Computational Geometry

Compute properties of a set of points, lines or geometric objects (defined by points and lines)

Properties: extent, intersection, proximity relationships

Applications:

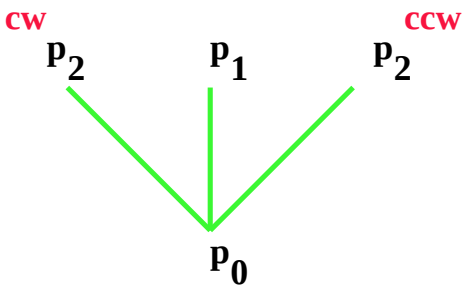
- graphics (e.g., hidden line removal)
- robotics (e.g., path planning, object avoidance)
- design (e.g., component placement, packaging)
- statistics (e.g., nearest neighbor)
- sensor planning (e.g., area of observation calculation)

Line Segments

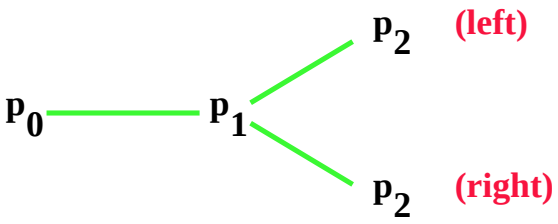
$$\overline{p_1 p_2}$$

Questions

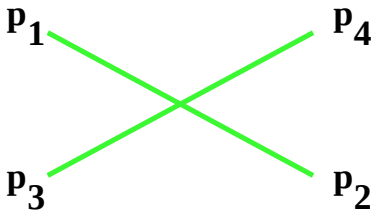
Is $\overline{p_0 p_1}$ clockwise (cw) or counterclockwise (ccw) of $\overline{p_0 p_2}$?



Turning direction from $\overline{p_0p_1}$ to $\overline{p_1p_2}$?



Do $\overline{p_1p_2}$ and $\overline{p_3p_4}$ intersect?



Cross Product

$$p_0\vec{p}_1 \times p_0\vec{p}_2$$

Assuming $p_0 = (0, 0)$, compute the signed area within (0,0), p_1, p_2, $p_1 + p_2$ $= (x_1 + x_2,\ y_1 + y_2)$.

$\vec{p_1} \times \vec{p_2} = x_1y_2 - x_2y_1$, the area of the parallelogram

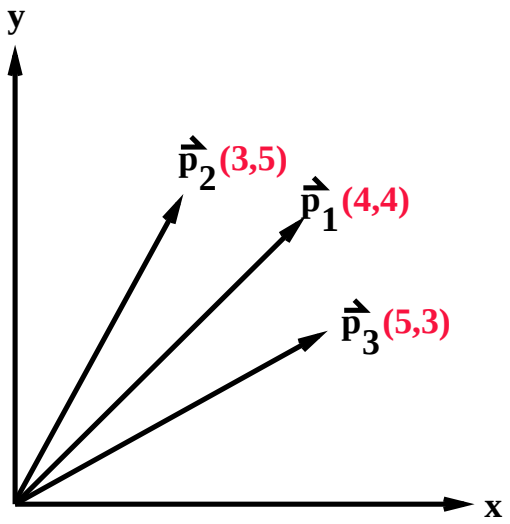
This is the determinant of the matrix

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$$

Note if $p_1 = (4,0)$ and $p_2 = (2,4)$, we are computing the signed area within (origin, p_1, p_2, $p_1 + p_2$) which is $\vec{p_1} \times \vec{p_2} = 16 - 0 = 16$.

If we compute $\vec{p_2} \times \vec{p_1}$ the signed area is $0 - 16 = -16$.

