ as its apex. The boundary of the new ridges consist of a $(d-3)$-face of the horizon and $P_{furthest}$. Since the $(d-3)$-faces have opposite orientation in neighboring new facets, the intervening new ridge has opposite orientation. All new ridges pair up so the orientation is coherent. ∎

LEMMA 5.10. *The union of exterior facets from find horizon and new facets from make cone is a face complex.*

PROOF. Each pair of adjacent horizon ridges share a $(d-3)$-face. Since the initial hull is a simplex and *make cone* is the only part of Quick_hull that adds facets, the $(d-3)$-face is unique. *Make cone* creates new ridges from a $(d-3)$-face and $P_{furthest}$. Since each $(d-3)$-face occurs in two ridges of the horizon, each new ridge occurs twice. Each horizon ridge occurs twice, so the evenness condition holds for ridges. Lower dimensional faces implicitly satisfy the evenness condition. Each new facet's

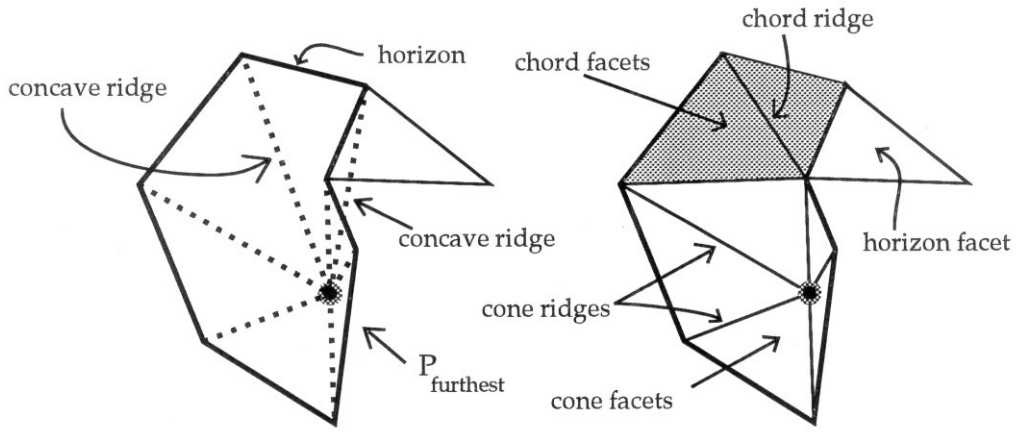


Figure 5.5: Creating chord facets from two concave ridges. The first figure shows the new facets after *make cone*. The second figure shows the chord facets created by retriangulating the concave ridges.

inner and outer plane contains its vertices. The boxes for intermediate features are defined implicitly. By Lemma 5.9 the orientation is consistent. ∎

The only computation is constructing a hyperplane and setting the centrums. The former takes $O(d^3)$ and the later $O(d)$, giving:

LEMMA 5.11. *The total computational complexity of make cone is* $O(nd^3mM)$.

7. *Merge cone*

Merge cone merges the new facets from *make cone* in order to remove coplanar and concave cone ridges. It creates chord facets and flipped facets, and merges cone facets. At the end of *merge cone*, all ridges between cone facets are *clearly convex*. Coplanar facets are merged and a new hyperplane is created (see "if R passes the simplex test‡"). Concave facets are retriangulated (Figure 5.5). Table 5.6 gives the algorithm.

Most of *merge cone* exists for bounding the facet width in 3-d and higher dimensions. In 2-d, there is only one cone ridge ($P_{furthest}$) so either it is *clearly convex* or *make cone* returns an empty cone.

```
while a cone ridge is not convex‡
    let S be the set of non-convex cone ridges

    for each concave ridge R in S
        build a non-concave cone facet F‡
        remove F's ridges from S‡
        build chord facets for F by recursively calling make cone and merge cone‡
        merge non-convex chord facets into F‡

    for each coplanar ridge R in S
        if R fails the simplex test†
            build a flipped facet F from R's neighbors‡
        else
            build a cone facet F from R's neighbors‡
        remove F's ridges from S‡

if there a fewer than d cone ridges
    reject the cone
```

Table 5.6: Algorithm for *merge cone*.

Each iteration of *merge cone* removes a cone ridge by merging facets. It does not create new cone ridges from $P_{furthest}$, so the following lemma holds:

LEMMA 5.12. *Merge cone terminates with a cone of facets from* $P_{furthest}$ *to the horizon.*

LEMMA 5.13. *The total worst-case computational complexity of merge cone is* $O(nd^3mM)$.

PROOF. It costs $O(d)$ to test a ridge. Each processed ridge causes the retesting of no more than $2d-3$ ridges (the ridges of two facets minus their horizon ridges and the processed ridge). The cost of testing is subsumed by the cost of defining a new hyperplane, $O(d^3)$. Inner and outer planes can be adjusted at the end of *merge cone* for a cost of $O(d^2mM)$. By definition of M, no more than mM ridges can be non-convex per *merge cone*. There can be no more than n top-level calls. ∎

8. *Fix cone*

Fix cone takes a cone of new facets from *merge cone*. At the end of *fix cone*, every new ridge is *clearly convex* and every horizon ridge is *clearly convex* or marked for merging in *fix horizon*. Table 5.7 gives the algorithm.

Box complexes are preserved under merging (see "merge facet‡") and non-convex ridges are removed. This proves:

LEMMA 5.14. *At the end of fix cone, all new ridges are clearly convex. The union of new and exterior facets is a box complex.*

LEMMA 5.15. *The total computational complexity of fix cone is* $O(ndMm^3)$

PROOF. The cost of retesting ridges after a merge is subsumed by the cost of finding the best merge. At most mM ridges can be merged at a cost of $O(dm^2)$ per merge. Note that testing only non-convex ridges would reduce the cost of "determine best merge‡". ∎

```
for all non-convex chord ridges, cone ridges, and horizon ridges R‡
    determine best merge (F into F_n) for R†
    if F_n is an horizon facet
        rename F as F_n‡
    else if F is an horizon facet
        mark F for merging into F_n‡
    else
        merge F into F_n†
if there a fewer than d cone ridges
    reject the cone
```

Table 5.7: Algorithm for *fix cone*.

9. *Partition points*

Partition points takes a cone of new facets and a list of outside and coplanar points. The list consists of the outside and coplanar lists for all interior facets. It returns a partitioning of the points into facets of the cone; each point is either *clearly inside*, on one outside list, or on a set of coplanar lists. Table 5.8 gives the algorithm.

Partition all does the same for all points of the input and all facets of the initial hull. *Partition outside* does the same for outside points of modified facets and all interior facets.

LEMMA 5.16. *After partition points, clearly inside points from this or an earlier iteration are clearly below all outer planes.*

PROOF. After *partition all*, a point is clearly inside only if it is *clearly below* all inner planes in the initial hull. So the lemma is initially satisfied. After that, a point becomes clearly inside only if it is *clearly below* all inner planes of the cone. The outer plane for a facet passes above the adjacent inner planes because an inner plane is below vertices that an outer plane is above. Because all inner planes meet at a *clearly convex* angle, an outer plane is above all inner planes. Similarly an outer plane is above all inner planes of interior facets. So an outer plane can not become below a previous inner plane, nor below a previously inside point. ∎

```
for each point P
    let C be a set of coplanar facets initialized to empty
    let F_o be a potential outside facet initialized to empty
    for each facet F
        if P is clearly below F's outer plane†
            if P is maybe above F's inner plane†
                add F to C
        else if P was outside and P is clearly above F's hyperplane†
            if P is further above F than F_o
                set F_o to F
        else /* P coplanar or P straddles F's outer and hyperplane */
            raise F's outer plane to include P†
            add F to C
    if F_o is defined
        add P to F_o's outside list
        if P is further away, update F_o's furthest point
    else if C is non-empty
        add P to the coplanar list of facets in C
    else
        /* P is clearly inside and no longer needed */
```

Table 5.8: Algorithm for *partition points*.

Since the maximum number of new facets is mM and the cost per facet per point is $O(d)$, the following lemma holds:

LEMMA 5.17. *The total worst case complexity of partition points is* $O(n^2dmM)$.

Unless the point set is designed to be bad for *partition points*, Quick_hull should partition the outside points of the interior facets fairly evenly among the outside lists of the new facets. Since there is at least d new facets, the average length of an outside list will often decrease by at least some constant. This is not true for short lists, but then the cost of *partition points* is bounded by the output size.

LEMMA 5.18. *If* mM *is bounded, long outside lists shorten geometrically per processing level, and the number of deleted vertices is a bounded proportion of all vertices, the total complexity of partition points is* $O(nd \log h + dh)$

PROOF. The third clause bounds the maximum size of the box complex to $O(h)$. So after outside lists are short, the remaining partitioning cost is $O(dh)$. Since long outside lists decrease geometrically, the maximum number of levels with long outside lists is $O(\log h)$. Each level partitions n points into no more than mM^2 facets apiece, giving the desired result. ∎

for all non-convex ridges R‡ determine best merge (F into F_n) for R† merge F into F_n†

Table 5.9: Algorithm for *fix horizon*.

10. *Fix horizon*

Fix horizon removes non-convex horizon ridges. When done, all ridges of the hull are clearly convex. Table 5.9 gives the algorithm.

Since *fix horizon* explicitly tests for convexity, the following is true:

LEMMA 5.19. *Fix horizon produces a locally convex box complex.*

LEMMA 5.20. *The total computational complexity of fix horizon is* $O(ndMm^3)$.

PROOF. The cost of retesting ridges after a merge is subsumed by the cost of finding the best merge. At most mM ridges can be merged at a cost of $O(dm^2)$ per merge. ∎

11. *Raise outers*

At the end of each iteration, Quick_hull produces a locally convex box complex. After the final iteration, Quick_hull executes *raise outers* to produce a convex box complex. Table 5.10 gives the algorithm.

LEMMA 5.21. *For a point p, let the set* $S_i(p)$ *be the neighborhood of facets whose inner planes are maybe below p. For all p,* $S_i(p)$ *includes all facets whose outer planes are maybe below p.*

PROOF. Assume that a point P fails this Lemma and consider the realization of P that is furthest above the polytope defined by the outer planes (Figure 5.6). It must be above some facet C. As a coplanar point or vertex, it is below the outer plane of some facet A and above its inner plane. There exists at least one facet

```
for each facet F
    for each unprocessed coplanar point or vertex P of F
        let S_i be a list of interior facets initialized to F
        for each facet F_i in S_i
            for each unprocessed incident facet F_n of F_i‡
                if P is incident to F_n‡
                    add F_n to S_i
                else if P is maybe above F_n's inner plane†
                    add P to F_n's coplanar list
                    raise F_n's outer hull to clearly include P†
                    add F_n to S_i
```

Table 5.10: Algorithm for *raise outers*.

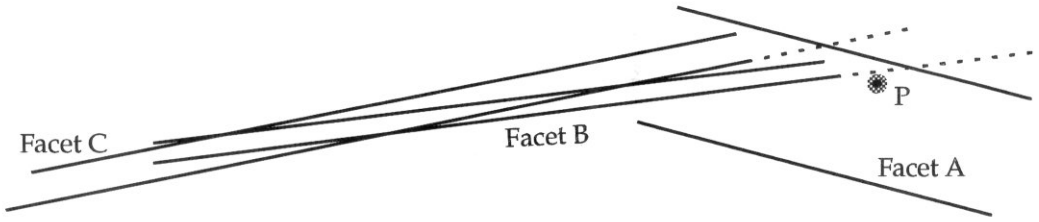


Figure 5.6: A coplanar point can not be above a distant facet. Assume P is above Facet C even though it is below C in the figure.

B between A and C whose inner plane is above P. By local convexity, these facets are convexly related. So the inner plane C is also above P. But this contradicts the belowness of C. ∎

LEMMA 5.22. *Raise outers produces a convex box complex.*

PROOF. By Lemma 5.16, inside points are inside all outer planes. By Lemma 5.21, *raise outers* places coplanar points and vertices below all outer planes. By Lemma 2.13, it produces a convex box complex. ∎

LEMMA 5.23. *The total computational complexity of raise outers is* $O(ndmM)$.

PROOF. There are up to n coplanar points and vertices. The maximum number of inner planes below a point is M, so the maximum facets tested per point is mM. Each test costs $O(d)$. ∎

12. Summary

THEOREM 5.24. *Quick_hull produces a convex box complex in worst case time:*

$$O(nd^3mM + nMdm^3 + n^2dmM).$$

PROOF. At the end of *fix horizon*, every ridge is *clearly convex* (Lemma 5.19). At the end of *raise outers*, the box complex is convex (Lemma 5.22). The termination lemma for Quick_hull (Lemma 5.1) completes the correctness proof for Quick_hull.

Except for initialization, *find horizon* and *raise outers*, each step of Quick_hull contributes to the total computational complexity. *Make cone* and *merge cone* create hyperplanes at a cost of $O(d^3)$. Determining the best facet can cost $O(dm^2)$ in *fix cone* and *fix horizon*. If all outside points are always above one facet, *partition points* can partition each point n times. ∎

If one ignores the constraints of locally convex box complexes, h, m, and M can be as large as $O(n^{\lfloor d/2 \rfloor})$ [Klee 1966]. If so, the worst case complexity is $O(n^{4\lfloor d/2 \rfloor + 1})$.

This compares poorly with Beneath-beyond at $O(n^{\lfloor d/2 \rfloor + 1})$. But empirically in 3-d, Quick_hull is $O(n)$ instead of $O(n^5)$ (Figure 7.7 in Chapter 7). Why? The number of ridges per facet is usually small (if not, the number of adjacent big facets is small–at least in 3-d); *partition points* divides an outside list into multiple outside lists; M is empirically a small constant; and the convexity constraints bound h.

LEMMA 5.25. *If m, M and d are bounded, long outside lists shorten geometrically per Quick_hull level, and the number of deleted vertices is a bounded proportion of all vertices, the total computational complexity of Quick_hull is $O(n \log h + h)$.*

PROOF. By Lemma 5.18, the result holds for *partition points*. As in Lemma 5.18, the maximum size of the box complex is $O(h)$ and the total cost after outside lists are short is $O(h)$. Similarly, the maximum number of levels with long outside lists is $O(\log h)$. No more than n points can be processed per level; giving the desired result. ∎

In 2-d, *partition points* almost always divides the outside list of one interior facet into two new facets. The exception occurs when the furthest point is not really the furthest point because of round-off error. If so, the furthest point may be deleted from the hull at a later iteration. If this seldom occurs and if partitioning decreases the length of a long outside list by at least some constant, then the time complexity of Quick_hull is optimum.

COROLLARY 5.26. *In 2-d, if long outside lists shorten geometrically per Quick_hull level and the number of deleted vertices is a bounded proportion of all vertices, the time complexity of Quick_hull is $O(n \log h)$.*

LEMMA 5.27. *If $\theta < \pi$ is the maximum ridge angle and Δ the radius of the point set, then the maximum number of facets in 2-d is $2\pi/(\pi - \theta)$ and in 3-d is $16/(\pi - \theta)^2$.*

PROOF. Consider a great circle of the Δ sphere around the point set. By the angle constraint, it can cross at most $2\pi/(\pi - \theta)$ regular facets. So the minimum diameter of a regular facet is $\Delta(\pi - \theta)$. If facet diameters differ, we can enlarge narrow facets and divide wide facets without increasing Δ. In 2-d, the diameter

is the facet length; in 3-d, the facet area is at least the same as a circle of that diameter, $\pi(\Delta(\pi-\theta)/2)^2$. Dividing into the surface area of a d-sphere ($2\pi\Delta$ [2-d]; $4\pi\Delta^2$ [3-d]) gives the result. ∎

LEMMA 5.28. *If Δ is the radius of the point set, and $\rho \approx 0$ is the centrum precision, the maximum number of facets in 2-d is $\pi\Delta/\sqrt{2\rho\Delta}$ and in 3-d $8\Delta/\rho$.*

PROOF. Use the same construction as before and let θ be the angle between adjacent facets. Let w be the width of a facet with a centrum in its center. Then $\sin(\pi-\theta) = 2\rho/w$. As above, $w = \Delta(\pi-\theta)$. Since $\rho \approx 0$, we can approximate the angle to get $w = \sqrt{2\rho\Delta}$ [3]. As above, taking the ratio with the surface area of a d-sphere gives the result. ∎

COROLLARY 5.29. *If maximum precision and d are fixed, long outside lists shorten geometrically per Quick_hull level, and the number of deleted vertices is a bounded proportion of all vertices, the total computational complexity of Quick_hull is $O(n)$.*

Clarkson et al [1992; Fortune 1992a], Guibas et al [1990], and Boissonnat and Devillers-Teillaud [1989] prove their probabilistic convex hull and Delaunay triangulation algorithms run in average time $O(n \log n + h')$ where h' is the maximum output size. Chazelle [1991] use derandomization to get the same results deterministically.

These algorithms are not output-sensitive. Output-sensitivity is important for convex hull algorithms because the output size can be much smaller than the worst case size. In 2-d, Kirkpatrick and Seidel [1989] found an optimal, output-sensitive algorithm for convex hull in $O(n \log h)$. The optimal output-sensitive result for 3-d is Chazelle and Matoušek, J. [1991]. Note that if the preconditions of Lemma 5.25 are meet, Quick_hull achieves the optimal output-sensitive result in all dimensions.

If points are in general position and Quick_hull selects outside points randomly, Clarkson et al.'s average-case analysis applies to Quick_hull. As argued in Chapter

[3] This implies that $\cos(\pi-\theta) = \sqrt{1-2\rho/\Delta} \approx 1-\rho/\Delta$. Since we specify an angle constraint via its cosine (see "if R is clearly convex†"), the two methods of constraining convexity are similar.

4, the two algorithms are fundamentally the same. Both algorithms add one point at a time, and both determine an interior facet by partitioning the point through prior, intermediate hulls. The difference is that Quick_hull uses the locally, furthest point above a facet and it handles non-convex ridges. Though Quick_hull's furthest points makes the selection order deterministic, the furthest point is more likely to be an extreme point than a random point. Empirically, $P_{furthest}$ is usually an extreme point.

13. Operations for Quick_hull

This section gives further details about operations used in the Quick_hull algorithm. When mentioned in the thesis, they are flagged by "‡".

Each subsection expands the pseudocode steps for the procedures of Quick_hull (see 5.1). Steps common to several procedures are listed alphabetically first. The title of a sub-subsection is the same as the name of the pseudo-code step. "F" refers to a facet, "P" refers to a point, "R" refers to a ridge, and "S" refers to a set.

13.1. Common pseudocode steps in Quick_hull.

centrum of a facet

For a ridge to be clearly convex, a point of each facet must be below the other facet's hyperplane. The point is called the facet's *centrum*. The centrum must be close to the center of the facet. Otherwise, a merge can increase the maximum facet width by an arbitrary amount.

DEFINITION 5.30. *Consider a facet in* $\mathbf{R}^d$ *whose vertices define a convex polytope in a hyperplane. A* centrum *for the facet is a point of the polytope such that for any half plane through the point, the absolute value of the ratio between maximum positive distance and minimum negative distance is at most* $d - 1$.

Consider a centrum whose precision is ρ and a neighboring hyperplane through an edge of the facet (Figure 5.7). Since the centrum bounds the ratio, the maximum

Figure 5.7: In 3-d, if a neighboring hyperplane intersects a centrum for a facet, the facet's vertices are at most 3ρ from the hyperplane.

distance of a vertex to the neighboring hyperplane is bounded. The error analysis in the next chapter depends on this bound.

Vertices of a δ-box may be non-coplanar and its ridges may form a non-convex polyhedron. To define a centrum for an imprecise facet, we project the vertices to the facet's hyperplane and take their convex hull.

Radon proved the following in 1916 [cf. Grünbaum 1961]:

THEOREM 5.31. *Any polytope in* $\mathbf{R}^{d-1}$ *has a centrum. The ratio* $1 : d-1$ *is tight for a simplex (its centrum is the arithmetic mean of its vertices).*

The following brute force algorithm will find a centrum. It is not used in practice. First take the convex hull of the projected vertices. For each diameter of the convex hull, construct perpendicular lines at $1/d$ and $(d-1)/d$. The strip of points inside the lines satisfy the centrum property for that diameter. The centrum is any point in the intersection of all the strips.

The cost of determining the centrum is dominated by the cost of finding the convex hull. This yields:

LEMMA 5.32. *Let* n *be the number of vertices in a facet in* $\mathbf{R}^d$ *and assume precise arithmetic. The complexity of finding the centrum is*

$$O(n \log n + n^{\lfloor (d-1)/2 \rfloor + 1})$$

Computing the convex hull of each facet is prohibitively expensive. In the current implementation of Quick_hull, a facet's centrum is the arithmetic mean of its vertices. We take the following liberty with our analysis: complexity analysis uses the arithmetic centrum but error analysis uses the computed centrum. Additional error introduced by using the arithmetic centrum may be accounted for by the maximum twist parameter of Quick_hull (see Chapter 6). A better algorithm for selecting the centrum is left as a open problem [cf. Grünbaum 1961].

create a hyperplane and inner and outer plane for a simplex

A new facet in $\mathbf{R}^d$ has d vertices. Its vertices define a simplex given as an apex (a vertex) and an oriented base (a ridge of $d-1$ vertices). A hyperplane is computed for the simplex (see "hyperplane through a simplex†"). The facet's centrum is its arithmetic center (see "centrum for a facet‡"). The inner plane is defined to be low enough to clearly include the vertices. Similarly the outer plane is defined to be clearly above the vertices (see "lower an inner plane‡" and "raise an outer plane‡").

The error analysis in Chapter 6 requires a minimum separation between the outer plane and hyperplane. In 2-d, it is $3\epsilon_\rho + 3\epsilon_\beta$, and in 3-d, it is $10\epsilon_\rho + 4\epsilon_\beta$. Empirically, these are excessive, with ϵ_ρ a better minimum separation.

The computational complexity for creating a hyperplane, centrum, inner and outer plane is $O(d^3)$ (see the respective subsections).

determine best merge (F into F_n) for R

In *fix cone* and *fix horizon*, Quick_hull performs a locally-optimal merge whenever a ridge is non-convex. Either the neighboring facets of the ridge are merged, or one of the facets is merged into another neighboring. In either case, the remaining facet's hyperplane is retained.

The operation "determine best merge‡" locates the neighbor whose hyperplane is closest to one of the non-convex facets. It could be the other facet or it could be a different facet. If it is the other facet, ridge R is deleted in the merge (see "merge facet F into facet F_n‡"). If it is a different facet, the merge will change one of the hyperplanes for ridge R. In *fix cone*, a horizon facet can only merge into the neighboring cone facet.

To determine the locally-optimal merge, We compute the new inner and outer plane for each possible merge(see "raise outer plane‡" and "lower inner plane‡"). The merge that minimizes the facet width (outer_plane-inner_plane) is the best merge. Other evaluations may be used (see "thickness of a set of facets‡").

Let m be the maximum number of vertices or ridges in a facet. Testing a possible merge compares each vertex to a hyperplane at a cost of $O(dm)$. Testing all merges for a ridge then costs $O(dm^2)$. This can be expensive for large facets. If

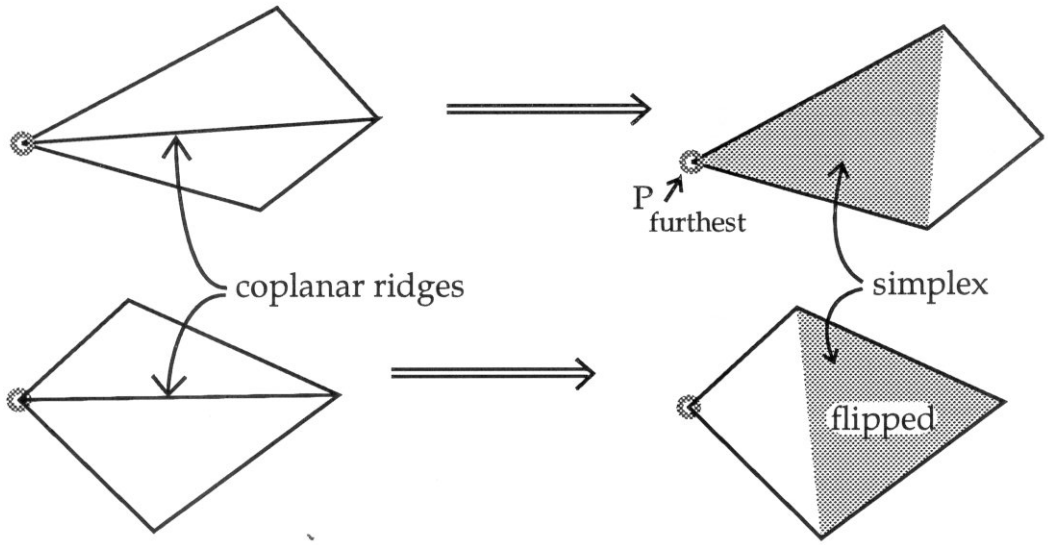


Figure 5.8: Merging two facets in *merge cone* can make $P_{furthest}$ a coplanar vertex. The choice depends on the simplex test.

a facet contains more than 10 ridges, only non-convex ridges are tested. This does not change the error analysis of Chapter 6.

if R fails the simplex test

In *merge cone*, Quick_hull reconstructs the hyperplane for a merged facet. In 3-d, two choices are considered for the facet's simplex (Figure 5.8). Either $P_{furthest}$ is the apex or the common, horizon vertex is the apex. The later case creates a flipped facet.

The simplex test compares the two possibilities. If $P_{furthest}$ is closer to the new base than the horizon vertex, the facet is flipped; otherwise $P_{furthest}$ is the new apex (see "build a flipped facet‡" under *merge cone* below). In four and higher dimension, there are several choices for the simplex of a merged facet. The best choice for Quick_hull has not been determined yet.

The simplex test takes $O(d^3)$ operations in $\mathbf{R}^d$.

The effect of the simplex test is to keep facets reasonably compact. Without it, a coplanar vertex could be far above the facet's hyperplane. The simplex test bounds the maximum expansion of facet width as follows:

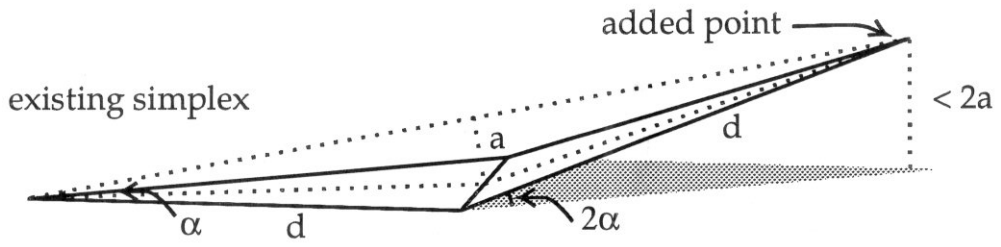


Figure 5.9: If a point passes the simplex test, it is no further than double the distance to the simplex's base

LEMMA 5.33. *Assume precise arithmetic in 3-d and consider a ridge R that passes the simplex test. If the ridge is a above the hyperplane at the simplex's base, the new coplanar vertex is at most $2a$ from the hyperplane.*

PROOF. Let d be the distance from apex to base. Consider a point on the ridge that is also d from the base (a closer point would be closer to the hyperplane) (Figure 5.9). Let α be the angle between the ridge and the plane, $\sin(\alpha) = a/d$. The height of the point is $d\sin(2\alpha)$ which is less than $2d\sin(\alpha)$. ∎

lower an inner plane to clearly include P

If P is at worst δ below the corresponding hyperplane (see "if P is clearly below a hyperplane†"), the inner plane is at least δ.

After this computation, P is *clearly above* the inner plane. Note that δ includes the additional roundoff error due to adding roundoff and precision.

