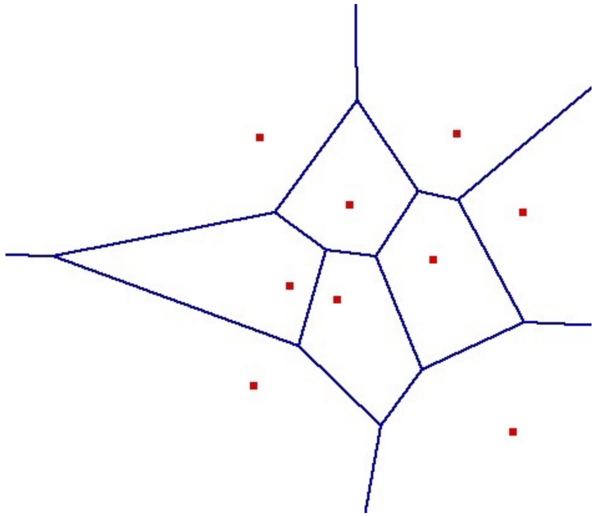
Computational Geometry



Constructing Voronoi Diagrams

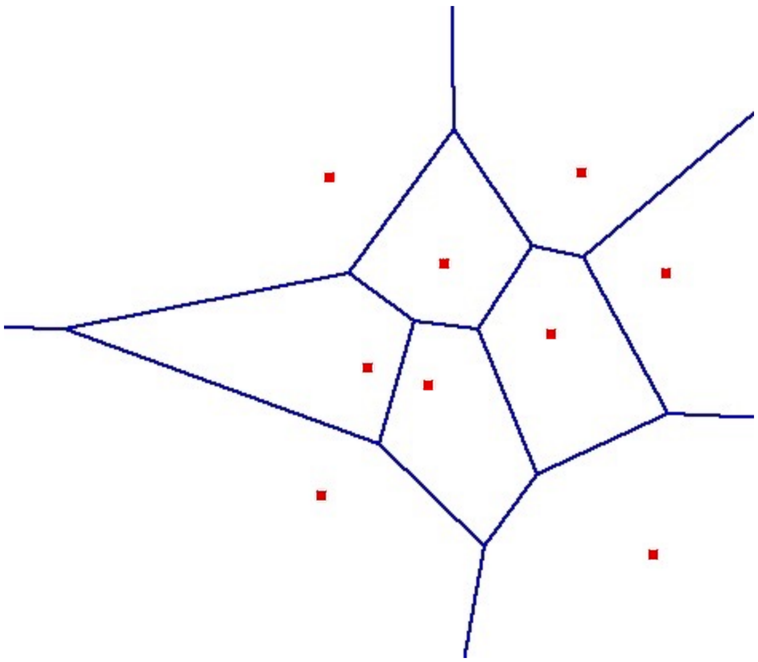
Michael Goodrich

with slides from Carola Wenk

Voronoi Diagram Definition

(Dirichlet Tesselation)

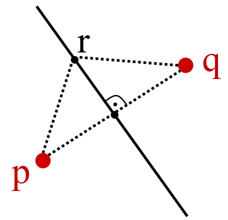
- **Given:** A set of point sites $P = \{p_1, \ldots, p_n\} \subseteq R^2$
- **Voronoi cell:** Partition R^2 into Voronoi cells
  $V(p_i) = \{q \in R^2 |\ d(p_i, q) < d\left(p_j, q\right)$ for all $j \neq i\}$
- **Voronoi diagram:** The planar subdivision obtained by removing all Voronoi cells from R^2.



