With lots of help from Richard Koch

Introduction

Over the 2^5 years of its existence, TEX has been ran on many platforms and has always been noted for its portability, so it's not surprising that when new devices appear, TEX would soon be ported there, too.

Today a new kind of device is in wide use that links computers to telephones and that represents a new challenge on the path of TEX because of reasons both technical and ergonomic, the so-called "smartphones". But it may sound insane to want to use TEX on a Blackberry or a Nokia N97—although word on the street is that Jonathan Kew, author of XeTEX and TEXworks, ported TEX to the iPhone last year—so that's not exactly what I will talk about here.

I will present the achievement of Richard Koch, amongst others author of TEXShop and MacTEX developer, who has successfully compiled and used TEX on Apple's iPad.

The latter does of course not qualify as a smartphone per se, but it shares a lot of features with them, above all the aim at mobility, while having the advantage of giving the user an experience closer to that of an actual computer. The iPad port of TEX, called TEX-8, should therefore give a good idea of what "mobile TEX" could be. It might even be that TEX-8 could be copied to the other Apple mobile devices with only minor changes (who wouldn't want to run TEX on an iPod touch?), but I will stick to describing the experiments I made, thanks to Richard, on the iPad, and hint that they would probably also apply to the other iThingies.

There are a number of issues arising from this task, that indeed could be qualified as Herculean. The way programs work on the iPad is that they're made into "applications" from which you control everything. It is the single entry point to the program, and in fact the only way we can interact with the operating system, because we can't run two applications at the same time. This of course doesn't mean that we can't program many things into an application, but it gives a very different touch: for TEX to be made into an iPad application, we have to embed not only the TEX program itself, the format file and the whole distribution into it, but also the editor itself! If we used a different application to edit the source file, TEX-8 couldn't access it because of Apple's very restrictive policy.

Hence, the interesting issue is that, while all the pieces we need are obviously already available, and the story of writing an iPad application for TEX mostly is the description of how we put the pieces together, it also consists not in small part in crucial design choices aiming at crafting a reasonable user interface.

A word of warning before we proceed to the description proper: I wanted to introduce this project today but it is yet very much a work in progress, and I cannot present anything else than a snapshot of the current development stage. I thus ask the reader to take the following pages with a little bit of salt. All the advertised features, or lack thereof, I review here, will, no doubt, be greatly improved in the future.

So here goes.

A user-friendly interface

A rich text editing environment

The natural entry point to TEX on the iPad seems to be the source code, and TEX-8 thus opens on the editor window, that displays by default a document stored in the aplication about Sylow theorems in group theory. The whole window is a text area, and the iPad virtual keyboard pops up, so that we can type. Typing on the iPad is "not for sissies", in Richard's words, and typing TEX code is rendered even more awkward by the fact that many of TEX's special characters that are ubiquitous (`\''{'}'` amongst several others) are not present on the default keyboard, but demand instead, in order to be typed, that one switch keyboard *twice* (using two different toggle keys), which would make the typing of any serious TEX document extremely painful. In order to remedy that, Richard devised an additional row on top of the standard keyboard, that pops up and disappears together with it. Though I personally regret that the simulated keys don't make the nice keyboard-like sound when tapped, they are immensely helpful and make for a reasonable tradeoff if one needs to type actual input on the iPad.

Another standard iPad feature that could come in handy at that point would be auto-correction of the typed input. It behaves like some kind of aggressive spell-checker, automatically correcting any words the user types, unless the latter directs otherwise. Having personally experienced how this behaviour can have very disruptive effects in some places (when typing URLs, for example), I don't especially recommend to turn it on by default, but it can be useful if one is typing lengthy text in a natural language, and could have spared me to have written "The whoel ndow ia a text qrea, and the ipad keybaoadr popz up, so thqt we cqntpe." when I first typed the above paragraph.

Apart from that feature, the editor is very basic since it borrows from the standard TextEdit application for typing plain text, and has therefore none of the capabilities one may expect from a development enviroment for TEX, apart from a "typeset" button that has the obvious effect. Encoding support is also very poor.

It should also be noted that for the moment, the TEX run freezes everything in the editor and doesn't stop until it completes its task: one can neither use the editor, nor interrupt TEX as it runs.