It takes $O(d)$ operations in $\mathbf{R}^d$.

merge facet F into neighboring facet F_n

Merging two facets does the following:

- The new ridges are the union of the old ridges with common ridges deleted (the *mod 2 union*).
- F_n's hyperplane is retained.
- The centrum is recomputed from the vertices of both facets (see "centrum for a hyperplane‡").
- The inner and outer planes of F_n are adjusted to clearly include the vertices and coplanar points of F (see "raise an outer plane‡" and "lower an inner plane‡").
- The outside lists of both facets are merged, as are the coplanar lists.
- To determine the new furthest point, the furthest point of F's outside list is tested against F_n's hyperplane (see "if P is clearly above†"). If below the hyperplane, the outside list is added as coplanar points (via "raise outer planes of modified facets‡" in Chapter 5). If above, the further of the furthest points is selected.

LEMMA 5.34. *Box complexes are preserved under merging neighboring facets.*

PROOF. Merging neighboring facets is the same as coalescing two top level nodes of a DAG and taking the mod 2 union of their subordinates. In terms of the subgraphs that are used for the evenness condition, this is equivalent to taking the union of two subgraphs and deleting 0 or more pairs of paths. Since no edge becomes duplicated, the evenness condition is maintained. The box of the merged facet contains the boxes of its subordinates. ∎

The computational complexity is $O(d)$ per new vertex for adjusting the inner and outer planes and computing the new centrum.

raise an outer plane to clearly include P

If P is at worst δ above the corresponding hyperplane (see "if P is clearly above a hyperplane†"), the outer plane is at least δ. After this computation, P is *clearly below* the outer plane. Note that δ includes roundoff due to adding roundoff error and precision.

It takes $O(d)$ operations.

thickness of a set of facets

Before adding new facets to the hull, Quick_hull evaluates the facets and the interior facets that they replace. If the new facets are worse than the old ones, then the hull is not modified. The evaluation depends on the application. If S is a set of facets, We use $\max_{f \in S}(\mathit{outer\_plane}(f) - \mathit{inner\_plane}(f))$. An alternative is root-mean-square of facet widths minus r.m.s. of the old facet widths. This balances minimizing the maximum facet width with minimizing the average facet width. The evaluation should match "determine best neighboring facet‡".

twist of a merge

In three and higher dimensions, consider merging facet F into facet F_n across ridge R when F_n's hyperplane intersects F's centrum. If a F's ridges form a convex polytope and R lies on both hyperplanes, then merging introduces an error of at most d times the centrum precision see "centrum for a facet‡"). With thick facets, non-simplicial ridges and vertices are rarely coplanar with the hyperplane. We adopt a more realistic ideal case: R is parallel to both hyperplanes and R supports F's vertices. As shown in Chapter 6, the maximum error due to a merge in the ideal case is bounded.

The *twist* of a merge measures the deviation from the ideal case. Twist is measured relative to the line through the centrum that is parallel to the intersection of the hyperplanes. To simplify calculations, the line and all vertices are projected to the nearest axis plane. The minimum and maximum distance to the line is determined for the ridge vertices and for the other vertices of F. Then twist is the maximum ratio between the distance of a ridge vertex to the line and the distance

of another vertex of F to the line. The ratio depends on whether or not the ridge vertices (and other vertices) are on the same side of the line or are on opposite sides. Twist is not measured if the hyperplanes are nearly coplanar.

If the twist for a merge is greater than $d-1$ (the centrum ratio), the merge is worse than a merge under the ideal case. High twist occurs if the line through the centrum passes close to the ridge vertices. Then coplanarity with the centrum does not bound the deviation of a vertex from F_n's hyperplane. The worst case merge occurs if the hyperplanes are twisted ninety degrees relative to each other. Then the new facet width for F_n may be the diameter of F.

If twist is greater than the maximum twist parameter ζ, a warning is reported. The user may view the merge and reject the convex hull.

Intersecting two hyperplanes costs $O(d^3)$. If m is the maximum number of vertices in a facet, the cost of testing twist is $O(d^3 + m(d-1))$.

13.2. Pseudocode steps for top level.

if cone is better than the old, interior facets.

We use $\max(outer\_plane - inner\_plane)$ to evaluate a set of facets (see "thickness of a set of facets‡"). The facet widths for interior facets includes $P_{furthest}$. The cone must include at least d facets, otherwise it has failed and the old hull is retained.

attach the cone to the horizon.

Replace each horizon ridge of the cone with the corresponding horizon ridge of the hull. The interior facets are deleted and the new facets are added.

raise outer planes to clearly include P.

If Quick_hull retains the interior facets, it raises the outer planes of interior facets to include $P_{furthest}$ (see "raise an outer plane‡"). Outer planes can be adjusted during *find horizon* when interior facets are identified.

This operation takes $O(d)$ per facet.

partition outside lists of modified facets.

If an old outside point is now below the outer plane, it could still be outside a neighboring facet. Each point is repartitioned with *partition points.*

13.3. Pseudocode steps for *Initial hull* and *partition all.*

let S be a most distant pair of maximum points.

The goal of *initial hull* is to produce a large volume at minimum cost. Since the number of maximum points is at most $2d$, an $O(d^2)$ algorithm is suitable; each computation takes $O(d)$ operations, yielding a $O(d^3)$ cost. Ties are broken arbitrarily.

determine a maximum point P furthest from the flat through S.

The flat through S is computed at a cost of $O(d^3)$ (see "hyperplane through a simplex†"). Since $|S| \leq d+1$, computing the flat is the dominate cost. When $|S| = d$, testing the volume of a simplex is a good alternative (see "orientation of a simplex†"). Ties are broken arbitrarily.

determine any point P furthest from the flat through S.

As above, the flat costs $O(d^3)$ to create. The total cost of testing each point is $O(dn)$. Ties are broken arbitrarily.

create an initial hull from S.

The set contains $d+1$ points. *Initial hull* creates a simplex and defines a hyperplane and centrum for each facet (see "create a hyperplane‡" and "centrum for a facet‡"). It defines inner and outer planes to clearly include the vertices.

It orients the hyperplanes to S's arithmetic center. The orientation of a facet's hyperplane defines an orientation for each of its ridges. Once an initial orientation is set geometrically, the remaining orientations for Quick_hull are set topologically.

The complexity is $O(d^3)$ per facet or $O(d^4)$ altogether.

if the initial hull contains non-convex ridges.

Each ridge is tested for convexity (see "if R is clearly convex†").

partition the remaining points into outside and coplanar lists.

Partition all is the same as *partition points*. Its input is the list of new facets in the initial hull. See the description of *partition points* for details.

13.4. Pseudocode steps for *Find horizon*.

for each unprocessed ridge R in F.

Find horizon finds all adjacent facets below $P_{furthest}$. One way to organize this is a queue of unprocessed ridges. Whenever a facet F is appended to S_i, the queue is mod 2 unioned with the ridges of F.

13.5. Pseudocode steps for *Make cone*.

pair up new ridges.

Linear search or hash lookup are practical solutions. It is not counted in the total computational complexity since it does not use floating point operations.

13.6. Pseudocode steps for *Merge cone*.

while a cone ridge is not convex.

All cone ridges are tested for convexity and classified as either convex, concave or coplanar (see "if R is clearly concave†" and "if R is clearly convex†"). If the number of cone ridges falls below d, an empty cone is returned.

build a non-concave cone facet F.

Table 5.11 gives the individual steps and Figure 5.5 shows an example.

while a cone ridge R is clearly concave†
let F be one of R's facets
merge the other neighbor of R into F‡
create a hyperplane and inner and outer plane for F‡
determine the centrum for F‡

Table 5.11: Algorithm for "build a non-concave cone facet F".

Since the orientation of cone facets matches the orientation of horizon facets, all cone ridges can not be concave. So the loop terminates with a cone facet F that contains one or more coplanar vertices.

Except for the cost of getting the centrum, the cost per concave ridge is $O(d^3)$.

remove F's ridges from S.

Merge cone removes the merged, concave ridges and F's cone ridges from S. This prevents retesting the ridges until the end of the current pass. This takes $O(1)$ per ridge.

build chord facets for F by recursively calling *make cone* **and** *merge cone*.

The new $P_{furthest}$ for the recursive call to *make cone* is the closest vertex in the chord ridge to the current $P_{furthest}$. Its adjacent ridges in the chord facet initialize the triangulation; the other ridges are the vertex's horizon. New ridges are created as cone ridges and become chord ridges after the call.

When building chord facets, facets are not flipped. Such an event should be rare since it implies a double fold in the horizon. As a result, all chord facets for a cone facet share a vertex. The error analysis in Chapter 6 uses this property.

merge non-convex chord facets into F.

See Table 5.12. The chord facets from the recursive call to *merge cone* are mutually convex but may be coplanar or concave with the cone facet F.

This operation takes $O(d)$ for testing plus $O(dm)$ for merging per facet.

until F's chord facet F_c is convex‡ merge F_c into F†

Table 5.12: Algorithm for "merge non-convex chord facets into F".

until F's chord facet F_c is convex.

This is the outer loop of Table 5.12.

F and F_c share a vertex. So F_c is convex if its centrum is *clearly below* F's hyperplane (see "if P is clearly below†") and the angle between them is less than θ (see "angle for ridge†").

Note that F's centrum may be coplanar with F_c's hyperplane. If so, F will be merged in *fix cone*.

build a flipped facet F from R's neighbors.

The simplex for the flipped facet consists of the chord ridge between neighboring cone ridges and the remaining vertex in R. See "create a hyperplane‡" and "centrum for a facet‡".

build a cone facet F from R's neighbors.

The simplex for the cone facet consists of the neighboring cone ridges. See "create a hyperplane‡" and "centrum for a facet‡".

13.7. Pseudocode steps for *Fix cone*.

for all non-convex chord ridges, cone ridges, and horizon ridges R.

First the cone is made *clearly convex* and then the horizon is made *clearly convex* (see "if R is clearly convex†"). The error analysis in Chapter 6 assumes this order. *Fix cone* returns an empty cone if the number of cone ridges falls below d.

rename F as F_n.

In *fix cone*, a new facet can not be merged into an horizon facet because the interior facets may be retained. Instead the facet is renamed as if it were the horizon facet; *fix horizon* will merge them. The hyperplane equation is duplicated and the centrum recomputed (see "centrum for a facet‡").

mark F for merging into F_n.

Similarly, an horizon facet can not be merged into a new facet. The facet is marked for merging in *fix horizon* and its inner and outer planes are widened to include the horizon.

13.8. Pseudocode steps for *Fix horizon*.

for each facet F of the horizon and their untested neighbors.

At the beginning of *fix horizon*, all facets are locally convex except for horizon facets. These are explicitly tested. If any fail, their neighbors are retested.

13.9. Pseudocode steps for *Raise outers*.

for each unprocessed incident facet F_n of F_i.

This performs a depth-first search from F. Incident facets are those incident to a vertex of F_i.

Chapter 6

Error analysis of Quick_hull

This chapter analyzes the maximum width of a facet at the completion of Quick_hull. It gives separate bounds for 2-d and 3-d. The 2-d analysis is complete. The 3-d analysis holds if Quick_hull rejects inputs that cause a merge with twist greater than ζ (see "twist of a merge‡"[1]).

After *make cone*, ridges can be convex, coplanar, or concave. There are two kinds of coplanar ridges (see "if R is clearly convex†"[2]): *θ-coplanar* when the angle between neighboring hyperplanes is greater than θ, and *ρ-coplanar* when a centrum is coplanar with a neighboring hyperplane *A facet F_1 is ρ-coplanar with facet F_2* if F_1's centrum is coplanar with F_2's hyperplane. *θ-coplanar* when the angle between neighboring hyperplanes is greater than θ, and *ρ-coplanar* when a centrum is coplanar with a neighboring hyperplane *A facet is θ-coplanar with another facet* if their hyperplanes meet at an angle wider than θ.

Quick_hull merges facets to remove coplanar ridges and retriangulates facets to remove concave ridges. At the end of an iteration, Quick_hull has tested every ridge for convexity. So each iteration ends with all ridges clearly convex.

When created, a facet is just wide enough to include its vertices. The facet may grow after merges with adjacent facets. Initially (in *make cone*), hyperplanes

[1]The symbol "‡" indicates an operation defined in Chapter 5.

[2]The symbol "†" flags basic computations and operations that are defined at the end of Chapter 2.

intersect at ridges radiating from $P_{furthest}$. When such facets are merged, the hyperplane is redefined (see Figure 5.8 in Chapter 5). Redefinition stops at the end of *merge cone* or when the merged facet spans too great an angle (see "if R fails the simplex test‡").

At the end of *merge cone*, new facets are almost convex with each other. Intuitively, they form an umbrella over the old, interior facets. Unfortunately, hyperplanes may not intersect at ridges. This makes it difficult to redefine hyperplanes after merging facets. So merged facets in *fix cone* and *fix horizon* retain one of the old hyperplanes.

At the start of *partition points*, all new ridges are clearly convex, but horizon ridges need not be. Horizon ridges are checked by *fix horizon* and merged if necessary. *Raise outers* is the final step of Quick_hull. It turns a locally convex box complex into a convex box complex.

We first perform the analysis for *initial hull* and other steps with little effect on the final error. Then 2-d is done in detail, followed by 3-d.

1. *Initial hull*, *find horizon*, *make cone*, **and** *partition points*

Initial hull builds an initial hull from $d+1$ independent points. It defines the inner and outer planes to *clearly include* these points. *Partition all* assigns each point that is outside the initial hull to one of the facets. If a point straddles both hyperplane and outer plane, the outer plane is raised to include the point.

LEMMA 6.1. *If inner and outer planes are the minimum possible, the maximum width of a facet after initial hull and partition all is* $3\epsilon_\rho$.

PROOF. The maximum computed size of a centrum is ϵ_ρ. The computed size includes the precision ρ and roundoff errors in computing distances. A vertex on the hyperplane has the same maximum size. So the maximum width of the inner plane is ϵ_ρ. If a point is on a plane half between the the outer plane and hyperplane, it can straddle both planes. So the outer plane can be $2\epsilon_\rho$ from the hyperplane. ∎

Note that the error analysis below assumes a minimum outer plane that prevents a point from straddling the outer plane and hyperplane (see "create hyperplane‡").

Find horizon does not change facet widths. *Make cone* creates a cone of new facets from $P_{furthest}$ and the horizon ridges identified by *find horizon*. Each facet is a simplex of new vertices. *Partition points* partitions potential outside and coplanar points into the cone of new facets. The only geometric effect of *partition points* is to raise an outer plane if a point straddles a hyperplane and outer plane.

LEMMA 6.2. *Find horizon, make cone, and partition points, do not change maximum facet widths.*

2. 2-d error analysis

Much of Quick_hull exists for problems that arise in three and higher dimensions. When Quick_hull is restricted to 2-d, the algorithm is simpler. For example, it only tests one edge in *merge cone* and two horizon edges in *fix cone*. To assist the error analysis, we restate Quick_hull for 2-d in Table 6.1.

A facet may become wider whenever facets are merged. There are two reasons for merging facets:

- Two new edges may not form a *clearly convex* angle, or
- a new edge and horizon edge may not form a *clearly convex* angle.

In either case, the angle may be more than θ degrees (θ-coplanar) or a centrum may be coplanar with an edge (ρ-coplanar). A new edge and a horizon edge may also form a concave angle. The rest of this section addresses each of these cases.

The *partition line* for $P_{furthest}$ is a line through $P_{furthest}$ and parallel to the edge below $P_{furthest}$. Because of *partition points*, all other points assigned to this edge must be on or below this line. So any new points can not be further than the maximum roundoff error (ϵ_β) above the line.

LEMMA 6.3. *An outside or coplanar point is at most ϵ_β above a neighboring edge.*

PROOF. Each new vertex was the computed, furthest point above an edge. Other points are at most ϵ_β further above the same edge (the grey region in Figure 6.1).

create an initial hull	(*initial hull*)
partition points into outside and coplanar lists	(*partition all*)
while there exists a furthest point $P_{furthest}$ for any facet F	*repeat*
find the horizon for $P_{furthest}$ starting with F	(*find horizon*)
make two new edges from $P_{furthest}$ to the horizon	(*make cone*)
if the new edges form a *clearly convex* angle	*if convex*
if either new edge is coplanar with its horizon edge	*check horizon*
tentatively perform merge	
if the new edges are better than the old edges	*if better*
partition points into outside and coplanar lists	(*partition points*)
attach the new edges to the horizon	(*attach cone*‡
if either new edge is coplanar with its horizon edge	*if coplanar*
merge the new edge and the horizon edge	(*merge facets*‡)
if the new edges are non-convex or worse than the old edges	*if worse*
raise outer planes to clearly include $P_{furthest}$	(*raise outer*‡)
partition outside lists of modified facets	(*partition outside*)

Table 6.1: Quick_hull algorithm restricted to 2-d.

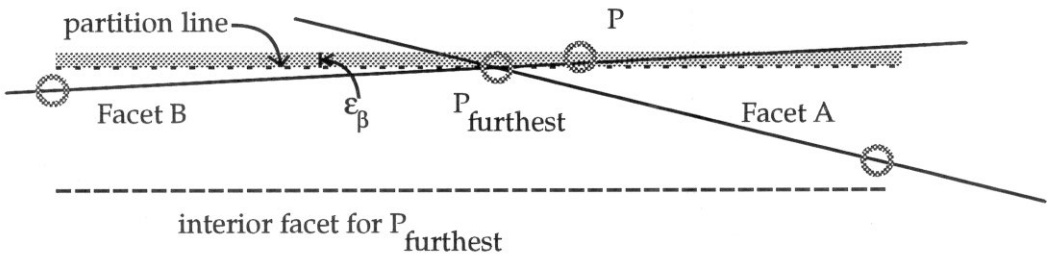


Figure 6.1: A new vertex is furthest above an edge. Outside points are at most roundoff error above a neighboring facet.

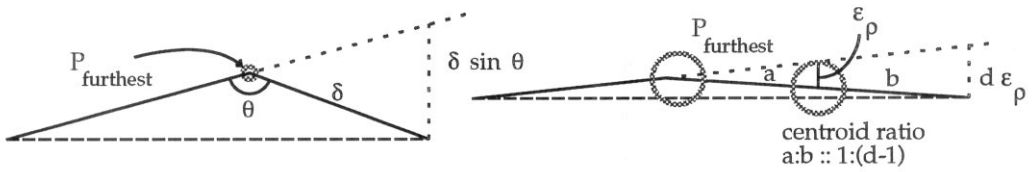


Figure 6.2: If new edges are θ- or ρ-coplanar, the deviation of $P_{furthest}$ from the old edge is bounded.

Since new edges are above the old interior edge and *partition points* assigns a point to the edge it is above, an outside point P can not be further than ϵ_β above a neighboring edge. ∎

LEMMA 6.4. *In 2-d, Quick_hull produces a convex box complex after each iteration. It does not need raise outers.*

PROOF. By Lemma 5.19, it produces a locally convex box complex. By Lemma 6.3, a point is at most ϵ_β above a neighboring edge. This is lower than the minimum outer plane, $3\epsilon_\rho + 3\epsilon_\beta$ (see "create a hyperplane‡"). So *fix horizon* in 2-d produces a convex box complex. ∎

LEMMA 6.5. *If two new edges are coplanar the maximum deviation of $P_{furthest}$ from the old edge is*

$$\max(\Delta \sin\theta, 2\epsilon_\rho).$$

PROOF. $P_{furthest}$ is on both new edges by construction. If the new edges are θ-coplanar, the maximum deviation of the new edges from an interior edge of length δ is $\delta \sin\theta$ (Figure 6.2). The parameter Δ is the diameter of the point set. If the new edges are ρ-coplanar, the maximum deviation of the edges at the centrum is ϵ_ρ by definition of ϵ_ρ. The maximum deviation of the new edge is double this because the centrum is the bisector. ∎

LEMMA 6.6. *If a new edge is ρ-coplanar with an horizon edge, the maximum deviation of $P_{furthest}$ from the horizon edge is $2\epsilon_\rho$.*

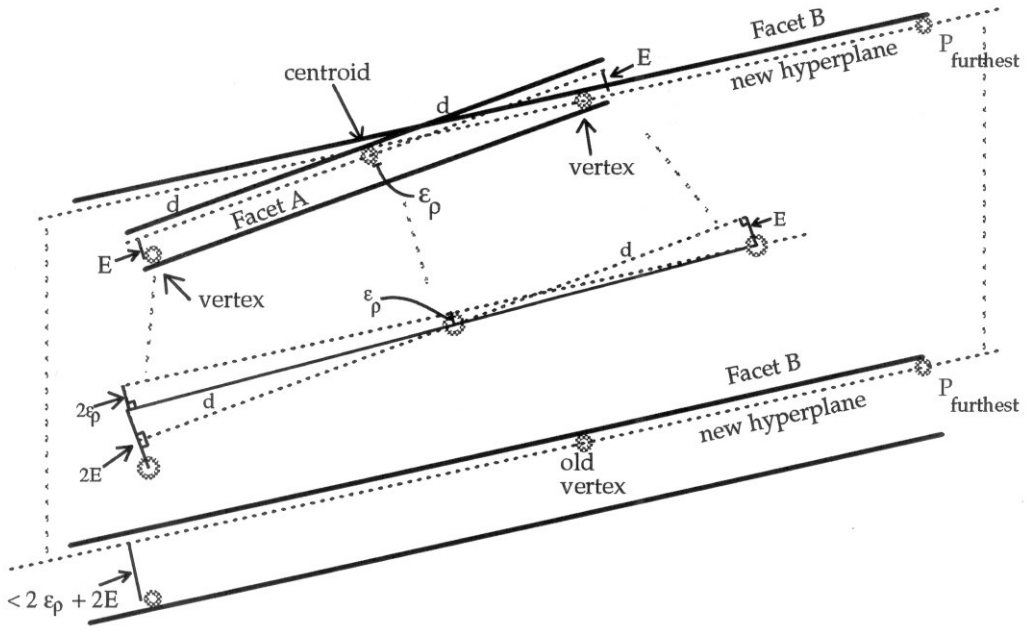


Figure 6.3: Merging an old facet into a new facet can increase the width to $2\mathcal{E}+2\epsilon_\rho$.

PROOF. By Lemma 6.3, $P_{furthest}$ is at most ϵ_β above a horizon edge. If below a horizon edge, the worst case occurs when the horizon vertex is on the horizon edge. The horizon edge intersects the centrum so $P_{furthest}$ can be twice as far away (same as Figure 6.2). ∎

The remaining possibility is a horizon edge that is ρ-coplanar with a new edge. The horizon vertex could be below the horizon edge.

LEMMA 6.7. *If the inner plane of an horizon edge is $\mathcal{E}$ and the horizon edge is ρ-coplanar with a new edge, the inner plane of the new edge is at most $2\mathcal{E} + 2\epsilon_\rho$ after merging.*

PROOF. A new edge exactly through the centrum is at most $\mathcal{E}$ from the horizon edge and at most $2\mathcal{E}$ from the inner plane. The new edge can be at most ϵ_ρ from the centrum's center point (Figure 6.3). ∎

The partition line prevents unbounded error growth due to repeated merges of

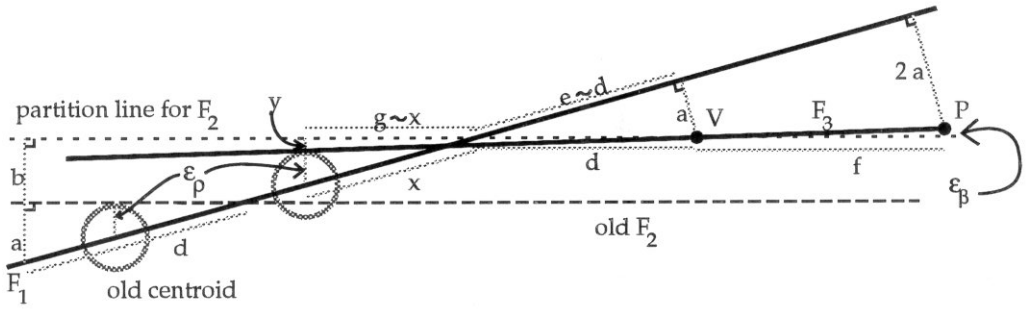


Figure 6.4: A partition line prevents rocking a facet more than once.

a horizon edge into a new edge. Consider (Figure 6.4). where Facet F_1 is an edge, V is a vertex a below F_1. Before being added to F_1, V was $P_{furthest}$ for facet F_2. The partition line for F_2 goes through V. A new furthest point P is ϵ_β above the partition line (the maximum possible) and $2a$ from F_1. Facet F_3 is a new edge from P to V. It just misses the centrum for F_1. The radius of the centrum is y and the top of the centrum is z below the partition line. The centrum is as high as it can be without violating the centrum ratio.

Facet F_2 just made being *clearly convex*, so a is $d\epsilon_\rho$. Note that $a << d$ so we can estimate $g \approx x$ and $e \approx d$.

LEMMA 6.8. *Consider horizon facets that are ρ-coplanar with a new facet. If the minimum offset to the outer plane is:*

$$d(\epsilon_\rho + \epsilon_\beta + a + \epsilon_\rho\epsilon_\beta/a) - 2a$$

a second merge due to a ρ-coplanar ridge can not double the offset to the inner plane.

PROOF. The first ratio is from the centrum property, the rest are by similar triangles:

$$\frac{h+x}{c+h+e} = \frac{1}{d}, \quad \frac{h+c}{a+b} = \frac{h}{a}, \quad \frac{h+f}{2a+\epsilon_\beta} = \frac{h}{a}$$
$$\frac{g+h}{f} = \frac{y}{\epsilon_\beta}, \quad \frac{h}{a} = \frac{x}{\epsilon_\rho + y}$$

Substitute x for g and d for e, and scale all variables so that $h = d$. Solving these gives:

$$\begin{aligned} c &= bd/a, \quad x = c/d + 2 - d = b/a + 2 - d \ \ (\text{since } ac = bd) \\ y &= \epsilon_\beta(d+x)/f = \epsilon_\beta(d+x)/f = \epsilon_\beta(b+2a)/(af) = \epsilon_\beta(b+2a)/(d(a+\epsilon_\beta)) \\ d\epsilon_\rho &= ax - dy = b + 2a - da - \epsilon_\beta(b+2a)/(a+\epsilon_\beta) = \frac{ab + 2a^2 - a^2 d - ad\epsilon_\beta}{a+\epsilon_\beta} \\ b &= d\epsilon_\rho(a+\epsilon_\beta)/a - 2a + ad + d\epsilon_\beta = d\left(\epsilon_\rho + \epsilon_\beta + a + \epsilon_\rho\epsilon_\beta/a\right) - 2a \end{aligned}$$

Increasing h lowers the centrum relative to F^3. So Figure 6.4 represents a worst case. ∎

COROLLARY 6.9. *Consider horizon facets that are ρ-coplanar with a new facet. In 2-d, a minimum outer plane of $3\epsilon_\rho + 3\epsilon_\beta$ limits the lower plane error to $-4\epsilon_\rho$.*

Combining the above lemmas, we get:

THEOREM 6.10. *In 2-d with Quick_hull, the maximum separation between outer plane and hyperplane is:*

$$\max(\Delta \sin\theta, 2\epsilon_\rho) + \epsilon_\rho + 3\epsilon_\beta$$

and the maximum separation between inner plane and hyperplane is:

$$2\max(\Delta \sin\theta, 2\epsilon_\rho) + \epsilon_\rho.$$

THEOREM 6.11. *In 2-d with exact inputs and no angle constraint, Quick_hull produces a convex $(8\epsilon_\rho + 3\epsilon_\beta)$-wide box complex. In terms of ρ and β, Quick_hull produces a a $(8\rho + 100\beta)$-convex box complex. The polytope of its hyperplanes is convex despite ρ perturbations of any extreme point.*

PROOF. The previous theorem bounds the maximum width of a facet. Lemma 5.2 gives upper bounds for ϵ_ρ and ϵ_β. The minimum size of a centrum is ρ which enforces the convexity. ∎

There are three results to compare this with: Fortune's [1989] inverse error analysis, Li and Milenkovic's [1990] ϵ-strongly convex δ-hull, and Guibas et al's [1990] similar result using global methods.

