Voronoi Diagrams

Subhash Suri

October 17, 2019

1 Voronoi Diagrams

- Voronoi diagram is one of the most important geometric structures. It encodes proximity: *who is close to whom*?
- Suppose we have a set of points $P = \{p_1, p_2, \cdots, p_n\}$, which we call "sites".
- Given some other set of points (non-sites) $x_1, x_2, \ldots, x_m$, assign each of them to its closest (nearest) site. *Do this for all the (infinitely many) points of* R^2. What does this assignment look like?
- Assume Euclidean distances throughout although the concepts extend naturally to other L_p norms, as well as non-L_p norms.
- In other words, each site *pulls* all points of the plane closer to it than any other site. The region claimed by each site is called its *Voronoi cell*, and is defined as follows:

$$V(p_i) \quad = \quad \{q \in R^2 \mid d(p_i, q) \ \leq \ d(p_j, q), \ \forall \ j \neq i\}$$

  (The points in the interior of $V(p_i)$ are strictly closer, while those on the boundary can have multiple nearest neighbors.)
- The **Voronoi Diagram** of P is the *subdivision of the plane* induced by the voronoi cells $V(p_i)$, for $i = 1, 2, \ldots, n$.
- The vertices, edges, and faces of this subdivision are called *voronoi vertices, voronoi edges, and voronoi cells.*
- It is easy to see that the (interior of) cells of two points are disjoint, and the union of all the cells tiles the plane.
- Basic questions: What does each region look like? What geometric and combinatorial properties does it have? How efficiently can we compute or store (time/space) it? etc.

Many applications. Voronoi diagrams have a huge number of applications.

- Nearest neighbor queries. Compute VoD, and use point location.
- Computational Morphology (shape analysis). Medial axis of a shape (e.g. a polygon) set of center points of all maximal disks contained in the shape. Generalizing VoD to points and line segments, the medial axis can extracted from this generalized VoD.
- Center-based clustering. Clusters defined by a set C of centers, and each data point assigned to its closest cluster center. Under this assignments, points assigned to a cluster center c are exactly those inside c's voronoi cell.
- Nearest-neighbor interpolation and classification. Measurements or classification known at some finite set of points, and for any other point x, it's measurement is interpolated from its nearest neighbors.
- In general, locations of various facilities, retail stores, emergency services etc require similar considerations.
- Many phenomena in natural sciences also follow Voronoi rule: prediction of the growth of tree saplings over time, crystal growth from seeds, forest fires, etc.
- Used in mobility models in cities using Voronoi diagrams of buildings (Mobicom paper).

Basic Properties and Definitions.

- Examples. Voronoi diagram of 2 points. Voronoi diagram of 3 points.
- Voronoi cells are (possibly unbounded) convex polygons. To see this, observe that

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

  where $h(p_i, p_j)$ is the closed half plane defined be the bisector of p_i, p_j.
- Each $V(p_i)$ can be computed in $O(n \log n)$ time (halfplane intersection algorithm), and has $n - 1$ edges in the worst case.
- Assume general position for points. Then, no four points are co-circular, and three points determine a unique circle. So, each vertex of VoD has degree 3.
- *Empty Circle Properties of Voronoi Edges and Vertices.*

  - Each point x on an edge of VoD has two nearest neighbors, p_i, p_j, so there is a circle centered at x passing through p_i, p_j and containing no other sites.
  - A voronoi vertex is on the boundary of three voronoi cells V_i, V_j, V_k, and so is equidistant to p_i, p_j, p_k. Therefore the circle centered at x and passing through p_i, p_j, p_k is empty.

- Convex Hull. A cell is unbounded if only if its site lies on the convex hull—unboundedness means that site is closest site from a point at infinity. Thus, given VoD we can extract CH in linear time.

- Size. Any individual cell can have size $\Omega(n)$ yet the overall size of VoD is only $O(n)$. This follows from Euler's formula. There are exactly n faces, each vertex has degree 3, which means that the number of edges and vertices is also $O(n)$.

