OPENTYPE FONTS

the generic loader

Hans Hagen – bachoTEX 2016

how engines sees a font

TEX

fields: width, height, depth, italic correction, kern table, ligature tree, vf commands, next size pointer, extensible specification **and** a set of text and math parameters

pdfTEX

extra fields: left protruding, right protruding, expansion factor **and** parameters to control these

LuaTEX

extra fields: math top accent, math bot accent, tounicode, adapted extensible specification, vertical variants, horizontal variants, name, index, used status, math kerns **and** extra parameters **and** math constants **and** no 8 bit limitations

XETEX

probably something similar

font handling

loading opentype font data

- till recently we used the built-in fontforge loader library
- but now we use a recently written Lua loader
- but use a similar feature handler
- in ConTEXt one can fall back to the old loader/handler

applying (opentype) features

generic modes: base, node
ConTEXt modes: base, node, auto, dynamic

locating (opentype) fonts

- **file**: kpse in generic, resolvers in ConTEXt
- **name**: simple in generic, extended in ConTEXt, different in LATEX
- **spec**: not in generic (uses font database)
- **virtual**: not in generic
- **lua**: delegated to low level interfaces

OP

preparations

after loading

- initialize format driven substitution
- initialize format driven positioning
- enable analysis of states/properties
- initialize additional data for engine (protrusion, expansion, extend, slant)
- apply user or TEX format extensions
- apply manipulations before and after loading
- (build virtual fonts)
- enable special script handlers (fuzzy side of opentype)
- pass metrics and some metadata to TEX

benefit

efficient access to all font properties for additional processing beforehand or afterwards

OPE

processing

steps

- (comes after hyphenation)
- first identifies to be handled modes
- normalization (in ConTEXt) node list
- delegate handling to TEX or Lua
- when using Lua features are applied in prescribed order: substitution, positioning, etc.
- as last step positioning is finalized (left/right kern injection, space kerning, anchoring, cursives)

remarks

- efficient contextual analysis is-non trivial
- discretionaries need special care: ...pre ...replace... post...
- there is no real limit in extensions
- it's not too hard to inject experimental code
- so users can add their own features
- some day there may be alternative handlers

OPEN

math

format

the opentype math specification stays close to TEX, but has extensions and more control (see articles & presentations by Ulrik Vieth)

loading

- maps more or less directly onto internal structures
- in ConTEXt we use(d) virtual unicode fonts awaiting lm/gyre

processing

character mapping and special element handling remains macro package dependent

construction

- we split code paths when needed: traditional or opentype (no longer heuristics)
- the LuaTEX engine provides much control over spacing and a bit more over rendering

OPENT

the basics of loading

the format

- it evolved out of competing formats by apple, microsoft and adobe
- two flavours can normally be recognized by suffix: `ttf` and `otf`
- main differences are bounding box info, global kern tables, cubic vs quadratic curves
- multiple sub fonts inside `ttc` files (font collections)
- it's considered a standard (so it should be possible to implement)

the specification

- the only useable reference is on the microsoft website
- (the iso mpeg standard is more or less a bunch of ugly rendered webpages)
- trial and error helps understanding/identifying fuzzy aspects

OPENTY

the available loaders

the fontforge loader

- offers the same view on the font as the editor (good for debugging)
- in order to process a font some optimal data structures are created after loading
- we cache fonts because loading and creating these structures takes time and it saves memory too
- fontforge has a lot of heuristics (catching issues collected over time) but these are hard to get rid of when they're wrong

the lua loader

- this started out as experiment for loading outlines in MetaFun
- it avoids the conversion to optimal structures for handling
- we can hook in better heuristics (data is more raw)
- it fits in the wish for maximum flexibility (next stage ConTEXt)
- it's rather trivial to extend and adapt without hard coding
- the performance can be a bit less on initial loading (pre-cache) but there is a bit of room to improve
- it's much more efficient in identifying fonts (not a real issue in practice)
- in practice most fonts behave ok (no recovery needed) but there are some sloppy fonts around

