Design and Analysis of Algorithm

Divide and Conquer strategy
(Convex Hull Problem)

Lecture -25

Overview

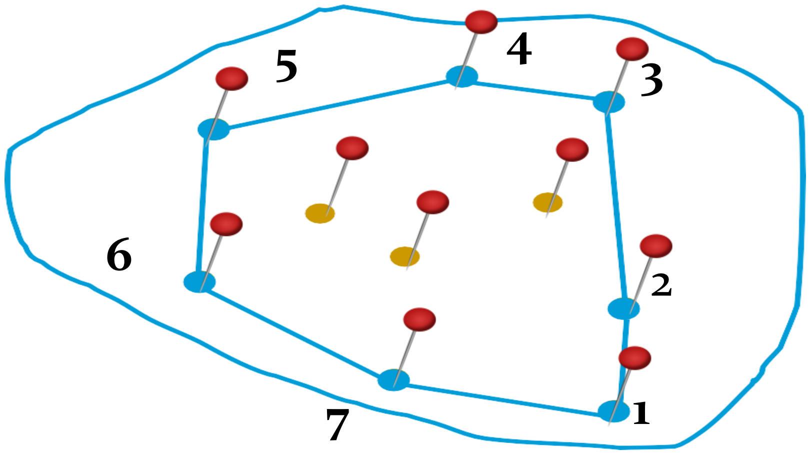
- *Learn the implementation techniques of "divide and conquer" in the context of the Convex Hull Problem with analysis.*

Convex Hull

- Given a set of pins on a pinboard, and a rubber band around them .
- How does the rubber band look when it snaps tight? Just imagine.

Convex Hull

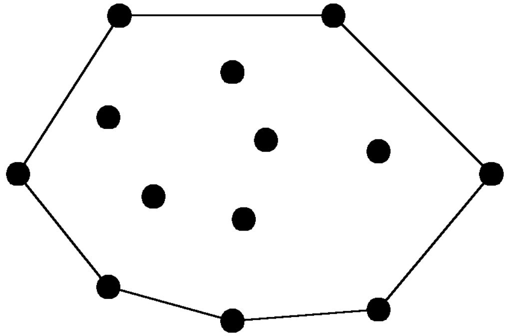
- Given a set of pins on a pinboard, and a rubber band around them .
- How does the rubber band look when it snaps tight? Just imagine.



- We represent the convex hull as the sequence of points on the convex hull polygon, in counter- clockwise order.

Convex Hull

- Definition:
  - Informal: Convex hull of a set of points in plane is the shape taken by a rubber band stretched around the nails pounded into the plane at each point.
  - Formal: The convex hull of a set of planar points is the smallest convex polygon containing all of the points.



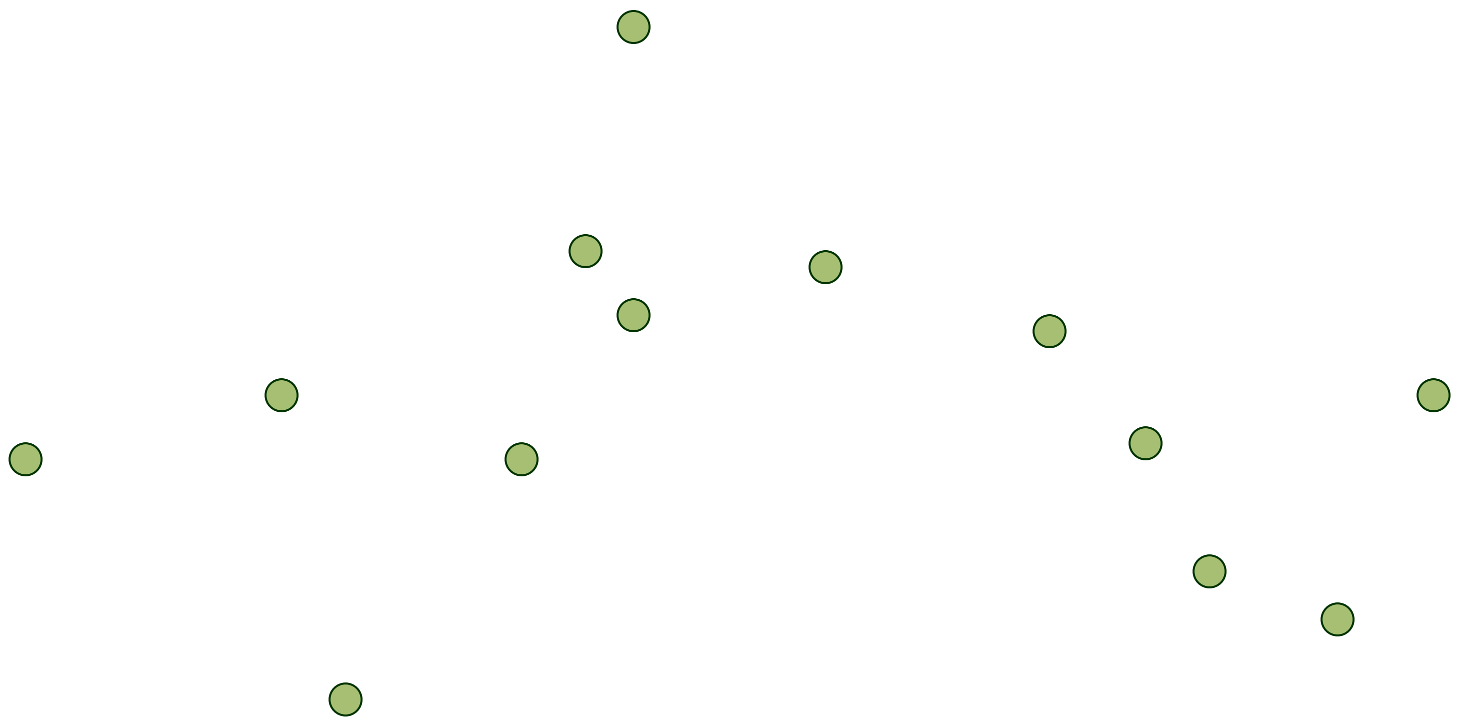
Graham Scan

- Concept:
  - Start at point guaranteed to be on the hull. (the point with the minimum y value)
  - Sort remaining points by polar angles of vertices relative to the first point.
  - Go through sorted points, keeping vertices of points that have left turns and dropping points that have right turns.

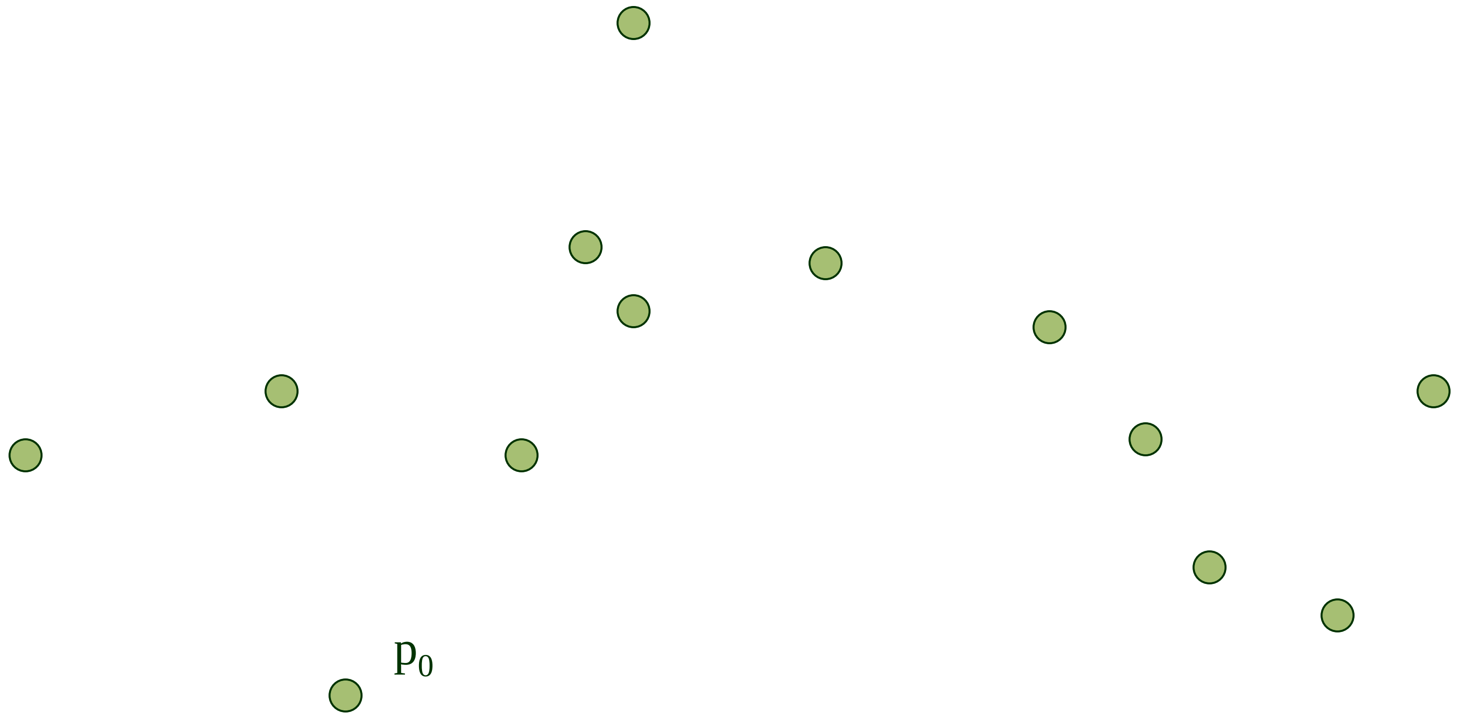
Graham Scan

- Concept:
  - Start at point guaranteed to be on the hull. (the point with the minimum y value)
  - Sort remaining points by polar angles of vertices relative to the first point.
  - Go through sorted points, keeping vertices of points that have left turns and dropping points that have right turns.

