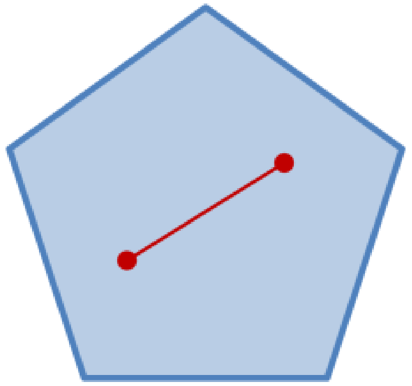
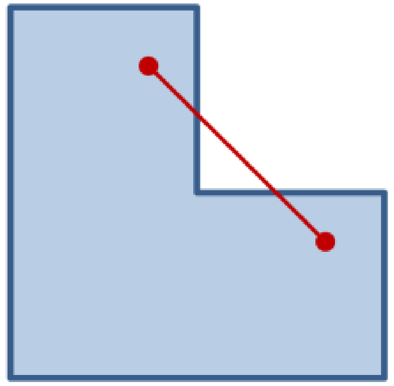
CONVEX HULL

II YEAR E

P.MOHAMED FATHIMAL
SRM-VDP

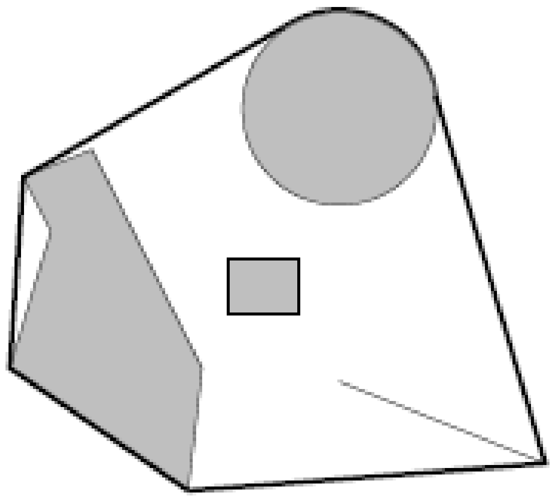
Convex


Convex


Concave

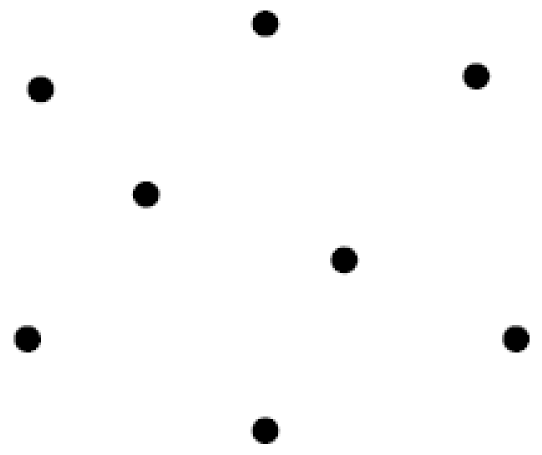
Convex Hull

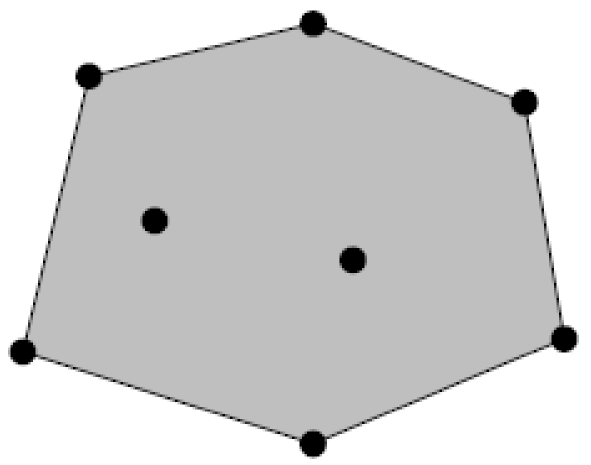
- The intersection of an arbitrary family of convex sets is convex (proof?).
- Let $\mathcal{S} \subset \mathbb{R}^d$. Its *convex hull* $\mathcal{CH}(\mathcal{S})$ is the intersection of all the convex sets that contain $\mathcal{S}$.
- $\mathcal{CH}(\mathcal{S})$ is the smallest convex set containing $\mathcal{S}$



Points in the plane

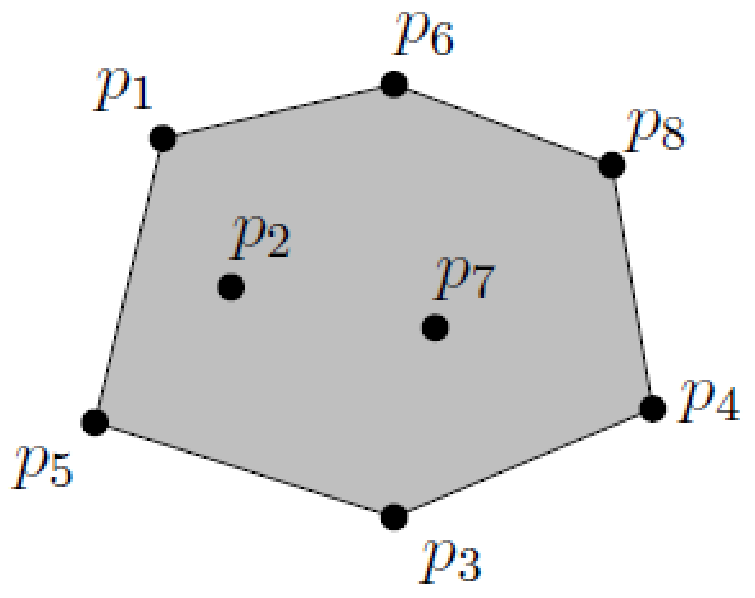
- Let $P = \{p_1, p_2, \ldots p_n\} \subset \mathbb{R}^2$.
- $\mathcal{CH}(P)$ is a convex polygon.


P


$\mathcal{CH}(P)$

Computing a convex hull

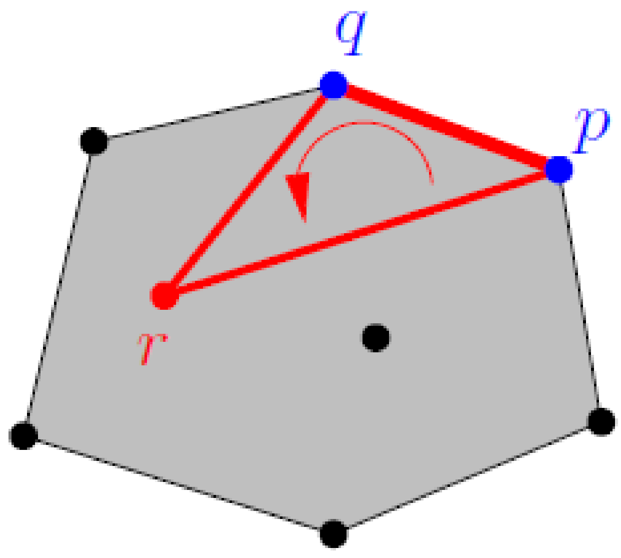
- Input: the set $P = \{p_1, p_2, \ldots p_n\} \subset \mathbb{R}^2$
- Output: a sequence $\mathcal{L} = (c_1, c_2, \ldots c_h)$ of vertices of $\mathcal{CH}(P)$ in counterclockwise order
- Example: $\mathcal{L} = (p_3, p_4, p_8, p_6, p_1, p_5)$



$\mathcal{CH}(P)$

Characterization

The directed edge (p, q) is an edge of $\mathcal{CH}(P)$ iff



$\forall r \in P \setminus \{p, q\}$ the triangle (p, q, r) is oriented counterclockwise.

