Zpravodaj Československého sdružení uživatelů TeXu

Timothy Eyre

Creating a Kanji Stroke Order Font

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 20 (2010), No. 3, 199–207

Persistent URL: http://dml.cz/dmlcz/150119

Jak na výrobu písma kandži s pořadím tahů

Timothy Eyre

Abstrakt

Tahy u každého jednotlivého kandži by měly být psány v přesně daném pořadí. Projekt Aaaa (A亜あ) Ulricha Apela sestavil sadu SVG souborů, které obsahují informace o tazích a jejich pořadí. Díky tomu byl autor článku schopen sestavit písmo tak, že zobrazuje správné pořadí tahů u 6373 kandži. Vzniklé písmo usnadňuje přípravu studijních pomůcek, které tuto informaci využívají.

Klíčová slova: japonština, kandži, pořadí tahů, tvorba písma, formát TrueType, formát SVG, program FontForge.

doi: 10.5300/2010-3/199

Introduction

The strokes of each Japanese character (kanji) should be written in a certain order. This correct stroke order is called *hitsujun* (筆順) in Japanese and helps with legibility and memorization. Students of kanji are examined on kanji stroke order in exams such as the Japanese Kanji Aptitude Test (日本漢字能力検定試験).

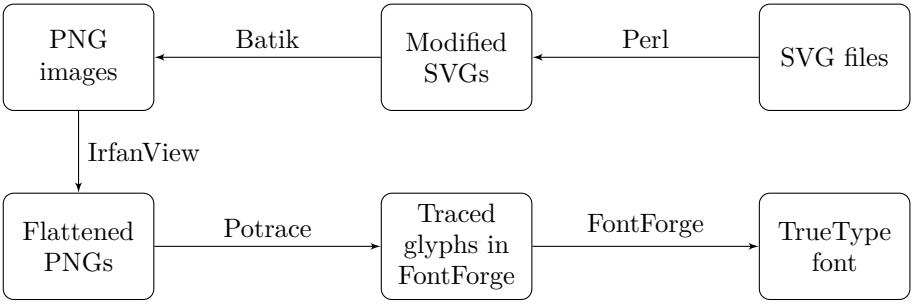
Although there are some general rules about the order in which the strokes of a kanji should be written, it is useful to have a reference available. There are indeed numerous printed and electronic kanji stroke order resources available. However, the coverage of these resources is not always as comprehensive as one might like and the presentation is not always as helpful as it might be. Various means of presenting of kanji stroke order diagrams appear. Step-by-step diagrams are popular in printed resources, such as [1] and animated diagrams are popular in electronic resources such as [2].

Dr. Ulrich Apel and Dr. Julien Quint have produced a large set of SVG files [3], [4] that present the stroke order of several thousand kanji in the static form used in [4]. This impressive data resource certainly excelled as a means of storing the information but retrieval required the use of the A亜あ(Aaaa)-project's web interface and Adobe's SVG browser plugin.

Dr. Apel was kind enough to send me the set of SVG files. A few years earlier it had crossed my mind that a simple way to present kanji stroke order diagrams would be to build them into a Unicode font. The SVG files from the A亜あ-project provided me with the data I needed to create such a font. This paper describes how I converted the SVG data into the Kanji Stroke Order Font and how I maintain the font now it exists.

1. Method of Creation

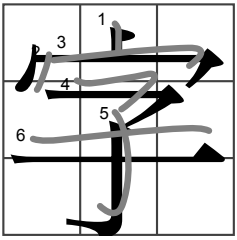
The following flowchart summarizes the steps involved in creating the Kanji Stroke Order Font.



The subsections below describe each of these steps in detail.

1.1. Modifying the SVG files

The original SVG files contained graphical information that was not necessary for the font. Here is an example of a raw SVG image:



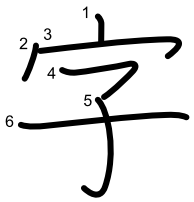
As you can see, while the pen stroke version of the kanji and numbers are required for the font, the grid and printed version of the kanji are not. By means of a Perl script, I processed the contents of each SVG file to remove the superfluous elements and change the colour of the pen strokes from grey to black. The fact that SVG is an XML-based file format and therefore plain text made this processing simple: I eliminated the large print kanji simply by changing its font size from 108 to 0, changed the colour of the grid from dark grey to white and the colour of the pen strokes from mid grey to black. A simplified version of the Perl script looks like this:

```
while ($line = <STDIN>)
{
  $line =~ s/#808080/#000000/;
  $line =~ s/#404040/#FFFFFF/;
  $line =~ s/font-size:108/font-size:0/;
  print $line;
}
```