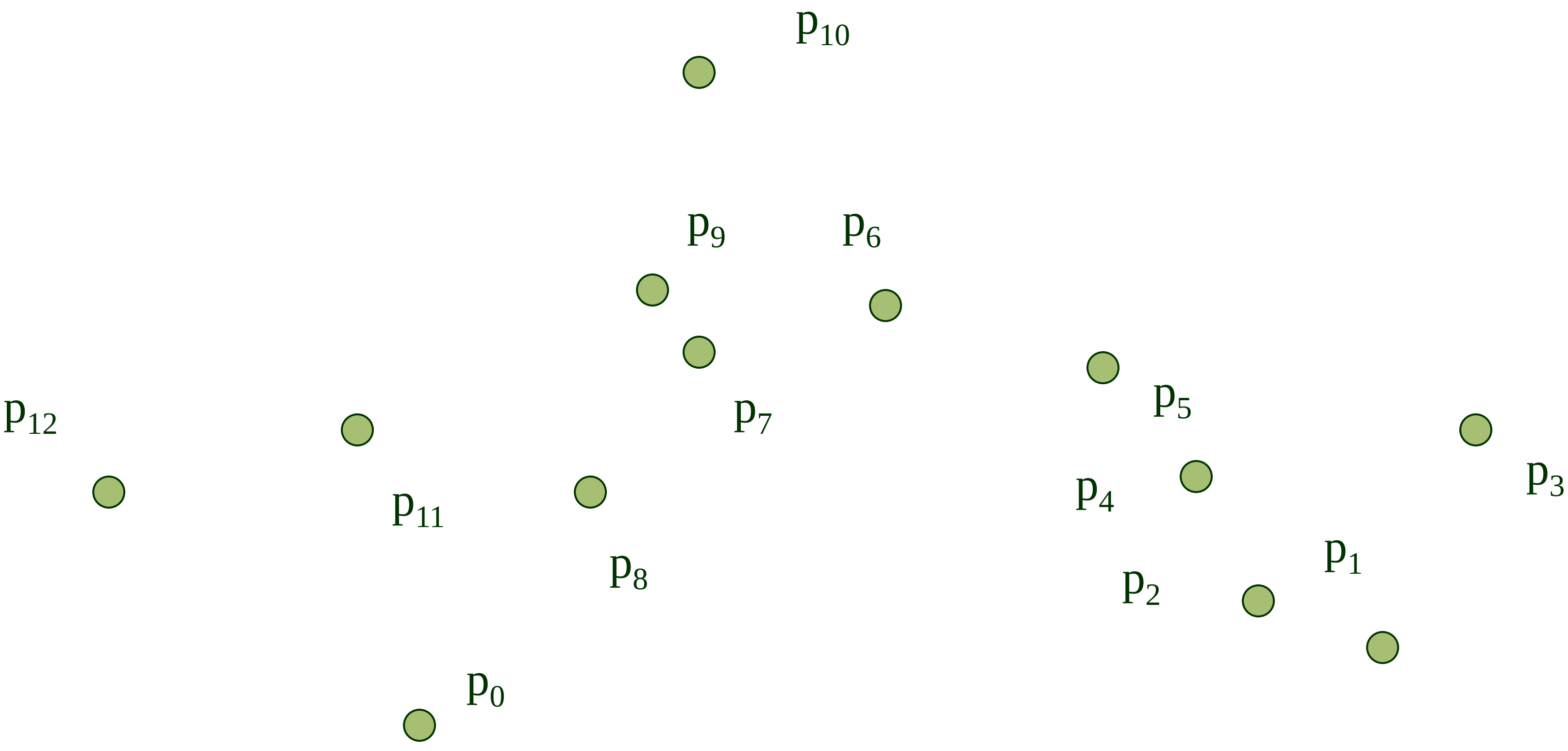
Graham Scan - Example



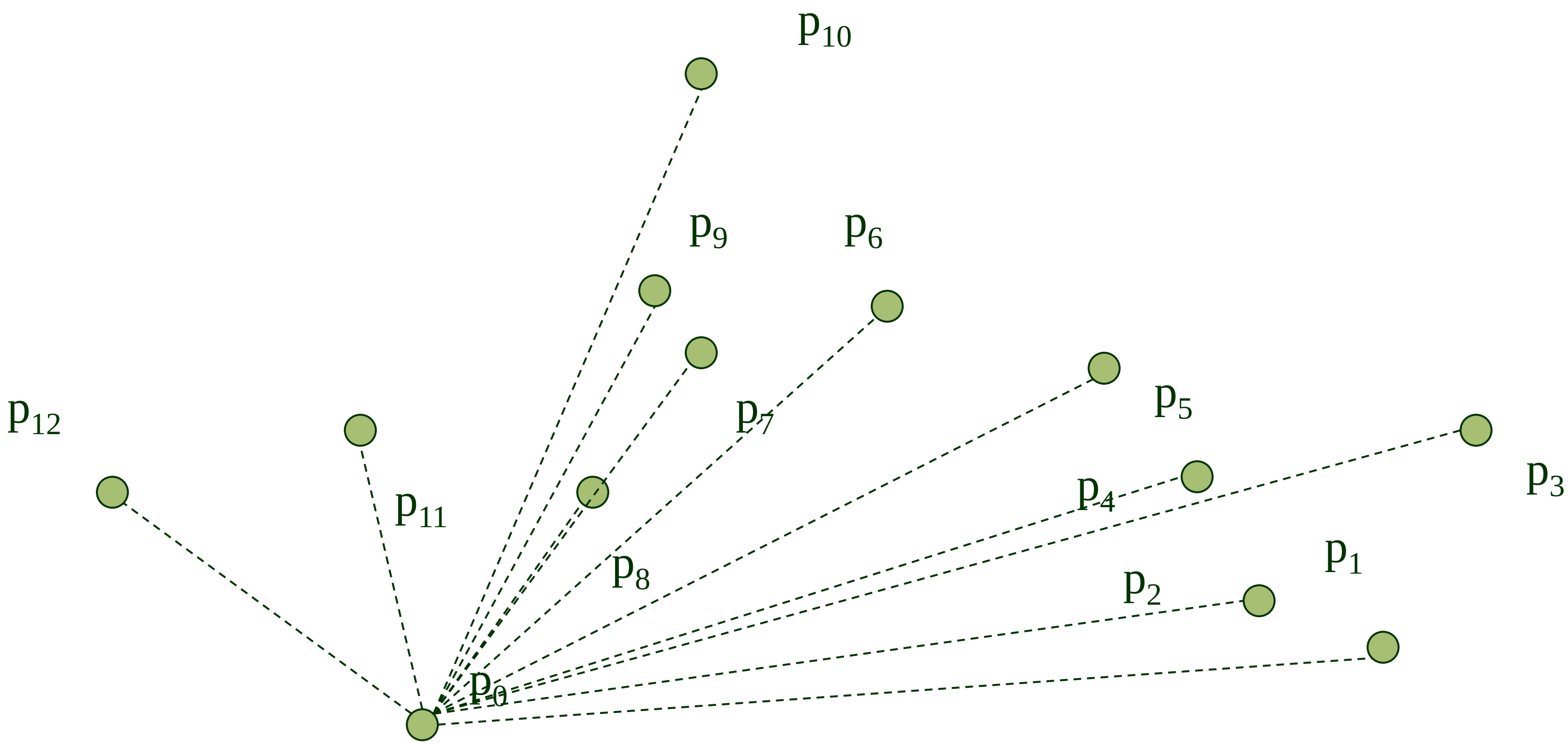
Graham Scan - Example



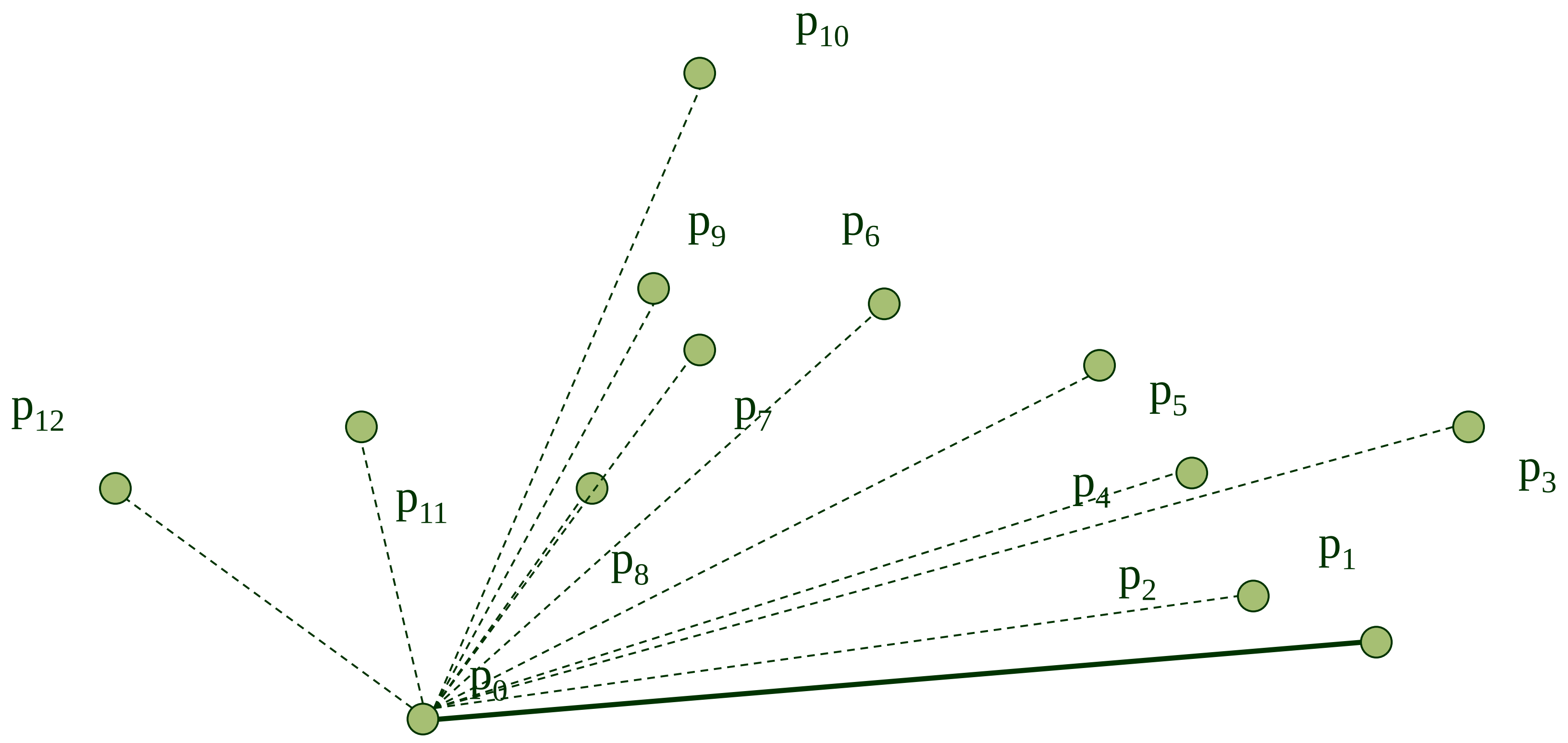
Graham Scan - Example



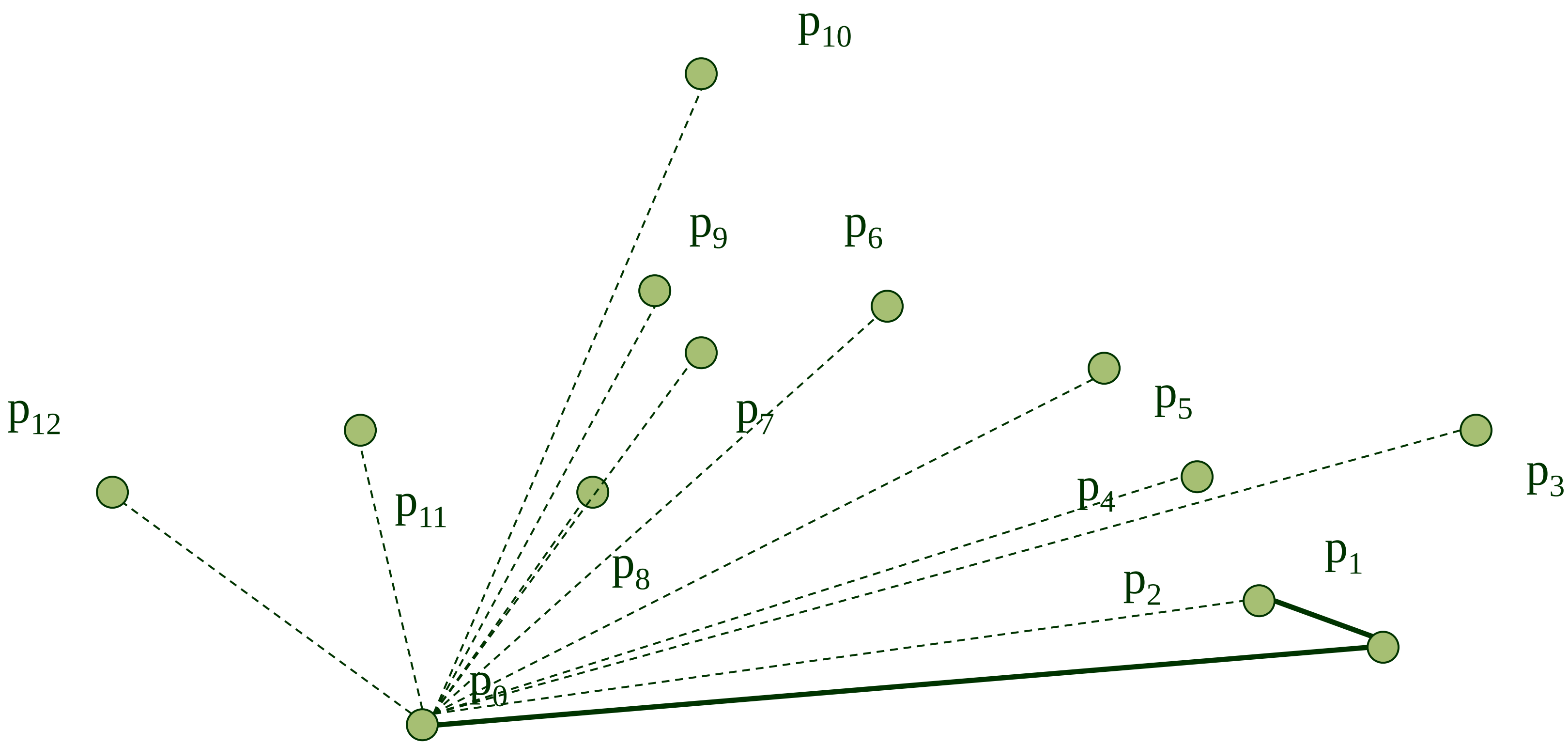
Graham Scan - Example



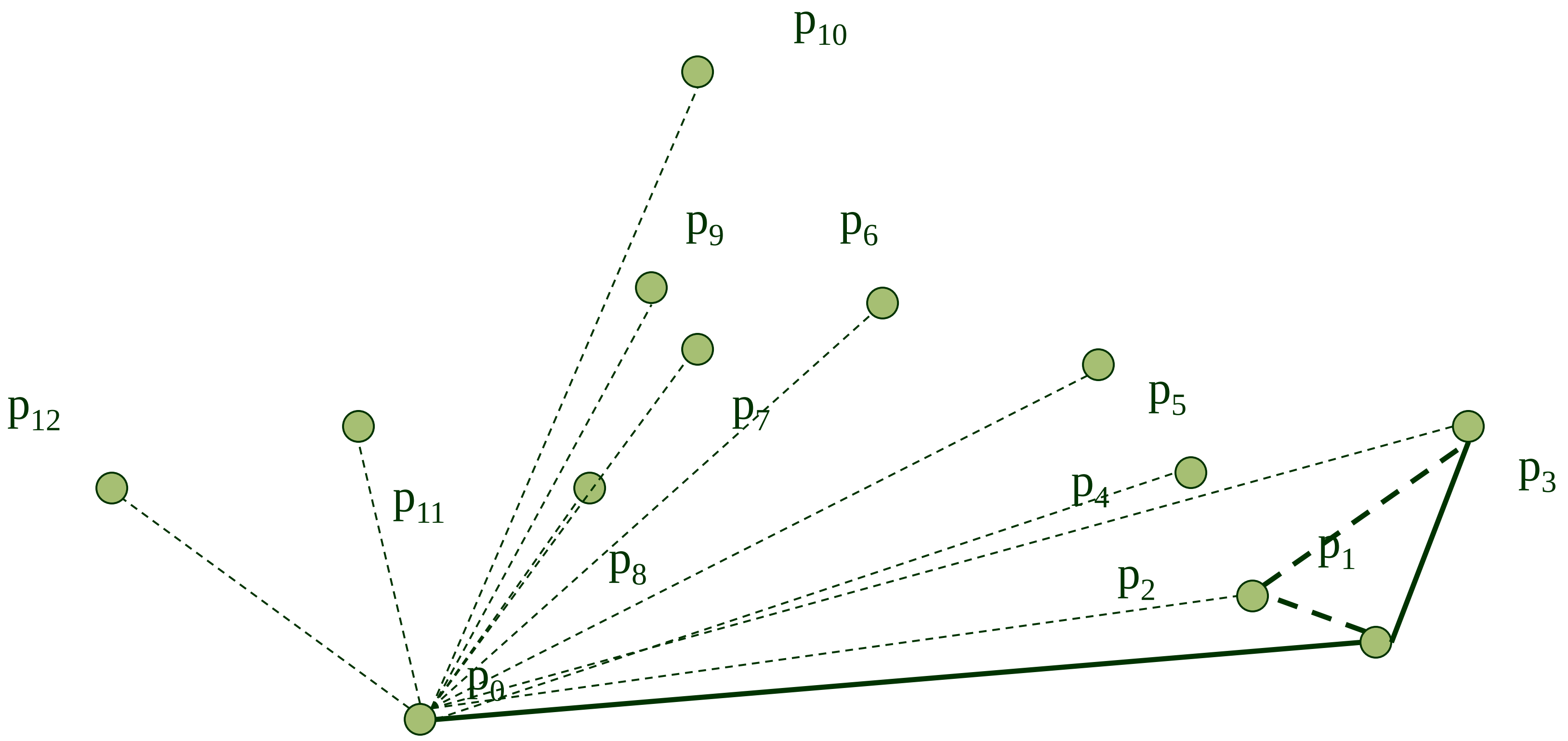
