7 Voronoi Diagrams

The Post Office Problem

Suppose you are on the advisory board for the planning of a supermarket chain, and there are plans to open a new branch at a certain location. To predict whether the new branch will be profitable, you must estimate the number of customers it will attract. For this you have to model the behavior of your potential customers: how do people decide where to do their shopping? A similar question arises in social geography, when studying the economic activities in a country: what is the trading area of certain cities? In a more abstract setting we have a set of

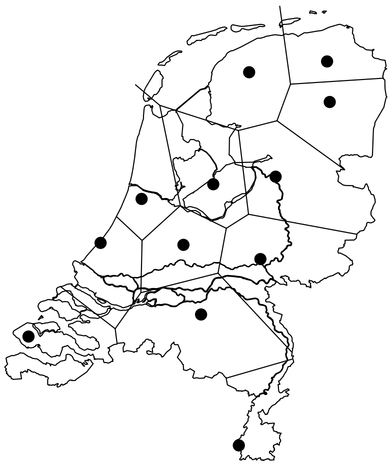


Figure 7.1
The trading areas of the capitals of the twelve provinces in the Netherlands, as predicted by the Voronoi assignment model

central places—called *sites*—that provide certain goods or services, and we want to know for each site where the people live who obtain their goods or services from that site. (In computational geometry the sites are traditionally viewed as post offices where customers want to post their letters—hence the subtitle of this chapter.) To study this question we make the following simplifying assumptions:

- the price of a particular good or service is the same at every site;
- the cost of acquiring the good or service is equal to the price plus the cost of transportation to the site;

- the cost of transportation to a site equals the Euclidean distance to the site times a fixed price per unit distance;
- consumers try to minimize the cost of acquiring the good or service.

Usually these assumptions are not completely satisfied: goods may be cheaper at some sites than at others, and the transportation cost between two points is probably not linear in the Euclidean distance between them. But the model above can give a rough approximation of the trading areas of the sites. Areas where the behavior of the people differs from that predicted by the model can be subjected to further research, to see what caused the different behavior.

Our interest lies in the geometric interpretation of the model above. The assumptions in the model induce a subdivision of the total area under consideration into regions—the trading areas of the sites—such that the people who live in the same region all go to the same site. Our assumptions imply that people simply get their goods at the nearest site—a fairly realistic situation. This means that the trading area for a given site consists of all those points for which that site is closer than any other site. Figure 7.1 gives an example. The sites in this figure are the capitals of the twelve provinces in the Netherlands.

The model where every point is assigned to the nearest site is called the *Voronoi assignment model*. The subdivision induced by this model is called the *Voronoi diagram* of the set of sites. From the Voronoi diagram we can derive all kinds of information about the trading areas of the sites and their relations. For example, if the regions of two sites have a common boundary then these two sites are likely to be in direct competition for customers that live in the boundary region.

The Voronoi diagram is a versatile geometric structure. We have described an application to social geography, but the Voronoi diagram has applications in physics, astronomy, robotics, and many more fields. It is also closely linked to another important geometric structure, the so-called Delaunay triangulation, which we shall encounter in Chapter 9. In the current chapter we shall confine ourselves to the basic properties and the construction of the Voronoi diagram of a set of point sites in the plane.

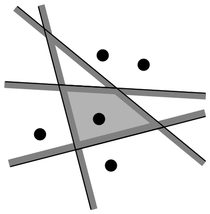
7.1 Definition and Basic Properties

Denote the Euclidean distance between two points p and q by $\mathrm{dist}(p,q)$. In the plane we have

$$\mathrm{dist}(p,q) := \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}.$$

Let $P := \{p_1, p_2, \ldots, p_n\}$ be a set of n distinct points in the plane; these points are the sites. We define the Voronoi diagram of P as the subdivision of the plane into n cells, one for each site in P, with the property that a point q lies in the cell corresponding to a site p_i if and only if $\mathrm{dist}(q,p_i) < \mathrm{dist}(q,p_j)$ for each $p_j \in P$ with $j \neq i$. We denote the Voronoi diagram of P by $\mathrm{Vor}(P)$. Abusing the terminology slightly, we will sometimes use '$\mathrm{Vor}(P)$' or 'Voronoi diagram' to indicate only the edges and vertices of the subdivision. For example, when we say that a Voronoi diagram is connected we mean that the union of its edges and vertices forms a connected set. The cell of $\mathrm{Vor}(P)$ that corresponds to a site p_i is denoted $\mathcal{V}(p_i)$; we call it the Voronoi cell of p_i. (In the terminology of the introduction to this chapter: $\mathcal{V}(p_i)$ is the trading area of site p_i.)

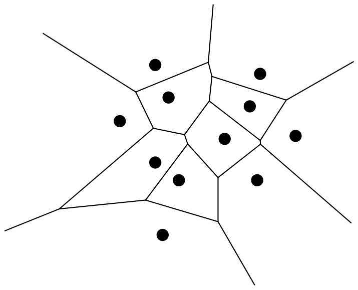
We now take a closer look at the Voronoi diagram. First we study the structure of a single Voronoi cell. For two points p and q in the plane we define the *bisector of p and q* as the perpendicular bisector of the line segment $\overline{pq}$. This bisector splits the plane into two half-planes. We denote the open half-plane that contains p by $h(p,q)$ and the open half-plane that contains q by $h(q,p)$. Notice that $r \in h(p,q)$ if and only if $\mathrm{dist}(r,p) < \mathrm{dist}(r,q)$. From this we obtain the following observation.



Observation 7.1 $\mathcal{V}(p_i) = \bigcap_{1\leqslant j\leqslant n, j\neq i} h(p_i,p_j)$.

Thus $\mathcal{V}(p_i)$ is the intersection of $n-1$ half-planes and, hence, a (possibly unbounded) open convex polygonal region bounded by at most $n-1$ vertices and at most $n-1$ edges.

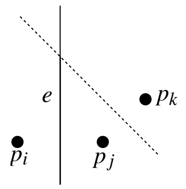
What does the complete Voronoi diagram look like? We just saw that each cell of the diagram is the intersection of a number of half-planes, so the Voronoi diagram is a planar subdivision whose edges are straight. Some edges are line segments and others are half-lines. Unless all sites are collinear there will be no edges that are full lines:



Theorem 7.2 *Let P be a set of n point sites in the plane. If all the sites are collinear then* $\mathrm{Vor}(P)$ *consists of $n-1$ parallel lines. Otherwise,* $\mathrm{Vor}(P)$ *is connected and its edges are either segments or half-lines.*

Proof. The first part of the theorem is easy to prove, so assume that not all sites in P are collinear.

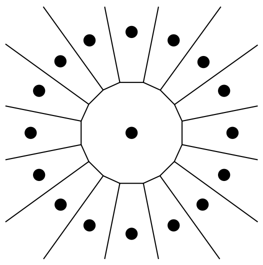
We first show that the edges of $\mathrm{Vor}(P)$ are either segments or half-lines. We already know that the edges of $\mathrm{Vor}(P)$ are parts of straight lines, namely parts of the bisectors between pairs of sites. Now suppose for a contradiction that there is an edge e of $\mathrm{Vor}(P)$ that is a full line. Let e be on the boundary of the Voronoi cells $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$. Let $p_k \in P$ be a point that is not collinear with p_i and p_j. The bisector of p_j and p_k is not parallel to e and, hence, it intersects e. But then the part of e that lies in the interior of $h(p_k,p_j)$ cannot be on the boundary of $\mathcal{V}(p_j)$, because it is closer to p_k than to p_j, a contradiction.



It remains to prove that $\mathrm{Vor}(P)$ is connected. If this were not the case then there would be a Voronoi cell $\mathcal{V}(p_i)$ splitting the plane into two. Because Voronoi cells are convex, $\mathcal{V}(p_i)$ would consist of a strip bounded by two parallel full lines. But we just proved that the edges of the Voronoi diagram cannot be full lines, a contradiction. $\square$

Now that we understand the structure of the Voronoi diagram we investigate its complexity, that is, the total number of its vertices and edges. Since there are n sites and each Voronoi cell has at most $n-1$ vertices and edges, the complexity of $\mathrm{Vor}(P)$ is at most quadratic. It is not clear, however, whether $\mathrm{Vor}(P)$ can actually have quadratic complexity: it is easy to construct an example where a single Voronoi cell has linear complexity, but can it happen that many cells have linear complexity? The following theorem shows that this is not the case and that the average number of vertices of the Voronoi cells is less than six.

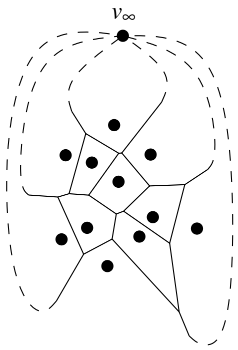
Theorem 7.3 *For $n \geqslant 3$, the number of vertices in the Voronoi diagram of a set of n point sites in the plane is at most $2n-5$ and the number of edges is at most $3n-6$.*



Proof. If the sites are all collinear then the theorem immediately follows from Theorem 7.2, so assume this is not the case. We prove the theorem using *Euler's formula*, which states that for any connected planar embedded graph with m_v nodes, m_e arcs, and m_f faces the following relation holds:

$$m_v - m_e + m_f = 2.$$

We cannot apply Euler's formula directly to Vor(P), because Vor(P) has half-infinite edges and is therefore not a proper graph. To remedy the situation we add one extra vertex v_∞ "at infinity" to the set of vertices and we connect all half-infinite edges of Vor(P) to this vertex. We now have a connected planar graph to which we can apply Euler's formula. We obtain the following relation between n_v, the number of vertices of Vor(P), n_e, the number of edges of Vor(P), and n, the number of sites:



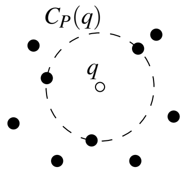
$$(n_v + 1) - n_e + n = 2. \tag{7.1}$$

Moreover, every edge in the augmented graph has exactly two vertices, so if we sum the degrees of all vertices we get twice the number of edges. Because every vertex, including v_∞, has degree at least three we get

$$2n_e \geqslant 3(n_v + 1). \tag{7.2}$$

Together with equation (7.1) this implies the theorem. □

We close this section with a characterization of the edges and vertices of the Voronoi diagram. We know that the edges are parts of bisectors of pairs of sites and that the vertices are intersection points between these bisectors. There is a quadratic number of bisectors, whereas the complexity of the Vor(P) is only linear. Hence, not all bisectors define edges of Vor(P) and not all intersections are vertices of Vor(P). To characterize which bisectors and intersections define features of the Voronoi diagram we make the following definition. For a point q we define the *largest empty circle of q with respect to P*, denoted by $C_P(q)$, as the largest circle with q as its center that does not contain any site of P in its interior. The following theorem characterizes the vertices and edges of the Voronoi diagram.