Orientation test

- We denote

$$CCW(p,q,r) = \begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix}$$

$$= (x_q - x_p)(y_r - y_p) - (x_r - x_p)(y_q - y_p)$$

- Triangle (p,q,r) is counterclockwise iff $CCW(p,q,r) > 0$
- How fast can we perform this test?
  - 2 multiplications and 5 subtractions
  - takes $O(1)$ time

ALgorithms

- Brute Force Approach
- Quick Hull
- Graham’s Scan
- Divide and Conquer

BRUTE FORCE PPROACH

First algorithm

```
Algorithm SlowConvexHull(P)
Input: a set $P$ of points in $\mathbb{R}^2$
Output: $\mathcal{CH}(P)$
1.   $E \leftarrow P^2$
2.   for all $(p, q, r) \in P^3$ such that $r \notin \{p, q\}$
3.       if $CCW(p, q, r) \leq 0$
4.           then remove $(p, q)$ from $E$
5.   Write the remaining edges of $E$ into $\mathcal{L}$ in
     counterclockwise order
6.   Return $\mathcal{L}$
```

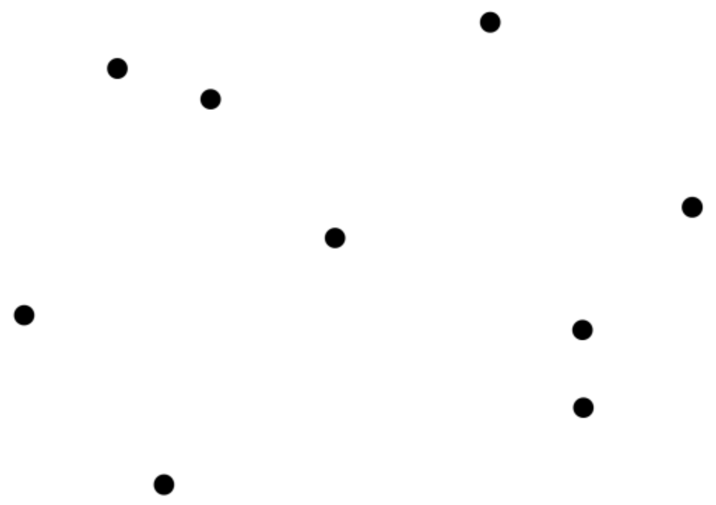
- Line 1: find all directed edges between two points of P
  $\longrightarrow O(n^2)$ time
- Lines 2-4: discard the edges that are not in the convex hull
  $\longrightarrow O(n^3)$ time
- line 5: how fast can you do it, and how?
  $\longrightarrow$ easy to do in $O(n^3)$ time

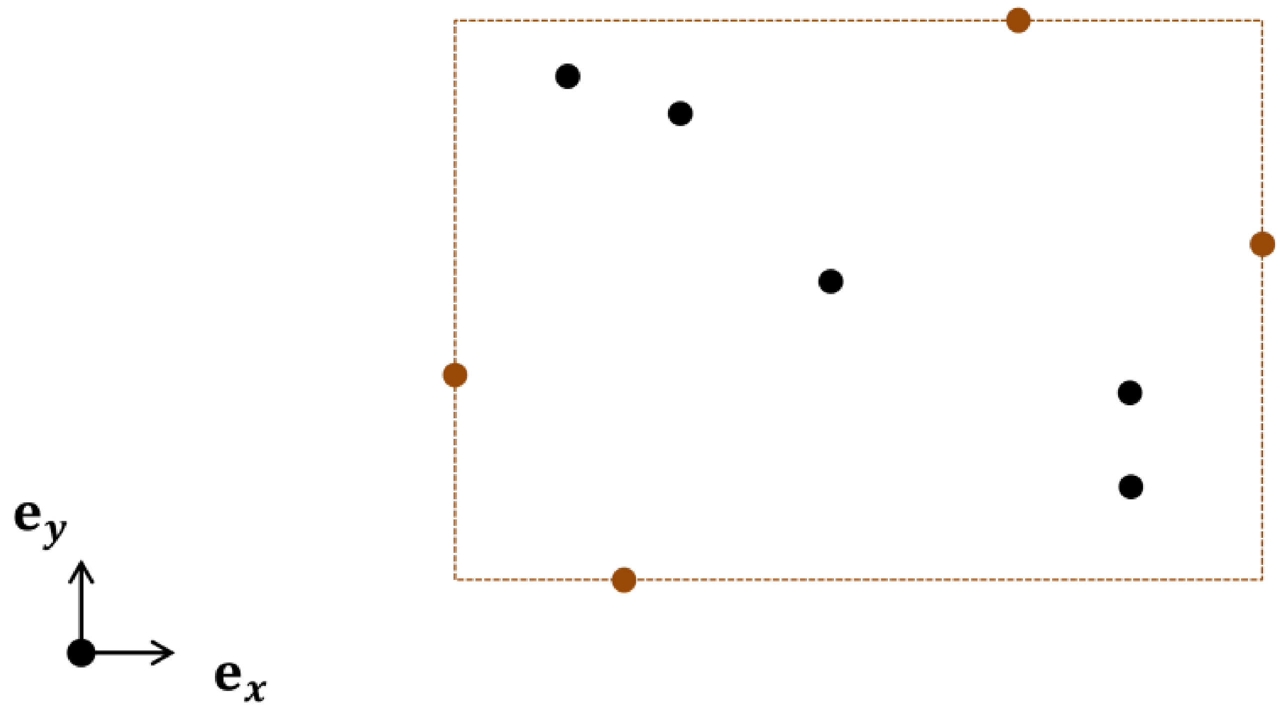
Conclusion: this algorithm runs in $O(n^3)$ time

Actually, $\Theta(n^3)$ time.

