Convex Hulls

Chapter 1 of the text

Chapter 3 of O'Rourke book

Convex Hulls

- Convex hulls are to CG what sorting is to discrete algorithms
- First order shape approximation

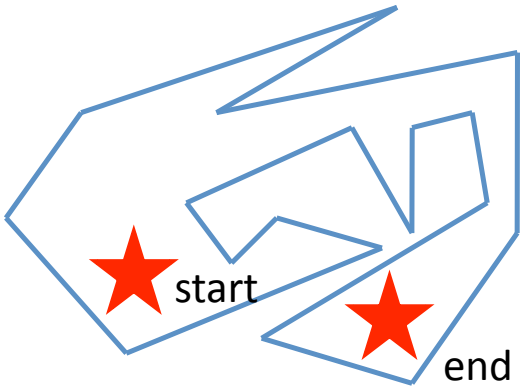
The shape approximated is invariant under rotation and translation

- Rubber band analogy
- Many applications in robotics, shape analysis, line fitting.

Convex Hulls (applications)

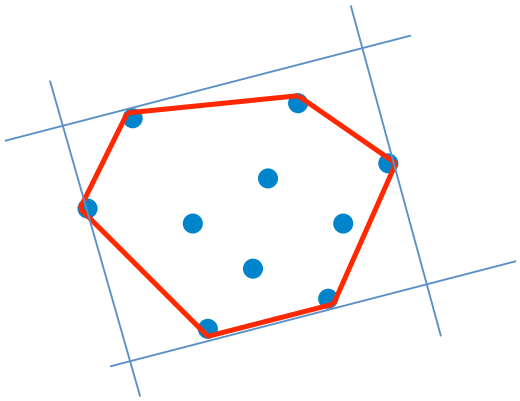
- **Collision Avoidance:**
  - Robot moving in a polygonal environment
  - Can the robot ★ be moved from the start position to the end position
    - Only translation
    - Translation and rotation

Observation: If the convex hull of the robot avoids collision with obstacles, so does the robot.



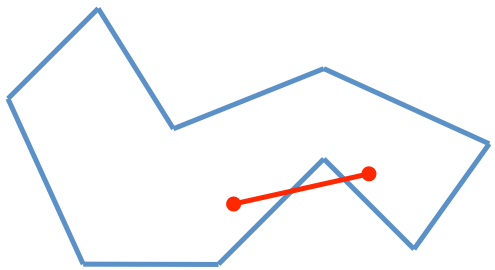
Convex Hulls (applications)

- **Smallest Box**
  - Given a set of points, determine the smallest area rectangle (not necessarily axis-aligned) that encloses the point set.
    - Needs to enclose the convex hull of the points
    - One of the sides of an optimal rectangle must be aligned with a convex hull edge

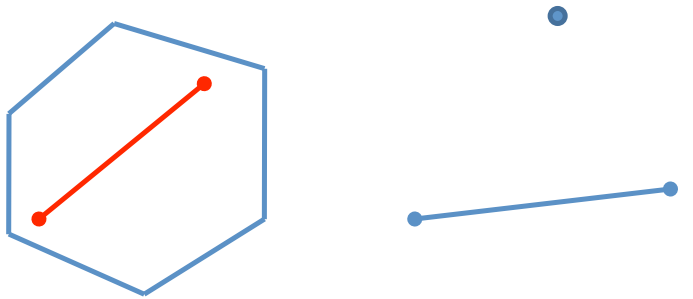


Convex Hulls

- **Convex Hull** Problem: Given a finite set of points S, compute its convex hull CH(S)
- A set is **convex** if every line segment connecting two points in the set is fully contained in the set



Non-convex



Convex

Convex Hull

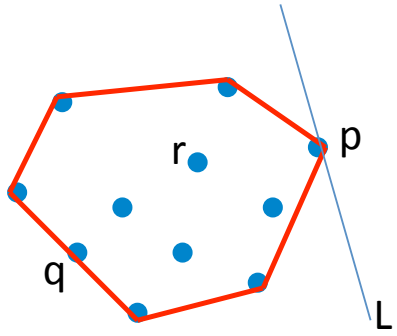
- What is a convex hull of a set of points S?
  - Smallest area/perimeter convex set containing S
  - Union of all points expressible by a convex combination of points in S, i.e points of the form:

$$\sum_{p \in S} c_p \cdot p \quad , c_p \geq 0 \ and \sum_{p \in S} c_p = 1$$

- The definition is not suitable for an algorithm

Convex Hulls

- **<u>Extreme Point:</u>** p is an extreme point of S if p can not be expressed as a convex combination of S – {p}
- Or: p is an extreme point of S if p admits a line of support of S

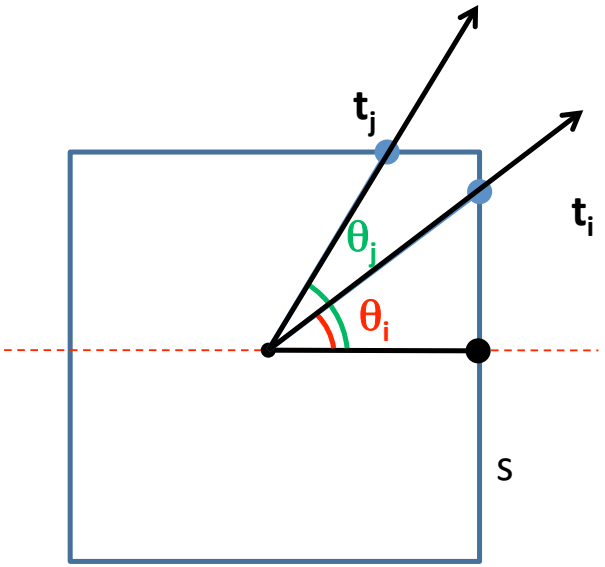


- p is an extreme point
- q is a boundary point
- r is an interior point
- L is the line of supported admitted by p

Computational Problem

- Given $S \subset R^2$, $|S| = n$, find the description of CH(S)
  - CH(S) is a convex polygon with at most n vertices
  - Want to find these (extreme) vertices in counterclockwise order.
  - Assume all points are distinct (otherwise sort and remove duplicates)

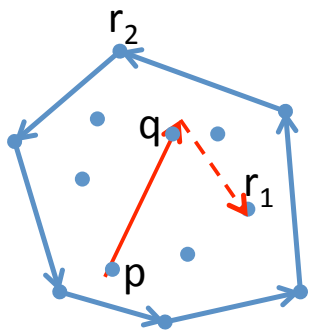
Comparing Two Angles



- For each edge determine the point of intersection with a unit length box
- Measure perimeter length $L(s \rightarrow t_i)$ and $L(s \rightarrow t_j)$
- $\theta_i < \theta_j$ if and only if $L(s \rightarrow t_i) < L(s \rightarrow t_j)$

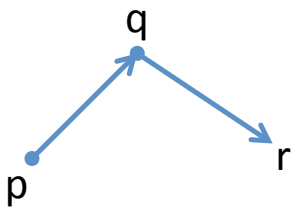
Left-Right Turn

- How does one test whether r is to the left or right of pq?

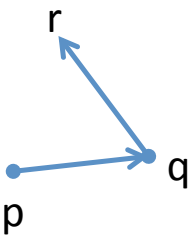


$$Let \quad D = \begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix} = \begin{vmatrix} 1 & p_x & p_y \\ 0 & q_x - p_x & q_y - p_y \\ 0 & r_x - p_x & r_y - p_y \end{vmatrix}$$

If D < 0
Right Turn



If D > 0
Left Turn



If D > 0
Collinear

Better Algorithms

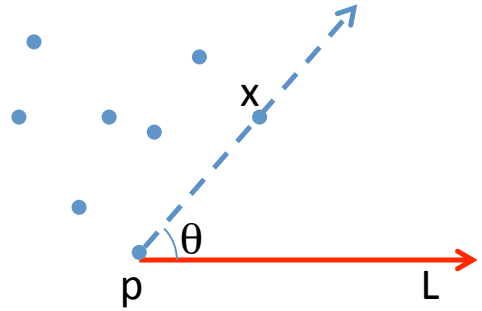
- R.A Jarvis, ***On the Identification of the Convex Hull of a Finite Set of Points in the Plane,*** Information Processing Letters, Vol. 2, pp 18-21, 1973.

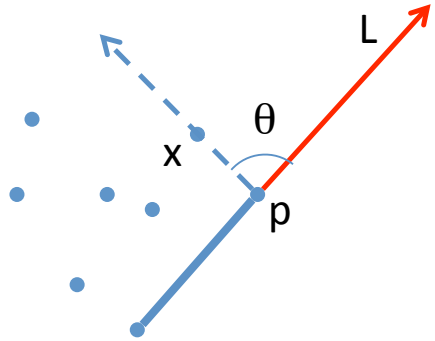
- S.G. Akl and G.T. Toussaint, ***A Fast Convex Hull Algorithm,*** Information Processing Letters, Vol.7, No. 5, pp 219-222, 1978.

- R.L. Graham, ***An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set***, Information Processing Letters, Vol. 1, pp. 132-133, 1972

Efficient Convex Hull Algorithm

- **Gift-Wrapping** [Jarvis 73] [Chand and Kapur 70]
  1. Start with the bottom extreme point p
  2. $p_{start} = p$
  3. Let L be the horizontal ray incident on p
  4. Determine the point x with the smallest counterclockwise angle with L at p
     - px is a convex hull edge
  5. Set L = ray at x containing to px
  6. Set p = x
  7. Repeat from step 4until $p = p_{start}$





- This algorithm is also known as **rope-fence** method

Gift-Wrapping (Left-Right Test Only)

/* Let S = $\{p_1, p_2, \ldots, p_n\}$;

$z_{(i)}$ = point with the smallest CCW angle $(p_-, p, z_{(i)})$ among $\{p_1, p_2, \ldots, p_i\}$, the first i points of the set. */

$Z_{(1)} = p_1$; (assuming $p_1 \neq p$, $p_1 \neq p_-$)

For i = 2 to n ; ($p_i \neq p$ **and** $p_i \neq p$) {

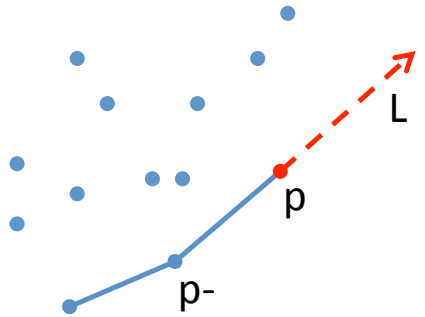
If (p, $z_{(i-1)}$, p_i) is a right turn)

$z_{(i)} = p_i$;

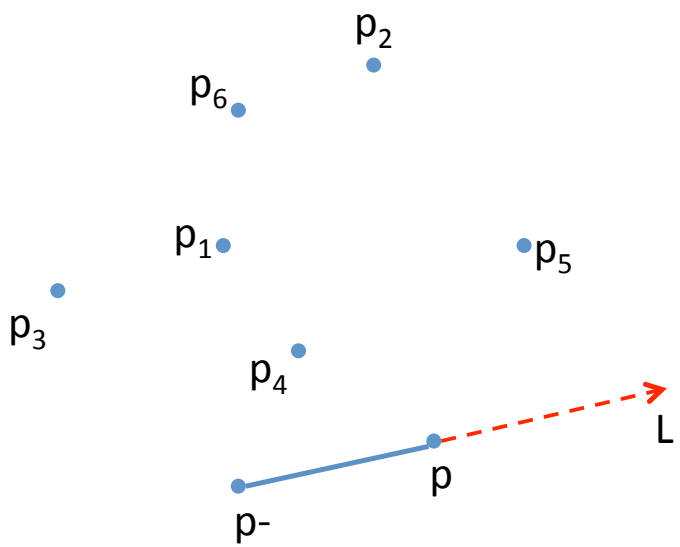
else

$z_{(i)} = z_{(i-1)}$;

}

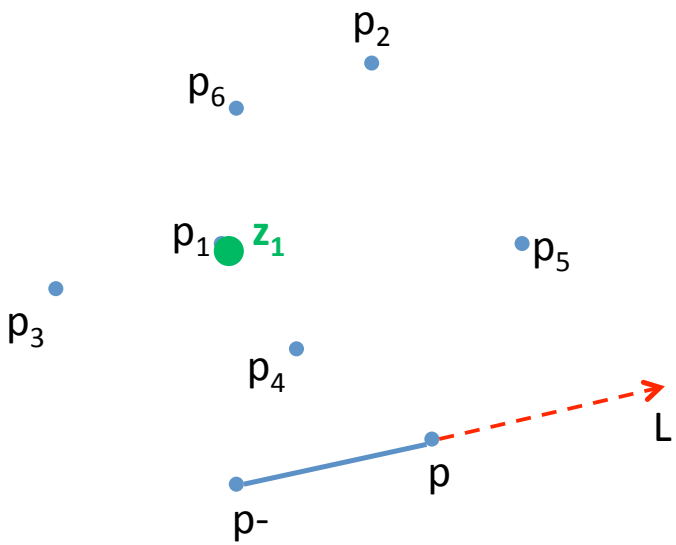


Gift-Wrapping (Example)

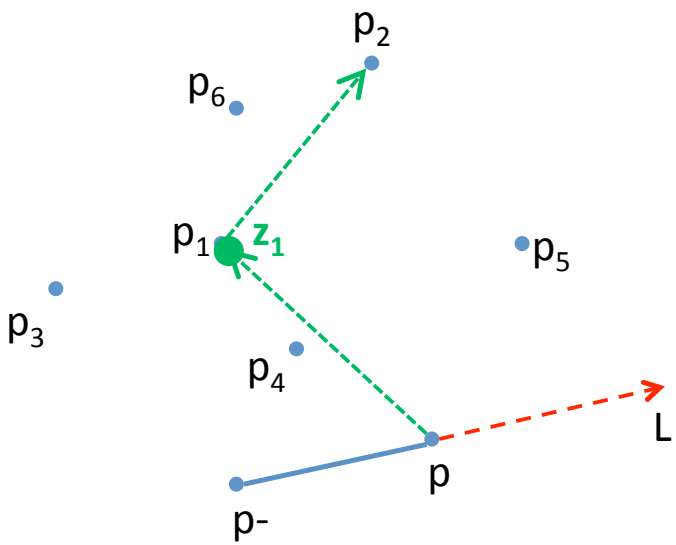


Gift-Wrapping (Example)

- $z_1 = p_1$
- $i = 2$

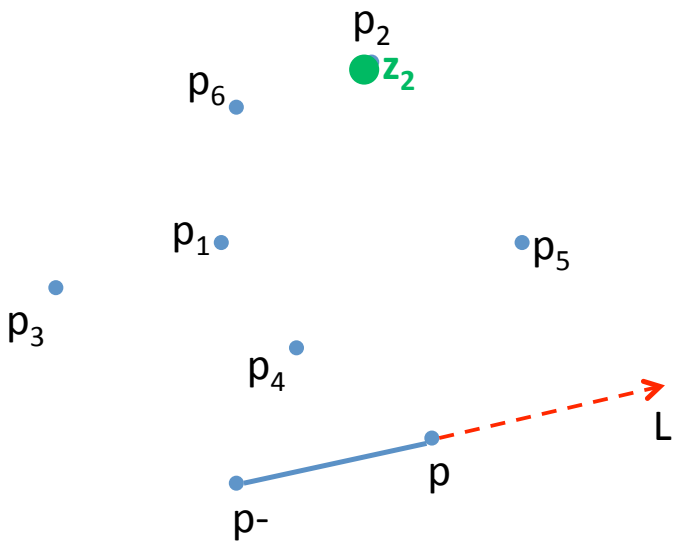


Gift-Wrapping (Example)



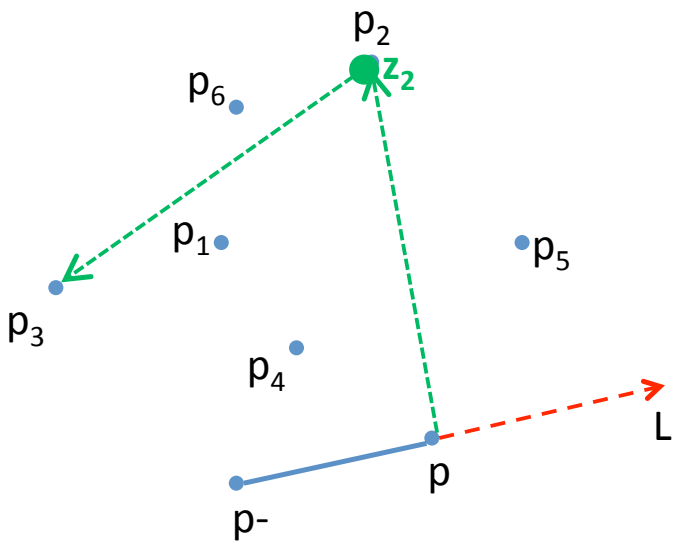
- $z_1 = p_1$
- $i = 2$
- **$(p, z_{(i-1)}, p_i)$ => Right Turn**

Gift-Wrapping (Example)



