Fonts

Making outline fonts from bitmap images

Karl Berry

Abstract

Tools for creating outline fonts from bitmaps have matured significantly in past years. Our purpose here is to describe those tools and how they fit together in a TeX context, in a practical way.

1 Introduction

I recently had occasion to create outline fonts (PostScript Type 1 or TrueType; see [24]) from a scanned bitmap image of a type specimen. I was pleasantly surprised to find that the tools for doing this conversion produce considerably better results than the last time I worked in this area. This note describes one procedure for putting the programs together, culminating in using the result from TeX.

Happily, all of the programs mentioned here can be compiled and installed with only minor variations on the standard procedure [18] first specified for the GNU project:

```
configure && make && make install
```

Web addresses for the programs are given in the references.

We should at least touch on legal questions [4], although a thorough discussion is far beyond the scope of this paper—or my knowledge, for that matter. My understanding is that font designs, as opposed to font programs, are still not copyrightable in the United States (a few have been patented, notably Lucida), but that designs are protected in most European countries. As a result, in today's world of widespread file sharing, especially in the TEX community, it would be unwise to attempt to create or distribute fonts for any design created after approximately the early 1900's, without specific knowledge for a specific design.

On the other hand, I did receive legal advice (from the Free Software Foundation's lawyer) that scanning old type specimen images, even when they are embedded in a book still under copyright, is defensible. Not any of the text or illustrations prepared specifically for the book, of course, but actual old specimens may be copied.

2 Scanned image to bitmap font: imageto

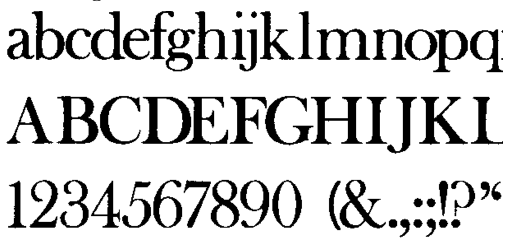
For our purposes, we will start with a single black and white image of a font specimen of a Baskerville type at a fairly large size (24 pt), scanned at a fairly high resolution (1200 dpi). The image includes the upper and lowercase alphabets, digits, and other principal characters. The first task is to extract these characters from the image into a bitmap font.

Not coincidentally, one of the programs in the GNU font utilities [3], written a decade or so ago by Kathryn Hargreaves and myself, does precisely this. This program is called `imageto`. There may be other programs to accomplish the same task, but since I knew about this one (for obvious reasons), I just used it.

The output from the scanner (a Xerox 9700) is in an unusual image format that can't be read directly by any modern program. (The scanning was also done a decade ago.) So, to see the image I was working with, I converted it to Encapsulated PostScript (EPSF [24]) and viewed it with `gv` [13], at its smallest scale factor (0.1):

```
imageto --epsf gbvr
gv gbvr.eps
```

Here's the resulting picture of the starting image (clipped to approximately the left half due to the small *TUGboat* column width), so we can see what we're dealing with:



The font name `gbvr`, by the way, stands for GNU Baskerville roman, according to the Fontname scheme [2]. (This whole project started under the aegis of GNU [19].)

`imageto` considers the input image as a series of 'image rows'. Each image row consists of all the scanlines between a nonblank scanline and the next entirely blank scanline. (A 'scanline' is a single horizontal row of pixels in the image.) Within each image row, `imageto` looks top-to-bottom, left-to-right, for 'bounding boxes': closed contours, i.e., an area whose edge you can trace with a pencil without lifting it. For example, an 'i' has two bounding boxes, while an 'a' has one.

In practice, scanned images have plenty of imperfections; for instance, a small printing blotch is seen as a bounding box which we have to ignore. Baselines jump up and down due to the printing process, as well as the natural baseline adjustments for characters with descenders or o-corrections (curved characters such as 'o' whose bottom point is slightly below the baseline). So we have to describe all of these special cases.

To extract characters from the bitmap, it's necessary to supply all of this descriptive information to `imageto` in a simple line-oriented text file, called an `ifi` file (image font information). The details of this file's syntax aren't important here — the full manual (and source code) for Fontutils are available at the url given in the references.

