Programming PostScript Type 1 Fonts Using METATYPE1: Auditing, Enhancing, Creating

Bogusław Jackowski
ul. Tatrzańska 6 m. 1, 80-331 Gdańsk, Poland
B_Jackowski@gust.org.pl

Janusz M. Nowacki
ul. Śniadeckich 82 m. 46, 86-300 Grudziądz, Poland
J.Nowacki@gust.org.pl

Piotr Strzelczyk
ul. Subisława 14, 80-354 Gdańsk, Poland
P.Strzelczyk@gust.org.pl

Abstract

METATYPE1 is a METAPOST-based package for producing outline PostScript fonts in the Type 1 format. Since its first unofficial release a few years ago, it has been applied to several tasks. We would like to share our experience with other fans of computer typography.

Résumé

METATYPE1 est un ensemble d'outils basé sur METAPOST pour produire des fontes PostScript de type 1. Depuis sa première sortie, il y a quelques années, il a eu un grand nombre d'applications. Nous souhaitons partager notre expérience avec d'autres amateurs de typographie informatique.

Introduction

We started the work on METATYPE1 some five years ago. Outline fonts had already been in vogue then, especially in electronic publishing. Being adherents of both computer typography and programming, we urgently needed a programming tool for generating outline fonts. Fortunately, neither TEXtrace [7] nor FontForge [9] existed at that time. We highly esteem both of these tools and therefore we would perhaps have given up — which would be a pity, because they do not comprise all possible application areas. Sometimes, full programmability is preferable. Anyway, we had no choice but to develop a home-grown engine.

In this paper, we describe a few sample tasks that we had the opportunity to accomplish using METATYPE1, thus demonstrating the pros and cons of the programming approach to generating of outline fonts.

A brief overview of METATYPE1

As the name (and logo) suggests, METAPOST is the kernel of METATYPE1. In other words, METATYPE1 sources are written in the METAPOST language. METATYPE1 provides a task-oriented library of METAPOST macros. It also makes use of popular open source tools: Gawk and T1Utils. The METAPOST output is processed by Gawk and then assembled by T1Utils (figure 1, left). The details of the process of generating of fonts using METATYPE1 can be found in [4].

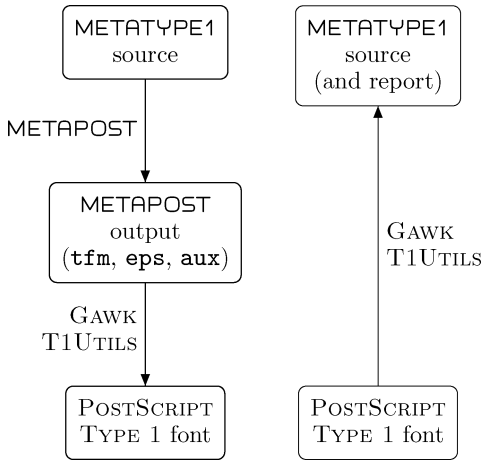

Fig. 1: A simplified scheme of the METATYPE1 engine (comments in the text).

An important part of the METATYPE1 package is a (freestanding) converter from the PostScript Type 1 format to METATYPE1 source, `pf2mt1` (figure 1, right), which also makes use of Gawk and T1Utils. As we shall see, it proves useful for auditing of fonts.

The installation of the METATYPE1 package is straightforward, once Gawk, T1Utils, and METAPOST have already been installed — the files should be copied to a chosen directory and a system variable `METATYPE1` should point to this directory. Originally, METATYPE1 was developed under DOS; its provisional Linux port is maintained by Włodek Bzyl (see [4] for the url). A simplified version of the Linux scripts is available from the authors; the scripts will be included in the next release of METATYPE1.

Many roads to the creation of fonts

Since we are no font designers whatsoever, we will assume that the design of a typeface is given. Still, the source of a font may be given in various forms — from lead prints to tentatively ready digital form. In the former case, one has to create a digitized version from scratch; in the latter, an audit of a font may prove sufficient. Below, we discuss a few different approaches to coping with fonts, starting from the simplest one, that is, from auditing.

