Note: These lecture notes are based on lecture notes by Jeff Erickson and the textbook "Computational Geometry" by Berg et al.

1 2-D Convex Hull Problem

Convex Hull Problem:
Given: A set of points P in the plane.
Goal: Find the smallest, convex polygon containing all points in P

Definitions:

- *Polygon:* A region of the plane bounded by a cycle of line segments, joined end-to-end. The line segments are called *edges* and the points where there are joined are called *vertices*
- *Convex:* For any line segment connecting any two points in the polygon, all points in the line segment are also in the polygon.
- *Smallest:* Removing any point from the convex hull will violate convexity or the fact that it contains P.

This problem is equivalent to:

- Finding the *largest* convex polygon whose vertices are some subset of the points in P.
- Finding the set of all convex combinations of points in P (i.e. all points on any line segment between any pair of points).
- Finding the intersection of all convex sets containing P

Questions:

- Can there be a convex set of points (not necessarily a polygon) that contains P and has smaller area than the convex hull?
- What is the maximum number of vertices on the convex hull when $|P| = n$, and all points in P are in general position (see below)?

1.1 Applications

- *Recipes:* Each recipe for a crepe is a point in a 4-D space that gives the amount of each ingredient in the crepe, e.g. egg, milk, water and flour. Fundamentally, a crepe **is defined as** the convex-hull of all these points. Similarly, pancakes and flan are defined as separate convex hulls in the same space.
- *Robotics:* Find convex-hull of obstacles to simplify motion-planning problems
- *Chemical-Engineering:* Have several input mixtures of a liquid containing component A and B, and "other", at certain fractions. For example, the fractions of A and B can be $(.2, .4)$, $(.7, .2)$, $(.9, 0)$ (note that these don't necessarily add up to 1 because of the additional fraction of "other"). Can you achieve target (x, y) by mixing the given input mixtures? Solution: Is (x, y) in the convex hull of the input points?

2 Finding the Convex Hull in 2 dimensions

2.1 General Position

Throughout this class we will assume points are in *general position*: no 3 points are on a line. This is analogous to assuming now two numbers are the same in sorting.

Technique to remove this assumption is *Symbolic Perturbation*. Intuitively, the idea is to perturb the coordinates of every point by a uniformly random amount chosen for a range small enough to have no impact on the output. Then, with all but negligible probability, the points will be in general position.

2.2 CounterClockwise Order of 3 points

Consider 3 points $(a, b), (c, d)$ and (e, f). Then the three points are in *counterclockwise (ccw) order* iff the cross-product of the two vectors $(c, d) - (a, b)$ and $(e, f) - (a, b)$ is greater than 0. This holds iff

$$(f - b)(c - a) - (d - b)(e - a) > 0.$$

This is also equivalent to the slope of the line through (a, b), (c, d) being less than the slope of the line through (a, b), (e, f). In particular, if (a, b) is to the left of (c, d) and (e, f), then ccw order holds only if

$$\frac{d - b}{c - a} < \frac{d - b}{e - a}$$

Cross-multiplying, we get that the points are in ccw order iff

$$(f - b)(c - a) > (d - b)(e - a)$$

This final equivalence is true even if (a, b) is not the leftmost point.

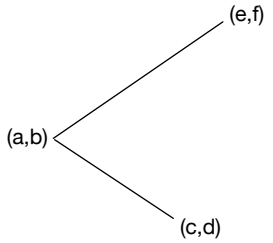

Figure 1. From the viewpoint of (a, b), (c, d) is ccw of (e, f). Thus, (a,b),(c,d) and(e,f) are in ccw order.

If $(a, b), (c, d), (e, f)$ are in ccw order, then we say that (e, f) is the *closest clockwise point* from (a, b).

