Computational Geometry

Steven Fortune
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

January 4, 1994

Abstract

Computational geometry, the study of algorithms involving relatively simple geometric objects, is an active, exciting field. This chapter samples current research in computational geometry. Three topics are discussed at some length: the theory of arrangements, a random-incremental convex hull algorithm, and robustness of geometric algorithms.

1 Introduction

As pointed out by Robin Forrest and others[39, 79], the term "computational geometry" could be used for a variety of fields, including geometric modeling using curves and surfaces, computer proofs of geometric theorems, geometric design software, and the theory of perceptrons[71]. The purpose of this chapter is to give an impressionistic view of recent research in computational geometry, defined as the study of algorithms involving relatively simple geometric objects such as points and lines. A typical topic in this field is the analysis of an algorithm involving n objects in d dimensions; examples are the computation of the convex hull of n points or the computation of an arrangement of n hyperplanes. Algorithm efficiency is usually measured asymptotically as a function of n and d; often the dependence upon d is exponential, either explicitly or implicitly, so d should be viewed as a small constant.

Papers by Graham[47] and by Shamos and Hoey[87] are often cited as the beginning of computational geometry. Certainly by the first ACM Computational Geometry Symposium in 1985, the field was well defined. The field is currently thriving. There are two dedicated annual conferences, the ACM Symposium and the Canadian Conference on Computational Geometry, and three dedicated journals, *Discrete and Computational Geometry*, *Computational Geometry: Theory and Applications*, and the *International Journal of Computational Geometry and Applications*; in addition the field has a healthy representation in other theoretical conferences and journals. Two textbooks are available[30, 79] and it is likely that more will appear soon. A number of research monographs on specialized topics have appeared[1, 76, 83].

Recent research in computational geometry has become increasingly sophisticated, both in topics and in techniques. A clear trend is towards research in higher dimension. Early research was often focused on linear objects in two dimensions, for example, simple polygons and collections of line segments. Current topics are algorithms involving lines or triangles sitting in three dimensions[6, 14] or more generally hyperplanes in d dimensions[35, 37]. A less distinct trend is to extend algorithms involving linear objects to curved objects[2, 27]. Randomization has become a fundamental design tool for geometric algorithms; often the simplest or most efficient algorithm known for a problem is one that uses random coin flips to guide its execution[17]. "Derandomizing" randomized geometric algorithms has become a favorite pastime[13, 63], motivated in part by complexity-theoretic questions about the fundamental role of randomness in algorithm analysis.

A strong influence on the field has been the interplay between computational geometry and combinatorial geometry. Combinatorial geometry is concerned with combinatorial properties of geometric objects; examples are counting the number of incidences among a set of points and lines, counting the number of faces of a convex polytope, or determining the intersection properties of a family of convex sets. Combinatorial and computational geometry have become almost indistinguishable: a combinatorial bound on a family of geometric objects may be best obtained by analyzing an (abstract) algorithm that enumerates the family of objects; conversely, the design and analysis of an algorithm may depend upon precise combinatorial knowledge of the geometric objects examined by the algorithm. A good example of the interplay between algorithms and combinatorial bounds is provided by a special issue of the journal *Discrete and Computational Geometry* devoted to the complexity of arrangements[19, 32, 33].

Along with the increasing mathematical depth of the field comes an increasing abstraction from implementation concerns. A naive reader might be pleased to encounter a paper claiming an asymptotically optimal algorithm for a problem of interest. Only dismay can result, however, when the reader discovers that not only is there no discussion of special cases or implementation performance, but that the algorithm requires very complicated data structures, has never been implemented, and perhaps depends upon several other complicated also-unimplemented algorithms. However, some research has been directed at implementation issues. Examples include the development of a theory of oriented projective geometry[89], representations of spatial subdivisions in general dimension[10, 62], perturbation schemes that symbolically eliminate degeneracies[34], and careful analysis of the details of fundamental algorithms[54]. However, it takes a good deal of sophistication to pick and choose among offered algorithms, and then to know what implementation suggestions are relevant.

This chapter samples recent research in computational geometry. Rather than attempt a comprehensive survey, the chapter considers three topics in some depth. The three topics are the theory of arrangements, a randomized convex hull algorithm, and robustness, representing respectively the areas of combinatorial geometry, randomization in computational geometry, and implementation issues in computational geometry.

An *arrangement* is the partition of space induced by a collection of hyperplanes. Arrangements are of central interest in computational geometry because they are simple to define, they have a rich combinatorial structure, and they form a unifying framework for many problems. Section 2 gives an elementary discussion of arrangements and then surveys recently obtained bounds on the combinatorial complexity of arrangements. This section is the most theoretical of all the sections in this chapter. However, many algorithms in fact manipulate subsets of arrangements, perhaps in some disguised fashion; the combinatorial bounds of section 2.4 may be useful in analyzing the behavior of such algorithms.

Randomization is a fundamental algorithm design tool, in computational geometry as well as in other algorithmic disciplines. Often an efficient algorithm can be obtained by appropriately ordering the input objects or by appropriately choosing an initial subset of the objects. Sometimes it is not so easy to find the order or subset explicitly, but with high probability a random choice is satisfactory. An example is provided by quicksort, which requires an approximately median element to split the input sequence into two smaller subsequences. It is possible to find a median element in time proportional to the length of the sequence. However, it is much easier and faster to choose an element of the sequence at random, which with high probability will do a good job of splitting the input sequence.

Randomization has been used in two principal ways in computational geometry. One is randomized divide-and-conquer: a random subset of the input is chosen, the problem is solved recursively on the subset, and then the whole problem is solved using the solution to the subproblem. This approach has been used for constructing data structures for nearest neighbor searching[15], constructing trapezoidal diagrams[18], and for searching in arrangements[16]. A second method is to randomize an incremental algorithm. Incremental algorithms are often very simple: objects are inserted one by one with the desired structure updated after each addition; a classic example is the incremental computation of a Delaunay triangulation or a Voronoi diagram in two dimensions[48, 54]. An objection to incremental algorithms is that if the insertion order is chosen badly, then each addition requires a lengthy update to the structure, and the algorithm runs slowly. A remarkable discovery for some problems is that if objects are inserted in random order, then the expected work at each addition is small, and the expected running time of the algorithm is small. Notice that the expectation of the running time is taken over the permutation used to order insertions, not over some probability distribution of the input objects. With some small probability a bad insertion order might be chosen, resulting in large running time. Such random-incremental algorithms have been given for a number of problems including convex hulls[21], Delaunay triangulations and Voronoi diagrams[8, 52], spherical intersection in three dimensions[21], trapezoidal diagrams[21, 74, 75], and linear programming in fixed dimension[84]. Clarkson[17] and Seidel[85] survey randomized geometric algorithms, both incremental and divide-and-conquer.

Section 3 gives a fairly simple random-incremental algorithm that computes the convex hull of a set of points. The algorithm is a version of the standard incremental or beneath-beyond algorithm for convex hulls[30, 79], modified so that points are inserted in random order. The algorithm is described for point sets in three dimensions; in fact the algorithm generalizes easily to arbitrary fixed dimension. The running time of the algorithm depends upon the point set and the dimension. In three dimensions, the expected running time for the worst-case set of n points is $O(n \log n)$, though for some point sets the algorithm is even faster. The case of higher dimensions is discussed in section 3. The only other known $O(n \log n)$ algorithm in three dimensions is the Preparata-Hong divide-and-conquer algorithm[78]; while it is conceptually simple, its details are much more intricate than the details of the random-incremental algorithm[23, 30]. The random-incremental algorithm has indeed been implemented; empirical performance is given in section 3. The algorithm and its analysis are due to Clarkson, Mehlhorn and Seidel[20]; an earlier random-incremental convex hull algorithm is due to Clarkson and Shor[21].

Geometric algorithms base logical decisions both on combinatorial data, such as graphs, and on geometric data, such as the coordinates of points. Combinatorial data can be represented exactly, and there is no question that an algorithm could decide for example if there is an edge between two vertices in a graph. Geometric data usually has a physical origin with inherent uncertainty, and there may ultimately be no way to decide for example whether a point lies to one side or the other of the line determined by two other points. Designers of geometric algorithms avoid the problem of geometric uncertainty by assuming the real random access machine (real RAM) as the model of computation[79]. The real RAM allows real numbers to be represented exactly and provides exact arithmetic operations. Now geometric data can be assumed to be exact, and for example point-line decisions are exact. To avoid nasty cases like a point lying exactly on a line, algorithm designers often also assume "general position"; the exact meaning of general position is usually whatever is convenient for the problem at hand.

Unfortunately, an algorithm designed for the real RAM with the general position assumption cannot be reliably implemented in a naive fashion. Often floating point arithmetic is substituted for exact real arithmetic and special cases ignored. However, real-world test data will violate the general position assumption, either approximately or exactly; thus for example three input points could be approximately collinear. Implementation failure, such as an infinite loop or wildly incorrect output, can result if the approximately collinear points are treated inconsistently, i.e. as collinear in one place and not collinear in another. The implementation can be made more reliable by special-case checks, but it is difficult to be confident that all cases have been handled correctly.

Section 4 discusses research in "robustness," that is, research examining the real RAM assumption and the general position assumption in more detail. There are essentially two ways to address the real RAM assumption: one is to perform exact arithmetic, say with integers or rationals; the other is to use approximate arithmetic and then to analyze carefully every geometric test in an algorithm. The difficulty with the first is that extended-precision arithmetic is almost certainly required, possibly incurring considerable overhead; the difficulty with the second is that the required analysis is mathematically challenging, to say the least. With exact arithmetic, there are schemes that allow the general position assumption to be simulated by symbolic perturbation.

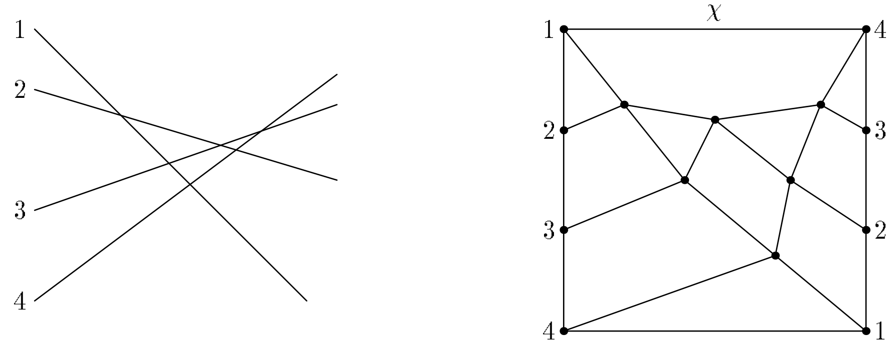
Robustness research has had some limited success for relatively simple geometric problems, for example involving polygonal objects in two dimensions and polyhedral objects in three dimensions. However, the field is still in the mode of illustrative examples, with different problems analyzed in very different ways. A general theory of robustness, applicable to a wide range of problems, is unavailable. For example, the implementation of the random-incremental convex hull algorithm described in section 3 is not completely reliable, since it uses floating point arithmetic for the primitives, though heuristics do improve reliability enough for the implementation to be a useful tool. Challenging problems, for example involving curves and surfaces in three dimensions, are largely unexplored.

2 Arrangements

Let $L = \{l_1, \ldots, l_n\}$ be a set of n lines in the plane. For each line l_i, arbitrarily label the two open halfplanes determined by l_i as positive and negative, and say that the *signature* of a point x with respect to l_i is $+1$, 0, or -1 as x is in the positive halfplane of l_i, on l_i, or in the negative halfplane of l_i. The *arrangement* of L, denoted $\mathcal{A}(L)$, is the partition of the plane into cells so that within each cell the signature of every point with respect to every line is invariant. See Figure 1(a). Cells of dimension 0 are *vertices*; they are the points of intersection of at least two lines. Cells of dimension 1 are *edges*; they are open segments of a line between two vertices (or possibly extending to infinity). Cells of dimension 2 are *regions*; they are connected open subsets of the plane not containing any point of any line. A cell f on the boundary of a cell c is a *face* of c. The faces of a region are alternately edges and vertices. In general, an *arrangement* can be defined for a set of hyperplanes in d-dimensional Euclidean space E^d; within each cell of the arrangement, the signature of every point is invariant with respect to every hyperplane.

The theory of arrangements is a central topic of computational geometry. In a fundamental paper, Edelsbrunner, O'Rourke, and Seidel[35] give an algorithm that determines the arrangement of a set of hyperplanes in arbitrary fixed dimension. Edelsbrunner's textbook[30] treats arrangements as the fundamental object of study, and shows how various geometric objects can be related to arrangements. Many known algorithms involve arrangements, for example constructing a single cell of an arrangement, traversing every cell of an arrangement, and determining the cell containing a query point.

This section has two goals. The first goal is to provide some explanation of the central importance of arrangements. Two examples illustrate the role of arrangements. The "convexity properties" of a set of points include many interesting properties. As described in section 2.2, all such convexity properties are determined by the arrangement of the lines dual



(a) (b)

Figure 1: An arrangement in two dimensions (a) and the corresponding combinatorial arrangement (b).

to the points. The Voronoi diagram of a set of sites is a fundamental tool in many applications of computational geometry; it partitions space on the basis of the closest site. As described in section 2.3, the Voronoi diagram can be obtained from an appropriate arrangement. The order-k Voronoi diagram partitions space on the basis of the k closest sites; all the order-k Voronoi diagrams can be obtained from the same arrangement. The second goal of this section is to give a brief survey of general bounds on the combinatorial complexity of sets of cells in an arrangement. The combinatorial complexity of a cell C is the total number of cells (of all dimensions) bounding C. Many geometric algorithms can be viewed as examining a subset of the cells in an arrangement; the time and space used by such an algorithm depend upon the total combinatorial complexity of the examined cells. Hence general combinatorial complexity bounds found in section 2.4 apply to a wide variety of algorithms. The survey in section 2.4 is extremely brief and covers only a few of the results that have been obtained recently; Guibas and Sharir[53] also survey recent results in arrangements.

2.1 Arrangements in two dimensions

Arrangements in two dimensions are simpler than in general dimension; to motivate later definitions and results, we consider two-dimensional arrangements first.

Let L be a set of n lines in the plane. The arrangement $\mathcal{A} = \mathcal{A}(L)$ has at most $\binom{n}{2}$ vertices, since each pair of lines determines at most one vertex. $\mathcal{A}$ has at most n^2 edges, since each line can be split into at most n pieces, and some manipulation with Euler's formula shows that $\mathcal{A}$ has at most $\binom{n+1}{2}+1$ regions. A single region of $\mathcal{A}$ can have n bounding edges; however, using Euler's formula the average number of edges per region is less than 4.

In two dimensions an arrangement can be represented as a planar graph in a purely combinatorial way. A *combinatorial line* (in a graph) is a simple path of edges. A *combinatorially embedded* planar graph is a planar graph with the order of edges incident to each

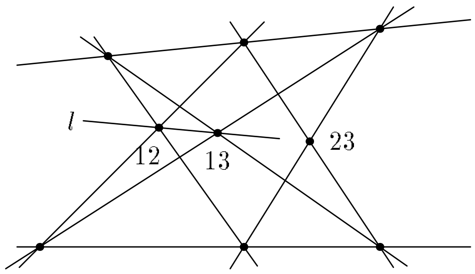


Figure 2: By Pappus's theorem, line l must pass through point 23.

vertex specified and having a plane embedding respecting the edge order. A *combinatorial arrangement* $\mathcal{C}$ is a combinatorially embedded planar graph and a partition of its edges into a simple cycle χ and a labeled set of combinatorial lines; each combinatorial line must lie inside χ with endpoints on χ and each pair of combinatorial lines must either not intersect or intersect at a unique vertex at which they cross. A combinatorial arrangement $\mathcal{C}$ can be obtained from an arrangement of lines $\mathcal{A}$ in the obvious way: the vertices of $\mathcal{C}$ are vertices of $\mathcal{A}$ together with two endpoint vertices for each line; the cycle χ connects endpoint vertices; and the edges of each combinatorial line in $\mathcal{C}$ are determined from the edges of the corresponding line in $\mathcal{A}$. Figure 1(b) gives the combinatorial arrangement corresponding to the arrangement in figure 1(a). (In higher dimensions a rather more complicated definition of a combinatorial arrangement is necessary[9].)

