Math in LuaTEX 0.40

Abstract
The math machinery in luaTEX has been completely overhauled in version 0.40. The handling of mathematics in luaTEX has been extended quite a bit compared to how TEX82 (and therefore pdfTEX) handles math. First, luaTEX adds primitives and extends some others so that Unicode input can be used easily. Second, all of TEX82's internal special values (for example for operator spacing) have been made accessible and changeable via control sequences. Third, there are extensions that make it easier to use OpenType math fonts. And finally, there are some extensions that have been proposed in the past that are now added to the engine.

Introduction

We (the luaTEX team) started thinking about OpenType Math support almost immediately after Cambria Math was released, but it took us more than a year to get around to actually writing the implementation. The extensions to the math engine are not complete yet, but there is now enough stuff worthy of publication. This article tries to give a complete overview of all work done so far, but that also means that it is sketchy on details in some places. For the definitive reference, you should read the Math chapter in the luaTEX reference manual.

Pre-existing math primitives

TEX82

Besides the math primitives found in TEX82, luaTEX has support for the extended math primitives that were added by Aleph and XeTEX.

The TEX82 primitives have been left untouched, except for the fact that when there is a character number needed on the left side of the equation sign (for \mathcode and \delcode), this number can make use of the full Unicode range.

Typical example code of TEX82 primitives:

```
\mathcode`\+="202B
\delcode`\(="028300
\mathchardef\alpha="010B
\mathchar"1270
\mathaccent"017E
\delimiter"3222378
\radical"270370
```

Aleph

The Aleph math primitives use a syntax that is a fairly straightforward extension of the TEX82 primitives. The difference is that everything has been extended to allow for 16-bit character codes and 256 families. For \odelcode, \odelimiter, and \oradical, this forced the syntax into using two integers for the value to be assigned (because more than 31 bits are needed) but other than that every extension is quite straightforward.

Once again, luaTEX extends the character code on the left side of the equals sign for \omathcode and \odelcode to the full Unicode range.

Typical example code of Aleph primitives:

```
\omathcode`\+="200002B
\odelcode`\(="000028 "030000
\omathchardef\alpha="001000B
\omathchar"1020070
\omathaccent"001007E
\odelimiter"3020022 "030078
\oradical"020070 "030070
```

XeTEX

The XeTEX primitives need to pack even more information: like Aleph, XeTEX has 256 math families, but each of those is encoded using the full Unicode range. This makes it hard to come up with a nice hexadecimal notation, so instead the values are split up into their class, family, slot segments, for example:

```
\def\overbrace {\Umathaccent 0 1 "23DE }
```

When a math class is required, this is given by the first integer, which ranges from 0 to 7. The next integer is the family number and ranges from 0 to 255. The last integer is the Unicode code point, which ranges from 0 to hexadecimal 0x10FFFF (1,114,111 in decimal).

There are always just two or three integers needed, because XeTEX never bothers to list ‘small’ and ‘large’ versions of delimiters. The use of large vs. small items is controlled via OpenType font parameters.

LuaTEX includes primitives that are fully compatible with their XƎTEX counterparts except for their names. Where XƎTEX uses the `\XeTeX` prefix, luaTEX uses `\U`.

Typical example code of XƎTEX-compatible primitives:

```
\Umathcode`\+="2 "0 "2B
\Udelcode`\(= "0 "28
\Umathchardef\alpha="0 "1 "B
\Umathchar "1 "2 "70
\Umathaccent "0 "1 "7E
\Udelimiter "3 "2 "22
\Uradical "2 "70
```

For the sake of completeness, the 'packed' XƎTEX primitives `\Umathcharnum`, `\Umathcodenum` and `\Udelcodenum` are also provided, but their use is discouraged.

General new math extensions

Cramped math styles

TEX's math engine has four main math styles: display style, text style, script style, and scriptscript style. Each of those four main styles can also appear in a 'cramped' form that is suitable for use in situations where something lives on top of the current sub-formula (like in the denominator part of a fraction). This makes for a total of eight styles. In TEX82, it is possible to force a particular main math style by using one of these primitives:

```
\displaystyle
\textstyle
\scriptstyle
\scriptscriptstyle
```

However, until now it was not possible to explicitly switch to one of the cramped modes. For this, luaTEX adds the following four new primitives:

```
\crampeddisplaystyle
\crampedtextstyle
\crampedscriptstyle
\crampedscriptscriptstyle
```

Math characters in text mode

LuaTEX allows `\mathchar`, `\omathchar`, and `\Umathchar` and control sequences that are the result of `\mathchardef`, `\omathchardef`, or `\Umathchardef` outside math mode.

