Winning The "Graph Wars"

Artur Roos

December 16, 2022

Contents

Foreword

Document made using LaTeX and compiled December 16, 2022. Contains 2761 words. Presented December 16, 2022.

Introduction

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 "Graphwar", 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".

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.

The plane that the function is plotted on spans 40 units vertically and 100 units horizontally. The units and the obstacles can only be placed at integer coordinates.

You can sketch any function using the list of allowed operations:

$+$	$-$	$*$	$/$
pow	sin	cos	tan
ln	log	exp	abs

Investigation

The task consists of the small independents subtasks: finding the points and fitting the function. The found points are passed into the fitting algorithm, so it in general it is a 2-step pipeline. Some alternative approaches use more stages, as mentioned in the "Limitations & Reflection" section.

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 predicate P, which in turn requires a function to count the amount of objects that block the sight between points p_1 and p_2. Let $I(p_1, p_2)$ be some function, equal to the amount of intersections of a line between points p_1 and p_2 with all the circles of a set C containing $\langle x, y, r \rangle$. So $I_n(p_1, p_2)$ is the amount of intersections of a line from p_1 to p_2 with some circle C_n. Using that we can define $I(p_1, p_2) = \sum_{n=1}^{|C|} I_n(p_1, p_2)$.

For simplicity let's assume the circle is at $\langle 0, 0 \rangle$ with some radius r. So let there be some line $y = kx + b$, the coefficients for each are easy to find.

$$k = \frac{p_{2_y} - p_{1_y}}{p_{2_x} - p_{1_x}} \tag{1}$$

$$b = p_{1_y} - kp_{1_x} = p_{2_y} - kp_{2_x} \tag{2}$$

Distance between the closest point on this line and the origin (where the center of the circle is) is a little trickier to find. We can look at the right-angled triangle, formed by the line and its intercepts. We may use such points as the origin $(0, 0)$, x-intercept $(-\frac{b}{k}, 0)$ and y-intercept $(0, b)$. We are using positive $\frac{b}{k}$ because it's the length of the side, not just the coordinate.

They will give us the area of the triangle, that being

$$S = \frac{\frac{b}{k} \times b}{2} = \frac{b^2}{2k} \tag{3}$$

Also, the hypothenuse of the triangle will be necessary

$$c = \sqrt{\left(\frac{b}{k}\right)^2 + b^2} = \sqrt{\frac{b^2 + k^2 b^2}{k^2}} = \sqrt{1 + k^2} \left|\frac{b}{k}\right| \tag{4}$$

From the two previous equations we can derive the height of the triangle.

$$h = \frac{2S}{c} = \frac{\frac{b^2}{k}}{\sqrt{1 + k^2} \left|\frac{b}{k}\right|} = \frac{|b|}{\sqrt{1 + k^2}} \tag{5}$$

The height of this triangle is the shortest distance between the line and the origin.

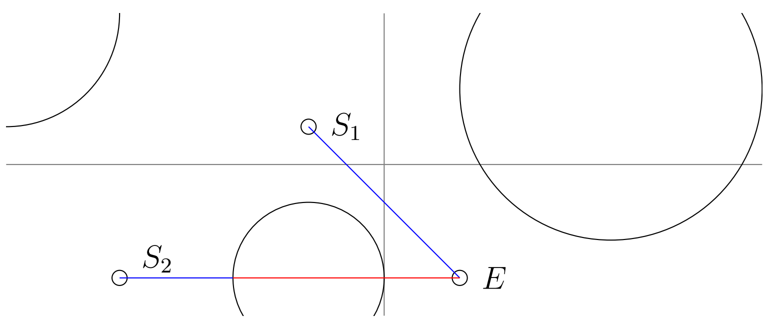
Now, based on this we can calculate the amount of intersections, if the distance of the line is greater than the circle's radius they don't intersect, if they are equal there is one point of contact and the line is tangential, otherwise there 2 intersections.

$$\begin{cases} I_n(p_1, p_2) = 0, h > C_{n_r} \\ I_n(p_1, p_2) = 1, h = C_{n_r} \\ I_n(p_1, p_2) = 2, h < C_{n_r} \end{cases} \tag{6}$$

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. Using that we can define our line of sight function. This function is 0 when no intersections are found and > 0 when there is at least one intersection, we can also convert it into a predicate $P(p_1, p_2)$ like this.

$$\begin{cases} P(p_1, p_2) = 1, I(p_1, p_2) = 0 \\ P(p_1, p_2) = 0, I(p_1, p_2) > 0 \end{cases} \tag{7}$$

The example below should make everything described above more graspable.



