Baskerville

The Annals of the UK TEX Users' Group **Guest Editor: Peter Abbott** **Vol. 6 No. 4**

ISSN 1354–5930 **October 1996**

Articles may be submitted via electronic mail to `baskerville@tex.ac.uk`, or on MSDOS-compatible discs, to Sebastian Rahtz, Elsevier Science Ltd, The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, to whom any correspondence concerning *Baskerville* should also be addressed.

This reprint of *Baskerville* is set in Times Roman, with Computer Modern Typewriter for literal text; the source is archived on CTAN in `usergrps/uktug`.

Back issues from the previous 12 months may be ordered from UKTUG for £2 each; earlier issues are archived on CTAN in `usergrps/uktug`.

Please send UKTUG subscriptions, and book or software orders, to Peter Abbott, 1 Eymore Close, Selly Oak, Birmingham B29 4LB. Fax/telephone: 0121 476 2159. Email enquiries about UKTUG to `uktug-enquiries@tex.ac.uk`.

Contents

1 Editorial

I can almost hear the comments *Late again!* I can only pay tribute to Sebastian who managed to produce so many issues **and on time**.

At the committee meeting where the work was shared out, it was stated that all the style files etc. were available and it was simply a matter of loading them and processing the provided articles. Just write an editorial and use the standard filler to complete the 24 pages.

I have been using TEX and LATEX since 1986 and have been using it on my Mac (an LC475) for some time with only minor problems. Robin Fairbairns kindly sent me the relevant files from *Baskerville* 6.3 and Sebastian Rahtz sent me most of the material. From the starting point it was just like snakes and ladders, as fast as I climbed a ladder there was another snake at the top. I have lost count of the number of times that I have changed the parameters and had to run `initex` besides increasing the memory for OzTEX and DVIPS.

Eventually, several weeks later, I managed to actually produce a printed copy. My knowledge of LATEX has increased considerably but I am sure that my other documents will never approach such complexity.

I can now appreciate all the effort expended by all the guest editors, but am still grateful to have 'had my turn'. I am even more pleased that it is over. I hope it is a long time before it comes around again. Like previous guest editors, I would have been unable to produce this issue without the help, encouragement and patient response to what were probably stupid questions on my part to other committee members. I even had to ask for help in the final stage of production. Perhaps I should reclassify myself as only an intermediate TEX user? Anyway read on and enjoy.

Peter Abbott

2 'Post Editorial'

Peter worked very hard on this issue of Baskerville, but was delayed by the unavailability of my Dubna report. I therefore suggested that he should send his material to me, and that I would finish the job. As you will have observed, it's taken me *far* too long to do that—only a couple of weeks of the lateness of this issue is due to Peter. Those who know me will know that I only did about four things on time last year: I apologise most profusely to the membership that this issue was one of the things I delayed. Robin Fairbairns

I The Joy of TEX2PDF—Acrobatics with an Alternative to DVI Format

Petr Sojka, Han The Thanh, and Jiří Zlatuška
Faculty of Informatics
Masaryk University Brno
Burešova 20, 602 00 Brno
Czech Republic

[*Editor's note:* At the TEX Users Group meeting held in Dubna, Russia, in July/August 1996, a prize in memory of Cathy Booth was awarded to this paper. Delegates to the conference gave it their overwhelming vote when asked to consider which paper would affect their TEX work in the future. Sebastian Rahtz and Robin Fairbairns, representing UKTUG at the meeting, were delighted to keep alive the memory of Cathy Booth with a paper she would have greatly enjoyed.

This paper is reprinted from *TUGboat* 16(3) with kind permission of Barbara Beeton and the authors.]

Summary

This paper presents a discussion about generating Portable Document Format (PDF) directly from TEX source using a prototype TEX2PDF program. This is a derivative made from the TEX source which allows us to bypass DVI output generation, and to produce documents in Adobe PDF directly. Motivations for the TEX2PDF approach are discussed and further possible enhancements are outlined.

1 Motivation

GO FORTH *now*
and create masterpieces
of the publishing art!
Don Knuth [18], p. 303.

General acceptance of TEX for the publishing of technical documents has spread enormously during the last two decades. Since TEX's inception, however, new standards have emerged in the publishing world. SGML and LATEX for markup, POSTSCRIPT and Portable Document Format as page description languages (PDL), are just a few of the buzzwords in the arena. Publishers are moving towards the art of creating *electronic* documents.

TEX's typesetting engine outputs its results in the device independent (DVI) page description format [9, 10]. To avoid duplication, and to be backward compatible, various extensions to the DVI format have been used via the `\special` command. Do you need color? Use color supporting `\special`s. Do you need PostScript fragments in the `dvi` file? Graphics in various formats? PDF fragments in the `dvi` file? Hypertext? Document/object structure markup for an SGML driver? Every new application usually ends up as a new set of `\special`s, which are unfortunately, not yet standardized [26, 24].

Do you need portable object reuse in your `dvi` file? Sound? Portable Multiple Master font parameters? No `\special`s for these are in sight.

As a result of all this, documents in DVI format are not really portable, as they usually contain a lot of `\special`s, and visual appearance depends on the device drivers available at the reader's site. These and similar problems and thoughts have led us to research on the possibility of generating portable electronic documents which will offer widest range of functionality from well established and widely used (LA)TEX sources.

We give an overview of current formats relevant to the electronic document storage, including the current possibilities for producing PDF—a possible format of choice for electronic documents. We suggest a new approach by means of the TEX2PDF program and discuss its merits with respect to other approaches. We conclude with a discussion of object reuse and future developments.

2 Formats for Electronic Document Delivery

Program(s)	without LZW compression	with LZW compression	without compression and PDF file gzipped
TEX2PDF (α-test version)	8 063 658	3 086 545	1 906 184
TEX + dvips 5.58 + Adobe Distiller 2.1	10 530 967	4 387 232	2 115 827
TEX + dvips 5.58 + Aladdin Ghostscript 4.0	16 908 552	not applicable	

Table 1. Size comparison of several ways of producing PDF file (`tex.pdf`) from a TEX file (`tex.tex`)

2.1 *DVI Format*

A `dvi` file is the standard output of a TEX run and is often used as a format for storage and exchange of typeset TEX documents.

DVI format is heavily (but not exclusively) used e.g. in the Los Alamos e-Print archive |http://xxx.lanl.gov/|. Several tens of thousands documents are available (typeset by autoTEXing scripts) from there. The disadvantage is that the documents are not 'self-embedded', which means that they rely on standardisation of font names and availability of fonts at the document consumer's site. Hypertext extensions to the DVI format have been accomplished by a set of HTML-like `\special`s defined by the HyperTEX project (`http://xxx.lanl.gov/hypertex/`) and special versions of previewers (xhdvi), dvihps and ghostscript (ghosthview) have been developed.

2.2 *Portable Document Format*

PDF [5] is a page description language derived from Adobe's PostScript language [2]. The design goals are:

- Rendering speed—algorithmic constructs were removed from the language.
- Portability—as a cross platform format, Acrobat Reader is available free of charge on major platforms.
- Compactness—The Lempel, Ziv, Welsh compressing algorithm was licensed from UNISYS for maximum compression of files.[1] Multiple Master font technology, partial font downloading and built-in fonts in the Acrobat Reader lead to a minimum size for portable documents.
- WWW support—hypertext links to other documents on the Internet are allowed. PDF version 1.2 and Acrobat 3.0 (Amber) introduced a linearized arrangement of objects within PDF documents, allowing for incremental downloading across the Internet.
- Extensibility—documents can be extended without losing the old version; notes (stickers) can be added to document by the readers.
- Password protection—access to a document can be protected by a password.
- Object structure—allows for access to individual pages, with possibility of one-pass generation.
- Easy exchange—ASCII (7bit) PDF files can be generated for better portability and email exchange.

PDF files can be embedded directly in an HTML page using the HTML `<EMBED>` tag [1]. These are becoming more and more popular in the WWW world, as they render faithfully what the author saw (modulo color rendering and resolution of an end-user's display).

2.3 *SGML*

Roll on SGML, *and real document storage.*
Not just this strange PDF *thing*
which traps the visuals like an insect in amber . . .
James Robertson on `comp.text.pdf`

SGML is a widely accepted international standard (ISO 8879) [12, 6, 3] for document markup. It is the format of choice for document storage chosen by many publishers [22, 7, 4]. It is a language for describing markup, aimed at long-term storage, but not at visual layout. As TEX's typesetting engine is still the state-of-the-art, the perspective of typesetting of SGML documents via LATEX3 with TEX based engine is a viable option.

3 Current Possibilities for Producing PDF from TEX

If PDF is required as the end format, with currently available programs one has to generate PostScript from a `dvi` file and then to 'distill' (using Adobe's Distiller program) the result to PDF. Some comments and suggestions on how to create PDF files from TEX are collected in [16]. Problems with configuring fonts are described in [28] and [8].

[1] Latest news from Adobe says that ZIP compression has been added as well, leading to even better compression ratios.

Program(s)	Time without compression	Time with compression (LZW)
TEX2PDF (α-test version)	1:57	2:38
TEX + dvips 5.58 + Adobe Distiller 2.1	6:34 (1:33+0:18+4:43)	6:56 (1:33+0:18+5:05)
TEX + dvips 5.58 + Aladdin Ghostscript 4.0	not applicable	40:23 (1:33+0:18+38:32)

Table 2. Speed comparison of several ways of producing PDF file (`tex.pdf`) from a TEX file (`tex.tex`)

4 The Name of the Game

There still are countless important issues
to be studied, relating especially to the many
classes of documents that go far beyond
what I ever intended TEX to handle.
Don Knuth [20], p. 640

Motivated by a note by Don Knuth to one of the authors (private communication, 1994), who mentioned he expected people would attempt to create derivations from TEX suitable for, e.g., outputting PostScript instead of DVI, a project for creating PDF files directly from the TEX source has been attempted [27], introducing the possibility of creating either DVI or PDF output. The working name of this game is TEX2PDF. An example of the TEX source taking advantage of the new possibilities is shown in figure 1 and the resulting document as viewed with Adobe Acrobat Reader is shown in figure 2 on page 7.

