COS 397 – week 10 GEOMETRIC ALGORITHMS

History

- **Ancient mathematical foundations.**
- **Most geometric algorithms less than 25 years old.**

Geometric Primitives

- **Point: two numbers (x, y).**
- **Line: two numbers a and b [ax + by = 1]**
- **Line segment: two points.**
- **Polygon: sequence of points.**

Primitive operations.

- **Is a point inside a polygon?**
- **Compare slopes of two lines.**
- **Distance between two points.**
- **Do two line segments intersect?**
- **Given three points p1, p2, p3, is p1-p2-p3 a counterclockwise turn?**

Other geometric shapes.

- **Triangle, rectangle, circle, sphere, cone, …**
- **3D and higher dimensions sometimes more complicated.**

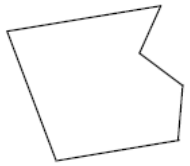
Intuition

Warning: intuition may be misleading.

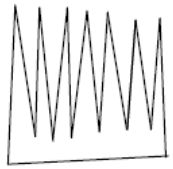
- **Humans have spatial intuition in 2D and 3D.**
- **Computers do not.**
- **Neither has good intuition in higher dimensions!**

Is a given polygon simple?

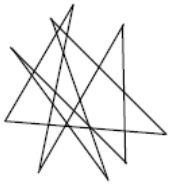
no crossings



1	6	5	8	7	2
7	8	6	4	2	1



1	15	14	13	12	11	10	9	8	7	6	5	4	3	2
1	2	18	4	18	4	19	4	19	4	20	3	20	3	20



1	10	3	7	2	8	8	3	4
6	5	15	1	11	3	14	2	16

we think of this

algorithm sees this

Programming visualization: a simple example

Prepare a text file "`data.txt`" with coordinates and names of 5 points, for example:

```
5
30 90 a
110 10 b
60 80 c
40 30 d
50 150 e
```

Create a Windows Application project. You will automatically get a file to display a simple window. Insert at the beginning of the file (just after `#include` clauses) the code like following:

```
#include<fstream>
using namespace std;

struct point{int x; int y; char c[2];};
point p[99];
int n;

void read_data()
{ifstream f;
 f.open("data.txt");
 f >> n;
 for(int i=0; i<n; i++)
  f >> p[i].x >> p[i].y >> p[i].c;
 f.close();
}
```

```
void show(HWND hwnd)
{ read_data();
  HDC hdcWindow = GetDC(hwnd);
  for(int i=0; i<n; i++)
   {SetPixel(hdcWindow, p[i].x, p[i].y, RGB(0,0,0));
    TextOut(hdcWindow, p[i].x+10, p[i].y, p[i].c, 1);
   }
  MoveToEx(hdcWindow, p[0].x, p[0].y, NULL);
  SelectObject(hdcWindow,
CreatePen(PS_SOLID,1,RGB(255,0,0)));
  for(int i=1; i<n; i++) LineTo(hdcWindow, p[i].x, p[i].y);
  LineTo(hdcWindow, p[0].x, p[0].y);
}
```

Find the place, where the function that creates the windows is called. It looks like:
`ShowWindow(hwnd,…);`
Just after it, write: `show(hwnd);` You may also place `show(hwnd)` in the loop which processes messages for the main window. Now you can see the visualization of the point set together with connecting line segments.

To draw a point (*i*, *j*) you may use:

```
HDC hdcWindow = GetDC(hwnd);
SetPixel(hdcWindow, i, j, RGB(0,0,0));
```

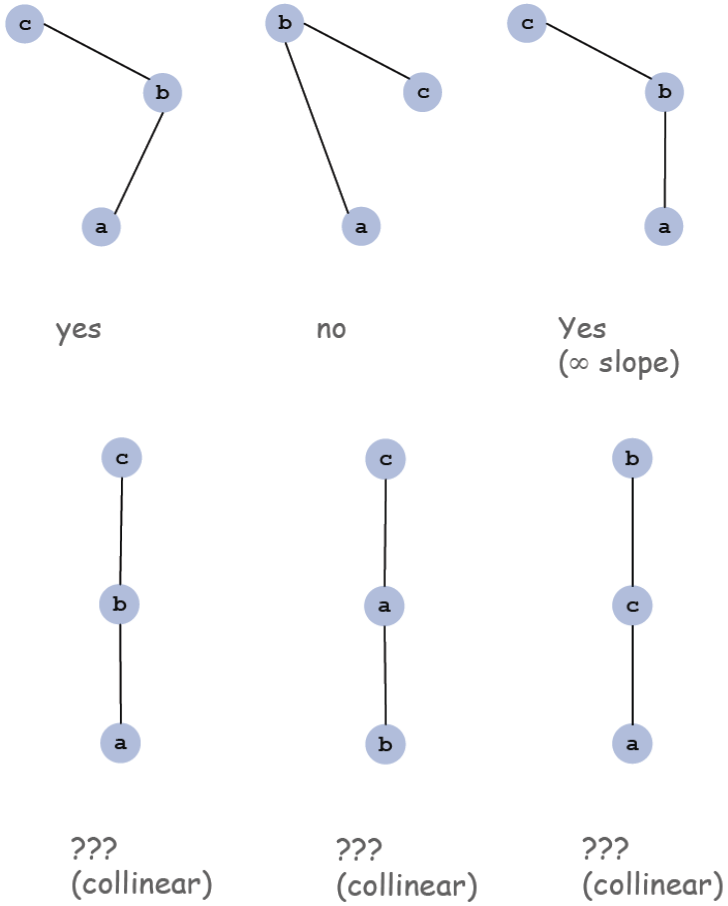
To draw a line segment starting at (0,200) and ending at (100,200) :

```
MoveToEx(hdcWindow,0,200,NULL);
SelectObject(hdcWindow,
CreatePen(PS_SOLID,1,RGB(255,0,0)));
LineTo(hdcWindow,100,200);
```