Theorem 7.4 *For the Voronoi diagram* Vor(P) *of a set of points P the following holds:*

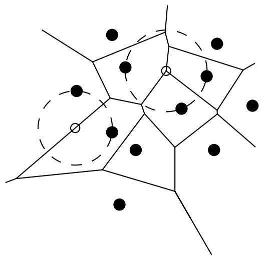
(i) A point q is a vertex of Vor(P) *if and only if its largest empty circle $C_P(q)$ contains three or more sites on its boundary.*
(ii) The bisector between sites p_i and p_j defines an edge of Vor(P) *if and only if there is a point q on the bisector such that $C_P(q)$ contains both p_i and p_j on its boundary but no other site.*

Proof. (i) Suppose there is a point q such that $C_P(q)$ contains three or more sites on its boundary. Let p_i, p_j, and p_k be three of those sites. Since the interior of $C_P(q)$ is empty q must be on the boundary of each of $\mathcal{V}(p_i)$, $\mathcal{V}(p_j)$, and $\mathcal{V}(p_k)$, and q must be a vertex of $\text{Vor}(P)$.

On the other hand, every vertex q of $\text{Vor}(P)$ is incident to at least three edges and, hence, to at least three Voronoi cells $\mathcal{V}(p_i)$, $\mathcal{V}(p_j)$, and $\mathcal{V}(p_k)$. Vertex q must be equidistant to p_i, p_j, and p_k and there cannot be another site closer to q, since otherwise $\mathcal{V}(p_i)$, $\mathcal{V}(p_j)$, and $\mathcal{V}(p_k)$ would not meet at q. Hence, the interior of the circle with p_i, p_j, and p_k on its boundary does not contain any site.

(ii) Suppose there is a point q with the property stated in the theorem. Since $C_P(q)$ does not contain any sites in its interior and p_i and p_j are on its boundary, we have $\text{dist}(q, p_i) = \text{dist}(q, p_j) \leqslant \text{dist}(q, p_k)$ for all $1 \leqslant k \leqslant n$. It follows that q lies on an edge or vertex of $\text{Vor}(P)$. The first part of the theorem implies that q cannot be a vertex of $\text{Vor}(P)$. Hence, q lies on an edge of $\text{Vor}(P)$, which is defined by the bisector of p_i and p_j.

Conversely, let the bisector of p_i and p_j define a Voronoi edge. The largest empty circle of any point q in the interior of this edge must contain p_i and p_j on its boundary and no other sites. $\square$

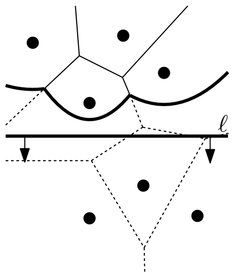


7.2 Computing the Voronoi Diagram

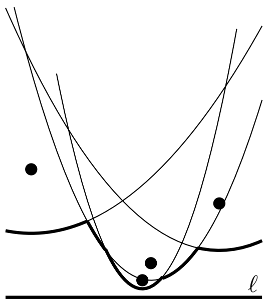
In the previous section we studied the structure of the Voronoi diagram. We now set out to compute it. Observation 7.1 suggests a simple way to do this: for each site p_i, compute the common intersection of the half-planes $h(p_i, p_j)$, with $j \neq i$, using the algorithm presented in Chapter 4. This way we spend $O(n \log n)$ time per Voronoi cell, leading to an $O(n^2 \log n)$ algorithm to compute the whole Voronoi diagram. Can't we do better? After all, the total complexity of the Voronoi diagram is only linear. The answer is yes: the plane sweep algorithm described below—commonly known as *Fortune's algorithm* after its inventor—computes the Voronoi diagram in $O(n \log n)$ time. You may be tempted to look for an even faster algorithm, for example one that runs in linear time. This turns out to be too much to ask: the problem of sorting n real numbers is reducible to the problem of computing Voronoi diagrams, so any algorithm for computing Voronoi diagrams must take $\Omega(n \log n)$ time in the worst case. Hence, Fortune's algorithm is optimal.

The strategy in a plane sweep algorithm is to sweep a horizontal line—the *sweep line*—from top to bottom over the plane. While the sweep is performed information is maintained regarding the structure that one wants to compute. More precisely, information is maintained about the intersection of the structure with the sweep line. While the sweep line moves downwards the information does not change, except at certain special points—the *event points*.

Let's try to apply this general strategy to the computation of the Voronoi diagram of a set $P = \{p_1, p_2, \ldots, p_n\}$ of point sites in the plane. According to the plane sweep paradigm we move a horizontal sweep line ℓ from top to bottom over the plane. The paradigm involves maintaining the intersection of the Voronoi diagram with the sweep line. Unfortunately this is not so easy, because the part of Vor(P) above ℓ depends not only on the sites that lie above ℓ but also on sites below ℓ. Stated differently, when the sweep line reaches the topmost vertex of the Voronoi cell $\mathcal{V}(p_i)$ it has not yet encountered the corresponding site p_i. Hence, we do not have all the information needed to compute the vertex. We are forced to apply the plane sweep paradigm in a slightly different fashion: instead of maintaining the intersection of the Voronoi diagram with the sweep line, we maintain information about the part of the Voronoi diagram of the sites above ℓ that cannot be changed by sites below ℓ.



Denote the closed half-plane above ℓ by ℓ^+. What is the part of the Voronoi diagram above ℓ that cannot be changed anymore? In other words, for which points $q \in \ell^+$ do we know for sure what their nearest site is? The distance of a point $q \in \ell^+$ to any site below ℓ is greater than the distance of q to ℓ itself. Hence, the nearest site of q cannot lie below ℓ if q is at least as near to some site $p_i \in \ell^+$ as q is to ℓ. The locus of points that are closer to some site $p_i \in \ell^+$ than to ℓ is bounded by a parabola. Hence, the locus of points that are closer to any site above ℓ than to ℓ itself is bounded by parabolic arcs. We call this sequence of parabolic arcs the *beach line*. Another way to visualize the beach line is the following. Every site p_i above the sweep line defines a complete parabola β_i. The beach line is the function that—for each x-coordinate—passes through the lowest point of all parabolas.



Observation 7.5 *The beach line is x-monotone, that is, every vertical line intersects it in exactly one point.*

It is easy to see that one parabola can contribute more than once to the beach line. We'll worry later about how many pieces there can be. Notice that the *breakpoints* between the different parabolic arcs forming the beach line lie on edges of the Voronoi diagram. This is not a coincidence: the breakpoints exactly trace out the Voronoi diagram while the sweep line moves from top to bottom. These properties of the beach line can be proved using elementary geometric arguments.

So, instead of maintaining the intersection of Vor(P) with ℓ we maintain the beach line as we move our sweep line ℓ. We do not maintain the beach line explicitly, since it changes continuously as ℓ moves. For the moment let's ignore the issue of how to represent the beach line until we understand where and how its combinatorial structure changes. This happens when a new parabolic arc appears on it, and when a parabolic arc shrinks to a point and disappears.

First we consider the events where a new arc appears on the beach line. One occasion where this happens is when the sweep line ℓ reaches a new site. The parabola defined by this site is at first a degenerate parabola with zero width: a vertical line segment connecting the new site to the beach line. As the sweep line continues to move downward the new parabola gets wider and wider. The part of the new parabola below the old beach line is now a part of the new beach line. Figure 7.2 illustrates this process. We call the event where a new site is encountered a *site event*.

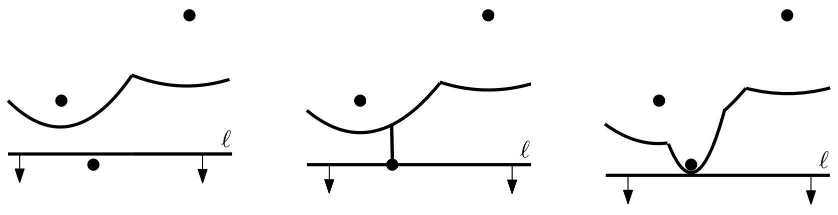
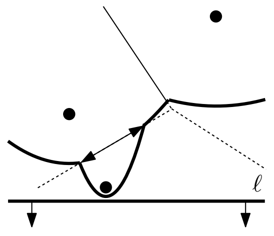


Figure 7.2
A new arc appears on the beach line because a site is encountered

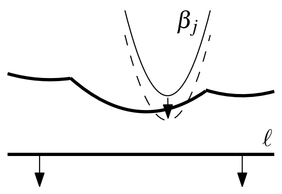


What happens to the Voronoi diagram at a site event? Recall that the breakpoints on the beach line trace out the edges of the Voronoi diagram. At a site event two new breakpoints appear, which start tracing out edges. In fact, the new breakpoints coincide at first, and then move in opposite directions to trace out the same edge. Initially, this edge is not connected to the rest of the Voronoi diagram above the sweep line. Later on—we will see shortly exactly when this will happen—the growing edge will run into another edge, and it becomes connected to the rest of the diagram.