Using a variation of Graham scan, Fortune computes a 2-d convex hull that is near the precise convex hull [Fortune 1989]. His analysis bounds the perturbed input to $4\beta\Delta$.

The other papers produce an ϵ-strongly convex δ-hull. This is a convex hull in which any vertex can be perturbed by ϵ and no point is further than δ outside the hull. Our convex δ-wide box complex differs in that our vertices define the topological relationships but not the geometric relationships. Our geometry is given as the intersection of hyperplanes. Convexity is preserved despite a ρ perturbation of an hyperplane intersection.

Li & Milenkovic [1990] produce an ϵ-strongly convex $(12\epsilon + 288\sqrt{2}\beta)$-hull with a two pass algorithm. The first pass, forward trimming, constructs a "spine of vertebrae". Adjacent vertebrae are *clearly convex*. Each vertebrae may be followed by an "extender" edge. The second pass, backward trimming, removes reflex angles from the extenders. The algorithm runs in $O(n \log n)$. They prove that an hull is ϵ-strongly convex if each vertex is at least 2ϵ from the chord between adjacent vertices.

Guibas, Salesin, and Stolfi [1990] use global methods to achieve tighter bounds. They prove that every set of points has a ϵ-strongly convex $((1+\sqrt{2})\epsilon)$-hull. They reduce the convex hull problem to the convex hull of a dozen points, and use the InStronglyConvex predicate from their [1989] paper on ϵ-geometry. This gives them a ϵ-strongly convex δ-hull where $\delta = (5+\sqrt{2})(\epsilon + \epsilon_\beta) + \epsilon_\beta$. Their algorithm runs in $O(n^3 \log n)$ time.

In 2-d, Quick_hull gives comparable error bounds. Their ϵ is our $\rho/2$ while their δ is our outer plane (about half the width of a facet). Their algorithms have not been extended to 3-d. Note that their algorithms are not output-sensitive.

3. 3-d error analysis

The following analysis considers bounds for the maximum facet width of Quick_hull in 3-d. The analysis is straightforward but tedious. As Quick_hull progresses, the constraints on neighboring facets decreases and the maximum width of facets increases. One goal of Quick_hull is to prevent loops in this process, i.e., prevent a facet from forever growing wider. This is achieved by processing the locally furthest point at each iteration, by redefining the hyperplanes of new facets in *merge cone*, and by limiting the maximum twist of a merge in *fix cone* and *fix horizon*.

Even though the error analysis of individual merges reflects the actual performance of Quick_hull, empirically the error analysis overstates the maximum facet width. The next chapter gives examples of Quick_hull's performance on a variety of inputs.

A limitation of the analysis is the maximum twist ζ of a ρ-coplanar merge (see "twist of a merge‡"). Quick_hull rejects an input if it causes a twist greater than ζ. Empirically, the maximum twist for an input is about three. Note that ζ includes the centrum ratio which is also three. To see what ζ should be small, consider the extreme case of perpendicular hyperplanes and let it be the last ridge to merge. The only way to get a ridge this badly twisted would be to have a badly twisted box complex; but then the ridges wouldn't be clearly convex. We conjecture that the geometric constraints of a locally convex box complex prevents bad twists.

Note that ζ also bounds two other sources of error expansion: missing simplex tests and improper centrums. A missing simplex test occurs when a simplex test succeeds for one vertex but would have failed for an earlier coplanar vertex (see "if R passes the simplex test‡"). An improper centrum occurs when the arithmetic center of a facet does not satisfy the centrum ratio (see "centrum for a facet‡"). This may happen with large asymmetric facets. Empirically, these sources have minor effect. Indeed a ρ-coplanar merge with a high twist looks bad when viewed graphically and has uncovered conceptual errors in Quick_hull.

A related issue is V, the maximum number of embedded, concave ridges. A concave ridge in *merge cone* causes a retriangulation of two or more new facets. The retriangulation could in turn cause additional concave ridges. An upper bound

$\mathcal{E}_\theta$	max deviation due to θ-coplanarity
$\mathcal{E}_\rho$	max deviation due to ρ-coplanarity
$\mathcal{E}_{\text{outer}}$	max deviation of vertex above hyperplane
$\mathcal{E}_{\text{inner}}$	max deviation of vertex below hyperplane
$\mathcal{E}_{\text{coplanar}}$	max deviation due to degenerate cone

Table 6.2: Error parameters for Quick_hull.

$\mathcal{E}_\rho$	successive ρ-coplanar merges
$\mathcal{E}_\theta$	successive θ-coplanar merges
$\mathcal{E}_{\text{chord\_cone}}$	merging a chord facet into a cone facet
$\mathcal{E}_{\text{chord}}$	coplanar vertex in a chord facet
$\mathcal{E}_{\text{cone}}$	coplanar vertex in a cone facet
$\mathcal{E}_{\text{flipped}}$	coplanar vertex in a flipped facet
$\mathcal{E}_{\text{flip\_furthest}}$	$P_{furthest}$ in a flipped facet

Table 6.3: Maximum deviations derived for *merge cone*.

for V is n, but empirically, $V = 1$ and on rare occasion 2 or 3. We conjecture that the geometric constraints of a locally convex box complex prevents large V's.

The error analysis determines bounds for five error parameters (see Table 6.2). It starts with *merge cone*, and proceeds to *fix cone* and *fix horizon*. Degenerate cones define $\mathcal{E}_{\text{coplanar}}$. When possible, the error analysis is written for $\mathbf{R}^d$.

4. Error analysis of *merge cone*

Merge cone turns the simplicial facets from *make cone* into cone, chord, and flipped facets. Cone facets are adjacent to $P_{furthest}$, chord facets are made of horizon vertices, and flipped facets use $P_{furthest}$ as a coplanar vertex. *Merge cone* may widen a facet whenever it merges two facets. There are several causes of merged facets: θ-coplanarity, ρ-coplanarity, chord facets, and flipped facets. Each is discussed in turn (see Table 6.3).

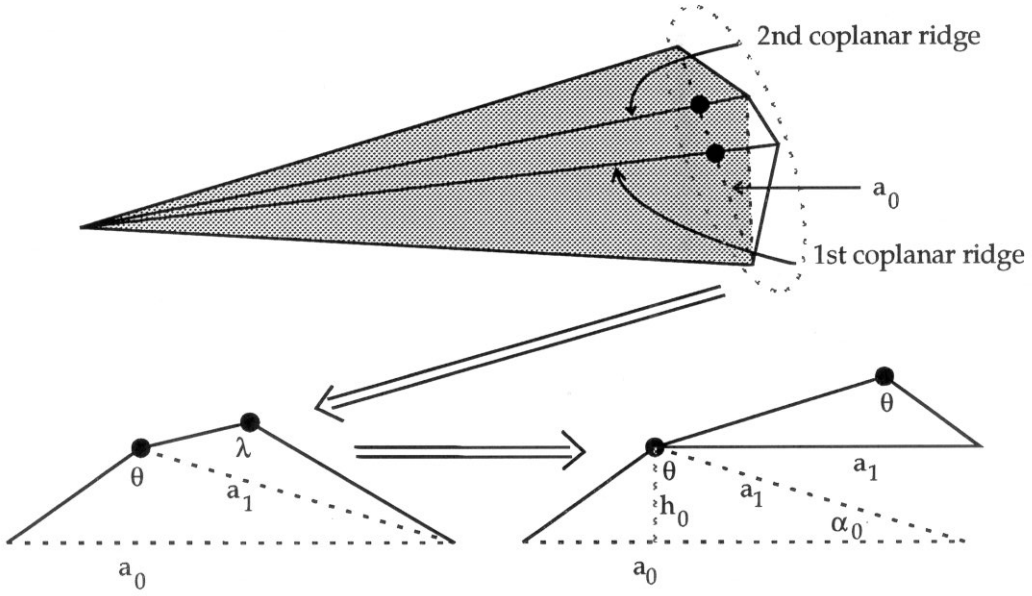


Figure 6.5: Successive merges due to θ-coplanarity.

4.1. Cone facets and θ-coplanarity. *Merge cone* merges θ-coplanar facets that intersect at a cone ridge. The facets passed the simplex test (see "if R fails the simplex test‡"). After merging the facets, *merge cone* defines a new hyperplane whose apex is $P_{furthest}$. At worst, a single merge can leave a vertex $\Delta \sin\theta$ above or below the new hyperplane (see Figure 6.7 below).

4.1. Cone facets and θ-coplanarity. θ-coplanarity can occur repeatedly during one instance of *merge cone*. Each time it occurs, the width of the facet increases. If the angle α between the new and old hyperplanes is large, the deviation of the corresponding vertex can be large; but then, the size a_1 of the old facet must be small. This inverse relationship prevents large deviations from the final hyperplane.

In 3-d, consider two successive θ-coplanar facets in Figure 6.5. The final plane is at the bottom. We want to bound the height of the topmost vertex. Construct a stack of triangles as follows:

1. The base of the largest triangle is ridge a_0. Its apex is the nearest point on the second coplanar ridge.

2. The base of the smaller triangle is the line a_1 from the previous apex to the ridge vertex. Its apex is the nearest point on the first coplanar ridge.

3. Flatten the triangles into a planar figure.

4. Flip every other triangle to give them the same orientation.

5. The original angles between planes are no less than θ. The apex angles (e.g., λ) are no less than the original angles. Decreasing these angles to θ increases the height of each apex (The last picture in Figure 6.5).

The height of the final stack H_{total} is at least twice the height of the final apex above the hyperplane.

If all points pass the simplex test, Lemma 5.33 limits the maximum height of a coplanar point to $2H_{total}/2$. Quick_hull does not guarantee this condition because flipping a simplex to an arbitrary vertex can cause a large deviation of $P_{furthest}$ from the hyperplane. Instead, Quick_hull tests the vertex of the current, coplanar ridge (see "if R passes the simplex test‡"). The twist ζ bounds the additional error due to untested vertices that would fail the simplex test.

The construction is easily extended to an arbitrary number of merges. The construction is an idealization in the following ways:

- Since *merge cone* performs all possible merges before retesting merged facets, the stack of triangles in Figure 6.5 should really be an inverted tree. We consider only one path through this tree.

- The stack of triangles assumes precise points and planes. While the planes were constructed from the points, roundoff error can perturb the planes. We ignore roundoff error because it most strongly affects flat triangles, it is small relative to edge lengths, and a tall stack of triangles is unlikely.

Consider an infinite stack of triangles (Figure 6.5). Let h_i be the height of the i'th triangle, a_i the length of its upper edge, and α_i the adjacent angle. The upper angle for all triangles is θ. Note that $2\pi/3 \leq \theta < \pi$.

The edge a_i for triangle i is the base of triangle $i+1$, so

$$\begin{aligned} a_i &= a_{i-1}\left(\cos\alpha_i + \frac{\sin\alpha_i}{\tan\theta}\right) \\ h_i &= a_i \sin\alpha_i \end{aligned}$$

Given $a_0 = 1$, the total height for an unbounded number of triangles is:

$$H_{\text{total}} = \sum_{i\geq 1} \sin\alpha_i \prod_{1\leq j\leq i} \left(\cos\alpha_j - \frac{\sin\alpha_j}{|\tan\theta|}\right).$$

To get an upper bound for H_{total}, first remove the trigonometric functions using $\cos\alpha \leq 1$, $\sin\alpha \leq \alpha$, and $\tan\alpha \geq \alpha$ $(\alpha < \pi/2)$. Then consider successively longer prefixes of the sum [Cai 1992]. Let $T = |\tan\theta|$. Note that $T > 0$.

$$H_{\text{total}} = \sum_{i\geq 1} \sin\alpha_i \prod_{1\leq j\leq i} \cos\alpha_j \left(1 - \frac{\tan\alpha_j}{T}\right) \leq \sum_{i\geq 1} \alpha_i \prod_{1\leq j\leq i} \left(1 - \frac{\alpha_j}{T}\right)$$

Call this sum $H(\alpha_1, \alpha_2, ...)$ and let H_n be the first n terms:

$$H_n(\alpha_1, ..., \alpha_n) = \sum_{1\leq i\leq n} \alpha_i \prod_{1\leq j\leq i} \left(1 - \frac{\alpha_j}{T}\right).$$

Since all terms are non-negative, $H_n \leq H_{\text{total}}$. The following recurrence holds:

$$H_n(\alpha_1, ..., \alpha_n) = \alpha_1\left(1 - \frac{\alpha_1}{T}\right) + \left(1 - \frac{\alpha_1}{T}\right) H_{n-1}(\alpha_2, ..., \alpha_n).$$

To remove the dependency on a particular choice of angles, let $\widehat{H}_n$ be the supremum over all possible arguments.

$$H_{n-1}(\alpha_2, ..., \alpha_n) \leq \widehat{H}_{n-1}(\alpha_2, ..., \alpha_n) = \widehat{H}_{n-1}(\alpha_1, ..., \alpha_{n-1}).$$

An induction argument proves that $\widehat{H}_n \leq |\tan\theta|$ for all n and therefore $H_{\text{total}} \leq \lim_{n\to\infty} \widehat{H}_n \leq |\tan\theta|$. The base case is:

$$\begin{aligned} H_1 &= \alpha_1\left(1 - \frac{\alpha_1}{T}\right) = \frac{T}{4} - \left(\frac{\sqrt{T}}{2} - \frac{\alpha_1}{\sqrt{T}}\right)^2 \\ \widehat{H}_1 &< T \end{aligned}$$

For the induction step, assume that $\widehat{H}_{n-1} \leq T$.

$$\begin{aligned} H_n &\leq \left(1 - \frac{\alpha_1}{T}\right)(\alpha_1 + T) = T - \frac{\alpha_1^2}{T} \\ \widehat{H}_n &\leq T \end{aligned}$$

This completes the derivation of $H_{total} \leq |\tan\theta|$.

LEMMA 6.12. *Under θ-coplanarity in merge cone with precise intersection of adjacent hyperplanes in 3-d, the maximum deviation of a vertex from a cone facet's hyperplane is:*

$$\mathcal{E}_\theta = \zeta \Delta \sin\theta.$$

PROOF. By the above derivation, the maximum height of a stack of triangles is $|\tan\theta|$. By construction, the maximum height of a vertex above a hyperplane is $\zeta|\tan\theta|/2$. In the range $2\pi/3 \leq \theta \leq \pi$, $\sin\theta \geq |\tan\theta|/2$. ∎

4.2. Cone facets and ρ-coplanarity. ρ-coplanarity occurs when a centrum is coplanar with an adjacent hyperplane. As with θ-coplanarity, *merge cone* merges ρ-coplanar facets if they pass the simplex test. After merging, *merge cone* defines a new hyperplane using $P_{furthest}$ as the apex. Merging can occur repeatedly. We limit the maximum deviation due to ρ-coplanarity by performing all merges before retesting merged facets.

When facets are ρ-coplanar, their hyperplanes are no more than ϵ_ρ apart at the centrum. The hyperplanes intersect at one ridge, so the other vertices are at most $d\epsilon_\rho$ from the plane (see "centrum for a facet‡").

When successive facets are merged, consider the base ridge that is closest to $P_{furthest}$. Construct a perpendicular hyperplane through the base ridge (Figure 6.6). Using $P_{furthest}$ as a focal point, map the other facets so that their base ridges are on this plane. The mapped centrums will still intersect the mapped hyperplanes, and the mapped vertices are at most $d\epsilon_\rho$ from the other hyperplane. Because all merges passed the simplex test, the actual vertices are at most $\zeta d\epsilon_\rho$ from the other hyperplane (see "if R fails the simplex test‡"). As with θ-coplanarity, ζ bounds the effect of untested vertices that would fail the simplex test.

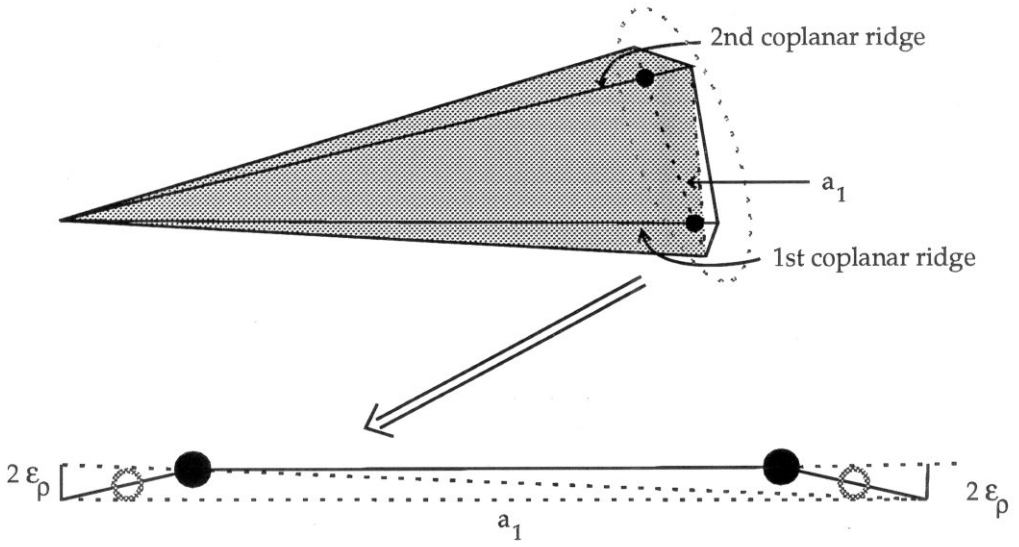


Figure 6.6: Successive merges due to ρ-coplanarity.

Each inner loop of *make cone* processes at least $1/(d-1)$ of the adjacent coplanar ridges[3]. If merges do not create new ρ-coplanar ridges, the maximum number of successive merges is $log_{d-1}(mM)$.

Merges can create new ρ-coplanar ridges, but the direction of deviation is usually reversed. The deviation may increase when variations in roundoff error cause a new ρ-coplanar ridge. Since these variations are bounded by the maximum roundoff error, we consider new ρ-coplanar ridges to be a second-order effect.

As with θ-coplanarity, roundoff error in computing the hyperplanes has a second-order effect (its strongest effect is on flat triangles, and it is small relative to edge lengths). Discounting second-order effects gives us:

LEMMA 6.13. *In 3-d under ρ-coplanarity in merge cone, the maximum deviation of a vertex from a cone facet's hyperplane is:*

$$\mathcal{E}_\rho = \zeta d \epsilon_\rho \log_{d-1}(mM).$$

[3]Each facet contains at most $d-1$ cone ridges, all paired with another facet. Each merge deletes one coplanar cone ridge and removes $2(d-1)$ ridges from the current pass. Of the removed ridges, the deleted ridge and other, adjacent coplanar ridges are double counted.

4.3. Coplanar chord facets. When one or more adjacent cone ridges are *clearly concave*, *merge cone* retriangulates the neighboring cone facets using recursive calls to *make cone* and *merge cone*. It creates one cone facet, and one or more chord facets with convex chord ridges. Chord facets share the same apex. On rare occasions, a newly created ridge may again be concave. If so, the process is repeated. The parameter V is the maximum number of embedded, concave ridges.

The chord ridge between the new cone facet and the first chord facet may be non-convex. *Merge cone* merges chord facets into the original cone facet until the chord ridge is convex. Details are in "merge non-convex chord facets[‡]".

LEMMA 6.14. *The maximum deviation of a vertex above a cone facet's hyperplane is unchanged by merging coplanar chord facets into the new cone facet.*

PROOF. Prior to the creation of a chord facet, its vertices were coplanar with one of the concave cone facets. The new cone facet is above these facets, preserving the prior bound. ∎

LEMMA 6.15. *Let $\mathcal{E}_{\text{chord}}$ be the maximum deviation of a vertex from a chord facet. If the centrum of a chord facet is maybe above the cone facet, the maximum deviation of the chord facet's vertices below the cone facet's hyperplane is: $d\epsilon_\rho + \mathcal{E}_{\text{chord}}$.*

PROOF. A chord facet shares a vertex with the cone facet[4]. By the centrum ratio, $d\epsilon_\rho$ is the maximum deviation of the chord facet's hyperplane from the cone facet. The second term is the maximum deviation of a chord vertex. ∎

LEMMA 6.16. *Let $\mathcal{E}_{\text{chord}}$ be the maximum deviation of a vertex from a chord facet. If a chord ridge is θ-coplanar, the maximum deviation of the chord facet's vertices from the cone facet's hyperplane is: $\Delta \sin\theta + \mathcal{E}_{\text{chord}}$.*

PROOF. The maximum separation of the hyperplanes is $\Delta \sin\theta$, adding in $\mathcal{E}_{\text{chord}}$ gives the result. ∎

Combining the above lemmas proves:

[4]This is the reason for disallowing flipped facets during recursive calls to *make cone*.

LEMMA 6.17. *Let $\mathcal{E}_{\text{chord}}$ be the maximum deviation of a vertex from a chord facet. The maximum deviation of a vertex due to merging chord facets into a cone facet is:*

$$\mathcal{E}_{\text{chord\_cone}} = \mathcal{E}_{\text{chord}} + \max(\Delta \sin\theta, d\epsilon_\rho).$$

4.4. Cone and chord facets. The three sources of coplanarity can occur in any order: coplanar chord facets, ρ-coplanarity and θ-coplanarity. Fortunately, each contribution to the total error can be separated out.

LEMMA 6.18. *Assuming precise intersection of adjacent hyperplanes, the maximum deviation of a cone facet's vertex is:*

$$\mathcal{E}_{\text{cone}} = \mathcal{E}_{\text{chord\_cone}} + \mathcal{E}_\rho + \mathcal{E}_\theta.$$

PROOF. A chord facet only occurs when a cone ridge was concave. Since the recursive calls to *make cone* builds a new cone facet, we can ignore prior coplanar chord facets. The deviation due to ρ-coplanarity depends on the height of the processing tree for coplanar ridges. These ridges can be removed from the tree and added up separately. The deviation due to θ-coplanarity depends on the lengths of the base edges. Removing ρ-coplanar ridges can only decrease the length of the remaining base edges. So the total height of a tree of mixed ridges is bounded by one coplanar chord facet plus a tree of all ρ-coplanar ridges plus a tree of all θ-coplanar ridges. ∎

Because of the recursive call to *make cone*, a chord facet is the same as a cone facet. At the lowest level,

$$\mathcal{E}_{\text{chord}}{}^{(i)} = \mathcal{E}_{\text{cone}}{}^{(i)} \Longrightarrow \mathcal{E}_{\text{chord}}{}^{(1)} = \mathcal{E}_{\text{cone}}{}^{(1)} = \mathcal{E}_\rho + \mathcal{E}_\theta.$$

At level i,

$$\begin{aligned}
\mathcal{E}_{\text{chord_cone}}{}^{(i)} &= \mathcal{E}_{\text{chord}}{}^{(i-1)} + \max(\Delta \sin\theta, d\epsilon_\rho) \\
\mathcal{E}_{\text{cone}}{}^{(i)} &= \mathcal{E}_{\text{chord_cone}}{}^{(i)} + \mathcal{E}_\rho + \mathcal{E}_\theta \\
\mathcal{E}_{\text{chord}}{}^{(i)} &= \mathcal{E}_{\text{chord}}{}^{(i-1)} + \max(\Delta \sin\theta, d\epsilon_\rho) + \mathcal{E}_\rho + \mathcal{E}_\theta
\end{aligned}$$

Telescoping to level V yields:

$$\mathcal{E}_{\text{chord}}{}^{(V)} = (V-1)\max(\Delta \sin\theta, d\epsilon_\rho) + V(\mathcal{E}_\rho + \mathcal{E}_\theta).$$

The $V(\mathcal{E}_\rho + \mathcal{E}_\theta)$ term is misleading since ρ-coplanar and θ-coplanar ridges use different vertices. Instead, $\mathcal{E}_\rho$ bounds the total contribution of ρ-coplanarity. For each θ-coplanar ridge, induce an equivalent θ-coplanar ridge for $P_{furthest}$. As argued in the proof of Lemma 6.18, these are bounded by a maximal tree of θ-coplanar ridges. So $\mathcal{E}_\rho + \mathcal{E}_\theta$ bounds the total contribution of ρ- and θ-coplanarity. Combining previous lemmas gives:

LEMMA 6.19. *Assuming precise intersection of adjacent hyperplanes, the maximum deviation of a vertex in a chord facet is:*

$$\mathcal{E}_{\text{chord}} = (V-1)\max(\Delta \sin\theta, d\epsilon_\rho) + \mathcal{E}_\rho + \mathcal{E}_\theta.$$

COROLLARY 6.20. *Assuming precise intersection of adjacent hyperplanes, the maximum deviation of a vertex in a cone facet is:*

$$\mathcal{E}_{\text{cone}} = V\max(\Delta \sin\theta, d\epsilon_\rho) + \mathcal{E}_\rho + \mathcal{E}_\theta.$$

4.5. Flipped facets. A flipped facet occurs when a cone ridge fails the simplex test (see "if R fails the simplex test‡"). When this happens, a vertex is further from the simplex's base than $P_{furthest}$. This situation is repaired by flipping the facet's simplex and making $P_{furthest}$ a coplanar point.

LEMMA 6.21. *Assuming precise intersection of adjacent hyperplanes, the maximum deviation of a vertex in a flipped facet is:*

$$\mathcal{E}_{\text{flipped}} = \mathcal{E}_{\text{cone}} + \max(\Delta \sin\theta, d\epsilon_\rho).$$

The maximum deviation of $P_{furthest}$ from a flipped facet is:

$$\mathcal{E}_{\text{flip\_furthest}} = \max(\Delta \sin\theta, d\epsilon_\rho).$$

PROOF. Either θ- or ρ-coplanarity can flip a cone facet. Consider a worst-case point P for θ-coplanarity (Figure 6.7). The angle between the facets is θ, so the maximum deviation of $P_{furthest}$ above the flipped simplex is $\delta \sin\theta$. Adding in $\mathcal{E}_{\text{cone}}$ bounds the maximum error. A similar derivation bounds the deviation for ρ-coplanarity. ∎

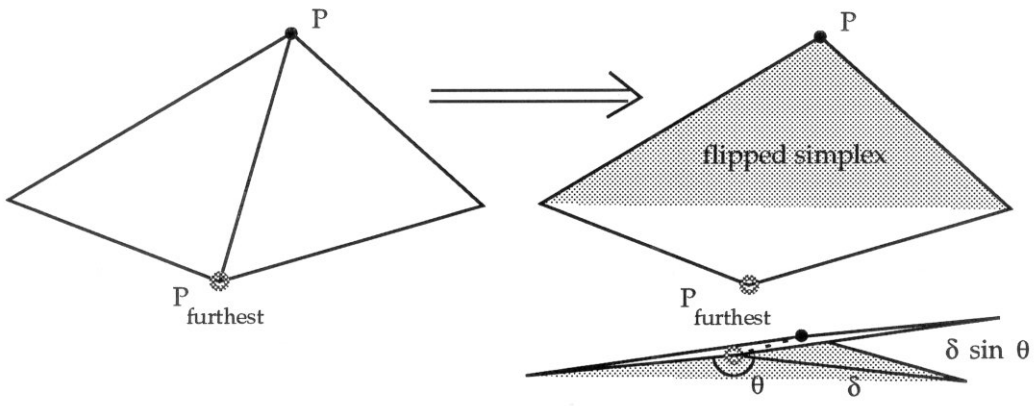


Figure 6.7: A flipped facet can leave $P_{furthest}$, $\delta \sin\theta$ above the hyperplane.

