CONVEX HULL ALGORITHMS

September 13, 2010

QuickHull

This is a divide-and-conquer algorithm, similar to quicksort, which divides the problem into two sub-problems and discards some of the points in the given set as interior points, concentrating on remaining points.

If we implement this algorithm, choosing at each stage the point farthest from line joining left and rightmost points, then the recurrence we get is :

T(n) = T(l) + T(r) + O(n) , n being the total no of points, r the no of points that lie to the right of perpendicular from point onto the line and l the points to the left of it.

In such a case if the points are arranged in a much skewed fashion, i.e.
if always $l >> r$
then $T(n) = \theta(n^2)$.

So we'll use randomization.

But first we see a deterministic algorithm which would be randomized later.

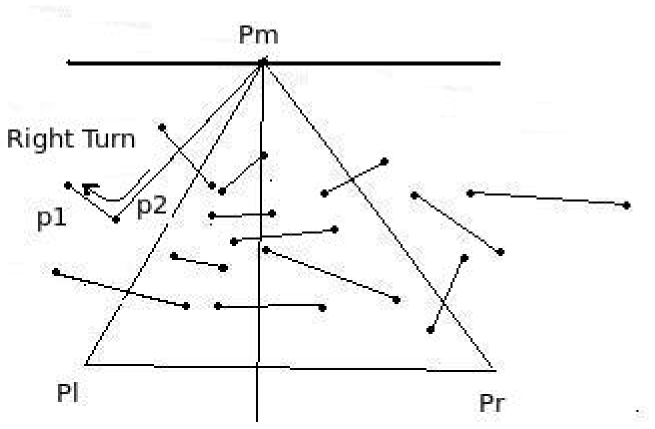
The steps of algorithm are as follows-

1. Pair the points arbitrarily. Denote the lines through these points by l_1, l_2, ... ,$l_{n/2}$.
2. Among these n/2 lines, find the line with median slope, say l_m.
3. Now find the extreme point in orthogonal direction to l_m, say p_m and draw a vertical line through it to partition the point set and divide the problem.
4. Prune some points using the following test :
    - In left subproblem, consider a pair p_1p_2 (p_2 being closer to the vertical line). If $p_mp_2p_1$ is a right turn, then we discard p_2.
    - Likewise in the right subproblem, if $p_mp_2p_1$ is a left turn, then we discard p_2.

5. Call this algorithm recursively on the left and right subset of remaining points(if there are some remaining points).

Observation :

For any pair $p_1p_2$that has slope less than (or equal to) l_m and is to the left of the vertical line from p_m, the triplet $p_mp_2p_1$will be a right turn and hence p_2 can be discarded on the basis of the "turn test".



Claim 1 :

When we call this algorithm recursively on the left and right sides of vertical line, the maximum no. of points in a sub-problem can be 3n/4.

Proof :

In the worst case, it may happen that all points lie to one side of the vertical line which, let us assume is the left side. So we have n/2 pairs of points on the left side. Now since half of the pairs have slopes less than the median slope, so n/4 such pairs which make a right turn with p_m. Hence n/4 points are discarded, so maximum no of points that remain are 3n/4.

Hence the recurrence for running time of this algorithm in terms of input and output points -

$T(n,h) = T(n_l,h_l) + T(n_r,h_r) + O(n)$

where n is total number of input points,

h is number of output points,

n_l is the number of points remaining on the left side,

n_r is the number of points remaining on the right side,

h_l is the number of output points from the left sub-problem and
h_r is the number of output points from the right sub-problem.

Also,

1. T(n,1) = c n (c is some constant)
2. n_l , n_r <= 3n/4
3. h_r = h - h_l - 1
4. n_l + n_r <= n

Since the above recurrence is not simple, we would guess and verify its solution rather than solving it.

