Aniruth Narayanan

Thought Process: Triangulation

This project was inspired by the TV show Arrow.

In Season 3, Episode 7, around 23:13, the episode's main antagonist (Carrie Cutter) threatened a hacker (I presume) for Arrow's hideout. He mentions that he used the average times it took the Arrow to get to certain places to triangulate his position within a block's range.

The idea, while it was (probably) fake in the show, intrigued me. Intuitively, it seemed to hold merit, so I decided to write an algorithm to tackle it.

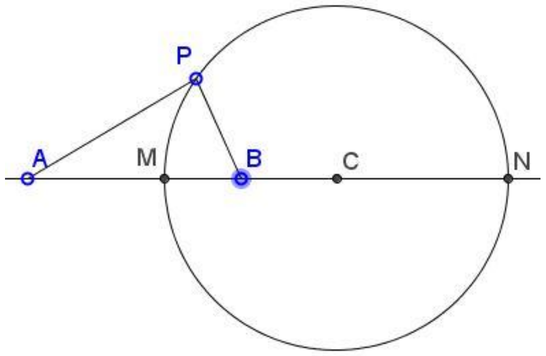
I began by thinking about the math that I already knew. The following explanation of the math might be complex but assumes some concepts are known (such as midpoint formula, formula for a circle, etc.). If I assume that the relative speed of the entity was the same and that there were no obstacles blocking any path, then relative time is what mattered through ratios (Since distance = rate * time, if the rate is constant, distance fluctuates with some time - except it is relative, not absolute).

Assuming all points are equidistant from the central location, the process of finding the point is relatively simple: this is the definition of the circumcenter, where the radius of the circle is the equal distance to every point that lies on a circle (that link has an interactive triangle to experiment with changing a triangle and observing how the circumcenter changes as well. This is also why only three points are needed to define any one circle.

The problem arises when the concept of relative time and ratios becomes involved. While perpendicular bisectors could be used for the circumcenter, they cannot be used for ratios as the locus of all points curves towards the shorter distance (due to the Pythagorean Theorem).

I tried a few different solutions to tackle this. I began with a solution of iterating through circles of varying size that were relative, but it would have been highly inefficient as every point along the circle would need to be checked. The biggest problem for me was that I couldn't tell the computer intuitively where to guess. As a human, I was able to guess in my sketches roughly where the point would be that satisfies the distance conditions but found it difficult to articulate why, since it was my human intuition. I wasn't trying to create a complex machine learning solution, so I perused the internet. I knew it was an arc, but I didn't know if it was a circle, a parabola, or some higher degree function.

After doing a bit of searching, I found Apollonian Circles. (The picture below is from this website which also has a proof behind the theorem if you're interested).



This image labels the points M and N that need to be found with the common ratio of distances regarding points A and B. This is from the above site.

The article goes through how to find the locus of points (the circle) whose distances to two other points (A and B) remain in a certain ratio and define it - by recognizing there are two points (M and N) on the line between the original two points (A and B) that satisfy the ratio that form the diameter of the circle. The midpoint of those two was the center (C) of the circle as well.

With the theory (for the most part) out of the way, I began to start coding. I believe most of the code is self-explanatory: the first two functions help to define the circle equation by solving for y by looking at the top part and bottom part of the circle based on sign and finding the length between two points, the inputs gather the information, and the algorithm iterates.

More specifically, the ratios determine the relative time (and thus relative distance) while the increments and margins are left to the user to customize based on their operating machine.

Then the math begins. By adjusting the midpoint formula and solving for the x and y of the inner and outer points, I was able to find the two points that formed the diameter of the circle. From there, I found the center by taking the midpoint of those two and the radius out to the points from the center. Finally, the program begins iterating.

One problem I initially ran into was the issue of using the for loop with a range parameter - but since that only accepts integer values, I used a while loop and an increment every passing of the loop. The loop works as follows: It first uses the x value to find the two corresponding y values (this is a circle, after all) and then the distance between each of the two points to the third point and the first point to check if the ratio upholds. This would not be exact, which is where the margin comes in, which then reports all triangulated points within that interval on that margin for the user to see and use. I thought about breaking it, but decided a full range was better than the very first point.

The last part of the program is the works condition. There are some circles and times for which a point/location would not exist to satisfy those conditions - if the works statement is never changed to true because no triangulated points were printed, the program informs the user that the scenario given is impossible. This was meant to prevent any errors when I was testing it. The points I used to test were (1,0), (5,0), and (2,5), and I was able to get the margin and increment significantly down which still operated very quickly.

Areas for Improvement

There are a couple of areas to improve.

1. I didn't account for the ratio being 1, simply because I felt the code would be rather redudant and roughly double the length of the code. However, in my tests this was not a significant problem, so I left it be.
2. I think that the process I used of iteration could probably be improved. While it is true that the process of triangulation would only provide a regional area instead of an absolute value, I'm fairly certain the computer should be able to figure out the margin of error using uncertainties.

3. This algorithm can definitely be optimized further by simplifying the math and using a few other heuristics beyond the Apollonian Circles - although, for the time being, the program seemed to run fine so I left it be.

For now, I'll leave these to another time.