Lecture 7
Voronoi Diagrams

We introduce a critical idea of EGC, the technique of root bounds. This will be addressed through the study of Fortune's algorithm for computing the Voronoi diagram of a set of points. The algorithms is based on the plane sweep paradigm. But unlike the algorithms we have seen until now, this one has an interesting twist – it requires the computation of square roots. Thus the direct use of a Big Rational package will not necessarily give us the geometric exactness we need. This leads us into the theory of constructive root bounds.

§1. Introduction

Consider a finite set $S \subseteq \mathbb{R}^2$. We define a certain skeleton (comprising a set of straightline segments) called the **Voronoi diagram** of S. See Figure 1 for an example of such a diagram where $|S| = 11$. The most striking feature of the diagram is the fact that it subdivides the plane into a set of $|S|$ polygonal regions, with each region containing a unique point s of S.

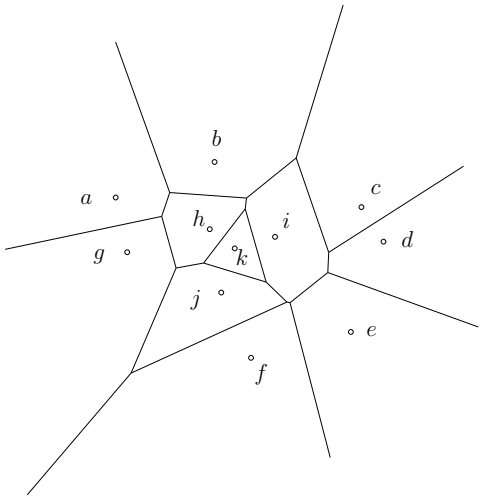

Figure 1: Voronoi diagram of a set $\{a, b, c, \ldots, j, k\}$ of 11 points

We introduce some usual terminology. The points in S is called **sites** because in some applications, S can be viewed as locations on a map. For instance, each site $s \in S$ may be the location of a postoffice, and the polygonal region containing s comprises those points q in the plane that are closest to this postoffice. Then a basic computational problem is to construct a data structure $D(S)$ that can quickly answer, for any query point $q \in \mathbb{R}^2$, the post office $s \in S$ that is closest to q. This is the well-known **Post Office Problem**.

The regions defined by the Voronoi diagram can be precisely discribed as follows: for $p, q \in \mathbb{R}^2$, let $d(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$ denote Euclidean distance between them. If $p \neq q$, let $b(p, q)$ denote the **bisector line** between p and q: this is the line that passes through the point $(p + q)/2$ and is normal to the segment $[p, q]$. Also, let $h(p, q)$ denote the open half-space bounded by $b(p, q)$ and containing p. When $p = q$, it is convenient to define $h(p, q)$ to be the entire plane. So $h(p, q) \neq h(q, p)$ unless $p = q$.

Although we have used $\mathbb{R}^2$ for illustration, these concepts are easily extended to any dimension. Fix a non-empty finite set $S \subseteq \mathbb{R}^n$. Define the the **Voronoi region of** s to be

$$V(s) = V_S(s) := \bigcap \{h(s, s') : s' \in S\}.$$

Since $V(s)$ is the intersection of half-spaces, it is is a convex polygonal set. It is non-empty since $s \in V(s)$. The corners of the boundary of $V_S(s)$ are called **Voronoi vertices** and the maximal open line segments bounding $V_S(s)$ are called **Voronoi edges**. Note that a Voronoi edge can be a finite segment, a half-line or a line. Let $K_0(S)$, $K_1(S)$ and $K_2(S)$ denote the set of Voronoi vertices, Voronoi edges and Voronoi regions, respectively.

Let us analyze the Voronoi facets (i.e., vertices, edges, regions) which we just defined. For any point p, let $d(p, S) = \min\{d(p, q) : q \in S\}$ and $C_S(p)$ denote the circle centered at p with radius $d(p, S)$. We call $C_S(p)$ the **largest empty circle** at p because the interior of $C_S(p)$ has empty intersection with S. On the other hand, the set $C_S(p) \cap S$ is non-empty, and the cardinality $|C_S(p) \cap S|$ yields very important information: we classify the point p according to this cardinality.

Lemma 1. *Suppose $p \in \mathbb{R}^2$.*

- *If $C_S(p) \cap S = \{s\}$ then p lies in the Voronoi cell $V_S(s)$.*
- *If $C_S(p) \cap S = \{s, s'\}$ where $s \neq s'$ then p lies on a Voronoi edge. In fact, $p \in b(s, s')$.*
- *If $|C_S(p) \cap S| \geq 3$ then p is a Voronoi vertex. In this case, p is the circumcenter of any three distinct sites $s, s', s'' \in C_S(p) \cap S$.*

We leave the proof as an Exercise.

> **Cell Complex.** Voronoi diagrams are best viewed as a special kind of cell complex. A **d-cell** is any subset of $\mathbb{R}^n$ that is homeomorphic to the open d-ball $B^d = \{p \in \mathbb{R}^d : \|p\| < 1\}$ ($d \geq 1$); a **0-cell** is just a singleton set. Also, the closure of B^d, the closed d-ball, is denoted $\overline{B}^d$. Let K be any non-empty finite collection of pairwise disjoint cells. The **support** of K is $|K| := \cup K \subseteq \mathbb{R}^n$. We call K a **cell complex** if, for each d-cell $C \in K$, $d \geq 1$, there is a continuous function $f_C : \overline{B}^d \to |K|$ such that the restriction of f_C to B^d is a homeomorphism between B^d and C. It can be shown [6, p. 215] that this implies that $f_C(\dot{B}^d)$ is equal to a union of cells in K.

A cell complex K in which every 2-facet is a convex is called a convex subdivision.

Theorem 2. *The union $K(S) = K_0(S) \cup K_1(S) \cup K_2(S)$ is a convex subdivision of the plane.*

Proof. By the previous lemma, p lies in a Voronoi region or a Voronoi edge or a Voronoi vertex depending on whether $|C_S(p) \cap S|$ is 1, 2 or ≥ 3. These conditions are mutually disjoint. Hence K is pairwise disjoint.

By definition, the boundary of each Voronoi region is a union of Voronoi edges and Voronoi vertices. Hence K is a complex.

Since each region is a convex, we conclude that K is a convex subdivision. **Q.E.D.**

In view of this theorem, $K(S)$ is called the **Voronoi complex** of S. Each $f \in K(S)$ is a **Voronoi facet**. The skeleton $K_0(S) \cup K_1(S)$ of this complex is called the **Voronoi diagram** and denoted $Vor(S)$. The Voronoi diagram can also refer to the support of this skeleton; this abuse of language should not cause confusion.

Lemma 3. *The following are equivalent for $|S| \geq 3$:*
(i) The Voronoi diagram of S is connected
(ii) S does not lie on a line.
(iii) Every Voronoi region has a connected boundary.

Proof. (iii) implies (ii): If S lies on a line, then the Voronoi diagram has no vertices and comprises $|S| - 1$ parallel lines.

(ii) implies (iii): A Voronoi region $V(s)$ is star-shaped with respect to s. If $V(s)$ is bounded, then clearly its boundary is connected. So let $V(s)$ be unbounded. Since S does not lie on a line, there exists $s', s'' \in S$ such that (s, s', s'') that forms a non-degenerate triangle. Then $h(s, s') \cap h(s, s'')$ forms a wedge W which contains $V(s)$. Being star-shaped and unbounded, there is a non-empty maximal cone $C \subseteq V(s)$ of rays emanating from s. Thus $C \subseteq W$. [It is easy to argue that the union of set of rays from s must form a

connected set.] It follows that if we shoot any ray from s in a direction outside this cone, the ray will hit a boundary of $V(s)$. Thus the boundary of $V(s)$ is connected.

(iii) implies (i): Let p, q be two points of the Voronoi diagram. Consider the straightline from p to q. It passes through an alternating sequence of Voronoi regions and points on the Voronoi diagram. Let this alternating sequence be $p_0, R_1, p_1, R_2, \ldots, R_k, p_k$ where $p = p_0$ and $q = p_k$. Since each R_i has connected boundary, it means p_{i-1} is connected to p_i. Thus $p = p_0$ is connected to $q = p_k$.

