Typography Day 2014
- Typography and Culture

Method to create personalized fonts based on a person's handwriting

Dhvanil Patel, Indian Institute of Technology Guwahati, India, dhvanilpatel2012@gmail.com
Sanuj Sharma, Indian Institute of Technology Guwahati, India, sanuj.sharma.in@gmail.com

Abstract: In this paper, we illustrate in detail, a method which can be efficiently used to convert the handwriting of a person into a true type font. A program was also created using open-source software which performs this function in a very simple way. The person is made to fill in a sheet of paper which is then scanned at high resolution. Afterwards, the program automatically extracts the letters from it and compiles them into a true type font. Furthermore, the paper also illustrates various ways in which this process can be improved. Randomized characters and alternate input characters are some of the approached suggested in this paper.

Key words: Personalized font, Latin script, Alternate characters, Randomized alphabets.

1. Introduction

Typefaces today are very dull and monotonous. They are highly repetitive. We can clearly identify the difference between handwritten text and typed/printed text. Although certain scripting fonts do contain alternate characters and ligatures but even they differ greatly from handwritten text.

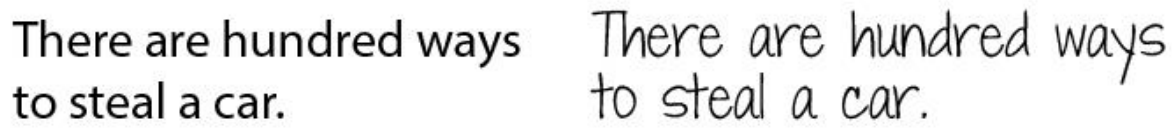

Figure 1. Typed text versus handwritten text

There is a need to create personalized fonts which could be used by common people. These personalized fonts would try to bridge the gap between the printed media and the handwritten media. This would help the people to increase the feeling of personalization in typed media.

In this paper, we explain in detail, the script which could enable anyone to make his/her own font which could then be used by him to personalize any written media. The script has many features which makes it a lot better than most handwriting this could be used in personal letters, invitation letters, digital signatures, etc.

2. Method

The following approach was used by us in order to create the personalized font. We tried to create a method which was as simple as possible, and which required very little technical expertise from the user. Although, the method is illustrated in various steps, a Shell script was developed which linked all these programs seamlessly. No additional expertise is required from the user. The machine performs all the functions on its own.

2.1 Taking in the handwriting sample from the person.

This is the first step of the process, in which a person is given an A4 paper, called the input sheet, in which he writes the alphabets, numbers and special characters in his own handwriting in predefined boxes. Please refer to the picture below for the format of the input sheet. This particular format of the sheet has been decided upon after a lot of user trials and computer testing.

Various criteria were taken into account:

1. The size of the boxes, which was directly related to the easiness with which the users could write the characters.
2. The thickness and the overall arrangement of the boxes, which was related with the code which was used to extract the characters from the scanned image. The working and the algorithm of the code has been explained in the next section.
3. The simplicity of the paper, which would improve the user-friendliness of the entire process. Various features were added to the original format on account of this criterion.

Since the terminals of the handwritten characters do not have consistency, four alternatives for each character were taken. This increases the possibility of smooth fitting of cursive characters. The end and the beginning of the characters must coincide, more or less.