There is not always an arrangement $\mathcal{A}$ of lines corresponding to a combinatorial arrangement $\mathcal{C}$. The difficulty comes from theorems of projective geometry that dictate intersections among lines; in figure 2, line l intersects points 12 and 13, hence by Pappus's theorem[77] l must intersect point 23 as well. However, a combinatorial arrangement could specify that l intersect the edge above or below point 23. A *pseudoline* is an unbounded simple curve and an *arrangement of pseudolines* is a collection of pseudolines so that each pair of pseudolines intersect at most once[50]. A combinatorial arrangement $\mathcal{C}$ does determine an arrangement of pseudolines.

The following algorithm computes the arrangement of a set of lines in the plane. The output is a combinatorial arrangement, as described above. For simplicity of description, assume that no three lines intersect in a point. The algorithm inserts lines one by one in order of increasing slope. Suppose a partial arrangement $\mathcal{A}$ has been computed and that l is a new line. The algorithm discovers the sequence of edges and regions of the current arrangement that are intersected by l in order along l. The starting region is the unbounded region containing points of l with arbitrarily negative x-coordinate; since lines are inserted in order of increasing slope, the starting region is below the leftmost edge of the previously inserted line. At the general step, line l has entered some region of the arrangement. The exiting edge can be found simply by comparing l to all edges bounding the current region. The next region is just the region on the other side of the exiting edge. When line l has intersected all lines, the current region must be unbounded. As each region is encountered and each edge is crossed, some straightforward graph manipulation is needed to update the combinatorial arrangement.

How long does it take to add a new line? The *zone* of line l in arrangement $\mathcal{A}$ is the set of regions of $\mathcal{A}$ intersected by l. Clearly, the time required to add line l is proportional to the total number of *zone edges* bounding the regions in the zone of l. If $\mathcal{A}$ has m lines, then l can intersect $m+1$ regions, and each region could have m edges, so a naive bound on the number of zone edges is $O(m^2)$. However, the number of zone edges is only $O(m)$[37], as we now show. Assume line l is horizontal. A zone edge bounding a region in the zone of l is *left* or *right* as the region lies to the right or left of the edge; some edges may be both left and right zone edges. An induction on the number m of lines in $\mathcal{A}$ shows that the number of left zone edges is at most $3m$; the basis case is trivial. For the induction step, consider the arrangement $\mathcal{A}'$ obtained by removing the line l' that has rightmost intersection with l; then adding l' back in adds at most three left zone edges, one along l' and possibly one each for the lines intersected by l' immediately above and below l. Hence the total number of zone edges is linear in the number of lines in $\mathcal{A}$, the time to add a single line is linear, and the time to build an arrangement of n lines is $O(n^2)$. This is asymptotically optimal, since the arrangement has size $\Omega(n^2)$.

2.2 Convexity properties

Let S be a set of n points in the plane. How can the geometric configuration of the points of S be characterized, without actually specifying the coordinates of the points? One way is to give the orientation of every triangle formed from points of S, defined as follows. For simplicity assume that no three points of S lie on a common line.

A *triangle* Δpqr is a cyclically ordered triple of distinct points, so $\Delta pqr = \Delta qrp = \Delta rpq \neq \Delta prq$. Triangle Δpqr is *positive* (or has *positive orientation*) if r lies to the left of the line directed from p to q; equivalently, Δpqr is positive if p, q, r are in counterclockwise order. We write $P(\Delta abc)$ to mean Δabc is positive. Triangle Δpqr is *negative* if Δprq is positive. The *configuration* of a point set S is the set of positive triangles formed from the points of S.

A *convexity property* of S is a property that can be expressed by a logical formula just using the basic predicate P. Equivalently, a convexity property is determined by the configuration. Many interesting properties are convexity properties. For example, point p_1 is inside positive triangle $\Delta p_2p_3p_4$ if

$$P(\Delta p_2p_3p_4) \wedge P(\Delta p_1p_2p_3) \wedge P(\Delta p_1p_3p_4) \wedge P(\Delta p_1p_4p_2).$$

Similarly, points p_1, p_2, p_3, p_4 form the vertices of a convex quadrilateral in counterclockwise order exactly if

$$P(\Delta p_1p_2p_3) \wedge P(\Delta p_2p_3p_4) \wedge P(\Delta p_1p_3p_4) \wedge P(\Delta p_1p_2p_4);$$

this predicate is denoted $Q(p_1, p_2, p_3, p_4)$. More generally, a sequence of points $p_1, p_2, \ldots, p_l$ forms the vertices of a convex polygon in counterclockwise order if

$$\bigwedge_{i<j<k} P(\Delta p_i p_j p_k).$$

A sequence $p_1, p_2, \ldots, p_l$ forms the vertices of a simple polygon exactly if segments $p_i p_{i+1}$ and $p_j p_{j+1}$ do not intersect, for each $i \neq j$. Two segments $p_i p_{i+1}$ and $p_j p_{j+1}$ intersect exactly if either $p_i p_j p_{i+1} p_{j+1}$ or $p_i p_{j+1} p_{i+1} p_j$ form a convex quadrilateral with vertices in counterclockwise order. Hence $p_1, p_2, \ldots, p_l$ forms a simple polygon if

$$\bigwedge_{i \neq j} \neg Q(p_i, p_j, p_{i+1}, p_{j+1}) \wedge \neg Q(p_i, p_{j+1}, p_{i+1}, p_j).$$

A rather more complicated formula can express that a point is inside a simple polygon. Here is an outline of one scheme to construct such a formula. Given a set of points $p_1, \ldots, p_l$, we can construct a formula that asserts that a set of triangles is a valid triangulation of $p_1, \ldots, p_l$. The formula asserts that each triangle is positively oriented and that no two edges from distinct triangles cross. Note that given a set of triangles, the triangles inside a simple cycle of edges can be determined in a purely combinatorial way. Given a polygon S, i.e. a simple cycle of edges, the required formula for asserting that q lies inside S can be obtained by a disjunction over all possible sets of triangles of the assertion that the triangles form a triangulation, S is a subset of the edges of the triangulation, and that q is inside some triangle inside S. The number of sets of triangles is finite (though exponential) so the formula has finite size.

Naively, to specify the configuration of S, it is necessary to specify each of the $\binom{n}{3}$ positive triangles. However, a representation of the configuration of size roughly quadratic in n is possible. It is obtained by dualizing the points to lines and determining the combinatorial arrangement of the resulting lines. The duality map $\mathcal{D}$ takes the point $p = (a, b)$ to the line $l = \{(x, y) : ax + by + 1 = 0\}$, and conversely. Geometrically, the dual of point $s = (a, b)$ at distance $r = \sqrt{a^2 + b^2}$ from the origin O is the line l perpendicular to line Os, intersecting line Os at a point at distance $1/r$ from O on the opposite side of O from s; see figure 3. Duality preserves incidences between points and lines; thus if point s lies on line l, then point $\mathcal{D}(l)$ lies on line $\mathcal{D}(s)$. (This duality is called *polar duality*; other dualities are possible.) Without loss of generality assume that set S is chosen in the first quadrant with no two points on a common line through the origin. Suppose points s, t, u are in counterclockwise order about the origin; that is, the direction vectors from the origin to s, t, and u are in counterclockwise order. Then Δstu is positive if and only if t lies on the opposite side of line su from the origin. This happens exactly if the point of intersection of lines $\mathcal{D}(s)$ and $\mathcal{D}(u)$ lies on the opposite side of line $\mathcal{D}(t)$ from the origin; see figure 3. Hence the orientation of every triple of points in S can be deduced from the combinatorial arrangement of the set of dual lines $\mathcal{D}(S)$ (assuming the region containing the origin is specified). Since the combinatorial arrangement is a planar graph with $O(n^2)$ edges and vertices, it has a representation of size about $O(n^2)$.

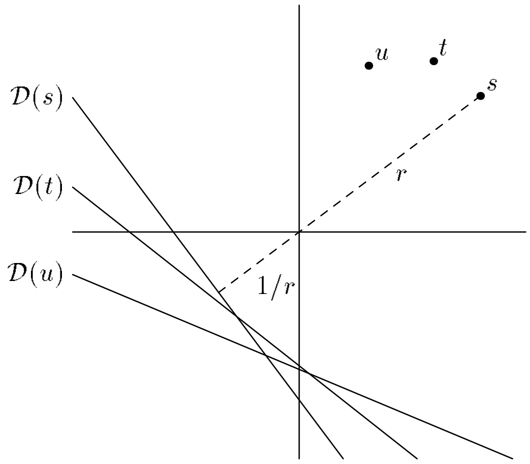


Figure 3: Duality and determining orientation.

Goodman and Pollack[45] and Alon[3] give the intriguing result that the number of distinct combinatorial arrangements resulting from an arrangement of n lines is at most n^{4n}. This means that it is possible to specify an arrangement with only $\log_2 n^{4n} = 4n \log_2 n$ bits; one method is to order the arrangements, say lexicographically, and specify an arrangement by its index in the order. The number of distinct combinatorial arrangements resulting from an arrangement of n pseudolines is about n^{n^2}; hence the Goodman-Pollack result implies that most combinatorial arrangements are only realized by pseudolines and not by lines. It is possible that a geometric algorithm involving lines could be improved by the use of a compact representation of a line arrangement. Unfortunately no compact representation is known that allows for efficient retrieval of information about the encoded lines.

In dimension $d \geq 3$, the convexity properties of a set of points $S \subset E^d$ can be defined in a similar fashion. The *orientation* of an ordered sequence of $d{+}1$ points in E^d determines whether the first point lies in the positive or negative halfspace of the oriented hyperplane through the remaining d points. The *configuration* of S is just the set of ordered sequences of $d{+}1$ points from S with positive orientation. A *convexity property* of S is a property determined by the configuration of S. Convexity properties include assertions about polytopes, convex and otherwise, though it can be a challenge to determine the formula expressing a property. The configuration of S is determined by the combinatorial arrangement of the hyperplanes dual to S, which has size $\theta(n^d)$. The disparity between the number of hyperplane and pseudohyperplane arrangements is even greater: there are about n^{n^d} combinatorial arrangements of pseudohyperplanes, but only about $n^{d^2 n}$ combinatorial arrangements result from hyperplanes.

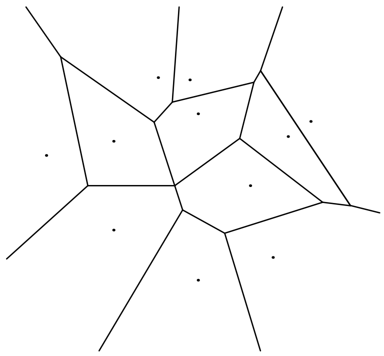


Figure 4: A Voronoi diagram in two dimensions.

2.3 Voronoi diagrams and arrangements

The Voronoi diagram of a set of n point sites in E^d partitions E^d into cells so that within each cell, all points have the same closest site. The Voronoi diagram can be obtained by mapping the sites to hyperplanes in E^{d+1}, determining the arrangement of the hyperplanes, and projecting a subset of the cells of the arrangement back to E^d. A generalization of the Voronoi diagram called the order-k Voronoi diagram also partitions E^d into cells, this time so that the k closest sites are invariant over a cell. It too can be obtained by projecting a different set of cells in the arrangement of hyperplanes in E^{d+1}.

This section describes the appropriate hyperplane arrangement. To make the geometry somewhat easier to visualize, we consider only the case where the set of sites is in dimension 2. The construction generalizes easily to arbitrary fixed dimension. We begin with a review of the precise definition of the Voronoi diagram; various references are available with more information[7, 30, 42].

Let S be a set of n points in the plane E^2. Elements of S are called *sites*. For $R \subseteq S$, the *Voronoi region* of R, $V(R)$, is the set of points of E^2 equidistant from every $r \in R$ and closer to every $r \in R$ than to any $r' \in S - R$. For a singleton set $R = \{r\}$, $V(R)$ is the set of points strictly closer to r than to any other site. Any point of the plane lies in $V(R')$ for some R', though R' may contain more than one site. The *Voronoi diagram* V is the collection of all nonempty Voronoi cells $V(R)$, for $R \subseteq S$. Figure 4 depicts a Voronoi diagram of 11 sites. The open regions are Voronoi cells of a single site; the edges are Voronoi cells of two sites; and the vertices are Voronoi cells of three or more sites.

We identify E^2 with the plane spanned by the first two coordinate axes of E^3; it is the *base plane*. We think of the third coordinate axis as the vertical direction. We write x for the point (x_1, x_2) in E^2 and if r is a real number, then (x, r) is the point (x_1, x_2, r) in E^3. The dot product $x \cdot p$ is $x_1 p_1 + x_2 p_2$.

For $s \in S$, consider the surface

$$U_s = \{(x, z) : x \in E^2, z = -x \cdot x + 2x \cdot s - s \cdot s\}.$$

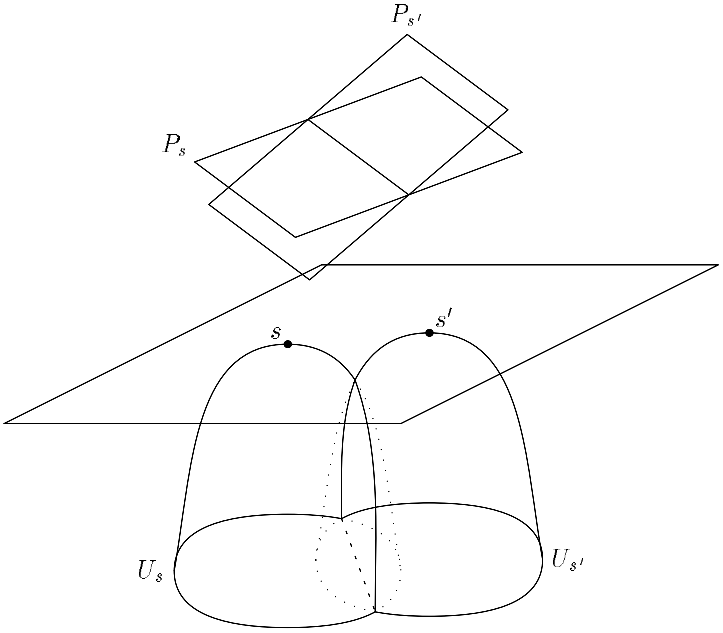


Figure 5: Vertical order of surfaces $\{U_s\}$ and planes $\{P_s\}$.

At a point x in the base plane, the vertical height of U_s is the negative square of the distance from x to s. Thus U_s is a paraboloid touching the base plane at s and below the base plane elsewhere. Now consider the collection of surfaces $\{U_s : s \in S\}$. At a point x in the base plane, the order on S determined by the vertical order of the surfaces $\{U_s\}$ below x is exactly the order on S determined by distance from x. See figure 5.

For site $s \in S$, consider the plane

$$P_s = \{(x, z) : x \in E^2, z = 2x \cdot s - s \cdot s\}.$$

Notice that plane P_s is not parallel to the vertical axis. If we think of P_s and U_s as functions from the base plane to the vertical direction, then we have $P_s(x) = U_s(x) + x \cdot x$. The crucial point is that the relationship between P_s and U_s is independent of s. Hence for any x in the base plane, the vertical order of the surfaces $\{U_s\}$ over x is identical to the vertical order of the planes $\{P_s\}$ over x.