(i) implies (iii): If a Voronoi region $V(s)$ does not have a connected boundary, then it must be bounded by two parallel lines. These two lines determine two connected components of the Voronoi diagram of S. Thus the Voronoi diagram is not connected. **Q.E.D.**

Theorem 4. *Let $|S| = n$. The number of Voronoi vertices is at most $2n - 5$ and the number of Voronoi edges is at most $3n - 6$.*

For each $n \geq 3$, these bounds can be achieved simultaneously by a single configuration.

Proof. If S lies on a line, the result is easy to see. Otherwise, we know that the skeleton is connected. Let v be the number of Voronoi vertices, e the number of Voronoi edges.

Let C be a circle large enough to contain all the Voronoi vertices and sites. We create a new vertex v_∞ outside C, and redirect each infinite Voronoi edge so that, outside of C, they each connect to v_∞ along pairwise disjoint curvilinear paths. See Figure 2.

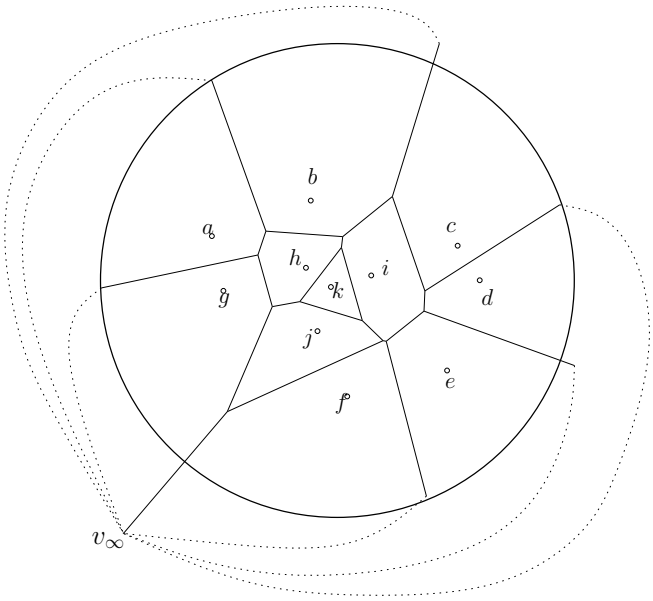

Figure 2: Graph with new vertex v_∞

This creates a connected graph G whose complement defines a set of connected regions. These regions has a one-one correspondence with the Voronoi regions. We now use the Euler formula for a planar graph embedded in the plane,

$$\nu_0 - \nu_1 + \nu_2 = 1 + \beta$$

where ν_i is the number of cells of dimension i and β the number of connected components. Since there are n Voronoi regions, this means $\nu_2(G) = n$. Similarly, we have $\nu_0(G) = v + 1$ (because of the vertex v_∞) and $\nu_1(G) = e$ (the new curves we added are considered part of the infinite edges which we modified). Finally, since G is connected, $\beta(G) = 1$. Thus, the Euler formula becomes

$$(v + 1) - e + n = 2.$$

We need another relation between v and e to get bounds on v and e. Let us count the number I of (vertex, edge) incidences. Counting from the viewpoint of edges, $I = 2e$ since each edge contributes two incidences.

Counting from the viewpoint of vertices, $I \geq 3(v+1)$ since each vertex yields at least three incidences. Thus $2e \geq 3v+3$. Plugging this into the Euler formula, we can either eliminate e or v. The result are the bounds on v and e claimed in this theorem.

We now show that the bounds of the theorem is tight. When $n = 3$, we let S comprise the vertices of an equilateral triangle. Then $v = 1$ and $e = 3$ achieve the bounds of the theorem. Now we can incrementally add successive sites to S so that each addition creates increases the number of Voronoi vertices by 2 and increases the number of Voronoi edges by 3.

For instance, for $n = 4$, we can place the next vertex right at the location of the unique Voronoi vertex (replacing it with 3 new Voronoi vertices), and adding three new Voronoi edges. Alternatively, the next three sites ($n = 4, 5, 6$) can be placed so that these points form the vertices of regular hexagon. In either case, the process can be continued indefinitely. **Q.E.D.**

§2. Fortune's Linesweep

Fix a set S of sites. Again consider a horizontal line $H(t)$ that sweeps from $t = -\infty$ to $t = \infty$. The key idea is to define a curve called the **beach line** $B(t)$ of $H(t)$: the set $B(t)$ comprises those points p such that

$$d(p, H(t)) = d(p, S(t))$$

where $S(t)$ denotes those points that has been swept by time t, i.e., points lying on or below $H(t)$. This is illustrated in Figure 3.

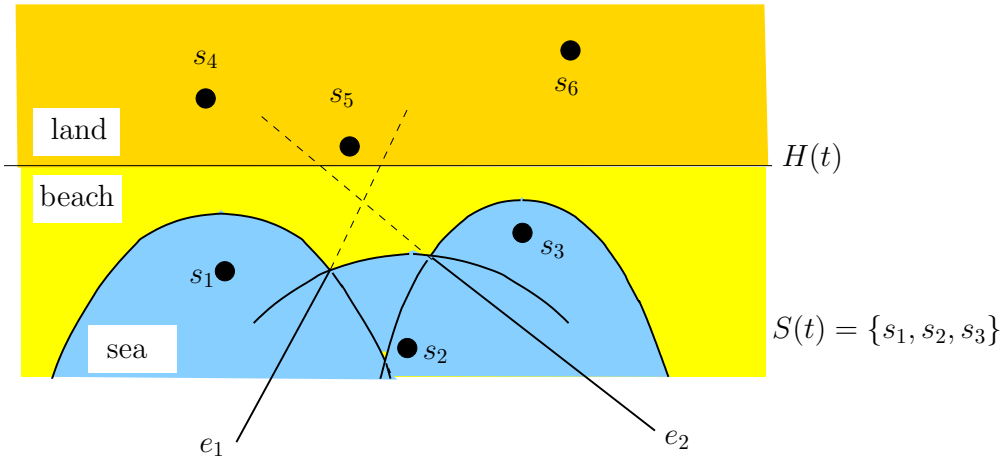

Figure 3: Beach line defined by active sites s_1, s_2, s_3

The next lemma will give a characterization $B(t)$. For each site $s \in S(t)$, define $B_s(t)$ to be the parabola defined by the line $H(t)$ as directrix and s as focus. In Figure 3, the parabolas defined by s_1, s_2, s_3 are indicated.

Lemma 5. *The beach line $B(t)$ is the upper envelop of the set $\{B_s(t) : s \in S\}$ of parabolas.*

Proof. By a fundamental property of parabolas, $B_s(t)$ is comprised of those points that are equidistant from $H(t)$ and s, i.e., $d(p, H(t)) = d(p, s)$. The upper envelop of these parabolas are therefore those points p which achieve $d(p, H(t)) = d(p, S(t))$. **Q.E.D.**

Thus the beach line is a union of a number of parabolic arcs. Each arc is associated with a site $s \in S$; by definition, such sites are said to be **active**. Active sites may become **inactive** when they no longer define arcs on the beach line. Points on the beach line where two consecutive arcs meet are called **breakpoints**. This beach line divides the plane into an upper part called **land** and a lower part called **sea**. The land itself is divided into two parts: the **beach** which lies between $H(t)$ and the beach line, and the **dry land** which lies above $H(t)$. The sites are thus classified as **wet** or **dry** depending on whether they are on dry land or

in the sea; there are no sites on the beach. Thus, active sites lies in the sea. These concepts are illustrated in Figure 3. As we sweep, we maintain the part of the Voronoi diagram *restricted to the sea.* This **partial Voronoi diagram** need not be connected even if the overall Voronoi diagram is connected. E.g., the partial diagram in Figure 3 is comprised of the two disconnected edges e_1 and e_2.

We next examine some basic properties of the beach line. Our main goal is to understand how the beach line transforms as $t \to \infty$. We are interested in the **critical moments** when $B(t)$ change combinatorially.

