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

1 Convex Hull

A set of points is said to be *convex* if it contains all points in the line segments connecting each pair of its points.

The *Convex hull* of a set of points P may be defined (equivalently) as:

1. The minimal convex set containing P
2. The intersection of all convex sets containing P
3. The union of all simplexes over points in P
4. The union of all convex combinations of points in P

A *simplex* is a polytope with the smallest number of vertices for a space of given dimensionality. In 2-D a simplex is a triangle; in 3-D, a tetrahedron.

A *convex combination* of a set of points $P' = \{p_1, \dots p_\ell\}$ is any point $\sum_{i=1}^{\ell} \lambda_i p_i$, for any set of numbers λ_i, $i \in [1, n]$ such that $\forall_{i\in[1,n]} \lambda_i \geq 0$ and $\sum_{i=1}^{\ell} \lambda_i = 1$

1.1 Algorithmic 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*
- *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)?

1.2 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?
- *Biology:* Determining the territory that is "owned" by an animal or a pack.
- *Economics:* Making Budget sets and preference functions convex, in order to apply theorems determining market equilibria.
- *Computer Science:* Via duality and projections, we can use a solution to convex hull to also find Voronoi Diagrams, Delaunay Triangulations and Upper Envelopes.

1.3 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, x, y, z. These points are in counterclockwise (ccw) order iff the crossproduct of the vectors $y - x$ and $z - x$ is positive.

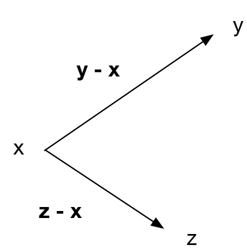

Figure 1. From the viewpoint of x, y is ccw of z. The case where x is on the other side of the line supported by y and z is also handled by using the cross product.

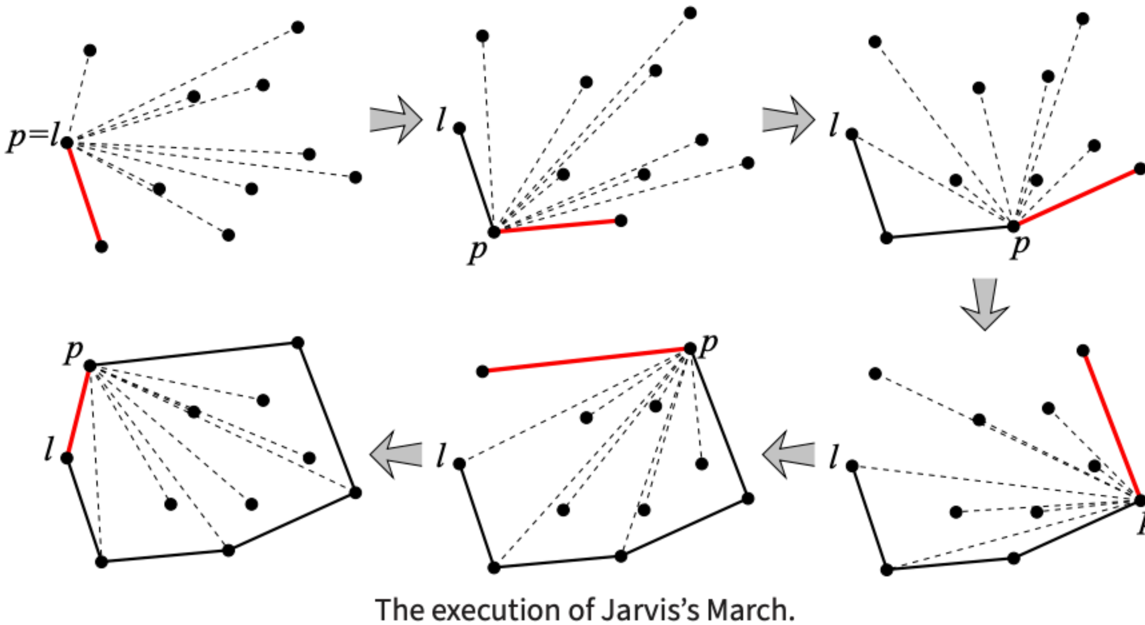

Figure 2. Jariv's march From Jeff Erickson's lecture notes.

