

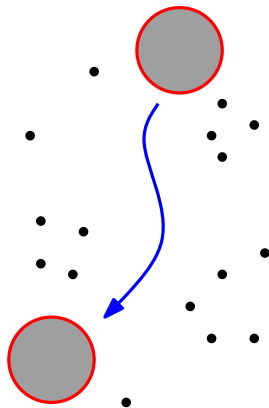
More on Voronoi diagrams

Computational Geometry

Lecture 13: More on Voronoi diagrams

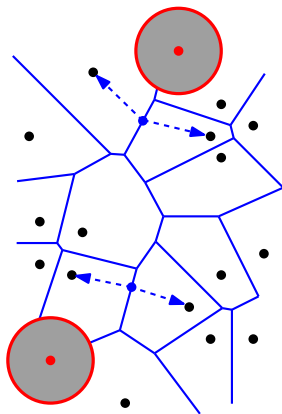
Motion planning for a disc

Can we move a disc from one location to another amidst obstacles?



Motion planning for a disc

Since the Voronoi diagram of point sites is locally “furthest away” from those sites, we can move the disc if and only if we can do so on the Voronoi diagram



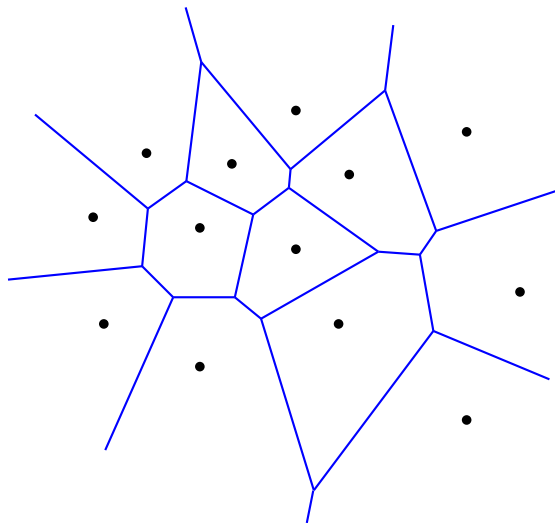
Retraction

Global idea for motion planning for a disc:

1. Get center from start to Voronoi diagram
2. Move center along Voronoi diagram
3. Move center from Voronoi diagram to end

This is called **retraction**

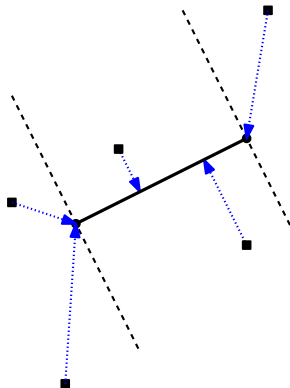
Voronoi diagram of points



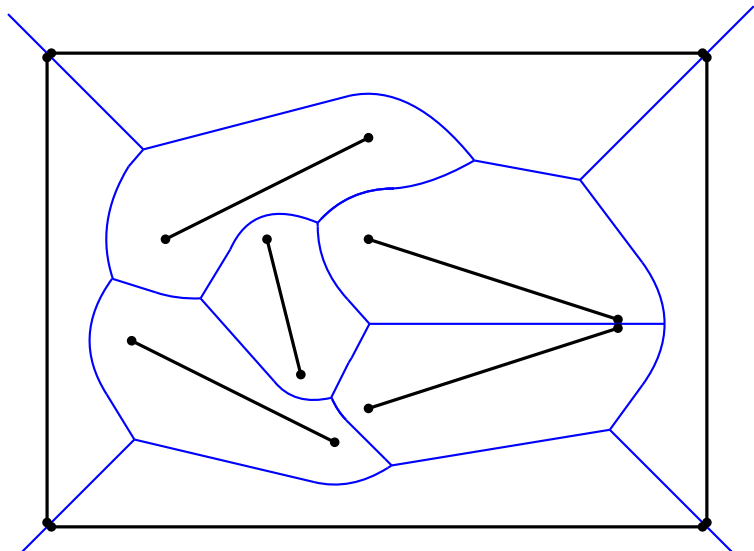
Voronoi diagram of line segments

For a Voronoi diagram of other objects than point sites, we must decide to which point on each site we measure the distance

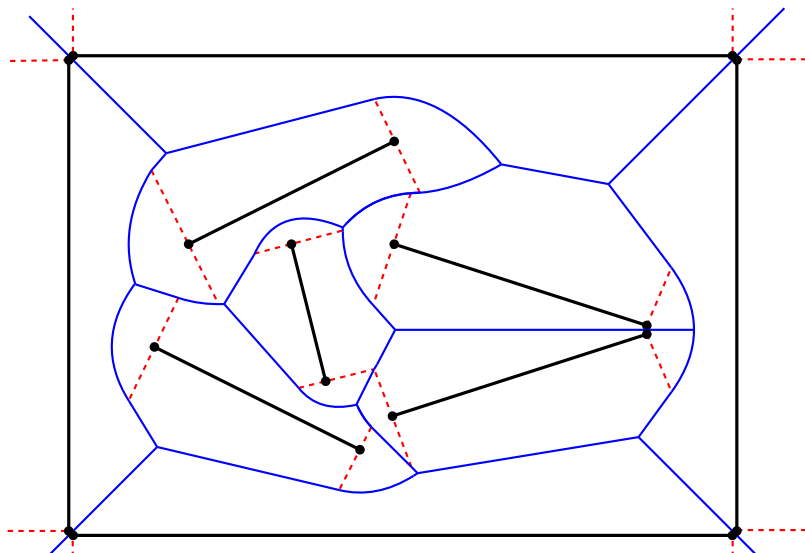
This will be the closest point on the site



Voronoi diagram of line segments

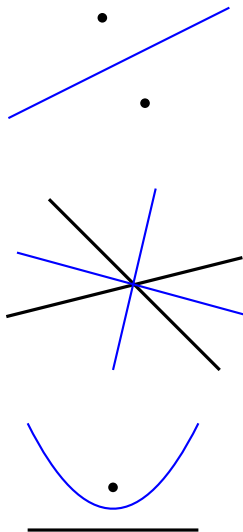


Voronoi diagram of line segments



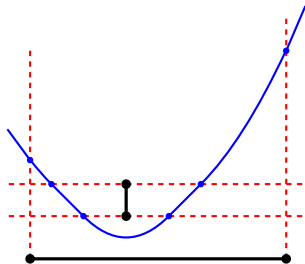
Voronoi diagram of line segments

- The points of equal distance to two points lie on a line
- The points of equal distance to two lines lie on a line (two lines)
- The points of equal distance to a point and a line lie on a parabola