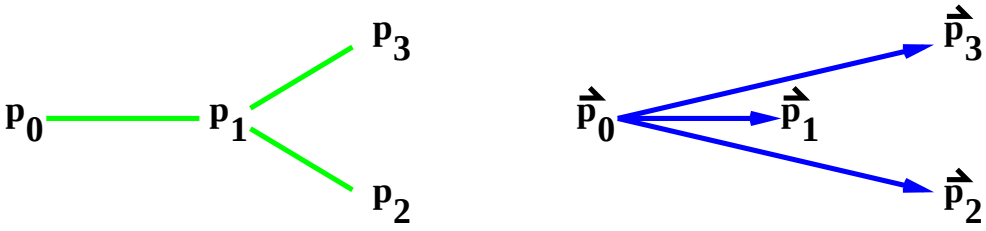
Example



$$\vec{p_1} \times \vec{p_2} > 0, \ \vec{p_2} \text{ ccw from } \vec{p_1}$$
$$\vec{p_1} \times \vec{p_3} < 0, \ \vec{p_3} \text{ cw from } \vec{p_1}$$

Therefore there is a cw relation by translating p_0 to the origin and taking the cross product.

Turn



ccw

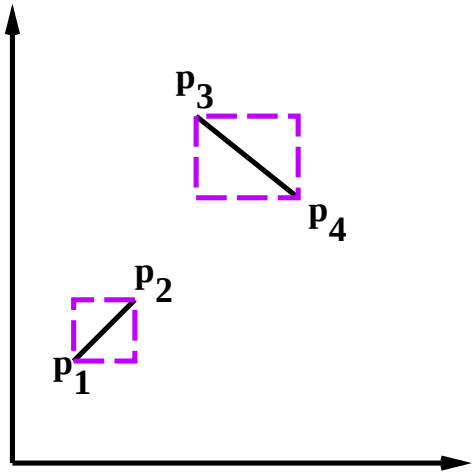
$$\vec{p_1} \times \vec{p_3} > 0 \longrightarrow \_\_\_\_\_$$

