Converting METAFONT Sources to Outline Fonts Using METAPOST

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

Abstract

The paper describes a multistep conversion process from METAFONT sources to outline fonts (Adobe Type 1 format). An important step, finding contours, is based on an accurate algorithm fitting the envelope curve of a stroke drawn by a pen along a cubic Bézier curve by the least square method, specially extended (adapted) for a rotated elliptical pen applied, for instance, in the Devanagari font design. After converting the EPS files produced by METAPOST to the corresponding outline representation the FontForge font editor is used for removing overlap, simplification, autohinting, generating outline fonts, and necessary manual modifications. The result of conversion, the faithful Indic Type 1 fonts (significantly close, precise and optimal than earlier attempts made by autotracing bitmaps) will be released.
KEYWORDS: font conversion, bitmap fonts, METAFONT, METAPOST, outline fonts, PostScript, Type 1 fonts, approximation, Bézier curves.

Introduction

In 2001 I experimented with approximate conversion METAFONT Indic fonts to the Type 1 format by autotracing bitmaps with the TEXtrace program [11]. I was not satisfied with results and decided to apply another, analytic approach, to achieve results more precise and also more optimized.

Conversion Process

A procedure consists of study of font definitions in METAFONTand preparing encoding files; then the glyph strokes produced by METAPOST are converted to outlines, the font is assembled, optimized, autohinted, and finally, generated as a Type 1 binary file with FontForge. After verification of visual proofsheet pages some steps are often repeated to correct or improve the final results.

Analysis of METAFONT sources We analyze the METAFONT source texts [7] of a font to select an appropriate strategy of conversion, to find the crucial parameters, like the font size, the italic angle, definitions of pens and strokes. Some parameters may be also hidden inside macros. Sometimes, a possibility of an efficient conversion is not apparent. Therefore it is also important to know about presence and quantity of METAFONT commands not available in METAPOST([5]), for example, using operations with bitmap picture variables.

Creating encoding files Encoding files and encoding vectors define a mapping between the glyph names and their number codes. METAFONT definitions usually do not contain unique glyph names in an explicit form but only comments. The glyph names have been taken from these comments to produce unambiguous list of PostScript names, i.e. we must to find the same names and to change them to be different. Our preliminary solution inherits METAFONT comments closely to make finding glyph identification easier.

Running METAPOST Invoking METAPOST processes the METAFONT sources and produces the EPS files. METAPOST together with a macro package `mfplain` ([5], p. 79) allows to process the original or modified (to eliminate METAFONT-specific commands) font sources written in METAFONT and to generate for each glyph a single file in the Encapsulated PostScript format, consisting only of PostScript commands like curves, strokes, affine transformations representing pens, etc., but no bitmap images contradictory to the METAFONT standard output. Some metric data, e.g. the glyph widths

Figure 1: Result of METAPOST.

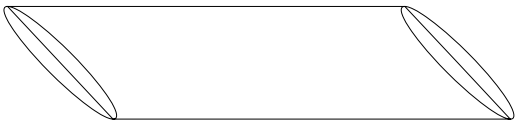

Figure 2: Primary conversion to outlines.

and also the italic angles, may be lost, we shall restore them later. We need also define a magnification factor. Because we have to transform the glyph images to a 1000-unit glyph coordinate system (we use this usual space) with the units in PostScripts *big* points (the transformation factor is 1.00375) and the font *designsize* in *pt* units. Then the magnification factor will be $1000 * 1.00375/designsize$. For the $designsize = 10\,pt$ it equals $1000 * 1.00375/10 = 100.375$, for $8\,pt$ 125.46875, for $17.28\,pt$ 58.087384, etc. Then a typical command to call METAPOST is:

```
mpost '&mfplain \mode=localfont;' \
  mag=100.375'; input' dvng10.mf
```

These files may contain various stroked paths (see figures 1, 9). It is necessary to find contour curves for single strokes and then also common envelope curves for overlapping strokes.

