Typesetting Bangla script with LuaLaTeX

Ulrike Fischer, Marcel Krüger

In [2], Md Qutub Uddin Sajib describes some experiences and insights concerning typesetting Bangla.

While we don't know Bangla, we want to share some additional information about the LuaLaTeX side, especially about how to use the possibilities of the new `HarfBuzz` library.

1 `HarfBuzz` in LuaLaTeX — the engine choice

Typesetting a script is more than placing glyphs side by side. Even in the rather simple western scripts there are already ligatures, kerning and accents to handle and many scripts have much more complex rules. Up to now LuaLaTeX wasn't very good with such scripts. This wasn't due to a fundamental deficiency, but a lack of manpower: To implement the shaping rules one needs people knowing the script, knowing Lua and having the time and the will to put both together. The experience with `xetex` was much better as this engine relied on an external library from the start — these days on `HarfBuzz` [3]. For quite some time there has been a wish for `luatex` to also support use of `HarfBuzz`.

In 2019, this became possible: with `harftex` [1] and `luahbtex`, two engines with built-in `HarfBuzz` support were available. After some discussion the LaTeX team decided to base LuaLaTeX in TeX Live 2020 on `luahbtex` and integrated in fall 2019 the necessary Lua code into `luaotfload`. As an engine change is a major step the two TeX distributions TeX Live and MiKTeX added the new engine already in November 2019 and the LuaLaTeX-dev format has been mapped to it. This allowed beginning "real world" tests.

So, from the various LuaLaTeX variants mentioned by Sajib, only the following should be considered if `HarfBuzz` is wanted:

TeX Live 2019 / before April 2020

`lualatex-dev`	= `luahbtex`	+ LaTeX-dev

TeX Live 2020 / after April 2020

`lualatex`	= `luahbtex`	+ LaTeX
`lualatex-dev`	= `luahbtex`	+ LaTeX-dev

To make the best use of the new `HarfBuzz` integration it is important to keep one's TeX distribution, notably including `luaotfload`, up to date to benefit from the development and corrections of bugs.

2 Using `HarfBuzz` in LuaLaTeX

With XeLaTeX `HarfBuzz` is always used to shape a font (with the exception of legacy TeX fonts). This doesn't hold for LuaLaTeX. Here one can choose on a font-by-font basis between the `base mode` (mostly used for math fonts), `node mode` (used for text) and the new `harf mode` (which has a number of submodes). `HarfBuzz` is an *addition* to, not a replacement for, the existing font shapers. If the library isn't used the new engine behaves like `luatex`.

Table 1: Example rendering with the various modes

`base`	কণ্‌যা এখন ক ি কর ি ব ে?
`node`	কণ্‌যা এখন কি করিবে?
`harf`	কণ্যা এখন কি করিবে?

Table 1 shows the rendering of an example text in the three modes. In comparing the output one can see — even if one doesn't understand the script — that `base mode` doesn't know much, `node mode` a bit and `harf mode` a lot about the script. E.g., the third word is `U+0995 U+09BF` (corresponding to the Bengali letters "KA": ক and "I": ি). While the simpler `base mode` just prints these letters next to each other, the more advanced modes `node` and `harf` reverse the order. The first word in the examples shows that `node mode` is still missing some shaping rules that `harf mode` applies.

When using the `fontspec` package the mode can be chosen with the `Renderer` option. It is also important to specify the correct script. The `harf mode` in table 1, for example, can be done with this font declaration:

```
\setmainfont{Noto Sans Bengali}
   [Renderer=HarfBuzz,Script=Bengali]
```

3 The "dotted circle" mystery

In [2], Sajib also discusses how to typeset a glyph without getting an unwanted dotted circle as a base character. With XeLaTeX it isn't easy to overrule the rendering that the `HarfBuzz` library considers to be correct (though `\XeTeXglyph` can be used). But in LuaLaTeX it can be done by switching to another mode of typesetting. Compare the output of e.g. `\char"90BE`:

`harf mode:` ◌া   `base/node mode:` া

4 Coloring glyphs

The standard LaTeX color commands insert specials or literals. This interrupts the input and so can disturb the font shaping. Figure 1 shows an example where the color commands inhibit the reordering of the glyphs.

