# Basics of Computational Geometry

Nadeem Mohsin

October 12, 2013

## 1 Contents

This handout covers the basic concepts of computational geometry. Rather than exhaustively covering all the algorithms, it deals with the simplest underlying ideas, and their applications. These will form a foundation on which more complex techniques can be built.

## 2 Vectors, Vectors Everywhere!

The first thing we're going to do is throw out the standard approaches you were taught in high-school geometry. For computational applications, it turns out that formulating things in terms of slopes and intercepts and what not is ugly and inelegant. There are far too many special cases involving division by zero and the like. Instead, we neatly sidestep most of these problems by formulating all our problems in terms of vectors.

You've probably seen these before - the standard definitions of 'a mathematical object that has magnitude and direction' apply. We're more interested in the geometric view of vectors, which we usually draw as a line with an arrow. The length of the line gives the magnitude of the vector, and its orientation gives the direction. (Hence the arrow.) Generally we call the start point (without the arrow) the *tail* of the vector, and the end point (with the arrow) the *head* of the vector.

### 2.1 Position Vectors

When dealing with points, we think in terms of **position vectors**. The position vector of a point $P$ is the vector whose tail is at the origin $O$, and whose head is at $P$. It's also pretty easy to represent - if $P = (x, y)$, then the vector is also represented by the two numbers $x$ and $y$. This is often written as $\begin{bmatrix} x \\ y \end{bmatrix}$. The position vector of $P$ is usually denoted by $\overline{P}$ or $\overline{OP}$.

In code, we usually use a class or a struct for this.

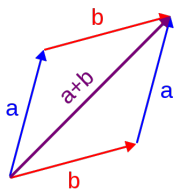
### 2.2 Addition and subtraction

We can add and subtract vectors pretty simply - just perform the operation with corresponding components.

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \end{bmatrix}$$

And analogously for subtraction.

The geometric view of addition is as follows. If you're adding position vectors $\overline{a}$ and $\overline{b}$, draw $\overline{a}$, then move $\overline{b}$ (without rotating it) so that its tail is at the head of $\overline{a}$. The position of $\overline{b}$'s head is the head of the




Vector addition

position vector $\overline{a}+\overline{b}$, while its tail is at the origin. If you do $\overline{b}$ first and then $\overline{a}$, the result will still be the same, as the figure shows.

The geometric view of subtracting $\overline{a}-\overline{b}$ is that it's the same as adding $\overline{a}$ and $-\overline{b}$. This is easy, since $-\overline{b}$ is the same as $\overline{b}$, but pointing in the exact opposite direction.

## 2.3 Scaling

Multiplying a vector by a scalar (a number) changes the magnitude but leaves the direction unchanged. For example, if you multiply a vector by 2, it will just become twice as long. Multiply by 1/3 and it will shrink to a third of its previous length.

$$c \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} cx \\ cy \end{bmatrix}$$

## 2.4 Magnitude

Sometimes we want to just talk about the magnitude of the vector $\overline{P}$, and we represent that as $||\overline{P}||$. If $P = (x, y)$, then $||\overline{P}|| = \sqrt{x^2+y^2}$. This should be pretty intuitive - it's just a basic application of the distance formula from coordinate geometry. (Or the theorem of Pythagoras - same difference.)

## 2.5 Direction

How do we measure the *direction* of a vector? The most natural way is to just use the angle it makes with the horizontal. The direction of $\begin{bmatrix} x \\ y \end{bmatrix}$ can be computed as $\theta = \arctan(y/x)$.

**Caution:** Division by zero is a major issue here. Fortunately, both C++ and Java have an `atan2` function that handles these things properly. Make sure you're familiar with it, and use it whenever you need a safe arctan. Also note that all these functions work with radians. A common novice mistake is to accidentally give it input in degrees, or to interpret the output in degrees. Remember this and convert as necessary.

## 2.6 Unit Vectors

Another way to represent a direction is by means of a *unit vector*. This is a vector of length 1, so it's only real significance is its direction. For example, the vector $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ is a unit vector pointing along the y-axis, while

$\begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ is a unit vector pointing 45° from the horizontal.

Given a vector $\overline{P}$, the unit vector along that direction is just $\frac{\overline{P}}{||\overline{P}||}$. In other words, rescale it to length 1 by dividing by the magnitude.

The unit vectors along the $x$ and $y$ axes are often referred to as $\hat{i}$ and $\hat{j}$ respectively. (If we're working in three dimensions, $\hat{k}$ is the z-axis unit vector.) This gives us another kind of notation. We can write the position vector of $(3,4)$ as $3\hat{i}+4\hat{j}$, for example.

### 2.7 Segments

We also frequently use vectors to represent a segment $AB$, where $A$ and $B$ are two points. The vector's magnitude is the length of the segment, and the direction is the direction of the segment.

