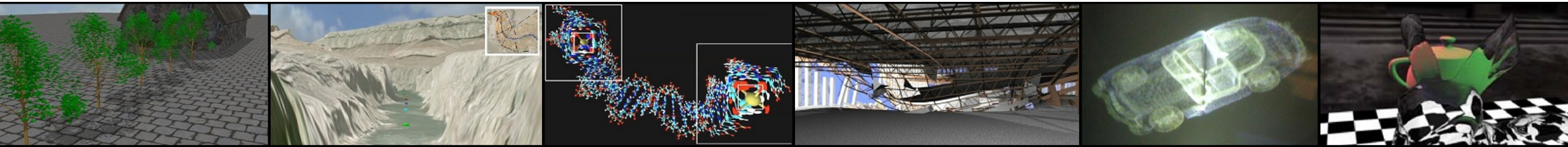
COT 4521: Introduction to Computational Geometry

Voronoi Diagram

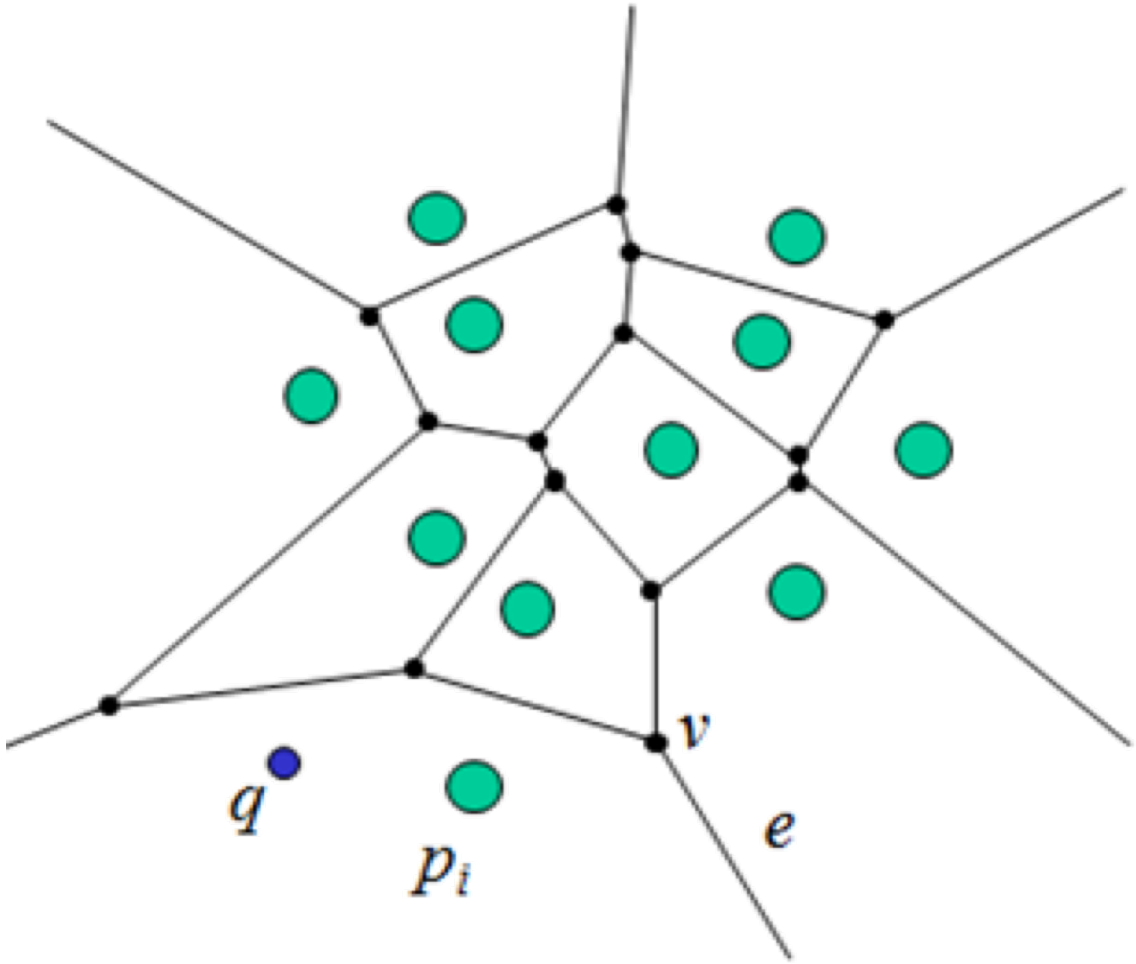
Paul Rosen
Assistant Professor
University of South Florida

Some slides from Valentina Korzhova



VORONOI DIAGRAMS

- p_i : SITE POINTS
- q : FREE POINT
- e :VORONOI EDGE
- v :VORONOI VERTEX



Definition of Voronoi Diagram

- Demo – http://alexbeutel.com/webgl/voronoi.html

Voronoi Diagrams: Applications

- Fire observation Towers
- Facility Planning
- Path Planning
- Crystallography have long been used to simulate crystal growth

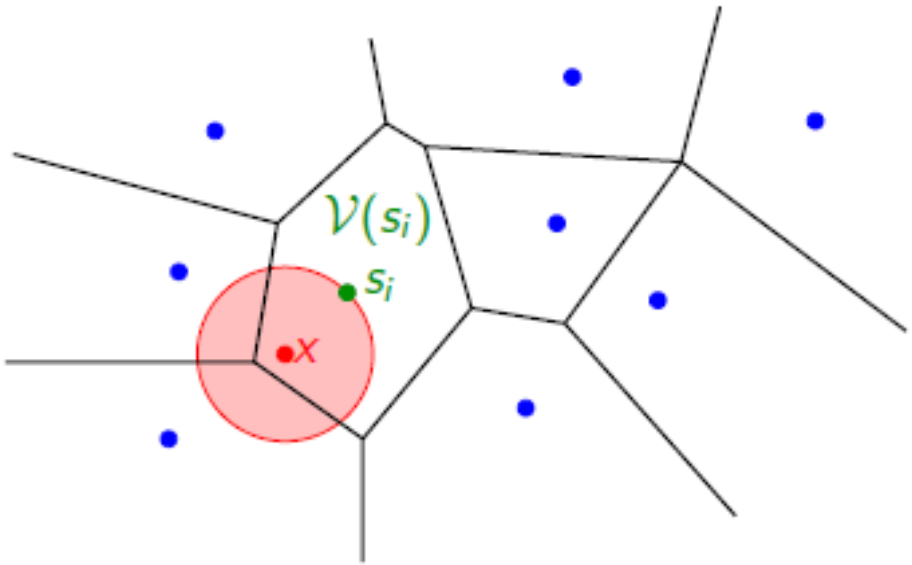
Definition of Voronoi Diagram

- Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n distinct points (sites) in the plane.
- The Voronoi diagram of P is the subdivision of the plane into n cells, one for each site.
- A point q lies in the cell corresponding to a site $p_i \in P$ iff

$$for\ each\ p_i \in P, j \neq i$$
$$Euclid\_dist(q, p_i) \leq Euclid\_dist(q, p_j)$$

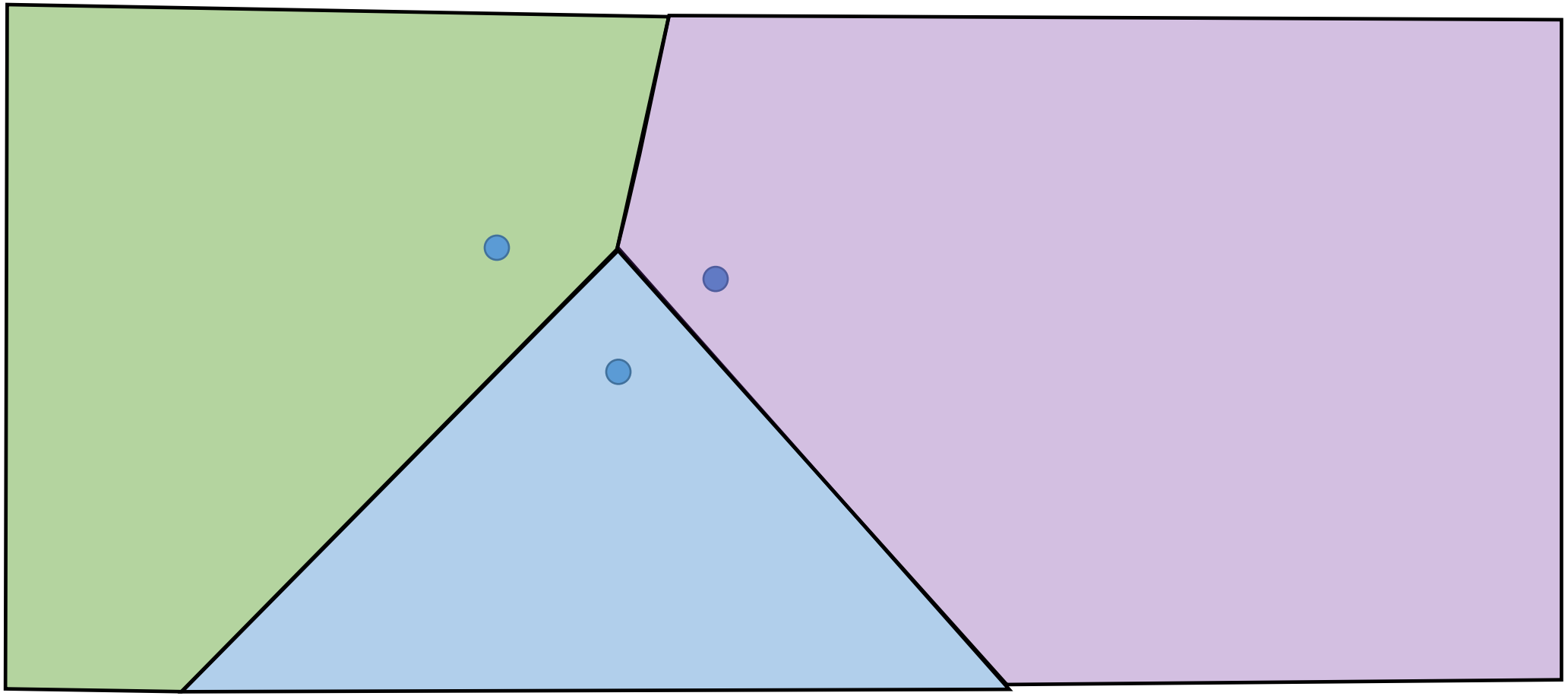
PROPERTIES

- VORONOI CELL
  $\forall x \in V(s_i)$, the disk through s_i centered at x contains no other site that s_i.



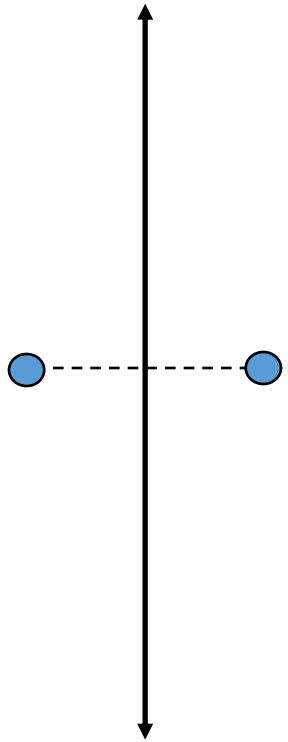
Definition of Voronoi Diagram

- $V(p_i) = \cap_{i \neq j} H(p_i, p_j)$,
  - where $H(p_i, p_j)$ is the closed halfplane



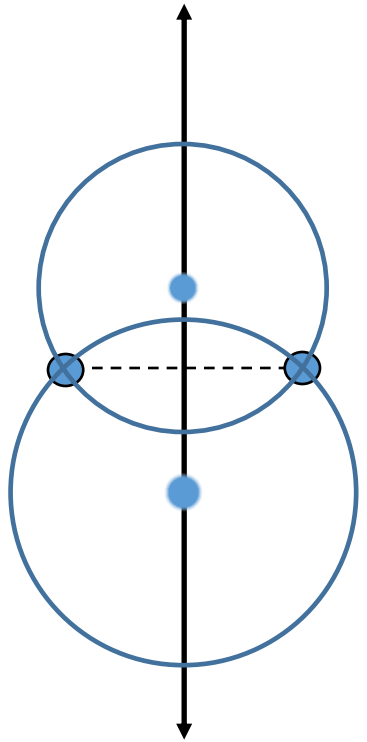
Two sites form a perpendicular bisector

- Voronoi Diagram is a line that extends infinitely in both directions, and the two half planes on either side.



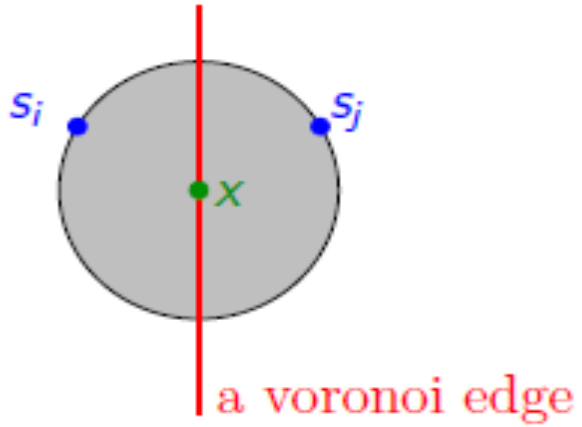
Two sites form a perpendicular bisector

- Bisector is the collection of all circles whose perimeters touch both sites



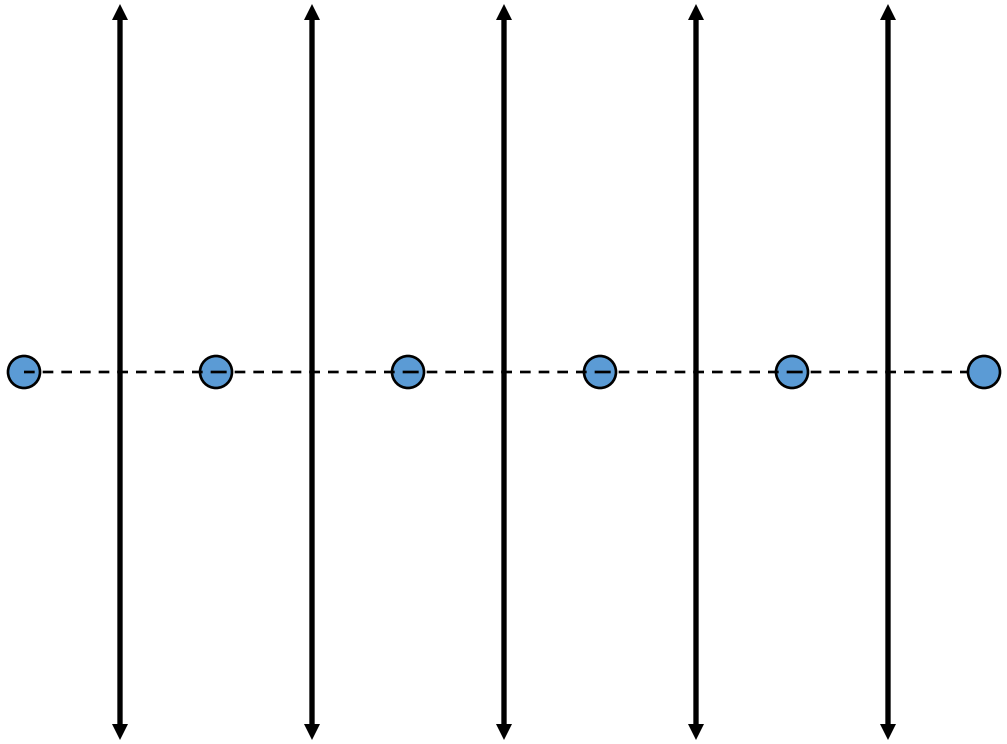
PROPERTIES

- A VORONOI EDGE:
  - A point x on a Voronoi edge is equidistant to two nearest sites s_i and s_j.
  - Hence the circle centered at x through s_i and s_j contains no site in its interior.



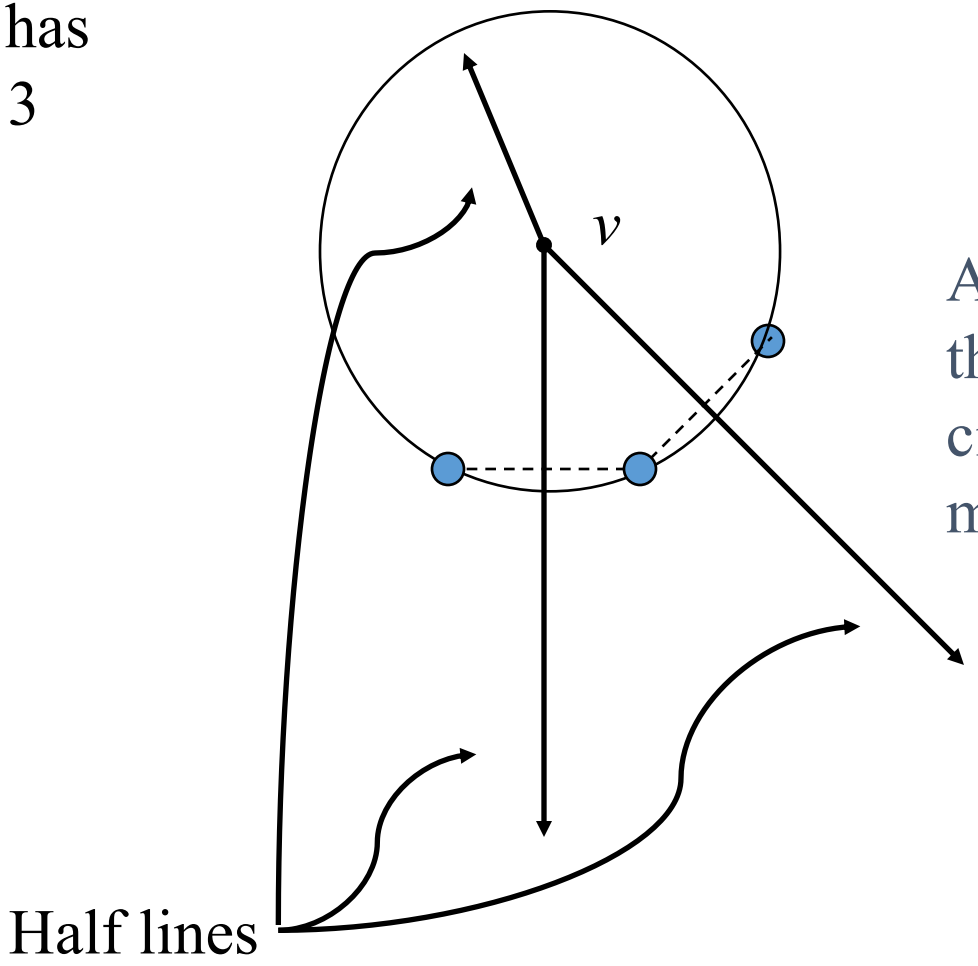
Collinear sites

- Collinear sites form a series of parallel lines



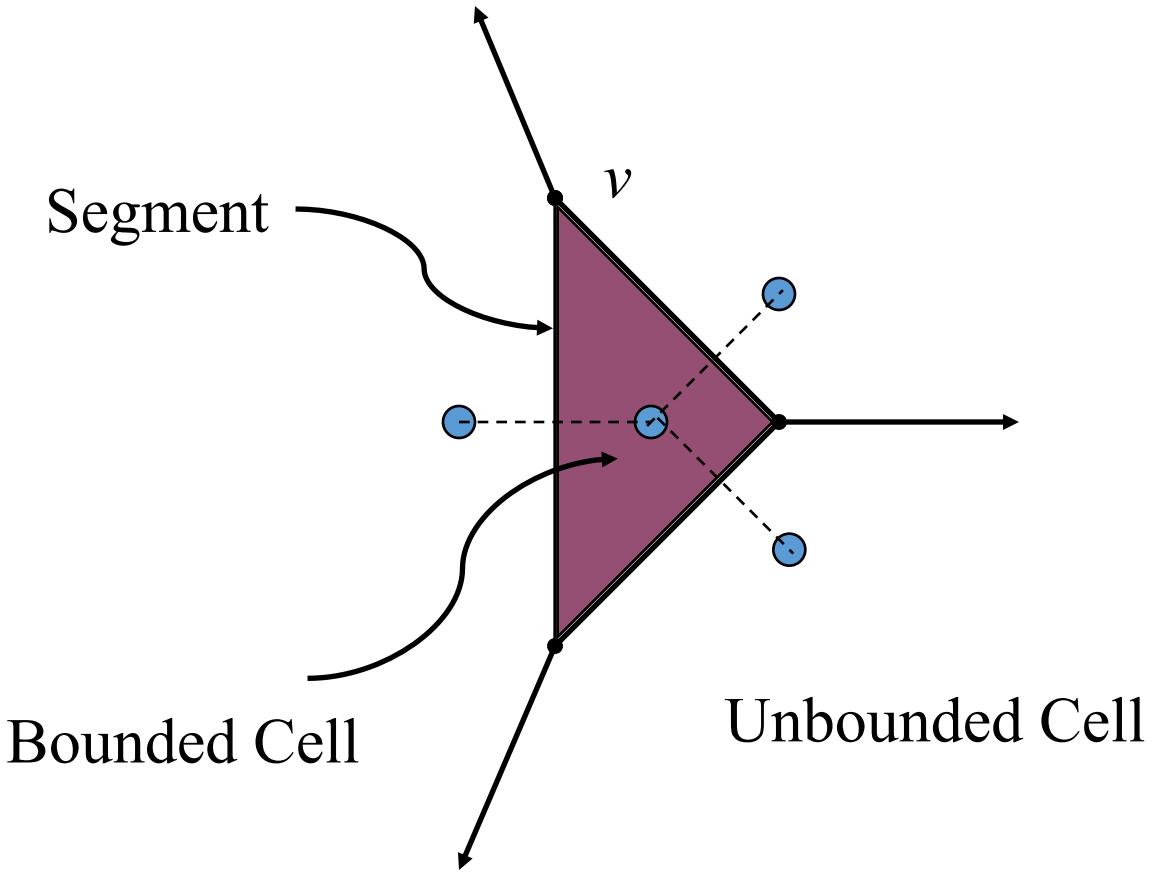
Non-collinear sites form Voronoi half lines that meet at a vertex