Now consider the arrangement $\mathcal{A}$ of the set of planes $\{P_s\}$. Let I be the cell of the arrangement that is the intersection of the open halfspaces above the planes $\{P_s\}$. See figure 6. Let $\mathcal{F}$ be the set of faces of I. Clearly, a cell is in $\mathcal{F}$ if it is contained in at least one plane and is above every plane not containing it.

Lemma 2.1 *The Voronoi diagram V is the set of projections of the cells of $\mathcal{F}$ onto the base plane.*

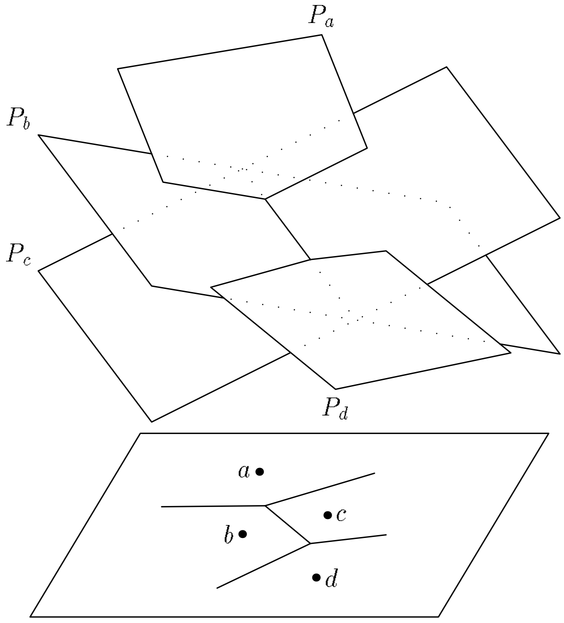


Figure 6: I is the cell above all planes P_a, P_b, P_c, P_d.

Proof. F is in $\mathcal{F}$ exactly if there is a subset R of S so that $F \subseteq P_r$ for every $r \in R$ and F is above $P_{r'}$ for every $r' \in S - R$. Hence if $(x, z) \in F$ for some x in the base plane, then x is equidistant from every $r \in R$ and closer to every $r \in R$ than to any $r' \in S - R$. Thus $x \in V(R)$. The converse direction is similar. $\square$

It is possible to give a visual interpretation of lemma 2.1. Use opaque paint to give each surface U_s a different color. An observer sitting very high above the surfaces will see the *upper envelope* of the collection of surfaces, that is, the portion of each surface not obscured by some surface above it. The pattern visible to the observer is the Voronoi diagram of S, since a surface U_s obscures all other surfaces exactly at points x of the base plane for which x is closer to s than to any other site. Now instead paint each plane P_s using corresponding colors. The observer sees exactly the same pattern, since for any x, the vertical order of the planes $\{P_s\}$ is the same as the vertical order of the surfaces $\{U_s\}$.

We now define the order-k Voronoi diagram and show that it can be obtained by projecting a different set of cells in the arrangement $\mathcal{A}$. We begin with a definition of the order-k Voronoi diagram. Some care is needed in the formal definition because the kth closest site to a point may not be uniquely determined. Let x be a point in E^d and consider the minimum radius sphere with at least k sites on or inside it. Let Q be the sites inside the sphere and R the sites on the sphere. The *k-neighbor signature* of x is the pair $(Q \cup R, \emptyset)$ if R has one element and (Q, R) if R has more than one element. The *order-k Voronoi cell* $V_k(Q, R)$ is the set of all points with k-neighbor signature (Q, R). The *order-k Voronoi diagram* V_k is the collection of all nonempty order-k Voronoi cells. Figure 7 gives the order-2 Voronoi diagram in solid lines and the ordinary Voronoi diagram in dotted lines.

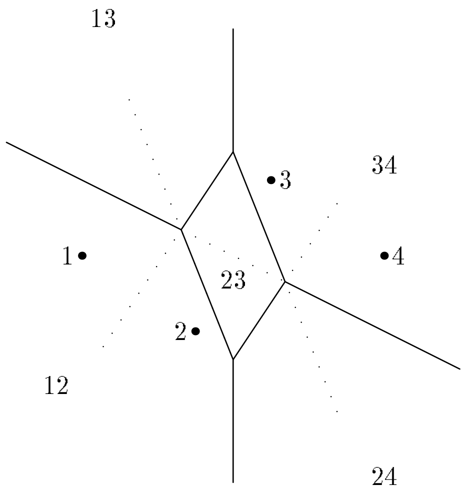


Figure 7: The order-2 Voronoi diagram is shown with solid lines; region labels are the closest pair of sites. The ordinary Voronoi diagram is dotted.

The ordinary Voronoi diagram V is the same as the order-1 Voronoi diagram V_1, using the identification $V(R) = V_1(R, \emptyset)$ if R is a singleton and $V(R) = V_1(\emptyset, R)$ if R has more than one element.

Consider again the arrangement $\mathcal{A}$ of the planes $\{P_s\}$. We assign indices to all cells of $\mathcal{A}$ except those of dimension 2. If a cell C is of dimension 3, then it is of *index k* if there are k planes above it and $n-k$ below it; if a cell C is of dimension 0 or 1, then it is of index k if there are fewer than k planes above it and fewer than $n-k$ below it. A cell of dimension 0 or 1 may have several indices.

The indexing of cells gives a "layered covering" of the base plane: first, for fixed k, the set of projections of index-k cells on the base plane forms a partition of the base plane; second, a cell of index i is either also a cell of index $i+1$, or it is above the set of cells of index $i+1$.

Lemma 2.2 *The order-k Voronoi diagram V_k is the set of projections onto the base plane of the index-k cells of the arrangement $\mathcal{A}$.*

The proof of this lemma is similar to the proof of lemma 2.1. A subtlety is that the mappings used for the order-1 Voronoi diagram V_1 are slightly different in the two lemmas. With lemma 2.1, a Voronoi cell $V(R)$ for a singleton set $R = \{r\}$ is obtained by projecting a face of I contained in a single plane. With lemma 2.2, the same Voronoi cell is obtained by projecting the open three-dimensional cell of the arrangement that sits directly below the face of I. For Voronoi cells $V(R)$ with $|R| > 1$, the mappings are identical in the two lemmas.

2.4 Combinatorial complexity of cells in an arrangement

The *combinatorial complexity* of a cell c in an arrangement, denoted $f(c)$, is the number of faces of all dimensions bounding c. The combinatorial complexity of a cell is relevant to algorithm analysis, since to examine a cell, an algorithm often must examine each of its faces.

Throughout this section, we take the dimension d as fixed and the number n of hyperplanes as variable; hence the big-O notation hides constants that depend on d, possibly exponentially. The following are basic facts about arrangements and combinatorial complexity. $|S|$ is the number of elements of set S and $\lfloor x \rfloor$ and $\lceil x \rceil$ are the integers obtained from x by rounding down and up, respectively.

Lemma 2.3 *[30] Let $\mathcal{A}$ be an arrangement of n hyperplanes in E^d.*

1. $|\mathcal{A}| \in O(n^d)$.
2. *For any cell c of $\mathcal{A}$,* $f(c) \in O(n^{\lfloor d/2 \rfloor})$.
3. $\sum_{c \in \mathcal{A}} f(c) \in O(n^d)$.

Here is a sketch of the proof of parts 1 and 3 of the lemma. Suppose $\mathcal{A}$ is *simple*, that is, any i-dimensional cell is contained in exactly $d-i$ hyperplanes. Then the signature vector of an i-dimensional cell c contains i zero entries and $n-i$ nonzero entries; by varying the zero entries, it is easy to see that c is a face of $3^{d-i}-1$ cells of higher dimension. For part 1, arbitrarily assign to each cell a vertex that lies on the boundary of the cell; since the number of vertices of $\mathcal{A}$ is at most $\binom{n}{d}$ and each vertex bounds at most 3^d-1 cells, the number of cells is $O(n^d)$. For part 3, the number of face-cell pairs is also only $O(n^d)$, since each cell is a face of at most a constant number of cells. (It is possible to give much sharper estimates of the constants[30].) If $\mathcal{A}$ is not simple, then a slight perturbation of the hyperplanes makes it simple and only increases the number of cells and face-cell pairs. Part 2 of the lemma follows from the upper bound theorem for convex polytopes[11, 30, 64], which is not elementary.

Part 3 implies that the combinatorial complexity of a cell is constant, averaged over the entire arrangement. Many of the results below can be viewed as giving the combinatorial complexity of cells averaged over various subsets of an arrangement.

Zone theorems. Let $\mathcal{A}$ be an arrangement and L an arbitrary set in E^d. The *zone* of L in $\mathcal{A}$, $Z_{\mathcal{A}}(L)$, is the set of cells of $\mathcal{A}$ intersected by L. The *zone complexity* of L in $\mathcal{A}$ is

$$ZC_{\mathcal{A}}(L) = \sum_{c \in Z_{\mathcal{A}}(h)} f(c);$$

the zone complexity counts a cell g once for each cell in the zone of L that has g as a face.

Edelsbrunner, O'Rourke and Seidel[35] give an algorithm to compute the arrangement of a set of hyperplanes; the algorithm is incremental, like the one described before for arrangements of lines. The following theorem is used in the analysis of the algorithm.

Theorem 2.4 *(Zone theorem) Let $\mathcal{A}$ be an arrangement of n hyperplanes in E^d, $d \geq 2$. Then for any hyperplane h, $ZC_{\mathcal{A}}(h) \in O(n^{d-1})$.*

The Zone Theorem implies that adding each hyperplane takes time $O(n^{d-1})$; hence the time required to compute the arrangement of n hyperplanes is $O(n^d)$. Notice that the cells in $Z_{\mathcal{A}}(h)$ are in one-one correspondence with the cells in the $d-1$-dimensional arrangement within h induced by the intersection of the other hyperplanes with h; hence $|Z_{\mathcal{A}}(h)| \in O(n^{d-1})$ and the average complexity of a cell in $Z_{\mathcal{A}}(h)$ is constant. Edelsbrunner, O'Rourke and Seidel give a rather complicated proof of the Zone Theorem; unfortunately, the proof is incorrect for $d \geq 3$. Edelsbrunner, Seidel, and Sharir[37] give a simpler (and presumably correct) proof, rescuing the analysis of the arrangement algorithm. The new proof is a clever simultaneous induction on n and d, somehow managing to bound zone complexity within an arrangement of n hyperplanes in d dimensions in terms of zone complexity with an arrangement of $n-1$ hyperplanes in d dimensions and zone complexity within an arrangement of n hyperplanes in $d-1$ dimensions.

The new proof technique actually gives a number of new results. One concerns the zone of more general objects than just hyperplanes. Suppose $\mathcal{A}$ is an arrangement of n hyperplanes in E^d and suppose for $p < d$ that σ is either an algebraic surface of constant degree or the boundary of an arbitrary convex set lying in a $(p+1)$-dimensional flat. Aronov, Pellegrini, and Sharir[5] show that

$$ZC_{\mathcal{A}}(\sigma) \in \begin{cases} O(n^{\lfloor \frac{d+p}{2} \rfloor}) & \text{if } d+p \text{ even} \\ O(n^{\lfloor \frac{d+p}{2} \rfloor} \log n) & \text{if } d+p \text{ odd.} \end{cases}$$

In the case that σ is a flat, this extended zone theorem interpolates between various previously known bounds. Consider the zone of a p-flat, as p varies from 0 to d. A 0-flat is a point, so the zone of a 0-flat is a single cell. The combinatorial complexity of a single cell is $O(n^{\lfloor d/2 \rfloor})$, which agrees with the extended zone theorem up to a possible factor of $\log n$. Every time the dimension p of the flat increases by 2, the number of zone faces increases asymptotically by a factor of n; when $p = d-1$ the flat is a hyperplane, and the result agrees with the zone theorem up to a possible factor of $\log n$.

Using similar techniques, Aronov, Matoušek and Sharir[4] show that

$$\sum_{c \in \mathcal{A}} f(c)^2 \in O(n^d \log^{\lfloor d/2 \rfloor - 1} n).$$

So not only is the average complexity of a cell constant, the variance of cell complexity is not very large either. The natural conjecture is that the odd log term in the sum can be eliminated; this would yield an asymptotically tight bound.

Many cells. The zone theorems describe the complexity of the cells in an arrangement selected by intersection with some nicely-behaved set. A different method is to allow the selection to be arbitrary, in particular worst-possible, and to measure the total complexity of the set of cells as a function of the number of cells selected. Let L be a set of n lines in the plane with arrangement $\mathcal{A}(L)$. Clarkson *et al.*[19] show that the total number of edges bounding any m regions in $\mathcal{A}(L)$ is

$$O(m^{2/3}n^{2/3} + n).$$

Stated another way, if m regions are selected, the average complexity of each region is $O(\max(n/m, n^{2/3}/m^{1/3}))$; the crossover from the first term to the second term comes at $m = O(\sqrt{n})$ when the average complexity of a region is $O(\sqrt{n})$. This bound is tight, in the sense that it is possible to choose an arrangement of n lines and m cells within the arrangement so that the total complexity of the chosen cells is $\Omega(m^{2/3}n^{2/3} + n)$. Clarkson *et al.* also give similar bounds for pseudolines, circles, and pseudocircles. Edelsbrunner, Guibas, and Sharir[32] show that given n lines and m points, the portion of the arrangement consisting of the regions that contain the m points can be constructed in time close to $O(m^{2/3}n^{2/3} + m + n)$; a similar bound holds if the lines are replaced with line segments.

In dimension 3, Edelsbrunner, Guibas, and Sharir[33] show that the total complexity of m cells in an arrangement of n hyperplanes is $O(m^{2/3}n \log n + n^2)$; this almost matches the known lower bound of $\Omega(m^{2/3}n + n^2)$. In dimension $d \geq 3$, Edelsbrunner, Guibas, and Sharir also give a bound of $O(m^{2/3}n^{d/3} \log n + n^d)$ on the total number of facets ($d-1$-dimensional cells) bounding m cells in an arrangement of n hyperplane. This bound does not count the number of lower-dimensional faces bounding the cells; it is possible that this number could be much larger. However, Aronov, Matoušek and Sharir[4] give a bound of approximately $O(m^{1/2}n^{d/2})$ that does include the lower-dimensional faces.

The same proof techniques give bounds for an old problem of combinatorial geometry, counting the maximum possible number of incidences between points and lines. Given m points and n lines, what is the maximum number of incidences, that is, pairs consisting of a point and a line so that the point lies on the line? Clarkson *et al.*[17, 19] show that the answer is $\theta(m^{2/3}n^{2/3} + m + n)$ in the plane. As the bound suggests, the proof technique is similar to the many-regions bound. Bounds are also available for incidences between points and circles[19], and in dimension 3, between points and planes[33] and between points and spheres[19].

3 A random-incremental convex hull algorithm

We give a fairly simple random-incremental algorithm that computes the convex hull of a set of points. The algorithm is a variant of the incremental or beneath-beyond algorithm[30, 79] for convex hulls. The algorithm inserts points one by one, updating the convex hull after each addition. The algorithm is "random-incremental" because the permutation used to

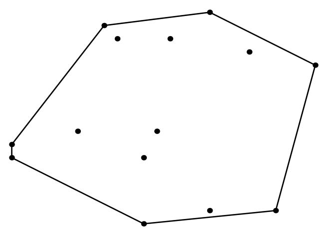


Figure 8: A convex hull in two dimensions.

order the sites is chosen randomly, with each permutation equally likely. We describe the algorithm in three dimensions, but it generalizes easily to arbitrary dimension.

For a set of n points in three dimensions, the algorithm has expected running time $O(n \log n)$. The expectation of the running time is taken over the permutations used to order insertion. The $O(n \log n)$ expected time bound holds for the worst-case set of sites; for some sets of sites the expected running time may be smaller, even linear. With some small probability, the algorithm will run for more than time $O(n \log n)$, with $O(n^2)$ the worst case. However, the output produced by the algorithm is always correct. The running time of the algorithm in dimension higher than three is discussed below.

