Parma University, Math. Department

Algorithms and Data Structures

Intersection of segments
Convex Hull

Marco Pellegrini

Spring 2002, Lecture 23

Summary of lecture 23

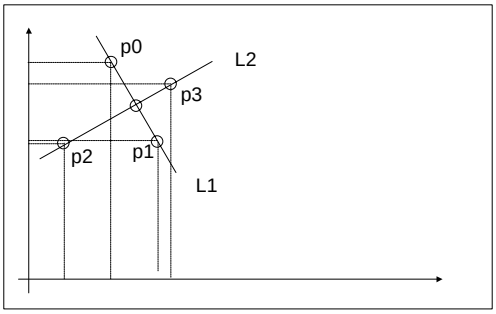
- Segment intersection test.
- Convex Hull,
- Jarvis march
- Graham's scan

Intersection of Segments

(method number 1)

- We are given segments (p0,p1) and (p2,p3) on the plane and we have to determine whether they intersect or not.
- The first method would be:
  (1) determine the line through p0 and p1,
  (2) determine the line through p2 and p3,
  (3) find the intersection point of the lines,
  (4) check that the intersection is in the segments.

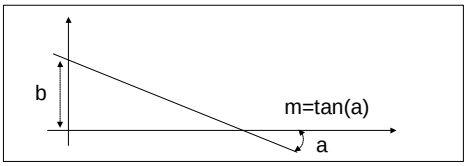
Intersection of segments



Equation of the lines.

(slope/intercept form)

- The slope/intercept equation of the line is $y = mx + b$, for non-vertical lines.
- m is the slope (i.e. the tangent of the angle of the line with the positive x-axis.
  b is the intercept on the y-axis.



Line through 2 points

- The line through 2 points p0=(x0,y0) and p1=(x1,y1) is:
- (y-y1)/(y0-y1) = (x-x1)/(x0-x1).
- If we put such line in slope/intercept form we get:
  m = (y0-y1)/(x0-x1),
  b = (1-m)x1
- So, m is defined only if x0 <> x1. The case x0=x1 must be treated separately.
- Using the above formula we find the equations of lines L1 through (p0,p1) and L2 through (p2,p3).

Finding the intersection point

- To find the intersection point we solve the following system of equations:
  (L1) y = m1x +b1
  (L2) y = m2x +b2
- The solution is:
  yi=(b1m2 -b2m1)/(m2-m1)
  xi=(b1-b2)/(m2-m1)
- Finally we have to check that:
  min(x0,x1) <= xi <= max(x0,x1) AND
  min(x2,x3) <= xi <= max(x2,x3).
- I do not check anything on yi. Why?
- The case m1=m2 needs a special treatment.

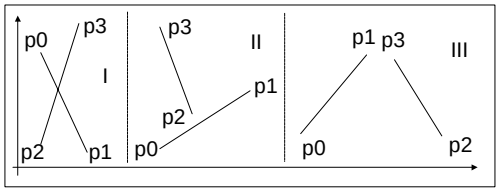
Detecting is easier than Finding.

Method number 2

- We have found the intersection (xi,yi) of the two segments (p0,p1), and (p2,p3).
- We have to handle several special cases and we have used division in several places.
- Division generates truncation errors that can produce the wrong final result.
- But, we do not have to FIND the intersection point in order to DETECT its existence.
- Moreover we can DETECT the existence of an intersection using only left-turn tests.

Proper Intersections.

