Following up on LuaTEX

Introduction

From the ConTEXt mailing list one can get an impression of how ConTEXt is used. It's mostly for typesetting books, manuals and sometimes special products and the input is either TEX or xml. I have no clue about articles, but in that domain MS Word, LATEX and maybe even Google Docs compete each other. As LATEX is encouraged (or maybe even enforced) on students for writing reports and such there is also not some body of default styles that are part of the package and probably hardly any user expects pre-cooked solutions. Anyway, I'm always surprised by the diversity of problems that users try to solve. From posts you can also deduce that using Lua becomes more popular as a means to deal with data from various sources. The only measure that we have to what extent the functionality provided is used is the quality of answers on mailing lists and support platforms and they are often of high quality.

It is interesting that outsiders sometimes think that ConTEXt development is driven by commercial usage but this mostly untrue. We started serious development in the 90's because we wanted the materials that we produced to look sort of nice and TEX could be used for that. Over a few decades the package evolved, but to a large extent that was driven by users as well as by exploration of specific user demands. The fact that we could use some features ourselves was nice, but the investment in time is in no way compensated by the revenues. It's the joy of playing with TEX, MetaPost and Lua as well as the interaction with the ConTEXt community that drives development.

Sometimes suggestions are made that one should use a 'proper' computer language instead of TEX lingua, but personally I would not use such an approach as it is rather painful. There is nothing wrong with coding a document in some specific document structure or content related language instead of clumsy function calls. Take alone the concept of grouping properties. It is therefore also not correct to conclude that ConTEXt development eventually will end up with a pure (e.g.) Lua interface to solve typesetting related problems. Hardly anyone asked us to provide solutions using TEX: if we use ConTEXt it's because we know how to use it efficiently. Using TEX coding is a convenient side effect no one really cares about as long as something works.

It must be noted that I, and probably also other users, see LuaTEX as a convenient Lua engine and especially in combination with the load of libraries that comes with ConTEXt, it can do a lot. An extreme example is that I run some domotica applications with LuaTEX and Lua scripts. At the last ConTEXt meeting Taco demonstrated how he uses LuaTEX and friends for his miniature railway tracks. Other presentations demonstrated massive number crunching and visualization of data collected over time.

It often puzzles me why we never run into customers who see(k) the potential of this TEX, MetaPost, and Lua mix. It is pretty hard to beat this combination in performance, quality, convenience and time spent to reach a solution, but it is probably too strange a mix. This lack of market also means that there is no real long term agenda needed and therefore development only relates to exploration of (often very specialized) user demand on the one hand and robust continuity on the other.

What next

Now that LuaTEX has reached a more or less stable state, the question is: what will happen next. Development has been bound rather tightly to ConTEXt development, but now that other macro packages also (can) use LuaTEX, we're sort of stuck and need to freeze functionality in order to guarantee long term stability within the TEX ecosystem. However, as the introduction suggests, we're in no way bound to this, simply because no one pays for development. We can move on as we like. Apart from changes in the way fonts and encodings are dealt with (more modern techniques) ConTEXt is pretty stable in most areas. The user interface is downward compatible so for users progress only has benefits.

There is a frontend that stays rather close to original TEX and MetaPost, there is a backend that produces efficient pdf and provides a high level of control and there are plenty of hooks to make the machinery do as you wish, for instance using Lua. Therefore, while with version 1.0 we got out of the beta stage (already

for years LuaTEX could be used in production but the interfaces evolved over time), with version 1.1 we're more or less frozen in functionality for the benefit of general usage.

But that doesn't stop ConTEXt development. We still want to improve some of the more tricky parts of typesetting (the TEX part), we like to add more graphic capabilities (the MetaPost part) and we will provide more interfaces (the Lua part). In doing so, a follow up on LuaTEX will happen, but in such a way that the version 1.x range will not be impacted much. Of course we might occasionally feed back improvements from the follow up to the ancestor. This further development is again closely related to ConTEXt, and just as in the beginning of LuaTEX development, we will not bother with usage elsewhere: we need the freedom to experiment and the ConTEXt ecosystem and user attitudes provide a healthy environment for that.

An impression

During LuaTEX development some ideas have been postponed but it was around the 2018 ConTEXt meeting that I started thinking about a follow up and actually did some explorative coding in the ConTEXt code base. At that meeting this follow up was mentioned and since there were no objections we can say that at that time the next stage was entered.

