COSC 262 Algorithms

1.1 Computational Geometry

Convex Hulls

R. Mukundan
Department of Computer Science and Software Engineering
University of Canterbury

Semester 1, 2013

Introduction

- Computational Geometry (or Geometric Algorithms) : Design and analysis of efficient algorithms for problems involving geometric inputs.
- For most problems, the input is a finite set of geometrical objects in 2D or 3D.
- The primary emphasis is on computational complexity.
- Several applications in Computer Graphics, Robotics, Geographical Information Systems CAD/CAM, Visualization and Computer Vision.

Introduction

Common characteristics of computational geometry problems:

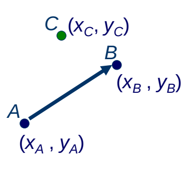
- Error thresholds: Required when two geometrical elements are compared.
- Special cases: Coincident points, Several lines intersecting at a point, Vertical lines, Parallel and overlapping lines
- Degenerate cases: A line segment with coinciding end points, A triangle with zero area.

Directed Line Segment

We can associate the following function with a 2D line segment joining two points A and B:

$L_{AB}(x, y) = (x_B - x_A)\,(y - y_A) - (y_B - y_A)\,(x - x_A)$

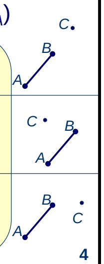
$L_{AB}(C) = (x_B - x_A)\,(y_C - y_A) - (y_B - y_A)\,(x_C - x_A)$



$L_{AB}(C) = 0$: The point C is on the line AB.

$L_{AB}(C) > 0$: The point C is on the left of the directed line AB. The path A B C is a left turn.

$L_{AB}(C) < 0$: The point C is on the right of the directed line AB. The path A B C is a right turn.

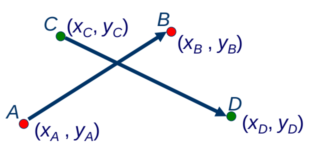


Line Segment Intersection

Two line segments as shown below intersect if and only if

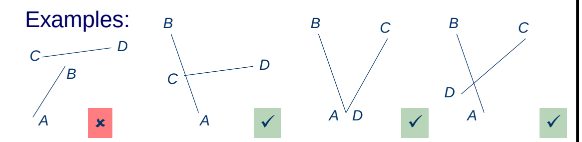
$$L_{AB}(C)\,L_{AB}(D) \le 0$$

AND $$L_{CD}(A)\,L_{CD}(B) \le 0$$



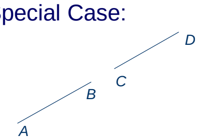
Line Segment Intersection

Examples:



$L_{AB}(C)\,L_{AB}(D) \le 0$	$L_{AB}(C)\,L_{AB}(D) = 0$	$L_{AB}(C)\,L_{AB}(D) = 0$	$L_{AB}(C)\,L_{AB}(D) < 0$
$L_{CD}(A)\,L_{CD}(B) > 0$	$L_{CD}(A)\,L_{CD}(B) \le 0$	$L_{CD}(A)\,L_{CD}(B) = 0$	$L_{CD}(A)\,L_{CD}(B) < 0$

Special Case:



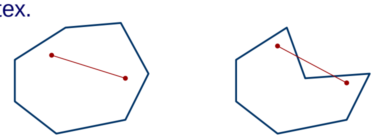
$L_{AB}(C)\,L_{AB}(D) = 0$
$L_{CD}(A)\,L_{CD}(B) = 0$

The test produces a wrong output!

Convex Polygons

A convex polygon satisfies the following properties:

- Any line segment connecting two interior points lies entirely within the polygon
- Every interior angle is less than 180 degs.
- Every anticlockwise traversal of a convex polygon either continues straight, or turns left at every vertex.

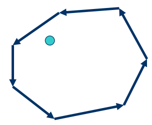


Convex Not Convex

Point Inclusion Test

"Point Inclusion Test" refers to the problem of determining whether a given point is inside a polygon.

A point lies inside a convex polygon if and only if it is on the left side of every edge for an anticlockwise traversal of the polygon.

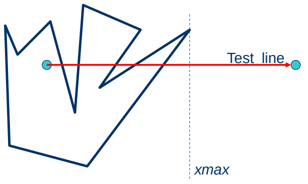


Complexity = $O(n)$

Point Inclusion Test

For a general polygon,

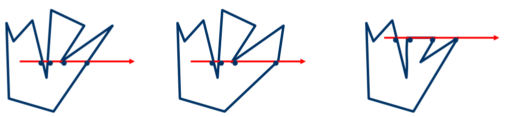
A point lies inside the polygon if and only if a semi-infinite horizontal line emanating from the point intersects the edges an odd number of times.



A horizontal "semi-infinite" line can be generated by defining a point *P* with *x*-coordinate greater than *xmax*.

