Chapter 6
Voronoi Diagrams

Let S be a finite point set in $\mathbb{R}^n$. Since S is compact, for every point $x \in \mathbb{R}^n$ there exists a closest point in S (which is not necessarily unique) with respect to the Euclidean norm $\|\cdot\|$. The set of all points in $\mathbb{R}^n$ that have a fixed point $s \in S$ as their nearest "neighbor" is a polyhedron. This mapping induces a decomposition of $\mathbb{R}^n$ into polyhedral "regions", the Voronoi diagram of S. Numerous applications of computational geometry begin with the computation of a Voronoi diagram.

We will first study the geometry of single Voronoi regions. To be able to discuss the arrangement of all Voronoi regions, we will introduce the general concept of a polyhedral complex. The main result of this chapter is the relationship between Voronoi diagrams and the convex hull problem from the previous chapter. We conclude the chapter by discussing an algorithm for the computation of Voronoi diagrams in the plane and its application to the post-office problem from the introduction.

6.1 Voronoi Regions

In this chapter, $S \subseteq \mathbb{R}^n$ always denotes a finite point set in $\mathbb{R}^n$ and $\|\cdot\|$ is the Euclidean norm. The Euclidean distance between two points $x, y \in \mathbb{R}^n$ is denoted by

$$\operatorname{dist}(x, y) := \|x - y\| = \sqrt{\langle x - y, x - y\rangle}.$$

For each point $s \in S$ we define the *Voronoi region*

$$\operatorname{VR}_S(s) := \left\{x \in \mathbb{R}^n : \operatorname{dist}(x, s) \leq \operatorname{dist}(x, q) \text{ for all } q \in S\right\}$$

as the set of points in $\mathbb{R}^n$ for which s is the nearest point from S. In this case, s is called a *nearest neighbor* (with respect to S).

Example 6.1 We study the case where $S = \{s, t\} \subseteq \mathbb{R}^n$ consists of exactly two distinct points. The set

$$h(s, t) := \left\{x \in \mathbb{R}^n : \operatorname{dist}(x, s) = \operatorname{dist}(x, t)\right\} = \operatorname{VR}_{\{s,t\}}(s) \cap \operatorname{VR}_{\{s,t\}}(t)$$

M. Joswig, T. Theobald, *Polyhedral and Algebraic Methods in Computational Geometry*, Universitext, DOI 10.1007/978-1-4471-4817-3_6,

consisting of those points which have both s and t as a nearest neighbor is an affine hyperplane: We have

$$\begin{aligned}\langle x-s, x-s\rangle - \langle x-t, x-t\rangle &= \sum_{i=1}^{n}(x_i-s_i)^2 - \sum_{i=1}^{n}(x_i-t_i)^2 \\ &= \sum_{i=1}^{n} 2(t_i-s_i)x_i + \sum_{i=1}^{n}\left(s_i^2-t_i^2\right),\end{aligned}$$

which implies that x is contained in $h(s,t)$ if and only if

$$\left(\sum_{i=1}^{n}\left(s_i^2-t_i^2\right), 2(t_1-s_1), \ldots, 2(t_n-s_n)\right)(1, x_1, \ldots, x_n)^T = 0. \tag{6.1}$$

In other words, the set $h(s,t) = \mathrm{VR}_{\{s,t\}}(s) \cap \mathrm{VR}_{\{s,t\}}(t)$ is precisely the affine hyperplane in $\mathbb{R}^n$ which has the homogeneous coordinates

$$\left[\sum_{i=1}^{n}\left(s_i^2-t_i^2\right) : 2(t_1-s_1) : \cdots : 2(t_n-s_n)\right]. \tag{6.2}$$

The Voronoi regions of s and t are the affine half-spaces which are defined by this hyperplane. We always define the orientation of $h(s,t)$ as in (6.2). Thus, we have $\mathrm{VR}_{\{s,t\}}(s) = h(s,t)^-$ and $\mathrm{VR}_{\{s,t\}}(t) = h(s,t)^+$. The vectors $s-t$ and $t-s$ are normal to the hyperplane $h(s,t)$ which (weakly) separates the two Voronoi regions.

The above observations about Voronoi regions of a two-element point set lead to the following statement.

Proposition 6.2 *Let $S \subseteq \mathbb{R}^n$ be finite. For $s \in S$ we have*

$$\mathrm{VR}_S(s) = \bigcap_{t \in S\setminus\{s\}} \mathrm{VR}_{\{s,t\}}(s) = \bigcap_{t \in S\setminus\{s\}} h(s,t)^-.$$

In particular, each Voronoi region is a (not necessarily bounded) polyhedron with at most $|S|-1$ facets.

Exercise 6.3 Give conditions which imply that all Voronoi regions are pointed polyhedra.

Exercise 6.4 Show that a point $s \in S$ lies on the boundary of the convex hull $\operatorname{conv} S$ if and only if its Voronoi region $\mathrm{VR}_S(s)$ is unbounded.

6.2 Polyhedral Complexes

We know from the previous section that the Voronoi regions of a finite point set in $\mathbb{R}^n$ are polyhedra. By construction, it is clear that these polyhedra cover the whole space $\mathbb{R}^n$. However, this alone does not reveal all of the important structural properties of Voronoi regions.

Definition 6.5 A *polyhedral complex* $\mathcal{C}$ is a finite set of polyhedra in $\mathbb{R}^n$ which satisfies the following conditions.

(a) $\emptyset \in \mathcal{C}$;
(b) If $P \in \mathcal{C}$, then all faces of P are also contained in $\mathcal{C}$;
(c) The intersection $P \cap Q$ of two polyhedra $P, Q \in \mathcal{C}$ is a (possibly empty) face of P and of Q.

The third condition is sometimes called the *intersection condition*. The elements of $\mathcal{C}$ are called *faces* and the *dimension* of $\mathcal{C}$ is the highest dimension of a face of $\mathcal{C}$. A polyhedral complex whose faces are polytopes is called a *polytopal complex*. A *simplicial complex* is a polytopal complex whose faces are simplices.

