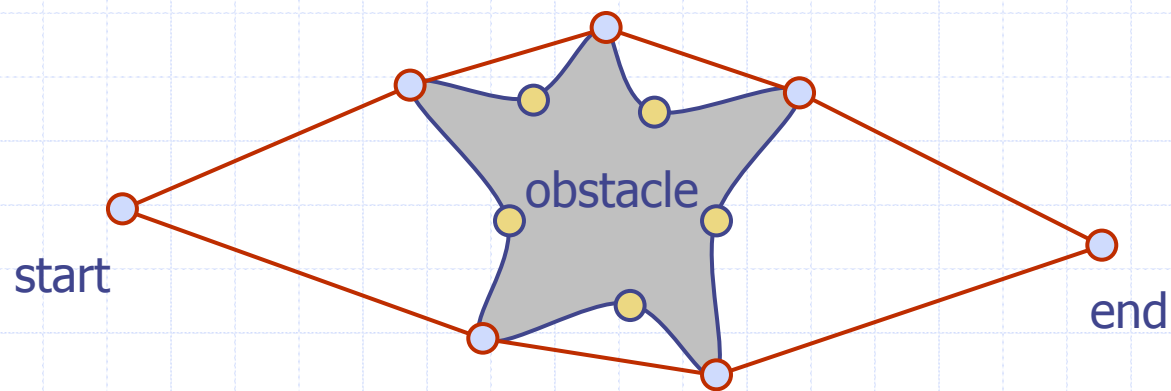


Convex Hull



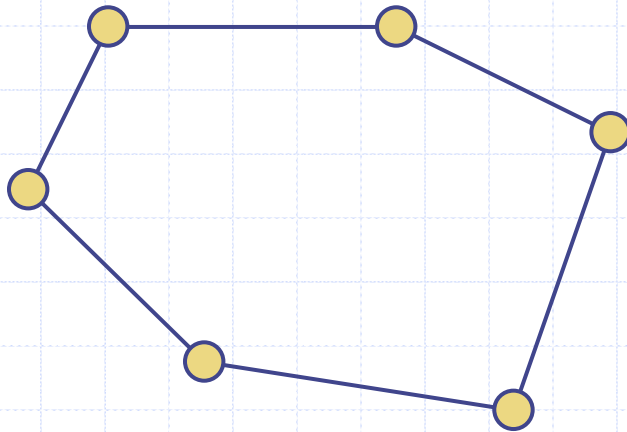
Outline and Reading

- ◆ Convex hull
- ◆ Orientation
- ◆ Sorting by angle
- ◆ Graham scan
- ◆ Analysis
- ◆ Incremental construction

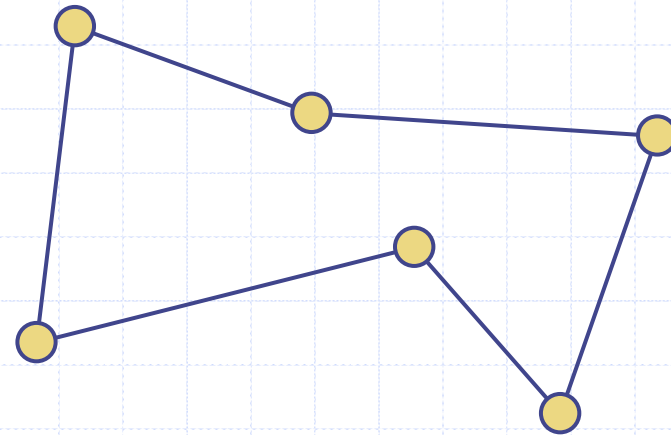
(See Supplement)

Convex Polygon

- ◆ A convex polygon is a nonintersecting polygon whose internal angles are all convex (i.e., less than π)
- ◆ In a convex polygon, a segment joining two vertices of the polygon lies entirely inside the polygon



