CS271 Computer Graphics II

Lecture 2

Computational Geometry – Convex Hull

Overview

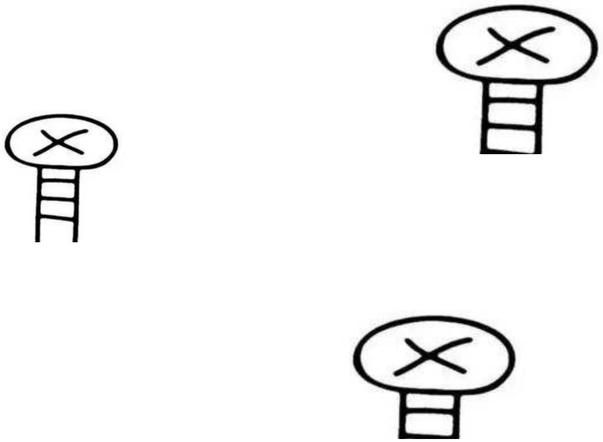
- Basics
- 2D convex hull
- 3D convex hull
- Convex hull of simple polygon
- Approximated convex hull

Warm up

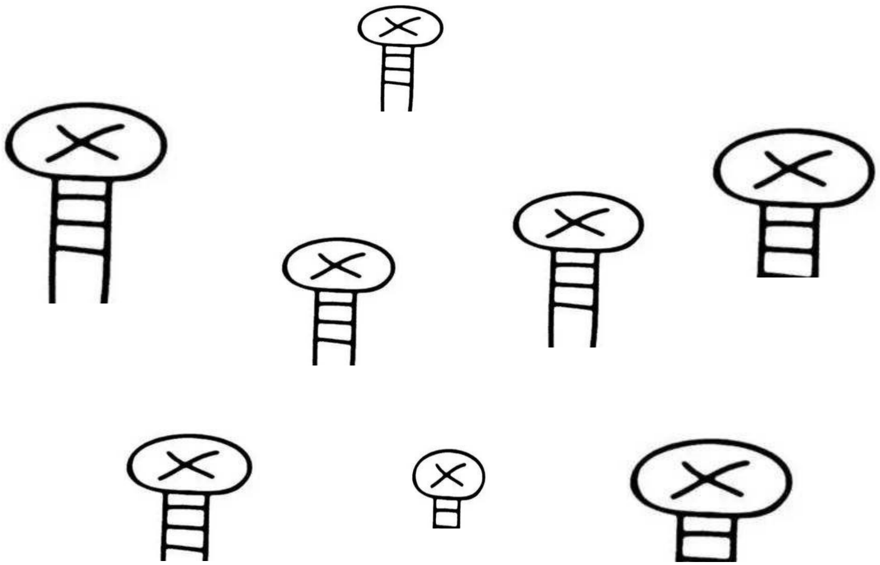
1. Build a warehouse
2. Sort
3. Color modulation
4. Postman problem

Trap all the nails

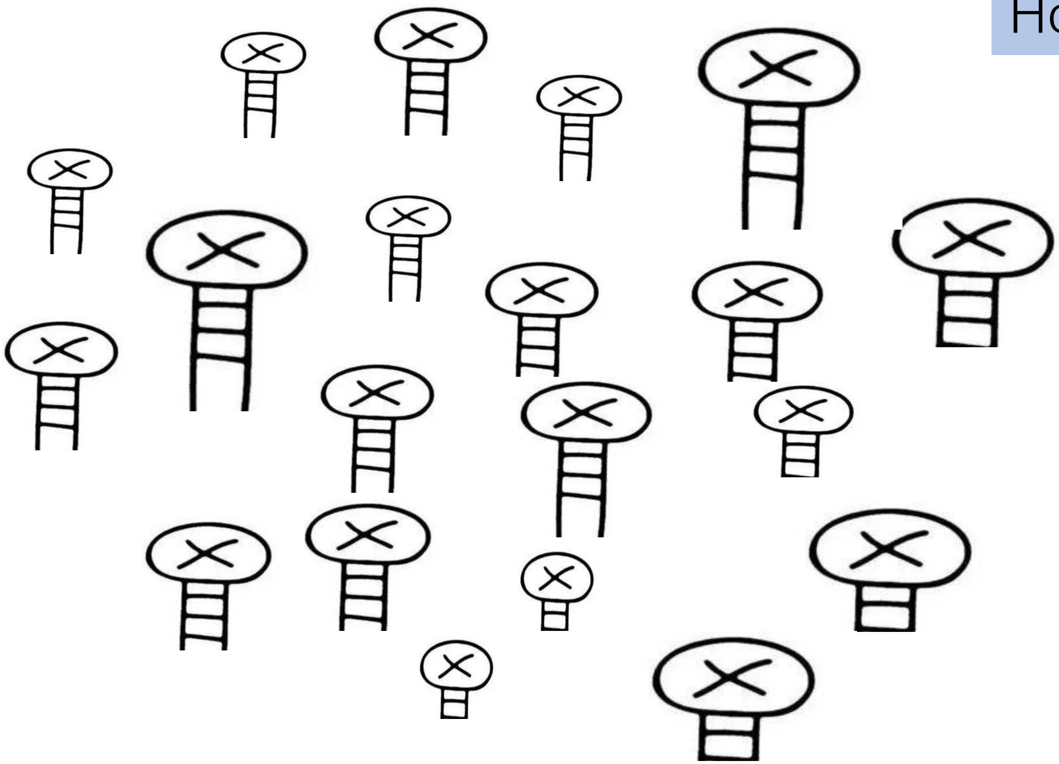
Trap all the nails



Trap all the nails



Trap all the nails



How could the computer calculate?

Basics

- E^d denotes d dimensional Euclidean space, and $p = (x_1, x_2, \ldots, x_d)$ is one point in the space. For any p_1 and p_2, we can use their linear combination as one **line** in E^d :

$$\alpha p_1 + (1 - \alpha) p_2, \alpha \in R.$$

  If $0 \leq \alpha \leq 1$, it denotes the **line segment** linking p_1 and p_2.

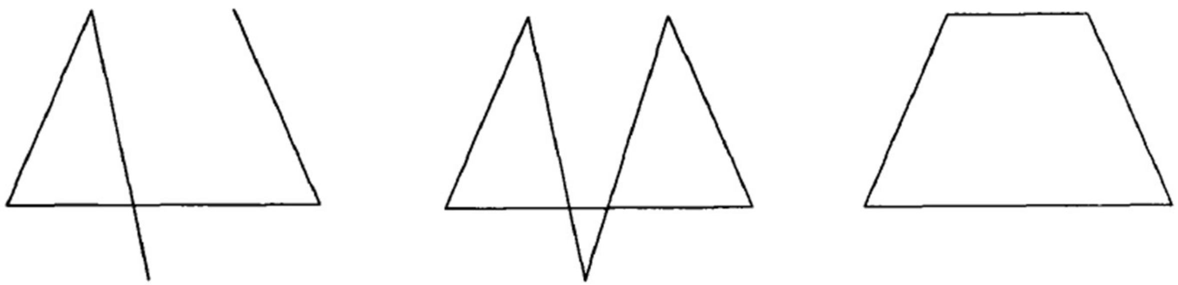
- The ***Convex Set*** for k points $p_1, p_2, \ldots, p_k$ in E^d is denoted as

$$\sum_{i=1}^{k} \alpha_i p_i. \quad (\alpha_i \geq 0, \sum_{i=1}^{k} \alpha_i = 1)$$

For example, the convex set of three points that are not collinear on a plane compose a triangle.

Polygon

- A polygon in E^2 is defined by a set of line segments, called edge, connected to form a closed polygonal chain.
- For each terminal of any edge, it is and only is the public terminal of two edges of the polygon.
- If any two edges are not intersected, the polygon is a **simple polygon**.



Planar Graph

A **planar graph** is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints and no edges cross each other.

- It meets ***Euler's formula***:

$$\boldsymbol{v - e + f = 2},$$

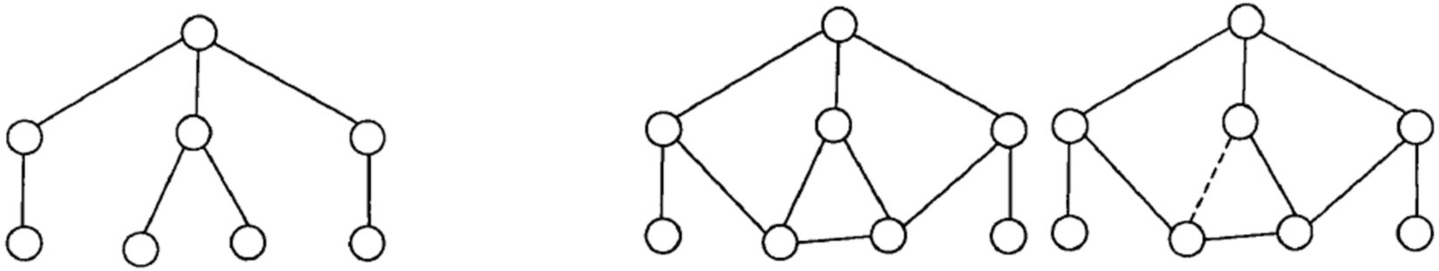
where v, e and f denotes the number of vertices, edges, and areas in the graph, respectively.

Euler's formula

Prove Euler's formula $\boldsymbol{v - e + f = 2}$.

Use Mathematical Induction:

1. Prove that the statement holds for $f = 1$.
2. Prove that if the statement holds for $f - 1$, then it holds for f.



(a) Planer graph without cycle (tree) (b) Planer graph with cycle (c) Delete one edge of (b)

$v = e + 1$

Basics

According to the Euler's formula, we can also prove the following formulas.

