All those TEX's

This is about TEX, the program that is used as part of the large suite of resources that make up what we call a 'TEX distribution', which is used to typeset documents. There are many flavors of this program and all end with tex. But not everything in a distribution that ends with these three characters is a typesetting program. For instance, latex launches the a macro package LATEX, code that feeds the program tex to do something useful. Other formats are Plain (no tex appended) or ConTEXt (tex in the middle. Just take a look at the binary path of the TEX distribution to get an idea. When you see pdftex it is the program, when you see pdflatex it is the macro package LATEX using the pdfTEX program. You won't find this for ConTEXt as we don't use that model of mixing program names and macro package names.

Here I will discuss the programs, not the macro packages that use them. When you look at a complete TEXLive installation, you will see many TEX binaries. (I will use the verbatim names to indicate that we're talking of programs). Of course there is the original tex. Then there is its also official extended version etex, which is mostly known for adding some more primitives and more registers. There can be aleph, which is a stable variant of omega meant for handling more complex scripts. When pdf became popular the pdftex program popped up: this was the first TEX engine that has a backend built in. Before that you always had to run an additional program to convert the native dvi output of TEX into for instance PostScript. Much later, xetex showed up, that, like Omega, dealt with more complex scripts, but using recent font technologies. Eventually we saw luatex enter the landscape, an engine that opened up the internals with the Lua script subsystem; it was basically a follow up on pdftex and aleph.

The previous paragraph mentions a lot of variants and there are plenty more. For cjk and especially Japanese there are ptex, eptex, uptex, euptex. Parallel to luatex we have luajittex and luahbtex. As a follow up on the (presumed stable) luatex the ConTEXt community now develops luametatex. A not yet mentioned side track is nts (New TEX system), a rewrite of good old TEX in Java, which in the end didn't take off and was never really used.

There are even more TEX's and they came and went. There was enctex which added encoding support, there were emtex and hugeemtex that didn't add functionality but made more possible by removing some limits on memory and such; these were quite important. Then there were vendors of TEX systems that came up with variants (some had extra capabilities), like microtex, pctex, yandytex and vtex but they never became part of the public effort.

For sure there are more, and I know this because not so long ago, when I cleaned up some of my archives, I found eetex (extended ε-TEX), and suddenly remembered that Taco Hoekwater and I indeed had experimented with some extensions that we had in mind but that never made it into ε-TEX. I had completely forgotten about it, probably because we moved on to LuaTEX. It is the reason why I wrap this up here.

In parallel there have been some developments in the graphic counterparts. Knuts metafont program got a Lua enhanced cousin mflua while metapost (aka mpost or mp) became a library that is embedded in LuaTEX (and gets a follow up in LuaMetaTEX). I will not discuss these here.

If we look back at all this, we need to keep in mind that originally TEX was made by Don Knuth for typesetting his books. These are in English (although over time due to references he needed to handle different scripts than Latin, be it just snippets and not whole paragraphs). Much development of successors was the result of demands with respect to scripts other than Latin and languages other than English. Given the fact that (at least in my country) English seems to become more dominant (kids use it, universities switch to it) one can wonder if at some point the traditional engine can just serve us as well.

The original tex program was actually extended once: support for mixed usage of multiple languages became possible. But apart from that, the standard program has been pretty stable in terms of functionality. Of course, the parts that made the extension interface have seen changes but that was foreseeable. For instance, the file system hooks into the kpse library and one can execute programs via the \write command. Virtual font technology was also an extension but that didn't require a change in the program but involved postprocessing the dvi files.

The first major 'upgrade' was ε-TEX. For quite a while extensions were discussed but at some point the first version became available. For me, once pdfTEX incorporated these extensions, it became the default. So what did it bring? First of all we got more than 256 registers (counters, dimensions, etc.). Then there are some extra primitives, for instance `\protected` that permits the definition of unexpandable macros (although before that one could simulate it at the cost of some overhead) and convenient ways to test the existence of a macro with `\ifdefined` and `\ifcsname`. Although not strictly needed, one could use `\dimexpr` for expressions. A probably seldom used extension was the (paragraph bound) right to left typesetting. That actually is a less large extension than one might imagine: we just signal where the direction changes and the backend deals with the reverse flushing. It was mostly about convenience.

The Omega project (later followed up by Aleph) didn't provide the additional programming related primitives but made the use of wide fonts possible. It did extend the number of registers, just by bumping the limits. As a consequence it was much more demanding with respect to memory. The first time I heard of ε-TEX and Omega was at the 1995 euroTEX meeting organized by the ntg and I was sort of surprised by the sometimes emotional clash between the supporters of these two variants. Actually it was the first time I became aware of TEX politics in general, but that is another story. It was also the time that I realized that practical discussions could be obscured by nitpicking about speaking the right terminology (token, node, primitive, expansion, gut, stomach, etc.) and that one could best keep silent about some issues.

