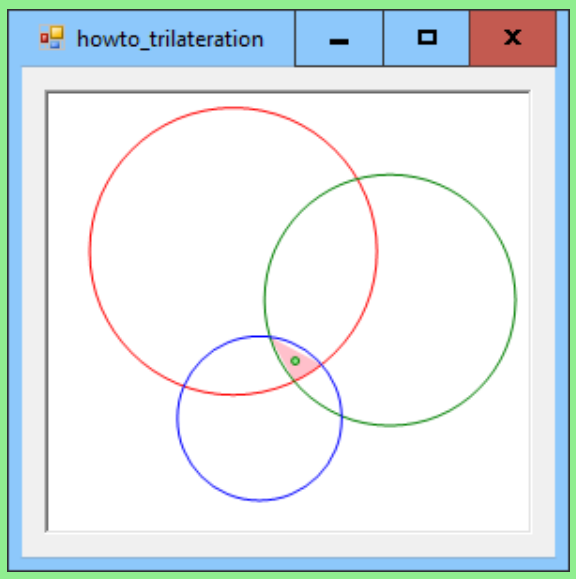
Title: Perform trilateration in C#

Download

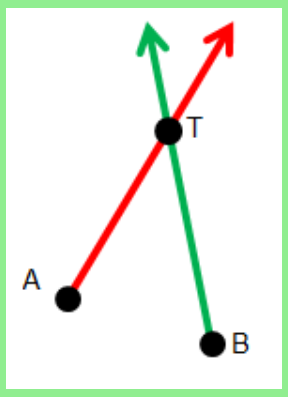


This post explains how a C# program can use trilateration to locate a point that is known distances from three other points.

Trilateration is somewhat similar to triangulation, so the following two sections begin by explaining what triangulation and trilateration are.

Triangulation

In triangulation, you know the directions from two or more points toward an unknown location. You draw rays leaving the known points in their directions, and the place where those rays intersect gives you the unknown location.



For example, suppose you have a directional antenna. You go to point A in the picture on the right, and it tells you that the transmitter (point T) is in the direction of the red arrow. You then move to another point, this time B, and the antenna tells you that the transmitter is in the direction of the green arrow. The place where the red and green arrows intersect is the (approximate) location of the transmitter.

You can take more than two readings if you like. If you do, the new arrows may not all intersect at exactly the same point, but they should give you an area where the transmitter is likely to be.

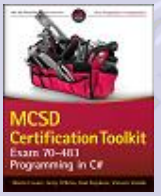
This technique is called triangulation because you're using a triangle (defined by the points A, B, and C) to location one of the points.

Trilateration

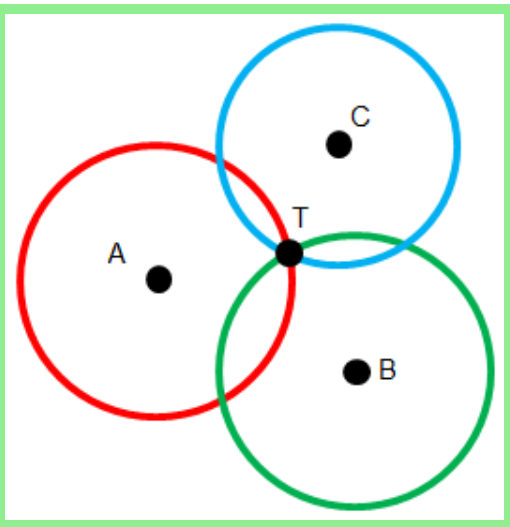
In trilateration you use distances instead of angles to find an unknown location.

For example suppose you know that a target point T is some distance Ra away from point A. In that case you know that T lies on a circle with radius Ra centered at point A.

If you also know that the target is distance Rb away from point B, then it also lies on a circle with radius Rb centered at point B. Ideally those circles should intersect at one or two places. If they intersect at a single point, then the target



should be at that point. If they intersect at two points, then the target lies at one of those points and you need a third measurement to figure out which.



If you have a third point and distance, then it defines a third circle. Ideally the three circles intersect at a single point, and that is where the target lies.

Unfortunately, the circles may not intersect at a single point. In that case they define a triangular area (where the sides of the triangle are actually arcs of the circles) and the target should be somewhere in that area.

This technique is called trilateration because it involves three lengths, in this example the segments AT, BT, and CT.

