Lapses in TEX — a look backward

Barbara Beeton

TEX has now been used "in the wild" for over 40 years, so it's possible to look back and examine this tool and its ecosystem to see what decisions might have been made differently to avoid problems that still exist today. Some deficiencies are the result of limited hardware or facilities (such as Unicode) that did not exist at the time when TEX was created (1978–79). However, others could reasonably have been implemented differently within the existing limitations, and these are what will be examined here.

By what authority?

Why can I claim authority to examine this topic?

I was sent to Stanford in the summer of 1979 to learn TEX under the tutelage of the TEX Project. In preparation for this assignment, I collected a number of "good bad examples", problems that had actually occurred in publication production at the American Mathematical Society (AMS). In the event, these examples proved to be well chosen; a number of them appear in Appendix D of *The TEXbook* [5], and the command `\firstmark` was newly created to address an unmet need, evidenced by the insertion of its syntax handwritten by Don Knuth in my copy of the ur-TEX manual.

For many years, I was Don's bug collector (or "TEX entomologist" as I preferred), distributing reports to individuals whom Don recognized as sufficiently knowledgeable to determine whether a problem was or was not a bug, communicating their analysis to the submitter, organizing the reports for communicating to Don on his predetermined schedule, and in turn communicating Don's response to the submitter of the first report. The bug collecting function was turned over to Karl Berry with the completion of the 2014 review cycle.

Initial conditions and philosophy

The rationale behind the creation of TEX has been thoroughly covered in Knuth's 1978 Gibbs Lecture [4]. The first implementation in SAIL was eagerly adopted by some local academics and visitors to Stanford who had access to the requisite hardware. But it soon became apparent that a portable implementation would serve a much larger audience who needed this tool, so a widely available subset of Pascal was chosen for TEX82, and a new tool, WEB, was developed that would make it possible to code the program and publish it in an intelligible "public" form that could be read and understood by a technically literate audience (*TEX: The Program* [6]).

doi.org/10.47397/tb/42-1/tb130beeton-lapses

We must recognize that the original target for TEX's output was *print*. TEX predates the World Wide Web, Unicode, and the high-resolution screens attached to very fast processors that have made it possible for someone to read a document directly from the electronic representation. In order to meet these "new" requirements most effectively, the document structure needs to be preserved in the final electronic form. Several elements important to this goal will be evident in what follows.

8-bit limitation

It's a bit unfair to fault this limitation, since Unicode didn't exist until 1987, long after Knuth had returned to his work on *The Art of Computer Programming*. TEX78 was based on 7-bit fonts; the basic ASCII arrangement was carried into TEX82, still with seven bits "live", although eight bits were built into the code structure, and full eight-bit input support was added in 1989.

A 256-character font encoding was devised in 1990 at a TEX meeting in Cork [2]. This arrangement didn't match any of the standard 8-bit European encodings; however, an attempt was made to accommodate all the accented letters, variants and digraphs required for Western European languages. Input encodings were then developed to permit direct input of these alphabetic characters from keyboards that provided them.

It was possible within this limitation to implement Cyrillic for most Slavic languages, with input based on the Mathematical Reviews transliteration, a rather complex ligaturing mechanism, and requiring only a few control sequences [1, p. 17]. Other alphabetic fonts created for this "unextended" version (as reported in *TUGboat*) were Vietnamese, Hebrew, modern Greek, Arabic, Croatian glagolitic, Ethiopic, and the International Phonetic Alphabet (IPA).

With the introduction of XETEX in 2005 and LuaTEX at about the same time, input was opened up to accept Unicode natively, and support was provided for OpenType and TrueType fonts. For the basic TEX engine (with LATEX preloaded) `\inputenc{utf8}` was implemented, but input of accented letters is converted to the `\cs`⟨*letter*⟩ form for processing, and TEX fonts are still limited to 256 characters.

For languages where the input order of characters does not necessarily match the display order, 8-bit input is insufficient. Supporting this would have required extensive (breaking) changes to the program, exceeding Don's requirements. An "early" extension to TEX, Omega [3], was first presented in 1994, but has since been abandoned. Nonlinear composition requirements are now implemented by font-shaping mechanisms, not by the main composition engine (cf. [8]).

Limitation of the character box

The "shape" of a TEX character is defined as a rectangle (or a parallelogram for sloped characters) with the origin at the baseline on the left side (for left-to-right scripts) and a width measured at the baseline. TEX uses only the metrics. While this simplicity permits efficient calculation of necessary values for line and paragraph breaking, without adjustments the spacing of adjacent characters may not be optimal, regardless of the quality of their design.