Very conveniently, $\overline{AB} = \overline{B} - \overline{A}$. Just subtract the position vectors, but keep in mind that you're doing 'head minus tail'. If you do it the other way around, you'll get the reverse vector instead.

### 2.8 Polygons

We usually represent a polygon as a sequence of points, with the understanding that the last point is also adjacent to the first point. There are three broad classes of polygons.

- Convex: Non-intersecting edges, all interior angles $< 180$ degrees
- Concave: Non-intersecting edges, one or more interior angles $> 180$ degrees
- Complex: One or more intersecting edges

Generally we'll name the points of an $n$-gon $P_0$, $P_1$, ..., $P_{n-1}$. Because polygons are closed, we can say $P_n = P_0$.

## 3 Dot and Cross Products

Multiplying vectors is a bit complicated, but very useful. There are two types of vector products - one that returns a number, and another that returns a new vector. Most of our algorithms will be based on interesting properties of these products.

For the following discussion $p$ and $q$ are the vectors corresponding to $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ respectively.

### 3.1 Dot Product

The *dot product*, also called the *scalar product* of $\overline{p}$ and $\overline{q}$ is denoted as $\overline{p} \cdot \overline{q}$, and is equal to $x_1x_2 + y_1y_2$.

Alternatively, you can compute it as $||\overline{p}|| \, ||\overline{q}|| \cos\theta$, where $\theta$ is the angle between the two vectors.

### 3.2 Cross Product

The *cross product*, also called the *vector product* of $\overline{p}$ and $\overline{q}$ is denoted as $\overline{p} \times \overline{q}$. Unlike the dot product, the result of a cross product is actually a vector.

The magnitude of $\overline{p} \times \overline{q}$ is $x_1 y_2 - x_2 y_1$. Another way to compute it as $\|\overline{p}\| \, \|\overline{q}\| \sin\theta$, where $\theta$ is the angle between the two vectors.

The direction is a vector perpendicular to both $\overline{p}$ and $\overline{q}$, and is given by the *right-hand rule*.

Technically the magnitude should always be positive, but the expression given here can be negative. For the basic algorithms we'll be covering, we rarely ever care about the direction, but having this signed magnitude is very useful.

## 4 Basic Applications

### 4.1 The zero tests

If the angle between two vectors is $\pi/2$ or $-\pi/2$ (i.e., they are perpendicular), the cosine factor will be zero, and therefore the dot product will be zero as well. This gives a very simple test for perpendicularity.

If the angle between two vectors is $0$ or $\pi$ (i.e., they point in the same direction or in opposite directions) then the sine will be 0, and therefore the cross product will be zero. The most common application of this test is to check if three points lie on a line. To test if $A$, $B$ and $C$ lie on a line, simply check if $\overline{AB} \times \overline{AC}$ is zero.

### 4.2 Magnitude

We can compute the magnitude of a vector $p$ in terms of the dot product, as $\sqrt{\overline{p} \cdot \overline{p}}$. When possible, you should work with the squared magnitude instead. This avoids the rounding errors of the square root, and avoids doubles altogether if all points have integer coordinates (pretty common).

### 4.3 Angle

To compute the angle between two vectors $\overline{p}$ and $\overline{q}$, we use the following relation:

$$\frac{\overline{p} \times \overline{q}}{\overline{p} \cdot \overline{q}} = \frac{\|\overline{p}\| \, \|\overline{q}\| \sin\theta}{\|\overline{p}\| \, \|\overline{q}\| \cos\theta} = \tan\theta$$

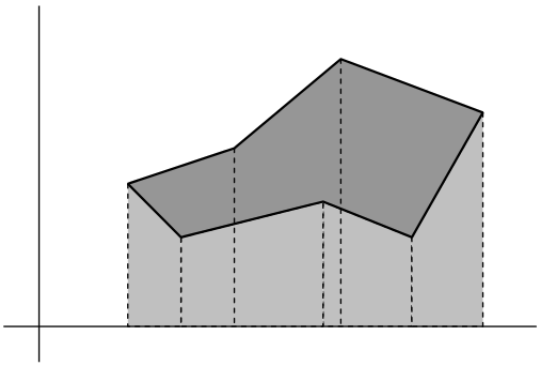
So to figure out $\theta$, just use `atan2` on the cross and dot products. Note that this angle can potentially be negative. This fact will be useful later for the point-in-polygon algorithm.

### 4.4 Area

The area of triangle ABC is $\frac{1}{2} \left|\left|\overline{AB} \times \overline{AC}\right|\right|$. If we have a parallelogram $ABCD$, its area is just $\left|\left|\overline{AB} \times \overline{AC}\right|\right|$.

A very important thing to note is that this area is a *signed* quantity, so it can be negative. Make sure you take the absolute value before using it.

