Illiterate programming

Introduction

I write this 20 years after we started the LuaTEX project, which was 2005. At that time I used pdfTEX, for obvious reasons: it was a mature engine, had a pdf backend built in, came with nice features like expansion, had a few extensions at the primitive level, and I was involved in the development, testing, discussions etc. I was quite happy with it and ConTEXt always supported the latest greatest smoothly. And after decades MkII still works well with the current pdfTEX, which has evolved a little but we never followed up on that.

During the 20 years we reported frequently about how we moved on to LuaTEX, which got frozen after some 15 years but got a follow up in LuaMetaTEX. Among the reasons mentioned for starting the project were the potential to extend the inner working of a TEX engine via a Lua interface, the fact that the world moved on to Unicode and OpenType fonts becoming more popular and widespread. We also had to support right-to-left typesetting in a more comfortable way which is why the Oriental TEX project became instrumental in the first five years.

In the next sections I will reflect a bit on how the code base evolved and how that fits into how the average TEX code base looks like. The bottom line is that we moved from literate to 'illiterate' programming.

Woven webs

If you look at the TEX live codebase for the familiar engines of course your first try to find the original. In the October 2025 snapshot we find:

```
1,031,999 tex.web
  180,075 tex.ch
```

Here the big file defines the program and the file is big because it has program code as well as documentation. If you open the file you'll notice that it's quite dense. If spacing had be more liberal and the various directives more verbose the file would be way bigger but think for a moment about those times: the first 10 MB hard disks in personal computers that showed up years after TEX was written were larger than a modern mini pc and weighted more than a decent laptop. You not only have that 1 MB text file, you also need to compile it, so you need a software infrastructure, you need to test the binary, which also takes space, save a memory dump, collect some fonts, generate bitmaps, etc. It's a miracle all worked out so well. Oh, and we didn't mention that in those times printing hundreds of pages in a decent resolution took some time too.

The small file is what is called a change file. From the big `.web` file we can 'weave' documentation (like the TEX the program book) and 'tangle' a Pascal program. The change file is applied in the process and it patches the to be woven web. Such a change file typically deals with system dependent issues but can also add or upgrade features. If you keep in mind that the first version of TEX was configured for 32 bit memory words, it will not come as a surprise that the first change was going 64 bit so that the engine for instance could handle more so called nodes.[1]

1. These are the times of big and huge TEX's and hooking into memory extenders on 640K personal computers. The times when processing pages could make for a coffee break.

An example of change file that adds functionality is ε-TEX, one of the first extension projects:[2]

```
225,207 etex.ch
```

That one is about a quarter of the original. There is no `.web` file because it patches the original TEX file. This is not the case for pdfTEX:

```
1,577,910 pdftex.web
   16,272 pdftex.ch
```

Here we see a copy of `tex.web` with adaptations but still there is some change file. It deals with the fact that the `tex.ch` file is also applied as well as some mlTEX functionality. Keep in mind that we operate in an infrastructure where code is shared, especially interfaces to the file system and environment (web2c & kpse). The pdfTEX program has a backend and interfaces to various graphic libraries and all those are done in C. So, in addition to these `.web` and `.ch` files we also find `.h` and `.c` files. We have a mix if literate and less literate code.

We see the same with XETEX, where due to less added functionality (code and documentation) the web file has grown less but that is compensated by more of the new front-end functionality done in C. Also, keep in mind that this program delegates the pdf backend to dvipdfmx by piping out dvi.

```
1,378,780 xetex.web
   31,800 xetex.ch
```

An interesting case is Aleph, the production version of Omega. At TEX meetings this multi-directional, large font variant was often presented in a way that made it sound like it was not coded like TEX due to various limitations, e.g. hardware. It would even be in a transition to C. However, when you look at the code base in TEXLive you'll notice that it is all change files and a few C files. It actually stays within the concept of patching the original and where in one way it does away with limitations that other engines work around, it otherwise is kind of conservative. Of course it might relate to the fact that the added functionality is in transforming input and feeding the backend. For instance, the directionality doesn't really need many changes in core code. The 35 change files add up to 813,829 bytes and files range from 124 to 174,831 bytes.

We end with MetaPost. Here we have no `.web` files but `.w` files because this program is not written in Pascal but in C. There are no change files in its directory. Some functionality is delegated to specific files, like the number systems that are supported. The binary, decimal and double variants depend on other code, so in the end we do use additional header and code files. The optional png and svg backends also add (quite huge) dependencies.

```
1,213,713 mp.w
   63,738 mpmath.w
   68,096 mpmathbinary.w
   73,831 mpmathdecimal.w
   52,224 mpmathdouble.w
   45,035 mpost.w
   12,369 mpstrings.w
```

Here the big one is actually the library and the small `mpost.w` implements the command line program.

2. I first heard about ε-TEX and Omega at the first TEX conference I attended, 1995, Arnhem NL. I was a bit amazed that there was this competition going on.

