GNU TeXmacs: A free, structured, wysiwyg, technical text editor

Joris van der Hoeven

Dépt. de Mathématiques (bât. 425)
Université Paris-Sud
91405 Orsay CEDEX
France

December 10, 2008

There is a common belief that wysiwyg technical editors are not suited for editing structured texts and generating documents with a high typographical quality. In this paper, we analyze the reasons behind this belief. We next discuss the program GNU TeX$_{\text{MACS}}$ and some of its innovations in relation to the difficulties of structured, wysiwyg, technical text editing.

1. Introduction

The introductions of popular books on TeX and especially LaTeX usually start with a celebration of the benefits of structured documents and generic markup. In this context, LaTeX is often compared to *wysiwyg* (what you see is what you get) text editors, which are claimed to concentrate only on visual design and not logical design. See for instance section 1.5 of Lamport's book on LaTeX for some widely accepted reasons why not to use wysiwyg editors.

In the past decade the empirical observation that it was difficult to write well structured documents with wysiwyg editors has made place for a doctrine that such a thing would be impossible. Many people adopted textual ASCII-based text editors like Emacs or Vi as a religion and claim that such editors would be the sole in which a structured text — a program — can be correctly visualized.

On the other hand, if systems like TeX and LaTeX are so much better than wysiwyg editors, why is it so that most people are still reluctant to learn TeX or LaTeX, and why do wysiwyg editors still dominate the market?

When we started to develop TeX$_{\text{MACS}}$ about four years ago, our main design aim was to solve this paradox, by creating a wysiwyg *and* structured technical text editor. The current version of TeX$_{\text{MACS}}$ can be downloaded from

`www.texmacs.org`

In order to design TeX$_{\text{MACS}}$, we first had to analyze the reasons why the wysiwyg editors from then were inadequate for the creation of structured documents. These reasons, which are often of a psychological order, will be studied in section 2. In a second stage, we observed that these problems were not inherent to wysiwyg editors, and we removed them by developing new editing techniques. These techniques will be presented in section 3.

Currently, TeX$_{\text{MACS}}$ is much more as just a technical text editor. For instance, it is possible to interface the program with several computer algebra systems. As its name suggests, TeX$_{\text{MACS}}$ has also been inspired by that Emacs editor. For instance, it supports the Guile/Scheme extension language. In section 4, we will shortly describe some of these additional features.

In section 5, we conclude with the prediction that structured wysiwyg editors will become very common within five years. Nevertheless, an important question is how this will happen, and whether some important lessons from the past will be retained.

2. Analyzing technical typesetting systems

2.1. Drawbacks of classical wysiwyg editors

When comparing classical wysiwyg editors with TEX or LATEX, the following drawbacks are often heard or implicitly felt:

Lack of primitives for creating structured text. Most well known wysiwyg editors support only a very limited set of such primitives (such as sections or HTML tags) and there is usually no macro expansion mechanism to create new user primitives.

Bad visualization of structured text. One solid common belief is that it impossible to adequately visualize structural markup using wysiwyg editors (assuming that such markup is supported). Indeed, when looking at document with a wysiwyg editor, you can not see *a priori* whether a section title corresponds to a LATEX-like `\section` command, or to a piece of text which has been typeset in an appropriate font.

Wysiwyg editors incite visual typesetting. Even if the above drawback were to be removed, one might still object that wysiwyg editors incite people towards visual and non structural typesetting. This thesis is supported by the ease with which it is possible to create visual markup in classical wysiwyg editors (and the difficulty to create structural markup).

Difficulties to create structured text. Another typical source of irritation is the omnipresence of fancy menus for symbols and mathematical constructions in wysiwyg editors: it is much faster to type `\frac` than to search for a fraction in a popup menu or on an icon bar. This disadvantage probably stems from the fact that, contrary to TEX and LATEX, the technical add-ons of wysiwyg editors have usually not been written by mathematicians, physicists or computer scientists who *use* their own program every day.

Wysiwyg editors are complex and slow. The design of wysiwyg editors being much more complicated than a mere typesetting program like TEX, such editors usually need much more resources, both in memory space and computation time.

Bad document formats. Many wysiwyg editors use proprietary data formats which are not readable by humans; some of these formats are even patented and one might question whether the user of such software is the owner of the documents he creates. In any case, text based document formats have the great advantage that you may edit them with virtually any tool, so that you do not necessarily need the original wysiwyg editor to modify them.

