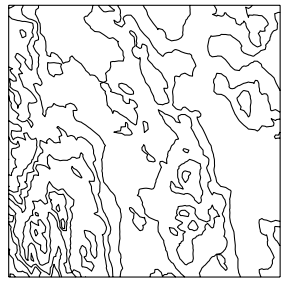
9 Delaunay Triangulations

Height Interpolation

When we talked about maps of a piece of the earth's surface in previous chapters, we implicitly assumed there is no relief. This may be reasonable for a country like the Netherlands, but it is a bad assumption for Switzerland. In this chapter we set out to remedy this situation.

We can model a piece of the earth's surface as a *terrain*. A terrain is a 2-dimensional surface in 3-dimensional space with a special property: every vertical line intersects it in a point, if it intersects it at all. In other words, it is the graph of a function $f : A \subset \mathbb{R}^2 \to \mathbb{R}$ that assigns a height $f(p)$ to every point p in the *domain*, A, of the terrain. (The earth is round, so on a global scale terrains defined in this manner are not a good model of the earth. But on a more local scale terrains provide a fairly good model.) A terrain can be visualized with a perspective drawing like the one in Figure 9.1, or with contour lines—lines of equal height—like on a topographic map.



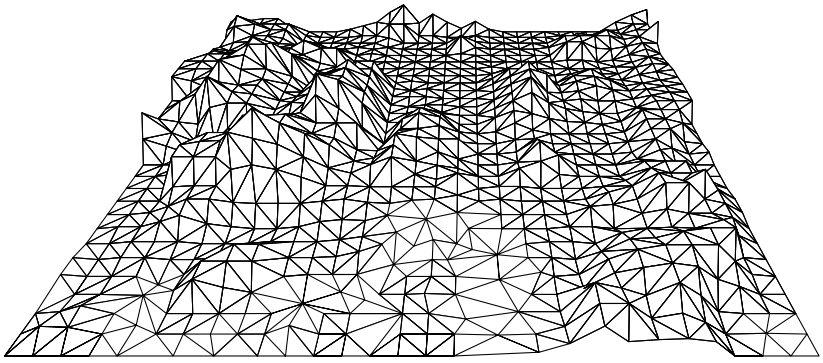
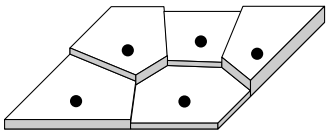


Figure 9.1
A perspective view of a terrain

Of course, we don't know the height of every point on earth; we only know it where we've measured it. This means that when we talk about some terrain, we only know the value of the function f at a finite set $P \subset A$ of sample points. From the height of the sample points we somehow have to approximate the height at the other points in the domain. A naive approach assigns to every $p \in A$ the height of the nearest sample point. However, this gives a discrete terrain, which



doesn't look very natural. Therefore our approach for approximating a terrain is as follows. We first determine a *triangulation* of P: a planar subdivision whose bounded faces are triangles and whose vertices are the points of P. (We assume that the sample points are such that we can make the triangles cover the domain of the terrain.) We then lift each sample point to its correct height, thereby mapping every triangle in the triangulation to a triangle in 3-space. Figure 9.2 illustrates this. What we get is a *polyhedral terrain*, the graph of a continuous function that is piecewise linear. We can use the polyhedral terrain as an approximation of the original terrain.

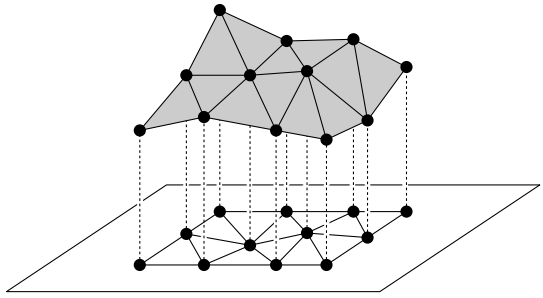


Figure 9.2
Obtaining a polyhedral terrain from a set of sample points

The question remains: how do we triangulate the set of sample points? In general, this can be done in many different ways. But which triangulation is the most appropriate one for our purpose, namely to approximate a terrain? There is no definitive answer to this question. We *do not know* the original terrain, we only know its height at the sample points. Since we have no other information, and the height at the sample points is the correct height for any triangulation, all triangulations of P seem equally good. Nevertheless, some triangulations look more natural than others. For example, have a look at Figure 9.3, which shows two triangulations of the same point set. From the heights of the sample points we get the impression that the sample points were taken from a mountain ridge. Triangulation (a) reflects this intuition. Triangulation (b), however, where one single edge has been "flipped," has introduced a narrow valley cutting through the mountain ridge. Intuitively, this looks wrong. Can we turn this intuition into a criterion that tells us that triangulation (a) is better than triangulation (b)?

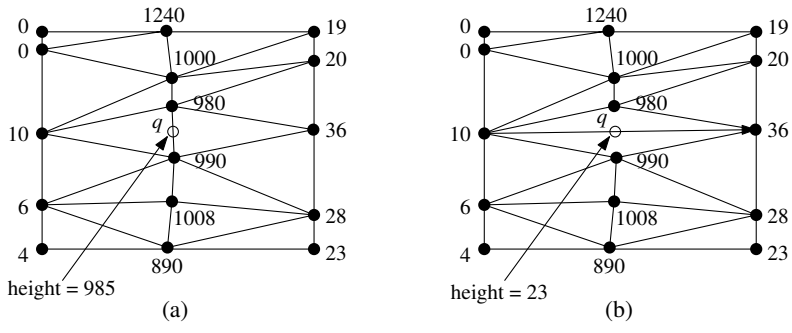


Figure 9.3
Flipping one edge can make a big difference

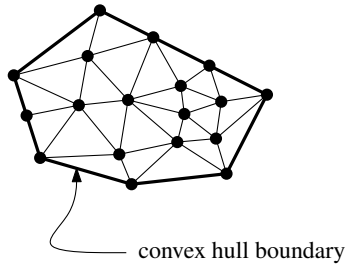
The problem with triangulation (b) is that the height of the point q is deter-

mined by two points that are relatively far away. This happens because q lies in the middle of an edge of two long and sharp triangles. The skinniness of these triangles causes the trouble. So it seems that a triangulation that contains small angles is bad. Therefore we will rank triangulations by comparing their smallest angle. If the minimum angles of two triangulations are identical, then we can look at the second smallest angle, and so on. Since there is only a finite number of different triangulations of a given point set P, this implies that there must be an optimal triangulation, one that maximizes the minimum angle. This will be the triangulation we are looking for.

9.1 Triangulations of Planar Point Sets

Let $P := \{p_1, p_2, \ldots, p_n\}$ be a set of points in the plane. To be able to formally define a triangulation of P, we first define a *maximal planar subdivision* as a subdivision $\mathcal{S}$ such that no edge connecting two vertices can be added to $\mathcal{S}$ without destroying its planarity. In other words, any edge that is not in $\mathcal{S}$ intersects one of the existing edges. A *triangulation* of P is now defined as a maximal planar subdivision whose vertex set is P.

With this definition it is obvious that a triangulation exists. But does it consist of triangles? Yes, every face except the unbounded one must be a triangle: a bounded face is a polygon, and we have seen in Chapter 3 that any polygon can be triangulated. What about the unbounded face? It is not difficult to see that any segment connecting two consecutive points on the boundary of the convex hull of P is an edge in any triangulation $\mathcal{T}$. This implies that the union of the bounded faces of $\mathcal{T}$ is always the convex hull of P, and that the unbounded face is always the complement of the convex hull. (In our application this means that if the domain is a rectangular area, say, we have to make sure that the corners of the domain are included in the set of sample points, so that the triangles in the triangulation cover the domain of the terrain.) The number of triangles is the same in any triangulation of P. This also holds for the number of edges. The exact numbers depend on the number of points in P that are on the boundary of the convex hull of P. (Here we also count points in the interior of convex hull edges. Hence, the number of points on the convex hull boundary is not necessarily the same as the number of convex hull vertices.) This is made precise in the following theorem.



Theorem 9.1 *Let P be a set of n points in the plane, not all collinear, and let k denote the number of points in P that lie on the boundary of the convex hull of P. Then any triangulation of P has $2n-2-k$ triangles and $3n-3-k$ edges.*

Proof. Let $\mathcal{T}$ be a triangulation of P, and let m denote the number of triangles of $\mathcal{T}$. Note that the number of faces of the triangulation, which we denote by n_f, is $m+1$. Every triangle has three edges, and the unbounded face has k edges. Furthermore, every edge is incident to exactly two faces. Hence, the total number of edges of $\mathcal{T}$ is $n_e := (3m+k)/2$. Euler's formula tells us that

$$n - n_e + n_f = 2.$$

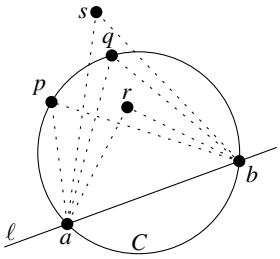
Plugging the values for n_e and n_f into the formula, we get $m = 2n-2-k$, which in turn implies $n_e = 3n-3-k$. □

Let $\mathcal{T}$ be a triangulation of P, and suppose it has m triangles. Consider the $3m$ angles of the triangles of $\mathcal{T}$, sorted by increasing value. Let $\alpha_1, \alpha_2, \ldots, \alpha_{3m}$ be the resulting sequence of angles; hence, $\alpha_i \leqslant \alpha_j$, for $i < j$. We call $A(\mathcal{T}) := (\alpha_1, \alpha_2, \ldots, \alpha_{3m})$ the *angle-vector* of $\mathcal{T}$. Let $\mathcal{T}'$ be another triangulation of the same point set P, and let $A(\mathcal{T}') := (\alpha'_1, \alpha'_2, \ldots, \alpha'_{3m})$ be its angle-vector. We say that the angle-vector of $\mathcal{T}$ is larger than the angle-vector of $\mathcal{T}'$ if $A(\mathcal{T})$ is lexicographically larger than $A(\mathcal{T}')$, or, in other words, if there exists an index i with $1 \leqslant i \leqslant 3m$ such that

$$\alpha_j = \alpha'_j \text{ for all } j < i, \qquad \text{and} \qquad \alpha_i > \alpha'_i.$$

We denote this as $A(\mathcal{T}) > A(\mathcal{T}')$. A triangulation $\mathcal{T}$ is called *angle-optimal* if $A(\mathcal{T}) \geqslant A(\mathcal{T}')$ for all triangulations $\mathcal{T}'$ of P. Angle-optimal triangulations are interesting because, as we have seen in the introduction to this chapter, they are good triangulations if we want to construct a polyhedral terrain from a set of sample points.

Below we will study when a triangulation is angle-optimal. To do this it is useful to know the following theorem, often called Thales's Theorem. Denote the smaller angle defined by three points p, q, r by $\measuredangle pqr$.