Equivalently, we can define the orientation of 3 points p, q, r as the determinant of the points when given in homogenous coordinates (i.e. adding a 1 to the end of each point). That is

$$\text{Orient}(p, q, r) = \det \begin{pmatrix} p_x & q_x & r_x \\ p_y & q_y & r_y \\ 1 & 1 & 1 \end{pmatrix}$$

Then the sequence p,q, r is in ccw order (or makes a *strict left-hand turn*) iff $\text{Orient}(p, q, r) > 0$

In the 1-dimension case, Orient(p, q) is just $q - p$. In d-dimensional space, the definition above generalizes to $d + 1$ points and is related to the notion of *chirality* from math, physics and biology.

Comparing three points for purposes of finding the convex hull will be analogous to comparing a single pair of numbers in the sorting problem.

2.3 Jarvis' Algorithm

We start with a simple algorithm that will always find the convex hull, but may take time that is quadratic in the size of the set P. This is called Jarvis' algorithm, or sometimes "wrapping" or Jarvis' march.

Note that Jarvis' algorithm is analogous to insertion sort.

```
procedure JARVIS(P)
    ℓ ← leftmost point in P
    p ← ℓ
    repeat
        q ← closest clockwise point from p, among all points not yet in hull
        next(p) ← q                          ▷ next(p) is next point in convex hull
        p ← q
    until p = ℓ
end procedure
```

The step where we find q takes $O(n)$ time (to find a minimum). Thus the entire algorithm takes $O(hn)$ time where h is the number of points on the convex hull.

2.4 Divide and Conquer

The Divide and Conquer algorithm is analogous to QuickSort. We partition the points; recursively compute the hull of each partition; and then patch up the two hulls to get the hull of P.

```
procedure DIVIDEANDCONQUER(P)
    p ← point chosen uniformly at random in P
    L ← all points with x-coordinate less than or equal to that of p
    R ← P − L
    C_L ← convex hull of L (computed recursively)
    C_R ← convex hull of R (computed recursively)
    Merge C_L and C_R to get convex hull of P as follows:
    Create two bridges between the rightmost point in C_L and leftmost point in C_R
    while Either endpoint of either bridge is in a concave corner do
        Remove the vertex at the middle of this corner from the hull
        Update that bridge to connect the endpoints of that concave corner
    end while
end procedure
```

For an ordered sequence of points, p, q, r in a potential convex hull, we say that p, q and r form a *concave corner* if from p's viewpoint, r is ccw of q. Then q is the corner vertex that is removed from the hull.

The figure shows the merging. The red line segments are the bridges. The yellow wedges represent concave corners found. Note that the middle point of the wedge is the point that is always removed.

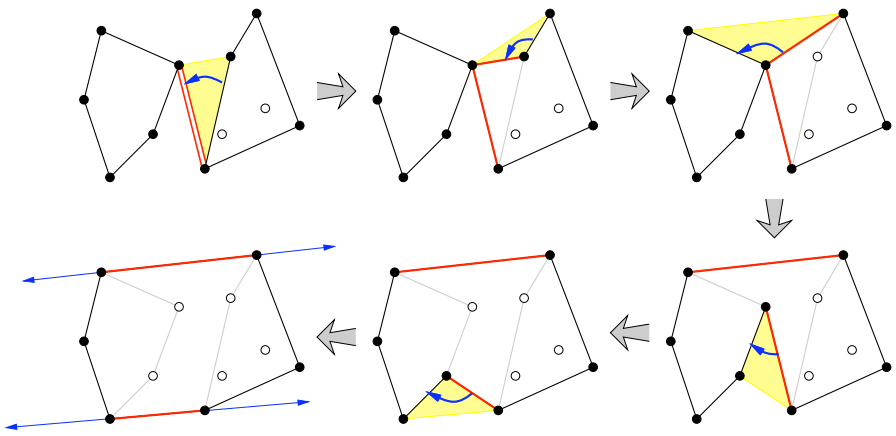

Figure 2. Merging Two Hulls in DivideAndConquer- From Jeff Erickson's lecture notes.

