TUG 2005 — program and information

Monday August 22 (tutorials)	8–9 am	*registration*	
	9 am	Hong Feng	*TEX as a compiler*
	10:30 am	*break*	
	10:45 am	Ross Moore	*LATEX to HTML conversion*
	12 pm	*lunch*	
	1:30 pm	Chris Rowley	*LATEX for beginners*
	2:45 pm	Hartmut Henkel	*MetaPost for beginners*
	4 pm	*break*	
	4:15 pm	Hans Hagen	*ConTEXt for beginners*
	5:30–7 pm	*reception*	
Tuesday August 23	9 am	Hong Feng, CTUG	*Welcome*
	9:15 am	Wai Wong, Chinese University of Hong Kong, China	keynote address: *Typesetting Chinese—A personal perspective*
	10:15 am	*break*	
	10:30 am	Jonathan Kew, SIL International	*XeTEX, the multilingual lion: TEX meets Unicode and smart fonts*
	11:15 am	Philip Taylor, University of London	*Typesetting the Byzantine Cappelli*
	11:45 am	Candy Yiu, Portland State University	*Qin's music notation generator*
	12:15 pm	*lunch*	
	1:15 pm	Nelson Beebe, University of Utah	*The design of TEX and METAFONT: A retrospective*
	2:15 pm	Karel Skoupý, ETH Zentrum	*Free-shape text formatting*
	3 pm	*break*	
	3:15 pm	Suki Venkatean, TnQ Books and Journals	*Moving from bytes to words to semantics*
	4 pm	Chris Rowley, Open University	*Beyond TEX: An introduction to new models for high quality document fomatting*
	4:45 pm	panel: CJKV and TEX	moderator: *Hong Feng; Jin-Hwan Cho, Hans Hagen, Jonathan Kew, Chris Rowley, Wai Wong*
Wednesday August 24	9 am	Hong Feng	*Wavelet transformations and Chinese font design*
	9:45 am	Karel Píška, Czech Academy of Sciences	*Converting METAFONT sources to outline fonts using MetaPost*
	10:30 am	*break*	
	10:45 am	Jin-Hwan Cho, University of Suwon	*Practical use of special commands in DVIPDFMx*
	11:30 am	Eitan Gurari, Ohio State University	*Spatial math exercises and worksheets*
	12:15 pm	*lunch*	
	1:15 pm	Klaus Höppner, DANTE e.V.	*Strategies for including graphics in LATEX documents*
	2 pm	Philip Taylor	*Grid typesetting in LATEX*
	2:45 pm	*break*	
	3 pm	Chris Rowley	*LATEX maintenance and development*
	3:45 pm	Ross Moore, Macquarie University	`PlanetMath.org` *and the Free Encyclopaedia of Mathematics*
	4:30 pm	Jerzy Ludwichowski, Nicolaus Copernicus University	*World wide TEX user groups review*
	5 pm	q & a	
	5:30 pm	*TUG annual meeting*	
Thursday August 25	9 am	Steve Grathwohl, Duke Univ. Press	*On ConTEXt*
	9:45 am	Hans Hagen, Pragma ADE & NTG	*XML, a natural companion to TEX*
	10:45 am	*break*	
	11 am	Volker R.W. Schaa, DANTE e.V.	*XML workflows and the EuroTEX 2005 proceedings*
	11:45 am	panel: Digital publishing	moderator: *Hong Feng; Nelson Beebe, Steve Grathwohl, Ross Moore, Volker R.W. Schaa, Philip Taylor*
	12:30 pm	*lunch*	
	1:30 pm	Panorama of Wuhan sightseeing tour	Hong Feng
	5:30 pm	Bus for Wudang departure	

Conference logistics

All conference events (except the banquet) take place at the East Lake Hotel.

(to be written)

TUG annual meeting

After the q & a on Thursday, we will hold the TUG annual meeting. Several TUG board members will be present at the conference: Steve Grathwohl, Klaus Höppner, Ross Moore, and Philip Taylor, as well as TUG's executive director, Robin Laakso. We will report on TUG's current status and future outlook.

More importantly, we invite discussion of any TUG-related business at this time: ideas for outreach to additional communities, ideas for additional initiatives TUG might undertake, existing projects which TUG might support, or anything else.

Spatial math exercises and worksheets

Nandan Bagchee, Eitan Gurari

LaTeX is a highly expressive authoring language considered to be the lingua franca of the mathematics community. Yet, except for a few contributions concerning long division, it offers very little support for expressing spatial forms of elementary mathematic operations.

We will present a highly configurable tool (written in Java) for producing spatial representations of elementary math exercises and worksheets. Current configurations produce verbatim and tabular forms of exercises and worksheets in regular and Nemeth braille formats for inclusion in LaTeX, MathML, HTML, and text files. Our current attention is devoted to the addition, subtraction, multiplication, division, and root operations.

We are interested in identifying potential users from the LaTeX community with the objective of developing widely acceptable LaTeX interfaces for requesting math exercises and worksheets.

Practical use of special commands in dvipdfm$x$

Jin-Hwan Cho

Special commands in TeX provides the only way to communicate arbitrary information with DVI drivers. DVIPDFMx, one of such drivers, translates the standard DVI output of TeX into the PDF format defined by Adobe for platform independent transmission of digital documents.

In this presentation, we discuss all the special commands supported by DVIPDFMx and show some practical applications for package designers as well as TeX end users.

Wavelet transformations and Chinese font design

Hong Feng

Originally, the fonts used for the TeX system were designed with the METAFONT program. In the past two decades, the wavelet transformation has seen wide application, and it can also be applied in the font design for TeX. The METAFONT (or MetaPost) and wavelet transformation can be mutually complementary in Chinese font design.

XeTeX, the Multilingual Lion: TeX meets Unicode and smart fonts

Jonathan Kew

This presentation will focus on XeTeX, a new system that extends TeX with direct support for modern OpenType and AAT fonts and the Unicode character set. This makes it possible to typeset almost any script and language with the same power and flexibility as TeX has traditionally offered in the 8-bit, simple-script world of European languages. Even languages such as Chinese, Arabic, or Indic scripts can be handled without the need for complex macro packages; the text "just works".

As is well known, Professor Donald Knuth's TeX is a typesetting system with a wide user community, and a range of supporting packages and enhancements available for many types of publishing work. However, it dates back to the 1980s and is tightly wedded to 8-bit character data and custom-encoded fonts, making it difficult to configure TeX for many complex-script languages.

One attempt to address this is the Omega project, with its extended versions of TeX font technologies, and the Omega Transformation Processes that can handle complex script behaviors. However, many potential users have found Omega complex and difficult to set up and use, and it appears to have found rather limited acceptance.

XeTeX (currently available on Mac OS X, but there is interest in porting to other platforms as well) integrates the TeX formatting engine with technologies from both the host operating system (Apple Type Services, Text Encoding Converter) and auxiliary libraries (ICU, TECkit). Thus, it provides a system that combines the power, flexibility, and typographic excellence of TeX with modern international standards for character encoding and font rendering.

Because XeTeX is integrated with the host operating system's font support, no complex configuration is required; any Unicode-compliant font installed on the user's computer is immediately available for typesetting. A wide range of fonts thus become available for use in TeX, and can be freely used within established macro packages such as LaTeX or ConTeXt.

LaTeX maintenance and development

Chris Rowley

This talk will give a brief history of the LaTeX Project, giving some insights into what is involved in the enhancement and maintenance of a robust and widely used software system for the automated formatting of complex documents.

Free-shape text formatting

Karel Skoupý

TeX's line-breaking algorithm needs to know the widths of all the resulting lines in advance. That limits its applicability to free-shape layouts because the line width may depend on the vertical position of the line which in turn may depend on a future page break and cannot be always known in advance.

We will present a generalised version of the line breaking algorithm which works inside a general shape and allows vertical stretching of the formatted text.

However, this generalised algorithm assumes interdependency of the line and page breaking and therefore cannot be easily integrated into TeX formatting model. We will discuss the necessary generalisations of TeX document and formatting model which would make reliable free-shape text formatting possible.

Typesetting the Byzantine *Capelli*

Philip Taylor

A small group of very gifted scholars, led by Miss Julian Chrysostomides with enormous assistance from Dr Charalambos Dendrinos, have spent much of the last five years researching and preparing the *Lexicon of Abbreviations & Ligatures in Greek Minuscule Hands*. I have been involved with this project virtually since its inception, and will discuss some of the technical challenges which arose, with particular reference to the challenge of sorting TeX markup for polytonic Greek using multiple concurrent sort keys.

Grid-based typesetting in LaTeX

Philip Taylor

The first edition of Rosalind Gibson's *Principles of Nutritional Assessment* was jointly typeset by her husband Ian and myself in the years preceding its publication in 1990; the preparation of this edition was the subject of one of my very first talks at a TUG meeting. Now, fifteen years later, Ian and I have again collaborated in the typesetting of the second edition, which—unlike the first—is typeset in two columns on a strict grid. LaTeX is not easily coerced into grid-based typesetting, so the main thread of this talk will be the various measures we used to achieve the desired effect.

Moving from bytes to words to semantics

S.K. Venkatean

Starting from several bytes of ASCII or Unicode strings one can construct a typeset output readable by the community that understands that script. Unfortunately, it still remains unreadable by large community of people who don't understand the scipt. Instead, if this had been coded at the level of a semantic-word, with each word standing for unique-semantic-identity, with sufficient markers (the curly bracket nesting being one such example) for grammar and flow, then it would be able display itself in each language without ambiguity. The eccentricities of ligatures, capitalization, joining of letters could then be handled accurately. The hyphenation, for example, could then be not pattern-based but semantic-word-based (hyphenation in English, for example, can be dependent on whether the word is a noun or a verb). In this work we discuss on the possible atomic words (atoms of course have their own protons, electrons, ...) of a language and semantic-markups that could lead us to such a dream.

The design of TeX and METAFONT: A retrospective

Nelson H. F. Beebe
University of Utah
Department of Mathematics, 110 LCB
155 S 1400 E RM 233
Salt Lake City, UT 84112-0090
USA
WWW URL: http://www.math.utah.edu/~beebe
Telephone: +1 801 581 5254
FAX: +1 801 581 4148
Internet: beebe@math.utah.edu, beebe@acm.org, beebe@computer.org

Abstract

This article looks back at the design of TeX and METAFONT, and analyzes how they were affected by architectures, operating systems, programming languages, and resource limits of the computing world at the time of their creation by a remarkable programmer and human being, Donald E. Knuth. This paper is dedicated to him, with deep gratitude for the continued inspiration and learning that I've received from his software, his scientific writing, and our occasional personal encounters over the last 25+ years.

— — ∗ — —

1 Introduction

More than a quarter century has elapsed since Donald Knuth took his sabbatical year of 1977–78 from Stanford University to tackle the problem of improving the quality of computer-based typesetting of his famous book series, *The Art of Computer Programming* [53–58, 60, 65–67].

When the first volume appeared in 1968, most typesetting was still done by the hot-lead process, and expert human typographers with decades of experience handled line breaking, page breaking, and page layout. By the mid 1970s, proprietary computer-based typesetting systems had entered the market, and in the view of Donald Knuth, had seriously degraded quality. When the first page proofs of part of the second edition of Volume 2 arrived, he was so disappointed that he wrote [68, p. 5]:

> I didn't know what to do. I had spent 15 years writing those books, but if they were going to look awful I didn't want to write any more. How could I be proud of such a product?

A few months later, he learned of some new devices that used digital techniques to create letter images, and the close connection to the 0s and 1s of computer science led him to think about how he himself might design systems to place characters on a page, and draw the individual characters as a matrix of black and white dots. The sabbatical-year project produced working prototypes of two software programs for that purpose that were described in the book *TeX and METAFONT: New Directions in Typesetting* [59].

The rest is of course history [6] ... the digital typesetting project lasted about a decade, produced several more books [64, 68–73], Ph.D. degrees for Frank Liang [79, 80], John Hobby [36], Michael Plass [88], Lynn Ruggles [92], and Ignacio Zabala Salelles [110], and had spinoffs in the commercial document-formatting industry and in the first laser printers. TeX, and the LaTeX system built on top of it [20–22, 76, 77, 83], became the standard markup and typesetting system in the computer science, mathematics, and physics communities, and have been widely used in many other fields.

The purpose of this article is to look back at TeX and METAFONT and examine how they were shaped by the attitudes and computing environment of the time.

2 Computers and people

Now that computers are widely available throughout much of the developed world, and when embedded systems are counted, are more numerous than humans, it is probably difficult for younger people to imagine a world without computers readily at hand. Yet not so long ago, this was not the case.

Until the desktop computers of the 1980s, a 'computer' usually meant a large expensive box, at least as long as an automobile, residing in a climate-controlled machine room with raised flooring, and fed electricity by power cables as thick as your wrist. At many universities, these systems had their own buildings, or at least entire building floors, called Computer Centers. The hardware usually cost hundreds of thousands to millions of dollars (where according to the US Consumer Price Index, a million dollars in 1968 is roughly the same as five million in 2000), and required a full-time professional staff of managers, systems programmers, and operators.

At most computer installations, the costs were passed on to users in the form of charges, such as the US$1500 per hour for CPU time and US$0.50 to open a file that I suffered with as a graduate student earning US$1.50 per hour. At my site, there weren't any disk-storage charges, because it was forbidden to store files on disk: they had to reside either on punched cards, or on reels of magnetic tape. A couple of years ago, I came across a bill from the early 1980s for a 200MB disk: the device was the size of a washing machine, and cost US$15 000. Today, that amount of storage is about fifty thousand times cheaper, and disk-storage costs are likely to continue to drop.

I have cited these costs to show that, until desktop computers became widespread, it was people who worked for computers, not the reverse. When a two-hour run cost as much as your year's salary, you had to spend a lot of time thinking about your programs, instead of just running them to see if they worked.

When I came to Utah in 1978, the College of Science that I joined had just purchased a DECSYSTEM 20, a medium-sized timesharing computer based on the DEC PDP-10 processor, and the Department of Computer Science bought one too on the same order. Ours ultimately cost about $750 000, and supplied many of the computing needs of the College of Science for more than a dozen years, often supporting 50–100 interactive login sessions. Its total physical memory was just over three megabytes, but we called it three quarters of a megaword. We started in 1978 with 400MB of disk storage, and ended in 1990 with 1.8GB for the entire College. Although computer time was still a chargeable item, we managed to recover costs by getting each Department to contribute a yearly portion of the expenses as a flat fee. The operating system's class scheduler guaranteed departmental users a share of the machine in proportion to their fraction of the budget. Thus, most individual users didn't worry about computer charges.

3 The DEC PDP-10

The PDP-10, first released in 1967, ran at least ten or eleven different operating systems:

- BBN TENEX,
- Compuserve modified 4S72,
- DEC TOPS-10 (sometimes humorously called BOTTOMS-10 by TOPS-20 users), and just called the MONITOR before it was trademarked,
- DEC TOPS-20 (a modified TENEX affectionately called TWENEX by some users),
- MIT ITS (Incompatible Timesharing System),
- Carnegie-Mellon University (CMU) modified TOPS-10,
- On-Line Systems' OLS-10,
- Stanford WAITS (Westcoast Alternative to ITS),
- Tymshare AUGUST (a modified TENEX),
- Tymshare TYMCOM-X, and on the smaller DECSYSTEM 20/20 model, TYMCOM-XX.

Although the operating systems differed, it was usually possible to move source-code programs among them with few if any changes, and some binaries compiled on TOPS-10 in 1975 still run just fine on TOPS-20 three decades later (see Section 3).

Our machines at Utah both used TOPS-20, but Donald Knuth's work on TeX and METAFONT was done on WAITS. That system was a research operating system, with frequent changes that resulted in bugs, causing many crashes and much downtime. Don told me earlier this year that the O/S was aptly named, since he wrote much of the draft of *The TeXbook* while he was waiting in the Computer Center for WAITS to come back up. By contrast, apart from hardware-maintenance sessions in a four-hour block each week, the Utah TOPS-20 systems were rarely down.

For about a decade, PDP-10 computers formed the backbone of the Arpanet, which began with just five nodes, at the University of California campuses at Berkeley, Los Angeles, and Santa Barbara, plus SRI (Stanford Research Institute) and Utah, and later evolved into the world-wide Internet [24, p. 48]. PDP-10 machines were adopted by major computer-science departments, and hosted or contributed to many important developments, including at least these:

- Bob Metcalf's *Ethernet* [Xerox PARC, Intel, and DEC];
- Vinton Cerf's and Robert Kahn's invention of the *Transmission Control Protocol* and the *Internet Protocol* (TCP/IP);
- the MACSYMA [MIT], REDUCE [Utah] and MAPLE [Waterloo] symbolic-algebra languages;
- several dialects of LISP, including MACLISP [MIT] and PSL (Portable Standard Lisp) [Utah];
- the systems-programming language BLISS [DEC and CMU];
- the shell-scripting and systems-programming language PCL (Programmable Command Language) [DEC, CMU, and FUNDP] [94];
- Dan Swinehart's and Bob Sproull's SAIL (Stanford Artificial Intelligence Language) Algol-family programming language in which TEX and METAFONT were first implemented;
- an excellent compiler for PASCAL [Hamburg/Rutgers/Sandia], the language in which TEX and METAFONT were next implemented;
- Larry Tesler's PUB document formatting system [101] [PUB was written in SAIL, and had a macro language based on a SAIL subset];
- Brian Reid's document-formatting and bibliographic system, SCRIBE [89, 90] [CMU], that heavily influenced the design of LATEX and BIBTEX [although SAIL co-architect Bob Sproull was Brian's thesis advisor, Brian wrote SCRIBE in the locally-developed BLISS language];
- Richard Stallman's extensible and customizable text editor, emacs [MIT];
- Jay Lepreau's port, pcc20 [Utah], of Steve Johnson's *Portable C Compiler*, pcc [Bell Labs];
- Kok Chen's and Ken Harrenstien's kcc20 native C compiler [SRI];
- Ralph Gorin's spell, one of the first sophisticated interactive spelling checkers [Stanford];
- Mark Crispin's mail client, mm, still one of the best around [Stanford];
- Will Crowther's adventure, Don Daglow's baseball and dungeon, Walter Bright's empire, and University of Utah student Nolan Bushnell's pong, all developed on PDP-10s, were some of the earliest computer games [Bushnell went on to found Atari, Inc., and computer games are now a multi-billion-dollar world-wide business driving the computer-chip industry to ever-higher performance];
- part of the 1982 DISNEY science-fiction film *TRON* was rendered on a PDP-10 clone [curiously, that architecture has a TRON instruction (Test Right-halfword Ones and skip if Not masked) with the numeric operation code 666, leading some to suggest a connection with the name of the film, or the significance of that number in the occult];
- Frank da Cruz's transport- and platform-independent interactive and scriptable communications software kermit [Columbia];
- Gary Kildall's [105] CP/M, the first commercial operating system for the Intel 8080, was developed using Intel's 8080 simulator on the PDP-10 at the Naval Postgraduate School in Monterey, California;
- Harvard University student Paul Allen's Intel 8080 simulator on the PDP-10 was used by fellow student Bill Gates to develop a BASIC-language interpreter before Intel hardware was available to them. [Both had worked on PDP-10 systems in Seattle and Portland in the late 1960s and early 1970s while they were still in school. They later founded Microsoft Corporation, and borrowed ideas from a subset of Kildall's CP/M for their MS-DOS. While IBM initially planned to offer both systems on its personal computer that was introduced in August 1981, pricing differences soon led to its dropping CP/M.]

Notably absent from this list is the Bell Laboratories project that led to the creation of the UNIX operating system: they wanted to buy or lease a PDP-10, but couldn't get the funding [93, Chapter 5].

The PDP-10 and its operating systems is mentioned in about 170 of the now nearly 4000 *Request for Comments* (RFC) documents that informally define the protocols and behavior of the Internet.

The PDP-10 had compilers for ALGOL 60, BASIC, BLISS, C, COBOL 74, FORTH, FORTRAN 66, FORTRAN 77, LISP, PASCAL, SAIL, SIMULA 67, and SNOBOL, plus three assemblers called MACRO, MIDAS, and FAIL (fast one-pass assembler). A lot of programming was done in assembly code, including that for most of the operating systems. Indeed, the abstract of the FAIL manual [108] notes:

> Although FAIL uses substantially more main memory than MACRO-10, it assembles typical programs about five times faster. FAIL assembles the entire Stanford time-sharing operating system (two million characters) in less than four minutes of CPU time on a KA-10 processor.

