On musical typesetting: Sonata for TEX and METAFONT, Op. 2

Federico Garcia

This article appears in the same TUGboat issue as an obituary of Daniel Taupin, author of MusiXTEX, who unexpectedly passed away in 2003. I'd like to take the opportunity to offer it as an homage to his memory.—FG

Modern software industry has provided tools for the typesetting of music for some time already. Two programs in particular — Finale and the more recent Sibelius — virtually exhaust the music community: almost all musicians use either one or the other. Both are WYSIWYG (what you see is what you get) applications, and both have reached that stage of development in which new versions consist of little more than new shortcuts, 'cookies' of questionable relevance, and so on. Competition for ever wider markets, on the other hand, has made these programs horribly 'intelligent': intended for standard situations and the less-than-careful user, they make customization beyond a certain point painstakingly bothersome.

The problem is that non-standard situations are the order of the day in music. On the one hand, there is the so-called 'new music', the contemporary form of the 'art music' tradition, whose notational requirements have proven to be hard to standardize at all;[1] traditional notational practices are quite sufficient for popular and commercial music (increasingly the target of commercial software), but 'new music' overwhelms them in infinite ways. On the other hand, there is musicology, in which not the composition of new but the analysis of old music might potentially require not-yet-conceived-of typographical upheavals — to circle or otherwise highlight certain notes, to tie them in unusual ways, to superimpose staves, ... Thus, any serious system of professional musical typesetting has to give up the hope of foreseeing all needs and possibilities. In a word, it has to be *programmable*.

This is what some years ago got me started thinking of TEX as a suitable environment for such a system. Recently, after having had some experience with MusiXTEX — the 'officially approved' (i.e., acknowledged by Donald Knuth in his web page) application of TEX to music — as well as a taste of what Lilypond has to offer, I have put hands to work on creating (yet another) system, with portability, flexibility and adaptability as priorities.[2] What follows is a report of what I have done so far. If I'm submitting it to the public, it's because I think I might be onto something.

I wrote this article in two stages. First (Op. 1) when I had just devised the main general model, and anticipated the main algorithms needed to implement it. That was toward the end of 2002. And then now, mid-2004, when publishing it in *TUGboat* became really plausible. Particularly the first part (a history of the main idea of the system) features a narrative approach, so I have avoided anachronism: what I know now but didn't know then should not interfere too much. Along the way of telling this story, there is a detailed review and 'refutation' of MusiXTEX. It's admittedly sharp, but sincerely well-minded and grateful.

Then, in the second part, I allowed myself to 'upgrade' the article a little more deeply, taking advantage of these two years of actual work of programming. It's interesting to see how getting concrete changes ideas in unexpected ways.

I think this article can be read without knowledge of either music, TEX programming, or METAFONT. Experience would surely make it easier to understand some lines here and there, but I've tried to make it all clear to any reader, above all to non-musicians. I myself wasn't very experienced with TEX, METAFONT, or musical typesetting — and the bulk of this article is re-telling the steps I've followed and what I've learned from scratch.

PART I: *Qualitative design*

1 The nature of musical typesetting (I)

It's no coincidence that music typesetting programs are, by far, closer to being graphic utilities than text processors. Musical text is multi-dimensional. As

[1] Perhaps I should remind the reader that today's music, like other arts, is far removed from the conventions and traditions of the 'common-practice period' (17^{th}–19^{th} centuries) — for which, incidentally, the 'standard' notation was developed. Everything has changed, from the nature of the musical sound itself to its graphic representation.

[2] MusiXTEX is Daniel Taupin's development of what was originally a joint effort with Andreas Egler, MusicTEX.

Lilypond is the TEX-related but independent system developed by Jan Nieuwenhuizen and Han-Wen Nienhuys. I am by no means familiar with Lilypond — in part because in 2002, when I wrote a first version of this article, I had to give up trying to install it under Windows. One might think that I should have become familiar with it before embarking on the creation of TEX*muse*; but my intuition is that there is no real overlap between the two. For one thing, Lilypond is not a TEX package; for another, it boasts (rightly) of beautiful fonts. TEX*muse*, on the contrary, boasts of not having fonts at all, and this shows how different their nature and approach are. I have grown convinced that, as explained later on, not having fonts is actually the best way to the flexibility I am aiming at, as a musician and as a programmer.

we all know from school, the horizontal dimension is time, and the vertical dimension is pitch. But that's only part of the matter, since different voices come in different staves (different vertical position). The horizontal position of a note in a given staff depends not only on what notes in the same voice come before or after, but also, and typographically most importantly, on what notes are played at the same time by other voices. There is in effect a third dimension involved. (A look at Figure 1 might be in place to visualize all this. This piece being for piano, each staff corresponds to one hand.)

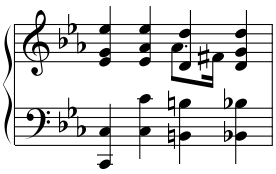


Figure 1: A fragment of a typical score (m. 9 of Chopin's piano prelude in C minor, Op. 28 No. 20).

Music is thus not a line of text (in which only the horizontal dimension matters), but also not a mathematical formula (in which the vertical dimension plays a role): it is an array of several two-dimensional 'strings' that are mutually lined up. Typographically, what this resembles most is, of course, a table. And yet there is an important difference: a musical score usually goes on and on, for many pages — the table has to be broken. Not broken as longtable breaks tables, because in this case the table is not too tall, it is too long. It has hundreds and hundreds of columns, and there is no way to know beforehand how many of them fit in each page.

2 The problem of horizontal spacing and line breaking

An idea that suggests itself naturally when one meditates about this kind of page-breaking is an analogy with TEX's output routine. Instead of collecting lines and deciding when to break the page 'vertically', here the task consists of collecting columns and deciding on horizontal breaks. This would indeed be the case when the table is input in the form of columns. Page-breaking when the table is thus pre-set is actually a rather simple matter, and we can see a solution in action in the workings of MusiXTEX. In fact, that's the latter's main breakthrough: "to address the above aim, a three pass system was developed" [8, p. 9].

The 'three-pass system' works through an external program (MusixFlex), that reads a file (`.mx1`) created by TEX, decides on page- (and, which is analogous, system-) breaking, and writes another file (`.mx2`) for TEX to read during a second TEX pass.

I always wondered whether invoking an external program was really necessary. TEX's handling of horizontal spacing, a substantial part of the line-breaking algorithm for which Knuth takes and deserves much credit — "this, in fact, is probably the most interesting aspect of the whole TEX system" [3, p. 94] — seems so powerful, so flexible, and intuitively so fit to musical spacing, that MusiXTEX appeared to be missing out. I had the impression that music offers an opportunity to really take advantage of this superb algorithm, whose capabilities go far beyond assigning a little extra space to periods and question marks in horizontal mode...

The authors of MusiXTEX lay aside this algorithm on the grounds that it

> implicitly assumes that a normal line of text will contain many words, so that inter-word glue need not stretch or shrink too much to justify the line. This strategy does not work very well for music. If *each bar of music is treated as a word*, in the sense that inter-bar glue is placed at the end of each bar, then the usual result is the appearance of unsightly gaps before each bar rule. This follows naturally from the fact that the number of bars per line is normally many fewer than the number of words in a line of text. [8, p. 9, my emphasis]

This is, in the main, correct. What is flawed is the implicit analogy: it is not "each bar of music" which has to be "treated as a word", because its inner components are also subject to spacing. Since the defining characteristic of a 'word', as far as the line-breaking algorithm is concerned, is that its components remain packed together with rigid space in between, measures (bars) do not qualify as words.