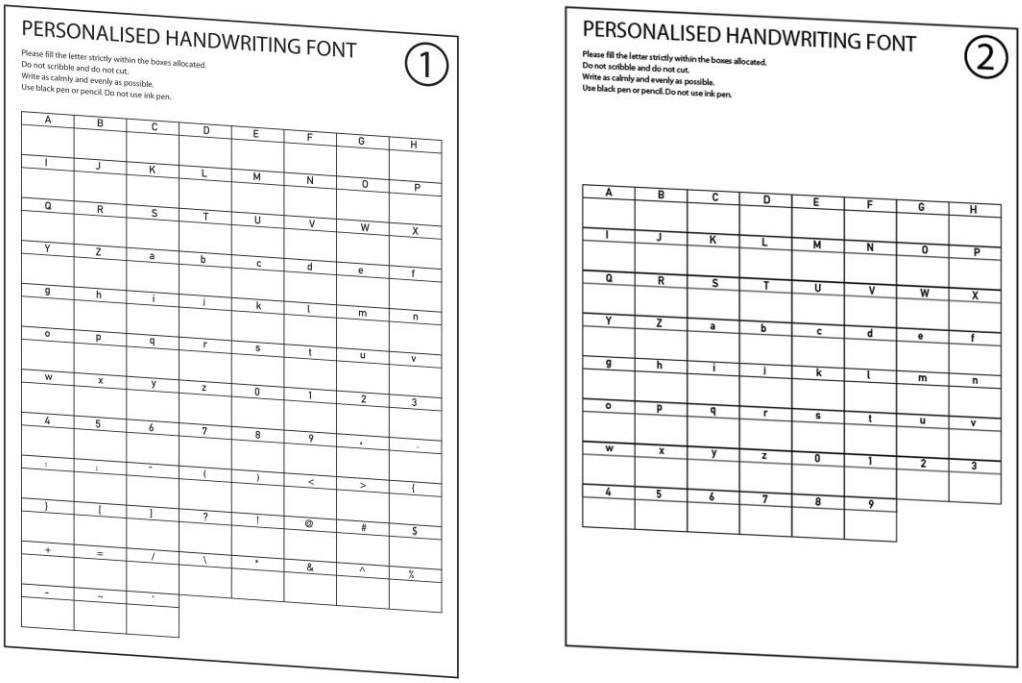

Figure 2. Input sheet with alternate characters

2.2 Scanning the sheet.

The input sheet are scanned with a normal desktop scanner. Various scan resolutions were tested on the code and finally, 600 dpi resolution was selected. The dimensions of the scanned image were about 2300 pixels by 1600 pixels. Sometimes, the more than one sheets were scanned to include the feature of alternate characters. Having multiple input sheets highly increases the likeliness of the person's handwriting with the developed font. This feature is explained in detail in the next section. The code also worked well at resolution of 150 dpi, although the quality was not up to the mark.

2.3 Extracting the glyphs.

The code then, performed image processing functions on the scanned image and recognized the boxes which housed the characters provided by the user. The algorithm and its functioning is explained in detail in the next section. After that, the boxes were cropped and saved into separate png files. Each image is named after the UNI code of the character housed within it.

2.4 Creating Scalable Vector Files (SVGs) from the cropped images.

Since font creating software do not accept direct bitmap images, we needed to convert individual bitmap images into SVG files which could then be imported into FontForge. An open source software, Potrace was used for this vectorization process after tweaking it slightly to suit our needs. Also, the characters were slightly shifted so that the spacing between them isn't affected when the font is finally made.

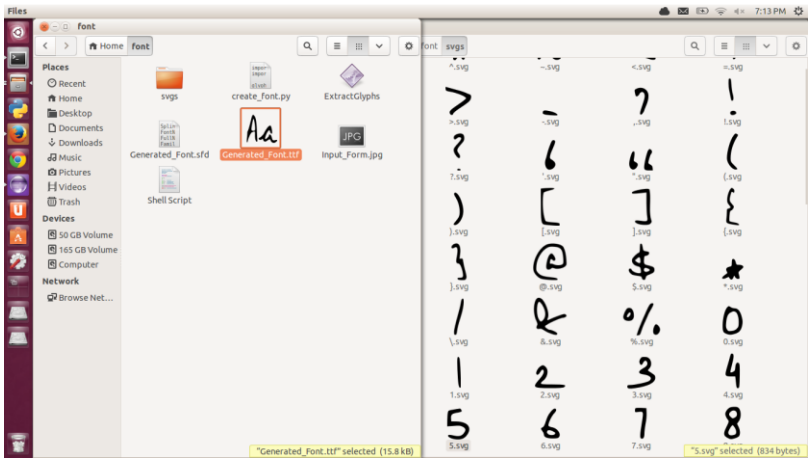

Figure 3. Python script creating SVGs from bitmap images.

2.5 Importing SVGs into FontForge.

After the SVGs were created by Potrace, they were imported into FontForge for the final stage of our font creation process [1] [2]. A python script was written which automatically imported the SVGs into the relevant glyphs. This could easily be done because the bitmaps were named according to the UNI code of the character they housed.

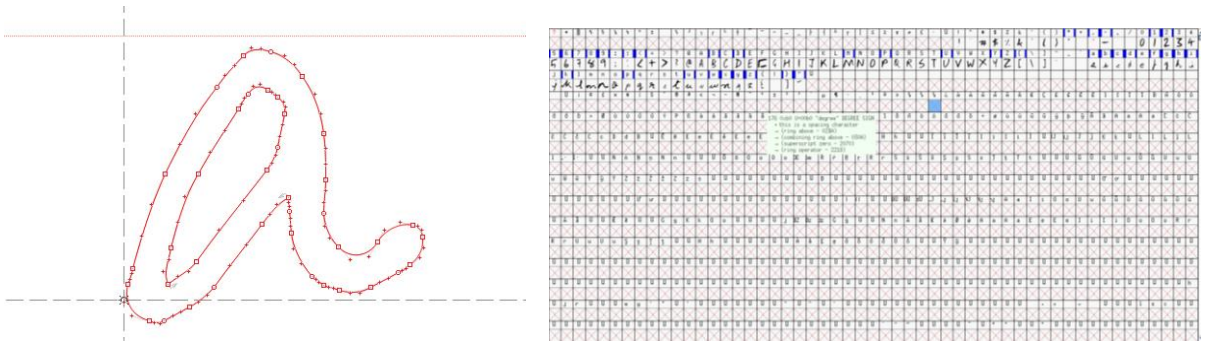

Figure 4. SVG for the glyph 'a' imported in FontForge and one character set of the font in FontForge.