Theorem 9.2 *Let C be a circle, ℓ a line intersecting C in points a and b, and p, q, r, and s points lying on the same side of ℓ. Suppose that p and q lie on C, that r lies inside C, and that s lies outside C. Then*

$$\measuredangle arb \;>\; \measuredangle apb \;=\; \measuredangle aqb \;>\; \measuredangle asb.$$

Now consider an edge $e = \overline{p_i p_j}$ of a triangulation $\mathcal{T}$ of P. If e is not an edge of the unbounded face, it is incident to two triangles $p_i p_j p_k$ and $p_i p_j p_l$. If these two triangles form a convex quadrilateral, we can obtain a new triangulation $\mathcal{T}'$ by removing $\overline{p_i p_j}$ from $\mathcal{T}$ and inserting $\overline{p_k p_l}$ instead. We call this operation an *edge flip*. The only difference in the angle-vector of $\mathcal{T}$ and $\mathcal{T}'$ are the six

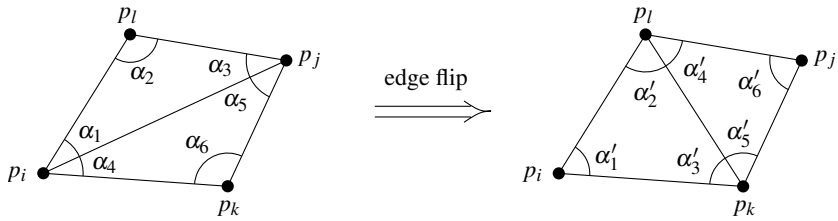


Figure 9.4
Flipping an edge

angles $\alpha_1, \ldots, \alpha_6$ in $A(\mathcal{T})$, which are replaced by $\alpha'_1, \ldots, \alpha'_6$ in $A(\mathcal{T}')$. Figure 9.4 illustrates this. We call the edge $e = \overline{p_i p_j}$ an *illegal edge* if

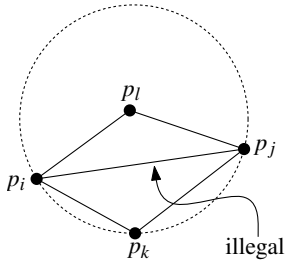
$$\min_{1 \leqslant i \leqslant 6} \alpha_i \;<\; \min_{1 \leqslant i \leqslant 6} \alpha'_i.$$

In other words, an edge is illegal if we can locally increase the smallest angle by flipping that edge. The following observation immediately follows from the definition of an illegal edge.

Observation 9.3 *Let $\mathcal{T}$ be a triangulation with an illegal edge e. Let $\mathcal{T}'$ be the triangulation obtained from $\mathcal{T}$ by flipping e. Then $A(\mathcal{T}') > A(\mathcal{T})$.*

It turns out that it is not necessary to compute the angles $\alpha_1, \ldots, \alpha_6, \alpha'_1, \ldots, \alpha'_6$ to check whether a given edge is legal. Instead, we can use the simple criterion stated in the next lemma. The correctness of this criterion follows from Thales's Theorem.

Lemma 9.4 *Let edge $\overline{p_i p_j}$ be incident to triangles $p_i p_j p_k$ and $p_i p_j p_l$, and let C be the circle through p_i, p_j, and p_k. The edge $\overline{p_i p_j}$ is illegal if and only if the point p_l lies in the interior of C. Furthermore, if the points p_i, p_j, p_k, p_l form a convex quadrilateral and do not lie on a common circle, then exactly one of $\overline{p_i p_j}$ and $\overline{p_k p_l}$ is an illegal edge.*



Observe that the criterion is symmetric in p_k and p_l: p_l lies inside the circle through p_i, p_j, p_k if and only if p_k lies inside the circle through p_i, p_j, p_l. When all four points lie on a circle, both $\overline{p_i p_j}$ and $\overline{p_k p_l}$ are legal. Note that the two triangles incident to an illegal edge must form a convex quadrilateral, so that it is always possible to flip an illegal edge.

We define a *legal triangulation* to be a triangulation that does not contain any illegal edge. From the observation above it follows that any angle-optimal triangulation is legal. Computing a legal triangulation is quite simple, once we are given an initial triangulation. We simply flip illegal edges until all edges are legal.

Algorithm LEGALTRIANGULATION($\mathcal{T}$)
Input. Some triangulation $\mathcal{T}$ of a point set P.
Output. A legal triangulation of P.
1. **while** $\mathcal{T}$ contains an illegal edge $\overline{p_i p_j}$
2. **do** ($*$ Flip $\overline{p_i p_j}$ $*$)
3. Let $p_i p_j p_k$ and $p_i p_j p_l$ be the two triangles adjacent to $\overline{p_i p_j}$.
4. Remove $\overline{p_i p_j}$ from $\mathcal{T}$, and add $\overline{p_k p_l}$ instead.
5. **return** $\mathcal{T}$

Why does this algorithm terminate? It follows from Observation 9.3 that the angle-vector of $\mathcal{T}$ increases in every iteration of the loop. Since there is only a finite number of different triangulations of P, this proves termination of the algorithm. Once it terminates, the result is a legal triangulation. Although the algorithm is guaranteed to terminate, it is too slow to be interesting. We have given the algorithm anyway, because later we shall need a similar procedure. But first we will look at something completely different—or so it seems.

9.2 The Delaunay Triangulation

Let P be a set of n points—or *sites*, as we shall sometimes call them—in the plane. Recall from Chapter 7 that the Voronoi diagram of P is the subdivision of the plane into n regions, one for each site in P, such that the region of a site $p \in P$ contains all points in the plane for which p is the closest site. The Voronoi diagram of P is denoted by $\mathrm{Vor}(P)$. The region of a site p is called

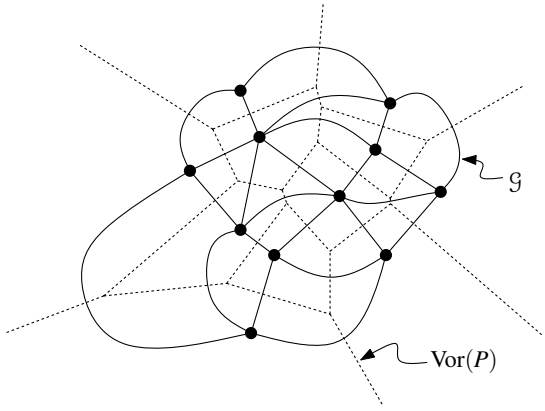


Figure 9.5
The dual graph of $\mathrm{Vor}(P)$

the Voronoi cell of p; it is denoted by $\mathcal{V}(p)$. In this section we will study the dual graph of the Voronoi diagram. This graph $\mathcal{G}$ has a node for every Voronoi cell—equivalently, for every site—and it has an arc between two nodes if the corresponding cells share an edge. Note that this means that $\mathcal{G}$ has an arc for every edge of $\mathrm{Vor}(P)$. As you can see in Figure 9.5, there is a one-to-one correspondence between the bounded faces of $\mathcal{G}$ and the vertices of $\mathrm{Vor}(P)$.

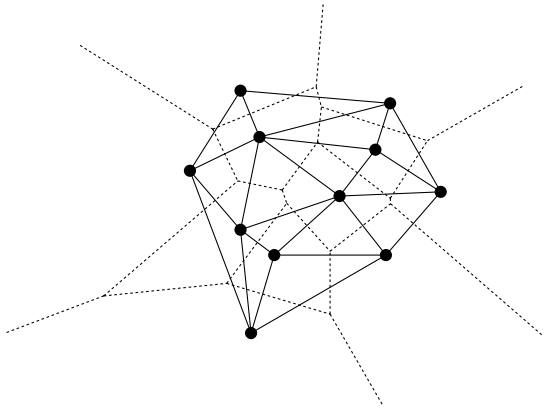


Figure 9.6
The Delaunay graph $\mathcal{DG}(P)$

Consider the straight-line embedding of $\mathcal{G}$, where the node corresponding to the Voronoi cell $\mathcal{V}(p)$ is the point p, and the arc connecting the nodes of $\mathcal{V}(p)$ and $\mathcal{V}(q)$ is the segment $\overline{pq}$—see Figure 9.6. We call this embedding the *Delaunay graph* of P, and we denote it by $\mathcal{DG}(P)$. (Although the name sounds French, Delaunay graphs have nothing to do with the French painter. They

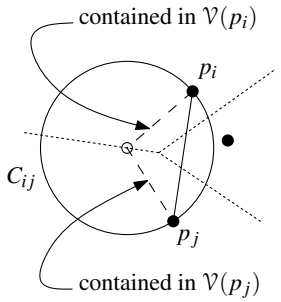
are named after the Russian mathematician Boris Nikolaevich Delone, who wrote his own name as "Борис Николаевич Делоне," which would be transliterated into English as "Delone." However, since his work was published in French—at his time, the languages of science were French and German—his name is better known in the French transliteration.) The Delaunay graph of a point set turns out to have a number of surprising properties. The first is that it is always a plane graph: no two edges in the embedding cross.

Theorem 9.5 *The Delaunay graph of a planar point set is a plane graph.*

Proof. To prove this, we need a property of the edges in the Voronoi diagram stated in Theorem 7.4(ii). For completeness we repeat the property, phrased here in terms of Delaunay graphs.

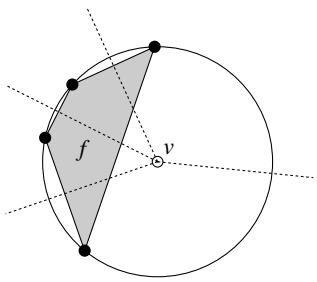
> The edge $\overline{p_i p_j}$ is in the Delaunay graph $\mathcal{DG}(P)$ if and only if there is a closed disc C_{ij} with p_i and p_j on its boundary and no other site of P contained in it. (The center of such a disc lies on the common edge of $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$.)

Define t_{ij} to be the triangle whose vertices are p_i, p_j, and the center of C_{ij}. Note that the edge of t_{ij} connecting p_i to the center of C_{ij} is contained in $\mathcal{V}(p_i)$; a similar observation holds for p_j. Now let $\overline{p_k p_l}$ be another edge of $\mathcal{DG}(P)$, and define the circle C_{kl} and the triangle t_{kl} similar to the way C_{ij} and t_{ij} were defined.