The following lines from the PostScript produced by METAPOST correspond to fig. 1:

```
 0 79.06227 dtransform truncate idtransform
 setlinewidth pop [] 0 setdash
 1 setlinecap 1 setlinejoin 10 setmiterlimit
gsave newpath 119.50958 284.54501 moveto
398.36119 284.54501 lineto
[-0.98387 0.98387 -0.17888 -0.17888 0 0] concat
stroke grestore
```

The `lineto` operator describes the line segment, the `concat` operator applies the affine transformation represented by the preceding normalized matrix (in brackets) denoting the rotated elliptical pen, and 79.06227 ... `setlinewidth` is the scale factor defining the stroke width.

Converting METAPOST products to outlines The results of METAPOST (strokes) are converted to "primary" outlines. To fit curves with the least square method is a typical approach to calculate a curve approximation. This method is nothing new and probably it has been used in conversion programs developed by Richard Kinch (MetaFog, [6]), Basil Malyshev [9], George Williams (FontForge, [13]) and other. We only apply a few additional conditions. We try to be more precise, but our attempts are still more fragile and unstable than programs listed above.

All the calculations are in the non-integer value space. We check each segment for accuracy and subdivide it if a chosen limit exceed; insert all horizontal and vertical extrema nodes; keep all horizontal/vertical straight lines and control vectors to be exactly horizontal/vertical. The inner part of a contour curve of drawing a rotated elliptical pen even along a simple Bézier path without any intersection may have selfintersections. Therefore we try to find a selfintersection points if it is possible and as precise as possible. Unfortunately, sometimes this iteration does not converge. A simplest conversion to outlines shows figure 2.

For a given time of the path segment using the affine transformation matrix and its inverse matrix (for a usual pen they are always regular) we can calculate the displacement corresponding to the point lying on the right parallel outline curve (the left one is located symmetrically). Knowing the coordinates of points on the outline curves and also on the pen boundary we can fit them by a cubic Bézier approximation. But a problem is we do not know whether the points are an the envelope curve or not because parts of the outline curves may create loops of arbitrary size being inside a closed area. It depends on complex correlations between the path and the pen.

We also recognize quarter-circles usually represented in METAFONT by two segments because METAFONT tends to divide curves to octants. To avoid further simplification problems we do not preserve the 45 degree middle nodes and change the quarter-circles to the accurate single-segment PostScript representation with relative lengths of control vectors $4/3(\sqrt{2}-1) \simeq 0.552285$, compare also with R. Kinch [6] (p. 236) or Luc Devroye [2]. For an example of our approximation circles see figure 3.

In summary, in the primary approximation the straight lines and the circles are represented by the minimal number of segments (because other nodes are unnecessary), and, on the other hand, other outline curves have redundant node points (to preserve a maximal starting accuracy). The intermediate results of the primary conversion to outline demonstrate figures 2 and 10.

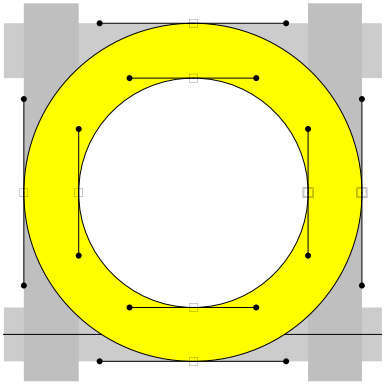

Figure 3: Representation of circles.

Creating a font with FontForge FontForge is a powerful open source font editor. Among its wide range of useful abilities we can find a background layer. It may contain bitmap images and line drawings. Therefore, we generate by METAFONT a high resolution bitmap 7254 *dpi* or 2400 *dpi* (supre) for a given font. The "7254 dpi" device corresponds to a relation 1 pixel in PK $\sim$ 1 unit in the PS glyph space for the 10 *pt*

```
% (72.27*1000.375/10dpi)=7254.1
mode_param (pixels_per_inch,4000+3254.1);
mode_param (blacker, 0);
mode_param (fillin, 0);
mode_param (o_correction, 1);
```

Sometimes, METAFONT with the very high resolution may fail (if the author did not design a font for an arbitrary resolution). The the PK or GF files can be imported to the background as a set of gray pixels to cover glyph images.