The KA-10 was one of the early PDP-10 models, so such performance was quite impressive. The high-level BLISS language [9, 10, 109] might have been preferred for such work, but it was comparatively expensive to license, and few sites had it. Anyway, Ralph Gorin's book on assembly language and systems programming [23] provided an outstanding resource for programmers.

Given the complexity of most assembly languages, it is instructive to look at the short example in Figure 1 that helps to illustrate why the PDP-10 assembly language was so popular among its users.

```
MOVE 4, B                      ; load B into register 4
CAML 4, FOO                    ; IF (b >= foo) THEN
 PUSHJ P, [                    ;   BEGIN
      HRROI A, [ASCIZ/.LT./]  ;     message = ".LT.";
      SETOM LESS               ;     less = -1;
      AOS (P)                  ;   END (skip around ELSE)
      POPJ P, ]                ; ELSE
 PUSHJ P, [                    ;   BEGIN
      HRROI A, [ASCIZ/.GE./]  ;     message = ".GE.";
      SETZM LESS               ;     less = 0;
      POPJ P, ]                ;   END;
PSOUT                          ; PRINT message;
```

Figure 1: MACRO-10 assembly language for the PDP-10 and its high-level pseudo-language equivalent, adapted from [15].

You can understand the assembly code once you know the instruction mnemonics: CAML (Compare Accumulator with Memory and skip if Low) handles the conditional, HRROI (Half word Right to Right, Ones, Immediate) constructs a 7-bit byte pointer in an 18-bit address space, SETOM (Set to Ones Memory) stores a negative integer one, SETZM (Set to Zeros Memory) stores a zero, AOS (Add One to Self) increments the stack pointer (P), PUSHJ and POPJ handle stack-based call and return, and PSOUT is a system call to print a string. Brackets delimit remote code and data blocks.

The prevalence of instructions that manipulate 18-bit addresses makes it hard to generalize assembly code for 30-bit extended addressing, but tricks with 18-bit memory segments alleviated this somewhat.

Document formatting was provided by runoff, which shared a common ancestor roff with UNIX troff, and by PUB. Later, SCRIBE became commercially available, but required an annual license fee, and ran only on the PDP-10, so it too had limited availability, and I refused to use it for that reason.

The PDP-10 had 36-bit words, with five seven-bit ASCII characters stored in each word. This left the low-order (rightmost) bit unused. It was normally zero, but when set to one, indicated that the preceding five characters were a line number that some editors used, and compilers could report in diagnostics.

Although seven-bit ASCII was the usual PDP-10 text representation, the hardware instruction set had general byte-pointer instructions that could reference bytes of any size from 1 to 36 bits, and the kcc20 compiler provided easy access to them in C. For interfacing with 32-bit UNIX and VMS systems, 8-bit bytes were used, with four bits wasted at the low end of each word.

The PDP-10 filesystems recorded the byte count and byte size for every file, so in principle, text-processing software at least could have handled both 7-bit and 8-bit byte sizes. Indeed, Mark Crispin proposed that Unicode could be nicely handled in 9-bit UTF-9 and 18-bit UTF-18 encodings [13]. Alas, most PDP-10 systems were retired before this generality could be widely implemented.

One convenient feature of the PDP-10 operating systems was the ability to define *directory search paths* as values of *logical names*. For example, in TOPS-20, the command

```
@define TEXINPUTS: TEXINPUTS:,
                   ps:<jones.tex.inputs>
```

would add a user's personal subdirectory to the end of the system-wide definition of the search path. The @ character was the normal prompt from the EXEC command interpreter. A subsequent reference to texinputs:myfile.tex was all that it took to locate the file in the search path.

Since the directory search was handled inside the operating system, it was trivially available to all programs, no matter what language they were written in, unlike other operating systems where such searching has to be implemented by each program that requires it. In this respect, and many others, to paraphrase ACM Turing Award laureate Tony Hoare's famous remark about ALGOL 60 [31], TOPS-20 "was so far ahead of its time that it was not only an improvement on its predecessors, but also on nearly all its successors."

In addition, a manager could readily change the system-wide definition by a single privileged command:

```
$^Edefine TEXINPUTS: ps:<tex.inputs>,
                     ps:<tex.new>
```

The new definition was immediately available to all users, including those who had included the name

TEXINPUTS: in their own search paths. The $ was the EXEC prompt when a suitably-privileged user had enabled management capabilities.

The great convenience of this facility encouraged those who ported TeX and METAFONT to provide something similar. Today, users of the *TeX Live* distributions are familiar with the kpathsea library, which provides an even more powerful, and customizable, mechanism for path searching.

The original PDP-10 instruction set had an 18-bit address field, giving a memory space of $2^{18} =$ 262 144 words, or about 1.25MB. Later designs extended the address space to 30 bits (5GB), but only 23 were ever implemented in DEC hardware, giving a practical limit of 40MB. That was still much more than most customers could afford in 1983 when the PDP-10 product line was terminated, and VAX VMS became the DEC flagship architecture and operating system.

The next generation of the PDP-10 was announced to be about ten to fifteen times faster than existing models, but early in 1983, rumors of trouble at DEC had reached the PDP-10 user community. At the Fall 1983 DECUS (DEC User Society) Symposium in Las Vegas, Nevada, that I attended, several PDP-10 devotees sported T-shirts emblazoned with

I don't care what they say,
36 bits are here to stay!

They were not entirely wrong, as we shall see.

DEC had products based on the KA-10, KI-10, and KL-10 versions of the PDP-10 processor. Later, other companies produced competing systems that ran one or more of the existing operating systems: Foonly (F1, F2, and F3), Systems Concepts (SC-40), Xerox PARC (MAXC) [16], and XKL Systems Corporation (TOAD-1 for *Ten On A Desk*). Some of these implemented up to 30 address bits (1GW, or 4.5GB). XKL even made a major porting effort of GNU and UNIX utilities, and got the X11 WINDOW SYSTEM running. Ultimately, none enjoyed continued commercial success.

The PDP-10 lives on among hobbyists, thanks to Ken Harrenstien's superb KLH10 simulator [30] with 23-bit addressing, and the vendor's generosity in providing the operating system, compilers, documentation, and utilities for noncommercial use. On a fast modern desktop workstation, TOPS-20 runs several times faster than the original hardware ever did. It has been fun revisiting this environment that was such a leap forward from its predecessors, and I now generally have a TOPS-20 window or two open on my UNIX workstation. I even carried this virtual PDP-10 in a laptop to the *Practical TeX 2005* conference, and it fits nicely in a memory stick the size of a pocket knife.

4 Resource limits

The limited memory of the PDP-10 forced many economizations in the design of TeX and METAFONT. In order to facilitate possible reimplementation in other languages, all memory management is handled by the programs themselves, and sizes of internal tables are fixed at compile time. Table 1 shows the sizes of those tables, then and now. To further economize, many data structures were stored compactly with redundant information elided. For example, while TeX fonts could have up to 128 characters (later increased to 256), there are only 16 different widths and heights allowed, and one of those 16 is required to be zero. Also, although hundreds of text fonts are allowed, only 16 mathematical families are supported. Ulrik Vieth has provided a good summary of this topic [103].

Table 1: TeX table sizes on TOPS-20 in 1984 and in *TeX Live* on UNIX in 2004, as reported in the trip test.

Table	1984	2004	Growth
strings	1819	98002	53.9
string characters	9287	1221682	131.5
memory words	3001	1500022	499.8
control sequences	2100	60000	28.6
font info words	20000	1000000	50.0
fonts	75	2000	26.7
hyphen. exceptions	307	1000	3.3
stack positions (i)	200	5000	25.0
stack positions (n)	40	500	12.5
stack positions (p)	60	6000	100.0
stack positions (b)	500	200000	400.0
stack positions (s)	600	40000	66.7

Instead of supporting scores of accented characters, TeX expected to compose them dynamically from an accent positioned on a base letter. That in turn meant that words with accented letters could not be hyphenated automatically, an intolerable situation for many European languages. That restriction was finally removed in late 1989 [63] with the release of TeX version 3.0 and METAFONT version 2.0, when those programs were extended to fully support 8-bit characters, and provide up to 256 hyphenation tables to handle multilingual documents. Examination of source-code difference listings shows that about 7% of TeX was changed in this essential upgrade.

The TeX DVI and METAFONT GF and TFM files were designed to be compact binary files that require special software tools to process. Recall from p. 502 that disk storage cost around US$100 per MB, so file compactness mattered! In contrast, in UNIX troff, the corresponding files are generally simple, albeit compact and cryptic, text files to facilitate use of filters in data-processing pipelines. Indeed, the UNIX approach of small-is-beautiful encouraged the use of separate tools for typesetting mathematics [43], pictures [41], and tables [39], each filtering the troff input stream, instead of the monolithic approach that TEX uses.

In any computer program, when things go awry, before the problem can be fixed, it is essential to know *where* the failure occurred. The same applies when a change in program behavior is called for: you first have to *find* the code that must be modified.

In either case, to better understand what is happening, it is very helpful to have a traceback of the routine calls that led to the failure or point of change, and a report of the source-code location where every step in the call history is defined. Unfortunately, PDP-10 memory limitations prevented TEX and METAFONT from recording the provenance of every built-in operator and run-time macro, yet every programmer who has written code for these systems has often asked: *where* is that macro defined, and *why* is it behaving that way? Although both programs offer several levels of execution tracing, the output trace is often voluminous and opaque, and no macro-level debugger exists for either program.

The need for a record of source-code provenance is particularly felt in the LATEX world, where it is common for documents to depend on dozens of complex macro packages collectively containing many tens of thousands of lines of code, and sometimes redefining macros that other loaded packages expect to redefine differently for their own purposes. During the course of writing this article, I discovered, tracked down, and fixed three errors in the underlying LATEX style files for the TEX User Group conference proceedings. Each time, the repairs took much longer than should have been necessary, because I could not find the faulty code quickly.

Finally, error diagnostics and error recovery reflect past technology and resource limits. Robin Fairbairns remarked in a May 2005 TEXhax list posting:

> Any TEX-based errors are pretty ghastly. This is characteristic of the age in which it was developed, and of the fiendishly feeble machines we had to play with back then. But they're a lot better than the first ALGOL 68 compiler I played with, which had a single syntax diagnostic "*not a program!*"

5 Choosing a programming language

When Donald Knuth began to think about the problem of designing and implementing a typesetting system and a companion font-creation system, he was faced with the need to select a programming language for the task. We have already summarized what was available on the PDP-10.

COBOL was too horrid to contemplate: imagine writing code in a language with hundreds of reserved words, and such verbose syntax that a simple arithmetic operation and assignment `c = a*b` becomes

```
MULTIPLY A BY B GIVING C.
```

More complex expressions require every subexpression to be given a name and assigned to.

FORTRAN 66 was the only language with any hope of portability to many other systems. However, its omission of recursion, absence of data structures beyond arrays, lack of memory management, deficient control structures, record-oriented I/O, primitive Hollerith strings (`12HHELLO, WORLD`) that could be used only in `DATA` and `FORMAT` statements and as routine arguments, and its restriction to six-character variable names, made it distinctly unsuitable. Nevertheless, METAFONT was later translated to FORTRAN elsewhere for a port to Harris computers [85].

PASCAL only became available on the PDP-10 in late 1978, more than a year after Don began his sabbatical year. We shall return to it in Section 6.

BLISS was an expensive commercial product that was available only on DEC PDP-10, PDP-11, and later, VAX, computers. Although DEC subsequently defined COMMON BLISS to be used across those very different 16-bit, 32-bit, and 36-bit systems, in practice, BLISS exposed too much of the underlying architecture, and the compilers were neither portable [9, 10] nor freely available. Brian Reid commented [90, p. 106]:

> BLISS proved to be an extremely difficult language in which to get started on such a project [SCRIBE], since it has utterly no low-level support for any data types besides scalar words and stack-allocated vectors.
>
> I began an implementation on the PDP-10 in September 1976, spending the first six months building a programming environment in which the rest of the development could take place. This programming environment included runtime and diagnostic sup-

port for strings, lists, and heap-allocated vectors, as well as an operating-system interface intended to be portable across machines.

Inside DEC, later absorbed by Compaq and then by Hewlett-Packard, BLISS was ported to 32-bit and 64-bit ALPHA in the early 1990s, to Intel IA-32 in 1995, and recently, to IA-64 [10], but has remained largely unknown and unused outside those corporate environments.

LISP would have been attractive and powerful, and in retrospect, would have made TEX and METAFONT far more extensible than they are, because any part of them could have been rewritten in LISP, and they would not have needed to have macro languages at all! Unfortunately, until the advent of COMMON LISP in 1984 [96, 97], and for some time after, the LISP world suffered from having about as many dialects as there were LISP programmers, making it impossible to select a language flavor that worked everywhere.

The only viable approach would have been to write a LISP compiler or interpreter, bringing one back to the original problem of picking a language to write *that* in. The one point in favor of this approach is that LISP is syntactically the simplest of all programming languages, so workable interpreters could be done in a few hundred lines, instead of the 10K to 100K lines that were needed for languages like PASCAL and FORTRAN. However, we have to remember that computer use cost a lot of money, and comparatively few people outside computer-science departments had the luxury of ignoring the substantial run-time costs of interpreted languages. A typesetting system is expected to receive heavy use, and efficiency and fast turnaround are essential.

PDP-10 assembly language had been used for many other programming projects, including the operating systems and the three assemblers themselves. However, Don had worked on several different machines since 1959, and he knew that all computers eventually get replaced, often by new ones with radically-different instruction sets, operating systems, and programming languages. Thus, this avenue was not attractive either, since he had to be able to use his typesetting program for all of his future writing.

There was only one viable choice left, and that was SAIL. That language was developed at Stanford, and that is probably one of the reasons why Don chose it over SIMULA 67, its Norwegian cousin, despite his own Norwegian heritage; both languages are descendents of ALGOL 60. SIMULA 67 did however strongly influence Bjarne Stroustrup's design of C++ [98, Chapter 1]. Although SAIL had an offspring, MAINSAIL (Machine Independent SAIL), that might have been more attractive, that language was not born until 1979, two years after the sabbatical-year project. Figure 2 shows a small sample of SAIL, taken from the METAFONT source file `mfntrp.sai`. A detailed description of the language can be found in the first good book on computer graphics [86, Appendix IV], co-written by one of SAIL's architects.

```
internal saf string array fname[0:2]
# file name, extension, and directory;

internal simp procedure scanfilename
# sets up fname[0:2];
begin integer j,c;
fname[0]_fname[1]_fname[2]_null;
j_0;
while curbuf and chartype[curbuf]=space
    do c_lop(curbuf);
loop    begin c_chartype[curbuf];
        case c of begin
        [pnt] j_1;
        [lbrack] j_2;
        [comma][wxy][rbrack][digit][letter];
        else done
          end;
        fname[j]_fname[j]&lop(curbuf);
        end;
end;
```

Figure 2: Filename scanning in SAIL, formatted as originally written by Donald Knuth, except for the movement of comments to separate lines. The square-bracketed names are symbolic integer constants declared earlier in the program.

The underscore operator in SAIL source-code assignments printed as a left arrow in the Stanford variant of ASCII, but PDP-10 sites elsewhere just saw it as a plain underscore. However, its use as the assignment operator meant that it could not be used as an extended letter to make compound names more readable, as is now common in many other programming languages.

The left arrow in the Stanford variant of ASCII was not the only unusual character. Table 2 shows graphics assigned to the normally glyphless control characters. The existence of seven Greek letters in the control-character region may explain why TEX's default text-font layout packs Greek letters into the first ten slots.

Besides being a high-level language with good control and data structures, and recursion, SAIL

Table 2: The Stanford extended ASCII character set, with table positions in octal. This table from RFC 698 [84] disagrees in a few slots with a similar table in the first book about TEX [59, p. 169]. CMU, MIT, and the University of Southern California also had their own incompatible modified versions of ASCII.

Although ASCII was first standardized in 1963, got lowercase letters in 1965, and reached its current form in 1967, the character set Babel has lasted far too long, with hundreds of variants of 7-bit and 8-bit sets still in use around the world. See Mackenzie's book [81] for a comprehensive history up to 1980, and the Unicode Standard [102] for what the future may look like.

000	·	001	↓	002	α	003	β
004	∧	005	¬	006	ϵ	007	π
010	λ	011	γ	012	δ	013	∫
014	±	015	⊕	016	∞	017	∇
020	⊂	021	⊃	022	∩	023	∪
024	∀	025	∃	026	⊗	027	↔
030	_	031	→	032	~	033	≠
034	≤	035	≥	036	≡	037	∨
		040–135 as in standard ASCII					
				136	↑	137	←
		140–174 as in standard ASCII					
		175	◊	176	}	177	^

```
@type hello.pas
program hello(output);
begin
   writeln('Hello, world')
end.

@load hello
PASCAL: HELLO
LINK:   Loading

@ddt
DDT
hello$b   hello+62$b   $g
$1B>>HELLO/   TDZA 0   $x
   0/   0   0/   0
<SKIP>
HELLO+2/   MOVEM %CCLSW   $x
   0/   0   %CCLSW/   0
HELLO+3/   MOVE %CCLDN   $x
   0/   0   %CCLDN/   0
HELLO+4/   JUMPN HELLO+11   $x
   0/   0   HELLO+11
HELLO+5/   MOVEM 1,%RNNAM   $p

OUTPUT     : tty:
$2B>>HELLO+62/   JRST .MAIN.   $$x
Hello, world
```

Figure 3: Debugging a PASCAL program with ddt. The at signs are the default TOPS-20 command prompt. The dollar signs are the echo of ASCII ESCAPE characters. Breakpoints ($b) are set at the start of the program, and just before the call to the runtime-library file initialization. Execution starts with $g, proceeds after a breakpoint with $p, steps single instructions with $x, and steps until the next breakpoint with $$x.

had the advantage of having a good debugger. Symbolic debuggers are common today, sometimes even with fancy GUI front ends that some users like. In 1977, window systems had mostly not yet made it out of Xerox PARC, and the few interactive debuggers available generally worked at the level of assembly language. Figure 3 shows a small example of a session with the low-level *Dynamic Debugging Tool/Technique*, ddt, that otherwise would have been necessary for debugging most programming languages other than SAIL (ALGOL, COBOL, and FORTRAN, and later, PASCAL, also had source-level debuggers).

SAIL had a useful conditional compilation feature, allowing Don to write the keyword definitions shown in Figure 4, and inject a bit of humor into the code.

A scan of the SAIL source code for METAFONT shows several other instances of how the implementation language and host computer affected the METAFONT code:

- 19 buffers for disk files;
- no more than 150 input characters/line;
- initialization handled by a separate program module to save memory;
- bias of 4 added to case statement index to avoid illegal negative cases;
- character raster allocated dynamically to avoid 128K-word limit on core image;
- magic TENEX-dependent code to allocate buffers between the METAFONT code and the SAIL disk buffers because, as Don wrote in a comment in the code, *there is all this nifty core sitting up in the high seg . . . that is just begging to be used*.

Another feature of the PDP-10 that strongly influenced the design of TEX and METAFONT was the way the loader worked. On most other operating systems, the linker or loader reads object files, finds the required libraries, patches unresolved references, and writes an executable image to a disk file. The PDP-10 loader left the program image in memory, relegating the job of copying the memory image to disk to the save command. If the image was not required again, the user could simply start the program without saving it. If the program was

```
# changed to ^P^Q when debugging METAFONT;
define DEBUGONLY = ^Pcomment^Q
...
# used when an array is believed to require
# no bounds checks;
define saf = ^Psafe^Q

# used when SAIL can save time implementing
# this procedure;
define simp = ^Psimple^Q

# when debugging, belief turns to disbelief;
DEBUGONLY redefine saf = ^P^Q

# and simplicity dies too;
DEBUGONLY redefine simp = ^P^Q
```

Figure 4: SAIL conditional compilation for generating additional debugging support. The two control characters displayed as ⊂ and ⊃ at Stanford (octal values 020 and 021 in Table 2).

```
@initex lplain
*\dump
@virtex &lplain
*^C
@save latex
@rename lplain.fmt texformats:
```

Figure 5: Creating a preloaded LATEX executable on TOPS-20.

The initex stage reads `lplain.tex` and dumps the precompiled result to `lplain.fmt`.

The leading ampersand in the virtex stage requests reading of the binary format file, instead of a normal TEX text file. The keyboard interrupt suspends the process, and the next command saves `latex.exe`.

The final command moves the format file to its standard location where it can be found should it be needed again. On TOPS-20, it normally is not read again unless a user wishes to preload further customizations to create another executable program.