P1 *The beach line is x-monotone, and its vertical projection covers the entire x-axis.* This follows from the basic properties of parabolas.

As corollary, we can represent $B(t)$ as a concatenation of parabolic arcs,

$$B(t) = (\alpha_1; \alpha_2; \cdots; \alpha_m) \tag{1}$$

listed in order of increasing x. Suppose α_i is part of the parabola defined by $s_i \in S$. Combinatorially, the beach line can be represented by the sequence

$$(s_1, s_2, \ldots, s_m) \tag{2}$$

of active sites. It is important to realize that a site s may appear more than once in this sequence. However, for each pair s, s' of distinct sites, these sites can be adjacent in (2) at most twice (once in the order s, s' and once in the reverse order s', s). That is because the parabolas defined by s and s' can intersect at most twice. We shall next see how this active sequence can change.

P2 *The sea is monotonically advancing with t.* To see this, let L be any vertical line. Then the intersection $B(t) \cap L$ moves monotonically upward with time.

P3 *Every point p in the sea is closer to some wet site s than to any dry site s'.* In proof, let the line segment $[p, s']$ intersect the beachline at some point p'. Note that p' may not be unique. There is an active site s that is closest to p'. Then

$$d(p, s') = d(p, p') + d(p', s') > d(p, p') + d(p', H(t)) = d(p, p') + d(p', s) \geq d(p, s).$$

P4 *Each break point q moves along a Voronoi edge (hence in straight lines).* In proof, let q be the common endpoint of the arcs generated by the active sites s and s'. To show that q lies on a Voronoi edge determined by s and s' we show that $d(q, s) = d(q, s') \leq d(q, s'')$ for any other site s''. If s'' is on the dry land, this follows from P4. If s'' is in the sea, consider the intersection q' between the vertical line L through q and parabola generated by the sweepline $H(t)$ and s''. Note that q' lies below q, since $B(t)$ is the upper envelop. Then $d(q', s'') = d(q', H(t)) > d(q, H(t)) = d(q, s')$. This proves s'' is not closer than s' to q.

P5 *Each point q on an arc of the beach line belongs to the Voronoi region of the cell that generates the arc.* By a similar argument to the one for P4.

P6 Consider three consecutive arcs $\alpha, \alpha', \alpha''$ separated by two consecutive breakpoints p and q. Let s, s', s'' be the sites that generates these arcs. If α, α'' are part of the same parabola (i.e., $s = s''$), then p, q will be diverging with t. Hence α' cannot disappear. This property is clear from the geometry.

P7 *When an arc α' contracts, the first time when it becomes a point v, there are three distinct sites, s, s', s'' such that v is the Voronoi vertex defined by s, s', s''.* This is called the **circle event** parametrized by (s, s', s''). This terminology will become clearer below. In proof, by P6, we know that $s \neq s''$. By P4, we know that the instant α' disappears, it is simultaneously closest to s, s', s''.

P8 *When a new site first becomes wet, it is active and it defines a degenerate arc that is just a line segment (or a ray, in case this is the first site).* This is called the **site event** parametrized by s.

P9 *The beach line can change combinatorially only with a site event or a circle event.* Pf: By P7 and P8, we know that these events can cause combinatorial change in the beach line. We must exclude all other possibilities of creating new arcs in the beach line. Suppose at time t, a new arc appears at position p on some arc define by an active site s. This new arc must come from some wet site s'. Let L be the vertical line through p. Let the parabola $B_s(t)$ intersects L at the point $p(t)$, and similarly, let $p'(t) = B_{s'}(t) \cap L$. At time t^-, $p(t^-)$ is above $p'(t^-)$ but at time t^+, $p(t^-)$ is below $p'(t^-)$. This is impossible (details omitted).

P10 *A site is initially dry, then active (thus wet). Finally it becomes inactive (still wet).* Pf: in other words, inactive sites do not become active again. After it becomes inactive, its Voronoi region is now contained in the sea. It will remain this way since the sea is advancing monotonically.

P11 A parabola can contribute up to n arcs on the beach line. Such examples are easy to generate.

P12 *The beach line has at most* $2n - 1$ *arcs.* Pf: this is because the only way that new arcs appear is through a site event, and there are n such events. Each such event increase the number of arcs by 2 (except the first site event increases the number by 1).

¶1. The Algorithm. As usual, we sweep the horizontal line $H(t)$ from $t = -\infty$ to $t = +\infty$. Call t a **critical moment** if there is a site event or a circle event. If t is non-critical, then the beach line $B(t)$ is a sequence of arcs as in (1). The combinatorial part of this information (i.e., (2)) will be represented by a balanced binary tree T.

We need to detect the successive critical moments using a priority queue Q. Initially, Q stores all the site events. Circle events are added to Q as they arise (see below). Using Q, we can correctly update T.

Finally, we must represent the known portion $Vor_t(S)$ of the Voronoi diagram: this is just the part of $Vor(S)$ that lies below $B(t)$ which we called the partial Voronoi diagram. We will use a half-edge data structure D.

We next consider these three data structures, T, Q, D in detail.

¶2. The Priority Queue Q. We have two kinds of events stored in Q: either a site event s with priority $s.y = s_y$, or a circle event (s, s', s'') with priority equal to the y-coordinate of the highest point of the circumcircle of s, s', s''. Initially, Q contains all the site events. We must discuss how to add circle events into Q. If (2) is the active sequence of sites, then we maintain the invariant that every consecutive triple (s_{i-1}, s_i, s_{i+1}) that can generate a circle event is already in Q. Hence, whenever T is modified we need to check to detect any new circle event that is formed, and to insert it into Q. If the priority of the triple is greater than the current time t, it is inserted into Q. A basic property to show is this: *Every circle event of priority t will placed in Q before time t.*

¶3. The Balanced Binary Tree T. If the arcs and active sites in $B(t)$ are represented as in (1) and (2), then T will have m nodes, where each node stores site. But what is the key value for searching the tree T? Clearly, the x-coordinates of these sites are inapproriate because a site may occur more than once in the sequence. Rather, we use the x-coordinates of the breakpoint defined by any two consecutive sites, s_i, s_{i+1}. This breakpoint must be defined as a function of t: let $b_t(s_i, s_{i+1})$ denote this x-coordinate at time t. For brevity, we may also write $x_i(t)$ for $b_t(s_i, s_{i+1})$. Also, let $x_0(t) = -\infty$ and $x_m(t) = +\infty$. Note that in general, two sites s, s' determine up to two breakpoints since two parabolas with the same directrix intersect in one or two points. If $s.y = s'.y$, then there is a unique break point. Otherwise, we distinguish these two breakpoints by writing $b_t(s, s')$ and $b_t(s', s)$ such that

$$b_t(s, s') < b_t(s', s)$$

provided $s.y < s'.y$. Below, we show how to compute $b_t(s, s')$.

An interesting issue arises in application of binary search trees. In general, there are two ways to store a set of items in a binary search tree. We can either store them exclusively in the leaves, or we can store them

in the internal nodes as well well as the leaves. We call[1] the former **exogenous** and the latter **endogenous** trees. Since keys $x_i(t)$ depends on two consecutive sites, it seems more natural to store the sites only in the leaves. We assume that T is a full-binary tree (i.e., each node has two children) and each node u of the tree stores a pointer to its successor $u.next$ and predecessor and $u.prev$. If u is an internal node, then $u.next$ and $u.prev$ are both leaves. If u is a leaf, then we see that $u.next^2 = u.next.next$ is the next leaf, and $u.prev^2 = u.prev.prev$ is the previous leaf. The leaves therefore stores the sequence $(s_1, \ldots, s_m)$ of active sites.

Note that maintaining all these pointers are straightforward and standard. Moreover, if T is a balanced tree (e.g., an AVL Tree), there will be rotation operations to maintain balance. Such pointers are unaffected by rotations.