- If the degree of each vertex is at least 3, then we have

$$v \leq \frac{2e}{3},\ e \leq 3f - 6, v \leq 2f - 4$$

- If each area is enclosed by at least 3 edges, then we have

$$f \leq \frac{2e}{3}, e \leq 3v - 6, f \leq 2v - 4$$

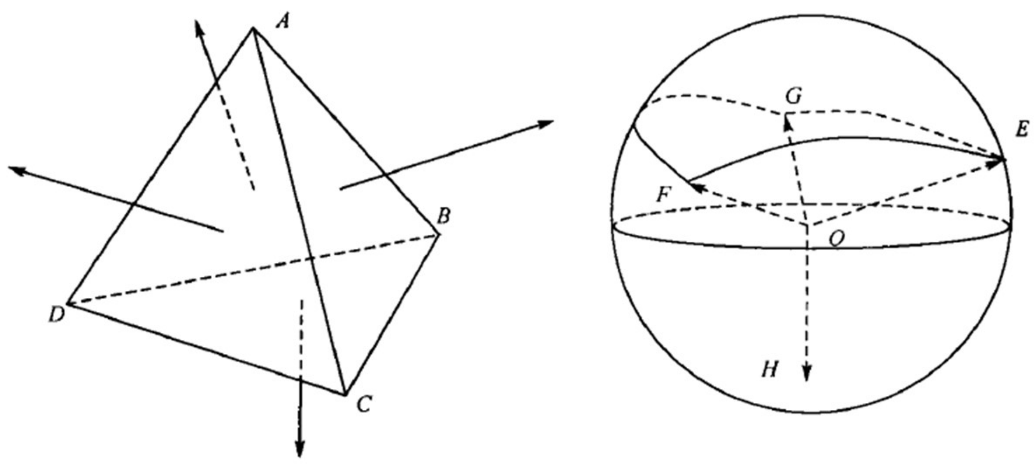
Geometry duality

Think about the cylinder and cone.

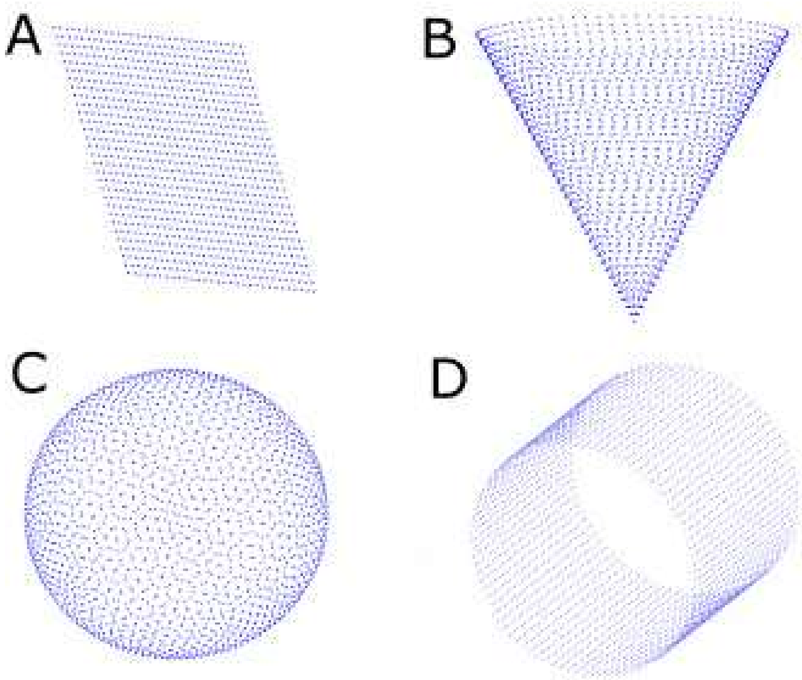
- We usually construct duality between two geometric objects and use the properties in the duality space to solve the problems in the original space.

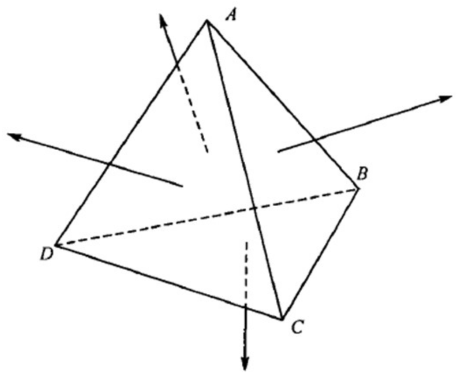
Gaussian Sphere is a unit sphere.

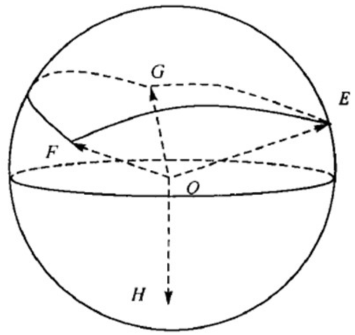
There are intersection points by moving the origins of the outward normal vector of every facet of the polyhedron to the Gaussian sphere center. These intersection points and the facets of the polygon are dual.



Geometry duality







Algorithm complexity

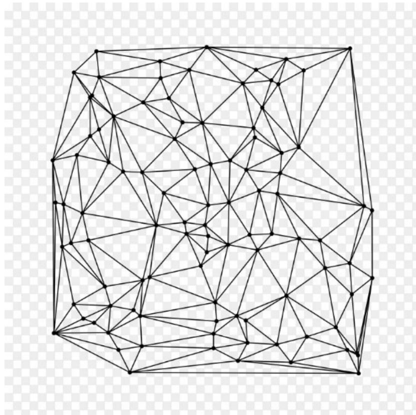
- In general, computational geometry has to solve large-scale problems.
- It is necessary to consider the memory and run time.

- The lower bound of the time complexity of sorting algorithms in worst case is

$$O(nlog(n))$$

Exercise

1. Prove the intersection of two convex set is still a convex set.
2. If a plane is divided into polygons by line segments, please design a data structure to store the division information so that for the given line passing two points p_1 and p_2 on the plane, it is efficient to find all the polygons intersected with the line. Please state the advantage and disadvantage of the data structure.



Convex Hull

- The ***Convex Set / Convex Combination*** for point set S $(p_1, p_2, \ldots, p_n)$ is denoted as

$$\sum_{i=1}^{n} \alpha_i p_i. \quad (\alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i = 1)$$

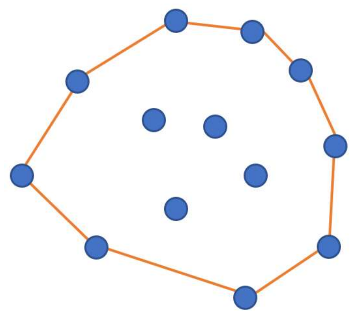
- The ***convex hull*** or ***convex envelope*** or ***convex closure*** of S is the **union** of all the convex combinations of S.
- **Other definitions:**

➢The intersection of all convex sets containing S.

➢The intersection of all half space containing S.

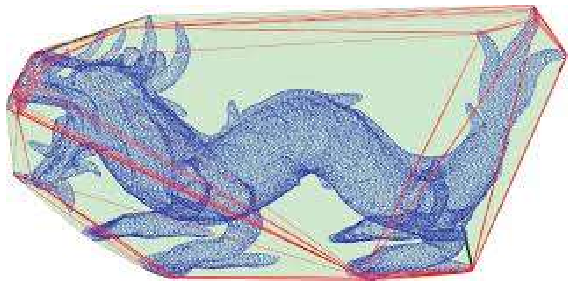
➢In 2D, the convex polygon containing S with smallest area.

➢In 2D, the convex polygon containing S with smallest perimeter.



Convex Hull

- Fast collision queries (games, robots…)
- Shape matching (convex deficiency trees, similarity analysis…)
- Serve for other geometric construction (Voronoi…)
- Many applications (building fence for a castle…)



2D convex hull

- Extreme edges algorithm
- Gift wrapping algorithm
- Quick hull algorithm
- Graham algorithm
- Incremental algorithm
- Divide and conquer algorithm

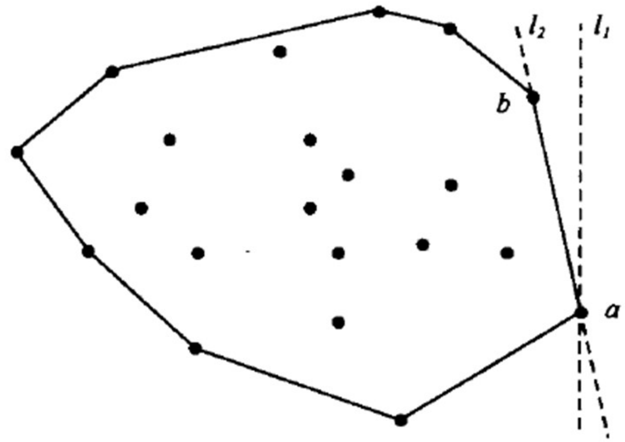
Extreme edges algorithm

- **Extreme points** (the vertices of convex hull)

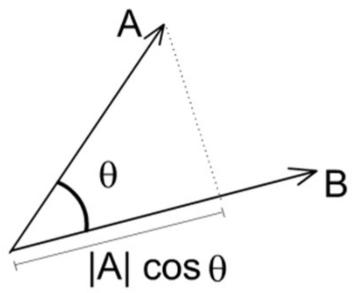
- **Extreme edges** (All the points of S are on one side of or just on the line passing the edge, like edge ab.)