4.6. Summary of error analysis for *merge cone.* *Merge cone* produces cone, chord, and flipped facets. The neighbor of a chord ridge may be a cone or flipped facet. We defined the following error parameters:

$$\begin{aligned}
\mathcal{E}_\theta &= \zeta \Delta \sin\theta \\
\mathcal{E}_\rho &= \zeta d\epsilon_\rho \log_{d-1}(mM) \\
\mathcal{E}_{\text{cone}} &= \mathcal{E}_\rho + \mathcal{E}_\theta + V \max(\Delta \sin\theta, d\epsilon_\rho) \\
\mathcal{E}_{\text{flipped}} &= \mathcal{E}_\rho + \mathcal{E}_\theta + (V+1) \max(\Delta \sin\theta, d\epsilon_\rho) \\
\mathcal{E}_{\text{flip_furthest}} &= \max(\Delta \sin\theta, d\epsilon_\rho)
\end{aligned}$$

They are used below.

5.0. Summary of error analysis for *merge cone.*

5. Error analysis of *fix cone*

At the end of *fix cone*, all new ridges are *clearly convex*, and horizon ridges are either *clearly convex* or marked for merging. This allows "if cone is better than interior‡" to evaluate the cone before attaching it to the horizon. If the cone is an improvement,

When *fix cone* or *fix horizon* locate a non-convex ridge, they determine the best neighboring facet for a merge (see "determine best merge‡"). There are four cases:

$\mathcal{E}_{\text{rho\_flipped}}$	when a new facet is merged into a flipped or cone facet
$\mathcal{E}_{\text{rho\_chord}}$	when a new facet is merged into a chord facet
$\mathcal{E}_{\text{two\_flipped}}$	when two flipped facets are merged
$\mathcal{E}_{\text{outer}}$	vertex above any facet

Table 6.4: Maximum deviations derived for *fix cone*.

concave new ridge, ρ-coplanar ridge, θ-coplanar ridge, or degenerate cone (see Table 6.4).

LEMMA 6.22. *In fix cone, concave new ridges do not increase the maximum facet width.*

PROOF. *Merge cone* retriangulates concave cone ridges so cone facets can not be concave. Since a chord facet after *merge cone* is clearly above $P_{furthest}$, chord can not be concave with cone or flipped facets. If a flipped and cone facet is concave, $P_{furthest}$ is below a plane through existing vertices. If two flipped facets are concave, either $P_{furthest}$ is below a plane through existing vertices, or both flipped facets are narrow. In the latter case, a simplex of their vertices lies below $P_{furthest}$ and $P_{furthest}$ is at most $\mathcal{E}_{\text{flip\_furthest}}$ from the simplex. In all cases, $P_{furthest}$ can be added to an interior facet without increasing the maximum outer hull. ∎

If facet A is ρ-coplanar with facet B then Quick_hull merges facet A into facet B. The order of merging in *fix cone* is: cone or chord facet ρ-coplanar with flipped facet, chord facet ρ-coplanar with cone facet, cone or flipped facet ρ-coplanar with chord facet, and flipped facet ρ-coplanar with flipped facet. The key lemma is a 3-d version of Lemma 6.7. The configuration below is the same as Figure 6.3 with ζ instead of 2.

LEMMA 6.23. *If the maximum deviation of facet A is $\mathcal{E}$ and facet A is ρ-coplanar with another facet B, the maximum deviation of A's vertices from B's hyperplane is at most: $\zeta(\mathcal{E} + \epsilon_\rho)$.*

PROOF. If B's hyperplane is exactly through A's centrum, then the maximum deviation of hyperplanes is $(\zeta - 1)\mathcal{E}$. Facet A's maximum deviation adds another $\mathcal{E}$ term, and the maximum, computed radius of a centrum adds a $\zeta\epsilon_\rho$ term. ∎

The following lemmas are immediate (note that $\mathcal{E}_{\text{flipped}} < \mathcal{E}_{\text{rho\_flipped}}$):

LEMMA 6.24. *If a chord or cone facet is ρ-coplanar with a flipped or cone facet, the maximum deviation of one of its vertices from the flipped or cone facet is:*

$$\mathcal{E}_{\text{rho\_flipped}} = \zeta(\mathcal{E}_{\text{cone}} + \epsilon_\rho).$$

LEMMA 6.25. *After merging new facets into flipped or cone facets, if a new facet is ρ-coplanar with a chord facet, the maximum deviation of one of its vertices from the chord facet is:*

$$\mathcal{E}_{\text{rho\_chord}} = \zeta(\mathcal{E}_{\text{rho\_flipped}} + \epsilon_\rho).$$

LEMMA 6.26. *If a flipped facet is ρ-coplanar with a flipped facet, the maximum deviation of one of its vertices from the flipped facet is:*

$$\mathcal{E}_{\text{two\_flipped}} = \zeta(\mathcal{E}_{\text{rho\_flipped}} + \epsilon_\rho).$$

LEMMA 6.27. *If two facets are θ-coplanar in fix cone, their maximum deviation is smaller than the deviations due to ρ-coplanarity.*

PROOF. The merging facet's hyperplane can deviate by $\Delta \sin\theta$ from the other hyperplane. But ρ-coplanarity increases the maximum deviation by at least $(\zeta - 1)\mathcal{E}_{\text{cone}}$. ∎

Degenerate cones and *Fix horizon* will not change the maximum deviation above a hyperplane, so:

$$\mathcal{E}_{\text{outer}} = \mathcal{E}_{\text{two\_flipped}} \leq \zeta^3(\mathcal{E}_{\text{cone}} + \epsilon_\rho).$$

6. Error analysis of degenerate cones

Merge cone and *fix cone* fail to build a cone if the simplex test reports a negative simplex or if fewer than d cone ridges remain. In this case, $P_{furthest}$ is added to an

$\mathcal{E}_{\text{neg\_simplex}}$	when a simplex is negative
$\mathcal{E}_{\text{degen\_furthest}}$	when degenerate cone in *fix cone*

Table 6.5: Maximum deviations of $P_{furthest}$ derived for degenerate cones.

interior facet as a coplanar vertex. This also occurs if the evaluation of the cone is worse than the evaluation of the interior facets (see "if cone is better‡"), but it doesn't effect the error analysis (see Table 6.5).

LEMMA 6.28. *If the simplex test reports a negative simplex, the maximum deviation of $P_{furthest}$ above an interior facet is:*

$$\mathcal{E}_{\text{neg\_simplex}} = \max(\Delta \sin\theta, d\epsilon_\rho) + \mathcal{E}_{\text{outer}}.$$

PROOF. If the simplex of the merged cone facets is negative, its base ridge is on the opposite side of $P_{furthest}$ from its coplanar vertices. If the simplex is flipped, the maximum deviation of $P_{furthest}$ is $\mathcal{E}_{\text{flip\_furthest}} = \max(\Delta \sin\theta, d\epsilon_\rho)$ and $P_{furthest}$ is inside the simplex. Since the simplicial vertices are already on the hull, the hyperplane must pass under the outer planes of the interior facets. Hence the result. Note that $\mathcal{E}_{\text{neg\_simplex}}$ does not effect $\mathcal{E}_{\text{outer}}$ ∎

In *merge cone* and *fix cone*, a cone is degenerate if it contains fewer than d cone ridges.

LEMMA 6.29. *If fix cone or merge cone produces fewer than d cone ridges, the maximum deviation of $P_{furthest}$ above an interior facet is:*

$$\mathcal{E}_{\text{degen\_furthest}} = 2\mathcal{E}_{\text{outer}}.$$

PROOF. The proof is similar to $\mathcal{E}_{\text{neg\_simplex}}$. There are fewer than d cone ridges so a new facet must undercut $P_{furthest}$. The corresponding simplex uses horizon vertices which are at most $\mathcal{E}_{\text{outer}}$ from an interior facet. $P_{furthest}$ could be $\mathcal{E}_{\text{outer}}$ above the hyperplane. ∎

$\mathcal{E}_{\text{rho\_horizon}}$	vertex below horizon facet

Table 6.6: Maximum deviations derived for *fix horizon*.

7. Error analysis of *fix horizon*

Fix horizon insures that all horizon ridges are *clearly convex*. This section considers concave ridges, ρ-coplanar ridges, and θ-coplanar ridges. The order of merging for ρ-coplanar ridges is: new facet ρ-coplanar with horizon facet and horizon facet ρ-coplanar with new facet (see Table 6.6).

LEMMA 6.30. *Concave horizon ridges do not increase the maximum facet width.*

PROOF. Since all vertices are below the outer planes of horizon facets, a concave horizon ridge can not increase an outer plane. Consider the new facet that forms a concave angle with the horizon facet. Since the new facet is further above the horizon facet than a coplanar facet would be, a concave horizon ridge is not further below the horizon facet than a coplanar facet would be. ∎

The following is a consequence of Lemma 6.7 and the maximum deviation below a hyperplane from *fix cone*, $\mathcal{E}_{\text{two\_flipped}}$:

LEMMA 6.31. *If a new facet is ρ-coplanar with a horizon facet, the maximum deviation of a vertex below the horizon facet is:*

$$\mathcal{E}_{\text{rho\_horizon}} = \zeta(\mathcal{E}_{\text{two\_flipped}} + \epsilon_\rho).$$

As in 2-d, there is a danger of repeatedly increasing the facet width. Similarly we depend on the partition plane for $P_{furthest}$.

LEMMA 6.32. *If a horizon facet is ρ-coplanar with a new facet, the maximum deviation of a vertex below the new facet is bounded by $\mathcal{E}_{\text{rho\_horizon}}$.*

PROOF. If two facets are just *clearly convex*, the maximum deviation from avoiding ρ-coplanarity is $3\epsilon_\rho$. Consider a plane through the centrum and a vertex

$3\epsilon_\rho$ below the facet. This corresponds to Figure 6.4 of Lemma 6.8. By Lemma 6.8, a minimum outer plane of $6\epsilon_\rho + 4\epsilon_\beta$, prevents ρ-coplanarity from more than doubling the deviation (to $6\epsilon_\rho$). So $\mathcal{E}_{\text{rho\_horizon}}$ dominates. ∎

The following uses the same proof as used in *fix cone*.

LEMMA 6.33. *If two facets are θ-coplanar in fix horizon, their maximum deviation is smaller than the deviations due to ρ-coplanarity.*

8. Summary

In the error analysis of *fix cone*, $\mathcal{E}_{\text{outer}}$ is the maximum deviation of a vertex above a hyperplane. In the error analysis of degenerate cones, $\mathcal{E}_{\text{degen\_furthest}}$ is the maximum deviation of a coplanar vertex above a hyperplane (i.e., $\mathcal{E}_{\text{coplanar}}$). In the error analysis of *fix horizon*, $\mathcal{E}_{\text{rho\_horizon}}$ is the maximum deviation of a vertex below a hyperplane (i.e., $\mathcal{E}_{\text{inner}}$). Restating the corresponding bounds in terms of $\mathcal{E}_{\text{cone}}$ and ϵ_ρ gives us:

$$\begin{aligned} \mathcal{E}_{\text{outer}} &\leq \zeta^3(\mathcal{E}_{\text{cone}} + \epsilon_\rho) \\ \mathcal{E}_{\text{coplanar}} &\leq 2\zeta^3(\mathcal{E}_{\text{cone}} + \epsilon_\rho) \\ \mathcal{E}_{\text{inner}} &\leq \zeta^4(\mathcal{E}_{\text{cone}} + \epsilon_\rho) \end{aligned}$$

To get a feeling for the size of these figures in 3-d, let $\Delta = 1$, $V = 1$, $mM = 8$ (the maximum number of adjacent ρ-coplanar ridges), and $\zeta = 4$. As noted in the footnote for Lemma 5.28, the $\sin\theta$ term is approximately $\cos(\pi - \theta)$ when convexity constraints determine the maximum facet diameter.

The errors build as follows:

$$\begin{aligned} \mathcal{E}_\theta &= 4\sin\theta \\ \mathcal{E}_\rho &= 36\epsilon_\rho \\ \mathcal{E}_{\text{cone}} &\leq 5\sin\theta + 37\epsilon_\rho \\ \mathcal{E}_{\text{outer}} &\leq 320\sin\theta + 2432\epsilon_\rho \end{aligned}$$

$$\begin{aligned} \mathcal{E}_{\text{coplanar}} &\leq 640 \sin\theta + 4864\epsilon_\rho \\ \mathcal{E}_{\text{inner}} &\leq 1280 \sin\theta + 9728\epsilon_\rho \end{aligned}$$

With these figures, the maximum width of a facet is:

$$\mathcal{E}_{\text{coplanar}} + \mathcal{E}_{\text{inner}} + 2\epsilon_\rho \leq 1920 \sin\theta + 14594\epsilon_\rho.$$

This bound is pessimistic. Empirically, the maximum facet width for a cospherical distribution is about $5 \max(3\cos(\pi - \theta), 3\epsilon_\rho)$.

Chapter 7

Implementation of Quick_hull

The author implemented Point_in_polyhedron and Quick_hull in 2-d and 3-d. Both algorithms take a list of points for their input; the input to Point_in_polyhedron also includes edge lists. This chapter discusses the implementation of Quick_hull. It demonstrates the practicality of Quick_hull and the error bounds that Quick_hull achieves. At the end of this chapter, we demonstrate how the implementation helps a researcher understand the internal behavior of Quick_hull.

1. Demonstration of Quick_hull

We start with examples. Each example tests some aspect of Quick_hull. The first figure shows the convex hull of two cospherical, regular polygons. The facets inbetween the polygons are rectangles. This is mathematically correct in the real number domain, incorrect if precise arithmetic was used to construct the convex hull of the given data, and correct if roundoff error is taken into account. With precise arithmetic, all or almost all of the facets would be triangles because the input points are not precisely generated. The bottom figures show the effect of roundoff error due to very small regular polygons. The basic structure is retained but more and more facets must be merged.

The next three pages show random data. Random data, especially when cospherical, is an effective test case because weak predicates indicate the lack of a relationship. If regular data is used, then regularities in the data can mask incorrect

n	input size	h	num. of facets and ridges/facet
Δ	input diameter	δ	max facet width
θ	max angle allowed	ζ	max twist
ρ	point precision		

Table 7.1: Parameters for examples.

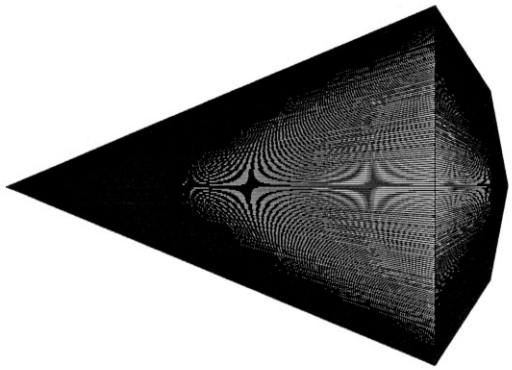
execution of the algorithm. We first show 2-d data. Since convex hull in $\mathbf{R}^2$ is a one dimensional polygon, the problem is constrained and easy to implement. The 3-d case is significantly more difficult. We show the effects of an angle constraint and of roundoff error. The last two pages of figures show the intermediate steps of Quick_hull.

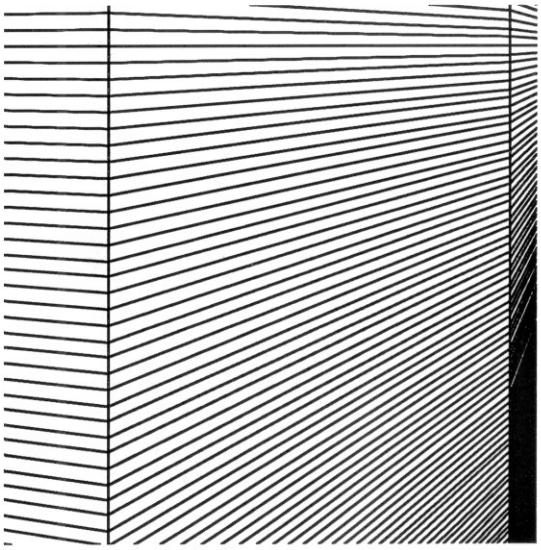
In most 3-d examples, the back side is not shown. If it is shown, dashed lines indicate the background edges. Irregularities along the edges of a figure are due to a simple procedure for hidden line removal. The random distributions are uniformly generated in the unit cube and then projected to the unit sphere, a spherical shell, or a cubical shell. To test the effect of roundoff error, uniformly random points are projected to a disk and the disk is projected to a sphere. If the disk is small enough, roundoff errors are significant. A similar construction is used for regular inputs. One or both poles may be included.

The inputs to Quick_hull include the number of points, the diameter of the point set, the maximum angle, and the point precision. The results include the number of facets generated, the average number of ridges per facet, and the maximum facet width (see Table 7.1). All computations are double precision arithmetic with machine roundoff $\beta = 2.2 \cdot 10^{-16}$.

The maximum angle is given as the deviation from $\cos(\pi)$. A single merge of new facets can widen a facet by $\max(\Delta \sin\theta, d\epsilon_\rho)$. The maximum facet width is given as a multiple of the single merge width. In these examples, the worst case multiple is $\delta = 4.7$. For angle constraints of spherical distributions, we treat the deviation from $\cos(\pi)$ as if it were the centrum's precision ρ (see Lemma 5.28).

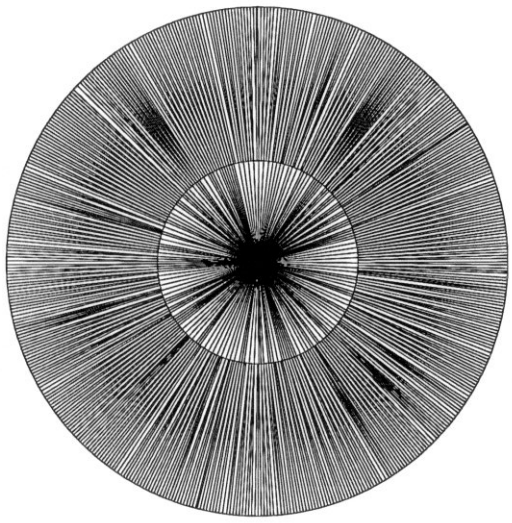
$n = 1002 \;\; \Delta = 2 \;\; \theta = 0 \;\; \rho = 0 \;\; h = 1500 \;\; 3.3 \;\; \delta = 1.0$





$n = 2002 \;\; \Delta = 2 \cdot 10^{-5} \;\; \theta = 0$
$\rho = 0 \;\; h = 1682 \;\; 3.5 \;\; \delta = 1.8 \;\; \zeta = 3.6$

$n = 2002 \;\; \Delta = 2 \cdot 10^{-6} \;\; \theta = 0$
$\rho = 0 \;\; h = 1061 \;\; 3.9 \;\; \delta = 2.3 \;\; \zeta = 2.0$



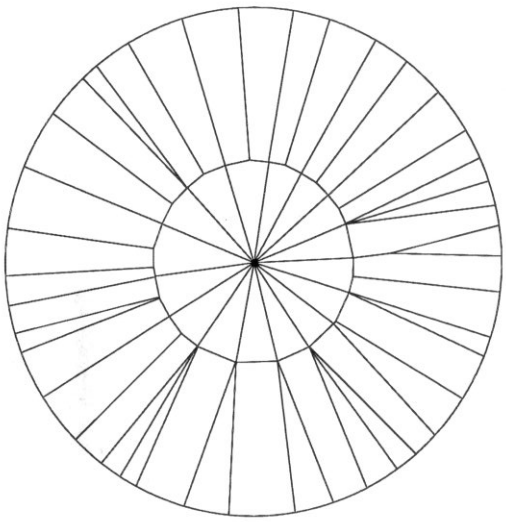


Figure 7.1: Convex hull of two regular polygons on the unit sphere. The top one is a 500-gon, the others are 1000-gons inside a very narrow disk. Note the rectangular facets in the blow-up. The narrow disks make roundoff error significant. The disk diameter of the bottom right figure is $2 \cdot 10^{-6}$.

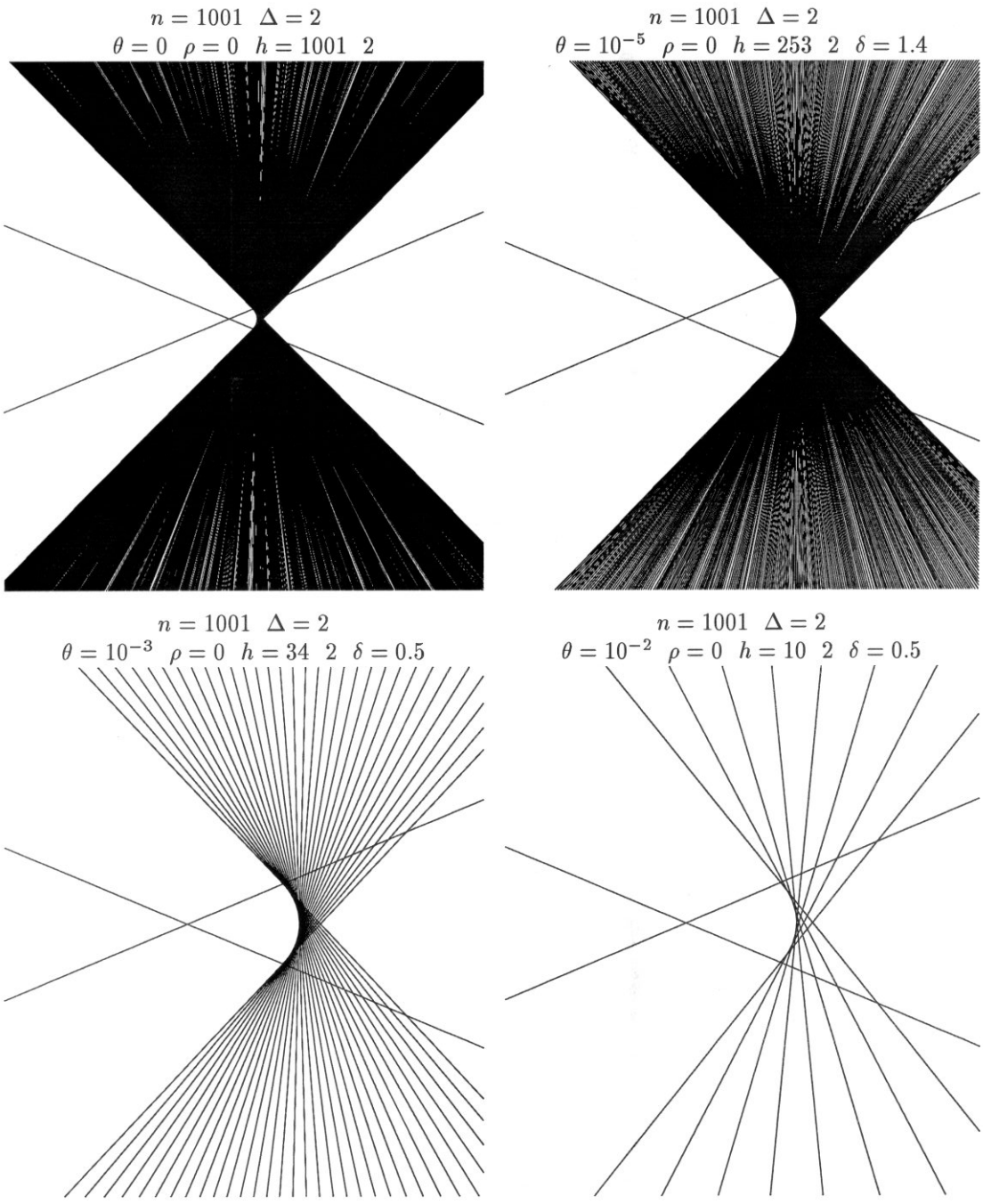
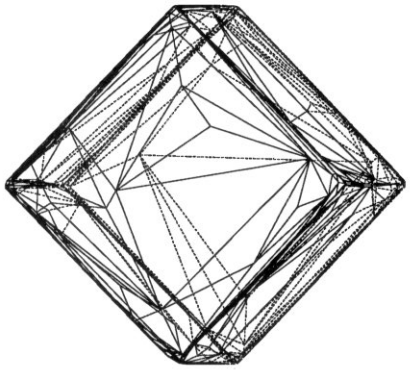
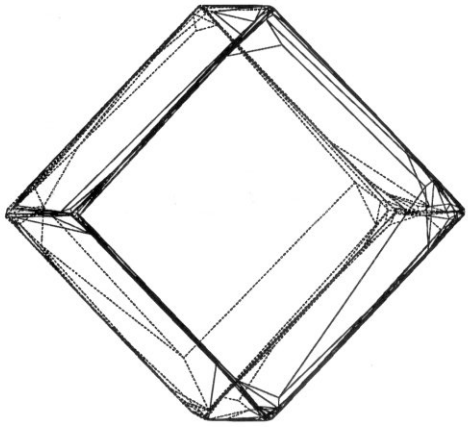


Figure 7.2: Tangent lines for convex hull of 1001 random, cocircular points. Note the increased regularity as the maximum angle decreases. The maximum angle *theta* for the bottom right figure is $\cos(\theta) = 0.99$.

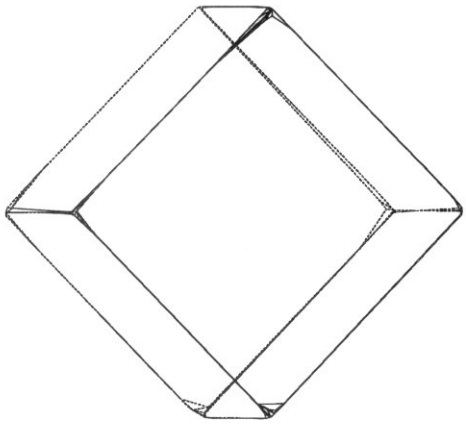
$n = 10000 \ \Delta = 1 \ \theta = 0$
$\rho = 0 \ h = 526 \ 3.0$



$n = 10000 \ \Delta = 1 \ \theta = 10^{-9}$
$\rho = 0 \ h = 416 \ 3.1 \ \delta = 0.36 \ \zeta = 1.8$



$n = 10000 \ \Delta = 1 \ \theta = 10^{-5}$
$\rho = 0 \ h = 112 \ 3.5 \ \delta = 0.27 \ \zeta = 3.7$



$n = 10000 \ \Delta = 1 \ \theta = 10^{-2}$
$\rho = 0 \ h = 10 \ 4.2 \ \delta = 0.24 \ \zeta = 0$

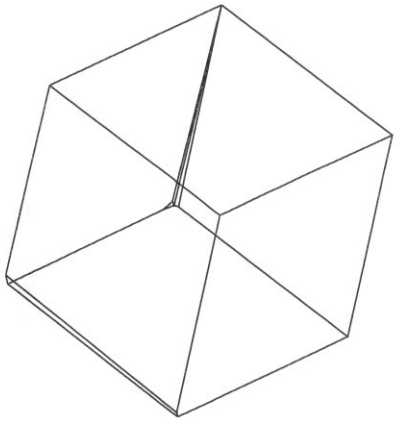


Figure 7.3: Convex hull of 10,000 random points within 10^{-3} of the boundary of the unit cube. Note the simplified output as the maximum angle decreases. The maximum angle *theta* for the bottom right figure is $\cos(\theta) = 0.99$.

