E-TEX: Guidelines for Future TEX Extensions

Frank Mittelbach
Electronic Data Systems (Deutschland) GmbH
Eisenstraße 56, D-6090 Rüsselsheim, Federal Republic of Germany
Tel. +49 6142 803267
Bitnet: mittelbach@mzdmza.zdv.uni-mainz.de

Abstract

With the announcement of TEX 3.0, Don Knuth acknowledged the need of the (ever growing) TEX community for an even better system. But at the same time, he made it clear, that he will not get involved in any further enhancements that would change *The TEXbook*.

TEX started out originally as a system designed to typeset its author's own publications. In the meantime it serves hundreds of thousands of users. Now it is time, after ten years' experience, to step back and consider whether or not TEX 3.0 is an adequate answer to the typesetting requirements of the nineties.

Output produced by TEX has higher standards than output generated automatically by most other typesetting systems. Therefore, in this paper we will focus on the quality standards set by typographers for hand-typeset documents and ask to what extent they are achieved by TEX. Limitations of TEX's algorithms are analyzed; and missing features as well as new concepts are outlined.

1 Introduction

Last year at Stanford we celebrated the tenth birthday of the TEX project. Up to now, TEX has served thousands of users well and we expect it will continue to do so in the future. The longevity of TEX lies in

- the quality of its output
- its universal availability
- and its stability.

In the last few years, more and more users brought TEX from the universities into industry where it was challenged by new applications [33]. But time does not stand still, and what was at the top of its profession yesterday might prove to be obsolete tomorrow. TEX is still state of the art for the tasks it was designed to accomplish, but, with the growing understanding from several years' usage, we can now see where it will fail in high quality typesetting.

As a result of user pressure [27], Don Knuth announced a new version of TEX at Stanford, acknowledging the fact, that he did not foresee the need for 8-bit input [19]. At the same time, he made it clear, that he had decided to retire from this project and return to his long delayed topic "The Art of Computer Programming".

So TEX is finally frozen, and any further development will result in a different system no longer maintained by Knuth. The main purpose, therefore, of this paper is to give an overview of high quality typesetting requirements (covered and not covered by TEX 3.0) thereby, we hope, channeling future developments so that we do not end up with several incompatible "TEX-based systems", but rather with one system that will provide the same characteristics (i.e., quality, portability, and availability) as the current program.

TEX was designed as a low-level formatter, a stable kernel, of a typesetting system where extensions at both ends would be possible to take into account developments in printing technology (back end) and in user interfaces (front end) [14]. Thus, complaints about user unfriendliness of TEX are uncalled for, since such requirements can be handled by front ends either written in the TEX language itself like LATEX and, therefore, fully portable, or in an external language like ArborText's Publisher, or VAX Document, etc. These systems use TEX or a TEX-based system as the ultimate formatter but provide a user-friendly interface [32].

When we discuss missing features, we must distinguish carefully between things which can and should be handled by a front end system and things that are truly tasks for a formatter and cannot be handled in TEX 3.0. In the following sections we analyze features required for high quality typesetting, discussing whether they can be handled by TEX primitives or by a suitable front end, or both. If they cannot be handled, we attempt to find ways to achieve the desired results. Finally, in section 12, we switch our attention to the concepts of the TEX language itself, outlining some ideas on how a language for a new system could describe the underlying concepts more clearly.

2 Line breaking

TEX's line breaking algorithm is clearly a central part of the TEX system. Instead of breaking a paragraph line by line, the algorithm regards paragraphs as a unit and searches for an 'optimal solution' based on the current values of several parameters. Consequently, a comparison of results produced by TEX and other systems will normally favour TEX's methods.

Such an approach, however, has its drawbacks, especially in situations requiring more than block style text of a fixed width. The final line breaks are determined at a time when information about the content of the current line has been lost (at least for the eyes of TEX, i.e., its own macro language), so that TEX provides no sort of post-processing of the final lines based on their content.

Furthermore, there is no way to influence the paragraph shape with regard to the current position on the page, since this information is not known *a priori*. See section 4 for further discussion of this topic.

The use of only four categories (tight, decent loose, very loose) to distinguish different glue–settings in adjacent lines seems somewhat inadequate The number of categories should be increased In addition, a more global approach (even beyond paragraph borders in certain circumstances), taking the overall variation of glue-setting into account might produce better results.