Quick Hull

Quickhull 2D: Initial Hull

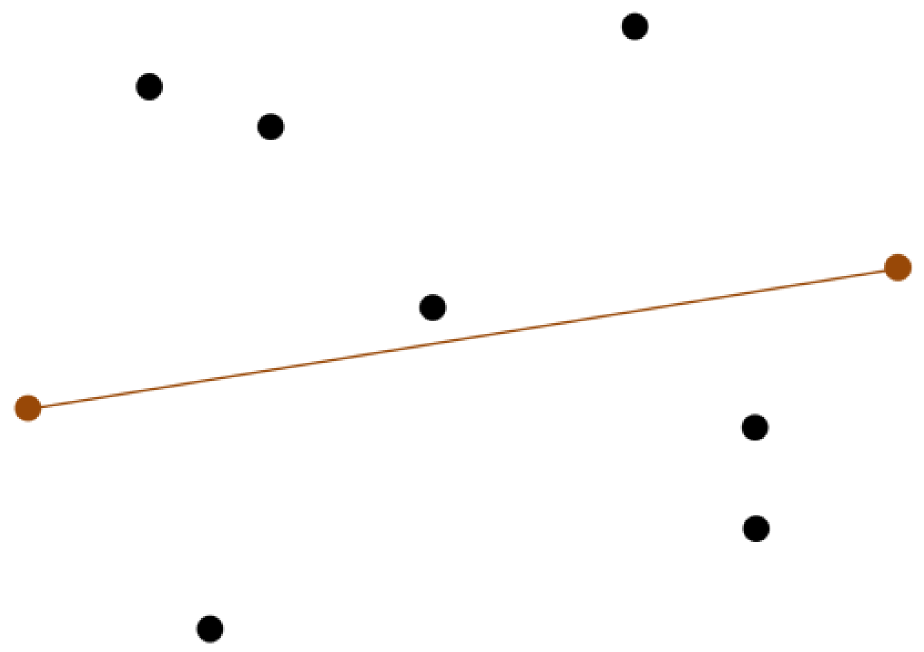




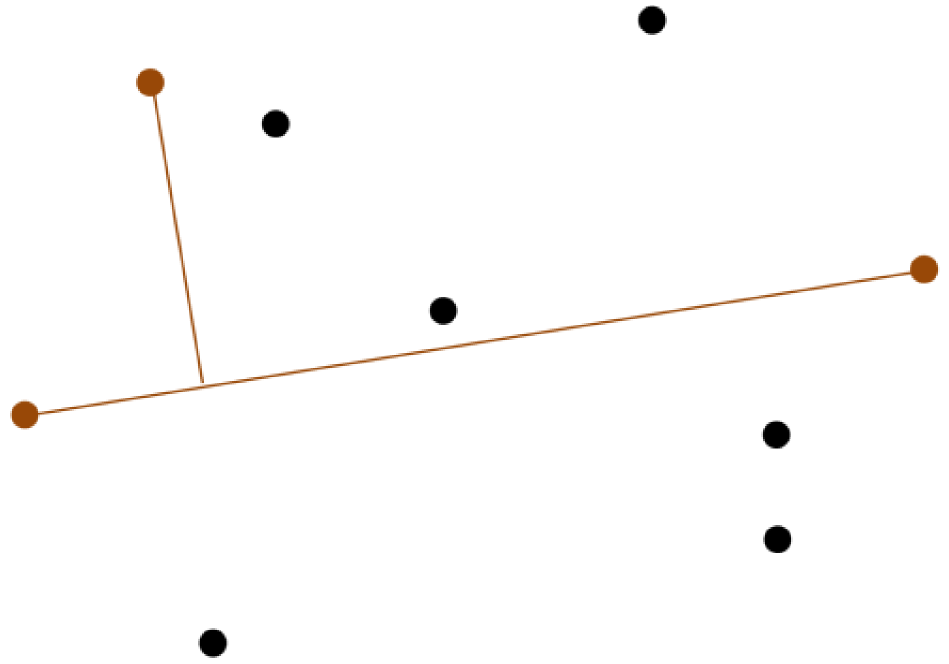
find this initial hull :

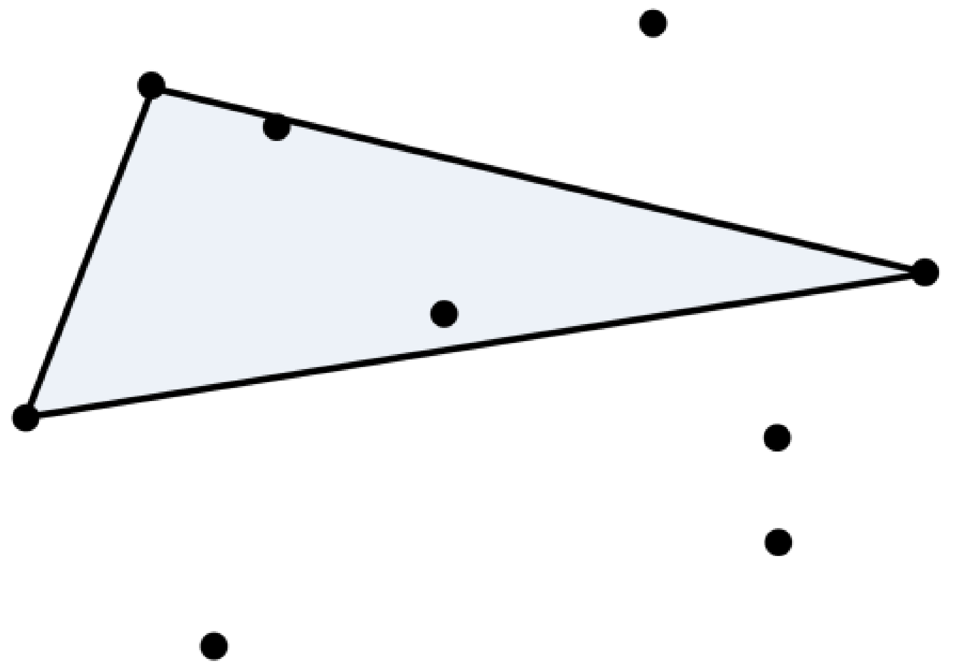
start by identifying the extreme points along each cardinal axis
- we find the points with the smallest and largest x and y values.

choose the pair which is furthest apart
-In this example this would be the left and right-most points

