Multiple Master Math Extension Fonts

September 28, 2003

From its inception, TEX relied on the concept of extensible (composite) characters to implement large glyphs that occur in mathematical typesetting. While thousands of papers have been successfully written using this technology, there are obvious shortcomings of this approach. This article demonstrates them, as well as an alternative solution, free from the problems.

1 The traditional approach

The extensible glyphs in TEX are composed out of several smaller characters, and, in some cases, rules. The construction itself is carried either by code inside the TEX compiler itself, using the special information in the `.tfm` files, or by macros, implemented as part of the format.

An example of the first situation is the construction of large delimiters, for example, brackets. Here, a vertical segment in inserted to stretch the delimiters vertically:

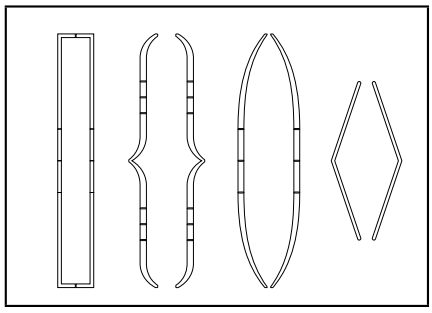
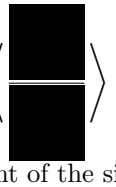
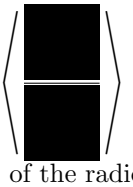


Figure 1: Vertical delimiters

One of the shortcomings of the extensible characters is that some character shapes do not lend themselves to the insertion of extension pieces; thus, the angular brackets are non-extensible, and, when used to encompass a large formula, would not scale to the required size. For example, typing

```
$\left< \vrule width 1cm height 1cm \over
        \vrule width 1cm height 1cm \right>$
```

results in  rather than in the anticipated 

A variant of the situation is the TEX's treatment of the radical symbol: the construction is accomplished by the TEX program itself, but while the vertical extension is driven by the `.tfm` information, the horizontal bar is supplied by the TEX program itself; and in this case the bar is a rule, rather than an extension character.

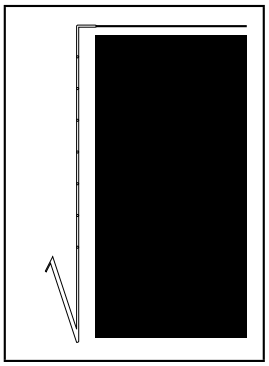
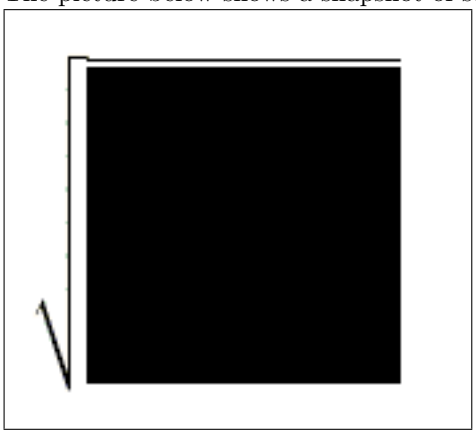


Figure 2: Radical delimiters

In the above picture, the rule component is shown filled.

The problem with the use of rule components in building a composite symbol is that rules are subjected to different roundoff rules than characters. In the prehistoric days of DVI files and bitmap fonts, these roundoff errors could be in principle handled by careful calculations within the TEX program and the DVI driver; but in the modern world of scalable fonts and PS/PDF targeting this becomes an impossibility.

The picture below shows a snapshot of square root constructed by PdfTEX:



The underlying TEX code was

```
$ \sqrt{\vrule height 2cm width 2cm} $
```

Depending on the magnification used in the PDF previewer, or resolution of the output device, the rule may be thinner, or thicker, or above, or below the part of the `sqrt` glyph it is supposed to connect to smoothly. And, sometimes, it would fit correctly.

The root of the problem here is that glyph rounding is subject to the font hinting, while the thickness and positioning of rules is not well controlled by the PS/PDF rendering.

