Daniel Carvajal

Analysis on Bezier curves and their possible application

Table of contents

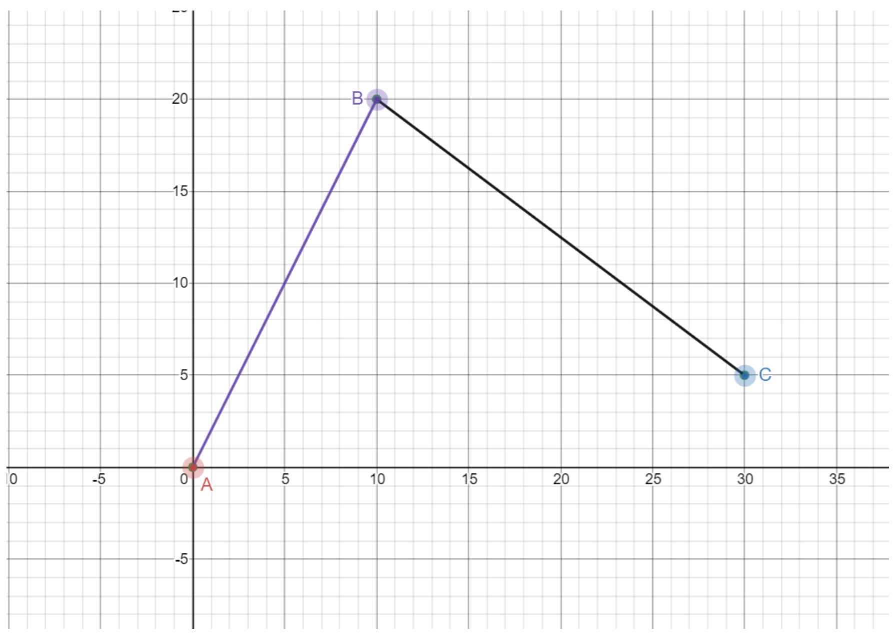
Introduction

As a programmer and robotics fanatic I have always wondered how I can model any curvilinear path for something to follow. This could be a robot following a predetermined path for a competition, or an AI following a path in a video game. In fact, both of those examples are real life scenarios which inspired me to write this IA. I realized that I needed not only a path, but also the angle at each point along the curve in order to turn the robot or correctly orientate the character. This meant I needed to find the derivative. Since we went over derivatives in class, I can now finally achieve this after years of struggling with this concept. I also realized that if I want to generalize it to any path, my formula could not be a function as not all paths pass the horizontal line test. Instead, I realized I needed a parametric equation to interpolate between any number of given points. I was happy to hear that parametric equations is also something In the HL math analysis coursework and I would finally be able to come up with an equation.