Claim 2 :

The solution of recurrence is T(n,h) = O(n log(h)) (for n >1)

Proof :

We'll prove it by induction. We will plug in the solution and verify that it holds.
$T(n_l,h_l) = k\ n_l \log(h_l)$
$T(n_r,h_r) = k\ n_r \log(h_r) = k\ n_r \log(h-h_l)$
R.H.S. of the recurrence = $k\ (\ n_l \log(h_l) + n_r \log(h-h_l)\ ) + cn$
Since $n_l <= 3n/4$,
R.H.S = $k\ n\ (x \log(h_l) + (1-x) \log(h-h_l)\ ) + cn$ (where $1/2 < x < 3/4$)
Maximum value of R.H.S is obtained when h_l=x h
R.H.S <= $k\ n\ (\ x \log(x\ h) + (1-x) \log((1-x)\ h)\ ) + cn$
<= $k\ n \log(xh) + cn$
<= $k\ n\ (\log(h) - \log(4/3)) + cn$
Now this maximum value of R.H.S must always be less than T(n,h)
=> $k\ n\ (\log(h) - \log(4/3)) + cn <= k\ n \log(h)$
=> $k >= c/(\log(4/3))$
If the above condition holds, then the solution of recurrence = n log(h) is verified.
Thus, the Running Time of this modified Quickhull algorithm is O(n log(h)).

Comparison to Quicksort :

Here at each stage we are finding the median slope. However if we truly emulate quicksort, in quickhull we must not find the median slope but must choose any random slope. Since in quicksort, if we choose a splitter at random, the running time is expected O(n logn), similar result follows for randomized quickhull i.e. the running time is expected O(n logn).

MergeHull

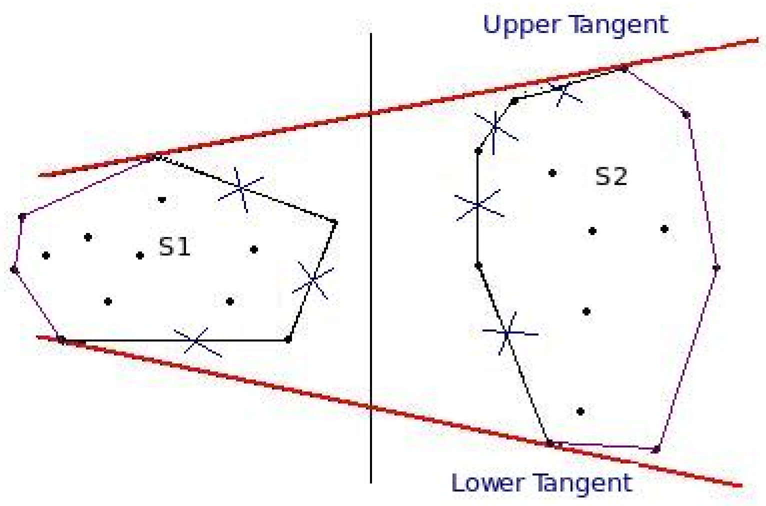
This algorithm works similar to merge-sort in the following way :

1. Partition the given set of points into two sets, say S1 and S2.
2. Recursively compute the convex hulls of the two sets, say CH(S1) and CH(S2).
3. Merge the two small convex hulls, CH(S1) and CH(S2) to give the final convex hull CH(S).

Now first let us assume that the partitioning of points is done in such a way that there are exactly n/2 points in both sets and are linearly separable. For this we would need to compute the median on the basis of x-coordinates. This operation is O(n).

How do we do merging ?

1. First, find the lines that are upper tangent, and lower tangent to the two hulls (the two red lines).
2. Then remove the points that are cut off (lying between the upper and lower tangents).



