OpenType and Ω: Past, Present and Future

Yannis Haralambous
Département Informatique
École Nationale Supérieure des Télécommunications de Bretagne
CS 83818, 29238 Brest Cédex, France
yannis.haralambous@enst-bretagne.fr

Gábor Bella
gabor.bella@enst-bretagne.fr

Abstract

This article presents our plans for integrating the OpenType font format into the Ω typesetting system. Beginning with a short summary of what we have achieved so far, we compare potential methods of adaptation. Since most OpenType-related issues we present are valid not only for Ω but for all TEX-like systems, the authors hope that the article will be interesting for the whole TEX community.

1 Past

The OpenType font format has been officially available since 1997. In contrast to its predecessors, TrueType and PostScript Type 1 and 2, it provides essential features for proper typesetting of non-LGC[1] scripts, as well as handling LGC ones. Although competing formats with similar capabilities (Apple GX/AAT and Graphite) were and still are available, the marketing force behind OpenType seems strong enough to make it a de facto standard.

The decision to bring OpenType support to Ω as a complement to the already available fonts was taken sometime in 2002 and effective work began around early 2003. At the *EuroTEX 2003* conference, we presented our initial development plans and our first results [1]. Since then, considerable progress has been made, so that in the final version of the EuroTEX proceedings we are finally able to announce a working implementation [1].

However, this implementation is not likely to become *the* final solution, for two reasons. First, it was never intended to be such. Rather, it was created to verify the validity of certain conversion methods and also to provide users with the possibility to install and use OpenType fonts as quickly as possible, even if not all advanced features are available yet. Secondly, as we will see later, complete and robust OpenType support cannot simply be patched onto the existing Ω: as the two do not always share the same philosophy, some parts of Ω will need to be reorganized in order to take advantage of certain OpenType features.

In the following, we give a quick overview of the present solution and then explain why and how it needs to be revised.

2 Present: OpenType Support, the Quick Way

The word ‘quick’ may seem ironic here: in total, it took us almost a year to produce our first results. However, this was due more to organizational problems than to the difficulty of the task. By ‘the quick way’, we mean the solution that was the most straightforward and the easiest to implement. It is described in detail in the EuroTEX article [1]; here, only a short summary is given.

The present solution is based on the approach that OpenType fonts should be converted to Ω’s own formats, i.e., OFM (metrics), OVF (virtual fonts) and ΩTP (Ω Translation Process[2]). Anish Mehta wrote several Python scripts to generate these files, of which the most interesting is perhaps the one that converts the whole OpenType GSUB table (see next section) into ΩTP’s.

Moreover, as the OpenType format is generally not understood by PostScript printers, a conversion to Type 1 (or Type 42) is necessary. To speed up the process, we create Type 1 charstring collections using our own *PFC* tables (see [1] for details) which

[1] Latin-Greek-Cyrillic.

[2] An ΩTP describes a *finite state automaton* that can be used to filter the input text to perform tasks such as contextual analysis.

are later used by odvips to create small, subsetted Type 1 fonts (a.k.a. *minifonts*) on the fly.

The above approach assumes that OpenType can be converted to OFM and ΩTP files without significant loss of information. This is not always the case. In the next section, we will show the main differences between the two formats and why we have decided to abandon some of our results in favour of a better concept. This does not mean that the work done so far was useless: first, we managed to prove that the conversion to Type 1 (which we are planning to keep) is a viable approach and secondly, we are able to provide Ω users with a working, albeit not complete and only temporary, OpenType support.

3 Future: Alternatives of Adaptation

An OpenType font consists mainly of the following parts:

1. font and glyph metrics;
2. Type 2 or TrueType glyph outlines (and hints or instructions);
3. advanced typographic features (mainly GSUB and GPOS);
4. other data tables.

In one form or another, all of them will certainly need to be dealt with either inside Ω or odvips. Below, we give a very concise description of our plans regarding each of these fields.

3.1 Metrics

OpenType provides extensive font metric information dispersed among various tables (post, kern, hmtx, hdmx, OS/2, VORG, etc.), both for horizontal and vertical typesetting. In most cases, Ω's (or TeX's) and OpenType's metrics can be converted from one to another, with few but important exceptions (e.g., height/depth, see [1] and [2]). Despite the occasional differences, it seems desirable to dispose of the intermediary OFM files and read OpenType metrics directly from the font file.