When luaTEX sees an object like this, it uses the `\textfont` from the requested math family to produce a normal glyph node.

For example, assume that `\alpha` is defined as before and that `\omega` is defined as a `\mathchardef` with value `"121`. Further assume that `\textfont1` is `\teni` (as it is in the plain macros). Under these conditions,

```
From \alpha\ to \omega.
```

and

```
From {\teni\char"B} to {\teni \char"21}.
```

are equivalent. Both will produce:

From α to ω.

Querying the math style

The new expandable primitive `\mathstyle` returns a value between 0 and 7 (in math mode), or −1 (all other modes). The returned number represents the current math style value.

Higher numbers represent smaller styles: 0 stands for `\displaystyle`, 1 for `\crampeddisplaystyle`, and 7 for `\crampedscriptscriptstyle`.

Using these new primitives, you can write code like this:

```
\def\uncramped#1{{\ifcase\mathstyle
    \or \displaystyle      \or
    \or \textstyle         \or
    \or \scriptstyle       \or
    \or \scriptscriptstyle \fi  #1}}
```

or even create a fully expandable version of `\mathchoice`:

```
\def\mathchoice#1#2#3#4{{\ifcase\mathstyle
    #1\or #1\or
    #2\or #2\or
    #3\or #3\or
    #4\or #4\fi}}
```

To make it easier to test the return value of `\mathstyle`, the four old and the four new math style commands have been altered so that they can be used as numeric values for testing. This allows constructs like this:

```
\ifnum\mathstyle=\textstyle
   \message{normal text style}
\fi
```

But there is a small catch: there are a few primitives (`\over`, `\atop`, `\overwithdelims`, `\atopwithdelims`) in TEX82 where the style that will be used is not known at the start, and these commands would therefore return wrong values for `\mathstyle`.

To make it possible to get the correct math style in all cases, luaTEX introduces the new primitive `\Ustack`, that can (should) be used as a prefix for the commands given above.

```
$\Ustack { ... a ... \over ... b ... }$
```

The `\Ustack` command will make sure that `\mathstyle` is returning the correct values, even inside the `... a ...` branch. A `\Ustack` can be nested inside another, if needed.

Bottom accents

Besides the normal top accents, luaTEX also supports bottom accents in math mode. For bottom accents, there is the new primitive `\Umathbotaccent`. For combined top and bottom accents, there is `\Umathaccents`. The latter takes two math accent specifications. Like all the new primitives that actively scan for mathchars or delimititers, these use the XƎTEX-style syntax:

```
$$
\Umathbotaccent"0"0"323 A
\Umathaccents "0"0"20D7 "0"0"323 A
$$
```

$$\underset{\cdot}{A}\underset{\cdot}{\vec{A}}$$

Horizontal extenders

On top of the normal vertical extensibles, luaTEX also has support for horizontal extensibles. This is particularly useful for wide accents, as the following example shows:

```
\def\overarrow{\Umathaccent"0"0"20D7}
$$
\overarrow{a+b+c+d+e}
$$
```

$$\overrightarrow{a+b+c+d+e}$$

Note that this feature depends on support from the math font that is being used. This article is typeset using MicroSoft's Cambria Math font and that actually has this support built in, but so far none of the standard TEX fonts provide the needed information.

Math parameters

In luaTEX, the font dimension parameters that TEX82 uses in math typesetting are now accessible via primitive commands. These parameters are initialized from the math fonts, or can be set by the user via explicit commands. Each math parameter exists in eight versions that match the math styles. Re-factoring of the math engine has resulted in more parameters than were accessible before, even when taking the font dimensions of the math fonts into account.

Math parameter commands

Each of the math parameters (the full list is given in table 1 at the end of this article) can be set by an explicit command, like this:

```
\Umathquad\displaystyle=1em
```

Such settings obey grouping, but only one value can be in effect for a single formula, and that is decided upon when the closing dollar sign is read in. Here is an example:

```
\centerline{
$
\Ustack{a \over b} ×  b
$ \kern 50pt $
\Umathfractiondenomvgap \textstyle = 8pt
\Ustack{a \over b} ×  b
$}
```

$$\frac{a}{b}\times b \qquad\qquad \frac{a}{\;b\;}\times b$$

You can use `\the\Umathquad\displaystyle` if the current value is needed (for example inside a space fine-tuning macro).