The second type of extension mechanism is via TEX macros; this is typically used to construct long horizontal symbols, as done by commands similar to `\underbrace`:

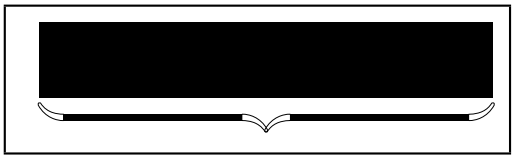
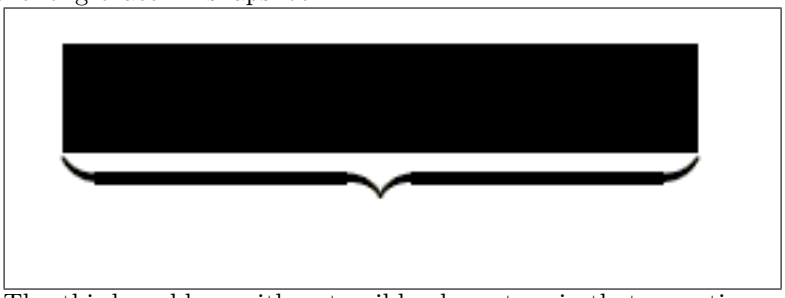


Figure 3: Underbrace extension

Once again, we get mismatches in the width and positioning of the connecting rules comparing to that of the symbols used in the middle and at the ends of the long brace. A snapshot:



The third problem with extensible characters is that sometimes (usually, in slides) it is desirable to outline, or shade characters; this cannot be done correctly with composite characters. In fact, the diagrams above that show the structure of extensible characters, are *unintended output* of an attempt to stroke extensible symbols, making them suitable for this article, but not for a use within a slide.

Summarizing, we have identified three separate problems with the TEX approach to the extensible symbols:

- Impossibility to correctly scale some of the delimiters (angular brackets, for example).
- Impossibility to correctly align rules and characters.

- Impossibility to correctly stroke or shade composite symbols.

All these problems can be solved, however, if we switch to a different mechanism of constructing large symbols: multiple-master fonts.

2 Multiple Master Fonts

For those unfamiliar with the MM fonts, they represent a PostScript World answer to MetaFont. Just like MetaFont allows creation of different designs from the same font program, so does the MM technology. While MM fonts are less flexible than MetaFont, they are easy to use, and the output of MM font instancing is a ready-to-use Type 1 font, whereas a generic MetaFont emits obsolete bitmapped fonts.

Additional information on the MM fonts can be found in the Adobe Technical Specification #5015 *"Type 1 Font Format Supplement"*.

The Multiple-Master (MM) Font technology was introduced by Adobe in 1991; the first description of the MM technology appeared in the PC Week article of the March 11, 1991 issue. The first MM font was Myriad (1992), with two axes: weight and width.

Since 1992 Adobe designed at least 36 MM font families with 99 fonts. Perhaps the most successful was the Minion family.

While Adobe originally intended to include MM in the OpenType specifications, this effort has been abandoned, and Adobe stopped making new MM fonts. The last Adobe's MM font, VerveMM, was designed in 1998; Adobe had announced that it was giving up on the MM technology at the 1999 ATypI Congress.

Despite the abandonment by Adobe, MM fonts represents a very convenient technology for use in typesetting aplications like TEX. While MM functionality is not supported within PDF documents, instances of MM fonts are, and the entire MM font can be made available to the TEX document.

VTEX has supported MM fonts since 2001, and the use of MM fonts in math extension fonts merely builds up on the existing MM support; we will start with outlining the existing support.

VTEX recognizes MM fonts by the name of the font; a font name that includes the bracket character is assumed to be an instance of a Multiple Master font. This does not represent much of a restriction on the font name selection, since use of brackets in file names is not common and is not legal on some operating systems.

The left bracket in the font name is followed by the instancing parameters. For example, in

```
\font\sm=xo______
\font\mm=xo______[300,10]
```

the `\sm \font` declaration defines the default shape of the Multiple Master CrononMM font; this is the "normal" way to use the typeface. The second

declaration, however, defines an instance of this MM font to be constructed dynamically.

In the above example, the first declaration causes VTEX to load the usual .tfm metrics file. (`xo______.tfm`); the second declaration causes VTEX to load the Multiple font Metrics file (`xo______.mfm`) and generate the instance metrics on the fly. Because of the use of `.mfm` files providing metrics file for each instance becomes unnecessary.

In some MM fonts, all the instances of the font have the same metrics; if so, it is possible to use the ordinary `.tfm` file. In most MM fonts, however, the metrics of different instances is different, and this made the development of the new metrics format necessary.