The algorithm maintains a triangulation of the current convex hull obtained from the sequence of intermediate convex hulls. The triangulation is used as a search structure; for a new point the search structure can be used to report either that the new point is inside the current hull or to find a facet of the current hull visible from the new point.

Convex hulls. Before considering the algorithm, we begin with a short discussion on convex hulls, triangulations, and the orientation primitive. For further discussion on convexity, see books by Brøndsted[11], Edelsbrunner[30], or Grunbaum[50].

Let S be a set of points in E^3. The *convex hull* of S is the intersection of all convex sets containing S; it is a convex polytope. A *face* of a convex polytope is the relative interior of the intersection of the convex polytope with a plane supporting the convex polytope. A two-dimensional face is a *facet*; it is the interior of a convex polygon. A one-dimensional face is an *edge*; it is an open line segment. A zero-dimensional face is a *vertex*. The facets, edges, and vertices have the structure of a planar graph. Figure 9 gives the convex hull of a point set chosen on a sphere in three dimensions. For comparison, figure 8 gives the convex hull of a point set in the plane, including points inside the convex hull. S is *affinely independent* if no three points lie on a common line and no four points lie on a common plane. If S is affinely independent, then every facet of the convex hull of S is the interior of a triangle.

What happens to the convex hull of a set of points when a new point is added? Suppose $S \cup \{s\} \subset E^3$ is a finite set of affinely independent points, H is the convex hull of S, and $s \notin H$. A facet of H is *visible* from s if s and the interior of H are on opposite sides of the plane through the facet; a facet is *invisible* from s if s and the interior of H are on

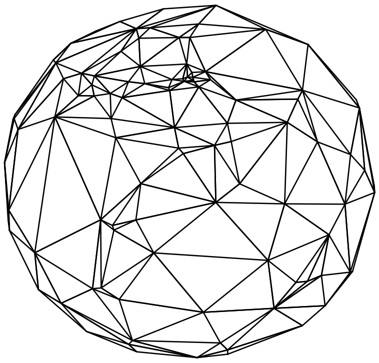


Figure 9: The convex hull of points chosen on a sphere in three dimensions.

the same side of the plane. An edge of H is a *horizon* edge from s if the interior of H lies to one side of the plane determined by the edge and s. Viewed from s, the horizon edges of H form a polygonal cycle separating the visible facets from the invisible facets. The facets of the convex hull H' of $S \cup \{s\}$ are exactly the facets of H invisible from s and facets formed by s and a horizon edge of H from s. For F a facet of H, let $\mathrm{tet}(F, s)$ be the tetrahedron formed from the vertices of F and s. Then the set $H' - H$ is the union of $\{\mathrm{tet}(F, s)\}$ for F a visible facet of H from s; the tetrahedra have disjoint interiors. See figure 11 for the evolution of the convex hull as points are added.

A *triangulation* of the convex hull is a decomposition of the convex hull into tetrahedra so that intersection of any two tetrahedra is either empty or a common face. A directed graph, the *triangulation graph*, can be used to represent the triangulation. The nodes of the graph are the tetrahedra; a node should be labeled with the points forming the tetrahedron. The arcs correspond to facets, with an arc in each direction between two tetrahedra sharing a common facet. There are four arcs incident to a tetrahedron; a useful correspondence is that the ith arc is the facet obtained by omitting the ith point defining the tetrahedron. A convenient way to traverse boundary facets of the triangulation is to add a symbolic point $\perp$, and for every boundary facet F, create an *external* tetrahedron $\mathrm{ext}(F)$ from the vertices of F and $\perp$. If facets F and F' have a common edge e, then $\mathrm{ext}(F)$ and $\mathrm{ext}(F')$ have a common facet formed from the vertices of e and $\perp$. The external tetrahedra do not have a geometric realization (at least in ordinary Euclidean space). The *boundary graph* is the subgraph of the triangulation graph consisting of the external tetrahedra and connecting arcs; clearly it captures the incidence structure of facets and edges on the boundary of the triangulation. Figure 10 shows the graph that results from the triangulation consisting of a single tetrahedron.

The only geometric primitive required by the algorithm is the orientation test on four points $p_1, p_2, p_3, p_4 \in E^3$; it decides whether p_1 lies on the positive side, on, or on the negative side of the oriented plane through p_2, p_3, p_4. Orientation can be decided by evaluating the sign of an appropriate determinant (see section 4). For repeated orientation tests with p_2, p_3, p_4

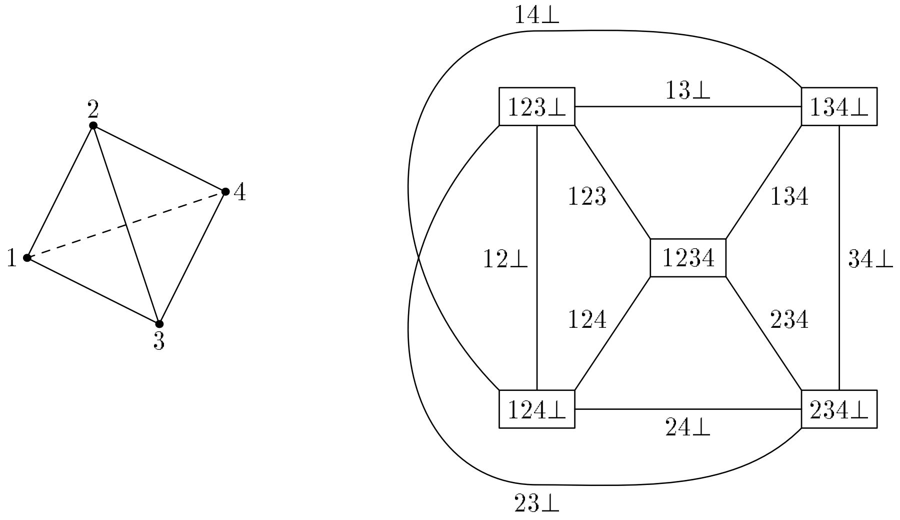


Figure 10: The graph of a triangulation containing one tetrahedron.

fixed, the coefficients of the plane equation through p_2, p_3, p_4 can be evaluated and the orientation test becomes a dot-product and a comparison. This optimization is appropriate in the algorithm described below if the points p_2, p_3, p_4 form a facet of an intermediate convex hull.

The algorithm. Let S be a set of $n \geq 4$ points in E^3. We assume that S is affinely independent. This assumption is not crucial; see the discussion below and the discussion of symbolic perturbation schemes in section 4. The algorithm starts by picking a random permutation used to order the points in S. Label the points $S = \{s_1, \ldots, s_n\}$ in the order given by the permutation.

At the top level of description, all the algorithm does is maintain a triangulation of the convex hull, using the triangulation graph described before. To start, the algorithm creates the triangulation T_4 consisting of the tetrahedron whose vertices are the first four points s_1, s_2, s_3, s_4; the tetrahedron has *parent* facet formed from s_1, s_2, s_3 and *apex* s_4. For the iterative step, suppose T_{i-1} is the current triangulation. If the next point s_i is inside some tetrahedron of T_{i-1}, then it is inside the current convex hull, and $T_i = T_{i-1}$. Otherwise, s_i is outside the convex hull. Let V_i be the set of boundary facets of the current hull visible from s_i. The triangulation T_i is obtained by adding the set of tetrahedra

$$W_i = \{\mathrm{tet}(F, s_i) : F \in V_i\}$$

to T_{i-1}. Facet $F \in V_i$ is the *parent* facet and s_i the *apex* of $\mathrm{tet}(F, s_i)$. Figure 11 shows the evolution of the triangulation T_i over a short sequence of insertions. The final triangulation graph is shown, omitting the external tetrahedra for clarity. In the triangulation graph, the arc leaving a tetrahedron node to the left corresponds to the parent facet of the tetrahedron.

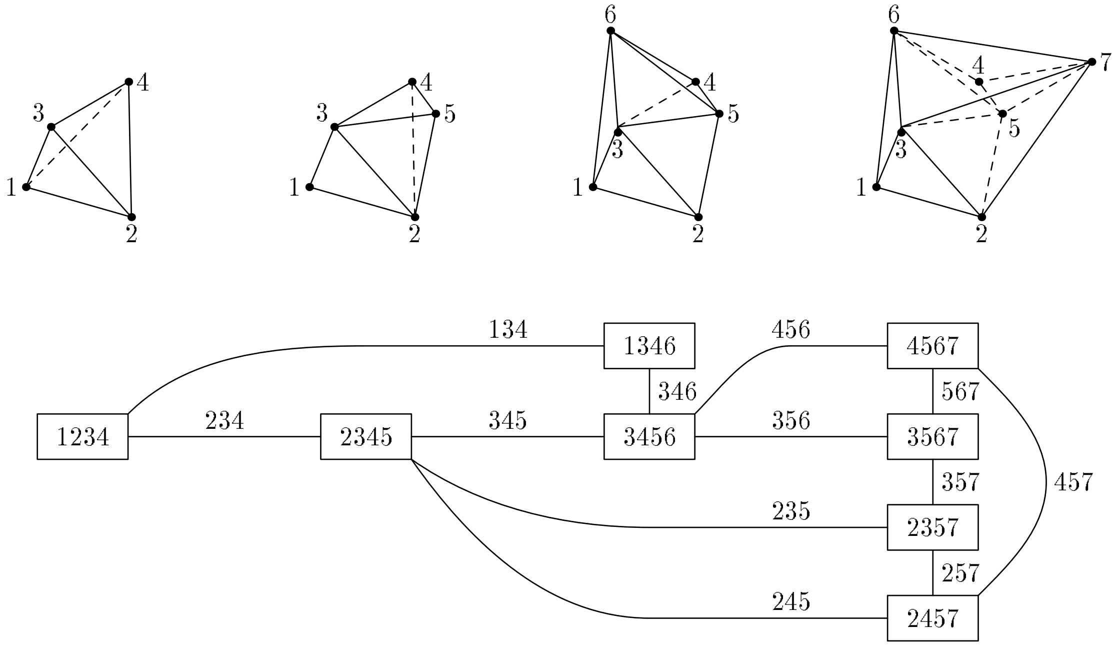


Figure 11: The evolution of the triangulation.

To expand the description of the algorithm, suppose s_i is the new point to be added. Say a tetrahedron Δ in the triangulation is *active* if s_i and Δ are on the same side of the plane through the parent facet of Δ. Consider a depth-first search in the triangulation graph of T_{i-1} starting at the initial tetrahedron but restricted to visit only active tetrahedra. (The initial tetrahedron may or may not be active, depending upon whether or not s_4 and s_i are on the same side of the plane through s_1, s_2, s_3, but the search should include the initial tetrahedron.) The search will either find a boundary facet $F_0 \in V_i$ or report that s_i is inside some tetrahedron of T_{i-1}. To see this, choose a point u inside the initial tetrahedron so that segment us_i avoids edges and vertices of T_{i-1}. Choose k maximal so that s_i is outside T_k; if $k \neq i-1$, then s_i must be inside some tetrahedron with apex s_k. Now any tetrahedron Δ intersected by segment us_i is active, since the plane through the parent facet of Δ separates u inside T_k from s_i outside T_k. Hence there is a path of active tetrahedra to the tetrahedron containing s_i or to a boundary facet of T_{i-1} visible from s_i.

Suppose the search reports that $F_0 \in V_i$. All the facets in V_i can be found by depth-first search in the boundary graph starting at F_0, backtracking at any transition to a facet not in V_i. For each facet $F \in V_i$, the tetrahedron $\mathrm{tet}(F, s_i)$ can be 'created' simply by relabeling $\perp$ as s_i in the tetrahedron $\mathrm{ext}(F)$. All the adjacencies among the relabeled tetrahedra should be preserved, as well as the adjacencies between the relabeled tetrahedra and other tetrahedra in T_{i-1}. In addition, for each new boundary facet G, the tetrahedron $\mathrm{ext}(G)$ must be created and linked into the graph. Each such new boundary facet G is formed from s_i and a horizon edge e between an old boundary facet $F \in V_i$ and an old

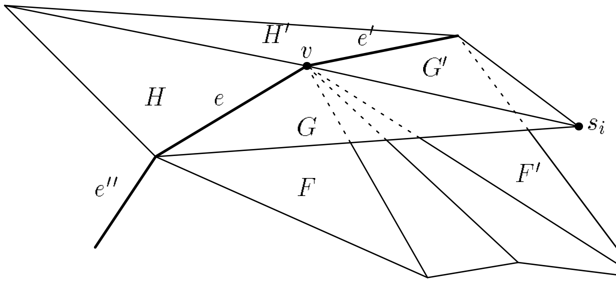


Figure 12: The walk to find adjacencies among new boundary facets; facet F'' not shown.

boundary facet $H \notin V_i$; each such edge e is encountered during the depth-first search. The tetrahedron $\mathrm{ext}(G)$ is adjacent to $\mathrm{ext}(H)$, $\mathrm{tet}(F, s_i)$ (called $\mathrm{ext}(F)$ before relabeling), and two new external tetrahedra $\mathrm{ext}(G')$ and $\mathrm{ext}(G'')$. Facet G' is formed from s_i and a horizon edge e' sharing a common vertex v with e. Edge e' can be found from edge e by a walk in the boundary graph through facets of V_i incident to v; e' is the edge between the last facet $F' \in V_i$ and the first facet $H' \notin V_i$. See figure 12. The walk can be repeated with the other vertex of e to find $\mathrm{ext}(F'')$.

The analysis. Though the algorithm has been described in dimension $d = 3$, we couch most of the analysis with d as a parameter. In part this is because the analysis is general, and in part this is to show where otherwise mysterious numeric constants arise.

A *random* element of S is a choice where each element of S has the same probability; similarly a *random* subset of S of size r is a choice where each subset of size r has equal probability. If S is labeled $\{s_1, \ldots, s_n\}$ using a permutation with each permutation equally likely, then s_1 and s_r are random elements and $\{s_1, \ldots, s_r\}$ is a random subset of size r.

For $r = 4, \ldots, n$, let h_r be the expected number of facets on the convex hull of a random subset of size r. For any $R \subseteq S \subset E^3$ of size r, the boundary of the convex hull of R has the structure of a planar graph, so the convex hull has at most twice as many facets as vertices, i.e. at most $2r$ facets. Hence $h_r \leq 2r$; for specific sets S this may be a considerable overestimate.

Lemma 3.1 *For i between $d+1$ and n, let t_i be the number of tetrahedra in T_i. The expectation of t_i is*

$$E[t_i] = d \sum_{r=d+1}^{i} \frac{h_r}{r}.$$

In dimension $d = 3$, it follows immediately that for any set S of size n, $E[t_n] \leq 6n$.

Proof. Let u_r be the number of tetrahedra created when s_r is added, for $r = d{+}1$ to $r = i$. Hence

$$t_i = \sum_{r=d+1}^{i} u_r.$$

Now u_r is also the number of boundary facets of T_r incident to s_r; notice $u_r = 0$ if s_r is inside T_r. The expected number of vertex-facet incidences in T_r is dh_r since the expected number of boundary facets of T_r is h_r and each facet is incident to d vertices. Hence the expectation of u_r is dh_r/r, since s_r is a random element of S_r. The lemma follows using the linearity of expectation. □

Lemma 3.2 *Let a_i be the number of active tetrahedra in T_{i-1} when s_i is inserted. The expectation of a_i is*

$$E[a_i] = -\frac{dh_i}{i} + d(d{-}1)\sum_{r=d}^{i-1} \frac{h_r}{r(r{-}1)}.$$

Proof. Consider running the algorithm on the points in the order $s_i, s_1, \ldots, s_{i-1}$. For $r = 3$ to $i{-}1$, let T'_r be the intermediate triangulation so obtained after inserting the $r{+}1$ points $s_i, s_1, \ldots, s_r$. Thinking of the triangulations as sets of tetrahedra, we claim that $T_{i-1} - T'_{i-1}$ is exactly the set of tetrahedra of T_{i-1} active when s_i is inserted, and that $T'_{i-1} - T_{i-1}$ is exactly the set of tetrahedra of T'_{i-1} incident to s_i. To see these claims, we compare the evolution of the two triangulations T and T' as points are added. (To handle the initial tetrahedron of T, think of it as being formed from the facet $s_1s_2s_3$ when s_4 is added.) Let s_r be the new point to be added. Suppose F is a boundary facet of T_{r-1} visible from s_r. Then F is not a boundary facet of T'_{r-1} exactly if s_i and s_r are on the same side of the plane through F. Hence $\mathrm{tet}(F, s_r)$ is created in T_r but not T'_r exactly if $\mathrm{tet}(F, s_r)$ is active when s_i is inserted into T_{i-1}. Now suppose F is a boundary facet of T'_{r-1} visible from s_r. Then F is not a boundary facet of T_{r-1} exactly if F is incident to s_i. Hence $\mathrm{tet}(F, s_r)$ is created in T'_r but not T_r exactly if $\mathrm{tet}(F, s_r)$ is incident to s_i.

