Using HarfBuzz as OpenType engine in LuaTEX

Introduction

When TEX was released its associated font format was the bitmap pk font format derived from TEX's companion system Metafont. Over the years TEX has been adapted to new font formats such as PostScript, TrueType, and, more recently, OpenType. With respect to this last format, two major TEX variants are currently available: XETEX and LuaTEX. Their approach to font management is quite different. XETEX uses external libraries available on the system such as AAT, HarfBuzz, and SIL Graphite. LuaTEX, on the other hand, can use any OpenType engine which is implemented in Lua and hooked into TEX's font mechanism. The ConTEXt system contains such an engine. ConTEXt's creator, Hans Hagen, has made this engine available outside ConTEXt by developing plainTEX code for LuaTEX that calls all relevant functionality of the OpenType engine.

HarfBuzz is a transliteration of the Persian word harf-bāz, meaning 'open type'. It is a free and open text shaping engine that renders texts for OpenType fonts, and recently also for other font formats. HarfBuzz is being active developed, and its use has become widespread. HarfBuzz is the preferred rendering engine for Firefox, Chrome, and XETEX. Basically, HarfBuzz converts Unicode text strings into glyph indices referring to specific glyphs in the font, and their positioning instructions.

Here I will describe how I was able to couple the HarfBuzz OpenType engine to LuaTEX.[1] First, I will describe what OpenType fonts and engines are. Next, I will give some reasons why it is interesting to enrich LuaTEX with HarfBuzz. Finally, I will disclose some technical details concerning the way in which I managed to couple HarfBuzz to LuaTEX.

OpenType fonts and engines

OpenType fonts contain glyphs that are specifications of character shapes using, for example, Bézier curves, and information describing the transformation of characters into positioned glyphs. These transformations concern either glyph positions stored in the GPOS table, or substitutions stored in the GSUB table. Glyphs are not synonymous with characters. For instance, next to the glyphs that correspond to the characters 'f' and 'i', many fonts also contain a glyph called **fi** ligature which corresponds to the combination of the characters 'f' and 'i'. In other words, there is no one-to-one correspondence between characters and glyphs. Other well-known ligatures in Latin are, for instance, the **ff**, **ffi**, **fb**, and the **ffb** ligature. OpenType engines convert texts by converting series of characters into positioned glyphs. Which glyph is chosen to represent a character depends on the font, its applied features, and the character's context. The character 'f' may be transformed into a **f** glyph, but when the font contains the glyph called **fi** ligature, and the character 'f' is followed by the character 'i', and the user has chosen to apply the ligature feature of the font, the 'fi' letter combination will be transformed into the **fi** ligature.

Why using HarfBuzz as OpenType engine in LuaTEX

The interplay between the characters and their context, the information in the font, and the features chosen by the user is far from trivial, especially for scripts like Arabic or – even more extreme – Devanagari. It is the rendering engine which has to combine all the information and return the appropriate glyphs and their positions. Although historically TEX had no shaping engine, currently several versions of TEX have been developed which have an engine at hand. For instance, XETEX is able to outsource the shaping to an external library. Interestingly, the 0.9999 release of XETEX uses HarfBuzz as its OpenType rendering engine, and at present XETEX creator Jonathan Kew is working on HarfBuzz. Behdad Esfahbod, the initiator of HarfBuzz, welcomes the association of TEX and HarfBuzz, and argues that, in the long term, 'pdfTEX's successor LuaTEX should be made to do the same thing. There is more to Unicode support than just shaping, and in those areas the TEX engines can gain a lot by building on top of existing libraries.'[2] In my opinion this remark is only partially appropriate

because it passes over the (potential) power of the ConTEXt OpenType engine currently available to LuaTEX. Although it is important that LuaTEX has a powerful rendering engine at its disposal, it is not essential that this should be an external library such as HarfBuzz. At the moment, the ConTEXt OpenType engine is able to handle many different font features and scripts, and in the rare cases in which it is not yet powerful enough, it has proven to have sufficient flexibility to be adjusted appropriately. For instance, rendering Devanagari requests eccentric competence that the engine initially lacked. However, some time ago I was able to overcome this deficit by supplementing the engine with extra code that now forms part of its core. I belief that the ConTEXt OpenType engine is satisfactory as a rendering engine for LuaTEX. Still, I have developed an approach which enables the use HarfBuzz. First, I will elaborate briefly on the ConTEXt OpenType engine, what it is and how it relates to LuaTEX, and after that I will give reasons why, in my view, it is interesting to have HarfBuzz have at one's disposal as alternative engine.

