TEX Gyre text fonts revisited

Bogusław Jackowski, Piotr Pianowski,
Piotr Strzelczyk

1 Introduction

The collection of the TEX Gyre (TG for short) family of text fonts, an extensive revision of the freely available 35 base PostScript fonts, was released by the GUST e-foundry in 2006–2009 [4, 6]. Having finished this task, the GUST e-foundry team started to work on the math companion (in the OpenType, OTF, format [7]) for the TG text fonts [5]. Work on the math companion was finished two years ago. It resulted in the broadening of the repertoire of glyphs that could be used not only in math mode but also in text mode in technical documents. Hans Hagen, indefatigably coming up with interesting ideas, proposed to migrate the relevant glyphs to the text TG fonts. Needless to say, we seized on Hans's suggestion.

The first step was to decide which glyphs are to be migrated (and/or improved). Obviously, the list of candidates grew and grew. All in all, about 1000 glyphs were designated to be added, mostly geometrical and math symbols. A math companion, so far, was provided only for serif fonts, thus the consistent enhancement of the repertoire of the sans-serif fonts was a working test for our font generator — cf. Section 2 below.

We started with two fonts — the serif TG Pagella and the sans-serif TG Adventor. The results were satisfying. Now we are ready for the next step: to enhance similarly the rest of the TG family (TG Chorus, which is hardly suitable for technical texts, needs an individual approach). We believe, however, that we're over the hump. Below, we describe the most difficult and thus most interesting (to us) aspects of this stage of the TG project.

2 The MetaType1 engine

The scheme of the new workflow for the authors' MetaType1 software is depicted in Figure 1.

The main change in the engine consists of the replacement of several components (AWK plus Perl plus T1utils) by Python code with the FontForge library (finally, the library is available both under Unix and Windows). However, the FontForge library does not allow for sufficiently detailed control over the contents of the AFM and PFM files being generated, necessitating additional steps for fine tuning these files (dashed arrows in Figure 1).

First published in *Die TEXnische Komödie* 3/2018, pp. 11–20. Reprinted with permission.

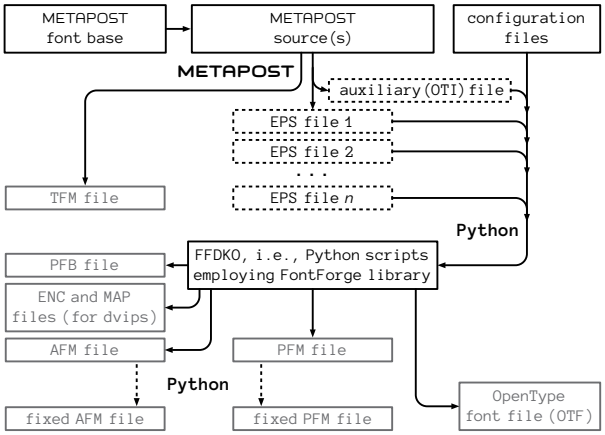

Figure 1: New MetaType1 engine: working scheme

The converter from Type 1 fonts to MetaType1 sources, implemented in AWK plus T1utils, has not yet been rewritten. We plan to rewrite it in Python with the FontForge library and enhance it to also process TrueType and OpenType files.