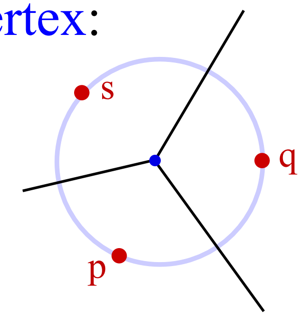
Bisectors

- Voronoi edges are portions of bisectors
- For two points p, q, the bisector $b(p,q)$ is defined as

$$b(p,q) = \{r \in R^2 \mid d(p,r) = d(q,r)\}$$



- Voronoi vertex:

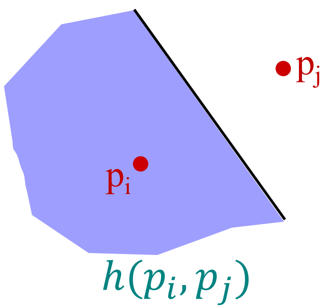


Voronoi cell

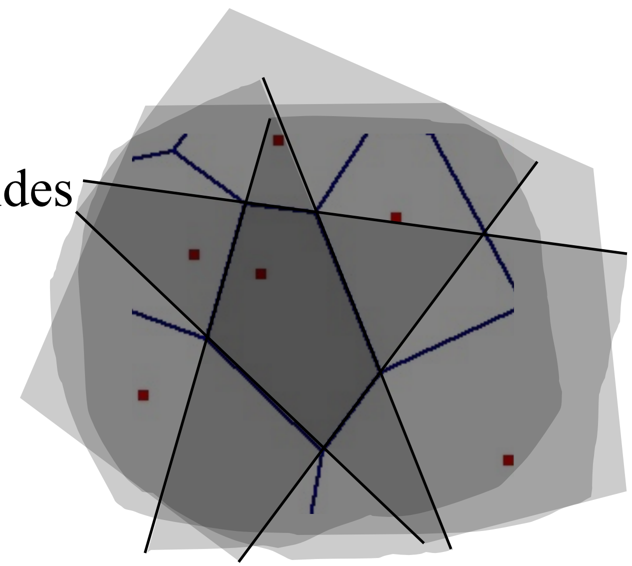
- Each Voronoi cell $V(p_i)$ is convex and

$$V(p_i) = \bigcap_{\substack{p_j \in P \\ j \neq i}} h(p_i, p_j),$$

where $h(p_i, p_j)$ is the halfspace that is defined by bisector $b(p_i, p_j)$ and that contains p_i



$\Rightarrow$ A Voronoi cell has at most $n-1$ sides



Voronoi Diagram

The Voronoi diagram is a planar, embedded, connected graph with vertices, edges (possibly infinite), and faces (possibly infinite).

- **Theorem:** Let $P = \{p_1, \ldots, p_n\} \subseteq R^2$. Let n_v be the number of vertices in $VD(P)$ and let n_e be the number of edges in $VD(P)$. Then

