A Future for TEX

Roger Hunter
TCI Software Research, Inc.
1190 Foster Road
Las Cruces, NM 88001
USA
Internet: roger@nmsu.edu

Abstract

The future of TEX is invisibility. The role of TEX should be similar to that of the microprocessor in a PC. The microprocessor is the heart of the system, but is completely invisible except for the sticker which says "intel inside." TEX must be made invisible with appropriate front-ends. These front-ends should emphasize the manipulation of content over appearance and reverse the trend toward WYSIWYG (What You See Is What You Get) interfaces with their emphasis on manipulation of appearance. Content-oriented interfaces provide far greater user productivity than WYSIWYG systems, and TEX is the ideal basis for such systems.

Introduction

TEX has a guaranteed future only if its use grows significantly. That growth can occur only if TEX is made much easier to use than it is now. Back-ends to TEX are necessary for any form of output so there are many of them. There must be strong pressure to create front-ends that make TEX much easier to use. The onslaught of WYSIWYG clickery makes the survival of TEX entirely dependent on good front-ends.

The Good and the Bad

Listing the good and bad features of TEX seems to be a favorite pastime of TEX lovers, and I am no different. The main difference in my list is that features often considered advantages are listed as disadvantages. First, the good features of TEX. The primary goals of Don Knuth's original TEX project head the list.

1. TEX produces superb output. This was Don's primary motive when he set out to create TEX.
2. TEX source is archival. The documents are in a standard ASCII form. The TEX language provides a linear, ASCII form which can be used as a standard for storage and interchange.
3. TEX is available on most platforms. This, together with its archival nature, ensures that TEX documents can be created and used anywhere there is a reasonably capable computer.
4. Many scientific journals accept compuscripts in TEX and provide style files.

And now the disadvantages.

1. The TEX language is a compromise. It has been said that the TEX language is understandable by everybody because instructions are written in plain English, not undocumented numerical codes. If Don Knuth had felt that the TEX language was the way we should read and write mathematics, then there would have been no need to create TEX, the program. Simply specifying the language would have been enough. The only justification for the form of the TEX language is as a linearized portable input to TEX, the program. In its present form, it is a compromise between the need to provide some support for direct entry and the need to process the result by computer. It would be wonderful to remove this compromise in favor of computer processing, but it is probably much too late.

   The two-dimensional mathematical notation evolved because it optimizes the use of the high-bandwidth human optical system. There are many mathematical expressions which are virtually impossible to grasp in TEX input form — all are instantly comprehensible in TEX output form.

   Publishers had hoped that TEX would be the solution to the rising cost of typesetting, and now they are not so sure. A major reason for this is that authors do not write style-independent TEX code. Leslie Lamport defines and discusses visual design and logical design in the LATEX *User's Guide & Reference Manual.*

Logical design is the key to good TEX documents, but without a way to enforce it, most authors end up with a large component of visual design in their code. This is a nightmare for publishers who need to typeset using a specific style, and is the main reason why author submissions are so costly. It invariably costs more to use the author's original TEX document than it does to re-key the entire thing. If the publishers will not champion TEX, the future cannot be bright. Stamp out abominations like `\it word \rm`!

TEX will have no future unless authors are completely isolated from the TEX input language. Although it is archival, the TEX language is unfit for humans. A proper front-end eliminates these problems.

2. TEX is in the public domain. As wonderful as this may seem at first, it means that now that its creator has stopped working on it, everybody wants a say. We have gone from a committee of one to a committee of the entire world. How much progress can a committee of this size make?
3. TEX is extensible. This is marvellous for developers of macro packages and styles. It is a disaster in the hands of authors. Authors delight in creating new sets of macros and in using them inconsistently in a document. Publishers find it much cheaper to re-key an entire document than to rewrite an author's macros to fit a style.

   The policy of the American Physical Society, the American Institute of Physics and the Optical Society of America on submitting documents is the right one. To have a paper accepted, you must use the REVTEX styles, but most importantly, you are prohibited absolutely from defining and using macros. Amusingly, the REVTEX guide carefully explains that there are two classes of macros, and then states flatly that you cannot use either kind!
4. TEX is stable and unchanging. Whatever the arguments or the reality, making this statement gives TEX a dead feel. Even if TEX itself does not change for the forseeable future, the continued development of packages like LATEX3.0 provide the necessary life. This is mostly a matter of public relations.

TEX Must be Invisible

TEX is the microprocessor, LATEX is the operating system, and appropriate front-ends are the application programs. Just as the average user has no need to know how a microprocessor works, and a user of an application program needs only a rudimentary knowledge of the operating system, the average user should never be exposed to TEX. These days, most drivers of cars do not know how an engine works. Although knowing how an engine works may somehow make you a better driver, requiring that you know how an engine works would be ridiculous. It would ensure that most people would not drive. We have roughly the same situation for TEX. Requiring that users know TEX will ensure its demise. A technology is mature when most of its users do not know how that technology works. We should strive to make TEXnology mature.

The High Cost of Visual Design

A recent study estimates that 2% of the United States gross domestic product is lost through unproductive use of computers. At the head of the list of offending behaviors is "font futzing" — endless fiddling with the appearance of a document. A section head at Sandia National Laboratories told me that his researchers spend huge amounts of time preparing reports using WYSIWYG Windows word processors. They spend most of the time changing fonts and page layout. When the documents are submitted, they must be reformatted to fit the required style. The process takes hours because all of the formatting is local and visual.