4.1 New primitives

New primitives have been introduced in TEX2PDF in order to allow for more straightforward use of hypertext features from within TEX-like source. Most of their parameters are taken implicitly from the context of use in TEX terms, which simplifies their use considerably. We do not specify the full syntax here, because it is not yet fully stable.

`\pdfoutput` changes TEX2PDF behaviour from DVI-producing mode to PDF-producing one.

`\pdfannottext` takes an argument which specifies the text of an annotation to be created at the current position.

`\pdfannotlink`, `\pdfendlink` allows the user to specify hypertext links with all of the link attributes available in the PDF specification. An integer argument is used as a key to the corresponding anchor. If no link border has been specified, it is computed for all boxes between `\pdfannotlink` and `\pdfendlink`, so the link will automatically become multiline if line break occurs in between.

`\pdfoutline` allows for the generation of bookmarks; bookmarks can be hierarchically structured.

`\pdfdestxyz`, `\pdfdestfit`, `\pdfdestfith`, `\pdfdestfitv` provide specification of various types of anchors with zooming and fitting possibilities.

`\pdfdestfitr`, `\pdfendfitr` specify the position of anchor corners. In this case, the anchor area is computed from the corners.

4.2 Font handling

Font handling in TEX2PDF is currently limited to Type1 fonts only. Metric information is extracted from the `pfb` file. Font name mapping is handled using an auxiliary font mapping configuration file introducing the list of fonts available, together with the information on the type of font embedding and its usage.

Virtual fonts [17] are supported in TEX2PDF. As they are in fact part of `dvi` files, they have to be unfolded before PDF is output, as in today's DVI drivers.

4.3 Compression

Compression is allowed in the PDF specification, and several types of compression filters can be used; JPEG compression for color graphics, LZW and ZIP compression for text and graphics, and CCITT Group, Run Length and LZW compression for monochrome images.

As the LZW compression algorithm is licensed by UNISYS, we cannot distribute TEX2PDF with LZW support, but we used it for testing runs to compare TEX2PDF with Distiller (see table 1). However, the even more effective ZIP compression will be available in PDF version 1.2, avoiding the need for LZW compression in TEX2PDF, and the patent problems. The test figures show that TEX2PDF generated even more compact PDF file than Adobe Distiller on standard text files.

```
%% LaTeX2e file ‘t.tex’
%%
\hsize 3in
\baselineskip 13pt
\pdfoutput=1              % we will produce PDF instead of DVI
\pdfannottext
    open                  % optional specification if the text annotation is implic-
itly opened
    {The text annotation} % the text itself
\def\BL{\pdfannotlink
    depth 3pt height 8pt  % optional specification for link size
    1                     % key of destination
    border 0 0 1          % optional specification for link border
}
\def\EL{\pdfendlink}
\pdfoutline
    1                     % key of destination
    0                     % number of sub-entries of this item
    {The outline entry}   % Text of this item
\pdfdestxyz
    1                     % key of this destination
    zoom 2                % optional zoom factor
%\pdfdestfit  1 or %\pdfdestfith 1 or %\pdfdestfitv 1
%\pdfdestfitr 1 ... \pdfendfitr

This is \TeX, a document compiler intended to produce typesetting of
high quality.  The PASCAL program that follows is the definition of
\TeX82, a standard version of \TeX\ that is designed to be highly
portable so that identical output will be obtainable on a great
variety of computers.

The main purpose of the following program is to explain the algorithms
of \TeX\ as clearly as possible. \BL As a result, the program will not
necessarily be very efficient when a particular PASCAL compiler has
translated it into a particular machine language.\EL\ However, the
program has been written so that ...
\bye
```

Figure 1. Example of new hypertext primitives added in the TEX2PDF source file

4.4 Graphics

`\special`s are not yet handled by TEX2PDF. As most of the graphics included in TEX documents are PostScript and TIFF, at least support for the PostScript to PDF and TIFF to PDF conversion will have to be included in the future.

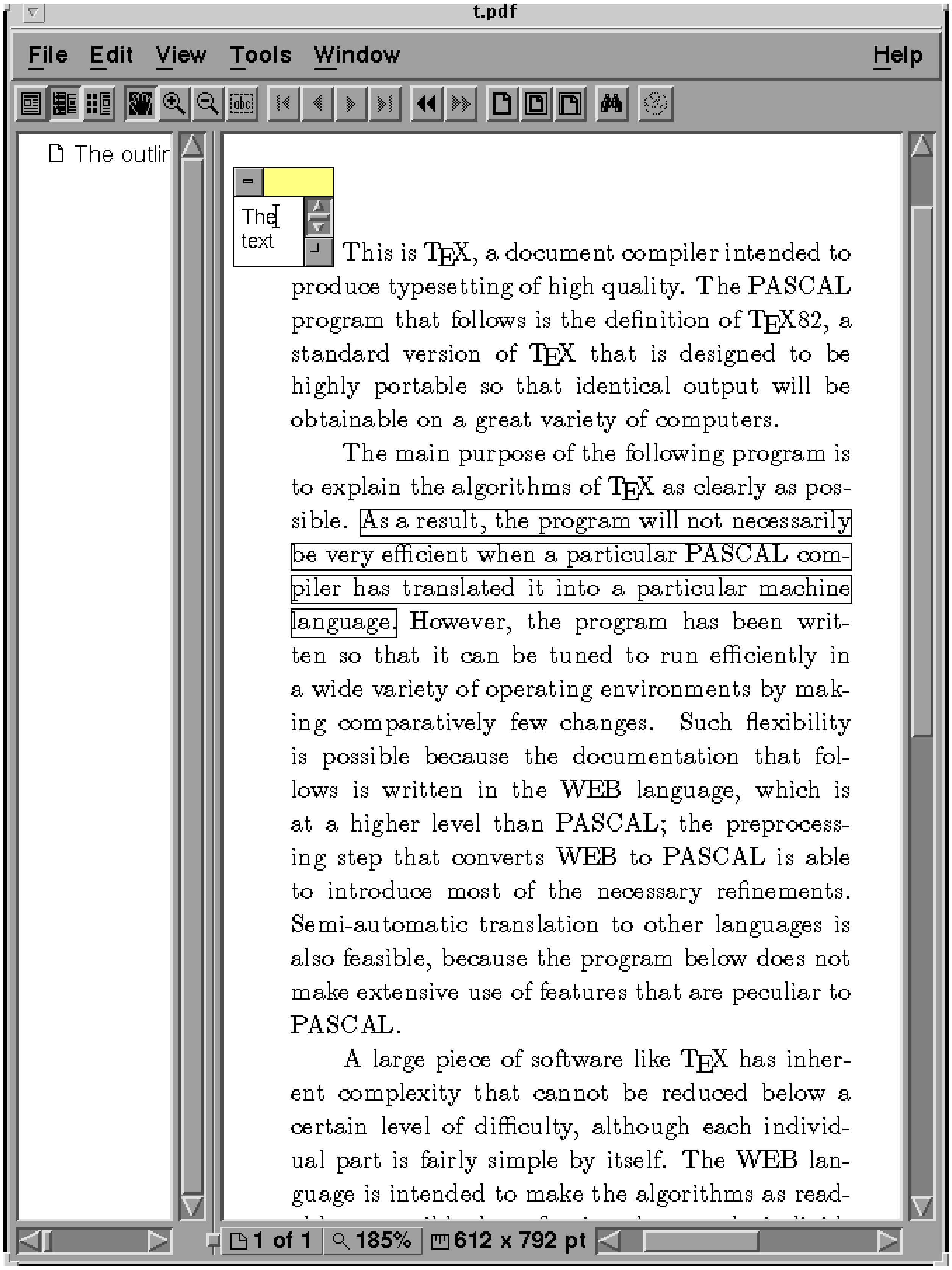
4.5 Implementation

The implementation of of TEX2PDF is realized as a `web` change file to the latest TEX source [19]. This implies that TEX2PDF is as portable as TEX itself is. Karl Berry's `web2c` package has been used for the development and for producing a running Unix version. We expect easy recompilation on any Unix platform.

5 Pros and Cons

I was constantly bombarded by ideas
for extensions, and I was constantly turning
a deaf ear to everything that did not fit
well with TEX as I conceived it at the time.
Don Knuth [20], p. 640

To compare TEX2PDF with the other methods of producing a hypertext PDF document from a TEX file, we did several testing runs. They were done on a Sun Sparc 10 under the Solaris 2.4 operating system. Measurements were done using the `time` program (CPU times are listed). We used `tex.tex`, generated from the TEX source (`tex.web`) file, as the



This is TEX, a document compiler intended to produce typesetting of high quality. The PASCAL program that follows is the definition of TEX82, a standard version of TEX that is designed to be highly portable so that identical output will be obtainable on a great variety of computers.

The main purpose of the following program is to explain the algorithms of TEX as clearly as possible. As a result, the program will not necessarily be very efficient when a particular PASCAL compiler has translated it into a particular machine language. However, the program has been written so that it can be tuned to run efficiently in a wide variety of operating environments by making comparatively few changes. Such flexibility is possible because the documentation that follows is written in the WEB language, which is at a higher level than PASCAL; the preprocessing step that converts WEB to PASCAL is able to introduce most of the necessary refinements. Semi-automatic translation to other languages is also feasible, because the program below does not make extensive use of features that are peculiar to PASCAL.

A large piece of software like TEX has inherent complexity that cannot be reduced below a certain level of difficulty, although each individual part is fairly simple by itself. The WEB language is intended to make the algorithms as read-

Figure 2. Result of TEX2PDF source in Fig. 1 viewed in Acrobat Reader

testing document. For the hypertext version we used a slightly changed version of `webmac.tex` (see `http://www.cstug.cz/~thanh/tex2pdf`).