As it turns out, signed areas really come in handy when we want to compute the area of a general polygon. For each edge of the polygon (two adjacent vertices), create a trapezoid by dropping a perpendicular from



each point onto the x-axis. Suppose you do this for the points $P_i(x_i, y_i)$ and $P_{i+1}(x_{i+1}, y_{i+1})$. Then the signed area of this trapezoid is given by:

$$A_i = \frac{1}{2}(x_{i+1} - x_i)(y_i + y_{i+1})$$

Sum up this quantity all around the polygon, and you get the signed area of the polygon! Again, don't forget to take the absolute value at the very end.

$$A = \sum_{i=0}^{n-1} A_i$$

As an exercise, prove to your satisfaction that the same result can be achieved by summing cross products of position vectors around the polygon.

$$A = \frac{1}{2} \sum_{i=0}^{n-1} ||\overline{P_i} \times \overline{P_{i+1}}||$$

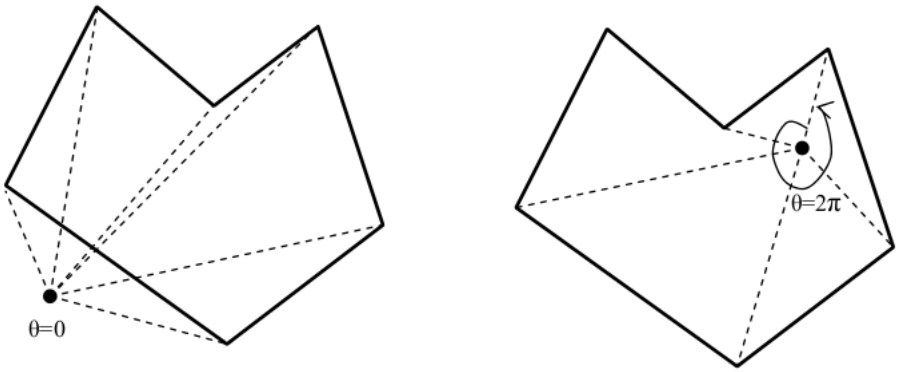
# 5 Point in Polygon

A common operation is to check if a point Q is contained inside a polygon or not. This can be done using the formula for the angle between two vectors. For each pair of adjacent vertices $P_i$ and $P_{i+1}$, take the vectors $\overline{QP_i}$ and $\overline{QP_{i+1}}$ and compute the angle between them. Sum this up all around the polygon, and take the absolute value of the result.

- If the result is 0, the point is outside the polygon.
- If it is $2\pi$, then it is inside.
- If it is $\pi$, then it is on an edge of the polygon.

This algorithm works for both convex and concave polygons. However, it fails when the point Q is the same as a vertex of the polygon, but this is quite easy to check.

When solving such a problem, make sure you clarify how it defines 'inside' and 'outside'. Cases where the point is on an edge or a vertex can be considered either of the two, or neither, depending on the specific problem.




The two major cases

# 6 Point-Line/Segment Distance

Several problems involve finding the distance from a point T to a line PQ. A variant of this problem is when PQ does not extend infinitely in both directions, i.e., it is a finite segment.

We define the distance from T to PQ as the length of the shortest segment connecting T to a point on PQ. For point-line distance, this is just the perpendicular dropped from T onto PQ.

If PQ is a segment, then it's possible that T is too far to one side, so that you can't drop a perpendicular onto PQ. In this situation, the shortest distance is from T to either P or Q, whichever is closer.

There are a couple of different ways to solve the problem. Here's the one that's intuitively easiest to grasp.

1. Rotate the coordinate system so that PQ is now horizontal.
2. Then just take the difference in y-coordinates.

That's all there is to it. Rotation turns the problem into a simple subtraction. Keep an eye out for other places to apply this sort of trick. Lots of geometric problems have very simple solutions if things are axis-aligned, or transformed in some similar way.

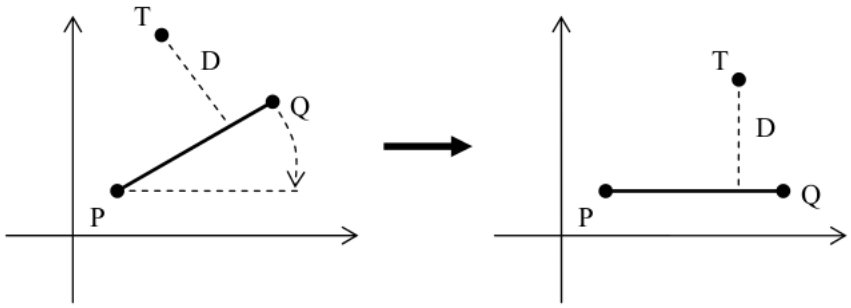
If you're trying to solve point-segment distance instead, first perform the rotation. Then look at the x-coordinates of T. If they lie between the x-coordinates of P and Q, then the situation is exactly the same as above. Otherwise, just pick the shorter of the two distances TP and TQ.

