Standardizing OpenType math fonts

Hans Hagen, Mikael P. Sundqvist

1 Introduction

ConTEXt has always had a good support for the typesetting of mathematics. ConTEXt MkII uses the pdfTEX engine and hence traditional (Type 1) fonts. Several math fonts are available, specifically designed to work seamlessly with TEX. ConTEXt MkIV, the successor version, utilizes the LuaTEX engine, providing support not only for traditional fonts but also for OpenType Unicode math fonts. Unlike the XƎTEX engine, which interpreted these new fonts in a manner similar to traditional TEX fonts, LuaTEX adheres more closely to the (unfortunately somewhat vague) OpenType specification.[1] When new fonts appeared, some were more like the traditional fonts, others more like OpenType Unicode math fonts. This leads to difficulties in achieving consistent results across different fonts and might be one reason that the Unicode engines are not yet used as much as they probably should.

In autumn 2021 we started to discuss how to improve the typesetting of OpenType Unicode mathematics, and it was natural to go on and do this for the LuaMetaTEX engine, and hence for ConTEXt LMTX. Since then, we have been engaging in daily discussions covering finer details such as glyphs, kerning, accent placement, inter-atom spacing (what we refer to as math microtypography), as well as broader aspects like formula alignment and formula line breaking (math macrotypography). This article will primarily focus on the finer details. Specifically, we will explore the various choices we have made throughout the process. The OpenType Unicode math specification is incomplete; some aspects are missing, while others remain ambiguous. This issue is exacerbated by the varying behaviors of fonts.

We make runtime changes to fonts, and add a few additional font parameters that we missed. As a result, we deviate from the standard set by Microsoft (or rather, we choose to interpret it in our own way) and exercise the freedom to make runtime changes to font parameters. Regarding this aspect, we firmly believe that our results often align more closely with the original intentions of the font designers. Indeed, the existence of “oddities” in these fonts may be attributed to the lack of an engine, during their creation, that supported all the various features, making testing difficult, if not essentially impossible. Within ConTEXt LMTX, we have the necessary support, and we can activate various helpers that allow us to closely examine formulas. Without them our work would not have been possible.

Ultimately, we hope and believe that we have made straightforward yet effective choices, rendering the existing OpenType Unicode math fonts usable. We hope that this article might be inspiring and useful for others who aim to achieve well-designed, modern math typesetting.

2 Traditional vs. OpenType math fonts

There is a fundamental difference between traditional TEX math fonts and OpenType Unicode fonts. In the traditional approach, a math setup consists of multiple independent fonts. There is no direct relationship between a math italic x and an $\hat{}$ on top of it. The engine handles the positioning almost independently of the shapes involved. There can be a shift to the right of $\hat{x}$ triggered by kerning with a so-called skew character but that is it.

A somewhat loose coupling between fonts is present when we go from a base character to a larger variant that itself can point to a larger one and eventually end up at an extensible recipe. But the base character and that sequence are normally from different fonts. The assumption is that they are designed as a combination. In an OpenType font, variants and extensibles more directly relate to a base character.

Then there is the italic correction which adds kerns between a character and what follows depending on the situation. It is not, in fact, a true italic correction, but more a hack where an untrue width is compensated for. A traditional TEX engine defaults to adding these corrections and selectively removes or compensates for them. In traditional TEX this fake width helps placing the subscript properly while the italic correction is added to the advance width when attaching subscripts and/or moving to the next atom.

In an OpenType font we see these phenomena translated into features. Instead of many math fonts we have one font. This means that one can have relations between glyphs, although in practice little of that happens. One example is that a specific character can have script and scriptscript sizes with a somewhat different design. Another is that there can be alternate shapes for the same character, and yet another is substitution of (for instance) dotted characters by dotless ones. However, from the perspective of features a math font is rather simple and undemanding.