The Algorithm

To find the area where the target should be, find the points where each pair of circles intersect. Then for each pair of circles, find the point of intersection that is closest to the center of the third circle. That point of intersection is part of the triangle that defines the target area.

If you want a specific point near the target, you can average the triangles' vertex coordinates to find the triangle's centroid.

The Code

The following `Trilaterate` method finds the triangle defined by three intersecting circles.

```
// Trilaterate.
// Throw an exception if there is a problem.
private PointF[] Trilaterate(RectangleF circle1,
    RectangleF circle2, RectangleF circle3)
{
    // Convert the circles from bounding rectangles
    // to centers and radii.
    float cx1 = circle1.X + circle1.Width / 2f;
    float cy1 = circle1.Y + circle1.Height / 2f;
    float cx2 = circle2.X + circle2.Width / 2f;
    float cy2 = circle2.Y + circle2.Height / 2f;
    float cx3 = circle3.X + circle3.Width / 2f;
    float cy3 = circle3.Y + circle3.Height / 2f;
    float r1 = circle1.Width / 2f;
    float r2 = circle2.Width / 2f;
    float r3 = circle3.Width / 2f;

    // Find the points of intersection.
    PointF
        intersection12a, intersection12b,
        intersection23a, intersection23b,
        intersection31a, intersection31b;
    if (FindCircleCircleIntersections(
            cx1, cy1, r1, cx2, cy2, r2,
            out intersection12a, out intersection12b) == 0)
        throw new Exception("circle1 and circle2 do not intersect.");
```

```
    if (FindCircleCircleIntersections(
            cx2, cy2, r2, cx3, cy3, r3,
            out intersection23a, out intersection23b) == 0)
        throw new Exception("circle2 and circle3 do not intersect.");
    if (FindCircleCircleIntersections(
            cx3, cy3, r3, cx1, cy1, r1,
            out intersection31a, out intersection31b) == 0)
        throw new Exception("circle3 and circle1 do not intersect.");

    // Find the points that make up the target area.
    PointF[] triangle = new PointF[3];
    PointF center1 = new PointF(cx1, cy1);
    PointF center2 = new PointF(cx2, cy2);
    PointF center3 = new PointF(cx3, cy3);
    if (Distance(intersection12a, center3) <
            Distance(intersection12b, center3))
        triangle[0] = intersection12a;
    else
        triangle[0] = intersection12b;
    if (Distance(intersection23a, center1) <
            Distance(intersection23b, center1))
        triangle[1] = intersection23a;
    else
        triangle[1] = intersection23b;
    if (Distance(intersection31a, center2) <
            Distance(intersection31b, center2))
        triangle[2] = intersection31a;
    else
        triangle[2] = intersection31b;

    return triangle;
}
```

The method takes as parameters three `RectangleF` structures that define the three triangles. The code first finds the centers and radii of those circles.

It then calls the `FindCircleCircleIntersections` method to see where the circles intersect each other. (You can learn more about that method in my earlier post Determine where two circles intersect in C#.)

For each pair of circles, the code determines which of the pair's points of intersection is closest to the center of the third circle and adds that point to the triangles list.

The following code shows how the program calculates the triangle's centroid.

```
// Return the triangle's centroid.
private PointF FindTriangleCentroid(PointF p1, PointF p2, PointF p3)
{
    return new PointF(
        (p1.X + p2.X + p3.X) / 3f,
        (p1.Y + p2.Y + p3.Y) / 3f);
}
```

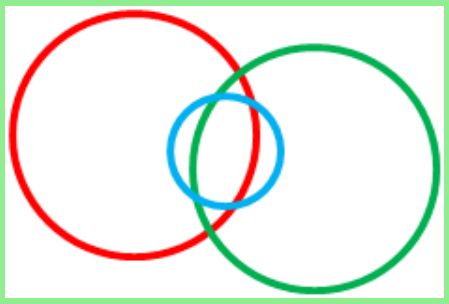
This method simply averages the coordinates of the triangle's vertices.

Conclusion

This method works in a "normal" situation, but there are several odd situations where it won't work. For example, if the three circles do not all intersect each other, then you can't find three points of intersection to define the target area and the method does not produce a result. In that case you may be able to gradually

increase the circles' radii until they do intersect and still get a meaningful target area.

This algorithm also doesn't produce a good result if the triangles' centers lie roughly along a line, as shown in the picture on the right. In that case, the intersections define two areas where the target point may lie.



Download the example to experiment with it and to see additional details.