OpenType Math

In ConTEXt

Dante 2010

Traditional approach

There are several reasons why the font implementation in traditional TEX (as used by CONTEXT MKII) is somewhat complex.

- The traditional TEX fonts come in design sizes.
- Math fonts are defined in triplets (3 sizes) called families.
- Relative sizes depend on the bodyfont size and mixed typefaces have additional relative sizing.
- Fonts have at most 256 characters so we end up with multiple fonts.
- Math fonts (as text fonts) can have different encodings and we need to be able to mix them.
- Some symbols have no real glyph but are constructed by macros.
- Math has always to be set up in advance (while text font definitions can be delayed).

But . . . UNICODE and OPENTYPE can make things easier (if we forget about a few annoyances).

Math In MKIV

In parallel with the development of LUATEX we adapted the MKIV math font mechanism in rather non-standard way:

- We only use one math family which simplifies the definition and speeds up initialization.
- We removed (replace) all artifacts that result from limitations in Computer Modern.
- We no longer support 8 bit fonts simply because we no longer need them.
- We use Cambria Math as benchmark for supporting traditional math fonts.
- Switching between alphabets is no longer a family matter but implemented using attributes.
- Dirty spacing tricks are dealt with by noad processing instead of active character manipulations.
- We initialize math characters using the MKIV character database.

Complications

Bold math (that is: a full switch to bold) is not yet supported simply because we don't have the proper fonts for doing it.

Gaps in UNICODE math vectors have been annoyingly delaying development (it just takes time to figure out things).

Changes in UNICODE result in fuzzy situations (especially with respect to some greek characters) as as there can be applications around that still use them.

Creating virtual glyphs for symbols that are traditionally made by macros (i.e. lack in the AMS symbol set or are not in the math extensions fonts due to lack of room) takes much time.

Some UNICODE alphabets have holes which forces us (like any application) to cook up workarounds.

The TEX community is not really involved in those new technologies (for instance we lag behind in font technology).

Side Effects

We gained some more insight in the reason why things were implemented in rather special ways in TEX (font limitations).

We learned a few more dirty tricks. It was fun to reverse engineer some of Don’s macros.

We could go ahead in spite of the Latin Modern and Gyre math fonts not being available (as we had Cambria).

We were forced to add quite some tracing options and visualizations.

But eventually we ended up with a cleaner, faster and better system and in the process got rid of math encodings, no longer have macro based characters, and could trash much code at the TEX end.

We had to adapt (and probably have to do it a few more times) the new mechanisms when we finally figured out what is intended.

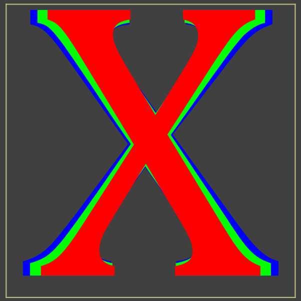
A nice testcase is the experimental Euler OPENTYPE font. It is good to have a more TEX related reference font alongside Cambria for testing.

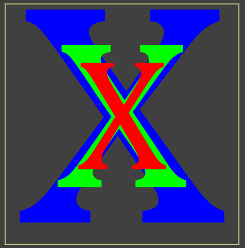
Features and Sizes

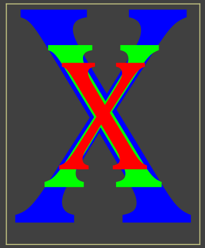
text	`lmmi12 at 12pt`	`cambria at 12pt with ssty=no`
script	`lmmi8 at 8pt`	`cambria at 8pt with ssty=1`
scriptscript	`lmmi6 at 6pt`	`cambria at 6pt with ssty=2`

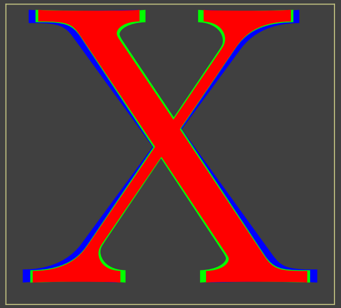
```
\definefontfeature[math][analyze=false,script=math,language=dflt]

\definefontfeature[text]        [math][ssty=no]
\definefontfeature[script]      [math][ssty=1]
\definefontfeature[scriptscript][math][ssty=2]
```