Changing the web

In the previous lists of programs we miss LuaTEX, so let's look at that now. Because we started from pdfTEX the first version was just a copy and therefore quite similar. The Lua interface gave access to a few internals and we could write back to the input. Of course by adding Lua some C files were added. Apart from some extra primitives the TEX system was just the same.

However, as we decided to go forward and really open up the internals it became clear that the single file approach would not accommodate that well. But, with future Lua interfaces being C driven and with upcoming demands with respect to fonts in mind, it made sense to move away from Pascal. Keep in mind that for instance for oriental TEX we had to load relatively large fonts (for which we initially took code from FontForge) and process the so called node lists that makes boxes and paragraphs. We didn't want to go the XETEX route by depending on operating system features and/or specific libraries that could change (and in the end did change).[3] For us, Lua was the way to go: after all, what is a standard (fonts, pdf, graphics) if they lock into specific libraries; we wanted a programmable extensions system, pretty much like TEX itself.

So, to summarize a few years into a few sentences: Taco and Hartmut split up the code base, moved from Pascal web (`.web`) to C (but kept the comments) and eventually ended up with C web (`.w`) files. The idea was even to write new code as a web but in practice that was just too much work and never really happened. It anyway meant that we no longer used the web2c route but the cweb route.

This process actually took years because at the same time we also added functionality, improved some existing mechanisms (like more efficient expansion), did some cleanup, etc. We tried out ideas, implemented solutions in Lua, adapted as we went and had to document as well as report on it: see ConTEXt documentation, TugBoat, Maps, etc. It was very convenient that we let ConTEXt MkII evolve into MkIV and that ConTEXt users just joined in testing. Basically LuaTEX could always be used in real production flows, as long as one kept ConTEXt and the engine in sync.

When we decided to more of less freeze the engine, around version 1 in 2016, we had a mixed code base, which was a good moment because Taco no longer could allocate time; keep in mind that this is all unpaid volunteer work. After that I started to add pending functionality and some more interfaces, while Luigi took care of integration in TEXLive. It kept us busy for a few years. In the process we change the `.w` files to just `.h` and `.c` files and keep the comments as such. We didn't expect anyone to ever generate a pdf from the source anyway, also because it never could match the original Knuth quality. One needs discipline for reaching that level and the combined workload (TEX, MetaPost, macro package, documenting, usage) simply didn't provide room for that. Also, editors with syntax highlighting, cross file searching, grepping, high resolution windowed display technology, and whatever, had evolved and looking at the documented source was as good as looking at a (to be generated) pdf on screen.

There was an exception to the rule: MetaPost. That big file was already in documented cweb and splitting it up is a major effort. It did mean that we had a dependency on a converter but we'll see how that worked out.

Becoming illiterate

We've now arrived at LuaMetaTEX. When the more definitive code freeze of LuaTEX was decided in 2018, the development moved to a new variant. At that time the code base can be summarized as follows. There is a set of files that can be divided in those implementing the TEX engine and user interface (interpreter), Lua libraries that are kind of stand alone, Lua libraries that interface with the engine, libraries that give

3. We do support XETEX in ConTEXt MkII but never used its features beyond loading fonts because they didn't match our approach. So, I can't really comment on how it works internally.

access to fonts or generate pdf, and of course the MetaPost library. Even libraries dealing with bitmap images could be replaced by Lua solutions. The files are stored in a TEXLive compliant structure, and use its build infrastructure. It's a bit messy mix. An optional library interface makes it possible to add specific features via Lua but ConTEXt will never depend on that.

Among the decisions made when I started LuaMetaTEX was that the font loader code could go away, because after all in ConTEXt we already did all of this in Lua. The same was true for the backend: we experimented that in LuaTEX in MkIV (but later removed that option there). The decision also made for a new version of ConTEXt: MkXL or, as distribution LMTX, because at that time it became clear that we had to distribute both at the same time.

I was surprised to find out that in the end compilation was not that hard, not even with the native windows compiler.[4] It made for a way more convenient development cycle but there was one dependency left: MetaPost was still webbed. In order not to be dependent on the web2c toolchain I wrote a Lua script that did the same and also had the possibility for plugging in Lua code to fix certain issues. A side effect was that I got better formatting of the C (which presumingly was not meant to be looked at, just compiled) and a few compiler warnings related to indentation. When I really started to extend MetaPost the `.c` file was used and back ported to `.w` so after a few years, sentiments set aside, the cweb was dropped from the source tree. It triggered another round of cleanups.

So, is there anything literate left? The answer is "partial". The various comments are spread over the sources as C comments. They come from original TEX, pdfTEX, LuaTEX and of course from the author. The original comments do not always apply but there is also a history to be told and the general ideas and approaches of TEX don't change. We deliberately always tried to stay close to the original when possible.

I have it on my todo list to clean it up a bit more. The challenge is to keep the explanations, even if they no longer apply that way. History also matters, and it's not like many people read it. A better source of explaining what the engine does are the articles, development documents, low level manuals and ConTEXt code base. It's more important to write for users than for whoever looks at the source.

