The METATYPE Project: Creating TrueType Fonts Based on METAFONT

Serge Vakulenko
Cronyx Engineering, Moscow, Russia
`vak@cronyx.ru`
`http://www.vak.ru/proj/metatype`

Abstract

The purpose of the METATYPE project is the development of freeware TrueType fonts for the general user community. The METAFONT language was chosen for creating character glyphs. Currently, glyph images are converted to cubic outlines using a bitmap tracing algorithm, and then to conic outlines.

Two font families are currently under development as part of the METATYPE project: the TeX font family, based on Computer Modern fonts by Donald E. Knuth, retaining their look and feel. A rich set of typefaces is available, including Roman, Sans Serif, Monowidth and Math faces with Normal, Bold, Italic, Bold Italic, Narrow and Wide variations; and second, the Maestro font family, which is designed as a Times look-alike. It also uses Computer Modern sources, but with numerous modifications.

Résumé

Le but du projet METATYPE est le développement de fontes TrueType libres pour la communauté générale d'utilisateurs. Le langage de programmation METAFONT est utilisé pour créer les glyphes. Actuellement, les glyphes sont convertis d'abord en contours cubiques, en utilisant un algorithme d'auto-traçage de bitmap, et ensuite en contours coniques.

Deux familles de fontes sont actuellement en développement, dans le cadre du projet METATYPE : la famille «TeX font», basée sur les fontes Computer Modern de Donald E. Knuth, et proposant du romain, du sans empattements, du mono-chasse, et les fontes mathématiques, dans les styles : régulier, gras, italique, gras italique, étroitisé et élargi ; la famille «Maestro», qui est un clone du Times. Cette dernière utilise également le code de Computer Modern avec un grand nombre de modifications.

Why TrueType?

The world is getting smaller. We can see how the Internet and globalization cause people of different cultures and races to come into contact with each other more and more. The world is thus becoming more multilingual, something the computer world has known for some time: the first draft of the Unicode standard is dated 1989 [1].

At the same time, free software is gaining in popularity. Linux is more and more widespread, and it needs fonts; lots of fonts; a variety of good fonts with Unicode support. We already have a standard format in TrueType. (In a super-format, combined with Type 1, it is called OpenType [2].)

The TrueType format has several advantages:

- It is an open standard; the specification is available online free-of-charge.
- High-quality (commercial) fonts are abundant.
- There exists a free implementation named FreeType [3], of very high quality.
- The XFree86 graphic environment fully supports TrueType fonts, including antialiasing.

The author is bold enough to assert that Unicode and TrueType are the future. "Young" programming languages, such as Java and C#, use Unicode for the basic representation of text strings. Visionary operating systems, such as Windows NT, support Unicode at the kernel level. Word processors use Unicode for storage of documents (XML, RTF and other formats). The number of web sites with Unicode encoding grows every day. Cellular telephones use Unicode for web browsing (WAP). The world is slowly moving toward Unicode. And TrueType seems to be a natural result of font technology development.

The problem is that currently available systems for font development do not support Unicode and/or TrueType, or are not free. The METATYPE project is intended to fill this gap.

METAFONT as a glyph source language

METAFONT was chosen as the method for glyph development for the following reasons:

- It is a powerful, advanced language for glyph representation, and importantly, it is well documented.
- Glyph parametrization provides a straightforward means of creating entire families of fonts: normal-bold, normal-slanted, narrow-wide, serif-sans serif, etc.
- There are a number of very high-quality fonts (Computer Modern, AMS, etc.), whose sources can be studied.

It is also important that METAFONT is implemented on the majority of modern platforms and is available free of charge.

For the METATYPE project, an existing METAFONT/Web2C implementation with no additional modifications was used.

What are the problems?

So, we have METAFONT, and an implementation of METAFONT in Web2C [4]. What prevents us from taking, say, the source for Computer Modern [5] and transforming it easily to TrueType format? We see several problems here.

Problem 1: 16-bit encoding. METAFONT supports only an 8-bit encoding of symbols. But we need 16-bit Unicode encoding. The situation is aggravated because Computer Modern, as well as the majority of other METAFONT fonts, is optimized for use with TeX. Thus, they have non-standard encodings, which, moreover, is different for different fonts of the family.