We have that the number of active tetrahedra in T_{i-1} when s_i is inserted is

$$a_i = |T_{i-1} - T'_{i-1}| = |T'_{i-1} - T_{i-1}| + |T_{i-1}| - |T'_{i-1}|.$$

Using lemma 3.1 and the fact that $\{s_1, \ldots s_r\}$ and $\{s_i, s_1, \ldots, s_r\}$ are both random subsets, we have

$$E[|T_{i-1}| - |T'_{i-1}|] = d\sum_{r=d+1}^{i-1} \frac{h_r}{r} - d\sum_{r=d+1}^{i} \frac{h_r}{r} = -\frac{dh_i}{i}.$$

For $r{=}1$ to i, let b_r be the number of tetrahedra incident to s_i that are created in T' when s_r is added; then

$$|T'_{i-1} - T_{i-1}| = \sum_{r=d}^{i-1} b_r.$$

Now b_r is also the number of boundary facets of T'_r incident to both s_i and s_r. The total number of incidences between a facet and a pair of vertices is $\binom{d}{2}$. Since s_i and s_r are both random vertices and h_r is the expected number of boundary facets of T'_r, we have

$$E[b_r] = \frac{\binom{d}{2} h_r}{\binom{r}{2}} = \frac{d(d-1)h_r}{r(r-1)}.$$

The lemma follows by combining all the terms and using the linearity of expectation. □

Theorem 3.3 *Let S be a set of n affinely independent points with $S \subset E^3$. The random-incremental algorithm computes the convex hull of S in expected time $O(n \log n)$ and expected space $O(n)$. The time bound holds for any set of sites S; the expectation is over the permutation used to insert sites.*

Proof. The total space used is determined by the number of tetrahedra; the linear expected bound follows using lemma 3.1. Consider the time spent adding point s_i. The search for a facet visible from s_i takes at most time proportional to the number of active tetrahedra. Using the bound $h_r \leq 2r$ and lemma 3.2, this time has expectation less than

$$\sum_{r=4}^{i} \frac{12}{r-1},$$

which is $O(\log i)$. The time spent updating the triangulation graph when s_i is added is proportional to the number of tetrahedra added: this is obvious, except possibly for the time spent walking the facets of V_i to discover adjacencies among new external tetrahedra. However, notice that each facet in V_i is visited during a walk at most once for each of its vertices, so the total time spent walking is $O(|V_i|)$. Summing the two running times over all i yields expected time $O(n \log n)$, since the expected total number of tetrahedra in the triangulation is linear. □

Affine independence. The algorithm was described assuming affine independence. This requirement could be simulated with symbolic perturbation schemes[34]; see section 4. In fact, the requirement is easily removed. One change concerns the initial tetrahedron. If the first four points are coplanar, then the algorithm could compute the two-dimensional convex hull in the spanning plane until some nonplanar point is encountered, and a proper three-dimensional triangulation can be constructed. Alternatively, the first four points could be chosen explicitly not to be coplanar. The second change concerns adding a new point s. It suffices to declare a convex hull facet invisible if s lies in the plane through the facet; a horizon edge is now defined to be an edge between a visible facet and an invisible facet. If the new point s lies in a facet, then it is just discarded. If s lies outside the hull, but possibly in the plane through a facet f, then the augmented convex hull will be triangulated using

s and facets visible from s. The augmented convex hull will have a new facet f', possibly not triangular, formed from f and s. The algorithm need not treat f' as a special case; f' is the union of triangles that are facets of tetrahedra triangulating the augmented convex hull. If desired, a post-processing step can merge coplanar triangular facets, and produce a description of the true facet structure of the convex hull. A subtle point is that the running time analysis given earlier requires affine independence. A reasonable conjecture is that the running time analysis can be extended to affinely dependent points (without using a symbolic perturbation scheme), but this is apparently not known.

Higher dimensions. The algorithm generalizes to compute the convex hull of a set of n points in d-dimensional Euclidean space E^d, for $d > 3$. As before, the algorithm maintains a triangulation of the convex hull of the inserted points. A triangulation is now a decomposition into d-simplices; a d-simplex is the convex hull of $d+1$ affinely independent points. The triangulation graph now has directed arcs corresponding to facets ($(d-1)$-simplices) between d-simplices; each d-simplex is adjacent to $d+1$ other d-simplices. Both the search through the graph to find a visible facet and the update of the graph to reflect the new d-simplices are very similar to the case in three dimensions. A technical point concerns the walk to discover adjacencies among new external tetrahedra; it is now a walk through the visible facets holding a ridge ($(d-2)$-simplex) constant. Lemmas 3.1 and 3.2 go through with little more change than substituting 'd-simplex' for 'tetrahedron'. For a more complete discussion, see the paper by Clarkson, Mehlhorn, and Seidel[20].

In dimension $d > 3$, the running time of the algorithm is very dependent upon the particular set of points. By the upper bound theorem for convex polytopes[50], the number of facets on the convex hull of a set S of n points in E^d is at most $O(n^{\lfloor d/2 \rfloor})$. An analysis using lemmas 3.1 and 3.2 and the fact that any subset of $R \subseteq S$ with $|R| = r$ has $O(r^{\lfloor d/2 \rfloor})$ facets on its convex hull shows that the expected running time of the random-incremental algorithm on S is at most $O(n^{\lfloor d/2 \rfloor})$. There are sets that have complexity $\Omega(n^{\lfloor d/2 \rfloor})$; an example is a set of n points chosen arbitrarily from the moment curve $\{(x, x^2, \ldots, x^d) : x \in R\} \subset E^d$[50]. Thus the algorithm is asymptotically optimal in the worst-case sense, since it takes time proportional to the size of its output.

For uniformly distributed points, the situation is very different. If S is a set of n points uniformly distributed inside a sphere in E^d, $d \geq 2$, then the expected number of convex hull facets is $O(n^{(d-1)/(d+1)})$[28, 82]. Again using lemmas 3.1 and 3.2, the expected running time of the convex hull algorithm on S is $O(n)$. Informally, all of the interior points will be eliminated very quickly in the search through the triangulation. For other distributions, the expected number of convex hull facets may be even smaller[28], again leading to a linear algorithm.

Convex hull algorithms can be used to compute Delaunay triangulations[30, 42, 79]. Define the lifting map $\lambda : E^d \to E^{d+1}$ by $\lambda(x_1, \ldots, x_d) = (x_1, \ldots, x_d, x_1^2 + \ldots x_d^2)$. Given a set S of n points in E^d, the Delaunay triangulation is isomorphic to the set of lower faces of the convex hull of $\lambda(S)$, and can be obtained in time linear in the number of faces[30, 42]. For any set S of n points in dimension 2, $\lambda(S) \subset E^3$, and the expected running time is $O(n \log n)$ by theorem 3.3. In each dimension $d \geq 3$, there is a set of n points $S \subset E^d$ so that the Delaunay triangulation of S has $O(n^{\lceil d/2 \rceil})$ d-simplices, and it is possible to show that the convex hull algorithm on $\lambda(S)$ has the same asymptotic expected running time. However, this worst-case bound is excessively pessimistic for many point sets. Dwyer[29] shows that if a set S of n points is chosen with a uniform distribution inside a sphere in E^d, then the expected number of facets on the convex hull of $\lambda(S)$ is linear in n, though with a constant of proportionality that grows exponentially with the dimension d. An analysis using lemmas 3.1 and 3.2 shows that the expected running time of the convex hull algorithm on $\lambda(S)$ is $O(n \log n)$. Dwyer conjectures that the same linear expected size bound holds if S is chosen from a uniform distribution inside any bounded convex set C. Plausibly the algorithm has expected running time $O(n \log n)$ on any "reasonably distributed" point set.

References. The original analysis of a random-incremental convex hull algorithm is due to Clarkson and Shor[21]. They analyze a somewhat different algorithm than the one presented here. Their algorithm maintains a "conflict graph" with an edge between an uninserted point and a facet of the current convex hull if the facet is visible from the point. The conflict graph makes finding the facets visible from a new point trivial; the work is in maintaining the conflict graph. Clarkson and Shor also give a different analysis of the running time of the algorithm.

The remarkable analysis given here, due to Clarkson, Mehlhorn, and Seidel[17, 20], gives exact bounds on the expected size of the triangulation and the number of active tetrahedra in terms of the values $\{h_r\}$. The values $\{h_r\}$ as a function of the size of the point set vary widely depending upon the distribution of the point set; it is useful to have a precise predictor of the performance of the algorithm based on some property of the point set.

Random-incremental algorithms have been analyzed for a variety of geometric problems. These include computing intersections of line segments and trapezoidal diagrams[21, 74, 75], intersection of balls in three dimensions[21], and Delaunay triangulations[8, 52]. Algorithms are available for deletions as well as insertions. For example, Clarkson, Mehlhorn and Seidel[20] show how to delete a point from the triangulation of the convex hull; the resulting triangulation is the same as the triangulation that would have been obtained by inserting the points in the original order, omitting the deleted point. Clarkson[17] and Seidel[85] survey randomized geometric algorithms.

Chazelle[13] shows how to "derandomize" the convex hull algorithm, that is, how to choose the insertion order in a deterministic fashion that guarantees the same asymptotic running time. Unfortunately, the algorithm to choose the order is quite complicated.

Empirical performance. The algorithm has been implemented in the C programming language. The implementation requires about 1000 lines of C; a third of this is the high-level algorithm and the remainder is input-output and geometric primitives. The dimension of

d	n=500	n=1000	n=5000	n=10000	n=50000	n=100000
2			3.9 (56)	7 (69)	54 (125)	112 (160)
3	1.5 (92)	2.8 (127)	12.2 (310)	23 (446)	157 (971)	304 (1408)
4	5.3 (190)	11.2 (300)	57.6 (845)	117 (1290)		
5	30.1 (290)	63.2 (490)	335.0 (1580)			
6	209.0 (382)					

Figure 13: Points uniformly distributed inside a sphere: running time in seconds and number of extreme points.

the data is an input parameter; examples up to dimension 7 have been tried. Floating-point arithmetic is used for the geometric primitives, with heuristics that improve reliability but do not guarantee correctness. Figures 13 and 14 give timings for the algorithm in seconds on a VAX 8550. Each time except the last in each row is an average of three runs; the times varied up to 10%. The timings should be taken as suggestive only, since the implementation has not been extensively optimized. Figure 13 gives the running time for points chosen inside the unit sphere; the points are generated using the library pseudorandom number generator. The parenthesized number in each entry is the approximate number of extreme points. If the algorithm is going to be used for points that really do come from a uniform distribution, it is useful to filter the input to quickly eliminate points that are not extreme[25, 46]. Figure 14 gives times for the computation of Delaunay triangulations. The points are generated uniformly in the unit cube and lifted to one higher dimension using the lifting map λ. Of course, all lifted points are extreme.

For large problems, storage may be more of an issue than running time. For example, approximately 70 megabytes of main storage were required to compute the Delaunay triangulation of 5000 points in 4 dimensions. The actual clock time was approximately a factor of 10 higher because of paging overhead. The implementation was not particularly designed to minimize storage usage. One heavy use of storage is the triangulation data structure. By lemma 3.1, the number of simplices in the triangulation is approximately d times larger than the number of simplices on the convex hull. A variant of the random-incremental algorithm that maintains a conflict graph[21] may allow a savings of a factor of d in storage.

4 Robustness

Geometric algorithms are usually described assuming "general position" and the "real random access machine" (real RAM)[79] as model of computation. The real RAM model allows real numbers to be represented exactly and provides exact unit-time arithmetic operations $=$, $<$, $+$, $-$, $\times$, $\div$. Sometimes more complicated operations such as $\sqrt{\ }$ or log or sin are allowed as well. The general position assumption varies from problem to problem; for a problem

d	n=500	n=1000	n=5000	n=10000	n=50000
2	2.2	4.5	24.2	51.0	285.0
3	8.4	18.5	102.4	210.0	-
4	45.3	103.0	638.0	-	-
5	287.0	-	-	-	-

Figure 14: Performance of the convex hull algorithm used for Delaunay triangulations.

involving points in the plane it might be that no three points lie on a common line and no two points have the same x- or y-coordinate. These two assumptions considerably simplify the description of geometric algorithms. However, the real RAM is not implementable and the data of interest may not be in general position. An implementation of a geometric algorithm can fail, even though the algorithm has purportedly been proven correct, because the implementation does not adequately address these assumptions.

The term "robustness" has been applied generically to research aimed at examining the real RAM assumption and the general position assumption in more detail. These two assumptions are related and interact with one another, but affect an implementation in different ways. The real RAM assumption is usually made because exact results are needed from arithmetic operations, not because the generality of real numbers is needed; for many problems it is acceptable to substitute integers or rationals with exact arithmetic. Implementation difficulties can arise from the naive substitution of approximate floating point arithmetic for exact real arithmetic. The general position assumption is made to avoid dealing with nasty special cases, such as three points lying on a common line. The general position assumption really only makes formal sense if arithmetic is exact. Thus using exact arithmetic and the exact coordinates of points, it is possible to detect the collinearity of three points. With floating point arithmetic, the best that can be done is to assert reliably either that the three points are not collinear or that they are "almost" collinear; exact collinearity probably cannot be detected.

This section surveys various approaches that can be used to address the real RAM assumption and the general position assumption. Other discussions of robustness are also available[55, 65, 92]. Section 4.1 enumerates some fundamental geometric primitives. Section 4.2 discusses the implementation of geometric algorithms in exact integer and rational arithmetic. Exact arithmetic guarantees that all primitives give correct answers, ensuring implementation correctness. In addition, there are schemes that in some cases allow input data to be symbolically perturbed into general position. However, exact representation of geometric data may require arithmetic of high bit complexity, increasing the running time of the implementation. Section 4.3 discusses ways to avoid high bit complexity, for example by periodically rounding coordinates to lower precision or by carefully structuring algorithms to avoid growth in bit complexity. Implementing geometric algorithms using floating point arithmetic is discussed in section 4.4. In general, there is no reason to expect that an exact-arithmetic algorithm implemented using floating point should be correct. However, there has been some research aimed at proving correctness in floating point arithmetic as well.

4.1 Geometric primitives.

Given a sequence of $d{+}1$ points $(p_1, \ldots, p_{d+1})$ in E^d, the *orientation test* decides on which side of the oriented hyperplane through $p_1, \ldots, p_d$ point p_{d+1} lies. Let $p_i = (p_{i1}, p_{i2}, \ldots, p_{id})$, $i = 1, \ldots, d{+}1$. The orientation of $p_1, \ldots, p_{d+1}$ is given by the sign of the determinant

$$\begin{vmatrix} p_{11} & p_{12} & \cdots & p_{1d} & 1 \\ p_{21} & p_{22} & \cdots & p_{2d} & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ p_{d+1,1} & p_{d+1,2} & \cdots & p_{d+1,d} & 1 \end{vmatrix}. \tag{1}$$

The value of this determinant is the signed volume of the simplex spanned by $p_1, \ldots, p_{d+1}$, up to a constant factor depending only on d. If the determinant is zero, then $p_1, \ldots, p_{d+1}$ lie on a flat of dimension less than d. Gaussian elimination or other techniques can be used to evaluate the determinant over the real numbers using $O(d^3)$ real arithmetic operations.