Of course, MetaPost is still the main module for generating glyph shapes. However, instead of spreading the auxiliary information into several output files (including EPS files), a single auxiliary output file, containing all the information needed for further processing, is generated. We will refer to this file as an *Olio Typographic Information* file, OTI. (*Olio* is a traditional name for a potpourri; it appears, e.g., in Robert Burns's *Address to a Haggis* — "French ragout or olio"). An OTI file is a container of "assorted bites and fragrances", indeed. Below is a fragment of an OTI file for TG Pagella Regular.

```
FNT FAMILY_NAME TeX Gyre Pagella
FNT HEADER_BYTE49 TeX Gyre Pagella
FNT GROUP_NAME TeX Gyre Pagella
FNT STYLE_NAME Regular
. . .
FNT WEIGHT Regular
FNT ITALIC_ANGLE 0
. . .
GLY A CODE 65
GLY A EPS 165
GLY A ANCHOR INBAS ALT.ogonek 623 -143
GLY A ANCHOR INBAS BOT_MAIN 392 -143
GLY A ANCHOR INBAS TOP_MAIN 392 819
GLY A WD 778 HT 692 DP 0 IC 6 GA 392
GLY A HSBW 778
GLY A BBX 15 -3 756 700
. . .
FNT FONT_DIMEN7 0.83
FNT DIMEN_NAME7 (extra space)
FNT FONT_DIMEN22 2.5
FNT DIMEN_NAME22 (math axis)
FNT HEADER_BYTE72 234
```

Each line of the OTI file contains either global information, concerning the whole font (prefix FNT), or local, concerning a given glyph (prefix GLY followed by the glyph name). We will not dwell too much on the details of the structure of OTI files as it will be documented elsewhere.

3 The glyph repertoire

As mentioned above, one of the important reasons for the "face-lifting" of the TG text fonts was our efforts on TG math fonts. Many symbols do not need the mathematical extension of the font structure (the MATH table in OTF files), but still prove useful in typesetting technical texts; for example, mathematical symbols (operators, relational symbols), arrows, geometrical symbols, etc. — see Figures 2 and 3. The number of glyphs grew from circa 750 to more than 1600, and may grow further in the future (see Section 5).

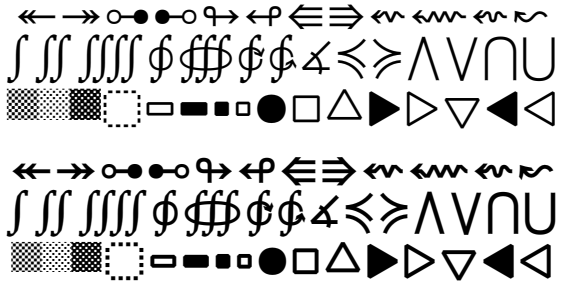

Figure 2: Sampling of added glyphs: TG Pagella regular (top) and bold (bottom)

The symbolic glyphs in the TG math fonts were designed only for regular serif variant fonts. The code, however, turned out to be flexible enough that with a few changes it was possible to generate bold and sans-serif variants.

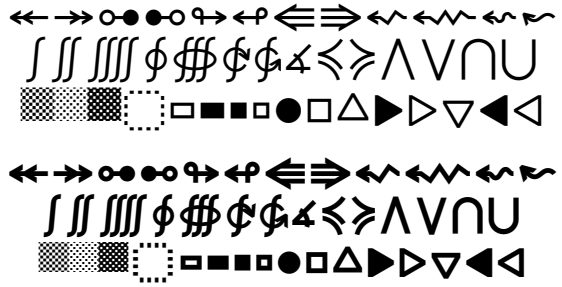

Figure 3: Sampling of added glyphs: TG Adventor regular (top) and bold (bottom)

Apart from enriching the repertoire, many glyphs were amended, due to, among other reasons, employing FontForge which, by default, minutely checks

$f(x) = 1/x$ $\quad$ $f(x) = 1/x$
$(x+1)(x-3) \geq 0$ $\quad$ $(x+1)(x-3) \geq 0$

Figure 4: Default math-oriented glyphs (left) vs. old glyphs produced by the OTF `ss10` feature (right)

glyph outlines. For example, a tilde in TG Adventor was drawn from scratch, axes in several glyphs were corrected and so on.

