CS 6301.008.18S Lecture—February 13, 2017

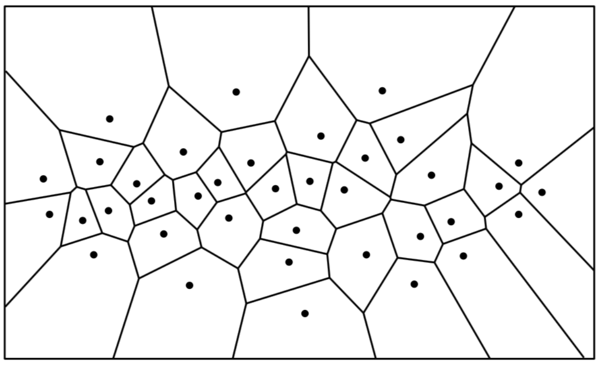
Main topics are #Voronoi-diagrams, #Fortune.

Quick Note about Planar Point Location

- Last week, I started giving a difficult analysis of the planar point location data structure.
- We should move on, but I do want to mention the final punchline.
- There is an O(n log n) *expected* time algorithm to construct a planar point location data structure with *worst-case* O(n) storage and O(log n) search time.
- Essentially, you'll probably build the good structure in your first try of running that randomized incremental construction.

Voronoi Diagrams

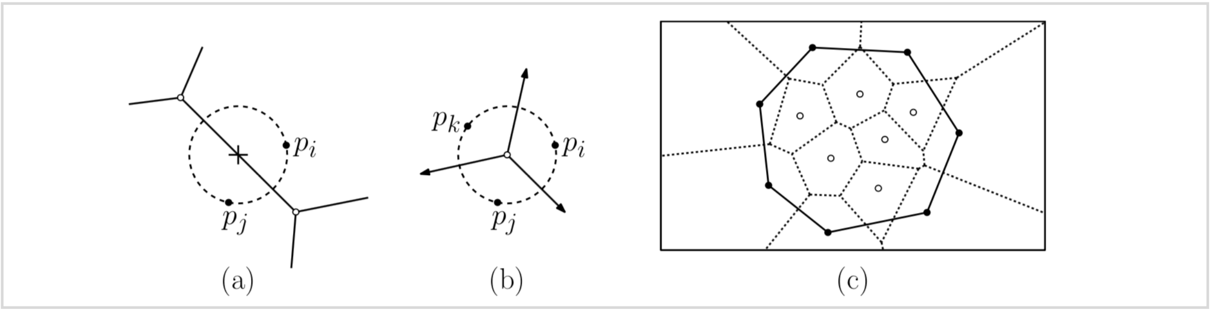
- Let P = {p_1, …, p_n} be a set of n points in R^d we call *sites*.
- Let ||pq| = sqrt((p_x - q_x)^2 + (p_y - q_y)^2) be the Euclidean distance between p and q.
- The Voronoi cell of site p_i, denote V(p_i) is the set of points closer to p_i than any other site.
    - V(p_i) = {q in R^2 : ||p_i q|| < ||p_j q||, for all j ≠ i}
- The union of the closure of the Voronoi cells forms the *Voronoi diagram*.



- The cells are (possibly unbounded) convex polyhedra, since V(p_i) is the intersection of all the halfspaces for points closer to p_i than each other point
- Voronoi diagrams have *a lot* of uses, including
    - nearest neighbor queries: to find the nearest neighbor of a point q, just do point location in the Voronoi diagram
    - shape analysis: we can get a useful sketch of a polygonal shape called the media axis if we compute the Voronoi diagram of its vertices and edges
    - center-based clustering: we want to partition a set P into subsets of points that are close together. If we choose some centers for these subsets, then the Voronoi diagram over the centers tells us which points belong in each cluster