- **Voronoi Size Lemma:** The number of vertices in $V(P)$ of n points is at most $2n$, and the number of edges at most $3n$.

- Proof. Use Euler's formula for planar graphs.
  - The subdivision has n faces, and each vertex has degree 3, assuming non-degeneracy.
  - But we also need to handle the half-lines, so we add a "vertex at infinity" as terminus of all unbounded edges.
  - Let V be the number of Voronoi vertices (including the vertex at infinity), E Voronoi edges, and $F = n$ Voronoi faces.
  - Then, by Euler's formula, we have $V - E + F = 2$.
  - The sum of vertex degrees is $2E$. Since each vertex has degree at least 3, we have $2E \geq 3V$, which gives $V \leq \frac{2}{3}E$.
  - Plugging this into Euler's formula, we get $\frac{2}{3}E - E + F \geq 2$. Put in $F = n$, and we get $E \leq 3n - 6$.

- In higher dimensions, the complexity grows exponentially with d. The worst-case complexity of VoD in d dim is $O(n^{\lceil d/2 \rceil})$.

Supplementary Proofs.

1. A point q is a vertex of $V(P)$ if and only the largest empty circle $C(q)$ with q as center passes through 3 or more sites of P.

2. The bisector of p_i and p_j forms an edge of $V(P)$ if and only if there is a point q on the bisector such that the largest empty circle $C(q)$ contains both p_i and p_j, and no other site.

3. **Non-degeneracy Condition:** Assume no 4 cocircular points, and no 3 collinear.

4. **Lemma:** A cell $V(p)$ is unbounded iff p is on the Convex Hull of P.

5. Proof.

    - If p on the hull, consider a ray normal to the supporting line of CH through p. For all points on this ray, the closest point is p, and so the entire ray lies inside $V(p)$.
    - Conversely, if p is not on the CH, it must lie inside a triangle formed by 3 points, say, a, b, c. In this case, no point sufficiently far away is closer to p than one of a, b, c. Thus, $V(p)$ is bounded.

Computing Voronoi Diagrams

- Since each $V(p_i)$ is the common intersection of $n-1$ halfplanes, it can be computed in $O(n \log n)$ time. This gives an $O(n^2 \log n)$ algorithm for he VoD. Historically, this was the state of art before CG came along—a number of $O(n^2)$ algorithms.

- The first breakthrough in CG was an $O(n \log n)$ time divide-and-conquer algorithm for VoD in the plane. However, these D-and-C algorithms are complex, difficult to understand and implement.

- Later a much more elegant and simpler algorithm was discovered by Fortune, based on plane sweep. Although plane-sweep, it is much more sophisticated and trickier than any we have seen so far. (Another, even simpler algorithm is due to Guibas-Knuth-Sharir, using a randomized incremental construction (in the style of our point location algorithm). We will discuss it in DT.)

- It is first worth mentioning some obvious difficulties with a plane sweep approach to VoD, which may explain why it was not discovered sooner.

- A plane sweep algorithm needs to "predict" (determine when it will occur) events. Now as we sweep the plane, suppose we have constructed the VoD of all the sites we have encountered so far. The problem is that a site not-yet-encoutered (lying ahead of the plane sweep) may generate a Voronoi vertex that *lies behind the sweep line*! Then, how could the sweep line know about this Voronoi vertex before it reaches that site? By the time, it reach the site, it's too late. These *unanticipated events* are the reason why plane sweep is challenging for computing VoD.

