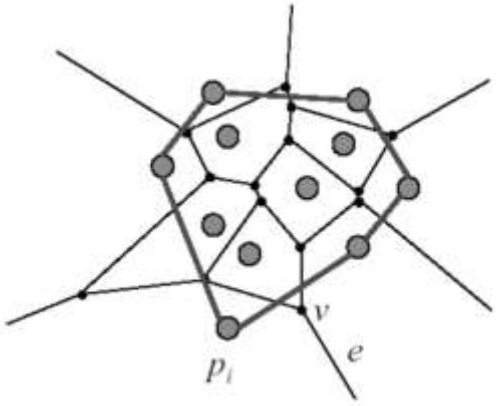
Voroni Diagrams



- Records information about what is closed to what.
- We denote the set of points $P = \{p_1, p_2, \dots, p_n\}$ as *sites*.

Voroni region

$V(p_i) = \{q \mid \forall j \neq i: \|p_i q\| < \|p_j q\|\}$ – the region of points that are closer to p_i than to any other sites p_j ($i \neq j$).

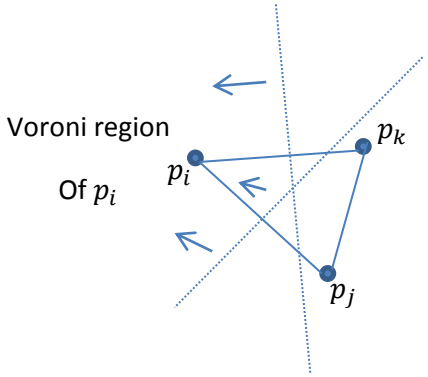
The subdivision of Voroni region imposes a partition of the space.

The definition of the Voroni diagram can equivalently be defined by intersections of half spaces.

Half-plain: a space bounded on one side by a line in the space.

Bisector: the line perpendicular to the segment between two points p_i, p_j. We denote the half-plain defined by the bisector between p_i and p_j by $h(p_i, p_j)$ – and any point there is closer to p_i than to p_j. That half plain is a **convex** set.

In the case of 3 points:

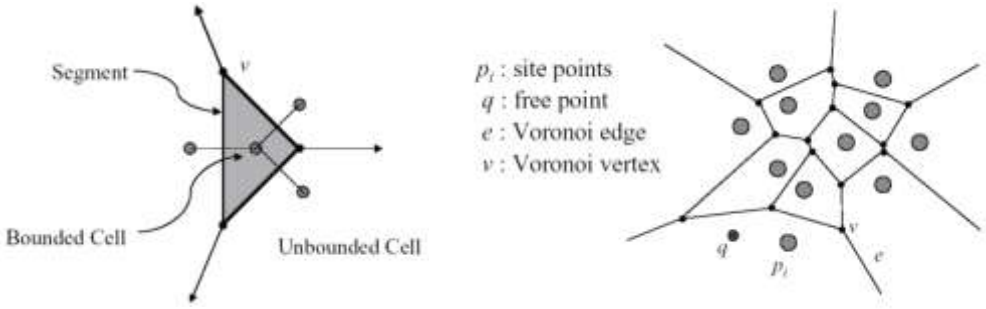


The intersection of $h(p_i, p_j)$ and $h(p_i, p_k)$.

In the general case: $V(p_i) = \bigcap_{\substack{p_i \in P \\ i \neq j}} h(p_i, p_j)$ – the intersection of all half-spaces as defined above.

Since any half-space is convex, the intersection is also convex.

Voroni diagram: If we remove the **interior** of the open sets which are the regions, the boundary of all regions together form the Voroni diagram.



For any shape we can look at the Voroni diagram of it as a skeleton of the shape (imagine we burn the regions / shapes – the points in which all "fires" meet is the Voroni diagram).

Delaunay Triangulation: The Dual of all Voroni diagram. If we know one, we can build the other from it.

General position assumptions:

- Degree: At most 3 points will be sitting in a circle. This means that any Voroni vertex will have a degree at most 3.
- Convex hull: looking at the faces (Voroni regions), they are convex.

Given n sites, we can add another vertex in infinity and connect all infinity edges of the VD and we will get a planar subdivision with n faces. The number of vertices will be at most $2n-5$ and the number of edges - $3n-6$ (by Euler's formula).

