Creation of a PostScript Type 1 logo font with MetaType 1

Klaus Höppner
Haardtring 230 a
64295 Darmstadt
Germany
klaus dot hoeppner (at) gmx dot de

Abstract

MetaType 1 is a tool created by Bogusław Jackowski, Janusz Nowacki, and Piotr Strzelczyk for creating PostScript Type 1 fonts. It uses METAPOST, t1utils and some AWK scripts to start from a METAPOST source with some special macros, resulting in the AFM, TFM and PFB files needed to use the font as any other PostScript font.

MetaType 1 was used to create the Latin Modern fonts, derived from Computer Modern fonts but including many more accented characters and nowadays part of most TeX distributions. Other new fonts such as Iwona and Kurier have also been created by the developers of MetaType 1.

I came into contact with METAPOST when I wanted to convert an existing logo font from METAFONT to PostScript Type 1. Unfortunately there doesn't yet exist a tutorial or cookbook for using MetaType 1. So I started to play with the example fonts supplied as part of MetaType 1 and to read the comments in the source. This tutorial will give an example and the lessons I learned.

1 Introduction

When Donald E. Knuth invented TeX, he also created his own description language for high quality fonts. It was named METAFONT. So the process from a TeX source to some paperwork was as follows: Compile the TeX source to get a DVI file that contains references to the fonts that were used in the document — in fact the only thing that TeX knows about a font is its metrics. To produce the document on paper, the DVI driver invoked METAFONT (the program) to convert the METAFONT source of the font, i. e. the geometrical description of the font outlines, to a bitmapped font suited for the resolution and technical details of the printer by using the METAFONT mode for this special printer.

While this approach works fine if you work alone and just send your documents to your personal printer, it has some disadvantages if you want to exchange documents electronically. Normally, distributing DVI isn't the best idea, since it requires that the recipient has a TeX system installed including all fonts that were used in your document — not to mention any graphics included in your document. So in most cases you will send a PostScript file or nowadays a PDF file. In this case, all the fonts from METAFONT sources will be embedded as bitmapped PostScript Type 3 fonts. When the recipient prints your document, it may look fine, but it may look poor if the METAFONT mode used to create the bitmapped font didn't match the printer, and the document will probably look very poor on the screen (especially in old versions of Acrobat Reader).

So when exchanging documents, it is preferable to embed the fonts as outline fonts. For these, the usual format used in the TeX world is PostScript Type 1 (though this is gradually being replaced by OpenType). The Type 1 format uses a subset of the well established PostScript language.[1]

Meanwhile, most of the fonts used in the TeX world are available as PostScript Type 1 fonts, starting with the Type 1 version of Knuth's CM fonts up to the Latin Modern fonts that augment CM with a complete set of diacritic characters.

2 MetaType 1

MetaType 1 is the tool that was used to create the Latin Modern fonts from the METAFONT sources of CM fonts, and for the creation of completely new fonts such as Iwona.

MetaType 1 relies on METAPOST, a variant of METAFONT producing small pieces of PostScript as output, written by John Hobby. Bogusław Jackowski, Janusz Nowacki, and Piotr Strzelczyk wrote a set of METAPOST macros and added some AWK scripts to create the input files that can be con-

[1] It is sometimes said that Type 1 fonts are outline fonts while Type 3 are bitmap fonts. That's not true, since Type 3 fonts may comprise both outlines and bitmaps.

verted to Type 1 with t1utils. Thus, one advantage of MetaType 1 is that it uses a source format that is very similar to the old METAFONT sources.

3 Our example

I came into touch with MetaType 1 when I wasn't satisfied with the DANTE logo being typeset from the old METAFONT source with all the disadvantages mentioned above. So I wanted to give MetaType 1 a try to convert the DANTE logo font into a PostScript Type 1 font.

Fortunately, the DANTE logo font contains just the characters needed to set the logo:

dante

So, it was just five characters for which the METAFONT source had to be made suitable to be processed with MetaType 1.

Unfortunately, I found out that the available documentation for MetaType 1 was rather limited: articles from conference talks [1, 2], the commented source for the MetaType 1 macros and two sample fonts that are part of the MetaType 1 distribution.

But in the end, I found my way, and as you will see, was able to create my own Type 1 font. To make things a bit simpler for this tutorial, I will show the steps I made for a small test font with just two characters, "a" and "t", simplified compared to the original characters from the DANTE logo font. Hopefully it will make the presented source more understandable, even if you haven't programmed in METAPOST before.

