Curves in Computer Graphics Report

160003289

March 2020

Contents

1 Overview

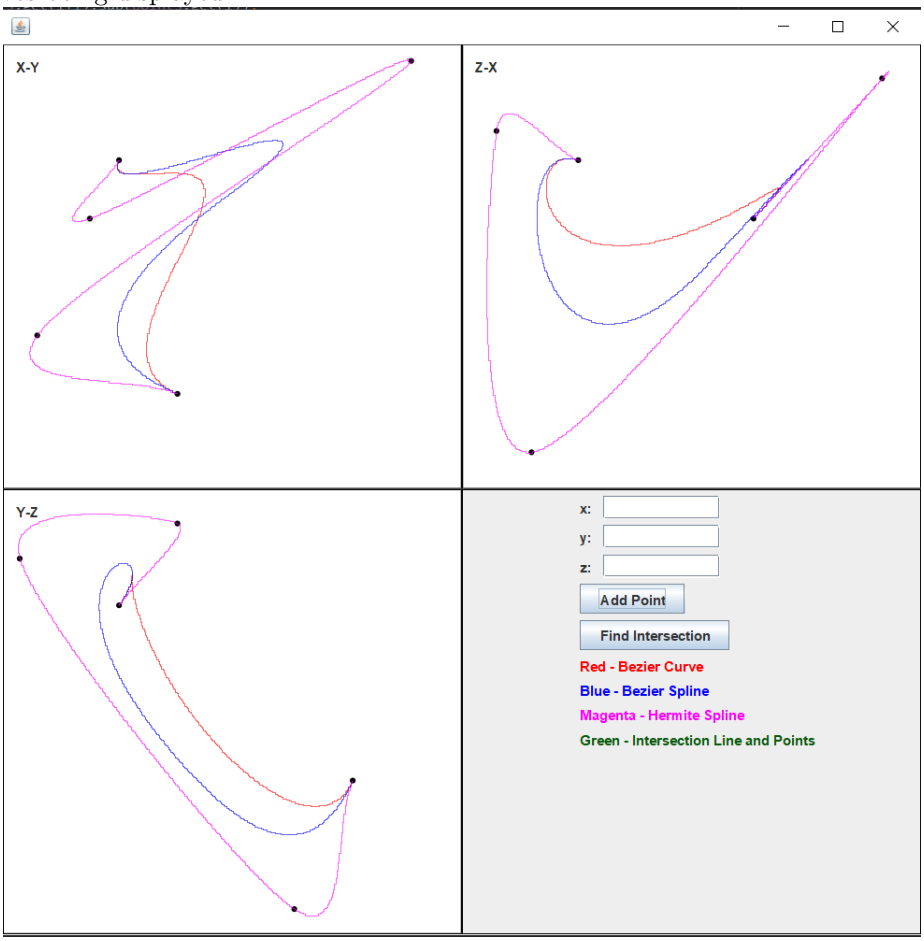
For this practical I was tasked with creating an application that allows a user to specify a set of control points which are then used to specify a set of curves.

2 Running Instructions

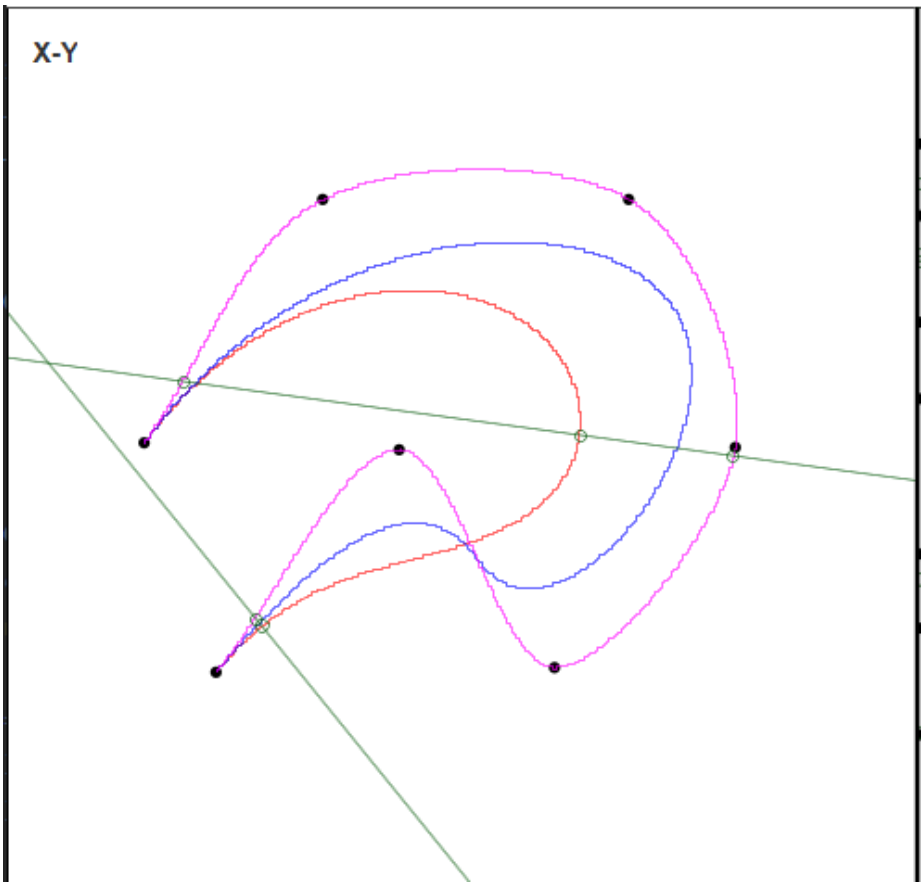
My submission is an Intellij project. Unfortunately I ran out of time to convert into a project that was easily runnable from the command line. When importing it into Intellij you should just have to run the main class "Curves" to run the application.