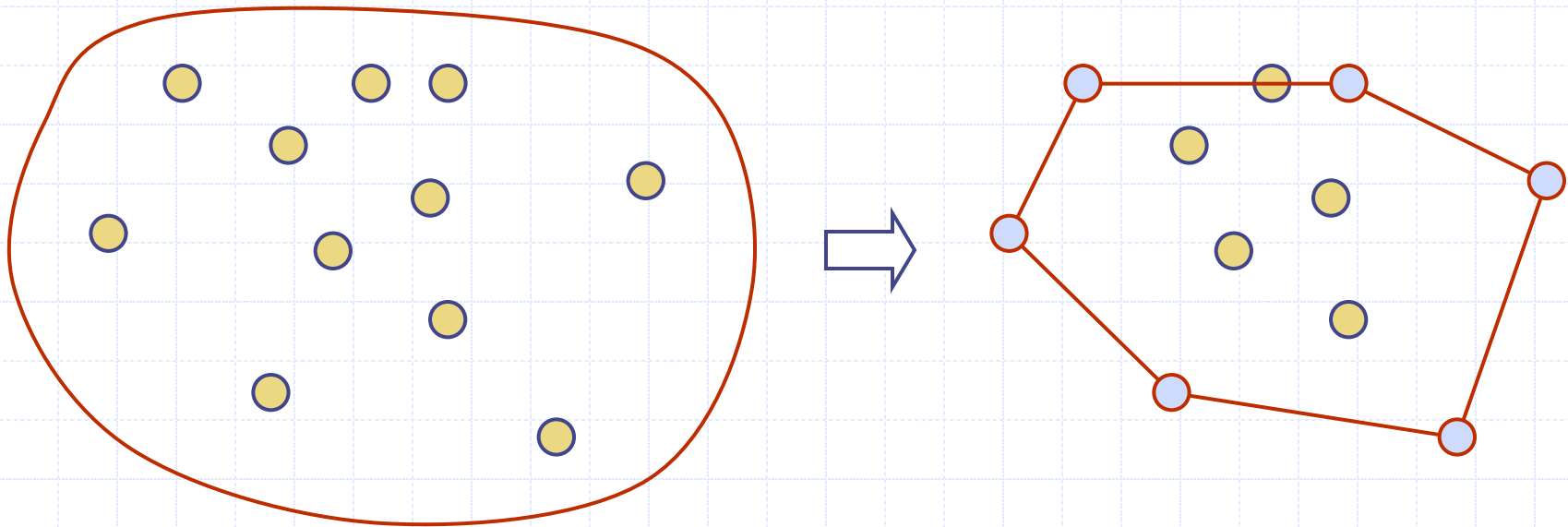
convex



nonconvex

Convex Hull

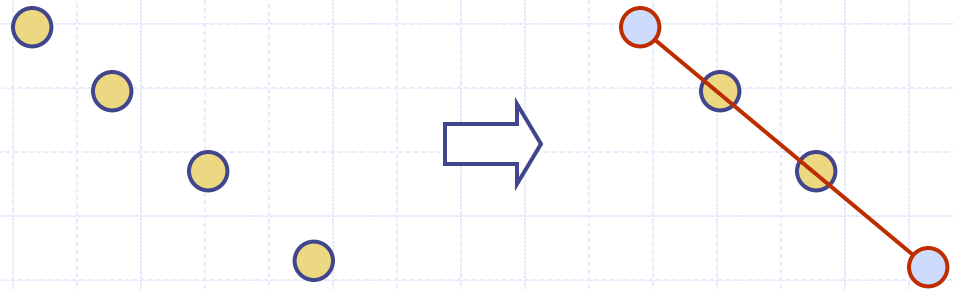
- ◆ The convex hull of a set of points is the smallest convex polygon containing the points
- ◆ Think of a rubber band snapping around the points