cw

$$\vec{p_1} \times \vec{p_2} < 0 \longrightarrow \_\_\_\_\_$$

Intersection

1. Quick rejection by bounding box

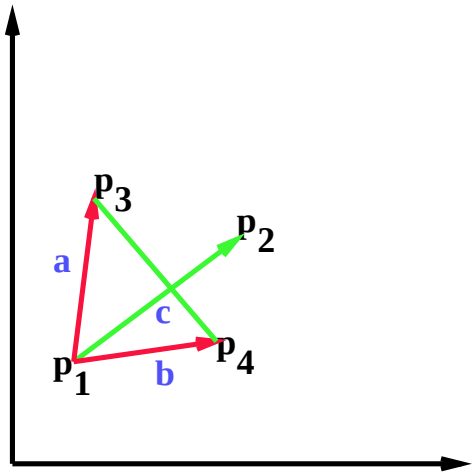


2. Straddle: One line segment l_1 straddles another line segment l_2 if

   - One point of l_1 on one side of l_2 and

- The second point of l_1 lies on the other side of l_2

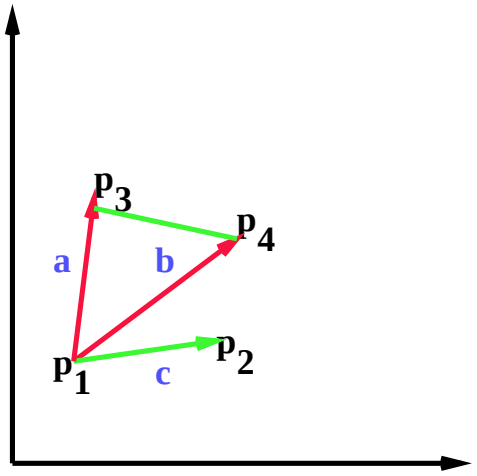
Example



$$\vec{a} \times \vec{c} < 0$$
$$\vec{b} \times \vec{c} > 0$$

Different Sign: Straddles

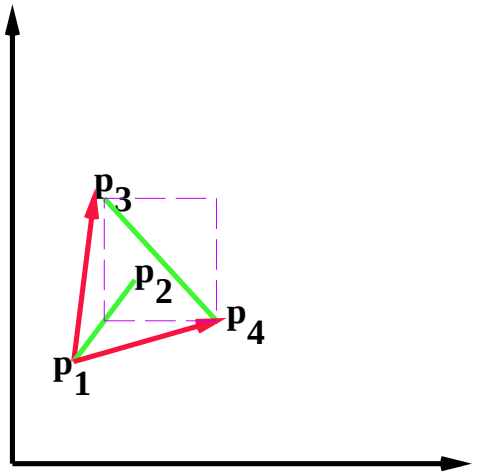
Example



$$\vec{a} \times \vec{c} < 0$$
$$\vec{b} \times \vec{c} < 0$$

Same Sign: No straddle

Example



Check p_1p_2 with p_3p_4, and p_3p_4 with p_1p_2

Any intersecting segments

Sort segments by left endpoints

Pass a vertical sweep line over the segments

Put segment in RB tree when hit left endpoint, order by y value

Check intersection between new and ABOVE and between new and BELOW

Remove segment from RB tree when hit right endpoint

Check intersection between ABOVE and BELOW

If two line segments intersect, they will eventually be neighbors

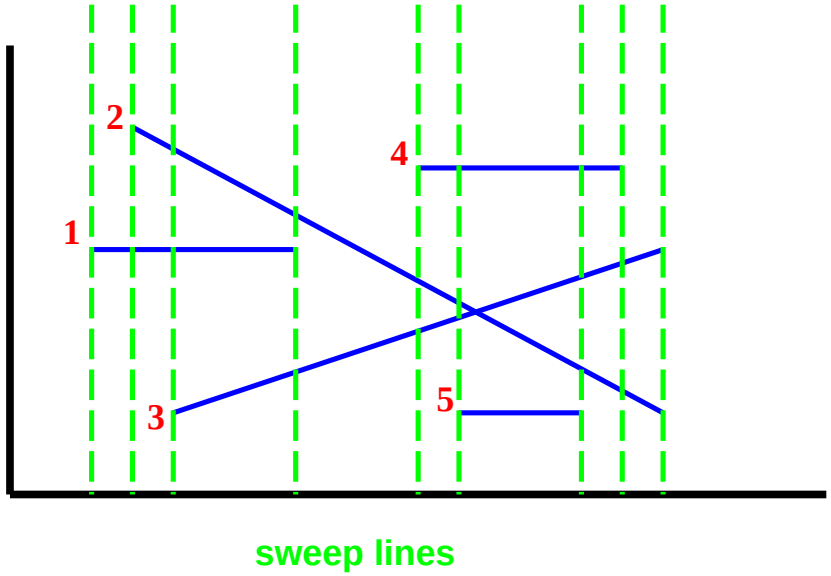
Neighbors when one of the segments is added, and the first check will catch the intersection

Neighbors when another segment is deleted, and the second check will catch the intersection

The intersecting lines will eventually intersect the sweep line.

No false positives

Assume no vertical lines, no 3 lines intersect at same point

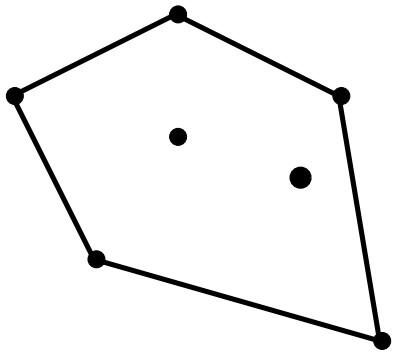


Sweep line: RB tree contains 1
Sweep line: RB tree contains 2, 1, no intersect
Sweep line: RB tree contains 2, 1, 3, no intersect
Sweep line: RB tree contains 2, 3, intersect
Sweep line: RB tree contains 2, 4, 3, no intersect
Sweep line: RB tree contains 2, 4, 3, 5, no intersect
Sweep line: RB tree contains 2, 4, 3, no intersect
Sweep line: RB tree contains 2, 3, intersect
Sweep line: RB tree is empty

```
For n segments:
   sort segments                    n lg n
   n RB-Inserts                     n lg n
   n RB-Deletes                     n lg n
   O(1) comparison
```