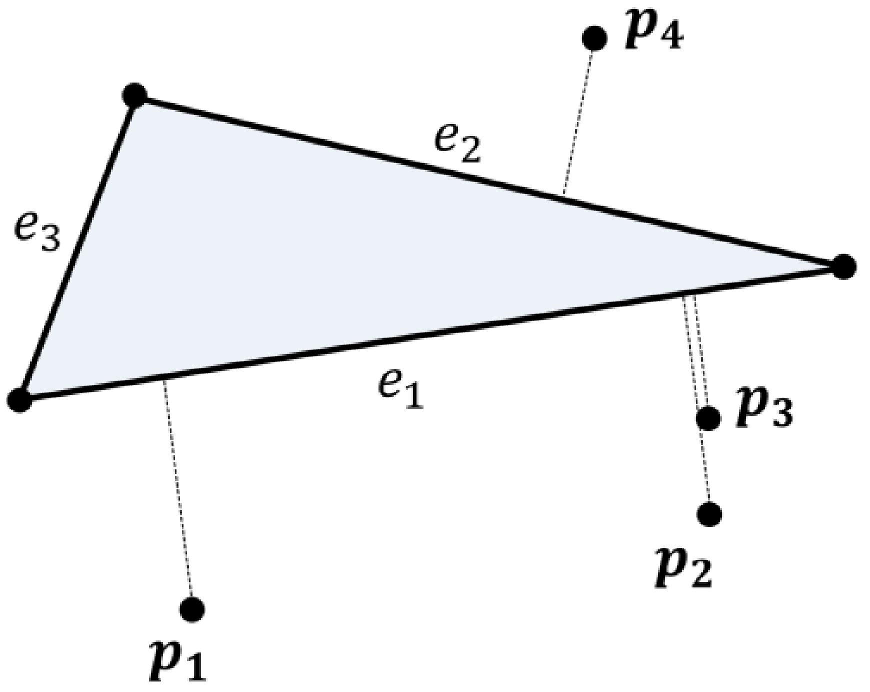


search for the furthest point from the line through these two extreme points





Three points build our initial hull
-In 2D the initial hull is simply a triangle

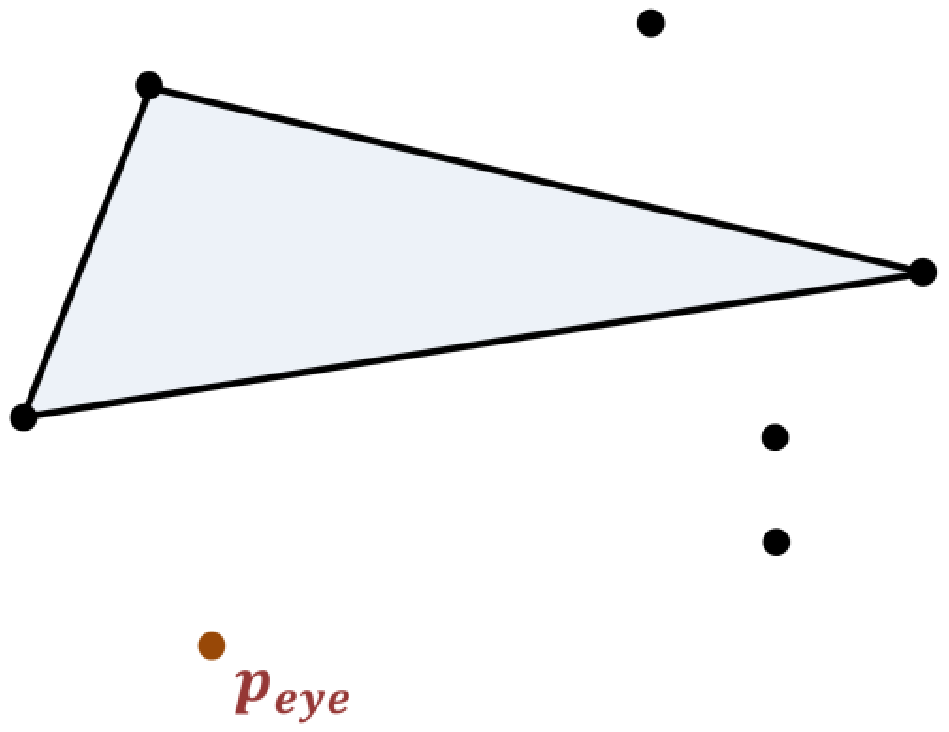


$$CL(e_1) = \{\, \boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3} \}$$

$$CL(e_2) = \{\, \boldsymbol{p_4} \}$$

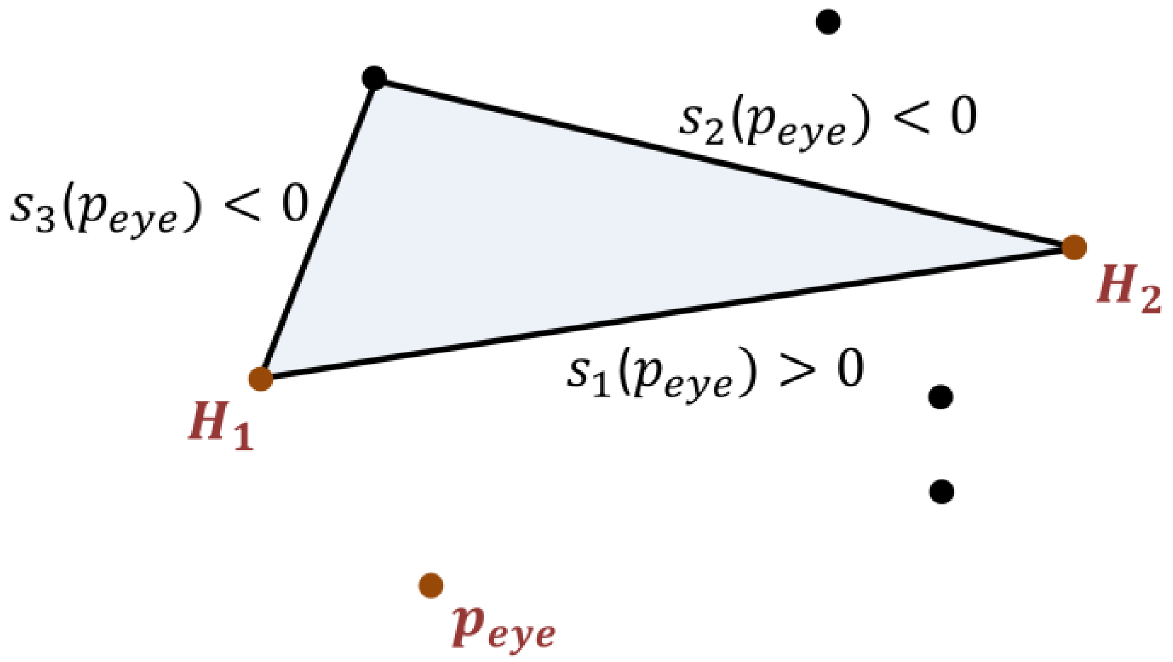
$$CL(e_3) = \{\ \}$$

- The next step is to partition the remaining points and assign each point to its closest face
- -We can also remove internal points since those cannot be on the final hull
  ‘conflict lists’ are the points can “see” the face and therefore are potentially on the final hull

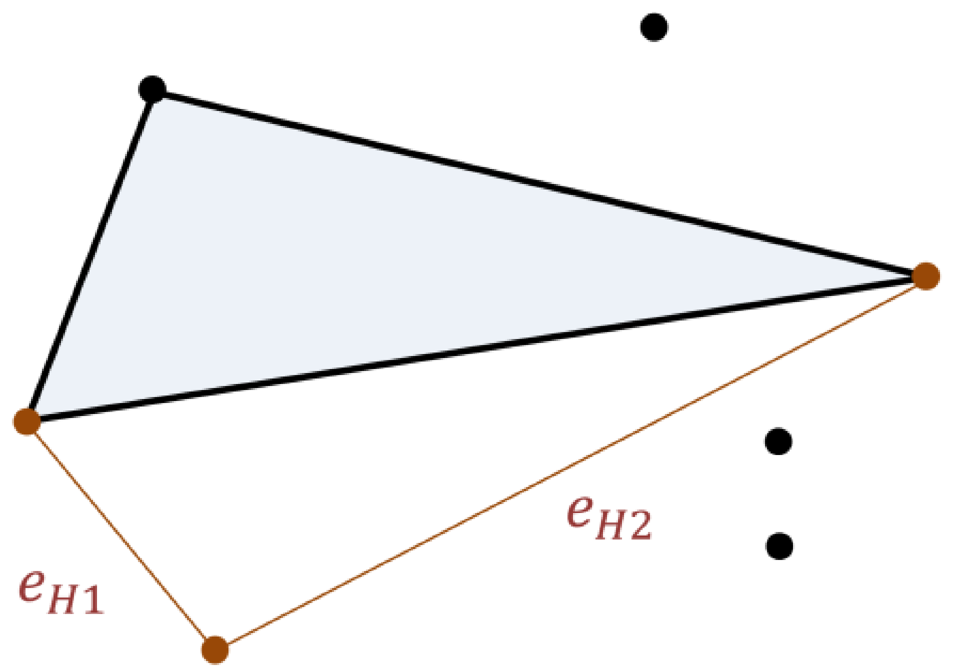


- To add a new point to our intermediate hull. We iterate conflict lists and find the point p with the largest distance from the hull:
- -Let’s call this point the **eye** point

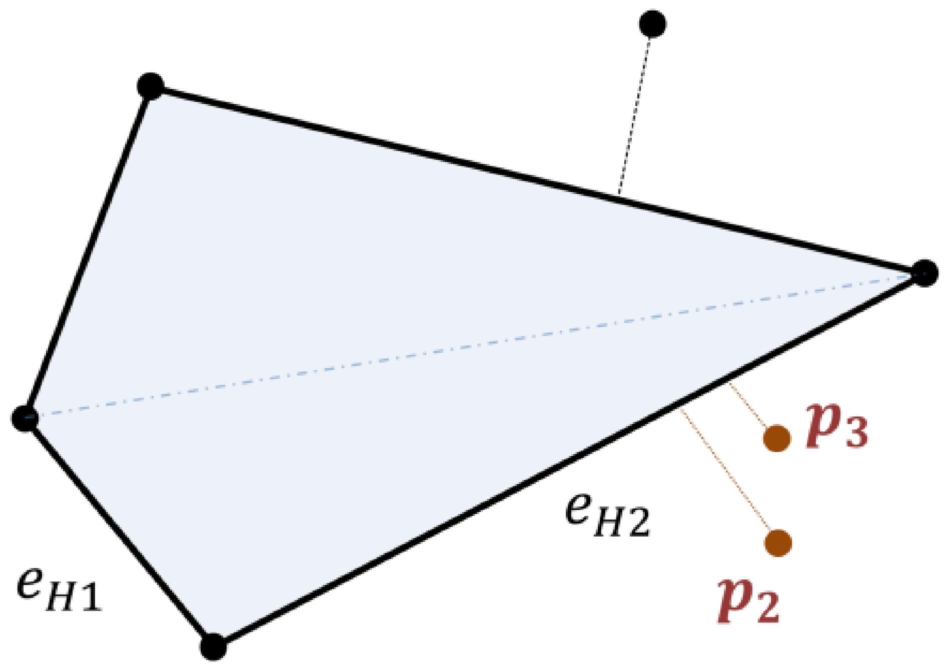
Identify all faces that are visible from the newly added point since these faces cannot be on the hull:
-A face is visible if the new point is in front of the face plane.
- use simple plane tests to classify the new point against each face!
- then find the two vertices that connect a visible with a non-visible face.
-These two vertices form the horizon