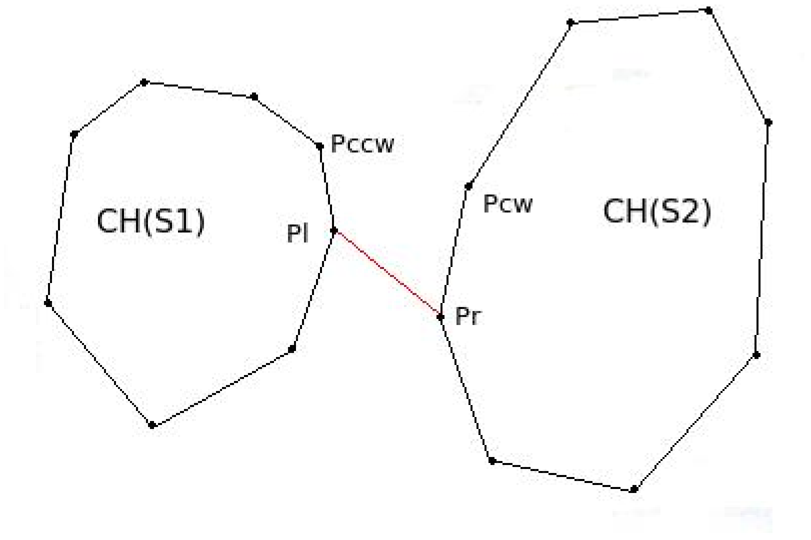
Finding Tangent Lines :

First we find the upper tangent in the following way -

1. Start with the rightmost point of the left hull, and the leftmost point of the right hull and join them assuming this line is the upper tangent.
2. While the line is not upper tangent to both left and right halves do :
    - If the line is not upper tangent to the left, move to the next counter-clockwise point on the left convex hull and check.
    - Else if the line is not upper tangent to the right, move to the next clockwise point on the right convex hull and check.

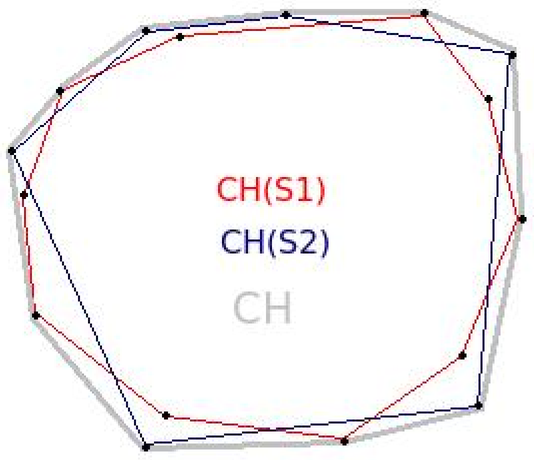
Now how do we check whether the line is tangent to left and right hulls ?

- If the points p_{CCW}, p_l, p_r make a right turn then line is a tangent to left hull.
- If the points p_l, p_r, p_{CW} make a right turn then line is a tangent to right hull.



In the same way, find the lower tangent.
So the entire process of finding common tangents takes O(n) time.
So Mergehull works in T(n) = 2T(n/2) + O(n) + O(n)
T(n) = O(n log(n))

But what if the points are actually partitioned arbitrarily and the two convex hulls are not linearly separable.



Now we merge using Graham's Scan. Since we have the two convex hulls we have the points on boundary in order. So 1st we will find the upper hull.

- For this we take the ordered list of points lying on the upper half of CH1 and then the ordered list of points lying in the upper half of CH2 and merge the two lists such that final list is also ordered on x. This operation takes O(n).
- Then starting from the leftmost point we do Graham's Scan on the entire list and get the upper hull in O(n).
- Similarly we find the lower hull. So the entire merge operation took O(n).

This implies that T(n) = 2T(n/2) + O(n)
So T(n) is again O(n logn).

