Creating Type 1 Fonts from METAFONT Sources: Comparison of Tools, Techniques and Results

Karel Píška
Institute of Physics, Academy of Sciences
182 21 Prague
Czech Republic
piska@fzu.cz
http://www-hep.fzu.cz/~piska/

Abstract

This paper summarizes experiences in converting METAFONT fonts to PostScript fonts with TEXtrace and mftrace, based on programs of autotracing bitmaps (AutoTrace and potrace), and with systems using analytic conversion (MetaFog and MetaType1, using METAPOST output or METAPOST itself). A development process is demonstrated with public Indic fonts (Devanagari, Malayalam). Examples from the Computer Modern fonts have been also included to illustrate common problems of conversion. Features, advantages and disadvantages of various techniques are discussed. Postprocessing — corrections, optimization and (auto)hinting — or even preprocessing may be necessary, before even a primary contour approximation is achieved. To do fully automatic conversion of a perfect METAFONT glyph definition into perfect Type 1 outline curves is very difficult at best, perhaps impossible.
KEYWORDS: font conversion, bitmap fonts, METAFONT, METAPOST, outline fonts, PostScript, Type 1 fonts, approximation, Bézier curves.

1 Introduction

In recent years, several free programs for creating PostScript outline fonts from METAFONT sources have been developed. The aim of this paper is to give a short comparison of these programs, with references to original sources and documentation, and to provide a brief description of their use. We will discuss advantages and drawbacks, and demonstrate numerous examples to compare important features and to illustrate significant problems. We omit technical details described in the original documentation and concentrate our attention on the quality of the output, including hinting issues.

The programs TEXtrace and mftrace read original METAFONT sources, generate high-resolution `pk` bitmaps, call autotracing programs (AutoTrace or potrace) and finally generate the files in the Type 1 format (`pfb` or `pfa`).

MetaType1 creates Type 1 output from METAPOST sources. Therefore it requires rewriting font definitions from METAFONT into METAPOST.

Similarly, MetaFog converts the PostScript files generated by METAPOST to other PostScript files containing only outlines, that can be subsequently assembled into Type 1 fonts. MetaFog is not a new product, but its excellent results remain, in our comparisons, unsurpassed.

Additionally, we may need adequate encoding files. If none are available, a TEX encoding (e.g., the standard TEX T1 encoding) is usually used as the default.

2 Autotracing Bitmaps

2.1 TEXtrace with AutoTrace

Péter Szabó developed TEXtrace [18]. It is a collection of Unix scripts. It reads the original METAFONT sources, rendering the font bitmaps into PostScript (via dvips). For converting the resulting bitmaps to outlines, it calls (in the version of 2001) the AutoTrace program [21] created by Martin Weber, and, finally, composites the final files in the Type 1 format. TEXtrace works fully automatically and can be invoked by a command like this:

`bash traceall.sh` *mfname psname psnumber*

where *mfname*`.mf` is the name of the METAFONT font, *psname*`.pfb` is the name of the Type 1 font file, and *psnumber* denotes a Type 1 `UniqueID` [1].

The *Adobe Type 1 Font Format* documentation [1, pp. 29–33] recommends observing certain *Type 1*

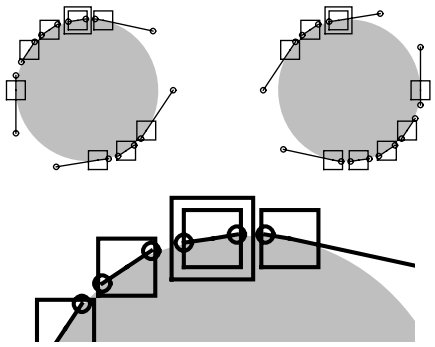

Figure 1: TEXtrace: "¨" in `cmr10`.

conventions: 1) points at extremes; 2) tangent continuity; 3) conciseness; and 4) consistency.

The outline results from TEXtrace (that is, from AutoTrace) are relatively faithful to the original bitmaps. Some artifacts exist, but they are invisible in usual font sizes and magnifications and for practical purposes may be negligible. Nonetheless, they spoil our attempts to automatically produce perfect, hinted, outline fonts.

The underlying reason is that the information about the control points in the original METAFONT is lost, and the Type 1 conventions are not satisfied, as exemplified in figure 1. The endpoints (double squares) are not placed at extremes (rule 1), most of the horizontal and vertical points of extrema are missing. On the other hand, the outline definition is not concise (rule 3) — due to the large numbers of control points in the glyph definitions, the font files generated by TEXtrace are huge. Furthermore, the two identical periods in the dieresis glyph "¨" are approximated by different point sets (rule 4).

The following examples show the results of conversion of Indic fonts submitted to TUG India 2002 [16], devanagari (`dvng10`) and Malayalam (`mm10`). Typical irregularities produced by conversion with TEXtrace are *bumps* and *holes*. Figure 2 demonstrates bumps caused by the envelope being stroked along a path with a rapid change of curvature, and by cases of transition from a straight line to a significantly small arc. The second clipped part of the letter "pha" shows a hole.