Once we identified the two horizon vertices we then create two new faces for each horizon vertex to connect the new vertex to hull

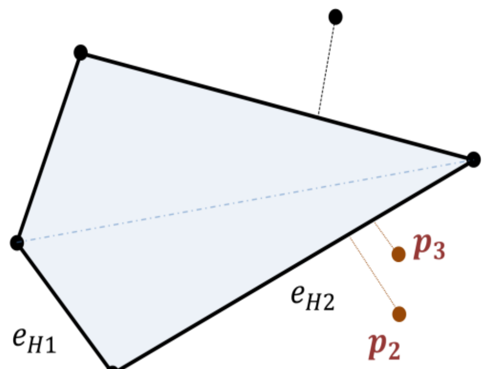


Handle their conflict lists since these conflict points can still be on the final hull
---→ by simply partitioning these orphaned vertices to the new faces



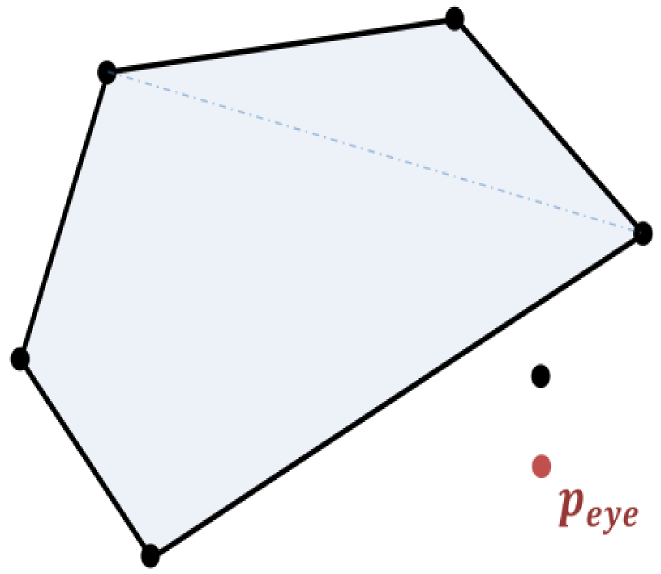
$CL(e_{H1}) = \{\,\}$

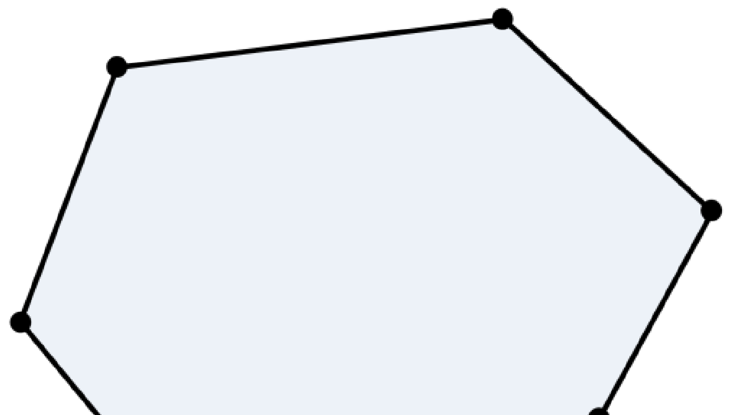
$CL(e_{H2}) = \{p_2, p_3\}$

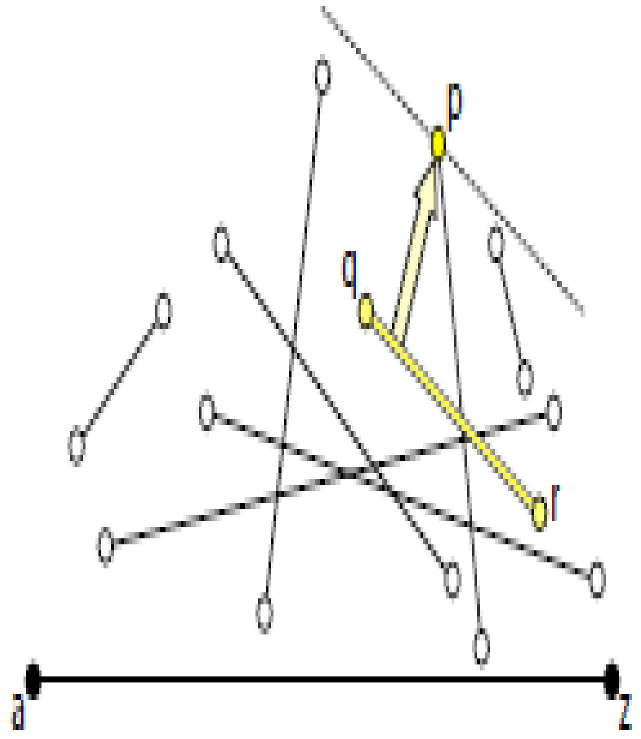


$CL(e_{H1}) = \{\,\}$

$CL(e_{H2}) = \{p_2, p_3\}$

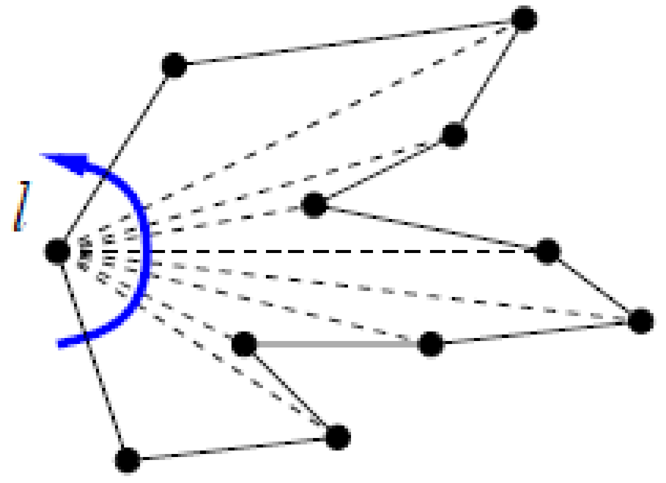






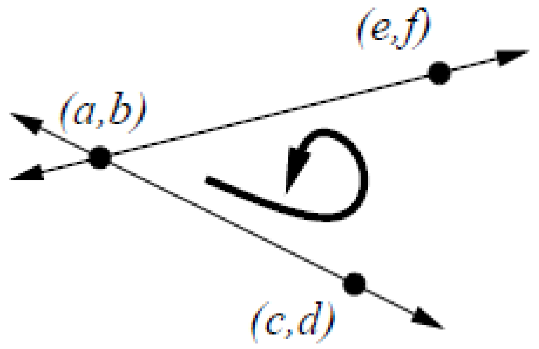
```
QUICKHULL($P, a, z$):
  ⟨⟨Precondition: Every point in $P$ lies to the left of $\overrightarrow{az}$⟩⟩
  if $P = \emptyset$
      $next[z] \leftarrow a$;  $pred[a] \leftarrow z$
  else
      $p \leftarrow$ point in $P$ furthest to the left of $\overrightarrow{az}$
      $L \leftarrow$ all points in $P$ to the left of $\overrightarrow{ap}$
      $R \leftarrow$ all points in $P$ to the left of $\overrightarrow{pz}$
      QUICKHULL($L, a, p$)
      QUICKHULL($R, p, z$)
```

Grahams Scan Algorithm



[illegible]

Grahams Scan Algorithm



Three points in counterclockwise order.