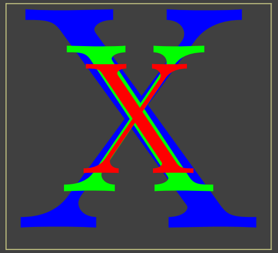
Subtle Differences


cambria
150pt (ssty)


cambria
120pt 80pt
60pt (ssty)


cambria
120pt
80pt 60pt


lm roman
150pt


lm roman
120pt
80pt 60pt

$$\Sigma_{i=0}^{n} \int_{i=0}^{n} \log_{i=0}^{n} \cos_{i=0}^{n} \Pi_{i=0}^{n}$$

Traditional CONTEXT scaling versus advertized scaling for Cambria.

Implementation

Support for Cambria is rather straightforward:

- We load the OPENTYPE font table, create a suitable math TFM table from it (size chain, extensibles, etc).
- We pass the OPENTYPE math equivalent parameters directly to LUATEX.

Support for traditional TEX fonts follows a different route:

- We have defined virtual font handlers for this class of fonts.
- We create the virtual font on the fly.
- Missing glyphs are composed runtime so that we have only direct references.
- Instead of traditional TEX parameters we fake OPENTYPE parameters.

This way we have similar support at the TEX end for both OPENTYPE and traditional fonts and can drop in future Latin Modern and Gyre easily.

Virtual MKIV Font

```
mathematics.make_font ( "lmroman10-math", {
  { name="lmroman10-regular.otf", features="virtualmath", main=true },
  { name="lmmi10.tfm", vector="traditional-mi", skewchar=0x7F },
  { name="lmsy10.tfm", vector="traditional-sy", skewchar=0x30, parameters=true } ,
  { name="lmex10.tfm", vector="traditional-ex", extension=true } ,
  { name="msam10.tfm", vector="traditional-ma" },
  { name="msbm10.tfm", vector="traditional-mb" },
  { name="lmroman10-bold.tfm", "traditional-bf" } ,
  { name="lmmib10.tfm", vector="traditional-bi", skewchar=0x7F } ,
  { name="lmsans10-regular.tfm", vector="traditional-ss", optional=true },
  { name="lmmono10-regular.tfm", vector="traditional-tt", optional=true },
} )
```

```
\definefontsynonym
  [LMMathRoman10-Regular]
  [LMMath10-Regular@lmroman10-math]
```

```
\starttypescript [math] [palatino] [name]
  \definefontsynonym [MathRoman] [pxmath@px-math]
  \loadmapfile[original-youngryu-px.map]
\stoptypescript
```

No More Macros

All constructs are defined as natural UNICODE points with proper chained sizes and extensibles.

There are trackers to see what MKIV does with math (as all intermediate noad lists are processed at the LUA end):

```
\enabletrackers[math.analyzing]
\ruledhbox{$a = \sqrt{b^2 + \sin{c} - {1 \over \gamma}}$}
\disabletrackers[math.analyzing]
```

$$a = \sqrt{b^2 + \sin c - \frac{1}{\gamma}}$$

. . . continued

U+0221A: √ √ square root

width: 860800, height: 1207040, depth: 54400, italic: 0

mathclass: radical, mathname: surd

next: U+F05A9 √ => U+F05AB √ => U+F05AC √ => U+F05AD √ => U+F05AE

=> variants: U+023B7 ⎷ => U+020D3 => U+F0959

```
[0x221A] = {
    adobename = "radical",
    category = "sm", cjkwd = "a", description = "SQUARE ROOT",
    direction = "on", linebreak = "ai",
    mathclass = "radical", mathname = "surd", unicodeslot = 0x221A,
}
```

Math Classes

We use the character database to define math.