Audit A useful yet a very simple tool for auditing of a font is the already mentioned converter from PostScript Type 1 format to human readable (and thus machine readable) METATYPE1 sources. We assume, of course, that the disassembling of a given font is legal.

Faulty hinting is perhaps most typical in freely available fonts. Even such diligently prepared fonts as the *Computer Modern* family in the PostScript Type 1 format, released as a freely available product by the American Mathematical Society in 1997, contain questionable hints.

Prior to the presentation of an example, let us briefly sketch the idea of the PostScript Type 1 hinting. Hints are special instructions, allowed only in a font program, providing additional information useful during the process of rendering of a glyph. There are two kinds of hints: vertical and horizontal; both convey information concerning the preferred width and position of a stem (which, in general, may have a varying width).

Consider now, as an example, the font `cmsy10` from the AMS *Computer Modern* family. The converter `pf2mt1` produces METATYPE1 sources, and, which is important here, a log file; here is an excerpt from the log (the first number denotes the position of a stem, the second one — width):

```
...
intersectionsq: semi-matching hstem; 558, 45
intersectionsq: non-matching vstem; 55, 40
intersectionsq: semi-matching vstem; 571, 34
```

The log indicates that there is something wrong with hints: why should such a regular character as *intersectionsq* have hints of different width (see figure 2)?

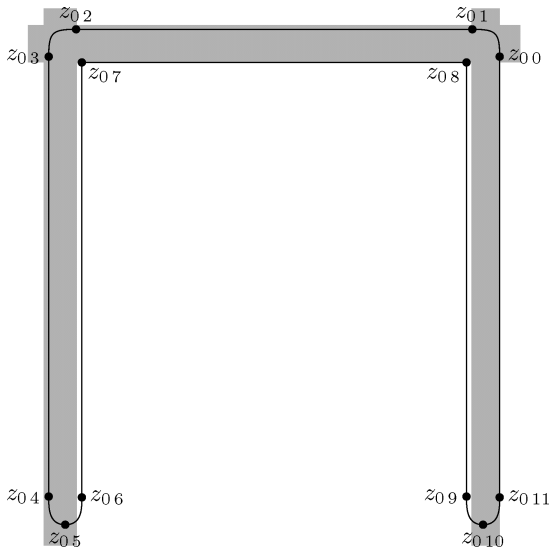

Fig. 2: Character *intersectionsq* from `cmsy10` from the collection provided by the AMS; gray rectangles mark the placement of hints — it can be seen that they do not precisely coincide with the respective stems.

The inspection of the code of the glyph *intersectionsq* reveals, as one would expect, that each stem has the same width (equal to 40; compare the following pairs of nodes: z_{04} and z_{06}, z_{09} and z_{011}, z_{08} and z_{01}):

```
beginglyph(_intersectionsq);
 save p; path p[];
 z0 0=(605,565); z0 0a=(605,594); z0 1b=(601,598);
 z0 1=(572,598);
 z0 2=(94,598); z0 2a=(65,598); z0 3b=(61,594);
 z0 3=(61,565);
 z0 4=(61,34); z0 4a=(61,20); z0 5b=(61,0);
 z0 5=(81,0); z0 5a=(101,0); z0 6b=(101,19);
 z0 6=(101,33);
 z0 7=(101,558);
 z0 8=(565,558);
 z0 9=(565,34); z0 9a=(565,20); z0 10b=(565,0);
 z0 10=(585,0); z0 10a=(605,0); z0 11b=(605,19);
 z0 11=(605,33);
 p0=compose_path.z0(11);
 correct_path_directions(p0)(p);
 if turningnumber p0>0: Fill else: unFill fi \\ p0;
 set_hstem (558,603); % stem width = 603 - 558
 set_vstem (55,95);   % stem width = 95 - 55
 set_vstem (571,605); % stem width = 605 - 571
 ghost_stem bot;
 standard_exact_hsbw("intersectionsq");
```

```
endglyph;
```

Note that the program is fairly legible and that even a superficial knowledge of the METAPOST notation suffices to guess what is wrong (here: with hints).

