Computational Geometry

Alvan Caleb Arulandu & Sameer Gabbita

March 2021

1 Introduction

Geometry is a branch of math concerning points, lines, and higher dimensional surfaces. It is also a common topic in higher divisions of competitive programming contests including but not limited to USACO, ICPC, and Codeforces. This lecture will cover common computational geometry algorithms as well as basic mathematical tricks to assist you in solving harder problems.

2 Vectors

A vector is a geometric object with a magnitude and direction that can be represented as a list of components in each dimension for most intensive purposes.

$$\vec{x} = \mathbf{x} = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^n$$

Vectors can be represented by drawing an arrow from an arbitrary starting point (say the origin) to an end point given by the components of the vector.

2.1 Elementary Operations

2.1.1 Scalar Multiplication

To multiply a vector $\vec{x} \in \mathbb{R}^n$ by scalar $s \in \mathbb{R}$, simply multiply each component by the scalar:

$$s\vec{x} = \vec{y} = [y_1, y_2, \cdots, y_n] : y_i = s \cdot a_i \, \forall \, 1 \leq i \leq n$$

2.1.2 Addition

To add to vectors $\vec{a}, \vec{b}$, simply add up each component:

$$\vec{a} + \vec{b} = \vec{c} = [c_1, c_2, \cdots, c_n] : c_i = a_i + b_i \, \forall \, 1 \leq i \leq n$$

To subtract vectors, simply add the $(-1)\vec{b}$ which can be calculated with scalar multiplication.

2.1.3 Magnitude

The magnitude of $\vec{v}$ can be interpreted as the length of the "arrow" from the start to end point. Using a generalized Pythagorean Theorem we have,

$$|\vec{v}| = v = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Unit vectors are vectors with magnitude 1. These can often make computations easier where magnitude is irrelevant.

$$\hat{v} = \frac{\vec{v}}{|\vec{v}|}$$

In 3D space $\hat{i}, \hat{j}, \hat{k}$ represent the 3 unit vectors lying on the axes x, y, z respectively. Note that every vector can be written as a sum of it's components times a unit vector along each component axis.

2.1.4 Dot Product

There are two types of vector multiplication, one of which is known as the dot product. The dot product computes a sum of element-wise multiplications of two vectors yielding a scalar.

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i$$

The vector dot product also has a nice geometric interpretation when it comes to vector projection:

$$\vec{a} \cdot \vec{b} = |a||b| \cos\theta$$

Here we say that $\vec{b}$ is projected onto $\vec{a}$. As shown by the above relationship with cos the vector dot product yields useful information about the angle between two vectors in an $O(n)$ computation.

2.1.5 Cross Product

The second type of vector multiplication is known as the cross product and is a method of multiplying two vectors to yield a third orthogonal vector. We can represent the vector cross product as a matrix determinant:

$$\vec{a} \times \vec{b} = |\vec{a}||\vec{b}| \sin\theta \cdot \hat{n}$$

Note that $\hat{n}$ is the unit vector orthogonal to $\vec{a}, \vec{b}$. For $n = 3$ the following notational trick can be used for computation:

$$= \det\left(\begin{bmatrix} \hat{x_1} & \hat{x_2} & \hat{x_n} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix}\right)$$

The cross product also has a ton of neat relationships and applications.

$$|\vec{a} \times \vec{b}| = |\vec{a}||\vec{b}| \sin\theta$$

2.1.6 Uses

Vector properties are frequently used in distance computations between various objects (commonly seen in multi-variable calculus). Additionally, the multiplicative properties of vectors can be exploited to determine if line segments are perpendicular.

3 Coordinate Systems

Most readers are well aware of Cartesian coordinates (x, y). However, in some cases other coordinate systems like polar can be used to solve problems. Instead of relying on rectangular position, polar coordinates represent points with a radius r and a phase/angle θ; (r, θ). In this system,

$$r^2 = x^2 + y^2$$

$$\tan\left(\frac{y}{x}\right) = \theta$$

Please note that this implies that the same point can be represented in infinite ways in polar coordinates. This only applies to two dimensions but there are similar systems available for three dimensions. Cylindrical coordinates are similar to polar coordinates with an added $z = z$ equality; (r, θ, z). Spherical coordinates introduce a third angular parameter ϕ which is the angle from the radius to the z axis; (ρ, θ, ϕ).

4 Sweep Line Algorithms

Sweep-line algorithms are a set of algorithms that follow a similar idea regarding, you guessed it - a sweeping line! These algorithms imagine sweeping a line across a plane stopping at points that intersect or are in close proximity to the line. A certain operation is then done on these points such that a complete solution is available by the time the line crosses the entire plane.