I tried to remove those bumps and holes, and (partially) other irregularities at the Type 1 level with a set of special programs manually marking places to be changed in a "raw" text, translated by `t1disasm` and by `t1asm` back after modifications (both programs are from the t1utils package [13]), which achieves a better outline approximation, as shown in figure 3. The postprocessing consisted of:

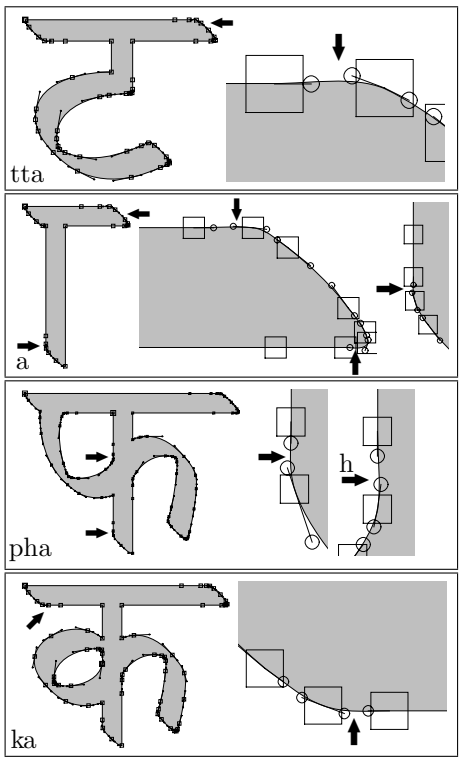

Figure 2: Results of TEXtrace (AutoTrace): bumps and a hole (h).

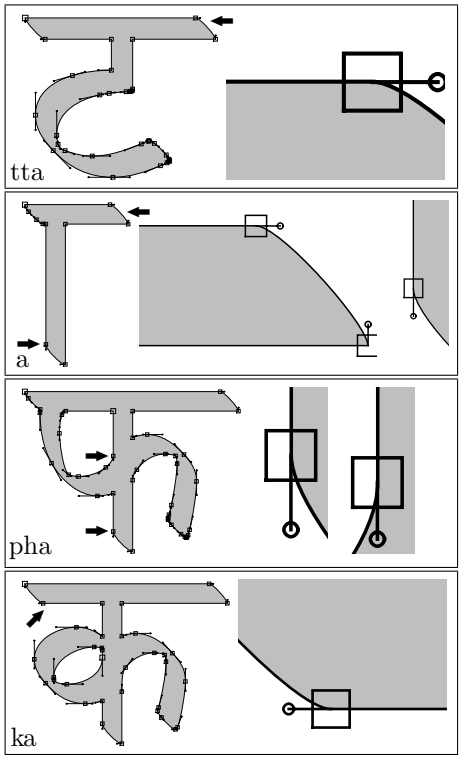

Figure 3: Improved results achieved with postprocessing.

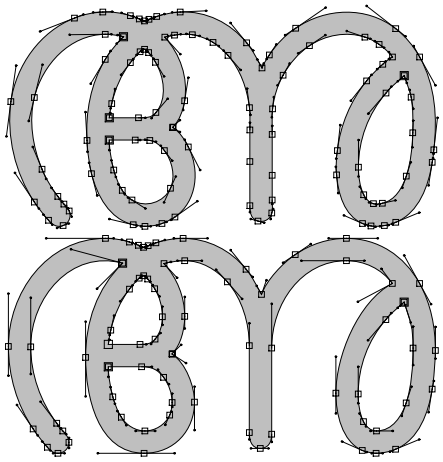

Figure 4: TEXtrace (AutoTrace) first without and then with postprocessing for the Malayalam "a", showing undetected corners.

inserting missing extrema points, changing the first nodes of contour paths (if desirable), and the optimization of merging pairs (or sequences) of Bézier segments together, and joining nodes in horizontal or vertical straight parts to eliminate redundant nodes.

However, when this process was applied to the Malayalam fonts, we meet another problem: undetected corners in Figure 4. Instead of attempting to correct them, I stopped my postprocessing attempts, and switched to experiments with analytic methods of conversion.

2.1.1 Examples of CM-super

Type 1 fonts [20] generated by Vladimir Volovich (first announced in 2001) inherit typical bugs produced by tracing bitmaps by AutoTrace (as invoked by TEXtrace) such as bumps and holes, improper selection of starting points of contour paths, and problems in distinguishing sharp corners and small arcs. We illustrate them in several following figures, in order to demonstrate that fixing such irregularities automatically is difficult.

In the period "." from the `sfrm1000` font (its source is the original `cmr10`), an optimization cannot exclude the redundant node (fig. 5) (it is still the starting point of the path).

The minus "−" derived from `cmr10` contains a bump, and minus from `cmtt10` two bumps (fig. 6). Moreover, these bumps have been hinted and have their own hints (probably as results of autohinting).

In the letter "M" from `cmtt10`, we observe missing dishes, a hole and a bad approximation of an arc

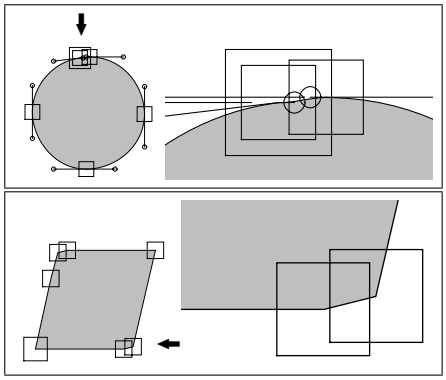

Figure 5: CM-super: period in `sfrm1000` and `sfsi1000`, with redundant node.

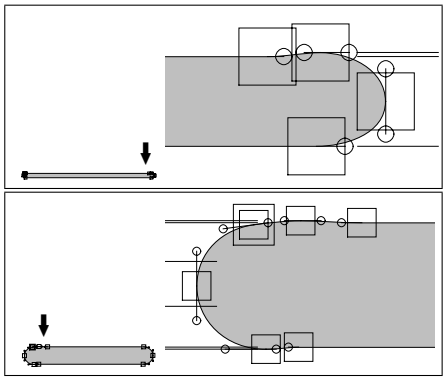

Figure 6: CM-super: minus in `sfrm1000` and `sftt1000`, with bumps.