A vertex has degree ≥ 3



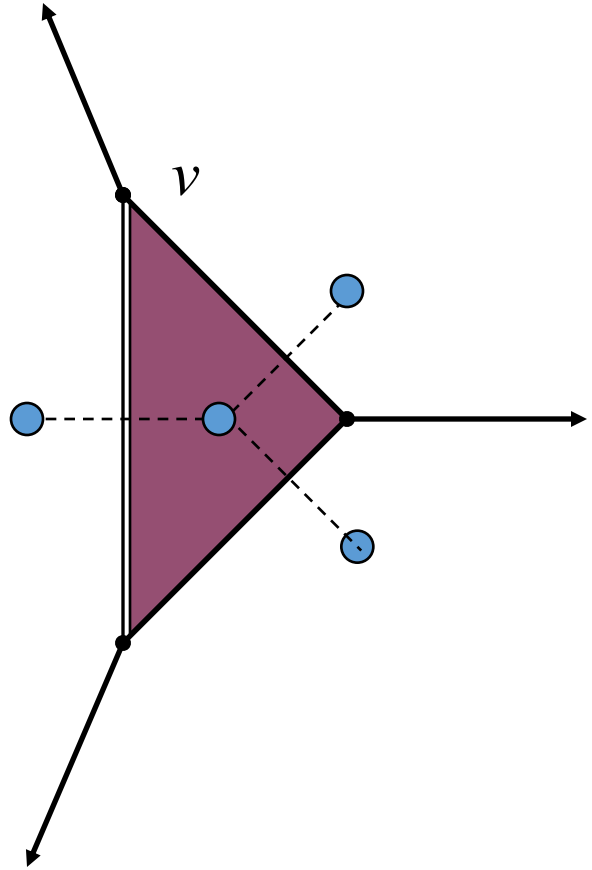
A Voronoi vertex is the center of an *empty* circle touching 3 or more sites.

Voronoi Cells and Segments

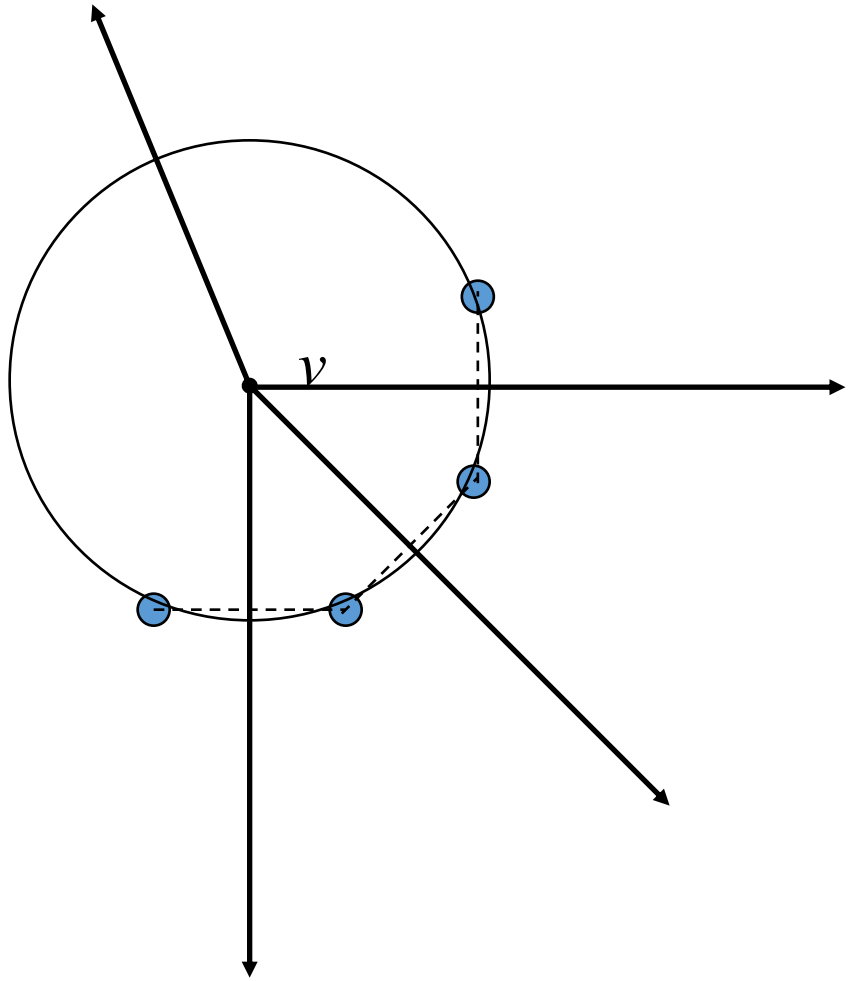


2-D Voronoi diagrams

- Which of the following is true for 2-D Voronoi diagrams?
  - Four or more non-collinear sites are…
    1. sufficient to create a bounded cell
    2. necessary to create a bounded cell
    3. 1 and 2
    4. none of above

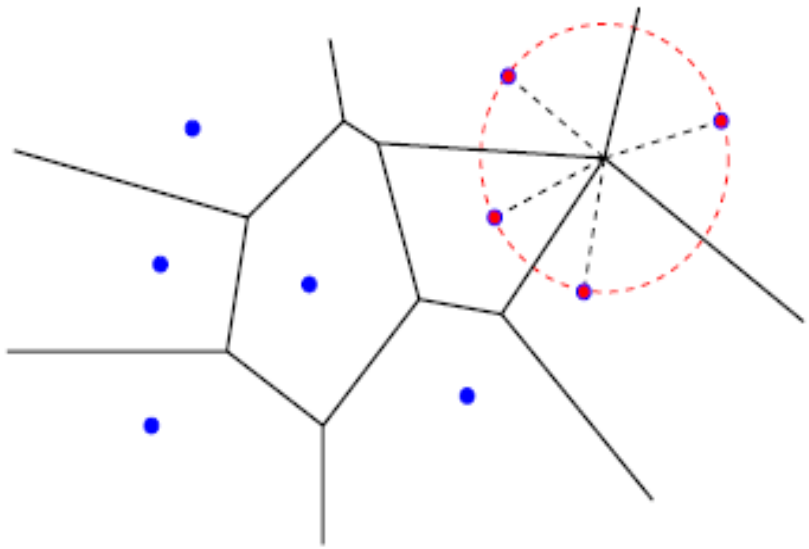


Degenerate Case: no bounded cells!



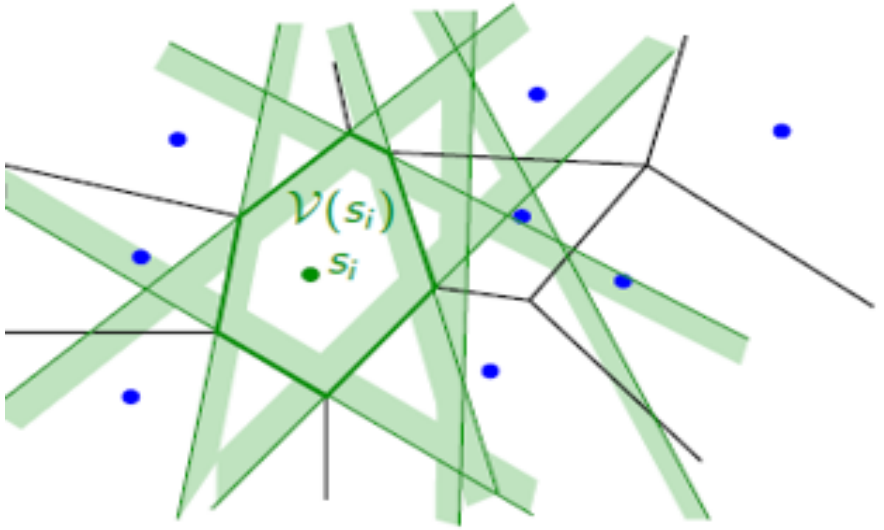
General Position Assumption

- General position assumption: No four sites are co-circular.



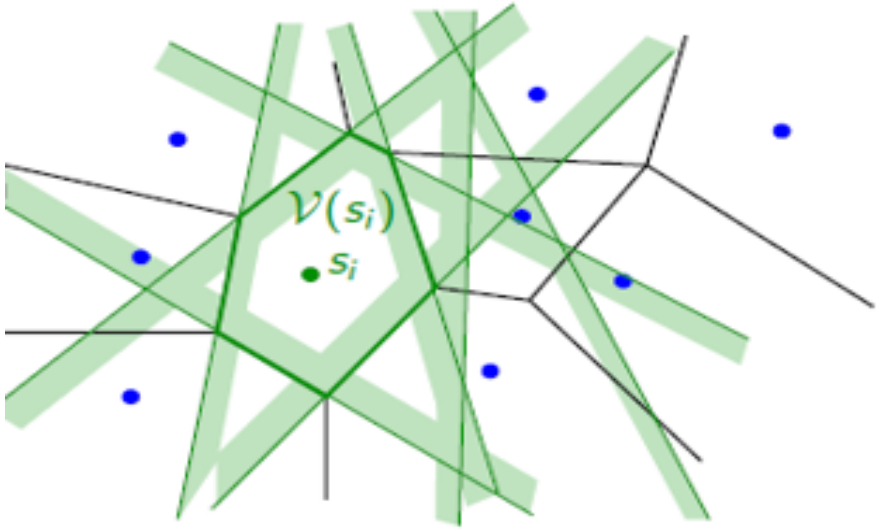
Naïve Algorithm: Intersection of Half-planes

- The equation for $V(s_i) = \bigcap_{i \neq j} H(s_i, s_j)$



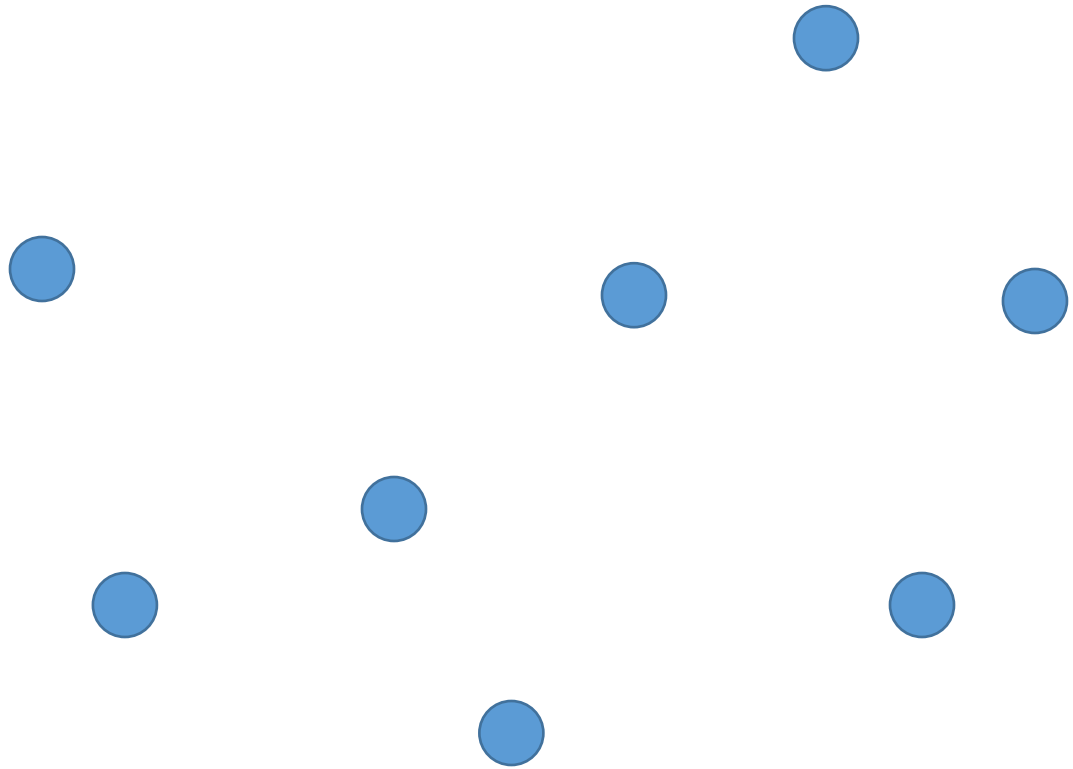
Naïve Algorithm: Intersection of Half-planes

- Each Voronoi region is constructed separately, by intersecting $n - 1$ half-planes
  - Dual to the task of constructing the convex hull
  - Complexity for one site is $O(n \log n)$
  - Overall $O(n^2 \log n)$



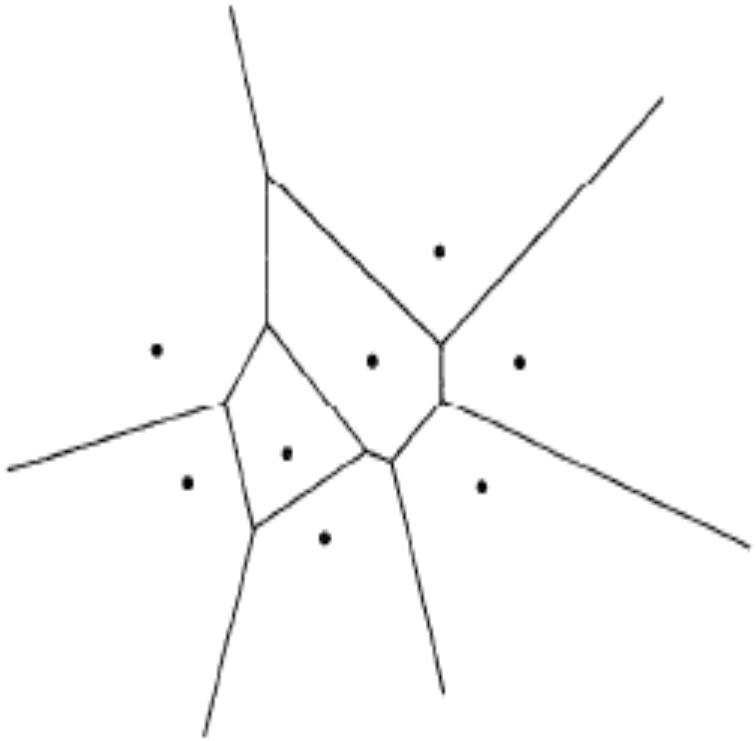
Voronoi Diagram

- Construct Voronoi diagram for eight sites



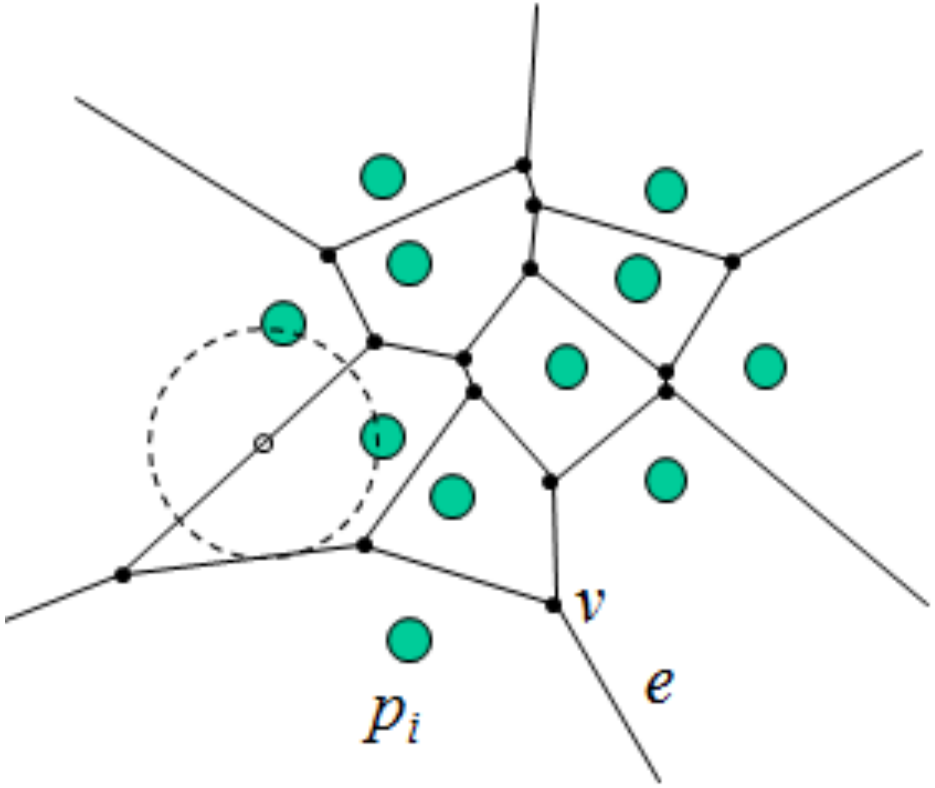
Voronoi Diagram Properties

- Since the regions are coming from intersecting $n-1$ half planes, they are convex polygons.
- Thus the boundary of a region consists of at most $n-1$ edges (maximal open straight-line segments) and vertices (their endpoints).
- Each point on an edge is equidistant from exactly two sites, and each vertex is equidistant from at least three.



