Computational Geometry

Line Segment Properties
Convex Hull

Computational Geometry

CHAPTER 33

Computational Geometry

- Is the branch of computer science that studies algorithms for solving geometric problems.
- Has applications in many fields, including
  - computer graphics
  - robotics,
  - VLSI design
  - computer aided design
  - statistics
- Deals with geometric objects such as points, line segments, polygons, etc.
- Some typical problems considered:
  - whether intersections occur in a set of lines.
  - finding vertices of a convex hull for points.
  - whether a line can be drawn separating two sets of points.
  - whether one point is visible from a second point, given some polygons that may block visibility.
  - optimal location of fire towers to view a region.
  - closest or most distant pair of points.
  - whether a point is inside or outside a polygon.

Computational Geometry

Cross products

- ***Line segments***
  - The convex combination of two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is any point $p_3 = (x_3, y_3)$ such that for some real number α with $0 \le \alpha \le 1$,

$$(x_3, y_3) = \alpha(x_1, y_1) + (1-\alpha)(x_2, y_2).$$

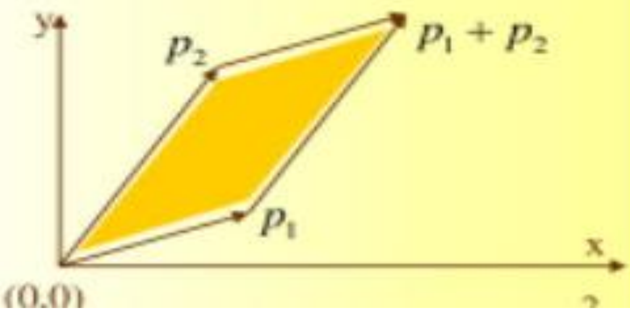
  - $\overline{p_1 p_2}$, the line segment joining p_1 and p_2, is the set of all convex combinations of p_1 and p_2.
  - *Intuition problem*: Show that if (x,y) is a convex combination of (x_1, y_1) and (x_2, y_2) then

$$\alpha = \frac{y - y_2}{x - x_2} = \frac{y_1 - y_2}{x_1 - x_2}$$

which is the standard equation of a line with slope α.

- ***Cross products***
  - let p_1 and p_2 be points on the plane
  - The cross product $p_1 \times p_2$ corresponds to the signed area in the parallelogram.

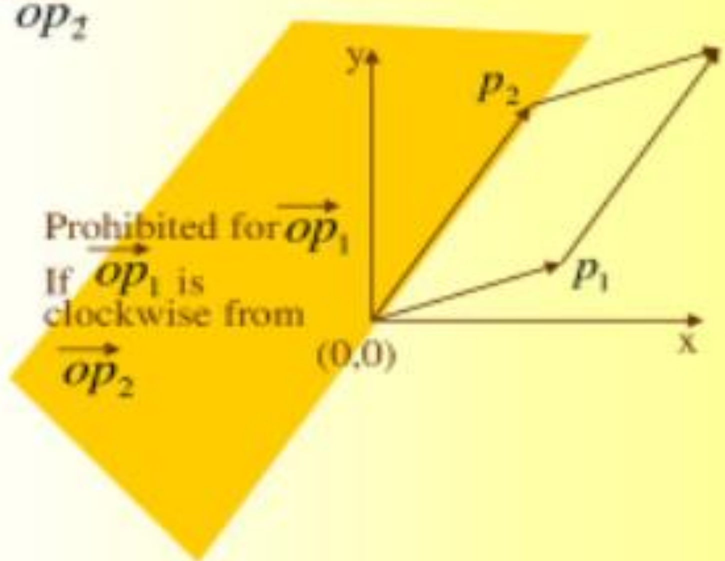
$$p_1 \times p_2 = \det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1 y_2 - x_2 y_1 = -p_2 \times p_1.$$



Computational Geometry

Cross products (cont.)

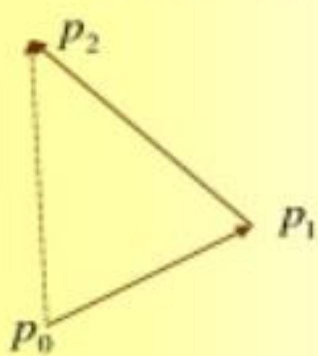
- If $p_1 \times p_2$ is negative, then $\overrightarrow{op_1}$ is counterclockwise from $\overrightarrow{op_2}$.
- If $p_1 \times p_2$ is positive, then $\overrightarrow{op_1}$ is clockwise from $\overrightarrow{op_2}$
- If $p_1 \times p_2 = 0$, then $\overrightarrow{op_1}$ and $\overrightarrow{op_2}$ are collinear.



