Computational Geometry

Prerequisites

- Graph Theory
- Shortest Path

Tools

This module discusses several algorithms that calculate various geometric properties, mostly based on only two operations described below: cross product and arctangent.

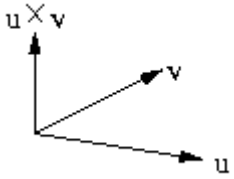
Cross Product

The cross product of u and v is written as $u \times v$. Computationally, the *cross product* of two three-dimensional vectors u and v is the vector determinant of the following matrix (where **i**, **j**, and **k** are unit vectors in the x, y, and z directions respectively):

```
| i  j  k  |
| ux uy uz |
| vx vy vz |
```

That equation works out to:

$(u_y v_z - v_y u_z)\mathbf{i} + (u_z v_x - u_x v_z)\mathbf{j} + (u_x v_y - u_y v_x)\mathbf{k}$

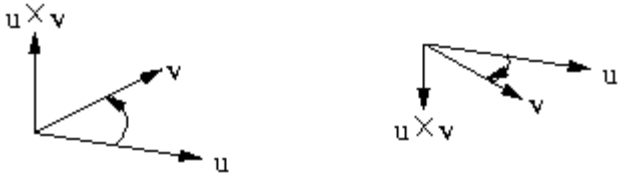


This definition can be used for vectors in two dimensions by using three-dimensional vectors with a z component of 0. The resulting vector will only have a z value.

The cross product has three properties:

- The *cross product* of two vectors is perpendicular to both vectors.
- The length of the cross product is equal to the product of:
  - the length of u,
  - the length of v, and
  - the sine of the angle between the vectors.

Of the two different directions that are perpendicular to both u and v, the direction the cross product points depends on whether u is ``to the right'' of $v$ or ``to the left.''



Dot product

The *dot product* of two vectors *u* and *v* is a scalar written as $u \cdot v$. Computationally, it is defined in three dimensions as: $u_x v_x + u_y v_y + u_z v_z$

The dot product is actually equal to the product of:

- the length of *u*
- the length of *v*
- the cosine of the angle between *u* and *v*.

Presuming *u* and *v* are non-zero, if the dot product if negative, *u* and *v* make an angle greater than 90 degrees. If it is zero, then *u* and *v* are perpendicular. If *u cdot v* is positive, then the two vectors form an acute angle.

Arctangent

The *arctangent*function calculates the (an) angle whose tangent is its argument and generally returns a real number between -pi/2 and pi/2. An additional function in C, *atan2*, takes two arguments: a *DELTA y* value and a *DELTA x* value (in that order!). It determines the angle between the given vector and the positive *x* axis and returns a value between -pi and pi. This has the advantage of removing concerns about dividing by zero or writing code to repair angles in order to handle the negative *x* cases. The *atan2* function is almost always easier to use than the simpler *atan* function that takes only one argument.

Particular Debugging Problems

The main problem with geometric problems is that they spawn **a lot** of special cases. Be on the lookout for these special cases and **make sure your program works for all of them**.

Floating point calculations also create a new set of problems. Floating point calculations are rarely precise, as the computer only maintains so many bits (digits) of accuracy: be aware of this. In particular, when checking if two values are equal, check to see if they are within some small tolerance of each other not precisely equal.

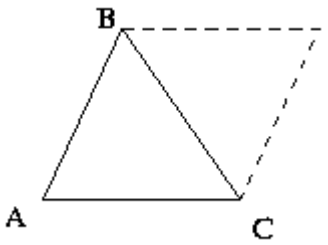
Geometric Algorithms

Here are some of snippets that can help you solve geometry problems.

Area of Triangle

To calculate the area of a triangle with vertices (a, b, c), pick a vertex (say a) and create a vector to the other two vertices (let *u* = b - a, and *v* = c - a). The area of the triangle

(a, b, c) is one half the length of cross product *u x v*.



An alternative method to find the area of triangle is to use Heron's (also known as Hero's) formula. If the lengths of the sides of a triangle are *a*, *b*, and *c*, let *s* = *(a+b+c)/2*. The area of the triangle is then

$$sqrt(s* (s-a)*(s-b)*(s-c)) .$$

Are Two Line Segments Parallel?

To check if two line segments are parallel, create vectors along each line segment and check to see if their cross product is (almost) zero.

Area of polygon

The area of a polygon with vertices (x_1, y_1), ..., (x_n, y_n) is equal to the determinant:

$$\frac{1}{2}\begin{vmatrix} x1 & x2 & \dots & xn \\ & & & \\ y1 & y2 & \dots & yn \end{vmatrix}$$

where the determinate is defined to be similar to the 2 by 2 determinant: $x_1 y_2 + x_2 y_3 + \dots + x_n y_1 - y_1 x_2 - y_2 x_3 - \dots - y_n x_1$

Distance from a point to a line

The distance from a point P to a line AB is given by the magnitude of the cross product. In particular, *d*(P,AB) = |(P - A) x (B - A)| / | B - A| .

To determine the distance from a point P to the plane defined by A, B, and C, let *n* = (B - A) x (C - A). The distance is then give by the following equation: *d*(P,ABC) = (P-A) · n / |n|.