3.1 Installation

Installing MetaType 1 was easy enough. I downloaded the ZIP archive file from CTAN [3] and copied the files to the appropriate locations of my local texmf tree: the `.mp` files into `metapost/mt1`, the `.mft` files into `mft/mt1`, the `.sty` files into `macros/generic/mt1`, and finally the `.awk` and `.dat` files into `scripts/mt1`.[2]

The main problem in my case was that MetaType 1 was shipped with a set of DOS batch files that are used to create the fonts, but I was using GNU/Linux. So I looked into these files to find out what they do — in fact they were rather simple, just calling METAPOST to produce a small PostScript file for every glyph in the font and then using some AWK scripts to merge and assemble these files into a raw PostScript font that is converted into PostScript Type 1 with t1asm (part of t1utils). So several immediate files and steps are involved, but the workflow is straightforward. Eventually, I wrote a small Makefile that does the job on a Unix system, as shown in listing 3. From this point, I could create the TFM, PFB and MAP files for a font with the command `make FONT=myfont`.

I also manually created an FD file for using the font in LaTeX. These files could all be installed into the appropriate locations inside a `texmf` tree. Testing of a font is convenient in pdfTeX since one can use a MAP file locally in a document using the `\pdfmapfile` primitive, while for a real font one normally will install the MAP file using the `updmap` script (or equivalent).

Listing 1: First definition of "a" and "t".

```
encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginglyph("a");
path pa, pb, pc;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle
     scaled (2radius-2strength) shifted z0;
pc = unitsquare xscaled strength
     yscaled 2radius shifted z1;
Fill pa;
unFill pb;
Fill pc;
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginglyph("t");
path pa, pb;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = height;
z3 = (hdist,height-3strength);
pa = z1
    -- (halfcircle rotated 180
         scaled 3strength shifted z0)
    -- (reverse halfcircle rotated 180
         scaled strength shifted z0)
    -- z2 -- cycle;
pb = unitsquare xscaled 5strength
     yscaled strength shifted z3;
Fill pa;
Fill pb;
fix_hsbw(2hdist+5strength,0,0);
endglyph;
```

[2] This location isn't required since these files aren't found by the Kpathsea library, but instead via an environment variable, but at least this location seemed to be meaningful.

3.2 The first font

After these prerequisites were done, I could start with my first font. I copied the file `tapes.mp` (a sample font that is part of the MetaType 1 distribution) into `myfont.mp`, found several settings with font parameters starting with `pf_info_*`, changed them where appropriate (font name, family, creator, etc.) and kept the rest unchanged.

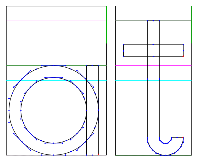
Then I defined the first two characters according to the following rule:

> *Characters consist of closed paths, filled or unfilled paths, where filled paths always turn counter clockwise and unfilled paths always clockwise.*

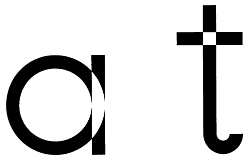
So when designing the letter “a”, I defined an outer circle that was filled and then an inner circle to be unfilled and then a rectangular shape as vertical stem. And the letter “t” was just built from a vertical stem (with a hook at the right bottom) and a horizontal bar. The definitions for the characters are shown in listing 1.

Please notice in the definition of letter “a”, that the path for the outer circle is a (counter clockwise) fullcircle, while the inner circle is a reverse fullcircle, since the former one is filled while the latter one is unfilled. Filling and unfilling of the paths is done by the macros `Fill` and `unFill`; these macros warn you if the turning direction of the path is wrong.

Proofs for the glyphs are produced by compiling the file `myfont.mp` with METAPOST. As you can see, they really do look like an “a” and a “t”:



Now let’s see how the Type 1 font looks:



Something went wrong. After taking a closer look, it becomes obvious. The regions where filled paths overlap become unfilled. This is due to the fact that filling of paths is done with an *exclusive-or* fill, i. e. when filling a path, regions inside that are already black become white. As this isn’t what we want to achieve, we formulate another rule to keep in mind:

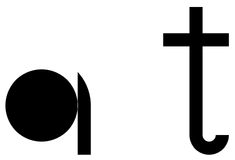
> *Paths must not overlap!*

Although it is possible with pure METAPOST to find the intersection points of paths to remove overlapping parts, this tends to be painful. Since MetaType 1 was used to attach cedilla and ogonek accents to various characters in the extension of CM to LM, this painful work of finding the outline of two overlapping paths was encapsulated into a macro that is part of MetaType 1, named `find_outlines`. Let’s see how this macro is utilized for the letter “a”:

```
find_outlines(pa,pc)(r);
Fill r1;
```

It finds the outline of the two overlapping paths `pa` and `pc`, with the result written in the path array `r`. The result is an array because the outline of the paths may consist of more than one path, but in our case it is just one path, accessible as `r1`. The same is applied for the letter “t” (just the names of the two paths slightly differ).