Graham Scan - Example



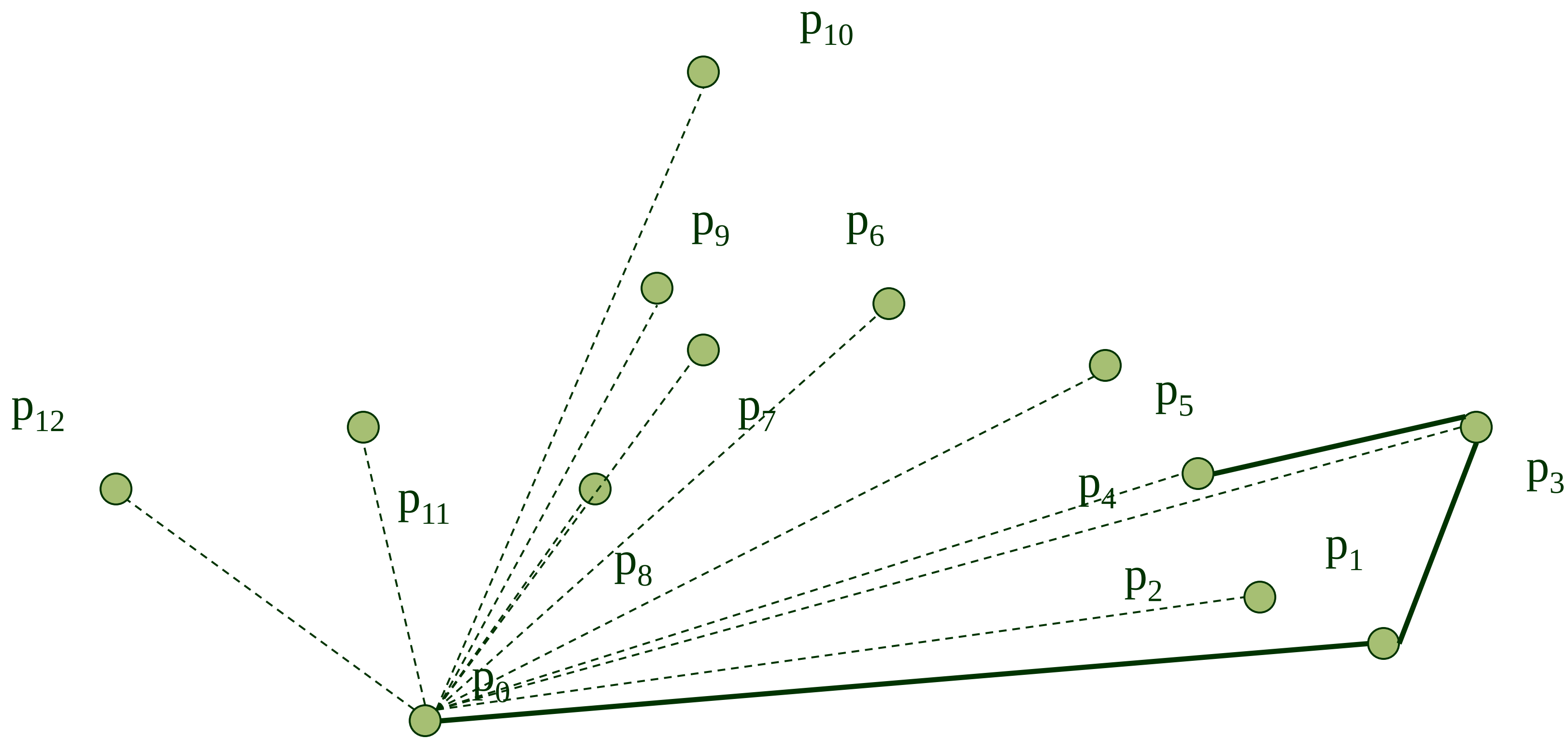
Graham Scan - Example



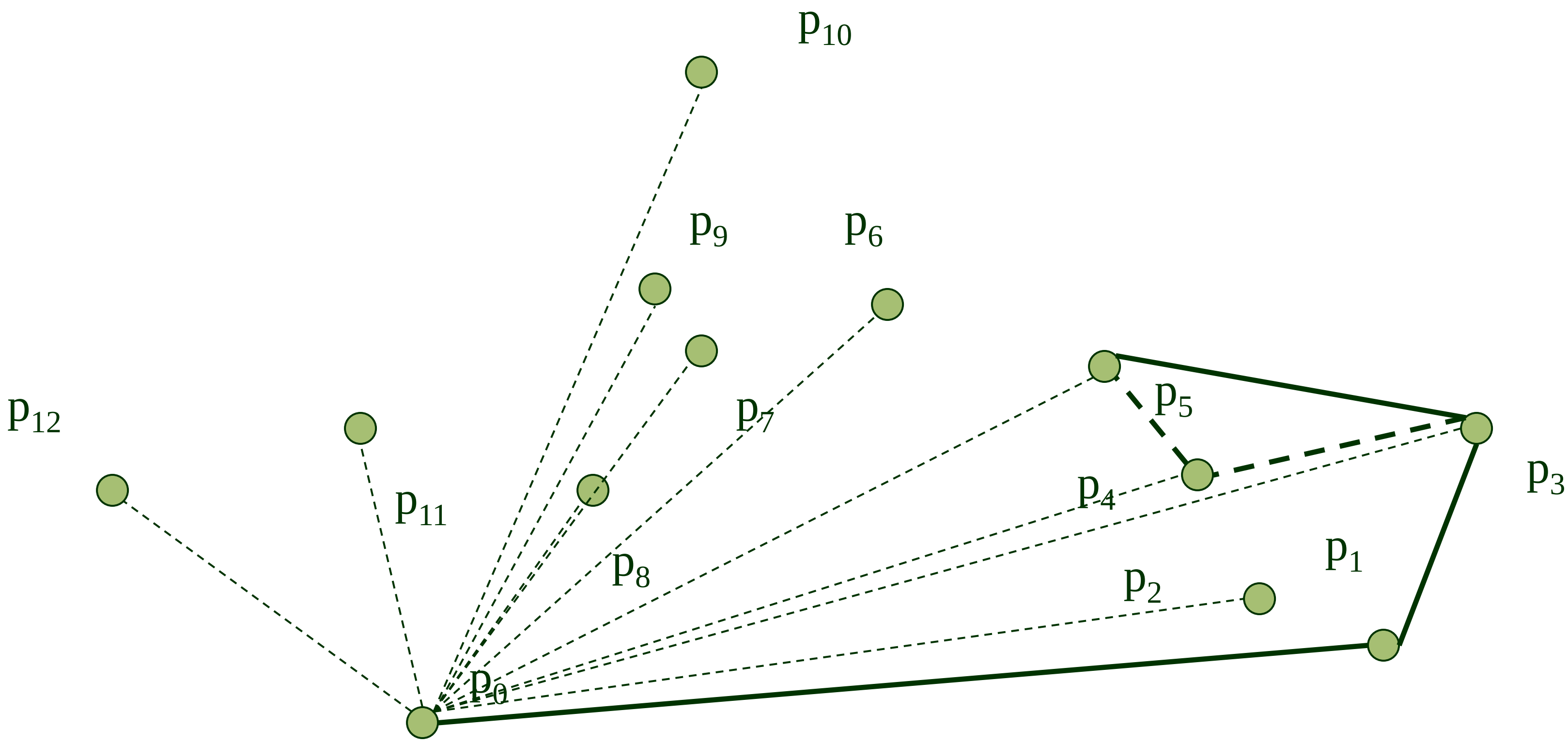
Graham Scan - Example



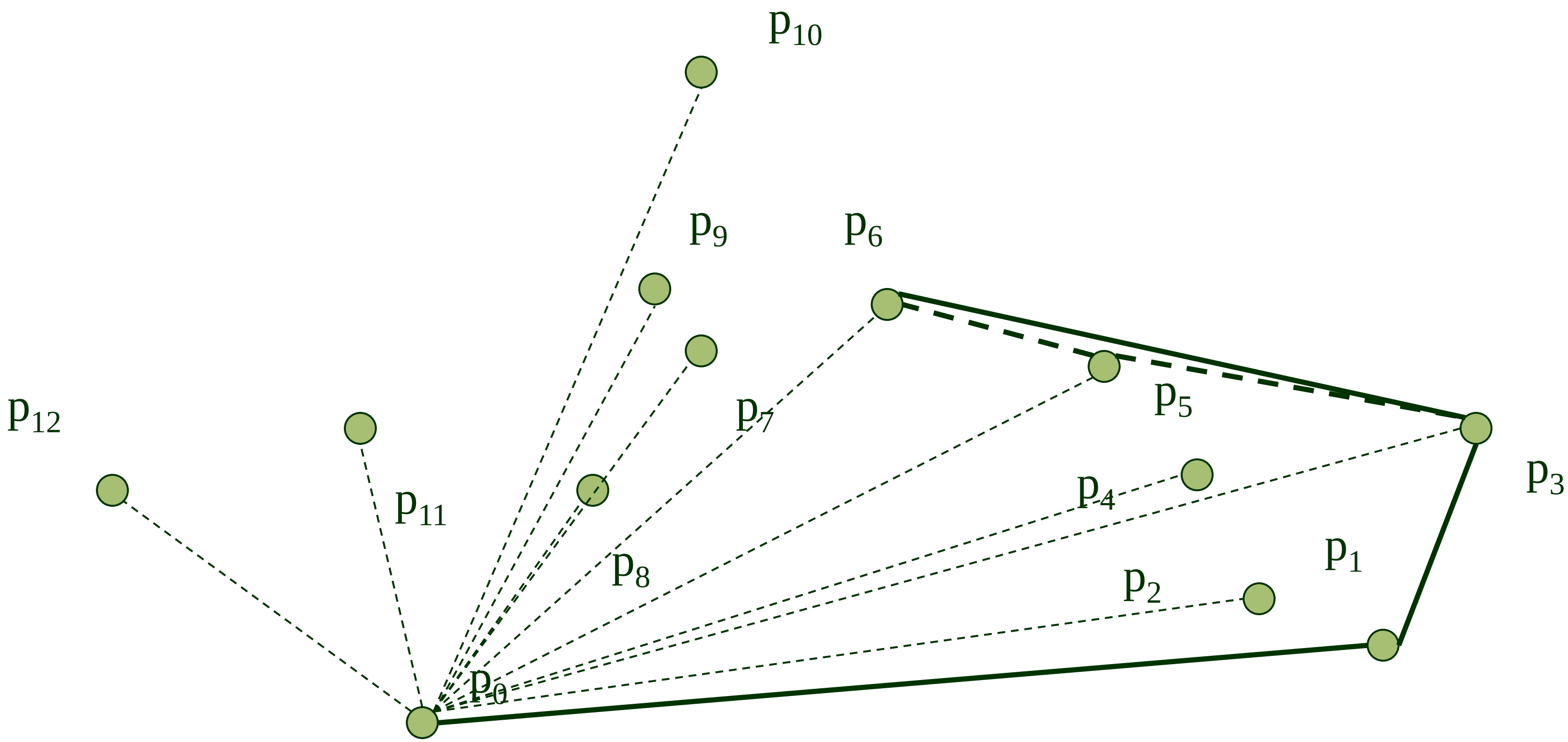
Graham Scan - Example



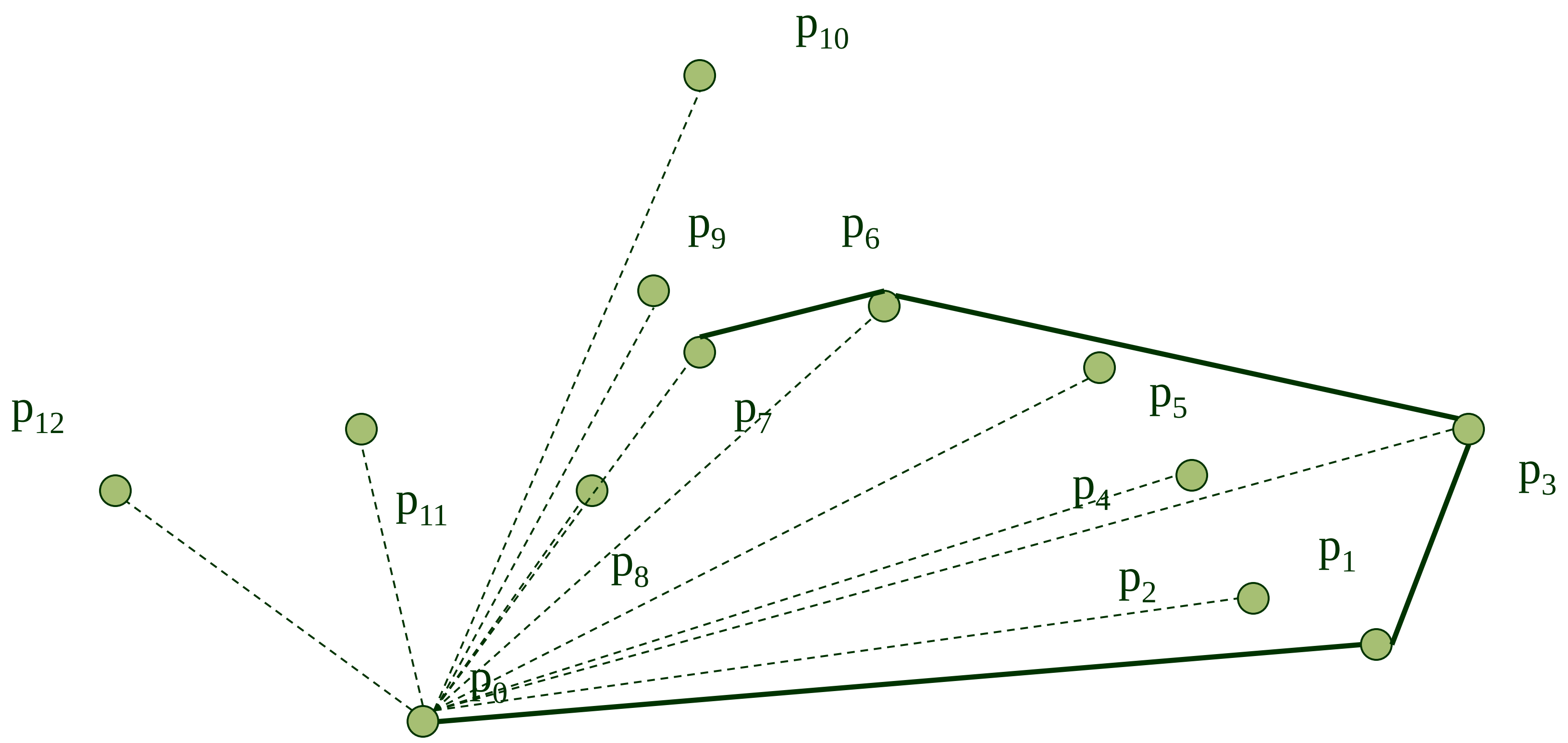
Graham Scan - Example