Font-based Math Parameters

While it is nice to have these math parameters available for tweaking, it would be tedious to have to set each of them by hand. For this reason, luaTEX initializes (almost) all these parameters whenever you assign a font identifier to a math family. This is based either on the traditional math font dimensions in the font (for assignments to math family 2 and 3 using TFM-based fonts like `cmsy` and `cmex`), or on the named values in a 'MathConstants' table (when an OpenType math font is loaded via Lua). If there is a 'MathConstants' table, this takes precedence over font dimensions, and in that case no attention is paid to which family is being assigned to: the 'MathConstants' tables in the last assigned family sets all parameters.

The eight math parameters are typically set by using the `\textfont` value for the display and text styles (cramped and normal), `\scriptfont` for the script styles, and `\scriptscriptfont` for the scripscript styles. In table 2 these automatic mappings are shown. Besides the parameters listed in that table, luaTEX also looks at the 'space' font dimension parameter. For math fonts, this should be set to zero.

Math spacing parameters

Inter-element math spacing in TEX82 is controlled by the 8 × 8 table of spacing values that is given in Chapter 18 of the TEXbook. In luaTEX, this table has been converted into 64 primitives of the form `\Umath...spacing`, for all the paired combinations of `bin`, `rel`, `ord`, `open`, `close`, `punct`, `inner`, and op. Here is an example:

```
\centerline{$
a × b
$ \kern 50pt $
\Umathordbinspacing \textstyle = 18mu
\Umathbinordspacing \textstyle = \thickmuskip
\thickmuskip = 10mu
a × b
$}
```

$a \times b \qquad\qquad a \quad \times \quad b$

Normally, one would assign explicit `mu` dimensions to these parameters, but a special case arises when the predefined muskip registers are used. When the assignment uses `\thinmuskip`, `\medmuskip`, or `\thickmuskip`, late binding is used, so that later (re)assignments to one of these registers is taken into account.

Verbose versions of character commands

luaTEX defines six new primitives that have the same function as ^, _, $, and $$.

primitive	explanation
`\Usuperscript`	for the functionality of ^
`\Usubscript`	for the functionality of _
`\Ustartmath`	for $, outside math.
`\Ustopmath`	for $, inside inline math.
`\Ustartdisplaymath`	for $$, outside math.
`\Ustopdisplaymath`	for $$, inside display math.

The `\Ustopmath` and `\Ustopdisplaymath` primitives check if the current math mode is the correct one (inline vs. displayed), but you can freely intermix the four mathon/mathoff commands with explicit dollar sign(s).

Lua math extensions

Setting and getting math parameters

The lua functions `tex.setmath()` and `tex.getmath()` can be used to get or set the internal math parameters.

To set a math parameter, use `tex.setmath()`:

```
tex.setmath(<string> n, <string> t, <number> n)
```

or

```
tex.setmath('global',
           <string> n, <string> t, <number> n)
```

In an attempt to cut down the verbosity level, the first string is the parameter name minus the leading 'Umath', and the second string is the style name minus the trailing 'style', for example:

```
tex.setmath('fractiondenomvgap','text',8*65536)
```

An optional first parameter can be given with the explicit string `'global'`, which indicates a global assignment. For now, you cannot use Lua for the math object spacing parameters (because there is no read interface for 'mu' lengths defined yet).

Querying a math parameter uses the inverse function `tex.getmath()`:

```
<number> n = tex.getmath(<string> n, <string> t)
```

which should not need further explanation.

Attributes in math mode

Starting with luaTEX 0.40, node attributes are now remembered in math mode even after the conversion from math back to the horizontal list that is eventually added to the typeset paragraph. New nodes that are created in this process (like the horizontal rule in a fraction) inherit their attributes from the most logical parent node.

The 'mlist_to_hlist' callback

A simple callback is offered that can be used to alter last-minute things in the math node list. When you use this callback, you have to run the math to hlist conversion process yourself. To make this easier, there is a builtin function that does exactly what luaTEX would have done if there was no callback set.