For a polyhedral complex $\mathcal{C}$ in $\mathbb{R}^n$ let

$$|\mathcal{C}| := \bigcup_{F \in \mathcal{C}} F \subseteq \mathbb{R}^n$$

be the *set covered by* $\mathcal{C}$. A *polyhedral* (respectively *polytopal* or *simplicial*) *decomposition* of a set $M \subseteq \mathbb{R}^n$ is a polyhedral (respectively polytopal or simplicial) complex $\mathcal{C}$ such that $|\mathcal{C}| = M$. A simplicial decomposition is also called a *triangulation*.

Example 6.6 Let $P \subseteq \mathbb{R}^n$ be an n-polyhedron. Then the face lattice $\mathcal{F}(P)$ is an n-dimensional polyhedral complex. The set of all proper faces defines an $(n-1)$-dimensional polyhedral complex that covers the boundary ∂P. This second complex is called the *boundary complex* of P.

The faces of a polyhedral complex $\mathcal{C}$ are partially ordered by inclusion; this is the *face poset* of $\mathcal{C}$. This notion agrees with the face lattice of a polytope if we view that polytope as a trivial polytopal complex as in the previous example.

Let $\mathcal{V}(S)$ be the set of all Voronoi regions of a finite set $S \subseteq \mathbb{R}^n$.

Theorem 6.7 *The set $\mathcal{V}(S)$ satisfies the intersection condition.*

Proof Let $s, t \in S$ be two distinct points. We can assume that the intersection

$$F := \mathrm{VR}_S(s) \cap \mathrm{VR}_S(t)$$

is non-empty. Proposition 6.2 states that $\mathrm{VR}_S(s) \subseteq h(s,t)^-$ and that $\mathrm{VR}_S(t) \subseteq h(s,t)^+$. This implies that $F \subseteq h(s,t)^- \cap h(s,t)^+ = h(s,t)$. Since we assumed

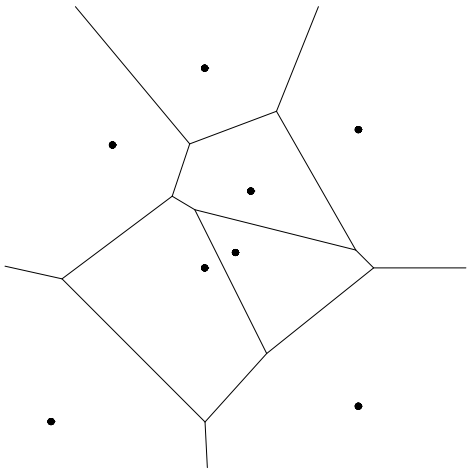

Fig. 6.1 The Voronoi diagram of a point set in the plane

$F \neq \emptyset$, we know that $h(s,t)$ is a supporting hyperplane of $\mathrm{VR}_S(s)$ and also of $\mathrm{VR}_S(t)$. Thus, $F = \mathrm{VR}_S(s) \cap \mathrm{VR}_S(t) = \mathrm{VR}_S(s) \cap \mathrm{VR}_S(t) \cap h(s,t)$ is a non-empty face of both Voronoi regions. □

Every non-empty finite set $\mathcal{C}$ of polyhedra in $\mathbb{R}^n$ that satisfies the intersection condition *generates* a polyhedral complex

$$[\mathcal{C}] := \{F : F \text{ is the face of a polyhedron in } \mathcal{C}\}.$$

The previous theorem motivates the following definition.

Definition 6.8 The polyhedral complex

$$\mathrm{VD}(S) := \left[\{\mathrm{VR}_S(s) : s \in S\}\right]$$

is called the *Voronoi diagram* of a finite set $S \subseteq \mathbb{R}^n$.

The faces of a Voronoi diagram are called *Voronoi cells*. The Voronoi regions are the maximal Voronoi cells (with respect to inclusion or dimension). Figure 6.1 depicts an example of a Voronoi diagram of a point set in the plane.

Remark 6.9 The definition of f-vectors can be extended to arbitrary polyhedral complexes.

6.3 Voronoi Diagrams and Convex Hulls

As we will see in the following chapters, Voronoi diagrams play a key role in several applications. Many interesting algorithms, e.g., the curve reconstruction algorithm

`NN-Crust` from Chapter 11 below, have the computation of a Voronoi diagram as their very first step. This motivates the questions of how a Voronoi diagram should be computed and what a suitable data structure would be for Voronoi diagrams.

A first observation is that convex hull algorithms are useful for the computation of Voronoi diagrams: Every region is given as a polyhedron in the $\mathcal{H}$-description. For m given points in $\mathbb{R}^n$ we obtain, by computing m dual convex hulls in $\mathbb{R}^n$, a $\mathcal{V}$-description of all Voronoi regions. Regardless of the efficiency of this method, the main disadvantage of it is that it does not directly provide a description of the relative position of the different Voronoi regions to one another. The main result of this chapter is the statement that a Voronoi diagram in $\mathbb{R}^n$ is a projection of an unbounded polyhedron in $\mathbb{R}^{n+1}$. Specifically, this reduces the construction of a Voronoi diagram to a single convex hull problem in $\mathbb{R}^{n+1}$.

To clarify the notation, we will embed $\mathbb{R}^n$ in $\mathbb{R}^{n+1}$ by adding the coordinate x_{n+1}. In particular, we will sometimes denote a point in $\mathbb{R}^{n+1}$ by (x, x_{n+1}) for $x \in \mathbb{R}^n$ and $x_{n+1} \in \mathbb{R}$.

Let

$$U := \left\{ x \in \mathbb{R}^{n+1} : x_{n+1} = x_1^2 + x_2^2 + \cdots + x_n^2 \right\} \tag{6.3}$$

be the *standard paraboloid* in $\mathbb{R}^{n+1}$. For a point $p \in \mathbb{R}^n$ let $T(p)$ denote the tangent hyperplane to the paraboloid U at $p_U := (p, \|p\|^2)$.