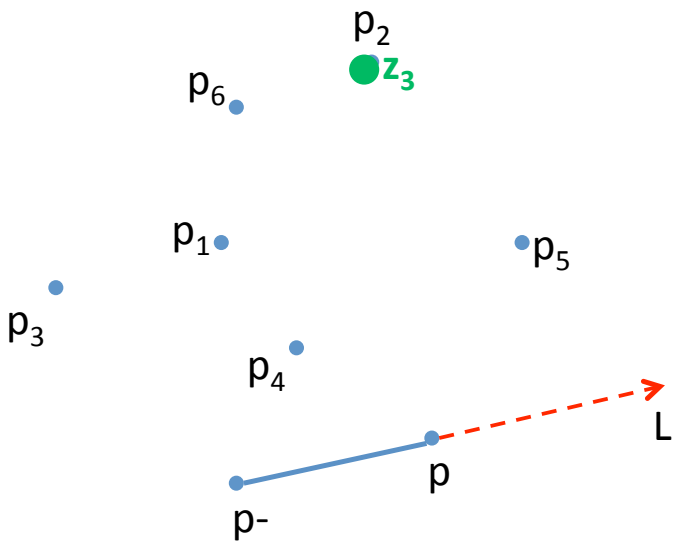
- $z_1 = p_1$
- i = 2
- (p, $z_{(i-1)}$, p_i) => Right Turn
  - **$z_2 = p_2$**
- **i = 3**

Gift-Wrapping (Example)



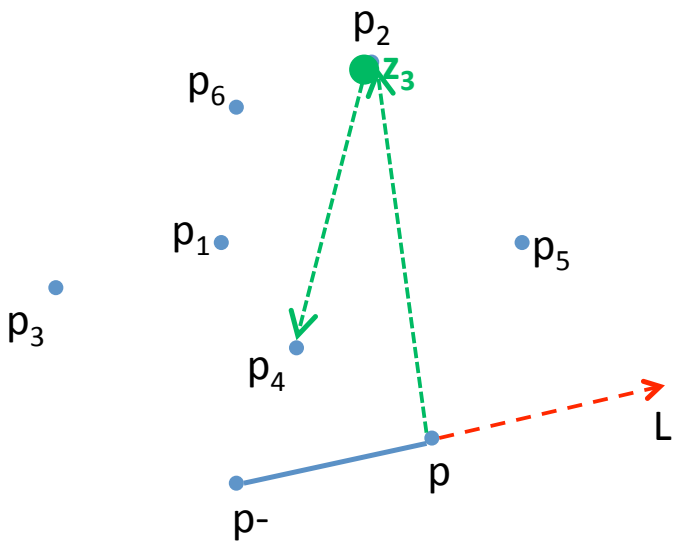
- $z_1 = p_1$
- i = 2
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_2 = p_2$
- i = 3
- **$(p, z_{(i-1)}, p_i)$ => Left Turn**

Gift-Wrapping (Example)



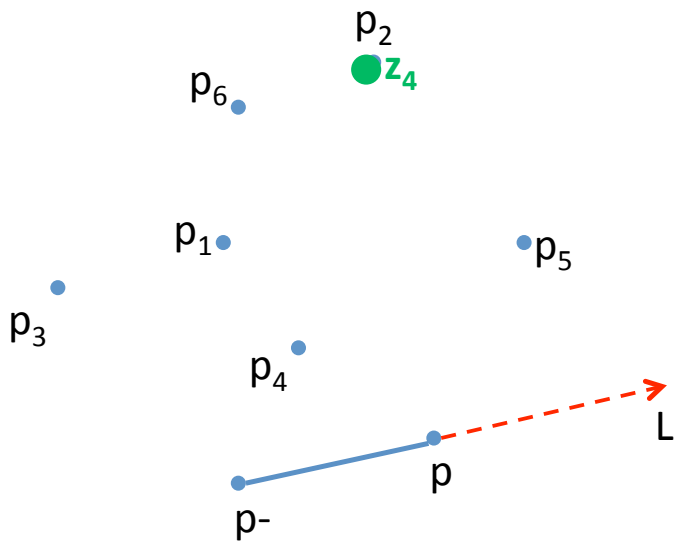
- $z_1 = p_1$
- i = 2
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_2 = p_2$
- i = 3
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - **$z_3 = z_2$**
- **i = 4**

Gift-Wrapping (Example)



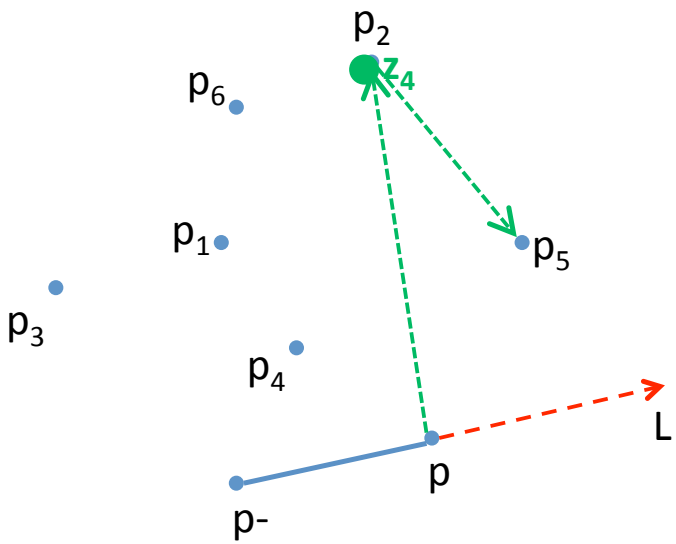
- $z_1 = p_1$
- i = 2
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_2 = p_2$
- i = 3
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_3 = z_2$
- i = 4
- **$(p, z_{(i-1)}, p_i)$ => Left Turn**

Gift-Wrapping (Example)



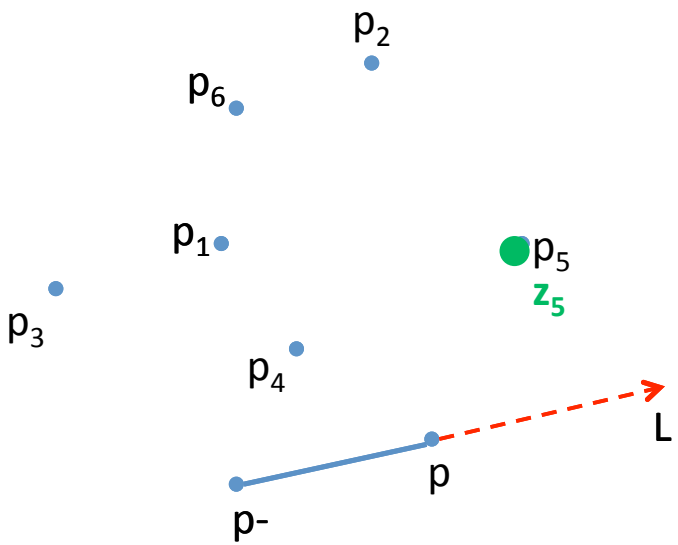
- $z_1 = p_1$
- i = 2
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_2 = p_2$
- i = 3
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_3 = z_2$
- i = 4
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - **$z_4 = z_3$**
- **i = 5**

Gift-Wrapping (Example)



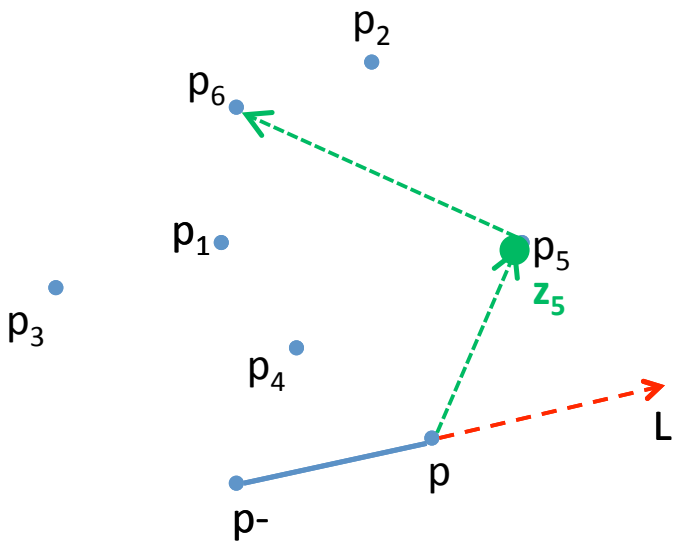
- $z_1 = p_1$
- i = 2
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_2 = p_2$
- i = 3
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_3 = z_2$
- i = 4
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_4 = z_3$
- i = 5
- **$(p, z_{(i-1)}, p_i)$ => Right Turn**

Gift-Wrapping (Example)



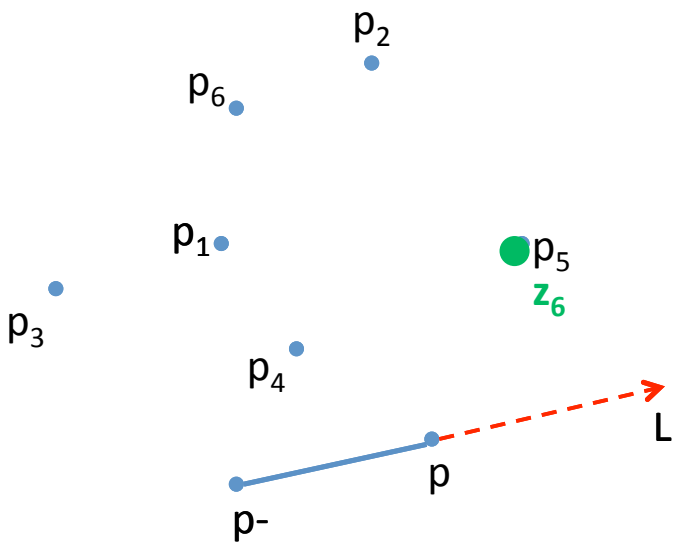
- $z_1 = p_1$
- i = 2
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_2 = p_2$
- i = 3
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_3 = z_2$
- i = 4
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_4 = z_3$
- i = 5
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - **$z_5 = p_5$**
- **i = 6**

Gift-Wrapping (Example)



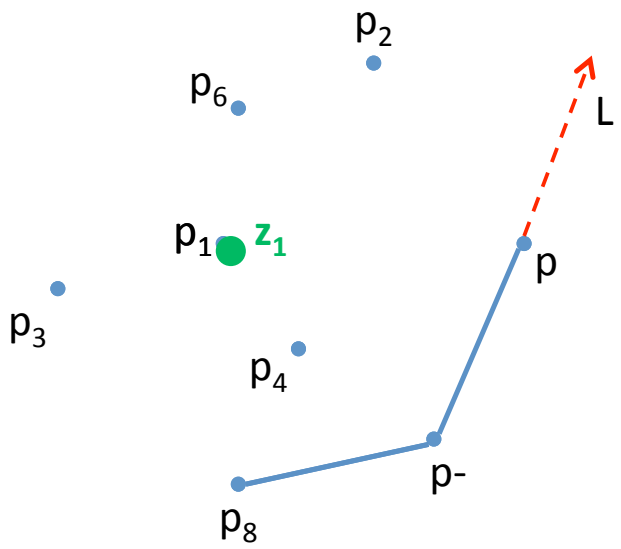
- $z_1 = p_1$
- i = 2
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_2 = p_2$
- i = 3
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_3 = z_2$
- i = 4
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_4 = z_3$
- i = 5
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_5 = p_5$
- i = 6
- **$(p, z_{(i-1)}, p_i)$ => Left Turn**

Gift-Wrapping (Example)



- $z_1 = p_1$
- i = 2
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_2 = p_2$
- i = 3
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_3 = z_2$
- i = 4
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - $z_4 = z_3$
- i = 5
- $(p, z_{(i-1)}, p_i)$ => Right Turn
  - $z_5 = p_5$
- i = 6
- $(p, z_{(i-1)}, p_i)$ => Left Turn
  - **$z_6 = z_5$**

Gift-Wrapping (Example)

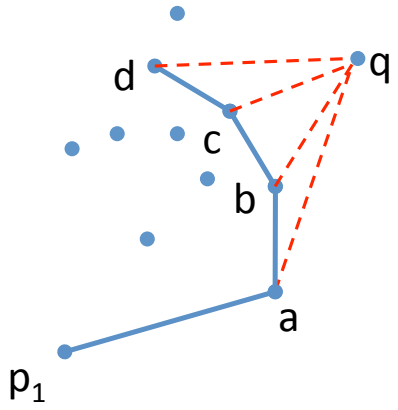


Complexity of Gift-Wrapping (Jarvis)

- Let h = number of extreme points
  - $3 \leq h \leq n$
- Complexity = O (nh)
  - Worst Case Complexity = $O(n^2)$ h = n
  - Best Case Complexity = O(n) h = constant
- Jarvis' algorithm is output-sensitive; the running time is dependent on the size of the output.

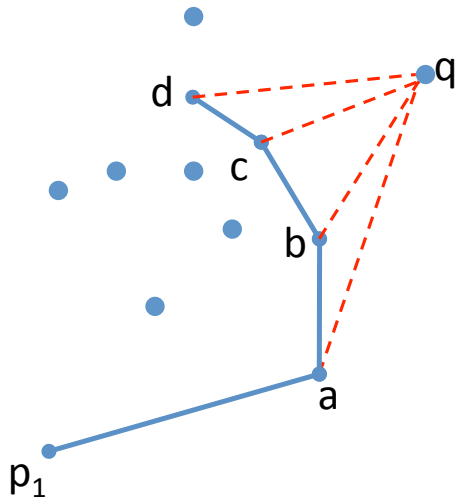
Graham's Algorithm

- In Jarvis' algorithm all points are treated equally. Here we eliminate a point if it is found to be inside
- **Algorithm:**
  - Sort S by y-coordinates; p1, p2, …, pn
  - Initialize : Push(p1), Push(p2)
  - For i = 3 to n do
    - While((next, top, pi) is a right turn)
      Pop()
    - Push(p_i)
  - Return Stack (giving the right convex hull chain)
  - Similarly compute the left convex hull chain

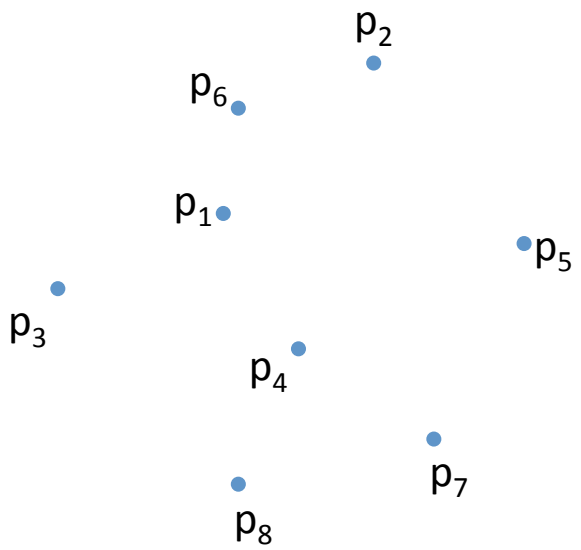


Invariant

- When Pi is being considered the stack gives the right chain of the convex hull of $\{p_1, p_2, \ldots, p_{i-1}\}$