In both time and size comparisons TEX2PDF beats its competitors (see tables 1 and 2). This is mainly due to the absence of intermediate DVI and PostScript formats in TEX2PDF, allowing for better PDF optimization. TEX2PDF is slightly slowed down by `pfb` file parsing.

The users familiar with the (emacs + TEX + xhdvi (+ ghostscript)) suite of programs might want to switch to (emacs + TEX2PDF + xpdf), thus speeding up the document debugging cycle considerably.

TEX2PDF is written in `web` so that its source blends naturally with the source of TEX the program. The obvious benefit is absolute compatibility with TEX proper; the actual code which drives the typesetting engine is that of Don Knuth (modulo `whatsits` use for the hypertext primitives added in TEX2PDF). While this conformance to TEX source greatly benefits from Don's appreciation of stability, it makes the implementor's life more difficult in the world where PDF still evolves. It is also hard to debug TEX2PDF without incremental compilation. When we come to add implementation of `\special` commands, maintenance will become tough.

The changes introduced in new versions of PDF are motivated by achieving better performance when handling Acrobat documents, and so TEX2PDF is bound to have the PDF-generating modules modified or rewritten so that maximum benefit of the features supported by PDF technology can be used. The fact that PDF specification has been made public is crucial to success of this approach.

The TEX2PDF aproach is naturally backward compatible with TEX—in fact, if PDF output is not switched on, it can still generate DVI output identical to that of TEX. Just by redefining some cross-referencing macros, the new hypertext features of TEX2PDF can be instantly used even without modifying the markup of old LATEX documents.

6 Object Reuse

Using well-designed formats results
in LATEX source that clearly reflects
the document structure.
T. V. Raman [23]

With PDF, there is the possibility of taking advantage of the object structure and manipulation specified within a PDF file to store elements of document structure (higher level document model) in the PDF file generated by the application (TEX2PDF). Some work has been already done in this direction by defining Encapsulated PDF (EPDF) blocks and their reuse [25]. This format, however, is not supported or used by a wide variety of applications.

The logical structure of a document model is also urgently needed in applications like AsTeR [23], which *reads* LATEX documents using a speech synthesizer. Developing an application that is able of reading aloud enriched PDF files might become possible.

Our suggestions for further work could lead to primitives which allow handling of PDF *objects* stored in the trailer of a PDF file indirectly. At least three primitives are foreseen:

`\setpdfbox` typesets its argument and stores the result as a PDF object. The reference to that object will stay in the internal register accessible by `\lastpdfbox`.

`\lastpdfbox` returns the reference to the last stored object by `\setpdfbox`.

`\usepdfbox` This primitive puts a *reference* to an object into the output stream.

7 Future Work

Few claim to know what will be the preferred
electronic format a century from now,
but I'am willing to go out on a limb
and assert that it will be none of TEX,
PostScript, PDF, Microsoft Word, nor any
other format currently in existence.
Paul Ginsparg [11]

TEX2PDF is currently under development and is available to beta testers only. We do not guarantee that the input syntax will remain unchanged. Support for object reuse and graphics when the PDF specification 1.2 comes out may be added.

For testing purposes, a `tex2pdf` option for the hyperref package [15] will be written, using the hypertext possibilities of TEX2PDF directly. This will allow using TEX2PDF for re-typesetting of LATEX documents just by loading with `hyperref` package with the `tex2pdf` option in the document preamble.

Support for the full usage of Multiple Master technology remains to be added, possibly in the combination with METAFONT [14, 13]. Extensions of the paragraph breaking algorithm [21] to take advantage of Multiple Master fonts with a variable width axis (but constant grayness) to help justification (`\emergencyfontwidthstretch`) is another possible direction of future work.

Acknowledgements

The support of TUG '96 bursary committee is acknowledged, having allowed presentation of preliminary version of this paper at the TUG '96 conference in Dubna, Russia.

References

[1] Adobe. Adobe acrobat 3.0 beta. `http://www.adobe.com/acrobat/3beta/main.html`, 1996.

[2] Adobe Systems. POSTSCRIPT *Language Reference Manual*. Addison-Wesley, Reading, MA, USA, 1985.

[3] American National Standards Institute and International Organization for Standardization. *Information processing: Text and Office Systems: Standard Generalized Markup Language (SGML)*. American National Standards Institute, 1430 Broadway, New York, NY 10018, USA, 1985.

[4] Association of American Publishers. *Association of American Publishers Electronic Manuscript Series Standard for Electronic Manuscript Preparation and Markup: an SGML Application Conforming to International Standard ISO 8879–Standard Generalized Markup Language. Version 2.0 Dublin, Ohio: Available from the Electronic Publishing Special Interest Group, c1987*. Association of American Publishers, Dublin, OH, USA, 1987.

[5] Tim Bienz, Richard Cohn, and James R. Meehan. *Portable Document Format Reference Manual, Version 1.1*. Addison-Wesley, Reading, MA, USA, 1996.

[6] Steven J. DeRose and David G. Durand. *Making Hypermedia Work*. Kluwer Academic Publishers Group, Norwell, MA, USA, and Dordrecht, The Netherlands, 1994.

[7] Andrew E. Dobrowolski. Typesetting SGML documents using TEX. *TUGboat*, 12(3):409–414, December 1991.

[8] Inc. Emerge. TEX and pdf: Solving font problems. |http://www.emrg.com/texpdf.html|, 1996.

[9] David Fuchs. The Format of TEX's DVI Files. *TUGboat*, 1(1):17, October 1980.

[10] David Fuchs. The Format of TEX's DVI Files. *TUGboat*, 3(2):14, October 1982.

[11] Paul Ginsparg. Winners and losers in the global research village. `http://xxx.lanl.gov/blurb/pg96unesco.html`, February 1996.

[12] Charles F. Goldfarb and Yuri Rubinsky. *The SGML handbook*. Clarendon Press, Oxford, UK, 1990.

[13] Michel Goossens, Sebastian Rahtz, and Robin Fairbairns. Using Adobe Type 1 Multiple Master Fonts with TEX. *TUGboat*, 16(3):253–258, June 1995.

[14] Yannis Haralambous. Parametrization of Postscript Fonts through METAFONT—an Alternative to Adobe Multiple Master Fonts. *Electronic Publishing*, 6(3):145–157, April 1994.

[15] Yannis Haralambous and Sebastian Rahtz. LATEX, Hypertext and PDF, or the Entry of TEX into the World of Hypertext. *TUGboat*, 16(2):162–173, June 1995.

[16] Berthold K. P. Horn. Acrobat pdf from TEX. `http://www.YandY.com/pdf_from.pdf`, 1996.

[17] Donald Knuth. Virtual Fonts: More Fun for Grand Wizards. *TUGboat*, 11(1):13–23, April 1990.

[18] Donald E. Knuth. *The TEXbook*, volume A of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[19] Donald E. Knuth. *TEX: The Program*, volume B of *Computers and Typesetting*. Addison-Wesley, Reading, MA, USA, 1986.

[20] Donald E. Knuth. The Errors of TEX. *Software–Practice and Experience*, 19(7):607–685, 1989.

[21] Donald E. Knuth and Michael F. Plass. Breaking paragraphs into lines. *Software–Practice and Experience*, 11:1119–1184, 1981.

[22] Sebastian P. Q. Rahtz. Another Look at LATEX to SGML Conversion. *TUGboat*, 16(3):162–173, September 1995.

[23] T. V. Raman. An Audio View of TEX Documents. *TUGboat*, 13(3):372–379, October 1992.

[24] Tomas G. Rokicki. A proposed standard for specials. *TUGboat*, 16(4):395–401, December 1995.

[25] Philip N. Smith. Block-Base Formatting with Encapsulated PDF. Technical Report NOTTCS-TR-95-1, Department of Computer Science, University of Nottingham, January 1995. `http://www.ep.cs.nott.ac.uk/~pns/pdfcorner/complete.pdf`.

[26] Mike Sofka. Dvi driver implementation and standardization issues. available as `http://www.rpi.edu/~sofkam/DVI/dvi.html`, 1996–.

[27] Han The Thanh. Portable document format and typesetting system TEX (in Czech). Master's thesis, Masaryk University, Brno, April 1996.

[28] Kendall Whitehouse. Creating quality Adobe pdf files from TEX with dvips. `http://www.adobe.com/supportservice/custsupport/SOLUTIONS/3a26.htm`, 1996.

II LaTeX, dvips, EPS and the web . . .

Sebastian Rahtz
Elsevier Science Ltd
The Boulevard, Langford Lane
Kidlington
Oxford, UK
`s.rahtz@elsevier.co.uk`

Summary
Browsers of TeX question *fora* like `comp.text.tex` will often be asked what are the issues surrounding Encapsulated PostScript, and how one goes about making EPS files from LaTeX output, and maybe using them on the World Wide Web. This short note[2] offers some suggestions.

1 What and why is EPS?

EPS stands for Encapsulated PostScript; EPS files *are* PostScript, but they conform to a minimum standard of good behaviour. This is so they can be included in other documents, possibly resized or rotated. In practice EPS means not using certain commands which have global effects (don't worry, this is quite rare), and inserting structured comments (starting with `%%`) which tell other programs something about the file. The *PostScript Language Reference Manual* goes into great depth describing what these comments can contain, but the minumum that is necessary for practical purposes are:

1. A first line starting `%!PS-Adobe`; dvips, for instance, puts `%!PS-Adobe-2.0 EPSF-2.0` in its output, meaning that it claims conformance with version 2 of the EPS standard (we are now at version 3);
2. A 'BoundingBox', like `%%BoundingBox: 33 101 584 715` which tells applications how much space on the page is occupied.

How do you turn PS files into EPS files? They probably are already, if they come from a reputable bit of software (avoid anything from MicroSoft)—a good check is to see if there is a BoundingBox.

You will come across three types of problem with files that look like EPS. Firstly, the BoundingBox may not be accurate; since this determines how much space will be left in enclosing applications like TeX, it matters. Keith Reckdahl's recent tutorial in *TUGboat* goes into detail on this problem.