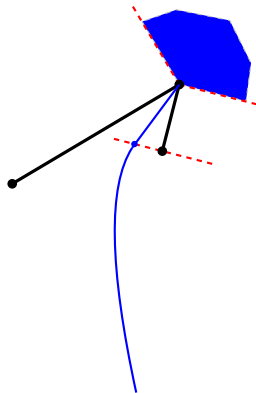
Bisector of two line segments

Two line segment sites have a
bisector with up to 7 arcs



Bisector of two line segments

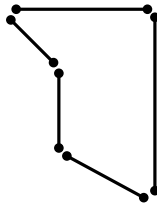
If two line segment sites share an endpoint, their bisector can have an area too



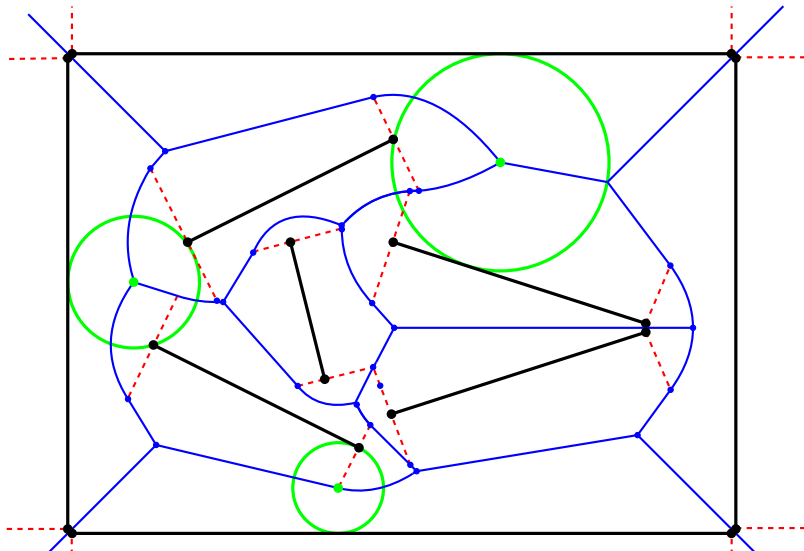
Bisector of two line segments

We assume that the line segment sites are fully disjoint, to avoid complications

We could shorten each line segment from a set of non-crossing line segments a tiny amount



Empty circles



Voronoi vertices

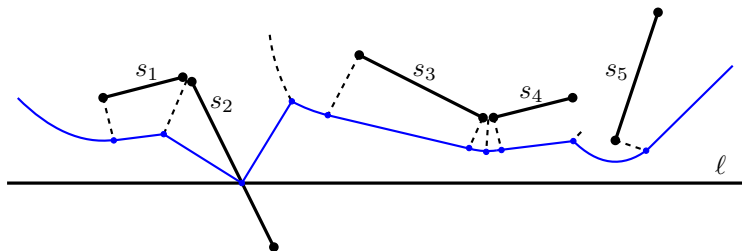
The Voronoi diagram has vertices at the centers of empty circles

- touching three different line segment sites (*degree 3 vertex*)
- touching two line segment sites, one of which it touches in an endpoint of the line segment, and the segment is also part of the tangent line of the circle at that point (*degree 2 vertex*)

At a degree 2 Voronoi vertex, one incident arc is a straight edge and the other one is a parabolic arc

Constructing the Voronoi diagram of line segments

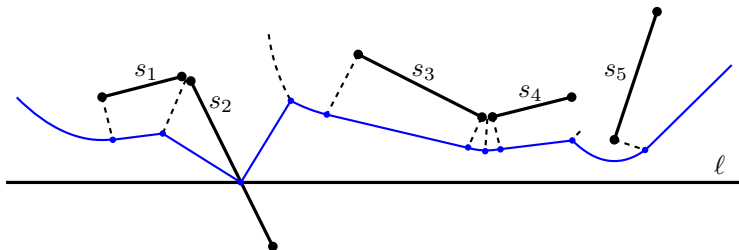
The Voronoi diagram of a set of line segments can be constructed using a plane sweep algorithm



Question: What site defines the leftmost arc on the beach line?

Breakpoints

Breakpoints trace arcs of equal distance to two different sites, or they trace segments perpendicular to a line segment starting at one of its endpoints, or they trace site interiors



Breakpoints

The algorithm uses 5 types of breakpoint:

1. If a point p is closest to **two site endpoints** while being equidistant from them and ℓ , then p is a breakpoint that traces a line segment (as in the point site case)
2. If a point p is closest to **two site interiors** while being equidistant from them and ℓ , then p is a breakpoint that traces a line segment
3. If a point p is closest to **a site endpoint and a site interior** of different sites while being equidistant from them and ℓ , then p is a breakpoint that traces a parabolic arc

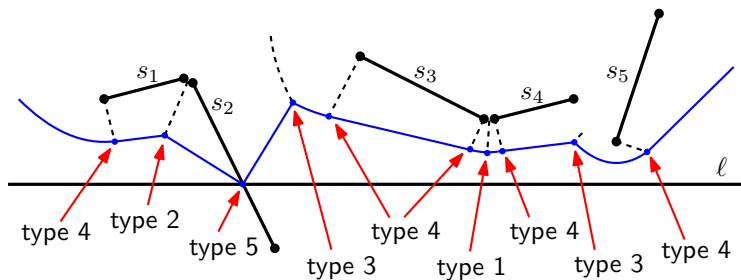
Breakpoints

The algorithm uses 5 types of breakpoint (continued):

4. If a point p is closest to a site endpoint, the shortest distance is realized by a segment that is perpendicular to the line segment site, and p has the same distance from ℓ , then p is a breakpoint that traces a line segment
5. If a site interior intersects the sweep line, then the intersection is a breakpoint that traces a line segment (the site interior)

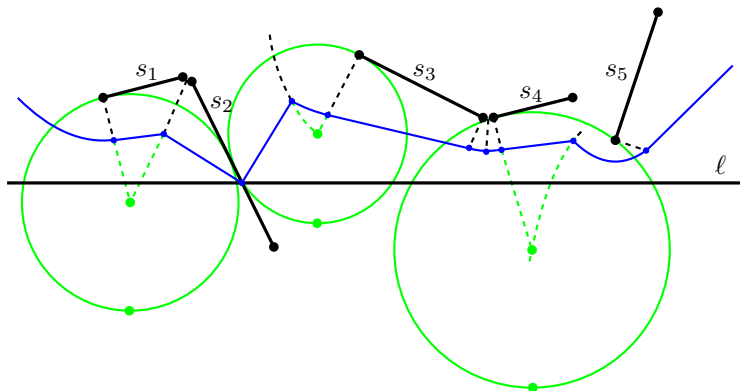
These two types of breakpoint do not trace Voronoi diagram edges but they do trace breaks in the beach line