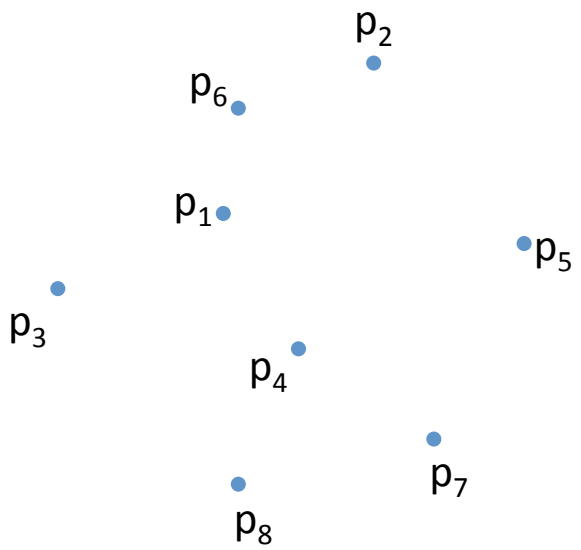


Graham's Algorithm



Stack

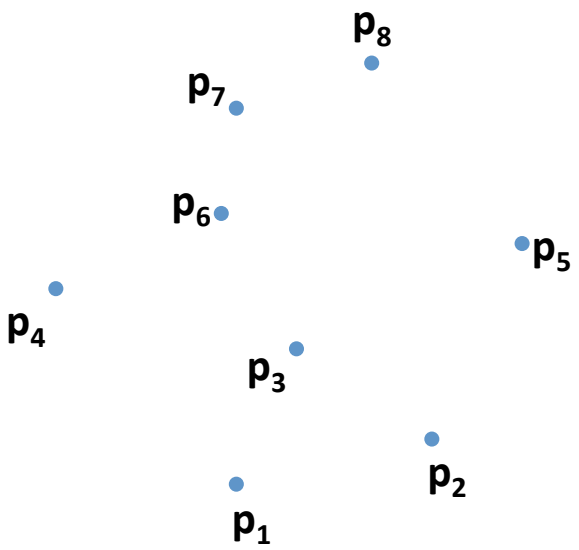
Graham's Algorithm



Stack

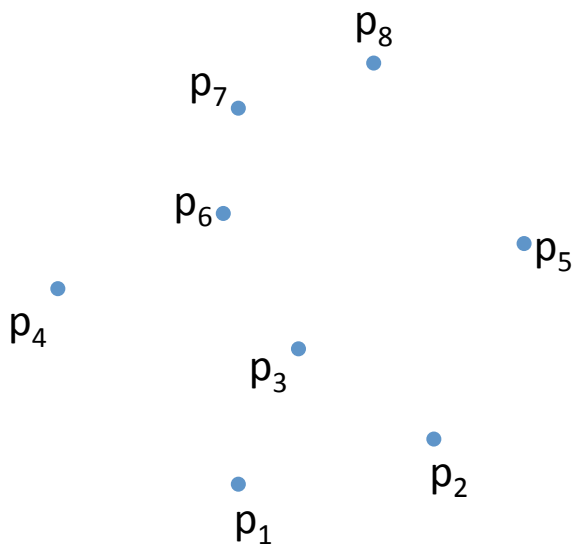
SORT

Graham's Algorithm



Stack

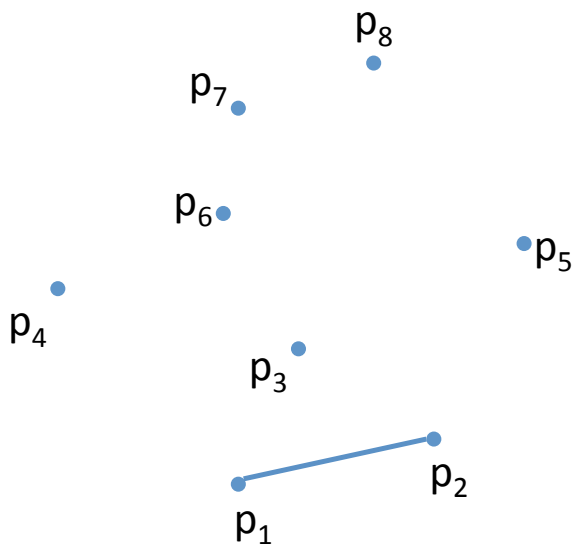
Graham's Algorithm



Stack

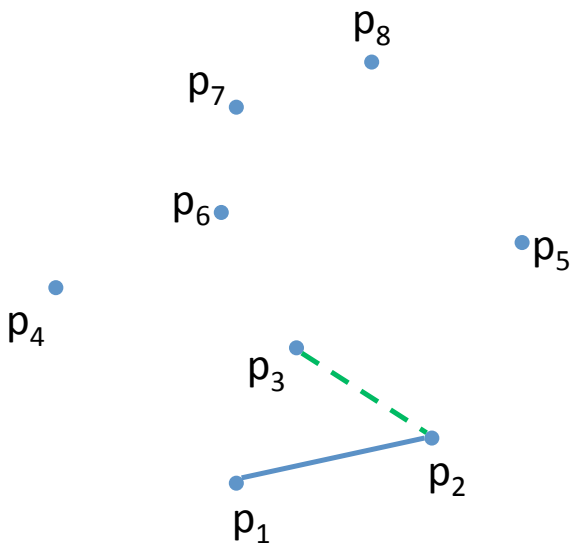
Push p_1 and p_2

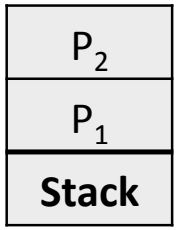
Graham's Algorithm



Stack
P_2
P_1

Graham's Algorithm

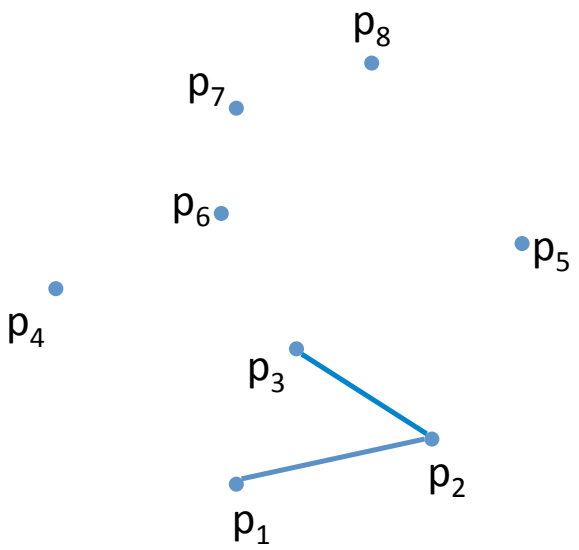




P_1, P_2, P_3 is a left turn
Push (p_3)

i = 3

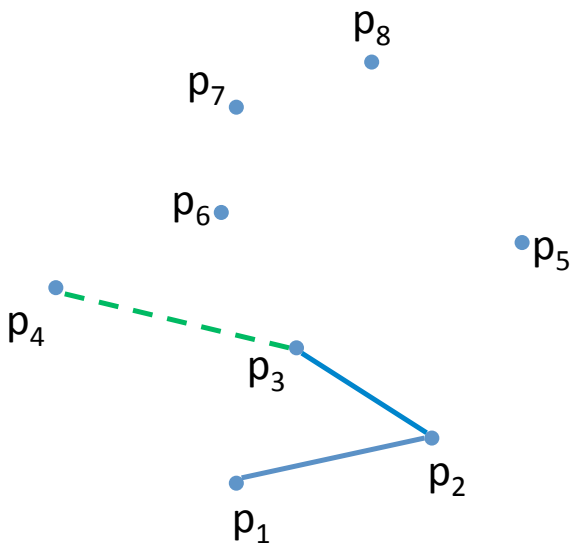
Graham's Algorithm



P3
P2
P1
Stack

i = 3

Graham's Algorithm

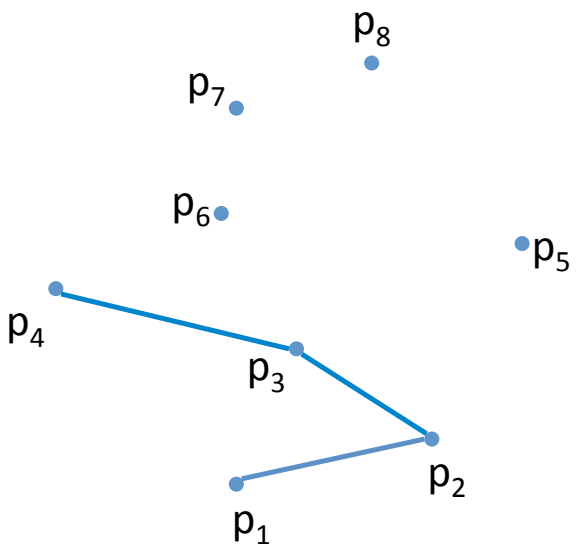


P_3
P_2
P_1
Stack

P_2, P_3, P_4 is a left turn
Push (p_4)

i = 4

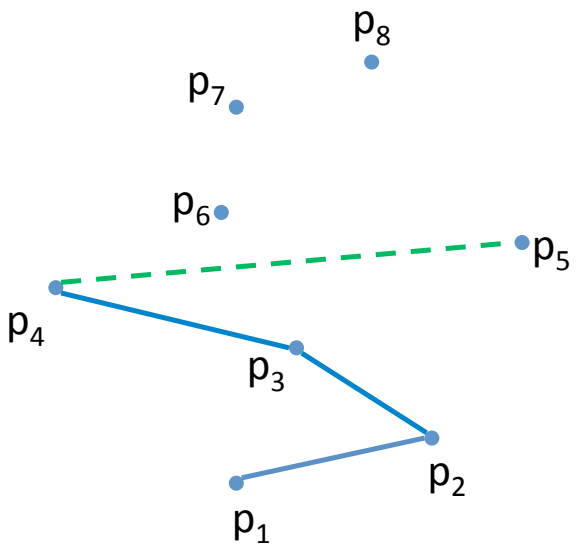
Graham's Algorithm



P_4
P_3
P_2
P_1
Stack

i = 4

Graham's Algorithm

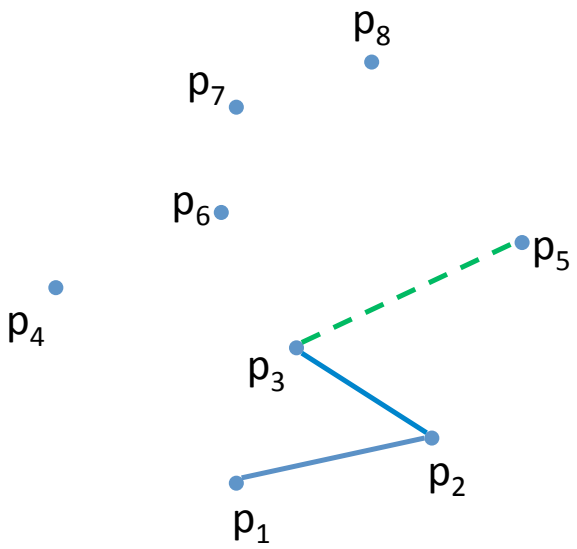


P_4
P_3
P_2
P_1
Stack

P_3, P_4, P_5 is a <u>right</u> turn
Pop()

i = 5

Graham's Algorithm

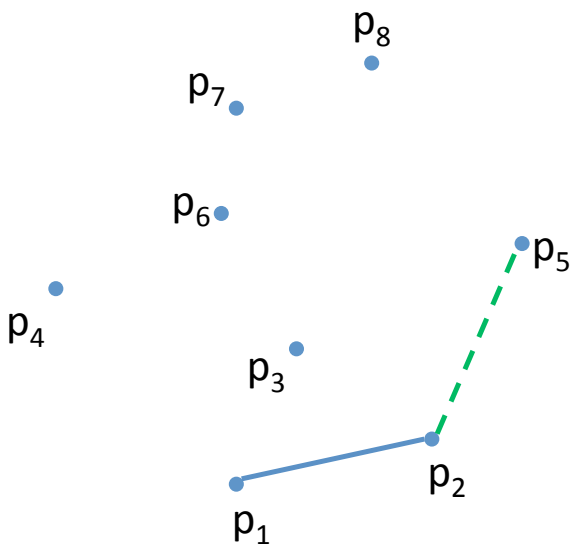


P_3
P_2
P_1
Stack

P_2, P_3, P_5 is a <u>right</u> turn
Pop()

i = 5

Graham's Algorithm

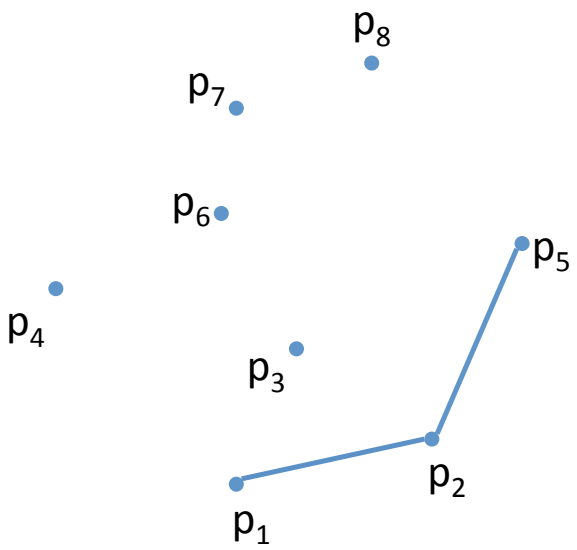


P_1, P_2, P_5 is a Left turn
Push(P_5)

P_2
P_1
Stack

i = 5

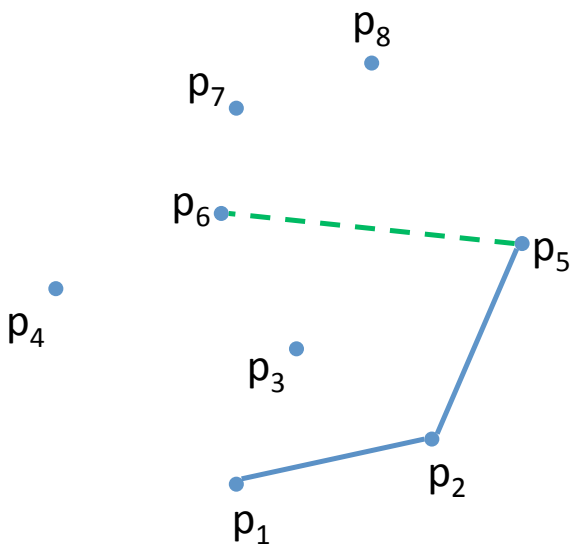
Graham's Algorithm

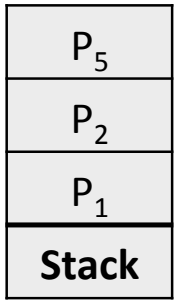


P_5
P_2
P_1
Stack

i = 5

Graham's Algorithm

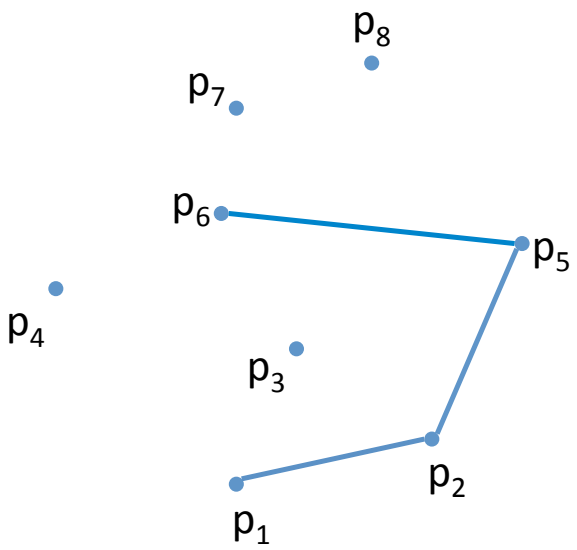




P_2, P_5, P_6 is a Left turn
Push(P_6)

i = 6

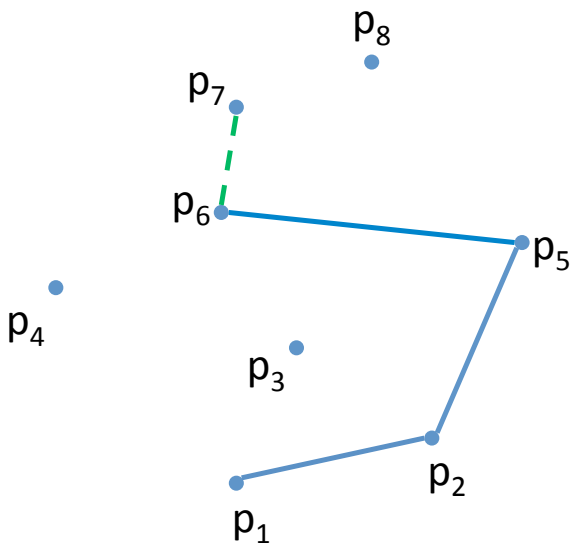
Graham's Algorithm



P6
P5
P2
P1
Stack

i = 6

Graham's Algorithm

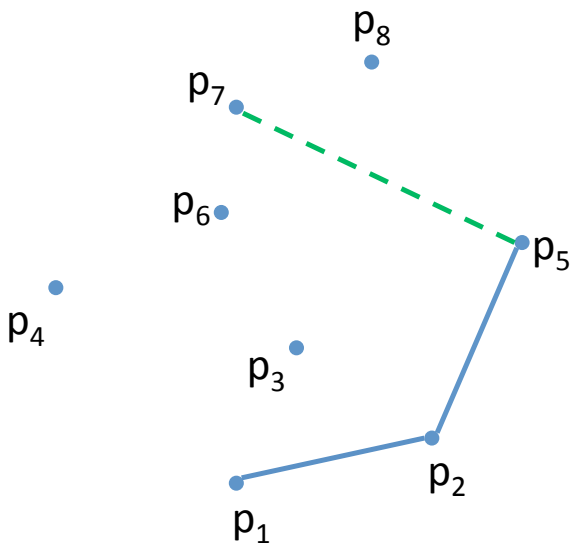


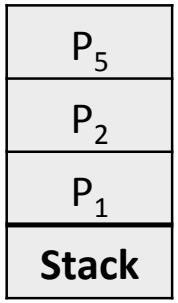
P_6
P_5
P_2
P_1
Stack

P_5, P_6, P_7 is a <u>right</u> turn
Pop()

i = 7

Graham's Algorithm

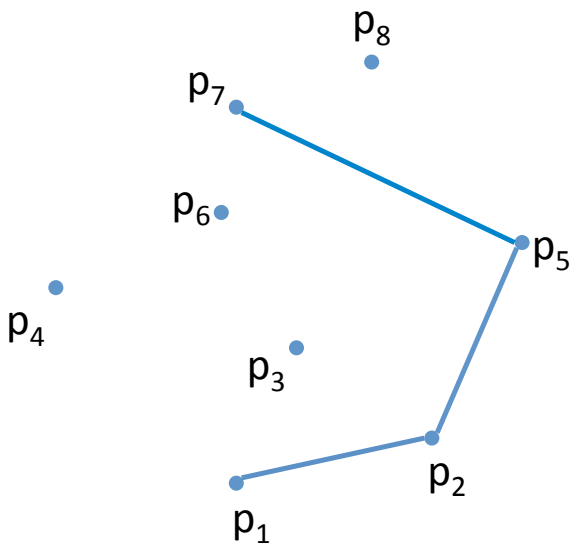




P_2, P_5, P_7 is a left turn
Push(P_7)