Properties

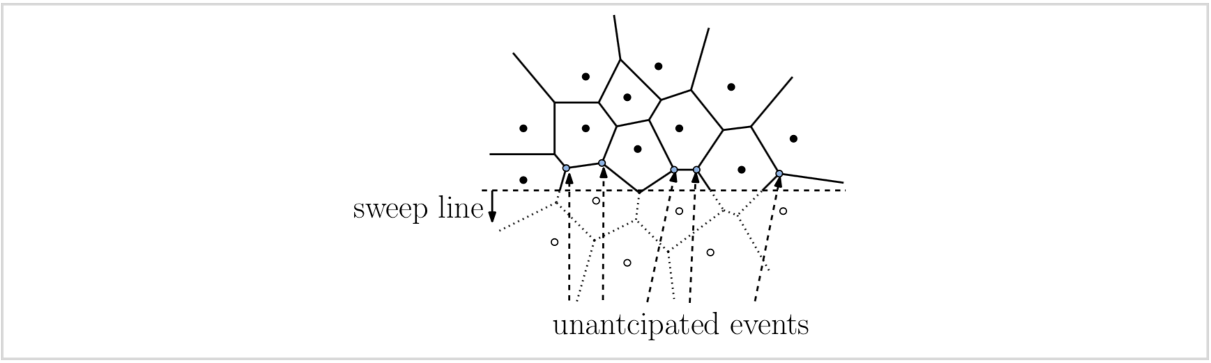
- Voronoi diagrams have several nice properties that make them useful and will be useful in their construction



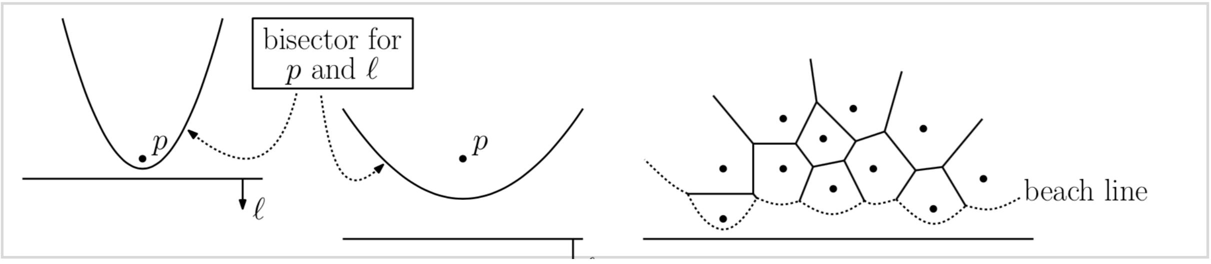
- Empty circle property: Each point on an edge of the Voronoi diagram is equidistant from its nearest neighbors. Therefore, there is a circle centered at that point through the neighbors with no other site interior to the circle.
- Voronoi vertices: Each Voronoi vertex is equidistant to three sites. Therefore, there is a circle centered at the vertex, passing through the three sites, with no other vertex interior.
- Assuming no four sites are cocircular, each vertex has degree 3.
- The Voronoi diagram has n faces, roughly 2n vertices, and roughly 3n edges. The book has a proof.

Computing the Voronoi Diagram

- We could compute the Voronoi diagram in O(n^2 log n) time by solving n halfspace intersection problems, one per site.
- Instead, I'll give an O(n log n) time sweep line algorithm due to Fortune ['87].
- There's also a randomized incremental construction algorithm. You should see a version of that next week for a related problem.
- So for this algorithm, I'm going to sweep top-down. Partly to match Mount's notes. Partly because certain drawings will be nicer that way.
- Now, it's temping to try designing the algorithm in the following way: sweep from top to bottom and maintain the whole Voronoi diagram from infinity down to the sweep line. Like in line segment intersection, we'll try to discover Voronoi vertices before we reach them and add them to the event queue if necessary.
- But it's not that simple:



- Sites we haven't reached yet will create Voronoi vertices we've already passed!
- So instead, we'll accept that we haven't computed everything up to the sweep line. Instead, we'll have computed everything up to an x-monotone curve called the *beach line* that lags behind the sweep line and essentially forms the boundary of what we know so far.
- More formally, the sweep line divides the plane into two halves, the stuff above it that we've swept already and the stuff below.
- We'll treat the sweep line as another infinitely long site. The beach line is just the boundary of its site: those points equidistant from their nearest neighbor above the line and the line itself.
- Anything we've computed strictly above the beach line belongs to the final Voronoi diagram: after all, the sweep line is already closer than any site that lies below it.
- OK, so what does the beach line look like and why do we call it a beach line?
- Given the sweep line ell and a site p, the points equidistant from both form an x-monotone *parabola*. As ell moves downward, the parabola gets fatter. In contrast, the parabola is a vertical ray shooting up if p lies on ell.
- The beach line is the lower envelope of the parabolas for all the sites, so its composed of several parabolic arcs.