Point Inclusion Test

Special Cases:

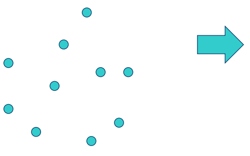


Method:

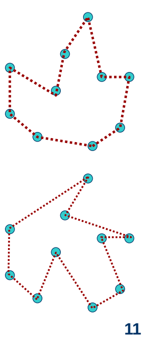
Initialize a counter to 0.
Travel around the polygon, visiting vertices in a sequence.
Ignore vertices that fall on the test line.
If two consecutive vertices are on either side of the test line, increment the counter.
If the counter value is odd, the point is inside the polygon.

Simple Closed Path

Problem statement: Given a set of *n* points on a plane, compute a simple closed path (or a closed polygonal line) that passes through all the points and does not intersect itself.



Input



Simple Closed Path

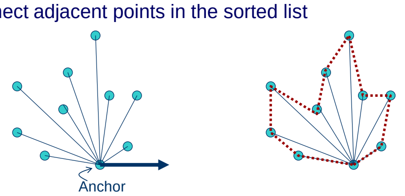
Solution:

Select the point with minimum *y* value as the anchor point.

For each point, compute the angle between the line segment from the anchor to the point and a horizontal line through the anchor point.

Sort the points according to the angle.

Connect adjacent points in the sorted list



Simple Closed Path

- The solution for the simple closed path takes $O(n\log n)$ time.
- It requires the computation of the angle between a line and a horizontal line.

A simple approximation for θ:

Function *Theta*(P_A, P_B):

$dx = x_B - x_A$ $dy = y_B - y_A$

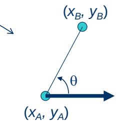
If (abs(dx) < 1.e–6 and abs(dy) < 1.e–6) $t = 0$

else $t = dy$ / (abs(dx) + abs(dy))

If (dx < 0) $t = 2 - t$

else if (dy < 0) $t = 4 + t$

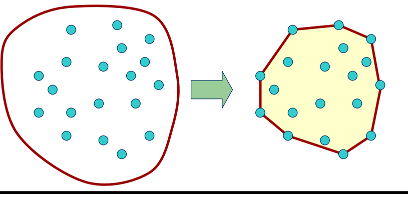
theta = t * 90



$$\theta = \tan^{-1}\left(\frac{y_B - y_A}{x_B - x_A}\right)$$

Convex Hulls

- Given a finite set of points $S = \{P_0, P_1 \ldots P_{n-1}\}$, the convex hull of S is the smallest convex polygon enclosing all of the points.
- Visualised as the shape of a stretched rubber band around the points so that every point lies within the band.



Properties of Convex Hulls

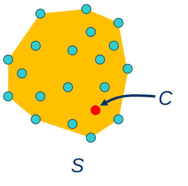
- A convex hull of a set of points S is a *unique* convex polygon that contains every point of S, and whose vertices are only points in S.
- A convex hull is the intersection of all convex polygons containing S.
- A convex hull is the union of all triangles determined by points of S.
- Points in S with minimum x, maximum x, minimum y, and maximum y coordinates are all vertices of the convex hull of S.

Properties of Convex Hulls

If we construct a new point C using a convex combination of points in S, then C lies inside the convex hull of S.

$C = w_0P_0 + w_1P_1 + w_2P_2 + \ldots + w_{n-1}P_{n-1}$

$0 \le w_i \le 1$ for all i. $w_0+w_1+\ldots+w_{n-1} = 1$



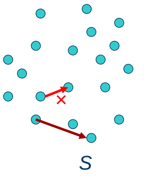
Applications of Convex Hulls

- Robotics
  - Path planning
  - Object recognition
- Graphics
  - Bounding volumes
  - Collision detection
- Geometry Algorithms
  - Voronoi diagrams
  - Delaunay triangulations

Computation of Convex Hull - [1]

- Naïve Method
  - Construct edges using pairs of points, and check if every point in S lies on the left side of the edge.

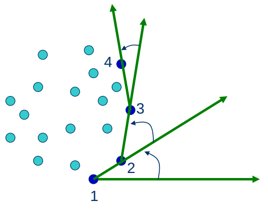
For each point $P \in S$ {
  For each point $Q \in S$, $P \ne Q$ {
    If $L_{PQ}(R) \ge 0$ for every $R \in S$
    then PQ is an edge of the convex hull
  }
}



Complexity: $O(n^3)$

Computation of Convex Hull - [2]

- Gift wrap (a.k.a Package wrap, Jarvis March)
  - Starting from the point with minimum *y*-coordinate, select the next point by identifying the edge that makes minimum angle with the current direction.



Gift Wrap

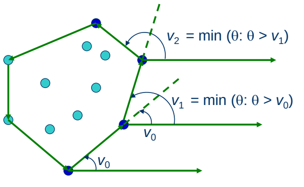
P_0 = point in S with minimum *y*-coordinate
Line = A horizontal line through P_0 towards $+x$.
i = 0;
Repeat {
    For every point $Q \in S$, $Q \neq P_i$ {
        Compute angle between *Line*, P_iQ.
        Choose Q that gives minimum angle
    }
    $P_{i+1} = Q$
    Line = P_iP_{i+1}
    i++;
} until ($P_i = P_0$)

Complexity: $O(mn)$

m = Number of vertices on convex hull

Simplified Gift Wrap

- The previous method requires the computation of angle between two lines. We could express this angle in terms of the angle θ measured from the horizontal line (Slide 13).