i = 7

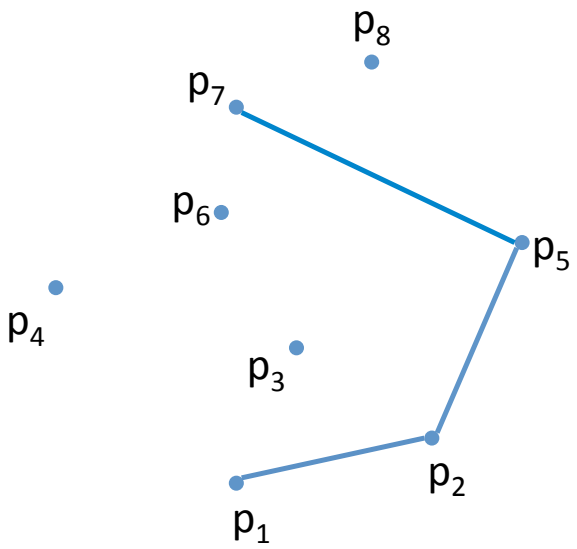
Graham's Algorithm



P_7
P_5
P_2
P_1
Stack

i = 7

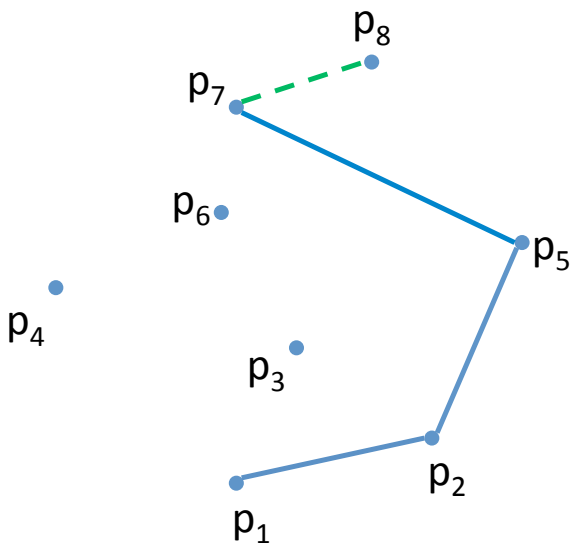
Graham's Algorithm



P_7
P_5
P_2
P_1
Stack

i = 7

Graham's Algorithm

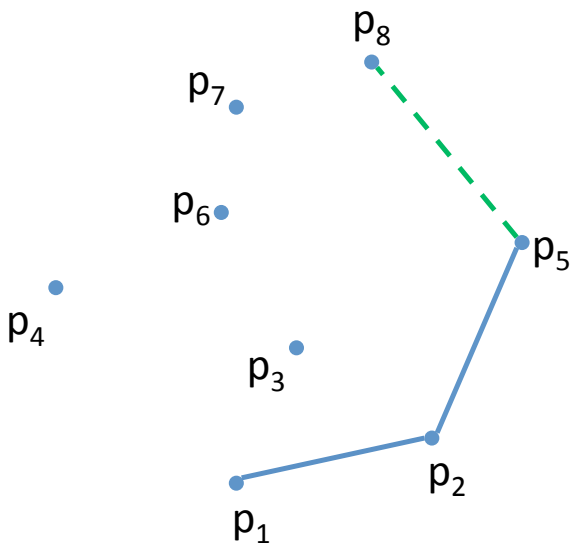


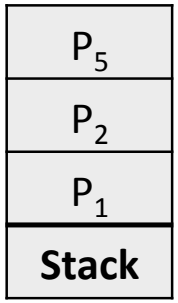
P_7
P_5
P_2
P_1
Stack

P_5, P_7, P_8 is a <u>right</u> turn
Pop()

i = 8

Graham's Algorithm

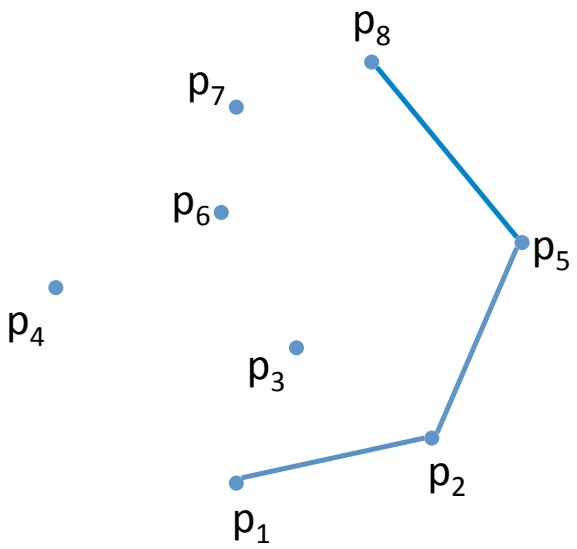




P_2, P_5, P_8 is a left turn
Push(P_8)

i = 8

Graham's Algorithm



P_8
P_5
P_2
P_1
Stack

P_2, P_5, P_8 is a left turn
Push(P_8)

i = 8

Analysis of Graham's Algorithm

- Invariant: $<p_1, \ldots, \text{top(stack)}>$ is convex
- Left-right test can be performed in O(1) time
- After sorting, the scan takes O(n) time:
  - In each step either a point is deleted or added to the stack

- Total Time: O(nlogn)

Lower Bound

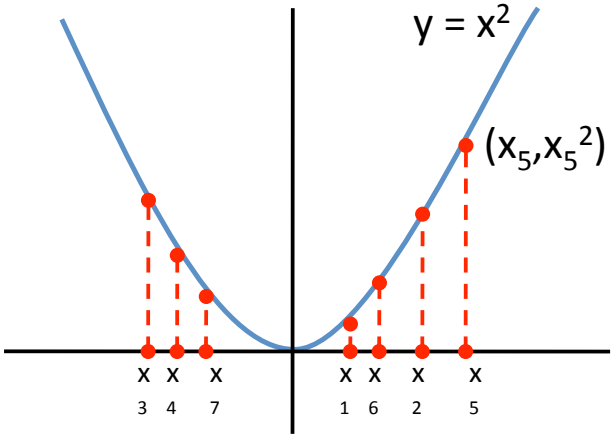
- Identifying one extreme edge at a time:
  - $O(n^3)$
- Gift-Wrapping/Jarvis' Algorithm:
  - O(nh)
  - Worst case: $O(n^2)$
- Quick Hull:
  - Expected : O(nlogn)
  - Worst Case: $O(n^2)$
- Graham's Algorithm
  - Worst Case: O(nlogn)

Lower Bound

- Can we do better?
- NO: worst case running time of any convex hull algorithm is $\Omega(n \log n)$
- Idea of proof:
  - If the answer was ‘yes’ we could sort n points in less than $\Omega(n \log n)$ time

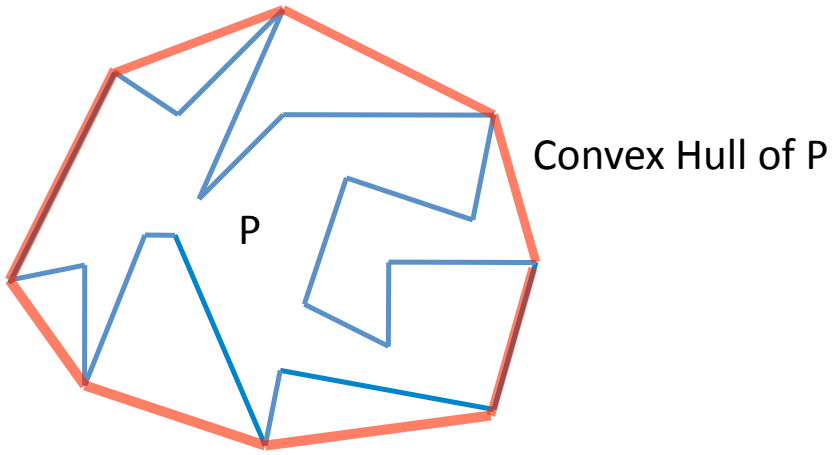
Lower Bound

- Reduce sorting to convex hull
- List of numbers to sort $\{x_1, x_2, \ldots, x_n\}$
- Create points $p_i = (x_i, x_i^2)$ for each I
- Compute the convex hull of $\{p_1, p_2, \ldots, p_n\}$.
- The boundary of the convex hull of realize the points in sorted order
- Therefore the worst case running time of any algorithm to compute the convex hull of n points must take Ω (nlogn) time.



Convex Hull of a Simple Polygon

- **Problem:** Given a simple polygon P, determine its convex hull.



- O(nlogn) algorithm is easy
- Linear time algorithm is tricky but can be designed.
- **Note:** Ω(nlogn) lower bound does not apply since the points of P are not unorganized points

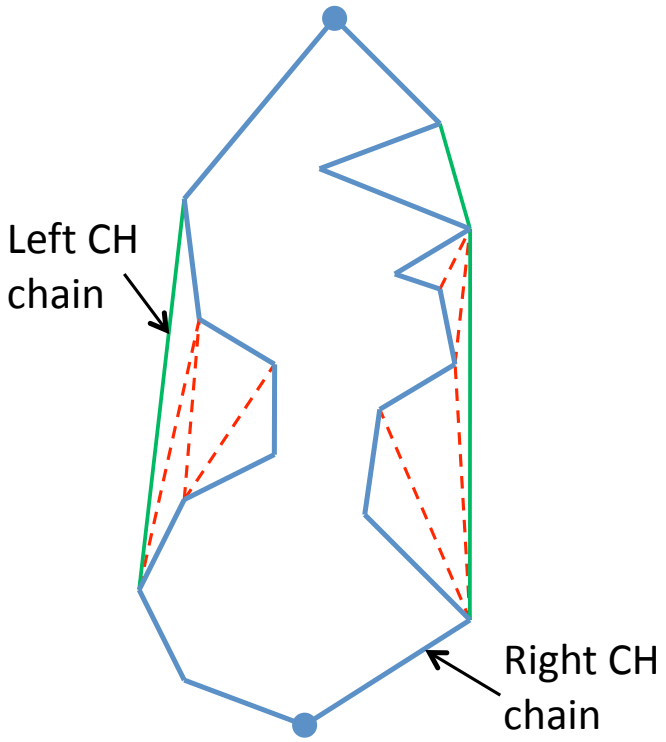
Convex Hull of a Simple Polygon in Linear Time

- Lots of published results (some of them are wrong). Please visit http://cgm.cs.mcgill.ca/~athens/cs601/ "A History of Linear-time Convex Hull Algorithms for Simple Polygons"
- The algorithm of Melkman is considered to be the simplest

- A. Melkman, ***"On-line Construction of the Convex Hull of a Simple Polyline",*** Information Processing Letters, Vol. 24, pp. 11-12, 1987

Monotone Polygon

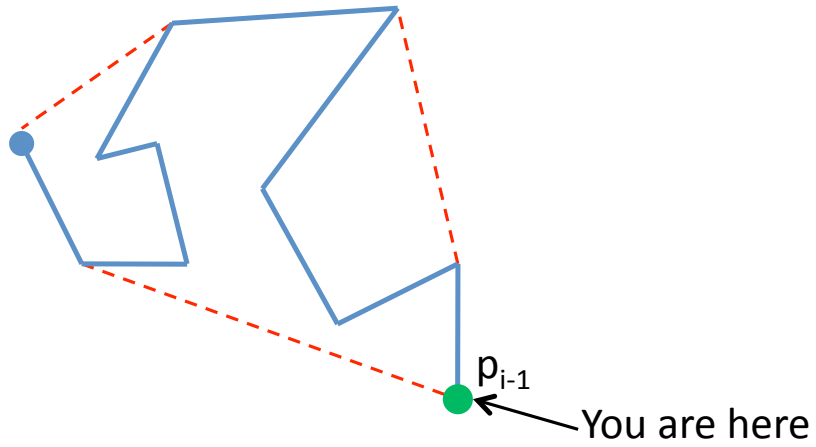
- Apply Graham scan on the right polygonal chain to obtain the right convex hull chain
- Apply Graham scan on the left polygonal chain to obtain the left convex hull chain.



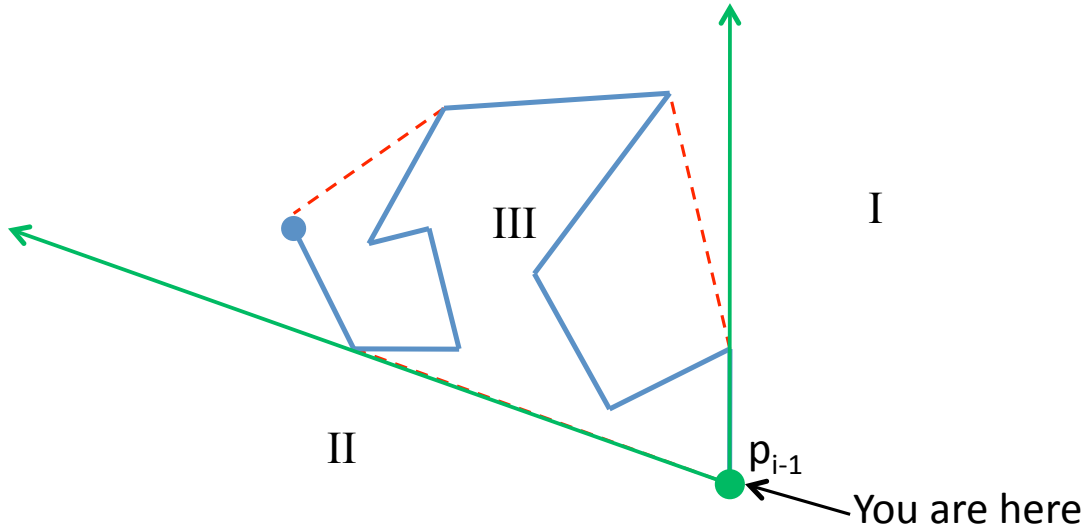
Simple Polygon

- We will consider the vertices in the given order and use an incremental approach very similar to Graham scan algorithm.
- Suppose we have the convex hull of $\{p_1, p_2, \ldots, p_{i-1}\}$. How should we update it to get the convex hull of $\{p_1, p_2, \ldots, p_{i-1}, p_i\}$ i.e. convHull { convHull $\{p_1, p_2, \ldots, p_{i-1}\}$, p_i } ?

Simple Polygon

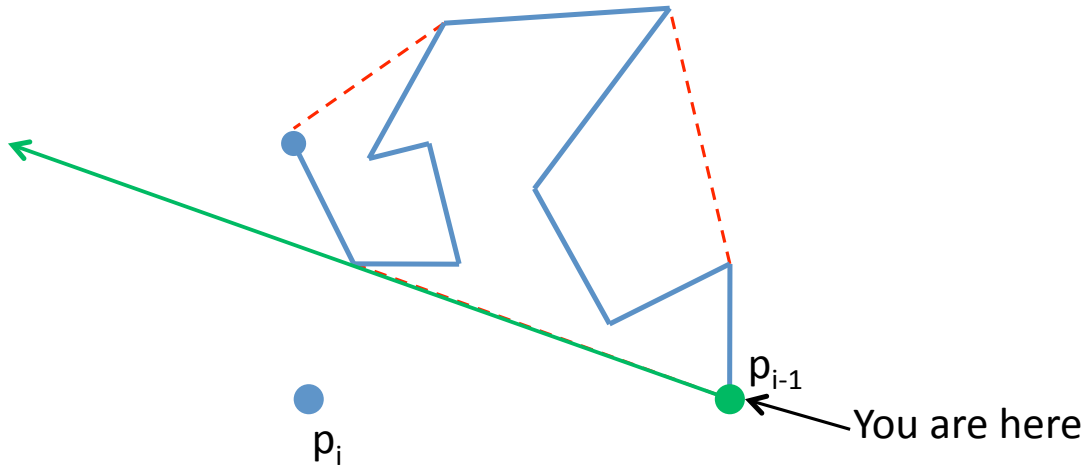


Simple Polygon



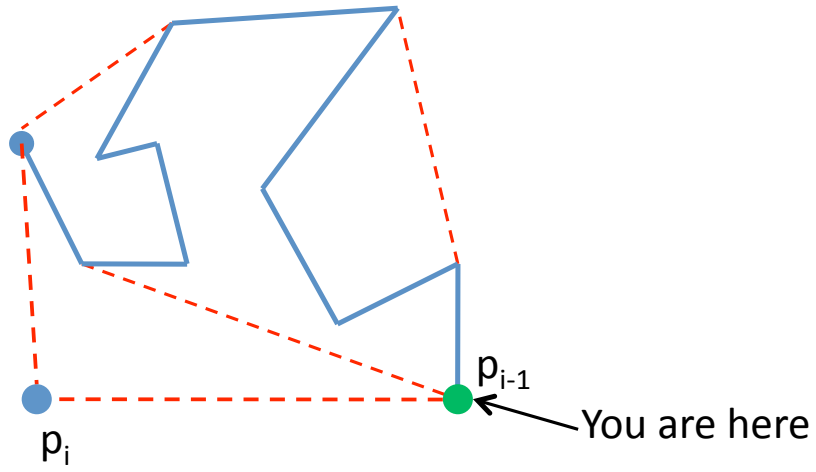
- The next point p_i lies in one of the regions I, II, or III