Lemma 6.10 *For every point $p \in \mathbb{R}^n$ we have*

$$T(p) = \left[-\|p\|^2 : 2p_1 : \cdots : 2p_n : -1 \right].$$

Proof We know from calculus that the tangent hyperplane to the graph of a differentiable function $u : \mathbb{R}^n \to \mathbb{R}$ at a point $(p, u(p))$ can be described by the linear equation

$$x_{n+1} = u(p) + \left\langle u'(p), x - p \right\rangle$$

(see, e.g., [73]). In our case, we have $u(p) = p_1^2 + p_2^2 + \cdots + p_n^2 = \|p\|^2$, and thus the gradient satisfies $u'(p) = (2p_1, \ldots, 2p_n) = 2p$. Substituting yields

$$x_{n+1} = \|p\|^2 + \langle 2p, x - p \rangle = -\|p\|^2 + 2\langle p, x \rangle,$$

and thus we obtain the desired representation of the tangent hyperplane in homogeneous coordinates. □

In the following, we imagine that the x_{n+1}-direction of the coordinate system points vertically upwards.

Lemma 6.11 *Let $p, x \in \mathbb{R}^n$ and $x_U = (x, \|x\|^2)$ be the point lying above x on U. Then x_U lies above $T(p)$, i.e., in the affine half-space $T(p)^+$ with respect to the homogeneous coordinates from Lemma 6.10. The vertical distance from x_U to $T(p)$ is $\|x - p\|^2$.*

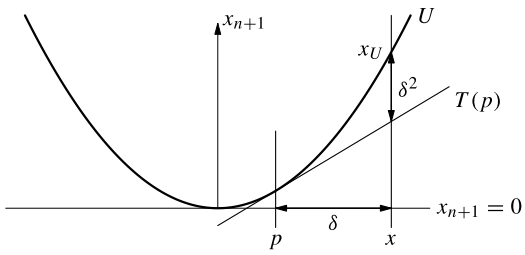

Fig. 6.2 The distance computation for $n = 1$. Due to the rotation invariance of U, the 2-dimensional figure suggests the proper intuition for higher dimensions. Here, we have $\delta = \|x - p\|$

Proof The x_{n+1}-coordinate of x_U is $\sum_{i=1}^{n} x_i^2$, and by Lemma 6.10 the x_{n+1}-coordinate of the point on the hyperplane $T(p)$ above x is

$$2p_1x_1 + \cdots + 2p_nx_n - p_1^2 - \cdots - p_n^2.$$

The distance from x_U to $T(p)$ is $(x_1 - p_1)^2 + \cdots + (x_n - p_n)^2 = \|x - p\|^2$. Figure 6.2 illustrates this computation. □

Let S be an m-element subset of $\mathbb{R}^n$. For a point $s \in S$ we have that $T(s)^+$ is the affine half-space above the tangent hyperplane at U.

Due to the monotonicity of the function $\delta \mapsto \delta^2$ on the positive half-line, we can interpret Proposition 6.2 using Lemma 6.11 in the following way: A point $x \in \mathbb{R}^n$ is contained in the Voronoi region $\mathrm{VR}_S(s)$ if and only if for all $T(t)$, where $t \in S$, the hyperplane $T(s)$ is the one that has the smallest vertical distance from the point x_U. This implies the following statement; see Fig. 6.3.

Theorem 6.12 *The Voronoi diagram of S is the orthogonal projection of the boundary complex of the polyhedron $\mathcal{P}(S) := \bigcap_{s \in S} T(s)^+$ to the hyperplane $x_{n+1} = 0$.*

Corollary 6.13 *The total number of cells of a Voronoi diagram of an m-element point set in $\mathbb{R}^n$ is of order $O(m^{\lceil n/2 \rceil})$.*

Proof The total number of cells of a Voronoi diagram can be bounded by the maximal number of faces of an $\mathcal{H}$-polyhedron with m facets in $\mathbb{R}^{n+1}$. The dual version of the asymptotic Upper-bound Theorem, Theorem 3.46, therefore implies that the total number of faces is of order $O(m^{\lceil n/2 \rceil})$, since $\lfloor (n+1)/2 \rfloor = \lceil n/2 \rceil$. □

Theorem 6.12 specifically states that the space is partitioned by the relative interior of the cells of $\mathrm{VD}(S)$. For an arbitrary point $x \in \mathbb{R}^n$ let

$$\mathbb{B}_S(x) := \left\{ y \in \mathbb{R}^n : \mathrm{dist}(x, y) < \mathrm{dist}(x, s) \text{ for all } s \in S \right\} \tag{6.4}$$

be the largest open ball with center x which does not contain a point of S. Furthermore, let

$$S(x) := \partial \mathbb{B}_S(x) \cap S.$$

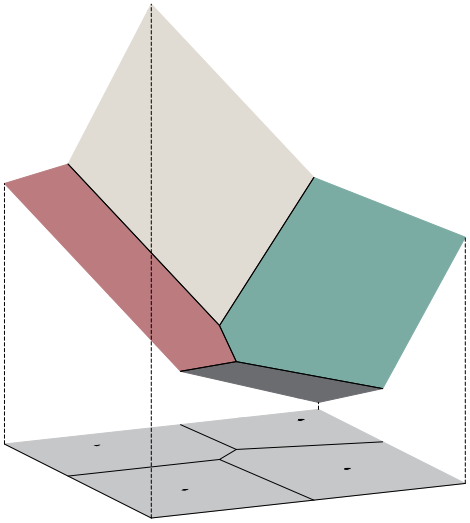

Fig. 6.3 A Voronoi diagram obtained by an orthogonal projection

Theorem 6.14 *The uniquely determined relatively open cell of* $\mathrm{VD}(S)$ *that contains a given point* $x \in \mathbb{R}^n$ *has dimension* $n - \dim \operatorname{aff} S(x)$.

Proof The point x is contained in a relatively open k-cell C of $\mathrm{VD}(S)$ if and only if there exists a series of facets $F_1, \ldots, F_{n-k+1}$ of the polyhedron $\bigcap_{s \in S} T(s)^+$ for which:

$$G_1 \supsetneq G_2 \supsetneq \cdots \supsetneq G_{n-k+1} =: G, \tag{6.5}$$

where $G_i := F_1 \cap \cdots \cap F_i$ and C is the orthogonal projection of G to $\mathbb{R}^n$; see Exercise 3.59. The decreasing chain condition in (6.5) is satisfied for the facets $F_1, \ldots, F_{n-k+1}$ if and only if $G = F_1 \cap \cdots \cap F_{n-k+1}$ is non-empty and the normals of the facets are linearly independent.