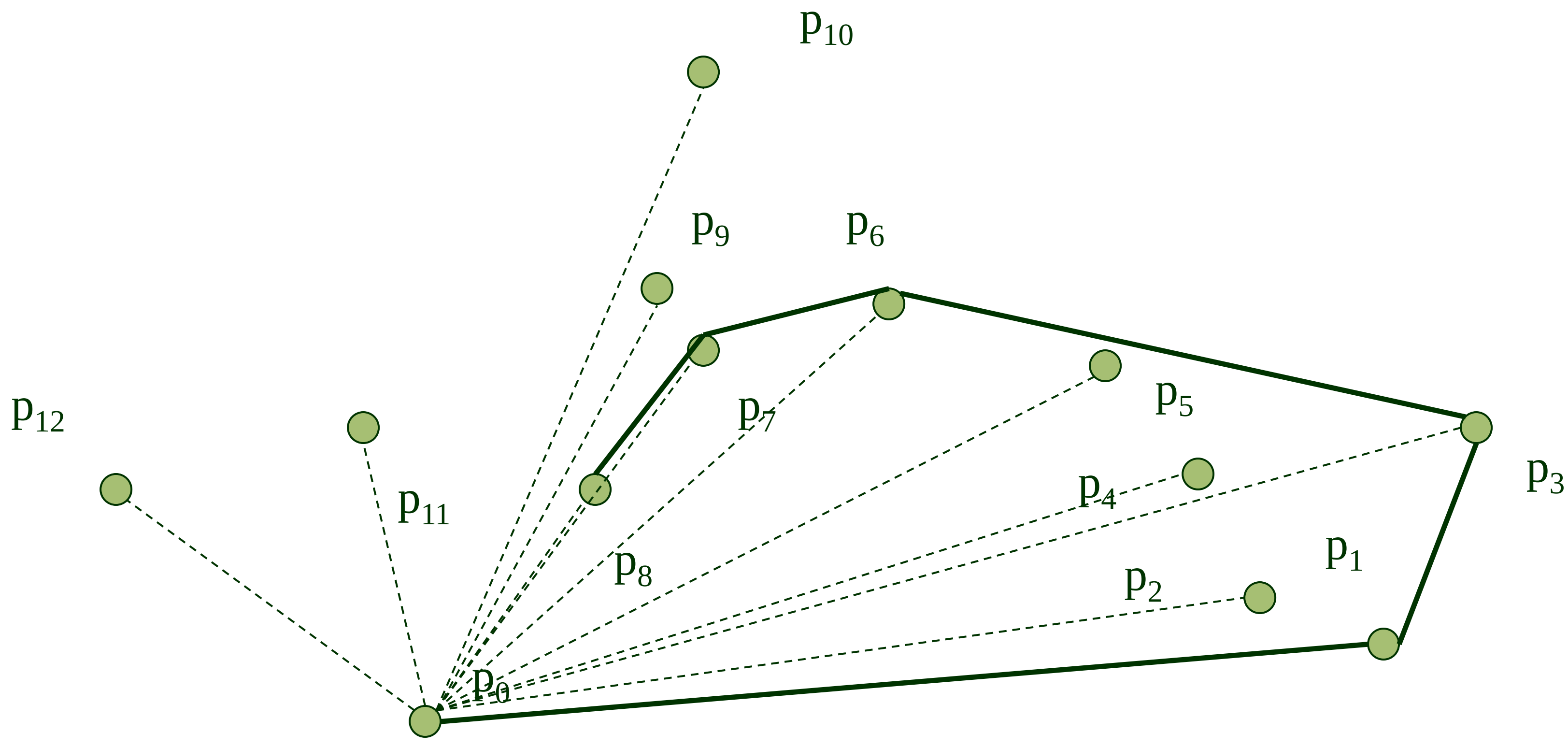
Graham Scan - Example



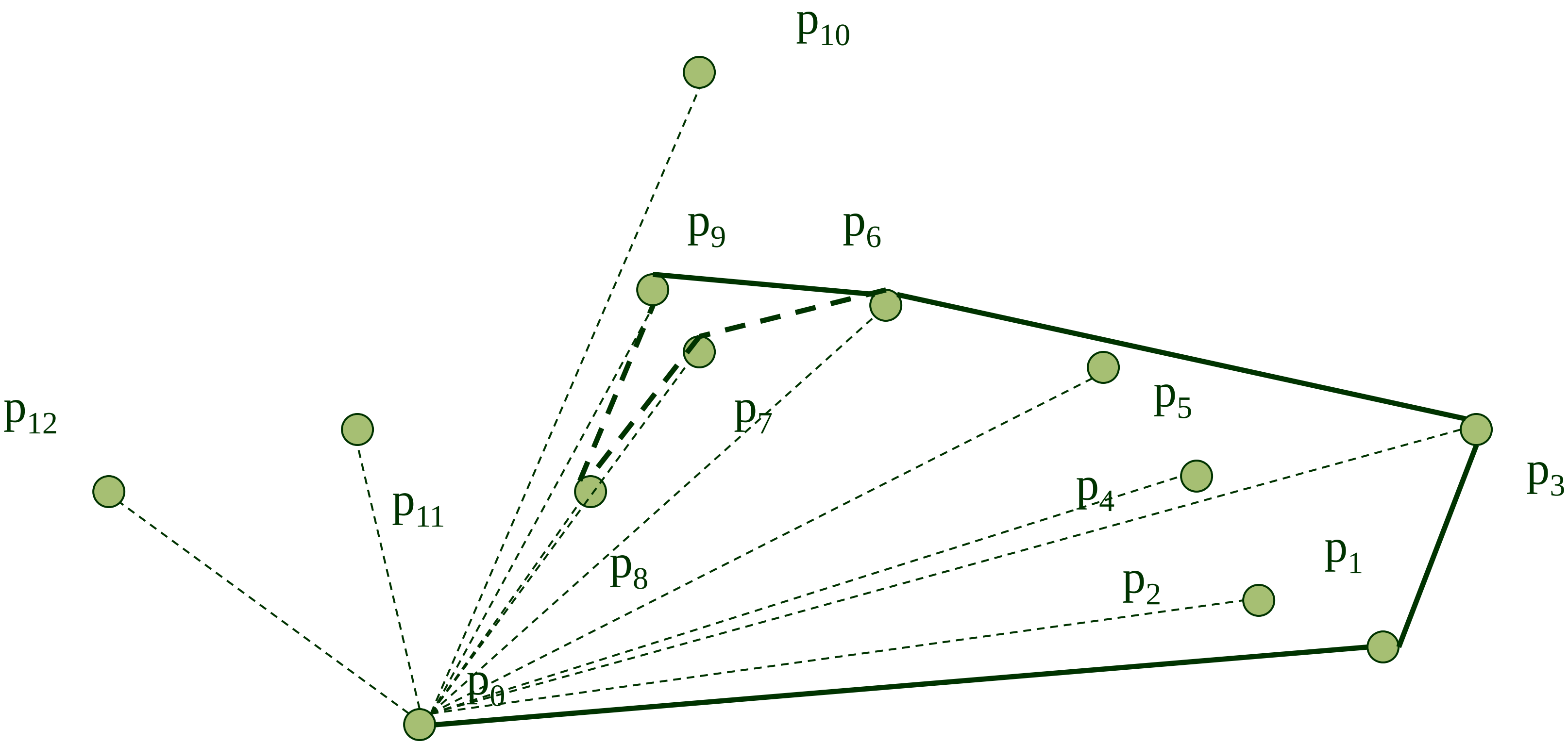
Graham Scan - Example



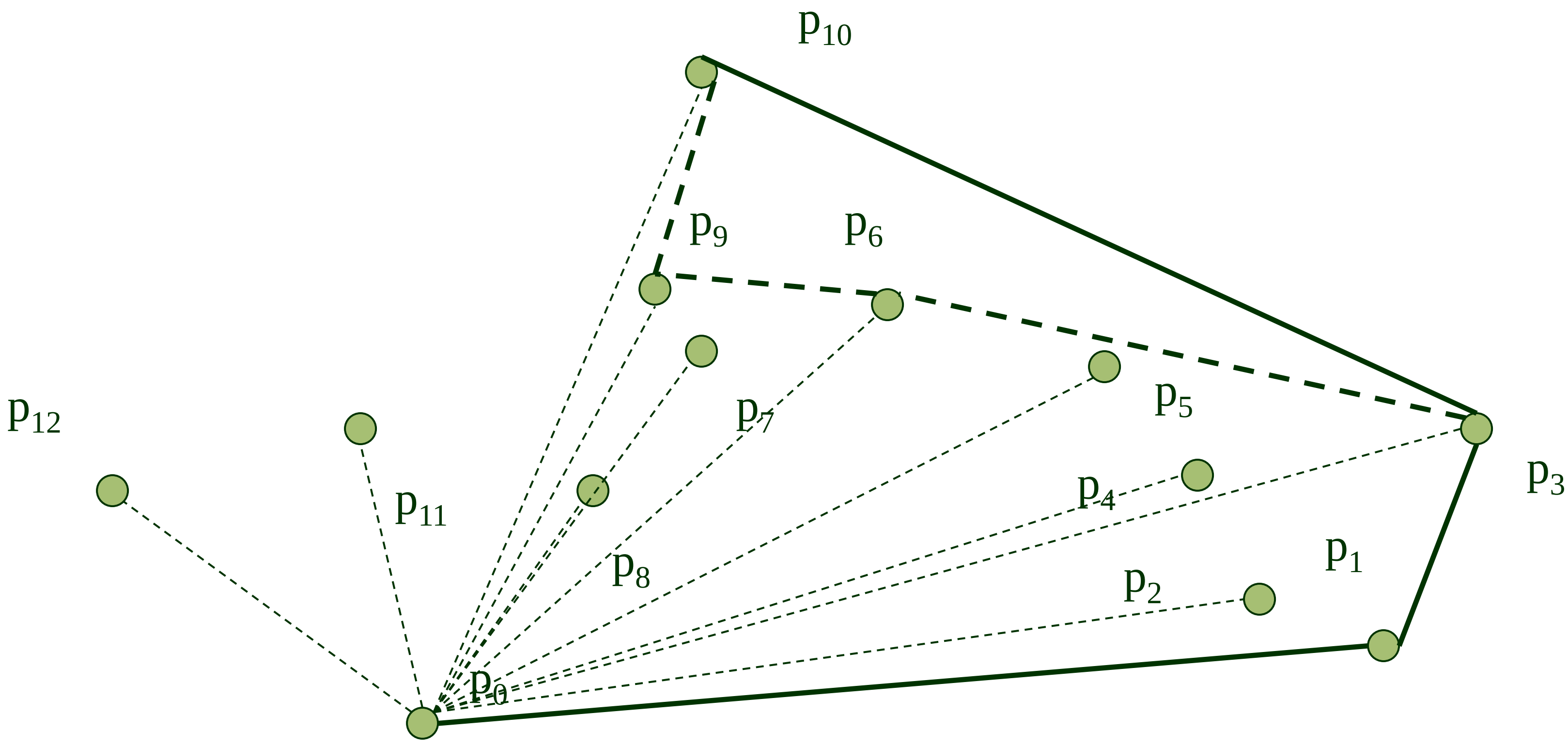
Graham Scan - Example



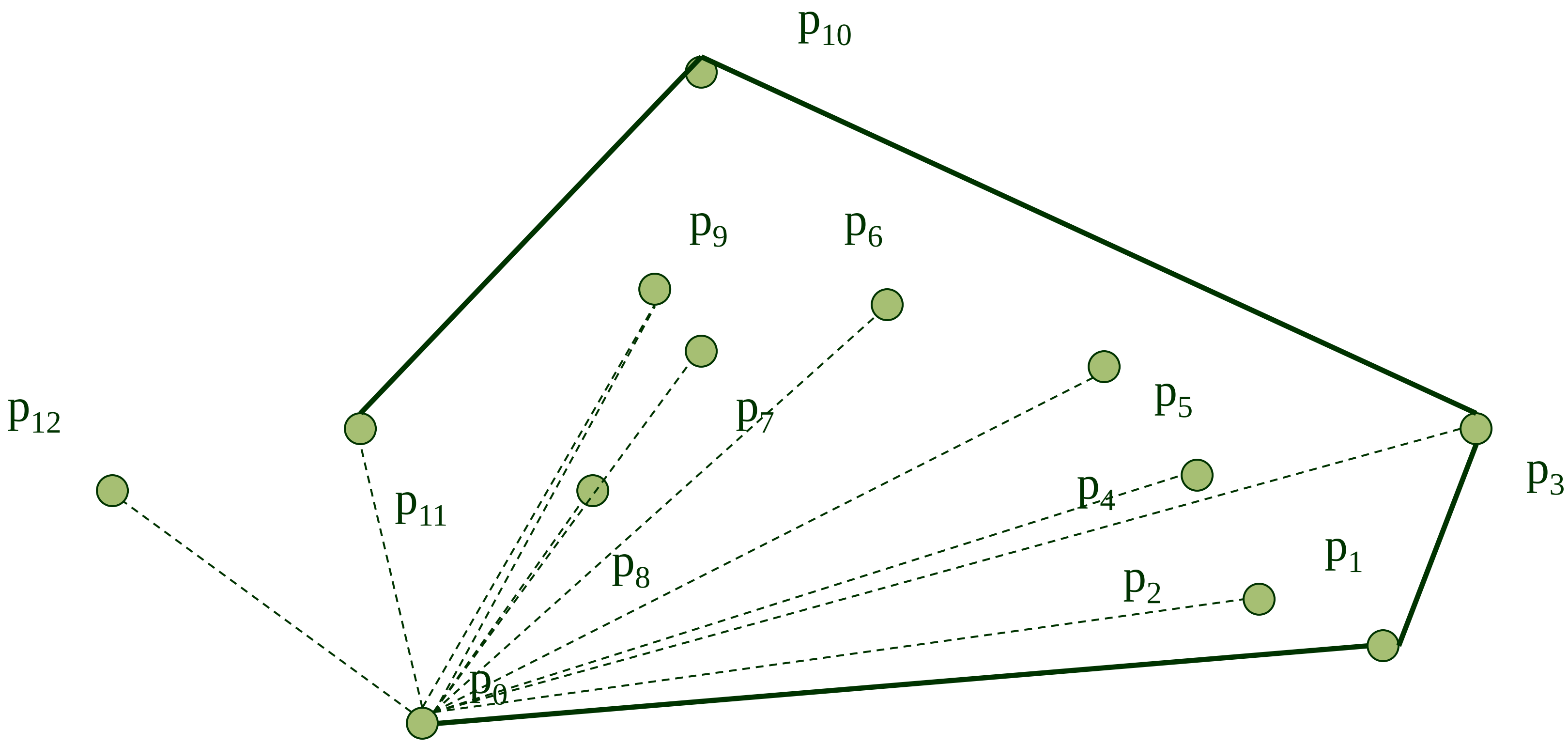
Graham Scan - Example



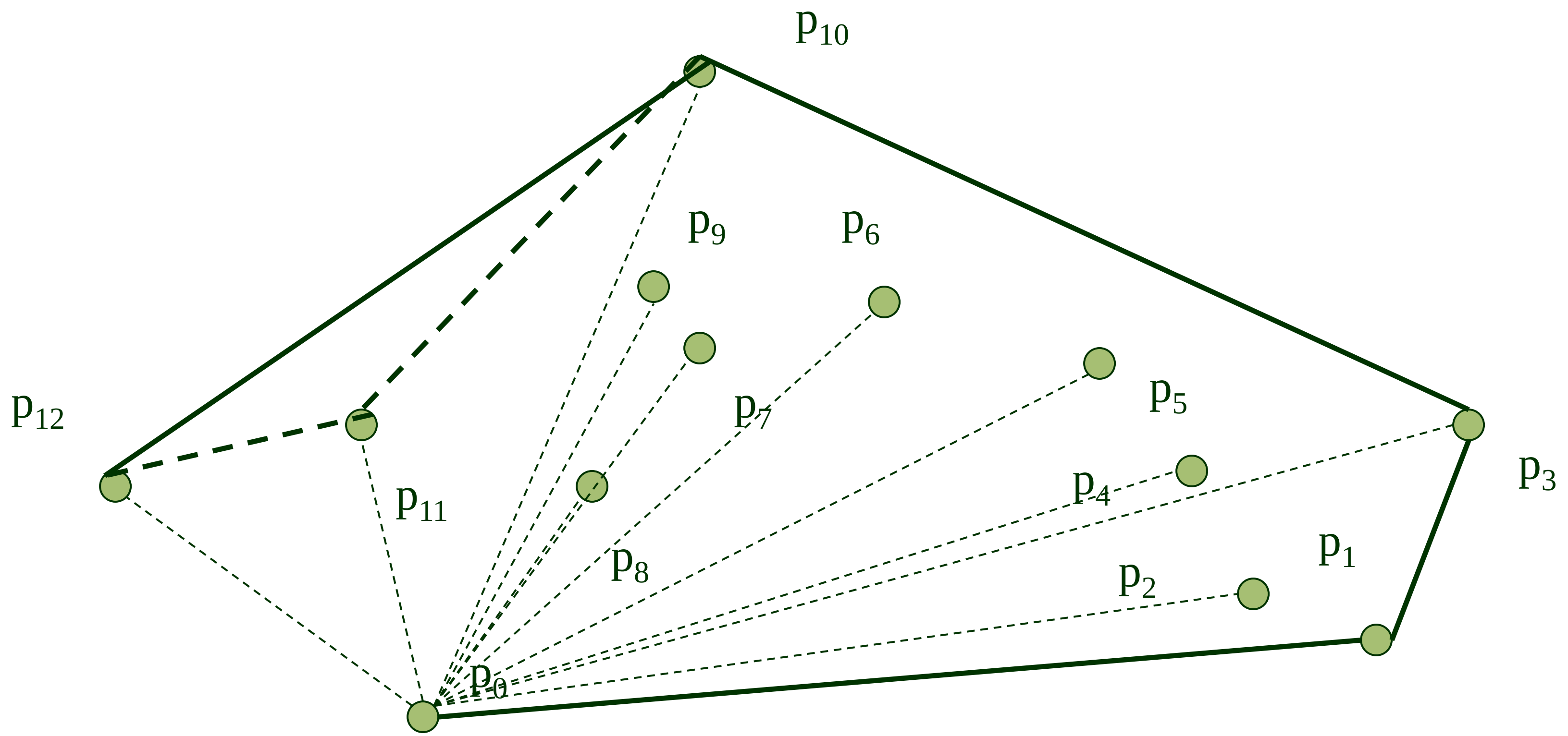
