Math typesetting in TEX:

The good, the bad, the ugly

Ulrik Vieth

EuroTEX 2001 · Kerkrade

Overview

- Introduction: What's the state of math typesetting with TEX?
  - Some good news
  - Some bad news
- Overview: How things work: Some technical background
  - What goes on in text mode ...
  - What goes on in math mode ...
- Analysis: What's wrong with TEX's math typesetting engine?
  - What are some specific problems?
  - What are the limitations and shortcomings?
  - What are missing features?
- Discussion:
  - What's good, what's bad, what's ugly?
  - What could be done to improve things?
  - What has already been done?

Math typesetting with TEX: Some good news

- TEX is still very good at typesetting math — even after 20 years of age
- TEX is also very good at typesetting text, but other systems are catching up
- TEX is still at its best in the domain of math typesetting
- Math typesetting has always played a central role in TEX
- Math typesetting has often been neglected by word processors
- Math typesetting has been a key feature to TEX's success
- TEX cannot handle everything by itself when typesetting math, but ...
  - experienced users can produce publication quality with manual tweaking
  - average users can produce "good enough" quality with default settings

Math typesetting with TEX: Some bad news

- While the output is good, the internals are far from perfect
- Math typesetting is an inherently difficult topic
- TEX's math typesetting engine is extremely complicated (see Appendix G)
- TEX's math typesetting engine has some very peculiar features
- TEX's math typesetting engine has some limitations and shortcomings
- TEX's math typesetting engine makes assumptions about math fonts
- Implementing math fonts for TEX alone is extremely difficult
- Implementing math fonts for TEX *and* other systems is almost impossible

Technical background

- **What goes on in text mode ...**
  - characters, codes and fonts
  - input to output translation
  - font handling
- **What goes on in math mode ...**
  - math symbols, math codes and families
  - input to output translation
  - interaction between typesetting engine and fonts

What goes on in text mode ...

- Essentially, when typesetting text, TEX does the following:
  - convert input codes to output codes using the current font
  - assemble sequences of boxes (glyphs) and glue (whitespace)
  - break paragraphs into lines and lines into pages
- Input codes:
  - keyboard characters (7-bit ASCII and ^^ notation), `\char` tokens
  - expansion of macros (`\chardef`), active characters, input ligatures
- Output codes:
  - code positions in the currently selected font, using font-specific encoding
- Font handling:
  - Whatever the macro package, fonts are eventually loaded by primitives
  - Loading a font: `\font\foo=<TFM name>`, selecting a font: `\foo`
  - Fonts represent specific font shapes of a specific family, size, encoding

What goes on in math mode (I)

- When it comes to typesetting math, many things are different:
  - TEX doesn't use fonts directly, it uses math families instead
  - there can be hundreds of fonts, but only 16 math families
  - each math family represents a set of fonts at 3 different styles
  - math symbols are represented by math codes (4-digit hex-numbers)
  - math codes encode type of symbol, family and character code
- Input codes:
  - keyboard characters translated through `\mathcode` assignments
  - expansion of macros (`\mathchardef`), active characters
- Internal representation:
  - math symbols, represented by math codes (4-digit hex-numbers)
- Output codes:
  - code positions in a font of a given math family and style
  - codes may be translated further through **charlist**s and **extensible** recipes

What goes on in math mode (II)

- When typesetting math, TEX does the following:
  - convert keyboard characters or math symbols to math codes
  - select the proper style based on the context and type of symbol
  - apply the proper amount of spacing based on the type of symbol
  - select the proper version of symbols for big operators, delimiters, radicals
  - resolve math codes to output codes from specific math fonts
  - assemble math list from math atoms, convert math list to horizontal list
- Interaction between math typesetting engine and math fonts:
  - information and intelligence is distributed between engine and fonts
  - typesetting rules are hard-wired in the math typesetting engine
  - certain parameters are set up in the format file
  - certain parameters are specified by `\fontdimen` parameters
  - `\fontdimen`s of families 2 and 3 apply to the whole set of math fonts

Specific problems of TEX's math fonts

- Glyph metrics of ordinary symbols
- Accent placement, `\skewchar` mechanism
- Glyph metrics of big symbols in math extension fonts
- Access to big symbols in math extension fonts
- Other problems (in brief)

Glyph metrics of ordinary math symbols

- for text fonts: 4 fields of per-glyph info in TFM files
  - width, height, depth, italic correction
- for math fonts: same fields, different interpretation
  - TFM width: offset from left for subscript position
  - TFM italic correction: offset for superscript position
  - glyph width = TFM width + TFM italic correction
- Problems:
  - when setting up math fonts, glyph metrics have to be adjusted
  - italic letters from text fonts cannot be used as given
  - lots of fine-tuning needed to arrive at optimal values
- Solution:
  - revise TFM file format, introduce additional fields

Placement of math accents, \skewchar mechanism

- for text fonts: standard mechanism for accent placement
  - accent glyphs are assumed to fit on top of lowercase glyphs
  - accents are centered and shifted up or down as needed
  - accents are shifted to the right, depending on font slant
- for math fonts:
  - glyph width differs from TFM width: standard mechanism doesn't work
  - \skewchar mechanism: one glyph per font designated as \skewchar
  - pseudo kern-pairs are used to store shift amounts for accents
- Problems:
  - when setting up math fonts, accent placement has to be adjusted
  - lots of fine-tuning needed to arrive at optimal values
- Solution:
  - revise TFM file format, introduce additional fields