2.6 Processing the font.

After the SVG files are imported into FontForge, the font was auto-spaced and auto-kerned using the in-built functions of FontForge. These functions were called via a script which then generated the whole font.

3. Algorithm

The algorithm (ExtractGlyphs.cpp) is based on another algorithm called squares.cpp which is distributed as a sample with OpenCV (Open Source Computer Vision Library) [3]. Squares.cpp returns an array of the coordinates and dimensions of the rectangles identified. It uses Canny (for edge detection) and findContours (to find contours). After finding all the edges it compares the angle by finding respective cosine values and discards those which are greater than 0.2 (threshold value). Once I get all the rectangles a function filters out redundant rectangles according to their areas and coordinates. Then sub-images are generated, in turn extracting all the rectangles which contain the glyphs from the input form filled by the user. To extract the glyph from the sub-images, some functions are used which bound the glyph by a rectangle. The generated ".bmp" files are converted into ".svg" by using a command line utility called Potrace. It uses the parametric cubic algorithm suggested by J. Gonczarowski [4]. The vectored glyphs are then pushed into FontForge by using a python script to get a truetype font file. The whole process is automated by a shell script which takes the responsibility of calling IdentifySquares, Potrace, and FontForge script respectively.

4. Results

About ten personalized fonts were created using this technique, with no manual intervention. All the fonts bore very high resemblance to the actual handwriting of the person. The use of alternate characters made it difficult to recognize it as a typeface. It brought in an element of randomness which is normally associated with handwritten media. However, in case of cursive letters, there was some visual discontinuity in the connectors of the small letters. However, in the case of manuscript type of writing, the script worked perfectly and automatically adjusted the character-spacing and kerning.

Figure 5. Alternate characters for the letter 'A'

5. Drawbacks

There were about five specific problems we identified with our model. These are explained below:

1. Discontinuity in the connectors.
   The connectors in the longhand handwriting style appeared sloppy at times. Since the handwritten letters are very irregular and the connector height remains very inconsistent, joining the connectors together for all the glyphs was not possible. Although this problem was, to some extent, taken care of by the incorporation of alternate characters.
2. Cumbersomeness of alternate characters.
   Except for graphic designers, very few people actually know about alternate characters and their application. And since commonly used text editors do not have a random select contextual alternatives option, the user will have to manually select the alternate glyphs from the extended glyphs menu.
3. Baseline inconsistency.
   The final font had some inconsistencies in the height of baseline which made it look slightly haphazard. The inconsistencies were particularly evident in the case of descending letters. This is because of the nature of the code and the approximations used.
4. Uneven thickness of the strokes.
   The ink sloshes on the paper and the uneven impressions left while writing lead to irregular thickness of the strokes sometimes. This happens mostly in lower case alphabets since they have too many curves.

6. Future Scope

There is immense scope for further improvisation of the program so that the gap between typed and handwritten media can be reduced even further. Some of the key points of interest are explained below:

1. Variable character spacing and kerning.
   One of the main features of handwritten text is that the character as well as the word spacing is never constant. Using Extensible Application Markup Language (XAML) [4], one can easily integrate a randomized functions for alternate glyphs as well as variable spacing.
2. Variable baseline.
   Very rarely do people have a consistent baseline in their handwriting. Although individual words have a fixed baseline, in common writing our baseline varies considerably, especially while writing on unruled sheet of paper [5].
3. Random characters.
   There are various algorithms which create random character shapes from one shape. This can be done by random perturbation of Bezier control points [6]. The degree of randomness can also be changed according.

7. Conclusion

This project successfully illustrates that fonts, which have been considered mechanical and morbid since the invention of mass printing, can now be converted into something less machine like, something more personalized. These personalized fonts will make the task of humans easier in many ways. Computer voice synthesizers are becoming more and more human, Computer programs are becoming more and more human, and it is time fonts too became more personalized, more human like.

8. References

[1] http://fontforge.org/python.html

[2] http://potrace.sourceforge.net/faq.html

[3] Canny, J. (1986) A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, (6), pp 679-698.

[4] Gonczarowski, J. (1991) A fast approach to auto-tracing (with parametric cubics), in Raster Imaging and Digital Typography II, Morris R. A. and J. Andre, eds., Cambridge University Press, 1991.

[5] http://msdn.microsoft.com/en-us/library/ms745109(v=vs.110).aspx

[6] André, J., & Vatton, I. (1994) Dynamic optical scaling and variable-sized characters. *Electronic Publishing*, *7*(4), pp 231-250.

[7] Devroye, L., & McDougall, M. (1995) Random fonts for the simulation of handwriting. Electronic Publishing (EP—ODD), 8, pp 281-294.