The runtime is a random variable. We can use linearity of expectation to create a recurrence relation for the expected run time. If $T(n)$ is the expected runtime of the algorithm when there are n points, then we have:

$$T(n) = \sum_{i=1}^{n} \frac{1}{n}(T(i) + T(n-i) + n).$$

This holds since we have the two recursive calls and then at most n time to merge the hulls. We c An inductive proof shows that the solution to this recurrence is $O(n \log n)$. We can show the induction by either using calculus to upper bound the sum with an integral and show that the integral is no more than $c(n \log n + n)$ for some constant c; or breaking up the sum into halves and bounding with algebra.

Finally, note that we can remove the randomness if we initially sort all points by the x coordinates and repeatedly by the point p as the median point in this sorted ordering.

2.5 Graham Scan

Graham scan is a deterministic $O(n \log n)$ algorithm that is efficient in practice. To simplify presentation of this algorithm, we'll separate the convex hull into two polygonal chains: the *upper hull*, and the *lower hull* (See Figure 3 left). Formally, a point p lies on the upperhull iff there is a support line passing through p such that all points in P lie on or below this line. Similarly, each vertex in the lower hull has a support line such that all points of P lie on or above that this line. The leftmost and rightmost vertices in P are common to both the upper and lower hull.

We define Graham's Scan for the upper hull. The algorithm for the lower hull is symmetric, and once both are computed, the upper and lower hull can be easily spliced together. The points of the upper hull will be stored on a stack, U.

The basic idea of the algorithm is to walk a candidate hull from left to right, making sure that the last 3 points under consideration - i.e. p_i and the top two elements on the stack, make a right-hand turn. If they fail to do so, we repeatedly pop off the middle point, until either there are fewer than 2 elements on the stack or p_i and the top two elements on the stack do make a right-hand turn.

See Figure 3 (right) illustrates the case where a right hand turn is formed by the points $p_{i_{j-2}}$, $p_{i_{j-1}}$, and p_{i_j}

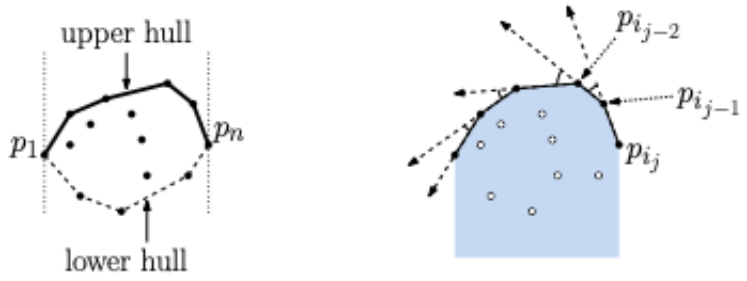

Figure 3. Left: upper and lower hull. Right:

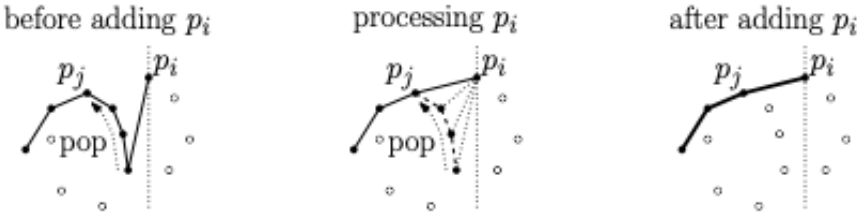

Figure 4. Graham Scan analysis

procedure GRAHAM'S SCAN(P)
Sort the points in increasing order of x-coordinates: $\langle p_1, p_2, \ldots p_n \rangle$.
Push p_1 and then p_2 onto U
for $i = 3$ `to` n **do**:
While $size(U) \geq 2$ and Orient$(p_i, first(U), second(U)) \leq 0$, pop U
Push p_i onto U
end for
end procedure

2.5.1 Correctness

For all $i \in [1, n]$, Let $P_i = \langle p_1, \ldots, p_i \rangle$

Lemma 1. *After the point p_i is pushed, the contents of U from top to bottom give the vertices of the upper hull of P_i in right to left order.*

Proof: By induction on i from 2 to n.