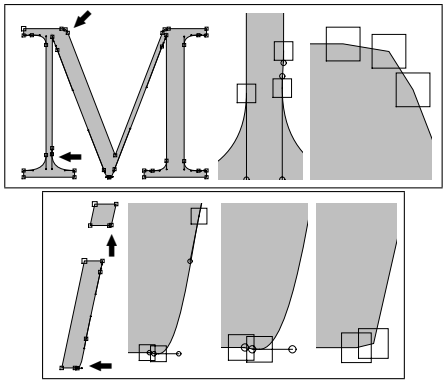

Figure 7: CM-super: "M" in `sfrm1000` and "i" in `sfsi1000`.

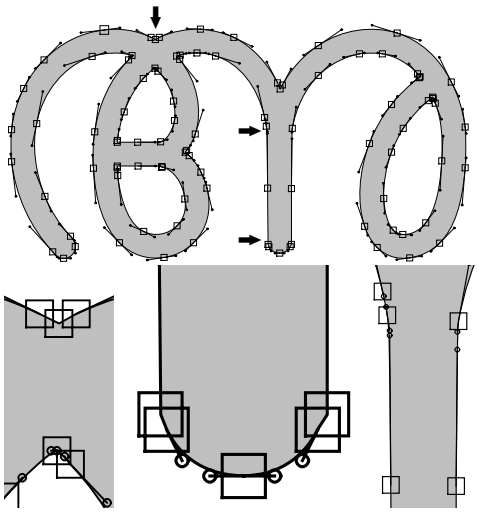

Figure 8: TEXtrace (using potrace), with different corners.

(fig. 7). On the contrary, in "i" the corners are not detected properly, we also have a hinted bump.

2.2 TEXtrace with potrace

The 2003 version of TEXtrace supports alternative bitmap tracing with `potrace` [17], developed by Peter Selinger. In this version, the real corners are detected or at least detected better than with AutoTrace (see fig. 8). Thus, bumps and holes have been suppressed, but smooth connections have often been changed to sharp corners (not present originally). While the bumps demonstrated violation of consistency and may produce invalid hinting zone coordinates (fig. 6), the unwanted sharp corners mean loss of tangent continuity (the middle clip in fig. 8). Unfortunately, the approximation does not preserve horizontal and vertical directions (the right clip), the stem edges are oblique — the difference between the two arrows on the left edge is 2 units in the glyph coordinate space.

2.3 mftrace

Han-Wen Nienhuys created mftrace [15, 3], a Python script which calls AutoTrace or potrace (as with TEXtrace) to convert glyph bitmap images to outlines. The results of tracing are thus expected to be very similar to those of TEXtrace. In fact, for the analyzed Indic fonts, they are identical, as we can see in the first image in figure 9 (compare with TEXtrace results in fig. 4). With the `--simplify` option, mftrace calls FontForge [22] (previously named PfaEdit) to execute postprocessing simplification; this helps to exclude redundant nodes from outline contours, as in the second image in figure 9.

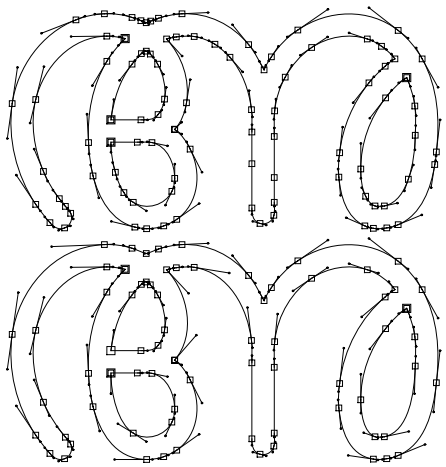

Figure 9: `mftrace` without and with `--simplify`.

3 Analytic Conversions

3.1 MetaType1

MetaType1 [8, 9] is a programmable system for auditing, enhancing and generating Type 1 fonts from METAPOST sources. MetaType1 was designed by Bogusław Jackowski, Janusz M. Nowacki and Piotr Strzelczyk. The MetaType1 package is available from `ftp://bop.eps.gda.pl/pub/metatype1` [10].

This "auditing and enhancing" is a process of converting the Type 1 font into MetaType1 (text) files, generating proof sheets, analysis, making corrections and regenerating modified Type 1 fonts. It is an important tool for checking, verifying and improving existing Type 1 fonts.

MetaType1 works with the METAPOST language. Therefore the METAFONT font sources must be converted/rewritten into METAPOST. Macro package extensions of METAPOST and other miscellaneous programs provide generation of proper structure of the Type 1 format, evaluate hints (not only the basic outline curves), and create `pfb` and also `afm` and `pfm` files.

During the rewriting process, users define several parameters of the Type 1 font, including the PostScript font encoding — PostScript glyph names and their codes — because METAFONT sources do not contain this data in a form directly usable for Type 1 encoding vectors. METAFONT output commands have to be changed to their METAPOST alternatives. Similarly, it is necessary to substitute METAFONT commands not available in METAPOST, to define METAPOST variants of pen definitions and pen stroking, etc.