Font composition We also run mftrace [10] with an appropriate encoding to make a PFB font file. From this file we build a frame for the created font, copy the glyph widths and the glyph names and move the outlines to the background layer (visible as green lines). During a subsequent processing of the font with FontForge we use its internal Spline Font Database format (SFD). The high resolution bitmap is always huge, we import it only before a comparison. But the outline contours of the font produced by mftrace are not large and we can store them in the working SFD files permanently. To the foreground layer we import the outlines from the EPS files calculated in the previous step from the original EPS files generated by METAPOST.

The high resolution pixel image gives a close visual bitmap representation of the original METAFONT source. Of course, an information about contour curves, intersection points, corners, etc., virtually calculated by METAFONT has been lost. The font outlines autotraced by mftrace from similar bitmaps, despite of the artifacts (bumps, holes, unrecognized corners, ...) give a correct information about glyphs. And our aim is to obtain another outline representation: more accurate and more optimal, to minimize the number of defects and a space amount.

Having a font in the SFD format built from the mftrace output our next step with FontForge is **removing overlap and optimization** (simplification). We continue processing in the non-integer value space to keep accuracy, especially do not change the slopes of the neighbor control vectors to preserve smooth transition between segments.

Rounding to integer, hinting and Type 1 font generation FontForge allows generating PostScript fonts with non-integer point coordinates and, maybe, many PostScript RIP devices render these fonts properly. But we have three significant reasons to round coordinates to integer and to generate the Type 1 fonts in integer representation:

- Non-integer values in the PostScript charstring occupy 3 items. Therefore the integer representation saves storage and the PFB files are smaller.
- The final Type 1 fonts do not need such accuracy after removing overlap and simplification.
- For hinting it would be inconvenient and impracticle to use a different discrete grid than integer.

In the following example the non-integer Type 1 command occupies 19 items:

```
18153 100 div 212 100 div
14437 100 div -407 100 div
7208 100 div -243 100 div
rrcurveto
```

and after rounding only 7 items:

```
182 2 144 -4 72 -2 rrcurveto
```

It is reasonable to minimize the number of items because the PostScript interpreters have internal memory limits per glyph. Exceeding limits causes a limicheck error and a crash of rendering.

The coordinates of the segments are rounded to integer by more complex algorithm than a trivial rounding of all the values. First we round the node points. Then we transform the control vectors according the changes of then nodes and try to find the control points in the integer grid near the transformed control vectors. Even this sophisticated

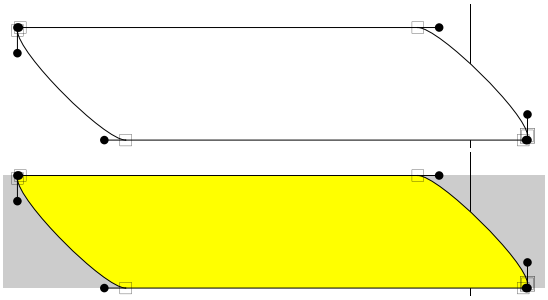

Figure 4: Final font in an outline form and a hinted proofsheet (clip).

rounding to integer is not without problems. Sometimes, if the change in x or y in the segment is very small (e.g. about 1 unit) or a segment is too short (in both directions) no good selection may exist and a manual adjustment is then necessary, probably with the lose of closeness, accuracy or symmetry of approximation.

No special additional program for hinting have been developed or applied. An automatic autohinting tool of FontForge is used and any unsatisfactory events should be corrected manually.

Finally, FontForge generates the Type 1 binary font, usually rounded to integer and (auto)hinted.

Results

To make font audit and verification more quick and efficient we developed tools for generation of visual proofsheets in PDF: to allow fast overlook all glyph images, outlines curves with node and control points and vectors, hinting zones, and also to detect some situations like missing nodes at extremes, presence of inflection inside a segment, connection between segments is not smooth, etc., and to append special warning signs. Our aim is to fulfill the *Type 1 conventions* [1]. Therefore we include the extrema nodes (they may be omitted if they are really redundant), exclude other unnecessary node points, preserve smooth connections between the adjacent segments. and also keep the straight lines, corners and arcs after conversion, do not append any false bumps, holes or steps absent in the original METAFONT sources. In some selected figures the node points (squares), the control points (bullets) and the control vectors have been enlarged to be visible in the printed version of the paper. In a real working process they are colored and small as in other proofsheets when we zoom interesting details only if we need to check them.