Secondly, your file may be *serious* EPS, and use all the facilities of structured comments to specify what sort of resources (fonts etc) it expects you to supply when you deal with it. This is bad news if you are in TeX world outside a Macintosh. Look out for lines with words like `ProcSetsNeeded`.

Thirdly, your file may think it is EPS, but in fact breaks the rules, and has weird PostScript in it. The rescue technique is to read it with a forgiving PostScript interpreter, and get a new version written out. Three programs to try are:

1. Adobe Acrobat Distiller; this turns PostScript files into PDF, and Acrobat Exchange can then load them, and save them as ordinary PostScript. Since it is written by Adobe, Distiller is an extremely powerful PostScript interpreter, and can cope with almost anything you throw at it. It is not cheap (except to academics), but worth having.
2. Recent versions of Adobe Illustrator share some of the Acrobat code, and can read PostScript files, as well as edit PDF files.
3. The free Ghostscript is now a very mature and sophisticated product. It understands all of the current Level 2 PostScript, and can turn it onto a wide variety of bitmap forms. Version 4 (released in June 1996) also performs many of the functions of Distiller, and it already reads PDF files and writes PostScript. Unfortunately, its handling of PostScript text to PDF is at present unfinished. However, you can still use Ghostscript to read your PostScript and write it out again as a bitmap (e.g. TIFF).

[2]Reprinted from *TUGboat* 16(3) with kind permission of Barbara Beeton.

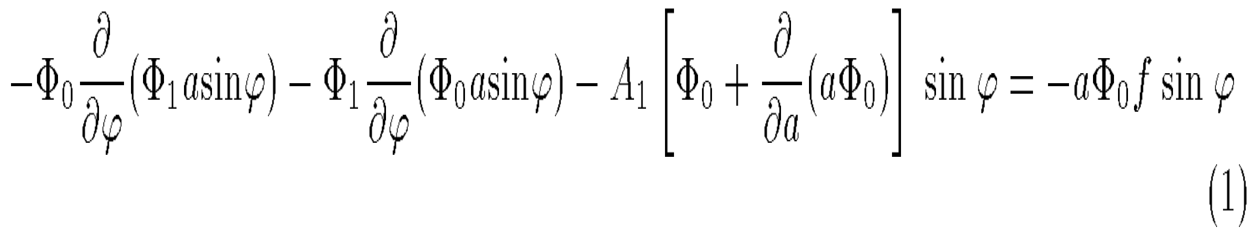


Figure 1. Bitmap EPS file, enlarged and distorted

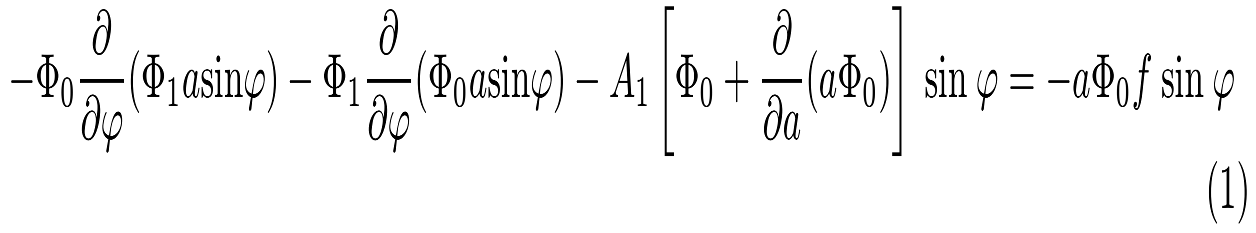


Figure 2. Outline font EPS file, enlarged and distorted

2 What about `dvi` to Encapsulated PostScript?

Most TEX systems, free or commercial, supply a `dvi` to PostScript driver; most of them write out more or less acceptable Encapsulated PostScript, but three are especially well-featured (in the author's experience): the Macintosh Textures driver, Y&Y's dvipsone for Windows and the free dvips. Since the latter is available for all platforms, is well-supported, and is probably the finest of its type,[3] we shall concentrate on that.

If you want to produce re-useable PostScript output from dvips (and this includes output destined for Acrobat Distiller), the absolute priority is to use outline fonts, not the PK fonts traditionally used by TEX. You can either use traditional fonts (usually commercial, like Adobe Times, but Ghostscript now comes with an excellent free set donated by URW) or Computer Modern itself in PostScript Type 1 format. Either buy these from Y&Y for Windows and Unix or Blue Sky for Macintosh, or use Basil Malyshev's BaKoMa set, of almost comparable quality.[4]

If you do not use outline fonts, and re-use your output scaled up, you will not like the effect of Figure 1 at all, compared to Figure 2. If you want to turn your documents into PDF, Distiller will produce vile results from PK fonts.

The second priority is to get the right bounding box. Surprisingly many applications cheat by simply making it the page size, regardless of whether the whole area is used. dvips does this by default too, but has a command-line option `-E`, which asks it to try and calculate the actual extent used. Note that EPS files are, by definition, only one page, so you also have to use dvips options to select just one page. There are two caveats when preparing the input. Firstly, make sure you do not include a page number (try `\pagestyle{empty}` in LATEX), or else the bounding box will cover that too. Secondly, dvips does not always work out the extent of text correctly. For instance, if you wrote (why, I have no idea):

```
Hello\raisebox{10pt}[0pt][0pt]{Up there}!
```

you would be asking LATEX to raise *Up there* off the baseline, but to pretend that it has no effect on the height calculation. dvips will believe this, and calculate a bounding box on the *claimed* height. If you use complicated add-in packages like PSTricks, which add in arbitrary PostScript code, you will also end up in real trouble. In these cases you can either adjust the BoundingBox by hand, or place invisible marks in LATEX to make sure that dvips recognizes the full extent.

A useful trick to remember if you think that TEX knows what you want, but dvips does not, is to make judicious use of color. Suppose you wanted to use PSTricks to encircle a mathematical symbol, you might write:

```
absurd \pscirclebox{$\surd$}
```

[3]For several years, dvipsone has offered partial downloading of fonts, a very powerful feature, but this is now coming into dvips; there are also flaws in dvips' use of structured EPS comments, and Textures is superior in this respect.

[4]Windows-worshippers may prefer to get into the world of TrueType fonts, which are available for Computer Modern from Kinch Computer Company.

TeX leaves the right space, since the PSTricks macros understand what is going on, but dvips is told to draw the circle in raw PostScript, and the bounding box calculation ignores that. The result is that the limits are set just around the size of the letters. If we wrote:

```
\framebox{absurd \pscirclebox{$\surd$}}
```

it would work correctly, because dvips would look at the enclosing frame, not just the words. But you end up with an unwanted box; so make it (in effect) invisible by writing:

```
{\color{white}\fboxsep{0pt}%
 \framebox{%
  {\color{black}absurd
  \pscirclebox{$\surd$}}%
  }%
}
```

This creates a white frame around black text; LaTeX proceeds happily, and so does dvips, calculating the right extents, but nothing shows on paper. Obviously, this only works in a monochrome environment.

3 LaTeX to EPS to GIF to Web

Why do we do all this in practice? Often, these days, because people want their LaTeX mathematical output on the World Wide Web, and their only recourse is to embed GIF images in their HTML. The sophisticated *latex2html* program does all this for you; its technique is worth understanding, as it has general utility; the sequence of events is:

1. Place bits of LaTeX in an special file, one fragment per page, and with no page numbers;
2. Run LaTeX to generate a multi-page `dvi` file;
3. Use dvips' `-i` and `-S` options to generate one self-contained output file per page;
4. Give each page to Ghostscript, and ask it to render them in PBM (Portable Bitmap) form;
5. Use the PBMplus/Netpbm utility *pnmcrop* to trim away white space;
6. Use the *ppmtogif* utility to convert the result to a GIF image.

Note that it does *not* use the `-E` option for dvips, but relies on simply removing all white pixels until just text is left. This has the advantage that it avoids the problem we saw in the last section, but it has three disadvantages:

1. The PBM utilities are primarily Unix tools, and many people do not have access to them;
2. The cropping process is memory-intensive, slow and eats temporary disk space;
3. The cropping forces everything to the baseline, effectively. A character like em-dash (—) which sits above the baseline, will be cropped above and below, so that the placed GIF looks wrong.

The core of the problem is the use of Ghostscript, which always creates a page-sized bitmap, even if there is only one word on the page. What we want is for Ghostscript to render just the portion of the image inside the bounding box, if we *do* use the `-E` flag for dvips. We can achieve this by giving Ghostscript a customized page size, which is the size of the bounding box. Then we can insert some extra PostScript code to move the image so that it starts at the 0,0 coordinate (adjusting the bounding box accordingly). Ghostscript then displays or converts the image just within the desired area, and no cropping is needed.