Events



Events

There are **site events** and **circle events**, but circle events come in different types



Events

The types of **circle events** essentially correspond to the types of breakpoints that meet

Not all types of breakpoint can meet

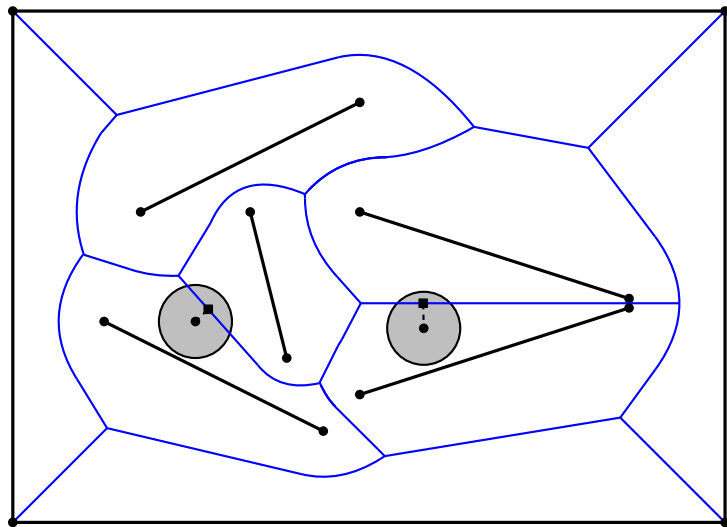
The sweep algorithm

Each event can still be handled in $O(\log n)$ time

There are still only $O(n)$ events

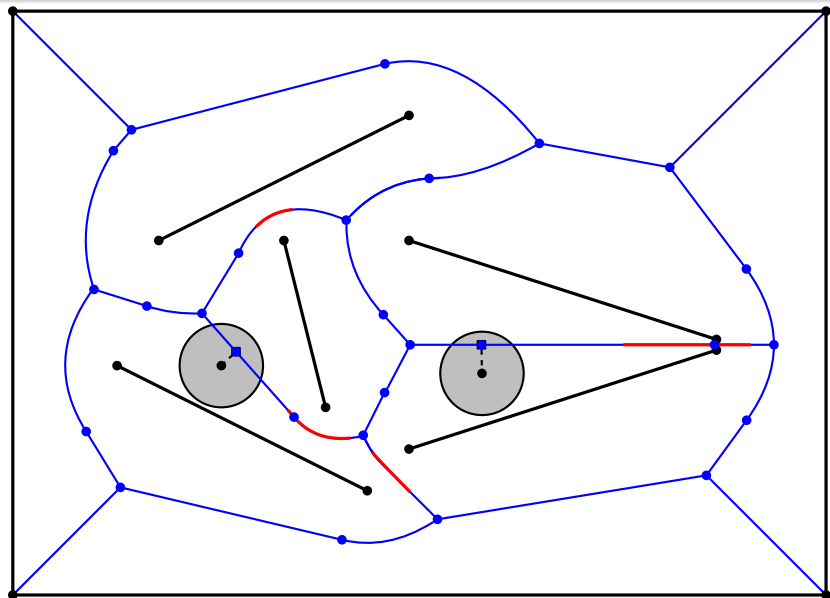
Theorem: The Voronoi diagram of a set of disjoint line segments can be constructed in $O(n \log n)$ time

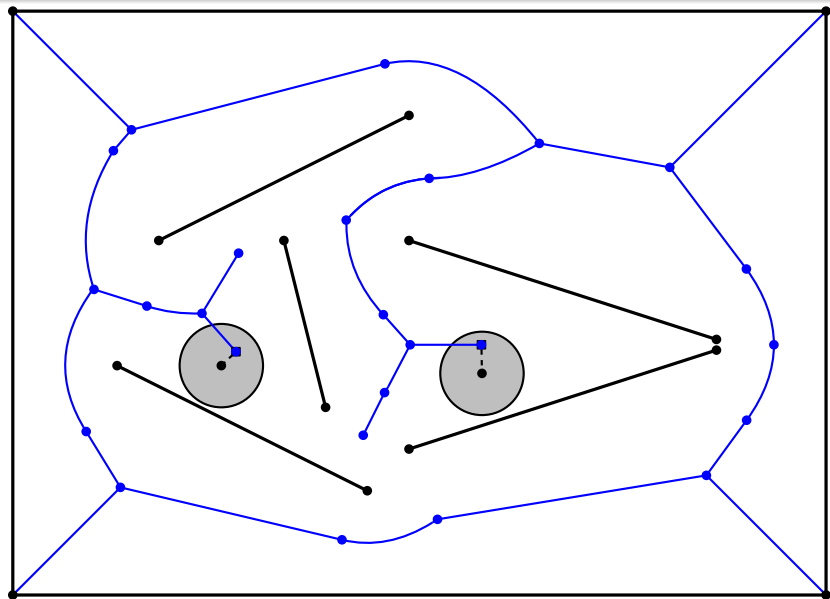
Retraction



Algorithm RETRACTION($S, q_{\text{start}}, q_{\text{end}}, r$)

1. Compute the Voronoi diagram $\text{Vor}(S)$ of S in a bounding box.
2. Locate the cells of $\text{Vor}(P)$ that contain q_{start} and q_{end} .
3. Determine the point p_{start} on $\text{Vor}(S)$ by moving q_{start} away from the nearest line segment in S . Similarly, determine the point p_{end} . Add p_{start} and p_{end} as vertices to $\text{Vor}(S)$, splitting the arcs on which they lie into two.
4. Let \mathcal{G} be the graph corresponding to the vertices and edges of the Voronoi diagram. Remove all edges from \mathcal{G} for which the smallest distance to the nearest sites is $\leq r$.
5. Determine with depth-first search whether a path exists from p_{start} to p_{end} in \mathcal{G} . If so, report the line segment from q_{start} to p_{start} , the path in \mathcal{G} from p_{start} to p_{end} , and the line segment from p_{end} to q_{end} as the path. Otherwise, report that no path exists.





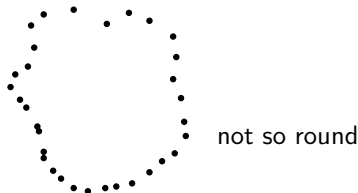
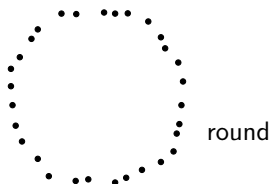
Result

Theorem: Given n disjoint line segment obstacles and a disc-shaped robot, the existence of a collision-free path between two positions of the robot can be determined in $O(n \log n)$ time using $O(n)$ storage.