Microsoft Windows users may check reference sources (e.g. msdn.microsoft.com) for more information about Windows API – win32 API (application programming interfaces available in the Microsoft Windows operating systems) and especially about the Microsoft Foundation Class Library (MFC) that wraps portions of the Windows API in C++ classes.

CCW. Given three point *a*, *b*, and *c*, is *a*-*b*-*c* a counterclockwise turn?

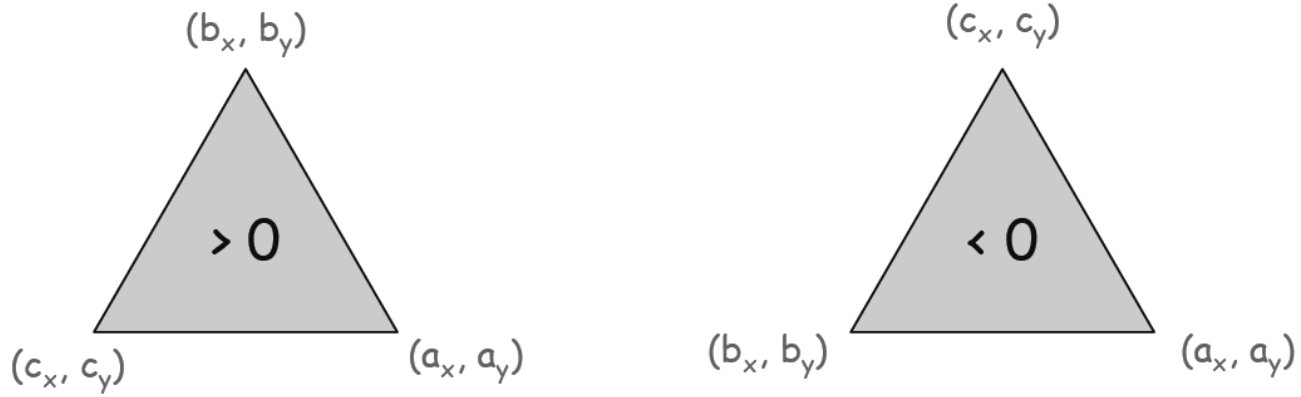
- Analog of comparisons in sorting.
- Idea: compare slopes.



Determinant gives twice area of triangle.

$$2 \times Area(a, b, c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$$

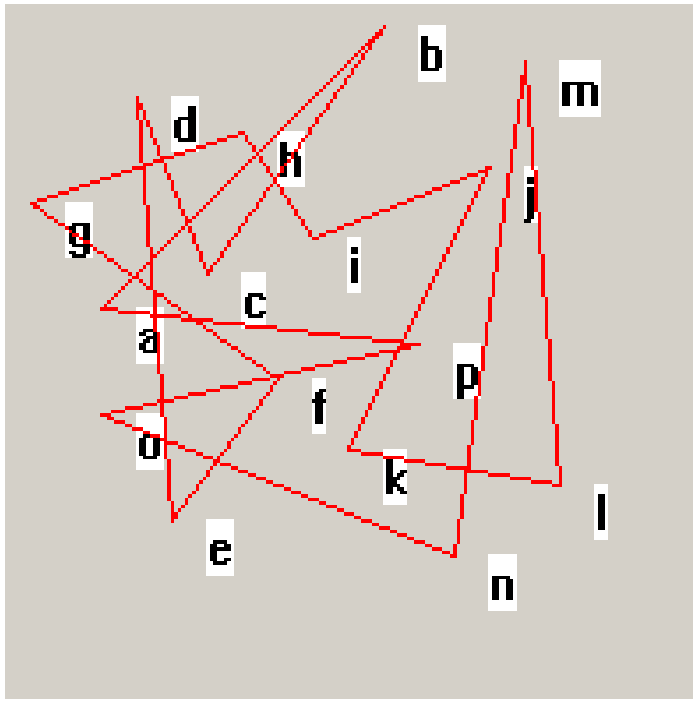
If area > 0 then *a*-*b*-*c* is counterclockwise.
If area < 0, then *a*-*b*-*c* is clockwise.
If area = 0, then *a*-*b*-*c* are collinear.