The transformations of the bounding box can be achieved using *epsffit*, which is part of Angus Duggan's `psutils` collection (CTAN:`support/psutils`); the page size change is most easily done using a Level 2 PostScript operator `setpagedevice`. Thus a PostScript file which starts:

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 135 528 284 668
...
```

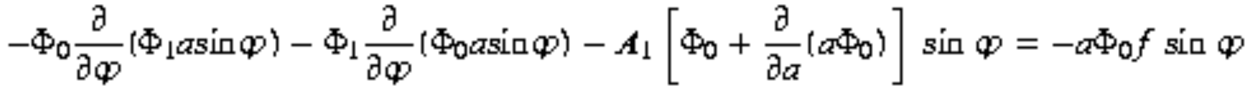
needs to be transformed to something like:

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 149 140
<< /PageSize [149 140] >> setpagedevice
gsave -135 -528 translate
...
grestore
```

Here we have worked out the width and height of the enclosing rectangle (149×140 units), moved the origin down to 0,0 on the page, and set the page size. PostScript purists will shudder at the `setpagedevice` command, and point

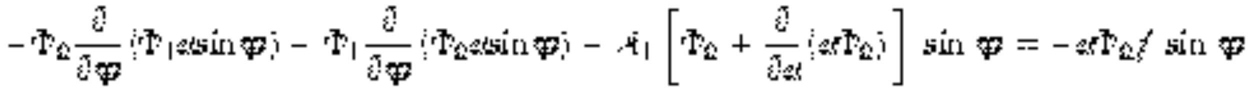
```
#!/usr/local/bin/perl
$bbneeded=1;
$bbpatt="[0-9\.\-]";
while (<>) {
 if ( /%%BoundingBox:(\s$bbpatt+)\s($bbpatt+)\s($bbpatt+)\s($bbpatt+)/ )
 {
     if ($bbneeded) {
         $width = $3 - $1;
         $height = $4 - $2;
         $xoffset = 0 - $1;
         $yoffset = 0 - $2;
         print "%%BoundingBox: 0 0 $width $height\n";
         print "<< /PageSize [$width $height] >> setpagedevice\n";
         print "gsave $xoffset $yoffset translate\n";
         $bbneeded=0;
    }
}
else {  print; }
}
print "grestore\n";
};
```

Figure 3. A Perl script to transform an EPS file for Ghostscript



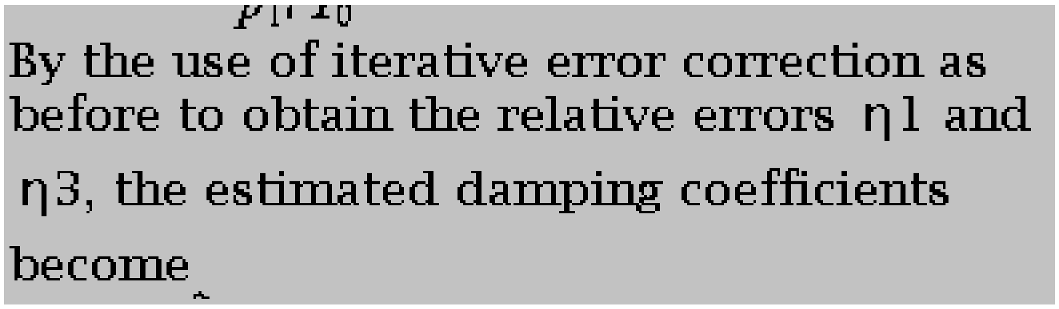
$$-\Phi_0\frac{\partial}{\partial\varphi}(\Phi_1 a\sin\varphi) - \Phi_1\frac{\partial}{\partial\varphi}(\Phi_0 a\sin\varphi) - A_1\left[\Phi_0+\frac{\partial}{\partial a}(a\Phi_0)\right]\sin\varphi = -a\Phi_0 f\sin\varphi$$

Figure 4. LaTeX → `dvi` → EPS → GIF



$$-\Phi_0\frac{\partial}{\partial\varphi}(\Phi_1 a\sin\varphi) - \Phi_1\frac{\partial}{\partial\varphi}(\Phi_0 a\sin\varphi) - A_1\left[\Phi_0+\frac{\partial}{\partial a}(a\Phi_0)\right]\sin\varphi = -a\Phi_0 f\sin\varphi$$

Figure 5. LaTeX → `dvi` → EPS → GIF, anti-aliased



By the use of iterative error correction as before to obtain the relative errors η1 and η3, the estimated damping coefficients become

Figure 6. Mid-aligned GIF image in Netscape

out that this is probably illegal in Encapsulated PostScript, but as long as we only use this file strictly in the controlled environment of Ghostscript, we are safe enough. Figure 3 lists a simple Perl script which performs the necessary changes to a PostScript file for Ghostscript to eat, without any need for *epsffit*.[5]

[5] I am aware that it does not cope with an `(atend)` bounding box . . .

Now that Ghostscript is only rendering the desired area, we can use its builtin bitmap output facilities. The Unix or DOS command line:

```
gs -dNOPAUSE  -q -r100 -sDEVICE=tiffg4 \
 -sOutputFile=foo.tif foo.ps -c quit
```

will generate a TIFF fax group 4 image (Ghostscript does not support GIF output directly, for legal reasons) at 100dpi of just the imaged area of the PostScript file `foo.ps` with no further ado. Ghostscript version 4 adds anti-aliasing facilities; using the Netpbm tools under Unix, we can create a variant GIF image, using the command line:

```
 gs -r100 -dNOPAUSE  -q -sOutputFile=-  \
  -sDEVICE=pnm -dTextAlphaBits=4 \
  -dGraphicsAlphaBits=4 foo.ps -c quit | \
  ppmtogif -interlace \
  -transparent \#ffffff > \
  equation.gif
```

Figures 4 and 5 show the result of transformations with and without anti-aliasing.

There is one remaining problem — the World Wide Web browsers can usually align images top, middle or bottom; but what if we have an image of some characters with descenders below the base line? Bottom alignment of the images places the bottom of the descenders on the baseline; top alignment is riduculous, and middle alignment is not quite right either. The answer is to use middle alignment, and make TeX lie to dvips (and thence down the chain) about the extent of the character; making its depth equal to its height, and then middle aligning it in the Web browser, has the desired effect. So how do we make TeX lie? Here is my suggestion:

```
\newsavebox{\@Fragment}
\def\Fragment#1{%
 \savebox{\@Fragment}{#1}%
 \@tempdima\ht\@Fragment
 \@tempdimb\dp\@Fragment
 \ifdim\@tempdima>\@tempdimb
  \dp\@Fragment\@tempdima
 \else
  \ht\@Fragment\@tempdimb
 \fi
 \fboxsep0pt
 \color{white}%
 \fbox{%
 {\color{black}%
  \box\@Fragment}%
 }%
}
```

I use the LaTeX box framing command to ensure that dvips thinks the depth is there, with the same color trick as we saw earlier.

Unfortunately, there is a side effect — an HTML browser loading the resulting GIF image mid-aligns the image and sticks the ‘ballast’ white space into the line below, making an unsightly gap (see Figure 6, where the Greek ηs have a small descender). With the current browser technology, there is little than be done about this. In practice, we will have to check first whether there *is* any descender; if so, we use the mid-align technique, and accept the gap; if there is not, we can make a simpler process and use bottom alignment.

It is imperative, of course, that Web-making readers do not take these examples as ‘recipes’, without both a precise specification of the desired Web page, or an understanding of some of the basic image-processing techniques. The aim here has simply been to show how relatively trivial and efficient it is to create bitmap output from LaTeX and dvips using the free facilities of Ghostscript.

III Newsletters with LaTeX

Andrew F. Lack
City University

Summary

I have typeset newsletters for 4 different organisations over the past three years using LaTeX. This article outlines some of my experiences of using LaTeX to produce them.

LaTeX and TeX were never designed to produce artwork for newsletters. And yet their flexibility, together with some clever minor style files, means that a very acceptable product can be produced. Just take a look at *Baskerville* to see this.

For the last three years I have typeset the newsletter of City University's Computing Services department. Prior to my involvement the usual mixture of software—and consequent styles—were tried. Packages like MicroSoft Word and Ventura Publisher were used.

I was convinced that, despite its rather formal page layout, LaTeX could be used to produce a good newsletter and after eight issues I'm still convinced. Besides, when I took the newsletter on I was *only* going to use LaTeX.

1 Text Columns

A newsletter, particularly one printed on A4 paper, is going to require multiple columns—probably 2 or 3. Because of the necessity to mix multi and single columnar styles on a page (single column for article headings), the minor style file multicol is more-or-less an essential starting point. However, the multicol style does have some limitations, most notable only allowing full textwidth floats. This complicates the placement of items such as tables and illustrations which are column-width, and these must be defined in the document at a fixed position.

With DTP packages, boxes or frames are defined where illustrations are to appear on the page, the text automagically flowing around them. This design not only simplifies the placement of illustrations, but also allows illustrations to be placed *between* columns—a popular way of including a photograph of the author, for example, at the start of an article. This isn't possible with LaTeX.

However, given these limitations typesetting articles with column-width illustrations is quite straight forward. The placement of these is probably best left to near the end of the typesetting process, adding them to the file starting at the beginning and working towards the end, checking page breaks as you go.

2 Short Articles

Using the rather formal full-width heading above a short article (perhaps using only one quarter of the page depth) looks rather odd. For short articles consider setting them in a box of width 0.75 or 0.8 of the full text width. Using box styles from the minor style file fancybox adds emphasis to the design. Another useful technique is to place a 5% or 10% gray tint behind the box using the psboxit minor style. If you do this avoid the fancy box styles and stick to the good old plain `fbox`, as the background PostScript shading extends beyond the box boundaries for `shadowbox` and `doublebox` and this look rather odd.

3 Graphics

Despite the fact that I produce final artwork using PostScript, I prefer working with graphics using the bm2font system, which converts graphics to TeX fonts. This has the advantage of producing images which are viewable using standard `dvi` previewers and they print much faster than encapsulated PostScript versions do.

Bm2font handles a wide variety of popular (PC) bitmap formats, including PCX and GIF.

If you are looking for copyright-free clip art, then take a look at the publications of *The Dover Press*. These US publications are available from stockists in the UK and are priced at around £5–6 per book.

4 Starting Articles

With ideal copy new articles will start at the top of a new page, with minimal wastage on the previous. However this rarely happens. The heading for an article is a key part of the typographic design of the newsletter. When the reader reaches the bottom of a column, below which a new article begins, the typography must make it clear that the reader should move up to the top of the page to continue.

A clear break under the previous article is required, together with a device which extends across the page. I leave a gap of 3em before printing the new article's heading. Under the heading I like to use textwidth *shaded face rule* (a rule of 0.2pt, gap of 2pt and a rule of 1pt) underneath the new article's title.

Deciding where to start a new article can be tricky. Luckily the multicols environment takes some optional arguments, one of which allows you to specify the minimum space which must remain on the page before starting the new environment. The other optional argument is text which is set full-width. This is where the article's heading can be set.

A typographic embellishment for the start of the article is common. Two are often employed; a large initial capital, possibly a dropcap, or setting the first three of four words in small caps. I like using dropcaps (which can be typeset using TEX's `\hangindent` and `\hangafter`). However, this has the disadvantage of making the entire first paragraph an unbreakable box, so only use this technique if you have sufficient vertical space on the page.