First, here is the syntax diagram for the callback:

```
function(<node> head,
         <string> displaytype,
         <boolean> need_penalties)
    return <node> newhead
end
```

The returned node has to be the head of the list that will be added to the vertical or horizontal list, the string `displaytype` argument is either 'text' or 'display' depending on the current math mode, the boolean `need_penalties` argument is `true` if penalties have to be inserted into the generated hlist, `false` otherwise.

If all you want to do is alter a few small things, than the easiest approach is to make those alterations first, and then call the following helper function:

```
<node> h = node.mlist_to_hlist(
              <node> n,
              <string> displaytype,
              <boolean> penalties )
```

This runs the internal mlist to hlist conversion, converting the math list in `n` into the horizontal list `h`. The interface is exactly the same as for the callback `mlist_to_hlist`, so that a simple working callback is this one:

```
callback.register ('mlist_to_hlist',
  function (h,d,n)
    return node.mlist_to_hlist(h,d,n)
  end )
```

OpenType Math features

As explained by Ulrik Vieth's articles on the subject, there is much more to 'OpenType Math' than just being able to handle Unicode input characters. A number of extensions have been made to luaTEX to handle specific features of OpenType Math.

OpenType font metrics

First, lets talk about OpenType font metrics. OpenType fonts in luaTEX are always loaded via lua code in the `define_font` callback, and OpenType Math fonts are no exception.

The lua function `fontloader.to_table()` outputs the OpenType Math information in two parts: there is a global part where the Math Constants are listed, and a per-glyph local part with data like italic correction and extensible recipe structures.

The global part looks like this:

```
["math"]={
 ["AxisHeight"]=585,
 ...
 ["FractionDenominatorDisplayStyleGapMin"]=260,
 ["FractionDenominatorGapMin"]=133,
 ["FractionDenominatorShiftDown"]=1030,
 ["FractionNumeratorDisplayStyleGapMin"]=260,
 ["FractionNumeratorDisplayStyleShiftUp"]=1550,
 ["FractionNumeratorGapMin"]=133,
 ...
 ["ScriptPercentScaleDown"]=73,
 ["ScriptScriptPercentScaleDown"]=60,
 ...
```

The full list is longer than this of course; all the math constants are listed in that table. Except for a few cases with the word 'Percent' in the name, values are expressed in design units, and these have to be converted by Lua code into scaled points before being passed back to luaTEX (as is the case for all font dimensions).

The example above is taken from Cambria Math which is a Truetype format font with 2048 design units per em, so the actual value of 'AxisHeight' if the font is loaded at 10pt would be

$$585/2048 * 10\text{pt} = 187\,200\text{sp}$$

The local part is easier to explain in two steps, because not all glyphs have the same set of extended information. The first example shows the relevant part of the data for the Unicode character 'MATHEMATICAL ITALIC SMALL F', f:

```
{
  ["name"]="u1D453",
  ["italic_correction"]=60,
  ["mathkern"]={
   ["bottom_right"]={
    {
      ["height"]=420,
      ["kern"]=-400,
    },
    {
      ["height"]=720,
      ["kern"]=-320,
    },
    {
      ["height"]=1020,
      ["kern"]=0,
    },
   },
   ["bottom_left"]={ ... }
   ["top_right"]={ ... }
  },
  ["top_accent"]=840,
  ...
},
```

As you can see, it has an italic correction of 60 design units, it has an entry `top_accent` that is used for the placement of math accents on top of the glyph, and it has a `mathkern` subtable. The `mathkern` table is used for super- and subscript placement: it can define kerning corrections for each of the four corners of the glyph. Any of those can be missing, in which case no correction is needed (this is the case for the `top_left` side of this glyph).

The next example shows part of the metric data for `SQUARE ROOT`, the extensible character that represents the root sign, √:

```
{
  ["name"]="radical",
  ["vert_variants"]={
   ["italic_correction"]=0,
   ["parts"]={
    {
      ["component"]="uni23B7",
      ["advance"]=2743,
      ["end"]=2500,
    },
    {
      ["component"]="uni20D3",
      ["advance"]=1211,
      ["end"]=1150,
      ["extender"]=1,
      ["start"]=1150,
    },
    {
      ["component"]="radical.top",
      ["advance"]=1211,
      ["start"]=600,
     }
    },
   ["variants"]="radical radical.vsize1 \
                radical.vsize2 radical.vsize3\
                radical.vsize4 radical.vsize5"
  }
  ...
},
```

This glyph does not have either of the `mathkern` and `top_accent` extended information items that were present in the previous example, but it does have a `vert_variants` subtable. This table contains information for extensible recipes, split into three parts:

- the `variants` string gives a sequence of versions of this glyph with ever increasing size,
- the `parts` table lists the extensible parts,
- the `italic_correction` gives the italic correction that is needed for a glyph constructed from such parts.

