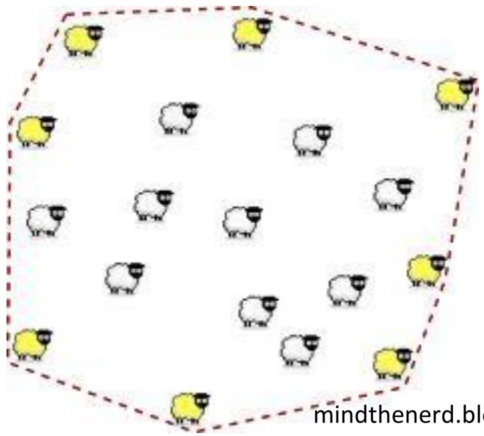
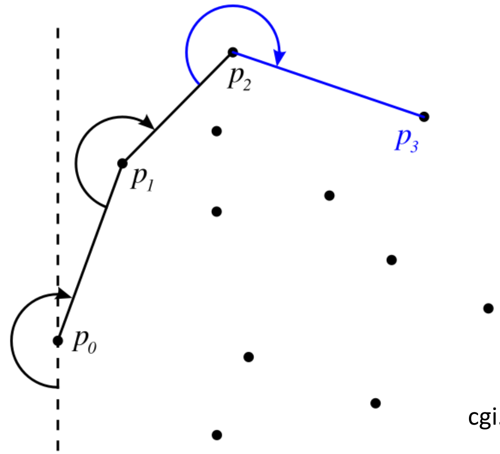
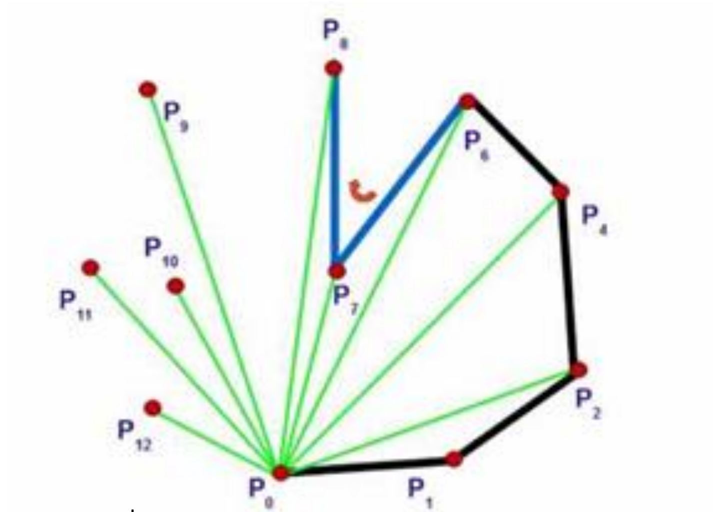

