

Fonts for paper-less T_EX: How to make them?

Alexander Berdnikov

Introduction

The subject of this Conference (and the well distinguished trend in modern Computer Assisted Typography) implies, that we need in T_EX the *Type 1* fonts in addition to the standard METAFONT fonts. Namely, we would like to compose the text in T_EX (since T_EX-based systems are still the most accurate systems) using the T_EX fonts. When (and if) we are satisfied with the output, we would like to convert it into something different—say, PDF document or PostScript document—based on *vector* fonts (*Type 1* fonts or *True Type* fonts).

We have at our disposal the technology how to use inside T_EX the *Type 1* fonts, and we also have the tools for converting T_EX output into electronic documents (*dvips*, *dvi2ps*, *pdftex*, *dvipdf*, *dvipdfm*)—where the electronic document will use the *Type 1* fonts, of course. What we *do not* have are the fonts which exist simultaneously as the METAFONT fonts and *Type 1* fonts being internally just the same. Currently there are just a few exceptions:

- Computer Modern fonts converted to *Type 1* format by several authors (the variant created by *AMS* and *BlueSky I* like best of all),
- Mathematical and Cyrillic AMS-fonts in *Type 1* format,
- Computer Modern fonts in *True Type* format created by Dr. Richard Kinch and included into his *TrueT_EX* system,
- Cyrillic fonts by Nana Glonti and Alexander Samarin converted to *Type 1* format by Basile Malyshev,
- Mathematical fonts for *New Mathematical Encodings* which are created by Taco Hoekwater parallelly in METAFONT and *Type 1* formats.

So, there are many native (and free) T_EX fonts still not converted into *Type 1* format while the *Type 1* format is necessary for electronic documents. We also have a lot of *Type 1* fonts (mostly commercial and hence not suitable for the public domain documents) without the analogs in T_EX. These two sets of fonts intersect only for Computer Modern family.

The METAFONT and *Type 1* fonts are different not only in their format, but also in their content. The native T_EX fonts contain many curious letters, symbols,

etc., generally absent as the *Type 1* fonts. From the other side the *Type 1* fonts are the high quality fonts for plain texts, and their absence in T_EX (I mean the absence of the *freeware fonts*, not the commercial ones) is really a sad moment. Although there are also the freeware fonts in *Type 1* format, they are mostly the funny font families not suitable for ordinary typing. The URW contribution created the standard Adobe *Type 1* fonts is really a nice, important and valuable exception.

Let us make the brief conclusion. The absence of the METAFONT fonts in *Type 1* format prevents to convert freely T_EX documents into electronic documents. The absence of freeware *Type 1* fonts in METAFONT format prevents to work with them in a pure T_EX frame and without PostScript devices. So, what we need is

- the freeware tool enabling to convert the METAFONT fonts into *Type 1* fonts (and may be backward as well),
- the freeware technology enabling to make the high quality METAFONT fonts and *Type 1* fonts in a parallel manner.

Tools we already have

Surprisingly, but the set of tools at our disposal is not infinitely small:

- the `mf2ps` program by Daniel Berry and Shimon Yanai,
- the `ps2mf` program by Marcel Dings and Erik-Jan Vens (and `ps4mf` package by Markus Neteler),
- the package `MF-PS` (METAFONT macros plus `.log`-processing scripts) by Bogusław Jackowski,
- the `METAPOST` program by John Hobby which produces the PostScript output from the METAFONT-like input,
- the `MetaFog` converter from METAFONT to *Type 1* format by Richard Kinch (based on `METAPOST` and special postprocessor),
- is there something else that I do not know?

Some of these tools do not work now (say, `mf2ps` based on currently obsolete Solaris library and METAFONT 2.0). Some are of no usage for our purpose (say, `ps2mf` which produces non-parametrized METAFONT output). The most promising one—namely, `MetaFog`—is not free. All of them require expert-level

knowledge to make them to work properly. The discussion of the advantages and disadvantages of these tools and the personal experience of the participants may be a good starting point of the workshop.

Problems with METAFONT to *Type 1* conversion

Here is the description of the conversion cycle based on METAPOST processing of the original METAFONT source and post-processing of the PostScript contours by some special program. (The reason is that it is the only conversion cycle I tried—great thanks to Taco Hoekwater for his hint.) Other experts can add more items to this list and/or describe their own technologies.

