# Computationnal geometry

Guillaume Derval

April 17, 2016

# Table of Contents

# Table of Contents

# Points

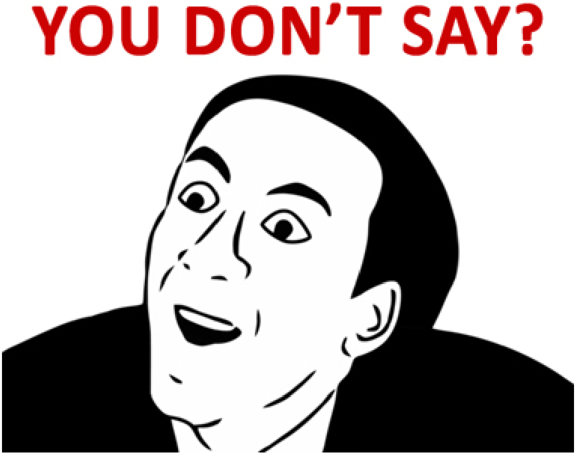
- Points in 2D: (x, y)
- Distance between two points $a = (x_1, y_1), b = (x_2, y_2)$:

$$\text{dist}(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

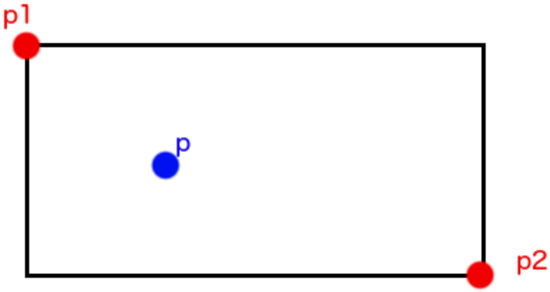
# Points

- Points in 2D: (x, y)
- Distance between two points $a = (x_1, y_1), b = (x_2, y_2)$:

$$\text{dist}(a, b) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$



# Check that a point is in a box




```
boolean inBox(Point p1, Point p2, Point p) {
    return
        Math.min(p1.x, p2.x) <= p.x &&
        p.x <= Math.max(p1.x, p2.x) &&
        Math.min(p1.y, p2.y) <= p.y &&
        p.y <= Math.max(p1.y, p2.y);
}
```

# Lines

General formula:

$$Ax + By = C$$

The line through $(x_1, y_1), (x_2, y_2)$ is given by:

$$A = y_2 - y_1$$

$$B = x_1 - x_2$$

$$C = Ax_1 + Bx_2$$

# Distance from a point to a line

Euclidian distance from a point $(x_0, y_0)$ to a line $Ax + By = C$ is

$$\frac{|Ax_0 + By_0 + C|}{\sqrt{A^2 + B^2}}$$

# Line intersections

Two lines $A_1x + B_1y = C_1$ and $A_2x + B_2y = C_2$ intersects iff

$$d := \det \begin{pmatrix} A_1 & B_1 \\ A_2 & B_2 \end{pmatrix} \neq 0$$

# Line intersections

Two lines $A_1x + B_1y = C_1$ and $A_2x + B_2y = C_2$ intersects iff

$$d := \det \begin{pmatrix} A_1 & B_1 \\ A_2 & B_2 \end{pmatrix} \neq 0$$

remeber that:

$$\det \begin{pmatrix} A_1 & B_1 \\ A_2 & B_2 \end{pmatrix} = A_1B_2 - A_2B_1 = d$$

# Line intersections

Two lines $A_1x + B_1y = C_1$ and $A_2x + B_2y = C_2$ intersects iff

$$d := \det \begin{pmatrix} A_1 & B_1 \\ A_2 & B_2 \end{pmatrix} \neq 0$$

remeber that:

$$\det \begin{pmatrix} A_1 & B_1 \\ A_2 & B_2 \end{pmatrix} = A_1B_2 - A_2B_1 = d$$

The intersection is then given by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A_1 & B_1 \\ A_2 & B_2 \end{pmatrix}^{-1} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \frac{1}{d} \begin{pmatrix} B_2 & -B_1 \\ -A_2 & A_1 \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}$$

# For those who don't like matrices

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A_1 & B_1 \\ A_2 & B_2 \end{pmatrix}^{-1} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \frac{1}{d} \begin{pmatrix} B_2 & -B_1 \\ -A_2 & A_1 \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix}$$

$$\rightarrow$$

$$x = \frac{B_2 C_1 - B_1 C_2}{d},$$

$$y = \frac{A_1 C_2 - A_2 C_1}{d}$$

# Perpendicular line in 2D

The line perpendicular to $Ax + By = C$ are

$$-Bx + Ay = D \text{ for } D \in \mathbb{R}$$

If you want that the line goes through $(x_0, y_0)$:

$$D = -Bx_0 + Ay_0$$

# Orthogonal symmetry

For a line $a$, and a point $x$, find its orthogonal symmetry point $x'$:

1. Compute the perpendicular $b$ of $a$ that goes throught $x$
2. find the intersection $y$ of $a$ and $b$
3. $x' = y - (x - y)$

# Orientation

$$orient(p, q, r) = \begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix}$$

$$orient(p, q, r) \begin{cases} = 0 & p, q, r \text{ are collinear} \\ < 0 & p \text{ -> } q \text{ -> } r \text{ is clockwise} \\ > 0 & p \text{ -> } q \text{ -> } r \text{ is counterclockwise} \end{cases}$$

$$|orient(p, q, r)| = 2 \cdot area \triangle(p, q, r)$$

```
double orient(Point p, Point q, Point r) {
    return q.x * r.y - r.x * q.y - p.x * (r.y - q.y) +
    p.y * (r.x - q.x);
}
```

# Angle visibility

$x$ lies strictly inside the angle formed by $p, q, r$ iff

$$sgn(orient(p, q, x)) = sgn(orient(p, x, r))$$
$$sgn(orient(p, r, x)) = sgn(orient(p, x, q))$$

To allow it to lie on the border simply check if

$$sgn(orient(p, q, x)) = 0 \text{ or } sgn(orient(p, r, x)) = 0$$

# Triangles

Notations and definitions:

- sides a, b, c
- angles $\alpha$, $\beta$, $\gamma$
- perimeter $p = a + b + c$
- semi-perimeter $s = \frac{p}{2}$
- Area

$$A = \frac{\text{base} * \text{height}}{2}$$

- Heron's formula

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

# Triangles (2)

- Inscribed circle radius

$$r = \frac{A}{s}$$

- Center of inscribed circle: intersection of the bisectors of the angles
- Circumscribed circle radius:

$$R = \frac{abc}{4A}$$

- Center of circumscribed circle: intersection of the heights

# Triangles (3)

- Law of sines

$$\frac{a}{\sin(\alpha)} = \frac{b}{\sin(\beta)} = \frac{c}{\sin(\gamma)} = 2R$$

- Law of cosines

$$c^2 = a^2 + b^2 - 2ab\cos(\gamma)$$

# Circles

Definitions and notations

- Radius $r$ and center $c = (a, b)$
- Equation that point on the circle respects:

$$(x - a)^2 + (y - b)^2 = r^2$$

- Diameter

$$D = 2r$$

- Area

$$A = \pi r^2$$

- Perimeter

$$p = 2\pi r$$

(also called circumference)

# Circles (2)

- Arc: connected section of the circumference of the circle. Given the angle $\alpha$ made, the size of an arc is

$$\frac{p}{\alpha} 2\pi$$

- Chord: ligne segment whose endpoints are on a circle. Given the angle $\alpha$ made, the size of the chord is:

$$\sqrt{2r^2(1 - \cos(\alpha))}$$

- Sector: area between two radiuses and the arc between these two radiuses. Given the angle $\alpha$ made:

$$A\frac{\alpha}{2\pi}$$

- Segment: sector minus the triangle made by the chord

# Table of Contents

# Computationnal geometry, in one slide

Four things to remember

# Computationnal geometry, in one slide

Four things to remember

- floating point operations make computation errors

# Computationnal geometry, in one slide

Four things to remember

- floating point operations make computation errors
- computing on floating point lead to precision errors

# Computationnal geometry, in one slide

Four things to remember

- floating point operations make computation errors
- computing on floating point lead to precision errors
- computations on floats and doubles are nearly always approximative

# Computationnal geometry, in one slide

Four things to remember

- floating point operations make computation errors
- computing on floating point lead to precision errors
- computations on floats and doubles are nearly always approximative
- I think you have understood

# Example of error

```
printf (" %.20f \n", 3.6);
```

Output:

```
3.60000000000000008882
```

# Handling errors

Define an "error threshold" $\epsilon$:

```
boolean eq(double a,double b){return Math.abs(a - b) <=
    E;}
boolean le(double a,double b){return a < b - E;}
boolean leq(double a,double b){return a <= b + E;}
```

# Table of Contents

# Storing a polygon

Polygon = serie of points. Store them in an ordered way, for example, clockwise.

```
Point[] polygon;
```