Special Cases

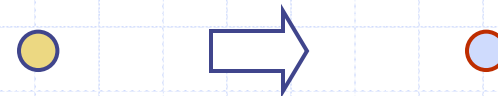
◆ The convex hull is a segment

- Two points
- All the points are collinear



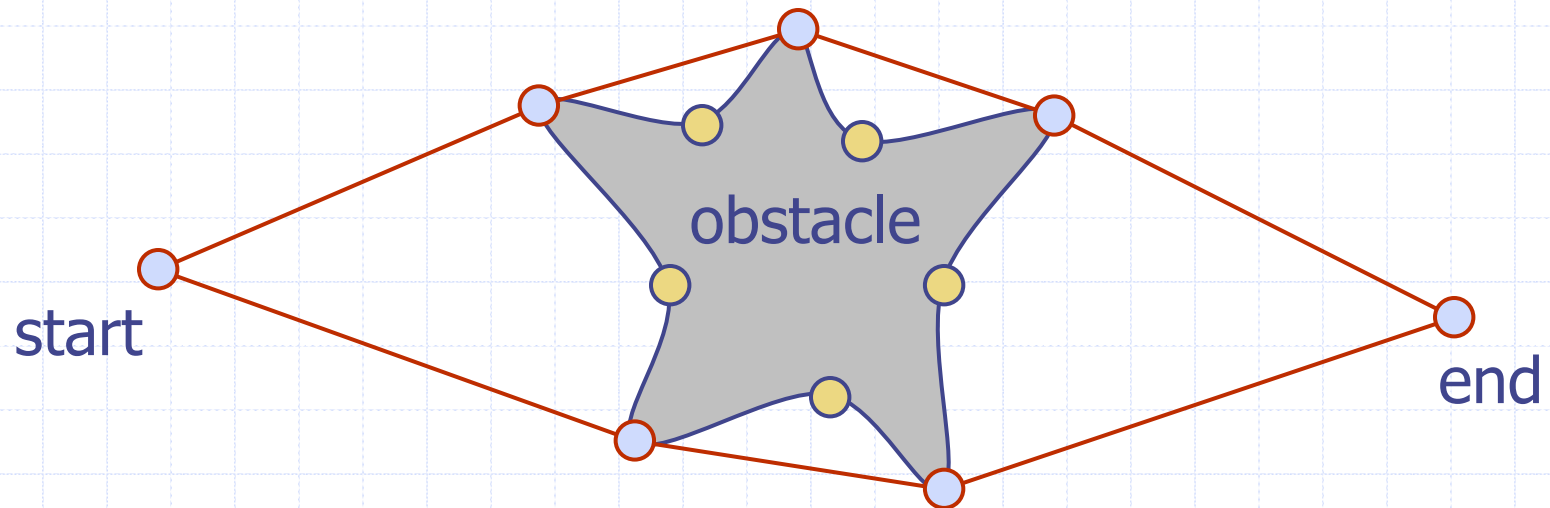
◆ The convex hull is a point

- there is one point
- All the points are coincident



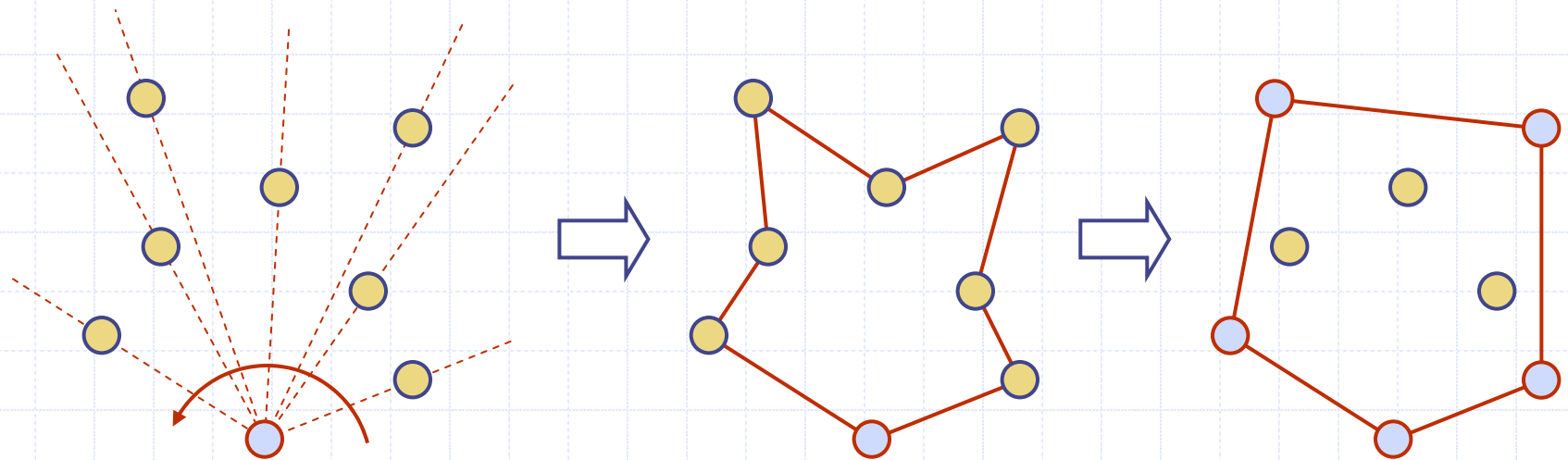
Applications

- ◆ Motion planning
 - Find an optimal route that avoids obstacles for a robot
- ◆ Geometric algorithms
 - Convex hull is like a two-dimensional sorting