The actual metric details will be explained later.

OpenType Math family sizes and ssty

When using OpenType Math fonts, it is important to load the `\scriptfont` and `\scriptscriptfont` at the sizes that are requested by the font designer (via `ScriptPercentScaleDown` and `ScriptScriptPercentScaleDown`).

If the font provides an `ssty` feature, then it is advisable to enable that feature (with values `ssty=1` for script size and `ssty=2` for script script size). The difference between doing so and simply loading the Cambria Math font at the 'normal' values of 70% and 50% can be seen in this example:

x^{x^x}	x^{x^x}
cambria	cambria
7pt/5pt	73%/60%+ssty

Extra Math Parameters

OpenType Math fonts have a few extra parameters compared to traditional TFM-based math fonts, and the effects of those extra parameters can be quite noticeable. The example below shows the difference between using the actual `\Umathfractiondenomvgap` and `\Umathfractionnumvgap` from Cambria Math versus the hardwired TEX82 value of three times `default_rule_thickness`:

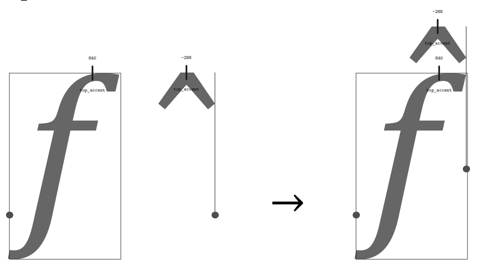
$\frac{p_p}{b^b}$	$\frac{p_p}{b^b}$
TEX82, 3*rule thickness	luaTEX, from font

Math accent placements

When a math (top) accent has to be placed and the accentee is a character that has a non-zero `top_accent` value, then this value will be used to place the accent instead of the `\skewchar` kern used by TEX82.

The `top_accent` value represents a vertical line somewhere in the accentee. The accent will be shifted horizontally such that its own `top_accent` line coincides with the one from the accentee. If the `top_accent` value of the accent is zero, then half the width of the accent followed by its italic correction is used instead.

In a picture, it looks like this:



The vertical placement of a top accent depends on the `x_height` of the font of the accentee (as explained in the TEXbook), but if that value turns out to be zero and the font has a 'MathConstants' table, then `AccentBaseHeight` is used instead.

If a math bottom accent has to be placed, the `bot_accent` value is checked instead of `top_accent`. Because bottom accents do not exist in TEX82, the `\skewchar` kern is ignored.

The vertical placement of a bottom accent is straight below the accentee, no correction takes place.

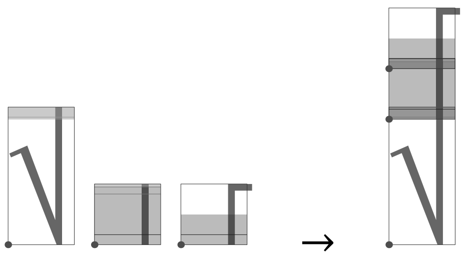
Also, remember that luaTEX has horizontal extensibles, and when present, these will be used by the accent placement primitives to build up longer versions when that is needed.

Overlapping extensibles

In TFM based fonts, extensible bits of glyphs are placed butt to butt, which normally works fine when printing, but often creates problems with PDF previewers. If you are looking at this article in PDF format, you will likely see small gaps appearing in the left side of the following example:

... but not on the right side, because the right side uses OpenType extensible recipes where a certain amount of overlap is built in.

Recall the `vert_variants` metrics representation that was listed earlier. Each of the `parts` had an `advance` key, but also type `start` and/or `end`. These latter two are combined with the 'MathConstants' value `MinConnectorOverlap` to define overlap zones. Once again here is an image to demonstrate the effect (the actual algorithm is documented in the OpenType Math specification, which is followed by luaTEX).



Extensible big operators

In OpenType Math (and therefore also in luaTEX), big operators can come in more than just the two sizes provided by TEX82. Big operators can even be build up from extensible parts.

