compact fonts

what are the advantages

context **2024** meeting

Before we had MkII

- It all started with rather plain `\font` definitions.
- More than just fonts need to be ‘switched’.
- So body font switching is wrapped into macros.
- Often in MkII more fonts get loaded than are needed.
- This comes cheap when using eight bit fonts.
- Design sizes complicate matters.

The MkII font model

- Eight bit fonts have a limited coverage.
- Hyphenation relates to font encoding.
- We need to handle font and input encodings.
- Using small caps and/or old style numerals demand a different font.
- This resulted in a multi-dimensional system.
- Design sizes have been complemented by a simpler model.
- At some point we had to support XeTeX, so support for features was introduced.
- Loading fonts is delayed when possible so that we can mix with little overhead.

The MkIV font model

- Font loading is delegated to Lua, we could not support Oriental TEX otherwise.
- Dealing with font features is also up to Lua.
- More dynamic par building experiments demanded interplay with fonts.
- Fonts are often large so there is more aggressive sharing and caching.
- Runtime support for virtual fonts is integrated.
- Way more trickery is possible because we have full access.
- Users can tweak and extend fonts as they wish (given available glyphs).
- Features (like small caps) can be applied dynamically.
- Variable and color fonts were supported as soon as they showed up.

The LMTX font model

- We assume LuaMetaTEX to be used.
- We have better control over how the backend deals with fonts. This was prototyped in MkIV but later removed.
- To a large extend the model used is the same.
- We have a bit more virtual font magic available.
- Tweaking math fonts has been extended and is also applied.
- Of course we also have expansion but we can change that on the spot.

Some new engine features

- Math fonts are demanding and are ‘loaded’ three times per size (three families) which means three times tweaking.
- For that reason compact math fonts were introduced: load once and select (ssty) and scale (script and scriptscript) on the fly.
- That meant that some additional scaling parameters had to be introduced.
- Which in turn triggered dynamic scaling in text mode.

Some new engine features

- The engine supports `\glyphscale`, `\glyphxscale`, `\glyphyscale`, `\glyphslant` and `\glyphweight`.
- There are also `\Umathxscale` and `\Umathyscale` (per math style).
- These properties ar bound to glyphs which means that dimensions (when needed) are calculated on the fly.
- Specific font (and other) glyph related features can be controlled locally: left/right kerning and ligaturing, expansion, protrusion, italic correction etc.
- A new primitive `\fontspecdef` can efficiently change the current combination of properties.

Intermezzo: glyph nodes

- In TEX they only contain font and character fields (in addition to the common `type`, `subtype` and `next` fields.
- In LuaTEX they are larger and of course also have the new common `prev` and `attr` fields plus two SyncTEX fields.
- In LuaMetaTEX glyph nodes are among the largest nodes, currently 14 times 8 bytes.
- There are 4 byte fields: `font`, `data`, `state`, `options`, `hyphenate`, `expansion`, `x_scale`, `y_scale`, `scale`, `raise`, `left`, `right`, `x_offset`, `y_offset`, `weight`, `slant` and `index` (math).
- There are 2 byte fields: `language`, `control`, `properties` (math) and `group` (math) and a few 1 byte fields: `protected`, `lhmin`, `rhmin` and `discpart`.

Intermezzo: font spec nodes

- The ‘spec’ in the `\fontspecdef` indicates a similarity with so called ‘glue spec’, as they also use so called nodes as storage container.
- Of course such a font switch is a bit more costly than a regular `\font` switch.
- There are some related query commands: `fontspecid`, `fontspecified-size`, `fontspecscale`, `fontspecxscale`, `fontspecyscale`, `fontspecslant` and `fontspecweight`.
- It is currently a 5 memory word node (5 times 8 bytes) with 4 byte fields: `state`, `identifier`, `scale`, `x_scale`, `y_scale`, `slant` and `weight`.

Compact mode

- Compact font mode is enabled at the top of the document (before fonts get defined):

```
\enableexperiments[fonts.compact]
```

- Often performance is the same, but for large fonts there is a gain. The same is true for math fonts.
- The produced pdf code can (!) be more efficient which compensates the larger overhead.
- The question is: will we make this default which means that we need a directive that enables traditional mode.