Testing the roundness

Suppose we construct a perfectly round object, and now wish to test how round it really is

Measuring an object is done with *coordinate measuring machines*, it is a scanner that determines many points on the surface of the object



Coordinate measuring machine

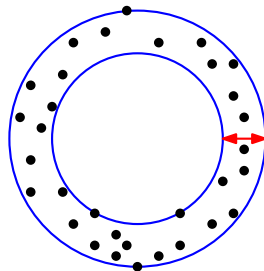


Roundness

The **roundness** of a set of points is the width of the smallest annulus that contains the points

An annulus is the region between two co-centric circles

Its width is the difference in radius



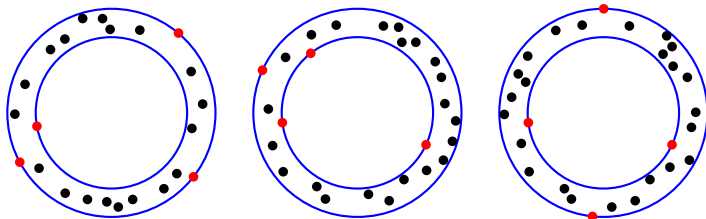
Smallest-width annulus

The smallest-width annulus must have at least one point on C_{outer} , or else we can decrease its size and decrease the width

The smallest-width annulus must have at least one point on C_{inner} , or else we can increase its size and decrease the width

Smallest-width annulus

- C_{outer} contains at least three points of P , and C_{inner} contains at least one point of P
- C_{outer} contains at least one point of P , and C_{inner} contains at least three points of P
- C_{outer} and C_{inner} both contain two points of P



Smallest-width annulus

The smallest-width annulus can not be determined with randomized incremental construction

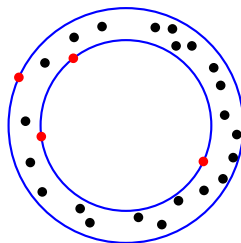
Smallest-width annulus

If we know the center of the smallest-width annulus (the center of the two circles), then we can determine the smallest-width annulus itself (and its width) in $O(n)$ additional time

The cases

Consider **case 2**: C_{inner} contains
(at least) three points of P and
 C_{outer} only one

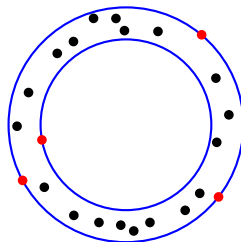
Then the three points on C_{inner}
define an empty circle, and the
center of C_{inner} is a Voronoi
vertex!



The cases

Consider **case 1**: C_{outer} contains
(at least) three points of P and
 C_{inner} only one

Then the three points on C_{outer}
define a “full” circle ...

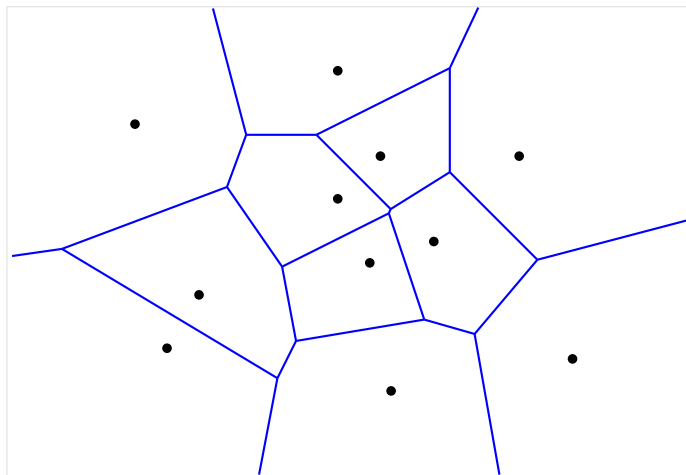


Intermezzo: Higher-order Voronoi diagrams

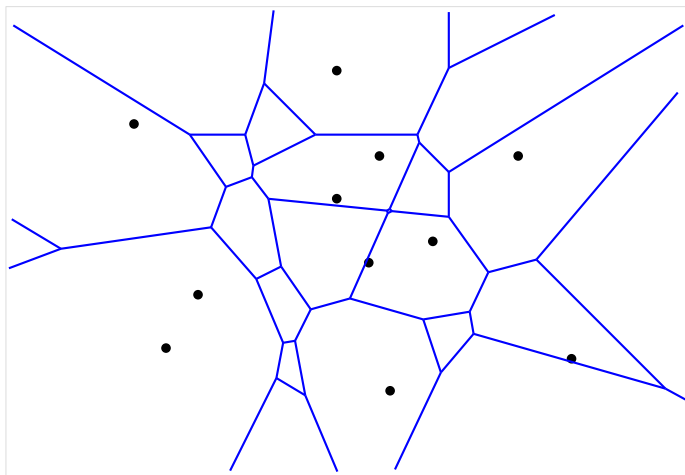
More closest points

Suppose we are interested in the **two** closest points, not only the one closest point, and want a diagram that captures that