In the figure above the blue lines represent the line of sight. The line is blue all the way if the value of P for the two points the line connects is 1, it becomes red when the first intersection occurs. There exists a clear line of sight between points S_1 and E, but not S_2 and E.

Assuming the algorithm is not ill-formed and given correct input information it should formulate a near-optimal path, some list A of cardinality n, where each point is represted as a pair of x and y coordinates $\langle x, y\rangle$. It is required that $\forall a_i, a_{i+1} \in A : a_{i_x} < a_{(i+1)_x}$. If this is not true at some point the path turns back on itself. This game situation is unresolvable (more of that is mentioned in the "Limitations & Reflection" section).

In some edge-case scenarios errors in floating-point arithmetic might cause an unintentional collision. Since built paths might be touching the radius of the circle (due to the nature of the pathfinding algorithm) it might be useful to artificially increase the radius through some function $h(r) = r+t$, to ensure that the path is built at least t units away from the circles.

Fitting The Function

The fitting algorithm will operate on the list A, output by the pathfinding algorithm and assume the same ordering and $n = |A|$.

A_n is guaranteed to be the rightmost point, which will make the calculations a lot more intuitive. 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 construcing the Lagrange Interpolation Polynomial which does fit this usecase perfectly. We may follow a specific example and try to generalise it later.

For example $A = \{\langle 0, 1\rangle, \langle 6, 9\rangle, \langle 17, 4\rangle, \langle 19, 22\rangle\}$. All the coordinates are purposfully picked to be in the first quadrant, since in this way they are much easier to visualize.

First we may try to make a function $\phi_i(x)$ such that it equals to 0 at every point, but x_i. For simplicity we may make it be equal 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_2.

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

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

$$\hat{\phi}_2(x_2) = \hat{\phi}_2(6) = (6 - 0)(6 - 17)(6 - 19) \tag{9}$$

$$\phi_2(x) = \frac{(x-0)(x-17)(x-19)}{\hat{\phi}_2(x_2)} \tag{10}$$

$$\phi_2(x) = \frac{(x-0)(x-17)(x-19)}{(6-0)(6-17)(6-19)} \tag{11}$$

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

Formally, in this context, ϕ_i is called a "Lagrange polynomial basis". To make ϕ_i completely representative of the point at x_i we need to scale its vertical component by y_i. Let's define a new helper function $\psi_i(x)$.

$$\psi_2(x) = y_2\phi_2(x) \tag{12}$$

$$\psi_2(x) = y_2 \times \frac{(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)} \tag{13}$$

Based on this we can express $\psi_i(x)$'s values for the set A.

	$\psi_1(x)$	$\psi_2(x)$	$\psi_3(x)$	$\psi_4(x)$
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 deduce 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 a polynomial of the same order n as the cardinality of the input points. 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 list A of cardinality n it would look like the following.

$$f(x) = \sum_{i=1}^{n} \psi_i(x) \tag{14}$$

We can also expand our helper functions ϕ and ψ, so we get it to our final form.

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

$$\phi_i = \prod_{j=1, j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \tag{15}$$

The final form is the following, a proper Lagrange Interpolation Polynomial, a member of P_n.

$$f(x) = \sum_{i=1}^{n} \left(y_i * \prod_{j=1, j \neq i}^{n} \frac{x - x_j}{x_i - x_j} \right) \tag{16}$$

This is the final form of the function f, which given a set of points fits a function through them. It's naïve computational complexity is $O(n^2)$, which is subpar, possible improvements are mentioned in the "Limitations & Reflection" section.

