VORONOI DIAGRAMS
and
DELAUNAY TRIANGULATIONS

STEVEN FORTUNE
AT&T Bell Laboratories, 600 Mountain Avenue
Murray Hill, NJ 07974, USA

ABSTRACT

The Voronoi diagram is a fundamental structure in computational geometry and arises naturally in many different fields. This chapter surveys properties of the Voronoi diagram and its geometric dual, the Delaunay triangulation. The emphasis is on practical algorithms for the construction of Voronoi diagrams.

1 Introduction

Let S be a set of n points in d-dimensional euclidean space E^d. The points of S are called *sites*. The *Voronoi diagram* of S splits E^d into regions with one region for each site, so that the points in the region for site $s \in S$ are closer to s than to any other site in S.

The *Delaunay triangulation* of S is the unique triangulation of S so that there are no elements of S inside the circumsphere of any triangle. Here 'triangulation' is extended from the planar usage to arbitrary dimension: a triangulation decomposes the convex hull of S into simplices using elements of S as vertices. The existence and uniqueness of the Delaunay triangulation are perhaps not obvious, but these properties follow easily since it is the dual of the Voronoi diagram.

The Voronoi diagram and Delaunay triangulation of the same 250 points appear side by side in figure 1. The points are chosen uniformly in the unit square.

The Voronoi diagram is ubiquitous, arising independently in many different fields. The eponymous Voronoi in 1908 gave a careful definition of the structure, motivated by the study of quadratic forms as initiated by Gauss and Dirichlet. Crystalographers including Delaunay studied the problem of filling space with congruent copies of a set of crystals, starting in the 1920s. Metallurgists call Voronoi regions Wigner-Seitz zones after Wigner and Seitz, who in 1933 used them to study equilibrium properties of alloys. A name in geography for Voronoi regions is Thiessen polygons, following Thiessen who in 1911 used them to improve the estimation of precipitation. Blum suggested Voronoi diagrams as a descriptor for the shape of a set of objects in 1967, hence the use of the Blum transform in pattern recognition.

More recently, Voronoi diagrams have become a central subject in computational geometry. The motivating problem is nearest-site search. Given a fixed set of point sites, this problem is to be able to answer repeated queries of the form "which site is closest to point q?" This problem can be solved by computing the Voronoi

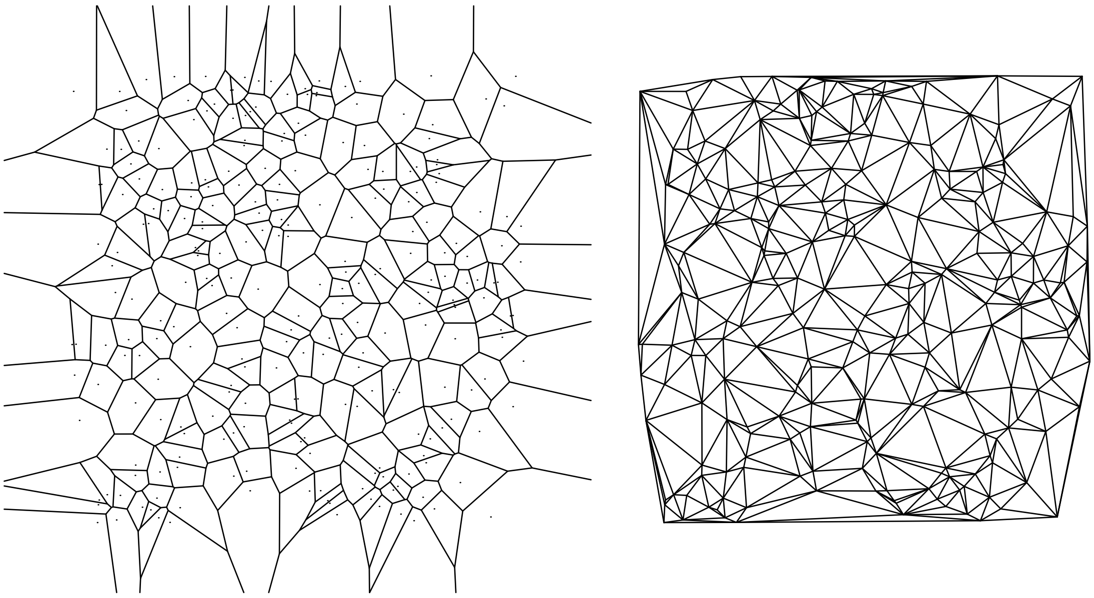


Figure 1: The Voronoi diagram (on the left) and the Delaunay triangulation (on the right).

diagram of the set of sites, and for each query locating the region of the Voronoi diagram containing the query point.

In an influential early paper, Shamos and Hoey[59] sketched a divide-and-conquer algorithm to compute the Voronoi diagram of a set of n planar points in time $O(n \log n)$. Independently, Green and Sibson[33] gave an incremental algorithm for the same problem with $O(n^{3/2})$ expected running time. The incremental algorithm was generalized to arbitrary dimension by Bowyer[8] and Watson.[61] Brown[11] observed that a Voronoi diagram in dimension d can be obtained from an appropriate convex hull in dimension $d+1$, so any convex hull algorithm can be used for Delaunay triangulations. Subsequently, algorithms were obtained for an amazing collection of generalizations of the Voronoi diagram, too numerous to mention except briefly. An easy generalization is to change the metric, either uniformly, for example by using a general L_p metric or a convex distance function, or locally, for example by additive or multiplicative weights on sites. Another generalization is to change the shape of sites; circles and line segments have been considered. 'Higher order' Voronoi diagrams result by partitioning space on the basis of the k nearest neighbors, for some fixed k; the 'furthest point' Voronoi diagram partitions space on the basis of the nth closest site, i.e., the furthest site. 'Constrained' Delaunay triangulations allow some edges to be specified before the triangulation is constructed. 'Geodesic' Voronoi diagrams are defined inside a polygonal environment, where the distance between two points is given by the length of the shortest path connecting the two points lying entirely inside the polygonal environment. 'Abstract' Voronoi diagrams can be used to study Voronoi diagram algorithms given only a set of axioms describing the behavior of the diagram.

Aurenhammer[4] gives a very inclusive survey of Voronoi diagrams, with more than 200 references. He summarizes the early history of Voronoi diagrams, gives many of the applications in mathematics and the natural sciences, and outlines many of the generalizations of Voronoi diagrams and algorithms for constructing them. References for all the topics just mentioned can be found in his paper.

This chapter also surveys Voronoi diagrams and Delaunay triangulations. The focus is narrow; the only topic is sets of point sites with the euclidean metric in E^d. This is the most important case for applications. This chapter gives a careful definition of the Voronoi diagram and the Delaunay triangulation and discusses elementary properties.

The emphasis of this chapter is on algorithms. One goal is to demonstrate that there are usable and efficient algorithms for computing Delaunay triangulations, both in two dimensions and in higher dimension. For, say, a set of points chosen from a uniform distribution inside a sphere, the combinatorial complexity of the Delaunay triangulation grows linearly with the number of sites. The random-incremental algorithm discussed below can compute the Delaunay triangulation of such a set of sites in time $O(n \log n)$, and the running time constant is small. Hence it is perfectly feasible to compute the Delaunay triangulation of "large" sets of points, say 10^5 or 10^6 sites in dimension two. A caveat is that the combinatorial complexity of the Delaunay triangulation grows exponentially with dimension. Hence the running time of any algorithm also grows exponentially with dimension. For small dimension, say dimension less than five, the Delaunay triangulation is still a useful tool.

Four fundamental algorithms are discussed in some detail. These are the flipping algorithm, the incremental algorithm, the random incremental algorithm, and the plane-sweep algorithm. These algorithms are fundamental because they are both theoretically important and reasonable implementation choices. The flipping algorithm is the simplest imaginable algorithm for computing the Delaunay triangulation in two dimensions. It starts with an arbitrary triangulation of the set of points and produces the Delaunay triangulation by a sequence of local modifications. The incremental algorithm is slightly more complicated. Sites are added one by one, with the Delaunay triangulation updated to include the new site after each addition. The advantage of the incremental algorithm is that it generalizes readily to higher dimension. The random incremental algorithm a recent innovation. It is the incremental algorithm with the insertion order chosen at random with all permutations equally likely. The random insertion order guarantees good worst-case performance. The plane-sweep algorithm is based on the plane-sweep paradigm. It provides a contrast because its analysis is based on different principles from the others.

All of the algorithms have been implemented; at least one has been used fairly extensively. The implementation of these algorithms is neither trivial nor excessively complicated. Some attention is given in the chapter to the geometric data structures and geometric primitives required to implement the algorithms, as well as implementation perils. A brief empirical comparison of the algorithms is included.

The background required for this chapter is a basic familiarity with algorithms and with geometry. Exposure to computational geometry is certainly helpful but is not essential. Standard references in computational geometry are books by Edelsbunner[24] and by Preparata and Shamos.[53] Many concepts from the theory of convex polyhedra are used throughout the chapter. Appendix 1 defines the terms that are used; it is adequate for reference and review but is certainly inadequate as a primer. Standard references on convex polyhedra are books by Grunbaum[34] and Brøndsted.[10]

The table of contents is given below. Section 2 defines the Voronoi diagram and the Delaunay triangulation and gives basic properties. One such property is a connection with convex polyhedra in dimension one higher. This connection is fundamental to the analysis of many of the algorithms and is used throughout the chapter. Section 3 of the chapter is an eclectic enumeration of results known about the Voronoi diagram and Delaunay triangulation. For example, the Delaunay triangulation has 'optimality' properties that makes it attractive for use in finite element analysis. The bulk of the chapter is the section 4 on algorithms; it is independent of section 3.

Contents

2 Definition of the Voronoi diagram and Delaunay triangulation

The Voronoi diagram of a set of sites partitions euclidean space E^d on the basis of the site closest to a particular point. To handle the cases where a point is closest to several equidistant sites, the Voronoi diagram is best defined as a cell complex. A *cell* is the intersection of a finite number of hyperplanes and open halfspaces, and a *cell complex* is a finite collection of pairwise disjoint cells so that every face of every cell is in the collection. Similarly, the Delaunay triangulation is a cell complex decomposing the convex hull of the set of sites and having the sites as vertices.

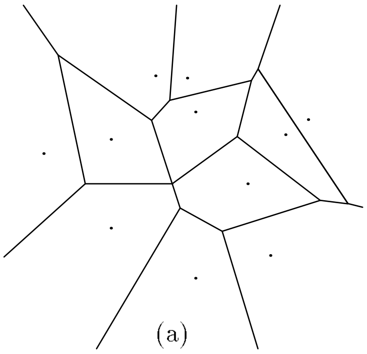
The next section gives a careful definition of the Voronoi diagram and Delaunay triangulation as cell complexes and discusses elementary properties of these two structures. Section 2.2 outlines a fundamental connection between Delaunay triangulations in E^d and convex hulls in E^{d+1}, and between Voronoi diagrams in E^d and intersection of halfspaces in E^{d+1}. The topic of section 2.3 is the combinatorial complexity of a Voronoi diagram, that is, the number of cells in the Voronoi diagram, as a function of the number of sites. Known upper and lower bounds are given, both in the worst case and in the 'typical' case.

2.1 Voronoi diagrams and Delaunay triangulations

Let S be a set of n point sites in E^d. For a nonempty subset R of S, the *Voronoi cell* of R, $V(R)$, is the set of all points of E^d equidistant from all sites in R and closer to every site of R than to any site not in R. Clearly, for a singleton set $R = \{r\}$, $V(R)$ is the set of all points strictly closer to r than to any other site. Any point of E^d lies in $V(R')$ for some $R' \subset S$, though R' may have more than one element. For $R \subseteq S$, $V(R)$ may be empty, either because there is no point equidistant from all $r \in R$ or because any point equidistant from all $r \in R$ is also equidistant from some $r' \in S - R$. The *Voronoi diagram* V is the collection of all nonempty Voronoi cells $V(R)$, for R a subset of S.

Figure 2(a) depicts a Voronoi diagram of a set S of 11 sites in two dimensions. For each site there is an open two-dimensional cell consisting of the points in the plane for which the site is the strict closest among all the sites. Each such cell has a boundary consisting alternately of Voronoi edges (one-dimensional cells) and Voronoi vertices (zero-dimensional cells). Voronoi edges are equidistant from two sites and Voronoi vertices are equidistant from at least three sites. The unbounded Voronoi cells are exactly the cells corresponding to sites on the convex hull of S. Figure 3(a) depicts the Voronoi diagram of a set S of 7 sites in three dimensions. The boundary of each 3-cell consists of facets (2-cells), edges, and vertices.

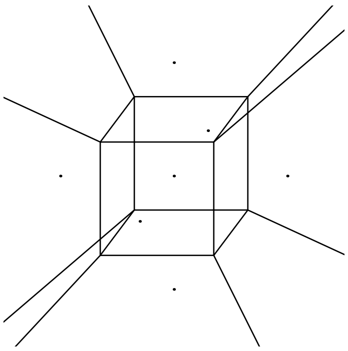
If $R \subset S$ and $V(R)$ is a nonempty Voronoi cell, then the *Delaunay cell* $D(R)$ is cell(R) (the relative interior of the convex hull of R). The *Delaunay triangulation* D is the collection of Delaunay cells $D(R)$, where R varies over subsets of S with $V(R)$ nonempty. An immediate property of the Delaunay triangulation is that it has *empty circumspheres*: for $R \subseteq S$, $D(R)$ is a Delaunay cell exactly if there is a circumsphere of R that contains no site of $S-R$ in its interior. Such a circumsphere



(a)

(b)

Figure 2: (a) A Voronoi diagram in dimension 2; (b) the Delaunay triangulation of the same sites.



(a)

(b)

Figure 3: (a) A Voronoi diagram in dimension 3; (b) the Delaunay triangulation of the same sites.

can be obtained with center any point in the Voronoi cell $V(R)$, so the circumsphere is unique exactly if $V(R)$ is a vertex.