Simplified Gift Wrap

- The algorithm need not examine points that have already been included in the hull.
- The program uses an array of points $P_0,\dots P_{n-1}$ as input. P_0 is swapped with the point with minimum y value. The point is also stored in P_n (sentinel).
- In the i^{th} iteration, the points $P_0,\dots P_{i-1}$ are the computed vertices of the hull. In this iteration, only points $P_i..P_{n-1}$ are examined, and the selected point with minimum angle is swapped with the point P_i. The algorithm stops when P_n is selected.

P_0	P_1	P_2	P_3	P_4	P_5					P_{n-1}	P_n

Simplified Gift Wrap

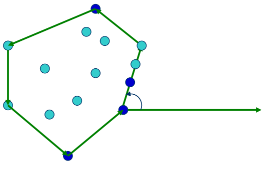
Input: $P_0, P_1, \dots P_{n-1}$.
Find the point P_k with minimum *y*-coordinate
$i = 0$; $v = 0$; $P_n = P_k$
Repeat {
    Swap P_i and P_k
    minAngle = 360
    For ($j = i+1,\dots,n$) {
        angle = *Theta*(P_i, P_j)
        If (angle < minAngle and angle ≥ v) {
            minAngle = angle; $k = j$
        }
    }
    $i = i + 1$; v = minAngle
} until ($k = n$)

Correction!

Gift Wrap

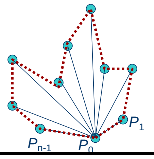
Special Case:

If there are more than one point with minimum angle, select the point with the smallest distance from the current point.

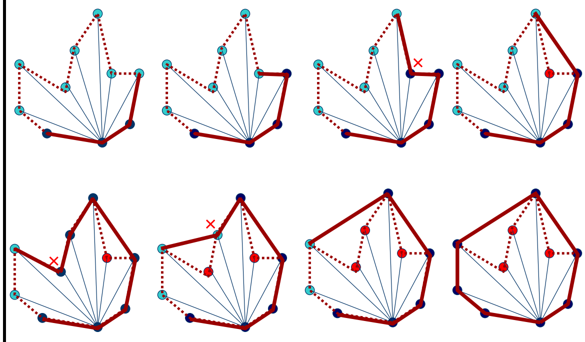


Computation of Convex Hull - [3]

- Graham Scan
  - Use "Simple Closed Path" algorithm (Slide 12) to obtain a sequential ordering of the input points, where P_0 is the point with minimum y coordinate. With this ordering of points, $P_{n\text{-}1}$, P_0, P_1 will always be on the convex hull. Add them to a list S.
  - In each iteration, the selected point is checked if it is on the left of line connecting the last two points in S. If it is, the point is added to S, otherwise, the last point in S is removed.



Graham Scan



● = Points in S ● = Point removed from S

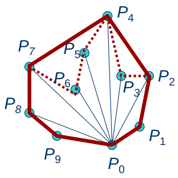
Graham Scan

- Sorting of n points based on angle can take $O(n\log n)$ time.
- Maximum number of comparisons required for finding the correct edge direction is $O(n)$
- The total time complexity of Graham Scan algorithm is $O(n\log n)$.
- The algorithm can be implemented using a stack.

Graham Scan: Pseudo Code

P_0 = Rightmost lowest point
Sort other points by angle about the horizontal line through P_0
Label points in the sorted set as $P_1, P_2, \ldots P_{n\text{-}1.}$
Stack S ; S.Push($P_{n\text{-}1}$); S.Push(P_0); S.Push(P_1); $i = 2$
while $i \le n-1$ do {
    $B = S$.Pop(); $A = S$.Pop();
    if P_i is strictly left of AB {
        S.Push(A); S.Push(B); S.Push(P_i); $i = i + 1$;
    } else {
        S.Push(A)
    }
}

Graham Scan



Trace of the algorithm:
$n = 10$; $S = \{P_9, P_0, P_1\}$
$i = 2$: $B = P_1$, $A = P_0$, S = $\{P_9\}$,
P_2 is on the left of AB
$S = \{P_9, P_0, P_1, P_2\}$; $i = 3$
$i = 3$: $B = P_2$, $A = P_1$, S = $\{P_9, P_0\}$,
P_3 is on the left of AB
$S = \{P_9, P_0, P_1, P_2, P_3\}$; $i = 4$
$i = 4$: $B = P_3$, $A = P_2$, S = $\{P_9, P_0, P_1\}$,
P_4 is NOT on the left of AB
$S = \{P_9, P_0, P_1, P_2\}$
$i = 4$: $B = P_2$, $A = P_1$, S = $\{P_9, P_0\}$,
P_4 is on the left of AB
$S = \{P_9, P_0, P_1, P_2, P_4\}$; $i = 5$ …. etc (Exercise!)

Convex Hull Algorithms

Degenerate case: A very special case of all points in the input set falling on a line is not usually considered by convex hull generation algorithms!