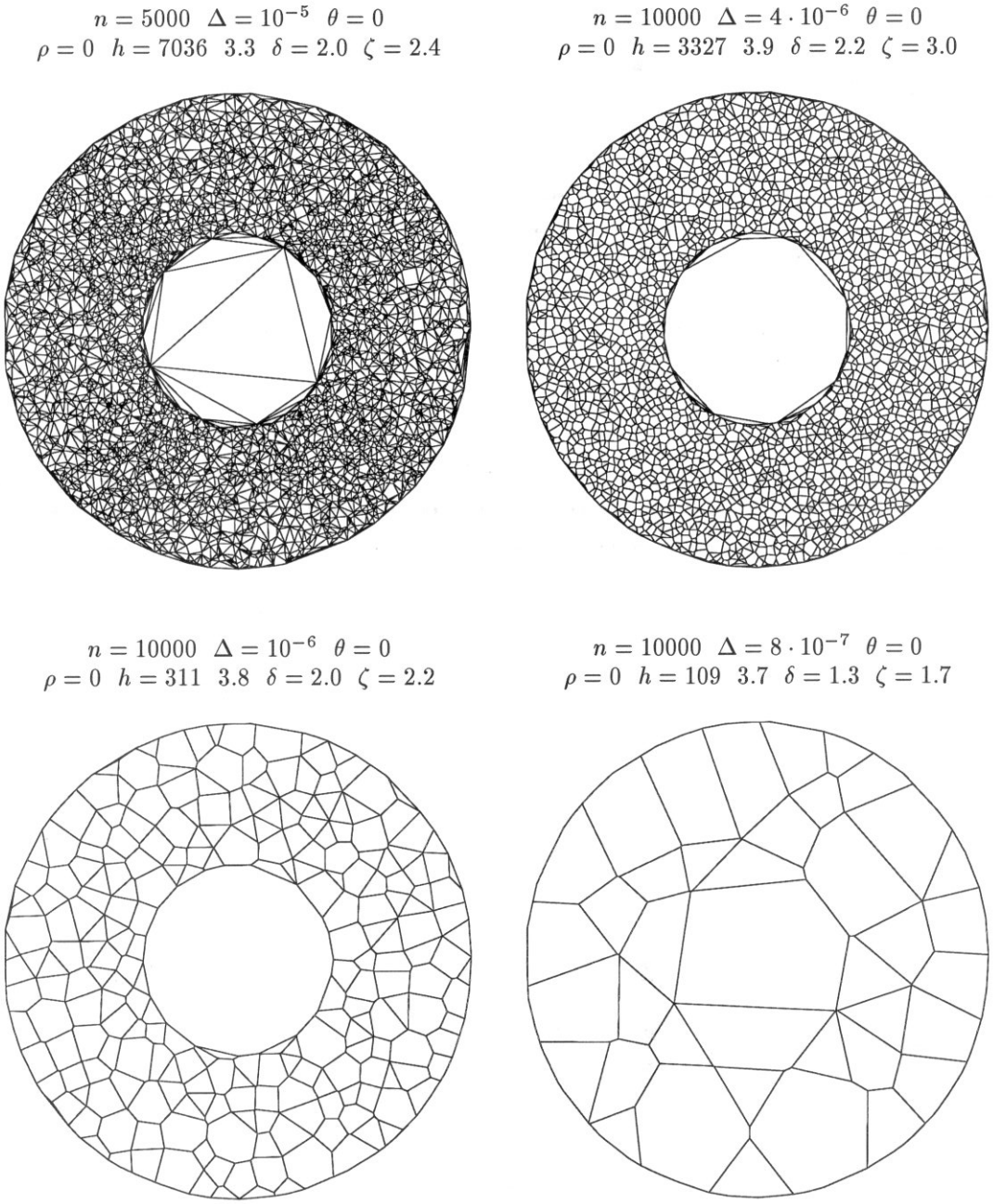


Figure 7.4: Convex hull of 10,000 random, cospherical points within narrow disks. An additional point is at the opposite pole of the unit sphere. Note that the diameters Δ decrease rapidly. The diameter of the smallest disk is $8 \cdot 10^{-7}$. Its z-coordinates consist of 13 "9"'s followed by four other digits.

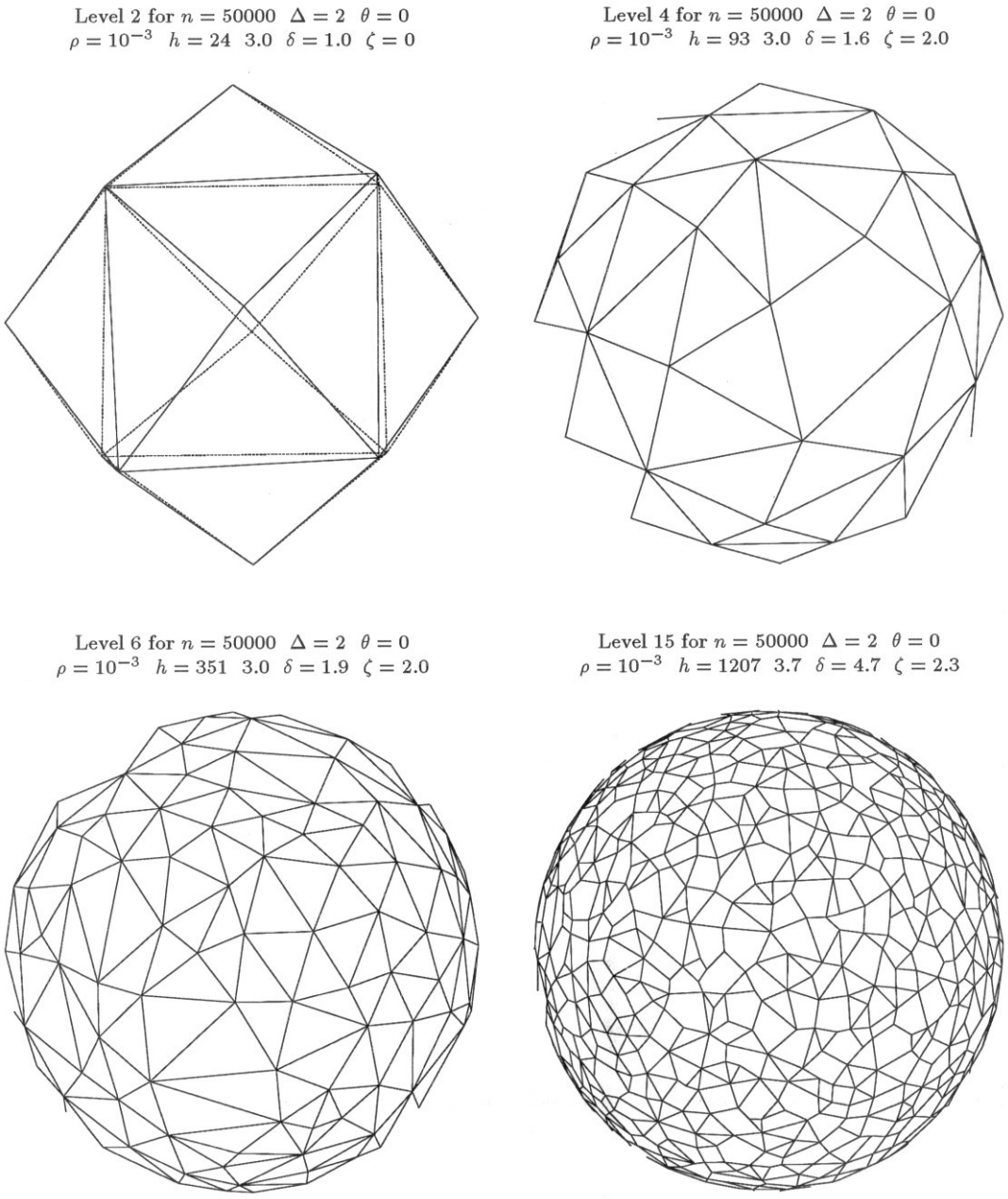


Figure 7.5: Processing levels of Quick_hull for 50,000 random points within a 0.1 shell of the unit sphere. Note the regular structure produced by selecting the locally, furthest point.

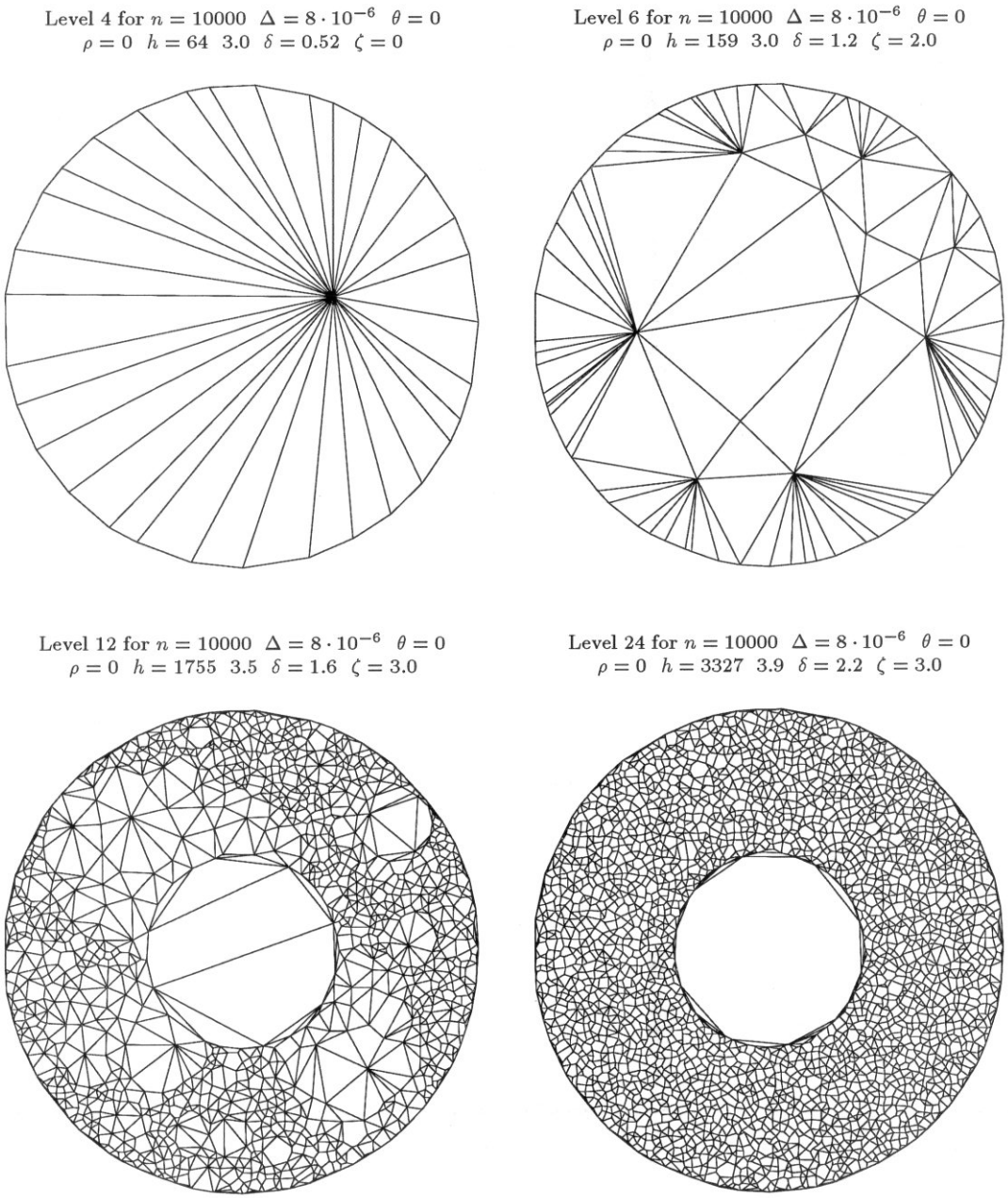


Figure 7.6: Processing levels of Quick_hull for 10,000 random points in a $4 \cdot 10^{-6}$ disk on the unit sphere. Note that the structure of the distribution does not appear until later. This is because the first $P_{furthest}$ points are on the edge of the disk.

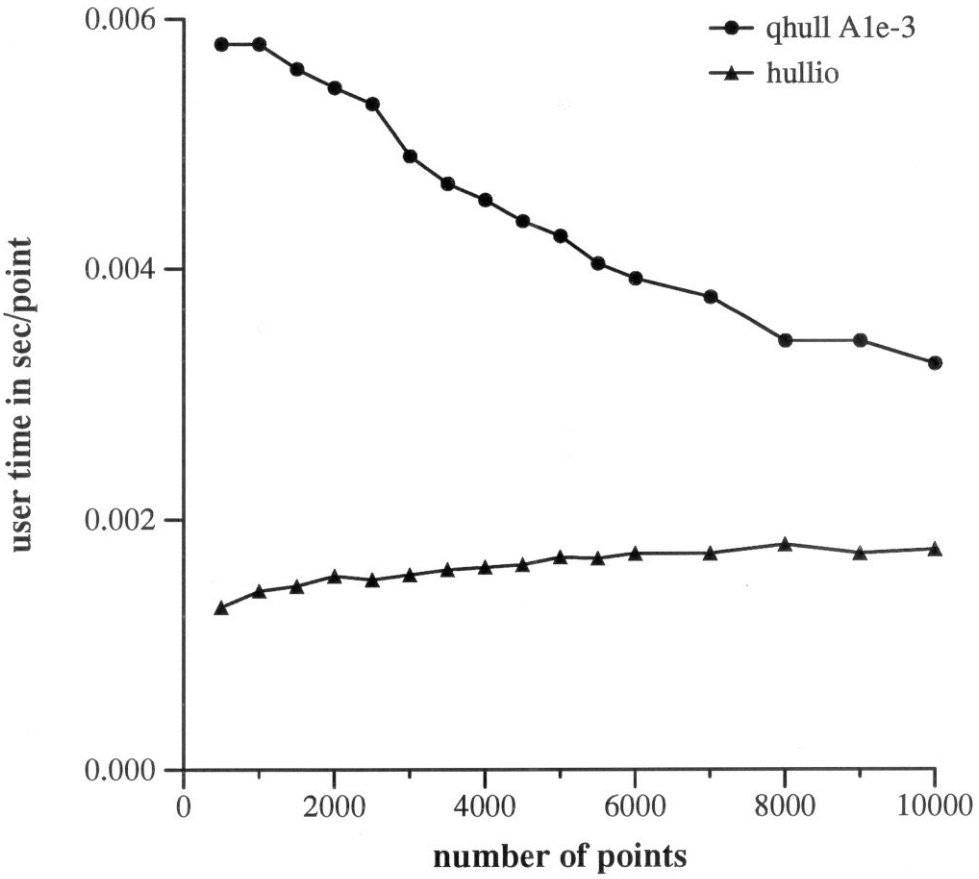


Figure 7.7: Time comparison for Quick_hull (qhull) vs. Clarkson et al.'s [1992] algorithm (hullio). Note the decreasing cost per point for Quick_hull. All points are randomly distributed on the unit sphere. The maximum angle for Quick_hull is $cos^{-1}(-0.999)$.
1

2. Statistics for Quick_hull

We compared the implementation of Quick_hull to Clarkson et al's [1992] randomized, convex hull algorithm. Dorward implemented the algorithm (hullio) in general dimension and tuned it for performance [Dorward 1992]. As discussed in Section 4.2, the two algorithms perform the same steps but in a different order. Quick_hull partitions all of the points for a facet when the facet is created, while Clarkson's algorithm tests a point against each facet when it is added to the hull. In addition, Quick_hull handles imprecise data and arithmetic while Clarkson's

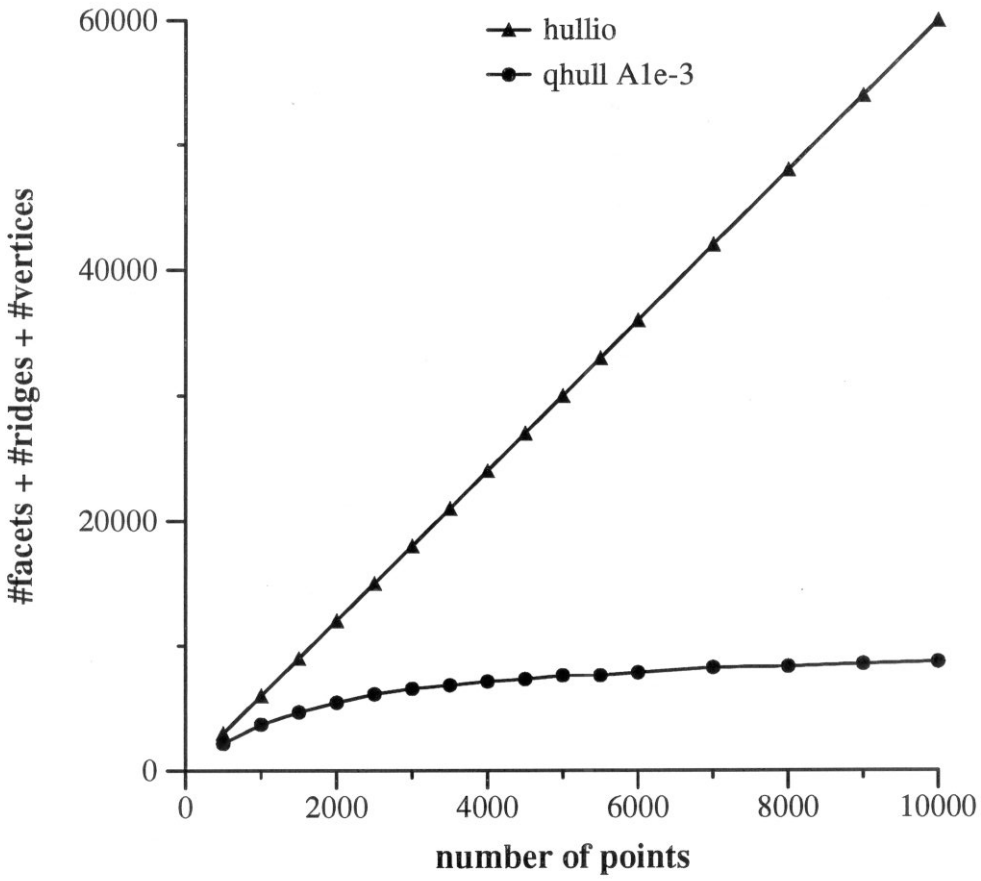


Figure 7.8: Output comparison for Quick_hull (qhull) vs. Clarkson et al.'s [1992] algorithm (hullio). Note the slow growth for Quick_hull. All points are randomly distributed on the unit sphere. The maximum angle for Quick_hull is $cos^{-1}(-0.999)$.

algorithm assumes precise data and arithmetic.

Figure 7.7 shows that hullio's performance is consistent with the $O(n \log n)$ analysis of Clarkson et al [1992]. It also shows that with increased input size, Quick_hull's cost per point decreases. The reason is that the maximum output size for Quick_hull is constrained by the maximum angle between adjacent facets [Lemma 5.28]. Figure 7.8 plots the total output size. Quick_hull's growth in output size is nearly flat, while the size of the precise convex hull grows linearly. In 4-d, the size of the precise convex hull will grow quadratically, while Quick_hull's growth will become flat. For many applications, the exact facets are not needed.

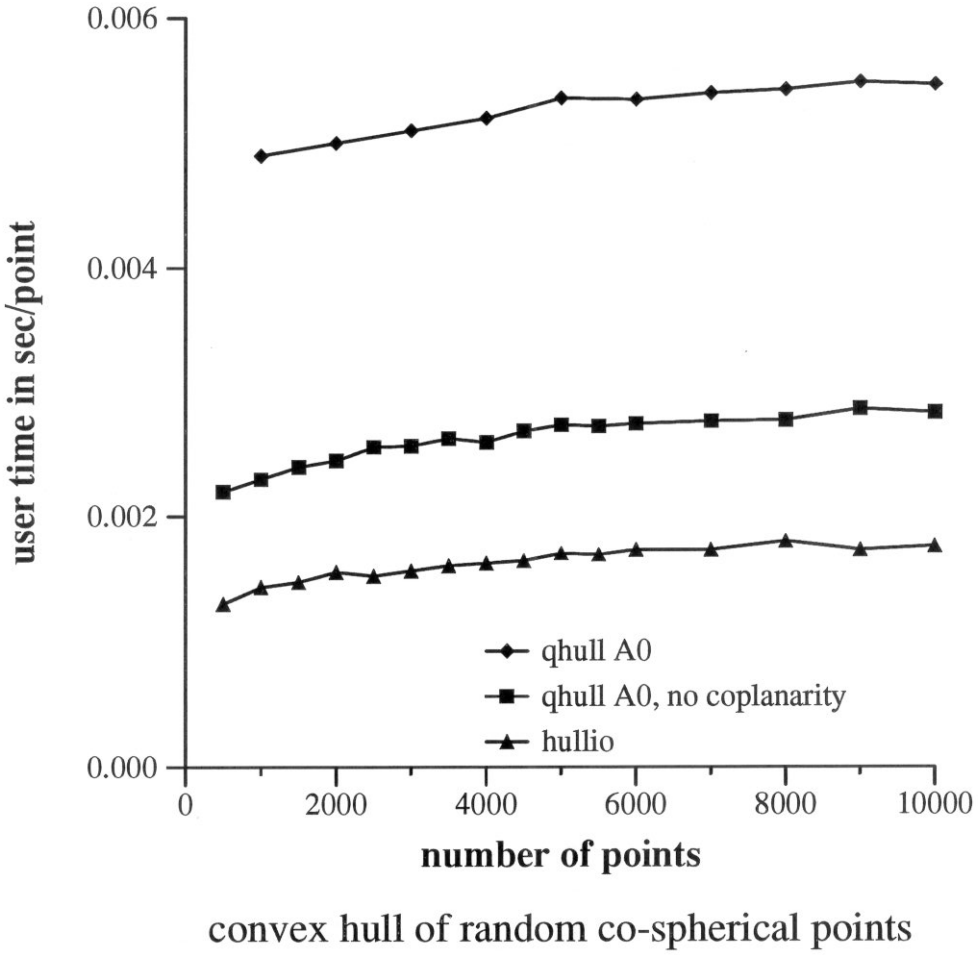


Figure 7.9: Cost of coplanarity testing in Quick_hull on points in general position. Without coplanarity testing (unnecessary in this case), Quick_hull runs about twice as fast. The hullio plot line is for comparison.

Figure 7.9 compares Quick_hull with and without coplanarity testing. The upper-most plotted line shows Quick_hull producing the exact convex hull of the input. Note that it is consistent with the $O(nlogn)$ complexity of Clarkson et al's algorithm (duplicated as the bottom-most line). Since the points are in general position, coplanarity testing is unnecessary. The middle line shows Quick_hull with coplanarity testing turned off. As currently implemented, coplanarity testing approximately doubles the execution time of Quick_hull.

One advantage of Quick_hull over Clarkson et al.'s algorithm is that it always processes a point that is locally furthest from a facet. While Clarkson et al. add a random point to the hull, Quick_hull usually adds an extreme point. This is best seen if a cospherical distribution of points is inscribed in a cube. The cube is just small enough to include one cospherical point on each face. In this case Quick_hull uses two processing levels: one to build the initial hull of maximum points, the other to add the cube's vertices. Clarkson et al.'s algorithm will construct large portions of the spherical hull only to destroy it when adding a cubical vertex.

Quick_hull collects statistics on all aspects of its behavior. Figure 7.11 shows the most important statistics for executing Quick_hull on 5,001 cospherical points in a narrow disk. Because of fixed precision arithmetic only a fifth of the points are vertices. The next page shows additional statistics. Some interesting features are:

- The errors due to *merge cone* are smaller than the errors due to *fix cone* and *fix horizon*. Failed furthest points give the highest outer plane while *fix cone* gives the lowest inner plane.

- *Partition points* incurs most of the cost. Work is measured in number of inner products performed, or their equivalents.

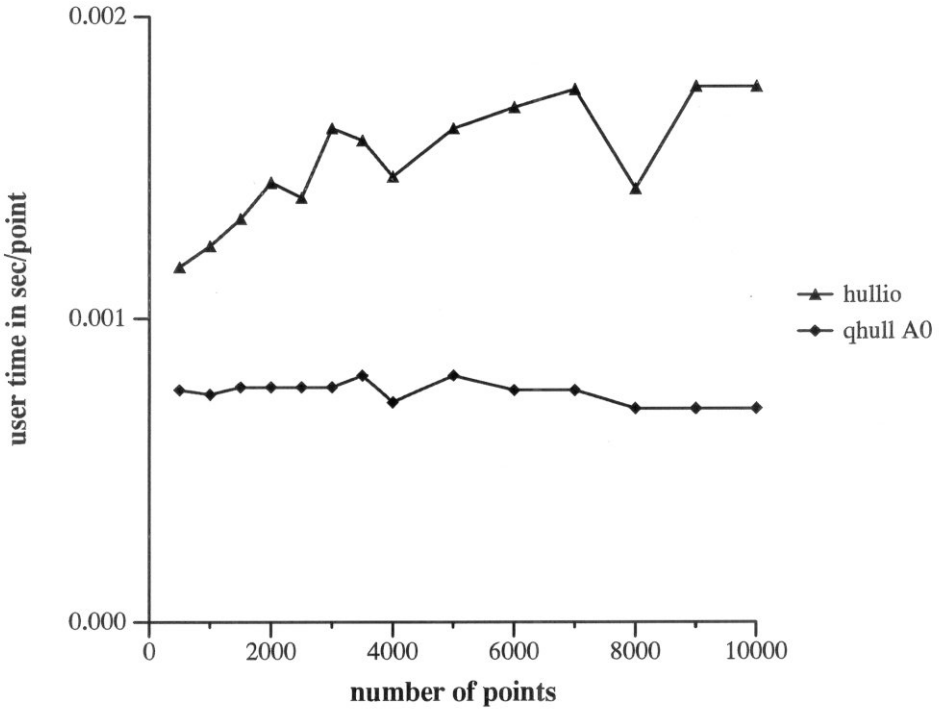
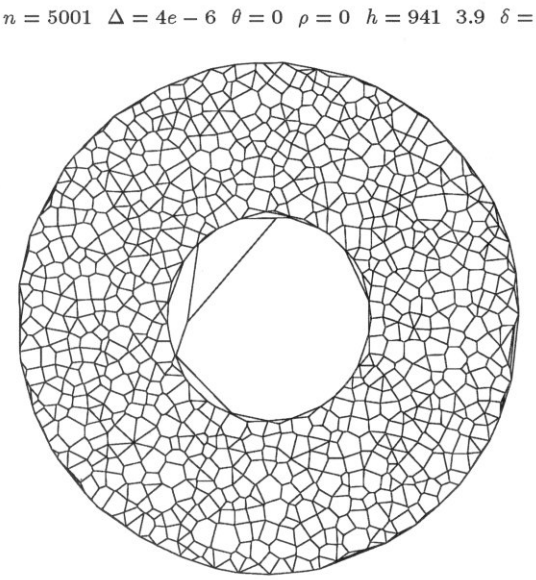


Figure 7.10: Advantage of Quick_hull's *partition points* over Clarkson et al's [1992] algorithm. When a sphere of 10,000 points is inscribed in a cube, Quick_hull runs more than twice as fast as hullio on 10,000 points. To prevent initialization from identifying the cube, the cube is reduced so that each face includes one point of the spherical distribution.