Research and understanding of the Bezier curve

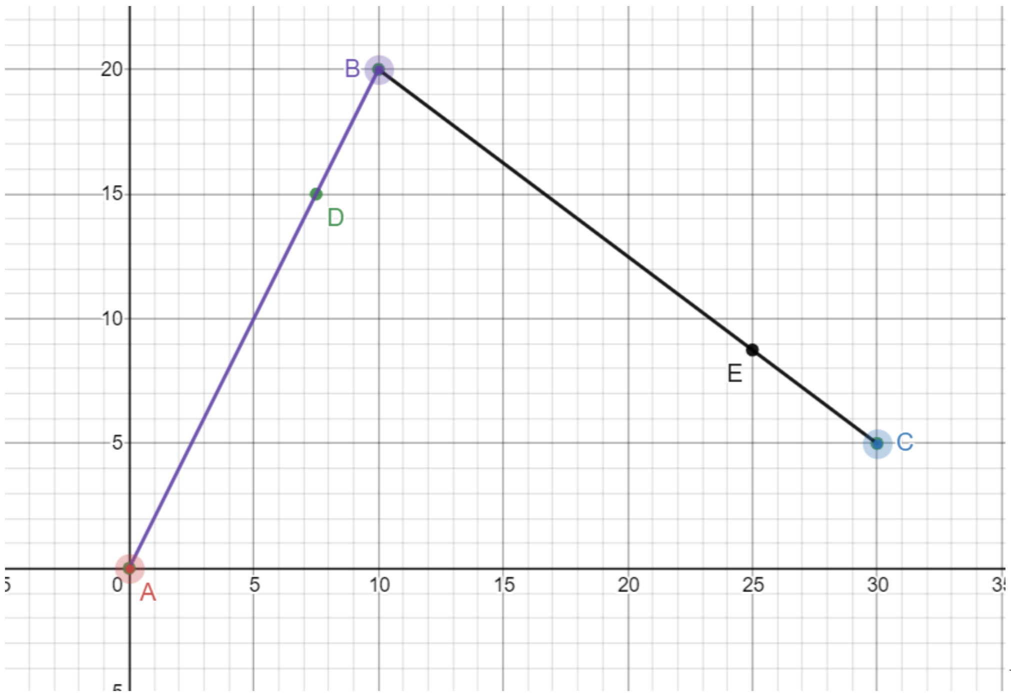
I realized that I had no idea how to start besides the fact that I wanted points for which my curve must go through and a way to change the path between these points. The simplest way was to make a simple equation for a line between each point by using point slope form, but this would not allow for manipulation of the path between said points. Then I thought of using quadratics that go through the first and third point, with the second point being the maximum or minimum. After further analysis I realized this would then result in only symmetrical curves. I wanted to have the most freedom possible so my formula could fit any equation. This is when I decided to stop trying to reinvent the wheel and took to research. I came across Bezier curves and realized that they were the answer. The form of Bezier curves I used is the one defined using De Casteljau's algorithm. The formula is based upon the basis of making lines between the points and interpolating between them.

Let's look at a quadratic Bezier curve to clarify. A 2nd order curve has 3 points. We are going to label them as A, B, and C. A and C are the endpoints, and B is used to change the path taken between those two points. The way the algorithm works is it first makes a line between A and B, and B and C

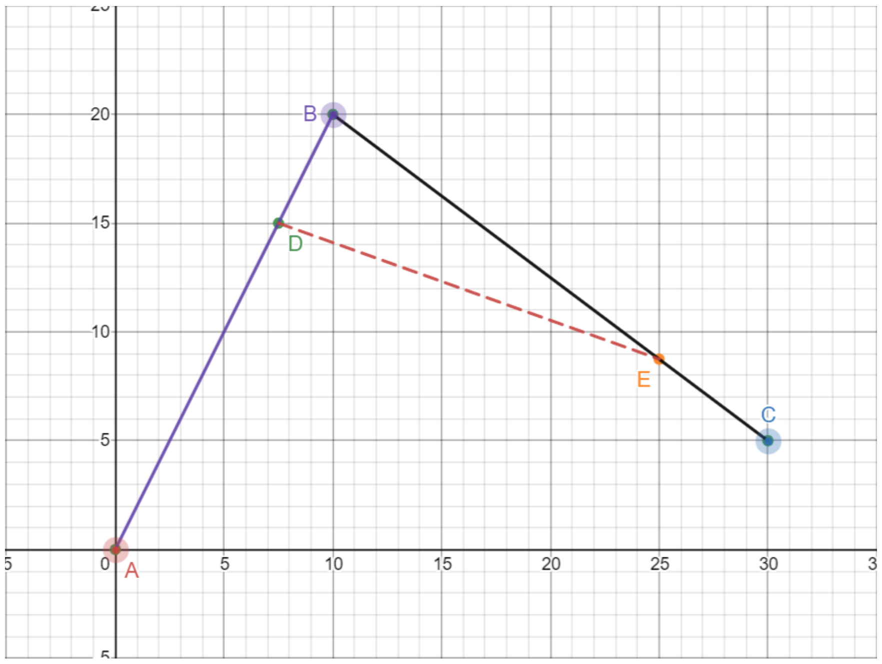


(Graph made using Desmos with equations derived in paper)