The `while` loop reads the standard input until the last line of the redirected file is reached. The three `$line =~ s/foo/bar/;` lines substitute `foo` for `bar`. This means, for example, that `$line =~ s/#808080/#000000/;` replaces light grey with black. This script would be executed as follows:

```
perl preproc.pl < infile.svg > outfile.svg
```

However, in practice we are handling several thousand files so some extra Perl globbing code is needed. After this simple processing, the SVG image is more suitable for inclusion in a font, looking as it does like this:



1.2. Conversion to PNG using Batik

The Batik SVG Toolkit [6] is a Java library that can be used to render SVG graphics. Rendering the several thousand SVG files created in the previous step simply required multiple executions of commands like the following:

```
java -jar batik-rasterizer.jar -d . -m image/png
                -h 2000 -dpi 2000 outfile.svg -bg 255.255.255.255
```

The parameters for this command are explained as follows:

- The parameter `-d .` simply specifies that the local directory should be used for the output.
- The parameter `-m image/png` is a MIME specification, indicating that the output should be PNG rather than any of the other formats that Batik can produce.
- The parameter `-h 2000` specifies the height of the output in pixels.
- As a result of the previous parameter, the parameter `-dpi 2000` ensures that the resulting PNG image is exactly one inch tall.

- The parameter `outfile.svg` specifies the SVG file to be rendered; it is called `outfile.svg` here because it is the output from the step described in section 1.1.
- The parameter `-bg 255.255.255.255` specifies the background fill colour, including the alpha (transparency) parameter.

Batik renders the SVG files relatively slowly, so rendering all of them requires an overnight run. At the end of this process we will have several thousand PNG files, one for each character that is going to go into the font.

1.3. Flattening the PNGs

Batik produces 32-bit PNG files. This generous image depth has two disadvantages for our purpose here. Firstly, it makes the PNG files needlessly large, an important consideration when there are literally several thousand of them. Secondly, Batik uses subpixel rendering (the technique used by CoolType and ClearType) to improve the appearence of the output. The grey fuzz around each kanji looks pleasing to the human eye but gives extra work to Potrace. Therefore, it is desirable to reduce the number of bitplanes in the PNG files to 1. The IrfanView [7] image viewing and processing utility has a GUI tool that makes this bitplane reduction straightforward, albeit difficult to document.

1.4. Tracing the PNGs

At last we are ready to start using FontForge [8], the BSD-licensed outline font editor that runs on Linux, Mac and even Microsoft Windows with the help of Cygwin [9]. FontForge includes a trace facility, which calls out to the Potrace [10] bitmap-to-vector tracing tool under the covers. FontForge can also use Autotrace [11] to perform the same function.

When doing bulk tracing into a Unicode font, FontForge requires the filename of the source image file to be of the form `uniXXXX.png`, where `XXXX` represents the 4-digit hexadecimal number corresponding to the Unicode number for the character. This fits in well with our workflow so far.

Invoking the trace feature only requires two menu option selections in FontForge. The first option is File→Import with the Format option set to 'Image Template'. This creates a background image for each of the characters. We then tell FontForge to trace these background images by selecting the option Element→Autotrace.

The tracing process takes some hours and uses copious resource on the host machine. One of the challenges I encountered in this project was finding a PC that was powerful enough to handle this task.

Once the tracing is complete the font can be exported from FontForge in whatever format you choose. I distribute the Kanji Stoke Order Font in the TrueType format.

2. Polishing the Font

As with all real-life projects, the font created by the workflow above is imperfect. The following subsections describe the polishing that I did to produce a usable font. At no stage in the creation of the font did I check each glyph for the accuracy of the stroke orders; with several thousand characters this was not practical.

2.1. Character Sizes

The most obvious glitch was that some of the kanji were too large in comparison with the other characters in the font. Correcting the rogue characters was simply a matter of invoking a scaling transformation for the character in FontForge. Finding the rogue characters was more of a challenge.

I tracked down the oversized characters in two ways, both using XeTeX and a list of the codepoints covered by the font.