Computing the Convex Hull

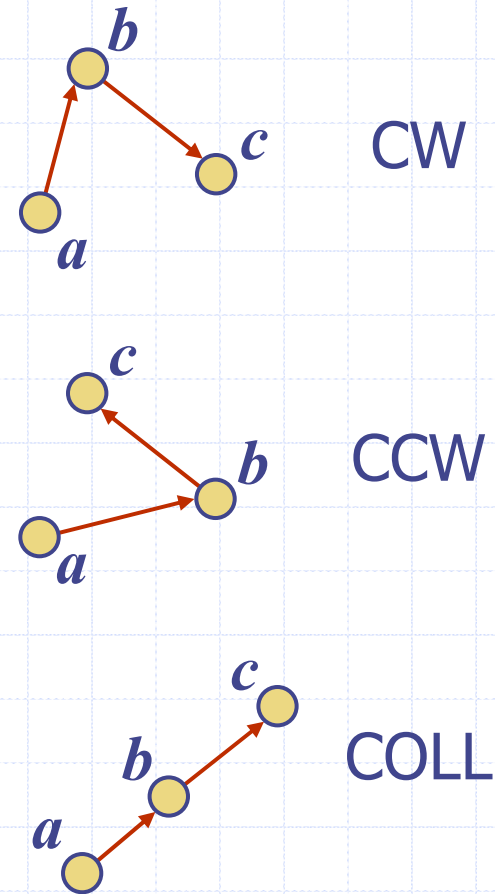
- ◆ The following method computes the convex hull of a set of points
 - Phase 1: Find the lowest point (anchor point)
 - Phase 2: Form a nonintersecting polygon by sorting the points counterclockwise around the anchor point
 - Phase 3: While the polygon has a nonconvex vertex, remove it



Orientation

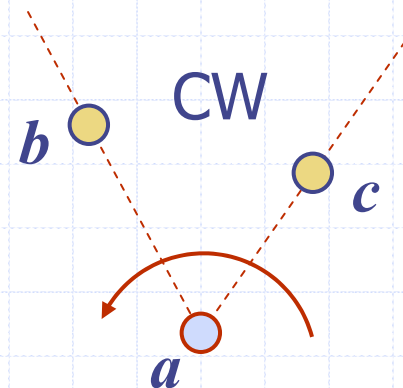
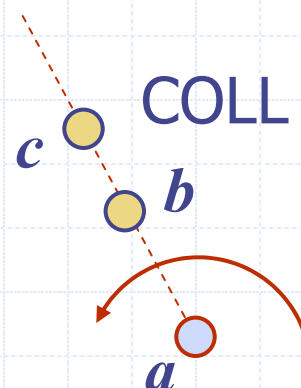
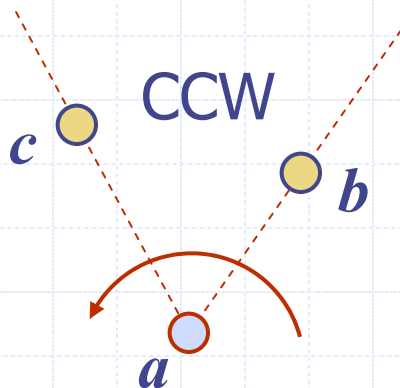
- ◆ The orientation of three points in the plane is clockwise, counterclockwise, or collinear
- ◆ **orientation**(a, b, c)
 - clockwise (CW, right turn)
 - counterclockwise (CCW, left turn)
 - collinear (COLL, no turn)
- ◆ The orientation of three points is characterized by the sign of the determinant $\Delta(a, b, c)$, whose absolute value is twice the area of the triangle with vertices a, b and c

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



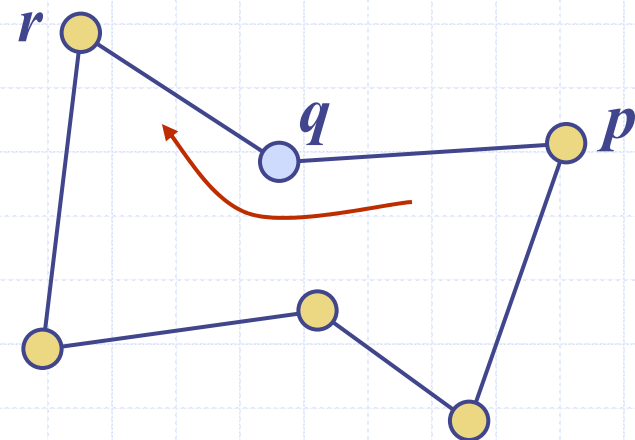
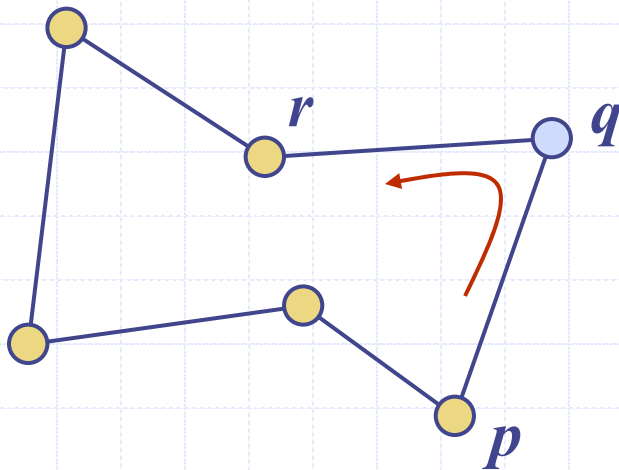
Sorting by Angle