2.1 Line breaking parameters

While the algorithm provides a variety of parameters to influence layout, some important ones for quality typesetting are missing. There is no way to deal with vertical stripes produced by interword gaps falling into the same vertical position. A similar problem involves identical words one above the other, especially at the beginning of a new line. Both problems are distracting to the eyes of the reader and will destroy any effort to produce a beautifully broken paragraph. A good example is shown at the beginning of the third paragraph of section 4 where "...breaking algorithm ..." is repeated on two lines.

Another aspect of fine print is the assurance that the last line of a paragraph will not be too short. This is especially important in layouts which use paragraph indentation, where an undesired gap would be produced if the last line of one paragraph is shorter than the indentation of the next paragraph. Unknown to most TEX users, this can be prevented by a special setting of TEX's line breaking parameters as shown in example 1 in section 14. While other parts of this paper use this setting, this paragraph shows the undesired effect.

Hyphenation of consecutive lines is handled for up to two lines (`\doublehyphendemerits`), but there is no possibility of avoiding paragraphs like the current one and the next one, in certain circumstances. As one can easily observe, the number of hyphens in these paragraphs is artificially forced by setting some of TEX's line breaking parameters to unusual values. But in non-English languages (with longer word lengths on the average), such situations present real-life problems.

Another problem is the discrepancy between the first and later lines of a paragraph, produced by the implementation of the paragraph indentation. This is especially crucial in layouts with zero indentation, because space at the beginning of the first line (for example, from `\mathsurround`) will not vanish into the margin because of the implicit `\hbox` representing the indentation (even if not visibly present), while such space will be removed at the beginning of later lines. This will result in strange starting gaps.

3 Spacing

When block text is to be produced, it is necessary to change the interword or the intercharacter spacing, or both. Since variable intercharacter spacing is frowned upon by the experts (except in rare circumstances), a line breaking algorithm has to stretch or shrink the interword space starting from an optimal value given by the font designer until the final word positions are determined. Again, TEX has a well designed algorithm to take such stretchability into account. Additionally, each character has a so called `\spacefactor` assigned to it which will influence a following space, so that it is possible to enlarge or reduce the interword space after certain characters. As an example, compare the spacing after punctuation characters in this paragraph with other paragraphs.

There is no provision, however, for influencing the interword gaps in relation to the current characters on both word boundaries. If it is necessary to shrink a given line, not all gaps should shrink by the same amount. Instead, it is best to shrink more af-

2 6 7 5 3 4 1

Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei
Das Aus kam in der letzten Runde, wobei

Figure 1: Interword spacing

The interword spaces are numbered in a way so that higher numbers denote spaces which should shrink less using the rules given by Siemoneit [28]. The last line shows the resulting overfull box which would be produced by standard TEX in this situation.

ter a comma, for example, than between 'then it' because of the different shape of the characters. There is no way to achieve such fine tuning in TEX except by manually adding `\hskip` in lines, which is intolerable. An example of this approach is shown in figure 1. Such a mechanism is clearly font dependent, and an implementation would, therefore, change both TEX and METAFONT, since the best place to store this information is in the `TFM` file. But even tables similar to `\sfcode` (fixed by the format or a macro) would be a big improvement, since most fonts in use tend to have similar shapes.

In TEX's concept for glue-setting an important distinction is made between stretchable and shrinkable glue: while the latter is only allowed to shrink to a fixed minimum (i.e., the natural width minus the shrink component), any given amount of stretchable glue is automatically allowed to stretch arbitrarily far.[1] The reason for this behavior is that it allows the line breaking algorithm to achieve 'emergency results' if no suitable line breaks are otherwise found. But this is undesirable in most circumstances, so that either the stretching should be bounded similarly to shrinking in all cases (resulting in some changes to the line and page breaking algorithms), or another class of glue should be added, for which the amount of stretching can be determined individually.

Don Knuth [18, pp. 394–395] gives an example of how to achieve hanging punctuation (together with special fonts, as he noted). Since this, too, is a sign of good quality typesetting, it is questionable whether such a scheme (that will make the ligature mechanism partly unusable, along with other side effects) is advisable or whether this should be a direct feature of a future program.[2]

4 Page breaking

A major problem with TEX is its page breaking algorithm. Page breaking is handled asynchronously by moving things at certain times from the list of recent contributions onto the "current page" until this list is filled with more items than will fit on the page in final form. The final page break is chosen by weighing badness (how full the page is if we break here) and penalties (how expensive it is to break here). Such penalties will be placed after some of the lines either by the line break algorithm or during macro expansion.