This model is most appropriate for Western alphabetic languages, which typically have reliably "restricted" shapes, even taking diacritics into consideration. Font-shaping mechanisms developed to handle more complex scripts have been mentioned in the previous section.

Within the existing design, two adjustments are provided, recorded in the `.tfm` files used by TEX82: explicit kerning and the "italic correction", a value indicating the overhang of a tall sloped letter. There is no corresponding adjustment for the left-hand side, which results in the following suboptimal appearance, depending on the letter beginning a new line:

Watch the left margin.
This is one example.
The normal flush left margin.
What about this?
One last line.

Hermann Zapf's microtypography addressed this.

In the Computer Modern fonts no kerning is specified between any lowercase letter and a following uppercase. Since CamelCase was not in heavy use when TEX was created, we ignore this omission here. But kerning between an uppercase letter and a following lowercase is also nearly nonexistent, leading to the unfortunate spacing of the combination "Av", among others, which could have been avoided.

Let's look at the situation where punctuation follows uppercase. This too is unkerned, and can yield particularly unsightly results in bibliographies, which are often overrun with initials (and not easily managed). Some examples, showing manual adjustments that have been used in this issue:

P.O. Box `P.O. Box`
P. O. Box `P.\thinspace O.~Box`
W.J. Martin `W.J. Martin`
W.J. Martin
`W\kern-.05em.\kern.07em J\kern.01em. Martin`

In math, the situation is somewhat different. While in text the spacing of adjacent letters is set based only on their origins and width, in math the

italic correction is always applied, increasing inter-character spacing. This is particularly noticeable in sub- and superscripts:

$$A_xB \quad A_fB \quad P_xQ \quad P_fQ$$

This can be adjusted manually by applying a negative or positive thin space, and Knuth in *The TEXbook* recommends manual attention. But this can get tiring, and some instances can be missed in proofreading, leading to inconsistent appearance.

A recent post on `tex.stackexchange.com`[1] contained a repetitious example of bad spacing that was easily addressed by an ad hoc definition.

$$\left(\frac{\partial f^0}{\partial \eta_t}\right) \qquad \left(\frac{\partial f^0}{\partial \eta_t}\right)$$

This definition is applied in the obvious location.

```
\def\partialf{\partial\mkern-2mu f}
$$ \biggl(
  { \partial f^0 \over \partial \eta_t }
  \biggr)
$$
```

Another `tex.stackexchange` post[2] asks for "optically balanced space" in expressions such as the following.

$$\mathbf{e}_i \cdot \mathbf{e}_j \neq \mathbf{e}_i \times \mathbf{e}_j \neq \mathbf{e}_i \wedge \mathbf{e}_j \neq \mathbf{e}_i \otimes \mathbf{e}_j$$

Compare the more uniform spacing of this expression, where subscripts don't disturb the "line".

$$\mathbf{a} \cdot \mathbf{b} \neq \mathbf{a} \times \mathbf{b} \neq \mathbf{a} \wedge \mathbf{b} \neq \mathbf{a} \otimes \mathbf{b}$$

The requested spacing is not possible without knowing more details about the shapes of all characters that can appear in math expressions. (Whether or not such a request is reasonable or desirable has been asked in a comment to the request. This question will be ignored here.)

Limitations imposed by the line-breaking algorithm

Baselines aren't "frozen" until the end of a paragraph, and it's bad style to break a page between text and a display, so in effect, the display is part of the preceding paragraph. If a display is set in a font size different from (usually smaller than) that of the preceding paragraph, the wrong baselines are applied to that text.

It's possible to adjust this manually, but many (most?) people are unaware of it, and scrunched paragraphs can be seen in otherwise fine math papers. Look at this paragraph; the display that follows is set in `\footnotesize`.

$$a + b = c$$
$$d + e = f$$

Even if the text following the different-sized display doesn't start a new paragraph, the normal baselines are restored. This code produced the example:

```
\begingroup \footnotesize
$$
\eqalign{
  a + b &= c \cr d + e &= f \cr }
$$
\endgroup
```

Line breaking by paragraph places some limitations on desirable formatting.

- If a paragraph is broken at the end of a page, and a different page width is wanted on the next page, that change can't be applied automatically.
- Sometimes, a by-line view is preferable to a by-paragraph view, for example, to facilitate communication of editorial suggestions. A change in line width affects line numbers. (Line numbers are required, for example, for some legal documents.)
- It's not easy to reflow material for, e.g., screen presentations.

Two-dimensional material vs. "the grid"

In addition to the baseline anomaly shown above, the design of TEX makes it difficult to maintain uniform baselines throughout a document. Historically, printers have tried to achieve layouts in which lines of type match up on the front and back of a page; this was particularly important on thin paper, where what's on the other side might "read through" if it is set between the lines on the reading side. This uniform spacing is known as "grid typesetting".