- ◆ Computing angles from coordinates is complex and leads to numerical inaccuracy
- ◆ We can sort a set of points by angle with respect to the anchor point a using a comparator based on the orientation function
 - $b < c \Leftrightarrow \text{orientation}(a, b, c) = \text{CCW}$
 - $b = c \Leftrightarrow \text{orientation}(a, b, c) = \text{COLL}$
 - $b > c \Leftrightarrow \text{orientation}(a, b, c) = \text{CW}$



Removing Nonconvex Vertices

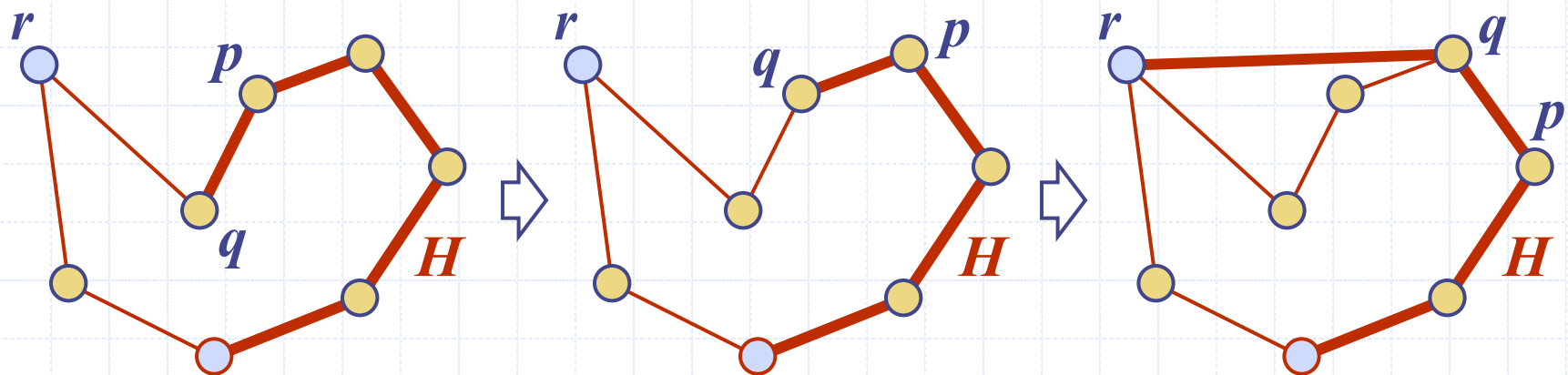
- ◆ Testing whether a vertex is convex can be done using the orientation function
- ◆ Let p , q and r be three consecutive vertices of a polygon, in counterclockwise order
 - q convex $\Leftrightarrow \text{orientation}(p, q, r) = \text{CCW}$
 - q nonconvex $\Leftrightarrow \text{orientation}(p, q, r) = \text{CW}$ or COLL



Graham Scan

- ◆ The Graham scan is a systematic procedure for removing nonconvex vertices from a polygon
- ◆ The polygon is traversed counterclockwise and a sequence H of vertices is maintained

```
for each vertex  $r$  of the polygon
  Let  $q$  and  $p$  be the last and second last
  vertex of  $H$ 
  while orientation( $p, q, r$ ) = CW or COLL
    remove  $q$  from  $H$ 
     $q \leftarrow p$ 
     $p \leftarrow$  vertex preceding  $p$  in  $H$ 
  Add  $r$  to the end of  $H$ 
```