The procedure for METAFONT is essentially the same; only the filenames have to be changed.

started, but then interrupted at a quiescent point, such as waiting for input, the memory image could be saved to disk.

Since some of the features of TEX and METAFONT are implemented in their own programming languages, they each need to read that code on every execution. For LATEX, the startup code can amount to tens of thousands of lines. Thus, for small user input files, the startup actions may be significantly more costly than the work needed for the user files. Don therefore divided both programs into two parts: the first parts, called initex and inimf, read the startup code and write their internal tables to a special compact binary file called a *format* file. The second parts, called virtex and virmf, can then read those format files at high speed. If they are then interrupted when they are ready for user input, they can be saved to disk as programs that can later be run with all of this startup processing already done [72, §1203], [70, §1331]. While this sounds complex, in practice, it takes just six lines of user input, shown in Figure 5. This normally only needs to be done by a system manager when new versions of the startup files are installed. It is worth noting that installers of both PDP-10 emacs and modern GNU emacs do a very similar preparation of a dumped-memory image to reduce program-startup cost.

On most other architectures, the two-part split is preserved, but the virtex and virmf programs are then wrapped in scripts that act as the tex and mf programs. On UNIX systems, the script wrappers are not needed: instead, virtex, tex, and latex are filesystem links to the same file, and the name of the program is used internally to determine what format file needs to be automatically loaded. Modern systems are fast enough that the extra economization of preloading the format file into the executable program is relatively unimportant: the fastest systems can now typeset the *TEXbook* at nearly 1100 pages/sec, compared to several seconds per page when TEX was first written. In any event, preloading is difficult to accomplish outside the PDP-10 world. It can be done portably, but much less flexibly, if the preloaded tables are written out as source-code data initializers, and then compiled into the program, as the GNU bc and dc calculators do for their library code.

TEX and METAFONT distributions come with the devious `trip` and `trap` torture tests that Don devised to test whether the programs are behaving properly. One of the drawbacks of the two-part split is that these tests are run with initex and inimf respectively, rather than with the separately-compiled virtex and virmf, which are the programs that users run as tex and mf. I have encountered at least one system where the torture tests passed, yet virtex aborted at run time because of a compiler code-generation error. Fortunately, the error was eliminated when virtex was recompiled with a different optimization level.

Although TEX and METAFONT were designed with great care and attention to detail, and programmed to give identical line-breaking and page-breaking decisions on all platforms, it would have been better if their user communities had collaborated on development of a much more extensive test suite, designed with the help of test-coverage analyzers to ensure that as much of the source code as possible is exercised. These compiler-based tools instrument software in such a way that program execution produces a data file that leads to a report of the number of times that each line of code is reached. This identifies the *hot spots* in the code, but it also reveals the unused, and therefore, untested and untrusted, parts of the program.

When I did such an analysis of runs with the `trip` and `trap` tests, I was surprised to find that just under 49% of all lines of code were executed. I reported these results to the *TEX Live* mailing list on 18 March 2004, in the hope of initiating a project to use the test-coverage feedback to devise additional tests that will exercise most of the other half of the code. It will never be possible to test all of it: there are more than 50 locations in the TEX and METAFONT source code where there is a test for a supposedly-impossible situation, at which point section 95 of TEX (section 90 in METAFONT) is invoked to issue a message prefixed with `This can't happen` and terminate execution.

6 Switching programming languages

Donald Knuth initially expected that TEX and METAFONT would be useful primarily for his own books and papers, but other people were soon clamoring for access, and many of them did not have a PDP-10 computer to run those programs on. The American Mathematical Society was interested in evaluating TEX and METAFONT for its own extensive mathematical-publishing activities, but it could make an investment in switching from the proprietary commercial typesetting system that it was then using *only* if it could be satisfied with the quality, the longevity, and the portability of these new programs.

Researchers at Xerox PARC had translated the SAIL version of TEX to MESA, but that language ran only on Xerox workstations, which, while full of great ideas, were too expensive ever to make any significant market penetration.

It was clear that keeping TEX and METAFONT tied to SAIL and the PDP-10 would ultimately doom them to oblivion. It was also evident that some of the program-design decisions, and the early versions of the Computer Modern fonts, did not produce the high quality that their author demanded of himself.

A new implementation language, and new program designs, were needed, and in 1979–1980, when Don and Ignacio produced prototype code for the new design, there was really only one possibility: PASCAL. However, before you rise to this provocation, why not C instead, since it has become the lingua franca for writing portable software?

UNIX had reached the 16-bit DEC PDP-11 computers at the University of California at Berkeley in 1974. By 1977, researchers there had it running on the new 32-bit DEC VAX, but the C language in which much of UNIX is written was only rarely available outside that environment. Jay Lepreau's pcc20 work was going on in the Computer Science Department at Utah in 1981–82, but it wasn't until about 1983 that TOPS-20 users elsewhere began to get access to it. Our filesystem archives show my first major porting attempt of a C-language UNIX utility to TOPS-20 on 11 February 1983.

PASCAL, a descendant of ALGOL 60 [5], was designed by Niklaus Wirth at ETH in Zürich, Switzerland in 1968. His first attempt at writing a compiler for it in FORTRAN failed, but he then wrote a compiler for a subset of PASCAL in that subset, translated it by hand to assembly language, and was finally able to bootstrap the compiler by getting it to compile itself [106].

Urs Ammann later wrote a completely new compiler [2] in PASCAL for the PASCAL language on the 60-bit CDC 6600 at ETH, a machine class that I myself worked extensively and productively on for nearly four years. That compiler generated machine code directly, instead of producing assembly code, and ran faster, and produced faster code, than Wirth's original bootstrap compiler. Ammann's compiler was the parent of several others, including the one on the PDP-10.

PASCAL is a small language intended for teaching introductory computer-programming skills, and Wirth's book with the great title *Algorithms + Data Structures = Programs* [107] is a classic that is still worthy of study. However, PASCAL is *not* a language that is suitable for larger projects. A fragment of the language is shown in Figure 6, and much more can be seen in the source code for TEX [70] and METAFONT [72].

PASCAL's flaws are well chronicled in a famous article by Brian Kernighan [40, 42]. That paper was written to record the pain that PASCAL caused in implementing a moderate-sized, but influential, programming project [44]. He wrote in his article:

> PASCAL, at least in its standard form, is just plain not suitable for serious programming. ... This botch [confusion of size and type] is the biggest single problem in PASCAL. ... I feel that it is a mistake to use PASCAL for

```
PROCEDURE Scanfilename;
 LABEL 30;
BEGIN
 beginname;
 WHILE buffer[curinput.locfield] = 32 DO
  curinput.locfield := curinput.locfield+1;
 WHILE true DO
 BEGIN
  IF (buffer[curinput.locfield] = 59) OR
     (buffer[curinput.locfield] = 37) THEN
     GOTO 30;
  IF NOT morename(buffer[curinput.locfield])
   THEN GOTO 30;
  curinput.locfield := curinput.locfield+1;
 END;
30:
    endname;
END;
```

Figure 6: Filename scanning in PASCAL, after manual prettyprinting. The statements beginname and endname are calls to procedures without arguments. The magic constants 32, 37, and 59 would normally have been given symbolic names, but this code is output by the tangle preprocessor which already replaced those names by their numeric values. The lack of statements to exit loops and return from procedures forces programmers to resort to the infamous goto statements, which are required to have predeclared numeric labels in PASCAL.

> anything much beyond its original target. In its pure form, PASCAL is a toy language, suitable for teaching but not for real programming.

There is also a good survey by Welsh, Sneeringer, and Hoare [104] of PASCAL's ambiguities and insecurities.

Donald Knuth had co-written a compiler for a subset of ALGOL 60 two decades earlier [4], and had written extensively about that language [47–49, 51, 52, 75]. Moreover, he had developed the fundamental theory of parsing that is used in compilers [50]. He was therefore acutely aware of the limitations of PASCAL, and to enhance portability of TEX and METAFONT, and presciently (see Section 7), to facilitate future translation to other languages, sharply restricted his use of features of that language [70, Part 1].

PASCAL has `new()` and `dispose()` functions for allocating and freeing memory, but implementations were allowed to ignore the latter, resulting in continuously-growing memory use. Therefore, as with the original versions in SAIL, TEX and METAFONT in PASCAL handle their own memory management from large arrays allocated at compile time.

One interesting PASCAL feature is *sets*, which are collections of user-definable objects. The operations of set difference, intersection, membership tests, and union are expected to be fast, since sets can be internally represented as bit strings. For the character processing that TEX carries out, it is very convenient to be able to classify characters according to their function. TEX assigns each input character a *category code*, or catcode for short, that represents these classifications. Regrettably, the PASCAL language definition permitted implementors to choose the maximum allowable set size, and many compilers therefore limited sets to the number of bits in a single machine word, which could be as few as 16. This made sets of characters impossible, even though Wirth and Ammann had used exactly that feature in their PASCAL compilers for the 60-bit CDC 6600. The PDP-10 PASCAL compiler limited sets to 72 elements, fewer than needed for sets of ASCII characters.

A peculiarity of PASCAL is that it does not follow the conventional open-process-close model of file handling. Instead, for input files it combines the open and read of the first item in a single action, called the `reset` statement. Since most implementations provide standard input and output files that are processed before the first statement of the user's main program is executed, this means that the program must read the first item from the user terminal, or input file, before a prompt can even be issued for that input. While some compilers provided workarounds for this dreadful deadlock, not all did, and Don was forced to declare this part of TEX and METAFONT to be system dependent, with each implementor having to find a way to deal with it.

The botch that Brian Kernighan criticized has to do with the fact that, because PASCAL is *strongly typed*, the size of an object is part of its type. If you declare a variable to hold ten characters, then it is illegal to assign a string of any other length to it. If it appears as a routine parameter, then all calls to that routine must pass an argument string of exactly the correct length.

Donald Knuth's solution to this extremely vexing problem for programs like TEX and METAFONT that mainly deal with streams of input characters was to not use PASCAL directly, but rather, to delegate the problem of character-string management, and other tasks, to a preprocessor, called tangle. This tool, and its companion weave, are fundamental for the notion of *literate programming* that he developed during this work [64, 74, 95].

The input to these literate-programming tools is called a WEB, and a fragment of TeX's own WEB code is illustrated in Figure 7. The output of the two utilities is shown in Figures 8 and 9, and the typeset output for the programmer is given in Figure 10.

In order to keep a stable source-code base, the WEB files are *never* edited directly when the code is ported to a new platform. Instead, tangle and weave accept simple *change files* with input blocks

```
@x
old code
@y
new code
@z
```

where the old-code sections must match their order in the WEB file. For TeX and METAFONT, these change files are typically of the order of 5% of the size of the WEB files, and the changes are almost exclusively in the system-dependent parts of those programs, and in the handling of command-line and startup files.

```
@ The |scan_optional_equals| routine looks
for an optional ‘\.=’ sign preceded by
optional spaces; ‘\.{\\relax}’ is not
ignored here.

@p procedure scan_optional_equals;
begin
@<Get the next non-blank non-call token@>;
if cur_tok<>other_token+"=" then back_input;
end;
```

Figure 7: Fragment of tex.web corresponding to section 405 of *TeX: The Program* [70, p. 167]. The vertical bars are a WEB shorthand that requests indexing of the enclosed text. The prose description begins with the command @, and the PASCAL code begins with the command @p. The text @<...> represents a block of code that is defined elsewhere.

Because PASCAL permits only one source-code file per program, WEB files are also monolithic. However, to reduce the size of the typeset program listing, change files normally include a statement \let \maybe = \iffalse near the beginning to disable DVI output of unmodified code sections. Having a single source file simplified building the programs on the PDP-10, which didn't have a UNIX-like make utility until I wrote one in 1988. Figure 11 shows how initex was built on TOPS-20.

In the early 1980s, few users had terminals capable of on-screen display of typeset output, so one of the system-dependent changes that was made in the PDP-10 implementations of TeX was the generation of a candidate command for printing the

```
PROCEDURE SCANOPTIONAL;BEGIN{406:} REPEAT
GETXTOKEN;UNTIL CURCMD<>10{:406};IF CURTOK<>3133
THEN BACKINPUT;END;{:405}{407:}
```

Figure 8: PASCAL code produced from the WEB fragment in Figure 7 by tangle. All superfluous spaces are eliminated on the assumption that humans never need to read the code, even though that may occasionally be necessary during development. Without postprocessing by a PASCAL prettyprinter, such as pform, it is nearly impossible for a human to make sense of the dense run-together PASCAL code from a large WEB file, or to set sensible debugger breakpoints.

To conform to the original definition of PASCAL, and adapt to limitations of various compilers, all identifiers are uppercased, stripped of underscores, and truncated to 12 characters, of which the first 7 must be unambiguous.

Notice that the remote code from the @<...> input fragment has been inserted, and that symbolic constants have been expanded to their numeric values. The braced comments indicate sectional cross references, and no other comments survive in the output PASCAL code.

```
\M405. The \\{scan\_optional\_equals}
routine looks for an optional ‘\.=’ sign
preceded by optional spaces; ‘\.{\\relax}’
is not ignored here.

\Y\P\4\&{procedure}\1\
\37\\{scan\_optional\_equals};\2\6
\&{begin} \37\X406:Get the next non-blank
non-call token\X;\6 \&{if}
$\\{cur\_tok}\I\\{other\_token}+\.{"="}$
\1\&{then}\5 \\{back\_input};\2\6
\&{end};\par \fi
```

Figure 9: TeX typesetter input produced from the WEB fragment in Figure 7 by weave.

405. The *scan_optional_equals* routine looks for an optional ‘=’ sign preceded by optional spaces; ‘\relax’ is not ignored here.

procedure *scan_optional_equals*;
  begin ⟨Get the next non-blank non-call token 406⟩;
  if $cur\_tok \neq other\_token + "="$ **then** *back_input*;
  end;

Figure 10: Typeset output from TeX for the weave fragment in Figure 9. Notice that the remote code block is referenced by name, with a trailing section number that indicates its location in the output listing. Not shown here is the mini-index that is typeset in a footnote, showing the locations elsewhere in the program of variables and procedures mentioned on this output page.

```
@tangle
WEBFILE    : TeX.web
CHANGEFILE : TeX.tops20-changes
PASCALFILE : TeX.pas
POOL       : TeX.pool
@rename TeX.pool TeX:

@set no default compile-switches pas
@load %"ERRORLEVEL:10 -
    INITEX/SAVE/RUNAME:INITEX" TeX.pas
@rename iniTeX.exe TeX:
@delete TeX.rel, TeX.pas
@expunge
```

Figure 11: Building and installing initex on TOPS-20. A similar procedure handled virtex: only the filenames change, and in both cases, the procedure was encapsulated in a command file that allowed a one-line command to do the entire job.

The last command shows a wonderful feature of TOPS-20: deleted files could be undeleted at any time until they were expunged from the filesystem.

Comments from 1986 in the command file noted that on the fastest DEC PDP-10 model, tangle took 102 seconds, and PASCAL compilation, 80 seconds.

When this build was repeated using the KLH10 simulator running on a 2.4GHz AMD64 processor, tangle took only 5 seconds, and PASCAL only 2.6 seconds.

For comparison with a modern TEX build on GNU/LINUX, I used the same AMD64 system for a fresh build. PASCAL generation with tangle took 0.09 seconds, the WEB-to-C conversion (see Section 7) took 0.08 seconds, and compilation of the 14 C-code files took 2.24 seconds. The KLH10 simulator times are clearly outstanding.

The change file on the PDP-10 inserted special compiler directives in a leading comment to select extended addressing. The memory footprint of TEX after typesetting its own source code is 614 pages of 512 words each, or just 1.4MB.

On GNU/LINUX on AMD64 with the 2004 *TEX Live* release, TEX needs 11MB of memory to typeset itself, although of course its tables are much larger, as shown in Table 1.

```
@tex hello.tex
This is TeX, Tops-20 Version 2.991
(preloaded format=plain 5.1.14)
(PS:<BEEBE>HELLO.TEX.1 [1])
Output written on PS:<BEEBE>HELLO.DVI.1
(1 page, 212 bytes).
Transcript written on PS:<BEEBE>HELLO.LST.1.
@TeXspool: PS:<BEEBE>HELLO.DVI.1
```

Figure 12: A TEX run on TOPS-20. The user typed only the first command, and in interactive use, TEX provided the second command, leaving the cursor at the end of the line, so the user could then type a carriage return to accept the command, or a Ctl-U or Ctl-C interrupt character to erase or cancel it.

This feature was implemented via a TOPS-20 system call that allowed a program to simulate terminal input. TEX thereby saved humans some keystrokes, and users could predefine the logical name TeXspool with a suitable value to select their preferred DVI translator. This shortcut is probably infeasible on most other operating systems.

output. A typical run then looked like the sample in Figure 12.

Because PASCAL had mainly been used for small programs, few compilers for that language were prepared to handle programs as large and complex as TEX and METAFONT. Their PASCAL source code produced by tangle amounts to about 20 000 lines each when prettyprinted. A dozen or so supporting tools amount to another 20 000 lines of code, the largest of which is weave.

Ports of TEX and METAFONT to new systems frequently uncovered compiler bugs or resource limits that had to be fixed before the programs could operate. The 16-bit computers were particularly challenging because of their limited address space, and it was a remarkable achievement when Lance Carnes announced TEX on the HP3000 in 1981 [11], followed not long after by his port to the IBM PC with the wretched 64KB memory segments of the Intel 8086 processor. He later founded a company, Personal TEX, Inc. About the same time, David Fuchs completed an independent port to the IBM PC, and that effort was briefly available commercially. David Kellerman and Barry Smith left Oregon Software, where they worked on PASCAL compilers, to found the company Kellerman & Smith to support TEX in the VAX VMS environment. Barry later started Blue Sky Research to support TEX on the Apple MACINTOSH, and David founded Northlake Software to continue support of TEX on VMS.

7 Switching languages, again

Because of compiler problems, UNIX users suffered a delay in getting TEX and METAFONT. Pavel Curtis and Howard Trickey first announced a port in 1983, and lamented [14]:

> Unhappily, the pc [PASCAL] compiler has more deficiencies than one might wish.

Their project at the University of California, Berkeley, took several months, and ultimately, they had to make several changes and extensions to the UNIX PASCAL compiler.

In 1986–1987, Pat Monardo, also at Berkeley, did the UNIX community a great service when he undertook a translation, partly machine assisted, and partly manual, of TEX from PASCAL to C, the result of which he called COMMON TEX. That work ultimately led to the WEB2C project to which many people have contributed, and today, virtually all UNIX installations, and indeed, the entire *TEX Live* distribution for UNIX, Apple MAC OS, and Microsoft WINDOWS, is based on the *completely-automated translation* of the master source files of all TEXware and METAFONTware from the WEB sources to PASCAL and then to C.

8 TEX's progeny

The limitations that stem from the resources and technologies that were available when TEX was developed have since been addressed in various ways. As we showed in Table 1, some of the internal table sizes are relatively easy to expand, as long as the host platform has enough addressable memory.

Growing tables whose indexes are limited to a small number of bits requires deeper changes, and combined with the addition of a small number of new primitives, and several useful extensions, resulted in e-TEX [100]. Its change file is about a quarter the size of tex.web.

TEX has been extended beyond the limitations of eight-bit characters in significant projects for typesetting with the UNICODE character set: OMEGA (Ω) [87, 99], ALEPH (ℵ) [7], and XeTEX [45, 46]. Each is implemented with change files for the TEX or e-TEX WEB sources. For OMEGA, the change files are about as large as tex.web itself, reflecting modification of about half of TEX, and suggesting that a new baseline, or a complete rewrite, may be desirable.

With few exceptions other than GNU groff (a reimplementation of UNIX troff), TEX's DVI file format is not widely known outside the TEX world. Indeed, commercial vendors usurped the DVI acronym to mean *Digital Video Interactive* and *Digital Visual Interface*. Today, electronic representation of typeset documents as page images in PDF format [1] is common. While this format is readily reachable from TEX with translation from DVI to POSTSCRIPT to PDF, or directly to PDF, there are some advantages to being able to access advanced features of PDF such as hypertext links and transparency from within TEX itself. Hàn Thế Thành's pdfTEX [28] is therefore an important extension of TEX that provides PDF output directly, and allows fine control of typography with new features like dynamic font scaling and margin kerning [27, 29]. The change file for pdfTEX is about a third the size of tex.web.

