Problems of the conversion of METAFONT fonts to PostScript Type 1

Basil K. Malyshev

Abstract

The paper describes problems pertaining to the automatic conversion of METAFONT fonts into the PostScript Type 1 font format. Several methods of conversion are discussed. A short description of the *Paradissa Fonts Collection* is presented. It contains Computer Modern fonts (used in (LA)TEX) in ATM compatible PostScript Type 1 format. The use of the collection and the problems related to it are discussed.

– – * – –

1 Introduction

Intensive quantification of human activities often implies rapid modifications of many methods of data production, processing, and use. Thus it seems necessary to adapt the collected data to efficient modern processing techniques.

One of these problems is the conversion of fonts defined in METAFONT into the PostScript font format and reports on the automatic conversion of fonts described in the METAFONT format to the PostScript Type 3 font format have been appearing since 1987 (see Carr, 1987; Henderson, 1989; Hobby, 1989; Berry & Yanai, 1990). However, these results are little used due to the poor rasterization of PostScript Type 3 fonts at low and middle resolutions on widely used output devices. Furthermore, the fonts are incompatible with Adobe Type Manager (ATM) which essentially narrows their application area.

The PostScript Type 1 font format as published by Adobe (Adobe, 1990) allows the possibility of performing METAFONT → PostScript Type 1 conversion that promises high quality fonts with a wide application area (including ATM).

The methods for automating such conversion are discussed in this paper.

2 METAFONT and PostScript Type 1

When comparing METAFONT and PostScript Type 1, we note the following features.

METAFONT is a high level language for font description. To draw a letter, METAFONT describes the curve traveled by the center of the pen, and the shape of this pen is allowed to vary as the pen moves. Coordinates of all reference points can be defined by parameters and linear equations governing these parameters. The main advantage of this approach is that the same definition readily yields a family of infinitely many related fonts of type, each font being internally consistent. Thus, infinitely many typeface styles can be obtained from a single definition by changing only a few parameters. METAFONT does not dictate its own font parametrization technique, but provides a designer with all the necessary tools.

METAFONT does not satisfy up-to-date standards of rasterization, because it yields acceptable results only at resolutions higher than 300dpi. For example, 240dpi × 216dpi is already poor, and at about 100dpi, typical of many graphic displays, the results are quite unsatisfactory.

PostScript Type 1 is a low level tool for font description. It reduces the description of a character outline to lines and Bezier cubic curves specifying some additional information (declarative hints) for the rasterizer. It gives excellent results on printers and other graphical devices with PostScript interpreters, and on systems with ATM. Furthermore, the font structure of PostScript Type 1 allows an application to perform rasterization only on characters that are actually used in the document, and to do it much faster than by METAFONT.

Adobe Type 1 rasterizers can generate variant fonts at any affine transformation of an original font, but this is not sufficient to generate several typeface styles with different optical sizes, weights, widths, and so on from a single description. To solve this problem, Adobe Corporation has developed a *Multiple Master Extension of the Adobe Type 1 font format* which can be used to generate a wide variety of typeface styles from a single font program (Adobe, 1992). This font format contains several outline typefaces called *master designs* which describe one or more design axes. Design axes represent a dynamic range of one typographic parameter, such as the weight or width.

METAFONT, however, provides font parametrization tools which are more flexible and natural for a designer. Also, a designer can readily add new symbols to a completed font definition in such a way that they are automatically consistent with the old ones.

Thus, combining of flexibility of a font description in METAFONT with the high quality of rasterization and simple use of PostScript Type 1 fonts seems to be desirable. Such a link can be represented by an automatic METAFONT → PostScript Type 1 converter generating fonts of rather high quality.

3 Steps and methods of MF → Type 1 conversion

The task of converting METAFONT fonts into the PostScript Type 1 font format may be divided naturally into two rather independent steps:

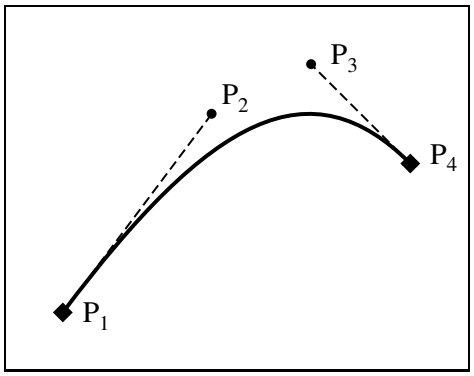

Figure 1: Bezier cubic curve defined by four points

- *extraction of character outlines* from the METAFONT font definitions;
- *generation of declarative hints*, which help the renderer to make best character rasterization.

The two steps can be performed by using different methods discussed below.

3.1 Extraction of character outlines

Each character in the PostScript Type 1 font is described by an outline specified by a set of lines and Bezier cubic curves. For a Bezier cubic curve four points are used: start point (P_1), endpoint (P_4), and two control points (P_2 and P_3), as shown in figure 1. The tangent vectors of endpoints are determined from the line segments P_1, P_2 and P_4, P_3. The algebraic equation for this curve is:

$$P(t) = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t)t^2 P_3 + t^3 P_4$$

for $0 \le t \le 1$.

Note that the labelling used in figure 1 for endpoints and control points will then be conserved for all elements forming character outlines.

Character outlines of high quality PostScript Type 1 fonts should not only describe all glyphs with sufficient accuracy, but also satisfy some important rules formulated in *Adobe PostScript Type 1 font format* (Adobe, 1990).

A *Points at extremes.* An endpoint should be placed at most horizontal or vertical extremes. This means that most curves should not include more than 90 degrees of arc.

