Bezier Curves

1 Introduction

The assignment asks to implement a program through which a user can create and interact with a Bezier Curve. This implementation covers all the requirements of the basic specification. A jar file is provided for easy execution. This report also includes a short discussion on three dimensional Bezier Curves.

2 Files

The system can be run with “java -jar BezierCurves.jar” from within the submission directory. This starts up an interactive application where the user can create Bezier Curves. The file “settings.props” is a configuration file for the elements of the GUI. It should be in the working directory when running the program.

The source code written for this assignment is in the packages ‘main’ and ‘components’. The classes in the ‘util’ package have been written for previous university assignments and are being re-used for convenience.

3 Drawing Decisions

Every one of the following elements appears in the main drawing area, hereby referred to as canvas. The canvas takes up the major part of the Application window.

On the right of the canvas is a panel of switches. They can be used to turn on and off the drawing of the different elements on the canvas. The elements always exists, but they can be invisible.

3.1 Control Points

Control points can be manipulated in the following ways: left click on the canvas to create, click-and-drag to move, left click on a control point to remove it.

The user does not have to specify the order of the Bezier Curve. A curve is automatically drawn every time the list of control points is updated.

A small hurdle in the development was the size of the Bernstein Coefficient. A Bernstein Coefficient is defined as $b_{i,n}(x)=\binom{n}{i}*x^i*(1-x)^{(1-i)}$. A Bezier Curve with $n+1$ control points (indexed from 0 to n), has the following equation $p(u)=\sum_{i=0}^{n} b_{i,n}(u)*cp_i$, where cp_i stands for the i^{th} control point. Binomial coefficients get large very quickly, and the initial, naive, calculation caused the program to send all points p(u) for $u\in(0,1)$ to (0,0), when the number of control points reached low 20ies. The following approach was taken.

A binomial coefficient is defined as $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ since the amount of numbers between n and n-k is exactly k, this expression can be re-written as $\binom{n}{k} = \frac{n}{1} * \frac{(n-1)}{2} * \frac{(n-2)}{3} * \ldots * \frac{(n-k+1)}{k}$ and they are calculated in this order. Between two consecutive numbers (n, n-1) one is divisible by 2, between three consecutive numbers (n, n-1, n-2) one is divisible by 3, and so on. This guarantees that there is no precision loss. Implementing the binomial coefficient like that allows up to 60 control points to be created before the system started misbehaving.

3.2 Line Drawing

Drawing a line happens in segments. Each segment is a line with a start and end point both specified in double precision. The configuration file sets “curve number of points”. This is how many points are calculated from the equation of the Bezier Curve. A segment is drawn between every consecutive pair. The Graphics2D class from the java.awt library was used to draw the segments. It draws in double precision, creating a very smooth curve.

The segments are calculated every time the control points are updated. The total length of the curve is calculated at the same time.

3.3 Samples

The number of samples can be specified in a text box on the left side of the screen. The samples themselves are invisible by default.

The curve is sampled uniformly with respect to arc length. The distance between two consecutive samples is calculated from the total length of the curve. The algorithm iterates through the segments, adding their different lengths. Every time it hits the pre-calculated distance, the current segment’s end point is used as a sample.

Samples are stored separately, so that they can be manipulated on their own.

3.4 Tangents

By default, tangents are visible, even though their samples are not. Tangents are calculated only when sampling and they are stored with the line segments. Therefore, if the Bezier Curve is sampled repeatedly, the tangents will be reused. This approach is time efficient, as the tangent vector is calculated as the first derivative of its respective sample point. Essentially, the equation for the Bezier Curve is computed for every tangent. In this case, reusing old calculations is very much preferred.

The first derivative for a sample point returns the direction of the tangent **from that point**. So, the equation of the line that needs to be drawn is $p + \lambda t$, where p is the sample point, t is the tangent vector, and λ is units along the line. The length of the tangent line is configured in the settings file.

