Lost in Fonts

Hans Hagen

Nowadays, it is rather common to use a (wide font) OpenType-aware engine but for quite a while, the eight-bit fonts dominated the TeX landscape. The pdfTeX engine could actually use wide TrueType fonts, but that was more of a hack: only an eight-bit subset could be used. This trick permitted for instance using large cjk fonts in TrueType format by splitting the metrics up in 256 character chunks. Traces of that approach can be found in TeX distributions where one such font comes with dozens of tfm files.

In order for TeX to do its work, font metrics are needed and these come, in the case of pdfTeX, from tfm files. These files contain metric information (width, height, depth and italic correction), recipes for ligatures (multiple glyphs replaced by one), and inter-character kerning data. The backend (build-in or external) will eventually filter the shapes from a font resource by index. Such a resource can be a pk file (direct index mapping), a Type1 (via an encoding vector) or a ttf file (also via an encoding vector).

A binary tfm file is either handcrafted or the product of a conversion, for instance by `afmtotfm` where the human readable afm file, which can come with a TrueType font, describes the font. Because TeX can handle ligatures, it is no surprise that when that information is available it will end up in the tfm file. The converter sees glyphs with names like `f`, `i` and `fi` and can add the information to the tfm file that an **f** followed by an **i** becomes **fi**.

Now, when we move on to a modern OpenType-aware engine, we no longer need the tfm file but instead we load the ttf file directly; and here is where we can be surprised. The following example is in ConTeXt speak, but the principles remain:

```
\definefont [MyFont] [file:TYFATECE.TTF*default @ 45pt]
```

Here we define a font and apply the default feature set that sets up the font to provide ligatures (the `liga` feature) as well as kerns (the `kern` feature).

```
\MyFont efficient fietsen
```

efficient fietsen

When Jano Kula was working on a schedule for a film festival, he wanted to use the font (based on a design by Josef Týfa) shown here, so when no ligatures showed up he was puzzled since it had worked before. That made me wonder what happened.

The font is two decades old and has not changed. But what did change was the engine being used. In LuaTEX (and LuaMetaTEX) we directly load the ttf file but that particular loaded file has no features! This can be seen in the so called `tma` file in the font cache, a human readable Lua file. This is because at the time that font was made, there were no features. The ttf file was just an alternative for a Type1 file and features had to be derived from the afm file.

So, how do we deal with this? Well, the solution is similar to what these aforementioned converter commands do. First we define an additional feature, then we use it.

```
\startluacode
fonts.handlers.otf.addfeature {
    name = "myliga",
    type = "ligature",
    data = {
        ["fi"]  = { "f", "i" },
        ["fl"]  = { "f", "l" },
        ["ffi"] = { "f", "f", "i" },
        ["ffl"] = { "f", "f", "l" },
        ["fff"] = { "f", "f", "f" },
    }
}
\stopluacode

\definefontfeature[myliga][default][myliga=yes]

\definefont [MyFontLiga] [file:TYFATECE.TTF*myliga @ 45pt]
```

```
\MyFontLiga efficient fietsen
```

For the sake of this summary we used `myliga` but one can also just use `liga` as name before the first font is loaded. However, by being explicit we know what gets applied. Here is the result:

efficient fietsen

The main reason for bringing this up is that a migration to newer technology can result in an (initial) unexpected loss of functionality. But fortunately in this case there was a way out. However, one can wonder if the loss of the ligatures was really that bad here. Just for the record, the following approach simulates the way TEX does it: pair-wise ligature building.

```
\startluacode
fonts.handlers.otf.addfeature {
    name   = "myliga1",
    type   = "ligature",
    data = {
        ["fi"]  = { "f", "i" },
    }
}
fonts.handlers.otf.addfeature {
    name   = "myliga2",
    type   = "ligature",
    data = {
        ["ffi"] = { "f", "fi" },
    }
}
\stopluacode

\definefontfeature[myliga][default][myliga1=yes,myliga2=yes]
```

But the earlier solution where we just put all lookups in one feature is of course more efficient. Other solutions can use so called contextual lookups but that would be an overkill here.

This article was first published in NTG's MAPS No.51.

Photo: Harald König