2.2. Towards criteria for analyzing the quality of text editors

The drawbacks we have mentioned are actually related to a certain number of criteria, which may be used to analyze the quality of a text editor or an electronic typesetting system. Again, we made up a non exhaustive list of possible such criteria. In order to be fair, we now also included a certain number of criteria which plead in favor of wysiwyg editors.

Support of structural markup. We will not explain the numerous advantages of structural markup here, since standard books on LATEX do this very well. Support for structural markup occurs at two levels:

Primitives. Support of some fixed primitives for structural markup, as in HTML.

Extendibility. Support of a macro expansion system to create new primitives and style files.

User friendlyness. The user friendliness of a typesetting system involves many aspects, some of which we list here:

Simplicity. The system should be sufficiently simple to use. This implies that the user interface should be standard, that the basic editing primitives should have a clear meaning, etc.

Comfort. The system should be comfortable to use. This implies for instance that the editor should have a high reactivity, that is a fast response to user events. By preference, it should not be necessary to manually rerun a typesetting engine after each modification. The readability of text on the screen is also an important and underestimated factor, which may avoid headaches and such. More generally, one should strive for maximal ergonomics.

Distraction. Just like a well typeset document allows the reader to concentrate himself on reading the actual content of a document (and not be distracted by typesetting issues), a good editor should allow the user to concentrate on the actual document (and not be distracted by programming issues and such). One main advantage of using an editor like TeXmacs is that you can do mathematics *while* typing a text.

Documentation. Well documented programs help the user to progress, whatever his level, and in any direction. It is preferable that the documentation comes with the program itself and that the program is self documenting (for instance, you learn about appropriate keyboard shorthands when using the menus).

Adequacy. Any program has one or several main objectives, and it is interesting to analyze the adequacy of the means provided by the program to reach these objectives.

Controlability. In the worst case, a program provides certain features which are so difficult to exploit that they become useless. This may for instance be the result of an inadequate user interface. Document formatters like TeX provide full control because of there textual document format. Yet, their controllability is not always optimal: think about inserting a new column in a complex table.

Faithfulness. It is also important that the user of a program obtains the result he expects. From this point of view, the wysiwyg-ness of an editor is an advantage.

Transparency. In order to have full control over a program, the user needs to understand at each moment what is going on. In particular, the structure of the document should be transparent for the user. In the case of a text formatter, this is ensured by the fact that you explicitly see all structural markup. In the case of a wysiwyg editor, one needs other mechanisms to ensure transparency.

Quality of the endproducts. Usually, the endproduct of a text editor is a printed document and it is natural to require the typographic quality of such a document to be optimal. Nevertheless, we may also see the structural richness of a document as a part of its quality, since this richness makes the document more or less adequate for future reuse, for instance as part of another book or a website. It is important that an editor makes it possible to produce high quality documents in our broader sense, and that it encourages the user in writing such documents.

Document format. The choice of a good document format is an important issue for the quality of a text editor. Good document formats should meet several criteria:

Readability. It is preferable to use a text based document format, so that the documents may in principle be edited by hand.

Simplicity. If the document format is also sufficiently simple, than it will indeed be able to edit documents by hand. Simplicity also makes it easier to write converters to other formats.

Comptability. If possible, one should opt for a format which already exists, or which is a particular case, an extension, or a modification of an existing format.

Permanence. The document format should not change in a predictable way and not to much from one version of the program to another.

Evolutivity. It is a major advantage of a program to be extendable and that it can be adapted to other needs in the future. This has been the case for TeX, which gave rise to LaTeX and many other clones. This is also the case for an editor like Emacs, which provides a so called *extension language*.

Freedom. From a scientific point of view and also in relation with many of the previous criteria, it is very important that programs and data formats are *free*. Here we mean free in the sense of freedom, not price; for a more precise definition, we refer to

`http://www.gnu.org/philosophy/free-sw.html`

The freedom of software is essential for its development in a scientific way, as has been the case for a system like TeX. It favorites for instance the evolutivity and the accessibility of the software and the readability and permanence of the data format.

3. Techniques for structured wysiwyg editing

In this section we discuss several new techniques, which have been implemented in TeXmacs in order to fulfill the criteria stated in the previous section. We focus on the techniques related to structured wysiwyg editing, without striving for exhaustion.

3.1. Structural markup

In order to implement structural markup, one first has to analyze the different types of structural markup that may occur. We have already distinguished between editors which provide a limited set of basic primitives like the `strong` tag in HTML and editors which enable the user to define his own macros.