Figure 2(b) gives the two-dimensional Delaunay triangulation of the same sites as in figure 2(a). Notice the correspondence between a Delaunay 2-cell and a Voronoi vertex (the cell need not contain the vertex), between a Delaunay edge and a Voronoi edge, and between a Delaunay vertex, i.e. a site, and a Voronoi 2-cell. In figure 2(b) one of the Delaunay 2-cells is a quadrilateral with four sites as vertices; this is because the four sites are cocircular. Thus the Delaunay triangulation need not be a proper triangulation, i.e., its cells need not all be simplices. The Delaunay quadrilateral corresponds to the vertex of degree four in the Voronoi diagram in figure 2(a). Figure 3(b) gives the three-dimensional Delaunay triangulation of the same sites as in figure 3(a). The correspondence is now between a Delaunay 3-cell and a Voronoi vertex, between a Delaunay facet and a Voronoi edge, between a Delaunay edge and a Voronoi facet, and between a site and a Voronoi 3-cell.

The following theorem gives fundamental properties of the Delaunay triangulation and the Voronoi diagram. One proof can be obtained using standard properties of convex polyhedra together with the connection to convex polyhedra discussed in the next section. A direct proof, possibly more illuminating, is sketched in Appendix 2.

Theorem 2.1 *Let S be a set of n points in E^d with Voronoi diagram V and Delaunay triangulation D.*

1. *V is a cell complex that partitions E^d.*
2. *D is a triangulation of S.*
3. *V and D are dual, that is, for $R, R' \subseteq S$, $V(R)$ is a face of $V(R')$ iff $D(R')$ is a face of $D(R)$.*
4. *If $R \subseteq S$, $V(R)$ is unbounded iff every site of R is on the boundary of the convex hull of S.*

A nonempty Voronoi cell $V(R)$ has the same dimension as the flat of points equidistant from all sites in R, since it is a relatively open subset of the flat. It is not hard to check that the flat has dimension d minus the dimension of the convex hull of R. Hence the dimensions of $V(R)$ and $D(R)$ always sum to d. Since the dimension of $D(R)$ is at most $|R|-1$, the dimension of $V(R)$ is at least $d+1-|R|$.

If $V(R)$ is not empty and the dimension of $V(R)$ exceeds $d+1-|R|$; then $V(R)$ and $D(R)$ are *degenerate*; this happens only if the sites of R lie on a common sphere and also on a common flat of dimension less than $|R|-1$. If $|R| = 1, 2, 3$, then $V(R)$ and $D(R)$ are never degenerate: this is obvious for $|R| = 1$ and $|R| = 2$, and for $|R| = 3$ follows from the observation that three sites cannot be both collinear and cospherical.

In low dimensions, degenerate Voronoi cells can be enumerated explicitly. In dimension 2, the only degenerate Voronoi cell is a vertex that is equidistant from four or more sites, as in Figure 2. The corresponding Delaunay cell is the interior of the convex hull of the sites. In dimension 3, a Voronoi vertex can be equidistant from five or more sites, and the corresponding Delaunay cell is the interior of a convex polyhedron. A second degeneracy in dimension 3 is a Voronoi edge equidistant from four or more sites. In this case the four sites are cocircular and also lie on a common plane, so the Delaunay cell is the interior of a circularly-inscribed polygon sitting in 3-space.

It can happen that $k+2$ sites lie on a common k-flat on the boundary of the convex hull of the set of sites, for some $k < d$. Then some boundary face of the convex hull is the union of several Delaunay cells.

S is in *general position* if it is affinely independent and if no $d+2$ points lie on a common sphere. Clearly, if S is in general position, then all Delaunay and Voronoi cells are nondegenerate, so every Delaunay cell $D(R)$ is a simplex, every Voronoi cell $V(R)$ has dimension $d+1-|R|$, and every face of the convex hull of S is a Delaunay cell.

It is convenient to have a local characterization of the Delaunay triangulation. Two opposite d-cells cell(R) and cell(R') are *locally Delaunay* if R and R' both have circumspheres C and C', and $R'-R$ is outside C (equivalently, $R-R'$ is outside C'). A triangulation T is *locally Delaunay* if every pair of opposite d-cells is locally Delaunay.

Lemma 2.2 *A triangulation is Delaunay iff it is locally Delaunay.*

Proof. (Sketch) If T is Delaunay, then easily T is locally Delaunay. Suppose T is locally Delaunay. Choose a d-cell cell(R) and site $s \in S-R$; it suffices to show that s is outside the circumsphere of R. We can choose a point $r \in$ cell(R) so that open segment rs avoids all k-cells of T, for $k \le d-2$, and intersects $(d-1)$-cells in a single point. Intersection with segment rs gives a linear order on an alternating sequence of d- and $(d-1)$-cells cell$(R) = P_0, Q_1, P_1, \ldots, P_{m-1}, Q_m, P_m$ ending at a d-cell P_m incident to s. An easy downwards induction, $i = m, \ldots, 1$, using the local Delaunay property shows that the hyperplane through Q_i separates s from P_{i-1} and that s lies outside the circumsphere of P_{i-1}. The lemma follows using the property that the Delaunay triangulation is the unique triangulation having empty circumspheres. □

2.2 Connection with convex polyhedra

The Delaunay triangulation of a set S of point sites in dimension d can be obtained by appropriately lifting the sites to $d+1$ dimensions and then projecting the lower convex hull of the lifted sites back to d dimensions. The Voronoi diagram in dimension d can be obtained by appropriately mapping sites to hyperplanes in dimension $d+1$, taking the intersection of the upper halfspaces, and projecting back to d dimensions. Much of the material in this section is from Edelsbrunner and Seidel[23]; they extend earlier work of Brown.[11] Edelsbrunner's book[24] is another source.

We now describe the appropriate mappings. To make the geometry easier to visualize, we consider the case when the set of sites S is in dimension 2. We identify

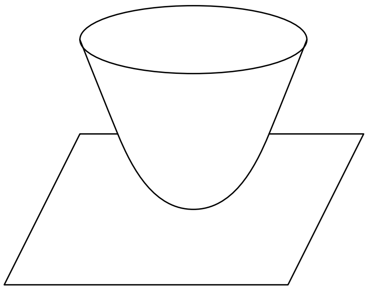


Figure 4: The paraboloid Λ.

E^2 with the plane spanned by the first 2 coordinate axes of E^3; it is the *base plane*. We think of the third coordinate axis as the vertical direction. We write for example x for the point (x_1, x_2) and we use dot products, thus $x \cdot p$ is $x_1p_1 + x_2p_2$. If $x = (x_1, x_2)$ and r is a real, then (x, r) is (x_1, x_2, r). This notation makes the generalization to higher dimension obvious.

For Delaunay triangulations, the appropriate *lifting map* $\lambda : E^2 \to E^3$ is $\lambda(x) = (x_1, x_2, x_1^2 + x_2^2) = (x, x \cdot x)$. The image of λ is

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

Λ is a paraboloid of revolution (about the vertical axis). See Figure 4.

A plane is *nonvertical* if it is not parallel to the vertical axis. Clearly, if P is a nonvertical plane, then any point in E^3 is either above, on, or below P. The crucial property of Λ is the following. Any nonvertical plane P either misses Λ, touches it at a tangent point, or the projection of $P \cap \Lambda$ onto the base plane is a circle. In the last case, the portion of Λ below the plane projects inside the circle, and the portion above projects outside. Conversely, given any circle in E^2 there is a plane that intersects Λ in a set whose projection is the circle. To see this, note that for any nonvertical plane P there is a real c and a point $p \in E^2$ so that

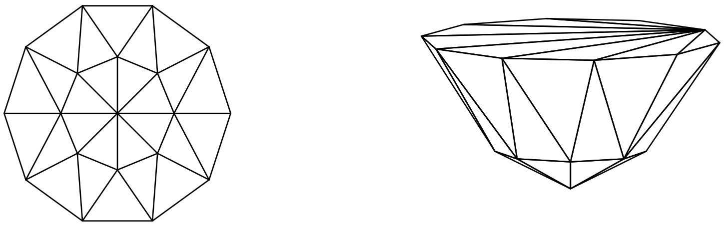
$$P = \{(x, z) : x \in E^2, z = 2x \cdot p - p \cdot p + c\}.$$

The vertical distance from the point $(x, 2x \cdot p - p \cdot p + c)$ of P to the point $(x, x \cdot x)$ of Λ is

$$x \cdot x - 2x \cdot p + p \cdot p - c = r^2 - c,$$

where r is the distance from x to p. If $c < 0$, this is positive for all x, and P is below Λ. If $c = 0$, then this is zero exactly at $x = p$, and it is easy to see that P is tangent to Λ at $(p, p \cdot p)$. If $c > 0$, then the projection of $P \cap \Lambda$ on the base plane is the circle with center p and radius $\sqrt{c}$. A point $(x, x \cdot x)$ of Λ is above P iff $r^2 > c$, i.e., iff x is outside the circle. The converse direction, obtaining a plane from a circle, is similar.

Suppose S is a set of sites in the base plane. Consider the convex hull H of $\lambda(S)$. A face of H is a *lower* face if it has a nonvertical supporting plane P so that H lies on or above P. Equivalently, a lower face is visible to an observer positioned at



(a) (b)

Figure 5: (a) A Delaunay triangulation; (b) the convex hull of the lifted sites.

a very negative vertical coordinate without looking through the interior of H. The Delaunay triangulation of a set of sites S is shown in figure 5(a); 5(b) shows the convex hull H of $\lambda(S)$.

Lemma 2.3 *Let H be the convex hull of $\lambda(S)$ and D the Delaunay triangulation of S. Then D is the set of projections of lower faces of H onto the base hyperplane.*

Proof. Suppose F is a lower face of H, then $F = \text{cell}(\lambda(R))$, for some nonempty $R \subseteq S$. Since F is lower, there is some nonvertical supporting plane P with $\lambda(R) \subseteq P$ and $\lambda(S-R)$ above P. The projection of $P \cap \Lambda$ onto the base plane is a circle C. Since $\lambda(R) \subseteq P \cap \Lambda$, R lies on C; since $\lambda(S-R)$ lies above P, $S-R$ is outside C. Hence $\text{cell}(R)$ is Delaunay, but $\text{cell}(R)$ is also the projection of $F = \text{cell}(\lambda(R))$ onto the base plane. The converse argument is essentially the reverse. □

A result that is similar to lemma 2.3 can also be obtained using stereographic projection.[11] Place a unit-radius 2-sphere Σ so that it is tangent to the base plane, and let t be the point of Σ diametrically opposite the point of tangency. The *stereographic projection* $\sigma(x)$ of a point x in the base plane is the point of intersection of Σ with the line segment tx. The crucial property of stereographic projection[53] is that a circle in the base plane maps to a circle on Σ. Let H_σ be the convex hull of $\{t\} \cup \sigma(S)$. An argument similar to lemma 2.3 shows that σ is a bijection between cells of D and the faces of H_σ that are not incident to t.

Now we consider Voronoi diagrams. The appropriate mapping sends the site s to the plane

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

Clearly P_s is tangent to Λ at $(s, s \cdot s)$. Consider also the surface

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

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 an upside-down paraboloid touching the base plane at s and below the base plane elsewhere. See Figure 6 for the situation in dimension 1. If we think of P_s, U_s and Λ as functions from the base plane to the vertical direction,

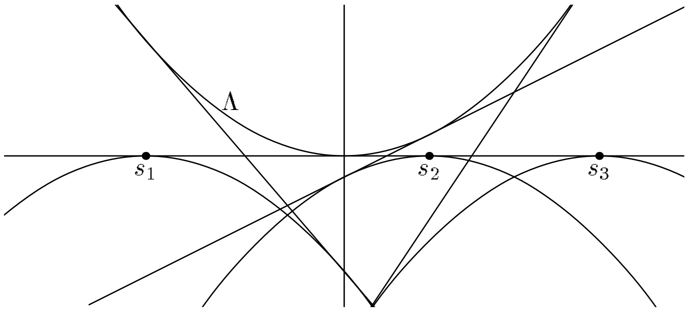


Figure 6: The relation among Λ, P_s, and U_s.

then obviously $U_s = P_s - \Lambda$. The crucial point is that Λ is independent of s: for sites s, s' and any x in the base plane, we have $P_s(x) > P_{s'}(x)$ iff $U_s(x) > U_{s'}(x)$ iff x is further from s than from s'. Similarly, $P_s(x) = P_{s'}(x)$ iff x is equidistant from s and s'.

Lemma 2.4 *Let I be the intersection of the closed halfspaces above the planes $\{P_s\}$. The Voronoi diagram V is the set of projections of faces of I onto the base plane.*

Proof. F is a face of I 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 $r' \in S - R$. Hence $x \in V(R)$. The converse is similar. □

It is possible to give a visual interpretation of lemma 2.4. Use opaque paint to give each surface U_s a different color. An observer sitting at a very positive vertical coordinate 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 sitting inside I at a very positive vertical coordinate 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 $\{U_s\}$.

2.3 Combinatorial complexity

Let S be a set of n sites in E^d. The *combinatorial complexity* of the Voronoi diagram V is the number of cells in V. Clearly, the number of cells of V is the same as the number of cells of the Delaunay triangulation D.

If S is in general position, then the total number of cells of D is at most the number of d-cells of D times a constant depending exponentially on d. This follows because any cell of D is a face of some d-cell of D, and if S is in general position, then each d-cell is a simplex and has at most $\binom{d+1}{k+1}$ k-faces. If S is not in general position, then for example the sites could all be cospherical, and there is only a single d-cell but many k-cells for $k < d$.