- First we deal with the case of proper intersections. We suppose that no 3 points among p0,p1,p2,p3 are collinear.
- Exercise: test left turn for: (p0,p1,p2), (p0,p1,p3), (p2,p3,p0), (p2,p3,p1) in 3 cases:



Observations

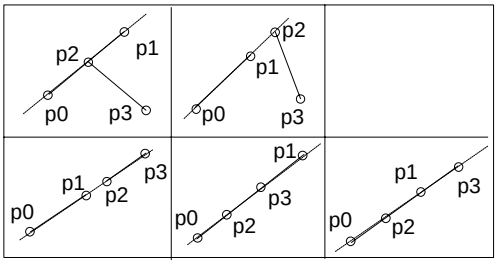
- The 3 points to test have been chosen in the following way: take two end-points of one segment and one of end-point of the other segment. The two points from the same segment are chosen in both cases in the same order.
- In cases II and III at least one pair of turns based on the same segment are equal.
- In case I the two turns based on each segment are different.
- Hypothesis: (p0,p1) meet (p2,p3) if and only if:
  (left-turn(p0,p1,p2) xor left-turn(p0,p1,p3) AND
  (left-turn(p2,p3,p0) xor left-turn(p2,p3,p1).

A formal proof.

- Consider the segments (p0,p1) and (p2,p3) and the lines L1 and L2.
- If (p0,p1) meets (p2,p3) then also L1 meets (p2,p3) and L2 meets (p0,p1).
- But the converse is also true: if L1 meets (p2,p3) and L2 meets (p0,p1), then they meet in the same point, and this point belongs to the two segments.
- The segment (p0,p1) meets L2 if p0 and p1 are on opposite sides of L2, which is the same as to require that (p2,p3,p0) and (p2,p3,p1) make different turns (one left and one right).
- A similar argument holds for L1 and (p2,p3).

Improper intersections.

- Here we have that 3 points among p0,p1,p2,p3 are collinear, so at least one of the tests cannot return a true/false value.



Improper intersections, 2.

We define a procedure Between that given 3 co-linear points (a,b,c) determines wether c is between a and b.

```
BETWEEN(a,b,c: points)
        IF NOT CO-LINEAR(a,b,c)
           THEN RETURN false
        IF x(a) ≠ x(b)
          THEN RETURN ( min(x(a),x(b) ≤ x(c) AND
                  max(x(a),x(b) ≤  x(c))
          ELSE RETURN ( min(y(a),y(b) ≤ y(c) AND
                  max(y(a),y(b) ≤ y(c))
        END IF
```

Segment intersection Test

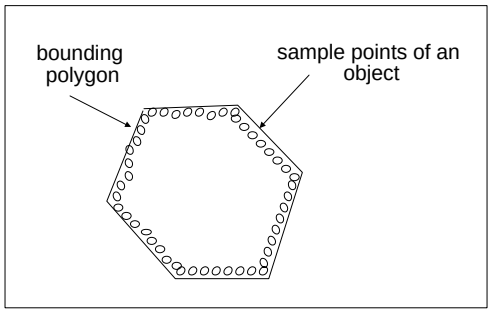
```
Intersect((a,b),(c,d): Segments)
      IF [Collinear(a,b,c) OR
         Collinear(a,b,d) OR
         Collinear(c,d,a) OR
         Collinear(c,d,b)]
        THEN
         RETURN [ Between(a,b,c) OR
                  Between(a,b,d) OR
                  Between(c,d,a) OR
                  Between(c,d,b)]
        ELSE
         RETURN [[ left(a,b,c) xor left(a,b,d)] AND
              [ left(c,d,a) xor left(c,d,b)] ]
      END IF
```

Conclusions 1

- Finding an intersection requires divisions, BUT detecting an intersection requires only left-turn and collinearity tests.
- We have now the whole method to test simplicity of polygons.
- The INTERSECT procedure is robust against truncation errors.
- More uses of these tests are to come.

Making shapes out of points

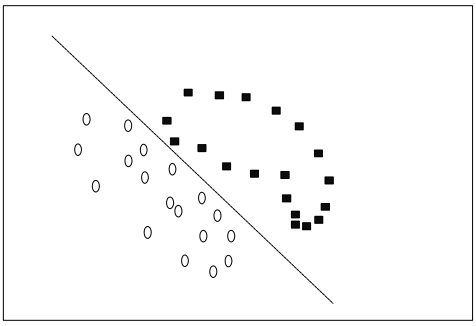
(shape = polygons)