Another property is that in an OpenType math font the real widths are used in combination with

[1] `learn.microsoft.com/en-us/typography/opentype/spec/math`

optional italic correction when a sequence of characters is considered text, with the exception of large operators where italic correction is used for positioning limits on top and below. Instead of abusing italic corrections this way, a system of staircase kerns in each corner of a shape is possible.

Then there are top (but not bottom) anchor positions that, like marks in text fonts, can be used to position accents on top of base characters or boxes. And while we talk of accents: they can come with so-called flat substitutions for situations where we want less height.

All this is driven by a bunch of font parameters that (supposedly) relate to the design of the font. Some of them concern rules that are being used in constructing, for instance, fractions and radicals but maybe also for making new glyphs like extensibles, which is essentially a traditional TEX thing.

So, when we now look back at the traditional approach we can say that there are differences in the way a font is set up: widths and italic corrections, staircase kerns, rules as elements for constructing glyphs, anchoring of accents, flattening of accents, replacement of dotted characters, selection of smaller sizes, and font parameters. These differences have been reflected in the way engines (seem to) deal with OpenType math: one can start with a traditional engine and map OpenType onto that; one can implement an OpenType engine and, if needed, map traditional fonts onto the way that works; and of course there can be some mix of these.

In practice, when we look at existing fonts, there is only one reference and that is Cambria. When mapped onto a traditional engine, much can be made to work, but not all. Then there are fonts that originate in the TEX community and these do not always work well with an OpenType engine. Other fonts are a mix and work more or less. The more one looks into details, the clearer it becomes that no font is perfect and that it is hard to make an engine work well with them. In LuaMetaTEX we can explicitly control many of the choices the math engine makes, and there are more such choices than with traditional TEX machinery. And although we can adapt fonts at runtime to suit the possibilities, it is not pretty.

This is why we gradually decided on a somewhat different approach: we use the advantage of having a single font, normalize fonts to what we can reliably support, and if needed, add to fonts and control the math engine, especially the various subsystems, with directives that tell it what we want to be done. Let us discuss a few things that we do when we load a math font.

3 Getting rid of italic corrections

OpenType math has italic corrections for using characters in text and large operators (for limits), staircase kerns for combining scripts, and top anchor for placement of accents. In LuaMetaTEX we have access to more features.

Let's remind ourselves. In a bit more detail, OpenType has:

- `An italic correction` is injected between characters in running text, but: a sequence of atoms is *not* text, they are individually spaced.
- An `italic correction` value in large operators that reflects where limits are attached in display mode; in effect, using the italic correction as an anchor.
- `Top anchors` are used to position accents over characters, but not so much over atoms that are composed from more than characters, e.g., including fractions, fences, radicals, and so on.
- `Flat accents` as substitution feature for situations where the height would become excessive.
- `Script and scriptscript` as substitution feature for a selection of characters that are sensitive for scaling down.

This somewhat limited view on math character positioning has been extended in LuaMetaTEX, and we remap the above onto what we consider a bit more reliable, especially because we can tweak these better. We have:

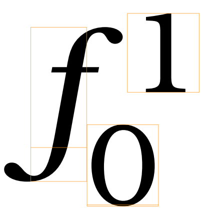
- `Corner kerns` that make it possible to adjust the horizontal location of sub- and superscripts and prescripts.
- Although `flat accents` are an existing feature, we extended them by providing additional scaling when they are not specified.
- In addition to script sizes we also have `mirror` as a feature so that we can provide right to left math typesetting. (This also relates to dropping in characters from other fonts, like Arabic.)
- In addition to the `top anchors` we also have `bottom anchors` in order to properly place bottom accents. These are often missing, so we need to construct them from available snippets.
- An additional `extensible italic correction` makes it possible to better anchor scripts to sloped large operators. This is combined with keeping track of `corner kerns` that can be specified per character.
- Characters can have `margins` which makes it possible to more precisely position accents that would normally overflow the base character and clash with scripts. These go in all four directions.

