OpenType Math Illuminated

Abstract
In recent years, we have seen the development of new TEX engines, XƎTEX and LuaTEX, adopting OpenType font technology for providing Unicode typesetting support. While there are already plenty of OpenType text fonts available for use, both from the TEX community and from commercial font suppliers, there is little support for OpenType math fonts so far. Ironically, it was left to Microsoft to develop a *de facto* standard for OpenType math font information and to provide the first reference implementation of a full-featured OpenType math font.

In order to develop the much-needed math support for Latin Modern and TEX Gyre fonts, it will be crucially important to develop a good understanding of the internals of OpenType math tables, much as it is necessary to develop a good understanding of Appendix G and TEX's `\fontdimen` parameters to develop math support for traditional TEX fonts. In this paper, we try to help improve the understanding of OpenType math internals, summarizing the parameters of OpenType math fonts as well as illustrating the similarities and differences between traditional TEX math fonts and OpenType math fonts.

Background on OpenType math

In recent years, the TEX community has been going through a phase of very significant developments. Among the most important achievements, we have seen the development of new TEX engines, XƎTEX and LuaTEX, providing support for Unicode and OpenType font technology. At about the same time we have also seen the development of new font distributions, Latin Modern and TEX Gyre, provided simultaneously in Type 1 format as a set of 8-bit font encodings as well as in OpenType format.

Together these developments have enabled TEX users to keep up with current trends in the publishing industry, providing users of the new TEX engines with a comprehensive set of free OpenType fonts and enabling them to take advantage of the many offerings by commercial font suppliers.

As far as text typesetting is concerned, support for OpenType font technology in the new TEX engines is already very advanced, supporting not only traditional typographic features of Latin alphabets, but also addressing the very complex and challenging requirements of Arabic typography.

However, when it comes to math typesetting, one of the traditional strongholds of TEX, support for Unicode and OpenType math is only just beginning to take shape.

Ironically, it was left to Microsoft to develop the first system to offer support for Unicode math. When Microsoft introduced support for math typesetting in Office 2007 [1, 2], they extended the OpenType font format and commissioned the design of Cambria Math [3] as a reference implementation of a full-featured OpenType math font.

Fortunately for us, Microsoft was smart enough to borrow from the best examples of math typesetting technology, thus many concepts of OpenType math are not only derived from the model of TEX, but also go beyond TEX and introduce extensions or generalizations of familiar concepts.

While OpenType math is officially still considered experimental, it is quickly becoming a *de facto* standard, as it has already been widely deployed to millions of installations of Microsoft Office 2007 and it is also being been adopted by other projects such as the FontForge [4] font editor or independent font designs such as Asana Math [5].

Most importantly, support for OpenType math has already been implemented or is currently being implemented in the new TEX engines, thus adopting OpenType math for the development of the much-needed Unicode math support for Latin Modern and TEX Gyre obviously seems to be most promising choice of technology.

Design and quality of math fonts

When it comes to developing math fonts, designing the glyph shapes is only part of the job. Another part, which is equally important, is to adjust the glyph metrics of individual glyphs and to set up the global parameters affecting various aspects of glyph positioning in math typesetting.

As we have discussed at previous conferences, the quality of math typesetting crucially depends on the fine-tuning of these parameters. Developing a good understanding of these parameters will therefore be-

come an important prerequisite to support the development of new math fonts.

In the case of traditional TEX math fonts, we have to deal with the many \fontdimen parameters which have been analyzed in Bogusław Jackowski's paper *Appendix G Illuminated* and a follow-up paper by the present author [6, 7].

In the case of OpenType math fonts, we need to develop a similar understanding of the various tables and parameters and how the concepts of OpenType math relate to the concepts of TEX.

Overview of the OpenType font format

The OpenType font format [8] was developed jointly by Adobe and Microsoft, based on elements of the earlier PostScript and TrueType font formats by the same vendors. The overall structure of OpenType fonts consists of a number of tables, some of which are required while others are optional [9].

In the case of OpenType math, the extension of the font format essentially consists of adding another optional table, the so-called MATH table, containing all the information related to math typesetting. Since it is an optional table, it would be interpreted only by software which knows about it (such as the new TEX engines or Microsoft Office 2007), while it would be ignored by other software.

Unlike a database table, which has a very rigid format, an OpenType font table can have a fairly complex structure, combining a variety of different kinds of information in the same table. In the case of the OpenType MATH table, we have the following kinds of information:

- a number of global parameters specific to math typesetting (similar to TEX's many \fontdimen parameters of Appendix G)
- instructions for vertical and horizontal variants and/or constructions (similar to TEX's charlists and extensible recipes)
- additional glyph metric information specific to math mode (such as italic corrections, accent placement, or kerning)

In the following sections, we will discuss some of these parameters in more detail, illustrating the similarities and differences between traditional TEX math fonts and OpenType math fonts.

Parameters of OpenType math fonts

The parameters of the OpenType MATH table play a similar role as TEX's \fontdimen parameters, controlling various aspects of math typesetting, such as the placement of limits on big operators, the placement of numerators and denominators in fractions, or the placement of superscripts and subscripts.