mindthenerd.blogspot.com

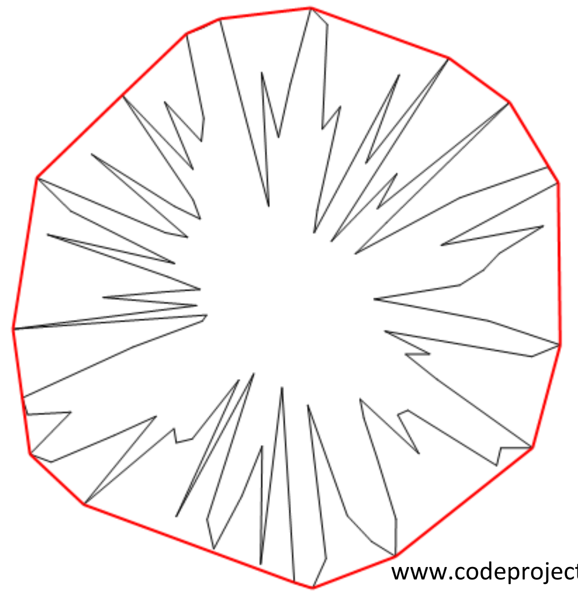


cgi.di.uoa.gr

Convex hull – Graham Scan



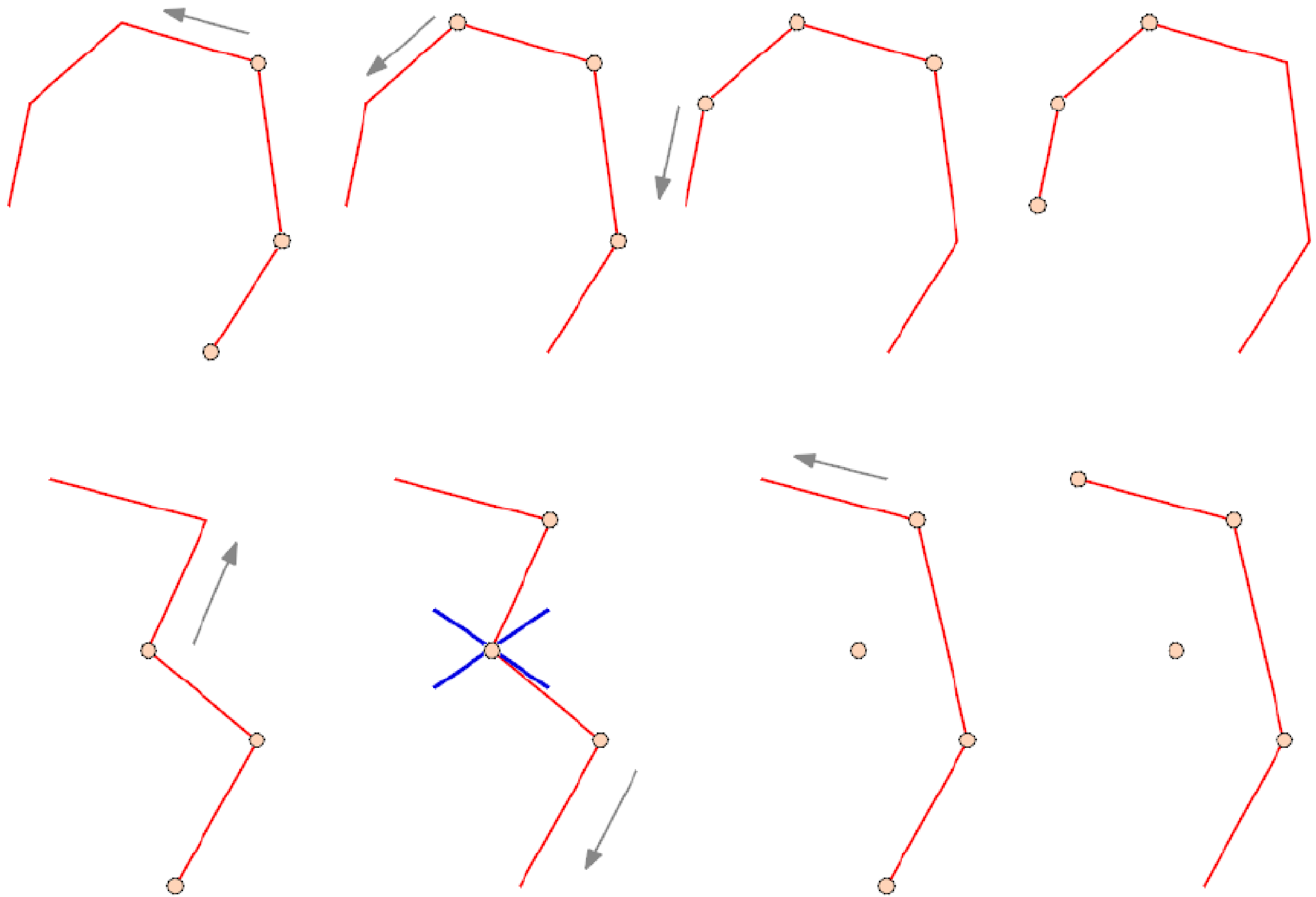
www.youtube.com


www.codeproject.com

Graham Scan

- By Ronald Graham (1972)
- Algorithm cannot be extended to 3D
- Complexity: *O(n.log(n))*
- Can be applied also to large datasets
- The core of the algorithm is based on the ***"left turn" criterion*** which has to be fulfilled by all sorted triplets of points on the convex hull

Left turn criterion



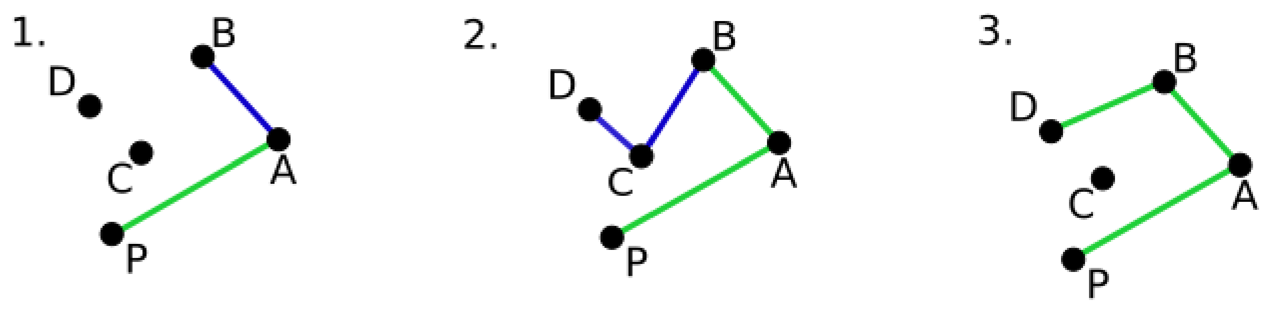
Graham Scan

- In the first step, we find point *P* with the smallest y-axis value (when there are more such points, we take that one with the smallest x-axis value)
- In the second step, we sort all points in the ascending order according to the angle between a given point *P* and x axis

Graham Scan

- Then we go through the list of sorted points and for the triplet of subsequent points we check the left turn criterion:
  - If we move counter-clockwise, these points are lying on the convex hull
  - If we move clockwise, the middle point of the triplet cannot lie on the convex hull and has to be removed – this is repeated until the triplets change the direction back to counter-clockwise

Graham Scan



- If the triplet lies on one line, we can remove the middle point or keep it (according to current algorithm requirements)

Graham Scan

- Implementation of the left turn criterion:
  - Using Half Edge
  - We don’t have to calculate the angle between two line segments:

    For three points (x_1, y_1), (x_2, y_2) a (x_3, y_3) we calculate the cross product of two vectors – from (x_1, y_1) to (x_2, y_2) and from (x_1, y_1) to (x_3, y_3):

$$\mathbf{(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)}$$

Graham Scan

- When the result is:

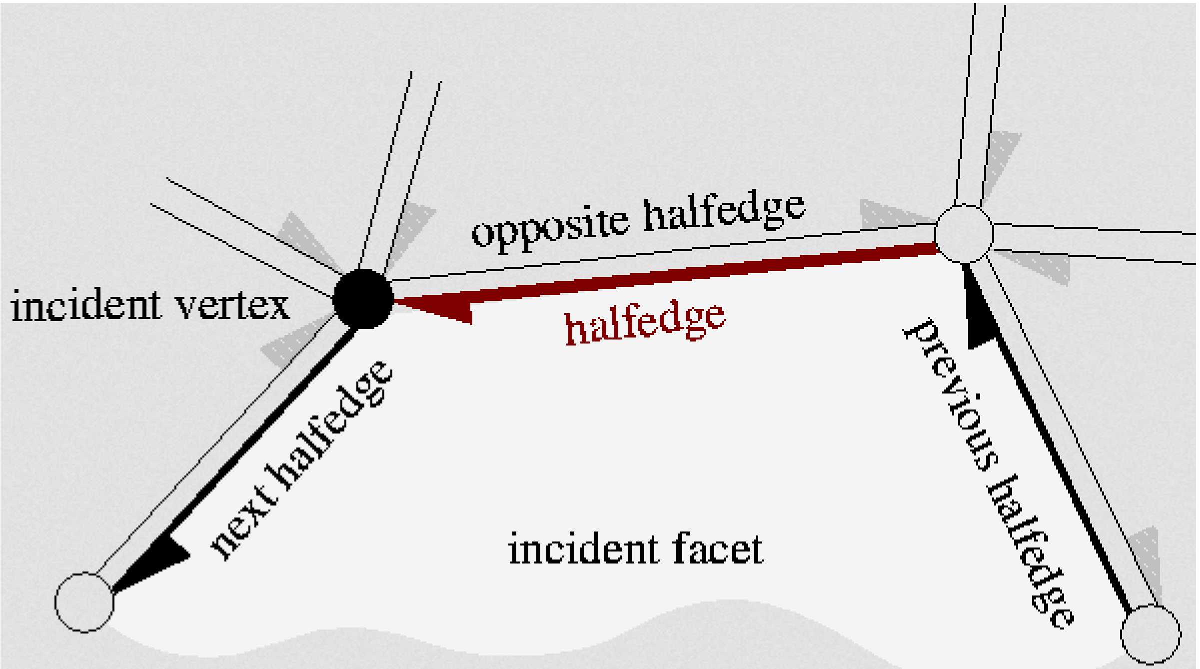
0	points lie on one line
> 0	points are oriented counter-clockwise (fulfill the left turn criterion)
< 0	points are oriented clockwise (fulfill the right turn criterion)