Graham Scan - Example



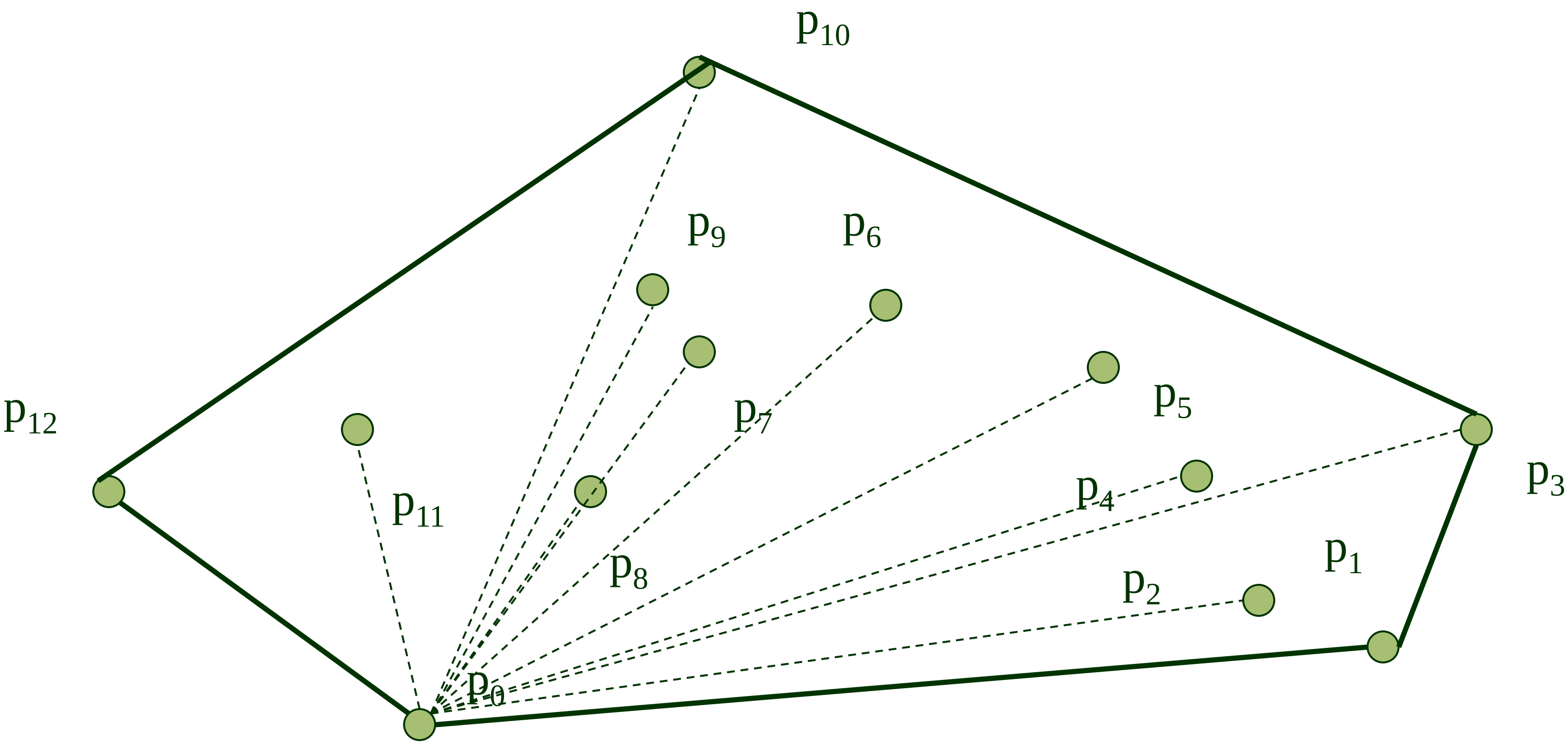
Graham Scan - Example



Graham Scan - Example



Graham Scan - Example



Graham Scan - Algorithm

```
GRAHAM-SCAN(Q)
let p0 be the point in Q with the minimum y-coordinate,
          or the leftmost such point in case of a tie
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)
PUSH(p0, S)
PUSH(p1, S)
PUSH(p2, S)
for i ← 3 to m
     do while the angle formed by points NEXT-TO-TOP(S), TOP(S),
                    and pi makes a nonleft turn
            do POP(S)
        PUSH(pi, S)
return S
```