While a number of parameters are specified in TEX through the \fontdimen parameters of math fonts, there are other parameters which are defined by built-in rules of TEX's math typesetting engine. In many such cases, additional parameters have been introduced in the OpenType MATH table, making it possible to specify all the relevant parameters in the math font without relying on built-in rules of any particular typesetting engine.

In view of the conference motto, it is interesting to note that the two new TEX engines, XETEX and LuaTEX, have taken a very different approach how to support the additional parameters of OpenType math fonts: While XETEX has retained TEX's original math typesetting engine and uses an internal mapping to set up \fontdimen parameters from OpenType parameters [10], LuaTEX has introduced an extension of TEX's math typesetting engine [11], which will allow to take full advantage of most of the additional OpenType parameters. (More precisely, while XETEX only provides access to the OpenType parameters as additional \fontdimens, LuaTEX uses an internal data structure based on the combined set of OpenType and TEX parameters, making it possible to supply missing values which are not supported in either OpenType math fonts or traditional TEX math fonts.)

For font designers developing OpenType math fonts, it may be best to supply all of the additional OpenType parameters in order to make their fonts as widely usable as possible with any typesetting engine, not necessarily limited to any specific one of the new TEX engines.

In the following sections, we will take a closer look at the various groups of OpenType parameters, organized in a similar way as they are presented to font designers in the FontForge font editor, but not necessarily in the same order.

We will use the figures from [6, 7] as a visual clue to illustrate how the various parameters are defined in TEX, while summarizing the similarities and differences between OpenType parameters and TEX parameters in tabular form.

Limits on big operators

In TEX math fonts, there are five parameters controlling the placement of limits on big operators (see figure 1), which are denoted as ξ_9 to ξ_{13} using the notation of Appendix G.

Two of them control the default position of the limits (ξ_{10} and ξ_{12}), two of them control the inside gap (ξ_9 and ξ_{11}), while the final one controls the outside gap above and below the limits (ξ_{13}).

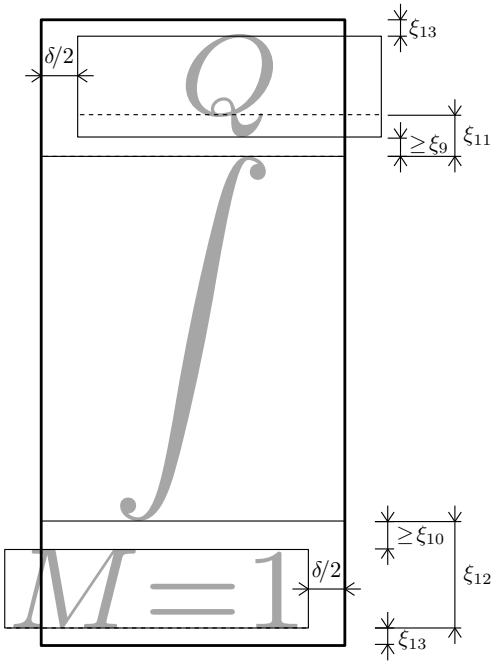

Figure 1. TEX font metric parameters affecting the placement of limits above or below big operators.

In OpenType math fonts, the MATH table contains only four parameters controlling the placement of limits on big operators. Those four parameters have a direct correspondence to TEX's parameters (as shown in table 1), while the remaining one has no correspondence and is effectively set to zero. (Considering the approach taken in other circumstances, it is very likely that if there were any such correspondence, there might actually be two parameters in OpenType instead of only one, such as `UpperLimitExtraAscender` and `LowerLimitExtraDescender`. In LuaTEX's internal data structures, there are actually two parameters for this purpose, which are either initialized from TEX's parameter ξ_{13} when using TEX math fonts or set to zero when using OpenType math fonts.)

OpenType parameter	TEX parameter
`UpperLimitBaselineRiseMin`	ξ_{11}
`UpperLimitGapMin`	ξ_9
`LowerLimitGapMin`	ξ_{10}
`LowerLimitBaselineDropMin`	ξ_{12}
(no correspondence)	ξ_{13}

Table 1. Correspondence of font metric parameters between OpenType and TEX affecting the placement of limits above or below big operators.

OpenType parameter	TEX parameter
`StretchStackTopShiftUp`	ξ_{11}
`StretchStackGapAboveMin`	ξ_9
`StretchStackGapBelowMin`	ξ_{10}
`StretchStackBottomShiftDown`	ξ_{12}

Table 2. Correspondence of font metric parameters between OpenType and TEX related to stretch stacks.

Stretch Stacks

Stretch stacks are a new feature in OpenType math fonts, which do not have a direct correspondence in TEX. They can be understood in terms of material stacked above or below stretchable elements such as overbraces, underbraces or long arrows.

In TEX, such elements were typically handled at the macro level and effectively treated in the same way as limits on big operators.

In LuaTEX, such elements will be implemented by new primitives using either the new OpenType parameters for stretch stacks (as shown in table 2) or the parameters for limits on big operators when using traditional TEX math fonts.