So the only question remaining is, how do we perform a rotation? As it turns out, vectors make that fairly easy too. The rotation matrix $R_\theta$ when multiplied by a vector returns the vector rotated by $\theta$.

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Thus the rotated version of $\begin{bmatrix} x \\ y \end{bmatrix}$ is

$$R_\theta \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$




Rotate the problem to make it simpler

To figure out how much we should rotate by, find the angle of PQ with the x-axis, and rotate by the *negative* of that. Then just follow the steps above.

# 7 Intersections

The final problem we'll deal with is actually something you've done in high-school geometry - how to find the intersection point of two lines. And as in the previous section, sometimes we want to intersect segments rather than lines, or a line with a segment.

One way to do this is the way you did it in school: set up a system of two equations and solve it to get the intersection. We'll use the same idea, but with a different kind of equation that generalizes better for our purposes.

## 7.1 Parametric form

Most problems you encounter will give you a line (or a segment) in the form of two points, rather than the equation of the line. Instead of turning this into a standard equation, we'll introduce a different representation, the *parametric form*.

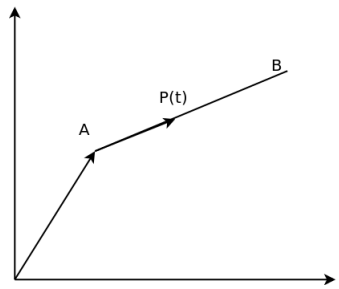
To understand the parametric equation for a line $AB$, we'll break it up into three pieces.

1. $\overline{A}$, which gives a starting point for our line
2. $\overline{AB}$, a vector that points along the line
3. $t$, a parameter that tells us how far along $\overline{AB}$ we should walk, starting from $\overline{A}$

Let $P(t)$ be the point that is a fraction $t$ along $AB$. For example, if $t = 1/2$, then it's at the midpoint between them. If $t = 0$, it's at $A$. For $t = 1$, it's at $B$. As long as $t$ is in the range $[0, 1]$, $P(t)$ is on the segment $AB$. If it goes out of that range, then it's on the extension of that segment.

We have

$$\overline{P(t)} = \overline{A} + t \cdot \overline{AB} = (1 - t)\overline{A} + t\overline{B}$$




Parametric view of a line segment

Remember that multiplying a vector by an integer scales it. So $t \cdot \overline{AB}$ is just a scaled version of $\overline{AB}$. Based on the geometric interpretation of vector addition, the equation basically says: Start at the origin, follow the position vector of A, and then travel along the scaled vector $\overline{AB}$. By varying $t$, we can get any point on the line. By restricting our attention to $0 \leq t \leq 1$, we can get any point on the segment.

Note that this a vector equation, which really corresponds to two equations - one for the x-component, and the other for the y-component.

To figure out the intersection of two lines AB and CD, write down the parametric equations of both. We'll use the variables $s$ and $t$ for the parameters of AB and CD respectively. Let $P(s)$ be the parametric point on AB, and $Q(t)$ be the parametric point on CD. If the two lines intersect, then $P(s) = Q(t)$. This gives us:

$$(1-s)\overline{A} + s\overline{B} = (1-t)\overline{C} + t\overline{D}$$

This is equivalent to the two scalar equations:

$$(1-s)x_a + sx_b = (1-t)x_c + tx_d$$

$$(1-s)y_a + sy_b = (1-t)y_c + ty_d$$

Which simplifies to

$$s(x_b - x_a) - t(x_d - x_a) = x_c - x_a$$

$$s(y_b - y_a) - t(y_d - y_a) = y_c - y_a$$

Plug in the coordinates, and solve the system for $s$ and $t$. A simple way to do this is with Cramer's rule.

Once you're done, you'll have values for the parameters. If you're dealing with lines, there are no constraints on the values, so just plug them into the original parametric equation to get the x and y coordinates of the intersection point. If you're intersecting segments, you have to make sure that the intersection point lies on both segments. If both $s$ and $t$ are in $[0, 1]$, then we're safe. Otherwise the two segments don't intersect, even if the lines do.

## 7.2 Cramer's Rule

Given the system of equations:

$$A_1x + B_1y = C_1$$

$$A_2x + B_2y = C_2$$

The solution is given by:

$$x = \frac{C_1B_2 - C_2B_1}{A_1B_2 - A_2B_1}$$

$$y = \frac{A_1C_2 - A_2C_1}{A_1B_2 - A_2B_1}$$

However, if the denominator is zero, then we have a problem. Either the two lines are parallel, in which case there is no intersection, or they are the same line - in which case they intersect at an infinite number of points. In the latter case, the numerators will be zero as well.