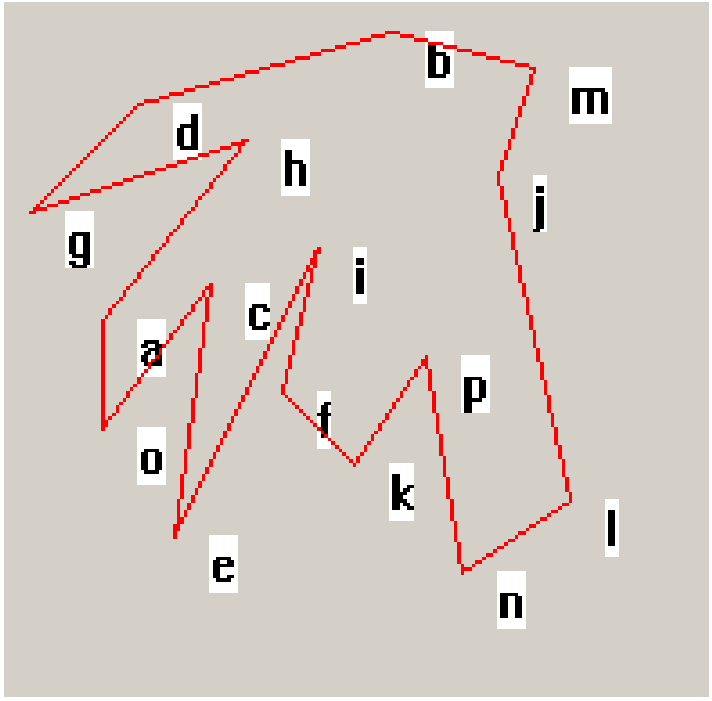



```
struct point{int x; int y;};
int ccw(point p0, point p1, point p2)
{
  int dx1=p1.x-p0.x;
  int dy1=p1.y-p0.y;
  int dx2=p2.x-p0.x;
  int dy2=p2.y-p0.y;
  return dx1*dy2-dy1*dx2;
}
```

As a good example of a set of points for testing purposes you may use 16 points:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
x	30	110	60	40	50	80	10	70	90	140	100	160	150	130	30	120
y	90	10	80	30	150	110	60	40	70	50	130	140	20	160	120	100




b
m
d
h
j
g
i
c
a
p
f
o
k
l
e
n

Simple Closed Path

```
void arrange()
{
 int min=0;
 for(int i=1; i<n; i++) if(p[i].y<p[min].y) min=i;
 for(int i=0; i<n; i++)
  if(p[i].y==p[min].y) if(p[i].x > p[min].x) min=i;
 swap(p[0],p[min]);
 sort(p+1,p+n,cmp);
}
```

where as a `cmp` function you may use:

```
struct dpoint{double x; double y;};
bool cmp(dpoint p1, dpoint p2)
{return
  atan2(p1.y-p[0].y, p1.x-p[0].x) <
  atan2(p2.y-p[0].y, p2.x-p[0].x);
}
```

Or, not to involve math library (`#include<cmath>`):