Overbars and Underbars

In TEX math fonts, there are no specific parameters related to the placement of overlines and underlines. Instead, there is only one parameter controlling the default rule thickness (ξ_8), which is used in a number of different situations where other parameters are expressed in multiples of the rule thickness.

In OpenType math fonts, a different approach was taken, introducing extra parameters for each purpose, even supporting different sets of parameters for overlines and underlines. Thus the MATH table contains the following parameters related to overlines and underlines (as shown in table 3), which only have an indirect correspondence in TEX.

OpenType parameter	TEX parameter
`OverbarExtraAscender`	$(= \xi_8)$
`OverbarRuleThickness`	$(= \xi_8)$
`OverbarVerticalGap`	$(= 3\,\xi_8)$
`UnderbarVerticalGap`	$(= 3\,\xi_8)$
`UnderbarRuleThickness`	$(= \xi_8)$
`UnderbarExtraDescender`	$(= \xi_8)$

Table 3. Correspondence of font metric parameters between OpenType and TEX affecting the placement of overlines and underlines.

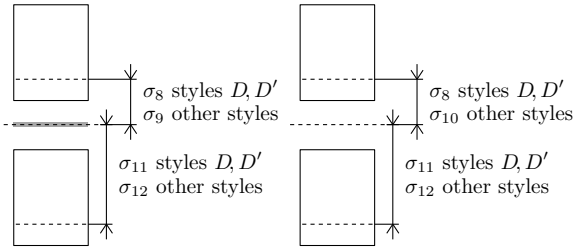

Figure 2. TEX font metric parameters affecting the placement of numerators and denominators in regular and generalized fractions.

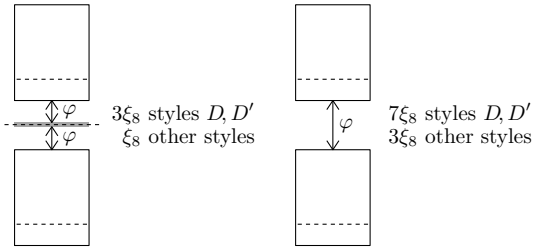

Figure 3. TEX's boundary conditions affecting the placement of numerators and denominators in regular and generalized fractions.

It is interesting to note that the introduction of additional parameters in OpenType math fonts provides for greater flexibility of the font designer to adjust the values for best results.

While TEX's built-in rules always use a fixed multiplier of the rule thickness regardless of its size, OpenType math fonts can compensate for a larger rule thickness by using a smaller multiplier.

An example can be found when inspecting the parameter values of Cambria Math: In relative terms the inside gap is only about 2.5 times rather than 3 times the rule thickness, while the latter (at about 0.65 pt compared to 0.4 pt) is quite a bit larger than in typical TEX fonts.

Obviously, making use of the individual OpenType parameters (as in LuaTEX) instead of relying on TEX's built-in rules (as in XETEX) would more closely reflect the intention of the font designer.

Fractions and Stacks

In TEX math fonts, there are five parameters controlling the placement of numerators and denominators (see figure 2), which are denoted as σ_8 to σ_{12} using the notation of Appendix G.

Four of them apply to regular fractions, either in display style (σ_8 and σ_{11}) or in text style and below (σ_9 and σ_{12}), while the remaining one applies to the special case of generalized fractions when the fraction bar is absent (σ_{10}).

Besides those specific parameters, there are also a number of parameters which are based on built-in rules of TEX's math typesetting engine, expressed in multiples of the rule thickness (ξ_8), such as the thickness of the fraction rule or the inside gap above and below the fraction rule (see figure 3).

In OpenType math fonts, a different approach was once again taken, introducing a considerable number of additional parameters for each purpose. Thus the MATH table contains 9 parameters related to regular fractions and 6 more parameters related to generalized fractions (also known as stacks).

As shown in table 4, there is a correspondence for all TEX parameters, but this correspondence isn't necessarily unique when the same TEX parameter is used for multiple purposes in fractions and stacks. Obviously, font designers of OpenType math fonts should be careful about choosing the values of OpenType parameters in a consistent way.

Analyzing the font parameters of Cambria Math once again shows how the introduction of additional parameters increases the flexibility of the designer to adjust the parameters for best results: In relative terms, `FractionDisplayStyleGapMin` is only about 2 times rather than 3 times the rule thickness. Similarly, `StackDisplayStyleGapMin` is only about 4.5 times rather than 7 times the rule thickness. In absolute terms, however, both parameters are about the same order of magnitude as in typical TEX fonts.