For the transition to Unicode, the decision was made to store every glyph source in a separate file. Glyphs are grouped by 16 symbols into subblocks, and subblocks are grouped into blocks. Each block NN contains up to 256 symbols in the range NN00-NNFF.

Definitions common to all glyphs of a font are collected in a separate file, `base.mf`. For certain subsets of symbols there may be separate definition files, for example, `cyrbase.mf` for Cyrillic and `greekbase.mf` for Greek characters. Parameters for various styles are located in separate `param.mf` files.

The existing Computer Modern source was manually processed and split into two fonts, which the author has named "TeX" [6] (381 symbols) and "TeX Math" [7] (104 symbols).

Problem 2: splines. For TrueType, glyphs must be represented as contours consisting of splines. The best solution would be to derive splines directly from METAFONT. Another possibility is to use METAPOST — the implementation of METAFONT with PostScript output — since it is possible to extract the required splines from PostScript output. There are several utilities that use this method to generate PostScript Type 3 fonts, for example mf2pt3 [8].

The first difficulty here is that the contour generated by METAPOST must be converted to "canonical form", i.e., without self-crossings. This is a separate interesting and difficult mathematical task, the solution for which must be a task for later. For now we use a simpler method — generating a contour by tracing the glyph raster image.

Autotrace [9] is used to transform the raster image of a glyph into a contour. Autotrace has been extended to allow direct input of files in GF format.

The second difficulty lies in the transformation of splines. Autotrace generates cubic splines (third-order Bézier curves), while TrueType requires conic splines (second-order Bézier curves). The solution to this mathematical problem is to cut a cubic spline into two "close enough" conic splines. Thus, another new feature was added to Autotrace to convert traced contours to conic splines and to output a special UGS (Unicode Glyph Source) format.

Problem 3: hints. For use on low resolution devices such as monitors, TrueType fonts are equipped with so-called *hints*. Hints are programs written in the pseudo-code of a special interpreter, which for every symbol define some changes of the shape of a symbol to enhance glyph rasterization. Creating good hints is a very hard manual task.

Hints are not currently implemented in METATYPE. Support for hints will likely be added in the future.

Some modern rasterizers, such as FreeType 2 [10], do not use hints (because of patent problems [11]). Instead, they use an "auto-hinting" technology. In this case real hints are not necessary.

Problem 4: low resolution bitmap glyphs. You can also use raster glyphs for low-resolution environments like monitors, embedding the raster glyphs into the TrueType font in addition to the contour. Such rasters can be created by means of METAFONT, commonly in 13, 14, 15, 16, 17, 18, 20 and 23 pixels. Thus the most frequently used point sizes are covered (see table 1).

METAFONT generates rasters in GF format. To convert them to UGS format, a special utility `gf2ugs` was developed.

Problem 5: building TTF files. A TrueType file has a very complex structure. Fortunately, there is a Python library (Fonttools [12]) that makes it possible to work with TTF files. All complexities of the organization of a font file are hidden. To build a font, it is enough to make a dozen XML data files, and then to call an appropriate library procedure.

Font height	METATYPE font size	96 dpi — X Windows	100 dpi — MS Windows, Small fonts	120 dpi — MS Windows, Large fonts
13 pixels	96gf	10 pt	9 pt	8 pt
14 pixels	102gf		10 pt	
15 pixels	108gf	11 pt	11 pt	9 pt
16 pixels	114gf	12 pt		
17 pixels	120gf	13 pt	12 pt	10 pt
18 pixels	132gf	14 pt	13 pt	11 pt
20 pixels	144gf	15 pt	14 pt	12 pt
23 pixels	168gf	17 pt	17 pt	14 pt

Table 1: Bitmap font height in pixels and covered point sizes

How it all works

The source code for every glyph is stored in a separate file in a METAFONT format. The scheme of glyph compilation is shown in figure 1. As a result, the file in UGS format is created, containing a contour and rasters for several different pixel sizes.

