CONVEX HULL

CS16: Introduction to Data Structures & Algorithms
Outline

1. Overview
2. Convex Hull Overview
3. Graham Scan Algorithm
4. Incremental Algorithm
**Convex Hull**

- The convex hull of a set of points is the smallest convex polygon containing the points.
- A convex polygon is a nonintersecting polygon whose internal angles are all convex (i.e., less than 180 degrees).
- In a convex polygon, a segment joining any two lies entirely inside the polygon.

![Convex and Nonconvex Polygons]

- **Convex**: All internal angles are less than 180 degrees, and no segment outside the polygon.
- **Nonconvex**: Exists at least one internal angle greater than 180 degrees, or a segment lies outside the polygon.
Convex Hull (2)

- Think of a rubber band snapping around the points
- Remember to think of special cases
  - Colinearity: A point that lies on a segment of the convex is not included in the convex hull
Applications

- Motion planning
  - Find an optimal route that avoids obstacles for a robot
- Bounding Box
  - Obtain a closer bounding box in computer graphics
- Pattern Matching
  - Compare two objects using their convex hulls
Finding a Convex Hull

- One algorithm for determining a convex polygon is one which, when following the points in counterclockwise order, always produces left-turns.
Calculating Orientation

- The orientation of three points \((a, b, c)\) in the plane is clockwise, counterclockwise, or collinear
  - Clockwise (CW): right turn
  - Counterclockwise (CCW): left turn
  - Collinear (COLL): no turn

- The orientation of three points is characterized by the sign of the determinant \(\Delta(a, b, c)\)

\[
\Delta(a, b, c) = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix}
\]

```python
function isLeftTurn(a, b, c):
    return (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x) > 0
```
Calculating Orientation (2)

Using the isLeftTurn() method:

\[(0.5-0) \times (0.5 - 0) - (1-0) \times (1-0) = -0.75 \text{ (CW)}\]

\[(1-0) \times (1-0) - (0.5-0) \times (0.5-0) = 0.75 \text{ (CCW)}\]

\[(1-0) \times (2-0) - (1-0) \times (2-0) = 0 \text{ (COLL)}\]
Calculating Orientation (3)

- Let $a, b, c$ be three consecutive vertices of a polygon, in counterclockwise order
  - $b'$ is not included in the hull if $a', b', c'$ is non-convex, i.e. $\text{orientation}(a', b', c') = \text{CW or COLL}$
  - $b$ is included in the hull if $a, b, c$ is convex, i.e. $\text{orientation}(a, b, c) = \text{CCW}$, and all other non-hull points have been removed
Graham Scan Algorithm

- Find the anchor point (the point with the smallest y value)
- Sort points in CCW order around the anchor
  - You can sort points by comparing the angle between the anchor and the point you’re looking at (the smaller the angle, the closer the point)
Graham Scan Algorithm

- The polygon is traversed in sorted order and a sequence $H$ of vertices in the hull is maintained
- For each point $a$, add $a$ to $H$
  - While the last turn is a right turn, remove the second to last point from $H$
- In the image below, $p, q, r$ forms a right turn, and thus $q$ is removed from $H$. Similarly, $o, p, r$ forms a right turn, and thus $p$ is removed from $H$. 

![Diagram of Graham Scan Algorithm](image_url)
Graham Scan: Pseudocode

function graham_scan(pts):
    // Input: Set of points pts
    // Output: Hull of points
    find anchor point
    sort other points in CCW order around anchor
    hull = []
    for p in pts:
        add p to hull
        while last turn is a “right turn”
            remove 2\textsuperscript{nd} to last point
    add anchor to hull
    return hull

Note: this is very high-level pseudocode. There are many special cases to consider!
Graham Scan: Run Time

function graham_scan(pts):
    // Input: Set of points pts
    // Output: Hull of points
    find anchor point // O(n)
    sort other points in CCW order around anchor // O(nlogn)
    create hull (empty list representing ordered points)
    for p in pts: // O(n)
        add p to hull
        while last turn is a “right turn” // O(1) amortized
            remove 2\textsuperscript{nd} to last point
    add anchor to hull
    return hull

Overall run time: O(n\text{log}n)
**Incremental Algorithm**

- What if we already have a convex hull, and we just want to add one point $q$?
  - This is what you’ll be doing on the Java project after heap!

- Get the angle from the anchor to $q$ and find points $p$ and $r$, the hull points on either side of $q$
- If $p$, $q$, $r$ forms a left turn, add $q$ to the hull
- Check if adding $q$ creates a concave shape
  - If you have right turns on either side of $q$, remove vertices until the shape becomes convex
  - This is done in the same way as the static Graham Scan
Incremental Algorithm

Original Hull:

Want to add point $q$:

Find $p$ and $r$:

$p$, $q$, $r$ forms a left turn, so add $q$:

$o$, $p$, $q$ forms a right turn, so remove $p$:

$n$, $o$, $q$ forms a right turn, so remove $o$:

Incremental Algorithm

Since $m, n, q$ is a left turn, we’re done with that side.

Now we look at the other side:

Since $q, r, s$ is a left turn, we’re done!

• Remember that you can have right turns on either or both sides, so make sure to check in both directions and remove concave points!
Incremental Analysis

- Let $n$ be the current size of the convex hull, stored in a binary search tree for efficiency
  - How are they sorted? Around the anchor
- To check if a point $q$ should be in the hull, we insert it into the tree and get its neighbors ($p$ and $r$ on the prior slides) in $O(\log n)$ time
- We then traverse the ring, possibly deleting $d$ points from the convex hull in $O((1 + d) \log n)$ time
- Therefore, incremental insertion is $O(d \log n)$ where $d$ is the number of points removed by the insertion