Normally, the OpenType font designer decides the size that is used in display mode via the 'MathConstants' table, but it can be fun to change the used value manually. For example:

```
\Umathoperatorsize\displaystyle = 15pt
$$\sum_{k=2}^4 k^2 = 2^2 + 3^2 + 4^2 = 29$$
\Umathoperatorsize\displaystyle = 55pt
$$\sum_{k=2}^4 k^2 = 2^2 + 3^2 + 4^2 = 29$$
```

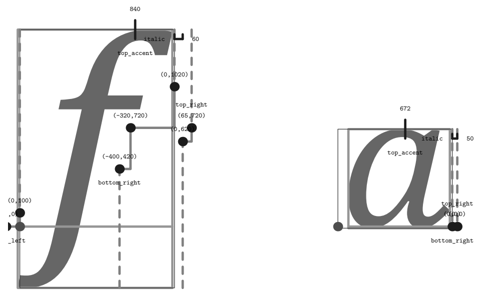
$$\sum_{k=2}^4 k^2 = 2^2 + 3^2 + 4^2 = 29$$

$$\sum_{k=2}^4 k^2 = 2^2 + 3^2 + 4^2 = 29$$

Script placements

As seen earlier, the character entries in an OpenType Math font can have a 'mathkern' table.

The 'mathkern value' at a specific height is the kern value that is specified by the next higher height and kern pair, or the highest one in the character (if there is no value high enough in the character), or simply zero (if the character has no mathkern pairs at all).



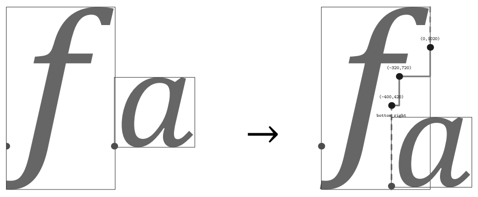
When a super- or subscript has to be placed next to a math item, luaTEX checks whether the super- or subscript and the nucleus are both simple character items. If they are, and if the fonts of both character items are OpenType fonts (as opposed to legacy TEX fonts), then luaTEX will use the OpenTypeMath algorithm for deciding on the horizontal placement of the super- or subscript. This works as follows:

- The vertical position of the script is calculated.
- The default horizontal position is flat next to the base character.
- For superscripts, the italic correction of the base character is added.
- For a superscript, two vertical values are calculated: the bottom of the script (after shifting up),

and the top of the base. For a subscript, the two values are the top of the (shifted down) script, and the bottom of the base.

- For each of these two locations, luaTEX:
  - finds out the 'mathkern value' at this height for the base (for a subscript placement, this is the `bottom_right` corner, for a superscript placement the `top_right` corner)
  - finds out the 'mathkern value' at this height for the script (for a subscript placement, this is the `top_left` corner, for a superscript placement the `bottom_left` corner)
- The horizontal kern to be applied is the smallest of the two results from previous step.

A picture should help make this clearer:



Legends on extensible items

The new `\Uunderdelimiter` and `\Uoverdelimiter` primitives allow the placement of a subscript or superscript on an extensible item and the complementary `\Udelimiterunder` and `\Udelimiterover` primitives allow the placement of an extensible item as a subscript or superscript on a nucleus.

In all four primitives, the vertical placements are controlled by `\...bgap` and `\...vgap` parameters using a similar method as for limit placements on large operators. The superscript in `\Uoverdelimiter` is typeset in a suitable scripted style, the subscript in `\Uunderdelimiter` is cramped as well.

```
$$
A \mathrel{\Uoverdelimiter 0 "2192 {a+b}}
B \mathrel{\Uunderdelimiter 0 "2192 {a+b}} C
$$
```

$$A \xrightarrow{a+b} B \xrightarrow[a+b]{} C$$

```
$$\Udelimiterover 0 "23DE {a+b}
+ \Udelimiterunder 0 "23DF {a+b} = C $$
```

$$\overbrace{a+b} + \underbrace{a+b} = C$$

Here it is the delimiter that is typeset in a script style.

Radicals with degrees

The new primitive `\Uroot` allows the direct construction of a radicals including a degree. Its syntax is a straightforward extension of `\Uradical`:

```
\Uradical <fam> <char> <radicand>
\Uroot    <fam> <char> <degree> <radicand>
```

The placement of the degree is controlled by the math parameters `\Umathradicaldegree...`, and the degree is typeset in `\scriptscriptstyle`.

```
$$
\Uroot 0 "221A {3}{x^3+y^3}
$$
```

$$\sqrt[3]{x^3+y^3}$$

This bit of the OpenType Math specification is a literal conversion of the plain TEX macro `\root \of`, with the important difference being that it shifts the adhoc values in the plain macro to the font, so that the font designer can come up with nice looking values.

