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 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 regions 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 oil containing component A and B at a certain ration. For example, have mixtures that are $(.3, .7)$, $(.5, .5)$, $(.9, .1)$. Can you achieve goal ratio (x, y) by mixing the input mixtures? Solution: Is (x, y) in the convex hull of the input points?

1.2 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 3 points

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

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

One can also think of the slope of the line given by (a, b), (c, d) is less than the slope of the line given by (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.

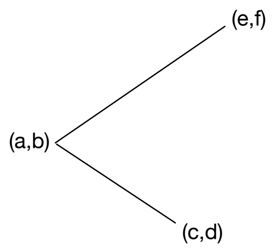


Figure 1. From the viewpoint of (a, b), (c, d) is ccw of (e, f)

2.1 Jarvis' Algorithm (Wrapping)

```
procedure JARVIS(P)
    $\ell \leftarrow$ leftmost point in P
    $p \leftarrow \ell$
    repeat
        $q \leftarrow$ closest clockwise point from $p$, among all points not yet in hull
        $next(p) \leftarrow q$                    ▷ next(p) is next point in convex hull
        $p \leftarrow q$
    until $p = \ell$
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.

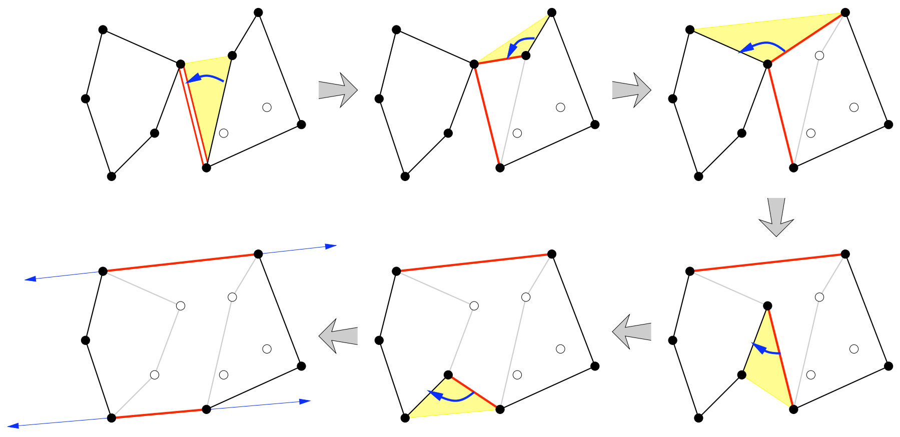


Figure 2. Merging Hull - From Jeff Erickson's lecture notes.

2.2 Divide and Conquer

In this algorithm, 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)
    $p \leftarrow$ point chosen uniformly at random in $P$
    $L \leftarrow$ all points with $x$-coordinate less than or equal to that of $p$
    $R \leftarrow P - L$
    $C_L \leftarrow$ convex hull of $L$ (computed recursively)
    $C_R \leftarrow$ convex hull of $R$ (computed recursively)
    "Merge" $C_L$ and $C_R$ to get convex hull of $P$
    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.

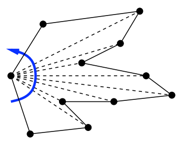
The figure shows the merging. The red line segments are the bridges. The yellow wedges represent concave corners found. Note that the middle point of the wedge is the point that is always removed.

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:

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

since we have the two recursive calls and at most n time to merge the hulls. An inductive proof shows that the solution to this recurrence is $O(n \log n)$. There is some trickiness in pushing through the induction - it helps to break the sum into halves.

2.3 Graham Scan



A simple polygon formed in the sorting phase of Graham's scan.

Figure 3. Sorting counter-clockwise wrt to leftmost point ℓ