3 GUI and User Interaction

For my program I chose to use Java with Swing for my GUI. My GUI has 3 panels for 3 separate viewing planes of the curves (X-Y, Z-X and Y-Z). The fourth panel allows the user to manually specify a point by inputting coordinates and ask for a random intersection. Here is an image of the GUI with a set of curves being displayed.



A user can input points in two ways. One is by clicking on a point in one of the 3 planes which will add a point at that location with the third coordinate value being 0. The second is by manually typing in a set of coordinates and clicking "Add Point". Clicking "Find Intersection" will flatten all the points onto the X-Y plane (0 the z-coordinate) and find any intersections between the normal of a random point on the Bezier curve (Red) and the Hermite Spline (Magenta). These are displayed as shown below.

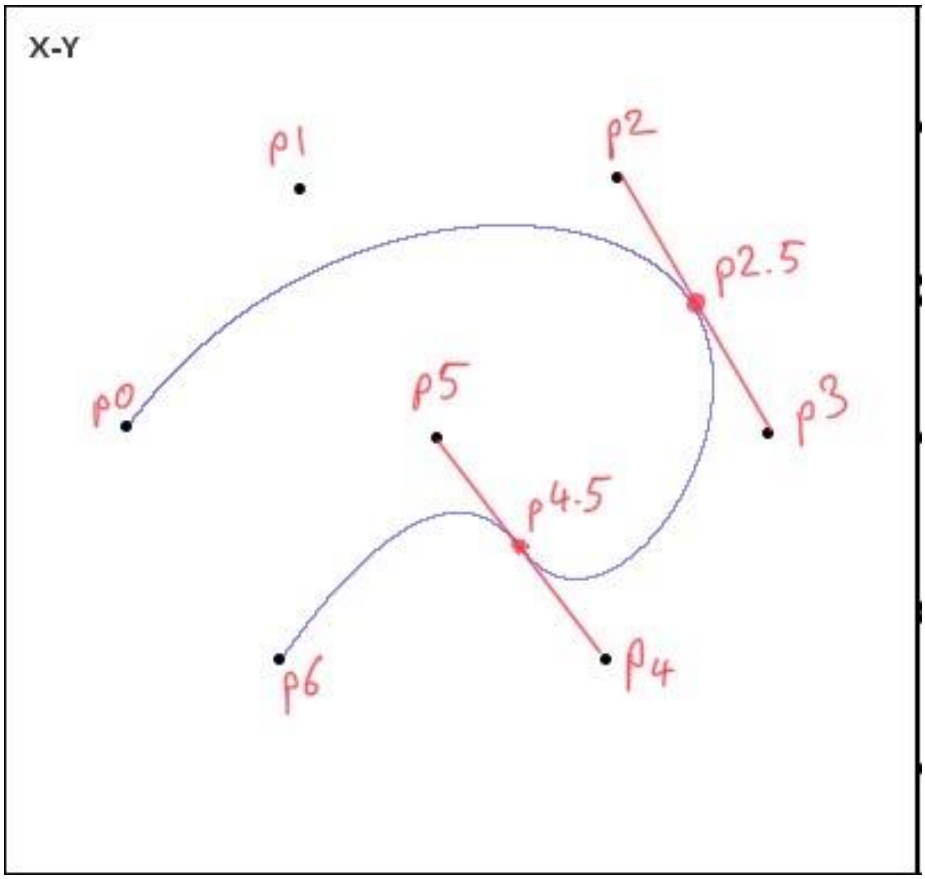


4 Bezier Curve

$$\boldsymbol{p}(u) = \sum_{i=0}^{n} \binom{n}{i} u^i (1-u)^{n-i} \boldsymbol{p}_i \tag{1}$$

This is the formula for a Bezier curve as stated in the slides. I implemented this to generate a general Bezier curve for any set of n points and I implemented the binomial coefficients using Pascal's triangle.