In dimension 2 it is easy to show using planarity that the number of vertices of V (or triangles of D) is at most $3n - 6$ and the number of edges of V (or edges of D) is at most $2n - 4$. In dimension higher than 2, it is possible to bound the number of cells of V using the upper bound theorem for convex polyhedra.[10,24] The upper bound theorem states that the number of j-faces of a convex polyhedron in dimension d with n facets is asymptotically $O(n^{\min(d-j,\lfloor d/2 \rfloor)})$. ($\lfloor x \rfloor$ and $\lceil x \rceil$ are the integers resulting from x by rounding down and up, respectively.) Using the correspondence of section 2.2 between the Voronoi diagram V in dimension d and the convex polyhedron I in dimension $d+1$, the number of j-faces of V is asymptotically at most $O(n^{\min(d+1-j,\lceil d/2 \rceil)})$. Hence the combinatorial complexity of V is at most $O(n^{\lceil d/2 \rceil})$.

A simple example attains this worst-case bound, at least within a constant factor independent of n. First consider dimension 3, with coordinate axes x_1, x_2, and x_3. Let P_1 be the segment along the x_1 axis with endpoints $(0,0,0)$ and $(1,0,0)$ and P_2 the segment in the $x_1 = 0$ plane parallel to the x_2 axis with endpoints $(0,0,1)$ and $(0,1,1)$ Put $n/2$ sites equally spaced along P_1 and $n/2$ sites equally spaced along P_2. It is easy to see that there is an empty sphere through any two adjacent sites on P_1 and any two adjacent sites on P_2. This gives a total of $\theta(n^2)$ Delaunay triangles. Notice that each site can be perturbed within a small ball while maintaining the $\theta(n^2)$ complexity. This example can be extended to give $\theta(n^{\lceil d/2 \rceil})$ cells in any odd dimension d.

Using known exact bounds on the number of faces of a convex polyhedron and the connection between convex polyhedra and Voronoi diagrams, Seidel[57] gives *exact* bounds on the worst-case number of j-faces of a Voronoi diagram in d dimensions, for all j and d.

For a set of point sites chosen 'at random', the combinatorial complexity of the Voronoi diagram is probably much smaller. However, the available results are not as general as might be desired. One model of 'random sites' is an infinite set of point sites given by a unit-intensity Poisson process[56] in E^d; then the expected number of sites in a subset of E^d is equal to the measure of the subset. Meijering[48] shows that the expected number of Voronoi neighbors of a site is a constant depending only on the dimension d. The constant is 6 for $d = 2$ and about 15.54 for $d = 3$.

A second model of 'random' sites is point sites chosen with a uniform distribution inside a bounded set $H \subset E^d$. The situation is more complicated because of edge effects near the boundary of H. Dwyer[21] shows that the expected number of Delaunay d-cells for a set of n sites chosen uniformly within a sphere in E^d is $\sim c_d n$. The constant c_d agrees with the constants given by Meijering for $d = 2$ and $d = 3$. For $d > 5$, an estimate[22] of the constant accurate to within 1% is

$$c_d \approx \frac{\sqrt[4]{e}\sqrt{2/\pi}}{d(d+1)}(2\pi d)^{d/2}.$$

Dwyer conjectures that the bound holds for point sets chosen uniformly from within an arbitrary convex set H. If only a small fraction of the points lie near the boundary of H, then edge effects are negligible.

3 Properties of the Voronoi diagram and Delaunay triangulation

3.1 Optimality of the Delaunay triangulation

Good finite element meshes avoid long, skinny triangles as much as possible. A classic result of Lawson[45] is that two-dimensional Delaunay triangulations optimize one formalization of this criterion.

Lemma 3.1 *Over all proper triangulations of a set of sites $S \subset R^2$ in general position, the Delaunay triangulation maximizes the minimum angle of any triangle.*

Proof. Let angles(T) be the sorted sequence of angles of triangles in T. The lemma follows by showing that the Delaunay triangulation D is the triangulation that lexicographically maximizes angles(T), where smaller angles are more significant in the ordering. We show that if T is not the Delaunay triangulation, then there is a lexicographically larger triangulation T'. If T is not Delaunay, then by Lemma 2.2 it is not locally Delaunay, so there must be two triangles abc and acd so that d is inside the circumcircle of triangle abc (and b is inside the circumcircle of acd). Necessarily $abcd$ is a convex quadrilateral. Some easy elementary geometry shows that the minimum angle of triangles abd and bcd is larger than the minimum angle of triangles abc and acd. Hence if T' is the triangulation obtained from T by replacing edge ac with edge bd, then angles(T') is larger in the lexicographic ordering than angles(T). □

If the set of sites S is not in general position, then some sites may be cocircular and a Delaunay cell may be the interior of a circularly-inscribed convex polygon with more than three sides. In this case the Delaunay triangulation D can be completed to a proper triangulation by adding diagonals to the convex polygon. The minimum angle in the resulting triangulation is independent of how diagonals are added, since it is determined by the shortest side of the convex polygon. Essentially the same argument as lemma 3.1 shows that any such completion of the Delaunay triangulation maximizes the minimum angle over all triangulations. However, one might require that the triangulation lexicographically maximize the sorted sequence of angles, in which case it matters very much how diagonals are added. Mount and Saalfeld[50] give an efficient algorithm that determines how to add diagonals so that the resulting triangulation is the lexicographic maximum.

No generalization of lemma 3.1 is known for higher dimensions, at least in terms of some angular measure of simplices. Rajan[54] gives a different optimality criterion that does hold in any dimension.

The *min-containment sphere* of a simplex is the minimum-radius sphere that contains the simplex. If the center of the circumsphere of the simplex lies inside the simplex, then the circumsphere is the min-containment sphere. Otherwise, the center of the min-containment sphere lies on some face of the simplex. For example, in two dimensions, if a triangle is acute, then its min-containment circle is its circumcircle, and if the triangle is obtuse, then its min-containment circle has center at the midpoint of the longest edge. For T a proper triangulation, let $\mathrm{maxrad}(T)$ be the maximum radius of any min-containment sphere of any simplex in T.

Lemma 3.2 *Over all proper triangulations T of a set of sites $S \subset E^d$ in general position, the Delaunay triangulation minimizes* $\mathrm{maxrad}(T)$.

Proof. (Sketch) Recall the lifting map λ from section 2.2. A proper triangulation T in E^d can be lifted to a polyhedral surface $\hat{T}$ in E^{d+1} by mapping $\mathrm{cell}(R)$ to $\mathrm{cell}(\lambda(R))$. An easily verified fact is that the square of the radius of the min-containment sphere of a cell C of T is the maximum over x in C of the vertical distance between the paraboloid Λ and the polyhedral surface $\hat{T}$ above x. The maximum vertical distance between Λ and the polyhedral surface $\hat{T}$ is minimized if $\hat{T}$ is the convex hull of $\lambda(S)$, i.e. if T is the Delaunay triangulation D. □

3.2 Geometric graph properties

A Delaunay triangulation or Voronoi diagram can be thought of as a geometrically embedded graph by restricting attention to just its vertices and edges. A number of results exploit this geometric graph structure.

Several classical graphs with vertex set $S \subset E^d$ capture some form of the neighborhood structure of S. The *euclidean minimum spanning tree EMST* of S is the spanning tree of S that minimizes the sum of the euclidean lengths of the edges in the tree. The *relative neighborhood graph*[60] *RNG* of S has an edge connecting s_i and s_j if there are no sites in the interior of the intersection of the two discs centered at s_i and s_j with radius the distance between s_i and s_j. The *Gabriel graph*[32] *GG* of S has an edge connecting s_i and s_j if there are no sites in the interior of the disc having as diameter the segment $s_i s_j$. It is easy to see that the edge sets of these graphs satisfy

$$EMST \subseteq RNG \subseteq GG \subseteq D.$$

In two dimensions, these containment relations can form the basis of algorithms that compute these graphs.[53]

A graph G with vertex set $S \subset E^d$ has *dilation* c if for any $s_i, s_j \in S$, the length of the shortest path between s_i and s_j along edges of G is at most c times the euclidean distance between s_i and s_j. Here the length of an edge is just the euclidean distance between its endpoints. Clearly the complete graph has dilation 1, but it has $O(n^2)$ edges. In two dimensions, Delaunay triangulations give planar graphs with small dilation but only $O(n)$ edges. For any set S, Chew[13] gives a planar graph $\Delta = \Delta(S)$ with dilation $c_\Delta = 2$; the edges of Δ are edges of the Delaunay triangulation of S using a convex distance function whose unit ball is an equilateral triangle. Naively it would seem that the dilation c_D of the graph with edges from the regular Delaunay triangulation D would be smaller, since a circle is much more symmetric than a triangle. However this has not been shown; Keil and Gutwin[43] show that $c_D \leq 2\pi/(3\cos(\pi/6)) \approx 2.42$. It is not hard to see that $c_D \geq \pi/2$. Chew[13] conjectures that the true bound for c_D is close to $\pi/2$. Using other methods, it is possible to get graphs with small dilation. For any set of n points in E^d and any $\epsilon > 0$, Vaidya[62] is able to construct a graph with dilation $1+\epsilon$ and $O(n/\epsilon^d)$ edges.

Given a cell complex whose union is the plane, one can ask if it is the Voronoi diagram of some set of sites S. Ash and Bolker[1] give a criterion that can decide this question. Let G be the edge graph of the cell complex and assume that each vertex has three edges incident. For each vertex v of G there are three rays with endpoint v so that the edges incident to v bisect the angles formed by the rays. Each ray enters one of the planar regions incident to v; the ray entering planar region R is the *central ray* from v into R. Ash and Bolker show that the cell complex is a Voronoi diagram iff for each region the central rays entering the region have a point in common. If the central rays entering the region intersect in a unique point, then it is the site determining the region (the point is unique in all but extremely degenerate situations). Ash and Bolker[2] give generalizations for other forms of planar Voronoi diagrams; Aurenhammer[3] generalizes to higher dimensions. The analogous question of whether a cell complex is a Delaunay triangulation is easy to answer: lemma 2.2 gives an easy local characterization of Delaunay triangulations.

3.3 Inscribability

Steinitz's theorem[34] states that the edge graphs of convex polytopes in three dimensions are exactly the three-connected planar graphs. A long-standing open question of Steiner was to characterize which of these graphs are *inscribable*, that is, which graphs are actually the edge graphs of convex polytopes with all vertices lying on a common sphere. Rivin and Smith[55] give the following answer. A three-connected planar graph G is inscribable iff the edges of G can be assigned real number weights so that the weight of a circuit around a face is exactly one and the weight of a circuit not bounding a face is strictly greater than one. Note that three-connected planar graphs have combinatorially unique embeddings, so the faces of G are uniquely determined. A variant of the ellipsoid algorithm for linear programming can determine whether the weights exist, in time polynomial in the number of vertices of G.

Rivin and Smith's theorem has some interesting consequences for Delaunay triangulations in two dimensions. One is a test to decide if an undirected graph is the edge graph of a Delaunay triangulation. Suppose G is a three-connected planar graph with a specified 'external' face and specified 'extreme' vertices incident to the external face. G has a *Delaunay realization* if it can be embedded in the plane so that its edges become Delaunay edges, the 'external' face is unbounded, and 'extreme' vertices are extreme on the convex hull of the embedded vertices. Notice that four or more vertices around an internal face must be embedded on a common circle and a chain of vertices bounding the external face between extreme vertices must be embedded on a common line. Let G' be the graph obtained from G by inserting a new vertex in the external face with edges to the extreme vertices. Using stereographic projection as described in section 2.2, it is easy to see that G has a Delaunay realization exactly

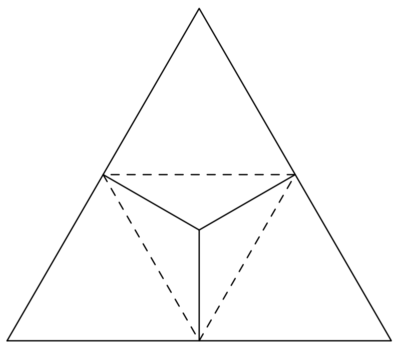


Figure 7: If the dashed edges are added, the resulting graph is not Delaunay

if G' is inscribable (the new vertex of G' gets mapped to the point t used to define the stereographic projection).

Another consequence is a condition characterizing when the sites of an improper Delaunay triangulation can be perturbed to obtain a particular completion of the triangulation. Suppose D is the Delaunay triangulation of a set S with edge graph G. Let G' be the graph as before, using the true extreme sites of S. Complete G' to a proper triangulation, then delete the new vertex and all edges incident to it, and call the new graph G''. G'' is like a completion of G with the additional property that nonextreme sites on the convex hull of S have been decided to be either extreme or inside the convex hull. Dillencourt, Rivin and Smith[19] show that G'' always has a Delaunay realization if G' is not bipartite. In the case that G' is bipartite, say with color classes red and blue, then G'' is still Delaunay realizable as long as both red-red and blue-blue edges were added to G to get G''. Notice that G' is bipartite only in an extremely degenerate situation: there must be an even number of edges between extreme vertices on the convex hull, and an even number of edges around each internal face; even in this degenerate situation, most completions are still Delaunay realizable. If G' is bipartite and only red-red or blue-blue edges are added, then G'' is not Delaunay realizable. For example, in figure 7, if the dashed edges are added, then the resulting graph is not Delaunay realizable. Dillencourt, Rivin, and Smith are also able to show that any triangulation without chords or nonfacial triangles is Delaunay realizable; a chord is an edge connecting two nonconsecutive vertices on the outer face, which is the only face that does not have to be a triangle.

Dillencourt[18] gives a necessary condition for a graph to be a Delaunay triangulation. For specific graphs this condition may be easier to apply than the circuit condition given by Rivin and Smith. A graph is *1-tough* if deleting any k vertices (and incident edges) splits the graph into at most k components. Dillencourt shows that the edge graph G of a nondegenerate Delaunay triangulation is 1-tough. If the Delaunay triangulation is degenerate, then G is 'almost' 1-tough: deleting any k vertices results in at most $k+1$ components. In either case a consequence is that G has a perfect matching, using a classic theorem of Tutte. Dillencourt offers the 1-toughness of Delaunay graphs as a partial explanation for the observed phenomenon that 'most'

Delaunay graphs have Hamiltonian cycles.

