Adapted from CS498TC [Eri22]

Convex Hulls

Sam Ruggerio

Outline

Section 1

Computational Geometry

What is Computational Geometry

- Algorithms and data structures with discrete geometric objects!
- Working with points, lines, polygons, planes, polytopes, etc.
- Low dimensional computational geometry has applications in graphics, motion planning, modeling, mesh processing, etc.
- High dimensional CG is the basis of many ML algorithms.
- (The part of theory with cool pictures)

Assumptions

- We will often deal with items in real space ($\mathbb{R}^d$)
- This means having to do math with real numbers (including roots if we ever want to find distances)
- So our model of computation is the **Real RAM**:
  - $+, -, /, \times, \sqrt{x}, x = 0?, x > 0?$ are all $O(1)$ time between real numbers
  - Each number takes $O(1)$ space (no bit complexity)
  - Stored exactly (no floating point)
  - Notably, transcendental functions are not supported (e.g. sin, cos, tan).

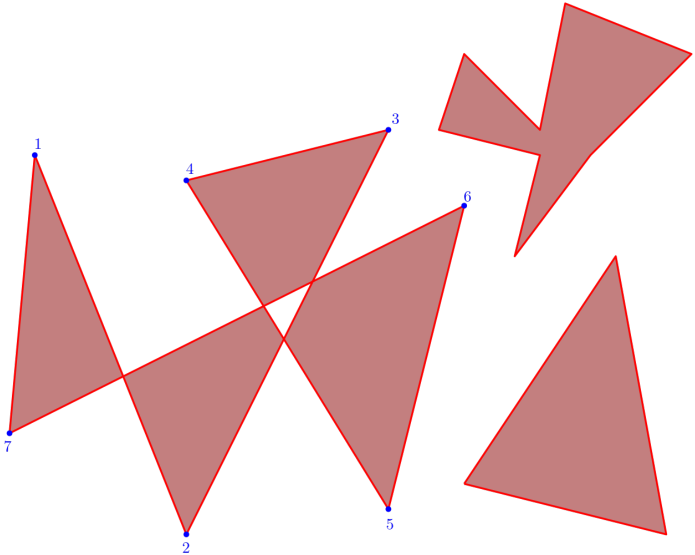
Assumptions

- Points placed in the plane can construct many other objects (circles, lines, etc...)
- Often, we want to refer to these structures (relatively) uniquely, without degenerate cases.
- Our assumption for most problems will be **General Position:**
  - No two points are in the same position
  - No three points are co-linear
  - No four points are co-circular
- General position is a simplification, you can (almost) always change an input set to general position, or design an algorithm to handle special position.

Definitions

- A Polygon Q is an *ordered* circular sequence of Points P.
- A polygon is *simple* if it does not self-intersect or have holes
- A polygon is *convex* if $\forall p, q \in Q, \overline{pq} \in Q$.
- A polygon is *closed* if there is a segment for each pair of adjacent points in P.

