David Fuchs

David Fuchs is renowned in the TeX community for his work with Donald Knuth in the earliest days of TeX.

[Interview completed 14 May 2007.]

Dave Walden, interviewer: How did you first come in contact with Donald Knuth and then get so deeply involved in TeX?

David Fuchs, interviewee: I came to Stanford as a Ph.D. student in 1978, in the Computer Science Department. Somehow, I got put in a group that was writing a Fortran compiler, which seemed rather retro to me even at the time. When I saw a bulletin-board posting that Prof. Knuth was looking for help on the TeX Project, I jumped at the chance; first, because of the thrill of working with someone famous, and second because I'd be working on software that might have hundreds, or, who knows, even thousands of users.

This was in the days of the old TeX ("TeX78") that was written in the Sail language. I worked on porting the code from the home-grown Waits operating system that Knuth used day-to-day, to the commercial TOPS-20 system that most DEC-20 mainframes used. The code was in five source files: TEXSYN, TEXSEM, TEXHYF, TEXPRE, and TEXOUT (if I remember correctly). TEXSYN handled the syntax of the language, while TEXSEM handled all the semantics. TEXHYF had a hard-coded English language hyphenation algorithm, and TEXPRE existed to populate the hash table with all the built-in control-sequence names. TEXOUT created files suitable for sending to our XGP printer. It was a special experimental printer that Xerox created based on one of their copier engines; only a few were ever made, and they were just about the only raster printers that existed at the time! It was about 200 dpi, though near the edges of the paper it was closer to 180.

Around 1980, we got a Versatec electrostatic printer with an 8-bit parallel interface. I had the job of getting TeX output onto it, since the XGP was getting old, and required almost the full time of a technician (who was also getting old) to keep it running. The XGP had a whole PDP-6 mainframe to control it, but I picked out a little Z80 microcomputer to handle the Versatec. Our mainframes only had serial RS-232 output, and you sure didn't want to be sending whole raster images over a 9600 baud line, so I had to program the Z80 to accept input that mixed character bitmaps to be cached in its small 64K memory along with commands as to where to place each character on the page. There wasn't enough room to build a complete bitmap of the page, and the printer itself simply accepted raster lines from the top of the page to the bottom, so getting it all to work efficiently was a bit of a trick. Also, if you didn't feed the printer the next raster line when it was ready, it would stop the motor that drove the paper, and since it was a wet-toner process (ugh), your output would get a big horizontal streak.

Prof. Knuth suggested that I write a new module like TEXOUT, to be called TEXVER, to produce output suitable for the new printer. I was astonished at what a bad idea that was; every time you wanted to be able to handle a new output device, you'd have to create a new version of TEX with the proper TEXOUT replacement; and users would have to run the right TEX that knew about their particular printer. It was quite a thrill to be able to tell the celebrated Donald Knuth that he was all wet, and that clearly the right way to go was to have TEX create its output in a device-independent format. I think he didn't like the idea of having to run another process after TEX just to get your output (if only because this meant reading and writing to disk twice as much data), but given the pros and cons, gave me the go-ahead. So, I designed the first version of the DVI output format. The goals were just to keep it small and concise and easy to interpret (remember, even our mainframes had a limit of about half a megabyte of core memory per process).

TEX was getting to have a bigger and bigger following, and there were already a number of departments that had installed clones of my Z80-and-electrostatic-printer scheme. Knuth and his students started work on the entirely new TEX (and Metafont), to be written in the (new) Web language on top of Pascal, and with all sorts of new features. I helped with various porting issues, but even more importantly, we had a brand new Alphatype CRS phototypesetter (with a claimed resolution of 5333 dpi) to interface to. Prof. Knuth had picked it out, based on the fact that the manufacturer was willing to divulge how it worked internally, and particularly how it encoded fonts. (In those days, each typesetter manufacturer had its own proprietary font scheme, not to mention its own proprietary typesetting software that sat on their own proprietary computer systems; those days are gone.) Knuth realized that the firmware in the integrated Intel 8008 system that ran the typesetter wouldn't be able to handle his Metafont-generated fonts, so he decided to re-write the entire firmware system for the device. Fortunately, there were cross-assemblers available for our mainframes, and the CRS would accept a new firmware "load" from a serial port.