Upon seeing a MM font used in the document, VTEX compiler would automatically use the build-in PostScript interpreter (GeX) to instance it; since the instancing is done by a PostScript execution, the instance font is always built correctly; and the TEX compiler automatically packs it into the output PDF file.

For example, the following TEX code would provide a set of instances of the Minion font (TM).

Additional details on the MM support can be found in the `mmsupp.pdf` document in VTEX distributions.

3 Math Extension Multiple Master Fonts

Supporting Math Extension MM fonts requires several additional steps.

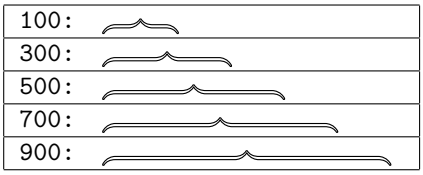
Firstly, such fonts needs to be developed themselves; currently there are two, `cmex10mm` and `paex10mm`. The first is intended for use with Computer Modern fonts, the second with the alternative PaMATH set, available from MicroPress.

Math Extension MM fonts include symbols for vertical delimiters, long horizontal symbols (like the ones constructed by the `\underbrace` macro), and the radical symbol. Since the radical requires two MM axes, the entire font is a two-axes MM font, even if the majority of symbols are actually "one-dimensional".

Specific sizes of delimiters can be constructed by loading the font with different instancing parameters. For example, for the `\overbrace` symbol, we can try font commands like

```
\font\f=cmex10mm[30,100] \f \char"7A
\font\f=cmex10mm[30,300] \f \char"7A
\font\f=cmex10mm[30,500] \f \char"7A
\font\f=cmex10mm[30,700] \f \char"7A
\font\f=cmex10mm[30,900] \f \char"7A
```

obtaining the shapes



(Only the second instancing parameter is used in long horizontal symbols.)

Notice that the symbols are built from single glyphs and thus do not suffer from the problems listed at the beginning of this paper.

Since TEX supports `overbrace` and other long horizontal symbols entirely through macros, switching TEX to supporting MM instances instead requires only changes to the macros. One of the tasks of the `mathexmm` style is to therefore redefine these macros.

Supporting vertical delimiters and radicals requires more radical changes: the `.tfm` mechanism of extensible characters needs to be replaced by an MM alternative. We accomplish this as follows:

Firstly, we prepare an alternative metrics file, `cmex10m`, which is mostly equivalent to `cmex10`, except that

- The encoding specified in the `.tfm` is `MMEXTENSION`; this is the signal to the TEX compiler that what normally would be interpreted as exten instructions in the `.tfm` instead should be seen as pointers to MM glyphs. (It is unfortunate that the `.tfm` syntax does not provide any space for new flags; this is what forces us to use the comment field.) In a `.PL` file, this would appear as

  ```
  (CODINGSCHEME MMEXTENSION)
  ```

- Each `exten` specification in the `.tfm` is replaced by the glyph number of the character to use in the `cmex10mm` MM font. For example, whereas the `cmex10.pl` listing contains

  ```
  (CHARACTER C 0
     (CHARWD R 0.875003)
     (CHARHT R 0.039999)
     (CHARDP R 1.760019)
     (VARCHAR
        (TOP C 0)
        (BOT O 100)
        (REP C B)
        )
     )
  ```

  specifying that the character `0` is to be built from glyphs `0`, `‘100` and `B`; in `cmex10m.pl` we set

  ```
  (CHARACTER C 0
     (CHARWD R 0.875003)
     (CHARHT R 0.039999)
     (CHARDP R 1.760019)
     (VARCHAR
        (REP O 303)
  ```

```
    )
  )
```

which means that the character `0` is to be built as an instance of the glyph `‘0303` in the corresponding MM font (`cmex10mm`).

`cmex10` and `cmex10m` are otherwise identical.

The `mathexmm` style, when used, would load the `cmex10m` font instead of `cmex10`. Unless extensible characters are invoked, it would function in exactly the same way as `cmex10`; by when TEX is about to build an extensible character, it would instead build an appropriately-sized instance from the `cmex10mm` MM font.

From the point of view of the user, this is all invisible, and no action is required except for including the the `mathexmm` style. This style currently exists for Plain TEX, LATEX, and AmSTEX.