OpenType parameter	TEX parameter
`FractionNumeratorDisplayStyleShiftUp`	σ_8
`FractionNumeratorShiftUp`	σ_9
`FractionNumeratorDisplayStyleGapMin`	$(= 3\,\xi_8)$
`FractionNumeratorGapMin`	$(= \xi_8)$
`FractionRuleThickness`	$(= \xi_8)$
`FractionDenominatorDisplayStyleGapMin`	$(= 3\,\xi_8)$
`FractionDenominatorGapMin`	$(= \xi_8)$
`FractionDenominatorDisplayStyleShiftDown`	σ_{11}
`FractionDenominatorShiftDown`	σ_{12}
`StackTopDisplayStyleShiftUp`	σ_8
`StackTopShiftUp`	σ_{10}
`StackDisplayStyleGapMin`	$(= 7\,\xi_8)$
`StackGapMin`	$(= 3\,\xi_8)$
`StackBottomDisplayStyleShiftDown`	σ_{11}
`StackBottomShiftDown`	σ_{12}

Table 4. Correspondence of font metric parameters between OpenType and TEX affecting the placement of numerators and denominators.

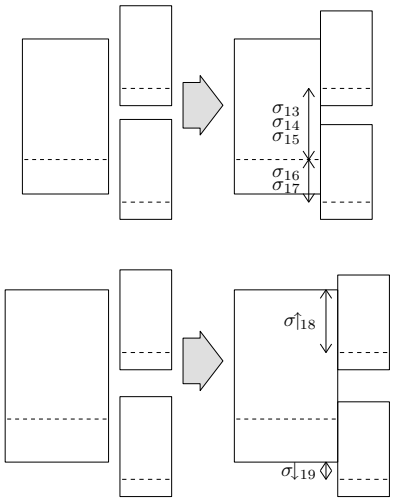

Figure 4. TEX font metric parameters affecting the placement of superscripts and subscripts on a simple character or a boxed subformula.

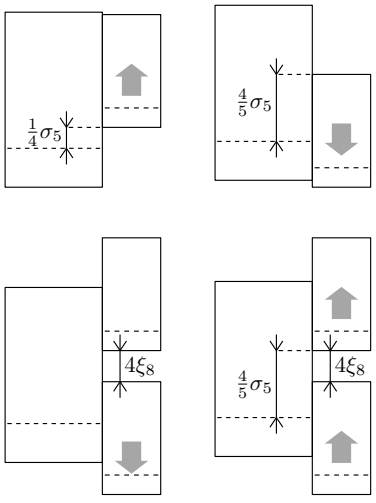

Figure 5. TEX font metric parameters affecting the placement of superscripts and subscripts in cases of resolving collisions.

Superscripts and Subscripts

In TEX math fonts, there are seven parameters controlling the placement of superscripts and subscripts (see figure 4), which are denoted as σ_{13} to σ_{19} using the notation of Appendix G.

Three of them apply to superscripts, either in display style (σ_{13}), in text style and below (σ_{14}), or in cramped style (σ_{15}), while the other two apply to the placement of subscripts, either with or without a superscript (σ_{16} and σ_{17}).

Finally, there are two more parameters which apply to superscripts and subscripts on a boxed subformula (σ_{18} and σ_{19}), which also apply to limits attached to big operators with `\nolimits`.

Besides those specific parameters, there are also a number of parameters which are based on TEX's built-in rules, expressed in multiples of the x-height (σ_5) or the rule thickness (ξ_8), most of them related to resolving collisions between superscripts and subscripts or adjusting the position when a superscript or subscript becomes too big (see figure 5).

In OpenType math fonts, we once again find a number of additional parameters for each specific purpose, as shown in table 5.

It is interesting to note that some of the usual distinctions made in TEX were apparently omitted in the OpenType MATH table, as there is no specific value for the superscript position in display style, nor are there any differences in subscript position in the presence or absence of superscripts.

While it is not clear why there is no correspondence for these parameters, it is quite possible that there was a conscious design decision to omit them, perhaps to avoid inconsistencies in alignment.

OpenType parameter	TEX parameter
SuperscriptShiftUp	σ_{13}, σ_{14}
SuperscriptShiftUpCramped	σ_{15}
SubscriptShiftDown	σ_{16}, σ_{17}
SuperscriptBaselineDropMax	σ_{18}
SubscriptBaselineDropMin	σ_{19}
SuperscriptBottomMin	$(= \frac{1}{4}\sigma_5)$
SubscriptTopMax	$(= \frac{4}{5}\sigma_5)$
SubSuperscriptGapMin	$(= 4\,\xi_8)$
SuperscriptBottomMaxWithSubscript	$(= \frac{4}{5}\sigma_5)$
SpaceAfterScript	`\scriptspace`

Table 5. Correspondence of font metric parameters between OpenType and TEX affecting the placement of superscripts and subscripts.

Radicals

In TEX math fonts, there are no specific parameters related to typesetting radicals. Instead, the relevant parameters are based on built-in rules of TEX's math typesetting engine, expressed in multiples of the rule thickness (ξ_8) or the x-height (σ_5).

To be precise, there are even more complications involved [6], as the height of the fraction rule is actually taken from the height of the fraction glyph rather than the default rule thickness to account for effects of pixel rounding in bitmap fonts.