Voronoi Diagram Properties

- A point q lies on a Voronoi edge between sites p_i and p_j iff the largest empty circle centered at q touches only p_i and p_j
  - A Voronoi edge is a subset of locus of points equidistant from p_i and p_j



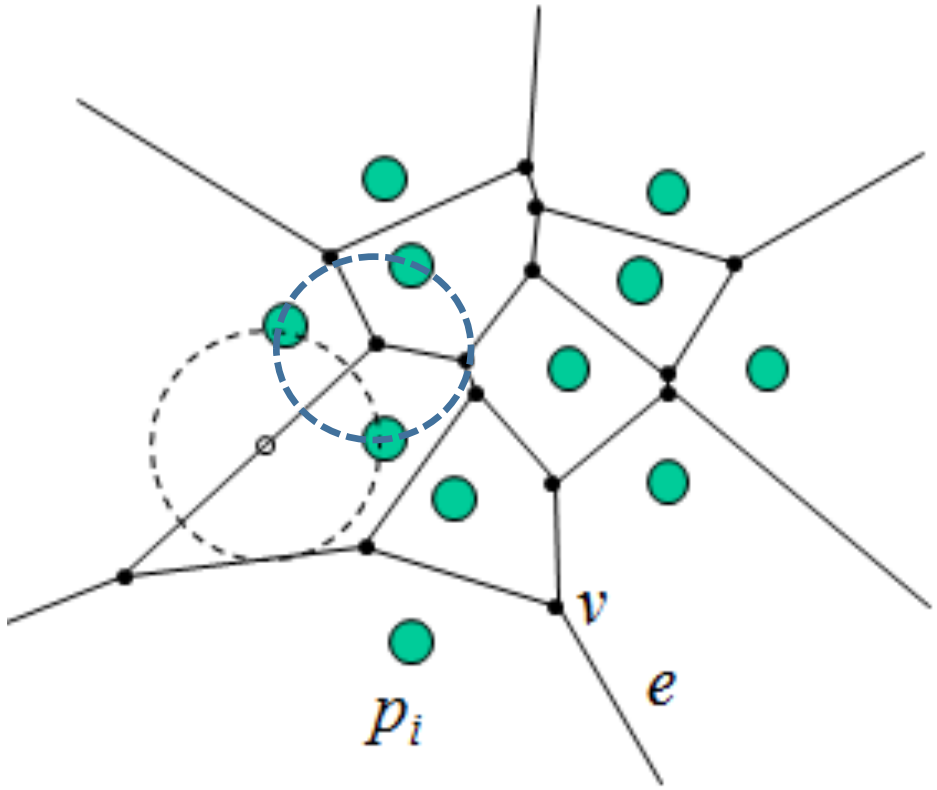
p_i : site points

e : Voronoi edge

v : Voronoi vertex

Voronoi Diagram Properties

- A point q is a vertex iff the largest empty circle centered at q touches at least 3 sites
  - A Voronoi vertex is an intersection of 3 more segments, each equidistant from a pair of sites



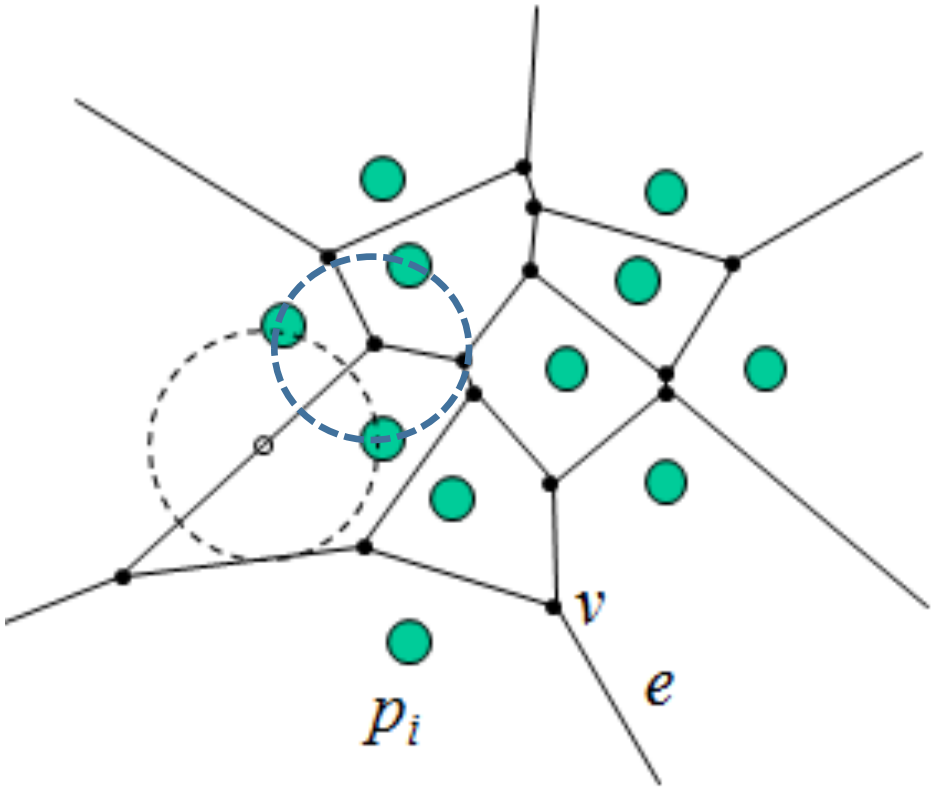
p_i : site points

e : Voronoi edge

v : Voronoi vertex

Complexity of Voronoi Diagram

- Voronoi diagrams have linear complexity $\{|v|, |e| = O(n)\}$
  - Three or more edges meet at a common vertex.
  - It should be observed that each vertex is the center of a circle that passes through at least three sites but encloses no site.

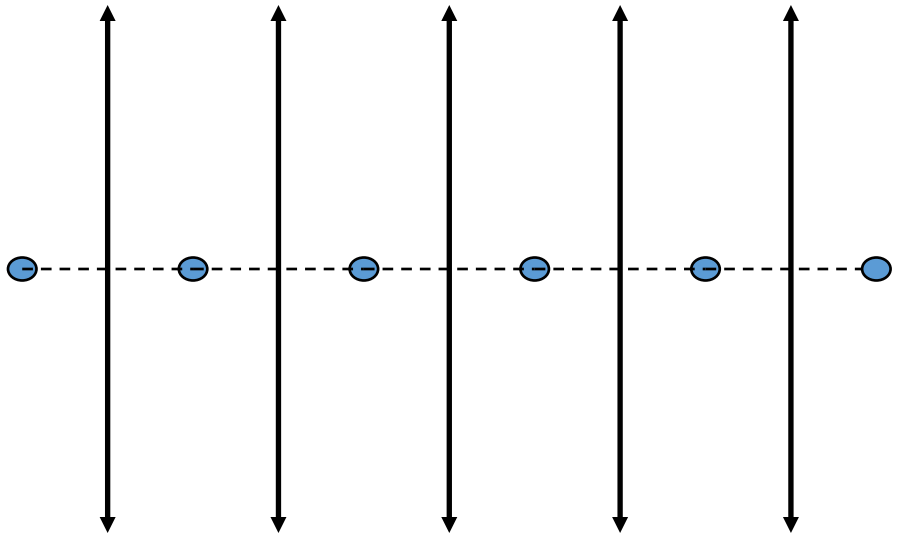


Complexity of Voronoi Diagram

- Although n sites give rise to $\binom{n}{2} = O(n^2)$ separators, only linearly many separators contribute an edge to $V(S)$.
  - This can be seen by viewing a Voronoi diagram as a planar graph with n regions and minimum degree of 3.
  - Each of the edges has two vertices, and each of V vertices belongs to at least three edges.

Complexity of Voronoi Diagram

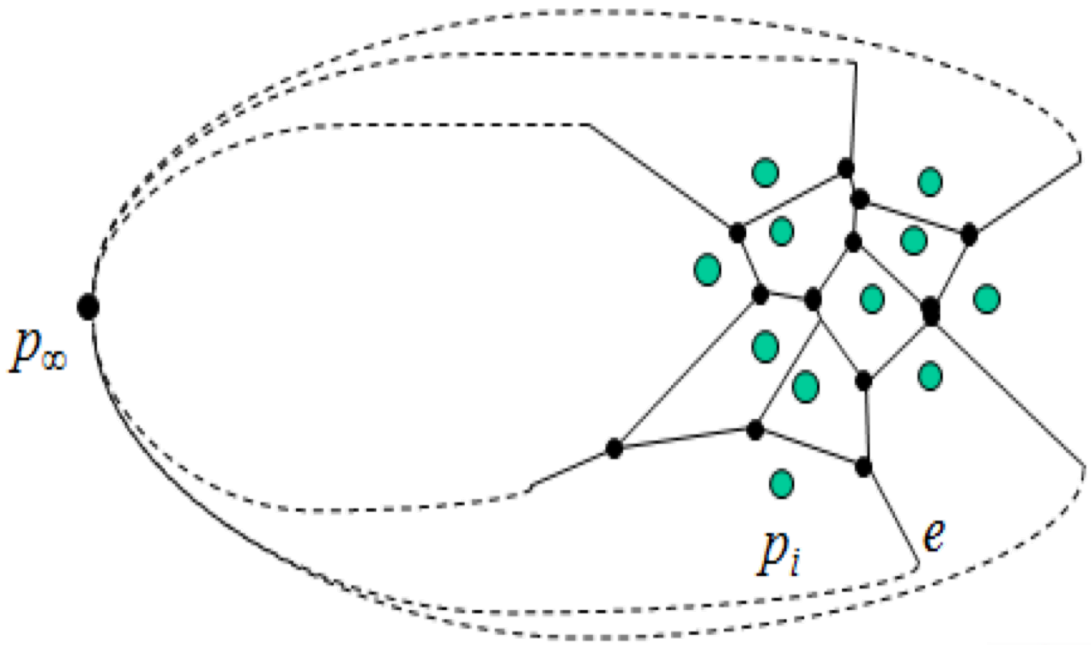
- Claim: For $n \geq 3, V \leq 2n-5$ and $E \leq 3n-6$
- Proof: Collinear sites form a series of parallel lines



$$V = 0, E = n - 1$$

Complexity of Voronoi Diagram

- Claim: For $n \geq 3$, $V \leq 2n-5$ and $E \leq 3n-6$
- Proof: (General Case)
  - To apply Euler's Formula, we "planarize" the Voronoi diagram by connecting half lines to an extra vertex.
  - Vertices in VD correspond to nodes in planar Graph



Euler's Formula

- Theorem: Let G be a connected planar graph, and let n, m and f denote, respectively, the numbers of vertices, edges, and faces in a plane drawing of G. Then $n - m + f = 2$.

Complexity of Voronoi Diagram

- Voronoi diagrams have linear complexity $\{|v|, |e| = O(n)\}$

$$\sum_{\forall V \in Vor(P)} \mathrm{DEG}(V) = 2E$$

$$\forall V \in Vor(P), \mathrm{DEG}(V) \geq 3$$

$$\therefore 2E \geq 3(V + 1)$$

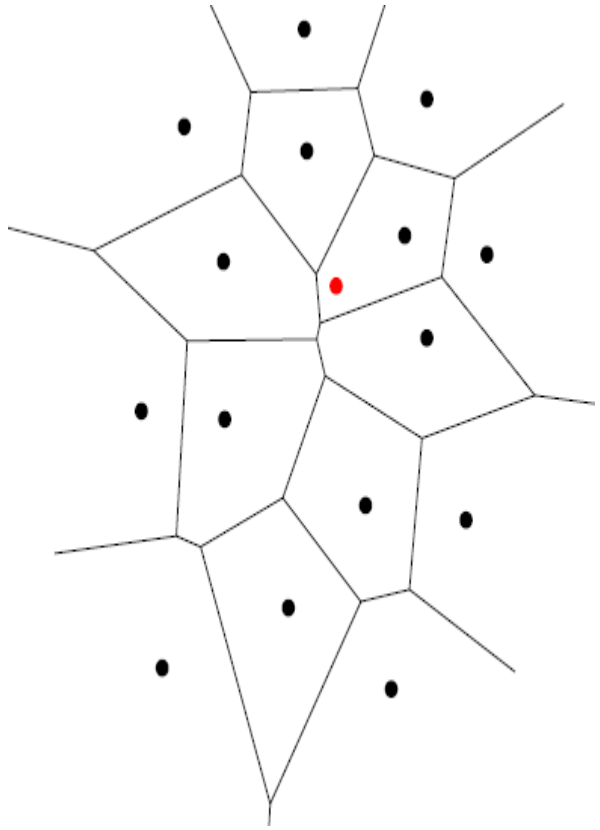
* +1 comes from planarized vertex

Complexity of Voronoi Diagram

- Given, $2E \geq 3(V + 1)$
- Using Euler formula, $(V + 1) - E + n = 2$
- For $n \geq 3, V \leq 2n - 5,\ E \leq 3n - 6$

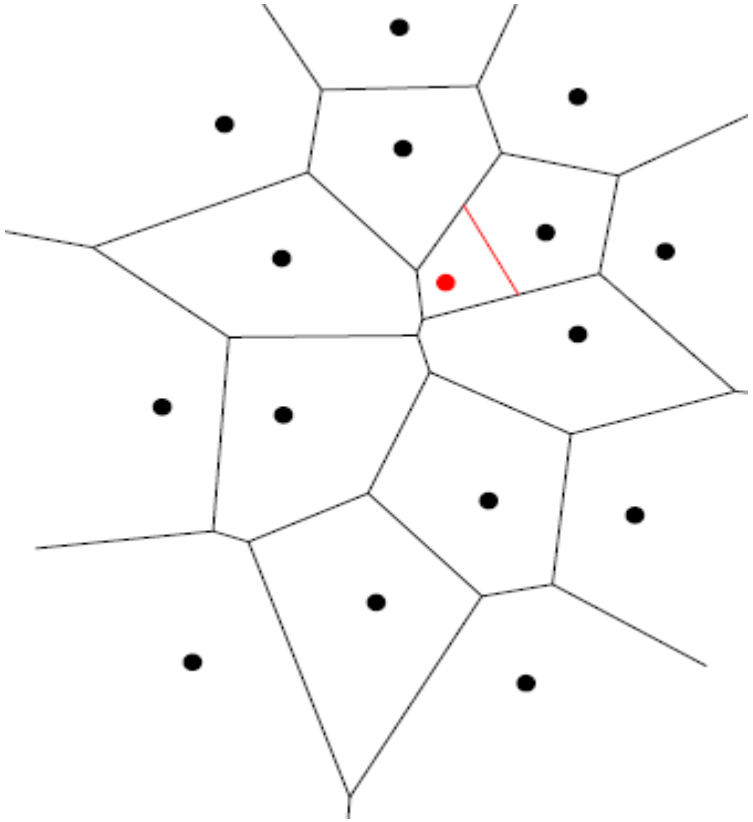
Incremental algorithm

1. Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$
2. Add point p_{i+1}
3. Explore all candidates to find the site p_j, $(1 \leq j \leq i)$ closest to p_{i+1}
4. Compute its region
   a. Built its boundary starting with bisector $b_{i+1,j}$
5. Prune the initial diagram
6. While building the Voronoi region of p_{i+1}, update DCEL