- **Extreme edges algorithm**

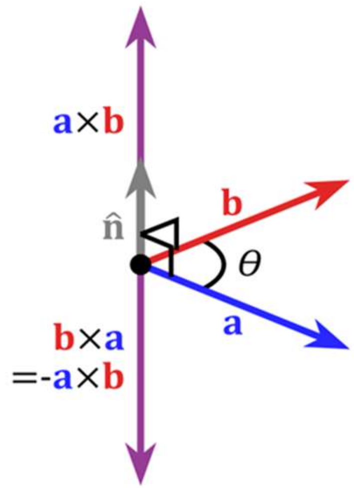
  ➡ Find all the extreme edges.



- Dot product

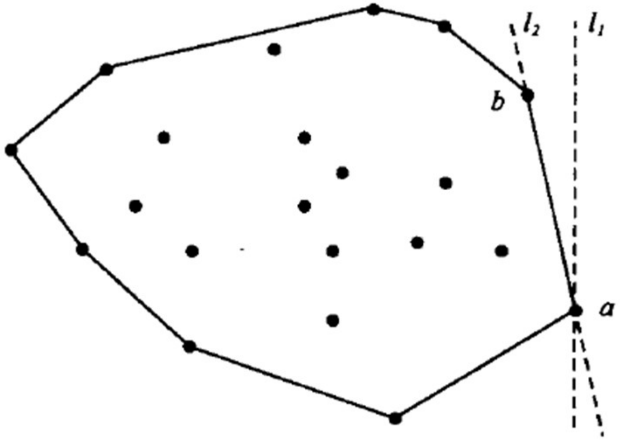


- Cross product



Extreme edges algorithm

- ***Input:*** 2D point set S
- ***Output:*** all the edges of the convex hull of S



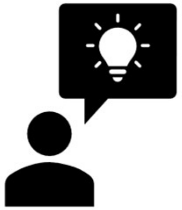

```
for each p_i in S
    for each p_j in S and i ≠ j
        for each p_k in S and k ≠ i and k ≠ j
            if ( p_k is not on the left side of p_i p_j )
                mark p_i p_j as deleted edge; // p_i p_j is not the extreme edge
```

The time complexity is $O(n^3)$ and the output is not in order.

Gift wrapping algorithm

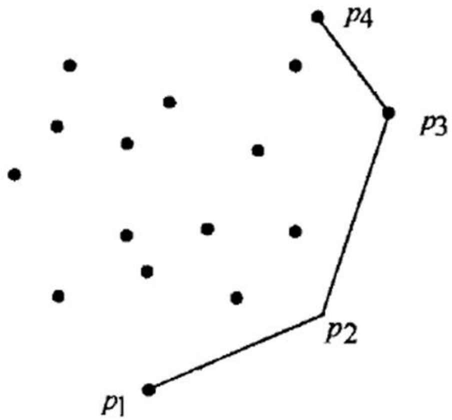
- How to find the next extreme edge according to the last one in $O(n)$?

- Assuming p_1p_2 is one extreme edge and p_2 is taken as the end point, how could we find the next extreme edge p_2p_3?



Search for the next point p_3 so that all other points are on the left side of p_2p_3.

$\Rightarrow$ $O(n^2)$



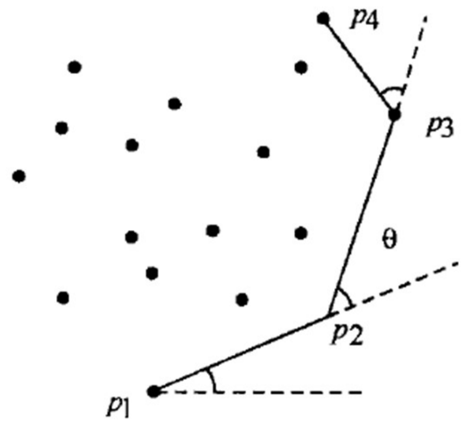
Gift wrapping algorithm

The angle formed by the extension line of p_1p_2 and p_2p_3 is the smallest.

- For each point p_i, compute the $\theta = 180^{\circ} - \angle p_1p_2p_3$。
- The point with the smallest θ is p_3. p_2p_3 is the next extreme edge.

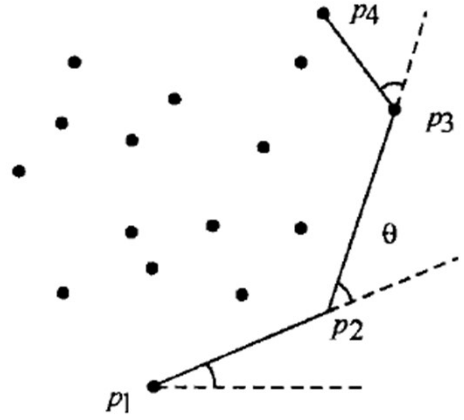
$O(n)$

The total time complexity is $O(nh)$, where h denotes the number of extreme edges of the convex hull. In worst cases, $h = O(n)$, and the time complexity of the algorithm is $O(n^2)$.



Gift wrapping algorithm

How to find the initial edge p_1p_2?



Gift wrapping algorithm

- ***Input:*** 2D point set S
- ***Output:*** the convex hull of S

1. Find the lowest point in S, i.e. the point with smallest y value. If there exist many such points, pick the point with maximal x value among them.
2. Assume the index of the point is i_0, $i = i_0$;
3. Take the parallel line of x axis passing p_i as one extreme edge.
4. Repeat{
5. $\quad\theta_i = \pi$;
6. $\quad$for each $j \neq i$ do
7. $\quad\quad$if $(\theta_i > \theta_j)$ $\{\theta_i = \theta_j; k = j;\}$
8. $\quad$output $p_i p_k$ as one extreme edge;
9. $\quad i = k$;
10. } until $(i == i_0)$

Quick hull algorithm

- Similar to Quicksort algorithm.

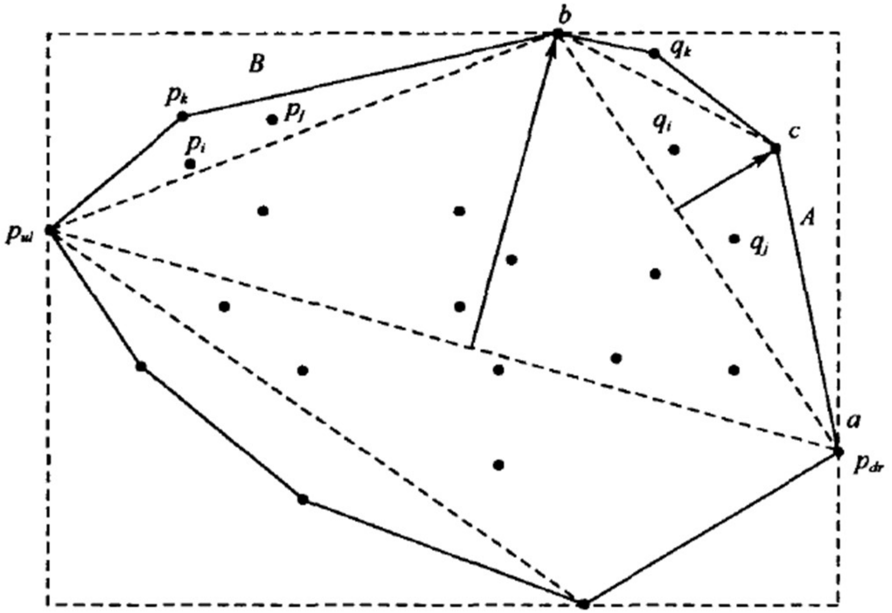
It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. Recursively apply the above steps to the sub-array of elements.

6 5 3 1 8 7 2 4

Quick hull algorithm

- **Main idea**

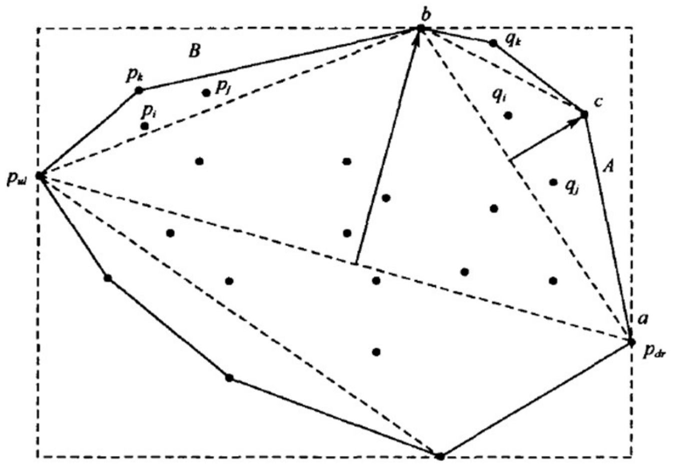
The convex hull is decided by the points nearby the boundary. If we filter inner points step by step and focus on the boundary points, we can improve the efficiency.



Link the most right-down point p_{dr} and the most left-up point p_{ul}. The convex hull is divided into right and left convex hull, which can be gotten by recursion.

