♢ ♢ ♢

MetaType1: *a* METAPOST*-based engine for generating* Type 1 *fonts*

Bogusław Jackowski*, Janusz M. Nowacki†, and Piotr Strzelczyk‡

ABSTRACT.
A package for preparing parameterized outline fonts in PostScript [6] Type 1 [8] format is described. The package makes use of METAPOST [3], awk [4], and T1utils [5], therefore is supposed to be easily portable to various computer platforms. Its beta version along with a sample font (Knuth's logo font) is available from: `ftp://bop.eps.gda.pl/pub/metatype1`

KEYWORDS: outline fonts, scalable fonts, parameterized fonts, PostScript Type 1 fonts, MetaFont, MetaPost

The situation of font collections available for the TEX [1] system can certainly be classified as *bad* if not *ugly*. METAFONT [2], with its bitmap fonts, nowadays seems more and more obsolete. The near future appears to belong to scalable outline fonts. But it would be a pity if METAFONT, with its marvellous engine for creating character shapes, were to remain unused.

Already relatively long ago it was recognized that the design of METAFONT is unsufficiently extensible. In 1989, at the tug meeting, Hobby announced the beginning of work on METAPOST, a program for the generation of a set of eps (*encapsulated PostScript*) files instead of a bitmap font; in 1990, the first version of METAPOST was running. In the same year, Yanai and Berry [23] considered modifying METAFONT in order to output PostScript Type 3 [7] fonts.

Type 3 fonts can be legitimately used with TEX. Actually, bitmap fonts are always implemented as Type 3 fonts by dvi-to-PostScript drivers. Recently, Bzyl [15] has put a lot of effort into the revival of Type 3 fonts in the TEX world. Nevertheless, Type 3 fonts have never become as popular as Type 1 fonts, and probably they never will. One cannot install Type 3 fonts under Windows, MacOS, or X Window, although there are no serious reasons for that—it would suffice to include a PostScript interpreter into an operating system, which is not an unthinkable enterprise. But the commercial world is ruled by its own iffy rights... Anyway, in order to

*B.Jackowski@GUST.org.pl
†J.Nowacki@GUST.org.pl
‡P.Strzelczyk@GUST.org.pl

preserve the compatibility with the surrounding world, one should rather think about Type 1 than Type 3 fonts.

Alas! The issue of converting automatically METAFONT sources to Type 1 format turned out to be more difficult than one could expect (cf. [18, 19, 20, 21, 22]) and after nearly twenty years since the birth of TEX no programming tool for generating Type 1 fonts has appeared. As a consequence there is a glaring scarcity of fonts created by the TEX community.

The MetaType1 package was developed as a response to that bitter situation. Whether it can be classified as *good*—the future will reveal. So far, MetaType1 helped us to prepare a replica of a Polish font designed in the second decade of the twentieth century, Antykwa Półtawskiego [16]. It also proved useful in improving some freely available families of fonts [17].

Which font format?

Among a plethora of currently attainable font formats (see [7] for formats devised by Adobe alone), two are predominant: Type 1 [8] and TrueType [9]. The Type 1 format was Adobe's top secret for six years. In 1990, Adobe decided to disclose the specification after Microsoft and Apple had published the TrueType format. TrueType fonts, despite their very obscure documentation, have become the "mother fonts" of interactive (window) systems. Type 1 fonts can also be used with these systems; however, an additional commercial program, atm (*Adobe Type Manager*), is needed.

From the point of view of TEX users, Type 1 fonts are certainly more suitable, because they are an intrinsic part of the PostScript language. Although a one-to-one conversion between TrueType and Type 1 formats is, in general, impossible, there exist converters that can be used (with care) for this purpose. There are free TrueType-to-Type 1 converters (e.g., [11]), but Type 1-to-TrueType converters seem to be available only as commercial products. Somewhere in between can be located a built-in Windows NT 3.5 converter from Type 1 to TrueType.

Incidentally, contemporary PostScript interpreters accept Type 42 fonts [7], which are essentially TrueType fonts "wrapped" in a PostScript structure. The conversion (one-to-one) between TrueType and Type 42 is pretty simple and free converters are easily available (e.g., [12]).

A few years ago, Microsoft and Adobe announced a joint initiative: they proclaimed that a new font format, OpenType, is to replace both TrueType and Type 1. Microsoft in their documentation on TrueType say, perhaps a bit prematurely, that the TrueType font file specification is "of historical interest only." At present, Adobe offers a cost-free (although licensed) converter from Type 1 to OpenType for the Macintosh and Windows NT platforms. We can expect that more such converters will emerge.

