TEX: from a typographic system to the typography of the system

Thierry Laronde
tlaronde @kergis.com
2025-10-31

1. **Stating the problem**. Chesterton once said that « *a conservative is someone refusing to correct errors, while a so-called progressive individual insists on adding new ones* », so that he was, himself, neither one nor the other.

Concerning typography by computer, new partial solutions continue to pop up while various solutions have been created previously, some gone to oblivion, some still extent. Amongst the latter, Donald E. Knuth's TEX system is widely used, its layout result, specially concerning mathematics, is recognized as excellent, but its apparently arcane compilation process as well as the insane size of its most known distribution—6 GiB for the full TeXlive—have achieved F.U.D. so that no-one imagine it could be backing the documentation needs of an operating system, being incorporated in its base.

But it remains that there should not be any incompatibility between the own OS needs for its documentation and the users' ones. Instead of accumulating various solutions, each with its shortcomings, it's time to address the few remaining issues and to put the pieces together.

Because there is one feature that puts D. E. Knuth's solution apart: it is *a complete solution*: it allows to define fonts, without which there can be no layout; it computes the layout, giving an interpreter that can be programmed i.e. customized; and a program derived from `METAFONT` can do figures—John Hobby's `MetaPost`.

But if the D.E.K.'s typographic system provides fonts, this is because `METAFONT` is a rasterizer... a RIP. The solution is hence complete.

2. **Small and portable**. TEX and `METAFONT` have been written using the WEB system of literate programming. WEB is, now, an unfortunate name because of the meaning it has acquired with WWW. But WEB (as well as CWEB, the version for C programming) is just a way to write a program in a given language (Pascal for WEB; C for CWEB), splitting it in small chunks that can be interpolated since they are identified by a short description serving as key, and describing a chunk or a related group of chunks when it is easier to describe their purpose or their implementation, the description, meant for humans, using the full power of TEX. An utility: `tangle(1)` for WEB (resp. `ctangle(1)` for CWEB) simply extract all the programs chunks, interpolating chunks called in other chunks and assembling them in the correct order for the compiler. This program is Pascal—or C, depending on the version used.

In fact, TEX and `METAFONT` have been written less in Pascal than in some sort of Algol: what remains is only the flow control, *à la* C. Few routines, identified, have to be provided by the external implementation.

Once the pseudo-pascal (since it is not anymore any Pascal, but just a very common Pascal) code is obtained, since the Pascal compilers are not ubiquitous and Pascal was not really standardized, this code is translated in raw C, by an utility called in kerTEX `pp2rc(1)`—Pseudo-Pascal to Raw C, that is obviously only used on the matrix, not on the target, since it is used only to obtain the C source for the compilation.

The resulting programs depend only on a libc, since what kerTEX has added is also programmed in standard C, and, until very recently, the programs were even not relying on something outside the standard libc—some new primitives require, now, functions outside standard C. But it remains C, the system dependencies are identified and this compiles on everything.

> TEX—or PRoTE—do **not** depend on a POSIX library—POSIX.1. If, under Plan9, the compilation is done apparently under APE, it is simply because the building framework uses `sh(1)`. But this is the building framework, not the programs that depend on a POSIX compliant lib: the programs are natively C89. The Bourne shell is also used by the packaging system for now.
>
> I may rewrite some day both the building framework—RISK—and the kerTEX recipes, to use `rc(1)`—it will be compiled by the framework on systems not providing it. Using the Bourne shell seemed first as a good idea since it was available on Unices, available on Plan9 (but `ksh(1)` is in fact not POSIX compliant...) and available on Windows via `msys`. But POSIX si so huge and the description of the Bourne shell is now so involved that targetting a really compliant Bourne shell is a daunting task—the comment on the Bourne shell in the `rc(1)` paper is true when one dives in the POSIX specification...

3. **One engine to rule them all.** LaTEX has some supplementary requirements, exceeding both the TEX primitives, and the ε-TEX ones.

An extension has been written: PRoTE, that offers the TEX compatibility mode; the ε-TEX compatibility mode; and further extensions. In fact, only one engine does the work: PRoTE, for whatever: `tex` (D.E.K.'s standard set of macros); `etex` ($\mathcal{NTS}$' standard set of macros); or `latex`.

TEX+ ε-TEX+ extensions = PROTE

4. PROTE **as engine for roff too.** `Roff(1)` is not a programmable engine for general purpose (text and mathematics) layout. It is a limited engine, with few primitives, doing general (text) layout. All the macros are in fact implemented as specialized pre-processing text filters.

