Link to Summary Slides: Google Slides

Video part 1: Watch here

Video part 2: Watch here

Webpages link: https://vbansal-29.github.io/cs184_final/

Font Rendering Pipeline - Milestone Report

Team: Vishal Bansal, Varun Mittal, Clara Kim, Saatvik Billa

Project Proposal

Project Summary

We are building a font rendering pipeline that rasterizes high-quality, anti-aliased text from TrueType fonts using both CPU and GPU backends. Our system interprets vector font lines-defined by Bézier curves—and converts them into grayscale bitmaps suitable for display. The project investigates the trade-offs between traditional CPU rasterization and parallel GPU rendering in terms of both visual fidelity and performance.

What We've Accomplished

Thus far, we have made significant progress on the CPU pipeline. We began by parsing `.ttf` font files using `freetype-py`. By loading characters with `FT_LOAD_NO_BITMAP`, we ensured access to the raw vector glyph data, including outline points, contour indices, and on/off-curve flags. This allowed us to isolate the fundamental geometric structure of each glyph.

We implemented a robust Bézier path extractor to convert contours into polylines of either straight lines or quadratic curves. Our segmentation logic correctly handles TrueType’s implicit midpoints for consecutive off-curve tags, ensuring precise glyph reconstruction. For validation, we visualized the paths using matplotlib, which confirmed the correctness of our outline extraction pipeline across several glyphs, such as “S”, “A”, and “O”.

In parallel, we began developing our rasterization backend. It takes the Bézier paths and performs scanline-style filling. Though still in progress, our system currently generates recognizable grayscale bitmap images of glyphs and handles multiple contours and complex shapes. Anti-aliasing via supersampling is under integration and early tests show visible improvements in edge smoothness.

Preliminary Results

Our CPU renderer has successfully rasterized glyphs into filled shapes that visually align with expectations from FreeType. The contours are closed correctly, interior holes are preserved using winding rules, and curve sampling is dense enough to ensure shape fidelity.

Although anti-aliasing is still being tuned, we’ve begun to compare aliased vs. smoothed results using supersampling. Early results demonstrate that supersampling significantly improves visual clarity around curve edges, validating our approach to pixel coverage estimation. These results are promising and indicate that our renderer is on the right track toward high-quality output.

Reflections on Progress

Compared to our original schedule, we are largely on track. Week 1 goals, including TTF parsing and glyph outline extraction, were completed as planned. Week 2’s focus on curve evaluation and rasterization is nearly complete. The rasterizer functions correctly for basic fills, and anti-aliasing integration is ongoing.

The main challenge so far has been managing Bézier curve decoding from raw font data. Understanding how off-curve tags are used in TrueType fonts, especially in sequences of multiple control points, required careful debugging. Visualizing these outlines was instrumental in refining our understanding and confirming that our outline interpretation was accurate.

We've also developed a clearer sense of how to modularize our system—splitting the pipeline into parsing, path evaluation, and rasterization stages—making it easier to swap in GPU acceleration later on.

Updated Work Plan

For Week 3, we will finish integrating supersampling-based anti-aliasing into the CPU rasterizer and begin optimizing performance. This includes reducing redundant point-in-polygon tests and cleaning up the scanline fill algorithm to make it more efficient.

We will also begin work on the GPU-based rasterizer. Our plan is to use WebGL

with fragment shaders, or GLSL via OpenGL in C++, to evaluate whether pixels lie inside Bézier-defined regions. Initially, we'll triangulate glyph paths and render them via the GPU, then explore analytical approaches to coverage estimation. Our goal by the end of Week 3 is to have at least one glyph rendering via the GPU and be able to toggle between CPU and GPU outputs.

For Week 4, we’ll focus on integrating both paths into an interactive demo that allows users to input text and toggle between rendering modes. We will also explore kerning and layout as stretch goals. At the same time, we’ll collect benchmark data on rendering speed and visual quality for various text strings and font sizes, supporting a final comparison and discussion of trade-offs.

We may adjust GPU work further depending on anti-aliasing complexity, but our core goals—accurate CPU rasterization with GPU exploration—remain well within reach.