By Lemma 6.10 we have that $(2s_1, \ldots, 2s_n, -1)^T$ is a normal vector to the facet $T(s)$ for $s \in S$. Therefore, the normal vectors to the facets corresponding to a subset $S' \subseteq S$ are linearly independent if and only if the points of S' are affinely independent. Altogether, this proves the statement. □

In the next section we will focus on the planar case $n = 2$. Therefore, we are interested in the following special cases of Theorem 6.14.

Corollary 6.15 *Let* $S \subseteq \mathbb{R}^2$ *be finite.*

(a) *A point* $x \in \mathbb{R}^2$ *is a vertex of the Voronoi diagram* $\mathrm{VD}(S)$ *if and only if* $S(x)$ *contains at least three points.*
(b) *A point* $x \in \mathbb{R}^2$ *lies in the relative interior of an edge of* $\mathrm{VD}(S)$ *if and only if* $S(x)$ *consists of exactly two points.*

For a vertex x of the Voronoi diagram $\mathrm{VD}(S)$ we call the ball $\mathbb{B}_S(x)$ from (6.4) the *Voronoi disk* around x. The boundary $\partial \mathbb{B}_S(x)$ is called the *Voronoi circle*.

Exercise 6.16 Show that if every $(n+2)$-element subset of $S \subseteq \mathbb{R}^n$ does not lie on a common $(n-1)$-sphere, then the lifted polyhedron is simple and therefore every Voronoi region is simple.

If this condition is satisfied, we say that the points in S are in *general position*. Note that we defined "general position" slightly differently in Chapter 3 and in Section 5.3; the term is always dependent on the context.

6.4 The Beach Line Algorithm

As in the computation of convex hulls in Section 5.3, there exist special algorithms for the computation of Voronoi diagrams in the planar case. We introduce here an algorithm due to Fortune [42]. First, we discuss the geometric idea, and then approach the question of determining its complexity. In this particular case, the complexity depends significantly on the data structures employed. With respect to this property, this algorithm is an exception within this text.

Fortune's beach line algorithm is a so-called *sweep line method*. The idea is to construct the Voronoi diagram of a finite point set $S \subseteq \mathbb{R}^2$ step-by-step. Here, we can imagine the vertical axis as a time-scale that is traversed from top to bottom. In this interpretation, at a certain time τ only a part of the Voronoi diagram has been revealed by the algorithm. For a point s from the input set S we then have that s is known at time τ if $s_2 \geq \tau$. The horizontal line $H_\tau = [-\tau : 0 : 1]$ is the sweep line for time τ and the affine half-space $[-\tau : 0 : 1]^+$ contains the previously detected points from S. The next natural question is which part of the Voronoi diagram is actually known at time τ.

The set of points in $\mathbb{R}^2$ that have the same distance from a point p and a (non-incident) line G is a parabola, which we denote here by $\mathrm{Par}(p, G)$ (see Exercise 6.18 below). For every point $s \in S$ with $s_2 > \tau$ which is known at time τ, all points which are closer to s than to any possible unknown point of S lie above the parabola $\mathrm{Par}(s, H_t)$. The term "above" makes sense here since the symmetry axis of $\mathrm{Par}(s, H_\tau)$ is parallel to the vertical axis. The time τ is called *generic* if $H_\tau \cap S = \emptyset$. If we denote the points on or above the parabola by $\mathrm{Par}(s, H_\tau)^+$, then, according to our notation for affine half-spaces, we get the following lemma.

Lemma 6.17 *The part of the Voronoi diagram which is known at time τ is contained in the set*

$$\bigcup_{s \in S} \mathrm{Par}(s, H_\tau)^+$$

for each generic time $\tau \in \mathbb{R}$.

If τ is generic, the set $\bigcup_{s \in S} \mathrm{Par}(s, H_\tau)^+$ is homeomorphic to an affine half-space. Its boundary B_τ is a union of parabolic arcs that resembles the appearance of waves approaching a beach; see Fig. 6.4. This is the reason why the boundary

curve is called the "beach line", and this term gives the algorithm its name. Note that each vertical line intersects the *beach line* B_τ in exactly one point; this property is inherited from the individual parabolas.

Exercise 6.18 Determine a parametrization of the parabola $\mathrm{Par}(s, H_\tau)$ for a given s and $\tau \in \mathbb{R}$. That is, search for $a, b, c \in \mathbb{R}$ such that

$$\mathrm{Par}(s, H_\tau) = \left\{ \begin{pmatrix} x \\ ax^2 + bx + c \end{pmatrix} : x \in \mathbb{R} \right\},$$

subject to the condition that $s_2 > \tau$.

A point $s \in S$ with the property that $\mathrm{Par}(s, H_\tau)$ is part of the beach line is said to be *active* at time τ.

Now we briefly discuss what happens at a non-generic time τ. For sufficiently small $\epsilon > 0$ we have that $\tau - \epsilon$ is a generic time. The smaller ϵ is, the steeper the parabola $\mathrm{Par}(s, H_{\tau-\epsilon})$ will be. This is rigorously formulated in the following exercise.

Exercise 6.19 Let $s = (s_1, s_2)^T \in S$ be a point with $\tau = s_2$. Show that

$$\lim_{\epsilon \to 0^+} \mathrm{Par}(s, H_{\tau-\epsilon}) = \left\{ \begin{pmatrix} s_1 \\ \sigma \end{pmatrix} \in \mathbb{R}^2 : \sigma \geq s_2 \right\}.$$

Here, we mean convergence with respect to the Hausdorff metric. How is it possible to use this to define the beach line for non-generic times? [*Hint*: Look at Snapshot 2 in Fig. 6.4.]

Lemma 6.20 *If τ is generic, then each parabolic arc in $\mathrm{Par}(s, H_\tau) \cap B_\tau$, for $s \in S$, is contained in the corresponding Voronoi region $\mathrm{VR}_S(s)$.*

The set $\mathrm{Par}(s, H_\tau) \cap B_\tau$ may consist of several parabolic arcs, e.g., snapshot 2 in Fig. 6.4. Here the parabolic arc for b is divided as soon as the point d becomes known, i.e., at time d_2.