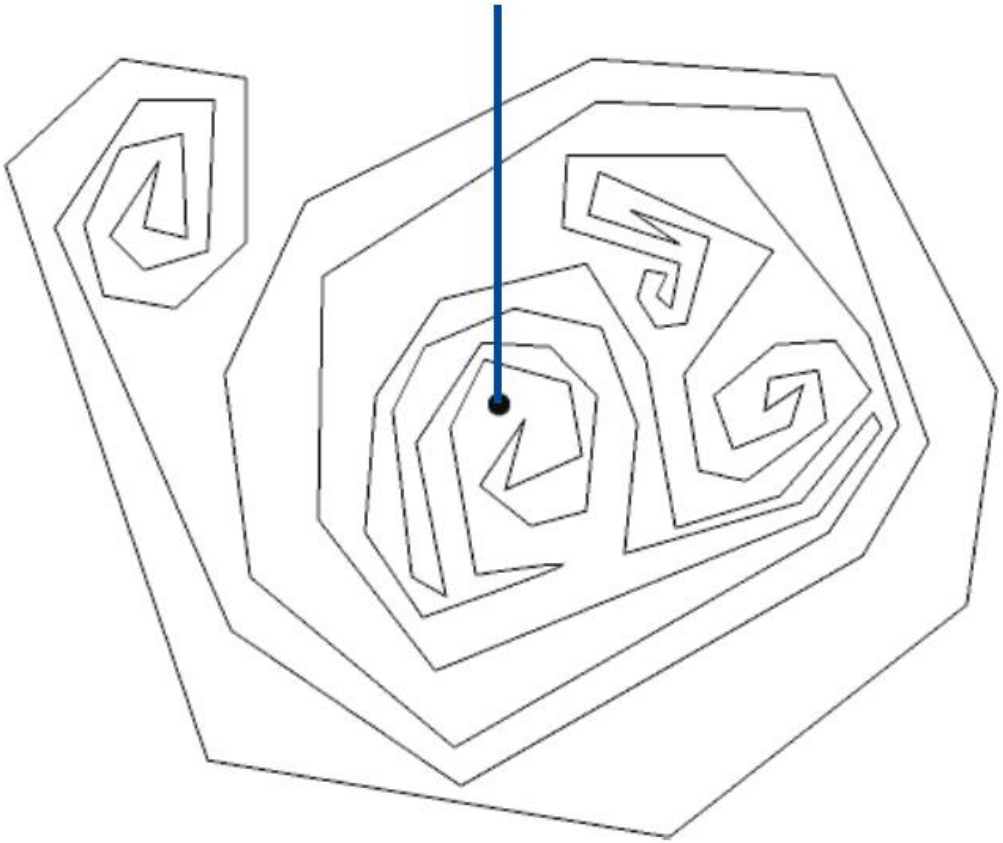
```
bool cmp(dpoint p1, dpoint p2)
{
  dpoint px;
  px.x = p[0].x;
  px.y = p[0].y+1.0;
  double v1 = ccw(p[0], px,
            p[0]+(1.0/norm(p1,p[0]))*(p1-p[0]));
  double v2 = ccw(p[0], px,
            p[0]+(1.0/norm(p2,p[0]))*(p2-p[0]));
  return v1 < v2;
}
```

where for shortening, a redefinition of `+`, `-`, *, and `norm` function for `point` is implemented:

```
dpoint operator+(dpoint p1, dpoint p2)
{ dpoint p; p.x=p2.x+p1.x; p.y=p2.y+p2.y; return p;}

dpoint operator-(dpoint p1, dpoint p2)
{ dpoint p; p.x=p2.x-p1.x; p.y=p2.y-p2.y; return p;}

dpoint operator*(double d, dpoint p)
{ dpoint p1; p1.x=d*p.x; p1.y=d*p.y; return p1;}

double norm(dpoint p1, dpoint p2)
{return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-
p2.y));}
```

Is a point inside a simple polygon?

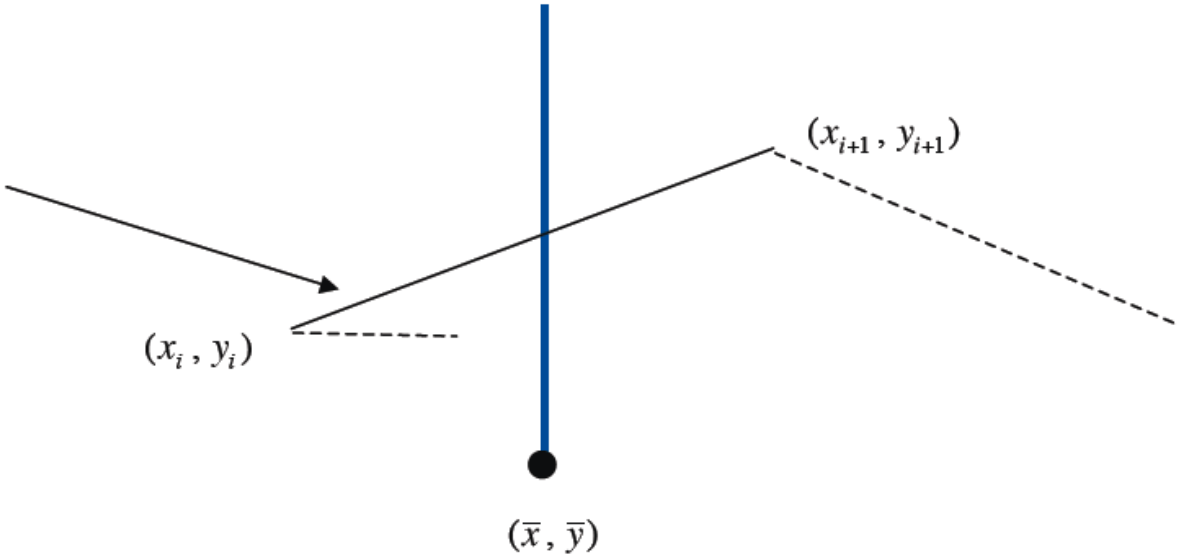
Jordan curve theorem. Any continuous simple closed curve cuts the plane in exactly two pieces: the inside and the outside.