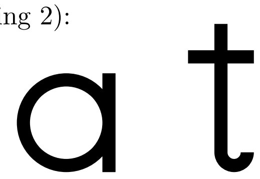
When filling the new outlines instead of the overlapping paths, we now get the following result:



So, obviously finding the outline path for the “t” worked, but it failed for the “a”. Why? Because in the case of the “a”, both paths touch in one point without crossing at the right side of the vertical stem, i. e. they have an intersection point with the same direction vector. This confuses the macro that finds the outlines since it doesn’t know which path to follow — and in this case it chooses wrong. So, let’s bear in mind another rule:

> *Paths must not touch tangentially!*

To resolve the problem, we use a simple trick: Shift the vertical stem a tiny amount to the right, so that the paths don’t touch anymore. In METAPOST you can use `eps` as a tiny positive number (in mathematics, an arbitrary small number is usually denoted by ϵ). The following lovely characters are the result (the METAPOST definitions are shown in listing 2):



3.3 Kerning

Our glyphs are ready, but a normal font has more features, such as kerning pairs and ligatures. In the

Listing 2: Definition of "a" and "t" with outlines.

```
encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginglyph("a");
path pa, pb, pc, r;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength+eps,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle
     scaled (2radius-2strength) shifted z0;
pc = unitsquare xscaled strength
     yscaled 2radius shifted z1;
find_outlines(pa,pc)(r);
Fill r1;
unFill pb;
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginglyph("t");
path pa, pb, r;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = height;
z3 = (hdist,height-3strength);
pa = z1
    -- (halfcircle rotated 180
        scaled 3strength shifted z0)
    -- (reverse halfcircle rotated 180 scaled
        strength shifted z0)
    -- z2 -- cycle;
pb = unitsquare xscaled 5strength
     yscaled strength shifted z3;
find_outlines(pa,pb)(r);
Fill r1;
fix_hsbw(2hdist+5strength,0,0);
endglyph;
```

former case, for a pair of characters the horizontal spacing between them is changed, while in the latter case a character pair is replaced by another glyph.

Defining a kerning pair in MetaType 1 is simple. *After* the definition of the glyphs, we can add a kerning table. In our case it looks like this:

```
LK("a") KP("t")(-3ku); KL;
```

In the list of ligatures and kernings for the letter "a" we define a kerning of -3 ku if it is followed by the letter "t" to remove the optical gap between them (the kerning unit 'ku' is defined elsewhere in the METAPOST source). The effect of kerning is shown in figure 1.

Ligatures don't make sense for our sample font, so I leave them out for this tutorial. In principle they work similarly; you merely define from which slot in the font the replacement for a specified character pair is to be taken.

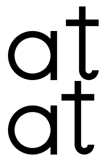


Figure 1: Our font without (top) and with (bottom) kerning.

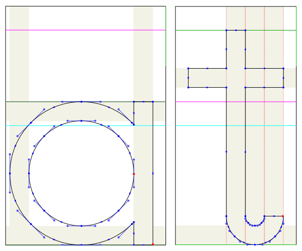


Figure 2: Hinting information (shaded areas).

3.4 Hinting

When you embed fonts as outline fonts, you leave the task of rasterizing the glyphs to your output device (printer or viewer). Unfortunately, this final result may look rather poor, especially on low resolution devices such as screens. Imagine the letter "H" and how it is rasterized into pixels. If we're unlucky, the left and right vertical stem will have a different width. On a printer with 1200 dpi it's nearly unnoticeable, but on the screen a difference of one pixel makes it look quite ugly.

To prevent this, high quality fonts use a mechanism called "hinting" to help the rasterizer (e. g. the PostScript RIP in a printer) to keep vertical or horizontal stems the same width.

MetaType 1 supports hinting by providing the macros `fix_hstem` and `fix_vstem` that try to find horizontal or vertical stems of a given width and add hinting information for them. For example, since we know that our letters "a" and "t" have stems of the width `strength`, we add hinting information by

```
fix_hstem(strength,pa,pb);
fix_vstem(strength,pa,pb);
```

You can see what hinting information was found as shaded areas in the proofs (figure 2).