- To determine if $\overrightarrow{p_0p_1}$ is clockwise from $\overrightarrow{p_0p_2}$, we translate p_0 to the origin and consider $p'_1 \times p'_2$ where $p'_1 = p_1 - p_0$, $p'_2 = p_2 - p_0$.

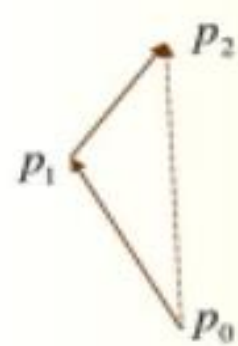
$$p'_1 \times p'_2 = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0).$$

- Consider now whether two consecutive line segments $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_1p_2}$ turn ***left*** or ***right*** at p_1 .
  - Check whether $\overrightarrow{p_0p_2}$ is clockwise from $\overrightarrow{p_0p_1}$



$(p_2 - p_0) \times (p_1 - p_0) < 0$

So, counterclockwise or ***left turn***



$(p_2 - p_0) \times (p_1 - p_0) > 0$

So, clockwise or ***right turn***

Determining whether two line segments intersect

- To determine whether two line segments intersect, we check whether each segment straddles the line containing the other.
- A segment p1p2 ***straddles*** a line if point p1 lies on one side of the line and point p2 lies on the other side.
- A boundary case arises if p1 or p2 lies directly on the line.
- Two line segments intersect if and only if either (or both) of the following conditions holds:

1. Each segment straddles the line containing the other.
2. An endpoint of one segment lies on the other segment. (This condition comes from the boundary case.)

Determining whether two line segments intersect

The following procedures implement this idea. SEGMENTS-INTERSECT returns TRUE if segments $\overline{p_1 p_2}$ and $\overline{p_3 p_4}$ intersect and FALSE if they do not. It calls the subroutines DIRECTION, which computes relative orientations using the cross-product method above, and ON-SEGMENT, which determines whether a point known to be colinear with a segment lies on that segment.

```
SEGMENTS-INTERSECT(p1, p2, p3, p4)
  d1 = DIRECTION(p3, p4, p1)
  d2 = DIRECTION(p3, p4, p2)
  d3 = DIRECTION(p1, p2, p3)
  d4 = DIRECTION(p1, p2, p4)
  if ((d1 > 0 and d2 < 0) or (d1 < 0 and d2 > 0)) and
        ((d3 > 0 and d4 < 0) or (d3 < 0 and d4 > 0))
        return TRUE
  elseif d1 == 0 and ON-SEGMENT(p3, p4, p1)
        return TRUE
  elseif d2 == 0 and ON-SEGMENT(p3, p4, p2)
        return TRUE
  elseif d3 == 0 and ON-SEGMENT(p1, p2, p3)
        return TRUE
  elseif d4 == 0 and ON-SEGMENT(p1, p2, p4)
        return TRUE
  else return FALSE
```

```
DIRECTION(pi, pj, pk)
  return (pk − pi) × (pj − pi)
```

```
ON-SEGMENT(pi, pj, pk)
  if min(xi, xj) ≤ xk ≤ max(xi, xj) and min(yi, yj) ≤ yk ≤ max(yi, yj)
        return TRUE
  else return FALSE
```

