Handling math: A retrospective

Hans Hagen

In this article I will reflect on how the plain TEX approach to math fonts influenced the way math has been dealt with in ConTEXt MkII and why (and how) we diverge from it in MkIV, now that LuaTEX and OpenType math have come around.

When you start using TEX, you cannot help but notice that math plays an important role in this system. As soon as you dive into the code you will see that there is a concept of families that is closely related to math typesetting. A family is a set of three sizes: text, script and scriptscript.

$$a^{b^c} = \frac{d}{e}$$

The smaller sizes are used in superscripts and subscripts and in more complex formulas where information is put on top of each other.

It is no secret that the latest math font technology is not driven by the TEX community but by Microsoft. They have taken a good look at TEX and extended the OpenType font model with the information that is needed to do things similar to TEX and beyond. It is a firm proof of TEX's abilities that after some 30 years it is still seen as the benchmark for math typesetting. One can only speculate what Don Knuth would have come up with if today's desktop hardware and printing technology had been available in those days.

As a reference implementation of a font, Microsoft provides Cambria Math. In the specification the three sizes are there too: a font can provide specifically designed script and scriptscript variants for text glyphs where that is relevant. Control is exercised with the `ssty` feature.

Another inheritance from TEX and its fonts is the fact that larger symbols can be made out of snippets and these snippets are available as glyphs in the font, so no special additional (extension) fonts are needed to get for instance really large parentheses. The information of when to move up one step in size (given that there is a larger shape available) or when and how to construct larger symbols out of snippets is there as well. Placement of accents is made easy by information in the font and there are a whole lot of parameters that control the typesetting process. Of course you still need machinery comparable to TEX's math subsystem but Microsoft Word has such capabilities.

I'm not going to discuss the nasty details of providing math support in TEX, but rather pay some attention to an (at least for me) interesting side effect of TEX's math machinery. There are excellent articles by Bogusław Jackowski and Ulrik Vieth about how TEX constructs math and of course Knuth's publications are the ultimate source of information as well.

Even if you only glance at the implementation of traditional TEX font support, the previously mentioned families are quite evident. You can have 16 of them but 4 already have a special role: the upright roman font, math italic, math symbol and math extension. These give us access to some 1000 glyphs in theory, but when TEX showed up it was mostly a 7-bit engine and input of text was often also 7-bit based, so in practice many fewer shapes are available, and subtracting the snippets that make up the large symbols brings down the number again.

Now, say that in a formula you want to have a bold character. This character is definitely not in the 4 mentioned families. Instead you enable another one, one that is linked to a bold font. And, of course there is also a family for bold italic, slanted, bold slanted, monospaced, maybe smallcaps, sans serif, etc. To complicate things even more, there are quite a few symbols that are not covered in the foursome so we need another 2 or 3 families just for those. And yes, bold math symbols will demand even more families.

$$a + \mathbf{b} + \boldsymbol{c} = \mathtt{d} + \mathtt{e} + \mathcal{F}$$

Try to imagine what this means for implementing a font system. When (in for instance ConTEXt) you choose a specific body font at a certain size, you not only switch the regular text fonts, you also initialize math. When dealing with text and a font switch there, it is no big deal to delay font loading and initialization till you really need the font. But for math it is different. In order to set up the math subsystem, the families need to be known and set up and as each one can have three members you can imagine that you easily initialize some 30 to 40 fonts. And, when you use several math setups in a document, switching between them involves at least some re-initialization of those families.