Polygons



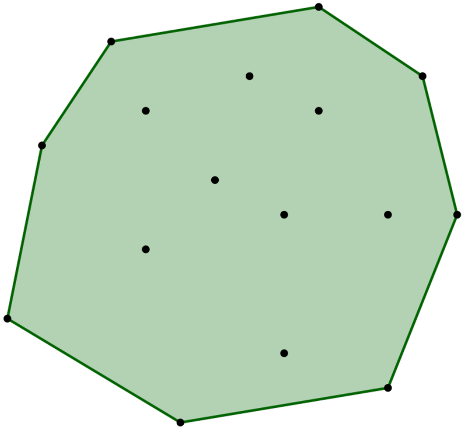
Σ

Section 2

Convex Hulls

Convex Hulls

- Problem: Given a point set P, we want to compute the *convex hull* Q



Convex Hulls

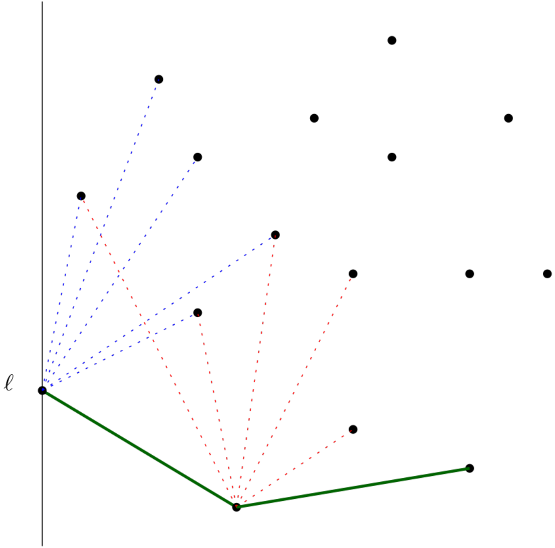
- Intuitively, we want to place a rubber band around all the points.
- Formally, we want to compute the smallest convex polygon containing all the points.
- We can describe an algorithm which just *wraps* the polygon.

Jarvis March

Jarvis March algorithm computes a convex hull via the following:

- Find the leftmost point ℓ, set as current point.
- Repeat until we return to ℓ:
  - Compare slope of each point relative to current point
  - Pick point with least slope, add to hull and set as current point.
- We could go clockwise and pick greatest slope, by convention polygons are given in counter-clockwise order.

Jarvis March



Jarvis March

Runtime?

- $O(n)$ to find leftmost point
- $O(n)$ slopes to compare
- Repeat for each point on the hull, h, $O(h)$ times.
- $O(nh) \implies O(n^2)$ in the worst case.

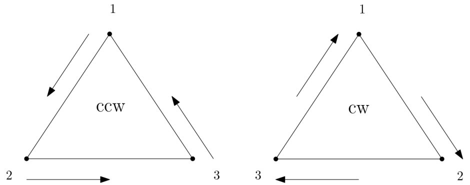
Jarvis March is *output sensitive*. It does better when the output structure is small.

Doing Better

- Jarvis March is the slightly-clever version of a brute force solution.
- But convex hulls are nice structures! Surely we can exploit some property
- We might be able to provide a greedy solution based off of local convexity
- If a 2D simple polygon is locally convex everywhere, it must be convex.
- But how do we test convexity?

Orientation

- We have a circular sequence of points, which means the ordering of points can be viewed as *clockwise* or *counter-clockwise*



- We determine orientation based off of the slope between one point and the rest.

Orientation Test

- Assume p_1 is the leftmost point of p_1, p_2, p_3, then:
  - If the slope $\overline{p_1 p_2}$ is greater than $\overline{p_1 p_3}$, it's clockwise
  - Otherwise, it's counterclockwise
  - (If it's equal, its flat, which shouldn't happen under general position)
- To simplify, we're doing the following test:

$$\frac{y_2 - y_1}{x_2 - x_1} > \frac{y_3 - y_1}{x_3 - x_1}$$

It's just math!

Following through:

$$\frac{y_2 - y_1}{x_2 - x_1} > \frac{y_3 - y_1}{x_3 - x_1}$$

$$y_2x_3 - y_2x_1 - y_1x_3 + y_1x_1 > y_3x_2 - y_1x_2 - y_3x_1 + y_1x_1$$

$$y_1x_2 + y_3x_1 - y_3x_2 + y_2x_3 - y_2x_1 - y_1x_3 > 0$$

It's just math!