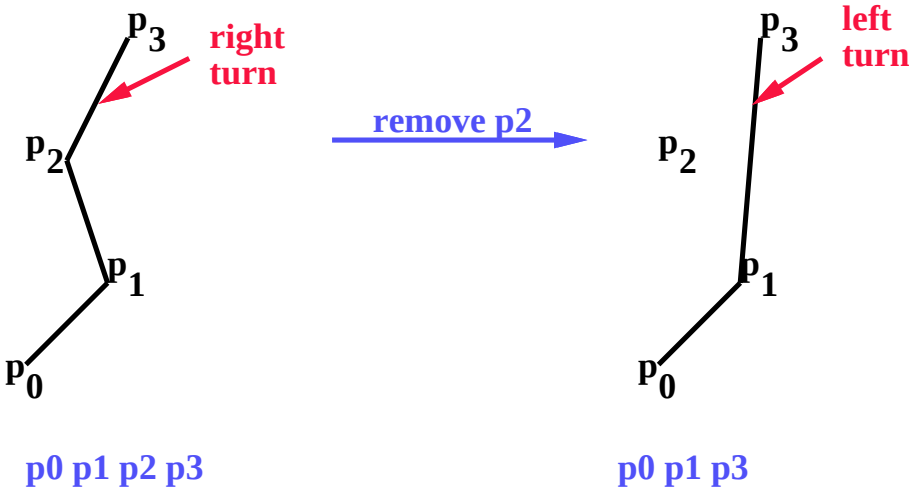
The algorithm takes _____ time.

Convex Hull

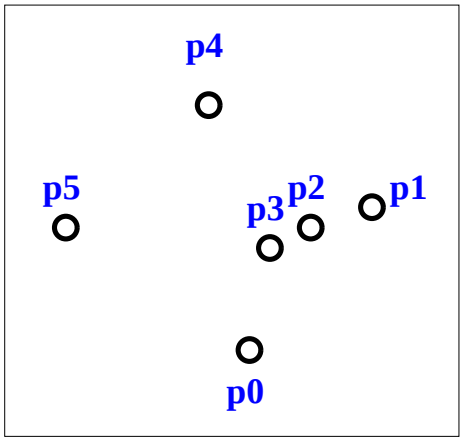


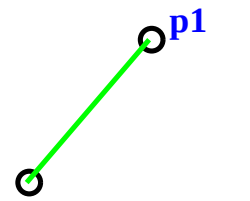
Graham's Scan

- Start with lowest point p_0 ($\Theta(n)$)
- Sort remaining points based on polar angle from p_0 ($\Theta(n \lg n)$)
- Build $p_0,\ p_1,\ p_2,\ ..,\ p_k$ as long as $p_{k-1}\ p_k\ p_{k+1}$ makes a left turn (p_k is on the top of the stack)
- If not, remove p_k's until it does