A second distinction concerns the nature of the structural markup during typesetting. Simple structural markup, like the above `strong` tag, only locally affects the typesetter. More complicated markup, like an enumeration tag, may necessitate all text after the tag to be retypeset each time we modify it. Changes in the most complicated forms of structural markup, like references or fields of a spreadsheet, may require the whole document to be retypeset.

A third distinction concerns the level of interactivity of the structural markup. In the case of complex markup like references or tables of contents, it is reasonable to perform the necessary recomputations on explicit request of the user. For computational markup, like Java scripts, it is reasonable to implement a mechanism to deactivate and reactivate the markup. For simpler markup, like the `strong` tag, it is nice if the arguments of the macro can be modified in a more direct way.

Currently, TeXmacs supports all of the above mechanisms. Likewise TeX, the editor transforms the logical source tree into a physical box tree. However, in our case, the boxes contain much more information. In particular, whenever possible, they contain the location of the corresponding code in the source tree. More details about this correspondence can be found in the help about the implementation of TeXmacs.

As to user macros, we both provide real *macros* and *lambda expressions*. When expanding a macro, the arguments of the expansion can be modified interactively by the user. For instance, the LaTeX `\section` command is a macro in TeXmacs. From the technical point of view, the typesetter remembers the source locations of the arguments of the macro expansion, which are necessary to associate source locations to the final boxes.

In older versions of TeXmacs, we only provided lambda expressions for the creation of user macros. The applications of lambda expressions have first to be disactivated, before they can be edited, and reactivated as soon as you are done. Lambda expressions are still implemented in order to support computational markup like increasing a section number.

Remark 1. It is also interesting to analyze the difference between TeX/LaTeX and XML from the point of view of structural markup. Indeed, a TeX document is very sequential in nature, because it is close to a program. An XML document is much closer to a tree, which may be traversed in parallel. One typical consequence of this difference is that, from the XML point of view, something like a table of contents is generated by a piece of an XSL style file which is independent from the sectional macros. On the one hand this independence is attractive; on the other hand the approach is less "object oriented" and breaks the "orthogonality". For instance, a new user-provided sectional macro based on a standard sectional macro will not be taken into account when generating a table of contents.

3.2. Efficiency of the typesetter

One major problem of wysiwyg (structured) editors is speed, although the constant improvement of CPU's plays in favor of such editors. The two main sources of difficulty are the speed of the typesetter and the speed of displaying typeset text. As a main objective, the required *reactivity* of the editor (i.e. the response time to user input), should be small with respect to the typical time between two keyboard hits.

In order to reduce the typesetting time, our main approach is to retypeset a minimal amount of text at each modification. For the simplicity of the design of the typesetter, we have nevertheless decided to retypeset the whole paragraph whenever a part of needs to be retypeset. This is reasonable, because it will at least be necessary to recompute all line-breaks; this recomputation usually necessitates a time which is proportional to retypesetting the whole paragraph.

The more complicated question is how to predict which paragraphs have to be retypeset, especially in presence of dynamic environment variables like section numbers. Our algorithm is as follows: we need to retypeset at least all paragraphs where a change occurred. For each paragraph we also remember all *changes* of the environment variables (like counters of section numbers) before and after the paragraph. When retypesetting a succession of paragraphs, we compare the new changes with the former changes. If the changes remain the same, then it is not necessary to retypeset the subsequent paragraphs.

Our strategy may be refined in the future in two ways. First of all, even when some environment variables change, it is only necessary to retypeset the paragraphs which effectively involve these variables. Secondly, one might decide to only retypeset the document up to those paragraphs which are visible on the screen.

As to the speed of redisplaying modified text, we apply the classical strategy of using a second invisible screen buffer in order to reduce flickering: when the whole text has been redisplayed, the second screen is copied to the visible screen. Actually, we start by redisplaying the text which is closest to the cursor and end with the text which is the farthest away. Whenever a keyboard event or a mouse click occurs, redisplaying is interrupted and only the available text is copied from the second screen to the visible screen. This allows the user to see the updated text close to the cursor in the case when he types very quickly.

3.3. User interface

A less complicated, yet very important issue is how the user may create structural markup. In the design of the TEXMACS user interface, we have followed the principle of redundancy: some users prefer toolbars or menus, others prefer to use the keyboard. For instance, a fraction can be obtained in the following ways:

1. By pushing the “fraction icon”.
2. Via the “Insert” menu.
3. By typing the TEXMACS keystrokes `Escape F`.
4. By typing the TEX command `\frac` and pressing `Enter`.