Simple Polygon



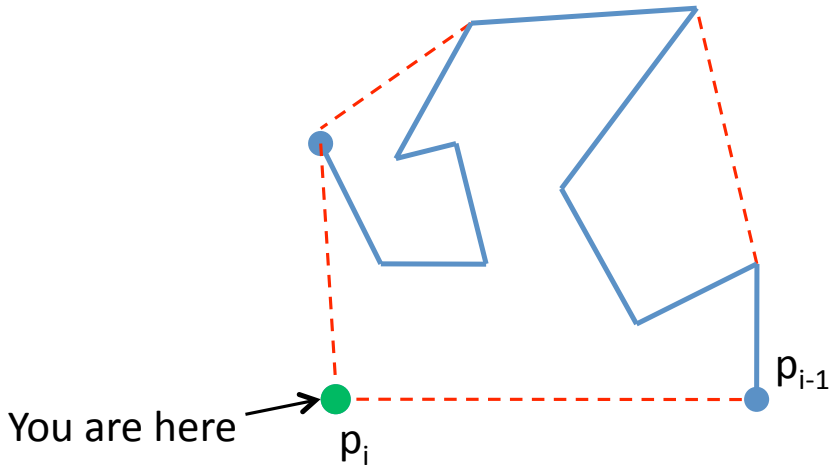
- **P_i in region $\mathbf{I}$ or $\mathbf{II}$**
  - Update the convex hull of $\{p_1, \ldots, p_{i-1}\}$
  - Move "you are here" pointer to p_i

Simple Polygon



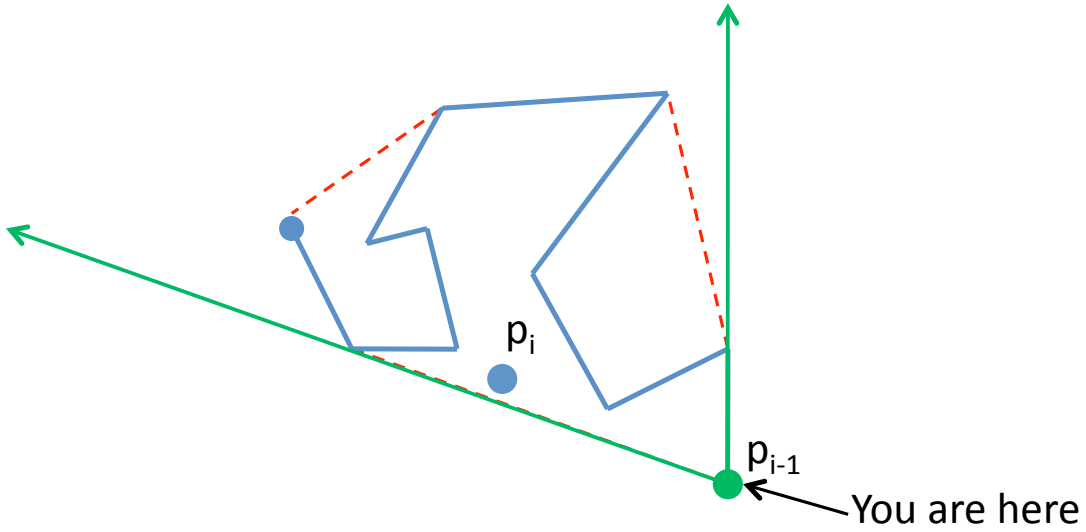
- **P_i in region $\mathbf{I}$ or $\mathbf{II}$**
  - Update the convex hull of $\{p_1, \ldots, p_{i-1}\}$
  - Move "you are here" pointer to p_i

Simple Polygon



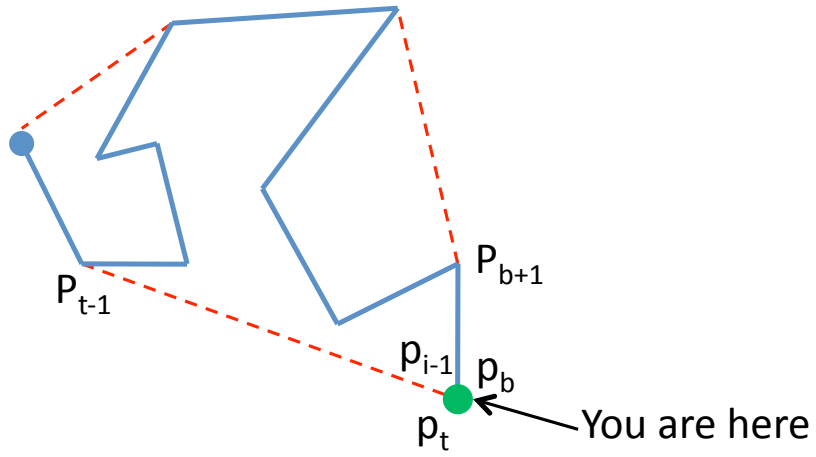
- **P_i in region $\mathbf{I}$ or $\mathbf{II}$**
  - Update the convex hull of $\{p_1, \ldots, p_{i-1}\}$
  - Move "you are here" pointer to p_i

Simple Polygon



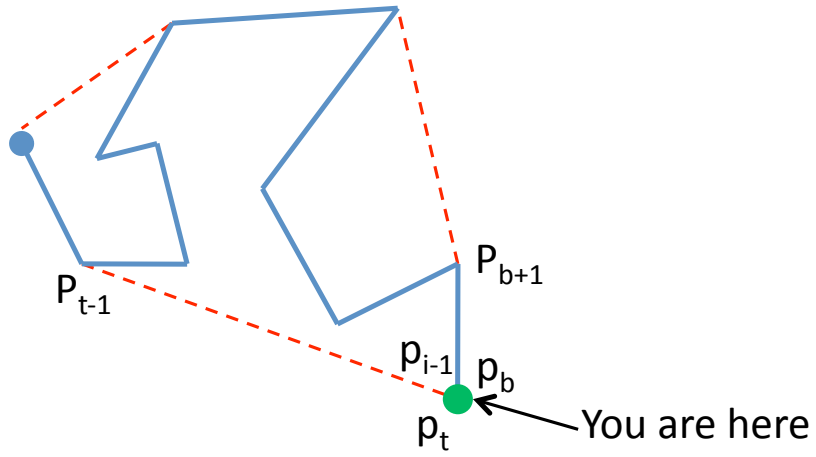
- **P_i in region III**
  - Convex hull does not change
  - “You are here” pointer does not change

Simple Polygon



- Use a deque data structure:
  - D = <p_b, p_{b+1}, …, p_{t-1}, p_t >
  - Four primitive operations:
    - PopBottom(D), PopTop(D), PushBottom(p, D), PushTop(p, D)

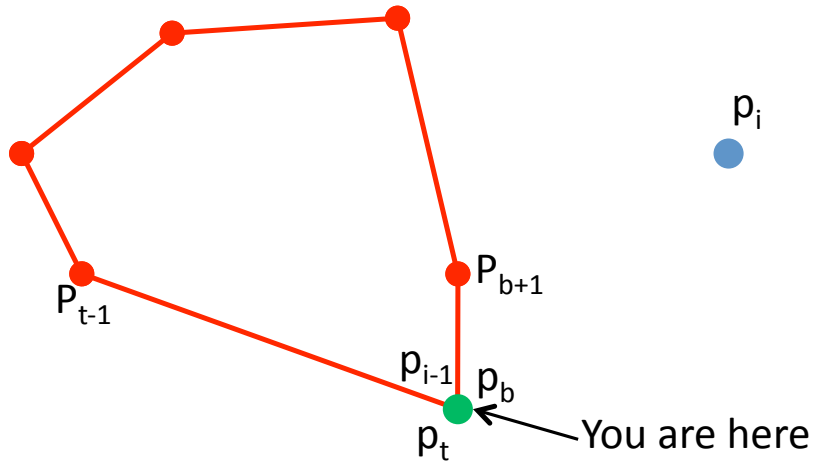
Simple Polygon



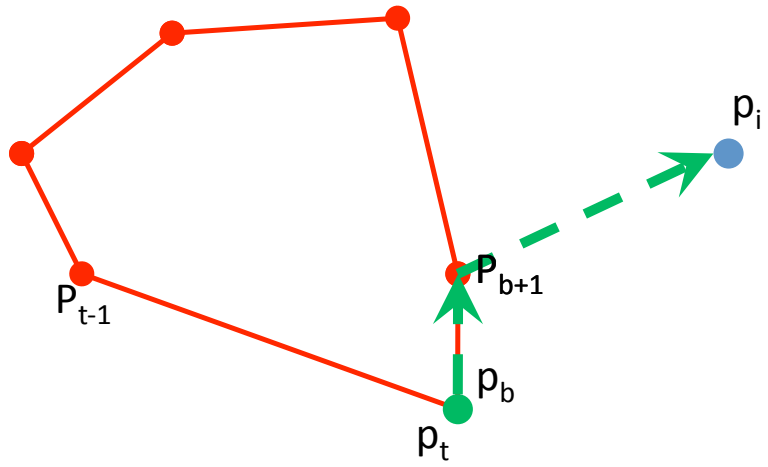
- **Algorithm:**
  - D ← <p_2,p_1, p_2> /*convex hull of {p1,p2}
  - **For** i = 3 to n do
    - **While** (p_b, p_{b+1}, p_i) is a right turn **do** PopBottom(D)
    - **While** (p_t, p_{t-1}, p_i) is a left turn **do** PopTop(D)
    - if p_i is is not in region III then

      PushBottom(p_i, D) ; PushTop(p_i, D)
  - Output D

Simple Polygon

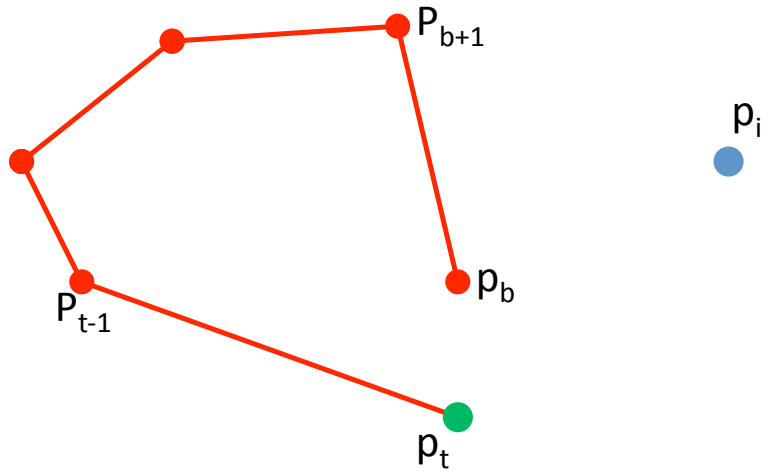


Simple Polygon

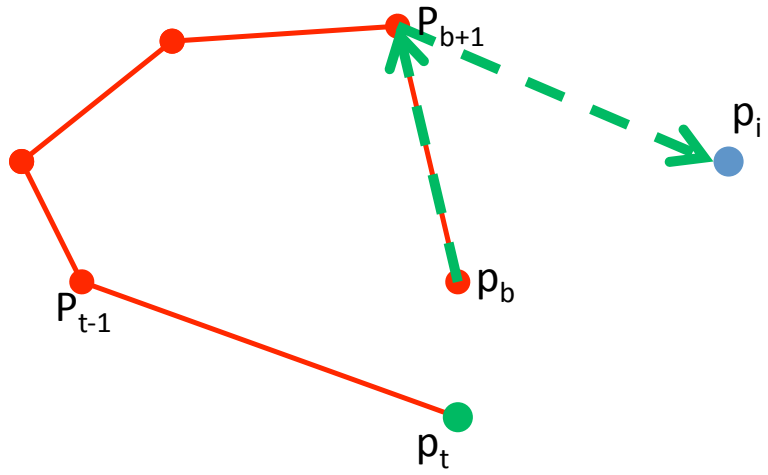


(p_b, p_{b+1}, p_i) is a <u>right</u> Turn
=> PopBottom(D)

Simple Polygon

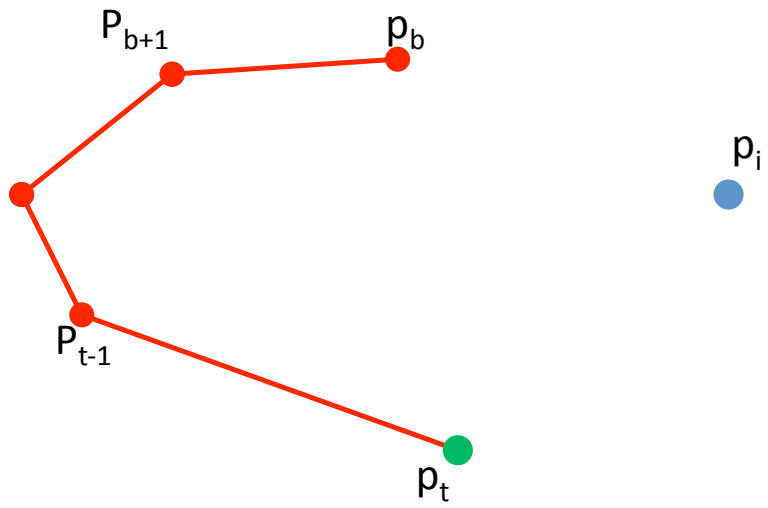


Simple Polygon

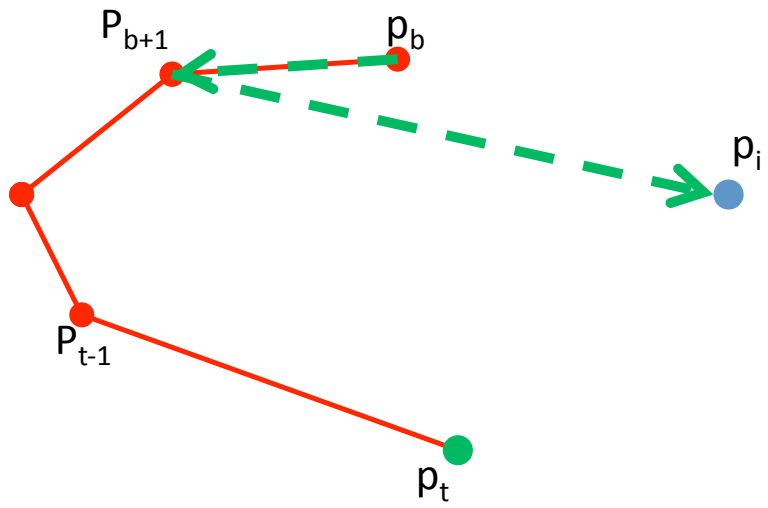


(p_b, p_{b+1}, p_i) is a <u>right</u> Turn
=> PopBottom(D)

Simple Polygon

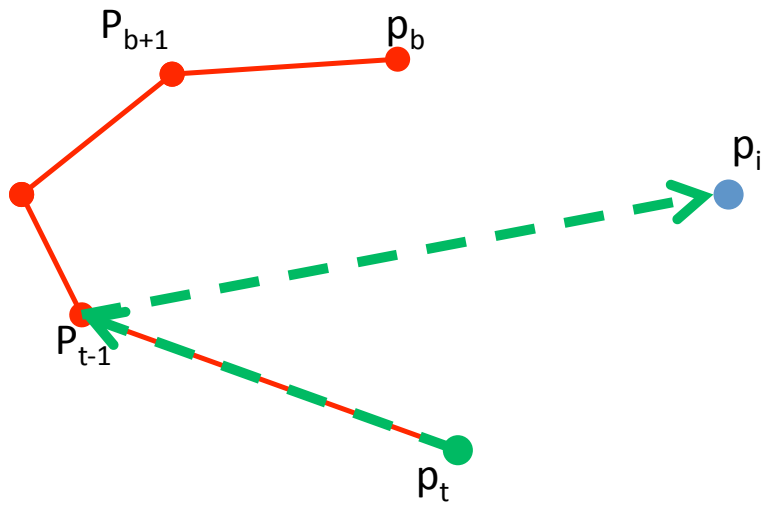


Simple Polygon



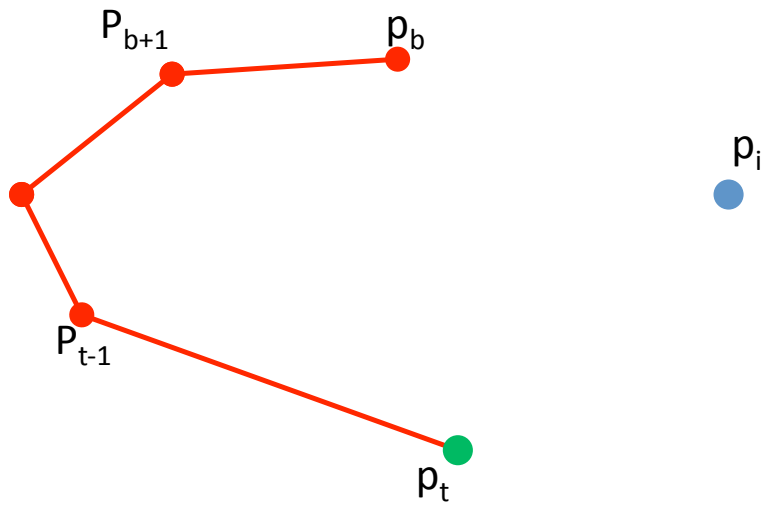
(p_b, p_{b+1}, p_i) is a **Left** Turn

Simple Polygon



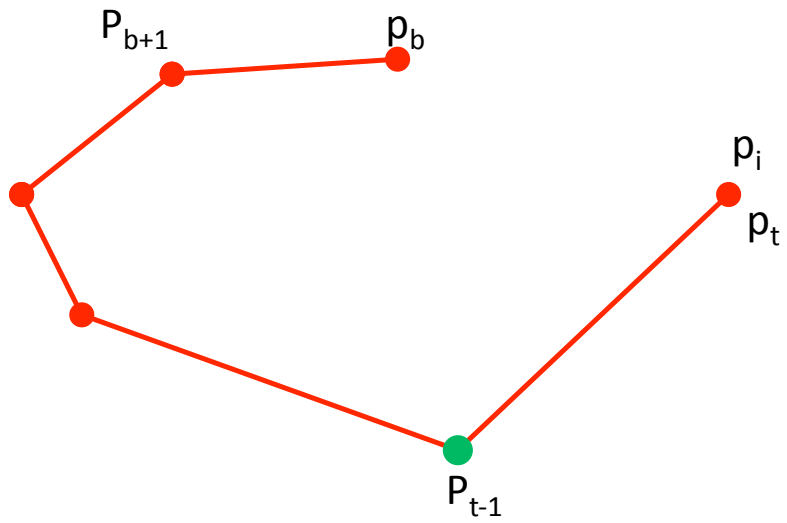
(p_t, p_{t-1}, p_i) is a <u>right</u> Turn