Determining whether two line segments intersect

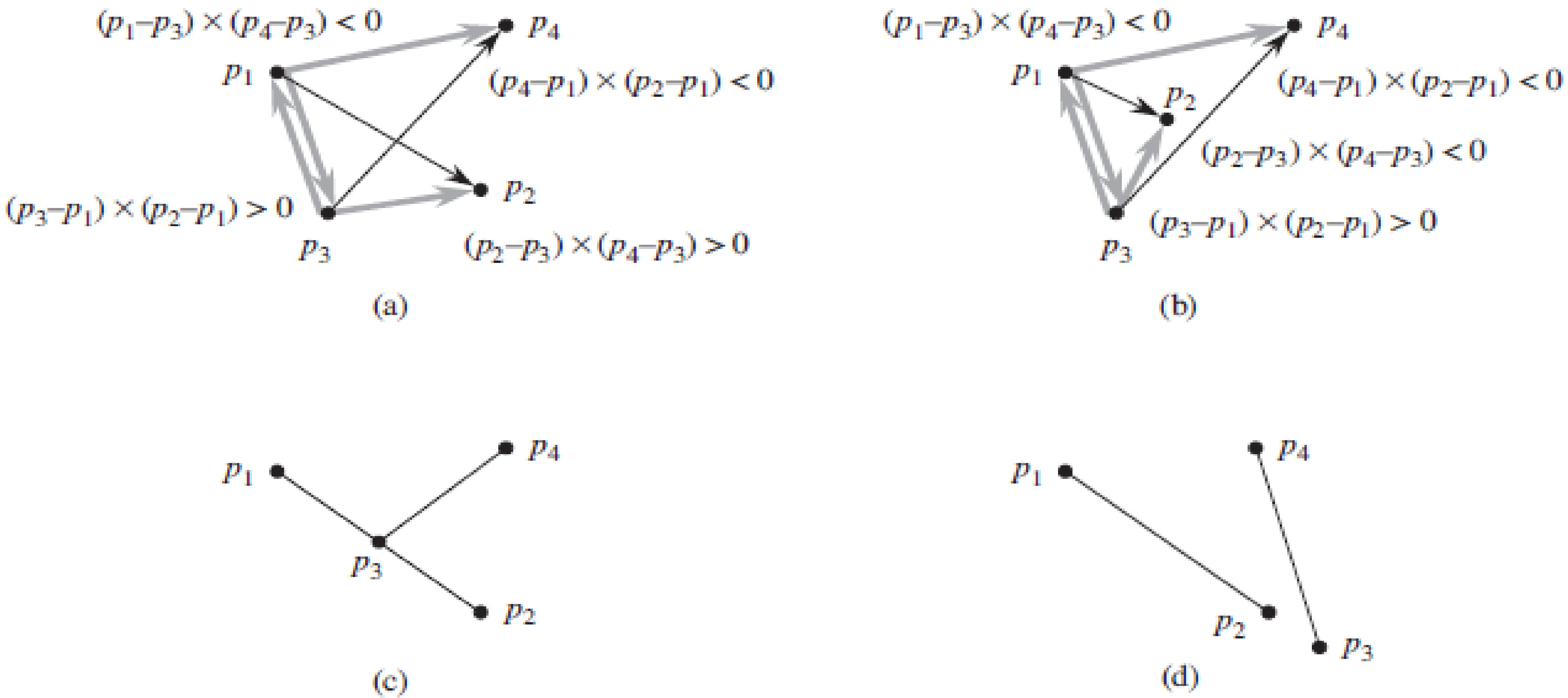


Figure 33.3 Cases in the procedure SEGMENTS-INTERSECT. (a) The segments $\overline{p_1p_2}$ and $\overline{p_3p_4}$ straddle each other's lines. Because $\overline{p_3p_4}$ straddles the line containing $\overline{p_1p_2}$, the signs of the cross products $(p_3-p_1)\times(p_2-p_1)$ and $(p_4-p_1)\times(p_2-p_1)$ differ. Because $\overline{p_1p_2}$ straddles the line containing $\overline{p_3p_4}$, the signs of the cross products $(p_1-p_3)\times(p_4-p_3)$ and $(p_2-p_3)\times(p_4-p_3)$ differ. (b) Segment $\overline{p_3p_4}$ straddles the line containing $\overline{p_1p_2}$, but $\overline{p_1p_2}$ does not straddle the line containing $\overline{p_3p_4}$. The signs of the cross products $(p_1-p_3)\times(p_4-p_3)$ and $(p_2-p_3)\times(p_4-p_3)$ are the same. (c) Point p_3 is colinear with $\overline{p_1p_2}$ and is between p_1 and p_2. (d) Point p_3 is colinear with $\overline{p_1p_2}$, but it is not between p_1 and p_2. The segments do not intersect.

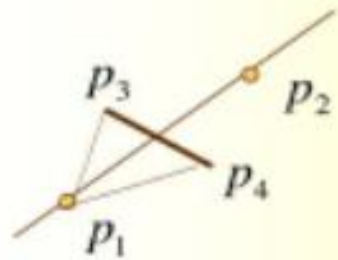
line segments intersection

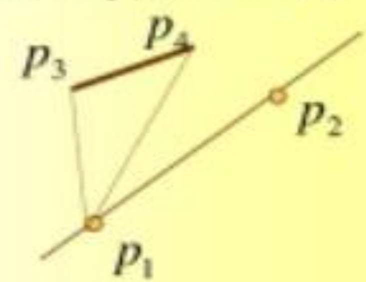
Intersection of two line segments (cont.)

