Math typesetting in TEX: The good, the bad, the ugly

Ulrik Vieth

1. Introduction

The topic of math typesetting (in general) and math fonts (in particular) has been a recurring theme at TEX conferences for many years. Most of the time these papers and talks have focussed on developing new math font encodings [1–3], updating and standardizing the set of math symbols in Unicode [4–6], or on implementing math fonts for use with a variety of font families [7–11]. However, fundamental technical issues and limitations of TEX's math typesetting engine have only rarely been addressed [12–15], usually in conjunction with a broader discussion of TEX's shortcomings.

In this paper we shall examine the latter topic in detail, trying to clarify what are the good, the bad, and the ugly bits of TEX's math typesetting engine.

2. Math typesetting: some good and some bad news

Let's start with the good news: Even after some twenty years of age, TEX is still very good at typesetting math. While some other systems such as Adobe InDesign have been catching up in the domain of text typesetting, even borrowing ideas from TEX's algorithms, math typesetting remains a domain where TEX is still at its best.

Whereas other systems usually tend to regard math as an add-on feature for a niche market that's very costly to develop and rarely pays off, math typesetting has always played a central role in TEX. In fact, math typesetting has been one of the main reasons why TEX was developed in the first place and why it has become so successful in the academic community and among math and science publishers.

While there are some subtle details that TEX can't handle automatically and that might benefit from a little manual fine-tuning, TEX usually does a very good job of typesetting properly-coded math formulas all by itself, without requiring users to care about how TEX's math typesetting engine actually works internally.

In general, an experienced TEX user, who has taken the time to learn a few rules and pay attention to a few details, can easily produce publication-quality output of even the most complicated math formulas by tastefully applying a few extra keystrokes to help TEX a little bit. And even an average TEX user, who is unaware of all the subtle details, can usually produce something good enough to get away with for use in seminar papers or thesis reports, often producing better result than what a casual user would get from so-called equation editors of typical word processors.

So, what's the bad news, after all? Actually, the problems only begin to emerge when leaving the user's perspective behind and looking at TEX's math typesetting engine from the implementor's point of view. While the quality of math typeset with TEX is probably still unmatched, some aspects of the underlying math typesetting engine itself are unfortunately far from perfect.

As anybody can tell, who has ever studied Appendix G of *The TEXbook*, trying to understand what's really going on when typesetting a math formula, TEX's math typesetting engine is a truly complicated beast, which relies on a number of peculiar assumptions about the way math fonts are expected to be built. Moreover, there are also some limitations and shortcomings where TEX begins to show its age.

3. Math typesetting: some technical background

Before we get into further details, it may be helpful to summarize how TEX's math mode differs from text mode and what goes on when typesetting text or math.

3.1. What goes on in text mode: `\char`s, fonts and glyphs

In text mode, when typesetting paragraphs of text, TEX essentially does nothing but translate input character codes to output codes using the currently selected font, assemble sequences of boxes and glue (i. e. letters or symbols represented by their font metrics and interword spaces) into paragraphs, and subsequently break paragraphs into lines and eventually lines into pages, as they are shipped out.

Whatever the levels of abstraction added by font selection schemes implemented in complex macro packages such as LATEX or context, all typeset output is essentially generated by loading a particular font (i. e. a specific font shape of a specific family at a specific design size encoded in a specific font encoding) using the `\font\f=`⟨*font*⟩ primitive, selecting that font as the current font, and accessing glyphs from that font through the code positions of the output encoding.

Most input characters typed on the keyboard (except for those having special `\catcode`s for various reasons) are first translated to TEX's internal encoding (based on 7-bit ascii and the `^^` notation for 8-bit codes), from which they are further translated to output codes by an identity mapping. (There is no such thing as a global `\charcode` table to control this mapping.) Additional letters and symbols (such as `\ss` for 'ß') can be accessed through the `\char`⟨*code*⟩ primitive or by macros using `\chardef\c=`⟨*code*⟩, where ⟨*code*⟩ depends on the font encoding.