We now consider insertion into and deletions from T. These are determined by the site and circle events:

- Site Event s: This amounts to insert a new site s into T using $s.x$ as key and using $t = s.y$ as the current time. Let u be initially the root of T. If u is not a leaf, we compute the breakpoint $b_t(u.prev, u.next)$. If $s.x \geq b_t(u.prev, u.next)$, we set u to the right child of u, else u is set to the left child of u. We continue this process until u is a leaf. Assume that u contains the site s_i. It follows that that

$$x_i(t) \leq s.x < x_{i+1}(t).$$

  Let us first assume the non-degenerate situation where $x_i(t) \neq s.x$. The sequence (2) is thereby transformed to

$$(s_1, \ldots, s_i, s, s_i, \ldots, s_m)$$

  In terms of modifying the tree T, we create a subtree of u with three new leaves. We store s in the middle leaf and store separate copies of s_i in the other leaves. This is illustrated in Figure 4.

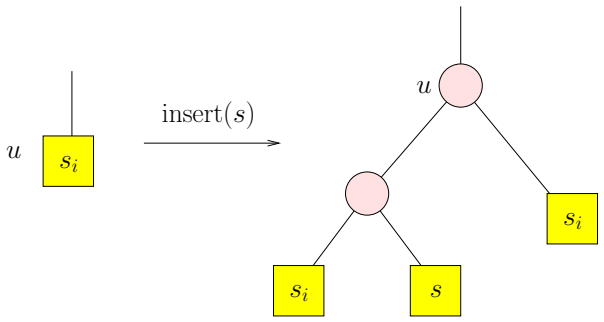

Figure 4: Inserting s into leaf u containing s_i

  We must next check if the new consecutive triples (s_{i-1}, s_i, s) and (s, s_i, s_{i+1}) define circle events in the future (time $> t$). If so, these events are inserted into Q. If $x_i(t) = s.x$, then we only need to create two new leaves as children of u, one storing s_i and the other storing s. We leave the details to the reader.

- Circle Event (s_{i-1}, s_i, s_{i+1}): this amounts to deleting the event s_i from T. Note that $s_{i-1} \neq s_{i+1}$ in this case. Therefore, we obtain a new break point defined by $b_t(s_{i-1}, s_{i+1})$. We must also check if $(s_{i-2}, s_{i-1}, s_{i+1})$ and $(s_{i-1}, s_{i+1}, s_{i+2})$ define circle events in the future (at time $> t$). If so, we insert them into Q.

[1] Our definition of exogenous/endogenous is slightly different than the literature. We think of each item as a pair, item=(data,key). Clearly, each internal node of a binary search tree must store a key. The question is where is the data stored? In the literature, endogenous tree is when each tree node directly stores the data associated with the key; it is exogenous when we only store a pointer to the data.

¶4. The Half-Edge Datastructure D. Note that the partial Voronoi diagram $K_t(S)$ at time t may not be connected even if $K(S)$ is connected. This is seen each time we initiate a site event, at which time an isolated partial edge appears, growing in both directions. To make the partial complex connected, we include the beach line in its representation. Then we can represent the sea as a cell complex whose boundaries are either edges in $K_t(S)$ or arcs from the beach line $B(t)$. Although $K_t(S)$ and $B(t)$ changes continuously with t, the combinatorial structure is unchanged except at an event. One classic representation of such a complex is the half-edge data structure. This structure is illustrated in Figure 5.

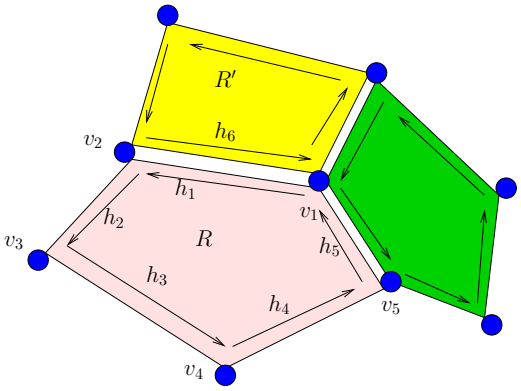

Figure 5: Half-edge data structure

A half-edge data structure D represents an planar graph embedded in the plane. It contains three lists, representing the sets of regions, half-edges and vertices (respectively). The unique feature of this representation is that it stores not "edges" but **half-edges**. Each half edge h is regarded as a directed edge from a start vertex $h.start$ to a stop vertex $h.stop$. E.g., in Figure 5, the half edge h_1 is a directed edge from v_1 to v_2.

Moreover, h has a **twin** h' which is another half-edge, directed from $h.stop$ to $h.start$. Thus,

$$h.start = h.twin.stop, \quad h.stop = h.twin.start.$$

In Figure 5, the twin of half edge h_1 is h_6. We can think of the pair $\{h, h'\} = \{h, h.twin\}$ as an edge e. Why this redundancy? The reason is that each edge e is incident on two regions R and R'. We think of the two two half-edges that represent e as represening the (e, R) and (e, R') incidences. Thus, for each half edge h, we store a field $h.region$ with value R if h represents the incidence between the underlying edge with R. Moreover, we orient the half-edges that are incident on a given region R in a counter-clockwise manner. More precisely, each half-edge h has a field $h.next$ which is another half edge with the property that

$$h.region = h.next.region, \quad h.stop = h.next.start.$$

In this way, by repeated application of the $.next$ field, we can go around the boundary of the $.region$. In Figure 5, the region R has 5 incident half-edges $h_1, \ldots, h_5$, and we have $h_{i-1}.next = h_i$ for $i = 1, \ldots, 5$ (and $h_0 = h_5$).

Each vertex stores its (x, y) coordinate, and also one of the half-edges that points away from the vertex. In some applications, we do not even need the list of regions and may omit the field $h.region$. However, for our Current application, we can store with each region the unique site the determines the corresponding Voronoi region. The boundary edges of regions need not be straightline segments (in this sense, Figure 5 is somewhat misleading). Indeed, for our regions, these edges might be (temporarily) parabolic arcs. With arcs, it makes sense to have regions that have only two bounding half-edges. Indeed, when we insert a new region into D with each site event, the region will initially have only two bounding edges. The reader will find it instructive to carry out the details of creating a new region during a site event.

It is clear that this data structure D stores most of its the information in half-edges. Using D, for instance, we can list the set of regions that are clockwise around a given vertex in linear time. There is

also no need to store the inverse of *.next* because we can also use the available fields to get to the previous half-edge in constant time. We leave these as Exercises.

§3. Algebraic Details

In order to perform the arithmetic operations correctly according to the principles of EGC, we need to make comparisons of numbers accurately. In Fortune's algorithm we encounter irrational numbers for the first time. These numbers involve square roots. Let us investigate this in some detail.

Suppose p, q, r are three sites and c is their circumcenter. If (x, y) is a point on this circle, we have

$$(x - c_x)^2 + (y - c_y)^2 = \rho^2 \tag{3}$$

where ρ is the radius. Plugging $p = (p_x, p_y)$ into this equation, we get $(p_x - c_x)^2 + (p_y - c_y)^2 = \rho^2$, re-written as

$$2p_x c_x + 2p_y c_y + (\rho^2 - c^2) = p^2$$

where $c^2 = c_x^2 + c_y^2$ and $p^2 = p_x^2 + p_y^2$. We can similarly plug q and r into (3) to get two more equations. Writing these 3 equations in the matrix form,

$$\begin{bmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{bmatrix} \begin{bmatrix} 2c_x \\ 2c_y \\ \rho^2 - c^2 \end{bmatrix} = \begin{bmatrix} p^2 \\ q^2 \\ r^2 \end{bmatrix}$$

Rewriting and solving, we obtain

$$\begin{aligned} c_x &= \frac{1}{2\Delta} \begin{vmatrix} p^2 & p_y & 1 \\ q^2 & q_y & 1 \\ r^2 & r_y & 1 \end{vmatrix} \\ c_y &= \frac{1}{2\Delta} \begin{vmatrix} p_x & p^2 & 1 \\ q_x & q^2 & 1 \\ r_x & r^2 & 1 \end{vmatrix} \\ \Delta &= \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix}. \end{aligned} \tag{4}$$

The priority of the circle event (p, q, r) is therefore

$$c_y + \rho$$

where

$$\rho = \sqrt{(p_x - c_x)^2 + (p_y - c_y)^2}.$$

In the following, *we assume that the input points p, q, r have L-bit integer coordinates.* It follows from (4) that the center c has coordinates whose numerator and denominators are $3L + O(1)$ and $2L + O(1)$-bit integers (respectively). We can rewrite the priority as a number of the form

$$c_y + \rho = \frac{A + \sqrt{B}}{D} \tag{5}$$

where A, B, D are integers with $3L + O(1)$, $6L + O(1)$ and $2L + O(1)$ bits respectively. In fact, we can choose $D = \Delta$.