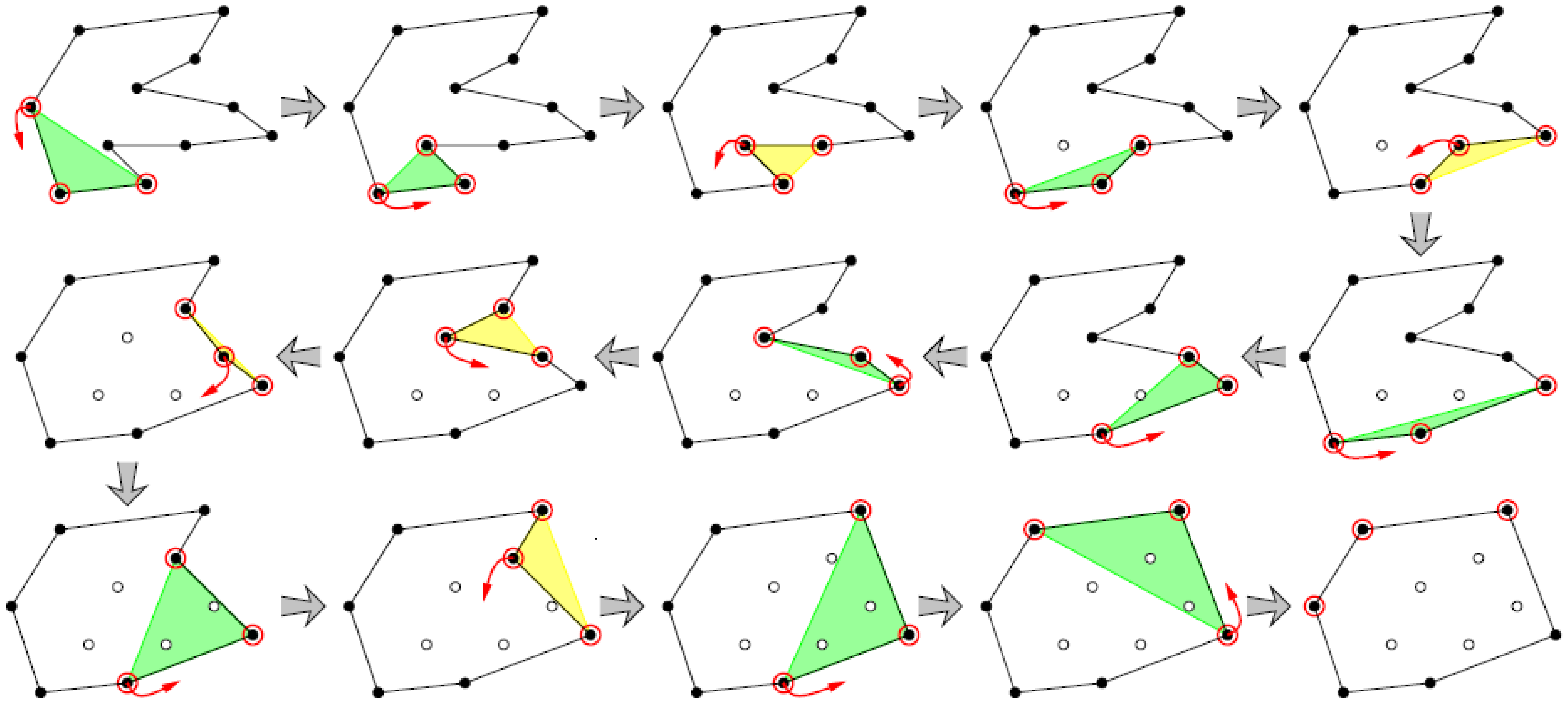
$$\text{counterclockwise} \iff \begin{vmatrix} c-a & d-b \\ e-a & f-b \end{vmatrix} > 0$$

$$\text{counterclockwise} \iff \begin{vmatrix} 1 & a & b \\ 1 & c & d \\ 1 & e & f \end{vmatrix} > 0$$

Grahams Scan Algorithm

- Find the leftmost point l.
- sort the points in counterclockwise order around l. (O(n log n) time with any comparison-based sorting algorithm (quicksort, mergesort, heapsort, etc.).
- To compare two points p and q,
  - check whether the triple l; p; q is oriented clockwise or counterclockwise.
- Once the points are sorted, we connected them in counterclockwise order, starting and ending at l.
- The result is a simple polygon with n vertices.

Graham's Three penny algorithm



- Take three pennies → three consecutive vertices p; q; r of the polygon;
- initially, these are l and the two vertices after l.
- apply the following two rules over and over until a penny is moved forward onto l.
- If p; q; r are in counterclockwise order, move the back penny forward to the successor of r.
- If p; q; r are in clockwise order, remove q from the polygon, add the edge pr, and move the middle penny backward to the predecessor of p.

Graham's scan

Graham's scan solves the convex-hull problem by maintaining a stack S of candidate points. Each point of the input set Q is pushed once onto the stack, and the points that are not vertices of CH(Q) are eventually popped from the stack. When the algorithm terminates, stack S contains exactly the vertices of CH(Q), in counterclockwise order of their appearance on the boundary.

The procedure GRAHAM-SCAN takes as input a set Q of points, where $|Q| \geq 3$. It calls the functions TOP(S), which returns the point on top of stack S without changing S, and NEXT-TO-TOP(S), which returns the point one entry below the top of stack S without changing S. As we shall prove in a moment, the stack S returned by GRAHAM-SCAN contains, from bottom to top, exactly the vertices of CH(Q) in counterclockwise order.

```
GRAHAM-SCAN(Q)
 1  let $p_0$ be the point in $Q$ with the minimum $y$-coordinate,
              or the leftmost such point in case of a tie
 2  let $\langle p_1, p_2, \ldots, p_m \rangle$ 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 $p_0$)
 3  PUSH($p_0$, S)
 4  PUSH($p_1$, S)
 5  PUSH($p_2$, S)
 6  for $i \leftarrow 3$ to $m$
 7       do while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                    and $p_i$ makes a nonleft turn
 8               do POP(S)
 9          PUSH($p_i$, S)
10  return S
```

Figure 33.7 illustrates the progress of GRAHAM-SCAN. Line 1 chooses point p_0 as the point with the lowest y-coordinate, picking the leftmost such point in case of a tie. Since there is no point in Q that is below p_0 and any other points with the same y-coordinate are to its right, p_0 is a vertex of CH(Q). Line 2 sorts the remaining points of Q by polar angle relative to p_0, using the same method—comparing cross products—as in Exercise 33.1-3. If two or more points have the same polar angle relative to p_0, all but the farthest such point are convex combinations of p_0 and the farthest point, and so we remove them entirely from consideration.