In actual fact, there are, of course, some further complications to typesetting text beyond this superficial description, such as dealing with ligatures, accented letters, or constructed symbols. Moreover, there are additional input methods than just converting characters typed on the keyboard or accessed through macros, such as using active characters or input ligatures to access special characters, but we don't want to go too far into such encoding issues in this paper.[1]

3.2. What goes on in math mode: `\mathchar`s, math symbols and math families

When it comes to math mode, things are, of course, a little more complicated than in text mode. For instance, TEX doesn't deal with specific fonts and character codes in math mode, but uses the concepts of math families and math codes instead. Whereas modern implementations of TEX provide room for several hundreds of text fonts, there is a limit of only 16 math families, each containing at most 256 letters or symbols. Compared to a text font, representing a specific font shape at a specific size, a math family represent a whole set of corresponding symbol fonts, which are loaded at three different sizes known as textstyle, scriptstyle and scriptscriptstyle.

In a typical setup of TEX, there should be at least four math families preloaded, where family 0 is a math roman font, family 1 is a math italic font, family 2 contains math symbols, and family 3 contains big operators and delimiters. Some assumptions about this are actually hard-wired into TEX, such as the requirement that the fonts used in families 2 and 3 have to provide a number of `\fontdimen` parameters controlling the placement of various elements of math formulas.

Any letter or symbol used in math mode, whether typed on the keyboard or accessed through a macro, is always represented by a math code, usually written as 4-digit hexadecimal number. In addition to specifying a math family and a character code, the math code also encodes information about the type of a symbol, whether it is an ordinary symbol, a big operator (such as $\int$), a binary operator (such as $+$), a relation (such as $=$), an opening or closing delimiter, or a punctuation character. (There is also a special type of ordinary symbols, which are allowed to switch math families. This particular type is mostly used for alphabetic symbols.)

The mapping of input characters typed on the keyboard to corresponding symbols is controlled through a `\mathcode` table, which by default maps letters to math italics and numbers to math roman. Additional math symbols including the letters of the greek alphabet can be accessed by macros using `\mathchardef\c=`⟨*code*⟩, where ⟨*code*⟩ is a math code composed of type, math family and character code. In a similar way, special types of symbols such as delimiters and radicals are handled using macros involving `\delimiter`⟨*code*⟩ or `\radical`⟨*code*⟩.

Considering the two-dimensional nature of typesetting math, it should be obvious that there is much more to it than simply translating input math codes to output character codes of specific fonts. In addition to choosing the proper symbols (based on the math families and character codes stored in the math codes), it is equally important to determine the proper size (based on the three sizes of fonts loaded in each math family) and to place the symbols at the proper position relative to other symbols with an appropriate amount of space

[1] LATEX uses the `inputenc` and `fontenc` packages to deal with 8-bit input and output encodings beyond 7-bit ascii. Most 8-bit input codes for accented letters are first mapped to replacement macros through active characters. These, in turn, are subsequently mapped back to 8-bit output codes. For a detailed discussion on what really goes on internally in the various processing stages and what constitutes the subtle differences between characters, glyphs, and slots, see [16].

in between. Here, the type information stored in the math codes comes into play, as TEX uses a built-in spacing table to determine which amount of space (i. e. either a thin space, medium space, thick space, or no space at all) will be inserted between adjacent symbols.

3.3. Interaction between the math typesetting engine and math fonts

It is interesting to note that TEX's math typesetting engine relies on a certain amount of cooperation between its built-in rules, parameters set up in the format file, and parameters stored in the font metrics of math fonts.

For example, when determining the spacing between symbols, the spacing table that defines which amount space will be inserted is hard-wired into TEX, while the amounts of space are determined by parameters such as `\thinmuskip`, `\medmuskip` or `\thickmuskip`, which are set up in the format file. These parameters are defined in multiples of the unit $1\ \mathrm{mu} = 1/18\ \mathrm{em}$, which, in turn, depends on the font size. Similarly, when processing complex sub-formulas, such as building fractions, attaching subscripts and superscripts, or attaching limits to big operators, the actual typesetting rules are, of course, built into TEX itself, whereas various parameters controlling the exact placement are taken from `\fontdimen` parameters.