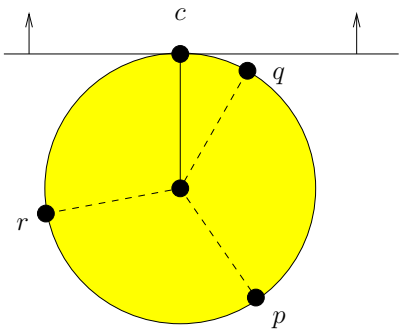

Figure 6: Circle event c generated by sites p, q, r

The λ-notation. If x is a rational number of the form p/q, and p, q are integers with $\lg|p| \leq m$ and $\lg|q| \leq m$, then we call x an $(m:n)$**-bit number**, and write

$$\lambda(x) = (m:n). \tag{6}$$

For instance, if x is an integer, then $\lambda(x) = (m:0)$ for any $m \geq \lg|x|$.
Like the standard big-O notation, the λ-notation is an *upper bound notation*. Thus the equality in (6) is not standard equality, but a form of "$\leq$". In particular, if $\lambda(x) = (m:n)$ holds, then $\lambda(x) = (m':n')$ holds for all $m' \geq n$ and $n' \geq n$. We can thus use big-O expressions in the righthand side of (6). E.g., $\lambda(x) = (O(L^2) : 2L + O(1))$. But it would not make sense to insert lower bound expressions into the righthand side. E.g., "$\lambda(x) = (\Omega(L) : 0)$" is nonsense.

For rough estimates, we can ignore the $O(1)$ terms and simply say that the coordinates of c are $(3L:2L)$-bit rational numbers. Nevertheless, in our implementation, we need to know explicit bounds on these $O(1)$ terms. Here are the explicit constants for the numbers derived in (4) and (5), collected here for future reference:

$$\begin{aligned}
\lambda(p_x) &= (L:0) && (7)\\
\lambda(p^2) &= (1+2L:0) && (8)\\
\lambda(\Delta) &= (4+2L:0) && (9)\\
\lambda(c_x) &= (4+3L:4+2L) && (10)\\
\lambda(D) &= (4+2L:0) && (11)\\
\lambda(A) &= (4+3L:0) && (12)\\
\lambda(B) &= (11+6L:0) && (13)
\end{aligned}$$

In our priority queue operations, we must compare the priorities of events. Priority of site events are L-bit integers while circle events have priority given by (5). Hence the most complicated comparisons are between the priorities of two circle events,

$$\frac{A+\sqrt{B}}{D} : \frac{A'+\sqrt{B'}}{D'} \tag{14}$$

where A, A' are $3L$ bits, etc. How can we make such comparisons exact?

¶5. Method of Repeated Squaring. We can use the well-known method of repeated squaring. This method seems quite reasonable for (14) if we only want to determine *equality or inequality*. But we need to

know more: in case of inequality, we must know which is bigger. We now demonstrate that in this case, the method of repeated squaring is considerably more involved to implement.

We may reduce the desired comparison to the simpler case of checking whether $\alpha + \beta\sqrt{n} \geq 0$ where $\alpha, \beta \in \mathbb{R}, n \in \mathbb{N}$. This is equivalent to the following three disjunctions:

$$(\alpha \geq 0, \beta \geq 0) \quad \vee \tag{15}$$
$$(\alpha \geq 0, \beta \leq 0, \alpha^2 - \beta^2 n \geq 0) \quad \vee \tag{16}$$
$$(\alpha \leq 0, \beta \geq 0, \alpha^2 - \beta^2 n \leq 0). \tag{17}$$

Recursively, if α, β are integer expressions involving other square-roots then we must further expand on each of the predicates $\alpha \geq 0$, $\beta \geq 0$, $\alpha^2 - \beta^2 n \geq 0$.

Let us apply this remark to the original comparison (14), but first rewriting it as $D'(A + \sqrt{B}) \geq D(A' + \sqrt{B'})$ or,

$$(I): \quad D'\sqrt{B} \geq D\sqrt{B'} - E$$

where $E = DA' - D'A$. First, let us only assume $D, D' \neq 0$. Then (I) is equivalent to the disjunction of the following three conjuncts:

$$(II): \quad (D' \leq 0), (D\sqrt{B'} - E \leq 0), (D'^2 B \leq (D\sqrt{B'} - E)^2).$$
$$(III): \quad (D' \geq 0), (D'^2 B \geq (D\sqrt{B'} - E)^2).$$
$$(IV): \quad (D' \geq 0), (D\sqrt{B'} - E \leq 0).$$

These can ultimately be expanded into a Boolean function of the sign of the following 6 expressions:

$$D, D', E, \tag{18}$$
$$D^2 B' - E^2, \tag{19}$$
$$D'^2 B - D^2 B' - E^2, \tag{20}$$
$$4D^2 E^2 B' - (D'^2 B - D^2 B' - E^2)^2. \tag{21}$$

Alternatively, we can expand $(II) \vee (III) \vee (IV)$ into a disjunction of 18 conjuncts involving these expressions.

Let us make a useful observation. The last expression represents a number with $20L + O(1)$ bits. Thus we conclude:

Lemma 6. *The method of repeated squaring, applied to the comparison of the priorities of two circle events can be reduced to the computation of integers that are at most* $20L + O(1)$ *bits long.*

In our application, it is reasonable to assume D, D' to be positive. Then the test (I) is equivalent to the disjunction $(III) \vee (IV)$. But we still have to evaluate a Boolean function of the sign of all the expressions, except D and D', in (18). Thus the $20L + O(1)$ bits bound of Lemma 6 cannot be improved. Below we will derive a true improvement.

¶6. Method of Root Bounds. Another way to do the comparison is to compute $\alpha = (A + \sqrt{B})/D$ and $\alpha' = (A' + \sqrt{B'})/D'$ to sufficient accuracy to distinguish them. Basically, we need a lower bound on $|\alpha - \alpha'|$. Consider the integer polynomials:

$$\begin{aligned} P(X) &= (D \cdot X - (A + \sqrt{B}))(D \cdot X - (A - \sqrt{B})) && (22)\\ &= D^2 X^2 - 2DA \cdot X + (A^2 - B), && (23)\\ P'(X) &= D'^2 X^2 - 2D'A' \cdot X + (A'^2 - B'). && (24) \end{aligned}$$

Then α and α' are roots of

$$\begin{aligned} Q(X) &= P(X)P'(X)\\ &= (DD')^2 X^4 - 2DD'(DA' + D'A)X^3\\ &\quad +(D^2(A'^2 - B') + D'^2(A^2 - B) + 4AA'DD')X^2\\ &\quad -2(AD(A'^2 - B') + A'D'(A^2 - B))X\\ &\quad +(A^2 - B)(A'^2 - B'). \end{aligned}$$

Note that coefficients of X^i has $(12-i)L + O(1)$ bits ($i = 0, \ldots, 4$). Thus $\lg \|Q\|_\infty = 12L + O(1)$. The root separation bound of Mahler (see Lecture 6 and also [12, Corollary 32, Chap. VI]) applied to the square-free polynomial $Q(X)$ of degree m says that

$$-\lg(|\alpha - \alpha'|) < (m-1)\lg(\|Q\|_2) + (m+2)(\lg m)/2.$$

Since $m = 4$,

$$-\lg(|\alpha - \alpha'|) = 36L + O(1).$$

We can improve this using another approach: form the resultant of $P(X)$ and $P'(X+Y)$ with respect to X,

$$R(Y) = Res_X(P(X), P'(X+Y)).$$

Note that $\alpha - \alpha'$ is a root of $R(Y)$. From [12, Lemma 35, Chap. VI], we conclude that

$$-\lg(|\alpha - \alpha'|) = 24L + O(1).$$

¶7. Repeated Squaring again, but for Root Bounds. We further improve our bound by using a trick[2] which superficially look like repeated squaring. Suppose we have an expression $\sum_{i=1}^n a_i\sqrt{b_i}$ whose sign we want to determine. Here $a_i \in \mathbb{Q}$ and $b_i \in \mathbb{N}$. We introduce a new variable ε via the equation $\varepsilon = \sum_{i=1}^n a_i\sqrt{b_i}$. By repeated squaring, we eventually obtain an integer polynomial equation,

$$P(\varepsilon) = 0.$$

Now we can apply the zero bound for ε.