Half Edge

- Data structure for storing the information about neighboring vertices, edges, and faces
- Each edge is divided into two “half-edges“ with opposite orientation
- Each half-edge points to one face and one vertex
- A reduced variant of this data structure can skip some information, e.g., pointers to faces (if we don’t need them)

Half Edge

- http://www.cgal.org/Manual/latest/doc_html/cgal_manual/HalfedgeDS/Chapter_main.html
- http://halfedgelib.sourceforge.net/



Graham Scan - pseudocode

1. Find pivot q, the most right point $q_x = \max(x_i)$, $q \in H$ (convex hull)
2. Sort all points according to the angle with q, index j corresponds to this sorted order
3. If we find two points with the same angle, remove the one closer to q
4. Initialize $j = 2$, create stack Q
5. *push*(q, p_1) to Q (indices of the two last points p_t, p_{t-1})
6. For $j < n$ (number of points) repeat:

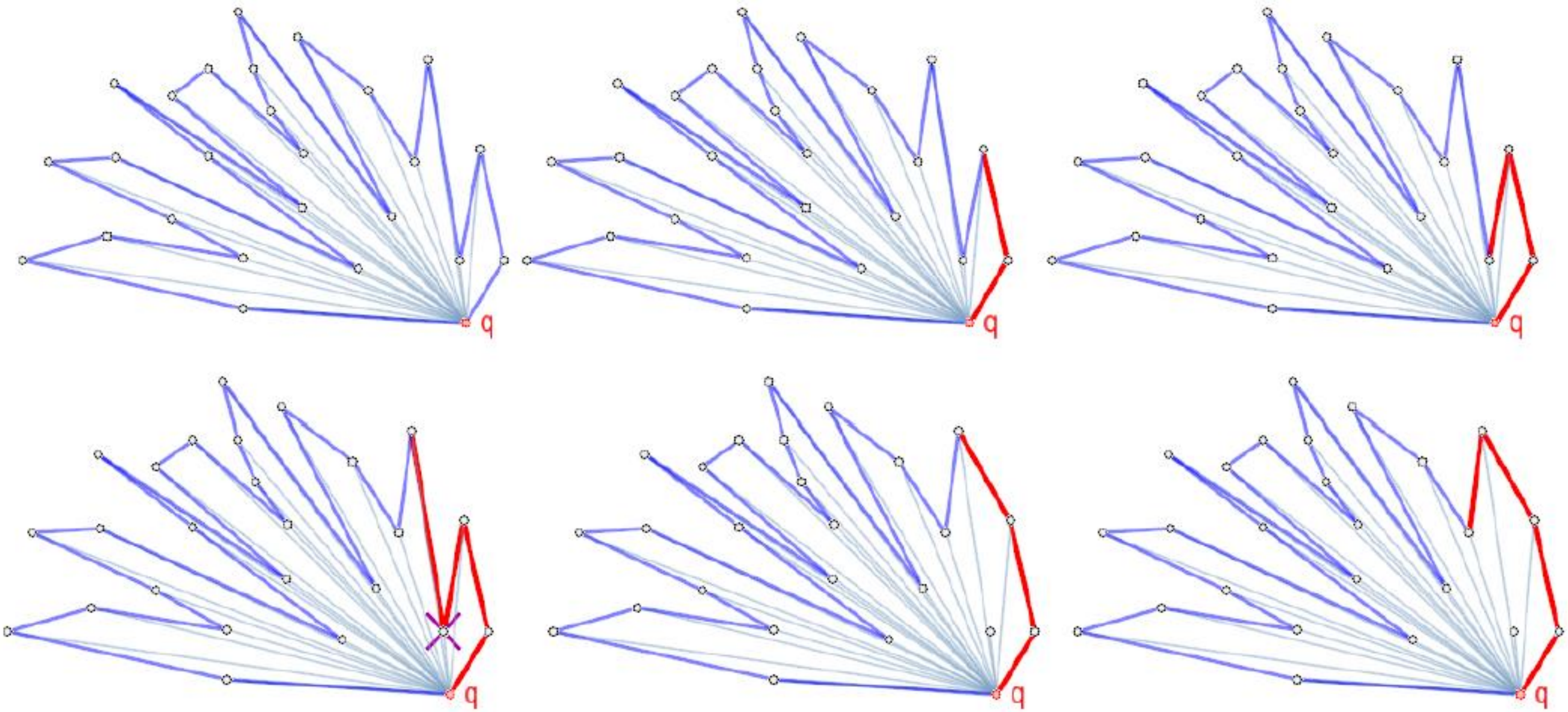
   if p_j is on the left side from p_{t-1}, p_t