5 Page Breaks

This is without doubt the most difficult aspect of newsletter construction. Due to the rather 'lumpy' nature of the copy, (particularly if your articles have many illustrations), LATEX will occasionally have problems finding good page breaks. LATEX2e's `\enlargethispage` helps a little here. [Well, being a 2.09 user I assume so.]

Sometimes bad page breaks can be avoided by making the copy slightly longer. This can be neatly achieved by the command `\looseness=1` within a longish paragraph. It'll cause TEX to increase the length of the paragraph by one line by loosening the glue. Do this a couple of times in an article and this may just be enough. Of course you could even do some copy-editing.

6 Typefaces

Computer Modern for a newsletter? No. Computer Modern has too much of a technical look about it and here in Britain we tend not to like Modern book faces, preferring the Oldstyle instead—faces like, well Baskerville and Garamond.

However for City University's newsletter I use *Lucida Bright*. I find it renders well on a 300dpi HP3Si laser printer, giving a clean black copy which our Print Room can work with. Computer Modern is too spidery on the 3Si. For display work (such as the article titles) I like New Century Schoolbook italic, and Bookman bold italic makes a good mast-head face.

IV An introduction to PSTricks, part 4

Sebastian Rahtz
Elsevier Science Ltd
The Boulevard, Langford Lane
Kidlington
Oxford, UK
s.rahtz@elsevier.co.uk

This article concludes my look at PSTricks. I hope you have enjoyed the show! The material has been drawn from a forthcoming book, entitled *The LaTeX Graphics Companion*, by Michel Goossens, Sebastian Rahtz and Frank Mittelbach, to be published by Addison Wesley in 1997. If you have enjoyed *Baskerville* articles in the last couple of years on Seminar, colour in LaTeX, and PSTricks, you may find the book of interest.

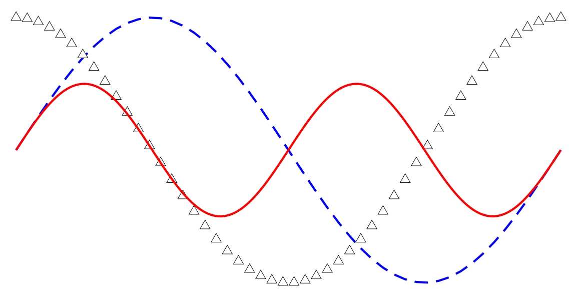
In this final part, we look briefly at data plotting, and then conclude with some finished examples of PSTricks, which are designed to show the power of the macros in unusual ways.

1 Data plotting

PSTricks has a set of high-level tools for common data-plotting functions; these can read data from external files, in a variety of formats. We will not tabulate the extra commands or graphical parameters this time, as they can be easily found in the PSTricks documentation. The simplest form of data is a set of comma or white-space delimited numbers, but values can also be enclosed in braces (`{}`) or round brackets (`()`). If the data is enclosed in a single set of *square* brackets (`[]`), and the opening `[` is at the start of a line, it will be read much faster; however, it will run TeX out of memory sooner. Data can also be read once, and then re-used, with the `\readdata` and `\savedata` commands. There is an important distinction between `\fileplot` or `\dataplot` which parse and validate the data in TeX, and `\listplot`, which simply passes the data on to PostScript; the latter approach means that there is no check on PostScript memory requirements, but has the advantage that raw PostScript can be provided to generate or manipulate the data. Just to complicate matters, use of the `\PSTtoEPS` command with the plotting commands can allow for even bigger datasets. An example of this is given below, in section 2.

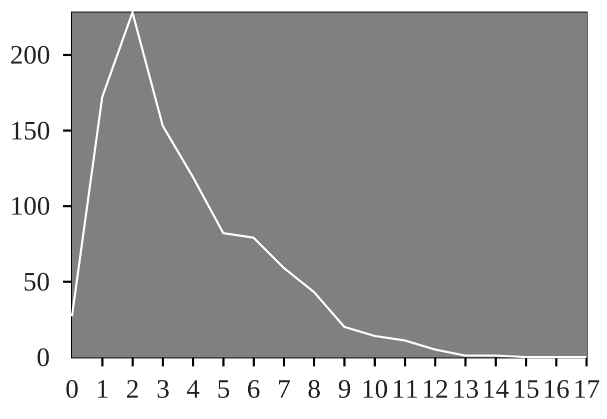
It is important to realize that it is up to the user to check the data extents and scaling; PSTricks does not make the data fit a predefined plot area, unlike many other packages. Normally, judicious setting of `xunit` and `yunit` will quickly produce nice results. Axes are generated separately, and there is no provision for supplying specific labels for axis tick points. It is, on the other hand, easy to superimpose multiple plots, and use the PostScript language to calculate functions, as the following example shows, overlaying $\sin(x)$, $\sin(x)\cos(x)$ and $\cos(x)$.

```
\begin{pspicture}(-1,-2.5)(9,2.5)
\psset{xunit=.20mm,yunit=1.75cm}
\psset{plotpoints=50}
\psplot[linestyle=dashed,linecolor=blue]
 {0}{360}{x sin}
\psplot[plotstyle=dots,dotstyle=triangle]
 {0}{360}{x cos}
\psset{plotpoints=200}
\psplot[linecolor=red]{0}{360}
 {x dup sin exch cos mul}
\end{pspicture}
```

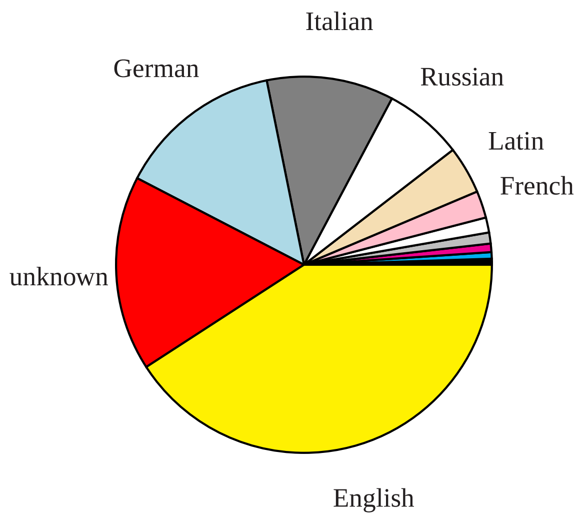


The next example shows how the `\psaxes` command can be used to create graph frames; in this mode, it obeys the graphical fill and colour parameters. This picture (showing word length (*x* axis) against occurrences (*y* axis) in a passage of Dickens) also shows how the axis labels can be manipulated.

```
\readdata{\foo}{words.dat}
\psset{yunit=.2mm,xunit=4mm}
\begin{pspicture}(-2,-50)(20,250)
\psaxes[axesstyle=frame,dy=50\psyunit,
  Dy=50,tickstyle=bottom,
  fillcolor=gray,fillstyle=solid](1,1)(18,229)
\dataplot[plotstyle=line,linecolor=white]{\foo}
\end{pspicture}
```



PSTricks is not designed to be a fully-fledged business graphics package; its plotting functions are really for simple scientific plots only. However, it should be clear that the lower-level PSTricks macros, and TEX's programmability, make it a good basis for whatever graphing is needed. We conclude this section with a crude pie chart, with the macros and some of the data used to create it; note that only segments above a certain size are labelled — providing a sensible label for all elements would involve considerably more care. Denis Girou has written a generalised pie-chart and bar chart creation package for PSTricks (under revision at the time of writing), which can produce very professional results.



```
\usepackage{calc,pstcol}
\newcommand{\lang}[4]{% name, value, proportion of
                      % 10000, colour
 \setcounter{thisval}{\value{lastval}}
 \addtocounter{thisval}{#3}
 \pswedge[fillcolor=#4]{1}{\thelastval}{\thethisval}%
 \setcounter{thishalf}{((\value{thisval}-
                           \value{lastval})/2)+
                             \value{lastval}}
 \ifnum#3>200\rput(1.3;\thethishalf){#1}\fi
 \setcounter{lastval}{\value{thisval}}
}
\psset{fillstyle=solid}
\degrees[10000]
\SpecialCoor
\setcounter{lastval}{0}
\lang{Romanian}{1}{3}{green}
\lang{Czech}{2}{6}{blue}
...
\lang{German}{508}{1421}{lightblue}
\lang{unknown}{599}{1676}{red}
\lang{English}{1462}{4085}{yellow}
```

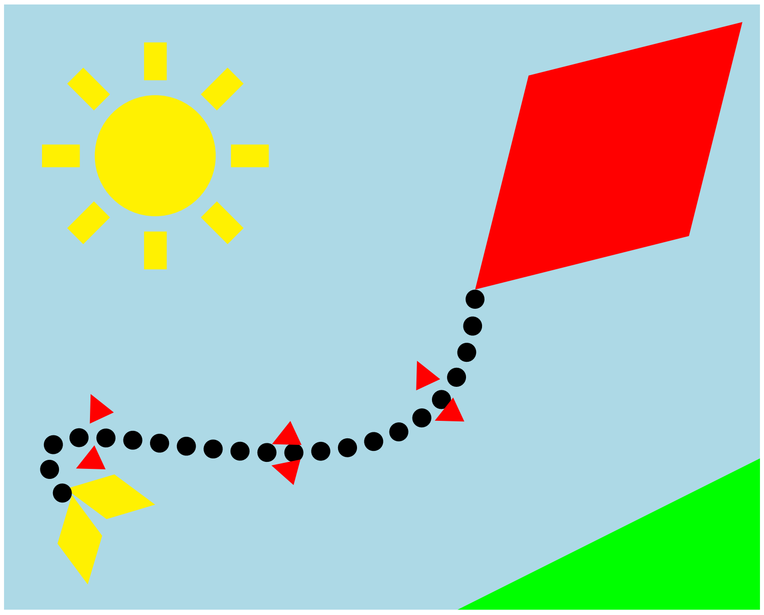
2 PSTricks programming examples

In the following pictures, we attempt to show some of the range of PSTricks possibilities, demonstrate the advantages of using a programming language for drawing pictures, and explore the various tools for simplifying and modularizing the code to make it more readable. We are especially glad to acknowledge Denis Girou for his input to this section, both in personal exchanges and in published examples.