When Taco Hoekwater and I were discussing LuaTEX and especially what was needed for math, it was sort of natural to extend the number of families to 256. After all, years of traditional usage had demonstrated that it was pretty hard to come up with math font support where you could freely mix a whole regular and a whole bold set of characters simply because you ran out of families. This is a side effect of math processing happening in several passes: you can change a family definition within a formula, but as TEX remembers only the family

number, a later definition overloads a previous one. The previous example in a traditional TEX approach can result in:

```
a + \fam7 b + \fam8 c = \fam9 d + \fam10 e
  + \fam11 F
```

Here the a comes from the family that reflects math italic (most likely family 1) and + and = can come from whatever family is told to provide them (this is driven by their math code properties). As family numbers are stored in the identification pass, and in the typesetting pass resolve to real fonts you can imagine that overloading a family in the middle of a definition is not an option: it's the number that gets stored and not what it is bound to. As it is unlikely that we actually use more than 16 families we could have come up with a pool approach where families are initialized on demand but that does not work too well with grouping (or at least it complicates matters).

So, when I started thinking of rewriting the math font support for ConTEXt MkIV, I still had this nicely increased upper limit in mind, if only because I was still thinking of support for the traditional TEX fonts. However, I soon realized that it made no sense at all to stick to that approach: OpenType math was on its way and in the meantime we had started the math font project. But given that this would easily take some five years to finish, an intermediate solution was needed. As we can make virtual fonts in LuaTEX, I decided to go that route and for several years already it has worked quite well. For the moment the traditional TEX math fonts (Computer Modern, px, tx, Lucida, etc) are virtualized into a pseudo-OpenType font that follows the Unicode math standard. So instead of needing more families, in ConTEXt we could do with less. In fact, we can do with only two: one for regular and one for bold, although, thinking of it, there is nothing that prevents us from mixing different font designs (or preferences) in one formula but even then a mere four families would still be fine.

To summarize this, in ConTEXt MkIV the previous example now becomes:

```
U+1D44E + U+1D41B + 0x1D484 = U+1D68D + U+1D5BE
        + U+02131
```

For a long time I have been puzzled by the fact that one needs so many fonts for a traditional setup. It was only after implementing the ConTEXt MkIV math subsystem that I realized that all of this was only needed in order to support alphabets, i.e. just a small subset of a font. In Unicode we have quite a few math alphabets and in ConTEXt we have ways to map a regular keyed-in (say) 'a' onto a bold or monospaced one. When writing that code I hadn't even linked the Unicode math alphabets to the family approach for traditional TEX. Not being a mathematician myself I had no real concept of systematic usage of alternative alphabets (apart from the occasional different shape for an occasional physics entity).

Just to give an idea of what Unicode defines: there are alphabets in regular (upright), bold, italic, bold italic, script, bold script, fraktur, bold fraktur, double-struck, sans-serif, sans-serif bold, sans-serif italic, sans-serif bold italic and monospace. These are regular alphabets with upper- and lowercase characters complemented by digits and occasionally Greek.

It was a few years later (somewhere near the end of 2010) that I realized that a lot of the complications in (and load on) a traditional font system were simply due to the fact that in order to get one bold character, a whole font had to be loaded in order for families to express themselves. And that in order to have several fonts being rendered, one needed lots of initialization for just a few cases. Instead of wasting one font and family for an alphabet, one could as well have combined 9 (upper and lowercase) alphabets into one font and use an offset to access them (in practice we have to handle the digits too). Of course that would have meant extending the TEX math machinery with some offset or alternative to some extensive mathcode juggling but that also has some overhead.

If you look at the plain TEX definitions for the family related matters, you can learn a few things. First of all, there are the regular four families defined:

```
\textfont0=\tenrm \scriptfont0=\sevenrm
  \scriptscriptfont0=\fiverm
\textfont1=\teni  \scriptfont1=\seveni
  \scriptscriptfont1=\fivei
\textfont2=\tensy \scriptfont2=\sevensy
  \scriptscriptfont2=\fivesy
\textfont3=\tenex \scriptfont3=\tenex
  \scriptscriptfont3=\tenex
```

Each family has three members. There are some related definitions as well:

```
\def\rm      {\fam0\tenrm}
\def\mit     {\fam1}
\def\oldstyle{\fam1\teni}
\def\cal     {\fam2}
```

So, with \rm you not only switch to a family (in math mode) but you also enable a font. The same is true for \oldstyle and this actually brings us to another interesting side effect. The fact that oldstyle numerals come from a math font has implications for the way this rendering is supported in macro packages. As naturally all development started when TEX came around, package design decisions were

driven by the basic fact that there was only one math font available. And, as a consequence most users used the Computer Modern fonts and therefore there was never a real problem in getting those oldstyle characters in your document.