Math-oriented glyphs existing already in the text fonts have been replaced with slightly different forms, better suited for math formulas. The old forms can be reached, if required, by using the OTF mechanism called features [2, 7, 8], namely, the 'stylistic set' feature `ss10`. Moreover, the Pagella Greek alphabet was taken from TG Pagella Math, that is, from Diego Puga's excellent Mathpazo with the kind permission of the author who agreed to let us use a fragment of his font under the GUST Font License (GFL [3]). The latter change involves significant change of the metric data. We are generally very reluctant to introduce such changes, but believe that the elegance of the Mathpazo Greek alphabet justifies that decision. Some glyphs from the Greek alphabet of TG Adventor (programmed in MetaType1) required improvements which also implied changes in metric data.

Rolling with the punches, we decided to abandon our initial idea of full compatibility with the metrics of the renowned Adobe 35 fonts [1]. The reason is twofold: first, Adobe metric data is, as we pointed out in the documentation of the TG fonts [4], itself inconsistent in several cases; second, preserving full compatibility makes sense only when the relevant metric files are used for previewing PostScript files to be printed on a printer with built-in Adobe Type 1 fonts. The TG fonts might have been used for such previewing, but, as it turned out, they have not (either in Ghostscript or in TEX Live; for example, the URW replacements for the Adobe 35 are typically used). Eventually, we decided to tune the TG metric data according to our experience whenever required. We believe that we will manage to avoid such changes in the future.

4 The font structure

The structure of the OTF fonts has been enhanced with the "backward compatible math style" feature (`ss10`) mentioned above and, moreover, with the mechanism of *anchors*, although the name "snaps" seems to us to be more accurate. Anchors enable putting accents precisely over glyphs. Roughly speaking, the anchor mechanism can be considered the analogue of the TEX `\accent` mechanism. Anchors, however, are implemented in a much more intricate

way: three features, obscurely documented in [2, 8], namely, `ccmp` (glyph composition / decomposition), `mark` (mark positioning, precisely, accent-to-base or mark-to-base positioning), and `mkmk` (mark-to-mark positioning, or, in other words, accent-to-accent positioning)[1] are used for this purpose, and yet the OTF anchor mechanism turns out insufficiently efficacious.

We were surprised by the complexity and laboriousness of the implementation of such a simple concept. Having read the explanations below, the reader and our virtual successors should feel forewarned and thus be less surprised.

"Anchors" or "marks" are actually pairs of numbers (planar points); the features `mark` and `mkmk` are supposed to position two glyphs in such a way that the respective anchors of the accent and accentee coincide. The former feature is used to position accents over or below base glyphs, the latter to position accents over or below accents. In the TG fonts, following common practice, only so-called combining accents (a subset of the block of combining diacritical marks [9]; that is, zero-width glyphs, protruding entirely to the left) are used for accenting and, thus, are equipped with anchors. In order to reduce the amount of anchor data, we decided to use as anchored accentees only accentless Latin letters plus letters "welded" with cedilla, horn, ogonek, and, additionally, l·, Ŀ, ł, Ł, ø, and Ø.

The `ccmp` feature enables the transformation of the input stream, namely: replacing glyphs and assembling a series of glyphs into a composed character or disassembling a composed character into a series of glyphs. The respective substitutions, in principle, must be defined in the font. Some engines, however, know better and perform such substitutions even if the font lacks relevant data. For example, Microsoft Word replaces 'i' (U+0069) followed by a combining top accent, say 'caroncomb' (U+030C), by a single glyph 'icaron' (U+01D0), provided that the latter is available in a given font; no further information, in particular, no `ccmp` feature, is required. Similarly, XeTeX joins accents with the base glyph into a single glyph, provided that the assembled form is present in the font; otherwise, accents are placed using anchors. This behaviour cannot be turned off — XeTeX simply uses system libraries which know better...

In the TG fonts, the `ccmp` feature is used to disassemble accented glyphs (but not glyphs with cedilla, ogonek, or horns) and to join into a single glyph letters followed by combining cedilla, ogonek, or horn (provided that the resulting glyph belongs to the repertoire of the font); otherwise, anchors are used. Moreover, `ccmp` is used to replace certain base glyphs and accents by their alternative forms; for example, 'i' and 'j' in the vicinity of top combining accents are replaced by their dotless forms, while top combining accents following an uppercase letter or ascender are replaced by their 'high' (flattened) variants.

