6.854J / 18.415J Advanced Algorithms
Fall 2008

18.415/6.854 Advanced Algorithms December 3, 2008

Lecture 23

Lecturer: Michel X. Goemans

1 Voronoi Diagrams

1.1 Introduction

Suppose we are given a set P of points in the Euclidean plane, and we are interested in the problem of, given a point x, find the closest point of P to x. One approach to this problem is to divide the plane into regions associated with each $p_i \in P$ for which x is closest to p_i. Finding these regions in two dimensions is the problem of constructing the Voronoi Diagram. One application of this structure is to compute the mimumum spanning tree of a complete graph of n vertices in the Euclidean plane in time $O(n \log n)$.

1.2 Definitions

We will focus on the two-dimensional case. We are given a set

$$P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$$

and we want to partition the plane into regions which correspond to points which are closest to a specific point.

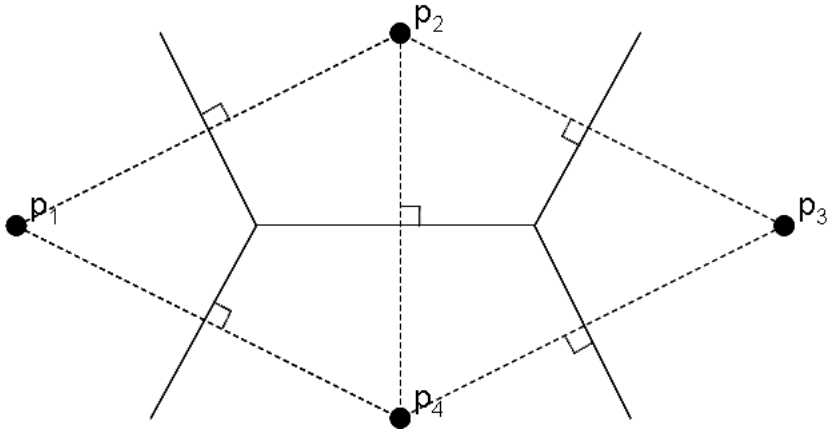

Figure 1: Voronoi Diagram (solid lines) for four points $p_1,\ p_2,\ p_3,\ p_4$.

Definition 1 (Voronoi Cell) *Given a set of points in* $\mathbb{R}^2$, $P = \{p_1, p_2, \ldots, p_n\} \subseteq \mathbb{R}^2$, *a Voronoi Cell* $V(p_i)$ *is defined by:*

$$V(p_i) = \{x : d(p_i, x) < d(p_j, x) \quad \forall j \neq i\}.$$

Another way to define a Voronoi Cell is by defining $h(p_i, p_j)$ to be the halfplane containing p_i defined by the bisector of p_i and p_j. A cell is then defined as:

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

This impies that every cell is convex and is a (convex) polygonal region with at most $n - 1$ sides.

Definition 2 (Voronoi Diagram) *A Voronoi Diagram is a collection of Voronoi cells that covers* $\mathbb{R}^2$.

1.3 Motivation

Why is a Voronoi Diagram useful? If the points represent firestations, the Voronoi cells represent the partition of the plane into regiosn which are closer to each firestation. More generally, given a point in a plane, it is useful to know the point from a set of points that is closest to it. Of course, this also requires a data structure to be able to answer the *point location problem* of, given x, finding the Voronoi cell that contains it. We will only learn how to construct the Voronoi diagram, not how to build a query datastructure for it. .

Having such a diagram is useful for many problems. For example, a Voronoi diagram allows computation of the Euclidian minimum spanning tree on a set of points in $O(n \log n)$ time, see the problem set.

1.4 Properties

The Voronoi cells are all disjoint and their closures cover the entire plane. The Voronoi diagram will consist of edges (possibly semi-infinite, extending to infinity) and vertices where 3 or more of these edges meet; these vertices will be equidistant to 3 or more points of P. One can characterize the vertices and the edges in the following way:

Lemma 1

1. *A point* $q \in \mathbb{R}^2$ *is a vertex of a Voronoi Diagram* $\iff$ *there exists an empty circle (i.e. its interior is empty) centered at* q *having at least 3 points of* P *on its boundary.*

2. *Part of the bisector between* p_i *and* p_j *is an edge of the Voronoi diagram* $\iff$ *there exists an empty circle centered at a point* q *having precisely* p_i *and* p_j *(and no other point) on its boundary.*

We look now at how 'complex' a Voronoi diagram can be. We know that each cell is delimited by at most $n-1$ sides (edges), but in the lemma below, we show that collectively all cells do not have too many edges and vertices.