2 Fortune's Sweep Line Algorithm

- Input is a set $P \subseteq R^2$ of n sites.
- Algorithm sweeps a horizontal line, from bottom ($y = -\infty$) to top ($y = +\infty$), and constructs the VoD along the way.
- As a visualization aid, imagine that each site has a distinct *color*.
- **Goal:** color each point of the 2D plane with the color of its nearest site.
- We simulate the coloring using the sweep line paradigm. A *stumbling block* is that the nearest site of x may not have been encountered by the sweepline yet! More precisely, at any position of the sweepline L, the Voronoi regions intersecting L depend on points lying on *both sides* of L, but the sweepline has not encountered those lying above it yet, so how can it correctly build those Voronoi regions?
- Therefore, we maintain a *weaker* invariant, where only those points are guaranteed to be colored whose nearest site has been discovered with certainty. (In other words, the algorithm is guaranteed to have constructed those portions of the Voronoi diagram that cannot change by points above L.)
- **Invariant:** For any point x below the sweepline L, we have correctly colored x if the nearest neighbor of x is known to lie below L.
- **Certification of Coloring:** How can we guarantee that the NN of x cannot be above the line L? *Answer:* If the distance $d(x, L) > d(x, p)$, for some site p lying below L, then no site above L can possibly be the NN of x.
- This motivates the use of a *parabola* as our certificate. A **parabola** is the locus of points with the property:

$$\Pi(p, L) \;=\; \{x \mid d(x, p) = d(x, L)\},$$

  where p is the *focus* and L the *directrix*. That is, if x is a below $\Pi(p, L)$ and q is any site above the line L, then necessarily $d(x, q) > d(x, p)$, and so all points below $\Pi(p, L)$ can be safely colored.
- Figure.
- It is *important* to realze that x's color is not necessarily the same as p's color, but it must have the color of some point below L (and so already encountered).

- Each site below L has its own parabola, each with its own growth depending on the distance of its site to L. As the sweepline moves, $\Pi(p, L)$ expands (opens up), because $d(x, L)$ increases. We are only interested in the boundary (*upper envelope*) of these parabolas.

- In particular, the Parabolic Front $\Pi(L)$, the **front boundary**, of these evolving parabolas. is the *upper envelope* of parabolas, one for each site that has been processed by the sweep line:

$$\bigcup_{p \text{ below } L} \Pi(p, L)$$

- Show 3 parabolas, their wavefront, and bisectors.

- **Lemma.** The paranolic front is an x-monotone curve made up of parabolic arcs, whose breakpoints trace out Voronoi edges of the final diagram.

Bisectors:

- Let p_i, p_j be two consecutive sites in the ordered list $\Pi(L)$. Then, the intersection of their parabolas $\Pi(p_i, L)$ and $\Pi(p_j, L)$ traces out the bisector B_{ij}, as L moves.

- **Proof.** The intersection point is equi-distant to p_i and p_j.

High Level Scheme and Event Processing

- Fortune's algorithm "simulates" the evolution of the parabolic front as the sweep line moves across the sites. (Algorithm does not maintain actual parabolas; only the ordered multi-list of sites, corresponding to the parabolic front, is maintained. The parabolas are for conceptual understanding.)

- Specifically, our *sweep line status data structure* maintains the *parabolic front* simply as a *multi-list* of sites, in the order those sites' parabolas appear on the upper envelope. (This list is maintained as balanced binary tree, for efficient search, insertion and deletion.)

- It's a multi-list because a single site can contribute multiple non-adjacent arcs to the front.

- **Dynamic Updates:** The parabolas change "continuously" with the shifting sweepline, but we are only interested in "events" that cause changes in the "structure" of the parabolic wavefront.

- What are those events? How can we schedule them before they occur? What data structures do we need to support this simulation?
- It turns out that there are precisely two types of events. Type I, where a new parabola is born, and Type 2 where a parabola disappears.

Event Type I: Site Evens (Birth of a Parabola)

- The first type of event is when the sweepline L meets a new site: a new parabola, corresponding to the new site, is born and must be inserted into the wavefront.
- Figure.
- The Before List is A, B, C, and the After List is A, B, p, B, C.
- When a new site p is encountered, an "infinitesimally thin" parabola (essentially a half-ray, directed downward) is added. This is the locus of points whose distance to p and L is equal.
- This is the **only** way a new parabola is inserted. See the book for a formal proof.
- How do we *detect* and process this event?
- Type I event occurs at sites, so we just keep the y-sorted list of sites.
- Total number of events of Type I processed is clearly n. Each takes $O(\log n)$ time.