Proof For $x \in \mathrm{Par}(s, H_\tau) \cap B_\tau$ let $\delta := \mathrm{dist}(x, s) = \mathrm{dist}(x, H_\tau)$ and assume $x \notin \mathrm{VR}_S(s)$. By Corollary 6.15 the open disk B around x with radius δ contains a point $r \in S$. Since $B \subseteq H_\tau^+$, we have that r is known at time τ. But x is above the parabola $\mathrm{Par}(r, H_\tau)$, which contradicts x being contained in the beach line B_τ. □

The next question is to determine how the beach line changes as the time τ changes (in the direction of smaller values). Here, of course, the relevant times are those when a certain point $s = (s_1, s_2)^T \in S$ is first detected; see Snapshot 2 in Fig. 6.4. This time s_2 will be called a *point event*. It is a consequence of Lemma 6.20, and of the convexity of the Voronoi regions, that new parabolic arcs can only arise at point events; the beach line cannot be pierced from behind by a parabola. For a

generic τ we have that, by construction, the beach line has only finitely many points where it is not differentiable, since it is the union of finitely many parabolic arcs; these points are called *breakpoints*.

Lemma 6.21 *If τ is generic then every breakpoint of B_τ lies on an edge of the Voronoi diagram.*

Proof Let x be a breakpoint of the beach line B_τ at time τ. Then there exist two active points $r, s \in S$ with $x \in \mathrm{Par}(r, H_\tau) \cap \mathrm{Par}(s, H_\tau)$ and the statement follows from Lemma 6.20. □

We assume that the vertical line $[-s_1 : 1 : 0]$ through s intersects the beach line B_{s_2} at a point x which is contained in a unique parabolic arc $\mathrm{Par}(r, H_{s_2})$; here $r \in S$ is an active point. By construction we have that $x \in \mathrm{VR}_S(r) \cap \mathrm{VR}_S(s)$ and $\mathrm{VR}_S(r) \cap \mathrm{VR}_S(s)$ is an edge of the Voronoi diagram; this edge is detected (partly) for the first time at time s_2. For a sufficiently small $\epsilon > 0$, a part of the parabola $\mathrm{Par}(s, H_{s_2-\epsilon})$ lies on the beach line, say with the breakpoints x and y. Then, the segment $[x, y]$ is the intersection of the Voronoi edge $\mathrm{VR}_S(r) \cap \mathrm{VR}_S(s)$ and the set above the beach line. Thus, new edges are discovered at point events.

By Corollary 6.15, every vertex v of $\mathrm{VR}(S)$ lies on a circle through at least three points of S. The point in time at which a circle through at least three points from S is detected is called a *circle event*. In other words, we have a circle event at time τ if the sweep line H_τ is the lower tangent to a circle through at least three points of S. By Corollary 6.15 only those circle events create vertices whose circular disks have no points of S in their interior.

Now we can examine how a parabolic arc γ vanishes from the beach line. Let γ' and γ'' be, respectively, the left and the right neighbor of γ in the beach line. Let $s, s', s'' \in S$ be the points corresponding to these three parabolic arcs. We assume now that the parabolic arc γ vanishes at time τ. At the slightly later generic point in time $\tau - \epsilon$, γ' and γ'' are neighbors in the beach line. Hence, by Lemma 6.21, we know that the Voronoi regions $\mathrm{VR}_S(s')$ and $\mathrm{VR}_S(s'')$ are neighbors in $\mathrm{VD}(S)$. At time τ, γ contracts to a point v. By construction, we have that $\delta := \mathrm{dist}(v, s) = \mathrm{dist}(v, s') = \mathrm{dist}(v, s'')$ and that v is a Voronoi vertex. Also, the distance between v and the sweep line is δ at time τ. This means that τ is a circle event for the triple of points (s, s', s''). This is illustrated in Snapshots 4 and 7 in Fig. 6.4.

Exercise 6.22 Show that there are at most $2|S| - 2$ breakpoints in the beach line B_τ for a generic time τ.

Data Structures The way in which geometric data is stored is crucial for the runtime analysis of the beach line algorithm. Here we only outline the most important ideas and refer the reader to the original work of Fortune [42], and to the books [31] and [71], for more details of the implementation.

First, we have to decide in which way we want to store the output, i.e., the Voronoi diagram of a point set in the plane. One special feature of the planar case is

