Patching Lucida Bright Math

Hans Hagen, Mikael P. Sundqvist

1 Introduction

During the last year we have been working on the typesetting of mathematics in ConTEXt LMTX. This system is using OpenType fonts, and in particular Unicode math fonts. In the last decade several such math fonts have been created, many of them by converting old fonts from the Type 1 format. Lucida Bright Math[1] is one of these fonts, though the designers, Charles Bigelow and Kris Holmes, made significant additions and changes when developing the OpenType version. It comes in two weights, regular and bold.

While working through the math engine we have been running tests with essentially all the freely available OpenType math fonts available, and we have noticed that, besides being different in the approach to italic corrections, kerning and metrics, they all come with small issues. This is not so surprising, given that the fonts typically have thousands of glyphs, many parameters, and the specification of the format is often vague.

We fixed many common font issues in so-called goodie files, and the patching takes place at runtime. (See the accompanying article in this issue for much more about this.) We have finally come to a point where we believe that we have a model where most of the fonts look OK, independent of whether they are old converted TEX fonts controlled by italic corrections or new fonts driven by staircase kerns. We consider Lucida Bright Math to be one of the better fonts, both in the sense that the design is beautiful and that we did not have to tweak it so much to get it look right.

Nevertheless, we found some flaws in the font, and reported a few of them on the Lucida mailing list. They were put on the list of corrections to be fixed for the next Lucida release. In the meantime, we began to discuss the possibility to do font fixes directly in the font editor FontForge. Combined with the possibility to debug with the available visual helpers in ConTEXt LMTX, we realized that we had a rather effective work flow for editing and fixing. The problem-solving part and the direct payoff when we could see things getting corrected live on screen also made the process a joy. Below we give an overview of the fixes we did. We hope that the Lucida users out there will benefit from these changes.

[1] Read more about this font at `tug.org/TUGboat/tb37-2/tb116bigelow-lucidamath.pdf`

After the fixing we need fewer tweaks in ConTEXt for Lucida, but we also have to make sure that tweaks could be applied per font version, because even with TUG's unique update policy it might take a while before all users have the new version.

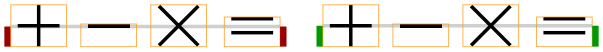
2 Correcting the math axis

This is what Microsoft writes[2] about the math axis (our emphasis):

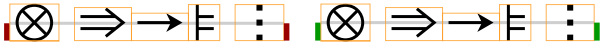
> "In math typesetting, the term axis refers to a horizontal reference line used for positioning elements in a formula. The math axis is similar to but distinct from the baseline for regular text layout. For example, *in a simple equation, a minus symbol or fraction rule would be on the axis*, but a string for a variable name would be set on a baseline that is offset from the axis. The axisHeight value determines the amount of that offset."

Let us look at the minus sign.[3]

The minus sign to the left is not centered vertically on the math axis. The value of the math axis in Lucida Math Bright has been set to 313, but the minus was centered on 325. At first we thought that this might have been a problem with the minus sign, but when looking at more glyphs, we realized that a large majority of the ones that one could argue should be aligned vertically around the math axis were in fact aligned to the height 325. We show below first some of the most common symbols



and some others, less often used



In fact, almost all symbols that one could argue should be placed vertically centered on the math axis are centered on 325. We conclude that the value 313 is not correct for this font, it should simply be 325. Thus, *we changed the math axis to 325*. This had some, not too big, consequences. We needed

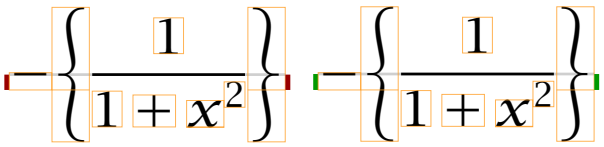
[2] `learn.microsoft.com/en-us/typography/opentype/spec/math#mathconstants-table`

[3] Here, and in the continuation, we show in our examples the output before our edits (on the left or above) between a pair of red rules (dark gray on paper) and the edited version (right or below) between green ones (medium gray). The red and green boxes indicate the height of the math axis. The gray rule in the background is centered at the height of the math axis. Orange (light gray) boxes show bounding boxes of glyphs (with an extra line for the baseline, if the glyph has a non-zero height and depth). The glyph shape itself is black.

