Short report on the state of LuaTEX, 2020

Luigi Scarso

Abstract

A short report on the current status of LuaTEX and its relatives: LuaHBTEX, LuaJITTEX and LuaJITHBTEX.

1 Background

First, let's summarize that there are four programs or "flavors" of LuaTEX:

- LuaTEX, with `lua`;
- LuaJITTEX, with `luajit` (just-in-time compilation capability);
- LuaHBTEX, with `lua` and HarfBuzz;
- LuaJITHBTEX, with `luajit` and HarfBuzz.

The build system manages the task of compiling and linking the shared components and the common components, so that, for example, they all have exactly the same TEX core, and share the same version number and development id.

On 30 May 2019 the first commit of LuaHBTEX, a LuaTEX variant with the ability to use HarfBuzz for glyph shaping, landed in the LuaTEX repository, after a discussion started around mid-February about the merging of HarfTEX by Khaled Hosny and the upcoming LuaTEX in TEX Live 2019. By that time LuaTEX was already frozen for the DVD and, materially, it was not possible to reopen the development. Also in 2019, LuaTEX entered its "bug fixing" phase (more on this below), further complicating the merge.

LuaHBTEX was integrated in the experimental branch of LuaTEX repository on 6 July 2019 and directly after, on 8 July 2019, it landed in the TEX Live repository; The first release of LuaHBTEX, tagged as version `1.11.1`, was on 19 October 2019, giving a wide margin for testing for the next (i.e., now the current) TEX Live 2020.

Together with HarfBuzz, the other component under "observation" in 2019 was the `pplib` library, the PDF reader library by Paweł Jackowski. This was the candidate to replace `poppler` in TEX Live — this eventually happened with the first commit to the TEX Live repository on 21 April 2020. This means that the next TEX Live 2021 will entirely use `pplib` instead of `poppler`, for both LuaTEX (as was the case in previous years), and now also XETEX; `poppler` is no longer in the TEX Live repository. (By the way, pdfTEX will continue to use its own semi-homegrown `libxpdf` to read PDF files until there is some clear reason to change.)

MetaPost gets related special treatment: the library in LuaTEX includes only the decimal and the floating-point mode, while the `mpost` program also includes the `mpfr` library for arbitrary precision support.

As result of mixing and matching all these variations, building LuaTEX and integrating it into TEX Live is quite a complex task, but thanks to the GNU autotools, things are manageable.

2 The current status of LuaTEX

As noted above, LuaHBTEX (version `1.12.0`) shipped for the first time with the TEX Live 2020 DVD, and it is already supported by LuaLATEX: At the TEX Live 2020 meeting, the talk "HarfBuzz in LuaLATEX" by Marcel Krüger has shown some differences between the HarfBuzz text shaping and the ConTEXt text shaping; also better memory management for large fonts with respect to LuaTEX, especially for 32-bit platforms.

On the other side, Petr Olšák in 2020 has published OpTEX, "...a LuaTeX format based on Plain TeX macros (by Donald Knuth) and on OPmac macros" (see `petr.olsak.net/optex`, and article in this issue), also included in TEX Live 2020. It's not clear if it will eventually support LuaHBTEX.

Finally, also at the TUG 2020 online meeting, Patrick Gundlach in his talk "Speedata Publisher — a different approach to typesetting using Lua" (see `tug.org/TUGboat/41-2/gundlach-speedata.pdf`) has shown an example of a working workflow that uses LuaTEX (and possibly LuaJITTEX) purely by means of the `lua` API — a sort of TEX without `\TeX`. The Speedata Publisher software has been actively developed for a decade.

In light of these continuing developments, it is therefore appropriate to clarify the meaning of "bug fixing" mode, because it is sometimes associated with the term "frozen".

LuaTEX is based on the `lua` release `5.3.5`, and it will stay on the `5.3` version at least for TEX Live 2021 and TEX Live 2022, possibly switching to the final release `5.3.6` (in release candidate 2 at the date of 2020-07-23) at some future point. The current release of `lua` is `5.4.0`, with approximately five years between two versions; it's good practice to have an year of transition between two different versions, so a rough estimation for the next `lua` transition is six years from now, i.e., around TEX Live 2026.