Please realize that the code that Knuth was rewriting had to simultaneously accelerate, run, and decelerate the two stepper motors that controlled the horizontal and vertical position of the lens that was focusing the character images onto the paper, while simultaneously feeding character outline information to the two sets of two full-size S-100 bus special hardware cards that would extract vertical strokes of the character being imaged to flash onto the CRT in front of the lens (two sets because that would let you pipeline the next character while the current character was being imaged; two cards per set because with kerning, you might have two characters appearing in a single vertical swath). Plus, data would be coming in the serial port asynchronously, and all this was happening in real time. Somehow, he got the thing working with only a little hex debugger that you accessed from the 16-button front panel. And, as we learned the hard way, if you sent too much data up the serial connection too quickly, you'd crash the shared mainframe (since it only expected human typists sitting at computer terminals to be on the other end, so it had never been tested for robustness in this regard), making everyone around the CS department pretty angry.

I, of course, wrote the DVI-to-CRS software, which also required some tricky font caching (the algorithm for which is the basis of my only published paper, which was actually written by my co-author, guess who?). Those were some intense days of working together, getting it all to work. The CRS was in a cramped basement room, and I had to load and unload each sheet of photographic paper in the darkness, and feed it into the triple-bath developer. Any bug could be potentially in my code or Knuth's firmware, and as I mentioned, debugging wasn't easy. After a number of very long days, I came in one morning, and came into Knuth's office just as he was arriving and handing a stack of legal paper to his secretary. "Here's a paper I wrote, please type it in," he told her. I was floored. We'd been working night and day; did he write the paper in his sleep?

DW: When I look at the *TUGboat* list of authors (`http://tug.org/TUGboat/Contents/listauthor.html`), I see that your last "News from the TEX project" report was in 1986. Did you leave the project at that time?

DRF: Prof. Knuth chose to wind down the TEX project in 1986. The idea was that the code and features should be stable (indeed, one of the main features is meant to be that the code and features are stable), and also that he wanted to devote more time to his work on his books and such.

DW: Please tell me a bit about your personal history and life since those early days with Knuth and TEX.

DRF: While I was at Stanford, I'd watched as Andy Bechtolsheim started Sun Microsystems, and Len Bosack and Sandy Lerner started Cisco. (Actually, there are interesting TEX connections to both companies: Unless I mis-remember, Lynn Ruggles worked on creating the first Cisco logo with Metafont. And when Andy needed to create the first high-resolution printed-circuit-board masks for his first Sun workstation, in order to save a few thousand dollars that a commercial "tape-out" would cost him, he wrote a dvi output module to the ecad system he was using, and we actually typeset the artwork onto the Alphatype CRS!)

Anyway, I figured that the start-up business looked fun, and when I got a call from the newly-started Frame Technology, looking for a consultant to put math into their WYSIWYG document system, FrameMaker, one thing led to another, and I became their fifth employee (behind the four founders). Of course, the joke is that I never did get around to putting a math-mode into FrameMaker; I worked on PostScript output, and on Book functions (multiple "Chapter" documents making up a large book, with proper auto-numbering of pages, section/subsections, footnotes, cross-references, table of contents, and index generation).

After nine years, Frame was acquired by Adobe, where I was promoted to "Principal Scientist" in the "Advanced Technology Group", where they put all the old curmudgeons and prima donnas. But I yearned for the start-up life, so one year later, I joined a small Java company called Random Noise, that was eventually bought by Vignette just before the height of the dot-com boom. Actually, there's a funny story there. Random Noise had a very creative product that was just a bit ahead of its time (kind of like DreamWeaver from MacroMedia, but flakier because of early JVMs being way too buggy). But we didn't have actual paying customers, so eventually the day came when the boss told us all to go home. Just 36 hours later, I got a call, saying "Can you come in tomorrow and look busy?" So, we all showed up and continued development, while the deal to sell the technology got struck in the conference room that day.

After that, I got into some high-tech investing, which is fun but frequently frustrating.

DW: Do you still use TEX or one of the TEX-based systems today? (I notice from the TUG web site that you attended the 2004 Practical TEX conference.)

DRF: TEX has always had a special place in my heart. We made the world a better place for scientists and engineers. What other piece of software is still in active use, essentially the same as it was 25 years ago? Some years ago, I got a notion that, given its continued relevance, TEX deserved an internal make-over. The idea is to change the internals to be object oriented, but in a step-wise fashion that guarantees complete upward compatibility. This can open the door to improved enhance-ability while making it even more portable. Just as an example, making everything an object means that all the internal arrays (mem, fmem, trie, etc.) are gone, and so virtually all internal limits disappear (macro definition space, fonts, save stack size for nested macro calls, etc., even number of character widths in a font, number of characters in a font, . . . ; I think the only one I haven't addressed yet is the maximum of 256 of columns per table!) I'm pretty far along with this (and the painstaking process has even found a few TEX bugs that will surprise everyone), but it's been quite time-consuming, and I'm not making any promises yet.