Therefore there is still a possibility of using TEX's line-breaking algorithm. My whole endeavor has been ultimately moved by the intuition that there must be some way to apply it. As we shall see, this *idée fixe* led me — round-about and through a faulty assumption — to the discovery of a promising possibility. But first, now that I have touched MusiXTEX, let me complete my review of it.

3 A word on MusiXTEX

I said above that the problem of horizontally breaking a table is a simple one when the table's columns are input as such: as columns. And that that was the case with MusiXTEX. In fact,

> the fundamental *macro* [of MusiXTEX is] `\notes ... & ... & ... \enotes` where the character

> `&` is used to separate the notes ... to be typeset on the respective staff [*sic*] of the various instruments, starting from the bottom. [7, p. 213]

So you input the first note of the bottom-most staff, then use the magical `&`, then the first note of the second-to-bottom staff, another `&`, the note in the third-to-bottom staff, etc. After getting to the top staff, you start again by going back to the bottom staff, and inputting its second note. The process is exactly analogous to LaTeX's `tabular`, only rotated 90°.

A way to picture what is involved here is imagining that to type a paragraph of text you have to type *a*) all the first words, line by line, *b*) all the second words, *c*) all the third words, *d*) ... It is terribly unnatural, and, as a result, the input is as unreadable as it can get; both creating and reading it is an excellent, but unwelcome, exercise in abstract thought. Because music, after all, is thought of much as text is: as horizontal lines. Many, interconnected lines, but lines still.[3]

Writing a MusiXTeX file is then most uncomfortable, and editing it can be really challenging. But consider the task of 'extracting parts' (selecting, for separate printing, a subset of the staves involved). This is a very common necessity, for players of an ensemble need only *their* staff (not the whole score), and the ability to do it automatically is probably the most important advantage of using a computer for music typesetting. With MusiXTeX's kind of input, this is simply unthinkable (it is analogous to, but more complicated than, extracting a single column from a `tabular` environment, even a simple one; there is no way to copy-paste, not even to automatically find the relevant pieces of the input).

David Salomon says that "the most important feature of TeX is its line-breaking capabilities.... The second most important feature of TeX is its programmability" [6, p. x]. As I have noted, MusiXTeX ignores the first feature; but the way it neglects the second one is actually more exasperating. I started this article by complaining about how customization is disregarded in commercial music software. For example, three steps are needed to change a note-head from its standard shape (the elliptical spot •), say into a ◇ or a × (both fairly usual nowadays). Of course, if you happen to need *many* such changes, you are doomed to follow the same three steps over and over...

What is this like in MusiXTeX? Well, by itself, MusiXTeX cannot do it. You have to go to `musixtex.tex` (naturally a very intricate file), find the relevant definitions, and create your new macros. Foremost, you have to know what you are doing — after all, you are programming TeX at a fairly low level. No hope of doing it if you are no TeXnician.[4] Let alone trying to change temporarily the number of lines in a staff, or creating non-standard key signatures, or connecting two notes from different staves — all of which is excruciating, but at least possible, in Finale and Sibelius. Requiring the user to verticalize what is naturally thought of horizontally is awkward; but prohibiting an efficient and logical programming is — it seems to me — not far short of deserting the very spirit of TeX.

In addition, little care was put in the choice of command names — some bear '`d`' and '`u`' for English *down* and *up*, others '`b`' and '`h`' for French *basse* and *haute*; musical names are applied wrongly ('accent' for articulation, '*sforzando*' for accent, '*pizzicato*' for *staccato*, etc.) — trying to memorize this could mean forgetting what's what. MusiXTeX is not flexible (the limitations of slurring, for example, are severe); it's not even efficient: only nine (or twelve with an 'extension library') staves can be included, because TeX registers are used for almost everything, and they are exhausted quickly. And, this review already becoming far more odious than my intention, I have to say that, with MusiXTeX, quality — O precious treasure and pride of TeX lovers — is poor: the output is ugly.[5]

To sum up, MusiXTeX does not in my view take advantage of TeX's unique possibilities. Its typesetting of non-standard music is as demanding as that of WYSIWYG commercial programs, adding the intricacy of an unreadable input. Sadly, I have had to recommend musicians (even those few well acquainted with TeX) to use Finale or Sibelius rather

[3] Taupin devotes section 1.1.1 of MusiXTeX's manual to argue that "the humanly logical way of coding music" is the procedure described. This claim is as wrong as it is apodeictic. He draws from his perception of what reading keyboard music is like. Even here his point is questionable, because the musician reads horizontal chunks from every staff (the whole tradition of sight-reading training is based on this assumption). But, in any case, the composer and the typist (who really matter) certainly do *not* conceive of music vertically. I have never seen anybody setting a score by columns, but, again, by chunks of horizontal material. Incomplete drafts by composers throughout history, including Bach's *Die Kunst der Fugue* (one of the first MusiXTeX projects), prove this. In my view, Taupin is here proving what he believes, rather than believing what he proves: understandably, his system was "the humanly logical way" of *programming* TeX for the task.

[4] The authors do provide an 'extension library' with ◇, ×, and other symbols implemented as note-heads; but this kind of *ad hoc* procedure is not a solution, for the possibilities are still limited.

[5] And this is why Lilypond puts so much emphasis on the beauty of its fonts (see note 2).

than MusiXTEX: the result is better and the effort is less if the situation is standard, and about the same otherwise.

I'm not being just politically correct, however, when I also have to say that, all in all, it is a wonder that MusiXTEX was programmed at all. I regard its creators with deep respect and admiration.

4 Premises for an alternative

The discussion above sets the grounds for what an ideal system would be like. These are my basic premises:

Programmability and flexibility. This is 'simply' a matter of constantly bewaring of rigid designs in programming.

TEX's glue has to be used. Somehow. It seems so fit for musical spacing...

Horizontal input is necessary if we want a natural and logical system, one that can compete, on ease of use, with Finale and Sibelius.

TEX's sufficiency No PostScript. Come on, no cheating, please.

About the first two points, I think it can be said that MusiXTEX simply slipped down the wrong way — far greater difficulties were being tackled by its authors, so we can forgive them.

But the third premise is a Pandora's box. 'Horizontal input' means that the user will input the different lines (staves), one by one. But there is nothing in a given particular line that indicates the relationship of its notes to the notes in other lines (and, remember, the horizontal position of a note depends foremost on the notes in other staves). For example, there is no way to know, by only reading line 3, whether its fifth note will be played before, at the same time, or after (graphically, to the left, directly above, or to the right of) the tenth note of line 6.

MusiXTEX solves the problem, as mentioned, by asking the user to input *columns*: it is the user who figures the vertical relationships out. But this is precisely what we want to spare the user. Could TEX possibly do it? Well, since these relationships depend on (and only on) the rhythmic characteristics of the notes involved, to figure them out TEX would need to 'understand' those rhythmic characteristics: nothing less than being taught music.

I still tried something else before committing to TEX's musical instruction: if the problem is to horizontalize a vertical input, why not simply rotate the score? The user will type staff by staff, as is natural to him, but for TEX this will mean column by column, which is natural for it. In addition, the problem of page-breaking a horizontally long table becomes the more manageable (and already solved) one of page-breaking a *vertically* long one. Even TEX's glue would be automatically recovered, only it would apply now to staves (rather than measures), which do behave as words in the sense that inner space is not stretched.

