From Bitmap to Bezier: An Approach to Personal Font Generation

Evan Dvorak, Joel Feske, Sihan Lin
24-681 Computer Aided Design
April 30, 2015

In this paper, we outline a method by which a personal, digital typeface can be generated from a scanned image of a user's handwriting. From the scanned image, each glyph is segmented and thresholded to produce a binary bitmap image, to which we apply a routine that finds the object's outside and inside boundaries. Bezier curves are fitted to the boundary data using a scheme that recursively splits any spline that does not meet a user-defined tolerance parameter. Lastly, we provide an example of word processing with the personal font using Python to typeset a sentence into a VRML file.

Introduction

The members of this team shared an interest in representing natural, analog handwriting in a computer system. This will add a new dimension to the experience of writing content on a computer, lending a personal touch to an otherwise rigid and standardized medium.

Methods

The task was divided into three processes:

Process 1: detect and extract the outer and inner boundary edges of each character
Process 2: fit Bezier curves to the boundaries of each character
Process 3: generate a VRML file to demonstrate typesetting the font

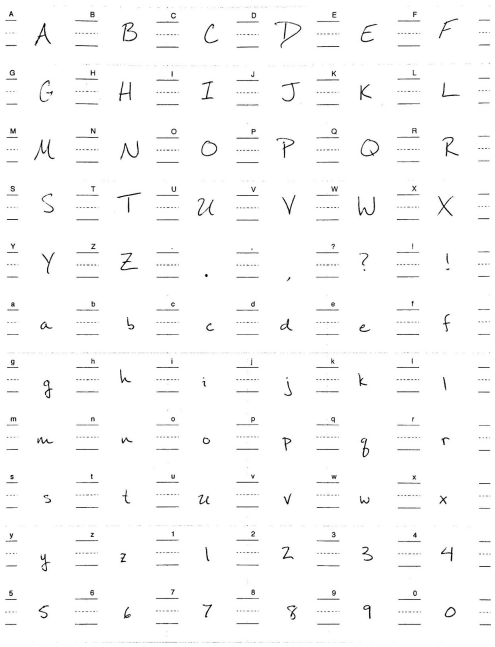

Fig. 1: Scanned image of template sheet containing upper- and lower-case Roman letters, the digits, and four symbols.

The first subtask in Process 1 was to separate each character from the template sheet. Next, each image was thresholded in order to generate a binary (black and white) bitmap image. MATLAB'S built-in function `bwboundaries` was used to trace along the glyph boundaries, producing arrays of boundary point coordinates. The last step in this process was to properly orient the object and hole boundary data for each character; object boundaries were oriented counter-clockwise, and hole boundaries were oriented clockwise. This feature allows the typesetting function to identify regions of the glyph to solid-fill.

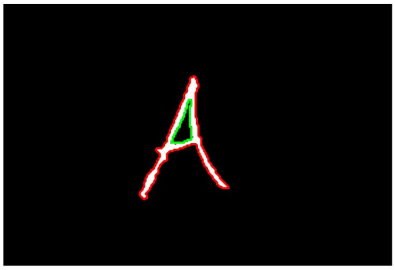

Fig. 2: Identified outer and inner boundaries for the character A

In Process 2, Bezier curves were fitted to the boundaries of each glyph. For each loop in a character, the curve was split in half. For each half, Bezier endpoints were chosen to be the endpoints of that half, and control points were chosen to equal the endpoints as a first guess.