Does line segment intersect a given ray?

$$y = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + y_i$$

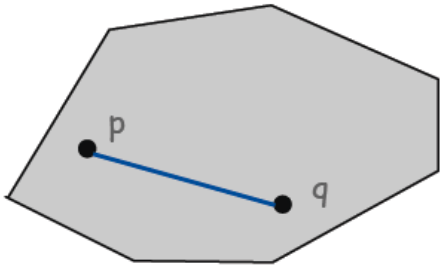
where $x_i \leq x \leq x_{i+1}$



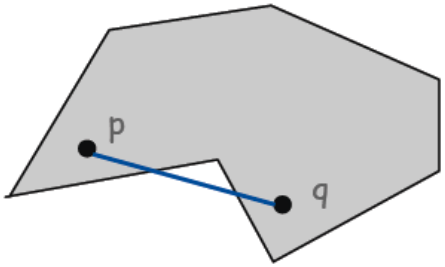
Convex Hull

A set of points is convex if for any two points p and q in the set, the line segment pq is completely in the set.

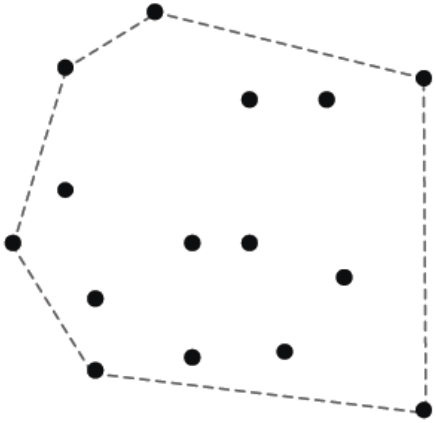
Convex hull: The smallest convex set containing all the points.



convex



not convex

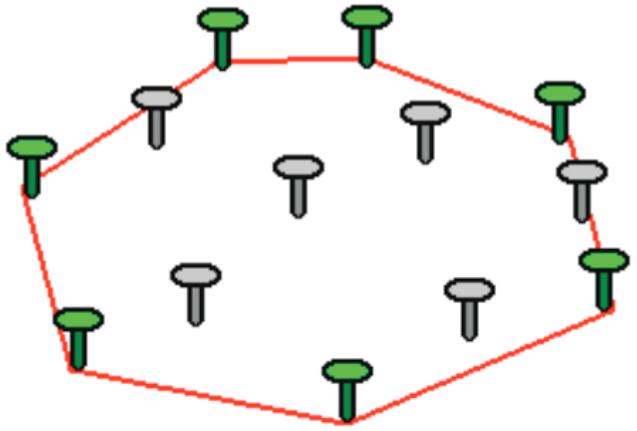


convex hull

Properties.

- "Simplest" shape that approximates set of points.
- Shortest (perimeter) fence surrounding the points.
- Smallest (area) convex polygon enclosing the points.

Mechanical Solution: Hammer nails perpendicular to plane; stretch elastic rubber band around points.

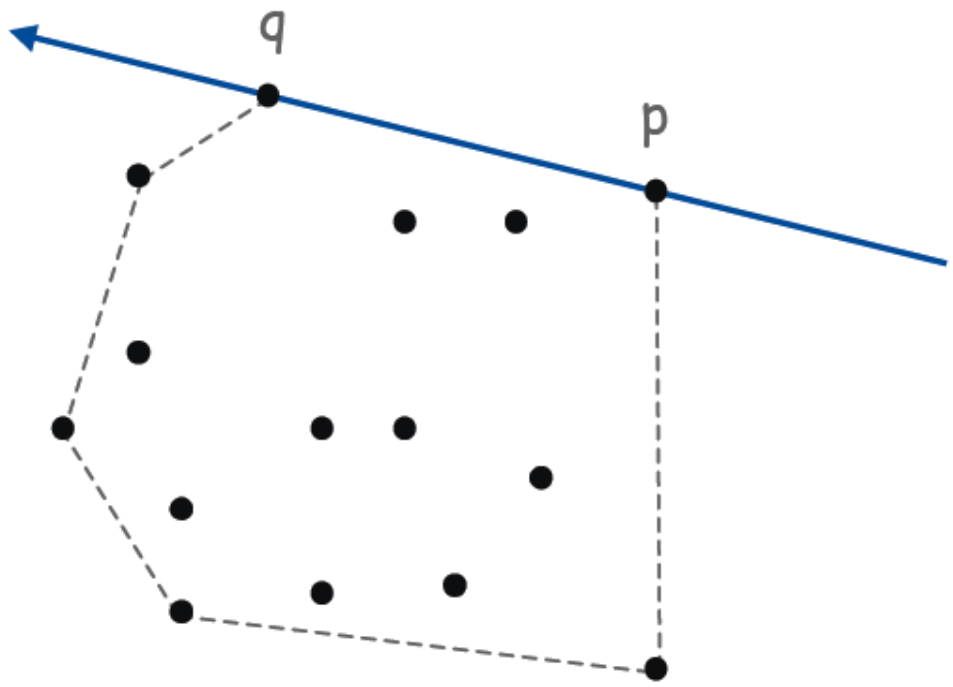


Brute Force

Observation 1. Edges of convex hull of P connect pairs of points in P.