U+0002F: / / solidus

width: 642560, height: 915840, depth: 275200, italic: 0

mathsymbol: U+02044 ⁄

```
[0x002F] = {
    adobename = "slash", contextname = "textslash",
    category = "po", cjkwd = "na", description = "SOLIDUS",
    direction = "cs", linebreak = "sy",
    mathsymbol = 0x2044, unicodeslot = 0x002F,
}
```

. . . continued

Future versions of MKIV will support the dictionary model as used in MATHML 3 (and OPENMATH).

```
[0x2044] = {
    adobename = "fraction",
    category = "sm",
    contextname = "textfraction",
    description = "FRACTION SLASH",
    direction = "cs",
    linebreak = "is",
    mathspec = {
        { class = "binary", name = "slash" },
        { class = "close", name = "solidus" },
    },
    unicodeslot = 0x2044,
}
```

Dynamic Alphabets

```
$abc \bf abc \bi abc$
$\mathscript abcdefghijklmnopqrstuvwxyz %
  1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ$\par
$\mathfraktur abcdefghijklmnopqrstuvwxyz %
  1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ$\par
$\mathblackboard abcdefghijklmnopqrstuvwxyz %
  1234567890 ABCDEFGHIJKLMNOPQRSTUVWXYZ$\par
$\mathscript abc IRZ \mathfraktur abc IRZ %
  \mathblackboard abc IRZ \ss abc IRZ 123$
```

*abc***abc*****abc*** $\mathscr{abcdefghijklmnopqrstuvwxyz}1234567890\mathscr{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$
$\mathfrak{abcdefghijklmnopqrstuvwxyz}1234567890\mathfrak{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$
abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
$\mathscr{abcIRZ}\mathfrak{abcIRZ}$abcIRZabcIRZ123

*abc***abc*****abc*** $\mathscr{abcdefghijklmnopqrstuvwxyz}1234567890\mathscr{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$
$\mathfrak{abcdefghijklmnopqrstuvwxyz}1234567890\mathfrak{ABCDEFGHIJKLMNOPQRSTUVWXYZ}$
abcdefghijklmnopqrstuvwxyz1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
$\mathscr{abcIRZ}\mathfrak{abcIRZ}$abcIRZabcIRZ123

Handy Automatisms

```
$ (a{,}b) = (1{.}20{,}3{.}40) $
```

```
\setupmathematics[autopunctuation=no]
```

```
$(a,b) = (1.20,3.40) $
```

$(a, b) = (1.20, 3.40)$

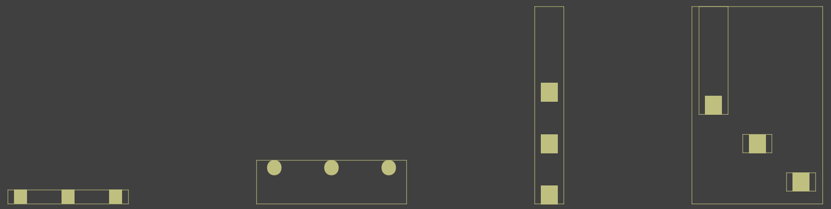
```
\setupmathematics[autopunctuation=yes]
```

```
$ (a,b) = (1.20,3.40) $
```

$(a{,}b) = (1{.}20{,}3{.}40)$

Reverse Engineering

```
\def\PLAINldots{\ldotp\ldotp\ldotp}
\def\PLAINcdots{\cdotp\cdotp\cdotp}
\def\PLAINvdots
  {\vbox{\forgetall\baselineskip.4\bodyfontsize\lineskiplimit\zeropoi
   \hbox{.}\hbox{.}\hbox{.}}}
\def\PLAINddots
  {\mkern1mu%
   \raise.7\bodyfontsize\ruledvbox{\kern.7\bodyfontsize\hbox{.}}%
   \mkern2mu%
   \raise.4\bodyfontsize\relax\ruledhbox{.}%
   \mkern2mu%
   \raise.1\bodyfontsize\ruledhbox{.}%
   \mkern1mu}
```



So . . .

In retrospect implementing support for OPENTYPE math has proven to be easier than expected, thanks to the fact that development and testing went smooth.

The virtual font building features made it possible to test OPENTYPE Cambria as well as traditional TEX fonts at the same time, something that was important because of all the new math parameters.

We're also confident that the upcoming extensions (like math alignment) can be tested and handled conveniently.