OpenType parameter	TEX parameter
RadicalExtraAscender	$(= \xi_8)$
RadicalRuleThickness	$(= \xi_8)$
RadicalDisplayStyleVerticalGap	$(= \xi_8 + \frac{1}{4}\sigma_5)$
RadicalVerticalGap	$(= \xi_8 + \frac{1}{4}\xi_8)$
RadicalKernBeforeDegree	e. g. $\frac{5}{18}$ em
RadicalKernAfterDegree	e. g. $\frac{10}{18}$ em
RadicalDegreeBottomRaisePercent	e. g. 60 %

Table 6. Correspondence of font metric parameters between OpenType and TEX affecting the placement of radicals.

In OpenType math fonts, we once again find a number of additional parameters for each purpose, as shown in table 6.

While there is a correspondence for all of the parameters built into TEX's typesetting algorithms, it is interesting to note that OpenType math has also introduced some additional parameters related to the placement of the degree of an n th root ($\sqrt[n]{x}$), which is usually handled at the macro level in TEX's format files `plain.tex` or `latex.ltx`:

```
\newbox\rootbox
\def\root#1\of{%
  \setbox\rootbox
  \hbox{$\m@th\scriptscriptstyle{#1}$}%
  \mathpalette\r@@t}
\def\r@@t#1#2{%
  \setbox\z@\hbox{$\m@th#1\sqrtsign{#2}$}%
  \dimen@=\ht\z@ \advance\dimen@-\dp\z@
  \mkern5mu\raise.6\dimen@\copy\rootbox
  \mkern-10mu\box\z@}
```

As shown in the listing, the definition of the `\root` macro contains a number of hard-coded parameters, such as a positive kern before the box containing the degree and negative kern thereafter, expressed in multiples of the font-specific math unit. In addition, there is also a raise factor expressed relative to the size of the box containing the radical sign.

Obviously, the extra OpenType parameters related to the degree of radicals correspond directly to the parameters used internally in the `\root` macro, making it possible to supply a set of font-specific values instead of using hard-coded values expressed in multiples of font-specific units.

In LuaTEX, this approach has been taken one step further, introducing a new `\Uroot` primitive as an extension of the `\Uradical` primitive, making it possible to replace the processing at the macro level by processing at the algorithmic level in LuaTEX's extended math typesetting engine [11].

OpenType parameter	TEX parameter
ScriptPercentScaleDown	e. g. 70–80 %
ScriptScriptPercentScaleDown	e. g. 50–60 %
DisplayOperatorMinHeight	?? (e. g. 12–15 pt)
(no correspondence)	σ_{20} (e. g. 20–24 pt)
DelimitedSubFormulaMinHeight	σ_{21} (e. g. 10–12 pt)
AxisHeight	σ_{22} (axis height)
AccentBaseHeight	σ_5 (x-height)
FlattenedAccentBaseHeight	?? (capital height)

Table 7. Correspondence of font metric parameters between OpenType and TEX affecting some general aspects of math typesetting.

General parameters

The final group of OpenType parameters combines a mixed bag of parameters for various purposes. Some of them have a straight-forward correspondence in TEX (such as the math axis position), while others do not have any correspondence at all. As shown in table 7, there are some very noteworthy parameters in this group, which deserve some further explanations in the following paragraphs.

`(Script)ScriptPercentScaleDown`.
These OpenType parameters represent the font sizes of the first and second level script fonts relative to the base font. In TEX math fonts, these parameters do not have a correspondence in the font metrics. Instead they are usually specified at the macro level when a family of math fonts is loaded.

If a font family provides multiple design sizes (as in Computer Modern), font loading of math fonts in TEX might look like the following, using different design sizes, each at their natural size:

```
\newfam\symbols
\textfont\symbols=cmsy10
\scriptfont\symbols=cmsy7
\scriptscriptfont\symbols=cmsy5
```

If a font family does not provide multiple design sizes (as in MathTime), font loading of math fonts will use scaled-down versions of the base font:

```
\newfam\symbols
\textfont\symbols=mtsy10 at 10pt
\scriptfont\symbols=mtsy10 at 7.6pt
\scriptscriptfont\symbols=mtsy10 at 6pt
```

The appropriate scaling factors depend on the font design, but are usually defined in macro packages or in format files using higher-level macros such as `\DeclareMathSizes` in LaTEX.

In OpenType math fonts, it will be possible to package optical design variants for script sizes into a single font by using OpenType feature selectors to address the design variants and using scaling factors as specified in the MATH table. (As discussed in [12], there are many issues to consider regarding the development of OpenType math fonts besides setting up the font parameters. One such issue is the question of font organization regarding the inclusion of optical design variants into the base font.)

The corresponding code for font loading of full-featured OpenType math fonts in new TEX engines might look like the following:

```
\newfam\symbols
\textfont\symbols="CambriaMath"
\scriptfont\symbols="CambriaMath:+ssty0"
  scaled <ScriptPercentScaleDown>
\scriptscriptfont\symbols="CambriaMath:+ssty1"
  scaled <ScriptScriptPercentScaleDown>
```

If the font provides optical design variants for some letters and symbols, they will be substituted using the `+ssty0` or `+ssty1` feature selectors, but the scaling factor of `(Script)ScriptPercentScaleDown` will be applied in any case regardless of substitutions.