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.

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

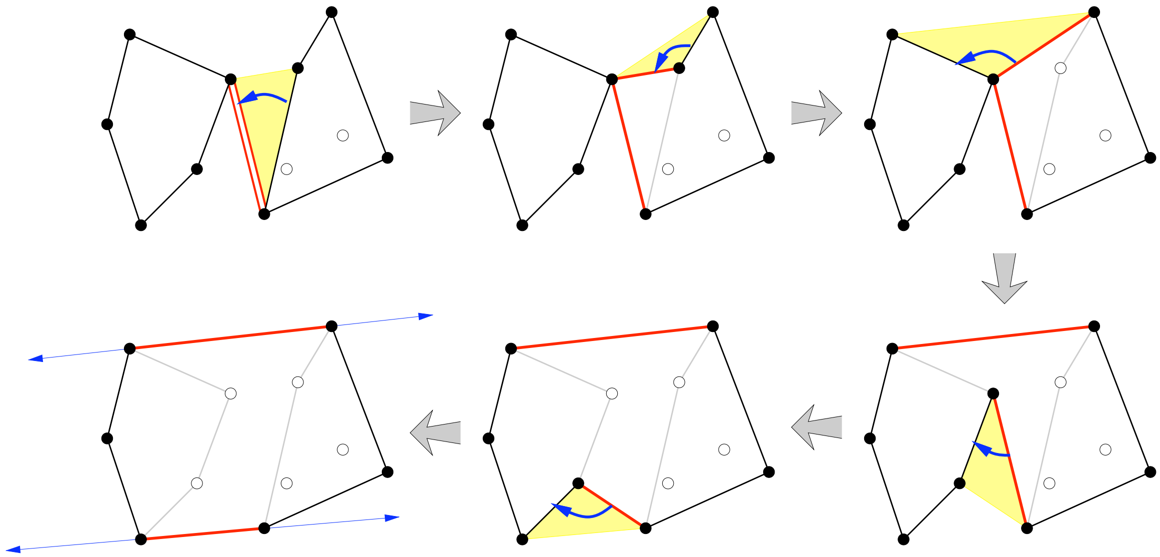

Figure 3. Merging Hull - From Jeff Erickson's lecture notes.

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

procedure GrahamScan(P)
  $\ell \leftarrow$ leftmost point in P
  Sort all points in P in CCW order with respect to ℓ