I was thrilled by this idea. I still think it is worth exploring for a table-like approach like MusiXTEX. But I soon realized this does not solve the big problem. The input is simplified, but the user still has to figure out the (now horizontal) rhythmic correspondences. And, again, TEX's spacing algorithm would be under-used, being applied to the relatively unimportant problem of inter-staff spacing. The output routine, on the other hand, has nothing resembling space factors with which to work on inter-note spacing (which was my main motivation to think about that algorithm).

So, no way out: for the third premise, TEX had to take music lessons, and I would be the appointed teacher. I was scared, and maybe that, along with my taste for TEX's line-breaking algorithm, is why I focused first on the second premise: bringing back TEX's glue. It will be seen that this actually involved a logical slip — but a fortunate one, because by tackling the wrong problem I found a solution to the real one.

What about the fourth premise? Why did I set this 'only-TEX' requirement to my system? I don't really know — it must have been sheer love of TEX.[6]

5 Bringing glue back

The intuition that TEX's spacing is useful for music can be stated basically thus: different (rhythmic) kinds of musical notes receive different amounts of space to their right, similar to the way periods, commas, and other characters, do in text.[7] A succession of notes can thus be imagined as a succession of words, separated by blank spaces; these spaces will be stretched by TEX (in the same way TEX

[6] And total ignorance of PostScript! (although I am willing to learn if it proves necessary; Van Zandt's miraculous PSTricks is a compelling taste of what can be achieved by 'cheating').

[7] A quick summary of the algorithm: TEX calculates how much the line has to be stretched to reach the right margin (to be justified). Then this total extra space is distributed between all the blank spaces — elastic spaces, TEX's 'glue' — in the line, proportionately to the 'space factor' of the characters before the spaces. Periods, for example, have a larger space factor than letters, and therefore spaces after periods will be given a bigger share of the total stretching.

Throughout this article I consider the algorithm only in its 'stretching' aspect, and without interference from infinite glue or 'badness' parameters.

stretches spaces in a normal line of text), so that if the space factors assigned to each note 'map' their rhythmical nature, the final stretching will agree to the rules of musical spacing.

So, the fruitful analogy is words = notes (as opposed to words = measures, which is 'semiotically' more natural, but typographically misleading — see section 2).[8] Another important aspect of this analogy will be treated in the following section.

However, it will be noted that the intuition paraphrased above is not perfectly rigorous. In all truth, as has been already said, the space at the right of the musical notes depends *not only* on their own rhythmic characteristics, but above all on the notes (which, and how many) present in other staves. The assumption that this spacing behaves similarly to the spacing for justification of texts holds true only in two cases: *a*) when there is only one staff; and *b*) when the note in question is the smallest (rhythmically) of all notes played by the different staves.[9]

OK, I can state *b*) now, with an *a posteriori* conceptualization. At the time I was struggling with 'bringing back TEX's glue', I could not possibly have thought of this as rigorously. Had I considered spacing for many voices, I myself would have probably abandoned the analogy with glue-spacing. But any thinking about many voices was then for me related to 'teaching music to TEX', which was a task I had decided to postpone. Thus I had automatically reduced the scope to one-voice situations — case *a*) — inadvertently making the intuition (and the analogy) true and useful.

An incomplete intuition generated an incomplete consequence: the analogy words = notes (which is right but not sufficient). But only through a further development of this consequence could I start imagining a complete strategy that would solve the *actual* problem of music typesetting, namely the problem of how to deal with polyphony.

6 The nature of musical typesetting (II)

> A page of music type consists of a great many small pieces joined together to represent continuous lines and characters. For instance, this group of notes  is composed of twenty-four pieces, thus:  .

This is how F. H. Gilson [2, p. 11] illustrated, back in 1885, the 'microstructure' of a musical text. In it we can see how much sense it makes to say that a note, in all its atomic character, is analogous to a word (rather than, for example, to a letter). Gilson analyzes his three notes into a series of smaller elements: these elements would be well thought of as letters. Indeed, Finale and Sibelius build up notes from several elements: note-heads, flags (𝅘𝅥) for short values, etc. So does MusiXTEX, whose fonts do not include complete notes at all.[10]

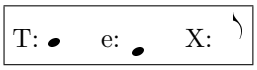


Table 1: Three musical 'letters'.

Notes, then, do seem to behave as words: aggregates of letters. For example, imagine a font including the three characters in Table 1. The word `TeX` would produce 𝅘𝅥.[11]

Now, there are some important things to note:

- Many letters of this musical 'alphabet' would have no width (such as the 'letters' `T` and `e` in the example), since musical 'words' would tend to be vertical rather than horizontal arrangements. In fact, probably the most appealing effect of 'rotating the score' (page 173) is that, when rotated, musical 'words' such as 𝅘𝅥 (now 𝅝) *really* resemble text. But with an unrotated score and zero-width characters, any disturbances of 'normal' vertical alignment (that occur frequently enough in music) would require the manual insertion of rigid space.
- In ordinary text, the space after a word depends on the space factor of *one* of its characters (generally the last one). In music, the space depends on the word — the note — *in its entirety*. The musical letter '𝅗𝅥', for example, will presumably be part of many notes, each of them requiring different spacing. But this is different from, say, what happens with character '.' in ordinary text: the spacing of the whole word is determined by that single character, *on its own and always in the same way*. In our musical font,

[8] I'm using the term 'word' in a TEXnical way: it means any succession of characters ('letters') bounded by spaces (or `\par`'s). So '`No!.`' is a four-letter word ('`!`' and '`.`' being two of the 'letters').

[9] In other words: when the note in question is the smallest in the whole polyphony, it follows that the next note will be present in the same staff. Then, the only consideration for its spacing is its own rhythmical nature — there will be no need to make room for intervening notes in other staves. For a 'special case' of the rule, pointed out by the authors of MusiXTEX, see note 20.

[10] With the annoying result that it is not possible easily to insert a note in ordinary text. Even the authors must have regretted it when they had to do heavy code for this relatively simple need when writing the program's manual.

[11] The vertical line missing here — called 'stem' — would be directly drawn by today's programs; Gilson included it as part of the types.

characters would have to be assigned different space factors each time they are used, which is kind of missing the point.

- The only difference between `T` and `e` (. and .) is the vertical position of the little ellipse. This is of course not the best solution — an infinite alphabet would be needed to put note-heads anywhere in the score. The musical alphabet should have only one letter for the note-head, and move it up and down as necessary.

Taking all this into account, our word would no longer be '`TeX`', but something along the lines of (in LaTeX) '`T\raisebox{-1ex}{T}\sfactor`\X X'. Much less attractive!

7 The advent of METAFONT

There are other reasons why this approach wouldn't work, but at the time I was unable to see them. I think it's the 'aesthetic' disappointment that left me unsatisfied. I rejected the approach, 'arbitrarily' if you will, by the feeling that an originally elegant use of TeX's properties had become a ugly series of *ad hoc* procedures.

I haven't spoken of METAFONT yet. But the truth is that, all along, I had been training in METAFONT — after all, I would soon be forced to create the fonts for my system. So every time I got tired of thinking of TeX, I would go for a METAFONT promenade. I was creating a musical font, imitating Finale's 'maestro'. Many of the symbols used here for musical examples actually come from those early, unexperienced trials. (That's partly why they are not very refined.) The point is that I was increasingly getting to know and love METAFONT, and many times I found myself, after having learned to take good advantage of a new resource, thinking 'Oh, if only TeX could do this...' This was mainly in connection to arithmetic and the use of variables, but in general a feeling was growing that METAFONT was curiously well fitted to the kind of graphic handling required by musical text.