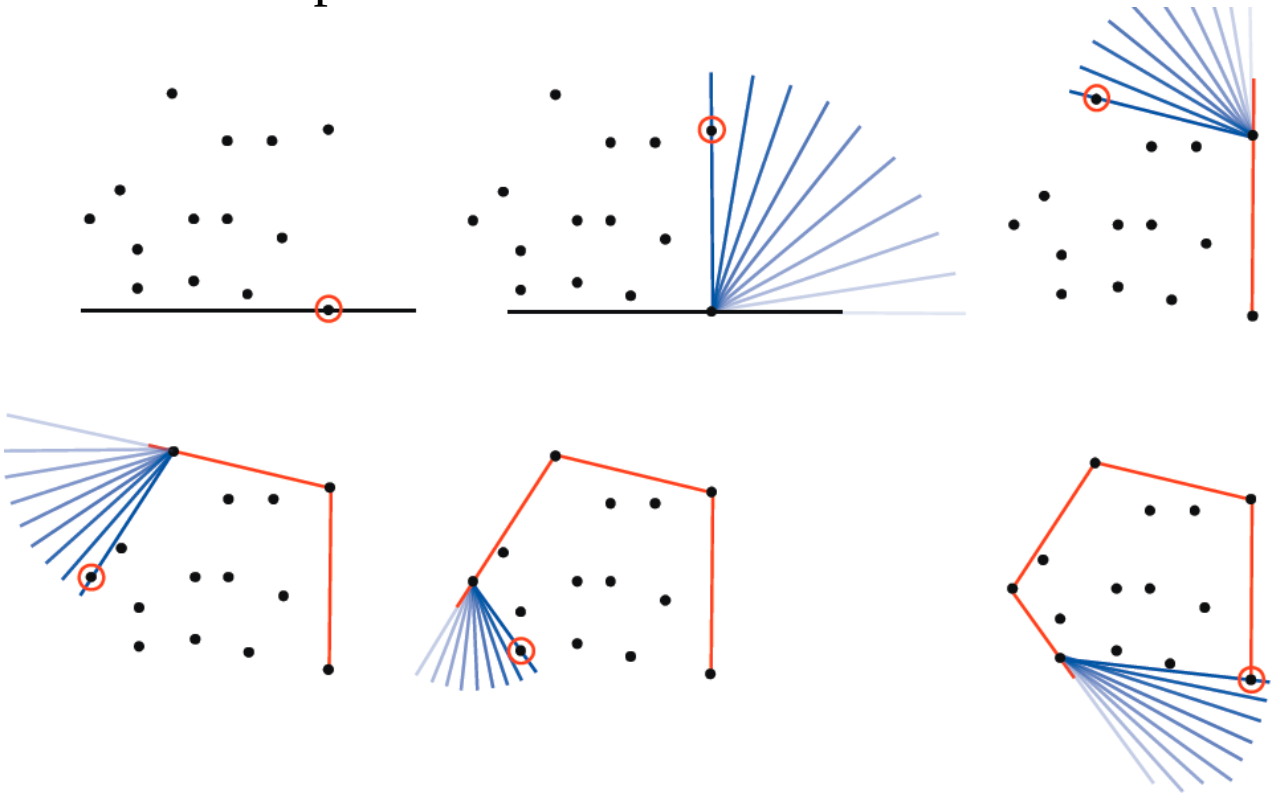
Observation 2. Edge p→q is on convex hull if all other points are counterclockwise of p→q.

$O(N^3)$ algorithm. For all points p and q in P, check whether pq is an edge of convex hull. Each such check requires O(N) ccw calculations, where N is the number of points in P.



Package wrap.

- Start with point with smallest y-coordinate.
- Rotate sweep line around current point in ccw direction.
- First point hit is on the hull.
- Repeat.



Implementation.

- Compute angle between current point and all remaining points.
- Pick smallest angle larger than current angle.
- O(N) per iteration.

Parameters.

- N = number of points.
- h = number of points on the hull.