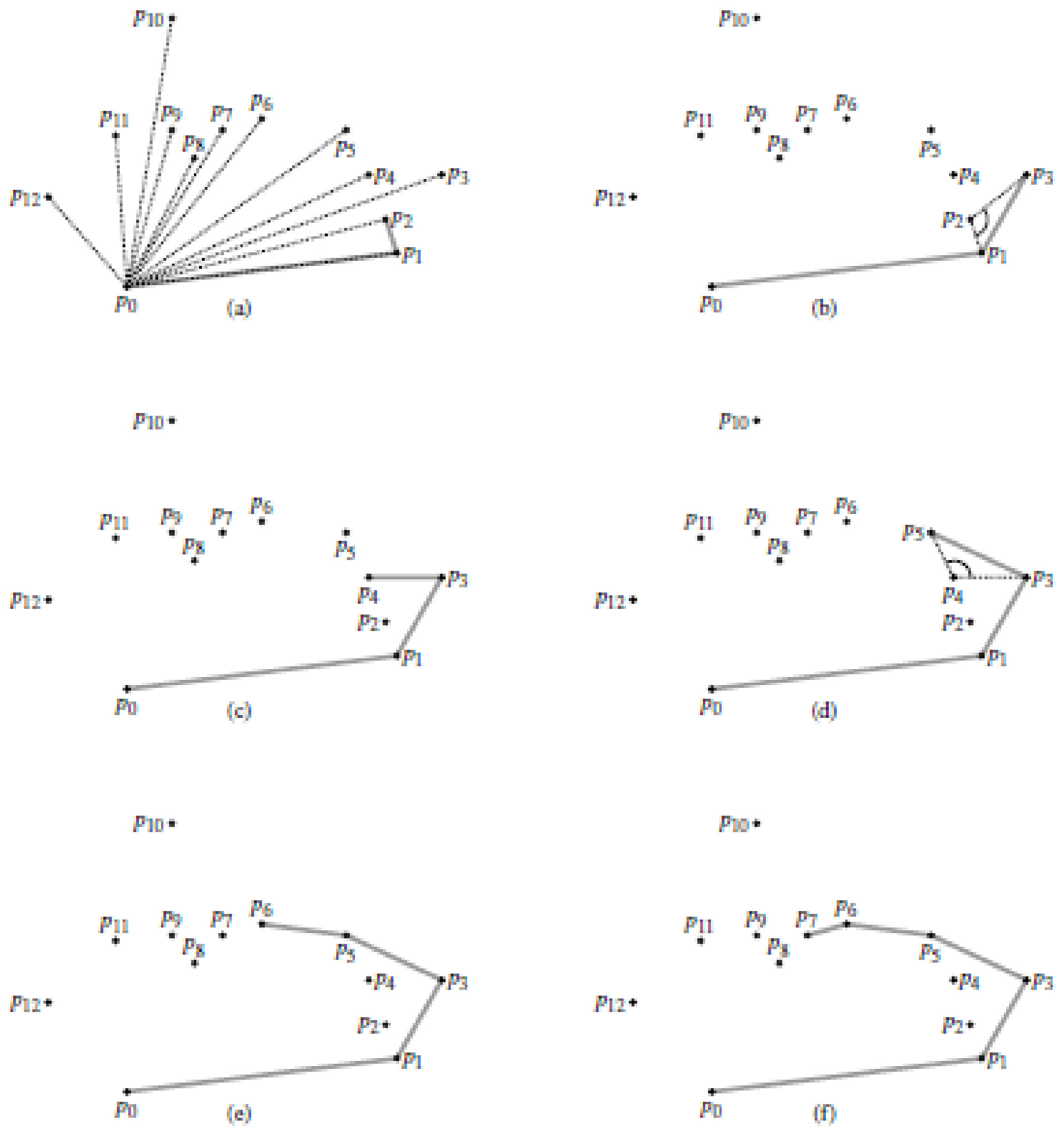
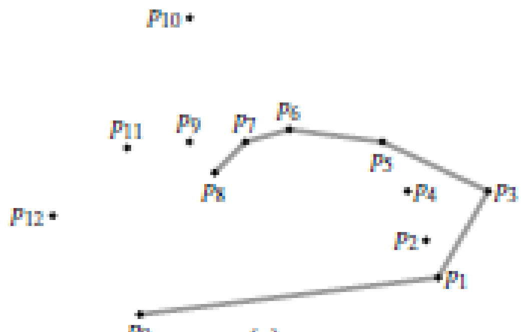
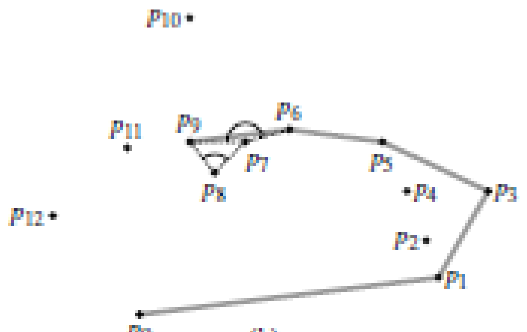


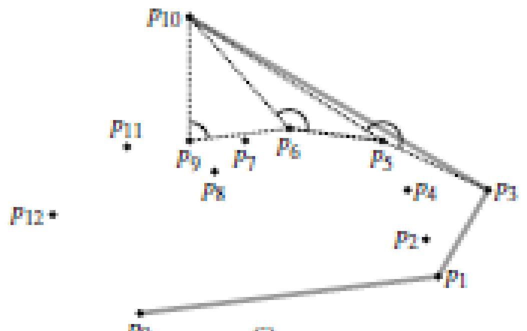
Figure 33.7 The execution of GRAHAM-SCAN on the set Q of Figure 33.6. The current convex hull contained in stack S is shown in gray at each step. **(a)** The sequence $\langle p_1, p_2, \ldots, p_{12} \rangle$ of points


p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_4
p_3
p_{12}
p_2
p_1
p_0

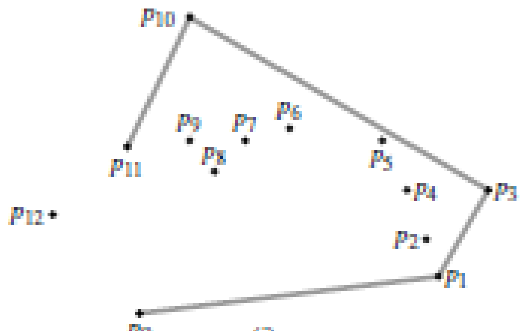
(g)


p_{10}
p_{11}
p_9
p_6
p_7
p_5
p_8
p_4
p_3
p_{12}
p_2
p_1
p_0

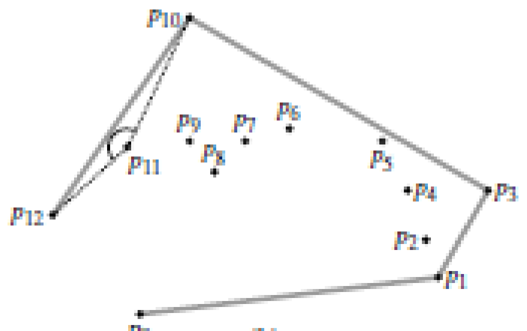
(h)


p_{10}
p_{11}
p_9
p_7
p_6
p_5
p_8
p_4
p_3
p_{12}
p_2
p_1
p_0

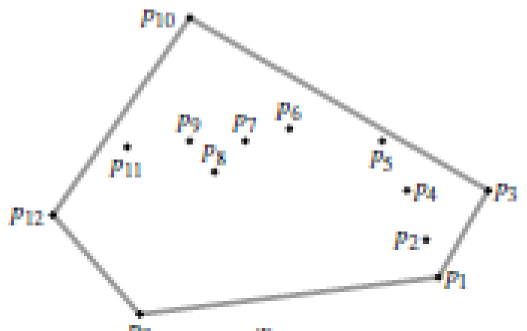
(i)


p_{10}
p_9
p_7
p_6
p_8
p_5
p_{11}
p_4
p_3
p_{12}
p_2
p_1
p_0

(j)


p_{10}
p_9
p_7
p_6
p_8
p_{11}
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0