If the orientation test is to be performed repeatedly with points $p_1, \ldots, p_d$ fixed but p_{d+1} varying, it is useful to compute the coefficients of the hyperplane through $p_1, \ldots, p_d$ explicitly. The hyperplane equation is

$$x_1\alpha_1 + \ldots + x_d\alpha_d + \alpha_{d+1} = 0.$$

Formally, the equation can be obtained by replacing $p_{d+1,i}$ by x_i in the matrix of equation 1 above, setting the determinant of the matrix to 0, and then expanding the determinant about the last row; thus the coefficients α_i are determinants of $d \times d$ submatrices of the first d rows of the matrix. It is not hard to see that all the coefficients of the equation can be determined by solving an appropriate system of linear equations, using a total of $O(d^3)$ arithmetic operations. Then for each new point p_{d+1}, only $O(d)$ arithmetic operations are necessary to evaluate the sign of the hyperplane equation.

A variant of the orientation test can be used to decide membership in a sphere. Suppose the sequence of $d{+}1$ points $(p_2, \ldots, p_{d+2})$ in E^d has positive orientation. Then p_1 is outside, on, or inside the circumsphere of $p_2, \ldots, p_{d+2}$ as the sign of the determinant

$$\begin{vmatrix} p_{11} & \cdots & p_{1d} & p_{11}^2 + \ldots + p_{1d}^2 & 1 \\ p_{21} & \cdots & p_{2d} & p_{21}^2 + \ldots + p_{2d}^2 & 1 \\ \vdots & & \vdots & \vdots & \vdots \\ p_{d+2,1} & \cdots & p_{d+2,d} & p_{d+2,1}^2 + \ldots + p_{d+2,d}^2 & 1 \end{vmatrix} \tag{2}$$

is positive, zero, or negative[42, 54]. Evaluation of the sign of this determinant on points $p_1, \ldots, p_{d+2}$ is the *incircle test*; it requires $O(d^3)$ arithmetic operations. As with the orientation test, it is possible to optimize the incircle test if $d{+}1$ of the points are fixed.

The computation of the point of intersection of two lines depends upon the representation of the lines. If line $l_i = \{(x, y) : a_i x + b_i y + c_i = 0\}$ is represented by its coefficients (a_i, b_i, c_i), then the intersection point (x, y) of l_1 and l_2 is

$$(x, y) = \left(\frac{\begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, -\frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}\right). \tag{3}$$

Notice that if l_1 and l_2 are parallel then the denominator determinant is zero. Alternatively a line could be specified by two points on it. If line l_1 passes through points p_1 and p_2 and line l_2 passes through q_1 and q_2, then the point of intersection of lines l_1 and l_2 is given by

$$(x, y) = \left(\frac{\begin{vmatrix} p_{21} - p_{11} & p_{11}p_{22} - p_{12}p_{21} \\ q_{21} - q_{11} & q_{11}q_{22} - q_{12}q_{21} \end{vmatrix}}{\begin{vmatrix} p_{22} - p_{12} & p_{21} - p_{11} \\ q_{22} - q_{12} & q_{21} - q_{11} \end{vmatrix}}, \frac{\begin{vmatrix} p_{22} - p_{21} & p_{11}p_{22} - p_{12}p_{21} \\ q_{22} - q_{21} & q_{11}q_{22} - q_{12}q_{21} \end{vmatrix}}{\begin{vmatrix} p_{22} - p_{12} & p_{21} - p_{11} \\ q_{22} - q_{12} & q_{21} - q_{11} \end{vmatrix}}\right). \tag{4}$$

This equation can be obtained by substituting the line equation coefficients of l_1 and l_2 into equation 3. Similar and more complicated formulae can be used to determine the line of intersection of two planes, the point of intersection of three planes, and so forth.

4.2 Exact arithmetic

Integer arithmetic. Consider an algorithm that uses only the orientation test; an example is the convex hull algorithm discussed in section 3. One way to eliminate real arithmetic in the algorithm is to use integer arithmetic. An input point is represented by a vector of integers; the set of input points may need to be scaled so that the points can be adequately represented by the integer lattice.

A coarse estimate of the bit-complexity of the arithmetic needed to evaluate the orientation test determinant can be obtained by viewing the orientation test as a multivariate polynomial in the coordinates of the input points. In dimension d, the orientation test is degree d. Hence if the point coordinates are represented using b bits, then approximately bd bits are necessary to evaluate the determinant. Thus with standard 32-bit integer arithmetic, in dimension $d = 3$, the input can be specified with at most $b = 10$ bits. For larger b and d, extended-precision integer arithmetic is necessary.

It is reasonable to take the dimension d and the input bit-complexity b as constant, so extended-precision arithmetic contributes only a constant multiplicative overhead to running time. Unfortunately, at least with naively implemented arithmetic, the overhead is large. Consider using Gaussian elimination to evaluate the orientation-test determinant in d dimensions with b-bit integer entries. Gaussian elimination can be implemented over the integers, avoiding division, but the bit-complexity of matrix entries grows during elimination. Rather than the $O(d^3)$ real arithmetic operations for Gaussian elimination over the reals, the number of b-bit arithmetic operations necessary is exponential in d[59].

An alternative to extended-precision integer arithmetic is modular arithmetic over a collection of finite fields[60]. An integer x is represented by its moduli $x \bmod p_1$, $x \bmod p_2$, ..., $x \bmod p_d$, for a collection of relatively prime numbers $p_1, \ldots, p_d$. Addition, subtraction, and multiplication are all performed componentwise; division is possible, though a bit more complicated. The Chinese remainder theorem can be used to reconstruct the determinant from its components mod p_i. If d distinct primes each near 2^b are chosen, then arithmetic of bit length up to approximately bd is possible. This results in $O(d^4)$ b-bit arithmetic operations to evaluate the orientation-test determinant in d dimensions with b-bit entries. For repeated orientation tests, the optimization of computing a hyperplane through d points is possible. Each coefficient α_i is a $d \times d$ determinant with b-bit integer entries; hence α_i is an integer small enough to be represented mod p_i, $i = 1$ to d. Evaluating the hyperplane equation at a new point in modular arithmetic takes $O(d^2)$ b-bit arithmetic operations, and determining the sign from the modular values takes an additional $O(d^2)$ b-bit arithmetic operations. Apparently, the use of modular arithmetic for geometric algorithms has not been reported in the literature; it is not clear how large b and d need to be for modular arithmetic to be appropriate.

Rational arithmetic. Using just integers to specify geometric data is insufficient for more complicated geometric algorithms. An example is polygon intersection. As is clear from equation 4, the point of intersection of two segments has rational coordinates even if the endpoints have integer coordinates. (However, it is possible to tell using only integer arithmetic if two segments with integer endpoints intersect.) A reasonable requirement of a polygon intersection algorithm is that its input and output representations be the same. Thus vertices should be allowed to have rational endpoints, and internal arithmetic computation requires rational arithmetic. Unfortunately, bit-complexity estimates go up dramatically using rational arithmetic. The appropriate measure of the bit complexity of a rational p/q is the sum of the bit-complexities of p and q. Even a single addition can double bit complexity: $p/q + r/s = (ps + qr)/qs$ has bit complexity the sum of the bit complexities of p/q and r/s. If two segments have rational endpoints, say with coordinates r_i/s_i for integer r_i, s_i, $i = 1$ to 8, then each coordinate of the point of intersection of lines through the segments can be expressed as the ratio of two polynomials in the r's and s's. Each polynomial has degree 8; thus intersecting two segments with rational endpoints might multiply bit complexity by a factor of 8.

With suitable engineering, rational arithmetic may be appropriate. Karasick, Lieber, and Nackman[59] implement a divide-and-conquer algorithm for Delaunay triangulations in two dimensions using adaptive-precision rational arithmetic. The input points have rational coordinates. Each rational number is approximated by an interval with low-precision endpoints. The geometric primitives needed by the algorithms are the orientation determinant and the incircle determinant; they are evaluated using interval arithmetic. If the computed interval contains zero, then the computation is repeated using smaller intervals with higher-precision endpoints. Karasick, Lieber, and Nackman report dramatic improvement of the performance of rational arithmetic using this and other optimizations. Before optimization, a rational-arithmetic implementation of the Delaunay triangulation algorithm runs a factor of about 10^4 slower than an implementation using floating point arithmetic; after all optimizations, the implementation is approximately a factor of 4 slower than floating point. However, Karasick, Lieber, and Nackman do not report on any algorithms that involve computed rational points, such as an intersection point of two segments.

Symbolic perturbation schemes. An advantage of exact arithmetic is that there are symbolic perturbation schemes that allow the input to be treated as if it were in general position[34, 93, 94]. Edelsbrunner and Mücke work out the details for one such scheme, which they call Simulation of Simplicity. Let $S = \{s_1, \ldots, s_n\}$ be a set of n points in E^d, with $s_i = (s_{i1}, s_{i2}, \ldots, s_{id})$. Let S' be the perturbed set of points obtained by replacing s_{ij} with $s_{ij} + \epsilon^{2^{id-j}}$, for ϵ very small. For all sufficiently small $\epsilon > 0$, S' is in general position, that is no $d{+}1$ points lie on a common hyperplane. Edelsbrunner and Mücke show how to simulate this perturbation symbolically, without any computation involving ϵ or even knowing ϵ. For an orientation test of a set of perturbed points, they evaluate a sequence of determinants constructed from the coordinates of the unperturbed points. The sign of the first nonzero determinant in the sequence is the same as the sign of the orientation determinant of the perturbed points. The first determinant in the sequence is same as the orientation determinant constructed from the unperturbed points, so overhead is incurred only if it is zero, i.e., the unperturbed points lie on a common hyperplane.

A possible disadvantage of Edelsbrunner and Mücke's scheme is the length of the sequence of determinants. For the orientation test in dimension 3 it is already of length 15. In principle, the determinants can be constructed as needed. Emiris and Canny[38] give a different perturbation scheme which in some circumstances may require less computation to resolve degeneracies. A second disadvantage is that there is no control over the perturbation. For example, if the scheme is used for a convex hull algorithm, then a nonextreme point may be perturbed away from the convex hull, in effect triangulating a facet of the convex hull, or it may be perturbed inside the hull, in effect disappearing. A third apparent disadvantage of the scheme is its limited applicability. Edelsbrunner and Mücke only work out their scheme for the orientation test. Because of the specific structure of the orientation test determinant, the sequence of determinant evaluations to resolve degeneracies is fairly simple. For different geometric primitives, degeneracy resolution may be more complex. Yap[93, 94] gives a scheme applicable to more general geometric algorithms, but it has only been worked out in a fairly abstract setting. A conceptual problem with any perturbation scheme is how to handle degeneracies that are implied mathematically. If point p is the intersection of segments s_1

and s_2, then surely p is collinear with the endpoints of segment s_1. A perturbation scheme is supposed to guarantee that no orientation determinant has value zero, but that is the only mathematically correct value for the orientation of p and the two endpoints of s_1.

4.3 Restricting bit complexity

Rounding. Consider the problem of finding the arrangement of a set of line segments with endpoints of fixed bit complexity. The coordinates of the point of intersection of two segments may have high bit complexity. However, it seems superfluous to store the coordinates exactly, since the segment endpoints have low precision and are probably approximations anyway. An obvious approach is to round the intersection point coordinates to low precision. The difficulty comes from preserving topological information.

Greene and Yao[49] propose one scheme that finds the arrangement of a set of line segments with endpoints on the integer lattice. To make the output uniform, they round the intersection point of two segments to the nearest integer lattice point. A segment involved in intersections is replaced with a path of smaller segments through the intersection points; hence the segment is no longer straight. An obvious approach is to compute the arrangement exactly and then round each intersection point. One problem is that the exact point of intersection of segments a and b could lie to one side of segment c, but the rounded point could lie to the other side of c. Another problem is that multiple intersections could cause a line segment to drift quite far from its original position.

To solve these problems, Greene and Yao require that rounding the exact intersection point of two segments to an integer lattice point not cause the intersection point to cross over another segment s; instead segment s must move out of the way. However, in order to bound the motion of segments, they require that no segment move past an integer lattice point. They suggest visualizing the integer lattice points as pins sticking up from the plane, and a line segment c as a rubber band attached to its endpoints. If the intersection point p of a and b is near c, then rounding p to an integer lattice point $\hat{p}$ may cause $\hat{p}$ to push against c. Segment c stretches and may become pinned against various integer lattice points. However, segment c cannot ever move very far, since it cannot move past an integer lattice point. See figure 15.

Greene and Yao give an algorithm that computes this rounded arrangement from a set of n line segments. The running time of the algorithm is $O(b((n+a)\log n+h))$ where a is the number of intersections among the segments, b is the number of bits needed to specify the endpoints of the line segments (the logarithm of the size of the integer lattice), and h measures how many times a segment was stretched because of a nearby rounded intersection point. (For comparison, the Bentley-Ottman plane-sweep algorithm for line segment intersection using unit-cost exact arithmetic has running time $O((n+a)\log n))$. Usually h should be small, but in the worst case h could be as large as $O(na)$. The Greene-Yao algorithm seems fairly complicated; it has an interesting analysis using Farey sequences to determine how to

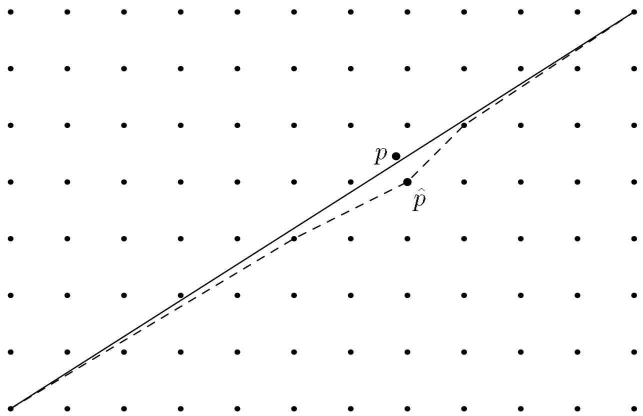


Figure 15: Stretching a line segment; point p is the intersection point of two segments, not shown.

stretch a line segment without moving past an integer lattice point.

Milenkovic[67] suggests a variant of this rounding scheme. Suppose a planar subdivision is given, with vertex locations specified to high precision. Suppose also a rounding function is given that maps high-precision coordinates to low-precision coordinates. The rounding function must be monotonic, that is, it must preserve x- and y- order, though distinct high-precision coordinates may collapse to the same low-precision coordinate. Milenkovic shows how to use the rounding function to round the planar subdivision to low precision while preserving the topology of the subdivision. As in Greene-Yao, line segments should be thought of as rubber bands, possibly bending to stay to the correct side of rounded vertices. Milenkovic's method probably introduces fewer bends because he does not require that segments bend around integer lattice points. His rounding function is also more general; it applies equally well to planar subdivisions specified with floating point, rational or integer coordinates. The running time of his algorithm is $O(n \log n + h)$, where n is the number of edges in the subdivision and h counts the number of nearby pairs of edges and vertices. Again, h is usually small but could be as large as $O(n^2)$.