So now we understand what happens at a site event: a new arc appears on the beach line, and a new edge of the Voronoi diagram starts to be traced out. Is it possible that a new arc appears on the beach line in any other way? The answer is no:

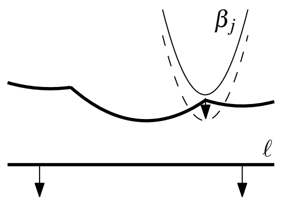
Lemma 7.6 *The only way in which a new arc can appear on the beach line is through a site event.*

Proof. Suppose for a contradiction that an already existing parabola β_j defined by a site p_j breaks through the beach line. There are two ways in which this could happen.



The first possibility is that β_j breaks through in the middle of an arc of a parabola β_i. The moment this is about to happen, β_i and β_j are tangent, that is, they have exactly one point of intersection. Let ℓ_y denote the y-coordinate of the sweep line at the moment of tangency. If $p_j := (p_{j,x}, p_{j,y})$, then the parabola β_j is given by

$$\beta_j \ := \quad y = \frac{1}{2(p_{j,y}-\ell_y)}\left(x^2 - 2p_{j,x}x + p_{j,x}^2 + p_{j,y}^2 - \ell_y^2\right).$$



The formula for β_i is similar, of course. Using that both $p_{j,y}$ and $p_{i,y}$ are larger than ℓ_y, it is easy to show that it is impossible that β_i and β_j have only one point of intersection. Hence, a parabola β_j never breaks through in the middle of an arc of another parabola β_i.

The second possibility is that β_j appears in between two arcs. Let these arcs be part of parabolas β_i and β_k. Let q be the intersection point of β_i and β_k at which β_j is about to appear on the beach line, and assume that β_i is on the beach line left of q and β_k is on the beach line right of q, as in Figure 7.3. Then there is a circle C that passes through p_i, p_j, and p_k, the sites defining the parabolas. This circle is also tangent to the sweep line ℓ. The cyclic order on C, starting at the point of tangency with ℓ and going clockwise, is p_i, p_j, p_k, because β_j is assumed to appear in between the arcs of β_i and β_k. Consider an infinitesimal motion of the sweep line downward while keeping the circle C tangent to ℓ; see Figure 7.3. Then C cannot have empty interior and still pass

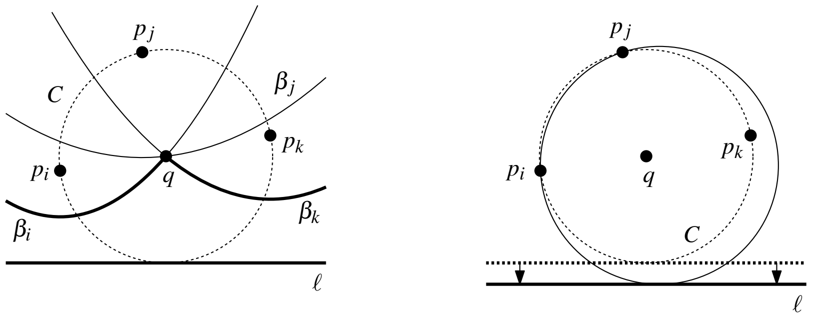


Figure 7.3
The situation when β_j would appear on the beach line, and the circle when the sweep line has proceeded

through p_j: either p_i or p_k will penetrate the interior. Therefore, in a sufficiently small neighborhood of q the parabola β_j cannot appear on the beach line when the sweep line moves downward, because either p_i or p_k will be closer to ℓ than p_j. $\square$

An immediate consequence of the lemma is that the beach line consists of at most $2n-1$ parabolic arcs: each site encountered gives rise to one new arc and the splitting of at most one existing arc into two, and there is no other way an arc can appear on the beach line.

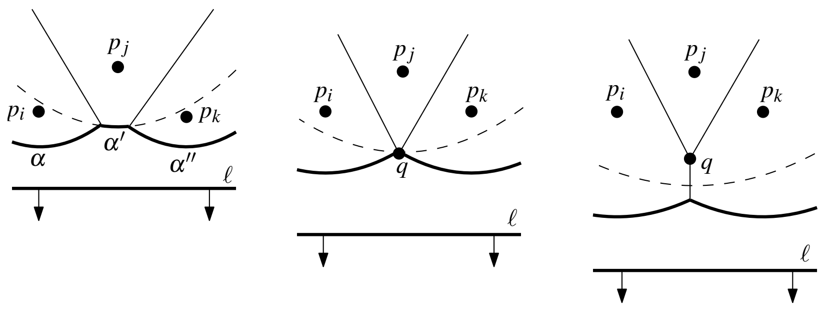


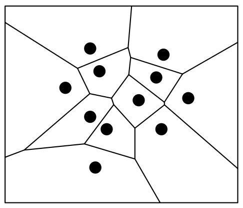
Figure 7.4
An arc disappears from the beach line

The second type of event in the plane sweep algorithm is where an existing arc of the beach line shrinks to a point and disappears, as in Figure 7.4. Let α' be the disappearing arc, and let α and α'' be the two neighboring arcs of α' before it disappears. The arcs α and α'' cannot be part of the same parabola; this possibility can be excluded in the same way as the first possibility in the proof of Lemma 7.6 was excluded. Hence, the three arcs α, α', and α'' are defined by three distinct sites p_i, p_j, and p_k. At the moment α' disappears, the parabolas defined by these three sites pass through a common point q. Point q is equidistant from ℓ and each of the three sites. Hence, there is a circle passing through p_i, p_j, and p_k with q as its center and whose lowest point lies on ℓ. There cannot be a site in the interior of this circle: such a site would be closer to q than q is to ℓ, contradicting the fact that q is on the beach line. It follows that the point q is a vertex of the Voronoi diagram. This is not very surprising, since we observed earlier that the breakpoints on the beach line trace out the Voronoi diagram. So when an arc disappears from the beach line and two breakpoints meet, two edges of the Voronoi diagram meet as well. We call the event where the sweep line reaches the lowest point of a circle through three sites defining consecutive arcs on the beach line a *circle event*. From the above we can conclude the following lemma.

Lemma 7.7 *The only way in which an existing arc can disappear from the beach line is through a circle event.*

Now we know where and how the combinatorial structure of the beach line changes: at a site event a new arc appears, and at a circle event an existing arc drops out. We also know how this relates to the Voronoi diagram under construction: at a site event a new edge starts to grow, and at a circle event two growing edges meet to form a vertex. It remains to find the right data structures to maintain the necessary information during the sweep. Our goal is to compute the Voronoi diagram, so we need a data structure that stores the part of the Voronoi diagram computed thus far. We also need the two 'standard' data structures for any sweep line algorithm: an event queue and a structure that represents the status of the sweep line. Here the latter structure is a representation of the beach line. These data structures are implemented in the following way.

- We store the Voronoi diagram under construction in our usual data structure for subdivisions, the doubly-connected edge list. A Voronoi diagram, however, is not a true subdivision as defined in Chapter 2: it has edges that are half-lines or full lines, and these cannot be represented in a doubly-connected edge list. During the construction this is not a problem, because the representation of the beach line—described next—will make it possible to access the relevant parts of the doubly-connected edge list efficiently during its construction. But after the computation is finished we want to have a valid doubly-connected edge list. To this end we add a big bounding box to our scene, which is large enough so that it contains all vertices of the Voronoi diagram. The final subdivision we compute will then be the bounding box plus the part of the Voronoi diagram inside it.



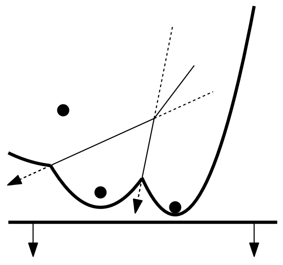
- The beach line is represented by a balanced binary search tree $\mathcal{T}$; it is the status structure. Its leaves correspond to the arcs of the beach line—which is x-monotone—in an ordered manner: the leftmost leaf represents the leftmost arc, the next leaf represents the second leftmost arc, and so on. Each leaf μ stores the site that defines the arc it represents. The internal nodes of $\mathcal{T}$ represent the breakpoints on the beach line. A breakpoint is stored at an internal node by an ordered tuple of sites $\langle p_i, p_j \rangle$, where p_i defines the parabola left of the breakpoint and p_j defines the parabola to the right. Using this representation of the beach line, we can find in $O(\log n)$

time the arc of the beach line lying above a new site. At an internal node, we simply compare the *x*-coordinate of the new site with the *x*-coordinate of the breakpoint, which can be computed from the tuple of sites and the position of the sweep line in constant time. Note that we do not explicitly store the parabolas.

In $\mathcal{T}$ we also store pointers to the other two data structures used during the sweep. Each leaf of $\mathcal{T}$, representing an arc α, stores one pointer to a node in the event queue, namely, the node that represents the circle event in which α will disappear. This pointer is **nil** if no circle event exists where α will disappear, or this circle event hasn't been detected yet. Finally, every internal node ν has a pointer to a half-edge in the doubly-connected edge list of the Voronoi diagram. More precisely, ν has a pointer to one of the half-edges of the edge being traced out by the breakpoint represented by ν.

- The event queue $\mathcal{Q}$ is implemented as a priority queue, where the priority of an event is its *y*-coordinate. It stores the upcoming events that are already known. For a site event we simply store the site itself. For a circle event the event point that we store is the lowest point of the circle, with a pointer to the leaf in $\mathcal{T}$ that represents the arc that will disappear in the event.