Alternative METAPOST commands are defined in the MetaType1 files `fontbase.mp`, `plain_ex.mp`,

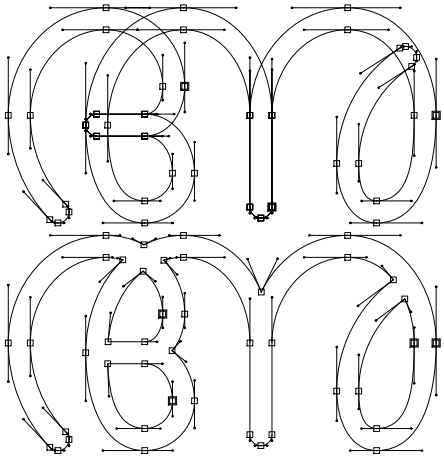

Figure 10: MetaType1 — primary outlines and overlap removal.

et al. Other (new) commands may be defined by the user. Correspondence between METAFONT and METAPOST is approximately as shown in the following table (of course, the details may vary from font to font):

METAFONT	METAPOST
`fill` *path;*	`Fill` *path;*
`draw` *path;*	`pen_stroke()(`*path*`)(`*glyph*`);`
	`Fill` *glyph;*
`penlabels(1,2);`	`justlabels(1,2);`
`beginchar( ...`	`beginglyph( ...`
`endchar;`	`endglyph;`

Many METAFONT commands have no counterpart in METAPOST [6]. For example, operations with bitmap pictures: in METAPOST, font data is represented as PostScript curves, not bitmaps. As a result, writing METAPOST code that would produce equivalent results as original METAFONT code using these or other such features would be very difficult.

After the basic conversion, the next step is removing overlaps (if any are present) using the MetaType1 command `find_outlines`. Figure 10 shows the results before and after overlap removal for the Malayalam vowel `a` (font `mm10` using pen stroking with a circular pen). This operation is not necessary in METAFONT, since it generates bitmaps. In the METAPOST environment of PostScript outlines, however, we need to reduce overlapping curves to single or pairs of paths.

MetaType1 also allows insertion of commands for automatic computation of horizontal and vertical hints (`FixHStems, FixVStems`). The Type 1 font can be visualized in a proof sheet form containing the control point labels (numbers) and hinting zones (figure 11).

So far, so good. But there are two crucial problems. First, the METAFONT Malayalam fonts designed by Jeroen Hellingman [5], use the command

```
currenttransform := currenttransform
                    shifted (.5rm, 0);
```

So all the glyphs should be shifted to the right. METAFONT saves the transformation command and does this operation automatically. By contrast, in METAPOST we need to insert the `shift` commands explicitly in all glyph programs. Also the labels must be shifted! In my experiments, I did this shift operation later, before final assembly of the Type 1 fonts.

The second problem is that in MetaType1 (I used MetaType1 version 0.40 of 2003) a regular pen stroking algorithm is not available, only a simplified method of connecting the points 'parallel' to the

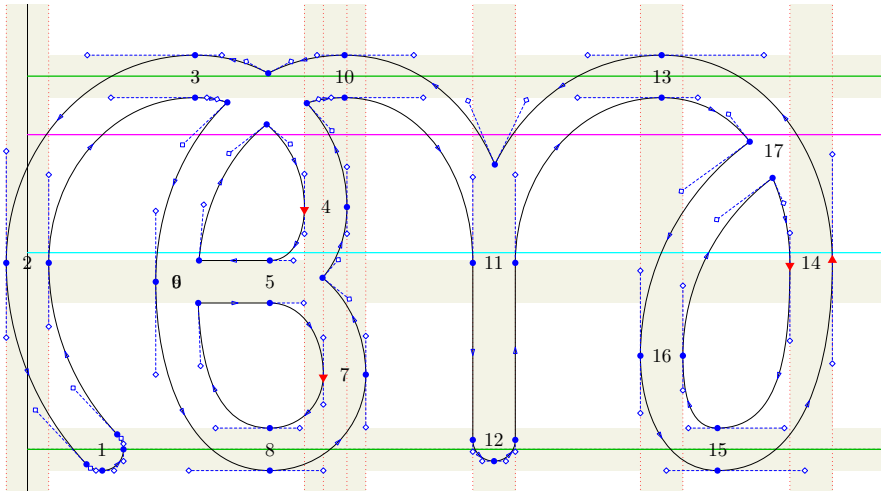

Figure 11. MetaType1 — proof sheet.

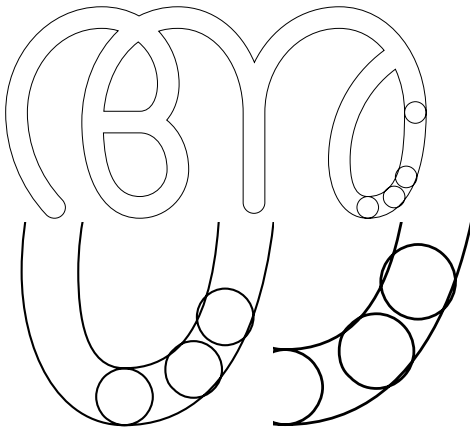

Figure 12: MetaType1 — bad pen stroking.

nodes on the path. Therefore the approximation of the envelope is not correct. For example, in Figure 12 it should be asymmetric, but it is symmetric. Inserting additional nodes cannot help, because the bisection results will again be asymmetric. The figure shows the outline curves do not correspond to the real pen in two midpoint locations. The envelope there looks narrow and it is in fact narrower than it should be. I hope that this problem could be solved in a future release, at least for pen stroking with a circular pen.