4 Algorithms

This section outlines some of the fundamental algorithms that are known for Delaunay triangulations and Voronoi diagrams. For various reasons, it is much more convenient to express algorithms in terms of the Delaunay triangulation than the Voronoi diagram, so we do so here. The Voronoi diagram is easily obtained from the data structure suggested below for Delaunay triangulations; the only additional information needed is the location of vertices, edges, etc.

The algorithms are described using the 'real random access machine' (real RAM) model.[53] Thus memory can contain real numbers; operations $+$, $-$, $\times$, $/$, $=$ and $>$ on reals are exact and take unit time. The algorithm descriptions also assume that the set S of sites is in general position. These two assumptions are discussed more fully in section 4.7.

The 'figure of merit' of an algorithm is usually its worst-case running time. In E^2, a worst-case lower bound of $\Omega(n \log n)$ is known for computing the Delaunay triangulation of n sites in the real-RAM model,[53] and in E^d, $d{\geq}3$, a lower bound of $\Omega(n^{\lceil d/2 \rceil})$ follows because the Delaunay triangulation can have that complexity. Algorithms that match these lower bounds are known, so in a sense there is no reason to consider any other algorithms. However, there are other more subjective criteria. One is difficulty, both conceptual and in terms of implementation. Another is running time 'in practice.' The probabilistic estimates of section 2.3 suggest that the combinatorial complexity of the Delaunay triangulation is linear for 'random' sites, so worst-case estimates are far too pessimistic. However, relatively little is known about expected running time on 'random' sites, and there are few empirical comparisons of Delaunay triangulation algorithms in the literature.

The next section discusses some basic tools used to implement the algorithms. It outlines a graph data structure for representing triangulations and convex hulls, and gives geometric primitives used by many of the algorithms. Perhaps the simplest Delaunay triangulation algorithm in two dimensions is the "flipping algorithm," discussed in section 4.2. It attempts to produce the Delaunay triangulation by local modification of an arbitrarily-chosen triangulation. Naively, it has no connection with convex hulls, but a good way to understand it turns out to be with the lifting map of section 2.2. Section 4.3 gives an incremental algorithm for Delaunay triangulations, where the sites are added one by one. The incremental algorithm generalizes nicely to arbitrary dimension; the best way of understanding it is as an incremental convex hull algorithm on the set of lifted sites, that is, as the beneath-beyond algorithm.[53] If the sites are inserted in random order, then it turns out that the expected worst-case running time of the incremental algorithm is improved; this is discussed in section 4.4. The plane-sweep algorithm in section 4.5 is one of the few algorithms that has no apparent connection with convex hulls. However, there is a way of understanding it using a different mapping to three dimensions. Section 4.6 briefly discusses other algorithms. The general position assumption and the real RAM model are reconsidered in section 4.7. Finally, some implementation issues and results are given in section 4.8.

4.1 Primitives for Delaunay triangulation algorithms

This section describes two geometric primitives, the "orientation test" and the "incircle test," that are sufficient for implementing many Delaunay triangulation algorithms. Also, there is a description of a simple graph data structure that can be used to represent a triangulation or convex hull.

The *orientation* of a sequence of $d{+}1$ points $(p_1, \ldots, p_{d+1})$ in E^d is either -1, 0, or $+1$; it is 0 if the set of $d{+}1$ points lie on a common k-flat for $k < d$. Otherwise it is defined recursively as follows: for $d = 1$, (p_1, p_2) has positive orientation if $p_1 < p_2$ and negative otherwise; for $d > 1$, identify E^d with the hyperplane spanned by points $p_2, \ldots, p_{d+1}$ so that p_1 lies in the positive halfspace of E^d, then the orientation of $(p_1, \ldots, p_{d+1})$ is the orientation of $(p_2, \ldots, p_{d+1})$ in E^d. Thus for $d = 2$, (p_1, p_2, p_3) has positive orientation if p_1 lies to the left of the line directed from p_2 to p_3, or equivalently if p_1, p_2, p_3 are in counterclockwise order. For $d = 3$, (p_1, p_2, p_3, p_4) has positive orientation if looking from p_1 to the plane spanned by p_2, p_3, p_4, (p_2, p_3, p_4) has positive orientation. Orientation is preserved under an even permutation of the points and negated under an odd permutation. If $p_i = (p_{i1}, p_{i2}, \ldots, p_{id})$, then the orientation of $p_1, \ldots, p_{d+1}$ is given by the sign of the determinant

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

(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). The *orientation test* on $d + 1$ points is just to evaluate the sign of this determinant. The orientation test is sufficient to determine 'convexity properties' of a set of points; Edelsbrunner and Mücke[25] give a number of examples of its use.

Suppose the sequence of $d{+}1$ points $(p_2, \ldots, p_{d+2})$ in E^d has positive orientation. By the results of section 2.2, p_1 is outside the circumsphere of $p_2, \ldots, p_{d+2}$ exactly if $\lambda(p_1)$ is above the hyperplane spanned by $\lambda(p_2), \ldots, \lambda(p_{d+2})$, which happens exactly if the orientation of $(\lambda(p_1), \ldots, \lambda(p_{d+2}))$ is positive, which in turn happens exactly if the determinant

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

is positive. If the determinant is zero, then it is easy to see that $p_1, \ldots, p_{d+2}$ are cospherical, and if the determinant is negative, then p_1 is inside the circumsphere of $p_2, \ldots, p_{d+2}$. The evaluation of the sign of this determinant is the *incircle test*[36] for points $p_1, \ldots, p_{d+2}$.

A triangulation algorithm requires some data structure to represent the triangulation. There are known data structures that will represent arbitrary d-dimensional cell complexes.[9,46] However, because of the general position assumption made earlier the Delaunay triangulation D of the set of sites S is proper, that is, all cells are simplices. Hence a fairly simple graph data structure can represent the triangulation. Each d-cell C of the triangulation should be represented by a node in memory; the node contains a vector of the indices of the $d+1$ sites that determine the cell and a vector of $d+1$ pointers to the nodes for the d-cells that are opposite C in the triangulation. A facet common to two d-cells is thus represented by two pointers, one in each direction between the nodes representing the two d-cells. A convenient correspondence is that the kth pointer stored in a node corresponds to the facet obtained by deleting the kth site from the node. Facets on the boundary of the triangulation could be distinguished with null pointers. A perhaps preferable alternative is to give the triangulation the topology of a sphere. One way to do this is to create a dummy vertex ν (thought of as somewhere in the unbounded complement of the triangulation) and to triangulate the complement topologically using ν, that is, to add a d-cell formed from ν and each boundary facet of the triangulation. Then the boundary facets are obtained by examining the d-cells containing ν.

Cells of dimension $d-2$ or less are not represented explicitly; each such cell is determined by a subset of the sites of some d-cell. Some care is needed to exploit the structure implied by lower-dimensional cells. For example, the facets and d-cells incident to a ridge form a cyclic order around the ridge. Traversal of this cyclic order is possible at the expense of some manipulation of facet pointers. Suppose cell(R) is a d-cell with ridge cell(Q). A convenient representation of cell(Q) is the pair of sites in $R - Q$. Each such site s determines a facet cell$(R - \{s\})$; these are the two facets on either side of cell(R) in the cyclic ordering. If the pointer for facet cell$(R - \{s\})$ is traversed to reach a d-cell cell(R'), then $R' - Q$ is obtained from $R - Q$ by deleting s and adding a site s'; s' can be deduced by examining all facet pointers of cell(R'), since the pointer for facet cell$(R' - \{s'\})$ points to cell(R). Similarly, the ridges, facets, and d-cells incident to a $(d-3)$-cell have the topology of a sphere, which can be traversed in an analogous though more cumbersome fashion.

Essentially the same data structure can be used to represent the boundary of the convex hull of a lifted set of sites, since all the faces are still simplices. Notice that the boundary of the convex hull already has the topology of a sphere. Clearly, the Delaunay triangulation can be obtained by just selecting the lower faces of the convex hull and ignoring the vertical coordinate of a lifted site.

4.2 Flipping

Let S be a set of $n \geq 3$ sites in E^2. The simplest algorithm for computing the Delaunay triangulation of S is edge flipping:

1. Determine some triangulation of S.

2. While there are two opposite triangles abc and acd that are not locally Delaunay, flip the diagonal, that is, replace the two triangles with triangles abd and bcd. This flip is a *Delaunay diagonal flip.*

A simple way to determine a triangulation of S is to add the sites one by one in x-sorted order (using y-order to break ties). The triangles for a new site s_i are constructed using boundary edges of the current triangulation visible from s_i; these can be determined by traversing the boundary of the triangulation first clockwise, then counterclockwise, from the previous last site s_{i-1}. For the second step, the incircle test from section 4.1 can be used to determine if two opposite triangles are not locally Delaunay. Notice that if two opposite triangles are not locally Delaunay, they together form a convex quadrilateral, so indeed it is possible to flip the diagonal. To save scanning the triangulation at every iteration, a queue of edges can be maintained between opposite triangles that are not locally Delaunay. The queue is initialized once at the beginning, and updated as necessary using the bounding edges of a quadrilateral involved in a flip. Working out the details, it is easy to see that the algorithm can be made to run in time $O(n \log n + f)$, where f is the number of flips.

Why is the number of flips finite? A triangulation T in E^2 lifts to a polyhedral surface $\hat{T}$ in E^3 by mapping a face $F = \text{cell}(R)$ into a face $\hat{F} = \text{cell}(\lambda(R))$. Suppose that two opposite triangles $\Delta_1 = abc$ and $\Delta_2 = acd$ are not locally Delaunay. An observer sitting at a negative vertical position looking up at the surface will see that the two lifted triangles $\hat{\Delta}_1$ and $\hat{\Delta}_2$ form a concave dihedral angle. Flipping the diagonal results in a triangulation T' with triangles $\Delta_1' = abd$ and $\Delta_2' = bcd$ substituted for Δ_1 and Δ_2. The lifted surface $\hat{T'}$ now has two triangles $\hat{\Delta}_1'$ and $\hat{\Delta}_2'$ forming a convex dihedral angle. The symmetric difference between $\hat{T}$ and $\hat{T'}$ is essentially the boundary of the tetrahedron $\lambda(a)\lambda(b)\lambda(c)\lambda(d)$; $\hat{T'}$ has resulted from T by substituting the two faces of the tetrahedron visible from below for the two faces obscured by the interior of the tetrahedron. Thus $\hat{T'}$ is everywhere equal to or below $\hat{T}$. In particular, the edge $\lambda(a)\lambda(c)$ lies above $\hat{T'}$, and above every subsequent lifted triangulation. Hence edge ac will never reappear in any subsequent triangulation. Since there are only $\binom{n}{2}$ possible edges among the n sites of S, and some edge disappears forever after every flip, the number of flips is at most $\binom{n}{2}$.

This quadratic bound can be achieved in the worst case. Consider the triangulation T in Figure 8a; the Delaunay triangulation D of the same set of sites is depicted in Figure 8b. At least $\Omega(n^2)$ Delaunay flips are required to transform T into D, no matter in what order the flips are performed.[28] Notice that only $O(n)$ diagonal flips are necessary to transform T into D–this bound holds for the flip distance be-

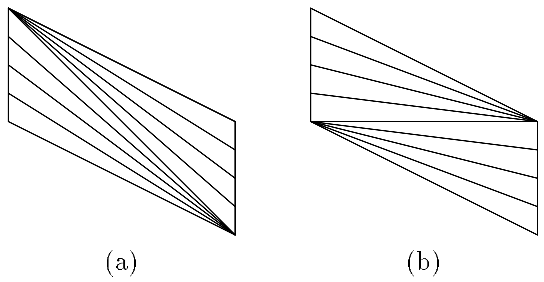


Figure 8: $\Omega(n^2)$ Delaunay flips are necessary to transform triangulation T in (a) into D in (b).

tween any two triangulations of a convex polygon–but not all the flips are Delaunay diagonal flips.

Not much is known about the number of flips required to transform a 'random' triangulation into the Delaunay triangulation, in part because of the lack of a satisfactory definition of a 'random' triangulation. Empirical evidence suggests that edge flipping is a reasonable algorithm in practice; see section 4.8. If the sites are added in random order, and flips are performed to update the triangulation to the Delaunay triangulation after each addition, then Guibas, Knuth and Sharir[35] show that the expected number of flips is linear. The expectation is over the permutation used to order insertions and holds for any set of sites.

Naively, it seems that the flipping algorithm should extend to higher dimension. The notion of a 'flip' is now more complicated; it does not suffice to consider just two adjacent simplices. Let S be a set of sites in E^d, and consider a set $R \subset S$ of $d+2$ sites so that all sites are on the boundary of the convex hull of R. The set $\lambda(R)$ forms a simplex in E^{d+1}. Viewed from below, some facets are visible and some facets are obscured by the interior of the simplex. A *Delaunay flip* in a triangulation T of S substitutes a set of cells V for a set of cells O if V and O are projections of the visible and obscured facets of such a simplex $\lambda(R)$. For example, for $S \subset E^3$, a simplex of $d+2$ points in E^4 has five tetrahedral faces, either two or three of which can be visible; hence a Delaunay flip either replaces two tetrahedra with three, or three tetrahedra with two. Clearly, if T' results from T by a Delaunay flip, then $\hat{T}'$ is everywhere at or below T. Since some simplex disappears forever at each flip, and there are only a finite number of simplices with vertices chosen from S, it would seem that the Delaunay triangulation results after a finite number of flips. Unfortunately, flipping can get stuck. Joe[39] shows that there are non-Delaunay triangulations in E^3 for which no Delaunay flip applies. Roughly speaking, the problem is that two opposite simplices can be locally nonDelaunay, but the union of the two is not convex, so no flip is possible. This is possible for the first time in E^3; in E^2 if two opposite triangles do not form a convex quadrilateral, then they are locally Delaunay. Joe[40] shows that an incremental form of flipping does work in three dimensions; the incremental flipping is similar to the incremental algorithm, discussed below. Incremental flipping was extended to general dimension by Rajan,[54] with a logarithmic overhead

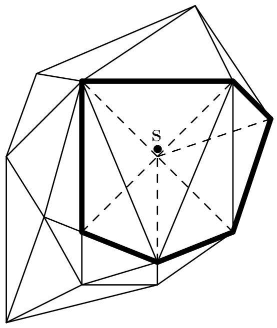


Figure 9: P is given by dark edges and its triangulation using dashed edges.

per flip, and by Edelsbrunner and Shah,[26] with an amortized constant overhead per flip.