The same effects exist in the TEX world. Most of us are familiar with people who fall in love with TEX and run around saying, "Look at this incredible effect I just produced" or, "Look at this fantastic macro I created." Highly paid professionals endlessly playing with TEX macros to get just the right visual effect are wasting their time doing work that is unproductive and for which they are not trained. The only way to avoid this problem is to provide a front-end which enforces or strongly encourages the principles of logical design.

Interface is Everything

Given that invisibility of TEX is essential and that logical design has a large productivity payoff, interface is everything. Attractive interfaces are the reason WYSIWYG word processors are simply taking over. They are addictive. Their addictive nature and their total focus on visual design makes them one of the most insidious productivity sinks in existence today. The salvation of TEX lies entirely in the development of good interfaces, and those interfaces must encourage and, if necessary, enforce logical design over visual design.

The main reason for using TEX instead of one of the leading word processors is to obtain the far superior output. Because there are no alternatives, you are willing to put up with the input language. The situation with symbolic systems like Maple and Mathematica is the same. The benefits of these systems must outweigh the disadvantages of their unnatural user interfaces before someone will choose to use them. This restricts use to a tiny fraction of the potential audience. By making the interface much better, the number of TEX users could be increased several orders of magnitude. The same argument applies to Maple and Mathematica.

The Right Interface

The essential features of a good TEX interface are as follows:

- The interface must encourage authors to work directly at the computer.
- The interface must encourage logical design over visual design.

The penalty for using the computer over the blackboard or a pencil and paper should be minimal. The language you use to read and think about your document should be the language you use to enter it into the computer. The time taken entering TEX codes is wasted time which could be used for developing the content.

The current crop of Windows-based word processors (Word for Windows, Ami Pro, and Word Perfect are the three most popular) define WYSIWYG. The essential feature is visual design. One manifestation is that you are encouraged to interact with an image of the printed page. Another is that you select text and give commands which determine the appearance such as the font face, point size, and weight.

The fact that all of the best-selling word processors use a WYSIWYG interface has lead to the perception that there is no other way. In fact, the use of a GUI (Graphical User Interface) has become synonymous with WYSIWYG. The result is that millions of people are forced to view crude representations of the printed page through screen windows which never match the pages. At the same time, they have come to spend much of their time at the computer worrying about page layout and typography.

Interfaces which emphasize logical design provide a much better way to create, edit, and interact with documents. The main features of a logical interface are as follows.

- Lines are broken to the screen window.
- You select text and designate it as a section head or apply an emphasis.
- Fonts and colors used on the screen are chosen to maximize screen readability and are independent of the choices made for the printed output.

Just as there is a perception that GUI implies WYSIWYG, there is a corresponding perception that logical implies linear. People seem to think that an interface which uses logical design requires that you enter obscure codes to get the results you want. The primary example in the TEX world is the notion that using the TEX input language directly is the only right way. This is simply false — it is possible to create a logical interface which displays and has you interact with mathematics in its natural (TEX output) form.

Some Interface Issues

TEX is a batch system. There are a number of interesting problems which arise when you consider implementing a much more interactive system.

The first problem has to do with TEX's line breaking algorithm. I have often heard people say that the ultimate system would allow you to interact with pages in the way you do with a WYSIWYG word processor, but the page layout would be updated instantly using TEX. Even if you translate this desire to a logical system, there are drawbacks. For example, you could be typing or editing toward the end of a paragraph and have all of the lines above you in the paragraph jiggling about as you type. This is because TEX's line breaking algorithm can change the breaks throughout a paragraph when you make a change anywhere in the paragraph. The effect could be very distracting.

Another question which simply doesn't arise in a batch system has to do with spaces. Who owns the spaces? When TEX puts extra space around operations, relations and punctuation in batch mode, the question makes no sense. When you are dealing with an interactive system, the insert cursor must be placed somewhere, and the choices made have a significant effect on the feel of the system. For example, where should the cursor be placed as you move through the expression $x + y$? TEX inserts extra space around binary operations. Should the cursor position between x and + be next to the x, next to +, or somewhere in between? If you take the position that the + owns the extra space, then the cursor should be placed next to the x. This seems like a very minor point, but it has a large effect on the feel of the system.

Blue Sky's Lightning Textures provides a way to enter TEX codes and see the resulting TEX output

almost instantly. This is a completely different approach which I view as complementary to the interface I have described. Lightning Textures provides the greatest value for typesetters and others performing high-end layout work. The interface I have described is meant for authors.

Scientific Word

Scientific Word is a Windows-based scientific word processor based on the principles that I have mentioned. It provides a logical interface to documents and stores LaTeX files. It includes Richard Kinch's TurboTeX for previewing and printing.

Experience with users of *Scientific Word* has been very interesting. Initially, many users feel extremely uncomfortable with the fact that they are not interacting with a page image. They spend a great deal of time previewing to see if they really will get the results they want. As they continue to use the system, the frequency of previews decreases. Once they have learned to trust the system, they relax and focus on the content exclusively. Only in the final stages do they concern themselves with the printed form. The habits developed by using WYSIWYG systems are difficult to break, but once they have been broken, users realize how much more productive they can be.

Direct interfaces between *Scientific Word* and symbolic systems are also being developed. An experimental version of *Scientific Word* lets you interact directly with Maple using the same principles employed for TeX. Maple's input language is invisible in this system — the notation for input and output is the standard, natural, mathematical notation.