Graham Scan - Algorithm

GRAHAM-SCAN(Q)

let p_0 be the point in Q with the minimum y-coordinate,
or the leftmost such point in case of a tie ------ $-O(n)$

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 -- $-O(nlogn)$
(if more than one point has the same angle, remove all but
the one that is farthest from p_0)

PUSH(p_0, S) ------ $-O(1)$

PUSH(p_1, S) ------ $-O(1)$

PUSH(p_2, S) ------ $-O(1)$

for $i \leftarrow 3$ **to** m

  do while the angle formed by points NEXT-TO-TOP(S), TOP(S),
  and p_i makes a nonleft turn

    do POP(S)

  PUSH(p_i, S)

$-O(n)$

return S

Graham Scan - Algorithm

GRAHAM-SCAN(Q)

let p_0 be the point in Q with the minimum y-coordinate,
    or the leftmost such point in case of a tie ------ $-O(n)$
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 -- $-O(nlogn)$
    (if more than one point has the same angle, remove all but
    the one that is farthest from p_0)
PUSH(p_0, S) ------ $-O(1)$
PUSH(p_1, S) ------ $-O(1)$
PUSH(p_2, S) ------ $-O(1)$
for $i \leftarrow 3$ **to** m
    do while the angle formed by points NEXT-TO-TOP(S), TOP(S),
        and p_i makes a nonleft turn
            do POP(S)
        PUSH(p_i, S)
return S

$-O(n)$

$$T(n) = O(n) + \left[2T\left(\frac{n}{2}\right) + O(n)\right\} + O(1) + O(1) + O(1) + O(n)$$

$$\mathbf{T(n) = O(n\,lg\,n) + O(n)}$$

Graham Scan - Algorithm

- Time complexity of Graham's scan:
  - $O(n \log n)$ time required to sort of angles in step 2.
  - $O(n)$ time required for visiting n points.(step 6 to step 9)

  Hence we can write the complexity of Graham's scan as:

$$T(n) = \mathrm{O}(n \lg n) + \mathrm{O}(n) = \mathrm{O}(n \lg n)$$

Divide and Conquer (Quickhull)

- QuickHull uses a Divide and Conquer approach similar to the Quick Sort algorithm.
- Benchmarks showed it is quite fast in most average cases.
- Recursive nature allows a fast and yet clean implementation.

Divide and Conquer (Quickhull)

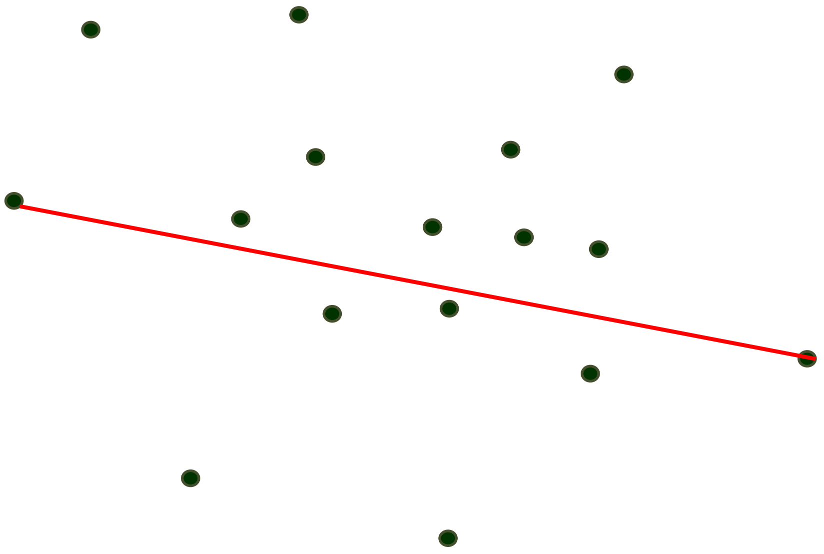
Initial Input

- The initial input to the algorithm is an arbitrary set of points as shown in the figure.

Divide and Conquer (Quickhull)

First Two Points on the Convex Hull

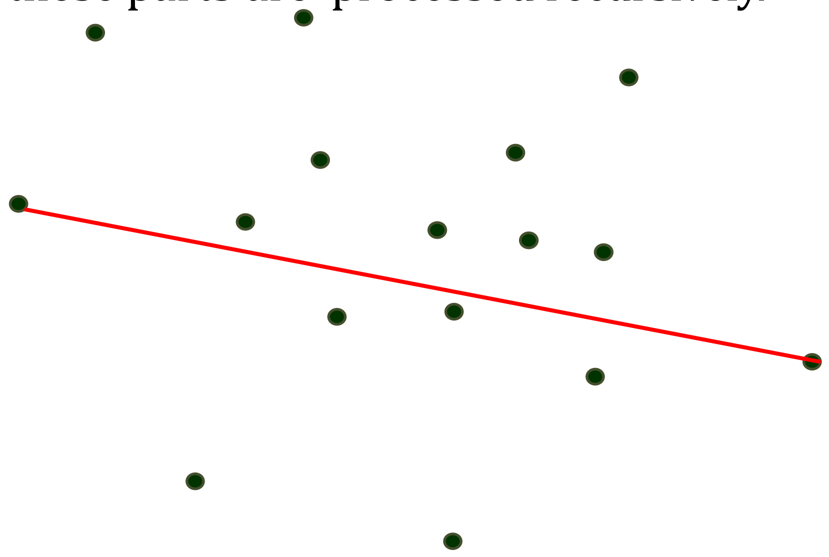
- Starting with the given set of points the first operation done is the calculation of the two maximal points on the horizontal axis. i.e.



Divide and Conquer (Quickhull)

Recursively Divide

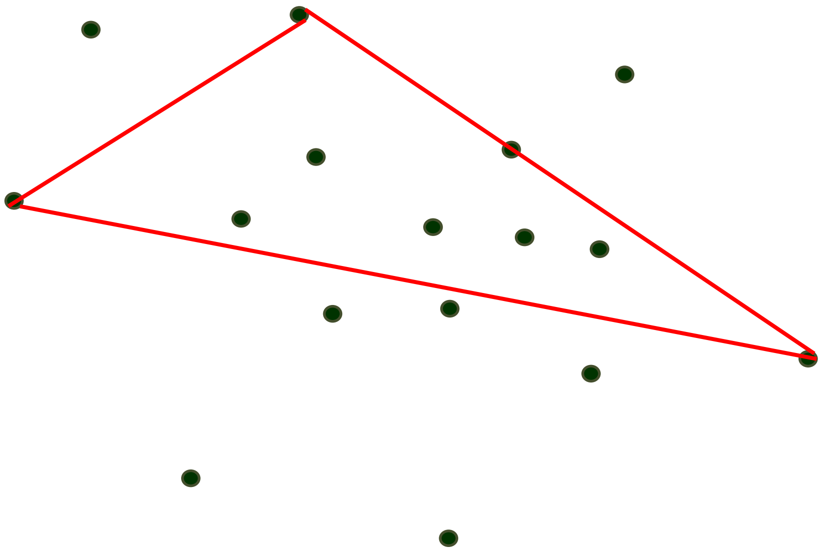
- Next the line formed by these two points is usedto divide the set into two different parts.
- Everything left from this line is considered onepart, everything right of it is considered another one.
- Both of these parts are processed recursively.



Divide and Conquer (Quickhull)

Max Distance Search

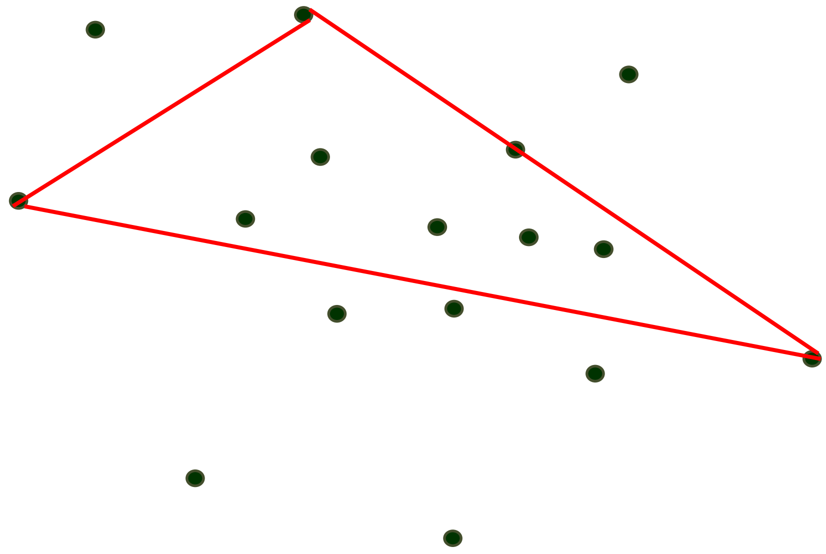
- To determine the next point on the convex hull a search for the point with the greatest distance from the dividing line is done.
- This point, together with the line start and end point forms a triangle.



Divide and Conquer (Quickhull)

Point Exclusion

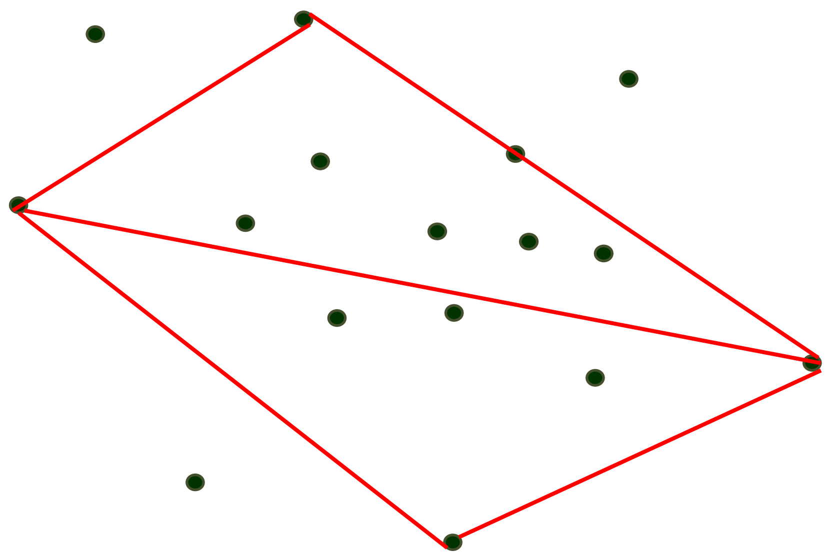
- All points inside this triangle can not be part of the convex hull polygon, as they are obviously lying in the convex hull of the three selected points.
- Therefore, these points can be ignored for every further processing step.