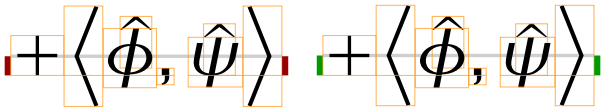
doi.org/10.47397/tb/43-3/tb135hagen-lucida

to adapt vertically the parentheses, since originally they were aligning vertically on the old math axis (i.e., they were correct before). We also needed to align integrals (some of them were in fact not centered at the math axis before).

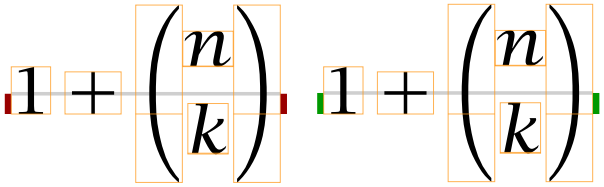
In practice this meant that many glyphs needed to be raised by 12 units in FontForge. We emphasize again that this was done to have a perfect vertical alignment with the minus and plus signs, and the other symbols that live symmetrically around the math axis. This is striking for the braces



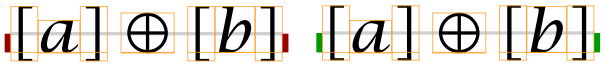
and for the angle brackets



but the difference is less obvious for the round parentheses

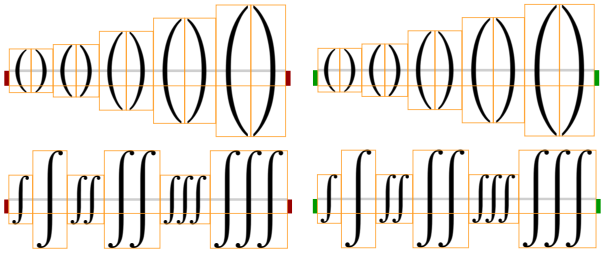


and the square brackets



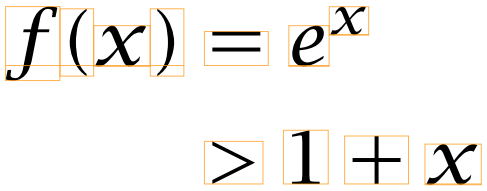
3 Aligning around the math axis

As mentioned, the different parentheses and similar glyphs were centered around the old math axis 313. We have shifted them up 12 steps so that they are now aligned with the corrected math axis. Below we give a few examples of glyphs that were shifted (all of them were of course reported to TUG).

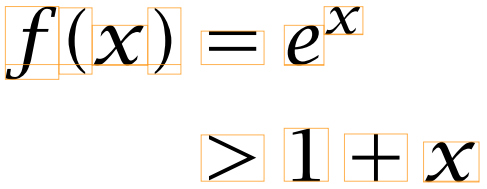


4 Modifying glyph sizes

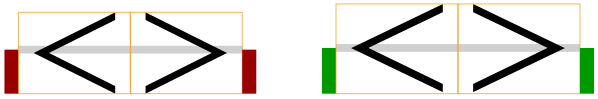
Our first mail to the Lucida mailing list was about the < and >. We noticed that their size was different from other similar relation symbols, like the equal sign =. This becomes a problem when one is aligning equations on different lines, since then one usually aligns on these characters, and if they are of different width, the result will not look good. This is how it could look in the old font:



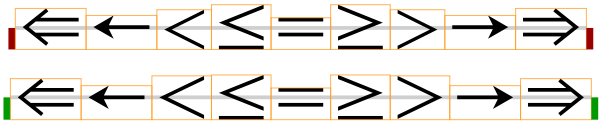
If you look carefully at the *e* and the 1, they are not horizontally aligned with each other, and the reason for that is simply that the bounding box of > is smaller than the bounding box of =. We added a temporary fix to ConTEXt (in the goodie file for Lucida) where we scaled these glyphs. But it is better to fix them in the fonts. After fixing the font, the example looks like this.



We not only changed the bounding box of the glyphs, but also scaled the glyph horizontally so that it had the same width as the equal sign (the glyph, not the bounding box!). We also scaled it vertically so that it became centered on the math axis. When that was done, we made sure that it ended up with the same side bearing as the equal sign.