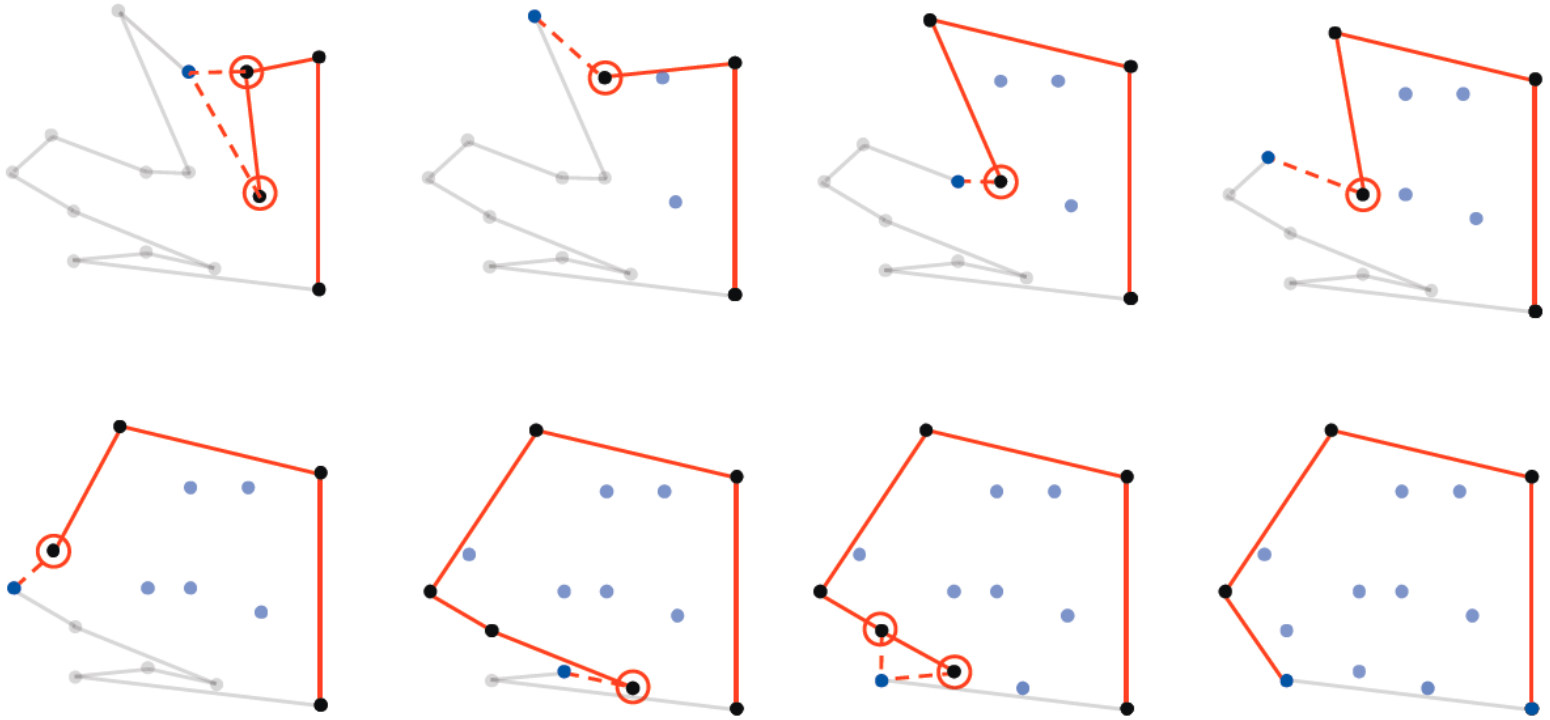
Package wrap running time. O(N h) per iteration.

Graham scan.

– Choose point p with smallest y-coordinate.

– Sort points by polar angle with p to get simple polygon.

– Consider points in order, and discard those that would create a clockwise turn.



Finding the Convex Hull by Graham scan

// a polygon is given by its vertexes p[0], p[1], .., p[n-1]

```
p[n]=p[0];
int m=2;
for(int i=3;i<=n;i++)
{
  while(ccw(p[m],p[m-1],p[i])>=0) m--;
  m++; swap(i,m);
}
n=m;
```

Quick elimination.

- Choose a quadrilateral Q or rectangle R with 4 points as corners.
- If point is inside, can eliminate.