  $n_v \leq 2n - 5$, and

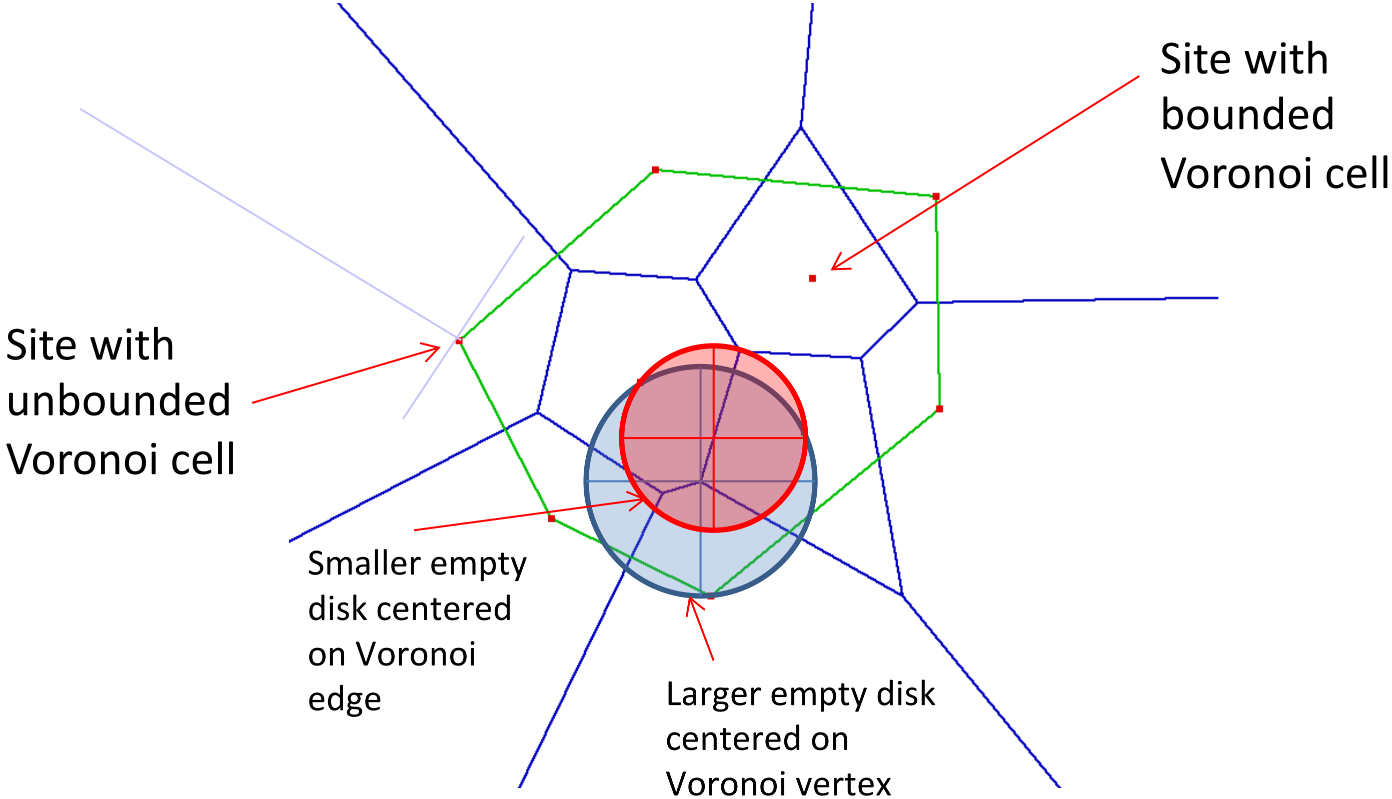
  $n_e \leq 3n - 6$

Add vertex at infinity

Proof idea: An application of Euler's formula $n_v + 1 - n_e + n = 2$ with "$=$" because the planar graph is connected, and $2n_e = \sum_{v \in V} \deg(v) \geq 3(n_v + 1)$.

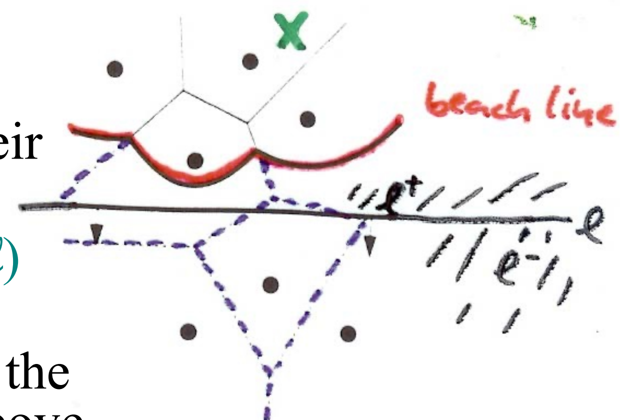
Properties

1. A Voronoi cell $V(p_i)$ is unbounded iff p_i is on the convex hull of the sites.
2. v is a Voronoi vertex iff it is the center of an empty circle (disk) that passes through three sites.



Fortune's sweep to construct the VD

Problem: We cannot maintain the intersection of the VD with sweepline ℓ since the VD above ℓ depends on the sites below ℓ.

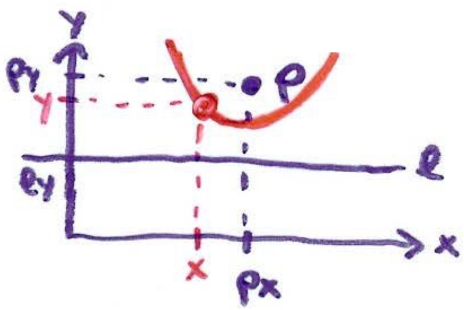


Sweep line status: "Beach line"

- Identify points $q \in \ell^+$ for which we know their closest site.
- If there is a site $p_i \in \ell^+$ s.t. $\mathrm{dist}(q,p_i) \leq \mathrm{dist}(q, \ell)$ then the site closest to q lies above ℓ.
- Define the "beach line" as the boundary of the set of points $q \in \ell^+$ that are closer to a site above ℓ than to ℓ.
  - → The beach line is a sequence of parabolic arcs
  - → The breakpoints (beach line vertices) lie on edges of the VD, such that they trace out the VD as the sweep line moves.