When UGS files for all symbols are ready, the TTF and BDF fonts are assembled (shown in figure 1).

This work is carried out by the Python scripts `mk_db.py`, `mk_ttx.py` and `mk_bdf.py`. The `mk_db.py` script collects all UGS glyphs into a single DBM database to speed up working with the data. The `mk_ttx.py` script builds XML font data files and, using the Fonttools library, creates a TTF file. Lastly, the `mk_bdf.py` script transforms raster data to the BDF format for use under the X Window System.

UGS file format

For intermediate storage of the glyph data, a special format named UGS (Unicode Glyph Source) was developed. It is an ASCII-encoded text file where each line contains a single statement. An example of a UGS file for the symbol `FULL_STOP` is given below:

```
symbol 0x2e design-size 2048
      advance-width 569
      contour
            path
                  dot-on 273 216
                  dot-off 258 213
                  dot-on 245 209
                  dot-off 231 203
                  ...
                  dot-on 273 216
            end path
      end contour
      left-bearing 176
      right-bearing 177
      ascend 218
      descend -1
end symbol

symbol 0x2e design-size 33
      advance-width 9
      bitmap width 4 height 4
             . * * .
             * * * *
             * * * *
             . * * .
      end bitmap
      left-bearing 2
      right-bearing 6
      ascend 4
      descend 0
end symbol
```

Installing METATYPE

The METATYPE distribution can be downloaded from SourceForge [14]. The current version is available by CVS [15].

METATYPE depends on the following 6 items:

1. Python 2.2.2 [16]
2. Fonttools 2.0b1 [17], with PyXML 0.8.2 [18], and Numeric Python 23.0 [19]
3. Netpbm 10.15 [20], with libpng 1.2.5 [21]
4. Freetype 2.1.4 [22]
5. Web2C 7.3.1, including METAFONT 2.7182 and METAPOST 0.641 (we used teTEX 1.0.7 [23])
6. Autotrace 0.31.1 [24], patched

Before installing Autotrace, you must apply the patch `autotrace.pch` to it:

1. Unpack Autotrace 0.31.1
2. Enter directory autotrace-0.31.1
3. Apply patch:
   ```
   patch -p1 < autotrace.pch
   ```
4. Execute `automake`
5. Run script `./configure`
6. Execute `make` and `make install`.

