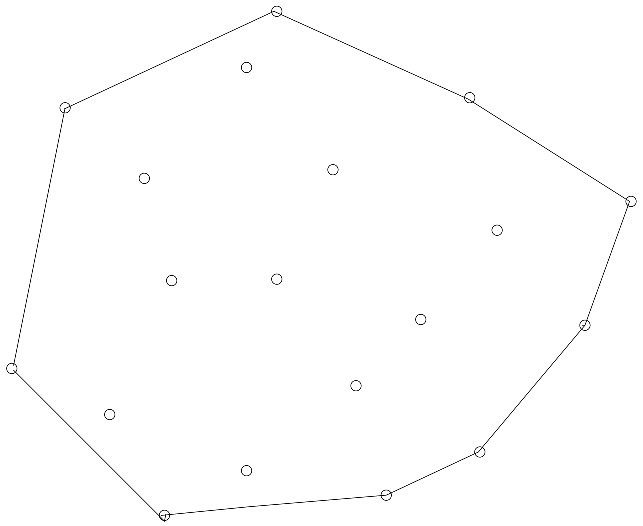
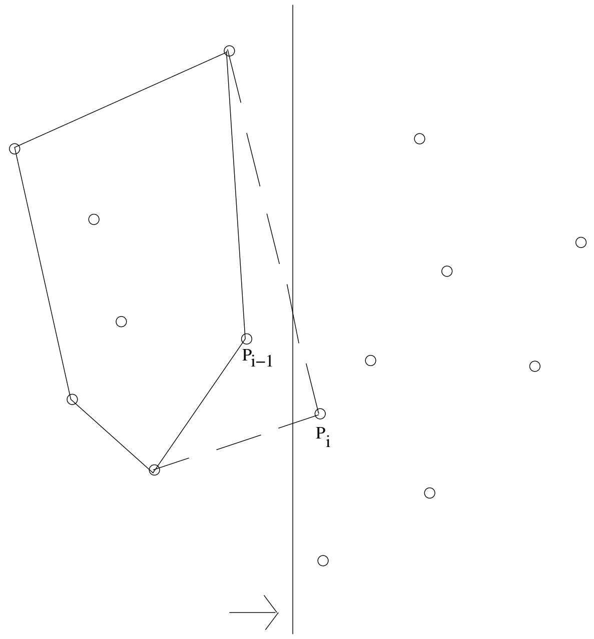


Introduction

Last class, we covered various solutions to the 2D convex hull problem: Given a list of points $\{p_1, p_2, \ldots, p_n\}$ in the plane, output a list of the hull vertices in counterclockwise order. We discussed the Jarvis March, Graham Scan (a.k.a. the three penny dance), Divide and Conquer, and Marriage Before Conquest. The fundamental building block for most solutions is the counterclockwise test: give points p, q, r are they in counterclockwise order or not?

Today, we continue looking at this problem, to examine additional techniques which will apply in a more general context. We cover, Sweep [Graham, Yao '83], Quickhull [Eddy '77], Prune and Search [Wegner '95] and Randomized Incremental [Clarckson, Shor '83].



Sweep Algorithm

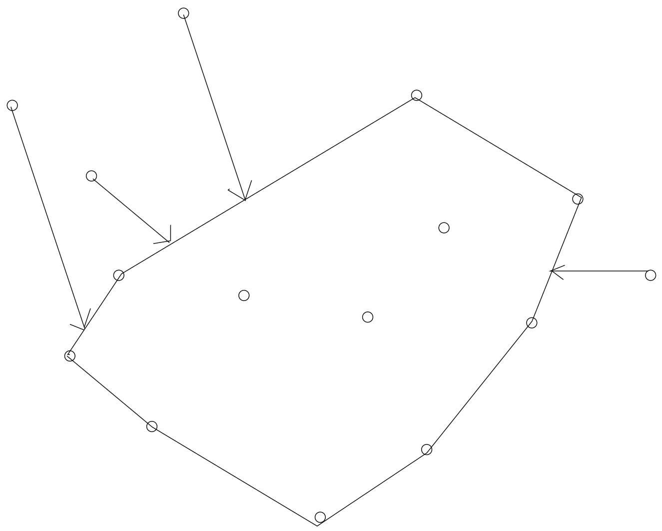
The **Sweep Algorithm** sorts the points by x-coordinate, then adds each new point to the current convex hull, as it sweeps from left to right. (This algorithm is essentially the same as the three penny dance). Even though the outer loop runs $O(n)$ times and the body of the loop can take $O(n)$ time, the entire loop requires only $O(n)$ time, since each point is added and discarded at most once. Thus, the running time is dominated by the $O(n \log n)$ time it takes to sort the points.

```
Sort points left to right
For i ← 4 to n
   connect p_i to p_{i-1}
   discard concave corners
```