Parabola

Set of points (x,y) such that dist((x,y), p) = dist(ℓ) for a fixed site $p = (p_x,p_y)$



$$(p_x - x)^2 + (p_y - y)^2 = (y - \ell_y)^2$$

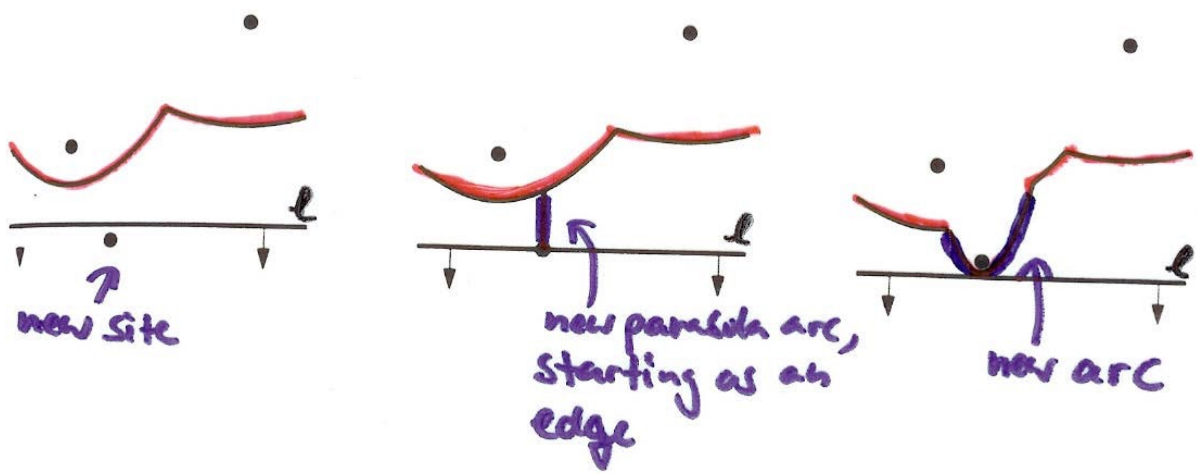
$$p_x^2 - 2p_x x + x^2 + p_y^2 - 2p_y y + y^2 = y^2 - 2y\ell_y + \ell_y^2$$

$$y = \frac{p_x^2 - 2p_x x + x^2 + p_y^2 - \ell_y^2}{2(p_y - \ell_y)}$$

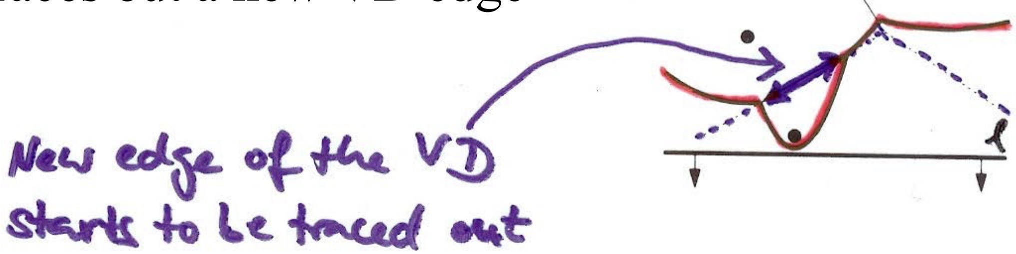
Site Events

Site event: The sweep line ℓ reaches a new site

$\Rightarrow$ A new arc appears on the beach line…



… which traces out a new VD edge

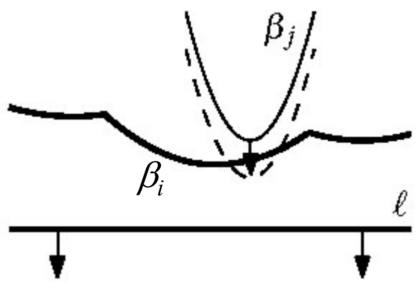


Site Events

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

Proof:

- **Case 1:** Assume the existing parabola β_j (defined by site p_j) breaks through β_i

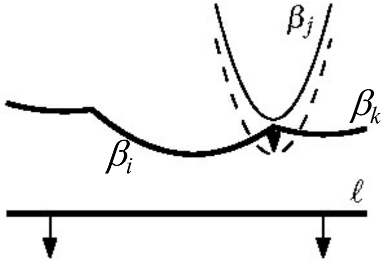


Formula for parabola β_j : $y = \frac{1}{2(p_{jy} - \ell_y)} \cdot (p_{jx}^2 - 2 p_{jx} x + x^2 + p_{jy} - \ell_y^2)$

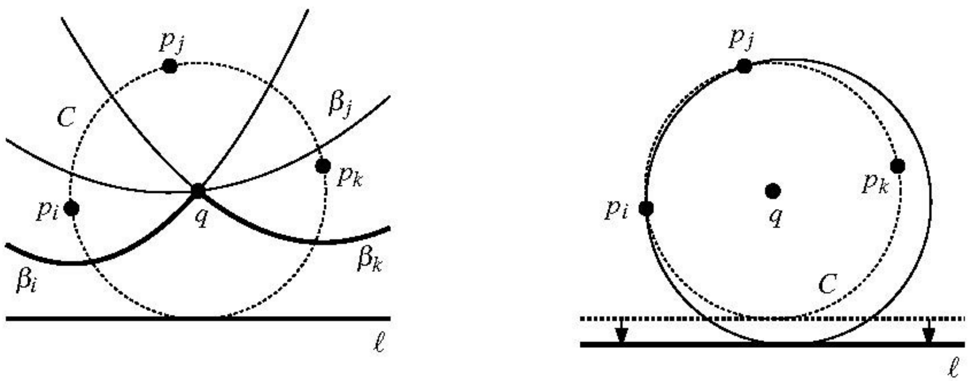
Using $p_{jy} > \ell_y$ and $p_{iy} > \ell_y$ one can show that it is impossible that β_i and β_j have only one intersection point. Contradiction.

Site Events

- **Case 2:** Assume β_j appears on the break point q between β_i and β_k



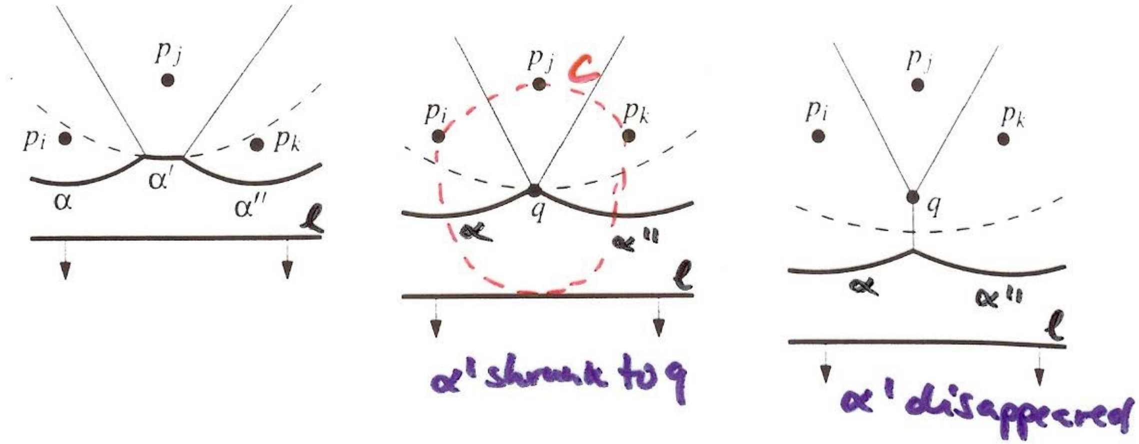
$\Rightarrow$ There is a circle C that passes through p_i, p_j, p_k and is tangent to ℓ:



But for an infinitesimally small motion of ℓ, either p_i or p_k penetrates the interior of C. Therefore β_j cannot appear on ℓ. □