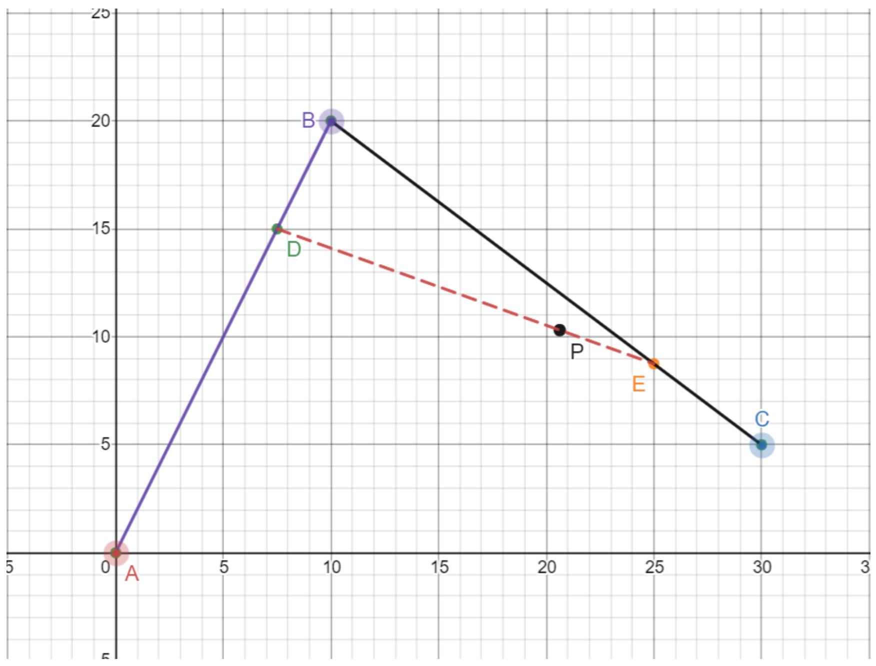
After constructing these lines comes the parametric part. We are to define two new points which start at A and B when $t = 0$ and end up at B and C when $t = 1$, Essentially, they follow the line. We will call these points D and E.



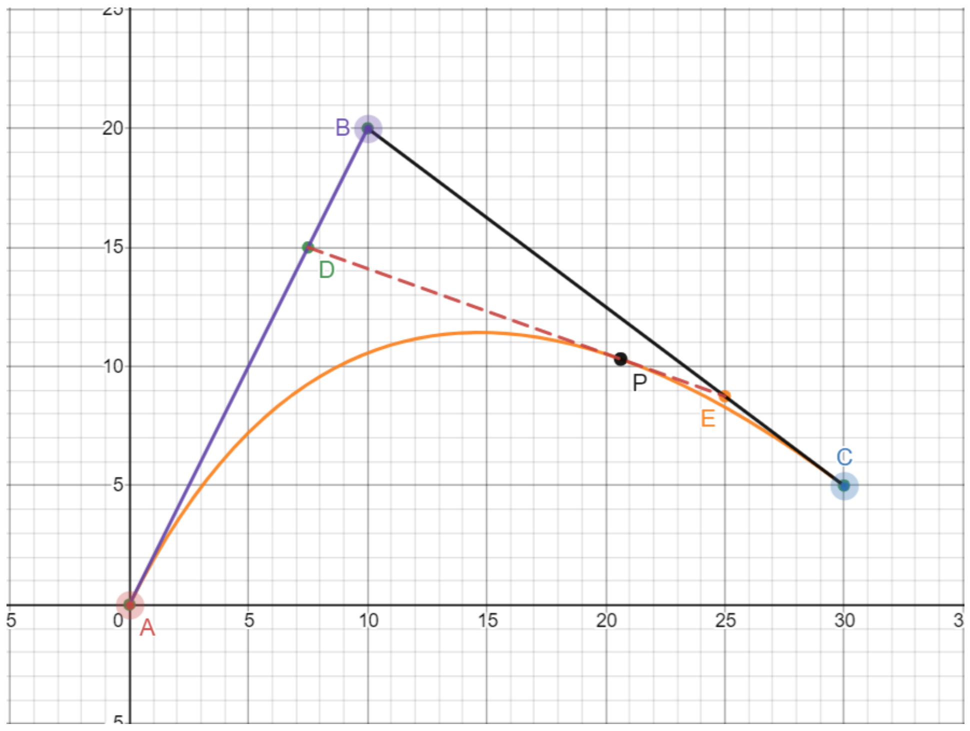
We then define a line that goes through D and E as such.



