INCLUDING RATIONAL QUADRATIC BÉZIER CURVES IN A FAST, USER-FRIENDLY 2D DRAWING PROGRAM

Lucas Vilsen (s224195), Alexander Wittrup (s224196)

Our editor: You can try out our editor at: `https://www.student.dtu.dk/~s224195/project/intro.html`

ABSTRACT

This project seeks to implement rational quadratic Bézier curves in a 2d drawing program with high performance by using matrix multiplication and the cubic Bézier basis matrix to efficiently compute the points on the Bézier curve. Furthermore, we extended this capability to include cubic and quadratic Bézier curves. This project also focuses on and discusses how to properly implement these features while maintaining a clean user interface and experience. The project discusses how to seamlessly make the tools available to the user in a self-explanatory manner. Ultimately, a logo editor is created, where the user can create logos, color them as they want, modify and save them in a high resolution for later use. All this while keeping a clean UI and performance high.

1. INTRODUCTION

In this project, we implement rational quadratic Bézier curves in a 2d drawing program, focusing on high performance and a strong user interface. We decide to implement rational quadratic Bézier curves through matrix multiplication and using the cubic Bézier basis matrix. The resulting computation of Bézier curves should be both quick and have a clean look. In addition to this, we implement the cubic Bézier curves, with a high focus on making it easy to use. While some of this project focuses on creating the Bézier curves, another part of it is creating the UI to seamlessly interact with the curves. For this reason, this project also discusses the best methods to implement these features and how to present them to the user. To create a complete program, multiple other features are implemented such as resetting canvas, saving images, and applying color. We also add undo/redo features with keybinds to give the feeling of a robust application.

2. METHODS

In the following section, we will describe our methods in more detail and our considerations for creating the user interface and user experience.

2.1. Rational Quadratic Bézier Curves

There are several methods to construct Rational Bézier curves. All methods rely on a t variable that is mapped from 0 to 1 with 0 being the starting point of the curve and 1 being the end point of the curve. An array of n equally spaced numbers in that interval, where n is the resolution of the curve, is then created. You can then either use a simpler mathematical formula to construct the Quadratic Bézier Curve [1] or a more advanced method to create either Quadratic or Cubic curves effectively using matrix multiplication [2] [3] [4]. We choose to implement the latter as this allows the user to experiment with both Quadratic and Cubic Bézier curves.

Here the cubic Bézier basis matrix is used to calculate the points at a timepoint t:

$$\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

This matrix is used in the following function:

$$P(t) = (t^3\ t^2\ t\ 1) \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

Quadratic Bézier curves can be constructed by having either P_1 or P_3 as 0. Solving this matrix equation gives a point on the curve at time t, which can then be displayed to the user.

2.2. Modifying existing curves in real-time

The above-mentioned method also gives us the option to show the curves to the user as they are creating them. This improves the user experience significantly and provides a much better interface for drawing. If the condition described below is true, then a line will appear.

$$p_0 = p_1 \quad \text{and} \quad p_2 = p_3$$

When the user then moves the mouse either when creating the line or when modifying the curve, the points are updated to the cursor position. As this method is relatively fast, this can be done without any noticeable latency for the user.

These points are kept in memory and shown to the user when constructing the logo. This allows the user to drag a control point around and thereby, again, modify a curve and see the changes in real-time.

Creating the lines that make up the curve is relatively simple, given the points that is created from evenly spaced t-values in the cubic Bézier equation because we simply create a thin rectangle between all points. In the following is the simple equations we use to create these line segments.

$$\text{len} = \sqrt{(P_2 - P_1)^2}$$

$$n_y = \frac{(p_{2x} - p_{1x})}{\text{len}} n_y = \frac{(p_{2y} - p_{1y})}{\text{len}}$$

And then with these values calculated we can create all four points using addition/subtraction or the first or second point.

$$Q_m = (p_{mx} \pm n_x, p_{my} \pm n_y, 0.0)$$

This reduces the resolution needed for a curve to appear clean. While other, more effective methods for doing so are already researched [3], we believe this process worked well enough for now.

2.3. Additional capabilities

A key aspect of this project is to create a complete solution that helps the user from start to finish. The above-mentioned methods now enable the user to create and modify both quadratic and cubic Bézier curves. In addition to this, we implement some features to help the user. This is always a balance between getting an effective solution and balancing time between tasks. As we did not use any external libraries, certain tools may not be as fully optimized as possible. However, we believe they perform adequately for this project.

- **Delete curve**: A way to delete a single curve.
- **Reset Canvas**: A way to delete all curves and start again from a clean canvas.
- **Color Logo**: We decided to implement an optimized grid search to initiate from a single point and draw until any edge is met, which allows the user to color the logo with a color of their choice.
- **Save Image**: You can save the image to your pc locally with a transparent background.
- **Undo & Redo**: When performing actions, the previous state is saved, so that the user can undo actions. This is also implemented for future states when the user undoes to revert if an undo is performed by mistake. These features is implemented as keyboard shortcuts.

With these features in place, the program hopefully appears more robust to the user.

2.4. Simple user-interface

The tools explained above are all implemented in a simple menu on the left side of the screen. A small piece of logic controls the buttons and the relationship between them. To further improve the user experience, a demonstration video, and an intro page is created that will guide the user and hopefully resolve any questions that they may have.