In view of the topic of this paper, it should be no surprise that such kind of close cooperation between the math typesetting engine and the math fonts does not come without problems. While there are good reasons why some of these parameters depend on the font metrics, it might be a problem that their scope is not limited to the individual fonts loaded in math families 2 and 3; they automatically apply to the whole set of math fonts. (This is usually not a problem when a consistent set of math fonts is used, but this assumption might break and might lead to problems when trying to mix and match letters and symbols from different sets of math fonts.)

4. Specific problems of TEX's math fonts

After reviewing the technical background of math typesetting, we shall now look into some specific problems of TEX's math typesetting engine. In particular, we will focus on those problems that make it hard to implement new math fonts.

4.1. Glyph metrics of ordinary symbols: When the tfm width isn't the real width...

Perhaps the most irritating feature of TEX's math fonts is the counter-intuitive way, in which glyph metrics are represented differently from those of text fonts. Normally, the font metrics stored in tfm files contain four fields of per-glyph information for each character or symbol: a *height* (above the baseline), a *depth* (below the baseline), a *width*, and an *italic correction* (which might be zero for upright fonts). In math fonts, however, glyph metrics are interpreted differently. Since additional information needs to be stored within the framework of the same four fields of per-glyph information, some fields are interpreted in an unusual way: The *width* field is used to denote the position where subscripts are to be attached, while the *italic correction* field is used to denote the offset between the subscript and superscript position. As a result, the real width isn't directly accessible and can only be determined by adding up the *width* and *italic correction* fields. Moreover, the information stored in the *width* field usually differs from the real width, which causes subsequent problems.

Most importantly, this peculiar representation of glyph metrics causes a lot of extra work for implementors of math fonts, since they can't simply take an italic text font and combine it with a suitable symbol font to make up a math font. Instead the metrics taken from an italic text font have to be tuned by a process of trial and error and subsequent refinements to arrive at optimal values for the placement of subscripts and superscripts as well as for the side-bearings of letters and symbols.

4.2. Placement of math accents: When you need a `\skewchar` to get it right...

Another problem related to glyph metrics arises as an immediate consequence of the previous one. Since the *width* field of the glyph metrics of math fonts doesn't contain the real glyph width, TEX's default algorithm for placing and centering accents or math accents doesn't work, and a somewhat cumbersome work-around was invented, the so-called `\skewchar` mechanism. The basic idea is to store the shift amounts to correct the placement of math accents in a set of special kern pairs in the font metrics. To this effect, a single character of each math font (usually a non-letter) is designated as the `\skewchar` and kern pairs are registered between all other characters that

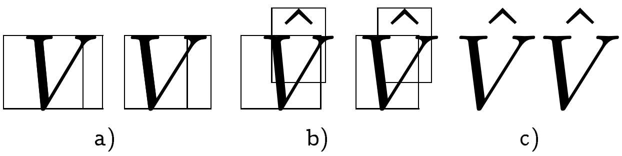

Figure 1: Placement of accents in text mode and math mode, comparing cmti10 and cmmi10 at 40 pt. (a) Comparing the glyph widths and side-bearings for text italic and math italic. (b) Comparing the results of TEX's default algorithm for accent placement, producing slightly different results for slightly different metrics of text italic and math italic. (c) Comparing the results of TEX's \accent and \mathaccent primitives, illustrating the correction due to \skewchar kerning.

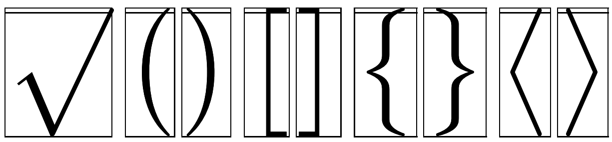

Figure 2: Glyph metrics of big radicals and delimiters in math extension fonts (showing cmex10 at 40 pt). The height above the baseline matches exactly the default rule thickness, as required for the rule part of radicals. All glyphs of the same size are placed in the same position to cope with limitations of the tfm file format.

may be accented (letters or letter-like symbols) and the selected \skewchar.

As in the previous case, the most important problem of the \skewchar mechanism (apart from being hack) is that it causes extra work to implementors of math fonts. Instead of being able to rely on TEX's default algorithm for the placement of accents, the \skewchar kern pairs have to be tuned to arrive at optimal values. Moreover, the choice of the \skewchar has to be considered carefully to avoid interference with normal kern pairs in math fonts, such as between sequences of ordinary symbols or between ordinary symbols and delimiters or punctuation.