Divide and Conquer (Quickhull)

<u>Recursively Divide</u>

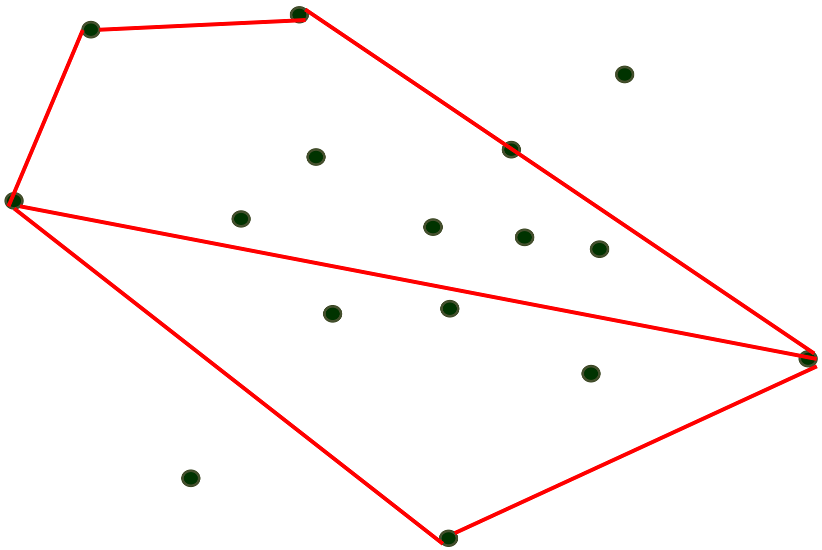
- Having this in mind the recursive processing can take place again.
- Everything right of the triangle is used as one subset, everything leftof it as anotherone.



Divide and Conquer (Quickhull)

<u>Recursively Divide</u>

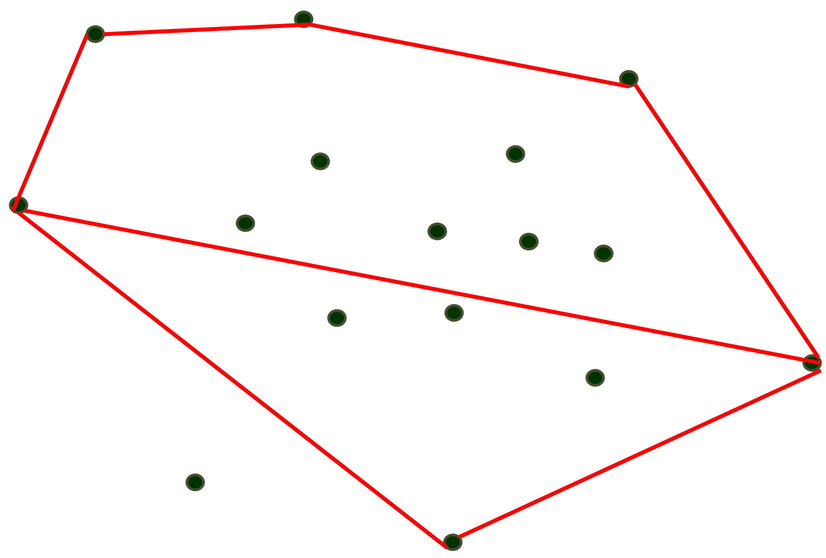
- Having this in mind the recursive processing can take place again.
- Everything right of the triangle is used as one subset, everything left of it as another one.



Divide and Conquer (Quickhull)

Recursively Divide

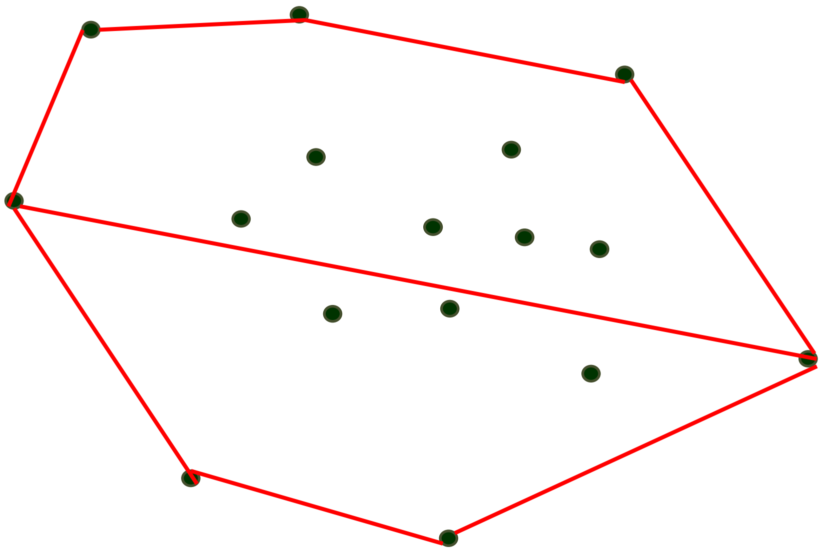
- Having this in mind the recursive processing can take place again.
- Everything right of the triangle is used as one subset, everything left of it as another one.



Divide and Conquer (Quickhull)

Recursively Divide

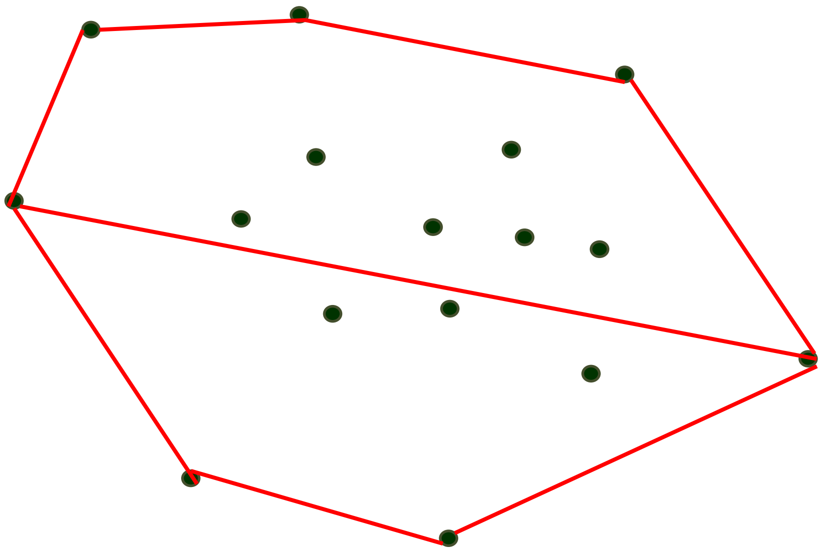
- Having this in mind the recursive processing can take place again.
- Everything right of the triangle is used as one subset, everything left of it as another one.



Divide and Conquer (Quickhull)

Recursively Divide

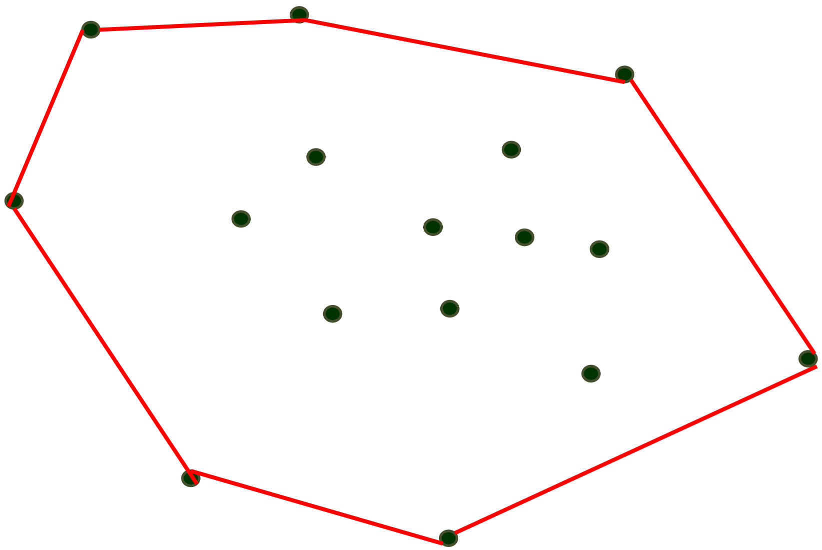
- Having this in mind the recursive processing can take place again.
- Everything right of the triangle is used as one subset, everything left of it as another one.



Divide and Conquer (Quickhull)

Abort Condition

- At some point the recursively processed point subset does only contain the start and end point of the dividing line.
- If this is case this line has to be a segment of the searched hull polygon and the recursion can come to an end.



Divide and Conquer (Quickhull)

Algorithm

Quick Hull(S)

//Find convex hull from the set S on n points Convex Hull= {}//

1. Find left and right most points, say A & B and add $\overline{AB}$ to Convex Hull.
2. Segment $\overline{AB}$ divides the remaining (n-2) points into two groups S_1 and S_2 . Where S_1 are points in S that are on the right side of the oriented line from $A\ to\ B$. And S_2 are points in S that are on the right side of the oriented line from $B\ to\ A$.
3. Find Hull (S_1 ,A, B)
4. Find Hull (S_2 ,B, A)

Divide and Conquer (Quickhull)

Algorithm

Find Hull(S_k ,P, Q)

//{Find points on Convex Hull from the set S_k points, that are on the right side of the oriented from P to Q}//

If S_k has no point

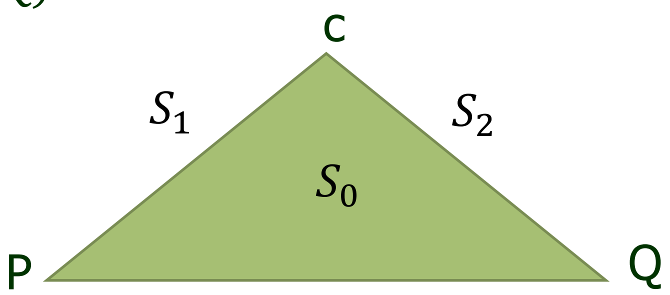
then return

- From the given set of points in S_k , find farthest point, say C. from segment PQ.
- Add point C to the Convex Hull at the location between P and Q. Three points P, Q, and C partition the remaining points of S_k into three subsets S_0 , S_1 , and S_2.

Divide and Conquer (Quickhull)

Algorithm

- Where S_0 are points inside triangle PCQ, and S_1 are points on the right side of the oriented line from P to C, and S_2 are the points on the right side of the oriented line from C to Q.
- Find Hull (S_1 ,P, C)
- Find Hull (S_2 ,C, Q)



Divide and Conquer (Quickhull)

Time Complexity of Quickhull

- The running time of Quickhull, as with QuickSort, depends on how evenly the points are split at each stage.
- If we assume that the points are ``evenly'' distributed, the running time will solve to $O(n \log n)$.
- if the splits are not balanced, then the running time can easily increase to $O(n^2)$.

Divide and Conquer (Quickhull)

Time Complexity of Quickhull

$$T(n) = T(l) + T(n-l) + \text{O}(n)$$

Where,

- $T(l) \longrightarrow$ Point in left side of AB.
- $T(n-l) \longrightarrow$ Point in right side of AB.
- $\text{O}(n) \longrightarrow$ To find the farthest point.

Assume that T(l) contain (n/2) points and T(n-l) contain (n/2) points.

Hence ,

$$T(n) = 2T\left(\frac{n}{2}\right) + \text{O}(n)$$

After applying Master Method

$T(n) = \Theta(n \lg n)$ in average case

$T(n) = \Theta(n^2)$ in worst case.

Thank u