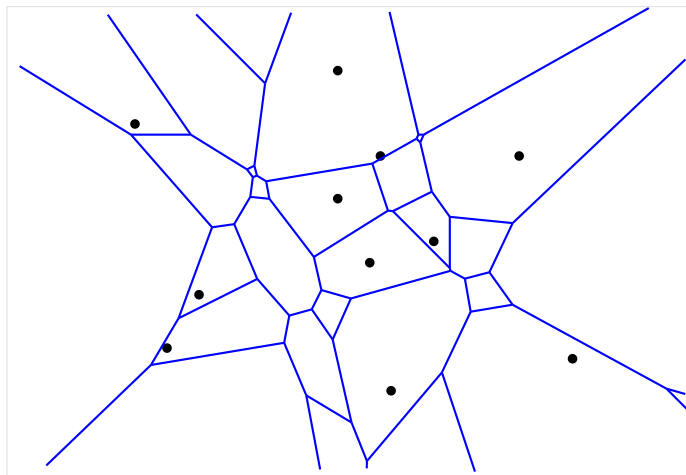
First order Voronoi diagram



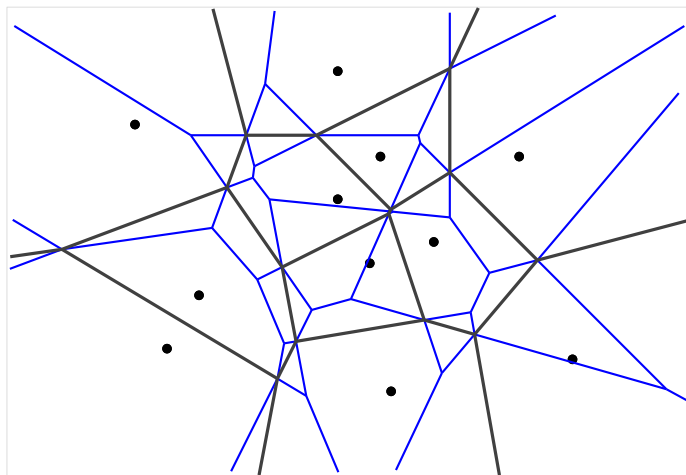
Second order Voronoi diagram



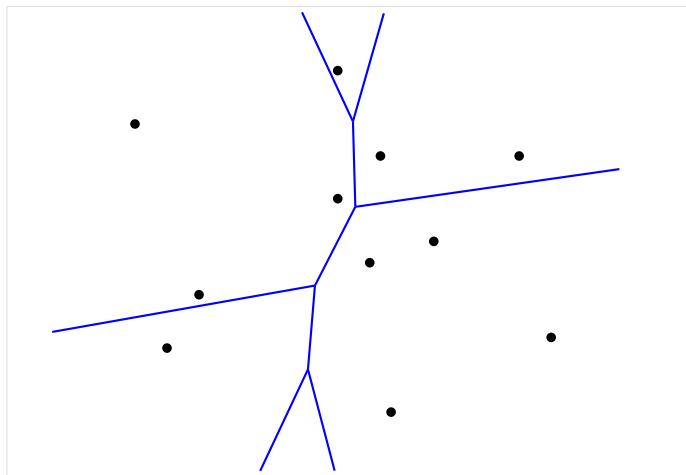
Third order Voronoi diagram



First and second order Voronoi diagram



Tenth order, or farthest-point Voronoi diagram



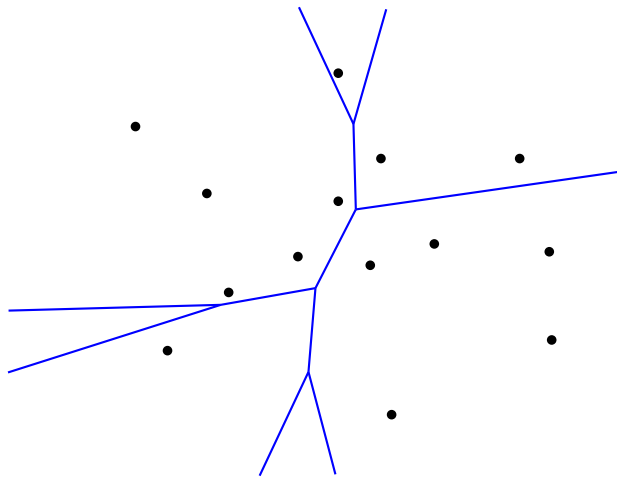
Farthest-point Voronoi diagrams

The **farthest-point Voronoi diagram** is the partition of the plane into regions where the same point is farthest

It is also the $(n - 1)$ -th order Voronoi diagram

The region of a site p_i is the common intersection of $n - 1$ half-planes, so regions are convex, and boundaries are parts of bisectors

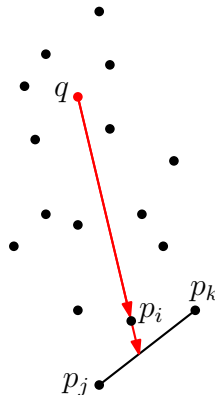
Farthest-point Voronoi diagrams



Farthest-point Voronoi diagrams

Observe: Only points of the convex hull of P can have cells in the farthest-point Voronoi diagram

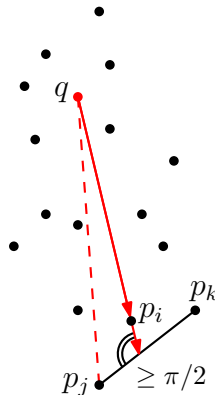
Suppose otherwise ...



Farthest-point Voronoi diagrams

Observe: Only points of the convex hull of P can have cells in the farthest-point Voronoi diagram

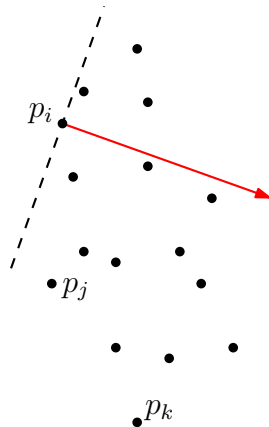
Suppose otherwise ...



Farthest-point Voronoi diagrams

Also observe: All points of the convex hull have a cell in the farthest-point Voronoi diagram

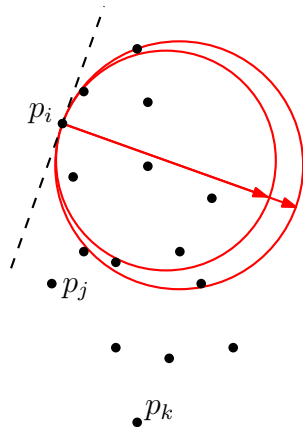
All cells of the farthest-point Voronoi diagram are unbounded



Farthest-point Voronoi diagrams

Also observe: All points of the convex hull have a cell in the farthest-point Voronoi diagram