Quick hull algorithm

- ***Input:*** 2D point set S
- ***Output:*** the convex hull of S in counterclockwise order



1. Find the most right-down point p_{dr} and the most left-up point p_{ul} in S.
2. Divide S into two subsets S_1 and S_2. The points in S_1 are on the right of $p_{dr}p_{ul}$, and the points in S_2 are on the left of $p_{dr}p_{ul}$.
3. return $((p_{dr}) + \text{HullLoop}(p_{dr}, p_{ul}, S_1) + (p_{ul}) + \text{HullLoop}(p_{ul}, p_{dr}, S_2))$

Function *HullLoop*($\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{M}$)

- ***Input:*** extreme points a and b, the point subset M on the right of ab
- ***Output:*** a link of the convex hull of M from $a\ to\ b$

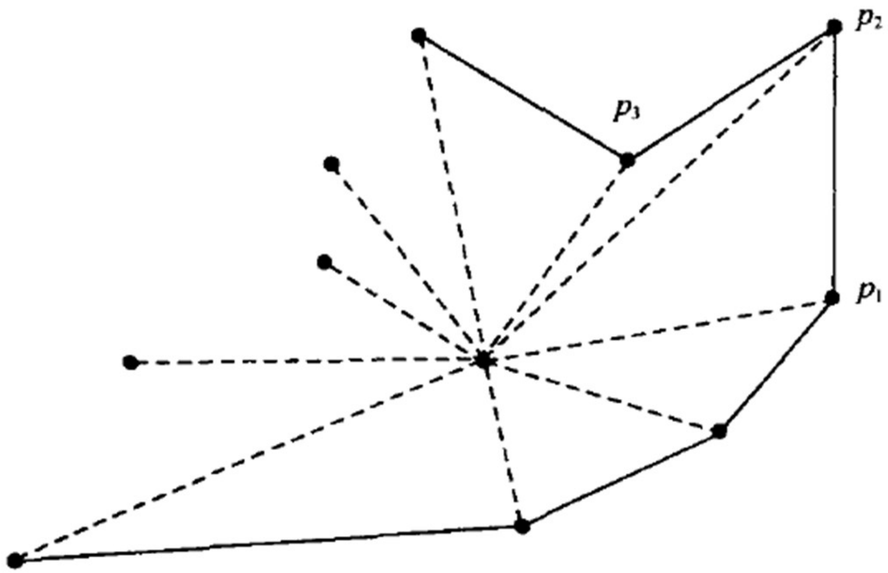
If $M = \emptyset$ then return();
else{
    c = the farthest point to the line ab;
    A= the subset of M locating on the right of ac;
    B = the subset of M locating on the right of cb;
    return ($HullLoop(a, c, A)$+(c)+ $HullLoop(c, b, B)$);
}

HullLoop ($\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{M}$)

1. If $M = \emptyset$ then return();
2. else{
3. $\quad c =$ the farthest point to the line ab;
4. $\quad A=$ the subset of M locating on the right of ac;
5. $\quad B =$ the subset of M locating on the right of cb;
6. $\quad$ return (*HullLoop*(a, c, A)+(c)+ *HullLoop*(c, b, B));
7. }

In best cases, $|A| = |B| = n/2$, the time complexity is $O(nlogn)$.
In worst cases, $|A|$ =0 and $|B| = n$-1, the time complexity is $O(n^2)$.

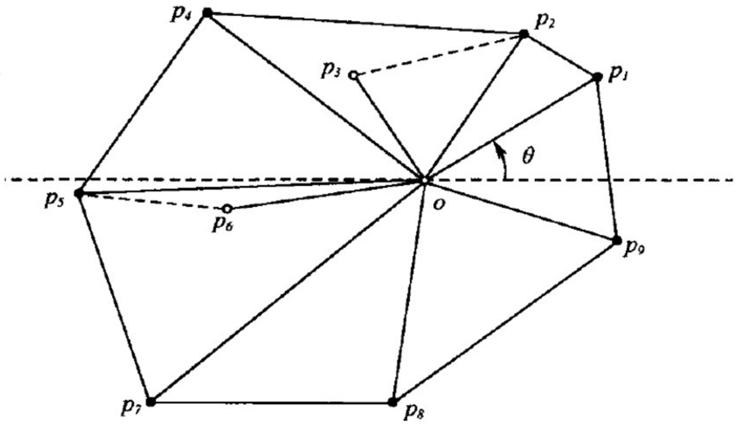
Graham algorithm



Graham algorithm

- ***Input:*** 2D point set S
- ***Output:*** the convex hull of S in counterclockwise order

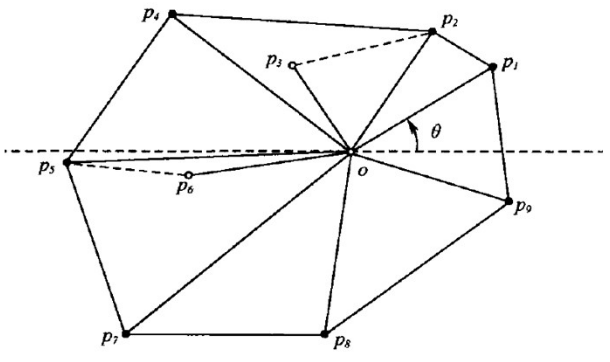
1. Find an inner point o in S and link it with all other points;
2. Sort all other points according to θ (the counterclockwise angle from x axis to op_i) and mark them as $p_1, p_2, \ldots, p_n$;
3. For the polygon ($p_1, p_2, \ldots, p_n$), delete all concave vertices.



Graham algorithm

1. Find an inner point o in S and link it with all other points;
2. Sort all other points according to θ (the counterclockwise angle from x axis to op_i) and mark them as $p_1, p_2, \ldots, p_n$;
3. Initialize stack C and push (p_1, p_2) in C. // t is the index of the top of C
4. i=3;
5. while i<n+1 do
6. if (p_i is on the left of $p_{t-1}p_t$)
7. *push* (p_i , C) and i=i+1
8. else *pop*(C)

$p_{t-1}p_t \times p_tp_i > 0$



Graham algorithm

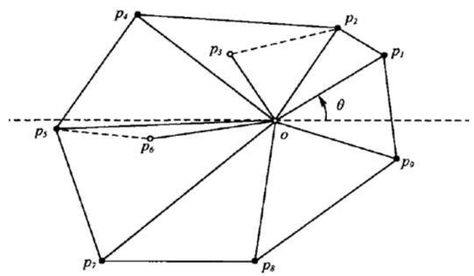
1. Find an inner point o in S and link it with all other points;
2. Sort all other points according to θ (the co $O(nlogn)$ ngle from x axis to op_i) and mark them as $p_1, p_2, \ldots, p_n$;
3. Initialize stack C and push (p_1, p_2) in C. // t is the index of the top of C
4. i=3;
5. while i<n+1 do
6. if (p_i is on the left of $p_{t-1}p_t$)
7. *push* (p_i , C) and i=i+1
8. else *pop*(C)

$O(n)$

$totally\ in\ O(nlogn)$

Graham algorithm

- How to make sure that p_0p_1 is extreme edge?
- How to deal with the case more than two points are on the same line?
- How to make the algorithm more robust?
- Is $O(nlogn)$ the best time complexity for convex hull computing? And how to prove your idea?
- Could Graham algorithm be extended to 3D space?



Incremental algorithm

How could we add points one by one and construct convex hull accordingly?

Incremental algorithm

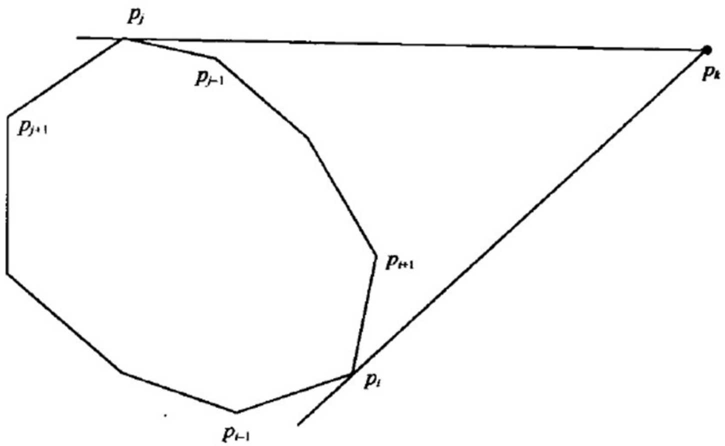
- ***Input:*** 2D point set S (assume any three points are not on a line)
- ***Output:*** the convex hull of S

1. $H_3 \leftarrow conv(p_1, p_2, p_3)$; // compute initial convex hull
2. for $k = 4$ to n do
3. $\quad H_k \leftarrow conv(H_{k-1}, p_K)$

$conv(H_{k-1}, p_K)$

1. $p_k \in H_{k-1}$

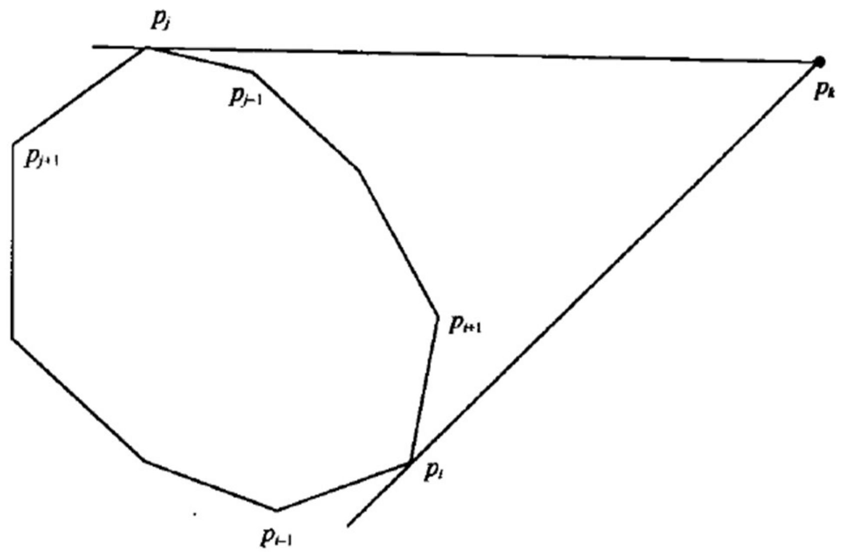
- Decided by checking whether p_k locates on the left side of all extreme edges.
- $H_k = H_{k-1}$



2. $p_k \notin H_{k-1}$

- Find two tangent points (p_i, p_j) according to p_k.
- Delete the inner linked link and insert p_k between (p_i, p_j).