4.3 The incremental algorithm

The incremental algorithm adds sites one by one; at each addition, the Delaunay triangulation of the previous set of sites is updated to include the new site.

In two dimensions, the incremental algorithm is easy. For simplicity, choose an initial triangulation so that all subsequent sites lie inside the triangulation. Given a new site s, find the set T_v of triangles of the current triangulation whose circumcircles contains s. The boundary of the union of triangles T_v is a polygon P star-shaped from s (that is, for any point p in P, segment sp is contained in P). See Figure 9. Let T_s be the triangulation of P whose triangles are each formed from an edge of P and s, and replace T_v by T_s. The resulting triangulation is Delaunay since it is locally Delaunay: each edge lies between two old triangles, or between an old triangle whose circumcircle does not contain s and a triangle formed using the edge and s, or between two new triangles formed from s and the vertices of an old triangle whose circumcircle contains s. In the worst case, every triangle of T could appear in T_v, so adding a single site takes linear time, and adding all sites takes quadratic time. The non-worst-case analysis is given below.

This algorithm generalizes to higher dimension. However, the generalization is best understood as an incremental convex hull algorithm, applied to the set of lifted sites $\lambda(S)$. Thus the incremental Delaunay triangulation algorithm is just a version of the beneath-beyond algorithm.[53]

A useful observation concerns the changes to the convex hull of a point set when a new point is added. Suppose $R \cup \{r\}$ is a finite affinely independent point set in E^d, H is the convex hull of R, r is outside H, and H' is the convex hull of $R \cup \{r\}$. Then the faces of H' are exactly the faces of H invisible from r and on the horizon from r, faces of the form cell$(F \cup \{r\})$ for some horizon face F of H, and r itself.[47] See Figure 10. (This characterization requires affine independence, in particular, that r is not contained in the affine span of any horizon face. The general position

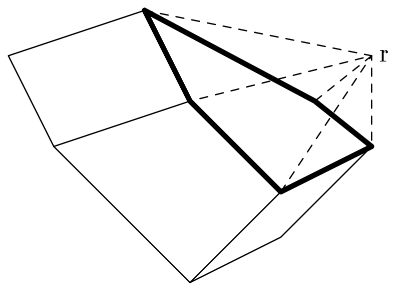


Figure 10: Obtaining H' from H. The heavy lines are the horizon faces.

assumption implies that the set of lifted sites $\lambda(S)$ is affinely independent. Figure 10 is drawn with nontriangular faces to make it appear more like a three-dimensional polyhedron.) Notice that any face of H visible from r lies in the interior of H'.

Let S be a set of at least $d+2$ sites in E^d. The incremental algorithm will produce the convex hull of the lifted sites $\lambda(S)$. The convex hull is represented using a graph data structure G, as described in section 4.1. The first step of the algorithm is to choose a subset of $d+2$ sites. The $d+2$ lifted sites form a simplex in E^{d+1} with $d{+}1$ d-faces, so G starts with $d{+}1$ nodes.

Suppose $R \subset S$ and the convex hull H of $\lambda(R)$ has been constructed. The iterative step of the algorithm is to choose a new site $s{\in}S{-}R$ and construct the convex hull H' of $\lambda(R \cup \{s\})$. Using the observation given before, adding s can be split into two steps:

1. Find the set H_v of d-cells of H visible from $\lambda(s)$.
2. Replace H_v with the set of d-faces $H_s = \{F_s : F_s = \mathrm{cell}(F \cup \{\lambda(s)\}),\ F \in H_h\}$, where H_h is the set of horizon $(d{-}1)$-faces of H from $\lambda(s)$.

(It should be clear that the algorithm in two dimensions given earlier is essentially the same, but described in terms of the triangulation itself.) For the first step, it suffices to find a single face in H_v, since as described in lemma 4.1 below, the remaining faces can then be found by a graph search in G.

Sets H_v and H_s form topological d-balls with the same boundary H_h. However, their worst-case sizes are different. H_v can be essentially all of H. Since H is a convex polytope in $d{+}1$ dimensions, by the upper bound theorem it has at most size $O(m^{\lceil d/2\rceil})$, where m is the number of sites determining H. H_s is the set of d-faces incident to a single vertex of a convex polytope in $d{+}1$ dimensions. This set has at most the size of a convex polytope in dimension d, i.e. $O(m^{\lfloor d/2\rfloor})$. One way to see this is to push a hyperplane supporting H' at $\lambda(s)$ into H' slightly. The resulting intersection is a convex polytope in d dimensions with faces in one-one correspondence with the faces of H_s (and also H_h).

Lemma 4.1 *Given a single face in H_v, it is possible to determine H_s in time $O(|H_v|)$.*

Proof. Here is a sketch of one algorithm. The data structure nodes for the d-faces in H_s can be created if the horizon $(d-1)$-faces H_h are known. These are exactly the $(d-1)$-faces between faces in H_v and faces not in H_v, and can be enumerated by a depth-first search through the graph data structure G starting at any face in H_v, backtracking at a transition from a face in H_v to a face not in H_v. The adjacencies among the faces in H_s can be discovered as follows. There is a $(d-1)$-face between d-faces F_s and F'_s of H_s exactly if $(d-1)$-faces F and F' in H_h have a common $(d-2)$-face. Furthermore, any $(d-2)$-face f of a face in H_h is incident to exactly two faces of H_h, since H_h is topologically a $(d-1)$-sphere. The d-cells of H incident to such a $(d-2)$-face f are cyclically ordered around f, and in particular the d-cells of H_v form a connected portion of the cyclic order. Hence given a $(d-2)$-face f of a face in H_h, the other face of H_h incident to f can be found by walking in this cyclic order through H_v until a d-cell not in H_v is discovered. The total time spent to construct all the cells H_s and to discover adjacencies is clearly $O(|H_v| + |H_h|) = O(|H_h|)$ plus the time spent spent traversing cyclic orders around $(d-2)$-faces. However, a d-cell of H_v is visited at most once for each of its $(d-2)$-faces, so this time is also $O(|H_v|)$. $\square$

Theorem 4.2 *Let S be a set of n sites in E^d, $d \geq 2$. The incremental algorithm to compute the Delaunay triangulation of S can be implemented so that in the the worst case it uses time $O(n^{\lceil (d+1)/2 \rceil})$ and space $O(n^{\lceil d/2 \rceil})$.*

Proof. The sites should be added in coordinate-sorted order (using lexicographic order to break ties). Then the next lifted site always sees one of the d-cells incident to the last lifted site added, so simply examining all such d-cells determines a d-cell visible to the new lifted site. The total time spent in the algorithm is determined by the number of d-cells ever created, since each d-cell gets charged once for being created, at most once for being deleted, and at most once for being examined. If the current hull is determined by m lifted sites, then adding a new lifted site creates at most $O(m^{\lfloor d/2 \rfloor})$ new d-cells. Summing from 1 to n yields $O(n^{\lceil (d+1)/2 \rceil})$. The space usage is determined by the maximum number of d-cells of the convex hull, which is $O(n^{\lceil d/2 \rceil})$. $\square$

Edelsbrunner[24] gives a careful description of this algorithm without assuming that the set of sites is in general position. Unfortunately, the details of the algorithm become much more complicated.

If the set of sites is 'random' the running time bound of theorem 4.2 is unduly pessimistic. The probabilistic estimates in section 2.3 suggest that $|H|$ is linear; hence one expects quadratic behavior from even a naive implementation that examines every face of H to find the faces of H_v. The probabilistic estimates also suggest that for a new 'random' site, $|H_v|$ is constant, so examining every face of H is unduly expensive. However, sorting the points as in theorem 4.2 destroys randomness (at least as far as is known), and the points may not all be available at the beginning of the algorithm. Hence an efficient way of finding a face in H_v is needed. Various heuristics have been suggested. One is to start at a site of the current Delaunay triangulation and walk through the triangulation in the direction of a new site s. The walk will find either a cell containing s or leave the triangulation; either case yields a face of H_v. Bowyer[8] suggests that a walk starting from a site near the centroid of the Delaunay triangulation should visit about $O(n^{1/d})$ cells, although no rigorous analysis is known. An improvement is to subdivide space with a regular grid. Before the algorithm starts, the set of sites in each grid cell is determined. The walk for a new site can then start at an already inserted site lying in the same or a nearby grid cell. In two dimensions, Ohya, Iri and Murota[51,52] give a 'quaternary incremental' algorithm that uses a hierarchy of grids in order to maintain an approximately uniform distribution of sites throughout the insertion process. They report linear performance for a variety of distributions. An alternative method of finding a face of H_v is presented in the next section.

4.4 The random incremental algorithm

The *random incremental algorithm* inserts sites in random order, with each permutation equally likely. One reason for considering the random incremental algorithm is that its expected worst-case running time is better than the worst-case running time of the incremental algorithm. Here the expectation is over the permutation used to order insertion; 'expected worst case' means the expected running time for the worst set of sites.

A requirement of the random incremental algorithm is a data structure to find a face of the current hull seen by a new lifted site. One such search data structure is based on a triangulation of the current hull obtained in a straightforward fashion from the sequence of intermediate convex hulls.

The initial triangulation is just the $(d+1)$-simplex H_0 formed by the first $d+2$ lifted sites. Suppose s is a new site, R is the current set of lifted sites, and H, H', H_v and H_s are as in the incremental algorithm. The triangulation of H' is obtained from the triangulation of H by adding, for each d-face F in H_v, the $(d+1)$-simplex G_{Fs} formed by F and $\lambda(s)$. The simplices $\{G_{Fs}\}$ fill out the volume between H and H' in the obvious way. Each simplex G_{Fs} has a single *parent* d-face F in H_v, possibly *child* d-faces in H_s, and possibly *sibling* d-faces shared with other simplices $G_{F's}$ just added. The triangulation can be represented using the data structure described in section 4.1, in one dimension higher. In fact, the triangulation simultaneously serves as the search data structure and as the representation of the boundary of the current convex hull.

Given a new site s, depth-first search in the triangulation of H can be used to find a face of H seen by $\lambda(s)$. The triangulation should be thought of as a directed graph. The nodes are the $(d+1)$-simplices and arcs correspond to d-faces. There is an arc directed from parent to child, and an arc in each direction between siblings. All of the faces of the initial simplex H_0 are child faces, directed away from H_0. Depth-first search starts at H_0; the search proceeds from a simplex G through a child or sibling pointer if $\lambda(s)$ is beyond the corresponding d-face of G, that is, if the hyperplane through the face separates G from $\lambda(s)$. The search stops if $\lambda(s)$ is beyond some d-

face of a simplex G and there is no opposite simplex; this happens only if the d-face is part of the boundary of the current hull H. There is one subtlety to the search: if the search enters a simplex, the search should continue only if the simplex is *active*, that is, if $\lambda(s)$ and the simplex are on the same side of the hyperplane through the parent d-face of the simplex. It may happen that an inactive simplex is entered through a sibling pointer; the search should backtrack immediately to save unnecessary work (and to make the analysis below go through).

Why is the search guaranteed to find a d-face of H seen by $\lambda(s)$? Choose a point h in the initial simplex H_0 so that the segment from h to $\lambda(s)$ intersects only d- and $(d+1)$-simplices, and those in vertices and open segments, respectively. The sequence of $(d+1)$-simplices intersected by the segment in order from h to $\lambda(s)$ can be traversed using sibling d-faces and d-faces from parent to child, since $\lambda(s)$ is beyond every intermediate convex hull. Any $(d+1)$-simplex in the sequence is active because the hyperplane through its father face separates h from some point on the segment and hence separates h from $\lambda(s)$. It follows that $\lambda(s)$ and the $(d+1)$-simplex are on the same side of the hyperplane. Clearly the last d-face intersected by the segment is a d-face of H seen by $\lambda(s)$, so there is at least one path through which the depth-first search can succeed.

Theorem 4.3 *Let S be a set of n sites in E^d. The expected running time and space of the random-incremental algorithm are in $O(n^{\lceil d/2 \rceil})$ if $d \geq 3$. If $d = 2$, then the expected running time is in $O(n \log n)$ and the expected space is in $O(n)$. This bound holds for the worst-case set of sites; the expectation is over the permutation used to order insertions.*

Proof. Let $\mathcal{B}$ be the subsets of S of size $d+1$. An element $B \in \mathcal{B}$ has *scope* k if there are exactly k sites in the interior of the circumsphere of B. Let b_k and $b_{\leq k}$ be the number of elements of $\mathcal{B}$ of scope k and of scope between 0 and k, respectively. Clearly, if B has scope 0, then cell(B) is a Delaunay d-cell; hence b_0 is the number of Delaunay d-cells and b_0 is in $O(n^{\lceil d/2 \rceil})$. A fundamental and not especially obvious fact[14,15] is that for $k \geq 1$,

$$b_{\leq k} \in O(n^{\lceil d/2 \rceil} k^{\lceil (d+1)/2 \rceil}).$$

Surprisingly, good bounds on b_k are not known.