4.1 Line Segment Intersection

The line segment intersection problem consists of detection and calculation. One task is to determine if a set of line segments have an intersection. The other is to determine all intersections of the line segments.

4.1.1 Two Segments

The "algorithm" for two segments is fairly simple and amounts to a plethora of if statements checking orientation cases for detection and Cramer's rule for calculation.

4.1.2 Bentley-Ottmann Algorithm

However, for $n > 2$ line segments, is harder. A naive approach (testing every pair) takes $O(n^2)$. But, using this sweep line algorithm allows us to calculate a list of intersection points from a set of line segments in $O((n+k)\log n)$ time where n is the number of line segments and k is the number of intersections. Note that worst case, $k = \frac{n(n-1)}{2}$.

4.1.3 Shamos-Hoey Algorithm

As a side-note, this algorithm applies binary search trees to this problem yielding $O(n \log n)$ complexity to detect line segment intersections.

5 Convex Hull

Convex Hull is an algorithm that can find the smallest convex polygon that contains all given points (aka the convex hull) for 2D or 3D points and are bounded by the upper bound theorem for higher dimensions. Naively, finding the convex hull takes $O(n^3)$ time where we can iterate through all possible line segments, and add it to the convex hull if all the other points are to its left.

5.1 Graham Scan

```
def graham_scan(points):
    st = []
    p0 = point with smallest y-coordinate
    sort_ccw(points)
    st = [points[0], points[1]]
    for i in range(2, len(points)):
        while len(stack) > 2 and points[i] is right of st[-1] and st[-2]:
            st.pop()
        st.push(points[i])
```

5.2 Andrew's Algorithm

```
def andrews_algo(points):
    sort_by_x_coordinate(points)
    upper = []
    lower = []
```

```
for point in points:
    while direction() == left:
        upper.remove(second_to_last_point)
    upper.append(point)
for point in points:
    while direction() == right:
        lower.remove(second_to_last_point)
    lower.append(point)
convex_hull = merge()
return convex_hull
```

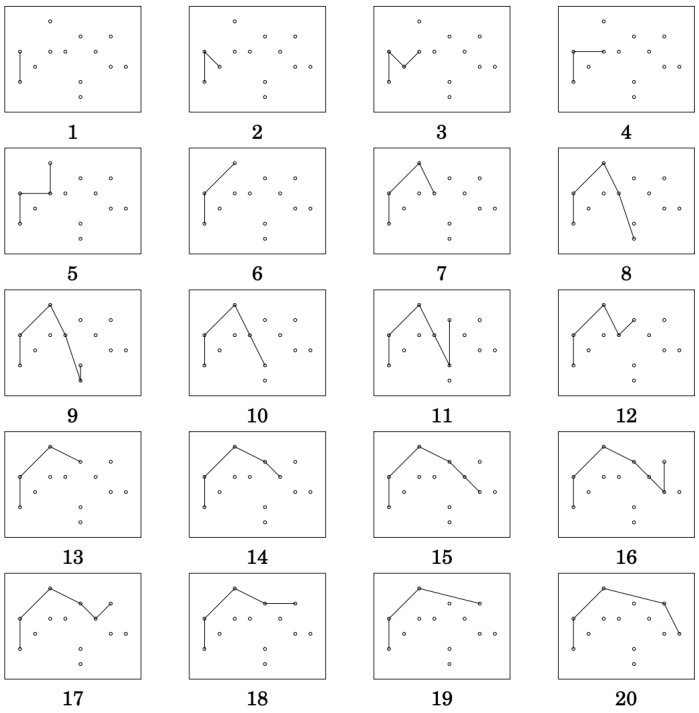

Figure 1: Andrew's Scan (From CPH)

6 Distances

6.1 Point-Plane

$$D = |\hat{n} \cdot (\vec{p} - \vec{q})|$$

where $\hat{n}$ is the unit normal vector of the plane, $\vec{p}$ is the point, and $\vec{q}$ is an arbitrary point on the plane.

6.2 Point-Point

$$D = |\vec{a} - \vec{b}|$$

where $\vec{a}, \vec{b}$ are vectors to the points of interest.

6.3 Point-Line

$$D = |\hat{n} \cdot (\vec{p} - \vec{q})|$$

where $\hat{n}$ is the unit normal vector from the line, $\vec{p}$ is the point, and $\vec{q}$ is an arbitrary point on the line.

6.4 Plane-Plane

If the planes are parallel, pick an arbitrary point on one and use Point-Plane. Otherwise, the planes must intersect so the distance is 0.

6.5 Line-Line

If the lines are parallel, pick an arbitrary point on one of the lines and use Point-Line. Otherwise, find the distance between the two parallel planes each containing one of the lines.

