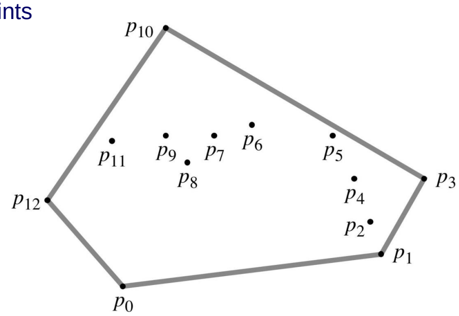
Unit 8: Computational Geometry

- Course contents:
  - Line-segment properties
  - Segment intersection
  - Convex hull
  - Closest pair of points
- Readings:
  - Chapter 33



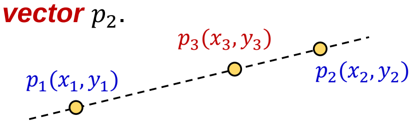
Computational Geometry

- Computational geometry studies algorithms for solving geometric problems:
  - Computer graphics, robotics, VLSI design, computer-aided design, molecular modeling, etc.
- The input is typically a description of a set of geometric objects:
  - A set of points, a set of line segments, the vertices of a polygon in counterclockwise order, etc.
- The output is often a response to a query about the objects:
  - Whether any of the lines intersect?
  - Maybe a new geometric object, ex: the convex hull of the set of points.

Line-Segment Properties

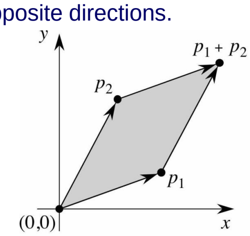
Terminologies

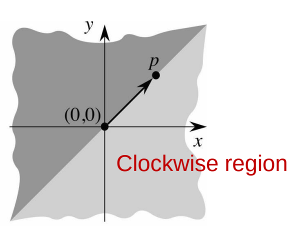
- Any point $p_3(x_3, y_3)$ is a ***convex combination*** of two distinct points $p_1(x_1, y_1)$ and $p_2(x_2, y_2)$ if:
  - $x_3 = \alpha x_1 + (1-\alpha)x_2$ and $y_3 = \alpha y_1 + (1-\alpha)y_2, 0 \le \alpha \le 1$.
  - $p_3 = \alpha p_1 + (1-\alpha)p_2, 0 \le \alpha \le 1$.
  - p_3 is any point on the line passing through p_1 and p_2 and is on or between p_1 and p_2
- The ***line segment*** $\overline{p_1p_2}$ is the set of convex combinations of p_1 and p_2, where p_1 and p_2 are ***endpoints***.
- ***Direct segment*** $\overrightarrow{p_1p_2}$: if p_1 is the origin $(0,0)$, $\overrightarrow{p_1p_2}$ is treated as the ***vector*** p_2.



Cross Products

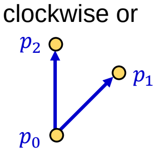
- Given two vectors p_1 and p_2, the ***cross product*** $\boldsymbol{p_1} \times \boldsymbol{p_2}$:
  - The signed area of the parallelogram formed by the points $(0,0)$, p_1, p_2, and $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$.
  - The determinant of a matrix
    $$p_1 \times p_2 = \det\begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} = x_1y_2 - x_2y_1 = -p_2 \times p_1$$
  - If $p_1 \times p_2$ is positive/negative, p_1 is clockwise/counterclockwise from p_2 with respect to the origin $(0,0)$.
  - If $p_1 \times p_2 = 0$, the vectors are ***colinear*** and point in the same or opposite directions.



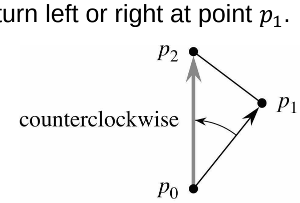


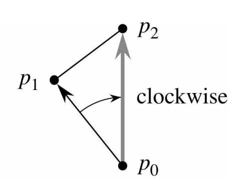
Orientation of Two Directed Segments

- To determine whether $\overrightarrow{p_0p_1}$ is closer to $\overrightarrow{p_0p_2}$ in a clockwise or counterclockwise direction w.r.t. p_0
  - Translate p_0 as the origin.
    - $p_1' = (x_1', y_1') = (x_1 - x_0, y_1 - y_0) = p_1 - p_0$
    - $p_2' = (x_2', y_2') = (x_2 - x_0, y_2 - y_0) = p_2 - p_0$
  - If $p_1' \times p_2' > 0$, $\overrightarrow{p_0p_1}$ is clockwise from $\overrightarrow{p_0p_2}$; otherwise, it is counterclockwise.