A convenient file browser

It is yet quite cumbersome to upload files on the iPad. As I have not tried it personally—or rather I did try, but nothing worked—I simply copy the notes by Richard:

My intention is that there will be three ways to proceed:

a. *Load and unload in iTunes, connecting to a regular Mac.*

b. *Mail source and output into the iPad and back out.*

c. ***Other** suitable programs on the iPad can send source to TEX-8, and TEX-8 can send source and output to them. Then these programs can communicate with the outside world.*

But c) isn't working at all, and b) is only working in the sense that you can mail both source and PDF out of TEX-8.

The iTunes method seems to be currently limited by Apple. Here's how it works. When you connect the iPad to iTunes, you'll see a list of programs which can communicate, and TEX-8 will be listed. You will see a list of folders, one for each program. For instance, one folder will be labeled "Mordell". You cannot look inside these folders in iTunes, but you can drag one of the iTunes window to your desktop or elsewhere. If you drag, say Mordell, then you'll get a folder containing the source, the PDF output, the log and sync files, and all the illustrations. So this method is clumsy but works to get stuff out of the iPad.

Unfortunately, Apple's software doesn't allow you to drag folders back in. You can only drag individual files in. I suppose the way to handle that is to zip up a folder and then drag the zip in and have TEX-8 unzip it to a folder the next time it runs. But for now, here's what you can do:

Drag your source and all supplemental files (i.e., illustrations) to TEX-8 in iTunes. Make sure you only drag ***one*** `.tex` *file (no input tex files allowed). The next time TEX-8 starts, it will look for a* `.tex` *file. If one and only one exists, it will create a folder with that name and put all the other* ***files*** *in that folder.*

That's the only current method to get illustrations into the machine.

A feature-full previewer

The preview window allows us to see the PDF file produced by TEX (for the moment, we use PDF as the only output format). We can slide back and forth through the pages, using a standard iPad feature. There is also a button to go to a specific page number, and two other buttons, to jump to the first and the last page. And that's all.

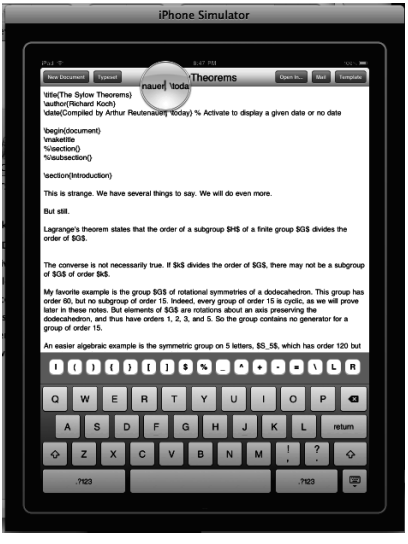

Figure 1 Editing a TEX file on the iPad

No, really. That's all.

A this point of development, there is no search feature and also no magnification possible of the page. And, as has already been hinted, the typesetting process gives absolutely no feedback to the user: TEX doesn't display the output as it does on other systems, and the log file, though stored on disk as usual, is not accessible at all. What's more, if compilation fails but some PDF file was already produced by earlier runs, the user is presented with that file, which is a rather confusing behaviour.

The diligent reader may wonder what "TEX" we really used in all of the above, so let the audince be reassured: we can *of course* use ConTEXt with TEX-8, just as LATEX and plain TEX; for the moment I only experimented with Mark II, though, as I have only been able to use pdfTEX as the underlying engine.

