CMSC 754: Spring 1996 Samir Khuller

Applications of Geometric Duality

We study two applications of geometric duality. The first one yields a fast algorithm for computing the minimum area triangle from among n points in the plane. The second one gives a preprocessing algorithm for answering "ray-shooting" queries quickly.

Min-Area Triangles (Chazelle, Guibas and Lee "The power of geometric duality" BIT (1985))

(Notes by Sudipto Guha)

To find the minimum area triangle, we first focus our attention on the problem of finding a min area triangle that has two fixed points P_i and P_j chosen as the "base" of the triangle. Our task reduces to finding for each pair of points, P_i and P_j, the point P_k whose normal distance to the line through P_i and P_j is the least. Say $\ell = P_iP_j$ has slope m and y intercept c. For all lines having slope m, through other points, the line through P_k has y intercept nearest to c. Call its intercept c'. (This follows from the fact that the normal distance between the lines is proportional to their y intercepts.)

In the dual, this line will be the first point vertically below (above) the point of intersection of the duals of P_i and P_j. This intersection point will have coordinates, (m, c), and the line closest to this, and vertically below (above), is the dual of P_k, passing through the point (m, c').

To compute point P_k, for every pair of points P_iP_j, we contruct the arrangement of the dual graph. While constructing the arrangement, we keep track of which line is immediately below which point. This can easily be done while constructing the arrangement itself. (Recall that in the proof of the zone theorem, we do a "walk" around each face that the new line is incident to. While doing this walk we can perform the required updates.) This gives us all the information we need. To compute the min area triangle, we minimize over all P_i and P_j, which can be done in $O(n^2)$ time, where n is the number of points.

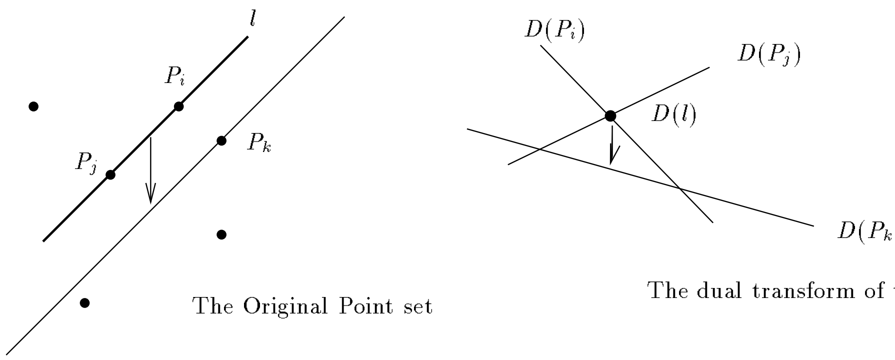

Figure 1: Finding minimum area triangles.

Ray-Shooting (Chazelle and Guibas, "Visibility and intersection problems" Disc and Comp Geom (1989))

(Notes by Samir Khuller)

Given a simple polygon P of n vertices, we wish to preprocess this polygon to answer ray-shooting queries of the following form: given a query point $q \in P$ and a direction $\vec{d}$, which is the *first* edge of the polygon that is hit when we shoot a ray from q? We will show that these queries can be answered in $O(\log^2 n)$ time. (Chazelle and Guibas provide a more complex $O(\log n)$ query time algorithm as well.)

The main idea is to decompose the polygon into a "hierarchy" of polygons. We use the "polygon-cutting theorem" of Chazelle's to do this. The polygon cutting theorem states that every simple polygon with n vertices ($n > 3$) contains a diagonal such that adding the diagonal, partitions the polygon into two polygons P_1 and P_2, with each P_i having at most $\frac{2}{3}n$ vertices.

We now construct a tree using this decomposition. The vertices of this tree correspond to polygons. The root corresponds to the polygon P. We apply the polygon cutting theorem to P, and the children of this vertex correspond to polygons P_1 and P_2 respectively. We now apply the polygon cutting theorem recursively to each node, until the corresponding polygon is a triangle. Observe that the height of this tree is $O(\log n)$, since we are reducing the size of the polygons by a constant factor at each step. For any polygon P, let $P(e)$ denote the diagonal that is used to decompose the polygon. We assume that each node in the tree has the diagonal information available.

The polygon cutting theorem can be proved using the fact that the dual of the triangulation is a tree, and we can prove that each such tree (vertices with degree at most 3) always contains an edge that partitions the tree into smaller trees with at most $\frac{2}{3}n$ vertices.

We solve the query $hit(P, q, \vec{d})$ as follows. We assume that $q \in P$. If $q \in P_1$ then the only way the ray can exit the polygon from δP_2 is if it crosses edge $P(e)$ without crossing any polygon boundary of P_1 (see Fig. 2).

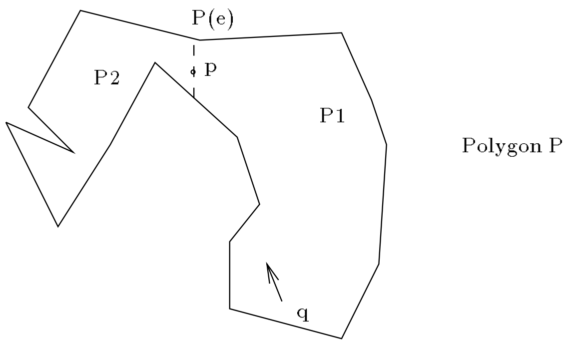

Figure 2: Ray-shooting from point q in direction $\vec{d}$.

Step 1. (Assume that $q \in P_1$.) If the ray from point q with direction $\vec{d}$ does not intersect edge $P(e)$, then recursively solve $hit(P_1, q, \vec{d})$.
Step 2. If the ray appears to pierce $P(e)$ at point p, then we have to check if the ray is able to reach $P(e)$ from q. If so, we recursively solve $hit(P_2, p, \vec{d})$ else we recursively solve $hit(P_1, q, \vec{d})$.

We still need to check if the ray from q reaches $P(e)$. We do this by shooting a ray from p towards q. If the ray reaches q, then the ray from q must reach $P(e)$. Compute $o = reach(P_1, p, -\vec{d})$ – if the segment $\overline{po}$ contains q then the ray from q in direction d reaches p, otherwise it crosses the boundary at an earlier point. This reduces the problem to another ray shooting query. Why is this query any easier to solve?

We will show that such queries are simpler, since they are being shot from a particular edge on the boundary of P_1, namely the edge $P(e)$. Chazelle and Guibas make use of a duality transformation that establishes a correspondence between points (on a two sided plane) and rays.

Any ray r_i shot from $P(e)$ becomes a point p_i in dual space. We decompose the dual space into "regions" with the property that if two rays r_1 and r_2 from $P(e)$ hit the same edge e' (perhaps at different places) then the points p_1 and p_2 are in the same "region". Moreover, they prove that this region is *convex*. The main idea is that any query ray r' shot from edge $P(e)$ can be transformed into a point p', and by doing a *point-location* query in $O(\log n)$ time, we can determine the edge that ray r' hits (based on the region it falls into). Once we know the edge, it is easy to compute the intersection point. This takes $O(\log n)$ time.

The entire algorithm takes $O(\log^2 n)$ time, since we start at the root of the polygon tree, and proceed down the tree doing a point location query at each level. The tree has $O(\log n)$ levels, and each point location takes $O(\log n)$ time. Further data structure tricks are used to reduce the query time to $O(\log n)$.