The first chunk of XeTeX code prints the height of each character to screen, allowing for redirection to file, sorting and subsequent identification of outliers.

```
\font\cf="KanjiStrokeOrders" at 40pt
\def\boxit#1{\hbox{\vbox{\hrule\hbox%
  {\vrule\vbox{#1}\vrule}\hrule}}}
\def\charfont#1{\setbox0=\hbox{\boxit{\hbox{\cf#1}}}%
                \immediate\write15{\the\ht0#1}}
\input test_data.tex\bye
```

The file `test_data.tex` is simple, containing just several thousand lines of the following form:

```
...
\charfont{\char"9051}{9051}%
\charfont{\char"9052}{9052}%
\charfont{\char"9053}{9053}%
...
```

The second chunk of XeTeX code prints the characters themselves, which enabled me to view the characters in bulk and identify ones that looked wrong.

```
\font\cf="KanjiStrokeOrders" at 40pt
\nopagenumbers\pdfpagewidth=331mm\pdfpageheight=207mm
\hsize=300mm\vsize=200mm
\parindent=0pt\hoffset=-0.5in\voffset=-0.9in\baselineskip=0pt
\def\boxit#1{\hbox{\vbox{\hrule\hbox%
  {\vrule\vbox{#1}\vrule}\hrule}}}
\def\charfont#1#2{\setbox0=\hbox{\boxit{\hbox{\cf#1}}}\box0\ }
\input test_data.tex\bye
```

Figure 1: Kanji Stroke Order Font test page

Figure 1 shows a typical page generated by this XeTeX fragment. There are 28 test pages in total, which means that looking through them for mis-formatted characters is a tractable task.

2.2. Making the character widths consistent

All Japanese kanji notionally occupy a square of constant width. Therefore it was necessary to adjust the width of the characters to all be the same. FontForge provides a scripting language, which makes tasks like this easy.

```
SelectWorthOutputting()
SelectFewer(0u0000,0u2e8b) # Remove non-Japanese chars
SelectFewer(0uFF61,0uFF9F) # Omit half-width katakana
foreach
  Print ("Doing next Jp ", GlyphInfo("Unicode"))
  SetWidth(1024);
  CenterInWidth();
endloop
```

Note the line of code that omits resizing the half-width katakana. These are a relic from the early days of Japanese-language computing [12]. I use a similar block of code with a call to `SetWidth(512)` to set the width of these characters.

2.3. Cleaning up the outlines

An effect of tracing the characters from image files is a surplus of points in the character outlines. FontForge has a simplification function that cuts down the number of points, thereby reducing the file size and improving maintainability.

FontForge also has a validation function, which enabled me to clean up some problems with the TrueType font. However, so powerful is FontForge's validation feature that it was not practical to fix every validation error.

2.4. Add missing characters

The source SVG dataset omitted some characters that are commonly used but not relevant to stroke order analysis. To make the Kanji Stroke Order Font more usable I merged it with another font to fill in the gaps; FontForge has a built-in feature for doing exactly this. Originally I used a public domain font called Tuffy [13] but once I had created the Unicode font Choumei [14] I used that to fill in the gaps in the Kanji Stroke Order Font. I describe the Choumei font in a separate section below.

Creation of the Kanji Stroke Order Font was now complete so I was able to publish it on my website [15].

3. Maintenance

Maintaining the font is fiddly but simple. I rely on reports from users to track down errors in the stroke order diagrams. When a user reports an error I verify that the proposed revision to the stroke order is correct and then use FontForge to change the affected character. The stroke order numbers are stored as shapes in the font, with no references or underlying structure. This means that the only way of changing the order of the stroke numbers is to cut and paste them.

At the time of writing, the current version of the Kanji Stroke Order Font is v2.014. This contains 146 corrections to the initial revision. By the time this paper is published I shall have produced v2.015 of the Kanji Stroke Order Font, which will contain several more corrections.

User feedback is important for the Kanji Stroke Order Font: it is not practical for me to check all the glyphs myself and most of the users of the font are undoubtedly more skilled in kanji than I am. Moreover, accuracy is important because the ease of use and broad coverage of the Kanji Stroke Order Font means there is a risk that some users might treat it as being authoritative, which it most certainly is not.

4. Future Possibilities

Dr. Apel has moved on to create the KanjiVG project [16], which uses a defined SVG format to store information about drawing kanji. It would be theoretically possible to automatically create the Kanji Stroke Order Font from this data using a script based on the technologies described in this paper.