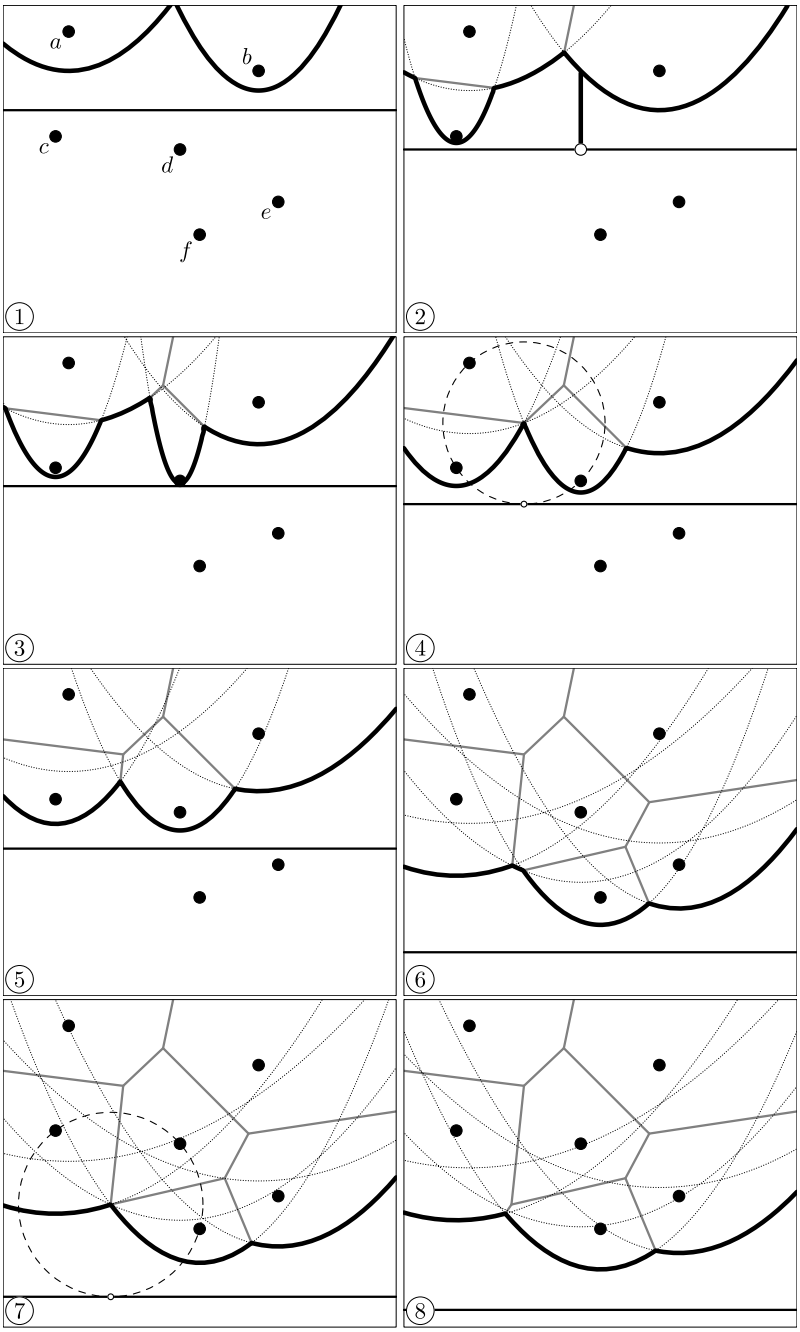

Fig. 6.4 Eight snapshots of the beach line algorithm

that we can restrict ourselves to the Voronoi edges. Every Voronoi region is a (not necessarily bounded) polygon whose edges can be cyclically ordered. Every edge is contained in exactly two regions. Therefore, if we store each edge twice with its vertices and orientation, then the regions are implicitly given by the sequence of their edges. Thus, each oriented edge stands for an incident pair of a Voronoi edge and a Voronoi region, and the Voronoi vertices are implicitly given as the endpoints of the edges.

Depending on the specific construction of the data structure, it can be problematic that some Voronoi regions are unbounded, and thus the cyclic sequence of edges does not form a complete circle. However, this can be easily addressed by using the ideal points of lines on which the unbounded edges lie as artificial Voronoi vertices. These ideal points can then be connected by artificial Voronoi edges on the ideal line so that every Voronoi region can be represented as a closed circle of (original or artificial) Voronoi edges.

In practical applications, it is common to use points on a sufficiently large bounding box, rather than artificial Voronoi vertices on the ideal line. This bounding box should be large enough to contain all points of S and all vertices of $\mathrm{VD}(S)$.

The data structure itself is then a *doubly linked list* of oriented edges, which are also called *half-edges*, such that each edge is stored with its two endpoints and with a reference to the next half-edge in the cyclic order. Furthermore, we store a reference to the parallel half-edge, i.e., the same edge with the opposite orientation. This data structure is also known as the *half-edge data structure*. We refer to [27, §10.2] for the implementation of doubly linked lists.

Note also that the half-edge data structure is useful for storing arbitrary planar graphs and arbitrary cell decompositions of oriented surfaces.

Before we study the beach line algorithm in detail, we have to determine a suitable way in which to code the beach line itself. Here, it is not necessary to trace the exact trajectory of each parabolic arc. We need only store the combinatorial information, i.e., the number of parabolic arcs in the beach line, the points of S to which they correspond, and the order in which they occur.

Example 6.23 The beach line from Fig. 6.5 can be coded, for example, by the ordered sequence of points $(s^{(1)}, s^{(2)}, s^{(3)}, s^{(4)}, s^{(5)})$ and the breakpoints correspond to neighboring pairs of points.

Some points can occur multiple times. For example, we see that the beach line in Snapshot 2 of Fig. 6.4, which appears shortly after a point event, can be written as (a, c, a, b, d, b).

However, coding the beach line as an ordered list is not beneficial for the runtime complexity. It is better to use a *binary search tree*. The leaves of this search tree contain points from S that each correspond to one parabolic arc on the beach line. An interior vertex stands for a breakpoint (r, s) if r is the biggest leaf in the left subtree, and s the smallest in the right subtree; see Fig. 6.5. In particular, we have that here, in contrast to the list description, the breakpoints are explicitly represented.

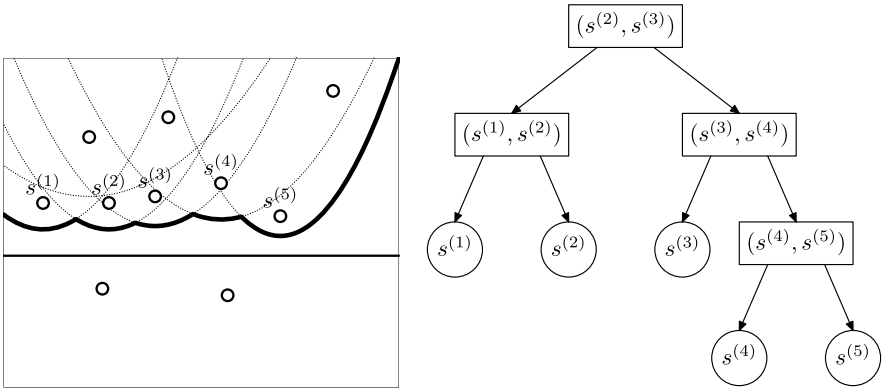

Fig. 6.5 A beach line consisting of five parabolic arcs and a representation as a search tree

For further details on the implementation of the search tree representation of the beach line, we refer to the book [31]. General binary search trees are described in [27, §12].

The search tree structure of the beach line is not sufficient to guarantee a good run-time of the algorithm. We also need the height of the search tree to be of size $O(\log m)$, where $m = |S|$, at every step of the algorithm. A search tree with this property is said to be *balanced*. Note that the coding length of the beach line, i.e., the number of parabolic arcs and breakpoints, is linear in m; see Exercise 6.22. Hence, it is possible to add or delete parabolic arcs in $O(\log m)$ time.

Various concepts are associated with the balance of search trees, for example, the so-called "red black trees" [27, §13].