(Another problem related to kerning in math fonts is that TEX doesn't support kerning between multiple fonts, so it isn't possible to define kern pairs between upright and italic letters taken from different fonts, but that's another story.)

4.3. Glyph metrics of big symbols: When glyphs hang below the baseline ...

Another quite different problem related to glyph metrics, which occurs only in the math extension font, is the placement of big symbols (big operators, big delimiters and radicals) within their bounding boxes. As anyone will have noticed, who has ever printed out a font table of cmex10 using testfont.tex or has looked at Volume E of *Computers & Typesetting*, most symbols in the math extension font have very unusual glyph metrics, where the glyphs tend to hang far below the baseline.

The reasons for this are a little complicated and quite involved. To some extent they are due to technical requirements, such as in the case of big radicals where the height above the baseline is used to determine the rule thickness of the horizontal rule on top of a root. However, in other cases, such as big operators and delimiters, there are no technical requirements for the unusual glyph metrics (at least not in TEX82) and the reasons are either coincidental or due to limitations of the tfm file format, which doesn't support more than 16 different heights and depths in one font.

(Incidently, the height of big radical glyphs is usually exactly the same as the default rule thickness specified in the \fontdimen parameters, so one could have used just that instead of relying on the glyph metrics to convey this information.)

What is particularly irritating about this problem is that math fonts featuring such glyph metrics are usually not usable with any other typesetting system besides TEX. While TEX automatically centers operators, delimiters and radicals on the math axis, most other systems expect to find glyphs in an on-axis position as determined by the type designer. It therefore becomes extremely hard, if not impossible, to develop math fonts equally usable with TEX and with other math typesetting systems. (The font set distributed with *Mathematica* avoids this problem by providing two different sets of radicals, occupying different code positions in the font encoding.)

4.4. Extensible delimiters: When the intelligence lies in the font...

Speaking of math extension fonts, there is another issue related to the way intelligence and knowledge is distributed between math fonts and TEX itself. As was mentioned before, TEX uses so-called math codes to represent all kinds of math symbols, encoding a type, a family and a code position in a 4-digit hexadecimal number. Depending on the type, however, this might not be everything to it, as further information might also be hidden in the font metrics of math fonts.

While ordinary symbols are represented by a single glyph in the output encoding, big operators usually come in two sizes known as textstyle and displaystyle. However, TEX's macro processor only uses a single math code (and hence, only a single code position) to represent the smaller version of a big operator, while it is left to the font metrics of the relevant math font to establish a link to the bigger version through a so-called charlist in the font metrics. (This kind of font information is, of course, also accessible to TEX's typesetting engine, but not to the macro processor.)

In a similar way, the big versions of delimiters and radicals also rely on the font metrics to establish a chain of pointers to increasingly bigger versions of the same glyph linked through a charlist. Additionally, the last entry of the chain might represent an entry point to a so-called extensible recipe, referencing the various building blocks needed to construct an arbitrarily large version of the requested symbol.

What is extremely confusing about this, is that the code positions used to access extensible recipes could be completely unrelated to the actual content of these slots. In some cases, they might be chosen carefully, so that the slots used as entry points are the same as those containing the relevant building blocks. In other cases, however, an entry point might be chosen simply because it isn't already used for anything else, but it might actually refer to glyphs taken from completely different slots.

5. Limitations and missing features of the math typesetting engine

So far, we have looked at some specific problems that are often brought up when discussing the difficulties of implementing new math fonts for TEX. While TEX works perfectly well as it is currently implemented, some of these very peculiar features may well be considered examples of bad or ugly design that are worth reconsidering. Apart from that, there are also some limitations as to what TEX's math typesetting engine can do and what it can't do. Therefore, there is also some food for thought regarding additional features that might be worth adding in a successor to TEX.

5.1. Size scaling and extra sizes in Russian typographical traditions

As explained in detail in Appendix G of *The TEXbook*, the functionality of TEX's math typesetting engine is based on a relatively small number of basic operations, such as attaching subscripts and superscripts, applying and centering math accents, building fractions, setting big operators and attaching limits, etc. In these basic operations, TEX relies on some underlying concepts of size, such as that there are four basic styles known as displaystyle, textstyle, scriptstyle and scriptscriptstyle, which are chosen according to built-in typesetting rules that can't be changed.