B *Tangent continuity.* Whenever one path element makes a smooth transition to the next element, the endpoint joining the two elements and the Bezier control point associated with that endpoint (for a curve) or the other endpoint (for a line) should all be collinear.

C *Consistency.* All character features (stem width, height, spacing, shapes) that are intended to be the same should be exactly the same.

D *Conciseness.* Character outline definitions must be as concise as possible, without breaking the other rules.

There are two main approaches to the extraction of character outlines

- generation of character raster images by METAFONT with a subsequent recovering of the outlines by *tracing the pixels* and approximating the resulting outline by lines and Bezier cubic curves (see section 3.1.1);
- *direct extraction of outlines* from the METAFONT program with subsequent removal of overlapping elements and geometric optimization of resulted outline (see section 3.1.2).

3.1.1 Tracing the pixels of the bitmaps

At first, we need to generate bitmapped fonts by METAFONT. In PostScript Type 1 font format the endpoints and control points are defined on a 1000×1000 grid. To avoid rounding errors after outline scaling, magnification should be chosen so that font design size is rasterized into 1000 pixels. Therefore, the resolution should be chosen as

$$\langle \text{resolution} \rangle \frac{dots}{\text{inch}} = \frac{72.27\text{pt/inch} \times 1000\, dots}{\langle \text{design size} \rangle \text{pt}}.$$

Thus, the best resolution for the design size of 10pt is 7227dpi. This resolution does not require additional scaling of the resulting outline, and the METAFONT program performs correct coordinate rounding.

It is easy to make METAFONT itself to compute the required magnification by adding

```
pixels_per_inch:=4000 + 3227;
pixels_per_inch:=pixels_per_inch*4pt#;
tmp:=designsize/2.5;
pixels_per_inch:=pixels_per_inch/tmp;
```

to `mode_def` macro definition in the `local.mf` file.

Many programs will reconstruct a raster image outlines by tracing the pixels. However, font generation requires that the outlines satisfy the rules discussed in section 3.1. Below we consider several sets of results, and focus on whether these rules are obeyed or violated.

The software developed by Neil Raine generates the outlines from the bitmaps by tracing the pixels. Graham Toal generated Computer Modern fonts at 3000dpi resolution and produced PostScript Type 3 fonts by using Raine's program. These fonts are available in CTAN in `/tex-archive/fonts/cm/ps-type3` directory. The examples of these outlines are