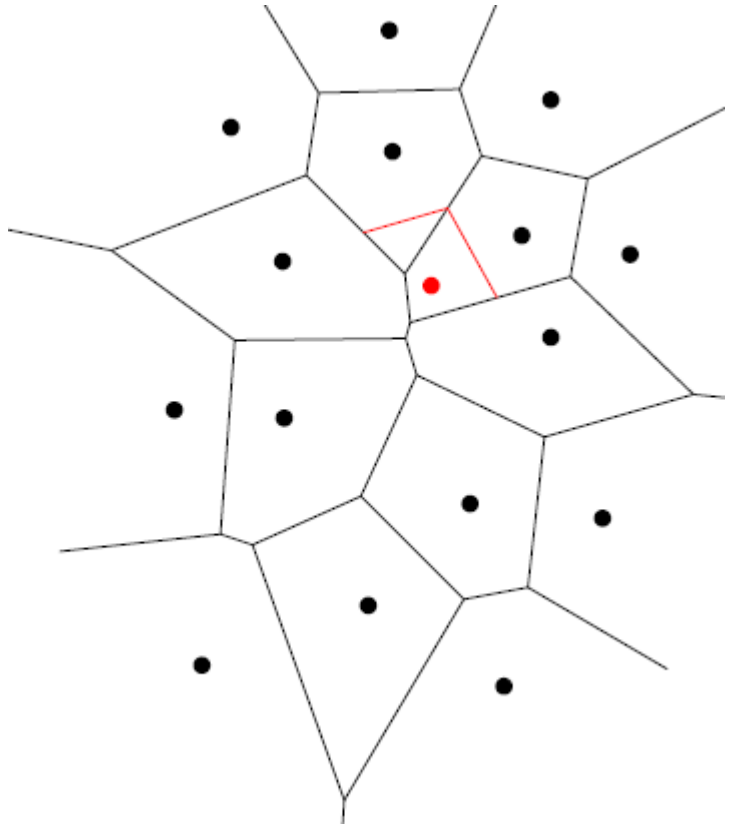
INCREMENTAL ALGORITHM

- Built its boundary starting with bisector $b_{i+1,j}$



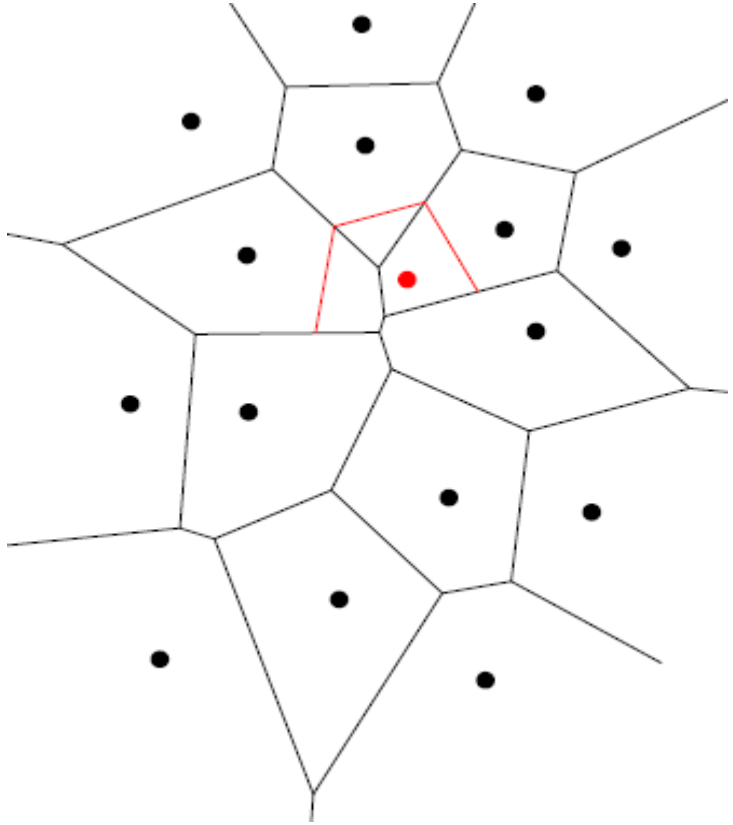
INCREMENTAL ALGORITHM

$\forall p_j, (1 \leq j \leq i)$ BUILT ITS BOUNDARY USING BISECTOR



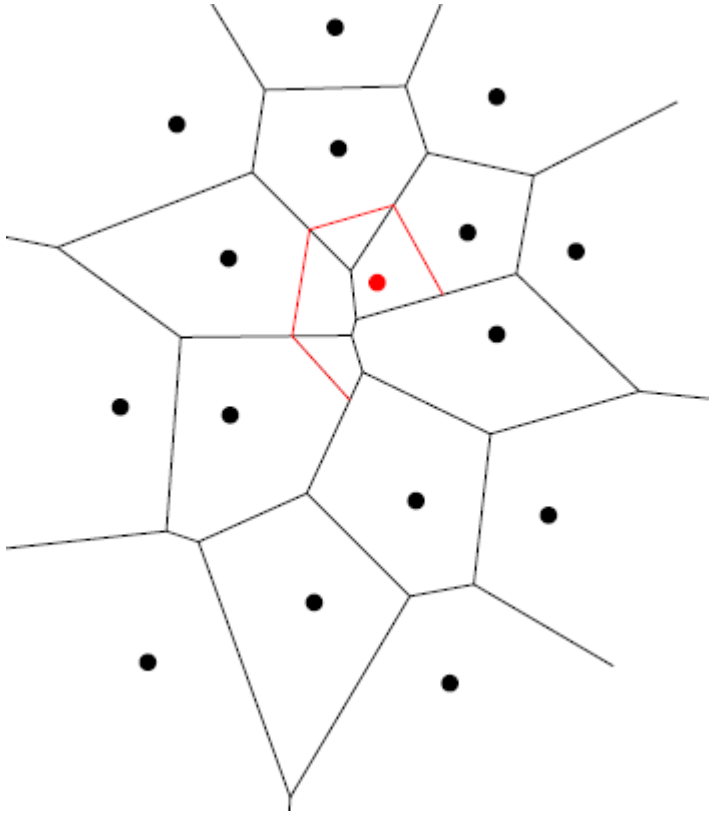
INCREMENTAL ALGORITHM

$\forall p_j, (1 \leq j \leq i)$ built its boundary using bisector



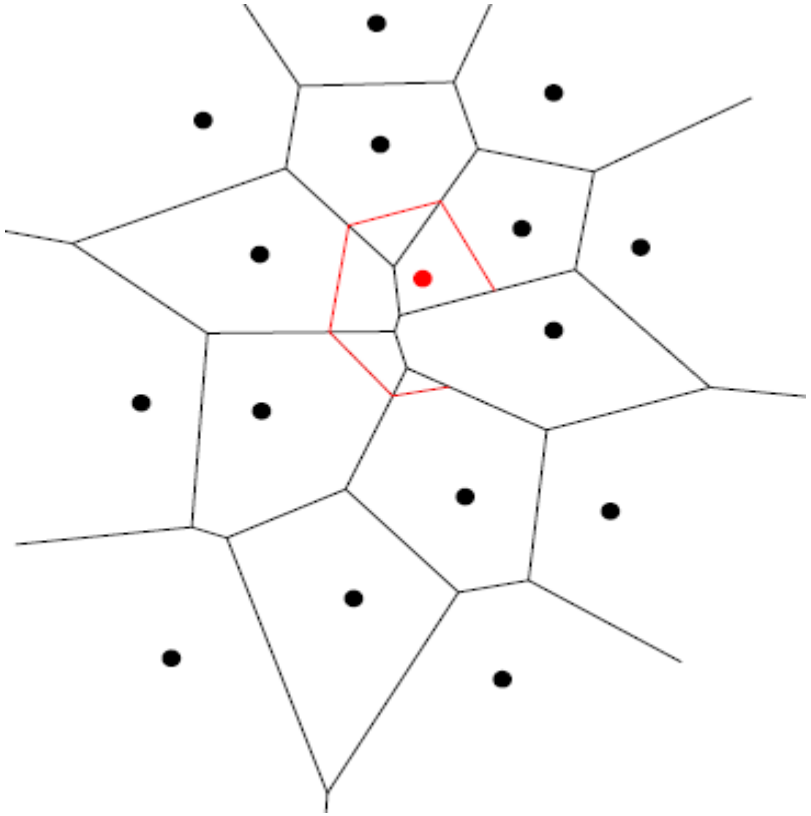
INCREMENTAL ALGORITHM

$\forall p_j, (1 \leq j \leq i)$ BUILT ITS BOUNDARY USING BISECTOR



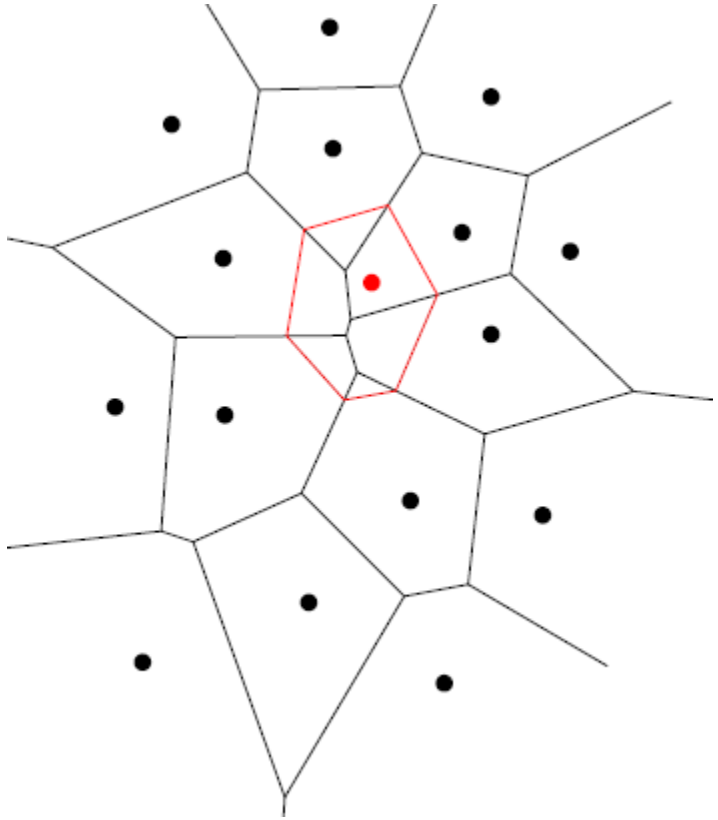
INCREMENTAL ALGORITHM

$\forall p_j, (1 \leq j \leq i)$ BUILT ITS BOUNDARY USING BISECTOR



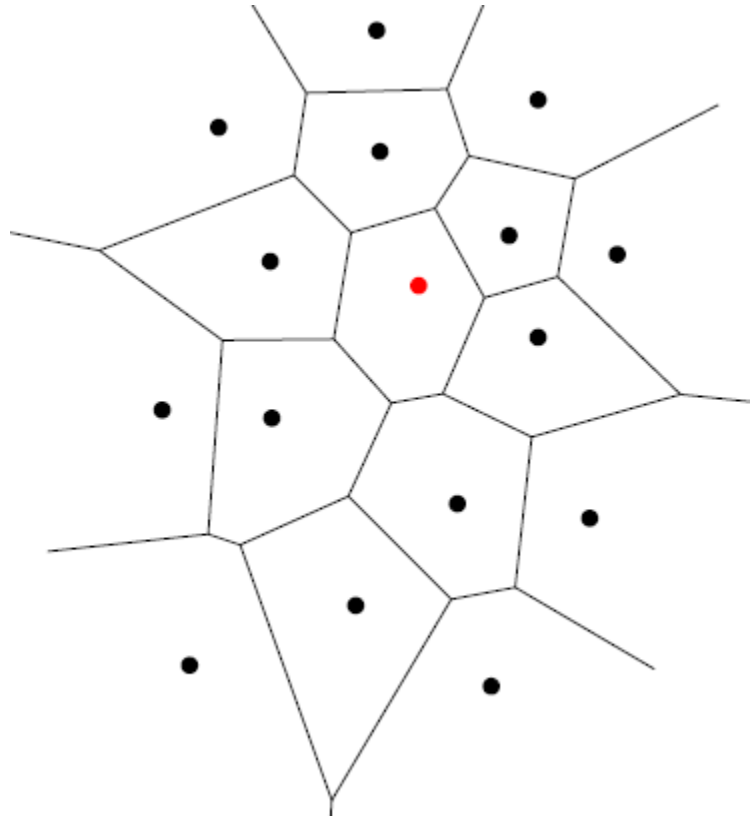
INCREMENTAL ALGORITHM

- Delete all edges of the region of p_j, that lie between new and old vertices in clock wise order
- Update $e(p_j) = e$
- Create v with $e(v) = e$



INCREMENTAL ALGORITHM

- PRUNE THE INITIAL DIAGRAM



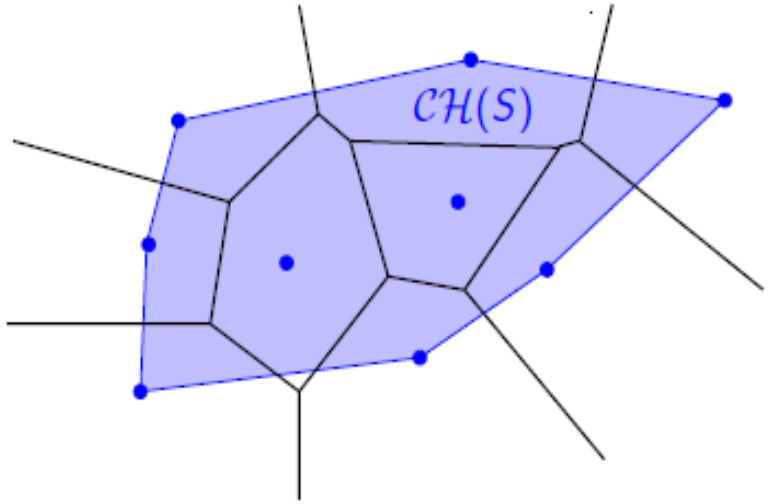
Incremental algorithm

1. Starting with Voronoi diagram of $\{p_1, \ldots, p_i\}$ and add p_{i+1}
2. Explore all candidates to find the site $p_j, (1 \leq j \leq i)$
3. Compute its region
   1. Built its boundary starting with bisector $b_{i+1,j}$
4. Prune the initial diagram

Running time: total running time of the algorithm $O(n^2)$

Voronoi Cells

- Each Voronoi region $V(s_i)$ is convex
- If $V(s_i)$ is bounded, then it is a convex polygon.
- $V(s_i)$ is unbounded if s_i is a vertex of $CH(S)$.



Consequence

- Knowing the Voronoi diagram, we can compute the convex hull in $O(n)$ time.
- Computing a Voronoi diagram naively takes $O(n^2 \log n)$ time.
- We just saw $O(n^2)$
- There is also a deterministic Plane-sweep algorithm with $O(n \log n)$ time algorithm.

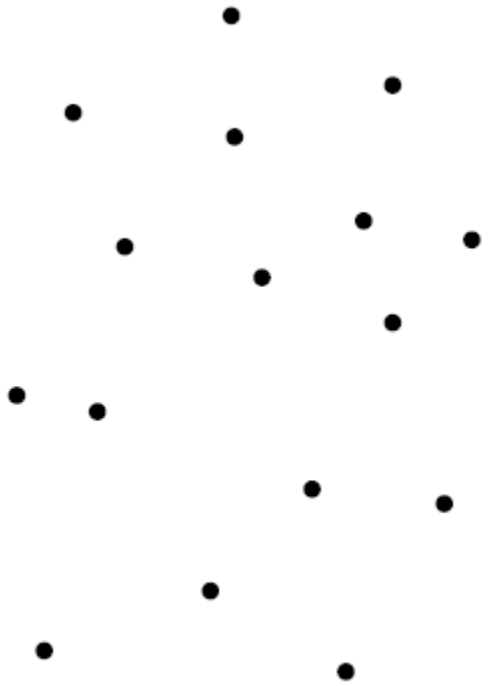
Divide and Conquer Algorithm

1. Sort the points of P by x-coordinate and vertically partition into two subsets R and L, of approximately the same size
2. Recursively compute V(R) and V(L)
3. Compute the separation chain
4. Prune the portion of V(R) lying to the left of the chain and the portion of V(L) to its right

The total running time of the algorithm is $O(nlogn)$
This running time is optimal

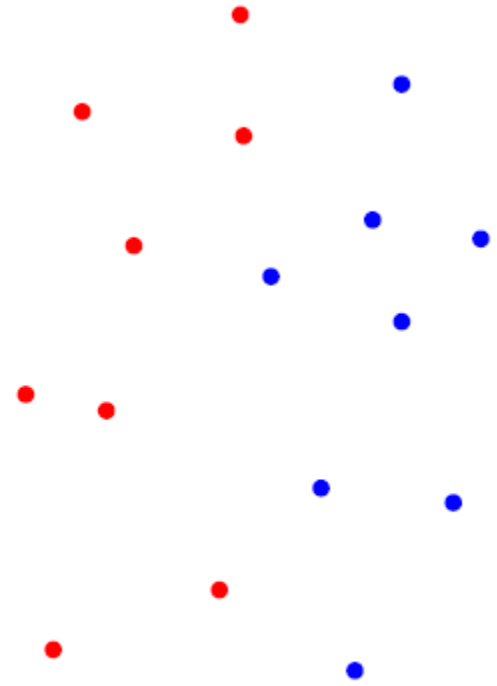
Divide and Conquer

- Let P be a set of n points in plane



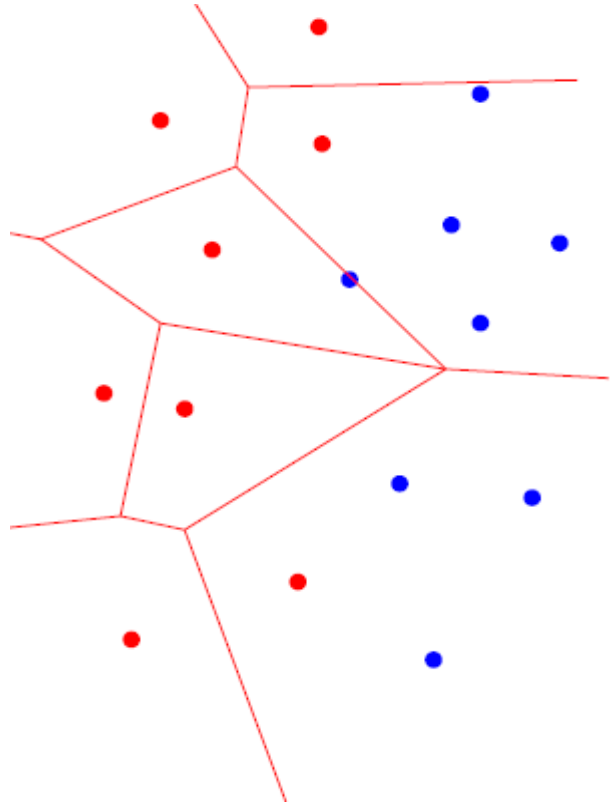
Divide and Conquer

1. Sort the points of P abscissa and vertically partition P into two subsets R and L, of approximately the same size



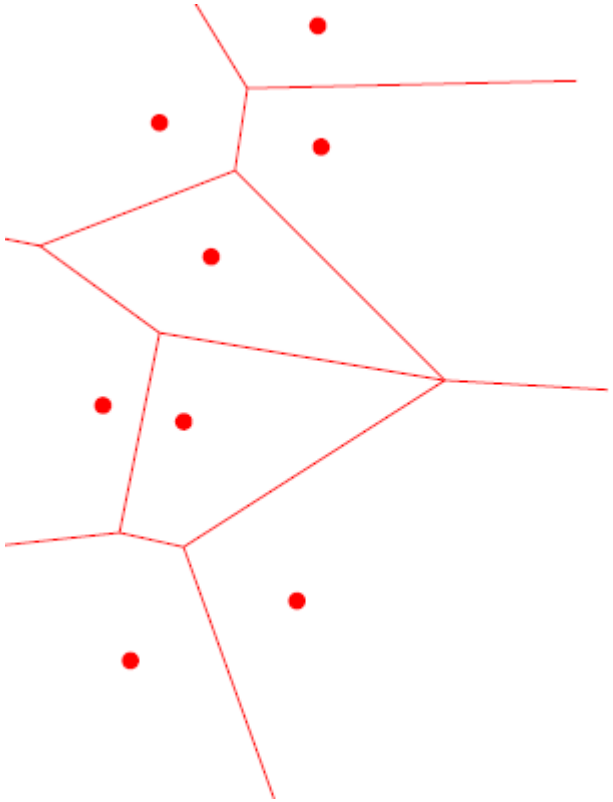
Divide and Conquer

2. Recursively compute V(L)



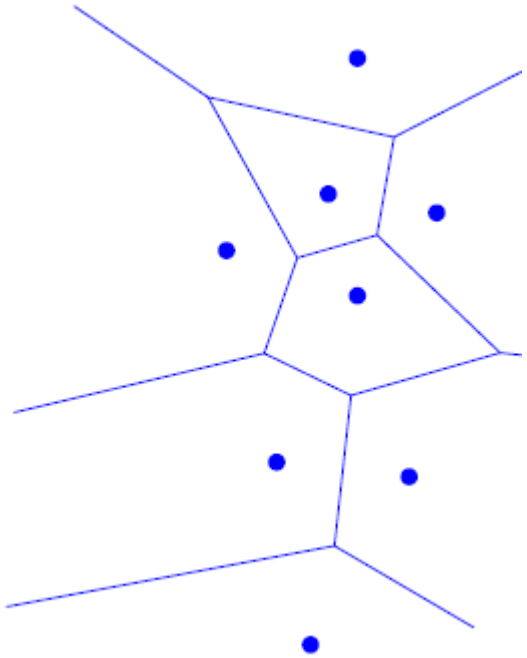
Divide and Conquer

- Consider the Voronoi diagram of L



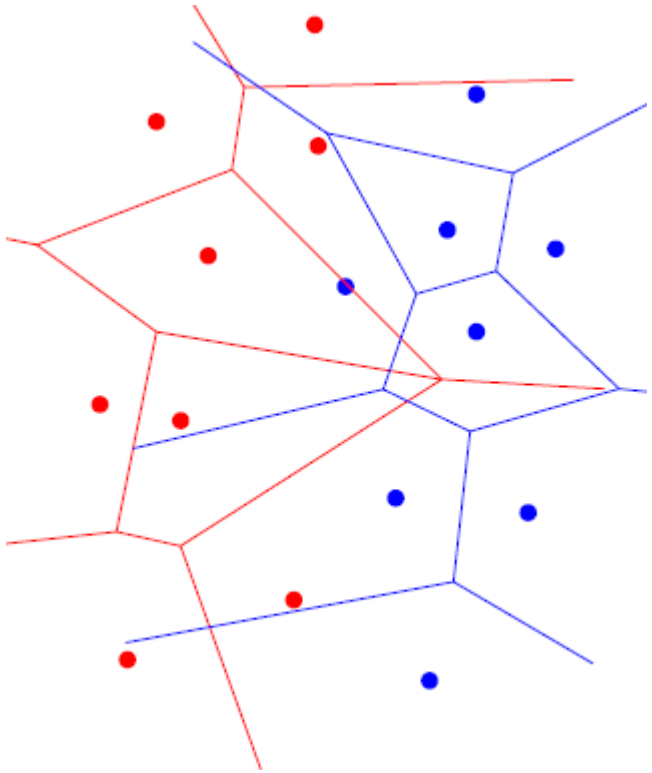
Divide and Conquer

- Recursively compute V(R)
- Consider the Voronoi diagram of R



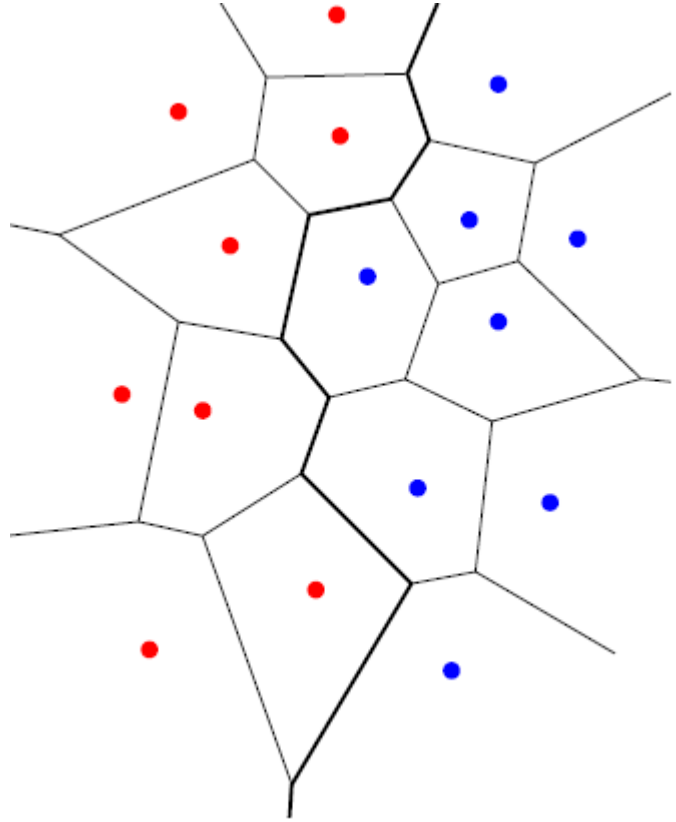
Divide and Conquer

- Combine two diagrams
- Then Voronoi diagram of P substantially coincides with the Voronoi diagram of R and L