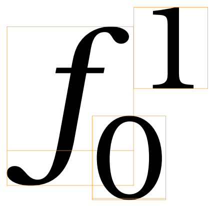
- In order to be able to place the degree in a radical more precisely (read: not run into the shape when there is more than just a single degree atom) we have `radical offsets`.

There are plenty more tuning options but some are too obscure to mention here. All high level constructors, like fences, radicals, accents, operators, fractions, etc. can be tuned via optional keyword and key/values at the macro end.

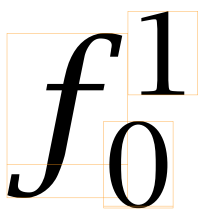
We eliminate the italic correction in math fonts, instead adding it to the width, and using a negative bottom right kern. If possible we also set a top and bottom accent anchor. This happens when we load the font. We also translate the italic correction on large operators into anchors. As a result, the engine can now completely ignore italic corrections in favor of proper widths, kerns and anchors. Let us look at a few examples.

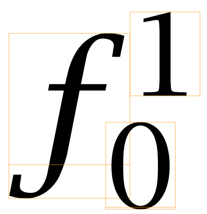
The italic f is used a lot in mathematics and it is also one of the most problematic characters. In TEX Gyre Bonum Math the italic f has a narrow bounding box; the character sticks out on both the left and right. To the right, this is compensated by a large amount of italic correction. This means that when one adds sub- and superscripts, it works well. We add italic correction to the width, and introducing a negative corner kern at the bottom right corner, and thus the placement of sub- and superscripts is not altered. Look carefully at the bounding boxes below.


original

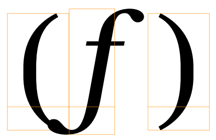

tweaked

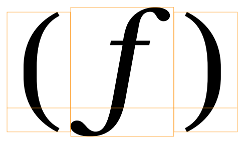
Compare with Lucida Bright Math, which comes with staircase kerns instead of italic correction. We convert these kerns into corner kerns.


original

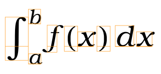

tweaked

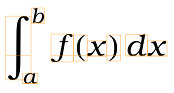
For characters that stick out to the left, we also increase the width and shift the glyph to ensure that it does not stick out on the left side. This prevents glyphs from clashing into each other.

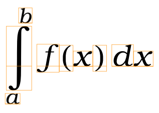

original


tweaked

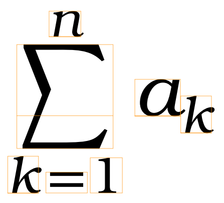
As mentioned, for the integral, one of the most common big operators, the limits are also placed with help of the italic correction. When the limits go below and on top, proper bottom and top anchor points are introduced, calculated from the italic correction. (The difference in size of the integral signs is a side effect of the font parameter `DisplayOperatorMinHeight` being tweaked, as we'll discuss more later. OpenType fonts can come with more than two sizes.)

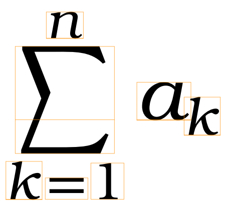

original


tweaked, nolimits

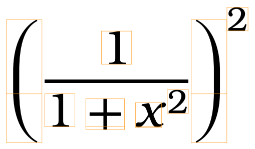

tweaked, limits

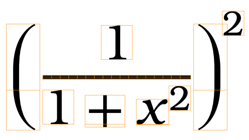
Compare these integrals with the summation, that usually does not have any italic correction bound to it. This means that the new anchor points end up in the middle of the summation symbol.


original


tweaked

We also introduce some corner kerns in cases where there were neither italic corrections nor staircase kerns. This is mainly done for delimiters, like parentheses. We can have a different amount of kerning for the various sizes. Often the original glyph does not benefit from any kerning, while the variants and extensibles do.