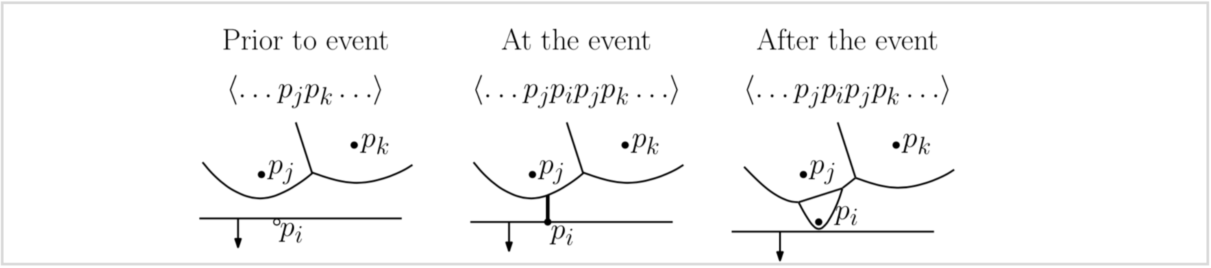


- The arcs intersect at *breakpoints* which are equidistance between the arcs' points and ell. Meaning breakpoints lie on Voronoi edges.
- Our goal is to simulate the movement of the beach line as the sweep line moves downward. Breakpoints will trace the path of the Voronoi edges.

Sweep Line Status and Events

- We don't have to track all continuous changes, so we just need to find figure out some useful sweep line status and important events.

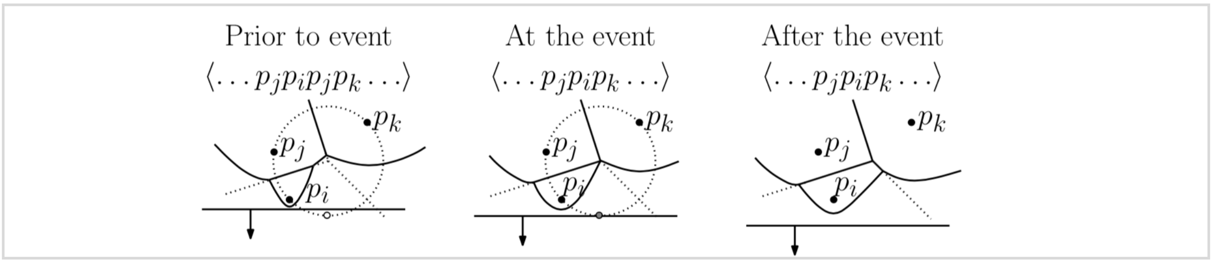
- Sweep line status:
  - y-coordinate of the sweep line
  - sites defining the beach line arcs in left-to-right order (a parabola can appear on the lower envelope multiple times, so some sites may appear in the list multiple times!)
  - We **do not** store the parabolas themselves or their equations.
- Events:
  - site events: when the sweep line passes over a site: a new parabolic arc joins the beach line
  - Voronoi vertex events: if the length of a beach line arc shrinks to 0, then its incident breakpoints collide, forming a vertex. The book calls these circle events.
- Let's look at the two event types in more details.
- Site events:
  - Say the sweep line passes over p_i.
  - At the moment of a site event, we get a degenerate parabolic arc shooting up from p_i.
  - Let's say that ray hits a beach line arc for p_j.
  - The arc gets split into two, and the new arc for p_i starts growing.
  - So, we replace a sweep line status entry for p_j with p_j p_i p_j



Prior to event
$\langle \ldots p_j p_k \ldots \rangle$
p_k
p_j
p_i
At the event
$\langle \ldots p_j p_i p_j p_k \ldots \rangle$
p_k
p_j
p_i
After the event
$\langle \ldots p_j p_i p_j p_k \ldots \rangle$
p_k
p_j
p_i