Suppose for a contradiction that $\overline{p_i p_j}$ and $\overline{p_k p_l}$ intersect. Both p_k and p_l must lie outside C_{ij} and so they also lie outside t_{ij}. This implies that $\overline{p_k p_l}$ must intersect one of the edges of t_{ij} incident to the center of C_{ij}. Similarly, $\overline{p_i p_j}$ must intersect one of the edges of t_{kl} incident to the center of C_{kl}. It follows that one of the edges of t_{ij} incident to the center of C_{ij} must intersect one of the edges of t_{kl} incident to the center of C_{kl}. But this contradicts that these edges are contained in disjoint Voronoi cells. □

The Delaunay graph of P is an embedding of the dual graph of the Voronoi diagram. As observed earlier, it has a face for every vertex of $\mathrm{Vor}(P)$. The edges around a face correspond to the Voronoi edges incident to the corresponding Voronoi vertex. In particular, if a vertex v of $\mathrm{Vor}(P)$ is a vertex of the Voronoi cells for the sites $p_1, p_2, p_3, \ldots, p_k$, then the corresponding face f in $\mathcal{DG}(P)$ has $p_1, p_2, p_3, \ldots, p_k$ as its vertices. Theorem 7.4(i) tells us that in this situation the points $p_1, p_2, p_3, \ldots, p_k$ lie on a circle around v, so we not only know that f is a k-gon, but even that it is convex.



If the points of P are distributed at random, the chance that four points happen to lie on a circle is very small. We will—in this chapter—say that a set of points is in *general position* if it contains no four points on a circle. If P is in general position, then all vertices of the Voronoi diagram have degree three, and consequently all bounded faces of $\mathcal{DG}(P)$ are triangles. This explains why $\mathcal{DG}(P)$ is often called the *Delaunay triangulation* of P. We shall be a bit more careful, and will call $\mathcal{DG}(P)$ the *Delaunay graph* of P. We define a *Delaunay triangulation* to be any triangulation obtained by adding edges to the Delaunay graph. Since all faces of $\mathcal{DG}(P)$ are convex, obtaining such a triangulation

is easy. Observe that the Delaunay triangulation of P is unique if and only if $\mathcal{DG}(P)$ is a triangulation, which is the case if P is in general position.

We now rephrase Theorem 7.4 about Voronoi diagrams in terms of Delaunay graphs.

Theorem 9.6 *Let P be a set of points in the plane.*
(i) Three points $p_i, p_j, p_k \in P$ are vertices of the same face of the Delaunay graph of P if and only if the circle through p_i, p_j, p_k contains no point of P in its interior.
(ii) Two points $p_i, p_j \in P$ form an edge of the Delaunay graph of P if and only if there is a closed disc C that contains p_i and p_j on its boundary and does not contain any other point of P.

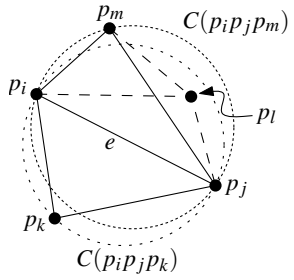
Theorem 9.6 readily implies the following characterization of Delaunay triangulations.

Theorem 9.7 *Let P be a set of points in the plane, and let $\mathcal{T}$ be a triangulation of P. Then $\mathcal{T}$ is a Delaunay triangulation of P if and only if the circumcircle of any triangle of $\mathcal{T}$ does not contain a point of P in its interior.*

Since we argued before that a triangulation is good for the purpose of height interpolation if its angle-vector is as large as possible, our next step should be to look at the angle-vector of Delaunay triangulations. We do this by a slight detour through legal triangulations.

Theorem 9.8 *Let P be a set of points in the plane. A triangulation $\mathcal{T}$ of P is legal if and only if $\mathcal{T}$ is a Delaunay triangulation of P.*

Proof. It follows immediately from the definitions that any Delaunay triangulation is legal.



We shall prove that any legal triangulation is a Delaunay triangulation by contradiction. So assume $\mathcal{T}$ is a legal triangulation of P that is not a Delaunay triangulation. By Theorem 9.6, this means that there is a triangle $p_ip_jp_k$ such that the circumcircle $C(p_ip_jp_k)$ contains a point $p_l \in P$ in its interior. Let $e := \overline{p_ip_j}$ be the edge of $p_ip_jp_l$ such that the triangle $p_ip_jp_l$ does not intersect $p_ip_jp_k$. Of all such pairs $(p_ip_jp_k, p_l)$ in $\mathcal{T}$, choose the one that maximizes the angle $\measuredangle p_ip_lp_j$. Now look at the triangle $p_ip_jp_m$ adjacent to $p_ip_jp_k$ along e. Since $\mathcal{T}$ is legal, e is legal. By Lemma 9.4 this implies that p_m does not lie in the interior of $C(p_ip_jp_k)$. The circumcircle $C(p_ip_jp_m)$ of $p_ip_jp_m$ contains the part of $C(p_ip_jp_k)$ that is separated from $p_ip_jp_k$ by e. Consequently, $p_l \in C(p_ip_jp_m)$. Assume that $\overline{p_jp_m}$ is the edge of $p_ip_jp_m$ such that $p_jp_mp_l$ does not intersect $p_ip_jp_m$. But now $\measuredangle p_jp_lp_m > \measuredangle p_ip_lp_j$ by Thales's Theorem, contradicting the definition of the pair $(p_ip_jp_k, p_l)$. □

Since any angle-optimal triangulation must be legal, Theorem 9.8 implies that any angle-optimal triangulation of P is a Delaunay triangulation of P. When P is in general position, there is only one legal triangulation, which is then the only angle-optimal triangulation, namely the unique Delaunay triangulation

that coincides with the Delaunay graph. When P is not in general position, then any triangulation of the Delaunay graph is legal. Not all these Delaunay triangulations need to be angle-optimal. However, their angle-vectors do not differ too much. Moreover, using Thales's Theorem one can show that the minimum angle in any triangulation of a set of co-circular points is the same, that is, the minimum angle is independent of the triangulation. This implies that any triangulation turning the Delaunay graph into a Delaunay triangulation has the same minimum angle. The following theorem summarizes this.

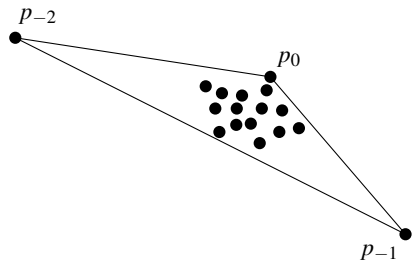
Theorem 9.9 *Let P be a set of points in the plane. Any angle-optimal triangulation of P is a Delaunay triangulation of P. Furthermore, any Delaunay triangulation of P maximizes the minimum angle over all triangulations of P.*

9.3 Computing the Delaunay Triangulation

We have seen that for our purpose—approximating a terrain by constructing a polyhedral terrain from a set P of sample points—a Delaunay triangulation of P is a suitable triangulation. This is because the Delaunay triangulation maximizes the minimum angle. So how do we compute such a Delaunay triangulation?

We already know from Chapter 7 how to compute the Voronoi diagram of P. From $\mathrm{Vor}(P)$ we can easily obtain the Delaunay graph $\mathcal{DG}(P)$, and by triangulating the faces with more than three vertices we can obtain a Delaunay triangulation. In this section we describe a different approach: we will compute a Delaunay triangulation directly, using the randomized incremental approach we have so successfully applied to the linear programming problem in Chapter 4 and to the point location problem in Chapter 6.

In Chapter 6 we found it convenient to start with a large rectangle containing the scene, to avoid problems caused by unbounded trapezoids. In the same spirit we now start with a large triangle that contains the set P. We will add two extra points p_{-1} and p_{-2} that, together with the highest point p_0 of P, form a triangle containing all the points. This means we are now computing a Delaunay triangulation of $P \cup \{p_{-1}, p_{-2}\}$ instead of the Delaunay triangulation of P. Later we want to obtain the Delaunay triangulation of P by discarding p_{-1} and p_{-2}, together with all incident edges. For this to work we have to choose p_{-1} and p_{-2} far enough away, so that they don't destroy any triangles in the Delaunay triangulation of P. In particular, we must ensure they do not lie in any circle defined by three points in P. We postpone the details of this to a later stage; first we have a look at the algorithm.



The algorithm is randomized incremental, so it adds the points in random order and it maintains a Delaunay triangulation of the current point set. Consider the addition of a point p_r. We first find the triangle of the current triangulation that contains p_r—how this is done will be explained later—and we add edges from p_r to the vertices of this triangle. If p_r happens to fall on an edge e of the triangulation, we have to add edges from p_r to the opposite vertices in the triangles sharing e. Figure 9.7 illustrates these two cases. We now have

a triangulation again, but not necessarily a Delaunay triangulation. This is because the addition of p_r can make some of the existing edges illegal. To remedy this, we call a procedure LEGALIZEEDGE with each potentially illegal edge. This procedure replaces illegal edges by legal ones through edge flips. Before we come to the details of this, we give a precise description of the main algorithm. It will be convenient for the analysis to let P be a set of $n+1$ points.

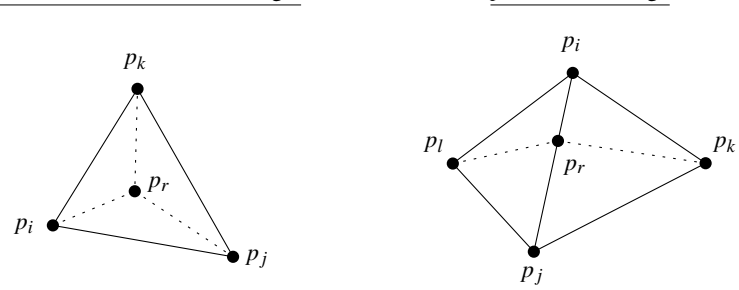


Figure 9.7
The two cases when adding a point p_r

Algorithm DELAUNAYTRIANGULATION(P)
Input. A set P of $n+1$ points in the plane.
Output. A Delaunay triangulation of P.

1. Let p_0 be the lexicographically highest point of P, that is, the rightmost among the points with largest y-coordinate.
2. Let p_{-1} and p_{-2} be two points in $\mathbb{R}^2$ sufficiently far away and such that P is contained in the triangle $p_0p_{-1}p_{-2}$.
3. Initialize $\mathcal{T}$ as the triangulation consisting of the single triangle $p_0p_{-1}p_{-2}$.
4. Compute a random permutation $p_1, p_2, \ldots, p_n$ of $P \setminus \{p_0\}$.
5. **for** $r \leftarrow 1$ **to** n
6. **do** ($*$ Insert p_r into $\mathcal{T}$: $*$)
7. Find a triangle $p_ip_jp_k \in \mathcal{T}$ containing p_r.
8. **if** p_r lies in the interior of the triangle $p_ip_jp_k$
9. **then** Add edges from p_r to the three vertices of $p_ip_jp_k$, thereby splitting $p_ip_jp_k$ into three triangles.
10. LEGALIZEEDGE($p_r, \overline{p_ip_j}, \mathcal{T}$)
11. LEGALIZEEDGE($p_r, \overline{p_jp_k}, \mathcal{T}$)
12. LEGALIZEEDGE($p_r, \overline{p_kp_i}, \mathcal{T}$)
13. **else** ($*$ p_r lies on an edge of $p_ip_jp_k$, say the edge $\overline{p_ip_j}$ $*$)
14. Add edges from p_r to p_k and to the third vertex p_l of the other triangle that is incident to $\overline{p_ip_j}$, thereby splitting the two triangles incident to $\overline{p_ip_j}$ into four triangles.
15. LEGALIZEEDGE($p_r, \overline{p_ip_l}, \mathcal{T}$)
16. LEGALIZEEDGE($p_r, \overline{p_lp_j}, \mathcal{T}$)
17. LEGALIZEEDGE($p_r, \overline{p_jp_k}, \mathcal{T}$)
18. LEGALIZEEDGE($p_r, \overline{p_kp_i}, \mathcal{T}$)
19. Discard p_{-1} and p_{-2} with all their incident edges from $\mathcal{T}$.
20. **return** $\mathcal{T}$

Next we discuss the details of turning the triangulation we get after line 9 (or line 14) into a Delaunay triangulation. We know from Theorem 9.8 that a triangulation is a Delaunay triangulation if all its edges are legal. In the spirit of algorithm LEGALTRIANGULATION, we therefore flip illegal edges until the triangulation is legal again. The question that remains is which edges may become illegal due to the insertion of p_r. Observe that an edge $\overline{p_ip_j}$ that was legal before can only become illegal if one of the triangles incident to it has changed. So only the edges of the new triangles need to be checked. This is done using the subroutine LEGALIZEEDGE, which tests and possibly flips an edge. If LEGALIZEEDGE flips an edge, other edges may become illegal. Therefore LEGALIZEEDGE calls itself recursively with such potentially illegal edges.