Lastly, we need to establish a data structure for the point and circle events. An important aspect here is the (time-wise) order, which would suggest a list, or a search tree as a suitable representation. However, here it is crucial to immediately see the next event at every step, without having to perform a search. Therefore, a search tree is not suitable. It is also important to be able to quickly add new events at the right position in the sorted order. Thus, a list is also not suitable. The solution is a *heap*, which allows us to immediately see the next event (i.e., in constant time), and to delete this event after processing it in logarithmic time. Furthermore, we need to be able to guarantee that arbitrary new events can be added in $O(\log m)$ time. An example of a suitable data structure is a *binomial heap* [27, §19].

6.4.1 The Algorithm

Using the data structures described above, we can now detail the actual algorithm. Let $\mathcal{B}$ be a balanced search tree that represents the beach line. The queue Q stores unprocessed events, which are listed in order of their appearance. Every event in the queue Q is represented by point coordinates. The sweep line is only implicitly represented by the next event at a given time.

Algorithm 6.1: The beach line algorithm

Input: Finite point set $S \subseteq \mathbb{R}^2$
Output: VD(S) in half-edge model

```
1 B ← ∅
2 Initialize Q with all point events from S.
3 while Q ≠ ∅ do
4     e ← next event in Q; remove e from Q
5     if e point event for s ∈ S then
6         Handle-Point-Event(s, Q, B)
7     else
8         Handle-Circle-Event(e, Q, B)
```

The order defined by the queue, and thus the heap structure, corresponds precisely to the ordering of the events by their y-coordinate. Since the sweep line is moving from top to bottom, points with a large y-coordinate represent early events. Here, a point event corresponding to $s \in S$ is coded by the point s itself. A circle event is represented by the lowest point of the circle; when the sweep line reaches the lowest point of the circle, the whole circular disk is visible.

For a correct implementation it is crucial that the events in Q are not stored in an isolated way. It is necessary to be able to distinguish between point and circle events. Moreover, it is also useful that a point that represents a circle event also refers to the points of S that define the circle. There are a few additional references of this kind between the data structures $\mathcal{B}$ and Q, but we restrict ourselves to the presentation of the crucial ideas. We mainly ignore the processing of the actual Voronoi diagram in the half-edge model in our pseudo-code. This has the consequence that Algorithm 6.1 lists VD(S) as output, but we never state a return value in the code.

For our analysis, we first assume that the points in S are in general position, i.e., at most three on one circle at a time. The case where this condition is not satisfied is discussed at the end of this section.

Before we discuss the two subroutines to process point and circle events on p. 95, we will estimate the complexity of the steps of the main program. Initializing the heap Q has time complexity $O(m \log m)$ (this can be reduced to $O(m)$ when a suitable implementation is used), since there are exactly m point events. Estimating the number of possible circle events is more difficult, since there may be circle events that do not lead to Voronoi vertices. An analysis of Steps 10 to 12 of the subroutine `Handle-Point-Event` shows that every Voronoi edge can trigger at most two (potential) circle events. Therefore, by Corollary 6.13, there are at most $O(m)$ events in total; Steps 3 to 8 in Algorithm 6.1 are hence performed at most $O(m)$ times. If Q is realized as a binomial heap, it takes $O(\log m)$ time to delete an event from Q. We will show below that each point and each circle event only requires logarithmic time. This implies that the total time complexity of the beach line algorithm is $O(m \log m)$.

Every parabolic arc γ is implicitly coded in the search tree $\mathcal{B}$ as a triple $[(r,s),s,(s,t)]$, where $r,s,t \in S$ are as in Fig. 6.5. The pairs of points (r,s) and (s,t) represent the breakpoints that bound the parabolic arc. In particular, we have that the parabolic arc on the left side of γ corresponds to r and the one on the right corresponds to t.

When checking the correctness of this subroutine, note that each circle event is matched to the lowest point of the corresponding Voronoi circle. Therefore, the point events that correspond to points on a Voronoi circle, i.e., that trigger a circle event, are always correctly processed at a time prior to the circle event. This is also true for the special case where the third point of S on a Voronoi circle is simultaneously the lowest point, i.e., when a circle event and the corresponding point event occur at the same time. In this case, the following occurs: the parabolic arc corresponding to the lowest point is generated and immediately afterwards deleted by the simultaneously occurring circle event. In particular, Algorithm 6.1 always begins with at least three point events before the first circle event can occur.

Simultaneously occurring point events can be processed in arbitrary order. The same is true for simultaneously occurring circle events, since we assumed the points to be in general position. Thus, two circle events may occur at the same time, but at different places. Simultaneous point and circle events that are unrelated do not pose a problem. The only critical case, i.e., when a circle event is triggered by a simultaneous point event, was discussed above.

Step 3 in `Handle-Circle-Event` can be seen as the reverse of Step 8 in `Handle-Point-Event`. There, only those parabolic arcs are deleted which were previously generated by a point event. Step 11 of `Handle-Point-Event` can also trigger redundant circle events, but these are detected and deleted in Step 4 of `Handle-Circle-Event`.

1 **Procedure:** `Handle-Point-Event`$(s, Q, \mathcal{B})$
2 **if** $\mathcal{B} = \emptyset$ **then**
3 Add s to $\mathcal{B}$.
4 **else**
5 Let $\gamma = ((p,q),q,(q,r))$ be the parabolic arc in $\mathcal{B}$ above s.
6 **if** γ *refers to a circle event in* Q **then**
7 delete this event
8 Replace γ in $\mathcal{B}$ by the three parabolic arcs
$$\big[(p,q),q,(q,s)\big],\ \big[(q,s),s,(s,q)\big],\ \big[(s,q),q,(q,r)\big].$$
9 Generate a pair of new half-edges for the Voronoi edge $\mathrm{VR}(q) \cap \mathrm{VR}(s)$.
10 Compute the intersection point $v = (v_1, v_2)^T$ of the Voronoi edge corresponding to the parabolic arc $\gamma := ((q,s),s,(s,q))$ and the Voronoi edge corresponding to the parabolic arc on the left.
11 Add $(v_1, v_2 - \mathrm{dist}(v,s))$ as a potential circle event e to Q.
12 The parabolic arc γ contains a reference to e and vice-versa.
13 Proceed analogously to Steps 10 to 12 with the parabolic arc on the right side of γ.