Applying this idea to the critical comparison in Fortune's algorithm, we have

$$\varepsilon = \frac{A + \sqrt{B}}{D} - \frac{A' + \sqrt{B'}}{D'} \tag{25}$$

By repeated squaring, we finally obtain the polynomial equation

$$0 = P(\varepsilon) = \sum_{i=0}^m b_i \varepsilon^i.$$

How large are the coefficients b_i? From the discussion of repeated squaring leading to Lemma 6, we conclude that each b_i has at most $20L + O(1)$ bits. Using Cauchy's bound, we conclude that

$$\begin{aligned} |\varepsilon| &> \left(1 + \frac{\max\{|b_1|, |b_2|, \ldots, |b_m|\}}{|b_0|}\right)^{-1} \\ &\geq (1 + \max\{|b_1|, |b_2|, \ldots, |b_m|\})^{-1} \\ -\lg|\varepsilon| &< 1 + \lg(\max\{|b_1|, |b_2|, \ldots, |b_m|\}) \\ &= 20L + O(1). \end{aligned} \tag{26}$$

¶8. Improved Bound and Explicit Constants. If we are implementing this algorithm, the additive constant "$+O(1)$" in (26) must be known. So let us explicitly compute it:

ε	$=$	$\frac{A+\sqrt{B}}{D} - \frac{A'+\sqrt{B'}}{D'}$	(definition of ε)
$DD'\varepsilon$	$=$	$D'(A+\sqrt{B}) - D(A'+\sqrt{B'})$	(clearing denominators)
$\sqrt{D^2B'}$	$=$	$\sqrt{D'^2B} + (E-\delta)$	(put $E = D'A - DA'$, $\delta = DD'\varepsilon$)
D^2B'	$=$	$D'^2B + (E-\delta)^2 + 2(E-\delta)\sqrt{D'^2B}$	(squaring)
$2(E-\delta)\sqrt{D'^2B}$	$=$	$(D^2B' - D'^2B) - (E-\delta)^2$	(rearranging)
	$=$	$F - (E-\delta)^2$	(put $F = D^2B' - D'^2B$)
$4(E-\delta)^2D'^2B$	$=$	$F^2 + (E-\delta)^4 - 2F(E-\delta)^2$	(squaring)
$(E-\delta)^4$	$=$	$2(E-\delta)^2(F + 2D'^2B) - F^2$	(rearranging)
	$=$	$2(E-\delta)^2G - F^2$	(put $G = D^2B' + D'^2B$)

[2] We are indebted to K. Mehlhorn for pointing out this.

Rewriting the last equation as the polynomial equation $Q(\delta) = 0$, we get

$$\begin{aligned} Q(\delta) &= \delta^4 - \delta^3[4E] + 2\delta^2[3E^2 - G] - 4\delta[E(E^2+G)] + [E^4 - 2E^2G + F^2]. \\ P(\varepsilon) &= Q(DD'\varepsilon) \\ &= \sum_{i=0}^{4} b_i \varepsilon^i. \end{aligned}$$

Since

$$\lg|D| \le 4 + 2L, \quad \lg|A| \le 4 + 3L, \quad \lg|B| \le 11 + 6L.$$

(see (7)), we conclude that

$$\lg|E| \le 9 + 5L, \quad \lg|F| \le 20 + 10L, \quad \lg|G| \le 20 + 10L.$$

We can now read off the coefficients of $P(\varepsilon)$:

$$\begin{aligned} b_4 &= (DD')^4 && (\text{so } \lg|b_4| \le 32 + 16L) \\ b_3 &= -4(DD')^3 E && (\text{so } \lg|b_3| \le 35 + 17L) \\ b_2 &= 2(DD')^2(3E^2 - G) && (\text{so } \lg|b_2| \le 38 + 18L) \\ b_1 &= -4DD'E(E^2 + G) && (\text{so } \lg|b_1| \le 40 + 19L) \\ b_0 &= E^4 - 2E^2G + F^2 && (\text{so } \lg|b_0| \le 41 + 20L) \end{aligned}$$

From (26), we finally obtain the following:

Lemma 7. *If the input numbers are L-bit integers, then the priorities α, α' of any two events in Fortune's algorithm must have a gap of at least*

$$-\lg|\alpha - \alpha'| < 41 + 19L.$$

This is a sharpening of Lemma 6. More importantly, it can be directly used in an implementation of Fortune's algorithm. We compute an approximate priority $\widetilde{\alpha}$ with absolute precision $43 + 19L$ bits, *i.e.*,

$$|\widetilde{\alpha} - \alpha| \le 2^{-43-19L}.$$

Similarly, $\widetilde{\alpha'}$ is a $43 + 19L$ bit approximation of α'. Then we compare $\alpha : \alpha'$ using the following procedure:

COMPARISON PROCEDURE
Input: $43 + 19L$-bit approximations $\widetilde{\alpha}$ and $\widetilde{\alpha'}$.
Output: The outcome of comparison $\alpha : \alpha'$

1. Compute $\varepsilon' = \widetilde{\alpha} - \widetilde{\alpha'}$.
2. If $\varepsilon' > 2^{-42-19L}$, return("$\alpha$ is larger than α'").
3. If $\varepsilon' < -2^{-42-19L}$, return("$\alpha$ is smaller than α'").
4. Return("α is equal to α'").

Justification: assume $\alpha = \widetilde{\alpha} + e$ and $\alpha' = \widetilde{\alpha'} + e'$ where $|e|$ and $|e'|$ is at most $2^{-43-19L}$. If $\varepsilon' > 2^{-42-19L}$ then

$$\begin{aligned} \alpha - \alpha' &= (\widetilde{\alpha} + e) - (\widetilde{\alpha'} + e') \\ &= \varepsilon' + (e - e)' \\ &> 2^{-42-19L} - |e| - |e'| \\ &\ge 0. \end{aligned}$$

The case $\varepsilon' < -2^{-41-19L}$ is similarly justified. The final case is $|\varepsilon'| \le 2^{-42-19L}$.

$$\begin{aligned} |\alpha - \alpha'| &= |(\widetilde{\alpha} + e) - (\widetilde{\alpha'} + e')| \\ &\le \varepsilon' + |e| + |e'| \\ &\le 2^{-42-19L} + 2^{-43-19L} + 2^{-43-19L} \\ &\le 2^{-41-19L}. \end{aligned}$$

This implies $\alpha = \alpha'$.

¶9. Discussion. The advantage of repeated squaring is that only integer operations are used. The disadvantage for using this for determining signs of expressions involving square roots is the need to evaluate a large Boolean function of many smaller predicates.

On the other hand, as first observed in [3], the existence of such root bounds shows that approximate square-roots can be used to make such comparisons. Note that the complexity of approximate square-roots is as fast as multiplication (both in theory and in practice). A bonus of our root bounds is that, we manipulated $19L + O(1)$ bit approximations, not $20L$-bit integers. But the most important advantage of using approximate square-roots seems to be this: suppose our priorities are α_i $(i = 1, 2, \ldots)$. Then for all $i < j$, the repeated squaring method requires the evaluation of $20L$-bit expressions that depend on both i and j. There could be $O(n \log n)$ such expressions to evaluate, assuming an $O(n \log n)$ time implementation of Fortune's algorithm. In contrast, when we use approximate arithmetic we only need to compute $20L$-bit approximate values for $O(n)$ expressions. Thereafter, all comparisons require no arithmetic, just a straight comparison of two $20L$-bit numbers. A further improvement to $15L$-bits was achieved by Sellen and Yap.