It is worth noting that yet another programming language has since been used to reimplement TEX: Karel Skoupý's work with JAVA [25]. One of the goals of this project was to remove most of the interdependence of the internals of TEX to make it easier to produce TEX-like variants for experiments with new ideas in typography.

Another interesting project is Achim Blumensath's *ANT: A Typesetting System* [8], where the recursive acronym means *ANT is not TEX*. The first version was done in the modern LISP dialect SCHEME, and the current version is in OCAML. Input is very similar to TEX markup, and output can be DVI, POSTSCRIPT, or PDF.

Hong Feng's NeoTEX is a recent development in Wuhan, China, of a typesetting system based on the algorithms of TEX, but completely rewritten in SCHEME, and outputting PDF. Perhaps this work will bring TEX back to its origins, allowing it to be reborn in a truly extensible language.

Although most users view TEX as a *document compiler*, Jonathan Fine has shown how, with small modifications, TEX can be turned into a *daemon* [17]: a permanently-running program that responds to service requests, providing typesetting-on-demand for other programs. At Apple [3], IBM [38], Microsoft [82], SIL [12], and elsewhere, rendering of UNICODE strings is being developed as a common library layer available to all software. These designers have recognized that typesetting is indeed a core service, and many programmers would prefer it to be standardized and made universally available on all computers.

9 METAFONT's progeny

Unlike TEX, METAFONT has so far had only one significant offspring: METAPOST, written by Don's doctoral student John Hobby [36], to whom *METAFONT: The Program* is dedicated. METAPOST is derived from METAFONT, and like that program, is written as a PASCAL WEB. METAPOST normally produces pictures, although it can also generate data for outline font files, and it supports direct output in POSTSCRIPT. METAPOST is described in its manuals [32–35] and parts of two books [22, Chapter 3], [37, Chapter 13].

Although METAFONT, METAPOST, and POSTSCRIPT offer only a two-dimensional drawing model, the 3DLDF program developed by Laurence Finston [18] and the FEATPOST program written by Luis Nobre Gonçalves [19] provide three-dimensional drawing front ends that use METAPOST at the back end. Denis Roegel's 3d.mp package [91] offers a similar extension using the METAPOST programming language.

The recent ASYMPTOTE program [26] credits inspiration from METAPOST, but is a completely independent package for creating high-quality technical drawings, with an input language similar to that of METAPOST.

10 Wrapping up

In this article, I have described how architecture, operating systems, programming languages, and resource limits influenced the design of TEX and METAFONT, and then briefly summarized what has been done in their descendants to expand their capabilities. This analysis is in no way intended to be critical, but instead, to offer a historical retrospective that is, I believe, helpful to think about for other widely-used software packages as well.

TEX and METAFONT, and the literate programming system in which they are written, are truly remarkable projects in software engineering. Their flexibility, power, reliability, and stability, and their unfettered availability, have allowed them to be widely used and relied upon in academia, industry, and government. Donald Knuth expects to use them for the rest of his career, and so do many others, including this author. Don's willingness to expose his programs to public scrutiny by publishing them as books [70, 72, 74], to further admit to errors in them [61, 62] in order to learn how we might become better programmers, and then to pay monetary rewards (doubled annually for several years) for the report of each new bug, are traits too seldom found in others.

11 Bibliography

[1] Adobe Systems Incorporated. *PDF reference: Adobe portable document format, version 1.3*. Addison-Wesley, Reading, MA, USA, second edition, 2000. ISBN 0-201-61588-6. URL http://partners.adobe.com/asn/developer/acrosdk/DOCS/PDFRef.pdf.

[2] Urs Ammann. On code generation in a PASCAL compiler. *Software—Practice and Experience*, 7(3): 391–423, May/June 1977. ISSN 0038-0644.

[3] Apple Computer, Inc. Apple Type Services for Unicode Imaging [ATSUI]. World-Wide Web document., 2005. URL http://developer.apple.com/intl/atsui.html;http://developer.apple.com/fonts/TTRefMan/RM06/Chap6AATIntro.html. Apple Type Services for Unicode Imaging (ATSUI) is a set of services for rendering Unicode-encoded text.

[4] G. A. Bachelor, J. R. H. Dempster, D. E. Knuth, and J. Speroni. SMALGOL-61. *Communications of the Association for Computing Machinery*, 4(11): 499–502, November 1961. ISSN 0001-0782. URL http://doi.acm.org/10.1145/366813.366843.

[5] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Revised report on the algorithmic language Algol 60. *Communications of the Association for Computing Machinery*, 6(1):1–17, January 1963. ISSN 0001-0782. URL http://doi.acm.org/10.1145/366193.366201. Edited by Peter Naur. Dedicated to the memory of William Turanski.

[6] Nelson H. F. Beebe. 25 years of TEX and METAFONT: Looking back and looking forward: TUG 2003 keynote address. *TUGboat*, 25(1): 7–30, 2004. URL http://www.math.utah.edu/~beebe/talks/tug2003/. Due to a journal production error, this article did not appear in the TUG 2003 proceedings volume, even though it was ready months in advance.

[7] Giuseppe Bilotta. Aleph extended TEX. World-Wide Web document and software, December 2004. URL http://ctan.tug.org/tex-archive/help/Catalogue/entries/aleph.html.

[8] Achim Blumensath. ANT: A typesetting system. World-Wide Web document and software, October 2004. URL http://www-mgi.informatik.rwth-aachen.de/~blume/Download.html.

[9] Ronald F. Brender. Generation of BLISSes. *IEEE Transactions on Software Engineering*, SE-6(6): 553–563, November 1980. ISSN 0098-5589. Based on Carnegie-Mellon University Computer Science Report CMU-CS-79-125 May 1979.

[10] Ronald F. Brender. The BLISS programming language: a history. *Software—Practice and Experience*, 32(10):955–981, August 2002. ISSN 0038-0644. DOI http://dx.doi.org/10.1002/spe.470.

[11] Lance Carnes. TEX for the HP3000. *TUGboat*, 2(3): 25–26, November 1981. ISSN 0896-3207.

[12] Sharon Correll. Graphite. World-Wide Web document and software, November 2004. URL http://scripts.sil.org/RenderingGraphite. Graphite is a project under development within SIL's Non-Roman Script Initiative and Language Software Development groups to provide rendering capabilities for complex non-Roman writing systems.

[13] M. Crispin. RFC 4042: UTF-9 and UTF-18 efficient transformation formats of Unicode, April 2005. URL ftp://ftp.internic.net/rfc/rfc4042.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc4042.txt.

[14] Pavel Curtis and Howard Trickey. Porting TEX to VAX/UNIX. *TUGboat*, 4(1):18–20, April 1983. ISSN 0896-3207.

[15] Frank da Cruz and Christine Gianone. The DECSYSTEM-20 at Columbia University (1977–1988). Technical report, The Kermit Project,

Columbia University, New York, NY, USA, December 1988. URL http://www.columbia.edu/kermit/dec20.html.

[16] Edward R. Fiala. MAXC systems. *Computer*, 11(5):57–67, May 1978. ISSN 0018-9162. URL http://research.microsoft.com/~lampson/Systems.html#maxc.

[17] Jonathan Fine. Instant Preview and the TEX daemon. *TUGboat*, 22(4):292–298, December 2001. ISSN 0896-3207.

[18] Laurence D. Finston. *3DLDF user and reference manual: 3-dimensional drawing with METAPOST output*, 2004. URL http://dante.ctan.org/CTAN/graphics/3DLDF/3DLDF.pdf. Manual edition 1.1.5.1 for 3DLDF version 1.1.5.1 January 2004.

[19] Luis Nobre Gonçalves. FEATPOST and a review of 3D METAPOST packages. In Apostolos Syropoulos, Karl Berry, Yannis Haralambous, Baden Hughes, Steven Peter, and John Plaice, editors, *TEX, XML, and Digital Typography: International Conference on TEX, XML, and Digital Typography, held jointly with the 25th Annual Meeting of the TeX Users Group, TUG 2004, Xanthi, Greece, August 30–September 3, 2004: Proceedings*, volume 3130 of *Lecture Notes in Computer Science*, pages 112–124, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2004. Springer-Verlag. ISBN 3-540-22801-2. DOI 10.1007/b99374. URL http://link.springer-ny.com/link/service/series/0558/tocs/t3130.htm.

[20] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LATEX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 1994. ISBN 0-201-54199-8.

[21] Michel Goossens and Sebastian Rahtz. *The LATEX Web companion: integrating TEX, HTML, and XML*. Tools and Techniques for Computer Typesetting. Addison-Wesley Longman, Harlow, Essex CM20 2JE, England, 1999. ISBN 0-201-43311-7. With Eitan M. Gurari, Ross Moore, and Robert S. Sutor.

[22] Michel Goossens, Sebastian Rahtz, and Frank Mittelbach. *The LATEX Graphics Companion: Illustrating Documents with TEX and PostScript*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, 1997. ISBN 0-201-85469-4.

[23] Ralph E. Gorin. *Introduction to DECSYSTEM-20 Assembly Language Programming*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 1981. ISBN 0-932376-12-6.

[24] Katie Hafner and Matthew Lyon. *Where wizards stay up late: the origins of the Internet*. Simon and Schuster, New York, NY, USA, 1996. ISBN 0-684-81201-0.

[25] Hans Hagen. The status quo of the $\mathcal{N_TS}$ project. *TUGboat*, 22(1/2):58–66, March 2001. ISSN 0896-3207.

[26] Andy Hammerlindl, John Bowman, and Tom Prince. ASYMPTOTE*: a script-based vector graphics language*. Faculty of Science, University of Alberta, Edmonton, AB, Canada, 2004. URL http://asymptote.sourceforge.net/. ASYMPTOTE is a powerful script-based vector graphics language for technical drawing, inspired by METAPOST but with an improved C++-like syntax. ASYMPTOTE provides for figures the same high-quality level of typesetting that LATEX does for scientific text.

[27] Hàn Thế Thành. Margin kerning and font expansion with pdfTEX. *TUGboat*, 22(3):146–148, September 2001. ISSN 0896-3207.

[28] Hàn Thế Thành and Sebastian Rahtz. The pdfTEX user manual. *TUGboat*, 18(4):249–254, December 1997. ISSN 0896-3207.

[29] Hàn Thế Thành. Improving TEX's typeset layout. *TUGboat*, 19(3):284–288, September 1998. ISSN 0896-3207.

[30] Ken Harrenstien. KLH10 PDP-10 emulator. World-Wide Web document and software, 2001. URL http://klh10.trailing-edge.com/. This is a highly-portable simulator that allows TOPS-20 to run on most modern Unix workstations.

[31] C. A. R. Hoare. Hints on programming language design. In *Conference record of ACM Symposium on Principles of Programming Languages: papers presented at the symposium, Boston, Massachusetts, October 1–3, 1973*, pages iv + 242, New York, NY 10036, USA, 1973. ACM Press. URL ftp://db.stanford.edu/pub/cstr/reports/cs/tr/73/403/CS-TR-73-403.pdf. Keynote address. Also available as Stanford University Computer Science Department Report CS-TR-73-403 1973.

[32] John D. Hobby. Introduction to METAPOST. In Jiří Zlatuška, editor, *EuroTEX 92: Proceedings of the 7th European TEX Conference*, pages 21–36, Brno, Czechoslovakia, September 1992. Masarykova Universita. ISBN 80-210-0480-0. Invited talk.

[33] John D. Hobby. *Drawing Graphs with METAPOST*. AT&T Bell Laboratories, Murray Hill, NJ, USA, 1995. URL http://ctan.tug.org/tex-archive/macros/latex/contrib/pdfslide/mpgraph.pdf.

[34] John D. Hobby. *The METAPOST System*, December 1997. URL file:///texlive-2004-11/texmf-dist/doc/metapost/base/mpintro.pdf.

[35] John D. Hobby. *A User's Manual for METAPOST*, 2004. URL file:///texlive-2004-11/texmf-dist/doc/metapost/base/mpman.pdf.

[36] John Douglas Hobby. *Digitized Brush Trajectories*. Ph.D. dissertation, Department of Computer Science, Stanford University, Stanford, CA, USA, June 1986. URL http://wwwlib.umi.com/dissertations/fullcit/8602484. Also published as report STAN-CS-1070 (1985).

[37] Alan Hoenig. *TEX Unbound: LATEX and TEX Strategies for Fonts, Graphics, & More*. Oxford University

Press, Walton Street, Oxford OX2 6DP, UK, 1998. ISBN 0-19-509686-X (paperback), 0-19-509685-1 (hardcover). URL http://www.oup-usa.org/gcdocs/gc_0195096851.html.

[38] IBM Corporation. International Component for Unicode (ICU). World-Wide Web document., 2005. URL http://www-306.ibm.com/software/globalization/icu/index.jsp. ICU is a mature, widely used set of C/C++ and Java libraries for Unicode support, software internationalization and globalization (i18n and g11n).

[39] B. W. Kernighan and M. E. Lesk. UNIX document preparation. In J. Nievergelt, G. Coray, J.-D. Nicoud, and A. C. Shaw, editors, *Document Preparation Systems: A Collection of Survey Articles*, pages 1–20. Elsevier North-Holland, Inc., New York, NY, USA, 1982. ISBN 0-444-86493-8.

[40] Brian W. Kernighan. Why Pascal is not my favorite programming language. Computer Science Report 100, AT&T Bell Laboratories, Murray Hill, NJ, USA, July 1981. URL http://cm.bell-labs.com/cm/cs/cstr/100.ps.gz. Published in [42].

[41] Brian W. Kernighan. PIC: A language for typesetting graphics. *Software—Practice and Experience*, 12(1):1–21, January 1982. ISSN 0038-0644.

[42] Brian W. Kernighan. Why Pascal is not my favorite programming language. In Alan R. Feuer and Narain Gehani, editors, *Comparing and assessing programming languages: Ada, C, and Pascal*, Prentice-Hall software series, pages 170–186. Prentice-Hall, Englewood Cliffs, NJ, USA, 1984. ISBN 0-13-154840-9 (paperback), 0-13-154857-3 (hardcover). See also [40].

[43] Brian W. Kernighan and Lorinda L. Cherry. A system for typesetting mathematics. *Communications of the Association for Computing Machinery*, 18(3):151–156, March 1975. ISSN 0001-0782.

[44] Brian W. Kernighan and P. J. Plauger. *Software Tools in Pascal*. Addison-Wesley, Reading, MA, USA, 1981. ISBN 0-201-10342-7.

[45] Jonathan Kew. The XeTEX typesetting system. World-Wide Web document., March 2004. URL http://scripts.sil.org/xetex.

[46] Jonathan Kew. The multilingual lion: TEX learns to speak Unicode. In *Twenty-seventh Internationalization and Unicode Conference (IUC27). Unicode, Cultural Diversity, and Multilingual Computing, April 6–8, 2005, Berlin, Germany*, pages $n{+}1$–$n{+}17$, San Jose, CA, USA, 2005. The Unicode Consortium.

[47] D. E. Knuth, L. L. Bumgarner, D. E. Hamilton, P. Z. Ingerman, M. P. Lietzke, J. N. Merner, and D. T. Ross. A proposal for input-output conventions in ALGOL 60. *Communications of the Association for Computing Machinery*, 7(5):273–283, May 1964. ISSN 0001-0782. URL http://doi.acm.org/10.1145/364099.364222. Russian translation by M. I. Ageev in *Sovremennoe Programmirovanie* **1** (Moscow: Soviet Radio, 1966), 73–107.

[48] Donald E. Knuth. Man or boy? *Algol Bulletin (Amsterdam: Mathematisch Centrum)*, 17:7, January 1964. ISSN 0084-6198.

[49] Donald E. Knuth. Man or boy? *Algol Bulletin (Amsterdam: Mathematisch Centrum)*, 19(7):8–9, January 1965. ISSN 0084-6198.

[50] Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8 (6):607–639, December 1965. ISSN 0019-9958. Reprinted in [78].

[51] Donald E. Knuth. Teaching ALGOL 60. *Algol Bulletin (Amsterdam: Mathematisch Centrum)*, 19:4–6, January 1965. ISSN 0084-6198.

[52] Donald E. Knuth. The remaining trouble spots in ALGOL 60. *Communications of the Association for Computing Machinery*, 10(10):611–618, October 1967. ISSN 0001-0782. URL http://doi.acm.org/10.1145/363717.363743. Reprinted in E. Horowitz, *Programming Languages: A Grand Tour* (Computer Science Press, 1982), 61–68.

[53] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 1968. ISBN 0-201-03803-X. Second printing, revised, July 1969.

[54] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 1969. ISBN 0-201-03802-1.

[55] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 1971. ISBN 0-201-03802-1. Second printing, revised, November 1971.

[56] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, second edition, 1973. ISBN 0-201-03809-9. Second printing, revised, February 1975.

[57] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 1973. ISBN 0-201-03803-X.

[58] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, March 1975. ISBN 0-201-03803-X. Second printing, revised.

[59] Donald E. Knuth. *TEX and METAFONT—New Directions in Typesetting*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 1979. ISBN 0-932376-02-9.

[60] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, second edition, 1981. ISBN 0-201-03822-6.

[61] Donald E. Knuth. The errors of TEX. Technical Report STAN-CS-88-1223, Stanford University, Department of Computer Science, September 1988. See [62].

[62] Donald E. Knuth. The errors of TEX. *Software—Practice and Experience*, 19(7):607–685, July 1989. ISSN 0038-0644. This is an updated version of [61]. Reprinted with additions and corrections in [64, pp. 243–339].

[63] Donald E. Knuth. The new versions of TEX and METAFONT. *TUGboat*, 10(3):325–328, November 1989. ISSN 0896-3207.

[64] Donald E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992. ISBN 0-937073-80-6 (paper), 0-937073-81-4 (cloth).

[65] Donald E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, third edition, 1997. ISBN 0-201-89683-4.

[66] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, third edition, 1997. ISBN 0-201-89684-2.

[67] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, second edition, 1998. ISBN 0-201-89685-0.

[68] Donald E. Knuth. *Digital Typography*. CSLI Publications, Stanford, CA, USA, 1999. ISBN 1-57586-011-2 (cloth), 1-57586-010-4 (paperback).

[69] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13447-0.

[70] Donald E. Knuth. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13437-3.

[71] Donald E. Knuth. *The METAFONTbook*, volume C of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13445-4.

[72] Donald E. Knuth. *METAFONT: The Program*, volume D of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13438-1.

[73] Donald E. Knuth. *Computer Modern Typefaces*, volume E of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986. ISBN 0-201-13446-2.

[74] Donald E. Knuth and Silvio Levy. *The CWEB System of Structured Documentation, Version 3.0*. Addison-Wesley, Reading, MA, USA, 1993. ISBN 0-201-57569-8.

[75] Donald E. Knuth and Jack N. Merner. ALGOL 60 confidential. *Communications of the Association for Computing Machinery*, 4(6):268–272, June 1961. ISSN 0001-0782. URL http://doi.acm.org/10.1145/366573.366599.

[76] Leslie Lamport. *LATEX—A Document Preparation System—User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, 1985. ISBN 0-201-15790-X.

[77] Leslie Lamport. *LATEX: A Document Preparation System: User's Guide and Reference Manual*. Addison-Wesley, Reading, MA, USA, second edition, 1994. ISBN 0-201-52983-1.

[78] Phillip Laplante, editor. *Great papers in computer science*. IEEE Computer Society Press, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1996. ISBN 0-314-06365-X (paperback), 0-7803-1112-4 (hardcover). URL http://bit.csc.lsu.edu/~chen/GreatPapers.html.

[79] Franklin Mark Liang. Word hy-phen-a-tion by com-pu-ter. Technical Report STAN-CS-83-977, Stanford University, Stanford, CA, USA, August 1983. URL http://www.tug.org/docs/liang/.

[80] Franklin Mark Liang. *Word Hy-phen-a-tion by Com-pu-ter*. Ph.D. dissertation, Computer Science Department, Stanford University, Stanford, CA, USA, March 1984. URL http://wwwlib.umi.com/dissertations/fullcit/8329742;http://www.tug.org/docs/liang/.

[81] Charles E. Mackenzie. *Coded Character Sets: History and Development*. The Systems Programming Series. Addison-Wesley, Reading, MA, USA, 1980. ISBN 0-201-14460-3.

[82] Microsoft Corporation. Unicode and character sets. World-Wide Web document., 2005. URL http://msdn.microsoft.com/library/en-us/intl/unicode_6bqr.asp.