As was pointed out in [17], however, these built-in typesetting rules and the underlying concepts of size might not really be sufficient to cover everything needed when it comes to dealing with specific requirements for traditional Russian math typesetting, which has quite different rules than what is built into TEX.

While TEX only supports two sizes of big operators in textstyle and displaystyle, Russian typography requires an additional bigger version (as well as an extensible version of a straight integral) for use with really big expressions. Similarly, while TEX essentially uses only three sizes to go from textstyle to scriptstyle and scriptscriptstyle in numerators and denominators of fractions or in subscripts and superscripts, Russian typography calls for another intermediate step, making it necessary to have a real distinction between the font sizes used in displaystyle and in textstyle.

5.2. Extensible wide accents and over- and underbraces

While changes to fundamental concepts such as the range of sizes in math mode would have far-reaching consequences that are very difficult to assess and to decide upon, there are other potentially interesting features that might be easier to implement, even within the framework of the existing tfm file format.

One such example would be extensible versions of wide accents, which might also be used to implement over- and underbraces in a more natural way. The reason why this would be possible is simply that the tfm file format supports charlist entries and extensible recipes for any glyph. It only depends on the context whether or not these items are looked at and taken into account by TEX. In the case of delimiters and radicals, TeX supports a series of increasingly bigger versions linked through a charlist as well as an extensible recipe for a vertically extensible version. In the case of wide accents, however, TeX only supports a series of increasingly wider versions linked through a charlist, but no extensible recipe for a horizontally extensible version, even if the font metrics would support that.

Given a new mechanism for horizontally extensible objects similar to the existing mechanism for vertically extensible delimiters, it would also be possible to reimplement over- and underbraces in a more natural way, without having to rely on complicated macros for that purpose. (The font set distributed with *Mathematica* already contains glyphs for over- and underbraces in several sizes as well as the building blocks for extensible versions. Moreover, the *Mathematica* font set also contains similar glyphs for horizontally extensible versions of parentheses, square brackets and angle brackets, which don't exist in any other font set.)

5.3. Under accents, left subscripts and superscripts

Two other examples of potentially interesting new features would be mechanisms for under accents and for left subscripts and superscripts. While support for under accents might be feasible to implement given that over accents are a special type of node in TeX's internal data structures anyway, adding support for left subscripts and superscripts would certainly be more complicated, considering that right subscripts and superscripts are an inherent feature of all types of math nodes.

As for an implementation of under accents in the framework of the existing tfm file format, it would probably be necessary to resort to another cumbersome workaround similar to the \skewchar mechanism in order to store the necessary offset information. A macro solution for under accents that uses reversed \skewchar kern pairs has already been developed in the context of experimental new math font encodings [2].

6. Summary and Conclusions

6.1. What are the reasons for all these problems?

It is pretty obvious that most of the problems of math fonts discussed in this paper can be traced back to the time when TeX was developed more than twenty years ago. Given the scarcity and cost of computing power, memory and disk space at that time (in the late 1970s and early 1980s), it is no surprise that file formats such as tfm files for font metrics were designed to be compact and efficient, providing only a limited number of fields per glyph and a limited number of entries in lookup tables.

Based on such a framework, compromises and workarounds such as overloading some fields in math fonts to store additional information were unavoidable, even though such hacks damaged the clarity of design and eventually lead to other problems, requiring even further hacks to deal with the consequences (such as the \skewchar mechanism to compensate for the fact that the tfm width didn't represent the real glyph width). In view of this, it is no surprise that overcoming limitations (such as being limited to 16 math families or 16 tfm heights and depths) is the highest priority on the wish list before cleaning up other problems or adding new features.

6.2. What's good, what's bad, what's ugly?

Speaking of good, bad and ugly bits, the conference motto suggests: *"First of all, keep up the good bits and extend them if possible. Analyze the ugly bits, learn from them, and find easy and generic ways to get around them. Finally, find the bad bits and eradicate them!"* By these standards most of the problematic features discussed in this paper can probably be classified as ugly bits, with very few exceptions that might also be considered bad bits, whereas some (but not all) of the suggested new features could be summarized as extending the good bits.

