We say that two points $\vec{u}, \vec{v} \in Y$ are in the same *connected component* of Y if there is a path in R^N from $\vec{u}$ to $\vec{v}$ such that all the points along the path are in the set Y. (There are two connected components in the figure.) We will make use of the following fundamental result on algebraic decision trees, due to Ben-Or. Intuitively, it states that if your set has M connected components, then there must be at least M leaves in any decision tree for the set, and the tree must have height at least the logarithm of the number of leaves.

Theorem: Let $Y \in R^N$ be any set and let T be any d-th order algebraic decision tree that determines membership in W. If W has M disjoint connected components, then T must have height at least $\Omega((\log M) - N)$.

We will begin our proof with a simpler problem.

Multiset Size Verification Problem (MSV): Given a multiset of n real numbers and an integer k, confirm that the multiset has exactly k distinct elements.

Lemma: The MSV problem requires $\Omega(n \log k)$ steps in the worst case in the d-th order algebraic decision tree

Proof: In terms of points in R^n, the set of points for which the answer is "yes" is

$$Y = \{(z_1, z_2, \ldots, z_n) \in R^n \mid |\{z_1, z_2, \ldots, z_n\}| = k\}.$$

It suffices to show that there are at least $k!k^{n-k}$ different connected components in this set, because by Ben-Or's result it would follow that the time to test membership in Y would be

$$\Omega(\log(k!k^{n-k}) - n) \;=\; \Omega(k \log k + (n-k)\log k - n) \;=\; \Omega(n \log k).$$

Consider the all the tuples $(z_1, \ldots, z_n)$ with $z_1, \ldots z_k$ set to the distinct integers from 1 to k, and $z_{k+1} \ldots z_n$ each set to an arbitrary integer in the same range. Clearly there are $k!$ ways to select the first k elements and k^{n-k} ways to select the remaining elements. Each such tuple has exactly k distinct items, but it is not hard to see that if we attempt to continuously modify one of these tuples to equal another one, we must change the number of distinct elements, implying that each of these tuples is in a different connected component of Y.

To finish the lower bound proof, we argue that any instance of MSV can be reduced to the convex hull size verification problem (CHSV). Thus any lower bound for MSV problem applies to CHSV as well.

Theorem: The CHSV problem requires $\Omega(n \log h)$ time to solve.

Proof: Let $Z = (z_1, \ldots, z_n)$ and k be an instance of the MSV problem. We create a point set $\{p_1, \ldots, p_n\}$ in the plane where $p_i = (z_i, z_i^2)$, and set $h = k$. (Observe that the points lie on a parabola, so that all the points are on the convex hull.) Now, if the multiset Z has exactly k distinct elements, then there are exactly $h = k$ points in the point set (since the others are all duplicates of these) and so there are exactly h points on the hull. Conversely, if there are h points on the convex hull, then there were exactly $h = k$ distinct numbers in the multiset to begin with in Z.

Thus, we cannot solve CHSV any faster than $\Omega(n \log h)$ time, for otherwise we could solve MSV in the same time.

The proof is rather unsatisfying, because it relies on the fact that there are many duplicate points. You might wonder, does the lower bound still hold if there are no duplicates? Kirkpatric and Seidel actually prove a stronger (but harder) result that the $\Omega(n \log h)$ lower bound holds even you assume that the points are distinct.

Lecture 27: Voronoi Diagrams

Reading: O'Rourke, Chapt 5, Mulmuley, Sect. 2.8.4.

Planar Voronoi Diagrams: Recall that, given n points $P = \{p_1, p_2, \ldots, p_n\}$ in the plane, the Voronoi polygon of a point p_i, $V(p_i)$, is defined to be the set of all points q in the plane for which p_i is among the closest points to q in P. That is,

$$V(p_i) = \{q \ : \ |p_i - q| \leq |p_j - q|, \forall j \neq i\}.$$

The union of the boundaries of the Voronoi polygons is called the *Voronoi diagram* of P, denoted $VD(P)$. The dual of the Voronoi diagram is a triangulation of the point set, called the *Delaunay triangulation*. Recall from our discussion of quad-edge data structure, that given a good representation of any planar graph, the dual is easy to construct. Hence, it suffices to show how to compute either one of these structures, from which the other can be derived easily in $O(n)$ time.

There are four fairly well-known algorithms for computing Voronoi diagrams and Delaunay triangulations in the plane. They are

Divide-and-Conquer: (For both VD and DT.) The first $O(n \log n)$ algorithm for this problem. Not widely used because it is somewhat hard to implement. Can be generalized to higher dimensions with some difficulty. Can be generalized to computing Vornoi diagrams of line segments with some difficulty.

Randomized Incremental: (For DT and VD.) The simplest. $O(n \log n)$ time with high probability. Can be generalized to higher dimensions as with the randomized algorithm for convex hulls. Can be generalized to computing Voronoi diagrams of line segments fairly easily.

Fortune's Plane Sweep: (For VD.) A very clever and fairly simple algorithm. It computes a "deformed" Voronoi diagram by plane sweep in $O(n \log n)$ time, from which the true diagram can be extracted easily. Can be generalized to computing Voronoi diagrams of line segments fairly easily.