Following through:

$$
\begin{aligned}
\frac{y_2 - y_1}{x_2 - x_1} &> \frac{y_3 - y_1}{x_3 - x_1} \\
y_2x_3 - y_2x_1 - y_1x_3 + y_1x_1 &> y_3x_2 - y_1x_2 - y_3x_1 + y_1x_1 \\
y_1x_2 + y_3x_1 - y_3x_2 + y_2x_3 - y_2x_1 - y_1x_3 &> 0 \\
\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} &> 0
\end{aligned}
$$

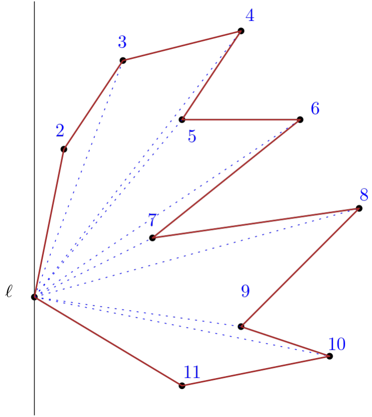
Recall that the determinant of 3 points in that form gives you 2 times the area of a triangle. Positive if it's ccw, negative if cw.

Orientation Test

- Take the (signed) determinant of 3 points
- If greater than 0, points are oriented counter clockwise
- If less than 0, points are oriented clockwise
- If equal to 0, points are flat.
- Runtime? $O(1)$

Graham's Scan

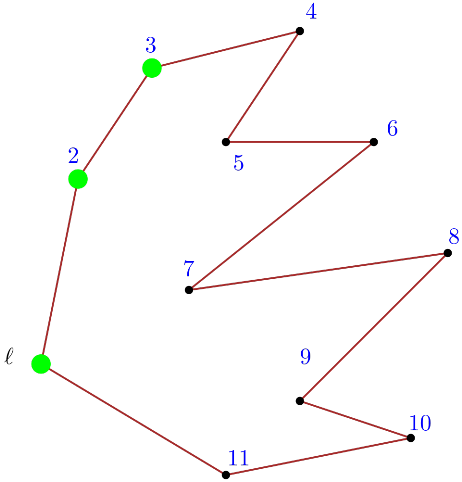
- First, find the leftmost point ℓ
- Then, create a polygon Q, sorted around ℓ



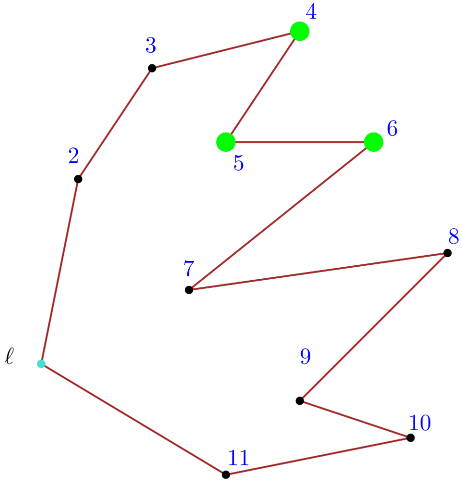
Graham's Scan

- Then, perform a repair on the polygon:
  - Mark the first 3 vertices
  - If they are convex, move all marks forward one vertex
  - If not, delete the middle mark, and mark the previous vertex
  - Repeat.

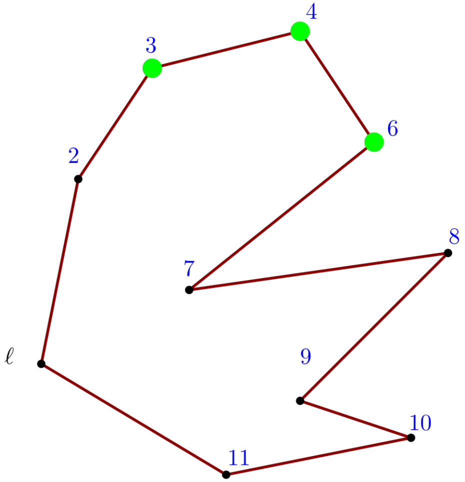
Graham's Scan



Graham's Scan



Graham's Scan



Graham's Scan

Runtime?