Points on a line

A point is on a line if the distance from the point to the line is 0.

Points on the same side of line

This notion only makes sense for two dimensions. To check if points C and D are on the same side of line AB, calculate the z component of *(B - A) x (C - A)* and *(B - A) x (D - A)*. If the *z* components have the same sign (i.e., their product is positive), then C and D are on the same side of the line AB.

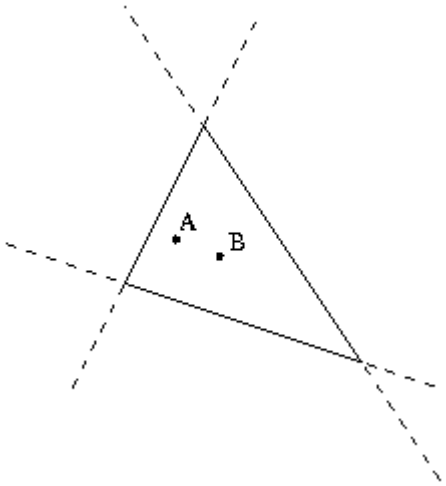
Point on line segment

To calculate if a point C is on the line segment AB, check if C is on the line AB. If it is,

then check if the length of AB is equal to the sum of the lengths of AC and CB.

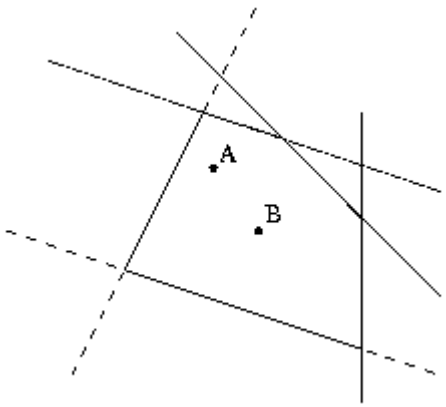
Point in triangle

To check if a point A is in a triangle, find another point B which is within the triangle (the average of the three vertices works well). Then, check if the point A is on the same side of the three lines defined by the edges of the triangle as B.



Point in convex polygon

The same trick works for a convex polygon:



Four (or more) points are coplanar

To determine if a collection of points is coplanar, select three points, A, B, and C. Now, if, for any other point D, $(B - A) \times (C - A)) \cdot (D - A) = \sim 0$, then the collection of points resides in some plane.

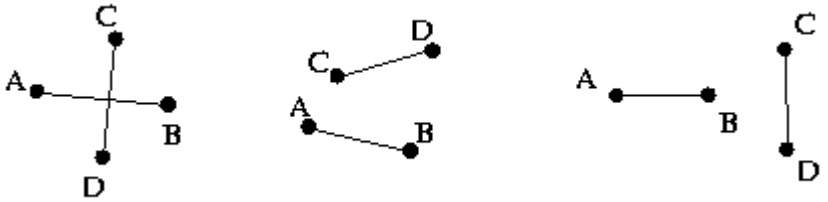
Two lines intersect

Two lines intersect if and only if they are not parallel in two dimensions.

In three dimensions, two lines AB and CD intersect if they are not parallel and A, B, C, and D are coplanar.

Two line segments intersect

In two dimensions, two line segments AB and CD intersect if and only if A and B are on opposite sides of the line CD and C and D are on opposite sides of line AB.



Note that both of the checks are necessary, as for the last case one of the checks returns true, while the other testifies to the fact that AB and CD do not intersect. In three dimensions, solve following system of equations, where *i* and *j* are the unknowns:

$$A_x + (B_x - A_x)\, i = C_x + (D_x - C_x)\, j$$
$$A_y + (B_y - A_y)\, i = C_y + (D_y - C_y)\, j$$
$$A_z + (B_z - A_z)\, i = C_z + (D_z - C_z)\, j$$

If this system has a solution (*i*, *j*), where $0 <= i <= 1$ and $0 <= j <= 1$, then the line segments intersect at: $(A_x + (B_x - A_x)i,\ A_y + (B_y - A_y)i,\ A_z + (B_z - A_z)\, i$.

Point of Intersection of Two Lines

For the lines AB and CD in two dimensions, the most straight-forward way to calculate the intersection of them is to solve the system of two equations and two unknowns:

$$A_x + (B_x - A_x)i = C_x + (D_x - C_x)\, j$$
$$A_y + (B_y - A_y)i = C_y + (D_y - C_y)\, j$$

The point of intersection is:

$$(A_x + (B_x - A_x)\, i,\ A_y + (B_y - A_y)\, i)$$

In three dimensions, solve the same system of equations as was used to check line intersection, and the point of intersection is:

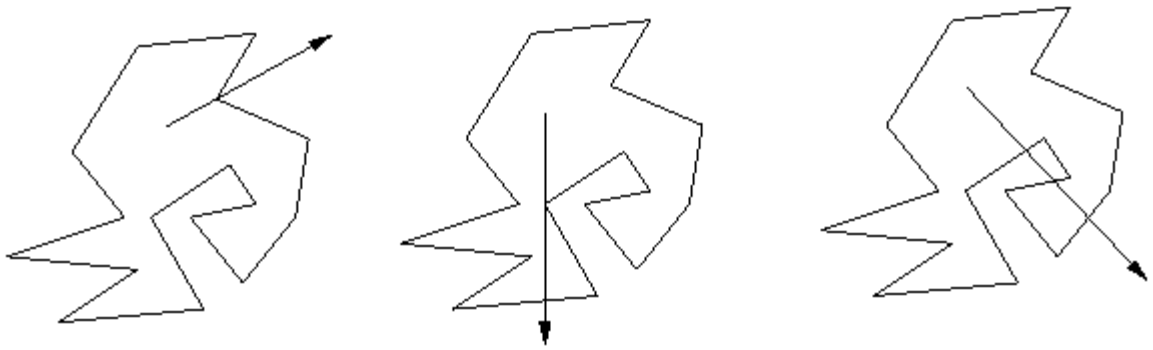
$$(A_x + (B_x - A_x)i,\ A_y + (B_y - A_y)i,\ A_z + (B_z - A_z)i)$$

Checking convexity of 2-dimensional polygon

To check the convexity of a 2-dimensional polygon, walk the polygon in clock-wise order. For each triplet of consecutive points (A, B, C), calculate the cross product (B - A) x (C - A). If the *z* component of each of these vectors is positive, the polygon is convex.

Point in non-convex polygon

To calculate if a point is within a nonconvex polygon, make a ray from that point in a random direction and count the number of times it intersects the polygon. If the ray intersects the polygon at a vertex or along an edge, pick a new direction. Otherwise, the point is within the polygon if and only if the ray intersects the polygon an odd number of times.



This method also extends to three dimensions (and higher), but the restriction on intersection is that it only intersects at faces and not at either a vertex or an edge.

Geometry Methodologies

Geometric problems introduce several different tricks that can be used to either reduce the run-time or approximate the solution.

Monte Carlo

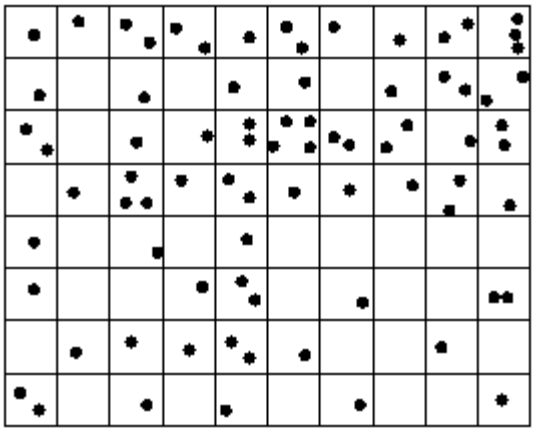
The first geometric trick is based on randomness. Instead of calculating the probability that something occurs, simulate a random event and calculate the fraction of times it occurs. If enough events are simulated, the difference between these two values becomes very small.

This can be helpful to determine something like the area of a figure. Instead of calculating the area directly, determine a bounding box, and throw ``darts'' at the box, and estimate what the probability of hitting the figure is. If this is calculated accurately enough, this can give a good estimate of the actual area.

The problem with this method is to get a good relative error (error divided by the actual value) requires a large number of successful events. If the probability of the event occurring is very small, the method does not yield good results.

Partitioning

Partitioning is a method to improve the speed of a geometric algorithm. This entails dividing the plane up into sections (usually by a grid but sometimes into radial sections or some other method), and bucketing the objects into appropriate section(s). When looking for objects within some figure, only those sections which have a non-zero intersection with that figure need to be examined, thereby greatly reducing the cost of the algorithm. This is helpful to determine the set of objects within some distance of a given point (the figure is a circle) or to check for intersections (the figure is a line).



Graph Problems

Sometimes what may look like a geometric problem is really a graph problem. Just because the input is points in the plane does not mean it's a geometric algorithm.

Example Problems

Point Moving

Given a set of line segments in the plane, and two points A and B, is it possible to move from A to B without crossing any of the segments?

The line segments partition the plane into regions. Determine these regions, and see if A and B reside in the same region.

Bicycle Routing

Given a collection of non-intersecting buildings along with start and end locations, find the shortest path from A to B that doesn't go through any buildings.

Analysis: This is really a graph problem. The nodes are the start and end locations, along with the vertices of the buildings. There are edges between any two nodes such that the line segment between them does not intersect any buildings, with weight equal to the length of the length of the line segments. Once that graph has been calculated, the problem is shortest path.

Maximizing Line Intersections

Given a collection of segments in the plane, find the greatest number of segments which can by intersected by drawing a single line.

Analysis: With a little bit of thought, it is clear that the line segment must pass through two of the vertices of the collection of line segments. Thus, try all pairs of vertices, and calculate the crossing for each. Combining this with partitioning gives an algorithm that runs fairly quickly.

Polygon Classification

Given a collection of segments defining a polygon, determine if it is simple (no two non-

consecutive line segments intersect) and convex.

Back to USACO Training Gateway | Comment or Question