Milenkovic[69] gives a generalization of the method applicable in higher dimensions. He defines a subdivision in E^3 as a set of triangles so that any intersection of two triangles is either a vertex or an edge of both triangles. Given a subdivision in E^3 with high-precision coordinates and a rounding function, he is able to create a new subdivision with low-precision coordinates. Old triangles may be refined into smaller triangles by the addition of new vertices lying on the triangles. The rounded locations of new vertices added to a triangle may not all lie on a common plane, so in the rounded subdivision a triangle may be replaced with a polyhedral surface. However, no old vertex will move from one side of a triangle to the other side of the resulting polyhedral surface. Milenkovic states that his technique is applicable in arbitrary dimension d. A subdivision is now a set of n simplices so that any two simplices intersect in a common lower-dimensional simplex. To do the rounding, simplices are refined by adding extra vertices. The number of resulting simplices could be as much as $n^{O(2^d)}$, which suggests that his scheme is far from practical.

Both the Greene-Yao method and Milenkovic's method change the input data in the sense that new vertices may be added and line segments may be replaced with polygonal chains. This might be viewed as undesirable, and one might wish to do rounding with as little change to the input as possible. Milenkovic and Nackman[70] give a result that suggests that such rounding is hard. Suppose integers p and k are given with $0 < k < p$. A real a' is a (p,k)-*approximation* to a real number $a \in [-1,1]$ if a' is an integral multiple of 2^{-p} so that $|a - a'| < 2^{k-p}$. Informally, the first $p-k$ bits of a' must agree with a, and the last k bits of a' can be arbitrary. A polygon P' is a (p,k)-*rounding* of a polygon P if P' results by (p,k)-rounding the coefficients of each line equation through an edge of P. Suppose $\mathcal{P}$ is a set of simple polygons so that no two polygons intersect, though polygons may nest. A set of polygons $\mathcal{P}'$ is a (p,k)-*rounding* of $\mathcal{P}$ if $\mathcal{P}'$ is obtained from $\mathcal{P}$ by (p,k)-rounding each polygon in $\mathcal{P}$ while preserving nesting structure, simplicity, and disjointness of polygons. Milenkovic and Nackman show that the problem of determining whether $\mathcal{P}$ has a (p,k)-rounding $\mathcal{P}'$ is NP-complete. The construction is a reduction from the problem of three-coloring planar graphs. Since the gadgets used in the reduction are rather intricate, it is possible that there are heuristics for rounding that are effective in practice, though not provably always applicable.

Double precision geometry. Milenkovic[67] gives a technique for reducing bit complexity that may be appropriate for the computation of line and segment intersections. Consider a set of lines with single-precision floating point coefficients. The point of intersection of two lines can be computed using equation 3. Milenkovic observes that if this computation is done using double-precision floating point arithmetic, each numerator and denominator can be evaluated exactly (some care is needed to scale the floating-point exponents uniformly). Assuming that the division rounds correctly, the result is the point of intersection of the two lines, correctly rounded to double precision. Milenkovic discusses how to use this calculation in line arrangement algorithms; he also gives a more complex procedure to intersect line segments. Schorn[81] gives a related technique that reduces the bit complexity of primitives needed for planar nearest-neighbor calculations.

Restructuring. Sugihara and Iri[92] consider the problem of polyhedral modeling in three dimensions. Polyhedral modeling allows complicated objects to be constructed from primitive polyhedral objects using boolean operations and transformations. A typical primitive object is a cube or a tetrahedron; boolean operations include union, intersection, and set difference; transformations include translation and rotation. Clearly, obtaining an explicit description of a constructed object is difficult from the point of view of precision requirements. Each boolean operation or transformation can increase coordinate bit complexity. Since boolean operations and transformations can be cascaded, bit complexity can grow arbitrarily.

By placing careful constraints on the design of the modeler, Sugihara and Iri are able to bound the bit complexity of geometric primitives. A primitive object is represented by the set of planes through its faces. Transformations can only be applied to primitive objects, which can then be used in boolean operations. Since boolean operations and transformations commute, this does not restrict the class of constructible objects. A transformation is applied to a primitive object by applying the transformation to the plane equations, and then rounding the plane coefficients to fixed bit complexity. The bit complexity of the geometric primitives needed for boolean operations can be bounded because only planes with coefficients of fixed bit complexity are involved in boolean operations. In fact, if plane coefficients are integers, then it suffices to use integer arithmetic with about five times the bit length of the plane coefficients.

The design has some subtle implications. It is not possible to apply a transformation to a constructed object. Instead, the transformation is applied to all the primitive objects and then the boolean operations that constructed the object are performed again. Because of the rounding, the transformation of a primitive object is only approximate, not exact. Hence the transformation applied to the constructed object is also only approximate. In nearly degenerate situations, the original and transformed objects may be rather different from one another. Consider an object with a notch in it so that the dihedral angle between opposite faces of the notch is very small. Since the equations of the faces are rounded independently, the notch may have very different lengths in the original and transformed objects.

Vertex locations need not ever be computed; all computation is done just with plane equations. In fact, vertex locations probably should not be constructed, since they have high bit complexity. It is not feasible to construct a new plane through a set of vertices, since the coefficients of such a plane would have high bit complexity.

4.4 Floating point

It is tempting to use floating point arithmetic to implement the primitives required for geometric algorithms. Floating point arithmetic is fast, widely available, well-supported by programming languages, and has useful engineering features; for example, scaling is automatic. Of course, floating point does not implement exact real arithmetic. A program may fail because a geometric primitive gives an incorrect answer due to rounding error. Milenkovic[65] and Hoffmann, Hopcroft, and Karasick[56] give clear examples of such failure.

Standard error analysis techniques can be applied to geometric primitives[22]. Suppose the input to an algorithm is a set of points. The algorithm can interpret the coordinates of the points as being exact, ignoring any error in the process that generated the points. Consider the orientation determinant for points $s_1, \ldots, s_{d+1}$ in E^d. In floating point arithmetic using, say, Gaussian elimination, it is possible to compute an approximate determinant $\hat{\Delta}$ and an error bound $\hat{\delta}$. If $|\hat{\Delta}| > \delta$, then the sign of Δ is correct. If $|\hat{\Delta}| < \delta$, then the sign of Δ

is unknown, but there are slightly perturbed points $s'_1, \ldots, s'_{d+1}$ so that the true determinant is zero, i.e. the points lie on a common $(d-1)$-flat. The slight perturbation moves points by an amount depending on the accuracy ϵ of floating point arithmetic, the dimension d, and the distance of the points from the origin. If the set of points is in "sufficiently general" position, i.e., no determinant changes sign when the points are slightly perturbed, then the algorithm using the floating-point orientation test will produce the correct answer. Of course, the input points may not be in "sufficiently general" position, and some determinant sign evaluation may give the "can't tell" answer. It can be disastrous to interpret the "can't tell" answer as a zero determinant. As an example, consider points s_1, s_2, s_4 forming a right angle in the plane, and add another point s_3 very near s_2. A naive implementation might take s_1, s_2, s_3 as being collinear, and s_2, s_3, s_4 as being collinear, implying the collinearity of s_1, s_3, s_4, which is manifestly false.

The algorithm may generate intermediate geometric values. Since an intermediate value is computed using floating point arithmetic, it is not exact; interval analysis techniques[73] can be used to estimate error and its effect on subsequent primitive computations. Salesin, Stolfi, and Guibas[80] consider two-dimensional geometric primitives on points, assuming that each point has some uncertainty ϵ. Dobkin and Silver[26] discuss the geometric applications of a "permutation-and-perturbation" technique that estimates a confidence interval for a computed value. Segal[86] discusses a solid modeler implemented with confidence intervals for every computed value. A difficulty with all interval analysis techniques is that they can be very pessimistic. The computed point of intersection of two lines is not the true intersection point, but a point lying very close to both lines; the true point of intersection lies in some diamond-shaped region centered at the computed point. To simplify the interval analysis, the diamond may be replaced by a disk (or box) that contains it. But if the lines are close to parallel, the diamond is long and skinny, and the replacement disk is very large.

Some attempt has been made to apply error analysis techniques to geometric algorithms as a whole, rather than just to the geometric primitives[41, 57, 65, 66]. It is not immediately clear what kind of statement can be made about a geometric algorithm implemented using floating point arithmetic. It is too much to require that the algorithm obtain the correct answer for all inputs, because it may be impossible to distinguish among inputs with distinct outputs using floating point arithmetic. The paradigm of forward error analysis, from numerical analysis, suggests that a geometric algorithm should produce an output that is near the correct output. A difficulty is to determine the meaning of "near." The output of a geometric algorithm includes combinatorial structure, and in general it is not obvious how to say that one combinatorial structure is near another. A somewhat more satisfactory paradigm is backward error analysis: the algorithm should produce output that is the correct output for some data near the true input.

It is possible to give a backward error analysis for some simple geometric algorithms. An example is the problem of computing convex hulls in two dimensions. The input is a set of n points in E^2. The output is a simple polygon, the indices of the points on the boundary of the convex hull in order around the hull. Fortune[41] gives a variant of Graham's algorithm that can be implemented in floating point arithmetic. The algorithm can be applied to any set of points, in general position or not. The correctness claim is that the computed polygon is the true boundary of the convex hull of a slightly perturbed set of points. The perturbation may have to move points that were decided to be inside the hull so that they actually are inside the hull, and similarly may have to move points decided to be on the boundary so that they actually are on the boundary. The perturbation bound is $O(\epsilon)$ where ϵ is the accuracy of floating point arithmetic. This means that each point moves at most by an amount proportional to ϵ times the largest coordinate of any point. Notice that the computed polygon passing through the unperturbed points may not be convex; indeed, it may have self-intersections. Milenkovic and Li[61] give a slightly more complicated variant that guarantees that the output is a convex polygon; points outside the convex polygon may have to be perturbed by $O(\epsilon)$ to lie inside.

For even slightly more complicated geometric algorithms, the paradigm of backward error analysis requires modification. This can be seen by considering the problem of computing the combinatorial arrangement of a set of lines in the plane. There is an incremental algorithm that computes the arrangement by adding lines one by one, threading the new line through the arrangement of the previously inserted lines[12, 35, 30] (see also section 2.1). The algorithm uses a version of the orientation test to decide how the new line exits the current region of the arrangement. If the orientation test is implemented in floating point arithmetic, the answers may not always be reliable, and the algorithm must disambiguate in some fashion. However, there are situations where there is only one way for the line to leave the current region. An example is shown in figure 2. The new line l passes through point 12 and 13, and by Pappus's theorem[77] from projective geometry, l must pass through point 23 as well. Unfortunately, the problem of deciding whether the line is forced to exit through a particular edge or vertex is equivalent to the problem of deciding the truth of a statement in the existential theory of the reals[72, 88]. The exact complexity of the existential theory of the reals is not known, but it is at least NP-hard and it is in PSPACE. It seems extremely inappropriate for a floating-point line arrangement algorithm to have to use a subroutine that must solve NP-hard problems.

The solution in this instance is to relax the requirements on the line arrangement algorithm. Rather than produce a combinatorial arrangement realized by a set of lines, the algorithm produces a combinatorial arrangement that may only be realized by a set of pseudolines (see section 2.1). Thus in figure 2, the algorithm is not forced to decide that l passes through point 23, but may allow it to pass above or below 23.

Milenkovic[66] was the first to analyze a line arrangement algorithm using this approach. The best current analysis is due to Fortune and Milenkovic[44]. They analyze the incremental algorithm and another algorithm based on the sweepline paradigm. For a set of n lines in the plane, the algorithms take time $O(n^2)$ and $O(n^2 \log n)$, respectively. Each line is specified by a triple (a, b, c) of floating point numbers. The output of the algorithms is a combinatorial arrangement. The correctness assertion for either algorithm is that the combinatorial arrangement is realized by a set of pseudolines approximating the lines. Each pseudoline deviates from its corresponding line by a relative distance of at most $O(n\epsilon)$, where ϵ is the accuracy of floating point. "Relative distance" allows a point on the pseudoline to deviate from the line by an amount proportional to the distance of the point from the origin. The proof of the existence of nearby pseudolines depends upon a careful analysis of the floating-point computation of the intersection point of two lines. Given two lines l, l' with floating point coefficients, it is possible to compute an approximate intersection point p that lies very close to both lines, though p can be arbitrarily far from the true point of intersection if l and l' are close to being parallel. However, pseudolines approximating l and l' can intersect at p without deviating much from l and l', respectively. The annoying multiplicative factor of n in the error bound arises because in the worst case, the deviation of pseudolines from lines can cascade. Fortune and Milenkovic are able to devise a different line arrangement algorithm with running time $O(n^2 \log n)$ and relative error bound $O(\epsilon)$, independent of the number n of lines.

Sugihara and Iri[90, 91] consider the problem of computing the Voronoi diagram of a set of point sites in the plane using floating point arithmetic. They show how to implement the incremental algorithm to guarantee the following combinatorial consistency property: the computed graph approximating the Voronoi diagram is always a planar graph with one region per site. When a new site is inserted, a portion of the old Voronoi diagram is deleted. The consistency property can be maintaining by requiring that the deleted portion be a tree. The deleted tree is always replaced with a cycle around the new site. Fortune[43] gives numeric bounds on the accuracy of Delaunay triangulation and Voronoi diagram algorithms in the plane. The true Delaunay triangulation of a set of point sites is characterized by the property that the circumcircle of every triangle has no sites in its interior. In an "approximate Delaunay triangulation" each triangle has approximately empty circumcircles. Fortune shows that various algorithms produce approximate Delaunay triangulations so that no site is more than a relative amount $O(n\epsilon)$ inside the circumcircle of any triangle. From an approximate Delaunay triangulation, an approximate Voronoi diagram can be constructed. This is a planar graph in which a point location algorithm can be used to answer nearest-neighbor queries. If the point location algorithm reports that a query point q is in the region of a Voronoi site s, then s may not be the true closest site, but any other site is closer only by a relative amount proportional to $O(n\epsilon)$.

Hoffmann, Hopcroft, and Karasick[56, 58] consider set operations on polyhedral solids using floating point arithmetic. They outline a procedure for intersecting the bounding "shells" of polyhedral solids in order to minimize the effect of numerical uncertainty. They give an elaborate hierarchy of tests for deciding vertex-vertex, edge-edge, and face-face incidences. They do not know how to analyze their procedure completely. They give an example of intersecting a solid with a slightly rotated copy of itself. For a very narrow range of rotation angles, their procedure fails. However, the failure range is much smaller than the failure range of other procedures that have been suggested.

4.5 Discussion

The techniques discussed above are clearly not adequate for many problems of interest. It is easy to find examples where computed coordinates are not rational numbers. For example, the vertices of the Voronoi diagram of a polygon (also called the medial axis of the polygon) in general are algebraic rather than rational numbers. Exact arithmetic on such numbers is in principle possible by representing them symbolically by their minimal defining polynomials, but for a problem of any size the required overhead would be overwhelming. More generally, operations involving curves and surfaces in two and three dimensions can be performed exactly, but polynomials of high degree and high coefficient bit-complexity are required. The use of floating point arithmetic on such problems is not very well understood at all.

It is possible to give some advice about the implementation of geometric algorithms, at least relatively simple algorithms. A fundamental decision is how to implement arithmetic. This decision will affect almost every aspect of an implementation, from the low-level details of the geometric primitives to the choice of high-level optimizations.

One choice to consider is exact arithmetic. This is most appropriate if all data can be represented using integers. Careful examination of the geometric primitives will give a bound on the bit complexity; often the primitives can be altered to lower bit complexity. As the example of the Sugihara-Iri modeler shows, perhaps bit complexity can be reduced only through considerable reorganization. It is likely that a software implementation of high-precision arithmetic has to be carefully tuned in order to get good performance.

A second choice is floating point arithmetic. A minimal requirement for reliable implementation is that each geometric primitive indicate whether or not its answer is reliable. If only reliable answers are given, then the output of the algorithm is correct. If some unreliable answer is given, then sometimes simple tricks are appropriate. For example, if the input is a set of points, then a second execution of the algorithm may be successful if the points are slightly perturbed. Alternatively, in a specific situation the unreliable answer might be disambiguated through further tests, or localized use of higher precision.