But good page layout usually requires taking pairs of facing pages into account as they will be seen by the reader. This is in itself not a real restriction, because one can view a double page as a huge case of two-column format, provided, of course, that both pages can be held simultaneously in memory. But, unfortunately, none of TEX's internal mechanisms can handle multi-column layout properly, so that such an approach has to avoid all internal features for page breaking like `\insert`, etc. Good examples that also show the limitations of TEX in this regard are the output routine of LATEX [22] and implementations of multi-column layout [4, 24], all bordering on the impossible.

But, even more important, the line breaking algorithm in conjunction with the page breaking algorithm pose unsolvable problems. When the final page break is chosen by TEX, all paragraphs which were once candidates for the current page are already divided up by the line breaking alogrithm, and this division cannot be undone for text carried over to the next page, since some of the necessary information (space at the line breaks, for example) is lost. This makes it impossible to change the page layout at a fixed place, e.g., at the top of a new page, to leave room for a small figure surrounded by text. Only in very restricted circumstances a solution can be found inside TEX [9], but documents of moderate complexity cannot be handled this way. A general solution to this problem can be included in the current TEX in an upward compatible manner. A prototype was designed at the University of Mainz shortly after the conference at Stanford [25].

It is an open question whether we should follow TEX's page breaking algorithm at all, since it was stopped short because of the space and time constraints of the computers available at the time of its development. In his PhD thesis [26], M. Plass considered several global optimization strategies, using a two pass system. His results open a wide field for future research work. Some of his ideas seem to be used in the Type & Set system [3].

1 Under normal circumstances, however, this is prevented by the badness function.

2 Starting with the next section the article uses hanging punctuation. The change in quality is clearly visible although improvement is still possible by making subtle ajustments to all characters (e.g., move the 'r' a tiny bit out, etc.) to reach a perfect alignment.

The main contribution of TEX82 to computer based typesetting was the step taken from a line-by-line paragraph breaking algorithm to a global optimizing algorithm.[3] The main goal for a future system should be to solve the similar, but more complex, problem of global page breaking.

5 Page Layout

For the tasks of page makeup, TEX provides the concept of output routines together with insertions and marks. The concepts of insertions and marks are tailored to the needs of a relatively simple page layout model involving only one column output, footnotes, and at the most simple figures once in a while.[4]

The mark mechanism provides some information about certain objects and their relative order on the current page, or more specifically, information about the first and last of these objects on the current page and about the last of these objects on any of the preceeding pages. Such information is necessary to construct certain kinds of running heads, e.g., one with the name of the current chapter or with information about the first and last word explained on the page, etc.

This is a global mechanism, however, so that only one class of objects can take advantage of the whole mechanism. If more than one class is implemented, some of the features of the mechanism are lost within one class.[5] As a consequence of this deficiency, one should extend the mark mechanism to a system of independent marks which can be allocated separately by a macro package.

The insertion mechanism seems to be derived from 'footnote applications', and later extended to allow for some simple kinds of floating insertions.[6] But placement of floating objects needs more than simply storing them in a huge box which is split at a certain point when the output routine is called. Floats are accompanied by captions and the like, which require differing treatment depending on their final placement on the page. Floats may vary in width even if they belong to the same class. On the other hand, deferred floats in one class may influence or even prohibit the placement of floats in other classes.

Some classes of insertions, like marginal notes, cannot be handled by the primitives at all. To provide such features LATEX, for example, defines its own memory management for floating objects. Naturally, such a mechanism is slow and space consuming. Additionally, the quality of page breaks is further reduced because it is difficult to maintain, for free, all the information provided by the insertion concept.

Another problem is the design decision that the page breaking mechanism is, at least in its crucial parts, available only in outer vertical mode; thus,

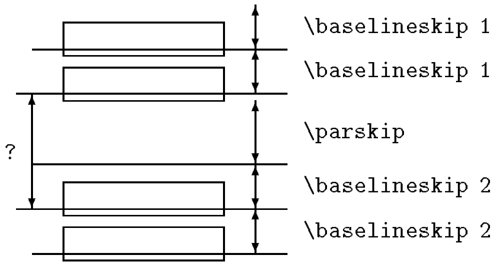

Figure 2: Baseline to baseline spacing

To implement a baseline to baseline dimension, for example between a paragraph and a heading (denoted by the question mark), the value for `\parskip` has to be determined in dependence of the `\baselineskip` of the second paragraph. Unfortunatly the value of `\baselineskip` used will be the one current at the end of the second paragraph while the `\parskip` has to be computed at its beginning.

for example, space for insertions is not taken into account when splitting a `\vbox`.