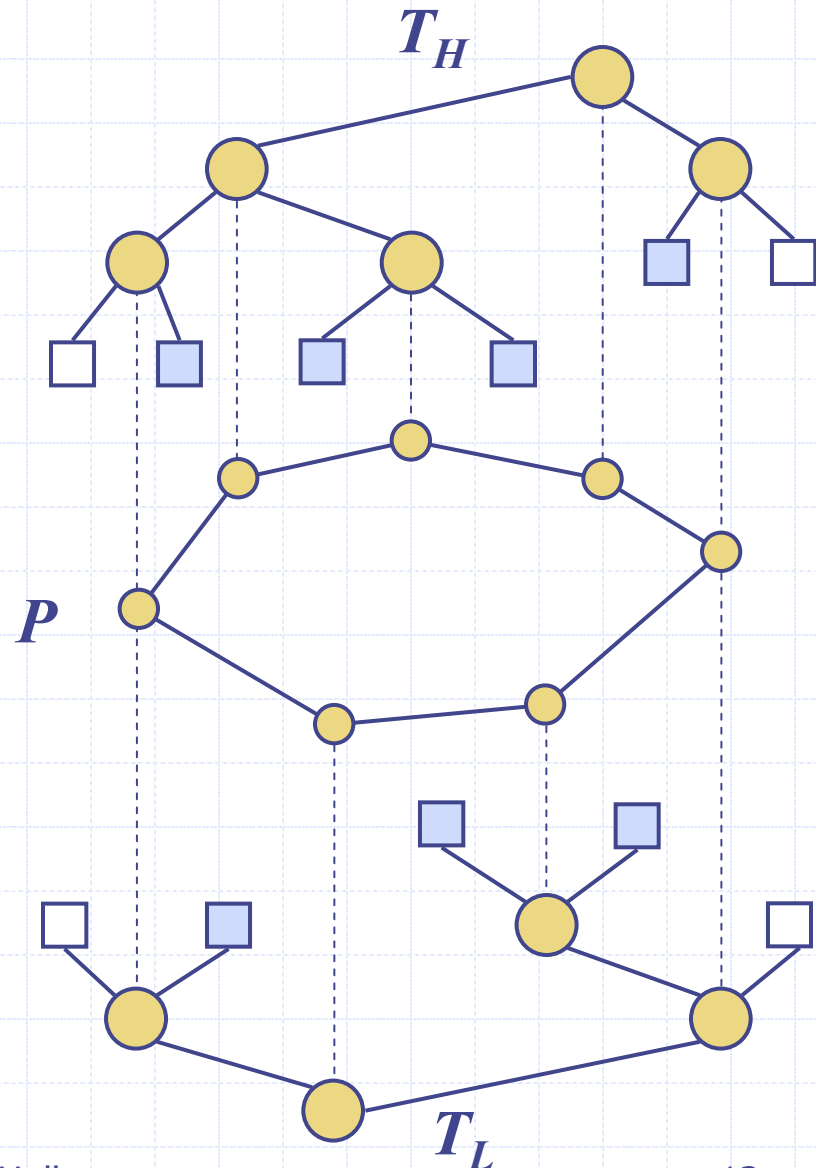


Analysis

- ◆ Computing the convex hull of a set of points takes $O(n \log n)$ time
 - Finding the anchor point takes $O(n)$ time
 - Sorting the points counterclockwise around the anchor point takes $O(n \log n)$ time
 - ◆ Use the orientation comparator and any sorting algorithm that runs in $O(n \log n)$ time (e.g., heap-sort or merge-sort)
 - The Graham scan takes $O(n)$ time
 - ◆ Each point is inserted once in sequence H
 - ◆ Each vertex is removed at most once from sequence H