Limitations & Reflection

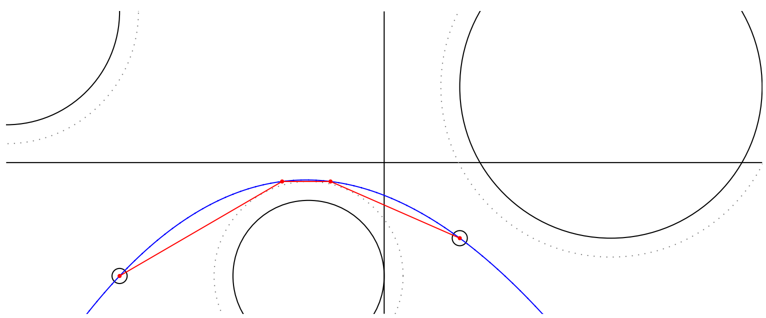
The described method of fitting covers a lot of scenarios, but it doesn't yield correct results when the next intended path point a_{i+1} on the path goes after some a_i. In this case, the pathfinding algorithm requires a path to turn back, which is impossible to do. This issue is most visible when reindexing: 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 two different values at the same x. Luckily, those scenarios never occurred 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 accidentally go through one of the circles. This oscillating effect (also known as Runge's phenomenon) can be mitigated by sampling more intermediate points equidistantly, 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. Using splines would be optimal, beyond that a spline of degree ∞ for a set of points *is* the Lagrange Interpolation. I would love to see them included in the curriculum.

All the solutions later developed by other people interested in the game employ similar approaches with some optimizations on the algorithm side. Some also make use of CV (Computer Vision) to automatically dispatch the data to the calculator which later forms a polynomial. Several applications use an additional step of the pipeline to optimize the resulting polynomial by approximating it further using other techniques.

The game has changes a lot since I started the project, the developer is very active on it, the latest features include customizable field size and backward-compatibility breaking changes, such as ignoring the y-intercept of a function entirely so the function starts at the position of the shooting soldier and much more.

Conclusion



Here is a representation of a game field with function f generated using my algorithm.

The radii provided to the algorithm were increased by 2.5, they are drawn in dotted gray. The path generated by Theta* is drawn in red. The path of the generated function is in blue. In this specific case the simplified polynomial is like this (the entire process of expansion is omitted for the sake of brevity, it would take an unreasonable amount of time to compute manually and an unreasonable amount of paper to print).

$$f(x) = 4.833... \times 10^{-5}x^3 - 1.181... \times 10^{-2}x^2 - (0.389...)x - 4.335...$$

In conclusion, I can for sure regard this research as successful. All the described techniques were applied and achieved great results. I won the standoff and got to explain my methodology to the defeated opponent, which was delightful. Beyond that, several possible improvements were already mentioned above. I would be interested in redoing that research in some time to see how my understanding of the topic has changed with the knowledge I will develop. It has opened my eyes to such areas of mathematics as Approximation Theory and Mathematical Analysis, which I now hope to study in the future.

References

[1] Archer, Branden, and Weisstein, Eric W. *Lagrange Interpolating Polynomial.* From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html.

[2] Weisstein, Eric W. "Lebesgue Constants." From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/LebesgueConstants.html

[3] Savateev, Aleksei. *Linejnaja algebra. Aleksej Savvateev i Aleksandr Tonis. Lekcija 1.1. Interpoljacionnyj mnogochlen.* Youtube, Uploaded by alexanderfilatov, 04 May 2018, https://www.youtube.com/watch?v=WgeW0oZTmxU.

[4] Lucas Catabriga. *Graphwar.* Graphwar, 17 Aug 2022, http://www.graphwar.com/.

[5] Holmér, Freya. *The Continuity of Splines.* Youtube, Uploaded by Freya Holmér, 07 Dec 2022, https://www.youtube.com/watch?v=jvPPXbo87ds.

[6] Runge, Carl (1901), "Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten", Zeitschrift für Mathematik und Physik, 46: 224–243. available at www.archive.org

[7] Stone–Weierstrass theorem. Encyclopedia of Mathematics. URL: https://encyclopediaofmath.org/wiki/Stone-Weierstrass_theorem

[8] Bishop, E., 1961. A generalization of the Stone-Weierstrass theorem. Pacific Journal of Mathematics, 11(3), pp.777-783.