Uniform baselines aren't difficult to achieve with straight text, but several forms of printed material are by definition two dimensional — most notably math, chemical structures and music. Forcing them onto a grid can result in either squeezing or inserting excess space, degrading comprehensibility.

A few TEX practitioners have devised means to achieve grid layout, but in the presence of especially complex math structures, the problem may be intractable.

A fraction anomaly

In math processing, the gap between a fraction line and the numerator or denominator is defined to be the same height as the thickness of the fraction line; this is governed by the setting of font dimension 8 in the math extension (family 3) `.tfm` file (rule 10 in Appendix G of *The TEXbook*. with its application to fractions explained in rule 15). If for some reason

[1] `https://tex.stackexchange.com/q/592191`
[2] `https://tex.stackexchange.com/q/581045`

the fraction line is made thicker, the gap quickly becomes too large.

The (primitive) command `\abovewithdelims` demonstrates the problem. An alternative is implied by the description of the command `\above`, which accepts just a ⟨*dimen*⟩. The example in *The TEXbook* shows `1pt` as the dimension, and this looks promising, but as it turns out, this is treated in the same way as `\abovewithdelims`, so if a larger dimension is specified, the gap expands accordingly.

$$\frac{af}{fa} \qquad \left.\begin{matrix} af \\ \blacksquare \\ fa \end{matrix}\right. \qquad \frac{af}{fa} \qquad \left.\begin{matrix} af \\ \blacksquare \\ fa \end{matrix}\right.$$

Ideally, the gap should either increase more slowly, or require an explicit setting. It's likely that the need to accommodate such a situation was never predicted; it's quite rare. But when it does occur, the result is a distinct surprise, and a search for documentation doesn't find any.

This code produces the example shown above:

```
$$
 {af \over fa} \qquad
 {af \abovewithdelims.. 3pt fa} \qquad
 {af \above1pt fa} \qquad {af \above4pt fa}
$$
```

Hyphenation discrimination

All permissible hyphenation points are weighted the same, but in English (as in many languages), it is often better to preferentially hyphenate a compound word at the junction of its lexical elements. This is especially important in chemical and similar terms.

For example, let's adopt a convention that a hyphen shows the position of a normal hyphen, while an equals sign shows the location of a lexical (preferable) breakpoint, separating elements of a compound.

```
pho-to=syn-the-sis
pa-ra=di-chlo-ro=ben-zene
```

(Aside: All proper breakpoints in "photosynthesis" are identified by TEX's (U.S.) hyphenation algorithm, but only the last in "paradichloroben-zene".)

The dictionary used for developing the U.S. patterns has hyphenation indicated at only one level, so this limitation is not surprising. But the dictionary underlying the British patterns records two levels, based primarily on etymology, but also on syllabification. (The U.S. patterns, based on pronunciation, sometimes coincide with etymology, but it's not guaranteed. Technical terms are more likely to be hyphenated according to etymology.)

A two-level mechanism is even more desirable for agglutinative languages like German, and alternative mechanisms have been devised where necessary, but this would have been simpler had a two-level mechanism been included in the design.

Finally, if a text is to be properly reconstructed from the printed output, it must be possible to distinguish between hyphens that are inherent in the text and those introduced by the hyphenation routine. This information is lost after composition, and failure to restore it properly may change the meaning. In addition, inclusion of language markers in the output would be a useful adjunct here.

Missing spaces

In the original design of TEX output, only characters and their (relative) positions are present. The space character is absent; what is seen on a page is the illusion of a space, provided by the gap separating not-quite-adjacent glyphs. This is not remedied by PDF, and spaces can be lost when text is cut-and-pasted, unless the gap exceeds a certain minimum width.

But a different approach might have been taken, namely the marking of word boundaries. Nelson Beebe states that this "could have been trivially included in the original SAIL version with no significant memory increase." The presence of such markers could support checking for delimiter balancing, spelling, grammar and syntax, all of which are best done on the typeset form, not the input.

One way! Do not back up!

TEX input is processed in a one-way stream, with no provision made for backtracking. This means that it may be impossible to trap and save the last character or token in a string without parsing the whole string or otherwise predefining some particular feature that can be used to isolate it. (It *is* possible to apply special processing to the last line of a paragraph, calling on `\lastskip`, `\lastbox`, etc.)