DW: That's fascinating and brings additional questions to mind: Is the work you are describing at the conceptual and design level or have you written code for what you describe; and, if the latter, what programming language are you using, and are you still using some version of web/tangle-and-weave?

DRF: In order to preserve 100 percent compatibility with TEX, the scheme involves making evolutionary, small-step changes to Knuth's original TEX sources. This is accomplished by modifying Tangle (and Weave) to be able to essentially handle a large number of change files, each one of which moves the code a small, but testable, way along toward the ultimate goal. Each step gets checked against the Trip test, plus a few other large TEX and LATEX documents. And, in the unfortunate event that a compatibility bug is found in, say, change number 100 while working on change 300, I can essentially "go back in time" and fix change 100, and then test again all the way through change 300 to make sure to fix any problems that may have cascaded. This way, I end up with code that I have a very high degree of confidence in, and which makes it relatively easy to fix any compatibility bugs that may be found, with minimal turmoil to any unaffected code. (Plus, it's relatively easy to accept a new `TeX.web` with any bug fixes, and run the clock forward over all my changes, to find the places that are affected. It's interesting to think how this scheme is similar to, but different than, a "source code control system".)

By now, I've made over 1700 of these "small steps". And, in fact, I lied, and there aren't 1700 different change files; rather there's one big change file that simply contains one set of changes after another. It's important to realize that the very fact that Tangle/Weave didn't work this way to begin with is that we were dealing with computers that were very address-space limited at the time; there was no way to fit the entire sources to TEX in memory all at once. But now that's no longer the case; in fact, there are myriad design decisions in the TEX code base that were made in consideration of limitations that just no longer apply. Just as a small example: Each character in a font has indices into a table of widths/heights/depths, rather than just knowing its actual width/height/depth directly. Why? Just to save 8 bytes per character per font, that's why. Knuth was forced to make this trade-off to save space, even at the expense of slower operation (finding a character depth involves fetching a word, extracting a few bits of depth index, and then looking the depth up in a table), not to mention artificially limiting the number of character depths within a font to 16. I get to un-do that decision along the way. Toss in all the other such changes, and I've already got a code base with none of the original global arrays from Knuth's code; everything is an object (and all integer indexes have become object references). Voila! All limits are removed (and not simply by making the arrays grow). I hasten to add that all this is done with great sensitivity to speed and efficiency; all the backward-compatibility tests are careful to also keep an eye on the speed and memory footprint of the resulting system.

In order to ease the process along, it's been done in using an object-oriented extension to Pascal (there's a commercial compiler from Borland, as well as an open-source compiler from Free Pascal). However, I've been very careful to use only the most basic aspects of the enhanced language, with the goal of eventually being able to support an automatic translation into C# (easy) and Java (a bit harder) and perhaps even C++ (but I hate it so). But this part hasn't had any direct work done on it (other than keeping it in mind while doing all the changes).

DW: Are you replicating the current version of actual TEX, or are you also thinking about the extensions that have gone into ε-TEX, pdfTEX, etc.?

DRF: I decided to start by modifying plain, vanilla TEX. This is so I can ensure absolute compatibility. Also, because some of the extensions aren't necessary, given the way I removed various limits in TEX. That being said, it's certainly my goal to "catch up" with the missing features from these systems, as appropriate. In fact, that will be a perfect way to put to the test my claim that the "object oriented, simplified" code that I'm creating will in fact be easier to modify and extend.

DW: How do you see TEX in 2007 in the context of QuarkExpress, InDesign, XML, Unicode, the ease of accessing OpenType fonts from various OS apps versus TEX and its TFM files, the most recent math additions to Word, etc. — can TEX still play a significant role that makes it worthwhile to try to recode it to enable more "portability"?

DRF: Well, that's really the big question. Of course, I ask myself this every day. I looked at the new version of Microsoft Word, and found the math stuff to be a bit limited (not to mention that Word documents regularly change formatting from release to release, etc.) I don't expect TEX's niche market to shift to Word or Express or InDesign, but time will tell. In a funny parallel, there's still no replacement for FrameMaker, with its particular strengths for large, structured documents. For both TEX and FrameMaker, we've got users who think that "final, paginated form" is important, which may seem anachronistic to some, but note that Adobe makes billions of dollars a year on PDF, which would have been subsumed by HTML if not for its "just like paper" functionality.

DW: Thank you very much. I've loved hearing your unique perspective of the early history of TEX and what you have being doing since then.