917 vertices. 941 facets. 3.9 ridges apiece. 19 processing levels.
min. angle in facet 171 (-cos (x))=8.95499e-07
max angle in facet 3317 (log(1-cos(x))= -15.9546
max outer hull in facet 5960= 9.47e-15 ave.= 3.86e-15
max inner hull in facet 5845=-1.05e-14 ave.=-2.57e-15e
max hull diff. in facet 3778= 1.86e-14

Figure 7.11: Convex hull of 5000 cospherical points in a $4 \cdot 10^{-6}$ disk. The most important statistics are here. A subset of the available statistics are on the next page. The second column gives the total or max/min value. The other columns give the value for every other processing level of Quick_hull.

description	all	1	3	5	7	9	11	13	15	17	19
**** initialization ****											
work done by initialization	45070	45070									
*** findhorizon ****											
num. of furthest points	1216	2	8	22	26	123	221	131	59	18	4
ave. distance above facet	5e-09	9.3e-07	1.5e-07	5.6e-09	9e-14	1.8e-14	7.8e-15	6.3e-15	5.9e-15	5e-15	6.8e-15
ave. number of interior facets	1	0.5	1	2.5	2.9	1.5	0.81	0.66	0.54	0.5	0.5
max. number of interior facets	14	1	1	14	8	7	5	5	2	1	1
ave. number of horizon facets	5.3	3.5	4	5.8	6.3	5.2	5.2	5.3	5.5	5.7	6.5
max. number of horizon facets	19	4	4	19	11	10	14	12	10	10	10
ave. distance below horizon	-2.8e-08	-1.2e-06	-1.9e-07	-4.2e-07	-1.5e-07	-1e-08	-1.9e-09	-4.6e-10	-1.2e-14	-1.4e-14	-1.4e-14
work done by findhorizon	8955	10	48	205	266	937	1549	915	418	130	32
work done by makecone	84513	91	416	1651	2145	8242	14924	9061	4251	1339	338
*** mergecone ****											
num. of concave cone ridges	23					2	4	3	3	1	
merge chord into cone facet	5					1		2	1		
num. of coplanar cone ridges	1315			3	10	81	246	184	87	35	9
num. of times all facets are coplanar	8							2	1		
furthest point inside simplex	7							3	1		
num. of unflipped facets	761			2	8	52	145	96	50	19	6
num. of flipped facets	534			1	2	29	101	81	35	15	3
num. of points inside simplex	2										
ave. facets after mergecone	4.2	3.5	4	5.6	6	4.5	4.1	3.8	4	3.8	4.2
cone facets: num. vertices above	166				2	16	33	20	12	5	
cone facets: ave. distance above	2.6e-15				3.4e-15	2.8e-15	2.6e-15	2.7e-15	2.6e-15	3.4e-15	
cone facets: max. distance above	5.8e-15				5.1e-15	4.2e-15	4.6e-15	5e-15	4.4e-15	5.8e-15	
cone facets: num. vertices below	100				1	7	16	8	8	3	2
cone facets: ave. distance below	1.2e-15				8.3e-16	1.6e-15	1.3e-15	1.6e-15	1.1e-15	1.9e-15	3.7e-16
cone facets: max. distance below	4.4e-15				8.3e-16	3.8e-15	3.7e-15	3.1e-15	1.9e-15	3.3e-15	4.3e-16
flipped facets: num. vertices above	167			1	1	14	28	21	10	6	
flipped facets: ave. distance above	2.2e-15			2.4e-15	3.1e-15	2.1e-15	2.1e-15	2.4e-15	2.1e-15	2e-15	
flipped facets: max. distance above	5.5e-15			2.4e-15	3.1e-15	3e-15	3.7e-15	5.5e-15	3.5e-15	2.6e-15	
flipped facets: num. vertices below:	72					4	12	13	4	3	1
flipped facets: ave. distance below	1.9e-15					3.4e-15	1.9e-15	2.6e-15	2e-15	8.5e-16	4.4e-15
flipped facets: max. distance below	5.5e-15					4.3e-15	3.3e-15	5.5e-15	4.5e-15	2.3e-15	4.4e-15
work done by mergecone	50805	21	116	468	753	3899	9348	6289	3050	1116	294
work due to concave facets	1635					135	264	274	201	98	
*** fixcone ****											
num. of chord ridges	18					1	4	1	2	1	
num. of non-convex chord ridges	7						2			1	
num. of non-convex cone ridges	301					5	52	49	22	10	2
num. of non-convex horizon ridges	788			3	13	81	166	86	41	13	2
num. of merges proposed	1729			3	14	106	347	227	115	40	4
merge new into horizon	1113			3	9	76	239	136	68	23	2
merge horizon into new	366				5	23	66	47	30	7	
merge new into new	250					7	42	44	17	10	2
ave. merged outer	3.7e-15			3.1e-15	3.7e-15	3.4e-15	3.6e-15	3.9e-15	3.8e-15	4.3e-15	3.9e-15
max. merged outer	8.9e-15			3.1e-15	8.3e-15	7.2e-15	7.6e-15	8.1e-15	7.5e-15	8.9e-15	6.4e-15
ave. merged inner	-4.1e-15			-3e-15	-4.2e-15	-4.2e-15	-4.1e-15	-4.1e-15	-4.4e-15	-4.2e-15	-4.5e-15
max. merged inner	-2.4e-14			-6.2e-15	-1.1e-14	-9.2e-15	-9.1e-15	-2.3e-14	-9.7e-15	-1.1e-14	-6.6e-15
work done by fixcone	54770	23	96	461	732	3969	10408	6811	3350	1217	160
*** cone better ****											
num. of points added to hull	966	2	8	22	26	123	173	89	41	11	2
max. distance of added point	1.9e-06	1.9e-06	1.9e-07	2e-08	3.1e-13	9.5e-14	3.4e-14	2.7e-14	1.9e-14	1e-14	1.7e-14
*** partition points ****											
ave. number of new facets	4.4	3.5	4	5.6	6	4.4	4.1	4	4.2	3.7	5.5
max. num. of new facets	19	4	4	19	10	9	8	7	7	6	7
ave. length of outside list	16		160	79	45	13	5.2	3.6	2.8	3.2	0.45
max. length of outside list	909	0	754	809	521	118	96	65	55	71	2
num. of times point is above	71025	4997	4982	9782	6960	6570	3107	974	380	165	
ave. distance above a facet	6.9e-09	9.1e-08	5.2e-10	4.8e-13	6.8e-14	2.2e-14	1.2e-14	1e-14	8.6e-15	1.1e-14	
max. distance above a facet	1.9e-06	1.9e-06	6e-08	1e-09	3e-13	8.2e-14	6.3e-14	3.5e-14	2e-14	2e-14	
a point coplanar to a facet	17365	20	15	271	744	2148	3114	1567	738	203	15
work done by partition points	479195		19934	119049	49004	39036	20364	7490	3113	786	48
*** fix horizon ****											
num. of merges	1203			3	14	100	233	149	76	24	2
ave. merged outer	3.5e-15			3.1e-15	3.7e-15	3.4e-15	3.5e-15	3.8e-15	3.6e-15	3.8e-15	3.1e-15
max. merged outer	8.3e-15			3.1e-15	8.3e-15	7.2e-15	6.9e-15	8.1e-15	7.5e-15	6.4e-15	3.1e-15
ave. merged inner	-3.8e-15			-3.1e-15	-4.2e-15	-4.1e-15	-3.7e-15	-3.7e-15	-4.2e-15	-4.4e-15	-3.6e-15
min. merged inner	-1.1e-14			-6.2e-15	-1.1e-14	-9.2e-15	-8.9e-15	-9.9e-15	-1.1e-14	-8e-15	-3.9e-15
work done by fixhorizon	61191			156	667	4856	11972	7808	3938	1390	107
*** cone worse ****											
num. of failed furthest points	250						48	42	18	7	2
ave. num. of interior facets	1.5						1.5	1.4	1.4	1.6	1
max. num. of interior facets	3						3	3	3	2	1
ave. length of outside list	4.1						4.4	4.4	5	8.9	1.5
failed due to mergecone	15							5	2		
max. distance if mergecone	4.5e-15							3.9e-15	3.4e-15		
failed due to fixcone	105						16	17	7	4	2
max. distance if fixcone	1e-14						5.8e-15	6e-15	5.3e-15	4e-15	3.5e-15
failed evaluation	129						32	19	9	3	
max. distance if failed	1.5e-14						7.1e-15	8e-15	7e-15	4.6e-15	
work due to failed furthest	2815						588	419	266	190	3
*** raise outers ****											
num. of points tested	45628										45628
num. of points above facet	24										24
ave. distance outer raised	2.1e-15										2.1e-15
max. distance outer raised	4.7e-15										4.7e-15
work done by raiseouters	45628										45628
*** other ****											
ave. twist for merged facet	1.2				0.74	1.3	1.1	1	1.2	1.2	
max. twist for merged facet	3	0			0.74	2	2.3	1.7	2	2	
redundant ridges for merging	53					2	9	8	5	1	
redundant ridges deleting a facet	4					1	1				
total work	836433	45215	20610	121990	53567	61074	69417	39067	18587	6266	48466

3. Visualization of Quick_hull

We used visualization to develop and implement Quick_hull. It was necessary because of the counter-intuitive behavior of imprecise data and arithmetic. Quick_hull displays its output and intermediate steps with Cheyenne, a device-independent graphics package [Dobkin & Koutsofios 1988]. Almost everything in Quick_hull's data structures could be displayed graphically or by annotation. For example, we verified geometric relationships by aligning a facet with the y-z plane.

One potential problem was that a large convex hull contains megabytes of information. To display large convex hulls, Quick_hull allows selective viewing. Options included: zooming, identifiers, distances, simplices, hyperplane intersections, coplanar points, outside points, precision balls, statistics, selective output of the data structures, rotations, background display, and simplex volumes.

To study the execution of Quick_hull, each procedure includes trace output that is optionally displayed. To study the behavior of Quick_hull, a movie facility traces the execution of Quick_hull within a geometric window. The important intermediate steps of a Quick_hull iteration can be displayed.

The user controls Quick_hull by commands issued at the graphical display. This allows for rapid exploration of the data structure and the algorithm's behavior. This was important for debugging, redesigning, and verifying the algorithm. Most conceptual and non-trivial implementation errors were detected by extensive error checks. At any time, the data structures can be checked for consistent topological and geometric information.

To illustrate the above features, consider the effect of a conceptual error: not testing acute angles against an interior point in "if R is clearly convex†". If this is not done, Quick_hull can create a loop of facets with clearly convex ridges. In 2-d, Figure 7.12 redisplays a figure from Chapter 2. Angles α, β, γ are locally convex but facets a and b have reverse orientations. Such configurations can not occur in 2-d since $P_{furthest}$ partitions the remaining points into two. In 3-d though, it does occur.

A geometric consistency test detected this problem (Table 3). Quick_hull displayed the offending facet graphically and as a data structure. The facet consists

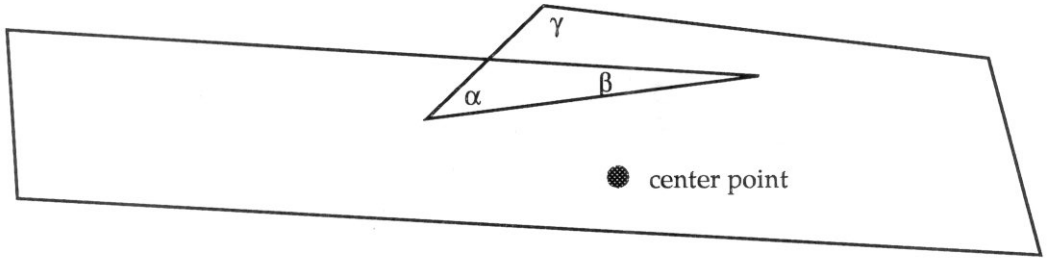


Figure 7.12: Clearly convex angles may wind more than once around an interior point

```
testhull: inside_point outside of facet 25646: dist 0.99 clearly? 1

f25646 -1 -1 0 0 0 v:422* 10011* 6119@ h
        ridges: r51952( -1.5 v6119 v10011 x12)
                r51957 ( 0.3 v422 v10011 x12)
                r46507 ( 0.3 v6119 v422 cx12)
        hyperplane in=-1.7e-15 out= 0.025 off= 0.99
                norm= 0.296735 0.711 872 -0.636543
```

Table 7.2: Error reported when executing Quick_hull. Redundant ridges are not merged, and acute angles are not tested.

of vertices 422, 10011, and 6119. Vertex 6119 was the apex of the facet's simplex (indicated by '@').

The next series of figures show how Facet 25646 was built:

It starts by tracing Vertex 6119. This creates four new facets, two of which include the corresponding horizon facets (Facet 22961 and 22965). Then Vertex 2299 is added to Facet 21005. It creates four new facets, three of which merge with the corresponding horizon facets. The next page shows four of the five steps in adding Vertex 6472 to Facet 22965. The third page shows the result. Facet 25646 is almost perpendicular to the other facets.

The local convexity of the facets is best seen in the last figure where Facet 24247 is horizontal. Facet 24247 consists of Vertices 422, 6119, 11373, and 2299. It is connected to Facet 25646 by Ridge 46507. Facet 25646 is folded underneath Facets 24247 and 24248; it consists of Vertices 422, 6119, and 10011. Facet 25646 is connected to Facet 25644 by Ridge 51952. Like Ridge

46507, Ridge 51952 is acute. Note that Vertex 6472 on Facet 25644 ends up above Facet 25646 (The vertices of Facet 25646 have x-coordinates of 0.999, 0.998, and 0.996; Vertex 6472 has an x-coordinate of 0.999). Facet 25644 is connected to Facet 25641 by Ridge 51945.

The fourth page gives a transcript of adding Vertex 6472 to the hull.

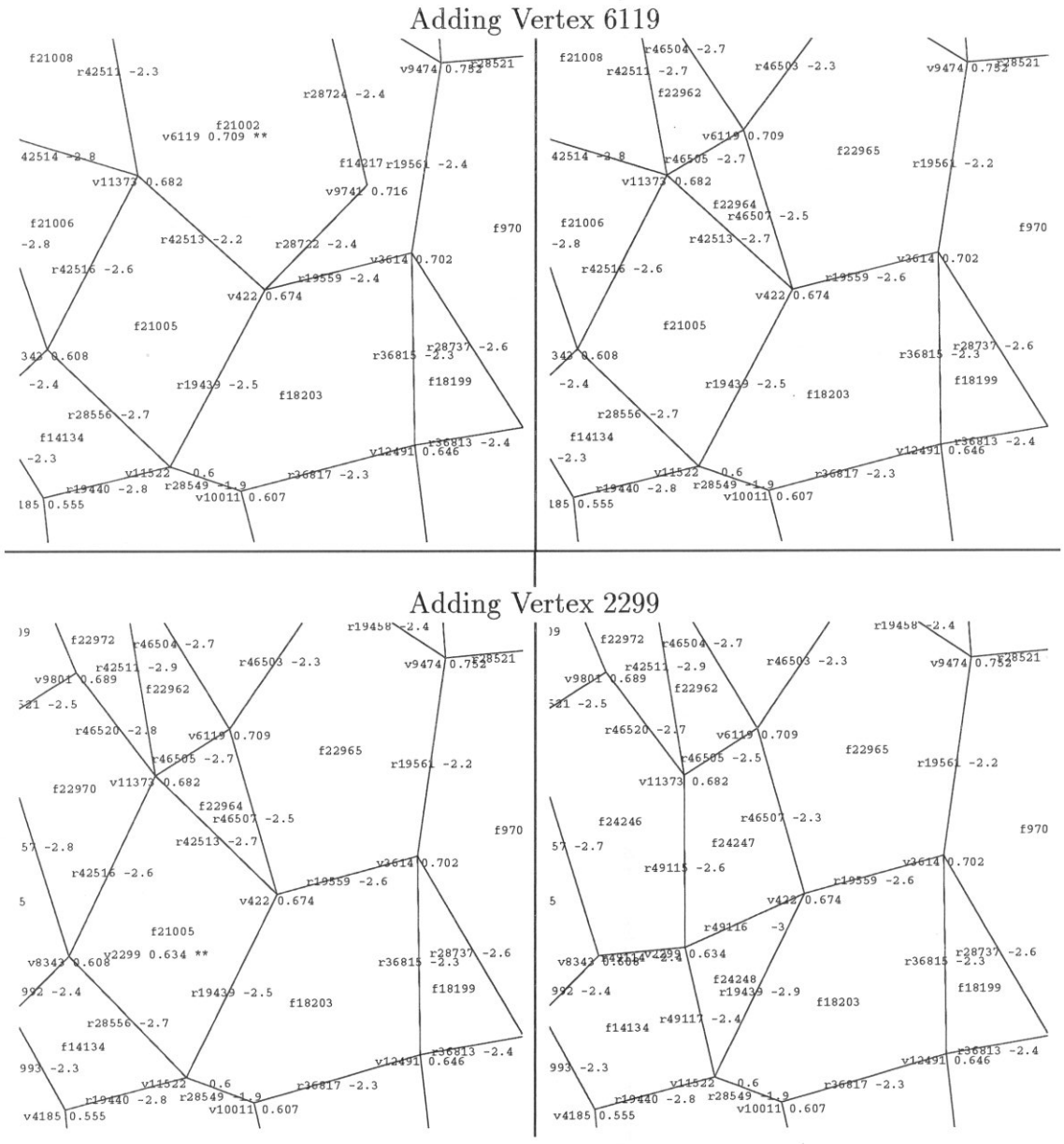


Figure 7.13: Adding two vertices prior to the creation of Facet 25646.

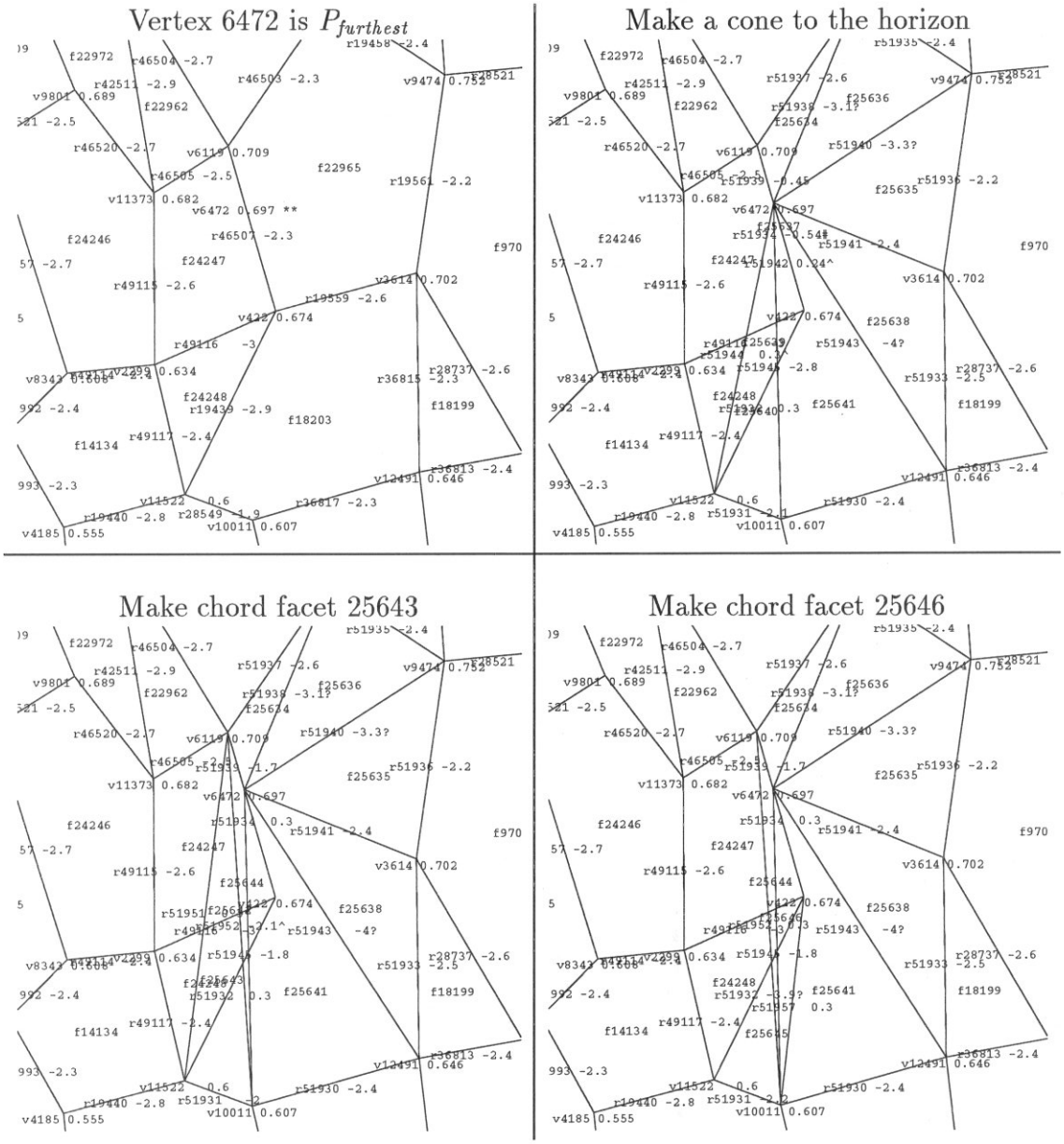


Figure 7.14: Four steps in erroneously processing Vertex 6472. These correspond to the execution log below.

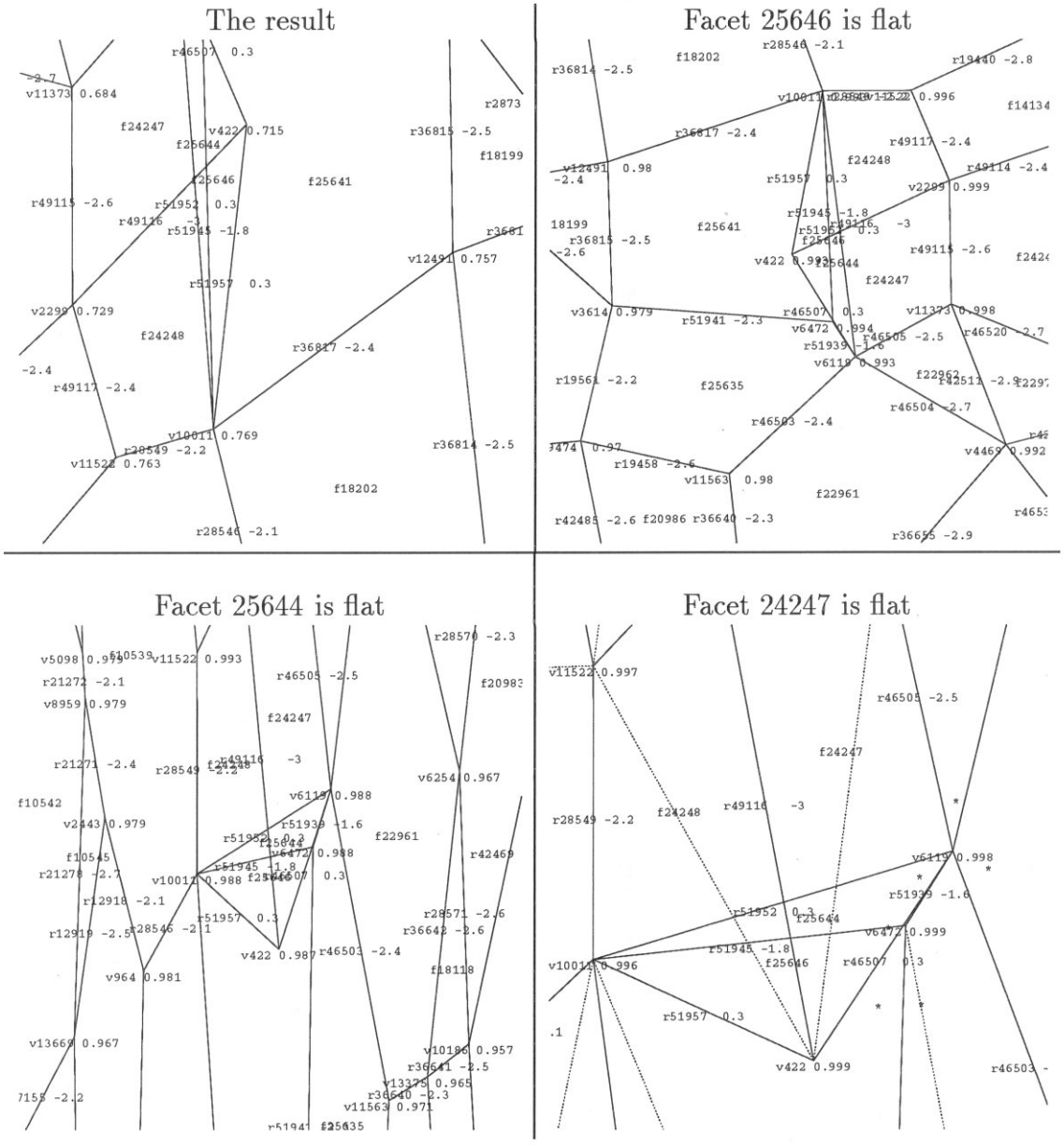


Figure 7.15: The final step in erroneously processing Vertex 6472. Plus views of three facets aligned with YZ plane.

```
********Start tracing at v6472 0.0014 above facet 18203********************
removeactive: facet 18203
findhorizon= for furthest 6472 on facet 18203
findhorizon.. interior facet 22965 dist 0.00046
findhorizon.. horizon facet 18202 dist -0.01 classify 1
findhorizon.. horizon facet 10539 dist -0.016 classify 1
findhorizon.. horizon facet 24248 dist -0.0021 classify 1
findhorizon.. horizon facet 18199 dist -0.0084 classify 1
findhorizon.. horizon facet 24247 dist -0.0002 classify 1
findhorizon.. horizon facet 20986 dist -0.0076 classify 1
findhorizon.. horizon facet 9703 dist -0.011 classify 1
findhorizon.. horizon facet 22961 dist -0.0015 classify 1
findhorizon: done
makecone= for point 6472 to horizon starting at 51937

******Cone built for new facets about 6472 **********
testridgelist= starting with cone ridge 51943, skipflipped?1
mergecone= of 8 cone ridges 51943 from new facets 25641
makenonconcave= 8 ridges with concave ridge 51944 and furthest 6472
mergefacets= 25640 and 25639 across ridge 51944
mergefacets= 25639 and 25637 across ridge 51942
makechordfacets= for furthest 6472, cone facet 25637, depth 1
makecone= for point 6119 to horizon starting at 51949
attachnewcone= attach chord facets 25644 in place of old cone facet 25637

******Cone of chord facets for new furthest 6119 **********
testridgelist= starting with cone of new chord ridges ridge 51951, skipflipped?1
mergecone= of 1 cone ridges 51951 from new facets 25644
makenonconcave= 1 ridges with concave ridge 51951 and furthest 6119
mergefacets= 25643 and 25642 across ridge 51951
makechordfacets= for furthest 6119, cone facet 25642, depth 2
makecone= for point 422 to horizon starting at 51956
attachnewcone= attach chord facets 25646 in place of old cone facet 25642

******Cone of chord facets for new furthest 422 **********
testridgelist= starting with cone of new chord ridges ridge -1, skipflipped?1
makechordfacets: finished depth 2
mergechordfacets= for cone facet 25646, point 422, chordlist 51957
testridgelist= starting with chord ridge 51957, skipflipped?0
mergecone: 1 chord ridges remain, first is 51957
testridgelist= starting with cone of new chord ridges ridge -1, skipflipped?1
makechordfacets: finished depth 1
mergechordfacets= for cone facet 25644, point 6119, chordlist 51952
testridgelist= starting with chord ridge 51952, skipflipped?0
mergefacets= 25638 and 25641 across ridge 51943
mergefacets= 25634 and 25636 across ridge 51938
mergecone: 1 chord ridges remain, first is 51952
testridgelist= starting with cone ridge 51945, skipflipped?1
capturevertices= inc inner/outer hulls of newfacets 108 for their vertices
fixcone= test all new and horizon ridges and merge if non-convex
testridgelist= starting with fixcone chord ridge 51957, skipflipped?0
testridgelist= starting with fixcone cone ridge 51945, skipflipped?0
bestmergeinside= find best merge for new facets across ridge 51940
bestmergeinside.. merge new facet 25636 into another 25635 across ridge 51940 outer 0.0014 inner -6.4e-05
testridgelist= starting with fixcone chord ridge 51957, skipflipped?0
testridgelist= starting with fixcone cone ridge 51945, skipflipped?0
testridgelist= starting with fixcone horizon ridge 51937, skipflipped?0
bestmergeinside= find best merge for new facets across ridge 51932
bestmergeinside.. need to merge new facet 25645 into old facet 24248 across horizon ridge 51932 outer 0.00052 inner -1.6e-15
markcoplanar: facet 25645 is coplanar to horizon facet 24248
testridgelist= starting with fixcone chord ridge 51957, skipflipped?0
testridgelist= starting with fixcone cone ridge 51945, skipflipped?0
testridgelist= starting with fixcone horizon ridge 51937, skipflipped?0
fixcone: done
evalcone: 0.0014
evalcone: 0.0021
attachcone= attach cone of new facets to horizon
fixhorizon= bestmerge non-convex ridges, starting with horizonlist 46503
bestmerge= find best merge for facets across ridge 19439
bestmerge.. merge facet 25645 into facet 24248 across ridge 19439 outer 0.00052 inner -1.6e-15
mergefacets= 25645 and 24248 across ridge 19439

*****finished cone for furthest 6472 above facet 18203 giving 25646
2600 vertices 2203 facets 4.4 ridges apiece. 18 maxlevel
max possible facets 7998.5 max error 0.002 beta 2.22e-16
build 4178 , failed cones 1542, lost vertices 25 tot size 9604
max distance roundoff 1.6e-15 max new inner/outer plane 1.9e-15
min angle in facet 25646 (-cos (x))= -0.99896
max angle in facet 10260 (log(1-cos(x))= -2.99984
max outer hull in facet 19436= 0.0047 ave.= 0.000851
max inner hull in facet 8935=-0.00439 ave.=-0.00045
max hull diff. in facet 25409= 0.00732 max twist=2.2 (¿4 0)
thickness=0.00171 work=1.95e+06 lvl/logh= 4.89
work/nlogh= 37.8 work/n= 139 work/h= 405 dist/n= 98.8
```

Table 7.3: Transcript of Quick_hull adding Vertex 6472.