The crucial and auxiliary algorithms have been under development and adaptations for new fonts

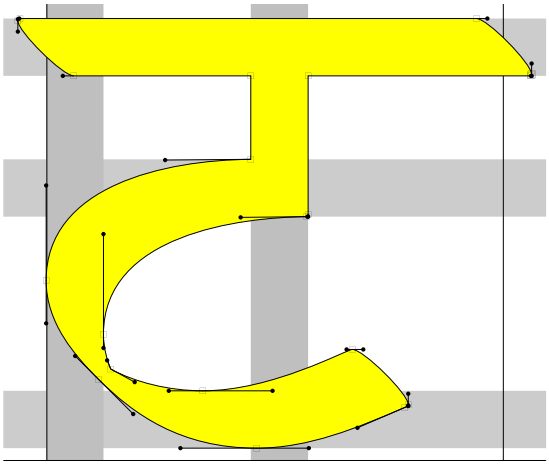

Figure 5: dvng10: tta of Frans Velthuis.

and the programs are still written in `awk` or `gawk` [3]. For Type 1 font handling `t1utils` [8] are used.

Several pictures illustrate intermediate and final results of conversion METAFONT fonts to the Type 1 format: figures 2, 4, 10, 11, 15, and 16.

Indic Fonts A basic goal of the work are more precise outline versions of the free METAFONT Indic fonts available from CTAN: Devanagari, Sanskrit, Gurmukhi, Punjabi, Bangla, Sinhala, Malayalam, Telugu, Kannada, Tamil, and Tibetan is also included. During preparing this text not all the present fonts have been converted and also the Oriya fonts are still missing because of they widely use METAFONT bitmap picture commands. Next results are shown in figures 12, 13 (Devanagari), 14 (Malayalam).

Chinese Fonts We have also tried to convert two small single fonts with Chinese signs created in METAFONT: the Hóng-Zì font (128 glyphs) designed by Javier Rodríguez Laguna [12] (version 0.5 of 050323): fig. 7; and china10, one font from the the china2e package [4] containing Chinese calendar symbols produced by Udo Heyl (1997): fig. 8.

Conclusion

In the article we describe a conversion process and shortly discuss some selected problems. Creating *precise* fonts is always difficult, time consuming and never ending work independently of the approach we choose. We plan to verify again all the glyphs to improve hinting and polish the outlines to remove tiny artifacts. It is useful to make the glyph names of the Indic glyphs common for all languages, it is not trivial because the fonts contain many various

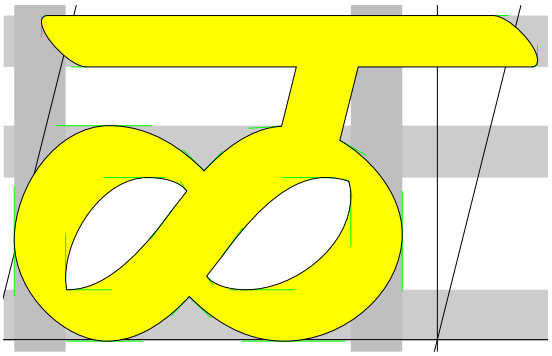

Figure 6: dvngbi10: lla of Frans Velthuis.