Among others, we tested our `pf2mt1` converter against a rich collection of fonts provided by Vladimir Volovich, namely, CM-Super [10]. The collection was prepared using TEXtrace [7] and tuned manually. The job Volovich did is really impressive. Still, a conversion to METATYPE1 sources revealed a couple of defects in nearly every font. It is not the proper place to analyse minutely the results of the conversion, we therefore confine ourselves to only one aspect: strange tiny paths ("scraps") that we spotted in about 25% of the fonts, on average in two characters per font. Two examples of such paths are depicted in figure 3. Note that the "scraps" are moderately harmful and that their (rare) occurrences can be expected only in autotraced fonts.

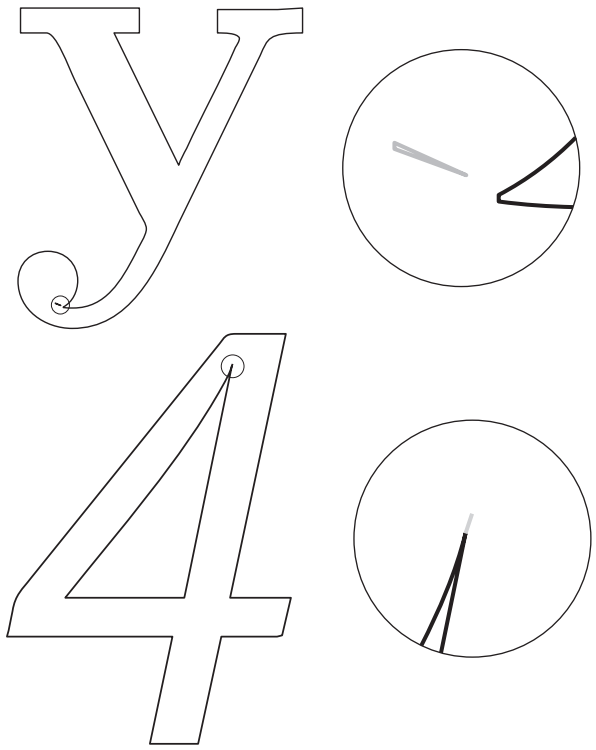

FIG. 3: Two examples of strange "scraps" (coloured gray in the magnified details) occurring in CM-Super fonts: character *y* from `sfbx0900` (top) and character *four* from `sfsi1095` (bottom).

Finding these weird paths was straightforward: we simply wrote a short Awk script that looked for tiny 3-node paths in METATYPE1 sources. For example, the relevant path in the glyph *y* from `sfbx0900` looked as follows:

```
z1 0=(98,-149);
z1 1=(98,-148);
z1 2=(109,-153);
p1=compose_path.z1(2);
```

Again, a superficial knowledge of the METAPOST notation suffices — even not knowing in detail how the operator `compose_path` works — one can easily detect such questionable objects. Note that in the case of the glyph *four* from `sfsi1095`, it would be nearly impossible to notice the superfluous path in the screen or even in a printed proof, while in the METATYPE1 source it is easy.

Let us emphasize that we point out the flaws of the CM-Super family not to deprecate it; we simply want to turn the reader's attention to the fact that METATYPE1, or its parts, can be used as an aid for different font creation software.

One can conceive of more advanced techniques of auditing, for example, using task-oriented METAPOST macros written specially to deal with a given case. But even without employing advanced techniques, many clues can be surmised from the report written during the conversion and from the inspection of the resulting METATYPE1 code.

Elementary enhancement An interesting lesson can be derived from the previous section: some improvements of a font are straighforward. For example, the elimination of the tiny "scraps" is just trivial. The correction of hinting in the program for the glyph *intersectionsq* from the font `cmsy10` is also not difficult. The proper code should read:

```
set_hstem (558,698);
set_vstem (61,101);
set_vstem (565,605);
```

or, equivalently (and more advisably):

```
set_hstem (y0 8,y0 1);
set_vstem (x0 4,x0 6);
set_vstem (x0 9,x0 11);
```

Thus, even a moderately experienced user should be reasonably able to modify METATYPE1 sources and to generate improved fonts in certain simple cases.

Another method of enhancing a font, even simpler than that of requiring the conversion to METATYPE1 sources and the examination of the results, is to use a compressing module from the METATYPE1 package. This module (actually, a short Awk script) performs an analysis of a disassembled PostScript Type 1 font (a PFB file), defines subroutines for repeating fragments of the code, and replaces the occurrences of these fragments by the respective subroutine calls. The method employed is based on an algorithm for finding the longest repeating substring of a given string. The "subroutinization" is one

of a few steps of the whole process of the generation of a font, but it can be easily disentangled.