Lemma 2 *For a Voronoi diagram with n points, the following relations hold:*

- *The number of vertices of a Voronoi diagram is* $n_v \leq 2n-5$.
- *The number of edges in any Voronoi diagram is* $n_e \leq 3n-6$.

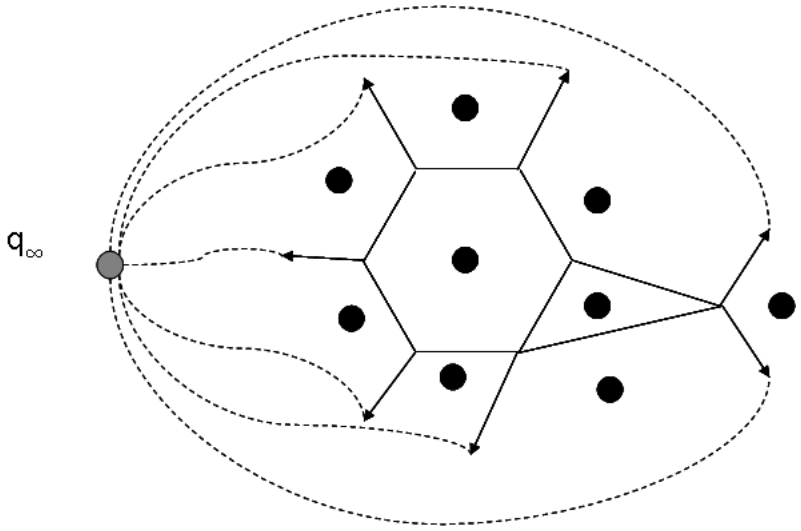

Figure 2: To prove Lemmma 2 we add a point q_∞ to the Voronoi Diagram (solid lines), and connect all of the infinite edges to this point (shown in dotted lines).

Proof: We can view the Voronoi diagram as a planar graph, G, with some edges extending out to infinity. We add a point at infinity q_∞ representing 'infinity' and connect edges that extend to infinity to this point as shown in Figure 2. Note that the resulting graph G' is still planar.

The number of vertices in G' is n_v+1; the number of edges is n_e, and the number of faces is n. By Euler's formula, we have

$$n_v + 1 - n_e + n = 2.$$

Since we know that vertices will have at least 3 edges incident to them, we obtain, by summing the degrees over all vertices, that:

$$\sum_{\text{vertices } v} d(v) = 2n_e \geq 3(n_v+1).$$

Combining this with Euler's formula, we get:

$$2(n_v + 1) + 2n \geq 4 + 3(n_v + 1)$$

or $2n - 5 \geq n_v$. Using this in Euler's formula, we now get

$$n_e = n_v - 1 + n \leq 3n - 6.$$

□

2 Computation of Voronoi Diagrams

2.1 Introduction

There are two primary algorithms we want to introduce. Both of these will be shown to compute the Voronoi diagram in time $O(n \log n)$. First, we can reduce the computation of the Voronoi diagram to that of a convex hull in $\mathbb{R}^3$, which is computable in time $O(n \log n)$; this is our first algorithm. Secondly, we will review the sweep line algorithm of Fortune [1].