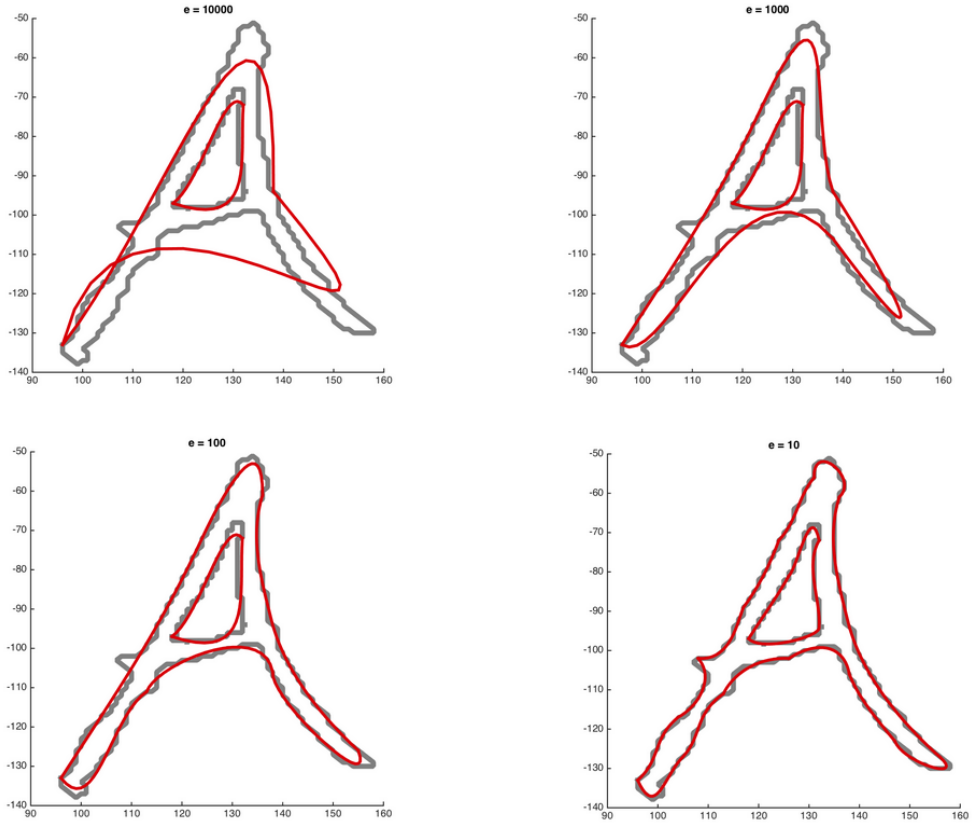

Fig. 3: Fitted curves with varying levels of error tolerance. Too high a tolerance results in a poor fit, while too low a tolerance results in overfitting.

The sum of the squared error at each point along the curve was calculated as a function of control point position, and that function was minimized using MATLAB's `fminunc` command, which performs an unconstrained optimization of a cost function. Unconstrained optimization was preferred because the unconstrained search space is convex. If the resulting optimal control points provided a curve which met the specified error threshold, the points were saved. If the error was too high, the fitting function was called recursively on the current half. Figure 3 shows the outcome of this process.

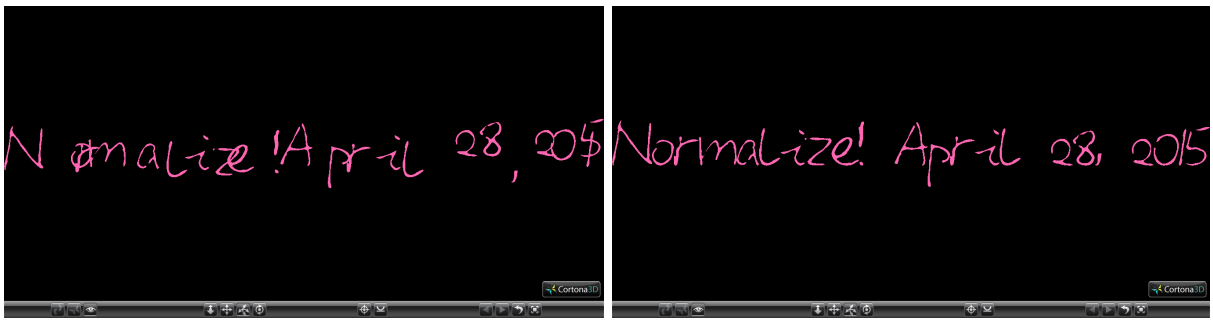

Fig. 4: Result of normalization of the letters

Finally, in Process 3, a personal font was created and displayed in a VRML file. To accomplish this task, a string and user-defined parameters such as letter spacing are input from the command line. For each character, the de Casteljau algorithm is used to produce point coordinates for each Bezier curve. To prevent overlapping or unnatural glyph placement, it was also necessary to normalize the horizontal position of each letter, number, and character (the positioning of each glyph on the template sheet has some random variation). Finally, each glyph is translated horizontally to form a sentence and the points are output as a VRML file. It is worth noting that in this step, if a letter has more than one hole (such as the lower case letter *g*), the extra holes should be separated as new shapes in the VRML file. Otherwise, the character would have an incorrect solid-fill.