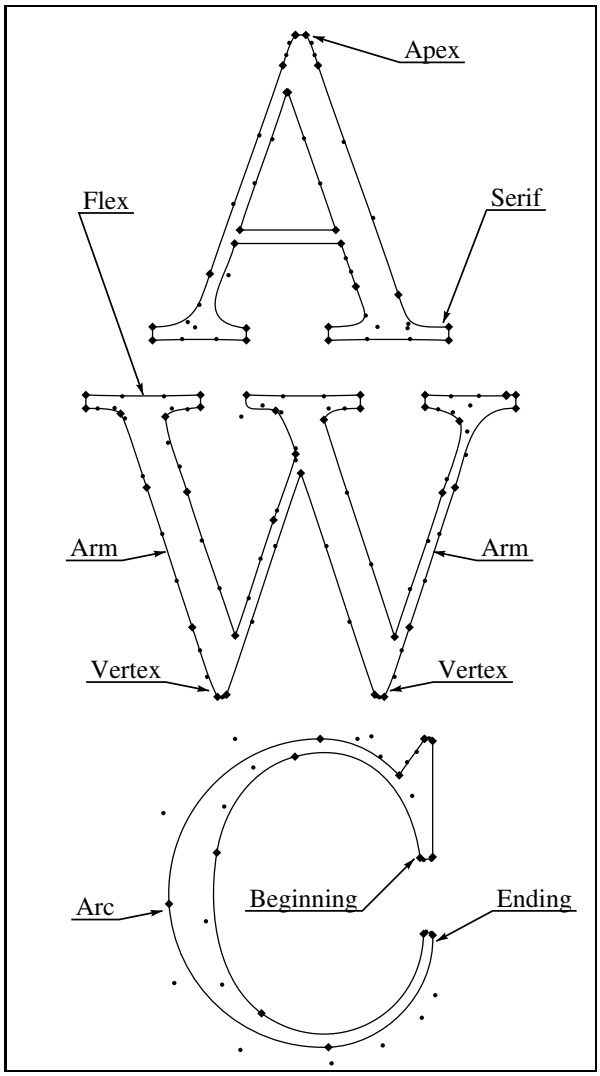

Figure 2: Neil Raine and Graham Toal cmr10 outlines

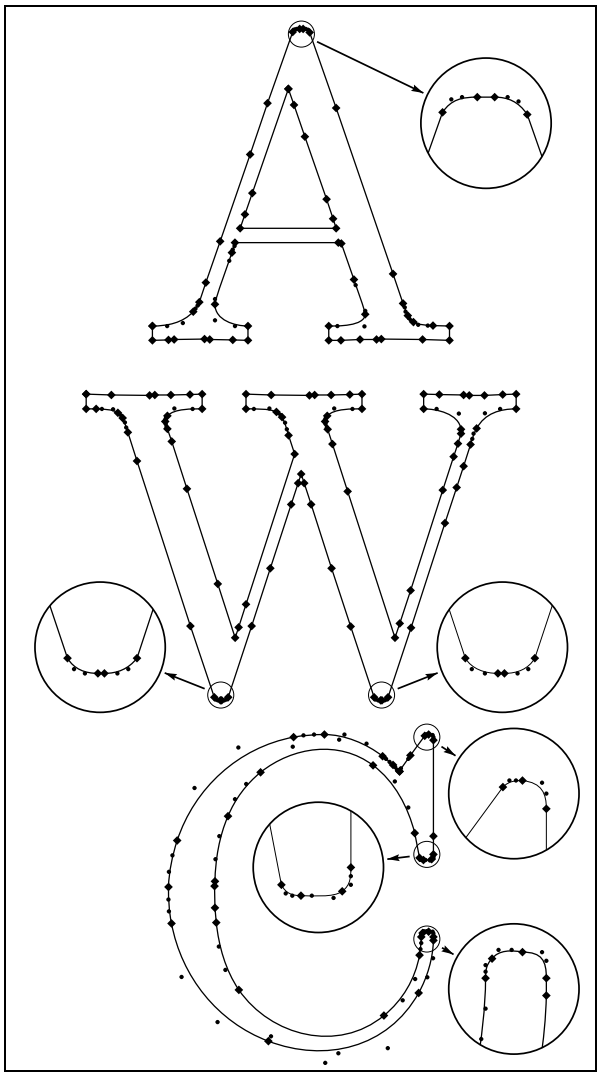

Figure 3: cmr10 outlines produced by `LIMN` from the GNU FONT UTILITIES

shown in figure 2. It is easy to see that these outlines suffer from some defects:

- The violation of rule **3.1.A** in the letter 'C'. This defect is not occasional, but is common to most curved stems.
- Asymmetrical *vertices* in the letter 'W' violate rule **3.1.C**.
- Bad transitions from *arms* to *serifs* in the letters 'A' and 'W'.
- A middle *serif* in the letter 'W' is unsatisfactory.

Karl Berry and Kathryn Hargreaves developed the GNU FONT UTILITIES, and announced them in TEXhax (Volume 92, Issue 8 and 17). These utilities contain a program `LIMN` which takes the bitmap fonts and generates outlines by tracing the pixels. The results of their operation are shown in figure 3.

The outlines presented in this figure also have some visible defects:

- The violation of rule **3.1.A** in the letter 'C'. Similar to the above example, this defect is regularly repeated.
- Each *flex* is split into many short line segments, violating rule **3.1.D**.
- Multiple consecutive collinear line segments violate rule **3.1.D**.

These outlines, when compared with Toal's, offer some evident advantages, namely the apex of the letter 'A' is symmetrical and the vertices of the letter 'W' are symmetrical and match each other.

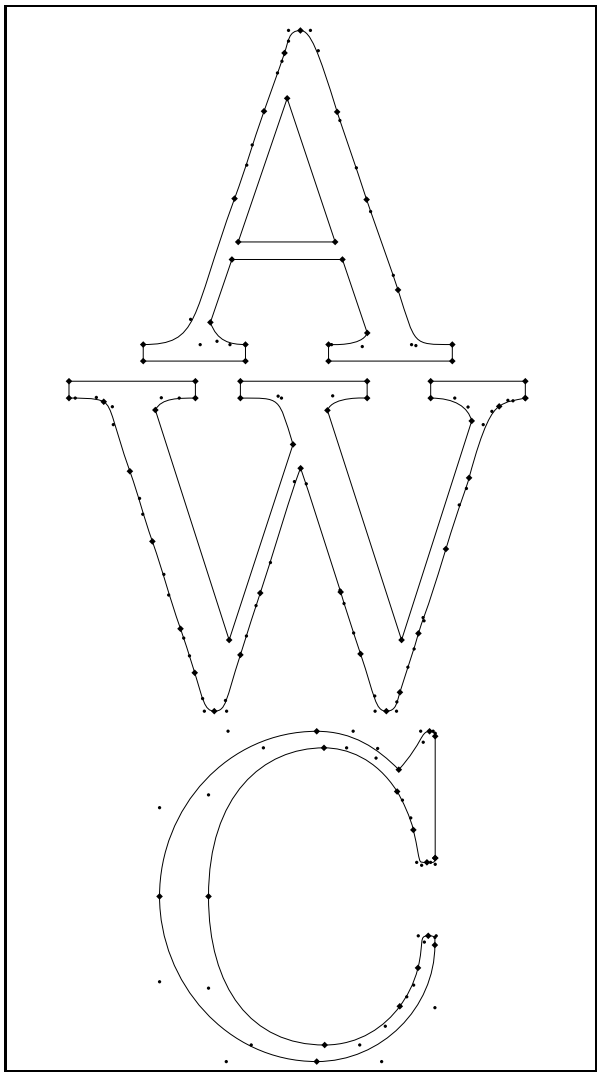

Figure 4: cmr10 outlines produced by `FontoGrapher`

To make more comparisons, bitmap fonts (at 3613dpi resolution) were traced by `FontoGrapher` (version 3.5.1). The resolution 3613dpi = 7227dpi/2 has been chosen because the tracing procedure built in `FontoGrapher` can break at too high a resolution. The result of this experiment is shown in figure 4. In spite of the fact that rule **3.1.A** is satisfied here (having been violated in the above examples), the outlines are not free from some defects:

- The *apex* in the letter 'A' is asymmetrical (violation of rule **3.1.C**).
- The *vertices* in the letter 'W' are asymmetrical and different (violation of rule **3.1.C**).
- *Arm – serif* transitions in the letters 'A' and 'W' are not satisfactory.

Evidently, it is difficult to obey rule **3.1.C** when directly tracing raster images. On the other hand, rule **3.1.A** seems to be easy to satisfy, and the violation of this important rule in the first two examples could probably be attributed to lack of authors' attention to it.