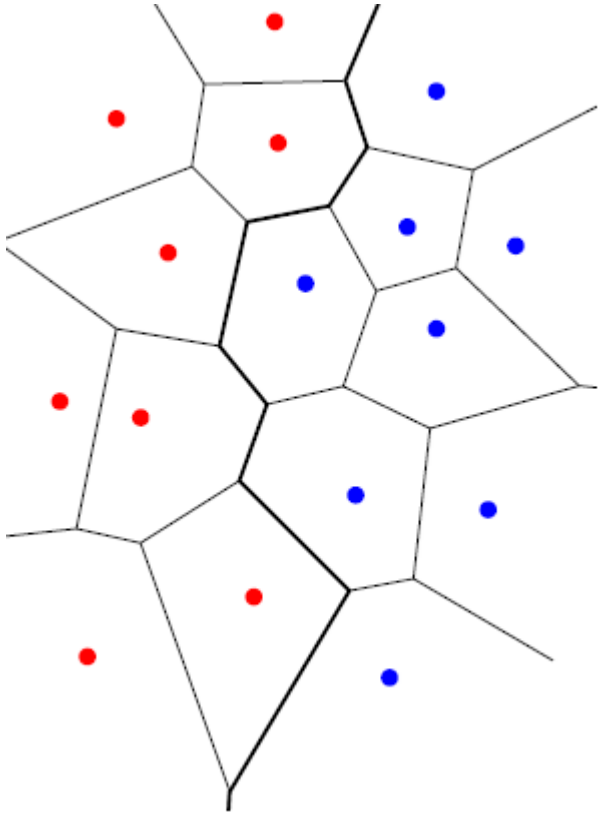
Divide and Conquer

- There exists a chain of edges of V such that V coincides with V (R) to the right of the chain, and it coincides with V(L) to its left



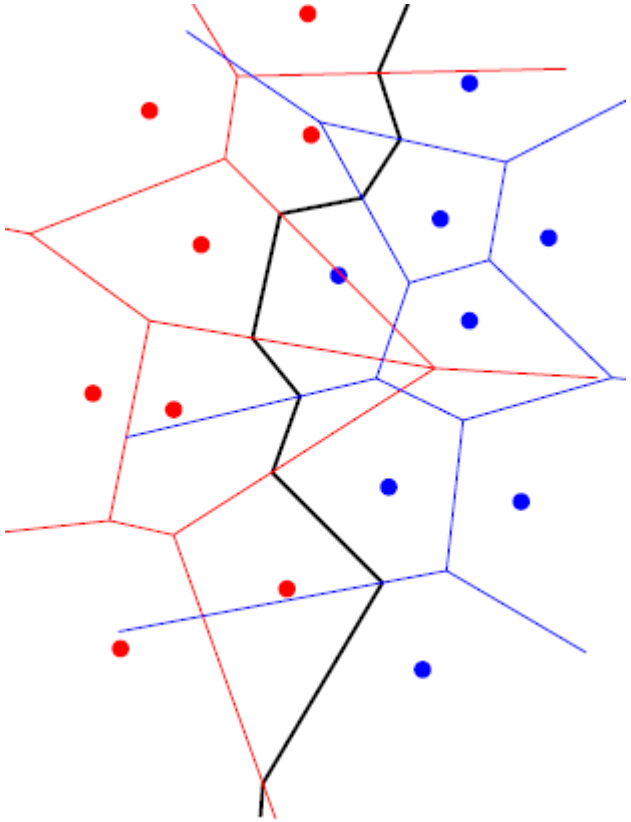
Divide and Conquer

- Definition: Let b(R,L) be the set of all edges and vertices of V (P) belonging to common boundary of the regions of some $p_i \in R \text{ and } p_j \in L$.
- Observation: The bisector b(R,L) is y-monotone chain leaving the regions of the points $p_i \in R$ to its right and those of $p_j \in L$ to its left



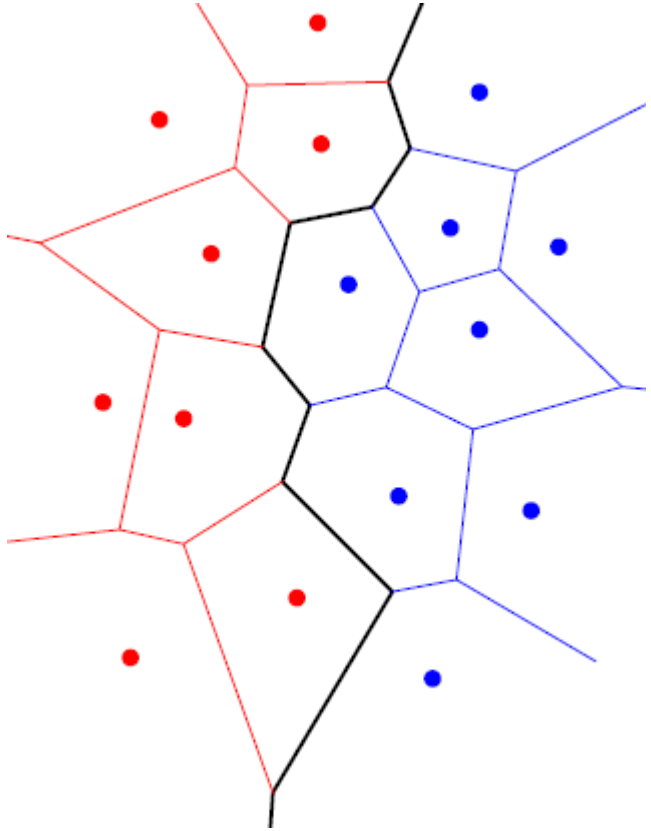
Divide and Conquer Algorithm

- Compute the separating chain



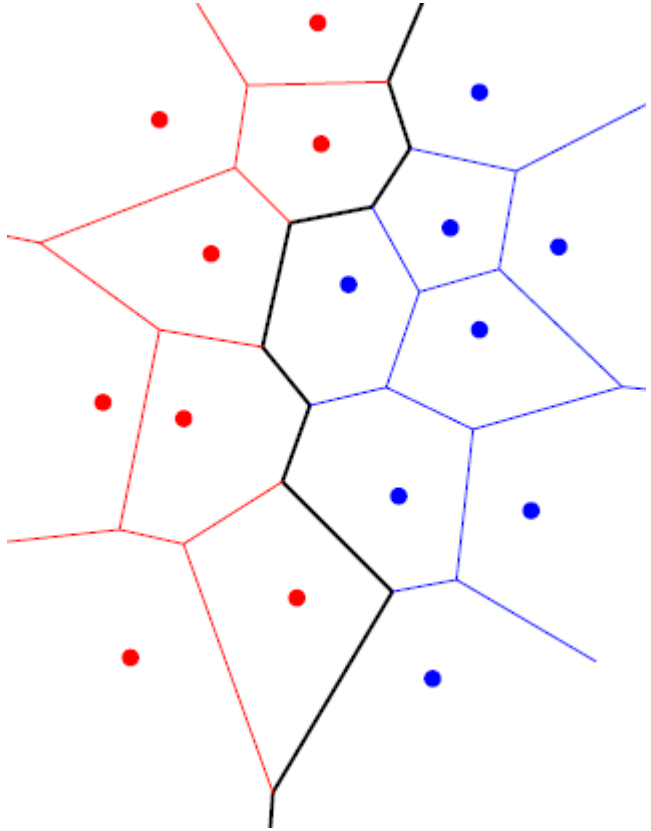
Divide and Conquer

- Prune the portion of V(R) lying to the right of the chain and the portion of V(L) lying to its left



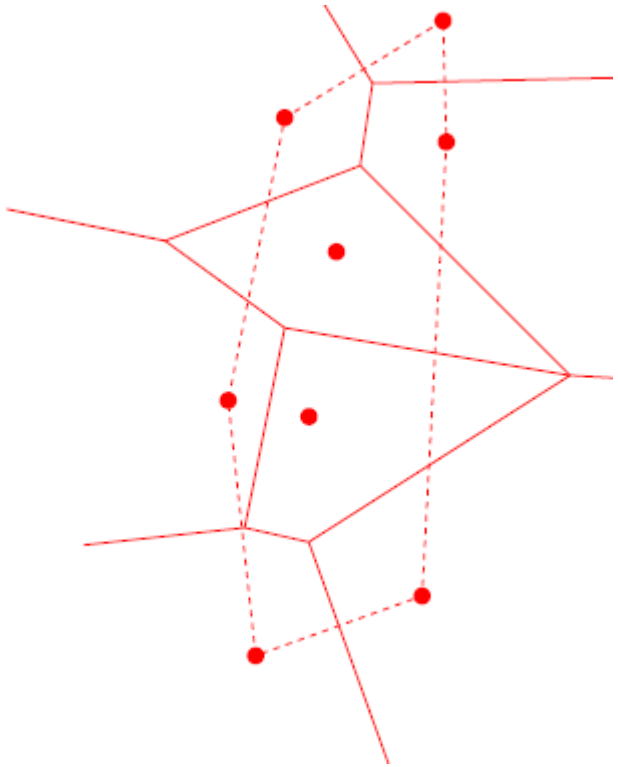
Divide and Conquer

- How to compute the chain?



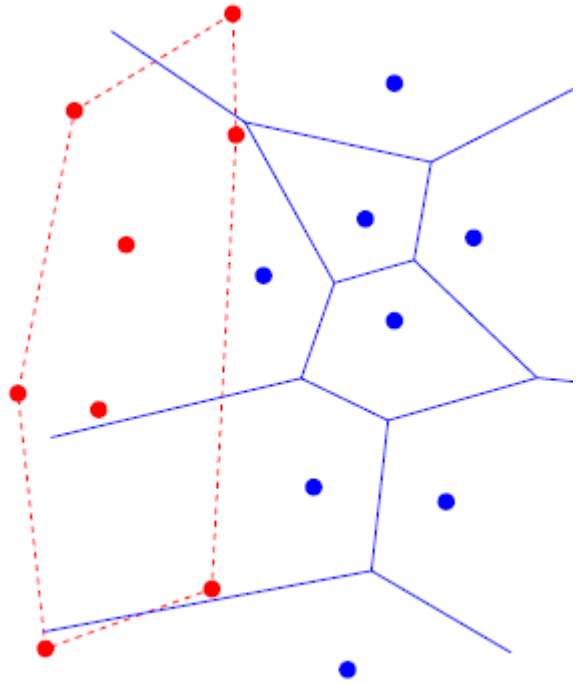
Divide and Conquer

- Initialization: Find the two half-lines



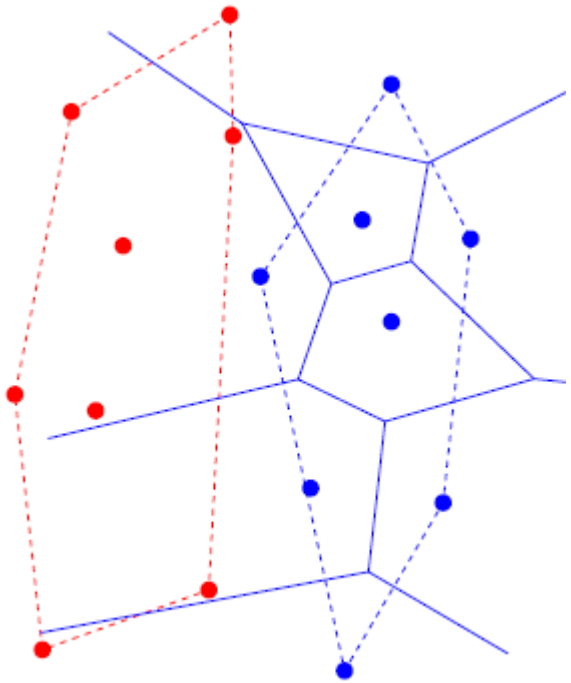
Divide and Conquer

- Find the convex hull for V(L)



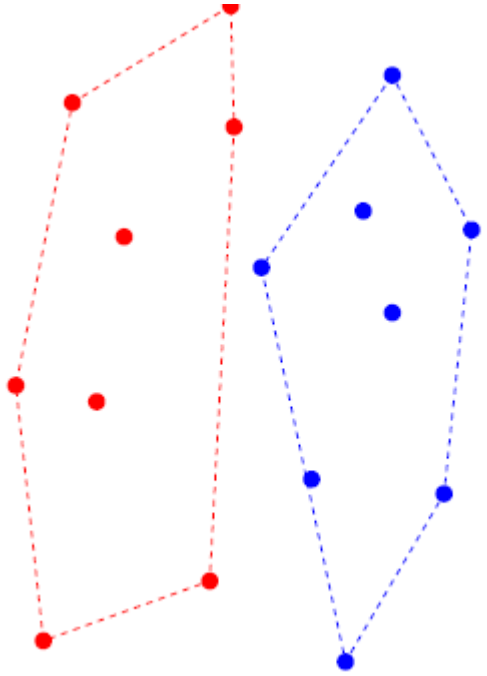
Divide and Conquer

- Find the convex hull for V(R)



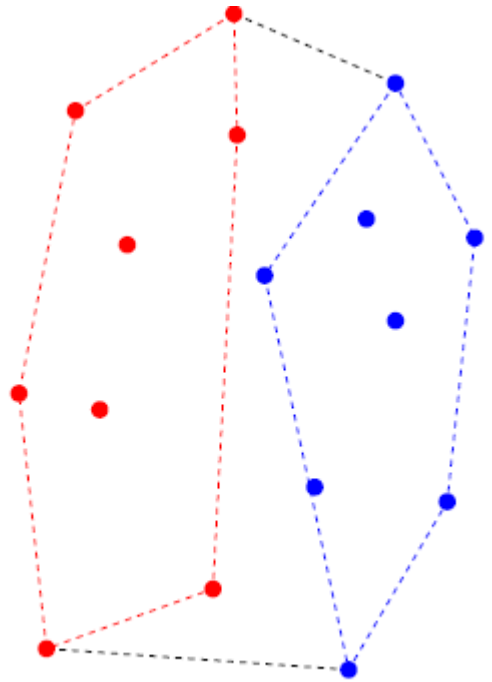
Divide and Conquer

- Find two highest and two lowest points



Divide and Conquer

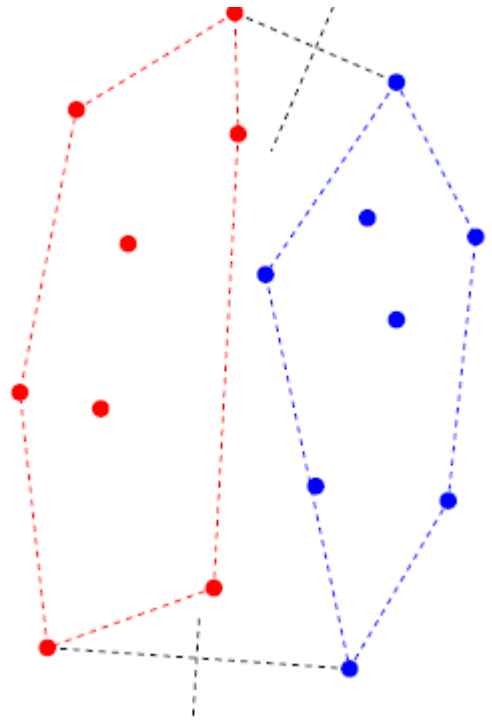
- We will connect the two highest points and the two lowest points



Divide and Conquer

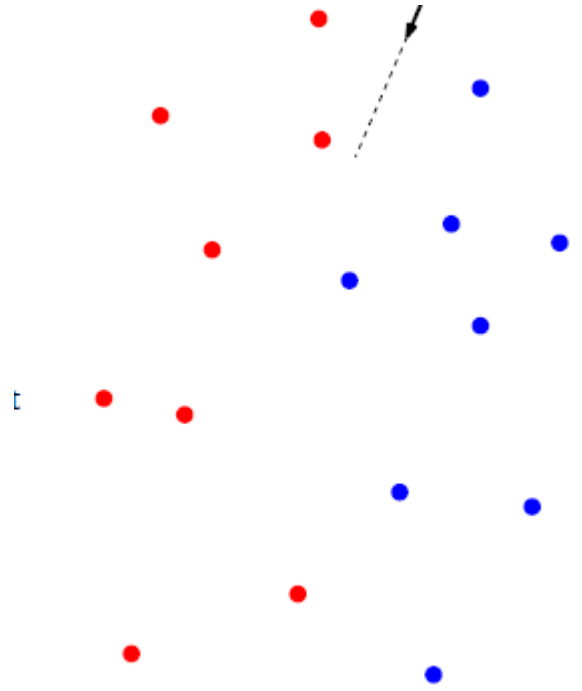
- Advance: Starting with one of the half-lines, and until getting to the other one

- Each time an edge $e \in b(R, L)$ begins, such that $e \in b_{ij}, p_i \in R \; and \; p_j \in L, do$:



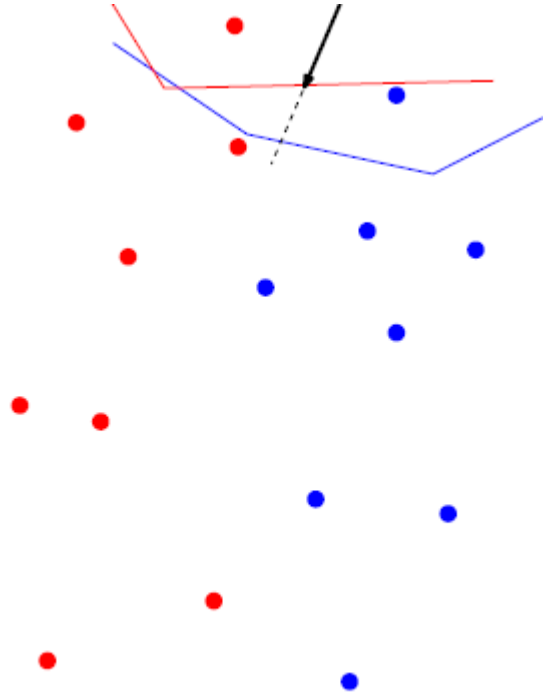
Divide and Conquer

- Detect intersection with $L(p_j)$ or $R(p_i)$



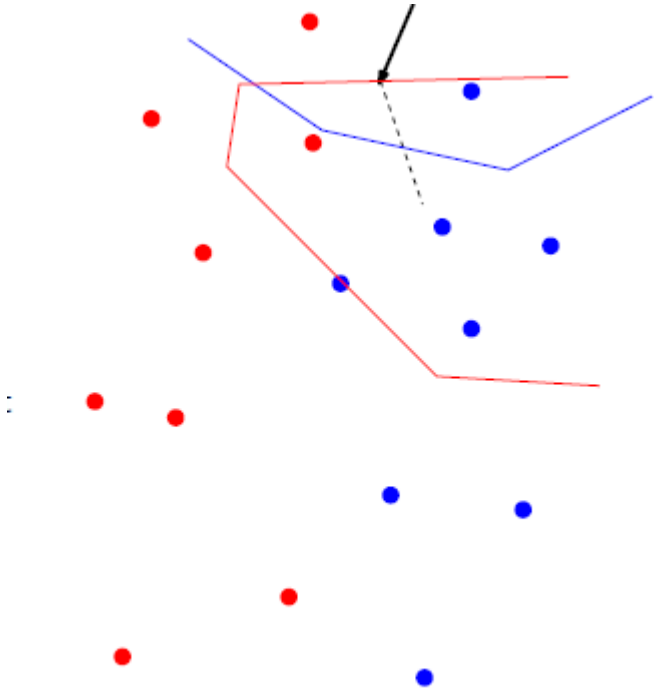
Divide and Conquer

- Chose the first of the two intersection points



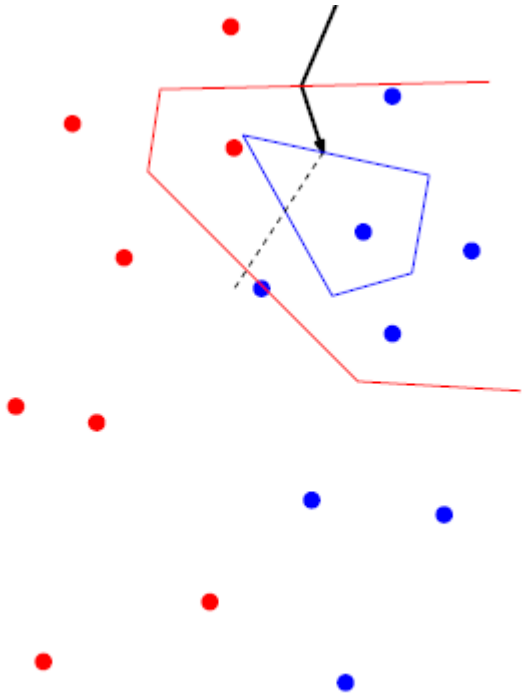
Divide and Conquer

- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j by p_k



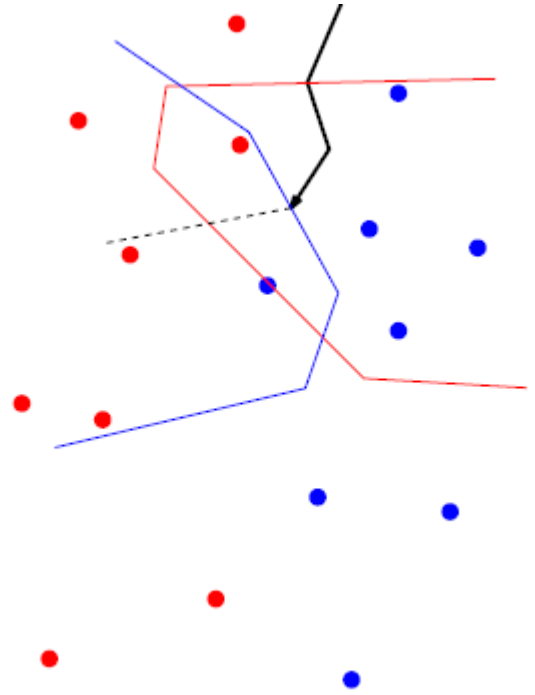
Divide and Conquer

- Restart with the new edge



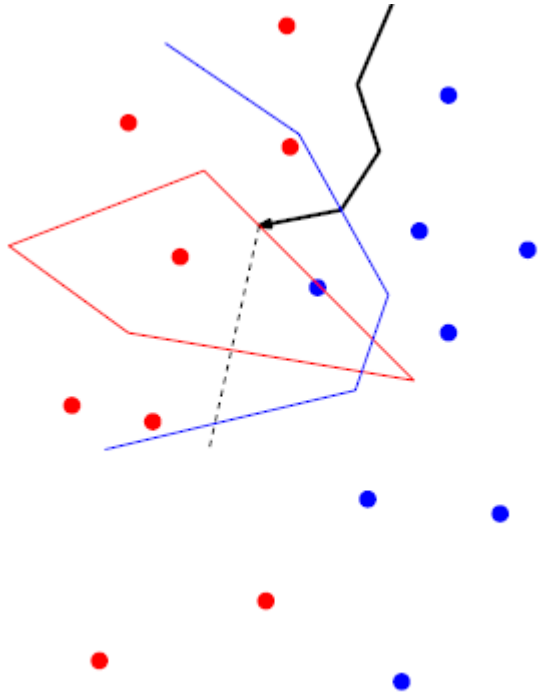
Divide and Conquer

- Continue the process



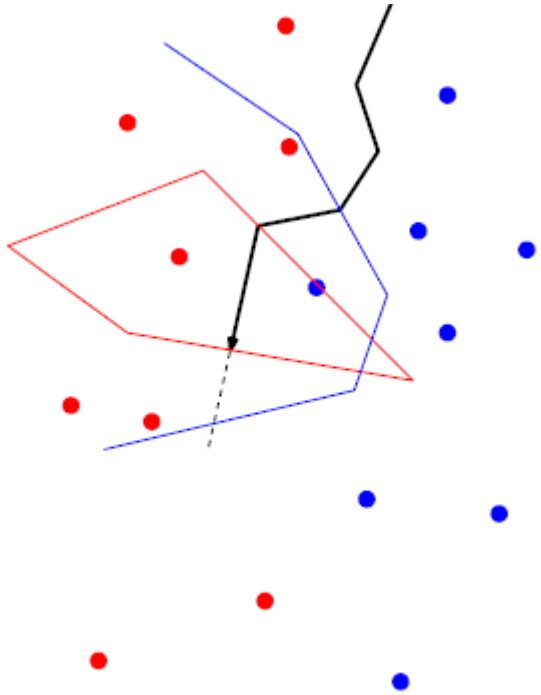
Divide and Conquer

- Continue



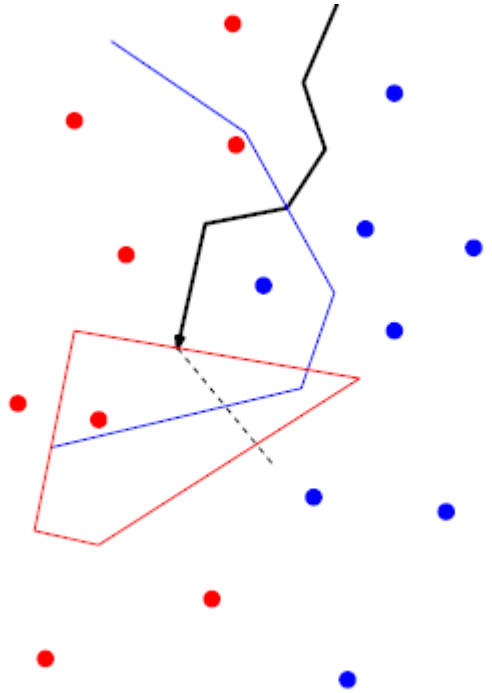
Divide and Conquer

- Restart with the new edge



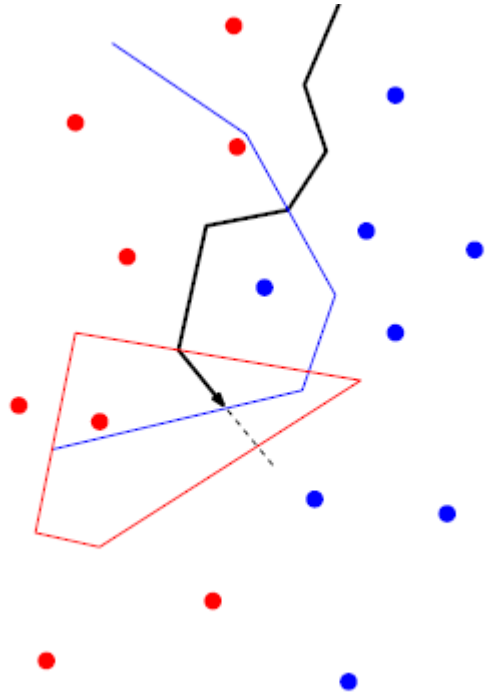
Divide and Conquer

- Restart with the new edge



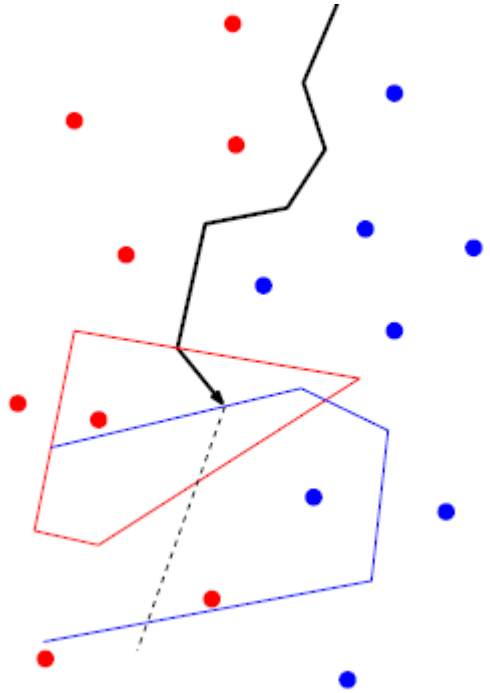
Divide and Conquer

- Restart with the new edge



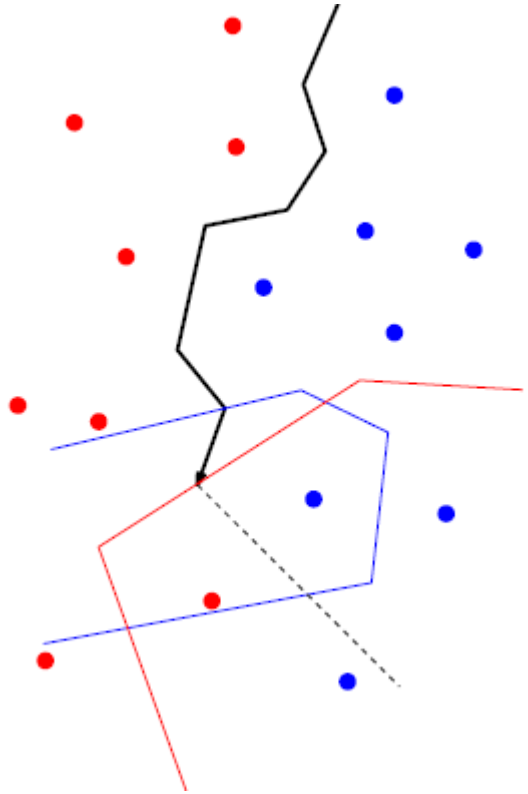
Divide and Conquer

- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j by p_k
- Restart with the new edge



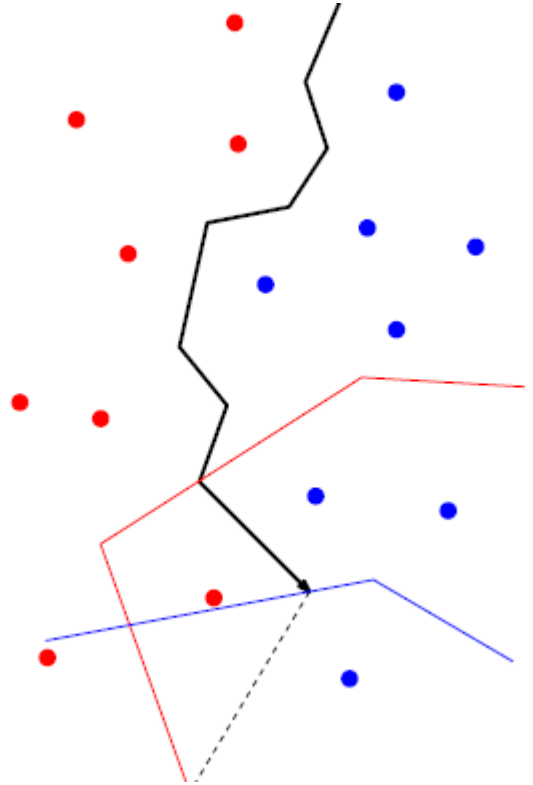
Divide and Conquer

- Replace p_i or p_j by p_k
- Restart with the new edge



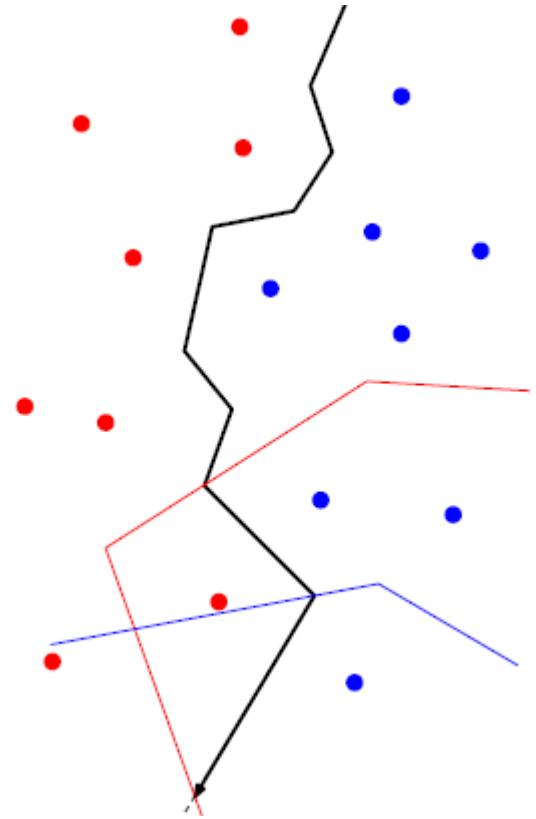
DIVIDE AND CONQUER

- REPLACE p_i OR p_j BY p_k
- RESTART WITH THE NEW EDGE



ALGORITHM OF CONSTRUCTING THE MONOTONE CHAIN

- ADVANCE: STARTING WITH ONE OF THE HALF-LINES, AND UNTIL GETTING TO THE OTHER ONE, DO:
- EACH TIME AN EDGE $e \in b(R, L)$ BEGINS, SUCH THAT $e \in b_{ij}, p_i \in R \; and \; p_j \in L, do$:
  - Detect intersection with V or $R(p_i)$
  - Detect intersection with V or $L(p_j)$
  - Chose the first of the two intersection points
  - Detect the site p_k corresponding to the new starting region
  - Replace p_i or p_j by p_k
  - Restart with the new edge



Divide and Conquer

- Performance Analysis?

Fortune's algorithm

- **Fortune's algorithm** is a sweep line algorithm for generating a Voronoi diagram from a set of points in a plane using $O(n \log n)$ time and $O(n)$ space

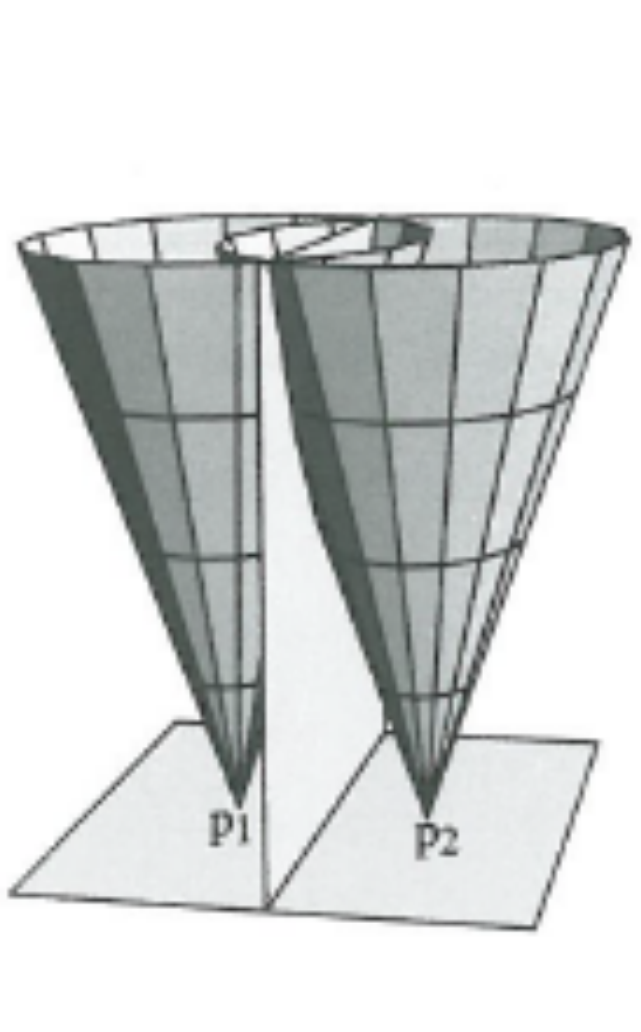
- The strategy in a plane sweep algorithm is to sweep a horizontal line (sweep line) (L) from top to bottom
  - Information maintained about the part of the Voronoi diagram of the sites above L that cannot be changed by sites below L

Fortune's Algorithm

- The algorithm maintains both a *sweep line* and a *beach line*, which both move through the plane as the algorithm progresses.
  - Horizontal sweep line maintains order of construction
  - Beach line maintains portion of diagram, which cannot change due to sites below sweep line