```
procedure GrahamScan(P)
    $\ell \leftarrow$ leftmost point in $P$
    Sort all points in $P$ in CCW order with respect to $\ell$
    Let $p$, $q$ and $r$ be consecutive vertices in the sorted $P$. (Imagine there are 3 pennies on these vertices.)
    repeat
        if $p$, $q$ and $r$ are in ccw order then
            Output vertex back penny is on as part of convex hull
            Move back penny forward to the successor of $r$ (in $P$)
        else
            Remove $q$ from $P$, move middle penny back to predecessor($p$)
        end if
    until r equals $\ell$
end procedure
```

2.3.1 Runtime

Whenever a penny moves forward, it is to a vertex never seen before, so the first part of the if statement happens $O(n)$ times.

Whenever a penny moves backward, a vertex is removed from P, so this step happens $O(n)$ times.

Thus, total runtime of algorithm is dominated by the sorting which takes $O(n \log n)$ time.

2.3.2 Correctness

Prove that the output contains all of the vertices of the convex hull.

For a vertex to get thrown out, it must be CCW of a vertex added to the hull (i.e. it is in a concave corner)

Prove that the output contains only vertices in the convex hull.

Show that we can compute the bottom hull. Top hull correctness proof is symmetric. (*Bottom hull* is set of points in hull starting from ℓ until next point in hull induces a line with non-positive slope.)

Proof for bottom hull is by induction. To show: When we've added points $p_1, \ldots p_i$ to the convex hull, that these points form the bottom convex hull of the sorted points in P up to point p_i

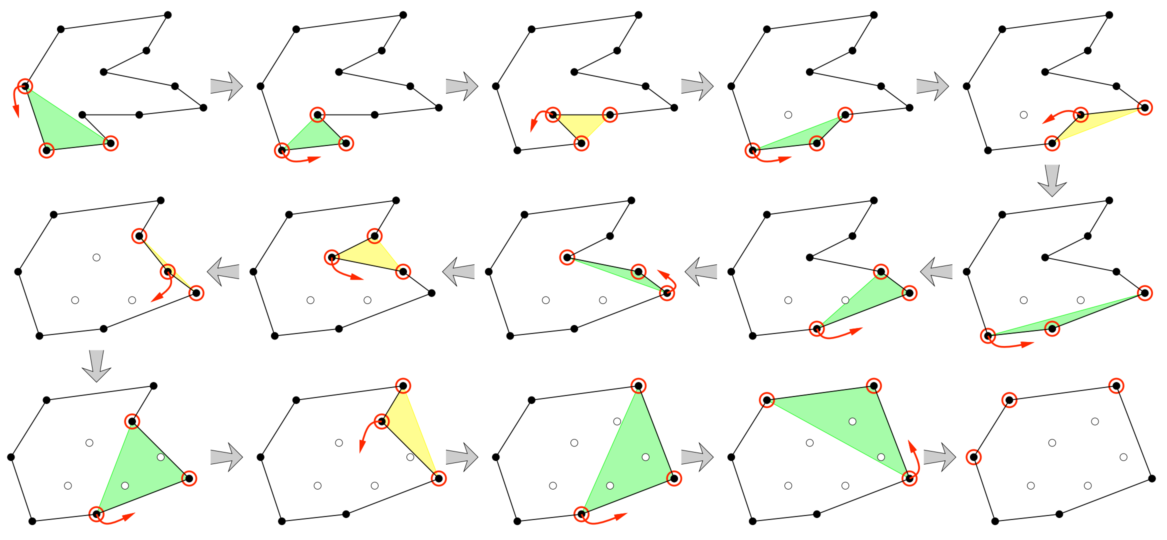


Figure 4. Steps of Graham Scan

Inductive step: Assume the first $i-1$ vertice processed $p_1 = \ell, p_2, p_{i-1}$ for the bottom hull of all vertices through p_{i-1}.

Now consider when we add p_i to the convex hull. Note: the chain of vertices $p_1, \ldots p_i$ is below

2.4 Chan's algorithm

Chan's algorithm is output sensitive in that the runtime is $O(n \log h)$.

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 “wrapping” (as in Jarvis’ march)
end procedure
```

Total runtime is $O(n/h(h \log h)) = O(n \log h)$ for the recursive calls. When we ”wrap”, we repeatedly need to find the tangent line between a vertex p and any sub-hull. This can be done in $O(\log h)$ time via a type of binary search. Since there are n/h subhulls, takes $O((n/h) \log h)$ time to find the successor of p. Since we find the successor $h-1$ times, wrapping takes $O(n \log h)$ time.

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

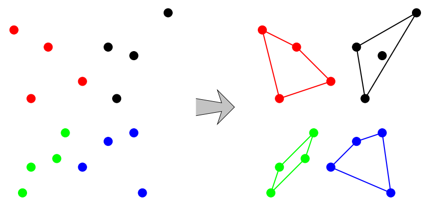


Figure 5. Sorting counter-clockwise wrt to leftmost point ℓ

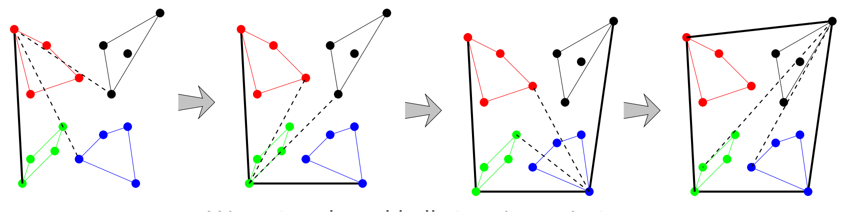


Figure 6. Steps of Graham Scan

2.4.1 Guessing h

In Chan's algorithm, we actually guess increasingly large values of h. Let $h' = 3$ 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

$$\sum_{i=1}^{k} O(n \log 3^{2^i}) = O\left(n \sum_{i=1}^{k} 2^i\right)$$

.

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

3 Open Problems

Convex-hull in the CONGEST model. Each point is a wireless node. Can compute in parallel, but in each round, can broadcase O(polylog(n)) bits. What is the minimum number of rounds needed to compute convex-hull?