Usually, the compression is not astoundingly efficient, for example, the fonts from the AMS *Computer Modern* family can be compressed only by 2–3%. The size of the CM-Super family of fonts, however, can be reduced by about 12% (7 MB) with this method. It is not much, but, on the other hand, it is not nothing.

Advanced enhancement The potential improvements discussed so far require only basic knowledge of both typography and METAPOST programming. Therefore, it is not a surprise that the likely results are only moderately significant — in order to achieve more, one has to know more. It is not extremely difficult, however, to obtain really useful results knowing just a little more than the rudiments.

In typical cases, an augmentation of a character set with accented characters is needed. This is a task that can be accomplished relatively easily using METATYPE1; namely, there is an operation `use_accent`, defined in the set of basic METATYPE1 macros, for precisely this purpose.

We have used this technique while preparing the collection of *Latin Modern* fonts [2]. Our starting point was, obviously, the AMS *Computer Modern* family of fonts converted to METATYPE1 format. We programmed most of the accented characters by adding just three lines of code per glyph, for example:

```
beginglyph(_aacute);
 use_accent(_a,_acute);
endglyph;

beginglyph(_abreve);
 use_accent(_a,_breve);
endglyph;

beginglyph(_acircumflex);
 use_accent(_a,_circumflex);
endglyph;
```

etc.; altogether, almost 200 characters of the *Latin Modern* family have been "programmed" using such three-liners.

In some cases, however, additional settings were needed. By default, the `use_accent` operator works similarly to the TEX `\accent` primitive — it just centres an accent over an accented character. This procedure is not adequate for such characters as *Lacute*. METATYPE1, unlike TEX, provides an optional (numeric) parameter for each character, `glyph_axis`, describing the position of the glyph axis to be used with accents. The axis is aligned either with the centre of an accent or with its axis, if it happens to be set.

For example, the program for the letter *L* from the font `lmb10` assigns the value of the x-coordinate of the central node of the top serif to the variable `glyph_axis` (see figure 4):

```
beginglyph(_L);
 save p; path p[];
 glyph_axis=x0 7; % manually added line
 z0 0=(643,274);
 z0 1=(596,274); z0 1a=(588,205); z0 2b=(571,47);
 z0 2=(392,47);
 z0 3=(289,47);
 z0 4=(289,639);
 z0 5=(424,639);
 z0 6=(424,686); z0 6a=(380,683); z0 7b=(271,683);
 z0 7=(222,683); z0 7a=(178,683); z0 8b=(77,683);
 z0 8=(39,686);
 z0 9=(39,639);
 z0 10=(147,639);
 z0 11=(147,47);
 z0 12=(39,47);
 z0 13=(39,0);
 z0 14=(612,0);
 p0=compose_path.z0(14);
 if turningnumber p0>0: Fill else: unFill fi \\ p0;
 set_hstem (0,47);
 set_hstem (639,686);
 set_vstem (596,643);
 set_vstem (147,289);
 standard_hsbw("L");
endglyph;
```

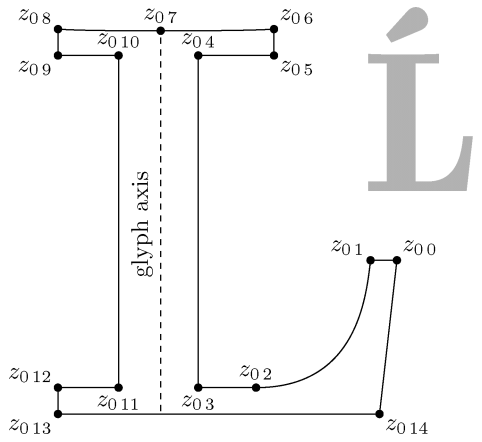

FIG. 4: An example of the setting of a glyph axis in a character *L* from a *Latin Modern* font (`lmb10`); such an axis facilitates the placement of accents in asymmetric characters such as *Lacute*.