Insertion Hull

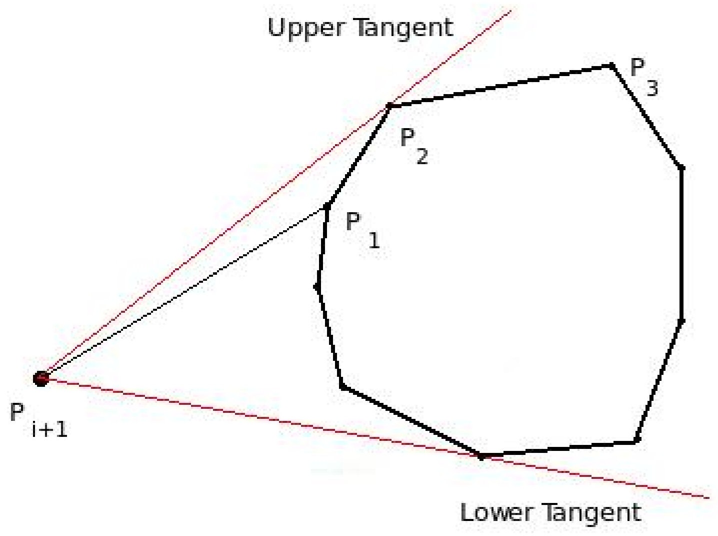
This works very similar to insertion sort.

- We start with 3 points that define a trivial convex hull, the triangle.
- Then we insert one point at a time, updating the convex hull, until we have exhausted all n points.

Suppose we are considering point p_{i+1}

- If the point falls within the interior of CH_i (the convex hull of 1st i points), then we ignore and move forward.
- Else we construct CH_{i+1} by finding the two tangents from point p_{i+1}.

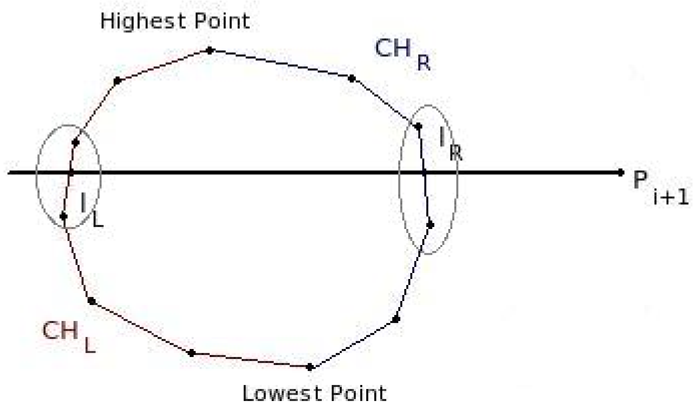
Construction of tangents from a point :



Clearly, the points $p_{i+1}p_1p_2$ make a left turn whereas the points $p_{i+1}p_2p_3$ make a right turn. So the upper tangent intersects the hull at a point where the turn switches from Left to Right. So we can do a binary search on the points and find this point in O(logn). Similarly the lower tangent occurs at a point where the turns switch from Right to Left.

But how do we check whether the point is in interior of the CH_i or not?

For finding if the point is included in the convex polygon CH_iwe will use a generalization of Ray Shooting. The idea is to draw a horizontal line through the point $p_{i+1}$and find which two sides of CH_iit intersects. If the point lies between the two sides, it is in the interior of CH_ielse it lies outside CH_i.



The steps of algorithm are :

- Find the points having the highest and lowest y-coordinates. Since the points are given to us in a sequence as they appear in CH_i, we can find these points by doing binary search. So time taken is O(logn).
- Then break the sequence of points into two parts. One sequence contains point from the highest to lowest and the other contains points from lowest to highest, call them $CH_R$and CH_L. Find the two points in both lists whose y-coordinates are just greater and just lesser than the y-coordinate of point. The line segments,$l_L$and l_R formed by these two points in both lists will be those line segments that are intersected by the horizontal line through p_{i+1}. Since the list contains points in a sorted order, finding the above mentioned points require a simple binary search. So again time required is O(logn).
- Now check whether p_{i+1}lies on the same side of both $l_L$and l_R. This can be done by simply plugging the point into the equation of these two line segments. If both give the same sign, then the point is on same sides and is hence exterior to CH_i else it lies in between the lines and is hence interior. This step is clearly an O(1) step.

Thus we can solve the problem of point inclusion in CH_iin O(logn).

Time Complexity of Insertion Hull :

We have seen that for a new point, checking its inclusion costs O(logn) and if it is exterior then finding two extreme tangents also costs O(logn).

So total running time, T_n = n * O(logn)

This implies $T_n = O(n \log n)$.