One situation requiring special treatment of a single terminal character is the different shape of a terminal sigma in Greek (ς vs. σ). In one approach, the letter 'c' is input following a final sigma, and the two letters are ligatured to produce the desired shape. A different mechanism, ⟨*boundarychar*⟩, was added with the 8-bit update, but its application is not entirely trivial, and most users understandably prefer a more explicit solution.

Where is the origin?

TEX itself completely avoids discussing page dimensions. The imaging software assumes `\hoffset=0pt`, `\voffset=0pt`, extending downward. This is opposite from what is assumed for traditional printing, where the origin is at the bottom left and extends

upward. The printing devices available when TEX was developed were (and most devices today still are) unable to print at the absolute edge of the output medium; this was often blocked by the gripping mechanism, and although it was only a small fraction of an inch, it might not have been the same for all devices, so the origin couldn't be set at zero. Instead, an easy to remember value for a position inside of this limit was assigned: `1in,1in` from the upper left corner, a setting that is a source of consternation for the myriad TEX users located in regions using the metric system.

What is the origin of this one-inch origin?

In the U.S., paper was usually assumed to be lettersize, 8.5 by 11 inches, and a common size for the text block was `\hsize=6.5in` by `\vsize=8.9in`, which results in one-inch side and top margins and a 1.1 inch bottom margin (into which a page number is often dropped) when a full page is centered. This value was selected for the fixed origin, in order to provide a consistent value for the software. David Fuchs, developer of the earliest output device driver, was (to the best of my memory) the person who established the value.

The inherent ability of laser printers to print on different paper sizes is limited by driver support, and nonstandard dimensions are blocked or lost. `pdftex` introduced primitives for page dimensions and several different page boxes required by PDF, but even when laser printers became generally available, the 1-inch value was retained for the sake of backward compatibility.

One last thing to think about

When TEX was created, the only way to read a TEX document was on paper or from the source file. But since then, considerable software has emerged for PDF text analysis, optical recognition from scanned text, etc. In order to reliably locate and recover raw text from such "final" electronic documents, it's necessary to be able to disambiguate different columns on a page, recognize page numbers, and record similar identifying information. TEX contains nothing to make such recognition easy, or even in some cases possible. It was undoubtedly premature to think of such details in 1980, but they should be considered for the future.

Acknowledgments

Several people who were there at the beginning, or at least have been involved with TEX for a very long time, have contributed their knowledge to this effort, both by providing missing information and by otherwise keeping me honest.

Chief among these contributors is David Fuchs. Nelson Beebe added points that were overlooked in the limited environment of 1980, but which could have been implemented within those limitations had they been foreseen. Karl Berry, as ever, politely called attention to my logical inconsistencies. Phil Taylor commented from a British point of view, helping to tidy up bits that might not be understood the same way on opposite sides of the pond.

Finally, Don Knuth was asked to read and comment. (If Don hadn't created TEX, there would have been no reason for this essay.) His comment was: "All these things and more will be fixed in TEX* as soon as the implementation team is ready." [7]

References

[1] B. Beeton. Mathematical symbols and Cyrillic fonts ready for distribution (revised). *TUGboat* 6(3):124–126, 1985. `tug.org/TUGboat/tb06-3/tb13beetcyr.pdf`

[2] Extended TEX font encoding scheme — Latin. *TUGboat* 11(4):516, 1990. `tug.org/TUGboat/tb11-4/tb30ferguson.pdf`

[3] Y. Haralambous, J. Plaice. First applications of Ω: Greek, Arabic, Khmer, Poetica, ISO 10646/UNICODE, etc. *TUGboat* 15(3):344–353, 1994. `tug.org/TUGboat/tb15-3/tb44haralambous-omega.pdf`

[4] D.E. Knuth. Mathematical typography. *Bull. Amer. Math. Soc* 1(2):337–372, March 1979. `https://www.ams.org/journals/bull/1979-01-02/S0273-0979-1979-14598-1/S0273-0979-1979-14598-1.pdf`

[5] D.E. Knuth. *The TEXbook*. Addison-Wesley, Reading, Massachusetts, 1984. Volume A of *Computers & Typesetting*.

[6] D.E. Knuth. *TEX: The Program*. Addison-Wesley, Reading, Massachusetts, 1986. Volume B of *Computers & Typesetting*.

[7] D.E. Knuth. An earthshaking announcement. *TUGboat* 31(2):121–124, 2010. `tug.org/TUGboat/tb31-2/tb98knut.pdf`

[8] S. Matteson. The road to Noto. *TUGboat* 41(2):145–154, 2020. `tug.org/TUGboat/tb41-2/tb128matteson-noto.pdf`

◇ Barbara Beeton
`https://tug.org/TUGboat`
`tugboat (at) tug dot org`

* [A bell rings at this point.]