TangentPoint ($\boldsymbol{p_k}, \boldsymbol{H_{k-1}}$)



- ***Input:*** $\boldsymbol{p_k}, \boldsymbol{H_{k-1}}$
- ***Output:*** the tangent points of $\boldsymbol{p_k}$ and $\boldsymbol{H_{k-1}}$

1. **for** i=1 **to** the number of vertices of H_{k-1}
2. **if** $(lefton(p_k, p_{i-1}, p_i) \geq 0\ \delta\delta\ lefton(p_k, p_i, p_{i+1}) \leq 0)$;
3. output the left tangent point p_i;
4. **if** $(lefton(p_k, p_{i-1}, p_i) \leq 0\ \delta\delta\ lefton(p_k, p_i, p_{i+1}) \geq 0)$;
5. output the right tangent point p_i;

Incremental algorithm

- ***Input:*** 2D point set S (assume any three points are not on a line)
- ***Output:*** the convex hull of S

1. $H_3 \leftarrow conv(p_1, p_2, p_3)$; // compute initial convex hull
2. for $k = 4$ to n do
3. $\quad H_k \leftarrow conv(H_{k-1}, p_K)$ $O(n)$

$$3 + 4 + \cdots + n = O(n^2)$$

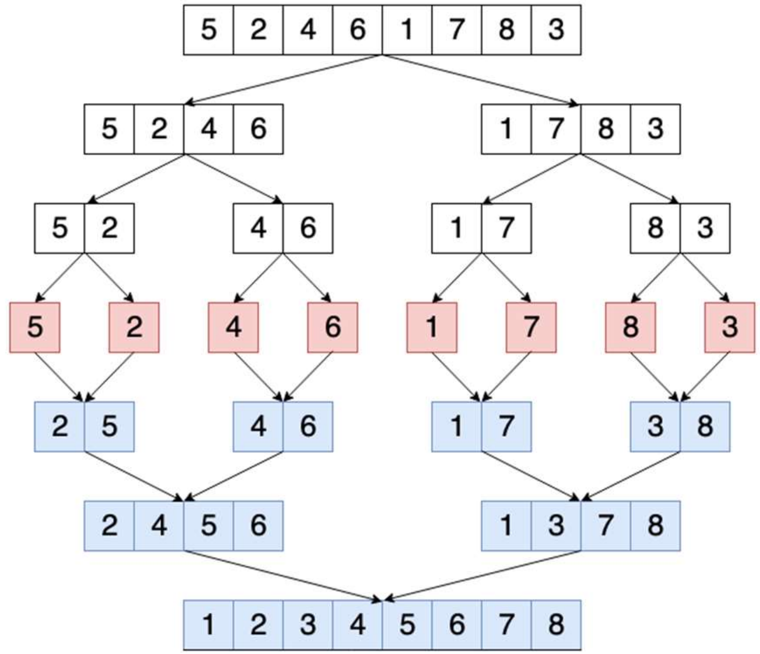
Divide and conquer algorithm

- Similar to MergeSort algorithm

Divide

Solve

Merge



$O(nlogn)$

Divide and conquer algorithm

- ***Input:*** 2D point set S
- ***Output:*** the convex hull of S

1. Sort all the points in S according to x coordinates;
2. Divide S into two subsets A and B, where A contains $ceil(n/2)$ points, B contains $floor(n/2)$ points.
3. Recursively compute the convex hull of A and B, $conv(A)$ and $conv(B)$.
4. Merge $conv(A)$ and $conv(B)$ and get the convex hull of S.

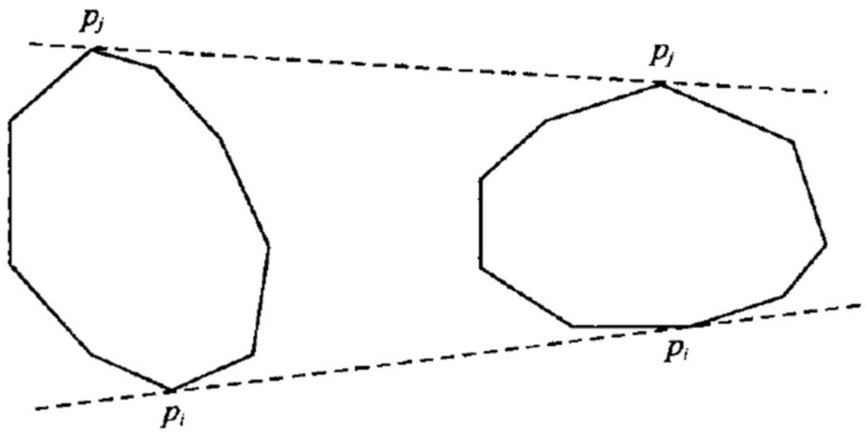
Divide and conquer algorithm

3. Recursively compute the convex hull of A and B, $conv(A)$ and $conv(B)$.

   When $n \leq 3$, stop recursion and construct a triangle.

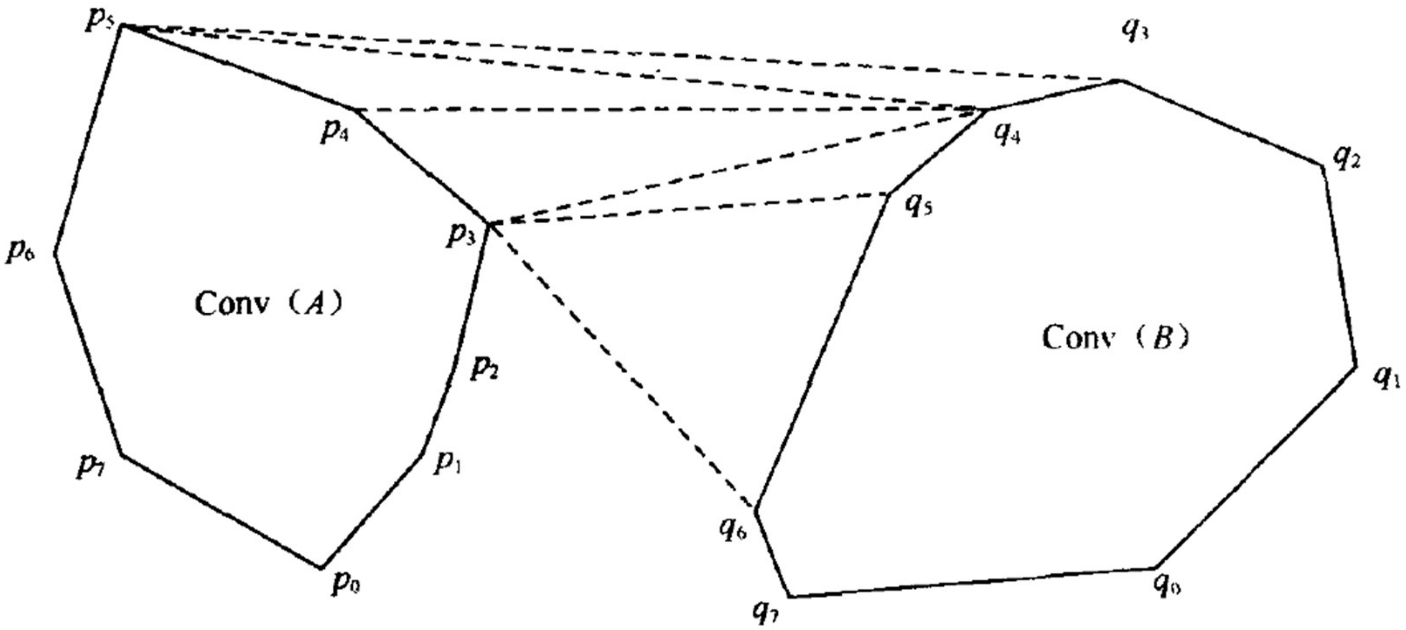
4. Merge $conv(A)$ and $conv(B)$ and get the convex hull of S.

   To make sure the total time complexity is $O(nlogn)$, this step should be $O(n)$.



$Up\_Tangent(conv(A), conv(B))$

- ***Input:*** $conv(A)$ and $conv(B)$ (counterclockwise sorted)
- ***Output:*** point a of $conv(A)$ and point b of $conv(B)$, where ab is the up tangent line of $conv(A)$ and $conv(B)$.



$Up\_Tangent(conv(A), conv(B))$

- ***Input:*** $conv(A)$ and $conv(B)$ (counterclockwise sorted)
- ***Output:*** point a of $conv(A)$ and point b of $conv(B)$, where ab is the up tangent line of $conv(A)$ and $conv(B)$.

1. a is the rightmost point of $conv(A)$;
2. b is the leftmost point of $conv(B)$;
3. while $T = ab$ is not the up tangent line of $conv(A)$ and $conv(B)$
4. { while T is not not the up tangent line of $conv(A)$ do
5. $a = a + 1;$
6. while T is not not the up tangent line of $conv(B)$ do
7. $b = b - 1;$
8. }

$$O(|A| + |B|) = O(n)$$

Divide and conquer algorithm

- ***Input:*** 2D point set S
- ***Output:*** the convex hull of S

1. Sort all the points in S according to x coordinates;
2. Divide S into two subsets A and B, where A contains $ceil(n/2)$ points, B contains $floor(n/2)$ points.
3. Recursively compute the convex hull of A and B, $conv(A)$ and $conv(B)$.
4. Merge $conv(A)$ and $conv(B)$ and get the convex hull of S.

$$O(nlogn)$$

2D convex hull

- Extreme edges algorithm
- Gift wrapping algorithm
- Quick hull algorithm
- Graham algorithm
- Incremental algorithm
- Divide and conquer algorithm

3D convex hull

- Basics
- Gift wrapping algorithm
- Divide and conquer algorithm
- Incremental algorithm

Polyhedron

- A polyhedron is a geometrical shape. It is a 3D shape with flat faces, and straight edges. Each face is a polygon surrounded by edges.