For a designer TEX's model of interline glue determination is very unfamilar because it does not allow to specify baseline to baseline spacing in a page-spec without using lengthly and complicated internal computations (see figure 2 on the next page). This also means that it is nearly impossible to implement grid-oriented specs, i.e., where (nearly) all baselines fall into predetermined positions. This article uses a grid-oriented spec (which was partly handprepared) to show this aspect of high quality typeseting. Details are given in example 3.

The design of suitable primitives for this complex must go hand in hand with a new algorithm for page breaking, comprising probably the most drastic changes to the TEX system.

3 It should be noted that a similar algorithm was developed independently by J. Achugbue [2]. A comparison might lead to further enhancements.

4 The term 'one column output' means that all text is assembled using the same line width. Problems with variable line width are discussed in section 4. Of course, this already covers a wide range of possible multi-column layouts, e.g., the footnote handling in this article. But a similar range of interesting layouts is not definable in TEX's box-glue-penalty model.

5 The LATEX implementation provides an extended mark mechanism with two kinds of independent marks with the result that one always behaves like a `\firstmark` and the other like a `\botmark`. The information contained in the primitive `\topmark` is lost.

6 This is only a guess from studying [17].

6 Penalties — measurement for decisions

Line and page breaks in TEX are determined chiefly by weighing the "badness" of the resulting output[7] and the penalty for breaking at the current point. Such penalties are either inserted directly by the user (during macro expansion) or added later by certain TEX formatting routines.

The main problem posed by the implicit penalties is that they cannot be removed. If, for example, the line breaking algorithm decides to put a penalty after a line (e.g., from `\widowpenalty`), there is no way to prohibit a page break at this place via macro expansion except, of course, by setting the penalty in question to infinity. This is the result of TEX's algorithm that consecutive penalties p_1 and p_2 behave like $p_3 := \min(p_1, p_2)$.

Local changes to the offending penalty parameters are error prone and time consuming, since after each change the following page breaks might fall in different places. Additionally, future editions of the document are made more difficult because every correction of this sort might produce undesired results after a single change.

If we think in terms of the current TEX (i.e., assuming all major algorithms are unchanged), it would be better to adopt a different strategy in the case of consecutive penalties, either $p_3 := \max(p_1, p_2)$ or $p_3 := {}^1\!/_2(p_1 + p_2)$. For both functions, the boundary cases $p_i = \pm\infty$ would need special care. Breaking the chain of penalties could be performed as usual by grouping or `\kern0pt` or similar procedures as is already done with ligatures, etc.

As we mentioned, this is a solution within the framework of TEX82. If a totally different algorithm for page breaking is designed, the concepts of local penalties should be reconsidered, too, and probably be replaced by a different strategy.

7 Hyphenation

When typesetting text, especially in narrow columns, hyphenation is often inevitable in order to avoid unreadable, spaced out lines. But readability has many faces; one of the golden rules says [28] "Avoid more than two hyphenated lines in a row." As we mentioned in section 2, this cannot be specified in TEX (unless one disables even two consecutive hyphens).

Another problem is hyphenation of words at places which are allowed but which distort the meaning:

Stiefel-tern	Stief-eltern
Spargel-der	Spar-gelder[8]

One should probably always forbid such problematical hyphens by choosing appropriate patterns for Liang's algorithm [23]. But readability is also distorted by the hyphenation of very short syllables which give no or almost no information about the word hyphenated and, therefore, slow down the reading process considerably. With TEX 3.0 it is now possible to adjust the minimal number of letters to the left and right of a hyphen. This is necessary for many languages which often have long words and, for example, many two letter syllables like German. But there is no provision for provision for assigning weights to hyphenation points, i.e., it is a simple yes or no situation. One possible solution to this problem would be to add another class of demerits which could be applied via

$$\frac{\text{users value}}{\text{length of broken part}}$$

or a similar function. Of course, one probably has to distinguish between pre-break and post-break text (and/or length) in the formula.

Since the quality of a certain breakpoint also depends on the word (i.e., the meaning of the word-parts), we should consider whether such information could be provided by the hyphenation algorithm itself.