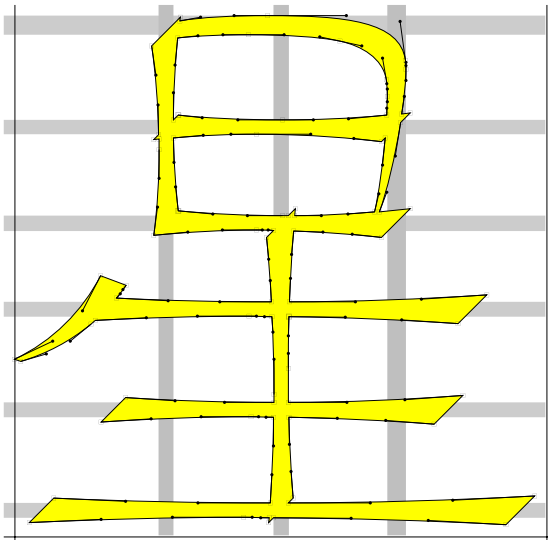

Figure 7: Hóng-Zì: xing1 of Javier Rodríguez.

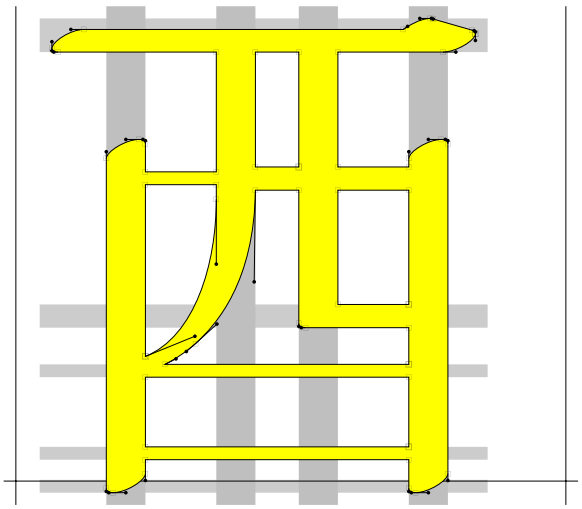

Figure 8: china10: yeu of Udo Heyl.

ligatures, special signs or variants not covered in the Unicode standards.

Acknowledgements

I would like to thank all the authors of the free conversion programs, the authors of the public METAFONT fonts for Indic languages, other sources and program packages used in the contribution,

References

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

[2] Luc Devroye. "Formatting Font Formats", *TUGboat*, **24**(3), pp. 588–596, 2003.

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

[4] Udo Heyl. `CTAN:macros/latex/contrib/china2e`, 1997.

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

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

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

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

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

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

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

[12] Javier Rodríguez Laguna. Hong-Zi – A Chinese METAFONT. `http://hongzi.sourceforge.net`, 2005.

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

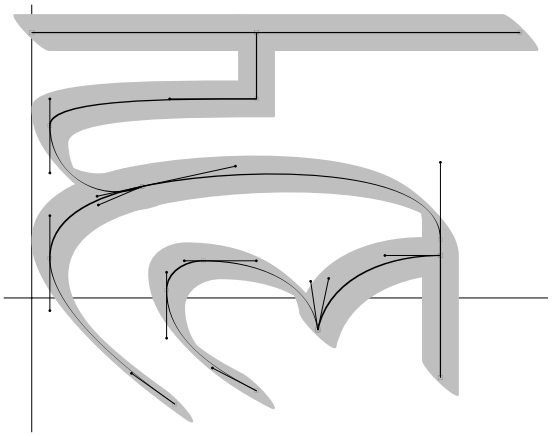

Figure 9: dvng10 l_h: METAPOST output.

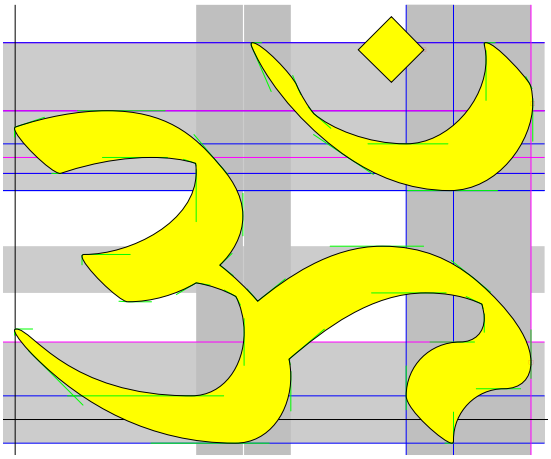

Figure 12: dvng10: om of Frans Velthuis..