However, oldstyle figures are a property of a font design (like table digits) and as such not specially related to math. And, why should one tag each number then? Of course it's good practice to tag extensively (and tagging makes switching fonts easy) but to tag each number is somewhat over the top. When more fonts (usable in TEX) became available it became more natural to use a proper oldstyle font for text and the `\oldstyle` more definitely ended up as a math command. This was not always easy to understand for users who primarily used TEX for anything but math.

Another interesting aspect is that with OpenType fonts oldstyle figures are again an optional feature, but now at a different level. There are a few more such traditional issues: bullets often come from a math font as well (which works out ok as they have nice, not so tiny bullets). But the same is true for triangles, squares, small circles and other symbols. And, to make things worse, some come from the regular TEX math fonts, and others from additional ones, like the American Mathematical Society symbols. Again, OpenType and Unicode will change this as now these symbols are quite likely to be found in fonts as they have a larger repertoire of shapes.

From the perspective of going from MkII to MkIV it boils down to changing old mechanisms that need to handle all this (dependent on the availability of fonts) to cleaner setups. Of course, as fonts are never completely consistent, or complete for that matter, and features can be implemented incorrectly or incompletely we still end up with issues, but (at least in ConTEXt) dealing with that has been moved to runtime manipulation of the fonts themselves (as part of the so-called font goodies).

Back to the plain definitions, we now arrive at some new families:

```
\newfam\itfam \def\it{\fam\itfam\tenit}
\newfam\slfam \def\sl{\fam\slfam\tensl}
\newfam\bffam \def\bf{\fam\bffam\tenbf}
\newfam\ttfam \def\tt{\fam\ttfam\tentt}
```

The plain TEX format was never meant as a generic solution but instead was an example of a macro set and serves as a basis for styles used by Don Knuth for his books. Nevertheless, in spite of the fact that TEX was made to be extended, pretty soon it became frozen and the macros and font definitions that came with it became the benchmark. This might be the reason why Unicode now has a monospaced alphabet. Once you've added monospaced you might as well add more alphabets as for sure in some countries they have their own preferences.[1]

As with `\rm`, the related commands are meant to be used in text as well. More interesting is to see what follows now:

```
\textfont        \itfam=\tenit
\textfont        \slfam=\tensl

\textfont        \bffam=\tenbf
\scriptfont      \bffam=\sevenbf
\scriptscriptfont\bffam=\fivebf

\textfont        \ttfam=\tentt
```

Only the bold definition has all members. This means that (regular) italic, slanted, and monospaced are not actually that much math at all. You will probably only see them in text inside a math formula. From this you can deduce that contrary to what I said before, these variants were not really meant for alphabets, but for text in which case we need complete fonts. So why do I still conclude that we don't need all these families? In practice text inside math is not always done this way but with a special set of text commands. This is a consequence of the fact that when we add text, we want to be able to do so in each language with even language-specific properties supported. And, although a family switch like the above might do well for English, as soon as you want Polish (extended Latin), Cyrillic or Greek you definitely need more than a family switch, if only because encodings come into play. In that respect it is interesting that we do have a family for monospaced, but that `\Im` and `\Re` have symbolic names, although a more extensive setup can have a blackboard family switch.