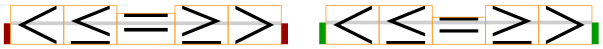


It might help to see these symbols together with some other similar ones.

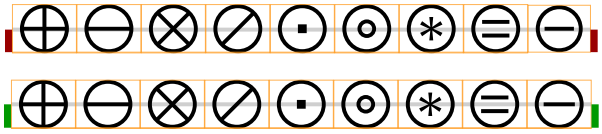


Lucida Bright Math also has some alternative, in fact smaller, versions of some glyphs. (They are the operators from the original Type 1 fonts; the font designers increased the operator size in OpenType based on their observations of usage and user requests, but some users prefer the original size, so both are available.) They are available by activating the ss03 style alternative, “small operators”. Here, the problem is that the equal sign has no alternate, so the big version is used.

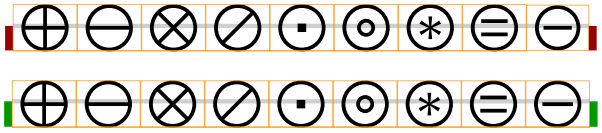
The previous less than and greater than fit well with the alternate stylistic set ss03. We thus added new slots at the end of the font for that purpose. Also, the equal sign lacked a version in ss03, so we added it. Note that the symbols in ss03 are not in general centered vertically around the math axis.



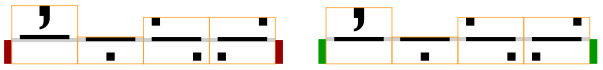
FontForge has a nice view where it is easy to browse the glyphs one by one with the position preserved. That makes it very clear when there are differences in following glyphs that should be similar. While browsing in this manner in the slots 0x2295 to 0x229D (various circles with decorations inside) we noticed that the last one had different dimensions than the other. This was the case both for the ordinary glyphs



and for the smaller variants in ss03.

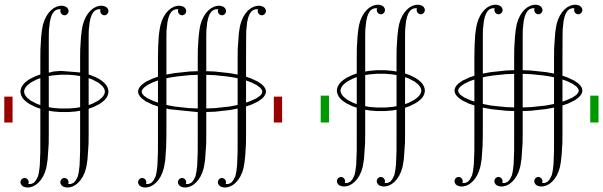


As you can see, the last one was a bit small, and is fixed. One more such symbol that stood out, 0x2A29, the minus sign with a comma on top (who uses that?):



5 Artifacts in the integral glyphs

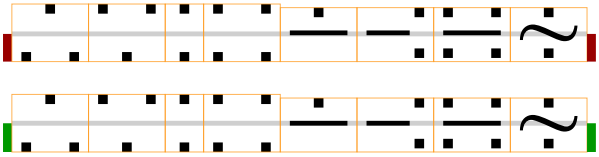
While going over the shifting of the parentheses and integrals, we noticed two glyphs that had defects. It was the display versions of 0x222F (`\oiint`) and 0x2230 (`\oiiint`), and in both cases it was the oval part of the symbol that was incorrect.



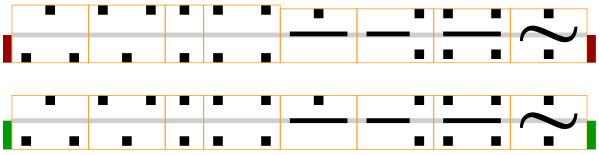
We realized that the glyphs had some extra points added that messed them up. We simply removed the extra points, made some point a corner point and tuned the control points to agree with the other corresponding ones in the glyph.

6 Dots, dots, dots

We also found some inconsistent combinations of glyphs that include dots (or, squares, as they are in Lucida). If we look at 0x02234 to 0x0223B, for example, we see that first four are different from the final four.

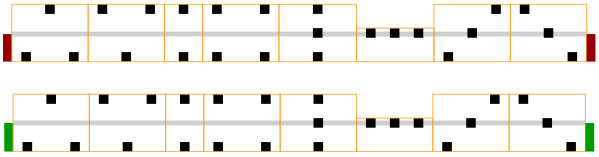


We first decided to make the first four of these adopt the spacing of the last four. It was mainly the ratio symbol (0x2236, the third in the list) that made us decide that, since we agreed that the two squares in it are simply too far away from each other (remember, this symbol is used in the `\colon` construction). We then ended up with this.



