Bezierbox – Milestone

by Team 5 – Jonah Bedouch, Brandon Wong, Richard Villagomez, and Nicolas DePalma

2025-04-20

This report is also available online at https://bezierbox-reports.vercel.app/milestone.

Links

A video update is available at: https://drive.google.com/file/d/1UJ8wmUFjk8Y7SM8j0GG6p-8me0XkNfn3/view?usp=sharing

The powerpoint used in that update is available at: https://drive.google.com/file/d/1mlKobVhQc8i1XiJg4n0LtbvXh19By9nL/view?usp=sharing

Results

To work on the first portion of this project, we split into two groups – a group focused on parsing a ttf file into bezier points, and a group focused on rendering those bezier points in a web canvas.

On the parser front, we were able to create working code that can load a `.ttf` file from within html, then extract the important tables from within the file that are needed to convert from unicode to glyph numbers, and to convert from a glyph number to a set of bezier points. We did this by using a tool called Emscripten, which compiles C++ into WASM that can be run in the browser. This turned out to be a more challenging task than expected, since there are many different unicode-to-glyph tables, and all of them have to be updated in order for the font to fully work correctly. For now, this is okay, but when writing back to the fonts we may run into issues that require the use of an external library, depending on exactly how modifying a curve ends up working.

For the renderer, we implemented a system that takes in the parsed glyph data from a .json file and renders it using the HTML Canvas API. The glyph data contains multiple contours, each defined as a sequence of points labeled as on-curve or off-curve (control) points. The renderer iterates through each contour, determines how to connect the points based on their types, and dynamically constructs the outline using a combination of lines and curves. It also adds midpoints when necessary to properly interpolate between control points and ensure continuity. The result is a smooth rendering of glyph outlines, with blue circles for on-curve points, red circles for control points, and light gray lines connecting handles to illustrate the Bezier structure. The viewer also supports live reloading of the glyph data, which will come in handy when we implement user editing of the Bezier curves.

Plan Progress

We did not have enough time to sync these two halves up together fully, so the renderer is currently relying on an older version of the parser that was compiled with GCC rather than Emscripten. Also, these two pages were developed using HTML + CSS and not React, and our end goal is to have a project in React since it will be easier to develop quickly. Converting between the two should not be too difficult, but our adjusted goal for Week 3 will be to consolidate the two proof-of-concept codebases into a single React codebase which we can use for the final site. Rendering even using plain JavaScript appears to be quite fast, so it may not be necessary to do a heavy amount of optimization, which should save time if we want to continue to focus on stretch goals in Week 4.