what do we load

tables

- opentype is mostly tables with lots of subtables
- there are required, truetype outline, postscript outline, (svg and bitmap), typography & additional ones
- the typographic tables specify transformations to apply (gdef, gsub, gpos)

calculations

- as we need ht/dp we need to calculate the boundingbox of postscript outlines (cff parser)
- internally we use unicodes instead of indices
- we need to identify/filter the right unicode information
- we want to do more so we need to carry around more info (tounicode etc)

pitfalls

- there is no real consistent approach to use of basic features: single, one to multiple, multiple to one & many to many replacements, and look ahead and/or back based solutions
- in principle consistent families like lm/gyre could share common data and logic but otherwise there is much diversity around

OPENTYPE

a few details

loading

- load the file (subfont if needed) in a Lua friendly format
- prepare for later processing and/or access
- optimize data structures
- cache the instance (and compile to bytecode)
- share loaded font data where possible
- initialize & mark enabled features
- pass metrics, parameters and some properties to TEX

processing

- we need to run over enabled features (also virtual non-opentype ones)
- we use lookup hashes to determine if action is needed
- if needed we access detailed data and apply it
- there can be a few but also many hundreds of loops over the node list
- contextual matching can make us end up with a real lot of access and analysis
- descending into discretionaries adds significant overhead (so it's optimized)

OPENTYPE F

traditional fonts

tfm

- there is a built-in loader for `tfm`, `ofm`, `vf` and `ovf` files
- encoding and filename mapping is as usual (`enc` and `map` files)
- (in the early days ConTEXt filtered info from those `enc` files too)

type one

- type one fonts have their own loader that gets information from `afm` files
- the `pfb` file is consulted to get the index (to unicode) mapping
- the `afm` loader was already written in Lua but we now can also use Lua for the `pfb` file

OPENTYPE FO

remarks

- features like additional character kerning don't belong in the font handler as they are (to some extent) macro package dependant
- the same is true for italic correction (often input related and therefore a macro package specific issue)
- setting up protrusion and expansion is again somewhat macro package dependent
- ConTEXt has many extra font related mechanisms and features (described in a more technical manual)

- this has to work well with the core subsystems: languages especially hyphenators, specific script demands, typesetting (all kind), builders (paragraph, page), etc.
- a complication is that we do this more and more in Lua, but still need to support the built-in mechanismsm too

- the interfacing to macro packages differs (for plain TEX we use code that ships with ConTEXt)
- for bugs and issues of with fonts in ConTEXt you use its mailing list (or mail me)
- the LATEX interface is handled by Philipp Gesang

OPENTYPE FON

future

- we'll improve handling of border cases (within the constraints of performance)
- we might provide a few more hooks for plug-ins
- the type one `pfb` reader will be extended to provide outlines (not complex, needed for MetaFun)
- we keep playing with extra new features and virtual fonts

- maybe some more code can be made generic (fwiw)

OPENTYPE FONT

credits

- Kai Eigner and Ivo Geradts for (experimental) patches in the handlers for rare, complex & creepy fonts
- Philipp Gesang for binding the generic code to LaTeX font mechanims.
- Idris Samawi Hamid for testing and providing the very complex and demanding Husayni font
- Hartmut Henkel for the initial cleaning up of expansion and protrusion
- Taco Hoekwater for the original loader and discussions and a lot more
- Boguslaw Jackowski and friends for the fonts and patience with us
- Dohyun Kim for testing and suggestions on CJK font support
- Mojca Miklavec for distributions, managing us, and basically everything
- Luigi Scarso for patiently testing and managing my patches and testing very beta code
- Thomas Schmitz for using betas in deadline critital book production and making sure we patch fast
- Ton Otten for permitting me to work on all this TeX related stuff for ever and ever (and using to the extreme)
- Wolfgang Schuster for knowing and testing every detail of ConTeXt and writing selectfont (for system fonts)

- and all (ConTeXt) users who patiently accept betas and testing

OPENTYPE FONTS