Even more serious is a situation with the rotated elliptic pen used in the Devanagari fonts designed by Frans Velthuis [19] (and also other Indic fonts derived from `dvng`). Absence of a regular pen stroking in MetaType1 makes it impractical for such complicated fonts. MetaType1 approximates the pen statically in path nodes, tries to connect their static end points, and ignores complicated dynamic correlations between the path, the pen and the envelope. Unfortunately, in this case the results of the envelope approximation are not correct and cannot be used (figure 13).

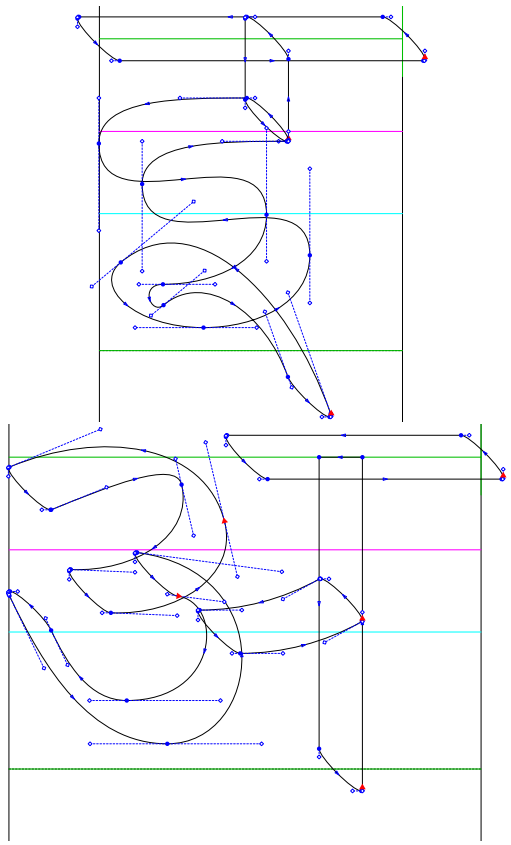

Figure 13: MetaType1 — Devanagari “i”, “a”.

3.2 MetaFog

Two programs using analytic conversion were presented in 1995. Basil K. Malyshev created his BaKoMa collection [14] and Richard J. Kinch developed MetaFog [11]. BaKoMa is a PostScript and TrueType version of the Computer Modern fonts. Malyshev's paper discusses some problems of conversion, especially regarding hinting, but his programs and detailed information about the conversion algorithm are not available.

R. Kinch created MetaFog along with weeder, which supports interactive processing of outlines, and a package for making final fonts from outlines generated by MetaFog in TrueType, Type 1 and other formats. MetaFog itself (I used an evaluation version graciously donated by Richard) reads the METAPOST output from the command:

`mpost '&mfplain` *options*`;' input` *fontname*`.mf`

Thus, the conversion (from METAFONT sources) is limited to fonts that can be processed by METAPOST, that is, do not contain METAFONT-specific definitions and commands. MetaFog generates another PostScript file consisting only of the outline structures. A conversion process is described also in the paper written by Taco Hoekwater [7].

MetaFog evaluates outline contours and precisely computes envelopes of an elliptical pen stroking along a Bézier curve. We must notice that the envelopes in general are not cubic Bézier curves and their representation in a Type 1 font must be an approximation. The results for a circular pen, on the other hand, can be considered perfect. Figures 14 and 15 show an example of the Malayalam letter “a” (font `mm10`): the initial and final contours and the final Type 1 font with control points (stroked

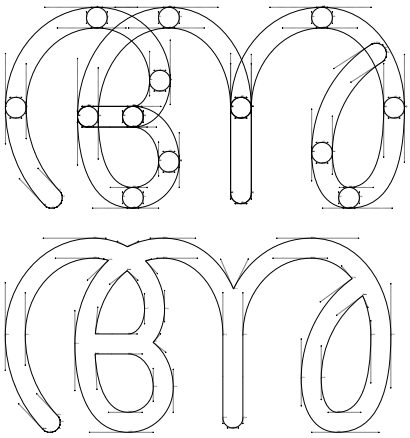

Figure 14: MetaFog — initial input contour and final result.

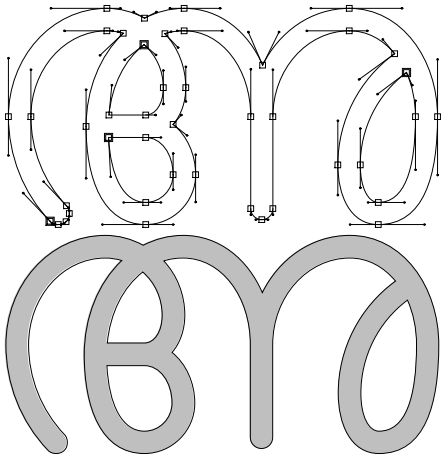

Figure 15: MetaFog — final Type 1 font.