All the site events are known in advance, but the circle events are not. This brings us to one final issue that we must discuss, namely the detection of circle events.



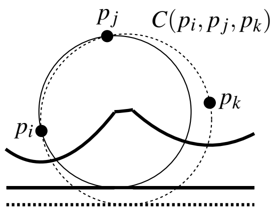
During the sweep the beach line changes its topological structure at every event. This may cause new triples of consecutive arcs to appear on the beach line and it may cause existing triples to disappear. Our algorithm will make sure that for every three consecutive arcs on the beach line that define a potential circle event, the potential event is stored in the event queue $\mathcal{Q}$. There are two subtleties involved in this. First of all, there can be consecutive triples whose two breakpoints do not converge, that is, the directions in which they move are such that they will not meet in the future; this happens when the breakpoints move along two bisectors away from the intersection point. In this case the triple does not define a potential circle event. Secondly, even if a triple has converging breakpoints, the corresponding circle event need not take place: it can happen that the triple disappears (for instance due to the appearance of a new site on the beach line) before the event has taken place. In this case we call the event a *false alarm*.

So what the algorithm does is this. At every event, it checks all the new triples of consecutive arcs that appear. For instance, at a site event we can get three new triples: one where the new arc is the left arc of the triple, one where it is the middle arc, and one where it is the right arc. When such a new triple has converging breakpoints, the event is inserted into the event queue $\mathcal{Q}$. Observe that in the case of a site event, the triple with the new arc being the middle one can never cause a circle event, because the left and right arc of the triple come from the same parabola and therefore the breakpoints must diverge. Furthermore, for all disappearing triples it is checked whether they have a corresponding event in $\mathcal{Q}$. If so, the event is apparently a false alarm, and it is deleted from $\mathcal{Q}$. This can easily be done using the pointers we have from the leaves in $\mathcal{T}$ to the corresponding circle events in $\mathcal{Q}$.

Lemma 7.8 *Every Voronoi vertex is detected by means of a circle event.*

Proof. For a Voronoi vertex q, let p_i, p_j, and p_k be the three sites through which a circle $C(p_i, p_j, p_k)$ passes with no sites in the interior. By Theorem 7.4, such a circle and three sites indeed exist. For simplicity we only prove the case where no other sites lie on $C(p_i, p_j, p_k)$, and the lowest point of $C(p_i, p_j, p_k)$ is not one of the defining sites. Assume without loss of generality that from the lowest point of $C(p_i, p_j, p_k)$, the clockwise traversal of $C(p_i, p_j, p_k)$ encounters the sites p_i, p_j, p_k in this order.

We must show that just before the sweep line reaches the lowest point of $C(p_i, p_j, p_k)$, there are three consecutive arcs α, α' and α'' on the beach line defined by the sites p_i, p_j, and p_k. Only then will the circle event take place. Consider the sweep line an infinitesimal amount before it reaches the lowest point of $C(p_i, p_j, p_k)$. Since $C(p_i, p_j, p_k)$ doesn't contain any other sites inside or on it, there exists a circle through p_i and p_j that is tangent to the sweep line, and doesn't contain sites in the interior. So there are adjacent arcs on the beach line defined by p_i and p_j. Similarly, there are adjacent arcs on the beach line defined by p_j and p_k. It is easy to see that the two arcs defined by p_j are actually the same arc, and it follows that there are three consecutive arcs on the beach line defined by p_i, p_j, and p_k. Therefore, the corresponding circle event is in $\mathcal{Q}$ just before the event takes place, and the Voronoi vertex is detected. $\square$



We can now describe the plane sweep algorithm in detail. Notice that after all events have been handled and the event queue $\mathcal{Q}$ is empty, the beach line hasn't disappeared yet. The breakpoints that are still present correspond to the half-infinite edges of the Voronoi diagram. As stated earlier, a doubly-connected edge list cannot represent half-infinite edges, so we must add a bounding box to the scene to which these edges can be attached. The overall structure of the algorithm is as follows.

Algorithm VORONOIDIAGRAM(P)
Input. A set $P := \{p_1, \dots, p_n\}$ of point sites in the plane.
Output. The Voronoi diagram Vor(P) given inside a bounding box in a doubly-connected edge list $\mathcal{D}$.

```
1.  Initialize the event queue $\mathcal{Q}$ with all site events, initialize an empty status
    structure $\mathcal{T}$ and an empty doubly-connected edge list $\mathcal{D}$.
2.  while $\mathcal{Q}$ is not empty
3.      do Remove the event with largest $y$-coordinate from $\mathcal{Q}$.
4.         if the event is a site event, occurring at site $p_i$
5.           then HANDLESITEEVENT($p_i$)
6.           else HANDLECIRCLEEVENT($\gamma$), where $\gamma$ is the leaf of $\mathcal{T}$ repre-
                 senting the arc that will disappear
7.  The internal nodes still present in $\mathcal{T}$ correspond to the half-infinite edges of
    the Voronoi diagram. Compute a bounding box that contains all vertices of
    the Voronoi diagram in its interior, and attach the half-infinite edges to the
    bounding box by updating the doubly-connected edge list appropriately.
```

8. Traverse the half-edges of the doubly-connected edge list to add the cell records and the pointers to and from them.

The procedures to handle the events are defined as follows.

HANDLESITEEVENT(p_i)
1. If $\mathcal{T}$ is empty, insert p_i into it (so that $\mathcal{T}$ consists of a single leaf storing p_i) and return. Otherwise, continue with steps 2– 5.
2. Search in $\mathcal{T}$ for the arc α vertically above p_i. If the leaf representing α has a pointer to a circle event in $\mathcal{Q}$, then this circle event is a false alarm and it must be deleted from $\mathcal{Q}$.
3. Replace the leaf of $\mathcal{T}$ that represents α with a subtree having three leaves. The middle leaf stores the new site p_i and the other two leaves store the site p_j that was originally stored with α. Store the tuples $\langle p_j, p_i\rangle$ and $\langle p_i, p_j\rangle$ representing the new breakpoints at the two new internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary.
4. Create new half-edge records in the Voronoi diagram structure for the edge separating $\mathcal{V}(p_i)$ and $\mathcal{V}(p_j)$, which will be traced out by the two new breakpoints.
5. Check the triple of consecutive arcs where the new arc for p_i is the left arc to see if the breakpoints converge. If so, insert the circle event into $\mathcal{Q}$ and add pointers between the node in $\mathcal{T}$ and the node in $\mathcal{Q}$. Do the same for the triple where the new arc is the right arc.

HANDLECIRCLEEVENT(γ)
1. Delete the leaf γ that represents the disappearing arc α from $\mathcal{T}$. Update the tuples representing the breakpoints at the internal nodes. Perform rebalancing operations on $\mathcal{T}$ if necessary. Delete all circle events involving α from $\mathcal{Q}$; these can be found using the pointers from the predecessor and the successor of γ in $\mathcal{T}$. (The circle event where α is the middle arc is currently being handled, and has already been deleted from $\mathcal{Q}$.)
2. Add the center of the circle causing the event as a vertex record to the doubly-connected edge list $\mathcal{D}$ storing the Voronoi diagram under construction. Create two half-edge records corresponding to the new breakpoint of the beach line. Set the pointers between them appropriately. Attach the three new records to the half-edge records that end at the vertex.
3. Check the new triple of consecutive arcs that has the former left neighbor of α as its middle arc to see if the two breakpoints of the triple converge. If so, insert the corresponding circle event into $\mathcal{Q}$. and set pointers between the new circle event in $\mathcal{Q}$ and the corresponding leaf of $\mathcal{T}$. Do the same for the triple where the former right neighbor is the middle arc.

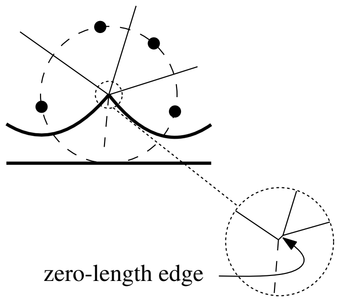
Lemma 7.9 *The algorithm runs in $O(n \log n)$ time and it uses $O(n)$ storage.*

Proof. The primitive operations on the tree $\mathcal{T}$ and the event queue $\mathcal{Q}$, such as inserting or deleting an element, take $O(\log n)$ time each. The primitive operations on the doubly-connected edge list take constant time. To handle an event we do a constant number of such primitive operations, so we spend

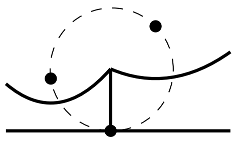
$O(\log n)$ time to process an event. Obviously, there are n site events. As for the number of circle events, we observe that every such event that is processed defines a vertex of $\mathrm{Vor}(P)$. Note that false alarms are deleted from $\mathcal{Q}$ before they are processed. They are created and deleted while processing another, real event, and the time we spend on them is subsumed under the time we spend to process this event. Hence, the number of circle events that we process is at most $2n-5$. The time and storage bounds follow. $\square$

Before we state the final result of this section we should say a few words about degenerate cases.

The algorithm handles the events from top to bottom, so there is a degeneracy when two or more events lie on a common horizontal line. This happens, for example, when there are two sites with the same y-coordinate. These events can be handled in any order when their x-coordinates are distinct, so we can break ties between events with the same y-coordinate but with different x-coordinates arbitrarily. However, if this happens right at the start of the algorithm, that is, if the second site event has the same y-coordinate as the first site event, then special code is needed because there is no arc above the second site yet. Now suppose there are event points that coincide. For instance, there will be several coincident circle events when there are four or more co-circular sites, such that the interior of the circle through them is empty. The center of this circle is a vertex of the Voronoi diagram. The degree of this vertex is at least four. We could write special code to handle such degenerate cases, but there is no need to do so. What will happen if we let the algorithm handle these events in arbitrary order? Instead of producing a vertex with degree four, it will just produce two vertices with degree three at the same location, with a zero length edge between them. These degenerate edges can be removed in a post-processing step, if required.