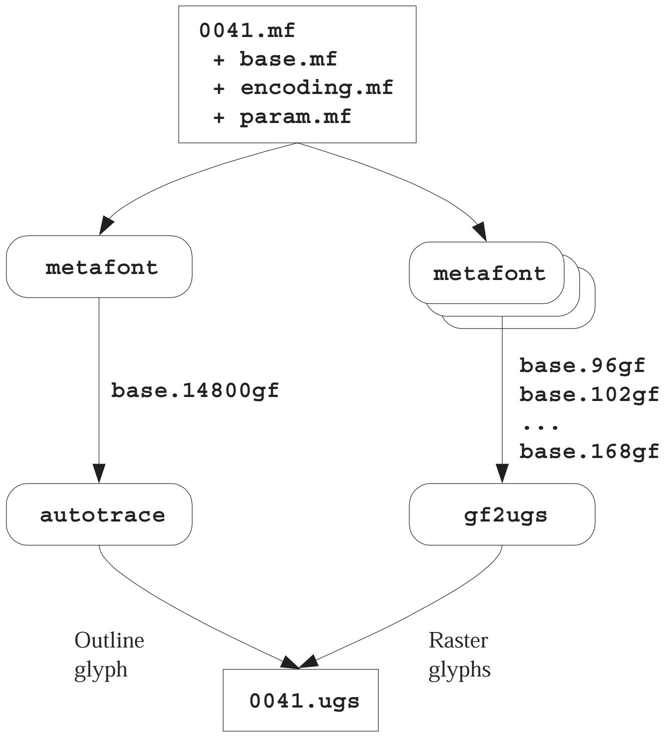


Fig. 1: Translating MF to UGS

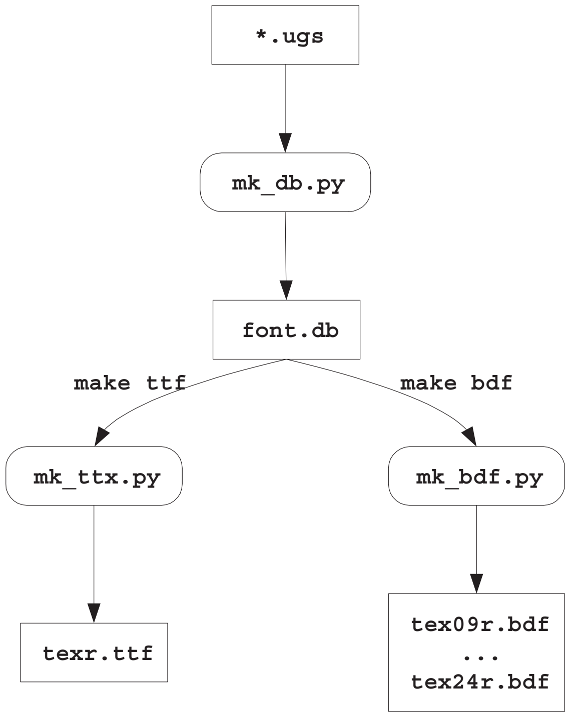


Fig. 2: Translating UGS to TTF and BDF

The top directory of the METATYPE package contains several utilities, scripts and makefiles necessary for processing fonts. The source code of fonts is contained in subdirectories, one directory per font family. For example, the family of fonts named "TeX" is placed in subdirectory `cm/`.

To prepare all the METATYPE utilities, execute `make` in the top directory of the METATYPE project.

Font source code is organized into three levels. All Unicode space is divided into blocks of 256 symbols, and each block is placed in a separate directory. Blocks are further divided into 16 subblocks with 16 characters each. An example of the file structure of a single font family is shown in figure 3.

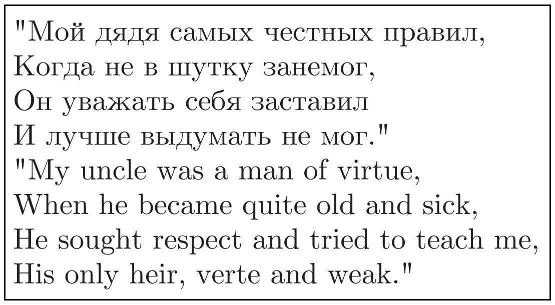
The `compile` directory is used when compiling a font. Each font style is compiled in a separate subdirectory with the appropriate name: `compile/roman`, `compile/roman-italic`, etc. During compilation, subdirectories for all blocks and subblocks are created automatically.

- To compile all glyphs for all styles, enter the `compile` directory and execute `make`.
- To build TrueType fonts, execute `make ttf`.
- To build fonts in BDF format (sizes 9, 10, 11, 12, 13, 14, 18 and 24 points at 100 dpi), run `make bdf`.

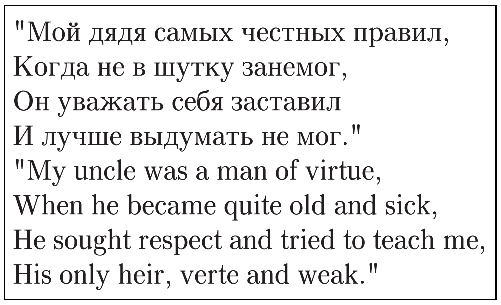
The result

Here is an example of the TeX Roman and Maestro Roman fonts created by METATYPE. These figures were made using Adobe Illustrator and imported into the article as EPS graphics.

TeX:


"Мой дядя самых честных правил,
Когда не в шутку занемог,
Он уважать себя заставил
И лучше выдумать не мог."
"My uncle was a man of virtue,
When he became quite old and sick,
He sought respect and tried to teach me,
His only heir, verte and weak."

Maestro Roman:


"Мой дядя самых честных правил,
Когда не в шутку занемог,
Он уважать себя заставил
И лучше выдумать не мог."
"My uncle was a man of virtue,
When he became quite old and sick,
He sought respect and tried to teach me,
His only heir, verte and weak."