All cells of the farthest-point Voronoi diagram are unbounded

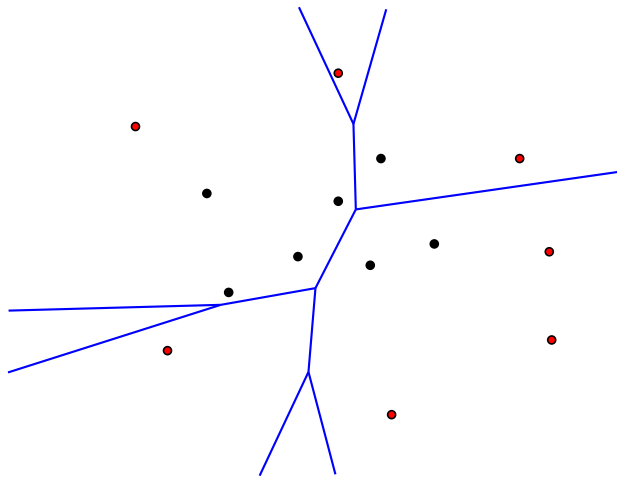


Farthest-point Voronoi diagrams

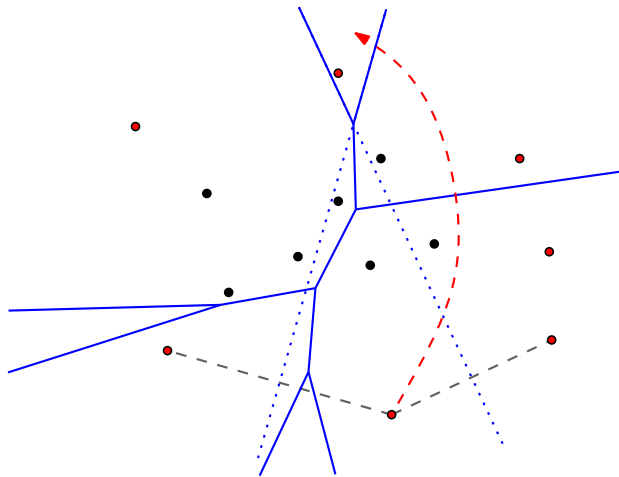
If all cells are unbounded, then the edges of the farthest-point Voronoi diagram form a tree of which some edges are unbounded

Question: For the normal Voronoi diagram, there was one case where its edges are not connected. Does such a case occur for the farthest-point Voronoi diagram?

Farthest-point Voronoi diagrams



Farthest-point Voronoi diagrams



Lower bound

$\Omega(n \log n)$ time is a lower bound for computing the farthest-point Voronoi diagram

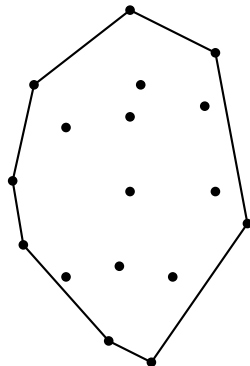
We could use it for sorting by transforming a set of reals x_1, x_2, \dots to a set of points $(x_1, x_1^2), (x_2, x_2^2), \dots$



Construction

So we may as well start by computing the convex hull of P in $O(n \log n)$ time

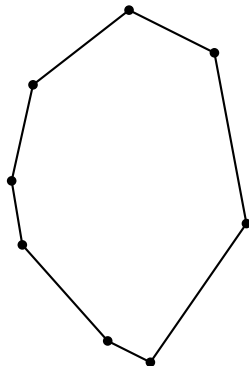
Let p_1, \dots, p_m be the points on the convex hull, forget the rest



Construction

So we may as well start by computing the convex hull of P in $O(n \log n)$ time

Let p_1, \dots, p_m be the points on the convex hull, forget the rest



Construction

The simplest algorithm to construct the farthest-point Voronoi diagram is **randomized incremental construction** on the convex hull vertices

Let p_1, \dots, p_m be the points in *random order*

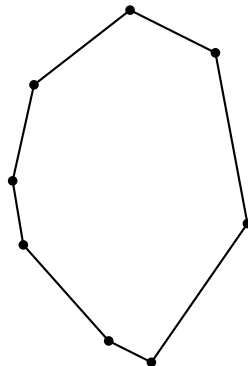
From the convex hull, we also know the *convex hull order*

Construction: phase 1

Phase 1: Remove and Remember

For $i \leftarrow m$ downto 4 do

Remove p_i from the convex hull;
remember its 2 neighbors $cw(p_i)$
and $ccw(p_i)$ (at removal!)

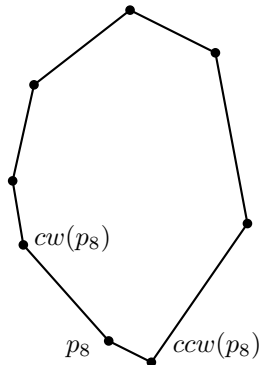


Construction: phase 1

Phase 1: Remove and Remember

For $i \leftarrow m$ downto 4 do

Remove p_i from the convex hull;
remember its 2 neighbors $cw(p_i)$
and $ccw(p_i)$ (at removal!)

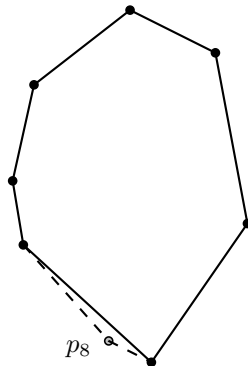


Construction: phase 1

Phase 1: Remove and Remember

For $i \leftarrow m$ downto 4 do

Remove p_i from the convex hull;
remember its 2 neighbors $cw(p_i)$
and $ccw(p_i)$ (at removal!)

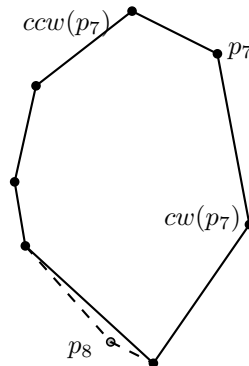


Construction: phase 1

Phase 1: Remove and Remember

For $i \leftarrow m$ downto 4 do

Remove p_i from the convex hull;
remember its 2 neighbors $cw(p_i)$
and $ccw(p_i)$ (at removal!)

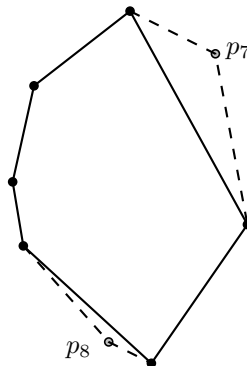


Construction: phase 1

Phase 1: Remove and Remember

For $i \leftarrow m$ downto 4 do

Remove p_i from the convex hull;
remember its 2 neighbors $cw(p_i)$
and $ccw(p_i)$ (at removal!)

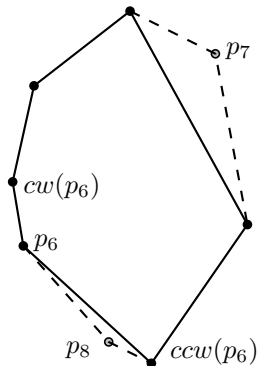


Construction: phase 1

Phase 1: Remove and Remember

For $i \leftarrow m$ downto 4 do

Remove p_i from the convex hull;
remember its 2 neighbors $cw(p_i)$
and $ccw(p_i)$ (at removal!)

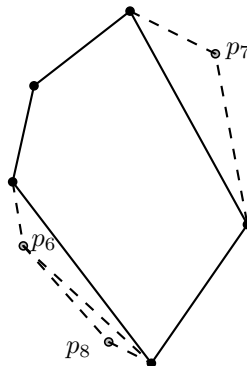


Construction: phase 1

Phase 1: Remove and Remember

For $i \leftarrow m$ downto 4 do

Remove p_i from the convex hull;
remember its 2 neighbors $cw(p_i)$
and $ccw(p_i)$ (at removal!)