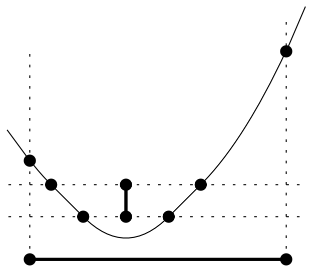
Besides these degeneracies in choosing the order of the events we may also encounter degeneracies while handling an event. This occurs when a site p_i that we process happens to be located exactly below the breakpoint between two arcs on the beach line. In this case the algorithm splits either of these two arcs and inserts the arc for p_i in between the two pieces, one of which has zero length. This piece of zero length now is the middle arc of a triple that defines a circle event. The lowest point of this circle coincides with p_i. The algorithm inserts this circle event into the event queue $\mathcal{Q}$, because there are three consecutive arcs on the beach line that define it. When this circle event is handled, a vertex of the Voronoi diagram is correctly created and the zero length arc can be deleted later. Another degeneracy occurs when three consecutive arcs on the beach line are defined by three collinear sites. Then these sites don't define a circle, nor a circle event.



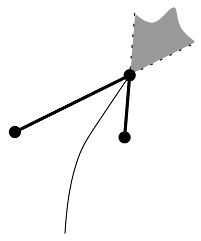
We conclude that the above algorithm handles degenerate cases correctly.

Theorem 7.10 *The Voronoi diagram of a set of n point sites in the plane can be computed with a sweep line algorithm in $O(n \log n)$ time using $O(n)$ storage.*

7.3 Voronoi Diagrams of Line Segments



The Voronoi diagram can also be defined for objects other than points. The distance from a point in the plane to an object is then measured to the closest point on the object. Whereas the bisector of two points is simply a line, the bisector of two disjoint line segments has a more complex shape. It consists of up to seven parts, where each part is either a line segment or a parabolic arc. Parabolic arcs occur if the closest point of one line segment is an endpoint and the closest point of the other line segment is in its interior. In all other cases the bisector part is straight. Although bisectors and therefore the Voronoi diagram are somewhat more complex, the number of vertices, edges, and faces in the Voronoi diagram of n disjoint line segments is still only $O(n)$.



Assume for a moment that we allow the line segments to be non-crossing, that is, we allow them to share endpoints. Then a whole region of the plane can be equally close to two line segments via their common endpoint, and bisectors are not even curves anymore. To avoid the complications that arise in defining and computing Voronoi diagrams of line segments that share endpoints, we will simply assume here that all line segments are strictly disjoint. In many applications we can simply shorten the line segments very slightly to obtain disjoint line segments.

The sweep line algorithm for points can be adapted to the case of line segment sites. Let $S = \{s_1, \ldots, s_n\}$ be a set of n disjoint line segments. We call the segments of S *sites* as before, and use the terms *site endpoint* and *site interior* in the following description.

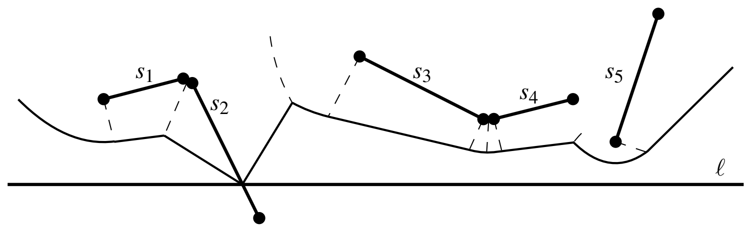


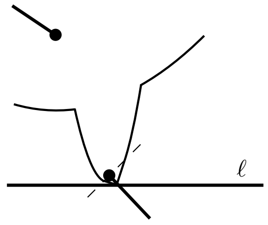
Figure 7.5
The beach line for a set of line segment sites. The breakpoints trace the dashed arcs, which include the Voronoi edges

Recall that our algorithm for point sites maintained a beach line: a piecewise parabolic x-monotone curve such that, for points on the curve, the distance to the closest site above the sweep line is equal to the distance to the sweep line. What does the beach line look like when the sites are segments? First we note that a line segment site may be partially above and partially below the sweep line. When defining the beach line, we consider only those parts of the sites that are above the sweep line. Hence, for a given position of the sweep line ℓ, the beach line consists of those points such that the distance to the closest portion of a site above ℓ is equal to the distance to ℓ. This means that the beach line now consists of parabolic arcs and straight line segments. A parabolic arc arises when that part of the beach line is closest to a site endpoint, and a straight line segment arises when that part of the beach line is closest to a site interior. Note that if a site interior intersects ℓ, then the beach line will have two straight line segments ending at the intersection—see site s_2 in Figure 7.5.

The breakpoints between parabolic arcs and straight segments on the beach line arise in several different ways. Figure 7.5 illustrates this. Assume any position ℓ of the sweep line during the downward sweep to analyze the types of breakpoint:

- If a point p is closest to two site endpoints while being equidistant from them and ℓ, then p is a breakpoint that traces a line segment (as in the point site case).
- If a point p is closest to two site interiors while being equidistant from them and ℓ, then p is a breakpoint that traces a line segment.
- If a point p is closest to a site endpoint and a site interior of different sites while being equidistant from them and ℓ, then p is a breakpoint that traces a parabolic arc.
- If a point p is closest to a site endpoint, the shortest distance is realized by a segment that is perpendicular to the line segment site, and p has the same distance from ℓ, then p is a breakpoint that traces a line segment.
- If a site interior intersects the sweep line, then the intersection is a breakpoint that traces a line segment (the site interior).

In the fourth and fifth cases, the breakpoint does not actually trace an arc of the Voronoi diagram, because only one site is involved. For the proper operation of the algorithm, dealing with such breakpoints and corresponding events is still necessary.



As in the sweep line algorithm for point sites, we again have site events and circle events. A site event occurs when the sweep line reaches a site endpoint. Obviously, site events at upper endpoints should be handled differently from site events at lower endpoints. At an upper endpoint, an arc of the beach line is split into two, and in between, four new arcs appear. The breakpoints between these four arcs are of the last two types. At a lower endpoint, the breakpoint that is the intersection of the site interior with the sweep line is replaced by two breakpoints of the fourth type, with a parabolic arc in between (for the newly discovered site endpoint).

Similarly, there are several types of circle event. They all correspond to the disappearance of an arc of the beach line, and they occur when the sweep line reaches the bottom of an empty circle that is defined by two or three sites above the sweep line. The centers of these empty circles are at locations where two consecutive breakpoints will meet. Depending on the types of the breakpoints that meet, several different cases can be distinguished and handled. If the two breakpoints are of any of the first three types, then three sites are involved. If one of the breakpoints is of the fourth type, then only two sites are involved. Breakpoints of the fifth type cannot be involved for disjoint line segments.

Notice that the Voronoi diagram that the algorithm computes is a subdivision with straight edges and parabolic arcs. Can we store this type of subdivision in a doubly-connected edge list? This is indeed possible, and the structure need not even be adapted. With each face, we store the corresponding site, so for any half-edge $\vec{e}$ we can determine the two sites that have e on their bisector (using *IncidentFace*$(\vec{e})$ and *IncidentFace*$(\mathit{Twin}(\vec{e}))$). Since we can also easily find the two vertices between which the edge lies (*Origin*$(\vec{e})$ and *Origin*$(\mathit{Twin}(\vec{e}))$), we can determine the shape of any edge in constant time.

The whole sweep line algorithm is now just an extension of the one for point sites, with more cases to be distinguished and handled. However, the algorithm still has only $O(n)$ events, and each can be handled in $O(\log n)$ time.

Theorem 7.11 *The Voronoi diagram of a set of n disjoint line segment sites can be computed in $O(n \log n)$ time using $O(n)$ storage.*

One of the applications of the Voronoi diagram for line segments is in *motion planning* (covered more extensively in Chapter 13). Assume that a set of obstacles is given, consisting of n line segments in total, and that we have a robot $\mathcal{R}$. We assume that the robot can move freely in all directions, and is approximated well by an enclosing disc D. Suppose that we wish to find a collision-free motion from one location of the robot to another, or to decide that none exists.

One motion-planning technique is called *retraction*. The idea of retraction is that the arcs of the Voronoi diagram define the middle between the line segments, and therefore define a path with the most clearance. So a path over the arcs of the Voronoi diagram is the best option for a collision-free path. Figure 7.6 shows a set of line segments inside a rectangle, together with a Voronoi diagram of the line segments and the sides of the rectangle.