For the sake of example, the final command line to process this image ended up being:

```
imageto --designsize=24 \
 --encoding=8r \
 --baselines=72,59,58 \
 --print-guidelines \
 --print-clean-info \
 gbvr
mv gbvr24.1200gf gbvr.1200gf
```

The result is a bitmap font in GF format [9] (same as METAFONT). So, from `gbvr.img`, we now have `gbvr.1200gf`. We explicitly remove the 24 in the filename because we want to make an outline font, named without a design size.

3 Bitmap font to outlines: autotrace

The best program I know of to fit outlines (i.e., Bézier curves) to bitmaps is `autotrace` [23]. (I found an alternative program `ttf2pt1` [1], but it did not seem as well developed). Some fairly intense mathematics is involved in doing the fitting [17]; fortunately, we don't need to go into that in order to use the program effectively.

The main barrier to using autotrace to convert fonts is that it reads images (such as PBM files [14]), and writes EPSF (among other formats); it has no knowledge of font formats. So our basic strategy is:

1. Convert each character in the bitmap font to an image in PBM format.

2. Run autotrace on that image.

3. Convert the PostScript output, which uses standard graphics operators such as **rmoveto** and **rrcurveto**, to Type 1 opcodes.

4. Reassemble the characters into a font.

This procedure is implemented by mftrace [11], a Python [22] program which pulls the pieces together: it uses gf2pbm [12] to convert individual characters from the GF font to bitmap images; calls autotrace with assorted options to do the fitting; and finally uses t1asm from t1utils [7] to assemble the output into a font again.

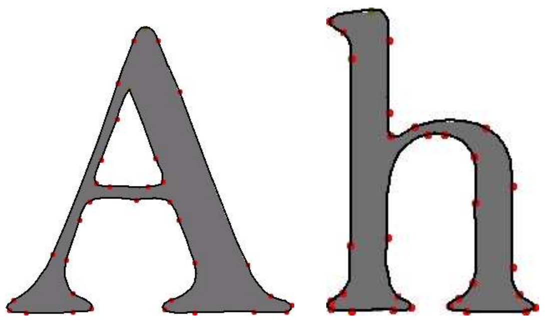
mftrace requires an encoding file to run, and since it does not do path searching, the encoding file must be present in the current directory. The default encoding is tex256.enc, and can be changed with the -e option. (If you don't happen to have tex256.enc on your system, it's available at http://tug.org/fontname/tex256.enc. It is another name for the T1 (Cork) encoding.)

Type 1 fonts have two equivalent formats: pfa (printer font ASCII), which uses only normal plain text characters; and pfb (printer font binary), which is partly binary. As you might expect, pfb files are noticeably smaller, since the font shapes can be compressed more when all eight bits can be used. TEX and friends are happy with either one, so we might as well use pfb.[1]

Here is a first approximation to our command line:

```
mftrace --pfb \
  --gffile=gbvr.1200gf gbvr
```

The result is gbvr.pfb. Here are two of the resulting outline characters showing the main control points:



There is an alternative program textrace [20], which seems equally worthy. I worked with mftrace only because it was easier for me to install and understand. Each program has its own drawbacks and benefits.

Historical aside: I was happy to see that autotrace is partially based on limn, another of the old Fontutils programs; so that work wasn't entirely wasted. It does a vastly better job than limn ever did, which I am even happier to see!

4 Testing the new font from TEX

The above does the real work of converting bitmaps to outlines. Now, to use the result in TEX, we have many configuration details to work out.

4.1 Scaling: mftrace

First, we need metrics to go along with the outlines, which mftrace will generate as an afm file (Adobe Font Metrics [24]), if we specify the --afm option.

We also have to convert between different coordinate systems. When we create a Type 1 font, the so-called 'character coordinate space' uses a 1000 to 1 scaling matrix. That is, 1000 character space units transform into one user space unit (one PostScript point, usually). Put another way, Type 1 expects a resolution of 1000 pixels in the design size, and the PostScript design size is simply 1. This is a much higher resolution than our scanned images have.

