

Geometry Concepts: Using Geometry in Topcoder Problems

By **lbackstrom** — *topcoder member*

[...read Section 2](#)

[PointInPolygon](#)

[TVTower](#)

[Satellites](#)

[Further Problems](#)

[PointInPolygon \(SRM 187\)](#)

Requires: [Line-Line Intersection](#), [Line-Point Distance](#)

First off, we can use our [Line-Point Distance](#) code to test for the "BOUNDARY" case. If the distance from any segment to the test point is 0, then return "BOUNDARY". If you didn't have that code pre-written, however, it would probably be easier to just check and see if the test point is between the minimum and maximum x and y values of the segment. Since all of the segments are vertical or horizontal, this is sufficient, and the more general code is not necessary.

Next we have to check if a point is in the interior or the exterior. Imagine picking a point in the interior and then drawing a ray from that point out to infinity in some direction. Each time the ray crossed the boundary of the polygon, it would cross from the interior to the exterior, or vice versa. Therefore, the test point is on the interior if, and only if, the ray crosses the boundary an odd number of times. In practice, we do not have to draw a ray all the way to infinity. Instead, we can just use a very long line segment from the test point to a point that is sufficiently far away. If you pick the far away point poorly, you will end up having to deal with cases where the long segment touches the boundary of the polygon where two edges meet, or runs parallel to an edge of a polygon — both of which are tricky cases to deal with. The quick and dirty way around this is to pick two large random numbers for the endpoint of the segment. While this might not be the most elegant solution to the problem, it works very well in practice. The chance of this segment intersecting anything but the interior of an edge are so small that you are almost guaranteed to get the right answer. If you are really concerned, you could pick a few different random points, and take the most common answer.

```
String testPoint(verts, x, y){
    int N = lengthof(verts);
    int cnt = 0;
    double x2 = random()*1000+1000;
    double y2 = random()*1000+1000;
    for(int i = 0; i<N; i++){
        if(distPointToSegment(verts[i],verts[(i+1)%N],x,y) == 0)
            return "BOUNDARY";
        if(segmentsIntersect((verts[i],verts[(i+1)%N]),{x,y},{x2,y2}))
            cnt++;
    }
    if(cnt%2 == 0) return "EXTERIOR";
    else return "INTERIOR";
}
```

TVTower(SRM 183)

Requires: [Finding a Circle From 3 Points](#)

In problems like this, the first thing to figure out is what sort of solutions might work. In this case, we want to know what sort of circles we should consider. If a circle only has two points on it, then, in most cases, we can make a slightly smaller circle, that still has those two points on it. The only exception to this is when the two points are exactly opposite each other on the circle. Three points, on the other hand, uniquely define a circle, so if there are three points on the edge of a circle, we cannot make it slightly smaller, and still have all three of them on the circle. Therefore, we want to consider two different types of circles: those with two points exactly opposite each other, and those with three points on the circle. Finding the center of the first type of circle is trivial — it is simply halfway between the two points. For the other case, we can use the method for [Finding a Circle From 3 Points](#). Once we find the center of a potential circle, it is then trivial to find the minimum radius.

```
int[] x, y;
int N;
double best = 1e9;
void check(double cx, double cy){
    double max = 0;
    for(int i = 0; i < N; i++){
        max = max(max, dist(cx, cy, x[i], y[i]));
    }
    best = min(best, max);
}
double minRadius(int[] x, int[] y){
    this.x = x;
    this.y = y;
    N = lengthof(x);
    if(N==1) return 0;
    for(int i = 0; i < N; i++){
        for(int j = i+1; j < N; j++){
            double cx = (x[i]+x[j])/2.0;
            double cy = (y[i]+y[j])/2.0;
            check(cx, cy);
            for(int k = j+1; k < N; k++){
                //center gives the center of the circle with
                //(x[i],y[i]), (x[j],y[j]), and (x[k],y[k]) on
                //the edge of the circle.
                double[] c = center(i, j, k);
                check(c[0], c[1]);
            }
        }
    }
    return best;
}
```

Satellites (SRM 180)

Requires: [Line-Point Distance](#)

This problem actually requires an extension of the Line-Point Distance method discussed previously. It is the same basic principle, but the formula for the **cross product** is a bit different in three dimensions.