2.2 Convex Hull

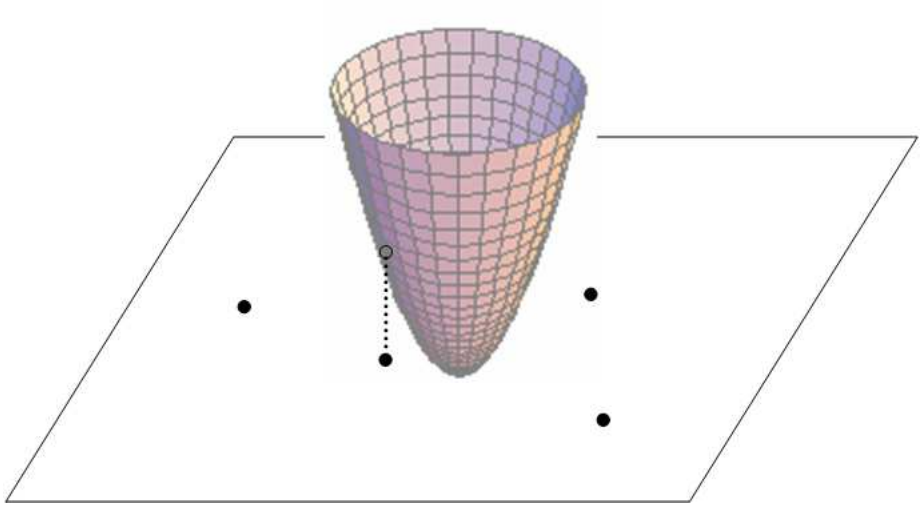

Figure 3: Projection of a point onto a paraboloid in $\mathbb{R}^3$. To use the convex hull to compute the Voronoi diagram, this projection is done for all points in the set of points for which we want to compute the Voronoi diagram.

Suppose we have a set $P \subseteq \mathbb{R}^2$ and we want to compute the corresponding Voronoi diagram. Let us consider the set $P' = \{(x_i, y_i, x_i^2 + y_i^2) : (x_i, y_i) \in P\}$. This projection onto a parabola is shown in Figure 3.

Consider the set of planes tangent to each point in P'. The intersection of the upper half spaces of these planes gives a polyhedral set Q whose projection back to $\mathbb{R}^2$ gives the Voronoi diagram in the following sense: the projection of the facets (resp. edges, vertices) of Q gives the Vornoi cells (resp. edges, vertices) of the Voronoi diagram. This computation can be done in $O(n \log n)$ time since this calculation is the geometric dual of the convex hull computation.

If, instead, we were to compute the convex hull of P' (rather than the halfspaces tangent to the paraboloid at P') and project it back to $\mathbb{R}^2$, we would obtain a straight-line drawing on P (dual to the Voronoi diagram) known as the *Delaunay Triangulation*, see problem set.

2.3 Sweep Line Algorithm

The idea of a sweep line algorithm is to advance a line (in 2D) or a plane (in 3D) down through space, processing events as they occur. We will construct the Voronoi diagram as we sweep the line from top to bottom, and at any instance we will only have needed to consider points at or above the sweep line.

We cannot construct the entire diagram above the sweep line, but we can construct pieces of it. If we look at a single point above the line, p_i, for some points, they will assuredly be closer to it than to any points below the sweep line. This forms a parabola $C(p_i)$ defined by the points equidistant from the point and the sweep line. We can find the parabola associated with each of the points. For any point that is above some parabola, we can correctly assign it to its Voronoi Cell.

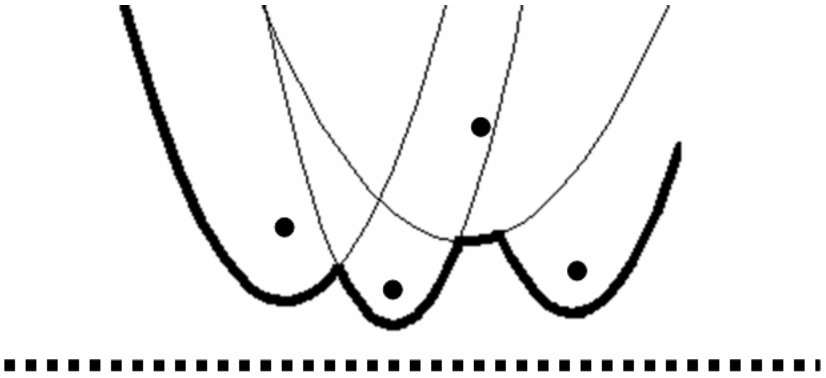

Figure 4: A set of parabolae $C(p_i)$ associated with four points p_i. Parabolae are denoted with thin lines, the beach line with a thick line, and the associated sweep line with a thick dashed line.