A major advantage of this approach would be that the two projects could be kept synchronized and corrections to either would be picked up by both. My experience with creating the Kanji Stroke Order Font showed that font creation requires a significant amount of manual tweaking, so it is not clear that such a process would work in practice. This is an area for future study.

5. Choumei Unicode font

Free (as in speech or beer) fonts that cover a large number of kanji are relatively few in number. This is to be expected because creating a quality kanji font requires a large amount of effort. However, with the source data for the Kanji Stroke Order Font already to hand, it was comparatively easy to create a comprehensive kanji font (albeit one of dubious quality) by omitting the stroke order numbers from the glyphs in the Kanji Stroke Order Font. I achieved this by running through the Kanji Stroke Order Font workflow steps described in this paper with a single change: I set the font size of the kanji stroke numbers to zero in the Perl script at the SVG modification stage. Thus, with minimal additional effort, I was able to publish a free kanji font [14]. Given its simple appearence, I named it Choumei for Kamo no Choumei (鴨長明, 1155?–1216), author of the essay *Life in a Ten Foot Square Hut*.

Having created this initial draft of the Choumei font, I used FontForge to add to its glyph compliment. I did this mostly by cutting, pasting and modifying existing glyphs rather than drawing glyphs afresh. The characters I added were mostly symbols and accented Latin letters. Having expanded Choumei significantly, I merged it with the Kanji Stroke Order Font to expand the latter font's glyph coverage, albeit without stroke numbers.

I now treat the Choumei font as unmaintained.

Conclusion

A font can be generated from SVG data using a workflow, each step of which uses free (as in speech or beer or both) tools. Each of these steps is scalable and therefore it is practical to use the workflow for a set of several thousand SVG files. This scalability made it possible to convert the A亜アあ-project's kanji stroke order SVG files into a TrueType font.

References

[1] Halpern, Jack: Kôdansha Kanji Learner's Dictionary, Kôdansha International, December 2001. ISBN 978-4-770-02855-6.

[2] Kanji Café website: `http://www.kanjicafe.com/`

[3] Quint, Julien; Apel, Ulrich: *Does Learning How to Read Japanese Have to Be So Difficult And Can the Web Help?*, Proceedings of the WWW 2005 conference, Chiba, Japan, May 2005. ISBN 1-59593-052-3. URL: `http://www2005.org/cdrom/docs/p1152.pdf`

[4] Quint, Julien; Apel, Ulrich: *Teaching and Reference Material on Japanese Kanji in SVG Stroke Order, Animated Drawings of Characters, Kanji Components and their relationships.* Proceedings of the SVG Open 2004 conference, Tokyo, Japan, September 2004. URL: `http://www.svgopen.org/2004/papers/svgopen/`

[5] O'Neill, P. G.: *Essential Kanji: 2,000 Basic Japanese Characters Systematically Arranged for Learning and Reference*, Weatherhill Inc., 1 Jan 1974. ISBN 978-0-834-80222-3.

[6] Batik SVG Toolkit: `http://xmlgraphics.apache.org/batik/`

[7] IrfanView: `http://www.irfanview.com/`

[8] FontForge: `http://fontforge.sourceforge.net/`

[9] Cygwin: `http://www.cygwin.com/`

[10] Potrace: `http://potrace.sourceforge.net/`

[11] Autotrace: `http://autotrace.sourceforge.net/`

[12] Hensch, Kurt: Research and Development in IBM, History of Far Eastern Languages in Computing, 2nd private edition, Roehm TYPOfactory GmbH, Sindelfingen, Germany, 2004. ISBN 3-937267-03-4.

[13] Tuffy font: `http://tulrich.com/fonts/`

[14] Choumei Unicode Font: `http://www.nihilist.org.uk/`

[15] Kanji Stroke Order Font: `http://www.nihilist.org.uk/`

[16] KanjiVG project: `http://kanjivg.tagaini.net/`

Summary: Creating a Kanji Stroke Order Font

This article describes how a font that displays Kanji Stroke Orders can be created from thousands of SVG files containing this information.

Keywords: Kanji, Stroke Order, Font, TrueType, SVG, FontForge.

Timothy Eyre, `mail@nihilist.org.uk`
CSTUG c/o FEL ČVUT, Technická 2
Prague, CZ-166 27, Czech Republic