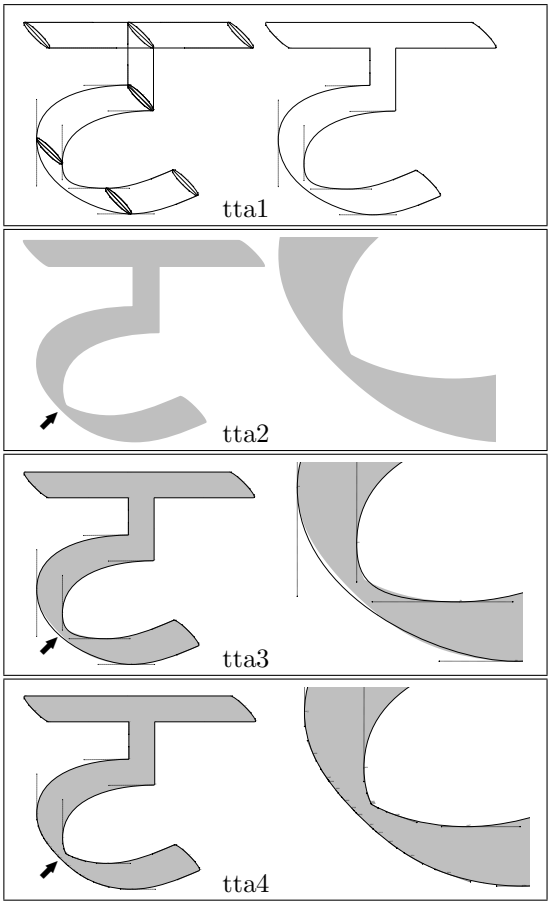

Figure 16: MetaFog contours, METAPOST output, primary and secondary conversion on the METAPOST background.

version) and its visual comparison with METAPOST output embedded in a Type 3 font, respectively.

3.2.1 Problems with Complex Pen-stroking

A more complicated situation is the conversion of fonts using pen stroking with a rotated elliptical pen, such as the Devanagari font. Figure 16 illustrates this case. The initial input contour and final result contour (tta1) look good — in the first image we can see the projections of the pen in nodes corresponding to METAFONT source. But exact comparison with the original METAPOST output embedded in a Type 3 font (tta2) and primary MetaFog conversion displayed together with the METAPOST source (tta3) shows that this approximation is not correct. Because these elements are very common in shapes of all but the simplest Devanagari glyphs, corrections are necessary.

I therefore applied a simple pen-dependent preprocessing step before the MetaFog conversion, thus adapting the METAPOST output as a modified form of bisection, as discussed in a paper by R. Kinch [11]. The preprocessing scans curves, searching for points where the path direction and the direction of main axis of the pen coincide (namely 135°) and inserts these points as additional path nodes. In our case, the transformation matrix is $\cos\theta * [1, 1, -1, 1]$, so we solve only a quadratic equation and can find 0, 1 or 2 (at most) of these points. This technique corrects the MetaFog approximation of all such occurrences in the `dvng` font. The result of this secondary MetaFog conversion with METAPOST source is shown in the last panel of Figure 16 (tta4).

Similar improvements for the Devanagari letters “a” and “pha” are shown in figure 17. For “pha”, the first 135 degree node was already present

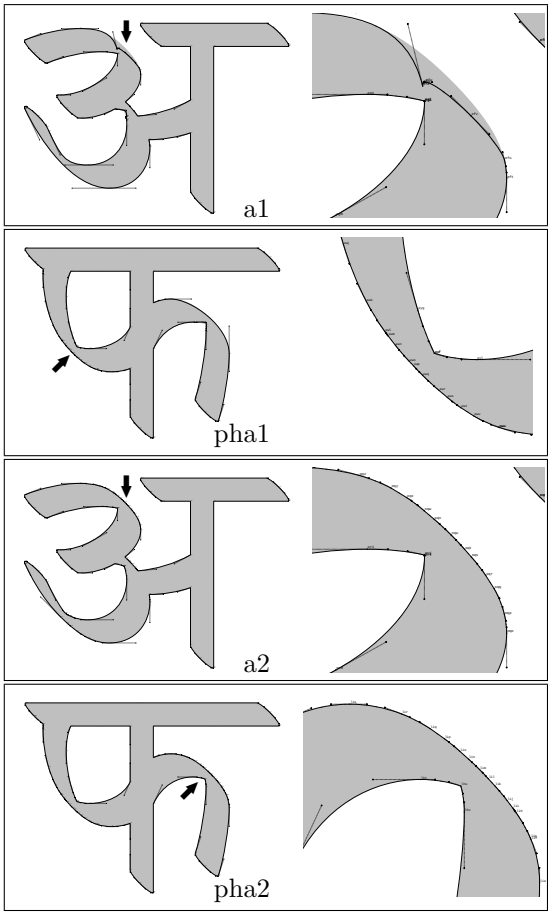

Figure 17: MetaFog output before and after modification of METAPOST source.

in the path defined by the METAFONT source (first panel, pha1); on the contrary, the second occurrence of a 135 degree point was absent, and therefore it was inserted in the METAPOST output (last panel, pha2).

Of course, this improvement is not universal, it only solves a special problem with a special pen for a special font.

Figure 18 illustrates movement of a rotated elliptical pen stepping along a "nice" path (panel 1). However, correlations with the pen are not trivial: changes of curvature of the outer wingtip curve do not have simple monotonic behavior, and the inner wingtip curve (panel 2) is even more complicated. This means that the pen-stroked wingtip curves along a single Bézier curve cannot be approximated by single Bézier curves (compare with the starting fig. 16, panel tta1), i.e., an envelope edge of a pen along a *simple* path is *not simple*.

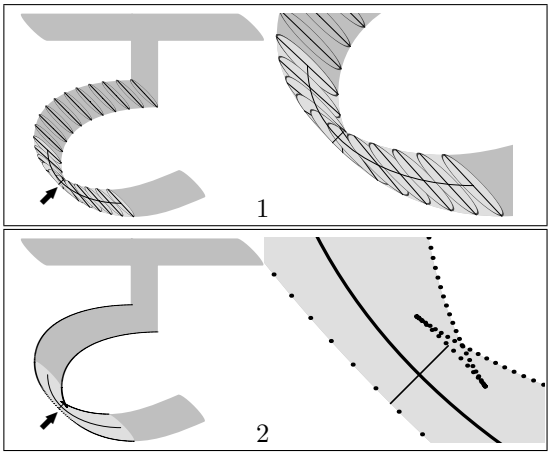

Figure 18: Wingtip curves in METAPOST source.