8 Box Rotation

TEX's concept of document representation is strictly horizontal and left to right oriented. Beside the problem of processing documents containing right-left or top-down oriented languages (which can be handled to some extent by special versions of TEX [21]), this also poses unnecessary restrictions in standard applications. Except by using `\special` (for POSTSCRIPT devices) it is impossible to rotate certain parts of the document. While arbitary rotation is indeed next to impossible for most output devices, rotation by 90° can be handled in a simple manner by rotating the character cells. It should be easy to include some sort of `\rotate` primitive to TEX's language which would allow rotating hboxes and vboxes by multiples of 90 degrees.[9] This would allow inclusion for example, of landscape tables, etc., in a document, without the need to use real glue and scissors to add the page number or the running head.

9 Font Information

The ISO Draft Standard [1] contains hundreds of properties describing a font resource. While some of

7 This is in some sense a measure of the difference between the optimal and actual amount of white space on the line or page in question.

8 The meanings of the words are 'step-parents' and 'savings', but the first parts of the words in the left columns mean 'boot' and 'asparagus', respectively.

9 This would also require changes to the `dvi` language and thus changes in all driver programs.

them are accessible through TEX via `\fontdimen`, the majority are not. It seems advisable to add more of them to the set of TEX's parameters (for example, a recommended `\baselineskip`), in order to be able to make font family changes in documents more easily.

9.1 Virtual Fonts

Use of font families in TEX, which differ in character position, etc., from the defaults in the Computer Modern family, is difficult but can be achieved as proved in several projects [7, 35]. The proposed use of virtual fonts [20] may help to simplify matters in this regard. If it is possible to agree on standards for the position and the method of access for certain accented characters for the most common Latin alphabet languages, it should be possible to typeset multilingual documents (using the new features of TEX 3.0) without introducing unnecessary variants of standard fonts differing only in the availability of certain accented characters as 'real' letters.[10] A good survey of accented characters in Latin alphabet languages and a proposal for their access via ligatures is given by Haralambous [8].

9.2 Ligatures and Kerns

Unfortunately, ligatures as well as kerns differ from language to language. Take, for example, the 'ffl' ligature which is not used in traditional German documents. On the other hand, such documents contain 'ch', 'ck' and 'ft' ligatures to obtain a better script. The following examples shows the difference:

Druckschrift	Druckschrift	(standard)
Druckschrift	Druckschrift	(German ligatures)

Since these special ligatures do not involve new letter shapes (at least not in most font families) it is possible to achieve the desired results simply with kerning. In the Computer Modern font family [15], both 'ch' and 'ck' are contained in kerning programs, but only for serif fonts. Other font families show similar deficiencies. Thus, for typesetting German documents, one either needs special physical fonts (or at least virtual fonts), or a way to manipulate ligature and kerning programs from within the TEX program. For reasons of portability, a controlled access to the ligature/kerning programs during font loading seems preferable.

10 Tables

Well-designed tables are difficult to typeset even for experienced hand composers. TEX's primitives `\halign` and `\valign` do a marvellous job in this respect, even in complex situations. There is one important subclass of tables, however, which cannot be handled at all, except with hand tailoring. It is not possible to specify combinations of horizontally and vertically spanned columns, e.g., an open curly brace spanning several rows in one column while the row structure is maintained on both sides.

Another feature often desired is the ability to specify tables spanning several pages. While this is difficult to achieve, since TEX's table primitives normally read the whole table before determining the column width, etc., it does not pose unsolvable problems with the advanced features of TEX 3.0.

11 Math

Mathematical typesetting is one of TEX's major domains where no other automatic typesetting system has been able to catch up. But even in this area several things could be improved.

The source code of $\mathcal{AMS}$-TEX [30] shows many interesting examples where Spivak circumvents limitations of TEX's formatting rules by introducing complex code to define functions that should perform standard tasks in mathematical typesetting. A detailed analysis of these problems (double accents, under accents, placement of equation numbers, etc.) would easily fill several pages; some comments can be found in Spivak's documentation [29].

While TEX's spacing rules for math are quite good, it seems at least questionable that many of them are hardwired into the program instead of being accessible through parameters. The table for spacing between different math-atoms is probably the most important example of this sort.

Another problematical feature of TEX's math typesetting routines is that sub-formulas are always boxed at natural width even if the top level math-list is subject to stretching or shrinking. This might produce ugly results in certain circumstances. The concept of boxing sub-formulas has the additional disadvantage that such parts of a formula cannot be broken across lines. Therefore programming constructs like `\left`...`\right` which automatically determine the height of variably sized delimiters, cannot be used in complex displays.