      push p_j to Q

      $j = j + 1$

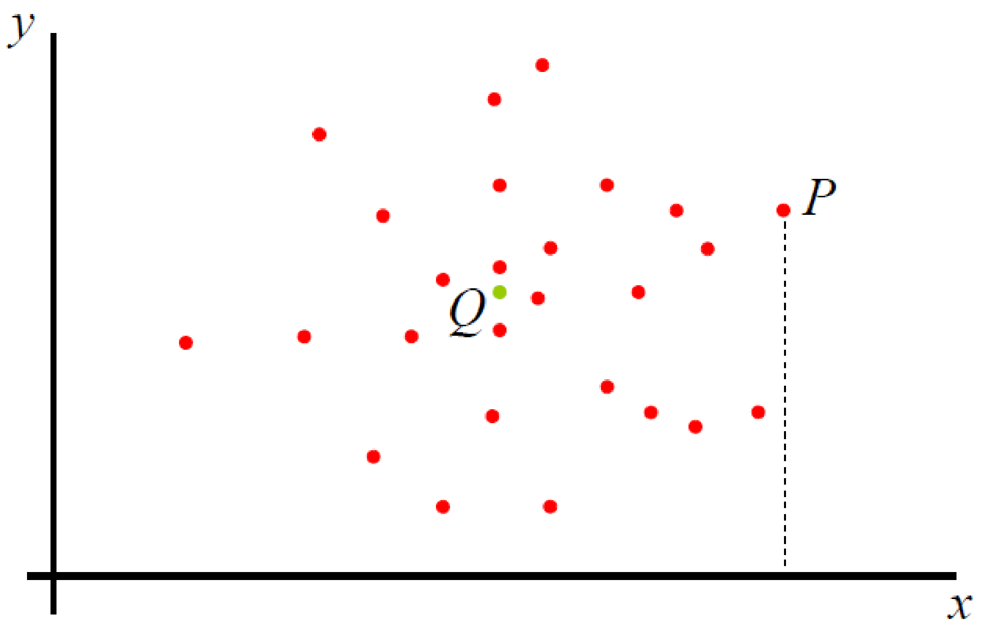
   else *pop* Q

Graham Scan



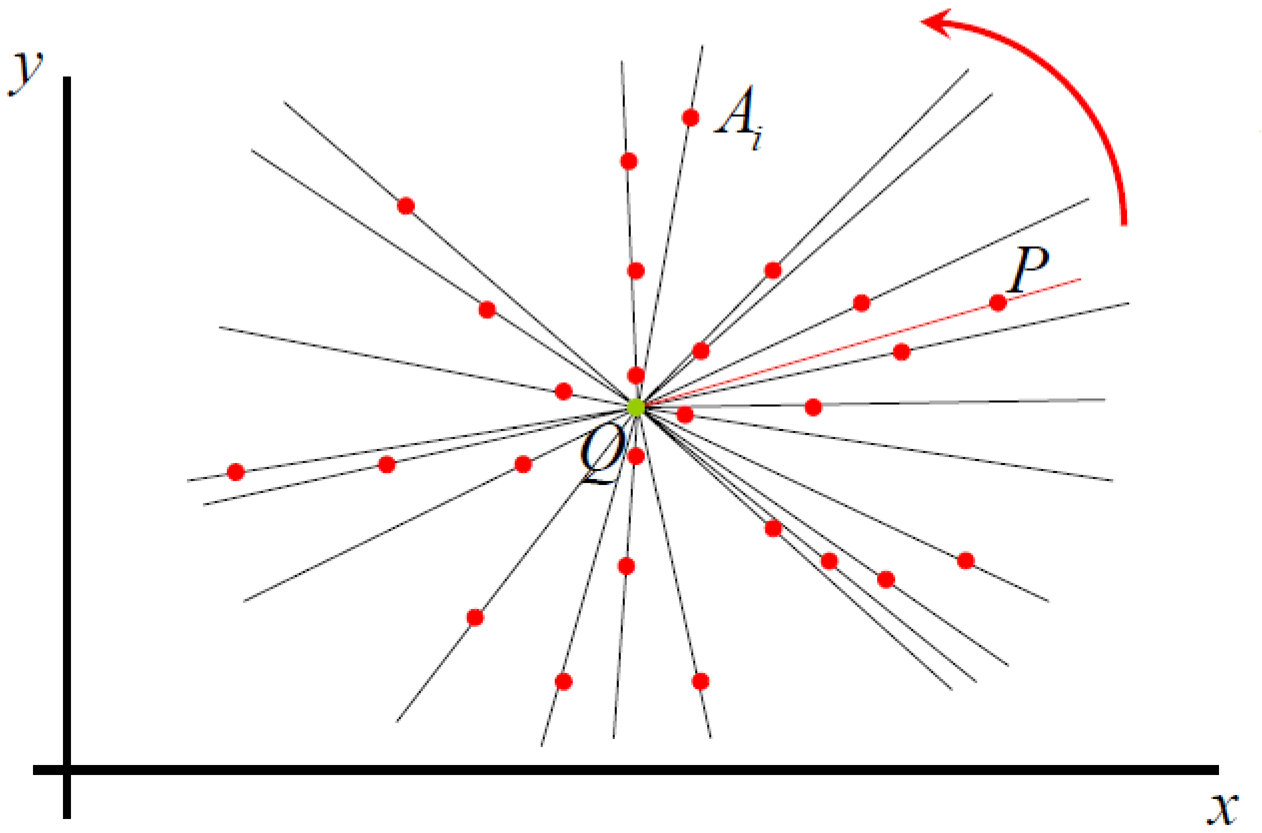
Similar approach

- We find point *P* with the highest x-axis value (it lies on the convex hull)
- We select a point *Q* inside the set (e.g., the center of mass)



Similar approach

- From Q we shoot rays QA_i and sort them starting from QP

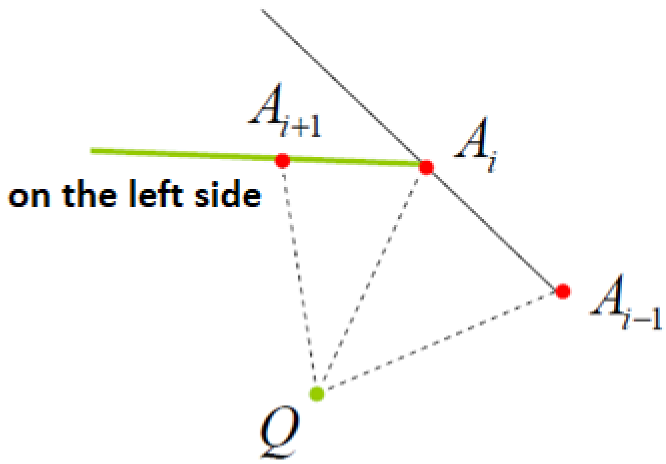


Similar approach

- We proces A_i one by one according to the sorting and determine if the given vertex lies on the convex hull
- Points $A_1, A_2, \ldots, A_n$ are sorted so for each point we know its **predecessor** and **successor**:
  - A_{i-1} (predecessor), A_i, A_{i+1} (successor)
- The criterion: The successor lies always on the left side from the line connecting A_{i-1} with A_i, if it belongs to the convex hull

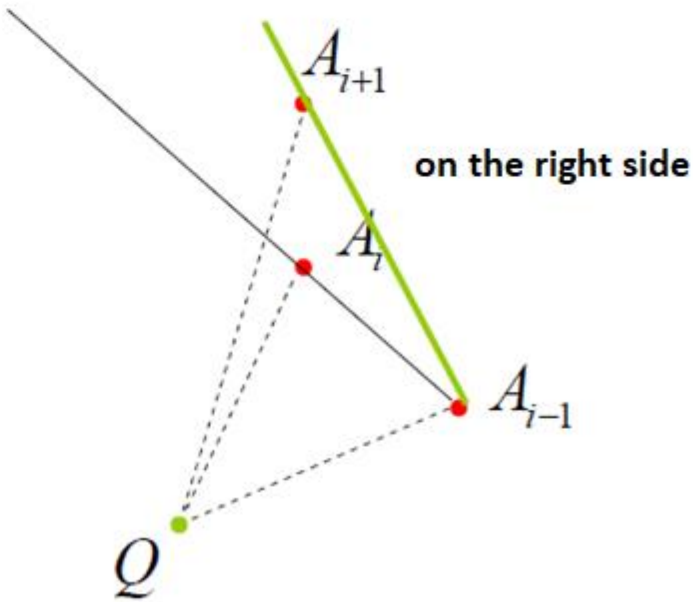
Similar approach

- For each triplet A_{i-1}, A_i, A_{i+1} we have to determine if A_{i+1} lies on the **left side** or on the **right side** from A_{i-1} A_i