LEGALIZEEDGE($p_r, \overline{p_ip_j}, \mathcal{T}$)
1. ($*$ The point being inserted is p_r, and $\overline{p_ip_j}$ is the edge of $\mathcal{T}$ that may need to be flipped. $*$)
2. **if** $\overline{p_ip_j}$ is illegal
3. **then** Let $p_ip_jp_k$ be the triangle adjacent to $p_rp_ip_j$ along $\overline{p_ip_j}$.
4. ($*$ Flip $\overline{p_ip_j}$: $*$) Replace $\overline{p_ip_j}$ with $\overline{p_rp_k}$.
5. LEGALIZEEDGE($p_r, \overline{p_ip_k}, \mathcal{T}$)
6. LEGALIZEEDGE($p_r, \overline{p_kp_j}, \mathcal{T}$)

The test in line 2 whether an edge is illegal can normally be done by applying Lemma 9.4. There are some complications because of the presence of the special points p_{-1} and p_{-2}. We shall come back to this later; first we prove that the algorithm is correct.

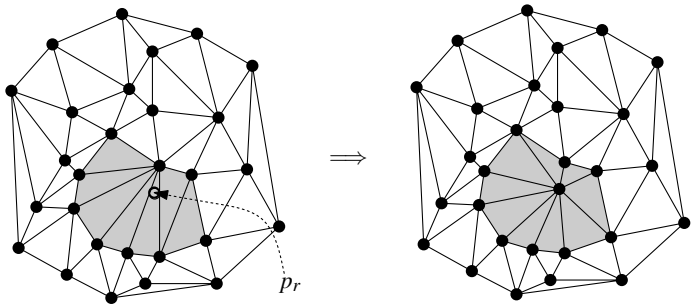
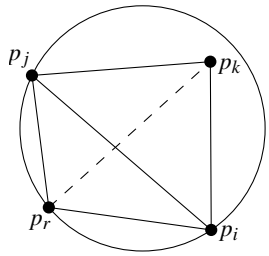


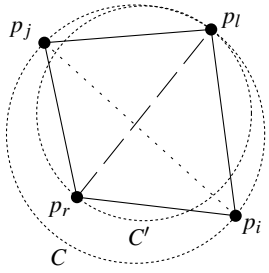
Figure 9.8
All edges created are incident to p_r

To ensure the correctness of the algorithm, we need to prove that no illegal edges remain after all calls to LEGALIZEEDGE have been processed. From the code of LEGALIZEEDGE it is clear that every new edge created due to the insertion of p_r is incident to p_r. Figure 9.8 illustrates this; the triangles that are destroyed and the new triangles are shown in grey. The crucial observation (proved below) is that every new edge must be legal, so there is no need to test them. Together with the earlier observation that an edge can only become illegal if one of its incident triangles changes, this proves that the algorithm tests any edge that may become illegal. Hence, the algorithm is correct. Note that, as in Algorithm LEGALTRIANGULATION, the algorithm cannot get into an infinite loop, because every flip makes the angle-vector of the triangulation larger.



Lemma 9.10 *Every new edge created in* DELAUNAYTRIANGULATION *or in* LEGALIZEEDGE *during the insertion of p_r is an edge of the Delaunay graph of* $\{p_{-2}, p_{-1}, p_0, \ldots, p_r\}$.

Proof. Consider first the edges $\overline{p_r p_i}$, $\overline{p_r p_j}$, $\overline{p_r p_k}$ (and perhaps $\overline{p_r p_l}$) created by splitting $p_i p_j p_k$ (and maybe $p_i p_j p_l$). Since $p_i p_j p_k$ is a triangle in the Delaunay triangulation before the addition of p_r, the circumcircle C of $p_i p_j p_k$ contains no point p_t with $t < r$ in its interior. By shrinking C we can find a circle C' through p_i and p_r contained in C. Because $C' \subset C$ we know that C' is empty. This implies that $\overline{p_r p_i}$ is an edge of the Delaunay graph after the addition of p_r. The same holds for $\overline{p_r p_j}$ and $\overline{p_r p_k}$ (and for $\overline{p_r p_l}$, if it exists).



Now consider an edge flipped by LEGALIZEEDGE. Such an edge flip always replaces an edge $\overline{p_i p_j}$ of a triangle $p_i p_j p_l$ by an edge $\overline{p_r p_l}$ incident to p_r. Since $p_i p_j p_l$ was a Delaunay triangle before the addition of p_r and because its circumcircle C contains p_r—otherwise $\overline{p_i p_j}$ would not be illegal—we can shrink the circumcircle to obtain an empty circle C' with only p_r and p_l on its boundary. Hence, $\overline{p_r p_l}$ is an edge of the Delaunay graph after the addition. □

We have proved the correctness of the algorithm. What remains is to describe how to implement two important steps: how to find the triangle containing the point p_r in line 7 of DELAUNAYTRIANGULATION, and how to deal correctly with the points p_{-1} and p_{-2} in the test in line 2 in LEGALIZEEDGE. We start with the former issue.

To find the triangle containing p_r we use an approach quite similar to what we did in Chapter 6: while we build the Delaunay triangulation, we also build a point location structure $\mathcal{D}$, which is a directed acyclic graph. The leaves of $\mathcal{D}$ correspond to the triangles of the current triangulation $\mathcal{T}$, and we maintain cross-pointers between those leaves and the triangulation. The internal nodes of $\mathcal{D}$ correspond to triangles that were in the triangulation at some earlier stage, but have already been destroyed. The point location structure is built as follows. In line 3 we initialize $\mathcal{D}$ as a DAG with a single leaf node, which corresponds to the triangle $p_0 p_{-1} p_{-2}$.

Now suppose that at some point we split a triangle $p_i p_j p_k$ of the current triangulation into three (or two) new triangles. The corresponding change in $\mathcal{D}$ is to add three (or two) new leaves to $\mathcal{D}$, and to make the leaf for $p_i p_j p_k$ into an internal node with outgoing pointers to those three (or two) leaves. Similarly, when we replace two triangles $p_k p_i p_j$ and $p_i p_j p_l$ by triangles $p_k p_i p_l$ and $p_k p_l p_j$ by an edge flip, we create leaves for the two new triangles, and the nodes of $p_k p_i p_j$ and $p_i p_j p_l$ get pointers to the two new leaves. Figure 9.9 shows an example of the changes in $\mathcal{D}$ caused by the addition of a point. Observe that when we make a leaf into an internal node, it gets at most three outgoing pointers.

Using $\mathcal{D}$ we can locate the next point p_r to be added in the current triangulation. This is done as follows. We start at the root of $\mathcal{D}$, which corresponds to the initial triangle $p_0 p_{-1} p_{-2}$. We check the three children of the root to see in which triangle p_r lies, and we descend to the corresponding child. We then

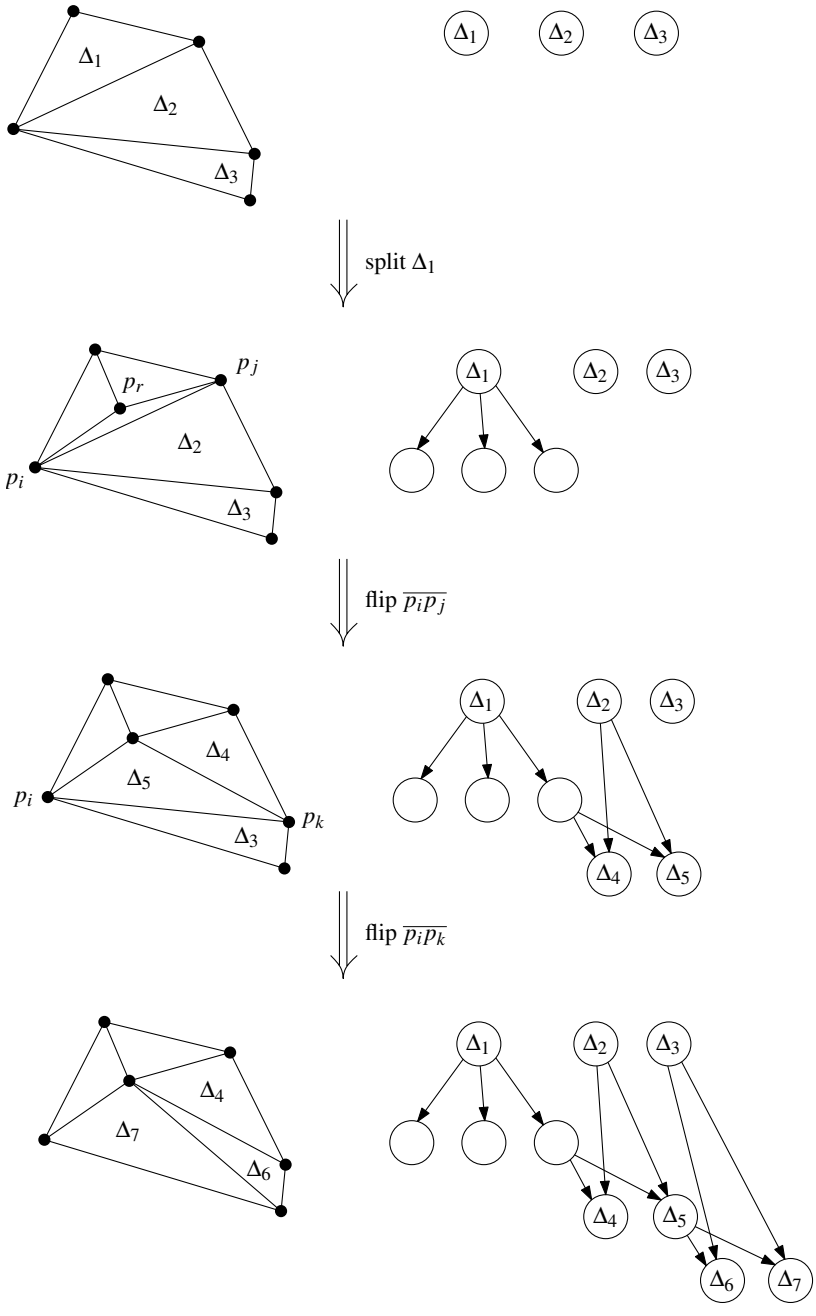


Figure 9.9
The effect of inserting point p_r into triangle Δ_1 on the data structure $\mathcal{D}$ (the part of $\mathcal{D}$ that does not change is omitted in the figure)

check the children of this node, descend to a child whose triangle contains p_r, and so on, until we reach a leaf of $\mathcal{D}$. This leaf corresponds to a triangle in the current triangulation that contains p_r. Since the out-degree of any node is at most three, this takes linear time in the number of nodes on the search path, or, in other words, in the number of triangles stored in $\mathcal{D}$ that contain p_r.