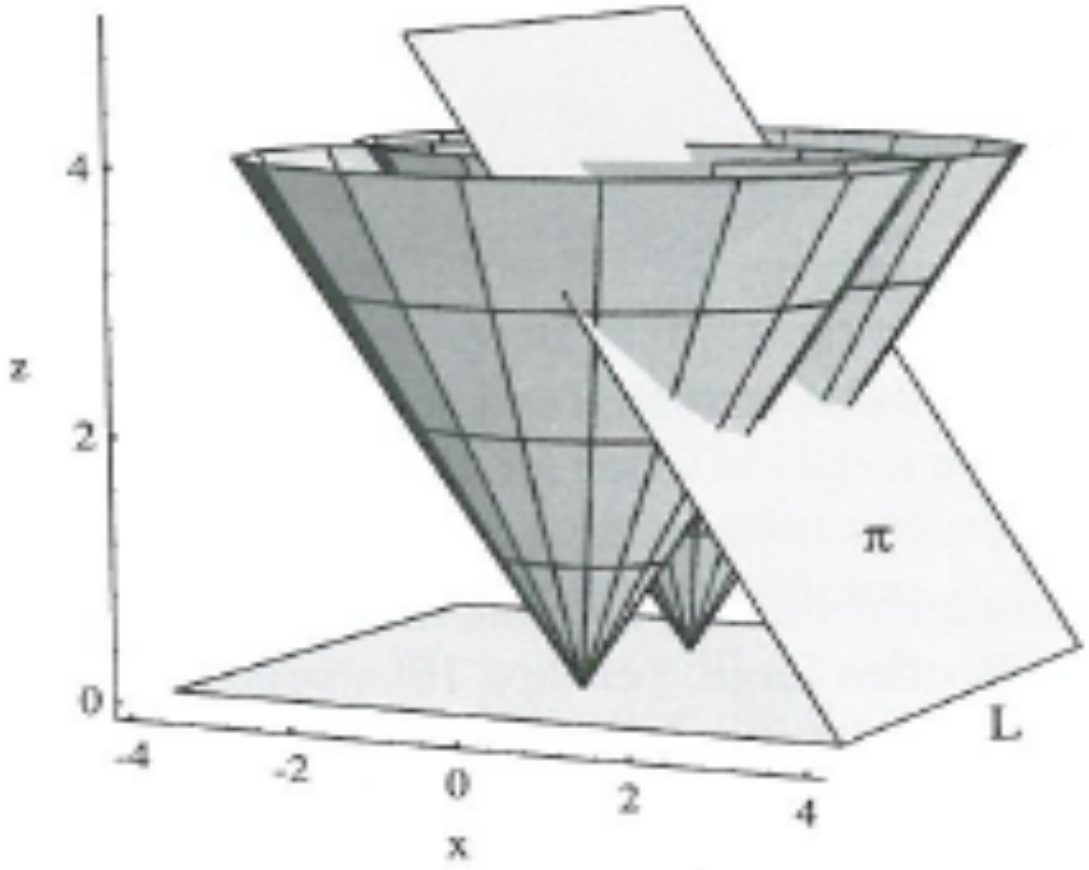
Fortune's algorithm

- Consider two nearby cones
- If cones over all sites are opaque, and they are viewed from $z \to -\infty$, what is seen is precisely the Voronoi diagram



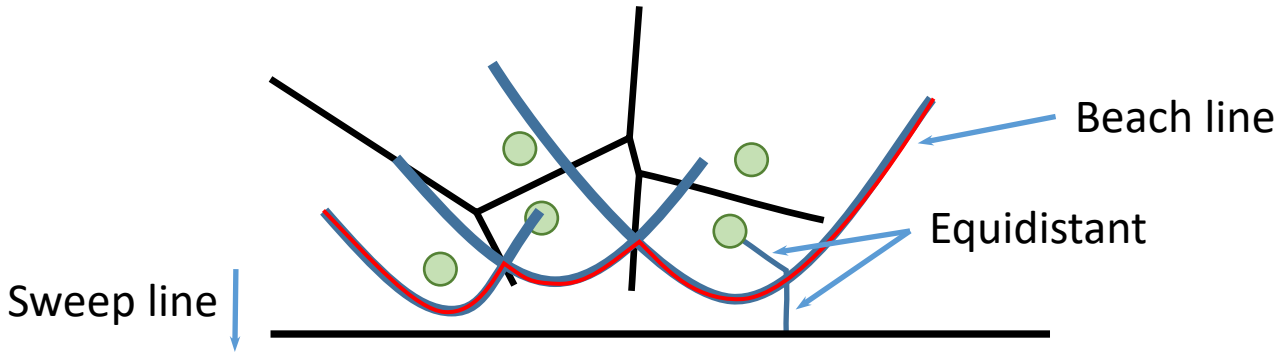
FORTUNE'S ALGORITHM

- FORTUNE'S IDEA: SWEEP THE CONES WITH A PLANE π, SLANTED AT 45° TO THE XY-PLANE



Beach line

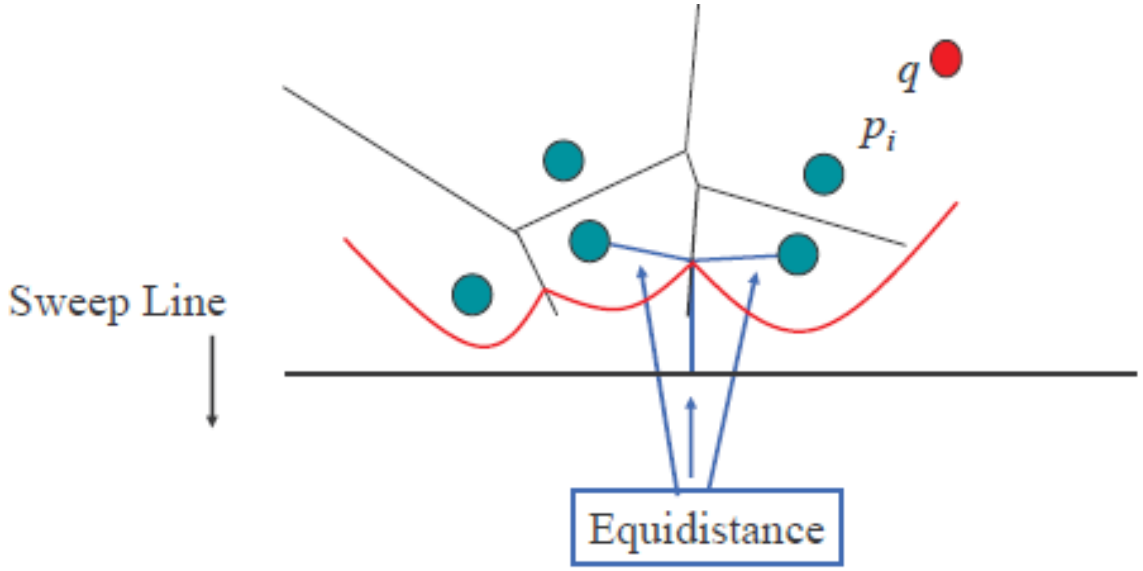
- Which points are closer to a site above the sweep line than to the sweep line itself?



- The set of parabolic arcs form a beach-line that bounds the locus of all such points
- The beach line is monotone

EDGES

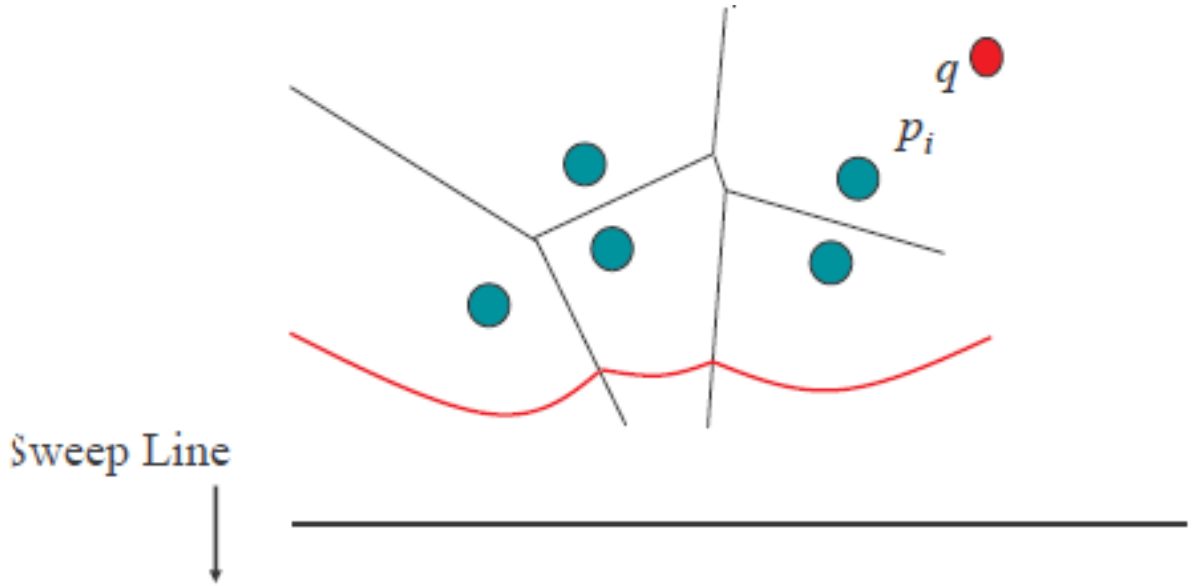
- BREAKPOINTS TRACE OUT VORONOI EDGES.



- THE ONLY WAY IN WHICH A NEW ARC CAN APPEAR ON THE BEACH LINE IS THROUGH A SITE EVENT

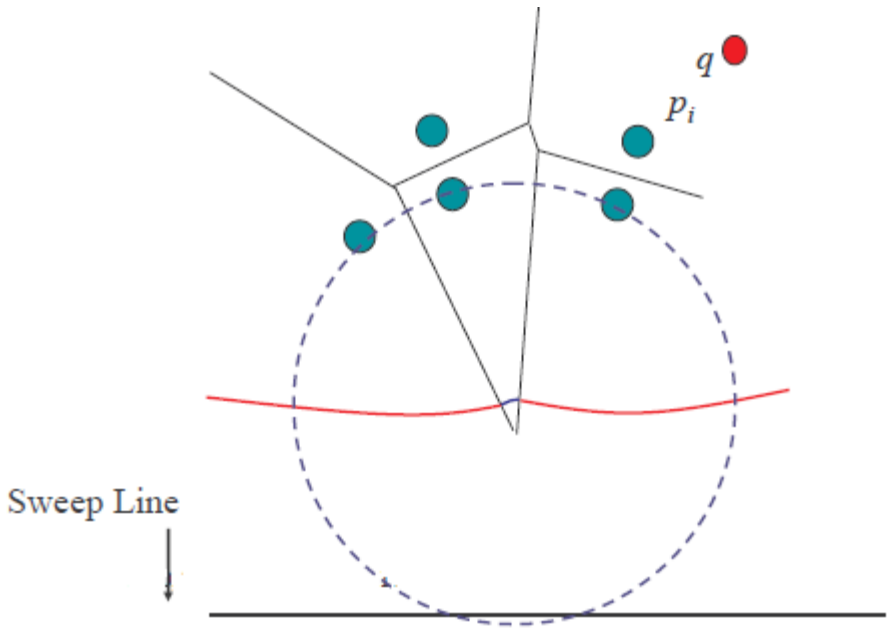
EDGES

- ARCS FLATTEN OUT AS SWEEP LINE MOVES DOWN.



EDGES

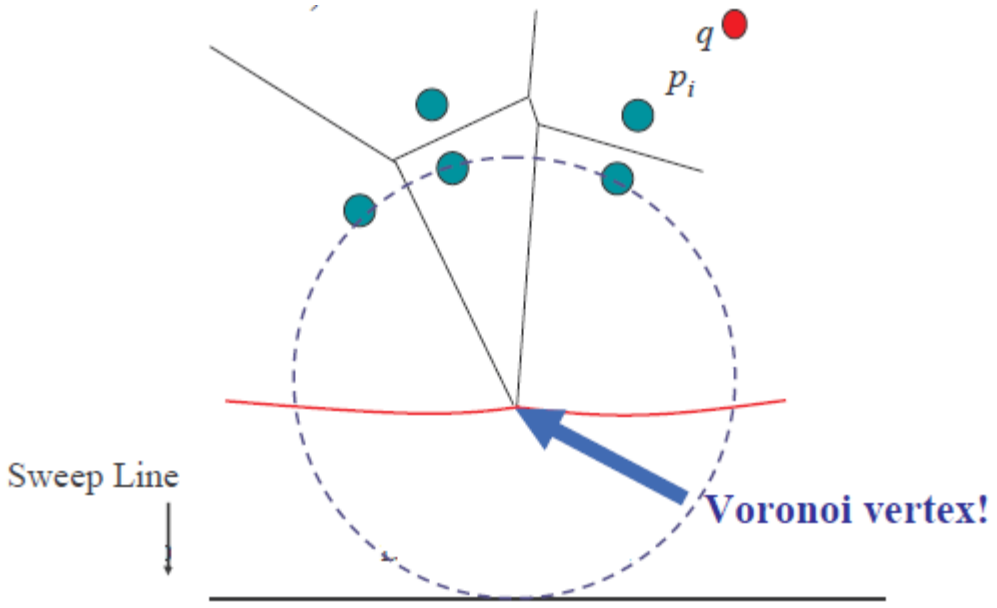
- EVENTUALLY, THE MIDDLE ARC DISAPPEARS.



- THE ONLY WAY IN WHICH AN EXISTING ARC CAN DISAPPEAR FROM THE BEACH LINE IS THROUGH A CIRCLE EVENT

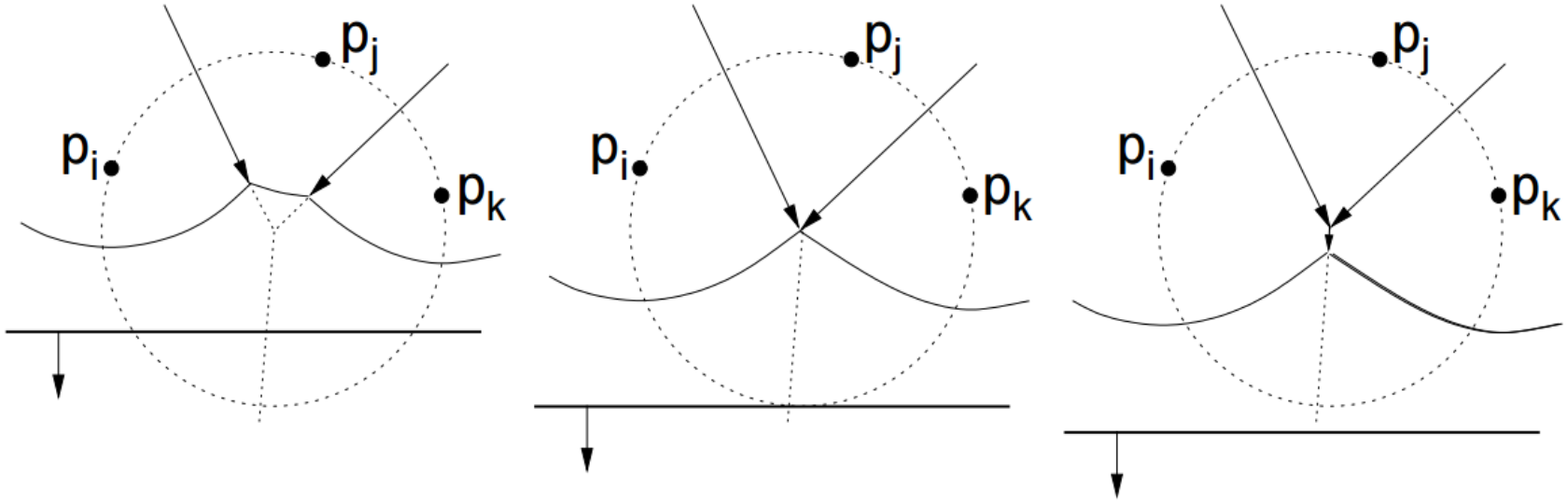
Circle Event

- We have detected a circle that is empty (contains no sites) and touches 3 or more sites.



Circle Event

- We have detected a circle that is empty (contains no sites) and touches 3 or more sites when the sweep line passes the lowest point in the circle.



Beach Line Properties

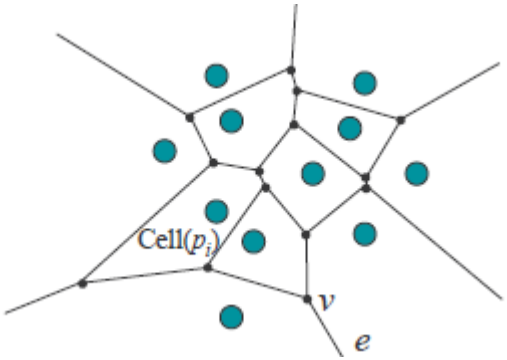
- Voronoi edges are traced by the break points as the sweep line moves down.
  - Emergence of a new break point(s) (from formation of a new arc or a fusion of two existing break points) identifies a new edge
  - Voronoi vertices are identified when two break points meet (fuse).
  - Decimation of an old arc identifies new vertex

Data Structures

- Current state of the Voronoi diagram
  - Doubly-linked list
- The beach line is represented by a balanced binary search tree
- The event queue is implemented as priority queue
  - Priority event queue sorted on decreasing y-coordinate

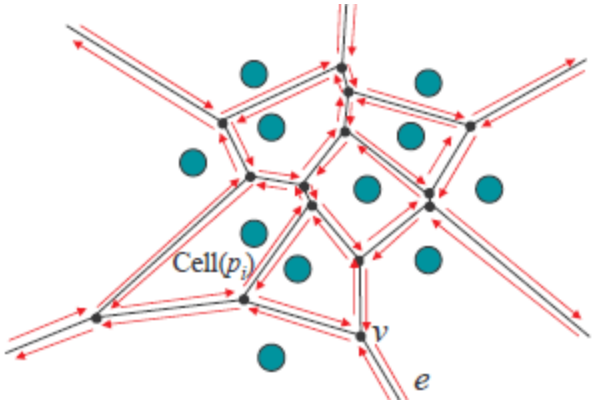
Doubly Linked List (*D*)

- Goal: a simple data structure that allows an algorithm to traverse a Voronoi diagram's segments, cells and vertices



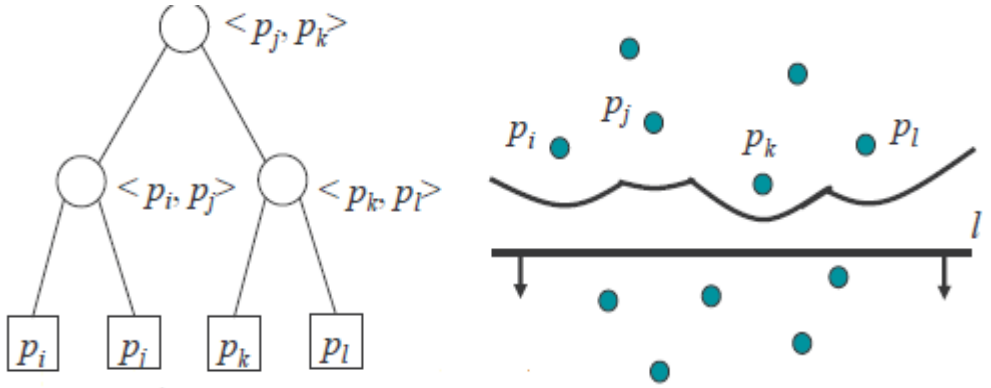
Doubly Linked List (*D*)

- Divide segments into unidirectional half-edges
- A chain of counter-clockwise half-edges forms a cell
- Define a half-edge's "twin" to be its opposite half-edge of the same segment



Balanced Binary Tree

- A balanced binary search tree is used to maintain the status of the beach line.
- Internal nodes represent break points between two arcs
  - Also contains a pointer to the *D* record of the edge being traced
- Leaf nodes represent arcs, each arc is in turn represented by the site that generated it
  - Also contains a pointer to a potential circle event