`DisplayOperatorMinHeight`.
This OpenType parameter represents the minimum size of big operators in display style. While TEX only supports two sizes of operators, which are used in text style and display style, OpenType can support multiple sizes of big operators and it needs an additional parameter to determine the smallest size to use in display style.

For font designers, it should be easy to set this parameter based on the design size of big operators, e. g. using 14 pt for display style operators combined with 10 pt for text style operators.

`DelimitedSubFormulaMinHeight`.
This OpenType parameter represents the minimum size of delimited subformulas and it might also be applied to the special case of delimited fractions.

To illustrate the significance, some explanations may be necessary to point out the difference between the usual case of fractions with delimiters and the special case of delimited fractions.

If a generalized fraction with delimiters is coded like the following

```
$ \left( {n \atop k} \right) $
```

the contents will be treated as a standard case of a generalized fraction, and the size of delimiters will be determined by taking into account the effects of `\delimiterfactor` and `\delimitershortfall` as set up in the format file.

As a result, we will typically get 10 pt or 12 pt delimiters in text style and 18 pt or 24 pt delimiters in display style. For typical settings, the delimiters only have to cover 90 % of the required size and they may fall short by at most 5 pt.

If a generalized fraction with delimiters is coded like the following

```
$ {n \atopwithdelims() k} $
```

the contents will be treated as a delimited fraction, and in this case the size of delimiters will depend on the `\fontdimen` parameters σ_{20} and σ_{21} applicable in either display style or text style.

As a result, regardless of the contents, we will always get 10 pt delimiters in text style and 24 pt delimiters in display style, even if 18 pt delimiters would be big enough in the standard case.

While `DelimitedSubFormulaMinHeight` may be the best choice of OpenType parameters to supply a value for TEX's `\fontdimen` parameters related to delimited fractions, it will be insufficient by itself to represent a distinction between display style and text style values needed in TEX. (Unless we simply assume a factor, such as $\sigma_{20} = 2\,\sigma_{21}$.)

In the absence of a better solution, it may be best to simply avoid using `\atopwithdelims` with OpenType math fonts in the new TEX engines and to redefine user-level macros (such as `\choose`) in terms of `\left` and `\right` delimiters.

`(Flattened)AccentBaseHeight`.
These OpenType parameters affect the placement of math accents and are closely related to design parameters of the font design.

While TEX assumes that accents are designed to fit on top of base glyphs which do not exceed the x-height (σ_5) and adjusts the vertical position of accents accordingly, OpenType provides a separate parameter for this purpose, which doesn't have to match the x-height of the font, but plays a similar role with respect to accent placement.

In addition to that, OpenType has introduced another mechanism to replace accents by flattened accents if the size of the base glyph exceeds a certain size, which is most likely related to the height of capital letters. At the time of writing, support for flattened accents has not yet been implemented in the new TEX engines, but it is being considered for LuaTEX version 0.40 [11].

In view of these developments, font designers are well advised to supply a complete set of values for all the OpenType math parameters since new TEX engines working on implementing full support for OpenType math may start using them sooner rather than later.

So far, we have discussed only one aspect of the information contained in the OpenType MATH table, focusing on the global parameters which correspond to TEX's \fontdimen parameters or to built-in rules of TEX's math typesetting algorithms.

Besides those global parameters, there are other data structures in the OpenType MATH table, which are also important to consider, as we will discuss in the following sections.

Instructions for vertical and horizontal variants and constructions

The concepts of vertical and horizontal variants and constructions in OpenType math are obviously very similar to TEX's concepts of charlists and extensible recipes. However, there are some subtle differences regarding when and how these concepts are applied in the math typesetting algorithms.

In TEX, charlists and extensible recipes are only used in certain situations when typesetting elements such as big operators, big delimiters, big radicals or wide accents. In OpenType math fonts, these concepts have been extended and generalized, allowing them to be used also for other stretchable elements such as long arrows or over- and underbraces.

Vertical variants and constructions

Big delimiters. When typesetting big delimiters or radicals TEX uses charlists to switch to the next-larger vertical variants, optionally followed by extensible recipes for vertical constructions. In OpenType math, these concepts apply in the same way.

It is customary to provide at least four fixed-size variants, using a progression of sizes such as 12 pt, 18 pt, 24 pt, 30 pt, before switching to an extensible version, but there is no requirement for that other than compatibility and user expectations. (At the macro level these sizes can be accessed by using \big (12 pt), \Big (18 pt), \bigg (24 pt), \Bigg (30 pt).)

Font designers are free to provide any number of additional or intermediate sizes, but in TEX they used to be limited by constraints such as 256 glyphs per 8-bit font table and no more than 16 different heights and depths in TFM files. In OpenType math fonts, they are no longer subject to such restrictions, and in the example of Cambria Math big delimiters are indeed provided in seven sizes.

Big operators. When typesetting big operators TEX uses the charlist mechanism to switch from text style to display style operators, but only once. There is no support for multiple sizes of display operators, nor are there extensible versions.