The pdfTEX follow up had quite some impact: as mentioned it had a backend built in, but it also permitted hyperlinks and such by means of additional primitives. It added a couple more, for instance for generating random numbers. But it actually was a research project: the frontend was extended with so called character protrusion (which lets glyphs hang into the margin) and expansion (a way to make the output look better by scaling shapes horizontally). Both these extensions were integrated in the paragraph builder and are thereby extending core code. Adding some primitives to the macro processor is one thing, adapting a very fundamental property of the typesetting machinery is something else. Users could get excited: TEX renders a text even better (of course hardly anyone notices this, even TEX users, as experiments proved).

In the end Omega never took off, probably because there was never a really stable version and because at some time XETEX showed up. This variant was first only available on Apple computers because it depends on third party libraries. Later, ports to other systems showed up. Using libraries is not specific for XETEX. For instance pdfTEX uses them for embedding images. But, as that is actually a (backend) extension it is not critical. Using libraries in the frontend is more tricky as it adds a dependency and the whole idea about TEX was that is is independent. The fact that after a while XETEX switched libraries is an indication of this dependency. But, if a user can live with that, it's okay. The same is true for (possibly changing) fonts provided by the operating system. Not all users care too strongly about long term compatibility. In fact, most users work on a document, and once finished store some pdf copy some place and then move on and forget about it.

It must be noted that where ε-TEX has some limited right to left support, Omega supports more. That has some more impact on all kinds of calculations in the machinery because when one goes vertical the width is swapped with the height/depth and therefore the progression is calculated differently.

Naturally, in order to deal with scripts other than Latin, XETEX did add some primitives. I must admit that I never looked into those, as ConTEXt only added support for wide fonts. Maybe these extensions were natural for LATEX, but I never saw a reason to adapt the ConTEXt machinery to it, also because some pdfTEX features were lacking in XETEX that ConTEXt assumed to be present (for the kind of usage it is meant for). But we can safely say that the impact of XETEX was that the TEX community became aware that there were new font technologies that were taking over the existing ones used till now. One thing that is worth noticing is that XETEX is still pretty much a traditional TEX engine: it does for instance OpenType math in a traditional TEX way. This is understandable as one realizes that the OpenType math standard was kind of fuzzy for quite a while. A consequence is that for instance the OpenType math fonts produced by the Gust foundation are a kind of hybrid. Later versions adopted some more pdfTEX features like expansion and protrusion.

I skip the Japanese TEX engines because they serve a very specific audience and provide features for scripts that don't hyphenate but use specific spacing and line breaks by injecting glues and penalties. One should keep in mind that before Unicode all kinds of encodings were used for these scripts and the 256 limitations of traditional TEX were not suited for that. Add to that demands for vertical typesetting and it will be clear that a specialized engine makes sense. It actually fits perfectly in the original idea that one could extend TEX for any purpose. It is a typical example of where one can argue that users should switch to for instance XETEX or LuaTEX but these were not available and therefore there is no reason to ditch a good working system just because some new (yet unproven) alternative shows up a while later.

We now arrive at LuaTEX. It started as an experiment in 2005 where a Lua interpreter was added to pdfTEX. One could pipe data into the TEX machinery and query some properties, like the values of registers. At some point the project sped up because Idris Hamid got involved. He was one of the few ConTEXt users who used Omega (which it actually did support to some extent) but he was not satisfied with the results. His oriental TEX project helped pushing the LuaTEX project forward. The idea was that by opening up the internals of TEX we could do things with fonts and paragraph building that were not possible before. The alternative, XETEX was not suitable for him as it was too bound to what the libraries provides (rendering then depends on what library gets used and what is possible at what time). But, dealing with scripts and fonts is just one aspect of LuaTEX. For instance more primitives were added and the math machinery got an additional OpenType code path. Memory constraints were lifted and all became Unicode internally. Each stage in the typesetting process can be intercepted, overloaded, extended.

Where the ε-TEX and Omega extensions were the result of many years of discussion, the pdfTEX, XETEX and LuaTEX originate in practical demands. Very small development teams that made fast decisions made that possible.