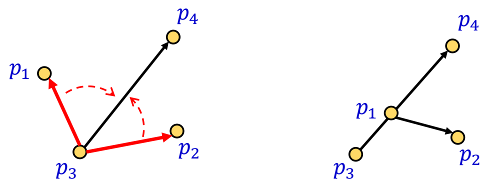
- Determining whether two consecutive segments $\overline{p_0p_1}$ and $\overline{p_1p_2}$ turn left or right at point p_1.





Intersection between Two Segments

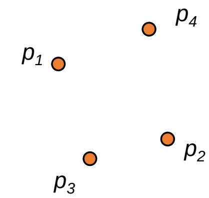
- A segment $\overline{p_1p_2}$ ***straddles*** a line if p_1 lies on one side of the line and p_2 lies on the other side.
- Two line segments intersects ⟺ either the following conditions holds
  1. Each segment straddles the line containing the other.
  2. An endpoint of one segment lies on the other segment.



$(\overrightarrow{p_3p_1} \times \overrightarrow{p_3p_4})(\overrightarrow{p_3p_2} \times \overrightarrow{p_3p_4}) < 0$ $\quad$ $\overrightarrow{p_3p_1} \times \overrightarrow{p_3p_4} = 0$

Segments-Intersect

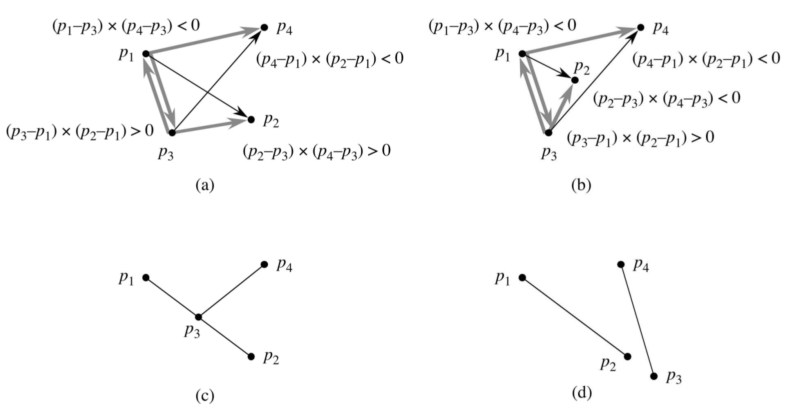
```
Segments-Intersect(p1, p2, p3, p4)
  d1 = Direction(p3, p4, p1)
  d2 = Direction(p3, p4, p2)
  d3 = Direction(p1, p2, p3)
  d4 = Direction(p1, p2, p4)
  if ((d1 > 0 and d2 < 0) or (d1 < 0 and d2 > 0)) and
     ((d3 > 0 and d4 < 0) or (d3 < 0 and d4 > 0))
      return TRUE
  elseif d1 == 0 and On-Segment(p3, 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) x (pj - pi)
```

```
On-Segment(pi, pj, pk) // whether a point known to be colinear with a segment lies on that segment.
  if min(xi, xj) ≤ xk ≤max(xi, xj) and
     min(yi, yj) ≤ yk ≤max(yi, yj)
      return TRUE
  else return FALSE
```


Cases in Segments-Intersect



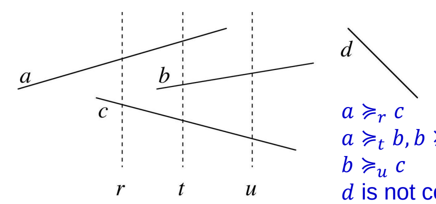
Determining Whether Any Pair of Segments Intersects

Any Segment Intersection?

- To determine whether any two line segments in a set of segments intersect:
  - An intuitive $O(n^2)$ algorithm examining all pair of segments.
  - A more efficient $O(n \lg n)$ algorithm using ***sweeping***, which determines only whether or not any intersection exists but not prints all the intersections (which takes $\Omega(n^2)$).
- In sweeping, an imaginary vertical ***sweep line*** passes through the given set of geometric objects, usually from left to right.
- Assumption
  - No input segment is vertical.
  - No three input segments intersect at a single point.