[83] Frank Mittelbach, Michel Goossens, Johannes Braams, David Carlisle, Chris Rowley, Christine Detig, and Joachim Schrod. *The LATEX Companion*. Tools and Techniques for Computer Typesetting. Addison-Wesley, Reading, MA, USA, second edition, 2004. ISBN 0-201-36299-6.

[84] T. Mock. RFC 698: Telnet extended ASCII option, July 1975. URL ftp://ftp.internic.net/rfc/rfc698.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc698.txt. Status: PROPOSED STANDARD. Not online.

[85] Sao Khai Mong. A Fortran version of METAFONT. *TUGboat*, 3(2):25–25, October 1982. ISSN 0896-3207.

[86] William M. Newman and Robert F. Sproull. *Principles of Interactive Computer Graphics*. McGraw-Hill Computer Science Series, Editors: Richard W. Hamming and Edward A. Feigenbaum. McGraw-Hill, New York, NY, USA, 1973. ISBN 0-07-046337-9.

[87] John Plaice and Yannis Haralambous. The latest developments in Ω. *TUGboat*, 17(2):181–183, June 1996. ISSN 0896-3207.

[88] Michael F. Plass. *Optimal pagination techniques for automatic typesetting systems*. Ph.D.

dissertation, Computer Science Department, Stanford University, Stanford, CA, USA, 1981. URL http://wwwlib.umi.com/dissertations/fullcit/8124134.

[89] Brian K. Reid. A high-level approach to computer document formatting. In *Conference record of the seventh annual ACM Symposium on Principles of Programming Languages. Las Vegas, Nevada, January 28–30, 1980*, pages 24–31, New York, NY 10036, USA, 1980. ACM Press. ISBN 0-89791-011-7. ACM order no. 549800.

[90] Brian Keith Reid. *Scribe: a document specification language and its compiler*. Ph.D. dissertation, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, December 1980. URL http://wwwlib.umi.com/dissertations/fullcit/8114634. Also issued as Report CMU-CS-81-100.

[91] Denis Roegel. Creating 3D animations with METAPOST. *TUGboat*, 18(4):274–283, December 1997. ISSN 0896-3207. URL http://ctan.tug.org/tex-archive/graphics/metapost/contrib/macros/3d/doc/paper1997corrected.pdf.

[92] Lynn Elizabeth Ruggles. *Paragon, an interactive, extensible, environment for typeface design*. Ph.D. dissertation, University of Massachusetts Amherst, Amherst, MA, USA, 1987. URL http://wwwlib.umi.com/dissertations/fullcit/8805968.

[93] Peter H. Salus. *A quarter century of UNIX*. Addison-Wesley, Reading, MA, USA, 1994. ISBN 0-201-54777-5.

[94] Ray Scott and Michel E. Debar. TOPS-20 extended Programmable Command Language user's guide and reference manual. Technical report, Carnegie Mellon University Computation Center and FNDP Computing Centre, Pittsburgh, PA, USA and Namur, Belgium, January 1983. URL http://www.math.utah.edu/~bowman/pcl.txt.

[95] E. Wayne Sewell. *Weaving a Program: Literate Programming in WEB*. Van Nostrand Reinhold, New York, NY, USA, 1989. ISBN 0-442-31946-0.

[96] Guy L. Steele Jr. *Common Lisp—The Language*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, 1984. ISBN 0-932376-41-X.

[97] Guy L. Steele Jr. *Common Lisp—The Language*. Digital Press, 12 Crosby Drive, Bedford, MA 01730, USA, second edition, 1990. ISBN 1-55558-041-6 (paperback), 1-55558-042-4 (hardcover), 0-13-152414-3 (Prentice-Hall). See also [96].

[98] Bjarne Stroustrup. *The Design and Evolution of C++*. Addison-Wesley, Reading, MA, USA, 1994. ISBN 0-201-54330-3.

[99] Apostolos Syropoulos, Antonis Tsolomitis, and Nick Sofroniou. *Digital typography using LaTeX*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 2003. ISBN 0-387-95217-9.

[100] Phil Taylor. ε-TeX V2: a peek into the future. *TUGboat*, 18(4):239–242, December 1997. ISSN 0896-3207.

[101] Larry Tesler. PUB: The document compiler. Stanford AI Project Operating Note 70, Department of Computer Science, Stanford University, Stanford, CA, USA, September 1972. URL http://www.nomodes.com/pub_manual.html.

[102] The Unicode Consortium. *The Unicode Standard, Version 4.0*. Addison-Wesley, Reading, MA, USA, 2003. ISBN 0-321-18578-1. URL http://www.unicode.org/versions/Unicode4.0.0/. Includes CD-ROM.

[103] Ulrik Vieth. Math typesetting in TeX: The good, the bad, the ugly. World-Wide Web document, September 2001. URL http://www.ntg.nl/eurotex/vieth.pdf. Lecture slides for EuroTeX 2001 Conference, Kerkrade, The Netherlands.

[104] J. Welsh, W. J. Sneeringer, and C. A. R. Hoare. Ambiguities and insecurities in Pascal. *Software—Practice and Experience*, 7(6):685–696, November/December 1977. ISSN 0038-0644.

[105] John Wharton. Gary Kildall, industry pioneer, dead at 52. created first microcomputer languages, disk operating systems. *Microprocessor Report*, 8 (10):1–2, August 1994. ISSN 0899-9341. URL http://www.ece.umd.edu/courses/enee759m.S2002/papers/wharton1994-kildall.pdf;http://en.wikipedia.org/wiki/Gary_Kildall. This obituary nicely describes the very many accomplishments of this industry pioneer.

[106] Niklaus Wirth. The design of a PASCAL compiler. *Software—Practice and Experience*, 1(4):309–333, October/December 1971. ISSN 0038-0644.

[107] Niklaus Wirth. *Algorithms + Data Structures = Programs*. Prentice-Hall Series in Automatic Computation. Prentice-Hall, Englewood Cliffs, NJ, USA, 1976. ISBN 0-13-022418-9.

[108] F. H. G. Wright II and R. E. Gorin. *FAIL*. Computer Science Department, Stanford University, Stanford, CA, USA, May 1974. Stanford Artificial Intelligence Laboratory Memo AIM-226 and Computer Science Department Report STAN-CS-74-407.

[109] W. A. (William A.) Wulf, D. B. Russell, and A. N. Habermann. BLISS: A language for systems programming. *Communications of the Association for Computing Machinery*, 14(12):780–790, December 1971. ISSN 0001-0782. URL http://doi.acm.org/10.1145/362919.362936.

[110] Ignacio Andres Zabala Salelles. *Interfacing with graphics objects*. PhD thesis, Department of Computer Science, Stanford University, Stanford, CA, USA, December 1982. URL http://wwwlib.umi.com/dissertations/fullcit/8314505.

Strategies for including graphics in LaTeX documents

Klaus Höppner
Nieder-Ramstädter Str. 47
64283 Darmstadt
Germany
klaus.hoeppner@gmx.de

Abstract

This talk presents strategies for including graphics into LaTeX documents. It shows the usage of the standard `graphics` packages of LaTeX as well as an introduction to different graphics formats. Some external tools for converting graphics formats are discussed.

Overview of graphics formats

In general, there exist two kinds of graphics formats: vector and bitmap graphics. For bitmaps, there exist different flavors: no compression (which can make your files truly huge, dependent on resolution and color depth, so I won't cover them from here on), compression methods which completely preserve the image quality while reducing the data size, and "lossy" compression methods which cause a consequent reduction in image quality.

So let's go more into detail:

Vector graphics are set up by drawing or filling geometrical objects such as lines, Bézier curves, polygons, circles and so on. The properties of these objects are stored mathematically. Vector graphics are in general device independent. It is easy to scale or rotate them without loss of quality, since the job of rasterizing them into actual pixels is done by the printer or printer driver.

Bitmaps without lossy compression store the image information as pixels, each pixel of a given color. In principle, the quality of a bitmap becomes better with increased resolution and color depth (e. g. GIF files use a color depth of 8 bits, leading to 256 different indexed colors while a bitmap with 24 bit color depth can have about 16 million colors). Scaling and rotating bitmap images will yield a loss of quality, and printing bitmaps to a device with a different resolution can produce bad results. Fig. 1 shows the difference between a scaled image as vector and bitmap graphics.

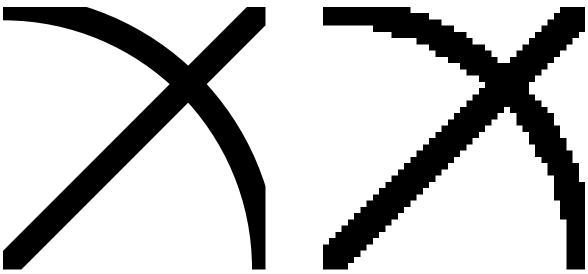


Figure 1: Zoomed view into a sample image as vector graphics (left) and bitmap (right).

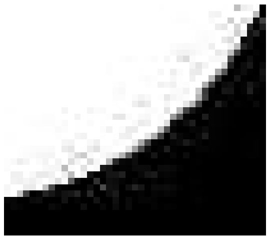


Figure 2: A low quality JPEG image showing some artifacts at the transition between black and white.

Bitmaps with lossy compression use the fact that the human eye is fairly good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. For this reason, components in the high frequency region can be reduced, leading to smaller file sizes. This works well for photographs that usually contain smooth transitions in color, but for graphics with a sharp border, artifacts can occur, as shown in fig. 2. The most prominent graphics format using lossy compression is JPEG.

Graphics formats in practice

There exist very many graphics formats, so I will concentrate on a few of those most often used:

EPS is the encapsulated PostScript format. It is mostly used for vector graphics but can also contain bitmaps.

PNG is the portable network graphics format. It was introduced due to the problem that Unisys claimed a patent for the compression algorithm used in GIF format. For this reason, it is often used nowadays on web pages. PNG is a bitmap format that supports compression both with and without loss of image quality.

JPEG is a bitmap format with lossy compression and is often used for photographs (e. g. most digital cameras produce JPEG files).

TIFF is a bitmap format sometimes used for high quality pictures — in part because it supports the CMYK color space important especially for commercial printing.

Now the question is: What format shall I use for what purpose? Though there is no one true answer to this question, my advice is as follows:

1. For drawings (e. g. technical drawings or data plots) use vector graphics. It gives you maximum freedom to manipulate the image when including it into a document where you often need to scale the image to fit into your layout. Additionally, it is independent of the output device, and thus you can zoom into the image in your document viewer without seeing single pixels.

   Drawing tools offered by TEX distributions — notably PSTricks and METAPOST — can usually produce EPS output natively. Most vector drawing programs like `xfig` and Corel Draw also offer export functionality for producing EPS output (though sometimes buggy).
2. If you are stuck with bitmaps, use PNG for images with sharp color transitions, such as black and white boundaries.
3. For photographs, you can use JPEG in most cases, since the quality loss by compression is normally imperceptible when printed. On most devices, a resolution of 100 to 200 dpi will be sufficient (remember that screen resolution is normally about 75 to 100 dpi, and color printers claim to have high resolutions but dither color prints, so you will hardly notice the difference compared to JPEGs with higher resolution).

The LATEX graphics package

Since the introduction of LATEX 2$_\varepsilon$, the `graphics` bundle is part of the standard package set accompanying the LATEX base distribution [1]. It consists of two style files, `graphics.sty` and `graphicx.sty`. While `graphics.sty` requires the use of `\scalebox` and `\rotatebox` for scaling or rotating graphics, the extended style `graphicx.sty` supports scaling and rotating using the `keyval` package, which provides a convenient interface for specifying parameters. In general, there is no reason not to always use `graphicx.sty`.

So the first step is to load the `graphicx` style file after the `\documentclass` statement:

```
\usepackage{graphicx}
```

In fact, the TEX compiler doesn't know anything about graphics, and including them is done by the DVI driver. So the `graphicx` package has to do two things:

1. find the bounding box of the image (this can be troublesome when you have e. g. an EPS file created by an application that wrote a wrong `BoundingBox` comment — in this case, it can be helpful to put the `\includegraphics` command into an `\fbox` to find out what `graphicx` *thinks* about the bounding box);
2. produce the appropriate `\special` for the output driver; thus, the usage of the `graphics` bundle is driver dependent.

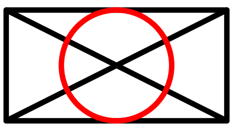
Nowadays, there are two main workflows for producing documents: using `latex` to produce a DVI file and then `dvips` for converting it to PostScript, and using `pdflatex` to produce a PDF file. Most modern TEX systems are configured to automatically check whether you are using `latex` or `pdflatex` and producing `dvips` `\specials` in the first case and the appropriate `\pdfimage` commands in the second case. So if you are using one of the above workflows, you shouldn't need to specify your output backend explicitly. If you are using another backend you have to specify it as an option, e. g.

```
\usepackage[dvipsone]{graphicx}
```

(for the Y&Y `dvipsone` driver), but be aware that other backends often don't support scaling or rotating. For example, DVI previewers like `xdvi` or `windvi` try to interpret the `dvips` specials, but rotations may not be displayed properly in DVI preview.

After the package is loaded, to include an image simply use

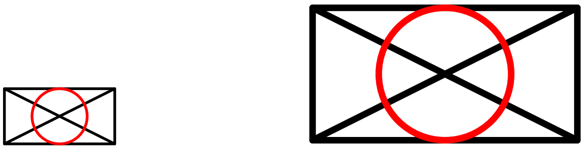
```
\includegraphics{sample}
```



Please notice that no extension for the file was given. The explanation why will follow later. In the case of using `\includegraphics` without options the image is included at its natural size, as shown above. When using the `graphicx` style, you can scale your image by a factor:

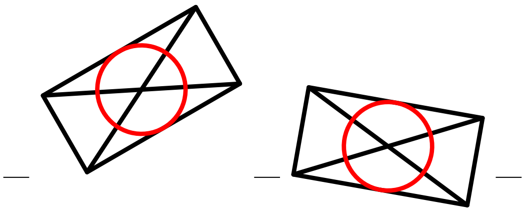
```
\includegraphics[scale=0.5]{sample}
```

```
\includegraphics[scale=1.2]{sample}
```



Another option supports rotating an image:

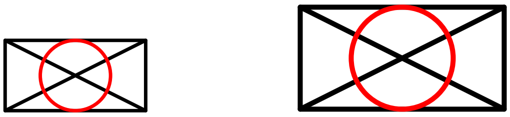
```
\includegraphics[angle=30]{sample}
\includegraphics[angle=-10]{sample}
```



Positive numbers lead to counterclockwise rotation, negative numbers to clockwise rotation. The origin for the rotation is the lower left corner of the image, so in the clockwise rotation above the result has not only a height but also a depth below the baseline (as shown by the rules).

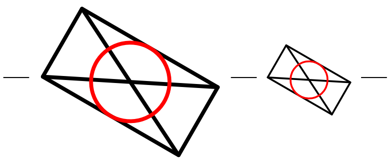
Images can not only be scaled by a given factor, you can specify a height and/or width for the resulting image instead:

```
\includegraphics[width=2cm]{sample}
\includegraphics[height=1.5cm]{sample}
```



`height` gives the height above the baseline. If your image has a depth, you can use `totalheight` instead, i. e. the sum of height and depth will be scaled to the given length.

```
\includegraphics[angle=-30,height=1cm]
    {sample}
\includegraphics[angle=-30,
    totalheight=1cm]{sample}
```



You can specify both `width` and `height`. In this case your image may be scaled differently in horizontal and vertical direction, unless you use the `keepaspectratio` option:

```
\includegraphics[width=1.5cm,height=1.5cm]
    {sample}
\includegraphics[width=1.5cm,height=1.5cm,
    keepaspectratio]{sample}
```

Source	Target	Tool
`latex+dvips`		
EPS		directly supported
PNG	EPS	ImageMagick/netpbm
JPEG	EPS	ImageMagick/netpbm
TIFF	EPS	ImageMagick/netpbm/`tif2eps`
`pdflatex`		
PDF		directly supported
EPS	PDF	`epstopdf`
PNG		directly supported
JPEG		directly supported
TIFF	PNG	ImageMagick/netpbm
TIFF	PDF	`tif2eps+epstopdf`

Table 1: Conversion of graphics formats supported by `latex+dvips` and `pdflatex`.

Please notice that usage of `angle` and `width` or `height` is sensitive to the order in which the options are given. Specifying the angle first means that your image is rotated first and then the rotated image is scaled to the desired width or height, while specifying a width or height first will first scale the natural image and rotate it afterwards.

Supported graphics formats

To make things a bit more complicated, `latex` with `dvips` and `pdflatex` support different graphics formats:

`latex+dvips`: EPS

`pdflatex`: PDF, PNG, JPEG, MPS

Table 1 shows ways to convert the standard graphics formats to supported formats. In particular, converting EPS graphics used with `latex+dvips` to PDF for `pdflatex` workflow is quite easy; just run the `epstopdf` Perl script, which uses Ghostscript to convert EPS to PDF.

This also explains why it is generally best to give the file names in `\includegraphics` commands without extensions. In this case the `graphics` package looks for a supported graphics format automatically. So if you have an image both as EPS and (e. g.) PDF, you can use both the `latex+dvips` and `pdflatex` workflows without changing your source.

One other useful special case: including the output of METAPOST is also easy; although it is technically an EPS file, it uses only a small set of com-

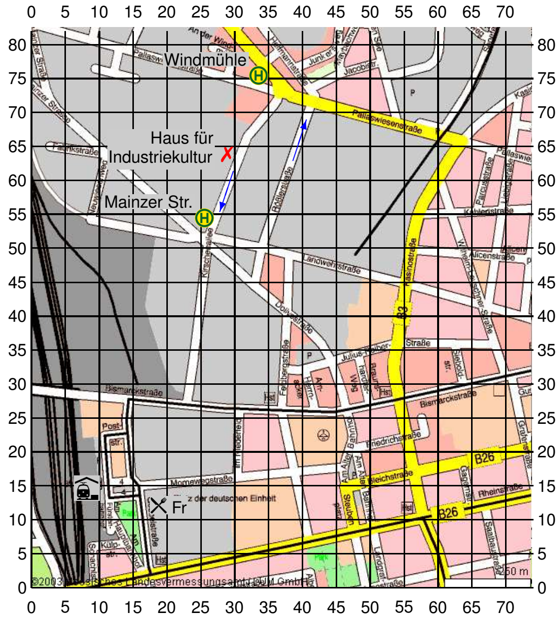


Figure 3: A map with additional marks produced with overpic

mands. So `pdflatex` can support the inclusion of METAPOST output directly. The only thing you have to do is to change the file extension of the output file to `.mps`.

Tools for image conversion

There exist several tools for conversion of graphics formats, both free and commercial. Besides free GUI-based tools like Gimp on Unix systems there are two command line tools available for Unix and Windows: ImageMagick [2] and netpbm [3].

ImageMagick can convert images directly, e. g. by typing

```
convert sample.gif sample.png
```

while netpbm uses the pnm format as intermediate format:

```
giftopnm sample.gif | pnmtopng - > sample.png
```

Another nice tool is `tif2eps` by Bogusław Jackowski et al. [4] which uses Ghostscript to convert a TIFF file to EPS, e. g.

```
gs -- tif2eps.ps sample.tif sample.esp -rh
```

which produces a RLE compressed and hex encoded EPSfile. In my experience EPS files produced with `tif2eps` are smaller than those produced by ImageMagick. Additionally it supports CMYK TIFF files smoothly.

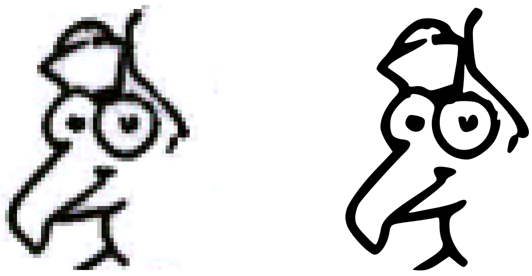


Figure 4: Zoomed view: bitmap (left) converted to vector graphics (right)

Additional tools

There are many other helpful tools. I will mention two I use quite often.

overpic is a LaTeX package written by Rolf Niepraschk [5]. It includes an image into a LaTeX picture environment, giving you the opportunity to add new elements into the image with normal LaTeX picture commands. Fig. 3 shows a map overlaid with symbols and text at some points. The source code for this picture looks like

```
\usepackage[abs]{overpic}
...
\begin{document}
\begin{overpic}[grid,tics=5]{map}
\put(32,74){\includegraphics[scale=.3]
    {busstop.mps}}
\put(32,77){\llap{\scriptsize
    \colorbox{back}{Windm\"uhle}}}
\put(28,63){\small\textcolor{red}{%
    \ding{55}}}
...
\put(17.5,11){\scriptsize\colorbox{back}%
    {{\Pisymbol{ftsy}{65} Fr}}}
\put(6.3,13){\colorbox{back}%
    {{\Pisymbol{ftsy}{68}}}}
\put(29.8,61.4){\color{blue}\vector(-1,-3){2}}
\put(38.6,63){\color{blue}\vector(1,3){2}}
\end{overpic}
\end{document}
```

potrace is a tool to convert a pure black and white bitmap to vector graphics [6]. Fig. 4 shows a sample bitmap converted to a vector image.