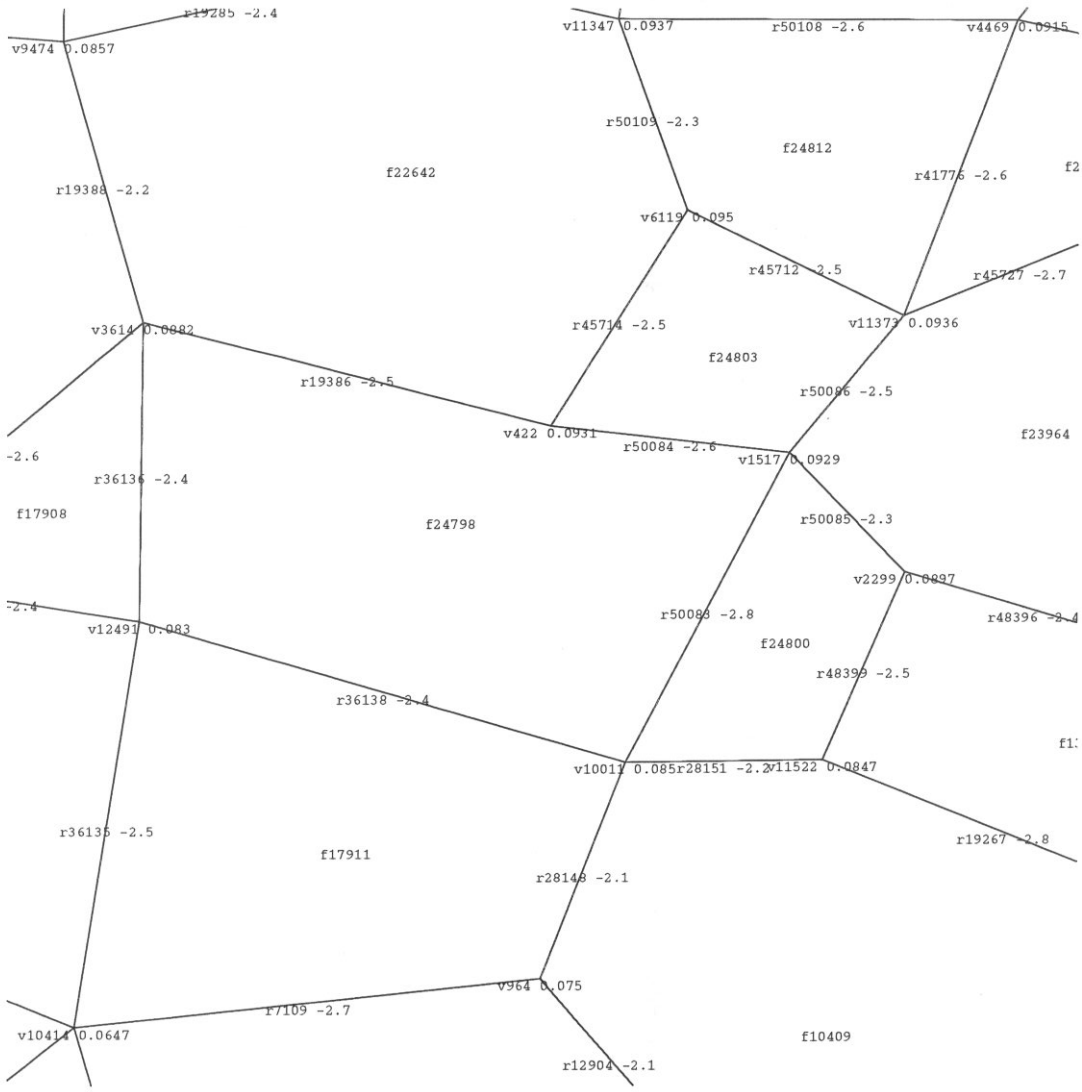


Figure 7.16: Testing acute angles against a known inside point prevents the previous error. The additional test changes the hull. The final result is displayed. Vertex 2299 is center left. Vertex 6119 is upper center. Point 6472 is coplanar and not shown.

Chapter 8

Discussion

Traditionally, computational geometry has assumed real numbers and precise arithmetic. If an implementation uses floating point arithmetic, it may fail when singularities occur or almost occur. This thesis introduces the box complex, a new domain for computational geometry that subsumes roundoff error, measurement error, representation error, and singularities. The primary feature of a box complex is the box associated to each face. A box delimits the possible locations of a face. Box complexes represent polyhedra, subdivisions of a manifold, open covers of a manifold, and objects that self-intersect or overlap.

We analyzed two algorithms for box complexes: Point_in_polyhedron and Quick_hull. The Point_in_polyhedron algorithm takes a box complex as input and produces a *clearly inside*, *clearly outside*, or *can't tell* output. We used homology theory to prove its correctness. The Quick_hull algorithm takes a set of imprecise points as input and produces a convex box complex as output. The trace of a convex box complex contains the boundaries of all possible exact convex hulls of the input.

We proved the correctness of Quick_hull, analyzed its time complexity, and analyzed the maximum facet width at the end of Quick_hull. Given balance conditions that are normally achieved, Quick_hull runs in average time $O(nlogh + h)$ where n is the size of the input and h is the size of the output. The algorithm works in general dimension and has been implemented in two and three dimensions. Empirically, the maximum facet width is a small multiple times the maximum width due to a single merge. We derived an upper bound for the maximum facet width in

2-d, and under tested restrictions, derived the corresponding bound in 3-d.

Often the exact geometry of an object is neither available, computable, nor desirable. Availability and computability is limited by measurement and roundoff error. Desirability is limited by the time and space required for building and storing a representation. This is especially true in four and higher dimensions since the maximum size of a polytope increases exponentially with dimension. Often the exact structure is not needed. A box complex can represent the level of detail required of the application.

Another advantage of box complexes is strong predicates that are satisfied despite perturbations of the data. For example, a Quick_hull user can guarantee a strongly convex box complex by specifying the maximum angle between adjacent facets. If a strong predicate returns negative results, an algorithm may report *can't tell* or delay topological decisions by storing information. Point_in_polyhedron delayed topological decisions by partitioning a surface into a front half and a back half. Quick_hull delayed topological decisions by merging facets.

Much work remains to be done on box complexes. While homology was a useful tool for Point_in_polyhedron, little is known about the geometric structure of locally convex box complexes. The lack of knowledge showed up in the twist parameter ζ. The principle study of the metric structure of polytopes is by the Russian mathematician Aleksandrov [1950]. his work is available in Russian and in German translation. Another Russian mathematician, Chernykh [1988], designed an incremental algorithm for constructing convex hulls. Like locally convex box complexes, it sandwiches exact convex hulls between inner and outer planes. The algorithm looks incomplete (no centrums, no facet merging), but the description is too abbreviated to refute. He does not bound the facet width.

For Quick_hull, additional reductions in maximum facet width are desirable. We can classify bad merges and identify remedies. As illustrated by the statistics in the previous chapter, several Quick_hull steps cause comparable maximum facet widths. This indicates that wider facets may be intrinsic to Quick_hull. An alternative remedy is iterative improvement: after completing a convex box complex, Quick_hull can remove the widest facets and reprocess their vertices.

The most important applications for Quick_hull may be in 4-d and higher dimensions. In particular, Voronoi diagrams, Delaunay triangulations and power diagrams are computable as convex hulls in one higher dimension [Brown 1979; Aurenhammer 1991]. While Quick_hull was designed for $\mathbf{R}^d$, a number of issues must be resolved:

- Quick_hull merges coplanar facets. After merging, the geometric intersection of hyperplanes no longer corresponds to vertices. How is a facet of a convex box complex turned into a Voronoi region?

- Quick_hull merges redundant ridges. In 4-d and higher, should a ridge be an arbitrary box complex, or should fully redundant ridges be the only ones to merge?

- An important part of *merge cone* was redefining the hyperplane. In 4-d and higher, several choices exist. Which choice is best?

- Visualization of Quick_hull's intermediate steps was crucial to the development of the algorithm. What is the equivalent in 4-d and higher? Will 3-d projections be required or can 2-d projections be effective?

A related issue is adjusting Quick_hull for applications. For example, Dobkin and Kirkpatrick [1983] have a fast intersection algorithm based on a hierarchical decomposition of a convex polytope. Quick_hull could produce a decomposition in which each layer meets increasingly strong convexity constraints. At the outer layer, it produces facets that include coplanar points. At this point, high precision arithmetic or exhaustive search could resolve ambiguous situations.

Quick_hull is preprocessor for other applications. Given a set of points, it produces an approximation to the convex hull or Voronoi diagram. Included with this approximation is a set of routines for querying the data structure. It may be useful in applications wherever a neighborhood is sufficient to answer a query.

Other variations of box complexes may be useful for computational geometry. For example, arrangements are an important structure [Edelsbrunner 1987]. A box complex for an arrangement would represent nearly incident intersections of

multiple hyperplanes. It may be best to cast the problem in dual space with hyperplanes as 0-faces and intersections as facets. This allows the hyperplanes to be exact. If done, 1-faces become important since they bound the cells of the arrangement.

Appendix A

Correctness proof of Point_in_polyhedron

This appendix gives the correctness proof for Point_in_polyhedron[1]. It reviews homology and cohomology theory, constructs a *realization* of a box complex, proves that realizations are homologous, proves a separation theorem for box complexes, and proves the reduction of Point_in_polyhedron to the parity of a subset of the vertices.

1. Review of Homology and Cohomology Theory

Homology theory and topology were originally developed in the 19th century by Poincaré and others. They studied path integrals and the effect of the path on the result. For example, in an open, convex, connected 2-d set, a vector field has a potential function if its partial derivatives are equal. The potential of a point is a path integral over the vector field. A necessary condition is that the integral is independent of the path taken. In particular, the integral is 0 for all paths starting and ending at a point. This may not be true if the domain is missing a point. Then the integral of a circle around the missing point can be 2π. In this case, there is an infinite set of equivalence classes of cyclic paths: those i times around the missing

[1]Joint work with Michael Hirsch.

point. If $i = 0$, the path is the boundary of an open disk [see Massey 1980, Ch. 1 for more examples].

The one-dimensional homology is the study of equivalence classes of paths. A path is zero in the homology if it is a boundary. Two paths are *homologous* if their difference is a boundary. Consider a box complex in $\mathbf{R}^2$. Because a box complex satisfies the evenness condition, a realization of the complex is a cyclic path. We want to prove that the parity test is independent of the realization and valid. To do this we need to show that all realizations are homologous and that all test rays are homologous.

Homology theory is part of algebraic topology. It reduces certain aspects of topological space to an algebra. In this section we will state some definitions and quote some theorems from algebraic topology. We do this to establish the version of the theorems. A good reference for this material is [Munkres 1984].

DEFINITION A.1. *A simplicial d-chain is a formal linear combination of d-dimensional simplices with coefficients in* $\mathbf{Z}_2$.

$\mathbf{Z}_2$ is the ring of coefficients consisting of 0 and 1. Other rings of coefficients are allowed, but we won't need to discuss them. Since coefficients are only $\mathbf{Z}_2$, one may (somewhat incorrectly) think of $\Delta_1 + \Delta_2 + \ldots + \Delta_k$ as being the union of the simplices, $\Delta_1 \cup \Delta_2 \cup \ldots \cup \Delta_k$.

DEFINITION A.2. *The* boundary $\partial\Delta$ *of a d-simplex* Δ *is the sum of the boundary* $(d-1)$*-simplices of* Δ. *The* boundary of a simplicial chain *is the sum of the boundaries of each simplex in the chain. A chain is* a cycle *if its boundary is 0.*

A simplicial d-chain is an abstraction of a collection of simplices. It may not correspond with their geometry. For example, $\Delta + \Delta = 0$ because $\mathbf{Z}_2$ coefficients are added mod 2, but $\Delta \cup \Delta = \Delta$.

However for our purposes, the algebra does reflect the geometry. Consider two simplices $\Delta_1, \Delta_2 \subset \mathbf{R}^n$ situated so that they share a boundary simplex. Then it is reasonable to say that the shared simplex should not appear in the boundary chain $\partial(\Delta_1 + \Delta_2) = \partial\Delta_1 + \partial\Delta_2$. This agrees exactly with the algebra.

Homology Theory is the study of equivalence classes of cycles. Two d-cycles C_1, C_2 are equivalent if there is a $(d+1)$-chain D such that $\partial D = C_1 + C_2$. The ith homology group of a space X is written $H_i(X)$.

More formally, let X be a topological space, i.e., a collection of sets that is closed under union and finite intersection. Let $C_n(X)$ be the set of n-chains of elements of X. An *n-chain* is the formal linear combination of n-dimensional elements of X. Define $Z_n(X) = \text{kernel } \partial_n = \{c \in C_n(X) : \partial(c) = 0\}$. The set $Z_n(X)$ is the set of n-cycles. Define $B_n(X) = \text{image } \partial_{n+1} = \partial_{n+1}(C_{n+1}(X))$. The set $B_n(X)$ is the set of n-chains that are the boundary of an $(n+1)$-chain. Then H_n is the quotient space: $Z_n(X)/B_n(X)$. This means that the elements of H_n are represented by cycles and that two cycles are representations of the same equivalence class of H_n if their difference is a boundary.

THEOREM A.3. $\partial_n \partial_{n+1} = 0$.

By Theorem A.3, a boundary is a cycle. For example, consider a 2-chain consisting of a single 2-simplex. It is a triangle made of three edges and each edge is between two vertices. The boundary of the 2-chain is the three edges, i.e., a 1-chain of three 1-simplices. The boundary of the 1-chain is zero because each vertex occurs twice (the evenness condition).

COROLLARY A.4. $B_n(X) \subset Z_n(X)$

THEOREM A.5. *Let $B \subset \mathbf{R}^d$ be an open contractible set. Suppose C is a simplicial n-chain in B, $n < d$. If $\partial C = 0 \pmod 2$ then there is a $(n+1)$-chain D in B such that $C = \partial D$.*

An example of Theorem A.5is an $(n+1)$-dimensional polyhedron D. Its boundary is a list of facets, an n-chain C. The boundary of its boundary is 0 because every $n-1$-dimensional edge of the polyhedron belongs to two facets (both elements of C). The theorem states that there is an $(n+1)$-chain D whose boundary is C. The polyhedron is D.

Cohomology is the dual concept to homology. Cohomology Theory is the study of functions on simplicial chains to the ring of coefficients. The class of all i-chains

with $\mathbf{Z}/\mathbf{Z}_2$ coefficients is $C_i^\star = \hom(C_i, \mathbf{Z}_2)$, i.e., formal linear combinations of elements of C_i. A *co-chain* $\gamma \in C^i$ is a member of the dual of $C_i^\star$. So a co-chain is a functional from chains to $\mathbf{Z}_2$. If $\{\Delta_i\}$ is a basis for C_i, then $\{\gamma_i(\Delta_j) = \delta_{ij}\}$ is a basis for the cohomology group C^i (δ_{ij} is the Kronecker delta). This defines an isomorphism between C^0 and C_0.

Definitions of co-boundary, co-cycles, and cohomology groups are the same as their dual versions under codimensions. For example, the coboundary operator γ is the dual of the boundary operator $\delta : C_{p+1} \mapsto C_p$. Thus $\gamma : C^p \mapsto C^{p+1}$. We use the reduced cohomology group $\widetilde{H}^i(X)$. It has the property:

$$H^i(X) = \begin{cases} \widetilde{H}^i(X), & i > 0 \\ \widetilde{H}^i(X) + \mathbf{Z}_2, & i = 0. \end{cases}$$

THEOREM A.6. *Let X be an open subset of $\mathbf{R}^d$. Then X has k arc-connected components if and only if $\widetilde{H}^0(X) = \mathbf{Z}_2^{k-1}$.*

The notation $\mathbf{Z}_2^k$ means k copies of the natural numbers Z mod 2. For example if X has 1 arc-connected component, then $\widetilde{H}^0(X) = \mathbf{Z}_2^0 \iff H^0(X) = \mathbf{Z}_2 \iff H_0(X) = \mathbf{Z}_2$. So each class of $H_0(X)$ is the chains of n copies of any point. A point x_0 can be a representative for a class of $H_0(X)$. Any other point x_1 is homologous to x_0 because they are the boundary of the arc between them. So $B_0(X) = X$. Because every 0-dimensional object is a cycle, $Z_0(X)$ is all possible chains of points in X. Since all points are homologous, any chain C is equivalent to a single point whose coefficient is the sum of C's coefficients. So distinct chains in the homology have a single distinct coefficient, i.e., the 0'th homology of X is $\mathbf{Z}_2$.

For a n-dimensional manifold M^n, Poincaré proved that there is a natural isomorphism between $H^i(M)$ and $H_{n-i}(M)$. Alexander proved the following extension:

THEOREM A.7. (ALEXANDER DUALITY) *Let $K \in \mathbf{R}^d$ be a simplicial complex. Then $\widetilde{H}^k(\mathbf{R}^d - K) \cong H_{d-k-1}(K)$*

DEFINITION A.8. *Two continuous maps* $f, g : X \mapsto Y$ *are* homotopic *if there exists a continuous map* $F : I \times X \mapsto Y$ *such that* $F(0,x) = f(x)$ *and* $F(1,X)=g(x)$ *for any* $x \in X$. *The function* F *is called a* homotopy. *Furthermore, homotopic maps* $f, g : I \mapsto Y$ *are* homotopic rel endpoints *if* $f(0) = g(0)$, $f(1) = g(1)$, $F(t,0) = f(0)$ $\forall t \in I$ *and if* $F(t,1) = f(1)$ $\forall t \in I$.

2. Realizations of Box Complexes

To prove that Point_in_polyhedron gives the correct result, we introduce the concept of a *realization* of a box complex. The realization does not enter into the computer calculations, it is an abstract construction with which we prove the correctness of Point_in_polyhedron.

DEFINITION A.9. *A* realization *of a box complex* $\mathcal{B}$ *with boxes* B^i_j *is an* n*-dimensional simplicial complex* $K = \bigcup K^i$ *where* K^i *is a subset of the* i*-skeleton of* $\mathcal{B}$. *Then level* K^i *represents the* i*-faces of* $\mathcal{B}$ *in the following sense:*

1. $K^i = \bigcup_{B^i_j} K^i_j$ *where* $K^i_j \subset B^i_j$. *We say* K^i_j *is* associated *to* B^i_j.
2. $\partial K^i_j = \bigcup_{B^{i-1}_k \prec B^i_j} K^{i-1}_k$.

We call K^i *the* i*th* level *of* K.

Intuitively, realizations can be thought of both as things that approximate the given face complex, and things that are approximated by the box complex.

That a realization approximates a box complex is essentially obvious from the definition. By construction, K is a box complex with the same DAG as $\mathcal{B}$. $K^i_{B^i} \subset B^i$ "approximates" B^i, and the boundary relations of the $K^i_{B^i}$ are exactly the neighbor relations of B^i. If the boxes of $\mathcal{B}$ are subsets of i-flats than K and $\mathcal{B}$ are the same.

Conversely, let $\mathcal{B}$ be a box complex approximating a polyhedron P. Then the triangulation of the polyhedron itself is easily seen to be a realization of $\mathcal{B}$. In this case, the ith level of the triangulation is exactly the triangulation of the i-skeleton of P.

THEOREM A.10. *Let $\mathcal{B}$ be a box complex in $\mathbf{R}^d$ of dimension $\leq d-1$ which satisfies the evenness condition. Then there exists a realization K of $\mathcal{B}$.*

PROOF. The proof is by induction on the levels of $\mathcal{B}$ and K.

The zero level of K, K^0, is constructed by picking one point from each 0-face of $\mathcal{B}$.

To construct K^1, consider a 1-face of $\mathcal{B}$ with box B^1. By the evenness condition, B^1 must have an even number of 0-faces in it, hence there are points $x_1, x_2, \ldots, x_{2l} \in K^0 \cap B^1$ representing each such 0-face.

As there are an even number of x_i, the class $\sum_{i=1}^{2l}[x_i]$ is 0 in $H_0(\mathcal{B}; \mathbf{Z}_2)$ and belongs to $B_0(\mathcal{B}; \mathbf{Z}_2)$. Hence, the x_i bound a simplicial 1-complex K_B in B^1.

K^1 is defined to be $\bigcup_B K_B$ where the union is taken over the boxes of the 1-skeleton. K_{B^1} is then the part of K associated to B^1.

Now assume that K^i has been constructed. Given B, the box of an $(i+1)$-face B, let $B_1, , B_2, \ldots, B_l$ be the boxes subordinate to B and let K_j^i be the element of K^i associated to B_j.

Let $C = \bigcup_{j=1}^{l} K_j^i$. Then by definition of the boundary operator:

$$\partial(C) = \sum_{j=1}^{l} \partial K_j^i$$

By the evenness condition, there is an even number of paths between B and each B_j^{i-1}. So each term of ∂K_j^i occurs an even number of times in the sum. Therefore, $\partial(C) = 0 \pmod 2$.

Since B is contractible, Theorem A.5 gives us $C = \partial(D)$, where D is a simplicial $(i+1)$-complex and $D \subset B$. Let $K_B^{i+1} = D$. Doing this for each $(i+1)$-box constructs $K^{i+1} = \bigcup_B K_B^{i+1}$. ∎

As an example of a realization, consider the box complex for a 2-d pentagon (Figure A.1). A realization selects a point for each 0-face and an edge between each point. The realization could be a pentagon. The previous proof guarantees the existence of a realization. The next proof says that all realizations are homologous. It constructs a higher dimensional set whose boundary is the realizations. Consider two realizations for the pentagon. The 0-dimensional homology is all

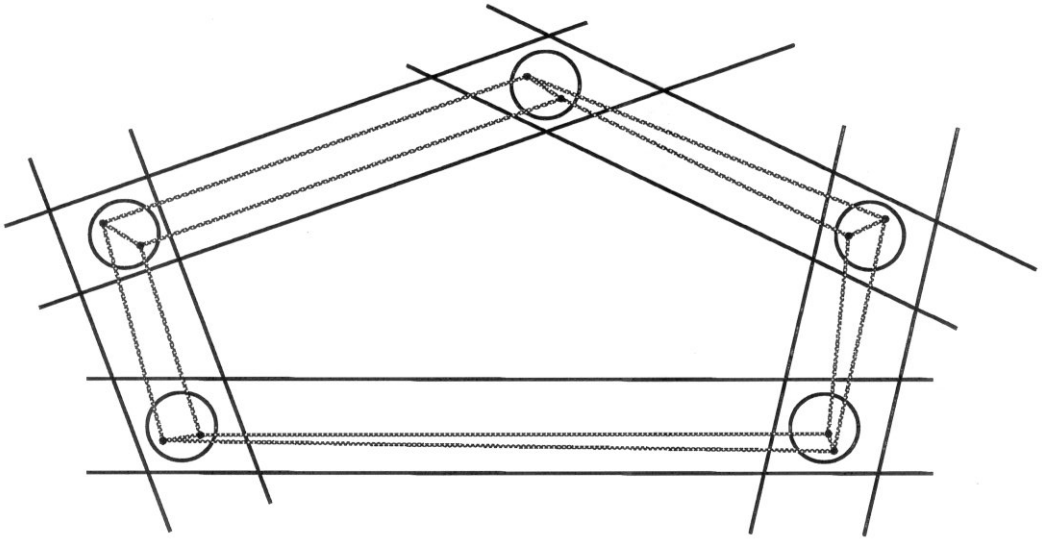


Figure A.1: Multiple realizations of a 1-box complex are homologous.

points that realize the vertices. The 1-dimensional homology is all edges between the points, and the 2-dimensional homology is all strips between the realizations. The 2-dimensional homology is made up for one rectangle for each edge. The boundary of each rectangle is the realizations for an edge and the homology for its vertices. Though the realizations for the pentagon are simple, the realizations and the homologies between them can be quite convoluted.

THEOREM A.11. *Two different realizations of the same box complex $\mathcal{B}$ are homologous within the trace of $\mathcal{B}$.*

PROOF. To prove two realizations are homologous, we need to prove that they are the boundary of a higher dimensional object. As in the proof of existence, we construct this homology inductively on the levels of K.

Let K and L be two realizations of $\mathcal{B}$. We inductively construct a homology M between K and L.

K^0 and L^0 differ in that they may be different points in the 0-boxes of $\mathcal{B}$. For a given 0-box B, let $M_B^0 = K_B^0 \cup L_B^0$ and let M_B^1 be the straight line homology between the pair of points in B. Let $M^1 = \bigcup M_B^1$ where the union is over all 0-boxes. Note that $\partial M_B^1 = \bigcup_{B_j \prec B} M_{B_j}^0 \cup K_B^0 \cup L_B^0 = K_B^0 \cup L_B^0$ and $\partial M^1 = K^0 \cup L^0$. Thus M^1 is

a homology between K^0 and L^0. The homology M^1 contains the same number of line segments as B has 0-faces.

Inductively, suppose we have produced M^{i+1}, a homology between K^i and L^i. M^{i+1} is $\bigcup M_B^{i+1}$ where the union is over all i-boxes and M_B^{i+1} is a homology within B between K_B^i and L_B^i. By definition, $\partial M_B^{i+1} = \bigcup_{B_j \prec B} M_{B_j}^i \cup K_B^i \cup L_B^i$. By the evenness condition, $\partial M^{i+1} = K^i \cup L^i$. We now construct M^{i+2} by constructing each M_B^{i+2}.