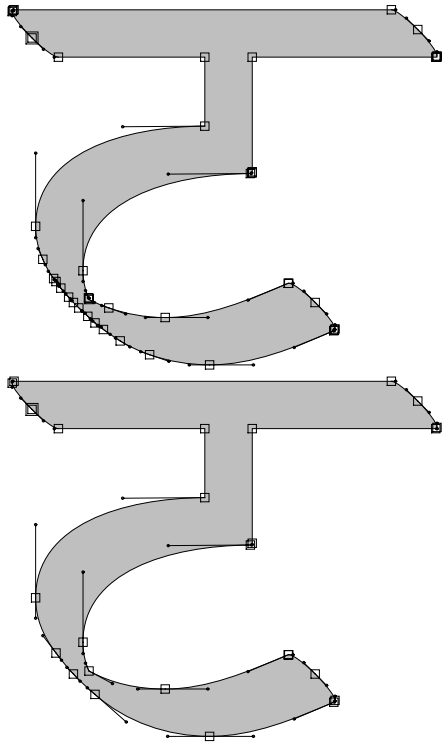

Figure 19: MetaFog converted to Type 1 — before and after postprocessing.

3.2.2 Automatic Conversion Problems

A "dark side" of improving the curve approximation is a fragmentation of an envelope curve into many segments (often more than 10, and up to 16 in Devanagari!). We achieve a faithful approximation (limited only by numerical accuracy) at the expense of conciseness. To make up for this, postprocessing is needed. The original MetaFog output and a

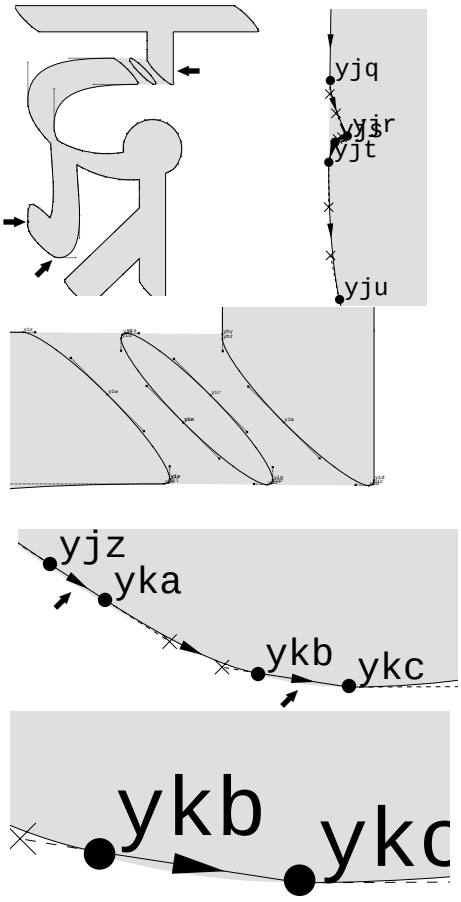

Figure 20: MetaFog — problems with automatic conversion.

result of my (preliminary) optimization assembled into Type 1 fonts is shown in Figure 19.

Unfortunately, even a small computational inaccuracy can make automatic conversion and optimization impossible, and even make it very difficult to design postprocessing algorithms. In Figure 20, we demonstrate problems with the primary approximation of an envelope stroked by a rotated elliptical pen, and also difficulties with automatic optimization of the Devanagari ligature "d+g+r".

In the first panel of fig. 20, we observe an artifact produced by MetaFog due to a complicated correlation of the pen and the path. Fortunately, those cases are very rare (less than 1 % of glyphs in Devanagari).

In the second panel, the path and subsequently the corresponding envelope edges are not absolutely horizontal, thus (probably) MetaFog cannot properly find intersection points and join reconstructed outline components. Those defects are present in more than 12 % of the Devanagari glyphs. In all cases, they have been successfully solved manually by the interactive weeder program.

In the last two details in fig. 20 (the lower ending part of the left stem) we can see that both nodes of the left segment are outside the filled area boundary defined by the METAPOST curve. The outer wingtip edge is split there into many segments, some being straight lines — and they should not be, e.g., the first and the third segment marked by 2 arrows in the clip — their curvatures are for us undefined. Additionally, we cannot detect the last segment (magnified in the figure) as horizontal because its angle is "greater than some ε".

Thus, neither node coordinates, nor segment directions, nor curvatures are reliable. It gives a *visual* comparison of the METAPOST output with its outline approximation. Therefore, my (first and "simple") idea cannot succeed. This was to classify the behavior of directions and curvatures of *all* the segments *automatically*, and then to divide segments into groups according to directions and curvatures, then *automatically* merging the groups to single Bézier segments. As demonstrated, this optimization may fail or produce incorrect results and, unfortunately, human assistance is needed.

4 Summary

Here we summarize the most important features of the conversion programs found in our experiments.