Having adequately defined the axis for a given character (and for relevant accents, if needed), the accented characters can be defined using our neat three-liners, for example:

```
beginglyph(_Lacute);
 use_accent(_L,_Acute);
endglyph;
```

(Note that a variant acute accent is used here — accents over capital letters and ascenders are flattened a little bit in *Latin Modern* fonts. It is one of many details that can be conveniently controlled using the programming approach.)

The programming of characters that contain diacritical elements such as *ogonek* or *cedilla* is more arcane, since it requires finding the common outline of the character and diacritic. For this purpose, METATYPE1 provides the operation `find_outlines`. Although fairly handy in practice, it is not as robust as one would wish, because it behaves unstably if the curves to be processed are tangent or nearly tangent. In general, tangency presents an intrinsic problem, because it is impossible to distinguish *numerically* between tangent curves and intersecting ones. Metaphorically speaking, tangency is an infinitesimal property, while computers are discrete, and thus finite.

With METAFONT, things become even worse. For example, the built-in METAFONT operator `intersectiontimes`, which we need, sometimes cheats when disjoint curves are touching each other. Another operator, `turningnumber`, important in dealing with paths, also is not sufficiently reliable (due to, for example, "tiny loops" — see [5]).

METAFONT has been designed to produce bitmaps, not outlines. In such applications, curves that nearly touch each other or tiny loops that change unpredictably the turning number of a curve do not cause too much harm. Its descendant METAPOST, although it is no longer bitmap-oriented, inherited METAFONT arithmetic with all its imprecision; the same problems can be thus likely encountered in METAPOST. One can live with that but the price is the necessity of carefully controlling the situations where nasty tangency-related issues emerge. It is worthy of emphasizing that although such control requires some proficiency in programming in METAPOST, it is usually effortless.

The most complex aspect of a font enhancement is obviously constructing a glyph from scratch. This topic, in fact, has more to do with the creation than with the enhancement of a font. Thus, we postpone it for a while.

From METAFONT to METATYPE1 Although the issue of an automatic conversion of METAFONT font programs into an outline form conforming to PostScript Type 1 standards was recognised relatively long ago, it has still not been fully solved.

A partial solution was announced by Péter Szabó during the EuroTEX 2001 meeting (Kerkrade, The Netherlands [7]).

Szabó's TEXtrace program produces outlines of acceptable quality, thanks to the efficacious Autotrace program [8] by Martin Weber, which is employed for the autotracing of high-resolution bitmaps. Nevertheless, we call Szabó's solution only "partial" since the process of autotracing cannot (yet?) be controlled in detail, while we firmly believe that every detail should be controllable and replicable in the realm of TEX and METAFONT/METAPOST.

Another solution which we prefer, but which might be called "partial" as well, is a manual alteration of METAFONT sources. We applied this approach to generate a METATYPE1 version of Donald E. Knuth's `logo` font. A sample page from the automatically generated documentation of the font is shown in figure 5.

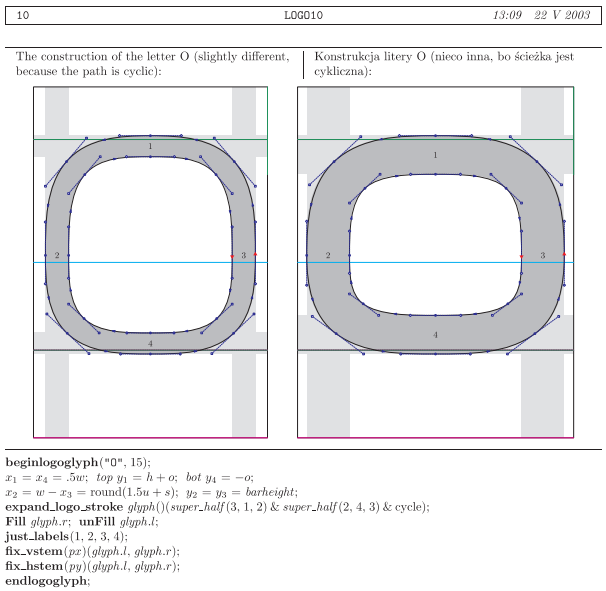

Fig. 5: A sample page from the automatically generated documentation of the METATYPE1 version of the `logo` font; light gray rectangles mark hints; a high-resolution bitmap generated from the original METAFONT sources (painted with a darker gray) is placed in the background for comparison.