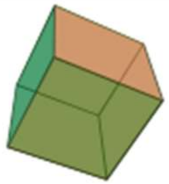
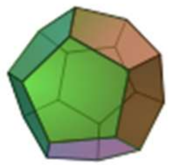
- Two types of polyhedron are convex and concave.

- The line connecting any two points of a convex polyhedron is inside the polyhedron.

- The line connecting two points of a concave polyhedron may go outside the polyhedron.

Platonic solid

- In three-dimensional space, a Platonic solid is a regular, convex polyhedron.
- All edges and angles are equal.

Tetrahedron	Cube	Octahedron	Dodecahedron	Icosahedron
Four faces	Six faces	Eight faces	Twelve faces	Twenty faces
				

How many Platonic solids exist?

Platonic solid

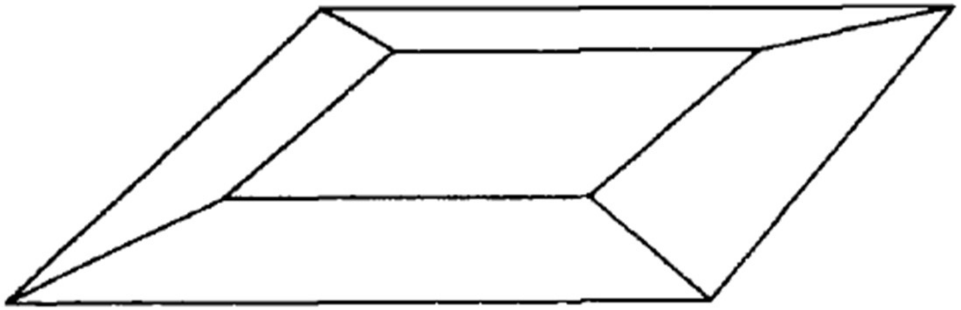
- Assume p is the number of vertices each facet, the sum of interior angles of the regular p polygon is $\pi(p-2)$.
- Then, each angle is $\pi(1-2/p)$.
- Assume w is the number of joint facets around one polyhedron vertex, the sum of interior angles of related polygons around the vertex is $w\pi(1-2/p)$.
- Then, $w\pi(1-2/p) < 2\pi$.
- Then, $(p-2)(w-2) < 4$.

p	3	4	3	5	3
w	3	3	4	3	5

Euler's formula for Polyhedron

$$\boldsymbol{v} - \boldsymbol{e} + \boldsymbol{f} = \mathbf{2}$$

where v, e and f denotes the number of vertices, edges, and facets of the polyhedron, respectively.



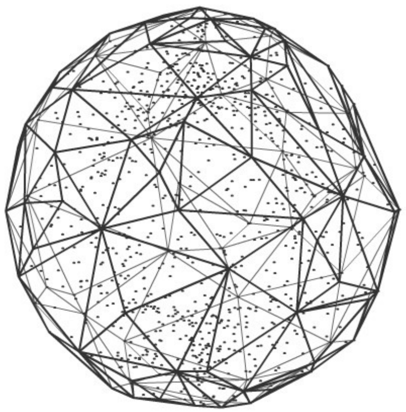
Gift wrapping algorithm

2D $\rightarrow$ 3D

find connected extreme edge $\rightarrow$ find connected extreme facet

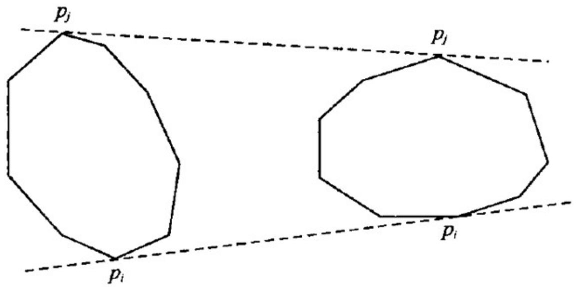
find initial pseudo extreme edge $\rightarrow$ find initial pseudo extreme facet

$O(n^2)$ $\rightarrow$ $O(n^2)$



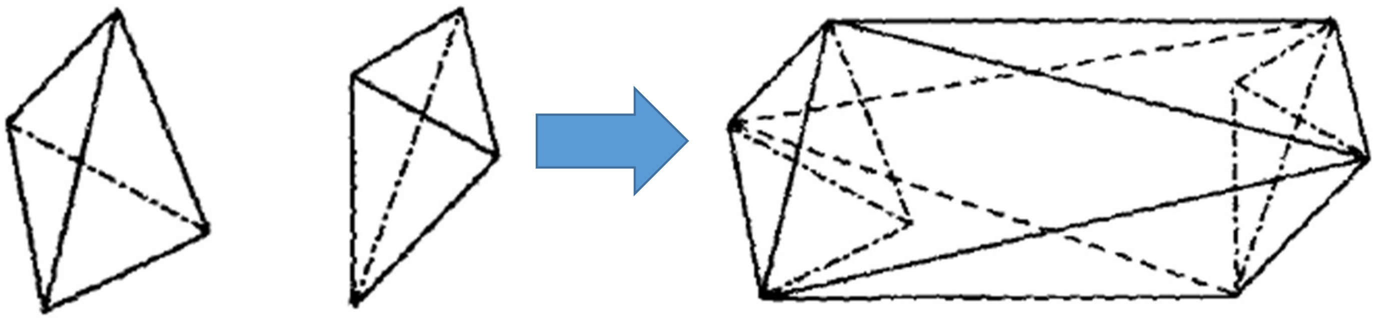
Divide and conquer algorithm

- 2D merge process



$O(n)$

- 3D merge process



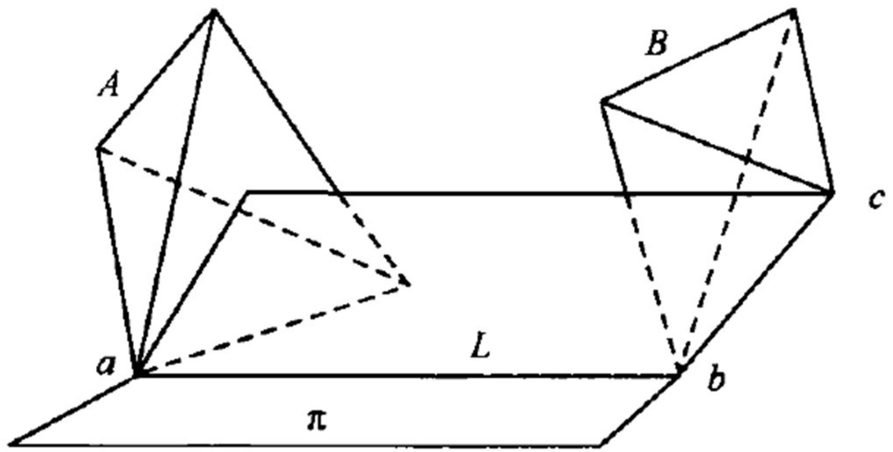
$O(n)$

Divide and conquer algorithm

- 3D merge process

Theory 1

When plane π is rotating along L, the first approached vertex c is the vertex on the convex hull and adjacent to a or b.

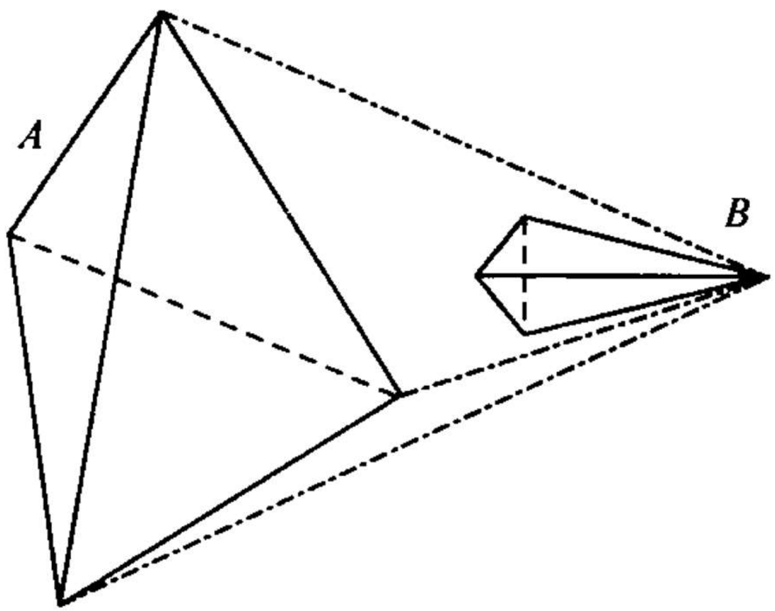


a and b is the vertex of polyhedron A and B, respectively. L is the line passing a and b. π is the plane passing L.

What's the time complexity of finding c?

How to guarantee the process of finding c in $O(1)$?