3.5 Curvature Vectors

By default, curvature vectors are invisible. As with tangents, they are calculated only when sampling and they are stored with the line segments. Since curvature is computed as the derivative of the first derivative of its respective sample point. Reusing old calculation results is even more important.

The second derivative for a sample point tells the location of the curvature vector with respect to the tangent. In order to determine the actual curvature vector, the angle between the first and second derivative is calculated. Effectively, it is the angle that starts at the first derivative and rotates counter-clockwise until it reaches the second derivative. If that angle is less than π, the curvature vector is the normal vector, on the left of the tangent. Otherwise, it's the normal vector on the right of the tangent. Note that the dot product is not sufficient. In the range 0 to 2π, there are two angles for every value of the cosine.

Like the tangent vector, the curvature vector is the direction **from the sample point**. The equation for the line to draw is $p+\lambda c$. The length of the curvature line is configured in the settings file.

4 Three-dimensional rendering

Several attempts were made to draw the Bezier Curve in three-dimensional space. However, all of them failed at compilation time with no explainable error. A couple of 3D graphics libraries such as J3D were attempting to access resources that couldn't be found. Two attempts were made with Processing as well. One directly writing Processing code and one where the Processing core was imported into Java. Both required native libraries that were configured in the class path and/or in the Djava.library.path to no avail.

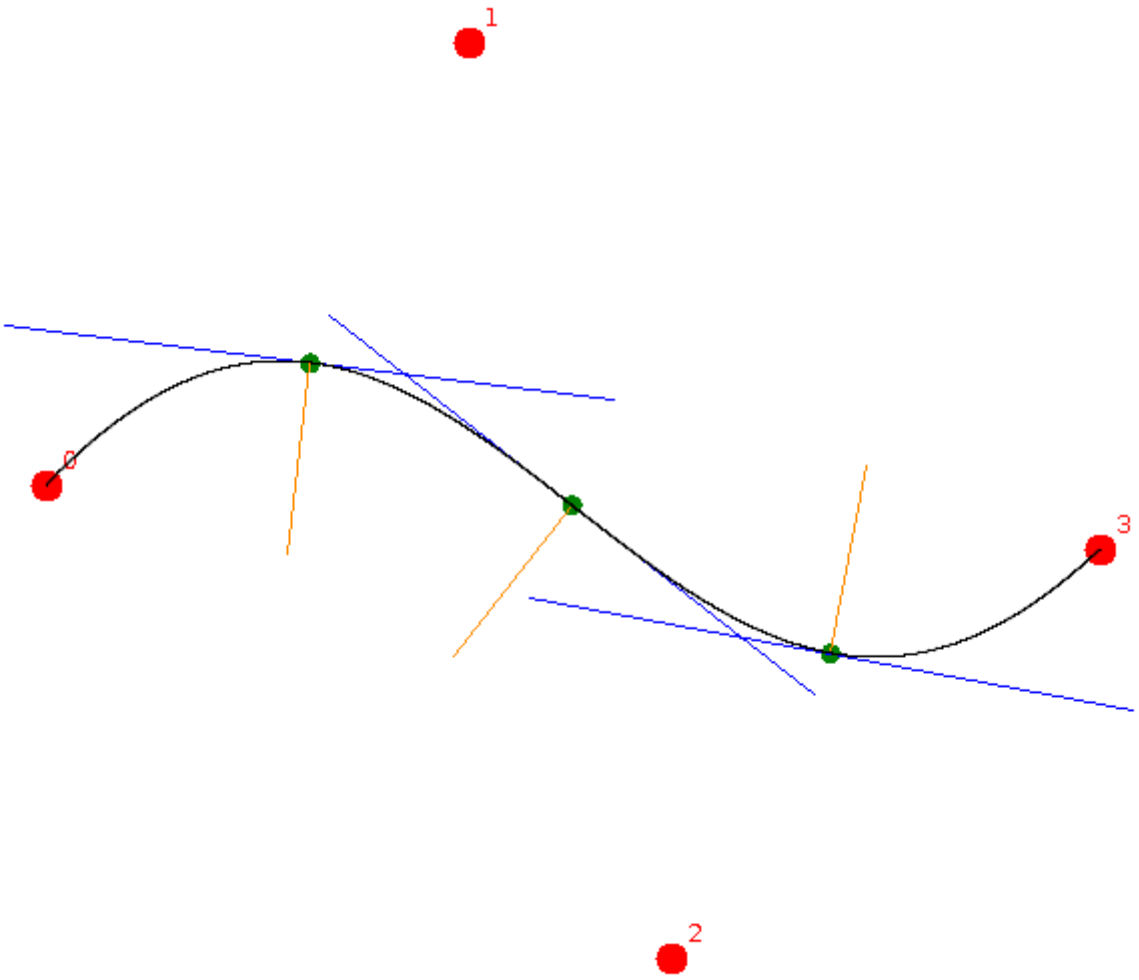
Here is the approach that would have been taken in developing a 3D Bezier Curve. The camera would be lockable, so that the curve can be interacted with, only when the field of view wouldn't change. When unlocked, only the camera would move.