First, the `.mf` files are renamed into `.mp` files. Then we start the METAPOST using the command like

```
mp386.exe &mfplain \mode=lowres; mag=83.333333; input cmr12
```

to force it to work like METAFONT, not like the PostScript-drawing routine. (Here the magnification is equal to $(1000/12.00)$ —the design size 12 pt should correspond to the grid 1000×1000 points in *Type 1* coordinate system.)

As the result we get the files `cmr12.000`, `cmr12.001`, ..., `cmr12.127`, where each file contains the PostScript contours for one character. Then with the help of Ghostscript and the `ps2ai.ps` script we convert the PostScript files into Encapsulated PostScript files—to make them more simple and to get the opportunity to upload it into professional font editors like FontLab.

This is the analysis of the output:

- METAPOST output consists of overlapping contours (this is what we expected) while the overlaps should be removed in *Type 1* fonts.
- Some contours are not oriented properly (i. e., as the PostScript and *Type 1* filling rules require).
- Most contours just touch each other without sharp intersection.
- Some contours are self-intersecting (sic!), and even self-touching.
- There are the contours which should be filled by black or white color (supported by *Type 1* syntax), but there are also the contours which should be drawn by the round pen with the specified width (not supported by *Type 1* syntax).

- The information about elliptical pens is lost—METAPOST substitutes some intermediate round pen in such cases.
- The coordinates of the contour points are not rounded according to the resolution 1000×1000 (as we would like it to have)—hence, we can get the inaccuracies ± 1 point for the PostScript output after rounding or truncating.
- Opposite to the previous item, the width of the round pen *is* rounded—as a result the filled contours and the drawn contours may be shifted a little. Since most contours are nearly *touching* each other, it may shift the intersection points significantly or add the parasitic intersection points.
- METAPOST is not exactly the same program as METAFONT—so, just re-defining the commands `hround()` and `vround()` in `mfplain.mf` (to force the correct rounding of stems, flexes, baseline overlaps, etc.) crushes its work due to some ‘dirty tricks’ inside Knuth’s *Computer Modern* source files.
- METAPOST cannot operate with the pictures—say, some European Computer Modern fonts by Jörg Knappen cannot be processed for this reason because they *use* such transformations.
- Since METAPOST is different from METAFONT, the brute force attempt to convert the European Computer Modern fonts fails—`ecbase.mf` contains too many ‘dirty tricks’ which are not recognized by METAPOST properly (may be some expert can create the base file `ecbase.mp` for METAPOST, but I definitely cannot do it).
- There may be no node points at strictly horizontal and strictly vertical tangentials for the output contours, and just the same is true for the points where the curvature changes its sign—we should add all these points to the contours because this is the requirement of the *Type 1* language.
- There is no hinting for the output contours (what we could predict from the very beginning ☺)—so, we should put the hints manually or by some additional semi-automatrical tool.

The rounding problem (i. e., the fact that the functions `hround()` and `vround()` should be empty in `mfplain.mp`—otherwise METAPOST crushes on Knuth’s fonts) is more serious than it can be imagined from the first glance. Due to this effect we get the rounding error ± 1 point for the *Type 1* PostScript program where all coordinate points should be the integer values. So, our stems which are coded in the METAFONT source file to be exactly equal, becomes not equal by one point. Just the same is true for vertical and horizontal flexes

which become non-symmetrical by one point, etc. While the difference by one point for the scale of 1000 points may be considered as negligible, it violates the mathematical purity of the font and, which is more important, crushes the agreements on which the *Type 1* hinting system is based.

It is worth to note that except the errors and inaccuracies introduced by METAPOST, there are the errors and inaccuracies in the original METAFONT source as well. Say, some lines are not connected strictly tangentially (especially the serifs), some contours do not fit each other exactly, etc. While all these features are negligible for METAFONT (which just fills and draws the bitmap discrete image with the resolution where all such defects are not seen), they makes the proper conversion of the original shape into correct *Type 1* presentation more difficult.

(It is worth to note that the immediate loading of the PostScript output into some font editor like FontLab fails as well: it cannot delete all the overlaps of the contours because the METAFONT contours are tuned too fine and are adjusted to each other too delicacy. But the detailed discussion of FontLab (Fontographer, FontMonger, etc.) mistakes, errors and disadvantages is surely not the subject of this workshop.)