There is only one detail left, namely how to choose p_{-1} and p_{-2}, and how to implement the test of whether an edge is legal. On the one hand, we have to choose p_{-1} and p_{-2} to be far away, because we don't want their presence to influence the Delaunay triangulation of P. One the other hand, we don't want to introduce the huge coordinates needed for that. So what we do is to treat these points *symbolically*: we do not actually assign coordinates to them, but instead modify the tests for point location and for illegal edges such that they work *as if* we had chosen the points to be very far away.

In the following, we will say that $p = (x_p, y_p)$ is higher than $q = (x_q, y_q)$ if $y_p > y_q$ or $y_p = y_q$ and $x_q > x_p$, and use the (lexicographic) ordering on P induced by this relation.

Let ℓ_{-1} be a horizontal line lying below the entire set P, and let ℓ_{-2} be a horizontal line lying above P. Conceptually, we choose p_{-1} to lie on the line ℓ_{-1} sufficiently far to the right that p_{-1} lies outside every circle defined by three non-collinear points of P, and such that the clockwise ordering of the points of P around p_{-1} is identical to their (lexicographic) ordering. Next, we choose p_{-2} to lie on the line ℓ_{-2} sufficiently far to the left that p_{-2} lies outside every circle defined by three non-collinear points of $P \cup \{p_{-1}\}$, and such that the counterclockwise ordering of the points of $P \cup \{p_{-1}\}$ around p_{-2} is identical to their (lexicographic) ordering.

The Delaunay triangulation of $P \cup \{p_{-1}, p_{-2}\}$ consists of the Delaunay triangulation of P, edges connecting p_{-1} to every point on the right convex hull of P, edges connecting p_{-2} to every point on the left convex hull of P, and the one edge $\overline{p_{-1}p_{-2}}$. The lowest point of P and the highest point p_0 of P are connected to both p_{-1} and p_{-2}.

During the point location step, we need to determine the position of a point p_j with respect to the oriented line from p_i to p_k. By our choice of p_{-1} and p_{-2}, the following conditions are equivalent:

- p_j lies to the left of the line from p_i to p_{-1};
- p_j lies to the left of the line from p_{-2} to p_i;
- p_j is lexicographically larger than p_i.

It remains to explain how to treat p_{-1} and p_{-2} when we check whether an edge is illegal. Let $\overline{p_ip_j}$ be the edge to be tested, and let p_k and p_l be the other vertices of the triangles incident to $\overline{p_ip_j}$ (if they exist).

- $\overline{p_ip_j}$ *is an edge of the triangle* $p_0p_{-1}p_{-2}$. These edges are always legal.
- *The indices* i, j, k, l *are all non-negative.* This is the normal case; none of the points involved in the test is treated symbolically. Hence, $\overline{p_ip_j}$ is illegal if and only if p_l lies inside the circle defined by p_i, p_j, and p_k.
- *All other cases.* In this case, $\overline{p_ip_j}$ is legal if and only if $\min(k,l) < \min(i,j)$.

Only the last case requires further justification. Since the situation where $\overline{p_ip_j}$ is $\overline{p_{-1}p_{-2}}$ is handled in the first case, at most one of the indices i and j

is negative. On the other hand, either p_k or p_l is the point p_r that we have just inserted, and so at most one of the indices k and l is negative.

If only one of the four indices is negative, then this point lies outside the circle defined by the other three points, and the method is correct.

Otherwise, both $\min(i,j)$ and $\min(k,l)$ are negative, and the fact that p_{-2} lies outside any circle defined by three points in $P \cup \{p_{-1}\}$ implies that the method is correct.

9.4 The Analysis

We first look at the *structural change* generated by the algorithm. This is the number of triangles created and deleted during the course of the algorithm. Before we start the analysis, we introduce some notation: $P_r := \{p_1, \ldots, p_r\}$ and $\mathcal{DG}_r := \mathcal{DG}(\{p_{-2}, p_{-1}, p_0\} \cup P_r)$.

Lemma 9.11 *The expected number of triangles created by algorithm* DELAUNAYTRIANGULATION *is at most* $9n+1$.

Proof. In the beginning, we create the single triangle $p_0p_{-1}p_{-2}$. In iteration r of the algorithm, when we insert p_r, we first split one or two triangles, creating three or four new triangles. This splitting creates the same number of edges in $\mathcal{DG}_r$, namely $\overline{p_rp_i}$, $\overline{p_rp_j}$, $\overline{p_rp_k}$ (and maybe $\overline{p_rp_l}$). Furthermore, for every edge that we flip in procedure LEGALIZEEDGE, we create two new triangles. Again, the flipping creates an edge of $\mathcal{DG}_r$ incident to p_r. To summarize: if after the insertion of p_r there are k edges of $\mathcal{DG}_r$ incident to p_r, then we have created at most $2(k-3)+3 = 2k-3$ new triangles. The number k is the degree of p_r in $\mathcal{DG}_r$; we denote this degree by $\deg(p_r, \mathcal{DG}_r)$. degree of p_r, over all possible permutations of the set P? As in Chapter 4 and 6 we use *backwards analysis* to bound this value. So, for the moment, we fix the set P_r. We want to bound the expected degree of the point p_r, which is a *random* element of the set P_r. By Theorem 7.3, the Delaunay graph $\mathcal{DG}_r$ has at most $3(r+3)-6$ edges. Three of these are the edges of $p_0p_{-1}p_{-2}$, and therefore the total degree of the vertices in P_r is less than $2[3(r+3)-9] = 6r$. This means that the expected degree of a random point of P_r is at most 6. Summarizing the above, we can bound the number of triangles created in step r as follows.

$$\begin{aligned} \mathrm{E}\big[\text{number of triangles created in step } r\big] &\leqslant \mathrm{E}\big[2\deg(p_r, \mathcal{DG}_r) - 3\big] \\ &= 2\,\mathrm{E}\big[\deg(p_r, \mathcal{DG}_r)\big] - 3 \\ &\leqslant 2 \cdot 6 - 3 \;=\; 9 \end{aligned}$$

The total number of created triangles is one for the triangle $p_0p_{-1}p_{-2}$ that we start with, plus the number of triangles created in each of the insertion steps. Using linearity of expectation, we get that the expected total number of created triangles is bounded by $1+9n$. □

We now state the main result.

Theorem 9.12 *The Delaunay triangulation of a set P of n points in the plane can be computed in $O(n \log n)$ expected time, using $O(n)$ expected storage.*

Proof. The correctness of the algorithm follows from the discussion above. As for the storage requirement, we note that only the search structure $\mathcal{D}$ could use more than linear storage. However, every node of $\mathcal{D}$ corresponds to a triangle created by the algorithm, and by the previous lemma the expected number of these is $O(n)$.

To bound the expected running time we first ignore the time spent in the point location step (line 7). Now the time spent by the algorithm is proportional to the number of created triangles. From the previous lemma we can therefore conclude that the expected running time, not counting the time for point location, is $O(n)$.

It remains to account for the point location steps. The time to locate the point p_r in the current triangulation is linear in the number of nodes of $\mathcal{D}$ that we visit. Any visited node corresponds to a triangle that was created at some earlier stage and that contains p_r. If we count the triangle of the current triangulation separately, then the time for locating p_r is $O(1)$ plus linear time in the number of triangles that were present at some earlier stage, but have been destroyed, and contain p_r.

A triangle $p_i p_j p_k$ can be destroyed from the triangulation for one of two reasons:

- A new point p_l has been inserted inside (or on the boundary of) $p_i p_j p_k$, and $p_i p_j p_k$ was split into three (or two) subtriangles.
- An edge flip has replaced $p_i p_j p_k$ and an adjacent triangle $p_i p_j p_l$ by the pair $p_k p_i p_l$ and $p_k p_j p_l$.

In the first case, the triangle $p_i p_j p_k$ was a Delaunay triangle before p_l was inserted. In the second case, either $p_i p_j p_k$ was a Delaunay triangle and p_l was inserted, or $p_i p_j p_l$ was a Delaunay triangle and p_k was inserted. If $p_i p_j p_l$ was the Delaunay triangle, then the fact that the edge $\overline{p_i p_j}$ was flipped means that both p_k and p_r lie inside the circumcircle of $p_i p_j p_l$.

In all cases we can charge the fact that triangle $p_i p_j p_k$ was visited to a Delaunay triangle Δ that has been destroyed in the same stage as $p_i p_j p_k$, and such that the circumcircle of Δ contains p_r. Denote the subset of points in P that lie in the circumcircle of a given triangle Δ by $K(\Delta)$. In the argument above the visit to a triangle during the location of p_r is charged to a triangle Δ with $p_r \in K(\Delta)$. It is easy to see that a triangle Δ can be charged at most once for every one of the points in $K(\Delta)$. Therefore the total time for the point location steps is

$$O(n + \sum_{\Delta} \mathrm{card}(K(\Delta))), \tag{9.1}$$

where the summation is over all Delaunay triangles Δ created by the algorithm. We shall prove later that the expected value of this sum is $O(n \log n)$. This proves the theorem. □

It remains to bound the expected size of the sets $K(\Delta)$. If Δ is a triangle of the Delaunay triangulation $\mathcal{DG}_r$, then what would we expect $\mathrm{card}(K(\Delta))$ to be?

For $r = 1$ we would expect it to be roughly n, and for $r = n$ we know that it is zero. What happens in between? The nice thing about randomization is that it "interpolates" between those two extremes. The right intuition would be that, since P_r is a random sample, the number of points lying inside the circumcircle of a triangle $\Delta \in \mathcal{DG}_r$ is about $O(n/r)$. But be warned: this is not really true for *all* triangles in $\mathcal{DG}_r$. Nevertheless, the sum in expression (9.1) behaves as if it were true.