3.1.2 Extraction of outlines from METAFONT

Each program considered in section 3.1.1 exhibits its own defects of outline generation. Besides, there are defects common to all programs, such as poor discovery of *flex* features or bad *serif–arm* transitions.

The information critical for good appearance of characters is evidently lost when tracing an outline on a bitmap. Therefore, the extraction of outlines from METAFONT definitions without raster representation of fonts seems to be more fruitful.

The first attempt at extracting the character envelopes from METAFONT was undertaken by Leslie Carr (1987). Carr's programs take as input the METAFONT log file which contains a description of all the paths that METAFONT traces out in drawing a character. But using this method one should take into account the METAFONT pen shape.

Later Daniel M. Berry and Shimon Yanai (1990) have developed a more successful program, `mf2ps`, that finds the internally generated METAFONT envelopes, used as the boundaries of the inked region, and uses these envelopes as the PostScript outlines. In both these attempts, PostScript Type 3 fonts have been generated. The outlines generated by the `mf2ps` program are shown in figure 5.

To present such envelopes for PostScript Type 3 font the `mf2ps` program reorders cycled subpaths and chooses black and white filling for each of them. This method, although suitable for Type 3 fonts, fails for Type 1.

To produce outlines suitable for Type 1 fonts, all envelope overlapping should be removed. The result of this operation is shown in figure 6. Note that the resulting outlines contain too many consecutive lines and curves split into many pieces. To obtain outlines free from such defects, I have made some geometrical optimization. The result is shown in figure 7.

Now the outlines look significantly better than those in figures 2, 3 and 4. All the rules from section 3.1 have been satisfied. The fragmentation of the inner side of arc in the letter 'C' occurs because of the high (probably too high) accuracy in the approximation of the original shape.

There are some slight peculiarities related to a different encoding of similar shapes. In the letter

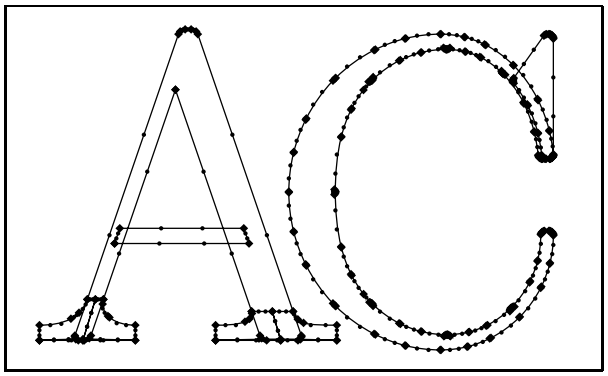

Figure 5: METAFONT internally generated envelopes

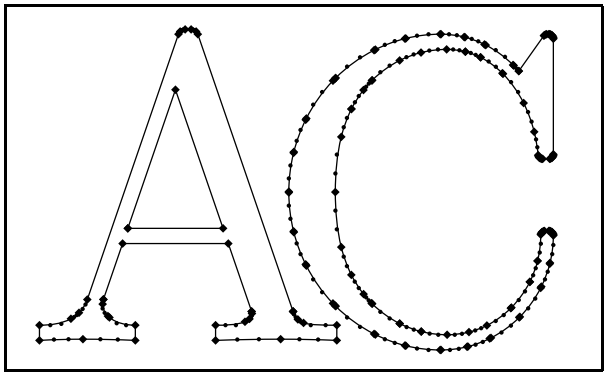

Figure 6: METAFONT envelopes after removing overlapping

'Φ', for example (figure 8), the outline representation is somewhat asymmetrical. The outline itself has a slight asymmetry, but in the process of rasterization its asymmetry of encoding may be exaggerated.

Nevertheless, this method is a step forward in improving the character outlines obtained from the METAFONT font definitions.

3.2 Generation of declarative hints

One of the main problems arising in font rasterization on a discrete grid is the conservation of the important geometrical properties of outlines. Identical parts of the letters differently located on the grid may take different shapes in a discrete representation.

In the PostScript Type 1 font format this problem is solved by using *declarative hints* which indicate where a horizontal or vertical stem occurs in certain coordinates. Those parts of the outlines which appear inside of the so-called stem hints will be rendered by special techniques.