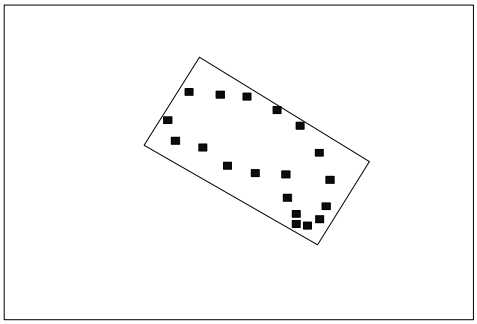
Convex polygon approximating points

- We can compute easily the Area of the polygon and consequently estimate the area spanned by the points.
- We can test efficiently whether an additional point falls "among" the given initial points.
- We can test whether two sets of points overlap or are separable (by a line).
- We can compute easily other bounding polygons (i.e. boxes of arbitrary orientation).

Example: separability of points.



Example: Finding bounding rectangles



Convex Hull: definition and properties

Let S be a set of points and CH(S) the convex hull of S.

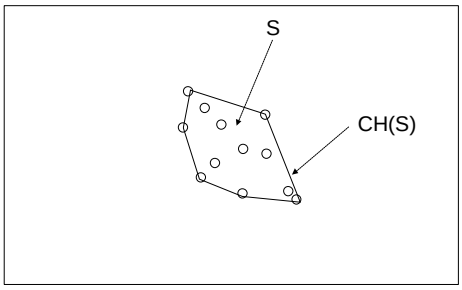
CH(S) is the *convex polygon of smallest area* containing all the points in S.

CH(S) is the *smallest convex polygon* containing S. If P is convex and contains S, then CH(S) is completely contained in P.

CH(S) is the union of all triangles whose vertices are points in S.

CH(S) is the *convex polygon of smallest perimeter* containing all points in S.

Sketch of properties of CH(S)

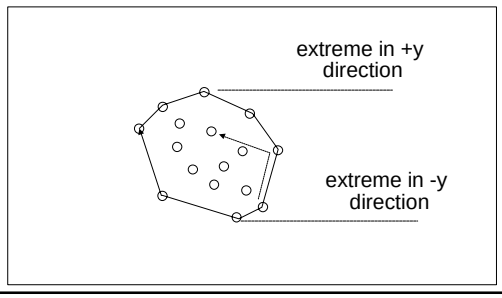


Other properties of CH(S)

- The vertices of CH(S) are points from S.
- If E=(p1,p2) is an edge of CH(S), then all points pj in S are on the same side of the line through p1 and p2. Consequently left(p1,p2,pj) has the same value for all points pj (not collinear with p1 and p2).
- Given S, CH(S) is unique.
- CH(S) is a convex polygon, so in particular all triples of consecutive vertices of CH(S) in counterclockwise order make a left turn.

Other properties of CH(S) 2

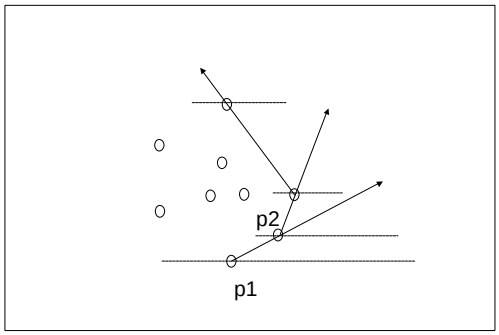
- A vertex of CH(S) is an "extreme" point for some direction.



Jarvis's march

- Jarvis's march is based on the idea of finding one by one all extreme points following the circle of directions.
- The first point we find is the point with smallest y coordinate, which is extremal in the -y direction: p1.
- The second extremal point we find is the point with smallest angle AROUND p1: p2.
- The third extremal point is the point with smallest angle AROUND p2:.. etc.
- Jarvis's march is also called gift-wrapping.