4 Conclusions

I found that MetaType 1 is a suitable tool to create PostScript Type 1 fonts. Though there is a lack of beginning documentation, I was able to create a first font quite quickly by relying on an existing METAFONT source. Of course, knowledge of METAPOST or METAFONT is highly desirable. Understanding hinting is a bit more difficult, but finally possible.

References

[1] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *MetaType 1: A MetaPost-based engine for generating Type 1 fonts*, Proc. of EuroTEX 2001, published in MAPS 26, 2001, 111–119. http://www.ntg.nl/maps/pdf/26_15.pdf

[2] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk, *Programming PostScript Type 1 fonts using MetaType 1: Auditing, enhancing, creating*, *TUGboat*, volume 24 (2003), no. 3. http://tug.org/TUGboat/Articles/tb24-3/jackowski.pdf

[3] http://ctan.org/fonts/utilities/metatype1/

Listing 3: Makefile for font creation with MetaType 1.

```
METATYPE1 = /home/klaus/texmf/scripts/mt1

.PHONY: tfm pfb proof all

all: pfb tfm
proof: $(FONT).pdf
pfb: $(FONT).pfb
tfm: $(FONT).tfm

%.p: %.mp
    mpost "\generating:=0; \input $<"
    gawk -f $(METATYPE1)/mp2pf.awk \
          -vCD=$(METATYPE1)/pfcommon.dat \
          -vNAME=‘basename $< .mp‘

%.pn: %.p
    gawk -f $(METATYPE1)/packsubr.awk \
          -vVERBOSE=1 -vLEV=5 -vOUP=$@ $<

%.pfb: %.pn
    t1asm -b $< $@

%.tfm: %.mp
    mpost "\generating:=1; \input $<"

%.pdf: %.ps
    ps2pdf $< $@

%.ps: %.dvi
    dvips -o $@ $<

%.dvi: %.tex
    tex $<

%.tex: %.mp
    mpost $<
    cp $< _t_m_p.mp
    mft _t_m_p.mp -style=mt1form.mft
    echo '\input mt1form.sty' > $@
    test -f piclist.tex && cat piclist.tex >> $@
    test -f _t_m_p.tex && cat _t_m_p.tex >> $@
    echo '\endproof' >> $@
```

Listing 4: The complete font.

```
% A sample font
input fontbase;

% Global parameters for all characters
size := 1000; depth := 0; math_axis := 1/2size;
radius := 300; hight := 900; strength := 80;
ku := 18; hdist := 3ku; round_hdist := 1ku;

% Font settings
pf_info_familyname "MyFont";
pf_info_fontname "MyFont-Regular";
pf_info_weight "Normal";
pf_info_version "0.01";
pf_info_capheight hight;
pf_info_xheight 2radius;
pf_info_space 10ku;
pf_info_adl size, 0, 0;
pf_info_author "Made by KH"
pf_info_overshoots (1000,10), (0, -10);
pf_info_encoding "at";
pf_info_creationdate;

beginfont

encode ("a") (ASCII "a");
introduce "a" (store+utilize) (0) ();
beginglyph("a");
path pa, pb, pc, r;
z0 = (round_hdist+radius,radius);
z1 = (round_hdist+2radius-strength+eps,0);
pa = fullcircle scaled 2 radius shifted z0;
pb = reverse fullcircle scaled (2radius-2strength)
     shifted z0;
pc = unitsquare xscaled strength yscaled 2radius
     shifted z1;
find_outlines(pa,pc)(r);
Fill r1; unFill pb;
fix_hstem(strength,pa,pb,pc);
fix_vstem(strength,pa,pb,pc);
fix_hsbw(2radius+round_hdist+hdist,0,0);
endglyph;

encode ("t") (ASCII "t");
introduce "t" (store+utilize) (0) ();
beginglyph("t");
path pa, pb, r;
z0 = (hdist+3.5strength,1.5strength);
x1 = hdist + 2strength;
x2 = x1 + strength;
y1 = y2 = hight;
z3 = (hdist,hight-3strength);
pa = z1 -- (halfcircle rotated 180
            scaled 3strength shifted z0)
    -- (reverse halfcircle rotated 180
         scaled strength shifted z0)
    -- z2 -- cycle;
pb = unitsquare xscaled 5strength yscaled strength
     shifted z3;
find_outlines(pa,pb)(r);
Fill r1;
fix_hstem(strength,pa,pb);
fix_vstem(strength,pa,pb);
fix_hsbw(2hdist+5strength,0,0);
endglyph;

LK("a") KP("t")(-3ku); KL;
endfont.
```