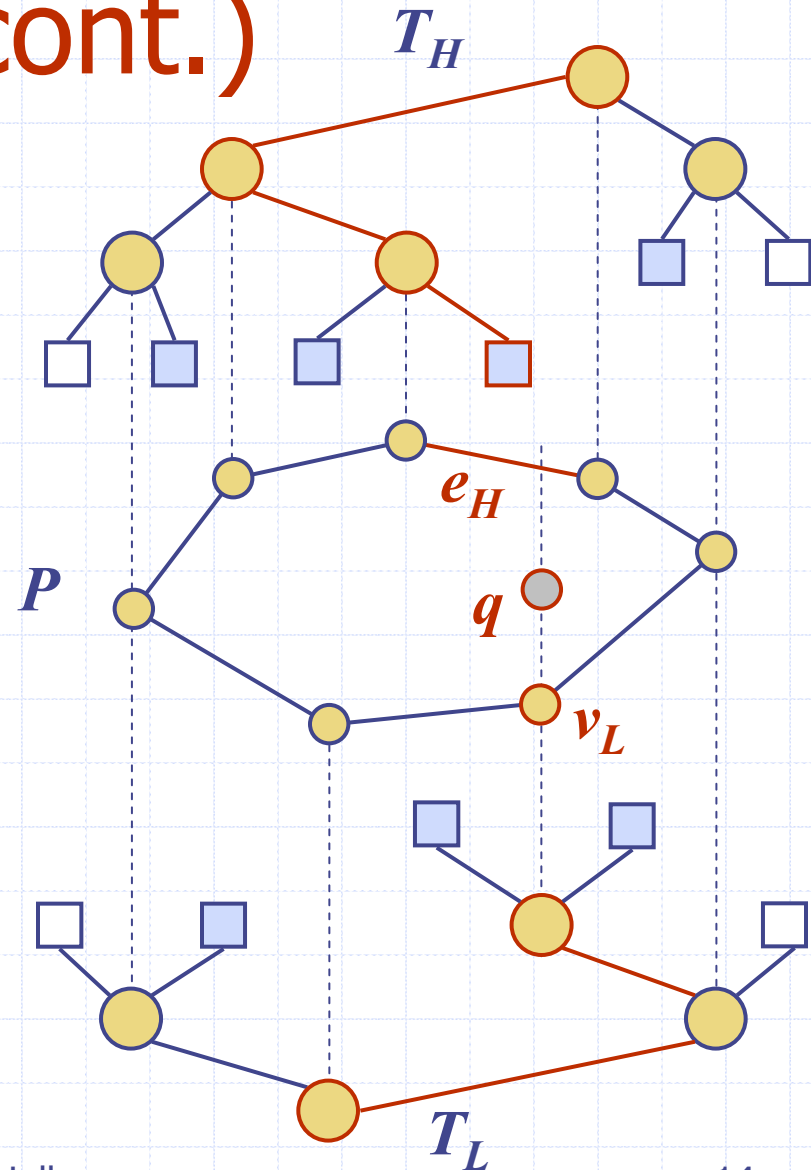
Point Location

- ◆ Given a convex polygon P , a point location query $locate(q)$ determines whether a query point q is inside (IN), outside (OUT), or on the boundary (ON) of P
- ◆ An efficient data structure for point location stores the top and bottom chains of P in two binary search trees, T_L and T_H of logarithmic height
 - An internal node stores a pair $(x(v), v)$ where v is a vertex and $x(v)$ is its x -coordinate
 - An external node represents an edge or an empty half-plane



Point Location (cont.)

- ◆ To perform *locate*(q), we search for $x(q)$ in T_L and T_H to find
 - Edge e_L or vertex v_L on the lower chain of P whose horizontal span includes $x(q)$
 - Edge e_H or vertex v_H on the upper chain of P whose horizontal span includes $x(q)$
- ◆ We consider four cases
 - If no such edges/vertices exist, we return OUT
 - Else if q is on e_L (v_L) or on e_H (v_H), we return ON
 - Else if q is above e_L (v_L) and below e_H (v_H), we return IN
 - Else, we return OUT