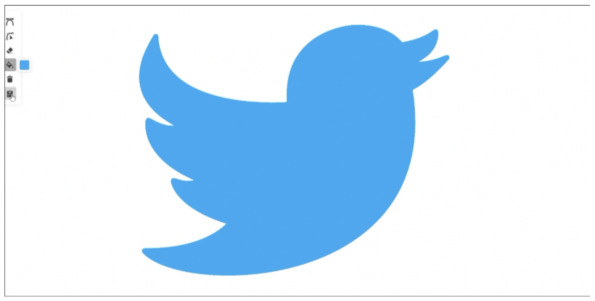
3. RESULTS

The resulting program can be seen and tried at: `https://www.student.dtu.dk/~s224195/project/main.html`.

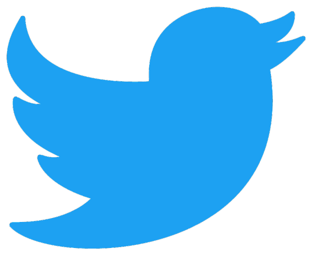
A demonstration video is available at: `https://www.student.dtu.dk/~s224195/project/intro.html`.

When opening the page, the editor starts with an empty canvas:

You can then create logos as demonstrated in the video:



The logos can be saved to be used wherever you want:



4. DISCUSSION

Our overarching goal was to balance the mathematical sophistication of rational quadratic and cubic Bézier curves with a straightforward and intuitive user experience. While the project demanded careful attention to computational details, our measure of success was whether users with no mathematical knowledge could interact easily with these tools. This section reflects on the key design decisions, technical optimizations, and practical considerations that guided our development process.

4.1. Optimizations

Performance and responsiveness were critical to ensuring a smooth user experience. One of our primary strategies involved carefully managing the data flow between CPU and GPU. By indexing and updating vertex data through `bufferSubData` calls in WebGL, we minimized overhead and significantly improved rendering speed. This optimization ensured that adjustments to control points, curve modifications, and other user actions could be reflected on the screen almost instantaneously, or at least with no noticeable latency to the user. As described earlier we decreased the resolution in the curves by creating lines between the points, creating more detail with fewer points.

These optimizations collectively enabled us to maintain high frame rates and low latency even as users generated many curves. The result is a fluid drawing environment that feels responsive, reinforcing the impression that the complexity of rational and cubic Bézier mathematics is elegantly managed behind the scenes.

4.2. UI and UX

From the start, we recognized that the best technical solutions would mean little if the user struggled to access and understand them. Thus, we prioritized an interface that limited on-screen clutter, relying on intuitive icons and tooltips rather than overwhelming users with unnecessary options. By presenting only essential controls—such as curve creation, editing, coloring, resetting, and saving, we believe we lowered the barrier to entry for those new to vector graphics.

Real-time curve manipulation was another key focus. As users drag control points, the curves update instantly, allowing them to see the direct impact of their actions. Achieving this required both fast computation and thoughtful UI design: without fluid interactivity, even a mathematically sound implementation would feel cumbersome. Ultimately, we believe the UI and UX decisions reflect this.

4.3. Simplicity and Features

Defining the scope of features was a delicate balancing act. While we identified numerous enhancements that could increase the tool's versatility—such as complex snapping tools, gradient fills, more complex anti-aliasing or faster cpu rendering options, we chose to focus on the more central features to ensure we could finish what we wanted and to not complicating the interface too much. Given our time constraints and the goal of delivering a fully functional program, we opted for a focused feature set that includes essential editing tools, basic coloring, undo/redo functionality, and the ability to save results.

This deliberate restraint ensured that users could explore and master the tool quickly. We focused on key tools, which we through our demonstration video showcased as enough to construct complex icons.

4.4. Creating a Complete Program

Our vision extended beyond a technical demonstration: we aimed to create a self-contained environment where users could move from a blank canvas to a finished, exportable design. We wanted to do more than just a proof of concept, as we believe we succeeded in that regard.

In retrospect, while there are always ways for more improvement, such as adding more advanced features or faster implementations, we hope that our project makes for a unique fun way to interact with Bézier curves in a 2d manner.

4.5. Opportunities for improvmement

There were a few times where we felt limited because we wanted to use pre-existing packages but we couldn't use external libraries, and therefore we went for a more hacky approach to solve some of the problems. For example, it would have been made for a better end product to use the method of triangulating polygons to fill-in shapes created with Bézier curves.

5. CONCLUSION

In this project, we successfully implemented rational quadratic bézier curves together with cubic bézier curves. By focusing on performance, we can generate and render many bézier curves on the screen simultaneously while keeping the resolution of the curve high. The user can modify and see the changes in real time. In addition to this, we successfully created a strong user interface for interacting, creating, modifying, and deleting curves, together with the ability to paint, reset, and save your logo. As shown in our demonstration video, our editor allows you to create sharp-looking logos quickly and seamlessly.

6. REFERENCES

[1] Alan Wolfe, “Bezier curves,” https://blog.demofox.org/2014/03/04/bezier-curves/, Accessed: 2024-12-18.

[2] Keith Peters, “Coding curves 08: Bézier curves,” https://www.bit-101.com/2017/2022/12/coding-curves-08-bezier-curves/, Accessed: 2024-12-18.

[3] Charles Loop Jim Blinn, “Resolution independent curve rendering using programmable graphics hardware,” https://www.microsoft.com/en-us/research/wp-content/uploads/2005/01/p1000-loop.pdf, Accessed: 2024-12-18.

[4] Michigan Technical University Unknown, “Rational bézier curves: Conic sections,” https://pages.mtu.edu/ shene/COURSES/cs3621/NOTES/spline/NURBS/RB-conics.html, Accessed: 2024-12-18.