4.1 Approximate Conversions: TEXtrace, mftrace

Advantages:

- approximation covers original METAFONT fonts and correspondence to `pk` bitmaps is (reasonably) close
- simple invocation, robust solution
- fully automatic processing can generate complete, final Type 1 fonts

Disadvantages:

- approximate conversions give only approximate outlines
- lost information about nodes and other control points
- final fonts do not satisfy the *Type 1 conventions*
- AutoTrace: problems with recognizing corners, generation of unwanted bumps and holes
- potrace: sharp connections, thus loss of tangent continuity, violation of horizontal or vertical directions
- automatic and correct (auto)hinting may yield poor results due to these irregularities

4.2 MetaType1

Advantages:

- complete support for Type 1 font generation
- manual insertion of hinting information possible via simple hinting commands
- font file compression via subroutines

Disadvantages:

- conversion of METAFONT to METAPOST often requires manual rewriting, possibly non-trivial and time-consuming
- bad pen stroking algorithm; in particular, results for complicated fonts using rotated elliptical pens are unusable
- difficulties with removing overlaps in tangential cases

4.3 MetaFog

Advantages:

- fully automatic conversion of METAPOST output to outlines
- "typical" fonts usually achieve perfect results
- even for very complex fonts (again, with rotated elliptical pens), adaptations of METAPOST output and manual editing with weeder make it plausible to obtain perfect outlines
- results fulfill the *Type 1 conventions* in most cases (except for those very complex fonts)

Disadvantages:

- MetaFog reads METAPOST output, thus cannot process METAFONT-specific definitions
- complex fonts may still need manual reduction with weeder or subsequent optimization of outlines to reach conciseness
- processing is slow

4.4 Final Font Processing and Common Problems

The conversion systems discussed here, with the exception of MetaType1, do not include internal hinting subsystems. To insert hints, we can use font editors, for example FontForge [22]. For successful automatic hinting, however, the font outlines must fulfill certain conditions. Irregularities — absence of nodes at extrema or presence of bumps and holes — are not compatible with autohinting, because extrema points correspond to hinting zones while bumps or holes do not fit them, thus causing outliers. The resulting difference of ±1 unit in the integer glyph coordinate system, after rounding to integers, is not acceptable for high-quality fonts. Problems may also be caused by other "rounding to integer" effects, and by the presence of close doublets or triplets.

In my view, these experiments show that the quality of primary outline approximation is crucial to achieve perfect final Type 1 fonts. It is virtually impossible to recreate discarded METAFONT information, or to find exact conditions for a secondary fit that corrects primary contours that were created with irregularities or artifacts. Starting with high-resolution bitmaps is problematic, as too much information has been lost, making subsequent processes of improvement, optimization and hinting difficult at best, not possible to automate and usually not successful.

5 Acknowledgements

I would like to thank all the authors of the free conversion programs, Richard Kinch for donating his MetaFog and weeder, the authors of the public METAFONT fonts for Indic languages and other sources used in the contribution, and Karl Berry for help with editing of this article.

References

[1] Adobe Systems Inc. *Adobe Type 1 Font Format*. Addison-Wesley, 1990.

[2] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1988.

[3] Karl Berry. "Making outline fonts from bitmap images." *TUGboat*, **22**(4), pp. 281–285, 2001.

[4] Free Software Foundation. GNU awk, `http://www.gnu.org/software/gawk`.

[5] Jeroen Hellingman. Malayalam fonts, `CTAN:language/malayalam`.

[6] John D. Hobby. A User's Manual for METAPOST. AT&T Bell Laboratories, Computing Science Technical Report 162, 1994.

[7] Taco Hoekwater. "Generating Type 1 Fonts from METAFONT Sources", *TUGboat*, **19**(3), pp. 256–266, 1998.

[8] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk. "MetaType1: A METAPOST-based Engine for Generating Type 1 Fonts", *Proceedings of the XII EuroTEX 2001 conference*, pp. 111–119, Kerkrade, the Netherlands, 23–27 September 2001.

[9] Bogusław Jackowski, Janusz M. Nowacki, Piotr Strzelczyk. "Programming PostScript Type 1 Fonts Using MetaType1: Auditing, Enhancing, Creating. *Preprints of the XIV EuroTEX 2003*

conference, pp. 151–157, Brest, France, 24–27 June 2003 (to appear in *TUGboat*).

[10] MetaType1distribution: ftp://bop.eps.gda.pl/pub/metatype1.

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

[12] Donald E. Knuth. *The METAFONTbook.* Addison-Wesley, 1986. Volume C of *Computers and Typesetting.*

[13] Eddie Kohler. t1utils (Type 1 tools), http://freshmeat.net/projects/t1utils.

[14] Basil K. Malyshev, "Problems of the conversion of METAFONT fonts to PostScript Type 1", *TUGboat*, **16**(1), pp. 60–68, 1995.

[15] Han-Wen Nienhuys. mftrace, http://www.cs.uu.nl/~hanwen/mftrace.

[16] Karel Píška. "A conversion of public Indic fonts from METAFONT into Type 1 format with TEXtrace." *TUGboat*, **23**(1), pp. 70–73, 2002.

[17] Peter Selinger. potrace, http://potrace.sourceforge.net.

[18] Péter Szabó. "Conversion of TEX fonts into Type 1 format", *Proceedings of the XII EuroTEX 2001 conference*, pp. 192–206, Kerkrade, the Netherlands, 23–27 September 2001. http://www.inf.bme.hu/~pts/textrace; http://textrace.sourceforge.net.

[19] Frans J. Velthuis. Devanagari fonts, CTAN:language/devanagari.

[20] Vladimir Volovich: CM-super fonts: CTAN:fonts/ps-type1/cm-super.

[21] Martin Weber. AutoTrace, http://autotrace.sourceforge.net.

[22] George Williams. FontForge: A PostScript Font Editor, http://fontforge.sourceforge.net.