In OpenType math, these concepts have been extended, so it would be possible to have multiple sizes of display style operators as well as extensible versions of operators, if desired.

While LuaTEX has already implemented most of the new features of OpenType math, it has not yet addressed additional sizes of big operators, and it is not clear how that would be done.

Most likely, this would require some changes to the semantics of math markup at the user level, so that operators would be defined to apply to a scope of a subformula, which could then be measured to determine the required size of operators.

In addition, such a change might also require adding new parameters to decide when an operator is big enough, similar to the role of the parameters \delimiterfactor and \delimitershortfall in the case of big delimiters.

Horizontal variants and constructions

Wide accents. When typesetting wide math accents TEX uses charlists to switch to the next-larger horizontal variants, but it doesn't support extensible recipes for horizontal constructions.

As a result, math accents in traditional TEX fonts cannot grow beyond a certain maximum size, and stretchable horizontal elements of arbitrary size have to be implemented using other mechanisms, such as alignments at the macro level.

In OpenType math, these concepts have been extended, making it possible to introduce extensible versions of wide math accents (or similar elements), if desired. In addition, new mechanisms for bottom accents have also been added, complementing the existing mechanisms for top accents.

Over- and underbraces. When typesetting some stretchable elements such as over- and underbraces, TEX uses an alignment construction at the macro level to get an extensible brace of the required size, which is then typeset as a math operator with upper or lower limits attached.

While it would be possible to define extensible over- and underbraces in OpenType math fonts as extensible versions of math accents, the semantics of math accents aren't well suited to handle upper or lower limits attached to those elements.

In LuaTEX, new primitives \Uoverdelimiter and \Uunderdelimiter have been added as a new concept to represent stretchable horizontal elements which may have upper or lower limits attached. The placement of these limits is handled similar to limits on big operators in terms of so-called 'stretch stacks' as discussed earlier in section .

Long arrows. In TEX math fonts, long horizontal arrows are constructed at the macro level by overlapping the glyphs of short arrows and suitable extension modules (such as − or =). Similarly, arrows with hooks or tails are constructed by overlapping the glyphs of regular arrows and suitable glyphs for the hooks or tails.

In OpenType math fonts, all such constructions can be defined at the font level in terms of horizontal constructions rather than relying on the macro level. However, in most cases such constructions will also contain an extensible part, making the resulting long arrows strechable as well.

In LuaTEX, stretchable long arrows can also be defined using the new primitives `\Uoverdelimiter` as discussed in the case of over- and underbraces. The placement of limits on such elements more or less corresponds to using macros such as `\stackrel` to stack text on top of a relation symbol.

Encoding of variants and constructions

In traditional TEX math fonts, glyphs are addressed by a slot number in a font-specific output encoding. Each variant glyph in a charlist and each building block in an extensible recipe needs to have a slot of its own in the font table. However, only the entry points to the charlists need to be encoded at the macro level and these entry points in a font-specific input encoding do not even have to coincide with the slot numbers in the output encoding.

In OpenType math fonts, the situation is somewhat different. The underlying input encoding is assumed to consist of Unicode characters. However these Unicode codes are internally mapped to font programs using glyph names, which can be either symbolic (such as `summation` or `integral`) or purely technical (such as `uni2345` or `glyph3456`).

With few exceptions, most of the variant glyphs and building blocks cannot be allocated in standard Unicode slots, so these glyphs have to be mapped to the private use area with font-specific glyph names. In Cambria Math, variant glyphs use suffix names (such as `glyph.vsize<n>` or `glyph.hsize<n>`), while other fonts such as Asana Math use different names (such as `glyphbig<n>` or `glyphwide<n>`).

For font designers developing OpenType math fonts, setting up vertical or horizontal variants is pretty straight-forward, such as

```
summation: summation.vsize1 summation.vsize2 ...
integral : integral.vsize1  integral.vsize2  ...
```

or

```
tildecomb: tildecomb.hsize1 tildecomb.hsize2
```

provided that the variant glyphs use suffix names.

Setting up vertical or horizontal constructions is slightly more complicated, as it also requires some additional information which pieces are of fixed size and which are extensible, such as

```
integral : integralbt:0 uni23AE:1 integraltp:0
```

or

```
arrowboth :
  arrowleft.left:0 uni23AF:1 arrowright.right:0
```

It is interesting to note that some of the building blocks (such as `uni23AE` or `uni23AF`) have Unicode slots by themselves, while others have to placed in the private use area, using private glyph names such as `glyph.left`, `glyph.mid`, or `glyph.right`.

Moreover, vertical or horizontal constructions may also contain multiple extensible parts, such as in the example of over- and underbraces, where the left, middle, and right parts are of fixed size while the extensible part appears twice on either side.

Additional glyph metric information

Besides the global parameters and the instructions for vertical and horizontal variants and constructions, there is yet another kind of information stored in the OpenType MATH table, containing additions to the font metrics of individual glyphs.