Ordering Segments

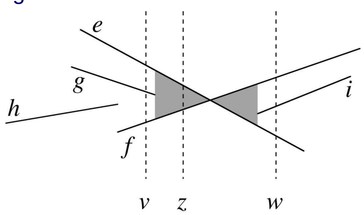
- Segments intersecting a vertical sweep line can be ordered according to the y-coordinate of the points of intersection.
- Two segments s_1 and s_2 are ***comparable*** at x:
  - The vertical sweep line with x-coordinate x intersects s_1 and s_2.
  - $s_1 \succcurlyeq_x s_2$: s_1 is ***above*** s_2, i.e., the intersection of s_1 with the sweep line at x is higher than the intersection of s_2 with the same sweep line, or if s_1 and s_2 intersect at the sweep line.



$a \succcurlyeq_r c$
$a \succcurlyeq_t b, b \succcurlyeq_t c, a \succcurlyeq_t c$
$b \succcurlyeq_u c$
d is not comparable with others

The Ordering of Intersected Segments

- An ordering is kept and differs for differing values of x.
  - A segment enters the ordering when its left endpoint is encountered by the sweep. It leaves the ordering when its right endpoint is encountered.
  - When the sweep line passes through the intersection of two segments, the segments reverse their positions in the ordering.
  - There must be some vertical sweep line for which two intersected segments are consecutive in the ordering.



$e \succcurlyeq_v f$
$f \succcurlyeq_w e$

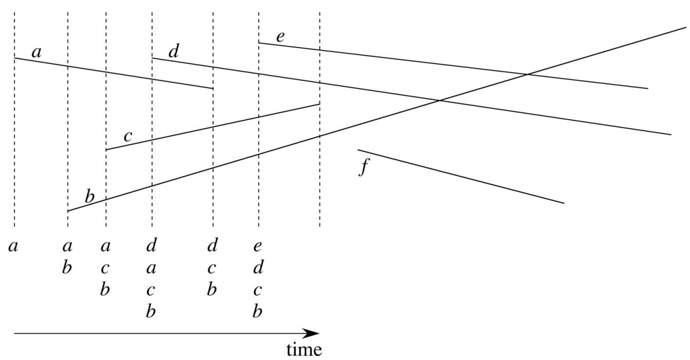
Any sweep line that passes through the shaded region has consecutive e and f in the ordering

Any-Segments-Intersect

```
Any-Segments-Intersect(S)
1.  T = ∅
2.  Sort the endpoints of the segments in S from left to right
    // break ties by putting left endpoints before right
    endpoints and break further ties by putting points with
    lower y-coordinates first
3.  for each point p in the sorted list of endpoints
4.      if p is the left endpoint of a segment s
5.          Insert(T, s)
6.          if (Above(T, s) exists and intersects s) or
              (Below(T, s) exists and intersects s)
7,              return TRUE
8.      if p is the right endpoint of a segment s
9.          if both Above(T, s) and Below(T, s) exist and
              Above(T, s) intersects Below(T, s)
10.             return TRUE
11.         Delete(T, s)
12. return FALSE
```

$O(n \lg n)$ by using RB-tree to implement T

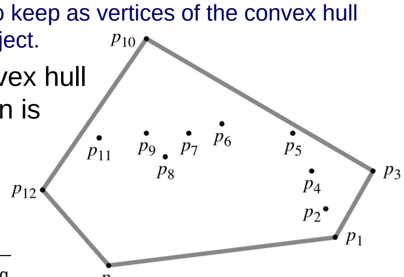
Example: Any-Segments-Intersect



Finding the Convex Hull

Convex Hull

- The ***convex hull*** of a set Q of points, denoted by $CH(Q)$, the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.
  - All points in the set Q are unique.
  - Q contains at least three points which are not colinear.
- Every vertex of $CH(Q)$ is a point in Q.
  - Decide which vertices in Q to keep as vertices of the convex hull and which vertices in Q to reject.
- When we traverse the convex hull counterclockwise, a left turn is made at each vertex