In the first picture, a kite drawing from a child's book of colours and shapes, notice how the tail is drawn as a curved node connection between two points, and the bunting is added as labels on that connection. Use of the node feature means that the calculation of the line and positions along it are left entirely to PSTricks. It is also worth recalling the basic POSTSCRIPT premise that objects are opaque unless otherwise stated; this means that we can draw a blue background to the whole picture, and then overlay solid blocks of colour for the shapes. Some parts of the picture have a regular, repeating, feature, and we take advantage of this to draw the rays of sun using the `\multido` macro; note that we use a TEX group to localize the effect of the `\psset` which changes colour and style.

```
\begin{pspicture}(10,8)
\psset{fillstyle=solid,linestyle=none,linewidth=0}
\psframe[fillcolor=lightblue](10,8)
% sun
\pscircle[fillcolor=yellow](2,6){.8}
% rays
```

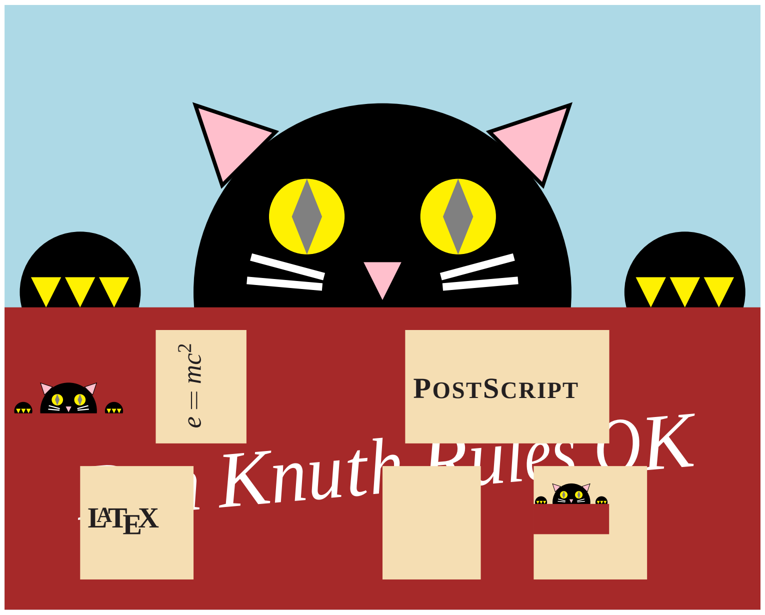
```
{%
 \psset{linecolor=yellow,linestyle=solid,
    linewidth=.3}
 \degrees[8]
 \multido{\i=1+1}{8}{\rput{\i}(2,6)
   {\psline(1,0)(1.5,0)}}
}%
% grass
\pspolygon[fillcolor=green](6,0)(10,2)(10,0)
% kite
\psdiamond[fillcolor=red,gangle=-45](8,6)(1.5,2.5)
\rput{45}(8,6){\pnode(-2.5,0){Kitetail}}
\rput{-10}(.8,1.5){\psdiamond[fillcolor=yellow]
 (.6,.1)(.6,.3)}
\rput{-80}(.8,1.5){\psdiamond[fillcolor=yellow]
 (.6,.1)(.6,.3)}
\pnode(.8,1.5){Tailend}
\nccurve[fillstyle=none,angleA=270,angleB=125,
 ncurvB=.9,ncurvA=1.4,linestyle=dotted,
 dotstyle=square,linewidth=.25]{Kitetail}{Tailend}
\newcommand{\bunting}{\pstriangle(.35,.35)}
\psset{fillcolor=red,labelsep=.01}
\naput[nrot=115,npos=.15]{\bunting}
\nbput[nrot=25,npos=.15]{\bunting}
\naput[nrot=75,npos=.4]{\bunting}
\nbput[nrot=115,npos=.4]{\bunting}
\naput[nrot=115,npos=.7]{\bunting}
\nbput[nrot=25,npos=.7]{\bunting}
\end{pspicture}
```



In the next child’s picture, we again take advantage of the strikeout nature of POSTSCRIPT blocks of solid colour to draw the cat head as a whole circle, and superimpose the wall (and on top of that the bricks) so that we do not worry about creating a precise wedge of just over a semicircle. `\rput` is used extensively to place objects at an angle. The writing on the bricks demonstrates the importance of understanding the reference point of objects that are placed. Since the bricks and their legends are drawn *after* the graffito, they partly obscure it. We group objects of similar characteristics together, and use TEX’s standard grouping to set PSTricks values for the items in that group. We also break the picture into different elements, describe each in a separate macro, and group them into high-level objects. This technique allows us to built up a library of objects, and serves to make the final picture description considerably more readable. The cat sitting on its portion of wall sets its size according to a parameter, allowing us to reproduce it several times at different sizes; note how the `\rput` command resets the coordinate system, so that the cat is drawn relative to the position of the `\rput`.

```
\DeclareFixedFont{\curly}{T1}{pzc}{m}{it}{30}
%
% The cat is designed to appear on a 10 x 10 grid
% cat head
\newcommand{\Cathead}{%
 \pscircle[fillcolor=black](5,4.2){2.5}%
% ears
 {%
  \psset{linecolor=black,fillcolor=pink,linewidth=.05,
   linestyle=solid}
  \rput{45}(5,4.2){\pspolygon(2.5,.5)(2.5,-.5)(3.5,0)}
  \rput{135}(5,4.2){\pspolygon(2.5,.5)(2.5,-.5)(3.5,0)}
 }%
}
% eyes, nose and whiskers
\newcommand{\Catface}{%
 \pscircle[fillcolor=yellow](4,5.2){.5}
 \psdiamond[fillcolor=gray](4,5.2)(.2,.5)
 \pscircle[fillcolor=yellow](6,5.2){.5}
 \psdiamond[fillcolor=gray](6,5.2)(.2,.5)
% nose
 \rput{180}(5,4.6){\pstriangle[fillcolor=pink](.5,.5)}
 % whiskers
 {%
  \psset{linecolor=white,linestyle=solid,linewidth=.1}
  \rput{5}(5,4.2){\psline(.8,0)(1.8,0)}
  \rput{15}(5,4.2){\psline(.8,0)(1.8,0)}
  \rput{165}(5,4.2){\psline(.8,0)(1.8,0)}
  \rput{175}(5,4.2){\psline(.8,0)(1.8,0)}
 }%
}
% paws
\newcommand{\Catpaws}{%
 \pscircle[fillcolor=black](1,4.2){.8}
 \pscircle[fillcolor=black](9,4.2){.8}
% claws
 {%
  \psset{fillcolor=yellow}
  \newcommand{\clawsize}{.4,.4}
  \rput{180}(1,4.4){\pstriangle(\clawsize)}
  \rput{180}(1,4.4){\pstriangle(-.45,0)(\clawsize)}
  \rput{180}(1,4.4){\pstriangle(.45,0)(\clawsize)}
  \rput{180}(9,4.4){\pstriangle(\clawsize)}
  \rput{180}(9,4.4){\pstriangle(-.45,0)(\clawsize)}
  \rput{180}(9,4.4){\pstriangle(.45,0)(\clawsize)}
 }%
}
% wall
\newcommand{\Wall}{%
 \psframe[fillcolor=brown](0,0)(10,4)
}
% The whole cat on its wall
\newcommand{\Cat}[1]{%
 {\psset{unit=#1}
  \Cathead\Catface\Catpaws\Wall}%
}
% bricks
\newcommand{\Bricks}{%
```

```
 \bfseries\large
 \psset{fillcolor=wheat}
 \psframe(1,.4)(2.5,1.9)
 \rput[bl](1.1,1){\LaTeX}
 \psframe(5,.4)(6.3,1.9)
 \psframe(7,.4)(8.5,1.9)
 \psframe(2,2.2)(3.2,3.7)
 \rput[bl]{90}(2.6,2.4){\normalsize$e=mc^2$}
 \psframe(5.3,2.2)(8,3.7)
 \rput[bl](5.4,2.8){\textsc{PostScript}}
}
\begin{pspicture}(10,8)
\psset{fillstyle=solid,linestyle=none,linewidth=0}
\psframe[fillcolor=lightblue](10,8)
\Cat{1}
\rput[bl]{5}(1,1){\curly\color{white}
 Don Knuth Rules OK}
\Bricks
\rput(7,1){\Cat{.1}}
\rput(.1,2){\Cat{.15}}
\end{pspicture}
```



The third complete picture is more complex, a circuit diagram. We have chosen to make this by programming a small language for circuit diagrams, which implements the actions `\Battery`, `\Resistor`, `\Switch` and `\Inductor`, with the movement commands `\Up`, `\Down` `\Left`, and `\Right` in the spirit of pic. Thus the final part of the input is quite simple, apart from the node connection and label commands which are used in their normal way (some care is needed to consider the angles at which connection lines leave and join each node).

```
\Up{1}
\Battery{A}
\Up{1}
\Right{2}
\Resistor{B}
\Right{1}
\Down{.3}
\Inductor{C}
\Down{.5}
\Left{1}
\Switch{D}
```