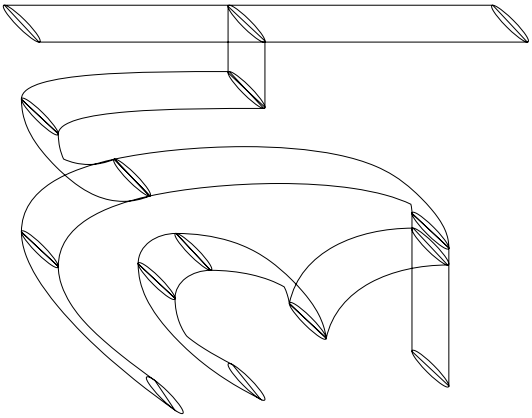

Figure 10: dvng10 l_h: primary outlines.

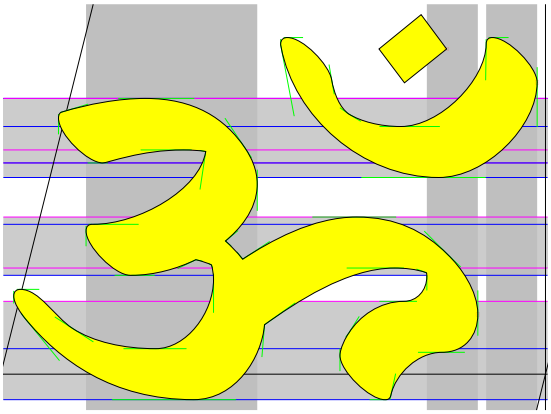

Figure 13: dvngbi10: om of Frans Velthuis.

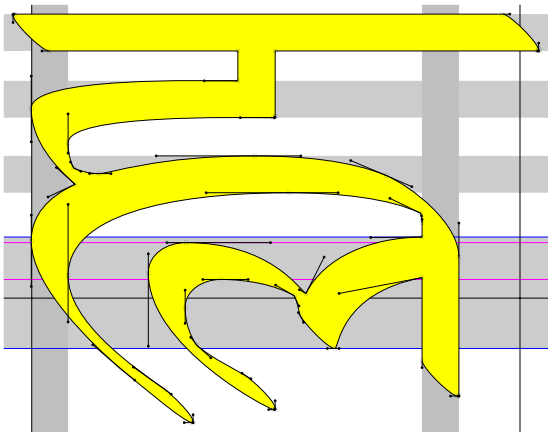

Figure 11: dvng10 l_h: Type 1 font proofsheet.

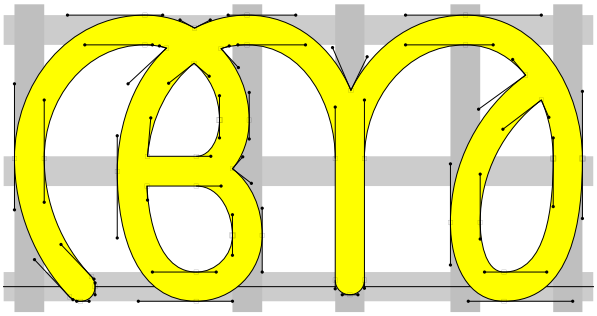

Figure 14: mm10: a of Jeroen Hellingman.