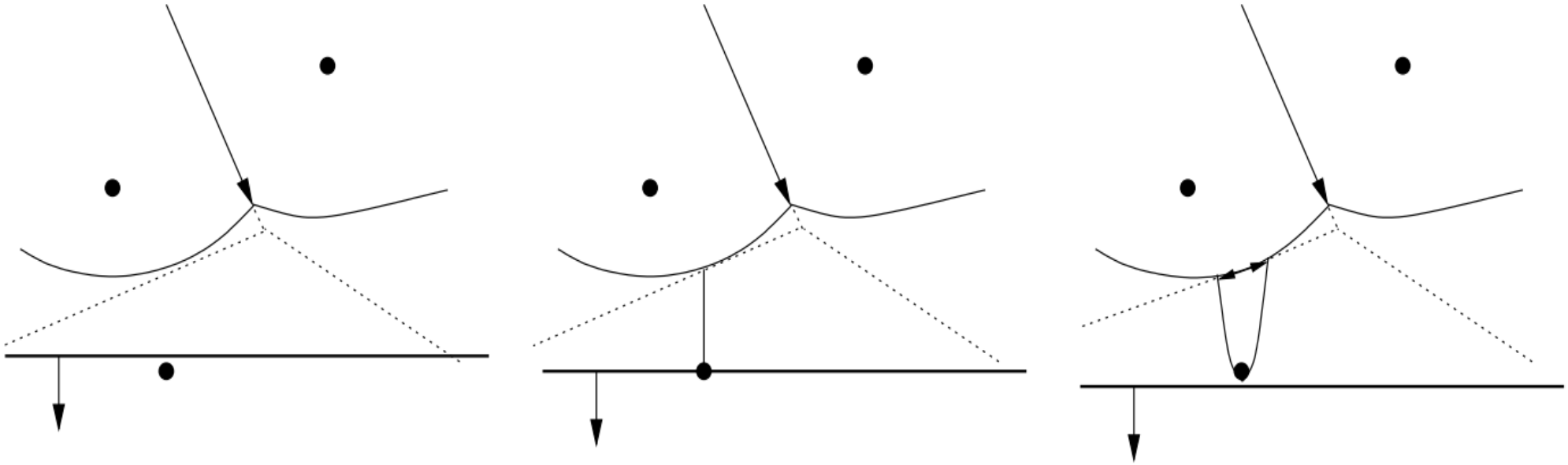


Event Queue (*Q*)

- An event is an interesting point encountered by the sweep line that makes discrete stops at events as it sweeps from top to bottom
- Consists of:
  - Site Events – when the sweep line encounters a new site point
  - Circle Events – when the sweep line encounters the *bottom* of an empty circle touching 3 or more sites
- Events are prioritized based on y-coordinate

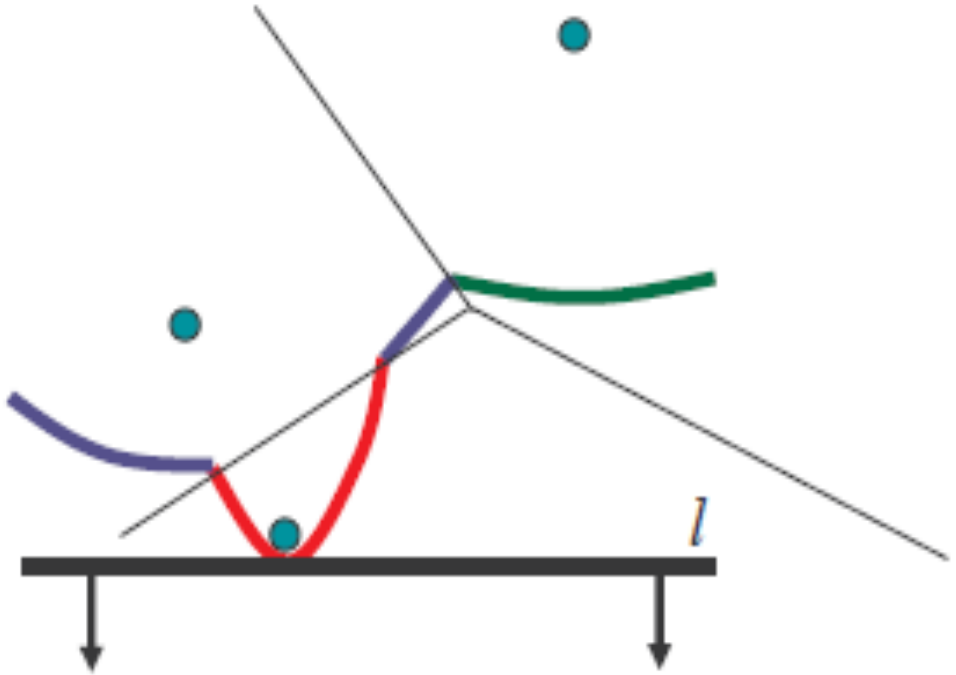
Site Event

- A new arc appears when a new site appears.



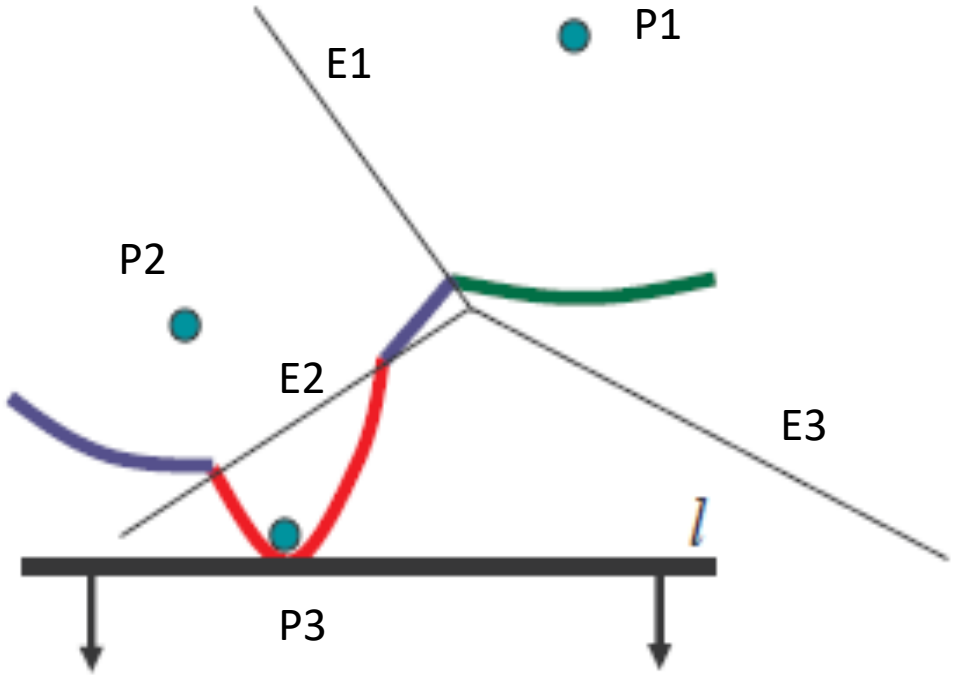
Site Event

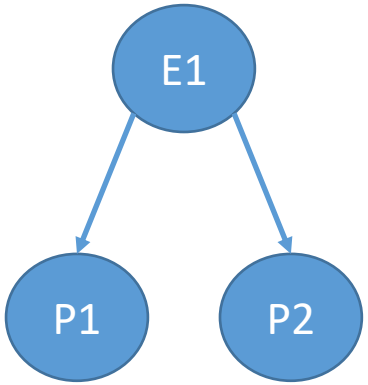
- Original arc above the new site is broken into two arcs
  - Number of arcs on beach line is O(n)

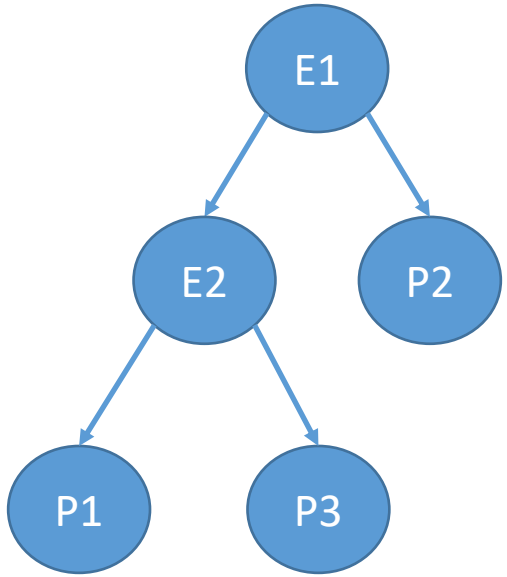


Site Event

- Effects on Binary Tree

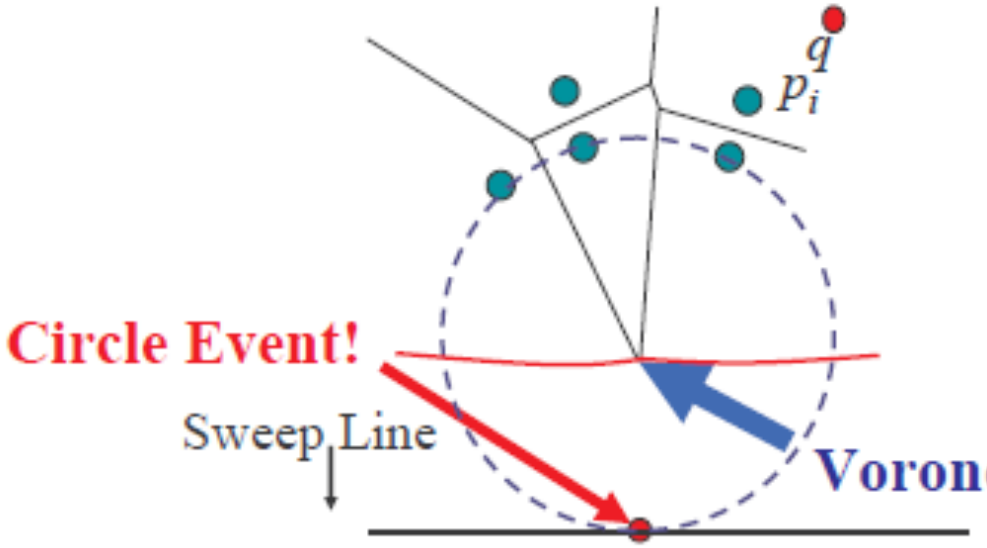






Circle Event

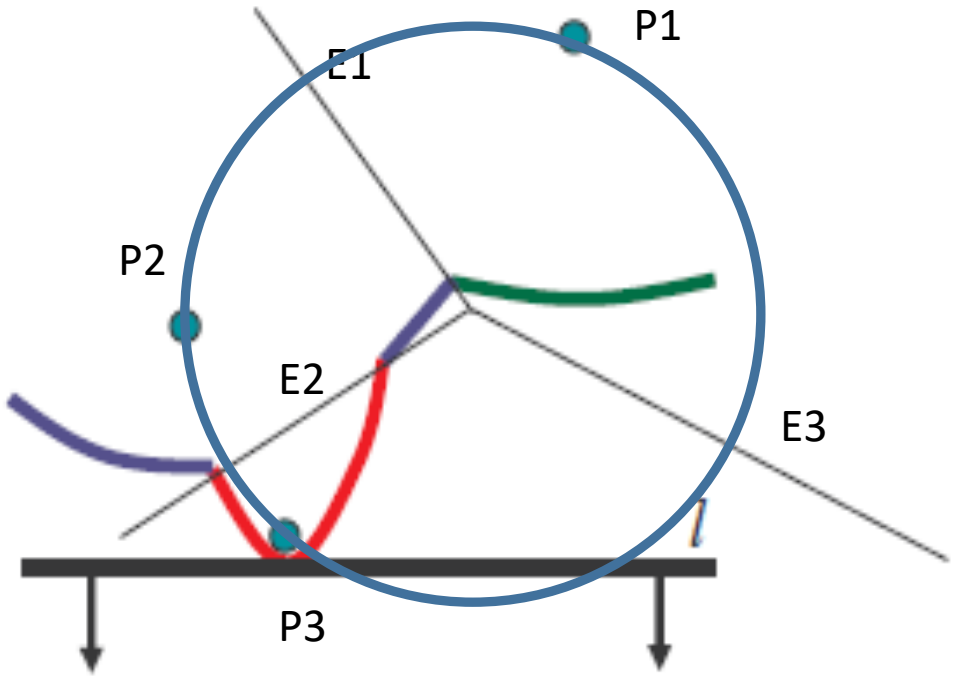
- An arc disappears whenever an empty circle touches three or more sites and is tangent to the circle.

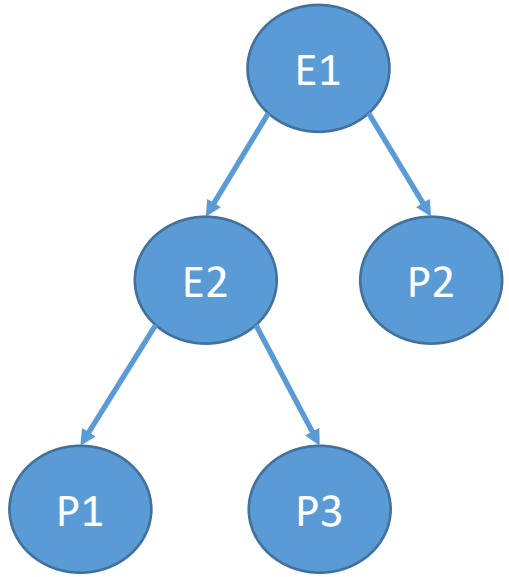


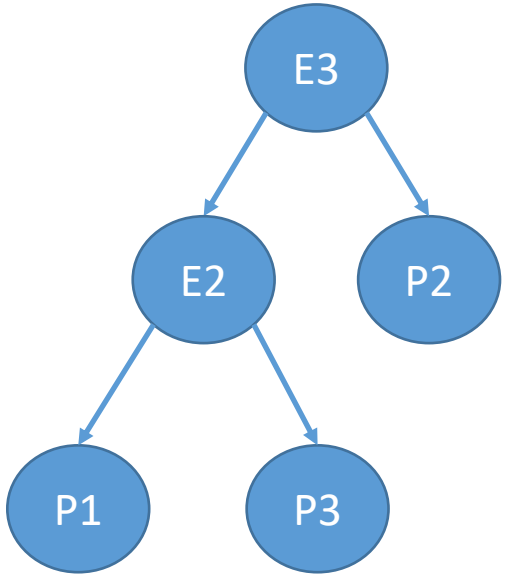
- Sweep line helps determine that the circle is indeed empty.

Circle Event

- Effects on Binary Tree
  - Other circle events will remove nodes from the tree



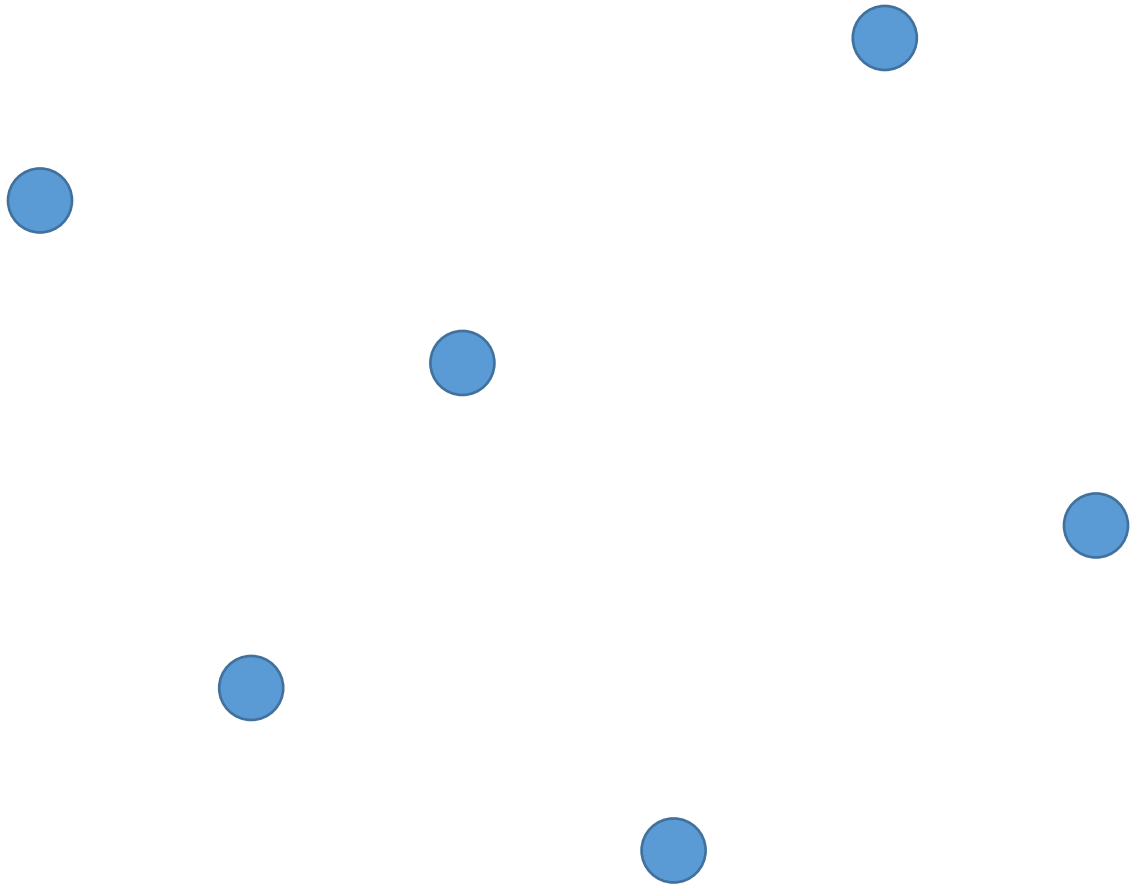




Event Queue Summary

- Site Events are
  - given as input
  - represented by the (x,y)-coordinate of the site point
- Circle Events are
  - represented by the (x,y)-coordinate of the lowest point of an empty circle touching three or more sites
  - computed on the fly (intersection of the two bisectors in between the three sites)
  - "anticipated": these newly generated events may represented by the (x,y)-coordinate of the lowest point of an empty circle touching three or more sites; they can be false and need to be removed later
- Event Queue prioritizes events based on their y-coordinates

EXAMPLE



Summarizing Data Structures

- Current state of the Voronoi diagram
  - Doubly linked list of half-edges, vertices, cell records
- Current state of the beach line
  - Keep track of break points
- Inner nodes of binary search tree; represented by a tuple
  - Keep track of arcs currently on beach line
- Leaf nodes of binary search tree; represented by a site that generated the arc
- Current state of the sweep line
  - Priority event queue sorted on decreasing y-coordinate

Degenerate Cases

- Events in Q share the same y-coordinate
  - Can additionally sort them using x-coordinate
- Circle event involving more than 3 sites
  - Current algorithm produces multiple degree 3 Voronoi vertices joined by zero-length edges
  - Can be fixed in post processing

Degenerate Cases

- Site points are collinear (break points neither converge or diverge)
  - Bounding box takes care of this
- One of the sites coincides with the lowest point of the circle event
  - Do nothing

Handling Site Events

1. Locate the leaf representing the existing arc that is above the new site • Delete the potential circle event in the event queue	O(N log N)
2. Break the arc by replacing the leaf node with a sub tree representing the new arc and break points	O(1)
3. Add a new edge record in the link list	O(1)
4. Check for potential circle event(s), add them to queue if they exist • Store in the corresponding leaf of T a pointer to the new circle event in the queue	O(1)

Handling Circle Events

1. Delete from T the leaf node of the disappearing arc and its associated circle events in the event queue	O(N log N)
2. Add vertex record in doubly link list	O(1)
3. Create new edge record in doubly link list	O(1)
4. Check the new triplets formed by the former neighboring arcs form potential circle events	O(1)

Total Running Time

- Each new site can generate at most two new arcs beach line can have at most $2N - 1$ arcs
- Each "false circle event" can be changed to a real event O(N) events
- Site/Circle Event Handler O(log N)

O(N log N) total running time