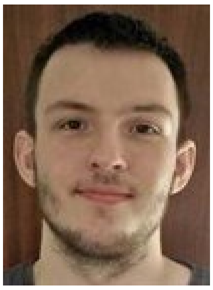
Technical University of Denmark

02561 - COMPUTER GRAPHICS

Introduction of Quadratic and Cubic Bezier curves into 2D drawing program.



JOACHIM S. MORTENSEN
S175179

DECEMBER 19, 2020

Abstract

Bezier curves are ubiquitous in computer graphics because of their efficiency and ease of use. They are used often in vector graphics, and even things such as CSS animation.

This project relates to the second worksheet of the course, expanding the 2D drawing program to include support for drawing rational quadratic Bezier curves, as well as cubic Bezier curves.

The result of this project, the expanded version of the drawing program is available at the following URL: `http://www.student.dtu.dk/~s175179/02561-ComputerGraphics/proj/proj.html`

The lab journal is also available at this url: `http://www.student.dtu.dk/~s175179/02561-ComputerGra`
`index.html`

Introduction and Maths

Bezier curves are a way of representing a polynomial through n control points $(\mathbf{P_0}, ..., \mathbf{P_n})$.

The formula looks as follows:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P_i}$$

Eq.1 ; Explicit definition of Bezier Curve

Where n and i are the binomial coefficients defined by:

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

Eq.2 ; Definition of binomial coefficients

Expanding $Eq.1$, we get:

$$B(t) = (1-t)^n \mathbf{P_0} + \binom{n}{1} (1-t)^{n-1} t \mathbf{P_1} \dots \binom{n}{n-1} (1-t) t^{n-1} \mathbf{P_{n-1}} + t^n \mathbf{P_n}$$

Eq.3 ; Expanded definition of Bezier Curve

The above equation holds for $0 <= t <= 1$.
To give an example, lets try with $n = 3$:

$$B(t) = (1-t)^3 \mathbf{P_0} + 3(1-t)^2 t \mathbf{P_1} + t^3 \mathbf{P_3}$$

Eq.4 ; Specific equation for Beizer curve with 3 control points

Implementation

In order to implement bezier curves into the drawing program, a couple of changes must be made. Implementing bezier curves can be done in a couple of ways, an interesting property of quadratic bezier curves is that the curve itself will be completely inside the triangle defined by the three control points.

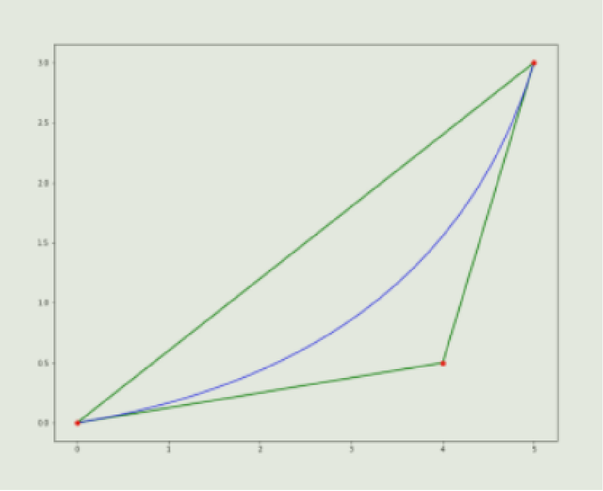


Fig. 1 ; Polygon described by controlpoints

This makes it possible to draw the curve in the fragment shader.
Another way of drawing a bezier curve, is as a line strip, with a sufficient amount of vertices as to produce a smooth curve.
This is the approach I have used in my implementation, linearly interpolating t by a stepsize of $\frac{1}{64}$. This results in 64 vertices per line segment, leading to a nice smooth curve.

My expansion to the drawing program also allows for coloration of the bezier curve, interpolating smoothly between the colors of the first two placed points along the curve as seen here:

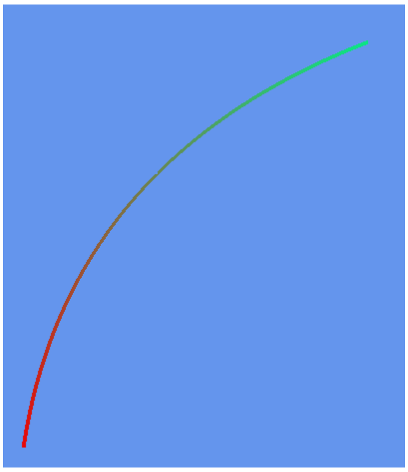


Fig. 2 ; interpolation of color, from red to green

Apart from just implementing quadratic bezier curves, I have also implemented cubic bezier curves, allowing one to describe a curve with four control points.

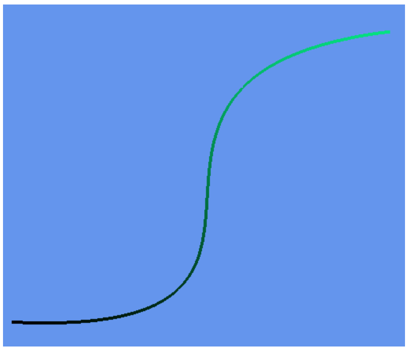


Fig. 3 ; Cubic Bezier curve

This is done through line segments again and expanding this code to support bezier curve to the nth-degree allowing for bezier curves with as many control points as one wishes would be very simple, whereas doing the same with a shader based solution would be less so.

Conclusion

Rendering curves through line segments are fast and requires very little shader work, however it makes live editing less optimal, as one would have to recalculate all vertices of a segment, and push all of them to the buffer. Rendering the curve through a fragment shader is more optimal for editing, as much less data has to be pushed to buffers, however it is also more complex, and obviously requires more shader work. However the current solution is also not quite as dynamic as one might have hoped and would definitely be an area for improvement.