# Perimeter of a polygon

Simply iterate on all the points

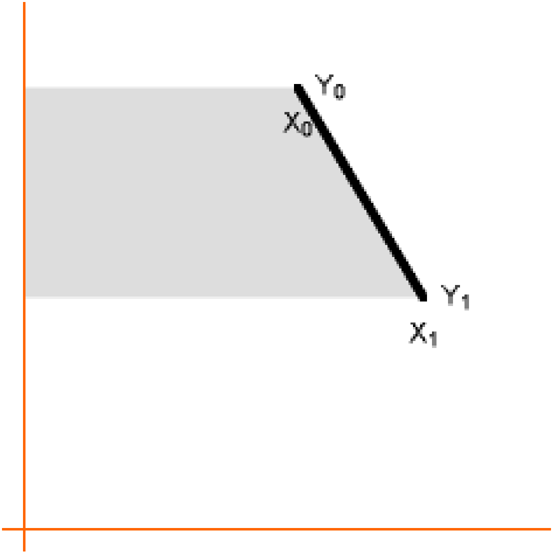
```
double p = 0;
for(int i = 0; i < polygon.length; i++) {
  p += dist(polygon[i], polygon[(i+1)%polygon.length
  ]);
}
```

# Area of a polygon

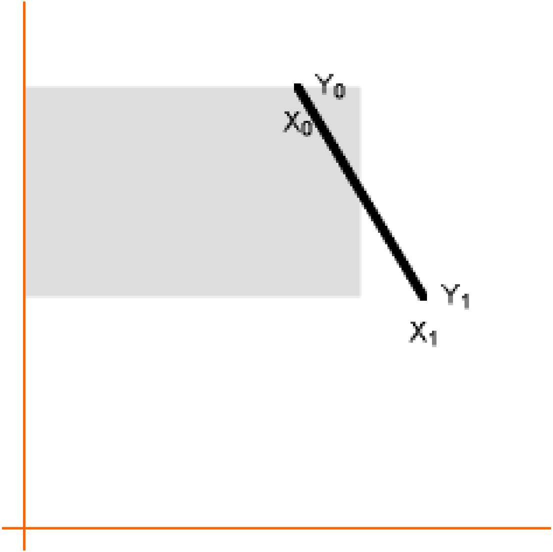
# Area of a polygon

Idea: get the area below all successive pair of points, and add/substract it depending on orientation

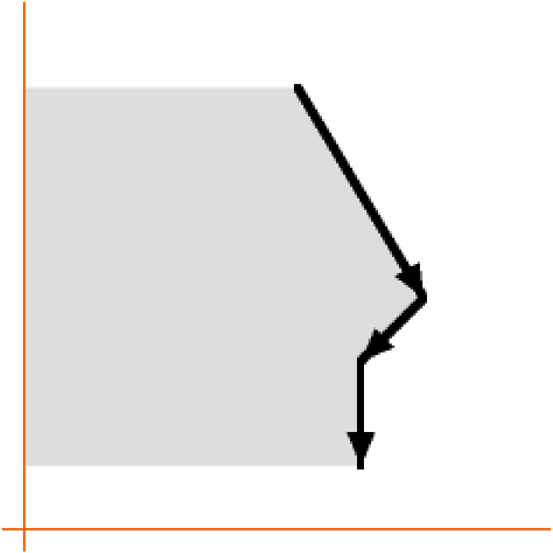
# Area of a polygon (2)



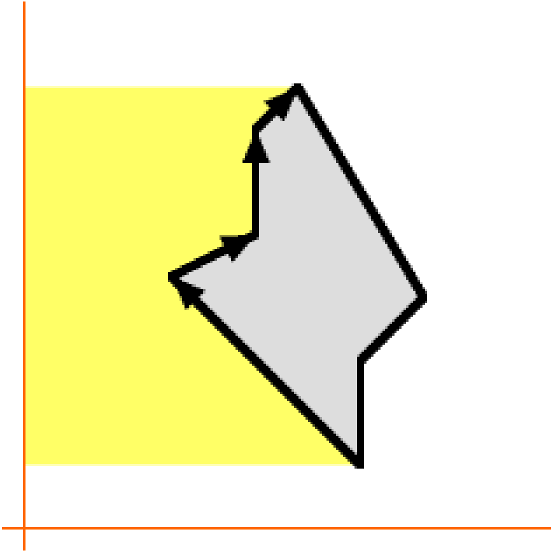
# Area of a polygon (3)



# Area of a polygon (4)



# Area of a polygon (5)



# Area of a polygon (6)

$$A = \frac{1}{2}\begin{vmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ \dots & \dots \\ x_n & y_n \end{vmatrix} = \frac{1}{2} * (x_0y_1 + x_1y_2 + \ldots + x_ny_0 - x_1y_0 - x_2y_1 - \ldots - x_0y_n)$$