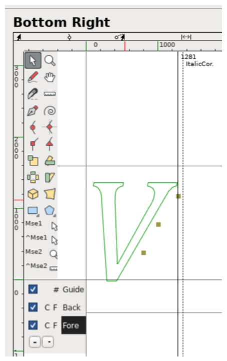

original


tweaked

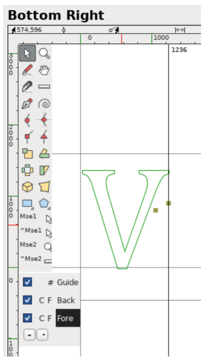
Note also the different sizes of the parentheses in the example above. Both examples are set with `\left(` and `\right)`, but the font parameters are chosen differently in the tweaked version. Font designers should have used the opportunity to have more granularity in sizes. Latin Modern Math has four, many others have steps in between, but there is a lack of consistency.

4 Converting staircase kerns

We simplify the staircase kerns, which are often somewhat sloppy and seldom complete (see figure below), into more reliable corner kerns. It's good enough and looks better on the whole. We also avoid bugs that way.



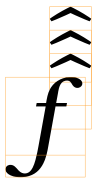
italic V



upright V

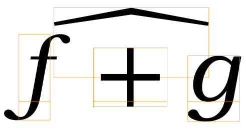
5 Tweaking accents

We ignore the zero dimensions of accents, simply assuming that one cannot know if the shape is centered or sticks out in a curious way, and therefore use proper widths with top and bottom anchors derived from the bounding box. We compensate for negative llx values being abused for positioning. We check for overflows in the engine. In case of multiple accents, we place the first one anchored over the character, and center the others on top of it.

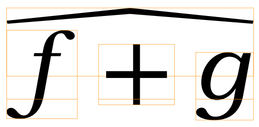


We mentioned in an earlier *TUGboat* article[2] that sometimes anchor points are just wrong. We have a tweak that resets them (to the middle) that we use for several fonts and alphabets.

Some accents, like the hat, can benefit from being scaled. The fonts typically provide the base size and a few variants.



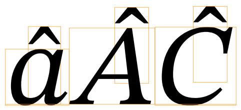
original



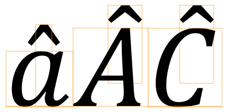
tweaked

The only fonts we have seen that support flattened accents are Stix Two Math and Cambria Math.

[2] "New directions in math fonts", **43**:3, `tug.org/TUGboat/tb43-3/tb135hagen-mathchange.pdf`



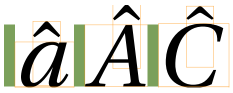
Stix Two



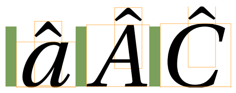
Cambria

If you look carefully, you notice that the hats over the capital letters are not as tall as the one over the lowercase letter. There is a font parameter `FlattenedAccentBaseHeight` that is supposed to specify when this effect is supposed to kick in. Even though other fonts do not use this feature, the parameter is set, sometimes to strange values (if they were to have the property). For example, n Garamond Math, the value is 420.

We introduced a tweak that can fake the flattened accents, and therefore we need to alter the value of the font parameter to more reasonable values. We communicated to Daniel Flipo, who maintains several math fonts, that the parameter was not correctly set in Erewhon math. In fact, it was set such that the flattened accents were used for some capital letters (C in the example below) but not for others (A below). He quickly fixed that. The green (gray in print) rules in the picture have the height of `FlattenedAccentBaseHeight`; it did not need to be decreased by much.



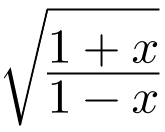
Erewhon, not fixed



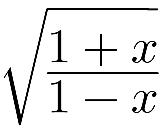
Erewhon, fixed

6 Getting rid of rules

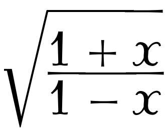
We get rid of rules as pseudo-glyphs in extensibles and bars. This also gives nicer visual integration because flat rules do not always fit in with the rest of the font. We also added support for this in the few (Polish) Type 1 math fonts that we still want to support, like Antykwa Toruńska.



with rule

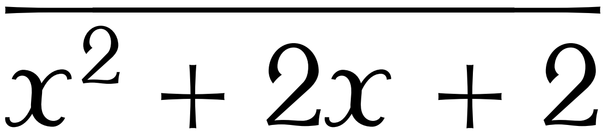


with glyph



Antykwa

Here is an enlarged example of an Antykwa rule. Latin Modern has rounded corners, here we see a rather distinctive ending.