- Second stage: Decide whether each segment meets ("straddles") the line containing the other.
- A segment $\overline{p_1p_2}$ straddles a line if p_1 lies on one side of the line and p_2 on the other side. (the segment straddles the line also if p_1 or p_2 lies on the line)
- ***Observation:*** Two segments intersect iff they pass the quick rejection test and each segment straddles the other.
- Testing straddle with cross products:
  - we show how to check if $\overline{p_3p_4}$ straddles the line L determined by p_1 and p_2

If $\overline{p_3p_4}$ does straddle the line containing p_1 and p_2, then the following have different signs.

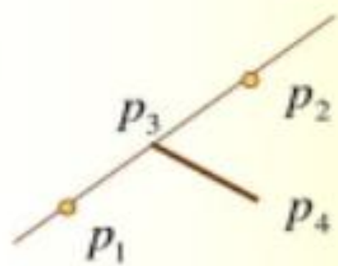
$$(p_3 - p_1)\times(p_2 - p_1)$$
$$(p_4 - p_1)\times(p_2 - p_1)$$

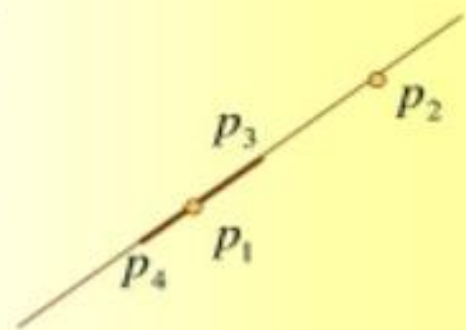




Boundary cases where $\overline{p_3p_4}$ straddles L

At least one cross product is zero. Both cases pass the quick rejection test.