The sites are inserted in some random order. We first bound the expectation of the total number of d-faces that appear on any intermediate convex hull. For $B \in \mathcal{B}$ of scope k, cell$(\lambda(B))$ appears on some intermediate convex hull exactly if the $d+1$ elements of B appear in the random order before any of the k elements lying inside the circumsphere of B. The probability of this happening is

$$p_k = \frac{(d+1)!k!}{(k+d+1)!} = \frac{(d+1)!}{(k+1)(k+2)\cdots(k+d+1)}.$$

Thus the expected number of d-faces is

$$\begin{aligned}\sum_{k=0}^{n-(d+1)} p_k b_k &= b_0 + \sum_{k=1}^{n-(d+1)} p_k(b_{\leq k} - b_{\leq k-1}) \\ &= b_0 + \sum_{k=1}^{n-(d+1)} (p_k - p_{k+1})b_{\leq k} \\ &\in O(n^{\lceil d/2\rceil}) + \sum_{k=1}^{n-(d+1)} O(k^{-(d+2)}n^{\lceil d/2\rceil}k^{\lceil (d+1)/2\rceil}) = O(n^{\lceil d/2\rceil}).\end{aligned}$$

The space usage of the algorithm is determined by the total number of d-faces and has expectation $O(n^{\lceil d/2\rceil})$.

Now we bound the total time spent traversing the search data structure for all sites. Consider the time spent at a single non-root simplex for a single site s. We charge the time spent leaving the simplex through child and sibling faces to the parent face $F = \text{cell}(\lambda(B))$ of the simplex. The total charge is constant, since there are only d child and sibling faces. Such a charge happens only if the simplex is active, that is, if $\lambda(s)$ and the simplex are on the same side of the hyperplane through the parent face, or equivalently if s is inside the circumsphere of B. Hence the total charge to a parent face $F = \text{cell}(\lambda(B))$ is a constant times the scope of B. The expected time spent traversing the data structure is at most the sum over all potential faces of the probability that the face gets created times the total possible charge to the face:

$$\sum_{k=1}^{n-(d+1)} k p_k b_k.$$

If $d \geq 3$, essentially the same analysis as above shows that the sum is $O(n^{\lceil d/2\rceil})$. The case $d=2$ is special: the sum becomes

$$\sum_{k=1}^{n-(d+1)} O(k^{-(d+1)}n^{\lceil d/2\rceil}k^{\lceil (d+1)/2\rceil}) = O(n\sum_{k=1}^{n-3} k^{-1}) = O(n\log n).$$

In either case it is easy to see that the sum bounds the total expected running time of the algorithm. □

Other analyses of this algorithm are possible. Clarkson, Mehlhorn and Seidel[14,16] give an elegant self-contained analysis that does not require the bound on the magic value $b_{\leq k}$; Boissonnat *et al.*[7] give another. The analysis given here follows the outline given by Guibas, Knuth, and Sharir[35] for a random-incremental diagonal-flipping algorithm in two dimensions. The original random-incremental algorithm for convex hulls is due to Clarkson and Shor.[15] That algorithm maintains a "conflict graph" that has an edge connecting each uninserted site with each ridge of the current hull that would be destroyed by the addition of the new site. The conflict graph makes finding the faces visible from an uninserted site trivial; the work is in maintaining the conflict graph. The data structure described here is very similar to one suggested by Boissonnat and Teillaud[6]; the idea of using intermediate Voronoi diagrams to aid searching can be traced to Green and Sibson.[33]

Chazelle[12] shows that it is possible to 'derandomize' this algorithm, that is, to choose deterministically an insertion order that guarantees the same asymptotic running time. Unfortunately, the algorithm that computes the insertion order is quite complicated.

As in the case of the incremental algorithm, the worst-case bound of theorem 4.3 is unduly pessimistic. Suppose the set of sites S has the property that the expected size of the Delaunay triangulation of a random subset $R \subseteq S$ is linear in $|R|$. A set of points drawn uniformly in the unit sphere satisfies this property, as discussed in section 2.3. Then the expected running time of the random incremental algorithm is $O(n \log n)$.[14,16]

4.5 The plane-sweep algorithm

The plane-sweep method constructs a cell complex in the plane by observing the intersection of the cell complex with a line, the 'sweepline', as the line sweeps across the plane. For a general discussion of the plane-sweep method, see for example the book by Preparata and Shamos.[53] We use the plane-sweep method to construct the Voronoi diagram or Delaunay triangulation of a set of points in two dimensions.

It is perhaps not immediately obvious how to use the plane-sweep method to compute a Voronoi diagram V efficiently in two dimensions. Suppose the sweepline is parallel to the x-axis and is moving upwards, that is in the plus y-direction. The vertices of V can be split into two categories: those with two edges incident from below and one from above, and those with one edge incident from below and two from above. It is easy to predict the first kind of vertex, because the two edges incident from below are adjacent on the sweepline data structure. It is not so easy to predict the second type of event. In effect, it is caused by an edge intersected by the sweepline and a site lying above the sweepline. However, it is undesirable to consider all quadratically many such pairs.

In fact, there is a plane-sweep algorithm that computes the Voronoi diagram of a set S of n sites in the plane in time $O(n \log n)$.[29] It is convenient to describe the algorithm as producing the Delaunay triangulation D. In addition to the general position assumption, assume that no two sites of S lie on a common horizontal line and that no site is the topmost point of the circumcircle of any Delaunay triangle.

Let l_y be the horizontal line with y-coordinate y (l_y will be the sweepline). Let $y_{\min}$ be the minimum y-coordinate of a site in S and assume $y > y_{\min}$. For a point $p \in l_y$, grow a circle C_p with topmost point p until it touches a site in S. In general, C_p touches a single site s, and in fact for every point p in some open interval $I_y(s) \subseteq l_y$, C_p touches exactly s. See figure 11. At an endpoint p of $I_y(s)$, C_p also touches a site t with interval $I_y(t)$ also incident to p; possibly C_p touches another site as well. The sequence of intervals $I_y(s_1), I_y(s_2) \ldots, I_y(s_k)$ along l_y determines a path of Delaunay edges $F_y = s_1s_2, s_2s_3, \ldots, s_{k-1}s_k$. F_y need not be simple: a vertex can

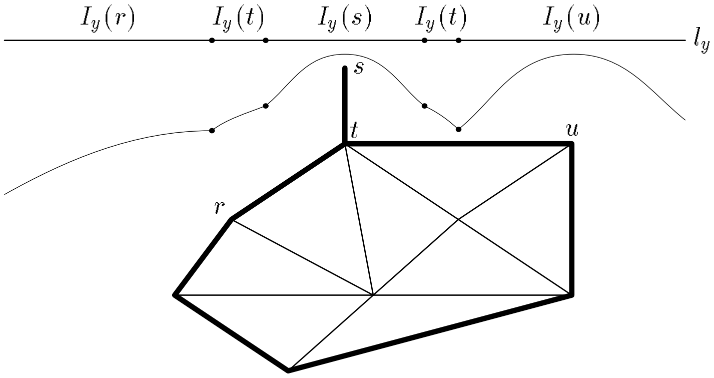


Figure 11: The heavy path is F_y; the curved path is the center of circles C_p for $p \in l_y$.

appear arbitrarily many times (though an edge can appear at most twice). F_y can be thought of as the 'frontier' of the set of Delaunay cells that have some circumcircle with interior below l_y.

What happens as y increases and l_y moves upwards? Path F_y can change in two ways. First, l_y can become tangent to the top of the circumcircle of three sites s, t, u that form two consecutive edges st and tu along F_y. Edges st and tu form a concave angle in the sense that s, t, u and the topmost point of the circle appear in that order around the circle. In this case, the interval $I_y(t)$ disappears and edge su replaces edges st and tu along F_y. Notice that stu must be a Delaunay triangle and su a Delaunay edge. Second, l_y can meet some site s. If s were not a site, then point s would lie in some interval $I_y(t)$ with intervals $I_y(u_1)$ before and $I_y(u_2)$ after. Since s is a site, for slightly larger y there will be an open interval $I_y(s)$, and the subpath of F_y consisting of u_1t, tu_2 is replaced with u_1t, ts, st, tu_2. Notice that st is a Delaunay edge, and that this is the first time site s has appeared. Eventually all sites are passed and path F_y forms the convex hull of S; then there are no further changes to F_y.

Using this analysis, a plane-sweep algorithm can be designed that computes the Delaunay triangulation. The algorithm maintains the current path F_y and an event queue ordered by y-coordinate. The two events are passing a site and passing the top of a circle. Initially, the sweep line is at $y = y_{\min}$, the event queue contains all sites, and the path F_y consists only of the lowest site. An invariant assertion of the algorithm is that the event queue contains an event for each circle through sites s, t, u so that edges st and tu are consecutive on F_y and form a concave angle. If the next event is to pass the top of circle formed by edges st and tu, then the action is to replace these edges with su, and to record stu as a Delaunay cell and tu as a Delaunay edge. If the next event is a site s, then the site t to be connected to s must be discovered. Site t can be found by binary search in path F_y; the necessary primitive decides if s_x lies to the left or right of the topmost point of a circle tangent to l_y and touching two sites u and v so that uv is an edge of F_y. Once t is found, the path is augmented with edges ts and st, and edge st recorded as a Delaunay edge.

For either event, path F_y changes, so insertions and deletions to the event queue are necessary to maintain the invariant. However, since the changes to F_y are local, only a constant number of event queue operations are required.

Theorem 4.4 *The sweepline algorithm can compute the Delaunay triangulation of n sites $S \subset E^2$ in time $O(n \log n)$ and $O(n)$ space.*

Proof. The path data structure can be implemented using a balanced binary tree, so that insertions, deletions, and searches each take time $O(\log n)$. Similarly, the event queue can be implemented as a heap, so insertions, deletions, and determining the next event also each take time $O(\log n)$. The number of events is equal to the number of sites plus the number of Delaunay triangles, which is $O(n)$. Each event uses only a constant number of operations on the path data structure and the event queue, so the total running time is $O(n \log n)$. The total space is determined by the maximum length of the path F_y, which is linear, since the number of edges in the triangulation is linear. □

There is an interpretation in three dimensions that may clarify the algorithm. We sketch the interpretation here; Guibas and Stolfi[37] discuss it at greater length. Call the plane spanned by the x- and y-axes the base plane, as usual, and now let the vertical direction be the z-direction. Put a three-dimensional cone $C(s)$ over each site s; the height of the cone over a point x in the plane is equal to the distance from x to s. Paint each cone a different opaque color; then an observer sitting below the plane looking up sees the lower envelope of the cones. Clearly the vertical projection of the lower envelope of the cones on the base plane is the Voronoi diagram.

Consider a plane P that makes the same angle with the base plane as the cones and that intersects the base plane in a line l parallel to the x-axis. The intersection of P with a cone projects onto the base plane as a parabola; the intersection of P with the lower envelope of the cones projects to a sequence of a parabolic arcs. In fact, the sequence is just the locus of centers of the circles C_p, as p varies along l. See figure 11.

Now imagine sweeping the plane P in the positive y-direction, so that l moves in the positive y-direction as well. For small motions of P, P intersects the same sequence of cones on the lower envelope; however, the parabolic arcs obtained by projecting on the base plane become less sharply curved. The intersection of P and the lower envelope can change combinatorially in two ways. First, l can hit a new site and P the corresponding cone; notice that in fact P becomes tangent to the cone. After a slight additional motion of l and P, the new cone yields a new parabolic arc in the projection of the intersection of P and the lower envelope. Second, P can pass through the point of intersection of three cones on the lower envelope; this corresponds to the sweepline discovering a vertex of the Voronoi diagram (or a Delaunay circle). One of the cones now disappears from the intersection of the lower envelope and P, and the projection onto the base plane loses a parabolic arc.

What happens if the set of sites S lies in three dimensions? One might hope that a 'space-sweep' algorithm would be output-sensitive, that is, its running time would be approximately proportional to the number of Delaunay tetrahedra. The algorithm does generalize to three dimensions. It is perfectly possible to define the 'frontier' F_z as the set of Delaunay cells that have a circumsphere touching a sweep plane parallel to the xy-plane at height z. F_z can be given the structure of a plane, just as in two dimensions the frontier has a linear structure. There are now three kinds of events: discovering a new site (and connecting the site by a Delaunay edge to a previously discovered site), discovering a triangle, and discovering a tetrahedron. The first event is of course easy to predict; finding the connecting edge requires point location in a dynamic planar subdivision, but there are reasonably efficient algorithms for that. The third event can also be predicted, because at least two triangles sharing a common edge are involved. The difficult event to predict is the creation of a triangle. Conceivably the current frontier has a vertex with $O(n)$ Delaunay edges incident to it. The next Delaunay triangle may be formed from two of those edges. However, there are $O(n^2)$ possible pairs, and it seems necessary to consider all of them. Hence, as far as is known, space-sweep algorithms require at least quadratic complexity, even if the Delaunay triangulation has linear complexity.

4.6 Other algorithms

The divide-and-conquer algorithm was the first worst-case optimal algorithm for Voronoi diagrams in two dimensions.[59] It splits the set of sites into two halves of approximately the same size with a vertical line, recursively computes the Voronoi diagram of the two halves, and then merges the two triangulations. The merge takes linear time in the worst case, so the worst-case running time of the algorithm on n sites is $O(n \log n)$. Guibas and Stolfi give a careful description of the algorithm in terms of the Delaunay triangulation.[36]

If the set of sites is uniformly distributed in the unit square, then the expected running time can be improved. Bentley, Weide, and Yao[5] show that a combination of bucketing techniques and any $O(n \log n)$ algorithm leads to an algorithm with expected running time $O(n)$. Dwyer shows that a simpler algorithm, essentially first splitting with a vertical line, then a horizontal line, has expected running time $O(n \log \log n)$. This is further improved to $O(n)$ by Katajainen and Koppinen.[41]

A Delaunay triangulation can always be computed by lifting the set of sites and using a convex hull algorithm. One alternative to the beneath-beyond algorithm is the 'gift wrapping' algorithm.[53] The gift-wrapping algorithm takes time $O(ns)$, where n is the number of points and s is the number of facets on the convex hull of the set of points. Hence, in the worst case it is slower than the beneath-beyond algorithm. However, Dwyer[21] is able to show that if the set of sites is uniformly distributed in a sphere, then a version of the gift-wrapping algorithm applied to the lifted sites takes linear time. The linear time results from the use of bucketing techniques in addition to the linear expected combinatorial complexity of the Delaunay triangulation. A second convex hull algorithm is due to Seidel[58]; it takes time $O(n^2 + s \log n)$, where s and n are as before. Conceivably there are situations where its use could lead to an efficient algorithm for Delaunay triangulations.