§4. A Practical Implementation

Fortune gave an implementation of his algorithm in the `C` langauge. We have converted this program into a Core Library program by introducing the minimal necessary changes. The program can output postscript scripts (to display images) or combinatorial structure of the Voronoi diagram.

Fortune's algorithm is practical and dispenses with the balanced binary tree T. Instead, it uses a bucketing scheme. An empirical study of Delaunay Triangulation algorithms, Drysdale and Su [10] found that for uniformly randomly generated points, the use of an $O(\log n)$ priority queue in Fortune's algorithm did not improve upon the bucketing scheme for less than 2^{17} points. At 2^{17} points, there was a 10% improvement.

The original implementation, assuming machine arithmetic, do not explicitly handle degenerate data: for instance, it assumes that all Voronoi vertices have degree 3. To investigate its behavior with respect to degeneracies, we compute the Voronoi diagram of a set of points on the uniform grid. SHOW RESULTS!

§5. Voronoi Diagram of Line Segments and Circular Arcs

The definition of Voronoi diagrams is capable of many generalizations. We consider the generalization where S is now a set of x-monotone curve segments. The reason for monotonicity property is that we wish to preserve the linesweep paradigm in our algorithm. We assume these segments are either line segments or circular arcs.

To understand this, consider the generalized parabola $B_s(t)$ where s is either a line segment or a circular arc. Note that $H(t)$ can now intersect s over a range of values: in other words, instead of a "site event" we have "start site" and "stop site" events.

Given a site s, we introduce the lines through the endpoints of s which are normal to s. When s is a line segment, this divides the plane into 4 **zones**, corresponding to the influence of the various features of the line segment. See Figure 7(a). Zones z_0 and z_1 comprise those points that are (respectively) closest to one of the two "sides" of s. Zones z_ℓ and z_r comprise those those points that are (respectively) closest to the two endpoints of s. When s is a circular arc, we introduce a similar set of zones. In this case, the side of the arc containing the center of the circle will have a bounded zone z_0, as seen in Figure 7(b). In either case, there is a left and right zone z_ℓ, z_r corresponding to the left and right endpoints of the arc.

Let us consider the **beach line** of a line segment s: this is basically the bisector $b(s, H(t))$ comprising all points that are equidistant from s and $H(t)$. Assume s is non-horizontal with endpoints p and q. In Figure 8(a) we show three arcs $\alpha, \alpha', \alpha''$ of the bisector. These arcs are part of the bisectors of $H(t)$ with (respectively) $q, \ell(s)$ and p. Here $\ell(s)$ is the line defined by s.

It turns out that unless s is a horizontal segment, there are two more arcs α''' and α'''' which may not be easy to see if s is close to horizontal. Altogether $b(s, H(t))$ has 5 arcs, where the arcs alternate between straight and parabolic. To see the existence of α''', we need to show that the parabola α'' intersects the normal through the endpoint p of the segment s (see Figure 9). The parabola will intersect the normal line at p at the point c which is the center of a circle through p and tangent to the current sweepline $H(t)$. It is clear that c exists. Hence α''' exists, and is the angle bisector of the lines $\ell(s)$ and $H(t)$. So α''' is a straight

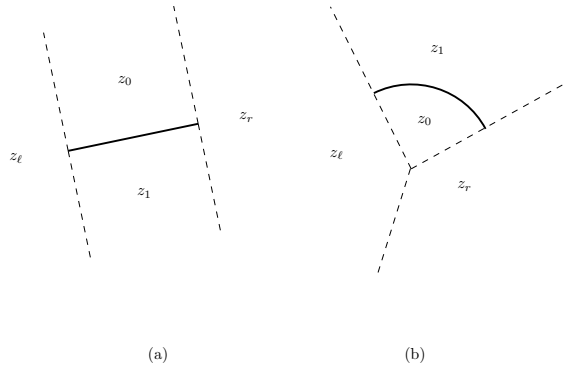

Figure 7: Zones of (a) Line Segment and (b) Circular Arc

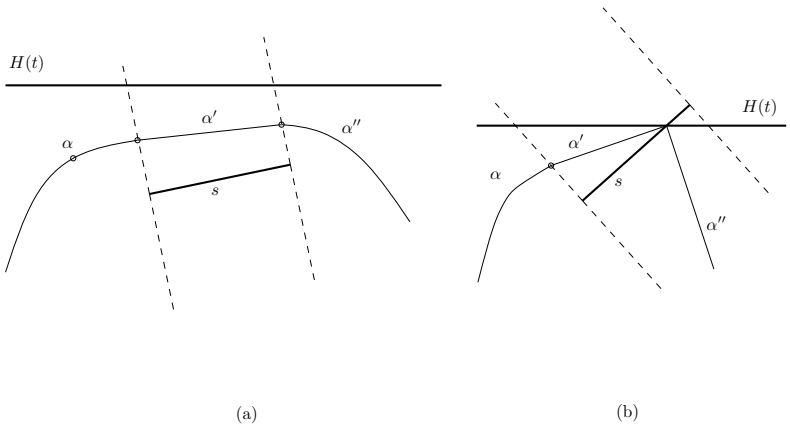

Figure 8: Beachline of a line segment s

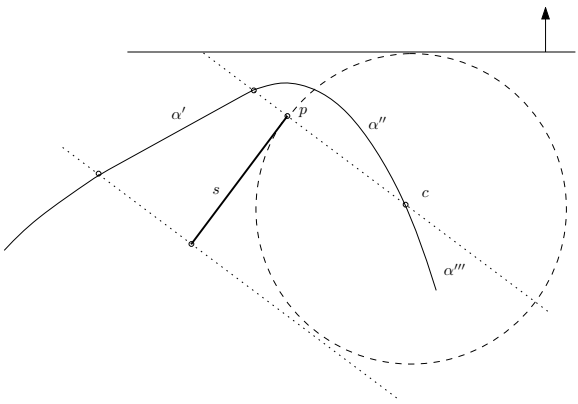

Figure 9: The existence of arc α'''

line segment. But α''' will intersect the normal line through the other endpoint of s, thereby ensuring that α'''' exists. Here α'''' is a parabolic segment with focus q and directrix $H(t)$.

Another possibility is for $H(t)$ to intersect s as in Figure 8(b). In this case, we are only interested in the part of this bisector that lies below $H(t)$. If $H(t)$ intersects s at some point q, then there are two angle bisectors emanating from q. To see that $b(s, H(t))$ will comprise of 4 arcs note that at the moment $H(t)$ touches s, the beach line generated by s is a half-ray emanating southward from s, representing a degenerate parabola, as in the case of point sites. But immediately after this, this parabola would be split into two parts, and joined by two straight line segments. This gives us the 4 arcs illustrated in Figure 8(b). Note that this is the case even when s is vertical.

We can extend this analysis to the case where $H(t)$ has swept past the other endpoint of s. At that moment, the two straightline segments become separated by a new parabolic arc generated by the second endpoint of s.

Next, we consider bisectors $b(s, H(t))$ between a circular arc s and the sweepline $H(t)$. Initially, assume $H(t)$ does not intersect s. The arc is either a cap or a cup (convex upwards or downwards). If the circular arc is a cap, the bisector $B_s(t)$ has a parabolic part separating two linear parts. The focus and directrix of the parabola are (respectively) the center of the arc and the displaced sweepline $H(t + r)$, where r is the radius of the arc. If a cup-arc, then again we have a parabolic arc with directrix $H(t - r)$ and focus at the center of the arc. The parabolic arc is confined to the bounded zone z_0 of the arc Figure 7(b). Outside, the zone, we have parabolic arcs determined by the endpoints of the arc. Note that if $H(t)$ intersects s, the bisector $b(s, H(t))$ ough to be limited to the half-space below $H(t)$. These curves are essentially restrictions of the curves explained as below.

What about circle events? Now we can have Voronoi vertices generated by i linesegments and $3 - i$ endpoints, for any $i = 0, 1, 2, 3$. Call this a **i-circle event**. The original circle event corresponds to a 0-circle event. See Figure 10.