References

[1] CTAN:macros/latex/required/graphics
[2] http://www.imagemagick.org
[3] http://netpbm.sourceforge.net
[4] CTAN:support/pstools/tif2eps
[5] CTAN:macros/latex/contrib/overpic
[6] http://potrace.sourceforge.net

Converting METAFONT Sources to Outline Fonts Using METAPOST

Karel Píška
Institute of Physics, Academy of Sciences
182 21 Prague
Czech Republic
piska@fzu.cz
http://www-hep.fzu.cz/~piska/

Abstract

The paper describes a multistep conversion process from METAFONT sources to outline fonts (Adobe Type 1 format). An important step, finding contours, is based on an accurate algorithm fitting the envelope curve of a stroke drawn by a pen along a cubic Bézier curve by the least square method, specially extended (adapted) for a rotated elliptical pen applied, for instance, in the Devanagari font design. After converting the EPS files produced by METAPOST to the corresponding outline representation the FontForge font editor is used for removing overlap, simplification, autohinting, generating outline fonts, and necessary manual modifications. The result of conversion, the faithful Indic Type 1 fonts (significantly close, precise and optimal than earlier attempts made by autotracing bitmaps) will be released.

Keywords: font conversion, bitmap fonts, METAFONT, METAPOST, outline fonts, PostScript, Type 1 fonts, approximation, Bézier curves.

Introduction

In 2001 I experimented with approximate conversion METAFONT Indic fonts to the Type 1 format by autotracing bitmaps with the TEXtrace program [11]. I was not satisfied with results and decided to apply another, analytic approach, to achieve results more precise and also more optimized.

Conversion Process

A procedure consists of study of font definitions in METAFONTand preparing encoding files; then the glyph strokes produced by METAPOST are converted to outlines, the font is assembled, optimized, autohinted, and finally, generated as a Type 1 binary file with FontForge. After verification of visual proofsheet pages some steps are often repeated to correct or improve the final results.

Analysis of METAFONT sources We analyze the METAFONT source texts [7] of a font to select an appropriate strategy of conversion, to find the crucial parameters, like the font size, the italic angle, definitions of pens and strokes. Some parameters may be also hidden inside macros. Sometimes, a possibility of an efficient conversion is not apparent. Therefore it is also important to know about presence and quantity of METAFONT commands not available in METAPOST([5]), for example, using operations with bitmap picture variables.

Creating encoding files Encoding files and encoding vectors define a mapping between the glyph names and their number codes. METAFONT definitions usually do not contain unique glyph names in an explicit form but only comments. The glyph names have been taken from these comments to produce unambiguous list of PostScript names, i.e. we must to find the same names and to change them to be different. Our preliminary solution inherits METAFONT comments closely to make finding glyph identification easier.

Running METAPOST Invoking METAPOST processes the METAFONT sources and produces the EPS files. METAPOST together with a macro package `mfplain` ([5], p. 79) allows to process the original or modified (to eliminate METAFONT-specific commands) font sources written in METAFONT and to generate for each glyph a single file in the Encapsulated PostScript format, consisting only of PostScript commands like curves, strokes, affine transformations representing pens, etc., but no bitmap images contradictory to the METAFONT standard output. Some metric data, e.g. the glyph widths

Figure 1: Result of METAPOST.

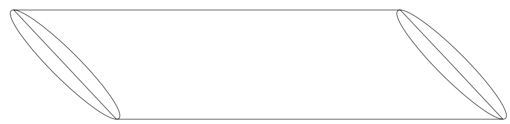


Figure 2: Primary conversion to outlines.

and also the italic angles, may be lost, we shall restore them later. We need also define a magnification factor. Because we have to transform the glyph images to a 1000-unit glyph coordinate system (we use this usual space) with the units in PostScripts *big* points (the transformation factor is 1.00375) and the font *designsize* in *pt* units. Then the magnification factor will be $1000 * 1.00375/designsize$. For the $designsize = 10\,pt$ it equals $1000 * 1.00375/10 = 100.375$, for $8\,pt$ 125.46875, for $17.28\,pt$ 58.087384, etc. Then a typical command to call METAPOST is:

```
mpost '&mfplain \mode=localfont;' \
  mag=100.375'; input' dvng10.mf
```

These files may contain various stroked paths (see figures 1, 9). It is necessary to find contour curves for single strokes and then also common envelope curves for overlapping strokes.

The following lines from the PostScript produced by METAPOST correspond to fig. 1:

```
 0 79.06227 dtransform truncate idtransform
 setlinewidth pop [] 0 setdash
 1 setlinecap 1 setlinejoin 10 setmiterlimit
gsave newpath 119.50958 284.54501 moveto
398.36119 284.54501 lineto
[-0.98387 0.98387 -0.17888 -0.17888 0 0] concat
stroke grestore
```

The `lineto` operator describes the line segment, the `concat` operator applies the affine transformation represented by the preceding normalized matrix (in brackets) denoting the rotated elliptical pen, and 79.06227 . . . `setlinewidth` is the scale factor defining the stroke width.

Converting METAPOST products to outlines The results of METAPOST (strokes) are converted to "primary" outlines. To fit curves with the least square method is a typical approach to calculate a curve approximation. This method is nothing new and probably it has been used in conversion programs developed by Richard Kinch (MetaFog, [6]), Basil Malyshev [9], George Williams (FontForge, [13]) and other. We only apply a few additional conditions. We try to be more precise, but our attempts are still more fragile and unstable than programs listed above.

All the calculations are in the non-integer value space. We check each segment for accuracy and subdivide it if a chosen limit exceed; insert all horizontal and vertical extrema nodes; keep all horizontal/vertical straight lines and control vectors to be exactly horizontal/vertical. The inner part of a contour curve of drawing a rotated elliptical pen even along a simple Bézier path without any intersection may have selfintersections. Therefore we try to find a selfintersection points if it is possible and as precise as possible. Unfortunately, sometimes this iteration does not converge. A simplest conversion to outlines shows figure 2.

For a given time of the path segment using the affine transformation matrix and its inverse matrix (for a usual pen they are always regular) we can calculate the displacement corresponding to the point lying on the right parallel outline curve (the left one is located symmetrically). Knowing the coordinates of points on the outline curves and also on the pen boundary we can fit them by a cubic Bézier approximation. But a problem is we do not know whether the points are an the envelope curve or not because parts of the outline curves may create loops of arbitrary size being inside a closed area. It depends on complex correlations between the path and the pen.

We also recognize quarter-circles usually represented in METAFONT by two segments because METAFONT tends to divide curves to octants. To avoid further simplification problems we do not preserve the 45 degree middle nodes and change the quarter-circles to the accurate single-segment PostScript representation with relative lengths of control vectors $4/3(\sqrt{2} - 1) \simeq 0.552285$, compare also with R. Kinch [6] (p. 236) or Luc Devroye [2]. For an example of our approximation circles see figure 3.

In summary, in the primary approximation the straight lines and the circles are represented by the minimal number of segments (because other nodes are unnecessary), and, on the other hand, other outline curves have redundant node points (to preserve a maximal starting accuracy). The intermediate results of the primary conversion to outline demonstrate figures 2 and 10.

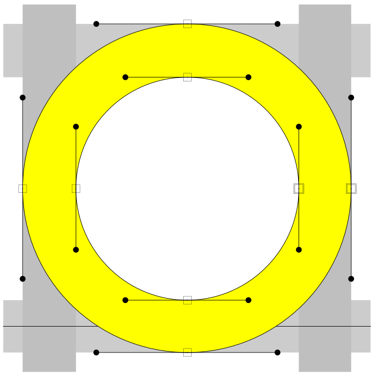


Figure 3: Representation of circles.

Creating a font with FontForge FontForge is a powerful open source font editor. Among its wide range of useful abilities we can find a background layer. It may contain bitmap images and line drawings. Therefore, we generate by METAFONT a high resolution bitmap 7254 *dpi* or 2400 *dpi* (supre) for a given font. The "7254 dpi" device corresponds to a relation 1 pixel in PK $\sim$ 1 unit in the PS glyph space for the 10 *pt*

```
% (72.27*1000.375/10dpi)=7254.1
mode_param (pixels_per_inch,4000+3254.1);
mode_param (blacker, 0);
mode_param (fillin, 0);
mode_param (o_correction, 1);
```

Sometimes, METAFONT with the very high resolution may fail (if the author did not design a font for an arbitrary resolution). The the PK or GF files can be imported to the background as a set of gray pixels to cover glyph images.

Font composition We also run mftrace [10] with an appropriate encoding to make a PFB font file. From this file we build a frame for the created font, copy the glyph widths and the glyph names and move the outlines to the background layer (visible as green lines). During a subsequent processing of the font with FontForge we use its internal Spline Font Database format (SFD). The high resolution bitmap is always huge, we import it only before a comparison. But the outline contours of the font produced by mftrace are not large and we can store them in the working SFD files permanently. To the foreground layer we import the outlines from the EPS files calculated in the previous step from the original EPS files generated by METAPOST.

The high resolution pixel image gives a close visual bitmap representation of the original METAFONT source. Of course, an information about contour curves, intersection points, corners, etc., virtually calculated by METAFONT has been lost. The font outlines autotraced by mftrace from similar bitmaps, despite of the artifacts (bumps, holes, unrecognized corners, ...) give a correct information about glyphs. And our aim is to obtain another outline representation: more accurate and more optimal, to minimize the number of defects and a space amount.

Having a font in the SFD format built from the mftrace output our next step with FontForge is **removing overlap and optimization** (simplification). We continue processing in the non-integer value space to keep accuracy, especially do not change the slopes of the neighbor control vectors to preserve smooth transition between segments.

Rounding to integer, hinting and Type 1 font generation FontForge allows generating PostScript fonts with non-integer point coordinates and, maybe, many PostScript RIP devices render these fonts properly. But we have three significant reasons to round coordinates to integer and to generate the Type 1 fonts in integer representation:

- Non-integer values in the PostScript charstring occupy 3 items. Therefore the integer representation saves storage and the PFB files are smaller.
- The final Type 1 fonts do not need such accuracy after removing overlap and simplification.
- For hinting it would be inconvenient and impracticle to use a different discrete grid than integer.

In the following example the non-integer Type 1 command occupies 19 items:

```
18153 100 div 212 100 div
14437 100 div -407 100 div
7208 100 div -243 100 div
rrcurveto
```

and after rounding only 7 items:

```
182 2 144 -4 72 -2 rrcurveto
```

It is reasonable to minimize the number of items because the PostScript interpreters have internal memory limits per glyph. Exceeding limits causes a limicheck error and a crash of rendering.

The coordinates of the segments are rounded to integer by more complex algorithm than a trivial rounding of all the values. First we round the node points. Then we transform the control vectors according the changes of then nodes and try to find the control points in the integer grid near the transformed control vectors. Even this sophisticated

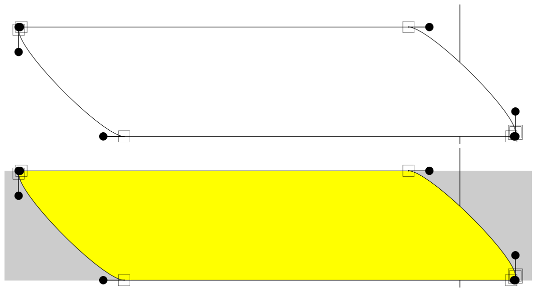


Figure 4: Final font in an outline form and a hinted proofsheet (clip).

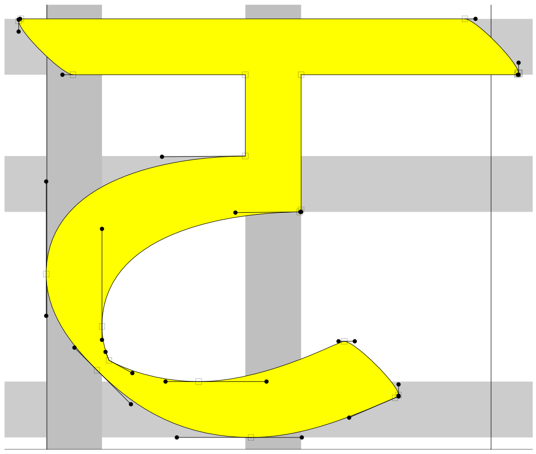


Figure 5: dvng10: tta of Frans Velthuis.

rounding to integer is not without problems. Sometimes, if the change in x or y in the segment is very small (e.g. about 1 unit) or a segment is too short (in both directions) no good selection may exist and a manual adjustment is then necessary, probably with the lose of closeness, accuracy or symmetry of approximation.

No special additional program for hinting have been developed or applied. An automatic autohinting tool of FontForge is used and any unsatisfactory events should be corrected manually.

Finally, FontForge generates the Type 1 binary font, usually rounded to integer and (auto)hinted.

Results

To make font audit and verification more quick and efficient we developed tools for generation of visual proofsheets in PDF: to allow fast overlook all glyph images, outlines curves with node and control points and vectors, hinting zones, and also to detect some situations like missing nodes at extremes, presence of inflection inside a segment, connection between segments is not smooth, etc., and to append special warning signs. Our aim is to fulfill the *Type 1 conventions* [1]. Therefore we include the extrema nodes (they may be omitted if they are really redundant), exclude other unnecessary node points, preserve smooth connections between the adjacent segments. and also keep the straight lines, corners and arcs after conversion, do not append any false bumps, holes or steps absent in the original METAFONT sources. In some selected figures the node points (squares), the control points (bullets) and the control vectors have been enlarged to be visible in the printed version of the paper. In a real working process they are colored and small as in other proofsheets when we zoom interesting details only if we need to check them.

The crucial and auxiliary algorithms have been under development and adaptations for new fonts and the programs are still written in `awk` or `gawk` [3]. For Type 1 font handling `t1utils` [8] are used.

Several pictures illustrate intermediate and final results of conversion METAFONT fonts to the Type 1 format: figures 2, 4, 10, 11, 15, and 16.

Indic Fonts A basic goal of the work are more precise outline versions of the free METAFONT Indic fonts available from CTAN: Devanagari, Sanskrit, Gurmukhi, Punjabi, Bangla, Sinhala, Malayalam, Telugu, Kannada, Tamil, and Tibetan is also included. During preparing this text not all the present fonts have been converted and also the Oriya fonts are still missing because of they widely use METAFONT bitmap picture commands. Next results are shown in figures 12, 13 (Devanagari), 14 (Malayalam).

Chinese Fonts We have also tried to convert two small single fonts with Chinese signs created in METAFONT: the Hóng-Zì font (128 glyphs) designed by Javier Rodríguez Laguna [12] (version 0.5 of 050323): fig. 7; and china10, one font from the the china2e package [4] containing Chinese calendar symbols produced by Udo Heyl (1997): fig. 8.

Conclusion

In the article we describe a conversion process and shortly discuss some selected problems. Creating *precise* fonts is always difficult, time consuming and never ending work independently of the approach we choose. We plan to verify again all the glyphs to improve hinting and polish the outlines to remove tiny artifacts. It is useful to make the glyph names of the Indic glyphs common for all languages, it is not trivial because the fonts contain many various

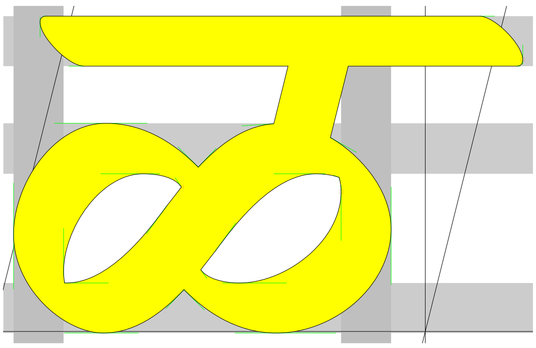


Figure 6: dvngbi10: lla of Frans Velthuis.

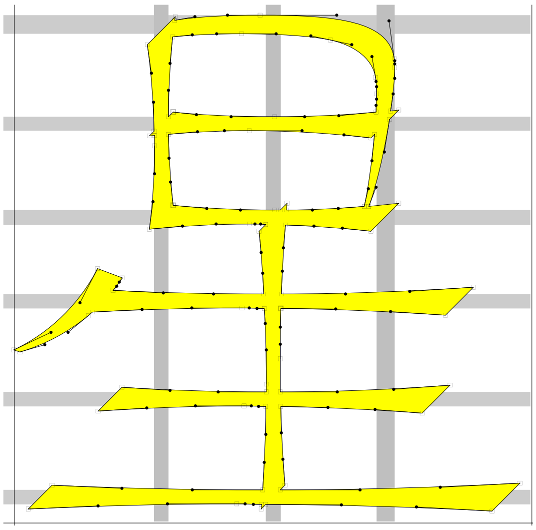


Figure 7: Hóng-Zì: xing1 of Javier Rodríguez.

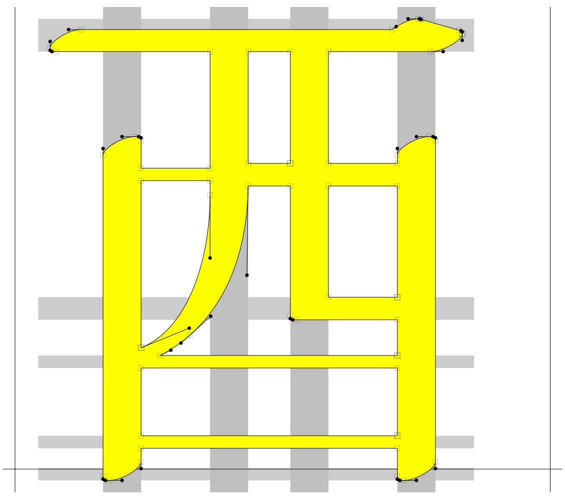


Figure 8: china10: yeu of Udo Heyl.

ligatures, special signs or variants not covered in the Unicode standards.

Acknowledgements

I would like to thank all the authors of the free conversion programs, the authors of the public METAFONT fonts for Indic languages, other sources and program packages used in the contribution,

References

[1] Adobe Systems Inc. *Adobe Type 1 Font Format.* Addison-Wesley, 1990.

[2] Luc Devroye. “Formatting Font Formats”, *TUGboat*, **24**(3), pp. 588–596, 2003.

[3] Free Software Foundation. GNU awk, `http://www.gnu.org/software/gawk`.

[4] Udo Heyl. `CTAN:macros/latex/contrib/china2e`, 1997.

[5] John D. Hobby. A User’s Manual for METAPOST. AT&T Bell Laboratories, Computing Science Technical Report 162, 1994.

[6] Richard J. Kinch. “MetaFog: Converting METAFONT Shapes to Contours”, *TUGboat*, **16**(3), pp. 233–243, 1995.

[7] Donald E. Knuth. *The METAFONTbook.* Addison-Wesley, 1986. Volume C of *Computers and Typesetting.*

[8] Eddie Kohler. t1utils (Type 1 tools), `http://freshmeat.net/projects/t1utils`.

[9] Basil K. Malyshev, “Problems of the conversion of METAFONT fonts to PostScript Type 1”, *TUGboat*, **16**(1), pp. 60–68, 1995.

[10] Han-Wen Nienhuys. mftrace, `http://www.cs.uu.nl/~hanwen/mftrace`.

[11] Karel Píška. “A conversion of public Indic fonts from METAFONT into Type 1 format with TEXtrace.” *TUGboat*, **23**(1), pp. 70–73, 2002.

[12] Javier Rodríguez Laguna. Hong-Zi – A Chinese METAFONT. `http://hongzi.sourceforge.net`, 2005.

[13] George Williams. FontForge: A PostScript Font Editor, `http://fontforge.sourceforge.net`.