What would have made one think that METAFONT would be fitter for music than graphic utilities? It was indeed curious. But then one day — it was the summer of 2002 — I made a slightly different comparison, and the whole business became defining and foundational. I would actually say that's the moment TeX*muse* was born. The point is not that METAFONT is fitter than PostScript, or than PSTricks, or than PiCTeX. *Truly* relevant is the fact that

METAFONT is fitter *than TeX*

I'm not claiming priority for this. Back in 1992, it occurred to Tom [Thomas E. Leathrum] that it might be possible to take advantage of the fact that METAFONT is *designed* for drawing things. [5, p. 1]

From this occurrence, Leathrum eventually developed the program mfpic.[12] It took me a relatively long time (probably because of my *idée fixe* of applying TeX's glue), but I finally realized the rather obvious fact that building notes up from elements was much more of a 'drawing-things' than of a 'compiling-words' business. Then, Leathrum's solution came to my mind automatically: building the notes is a graphic activity, and it should be done by METAFONT, not by TeX. The latter only gives directions.

8 TeX*muse*

So, that's it: TeX collects the information for the notes (how many note-heads, what kind, where to put them, the direction of their stems, additional signs, etc.), and writes a METAFONT program; running it, METAFONT creates a character for each note. After that, everything is trivial: each note will be like a single-letter word, with its own space factor. TeX applies its stretching algorithm and, almost without knowing it, spaces the music automatically:

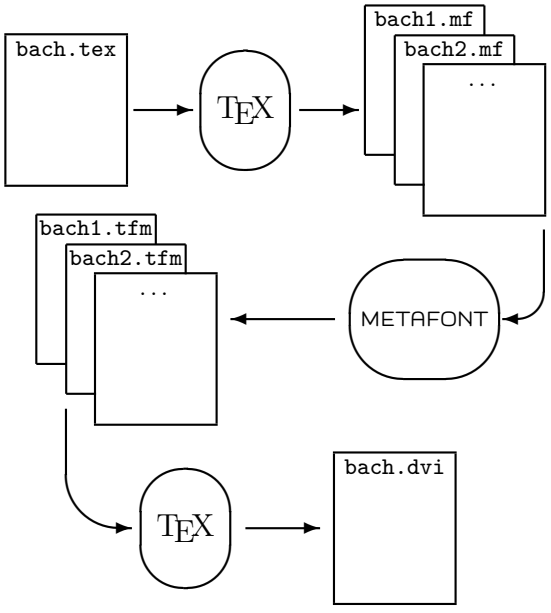


Figure 2: TeX*muse* at work.

[12] Ramón Casares too had a similar idea in 1994. His METATeX is a more direct application of it, different from mfpic (and TeX*muse*) in that it requires the user to know how to use METAFONT — it "only builds the necessary bridges to use TeX and METAFONT in a cooperative way" [1, p. 317].

With this idea, I finally had the outline of a system that seemed to work and fulfill the needs of musical typesetting. This outline subsequently succeeded at all the tests that had beaten all other systems and ideas (the many-voice problem, the four premises, and some other needs of music I haven't mentioned yet). I never performed these tests in any 'rigorous' way, but I qualitatively (and more or less intuitively) felt that the system would hold fair. That's when I went ahead and thought of a name, designed a logo, and wrote an article about it, toward the end of 2002. That article has formed the basis of the present account, and I've so far respected most of its contents and its structure, only struggling to make it more clear.

Let's have a look at how TEX*muse* deals with the mentioned challenges.

8.1 The many-voice problem

It has already been decided that TEX will be taught the secrets of musical rhythm. This means that, reading the horizontal input for each voice, it will know what notes start sounding at any given time — which, typographically, means knowing what notes, and in what staves, are present in any given vertical axis. So, in programming METAFONT, TEX is not limited to building up single notes in each character: instead, it is able to write a program that includes *all* the notes of a vertical axis in a single, rather tall character. Since TEX also knows what the next note is (anywhere in the score), it can figure out the shortest rhythmic value of the current character — which, typographically, determines the spacing and the character's space factor. A correct application of TEX's glue — case *b*) on page 174 — is then possible.

With this the journey is complete: from the wrong analogy 'words=measures', through the incomplete halfway-station 'words=notes', we have arrived at the solution 'words=moments of music'; from the score as a table exasperatingly input by columns, through the score as a table impossibly input by lines, to the score as a single line of text with words automatically designed.

This raises only a minor (but potentially devastating) concern: imagine a score with 24 staves, normal for a regular-size orchestra. A character including notes for 24 staves can be several inches tall, and will usually fill a page up. Is METAFONT really able to handle such tall characters? Luckily, METAFONT allows a maximum height of $2048pt\#$ [4, p. 316]: about 28.5 inches — enough for most needs.[13]

[13] The largest score produced by regular, home-computing, is usually set in pages of the 'Folio' size: 11×17. I have certainly seen larger scores (some, but only a few, taller than 28 inches). But to be noted is that *all* of them are set in handwriting.

The other restriction on character height is that it can be up to 16 times the design size. This could lead to fonts of design size as large as 72pt (for a 16-inch-tall score). I can't foresee any problem with that.

8.2 Premises and promises

The two middle premises (no Aristotelian pun intended) of section 4, namely TEX's glue and horizontal input, were the driving impulses behind the devising of the system in the first place — we can assume they are met. About 'only-TEX', OK, now METAFONT is involved, but we will all agree that there is no cheating there. Look at the root directory of the TDS (TEX Directory Structure): it's called `texmf`. METAFONT is 'part of the family', and in any case it was always intended to be the source of any associated fonts.

Flexibility, on the other hand, is much favored by a system that has no fixed alphabet of graphic elements but constantly creates its own. Thanks to this, the 'core' of TEX*muse* — what it will have built in — can reach a level of flexibility that ensures that only the industrious user will have to resort to lower-level programming. And then, good documentation and an intelligent use of literate programming for the METAFONT part of TEX*muse* (as well as for the TEX part, needless to say) will enable the industrious to further customize the system.[14]

8.3 Other needs

The main idea of TEX*muse* was found by realizing that verticality is, typographically, the main substance of music. But, musically, horizontal constructions are also of the utmost importance. The beams that connect eighth-notes, or the slurs and ties that group notes to indicate 'phrasing', for example, fall in this category. How is TEX*muse*, that builds characters vertically, going to deal with this?

The common feature of these 'horizontal elements' is that they are entirely defined by the last note involved. By 'remembering' which notes in which staves have to be included in the group (the beaming, the slur, etc.), TEX can postpone its typesetting to the last note. The METAFONT program for this last character can include the horizontal element, sticking out to the left. (This has in fact already been implemented for beams; slurs and other

[14] Is this not, after all, what happens with TEX itself? Most needs are already met, but in addition anybody could program his own, learning from existing examples, notoriously those of LATEX and its packages, and, of course, Plain TEX and *The TEXbook*. Look at me: I am a total amateur, I have never taken even a lesson in programming. But here I am, planning to teach music and METAFONT-programming to TEX.

elements are analogous. The 'remembering' takes place in METAFONT, which is much better at handling information.) Horizontal elements remain as such: they are not 'broken' into different pieces for different vertical agglomerates.