What I didn't stress enough is that the code base is independent. There is some third party module code, like (of course) Lua, memory management, compression, pdf parsing, specialized graphic stuff related to MetaPost but all of it is in the tree and distributed with ConTEXt. Updates never happen automatically, changes a are checked (diffed) first. When a user installs cmake and a compilers building a binary is easy. The codebase can quite well be explored from Visual Studio, VSCode or likewise. (But rest assured: it's free of AI interference.)

Conclusion

The LuaTEX and LuaMetaTEX code bases differ from other TEX engines. Where LuaTEX integrates in the TEXLive code base, LuaMetaTEX is independent. Both don't use the web main file plus change files approach, although LuaTEX does so for mplib, which actually then has its own source tree. So, both don't really qualify any longer as literate programs, but they do have plenty traces of the idea that the source should explain a bit what happens.

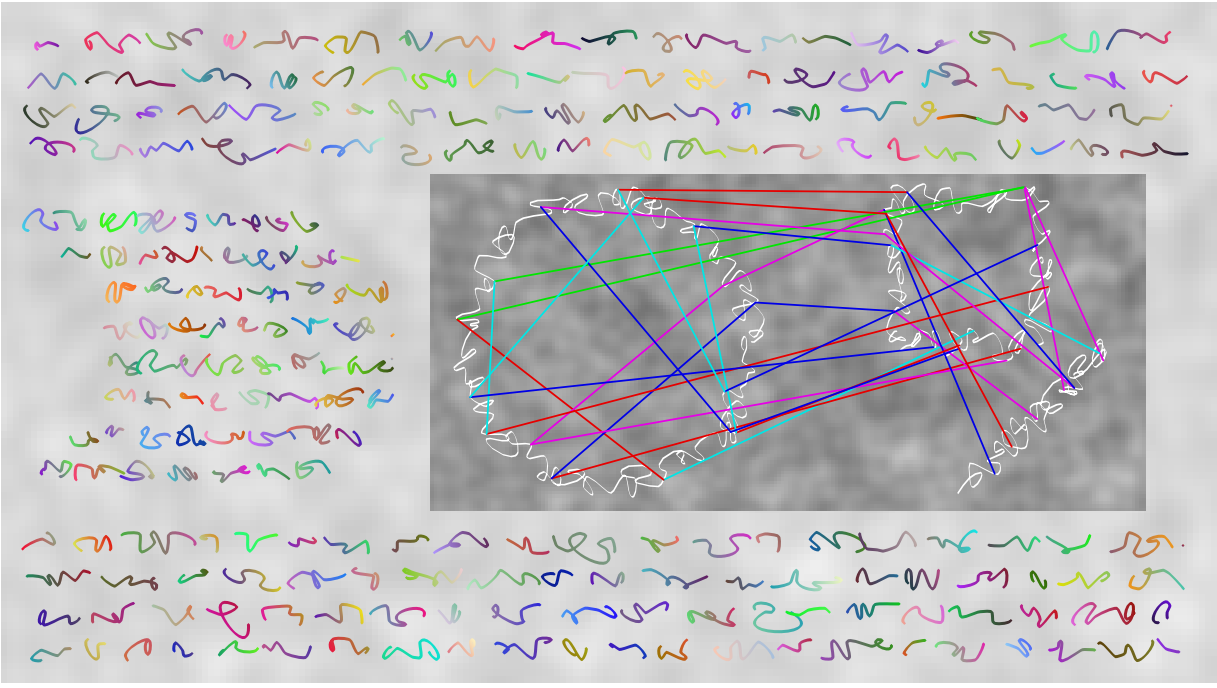
I never realized that LuaTEX is an outlier because it doesn't have this change file model. The objectives of the project also made us change for instance the size of nodes and font resources which means that there are lots of accumulated changes.

4. When working on LuaTEX, I cross compile on the Linux subsystem for Windows using a build script that Luigi made. The files are organized in the TEXLive way, and the build script fits into that. Cross compiling LuaMetaTEX is faster and for various platforms we use Mojca's ConTEXt garden build infrastructure. Installing ConTEXt and LuaMetaTEX is done by the engine itself as it can act as Lua interpreter and therefore download and install files. It is a self-contained setup.

In an other musing I talk a bit about the way character nodes are managed in original TEX and derivates. The change file model is a sort of insurance that the related optimizations are kept but also might limit the solution space. So it's a blessing on the one hand and maybe a curse on the others. I can safely say that we would not have gone as far as we went if we'd locked into the approach that the other engines took.

So, kudos to Hartmut Henkel and Taco Hoekwater for having the courage to divert soon after we started and of course thanks to Don Knuth for providing the well known and very literate documented source to do all this, the ultimate source I still go back to when in doubt.

In a ConTEXt development team discussion, Keith McKay offered to write some documentation and we all agreed to adopt his more futuristic style of literate programming. An example is given below.

After all, there is no need to stick to TEX alone and we have to make sure that only humans can get the picture what we want to achieve: it's all about beauty, not about turning what we do in artificial slop.

Hans Hagen