Construction: phase 1

Phase 1: Remove and Remember

For $i \leftarrow m$ downto 4 do

Remove p_i from the convex hull;
remember its 2 neighbors $cw(p_i)$
and $ccw(p_i)$ (at removal!)

$p_8, cw(p_8), ccw(p_8)$

$p_7, cw(p_7), ccw(p_7)$

$p_6, cw(p_6), ccw(p_6)$

$p_5, cw(p_5), ccw(p_5)$

$p_4, cw(p_4), ccw(p_4)$

p_3, p_2, p_1

Construction: phase 2

Phase 2: Put back and Construct

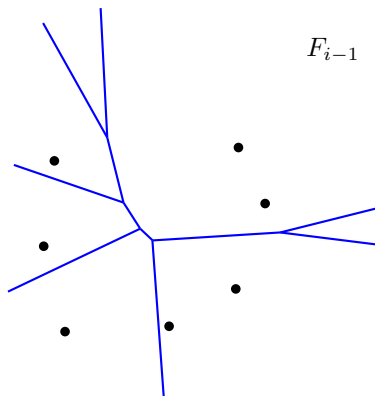
Construct the farthest-point Voronoi diagram F_3 of p_3, p_2, p_1

For $i \leftarrow 4$ to m do

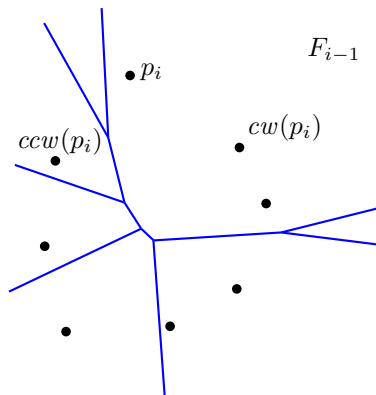
 Add p_i to the farthest-point Voronoi diagram
 F_{i-1} to make F_i

We simply determine the cell of p_i by traversing F_{i-1} and
update F_{i-1}

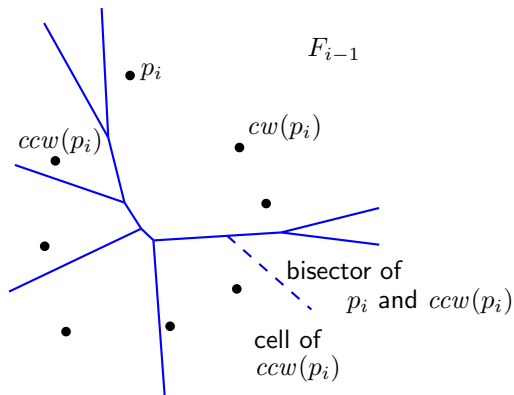
Construction: phase 2



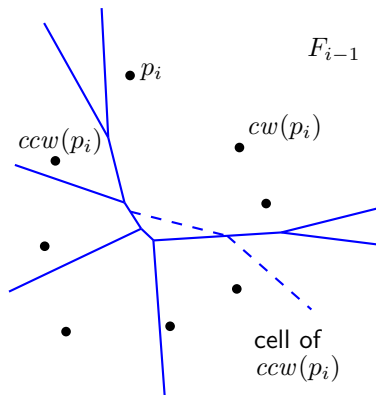
Construction: phase 2



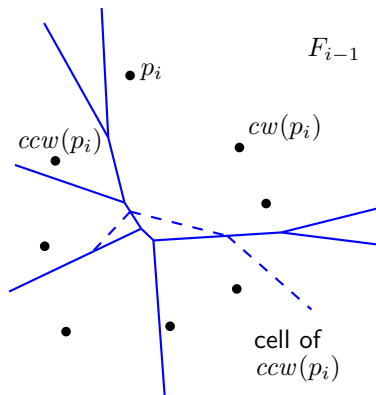
Construction: phase 2



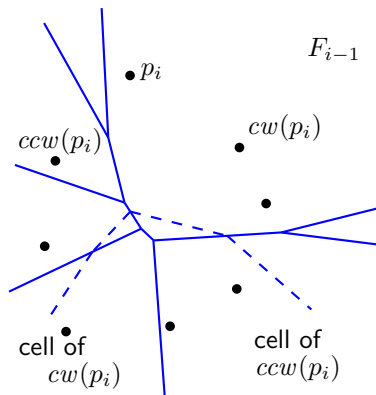
Construction: phase 2



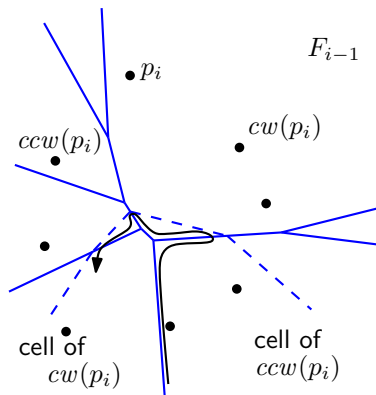
Construction: phase 2



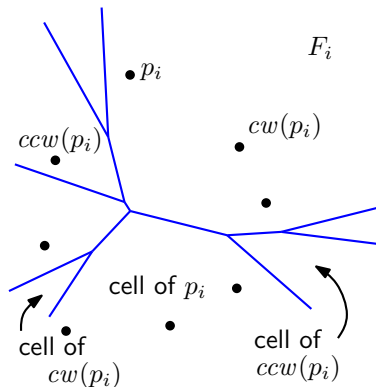
Construction: phase 2



Construction: phase 2



Construction: phase 2



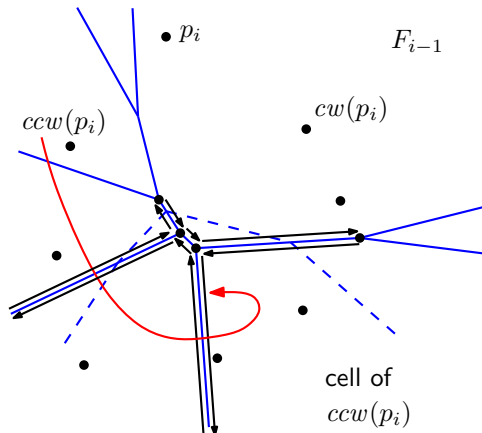
Construction: phase 2

The implementation of phase 2 requires a representation of the farthest-point Voronoi diagram

We use the doubly-connected edge list (ignoring issues due to half-infinite edges)

For any point among p_1, \dots, p_{i-1} , we maintain a pointer to the most counterclockwise bounding half-edge of its cell

