Generating Type 1 Fonts from METAFONT Sources

Apostolos Syropoulos

366, 28th October Str
GR-671 00 Xanthi, Greece
E-mail: apostolo@obelix.ee.duth.gr

Abstract

Nowadays most Printers demand PostScript files with scalable fonts instead of bitmapped fonts, as the later are not adequate for most cases. In addition, PDF files generated from PostScript files with embedded bitmapped fonts are poorly rendered on a computer screen. On the other hand, traditionally, PostScript files generated from TeX sources contained bitmapped fonts just because METAFONT generates bitmaps at specific resolutions. This simply implies that it is more than necessary to have scalable versions of the METAFONT sources. And here we present a simple yet effective way to generate vector fonts from METAFONT sources.

1. What is a Scalable Font?

A scalable font is a type font that can be resized (enlarged or reduced) without introducing distortion. The outline of each character (the typeface) is described by equations that represent line segments and curves. The set of all such equations for a complete set of characters is called an *outline font.* The outline font remains essentially the same regardless of the size of the typefaces. Given a typeface definition, a scalable-font system can produce characters at any size (or scale). Aside from this, scalable fonts have an added advantage in that they make the most of an output device's resolution. The more resolution a printer or monitor has, the better a scalable font will look.

The most common scalable font formats are the PostScript Type 1 and Type 3 fonts, the TrueType fonts (PostScript Type 42 fonts are just TrueType fonts with a PostScript wrapper) and the OpenType fonts, which consist of many PostScript fonts or one TrueType font.

2.   What is METAFONT?

Roughly speaking, METAFONT is the program that generates the fonts TEX
uses and, at the same time, a font design programming language. METAFONT
programs describe the typefaces of a new font. Each typeface is described by a
set of equations. So strictly speaking, METAFONT sources are actually scalable
fonts. However, METAFONT the program can process these font descriptions
to generate only bitmapped fonts and font metrics. The font metrics are used
by TEX to typeset source files, while the bitmapped fonts are used to render
the typeset text on the screen or to produce a resolution dependant PostScript
file. Although TEX was designed to be able to deal with fonts generated with
METAFONT, soon it become clear that TEX should be able to deal with other
font formats (mainly scalable fonts). This observation led to the design of the
necessary tools, which are now available in every standard TEX installation.
Still today, the bulk of TEX documents use the "standard" METAFONT fonts,
while many people find it is far easier to create fonts using METAFONT instead
of fancy font editor. So this practically means, that after so many years of
progress we are still in the situation people were 10 (or even more) years ago!

Fortunately, nowadays there are some freely available tools that can be used
to create scalable versions of METAFONT sources in a very easy and systematic
way. In what follows, we describe what we have done to generate Type 1 fonts
for each font in the CB-font family[1].

3.   From METAFONT source to Type 1 Fonts

TEXtrace is a set of Unix shell and Perl scripts developed Szabó Péter
that can automatically create Type 1 fonts from METAFONT sources. The set
of scripts can operate only if we do have a complete TEX installation and
Ghostscript. To generate a Type 1 font, we need to issue the following command

> ./traceall.sh *TEX-fontname Type1*.pfb *UniqueID*

The *UniqueID* can be any 7-digit integer. Although the Type 1 Font Specifi-
cation requires that this number is unique, you can choose any 7-digit number.
Since we needed to create 888 Type 1 fonts, we had to write a Perl script that
would generate a shell script, which, in turn, would be used to actually generate
the fonts. As a side-effect, the script generates the various METAFONT driver-
files and the necessary `cbgreek.map` file, which will be consulted by `dvips` to

---

[1]   The CB-font family is named after Claudio Beccari who undertook the difficult task to
disign a complete set of Greek fonts using METAFONT.

embed the Type 1 fonts in a PostScript file. The following code fragment shows one of the two loops that are used to generate the driver files and populate the shell script, which was used to generate the fonts, and the map file:

```
foreach $size (@lstd_sizes) {
  foreach $name (@lstd_names) {
    $font_name = "$name$size";
    $Font_name = "$name$size.mf";
    open(FONT, ">$Font_name") or
    die "Can't create file $Font_name\n";
    print FONT "%Generated by mkcbfonts\n";
    print FONT "input cbgreek;\n";
    close FONT;
    print TYPE1 "traceall.sh $font_name $font_name.pfb $UID\n";
    $UID++;
    print CB "$font_name TeX-$font_name <$font_name.pfb\n";
  }
}
```

The generated Unix shell script was used to generated 888 Type 1 fonts. The font generation task was performed on a P4 running the latest version of the Solaris 8 x86 operating system and it took about 100 hours to complete! Let us now briefly describe how TeXtrace generates Type 1 fonts from METAFONT sources.