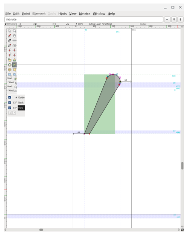


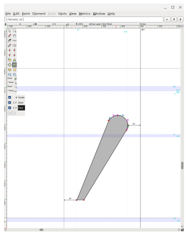
7 Tweaking primes

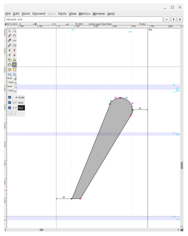
We make it no secret that we consider primes in math fonts a mess. For some reason no one could convince the Unicode people that a ‘prime’ is not a ‘minute’ (that is, U+2032 PRIME is also supposed to be used as the symbol for minutes); in case you’d like to argue that “they often look the same”, that is also true for the Latin and Greek capital ‘A’. This lost opportunity means that, as with traditional TEX fonts, we need to fight a bit with placement. The base character can or cannot be already anchored at some superscript-like position, so that makes it basically unusable. An alternative assumption might be that one should just use the script size variant as a superscript, but as we will see below, that assumes that they sit on the baseline so that we can move it up to the right spot. Add to that the fact that traditional TEX has no concept of a prime, and we need some kind of juggling with successive scripts. This is what macro packages end up doing.

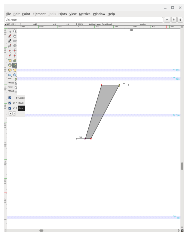
But this is not what we want. In ConTEXt MkIV we already have special mechanisms for dealing with primes, which include mapping successive primes onto the multiple characters in Unicode, where we actually have individual triple and quadruple primes and three reverse (real) primes as well. However, primes are now a native feature, like super- and subscripts, as well as prescripts and indices. (All examples here are uniformly scaled.)

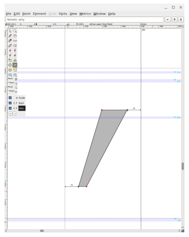
Because primes are now a native feature, we also have new font parameters `PrimeShiftUp` and `PrimeShiftUpCramped`, similar to `SuperscriptShiftUp` and `SuperscriptShiftUpCramped`, which add a horizontal axis where the primes are placed. There is also a `fixprimes` tweak that we can use to scale and fix the glyph itself. Below, we see how very different the primes from different fonts look (all examples are uniformly scaled), and then examples comparing the original and tweaked primes.