The process of accenting using anchors, seemingly a trivial task, is, in fact, quite sophisticated. The Unicode standard recommends that if a text processor is being fed with a stream of text data containing a glyph, having assigned a Unicode slot, which is followed by a series of combining accents, then the text processor may position these accents over the main glyph [10], provided that the font contains the relevant positioning information. A typical example of the application of the anchor mechanism involving the `ccmp`+`mark`+`mkmk` features (as implemented in the new TG fonts) is depicted in Figure 5.

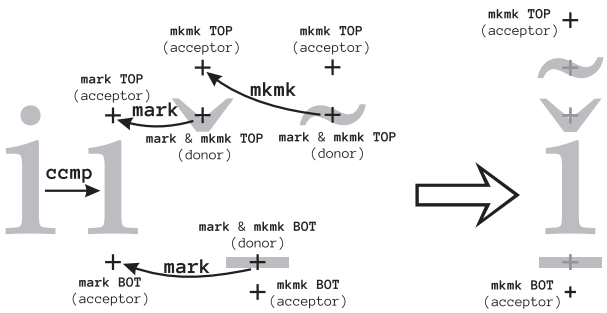

Figure 5: Anchor mechanism scheme — an example (explanations in text)

In the picture, feature names written in a small size denote the type of anchor (mark), large ones denote application of the respective features, labels 'TOP' and 'BOT' are defined by the user; the assumed input string is: 'i', 'macronbelowcomb', 'caroncomb', 'tildecomb' (that is: U+0069 U+030C U+0331 U+0303). The anchors have descriptors given in braces: *donor* and *acceptor* (taken from physical chemistry).

The process of accenting works here as follows:

- first, the `ccmp` feature enters the scene: the letter 'i', when followed by a combining upper accent, is replaced with 'dotlessi';
- next, the `mark` feature acts: the 'caroncomb' glyph is placed over 'dotlessi' in such a manner that its 'TOP' donor anchor coincides with the 'TOP' acceptor anchor of the glyph 'dotlessi'; as a result, both anchors become inactive;
- next, the `mark` feature enters once again: the 'macronbelowcomb' glyph is placed below 'dotlessi' in such a manner that its 'BOT' donor

[1] There is yet one more anchor feature `mset` (mark positioning via substitution) meant for handling peculiarities of the typesetting of Arabic texts.

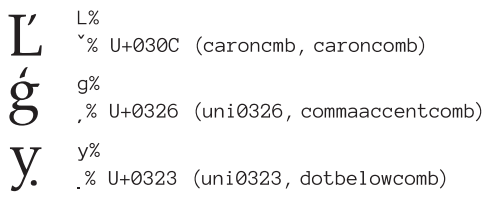

Figure 6: Peculiar positioning of certain accents (TEX source — right; the result — left)

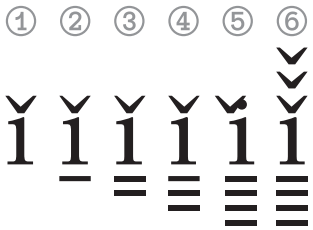

Figure 7: Troublesome replacement of 'i' by 'dotlessi' (explanations in the text)

anchor coincides with the 'BOT' acceptor anchor of the letter 'dotlessi'; as a result both anchors become inactive;

- finally, the `mkmk` feature intervenes: the 'tildecomb' glyph is placed above the newly placed 'caroncomb' in such a manner that its 'TOP' donor anchor coincides with the 'BOT' acceptor anchor of the 'caroncomb' glyph; as a result, both anchors become inactive;
- the resulting assembled glyph still has two active anchors, 'TOP' and 'BOT', that could be used by the `mkmk` feature, provided that the relevant glyphs appear in the input stream (immediately after 'tildecomb' in this case).