- We replace A_{i-1}, A_i, A_{i+1} by their successors

Similar approach

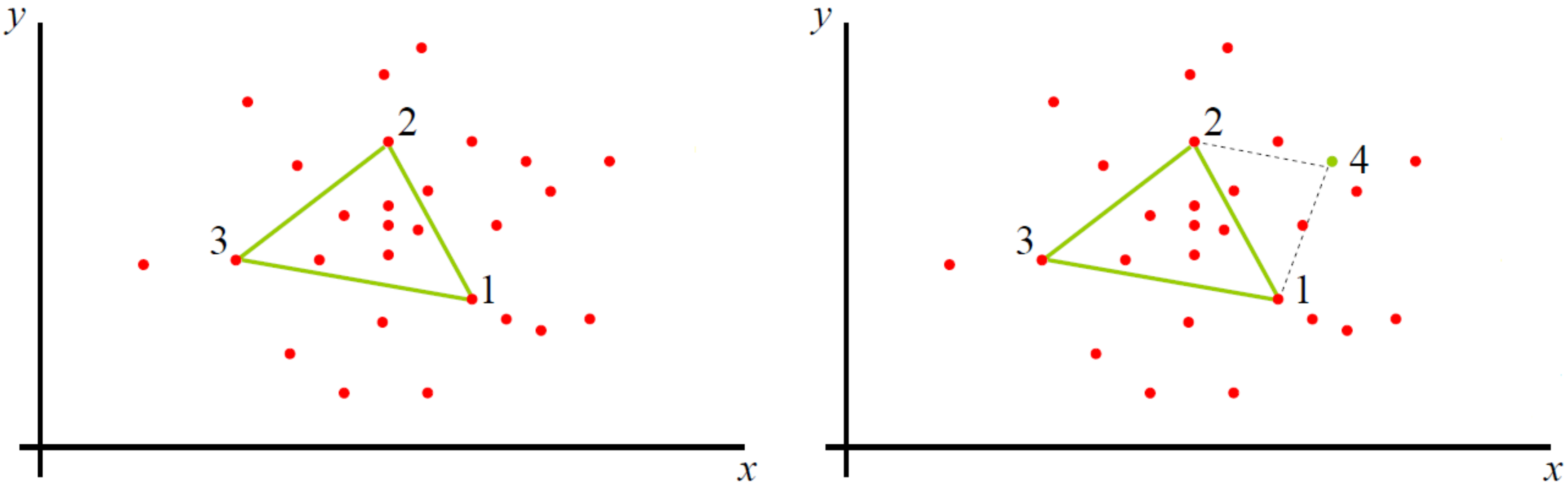


- We replace A_i by A_{i+1} and delete A_i

Incremental algorithm

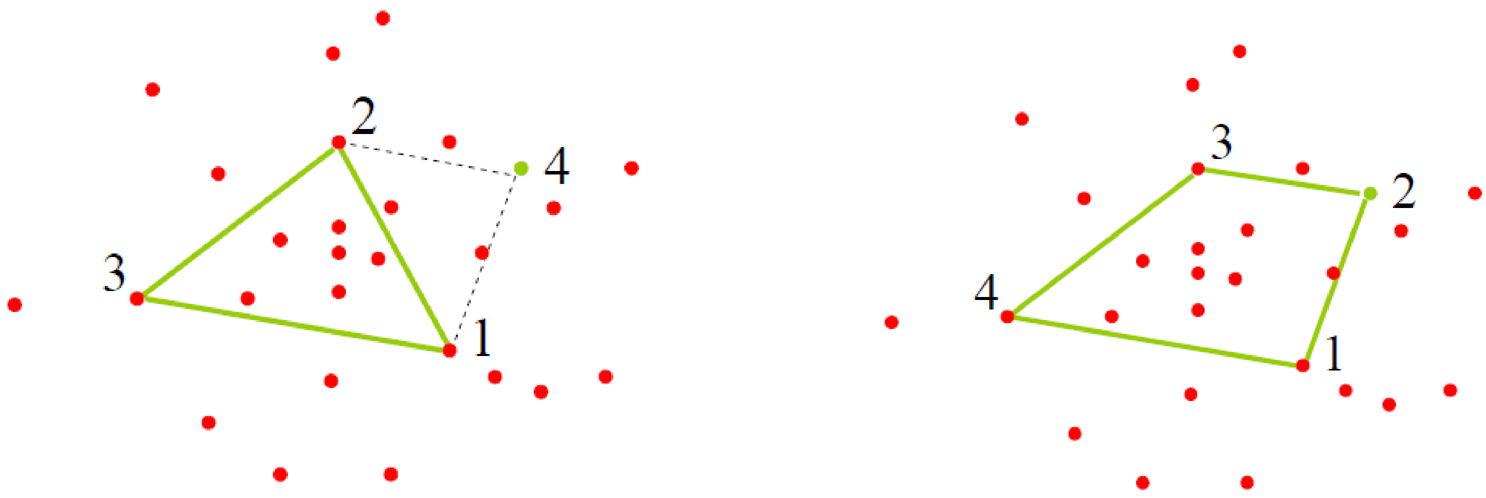
- We select arbitrary three points forming a triangle (these will form an initial convex hull), we sort them counter-clockwise
- We find a set of so-called **inner points** (i.e., points lying outside this triangle)
- An arbitrary point from the inner points is added to the convex hull