Computing VD:

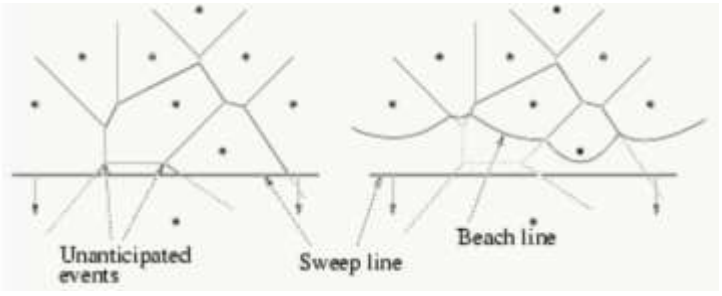
Naïve algorithm: computing each $V(p_i)$ by the intersection with all $n-1$ corresponding bisector half-planes $h(p_i, p_j), i \neq j$ - $O(n^2 \lg n)$.

Fortune's algorithm

This is a plane-sweep algorithm that at any given point of time it saves:

- The sweep structure
- Events – not trivial as in the segment intersection algorithm.

To not allow infinity edges, and we know the number of edges is linear, we can bound those edges.



How can the sweep algorithm know the existence of this vertex until it sees the site? By the time it sees it, it's too late – the events are unanticipated. In the diagram: only when we get to the lowest vertex we will know the events marked as "unanticipated events".

The **beach line** participates in our sweep line structure. The vertices of the beach line:

- Have the same distance from the 2 sites adjacent to the segment that passes through that vertex.
- Will have the same distant from the sweep line as the distant from the sites above.

Meaning, if we draw a circle centered at those vertices with the two sites mentioned, the circle will be tangent to the sweep line.

As we advance the sweep line, the regions behind the beach line (those who don't change) are set for the rest of the algorithm and will not change.

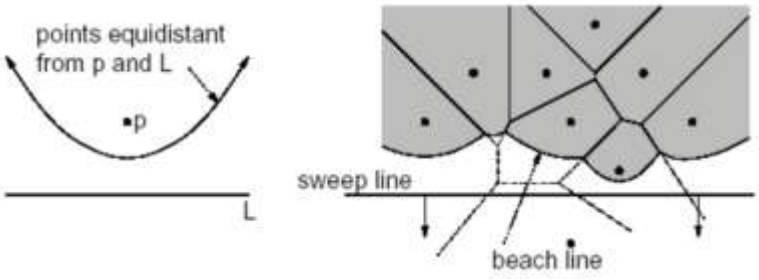
Instead of distorting the boundaries of the regions, we will distort the sweep line. The position of the sweep line and the beach line defines our structure.

Two types of events:

- Site events – know ahead.
- Vertex events – might be unanticipated.

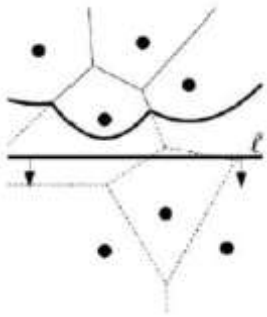
The portion above the sweep line will be maintained in a DCEL.

Every point that is part of the beach line have the same distance from the sweep line as one of the sites. Every site p will have its own parabola.



The beach line is the lower envelope of all those parabolas. In the degenerate case when p is on l, the parabola is actually the perpendicular of l on p.

Note that any site may have more than 1 segment of its corresponding parabola to maintain – how many of those could there be? That number will affect the complexity. The beach line is a x-monotone object.



The algorithm:

- We track the breakpoints along the edges of the VD.
- On the move of the sweep line, the parabolas constructing the beach line constantly change.

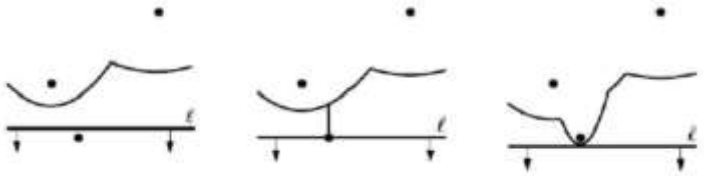
Status:

- Current location y-coordinate of the sweep line.
- Left to right set of sites that define the beach line.
- The parabolic arcs can be calculated from the above, no need to hold them.