Definition 3 (Beach line) *We define a Beach Line as the lower envelope of all parabolae $C(p_i)$ for all points above the sweep line. A beach line is shown in Figure 4.*

Definition 4 (Breakpoint) *A breakpoint q is a point on the beach line that belongs to at least two parabolae.*

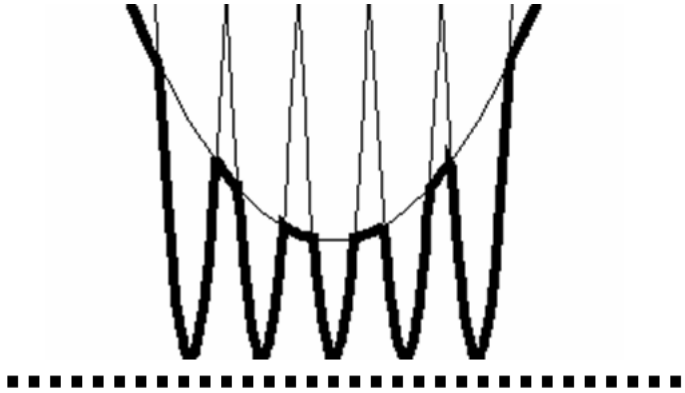

Figure 5: Sample beach line illustrating multiple break points originating from the same parabola

The beach line is a series of segments of parabolae. A breakpoint q corresponding to the parabolas $C(p_i)$ ad $C(p_q)$ must be equidistant from both p_i and p_j since we know that $d(q, p_i) = d(q, sweep) = d(q, p_j)$. Furthermore, no other point of P is closer to q. Thus, by Lemma 1, q is part of an edge of the Voronoi diagram, and is part of the bisector between p_i, p_j. An example is shown in Figure 6.

We will keep track of which p_i the breakpoints are associated with in order. Note a beach line could have several segments from the same parabola, as illustrated in Figure 5.

2.3.1 Events

As we sweep the line, we are not going to keep track of the precise location of the beach line (as it constantly changes) but we will just keep track of the points p_i corresponding to the parabola segments of the beach line from left to right. Several *events* can happen that modify this sequence of points p_i.

1. A 'Site Event' occurs when the sweep line goes through a new point p_l. This results in addition of an arbitrarily narrow parabola around p_l to the beach line. A sample site event is shown in Figure 7. If p_l intersects the parabola associated with p_j, we could write the change in the sequence of points as:

   $$p_i \; p_j \; p_k \rightarrow p_i \; p_j \; p_l \; p_j \; p_k$$

   Note we insert exactly one segment per site event, so there are n in total. Notice that each such addition increases the number of segments by 2, as shown above.

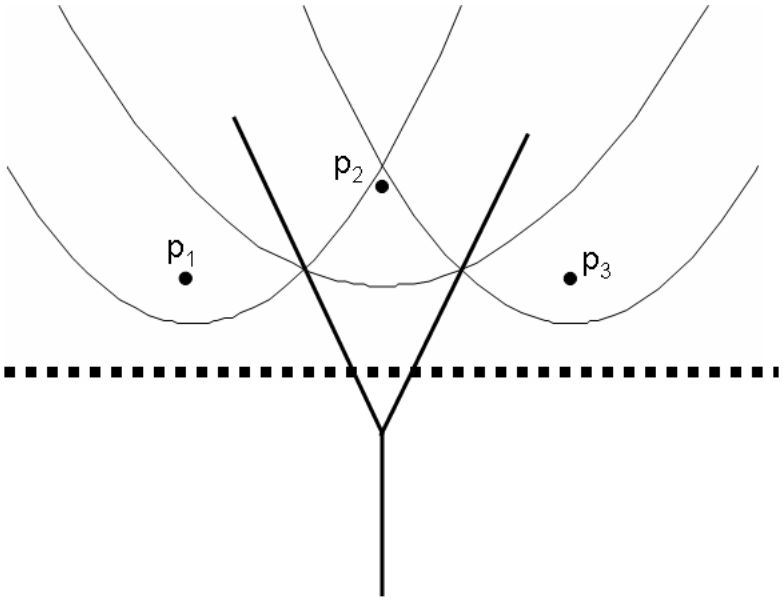

Figure 6: Illustration of points q on an edge of a Voronoi diagram as constructed by a moving sweep line.

We'll see that this is the only way of creating a new segment in the beach line, so this implies that the total number of segments in the beach line is at most $2n - 1$ (1 segment for the first site event, and 2 more for each subsequent site event).

