Project Proposal

by Jonah Bedouch, Brandon Wong, Richard Villagomez, and Nicholas DePalma

2025-04-03

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

Project Description

Our goal is to create a website called BezierBox, which allows users to upload fonts through .ttf files, view each rasterized character within the font, edit the bezier curves for any letters, and save the resulting modified font out to a new .ttf file. Ideally, this could be a progressive web app that runs entirely offline, and uses a mix of WebGPU/WebAssembly and Service Workers to run entirely offline, allowing users to work with fonts locally without a network connection.

Our team consists of Brandon James Wong, Jonah Alex Bedouch, Nicholas DePalma, and Richard Villagomez.

Problem Description

Oftentimes you might find a font that you like but want to edit it to resolve some small imperfection. Unfortunately, there's no intuitive way of doing this. Fonts are fundamentally made up of bezier curves, and unpacking and modifying a .ttf file can be quite challenging. To resolve this, we intend to create a website which converts a .ttf file into an abstract JavaScript representation (by following the Apple Truetype Fonts documentation and unpacking what we need from each ttf file), then shows the user the set of glyphs that the font contains. When a user selects a glyph, it will show the full bezier curve that results in the rasterization, and allow users to modify the curves or add/remove curves in order to change the appearance of the glyph. Then, we save this modified glyph into our abstract representation, which we can convert back into a TrueType font when the user is done editing.

Goals and Deliverables

We want to create a tool that can:

- Load .tff files and write a Bezier curve rasterizer to render them on a webpage
- Implement an "editing" feature that allows the user to edit properties of letters of the font using Bezier curves
- "Write back" a new TTF file with the user-edited font

As mentioned, this tool should work locally without a network connection.

We hope to answer the following questions:

- What is the most intuitive way to write our Bezier curve rasterizer to work on .ttf files? What are some of the specific features of .ttf files that we need to consider?
- How can we "work backwards" from the implementation of .ttf files so that users are able to edit the bezier curves, as well as attributes such as per-character kerning?
- How do we "write back" to the .ttf file so that user-generated edits are exported properly? Are there any edge cases we might have to consider?

We plan to deliver a tool that can:

- Load and render fonts from .ttf files via a Bezier curve rasterizer
- Interact with and edit the properties of letters in the font with Bezier curves
- Save these edited fonts as new .ttf files

If we are ahead of schedule, we hope to deliver a tool that can (in order of interest):

- Utilize Image Tracing to allow a bitmap to be converted into a series of bezier curves to create characters from a raster input.
- Modify other important characteristics of a font (such as per-character kerning).
- Persist changes by storing to and loading from LocalStorage
- Add quality of life features like undo/redo button, and the ability to type unique sentences with your font before exporting to test how it looks.

Schedule:

Week 1

- Figure out exactly how to open / write to a .ttf file.
- Create initial mockups of the website we want to build in Figma.
- Research WebGPU and WebAssembly as well as bezier shape rendering techniques.

Week 2

- Get bezier shape rendering working onto a web canvas
- Have ttf upload / opening working, and figure out the internal JS model we are using to store fonts

Week 3

- Get font rendering working and add interactive components to web canvas.
  - Might have to optimize font rendering to work fast enough – our goal is real time. This could be accomplished with a WebGPU pipeline
- Have unpolished frontend working that updates the underlying font data representation

Week 4

- Polish website
- Attempt to add any additional features from our hopeful list that we have time for.
- Add support for writing back to a .ttf file

Resources:

- Next.js and TailwindCSS will be used for the website itself (alongside HTML, CSS, and JavaScript)
- We may use WebAssembly or WebGPU (or ideally both, to have a fallback from WebGPU) to allow us to write the font rasterization code in C++ and run it more quickly on the web.
- We found a 300 page book with a section called “Font Rasterization: The State of the Art” from 1994 which describes the math behind rasterizing Quadratic Bezier curve based fonts.
- The TrueType reference manual might be useful for learning how to extract information from / store information to a .ttf file.