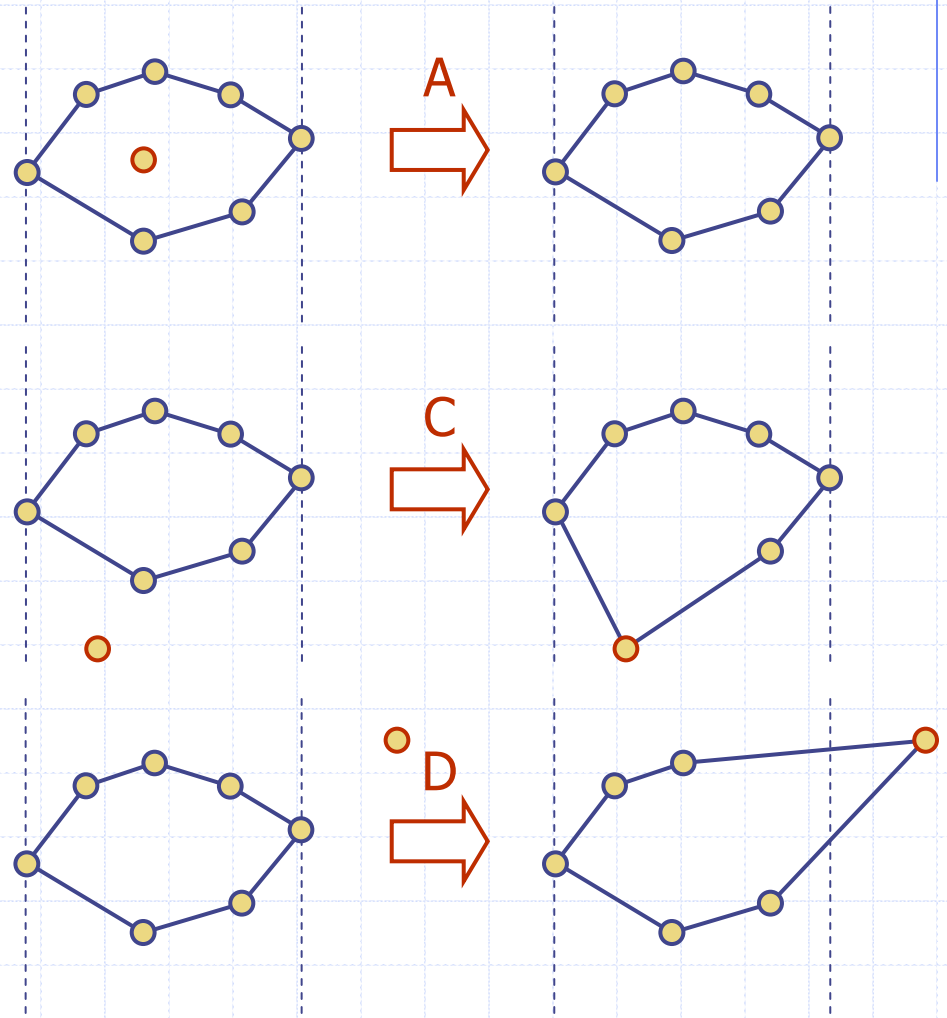
Incremental Convex Hull

- ◆ The incremental convex hull problem consists of performing a series of the following operations on a set S of points
 - **locate**(q): determines if query point q is inside, outside or on the convex hull of S
 - **insert**(q): inserts a new point q into S
 - **hull**(\cdot): returns the convex hull of S
- ◆ Incremental convex hull data structure
 - We store the points of the convex hull and discard the other points
 - We store the hull points in two red-black trees
 - ◆ T_L for the lower hull
 - ◆ T_H for the upper hull

Insertion of a Point

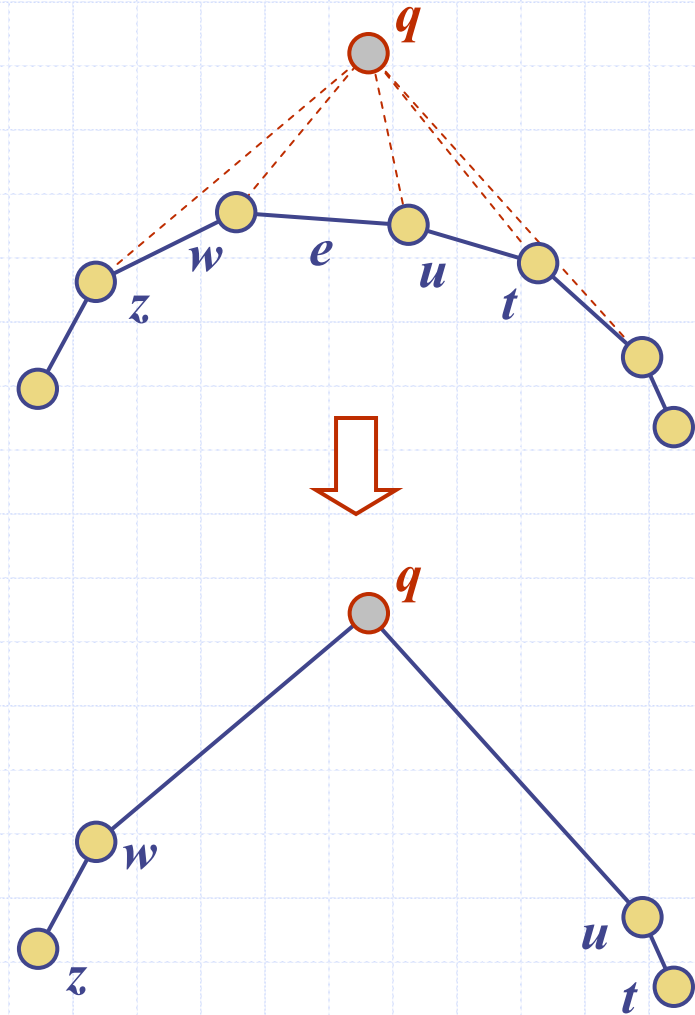
◆ In operation $\text{insert}(q)$, we consider four cases that depend on the location of point q

- A IN or ON: no change
- B OUT and above: add q to the upper hull
- C OUT and below: add q to the lower hull
- D OUT and left or right: add q to the lower and upper hull



Insertion of a Point (cont.)

- ◆ Algorithm to add a vertex q to the upper hull chain in Case B (boundary conditions omitted for simplicity)
 - We find the edge e (vertex v) whose horizontal span includes q
 - $w \leftarrow$ left endpoint (neighbor) of e (v)
 - $z \leftarrow$ left neighbor of w
 - While **orientation**(q, w, z) = CW or COLL
 - ◆ We remove vertex w
 - ◆ $w \leftarrow z$
 - ◆ $z \leftarrow$ left neighbor of w
 - $u \leftarrow$ right endpoint (neighbor) of e (v)
 - $t \leftarrow$ right neighbor of u
 - While **orientation**(t, u, q) = CW or COLL
 - ◆ We remove vertex u
 - ◆ $u \leftarrow t$
 - ◆ $t \leftarrow$ right neighbor of u
 - We add vertex q



Analysis

- ◆ Let n be the current size of the convex hull
 - Operation locate takes $O(\log n)$ time
 - Operation insert takes $O((1 + r)\log n)$ time, where r is the number of points removed from the convex hull
 - Operation hull takes $O(n)$ time
- ◆ Constructing the convex hull of n points by n repeated insertions takes $O(n \log n)$ time