It's easiest to explain the scaling factor by looking at a concrete example. Our example image was scanned at a resolution of 1200 dpi, which comes to about 16.6 pixels per point (1200/72.27). Our design size is 24 pt. Therefore, we have about 399 pixels per design size ($16.6 * 24$). We give this value to the mftrace as the magnification (same concept of magnification as in TEX).

mftrace will then scale all the numbers in the outlines by $1000/399 = 2.506$. For example, the image of our capital 'A' is 275 pixels wide. This becomes about 690 in character space coordinates. As a check, the device-independent width in TEX terms turns out to be 6.9 on a 10 pt designsize; yay.

[1] Warning: I found out to my sorrow that it does not work to directly edit the contents of a pfb file in any way, including the header comments; it becomes internally inconsistent and dvips will complain about a a 'non-MSDOS header'.

Bottom line, the magic number is the image design size multiplied by the image resolution in pixels per point. Here's the resulting `mftrace` command line (this is the real one, no approximation):

```
mftrace --pfb \
  --afm --magnification=399 \
  --gffile=gbvr.1200gf gbvr
```

We now have two files: `gbvr.pfb` and `gbvr.afm`.

4.2 Metrics: afm2tfm

Our next job is to convert `gbvr.afm` into TEX font metric files. The easiest way I know of to do this is to use `afm2tfm`'s `-T` option, which lets us get away without using virtual fonts [26]:

```
afm2tfm gbvr.afm -v gbvr.vpl \
  -T tex256.enc >gbvr.xmap
pltotf gbvr.vpl gbvr.tfm
```

`pltotf` will issue warnings about unknown `VTITLE` and `MAPFONT` properties, but no harm is done. We don't need all the TEX virtual font machinery [5], since we're not actually combining multiple fonts, just reencoding a single font.

4.3 Running TEX and dvips

Ok, let's run TEX (`testfont.tex` is a standard file from Knuth):

```
tex testfont
...
Name of the font to test = gbvr at 24pt
Now type a test command...
*\text\bye
...
Output written on testfont.dvi...
Transcript written on testfont.log.
```

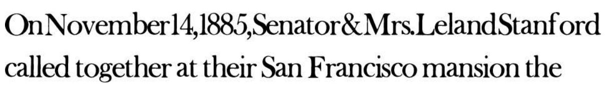
We're almost ready to look at some output. Our last preliminary step is to specify downloading `gbvr.pfb` when the font is used. For `dvips` [15], this is done in a one-line `.map` file. `afm2tfm` gave us the initial line, we just append the download instruction:

```
sed -e 's/$/ <gbvr.pfb/' gbvr.xmap \
  >gbvr.map
```

Finally, we tell `dvips` [15] to use that map file and process the document:

```
dvips -u +gbvr.map testfont.dvi -o
```

The output is `testfont.ps`:


OnNovember14,1885,Senator&Mrs.LelandStanford
called together at their San Francisco mansion the

There are some artifacts in that image due to the conversion process from the screen capture. My apologies, but the main point is the process, after all, not the particular image we used for an example.

The most obvious other problem is that there is no letter spacing. Some methods for addressing that are mentioned in the next section.

5 Final outline output: frontline, pfaedit

We've gone through the process of making a new outline font and typesetting with it in TEX. Now comes the truly hard part: actually making the font as good as it can be. Although `autotrace` does a very respectable job with its default settings, it's inevitable that hand editing of the outlines will be required for best results.

One method for doing this is to change the (numerous) parameters to `autotrace` itself. This can be done from the command line (use `--help` to get a list of options). In addition, a graphical front-end to `autotrace` named `frontline` exists to make experimenting with the option setting easier; it is available from the `autotrace` home page [23].

The other method is to use an outline font editor; the best one I know of is `pfaedit` [25]. As well as straightforward outline editing, `pfaedit` has numerous other significant features:

- Bitmap editing (supports GF and PK [16] formats).
- TrueType output. The option `--truetype` to `mftrace` will call `pfaedit` to get TrueType output, if that's desired.
- Metrics editing: getting the character spacing is as important to the final outcome as the character shapes [8, 21]. `pfaedit` can supply initial side bearings and kerns via the `Auto Width` and `Auto Kern` options on the `Metrics` menu. This is nice, since scanned images generally lack any useful side bearing specifications. (Alternatively, the Fontutils program `charspace` is a non-interactive way of preparing initial side bearings.)
- Autohinting.