When a control point is entered by clicking, the coordinates of the click would be rotated and translated via matrix operations to obtain 3D coordinates. The computations of the line segments, sample points, and tangents would remain the same, except there would be 3 values (x, y, z) by which to multiple the Bernstein Coefficient for each control point.

Calculating the curvature vector from the three-dimensional derivatives will require the following computations. Calculate the surface normal as the cross product of the derivatives. Then calculate the cross product of the second derivative and the surface normal. This gives a vector perpendicular to the second derivative, and in the same plane as both derivatives. Then calculate the dot product of that and the first derivative. If the result is positive, then the curvature is on the left of the tangent, otherwise it's on the right of the tangent.

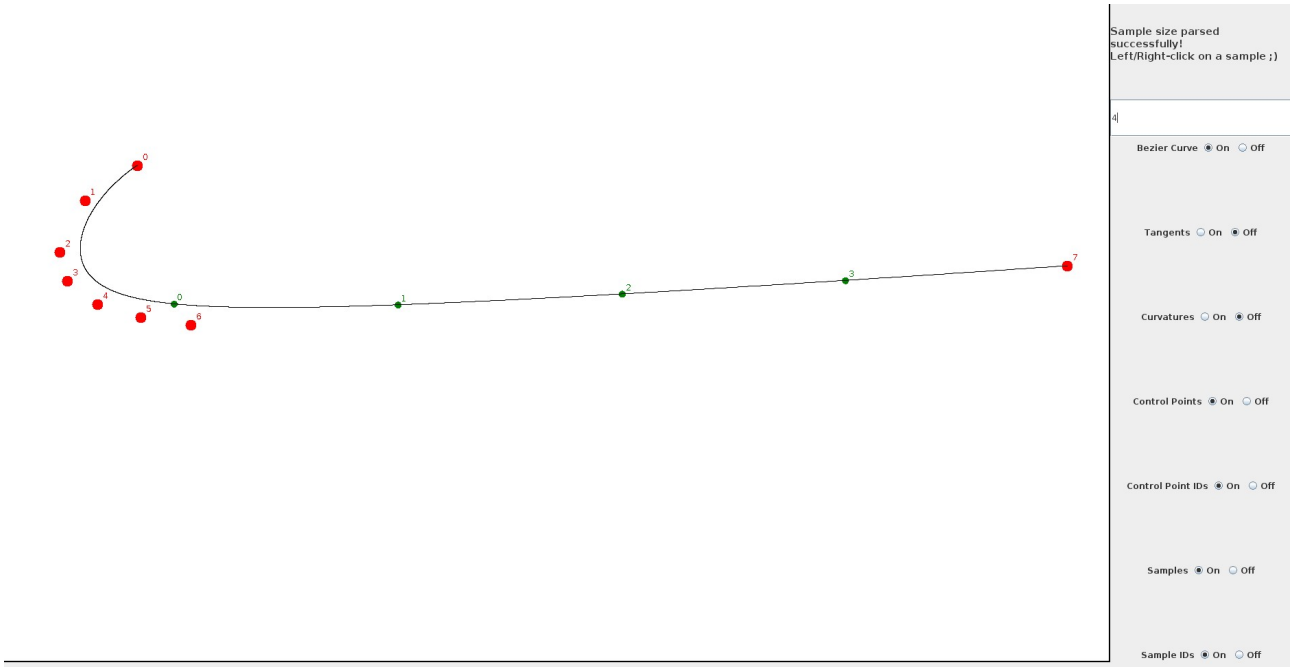
5 Appendix

Here is a Bezier Curve, similar to the example in the lectures.



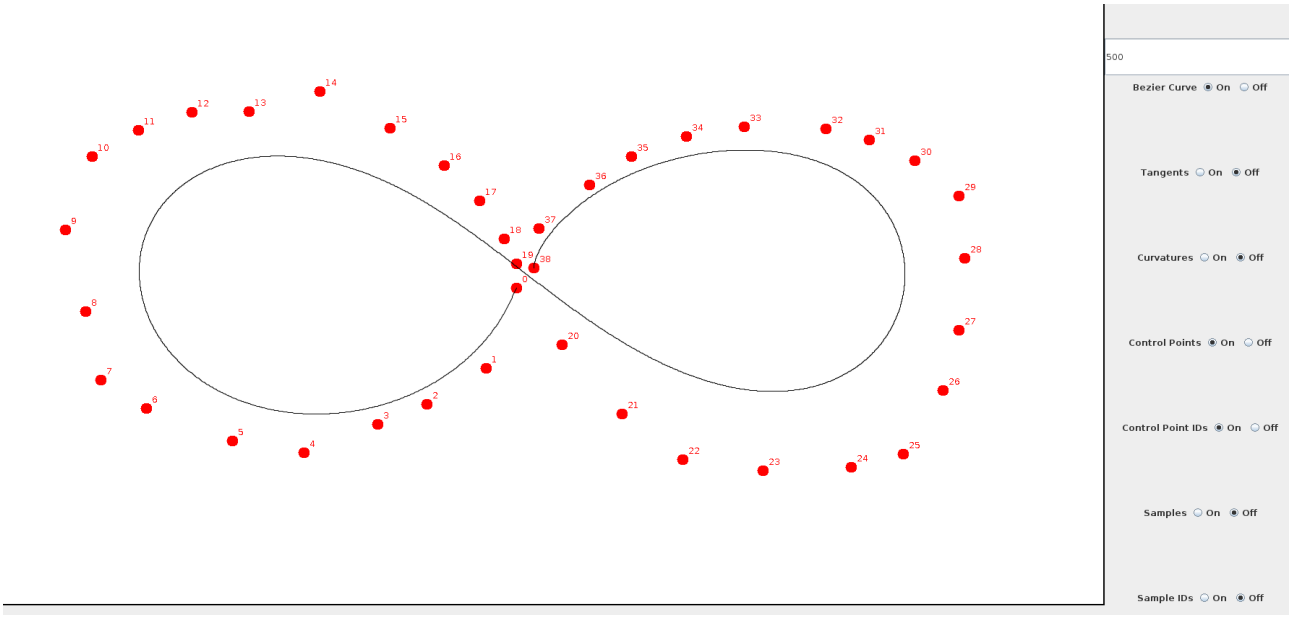
It shows that tangents, curvatures, and the curve itself are correctly calculated.

The next example shows that the curve is sampled uniformly with respect to arc length.



The gaps between consecutive samples are equal. If the curve had been sampled uniformly with respect to the parameter u, the samples would be more on the left, towards the cluster of control points.

The last example shows that the system can adequately handle a large amount of control points.



Now, using the switches on the right to turn off everything except tangents, a good-looking infinity can be obtained.