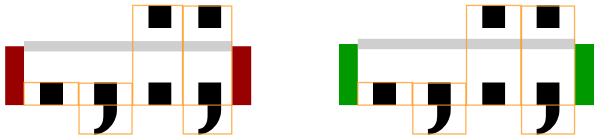
As there are so many symbols, we eventually found out that our local changes introduced new inconsistencies.

We therefore decided to ditch our first changes, but instead of throwing them away, we added the glyphs as a new set of style alternates, ss06. Thus, these glyphs stay unchanged (note that the new choice of math axis also agree with them):



We are still not completely happy with the ratio, and prefer the version from the newly added alternate set (below shown to the right).

While discussing the ratio, we looked also at the normal colon (0x3A) and semicolon (0x3B) characters, and we noticed that their side bearings were not symmetrical. We thus fixed that, and made them consistent with the period and the comma.



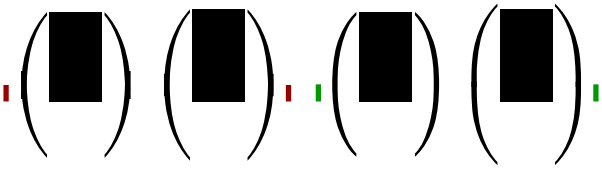
Finally, we also saw that the ellipses on the baseline (0x22EF) are wider spaced than the ones on the math axis (0x2026). We decreased the right side bearing of 0x22EF so that became consistent with similar constructions.

We added yet another style alternative, ss07, with a version of 0x22EF with the small squares spaced in the same way as the other similar glyphs.

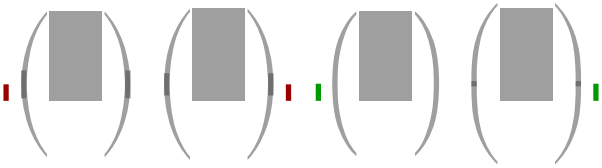
One can still question the fact that the squares in the ellipses are smaller than the squares in the comma and the period.

7 Extensible recipes

We did not want to touch the extensible recipes. But then we saw that the top and bottom pieces of the round parentheses, when just too large for the largest variants, clash into each other with bad results.

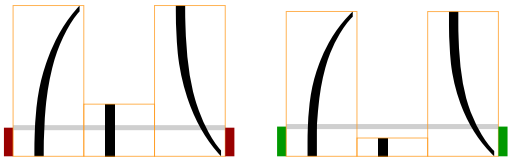


The problem becomes more apparent if we use transparent colors.



Note that in the fixed version, the largest variant is still used in the left example, while the first extensible is used in the right one. This means, and that is unavoidable, that the parentheses are slightly taller than the content. This can of course also happen when we use the variants.

So, how should the extensible recipe be built? Let us look at the left parenthesis. It consists of three parts, the “left parenthesis upper hook” (0x239B), the “left parenthesis extension” (0x239C) and “left parenthesis lower hook” (0x239D).



There is a table in the font, a recipe, that decides how the extensible is to be built. For the left parenthesis, the first and the third are always used once, and then there can be as many middle ones as needed (including zero). Looking at Table 1 we see that the size of the glyphs are 1648 (top and bottom) and 570 (the middle one). Since the top and bottom glyphs are needed, the minimum height plus depth will be $2 \times 1648 = 3296$. Or, that is what one could imagine. But there is a font parameter `MinConnectorOverlap`, set to 40 in Lucida Bright Math. It is supposed to be the minimal overlap of the glyphs. This means that the true minimal height is 3256. This can be compared with the height plus depth of the largest variant of the left parenthesis, $1686 + 1061 = 2747$.

Table 1: Original extensible values.