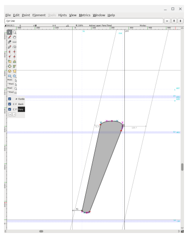

Latin Modern

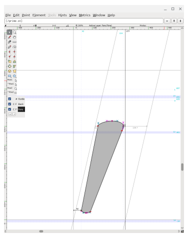

st

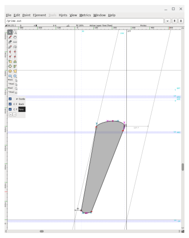

sts

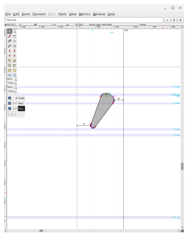

Lucida

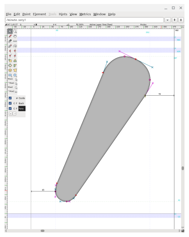

ssty


Erewhon


st


sst


Libertinus


ssty1

$$f'(x)+e^{f'(x)} \qquad f'(x)+e^{f'(x)}$$

Latin Modern original tweaked

$$f'(x)+e^{f'(x)} \qquad f'(x)+e^{f'(x)}$$

Lucida original tweaked

$$f'(x)+e^{f'(x)} \qquad f'(x)+e^{f'(x)}$$

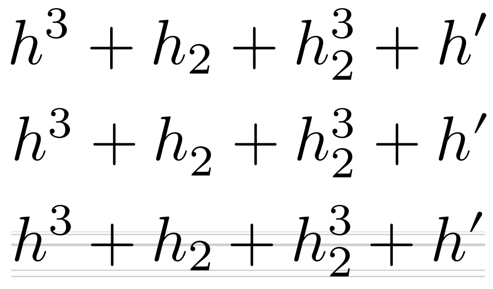
Erewhon original tweaked

$$f'(x)+e^{f'(x)} \qquad f'(x)+e^{f'(x)}$$

Libertinus original tweaked

8 Font parameters

We add some font parameters, ignore some existing ones, and fix at runtime those that look to be suboptimal. We have no better method than looking at examples, so parameters might be fine-tuned further in the future. Following are examples of pdfLATEX math, LuaLATEX math, and (as of this writing) ConTEXt LMTX:



We have already mentioned that we have a few new parameters, `PrimeShiftUp` and `PrimeShiftUpCramped`, to position primes on their own axis, independent of the superscripts. They are also chosen to always be placed outside superscripts, so the inputs `$f'^2$` and `$f^2'$` both result in $f^{2\prime}$. Authors should use parentheses in order to avoid confusion.

Let us briefly mention the other parameters. These are the adapted parameters for TEX Gyre Bonum:

```
AccentTopShiftUp                 =  -15
FlattenedAccentTopShiftUp        =  -15
AccentBaseDepth                  =   50
DelimiterPercent                 =   90
DelimiterShortfall               =  400
DisplayOperatorMinHeight         = 1900
SubscriptShiftDown               =  201
SuperscriptShiftUp               =  364
SubscriptShiftDownWithSuperscript
  = "1.4*SubscriptShiftDown"
PrimeShiftUp
  = "1.25*SuperscriptShiftUp"
PrimeShiftUpCramped
  = "1.25*SuperscriptShiftUp"
```

Some of these are not in OpenType. We can set up much more, but it depends on the font what is needed, and also on user demands.

We have noticed that many font designers seem to have had problems setting some of the values; for example, `DisplayOperatorMinHeight` seems to be off in many fonts.

9 Profiling

Let us end with profiling, which is only indirectly related to the tweaking of the fonts. Indeed, font parameters control the vertical positioning of sub- and superscripts. If not carefully set, they might force a non-negative `\lineskip` where not necessary. In the previous section we showed how these parameters were tweaked for Bonum.

Sometimes formulas are too high (or have a too large depth) for the line, and so a `\lineskip` is added so that the lines do not clash. If the lowest part of the top line (typically caused by the depth) and the tallest part of the bottom line (caused by the height) are not close to each other on the line, one might argue that this `\lineskip` does not have to be added, or at least with reduced amount. This is possible to achieve by adding `\setupalign[profile]`. An example is in figure 1.

In the figure, we enabled a helper that shows us where the profiling feature kicks in. We also show the lines (`\showmakeup[line]`). Below we show the example without those helpers. You can judge for yourself which one you prefer.

