CMSC 754: Spring 1996 Samir Khuller

Randomized Incremental Convex Hull Algorithm

Incremental convex hull algorithms are usually relatively simple to implement, and generalize well to higher dimensions. (Gift wrapping is the other algorithm that is often used in higher dimensions, but it's worst case performance is not good even in the plane.) This algorithm operates by inserting points one at a time and incrementally updating the hull. We describe a simple 2-dimensional *randomized* version of this algorithm, and analyze its expected running time. As with other randomized algorithms, the expected running time depends on the random order of insertion, but is independent of the point set.

The Algorithm. The randomized algorithm operates as follows. Permute the points randomly. Let $P = \langle p_0, p_1, \ldots, p_{n-1}\rangle$ be this sequence. Let $H_2 = \text{conv}(p_0, p_1, p_2)$ (a triangle). For i running from 3 to $n-1$, insert p_i into the existing hull H_{i-1}. How this is done is explained below.

We assume that each point that has yet to be inserted has a pointer to one edge of the current convex hull such that this point lies "outside" this edge (in the half-plane that does not contain the convex hull). If the point lies inside the hull, then it is marked as "inside", and no further processing is needed. We make this association by considering a point O that lies in the interior of the convex hull (e.g. the centroid of the initial triangle). Imagine a line segment drawn from each point p_i to O. This segment must hit some edge of the hull. We associate p_i with this edge. In addition, each edge of the hull knows which points are associated with it. For edge e, let $L(e)$ denote the set of points associated with e in this way, and let $l(e)$ denote the size of this list.

For example, in the figure part (a) shown below, each point is shown connected by a line segment to its associated edge. $L(e) = \{p_6, p_9\}$.

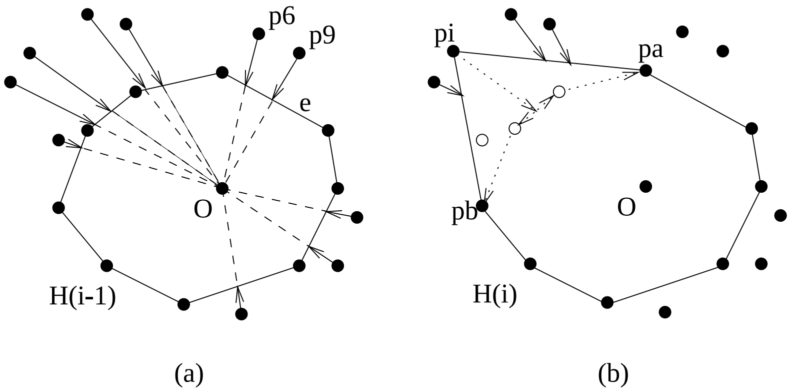

Figure 1: Incremental convex hull.

When a new point, p_i, added to the current hull, H_{i-1}, to form the new hull H_i, we follow p_i's pointer to the edge that p_i lies outside of. See Figure 1(b). Notice that this edge can no longer be on the convex hull. We walk both clockwise and counterclockwise around H_{i-1} starting at this edge until finding the points p_a and p_b of tangency between p_i and H_{i-1}. We connect p_i to these points, and discard the edges that were traversed from the hull. The resulting polygon is H_i. There is one more thing we have to do. For each of old hull edge e that was discarded, we access each of the points in $L(e)$ and determine whether (1) it now lies outside of (p_a, p_i), or (2) it now lies outside of (p_i, p_b) or (3) it does not lie outside of either. In the first case the point is transferred to $L(p_a, p_i)$, in the second to $L(p_i, p_b)$ and in the third it is marked as "inside", because it lies within the current hull.

Analysis. It is not hard to see that the algorithm is correct. What is the expected running time of this algorithm? There are 3 things that the algorithm does at each stage.

Destruction: It destroys some number of old edges.

Creation: It creates 2 new edges.

Reclassification: It reclassifies some number of points according to the edge that they now lie outside of.

Observe that each edge that is destroyed by the algorithm must have been created at some time in the past, thus the total time spent destroying edges cannot be asymptotically different from the total time spent creating edges. Since the algorithm creates 2 edges for each of $n-3$ stages, the total time spent creating, and hence destroying, edges is $O(n)$. (Formally, one can argue this by an accounting scheme. Each time an edge is created on the hull boundary, we store 1 $ with it. This money can be used to pay for the work that we do to delete the edge when we search for a supporting line from the newly inserted point.)

The only thing left to analyze is the expected time of reclassification. We claim that the total reclassification time will be $O(n \log n)$ in the expected case. (In other words, on average, each point is reclassified $O(\log n)$ times.) This is quite easy to prove by a backwards analysis. Reclassification can result in a point being labeled as "inside". However, observe that this only happens once for each point in the interior of the final hull, so the total cost for these types of reclassification cannot exceed n. Hence we only need to worry about counting the expected number of reclassifications of points that remain outside the current hull. Consider the *last* point to be added into H_i. With equal probability, $1/(i+1)$, each point in $\{p_0, p_1, \ldots, p_i\}$ was the last to be added. Let p be this point. If p is interior to the current hull, then no work was needed in reclassification. Otherwise, p will be on the current hull, H_i. Let e and e' be the two edges incident to p on H_i. Observe that the uninserted points that would have to be reclassified, are just the points that are currently in $L(e)$ and $L(e')$. Thus, the cost for reclassification would be $l(e) + l(e')$ if p were last to be inserted. To determine the expected cost, we sum over these values over all $i+1$ points, and multiply by the probability $1/(i+1)$ that this was the last point added. Observe that in this sum, for each edge e, $l(e)$ is added twice (once for the point on the left and one for the point on the right). Thus the expected cost of reclassification in the construction of H_i is

$$\frac{1}{i+1} \sum_{e \in H_i} 2l(e) = \frac{2}{i+1} \sum_{e \in H_i} l(e).$$

Since the sum of $l(e)$ counts each of the remaining $n-i$ points at most twice (exactly twice if it lies outside the current hull), this sum evaluates to $2(n-1-i)$ and thus the expected cost of reclassification is

$$\frac{2(n-1-i)}{i+1} \leq \frac{2n}{i+1}.$$

Now, to get the total time spent in reclassification, we sum over all stages, giving

$$T(n) = \sum_{i=3}^{n-1} \frac{2n}{i+1} = 2n \sum_{i=3}^{n-1} \frac{1}{i+1} \approx 2n \ln n \in O(n \log n).$$

This cost dominates the running time of the algorithm. This completes the analysis. The same algorithm can be generalized to 3-space without too much difficulty (although note that each point will be attached not by 2-tangents, but by a *polygonal cone* whose apex is the new point. The analysis is more complicated, because the number of faces in this cone may not be a constant.