- Find leftmost point: $O(n)$
- Sort points around ℓ to construct a polygon: $O(n \log n)$
- 3 mark repair?
  - You could delete at most $O(n)$ points
  - Each deletion moves the marks back one spot
  - Otherwise, the scan moves forward one spot $\implies O(n)$ time.
- $O(n \log n)$ total runtime (dominated by sort)

Section 3

Optimal Convex Hulls

Doing Even Better

- We have an output sensitive but slow algorithm in $O(nh)$
- We have a fast, sorting-bound algorithm in $O(n \log n)$
- Can we get a fast output sensitive algorithm?

Chan's Algorithm

- In 1996, Timothy Chan (UIUC) came up with an output-sensitive *optimal* 2D convex hull algorithm
- We'll use both Jarvis March and Graham's Scan within our process.
- The high level idea is to construct a cluster of small convex hulls and merge them

Shattering the Hulls

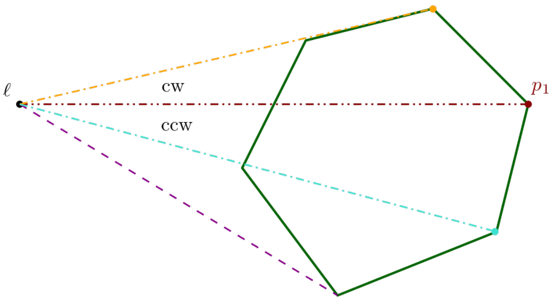
- Let h be a number we'll determine later.
- Break our input set P into $O(n/h)$ subsets of size $O(h)$.
- Compute the convex hull of each subset using Graham's Scan:
  - $O(n/h)$ subsets, each of size $O(h)$, thus taking $h \log h$ time, in total $n \log h$ time.

Candidate Points

- We need to pick final output vertices
- In each hull, we need a candidate point, we pick the lowest tangent point relative to our current output vertex.
- We'll find this candidate point via binary search on the hull.

Binary Search on Hulls

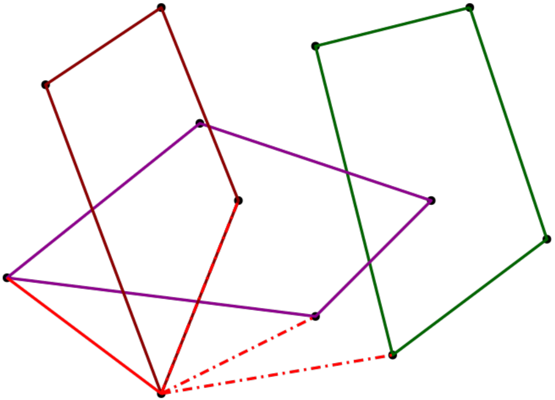
- We want to find the lower tangent of a hull H from some start point ℓ



- Once the orientation changes from cw, ccw, we found our point.

Putting It Together

- We can now select candidate points from our sub-hulls and create a global convex hull:



Runtime

- For each output vertex, in total h times:
  - We find our next candidate point via binary search in each subset
  - $O(n/h) \times O(\log h)$
  - Total: $O(n \log h)$
- Combined with our Graham's scan, we get a final runtime of $O(n \log h)$
- How do we determine the number of output points ahead of time??

Exponential Search

- Search h via exponential search, $h = 3, 9, 81, \ldots, 3^{2^k} = h_k$
- If you don't output a hull with the number of vertices guessed, you try again!
- $O(n \log h_1 + n \log h_2 + n \log h_3 + \cdots n \log h^2)$
  - This exponential doubling will eventually exceed h, but no more than h^2
- $O(n \log 3 + 2n \log 3 + \ldots 2^k n \log 3) \leq O(n \log h^2) \implies O(n \log h)$

Other things to think about!

- Convex Hulls in 3D? in n-D?
- Triangulating polygons?
- Test if a point is inside a non-simple polygon?
- Linear Programming from a CG perspective
- Shortest Paths in 2D space, or in planar graphs?

Questions?

Bibliography I

Jeff Erickson.

498tc spring 2022.

https://jeffe.cs.illinois.edu/teaching/compgeom/, 2022.

Accessed: 03-04-2024.