Graham's Scan

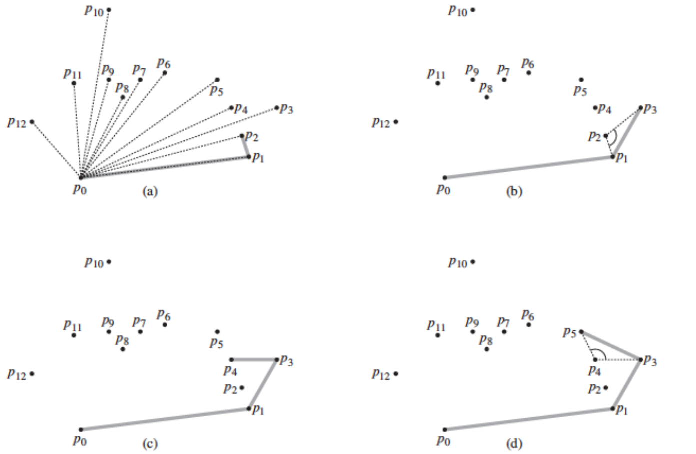
```
Graham-Scan(Q)
1.  Let p0 be the point in Q with the minimum y-coordinate
    // break ties by choosing the leftmost one
2.  let <p1, p2,…, pm> be the remaining points in Q, sorted by
    polar angle in counterclockwise order around p0
    // if more than one point has the same angle, remove all
      but the one that is farthest from p0
3.  Let S be an empty stack
4.  Push(p0, S)
5.  Push(p1, S)
6.  Push(p2, S)
7.  for i = 3 to m
8.      while the angle formed by points Next-To-Top(S),
              Top(S), and pi makes a non-left turn
9.          Pop(S)
10.     Push(pi, S)
11. return S
```

How?

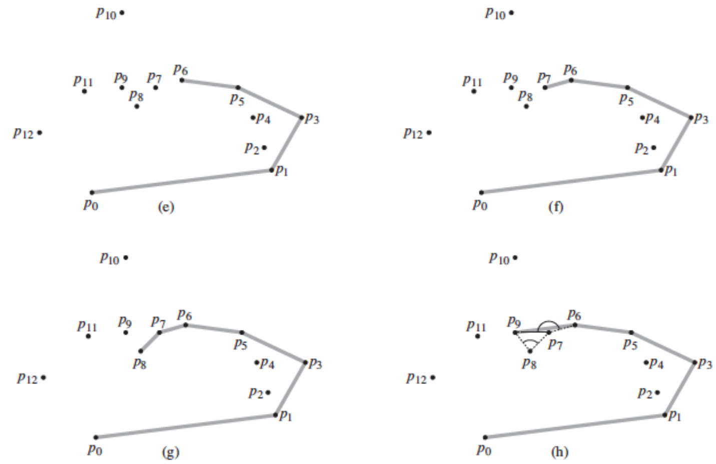
$O(n \lg n)$

- Next-To-Top(S): The point one entry below the top of stack S.

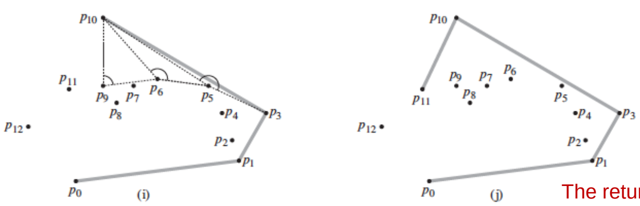
Example: Graham's Scan



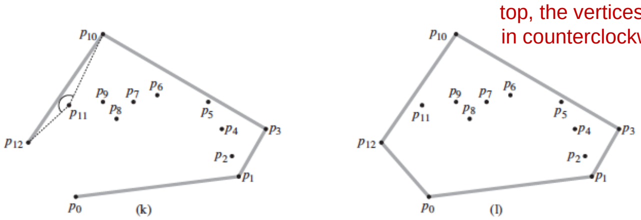
Example: Graham's Scan (cont'd)



Example: Graham's Scan (cont'd)



The returned stack contains, from bottom to top, the vertices of $CH(Q)$ in counterclockwise order



Jarvis's March

- Jarvis's march simulates wrapping a taut piece of paper around the set Q.
  - Start with p_0, the lowest point in the set.
  - The next vertex p_1 in $CH(Q)$ has the smallest polar angle w.r.t. p_0.
  - Find the subsequence vertices until the highest vertex, p_k, is reached, and the ***right chain*** is constructed.
  - To construct the ***left chain***, start at p_k and find p_{k+1} with the smallest polar angle w.r.t. p_k, but from the negative x-axis.
  - Form the left chain until the original vertex p_0 is reached.
- Jarvis's march has a running time of $O(nh)$, where h is #vertices of $CH(Q)$.

Example: Jarvis's March