There are other elements that could imply a disturbance of the basic model of TEX*muse*. Notes are often 'adorned' with all kinds of symbols: the ♯ and ♭ signs to begin with, but also many different 'accents' (dots, lines, hats, ...), fingering indications, dynamic markings, etc. Many of them are actually vertically aligned with the notes, so that they can be included in the characters just as note-heads are — no special treatment is needed. But others go to the left or the right of the notes. The important implication of this is that these elements make spacing more complicated: in general, spacing is affected by them *only* if they generate collisions. In other words: if there is enough room for the element to attach to a note without colliding with the previous (or the next) one, it is simply appended; but if it doesn't fit in the available space, the note has to be moved (and, with it, all the notes in the same vertical axis).

TEX*muse* has to keep track of the width of the characters and the space between them. This provided, the task is trivial: the extra space needed can always be added to the right-most character of the potential collision. It will never be necessary to modify a previously built character (which was the potential threat to the whole system). This bears some relation to the matter of horizontal constructions treated above: there is actually no problem, because additions and adjustments can always be made in the 'current' character, without second thoughts about shipped-out notes.

9 Conclusion of the first part

The implementation of these two things — horizontal and offset elements — has involved a deep change in the nature of the whole system. But this was reserved for me to discover only when I started programming. We will see that many interesting things would happen in the process: I found unexpected and risible redundancies in my ideas (section 12.3), a particular problem that seemed relatively easy but is giving me trouble to this very day (section 14), and, most strikingly, a slip on the part of the Grand Wizard: a minor but unjustifiable omission in METAFONT, hard to believe coming from him. I shall have occasion to complain later (note 23).

What matters is that the phase of qualitative design was over: I had found a promising general idea and I had tested it with all that was available to me — all the challenges that had beaten other system and other ideas. TEX*muse* had survived, and it was now due to come into existence.

PART II
Toward and into implementation

In the original version of this article (written before any code whatsoever), this second part was devoted to a more detailed description of the planned workings of TEX*muse*, all in future tense. Discussed were the actual way TEX's glue would be applied, the handling of horizontal elements (beams, slurs, and the like), and the basic idea behind 'teaching music to TEX'. After that came an example, in which I played a computer running TEX*muse* on a hypothetical input, so as to illustrate the whole process.

Today, however, the actual activity of coding it all has revealed many 'flaws', to name them generically. They are all amusing. For example, influenced by Figure 2 above — and I don't know by what else — I was thinking that the TEXing of a TEX*muse* file would require three passes. I was even thinking about ways to let the user know that more passes were needed, and ways to 'fake' the final TEX box just as the `draft` option does for the graphicx LATEX package! Soon I was to realize that there was no need for all this...

The point is that in the process of coding I found these flaws, and the changes this has meant for the system are of far greater importance than many of the detailed descriptions of the original prospective account. This present second part will therefore omit many of those descriptions in favor of the latest developments. An original section on 'teaching music to TEX' stays pretty much the same; the section on spacing is still there, but essentially changed, since here the most important revision took place. Finally, I'm happy now to present a 'sample' instead of the old 'example' — now I *have* a computer running (part of) TEX*muse* on a real input. After all that, I present a totally new set of open questions, questions that were always present but have started to come to focus as their eventual implementation approaches.

10 Teaching music to TEX

The main consequence of the premise of horizontal output, as discussed in section 10, is that TEX has to understand the rhythmic characteristics of the user's input music. Thanks to that, it will be able to extract, from the input for each staff, the notes that need to be typeset in any given vertical axis (representing a 'moment' in the music).

Notes played at the same time have the same x coordinate (the different instruments in which they are played are represented by different y's).[15] All that TEX needs is to assign a value — the value of its x — to each note of each staff; then, it will sweep all these values, and build vertical characters with notes that have the same x. We are not yet dealing with 'material' dimensions, measurable, say, in millimeters. The actual horizontal placement of the notes in the staff is not a direct function of their x. So far the procedure gives TEX only the vertical correspondences between horizontal input.

A 'unit' has to be defined for this virtual axis. Since the rhythmically smallest note in usual practice is the 128^{th}-note (a note with 5 flags, 32 of which equal a 'crotchet' or quarter-note), a fourth of this value would be safe as a unit.[16] The quantum q is then defined as the duration of a 512^{th}-note. All other notes can be interpreted in terms of q. For example, the quarter note will be $64q$, the whole note will be $512q$.

Armed with these 'map of musical time', TEX reads the input a first time.[17] All the first notes in each staff occur at $0q$. From then on, according to the duration of the previous note, TEX finds the value at which all the notes occur. An example is the fragment in Figure 3. The first note (at $0q$) in the bottom staff is known to be a quarter-note (the user specifies this), and therefore the second note will be at $64q$. This second note is an eighth-note, so that the third one will come at $92q$. The count goes on, and then the procedure is done for the top staff, for which the values 0, 16, 32, 64, 72, 80, and 112 are found.

From this, TEX knows: there are notes in both staves for position $0q$; at $16q$ and $32q$, on the other hand, only the right hand has a note; and so on. The vertical correspondences are thus understood.

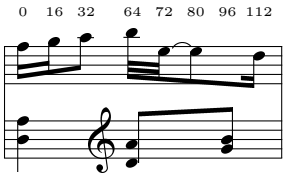
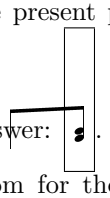
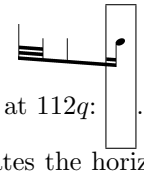


Figure 3: A fragment (m. 10 of Bartók's *Sonatina*) with its rhythmic 'quantization'.

11 Horizontal elements

Figure 3 features also instances of the most important kind of 'horizontal element': notes 2 and 3 in the bottom staff, or notes 1–3 and 4–7 in the top staff, are connected by 'beams'. How are beams created by TEX*muse*? A boxed insertion of the second-to-last character of Figure 3 disturbs the layout of the present paragraph, but it gives the key to the answer:  . This character is fairly tall — it makes room for the top staff, although there is nothing there (moment $96q$). But the important point is that it contains, sticking to the left, the *complete* beam that connects that note to the previous one (as well as the stems for both notes).

Since the stems of all the notes included in the beam are typeset when the beam itself is typeset, every note in a beam except the last one features only the note-head. For example, the character for $80q$ illustrates this (and also the procedure for musical 'ties', which is basically the same but simpler):  . This beam is completed only at $112q$:  .[18]

This is how METAFONT creates the horizontal elements. The other part of the problem is getting TEX to instruct METAFONT to do so. When TEX

[15] Here I am ignoring the dimension of 'pitch', and talking about the 'third dimension' of musical notation, as introduced on page 171.

[16] I have never seen a note of this kind in print, but it's good to provide for it. Besides, the smaller the unit, the easier and more precise will be irregular subdivisions (triplets, quintuplets, etc.), of frequent appearance in music.

[17] Here it is that a 'several-pass' system is implied: TEX reads the input a first time to deduce the correspondences, and a second time to actually build the characters according to those correspondences. In fact, the main programming mechanism for this is to define the same commands differently for each of the stages. Usual implementations of TEX today run METAFONT automatically when `tfm` or `pk` files are not present, so that the procedure does *not* require TEX to run twice on the file, as I first thought.