The future is always hidden; we believe, however, that today we can safely invest our efforts in the creation of Type 1 fonts.

Interactive or programming tool?

There are several interactive programs for creating outline fonts. We doubt whether a satisfactorily uniform font can be produced using an interactive tool alone. Fonts are complex monsters and one cannot expect that creating them will ever be an easy task. They are governed by a multitude of parameters such as a stem thickness, serif size and shape, italic angle, the height of majuscules and minuscules, the position of ascenders and descenders, the width of a particular group of characters (e.g., digits should have identical width if we want to use the font in numerical tables), etc. It is particularly difficult to preserve the similarity of shapes appearing repeatedly in a group of characters, e.g., ovals in the letters 'b', 'd', 'o', 'p', and 'q'.

In general, the more irregular the font, the more adequate is an interactive tool. Fonts used for book typesetting, however, are exceptionally uniform. Therefore, some interactive programs provide a programming interface that facilitates controlling uniformness; for example, the commercial FontLab program offers a Python interface in addition. Kinch's approach [18] can be considered as a step further. His MetaFog package is meant for the (semi)manual tuning of glyphs programmed in METAPOST. Despite many advantages, such a "hybrid" approach has a principal drawback: a slight modification of a font may lead to a lot of laborious manual intervention. In particular, parameterization, the boon of the programming approach, is lost.

Only an entirely programmable tool overcomes all these hindrances, but it brings with it its own disadvantages, as programming is apparently difficult for most present-day computer users. This means that the number of MetaType1 users will be limited. Since we are very fond of programming, we can make the prognosis that the number of users will not be less than three.

METAFONT, METAPOST, or ...?

From the very beginning, we abandoned the idea of writing one more stand-alone program (by, e.g., modifying METAFONT or METAPOST) as we didn't want to be involved in technical implementation details. We wanted to make use of existent reliable programs and to focus our attention on the problem of generating Type 1 fonts. Therefore, we had to choose: METAFONT or METAPOST?

The problem with METAFONT is that it writes only gf, tfm, and log files, hence transforming the output from METAFONT to a completely different format, such as PostScript Type 1, is somewhat inconvenient. Its successor, METAPOST, is capable of writing several (text) files, although pictures generated by METAPOST do not form any structure. Fortunately, METAPOST inherited from METAFONT the ability of writing tfm files, which significantly eases the process of generating fonts for TEX, since no extra font installation programs are needed. An argument that can be raised against using METAPOST is that it does not belong to Knuth's canonical distribution; but one cannot avoid using non-canonical software anyway if one wants to produce Type 1 fonts. All in all, we decided to use METAPOST as the bedrock of our package.

Transforming METAPOST output appropriately and imposing a font structure on it has to be done by means of an external program. Our presumption was that it should be a freely available, popular, and portable program.

We believe that AWK (actually, GAWK [13]) meets these requirements. The main drawback of AWK is that it handles only text files. METATYPE1 output files are mostly text files, e.g., AFM (*Adobe font metric*) files, with one pivotal exception, however: many TYPE 1 applications, notably ATM, require a binary form of TYPE 1, PFB (*POSTSCRIPT font binary*). We have considered writing our own assembler in the POSTSCRIPT language (to be used with GHOSTSCRIPT [14]), but finally we gave up and employed a TYPE 1 assembler from the T1UTILS package.

For reasons that are hard to explain, another binary file, PFM (*printer font metric*), is required in order to install TYPE 1 on a Windows system. We decided to prepare our own PERL script for generating PFMs out of AFMs, because the task was relatively simple in spite of the somewhat obscure PFM documentation (sigh) and, moreover, it is easier to manage home-grown software. The script can be used independently of the rest of the engine and, actually, it is not part of the METATYPE1 package.

METATYPE1: AN OVERVIEW

Having answered the fundamental question, i.e., why we are reinventing the wheel, we can start to survey the METATYPE1 engine. Figure 1 shows the structure of the METATYPE1 engine. The boxes drawn with bold lines denote final results, the boxes drawn with dashed lines denote temporary results.

Step 1: METAPOST *processing*
The METAPOST font base contains quite a few macros which are useful in preparing a font. They are similar to Knuth's CMBASE macros. For example, the base contains the macros `beginglyph` and `endglyph`, analogous to `beginchar` and `endchar`. Obviously, the differences are more profound than a simple renaming of some macros, e.g., a lion's share of the code is related to the rules with which TYPE 1 should comply.