For the final step we then define a point which starts at D when $t = 0$ and ends at E when $t = 1$



With this we have the construction of our Bezier curve. As τ goes from 0 to 1, point P moves along a path between A and C.



Mathematical Construction of the Bezier curve formula

Now that I understand the basics behind Bezier curves, I have to derive the equation. In order to do so I first look at how to get to point P in vector form. I realized that this equation is basically a recursive way of finding a point that starts at point X when $t = 0$ and goes to point Y when $t = 1$; With this in mind I realized that a simple mathematical way to write this new point P is $P = X + ((Y - C) * t)$

Using that same type of thinking we can then substitute in the actual points used to define our point P, thus arriving at $P = D + ((E - D) * t)$

Points E and D are not given to us but instead defined using the same method described above se we can then substitute in their values in terms of A, B, and C which are all the given points. this leaves us with an equation of

$$P = A + (B - A) * t + (B + (C - B) * t - (A + (B - A) * t)) * t$$

This equation can then be expanded and terms canceled out to leave us with a polynomial in terms of A, B, C, and T of

$$P = Ct^2 - 2Bt^2 + 2Bt + At^2 - 2At + A$$

While this parametric vector formula would indeed make a bezier curve, we can not currently find the derivative of an equation including vectors and this is not standard parametric form. Thus we need to convert all of the vectors into their cartesian coordinates. This would also allow me to have a standard parametric equation where we have an equation for Y defined in terms of t and an equation for X in terms of t. Luckily for us, our equation does not use any vector math besides addition, subtraction, and scaling. Since both addition and subtraction of vectors is just adding or subtracting each component and scaling a vector is merely multiplying each component by the scalar, we can simply replace each vector with its X component to arrive at the x coordinate for the parametric equation and do the same for Y. This leaves us with a final parametric equation of

$$X = C_X t^2 - 2B_X t^2 + 2B_X t + A_X t^2 - 2A_X t + A_x$$

$$Y = C_Y t^2 - 2B_Y t^2 + 2B_Y t + A_Y t^2 - 2A_Y t + A_Y$$

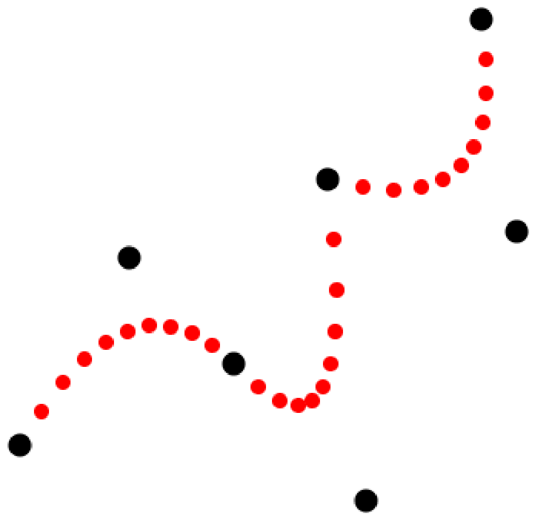
In order to find the derivative of a parametric equation you have to first find the derivative of both the X and Y equations and then do $\frac{Y'}{X'}$

Since both are just polynomials we can use the power rule to get an overall derivative of

$$\frac{2C_Y t - 4b_Y t + 2B_Y + 2A_Y t - 2A_Y}{2C_X t - 4b_X t + 2B_X + 2A_X t - 2A_X}$$

Connecting Bezier curves together to make one more flexible curve

The end goal is using these bezier curves in a program so as to have an object in a videogame or a robot or anything else follow any type of path. With this curve you can not have any points of inflection between points A and C. The only way to do this would be by using multiple 2nd degree Bezier curves with the endpoint of one being the starting point of the next. resulting in something like this



(made using javascript with code done by me which utilizes the math described in this paper)

The issue with this is that if not given the correct points, our supposed curve now has a sharp turn. In practicality this is not useful as in robotics or videogames or almost any other application the desired movement would be smooth and gradual, not sharp turns. In order for an equation to be without sharp turns the derivative of the equation must be continuous. Between the first and second Bezier curve segments there is no sharp turn, yet during the change between the 2nd and 3rd Bezier curve segments there is a significant sharp turn. Why? This is due to the fact that the 2nd, 3rd, and 4th points are all collinear.