Initially, TeXtrace generates a PostScript file that corresponds to a document with 256 pages, with one page for each possible glyph of the font. The glyphs are generated in a resolution that is calculated with a simple "algorithm" described in the appendix of this article. This file is used to generate bitmap graphics files–one for each glyph. These bitmap graphics files are transformed to trully scalable EPS files with `autotrace`. In general, `autotrace` converts bitmap image data to vector graphics. The program can handle the following formats: BMP (Windows bitmap format), PBM (Portable BitMap format), PGM (Portable Graymap format), PNM (Portable Anymap format), PPM (Portable Pixmap format), and TGA (Targa format). The supported output formats are: AI (Adobe Illustrator), CGM (Computer Graphics Metafile), DXF (AutoCAD Drawing Exchange format), DXF12 (AutoCAD Release 12 DXF [without splines]), EMF (Windows Enhanced Metafile format), EPD (Encapsulated Vectorial Graphics format), EPS (Encapsulated PostScript), ER (Elastic Reality Shape format), FIF (xfig 3.2), MIF (FrameMaker MIF format), PDF (Adobe's Portable Document Format), p2e (pstoedit frontend), and sk (Sketch). Now, the generated EPS files are used to construct the Type 1 fonts. So practically, the most difficult task is performed by `autotrace`. The

generated Type 1 fonts do not have hints and in many instances contain far too many control points. This means that we need to polish the fonts with a font editor. We have used `pfaedit` to polish our fonts and in the next section we show exactly what we did.

4.   Polishing the Type 1 Fonts

PfaEdit is a freely available font editor designed by George Williams. PfaEdit will let you create your own PostScript, TrueType, OpenType, CID-keyed (for oriental languages) and bitmap (bdf) fonts, or edit existing ones. Let us now go on with the description of the font polishing procedure.

After loading the font, we need to select all the glyphs of the font. This is necessary if we want to apply various operations on all glyphs at once. The only (undocumented?) drawback of this approach is that one has to apply every operation two or three times or, else, one will not get the expected results. In order to select all glyps we choose the *Select All* option from the *Edit* menu as the the following screen capture shows:
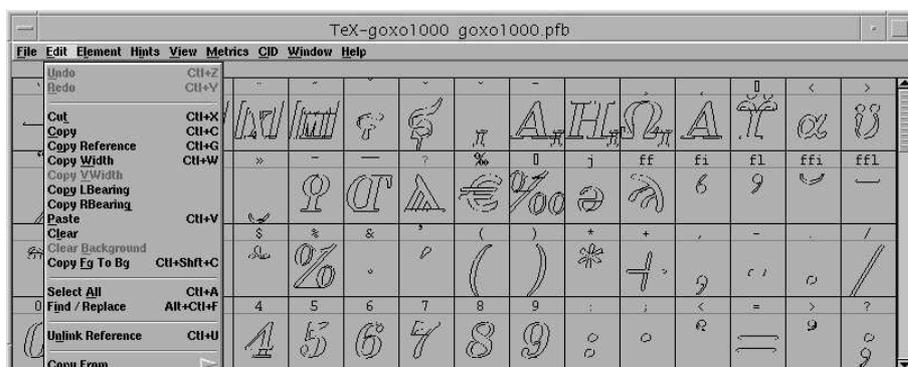
Now that we have selected all the glyphs, we can proceed with the application of various operations to all glyphs. The first thing we must do is to simplify the glyphs. Practically, this means that PfaEdit will examine each glyph and it will remove all "redundant" control points. Of course, the term redundant is a little fuzzy in the sense that we have no real control in what is really redundant or not. However, we do trust PfaEdit as in most cases it does a really decent job. To simplify all the glyphs, we choose the *Simplify* option from the *Element* menu:
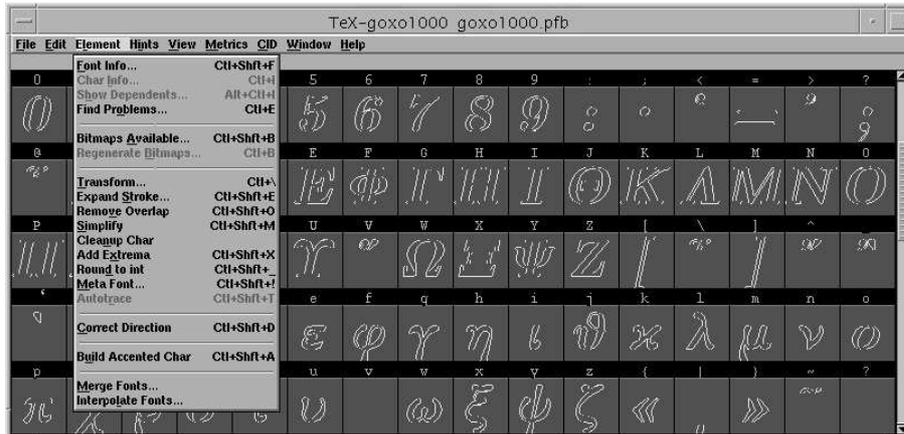
We can also correct the direction of the various line segments and clean-up the glyphs by selecting the appropriate options from the *Element* menu. Now, we are ready to hint the font. But what does this mean? Roughly speaking, hinting a font is a method defining exactly which pixels are turned on in order to create the best possible character bitmap shape at small sizes and low resolutions. Again, we trust PfaEdit's autohinting mechanism to hint the font. To do this, we select the *Autohint* option from the *Hints* menu:
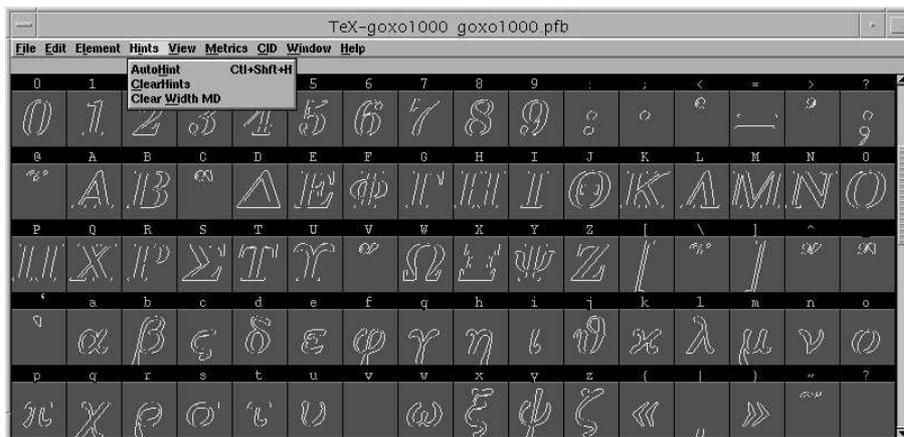
Now that we have autohinted our font, we can generate an improved version of our original Type 1 font. The following screen capture shows exactly what we have to do in order to genearate the font. Note that we have a large number of output formats. And of course this is the last thing we have to do!