12 TEX's language

The language of TEX is divided into two parts which are described as the mouth and the stomach of TEX [18]. This distinction is crucial in many applications since the output produced by TEX's gastronomical routines cannot be fed again into its mouth, i.e., its scanner. Actually, constructed boxes are post-processable to a limited extent (via `\lastbox`, `\unpenalty`, etc.) but arbitrary constructions cannot be handled this way because primitives for manipulating things like characters, rules, etc., are missing. In general, this distinction is the reason for

10 The use of the accent primitive of TEX is not recommended for standard accents of a language [18, p. 54] since it disables the hyphenation facility.

many obstacles in TEX-programming, which sometimes prevent any solution at all. A new system should remove this two-class society of internal commands.

Another severe problem of the language is its incompleteness regarding standard programming constructs (such as certain conditionals, an acceptable arithmetic parser, etc.), as well as special constructs suitable for typesetting. To determine, for example, the length of the last line in a paragraph (which is done automatically by TEX when typesetting displays), one has to use a complicated and lengthy computation, as shown in example 2, below. This example also shows one of the inconsistencies of TEX's language: `\prevgraf` has to be advanced using a scratch register because the direct use of `\advance` is forbidden. Many problems of this sort can be found by looking at appendix D of the *The TEXbook* [18, pp. 373–401] which is entitled "Dirty Tricks". Actually nine out of ten examples therein are used in the implementation of LATEX [22], which shows that these examples are far less exotic than the preface to this appendix suggests. As examples of missing programming constructs, conditionals, such as `\ifmathopen`, for determining math atoms should be mentioned.

Such problems explain the fact that general application software written in TEX (like LATEX) easily takes up more than a third of the available memory without typesetting even a single letter. For better and more stable front ends one needs a language where such tasks can be specified in a more elegant manner.

Some of TEX's restrictions in the language are due to the representation of dimensions in most TEX installations as 'real numbers', which are machine-dependent.[11] To make TEX nevertheless machine-independent, Knuth tried to prevent machine-dependent results generated in TEX's stomach from creeping into parts accessible to the scanner, or to influence internally any decisions about line or page breaks. As a result of this strategy, the use of the code in example 2 together with a finite `\parfillskip` (as in example 1) will nearly always produce the value `\maxdimen` instead of a decent one.[12] For the same reason, Knuth said in a conversation with the author at Stanford that he cannot permit the removal of arbitrary items in a constructed list since this would allow access to floating-point arithmetic. But there exists another way to achieve machine-independence, which would also eliminate the restrictions mentioned, namely to change from floating-point to fixed-point arithmetic[13] which can be done in a straightforward way, as Knuth himself has acknowledged [16, p. 46].

TEX is a macro-language with all the advantages and disadvantages. Anyone who ever wrote a relatively long application in TEX knows that debugging is extremely difficult. Transparent programming, as proposed by the author of TEX [10], is next to impossible: it is no problem to write three lines of TEX code that cannot be understood even by TEXperts without a second and third look. But it is much more difficult to write TEX code that performs a desired function, and is, at the same time, understandable to the average user. The examples given in section 14 are good test cases; they are all straightforward TEX-coding, but their precise meanings are difficult to understand without explanatory text.

Many problems arise from design decisions based on totally different semantic constructs, which have similar or identical syntactical structure. The most important examples are the curly braces and the dollar sign.[14] The curly braces are used both for delimiting arguments during macro expansion, as well as for the start and end of block structures that define the scope of certain declarations. In math mode they have the additional meaning of delimiting the scope of a sub-formula. Two consecutive dollar signs normally start or end a display formula, but in restricted horizontal mode they simply denote an empty math formula. Such concepts should be unraveled for the sake of clarity.

TEX's language is suitable for simple programming jobs. It is like the step taken from machine code (of the formatter) to assembly language. For complex programing tasks of general application software, especially from the viewpoint of logically tagged documents [5], a more powerful language with well-defined concepts for variable-bindings, procedures, etc, is preferable. While this aspect can be achieved in a front end programming language (which compiles into the TEX language) it is better to include it, for the sake of portability, in the TEX kernel. In the author's opinion, an ideal language should combine the advantages of procedural languages with the goodies of interpreter features.[15] As a side effect, such a language would be partly compilable.

13 Conclusion

The current TEX system is not powerful enough to meet all the challenges of high quality (hand) ty-

11 Actually, this only applies to internal dimensions representing stretching or shrinking of glue, computed kerns for accents, and some others.