# Area of a polygon (7)

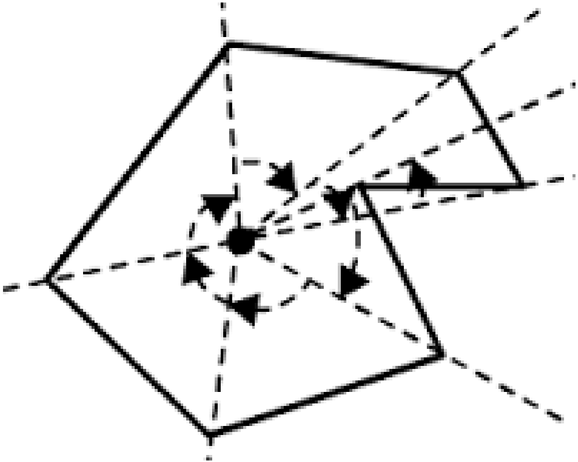
```
double result = 0;
for(int i = 0; i < polygon.length()-1; i++) {
  result += (P[i].x * P[i+1].y - P[i+1].x * P[i].y);
}
```

# Checking if a polygon is convex

→ check that *orient(p,q,r)* is always positive for all successive triplet of points

# Check if point is inside a polygon

$\rightarrow$ compute the angle made by all the successive pairs of points and the point we want to check. If it is 360 degrees, the point is inside the polygon.



# Table of Contents

# Convex Hull

Given a set of points in the plane, compute the smallest convex polygon in the plane that contains all the points.

# Graham Scan

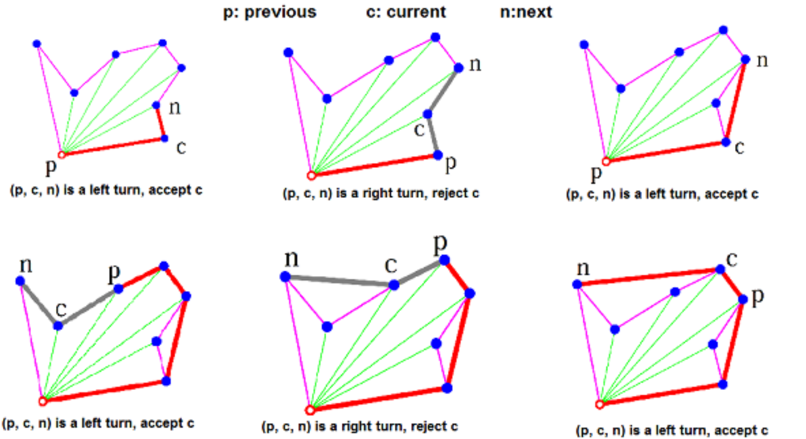
Method of computing the convex hull of a finite set of points in the plane with time complexity O(n log(n)). The algorithm finds all vertices of the convex hull ordered along its boundary.

# Graham Scan

## Idea

- find the point with the lowest y-coordinate. if there are multiple points with the lowest y-coordinate, the pick that one of them with the lowest x-coordinate. call this point P
- now sort all the points in increasing order of the angle they and point P make with the x-axis
- consider each point in the sorted array in sequence. For each point determine whether coming from the 2 previous points it makes a left of a right turn.
- if it makes a left turn, proceed with the next point
- if it makes a right turn, the second-to-last point is not part of the convex hull and should be removed from the convex hull, continue this removing for as long as the last 3 points make up a right turn

# Graham Scan




In the above algorithm and below code, a stack of points is used to store convex hull points. With reference to the code, p is next-to-top in stack, c is top of stack and n is points[i].

# Graham Scan

## Direction of the turn

To determine whether 3 points constitute a left or a right turn we do not have to compute the actual angles but we can use a cross product.
Consider the 3 points $(x_1, y_1)$, $(x_2, y_2)$ and $(x_3, y_3)$, which we will call $P_1, P_2$ and $P_3$.
Now compute the z-component of the cross product of the vectors $P_1P_2$ and $P_1P_3$. Which is given by the expression
$(x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$.
If the result is 0, the points are collinear. If the result is positive, the 3 points constitute a left turn. If the result is negative, the 3 points constitute a right turn.

# Sorting points by angle

Here is a comparator (in C++)

```
point pivot(0, 0);
bool angleCmp(point a, point b) {
  if(orient(pivot, a, b) == 0) //collinear: return
the closer one
    return dist(pivot, a) < dist(pivot, b);
  double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
  double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
  return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0;
//compare angles
}
```

# Exercices

Do them in this order!

- 634
- 10060
- 478
- 681
- 109