2. A 'Circle Event' occurs when lowering the beach line causes a segment to disappear from the beach line. This boundary case is illustrated in Figure 8, which can be compared to Figure 6 to show the efffect of a moving sweep line.

   When a segment disappears, we have discovered a new vertex in the Voronoi diagram. Indeed, when a circle event occurs, we must have the three closest points equidistant to the vertex, and thus we have a vertex by Lemma 1.

   The center of the circle is determined by p_1, p_2 and p_3 (corresponding to 3 consecutive segments on the beach line), and the circle event will happen when the sweep line is tangent to the circle (below it). When a circle event happens, the beach line is modified in the following way:

   $$p_1\ p_2\ p_3 \rightarrow p_1\ p_3$$

   .

Claim 3 *The only way for the beach line to change is through a site event or a circle event. In other words, these are the only ways to create and remove segments.*

We will not formally prove this – this is intuitive.

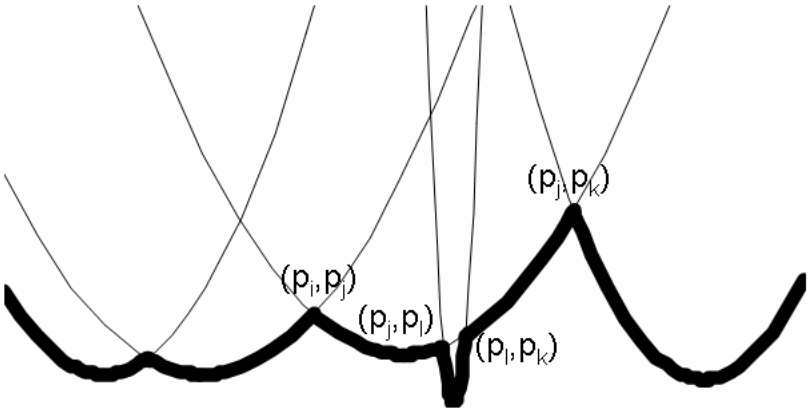

Figure 7: Site event. Parabolae shown with thin lines and the beach line shown as a thick line.

2.3.2 Data Structures

In order to construct a diagram, we will describe three data structures:

1. **Event queue**:

   Construct a priority queue containing events. The key of an event is its y-coordinate. For a site event the y-coordinate is the y-coordinate of the associated point. For a circle event, this is the position of the sweep line which is (lower) tangent to the circle.

   We first insert the n site events into the priority queue, as we know the y coordinate of all the points. Consider moving the line down and processing events as they occur. Circle events are defined by looking at three consecutive segments of the beach line. Every time we introduce a new segment in the beach line, as happens in a site event, we potentially create two new circle events (potentially, since three consecutive segments create a circle event only if the 3 points are distinct). We may also need to delete some circle events.

   Let us consider the addition shown in Figure 7. We will have removed the potential circle event $p_i\ p_j\ p_k$ and added potential circle events $p_i\ p_j\ p_l$ and $p_j\ p_l\ p_k$. Note that the deleted event can be thought of as a fake event because it was removed before it really happened and was processsed. Still such a circle event was added to the event queue and then removed. There is at most one deleted (fake) circle event for each site event processed. Notice that the number of real circle events is equal to the number of vertices of the Voronoi diagram, $n_v \leq 2n - 5$.

   Any circle event that is processed is real, and leads to a segment of the beach line disappearing. In terms of Figures 6 and 8, we would take $p_1\ p_2\ p_3$ to $p_1\ p_3$.