Why do we need the parallel technology

The problems with METAFONT to *Type 1* conversion are caused by the fact that these two systems are too different. Not all hacks used in METAFONT are suitable for *Type 1* fonts, and there are a lot of features in *Type 1* fonts which are not supported by METAFONT.

But contrary to that, METAFONT is currently the only tool enabling to describe the font shape mathematically strictly (OK, let us say ‘the only opened freeware tool’). Great bonus is that it describes the font in the parametrized manner—from a single METAFONT program we can derive much more and much better fonts than from the loudly advertised Adobe’s *Multiple Master Font Format*. It is also a very flexible and powerful tool as compared with the WYSIWYG font editors—like \TeX is more powerful and flexible than the WYSIWYG text processors.

But METAFONT definitely does not fit the task of designing the *Type 1* and *True Type* fonts. So may be the best solution is to create some intermediate system for font description—like SGML is the intermediate language for the documents. (Say, it is a serious problem to convert an arbitrary \TeX / \LaTeX document into RTF or HTML format. But using the SGML description there is no

problem to convert the document into L^AT_EX, or RTF, or HTML, or something else by using the proper filter.)

Here are some features which may be useful in such a system:

- The language should be structured like the procedure/function descriptive language, not like the macro language (no hacks!), but it should support variables, expressions, special types, etc., as it is done in METAFONT.
- Output data (METAFONT, *Type 1*, etc.), should be transparent enough—so that it can be read by the User and that the correspondence between the output code and the input code can be established easily.
- The IDE enabling to debug the font description and to see on the screen what you get, is desirable.
- The cursor moving over the graphical representation of the character with automatic re-calculation of its position in terms of the base variables controlling the shape of the character—is desirable as well.
- The language should be more contour-oriented than drawing-oriented (but say, addition, subtraction and transformation of the contours is supported as well as the drawing over the contour with the specified pen).
- The sub-language for hint description is necessary so that the hint information is parametrized like the font shape and is calculated automatically (although it may be difficult to make the hint description suitable for *Type 1* and *True Type* fonts simultaneously).
- Similarly, data necessary for *Type 1* and *True Type* headers should be described together with the font contours (as some meta-information, may be).
- While METAFONT cannot calculate the intersection points between the contour segments except for lines, the system should be able to define the intersection points automatically and to use them as if they are the user-specified points.
- The smooth connection of the contours should be forced more strongly than in METAFONT does where the connection angle $\approx 180^\circ$ is not the error.
- The cases where it is difficult to delete the intersections between the contours and to join them should be analyzed automatically with issuing warning/error messages to the user.

- The contours which are close to each other should be made identical (maybe, by explicit User' commands), and just the same is true for closely positioned reference points.
- Contrary to the previous item, the calculations should be done with extremely high accuracy because sometimes the accuracy implemented in METAFONT is not sufficient to calculate the intersection points for the curves which are nearly parallel.
- It should be possible to decrease automatically the number of curves necessary to describe the contour when two subsequent curves are close to be just the same curve.
- Similarly, too short segment should be deleted automatically with issuing the corresponding warning messages.
- Similarly, the language should force the User to use the *tension* and *curl* parameters to control the outline, not the additional points (a good compromise is to calculate automatically the Bezier curve passing through several reference points).
- Elliptical and polynomial pen shapes should be supported, but when the mathematically strict envelope of the curve drawn by such pen is created, small line segments connecting the subcurves should be deleted.
- Too fine details should be analyzed, and the User should be warned about them—say, in Knuth's Computer Modern fonts the serif flexes are so small (2 points at 1000×1000 grid) that they are the flat line in nearly all cases: you can see it as 1 (!) printer point for the letter 'A' with the height 2 cm when the resolution is 600 dots per inch.

Conclusion

It may be a nice idea to make the freeware (!) tool to convert the METAFONT files into *Type 1* files. It may be much better idea to introduce the SGML-like technology into the font designing. But *who* will do all this job?!

Address

Alexander Berdnikov
Institute of Analytical Instrumentation
St. Petersburg
Russia
E-Mail: berd@ianin.spb.su