4.7 The real RAM and the general position assumption

The 'real RAM' model and the general position assumption for sites considerably simplify the description of geometric algorithms. However, the real RAM is unimplementable and the set of sites of interest may not be in general position. It is not clear how best to proceed without these assumptions; this section discusses some of the issues. This discussion is appropriate for all geometric algorithms, not just Delaunay triangulation algorithms.

A plausible approach is to use integers instead of reals. Thus an input point is specified by a vector of integers; scaling of the input may be necessary to approximate the input adequately on the integer grid. Most of the Delaunay triangulation algorithms can be implemented using the geometric primitives in section 4.1 (the sweepline algorithm has a different set of primitives). Each of these primitives is a sign-evaluation of a determinant formed from the coordinates of the input points and can be evaluated using integer arithmetic. The issue is the bit complexity of the arithmetic. As far as is known, it is necessary to evaluate the determinant in order to determine its sign. A coarse estimate of the bit complexity of the determinant evaluation can be obtained by viewing the determinant as a polynomial in the input coordinates. If a polynomial has degree d and the variables have bit length b, then approximately bd bits are required to evaluate the polynomial. In dimension d, the orientation test has degree $d+1$ and the incircle test degree $d+2$. Thus with only standard 32-bit integer arithmetic, in dimension $d = 2$ the input coordinates can have bit length at most $b = 7$, i.e. the input sites have to lie on a 128×128 grid. It is reasonable to take the dimension d to be constant, so extended-precision integer arithmetic adds only a constant multiplicative factor of overhead.

Unfortunately, at least with naive implementations of extended-precision integer arithmetic, the constant is large. Consider Gaussian elimination to evaluate the determinant of a $d \times d$ matrix with integer entries. The usual estimate is that Gaussian elimination takes $O(d^3)$ arithmetic steps. However, to implement Gaussian elimination using integers without division, the bit length of matrix entries increases. A better estimate of the number of b-bit primitive arithmetic operations needed to evaluate a $d \times d$ determinant with b-bit entries is exponential in d.[42] An alternative is to use modular arithmetic over a collection of finite fields, so division is possible, and then use the Chinese remainder theorem to reconstruct the determinant.[44] This results in about $O(d^4)$ b-bit arithmetic operations to evaluate a $d \times d$ determinant with b-bit entries. No extensive comparison of determinant evaluation methods appears to be available, at least in the context of geometric algorithms.

It is a considerable simplification to allow only integer coordinates. Rational coordinates arise naturally; for example, the coordinates of the point of intersection of two line segments are rational even if the endpoint coordinates are integer. However, the bit-complexity estimates go up dramatically if the input coordinates are rational. Even a single addition can double bit complexity: the appropriate measure of the bit-complexity of p/q is the sum of the bit lengths of p and q, thus the bit-complexity of $p/q + r/s = (ps + qr)/(qs)$ is the sum of the bit-complexities of p/q and r/s.

Exact arithmetic has an advantage that certain symbolic perturbation schemes guarantee that the input can be treated as if it were in general position.[25,27,63,64] Edelsbrunner and Mücke[25] suggest a scheme called 'simulation of simplicity;' they work out the details for algorithms that use the orientation test and the incircle test. Suppose that the input sites are $S = \{s_1, s_2, \ldots, s_n\} \subset E^d$, where $s_i = (s_{i1}, s_{i2}, \ldots, s_{id})$. Imagine perturbing s_{ij} by an amount $\epsilon^{2^{id-j}}$, for ϵ very small. For sufficiently small $\epsilon > 0$ the resulting set of sites is guaranteed to be in general position. Edelsbrunner and Mücke show that this perturbation can be simulated symbolically, without any computation involving ϵ or even knowing ϵ. For the orientation test on $d+1$ points, they give a sequence of determinants constructed using the coordinates of the unperturbed points. They show that the sign of the first nonzero determinant in the sequence is the same as the sign of the determinant obtained from the perturbed points. The first determinant in the sequence is just the determinant obtained from the unperturbed points, so overhead is incurred only in the case that it is zero, i.e., in the degenerate case. A possible disadvantage of the method is the length of the sequence of determinants; for the incircle test in dimension 2, the length is already 15. At least in principle the determinants can be generated as needed. A second disadvantage is that there is no control over the perturbation. Thus a nonextreme site on the convex hull of the triangulation may be perturbed to become extreme, in effect adding a flat triangle.

The alternative to exact arithmetic is floating point arithmetic, and the bit-complexity issue no longer arises. However, the general position assumption becomes much more stringent. Standard error analysis techniques from numerical analysis can be applied to the geometric primitives.[17] Consider the orientation-test determinant obtained from a set of sites $s_1, \ldots, s_{d+1}$. In floating point arithmetic using, say, Gaussian elimination, it is possible to compute an approximate determinant $\hat{\Delta}$ and an error bound $\delta > 0$. If $|\hat{\Delta}| \geq \delta$, then the sign of $\hat{\Delta}$ is correct; if $|\hat{\Delta}| < \delta$, then the true sign is unknown, but there are slightly perturbed sites $s'_1, \ldots, s'_{d+1}$ so that the true determinant is zero. The 'slight perturbation' moves sites by an amount depending upon the relative error ϵ of floating point arithmetic, the dimension d, and the distance of the sites from the origin. Hence if the set S of sites is in 'sufficiently general' position, that is, no determinant changes sign when the sites are slightly perturbed, then the floating-point implementation is correct.

What happens if the set of sites is not in 'sufficiently general' position? A naive implementation might interpret the 'can't tell' answer as implying that the determinant can be taken as zero. Unfortunately, this can be disastrous: choose three sites s_1, s_2, s_4 in the plane that form a right angle, and then add a site s_3 very close to s_2. The implementation might take s_1, s_2, s_3 as collinear and s_2, s_3, s_4 as collinear, implying the collinearity of s_1, s_2, s_4, which is manifestly false. Such problems can lead to catastrophic failure of algorithms implemented naively with floating point arithmetic.

There has been some work applying the techniques of numerical analysis to geometric algorithms as a whole.[30,38,49] Fortune[31] shows that the flipping algorithm can be implemented using floating-point arithmetic to produce an 'approximate Delaunay triangulation'. This means that the triangulation has almost-empty circumcircles: any site lying inside a circumcircle is only 'slightly' inside. The meaning of 'slightly' depends upon the relative error ϵ of floating point arithmetic, the number n of sites, and the distance of the sites from the origin, but not on any geometric properties of the configuration of the sites.

4.8 Implementation issues

It is not trivial to obtain a reliable and complete implementation of any of the Delaunay triangulation algorithms. This section discusses a few implementation issues, offers some advice, and reports experience with four of the algorithms.

A fundamental implementation decision is the choice of exact or floating point arithmetic. This choice likely influences every other implementation decision. Exact arithmetic is the only known way to obtain a completely reliable implementation, at least in general. The disadvantage is the inconvenience and potential cost of exact arithmetic. Edelsbrunner and Mücke[25] use multiple-word integer arithmetic to implement orientation-test primitives on points on an integer grid. They report approximately a factor of 10 degradation in performance as compared to using single-word arithmetic on points on an integer grid drastically reduced in size. They suggest the use of special-purpose hardware to accelerate integer determinant calculations. Karasick, Lieber, and Nackman[42] implement the divide-and-conquer algorithm in two dimensions using adaptive-precision rational arithmetic. Their effort was principally addressed at accelerating the performance of rational arithmetic. Their ultimate implementation was at least three orders of magnitude faster than a naive implementation of rational arithmetic, and only a factor of about 4 slower than floating point arithmetic.

An implementation using floating point arithmetic is not guaranteed to be completely reliable (except as cited before). However, floating point arithmetic is the typical implementation choice and may be reliable enough in some circumstances. A minimal requirement is to raise a flag if some primitive cannot reliably deduce the correct answer. (This is often not done, perhaps because the required interval analysis techniques are not understood.) If some primitive does give an ambiguous answer, a simple trick is to repeat the algorithm after a small random perturbation of the input data. If perturbation is not acceptable, an alternative is to examine each use of a primitive test. Some ambiguous answers truly are 'don't cares' and are easily handled; others may require localized use of higher precision. Such analysis is still a black art.

Whether exact or floating point arithmetic is required, it is better to construct the Delaunay triangulation than the Voronoi diagram. One reason is to avoid manipulating computed values: in exact arithmetic, Voronoi vertices are rationals with high bit complexity; in floating point arithmetic, Voronoi vertices may be poorly determined if the defining sites are close to being affinely dependent. Another reason is that it is simpler to manipulate a cell complex with regular bounded cells than one

	lines	d	n=500	n=1000	n=5000	n=10000	n=50000
Flipping	300	2	2.1	4.5	26.2	59.3	316.9
Divide-and-conquer	900	2	2.0	3.8	21.5	42.3	222.0
Plane sweep	900	2	1.2	2.5	12.1	24.1	128.0
Random-incremental	1500	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 12: Performance of Delaunay triangulation algorithms, in seconds.

with irregular unbounded cells. The Voronoi diagram can be obtained in linear time from the completed Delaunay triangulation, if required.

The worst-case $O(n^{\lceil d/2 \rceil})$ complexity of the Delaunay triangulation should not be taken too seriously. The linear complexity $c_d n$ suggested by the probabilistic estimates of section 2.3 is more realistic in practice. However, the constant c_d indeed does grow exponentially with d, so Delaunay triangulations in dimension 5 or bigger should be approached with caution.

Figure 12 gives timings for various algorithms. All algorithms are implemented in the C programming language and run on a VAX 8550. The primitives are implemented using floating point arithmetic, in some cases with heuristics to improve reliability. The input points are chosen in the unit cube using the library pseudo-random number generator. Both the timings and the number of lines of source code should be taken as suggestive only, since the implementations were done by different people and with different levels of optimization. For example, an unoptimized version of the plane-sweep algorithm requires only half as many lines of code but is considerably slower. The flipping algorithm sorts points by angle rather than by coordinate. The divide-and-conquer algorithm takes advantage of the uniform distribution of input points by using both horizontal and vertical splitting directions.[20] The plane-sweep algorithm uses bucketing for both the priority queue and sweepline data structure, giving good performance on uniform data. An important optimization for the random-incremental algorithm is to use a dot-product rather than an orientation test in the search data structure; the child face normal is calculated when a triangle is added to the triangulation. The running times include input and output; at least for the plane-sweep algorithm, the time for conversion between character and floating point representations is comparable to the time actually required to compute the Delaunay triangulation.

In two dimensions, all of the algorithms are reasonable. The times are sufficiently close that any one of the algorithms could be made the fastest by sufficient optimization. The times are also sufficiently small that optimization may be irrelevant. In higher dimension, the random-incremental algorithm is attractive because it is relatively simple to implement and has good running time, both worst-case and expected-case. One alternative is to use the incremental algorithm with a different search data structure to find a face seen by a new lifted site; another alternative is the variant of gift-wrapping suggested by Dwyer.[21]

For large problems storage may be more relevant than running time. The 5000 point example in four dimensions required roughly 70 megabytes of main storage. Because of paging overhead, the actual clock time for the example was about a factor of 10 larger than the running time. The implementation of the random-incremental algorithm was not optimized to minimize storage. The search data structure is the predominant use of storage. A variant of the random-incremental algorithm that uses a conflict graph rather than than search data structure might require much less storage.[15,14]

Appendix 1: Definitions from the theory of polyhedra

We review some basic definitions from the theory of convex polyhedra. The definitions are more or less standard, except perhaps for the definition of a cell complex as consisting of relatively open sets. For more extensive discussion, see books by Brøndsted,[10] Edelsbrunner,[24] and Grunbaum.[34]

E^d is d-dimensional euclidean space. A *k-flat* is an affinely closed subset of E^d of dimension k; *lines, planes*, and *hyperplanes* are flats of dimension 1,2, and $d-1$, respectively. A set of points in E^d is *affinely independent* if no $k+2$ points lie on a common k-flat, for $k<d$. An *open halfspace* is the set of points to one side of a hyperplane; a *closed halfspace* includes the hyperplane. A hyperplane *separates* two sets if one set is contained in one of the closed halfspaces and the other is contained in the complementary open halfspace. A *convex polyhedron* is the intersection of a finite number of closed halfspaces; if it is bounded it is a *convex polytope*. A *cell* is the intersection of a finite set of flats and open halfspaces; equivalently a cell is the relative interior of a convex polyhedron. If $R \subseteq E^d$, then cell(R) is the relative interior of the convex hull of R. A *simplex* is cell(R) for an affinely independent set of points R; thus a two-dimensional simplex is the interior of a triangle and a three-dimensional simplex is the interior of a tetrahedron. A *sphere* in E^d is the set of points in E^d at a fixed distance from its center. A point of E^d is either inside, on, or outside a sphere; a *circumsphere* of a set $R \subset E^d$ is a sphere with all points of R on the sphere.

A closed halfspace *supports* a cell or convex polyhedron P if it contains P and its bounding hyperplane intersects the closure of P. A hyperplane *supports* P if one of its closed halfspaces supports P. A *face* of P is the relative interior of the intersection of a hyperplane supporting P with the closure of P. Basic facts are that a face is a cell and that a face of a face of P is a face of P. If F is a face of G, then F and G are *incident*. A face is a *k-face* if its affine closure has dimension k; *vertices, edges, ridges*, and *facets* are faces of dimension 0,1,$d-2$, and $d-1$, respectively. An *extreme point* of a polyhedron is a vertex of the polyhedron. A face F of a convex polyhedron P is *visible* from a point x not in P if every hyperplane supporting P through F separates x from P; F is a *horizon face* from x if there is a hyperplane supporting P containing x and F; F is *invisible* from x if no hyperplane supporting

P through F separates x from P.