Absolutely nothing could prevent using the PROTE engine as the `roff(1)` engine for the layout, adding only support in DVI for two primitives that are available for `roff(1)` but that are not supported in legacy DVI. For preprocessing (the `man` or `mdoc` macros), a switch can be added to PROTE to recognize the beginning of the line as a prevision character (if what follows is a dot, it is a macro), so that PROTE will be used for preprocessing too.

There is no necessity to change the handy `mdoc` set of macros: TEX (in fact PROTE) could be programmed in order to be called as `mdoc(1)` or `man(1)` to handle the `roff` standard macros and to do the layout, replacing all the filters by a set of macros à la LaTEX—LaTEX is not an engine: it is only a set of macros running on a compatible engine.

PROTE has been extended with the very first `roff(1)` compatibility feature: treating a leading dot in a new line as an escape command—hence interpreting the remaining token as a macro.

5. **Pre-digesting (compiling) the set of macros.** TEX is an interpreter. In order to not have to first interpret all the macros before doing the processing (as `roff(1)` does), a special version of TEX is used to pre-digest the set of macros and then to dump this pre-digested set. This is why users think there is such a thing as a LaTEX engine, because there is apparently a `latex(1)` program. But this is just `virtex(1)` (or, in kerTEX, `virprote(1)`) masquerading as `latex(1)`, the basename of the utility being used to load the corresponding pre-digested set of macros.

6. **How** PROTE **could masquerad as** `roff(1)`**.** In exactly the same way, `roff(1)` could be simply `virprote(1)` loading the pre-digested set of macros whether allowing to treat what `roff(1)` now treats, or allowing to do also all the pre-processing needed by the `mdoc` or `man` set of `roff` macros. Since `roff(1)` has limited interpreter capabilities, in Unix like spirit, more involved pre-processing is done with text utilities—filters. This could be replaced by PROTE using the TEX interpreter capabilities and using the pre-digested version of the macros (a feature `roff(1)` doesn't have) to speed up the process.

In fact, letting, underneath, TEX (PROTE) do the layout computation, the `roff` like set of macros, unchanged, could offer a sufficiently large and comparatively simpler set of macros to be used instead of or concurrently to LaTEX.

7. **Pushing it further...** TEX computes a layout. One could think of (dream of) TEX able to master html or xhtml and then having TEX (PROTE) masquerading as, say, `html(1)` to render Internet pages. Writing a browser would be simply displaying the images defined by the TEX layout engine...

8. **More about fonts**. The Computer Modern series of fonts is provided. With support for virtual fonts, allowing a glyph algebra, one can combine existing glyphs in order to produce new fonts—for example to provide a font with accented letters. So a system can have an already rich set of fonts with these standard fonts.

TEX has been partially rewritten to be 8bits clean. This does mean that it can handle UTF8. Support for more than 256 codepoints can be added without a lot of modifications, in a compatible way: using a font not as a leaf filename but as a directory, with 256 glyphs chunks. Depending on the codepoint computed from UTF8 imput, the font will remain the same, but the DVI will have an instruction to select another 256 glyphs chunk in the directory. If the font is not a directory, then the legacy behavior will happen.

9. **... and more fonts.** But there exist also the *Hershey's fonts*. These fonts have not been designed by Dr Hershey, but have been digitized: a huge set of fonts, including Cyrillic, Greek and even three versions of Japanese ones, are described as line strings, with various resolutions. The result can be quite good.

These fonts were heavily used in CAD systems, early, because such glyphs composed only of linestrings can easily be subjected to an affine transformation—rotation and scaling of text were not available in the early windowing systems; CAD softwares did them using the Hershey's fonts, that are indeed still provided by some of them.

In fact, these fonts could be resurrected, borrowing their metrics (unknown) from the metrics published for standard PostScript fonts.

And the rectification of fonts (only linestrings) could be an intermediary format used in the DVI processing, allowing correct scaling in some defined range, without relying on PostScript and T1 or on some computing intensive (relatively) processing of quadratic curves.

10. **Extending the drawing capabilities**. Using as a template `METAFONT`, John Hobby wrote `MetaPost`. `MetaPost` uses the same language as `METAFONT` but produces basic PostScript as output.

The aim is to extend `METAFONT` to create, let's say, `MetaDRAW`, retaining the fonts capabilities and adding the general drawing capabilities present in MetaPost, but producing DVI as output format.

DVI can be extended. The legacy 256 range opcodes (some still non affected) can be extended to use one free opcode to mean: switch to another page of opcodes. Thus allowing extensions, that could be ignored for security or implementation reasons by the driver handling the DVI+. One extension number will be reserved for "private" use.