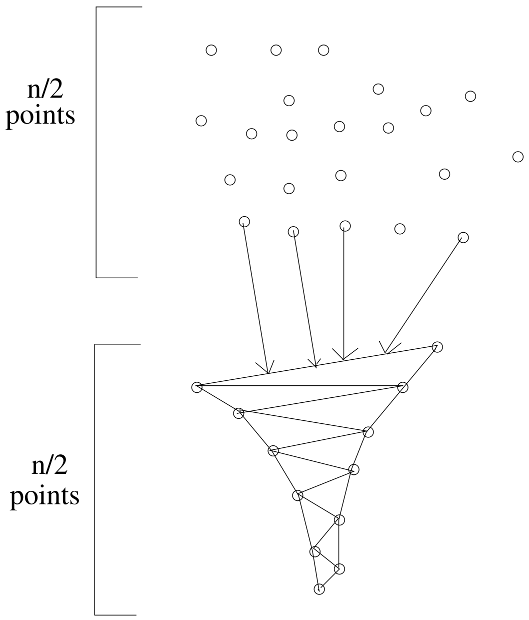
Randomized Incremental

In the Sweep Algorithm, it was the sorting phase which was most costly. The **Incremental Algorithm** draws on the idea that it is not necessary to sort the points before inserting them. The idea is that each point can "see" one edge of the hull. If the point lies outside the hull, when it is encountered, we attach edges from the point to the two endpoints of the edge of the hull which it sees, then discard concave edges as in Sweep.

As long as the edge which is seen by a point doesn't change too frequently, this has potential for good running time. Unfortunately, as is often the case with deterministic algorithms, if the order in which the points are inserted is known, an adversary can construct input that will run in $O(n^2)$ time rather than our goal of $O(n \log n)$.



Above is an example of Incremental in progress. Below is an example of input that will make Incremental run in $O(n^2)$ time, rather than the desired $O(n \log n)$. Each time a new point in the bottom half is added to the hull, all the points in the top half must update the edge that they see.



As is often the case when input can be constructed to give an algorithm a bad worst case running time, we turn to randomization. Whatever input we are given, we randomly permute the order of the points so that each ordering is equally likely. Then, though it is still possible to achieve the situation above (or one similar), it is very unlikely, and the algorithm achieves expected running time of $O(n \log n)$.

To analyze the running time of **Randomized Incremental**, we use the technique of Backwards Analysis (this is an important technique which is used frequently). By running the algorithm in reverse, we see that as we remove points from the hull, the edge that a point (not on the hull) sees, will change only if either of the edges endpoints is removed.

$$\begin{aligned}
E[time] &= O(E[\text{changes in visibility}]) \\
&= O(\sum_{i=1}^{n} \sum_{j=1}^{n} Pr[e_j(p_i) \neq e_{j-1}(p_i)]) \\
&= O(\sum_{i=1}^{n} \sum_{j=1}^{n} Pr[p_j \text{ is an endpoint of } e_j(p_i)]) \\
&\leq O(\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{2}{j}) \\
&= O(n \log n)
\end{aligned}$$

Quick Hull

The idea of **QuickHull** is that every point that it adds to the hull as it progresses is on the final hull.

```
QuickHull(P, l, r)
   Find point F ∈ P furthest from lr (overlined)
   Split P into L, M, R
   QuickHull(L, l, t)
   QuickHull(R, t, r)
```

This gives the recurrence $T(n, h) = O(n) + T(n_1, h_1) + T(n_2, h_2)$. However, since we have no bound on the size of n_1 and n_2 other than n, the recurrence may not be balanced and we can only guarantee $O(n^2)$ running time. Ironically, in practice, **QuickHull** outperforms all other algorithms.

In order to achieve a theoretical running time of $O(n \log n)$, the following modifications were made:

1. Pair up the points arbitarily.
2. Choose the pair p, q with median slope.
3. Find the point t furthest from $\overline{pq}$
4. Prune points: for each pair $\{a, b\}$ discard anything not on conv$\{l, r, t, a, b\}$

The key result of these modifications is that each of the resulting sets on which we recurse is at most 3/4 the size of the original set. This results in the recursion being balanced enough that we achieve a running time of $O(n \log n)$, as we show below.

Lemma 0.1. $|L| \leq \frac{3}{4}n$

Location of pair	Contribution to L
Neither point above $\overline{lt}$	0
One point above $\overline{lt}$	1
Both points above $\overline{lt}$	
slope $(\overline{ab})$ >median	≤ 2
slope $(\overline{ab})$ <median	≤ 1

Since the slope of $\overline{ab}$ is greater than the median only half of the time, at most half of the time are both a and b in L. This gives us a bound of $\frac{3}{4}n$ on the size of L.

Finally, we analyze this version of **QuickHull**, given our lemma:

$$\begin{aligned}
T(n,h) &= O(n) + T(n_1,h_1) + T(n_2,h_2) \\
&\leq \alpha n + \beta n_1 \log h_2 + \beta n_2 \log h_2 \\
&\leq \alpha n + \beta \frac{3}{4} n \log h_2 + \beta \frac{1}{4} n \log h_2 \\
&\leq \alpha n + \beta \frac{n}{4} (\max_x (3 \log x + \log(h-x))) \\
&\leq \alpha n + \beta n \log h + \beta n (\frac{3}{4} \log \frac{3}{4} + \frac{1}{4} \log \frac{1}{4})
\end{aligned}$$

Going from the second to the third line, we use the lemma. Going from the fourth to the fith line, we use calculus. The constant term evaluates to $\approx -.8$, so that when $\beta \geq 2\alpha$, we simplify to

$$T(n,h) \leq \beta n \log h$$