Procedure: `Handle-Circle-Event`$(e, Q, \mathcal{B})$

Let γ be a parabolic arc that vanishes at the circle event e.
Remove γ from $\mathcal{B}$ and update the neighboring inner vertices.
Remove all circle events from Q which are referred to by γ or by one of its two neighbors.
Generate the center z of the circle corresponding to e as a new Voronoi vertex.
Generate a pair of new half-edges for the new breakpoint that emerges due to the removal of γ.
Store z as an endpoint of the two involved edges.
Link the edges to one another with respect to the half-edge model.

Example 6.24 We want to show how the point event illustrated in Snapshot 2 in Fig. 6.4 affects the event queue Q. Before the point event corresponding to the point d is processed, the queue contains three point events and one circle event:

$$Q = \big(d, (a, b, c), e, f\big).$$

The point event d triggers two new circle events. After this, at the generic time $\tau = d_2 - \epsilon$, we have:

$$Q = \big((a, b, d), (a, c, d), e, f\big).$$

Later, the two circle events (a, b, d) and (a, c, d) will generate Voronoi vertices. The circle event (a, b, c) vanishes at time d_2 (`Handle-Point-Event`, Step 7), since we then know that d is contained in the circumcircle of a, b and c.

It remains to be discussed what occurs when the points in S are not in general position. It is perhaps surprising that our algorithm works here with only a few modifications. Actually, we have that the beach line algorithm produces a valid Voronoi diagram that may contain some edges of length 0. It is simple to detect and delete these edges in linear time after the algorithm has terminated.

6.5 Determining the Nearest Neighbor

We now discuss the problem of finding the nearest neighbor, or the nearest post office respectively, which we mentioned in the introduction. Given a finite point set $S \subseteq \mathbb{R}^2$ and a point $p \in \mathbb{R}^2$, we want to determine the point $s \in S$ which minimizes $\text{dist}(p, s)$. This problem has, of course, a very simple solution, i.e., we can compare each distance from p to every point of S. If S consists of m points, this method needs $O(m)$ steps.

But, when the configuration of the point set S is always the same and only the point p changes with each call, a different approach may be better. If we expect

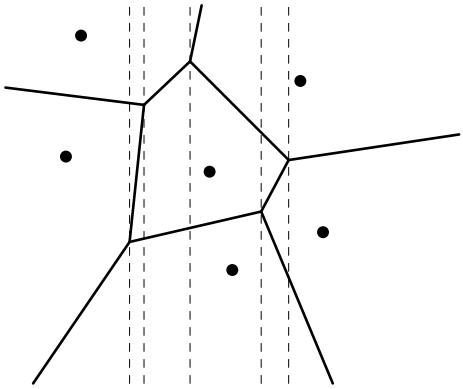

Fig. 6.6 Vertical layers in the Voronoi diagram for answering the nearest neighbor problem

many calls, it pays off to invest more time in the beginning to be able to process each later call more quickly. In the following, let m be the cardinality of S.

Our goal is to describe a data structure that enables the answer of each call in logarithmic time. To do this, we compute the Voronoi diagram of S using Fortune's beach line algorithm in $O(m \log m)$ steps.

Then, we draw a vertical line through each Voronoi vertex as depicted in Fig. 6.6. These additional lines divide the Voronoi diagram into triangles and trapezoids, and into unbounded polyhedra in the outer regions. These vertical layers are ordered from left to right. If these are stored in a balanced search tree, we can detect the layer of each point $p \in \mathbb{R}^2$ via its first coordinate p_1 in $O(\log m)$ time.

By construction we can guarantee that no vertical layer contains a vertex in its interior, so that all Voronoi edges are vertically ordered within each layer. If we also store the edges in each layer in a balanced search tree, we can detect the pair of edges that lies directly above and below p in $O(\log m)$ steps using the second coordinate p_2.

Theorem 6.25 *For an m-element point set $S \subseteq \mathbb{R}^2$ it is possible to generate a data structure in $O(m^2 \log m)$ time such that the solution to the nearest neighbor problem in S can be found in $O(\log m)$ time.*

Proof It is possible to compute the Voronoi diagram $\mathrm{VD}(S)$ in $O(m \log m)$ time. Since there exist linearly many Voronoi vertices, there exist linearly many vertical layers. In each layer there are at most linearly many edges. In total, we have to initialize $O(m)$ balanced search trees each with $O(m)$ vertices. □

6.6 Exercises

Exercise 6.26 Let S be the vertex set of the n-dimensional cross-polytope. Determine the f-vector of the Voronoi diagram $\mathrm{VD}(S)$.

Exercise 6.27 Let $e^{(1)}, \ldots, e^{(n)}$ denote the standard basis vectors of $\mathbb{R}^n$. The vertices of the standard cube $[0,1]^n$ are precisely the sums of pairwise distinct standard basis vectors. Show that the $n!$ simplices

$$\Delta(\sigma) := \operatorname{conv}\left\{0, e^{(\sigma(1))}, e^{(\sigma(1))} + e^{(\sigma(2))}, \ldots, e^{(\sigma(1))} + e^{(\sigma(2))} + \cdots + e^{(\sigma(n))}\right\}$$

generate a triangulation of $[0,1]^n$, where σ runs through all elements of the symmetric group $\mathrm{Sym}\{1, \ldots, n\}$. Show that every simplex $\Delta(\sigma)$ has the same volume (i.e., $1/n!$).

Exercise 6.28 Let $m \in \mathbb{N}$ be arbitrary. Describe an m-element point set in $\mathbb{R}^2$ (in general position) for which the beach line algorithm first treats all point events and then all circle events.

6.7 Remarks

Voronoi diagrams have appeared independently over the last few centuries in different scientific disciplines. Their methodical usage in mathematics can be traced back to Dirichlet (1850) and Voronoi (1908), who used the diagrams to study quadratic forms. The presentation of a Voronoi diagram can be found as early as in Descartes' (1644) work on visualizing the mass distribution in our solar system.

Detailed discussions of this topic can be found in the books of Edelsbrunner [38], Boissonat and Yvinec [15] and de Berg et al. [31].

`polymake` computations with Voronoi diagrams will be explained in Section 7.6 below. `CGAL` offers a variety of methods to compute Voronoi diagrams and their generalizations, including the beach line algorithm.