Events:

Site events:

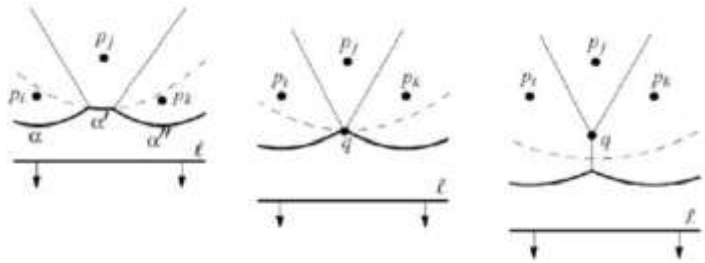
A new arc is inserted to the structure.



Vertex events:

The moment 3 arcs become adjacent on the beach line, we know at what distance (by calculating the circle of their 3 sites) the vertex will be.

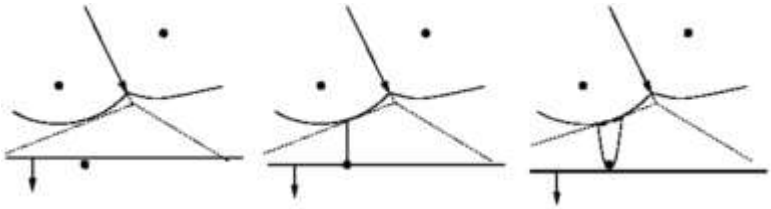
The distance $p_iq = p_jq = p_kq$.



All the complications of updating the structure are linear.

Reminder: we use a bounding box, and a DCEL representation.

Site events:



- Known ahead of time, as we get them we sort them.

In the worst case, when all sites participate in the beach line, how many active segments can we have? at most $2n - 1$:

When we encounter a new site, it shoots a ray up. Any site may shoot a ray to partition its previous parabola, so we get this number.

Whenever a new site is encounter we need to identify where it will be inserted, and since we have $O(n)$ arcs, we can hold the active sites in a structure that will allow logarithmic insert operations.

Generation of the vertex events is constant in the number of site events.

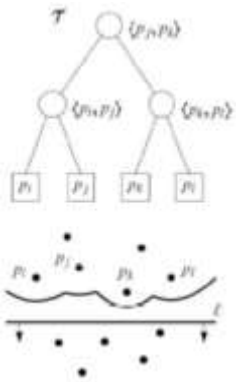
Vertex events:

Those are generated dynamically as the algorithm runs. Each event is generated locally. When we process a new site, we make sure the beach line is maintained properly, and make sure we catch all vertex events that we can now discover (in constant time).

Let p_i, p_j, p_k be adjacent sites, i.e. the corresponding segments for p_i, p_j share a vertex, and p_j, p_k share one as well – a potential for creating a vertex event. The moment the sweep line gets to the point that is tangent to the circle that goes through all p_i, p_j, p_k then this is an event – and we can compute the vertex.

From that point on p_i, p_k are adjacent, and if p_h, p_l are before p_i and after p_k respectively, there might be a new event after a new vertex is found and p_j is removed.

Beach line:



- The beach line is represented by a dictionary (balanced tree).
- We do not explicitly store the parabolic arcs, we can compute them from the sites and the sweep line.
- The breakpoint, center of the circle that goes through p_i, p_j and tangent to the sweep line l, can be computed from those.

Supported operations:

- Given a location of the sweep line, determine the arc of the breach line that intersects a given vertical line – simply apply a binary search to find p_j.
- Successor and predecessor can help find p_i, p_k to complete the calculation.
- Deleting an arc.

Every operation is logarithmic

Event queue:

- A priority queue with the ability to insert and delete events – can use a heap. The key: max y-coordinate.
- Once we have a vertex, we need to insert it to the DCEL – constant given p_i, p_j, p_k (knowing the site means knowing the face).

Vertex event:

p_i, p_j, p_k are sites that create the event. We know that when l gets to the position, we have a vertex that is distant the same from all the above. At this point p_i, p_k become adjacent on the beach line. We initiate the edge that starts at that vertex downwards. The region of p_j is not relevant anymore.

If we have $p_{i-1}, p_i, p_j, p_k, p_{k+1}$, p_j may create a vertex event in one of the three consecutive triplets. But, once p_j forms a vertex with p_i, p_k, the other two events it might have been taking part of, are irrelevant and we drop these 2 events.

Running time:

- Each event is $O(1)$ processing time plus a constant number of accesses to various structures.
- Each access is $O(\lg n)$ and the structures are all of size $O(n)$.
- The total running time is then $O(n \lg n)$.

The total complexity is that of creating the DCEL times how much each costs – since the DCEL is linear, and each operation is $O(\lg n)$ we end up with $O(n \lg n)$.