Obstacle-avoidant Function Fitting

Artur Roos

October 2022

Contents

Document compiled December 12, 2022, made with LaTeX.

1 Introduction

1.1 Rationale

Not longer than two days ago I was challenged to a mathematical standoff by a good friend of mine, the weapons would be polynomial functions and their graphs. Winning this competition was a matter of honor, so I used all of my skill and wits to prepare. The fight would be held in "Graph Wars", a small competitive 1v1 game published on the 23rd of February 2022.

The goal of the game is to design a function that can be plotted from one point to another without intersecting any circles in shortest time. You are given 60 seconds to input as many functions as you want, then the turn is passed onto your opponent. This is wrapped into a war-like setting where your function is a projectile, directed by your soldiers against your enemy's.

It is also usually played without assistance of any external programs like plotters or calculators, that's for good reason. Most importantly, game is considered solved: for any initial state of the field there can be deduced a function to guarantee a "hit".

1.2 Aim

My ultimate goal set out to be simply designing an algorithm, which when given a list of all circle obstacles, their radii, and the edge points would express the path to go from to another without colliding with any of the obstacles.

2 Investigation

The task consists of the small independents subtasks: finding the points and fitting the function. Task-specific restrictions are mentioned in the corresponding section.

2.1 Finding The Points

This problem can be boiled down to pathfinding. However, usual pathfinding techniques operate on discrete spaces like graphs and grids which our plane is not, we need to convert it to one. Cells of the grid can be in two states: occupied or empty. The occupied cells are the ones covered by an obstacle. Since obstacle circles only occur at integer coordinates and the smallest radius of a circle is 1, we can safely assume that the smallest size of a grid cell we might need is 1 respectively. After that we can apply the **Theta***. It is preferable to Dijkstra's, A* and others since it is designed to fit any-angle pathfinding. It can find near-optimal paths with runtimes comparable to A*. Even though our points are aligned on a grid, the "projectile" is allowed to move in all directions, not just the cardinal ones. Only additional piece of information it requires is a line of sight function (L). Let $I(A, B)$ be some function, equal to the amount of intersections of a line between points A and B with all the circles of a set C containing $\langle x, y, r\rangle$. So $I_n(A, B)$ is the amount of intersections of a line from A to B with some circle C_n. Using that we can define $I(A, B) = \sum_{n=1}^{|C|} I_n(A, B)$.

For simplicity let's assume the circle is at $\langle 0, 0\rangle$ with some radius r. But first let's find the point $p_0 = (x_0, y_0)$ on the line closest to the circle. We can use an alternative line equation $ax + by + c = 0$ for which we can for sure say that vector $\vec{v''} = (a, b)$ will be perpendicular to the line.

Coordiantes of p_0 should be proportional to the vector $\vec{v''}$. Now we may just normalize the vector to the length.

First we normalize by dividing the vector by it's length.

$$\vec{v'}_{normalized} = \left(\frac{a}{\sqrt{a^2 + b^2}}, \frac{b}{\sqrt{a^2 + b^2}} \right) \tag{1}$$

Then we scale the vector by the distance of the line to the origin.

$$\vec{v'} = \vec{v'}_{normalized} * \frac{c}{\sqrt{a^2 + b^2}} \tag{2}$$

By simplifying the denominators we get

$$\vec{v} = \left(a * \frac{c}{a^2 + b^2}, b * \frac{c}{a^2 + b^2} \right) \tag{3}$$

This vector is inverted, so by multiplying everything by -1 we get the actual coordinates of the point. Since the vector and the point lie in the same coordinate system we can claim their equivalence, therefore

$$p_0 = -\vec{v} = (-a * \frac{c}{a^2 + b^2}, -b * \frac{c}{a^2 + b^2}) = (x_0, y_0) \tag{4}$$

So p_0 is the closest point to the circle. Now, based on this we can calculate the amount of intersections: if p_0 is inside the circle there are 2 intersections, it it lies on the radius there is 1 intersection and there are none otherwise.

$$\begin{cases} I_n(A, B) = 0, |\vec{v}| > C_{n_r} \\ I_n(A, B) = 1, |\vec{v}| = C_{n_r} \\ I_n(A, B) = 2, |\vec{v}| < C_{n_r} \end{cases} \tag{5}$$

When evaluating the function I_n for simplicity we just subtract the coordinates (C_{n_x}, C_{n_y}) from all the other coordinates. This shifts the entire plane so that the circle is at the origin, the precise usecase the solution above covers.