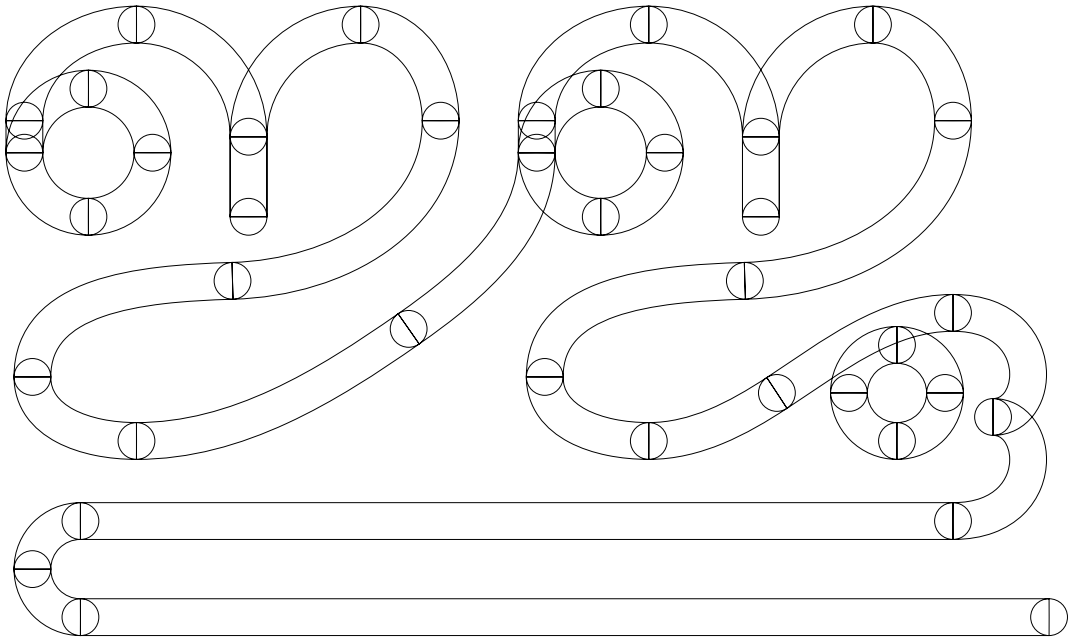

Figure 15: mm10 j_juu: METAPOST output converted to primary outlines.

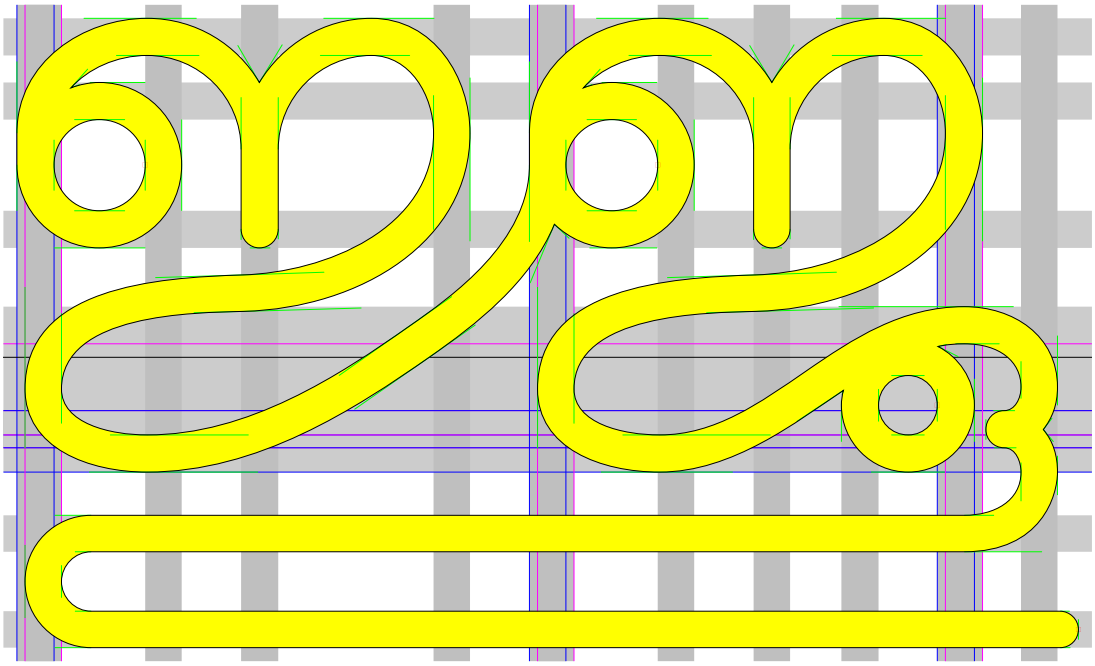

Figure 16: mm10 j_juu: Type 1 font proofsheet with hints.