Fix B, a $(i+1)$-box of $\mathcal{B}$. For each B_j subordinate to B, we have a homology $M_{B_j}^{i+1}$ between $K_{B_j}^i$ and $L_{B_j}^i$ so that

$$\begin{aligned}
\partial((\bigcup_{B_j} M_{B_j}^{i+1}) \cup K_B^{i+1} \cup L_B^{i+1}) &= \bigcup_{B_j} (\partial M_{B_j}^{i+1}) \cup \partial K_B^{i+1} \cup \partial L_B^{i+1} \\
&= \bigcup_{B_j} (\bigcup_{B_k \prec B_j} M_{B_k}^i \cup K_{B_j}^i \cup L_{B_j}^i) \cup \bigcup_{B_j} K_{B_j}^i \cup \bigcup_{B_j} L_{B_j}^i \\
&= \bigcup_{B_j} (K_{B_j}^i \cup L_{B_j}^i) \cup \bigcup_{B_j} K_{B_j}^i \cup \bigcup_{B_j} L_{B_j}^i \\
&= 0 \pmod 2.
\end{aligned}$$

Therefore there is a complex M_B^{i+2} in B such that

$$\partial M_B^{i+2} = (\bigcup_{B_j} M_{B_j}^{i+1}) \cup K_B^{i+1} \cup L_B^{i+1}.$$

∎

3. Separation Theorems

THEOREM A.12. *Let K be a realization of a d-box complex $\mathcal{B}$. Then $H_d(K; \mathbf{Z}_2) = \mathbf{Z}_2^k$ for some $k > 0$.*

PROOF.

Let $\sigma = \sum_{\Delta^d \in K} \Delta^d$, i.e., σ is a d-chain that is the sum of all the d-simplices in K. Every $\Delta^d \in K$ is in σ.

Then $\partial(\sigma) = \partial K = \partial \sum_{B^d} K_{B^d}^d = \sum_{B^d} \partial K_{B^d}^d = \sum_{B^d} \sum_{B^{d-1} \prec B^d} K_{B^{d-1}}^{d-1}$.

Since each B^{d-1} occurs in an even number of B^d, the coefficients are all even and hence $\partial(\sigma) = 0 \pmod 2$.

Thus σ is a d-cycle in $Z_d(K;\mathbf{Z}_2)$. Since K is a simplicial d-complex, its highest dimensional chains are d-simplices. So σ is not in $B_d(K;\mathbf{Z}_2)$ and is in $H_d(K;\mathbf{Z}_2)$. This proves the theorem. ∎

COROLLARY A.13. *Let K be a realization of a box complex $\mathcal{B}$. Then $\mathbf{R}^d - K$ has at least two components.*

PROOF. By Alexander duality, $\widetilde{H}^0(\mathbf{R}^d - K) = H_d(K;\mathbf{Z}_2) = \mathbf{Z}_2^k$ for some $k > 0$, hence by Theorem A.6, $\mathbf{R}^d - K$ has more than one component. ∎

REMARK A.14. *Since K is a simplicial complex, it is compact. Thus only one of the components of $\mathbf{R}^d - K$ is unbounded and the rest are bounded.*

This completes the first part of the proof of Point_in_polyhedron. The next part defines the parity test and proves that all realizations give the same result.

DEFINITION A.15. *An arc is in* general position *with respect to a realization K, if it intersects K at the interior of K's facets and if at each intersection, it passes from one component of $\mathbf{R}^d - K$ to the other.*

DEFINITION A.16. *Let K be a realization and c be an arc in general position between points not in K. The* mod 2 intersection number *of c with K is the number of points (mod 2) in $c \cap K$.*

THEOREM A.17. *Let K and L be realizations of $\mathcal{B}$, and let c be an arc in general position between points $p_1, p_2 \in \mathbf{R}^d - \text{trace}(\mathcal{B})$. Then the mod 2 intersection number of c and K is equal to the mod 2 intersection number of c and L.*

PROOF. Let M be a homology from K to L. M may be represented by a map of a "geometric complex" (i.e., a manifold except for co-dimension 2 singularities) into $\mathbf{R}^d$. We may assume the map f is smooth away from the singularities and that the curve c never meets its singularities.

$M \cap c$ is then a compact, one dimensional manifold with boundary, $\partial(M \cap c) \subset \partial M = K \cup L$. We need to show that the number of points in K is the same as the number of points (mod 2) in L.

Consider the components of $M \cap c$. Some are between two points of L or two points of K. These contribute 0 to either mod 2 intersection number. The remaining components must be between a point of L and a point of K (since the end points of c are not in M). Thus, the mod 2 intersection number of c with K is the same as the mod 2 intersection number of c with L. ∎

DEFINITION A.18. *Given a box complex $\mathcal{B}$ and a point $p \in \mathbf{R}^d - \text{trace}(\mathcal{B})$, the* crossing parity *for p is the mod two intersection number of a realization K of $\mathcal{B}$ and an arc from p to any point in the unbounded component of $\mathbf{R}^d - \text{trace}(\mathcal{B})$.*

THEOREM A.19. *Given a box complex $\mathcal{B}$ and a point $p \in \mathbf{R}^d - \text{trace}(\mathcal{B})$, the crossing parity for p is well defined.*

PROOF. Given the results of the previous theorem, all we need to show is that the mod two intersection number of an arc from p to a point in the unbounded component (not in trace($\mathcal{B}$)) is independent of the arc.

The proof is quite similar to the proof of the previous theorem. Suppose we have two arcs, c_1 and c_2, from p to y_1 and y_2, respectively. The unbounded component is connected so we can extend the arc c_2 to go from p to y_1 without changing the intersection number.

The arcs c_1 and c_2 are now homotopic to each other rel endpoints; let C be such a homotopy. As we did before, represent K as the image of a d-dimensional geometric complex M mapped to $\mathbf{R}^d$. Again, we may assume a transversality condition between C and M so that $C \cap M$ is a smooth 1-manifold with boundary points in $c_1 \cup c_2$. As before, we need concern ourselves only with the boundary points on c_1 that are connected to boundary points on c_2, and vice versa. These are in 1-to-1 correspondence so the crossing parity is the same. ∎

If a realization for a box complex divides $\mathbf{R}^d$ into two components, the crossing parity is odd for points in the interior and even for points in the exterior. If the realization divides $\mathbf{R}^d$ into more than two components, the crossing parity is even for the unbounded component, and either even or odd for all points in a bounded component. So crossing parity classifies all points in $\mathbf{R}^d - \text{trace}(\mathcal{B})$.

This completes the second part of the proof of Point_in_polyhedron.

4. Reduction of Point_in_polyhedron

A point p can now be classified as inside or outside by testing its crossing parity. We can use any realization of $\mathcal{B}$ and any suitable arc. There are two problems though:

1. Constructing a realization for a box complex takes time.
2. With fixed precision arithmetic, we can not always tell if an arc crosses a realization or guarantee that the arc is in general position.

It looks like we've come back to our starting point. We want to test if a point is inside a polyhedron. By the Jordan-Brouwer separation theorem, crossing parity classifies the point. Instead we represent the polyhedron by a box complex. We construct a realization for the box complex. Our polyhedron is one such realization. The crossing parity w.r.t. the realization classifies the point. But we already knew this.

Box complexes are useful because another solution to Point_in_polyhedron is never constructing a realization nor computing the crossing parity. Instead we use the strong predicates *not infront* and *not behind*. These predicates are independent of the realization and the precision of the arithmetic. In terms of box complexes, *not infront* means the realization for a box is clearly not in front of the test ray. The realization could be anywhere in the box, and roundoff error could be anything smaller than the maximum roundoff error.

Suppose we are given $p \in \mathbf{R}^{d+1}$, a direction $\vec{v}$ and a d-dimensional box complex $\mathcal{B}$ with the evenness property. Let H be the plane through p perpendicular to $\vec{v}$ which we may identify with $\mathbf{R}^d$, and let $\pi : \mathbf{R}^{d+1} \to H$ be orthogonal projection onto H. Let C be the contour of p's front half relative to $\vec{v}$. Recall (Theorem 2.8) that C is a $(d-1)$-face complex with the evenness property.

DEFINITION A.20. *The* projection of C to H, *which we write* πC, *is given by the same DAG as* C *with the label* B^i_j *replaced by* $\pi(B^i_j)$.

Note that in general πC is not a box complex as there is no guarantee that $\pi(B_j^i)$ is contractible. Whenever this is guaranteed, πC is clearly a $d-1$ dimensional box complex, however. By construction, we clearly will always have $p \notin \text{trace}(\pi C)$.

THEOREM A.21. *Let H be the plane perpendicular to a test ray through a point p, and C be the contour of p's front half in a box complex $\mathcal{B}$. Assume that πC is a box complex. Then crossing parity of p relative to $\mathcal{B}$ is the same as the crossing parity of p relative to the projection of C.*

PROOF. The idea of this proof is to show that πC is embedded inside $\mathcal{B}$ in such a way that the calculation of the crossing parity p relative to πC is exactly the same as the calculation of the crossing parity of p relative to $\mathcal{B}$.

Let $\mathcal{B}_+$ (resp. $\mathcal{B}_-$) be the front (resp. back) half of $\mathcal{B}$. Let K be a realization for $\mathcal{B}$. Let K_+ (resp. K_-) be the part of K representing $\mathcal{B}_+$ (resp. $\mathcal{B}_-$), and let K_0 be the part of K corresponding to C. For $X \subset \mathbf{R}^{n+1}$, let $T_t(X)$ represent the translation of X in the $\vec{v}$ direction by the amount t.

By the compactness of K, there is a positive constant $M \in \mathbf{R}$ such that $T_M(K_+)$ (resp. $T_M(K_-)$) is entirely contained in the positive (resp. negative) side of H. We construct a new box complex $\mathcal{B}'$ as follows.

The DAG for $\mathcal{B}'$ is that same as the DAG for $\mathcal{B}$. Given a node B_j^i of $\mathcal{B}$ we must construct the corresponding box of $\mathcal{B}'$. There are three cases.

1. If B_j^i is in C, then
$$B_j^{i'} = \bigcup_{t \in [-M,M]} T_t(B_j^i).$$
Intuitively, the contour is stretched out from well in front of H to well behind H. Call the induced $(d-1)$-box complex C'.

2. If B_j^i is in $\mathcal{B}_+$, then
$$B_j^{i'} = T_M(B_j^i) \cup \bigcup_{\substack{B_l^k \in C \\ B_l^k \prec B_j^i}} B_l^{k'}.$$

The front half is just translated to well in front of H, except the parts overlapping the contour are stretched like the contour. Call the induced box complex $\mathcal{B}'_+$.

3. Similarly, if B^i_j is in $\mathcal{B}_-$, then

$$B^{i\prime}_j = T_{-M}(B^i_j) \cup \bigcup_{\substack{B^k_l \in C \\ B^k_l \prec B^i_j}} B^{k\prime}_l.$$

The back half is similarly translated to well in back of H, except the parts overlapping the contour are stretched like the contour. Call the induced box complex $\mathcal{B}'_-$.

Since $\vec{v}$ is perpendicular to H, each box in $C' \cap H$ is contained inside the corresponding face of πC, thus any realization of $C' \cap H$ is also a realization of πC.

Now we construct K', a realization of $\mathcal{B}'$. If B^i_j is in $\mathcal{B}_-$ or $\mathcal{B}_0$ then let $K'_{B^i_j} = T_{-M}(K_{B^i_j})$ and we say $K'_- = T_{-M}(K_-)$ and $K'_0 = T_{-M}(K_0)$. These are clearly realizations of the corresponding box complex.

If B^i_j is in $\mathcal{B}_+$ then let

$$K'_{B^i_j} = T_M(K_{B^i_j}) \cup \bigcup_{\substack{B^k_l \in C \\ B^k_l \prec B^i_j}} K'_{B^k_l}.$$

In this case we say

$$K'_+ = T_M(K_+) \cup \bigcup_{t\in[-M,M]} T_t(K_0).$$

In the first two cases $K'_{B^i_j}$ is clearly a simplicial complex in $B^{i\prime}_j$, but when B^i_j is in $\mathcal{B}_+$, $K'_{B^i_j}$ is not quite a simplicial complex. It is rather the union of a simplicial complex with the piece-wise linear image of the product of another simplicial complex and a closed interval. After a suitable subdivision, however, we may assume this image is a simplicial complex.

The following lemmata are immediate from the definitions.

LEMMA A.22. *The simplicial complexes K'_+, K'_0, and K'_- are valid realizations of $\mathcal{B}'_+$, the stretched contour C', and $\mathcal{B}'_-$. The union $K'_+ \cup K_0 \cup K'_-$ is a realization of $\mathcal{B}'$.*

LEMMA A.23. *Let $K'' = (K'_+ \cup K_0 \cup K'_-) \cap H$. Then K'' is a realization of C'. Note that $K'' = K'_+ \cap H$.*

We are now ready to prove the theorem. The crossing parity for p relative to $\mathcal{B}$ can be computed using the ray in the $\vec{v}$ direction. In the $\vec{v}$ direction, $\mathcal{B}'$ differs from $\mathcal{B}$ only in that it has been translated further from p. So the crossing parity of $\mathcal{B}'$ in the $\vec{v}$ direction is the same as the crossing parity of $\mathcal{B}$. Therefore the crossing parity for p relative to $\mathcal{B}$ is the same as the crossing parity for p relative to $\mathcal{B}'$.

Now consider the crossing parity for p relative to $\mathcal{B}'$. It is independent of the path with which we compute it (Theorem A.19) so pick a path from p to ∞ within H which meets K' transversely and count its mod 2 crossing number. Since $K'' = K' \cap H$ and the path meets K' transversely in $\mathbf{R}^{d+1}$, the path and K'' must be transverse in H, and thus this path gives the same mod 2 crossing number for K' in $\mathbf{R}^{d+1}$ and for K'' in H. Then by Lemma A.23 the crossing parity for p relative to $\mathcal{B}'$ is the same as the crossing parity for p relative to C inside H.

Combining the results of these two paragraphs we get that the crossing parity for p relative to $\mathcal{B}$ is the same as the crossing parity for p relative to C inside H, which proves Theorem A.21. ∎

Point_in_polyhedron is an implementation of this theorem with Theorem 3.3 ending the recursion. This completes its proof of correctness:

THEOREM A.24. *Let $\mathcal{B}$ be a box complex and p be a point in $\mathbf{R}^d - \text{trace}(\mathcal{B})$. If the boxes of projected contours are convex, Point_in_polyhedron computes the crossing parity of p.*

Bibliography

Aleksandrov, A.D. 1950. *Convex polyhedra* (in Russian), Moscow. German translation *Die innere Geometrie der konvexen Flachen*, Berlin 1955.

Aurenhammer, F. 1991. Voronoi diagrams – A survey of a fundamental geometric data structure. *ACM Computing Surveys 23*, 345-405.

Baumgart, B.G. 1975. A polyhedron representation for computer vision. *1975 National Computer Conference 44*, 589-596, AFIPS Press.

Beichl & Sullivan [1992],

Bentley, J.L., and Faust, M.G. 1982. Approximation algorithms for convex hull. *Communications of the ACM 25*, 64-68.

Berger, M. 1990. Convexity. *The American Mathematical Monthly 97*, 650-678.

Boissonnat, J.-D., and Devillers-Teillaud, M. 1989. On the randomized construction of the Delaunay tree. Report 1140, Institut National de Recherche en Informatique et en Automatique, Le Chesnay Cedex, France. To appear in *Theoretical Computer Science* .

Bowyer, A., and Woodwark, J. 1983. *A programmer's geometry*, London: Butler and Tanner.

Brondsted, A. 1983. *An introduction to convex polytopes*, Berlin: Springer-Verlag.

Bykat, A. 1978. Convex hull of a finite set of points in two dimensions. *Information Processing Letters 7*, 296-298.

Cai, J.-Y. 1992. Personal communication. Princeton University.

Char, B.W., Geddes, K.O., Gonnet, G.H., Monagan, M.B., Watt, S.M. 1988. *MAPLE Reference Manual*, Waterloo, Ontario: University of Waterloo, Dept. of Computer Science.

Chand, D.R., and Kapur, S.S. 1970. An algorithm for convex polytopes. *Journal of the ACM 17*, 78-86.

Chazelle, B. 1991. An optimal convex hull algorithm and new results on cuttings. *Proc. 32nd Annual Symp. on Foundations of Computer Science*, 29-39, IEEE Computer Society.

Chazelle, B. and Matoušek, J. 1991. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Computational Geometry: Theory and Applications.*

Chernykh, O.L. 1988. Construction of the convex hull of a finite set of points when the computations are approximate. *U.S.S.R. Comput. Maths. Math. Phys. 28*, 71-77.

Clarkson, K.L., Mehlhorn, K., Seidel, R. 1992. Four results on randomized incremental constructions, *Symposium on Theoretical Aspects of Computer Science*, to appear.

Corkum, B.T., and Wyllie, J.A. 1990. Program for Insidepolytope. Dept. of Civil Engineering, University of Toronto, News group comp.graphics message 1990Oct3.-121706.11932@jarvis.csri.toronto.edu.

Dobkin, D.P., and Kirkpatrick, D.G. 1983. Fast detection of polyhedral intersection. *Theoretical Computer Science 27* 241-53.

Dobkin, D., and Koutsofios, E. 1988. Cheyenne-device independent graphics package. manual page /u/graphics/man/cheyenne.3@princeton.edu, Princeton University, Dept. of Computer Science.

Dobkin, D.P., and Laszlo, M.J. 1989. Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica 4*, 3-32.

Dobkin, D.P., and Silver, D. 1988. Recipes for geometry and numerical analysis, Part I: An empirical study. *Proceedings of the Symposium on Computational Geometry*, ACM, 93-105.

Dobkin, D.P., and Silver, D. 1990. Applied computational geometry: Towards robust solutions of basic problems. *Journal of Computer and System Sciences*, 40.1:70-87.

Dorward, S. 1992. Personal communication, Princeton University.

Eddy, W. 1977. A new convex hull algorithm for planar sets. *ACM Transactions on Mathematical Software*, 3.4:398-403.

Edelsbrunner, H. 1987. *Algorithms in Combinatorial Geometry*, Berlin: Springer-Verlag.

Edelsbrunner, H., and Mücke, E.P. 1988. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *Proceedings of the Symposium on Computational Geometry*, ACM, 118-133.

Edelsbrunner, H., and Shi, W. 1989. An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem, Report UIUCDCS-R-89-1533, Department of Computer Science, Univ. Illinois at Urbana-Champaign.

Emiris, I., Canny, J. 1991. A general approach to removing degeneracies. *Proc. 32nd Annual Symposium on Foundations of Computer Science*, 405-413.

Floyd, R.W. 1976. Private communication to Preparata & Shamos on Quickhull.

Fortune, S. 1989. Stable maintenance of point-set triangulation in two dimensions. *30th Annual Symposium on the Foundations of Computer Science*, IEEE.

Fortune, S. 1992a. Computational geometry. Martin, R. (ed.) *Directions in Geometric Computing*, Information Geometers, in preparation.

Fortune, S. 1992b. Personal communication, Murray Hill: Bell Laboratories.

Fortune, S., and Milenkovic V. 1991. Numerical stability of algorithms for line arrangements. *Proc. Seventh Annual Symposium on Computational Geometry*, 334-341.

Goldberg, D. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23.1:5-48.

Golin, M., and Sedgewick, R. 1988. Analysis of a simple yet efficient convex hull algorithm. *Proc. of the 4th Annual Symp. on Computational Geometry*, 153-163.

Golub and van Loan 1983. *Matrix Computations*, Baltimore: Johns Hopkins University Press.

Graham, R. 1972. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters 1*, 132-133.

Greene, D.H., and Yao, F.F. 1986. Finite-resolution computational geometry. *Symposium on Foundations of Computer Science*, IEEE, p. 143-152.

Green, P.J., and Silverman, B.W. 1979. Constructing the convex hull of a set of points in the plane. *Computer Journal* 22:262-266.

Grünbaum, B. 1961. Measures of symmetry for convex sets. *Proceedings of the Seventh Symposium in Pure Mathematics of the American Mathematical Society, Symposium on Convexity* 233-270.

Grünbaum, B. 1967. *Convex Polytopes*, London: Interscience Publications.

Guibas, L.J., Knuth, D.E., and Sharir, M. 1990. Randomized incremental construction of Delaunay and Voronoi diagrams. Paterson, M.S. (Ed.), *Automata, Languages and Programming*, Lecture Notes in Computer Science, Berlin: Springer-Verlag, 443:414-431.

Guibas, L., Salesin, D., and Stolfi, J. 1989. Epsilon geometry: Building robust algorithms from imprecise computations. *Proceedings of the Symposium on Computational Geometry*, ACM, 208-217.

Guibas, L., Salesin, D., and Stolfi, J. 1990. Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, 1992 (in press). An earlier version of the paper appeared in the *Proceedings of the International Symposium on Algorithms (SIGAL '90)* (Tokyo, August, 1990) 261-270.

Guibas, L., and Stolfi, J. 1985. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. on Graphics 4*, 74-123.

Hanrahan, P. 1989. A survey of ray-surface intersection algorithms. Glassner, A. (ed.), *An introduction to ray tracing*, London: Academic Press.

Hoffmann, C. 1989a. *Geometric and Solid Modeling*, San Mateo CA: Morgan Kaufmann.

Hoffmann, C. 1989b. The problems of accuracy and robustness in geometric computation. *Computer 22*, 31-42.

Hoffman, C.M., Hopcroft, J.E., and Karasick, M.S. 1988. Towards implementing robust geometric computations. *Proceedings of the Symposium on Computational Geometry*, ACM, p. 106-117.

Hopcroft, J.E., and Kahn, P.J. 1989. A paradigm for robust geometric algorithms, TR 89-1044, Dept of Computer Science, Cornell University.

Horn, W.P. and Taylor, D.L. 1989. A Theorem to determine the spatial containment of a point in planar polyhedron. *Computer Vision, Graphics, and Image Processing 45*, 106-116.

Iri, M. 1984. Simultaneous computation of functions, partial derivatives and estimates of rounding errors. *Japan J Applied Mathematics 1*, 223-252.

Kahan, W. 1972. A survey of error analysis. *Information Processing 71*, 2:1215-1239.

Kalay, Y.E. 1982. Determining the spatial containment of a point in general polyhedra. *Computer Graphics and Image Processing 19* , 303-334.

Kallay, M. 1981. Convex hull algorithms in higher dimensions. Unpublished manuscript, Dept. Mathematics, Univ. of Oklahoma, Norman, Oklahoma. See also [Preparata & Shamos 85, p. 137-140].

Kao, T.C., and Knott, G.D. 1990. An efficient and numerically correct algorithm for 2D convex hull problem. *BIT 30*, 289-300.

Karasick, M., Lieber, D., Nackman, L. 1990. Efficient Delaunay triangulation using rational arithmetic. *ACM Transactions on Graphics 10*, 71-91.

Kirkpatrick, D.G., and Seidel, R. 1986. The ultimate planar convex hull algorithm? *SIAM J Computing*, 12:287-299.

Klee V. 1966. Convex polytopes and linear programming. *Proc. IBM Sci. Comput. Symp.: Combinatorial Problems*, 123-158, White Plains NY: IBM.

Kulisch, U., and Miranker, W.L. 1983. *A New Approach to Scientific Computing*, New York: Academic Press.

Lane, J., Magedson, B. and Rarick, M. 1984. An efficient point in polyhedron algorithm. *Computer Vision, Graphics, and Image Processing 26*, 118-125.

Li, Z., and Milenkovic, V. 1990. Constructing strongly convex hulls using exact or rounded arithmetic. *Proceedings of the Symposium on Computational Geometry*, ACM, 197-207.

Mudur, S.P., and Koparkar, P.A. 1984. Interval methods for processing geometric objects *IEEE Computer Graphics and Applications 2/84*, 7-17.

Massey, W.S. 1980. *Singular Homology Theory*, New York: Springer-Verlag 1980.

McMullen, P., and Shephard, G.C. 1971. *Convex Polytopes and the Upper Bound Conjecture*, Cambridge, England: Cambridge University Press.

Milenkovic, V.J. 1988. Verifiable implementation of geometric algorithms using finite precision arithmetic. *Artificial Intelligence 37*, 377-401.

Milenkovic, V. 1989a. Calculating approximate curve arrangements using rounded arithmetic. *Proceedings of the Symposium on Computational Geometry*, ACM pp. 197-207.

Milenkovic, V. 1989b. Double precision geometry: A general technique for calculating line and segment intersections using rounded arithmetic. *30th Annual Symposium on the Foundations of Computer Science*, IEEE, pp. 500-505.

Miller, W. 1975. Computer search for numerical instability. *J ACM 22*, 512-521.

Moore, R.E. 1979. *Methods and Applications of Interval Analysis*, SIAM.

Munkres, J.R. ,1984. *Elements of Algebraic Topology*, Menlo Park, California: Addison-Wesley.

Nordbeck, S., and Rystedt, B. 1967. Computer cartography point-in-polygon programs. *BIT 7*, 39-64.

Ottmann, Th., Thiemt, G., and Ullrich, Ch. 1988. On arithmetical problems of geometric algorithms in the plane. *Computing*, Supplement 6, 123-136.

Preparata, F.P., and Hong, S.J. 1977. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20:87-93.

Preparata, F.P., and Shamos, M.I. 1985. *Computational Geometry. An Introduction,* Berlin: Springer-Verlag.

Segal, M 1990. Using tolerances to guarantee valid polyhedral modeling results. *SIGGRAPH '90*, ACM.

Segal, M., and Sequin C.H. 1988. Partitioning polyhedral objects into non-intersecting parts. *IEEE Computer Graphics & Applications 8*, 53-67.

Milenkovic, V.J. 1988. Verifiable implementation of geometric algorithms using finite precision arithmetic. *Artificial Intelligence 37*, 377-401.

Milenkovic, V. 1989a. Calculating approximate curve arrangements using rounded arithmetic. *Proceedings of the Symposium on Computational Geometry*, ACM pp. 197-207.

Milenkovic, V. 1989b. Double precision geometry: A general technique for calculating line and segment intersections using rounded arithmetic. *30th Annual Symposium on the Foundations of Computer Science*, IEEE, pp. 500-505.

Miller, W. 1975. Computer search for numerical instability. *J ACM 22*, 512-521.

Moore, R.E. 1979. *Methods and Applications of Interval Analysis*, SIAM.

Munkres, J.R. ,1984. *Elements of Algebraic Topology*, Menlo Park, California: Addison-Wesley.

Nordbeck, S., and Rystedt, B. 1967. Computer cartography point-in-polygon programs. *BIT 7*, 39-64.

Ottmann, Th., Thiemt, G., and Ullrich, Ch. 1988. On arithmetical problems of geometric algorithms in the plane. *Computing*, Supplement 6, 123-136.

Preparata, F.P., and Hong, S.J. 1977. Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM*, 20:87-93.

Preparata, F.P., and Shamos, M.I. 1985. *Computational Geometry. An Introduction,* Berlin: Springer-Verlag.

Segal, M 1990. Using tolerances to guarantee valid polyhedral modeling results. *SIGGRAPH '90*, ACM.

Segal, M., and Sequin C.H. 1988. Partitioning polyhedral objects into non-intersecting parts. *IEEE Computer Graphics & Applications 8*, 53-67.

Sugihara, K. 1989. On finite-precision representations of geometric objects. *Journal of Computer and System Sciences 39*, 236-247.

Sugihara, K., and Iri, M. 1989. Construction of the Voronoi diagram for one million generators in single-precision arithmetic. *First Canadian Conference on Computational Geometry* .

Sutherland I.E., Sproull, R.F., and Schumacker, R.A. 1974. A characterization of ten hidden-surface algorithms. *Computing Surveys 6*, 1-55.

Taylor, J.R. 1982. *An Introduction to Error Analysis*, Mill Valley, CA: University Science Books.

Tilove, R.B. 1980. Set membership classification: A unified approach to geometric intersection problems. *IEEE Trans. on Computers C-29*, 874-883.

Vignes, J. 1988. Review of stochastic approach to round-off error analysis and its application. *Mathematics and Computers in Simulation 30*, 481-491.

Wilkinson, J.H. 1965. *The Algebraic Eigenvalue Problem*, Oxford, England: Clarendon Press.

Yap, C.-K. 1990. Symbolic treatment of geometric degeneracies. *Journal of Symbolic Computations 10*, 349-370.