In the last few months of 2018 and January 2019 the first versions of what we will tag as LuaMetaTEX (which identifies itself as LuaTEX 2.0 internally) became production ready. At that time the ConTEXt code could also emulate the new behavior on regular LuaTEX, and when tested by some other developers no real problems surfaced. Because the beginning of the year is also the moment that TEXLive code freeze happens the public ConTEXt code is not affected. Also, real experiments can only start when we also release the next generation engine, which will happen later this year.

Therefore, the roadmap is kind of like this. After the code freeze, some of the ConTEXt internals will change and although I expect users reporting issues, there is not much that can't be solved fast. And, ConTEXt users have always been willing to update. Then, later in 2019 the alternative engine will surface and at some point we will switch to this one as default. Binaries can be generated using the compile farm at the ConTEXt garden. Of course it will be possible for users to use stock LuaTEX, but just as when we froze MkII, we might bind some functionality to a specific engine. We will still mostly use MkIV code, but the follow up will be tagged lmtx, which stands for Lua, MetaPost, TEX and xml.

A few details

So what is this new engine and why does ConTEXt need it? And can't we do the same with LuaTEX already? I leave answering this to the reader and focus on what we are doing instead. The current LuaTEX program is constructed from a TEX kernel that itself is derived from pdfTEX and snippets of Aleph. It has the MetaPost library embedded and uses Lua as extension language. The Lua instance is accompanied by some libraries. The ConTEXt distribution is a bunch of TEX, MetaPost and Lua files as well as some resources (mostly fonts) that are a subset of what TEXLive comes with. The process of typesetting is managed by some scripts (`mtxrun` and `context`) and on MS Windows a binary stub is used to launch them. In principle, if we assume MkIV being used, the only binary needed is LuaTEX, plus those stubs.

As it had the complete MetaPost library on board, the dependencies in LuaTEX of a few years back were kind of complex, mostly due to the optional png output. But we don't need that at all. So, when we removed that option, the binary shrunk from some 10MB to about 7MB. These bytes contain the frontend and backend code and some libraries (in some cases the Lua and kpse libraries are external to the main binary). But because ConTEXt doesn't need all these libraries and because only a small part of the backend code is used, the question surfaced "What if we go lean and mean?"

So, as a first step I copied the experimental branch and started pruning. It took a couple of iterations to figure out how to do that best while still being able to produce documents, but step by step code could be removed with success. I started with stripping down the backend. For instance, given the nature of ConTEXt code, it was quite doable to generate the page stream ourselves and at some point only font embedding, image inclusion and object management was left. Actually, pdf inclusion could already be brought under ConTEXt control so only bitmaps had to be dealt with. Replacing the bitmap inclusion by Lua code made it possible to drop the png libraries (jpeg inclusion didn't use libraries). For the record: this actually opens up some possibilities for future image inclusion.

Next came font embedding, which involves subsetting. Because we support variable fonts there was already quite some work done to enable kicking out the code that deals with it possible. As starting from the existing code did not make much sense the standard was taken as reference and eventually embedding worked okay. The only drawback here is that we need a bit more font caching but that is hidden from the user. On the positive side we find ourselves with a potentially more powerful virtual font system.

With respect to the backend, that left object management and because most work was already done in ConTEXt itself, kicking out the whole backend was not that much work. In the end all backend code was indeed removed and the PDF file was generated completely by ConTEXt itself. Interesting is that the generated PDF code looks a bit nicer on the one hand and it also turned out to be a bit more efficient. One reason for this is that the backend kind of knows what the frontend does. There was still a performance hit when comparing for instance compiling the LuaTEX manual but at this time there is no such penalty anymore, simply because other ConTEXt components take advantage of this (and maybe also because in the meantime some code in the engine has been optimized).

With the backend code gone the binary already shrunk significantly. Next to go was the (not used in ConTEXt) `slunicode` library. The Lua image library went away too as did the PDF backend interface (I kept `pplib`). Then removing the font loader was on the agenda and it could be dropped easily because it wasn't used at all. Apart from creating an again smaller binary, it also removed dependencies and compilation became easier and faster. When you look at how TEX deals with fonts, it is clear that not that much is needed. For instance OpenType fonts are processed in Lua and that code uses resources not present (and needed) by TEX, and with the backend gone even less is needed. That meant that I could limit the amount of data passed to and stored in TEX for traditional font handling: TEX only needs dimensions, some math properties, and (only) in so called base mode, kerns and ligatures. This makes the memory footprint smaller and might make loading huge fonts a bit faster. When that code was simplified, it was a small step to removing the traditional TFM loading code, and because virtual fonts are only dealt with in the (no longer hard coded) backend that code could also go. Support for traditional eight bit fonts based on TFM files (normally we turn such fonts into wide fonts using information from AFM and PFB files) is still present but a Lua font loader is used instead.