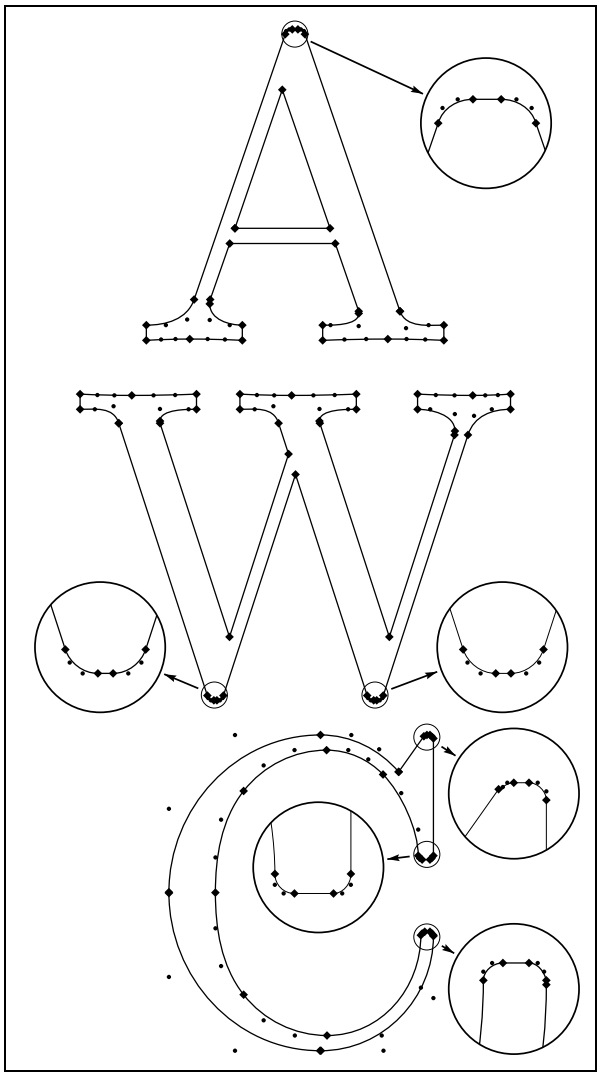

Figure 7: METAFONT envelopes after geometrical optimization

Declarative hints for the outlines obtained at the previous stage can be generated by font editors like `FontMonger` or `FontoGrapher` as follows:

- styling the outlines in the form of ATM compatible font format;
- loading the outlines to a font editor;
- saving the completed font on a disk.

As a result, the saved font contains the declarative hints.

The hints generated by `FontMonger` (version 1.0.4) are shown in figure 9. Comparing these outlines with the original ones from figure 3, it is easy to see that `FontMonger` has changed the outlines so that rule **3.1.A** is satisfied. However, stem hints for the serifs in the letters 'A' and 'W' are missing.

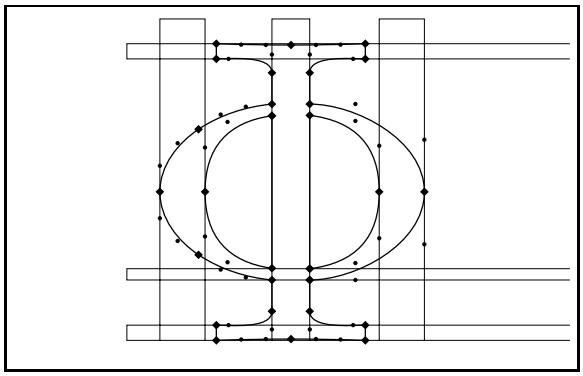

Figure 8: Asymmetrical coding of the like symmetrical form in the letter Φ

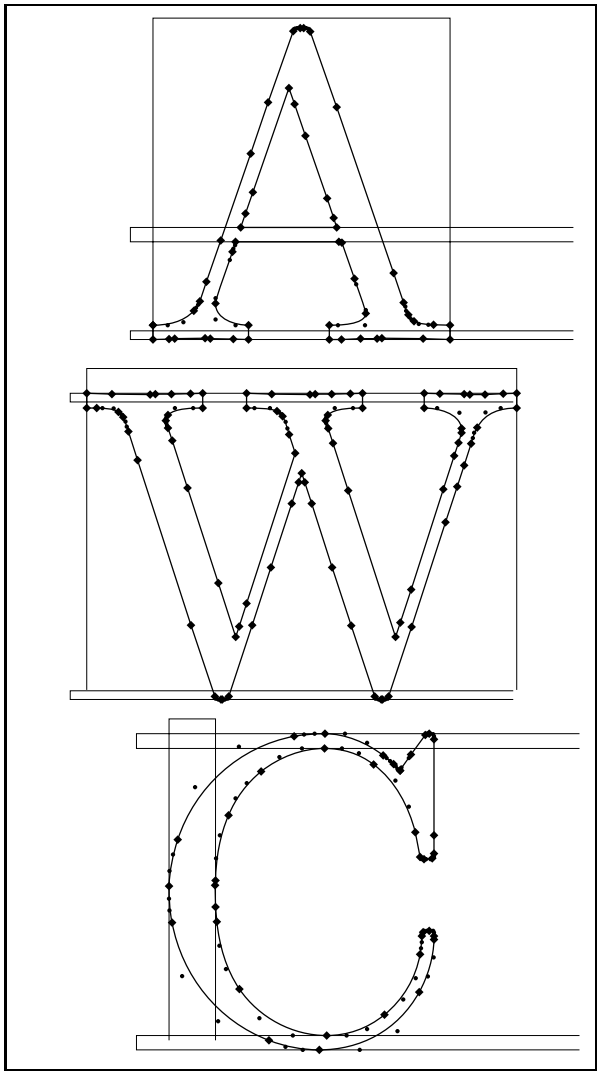

Figure 9: `LIMN` generated outlines hinted by `FontMonger`

The declarative hints generated by `FontoGrapher` for the same outlines are shown in figure 10. In this figure the outlines are unchanged, while the hints for the curved stems in the letter 'C' are missing. That is, `FontoGrapher` does not even try to correct the violation of rule **3.1.A**, and this violation has a pernicious effect on its operation. However, unlike the latter case, all hints for the serifs in the letters 'A' and 'W' are found correctly now.

If the outlines are not too accurate (such as those generated by the `LIMN` program), then the operation of both `FontMonger` and `FontoGrapher` is not good enough. However, if a font processed by `FontMonger` is subsequently processed by `FontoGrapher`, the obtained results can be significantly better (figure 11).

In the case of more accurate outlines directly extracted from METAFONT (figure 7), this approach is quite suitable, but `FontMonger` can hardly find the serifs (figure 12), while the results obtained by `FontoGrapher` are quite acceptable (figure 13).

Thus, one can see that `FontoGrapher` performs hinting somewhat better than `FontMonger`. Still `FontMonger` also has an advantage useful for mass conversion of fonts because it has a batch conversion utility.