In traditional TEX math fonts, the file format of TFM fonts only provides a limited number of fields to store font metric information. As a workaround, certain fields which are needed in math mode only are stored in rather non-intuitive way by overloading fields for other purposes [13].

For example, the nominal width of a glyph is used to store the subscript position, while the italic correction is used to indicate the horizontal offset between the subscript and superscript position.

As a result, the nominal width doesn't represent the actual width of the glyph and the accent position may turn out incorrect. As a secondary correction, fake kern pairs with a so-called skewchar are used to store an offset to the accent position.

In OpenType math fonts, all such non-intuitive ways of storing information can be avoided by using additional data fields for glyph-specific font metric information in the MATH table.

For example, the horizontal offset of the optical center of a glyph is stored in a `top_accent` table, so any adjustments to the placement of math accents can be expressed in a straight-forward way instead of relying on kern pairs with a skewchar.

Similarly, the italic correction is no longer used for the offset between superscripts and subscripts. Instead, the position of indices can be expressed more specifically in a `math_kern` array, representing cut-ins at each corner of the glyphs.

Summary and conclusions

In this paper, we have tried to help improve the understanding of the internals of OpenType math fonts. We have done this in order to contribute to the much-needed development of math support for Latin Modern and TEX Gyre fonts.

In the previous sections, we have discussed the parameters of the OpenType MATH table in great detail, illustrating the similarities and differences between traditional TEX math fonts and OpenType math fonts. However, we have covered other aspects of OpenType math fonts only superficially, as it is impossible to cover everything in one paper.

For a more extensive overview of the features and functionality of OpenType math fonts as well as a discussion of the resulting challenges to font developers, readers are also referred to [12].

In view of the conference motto, it is interesting to note that recent versions of LuaTEX have started to provide a full-featured implementation of OpenType math support in LuaTEX and Context [14, 15], which differs significantly from the implementation of OpenType math support in XETEX [10]. In this paper, we have pointed out some of these differences, but further discussions of this topic are beyond of the scope of this paper.

Acknowledgments

The author once again wishes to thank Bogusław Jackowski for permission to reproduce and adapt the figures from his paper *Appendix G Illuminated* [6]. In addition, the author also wishes to acknowledge feedback and suggestions from Taco Hoekwater and Hans Hagen regarding the state of OpenType math support in LuaTEX.

References

[1] Murray Sargent III: Math in Office Blog. http://blogs.msdn.com/murrays/default.aspx

[2] Murray Sargent III: High-quality editing and display of mathematical text in Office 2007. http://blogs.msdn.com/murrays/archive/2006/09/13/752206.aspx

[3] John Hudson, Ross Mills: Mathematical Typesetting: Mathematical and scientific typesetting solutions from Microsoft. Promotional Booklet, Microsoft, 2006. http://www.tiro.com/projects/

[4] George Williams: FontForge. Math typesetting information. http://fontforge.sourceforge.net/math.html

[5] Apostolos Syropoulos: Asana Math. www.ctan.org/tex-archive/fonts/Asana-Math/

[6] Bogusław Jackowski: Appendix G Illuminated. Proceedings of the 16th EuroTEX Conference 2006, Debrecen, Hungary. http://www.gust.org.pl/projects/e-foundry/math-support/tb87jackowski.pdf

[7] Ulrik Vieth: Understanding the æsthetics of math typesetting. *Biuletyn* GUST, 5–12, 2008. Proceedings of the 16th BachoTEX Conference 2008, Bachotek, Poland. http://www.gust.org.pl/projects/e-foundry/math-support/vieth2008.pdf

[8] Microsoft Typography: OpenType specification. Version 1.5, May 2008. http://www.microsoft.com/typography/otspec/

[9] Yannis Haralambous: Fonts and Encodings. O'Reilly Media, 2007. ISBN 0-596-10242-9 http://oreilly.com/catalog/9780596102425/

[10] Will Robertson: The unicode-math package. Version 0.3b, August 2008. http://github.com/wspr/unicode-math/tree/master

[11] Taco Hoekwater: LuaTEX Reference Manual. Version 0.37, 31 March 2009. http://www.luatex.org/svn/trunk/manual/luatexref-t.pdf

[12] Ulrik Vieth: Do we need a 'Cork' math font encoding? TUG*boat*, 29(3), 426–434, 2008. Proceedings of the TUG 2008 Annual Meeting, Cork, Ireland. Reprinted in this MAPS, 3–11. https://www.tug.org/members/TUGboat/tb29-3/tb93vieth.pdf

[13] Ulrik Vieth: Math Typesetting: The Good, The Bad, The Ugly. MAPS, 26, 207–216, 2001. Proceedings of the 12th EuroTEX Conference 2001, Kerkrade, Netherlands. http://www.ntg.nl/maps/26/27.pdf

[14] Taco Hoekwater: Math extensions in LuaTEX. Published elsewhere in this MAPS issue.

[15] Hans Hagen: Unicode math in Context Mk IV. Published elsewhere in this MAPS issue.

Ulrik Vieth
Vaihinger Straße 69
70567 Stuttgart
Germany
ulrik dot vieth (at) arcor dot de