[18] You might be wondering about the different number of beams for different notes in the last illustration. This is another thing that depends on the rhythm of the notes involved, and this dependence is actually much more interesting and challenging to express algorithmically than the 'simple' matter of compiling the map of section 10. I have found an algorithm for this — a very nice one, I think, that uses METAFONT's weights and `undraw` technique — which I would like to present. Unfortunately, the mere explaining of the requirements would take so much space that it's hardly feasible.

On the other hand, a problem that I haven't solved is how to decide the angle of inclination of the beam. (In part, it's unsolved because I've been unable to find, explicitly stated, the very complex rules concerning this angle.) For the present examples I cheated, deciding the inclination on my own.

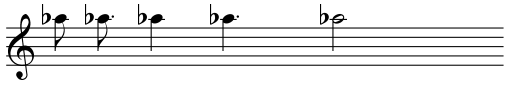


Figure 4: Different spacing after different note-values. (Excerpt from the violin part of Lustosławski's *Partita for Violin and Piano*.)

finds an opening '|' character in the user's input, the convention is that the following notes are to be part of a beam. So it suppresses the generation of stems, and starts building a list of the notes for the beam. When it finds the matching '|' (that closes the beam), the list is complete, and TEX passes it to METAFONT. Having kept track of exactly where all notes were placed, METAFONT is able to draw their stems and the connecting beam.

A paired '(' and ')' analogously indicates slurs (TEX starts a list of notes when it finds the first, and gives it to METAFONT when it finds the second); while '=' (as in Finale) indicates ties.

This is, in the main, how TEX*muse* generates horizontal elements from straightforward user instructions.

12 The spacing

Here we come to the heart of the matter. As noted before, the main motivation for this whole project was how to get spacing automatically without involving the user at all (unless he *wants* to be involved). It is finally time to put forward my solution.

Figure 4 will clarify — for the case of a single voice — what the goal is when talking about 'spacing'. The five notes involved are different from each other only in their rhythmic value, which increases from note to note: eighth-note, dotted-eighth-note, quarter-note, dotted-quarter-note, half-note. The spacing has to increase accordingly.[19]

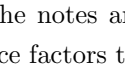
12.1 How to space the notes

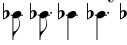
The first thing to do is to generalize the goal, rigorously, for the many-voices situation. How is a many-staff character to be treated, since it usually contains notes of different rhythmic values? For example, back in Figure 3, the first note contains a quarter-note and a 16th-note. Which one defines the spacing? (The answer might seem obvious, but we need to express it algorithmically).

Imagine the whole 'map' of musical time (as defined in section 10) collapsing into $y = 0$ — that is, the projection of the whole thing into the horizontal axis. (This projection, a series of dots scattered along a line, is a representation of what is known as the 'compound rhythm' of the entire ensemble.) Spacing is inferred from this projection. For every 'musical moment', TEX knows how long it will take for the next note to sound anywhere. Since between the first two notes of Figure 3 there are $16q$ from note 1 to note 2, TEX will treat the character as precisely that: as a $16q$ (a 16th-note).

But don't leap to conclusions too fast. Take what happens at $80q$: the next note comes in the bottom staff at 96, i.e., $16q$ afterwards. Therefore, in spite of being an eighth-note, this character will also be treated as a 16th-note ($16q$ long). It's not simply the shortest note present that determines spacing; it's the shortest note *in the compound rhythm.*[20]

12.2 Spacing them

Now that we know how to treat the notes, spacing-wise, according to their rhythmical nature, let's have a closer look at how TEX actually does it in Figure 4. The notes are  , and TEX knows what space factors to apply to them: the respective values are 1000, 2000, 4000, 8000, and 16000. Thus, the spacing required to fill out the line — i.e., the amount of stretching of the inter-note spaces — will be proportionally distributed in 1 : 2 ratios.

This alone doesn't work. TEX's algorithm is designed to *minimize* stretching for text, so the difference between spaces will be hardly noticeable. In fact, this is what you would get from regular TEX spacing:  . The difference is there, but it's by far insufficient for a musician's needs. So you need to 'fool' TEX, telling it the box is much wider

[19] There are several models as to how the space is a function of the rhythmic value. One is called 'Fibonacci', and works by assigning to each note a space equal to 0.618 the space of the immediately longer note. I tend to go for this model; but Figure 4 is actually set with a binary model, whereby each note receives twice the space of the immediately shorter one. This seems to work better when, as in the example, there are dotted notes. Obviously, this parameter will be user-modifiable in TEX*muse*.

[20] Under the heading "The spacing of notes", the authors of MusiXTEX write that "it can lead to interesting algorithms." For them, however, "it is not an important point in practice." Why? Because, owing to the fact that sometimes the spacing is not that of the shortest note present — as happens at $80q$ in Figure 3 — "the typesetter has to take care of good readable spacings on his own." [8, pp. 6–7] OK, to me this is plain upside-down reasoning: 'since we're leaving this task to the user, the algorithm has no practical relevance.' My preference would be something like 'since it is totally impractical for the user to figure the spacing out, we should go and look for the algorithm.'

Devising the algorithm is not even the real difficulty. As we saw, it's immediately solved by inserting 'in the compound rhythm'. The hard part is how to *implement* the rule in TEX. The quantized map of musical time offers a solution.

than it actually is, so that it will stretch more dramatically. For example, telling TEX the box is twice as wide gives .[21] In Figure 4 TEX was told the box was 2.5 times wider. The correct setting of this parameter, or more likely the dependence of this parameter on the music involved, is yet to be discovered.

Anyway, the same procedure took place in Figure 3 (this time the box was doubled in width), only applying space factors according to the compound rhythm.

12.3 But...

This looks great. It is as satisfactory an application of TEX's stretching algorithm as it can get. My main point is made.

And yet... it's redundant. For one thing, you'll have noticed that the in-text illustrations above lack a staff (the array of five lines). In the figures, the staff has been printed on top of METAFONT's characters with TEX's `\rule` commands. Since the beginning this was known to be a temporary solution. A professional system needs to provide for modifiable 'staff-ing', both vertically (more or less than 5 lines), and horizontally (a portion with 5 lines, followed by a portion without lines, followed by ...). The most reasonable solution is to attach the number of lines to each note: METAFONT will draw the necessary lines when drawing the note. But if the spacing is simply the stretching of blank spaces, who will draw the lines there?

More important, however, is that the drawing of horizontal elements involving many notes assumes, as explained in section 11, that METAFONT knows "exactly where those notes were placed". Otherwise it will be unable to draw slurs or beams that actually 'hit' the notes involved. I had concluded this already in 2002: "METAFONT needs a way to calculate *beforehand* the stretching that the construction of the line will perform. This amounts to implementing the relevant equations of the algorithm in a METAFONT function".

So I went ahead and did just that: I derived the equations (see appendix B), implemented them in METAFONT, and thus made it able to 'foresee' where TEX would finally place the notes. In this way, it knew how long to draw the staff lines, and where to find the notes under a beam.

OK, until one day something strange happens: I see that METAFONT's beams don't quite hit their notes. It took a lot of burdensome debugging for me to realize that, owing to a small mistake in the formulas, TEX and METAFONT were applying slightly different 'versions' of the algorithm. I even corrected my formulas diligently... Only a couple days later, I laughed when I noticed the real nonsense: the algorithm was being applied twice!