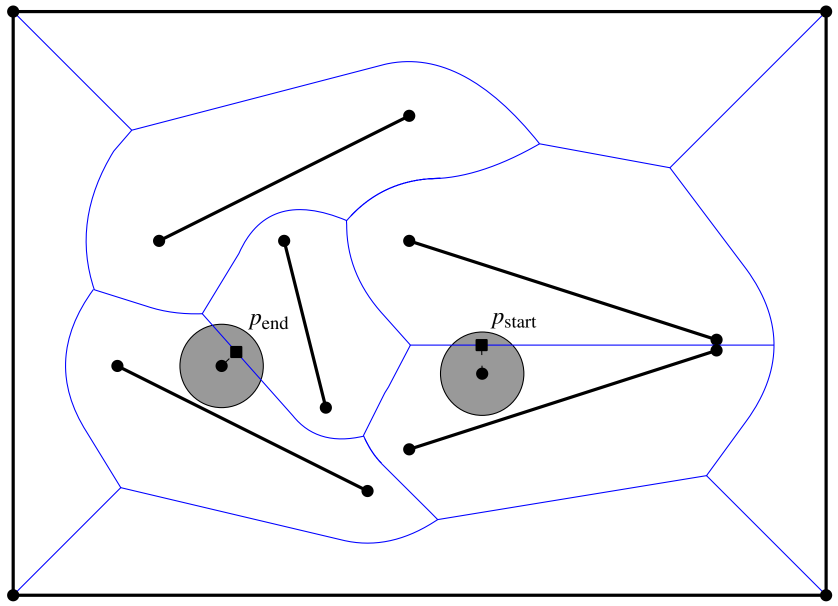


Figure 7.6
Voronoi diagram of line segments, and start and end positions of a disc

We can determine a collision-free path between two disc positions amidst a set of line segments with the following algorithm.

```
Algorithm RETRACTION($S, q_{\text{start}}, q_{\text{end}}, r$)
Input. A set $S := \{s_1, \dots, s_n\}$ of disjoint line segments in the plane, and two
    discs $D_{\text{start}}$ and $D_{\text{end}}$ centered at $q_{\text{start}}$ and $q_{\text{end}}$ with radius $r$. The two disc
    positions do not intersect any line segment of $S$.
Output. A path that connects $q_{\text{start}}$ to $q_{\text{end}}$ such that no disc of radius $r$ with its
    center on the path intersects any line segment of $S$. If no such path exists, this
    is reported.
1.  Compute the Voronoi diagram Vor($S$) of $S$ inside a sufficiently large bound-
    ing box.
2.  Locate the cells of Vor($P$) that contain $q_{\text{start}}$ and $q_{\text{end}}$.
3.  Determine the point $p_{\text{start}}$ on Vor($S$) by moving $q_{\text{start}}$ away from the nearest
    line segment in $S$. Similarly, determine the point $p_{\text{end}}$ on Vor($S$) by moving
    $q_{\text{end}}$ away from the nearest line segment in $S$. Add $p_{\text{start}}$ and $p_{\text{end}}$ as vertices
    to Vor($S$), splitting the arcs on which they lie into two.
4.  Let $\mathcal{G}$ be the graph corresponding to the vertices and edges of the Voronoi
    diagram. Remove all edges from $\mathcal{G}$ for which the smallest distance to the
    nearest sites is smaller than or equal to $r$.
5.  Determine with depth-first search whether a path exists from $p_{\text{start}}$ to $p_{\text{end}}$
    in $\mathcal{G}$. If so, report the line segment from $q_{\text{start}}$ to $p_{\text{start}}$, the path in $\mathcal{G}$ from
    $p_{\text{start}}$ to $p_{\text{end}}$, and the line segment from $p_{\text{end}}$ to $q_{\text{end}}$ as the path. Otherwise,
    report that no path exists.
```

The line segment connecting q_{start} to p_{start} cannot give a collision, because the disc only moves further away from the nearest obstacle. Similarly, the line segment between p_{end} and q_{end} is collision-free. For any two discs centered on the Voronoi diagram, a collision-free path between them exists on the Voronoi diagram if and only if such a path exists at all. Hence, for a disc-shaped robot, a path is found if one exists.

Theorem 7.12 *Given n disjoint line segment obstacles and a disc-shaped robot, the existence of a collision-free path between two positions of the robot can be determined in $O(n \log n)$ time using $O(n)$ storage.*

7.4 Farthest-Point Voronoi Diagrams

We now continue with a different application where Voronoi diagrams are needed. When objects are manufactured, slight deviations in the shapes of the objects will occur. When the objects need to be perfectly round, the manufactured objects are tested for their roundness. This is done by *coordinate measurement machines*, which sample points on the surface of the object. Assume that we have constructed a disc, and wish to determine its roundness. The machine gives us a set P of points in the plane that lie nearly on a circle. The *roundness* of a set of points is defined as the width of the smallest-width annulus that contains the points. An annulus is the region between two concentric circles, and its width is the difference between the radii of those circles.

The smallest-width annulus must of course have some points of the set P on its bounding circles. Let us call the outer circle C_{outer} and the inner circle C_{inner}.

Clearly, there must be at least one point on C_{outer}, otherwise we can reduce the size of C_{outer}, and at least one point on C_{inner}, otherwise we can increase the size of C_{inner}. But one point on each bounding circle cannot give us a smallest-width annulus yet. There appear to be three different cases, each with a total of four points on the two circles (Figure 7.7):

- C_{outer} contains at least three points of P, and C_{inner} contains at least one point of P.
- C_{outer} contains at least one point of P, and C_{inner} contains at least three points of P.
- C_{outer} and C_{inner} both contain two points of P.

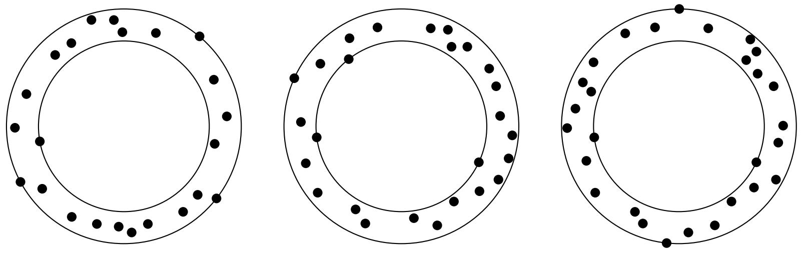
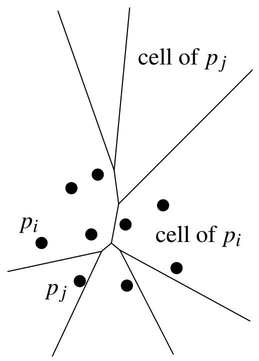


Figure 7.7
Three cases of the smallest-width annulus

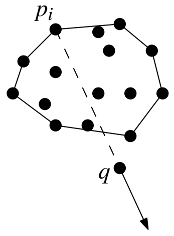
If C_{inner} or C_{outer} contains fewer points than listed in any of these cases, then we can always find an annulus with a smaller width. The problem of finding the smallest-width annulus enclosing a given point set looks similar to the problem of finding the smallest disc enclosing a point set, studied in Section 4.7. The technique we used for the smallest-disc problem, however, does not work for the smallest-width annulus: the property that an added point that does not lie in the optimal annulus so far must always lie on the boundary of the new optimal annulus does not hold.



Finding the smallest-width annulus is equivalent to finding its center point. Once the center point—let's call it q—is fixed, the annulus is determined by the points of P that are closest to and farthest from q. If we have the Voronoi diagram of P, then the closest point is the one in whose cell q lies. It turns out that a similar structure exists for the farthest point, namely the *farthest-point Voronoi diagram*. This divides the plane into cells in which the same point of P is the farthest point. The farthest-point Voronoi cell of a point p_i is the intersection of $n-1$ half-planes, just as for a standard Voronoi cell, but we take the "other sides" of the bisectors, the sides where p_i is farther away. Hence, all cells of the farthest-point Voronoi diagram are convex. Not every point of P has a cell in the farthest-point Voronoi diagram: the intersections of the half-planes can be empty. It is not hard to see that for any point in the plane, its farthest point in the set P must be a point that lies on the convex hull of P. Therefore, a point that lies inside the convex hull cannot have a cell in the farthest-point Voronoi diagram.

Observation 7.13 *Given a set P of points in the plane, a point of P has a cell in the farthest-point Voronoi diagram if and only if it is a vertex of the convex hull of P.*

We can prove more properties of the farthest-point Voronoi diagram. Suppose that a point $p_i \in P$ lies on the convex hull, and let q be some point in the plane for which p_i is the farthest point. Let $\ell(p_i,q)$ be the line through p_i and q. Then all points on the half-line starting at q, contained in $\ell(p_i,q)$, and not containing p_i, must also be in the farthest-point Voronoi cell of p_i. This implies that all cells are unbounded. The vertices and edges of the farthest-point Voronoi diagram form a tree-like structure (in the graph sense), because the diagram is connected and does not have cycles. A cycle would imply a bounded cell.



We can show that the farthest-point Voronoi diagram of n points has $O(n)$ vertices, edges, and cells (see also Exercise 7.14). There is another interesting property: the center of the smallest enclosing disc (see Section 4.7) is either a vertex of the farthest-point Voronoi diagram or the midpoint of two sites defining an edge of the farthest-point Voronoi diagram. In the former case, there are three equidistant farthest points, and in the latter case, two. Clearly, the center of the smallest enclosing disc cannot have just one point that is farthest from it.

Since the farthest-point Voronoi diagram has half-infinite edges, we cannot store it in a doubly-connected edge list, but we can adapt the structure slightly to deal with such subdivisions. We use a special vertex-like record as the origin of each half-edge that has no real vertex as its origin. These new records store the direction of the half-infinite edge instead of coordinates. Furthermore, half-edge records corresending to half-infinite edges have either $Next(\vec{e})$ or $Prev(\vec{e})$ undefined. We shall still use the term "doubly-connected edge list" for this adapted version.

We now present an algorithm to compute the farthest-point Voronoi diagram of a set P of n points in the plane. First, we compute the convex hull of P, take its vertices, and put them in random order. Let this random order be $p_1,\dots,p_h$. We remove the points $p_h,\dots,p_4$ one by one from the cyclic order, and when removing p_i, store its clockwise neighbor $cw(p_i)$ and counterclockwise neighbor $ccw(p_i)$ at the time of removal. After a point has been removed, it cannot be the clockwise or counterclockwise neighbor anymore of points removed later.