5.   We don't Need Commercial Software!

All the tools that are described in this article are what we call Open Source software, that is the source code of all programs is freely available. And TEX was actually a forerunner of the concept of Open Source software. Now, some people may think that Open Source software is not as good as commercial software, but this is simply not true. However, we feel this is not the right forum to advocate Open Source software, in general. But, we have demonstrated that we can do really complicated things with Open Source software. The following figures can be used to judge whether PfaEdit does a really good job in simplifing and auto-hinting fonts.

Original glyph   After simplification

After hinting

6.   Conclusions

We have shown all the steps that are necessary to convert METAFONT fonts to high-quality Type 1 fonts. This procedure has been applied to the conversion of the Greek CB-fonts by Claudio Beccari. We note only that it would be really wonderful if we could have batch oriented tool that could do the simplification and the autohinting of the generated fonts. Any volunteer for a MSc thesis or even a PhD thesis?

Appendix: How does T_EXtrace create the glyph table?

As we have explained TEXtrace creates a PostScript file with 256 pages, one for each glyphs (empty slots in the font generate empty pages). To create this file it sets two environment variables:

$TR˙DPI Holds the default installation resolution (e.g., 600).

$TR˙PTSIZE Equals to $TR˙EM × 72.27/$TR˙DPI, where $TR˙EM is by default equal to 1000.

Variable $TR˙DPI is extracted from a simple PostScript file which, in turn, is generated from a T_EX file that contains the following line:

```
\setbox0=\hbox{\vrule height 10pt width 10pt}\shipout\box0\end
```

This little file creates a little black square that is $10\,\text{pt} \times 10\,\text{pt}$. The PostScript file should have a line as the following one

```
%DVIPSParameters: dpi=600, compressed
```

or else T_EXtrace will fail! Next, T_EXtrace generates the PostScript file with the glyphs by feedin to T_EX a file which looks like the following one:

```
\newdimen\ptsize
\ptsize=120.45pt
\def\whatfont{grmn1200 at 120.45pt}
\def\PutBox{\kern-\ht1\box1}
\def\whatmul{8.302200083022}
\input dump256
```

Note that $120.45 = 1000 \times 72.27/600$ and $8.302200083022 = 600/72.27$. Note also that the PostScript file uses a special paper size.