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 Metatype project is the development of freeware TrueType fonts for generic user community.

METAFONT language is chosen for creating character glyphs. Currently, glyph images are converted to cubic outlines using bitmap tracing algorithm, and then to conic outlines.

Two font families are currently been under development as a part of Metatype project:

1. 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, Monotype and Math faces with Normal, Bold, Italic, Bold Italic, Narrow and Wide variations.
2. Maestro font family is designed, as a Times-lookalike. It also uses Computer Modern sources, with a lot of modifications.

Why TrueType?

The world becomes closer. Now we observe, how the Internet and globalization of economics cause increase of intensity of contacts between representatives of different peoples, cultures, races. The world becomes more and more multilingual. The computer branch for a long time has realized this fact: the draft variant of standard Unicode is dated 1989 [1].

On the other hand, in the field of the software there is a recognition and growth of popularity of the free software. Linux goes, and surely it needs fonts. Many, many fonts, good and different, with Unicode support. And we have such standard on a format of a font — TrueType. Later, with addition of support Type 1, it was renamed to OpenType [2].

It is possible to list the following advantages of format TrueType:

- Open standard; the specification is available on Internet free-of-charge.
- There is a plenty of (commercial) high quality fonts.
- There exists a free implementation named FreeType [3], of very high quality.
- Graphic environment XFree86 fully supports TrueType fonts, including smoothing.

The author incurs boldness to assert, that Unicode and TrueType are the future. "Young" programming languages, such as Java and C#, use Unicode as the basic representation of text strings. Far-seeing operational systems, such as Windows NT, provide support Unicode at a kernel level. Word-processors use Unicode for storage of documents (XML, RTF and other formats). The amount of Internet sites with Unicode encoding grows every day. Cellular telephones use Unicode for Internet 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.

- Powerful, advanced language of glyph representation. And, which is very important, well documented.
- Glyph parametrization provides an opportunity of creation of families of fonts: normal-bold, normal-slanted, narrow-wide, serif-sans serif etc.
- Availability of a number of fonts of very high quality: Computer Modern, AMS etc.

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

For MetaType project, it was decided to use an existing metafont/web2c implementation with no additional modifications.

What is the problem

So, we have Metafont language, and an implementation of metafont in Web2C [4] package. What prevents us to take, say, sources of fonts Computer Modern [5] fonts and to transform them easily to TrueType format? We see several problems here.

Problem 1: 16-it encoding. Metafont supports only an 8-bit encoding of symbols. But we need 16-bit Unicode encoding. The situation is aggravated with that Fonts Computer Modern, as well as the majority of others, are focused on usage with system TeX. They have non-standard encoding, moreover, the encoding is different for different fonts of the family.

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

The definitions, which are common to all glyphs of font, are allocated in a separate file `base.mf`. For the certain subsets of symbols there might exist separate files of definitions, for example `cyrbase.mf` for cyrillic and `greekbase.mf` for greek characters. Parameters of fonts of various styles are located in separate files `param.mf`.

The existing sources of Computer Modern fonts were 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. Or to apply Metapost — the implementation of Metafont with PostScript output, here it is possible to extract required splines from PostScript output. There exist several utilities which use this method for generation of 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. It is a separate interesting and difficult mathematical task. In future, it would be very promising to solve it. But for now it was decided to use a more simple method — generation of a contour by tracing the glyph raster image.

Autotrace [9] utility is used for transforming raster image of glyph into a contour. Autotrace was extended by a feature of directly input files in GF format.

The second difficulty consists in transformation of a splines. Autotrace generates cubic splines (Bézier curves of third order), and for TrueType are required conic splines (Bézier curves of second order). This mathematical problem was solved by a method of cutting a cubic spline into two “close enough” conic splines. A new feature was added to Autotrace: convert traced contour to conic splines and output it in a special UGS format (Unicode Glyph Source).

Problem 3: hints. For using fonts on low resolution devices, for example on the screen of a monitor, TrueType fonts are equipped with so-called *hints*. Hints are programs written on a pseudo-code of a special interpreter, which for every symbol define some changes of the shape of a symbol, with the purpose of enhancing glyph rasterization. Creating good hints is a very hard task, actually it is a manual work.

Now hints are not used in Metatype project. Probably, in the future, the support for hints will be added.

Some modern rasterizers, for example 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. For display to the screen of the monitor it is offered to use another method — raster glyphs. TrueType font can contain for each symbol, in addition to a contour, also a set of raster glyphs, for use at very low resolution. Such rasters can be created by means of Metafont, and added to TrueType font file. It is offered to create 8 raster fonts in the size 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 file in TTF format. The file in TrueType format has very complex structure. Fortunately, there exists a library Fonttools [12], which makes it possible to work with TTF files, from Python [13] language. All complexities of the organization of a font file are hidden. To build a font, it is enough to make a dozen of data files in XML format, and then to call an appropriate library procedure.

How it all works

The source code of 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 several rasters for different pixel sizes.

When UGS files for all symbols are ready, an assembly of fonts TTF and BDF is performed (shown nn figure 2).