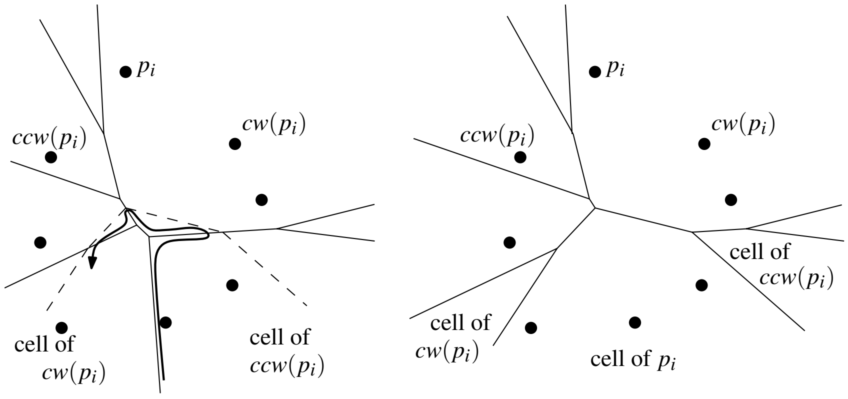
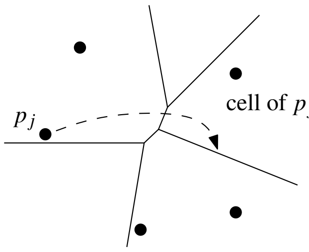


Figure 7.8
Addition of a point p_i to the farthest-point Voronoi diagram of $p_1,\dots,p_{i-1}$

We compute the farthest-point Voronoi diagram of p_1,p_2,p_3 to initialize the incremental construction. Then we insert the remaining points $p_4, \ldots, p_h$ while constructing the farthest-point Voronoi diagram. To be able to add the farthest-point Voronoi cell of p_i efficiently, given the farthest-point Voronoi diagram of $\{p_1, \ldots, p_{i-1}\}$, we maintain a pointer for each point p_j, $1 \leqslant j < i$, to the half-infinite half-edge of the doubly-connected edge list that is most counterclockwise in a traversal of the boundary of the farthest-point Voronoi cell of p_j.



We now look at the addition of the cell of p_i in more detail, see Figure 7.8. The cell will come "in between" the cells of $cw(p_i)$ and $ccw(p_i)$. Just before p_i is added, $cw(p_i)$ and $ccw(p_i)$ are each other's neighbors on the convex hull of $\{p_1, \ldots, p_{i-1}\}$, so their cells are separated by a half-infinite edge that is part of their bisector. The point $ccw(p_i)$ has a pointer to this edge. The bisector of p_i and $ccw(p_i)$ will give a new half-infinite edge that lies in the farthest-point Voronoi cell of $ccw(p_i)$, and is part of the boundary of the farthest-point Voronoi cell of p_i. We traverse the cell of $ccw(p_i)$ in the clockwise direction to see which edge the bisector intersects. On the other side of this edge is the farthest-point Voronoi cell of another point p_j from $\{p_1, \ldots, p_{i-1}\}$, and the bisector of p_j and p_i will also give an edge of the farthest-point Voronoi cell of p_i. We again traverse the cell of p_j in the clockwise direction to determine where the other insertion of the cell boundary and the bisector is located. By tracing cell boundaries in clockwise order, we trace the farthest-point Voronoi cell in counterclockwise order. The last bisector that we will find is with $cw(p_i)$, and it will give a new half-infinite edge in the farthest-point Voronoi diagram. All new edges found are added to the doubly-connected edge list representation, after which all edges that lie inside the farthest-point Voronoi cell of p_i are removed. They are no longer valid edges of the farthest-point Voronoi diagram of $\{p_1, \ldots, p_i\}$.

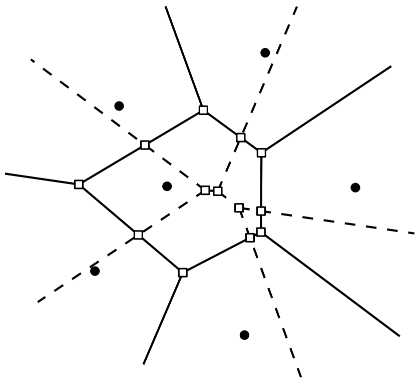
In short, the insertion of the next farthest-point Voronoi cell is done by tracing the new cell with the help of the existing diagram, adding the new edges, and removing the edges that have become obsolete.

Theorem 7.14 *Given a set of n points in the plane, its farthest-point Voronoi diagram can be computed in $O(n \log n)$ expected time using $O(n)$ storage.*

Proof. It takes $O(n \log n)$ time to compute the h points on the convex hull in counterclockwise order. The farthest-point Voronoi diagram actually takes only $O(h)$ expected time to construct after we have the points on the convex hull in sorted order. To see this, we apply backwards analysis. We consider the situation after the insertion of the cell of p_i. We observe that if the cell of p_i has k edges on its boundary, then the traversal performed to trace this cell visited k cells in the farthest-point Voronoi diagram of $\{p_1, \ldots, p_{i-1}\}$, and visited at most $4k-6$ boundary edges of these cells in total.

The farthest-point Voronoi diagram of $\{p_1, \ldots, p_i\}$ has at most $2i-3$ edges (see Exercise 7.14), each used by two cells. Since every point of $\{p_1, \ldots, p_i\}$ has the same probability of having been the last one added, the expected size of the cell of p_i is less than four. Hence, the expected time needed for each insertion is $O(1)$, and the algorithm runs in $O(h)$ expected time. $\square$

Now we return to the problem of computing the smallest-width annulus. Suppose that the smallest-width annulus is such that C_{inner} contains at least three points of P. Then its center is a vertex of the normal Voronoi diagram of P. Similarly, if the smallest-width annulus is such that C_{outer} contains at least three points of P, its center is a vertex of the farthest-point Voronoi diagram of P. Finally, if the smallest-width annulus is such that C_{inner} and C_{outer} both contain two points of P, then its center must lie on an edge of the Voronoi diagram and on an edge of the farthest-point Voronoi diagram simultaneously. This means that we can obtain a reasonably small set of points that must contain the center of a smallest-width annulus.



To do this, we generate the vertices of the *overlay* of the Voronoi diagram and the farthest-point Voronoi diagram. The vertices of the overlay are exactly the candidate centers of the smallest-width annulus, covering all three cases. We don't really need to compute the overlay itself. Once we know a vertex and the four points that determine C_{inner} and C_{outer}, we can compute the smallest-width annulus of those four points directly in $O(1)$ time. This is a candidate for the smallest-width annulus.

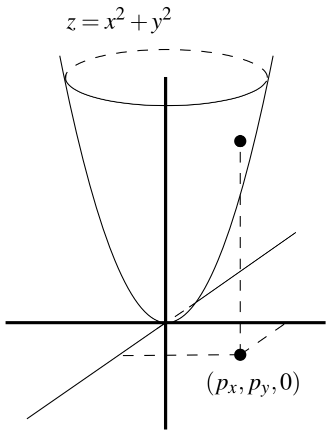
The whole algorithm to compute the smallest-width annulus of a set P of n points in the plane is as follows. Compute the Voronoi diagram and the farthest-point Voronoi diagram of P. For each vertex of the farthest-point Voronoi diagram, determine the point of P that is closest. For each vertex of the normal Voronoi diagram, determine the point of P that is farthest. This gives us $O(n)$ sets of four points that define the candidate annuli in the first and second cases. Next, for every pair of edges, one from each of the diagrams, test if they intersect. If so, we have another set of four points that forms a candidate annulus. For all candidates of all three types, choose the one that gives the smallest-width annulus as the solution.

Theorem 7.15 *Given a set P of n points in the plane, the smallest-width annulus (and the roundness) can be determined in $O(n^2)$ time using $O(n)$ storage.*

7.5 Notes and Comments

Although it is beyond the scope of this book to give an extensive survey of the history of Voronoi diagrams it is appropriate to make a few historical remarks. Voronoi diagrams are often attributed to Dirichlet [148]—hence the name *Dirichlet tessellations* that is sometimes used—and Voronoi [379, 380]. They can be found in Descartes's treatment of cosmic fragmentation in Part III of his *Principia Philosophiae*, published in 1644. In the twentieth century, the Voronoi diagram was rediscovered several times. In biology this even happened twice in a very short period. In 1965 Brown [75] studied the intensity of trees in a forest. He defined the *area potentially available* to a tree, which was in fact the Voronoi cell of that tree. One year later Mead [272] used the same concept for plants, calling the Voronoi cells *plant polygons*. Now, there is an impressive amount of literature concerning Voronoi diagrams and their applications in all kinds of research areas. The book by Okabe et al. [297]

contains an ample treatment of Voronoi diagrams and their applications. We confine ourselves in this section to a discussion of the various aspects of Voronoi diagrams encountered in the computational geometry literature.



In this chapter we have proved some properties of the Voronoi diagram, but it has many more. For example, if one connects all the pairs of sites whose Voronoi cells are adjacent then the resulting set of segments forms a triangulation of the point set, called the Delaunay triangulation. This triangulation, which has some very nice properties, is the topic of Chapter 9.

There is a beautiful connection between Voronoi diagrams and convex polyhedra. Consider the transformation that maps a point $p = (p_x, p_y)$ in $\mathbb{E}^2$ to the non-vertical plane $h(p): \; z = 2p_x x + 2p_y y - (p_x^2 + p_y^2)$ in $\mathbb{E}^3$. Geometrically, $h(p)$ is the plane that is tangent to the unit paraboloid $\mathcal{U}: \; z = x^2 + y^2$ at the point vertically above $(p_x, p_y, 0)$. For a set P of point sites in the plane, let $H(P)$ be the set of planes that are the images of the sites in P. Now consider the convex polyhedron $\mathcal{P}$ that is the intersection of all positive half-spaces defined by the planes in $H(P)$, that is, $\mathcal{P} := \bigcap_{h \in H(P)} h^+$, where h^+ denotes the half-space above h. Surprisingly, if we project the edges and vertices of the polyhedron vertically downwards onto the xy-plane, we get the Voronoi diagram of P [167]. See Chapter 11 for a more extensive description of this transformation. A similar transformation exists for the farthest-point Voronoi diagram.