Reduction to convex hulls: (For DT.) Computing a Delaunay triangulation of n points in dimension d can be reduced to computing a convex hull of n points in dimension $d + 1$. Use your favorite convex hull algorithm. Unclear how to generalize to compute Voronoi diagrams of line segments.

We will cover all of these approaches, except Fortune's algorithm. O'Rourke does not give detailed explanations of any of these algorithms, but he does discuss the idea behind Fortune's algorithm. Today we will discuss the divide-and-conquer algorithm. This algorithm is presented in Mulmuley, Section 2.8.4.

Divide-and-conquer algorithm: The divide-and-conquer approach works like most standard geometric divide-and-conquer algorithms. We split the points according to x-coordinates into 2 roughly equal sized groups (e.g. by presorting the points by x-coordinate and selecting medians). We compute the Voronoi diagram of the left side, and the Voronoi diagram of the right side. Note that since each diagram alone covers the entire plane, these two diagrams overlap. We then merge the resulting diagrams into a single diagram.

The merging step is where all the work is done. Observe that every point in the the plane lies within two Voronoi polygons, one in $VD(L)$ and one in $VD(R)$. We need to resolve this overlap, by separating overlapping polygons. Let $V(l_0)$ be the Voronoi polygon for a point from the left side, and let $V(r_0)$ be the Voronoi polygon for a point on the right side, and suppose these polygons overlap one another. Observe that if we insert the bisector between l_0 and r_0, and through away the portions of the polygons that lie on the "wrong" side of the bisector, we resolve the overlap. If we do this for every pair of overlapping Voronoi polygons, we get the final Voronoi diagram. This is illustrated in the figure below.

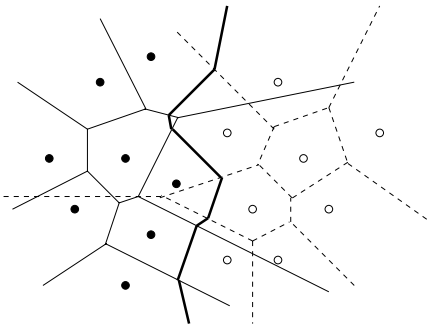
The union of these bisectors that separate the left Voronoi diagram from the right Voronoi diagram is called the *contour*. A point is on the contour if and only if it is equidistant from 2 points in S, one in L and one in R.

(0) Presort the points by x-coordinate (this is done once).

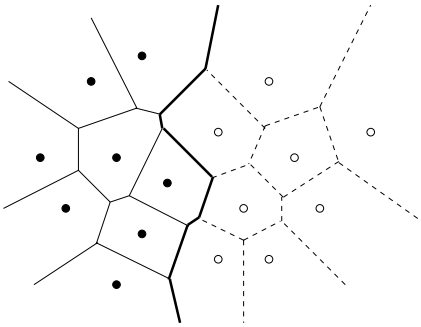
(1) Split the point set S by a vertical line into two subsets L and R of roughly equal size.

(2) Compute $VD(L)$ and $VD(R)$ recursively. (These diagrams overlap one another.)

(3) Merge the two diagrams into a single diagram, by computing the *contour* and discarding the portion of the $VD(L)$ that is to the right of the contour, and the portion of $VD(R)$ that is to the left of the contour.



Left/Right Diagrams and Contour



Final Voronoi Diagram

Figure 96: Merging Voronoi diagrams.

Assuming we can implement step (3) in $O(n)$ time (where n is the size of the remaining point set) then the running time will be defined by the familiar recurrence

$$T(n) = 2T(n/2) + n,$$

which we know solves to $O(n \log n)$.

Computing the contour: What makes the divide-and-conquer algorithm somewhat tricky is the task of computing the contour. Before giving an algorithm to compute the contour, let us make some observations about its geomtetric structure. Let us make the usual simplifying assumptions that no 4 points are cocircular.

Lemma: The contour consists of a single polygonal curve (whose first and last edges are semiinfinite) which is monotone with respect to the y-axis.

Proof: A detailed proof is a real hassle. Here are the main ideas, though. The contour separates the plane into two regions, those points whose nearest neighbor lies in L from those points whose nearest neighbor lies in R. Because the contour locally consists of points that are equidistant from 2 points, it is formed from pieces that are perpendicular bisectors, with one point from L and the other point from R. Thus, it is a piecewise polygonal curve. Because no 4 points are cocircular, it follows that all vertices in the Voronoi diagram can have degree at most 3. However, because the contour separates the plane into only 2 types of regions, it can contain only vertices of degree 2. Thus it can consist only of the disjoint union of closed curves (actually this never happens, as we will see) and unbounded curves. Observe that if we orient the contour counterclockwise with respect to each point in R (clockwise with respect to each point in L), then each segment must be directed in the $-y$ directions, because L and R are separated by a vertical line. Thus, the contour contains no horizontal cusps. This implies that the contour cannot contain any closed curves, and hence contains only vertically monotone unbounded curves. Also, this orientability also implies that there is only one such curve.