It is possible to structure geometric algorithms implemented using floating point to improve reliability. One technique is to separate combinatorial primitives and geometric primitives carefully. Only combinatorial operations that preserve combinatorial structure should be allowed, no matter what results the geometric primitives give [56, 90, 91]. For example, if the output of the algorithm is a planar graph, then all graph operations should preserve planarity. A second technique is to evaluate the geometric primitives using numerical techniques that are as stable as possible. It is possible that standard numerical techniques can be improved for geometric applications[43]. A third technique is to filter output, so that if it is nonsensical, at least some indication is given.

The state of the art is that, except for a few algorithms such as the ones mentioned earlier, completely reliable floating point implementation is not possible. However, implementations are often not completely reliable anyway, for reasons having nothing to do with the implementation of arithmetic.

Acknowledgements. I would like to thank Ken Clarkson, Rex Dwyer, John Hobby, Bob Melville, Victor Milenkovic, Gordon Wilfong, Margaret Wright, and Chris Van Wyk for helpful comments.

References

[1] P. Agarwal, Intersection and decomposition algorithms for planar arrangements, Cambridge U. Press, 1991.

[2] P. Agarwal, M. van Kreveld, M.H. Overmars, Intersection queries for curved objects, *Proc. Seventh Ann. Symp. Comp. Geom.* 41–50, 1991.

[3] N. Alon, The number of polytopes, configurations, and real matroids. *Mathematika* **33**(1):62–71, 1986.

[4] B. Aronov, J. Matoušek, M. Sharir, On the sum of squares of cell complexities in hyperplane arrangements, *Proc. Seventh Ann. Symp. Comp. Geom.* 307–313, 1991.

[5] B. Aronov, M. Pellegrini, M. Sharir, On the zone of a surface in a hyperplane arrangement, manuscript, 1991 (to appear in *Disc. and Comp. Geom.*)

[6] B. Aronov, M. Sharir, Triangles in space or building (and analyzing) castles in the air, *Combinatorica* **10**: 137–173, 1990.

[7] F. Aurenhammer, Voronoi diagrams – a survey of a fundamental geometric data structure, *ACM Computing Surveys* **23**:345–405, 1991.

[8] J.-D. Boissonnat, O. Devillers, R. Schott, M. Teillaud, M. Yvinec, Applications of random sampling to online algorithms in computational geometry, research report 1285, INRIA-Sophia Antipolis, 1990.

[9] A. Bjoerner, M. Las Vergnas, B. Sturmfels, N. White, G. Ziegler, *Oriented Matroids*, Cambridge University Press, to appear, 1992.

[10] E. Brisson, Representing geometric structures in d dimensions: topology and order, *Proc. Fifth Ann. Symp. Comb. Geom.*, 218–227, 1989.

[11] A. Brøndsted, *An Introduction to Convex Polytopes*, Springer-Verlag, Berlin, 1983.

[12] B. Chazelle, L.J. Guibas, D.T. Lee, The power of geometric duality, *Proc. 24th Annual Symp. Found. Comp. Science* 217–225, 1983.

[13] B. Chazelle, An optimal convex hull algorithm and new results on cuttings, *Proc. 32nd Ann. Symp. Found. Comp. Sci.* 29–38, 1991.

[14] B. Chazelle, H. Edelsbrunner, L.J. Guibas, M. Sharir, Lines in space – combinatorics, algorithms, and applications. *Proc. 21st Ann. Symp. Theory Comp.* 382–393, 1989.

[15] K.L. Clarkson, A probabilistic algorithm for the post office problem, *Proc. 17th Ann. Symp. Theory Comp.* 175–184, 1985.

[16] K.L. Clarkson, New applications of random sampling in computational geometry, *Disc. Comp. Geom.* **2**:195–222, 1987.

[17] K.L. Clarkson, Randomized geometric algorithms, in *Computing in Euclidean Geometry*, D. Du, F. Hwang, ed., World Scientific Publishing Co., to appear.

[18] K.L. Clarkson, R. Cole, R. Tarjan, Randomized parallel algorithms for trapezoidal diagrams, *Proc. Seventh Ann. Symp. Comp. Geom.* 152–161, 1991.

[19] K.L. Clarkson, H. Edelsbrunner, L. J. Guibas, M. Sharir, E. Welzl, Combinatorial complexity bounds for arrangements of curves and spheres, *Disc. Comp. Geom.* **5**:99–160, 1990.

[20] K.L. Clarkson, K. Mehlhorn, R. Seidel, Four results on randomized incremental constructions, to appear, *Symp. Theor. Aspects of Comp. Sci.*, 1992.

[21] K. Clarkson, P. Shor, Applications of random sampling in computational geometry, II. *Disc. Comp. Geom.*, **4**:387–421 (1989).

[22] G. Dahlquist, Ȧ. Björck, N. Anderson, *Numerical Methods*, Prentice-Hall, 1974.

[23] A.M. Day, The implementation of an algorithm to find the convex hull of a set of three-dimensional points, *ACM Trans. Graphics* **9**(1):105–132, 1990.

[24] T.J. Dekker, A floating-point technique for extending the available precision. *Numer. Math.* **18**:224–242 (1971).

[25] L. Devroye, How to reduce the average complexity of convex hull finding algorithms, *Computers and Mathematics with Applications* **7**:229–308, 1981.

[26] D. Dobkin, D. Silver, Recipes for geometry and numerical analysis–part I: an empirical study *Proc. Fourth Ann. Symp. Comp. Geom.* 93–105, 1988.

[27] D. Dobkin, D. Souvaine, Computational geometry in a curved world, *Algorithmica* **5**(3):421–457, 1990.

[28] R. A. Dwyer, Kinder, gentler average-case analysis for convex hulls and maximal vectors, *SIGACT News* **21**(2)64–71, 1990.

[29] R. A. Dwyer, Higher-dimensional Voronoi diagrams in linear expected time, *Disc. and Comp. Geom.*, **6**:343-367, 1991.

[30] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[31] H. Edelsbrunner, An acyclicity theorem for cell complexes in d dimensions, *Proc. Fifth Ann. Symp. Comp. Geom.* 145–151.

[32] H. Edelsbrunner, L.J. Guibas, M. Sharir, The complexity and construction of many faces in arrangements of lines and segments, *Disc. Comp. Geom.* **5**:161–196, 1990.

[33] H. Edelsbrunner, L.J. Guibas, M. Sharir, The complexity of many cells in arrangements of planes and related problems, *Disc. Comp. Geom.* **5**:197–216, 1990.

[34] H. Edelsbrunner, E. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graphics* **9**(1):66-104, 1990.

[35] H. Edelsbrunner, J. O'Rourke, R. Seidel, Constructing arrangements of lines and hyperplanes with applications, *SIAM J. Computing* **15**:341–363, 1986.

[36] H. Edelsbrunner, R. Seidel, Voronoi diagrams and arrangements, *Disc. and Comp. Geom.* **8**(1):25–44, 1986.

[37] H. Edelsbrunner, R. Seidel. M. Sharir, On the zone theorem for hyperplane arrangements, manuscript, 1990.

[38] I. Emiris, J. Canny, A general approach to removing degeneracies, *Proc. 32nd Annual Symp. on Found. of Comp. Sci.* 405–413, 1991.

[39] A. R. Forrest, Computational geometry in practice, *Fundamental Algorithms for Computer Graphics*, 707–724, R.A. Earnshaw, ed., NATO ASI Series, Springer-Verlag, 1985.

[40] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica* **2**:153-174 (1987).

[41] S. Fortune. Stable maintenance of point-set triangulation in two dimensions, manuscript, AT&T Bell Laboratories. An abbreviated version appeared in *Proc. 30th Annual Symp. Found. Comp. Sci.* 494–499, 1989.

[42] S. Fortune, Voronoi diagrams and Delaunay triangulations, in *Computing in Euclidean Geometry*, D. Du, F. Hwang, eds., World Scientific Publishing Co., to appear.

[43] S. Fortune, Numerical stability of algorithms for Delaunay triangulations and Voronoi diagrams in two dimensions, manuscript, in preparation, AT&T Bell Labs, 1992.

[44] S. Fortune, V. Milenkovic. Numerical stability of algorithms for line arrangements. *Proc. Seventh Ann. Symp. Comp. Geom.*, 334–341, 1991.

[45] J.E. Goodman, R. Pollack, Upper bounds for configurations and polytopes in R^d, *Disc. Comp. Geom.* **1**:219-227, 1986.

[46] M. Golin, R. Sedgewick, Analysis of a simple yet efficient convex hull algorithm, *Proc. Fourth Ann. Symp. Comp. Geom.*, 153–163, 1988.

[47] R. Graham, An efficient algorithm for determining the convex hull of a finite planar set, *Inf. Proc. Letters* **1**:132–133, 1972.

[48] P.J. Green, R. Sibson, Computing Dirichlet tessellations in the plane, *Computer Journal*, **21**(22):168-173, 1977.

[49] D. Greene, F. Yao, Finite-resolution computational geometry, *Proc. 27th Ann. Symp. Found. Comp. Science*, 143–152, 1986.

[50] B. Grunbaum, *Convex Polytopes*, John Wiley and Sons, New York, 1967.

[51] B. Grunbaum. *Arrangement and Spreads*, Conference Board of the Mathematical Sciences Regional Conference Series in Mathematics, No. 10, American Mathematical Society, 1972.

[52] L.J. Guibas, D.E. Knuth, M. Sharir, Randomized incremental construction of Delaunay and Voronoi diagrams, *ICALP 90*, 414–431, Springer-Verlag, Berlin, 1990.

[53] L.J. Guibas, M. Sharir, Combinatorics and algorithms in arrangements, *New Trends in Discrete and Computational Geometry*, J. Pach, ed., Springer-Verlag, to appear.

[54] L.J. Guibas, J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Trans. Graphics* **4**(2):74-123, 1985.

[55] C. Hoffmann, The problems of accuracy and robustness in geometric computation. *Computer* **22**:31–42 (1989).

[56] C. Hoffmann, J. Hopcroft, M. Karasick, Robust set operations on polyhedral solids, *IEEE Comp. Graph. Appl.* **9**(6):50–59, 1989.

[57] C. Hoffmann, J. Hopcroft, M. Karasick, Towards implementing robust geometric computations, *Proc. Fourth Ann. Symp. Comp. Geom*, 106–117, 1988.

[58] M. Karasick, On the representation and manipulation of rigid solids, Ph.D. thesis, McGill U., 1988.

[59] M. Karasick, D. Lieber, L. Nackman, Efficient Delaunay triangulation using rational arithmetic, *ACM Trans. Graphics* **10**(1):71–91, 1990.

[60] D. E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, Addison-Wesley, 1969.

[61] Z. Li, V. Milenkovic, Constructing strongly convex hulls using exact or rounded arithmetic, *Proc. Sixth Ann. Symp. Comp. Geom.* 235–243, 1990.

[62] P. Lienhardt, Subdivisions of n-dimensional spaces and n-dimensional generalized maps, *Proc. Fifth Ann. Symp. Comp. Geom.*, 218–227, 1989.

[63] J. J. Matoušek, Efficient partition trees, *Proc. Seventh Ann. Symp. Comp. Geom.* 1–9, 1991.

[64] P. McMullen, G.C. Shephard, *Convex Polytopes and the Upper Bound Conjecture*, Cambridge University Press, 1971.

[65] Victor Milenkovic, Verifiable implementations of geometric algorithms using finite precision arithmetic. *Artificial Intelligence*, **37**:377–401, 1988.

[66] V. Milenkovic, *Verifiable Implementations of Geometric Algorithms using Finite Precision Arithmetic*. Ph.D. Thesis, Carnegie-Mellon, 1988. Technical Report CMU-CS-88-168, Carnegie Mellon University, 1988.

[67] V. Milenkovic, Rounding face lattices in the plane, *1st Can. Conf. Comp. Geom.*, 1989.

[68] V. Milenkovic, Double precision geometry: a general technique for calculating line and segment intersections using rounded arithmetic, *Proc. 30th Ann. Symp. Found. Comp. Sci* 500–505, 1989.

[69] V. Milenkovic, Rounding face lattices in d dimensions, Abstract, *Second Can. Comp. Geom. Conf.*, 1990.

[70] V. Milenkovic, L. Nackman, Finding compact coordinate representations for polygons and polyhedra, *IBM J. Res. Dev.* **34**(5):752–768, 1990. A version also appeared in *Proc. Sixth Ann. Symp. Comp. Geom.* 244–252, 1990.

[71] M. Minsky, S.Papert, *Perceptrons*, MIT Press, 1969.

[72] N.E. Mnev, The Universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In O.Y. Viro, ed., *Topology and Geometry - Rohlin Seminar*, LN Mathematics, 1346, pp. 527–544, Springer-Verlag, 1989.

[73] R. E. Moore, *Methods and Applications of Interval Analysis*, SIAM, 1979.

[74] K. Mulmuley, A fast planar partition algorithm: part I, *Proc. 29th Symp. Found. Comp. Sci.*, 580–589, 1988.

[75] K. Mulmuley, A fast planar partition algorithm: part II, *Proc. Fifth Ann. Symp. Comp. Geom.*, 33–43, 1989.

[76] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford U. Press, 1987.

[77] D. Pedoe, *Geometry: a Comprehensive Course*, Cambridge U. Press, 1970, available from Dover Publications, 1988.

[78] F.P. Preparata, S.J. Hong, Convex hulls of points sets in two and three dimensions, *CACM* **2**(20):87–93, 1977.

[79] F.P. Preparata, M.I. Shamos, *Computational Geometry*, Springer-Verlag, 1985.

[80] D. Salesin, J. Stolfi, L. Guibas, Epsilon geometry: building robust algorithms from imprecise calculations, *Proc. Fifth Ann. Symp. Comp. Geom.*, 208–217, 1989.

[81] P. Schorn, *Robust Algorithms in a Program Library for Geometric Computations*, Ph.D. Thesis, ETH, 1991.

[82] R. Schneider, Random approximation of convex sets, *J. Microscopy*, **151**(3):211–227, 1988.

[83] J.T. Schwartz, C.K. Yap, *Algorithmic and Geometric Aspects of Robotics, Vol. 1*, Erlbaum, 1987.

[84] R. Seidel, Linear programming and convex hulls made easy, *Proc. Sixth Ann. Symp. Comp. Geom.* 211–215, 1990.

[85] R. Seidel, Backwards analysis of randomized geometric algorithms, *New Trends in Discrete and Computational Geometry*, J. Pach, ed., Springer-Verlag, to appear.

[86] M. Segal, Using tolerances to guarantee valid polyhedral modeling results, *SIGGRAPH '90 Conf. Proc., Comp. Graphics.* **24**(4) 105–114, 1990.

[87] M. Shamos, D. Hoey. Closest-point problems*Proc. 16th Ann. Symp. Found. Comp. Science*, 151–162, 1975.

[88] P.W. Shor, Stretchability of pseudoline arrangements is NP-hard, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 4, Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, 531–554, P. Gritzman, B. Sturmfels, eds., AMS, 1991.

[89] J. Stolfi, Oriented projective geometry, *Proc. Third ACM Symp. Comp. Geom.* 76–85, 1987.

[90] K. Sugihara, M. Iri, Geometric algorithms in finite-precision arithmetic, Research Memorandum RMI 88-10, University of Tokyo, September, 1988.

[91] K. Sugihara, M. Iri, Construction of the Voronoi diagram for one million generators in single precision arithmetic, *First Can. Conf. Comp. Geom.*, 1989.

[92] K. Sugihara, M. Iri, A solid modeling system free from topological inconsistency, *J. Information Processing, Information Processing Society of Japan* **12**(4):380–393, 1989.

[93] C. Yap, Symbolic treatment of geometric degeneracies, *J. Symb. Comp.* **10**:349-370, 1990.

[94] C. Yap, A geometric consistency theorem for a symbolic perturbation scheme *J. Comp. Sys. Sci.* **40**(1):2-18, 1990.