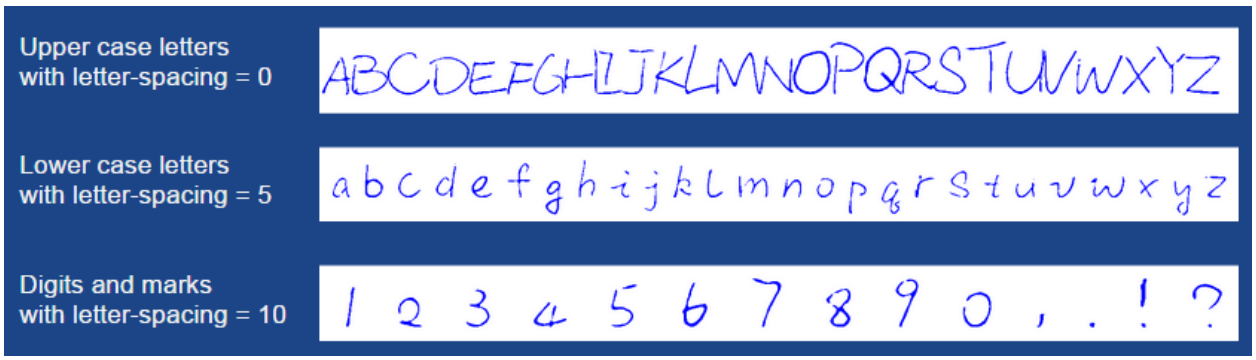

Fig. 5: Display of letters, digits and marks.

Results

We tested our code on handwriting from four users, and were able to generate text that subjectively resembled the original handwriting. Figure 6 demonstrates typesetting the string *My heart is in the work* using the font displayed in figure 5.

Fig. 6: Andrew Carnegie's well-known quotation, typeset by our word processing program with Sihan's converted handwriting.

From figure 6, some artifacts can be found in the glyphs, such as the discontinuities in letters *y*, *a*, and *s*. However, these artifacts are caused by the way the pen deposited ink, and are not a result of our processing routine; hence, they should be considered as part of the nature of handwriting and kept as-is. Another flaw is the unnatural horizontal spacing between some of the characters, which is caused by the monospaced typesetting. Horizontal spacing could be improved by explicitly handling glyph kerning.

Discussion

The process described in this paper is a general approach to the problem of generating a custom, handwritten digital typeface. In order to demonstrate the main components of our approach (boundary extraction, recursive Bezier curve fitting, and the use of de Casteljau algorithm), we constrained the problem in a few ways: we limited the number of non-letter symbols and characters, and do not handle character kerning, connected letters (as in cursive script, discussed in Figure 7), diacritical marks, or boldface font. Additionally, we did not generate a digital typeface in a common digital font format, but rather produced our own rudimentary word processor program using Python to output VRML files; this decision was made in order to demonstrate the use of de Casteljau's algorithm and avoid the complexity of coding common digital font files such as TrueType and OpenType.