By the time all this was done I decided to remove the KPSE library because in ConTEXt we use a Lua variant. A future version can bring KPSE back as loadable library (in principle one could use the `ffi` interface for this). By removing all kind of dependencies the startup code was also adapted. The engine can now function as a stub too, which means that a ConTEXt distribution becomes simpler as well. At some point we will default to MkIV only and then only one (or two if you add the LuaJIT companion) binary is needed.

I will not go into too much of the details this time (and there are many). As with the development of LuaTEX, I keep track of choices and experiments in a document and some chapters can make it into articles. In between the mentioned stripping down I decided to also clean up the directional model. In spite of what one might think, there is not that much code dealing with this. If we stick to bidirectional typesetting, the frontend doesn't really care what it sees. It was the backend that took care of reversed text streams and now we provide that backend ourselves. So, when wondering what to do with vertical typesetting, I realized that this was actually not supported in a useable fashion at all so why not let that go? And so it went away and directional support was simplified: we only have two directions now. It is worth mentioning that the concept of a body and page direction already didn't make much sense (it is the macro package that deals with this) so these are gone. Also gone are (related) page dimensions: no hard coded backend means that there is no need for them. The keyword driven direction directives with the three letter acronyms have been removed so we only have numeric ones now. It is trivial to define commands that provide the old syntax.

So how about vertical typesetting? As an experiment I've added some properties to boxes: orientation and offsets; more about that in another article. This is something that is dealt with in the backend so it was a nice experiment to see how easy that could be done. This (plus the new directional approach) is something that might be ported back to stock LuaTEX, but let's first see how it will be used in ConTEXt so that it can stabilize.

The current state

By the time we achieved all this (plus a bit of cleanup) the binary had shrunk to well below 3MB which is nice because it is also used as Lua interpreter. There are hardly any dependencies left. It was a bit cumbersome to explore how much of the code base could be stripped and how the build related scripts and setups could be simplified. One can wonder how simple we can get, given the pretty complex TEX build structure, but this is the motto: "If on some mainstream platform Lua can be compiled without hassles, then compiling LuaMetaTEX should be easy too."

This brings us to some of the main objectives of this project. First of all, the idea is to converge to a relatively small LuaMetaTEX, independent of libraries outside our control, one that is easy to compile. It provides an opened up more or less traditional core TEX but delegates much of the modern font handling to Lua. The backend can be a Lua job too. Generic font code we already provide in ConTEXt, and maybe at some point some of the code might show up as generic; we'll see. Apart from providing hooks (callbacks) the engine stays close to original TEX in concept. After all, that core also delegates the backend to extensions or external programs. We kind of go back to the basics.

I use the LuaTEX manual as well as the ConTEXt test suite for testing but for sure user input is needed. Among the issues to take care of is for instance the ability for users to use `tikz`, which dumps low level PDF code and creates objects. So, for that I needed to provide a compatibility layer. However, that is probably the only package from outside ConTEXt that is used, so I might as well make that bit of the interface a runtime option at some point. After all, there is no need to pollute the regular code. In a similar fashion I provide an `img` library mockup but that one might be dropped: users should use the high level interfaces instead. These are typical issues discussed at meetings.

For quite some time this variant will be a moving target that aims at ConTEXt and the reason for this is in the beginning of this article: we do this for the benefit of users but also for fun. We also want a vehicle that permits us to freely experiment (for instance with more advanced virtual fonts) without worrying about usage elsewhere. And eventually it will be, like its ancestor LuaTEX, a stable general purpose alternative, but one a bit closer to where it all started: original TEX.

In order to make experimenting easier, there will be an updated installer for the ConTEXt garden. That one will use HTTP instead of Rsync and use a MkIV only set of resources. All binaries except for the engine and two runners are dropped and no MkII specific and generic files are present. Apart from power users nobody might notice this move.

Hans Hagen