On the side of the TEX core, the plan is for bug fixing and marginal improvements, for example the `\tracinglostchars` ≥ 3 that now raises an error (a new feature added across engines by David Jones), but not new features. From what we have seen previously, stressing LuaTEX in different areas (e.g., only with the `lua` API or with the new HarfBuzz

shaping library) can reveal hidden bugs, but it should be noted that bug fixing is a complex task because the fix must be well harmonized with the rest of the code: For example, some issues with DVI output that need to be checked carefully are still open.

Nevertheless, there are three areas that are still marked as "under active development": the first is the `ffi` (foreign function interface) library, that in LuaTEX is not yet finished and not as functional as its counterpart in LuaJITTEX. Admittedly it is not a key feature of LuaTEX and probably useful only in the context of automated workflows.

The second is the binding with HarfBuzz library, currently given by the `luahrfbuzz` module. If necessary the binding can still be expanded and/or modified, preserving as much as possible the current API, because LuaHBTEX is in an early phase of adoption.

The third area is the `pplib` library that surely needs more testing.

Finally, the bug fixing phase certainly also involves the MetaPost library.

3 The current status of LuaJITTEX

LuaJITTEX is (or in some way is considered) a niche engine. One issue is that while LuaTEX is based on Lua `5.3.5`, LuaJITTEX is still based on `5.1` with some partial coverage of `5.2`. LuaJITTEX also has some intrinsic limits, such as the fixed number of nested tables, which has a serious impact on the table serialization. By design, LuaTEX makes heavy use of C functions bound via the classic Lua/C API; the just-in-time (JIT) compiler doesn't play well in this situation, but this is not a serious issue, given that it can be turned off on demand (and indeed it's off by default). Finally, LuaJIT doesn't support all of Lua's platforms, although the most important ones are available.

On the other hand, the LuaJIT virtual machine is much faster than Lua and the compilation of an article can have a significant speed-up. For this article, LuaJITHBTEX is 2.5 times faster than LuaHBTEX with exactly the same LuaLATEX format; although for complex documents the gain is smaller, around 15%–20%.

The lack of a specific format for LuaJIT does fake the results a bit, but maintaining an additional format in this case is not an easy task: To take advantage of the JIT, where LuaJIT shines, one has to write specialized Lua code and using the `ffi` module requires rather in-depth knowledge of C to achieve significant results. Currently only ConTEXtMkIV has some support for LuaJITTEX.

Probably LuaJITTEX and LuaJITHBTEX are better suited for specialized tasks (e.g. database publishing) or as software as service in cloud, possibly in a containerized environment, but they should also be considered a research tool in digital typesetting.

Currently LuaJIT in TEX Live is still using the `2.1-beta3` release (from 2017), but it is likely it will sync with the official repository by the end of the year. Although LuaJIT development is not proceeding at a rapid pace, there have been important updates (e.g., all LuaJIT 64-bit ports now use 64-bit GC objects by default; and there is support for more platforms). There are some mismatches with Lua (a few functions in Lua that are not available in LuaJIT, notably the `utf8` module) still to be fixed.

4 Conclusion

At the TEX core, LuaTEX and LuaHBTEX are exactly the same and the choice between one or the other depends only on whether or not one accepts HarfBuzz as a dependency. As OpTEX has shown, LuaHBTEX is not always the necessary choice. In any case, the current state is better described by "bug fixing mode with marginal improvements" rather than "frozen", with an emphasis on stability. The area marked as "under active development" may change more significantly, but this should have a minimal impact on stability.

LuaJITTEX and LuaHBTEX are more or less still out of the mainstream and that gives a wider range for maneuvering; given the high efficiency of the implementation of LuaJIT, it's often better to code a module directly in Lua rather than compile and link a C module. Admittedly, it's a rather specialized topic, but efficiency has its costs.

◇ Luigi Scarso
luigi.scarso (at) gmail dot com