Why must the points be collinear in order to maintain continuity?

A sharp turn in an equation happens when its derivative is not continuous at that point. This happens when $\lim_{t \to c-} f'(x) \neq \lim_{t \to c+} f'(x)$

Looking back at our conjoined Bezier curve, the only possible suspects for a point c where there would be a discontinuity are at the joints between two Bezier curve segments. This is because the middle points merely change the shape of the curve, thus it will still forever be a curve no matter its position, however when two curves are joined there is no guarantee that they together will form one curve.

Thus, in order to remove sharp turns, we need the limit of the derivatives of the segments to equal each other as they approach each other. Hence if we represent the first curve as $f(t)$and the second curve as $g(t)$ then $\lim_{t \to 1} f'(t) = \lim_{t \to 0} g'(t)$ in order for the curves to have no point of discontinuity between their intersection.

Since the equation for the derivative of each Bezier curve segment is continuous, $\lim_{t \to c} f'(t) = f'(t)$. This means that we merely need to substitute in 0 for $g'(t)$and 1 for $f'(t)$ and set them equal to each other.

following through with the substitution for $t = 0$ of the derivative we get

$$\frac{2B_Y - 2A_Y}{2B_X - 2A_X}$$

following through with the substitution for $T = 0$ of the derivative we get

$$\frac{2C_Y - 2B_Y}{2C_X - 2B_X}$$

(keep in mind that each A,B, and C is for each respective curve, thus B in the first equation is not the same thing as B in the second equation)

We then need only to set them equal to each other if they are to be collinear due to the explanation above. If we then both realize that the $g'(0)$ has only its first and second point in its equation and $f'(1)$ has only its second and third point, along with the fact that the startpoint of $g(t)$ is the same as the endpoint of $f(t)$ we can then denote these equations in terms $a,\ b,\ and\ c$ where $a\ =$ the middle point of $f(t)$, $b =$ the endpoint of $f(t)$ or otherwise known as the startpoint of $g(t)$, and $c\ =$ the middle point of $g(t)$. We can now substitute in our new terms and set them equal to each other to end up with an equation of

$$\frac{2B_Y - 2A_Y}{2B_X - 2A_X} = \frac{2C_Y - 2B_Y}{2C_X - 2B_X}$$

Upon closer inspection you can then see that both of these are a form of the formula to find the slope of a line between two points. They are both $\frac{\Delta Y}{\Delta X}$. This means that the slope between points a and b must equal the slope between points b and b. Thus, points a, b, and c must all be collinear in order for the entire jointed curve to be continuous. This only proves it for two joined curves, but since those two curves make one big curve, one can then apply the same logic to two bigger curves resulting in an infinite amount of small curves being possibly joined together as long as they meet the requirement above

Conclusion

In this paper we derived the formula for a 2nd degree bezier curve and discussed how it could be used to model any path for a robot to follow or other applications. We started off with understanding the geometrical derivation of the bezier curve. We then derived a parametric vector equation from our understanding of the formation of a bezier curve. This then proved to not be useful in application because it was not the standard form needed to find the derivative of a parametric equation. We then solved for X and Y in terms of A, B, C, and t in order to have a useful equation. This allowed us to find the derivative needed to orientate robots or other applications along the path. After some reflection we realized it was necessary to join bezier curves together in order to truly model any path between 2 endpoints. This resulted in the analysis of what made two joined bezier curves differentiable or not. Upon further analysis we concluded that if the 3 points surrounding a joint were collinear then the entire curve would be continuous. This can then be applied to robotics, or video games, or anywhere where paths must be modeled by using the formulas to move something along any desired path which goes through defined points.

Bibliography

Javascript.info Authors. “Bezier Curve.” *Javascript.info*, https://javascript.info/bezier-curve.

Luberodd, Eli. “Desmos graphing calculator.” *Desmos*, https://www.desmos.com/calculator.