As it turns out, the `mftrace --afm` option that we used above also implies `--simplify`, which runs the font through `pfaedit` in order to simplify and autohint the outlines. Thus, no additional options are needed to take advantage of those features.

Additional files and procedures are needed to use new fonts with LATEX [6, 10]. Those articles also describe creating oblique, small caps, and other variants.

Happy fontmaking!

References

[1] Sergey Babkin. ttf2pt1. http://ttf2pt1.sourceforge.net.

[2] Karl Berry. Fontname: Filenames for TeX fonts. http://tug.org/fontname.

[3] Karl Berry and Kathryn Hargreaves. GNU fontutils. http://www.gnu.org/software/fontutils.

[4] Charles Bigelow. Notes on typeface protection. *TUGboat*, 7(3):146–151, October 1986.

[5] Robin Fairbairns. Virtual fonts. http://www.tex.ac.uk/cgi-bin/texfaq2html?label=virtualfonts.

[6] Peter Flynn. Installing PostScript fonts. http://www.silmaril.ie/downloads/documents/installpsfonts.pdf.

[7] I. Lee Hetherington and Eddie Kohler. Type 1 utilities (t1utils). http://www.lcdf.org/~eddietwo/type.

[8] David Kindersley. *Optical Letter Spacing for New Printing Systems*. Wynkyn de Worde Society, distributed by Lund Humphries Publishers Ltd., 26 Litchfield St. London WC2, 1976.

[9] Donald E. Knuth. GF (generic font) format. http://www.ctan.org/tex-archive/systems/knuth/mfware/gftype.web (among other programs).

[10] Philipp Lehman. The font installation guide. http://www.ctan.org/tex-archive/info/Type1fonts/fontinstallationguide.pdf.

[11] Han-Wen Nienhuys. mftrace. http://www.cs.uu.nl/~hanwen/mftrace.

[12] Han-Wen Nienhuys and Paul Vojta. gf2pbm. http://www.cs.uu.nl/~hanwen/mftrace.

[13] Johannes Plass. GV: a PostScript and PDF previewer. http://wwwthep.physik.uni-mainz.de/~plass/gv.

[14] Jef Poskanzer and Bryan Henderson et al. Netpbm. http://netpbm.sourceforge.net.

[15] Tomas Rokicki. Dvips. http://www.ctan.org/tex-archive/dviware/dvips.

[16] Tomas Rokicki. PK (packed font) format. http://www.ctan.org/tex-archive/systems/knuth/mfware/pktype.web (among other programs).

[17] Philip J. Schneider. Phoenix: An interactive curve design system based on the automatic fitting of hand-sketched curves. Master's thesis, University of Washington, 1988. http://autotrace.sourceforge.net/Interactive_Curve_Design.ps.gz.

[18] Richard M. Stallman. GNU coding standards. http://www.gnu.org/prep/standards_48.html. Node: Managing Releases.

[19] Richard M. Stallman. Project GNU (GNU's Not Unix). http://www.gnu.org.

[20] Péter Szabó. textrace. http://textrace.sourceforge.net.

[21] Walter Tracy. *Letters of Credit*. David R. Godine, Publisher, Boston, MA, USA, 1986.

[22] Guido van Rossum. Python. http://python.org.

[23] Martin Weber. Autotrace. http://autotrace.sourceforge.net.

[24] George Williams. Font file formats. http://pfaedit.sourceforge.net/index.html#Formats. This has contains links to PostScript Type 1 and AFM documents, and both the Apple and Microsoft TrueType and OpenType standards documents, among many others.

[25] George Williams. Pfaedit. http://pfaedit.sourceforge.net.

[26] Y&Y. Single TFM file for Type 1 Fonts. http://www.yandy.com/maketfm.htm.

◇ Karl Berry
685 Larry Ave. N
Keizer, OR 97303
USA
karl@freefriends.org
http://freefriends.org/~karl/