The *Paradissa Font Collection* has been created using a homegrown algorithm for generating character hints. This algorithm has been developed especially for processing the outlines generated by the `LIMN` program. The result of its operation is shown in figure 14 where it can be seen that almost all the hints have been found, but the outlines have not been corrected as in the case of running `FontMonger` (figure 9).

In the case of outlines directly extracted from METAFONT (figure 7), our homegrown algorithm gives results (figure 15) competitive with those obtained by `FontoGrapher` (figure 13).

4 Paradissa Font Collection

The *Paradissa Font Collection* has been developed using the outlines generated by `LIMN` and a specially developed outline filter and hinting algorithm. This font collection is available from CTAN in the

`/tex-archive/fonts/cm/ps-type1/paradissa`

directory. The examples of the hinted outlines contained in this collection have already been presented in figure 14.

The Paradissa Font Collection contains:

- Computer Modern, designed by D. Knuth;
- Euler by H. Zapf;
- CM Cyrillic by N. Glonty & A. Samarin;

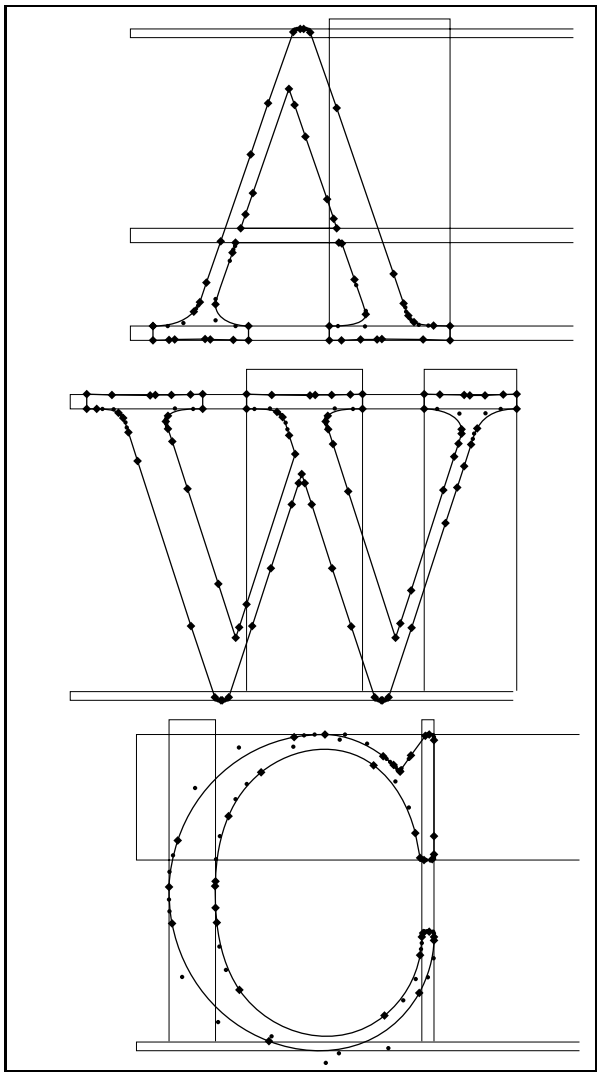

Figure 10: `LIMN` generated outlines hinted by `FontoGrapher`

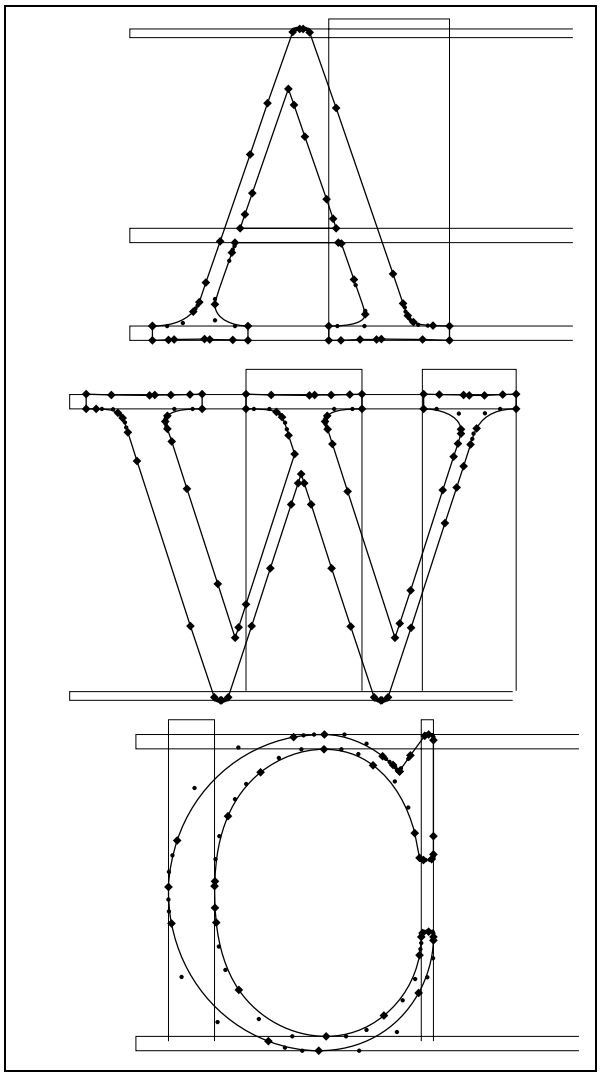

Figure 11: `LIMN` generated outlines hinted first by `FontMonger` and then by `FontoGrapher`

- Special LaTeX fonts.

Altogether it contains 165 fonts in ATM compatible PostScript Type 1 format with `AFM` and `PFM` files. This set of fonts may be used for printing most (LA)TeX documents. It is used by the Russian-English LaTeX version developed and supported by the ProTeX group at IHEP.