2.2 Fitting The Function

Let A be a set of cardinality n where each point is like $\langle x, y \rangle$. For simplicity we can impose a strict total ordering on the set, and reindex it in such way that $\forall a_1, a_2 \in A : a_1 < a_2 \Leftrightarrow a_{1_x} < a_{2_x}$. In this way the A_n is guaranteed to be the rightmost point, which will make the calculations a lot easier. Let's also move the coordinate system so that the point at bottom left $\langle -50, -20 \rangle$ is at the origin. That will make all the coordinates positive

and they are easier to reason about. We may also ignore all the x values in ranges $(-\infty; a_{1_x})$ and $(a_{n_x}; \infty)$, since the function will not be evaluated there at any point.

There is an infinite number of ways to make a function that goes through a set of points, the simplest one is the Lagrange's Interpolation which does fit this usecase perfectly. We may follow a specific example and try to generalise it later. For example $A = \{\langle 0, 1\rangle, \langle 6, 9\rangle, \langle 17, 4\rangle, \langle 19, 22\rangle\}$.

First we may try to make a function $\phi_i(x)$ such that it equates to 0 at every point, but x_i it should be. For simplicity we may make it equate to 1, this will allow for better composability in the future. For example the process for the second point might look something like this.

First we use a helper function $\hat{\phi}_2(x)$ make it be zero at all the points, but x_i.

$$\hat{\phi}_2(x) = (x - 0)(x - 17)(x - 19) \tag{6}$$

Now the value at x_2 is non-zero, we can just devide it by $\hat{\phi}_2(x_2)$ so it is 1 at x_2.

$$\hat{\phi}_2(x_2) = (6 - 0)(6 - 17)(6 - 19) \tag{7}$$

$$\phi_2(x) = \frac{(x - 0)(x - 17)(x - 19)}{\hat{\phi}_2(x_2)} \tag{8}$$

$$\phi_2(x) = \frac{(x - 0)(x - 17)(x - 19)}{(6 - 0)(6 - 17)(6 - 19)} \tag{9}$$

x	x_1	x_2	x_3	x_4
$\phi_2(x)$	0	1	0	0

To make ϕ_i completely representative of the point at x_i we need to scale it by y_i. Let's define a new helper function $\psi_i(x)$.

$$\psi_2(x) = y_2\phi_2(x) \tag{10}$$

$$\psi_2(x) = \frac{(x - x_1)(x - x_3)(x - x_4)}{(x_2 - x_1)(x_2 - x_3)(x_2 - x_4)} * y_2 \tag{11}$$

Based on this we can express $\psi_i(x)$'s values for any x_i.

	$\psi_1(x)$	$\psi_2(x)$	$\psi_3(x)$	$\psi_4(x)$
x_1	y_1	0	0	0
x_2	0	y_2	0	0
x_3	0	0	y_3	0
x_4	0	0	0	y_4

Looking at the table of all the values of ψ_i we can deducde that their linear combination will satisfy our condition of $\psi_i(x_i) = y_i$ and $\psi_i(x_j) = 0, j \neq i$, yielding an $|A|$th order polynomial. So for the example set A the fit function is $f(x) = \psi_1(x) + \psi_2(x) + \psi_3(x) + \psi_4(x)$.

If we generalise this function to any set A it would look something like the following.

$$f(x) = \sum_{i=1}^{|A|} \psi_i(x)$$

$$\psi_i(x) = y_i \phi_i(x)$$

$$\phi_i = \prod_{j=1, j\neq i}^{|A|} \frac{x - x_j}{x_i - x_j} \tag{12}$$

$$f(x) = \sum_{i=1}^{|A|} \left(y_i * \prod_{j=1, j\neq i}^{|A|} \frac{x - x_j}{x_i - x_j} \right) \tag{13}$$

3 Limitations & Reflection

I used a geometric approach to finding intersections of lines with circles. The algebraic solution has a higher computational error due to imprecisions when using floating-point arithmetic. This doesn't impact the current use-case, however can be important when dealing with more fine point placement.

The described method of fitting covers a lot of scenarios, but it doesn't yield correct results when the next intended path point a_{i+1} on the path which goes after some a_i. In this case the pathfinding algorithm requires a path to turn back, which is impossible to do. This issue is most visible when reindexing: the order of points will be changed. Beyond that, this arises only in situations when the $f(x)$ **can't be a well-defined function**, since that requires it to have to two different values at same x. Luckily, those

scenarios never occured in a real game. Perhaps the game accounts for that and intentionally avoids it.

My method produces good results on the given dataset of densely packed points, however, the more spread apart the points are the greater the algorithm overshoots in spaces between two points. That can be a problem since it may actually accidentally go through one of the spheres. This effect can be mitigated by sampling more intermediate points, which in turn increases computational complexity. Since the game field is limited to span 100 units horizontally and 40 units vertically this is not a big problem. One of the alternative solutions would be using splines to form Bézier Curves, Hermite Curves Catmul-Rom or even B-Splines, but those are **barely touched on in the book** if at all.

4 Conclusion

TODO

5 Bibliography

TODO