BC: When $i = 2$, there are only 2 points on U. These two points clear form the upper hull of P_2.

IS: First, note that p_i is the rightmost point in P_i and so must be the rightmost point in the upper hull of P_i. Let p_j be the point in the upper hull of P_i that precedes p_i. To establish the inductive step, we will show three things for iteration i of the for loop.

1. When entering iteration i, p_j is on the stack
2. Every point above p_j on the stack will be popped off during iteration i.
3. The point p_j will note be popped off during iteration i.

[**Jared:** *remainder of proof is a writeup of the in-class exercise*] To show part (1), note that by the IH, when entering iteration i, the stack contains the upper hull of P_{i-1}. Point p_j is on the upper hull of P_{i-1} iff it supports a line such that all points in P_{i-1} are below that line. But the line containing p_i to p_j is exactly such a line, since all points in P_i - and thus all points in P_{i-1} - are below this line, by definition of the upper hull of P_i. Thus, p_j is on the stack when entering iteration i.

To show parts (2) and (3), again by the IH, at the beginning of the iteration the stack contains a convex hull of P_{i-1}, which means that for all k, $j < k < i$, for vertex p_k that lies on the upper hull of P_{i-1}, the orientation of p_i, p_k and p_k's predecessor on the upper hull will be negative (See Figure 4 (middle)). But, when $k = j$, the orientation switches to positive (See Figure 4 (right)).

Another way to think about parts (2) and (3) is to note that for a fixed point p_i, the slopes of lines formed with p_i and the vertices of the hull as we go from left to right are bitonic (i.e. the slopes are first decreasing and then increasing). [1] Then, note that the minimum slope is for the line ℓ_{p_j,p_i}, since the points to the left of p_j are below this line. Thus, the slopes between p_i and points to the right of p_j on the upper hull of P_{i-1} are *increasing*. Hence, if we draw a line from p_i to the predecessor of p_k, for some $j < k < i$ on the upper hull, then p_k will be *below* that line. Thus, the orientation of p_i, p_k and the predecessor of p_k is negative.

By the above argument, all the points in between p_i and p_j on the stack U will be popped. □

2.5.2 Runtime Analysis

Lemma 2. *Graham's Scan runs in $O(n \log n)$ time.*

Proof: The initial sorting takes $O(n \log n)$ time. Let d_i be the number of points popped on processing point p_i. The runtime for the i-th iteration is thus $d_i + 1$. So total runtime in loop is

$$\sum_{i=1}^{n} d_i + 1 = n + \sum_{i=1}^{n} d_i$$

But $\sum_{i=1}^{n} d_i = n$, since each point can be popped at most once. Thus, the total time spent in the loop is $O(n)$, and so the total runtime of the algorithm is dominated by the sorting cost, which is $O(n \log n)$. □

2.6 Chan's algorithm

Can we get $o(n \log n)$ when the convex hull has $o(n)$ vertices? Yes, via Chan's algorithm. Chan's algorithm is output sensitive in that the runtime is $O(n \log h)$, where h is the number of vertices in the hull.

To understand it better, assume first that we know h in advance. Then we do the following

procedure CHAN(P)
  Partition P arbitrarily into n/h sets of size h
  Compute convex hulls of each partition (using say Graham scan)
  Compute convex hull of all these hulls via Jarvis' algorithm (i.e. "wrapping")
end procedure

2.6.1 Runtime if we know h

Mini Hulls: Total runtime to compute mini-hulls is $O((n/h)(h \log h)) = O(n \log h)$.

[1] We'll make use of this fact again in Chan's algorithm!