From the point of view of a font creator the main difference is that pens are not allowed in METATYPE1 programs. A user is responsible for constructing a proper outline of a glyph: paths should not cross each other (in particular, no self-intersections should occur) and should be properly oriented, i.e., outer paths run anti-clockwise, inner paths run clockwise. There are some macros that facilitate finding the contour of a stroke drawn with a pen ("expanding stroke") or finding the common part of two areas ("removing overlaps"), but, in general, such operations cannot be reliably programmed in METAFONT/METAPOST and therefore users are expected to know what they are doing.

METAPOST works in two passes:

- During the first pass, character glyph files (EPS) and a few auxiliary files (e.g., containing the kerning information) are being generated. The files from the first pass are subsequently processed by AWK.

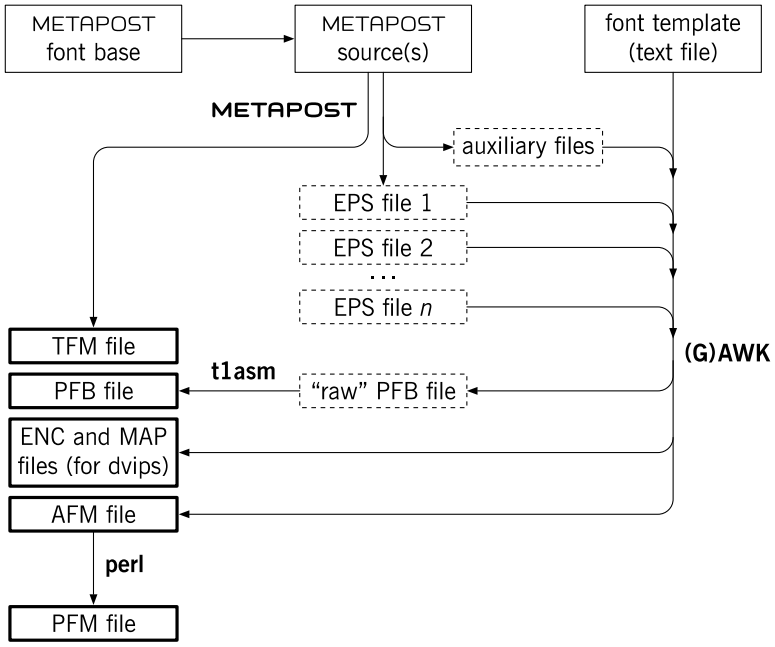

figure 1: The general scheme of the MetaType1 engine.

◇ During the second pass, only TFM files are being generated. All drawing operations are switched off. Writing EPS files, however, cannot be switched off. One can live with it, but it is a somewhat vexing situation—why generate lots of useless files which are only removed later on? The following trick is exploited during the TFM-generating pass: usually, METAPOST appends a numeric extension to the file name; however, if the character code is negative, the extension is simply `ps`; thanks to this, one has to remove only a single dummy file instead of hundreds of them.

Additionally—similar to CMBASE—a user may on demand generate a proof version of character glyphs, e.g., for documentation purposes. Appreciating Knuth's idea of literate programming, we tried to implement it in MetaType1. We wanted METAPOST sources to be simultaneously the ultimate documentation. The MFT utility from the canonical TEX package fits here very well. We slightly enhanced the TEX macros that accompany the program (`mftmac.tex`) in order to facilitate self-documentation. The altered version allows one to include easily a proof version of the glyphs into the formatted source. Figure 2 shows what such a documentation looks like. The displayed sample page is taken from the source of Knuth's LOGO font adapted to MetaType1.

We tried to keep the font base as independent of the Type 1 specification as possible, although, as was mentioned, some peculiarities of Type 1 cannot be ignored. Anyway, we hope that in the (far) future, when Type 1 fonts are finally superseded by a