Glyph metrics of big symbols in math extension fonts

- glyph metrics of big symbols have unusual properties:
  - big operators, delimiters and radicals all hang below the baseline
  - big symbols are not centered on the math axis
- What are the reasons for this?
  - radicals are constructed using rules instead of glyphs for the rule part
  - height of radical glyph determines rule thickness of horizontal rule
  - TFM file format is limited to only 16 different heights and depths
  - delimiters have to be placed in the same position as radicals
- Problems:
  - math extension fonts are specific to TEX, not usable for other systems
  - fonts for use with TEX *and* other systems need two sets of glyphs
- Solution:
  - revise algorithm for radicals, maybe using glyphs instead of rules
  - revise TFM file format, remove limitations (16 different heights and depths)

Access to big symbols in math extension fonts

- ordinary math symbols:
  - a single math code encodes type, family, code position
  - each symbol is represented by a single math code
- What's different for big symbols?
  - big delimiters and radicals have a small and a big version
  - big version may be followed by a sequence of bigger versions (**charlist**)
  - biggest version may be followed by an extensible recipe (**extensible**)
  - extensible recipe specifies building blocks (top, bottom, middle, etc.)
- Problems:
  - only the entry point to a **charlist** is represented by a math code
  - entry points to **extensible** may be unrelated to building blocks
- Confusing, isn't it?

Other problems of TEX's math fonts (in brief)

- Boxing and unboxing subformulas:
  - Subformulas are always set to their natural width
  - Overall formula is subject to glue setting (stretch / shrink)
  - Glue is distributed unevenly between subformulas and top-level elements
- Semantics of `\abovedisplayshortskip`:
  - `\abovedisplayshortskip` depends on short line before display
  - `\belowdisplayshortskip` is coupled to `\abovedisplayshortskip`
  - `\belowdisplayshortskip` cannot look ahead into next paragraph
  - decision may be wrong if you have a short / long or long / short

Limitations of TEX's math typesetting engine

- More flexible size-scaling and extra sizes
- Extensible wide accents or over- and underbraces
- Under accents, left subscripts and superscripts
- Other features (in brief)

Size scaling and additional sizes

- TEX's math typesetting engine is based on concepts of size:
  - two sizes of big operators (textstyle, displaystyle)
  - three size of ordinary symbols (textstyle, scriptstyle, scriptscriptstyle)
  - built-in rules for choosing the size in fractions or indices
- Requirements for Russian typography go beyond the default:
  - three sizes of big operators, including a new extra-large version
  - four size of ordinary symbols (displaystyle bigger than textstyle)
  - different built-in rules for choosing the size in fractions or indices
- Solutions:
  - Sorry! This kind of change would go too far.

Extensible wide accents

- present situation:
  - delimiters and radicals have big and extensible versions
  - math accents have wide, but no extensible versions
  - both use the same TFM mechanisms (charlist, extensible)
  - it only depends on the type of symbol, whether TEX uses the TFM info
- Solutions:
  - implmentation should be straight-forward
  - no changes to TFM file format needed
  - could be useful to redefine over- and underbraces in a better way

Under accents, left subscripts and superscripts

- present situation:
  - TEX only provides over accents and right subscripts and superscripts
  - under accents are rare, but do exists (presently implemented by macros)
  - left sub- and superscripts can be attached to the right of an empty group
- Solutions:
  - under accents can be implemented by macros, but very messy
  - under accents might require reverse `\skewchar` kern-pairs
  - under accents would be new type of math nodes
  - implementation should be be straight-forward
  - subscripts and superscripts fields are attached to each math node
  - implementation would certainly be more involved
  - semantics need to be clarified first (special catcodes?)

Other suggested features (in brief)

- Access to hard-wired information:
  - spacing table between types of symbols
  - kerning table (Ord-Ord, Ord-Open, Open-Ord, Ord-Close)
- Kerning:
  - kerning is not possible across different fonts
  - upright and italic Greek and Latin alphabets are in different fonts
  - need for bigger fonts to allow kerning between all letters
  - need for new encodings (all alphabetic symbols in one font)
- Ligatures:
  - ligatures are not possible across different fonts
  - input ligatures, e.g. >>, >= can be implemented by macros
- More information:
  - `\mathstyle` to report style (SS, SS',S, S', D, D', T, T')
  - `\ifcramped` to report whether or not cramped style

Discussion: What's good, what's bad, what's ugly?

- Conference motto:
  - Keep up the good bits and extend them if possible!
  - Analyze the ugly bits and find ways to get around them!
  - Find the bad bits and eradicate them!
- Extending the good bits:
  - extensible wide accents, primitive support for under accents
  - better implementation of over- and underbraces
  - left subscripts and superscripts (???), more flexible size scaling (???)
- Improving the ugly bits:
  - get rid of limitations (16 families, 256 symbols, 16 TFM heights/depths)
  - extend TFM file format (avoid overloading of TFM fields and `\fontdimens`)
  - solve problems about interpretation of glyph metrics and accent placement
- Eradicate the bad bits:
  - re-implement `\radical` using new algorithm
  - solve problems about unusual glyph metrics of radicals and delimiters