ConTEXt is a system for typesetting documents build on top of LuaTEX, which is a version of TEX with a Lua scripting engine embedded. One component of ConTEXt is its OpenType engine that is completely written in Lua. This engine is designed in such way that can also be used in LuaTEX independently from ConTEXt. To do that, one should use the 'generic' mode of the ConTEXt package. In that way OpenType fonts can be used in LuaTEX. These fonts can be applied not only by specifying their font name or filename, but also by language, script, and font features. For instance,

```
\font\f = {file: MinionPro.otf: mode=node;
           language=dflt; script=latn;
           liga=yes; kern=yes;} at 20pt
```

calls the font Minion Pro, for which the language is set to default, the script to latin, and for which several font features are set, such as the implementation of kerning and ligatures. The ConTEXt OpenType engine processes this call by building a font table in Lua that can be used during the phase of rendering the text. How the engine renders text can best be made clear by giving an example created by means of the \showotfcomposition command in ConTEXt (see Figure 1), which displays the positioning and substitutions steps executed by the rendering engine.

The example demonstrates the implementation of the font features 'kern' (kerning) and 'liga' (ligatures) while rendering the musical term 'offbeat'. Step by step the relevant operations are

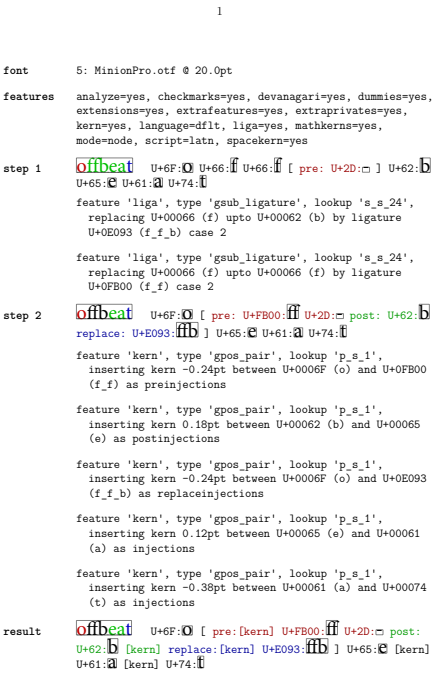

Figure 1.

applied by the OpenType engine. Interesting in this example is the interplay with discretionaries, which are constructs in which TEX stores information about the material to be displayed at hyphenation points. Discretionaries consist of three parts called pre, post, and replace. The pre contains the material to be inserted before the line break, the post contains the material to be inserted after the line break, and the replace contains the material to be inserted if the hyphenation point isn't chosen. In simple cases the pre may contain a hyphen char while the other two are empty. However, in this example, due to the hyphenation point between **off** and **beat**, it depends on the line breaking whether the **ff**-ligature or the **ffb**-ligature will be displayed. In order to have both options available, both alternatives are built into the discretionary.

The ConTEXt OpenType engine is very powerful and flexible, and can also be used in LuaTEX independently from ConTEXt. So, why do I think that it is desirable to have HarfBuzz available as alternative engine for LuaTEX, and ConTEXt too? I will give two reasons. First, because rare cases exist for which the

ConTEXt OpenType engine is not (yet) equipped to deal with, and, second, because having an alternative engine at hand is helpful for the process of developing and testing the ConTEXt engine further.

The ConTEXt OpenType engine can handle many scripts and their corresponding font features. Still, there are some scripts for which not all features are covered. As can be seen in Microsofts OpenType Specification[3] version 2, several scripts have peculiar features. Take for instance the script Syriac. Next to the regular feature 'fina' this script also has the features 'fin2' and 'fin3' at its disposal, which are quite eccentric. Whether they should be applied to replace the **alaph** glyph at the end of Syriac words with its appropriate form depends on certain properties of the preceding character. Actually, what is eccentric here is not the fact that the application of features depends on the properties of the neighbouring characters – that is quite common – but that instead of being stored into the font, this information has to be known by the OpenType engine. At this moment, as far as I know, the ConTEXt OpenType engine lacks this specific knowledge concerning the appropriate application of 'fin2' and 'fin3'. Correct typesetting of Syriac is therefore not (yet) straightforward in ConTEXt or LuaTEX. Having HarfBuzz at hand as an alternative would overcome this lack.