In luaTEX, this functionality could have been implemented by a macro. However, doing so felt clumsy because of the need to take care of the different sizes.

Open OpenType issues

There are a few remaining problems that have not been dealt with at the time of the writing of this article:

- It is not clear how `\atopwithdelims` and `\overwithdelims` should be implemented with OpenType Math fonts. For the moment, it is best to avoid using these primitives.
- It is unclear whether stretch stacks (the four primitives explained in the 'Legends on extensible items' section) should be centered on the math axis or not.
- Some confusion remains on what the Math constant 'DelimitedSubFormulaMinHeight' is meant to represent.

Two features of OpenType Math have not been implemented yet:

- Skewed (text-style) fractions.
- Flattened accents for high characters.

These, as well as some other math extensions, are planned for the luaTEX 0.50 release.

Taco Hoekwater

Primitive name	**Description**
`\Umathquad`	the width of 18mu's
`\Umathaxis`	height of the vertical center axis of the math formula above the baseline
`\Umathoperatorsize`	minimum size of large operators in display mode
`\Umathoverbarkern`	vertical clearance above the rule
`\Umathoverbarrule`	the width of the rule
`\Umathoverbarvgap`	vertical clearance below the rule
`\Umathunderbarkern`	vertical clearance below the rule
`\Umathunderbarrule`	the width of the rule
`\Umathunderbarvgap`	vertical clearance above the rule
`\Umathradicalkern`	vertical clearance above the rule
`\Umathradicalrule`	the width of the rule
`\Umathradicalvgap`	vertical clearance below the rule
`\Umathradicaldegreebefore`	the forward kern that takes place before placement of the radical degree
`\Umathradicaldegreeafter`	the backward kern that takes place after placement of the radical degree
`\Umathradicaldegreeraise`	this is the percentage of the total height and depth of the radical sign that the degree is raised by. It is expressed in `percents`, so 60% is expressed as the integer 60.
`\Umathstackvgap`	vertical clearance between the two elements in a `\atop` stack
`\Umathstacknumup`	numerator shift upward in `\atop` stack
`\Umathstackdenomdown`	denominator shift downward in `\atop` stack
`\Umathfractionrule`	the width of the rule in a `\over`
`\Umathfractionnumvgap`	vertical clearance between the numerator and the rule
`\Umathfractionnumup`	numerator shift upward in `\over`
`\Umathfractiondenomvgap`	vertical clearance between the denominator and the rule
`\Umathfractiondenomdown`	denominator shift downward in `\over`
`\Umathfractiondelsize`	minimum delimiter size for `\...withdelims`
`\Umathlimitabovevgap`	vertical clearance for limits above operators
`\Umathlimitabovebgap`	vertical baseline clearance for limits above operators
`\Umathlimitabovekern`	space reserved at the top of the limit
`\Umathlimitbelowvgap`	vertical clearance for limits below operators
`\Umathlimitbelowbgap`	vertical baseline clearance for limits below operators
`\Umathlimitbelowkern`	space reserved at the bottom of the limit
`\Umathoverdelimitervgap`	vertical clearance for limits above delimiters
`\Umathoverdelimiterbgap`	vertical baseline clearance for limits above delimiters
`\Umathunderdelimitervgap`	vertical clearance for limits below delimiters
`\Umathunderdelimiterbgap`	vertical baseline clearance for limits below delimiters
`\Umathsubshiftdrop`	subscript drop for boxes and sub-formulas
`\Umathsubshiftdown`	subscript drop for characters
`\Umathsupshiftdrop`	superscript drop (raise, actually) for boxes and sub-formulas
`\Umathsupshiftup`	superscript raise for characters
`\Umathsubsupshiftdown`	subscript drop in the presence of a superscript
`\Umathsubtopmax`	the top of standalone subscripts cannot be higher than this above the baseline
`\Umathsupbottommin`	the bottom of standalone superscripts cannot be less than this above the baseline
`\Umathsupsubbottommax`	the bottom of the superscript of a combined super- and subscript be at least as high as this above the baseline
`\Umathsubsupvgap`	vertical clearance between super- and subscript
`\Umathspaceafterscript`	additional space added after a super- or subscript
`\Umathconnectoroverlapmin`	minimum overlap between parts in an extensible recipe

Table 1 Named math parameters.