Incremental algorithm



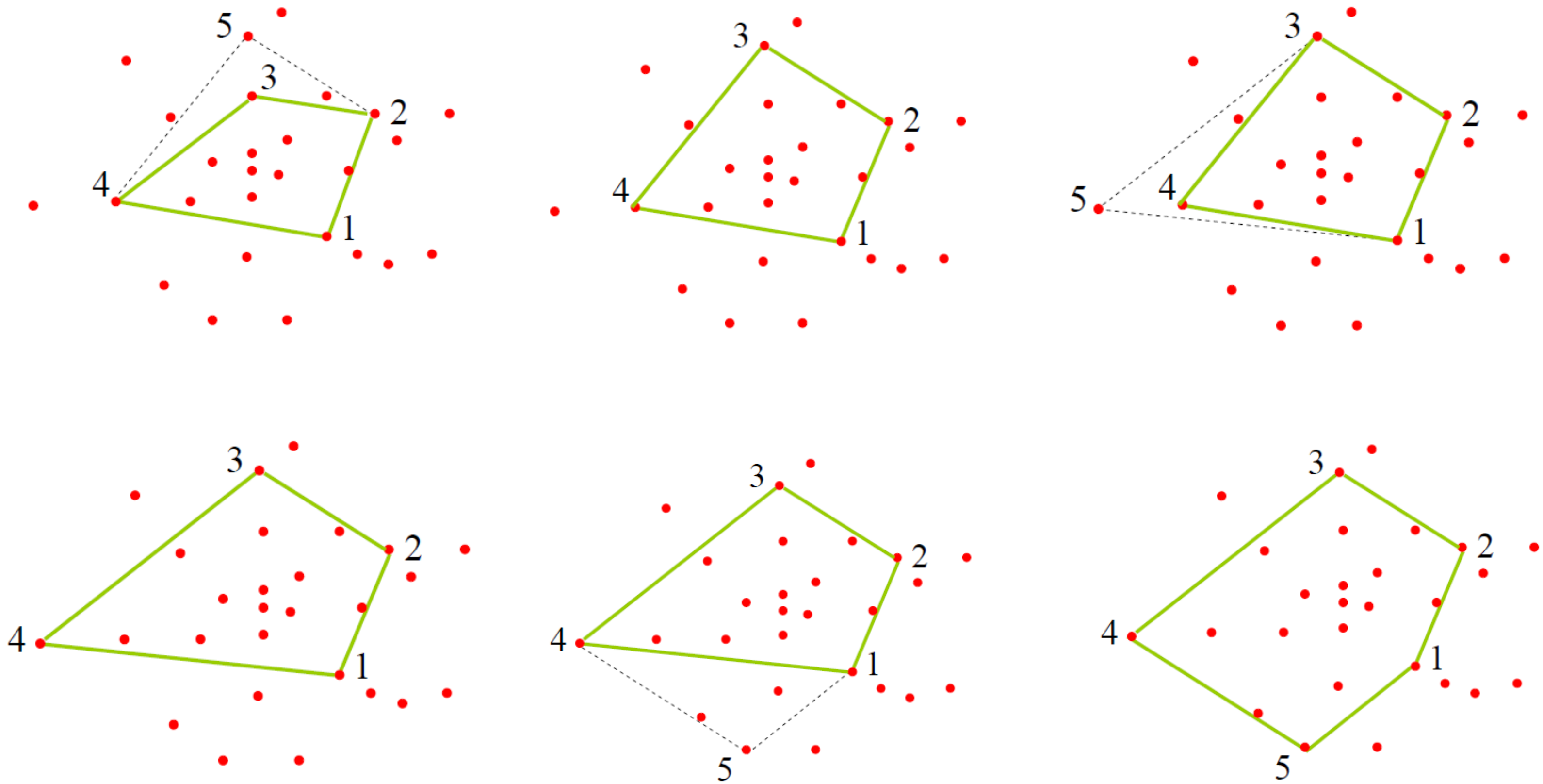
Incremental algorithm

- We have to find out (e.g., using determinant or Half Edge test) edges which are visible from the currently added point and those have to be removed from the convex hull