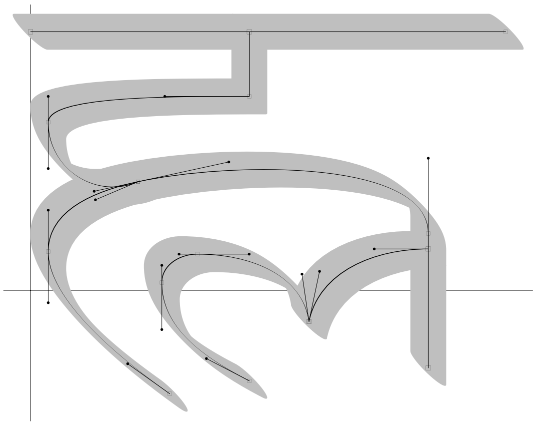


Figure 9: dvng10 l_h: METAPOST output.

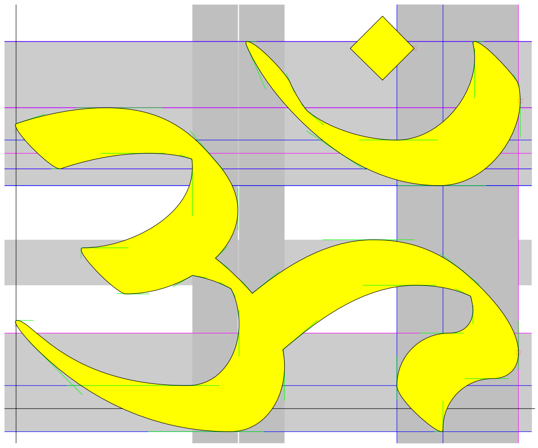


Figure 12: dvng10: om of Frans Velthuis..

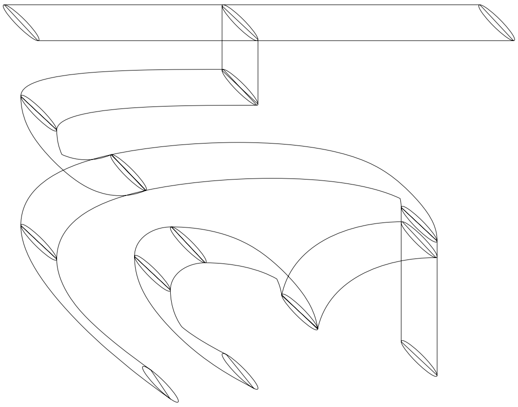


Figure 10: dvng10 l_h: primary outlines.

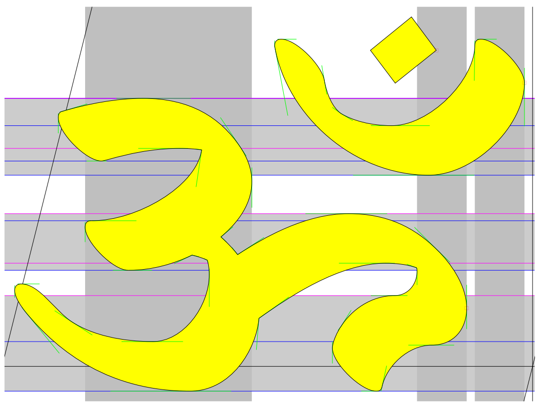


Figure 13: dvngbi10: om of Frans Velthuis.

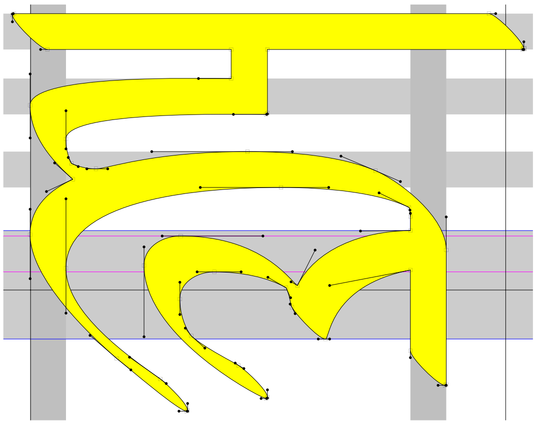


Figure 11: dvng10 l_h: Type 1 font proofsheet.

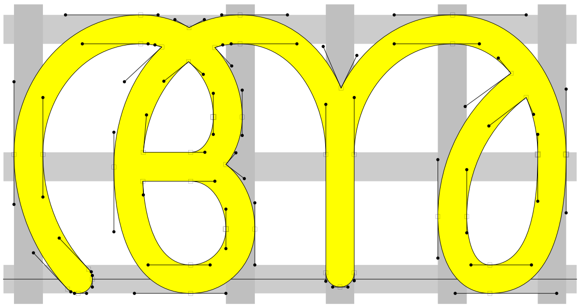


Figure 14: mm10: a of Jeroen Hellingman.

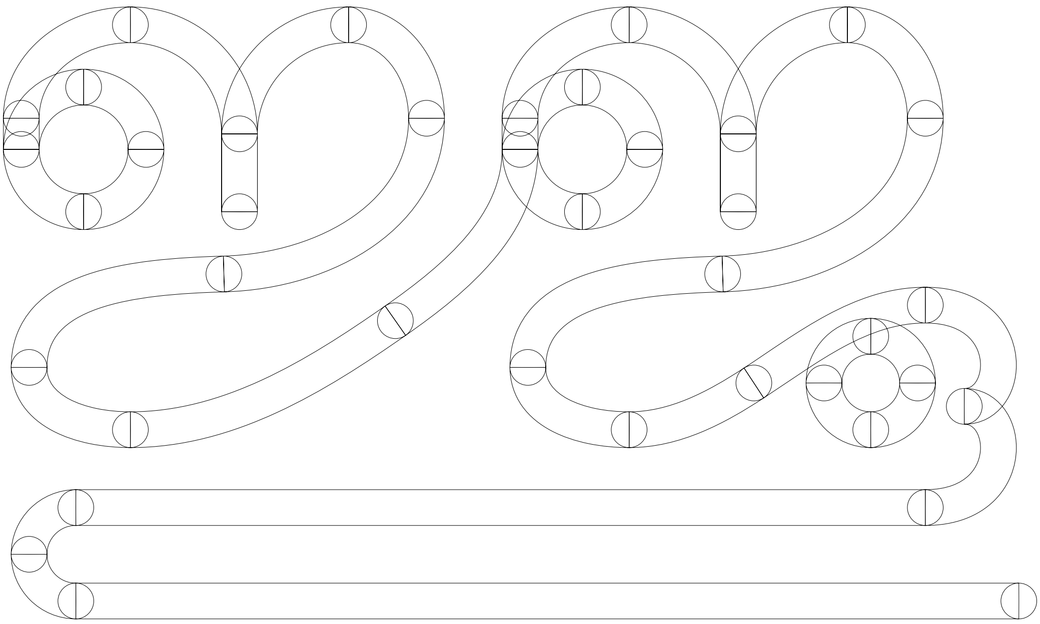


Figure 15: mm10 j_juu: METAPOST output converted to primary outlines.

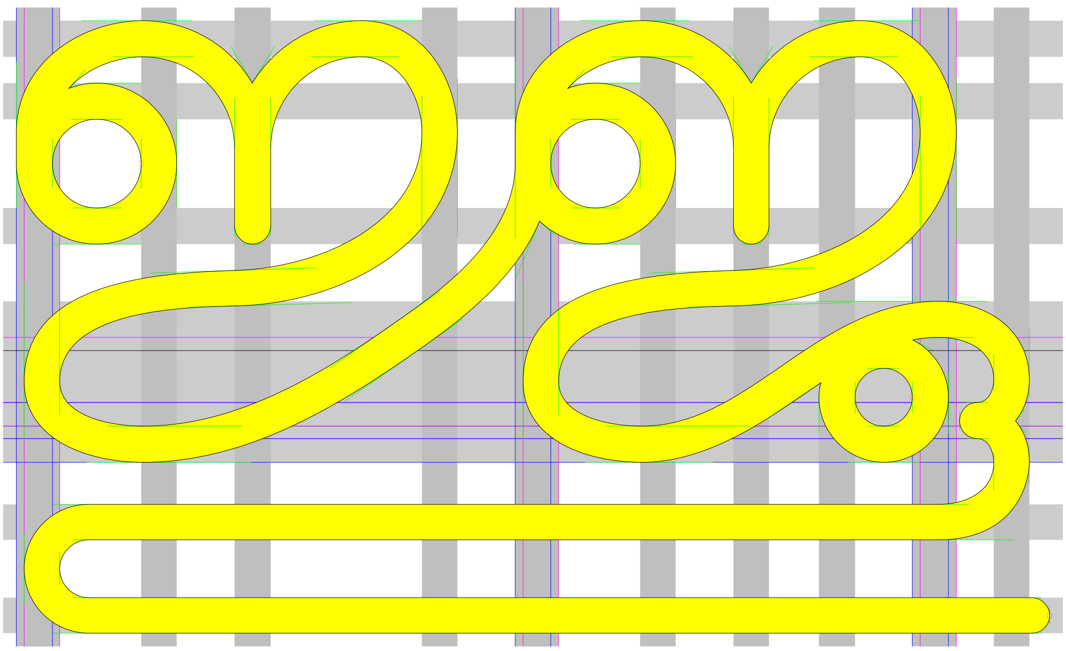


Figure 16: mm10 j_juu: Type 1 font proofsheet with hints.

XML Workflows and the EuroTeX 2005 Proceedings

Volker RW Schaa

DANTE e.V.
Heidelberg, Germany

TUG 2005
Wuhan
China P.R.
August, 2005

Motivation

- In 2002 I was appointed as Proceedings Editor for two conferences
- Responsibility for preparation of abstract and paper submissions, web presentations, the conference volume (*Proceedings*), and CD-ROM
- When I found that the scale of work for these conferences was too large for manual production:
  - DIPAC2003 85 papers, 300 pages, 290 authors
  - LINAC2004 280 papers, 1000 pages, ~1000 authors
- and that even bigger conferences like
  - EPAC 1200 papers, 4000 pages, 3500 authors
  - PAC 1400 papers, 4800 pages, 4300 authors
- were using inadequate tools,
- I decided to write scripts which could do the job

Starting point

the idea
- typesetting: pdfTeX
- scripting: Perl
- data: XML

the method
- database export in XML
- interpretation of XML by Perl scripts
- transformation to <html> and \pdfTeX

≈3 years later:
- the scripts have been used on 5 conferences,
- they have been extended,
- they are now integral part of the conference software under GPL,
- I given several talks about the software (i.e. PracticalTeX 2004)
- but, I never thought of using it for a TeX conference. . .

New directions, differences

- GUTenberg and DANTE organized EuroTeX 2005 in Pont-à-Mousson/France (March, 7–11)
- 2 weeks before the conference the organizers (we) hadn't make up their mind about preprints
- I volunteered to do it (having my scripts in mind),
- then I realized I had to simplify them (sigh).
- So what are the differences between particle physics and TeX conferences in input and output?

XML Definition for Particle Physics Conferences

```
<conference>
  <session>
    <session data, times, location, ...., .../>
    <chair/>
    <chair person's data, .../>
    <paper>
      <paper data, grants, funding, .../>
      <title/>
      <abstract/>
      <institute>
         <institute data, country, name, ..., ..., .../>
         <author>
             <author data, notes, leave of absence, .../>
         </author>
         (more »authors«)
      </institute>
      (more »institutes«)
      <keywords/>
    </paper>
    (more »papers«)
  </session>
  (more »sessions«)
</conference>
```


New XML Definition for TeX Conferences

```
<conference>
  <session>
    <paper>
       <title/>
       <abstract/>
       <author/>
       (more »authors«)
    </paper>
    (more »papers«)
  </session>
  (more »sessions«)
</conference>
```

XML Definition for a single paper Particle Physics Conferences

```
- <paper>
    <code>THZCH03</code>
    <pages>4</pages>
    <toc>249</toc>
    <main_class>Opening, Closing and Special Presentations</main_class>
    <sub_class>Special Presentation</sub_class>
    <presentation type="Oral" option="Invited Oral Presentation">Invited Oral Presentation</presentation>
    <dot>Green</dot>
  - <title>
      JACoW, a Collaboration Serving the Accelerator Community
    </title>
  + <abstract></abstract>
  + <video_URL></video_URL>
  - <contributors>
    - <contributor type="Author">
        <lname>Poole</lname>
        <fname>John</fname>
        <iname>J.</iname>
      - <institutions>
        - <institute>
          + <full_name abbrev="CERN"></full_name>
            <name1>European Organization for Nuclear Research</name1>
            <department>AB Department</department>
            <URL>http://www.cern.ch</URL>
            <town>Geneva</town>
            <postal_code>1211</postal_code>
            <zip_code>23</zip_code>
            <country_code abbrev="CH">Switzerland</country_code>
          </institute>
        </institutions>
      - <emails>
          <email>john.poole@cern.ch</email>
        </emails>
      </contributor>
    + <contributor type="Co-Author"></contributor>
    + <contributor type="Speaker"></contributor>
    </contributors>
  - <files>
    + <file></file>
    + <file></file>
    + <file></file>
    </files>
  </paper>
```


XML Definition for a single paper in TEX Conferences

```
- <paper code="MOT02" abstract="yes" pages="12">
    <title>Omega Becomes a Sign Processor</title>
  - <author main="yes">
    - <name>
        <initials>Yannis</initials>
        <lastname>Haralambous</lastname>
      </name>
    </author>
  - <author>
    - <name>
        <initials>Gábor</initials>
        <lastname>Bella</lastname>
      </name>
    </author>
  </paper>
```


more differences

Particle Physics Conferences

- Abstracts only – Abstract booklet before conference
- Proceedings after conference
  - 1–3 years using old methods (Word, Quark, VB scripts, ...)
  - now: 1 week on the web
  - <9 months on paper (mostly due to waiting for *special* authors)
- CD-ROM (due to the size of proceedings the trend is CD only)

TEX Conferences

- Abstract (always)
- Papers (>60%) before conference (⇒ *Preprints*)
- Proceedings volume with all paper up to now only by TUGboat
- no CD-ROM

What's similar or the same?

Contents setup in LATEX terms

`\frontmatter`

- Conference details
- Committees
- Time table
- Table of contents

`\mainmatter`

- Papers (generated automatically)

`\backmatter`

- Authors
- Participants
- Sponsors, vendors, exhibitors, ...
- Production notes

What is the script doing?

1. It reads a configuration files with specifications, what to do and where to put files,
2. reads XML and generates `<html>` for
   - Session list,
   - Authors list,
3. generates \pdfTEX wrappers
   - for each single (raw) pdf-file,
   - for proceedings file,
4. writes command files for
   - generating pdf-files with author and title information,
   - building of proceedings file(s).

pdfTEX: complete code for one paper

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```

»geometry« helps to keep the tight frame

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```


»fancyhdr« prints header and footer information

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```


»pdfinfo« transfers all meta info into the pdf file

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```


»pdfpages« imbeds the (raw) paper

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```


»\IfFileExists« ensures that there is at least a paper with a note

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```


»pagenumber« is set after checking/counting all pages

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```

»path« information are set in the config file

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```

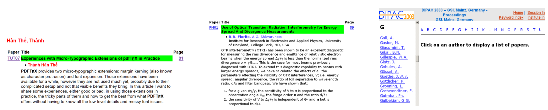

»scaling« is determined by maximum of crop/media-box sizes

```
\documentclass[twoside]{book}
\usepackage[papersize={595pt,792pt}, body={483pt, 680pt},
            top=54pt, left=56pt, head=18pt, headsep=15pt, footskip=32pt]{geometry}
\usepackage{fancyhdr}\pagestyle{fancy}
\usepackage{pdfpages}

\begin{document}
\pdfinfo{%
  /Title    (Omega Becomes a Sign Processor)
  /Author   (Yannis Haralambous, Gábor Bella)
  /Subject  (Preprints EuroTEX2005 -- Pont-à-Mousson, France)
}
\setcounter{page}{8}
\fancyhead[LE,RO]{\large\sffamily Preprints EuroTEX2005 -- Pont-à-Mousson, France}%
\fancyhead[RE,LO]{\large\sffamily MOT02}%
\fancyfoot[RO,LE]{\large\sffamily\thepage}%
\fancyfoot[RE,LO]{\large\sffamily Omega Becomes a Sign Processor\\Yannis Haralambous, Gábor Bella}

\IfFileExists{../papers-final/MOT02.pdf}{%
              \includepdf[pages=-, scale=1.0,
                          pagecommand={}]{../papers-final/MOT02.pdf}}%
              {\Huge\mbox{}\vfill
               \centering\textsf{\textbf{PAPER NOT YET RECEIVED}}
               \vfill}
\end{document}
```


<html>features</html>

Built-in features:

1. Web pages and proceedings honor special characters,
2. Web pages are in Unicode (UTF8),
3. All names get proper accented characters and umlauts,
4. Proper math characters (in abstracts) on web pages,
5. Rule based sorting of names (accented letters, umlauts, ...)



<html>feature="Accented Characters"</html>

Hàn Thế, Thành

Paper	Title	Page
TUT07	**Experiences with Micro-Typographic Extensions of pdfTEX in Practice**	81

- **Thành Hàn Thế**

PDFTEX provides two micro-typographic extensions: margin kerning (also known as character protrusion) and font expansion. Those extensions have been available for a while, however they are not used much yet, probably due to their complicated setup and not that visible benefits they bring. In this article I want to share some experiences, either good or bad, in using those extensions in practice, the tricky parts of them and how to get the best from what **PDFTEX** offers without having to know all the low-level details and messy font issues.

<html>feature="Math"</html>

Paper	Title	Page
PM01	**Use of Optical Transition Radiation Interferometry for Energy Spread And Divergence Measurements**	89

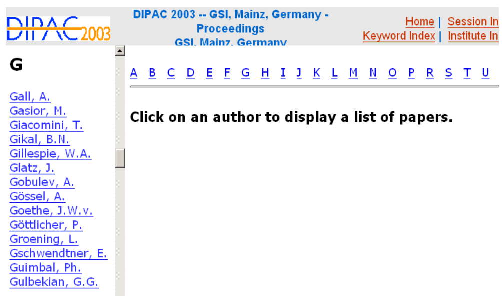
- **R.B. Fiorito**, A.G. Shkvarunets
  IREAP, Institute for Research in Electronics and Applied Physics, University of Maryland, College Park, MD, USA

OTR interferometry (OTRI) has been shown to be an excellent diagnostic for measuring the rms divergence and emittance of relativistic electron beams when the energy spread $\Delta\gamma/\gamma$ is less than the normalized rms divergence $\sigma = \gamma\Theta_{rms}$. This is the case for most beams previously diagnosed with OTRI. To extend this diagnostic capability to beams with larger energy spreads, we have calculated the effects of all the parameters effecting the visibility of OTR interferences, V; i.e. energy spread, angular divergence, the ratio of foil separation to wavelength ratio, d/λ and filter bandpass. We have shown that:

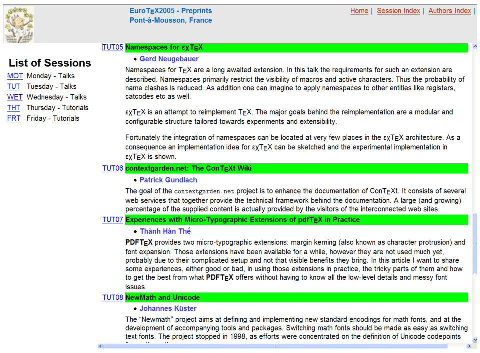
1. for a given $\Delta\gamma/\gamma$, the sensitivity of V to σ is proportional to the observation angle Θ_0, the fringe order **n** and the ratio d/λ;
2. the sensitivity of V to $\Delta\gamma/\gamma$ is independent of Θ_0 and **n** but is proportional to d/λ.

Thus, by adjusting d/λ, and choosing the appropriate fringe order, one can separate out and measure both the energy spread and divergence. However, the filter bandpass must decrease with Θ_0 and **n**. Results of our calculations will be given for various beams of interest.

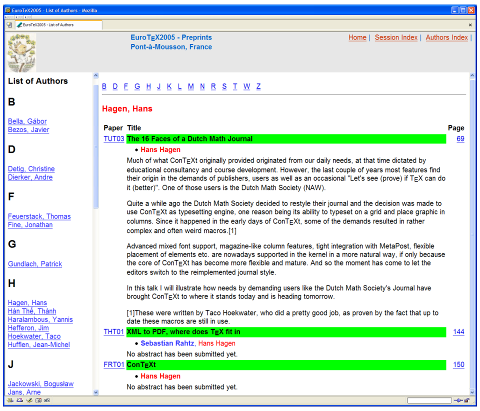
<html>feature="Sorting Order" (i.e. ö ⟺ oe)</html>



<html>feature="Web Session page"</html>



<html>feature="Web Authors page"</html>



\pdfTEX{features}

Built-in features:

1. printing of header and footer information,
2. transfer of all meta-information into pdf-file,
3. (down)scaling depending on size of `crop`/`media`-box,
4. setting of page numbers after counting of all pages,
5. the author index has links to articles,
6. inclusion of paper or "missing" note,
7. `config` file with settings for directories, sort-rules, dependencies etc.