Simple Polygon

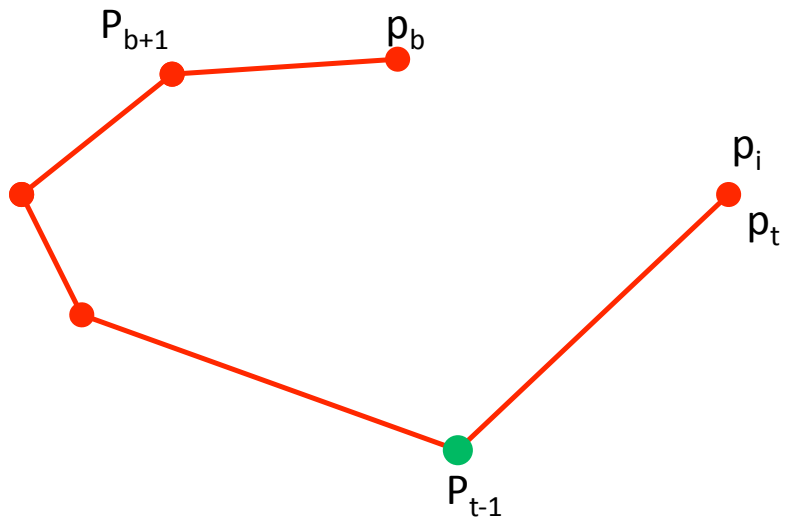


PushTop(p_i, D)

Simple Polygon

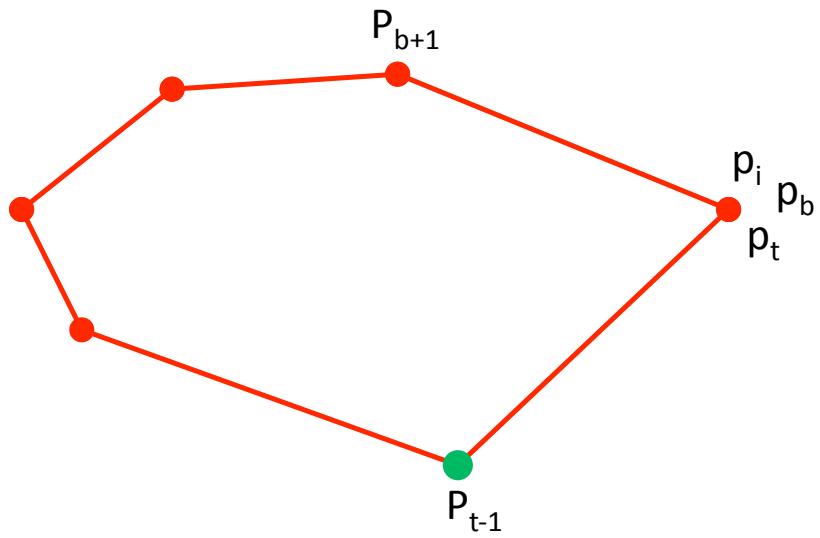


Simple Polygon

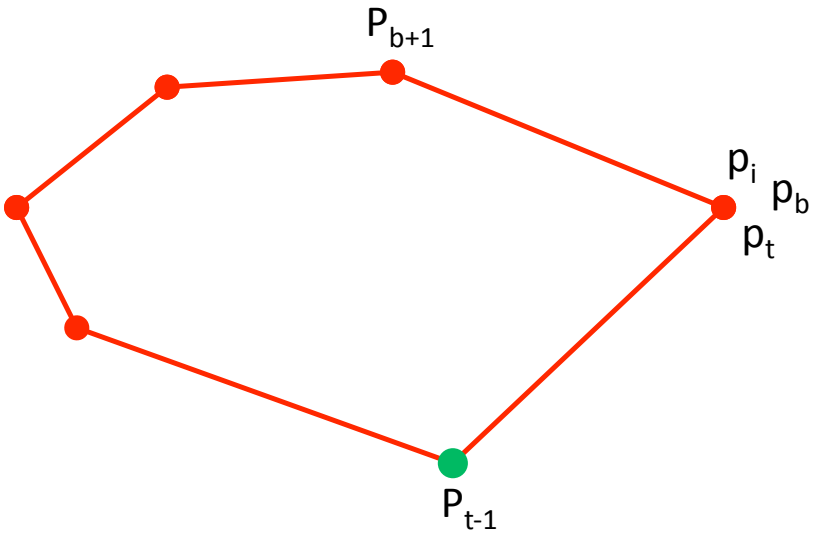


PushBottom (p_i, D)

Simple Polygon

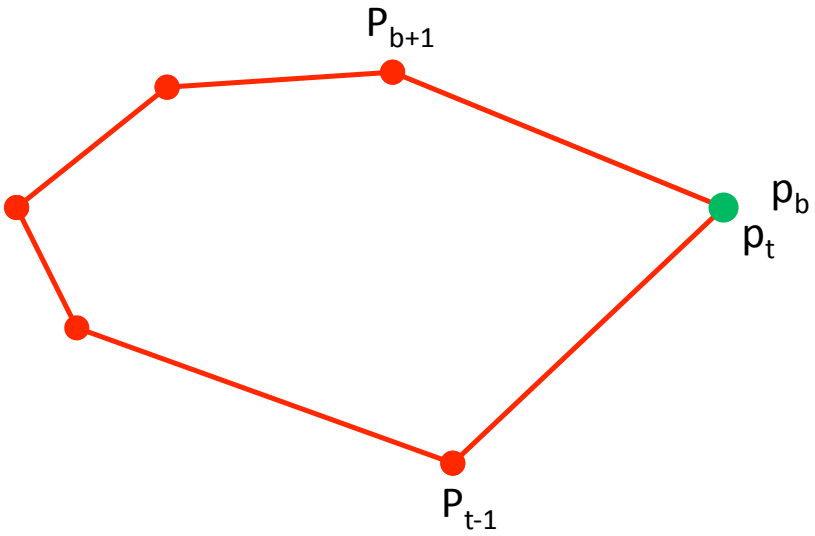


Simple Polygon



Update “You are here” pointer
Output(D)

Simple Polygon



Complexity

- Every point is either discarded or pushed onto D exactly twice.
- Each element of D is deleted exactly twice.
- At most 2n pushes and 2n-3 pops
- Running time : O(n)

Output-sensitive Convex Hull algorithms

- Kirkpatrick-Seidel (1986) described an O(nlogh) worst case algorithm. The size of the convex hull is h.
- Chan(1996) gave two O(nlogh) algorithms. One of them combines two slower algorithms (Jarvis' and Graham's) to get O(nlogh) algorithm. Note that Jarvis' algorithm is good if h is small, and Graham's algorithm is good if h is $O(n^{\alpha})$, $\alpha > 0$.
- "Output-sensitive results on convex-hulls, extreme points, and related problems", Timothy Chan, Discrete Computational Geometry, Vol. 16, pp. 369-387, 1996.

Grouping Algorithm (Key Idea)

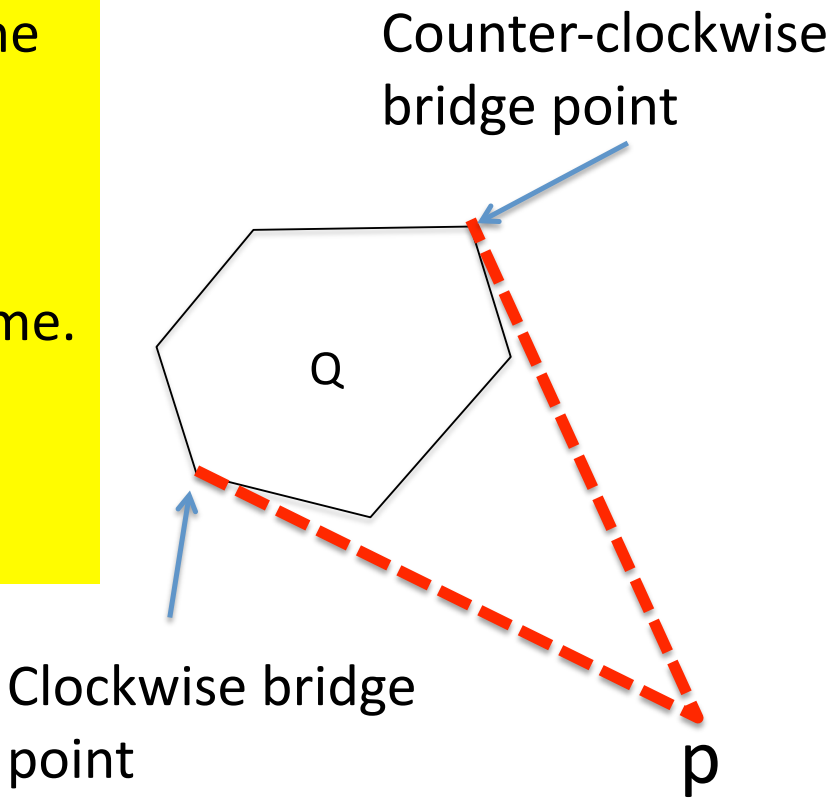
- Partition n points into groups of size m; the number of groups is r=ceiling(n/m).
- Compute the convex hull of each group using Graham's algorithm.
- Next, run Jarvis' algorithm on the groups.

Chan's Algorithm
(Assuming that the number of extreme points h of the given set S of points is known.

- $m \leftarrow h$
- Partition P into r groups A_i of size at most m. {r=ceiling(n/m)}
- Compute $CH(A_i)$ using Graham's algorithm, i=1, 2, ...,r.
- Apply Jarvis' algorithm on groups of convex hulls, $CH(A_i)$, i=1, 2, ...,r.

Computing CH(CH(A_1) U .... U CH(A_r))

We will use the fact that the two bridges from a point p lying outside a convex polygon Q to Q can be computed in logarithmic time. We are assuming that the vertices of the polygon is available in cyclic order.



Computing CH(CH(A_1) U .... U CH(A_r))

- Starting from an extreme point p of S. Without any loss of generality we assume that p is in A_1. Let p^+ be the counter-clockwise neighbor of p in CH(A1).
- Determine the counter-clockwise bridge point q_i of CH(A_i), i= 2, 3, ..., r.
- Determine the counter-clockwise extreme edge (p,q) incident on p of CH(A_1U .... UA_r).
  - q ε {p^+, q_2, q_3, ..., q_r}; q is an extreme point
- Repeat the process with p = q.

Total running time to run Jarvis' algorithm on the groups

- To compute p: O(n)
- To compute q_i: O(logm)
- To determine q: O(r)
- If there are m extreme edges in CH(A_1U .... UA_r), the total cost of the merging step is O(mrlogm + n).

Total Complexity

- Graham's algorithm
  - r.O(mlogm) = O(n/m).O(mlogm) = O(nlogm)
- Jarvis' algorithm on the groups
  - O(mrlogm + n) = O(m.ceiling(n/m).logm) = O(nlogm)

- Total time = O(nlogm) which is O(nlogh) since we assumed m=h.

Problem: We don't know h beforehand!!

- Solution: Trial and error on the size.
- for t=1, 2, 3, ...
  - Let m = min ($2^{2^{\wedge}t}$,n)
  - Run the algorithm with m. Stop Jarvis' algorithm if it has produced more than m extreme edges, otherwise return the convex hull of S.

Analysis

- Iteration t takes time O(nlog $2^{2\wedge t}$), i.e. O(n2^t).
- Maximum value of t is loglogh, sence we succeed as soon as $2^{2\wedge t} > n$.
- Running time :
  - $n.2^1 + n.2^2 + \ldots + n.\ 2^t \leq n.\ 2^{t+1} = O(nlogh)$.

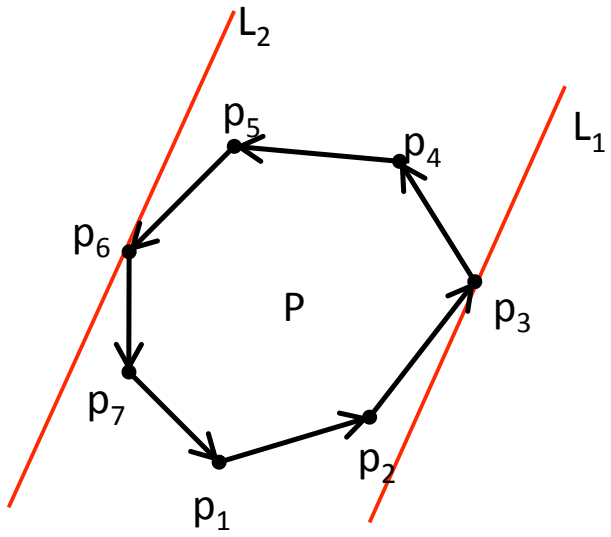
- This algorithm is optimal since Ω(nlogh) is the refined lower bound of the convex hull problem.

Rotating Calipers

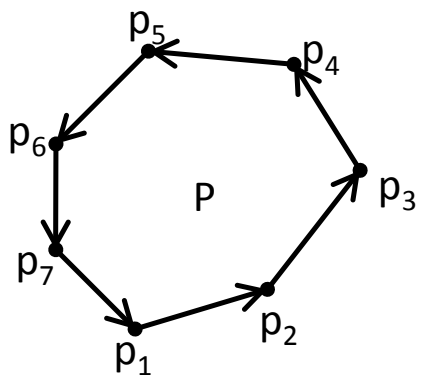
- Godfried Toussaint, ***Solving Geometric Problems with the Rotating Calipers***, Proc. IEEE, 1983
- Let P = (p_1, p_2, …, p_n) be a convex polygon given in CCW order.
- **Definition:** A pair of points p_i & p_j are an **antipodal pair** if they admit parallel lines of support

Antipodal pairs:

(p_3,p_6)
(p_3,p_7)
(p_4,p_7)
(p_1,p_4)
(p_5,p_1)
(p_5,p_2)
(p_3,p_5)



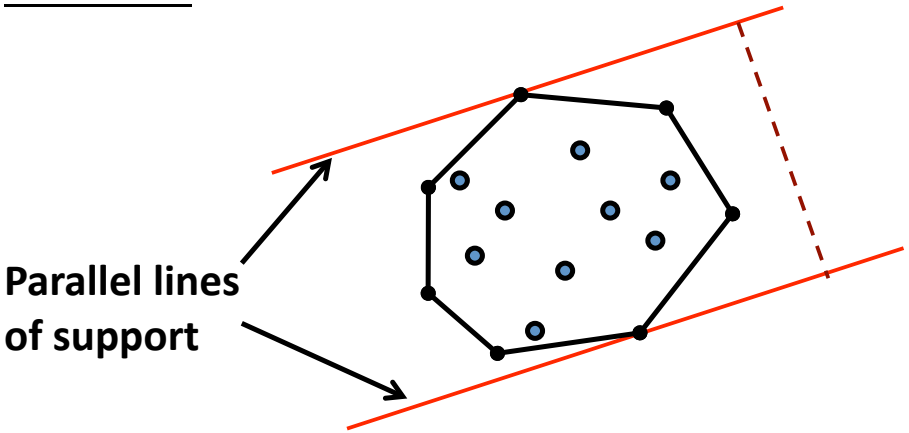
Rotating Calipers



- **Theorem:** There are O(n) antipodal pairs in the worst case.
- **Sketch:** Every edge can realize at most two antipodal pairs.

Convex Hulls (applications)

- **A Problem in Statistics:**
  - Given a set of n data points in R^2, fit a line that minimizes the maximum error (A data point's error is its L_2 norm (eclidean norm) distance to the line)
    - Minimizing max error = parallel lines of support with **minimum separation**
    - Minimum separation between parallel support lines is also called the **width** of S.
    - Maximum separation between parallel support lines is called the **diameter** of S.



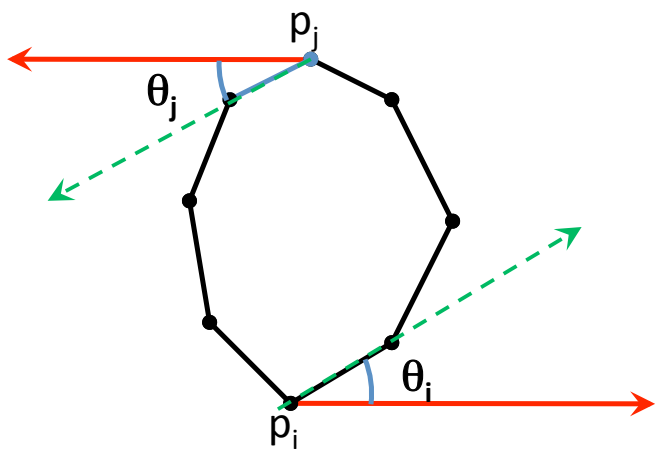
Convex Hulls (applications)

- **Definition:** a and b are antipodal pairs of S if they admit parallel lines of support of S.
- Can be shown that:
  - In the plane, S has at most **O(n)** antipodal pairs
  - If L_1 & L_2 are parallel lines of support with minimum separation, at least one of the lines contains an edge of CH(S)
  - If L_1 & L_2 are parallel lines of support with maximum separation, L_1 & L_2 pass through a & b respectively with ab orthogonal to L_1 & L_2 and no other points of S – {a,b} lies on L_1 or L_2.