Circle Events

Circle event: Arc α' shrinks to a point q, and then arc α' disappears



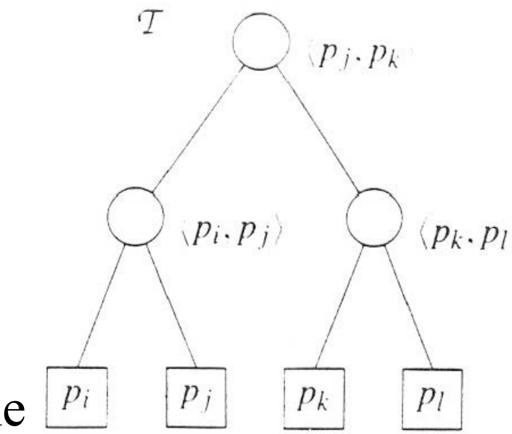
⇒ There is a circle C that passes through p_i, p_j, p_k and touches ℓ (from above).

⇒ There is no site in the interior of C. (Otherwise this site would be closer to q than q is to ℓ, and q would not be on the beach line.)

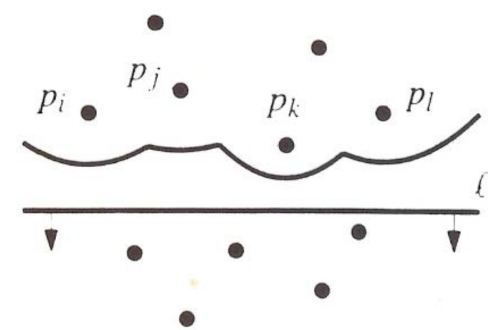
⇒ q is a Voronoi vertex (two edges of the VD meet in q).

⇒ **Note:** The only way an arc can disappear from the beach line is through a circle event.

Data Structures



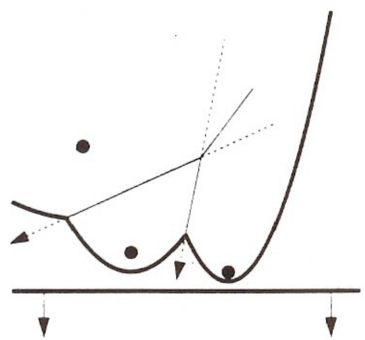
- **Store the VD** under construction in a DCEL
- **Sweep line status** (sweep line):
  - Use a balanced binary search tree $\mathcal{T}$, in which the leaves correspond to the arcs on the beach line.
  - Each leaf stores the site defining the arc (it only stores the site and not the arc)
  - Each internal node corresponds to a break point on the beach line



- **Event queue**:
  - Priority queue Q (ordered by y-coordinate)
  - Store each point site as a site event.
  - Circle event:
    - Store the lowest point of a circle as an event point
    - Store a point to the leaf/arc in the tree that will disappear

How to Detect Circle Events?

Make sure that for any three consecutive arcs on the beach line the potential circle event they define is stored in the queue.



⇒ Consecutive triples with breakpoints that do not converge do not yield a circle event.

⇒ Note that a triple could disappear (e.g., due to the appearance of a new site) before the event takes place. This yields a false alarm.

Sweep Code

Algorithm VORONOIDIAGRAM(P)

Input. A set $P := \{p_1, \ldots, p_n\}$ of point sites in the plane.

Output. The Voronoi diagram Vor(P) given inside a bounding box in a doubly-connected edge list $\mathcal{D}$.

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

Sweep Code

- **Degeneracies:**
  - If two points have the same y-coordinate, handle them in any order.
  - If there are more than three sites on one circle, there are several coincident circle events that can be handled in any order. The algorithm produces several degree-3 vertices at the same location.

- **Theorem:** Fortune's sweep runs in $O(n \log n)$ time and $O(n)$ space.

Handling a Site Event

HANDLESITEEVENT(p_i)

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

Runs in O(log *n*) time per event, and there are *n* events.

Handling a Circle Event

HANDLECIRCLEEVENT(γ)

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

Runs in O(log *n*) time per event, and there are O(*n*) events because each event defines a Voronoi vertex. False alarms are deleted before they are processed.