As one can see, the process of assembling glyphs using anchors is fairly complex. It should be admitted, however, that it enables handling such peculiarities as replacing a caron glyph with a comma-like variant if glyphs 'l', 'L' or 'J' are to be accented with caron, replacing a comma accent by a turned comma accent above 'g' (normally comma accent goes below a letter), or a singular positioning of a dot below accent at a letter 'y', as shown in Figure 6.

Unfortunately, not all cases of practical importance can be reliably handled. A notable example is the replacement of letters 'i' and 'j' by their dotless forms: the result depends to a large extent on the order of the glyphs in the input stream. In Figure 7, six cases are shown with different orders of glyphs in the input stream, namely (here, *i* stands for the letter 'i', *c* stands for 'caroncomb', and *m* stands for 'macronbelowcomb'): 1. *ic*; 2. *imc*; 3. *immc*; 4. *immmc*; 5. *immmmc*; 6. *icccmmmm*. Observe a malpositioned caron in case 5 — it is the result of our "design decision". The replacement 'i'→'dotlessi' is performed only if the top accents occur close to the letter 'i', preferably immediately after it. The OTF feature specification permits contextual replacements, that is, a certain number of bottom accents may precede the top one, but the preceding sequences must be enumerated explicitly. We decided to limit the length of the context to three glyphs (case 4 in Figure 7). If more bottom accents intervene between the letter 'i' and the top accent, the replacement is not performed and the glyphs are just overlapped (case 5 in Figure 7; as mentioned, combining accents have zero width and protrude to the left). Some fonts define longer contexts (for example, Charis SIL), but we decided that for practical purposes three is enough.

In order to avoid such situations, we recommend that the top accents go first, then the bottom accents (case 6 in Figure 7). The problem with our recommendation is that the order can be reversed by a text processing agent: according to the Unicode Standard recommendation, the bottom accent should go first and "canonical ordering behavior cannot be overridden by higher-level protocols" [11]. Some text processing agents apply the algorithm defined in [11] at the phase of reading the Unicode stream. In general, a typesetter cannot rely safely on the text processor. Even in TEX, the same text may be processed differently depending on the implementation.

In TEX, selected features, such as `ccmp`, `mark`, `mkmk`, etc., can be switched on or off on demand. Not all text processors offer such a possibility. A notable example is Microsoft Word which has these features switched on by default (it is not obvious whether it makes use of the Unicode ordering algorithm). As was mentioned, not all engines (in particular Microsoft Word, but also XeTEX) obey rules coded in the features `ccmp`, `mark`, `mkmk`. Incidentally, Figure 7 was created using LuaTEX.

In our opinion, the complexity of the implementation of anchors, resulting in a variety of approaches and implementations, is caused by the oversimplified mechanism of the OTF specification: the only allowed operations on a glyph are (re)positioning and substitution which is directly related to the OTF table structure and the basic tables, namely, `GPOS` and `GSUB`. The former operation is restricted merely to shifting, the latter to one-to-one, one-to-multiple and multiple-to-one replacements (which excludes reordering). Replacements can be either explicit or contextual, which adds complexity and does not help too much. In particular, fairly aged, not to say fossil, regular expressions are not allowed in contextual replacements.

5 Plans for the future

The next step (besides obvious cleaning of the sources, both Python and MetaPost) will undoubtedly be extending in a similar way the remaining TG text fonts, both sans-serif (Heros) and serif (Bonum, Cursor, Schola, and Termes). TG Chorus, as a chancery font, is not suitable for such an extension. We consider naming the stylistic features used in the TG fonts — it needs consideration, however; wrong names may likely introduce mess rather than order.

Having gathered experience with the text fonts, we would like to revisit the TG math fonts, with attention paid to sidebearings and math "staircase" kerns.