- The book proves that site events are the only way to add new arcs to the beach line. Each site event after the first creates a net of two new arcs, so the beach line and therefore sweep line status contains at most 2n - 1 parabolic arcs.
- Voronoi vertex events:
  - Remember, Voronoi vertices are made when beach line arcs shrink to nothing.
  - So we focus on the three adjacent arcs involved in this process, so p_i, p_j, and p_k's arcs appear consecutively in left-to-right order.
  - Suppose their circumcenter will eventually become a Voronoi vertex, where the breakpoint/bisector for p_i and p_j will meet the one for p_j and p_k.
  - Immediately before their breakpoints meet, the circumcircle lies partially below the sweep line, but there is no other site reached by the sweep line before the breakpoints meet.
  - At the moment the sweep line goes tangent to the circumcircle, the breakpoints meet and p_j's arc disappears.
  - From that point forward, p_i and p_k share a breakpoint which traces out a new

Voronoi edge from the vertex we just discovered.



Prior to event
$\langle \dots p_j p_i p_j p_k \dots \rangle$
At the event
$\langle \dots p_j p_i p_k \dots \rangle$
After the event
$\langle \dots p_j p_i p_k \dots \rangle$
p_k
p_j
p_i

The Algorithm

- So now we've seen enough to present the algorithm.
- Data structures:
  - A (partial) Voronoi diagram stored as a doubly connected edge lists. Some edges are unbounded or dangling, but we can pretend that the unbounded edges all connect at an imaginary Voronoi vertex at infinity. I'll assume we can update the diagram in constant time per change.
  - The beach line is stored as a sequence of sites owning its arcs in left-to-right order using an ordered dictionary. It needs to support some operations in O(log n) time each:
    - Search: Given the current y-coordinate of the sweep line and a new site p_i, find the arc lying above p_i. To do this fast, we need some way to say whether p_i lies left or right of a breakpoint so we can binary search. Suppose p_j and p_k share a breakpoint. That breakpoint is at the center of the circle through p_j and p_k that is tangent to the sweep line, so compare p_i to the center of that circle.
    - Insert and split: Insert an arc for p_i within a given arc for p_j, replacing beach line <…, p_j, …> with <…, p_j, p_i, p_j, …>
    - Delete: Remove a particular arc for p_j, changing the status from <…, p_i, p_j, p_k, …> to <…, p_i, p_k, …>
  - Event queue needs to support insert, deletion, and finding the upcoming event with the largest y-coordinate in O(log n) time.
    - Site events can be precomputed before we start sweeping since we already know the y-coordinates of the sites.
    - For Voronoi vertex events, we'll take consecutive triples p_i, p_j, p_k from *distinct points* on the beach line, and compute their circumcircle. The y-coordinate for the event is the y-coordinate of the bottom of the circle, since that's where the sweep line goes tangent to the circle.
  - And we have pointers/references going back and forth to figure who is responsible for what and why.
  - Finally, he's a sketch of how we handle events.

- Site events at site p_i.
    - Advance sweep line past p_i. Search for p_j whose arc is above p_i.
    - Insert-and-split p_i at that arc.
    - Create a new dangling edge lying on the bisector of p_i and p_j that follows the new breakpoints. It has no (finite) endpoints yet!
    - Remove vertex events for triples that no longer exist and add new ones. Note new triple p_i p_j p_i does not get an event because it has only two distinct sites.
- Voronoi vertex events at p_i, p_j, and p_k:
    - Remove p_j's arc from beach line status.
    - Create new Voronoi vertex at circumcenter of p_i, p_j, and p_k and join Voronoi edges for bisectors (p_i, p_j) and (p_j, p_k) at this vertex.
    - Create a new dangling edge for bisector between p_i and p_k. One endpoint is on the new vertex.
    - Delete events from triples involving p_j and add new events for triples involving p_i and p_k.
- Events take a constant number of data structure operations to find and process. Sweep line status has O(n) arcs and there are O(n) Voronoi vertex events in the queue at once so all operations take O(log n) time each. The total number of events is O(n) so the running time is O(n log n).