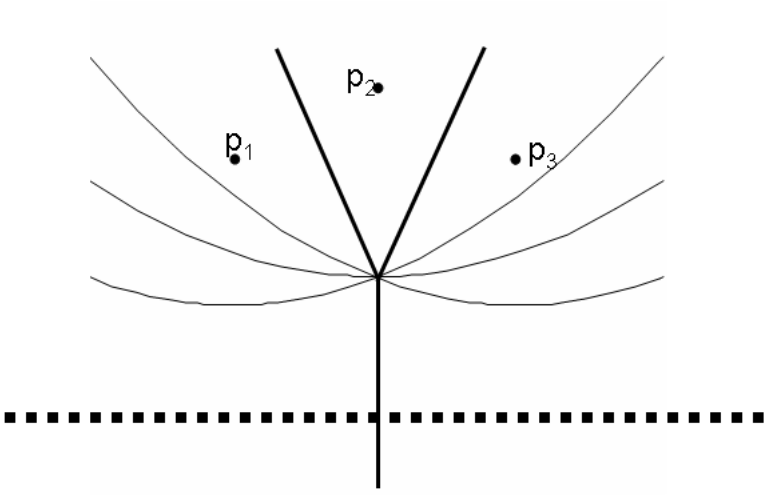

Figure 8: Circle event. Parabolae shown with thin lines, Voronoi diagram with thick lines, and the sweep line with a thick dashed line.

In general, we can write this as "Go from $p_i\ p_j\ p_k\ p_l\ p_m$ to $p_i\ p_j\ p_l\ p_m$". We may need to delete up to two circle events corresponding to the lost segment and add two new events, corresponding to the new order. In this example, we are deleting circle events $p_i\ p_j\ p_k$ and $p_k\ p_l\ p_m$ and adding $p_i\ p_j\ p_l$ and $p_j\ p_l\ p_m$ (or a subset of them if some of the indices are equal). We are always adding and deleting a constant number of events (for each site event and real circle event), thus the total number of additions and deletions to the priority queue will be linear. Since we must process $O(n)$ events corresponding to $O(n)$ priority queue operations, the total runtime will be $O(n \log n)$.

2. **Beach line encoding**:

   We keep track of the points corresponding to the parabola segments constituting the beach line and the breakpoints $p_i\ p_j$ by creating a binary search tree in which points are leaves and internal nodes are breakpoints.

   Note that this is an extension of the standard binary search tree because we have two different types of nodes (parabola segments and breakpoints). This prevents us from directly using a splay tree, since the splay action permutes the leaves and branches of the tree. One way to deal with this is to forget about parabola segments, and keep track of the breakpoints (as pairs of points), keyed from left to right.

   When a site event occurs, we need to be able to locate the x value in the beach line. To use a binary search tree, we thus need to be able to perform binary comparisons to determine if the desired x value is to the left or right

of a breakpoint. Given a breakpoint as an ordered pair (p_i, p_j) and a sweep line, we can easily compute the x position of the breakpoint and decide if we must move to the right or to the left. In a circle event, we have three parabola segments and must remove the middle one. This is a delete operation. Thus there are a constant number of BST operations per circle or site event. Using a BST with amortized cost $O(\log n)$ time per operation, maintaining the beach line is therefore $O(n \log n)$ time.

3. **Voronoi Diagram**:

   Let us replace each edge (shared by 2 cells) of the Voronoi diagram with two corresponding directed half-edges which are 'twin' to each other. Each half edge corresponds to one of the two cells, and each is oriented counterclockwise (with respect to its cell). For each half-edge, we define pointers:

   - to its twin,
   - to the next half-edge on the cell,
   - to the previous half-edge on the cell.

   From a given vertex we can follow the half-edges around a cell; by calling twin, we can move between cells and we can for example enumerate all half-edges incident to a vertex.

   Let us consider how to modify this structure upon processing a site (Figure 7) and circle (Figure 8) events. In a site event, the two new breakpoints are equidistant from p_j and p_k, and are part of an edge of the Voronoi diagram. This will create two new half-edges. In a circle event we link the half edges that meet to construct the diagram. Thus there are a linear number of operations on this data structure as wll.

In summary, the first structure requires a linear number of operations each taking $O(\log n)$ time. Similarly, for the second data structure, with a balanced BST. The last one requires constant time per event, for a linear number of events. Hence the total time to construct a Voronoi diagram is $O(n \log n)$.

We can show this is optimal because the Voronoi diagram of the set of points given by $P = \{(x_i, \pm 1)\}$ solves the problem of sorting P, hence the diagram must take at least $O(n \log n)$ time to sort.

Note we use ± 1 since we have assumed throughout this that we are not in the purely degenerate case in which all points are colinear; one can show that this is indeed the only case in which the Voronoi diagram has infinite lines and no vertices.

References

[1] S Fortune. A sweepline algorithm for voronoi diagrams. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 313–322, New York, NY, USA, 1986. ACM.