Moreover, we plan to remove all non-Python modules. As was mentioned, the path MetaType1 sources → OTF and Type 1 fonts is governed by Python; the reverse path, OTF and Type 1 fonts → MetaType1 sources, currently employs AWK and T1utils, thus, it cannot be used for converting TTF and OTF fonts to MetaType1 sources. We believe that the employing of FontForge (as a Python library) is the remedy.

We have no clear answer to the question of whether "small figures", accessible by features `subs` (subscripts), `sups` (superscripts), `sinf` (scientific inferiors) `numr` (numerators), and `dnom` (denominators), should be included in the text fonts; in math fonts math sub- and superscripts can be used instead. If we include these glyphs, then the next question arises: do we need special figures for small caps, `smcp`, other than, traditional in the TEX realm, old-style figures, also dubbed nautical? And do the small figures need variants commonly used for "normal" figures, that is, `lnum` (lining figures), `onum` (old-style figures), `pnum` (proportional figures), and `tnum` (tabular figures)? We are somewhat reluctant to add such a hodgepodge to an already intricate font structure.

6 Acknowledgements

We are indebted to all people and TEX groups that have supported our font enterprises. Almost all the GUST e-foundry projects were kindly supported by the Czechoslovak TEX Users Group CS TUG, the German-speaking TEX Users Group DANTE, the Polish TEX Users Group GUST, the Dutch-speaking TEX Users Group NTG, TUG India, UK-TUG, and, last but not least, TUG. In a few cases, GUTenberg, the French-speaking TEX Users Group, supported us too.

The exceptional, personal thanks we owe to our friends who have kept our spirits up for many years and tirelessly encouraged us to work on fonts: Hans Hagen, Johannes Küster, Jurek Ludwichowski, Volker RW Schaa, Jola Szelatyńska, Ulrik Vieth — hearty thanks! All trademarks belong to their respective owners and have been used here for informational purposes only.

References

[1] Adobe Systems Inc. Adobe metric files. `ftp://ftp.adobe.com/pub/adobe/type/win/all/afmfiles/base35/`

[2] Adobe Systems Inc. Feature file syntax. `adobe.com/devnet/opentype/afdko/topic_feature_file_syntax.html`

[3] GUST e-Foundry. GUST Font License. `gust.org.pl/projects/e-foundry/licenses`

[4] B. Jackowski, J. M. Nowacki, and P. Strzelczyk. TEX Gyre fonts collection. `gust.org.pl/projects/e-foundry/tex-gyre`

[5] B. Jackowski, P. Strzelczyk, and P. Pianowski. TEX Gyre math fonts collection. `gust.org.pl/projects/e-foundry/tg-math`

[6] B. Jackowski, P. Strzelczyk, and P. Pianowski. GUST e-foundry font projects. *TUGboat* 37(3):317–336, 2016. `tug.org/TUGboat/tb37-3/tb117jackowski.pdf`

[7] Microsoft Corp. OpenType Font Format, ver. 1.60, ISO/IEC 14496-22. `microsoft.com/typography/otspec160/`

[8] Microsoft Corp. Registered features. `microsoft.com/typography/otspec/featurelist.htm`

[9] Unicode Consortium. Combining diacritical marks. `unicode.org/charts/PDF/U0300.pdf`

[10] Unicode Consortium. The Unicode Standard 10.0.0; chapters 2.3 Compatibility Characters, 2.11 Combining Characters, 2.12 Equivalent Sequences and Normalization. `unicode.org/versions/Unicode10.0.0/ch02.pdf`

[11] Unicode Consortium. The Unicode Standard 10.0.0; chapter 3.11 Normalization Forms. `unicode.org/versions/Unicode10.0.0/ch03.pdf`

◇ Bogusław Jackowski
Piotr Pianowski
Piotr Strzelczyk
Rzeczypospolitej 8
80-369 Gdańsk, Poland
`b_jackowski` ,
`p.pianowski` ,
`p.strzelczyk`
`(at) gust dot org dot pl`