Lemma: The topmost (bottommost) edge of the contour is the perpendicular bisector for the two points forming the upper (lower) tangent of the left hull and the right hull.

Proof: This follows from the fact that the vertices of the hull correspond to unbounded Voronoi polygons, and hence upper and lower tangents correspond to unbounded edges of the contour.

These last two theorem suggest the general approach. We start by computing the upper tangent, which we know can be done in linear time (once we know the left and right hulls, or by prune and search). Then, we start tracing the contour from top to bottom. When we are in Voronoi polygons $V(l_0)$ and $V(r_0)$ we trace the bisector between l_0 and r_0 in the negative y-direction until its first contact with the boundaries of one of these polygons. Suppose that we hit the boundary of $V(l_0)$ first. Assuming that we use a good data structure for the Voronoi diagram (e.g. quad-edge data structure) we can determine the point l_1 lying on the other side of this edge in the left Voronoi diagram. We continue following the contour by tracing the bisector of l_1 and r_0.

However, in order to insure efficiency, we must be careful in how we determine where the bisector hits the edge of the polygon. Consider the figure shown below. We start tracing the contour between l_0 and r_0. By walking along the boundary of $V(l_0)$ we can determine the edge that the contour would hit first. This can be done in time proportional to the number of edges in $V(l_0)$ (which can be as large as $O(n)$). However, we discover that before the contour hits the boundary of $V(l_0)$ it hits the boundary of $V(r_0)$. We find the new point r_1 and now trace the bisector between l_0 and r_1. Again we can compute the intersection with the boundary of $V(l_0)$ in time proportional to its size. However the contour hits the boundary of $V(r_1)$ first, and so we go on to r_2. As can be seen, if we are not smart, we can rescan the boundary of $V(l_0)$ over and over again, until the contour finally hits the boundary. If we do this $O(n)$ times, and the boundary of $V(l_0)$ is $O(n)$, then we are stuck with $O(n^2)$ time to trace the contour.

We have to avoid this repeated rescanning. However, there is a way to scan the boundary of each Voronoi polygon at most once. Observe that as we walk along the contour, each time we stay in the same polygon $V(l_0)$, we are adding another edge onto its Voronoi polygon. Because the Voronoi polygon is convex, we know

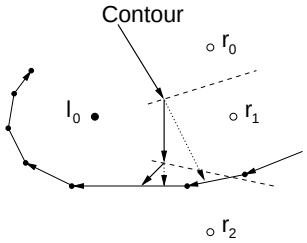


Figure 97: Tracing the contour.

that the edges we are creating turn consistently in the same direction (clockwise for points on the left, and counterclockwise for points on the right). To test for intersections between the contour and the current Voronoi polygon, we trace the boundary of the polygon clockwise for polygons on the left side, and counterclockwise for polygons on the right side. Whenever the contour changes direction, we continue the scan from the point that we left off. In this way, we know that we will never need to rescan the same edge of any Voronoi polygon more than once.

Lecture 28: Delaunay Triangulations and Convex Hulls

Reading: O'Rourke 5.7 and 5.8.

Delaunay Triangulations and Convex Hulls: At first, Delaunay triangulations and convex hulls appear to be quite different structures, one is based on metric properties (distances) and the other on affine properties (collinearity, coplanarity). Today we show that it is possible to convert the problem of computing a Delaunay triangulation in dimension d to that of computing a convex hull in dimension $d + 1$. Thus, there is a remarkable relationship between these two structures.

We will demonstrate the connection in dimension 2 (by computing a convex hull in dimension 3). Some of this may be hard to visualize, but see O'Rourke for illustrations. (You can also reason by analogy in one lower dimension of Delaunay triangulations in 1-d and convex hulls in 2-d, but the real complexities of the structures are not really apparent in this case.)

The connection between the two structures is the *paraboloid* $z = x^2 + y^2$. Observe that this equation defines a surface whose vertical cross sections (constant x or constant y) are parabolas, and whose horizontal cross sections (constant z) are circles. For each point in the plane, (x, y), the *vertical projection* of this point onto this paraboloid is $(x, y, x^2 + y^2)$ in 3-space. Given a set of points S in the plane, let S' denote the projection of every point in S onto this paraboloid. Consider the *lower convex hull* of S'. This is the portion of the convex hull of S' which is visible to a viewer standing at $z = -\infty$. We claim that if we take the lower convex hull of S', and project it back onto the plane, then we get the Delaunay triangulation of S. In particular, let $p, q, r \in S$, and let p', q', r' denote the projections of these points onto the paraboloid. Then $p'q'r'$ define a *face* of the lower convex hull of S' if and only if $\triangle pqr$ is a triangle of the Delaunay triangulation of S. The process is illustrated in the following figure.

The question is, why does this work? To see why, we need to establish the connection between the triangles of the Delaunay triangulation and the faces of the convex hull of transformed points. In particular, recall that

Delaunay condition: Three points $p, q, r \in S$ form a Delaunay triangle if and only if the circumcircle of these points contains no other point of S.

Convex hull condition: Three points $p', q', r' \in S'$ form a face of the convex hull of S' if and only if the plane passing through p', q', and r' has all the points of S' lying to one side.