6.6 Plane-Line

If the line is parallel to the plane, then use Point-Plane with an arbitrary point on the line. Otherwise, they must intersect so the distance is 0.

6.7 Shoelace Theorem

6.7.1 Statement

Suppose polygon P has vertices $(a_1, b_1), (a_2, b_2), \cdots, (a_n, b_n)$ in clockwise order. Then the area A of P is

$$A = \frac{1}{2}|\sum_{i=1}^{n}(x_{i+1} + x_i)(y_{i+1} - y_i)|$$

$$= \frac{1}{2}|(a_1b_2 + a_2b_3 + \cdots + a_nb_1) - (b_1a_2 + b_2a_3 + \cdots + b_na_1)|$$

6.7.2 Algorithm

This method works by dividing up the polygon into many triangles formed from 2 vertices of the polygon and an origin O. We can then sum the areas of those individual triangles to find the total area of a polygon. To find the area of these individual triangles, we can use the cross-product, which gives the area of a parallelogram, and divide it by 2 to find the area of the triangle. As a result, this algorithm runs in $O(n)$ time where n is the number of points in the polygon.

6.8 Pick's Theorem

Pick's theorem can be used for lattice-point polygons as states,

$$\text{Area} = a + \frac{b}{2} - 1$$

where a is the number of lattice points strictly inside a polygon and b is the number of lattice points that are on the edges of a polygon.

6.9 Cramer's Rule

Cramer's rule is a way to analytically solve a system of equations using the determinant of a matrix. Given the following system of equations

$$\begin{aligned} a_1x + b_1y + c_1z &= d_1 \\ a_2x + b_2y + c_2z &= d_2 \\ a_3x + b_3y + c_3z &= d_3 \end{aligned}$$

we can solve the system of equations where $x = D_x/D$, $y = D_y/D$, $z = D_z/D$ given that $D \neq 0$ where

$$D = \det\left(\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}\right)$$

$$D_x = \det\left(\begin{bmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{bmatrix}\right)$$

$$D_y = \det\left(\begin{bmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{bmatrix}\right)$$

$$D_z = \det\left(\begin{bmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{bmatrix}\right)$$

7 Miscellaneous

This section contains a list of miscellaneous topics that you may/may not encounter in competitive programming problems but is included for sake of completeness and the reader's own interest.

7.1 Upper Bound Theorem

The theorem states that cyclic polytopes (polytopes formed as a convex hull around a rational normal curve where $n > d$) have the largest possible number of faces among all convex polytopes with a given dimension and number of vertices. Note that n is the number of points and d is the dimension of the space.

7.2 Area of a Triangle

$$[\triangle ABC] = \frac{bh}{2} = \frac{ab\sin C^\circ}{2} = \frac{|\vec{u} \times \vec{v}|}{2} = \sqrt{s(s-a)(s-b)(s-c)} = rs = \frac{abc}{4R}$$

7.3 Approximations / Expansions

$$\sin\theta = \sum_{k=0}^{\infty} \frac{(-1)^k \theta^{2k+1}}{(2k+1)!}$$

$$\cos\theta = \sum_{k=0}^{\infty} \frac{(-1)^k \theta^{2k}}{(2k)!}$$

8 Problems

8.1 Practice

- Cow Steeplechase II
- Water Balance
- CSES Geometry Problems
- Imperfect GPS
- Polygon Area
- Cut Length

8.2 Custom: Hurdles

8.2.1 Problem

Gleb is running the "free-meter" hurdles. There are N hurdles given as points (x_i, y_i) in the Cartesian plane such that $|x|, |y| < 10^9$ and no two points $(x_1, y_1), (x_2, y_2)$ are co-linear with the origin $(0, 0)$. Gleb starts at $(1, 0)$ and must cross each hurdle exactly once in any particular order before returning to the starting point to finish the race. There is no hurdle at the starting point. If Gleb runs at πe meters per second, what is the floor of the minimum time to finish the race so that Gleb can smoke the competition?

8.2.2 Input

Input starts with one line containing the integer $1 < N < 10^8$. The next N lines contain two comma separated integers x, y. Ex:

3
1 ,1
−1 ,0
0 ,−1

8.3 Output

Output should consist of one integer, the floor of the minimum time. For this example, the total distance is $1 + \sqrt{5} + 2\sqrt{2} \approx 6.06 \to 6$. Ex:

6

9 Sources

- Shoelace Theorem (AOPS)
- Shoelace Theorem (Mathologer)
- Pick's Theorem (AOPS)
- Andrew's Algorithm (USACO Guide)
- Graham Scan (CS133)