5 Bezier Spline



For the Bezier spline I added points between some pairs of points, in order to create a sequence of cubic Bezier curves that were at least C_1 continuous at the endpoints of each individual curve. As shown in the example above, which has the added points marked on, this was done by adding new points half way between 2 supplied control points. Depending on the number of control points there is sometimes a quadratic curve at the end of the spline as there is above. MAYBE ADD ABOUT HOW CURVES ARE JUST STANDARD CURVES

6 Interpolated Cubic Spline

6.1 First Attempt

For the third part of the practical I had to add a interpolated cubic spline between the control points. At first I implemented the interpolated spline found at https://www.value-at-risk.net/cubic-spline-interpolation/. This curve successfully interpolated between the points as long as their x coordinates were moving

in the same direction but it caused large spikes if points were close together since it is C_2 continuous and every part of the curve is dependent on every point. The curve was also not parametric which was the cause of it not being able to double back on itself, causing multiple y coordinates for a single x value. I left the code in my submission but it isn't being drawn to the screen.

6.2 Cubic Hermite Spline

After trying the first type of interpolated cubic spline I decided to implement a cubic Hermite spline (https://en.wikipedia.org/wiki/Cubic_Hermite_spline). Cubic Hermite splines are parametric and C_1 continuous and each curve only depends on the ones directly beside it to control the gradients at each end of the curve. I used the finite difference method to calculate the inner tangents with the x interval set to 1 (Since I was using equation 2 shown below paramaterised by t over a range of 0 to 1). This simplified the equation down to equation 3 shown below. The tangents at each end of the curve were simply the Bezier curve tangents which were the difference between the first two and last two points respectively. C_1 continuity is observed since the tangents are the same for both curves around each point.

6.2.1 Formulae

$$\boldsymbol{p}(t) = (2t^3-3t^2+1)\boldsymbol{p}_0+(t^3-2t^2+t)\boldsymbol{m}_0+(-2t^3+3t^2)\boldsymbol{p}_1+(t^3-t^2)\boldsymbol{m}_1, t \in [0,1] \tag{2}$$

$$\boldsymbol{m}_k = \frac{1}{2}(\boldsymbol{p}_{k+1} - \boldsymbol{p}_{k-1}), k = 1, \ldots, n-2 \tag{3}$$

$$\boldsymbol{m}_0 = \boldsymbol{p}_1 - \boldsymbol{p}_0 \tag{4}$$

$$\boldsymbol{m}_{n-1} = \boldsymbol{p}_{n-1} - \boldsymbol{p}_{n-2} \tag{5}$$

7 Intersection

When the user presses the find intersection button all points are flattened into the X-Y plane since the normal of a point on a line is only a line in 2D. To find the intersection between a normal and the Hermite spline I first generated a random value of u between 0 and 1. This value was then put into equation 1 to obtain the point and equation 6 to obtain the tangent at the point on the Bezier curve. These values were used to define a line using the fact that the gradient of the normal is defined as shown in equation 7.

$$\boldsymbol{p}'(u) = \sum_{i=0}^{n-1} \binom{n-1}{i} u^i (1-u)^{n-i} n(\boldsymbol{p}_{i+1} - \boldsymbol{p}_i) \tag{6}$$

$$\boldsymbol{m}_{normal} = \begin{pmatrix} -\boldsymbol{m}_{tangent}.y \\ \boldsymbol{m}_{tangent}.x \end{pmatrix} \tag{7}$$

I then had to try and find intersections between the line and each section of the Hermite spline. I did this using the method found here: https://stackoverflow.com/questions/1813719/intersection-between-bezier-curve-and-a-line-segment. This method works by transforming the line onto the x-axis and solving the cubic equation for the curve since the intersections are at y=0.

Firstly you let equation up the equation 2 be equal to the straight line ($u+\lambda v$ where u is your point on the line and v is the gradient of the normal). Then you subtract u from both sides to translate to the origin. Then you rotate the line onto the x-axis by using the rotation matrix:

$$\begin{pmatrix} \frac{\boldsymbol{v}_x}{|v|} & -\frac{\boldsymbol{v}_x}{|v|} \\ \frac{\boldsymbol{v}_y}{|v|} & \frac{\boldsymbol{v}_y}{|v|} \end{pmatrix} \tag{8}$$

This is just the standard rotation matrix with SOH CAH TOA used to fill in $sin(\theta)$ and $cos(\theta)$. Since we only care about the y coordinate you only perform the transformation that affects the y value. I used a cubic root solver found here (https://www.cs.rit.edu/ ark/pj/lib/edu/rit/numeric/Cubic.shtml), to solve this equation for y = 0. Each real root that is between 0 and 1 (The part of each curve we are using) is then an intersection.

There will always be at least one intersection point. I wasn't able to think of a good mathematical answer but the Bezier curve always remains inside the convex hull while the Hermite spline goes through each point at the edge of the hull. Therefore when leaving the hull in one direction, the normal line must intersect with the Hermite spline since it must leave between 2 points which are connected by the Hermite spline.

8 Extension

I was unable to find a good solution for making a truly 3D plot in Java but I show three viewing planes in my application to show the fact that my program works with 3D points apart from the intersection code.