So today this has changed. It is METAFONT, not TEX, who applies stretching, according to formulas derived from TEX's stretching algorithm. The spacing is thus included in the font, and TEX limits itself to printing the characters out, one after another, without any space.[22]

12.4 Offset elements

The last aspect of spacing concerns elements that stick out to the left or to the right of their notes. Many musical elements do: an example is the treble clef 𝄞 in the middle of the bottom staff in Figure 3. It sticks out to the left of the next note, being separated from it 'rigidly'. If the note moves, the clef moves with it.

And, conversely, if the clef had to be moved, the note would move with it. That is how offset elements could affect spacing. In this case it does not, because the clef doesn't have to be moved — there's plenty of room for it. But sometimes an offset element will collide with something else, and then it has to be moved. Its note will also move, and, with it, all other notes on the same vertical axis.

This is implemented as follows: when METAFONT completes a note, it finds its right extreme r (with respect to the note's axis), and saves this value in memory. Then, when it is drawing the next note, it will calculate the left extreme l (of the current note, with respect to its axis).[23] As seen in the previous subsection, METAFONT will know the space s

[21] This example was achieved, in the present LATEX article, with `\makebox[2\width][s]`...

[22] This change actually raises a basic, qualitative question: in addition to the spacing of the notes, there are many other things whose handling can be done either by TEX or by METAFONT (for example, line breaking and justification). Deciding where to handle them amounts to deciding on the nature of TEX*muse*: is it a TEX package with a METAFONT underpinning, or a METAFONT package with TEX interface? Needless to say, the criteria for the decision are too subtle — it most likely will be an 'arbitrary' decision.

[23] Now, just how is METAFONT to find the right and left extremes of a character? When I faced this question, I went confidently to the index of *The METAFONTbook*: there had to be a function for this. Or at least one that would give me the width of a character. I was amazed to learn that METAFONT offers no '`widthof` ⟨*picture*⟩' function of any kind. The Grand Wizard introduces, in his 'Dirty tricks' appendix, an algorithm for METAFONT to find the extremes of a pen (this is necessary for some pen functions). The trick is truly dirty, and Knuth is justified if he takes pride on it. But does this not really hide an omission? I'm not the one to tell, but given the way pictures are internally represented in METAFONT, I would imagine that finding the extremes would be

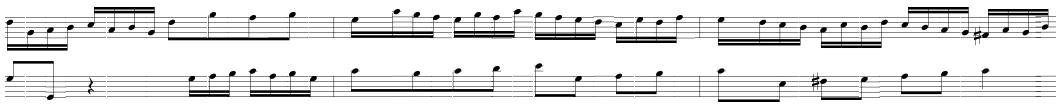


Figure 5: Mm. 2–4 of Bach's *Invention in C*, typeset by an incomplete TEX*muse* on Aug. 25th, 2004.

between these notes (more precisely between their axes) would be in the normal case, when no offset elements are involved and spacing is simply a result of rhythmic circumstances. It can then calculate whether this regular space will be enough to accommodate the notes (separated at least by a minimum 'framing space' f): is $s \geq r + l + f$? If it is, there's no problem; if it is not, the current note is moved to the right so that the collision will be avoided. Everything solved, the right extreme is calculated for the current note, and METAFONT is ready to go to the next.

'Moving the note to the right' means increasing its axis, which is shared by all other notes in the character. Since space can thus only *increase*, the adjustment can be done incrementally as offset elements in other notes are discovered to create additional collisions. The character is actually shipped out when all the notes have been added, all the collision tests have been made, and the axis has been moved as necessary.

13 A working sample

Figure 5 is a sample of what TEX*muse* is capable of (as of today, mid-2004). I've been actually developing the system with this sample — it's like a snapshot of TEX*muse*'s growth. The present article actually loads the current TEX*muse* version, and the input cited below is a verbatim copy of the commands used to generate Figure 5. I am willing to distribute the code to anyone interested.

Measures 2–4 of Bach's *Invention in C* were chosen for several reasons: rhythmic simplicity, one-voice-per-measure, no slurs or ties, no key- or time-signatures, no clefs. These things are not yet implemented; but none of them pose challenges, I simply haven't implemented these parts of the code (which are rather boring). *Challenging* things missing from the system will be mentioned in section 14.

an easy thing to program at the low level — just as METAFONT routinely calculates the 'total weight' of a picture, why should it not compute its extremes?

Knuth's roundabout method of finding the extremes of a pen can't be always applied, since a pen is a continuous and convex picture. Most pictures, and in particular musical characters, seldom fulfill those conditions. I had to develop a function from scratch (see note 25), but it turns out it's not right. A solution is yet to be found.

What I do want to show with this sample is that the system seems to be in fact possible. I would point mainly at the simplicity and naturality of the input that TEX*muse* demands from the user. Comparison is odious, and I shall refrain from citing the code that would be necessary in MusiXTEX to produce the three measures of Figure 5. But the point is important enough to ask MusiXTEX users to imagine it.

In TEX*muse*, all starts with the definition of the instruments involved. The system, as it stands today, rigidly assumes the top staff to be in treble-clef and the bottom one in bass-clef; of course, it will eventually provide for changes.

```
\newinstrument{righthand}
\newinstrument{lefthand}
```

Then, in the `texmuse` environment, actual music is input for each instrument. The initial string `DGAB`, for example, corresponds to the first four notes of the right-hand staff (whose standard names are, precisely, D, G, A, and B). Conventions are stated more fully in appendix A.

```
\begin{texmuse}
\meter44
\righthand{\rangefrom{G4}
    3|DGAB||CABG|\rangefrom{C5}|4DGFG|
    |3EAGF||EGFA||GFED||CEDF|
    \rangefrom{F4}|EDCB||ACBD||CBAG||\#FAGB|}
\lefthand{\rangefrom{G3}4|GG-|5R3R|GAB|
    |CABG||4CBCD||EG||AB|
    |CE-||\#F-G||AB|5C}
\end{texmuse}
```

Up to this point nothing has been typeset. The command `\musicbox` now tells TEX*muse* which instruments, and in what order, to build the fragment from. The user can thus use instruments flexibly, extract parts, re-order them, etc. There will also be additional musical-box commands, for example a `\musicparbox`, maybe a `\musicpage`, etc. That way, the layout is — for good — dissociated from the notes themselves.

In this case, we want a figure, so:

```
\begin{figure*}
\centering \musicbox{lefthand,righthand}
\caption{Mm.~2--4 of Bach's ..., 2004.}
\label{sample}
\end{figure*}
```

The result is in Figure 5. I'm satisfied with the way TEX is able to find all that it cares about, without making the user care about it too.

14 'The unanswered question'

In addition to obvious incompleteness, the sample shows signs of two particular problems that haven't been totally solved: the barlines, to begin with. Not only should they be drawn across the two staves (not just on them), but, if you look carefully, you'll notice that they have enlarged the spacing of the second note next to them. Barlines have proven a hard thing to implement, even qualitatively.[24]

The second problem can be seen in the last sharp-sign (♯): it features a spurious horizontal line. This is a result of METAFONT not having a direct way to find the extremes of its pictures (see note 23). To do that, the picture has to be made continuous, and therefore all 'additions' to the notes, such as offset elements like the sharp-sign, have to be joined to the note-head. The first ♯ also has this spurious line, only it's covered by the staff line. The horizontal line would of course be deleted after the extremes have been found — I haven't just coded that. The procedure strikes me as *ad hoc*, and I am definitely on the watch for better options.[25]

It was mentioned that TEX*muse* is not yet able to decide the angle of inclination for beams. There are other problems of this very interesting kind, that involve looking for (or deducing) the explicit list of requirements for good music typesetting: the most important ones are an algorithm to decide the order and placement of accidentals in a chord; a function for the ideal, context-dependent, ratio of space factor from rhythmic values to each other; a procedure to shift note-heads in chords with seconds; and the rigorous definition of the aspect of slurs and ties.