By the way, the fact that TEX came with italic alongside slanted also has some implications. Normally a font design has either italic or something slanted (then called oblique). But, Computer Modern came with both, which is no surprise as there is a metadesign behind it. And therefore macro packages provide ways to deal with those variants alongside. I wonder what would have happened if this had not been the case. Nowadays there is always this regular, italic (or oblique), bold and bold italic set to deal with, and the whole set can become lighter or bolder.

In ConTEXt MkII, however, the set is larger as we also have slanted and bold slanted and even

[1] At the Dante 2011 meeting we had interesting discussions during dinner about the advantages of using Sütterlinschrift for vector algebra and the possibilities for providing it in the upcoming TEX Gyre math fonts.

smallcaps, so most definition sets have 7 definitions instead of 4. By the way, smallcaps is also special. if Computer Modern had had smallcaps for all variants, support for them in ConTEXt undoubtedly would have been kept out of the mentioned 7 but always been a new typeface definition (i.e. another fontclass for insiders). So, when something would have to be smallcaps, one would simply switch the whole lot to smallcaps (bold smallcaps, etc.). Of course this is what normally happens, at least in my setups, but nevertheless one can still find traces of this original Computer Modern-driven approach. And now we are at it: the whole font system still has the ability to use design sizes and combine different ones in sets, if only because in Computer Modern you don't have all sizes. The above definitions use ten, seven and five, but for instance for an eleven point set up you need to creatively choose the proper originals and scale them to the right family size. Nowadays only a few fonts ship with multiple design sizes, and although some can be compensated with clever hinting it is a pity that we can apply this mechanism only to the traditional TEX fonts.

Concerning the slanting we can remark that TEXies are so fond of this that they even extended the TEX engines to support slanting in the core machinery (or more precisely in the backend while the frontend then uses adapted metrics). So, slanting is available for all fonts.

This brings me to another complication in writing a math font subsystem: bold. During the development of ConTEXt MkII I was puzzled by the fact that user demands with respect to bold were so inconsistent. This is again related to the way a somewhat simple setup looks: explicitly switching to bold characters or symbols using a `\bf` (alike) switch. This works quite well in most cases, but what if you use math in a section title? Then the whole lot should be in bold and an embedded bold symbol should be heavy (i.e. more bold than bold). As a consequence (and due to limited availability of complete bold math fonts) in MkII there are several bold strategies implemented.

However, in a Unicode universe things become surprisingly easy as Unicode defines those symbols that have bold companions (whatever you want to call them, mostly math alphanumerics) so a proper math font has them already. This limited subset is often available in a font collection and font designers can stick to that subset. So, eventually we get one regular font (with some bold glyphs according to the Unicode specification) and a bold companion that has heavy variants for those regular bold shapes.

The simple fact that Unicode distinguishes regular and bold simplifies an implementation as it's easier to take that as a starting point than users who for all their goodwill see only their small domain of boldness.

It might sound like Unicode solves all our problems but this is not entirely true. For instance, the Unicode principle that no character should be there more than once has resulted in holes in the Unicode alphabets, especially Greek, blackboard, fraktur and script. As exceptions were made for non-math I see no reason why the few math characters that now put holes in an alphabet could not have been there. As with more standards, following some principles too strictly eventually results in all applications that follow the standard having to implement the same ugly exceptions explicitly. As some standards aim for longevity I wonder how many programming hours will be wasted this way.

This brings me to the conclusion that in practice 16 families are more than enough in a Unicode-aware TEX engine especially when you consider that for a specific document one can define a nice set of families, just as in plain TEX. It's simply the fact that we want to make a macro package that does it all and therefore has to provide all possible math demands into one mechanism that complicates life. And the fact that Unicode clearly demonstrates that we're only talking about alphabets has brought (at least) ConTEXt back to its basics: a relatively simple, few-family approach combined with a dedicated alphabet selection system. Of course eventually users may come up with new demands and we might again end up with a mess. After all, it's the fact that TEX gives us control that makes it so much fun.

◇ Hans Hagen
`http://pragma-ade.com`