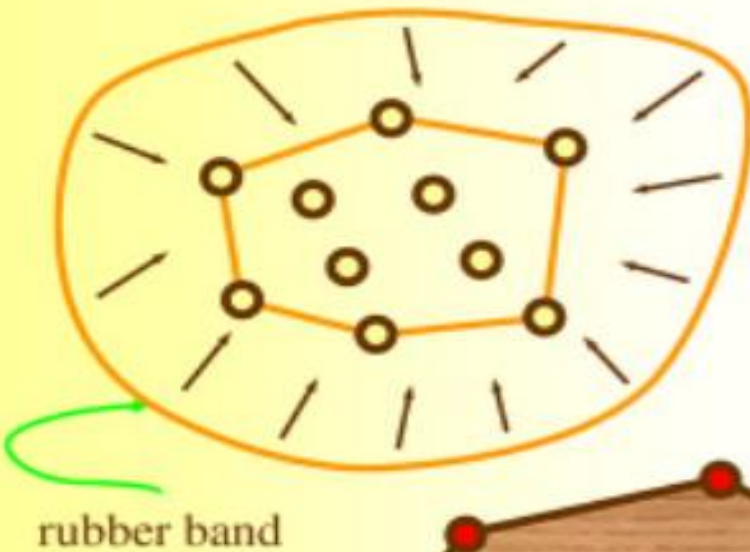




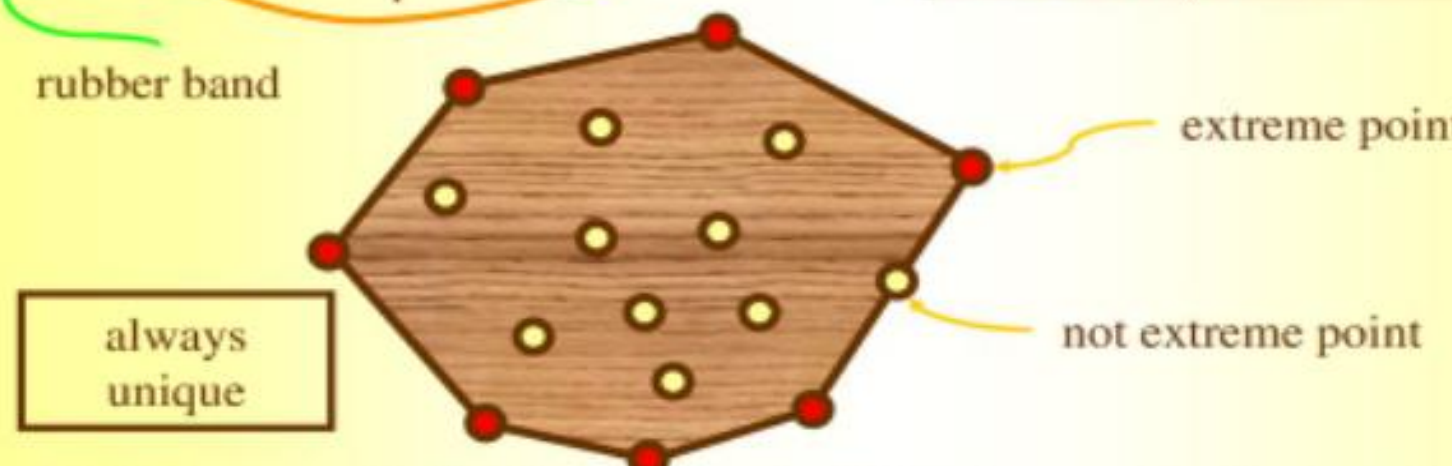
Convex Hull

Convex Hull Algorithms

- Definitions and Properties: Given n points on the plane $Q = \{p_1, p_2, \ldots, p_n\}$.
  - Intersection of all convex sets containing Q
  - Smallest convex set containing Q
  - Intersection of all half-planes containing Q
  - Union of all triangles formed by points of Q
  - **Smallest *convex polygon* containing Q**



– *All vertices of hull are some points of Q*



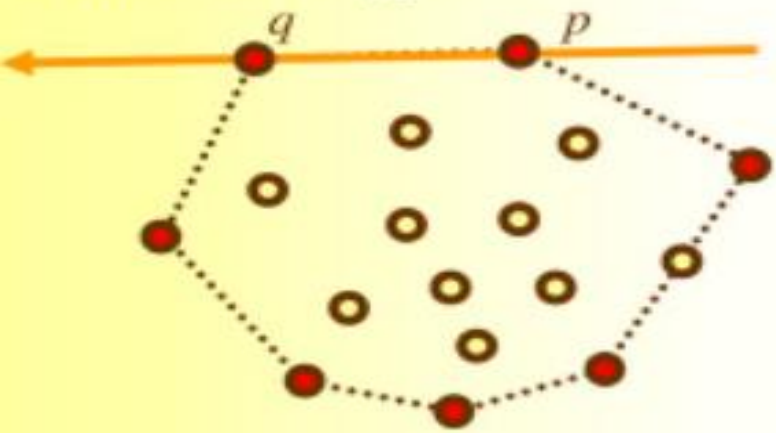
NOTE: *convex-hull(Q)* is the closed solid region, not just the boundary *CH(Q)*

Convex Hull

The Problem and Approaches

- ***Problem:*** Given n points on the plane $Q = \{p_1, p_2, \ldots, p_n\}$, find $CH(Q)$.
- ***Approaches:***
  - Brute Force
  - Gift Wrapping
  - QuickHull
  - Graham Scan
  - Incremental
  - Divide and Conquer
  - By Delaunay Triangulation & Voronoi Diagram