Based on my experience with the flexibility of the ConTEXt OpenType engine, I expect that it will not be difficult to supplement the engine with the necessary functionality to render Syriac texts. During the past year, the engine has gone through various stages of development through which it acquired increasingly complicated skills. One, rendering Devanagari, I implemented myself. So, I believe that the ConTEXt OpenType engine will eventually meet all conceivable needs. However, my experience is that, in the process of improving the engine, it is very useful to have a reference point at hand. Although the renderings offered by HarfBuzz may not be faultless or indisputable – HarfBuzz is also still in development – they are very useful as benchmark for testing the ConTEXt OpenType engine. Therefore, it is desirable to have HarfBuzz as an alternative OpenType engine in LuaTEX and ConTEXt.

Technical details concerning coupling HarfBuzz to LuaTEX

HarfBuzz is written in C++. Its functions can be accessed from outside using its application programming interface (API), which, in this case, is a software library. I will discuss two ways that LuaTEX is able to communicate with HarfBuzz via its API. The first is by means of SwigLib, which is a subproject of the LuaTEX project that concentrates on making external libraries available to LuaTEX using SWIG (Simplified Wrapper and Interface Generator). The second is by means of FFI (Foreign Function Interface), which is a method directly available when using LuaJitTEX. After discussing these two ways to call HarfBuzz from LuaTEX, I will discuss some of the technical details concerning the most important application of this technique, namely the use of HarfBuzz as OpenType engine in LuaTEX.

SwigLib can be regarded as a repository concerning the connection between application libraries and LuaTEX. For such connections an interface or binding between the application library and LuaTEX is needed. This interface can be created by means of so-called 'wrapper code' which can be generated using SWIG – a software development tool that connects programs written in C and C++ with high-level programming languages such as Lua. Recipes by means of which SWIG can produce the wrapper code for a specific application library are stored in SwigLib. In order to create the interface, the wrapper code has to be compiled. Luigi Scarso, who manages the SwigLib project, has developed the SwigLib recipe for several bindings such as those for Ghostscript and GraphicsMagick. Although he also looked at HarfBuzz, in SwigLib this binding is still in the experimental phase. (Furthermore, the material that can be found at SwigLib concerning HarfBuzz is restricted to Microsoft Windows, which happens to be not the operating system I use.) By tweaking Luigi's recipe, and also by adding the necessary code for working with specific arrays and pointers in HarfBuzz via the binding, I managed to generate a functional interface between HarfBuzz and LuaTEX.

LuaJitTEX is a version of LuaTEX that uses LuaJIT, which is a just-in-time implementation of Lua. Due to this, it can make use of the FFI of LuaJIT to access external software libraries. The use of FFI does not involve bindings or interfaces that have to be compiled. FFI allows calling external C functions and using C data structures directly from pure Lua code. In my opinion using LuaJitTEX – which for Lua-intensive TEX runs is also much faster than LuaTEX – is preferable to using LuaTEX. However, using LuaJitTEX may have some safety risks. The FFI library is a low-level library which implies it needs to be used with care. It is not safe for use by untrusted code. In my design of the LuaJIT code by means of which I bound HarfBuzz to LuaTEX via FFI I choose to use LuaJIT functions with the same name and functionality as the Lua functions that communicate with HarfBuzz via the SwigLib binding discussed above. For instance, both in LuaTEX and LuaJitTEX, I wrote a function called 'add_utf8' by

means of which a UTF-8 string can be delivered to HarfBuzz ready for rendering. This design of the functions enables the implementation of HarfBuzz, in the process of rendering, to be independent from the chosen binding method.[4]

To use HarfBuzz as LuaTEX's rendering engine, one must replace the ConTEXt OpenType engine with a procedure that translates between LuaTEX and HarfBuzz via the HarfBuzz API. In principle, processing a simple series of characters in LuaTEX in this way is rather straightforward. LuaTEX passes them to HarfBuzz as UTF-8 characters, and, in return, HarfBuzz returns information about the glyph indices, i.e., which glyphs in the font have to be picked, and how they have to be positioned. In practice, however, even the simple case of a series of characters has its issues. One issue concerns glue. LuaTEX can use glue with an arbitrary width. HarfBuzz does not know this concept. Instead, it uses the space character. When calling the HarfBuzz API, some of the glues may be regarded as spaces. This identification can, for instance, be assigned depending on the glue width. In my implementation of HarfBuzz, I choose to represent glue by one space when its width is greater than zero. Similarly, the ConTEXt OpenType engine also represents such glue as a space in its OpenType rules that involve spaces. Another issue concerns attributes. In LuaTEX, characters can be enriched with information by means of so-called attributes. This is very useful, for instance to implement colour handling. In the process of rendering, these attributes have to be preserved. However, due to the formation of ligatures and other constructs several characters can turn into just one glyph. Conversely, it is also possible that one character turns into more than one glyph. Therefore, even for a simple series of characters, some bookkeeping is required to ensure that character attributes are assigned to the corresponding glyphs.