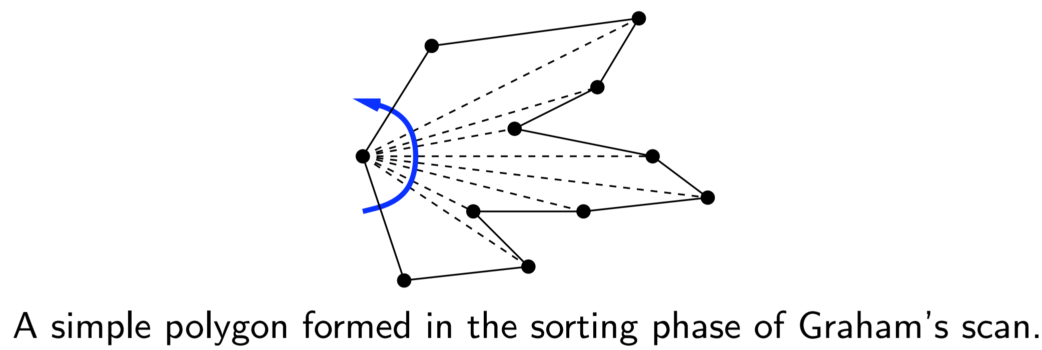

Figure 4. Sorting counter-clockwise wrt to leftmost point ℓ

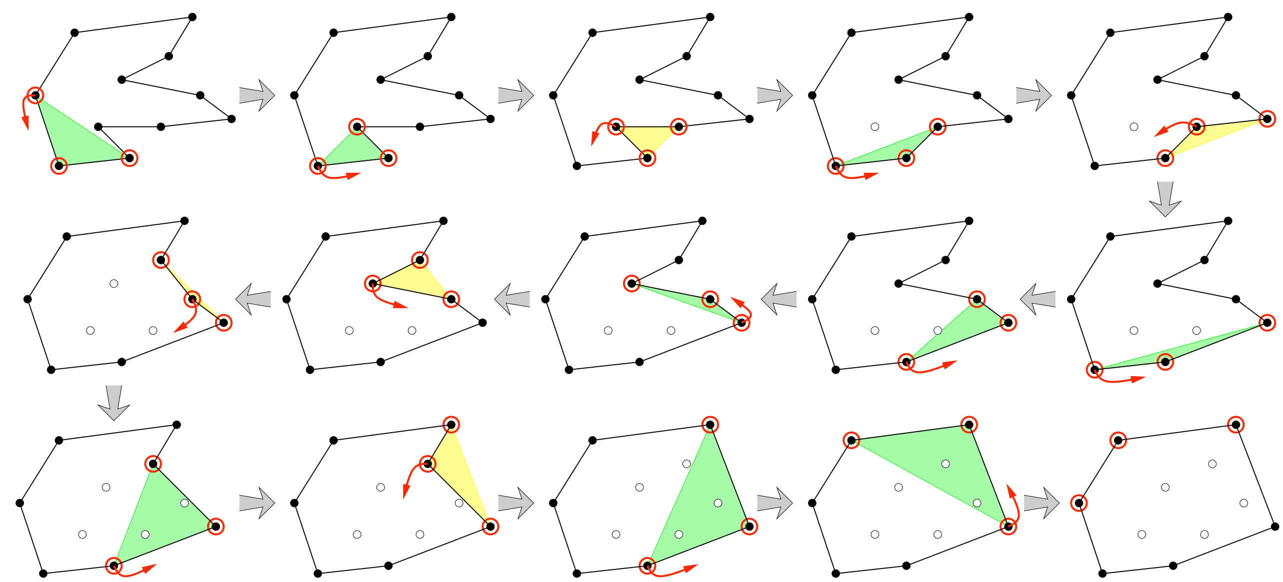

Figure 5. Steps of Graham Scan

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 ℓ
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.

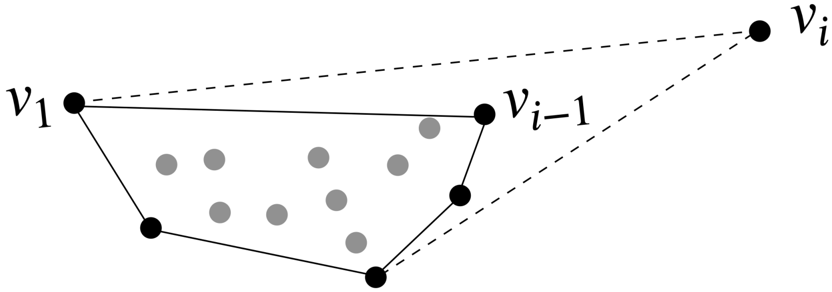

Figure 6. Inductive Step for Graham Scan

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

The following is based on the proof from: http://www.cs.cmu.edu/afs/cs/academic/class/15456-s10/Homeworks/hw1-answers.pdf

Theorem: Graham Scan outputs the convex hull of P.

Proof: Let $v_1, v_2, \ldots v_n$ be the points of the input sorted in CCW order. For any $1 \leq j \leq n$, let $V_j = v_1, v_2, \ldots v_j$; and let $CH(V_j)$ be the convex hull of these j points. Define $OS(V_i)$ (for output stack) to be the vertices output by the algorithm as being on the hull, plus the 3 vertices covered by pennies **when the condition of if statement is met (i.e. we have a green triangle in Figure 2.3)** after processing points V_i.

We will prove that for all i, that $OS(V_i) = CH(V_i)$. We do this via induction on i.

BC: (i=3) $OS(V_3)$ is the set of all 3 vertices in V_3 in CCW order, and this is $CH(V_3)$.

IH: $OS(V_{i-1}) = CH(V_{i-1})$.

IS: Consider $OS(V_i)$. By the IH, $OS(V_{i-1})$ is in CCW order, and by the properties of the algorithm, when v_i is placed on the output stack, $OS(V_i)$ is in CCW order (See Figure 2.3.2).