In the remainder of this section we will give a quick proof of this fact for the case of a point set in general position. The result is true for the general case as well, but to see that we have to work a little bit harder, so we postpone that to the next section, where we treat the problem in more generality.

Lemma 9.13 *If P is a point set in general position, then*

$$\sum_{\Delta} \operatorname{card}(K(\Delta)) = O(n \log n),$$

where the summation is over all Delaunay triangles Δ created by the algorithm.

Proof. Since P is in general position, every subset P_r is in general position. This implies that the triangulation after adding the point p_r is the unique triangulation $\mathcal{DG}_r$. We denote the set of triangles of $\mathcal{DG}_r$ by $\mathcal{T}_r$. Now the set of Delaunay triangles created in stage r equals $\mathcal{T}_r \setminus \mathcal{T}_{r-1}$ by definition. Hence, we can rewrite the sum we want to bound as

$$\sum_{r=1}^{n} \Bigg(\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \operatorname{card}(K(\Delta)) \Bigg).$$

For a point q, let $k(P_r, q)$ denote the number of triangles $\Delta \in \mathcal{T}_r$ such that $q \in K(\Delta)$, and let $k(P_r, q, p_r)$ be the number of triangles $\Delta \in \mathcal{T}_r$ such that not only $q \in K(\Delta)$ but for which we also have that p_r is incident to Δ. Recall that any Delaunay triangle created in stage r is incident to p_r, so we have

$$\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \operatorname{card}(K(\Delta)) = \sum_{q \in P \setminus P_r} k(P_r, q, p_r). \tag{9.2}$$

For the moment, we fix P_r. In other words, we consider all expectations to be over the set of permutations of the set P where P_r is equal to a fixed set P_r^*. The value of $k(P_r, q, p_r)$ then depends only on the choice of p_r. Since a triangle $\Delta \in \mathcal{T}_r$ is incident to a random point $p \in P_r^*$ with probability at most $3/r$, we get

$$\mathrm{E}\big[k(P_r, q, p_r)\big] \leqslant \frac{3k(P_r, q)}{r}.$$

If we sum this over all $q \in P \setminus P_r$ and use (9.2), we get

$$\mathrm{E}\Big[\sum_{\Delta \in \mathcal{T}_r \setminus \mathcal{T}_{r-1}} \operatorname{card}(K(\Delta))\Big] \leqslant \frac{3}{r} \sum_{q \in P \setminus P_r} k(P_r, q). \tag{9.3}$$

Every $q \in P \setminus P_r$ is equally likely to appear as p_{r+1}, and so we have

$$\mathrm{E}\big[k(P_r, p_{r+1})\big] = \frac{1}{n-r} \sum_{q \in P \setminus P_r} k(P_r, q).$$

We can substitute this into (9.3), and get

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}}\mathrm{card}(K(\Delta))\Big] \leqslant 3\Big(\frac{n-r}{r}\Big)\,\mathrm{E}\big[k(P_r,p_{r+1})\big].$$

What is $k(P_r,p_{r+1})$? It is the number of triangles Δ of $\mathcal{T}_r$ that have $p_{r+1}\in K(\Delta)$. By the criterion from Theorem 9.6 (i), these triangles are exactly the triangles of $\mathcal{T}_r$ that will be destroyed by the insertion of p_{r+1}. Hence, we can rewrite the previous expression as

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}}\mathrm{card}(K(\Delta))\Big] \leqslant 3\Big(\frac{n-r}{r}\Big)\,\mathrm{E}\big[\mathrm{card}(\mathcal{T}_r\setminus\mathcal{T}_{r+1})\big].$$

Theorem 9.1 shows that the number of triangles in $\mathcal{T}_m$ is precisely $2(m+3)-2-3=2m+1$. Therefore, the number of triangles *destroyed* by the insertion of point p_{r+1} is exactly two less than the number of triangles *created* by the insertion of p_{r+1}, and we can rewrite the sum as

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}}\mathrm{card}(K(\Delta))\Big] \leqslant 3\Big(\frac{n-r}{r}\Big)\Big(\mathrm{E}\big[\mathrm{card}(\mathcal{T}_{r+1}\setminus\mathcal{T}_r)\big]-2\Big).$$

Until now we considered P_r to be fixed. At this point, we can simply take the average over all choices of $P_r\subset P$ on both sides of the inequality above, and find that it also holds if we consider the expectation to be over all possible permutations of the set P.

We already know that the number of triangles created by the insertion of p_{r+1} is identical to the number of edges incident to p_{r+1} in $\mathcal{T}_{r+1}$, and that the expected number of these edges is at most 6. We conclude that

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}}\mathrm{card}(K(\Delta))\Big] \leqslant 12\Big(\frac{n-r}{r}\Big).$$

Summing over r proves the lemma. □

9.5* A Framework for Randomized Algorithms

Up to now we have seen three randomized incremental algorithms in this book: one for linear programming in Chapter 4, one for computing a trapezoidal map in Chapter 6, and one for computing a Delaunay triangulation in this chapter. (We will see one more in Chapter 11.) These algorithms, and most other randomized incremental algorithms in the computational geometry literature, all work according to the following principle.

Suppose the problem is to compute some geometric structure $\mathcal{T}(X)$, defined by a set X of geometric objects. (For instance, a Delaunay triangulation defined by a set of points in the plane.) A randomized incremental algorithm does this by adding the objects in X in random order, meanwhile maintaining the structure $\mathcal{T}$. To add the next object, the algorithm first finds out where

the current structure has to be changed because there is a conflict with the object—the *location step*—and then it updates the structure locally—the *update step*. Because all randomized incremental algorithms are so much alike, their analyses are quite similar as well. To avoid having to prove the same bounds over and over again for different problems, an axiomatic framework has been developed that captures the essence of randomized incremental algorithms. This framework—called a *configuration space*—can be used to prove ready-to-use bounds for the expected running time of many randomized incremental algorithms. (Unfortunately, the term "configuration space" is also used in motion planning, where it means something completely different—see Chapter 13.) In this section we describe this framework, and we give a theorem that can be used to analyze any randomized incremental algorithm that fits into the framework. For instance, the theorem can immediately be applied to prove Lemma 9.13, this time without assuming that P has to be in general position.

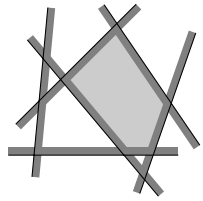
A *configuration space* is defined to be a four-tuple (X, Π, D, K). Here X is the input to the problem, which is a finite set of (geometric) *objects*; we denote the cardinality of X by n. The set Π is a set whose elements are called *configurations*. Finally, D and K both assign to every configuration $\Delta \in \Pi$ a subset of X, denoted $D(\Delta)$ and $K(\Delta)$ repectively. Elements of the set $D(\Delta)$ are said to *define* the configuration Δ, and the elements of the set $K(\Delta)$ are said to be *in conflict with*, or to *kill*, Δ. The number of elements of $K(\Delta)$ is called the *conflict size* of the configuration Δ. We require that (X, Π, D, K) satisfies the following conditions.

- The number $d := \max\{\text{card}(D(\Delta)) \mid \Delta \in \Pi\}$ is a constant. We call this number the *maximum degree* of the configuration space. Moreover, the number of configurations sharing the same defining set should be bounded by a constant.
- We have $D(\Delta) \cap K(\Delta) = \emptyset$ for all configurations $\Delta \in \Pi$.

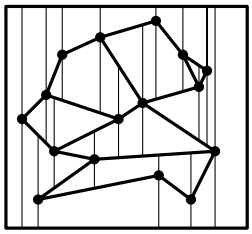
A configuration Δ is called *active* over a subset $S \subseteq X$ if $D(\Delta)$ is contained in S and $K(\Delta)$ is disjoint from S. We denote the set of configurations active over S by $\mathcal{T}(S)$, so we have

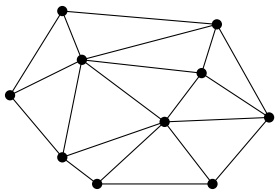
$$\mathcal{T}(S) := \{\Delta \in \Pi \ : \ D(\Delta) \subseteq S \text{ and } K(\Delta) \cap S = \emptyset\}.$$

The active configurations form the structure we want to compute. More precisely, the goal is to compute $\mathcal{T}(X)$. Before we continue our discussion of this abstract framework, let's see how the geometric structures we have met so far fit in.



Half-plane intersection. In this case the input set X is a set of half-planes in the plane. We want to define Π, D, and K in such a way that $\mathcal{T}(X)$ is what we want to compute, namely the intersection of the half-planes in X. We can achieve this as follows. The set Π of configurations consists of all the intersection points of the lines bounding the half-planes in X. The defining set $D(\Delta)$ of a configuration $\Delta \in \Pi$ consists of the two lines defining the intersection, and the killing set $K(\Delta)$ consists of all half-planes that do not contain the intersection point. Hence, for any subset $S \subset X$, and in particular for X itself, $\mathcal{T}(S)$ is the set of vertices of the common intersection of the half-planes in S.





Trapezoidal maps. Here the input set X is a set of segments in the plane. The set Π of configurations contains all trapezoids appearing in the trapezoidal map of any $S \subseteq X$. The defining set $D(\Delta)$ of a configuration Δ is the set of segments that are necessary to define Δ. The killing set $K(\Delta)$ of a trapezoid Δ is the set of segments that intersect Δ. With these definitions, $\mathcal{T}(S)$ is exactly the set of trapezoids of the trapezoidal map of S.

Delaunay Triangulation. The input set X is a set of points in general position in the plane. The set Π of configurations consists of triangles formed by three (non-collinear) points in X. The defining set $D(\Delta)$ consists of the points that form the vertices of Δ, and the killing set $K(\Delta)$ is the set of points lying inside the circumcircle of Δ. By Theorem 9.6, $\mathcal{T}(S)$ is exactly the set of triangles of the unique Delaunay triangulation of S.

As stated earlier, the goal is to compute the structure $\mathcal{T}(X)$. Randomized incremental algorithms do this by computing a random permutation $x_1, x_2, \ldots, x_n$ of the objects in X and then adding the objects in this order, meanwhile maintaining $\mathcal{T}(X_r)$, where $X_r := \{x_1, x_2, \ldots, x_r\}$. The fundamental property of configuration spaces that makes this possible is that we can decide whether or not a configuration Δ appears in $\mathcal{T}(X_r)$ by looking at it *locally*—we only need to look for the defining and killing objects of Δ. In particular, $\mathcal{T}(X_r)$ does not depend on the order in which the objects in X_r were added. For instance, a triangle Δ is in the Delaunay triangulation of S if and only if the vertices of Δ are in S, and no point of S lies in the circumcircle of Δ.

The first thing we usually did when we analyzed a randomized incremental algorithm was to prove a bound on the expected structural change—see for instance Lemma 9.11. The next theorem does the same, but now in the abstract configuration-space framework.