This collection can be used for

- printing documents on a PostScript printer by using, for example, Rokicki's `DVIPS` driver. It should be noted that the typesetting of even a simple LaTeX document may require a lot of printer memory to download fonts. This problem is solved, for instance, by the commercial program `DVIPSONE` which uses a special technique for *partial font downloading* to conserve the printer's memory.
- printing documents on a large collection of matrix printers by using `DVIPS` and `ghostscript`.
- drawing slides on vector plotters by using the PostScript `plot.ps` program which is supplied with the collection. For drawing documents on HPGL plotters, the `ps2hpgl` utility can be used. It is available on `ftp.mathworks.com` host in the `/pub/contrib/tools` directory.
- displaying documents under MS Windows with ATM by using the commercial `DVIWindo` program. We also expect that the capability of using Type 1 fonts will be added to Hippocrates Sendoukas' `DVIWIN` program.

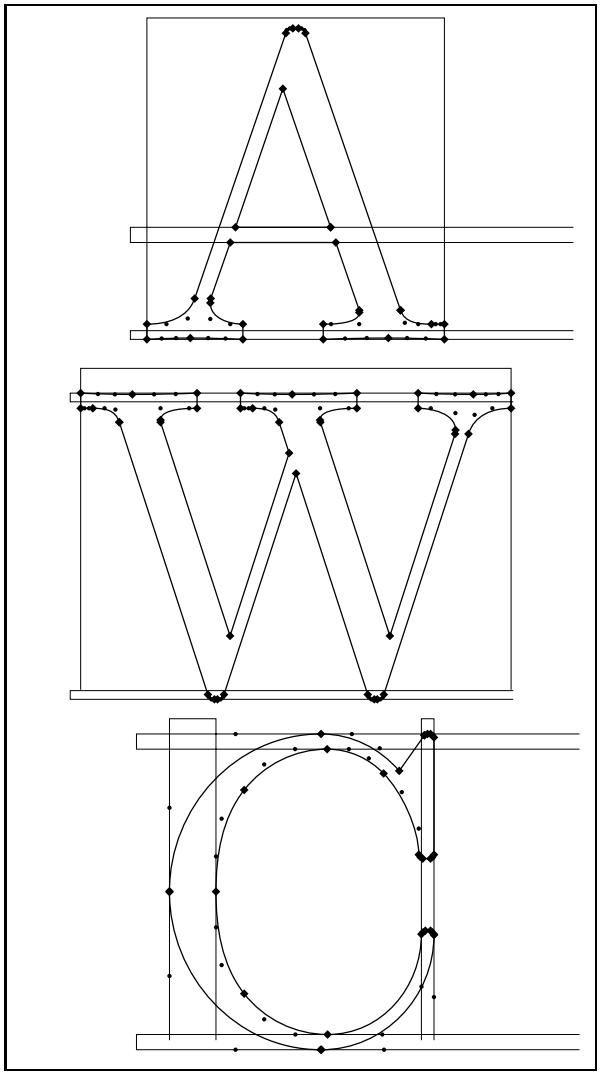

Figure 12: BKM outlines hinted by `FontMonger`

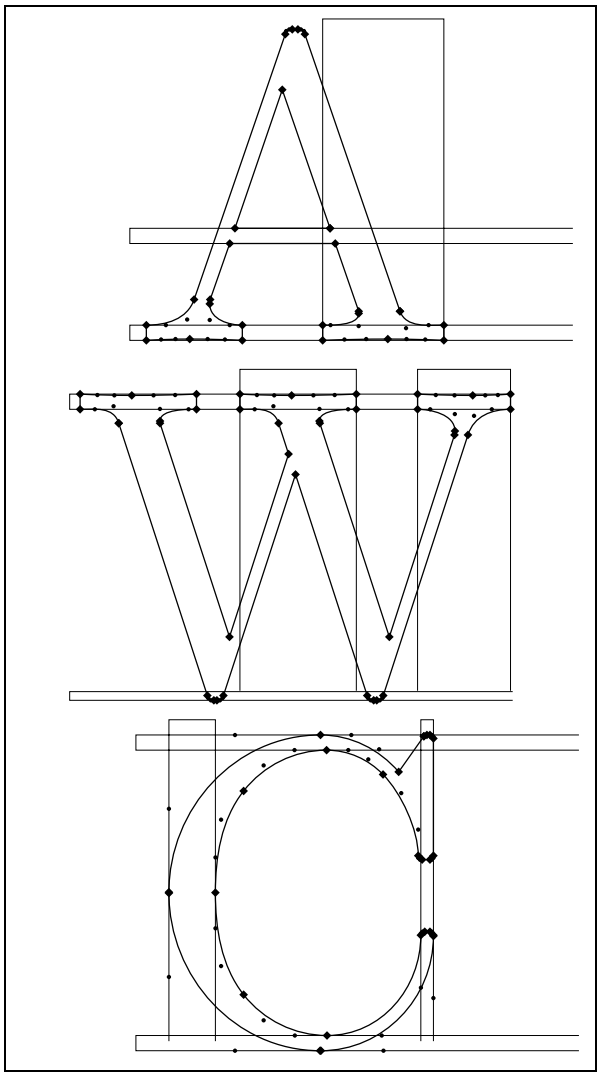

Figure 13: BKM outlines hinted by `FontoGrapher`

References

Adobe Systems Inc. *PostScript Language Reference Manual*. Addison-Wesley, 1985.