3.2 Conversion

As explained in the last section, conversion of OpenType's Type 2 and TrueType outlines to Type 1 has already been implemented. We are also planning to provide Type 42 support for TrueType-flavoured OpenType that would also preserve instructions.

3.3 Advanced features: GSUB and GPOS

These advanced features are the most interesting part of OpenType. The GSUB (glyph substitution) and GPOS (glyph positioning) tables are essential for typesetting many non-LGC scripts. In Ω, the equivalent of GSUB features are the ΩTP's: they can do everything GSUB features can, including contextual operations. Glyph positioning is a different issue: since the ΩTPs are designed for text rearrangement (substitutions, reordering etc.), they are not suitable for doing glyph placement as easily. In fact, the idea of *microengines*—ΩTP-like Ω plugins—was introduced some time ago, exactly to provide modular, script- and language-specific positioning methods, along the lines of ΩTP files. With the appearance of OpenType fonts, it became clear that positioning features as implemented by the GPOS table and microengines provide essentially the same functionality. It should then be possible to implement microengines as GPOS features and vice versa.

A closely related problem is the fundamental difference between Ω's and OpenType's way of describing features. Although both use the Unicode encoding, OpenType's GSUB and GPOS features are based on strings of glyph ID's and not of Unicode characters. Ω and some of its ΩTP's, on the other hand, perform tasks such as contextual analysis or hyphenation on character sequences and the passage from characters to 'real' glyph ID's happens only when (o)dvips replaces virtual fonts by real ones. Moreover, it is theoretically impossible to convert a glyph-based OpenType feature into a character-based ΩTP, as some glyphs (allographs) may not even have Unicode equivalents.[3] The solution to this problem is either to use glyph- and character-based ΩTP's at the same time or else to read GSUB and GPOS features directly from the OpenType font, without conversion to ΩTP's or microengines.

Whichever method we choose, Ω will need to maintain both glyph and character representations of the same text in parallel to be able to perform font-specific (OpenType features) and font-indepen dent (hyphenation) operations at the same time. This dual representation of text is also crucial for the searchability and modifiability of the output (PDF, PS, SVG or any other) document.

3.4 Extensibility

Finally, the OpenType format has the important feature of being extensible: new tables can be added into the font file, containing, for example, data needed by Ω with no OpenType-equivalents (such as metrics or PFC charstrings). Of course, it is necessary that the given font's license allows such additions.

[3] There exist some workarounds though, for example to map these glyphs to the Private User Area of Unicode, but this solution is not very elegant, to say the least.

4 Conclusions

It is time to draw conclusions from the arguments made above. First, OpenType seems capable of being a base font format for Ω. Its tabular file structure is flexible enough so that missing Ω-specific information can easily be added to fonts. Metrics can be read directly from the font file (with some exceptions) and then converted on the fly, if necessary. Also, virtual fonts are not useful for OpenType, as it is more reasonable to use a Unicode-based encoding. Thus, neither OFM nor OVF files will be needed when using OpenType fonts. This is one of the arguments against the approach presented in the previous section.

Secondly, in terms of capabilities, ΩTP's and GSUB features are very much compatible. The same can be said for GPOS and microengines, although no implementation exists for the latter yet. However, incompatibility is present on the character/glyph level. As was shown, a dual approach can be helpful, keeping strings of glyph ID's and characters in parallel—another reason why the simple conversion method described in the previous section is not adequate.

In summary, on the one hand we have opted for a tighter integration of Ω and OpenType, by directly reading from (e.g., metrics) and writing to the font file. On the other hand, advanced OpenType features such as glyph positioning and substitution necessitate modifications in Ω's character handling as well as in the ΩTP/microengine concept. The companion article in this volume will give further details on how we are planning to implement these changes.

References

[1] Gábor Bella and Anish Mehta: *Adapting Ω to OpenType Fonts.* EuroTEX 2003 Proceedings. Final version to appear in *TUGboat*.

[2] Yannis Haralambous and John Plaice: *Ω and OpenType Fonts.* Kyōto University 21st Century COE Program, 2003.

[3] The OpenType Specification v1.4. `http://www.microsoft.com/typography/otspec/default.htm`