TeX as an eBook reader

Kaveh Bazargan

Abstract

An important advantage of eBook readers is their ability to modify text size and page orientation for the most comfortable reading configuration. The eBook reader has to reformat the text on the fly and with minimum delay. Current eBook readers (e.g. Stanza on the iPhone) can do this reformatting, but cannot deal with complicated text such as mathematics. We have been experimenting with using TeX as the formatting engine. Of course it can handle complex mathematics, but it also creates the best line breaks of any eBook reader. We will report our experiments with using TeX as an ebook reader on the iPhone.

1 Problems in the eBook world

1.1 Format wars

There is a 'format war' going on in the world of eBooks. Several formats are proprietary, and several openly specified. This has produced confusion for the publishers, as they don't know which format to distribute their books on, and don't know whether or not they will have to redistribute in the future, in other formats. The XML format was originally envisaged for just this sort of format war. By using a single format which embodies all the logical structure and the content of the document, any other format, e.g. PDF, can be created automatically.

ePub is a comprehensive XML format for eBooks. In particular it supports MathML for mathematical text. So the ideal scenario would be that an eBook is saved in only one format, namely ePub, and an eBook reader would render this on the fly and present a readable view. We'll come back to this after discussing the workflow in our company.

1.2 Ugly output

One of the most compelling features of eBooks is that they can reflow the text to the user's taste. But the line breaking engine on these small devices is not that sophisticated, and the output does not look professionally typeset. For example, eBook readers on the iPhone show lots of bad breaks and large word gaps when the font size is increased. Again, we will come back to this.

2 Evolution of the workflow at River Valley Technologies

The main activity at our company is typesetting mathematical text. To explain most simply, we need to go from a TeX file submitted by the author, to a PDF file. Before the need for generation of XML, this was a straightforward process. We would put the TeX file into style, typeset it, proofread, etc. A few years ago, publishers started requesting XML, and rightly so. As we deal with mathematical material, we needed to use MathML in order to keep the structure for future re-use of XML.

But here is a problem with XML. It is easy to produce a file that parses and validates, but it is not easy to check the content. Checking the PDF, on the other hand is simple — it is called reading the document, or proofreading it. We had to find a way of guaranteeing the fidelity of the XML and PDF. The only solution we could think of was an automated way of generating the PDF from the XML. But this was a non-trivial task. XML deals only with content and logical structure. It does not deal with visual style, placement of figures, hyphenation, etc. There are other anomalies. For instance an XML file might list all figures at the top of the file, but the PDF will have them in the order that they appear.

Here is the workflow we came up with:

$$\text{Author TeX} \to \text{Structured TeX} \to \text{XML} \to \text{Slave TeX} \to \text{PDF}$$

The structured TeX is not for typesetting, but simply uses TeX as markup for tagging elements; for instance `\firstname{John}` or `\journalyear{1985}`. This file is then transformed automatically into the XML. The program we use for this is a highly configured version of TeX4ht, written by the late Eitan Gurari. But for the purposes of the present discussion, it is the stages from the XML onwards that are important.

The XML file is transformed to a TeX file using XSLT. This is what we call a 'slave' TeX file which is not normally looked at. It is simply created in order to produce the final PDF file. The important point is that the translation of the XML to PDF is done with 100% automation. And the intermediate TeX file is not modified in any way, but simply run through TeX to produce the final PDF.

So we have two fully automated processes, one going from a TeX file to an XML file, and one from the XML to a second TeX file. The filters for these two processes were written independently, and checked rigorously. So let us suppose a proofreader is comparing the output from the author file and that from the final TeX file, and checks that the mathematical symbols match. It is then almost certain that the XML also matches. So the method gives us a high degree of confidence in the content of the XML.

3 Back to eBooks

So how does all this relate to eBooks? Well, we said in the section about 'format wars' that ideally ePub should be the one and only format, and it should be rendered on the fly by the eBook reader. So we need a rendering engine that renders ePub in near-real time. Well, our XML → Slave TEX → PDF is just such an engine. So we decided to try to implement this on an eBook reader. The only device which was accessible to us and had a software development kit available was the iPhone from Apple.

So there are two elements to the full process — transforming XML to TEX, and typesetting the TEX. The second seemed the most challenging and we sought the help of Jonathan Kew. Within a short time he managed to port the full TEX program to the iPhone. We decided to created a DVI file rather than a PDF, for speed and for compact file size. Jonathan also wrote a DVI reader for the device. The program runs very quickly indeed, and turning the iPhone on its side instantly reformats the output to the new page aspect ratio.

The transformation from ePub to the slave TEX has not yet been done, but it seems to be the easier part of the problem.

3.1 Quality of output

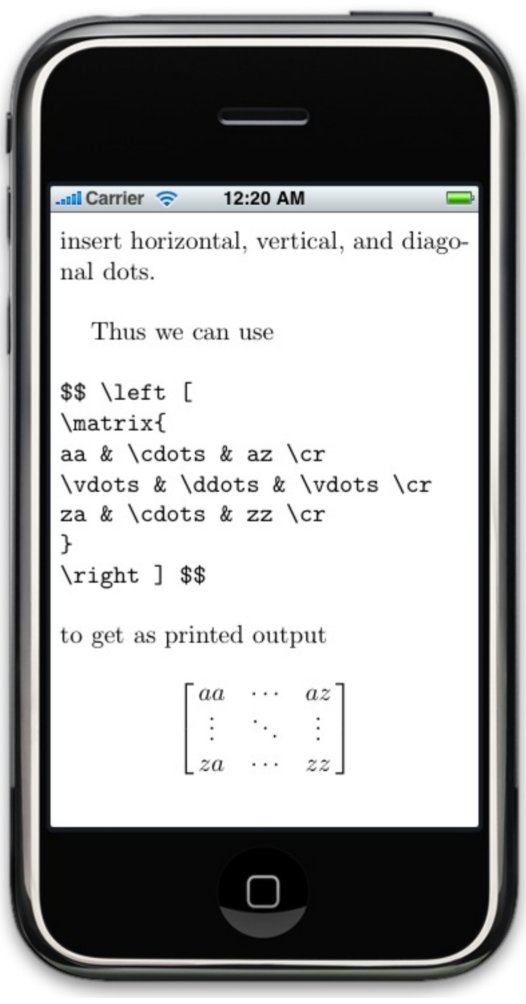
Regarding the 'ugly output' referred to above, we compared our output through TEX for a purely textual file, to the output from the other eBook readers on the iPhone and we think TEX does much better. Of course TEXies should not be surprised by this. We all know that the line breaking algorithm of TEX is the best!

4 Conclusions

Our automated workflow, from XML to PDF, can be modified and applied to eBook readers. The output looks good with good spacing and hyphenation, and only one format, ePub, need be produced by the publisher.

5 Acknowledgements

I am merely the 'Steve Jobs' of our company, others do the clever work and I take to the stage and take the credit! Here the main credit must go to Jonathan Kew who did the work reported here. Credit of course also to Radhakrishnan CV for discussions on the overall concept.



insert horizontal, vertical, and diagonal dots.

Thus we can use

```
$$ \left [
\matrix{
aa & \cdots & az \cr
\vdots & \ddots & \vdots \cr
za & \cdots & zz \cr
}
\right ] $$
```

to get as printed output

$$\left[\matrix{aa & \cdots & az \cr \vdots & \ddots & \vdots \cr za & \cdots & zz \cr}\right]$$

◇ Kaveh Bazargan
River Valley Technologies
kaveh (at) river-valley dot com
http://www.river-valley.com