12 The reason is given in module 1148 of the TEX program [16, p. 470].

13 Perhaps, using always the same floating-point algorithm (either available in the compiler library or simulated by the program) would be even better.

14 To be more exact, the three characters with `\catcode` one, two, and three.

15 Some sort of LISP-like system, but with primitives suitable for typesetting.

pesetting. The author shares Knuth's dream of a stable, low-level formatter which is able to produce documents of highest quality. But, unlike Knuth, he views the current TEX only as a very good prototype on the way to reach this goal.

As outlined in this paper, many important concepts of high quality typesetting are not supported by TEX 3.0. Further research is necessary to design a typesetting language which can handle these tasks properly.

The TEX user community needs an open mind for new developments that keep the 'TEX-System' the state of art in the field of computer typesetting. As Knuth is no longer involved in research on typography it is important for TUG to find an identity in *supporting* and *maintaining* 'the best typesetting program' and not only promoting the program that Knuth has given to the world. If we don't strike for even further quality our large community might fall back to insignificance.

One important step for TUG would be to initiate and (when advisable) support further research projects which will take up the challenges posed by the Stanford project.

14 Examples

Example 1 To avoid nearly empty lines at the end of a paragaph, the following code could be used:

```
\parfillskip \columnwidth
\advance \parfillskip -1.5\parindent
\advance \parfillskip 0pt minus \parfillskip
\advance \parfillskip 0pt minus  -1em
```

This setting was used throughout this article, for example, which led to some changes in section 2. With the standard setting (e.g., `0pt plus 1fil`), the first and third paragraph therein would have ended with the word part 'ods' (from 'meth-ods') and the word 'topic' respectively. Unfortunately, this solution is not perfect either, since it produces somewhat funny results with lines consisting of two very short words.

Example 2 The following code determines the length of the last line of the preceeding paragraph, using a feature of TEX built into mathematical displays. This code can be used to determine, for example, the amount of white space before an itemized list, or something similar. The example of code given is not really suitable for direct applications of this sort, since it simply displays the value found on the terminal. But it could easily be extended.

```
\def\getlastlinewidth{\ifhmode $$%
 \predisplaypenalty\@M \postdisplaypenalty\@M
 \abovedisplayskip-\baselineskip \belowdisplayskip\z@
 \abovedisplayshortskip\abovedisplayskip
 \belowdisplayshortskip\belowdisplayskip
 \showthe\predisplaysize
 $$\count@\prevgraf \advance\count@-\thr@@
 \prevgraf\count@ \else\typeout{*Not hmode*}\fi}
```

This example illustrates several important things. First, there is no elementary way to compute such important information. Second, it is one of the (not unusual) cases where information about the typesetting process can only be got by introducing undesired (since space-consuming) penalties, glues, and null-boxes in the output.

Example 3 To introduce a grid-oriented spec all flexible glue on the page has to be disposed off (execpt for `\skip\footins`) and the `\vsize` must be adjusted. Titles are set with `8pt` + `4pt` = `\baselineskip` leading and we have to ensure that the above space is kept after a page break. Lists are set with `6pt` + `6pt` so the inner lines are half way off. Page breaks insides lists would need special treatments, e.g., by increasing `\topskip` to keep the subgrid. Figures and examples in different type sizes are measured and necessary kerns added to keep the surrounding material in line. Again this approach only works if no page break intervence, which happens to be the case for this article. To use the badness calculation of TEX for determing page breaks a stretchable `\topskip` can be used. During the output routine this extra stretch must then be canceled again.

References

1. *Information Processing — Font Information Interchange, ISO/IEC JTC 1/SC 18/WG8 N1036*, February 1990.
2. Achugbue, James O. "On the line breaking problem in text formatting." *Proc. of the ACM SIGPLAN/SIGOA*, 2(1, 2), 1981.
3. Asher, Graham. "Type & Set: TEX as the engine of a Friendly Publishing System." Pages 91–100 in *TEX applications, uses, methods*, Malcolm Clark [6].
4. Benson, Gary, Debi Erpenbeck, and Jannet Holmes. "Inserts in a multiple-column format." Pages 727–742 in *1989 Conference Proceedings*, Christina Thiele [31].
5. Bryan, Martin. *SGML: an author's guide to the standard generalized markup language*. Addison-Wesley, Woking, England; Reading Massachusetts, second edition, 1988.
6. Clark, Malcolm, editor. *TEX applications, uses, methods*, Chichester,West Sussex, England, 1990. Ellis Horwood Limited. Exeter con-