Enumerating all the Antipodal Pairs

- Antipodal pairs of a set S must be extreme points and therefore must be the vertices of CH(S)
- **<u>Rotating Calipers</u>**: (Proposed by Shamos)

- Find the convex hull
- Find an antipodal pair (p_i,p_j)
- Generate the next antipodal pair:
  - Determine θ_i & θ_j
  - (suppose $\theta_i < \theta_j$) rotate the lines of support by θ_i
  - Output (p_{i+1}, p_j) as the next antipodal pair
- Repeat step 2 until L or R is rotated by 180^o

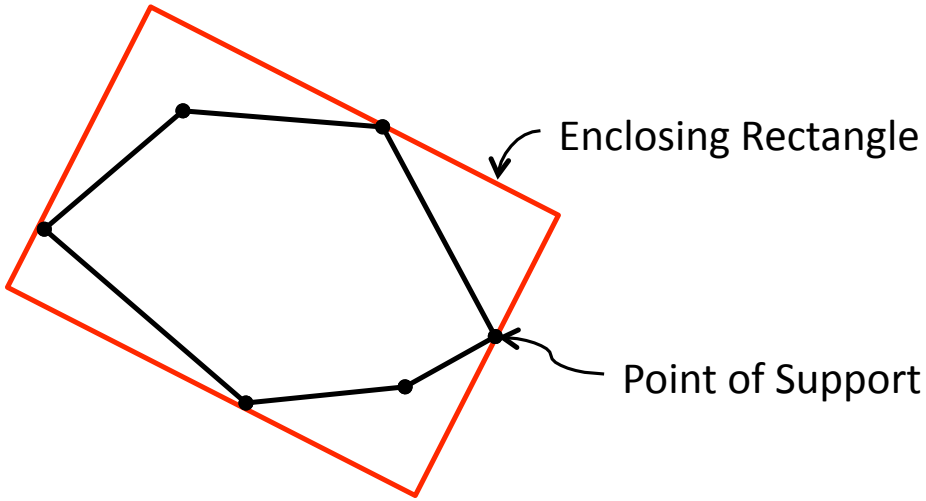


Analysis of Rotating Calipers

- **Theorem:** The Rotating Calipers method determines the antipodal pairs of a convex polygon in O(n) time.

Smallest-Area Enclosing Rectangle

- **Problem:** Given a convex polygon P determine the smallest-area rectangle that encloses P.

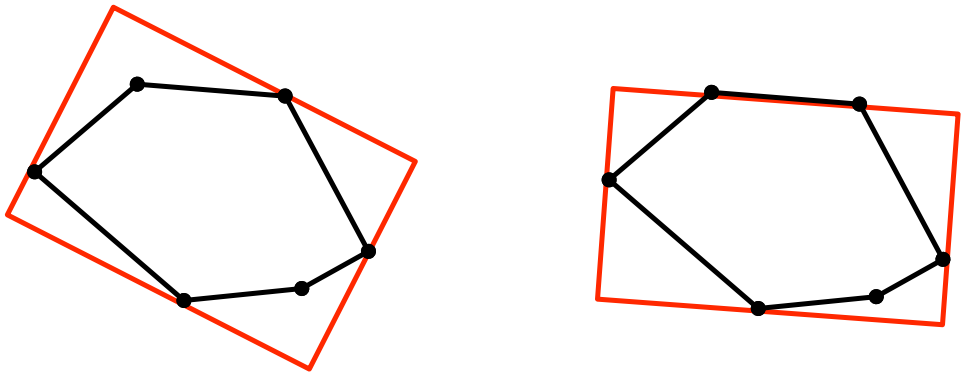


- Each side of the smallest-area enclosing rectangle must support P

Smallest-Area Enclosing Rectangle

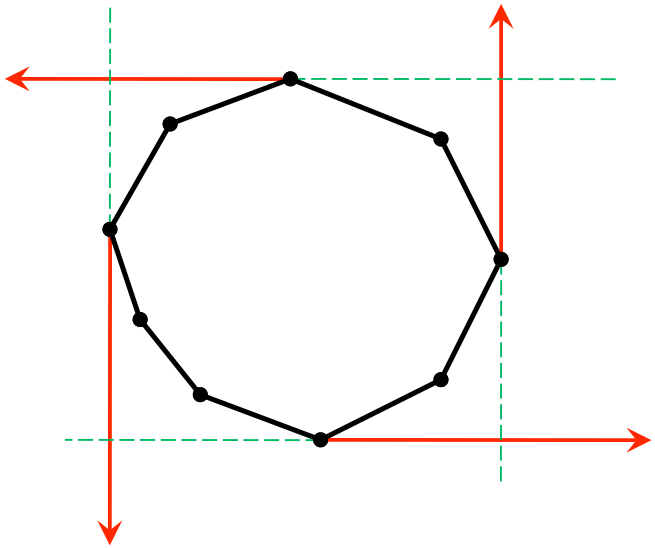
- **Crucial Lemma:**
  - The rectangle of minimum area enclosing a convex polygon has a side collinear with one of the edges of the polygon

- $O(n^2)$ algorithm is easy:
  - Determine an enclosing rectangle for each edge in O(n) time

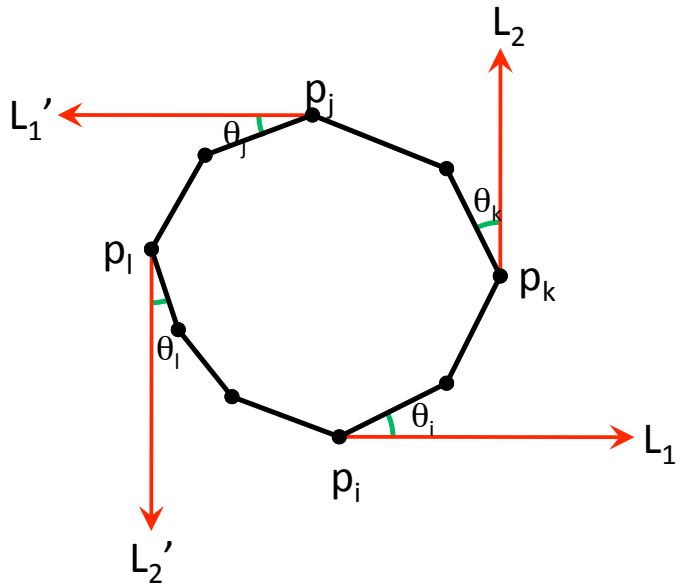


Smallest-Area Enclosing Rectangle

- The problem can be solved in O(n) time using two pairs of calipers orthogonal to each other



Smallest-Area Enclosing Rectangle



- p_i, p_j, p_k, & p_l are supporting vertices of the calipers
- (L_1, L_1') , (L_2, L_2') , (p_i, p_j), and (p_k, p_l) can all be found in O(n) time
- There are now four angles (instead of two) to consider
  - Let $\theta = \min \{\theta_i, \theta_j, \theta_k, \theta_l\}$
  - Rotate all four calipers by θ
  - Repeat with the new angles

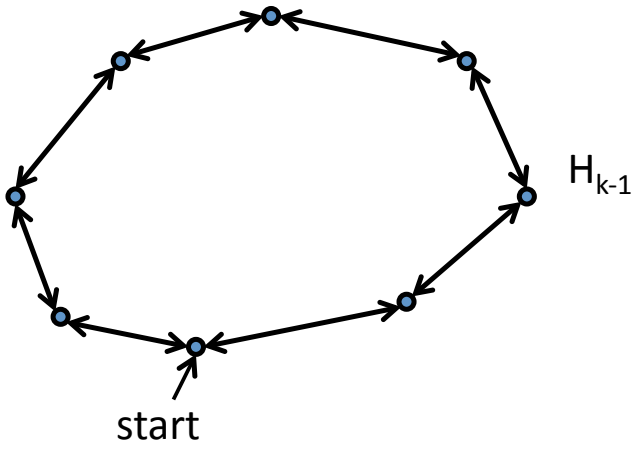
Incremental Algorithm

- Allows easy extension to three or higher dimensions
  - Graham's algorithm does not extend to higher dimension
- Let P = { p_0, p_1, …, p_{n-1} } be the point set in the plane
  - Without any loss of generality we assume that no three points are collinear
- Let H_k = convHull { p_0, p_1, …, p_k}
- Let H_2 = convHull { p_0, p_1, p_2}
- For k = 3 to n-1 do

  $H_k \leftarrow$ convHull { $H_{k-1} \cup p_k$ }

Computing convHull { $H_{k-1} \cup p_k$ }

- **Data Structure:**
  - The convex hull H_{k-1} is available in a circular doubly linked list
  - The vertices are traversed in CCW order

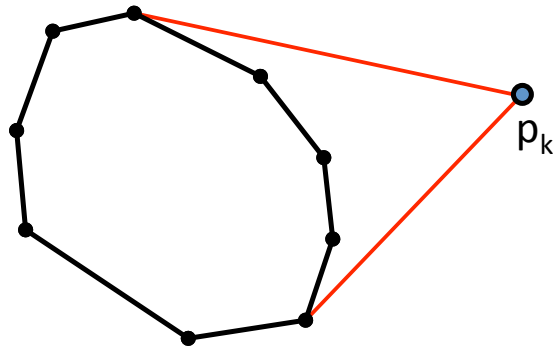


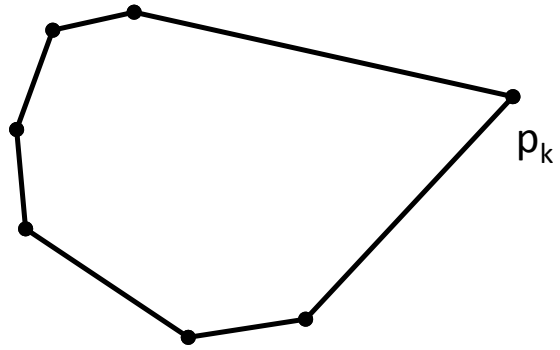
Computing convHull { $H_{k-1} \cup p_k$ }

- **Two Possibilities:**
  - **$P_K \in H_{k-1}$**
    - If P_k is determined to be inside H_{k-1}, **$H_k = H_{k-1}$**
    - If $P_K \in H_{k-1}$, p_k is to the left of every directed edge of H_{k-1} (possibly on one edge) . Clearly this takes time linear in the number of vertices of H_{k-1}
  - **$P_k \notin H_{k-1}$**
    - If p_k lies to the right of any directed edge of H_{k-1}, p_k lies outside of H_{k-1}
    - Two lines of tangency from p_k to H_{k-1} can be found and can be used to modify the hull accordingly.

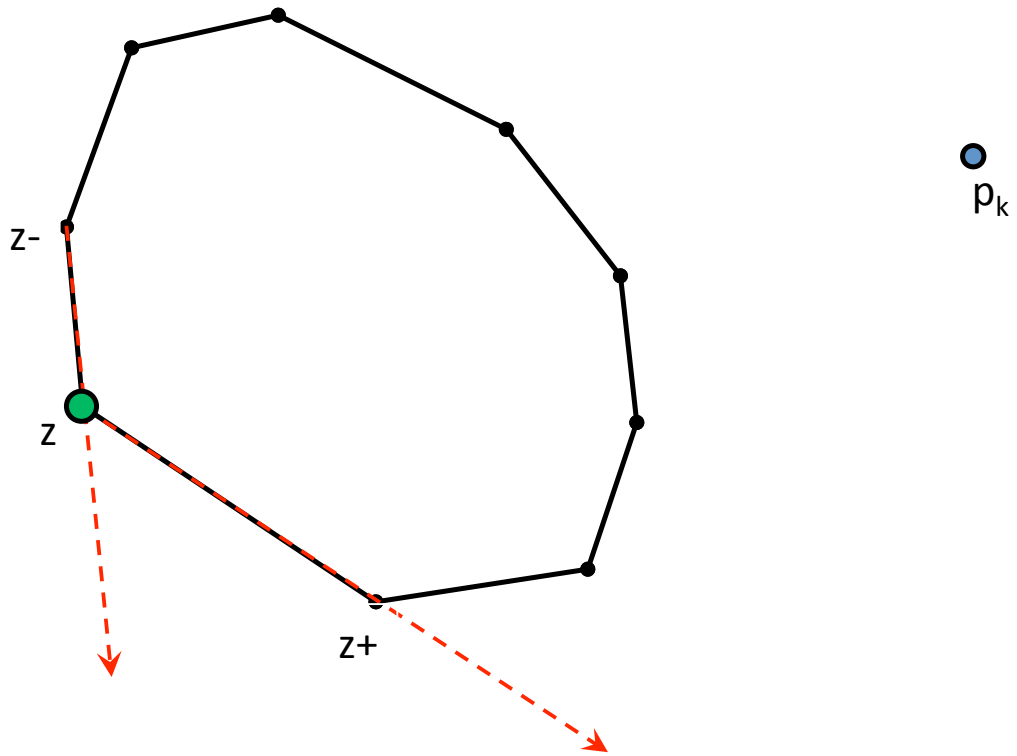
Computing convHull { $H_{k-1} \cup p_k$ }

- **<u>Algorithm for Tangent Points</u>**
  - For each vertex z of H_{k-1}
    - If XOR(p_k is left of z^-z, p_k is left of zz^+)
      - Z is a point of tangency
- Once the tangent points are found, the convex hull H_{k-1} can be updated in time proportional to the size of the parts deleted from H_{k-1} to obtain H_k





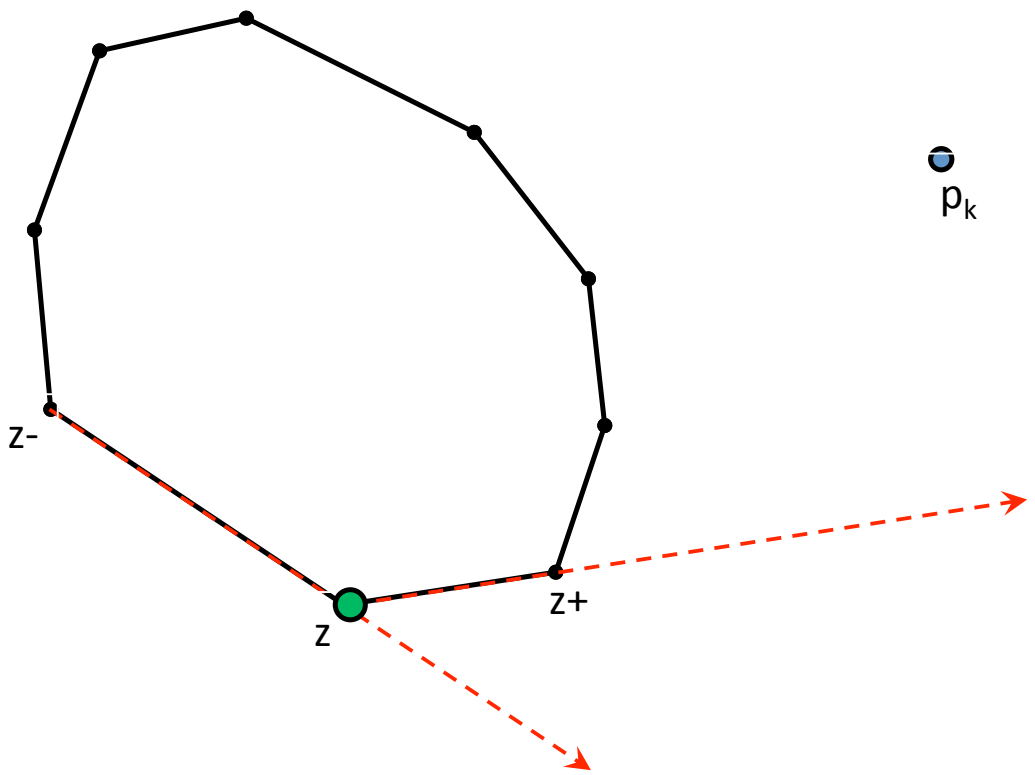
Computing convHull { $H_{k-1} \cup p_k$ }