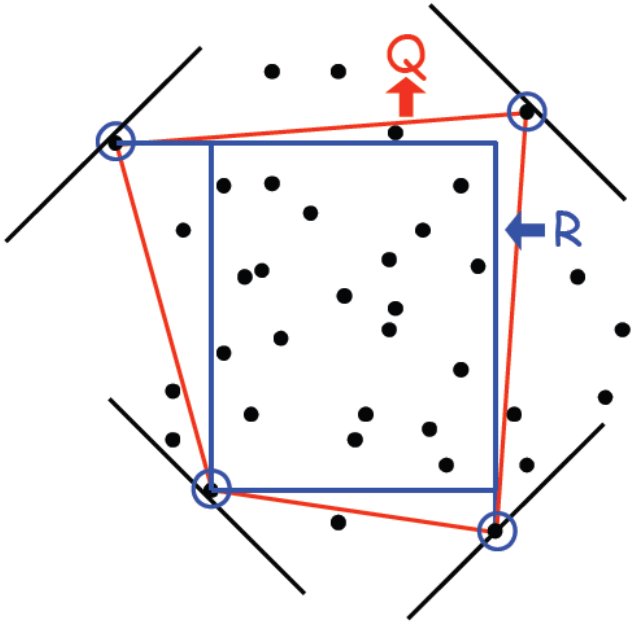
– 4 ccw tests for quadrilateral

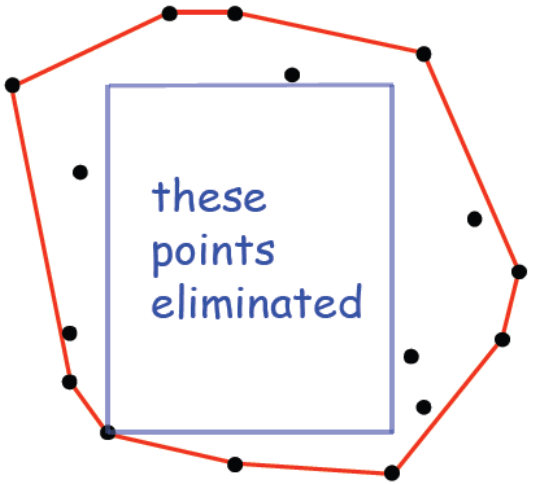
– 4 comparisons for rectangle

Three-phase algorithm

- Pass through all points to compute R.
- Eliminate points inside R.
- Find convex hull of remaining points.

Practice. Can eliminate almost all points in linear time.

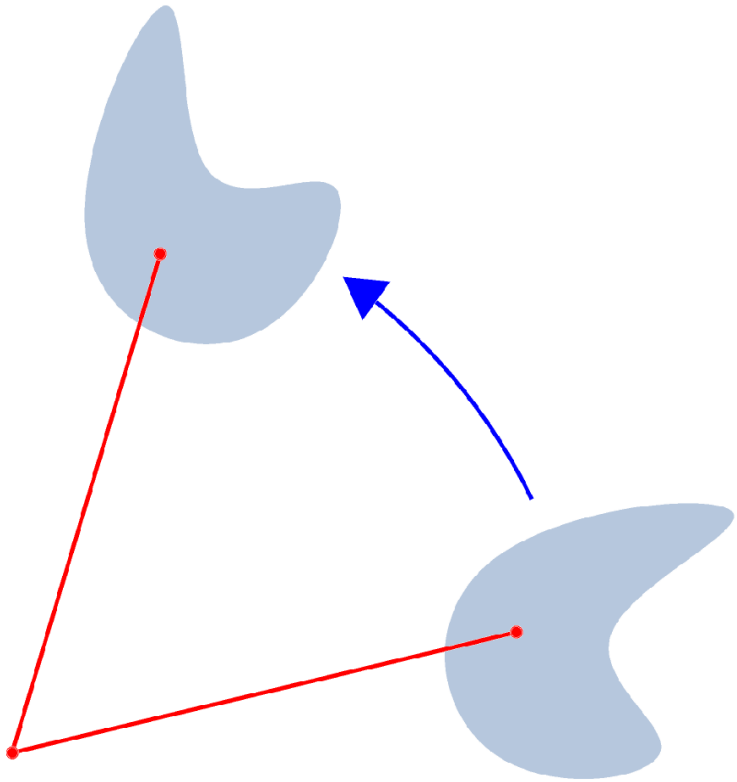




Geometric transformations

1. Translation (moving every point on a constant distance in a specified direction**)**

2. Rotation (movement which keeps one point fixed)



$$x' = x\cos\theta - y\sin\theta,$$
$$y' = x\sin\theta + y\cos\theta.$$

Matrix representation $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

3. Axial symmetry (Symmetry about an Axis) **- reflection** (also spelled **reflexion**) is a transform that maps an object into its mirror image.
A reflection about a line L through the origin which makes an angle θ with the x-axis is described by:

Matrix representation $\begin{bmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{bmatrix}$

4. Point symmetry (Symmetry about a Point)

Exercises:

1. Find a formula for the area of a given polygon P_1, P_2, ..., P_N. Suppose that there exists an inner point P that divides the area into disjoint triangles, formed by vertices (P, P_i, P_{i+1}), i = 1, 2, ..., N – 1 and (P,P_N,P_1).
2. Write a program to find if a given point is placed inside a given polygon.
3. Write a program to check if a given polygon is convex.
4. Write a program to check if two given line segments intersect.
5. Write a program to find the coordinates of a common point of two straight lines and two straight line segments.
6. Given points A, B, and C, write a program to find a point D, such that CD is perpendicular to AB and D lies on the straight line AB.
7. Given points A, B, and C, write a program to find a point D, which is symmetrical to C in respect of straight line symmetry of AB.
8. Derive a formula for transformation of coordinates intended for Symmetry about a Point.
9. Write a complete program to find the convex hull of a given set of points.