This work is carried out by scripts `mk_db.py`, `mk_ttx.py` and `mk_bdf.py` (written in Python language). Script

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

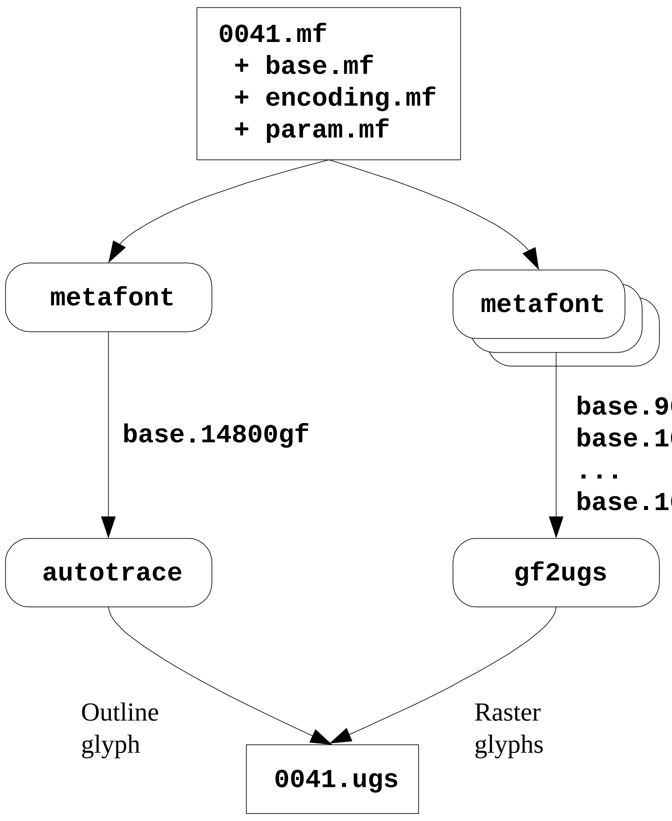

Figure 1: Translating MF to UGS

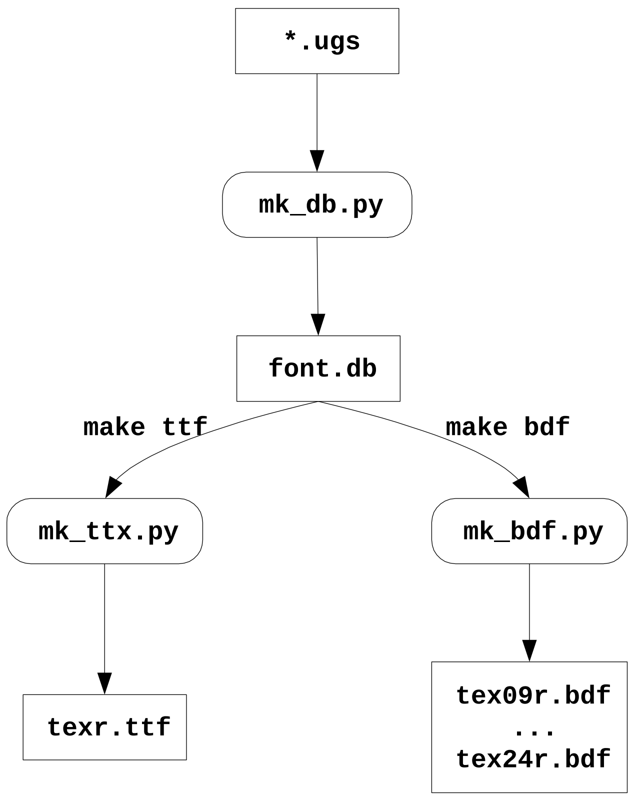

Figure 2: Translating UGS to TTF and BDF

`mk_db.py` collects all UGS glyphs to a single DBM database, for acceleration of work with the data. Script `mk_ttx.py` builds XML font data files and, using Fonttools library, creates TTF file. Script `mk_bdf.py` transforms raster data to BDF format, which could be used in X Windows.

UGS file format

For intermediate storage of the glypht 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. The example of a UGS file for 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] site. The current version is available by CVS [15].

Before using Metatype, you must install the following software:

- Python 2.2.2 [16]
- Fontools 2.0b1 [17], with PyXML 0.8.2 [18], and Nymeric Python 23.0 [19]
- Netpbm 10.15 [20], with libpng 1.2.5 [21]
- Freetype 2.1.4 [22]
- Web2C 7.3.1, including Metafont 2.7182 and Metapost 0.641 (author uses the package teTeX 1.0.7 [23])
- 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`.

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

To prepare all utilities of package Metatype, execute

```
make
```

in the top directory of Metatype project.

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

Compilation of a font is performed in the directory `compile`. 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 automatically will be created.

To start compilation of all glyphs for all styles, enter directory `compile` 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 the example of using fonts TeX Roman and Maestro Roman, created by Metatype package. These figures were made using Adobe Illustrator and imported into the article as EPS graphics.

```
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                 — Defines for all glyphs
       ├──── cyrbase.mf              — Defines for cyrillic glyphs
       └──── greekbase.mf            — Defines for greek glyphs
```

Figure 3: Structure of Metatype font source directory

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 virture,
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 virture,
When he became quite old and sick,
He sought respect and tried to teach me,
His only heir, verte and weak."

(A.S.Pushkin. “Eugeny Onegin”. English translation Dennis Litoshick.)

Unsolved problems

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

The problem of kerning is not solved. In the future it is supposed to use one of algorithms of automatic kerning.

The character set of “TeX” and “Maestro” font families is incomplete. To cover a minimum needs, adding symbol sets Latin-1, Latin Extended-A and Latin Extended-B is highly desirable.

The contours created by tracing rasters are not optimal. It is desirable to build an optimizer, which will reduce the amount of spline segments without distortion of the glyph appearance.

Acknowledgments

The author wishes to thank Cronyx Engineering Company for providing the necessary resources for this 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/

[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