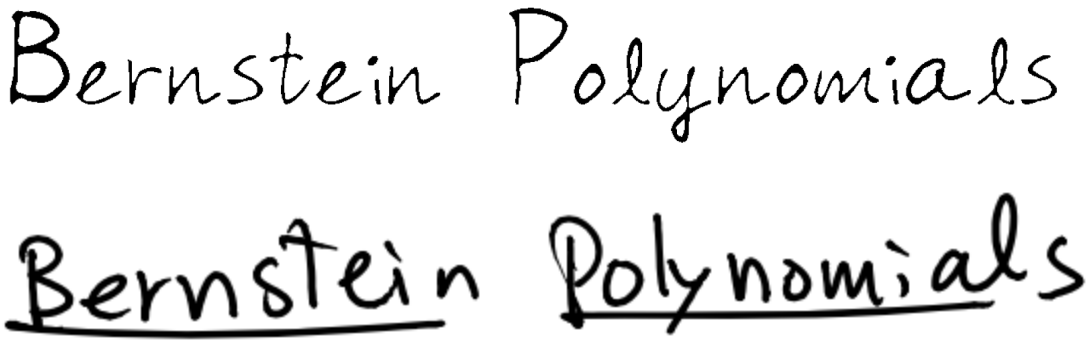

Fig. 7: Comparing our typeset custom font (top) to a sample from the same user's actual handwriting (bottom). Whereas the writer may connect glyphs when writing by hand, our current method does not handle ligatures.

The error threshold chosen when fitting curves is somewhat arbitrary, and the threshold producing the best results will vary depending on the handwriting being mimicked, the specific letter, and personal preference. The user may chose a higher error tolerance if a smoother font is desired, or a lower tolerance for a more accurate rendering of the imperfections of natural handwriting.

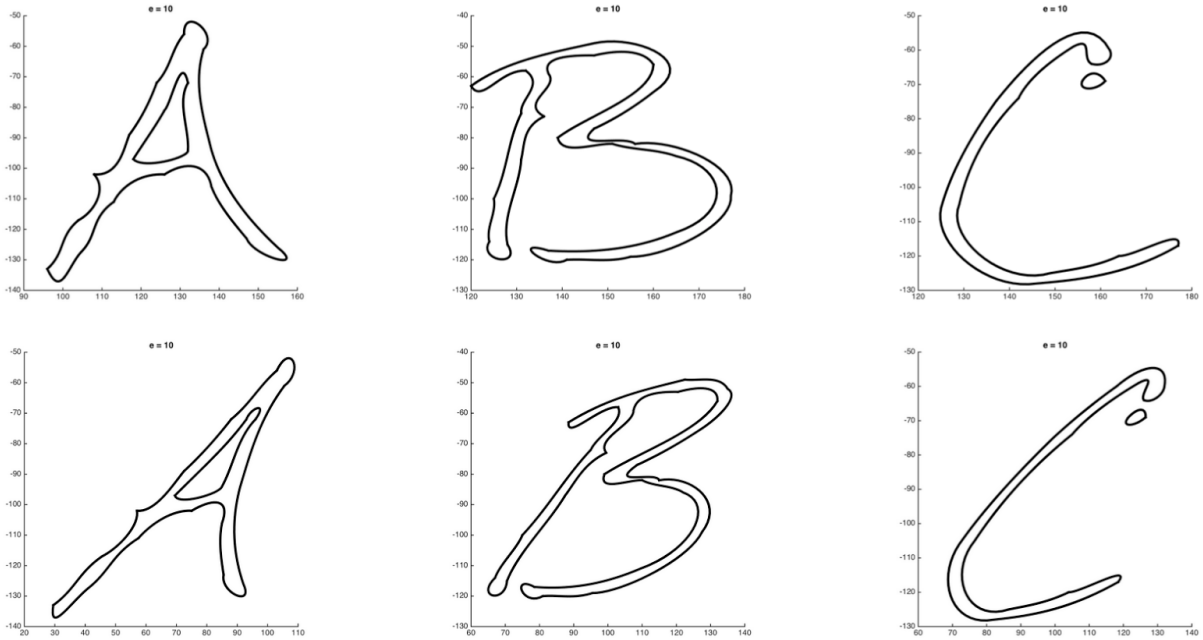

Fig. 8: A shear transformation is used to create an italicized version of the font

Representation of the typeface as sets of Bezier splines allows for easy transformation of the letters. Figure 8 shows the result of a simple shear transformation used to create an italicized version of the personalized font. In this way, families of typefaces sharing the general æsthetic of the user's handwriting could be created from a single sample.

Future work may involve incorporating the more advanced typeface parameters discussed in this section, and outputting to a common digital font format for use in general computing applications.