Let's give some more examples of extensions in LuaTEX. Because pdfTEX is the starting point there is protrusion and expansion, but these mechanisms have been promoted to core functionality. The same is true for embedding images and content reuse: these are now core features. This makes it possible to implement them more naturally and efficiently. All the backend related functionality (literal pdf, hyperlinks, etc) is now collected in a few extension primitives and the code is better isolated. This took a bit of effort but is in my opinion better. Support for directions comes from Omega and after consulting with its authors it was decided that only four made sense. Here we also promoted the directionality to core features instead of extensions. Because we wanted to serve Omega users too extended tfm fonts can be read, not that there are many of them, which fits nicely into the whole machinery going 32 instead of 8 bits. Instead of the ε-TEX register model, where register numbers larger than 255 were implemented differently, we adopted the Omega model of just bumping 256 to 65536 (and of course, 16K would have been sufficient too but the additional memory it uses can be neglected compared to what other programs use and/or what resources users carry on their machines).

The modus operandi for extending TEX is to take the original literate web sources and define change files. The pdfTEX program already deviated from that by using a monolithic source. But still Pascal is used for the body of core code. It gets translated to C before being compiled. In the LuaTEX project Taco Hoekwater took that converted code and laid the foundation for what became the original LuaTEX code base.

Some extensions relate to the fact that we have Lua and have access to TEX's internal node lists for manipulations. An example is the concept of attributes. By setting an attribute to a value, the current nodes (glyphs, kerns, glue, penalties, boxes, etc) get these as properties and one can query them at the Lua end. This basically permits variables to travel with nodes and act accordingly. One can for instance implement color support this way. Instead of injecting literal or special nodes that themselves can interfere we now can have information that does not interfere at all (apart from maybe some performance hit). I think that conceptually this is pretty nice.

At the Lua one has access to the TEX internals but one can also use specific token scanners to fetch information from the input streams. In principle one can create new primitives this way. It is always a chicken-egg question what works better but the possibility is there. There are many such conceptual additions in LuaTEX, which for sure makes it the most 'aggressive' extension of TEX so far. One reason for these experiments and extensions is that Lua is such a nice and suitable language for this purpose.

Of course a fundamental part of LuaTEX is the embedded MetaPost library. For sure the fact that ConTEXt integrates MetaPost has been the main reason for that.

The ConTEXt macro package is well adapted to LuaTEX and the fact that its users are always willing to update made the development of LuaTEX possible. However, we are now in a stage that other macro packages use it so LuaTEX has entered a state where nothing more gets added. The LATEX macro package now also supports LuaTEX, although it uses a variant that falls back on a library to deal with fonts (like XETEX does).

With LuaTEX being frozen (of course bugs will be fixed), further exploration and development is now moved to LuaMetaTEX, again in the perspective of ConTEXt. I will not go into details apart from saying that is is a lightweight version of LuaTEX. More is delegated to Lua, which already happened in ConTEXt anyway, but also some extra primitives were added, mostly to enable writing nicer looking code. However, a major aspect is that this program uses a lean and mean code base, is supposed to compile out of the box, and that sources will be an integral part of the ConTEXt code base, so that users are always in sync.

So, to summarize: we started with `tex` and moved on to `etex` and `pdftex`. At some point `omega` and `xetex` filled the Unicode and script gaps, but it now looks

like luatex is becoming popular. Although luatex is the reference implementation, LaTeX exclusively uses luahbtex, while ConTeXt has a version that targets at luametatex. In parallel, the [e][u][p]tex engines fill the specific needs for Japanese users. In most cases, good old tex and less old etex are just shortcuts to pdftex which is compatible but has the PDF backend on board. That 8 bit engine is not only faster than the more recent engines, but also suits quite well for a large audience, simply because for articles, thesis, etc. (written in a Latin script, most often English) it fits the bill well.

I deliberately didn't mention names and years as well as detailed pros and cons. A user should have the freedom to choose what suits best. I'm not sure how well TeX would have evolved or will evolve in these days of polarized views on operating systems, changing popularity of languages, many (also open source) projects being set up to eventually be monetized. We live in a time where so called influencers play a role, where experience and age often matters less than being fancy or able to target audiences. Where something called a standard today is replaced quickly by a new one tomorrow. Where stability and long term usage of a program is only a valid argument for a few. Where one can read claims that one should use this or that because it is todays fashion instead of the older thing that was the actually the only way to achieve something at all a while ago. Where a presence on facebook, twitter, instagram, whatsapp, stack exchange is also an indication of being around at all. Where hits, likes, badges, bounties all play a role in competing and self promotion. Where today's standards are tomorrow's drawbacks. Where even in the TeX community politics seem to creep in. Maybe you can best not tell what is your favorite TeX engine because what is hip today makes you look out of place tomorrow.

Hans Hagen
February 2020