Furthermore, TEXMACS attempts to teach users about these alternative ways via messages on the footer and help balloons. In this way a user can start by exploring the possibilities of TEXMACS via the menus. As soon as he often needs a certain feature, he will naturally learn a keyboard shorthand for using this feature in a more efficient way.

Actually, the TEXMACS user interface provides several additional tricks to maximize the user's efficiency in entering texts. For instance, in order to obtain the mathematical symbol α, it suffices to type `Alter-a`. In order to obtain $\rightarrow$, $\oplus$ or $\leqslant$, you may use the keystrokes `- >`, `@ +`, resp. `< | =`. The combination of these tricks and the self documenting properties of the user interface make it very simple and efficient to learn and use TEXMACS, in comparison with more classical programs like TEX and LATEX.

In order to obtain a well structured document as an endproduct, we already mentioned the fact that a good editor should encourage the user to structure his work. In TEXMACS this is accomplished by suggesting the reader to first choose a document style (without which he will not be able to do much), or by choosing one for him. Next, the user interface naturally depends on the chosen document style and on the editing mode (math mode, preamble, etc.). Furthermore, structural markup is always put in a prominent place.

A more complicated problem, which has not completely been (and probably never will be) solved is the standardization of structural markup. One indeed has to be careful not to provide a too abundant number of macros in new styles: this may result in the users getting lost, so that they will prefer to fall back to more visual markup.

3.4. Transparency and controllability

Another interesting question is how to make the structure of a document transparent inside a wysiwyg editor and how to give the user full control over this structure. This question is mainly a psychological one: the user has to *feel* the structure in a way which is closest to his mental representation and he has to have the *impression* that he has optimal control over his document.

From our point of view, showing an ASCII text which represents the document in coded form is not necessarily optimal for both of these purposes. Indeed, an expression like

```
\frac{1}{\alpha_1+\frac{1}{\alpha_2+\frac{1}{alpha_3+\ddots}}}
```

is not clearer from a structural point of view than its graphical representation

$$\frac{1}{\alpha_1+\frac{1}{\alpha_2+\frac{1}{\alpha_3+\ddots}}}.$$

Similarly, the user does not necessarily have optimal control: what about inserting a new column in a complex TEX table?

Of course, there are also examples where the structure of the document is *a priori* more transparent and controllable in its coded form. A good example is the piece of LATEX code

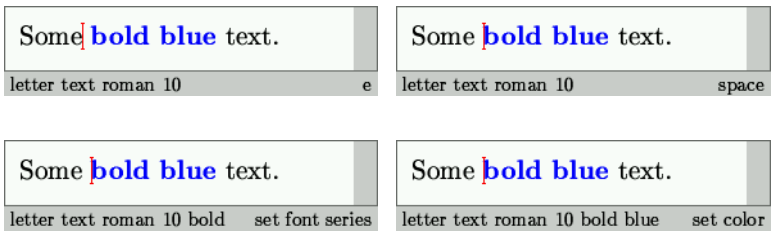
```
Some \textbf{\red{bold blue}} text.
```

In a wysiwyg editor, it is not clear whether the text "bold blue" is bold and blue or blue and bold. Furthermore, how to put the cursor in a position where it is possible to type bold but not blue text?

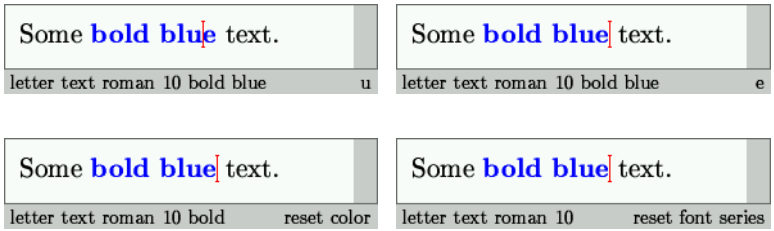
In the TEXMACS implementation, we were guided by the opinion that the global structure of a document is clear from its graphical appearance, and that the only place where we need more information about the structure is at the current cursor position. Consequently, the footer of the editor both displays the physical and logical properties of the text at the current position. To a lesser extent, the shape and color of the cursor itself indicate some structural properties.