DVI is in the same league as PDF. But not everything present in PDF has to be present in DVI. And things can be done differently and remain unencumbered. But some things can certainly be added.

11. `METAFONT` **is a RIP**. All the solutions, today, require finally a PostScript interpreter or at the very least a PDF library able to rasterize the drawing primitives. But `METAFONT` is a RIP: it is its function to transform the drawing (of fonts) in a matrix of pixels, with an excellent result.

Instead of rendering only the fonts, at a size and for a resolution fixed, and having the DVI format only indicating where to put these predefined images on a page, the DVI format could be extended to allow more primitives, and `METAFONT` (or `MetaDRAW`) could be used as the RIP, rasterizing in post-processing the DVI.

12. **Rethinking the user interface**. TEX user interface is rudimentary and when using an instance of an engine for interactive use—for example for computing numbers, with Jean-François Burnol's `xintsession`—, it would be more than welcome to be able to recall a previous command or to edit it.

In fact, since all these command manipulations have to take place before feeding the line to the engine, that ignores control characters, the TEX engine proper is untouched and the control characters can be used to interpret line editing; and this can be done with standard C functions. Furthermore, since the engine has chewed all the available commands (macros), one could also ask the engine to guide the composition of the command line by displaying the expected parameters of a given macro or indicate clearly that one parameter has not the expected type of value.

13. **A solution at reach, with potential powerful possibilities**. Since all the pieces are almost already there, an OS could provide as its base a small powerful layout and drawing engine, accommodating not only the OS own documentation needs, but the user needs as well.

But with a DVI format extended; a layout engine, including for mathematics—TEX—; and a RIP, all the pieces could be there to build GUIs. If, for example, TEX (or PROTE) could be programmed to absorb HTML macros, an HTML browser could be built with a few lines of code.

14. **The missing manual**. Donald E Knuth has written what is in some sense the definitive description of his typographic system. But, in fact, there is one missing manual: how all these pieces fit together. Writing this manual, that should be considerably shorter than any of the five volumes of D.E.K.' series, is also a task that remains to be done.

15. **Present status of the work**. There is indeed some work to do, but a major part of the work has already been done. For the remaining part, there are things that can be started now because they will be needed whatever choice is finally made to achieve the goal, and things that are not fixed yet because they do depend on the integration of the solution in an Operating System.

What has been done and is already available anywhere, including in Plan9:

- Extracting the needle from the haystack: kerTEX provides the core (the "kernel") of the solution, without depending on anything. The size is quite reasonable since the initial installation can be reduced to not more, in size, than the current `roff` incarnations, while providing more: fonts and rasterizing;
- Depending only on a libc (neglecting some supplementary primitives needed by LaTEX—that have a default dummy implementation if needed routines are not provided by the libc—it depends even only on standard C, and not even on POSIX);
- Allowing extensions, added to the core: there is a packaging system, that has even the property to allow "live" updating since what is called "recipes" in kerTEX can work with updated upstream releases as long as the way things are unpacked is unchanged—the contents can change, as long as the entry point to generate the files to install is unchanged;
- Extending TEX to support every set of macros: this is done with PROTE, allowing full compatibility with D. E. Knuth's plain, ε-TEX and LaTEX.

- Extending PROTE to offer features to master `roff` like macros: the first step has been done, with the new primitive “roffdotallowing to treat a leading dot in a new line as an escape commmand, i.e. calling the macro whose name is the following token.

What is currently worked on, because it will be needed whatever the final choices:

- Writing a DVI driver to handle DVI processing with the PostScript output, taken from `dvips(1)`, being only one of the output formats;
- Adding the generation of PDF from DVI, since it has less requirements for post-processing (all the programming capabilities of PostScript are useless, since the programmability, the flows of control and so on are available at the interpreters level: TEX, `METAFONT` or `MetaPost`);
- Extracting the rasterizing capabilities from `METAFONT` in a dedicated library;
- Resurrecting the Hershey’s fonts;
- Gathering all the notes I have about the TEX system and digest them in a well organized, easily understandable small document, providing the missing manual.

And finally what needs more thinking in order to not waste time implementing things that will be dropped because they do not fit:

- What commands to add to the DVI format in order, perhaps, to allow using DVI for a Graphical User Interface? (There is prior art in this area: X11 had a PostScript version; Windows has/had a GDI for drawing, whether a screen or a static image.)
- What choices to make for the input User Interface (line oriented)? From the current OSes User Interface in this area, there are mainly things to drop. But what to retain?