- Brute-Force Approach



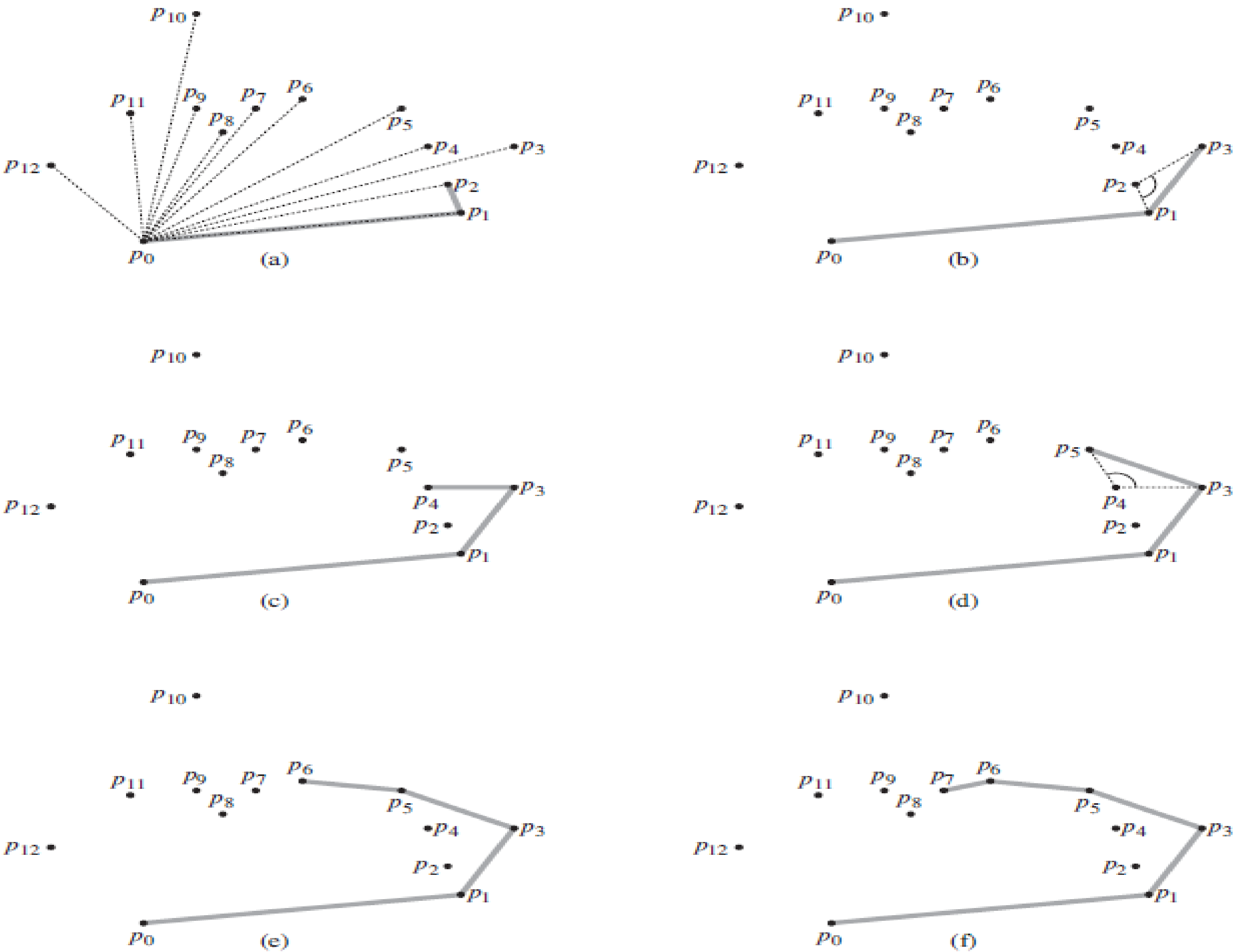
- Determine extreme edges:
  for each pair of points $p,q \in Q$ **do**
  if all other points lie on one side
  of line passing thru p and q
  then keep edge (p, q)
- Sort edges in counterclockwise order
  (we want output in counterclockwise)