We would like to stress that it was a purely experimental effort, because Taco Hoekwater had already prepared a PostScript Type 1 version of the `logo` font using Richard J. Kinch's MetaFog. The differences of glyph shapes between Hoekwater's fonts and ours are negligible. More interesting is perhaps that we managed to obtain metric files (tfm) identical with the original. Those who are interested in technicalities may wish to download both the sources and the results from the same ftp address as the METATYPE1 package. A conversion of Hoekwater's PostScript Type 1 fonts into METATYPE1 sources using `pf2mt1` can be also instructive.

In the *Latin Modern* family, several glyphs were also prepared using this method. In particular, we adapted

the program for the letter *C* from the *Computer Modern* METAFONT sources in order to construct the *Euro* currency symbol (with the addition of a bottom serif, since the *Euro* symbol is philosophically based on a script *E*; thanks to Werner Lemberg).

The point of the story is that, in principle, such a conversion is possible. It remains an open question, however, whether it is worthwhile to attempt to convert all available METAFONT sources. On one hand, such outline-oriented programs would certainly be useful, especially for those who would like to generate new variants of old fonts by playing around with parameters; on the other hand, such a conversion is a laborious task, although in most cases it is routine. In our opinion, however, converting of existing METAFONT sources into an outline form is not as urgent as the problem of the scarcity of new fonts. But the creation of a new font is a challenge, indeed.

Creation The issue of font creation, even limited to the computer-oriented aspects, is a topic rather for a book than for a section in a short article. Therefore, we will not dwell too much on this inexhaustible subject. Instead, we simply sum up the experience we gathered during the work on two replicas of Polish traditional typefaces (figure 6): Antykwa Półtawskiego [3] and Antykwa Toruńska, the latter being intensively developed by Janusz M. Nowacki.

Antykwa Półtawskiego
Antykwa Półtawskiego

Antykwa Toruńska
Antykwa Toruńska
Antykwa Toruńska
Antykwa Toruńska

Fig. 6: Two replicas of Polish traditional typefaces, prepared with METATYPE1.

For both typefaces, only lead prints were available. In many cases, therefore, we had to conjecture as to the details. Surprisingly, this seemingly difficult task of reconstructing and programming of glyph shapes, turned out to be relatively simple. The adjusting of sidebearings and kerns was significantly more difficult. But it is the parameterization of a font that is really a tough problem. We do not want to juggle lots of incoherent parameters. Frankly speaking, we would not be able to. Recall that the *Computer Modern* fonts are governed by more than 60 parameters — this would be certainly too much for us.

After many experiments and discussions, we are still unsatisfied with the current parameterization of Antykwa Półtawskiego. This is the main reason why we have not publicly released its METATYPE1 sources so far. We have not given up yet — we certainly hope that before long we will eventually release tolerably tidy METATYPE1 sources. On demand, however, they can be obtained from the authors for private use.[1]

We have not heard about other book typefaces created using METATYPE1. Our program has rather a small number of users and is mainly employed for auditing and enhancing fonts, although we know about a few fonts containing geometrically regular glyphs (for example, geographical symbols) created using METATYPE1; among them, perhaps the most interesting is a humorous font displayed in figure 7.

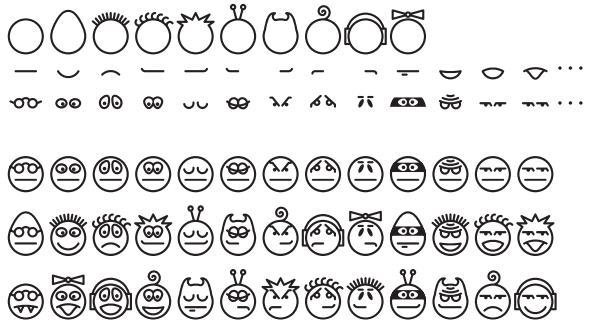

Fig. 7: A dingbats font created with METATYPE1 by Jacek Kmiecik: the top three rows contain characters of the font, the bottom rows present an effect of superimposing of three characters (one from each row); the inspiration for this work was the font `Head-DingMaker`, signed by Nigma Fonts (`kentpw@norwich.net`).