ference July 1988.
7. Conrad, Arvin C. "Fine typesetting with TEX using native autologic fonts." Pages 521–528 in *1989 Conference Proceedings*, Christina Thiele [31].
8. Haralambous, Yannis. "TEX and latin alphabet languages." *TUGboat*, 10(3):342–345, November 1989.
9. Honig, Alan. "Line-Oriented Layout with TEX." Pages 159–184 in *TEX applications, uses, methods*, Malcolm Clark [6].
10. Knuth, Donald E. "Literate programming." *The Computer Journal*, 27:97–111, 1984. An expository introduction to `WEB` and its underlying philosophy.
11. Knuth, Donald E. *Computers & Typesetting*. Addison-Wesley, Reading, Massachusetts, 1986. Consists of [18, 16, 13, 12, 15].
12. Knuth, Donald E. METAFONT*: The Program*. Volume D of *Computers & Typesetting* [11], 1986.
13. Knuth, Donald E. *The* METAFONT*book*. Volume C of *Computers & Typesetting* [11], 1986.
14. Knuth, Donald E., September 1987. Talk given at Gutenberg Museum Mainz.
15. Knuth, Donald E. *Computer Modern Typefaces*. Volume E of *Computers & Typesetting* [11], July 1987. Reprint with corrections.
16. Knuth, Donald E. *TEX: The Program*. Volume B of *Computers & Typesetting* [11], May 1988. Reprint with corrections.
17. Knuth, Donald E. "The errors of TEX." Technical Report STAN-CS-88-1223, Stanford University, Department of Computer Science, Stanford, California 94305, September 1988.
18. Knuth, Donald E. *The TEXbook*. Volume A of *Computers & Typesetting* [11], May 1989. Eighth printing.
19. Knuth, Donald E. "The new versions of TEX and METAFONT." *TUGboat*, 10(3):325–328, November 1989.
20. Knuth, Donald E. "Virtual Fonts: More fun for Grand Wizards." *TUGboat*, 11(1):13–23, April 1990.
21. Knuth, Donald E. and Pierre MacKay. "Mixing right-to-left text with left-to-right text." *TUGboat*, 8(1):14–25, April 1987.
22. Lamport, Leslie. `latex.tex`, February 1990. LATEX source version 2.09.
23. Liang, Franklin Mark. *Word Hy-phen-a-tion by Com-put-er*. PhD thesis, Stanford University, Department of Computer Science, Stanford, CA 94305, August 1983. Report No. STAN-CS-83-977.
24. Mittelbach, Frank. "An environment for multicolumn output." *TUGboat*, 10(3):407–415, November 1989.
25. Mittelbach, Frank. Letter to Don Knuth, September 1989. Suggestions for the TEX 3.0 release. Published by R. Wonneberger in [34].
26. Plass, Michael Frederick. *Optimal Pagination Techniques for Automatic Typesetting Systems*. PhD thesis, Stanford University, Department of Computer Science, Stanford, CA 94305, June 1981. Report No. STAN-CS-81-970.
27. Rynning, Jan Michael. "Proposal to the TUG meeting at Stanford." *TEXline*, 10:10–13, May 1990. Reprint of the paper that triggered TEX 3.0.
28. Siemoneit, Manfred. *Typographisches Gestalten*. Polygraph Verlag, Frankfurt am Main, second edition, 1989.
29. Spivak, Michael. *`amstex.doc`*, 1990. Comments to [30].
30. Spivak, Michael. `amstex.tex`, 1990. AMS-TEX source version 2.0 (without comments).
31. Thiele, Christina, editor. *1989 Conference Proceedings*, volume 10#4 of TUGboat. TEX Users Group, December 1989.
32. Wittbecker, Alan E. "TEX enslaved." Pages 603–606 in *1989 Conference Proceedings*, Christina Thiele [31].
33. Wonneberger, Reinhard. "TEX in an industrial environment." In Brüggemann-Klein, Anne, editor, *1989 EuroTEX Conference Proceedings*, 1990. To appear.
34. Wonneberger, Reinhard. "TEX yesterday, today, and tomorrow." *TEXhax*, 90(5), January 7 1990.
35. Youngen, R. E., W. B. Woolf, and D. C. Latterner. "Migration from computer modern fonts to times fonts." Pages 513–519 in *1989 Conference Proceedings*, Christina Thiele [31].