(A. S. Pushkin. *Yevgeny Onegin*; English translation by Dennis Litoshick.)

Unsolved problems

At present the Fonttools library does not support adding raster data to TrueType fonts. Extending this library by adding support for the tables EBDT, EBLC and EBSC is required.

The problem of kerning is not yet solved. We plan on adding an algorithm for automatic kerning.

Further, the character set of the TeX and Maestro font families is incomplete. To cover even the simplest of needs, adding support for Latin-1, Latin Extended-A and Latin Extended-B is highly desirable.

Lastly, the contours created by tracing rasters are not optimal. It would be nice to build an optimizer to reduce the number of spline segments without distorting the glyph's appearance.

Acknowledgments

The author wishes to thank Cronyx Engineering Company for making available the necessary resources for the project.

References

[1] `http://www.unicode.org/history/`
[2] `http://www.microsoft.com/typography/specs/`
[3] `http://www.freetype.org/`
[4] `http://www.tug.org/web2c/`
[5] `ftp://cam.ctan.org/tex-archive/fonts/cm/mf/`
[6] `http://www.vak.ru/proj/metatype/cm/roman/`
[7] `http://www.vak.ru/proj/metatype/cm-math/roman/`
[8] `http://obelix.ee.duth.gr/~apostolo/mf2pt3.html`
[9] `http://autotrace.sourceforge.net/`
[10] `http://www.freetype.org/freetype2/index.html`
[11] `http://www.freetype.org/patents.html`
[12] `http://sourceforge.net/projects/fonttools/`
[13] `http://www.python.org/`
[14] `http://sourceforge.net/project/showfiles.php?group_id=41605`
[15] `cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/metatype checkout metatype`
[16] `http://python.org/2.2.2/`

```
cm/                                      — TeX font family, based on Computer Modern
├─ 00/                                   — Unicode block 0x0000-0x00FF
│   ├──── 002/                           — Unicode block 0x0020-0x002F
│   │      ├──────── 0020.mf             — Glyph 0x0020 — SPACE
│   │      ├──────── 0021.mf             — Glyph 0x0021 — EXCLAMATION_MARK
│   │      └──────── ...
│   │                002f.mf             — Glyph 0x002F — SOLIDUS
│   ├──── 003/                           — Unicode block 0x0030-0x003F
│   ├──── ...
│   │     00f/                           — Unicode block 0x00F0-0x00FF
│   └──── encoding.mf                    — List of charcodes for block 0x0000-0x00FF
├─ 02/                                   — Unicode block 0x0200-0x02FF
├─ 03/                                   — Unicode block 0x0300-0x03FF
├─ ...
└─ compile/                              — Compilation directory
     ├──── roman/                        — Roman style font
     │      ├──────── Makefile           — Build script
     │      └──────── param.mf           — Parameters for Roman style font
     ├──── roman-bold/                   — Roman Bold style font
     ├──── ...
     ├──── sans-italic/                  — Sans Italic style font
     ├──── Makefile                      — Build script
     ├──── base.mf                       — Definitions for all glyphs
     ├──── cyrbase.mf                    — Definitions for Cyrillic glyphs
     └──── greekbase.mf                  — Definitions for Greek glyphs
```

Fig. 3: Structure of the METATYPE font source directory

[17] http://prdownloads.sourceforge.net/fonttools/fonttools-2.0b1.tgz

[18] http://prdownloads.sourceforge.net/pyxml/PyXML-0.8.2.tar.gz

[19] http://prdownloads.sourceforge.net/numpy/Numeric-23.0.tar.gz

[20] http://prdownloads.sourceforge.net/netpbm/netpbm-10.15.tgz

[21] http://prdownloads.sourceforge.net/libpng/libpng-1.2.5.tar.gz

[22] http://prdownloads.sourceforge.net/freetype/freetype-2.1.4.tar.gz

[23] http://www.tug.org/teTeX/

[24] http://prdownloads.sourceforge.net/autotrace/autotrace-0.31.1.tar.gz