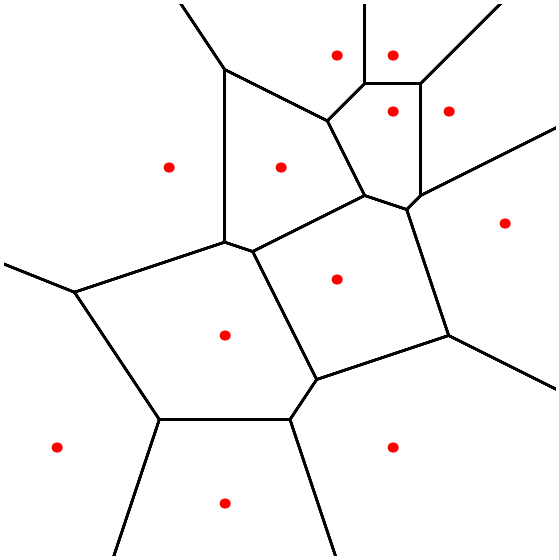


1 Voronoi Diagrams

Given a set of sites (points) in the plane, $s_1, s_2, \ldots s_n$ the *Voronoi diagram* partitions the plane into regions where the region associated with s_i is the set of points in the plane that are closer to site s_i than any other. Let R_i denote the region associated with s_i. The diagram above shows a Voronoi diagram with twelve sites.

The Voronoi regions are convex polygons. To see this note that R_i is the intersection of $n-1$ half-planes, one for each other site j. It's the half space of the plane (bounded by the perpendicular bisector between s_i and s_j) of the points closer to s_i than to s_j. So the Voronoi regions are (possibly infinite) convex polygons. Because the Voronoi diagram is a planar graph, the number of vertices and edges in it is linear in n (by virtue of Euler's formula).

We will talk about Voronoi regions (mentioned above), Voronoi edges (boundaries between Voronoi regions) and Voronoi vertices (where edges meet other edges).

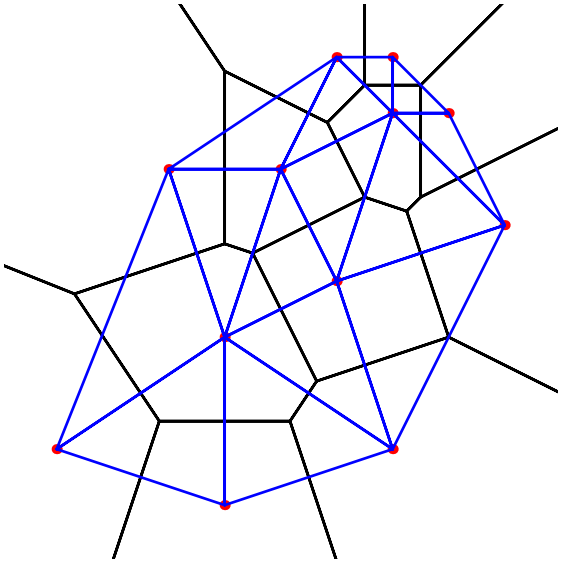
When we say that two Voronoi regions have a "boundary in common" or "share a boundary" we mean a boundary of non-zero length. So, for example, if there are four sites that are at the corners of a square, then the vertex of the Voronoi diagram has degree four. We don't consider diagonally opposite regions to have a boundary in common.

Unless otherwise specified, in these notes we're going to assume that the Voronoi diagram has only vertices of degree three. (This will happen if no four of the sites are co-circular.)

2 Delaunay Triangulations

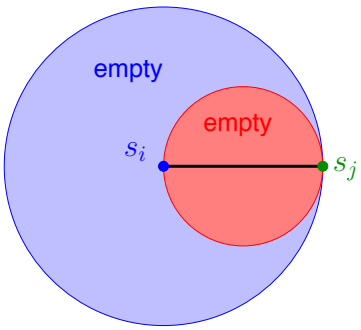
A *triangulation* of a set of sites is a way of partitioning the convex hull of the set of sites into triangles, where the vertices of the triangles consist of sites.