Theorem 9.14 *Let (X, Π, D, K) be a configuration space, and let $\mathcal{T}$ and X_r be defined as above. Then the expected number of configurations in $\mathcal{T}(X_r) \setminus \mathcal{T}(X_{r-1})$ is at most*

$$\frac{d}{r} E[\mathrm{card}(\mathcal{T}(X_r))],$$

where d is the maximum degree of the configuration space.

Proof. As in previous occasions where we wanted to bound the structural change, we use backwards analysis: instead of trying to argue about the number of configurations that appear due to the addition of x_r into X_{r-1}, we shall argue about the number of configurations that disappear when we remove x_r from X_r. To this end we temporarily let X_r be some fixed subset $X_r^* \subset X$ of cardinality r. We now want to bound the expected number of configurations $\Delta \in \mathcal{T}(X_r)$ that disappear when we remove a random object x_r from X_r. By definition of $\mathcal{T}$, such a configuration Δ must have $x_r \in D(\Delta)$. Since there are at most $d \cdot \mathrm{card}(\mathcal{T}(X_r))$ pairs (x, Δ) with $\Delta \in \mathcal{T}(X_r)$ and $x \in D(\Delta)$, we have

$$\sum_{x \in X_r} \mathrm{card}(\{\Delta \in \mathcal{T}(X_r) \mid x \in D(\Delta)\}) \leqslant d \cdot \mathrm{card}(\mathcal{T}(X_r)).$$

Hence, the expected number of configurations disappearing due to the removal of a random object from X_r is at most $\frac{d}{r}\operatorname{card}(\mathcal{T}(X_r))$. In this argument, the set X_r was a fixed subset $X_r^* \subset X$ of cardinality r. To obtain the general bound, we have to average over all possible subsets of size r, which gives a bound of $\frac{d}{r}\operatorname{E}\big[\operatorname{card}(\mathcal{T}(X_r))\big]$. □

This theorem gives a generic bound for the expected size of the structural change during a randomized incremental algorithm. But what about the cost of the location steps? In many cases we will need a bound of the same form as in this chapter, namely we need to bound

$$\sum_{\Delta} \operatorname{card}(K(\Delta)),$$

where the summation is over all configurations Δ that are created by the algorithm, that is, all configurations that appear in one of the $\mathcal{T}(X_r)$. This bound is given in the following theorem.

Theorem 9.15 *Let (X,Π,D,K) be a configuration space, and let $\mathcal{T}$ and X_r be defined as above. Then the expected value of*

$$\sum_{\Delta} \operatorname{card}(K(\Delta)),$$

where the summation is over all configurations Δ appearing in at least one $\mathcal{T}(X_r)$ with $1 \leqslant r \leqslant n$, is at most

$$\sum_{r=1}^{n} d^2 \Big(\frac{n-r}{r}\Big)\Big(\frac{\operatorname{E}\big[\operatorname{card}(\mathcal{T}(X_r))\big]}{r}\Big),$$

where d is the maximum degree of the configuration space.

Proof. We can follow the proof of Lemma 9.13 quite closely. We first rewrite the sum as

$$\sum_{r=1}^{n}\Bigg(\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}} \operatorname{card}(K(\Delta))\Bigg).$$

Next, let $k(X_r,y)$ denote the number of configurations $\Delta \in \mathcal{T}(X_r)$ such that $y \in K(\Delta)$, and let $k(X_r,y,x_r)$ be the number of configurations $\Delta \in \mathcal{T}(X_r)$ such that not only $y \in K(\Delta)$ but for which we also have $x_r \in D(\Delta)$. Any new configuration appearing due to the addition of x_r must have $x_r \in D(\Delta)$. This implies that

$$\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}} \operatorname{card}(K(\Delta)) = \sum_{y\in X\setminus X_r} k(X_r,y,x_r). \tag{9.4}$$

We now fix the set X_r. The expected value of $k(X_r,y,x_r)$ then depends only on the choice of $x_r \in X_r$. Since the probability that $y \in D(\Delta)$ for a configuration $\Delta \in \mathcal{T}(X_r)$ is at most d/r, we have

$$\operatorname{E}\big[k(X_r,y,x_r)\big] \leqslant \frac{dk(X_r,y)}{r}.$$

If we sum this over all $y \in X \setminus X_r$ and use (9.4), we get

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}} \mathrm{card}(K(\Delta))\Big] \leqslant \frac{d}{r}\sum_{y\in X\setminus X_r} k(X_r,y). \tag{9.5}$$

On the other hand, every $y \in X \setminus X_r$ is equally likely to appear as x_{r+1}, so

$$\mathrm{E}\big[k(X_r,x_{r+1})\big] = \frac{1}{n-r}\sum_{y\in X\setminus X_r} k(X_r,y).$$

Substituting this into (9.5) gives

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}} \mathrm{card}(K(\Delta))\Big] \leqslant d\Big(\frac{n-r}{r}\Big)\,\mathrm{E}\big[k(X_r,x_{r+1})\big].$$

Now observe that $k(X_r,x_{r+1})$ is the number of configurations Δ of $\mathcal{T}(X_r)$ that will be destroyed in the next stage, when x_{r+1} is inserted. This means we can rewrite the last expression as

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}} \mathrm{card}(K(\Delta))\Big] \leqslant d\Big(\frac{n-r}{r}\Big)\,\mathrm{E}\big[\mathrm{card}(\mathcal{T}(X_r)\setminus\mathcal{T}(X_{r+1}))\big]. \tag{9.6}$$

Unlike in the proof of Lemma 9.13, however, we cannot simply bound the number of configurations destroyed in stage $r+1$ by the number of configurations created at that stage, because that need not be true in a general configuration space. Hence, we proceed somewhat differently.

First we observe that we can take the average over all choices of X_r on both sides of (9.6) and find that it also holds if the expectation is over all permutations of X. Next, we sum over all r, and rewrite the sum as follows:

$$\sum_{r=1}^{n} d\Big(\frac{n-r}{r}\Big)\,\mathrm{card}(\mathcal{T}(X_r)\setminus\mathcal{T}(X_{r+1})) = \sum_{\Delta} d\Big(\frac{n-[j(\Delta)-1]}{j(\Delta)-1}\Big), \tag{9.7}$$

where the summation on the right hand side is over all configurations Δ that are created and later destroyed by the algorithm, and where $j(\Delta)$ denotes the stage when configuration Δ is destroyed. Let $i(\Delta)$ denote the stage when the configuration Δ is created. Since $i(\Delta) \leqslant j(\Delta)-1$, we have

$$\frac{n-[j(\Delta)-1]}{j(\Delta)-1} = \frac{n}{j(\Delta)-1} - 1 \leqslant \frac{n}{i(\Delta)} - 1 = \frac{n-i(\Delta)}{i(\Delta)}.$$

If we substitute this into (9.7), we see that

$$\sum_{r=1}^{n} d\Big(\frac{n-r}{r}\Big)\,\mathrm{card}(\mathcal{T}(X_r)\setminus\mathcal{T}(X_{r+1})) \leqslant \sum_{\Delta} d\Big(\frac{n-i(\Delta)}{i(\Delta)}\Big).$$

The right hand side of this expression is at most

$$\sum_{r=1}^{n} d\Big(\frac{n-r}{r}\Big)\,\mathrm{card}(\mathcal{T}(X_r)\setminus\mathcal{T}(X_{r-1}))$$

(the difference being only those configurations that are created but never destroyed) and so we have

$$\mathrm{E}\Big[\sum_{r=1}^{n}\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}}\mathrm{card}(K(\Delta))\Big] \leqslant \sum_{r=1}^{n} d\Big(\frac{n-r}{r}\Big)\,\mathrm{E}\big[\mathrm{card}(\mathcal{T}(X_r)\setminus\mathcal{T}(X_{r-1}))\big].$$

By Theorem 9.14, we get the bound we wanted to prove:

$$\mathrm{E}\Big[\sum_{r=1}^{n}\sum_{\Delta\in\mathcal{T}_r\setminus\mathcal{T}_{r-1}}\mathrm{card}(K(\Delta))\Big] \leqslant \sum_{r=1}^{n} d\Big(\frac{n-r}{r}\Big)\frac{d}{r}\,\mathrm{E}\big[\mathrm{card}(\mathcal{T}(X_r))\big]. \qquad \square$$

This finishes the analysis in the abstract setting. As an example, we will show how to apply the results to our randomized incremental algorithm for computing the Delaunay triangulation. In particular, we will prove that

$$\sum_{\Delta}\mathrm{card}(K(\Delta)) = O(n\log n),$$

where the summation is over all triangles Δ created by the algorithm, and where $K(\Delta)$ is the set of points in the circumcircle of the triangle.

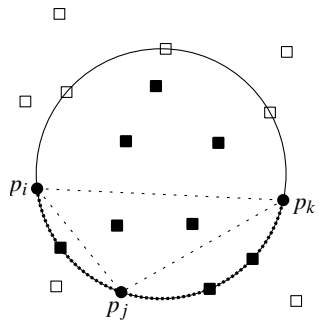
Unfortunately, it seems impossible to properly define a configuration space whose configurations are triangles when the points are not in general position. Therefore we shall choose the configurations slightly differently.

Let P be a set of points in the plane, not necessarily in general position. Let $\Omega := \{p_0, p_{-1}, p_{-2}\}$ denote the set of three points we used to start the construction. Recall that p_0 is the lexicographically largest point from P, while points p_{-1} and p_{-2} were chosen such that they do not destroy any Delaunay edges between points in P. We set $X := P\setminus\{p_0\}$. Every triple $\Delta = (p_i, p_j, p_k)$ of points in $X\cup\Omega$ that do not lie on a line defines a configuration with $D(\Delta) := \{p_i, p_j, p_k\}\cap X$ and $K(\Delta)$ is the set of points of X that lie either in the interior of the circumcircle of the triangle $p_ip_jp_k$ or on the circular arc on the circumcircle from p_i to p_k containing p_j. We call such a configuration Δ a *Delaunay corner* of X, because Δ is active over $S\subseteq X$ if and only if p_i, p_j, and p_k are consecutive points on the boundary of one face of the Delaunay graph $\mathcal{DG}(\Omega\cup S)$. Note that any set of three non-collinear points defines three different configurations.

The important observation is that whenever DELAUNAYTRIANGULATION creates a new triangle, this triangle is of the form $p_ip_rp_j$, where p_r is the point inserted in this stage, and $\overline{p_rp_i}$ and $\overline{p_rp_j}$ are edges of the Delaunay graph $\mathcal{DG}(\Omega\cup P_r)$—see Lemma 9.10. It follows that when the triangle $p_ip_rp_j$ is created, the triple (p_j, p_r, p_i) is a Delaunay corner of $\mathcal{DG}(\Omega\cup P_r)$ and, hence, it is an active configuration over the set P_r. The set $K(\Delta)$ defined for this configuration contains all points contained in the circumcircle of the triangle $p_ip_rp_j$. We can therefore bound the original sum by

$$\sum_{\Delta}\mathrm{card}(K(\Delta)),$$

where the sum is over all *Delaunay corners* Δ that appear in some intermediate Delaunay graph $\mathcal{DG}(\Omega\cup P_r)$.