XOR(p_k is left of z^-z, p_k is left of zz^+) = **false**

=> z is **not** a tangent point

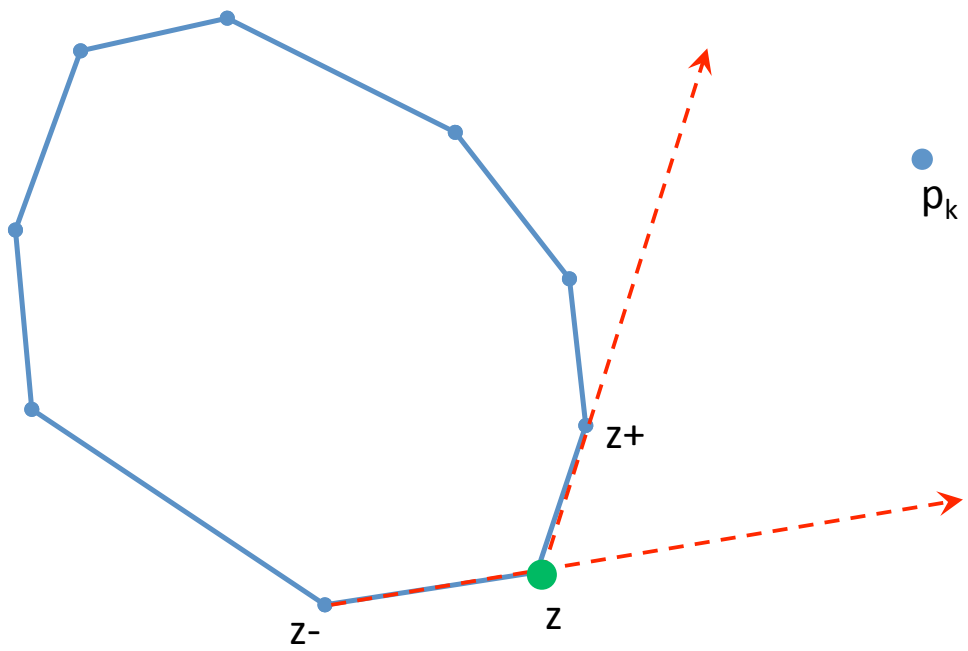
Computing convHull { $H_{k-1} \cup p_k$ }



XOR(p_k is left of z^-z, p_k is left of zz^+) = **false**

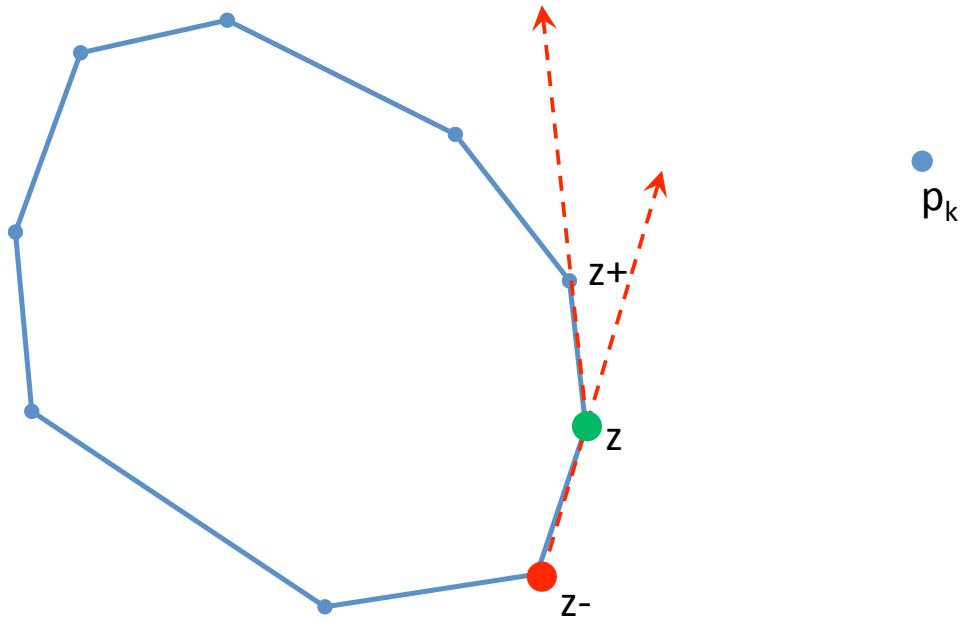
=> z is **not** a tangent point

Computing convHull { $H_{k-1} \cup p_k$ }



XOR(p_k is left of z^-z, p_k is left of zz^+) = **True**
=> z **is** a tangent point

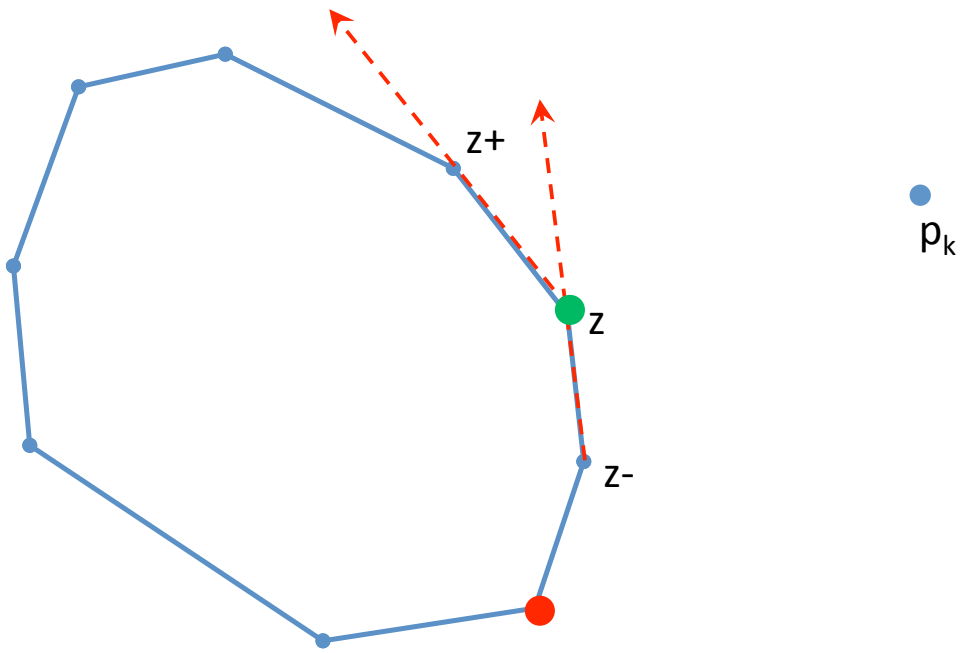
Computing convHull { $H_{k-1} \cup p_k$ }



XOR(p_k is left of z^-z, p_k is left of zz^+) = **false**

=> z is **not** a tangent point

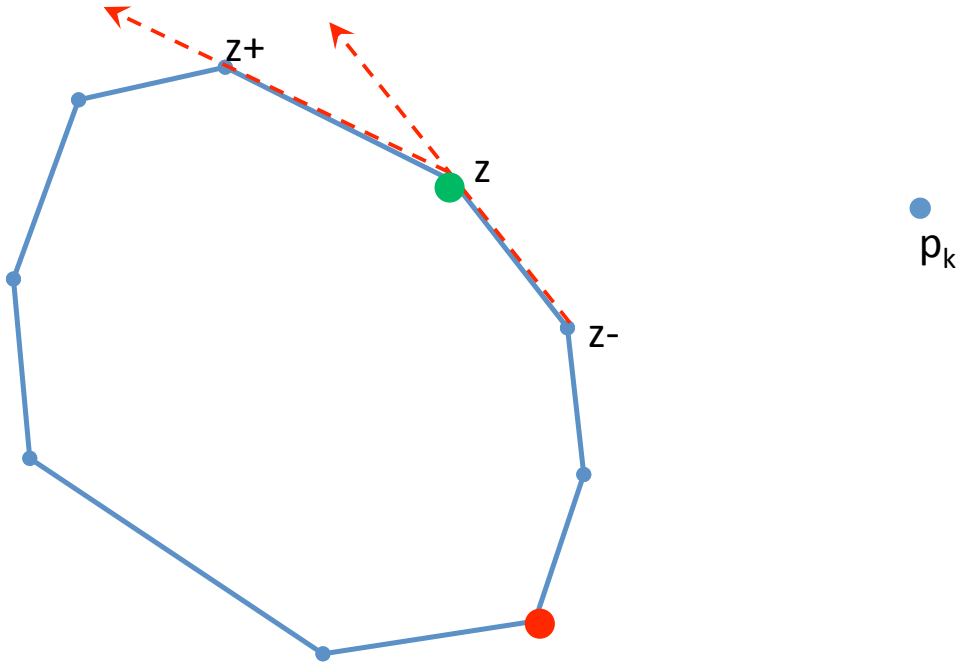
Computing convHull { $H_{k-1} \cup p_k$ }



XOR(p_k is left of z^-z, p_k is left of zz^+) = **false**

=> z is **not** a tangent point

Computing convHull { $H_{k-1} \cup p_k$ }



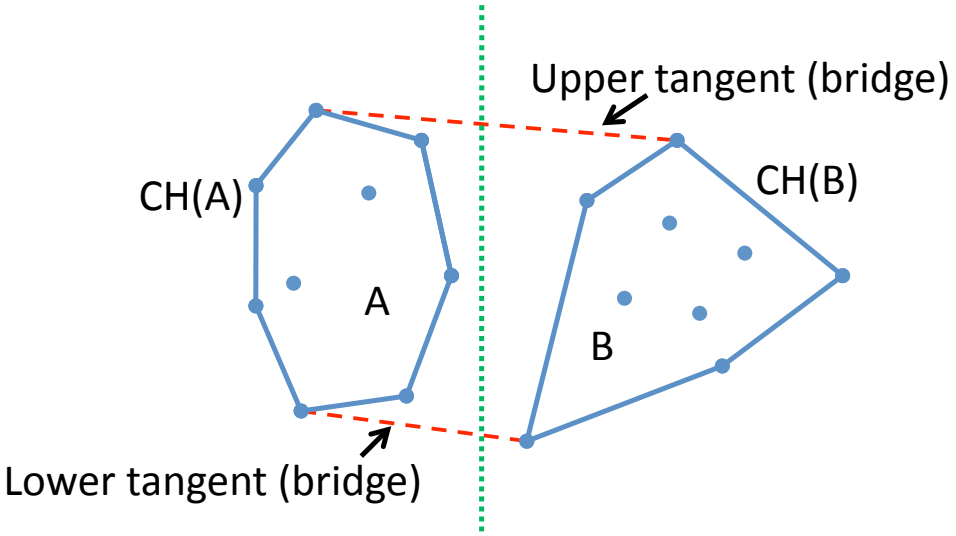
XOR(p_k is left of z^-z, p_k is left of zz^+) = **false**

=> z is **not** a tangent point

Divide and Conquer Algorithm

- Sort the points by x-coordinates
- Let A be the set of n/2 leftmost points and B the set of n/2 rightmost points
- Recursively compute CH(A) and CH(B)
- Merge CH(A) and CH(B) to obtain CH(S)

Divide and Conquer Algorithm



- The rotating calipers technique can be generalized to determine the upper and lower tangents of CH(A) and CH(B)
- The time required is O(|CH(A)| + |CH(B)|)

Analysis of Divide and Conquer

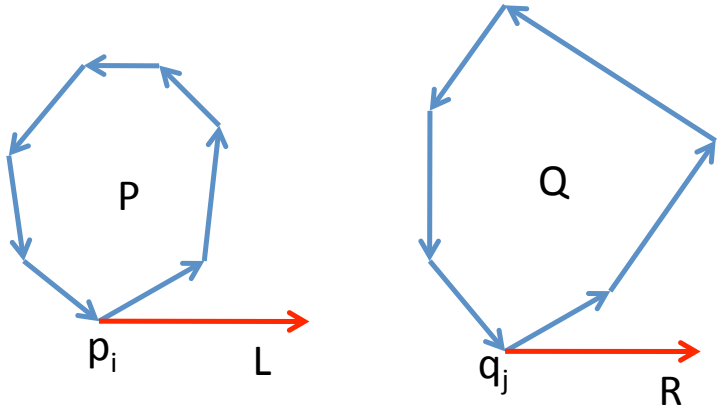
- Initial sorting takes O(nlogn) time
- Recurrence:

$$T(n) = \begin{cases} 2\,T\left(\frac{n}{2}\right) + O(n) & n > 1 \\ 1 & n = 1 \end{cases}$$

- One can show that T(n) is O(nlogn)

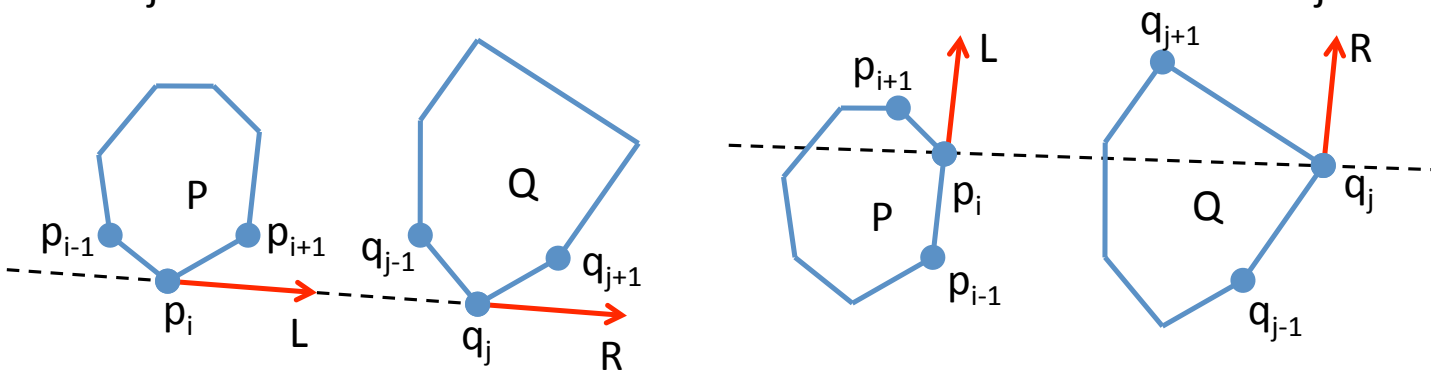
Computing upper & Lower Tangents

- Let P & Q be two convex polygons
- Two vertices $p_i \in P$ and $q_j \in Q$ that admit parallel lines of support in the same direction will be referred to as a **co-podal pair**
- **Lemma:** All the co-podal pairs of P and Q can be found in $O(|P| + |Q|)$



Computing upper & Lower Tangents

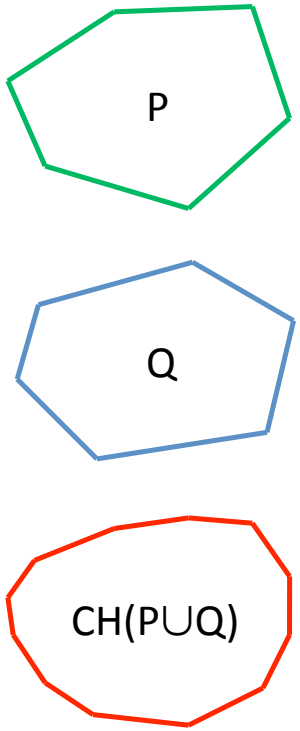
- **Theorem:** Two vertices $p_i \in P$ and $q_j \in Q$ are bridge points if and only if they form a co-podal pair in the direction along p_i and q_j and the vertices p_{i-1}, p_{i+1}, q_{j-1}, and q_{j+1} all lie on the same side of the line (p_i, q_j)

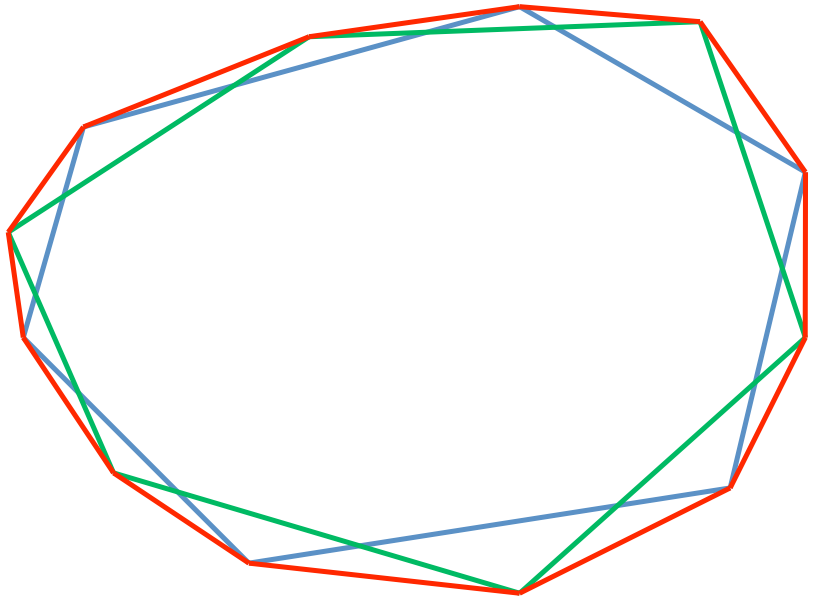


- **Corollary:** The above theorem is still valid when P and Q are not disjoint. (i.e. when they overlap)

Computing upper & Lower Tangents

- If P and Q are not disjoint, then CH(P∪Q) may contain O(n) bridges.





Known Distributions

- **<u>Uniform Distribution in a Unit Square:</u>**
  - Expected size of CH(S) is $O(logn)$

- **<u>Normal Distribution N(0,1):</u>**
  - Expected size of CH(S) is $O(\sqrt{logn})$

Divide and Conquer (slightly different)

- Let A be the first half of the array and B the second half of the array
- Recursively compute CH(A) and CH(B)
- Merge CH(A) and CH(B) to obtain CH(S)

- **<u>Recurrence Relation:</u>**

$$T(n) = \begin{cases} 2\,T\left(\frac{n}{2}\right) + O(|CH(A)| + |CH(B)|) & n > 1 \\ 1 & n = 1 \end{cases}$$

- If $|CH(A)| + |CH(B)| \in O(n^{\alpha})$, $\alpha < 1$, then **$T(n) \in O(n)$**
- In the worst case, **$T(n) \in O(n \log n)$**