\pdfTEX{feature="Time Table"}

Preprints EuroTEX2005 – Pont-à-Mousson, France

11:00	TUT03	Hans Hagen The 16 Faces of a Dutch Math Journal
11:45	TUT04	Adam Twardoch Typographic Perfection with OpenType?
12:30 – 14:00	Lunch	
14:00	TUT05	Gerd Neugebauer Namespaces for εχTEX
14:30	TUT06	Patrick Gundlach contextgarden.net: The ConTEXt Wiki
15:00	TUT07	Thành Hàn Thế Experiences with Micro-Typographic Extensions of pdfTEX in Practice
15:30 – 16:00	Coffee Break	
16:00	TUT08	Johannes Küster NewMath and Unicode
16:30	TUT09	Bogusław Jackowski, Janusz M. Nowacki Latin Modern fonts: how less means more
17:00 – 19:00	TUT10	Panel discussion with Hermann Zapf and Donald Knuth 'With a little help from the wizards'
20:00	Gala Diner	

Wednesday, March 9

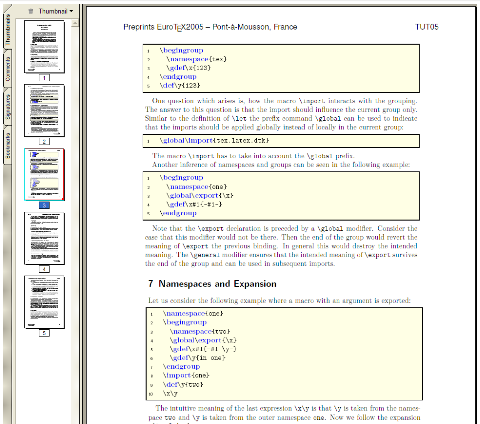
8:30	WET01	Thomas Feuerstack ProTEXt, a new TEX-Collection for Beginners
9:00	WET02	Jean-Michel Hufflen Bibliography Styles Easier with MlBibTEX

\pdfTEX{feature="Table of Contents"}

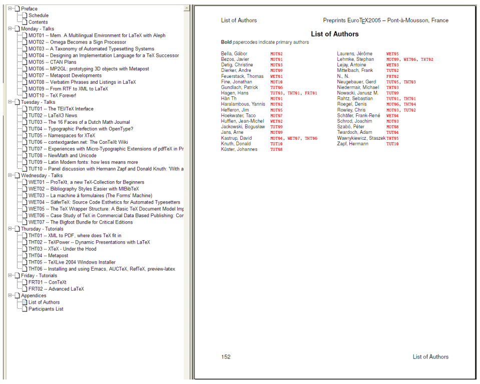
Preprints EuroTEX2005 – Pont-à-Mousson, France

Contents

\pdfTEX{feature="Preprints"}



\pdfTEX{feature="List of Authors"}



History
<html>/pdfTEX
Summary
Scripts
Generated script for pdf-file
«html>features</html>
\pdfTEX{features}

\pdfTEX{feature="List of Participants"}

Participants List — Preprints EuroTEX2005 – Pont-à-Mousson, France

Karin **Dornacher**
DANTE e.V.
`office@dante.de`
Postfach 101840
D-69008 Heidelberg
Germany

— E —

Martin **Etter**
`martin.etter@gmx.de`
Bergstraße 5
D-70806 Kornwestheim
Germany

Christoph **Eyrich**
`eyrich@math.tu-berlin.de`
Skalitzer Straße 74a
D-10997 Berlin
Germany

— F —

Robin **Fairbairns**
`rf@cl.cam.ac.uk`
30 Mill End Road
Cambridge, CB1 9JP
United Kingdom

Hong **Feng**
RON's Datacom Co., Ltd.
`fred@mail.rons.net.cn`
Suite 3-3, WuZhong Str. 200, Don District
430040 Wuhan
China

Robert **Fischer**
`derfischer@gmx.net`
Am Krümmelweg 8
D-54311 Trierweiler
Germany

Daniel **Flipo**
U.S.T.L.
`daniel.flipo@univ-lille1.fr`
Cité scientifique
F-59655 Villeneuve d'Ascq Cedex
France

David **Fuchs**
`drfuchs@yahoo.com`
1775 Newell Rd.
Palo Alto, California, 94303
USA

— G —

Ralf **Gärtner**
`ralf.gaertner@t-systems.com`
Ötztalerstr. 5b
D-81373 München
Germany

Falk **Gerwig**
`flak2k@gmx.de`
Im Schlehbusch 9
D-75397 Simmozheim
Germany

Michel **Goossens**
CERN
`michel.goossens@cern.ch`
Departement IT

Patrick **Gundlach**
`patrick@gundla.ch`
Universitätsstraße 71
D-44789 Bochum
Germany

Michael **Guravage**
Literate Solutions
`guravage@literatesolutions.com`
Mijndensedijk 11a
NL-3632NT Loenen aan de Vecht
The Netherlands

— H —

Hans **Hagen**
PRAGMA
Advanced Document Engineering
`pragma@wxs.nl`
Ridderstraat 27
NL-8061GH Hasselt
The Netherlands

Thê Thành **Hàn**
University of Education
in Ho Chi Minh City
`hanthethanh@gmx.net`
280 An Duong Vuong
Ho Chi Minh
Vietnam

Yannis **Haralambous**
ENST Bretagne
`yannis.haralambous@enst-bretagne.fr`
CS 83818
F-29238 Brest
France

Volker RW Schaa · XML Workflows and the EuroTEX 2005 Proceedings

History
<html>/pdfTEX
Summary

Summary

- The whole exercise was to show that with pdfTEX and the help of Perl we have all means to put proceedings and preprints together in an easy way with a convincing quality in print.
- What's left to do:
  - translation of special characters to Unicode has to be extended
  - actually there are 65 accented characters,
  - 117 special characters,
  - 113 math symbols, and
  - 39 Greek letters.
  - at least one Vietnamese `:-)`

Volker RW Schaa · XML Workflows and the EuroTEX 2005 Proceedings

History
<html>/pdfTEX
Summary

Thank you!

Volker RW Schaa · XML Workflows and the EuroTEX 2005 Proceedings

The Porphyrogenitus Project :

— Typesetting the Byzantine *Cappelli*

1

Background :

- Latin palæographers have for over a century been able to refer to Adriano Cappelli's *Lexicon abbreviaturarum. Dizionario di abbreviature latine ed italiane* (Milan: Ulrich Hoepli)
- No similar work exists for Byzantine (early Greek) texts
- This lacuna is being filled by the work of Miss Julian Chrysostomides and Dr Charalambos Dendrinos

2

Purpose of the project :

- To collate, transcribe and document several thousand examples of Greek palæography ...
- To present the results of this research in printed form ...
- And, ultimately, to make it available on multi-indexed, searchable, zoomable CD

3

The problem :

- Thousands of pages of handwritten Byzantine text dating back over 1000 years
- Text full of abbreviations, ligatures, and other scribal devices which make them virtually inaccessible to all but the most dedicated scholars

4

The solution :

- A "Lexicon of Abbreviations & Ligatures in Greek Minuscule Hands (ca. 8th century to ca. 1600)"
- Each entry to contain a scanned image with transcription, transliteration and provenance

5

Macrostructure of the Dictionary :

- The alphabet (variant forms of each letter)
- Abbreviations
- Ligatures
- Incunabulæ (early printed books)
- Tachygraphy (speed-writing, "shorthand")
- Monocondyliæ (a word or words [frequently signatures] written with one stroke [i.e., the words are not separated])
- Cryptography (secret writing)
- Symbols
- Punctuation
- Numbers

6

Microstructure of the Dictionary :

- Scanned image (normalised for size)
- Transliteration (the exact glyphs used)
- Explanation (the full form, with omitted glyphs interpolated)
- Provenance (usually date, occasionally more)

7

Preparation of entries

- Manuscript read by Charalambos or Julian
- Interesting regions marked and scanned
- Saved as a file, transcribed, and entered into an index system
- The computer does the rest !

8

The original methodology

- Scanned using an HP Scanjet IIc
- Edited using PaintShop PRO and a FastPoint light pen
- Traced using Corel Trace
- Further edited using Corel Draw
- Exported as EPS
- Transcribed onto a CardFile file
- Each entry appended to Byz-Data.dat
- Processed using Eberhard Mattes "emTEX" with Silvio Levy's Greek fonts (PK)
- Converted to PostScript using ArborText's DVILASER/PS
- Viewed using Russell Lang's GS-View
- Proofs produced on a PostScript printer if necessary

9

How we do things in the 21st Century

- Scanned using an HP Scanjet 6430
- Edited using PaintShop PRO and a FastPoint light pen
- Saved as PDF
- Transcribed into an Excel spreadsheet
- Processed using WinEDT, TeX-Live 2003 & Hàn Thế Thành's/Fabrice Popineau's PdfLATEX and Claudio Beccari's Greek fonts (Type-1)
- Viewed using Adobe Acrobat
- Proofs produced on any Windows printer when needed

10

Markup needed for Greek palæography

- The Greek alphabet, transliterated into English characters (52 in all)
- Breathings (rough, smooth)
- Accents (acute, grave, circumflex)
- Iota subscript
- Ornamentations (raised, overbar)
- Diaresis

11

Sorting the data

- Dictionary divided into ten main sections
  - The alphabet (variant forms of each letter)
  - Abbreviations
  - Ligatures
  - Incunabulæ (early printed books)
  - Tachygraphy (speed-writing, "shorthand")
  - Monocondyliæ (words written with one stroke, frequently signatures)
  - Cryptography (secret writing)
  - Symbols
  - Punctuation
  - Numbers

12

Intra-section sorting

- Sort by multiple keys
  - letters
  - breathings
  - accents
  - iotas
  - ornaments
  - diareses
  - cases

13

The problem :

- Files to be sorted are large, with embedded TEX markup
- TEX is good at parsing its own markup, but weak at sorting
- PERL is good at sorting,but weak at parsing TEX markup

14

The solution (with thanks to Prof. Klaus LAGALLY)

- Use TEX to parse the source file
- Use PERL to sort the source file based on the information generated by TEX

15

The implementation

- The TEX parser writes multiple output files
- Each output file represents one key for the PERL sorter
- Each record in each output file contains one fixed-width integer for each lexeme in the input record
- PERL is then asked to perform a detached-key sort using multiple keys of this format

16

A peek inside the files

- A fragment of the raw (unsorted) TEX input
- A fragment of an intermediate key file, written by TEX
- A fragment of sorted TEX output

17

Additional refinements

- Sort first by transliteration, then by explanation if transliteration identical or omitted
- Two additional keys added during refinement :
  - Date (provenance)
  - Original sequence number

18

The programs

- Sort.TeX
- Sort.Perl

19

Optimising the layout

- Scanned images vary in size (width) even after normalising for height
- TEX would require a multi-pass approach to optimise image placement
- Excel has a very powerful add-in function (Tools/Data analysis/Histogram/Cumulative percentage)
- Given the set of all possible widths for images, this will immediately allow the book designer to see what fraction would fit in a given width

20

The implementation

- TEX is asked to output an auxiliary file containing the width of each image encountered
- Excel can easily import such a file into a spreadsheet (use "p" as column separator : $32.10547pt -> 32.10547$ t)
- Using Excel's data analysis tools, these widths are sorted and frequency & cumulative % age ascertained
- The book designer and researchers then jointly look at these to decide how much space to allow for the images

21

Further statistical input to the book design

- Not only image width but transcription width, explanation width and provenance width can be analysed
- For wrappable fields, minimum width can be computed using TEX's box constructor/box destructor methods
- Statistical information such as this can do much to ensure that the author(s) and book designer are able to make informed design decisions

22

Conclusions

- Modern tools such as PdfLaTEX make life much simpler
- TEX is an ideal tool for typesetting polytonic Greek
- Splitting the sorting task into two distinct phases offers enormous benefits
  - TEX is ideal for parsing its own markup but poor for sorting
  - PERL is sub-optimal for parsing TEX markup but perfect for sorting using multiple detached keys
- Excel is a very powerful tool for providing statistical input into the task of book design
- The synthesis of TEX & PERL is a splendid example of synergy, as is the synthesis of TEX & Excel.

23

Principles of Nutritional Assessment: 2nd edition

— 2-column typesetting on a grid using (Pdf)LaTeX2e

1

Background

- Professor Rosalind Gibson's *Principles of Nutritional Assessment* first published by OUP (NY) in 1990
- Publication followed five years of collaboration between author, her husband (Professor Ian Gibson) and self
- Original design was typeset in a single column to a fairly wide measure (6 1/8" x 9 1/4")
- OUP insisted on Times Roman which we were forced to scale anamorphically (by a factor of 24/25) to suit the wide measure
- OUP (unaware of the scaling) pronounced our version "one of the nicest instances of Times Roman we have seen" !

2

Preparations for the second edition

- Ros started work on the 2nd edition about five years ago
- An early design decision was to typeset in two columns
- Having looked at standard LaTeX 2-column output, a secondary design decision was taken to try to enforce typesetting on a grid
- Since we were now working in narrow measure, unscaled Times Roman was suitable for the main text font
- Optima was selected as the font of choice for headings

3

The team members

- Professor Rosalind Gibson is the author, driving force, and ultimate authority on all decisions
- Her husband, Professor Ian Gibson, a geologist by profession, undertook the task of typesetting
- Philip Taylor was technical advisor, as with the previous edition
- OUP (NY) undertook to publish the work from CRC prepared by the team

4

Moving into the 21st Century

- Ian was unaware of TEX-Live and took some persuading before he would willingly migrate to it
- He was similarly reticent about migrating to Pdf(La)TEX
- He is now totally convinced that these migrations were justified!

5

The challenges of typesetting on the grid

- Chapter headings
- Section and subsection headings
- Quotations
- Lists
- Displayed maths
- Figures
- Tables
- \textheight, \baselineskip, \topskip, ...

6

To automate or to kludge ?

- With author & typesetter in New Zealand and technical advisor 12000 miles away, necessary to evolve a working methodology that allowed each to work effectively without requiring immediate feedback from the other
- The task of ensuring grid-based compliance split between typesetter and advisor
- Typesetter would adopt *ad hoc* solutions, whilst advisor would work towards automation of the task
- Figures and tables were left to the typesetter, other challenges were resolved by the advisor

7

Chapter headings

- Chapter headings were set in a zero-depth \vtop
- Space was then left using a \kern of 11 or 13 \normalbaselineskip

8

Section and subsection headings

- All parameters to \@startsection were expressed as integral multiples of \normalbaselineskip
- \@startsection was itself hacked to perform a \vskip of -1 \normalbaselineskip
- \@sect was hacked to ascertain the natural height/depth of the heading and then to replace this with the most appropriate of a small finite set of pre-determined dimensions
- \@sect has some of the worst kludges ever seen, with hard-wired, empirically-determined, real constants !

9

Quotations

- Treated as lists, with \topsep set to 0,5\normalbaselineskip

10

Lists

- Use normal LaTeX lists with \topsep set to 0,5 \normalbaselineskip or 1,0 \normalbaselineskip as appropriate

11

Displayed maths

- Uses \vadjusted nested 0 pt \vtops with a 0 \baselineskip \vskip before and a 1 \baselineskip \vskip after, all within a real displayed maths environment
- Within the display, each line set using \maths {}, which puts its parameter into an \hbox in maths mode

12

\baselineskip

- Fill elements removed using \baselineskip = 1 \baselineskip
- \baselinestretch set to 0,88235
- Base font size is 11 pt, so we end up with something very close to 11/12 (actually 11/12.00002)

13

\textheight

- Set to 50 \baselineskip
- OUP would have preferred 50 pc

14

\topskip

- Set to 1 \baselineskip

15

The final product

- We leave judgement to the reader ...

16

Unforeseen problems (1)

- Section headings were typeset with more space above than below (correct practice), and we were retaining the space above when a heading occurred at the top of a column (so as to have consistent space below)
  - OUP insisted that these headings be set "aligned" with running text in the opposite column
  - They were unable to tell us whether they mean "baseline aligned", "x-height aligned", or "ascender-aligned", so we had to guess . . .

17

Unforeseen problems (2)

- Figures falling at the bottom of a column caused problems, in that the legend (below the figure) was never quite flush with the bottom of the adjacent column — it was always too high
  - We never did get to the bottom of this one, so Ian had to kludge it by hand wherever it occurred . . .

18

Why so many kludges ?

- Two main reasons :
  - Neither TEX nor LaTEX offer intrinsic support for grid-based typesetting
  - The "technical advisor" is a complete beginner when it comes to LaTEX and knows only how to hack "real" TEX, so some things that might have been easy to an experienced LaTEX programmer were pretty d@mned difficult !

19

Conclusions

- For 2-column work, grid-based typesetting *should* be the norm
- TEX & LaTEX offer little in the way of intrinsic support for grid-based typesetting
- Apart from a few fundamental constants, there are about half a dozen different classes of material that require special treatment in order to ensure that a grid-based layout is not violated
- Zero-depth \vtops augmented by \vskips of an integral number of \normalbaselineskips provide a useful tool for some cases
- In other cases, more pragmatic and empirical approaches are appropriate
- Since the benefits of grid-based typesetting are clear and indisputable, it would be worth expending some effort to produce a robust LaTEX-based solution

20

Epilogue

- Hàn Thế Thành has been looking into the possibilities of augmenting PdfTEX to allow grid-based typesetting
- His ideas include new node types and new primitives

21

Possible new node types (1)

- pdf_snap_ref_point_node
  - a whatsit node representing a reference point for snapping
  - no associated data

22

Possible new node types (2)

- pdf_snap_x_node:
  - a whatsit node representing a node that can be snapped in x-direction;
  - has a glue specifying:
    - the basic unit of the "grid"
    - how much it can be moved left/right
  - Example: 5pt plus 4pt minus 3pt – grid of 5pt for each cell, snap nodes can be moved as much as 4pt forward (right) and 3pt backward (left). Any 'fil' or higher order means that the movement amount is unlimited

23

Possible new node types (3)

- pdf_snap_y_node:
  - As pdf_snap_x_node, but for y-direction

24

Possible new primitives

- \pdfsnaprefpoint
  - insert a reference point
- \pdfsnapx
  - insert a whatsit node that can be snapped in x-direction
- \pdfsnapy
  - as \pdfsnapx, but for y-direction
- \pdflinesnapx
  - specify a snap_x node that will be automatically prepended to each line after line-breaking
- \pdflinesnapy
  - as \pdflinesnapx, but for y-direction

25

Example usage

- Let there be \pdfsnaprefpoint in each page (e.g., in the header)
- assume \baselineskip = 10 pt
- then an early declaration such as
  - \pdflinesnapy = 10pt plus 5pt minus 4pt

  would have the effect of snapping every line to a grid with the reference the location of \pdfsnaprefpoint to the nearest multiple of 10pt, given that the movement amount is in range (-4pt, 5pt)
- to snap the reference point of a box, insert a \pdfsnapy somewhere inside the box so that it ends up at the baseline of the box

26

As-yet unresolved problems

- At the moment, \pdflinesnapy causes all lines produced by line-breaking to be snapped
  - This is undesirable for some cases such as listing environments, verbatim, headings and so on. We therefore need a means to turn snapping on and off, or to find another way to snap rather than to apply it to every line
- Snapping can mess up the layout in some case
  - For example, after displayed maths, snapping can cause the next line to move up or down depending on the current setting, which can in turn lead to the case when the space after the displayed maths is not in proportion to the space before the display

27

Interactions with LaTEX

- To apply snapping with success, many LaTEX definitions may have to be rewritten to take it into account. A typical case in point is that before and after every element (an environment, a listing or a heading), there is some glue with both stretchability and shrinkability, and therefore around each element is some elastic space. Snapping allows us to snap lines after such an element to align with the grid again, but it does so by changing the space *after* the element only (by moving the next line up/down).

28

An alternative paradigm

- An alternative possibility now being considered by Thành is the idea of "discrete glue" :
- "Discrete glue" would stretch or shrink like conventional glue, but only by discrete amounts
- Not be be confused with "discreet glue", which is so small that it cannot be seen but which adds enormously to the æsthetics of the page :-)

29