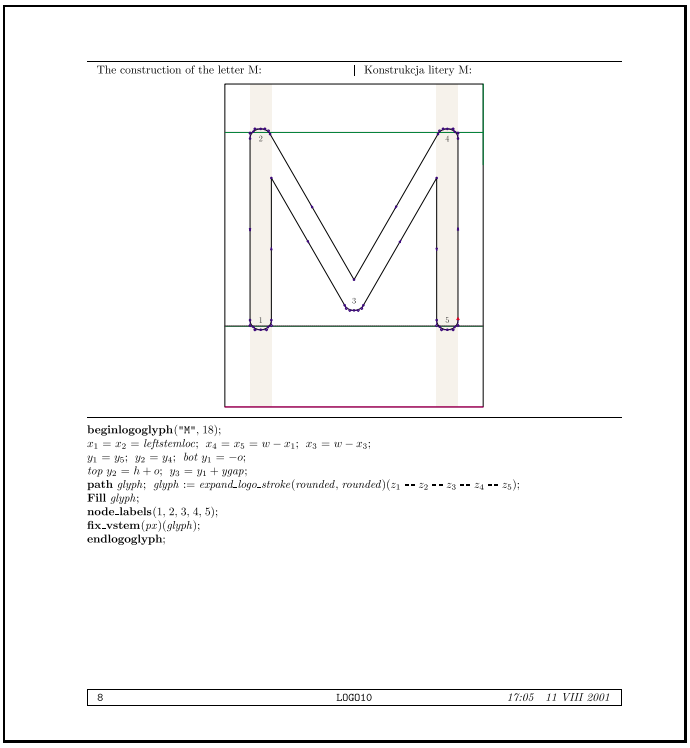

FIGURE 2: AN EXAMPLE OF A SELF-DOCUMENTING METATYPE1 SOURCE.

"Brave New Format," appropriate modification of the base will not be an exceedingly difficult task.

Step 2: AWK *processing*

The main duty of the AWK module is to convert glyphs (EPS) from plain POSTSCRIPT to a form required by the TYPE 1 specification (and accepted by T1UTILS). The TYPE 1 format accepts only integer numbers, therefore rounding is necessary (a user should control the process of rounding at crucial points in METAPOST programs). The only exception is the width of a character. A non-integer width is replaced not by a single number but by a division operation which is allowed in TYPE 1. For example, the width of the character '`T`' from the LOGO10 font is 5.77776 pt, i.e., 577.776 in TYPE 1 grid units; this quantity is represented in the resulting PFB file by '`17911 31 div`' which yields ≈ 577.7742. Appropriate numerators and denominators are computed by means of continued fractions.

In TYPE 1, all dimensions have to be given in relative units, while plain POSTSCRIPT (as output by METAPOST) uses absolute coordinates. Conversion to relative coordinates is also done by AWK.

Perhaps the most complex part of the AWK job is arranging the data properly for hinting. Hints are Type 1 commands that control the discretization of outlines. In a METAPOST source, a user specifies relevant paths and stem values; METAPOST finds acceptable coordinates for the stems and writes this information (embedded as structured comments) into the EPS files. This, however, is not the end of the story. The Adobe Type 1 specification requires that *no hints* of the same kind (horizontal or vertical) *can overlap*. If such a situation occurs, a special routine, called *hint replacement*, should be launched ([8], pp. 69–71). The AWK script does its best to prepare the data properly for the hint replacement process. But it uses some heuristics, therefore the applied algorithm may fail (rather unlikely under normal circumstances).

The result of the AWK run is a disassembled ("raw") PFB file and an AFM file; moreover, ENC and MAP files, to be used with a DVIPS driver, are generated. Optionally, prior to assembling the "raw" PFB file can be processed again by AWK. During this pass, another AWK script is used to search for repeated fragments of code; subroutine definitions are added for such fragments and all their occurrences are replaced by the respective subroutine calls. Usually, this process shortens the PFB file by some 10%. Besides optimization, it also provides an audit of the font, e.g., accented letters should afterwards contain only subroutine calls and no explicit drawing commands.

The AWK stage of font generation is entirely PostScript-oriented. For a different output font format the AWK scripts would have to be rewritten nearly from scratch.

Step 3: Assembling the PFB *file*
This is the simplest step of all: a one-to-one conversion from the disassembled to the final (binary) version of the PFB file is done by a stand-alone freeware utility, T1ASM.

Step 4 (optional): generating the PFM *file*
As was mentioned, PFM files are required if the Type 1 fonts are to be installed on a Windows system. A PFM file contains similar information to that contained in the AFM file, character dimensions, kerning, etc. The main difference is that a PFM file does not contain glyph names, therefore the encoding must be specified in addition. There is a secret byte in a PFM file (85th counting from 0) that contains the relevant information; e.g., the value 238 denotes East European encoding, 206 Central European. Can you guess why? It's simple: $238_{10} = \mathrm{EE}_{16}$, $206_{10} = \mathrm{CE}_{16}$. In order to generate a PFM file conforming to a particular Windows installation, one has to know which number is appropriate. It cannot be excluded that more bombshells of that kind await Windows users. Fortunately, the PERL script is fairly legible and can easily be adjusted if needed.

End or start?