Compact mode

The print version of “Math in ConTEXt” currently has 290 pages.

	run time	file size
normal	13.6	2.457.962
compact	10.6	2.456.630

105 font files loaded (see next page)

	instances	backend	vectors	hashes	load time
normal	317	217	76	141	5.0
compact	110	43	41	2	1.9

105 font files loaded:

koeielettersot.ttf, lucidabrightmathot.otf, lucidabrightot.otf, lucidasanstypewriterot.otf,
latinmodernmath-companion.otf, ralphsmithsformalscript-companion.otf, texgyrebonummath-companion.otf,
texgyrepagellamath-companion.otf, texgyretermesmath-companion.otf, concrete-math.otf, ebgaramond-regular.otf,
garamond-math.otf, erewhon-math.otf, erewhon-regular.otf, euler-math.otf, kpmath-bold.otf, kpmath-regular.otf,
kpmono-regular.otf, kproman-regular.otf, libertinusmath-regular.otf, libertinusmono-regular.otf,
libertinusserif-regular.otf, cambria.ttc, xcharter-math.otf, xcharter-roman.otf, iwona-regular.otf,
iwonalight-regular.otf, kurier-regular.otf, kurierlight-regular.otf, antykwatorunska-bold.otf,
antykwatorunska-italic.otf, antykwatorunska-regular.otf, antykwatorunskacond-regular.otf,
antykwatorunskalight-regular.otf, latinmodern-math.otf, lmmono10-regular.otf, lmmonoltcond10-regular.otf,
lmmonoproplt10-regular.otf, lmroman10-regular.otf, texgyrebonum-math.otf, texgyredejavu-math.otf,
texgyrepagella-math.otf, texgyreschola-math.otf, texgyretermes-math.otf, texgyrebonum-bold.otf,
texgyrebonum-italic.otf, texgyrebonum-regular.otf, texgyrepagella-bold.otf, texgyrepagella-bolditalic.otf,
texgyrepagella-italic.otf, texgyrepagella-regular.otf, texgyreschola-regular.otf, texgyretermes-regular.otf,
ex-iwonal.tfm, ex-iwonam.tfm, ex-iwonar.tfm, mi-iwonabi.tfm, mi-iwonali.tfm, mi-iwonami.tfm, mi-iwonari.tfm,
rm-iwonab.tfm, rm-iwonal.tfm, rm-iwonam.tfm, rm-iwonar.tfm, sy-iwonalz.tfm, sy-iwonamz.tfm, sy-iwonarz.tfm,
ex-kurierl.tfm, ex-kurierm.tfm, ex-kurierr.tfm, mi-kurierhi.tfm, mi-kurierli.tfm, mi-kuriermi.tfm, mi-kurierri.tfm,
rm-kurierh.tfm, rm-kurierl.tfm, rm-kurierm.tfm, rm-kurierr.tfm, sy-kurierlz.tfm, sy-kuriermz.tfm, sy-kurierrz.tfm,
ex-anttcr.tfm, ex-anttl.tfm, ex-anttr.tfm, mi-anttbi.tfm, mi-anttcbi.tfm, mi-anttcri.tfm, mi-anttli.tfm,
mi-anttri.tfm, rm-anttb.tfm, rm-anttcb.tfm, rm-anttcr.tfm, rm-anttl.tfm, rm-anttr.tfm, sy-anttcrz.tfm,
sy-anttlz.tfm, sy-anttrz.tfm, dejavusans-bold.ttf, dejavusans.ttf, dejavusansmono-bold.ttf,
dejavusansmono-oblique.ttf, dejavusansmono.ttf, dejavuserif.ttf, stixtwomath-regular.ttf, stixtwotext-regular.ttf

Summary

- Compact font mode is the future, but it only works with LuaMetaTEX and ConTEXt LMTX.
- The engine has to work harder, but the extra overhead can be neglected.
- Larger fonts have less impact.
- Using many fonts also has less impact.
- In math it is now the default anyway.
- We have larger nodes but the increase in memory usage is compensated by less fonts.
- One has to use dimension related helpers in Lua (they do the scaling).
- The backend is more complex with respect to fonts so that compensates the performance we gain on regular documents.