- Running time: $O(n^3)$
  – bad but not the worst yet

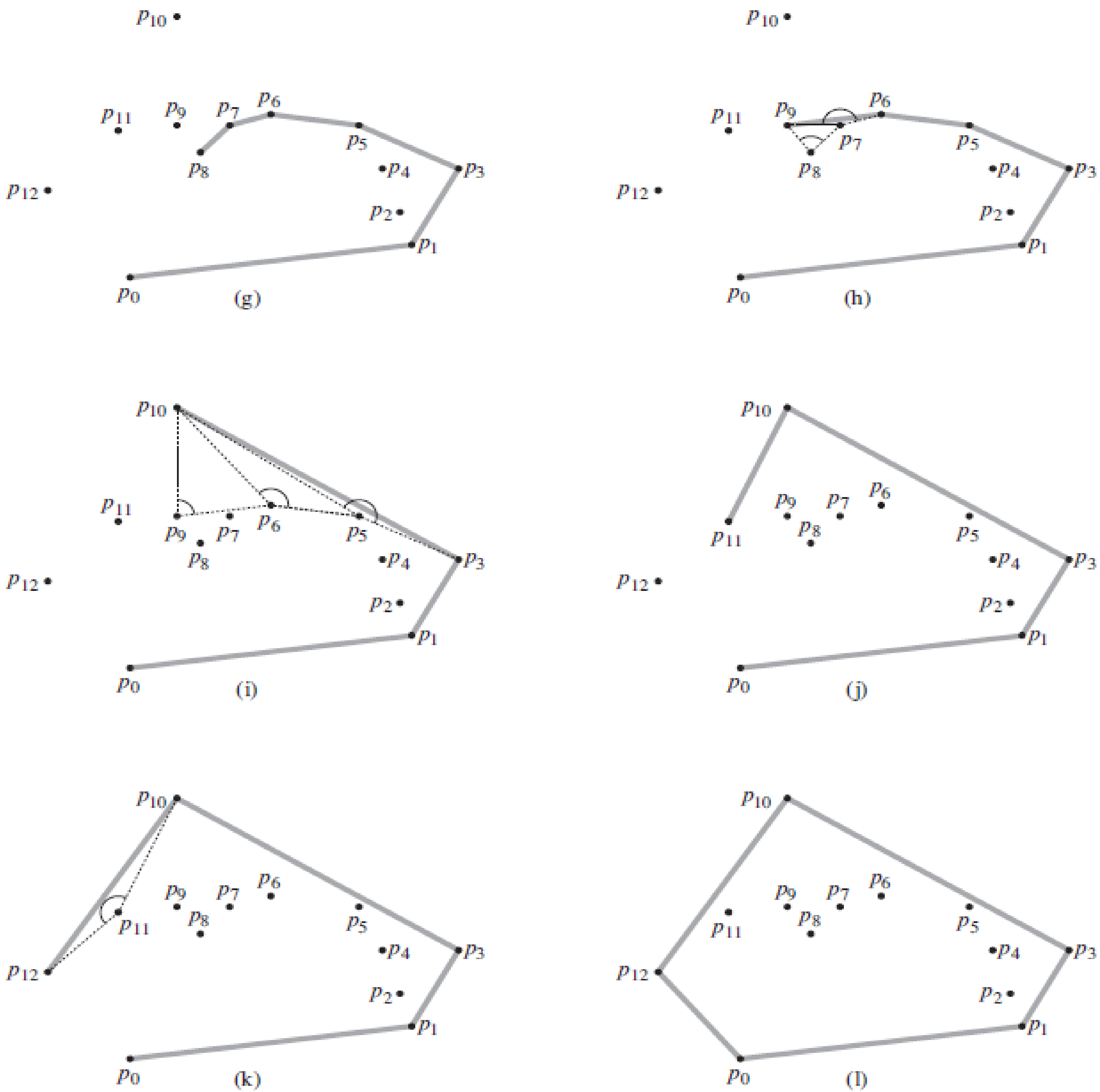
Graham's Scan

```
GRAHAM-SCAN(Q)
let p_0 be the point in Q with the minimum y-coordinate,
      or the leftmost such point in case of a tie
let ⟨p_1, p_2, ..., p_m⟩ be the remaining points in Q,
      sorted by polar angle in counterclockwise order around p_0
      (if more than one point has the same angle, remove all but
      the one that is farthest from p_0)
let S be an empty stack
PUSH(p_0, S)
PUSH(p_1, S)
PUSH(p_2, S)
for i = 3 to m
      while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                  and p_i makes a nonleft turn
            POP(S)
      PUSH(p_i, S)
return S
```

Graham's Scan



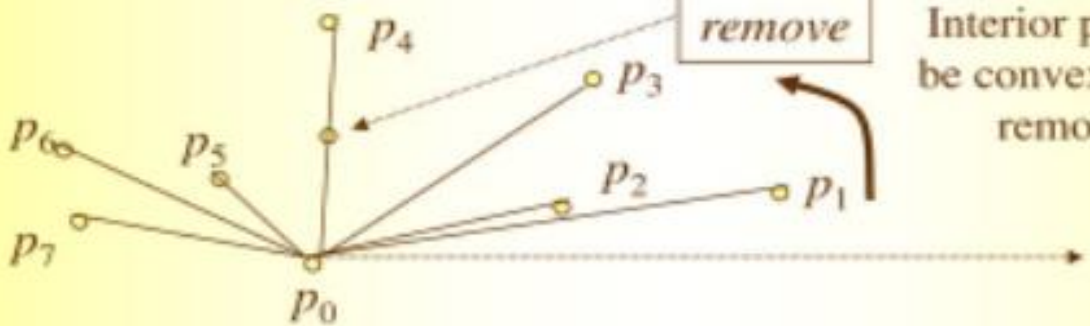
Graham’s Scan



Graham's Scan

Graham Scan Algorithm

- First a base point p_0 is selected. Normally this is the point with minimum y-coordinate (select leftmost in case of tie)
- Next all points are sorted w.r.t. the polar angle they make with a half-ray with left endpoint p_0 and parallel to x-axis. (!!! We do not need to compute those angles!!!)