The *Delaunay triangulation* of a set of sites is a specific triangulation that is the "dual" of the Voronoi diagram. Two sites s_i and s_j are connected by an edge in the Delaunay triangulation if and only if R_i and R_j share a boundary in the Voronoi diagram. The following diagram superimposes the Delaunay triangulation on the Voronoi diagram above.



The Delaunay triangulation has a number of nice properties. Here are some of them.

* The closest pair is an edge of the Delaunay triangulation. (The following figure, in conjunction with the lemma below proves this.)



* The minimum spanning tree of the sites (using the Euclidean distance as the length of an edge) is a subset of the edges of the Delaunay triangulation.

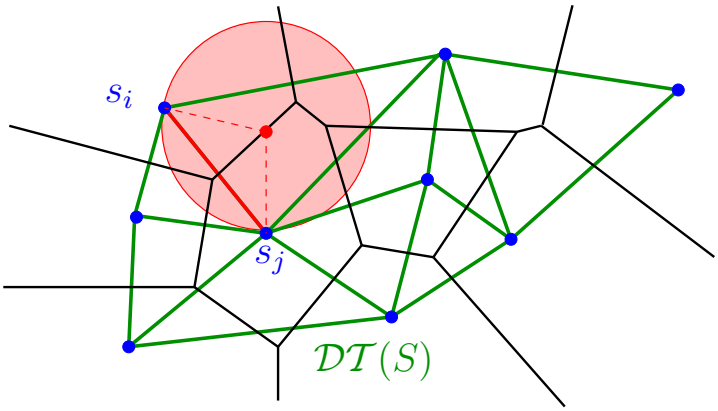
* The "smallest angle" in a triangulation is the minimum angle among all of the triangles in it. The Delaunay triangulation, among all triangulations, has the maximum smallest angle.

* You can use the Delaunay triangulation to compute the Voronoi diagram.

Lemma: The edges of the convex hull of the sites are edges in the Delaunay triangulation.

Proof: If two sites are neighbors on the convex hull, then if we follow the perpendicular bisector between the points sufficiently far away, eventually we must reach a point which is closest to the two neighbors than any other site. This shows that the two Voronoi regions have this boundary incommon. QED.

Lemma: (s_i, s_j) is in the Delaunay triangulation iff there exists a circle through s_i and s_j containing no other site inside of it.



Proof: $\Rightarrow$: (s_i, s_j) being in the Delaunay triangulation means that R_i and R_j have a boundary in common. Take a circle centered on a point on that boundary. This circle cannot contain another site inside of it (this would violate the fact that this point is closer to s_i and s_j than any other site.)

$\Leftarrow$: There exists a circle through s_i and s_j containing no other site. Let p be the center of this circle. Point p must be on the boundary between R_i and R_j. If p is not a vertex of the Voronoi diagram then it must be along an edge of the diagram, and we're done. If it is a vertex, then one of the three lines eminating from that vertex is the boundary between R_i and R_j. QED.

Lemma: Assuming no four sites are co-circular, then the Deleunay triangulation is unique.

Proof: Every vertex of the Voronoi diagram has degree 3. These correspond to the triangles of the Delaunay triangulation. QED.

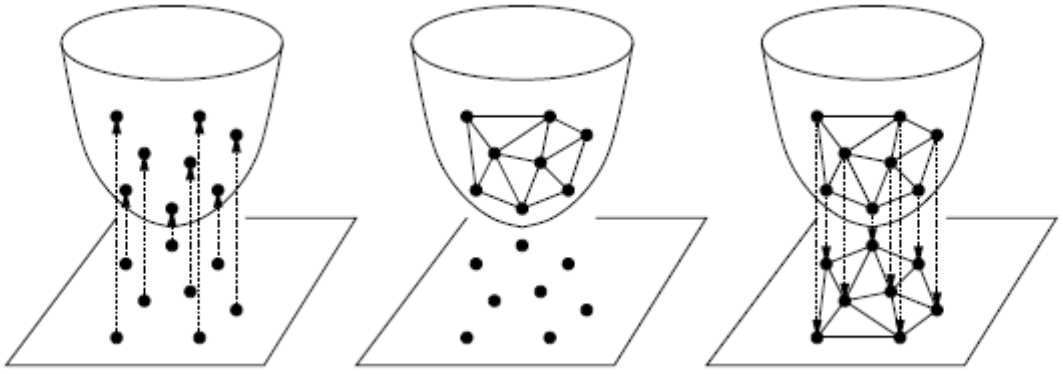
Note: Furthermore if there exists four or more co-circular sites, the Delaunay triangulation is not unique. It is unique if you allow the Delaunay "triangulation"" to have non-triangulated polygons (all of whose vertices are co-circular.)