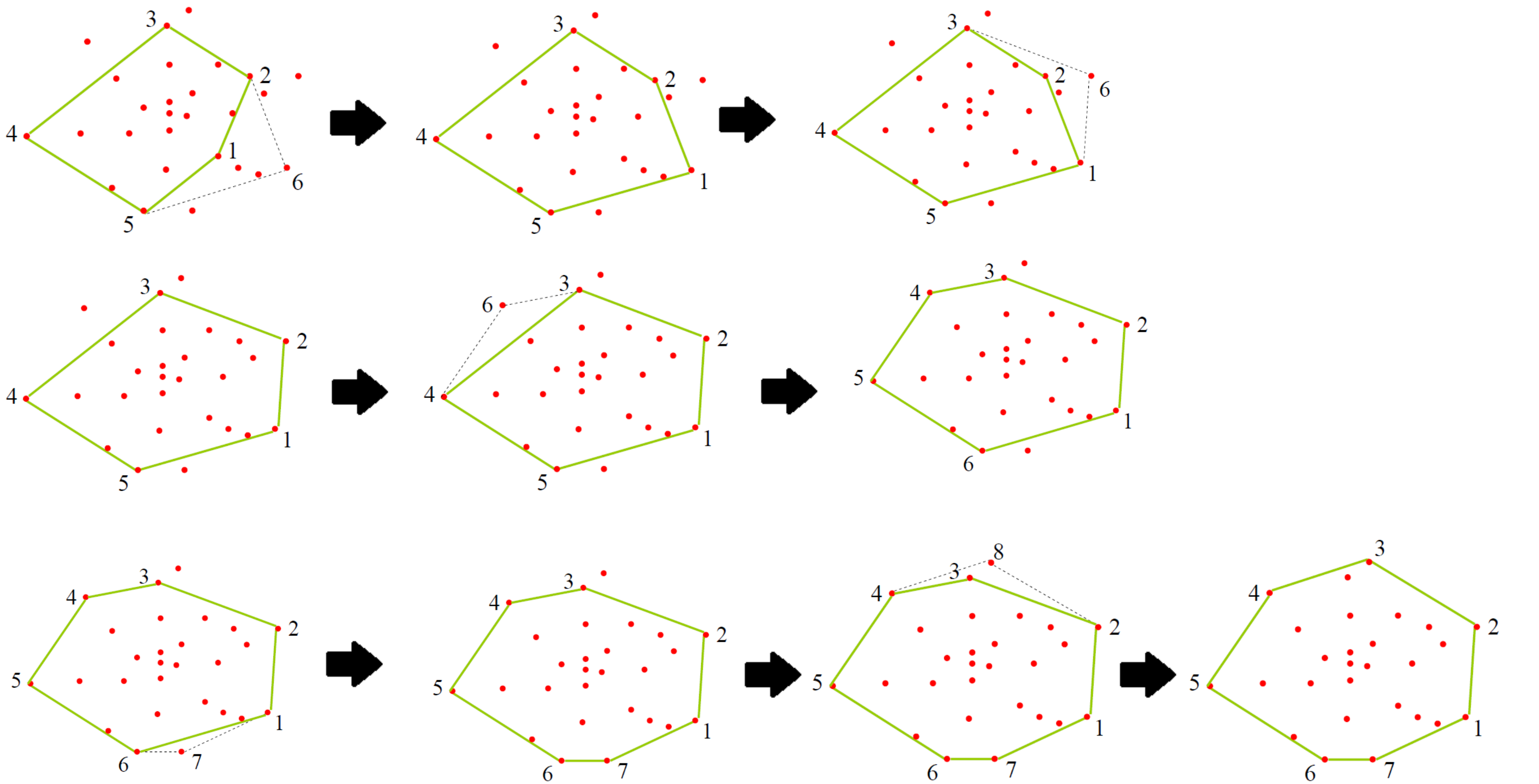


Incremental algorithm

- This is repeated until the inner points set is empty



Incremental algorithm



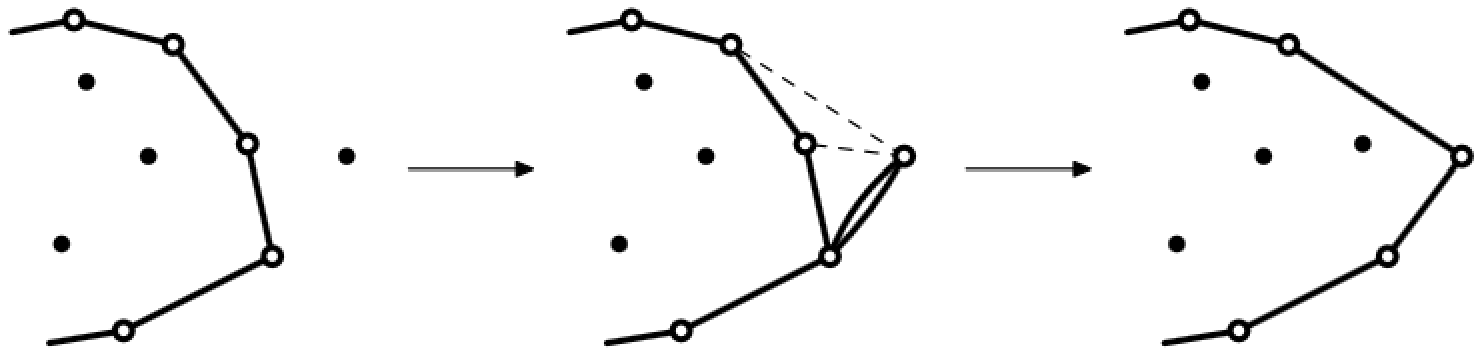
Incremental algorithm – sweep plane

- For simplicity, lets assume that all points have different x-axis values
- This incremental approach first sorts all points in the ascending order, according to their x-axis value. Then it traverses this list “from left to right” and incrementally constructs the convex hull

Incremental algorithm – sweep plane

- Adding a next point to the convex hull:
  - From the current convex hull we select the “rightest” point and we connect it with the newly adding point from the list. This results in a non-convex hull which has to be corrected.
  - This correction removes points in both directions along the previous convex hull until we reach new correct convex hull

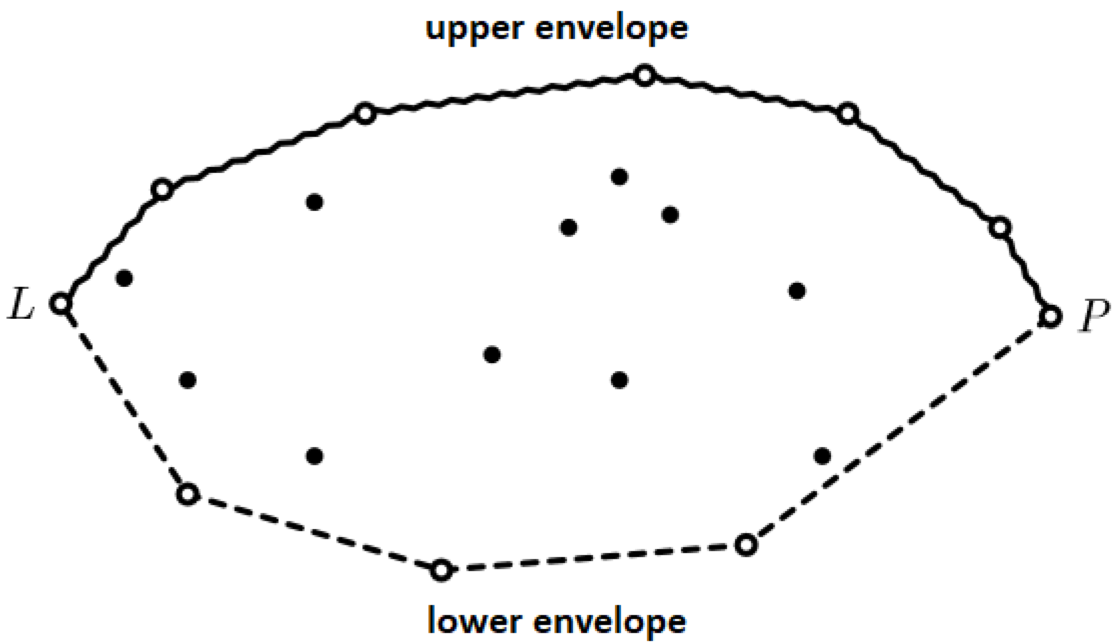
Incremental algorithm – sweep plane



- In this case we don't have to remove any point in the clockwise order, but we have to remove points in the counter-clockwise order

Implementation

- For better understanding and implementation, we divide the convex hull to two parts, **upper** and **lower envelope**, separated by the most left and most right points L and P



Implementation

- Both envelopes are formed by polylines, the upper one always turning to the right, lower to the left
- We need two stacks to keep the envelope points
- In the *k*-th step of the algorithm we add the *k*-th point to both upper and lower envelope and then we have to solve potential problems with the change of turn of such updated envelopes. So we will first remove the points from the envelope and the *k*-th point will be added only when it doesn’t change the turn of the envelope

Pseudocode

1. Sort points according to their *x*-axis value, mark them as $b_1, \ldots, b_n$
2. Insert point b_1: $H = D = (b_1)$ to the upper and lower envelope
3. For each point $b = b_2, \ldots, b_n$:
    1. Recompute the upper envelope:
        1. Until $|H| \geq 2$, $H = (\ldots, h_{k-1}, h_k)$ and angle $h_{k-1}h_kb$ is oriented to the left:
            1. Remove the last point h_k from envelope H
        2. Add point b to envelope H
    2. Symmetrically for the lower envelope (with orientation to the right)
4. Resulting hull is formed by points in H and D

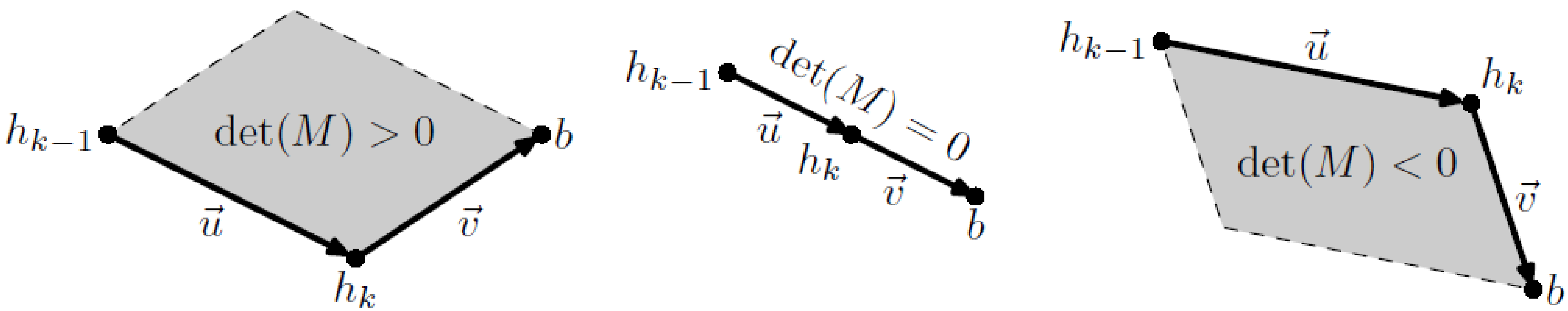
Angle orientation from determinant

- Lets assume a classical coordinate system in 2D, we want to determine the orientation of angle $h_{k-1}h_k b$
  - We define vector $u = (x_1, y_1)$ as a difference between coordinates of h_k and h_{k-1} and a vector $v = (x_2, y_2)$ as a difference between coordinates of b and h_k
  - Matrix M is defined as