Adobe Systems Inc. *Adobe Type 1 Font Format*. Addison-Wesley, August 1990, Version 1.1.

Adobe Systems Inc. *Adobe Type 1 Font Format: Multiple Master Extensions*. Adobe Developer Support, 14 February 1992.

Berry, Daniel, and Shimon Yanai. "Environment for Translating METAFONT to PostScript." *TUGboat* **11** (4), p. 525–541, 1990.

Carr, Leslie. "Of Metafont and PostScript." *TEXniques* **5**, p. 141–152, August, 1987.

Henderson, Doug. "Outline fonts with METAFONT." *TUGboat* **10** (1), p. 36–38, 1989.

Hobby, John D. "A METAFONT–like System with PostScript Output." *TUGboat* **10** (4), p. 505–512, 1989.

Knuth, Donald E. *The METAFONTbook*. Reading, Mass.: Addison-Wesley, 1986.

Knuth, Donald E. *METAFONT: The Program*. Reading, Mass.: Addison-Wesley, 1987.

◇ Basil K. Malyshev
Institute for High Energy Physics,
IHEP, OMVT, Moscow Region,
RU-142284 Protvino, Russia
Email: `malyshev@mx.ihep.su`

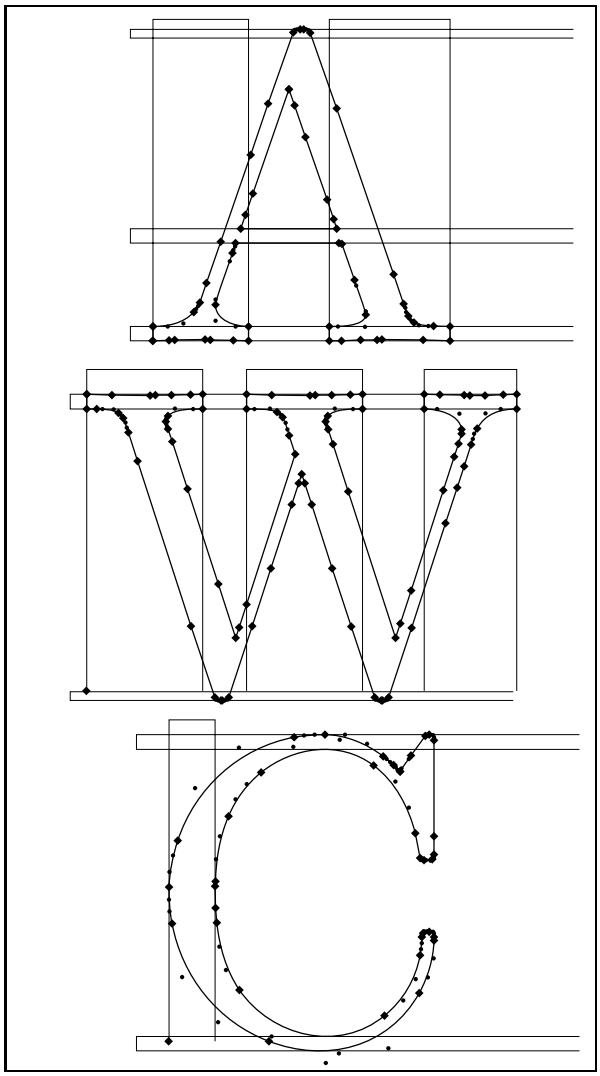

Figure 14: LIMN generated outlines hinted by homegrown hinter

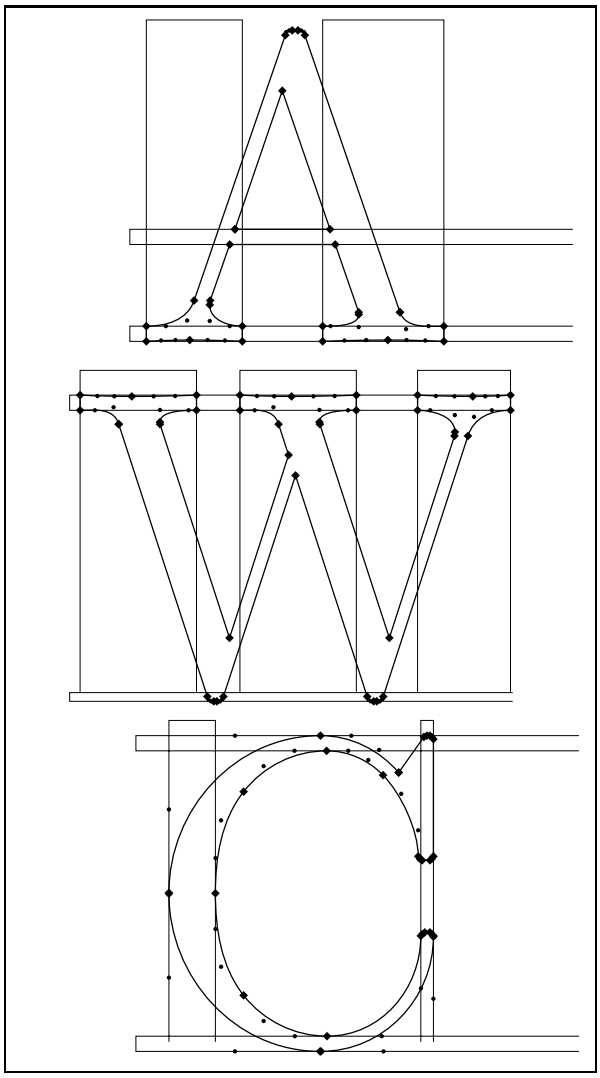

Figure 15: BKM outlines hinted by homegrown hinter