Voronoi Diagrams—CSC/Math 870 Lecture, April 5, 2007
Brendan Colloran

Motivation

Voronoi diagrams are used to partition a metric space by proximity to a discrete set of objects. Some example of problems for which Voronoi diagrams are useful include:

- Post office problem
- Trade influence of cities
- Local resource use for plants ("potential area available to a tree")
- Territory of central place foragers (and other types of animal territoriality)
- Modeling grain growth in metals
- Regional gravitational influence of astronomical objects

For many more examples, see http://www.ics.uci.edu/~eppstein/gina/scot.drysdale.html

Definition

Given a set of points (or "*sites*") $P := \{p_1, p_2,\ldots,p_n\}$, the Voronoi diagram of P is a subdivision of the plane into n cells (one for each element of P) such that a point q is in cell i if and only if q is closer to p_i than it is to any other element of P.

Notation

The Voronoi diagram of a point set P will be denoted 'Vor(P)'; abusing this terminology, we will also use 'Vor(P)' to denote the vertices and edges of this planar subdivision.

We denote the Voronoi cell of site p_i by '$V(p_i)$'.

Glossary

Site — A point p_i in the set $P := \{p_1, p_2,\ldots,p_n\}$.

Voronoi cell of p_i — The portion of the plane that is closer to site p_i than any other site.

The beach line: — For a given position of the sweep line L, each site p_i above L defines a parabola Π_i with focus p_i and directrix L. The *beach line* is the function

$$f(x) = \min\{\Pi_i(x)\}$$

for all i such that p_i is above L.

Breakpoints: — The points at which consecutive parabolic arcs on the beach line meet.

Site events: The event that occurs when the sweep line encounters a new site.

Circle events: The event that occurs when the sweep line reaches the lowest point on the circle through the sites defining three consecutive points on the beach line.

False alarm: A potential circle event that is deleted from the event queue before it can take place.

Theorems, Observations, Lemmas

Observation 7.1 Let Bis(p_i,p_j) denote the perpendicular bisector of the line segment connecting p_i and p_j, and let $h(p_i,p_j)$ denote the half-plane containing p_i that is defined by Bis(p_i,p_j). Then $V(p_i)$ is the intersection of the half-spaces $h(p_i,p_j)$ with $i \neq j$.

Observation 7.1.a Each cell $V(p_i)$ has at most n - 1 vertices and edges.

Theorem 7.2 Let P be a set of points in the plane. If all the points are collinear, then Vor(P) consists of n - 1 parallel lines. Otherwise, Vor(P) is connected, and its edges are segments or half-lines.

Theorem 7.3 For $n \geq 3$, Vor(P) has has at most $2n$ - 5 vertices and $3n$ - 6 edges.

Theorem 7.4 For a set of points P, define the largest empty circle about q with respect to P, denoted $C_P(q)$, as the largest circle with center q that does not contain any other points in P. Then:

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

Lemma 7.6 New arcs can appear on the beach line only by way of site events.

Lemma 7.7 Existing arcs can disappear from the beach line only by way of circle events.

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

Lemma 7.9 Fortune's algorithm runs in $O(n \log n)$ time and uses $O(n)$ storage.

Data Structures

- The Voronoi diagram is stored in a doubly-connected edge list D (see Ch. 2). (Note that because a Voronoi diagram has half-lines as well as full lines, we must add a bounding box to complete the doubly connected edge list.)

- Events are stored in a priority queue Q, where an event's priority is its y-coordinate.

- The beach line is stored in a balanced binary search tree T, in which the leaves correspond to arcs on the beach line and internal nodes correspond to breakpoints. Breakpoints are stored as ordered tuples (p_i,p_j), where p_i represents arc to the left of the breakpoint and p_j represents the arc to the right of the breakpoint. This allows us to calculate the x-coordinate of the breakpoints at each site event, and hence to find the arc of the beach line that is above a new site.
  We also store pointers in T to our other data structures. Each leaf (representing an arc) has a pointer to the circle event in Q that will cause the arc to disappear (this is set to nil if no such event has been detected), and each internal node (p_i,p_j) has a pointer to a half edge in the doubly connected edge list that is traced by the breakpoint (p_i,p_j).

The Algorithm

VoronoiDiagram(P)

Input A set P of point sites in the plane.

Output A doubly connected edge list D representing Vor(P) inside a bounding box

1. Initialize Q with all site events, and initialize T and D (both empty).
2. **while** Q is not empty
3. **do** Remove the highest priority event from Q
4. **if** the event is a site event occurring at p_i
5. **then** HandleSiteEvent(p_i)
6. **else** HandleCircleEvent(a), where a is the leaf of T representing the arc that will disappear
7. The internal nodes still in T correspond to half-infinite edges. Compute a bounding box that contains all the sites of P and all the vertices of Vor(P), and attach the half-infinite edges to the bounding box.
8. Complete the doubly-connected edge list by adding cell records and pointers to corresponding edges of Vor(P)

HandleSiteEvent(p_i)

1. If T is empty, insert p_i into T and return; otherwise, proceed with steps 2-5.
2. Search T for the arc α vertically above p_i. If this arc has a corresponding circle event in Q, that circle event is a false alarm and must be deleted.
3. Replace the leaf of T representing α with a subtree having three leaves: the middle leaf stores the new site p_i, and the two other leaves store the site p_j that was originally stored with α. Store the tuples (p_j,p_i) and (p_i,p_j) representing the new breakpoints at the two new internal node. Perform balancing operations on T.
4. Create new half-edge records in D for the edge separating $V(p_i)$ and $V(p_j)$.
5. Check the triple of consecutive arcs with p_i as the left arc to see if the breakpoints converge; if they do, insert a circle event in Q and add pointers between the nodes in T and Q. Do the same for the triple with the new arc on the right.

HandleCircleEvent(a)

1. Delete the leaf a that represents the arc α disappearing from T. Update the tuples at internal nodes representing breakpoints. Rebalance T. Delete all circle events involving α from Q (these can be found using the pointers from the predecessor and successor of a in T).
2. Add the center of the circle causing the event to D as a vertex. Create two half-edge records in D corresponding to the new breakpoint, and set the appropriate pointers. Attach the three relevant half-edges, including the new one, to the new vertex.
3. Check the new triple of consecutive arcs that has the former left neighbor of α as it middle arc to see if its breakpoints converge; if they do, insert a circle event in Q and add pointers between the nodes in T and Q. Do the same for the triple with the former right neighbor of α as it middle arc.