Construction: phase 2



Analysis of randomized incremental construction

Due remembering $ccw(p_i)$, we have $ccw(p_i)$ in $O(1)$ time, and get the first bisector that bounds the cell of p_i

Due to the pointer to the most counterclockwise half-edge of the cell of $ccw(p_i)$, we can start the traversal in $O(1)$ time

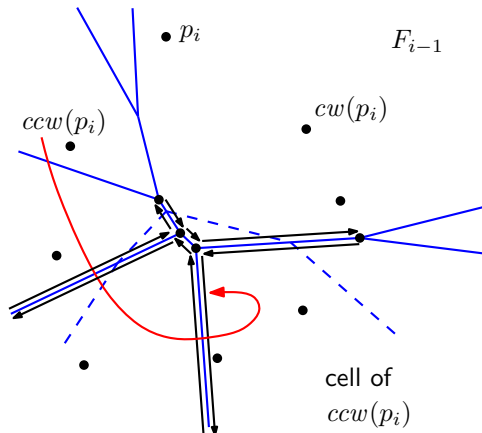
Analysis of randomized incremental construction

If the cell of p_i has k_i edges in its boundary, then we visit $O(k_i)$ half-edges and vertices of F_{i-1} to construct the cell of p_i

Also, we remove $O(k_i)$ vertices and half-edges, change (shorten) $O(k_i)$ half-edges, and create $O(k_i)$ half-edges and vertices

\Rightarrow adding p_i takes $O(k_i)$ time, where k_i is the complexity of the cell of p_i in F_i

Analysis



Backwards analysis

Backwards analysis:

Assume that p_i has already been added and we have F_i

Each one of the i points had the same probability of having been added last

The *expected time* for the addition of p_i is linear in the *average complexity* of the cells of F_i

Backwards analysis

The farthest-point Voronoi diagram of i points has at most $2i - 3$ edges (fewer in degenerate cases), and each edge bounds exactly 2 cells

So the average complexity of a cell in a farthest-point Voronoi diagram of i points is

$$k_i \leq \frac{2 \cdot (2i - 3)}{i} = \frac{4i - 6}{i} < 4$$

The expected time to construct F_i from F_{i-1} is $O(1)$

Result

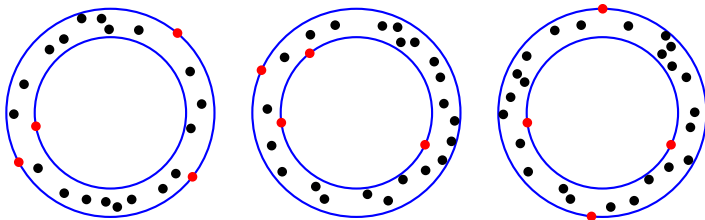
Due to the initial convex hull computation, the whole algorithm requires $O(n \log n)$ time, plus $O(m)$ expected time

Theorem: The farthest-site Voronoi diagram of n points in the plane can be constructed in $O(n \log n)$ expected time. If all points lie on the convex hull and are given in sorted order, it takes $O(n)$ expected time

End of intermezzo; back to smallest-width annulus

Smallest-width annulus

- C_{outer} contains at least three points of P , and C_{inner} contains at least one point of P
- C_{outer} contains at least one point of P , and C_{inner} contains at least three points of P
- C_{outer} and C_{inner} both contain two points of P



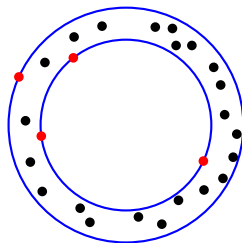
Smallest-width annulus

If we know the center of the smallest-width annulus (the center of the two circles), then we can determine the smallest-width annulus itself (and its width) in $O(n)$ additional time

Case 2

Consider **case 2**: C_{inner} contains
(at least) three points of P and
 C_{outer} only one

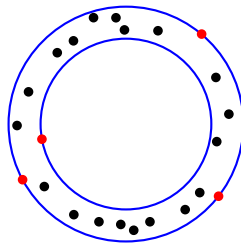
Then the three points on C_{inner}
define an empty circle, and the
center of C_{inner} is a Voronoi
diagram vertex!



Case 1

Consider **case 1**: C_{outer} contains
(at least) three points of P and
 C_{inner} only one

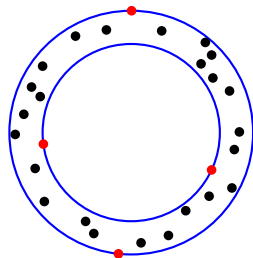
Then the three points on C_{outer}
define a “full” circle, and the
center of C_{outer} is a farthest-point
Voronoi diagram vertex!



Case 3

Consider **case 3**: C_{outer} and C_{inner}
each contain two points of P

Then the two points on C_{inner} define
a set of empty circles and the two
points of C_{outer} define a set of full
circles

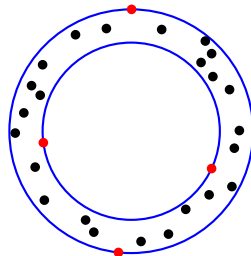


Case 3

The two points on C_{inner} define a set of empty circles whose centers lie on the Voronoi edge defined by these two points

The two points of C_{outer} define a set of full circles whose centers lie on the farthest-point Voronoi edge defined by these two points

The center lies on an intersection of an edge of the Voronoi diagram and an edge of the farthest-point Voronoi diagram!



Algorithm

To solve **case 3**:

1. Compute the Voronoi diagram of P
2. Compute the farthest-point Voronoi diagram of P
3. For each pair of edges, one of each diagram
 - defined by p, p' of the Voronoi diagram and by q, q' of the farthest-point Voronoi diagram
 - determine the annulus for p, p' and q, q' if the edges intersect
4. Keep the smallest-width one of these

This takes $O(n^2)$ time

Algorithm

To solve **case 1**:

1. Compute the farthest-point Voronoi diagram of P
2. For each vertex v of the FPVD
 - determine the point p of P that is closest to v
 - determine C_{outer} from the points defining v
 - determine the annulus from p , v , and C_{outer}
3. Keep the smallest-width one of these

This takes $O(n^2)$ time

Algorithm

To solve **case 2**:

1. Compute the Voronoi diagram of P
2. For each vertex v of the Voronoi diagram
 - determine the point p of P that is farthest from v
 - determine C_{inner} from the points defining v
 - determine the annulus from p , v , and C_{inner}
3. Keep the smallest-width one of these

This takes $O(n^2)$ time

Result

Theorem: The roundness, or the smallest-width annulus of n points in the plane can be determined in $O(n^2)$ time