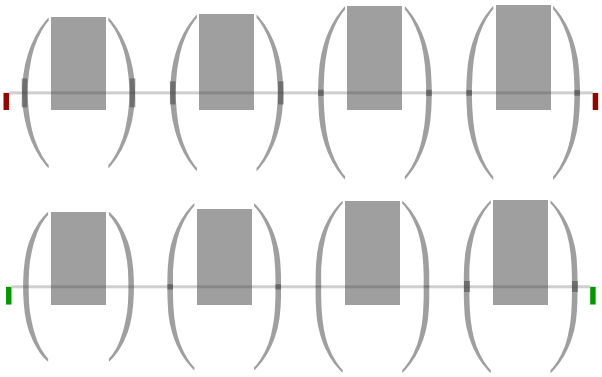
Glyph	Extender	StartLen	EndLen	FullLen
uni239D	false	549	549	1648
uni239C	true	190	190	570
uni239B	false	549	549	1648

The biggest problem is not the values in the table, but the fact that the top and bottom parts do not have ends that can overlap aesthetically. No matter how little we overlap, the pieces will not fit perfectly with each other, or with the rectangular middle piece, since they are not rectangular themselves at the end. Thus, we wanted to add a rectangular part to the first and the last pieces. But then we got another problem. If we just added a rectangular piece, the glyphs would be too large, and the extensibles would kick in too late. After some trial and error, we decided to scale the original top and bottom parts just slightly, and then to add a rectangular piece of height 100. We changed the height of the extensible part to 200. Inspired by the TEX Gyre fonts, we ended up with the values in Table 2.

Table 2: Updated extensible values.

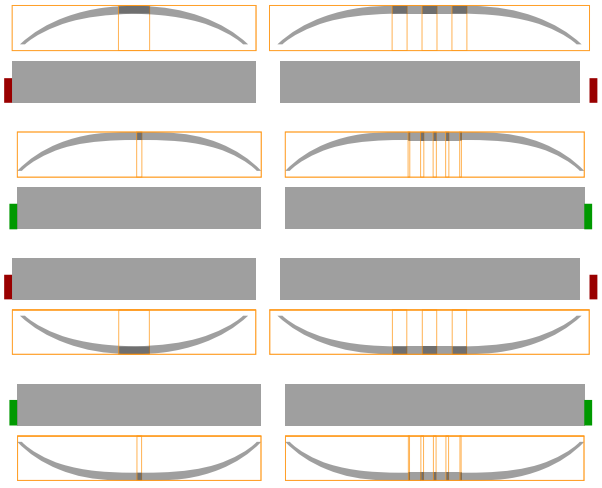
Glyph	Extender	StartLen	EndLen	FullLen
uni239D	false	0	100	1583
uni239C	true	200	200	200
uni239B	false	100	0	1583

The top and bottom pieces are allowed to overlap by 100 units with each other (or with the middle piece), exactly the size of the added rectangle. For maximum flexibility, the middle piece is allowed to overlap 200 in both directions. Let us look at a few examples.

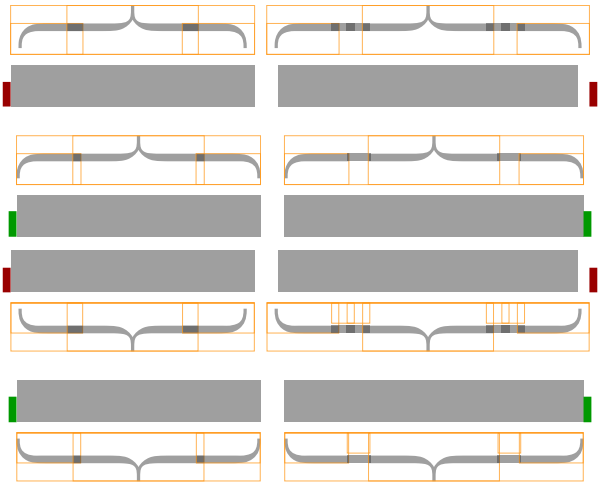


In the first case (to the left), the content is high enough to trigger the extensible in the unfixed font, but not in the fixed one. In the next, the content is precisely sufficiently high to trigger the extensible also in the fixed font. Observe that the parentheses are slightly bigger than the content. The grayer area shows the overlap, and it corresponds approximately to the value 100 in Table 2. If we increase the content a bit more we still get no middle piece, but the overlap is now very small. The size now fits the content well. Finally, with slightly larger rule, the new version adds a rectangular middle piece. If you look carefully, you will see that almost all of it is overlapping with the other two pieces. The unfixed font still has no middle piece.