(k)


p_{10}
p_9
p_7
p_6
p_8
p_{11}
p_5
p_4
p_3
p_{12}
p_2
p_1
p_0

(l)

We let m denote the number of points other than p_0 that remain. The polar angle, measured in radians, of each point in Q relative to p_0 is in the half-open interval $[0, \pi)$. Since the points are sorted according to polar angles, they are sorted in counterclockwise order relative to p_0. We designate this sorted sequence of points by $\langle p_1, p_2, \ldots, p_m \rangle$. Note that points p_1 and p_m are vertices of $\text{CH}(Q)$ (see Exercise 33.3-1). Figure 33.7(a) shows the points of Figure 33.6 sequentially numbered in order of increasing polar angle relative to p_0.

The remainder of the procedure uses the stack S. Lines 3–5 initialize the stack to contain, from bottom to top, the first three points p_0, p_1, and p_2. Figure 33.7(a) shows the initial stack S. The **for** loop of lines 6–9 iterates once for each point in the subsequence $\langle p_3, p_4, \ldots, p_m \rangle$. The intent is that after processing point p_i, stack S contains, from bottom to top, the vertices of $\text{CH}(\{p_0, p_1, \ldots, p_i\})$ in counterclockwise order. The **while** loop of lines 7–8 removes points from the stack if they are found not to be vertices of the convex hull. When we traverse the convex hull counterclockwise, we should make a left turn at each vertex. Thus, each time the **while** loop finds a vertex at which we make a nonleft turn, the vertex is popped from the stack. (By checking for a nonleft turn, rather than just a right turn, this test precludes the possibility of a straight angle at a vertex of the resulting convex hull. We want no straight angles, since no vertex of a convex polygon may be a convex combination of other vertices of the polygon.) After we pop all vertices that have nonleft turns when heading toward point p_i, we push p_i onto the stack. Figures 33.7(b)–(k) show the state of the stack S after each iteration of the **for** loop. Finally, GRAHAM-SCAN returns the stack S in line 10. Figure 33.7(l) shows the corresponding convex hull.

Time Analysis for Grahams Algorithm

- bottleneck
- → sorting the points by polar angles→ O(n log n) time.
- Since every point is added to H exactly once and every point is removed from H at most once, iterating through the points and forming H after the sorting takes O(n) time.
- Thus the whole algorithm takes O(n log n) time.

Divide and Conquer convex hull Algorithm

----→resembles quicksort.

Step1 :choose a pivot point p.

Step 2:Partitions the input points into two sets L and R,

L- >containing the points to the left of p, including p itself,

R-> and the points to the right of p, by comparing x-coordinates.

Step 3: Recursively compute the convex hulls of L and R.

Step 4: Finally, merge the two convex hulls into the final output.

Divide and Conquer

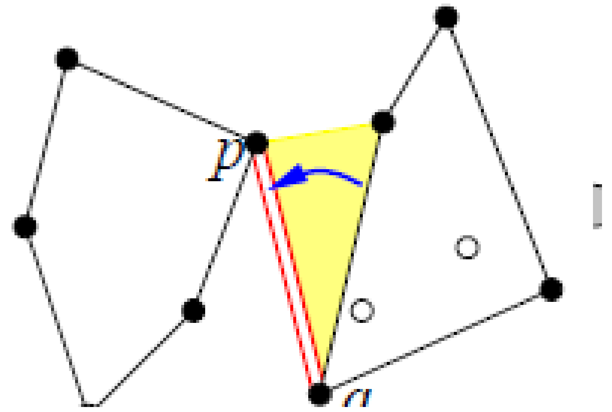
Merge Step:

4.1 connect the two hulls with a line segment between the rightmost point of the hull of L with the leftmost point of the hull of R.

4.1.1. Call these points p and q, respectively.

4.1.2. Add two copies of the segment pq and call them bridges.

4.1.3. Since p and q can `see' each other, this creates a sort of dumbbell-shaped polygon, which is convex except possibly at the endpoints of the bridges.



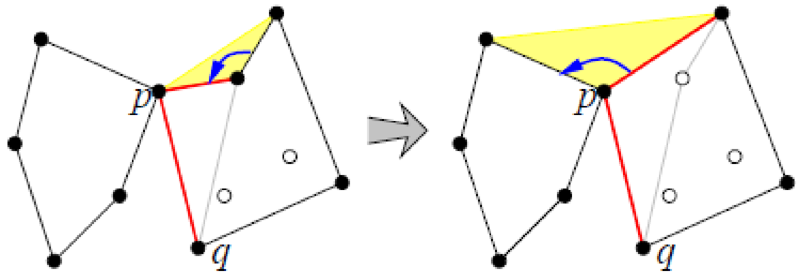
4.1.4. Expand this dumbbell into the correct convex hull as follows.

As long as there is a clockwise turn at either endpoint of either bridge,

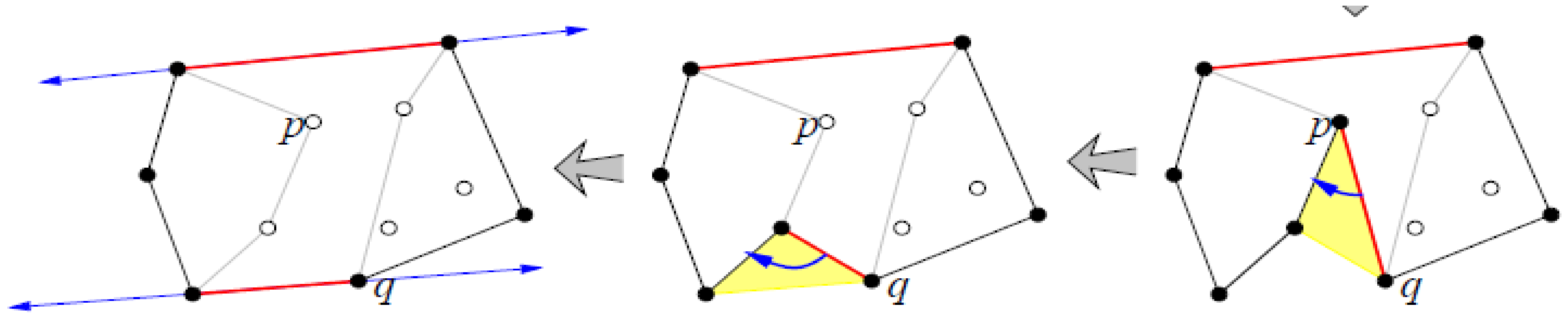
-> remove that point from the circular sequence of vertices and connect its two neighbors.

4.1.5. As soon as the turns at both endpoints of both bridges are counterclockwise, we can stop.

•



- At that point, the bridges lie on the upper and lower common tangent
- lines of the two subhulls.
- These are the two lines that touch both subhulls, such that both subhulls lie below the upper common tangent line and above the lower common tangent line.



Time Analysis

- Merging the two subhulls takes $O(n)$ time in the worst case.
- Thus, the running time is given bythe recurrence
- $T(n) = O(n)+ T(k)+ T(n-k)$, just like quicksort,

  where k the number of points in R.
- the worst-case running time of this algorithm is $O(n2)$.
- If we choose the pivot point randomly, the expected running time is $O(n \log n)$.