Although we have been working on MetaType1 for a few years, only recently has it stabilized sufficiently to make it available publicly. We must warn potential fearless MetaType1 users, however, that our experience with the package is limited to one complete font (Antykwa Półtawskiego), a few geometric symbol fonts, several improved fonts and one experiment: while preparing this paper we tested MetaType1 against

Knuth's LOGO font. Within *three working days* we were able to modify the METAFONT sources and adjust them to the requirements of METATYPE1. The modified LOGO font is enclosed with the METATYPE1 distribution package. It should be emphasized that the resulting TFM files are 100% compatible with the original ones.

Needless to say, the experiment also unveiled the existence of a few bugs in METATYPE1, both in the METAPOST and AWK parts. Therefore, we consider the public release of METATYPE1 as the start of a new phase rather than the completion of the design process.

Users' feedback cannot be underestimated in this respect. We count upon users' contributions, although we cannot promise a royal road to creating fonts; instead, we can promise satisfaction once a font is ready. We can also assert that programming a font is not reserved for Donald E. Knuth—so, let us go forth now and create masterpieces of digital typography in TYPE 1 format.

REFERENCES

[1] Knuth, D. E., *The TEXbook*. Addison-Wesley, eleventh printing, 1990.

[2] Knuth, D. E., *The METAFONTbook*. Addison-Wesley, seventh printing, 1992.

[3] Hobby, J. D., *The METAPOST Page*. http://cm.bell-labs.com/who/hobby/MetaPost.html

[4] Aho A. V., Kernighan B. W., Weinberger P. J., *The AWK Programming Language*. Addison-Wesley, 1988.

[5] *Type tools*. http://www.lcdf.org/~eddietwo/type/#t1utils

[6] *POSTSCRIPT Language Reference Manual, 3rd Edition*. http://partners.adobe.com/asn/developer/PDFS/TN/PLRM.pdf

[7] *Adobe Font Formats and File Types*. http://partners.adobe.com/asn/developer/typeforum/ftypes.html

[8] *Adobe TYPE 1 Font Format*. Addison-Wesley, 1990. http://partners.adobe.com/asn/developer/pdfs/tn/T1_SPEC.PDF

[9] *TrueType Specification, ver. 1.3*. http://www.microsoft.com/typography/tt/ttf_spec/ttspec.zip

[10] *OpenType specification, ver. 1.3 (last update: April 2001)*. http://www.microsoft.com/typography/otspec/otsp13p.zip

[11] *TrueType to PS Type 1 Font Converter*. http://sourceforge.net/projects/ttf2pt1/

[12] Baron, D., *A TRUETYPE to TYPE 42 Converter*. http://ftp.giga.or.at/pub/nih/ttftot42

[13] *GNU AWK User's Guide*. http://www.gnu.org/manual/gawk/index.html

[14] *GHOSTSCRIPT, GHOSTVIEW and GSVIEW*. http://www.cs.wisc.edu/~ghost/

[15] Bzyl, W., *Reitroducing TYPE 3 fonts to the TEX world*. Proc. of EuroTEX 2001, 24th–27th September, 2001, Kerkrade, The Netherlands.

[16] Jackowski, B., Nowacki, J. M., Strzelczyk, P., *Antykwa Półtawskiego: A Parameterized Outline Font*. Proc. of EuroTEX'99, 20th–24th September, 1999, Heidelberg, Germany, pp. 109–141.

[17] Jackowski, B., Nowacki, J. M., Strzelczyk, P., *Localizing Type 1 Fonts from Ghostscript Distribution*. BachoTEX'2001, 9th GUST Conference, 29th April–2nd May, 2001, Bachotek, Poland,
`http://www.gust.org.pl/BachoTeX/2001/GSFONTS.PDF`

[18] Kinch, R. J., *MetaFog: Converting METAFONT Shapes to Contours*. TUGboat **16** (3), pp. 233–243, 1995.

[19] Kinch, R. J., *Belleek: A Call for METAFONT revival*. Proc. of 19th Annual TUG Meeting, AuGUST 17–20, 1998, Toruń, Poland, pp. 131–136.

[20] Hoekwater, T., *Generating Type 1 Fonts from METAFONT Sources*. Proc. of 19th Annual TUG Meeting, AuGUST 17–20, 1998, Toruń, Poland, pp. 137–147.

[21] Haralambous, Y., *Parametrization of PostScript Fonts Through METAFONT—an Alternative to Adobe Multiple Master Fonts*. Electronic Publishing, **6** (3), pp. 145–157, 1993.

[22] Malyshev, B. K., *Problems of the Conversion of METAFONT Fonts to PostScript Type 1*. TUGboat, **16** (1), pp. 60–68, 1995.

[23] Yanai, S., Berry D. M., *Environment for Translating METAFONT to PostScript*. TUGboat **11** (4), pp. 525–541, 1990.