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	^

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;

```
@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.

- 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).

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

```
@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.

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 $\mathcal{NTS}$ 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.