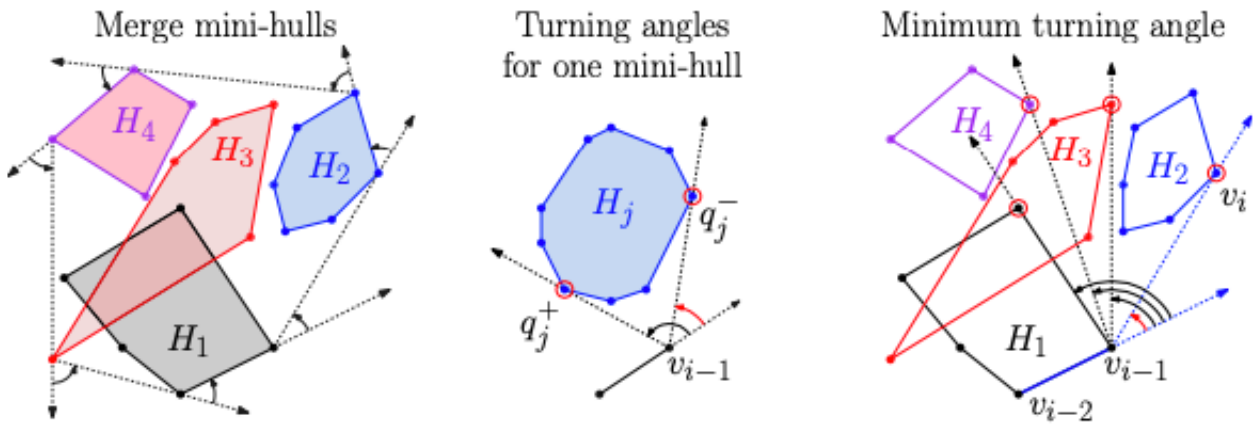

Figure 5. "Wrapping" the mini-hulls

Wrapping: To wrap, we repeatedly need to find the tangent line between a point p and each sub-hull. This can be done in $O(\log h)$ time per sub-hull via a type of binary search. In particular, we have the following lemma, which is left as an (interesting) exercise. Hint: Do binary search for inflection points. [**Jared:** *In-class exercise.*]

Lemma 3. *Assume we are given (1) a convex polygon $Q = \langle q_1, \ldots q_m \rangle$, where the m vertices are stored in an array in sorted CCW order; and (2) a point p outside of Q. Then in $O(\log m)$ time, we can compute the two vertices q^+, $q^- \in Q$ such that the lines ℓ_{p,q^+} and ℓ_{p,q^-} are support lines of Q.*

Proof hint: Think about doing binary search for points of inflection in the slopes.

Since there are n/h mini-hulls, using Lemma 3, it takes $O((n/h)\log h)$ time to find the successor of any point p. Since we find the successor $h-1$ times, the wrapping takes $O(n \log h)$ time.

So everything works great if we know h in advance. What if we don't?

2.6.2 Guessing h

In the real Chan's algorithm, we guess increasingly large values of h. Let $h' = 2$ be our first guess. If our guess is too small, we square h' and try again. In the final iteration, $h' \leq h^2$, so the total cost of last iteration is $O(n \log h^2) = O(n \log h)$.

Say that there are k total iterations, then the cost of all iterations is at most:

$$\sum_{i=1}^{k} Cn \log 2^{2^i} = O\left(n \sum_{i=1}^{k} 2^i\right).$$

Since a geometric summation is always a constant times its largest term, total runtime is big-O of the runtime of the last iteration which is $O(n \log h)$.

3 Open Problems

There are still many things we don't know about the convex hull. Below is an example problem in distributed computing that I'm interested in.

Convex-hull in the CONGEST model. Each point is a node with connections to all neighbors correspondign to points within a certain distance (for a point on the hull, this distance is at least as large as the distance to both neighbors on the hull). The nodes can all compute and broadcast in parallel, but in each round, each node can send only O(polylog(n)) bits to all its neighbors. What is the minimum number of rounds needed to compute the convex-hull? At the end of the computation, each node should output: (1) whether or not it is on the hull; and (2) if it's on the hull, it's clockwise and counterclockwise neighbors.