With LuaLaTeX one can use attributes instead by loading the `luacolor` package. With it, the code in figure 1 gives this output: কি কি

At first glance this looks okay but the coloring is actually odd. The code asks to color the KA red

```
\char"0995\char"09BF \quad
\color{red}\char"0995
\color{green}\char"09BF
```

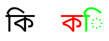

Figure 1: Wrong shaping due to color commands

(ক) and the I green (ি) but in the output the colors are reversed. This shows a general problem with this approach: the attributes are attached to the *input* chars. When getting back the shaped output from `HarfBuzz` (in this case in reversed order) `luaotfload` doesn't always know how the input and the output relate and has to guess how to reattach the attributes. Switching to `node mode` solves the problem for this specific case as in this mode `luaotfload` has full control over the shaping and so can make a better decision how to set the color.

But the core of the problem lies deeper: font shaping takes a cluster of n input chars and outputs m glyphs and there is not always a clear mapping between attributes of an input char to the attributes of the output glyphs. As an example, take the input `\color{red}f\color{green}f\color{blue}i`: what color should the output ffi-ligature glyph be?

5 Coloring the output glyphs

We've seen that adding color commands to the input does not always lead to satisfactory results. What about color instructions that target the *output* glyphs? The code in figure 2 (which requires `luaotfload 3.12`) shows that this is possible. The code defines a color scheme that maps colors to output glyphs and uses it in the font declaration.

The difficulty with this method is to correctly reference the desired output glyphs. The code shows two different methods: by name ("ivowelsignbeng") and by index, the GID, of the glyph. Both methods have drawbacks.

Glyph names can fail as not every font uses the same names for the same glyphs, and some fonts don't contain them at all. The name "`t`" for example works fine with TEX Gyre Heros but not with Arial. The handling of glyph names also differs between the modes: With the `HarfBuzz` renderer glyph names currently work only with `ttf` fonts (this will probably change with the next `HarfBuzz` version).

Index numbers are more reliable but they are different for every font (and can change if a font is updated). The GID of "`t`" is 87 in Arial and 106 in TEX Gyre Heros.

A third possibility, more difficult to implement, is to use the ToUnicode mapping of the glyph. The problems here are several: different glyphs can have the same ToUnicode, some glyphs (e.g., accented

```
\directlua{
  luaotfload.add_colorscheme("my_scheme",
  { ["FF0000"] = {"kabeng"},
    ["00FF00"] = {"ivowelsignbeng"},
    ["0000FF"] = {369} % GID of "nadarabeng"
  })}
\newfontface\colorbengali{Noto Sans Bengali}
  [Renderer=Harfbuzz,
   Script=Bengali,
   RawFeature={color=my_scheme}]

{\colorbengali
\char"0995 \char"09BF
\char"09A8 \char"09CD \char"09A6
\char"09CD \char"09B0}
```

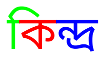

Figure 2: Coloring glyphs in a font

characters) can have more than one ToUnicode value (and which one is used in a document cannot always be easily predicted), some glyphs are clusters and so have quite long values, and finally, some have no ToUnicode mapping at all.

So while the results of coloring output glyphs can be quite good, this clearly requires some skill and the right fonts.

6 Coloring parts of a glyph

The methods mentioned above don't help to color the *parts* of a single glyph, e.g., ন্দ্র. While the *input* consists of five chars (ন্দ্র) the *output* is *one* glyph in the font and coloring parts of a glyph can only be done if the font has special support for it. Here the only option is to enhance the font with color support or to draw the glyph with Ti*k*Z or some graphic application and color it manually.

References

[1] K. Hosny. Bringing world scripts to LuaTEX: The HarfBuzz experiment. *TUGboat* 40(1):38–43, 2019. `https://tug.org/TUGboat/tb40-1/tb124hosny-harfbuzz.pdf`

[2] M. Q. U. Sajib. Typesetting the Bangla script in Unicode TEX engines — experiences and insights. *TUGboat* 40(3):263–269, 2019. `https://tug.org/TUGboat/tb40-3/tb126sajib-bangla.pdf`

[3] The FreeType Project. HarfBuzz, a text shaping library. `https://harfbuzz.org`

⋄ Ulrike Fischer
Mönchengladbach
`ulrike.fischer (at) latex-project.org`

⋄ Marcel Krüger
Hamburg
`marcel.krueger (at) latex-project.org`