Another problem has to do with rhythms that are not binary: triplets, quintuplets, etc. There is an embryonic model for this, but it is as yet only qualitative: 'tuplets' can be treated as horizontal elements. Long pieces, on the other hand, will raise problems not yet dealt with, ranging from defining a concept of system-breaking (like a mixture of line-breaking and page-breaking procedures) to the good handling of space and justification of lines.

The most important issue is, however, keeping the design flexible. Up to this moment, only standard notation has been implemented — and there's still a long way to go with it. 'Programmability' is the only premise in which the sample shows no real, positive progress.

15 Conclusion

I hope this article has been interesting, or at least provocative. This report has to end here, but the project is only started. I look forward to any reaction from the TEX community, particularly of course from those interested in typesetting music with TEX. Just as happened with the first version of this article, the task of writing this has clarified many things to me. With the picture a little more clear now, I'm ready for a new session of actual coding. I'll keep in touch if anything interesting comes along.

References

[1] Ramón Casares. METATEX. *TUGboat*, 23(3/4):313–318, 2002.

[2] F. H. Gilson. *Music typography: Specimens of music types*. F. H. Gilson, Boston, 1885.

[3] Donald E. Knuth. *The TEXbook*. Addison Wesley, Reading, Mass., 1986.

[4] Donald E. Knuth. *The METAFONTbook*. Addison Wesley, Reading, Mass., 1986.

[5] Thomas E. Leathrum, Geoffrey Tobin, and Daniel H. Luecking. *Pictures in TEX with Metafont and MetaPost*. 2002. File `mfpicdoc.tex`, documentation to mfpic 0.6.

[6] David Salomon. *The Advanced TEXbook*. Springer Verlag, New York, 1995.

[7] Daniel Taupin. MusicTEX: Using TEX to write polyphonic or instrumental music. *TUGboat*, 14(3):212–220, 1993.

[8] Daniel Taupin, Ross Mitchell, and Andreas Egler. *MusiXTEX: Using TEX to write polyphonic or instrumental music*. 2001. File `musixdoc.tex`, documentation to MusiXTEX version T.98.

[24] There are in principle three ways of interpreting barlines in TEX*muse* terms: they could be treated as an extra note, 0q-long; they could be thought of as 'offset elements' to the left of the next note; or as 'offset elements' to the right of the previous one. Each of these approaches generates a series of side effects, surprising enough to make me unable to recall them all. The sample uses the second approach, and the repeated move of the note's axis confounds TEX*muse* about its real extremes.

This whole issue has led me to a recent rethinking of the whole process of drawing the characters. It's been decided that TEX (or METAFONT? — see note 22) 'orders' the elements of the notes, rather than following a first-come-first-drawn attitude: note-heads are drawn first, stems second, offset elements third, horizontal elements fourth, barlines fifth (or something like that). This ordering frightens me, though, because it might cut down on flexibility: what if the user wants something that cannot be easily classified and ordered?

[25] In fact, this solution is incomplete: there is no easy way of deleting precisely and only the line. I had imagined it would be done by `culling` appropriately (the line has a different weight from the signs), but it's much more complicated than that.

Appendices

A The user's input

The conventions for the input of the notes are designed to minimize the burden on the user's memory:

Rhythm is indicated through a convention standardized by Finale (and taken on by Sibelius): `5` means 'quarter-note' (crotchet); higher numbers are longer notes (`6`=half-note, `7`=whole note, etc.); and lower numbers are shorter notes (`4`=eighth-note, `3`=16^{th}-note, etc.). So, when TEX finds a `3`, it takes all following notes to be 16^{th}-notes, until a new number changes the value.

Pitch is given by upper-case letters following the usual note names: `A` through `G` (and optionally `H` for German users). Since there are many A's, many B's, etc., TEX needs a way to know which one the user means, which is achieved by `\rangefrom`. For example, `\rangefrom{C3}` means that the following notes are in the octave of the middle-C (which is in fact C3). If a note needs to be an octave higher or lower than set by `\rangefrom`, the modifiers `+` and `-` can be used (there's a `G-` in the left hand in the sample). `\rangefrom` can be used multiple times to set new ranges for different passages.

Rests are indicated by an '`R`'. (In the sample, there's a quarter-rest, and there should be a 16^{th}-rest, but I haven't yet created the METAFONT picture for it.)

Beams are set, as mentioned, by enclosing the involved notes between two '`|`'s. TEX*muse* inserts beams appropriately according to rhythmic nature. Eventually, an optional argument to `|` will be available that allows the user to beam individual notes to each other, across measures, staves, etc.

Slurs are not yet implemented, but will be produced by enclosing the notes between '`(`' and '`)`'. As in Finale, a '`=`' will create a **tie**.

Accidentals and **accents**, being of frequent use, are taken care of by one-character, visually-suggestive commands (such as `\#` for the ♯s in the sample).

License is given to the user about things like extra, meaningless and unintended blank spaces (that are disastrous for MusiXTEX), or where to type the rhythmic-value numbers (in the sample there are both '`3|`' and '`|3`', but both work the same way).

B Formula for the horizontal position of a character

This appendix explains how the formula for the final horizontal position of a character has been found (because METAFONT needs to know it, see section 12.3).

The musical line consists of single-character words, separated by stretchable spaces. The position before the n^{th} character in the line is then the sum of the widths of all the previous characters and the spaces after each of them. These spaces are stretched according to TEX's glue rules given in [3, pp. 75ff.].

The particular use TEX*muse* makes of this algorithm implies the following assumptions:

- The line is always stretched.
- There is no infinite glue (`\hfil`, ...) in the line.
- The normal interword space and the extra space of the font are 0 (this to allow dealing with space factors less and greater than 2000 with no change in behavior).

Let l_i be the width ('length') of the i^{th} character of the line, and g_i the space (glue) added after it. It is then clear that the horizontal position for the n^{th} character is given by

$$\sum_{i=1}^{n-1}(l_i + g_i).$$

Now, g_i (the space after the i^{th} character) is a normal interword space plus the additional space due to stretching. But the normal interword space is 0, so g_i is limited to the stretching. If the line has a natural width of X, and a desired width of W, $W - X$ is the total amount of space added to it by stretching, this is, the total sum of g_i's in the line. Of this total stretchability, each character receives a portion according to its space factor f_i, thus:

$$\frac{g_i}{f_i} = \frac{\sum g_i}{\sum f_i} = \frac{W - X}{\sum f_i}$$

But X, the natural width of the line, is actually the natural width of all the characters, since the normal (non-stretched) space between them is 0. So, $X = \sum l_i$, and therefore

$$g_i = f_i \frac{W - \sum l_i}{\sum f_i}.$$

'Expanding' this gives the sought-for position for the n^{th} character (with z being the last character):

$$\sum_{i=1}^{n-1}\left[l_i + f_i \frac{W - \sum_{j=1}^{z} l_j}{\sum_{j=1}^{z} f_j}\right].$$

Note that the result does not depend on the stretchability of the font, and that because of the assumptions there is no need to invoke either the concept or the rigorous definition of *glue set ratio* r.

◇ Federico Garcia
Music Department
University of Pittsburgh
Pittsburgh, PA
feg8@pitt.edu