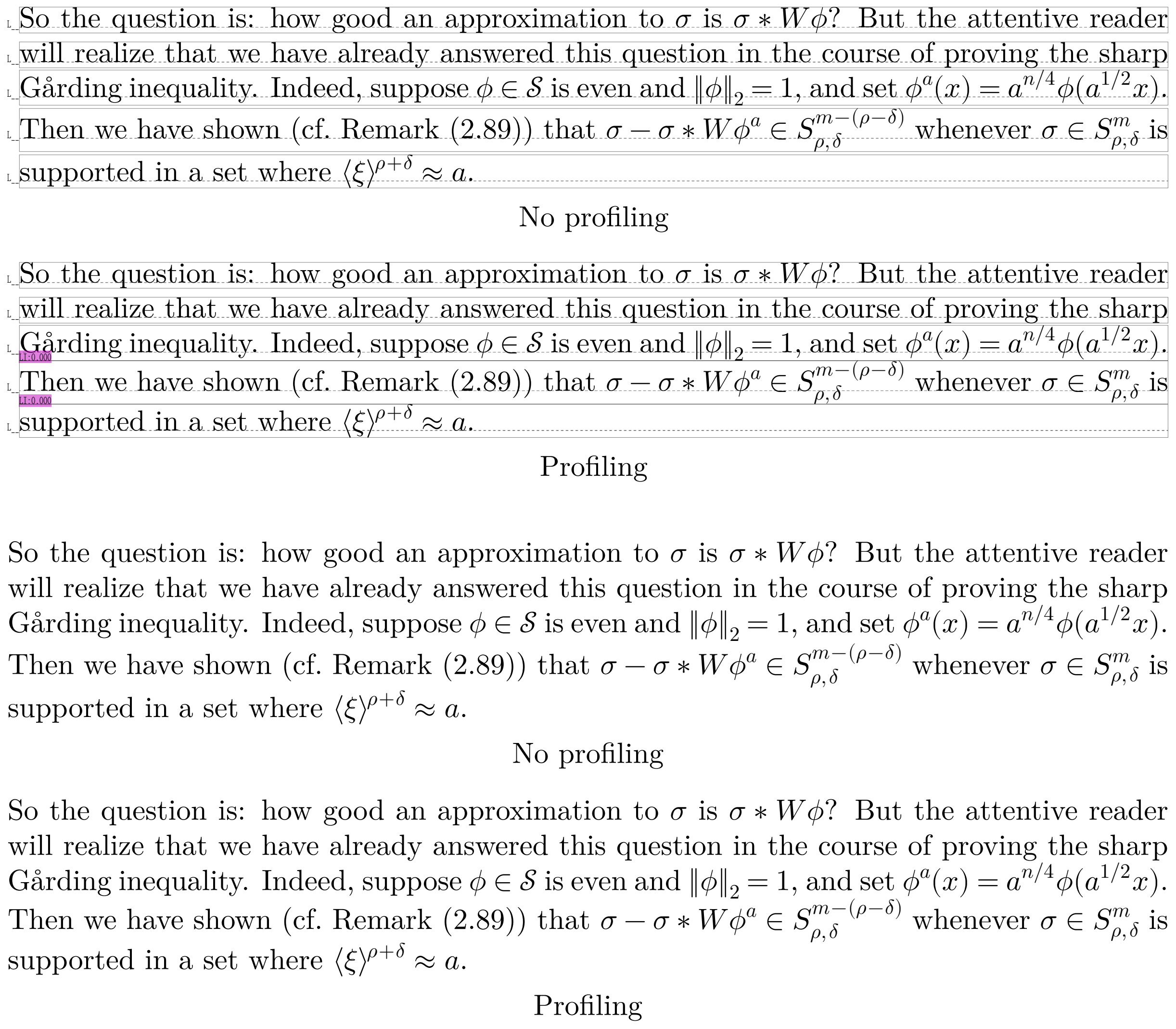
It is worth emphasizing that, contrary to what one might believe at first, the profiling does not substantially affect the compilation time. On a 300-page math book we tried, which usually compiles in about 10 seconds, profiling did not add more than 0.5 seconds. The same observation holds for the other math tweaks we have mentioned: the overhead is negligible.

10 Conclusions

All these tweaks can be overloaded per glyph if needed; for some fonts, we indeed do this, in so-called goodie files. The good news is that by doing all this we present the engine with a font that is consistent, which also means that we can more easily control the typeset result in specific circumstances.

The reader may wonder how we ended up with this somewhat confusing state of affairs in the font world. Here are some possible reasons. There is only one reference font, Cambria, and that uses its reference word processor renderer, Word. Then came XeTEX that as far as we know maps OpenType math onto a traditional TEX engine, so when fonts started coming from the TEX crowd, traditional dimensions and parameters sort of fit in. When LuaTEX showed up, it started from the other end: OpenType. That works well with the reference font but less so with that ones coming from TEX. Eventually more fonts showed up, and it's not clear how these got tested because some lean towards the traditional and others towards the reference fonts. And, all in all, these fonts mostly seem to be rather untested in real (more complex) math.