Example of Jarvis's march

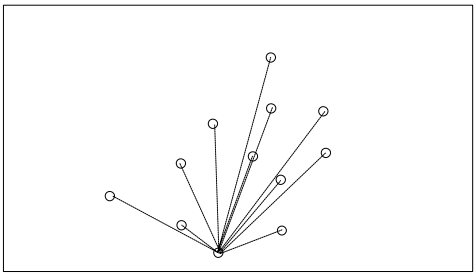


Observations on Jarvis's march

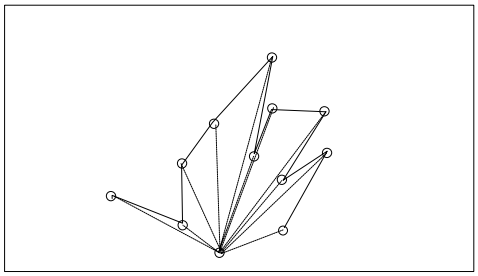
- We can easily find the point p1 with smallest y-coordinate.
- We can also easily fond the point p2 with smallest angle around p1 by using only left-turn tests.
- Finding each extreme point takes time O(n).
- If the CH(S) has h vertices, the total time is O(nh).
- Thus Jarvis's march is fast when the CH has few vertices. In the worst case it is $O(n^2)$.

Graham's scan

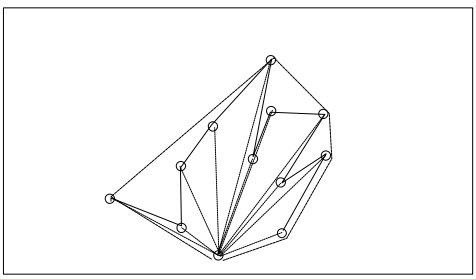
- Sort the points around the lowest point p0.



- Connect the sorted sequence of points to make a simple polygon



- Draw the convex hull.



Observations

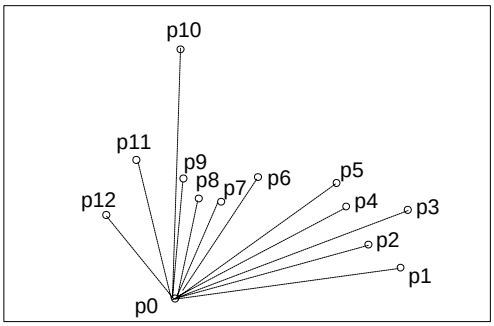
- The vertices of the CH(S) are a sub-sequence of the points in sorted angular order around p0.
- Triple of consecutive vertices of the CH(S) make left turns.
- Graham's scan is based on building CH(S) by adding points in sorted angular order, and deleting those point that violate the left-turn rule.
- In order to keep track of the points we have added to the CH we use a STACK.

Graham's scan

- Find lowest point p0.
  Sort all points by angle around p0,
  getting sequence [p0,p1,.p2,...pn]
  Set up an empty stack S
- GRAHAM-SCAN()
  Push(p0,S), Push(p1,S), Push(p2,S)
  FOR i=3 TO n
  DO WHILE RIGHT(NEXT-TO-TOP(S), TOP(S), pi)
  DO POP(S)
  END WHILE
  PUSH(pi,S)
  END FOR
- The stack S at the end contains the vertices of the CH

Exercise:

Show changes of the stack for these points:



Observations on Graham's scan

- At the end on the stack there are only triple of consecutive points making a left turn.
- Every point popped from the stack cannot be a CH vertex, since it makes a right turn with points chosen in order of increasing angle.
- Since we never pop a vertex that is in the CH, in the stack are left exactly the vertices of the CH.
- Running time of Graham's scan. Each point is pushed at most once and it is popped at most once from the stack. So tha main loop takes
- linear time. The inital sorting take time O(n log n).

Conclusions 2

- One basic operation in many situations is to make a polygon (convex, rectangle,etc) out of a set of points.
- Then, we can do tests on the polygon instead of the original points. Usually the polygon is much smaller than the original set.
- We have seen 2 algorithms: Jarvis's march and Graham's scan. Both use only left-turn/ right-turn tests.
- Graham's scan uses an auxiliary data structure: a Stack.