A well-designed programming interface

Now to some technical issues: the main problem one is faced when making TEX into a iPad application is that you simply can't run TEX as a separate process from the application. Apple policy forbids it. You need to make a library out of the TEX program, and to call the `main` function (the entry point to any program written in C) from the application. And TEX is not thought out for that use, which makes that task awkward, even if of course possible. A funny issue is that, for example, in the sources for pdfTEX which Richard took, there are two functions that are called main (in `texk/web2c/lib/main.c` and `texk/web2c/lib/texmfmp.c` in the source hierarchy, respectively); but of course only one is the real entry point to the TEX program. Another much more serious problem is memory management. Because traditionally each TEX run has been autonomous, TEX's memory doesn't need to be managed so meticulously and one can rely on the operating system to clean up TEX's memory upon exit, because TEX calls the C library function that is in fact called `exit` and that takes care of that. When TEX is used as a library, though, calling `exit` shuts down the entire program, therefore killing the editing window, and returning to the iPad "Home Screen", which is a bit ridiculous; and while the solution to this particular problem is obvious and immediate to implement (just remove the call to `exit`), the underlying problem remains: TEX's memory isn't cleaned up and the system loses track of it after the processing, leading to *memory leaks*. These leaks are in fact so important that the limited iPad memory (256 megabytes), while vastly sufficient for any single run of TEX, can't handle the next run and, as of today, *you can't yet run TEX twice in a row*. You need to quit the application and relaunch it. Thankfully, the application reopens in the exact same state as it was wen we left it. This is of course far from optimal, and unlike most of the problems outlined above, it will take work to be solved; but we shouldn't despair.

Other than that, the typesetting speed is reasonable (for the first and only run), which is not that suprising: though limited compared to today's computers, the iPad resources are still immensely more powerful than what was available thirty years ago, when TEX was first developed.

Finally, as has already been said, we used pdfTEX; I have managed to compile LuaTEX as a library as to use for compiling simple documents, but, aside from having one more `main` function (in `texk/web2c/luatexdir/luatex.c`, as it is), the memory management problems are even

worse than with pdfTEX, and when I tried it, the application systematically crashed towards the end of the run.

Conclusion

This is thus the point where we are now. As a conclusion I would like to quote Richard's words:

"[...] in the iPad world, I don't know the right approach. This machine seems like a new category to me—one that will take several years of experiments before we know what works best. It is more a minimalist machine, and I suspect the best programs will be those that throw away features we have come to depend upon to concentrate on a few new paradigms. But I don't know what they will be."

Having been lucky enough to test the application and to use it to typeset a few documents of my own, I fully subscribe to this opinion, of course: the user interface is lacking in many respect at this early stage of development stage, but it will take (programmer) time and (user) experience to improve it and make it really usable.

One thing we certainly shouldn't do, for example, is to port indiscriminately all the capabilities of TEX editors and PDF previewer directly to the iPad application, that would certainly not give a very enjoyable experience.

As a simple example, let us consider the problem of error reporting. While dealing with the problem of a faulty TEX run is straightforward (we just display an error message to the user if the PDF file is not newer than the TEX source), it is not immediately obvious—at least to me—how to display the log file to the user: it should of course be available in some way to people wanting to know what exacly went wrong, but do we need to display the full output of the TEX run, like we have when we run TEX in a terminal window, or from a specialized editor like TEXMaker or TEXworks? I'm not sure it is really useful, nor even desirable, on such a minimalist device as the iPad (not to mention actual smartphones, where it would probably be unreadable). Thus, for the moment, TEX runs in "non-stop mode", as if one had typed '`s`' during an interactive run, and returns even if it produces no PDF (as opposed to waiting for the user's input to complete its task).

And when it comes to the future, I like to think that experimenting with TEX on unusual platforms is a chance to think about new ways to use TEX, and that ConTEXt, that has already been providing complete chaintools in TEX processing for decades, is certainly well prepared for such a challenge!

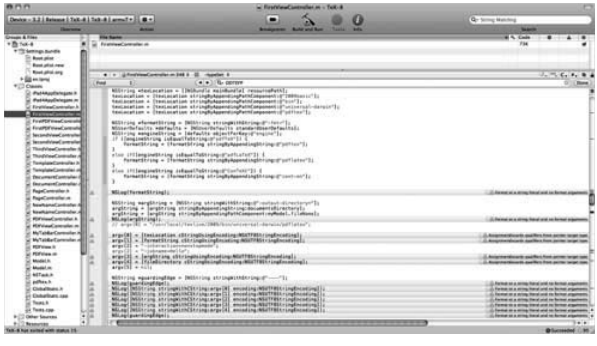

Figure 2 Xcode sources for TEX-8

Figure 3 The TEX-8 application in its natural habitat