We have studied Voronoi diagrams in their most basic setting, namely for a set of point sites in the Euclidean plane. The first optimal $O(n \log n)$ time algorithm for this case was a divide-and-conquer algorithm presented by Shamos and Hoey [350]; since then, many other optimal algorithms have been developed. The plane sweep algorithm that we described is due to Fortune [183]. Fortune's original description of the algorithm is a little different from ours, which follows the interpretation of the algorithm given by Guibas and Stolfi [203].

Voronoi diagrams can be generalized in many ways [28, 297]. One generalization is to point sets in higher-dimensional spaces. In $\mathbb{E}^d$, the maximum combinatorial complexity of the Voronoi diagram of a set of n point sites (the maximum number of vertices, edges, and so on, of the diagram) is $\Theta(n^{\lceil d/2 \rceil})$ [239] and it can be computed in $O(n \log n + n^{\lceil d/2 \rceil})$ optimal time [93, 133, 346]. The fact that the dual of the Voronoi diagram is a triangulation of the set of sites, and the connection between Voronoi diagrams and convex polyhedra as discussed above still hold in higher dimensions.

Another generalization concerns the metric that is used. In the L_1-metric, or Manhattan metric, the distance between two points p and q is defined as

$$\text{dist}_1(p, q) := |p_x - q_x| + |p_y - q_y|,$$

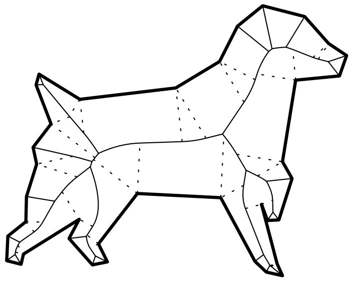
the sum of the absolute differences in the x- and y-coordinates. In a Voronoi diagram in the L_1-metric, all edges are horizontal, vertical, or diagonal (at an angle of 45° to the coordinate axes). In the more general L_p-metric, the distance between two points p and q is defined as

$$\text{dist}_p(p, q) := \sqrt[p]{|p_x - q_x|^p + |p_y - q_y|^p}.$$

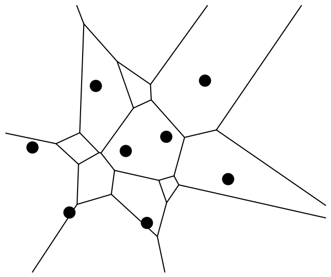
Note that the L_2-metric is simply the Euclidean metric. There are several papers dealing with Voronoi diagrams in these metrics [118, 248, 252]. One can also define a distance function by assigning a weight to each site. Now the distance from a site to a point is the Euclidean distance to the point, plus its additive weight. The resulting diagrams are called weighted Voronoi diagrams [183]. The weight can also be used to define the distance from a site to a point as the Euclidean distance times the weight. Diagrams based on this multiplicatively weighted distance are also called weighted Voronoi diagrams [29]. Power diagrams [25, 26, 27, 30] are another generalization of Voronoi diagrams where a different distance function is used. It is even possible to drop the distance function altogether and define the Voronoi diagram in terms only of bisectors between pairs of sites. Such diagrams are called abstract Voronoi diagrams [240, 241, 242, 274].

Other generalizations concern the shape of the sites. We have seen the Voronoi diagram of a set of disjoint line segments in this chapter. We discussed the application of this diagram to motion planning using the retraction technique; Chapter 13 discusses motion planning in general.

An important special case of the Voronoi diagram of line segments is the Voronoi diagram of the edges of a simple polygon, interior to the polygon itself. Since the edges share endpoints, there can be whole regions inside the polygon where two edges are equally close. This occurs at every reflex vertex of the polygon. The Voronoi diagram is the subdivision of the interior of the polygon into faces where one or two edges are the closest. This Voronoi diagram is also known as the medial axis or skeleton, and it has applications in shape analysis [366, 377]. The medial axis can be computed in time linear in the number of edges of the polygon [123].



Instead of partitioning the space into regions according to the closest sites, one can also partition it according to the k closest sites, for some $1 \leqslant k \leqslant n-1$. The diagrams obtained in this way are called higher-order Voronoi diagrams, and, for given k, the diagram is called the order-k Voronoi diagram [6, 31, 70, 98]. Note that the order-1 Voronoi diagram is nothing more than the standard Voronoi diagram. The order-$(n-1)$ Voronoi diagram is the farthest-point Voronoi diagram, because the Voronoi cell of a point p_i is now the region of points for which p_i is the farthest site. The maximum complexity of the order-k Voronoi diagram of a set of n point sites in the plane is $\Theta(k(n-k))$ [249]. Currently the best known algorithms for computing the order-k Voronoi diagram run in $O(n\log^3 n + nk)$ time [6] and in $O(n\log n + nk2^{c\log^* k})$ time [326], where c is a constant.



The farthest-point Voronoi diagram takes $O(n\log n)$ time to compute, but if the points are in convex position and are given in the order along the convex hull, then there exists a simple $O(n)$ expected-time algorithm [116], given in this chapter, and also an $O(n)$ time deterministic algorithm [11]. Testing the roundness of an object or set of points is a problem that arises in metrology, the science of measurement. Several definitions of roundness exist, the one used in this chapter being the most widely accepted one. A quadratic-time algorithm for the roundness problem was given by Ebarra et al. [155]. A complex, subquadratic-time algorithm was suggested by Agarwal and Sharir [9]. In special cases that correspond to point sets that may occur in practice, linear-time or near-linear-time algorithms exist [52, 142, 187]. A survey of computational metrology has been given by Yap and Chang [396].

7.6 Exercises

7.1 Prove that for any $n > 3$ there is a set of n point sites in the plane such that one of the cells of $\text{Vor}(P)$ has $n-1$ vertices.

7.2 Show that Theorem 7.3 implies that the average number of vertices of a Voronoi cell is less than six.

7.3 Show that $\Omega(n \log n)$ is a lower bound for computing Voronoi diagrams by reducing the sorting problem to the problem of computing Voronoi diagrams. You can assume that the Voronoi diagram algorithm should be able to compute for every vertex of the Voronoi diagram its incident edges in cyclic order around the vertex.

7.4 Prove that the breakpoints of the beach line, as defined in Section 7.2, trace out the edges of the Voronoi diagram while the sweep line moves from top to bottom.

7.5 Give an example where the parabola defined by some site p_i contributes more than one arc to the beach line. Can you give an example where it contributes a linear number of arcs?

7.6 Give an example of six sites such that the plane sweep algorithm encounters the six site events before any of the circle events. The sites should lie in general position: no three sites on a line and no four sites on a circle.

7.7 Do the breakpoints of the beach line always move downwards when the sweep line moves downwards? Prove this or give a counterexample.

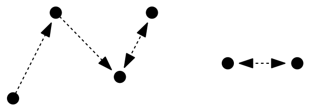
7.8 Write a procedure to compute a big enough bounding box from the incomplete doubly-connected edge list and the tree $\mathcal{T}$ after the sweep is completed. The box should contain all sites and all Voronoi vertices.

7.9 Write a procedure to add all cell records and the corresponding pointers to the incomplete doubly-connected edge list after the bounding box has been added. That is, fill in the details of line 8 of Algorithm VORONOIDIAGRAM.

7.10 Let P be a set of n points in the plane. Give an $O(n \log n)$ time algorithm to find two points in P that are closest together. Show that your algorithm is correct.

7.11 Let P be a set of n points in the plane. Give an $O(n \log n)$ time algorithm to find for each point p in P another point in P that is closest to it.

7.12 Let the Voronoi diagram of a point set P be stored in a doubly-connected edge list inside a bounding box. Give an algorithm to compute all points of P that lie on the boundary of the convex hull of P in time linear in the output size. Assume that your algorithm receives as its input a pointer to the record of some half-edge whose origin lies on the bounding box.



7.13 For each of the ten breakpoints shown in Figure 7.5, determine which of the five types it corresponds to.

7.14 Show that the farthest point Voronoi diagram on n points in the plane has at most $2n-3$ (bounded or unbounded) edges. Also give an exact bound on the maximum number of vertices in the farthest point Voronoi diagram.

7.15 Show that the smallest width annulus cannot be constructed with randomized incremental construction. To this end, show that a point p_i can be added to a set P_{i-1} that does not lie in the minimum width annulus, but does not lie on the boundary of the smallest width annulus of $P_i := P_{i-1} \cap \{p_i\}$.

7.16 Show that for some set P of n points, there can be $\Omega(n^2)$ intersections between the edges of the Voronoi diagram and the farthest site Voronoi diagram.

7.17 Show that if there are only $O(n)$ intersections between the edges of the Voronoi diagram and the farthest site Voronoi diagram, then the smallest width annulus can be computed in $O(n \log n)$ expected time.

7.18* In the Voronoi assignment model the goods or services that the consumers want to acquire have the same market price at every site. Suppose this is not the case, and that the price of the good at site p_i is w_i. The trading areas of the sites now correspond to the cells in the weighted Voronoi diagram of the sites (see Section 7.5), where site p_i has an additive weight w_i. Generalize the sweep line algorithm of Section 7.2 to this case.

7.19* Suppose that we are given a subdivision of the plane into n convex regions. We suspect that this subdivision is a Voronoi diagram, but we do not know the sites. Develop an algorithm that finds a set of n point sites whose Voronoi diagram is exactly the given subdivision, if such a set exists.