Divide and conquer algorithm



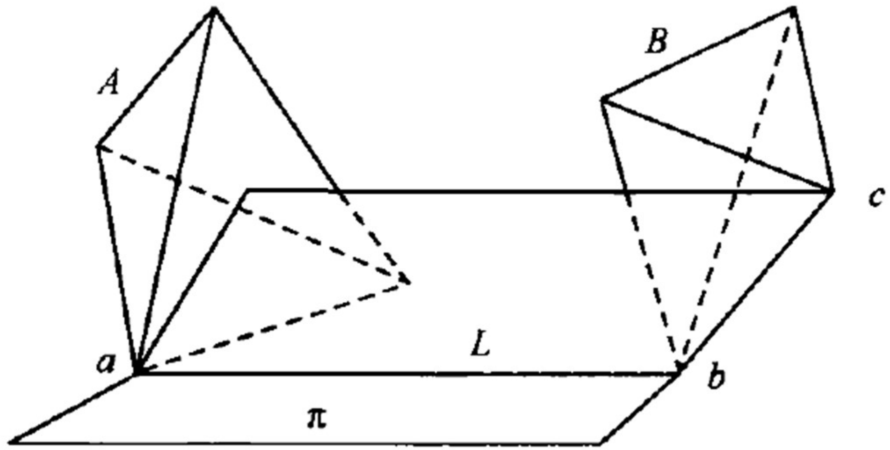
What's the time complexity of finding c?

Divide and conquer algorithm

- 3D merge process

Theory 2

We search the neighboring vertices of a and b in anti-clockwise direction (from each other's view) and do not need to go back.



The time of finding encapsulated facets has linear relationship with the e or v.

Total time complexity is $O(nlogn)$

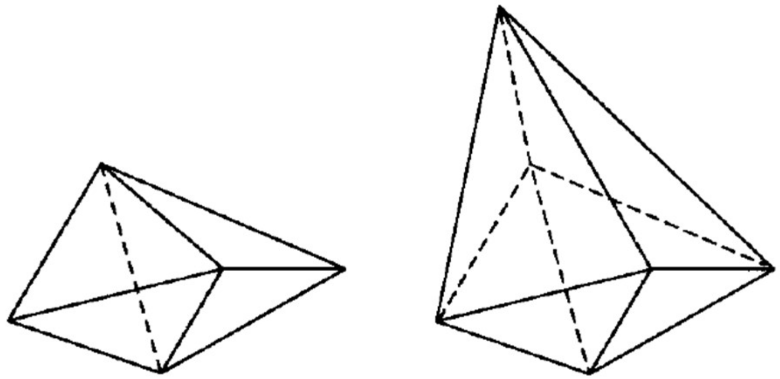
Incremental algorithm

1. $p_k \in H_{k-1}$

- Decided by checking whether p_k locates on one side of all facets.
- The order of three vertices and the outward normal vector of the facet should be in right-hand system.
- $H_k = H_{k-1}$

2. $p_k \in H_{k-1}$

- Find tangent facets from p_k to H_{k-1}.
- Delete inner facets.



(a) (b)

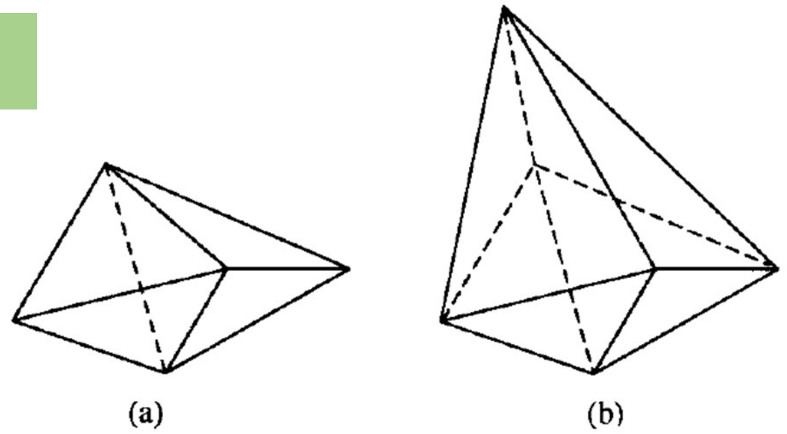
H_{k-1} H_k after adding one point

Incremental algorithm

Find tangent facets from p_k to H_{k-1}.

Find the edge connecting visible and invisible facets for p_k

$4 + 5 + \cdots + n = O(n^2)$



H_{k-1} H_k after adding one point

Convex hull of simple polygon

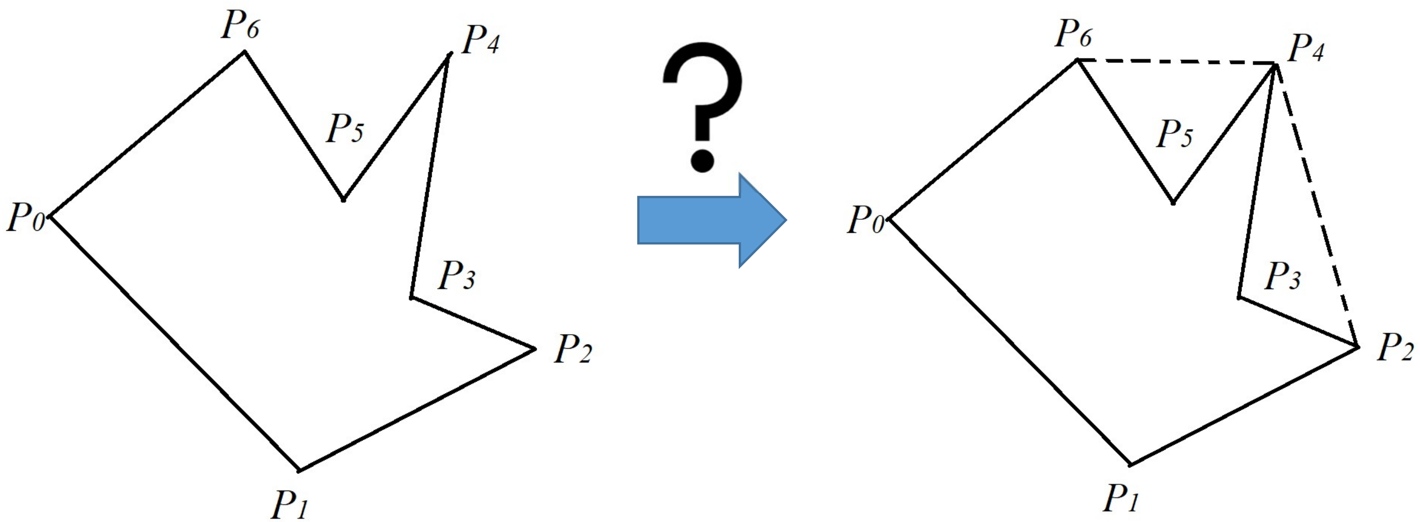
- If any two edges are not intersected, the polygon is a **simple polygon**.

- Compare the simple polygon with an arbitrary point set, its vertices are in order.

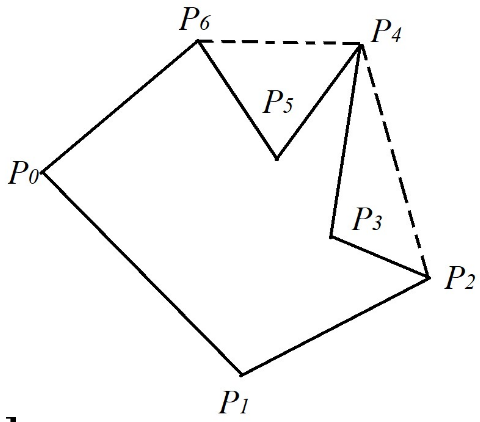
$O(nlogn)$ → $O(n)$

Partial convex hull algorithm of simple polygon

- Assume the vertices of the simple polygon are $p_0, p_1, p_2, \dots, p_n$. The polygon is always on the left side if walking along the vertices in order.
- Assume $p_i = succ(p_{i-1}), p_{i-1} = pred(p_i), i = 1,2, \dots, n$ and p_0 is extreme vertex.



Partial convex hull algorithm of simple polygon

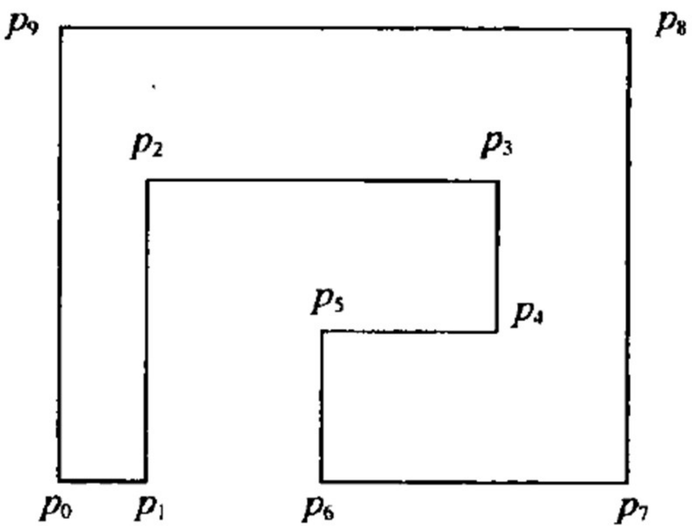


- ***Input:*** the vertices of the simple polygon $p_0, p_1, p_2, \dots, p_n$
- ***Output:*** the convex hull of the simple polygon

$p = p_0;$
do{
    if $succ(p)succ(succ(p))$ is on the right of $psucc(p)$ then
    {   delete $succ(p)$;
        if $p \neq p_0$ then $p = pred(p)$;
    }
    else
        $p = pred(p)$
}

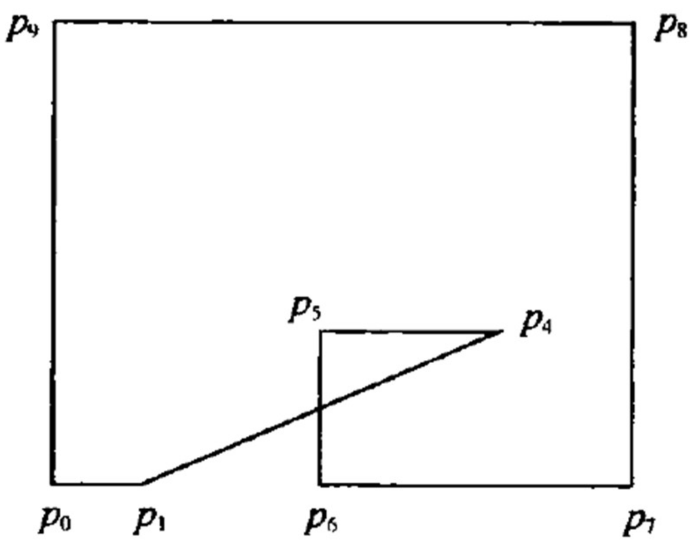
Does it work for all the cases?