THIS IS A PLACE HOLDER – NO FIGURE YET

Figure 10: Four kinds of Circle Events

We can now modify the original algorithm of Fortune so that instead of site events, we now have **start event** and **stop event** corresponding to the first and last encounter of a site. At a start event, we insert four arcs as in Figure 8(b). At a stop event, we insert one more arc into (whatever remains of) the original four.

The modification for the rest of the algorithm is left as an exercise. The analysis of the geometry and algebraic root bounds, together with implementation, have been carried out by Burnikel [1] in his PhD Thesis (University of Saarbrücken, 1996).

Exercises

Exercise 5.1: Prove Lemma 1. ♢

Exercise 5.2: Describe some concrete ways in which the algorithm of Fortune can fail because of numerical errors in deciding the priority of events. You must provide concrete numerical examples to illustrate how it fails. HINT: think of very degenerate inputs where 4 or more points are co-circular so Voronoi vertices are very close to each other. ♢

Exercise 5.3: Re-derive our root bounds for Fortune's algorithm if the input numbers are binary floats with L-bit mantissas and exponents at most E. ♢

Exercise 5.4: Here are some generally open questions for exploration with Voronoi diagrams:
(i) Given $n \geq 1$, to find a set S with n sites which is the optimal location for n post offices. Various optimality criteria can be studied. For instance, assume S is restricted to the interior of a simple polygon P, and we want to minimize the area of the largest Voronoi region of S within P.
(ii) Suppose S is given, and we want to locate a new site s so that $S \cup \{s\}$ will optimize the criteria of (i). ♢

Exercise 5.5: Show how to do basic operations using the half-edge data structure: (i) getting to the previous half-edge in constant time, (ii) listing all the regions around a vertex in linear time. ♢

Exercise 5.6: Work out the other predicates needed for Fortune's sweep, and compute their corresponding root bounds. ♢

Exercise 5.7: Suppose the input numbers are IEEE Machine doubles. What is the corresponding bound in Lemma 7? ♢

Exercise 5.8: (i) Give the minimal polynomials $P_2(X)$ and $P_3(X)$ for the following numbers: $\alpha_2 = \sqrt{A} + \sqrt{B}$, $\alpha_3 = \sqrt{A} + \sqrt{B} + \sqrt{C}$ for positive but indeterminate integers A, B, C.
(ii) The constant terms of these polynomials are squares. Is this true for $P_n(X)$ for all n? ♢

Exercise 5.9: If $p, q, p', q' \in \mathbb{R}^2$ are points whose coordinates are IEEE machine single precision floats (i.e., numbers of the form $m2^{e-23}$, $|m| < 2^{24}$ and $|e| \leq 127$) we want a separation bound on

$$\delta = \|p - q\| - \|p' - q'\|$$

where $\|p - q\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$. In fact, show that if $\delta \neq 0$ then

$$|\delta| > 2^{431}.$$

HINT: Apply Cauchy's bound to the polynomial $P_2(X)$ obtained in the previous exercise. ♢

Exercise 5.10: The circle through three points p, q, r has equation $C(X, Y) = 0$ where

$$C(X,Y) \quad = \quad \det \begin{bmatrix} X^2+Y^2 & X & Y & 1 \\ p^2 & p_x & p_y & 1 \\ q^2 & q_x & q_y & 1 \\ r^2 & r_x & r_y & 1 \end{bmatrix} \tag{27}$$

♢

Exercise 5.11: Suppose $p, q, p', q' \in \mathbb{R}^2$ are points whose coordinates are represented by machine floats (i.e., IEEE single precision floating point numbers). For this question, they are just numbers of the form $x = m2^{e-23}$ where m, e are integers satisfying $|m| < 2^{24}$ and $|e| \leq 127$. Let

$$\delta = \|p - q\| - \|p' - q'\| \tag{28}$$

where $\|p - q\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$. We want a separation bound on $|\delta|$.
(i) Let $\Delta = M\delta$ where M is the smallest positive integer such that $M\delta$ is equal to $\sqrt{A} - \sqrt{B}$ for some integers A, B. Give upper bounds on M, A and B.
(ii) Give a polynomial $P(X)$ such that $P(\Delta) = 0$, written in terms of A, B.
(iii) Apply Cauchy's bound to conclude that if $\delta \neq 0$ then $|\delta| > 2^{-431}$.
(iv) Extra Credit: Give an actual numerical example that comes as close to your lower bound as possible. ♢

Exercise 5.12: Let $e(x) = f$ where $x = m2^{f-23}$ is a IEEE single precision float, with $|m| < 2^{24}$ and $|f| \leq 127$. In the previous question, suppose $e_0 = \min\{e(p_x), e(p_y), e(q_x), e(q_y), e(p'_x), e(p'_y), e(q'_x), e(q'_y)\}$ and $e_1 = \max\{e(p_x), e(p_y), e(q_x), e(q_y), e(p'_x), e(p'_y), e(q'_x), e(q'_y)\}$. Redo parts (i) and (iii) of the question, where e_0 and e_1 are taken into account. ♢

End Exercises

§6. Additional Notes

The book of Okabe, Boots and Sugihara [9] is devoted to Voronoi diagrams, their generalizations and applications. We follow the site/circle event terminology of Guibas and Stolfi [5]. The beachline terminology is from [4]. The detailed analysis of the root bounds necessary for Fortune's algorithm is from Dubé and Yap [3]. This seems to be the first time[3] when it was realized that numerical approximations could be used to solve geometric computation problems in the "exact geometric sense". The retraction approach to motion planning was first applied to a disc in O'Dunlaing and Yap [8]; this can be extended to other kinds of motion planning problems [7, 11, 2]. Yap had obtained the first correct $O(n \log n)$ algorithm for Voronoi diagrams for line segments and circular arcs using a divide-and-conquer algorithm. Unlike Fortune's linesweep approach, it is based on divide and conquer paradigm, and it can be used in parallel algorithms as well as for constrained Voronoi diagrams.

References

[1] C. Burnikel. *Exact Computation of Voronoi Diagrams and Line Segment Intersections*. Ph.D thesis, Universität des Saarlandes, Mar. 1996.

[2] J. Cox and C. K. Yap. On-line motion planning: case of a planar rod. *Annals of Mathematics and Artificial Intelligence*, 3:1–20, 1991. Special journal issue. Also: NYU-Courant Institute, Robotics Lab., No.187, 1988.

[3] T. Dubé and C. K. Yap. A basis for implementing exact geometric algorithms (extended abstract), September, 1993. Paper from `http://cs.nyu.edu/exact/doc/`.

[4] M. T. Goodrich, C. Ó'Dúnlaing, and C. Yap. Constructing the Voronoi diagram of a set of line segments in parallel. *Algorithmica*, 9:128–141, 1993. Also: Proc. WADS, Aug. 17-19, 1989, Carleton University, Ottawa, Canada. *Lecture Notes in C.S.* No.382, Springer (1989)12–23.

[5] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, 1985.

[6] J. R. Munkres. *Topology: A First Course*. Prentice-Hall, Inc, 1975.

[7] C. Ó'Dúnlaing, M. Sharir, and C. K. Yap. Retraction: a new approach to motion-planning. *ACM Symp. on Theory of Computing*, 15:207–220, 1983.

[8] C. Ó'Dúnlaing and C. K. Yap. A "retraction" method for planning the motion of a disc. *J. Algorithms*, 6:104–111, 1985. Also, Chapter 6 in *Planning, Geometry, and Complexity*, eds. Schwartz, Sharir and Hopcroft, Ablex Pub. Corp., Norwood, NJ. 1987.

[9] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations — Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons, Chichester, 1992.

[10] P. Su and R. L. S. Drysdale. A comparison of sequential delaunay triangulation algorithms. In *Proc. 11th ACM Symp. Comp. Geom.*, pages 61–70, 1995. Vancouver, British Columbia, Canada.

[11] C. K. Yap. Coordinating the motion of several discs. Robotics Report 16, Dept. of Computer Science, New York University, Feb. 1984.

[12] C. K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, 2000.

[3] Until then, most papers assumed that to compute exactly, one must reduce all computation to exact integer or rational computations.