The more we looked into the specific properties of OpenType math fonts and rendering, the more we got the feeling that it was some hybrid of what TEX does (with fonts) and ultimately desired behavior. That works well with Cambria and a more or less frozen approach in a word processor, but doesn't suit well with TEX. Bits and pieces are missing, which could have been added from the perspective of generalization and imperfections in TEX as well. Lessons learned from decades of dealing with math in macros



So the question is: how good an approximation to σ is $\sigma * W\phi$? But the attentive reader will realize that we have already answered this question in the course of proving the sharp Gårding inequality. Indeed, suppose $\phi \in \mathcal{S}$ is even and $\|\phi\|_2 = 1$, and set $\phi^a(x) = a^{n/4}\phi(a^{1/2}x)$. Then we have shown (cf. Remark (2.89)) that $\sigma - \sigma * W\phi^a \in S^{m-(\rho-\delta)}_{\rho,\delta}$ whenever $\sigma \in S^m_{\rho,\delta}$ is supported in a set where $\langle\xi\rangle^{\rho+\delta} \approx a$.

No profiling

So the question is: how good an approximation to σ is $\sigma * W\phi$? But the attentive reader will realize that we have already answered this question in the course of proving the sharp Gårding inequality. Indeed, suppose $\phi \in \mathcal{S}$ is even and $\|\phi\|_2 = 1$, and set $\phi^a(x) = a^{n/4}\phi(a^{1/2}x)$. Then we have shown (cf. Remark (2.89)) that $\sigma - \sigma * W\phi^a \in S^{m-(\rho-\delta)}_{\rho,\delta}$ whenever $\sigma \in S^m_{\rho,\delta}$ is supported in a set where $\langle\xi\rangle^{\rho+\delta} \approx a$.

LI:0.000

LI:0.000

Profiling

So the question is: how good an approximation to σ is $\sigma * W\phi$? But the attentive reader will realize that we have already answered this question in the course of proving the sharp Gårding inequality. Indeed, suppose $\phi \in \mathcal{S}$ is even and $\|\phi\|_2 = 1$, and set $\phi^a(x) = a^{n/4}\phi(a^{1/2}x)$. Then we have shown (cf. Remark (2.89)) that $\sigma - \sigma * W\phi^a \in S^{m-(\rho-\delta)}_{\rho,\delta}$ whenever $\sigma \in S^m_{\rho,\delta}$ is supported in a set where $\langle\xi\rangle^{\rho+\delta} \approx a$.

No profiling

So the question is: how good an approximation to σ is $\sigma * W\phi$? But the attentive reader will realize that we have already answered this question in the course of proving the sharp Gårding inequality. Indeed, suppose $\phi \in \mathcal{S}$ is even and $\|\phi\|_2 = 1$, and set $\phi^a(x) = a^{n/4}\phi(a^{1/2}x)$. Then we have shown (cf. Remark (2.89)) that $\sigma - \sigma * W\phi^a \in S^{m-(\rho-\delta)}_{\rho,\delta}$ whenever $\sigma \in S^m_{\rho,\delta}$ is supported in a set where $\langle\xi\rangle^{\rho+\delta} \approx a$.

Profiling

Figure 1: Above: comparison of standard (no profiling) and math profiling typesetting, with guides for where profiling occurred; namely, the fourth and fifth baselines are altered. Below: the same, without the guides.

and math fonts were not reflected in the OpenType fonts and approach, which is of course understandable as OpenType math never especially aimed at TeX. But that also means that at some point one has to draw conclusions and make decisions — which is what we do in ConTeXt, LuaMetaTeX and the runtime-adapted fonts. And it gives pretty good and reliable results.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University