$$M = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix}$$

Angle orientation from determinant

- Angle is **left-oriented** when *det* $M = x_1y_2 - x_2y_1$ je **non-negative**



Divide and conquer

- Complexity *O(n log(n))*
- More complex implementation
- Principle:
  - Dividing the input set *S* into two subsets S_1 and S_2 of the same size, processing them separately
  - Both solutions are subsequently merged using upper and lower common tangents t_1 and t_2 – merging takes *O(n)*

Divide and conquer

- Two subsets can be further divided until the solution is geometrically trivial (triplet of points forming a triangle)
- We demonstrate the pseudocode on two subsets *A, B*
  – We assume that any three points lie on a line and any two points share the same x-coordinate value

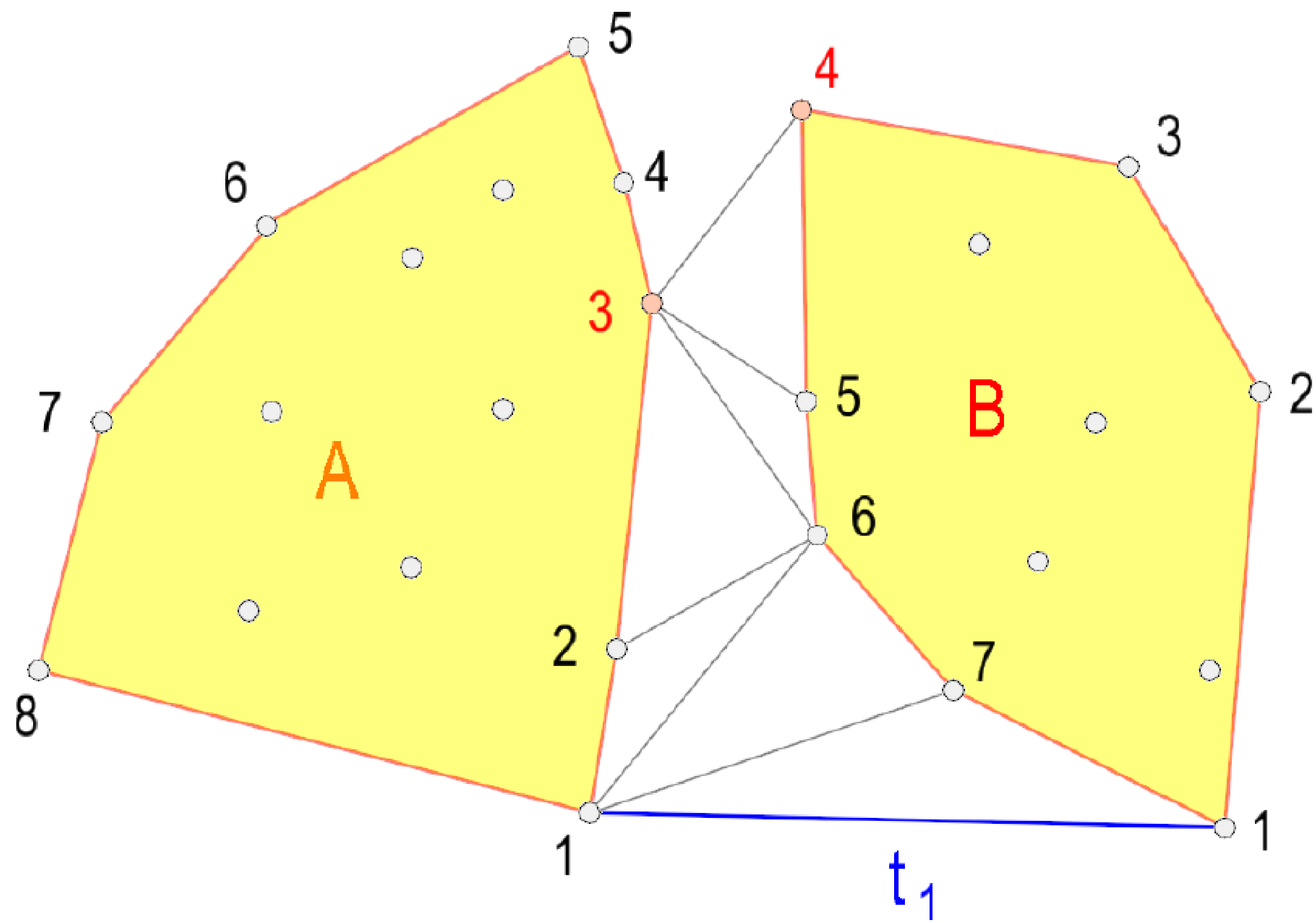
Pseudocode

1. Sort *S* according to *x* value
2. Divide *S* to two subsets *A, B*, each containing *n/2* points
3. Construct convex hulls *H(A)* and *H(B)*
4. Merge convex hulls *H = H(A)* U *H(B)*
   1. Find lower tangent t_1
   2. Find upper tangent t_2
   3. Replace segments of hulls *A, B* between these tangents

Finding the lower tangent

- From the extreme points *a, b* of *A, B* we find a corresponding point *b* to a given point *a* whose connecting line forms the lower tangent of subset *B*
- From point *b* we search for such a point *a* whose connecting line forms the lower tangent of subset *A*
- We repeat this until *a, b* is not the tangent of both *A* and *B*

Finding the lower tangent



Finding the lower tangent

1. Search for point a = the most right point of A
2. Search for point b = the most left point of B
3. Repeat until $t_1 = ab$ is not a lower tangent of A anf B
   1. Repeat until t_1 is not the lower tangent of A
      1. $a = a - 1$
   2. Repeat until t_1 is not the lower tangent of B
      1. $b = b + 1$

Finding the upper tangent

- Symmetric…

TASK 3

- Implement the Graham-Scan algorithm (according to your preference ☺)