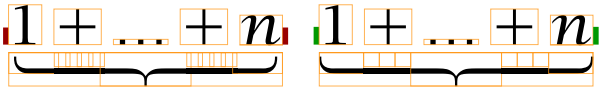
A similar situation is present for symbols that scale horizontally, and in particular for the parentheses. We decided to scale equally as much as for the vertical parentheses, and then also add a rectangular piece.



An analogous change was made to the horizontal up and down braces.

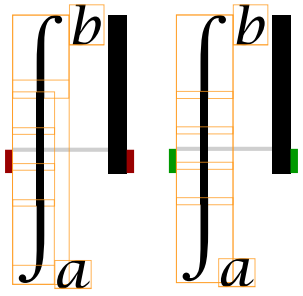


The attentive reader may notice that the side bearings of the glyphs are set to zero. We could not find any other math font with a non-zero side bearing for these glyphs. That makes sense, since they only complicates the calculations.



A character in an OpenType font can have two variant lists, two extensible recipes and two extra italic corrections, meant to support both horizontal and vertical extensibles. Only a few fonts have these extra italic corrections set, and we have only observed them on integral signs, and we haven't seen corrections set on horizontal extensibles at all. The (vertical) italic correction is used for positioning the subscript and limits (on n-ary operators).

Lucida is one of those fonts that has an extensible integral. Although the LuaMetaTEX engine will use the maximum width of a snippet, to be coherent with the other fonts with extensible integrals, we have made sure that in the updated Lucida all snippets have the same width, as shown in the example below (width of middle and bottom pieces now matches width of top piece). We did not reset the corrections on individual snippets, because these are ignored anyway.

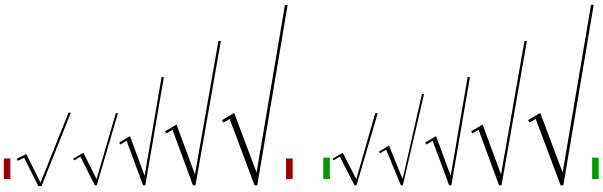


8 The radical symbol

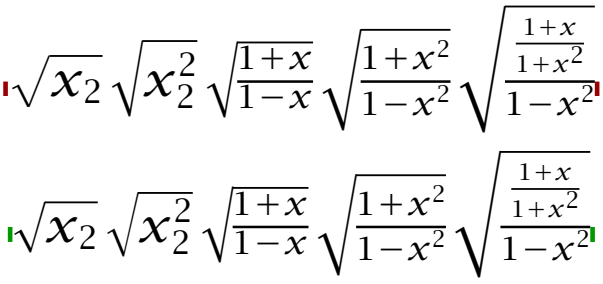
Our last stop is the radical symbol, and the reason to stop here can be seen in the following simple expression:

$$\sqrt{x}$$

The radical symbol and the horizontal rule do not fit together. When we were looking at the variants of the radical symbol, we became surprised by two things.



We first noted that the base glyph and the variants look different. Next, we realized that the first variant will probably never kick in, since it has the same size as the base glyph. We decided to move the first variant into the base glyph, to get a consistent look, and to modify the size of the first variant slightly. In the following set of examples we see that the new size is used (second from left) in the fixed version, while we get the slightly larger radical for the unfixed one.



Regarding the mismatch between the radical and the horizontal rule, this turned out to not be a problem in the font (except for the old radical base character). It was instead related to a backend related snapping feature in ConTEXt, used to prevent loading fonts in too many sizes due to rounding errors.

This is comparable to cases where TEX's scaling of 1000 means 1.000 with three digit precision, although that often goes unnoticed because it happens consistently in the whole document. The mismatch doesn't happen when we operate in PostScript points (bp) that are natural to both OpenType fonts and PDF, but it does when we use TEX points (pt) which means that when going to PDF's points we lose some accuracy. The difference between 0.9963 and 0.9954 is noticeable to the sensitive eye when you blow up these composed glyphs for testing, and we can't tolerate that, can we? So we now go for more precision at the cost of (possibly) some font sizes. There is currently an experimenting mode in ConTEXt, the compact font mode, where any font is loaded just once, anyway, but that has to be discussed elsewhere.

◇ Hans Hagen
Pragma ADE

◇ Mikael P. Sundqvist
Department of Mathematics
Lund University
Box 118
221 00 Lund
Sweden
mickep (at) gmail dot com