A *cell complex* is a finite collection of pairwise disjoint cells so that the face of every cell is in the collection. Two k-cells of a cell complex are *opposite* if they have a common $(k-1)$-face. A *triangulation* T of a finite point set S is a cell complex whose union is the convex hull of S and whose vertex set is S. Any cell of a triangulation is cell(R) for some $R \subseteq S$. This is a convenient but nonstandard definition because cells are not required to be simplices. A *proper triangulation* is a triangulation all of whose cells are simplices. Any triangulation can be *completed* to a proper triangulation by appropriately subdividing nonsimplicial cells. A convex polyhedron can be partitioned into a cell complex, in fact, a triangulation: one cell is the relative interior of the polyhedron and the other cells are its faces.

Appendix 2: Proof of theorem 2.1

We first show that if $V(R)$ and $V(R')$ are both Voronoi cells, then $V(R')$ is a face of $V(R)$ iff $R \subset R'$. Suppose $R \subset R'$. For $r \in R'$, $s \in S-R'$, let H_{rs} be the hyperplane equidistant from r and s with positive halfspace containing R'. Let I be $\cap H_{rs}$ and Q the closure of $V(R)$. We can choose a hyperplane H so that $Q \cap I = Q \cap H$ by choosing H through B avoiding the intersection of the positive halfspaces and the intersection of the negative halfspaces. Thus the relative interior of $Q \cap I$ is a face. However the relative interior of $Q \cap I$ is $V(R')$, since $V(R')$ is a relatively open subset of I, and any point of Q is equidistant from all sites in R'. The converse direction, that $V(R')$ a face of $V(R)$ implies $R \subset R'$, is easy: any point in a face of $V(R)$ is in the closure of $V(R)$, and hence is in $V(R')$ for some $R' \supset R$.

The first claim is that $V(R)$ is a cell complex with union E^d. Clearly, $V(R)$ is a cell and any point of E^d lies in some Voronoi cell. If F is a face of $V(R)$, then any point $x \in F$ is on the boundary of $V(R)$, so $x \in V(R')$ for some $R' \supset R$. Therefore $F = V(R')$ since $V(R')$ is a face and x lies in exactly one face of $V(R)$.

The second claim is that the Delaunay triangulation is a triangulation. We must show that Delaunay cells are pairwise disjoint, that the face of a Delaunay cell is a Delaunay cell, and that the union of the Delaunay cells is the convex hull of S. The disjointness of distinct Delaunay cells $D(R)$ and $D(R')$ follows by considering circumspheres C and C' of R and R', respectively, and observing that either C and C' have disjoint interiors, or if not, then $R \cap R' \subset C \cap C'$ and by the empty-sphere property, $R-R'$ and $R'-R$ lie on opposite sides of the hyperplane through $C \cap C'$. A face of a Delaunay cell $D(R')$ is cell(R) for $R = R' \cap H$ and H a hyperplane supporting the convex hull of R'; since R lies on a hyperplane, it is easy to see that the empty circumsphere of R' can be perturbed to be an empty circumsphere of R, and so cell(R) is a Delaunay cell. Let q be a point in the convex hull of S; we show that q is in some Delaunay cell. Assume that there is some Delaunay d-cell $D(R)$ (if not, then S lies in some lower-dimensional flat, and the proof works in that flat). Choose a point $p \in D(R)$ so that pq avoids all k-cells for $k < d-1$ and so that pq intersects any Delaunay $(d-1)$-cell in at most a point. The sequence of Delaunay d-

and $(d-1)$-cells intersected by pq in order from p to q can end only at q: a Delaunay d-cell intersecting pq has a $(d-1)$-face that is also Delaunay and further along pq; a Delaunay $(d-1)$-cell intersecting pq must have an incident Delaunay d-cell further along pq, since q is in the convex hull of S. Hence q is in the closure of some Delaunay d-cell, and in some Delaunay cell.

The third claim is duality. Suppose $V(R)$ and $V(R')$ are both Voronoi cells, so both $D(R)$ and $D(R')$ are Delaunay cells. Then all the following steps are easy or have been established: $V(R')$ is a face of $V(R)$ iff $R \subset R'$ iff $R = R' \cap H$ for some hyperplane H supporting the convex hull of R' iff $D(R)$ is a face of $D(R')$.

The fourth claim is an easy exercise.

Acknowledgements.

I would like to thank Jon Bentley, Paul Chew, Ken Clarkson, Michael Dillencourt, Rex Dwyer, Herbert Edelsbrunner, John Hobby, Allan Wilks, and Margaret Wright for helpful comments.

References

1. P.F. Ash, E.D. Bolker, Recognizing Dirichlet tessellations, *Geometriae Dedicata* **19**:175–206, 1985.
2. P.F. Ash, E.D. Bolker, Generalized Dirichlet tessellations, *Geometriae Dedicata* **20**:209–243, 1986.
3. F. Aurenhammer, Recognizing polytopical cell complexes and constructing projection polyhedra, *J. Symbolic Computation 3*:249–257, 1987.
4. F. Aurenhammer, Voronoi diagrams – a survey of a fundamental geometric data structure, *ACM Computing Surveys* **23**:345–405, 1991.
5. J. Bentley, B. Weide, A. Yao, Optimal expected-time algorithms for closest-point problems, *ACM Trans. Math. Software* **6**, 563–580, 1980.
6. J.-D. Boissonnat, M. Teillaud, An hierarchical representation of objects: the Delaunay tree, *Proc. 2nd Ann. Symp. Comp. Geom.* 260–268, 1989.
7. 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.
8. A. Bowyer, Computing Dirichlet tessellations, *The Comp. Journal* **24**(2):162–166, 1981.
9. E. Brisson, Representing geometric structures in d-dimensions: topology and order, *Proc. of the Fifth Annual Symp. on Comp. Geom.*, pp. 218–227, 1989.
10. A. Brøndsted, *An Introduction to Convex Polytopes*, Springer-Verlag, Berlin, 1983.

11. K.Q. Brown, Voronoi diagrams from convex hulls, *Information Processing Letters* **9**:223–228, 1979.
12. B. Chazelle, An optimal convex hull algorithm and new results on cuttings, *Proc. 32nd Ann. Symp. Found. of Comp. Sci.* 29–38, 1991.
13. P. Chew, There are planar graphs almost as good as the complete graph, *J. Comp. System Science* **39**(2):205–219, 1989.
14. K.L. Clarkson, Randomized geometric algorithms, this volume.
15. K.L. Clarkson, P.W. Shor, Applications of random sampling in computational geometry, II, *Disc. Comp. Geom.* **4**:387–421, 1989.
16. K.L. Clarkson, K. Mehlhorn, R. Seidel, Four results on randomized incremental constructions, to appear, *Symp. Theor. Aspects of Comp. Sci.*, 1992.
17. G. Dahlquist, Å. Björck, N. Anderson, *Numerical Methods*, Prentice-Hall, 1974.
18. M.B. Dillencourt, Toughness and Delaunay triangulations, *Disc. and Comp. Geom.* **5**(6):575–601, 1990.
19. M.B. Dillencourt, I. Rivin, W.D. Smith, Combinatorial structure of Delaunay triangulations in the plane, manuscript, 1991.
20. R. A. Dwyer, A faster divide-and-conquer algorithm for constructing Delaunay triangulations, *Algorithmica* **2**(2):137–151, 1987.
21. R. A. Dwyer, Higher-dimensional Voronoi diagrams in linear expected time, *Disc. and Comp. Geometry*, **6**:343–367, 1991.
22. R. Dwyer, private communication.
23. H. Edelsbrunner, R. Seidel, Voronoi diagrams and arrangements, *Disc. and Comp. Geom.* **8**(1):25–44, 1986.
24. H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.
25. H. Edelsbrunner, E.P. Mücke, Simulation of Simplicity, a technique to cope with degenerate cases in geometric computations, *ACM Trans. Graphics* **9**(1):66–104, 1990.
26. H. Edelsbrunner, N. R. Shah, Incremental topological flipping works for regular triangulations, *Proc. of the Eighth Annual Symp. on Comp. Geom.* pp. 43–52, 1992.
27. I. Emiris, J. Canny, A general approach to removing degeneracies, *32nd Annual Symp. on Found. of Comp. Sci.* 405–413, 1991.
28. S. Fortune, A note on Delaunay diagonal flips, to appear, *Pattern Recognition Letters.*
29. S. Fortune, Sweepline algorithms for Voronoi diagrams, *Algorithmica* **2**:153–174, 1987.
30. S. Fortune, Stable maintenance of point-set triangulation in two dimensions,

30th Ann. Symp. on the Found. of Comp. Sci., 494–499, 1989.

31. S. Fortune, Numerical stability of algorithms for Delaunay triangulations, to appear, *International Journal on Computational Geometry with Applications.* An earlier version appeared in *Proc. of the Eighth Annual Symp. on Comp. Geom.* pp. 83–92, 1992.
32. K.R. Gabriel, R.R. Sokal, A new statistical approach to geographic variation analysis, *Systematic Zoology* **18**:259–278, 1969.
33. P.J. Green, R. Sibson, Computing Dirichlet tessellations in the plane, *Computer Journal*, **21**(22):168–173, 1977.
34. B. Grunbaum, *Convex Polytopes*, John Wiley and Sons, New York, 1967.
35. L.J. Guibas, D.E. Knuth, M. Sharir, Randomized incremental construction of Delaunay and Voronoi diagrams, *ICALP 90*, 414–431, Springer-Verlag, Berlin, 1990.
36. 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.
37. L. J. Guibas, J. Stolfi, Ruler, compass, and computer: the design and analysis of geometric algorithms, *Theoretical Foundations of Computer Graphics and CAD* 111–165, R.A. Earnshaw, ed., Springer-Verlag, New York, 1988.
38. C. Hoffmann, The problems of accuracy and robustness in geometric computation. *Computer* **22**:31–42, 1989.
39. B. Joe, Three dimensional triangulations from local transformations, SIAM J. Sci. Stat Computing **10**:718–741, 1989.
40. B. Joe, Construction of three-dimensional Delaunay triangulations using local transformations, Computer Aided Geometric **8:123–142**, 1991.
41. J. Katajainen, M. Koppinen, Constructing Delaunay triangulations by merging buckets in quadtree order, manuscript, 1987.
42. M. Karasick, D. Lieber, L. Nackman, Efficient Delaunay triangulation using rational arithmetic, *ACM Trans. Graphics* **10**(1):71–91, 1990.
43. J.M. Keil, C.A. Gutwin, The Delaunay triangulation closely approximates the complete euclidean graph, *Algorithms and Data Structures, Workshop WADS 89* 47–56, F. Dehne, J.-R. Sack, N. Santoro, eds, Springer-Verlag LNCS 382, Berlin, 1989.
44. D. E. Knuth, *The Art of Computer Programming, Vol.2, Seminumerical Algorithms*, Addison-Wesley, 1969.
45. C.L. Lawson, Software for C^1 surface interpolation, *Mathematical Software III* 161–194, J.R. Rice, ed., Academic Press, 1977.
46. P. Lienhardt, Subdivisions of n-dimensional spaces and n-dimensional generalized maps, *Proc. of the Fifth Annual Symp. on Comp. Geom.*, 228–236, 1989.

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

48. J.L. Meijering, Interface area, edge length and number of vertices in crystal aggregates with random nucleation, *Phillips Res. Rep* **8**:270–290, 1953.

49. V. J. Milenkovic. *Verifiable implementations of geometric algorithms using finite precision arithmetic*, Ph.D. Thesis, Carnegie-Mellon, 1988.

50. D.M. Mount, A. Saalfeld, Globally-equiangular triangulations of cocircular points in $O(n \log n)$ time, *Proc. of the Fourth Annual Symp. on Comp. Geom.*, 143–152, 1988.

51. T. Ohya, M. Iri, K. Murota, A fast Voronoi-diagram with quaternary tree bucketing, *Inf. Proc. Letters* **18**(4):227–231, 1984.

52. T. Ohya, M. Iri, K. Murota, Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms, *J. of the O.R. Res. Society of Japan* **27**:306–337, 1984.

53. F.P. Preparata, M.I. Shamos, *Computational Geometry*, Springer-Verlag, 1985.

54. V.T. Rajan, Optimality of the Delaunay triangulation in R^d, *Proc. of the Seventh Annual Symp. on Comp. Geom*, 357–363, 1991.

55. I. Rivin, W.D. Smith, Inscribable graphs, manuscript, NEC Corporation, 1991.

56. L.A. Santaló, *Integral Geometry and Geometric Probability*, Encyclopedia of Mathematics and its Applications, Addison-Wesley, 1976.

57. R. Seidel, The complexity of Voronoi diagrams in higher dimension, *Proc. of the 20th Allerton Conference on Communication, Control, and Computing*, 1982.

58. R. Seidel, Constructing higher-dimensional convex hull algorithms at logarithmic cost per face, *Proc. 18th Ann. Symp. Theory. Comp.* 404–413, 1986.

59. M. Shamos, D. Hoey, Closest-point problems, *Proc. 16th Ann. Symp. Found. Comp. Sci.* 151–162, 1975.

60. G.T. Toussaint, The relative neighborhood graph of a finite planar set, *Pattern Recognition* **12**(4):262–268, 1980. *The Comp. Journal* **24**(2):167-172, 1981.

61. D.F. Watson, Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes, *The Computer Journal* **24**(2):167–172, 1981.

62. P. Vaidya, A sparse graph almost as good as the complete graph on points in k dimensions, *Disc. Comp. Geom* **6**4:369–381, 1991.

63. C. Yap, Symbolic treatment of geometric degeneracies, *J. Symb. Comp.* **10**:349–370, 1990.

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