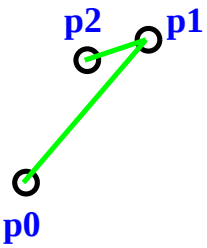


Example

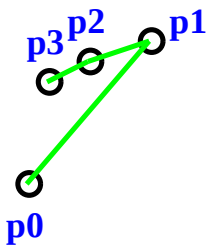




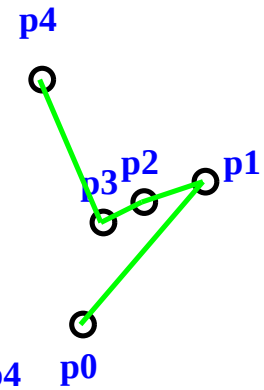
p0 p1



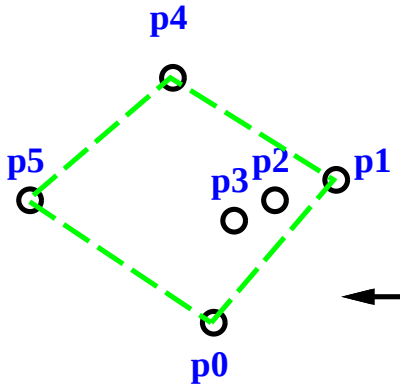
p0 p1 p2



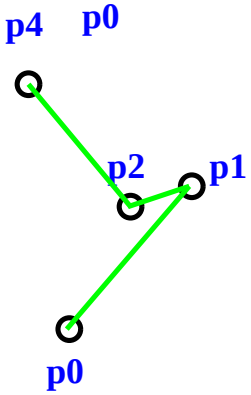
p0 p1 p2 p3



p0 p1 p2 p3 p4



p0 p1 p4 p5



p0 p1 p2 p4

Analysis

Stack Implementation

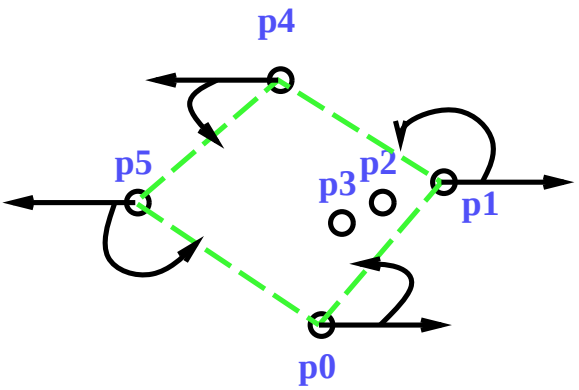
Sort: O(n lg n)
Push/Pop Naive: O(n^2)
Push/Pop Aggregate: _____

Graham's Scan algorithm takes O(n lg n) time.

Jarvis' March

Jarvis-March(P) ; P = $\langle p_0, .., p_{n-1} \rangle$ points
    $p = p_0$, the lowest point
    while p not highest point
        find point p_m with minimum polar angle from $p \longrightarrow$
        add p_m to convex hull
        $p = p_m$
    while $p \neq p_0$
        find point p_m with minimum polar angle from $\longleftarrow p$
        add p_m to convex hull
        $p = p_m$

Example



Analysis

Both while loops total of h times, where h is the number of points on the convex hull.

Body of while loops takes _____ to compute minimums.

Jarvis March algorithm takes _____ time.

Closest Pair Of Points

Brute force solution: n^2

Better solution:

- Given points P
- Sort points by x coordinate into X
- Sort points by y coordinate into Y
- Execute Closest-Pair(P,X,Y)

Closest-Pair(P, X, Y)

if $|P| \leq 3$

then compute closest pair and return ; O(1)

divide points evenly along x axis at x = l into $P_L(X_L, Y_L)$ and $P_R(X_R, X_R)$

d_L = distance(Closest-Pair(P_L,X_L,Y_L))

d_R = distance(Closest-Pair(P_R,X_R,Y_R))

d = min(d_L,d_R)

foreach point p in (l - d) $\leq$ x $\leq$ (l + d)

check 7 points p' closest to p by y-coordinate

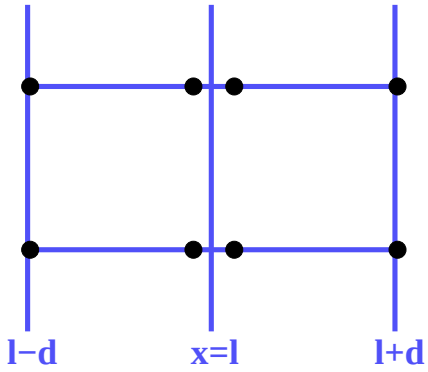
d' = distance(p,p')

if d' < d

then retain new closest pair

return closest pair

Why only 7 points?



Only 8 points define the rectangle whose pairwise distance is d.

For each point we need to check the 7 others in sorted order by y coordinate value.

If a closer d exists then it will be to one of these.

Analysis

Initial Sorts: O(n lg n)

Closest Pair:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq 3 \\ 2T(n/2) + \Theta(n) & \text{if } n > 3 \end{cases}$$

a = 2, b = 2, f(n) =$\Theta(n) = \Theta(n^{\lg 2})$, case 2

T(n) = $\Theta(n \lg n)$

Closest-Pair algorithm takes $\Theta(nlgn)$ time.

Applications