As to the cursor movement, TEXMACS implements a graphically natural (and not structurally natural, like many other wysiwyg technical editors) cursor movement. In other words, if you press on the right arrow key, then your cursor may be expected to go to the right. More precisely, when moving the cursor around, the user modifies the position of a so called *ghost cursor* which may be any position on the screen. The real cursor is chosen as close as possible to the ghost cursor, in a way that "makes sense".

Moreover, the cursor position is determined by an x and a y coordinate as well as an infinitesimal horizontal δ-coordinate, which allows the user to position the cursor in a very precise way in presence of structural markup. The figure below illustrates the use of this coordinate in the above example of bold blue text. We start with the cursor being positioned after the 'e' in 'Some' and show what happens if one pushes thrice the cursor key '$\rightarrow$'.



Similarly, the following figure illustrates what happens when the cursor is initially positioned after the 'u' in 'blue:



4. Additional features of TeXmacs

Let us now briefly mention some additional features of TEXMACS, which are not related so to say to structured wysiwyg editing, but which do make the system interesting in comparison with similar existing software.

4.1. Computer algebra systems

It is reasonably easy to interface a computer algebra system with TEXMACS. Such interfaces already exist for the systems Pari GP, Macaulay 2 and Gtybalt. More interfaces are currently under development. In this context, TEXMACS can both be used as a graphical front-end to the computer algebra system and as a text editor, which enables the user to directly include the results of his computations in an article.

The objective that it should be possible to interface TEXMACS with computer algebra systems has some repercussions on its design. For instance, formulas should have semantics in a natural way. We adopted the strategy that an intelligent parser should be able to assign a semantics to a formula. For this reason, we demand the user to explicitly type the multiplication symbol, since in the TEX formulas

```
$a(b+c)$ $f(b+c)$
```

it is not clear *a priori* when we implicitly understand a multiplication to be present. Another problem when interfacing TEXMACS with a computer algebra system is that automatically generated formulas should be typeset in a satisfactory way. This necessitates new hyphenation techniques, which are still under development.

4.2. Typesetting innovations

It is often believed that TEX is the ultimate program for professional typesetting. Indeed, TEXMACS incorporates many techniques from TEX. Nevertheless, there are a few points where we think that we made useful innovations:

- We use a global algorithm for page breaking.
- Consecutive lines, such that the upper line descends to much downwards and the lower line ascends to much upwards (at different places), are "crunched together" as far as possible.
- TEXMACS both supports vertical space before and after a paragraph; the maximum of both is taken. The vertical space between two consecutive theorems without proof is not doubled.

Several other optimizations can be foreseen and the "technical art" of electronic typesetting is fortunately not yet dead.

4.3. Extension language

In a similar way as Emacs comes with the Emacs/Lisp extension language, TEXMACS provides Guile/Scheme as an extension language. At the moment, the extension language is mainly useful for programming the keyboard shorthands and the menus of the user interface. We also implemented regions of text with an associated scheme program, which is executed each time one clicks on the region.

In the future, other extension languages like Python or Caml might be supported. We also plan to use them for other purposes, like executable markup (similar to Java scripts). Executable markup may actually occur in several ways: as scripts, as formulas in a spreadsheet, or as executable enhancements of a style file.

5. Conclusion

We hope that the present paper (or, even better: the actual TEXMACS program) convinced the reader that structured wysiwyg editing of mathematical texts is possible and desirable. A careful analysis of the psychological factors that lead to the rejection of (structured) wysiwyg editors in the past makes it possible to eliminate these drawbacks in the future. Despite a few temporary shortcomings (mainly speed), TEXMACS presently shows that it indeed possible to efficiently edit structured documents in a wysiwyg manner.

Actually, we conjecture that, within five years, most mathematicians, physicists, computer scientists, etc. will use wysiwyg editors to write their documents. Furthermore, such wysiwyg editors may grow into real "technical office platforms", capable of producing and visualizing documents or web pages, and interacting with computer algebra systems and numerical analysis programs.

In our opinion it is important that such tools can be used freely by scientists and others. In this light, TEXMACS is not just a structured wysiwyg editor: it is also a *free* editor. The fact that TEX and LATEX are free programs made it possible to share knowledge about techniques for beautiful and efficient electronic typesetting, and many people contributed to their development with pleasure. This has been a good tradition: we have to ensure that the benefits of TEX and LATEX will live and honor the scientific tradition of free exchange by carrying it on.

P.S.: the present document was written using TEXMACS.