Claim: Given a Delaunay triangulation for a set of sites we can compute the Voronoi diagram in O(n) time. And conversely.

3 Computing the Delaunay Triangulation

3D-Convex Hull Method: Take each site $s_i = (x_i, y_i)$ and replace it by a point in three dimensions $(x_i, y_i, x_i^2 + y_i^2)$. This is a point on a parabaloid of revolution.

Now take the convex hull of this set of points in 3D. The triangles of this convex hull (visible from the x, y plane), when projected back down to the plane, form the the Delaunay triangulation. No Proof. Note that the 3D convex hull can be computed in $O(n \log n)$ time. The following figure illustrates this process.



Today I will present a simple algorithm for computing the Delaunay triangulation in $O(n^2)$ time. Again we assume that no three sites are co-circular. (Although this assumption can easily be removed if we allow the algorithm to output co-circular polygons, instead of only triangles.)

Simple Delaunay Algorithm:

(1) Find an edge that must be in the Delaunay triangulation. (The closest pair has his property, as well as any edge of the convex hull of the sites.)

We're going to maintain a queue of directed Delaunay edges that need to have the triangle to "their right" be explored. Put the starting edge just found (both directions of it) into this queue to start with, and also add them to the Delaunay triangulation we're building.

(2) Process the queue in the following way until the queue is empty.

Let (i, j) be an edge in the queue. Consider all the other sites k such that s_k is to the right of ray $s_i \to s_j$. If there is no such k, then (i, j) is a convex hull edge. Delete it from the queue.

Otherwise, find the s_k to the right of $s_i \to s_j$ so that the circle (s_i, s_j, s_k) does not contain any other sites. We do this by simply trying them all, and keeping the best one found so far. If a new one is inside the circle of the best previously found one, this one becomes our new best. (This requires the incircle test described in an earlier lecture.)

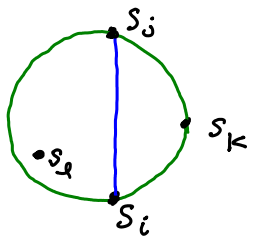
Now, having found our best site s_k, we add edges (i, k), (k, j) to the queue, and to the Delaunay triangulation we're building. We also remove (i, j), (j, k), and (k, i) from the queue.

A couple of comments:

* Each edge generated is a Delaunay edge. The first one (the closest pair) clearly is – just draw the circle with this edge as its diameter. It must contain no other sites.

  Every additional edge that we add comes with an associated empty circle that certifices it as a Deleunay edge.

* The circle through the points (s_i, s_j, s_k) found in the search above must be an empty circle. We know this because we have inductively certified that (s_i, s_j) is a Delaunay edge. Specifically the following situation cannot arise:



  There cannot be a site (such as s_l shown above) on the left side of the ray $s_i \rightarrow s_j$, and inside the circle. The existance of such a site proves that there is NO empty circle through sites s_i and s_j. And we know that is impossible by virtue of our proof that (s_i, s_j) is a Delaunay edge.

* When the algorithm stops, the entire triangulation has been generated. This follows from the fact that the triangulation is connected.

* The running time is $O(n^2)$ because for each Delaunay edge generated we may have to do a scan of all the other sites.

4 An Additional Links

A lecture by Ken Clarkson:
`http://cm.bell-labs.com/who/clarkson/cis677/lecture/5/`

This applet lets you insert, delete, and move sites around while viewing the Delaunay triangulation and/or the Voronoi diagram:
`http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/Voronoi/Incremental2/incremental2.htm`

Here's my Ocaml implementation of the $O(n^2)$ algorithm described above:
`http://codeforces.com/contest/275/submission/3201063`