Partial convex hull algorithm of simple polygon



(a)

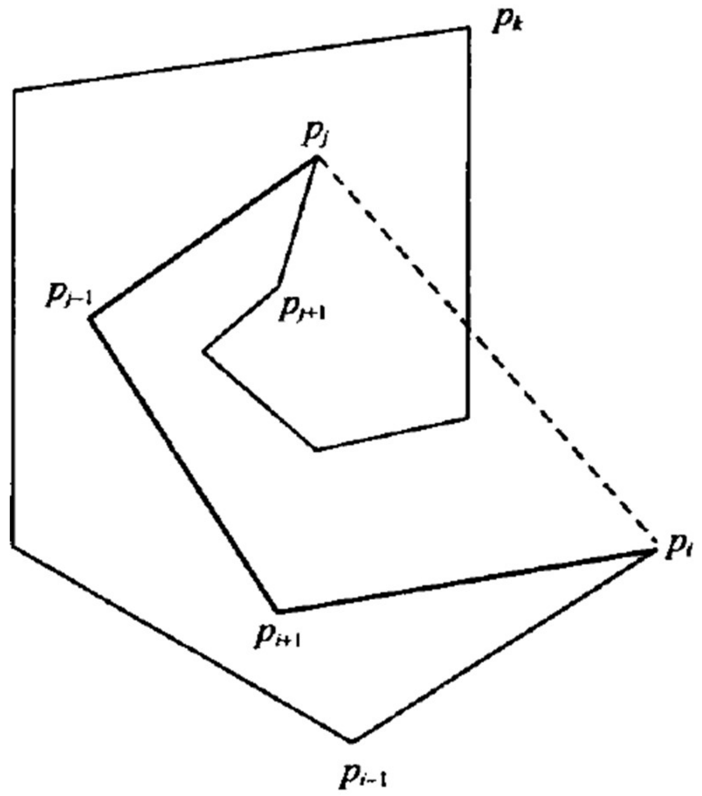
Original polygon



(b)

Computational result

Partial convex hull algorithm with a trap

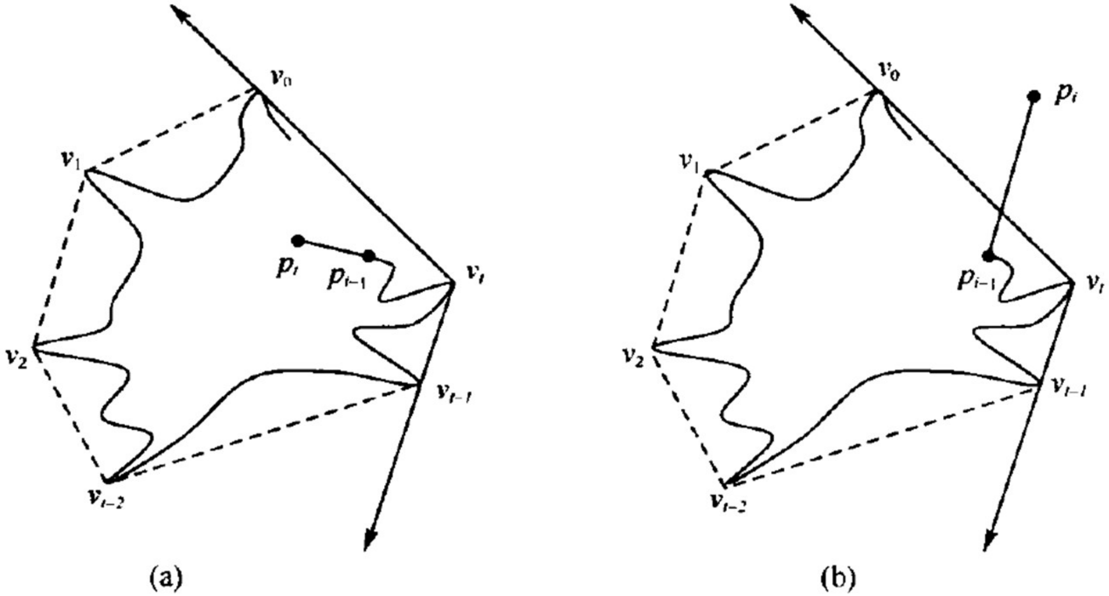


Use a trap!

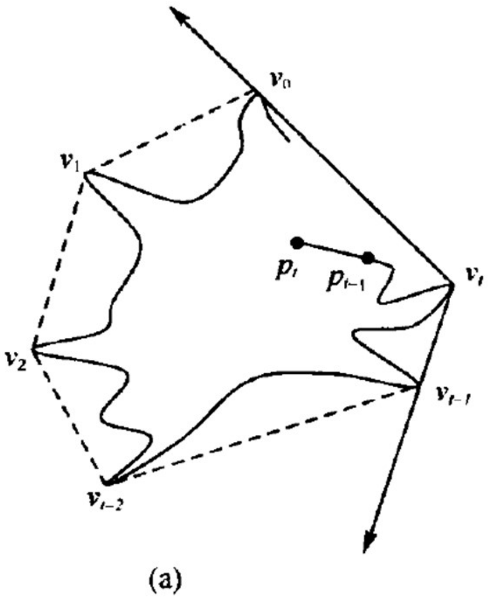
$O(n)$

Melkman algorithm for simple polygon

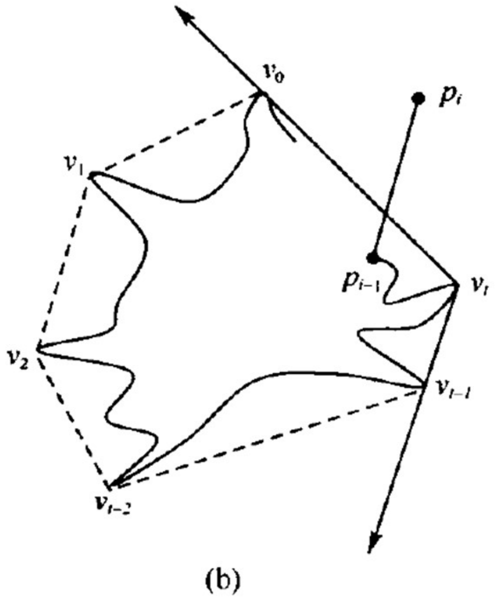
Could we extend incremental algorithm for computing convex hull of simple polygon in $O(n)$?



Melkman algorithm for simple polygon



p_i is inside H_{i-1}



p_i is outside H_{i-1}

$O(n)$

Why?

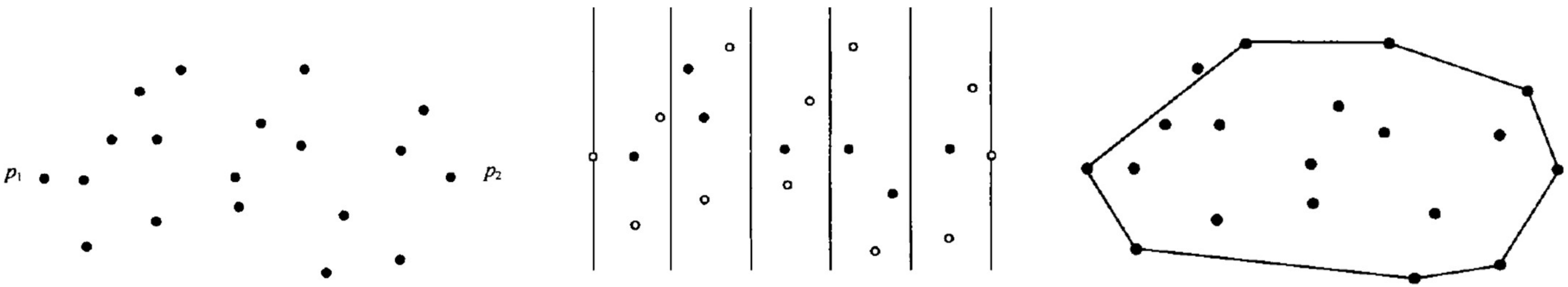
Decide whether p_i is inside the angle $\angle v_{t-1}v_tv_0$.

Approximated convex hull

a) Find p_1 and p_2 with the minimal and maximal x value.

b) Divide the space between p_1 and p_2 into k strips and find the points with minimal and maximal y value in each strip. These points form the set S^*.

c) Compute the convex hull of S^* (the points are in order according to x value).



(a) (b) (c)

Approximated convex hull

- The precision of the approximated convex hull.
- For any point of S not in the approximated convex hull, the distance to the approximated convex hull is less than $(x_{max} - x_{min})/k$.