Variable	Default value (OpenType)	Default value (TFM)
\Umathaxis	AxisHeight	axis_height
\Umathoperatorsize	MinimumDisplayOperatorHeight	<not set>
\Umathfractiondelsize	0	delim1, delim2
\Umathfractiondenomdown	FractionDenominator[DisplayStyle]ShiftDown	denom1, denom2
\Umathfractiondenomvgap	FractionDenominator[DisplayStyle]GapMin	3*rule, rule
\Umathfractionnumup	FractionNumerator[DisplayStyle]ShiftUp	num1, num2
\Umathfractionnumvgap	FractionNumerator[DisplayStyle]GapMin	3*rule, rule
\Umathfractionrule	FractionRuleThickness	rule
\Umathlimitabovebgap	UpperLimitBaselineRiseMin	big_op_spacing3
\Umathlimitabovekern	0	big_op_spacing5
\Umathlimitabovevgap	UpperLimitGapMin	big_op_spacing1
\Umathlimitbelowbgap	LowerLimitBaselineDropMin	big_op_spacing4
\Umathlimitbelowkern	0	big_op_spacing5
\Umathlimitbelowvgap	LowerLimitGapMin	big_op_spacing2
\Umathoverdelimitervgap	StretchStackGapBelowMin	big_op_spacing1
\Umathoverdelimiterbgap	StretchStackTopShiftUp	big_op_spacing3
\Umathunderdelimitervgap	StretchStackGapAboveMin	big_op_spacing2
\Umathunderdelimiterbgap	StretchStackBottomShiftDown	big_op_spacing4
\Umathoverbarkern	OverbarExtraAscender	rule
\Umathoverbarrule	OverbarRuleThickness	rule
\Umathoverbarvgap	OverbarVerticalGap	3*rule
\Umathquad	<font_size(f)>	math_quad
\Umathradicalkern	RadicalExtraAscender	rule
\Umathradicalrule	RadicalRuleThickness	<not set>
\Umathradicalvgap	Radical[DisplayStyle]VerticalGap	(rule+(abs(math_x)/4)), (rule+(abs(rule)/4))
\Umathradicaldegreebefore	RadicalKernBeforeDegree	<not set>
\Umathradicaldegreeafter	RadicalKernAfterDegree	<not set>
\Umathradicaldegreeraise	RadicalDegreeBottomRaisePercent	<not set>
\Umathspaceafterscript	SpaceAfterScript	script_space
\Umathstackdenomdown	StackBottom[DisplayStyle]ShiftDown	denom1, denom2
\Umathstacknumup	StackTop[DisplayStyle]ShiftUp	num1, num3
\Umathstackvgap	Stack[DisplayStyle]GapMin	7*rule, 3*rule
\Umathsubshiftdown	SubscriptShiftDown	sub1
\Umathsubshiftdrop	SubscriptBaselineDropMin	sub_drop
\Umathsubsupshiftdown	SubscriptShiftDown[WithSuperscript]	sub2
\Umathsubtopmax	SubscriptTopMax	(abs(math_x*4)/5)
\Umathsubsupvgap	SubSuperscriptGapMin	4*rule
\Umathsupbottommin	SuperscriptBottomMin	(abs(math_x)/4)
\Umathsupshiftdrop	SuperscriptBaselineDropMax	sup_drop
\Umathsupshiftup	SuperscriptShiftUp[Cramped]	sup1, sup2, sup3
\Umathsupsubbottommax	SuperscriptBottomMaxWithSubscript	(abs(math_x*4)/5)
\Umathunderbarkern	UnderbarExtraDescender	rule
\Umathunderbarrule	UnderbarRuleThickness	rule
\Umathunderbarvgap	UnderbarVerticalGap	3*rule
\Umathconnectoroverlapmin	MinConnectorOverlap	0

Table 2 Initialization of math parameters from font information. In the last column, 'rule' stands for default_rule_thickness, and 'math_x' stands in for math_x_height. Where multiple values or square brackets are present some of the eight parameter instances are based on some values and others on other values. See the luaTEX reference manual for more detailed information.