■ points in $K(\Delta)$

□ points not in $K(\Delta)$

Now Theorem 9.15 applies. How many Delaunay corners are there in the Delaunay graph of $S \cup \Omega$? The worst case is when the Delaunay graph is a triangulation. If S contains r points, then the triangulation has $2(r+3)-5$ triangles, and therefore $6(r+3)-15 = 6r+3$ Delaunay corners. It follows from Theorem 9.15 that

$$\sum_{\Delta} \text{card}(K(\Delta)) \leqslant \sum_{r=1}^{n} 9\Big(\frac{n-r}{r}\Big)\Big(\frac{6r-3}{r}\Big) \leqslant 54n \sum_{r=1}^{n} \frac{1}{r} \leqslant 54n(\ln n + 1).$$

This finally completes the proof of Theorem 9.12.

9.6 Notes and Comments

The problem of triangulating a set of points is a topic in computational geometry that is well known outside this field. Triangulations of point sets in two and more dimensions are of paramount importance in numerical analysis, for instance for finite element methods, but also in computer graphics. In this chapter we looked at the case of triangulations that only use the given points as vertices. If additional points—so-called Steiner points—are allowed, the problem is also known as *meshing* and is treated in more detail in Chapter 14.

Lawson [244] proved that any two triangulations of a planar point set can be transformed into each other by flipping edges. He later suggested finding a good triangulation by iteratively flipping edges, where such an edge-flip improves some cost function of the triangulation [245].

It had been observed for some time that triangulations that lead to good interpolations avoid long and skinny triangles [38]. The result that there is—if we ignore degenerate cases—only one locally optimal triangulation with respect to the angle-vector, namely the Delaunay triangulation, is due to Sibson [360].

Looking only at the angle-vector completely ignores the height of the data points, and is therefore also called the *data-independent* approach. A good motivation for this approach is given by Rippa [328], who proves that the Delaunay triangulation is the triangulation that minimizes the *roughness* of the resulting terrain, no matter what the actual height data is. Here, roughness is defined as the integral of the square of the L_2-norm of the gradient of the terrain. More recent research tries to find improved triangulations by taking the height information into account. This *data-dependent* approach was first proposed by Dyn et al. [154], who suggest different cost criteria for triangulations, which depend on the height of the data points. Interestingly, they compute their improved triangulations by starting with the Delaunay triangulation and iteratively flipping edges. The same approach is taken by Quak and Schumaker [325], who consider piecewise cubic interpolation, and Brown [76]. Quak and Schumaker observe that their triangulations are small improvements compared to the Delaunay triangulation when they try to approximate smooth surfaces, but that they can be drastically different for non-smooth surfaces.

More references relevant to Delaunay triangulations as the dual of Voronoi diagrams can be found in Chapter 7.

The randomized incremental algorithm we have given here is due to Guibas et al. [196], but our analysis of $\sum_\Delta \text{card}(K(\Delta))$ is from Mulmuley's book [290]. The argument that extends the analysis to the case of points in degenerate position is new. Alternative randomized algorithms were given by Boissonnat et al. [69, 71], and by Clarkson and Shor [133].

Various *geometric graphs* defined on a set of points P have been found to be subgraphs of the Delaunay triangulation of P. The most important one is probably the *Euclidean minimum spanning tree* (EMST) of the set of points [349]; others are the *Gabriel graph* [186] and the *relative neighborhood graph* [374]. We treat these geometric graphs in the exercises.

Another important triangulation is the *minimum weight triangulation*, that is, a triangulation whose weight is minimal (where the weight of a triangulation is the sum of the lengths of all edges of the triangulation) [12, 42, 146, 147]. Determining a minimum weight triangulation among all triangulations of a given point set was recently shown to be NP-complete [291].

9.7 Exercises

9.1 In this exercise we look at the number of different triangulations that a set of n points in the plane may allow.

a. Prove that no set of n points can be triangulated in more than $2^{\binom{n}{2}}$ ways.

b. Prove that there are sets of n points that can be triangulated in at least $2^{n-2\sqrt{n}}$ different ways.

9.2 The degree of a point in a triangulation is the number of edges incident to it. Give an example of a set of n points in the plane such that, no matter how the set is triangulated, there is always a point whose degree is $n-1$.

9.3 Prove that any two triangulations of a planar point set can be transformed into each other by edge flips. *Hint:* Show first that any two triangulations of a convex polygon can be transformed into each other by edge flips.

9.4 Prove that the smallest angle of any triangulation of a convex polygon whose vertices lie on a circle is the same. This implies that *any* completion of the Delaunay triangulation of a set of points maximizes the minimum angle.

9.5 a. Given four points p, q, r, s in the plane, prove that point s lies in the interior of the circle through p, q, and r if and only if the following condition holds. Assume that p, q, r form the vertices of a triangle in clockwise order.

$$\det \begin{pmatrix} p_x & p_y & p_x^2+p_y^2 & 1 \\ q_x & q_y & q_x^2+q_y^2 & 1 \\ r_x & r_y & r_x^2+r_y^2 & 1 \\ s_x & s_y & s_x^2+s_y^2 & 1 \end{pmatrix} > 0.$$

b. The determinant test of part a. can be used to test if an edge in a triangulation is legal. Can you come up with an alternative way to implement this test? Discuss the advantages and/or disadvantages of your method compared to the determinant test.

9.6 We have described algorithm DELAUNAYTRIANGULATION by calling a recursive procedure LEGALIZEEDGE. Give an iterative version of this procedure, and discuss the advantages and/or disadvantages of your procedure over the recursive one.

9.7 Prove that all edges of $\mathcal{DG}(P_r)$ that are not in $\mathcal{DG}(P_{r-1})$ are incident to p_r. In other words, the new edges of $\mathcal{DG}(P_r)$ form a star as in Figure 9.8. Give a direct proof, without referring to algorithm DELAUNAYTRIANGULATION.

9.8 Let P be a set of n points in general position, and let $q \notin P$ be a point inside the convex hull of P. Let p_i, p_j, p_k be the vertices of a triangle in the Delaunay triangulation of P that contains q. (Since q can lie on an edge of the Delaunay triangulation, there can be two such triangles.) Prove that $\overline{qp_i}$, $\overline{qp_j}$, and $\overline{qp_k}$ are edges of the Delaunay triangulation of $P \cup \{q\}$.

9.9 The algorithm given in this chapter is randomized, and it computes the Delaunay triangulation of a set of n points in $O(n \log n)$ expected time. Show that the worst-case running time of the algorithm is $\Omega(n^2)$.

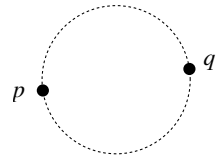
9.10 The algorithm given in this chapter uses two extra points p_{-1} and p_{-2} to start the construction of the Delaunay triangulation. These points should not lie in any circle defined by three input points, and so far away that they see the points of P in their lexicographic order. These conditions were enforced by implementing operations involving these points in a special way—see page 204. Compute explicit coordinates for the extra points such that this special implementation is not needed. Is this a better approach?

9.11 A *Euclidean minimum spanning tree* (EMST) of a set P of points in the plane is a tree of minimum total edge length connecting all the points. EMST's are interesting in applications where we want to connect sites in a planar environment by communication lines (local area networks), roads, railroads, or the like.

a. Prove that the set of edges of a Delaunay triangulation of P contains an EMST for P.
b. Use this result to give an $O(n \log n)$ algorithm to compute an EMST for P.

9.12 The *traveling salesman problem* (TSP) is to compute a shortest tour visiting all points in a given point set. The traveling salesman problem is NP-hard. Show how to find a tour whose length is at most two times the optimal length, using the EMST defined in the previous exercise.

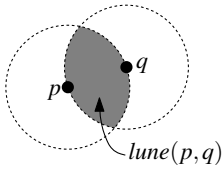
9.13 The *Gabriel graph* of a set P of points in the plane is defined as follows: Two points p and q are connected by an edge of the Gabriel graph if and only if the disc with diameter pq does not contain any other point of P.



a. Prove that $\mathcal{DG}(P)$ contains the Gabriel graph of P.
b. Prove that p and q are adjacent in the Gabriel graph of P if and only if the Delaunay edge between p and q intersects its dual Voronoi edge.
c. Give an $O(n \log n)$ time algorithm to compute the Gabriel graph of a set of n points.

9.14 The *relative neighborhood graph* of a set P of points in the plane is defined as follows: Two points p and q are connected by an edge of the relative neighborhood graph if and only if

$$d(p,q) \leqslant \min_{r \in P, r \neq p,q} \max(d(p,r), d(q,r)).$$



a. Given two points p and q, let $lune(p,q)$ be the moon-shaped region formed as the intersection of the two circles around p and q whose radius is $d(p,q)$. Prove that p and q are connected in the relative neighborhood graph if and only if $lune(p,q)$ does not contain any point of P in its interior.
b. Prove that $\mathcal{DG}(P)$ contains the relative neighborhood graph of P.
c. Design an algorithm to compute the relative neighborhood graph of a given point set.

9.15 Prove the following relationship between the edge sets of an EMST, of the relative neighborhood graph (RNG), the Gabriel graph (GG), and the Delaunay graph ($\mathcal{DG}$) of a point set P.

$$EMST \subseteq RNG \subseteq GG \subseteq \mathcal{DG}.$$

(See the previous exercises for the definition of these graphs.)

9.16 A k-clustering of a set P of n points in the plane is a partitioning of P into k non-empty subsets $P_1, \ldots, P_k$. Define the distance between any pair P_i, P_j of clusters to be the minimum distance between one point from P_i and one point from P_j, that is,

$$\text{dist}(P_i, P_j) := \min_{p \in P_i, q \in P_j} \text{dist}(p,q).$$

We want to find a k-clustering (for given k and P) that maximizes the minimum distance between clusters.

a. Suppose the mimimum distance between clusters is achieved by points $p \in P_i$ and $q \in P_j$. Prove that $\overline{pq}$ is an edge of the Delaunay triangulation of P.
b. Give an $O(n \log n)$ time algorithm to compute a k-clustering maximizing the minimum distance between clusters. *Hint:* Use a Union-Find data structure.

9.17 The *weight* of a triangulation is the sum of the lengths of all edges of the triangulation. A *minimum weight triangulation* is a triangulation whose weight is minimal. Disprove the conjecture that the Delaunay triangulation is a minimum weight triangulation.

9.18* Give an example of a geometric configuration space (X, Π, D, K) where $\mathcal{T}(X_r) \setminus \mathcal{T}(X_{r+1})$ can be arbitrarily large compared to $\mathcal{T}(X_{r+1}) \setminus \mathcal{T}(X_r)$.

9.19* Apply configuration spaces to analyze the randomized incremental algorithm of Chapter 6.