We also know that $OS(V_i)$ starts at vertex v_1 and ends at vertex v_i. Our goal is to show

1. All points in V_i are inside $OS(V_i)$
2. All points in V_i that are not on $CH(V_i)$ are not in $OS(V_i)$.

We first show (1). By the IH, all points in V_{i-1} are inside $OS(V_{i-1})$. When processing the point v_i, we may have removed some points in $OS(V_{i-1})$. But, by the properties of the algorithm, these are exactly the points that are in the wedge induced by the line from v_1 to v_i and the line from v_i to some previous point on $OS(V_{i-1})$ (See Figure 2.3.2). Thus all points in V_i are within $OS(V_i)$.

We next show (2). By the IH, $OS(V_{i-1})$ does not contain any points not on the hull $CH(V_{i-1})$. Since the algorithm never revisits a point that is discarded, these points will also not be on $OS(V_i)$. But, there may be points on $OS(V_{i-1})$ that are not on $CH(V_i)$. However, these are exactly the points that are strictly inside the wedge induced by v_1, v_i and the closest clockwise point to v_i on $CH(V_{i-1})$ (again See Figure 2.3.2). The algorithm pops off all of these points from the output stack while processing point v_i. Thus v_i is correctly added to $CH(V_{i-1})$. This creates a new convex polygon that includes v_i as a vertex. $\square$

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?

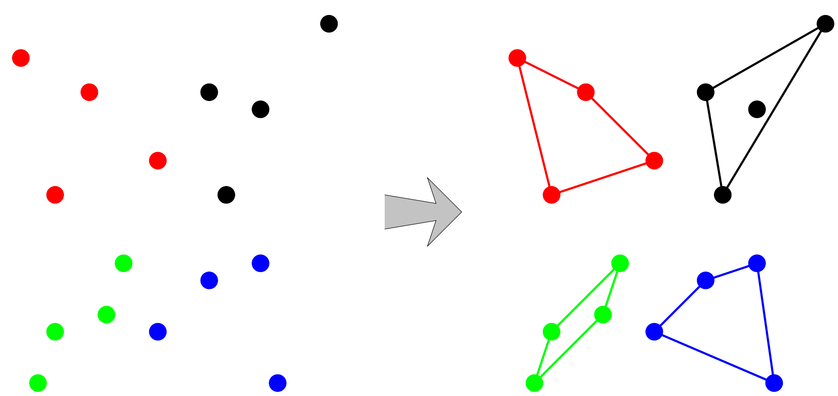

Figure 7. Chan's algorithm: Compute hulls of partitions

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 $O(n \log h)$.

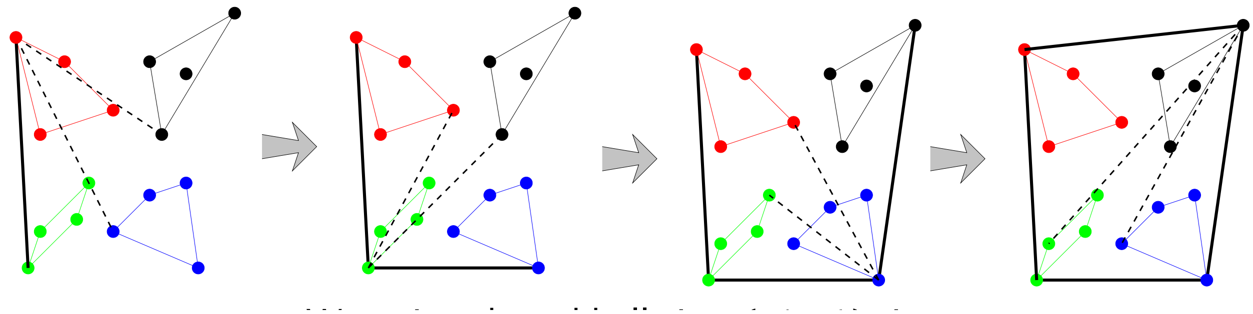

Figure 8. Chan's algorithm: "wrap" the subhulls

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?

4 Appendix

4.1 A note on cross products

Let $x = (a, b), y = (c, d)$ and $z = (e, f)$. 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.