From a TEX user's point of view, it is important that METATYPE1 is suitable for creating TEX-oriented math fonts (note that the popular `afm2tfm` program cannot be used for generating math tfm files). We know about a couple of such endeavours. In one of them we are involved: it is an ongoing project of the Polish TEX Users Group GUST of furnishing Antykwa Półtawskiego with a math extension. So far, however, nobody has announced a release of a math font created with METATYPE1. The bright side of this otherwise disquieting situation is that interesting events are still ahead of us.

1. In the near future, most probably before EuroTEX 2005, the METATYPE1 sources of Latin Modern family of fonts will be released publicly.

All in all, METATYPE1 promises well as far as we can tell. But please feel forewarned that we are incorrigible optimists. Also feel forewarned, as we emphasize repeatedly, that although there are many roads to the creation of fonts, the royal road does not exist.

Concluding remarks

We took for granted the PostScript Type 1 format is exactly what we need. We are aware that the future belongs to a successor of Type 1 and TrueType, namely, to OpenType, but we do not consider it a threat. Let us quote a Microsoft faq [6]: *OpenType* [...] *is an extension of Microsoft's TrueType Open format, adding support for Type* 1 *data.* Moreover, there is an official *Font Development Kit* for OpenType released by Adobe [1] which offers, among several less or more useful functions, a conversion of PostScript Type 1 fonts into their OpenType counterparts. We bumped into some obstacles trying to tame this tool, but finally managed to achieve the desired outcome. The results are not easily verifiable because OpenType, advertised since 1996, actually entered the scene only a few years ago. Nevertheless, it seems that such a conversion may be an appropriate solution for some time to come.

We also have tried FontForge as a converter to OpenType fonts. Again, we were successful, although not without trouble. It is too early, however, to evaluate the existing tools definitively, the more so as other tools of this kind can be expected to become available in the nearest future.

Is METATYPE1 a convenient tool for creating fonts at present and in the future? The answer is up to users. Certainly, it is a tool for creating *open source fonts*, that is, *truly open types.*

Acknowledgements

Very many thanks to Jerzy Ludwichowski for his kind support during the preparation of this paper.

References

[1] *Adobe Font Development Kit for OpenType*, http://partners.adobe.com/asn/developer/type/otfdk/

[2] Bogusław Jackowski, Janusz M. Nowacki, *Enhancing Computer Modern with accents, accents, accents*, *TUGboat* 24(1), Proc. of the 24th Annual Meeting and Conference of the TEX Users Group, p. 64–74.

[3] Bogusław Jackowski, Janusz M. Nowacki, *Antykwa Półtawskiego: a parameterized outline font*, EuroTEX 1999, 20th – 24th September, 1999, Heidelberg, Germany, pp. 109 – 141; the current version of Antykwa Półtawskiego is available from ftp://ftp.GUST.org.pl/pub/TeX/GUST/contrib/fonts/replicas

[4] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *METATYPE1: A METAPOST-based Engine for Generating Type* 1 *Fonts*, Proc. of EuroTEX 2001, 27th – 27th September, 2001, Kerkrade, the Netherlands, pp. 111 – 119; the current version of METATYPE1 is available from ftp://bop.eps.gda.pl/pub/metatype1; METATYPE1 for Linux can be downloaded from ftp://ftp.ctan.org/tex-archive/systems/unix/mtype13/

[5] Donald E. Knuth, *The METAFONTbook*, Addison-Wesley, edition VII, 1992, pp. 152, 228 – 229.

[6] *OpenType initiative FAQ*, http://www.microsoft.com/truetype/faq/faq9.htm

[7] Péter Szabó, *TEXtrace*, http://www.inf.bme.hu/~pts/textrace/

[8] Martin Weber, *Autotrace*, http://autotrace.sourceforge.net/

[9] George Williams, *FontForge — a PostScript Font Editor*, http://fontforge.sourceforge.net/

[10] Vladimir Volovich, *CM-Super Font Package*, ftp://ftp.vsu.ru/pub/tex/font-packs/cm-super/