The first step here is to convert from spherical coordinates into (x,y,z) triples, where the center of the earth is at the origin.

```
double x = sin(lng/180*PI)*cos(lat/180*PI)*alt;
double y = cos(lng/180*PI)*cos(lat/180*PI)*alt;
double z = sin(lat/180*PI)*alt;
```

Now, we want to take the cross product of two 3-D vectors. As I mentioned earlier, the cross product of two vectors is actually a vector, and this comes into play when working in three dimensions. Given vectors (x_1, y_1, z_1) and (x_2, y_2, z_2) the cross product is defined as the vector (i, j, k) where

```
i = y1z2 - y2z1;
j = x2z1 - x1z2;
k = x1y2 - x2y1;
```

Notice that if $z_1 = z_2 = 0$, then i and j are 0, and k is equal to the cross product we used earlier. In three dimensions, the cross product is still related to the area of the parallelogram with two sides from the two vectors. In this case, the area of the parallelogram is the norm of the vector: $\text{sqrt}(i*i + j*j + k*k)$.

Hence, as before, we can determine the distance from a point (the center of the earth) to a line (the line from a satellite to a rocket). However, the closest point on the line segment between a satellite and a rocket may be one of the end points of the segment, not the closest point on the line. As before, we can use the dot product to check this. However, there is another way which is somewhat simpler to code. Say that you have two vectors originating at the origin, S and R , going to the satellite and the rocket, and that $|X|$ represents the norm of a vector X .

Then, the closest point to the origin is R if $|R|^2 + |R-S|^2 \leq |S|^2$ and it is S if $|S|^2 + |R-S|^2 \leq |R|^2$

Naturally, this trick works in two dimensions also.

Further Problems

Once you think you've got a handle on the three problems above, you can give these ones a shot. You should be able to solve all of them with the methods I've outlined, and a little bit of cleverness. I've arranged them in what I believe to be ascending order of difficulty.

[ConvexPolygon \(SRM 166\)](#)

Requires: [Polygon Area](#)

[Surveyor \(TCCC '04 Qual 1\)](#)

Requires: [Polygon Area](#)

[Travel \(TCI '02\)](#)

Requires: [Dot Product](#)

[Parachuter \(TCI '01 Round 3\)](#)

Requires: [Point In Polygon](#), [Line-Line Intersection](#)

[PuckShot \(SRM 186\)](#)

Requires: [Point In Polygon](#), [Line-Line Intersection](#)

[ElectronicScarecrows \(SRM 173\)](#)

Requires: [Convex Hull](#), [Dynamic Programming](#)

[Mirrors \(TCI '02 Finals\)](#)

Requires: [Reflection](#), [Line-Line Intersection](#)

[Symmetry \(TCI '02 Round 4\)](#)

Requires: [Reflection](#), [Line-Line Intersection](#)

[Warehouse \(SRM 177\)](#)

Requires: [Line-Point Distance](#), [Line-Line Intersection](#)

The following problems all require geometry, and the topics discussed in this article will be useful. However, they all require some additional skills. If you got stuck on them, the editorials are a good place to look for a bit of help. If you are still stuck, there has yet to be a problem related question on the [round tables](#) that went unanswered.

[DogWoods \(SRM 201\)](#)

[ShortCut \(SRM 215\)](#)

[SquarePoints \(SRM 192\)](#)

[Tether \(TCCC '03 W/MW Regional\)](#)

[TurfRoller \(SRM 203\)](#)

[Watchtower \(SRM 176\)](#)

More Resources

[Member Tutorials](#)

Read more than 40 data science tutorials written by topcoder members.

[Problem Set Analysis](#)

Read editorials explaining the problem and solution for each Single Round Match (SRM).

[Data Science Guide](#)

New to topcoder's data science track? Read this guide for an overview on how to get started in the arena and how competitions work.

[Help Center](#)

Need specifics about the process or the rules? Everything you need to know about competing at topcoder can be found in the Help Center.

[Member Forums](#)

Join your peers in our member forums and ask questions from the real experts - topcoder members!

Get Connected

Your email address

[Submit](#)

© 2014 topcoder. All Rights Reserved.

[Privacy Policy](#) | [Terms](#)