A more complicated issue concerns the interplay between discretionaries and OpenType transformations – such as the formation of ligatures. As illustrated above (see Figure 1), the word 'offbeat' has a hyphenation point between 'off' and 'beat'. Because the concrete hyphenating is not established during the stage of implementing the OpenType transformations – that happens only in the stage of line breaking which comes later – the **ff** ligature and **ffb** ligature have to be incorporated into the discretionary. Ideally, this operation of incorporating glyphs into discretionaries would be left to HarfBuzz. LuaTEX would then offer HarfBuzz something like

```
off\discretionary{-}{}{}beat
```

as input and receive something like

```
o\discretionary{ff-}{b}{ffb}eat
```

plus information about kerning in return. However, HarfBuzz has no syntax for hyphenation. Therefore, it is not possible to send and receive material that contains discretionaries. To overcome this problem, I developed a routine that rephrases material containing hyphenation points into parts free of hyphenation points covering the situations with and without hyphenations. This is not too complicated for a series of characters with only one hyphenation point. For 'offbeat' the parts would consist, on the one hand, of 'off-' and 'beat', and, on the other, 'offbeat'. However, for series of characters with more than one hyphenation point, the number of parts that cover all hyphenation possibilities can become rather large. The general idea is that all these parts are presented to HarfBuzz and the result is subdivided back into discretionaries. The rationale behind this approach is that the OpenType transformations applied to characters can depend on the (series of) neighbouring characters. Because these (series of) neighbouring characters vary depending on the pending hyphenation, HarfBuzz has to be presented with all the possibilities. In the example, HarfBuzz returns **off-**, **beat**, and **offbeat** (plus information about kerning) which is rephrased in LuaTEX as

```
\discretionary{off-}{beat}{offbeat}
```

After that, a process is executed to clear out the discretionaries. As much material as possible is taken out from the discretionary. This applies, for instance, to the **o** in the example, which is present at the start of the first and second argument of the discretionary, but which can be moved to the left of it. The same holds for **eat** in the second and third argument of the same discretionary, which can be moved to the right of it. Clearing out the discretionary might not be absolutely necessary in this example: LuaTEX can handle series of characters that are completely wrapped up in discretionaries. However, in cases of more than one hyphenation point per series of characters, clearing serves a useful purpose for it can prevent the occurrence of nested discretionaries that would otherwise have to be eliminated. The result of this process is very much comparable to the result that would have been obtained from using the ConTEXt OpenType engine such as displayed at the bottom of the 'offbeat' example above (see Figure 1).

Results and conclusion

The approach I followed to implement HarfBuzz in LuaTEX has yielded positive results. HarfBuzz performs well. The similarity between texts rendered by the ConTEXt Opentype engine and HarfBuzz surpassed my initial expectations, even for tests with complicated scripts and sophisticated fonts. Although most tests led to exactly the same results, I found a few scattered instances in which the outcome differed. For some of these differences the ConTEXt OpenType engine could be hold accountable, such as the difference in shape of the **alaph** glyph at the end of some Syrian words discussed above. However, in other cases HarfBuzz clearly dropped a stitch.

It is desirable to have HarfBuzz available as alternative engine for LuaTEX. This would enable rendering texts with features the ConTEXt OpenType engine currently does not support. More importantly, it would benefit the process of developing and testing the ConTEXt engine further. This is indeed the case, as demonstrated by improvements made to the engine based on my reports of tiny faults discovered while comparing the results of the two engines. The ConTEXt OpenType engine has proven to be a powerful and flexible engine for LuaTEX in its own right. However, it will benefit from having HarfBuzz available next to it.

Notes

1 For source code and examples, see https://github.com/tatzetwerk/luatex-harfbuzz.
2 http://behdad.org/text/ (consulted: September 2016)
3 https://www.microsoft.com/en-us/Typography/SpecificationsOverview.aspx (consulted: September 2016)
4 Moreover, my design of these functions is adopted from a binding called luaharfbuzz (see https://github.com/deepakjois/luaharfbuzz, consulted: September 2016). This is binding is similar to the binding developed via SwigLib although the wrapper code is written by Deepak Jois instead of generated by means of SWIG. Consequently, next to the SwigLib and FFI binding also luaharfbuzz can be used as binding that enables my implementation of HarfBuzz as OpenType engine in LuaTEX.

Kai Eigner, TAT Zetwerk, Nederland, eigner@tatzetwerk.nl