Event Type II: Voronoi Vertex Events (Death of a Parabola)

- The second type of event occurs when a parabolic arc on the wavefront gets "swallowed up" by its two neighboring arcs.
- Figure.
- In this case, the parabolic arc corresponding to a site p needs to be deleted from the wavefront. The parabolic arc of p becomes swollen up by the neighbors.
- Before List is A, B, C and the After List is A, C.
- This is the **only** way a parabola can disappear.
- **Event Detetion.** The key geometric property to realize is that when this event occurs, we get

$$d(x, L) = d(x, A) = d(x, B) = d(x, C)$$

- That is, the 3 parabolas of A, B, C meet at a point x whose distance to A, B, C is also the same as its distance to L.
- Thus, the **circle** defined by A, B, C, whose circumcenter is at x, is tangent to L.
- We take all consecutive **triples** of sites in $\Pi(L)$, and compute the highest point (by y-coordinate) of their circumcircle. Put this value into the Event Queue.

Plane Sweep Data Structures

- [**Event Queues:**] Maintained a priority queue containing
  1. y coordinates of all the sites, and
  2. Highest y-coordinate of the circles defined by consecutive triples in $\Pi(L)$.
- Recall that a single site p can contribute multiple parabolic arcs in $\Pi(L)$. Example list

$$p, A, p, B, p, C, p, \ldots, p, Z, p$$

- Not all triples form a valid circle: the triple must include 3 distinct sites, and the highest point must be past the y coordinates of the three sites involved.
- So, for instance, in this example, we include triple A, p, B but not p, C, p.
- [**Sweep Line Status: Parabolic Front**] $\Pi(L)$ is the ordered (multi) list of sites corresponding to parabolic front.
- **Important:** The algorithm does not store the parabolic arcs of the front; they are shown only for explanation.

Pseudo Code:

1. Sort all sites by y-coordinates. Let p_{min} be the lowest site.
2. Set L to $y = y_{min}$, and $\Pi(L) = \{p_{min}\}$.
3. Initialize the event queue Q with y-coordinates of remaining sites. Each entry is a tuple (y, p), the y coordinate and the associated point.
4. While Q non-empty, do
   Let $(y, p) = \text{ExtractMin}(Q)$.
5. **Case I: (p is a site event)**

- $q = \text{Locate}(p_x, \Pi(L))$.
- Let $r = pred(q)$ and $t = succ(q)$.
- Delete circle event (r, q, t) from Q, if present.
- Modify $\Pi(L)$ to $(\cdots, r, q, p, q, t, \cdots)$
- Add circle events (r, q, p) and (p, q, t) to Q.

6. **Case II: (p is a circle event)**. (The arc corresponding to p disappears.)
   - Let $r = pred(p)$ and $t = succ(p)$.
   - Delete p from $\Pi(L)$.
   - Since p disappears, modify circle events by possibly adding (r', r, t) and (r, t, t'), and deleting (r', r, p) and (p, t, t').

Operations and Geometric Primitives:

1. Priority Queue.
   - y coordinates of the sites are given, so easy
   - Circle events require computing the highest point of the circle determined by triple (p, q, r).
   - $O(1)$ time operation. [Homework exercise.]
2. Operation Locate.
   - Need to locate the place where the new parabola for p will be inserted.
   - We perform a binary search in the parabolic front, but the problem is that while a site has unique x-coordinate, it can contribute multiple arcs, so we can't decide which arc contains p.
   - The key is to binary search not using x coordinates of sites, but rather the x coordinates of the intersection points between adjacent parabolic arcs.
3. Each priority queue and locate operation takes $O(\log n)$ time.
4. Number of operations is $O(n)$. Why?
5. Type I events are clearly n. Type II events each produce a Voronoi vertex, so also $O(n)$.
6. Algorithm runs in $O(n \log n)$ worst-case time.

7. The VoD is stored as a planar subdivision, say, using DCEL. There is one technical difficulty: how to store unbounded edges. We can either enclose the entire input into a large bounding box, so that it encloses all Voronoi vertices, or use an imaginary point at infinity.