As for extending the good bits, adding extensible versions of wide accents or support for under accents might be feasible examples, that could be implemented relatively easily, whereas other suggested new features such as adding support for left subscripts and superscripts or introducing additional sizes might have far-reaching consequences that should be considered with care, so as not to introduce new problems.

As for eradicating the bad bits, reconsidering the algorithm for typesetting radicals might be a high priority item on the wish list. As suggested by [13], using a repeated glyph for the rule part instead of a horizontal rule whose height depends on the glyph metrics might be a feasible solution for a better implementation.

As for learning from the ugly bits and finding better ways to get around them, starting over with a completely new font metrics format as suggested in [15] to overcome the current limitations would certainly help to avoid most of the remaining problems. Given that compactness of file formats and efficiency of store are no longer real issues with modern computers, it would be no problem to use a human-readable verbose file format and to extend the font metrics by any number of additional fields as needed to convey additional information. This way, many problems caused by overloading certain fields of the glyph metrics or resorting to workarounds such as the \skewchar mechanism could all be avoided. Considering that, there is hope that dealing with math fonts could eventually become much easier than it is today!!!

References

[1] Alan Jeffrey. Math font encodings: A workshop summary. *TUGboat*, 14(3):293–295, 1993.

[2] Matthias Clasen and Ulrik Vieth. Towards a new Math Font Encoding for (la)tex. *Cahiers GUTenberg*, 28–29:94–121, 1998. Proceedings of the 10th European TEX Conference, St. Malo, France, March 1998.

[3] Ulrik Vieth. Summary of math font-related activities at EuroTEX '98. *MAPS*, 20:243–246, 1998.

[4] Taco Hoekwater. An Extended Maths Font Set for Processing MathML. In *EuroTEX'99 Proceedings*, pages 155–164, 1999. Proceedings of the 11th European TEX Conference, Heidelberg, Germany, September 1999.

[5] Patrick Ion. MathML: A key to math on the Web. *TUGboat*, 20(3):167–175, 1999.

[6] Barbara Beeton. Unicode and math, a combination whose time has come – Finally! *TUGboat*, 21(3):176–186, 2000.

[7] Alan Jeffrey. PostScript font support in LATEX 2_ε. *TUGboat*, 15(3):263–268, 1994.

[8] Thierry Bouche. Diversity in math fonts. *TUGboat*, 19(2):121–134, 1998.

[9] Alan Hoenig. Alternatives to Computer Modern Mathematics. *TUGboat*, 19(2):176–187, 1998.

[10] Alan Hoenig. Alternatives to Computer Modern Mathematics. *TUGboat*, 20(3):282–289, 1999.

[11] Richard J. Kinch. Belleek: A call for METAFONT revival. *TUGboat*, 19(3):244–249, 1998.

[12] Berthold Horn. Where are the math fonts? *TUGboat*, 14(3):282–284, 1993.

[13] Matthias Clasen. Ideas for improvements to TEX's math typesetting in ϵ-tex/NTS. unpublished paper, available from http://www.latex-project.org/papers/etex-math-notes.pdf, 1998.

[14] David Carlisle. Notes on the Oldenburg ϵ-tex/LATEX3/context meeting. unpublished paper, available form http://www.latex-project.org/papers/etex-meeting-notes.pdf, 1998.

[15] NTG TEX Future Working Group. TEX in 2003: Part I: Introduction and Views on Current Work. *TUGboat*, 19(3):323–329, 1998.

[16] Lars Hellström. Writing ETX format font encoding specifications. unpublished paper, available from http://abel.math.umu.se/~lars/encodings/encspecs.tex, 2001.

[17] Alexander Berdnikov. Russian Typographical Traditions in Mathematical Literature. In *EuroTEX'99 Proceedings*, pages 211–225, 1999. Proceedings of the 11th European TEX Conference, Heidelberg, Germany, September 1999.

◇ Ulrik Vieth
ulrik.vieth@tesionmail.de
Vaihinger Str. 69,
D-70567 Stuttgart, Germany