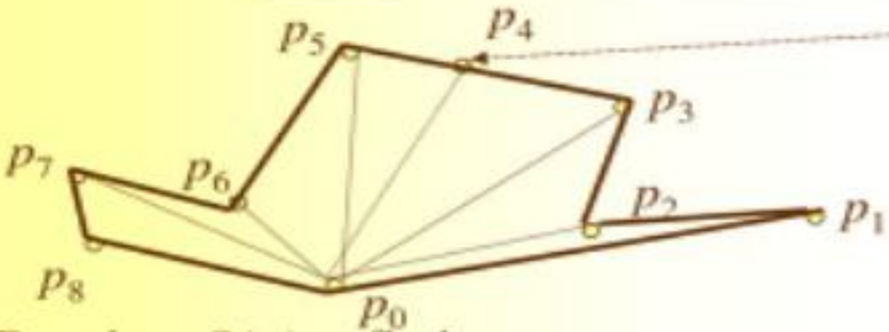


- Remaining points are stored in counterclockwise order w.r.t. p_0
- Let $p_0, p_1, p_2, p_3, \ldots, p_m$ $(m \leq n)$ be the sorted list of remaining points.
- Clearly, p_0 and p_1 are in $CH(Q)$.
- Let S be a stack in which points that are potentially convex hull points will be stored.
- Initially S = $\boxed{p_0 \mid p_1 \mid p_2}$. Remaining steps of the algorithm follow

```
for i ← 3 to m
    do while (the angle formed by points NEXT_TO_TOP(S) , TOP(S), and p_i
              make a non-left turn)  POP(S)
    PUSH(S, p_i)
return S
```

Graham’s Scan

Example and Runtime Computation



Note straight line here.
p_4 is eliminated, as non-left turn.

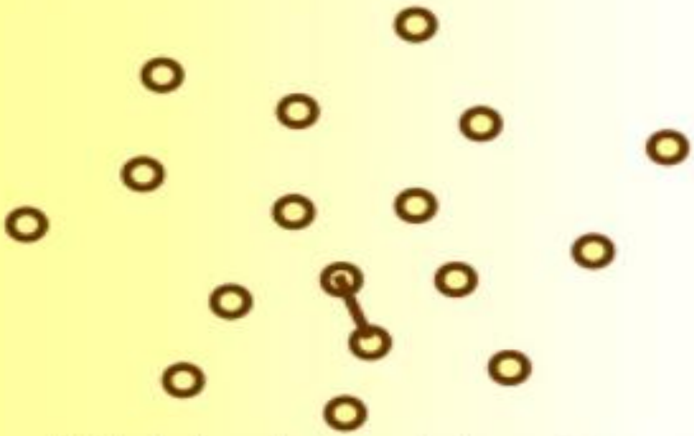
$CH(Q)=\{p_0, p_1, p_3, p_5, p_7, p_8\}$

- Requires $O(n)$ to find p_0
- Sorting based on polar angle takes $O(n \log n)$ time
- Removal of n-m points with duplicate angles takes $O(n)$.
- **For** loop is executed m-2 times, hence $O(n)$.
- Interior **while** statement is a “problem”. It may iterate as many as $O(n)$ time.
- Above observation can easily lead to an over-estimate of $O(n^2)$.
- Note that each pass through **while** statement, POP is executed.
- As in analysis of MULTIPOP, there is at most one POP operation for each PUSH operation (see amortized analysis)
- Since p_0, p_1, p_m are not popped, at most m-3 POP operations occur.
- Note, both POP and test for **while** take $O(1)$ time and $m \le n$. Hence amortized cost for each iteration of **while** loop is $O(1)$.
- Overall ***worst-case*** cost of **for**-loop is $O(n)$
- ***Worst-case*** running time of the algorithm is $O(n \log n)$ (= $O(n \log n) + O(n)$).

Closest Pair of Points

Closest Pair of Points

- Given n points on the plane, find closest pair of points.



  - The Euclidean distance between two points $p_1=(x_1,y_1)$ and $p_2=(x_2,y_2)$ is
    $$d(p_1,p_2)=\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$
  - An obvious but naïve approach is to compute the distance between any two points and take minimum. However, running time is $\binom{n}{2}=O(n^2)$.

- A high-level description of a much better algorithm (at least for large sets) is given below.
- Let Q be a set of n planar points.
- If $|Q|<4$, then the distances between all pairs of points are computed and the closest pair is reported.
- If $|Q|>3$, then a "Divide & Conquer & Combine" procedure is followed.
- Each recursive call receives as input
  - a set $P \subseteq Q$
  - arrays X and Y containing points P sorted by x and y coordinates, respectively