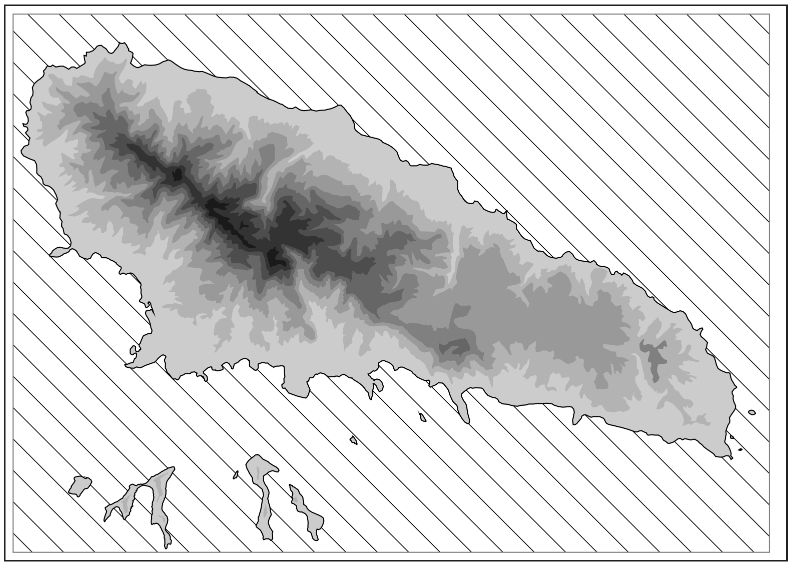
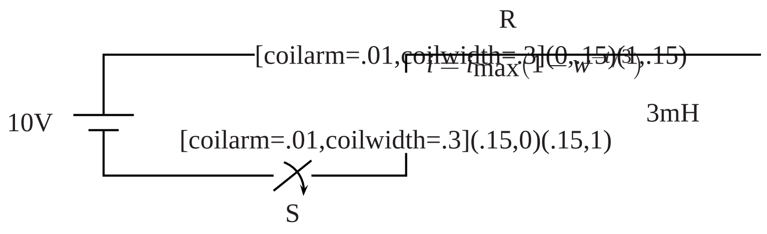


Figure 1. Autocad picture converted to PSTricks macros

```
\ncangle[angleA=90,angleB=180,armB=0]{A}{B}
\ncangle[angleA=0,angleB=90,armB=0]{B}{C}
\ncput[ref=l]{~~~$i=i_{\mbox{max}}(1-w^{-t/3})$}
\ncangle[angleA=-90,armB=0,angleB=0]{C}{D}
\ncangle[angleA=180,armB=0,angleB=-90]{D}{A}
\nput{180}{A}{10V}
\nput{90}{B}{R}
\nput{0}{C}{3mH}
\nput{270}{D}{S}
```


R
[coilarm=.01,coilwidth=.3](0,.15)(1,.15)
$i = i_{\text{max}}(1 - w^{-t/3})$
10V
3mH
[coilarm=.01,coilwidth=.3](.15,0)(.15,1)
S

The new commands are implemented in terms of a current *x* and *y* coordinate, which is changed by the movement commands. Thus `\Down` is defined (simplified) in the following code:

```
\newcommand{\Down}[1]{%
  \setcounter{CurY}}{-#1}%
  \def{Cirdirection}{3}%
  }
```

After each movement the current direction of movement is recorded, since this affects how we draw the new objects. These have a switch (which we have simplified in the following code example[6]) to check direction. The plain TEX `\ifcase` command is used to perform a 4-way switch between directions. The definition of the resistor is:

[6]since we possibly want sizes to be real numbers, `\CurX` and `\CurY` are in fact TEX dimensions, which we convert back to numbers before use.

```
\newcommand{\Resistor}[1]{%
 \ifcase\Cirdirection % right
   \rput[l](\CurX,\CurY){\rnode{#1}{%
   \pszigzag[coilarm=.01,coilwidth=.3](0,.15)(1,.15)%
   \MyBox{1}{.3}}}%
   \addtocounter{CurX}{1}%
 \or % left
   \addtocounter{CurX}{-1}%
   \rput[l](\CurX,\CurY){\rnode{#1}{%
   \pszigzag[coilarm=.01,coilwidth=.3](0,.15)(1,.15)%
   \MyBox{1}{.3}}}%
 \or % up
   \rput[b](\CurX,\CurY){\rnode{#1}{%
   \pszigzag[coilarm=.01,coilwidth=.3](.15,0)(.15,1)%
   \MyBox{.3}{1}}}%
   \addtocounter{CurY}{1}%
 \or % down
   \addtocounter{CurY}{-1}%
   \rput[b](\CurX,\CurY){\rnode{#1}{%
   \pszigzag[coilarm=.01,coilwidth=.3](.15,0)(.15,1)%
   \MyBox{.3}{1}}}%
 \fi
}
```

The macro `\MyBox` is very important; since by itself `\pszigzag` takes no space, it will create a node with no width or height, and connectors will go right to the middle. Therefore we put in some LATEX struts with the `\rule` command, to create an invisible box around the zigzag. The PSTricks units are converted to normal TEX lengths using `\pssetlength`.

```
\newlength{\Cirtemp}
\newcommand{\MyBox}[2]{% width,height
 \pssetlength{\Cirtemp}{#1}%
 \rule{\Cirtemp}{0pt}%
 \pssetlength{\Cirtemp}{#2}%
 \rule{0pt}{\Cirtemp}%
}
```

Our final example takes PSTricks into another subject area, that of cartography. The map in Figure 1 is created from an Autocad DXF file; this time the 36 polygons, comprising 9619 separate line segments, were converted (using a simple *ad hoc* conversion program) to a separate coordinate data file for each polygon. The overall map description consists simply of 36 lines of the form:

```
\ProcessVector{moh174}
\ProcessVector{moh170}
```

Since the number of coordinates is so large, many of the PSTricks tools which can read the files (like `\fileplot`) run out of TEX memory; however, for purely graphical objects like polygons, we have the more efficient and less memory-intensive option of writing Encapsulated POSTSCRIPT files on the fly, so we instantiate the `\ProcessVector` lines with the definition:

```
\newcommand{\ProcessVector}[1]{%
 \PSTtoEPS{#1.eps}{\fileplot{#1.dat}}
 \rput(0,0){\includegraphics{#1.eps}}%
}
```

The command `\PSTtoEPS` takes two arguments, a filename, and any pure PSTricks commands (*i.e.* not text). Instead of embedding the necessary POSTSCRIPT as `\special`s in the TEX output, a POSTSCRIPT file is written directly.

Using the very simple top level `\ProcessVector` command means that the master file is easily hand-edited; since the Autocad file identifies the polygons by their ‘layer’ (the map is a set of contour lines), we are able to set the fill colour separately for each layer, and so produce the more traditional map.

V The javaTEX project & `web2java`

Timothy Murphy
<tim@maths.tcd.ie>
School of Mathematics, Trinity College Dublin

Summary

The general aim of the javaTEX project is to examine the relation — if any — between Java and TEX, and in particular (as a first step) to translate the standard TEX programs (`tangle`, `tex`, `mf`, etc) into Java, using `web2java` — a simple modification of the standard UnixTEX utility `web2c`.

1 Why Java?

Java is a dialect of C; and 95% of the code produced by `web2java` is indistinguishable from C. At the same time, Java is much safer than C; for example, run-time checking of array-bounds catches many errors that would be missed by C.

Java is object-oriented, which is of course a Good Thing. Although in fact we have made very little use of this — apart from hiving off a few functions (I suppose we should say "methods"!) like `reset` and `rewrite` into `TeXlib.class`. But it would be nice to have a generic DVI driver which sent "messages" to an abstract `printer`, which could be "instantiated" as a PostScript printer, a screen viewer, etc.

Java has a simple graphics interface, although again we have made no use of this to date. It might prove *too* simple for the purposes of TEX.

Java also has a simple network interface, which could prove useful in integrating TEX into WWW (the World-Wide Web).

A theoretical advantage of Java is that it compiles into device-independent "bytecode" format, so that, for example, `tangle.class` compiled on a Sun should run under Windows 95.

There remains the intriguing possibility of DVI "Applets", as a way of putting TEX on the Web.

Against all these advantages there is at present one overwhelming disadvantage — Java is extremely slow. For example, it takes about 40 times as long to `tangle tex.web` in Java as it does in C. Doubtless this ratio can be somewhat reduced by better coding; but it seems improbable that it could be improved more than four-fold.

Our hope must be that a true Java compiler will soon be available, converting Java into machine code. Since Java is a standard programming language there seems no reason why such a compiler should not emerge; we would exect to see one before the end of the year.

2 How `web2java` works

`Web2java` — like `web2c` — is a post-processor to `tangle`. To create `foo.java` from `foo.web` and `foo.ch` one first runs `tangle`:

```
tangle foo.web foo.ch
```

This creates the Pascal (or pseudo-Pascal) file `tangle.p`.

(If you like driving in the slow lane, you could run the Java tangle instead:

```
java javaTeX.tangle foo.web foo.ch
```

Class files are supposedly machine independent, so `tangle.class` from the javaTEX distribution should run on any system. Note that this file, like all javaTEX programs, is defined to be in the `javaTeX` package, and so must be placed in a subdirectory called `javaTeX` relative to the `CLASSPATH`.)

This file is then passed through `web2java` to create `foo.java`:

$$\texttt{foo.web} + \texttt{foo.ch} \xrightarrow{\texttt{tangle}} \texttt{foo.p} \xrightarrow{\texttt{web2java}} \texttt{foo.java}.$$

Actually, this is a slight oversimplification. The file `common.defines` is prepended to `foo.p` *before* passing through `web2java`. And the resulting file is passed through `javafix.perl` *after* `web2java`:

$$\texttt{common.defines} + \texttt{foo.p} \xrightarrow{\texttt{web2java}} \mid \xrightarrow{\texttt{javafix.perl}} \texttt{foo.java}.$$

All this is completely analagous to `web2c`, except that we have replaced the C program `fixwrites.c` by the Perl script `javafix.perl`.

3 The `web2java` program

The filter `web2java` is created by the programs `flex` and `bison` (or `lex` and `yacc`) from the files `web2java.l` and `web2java.y`. This is completely analagous to `web2c`.

The file `web2java.l` is identical to `web2c.l`, with the addition of 5 new tokens: `try`, `catch`, `finally`, `new` and `cast`.

The syntax description in `web2java.y` has rather more changes, compared with `web2c.y`.

On the plus side, since Java has no pointers all the pointer-related material has been deleted. There is no attempt to determine if a function argument is “formal var” or not; and no need therefore to re-name functions with such arguments.

Classes have been introduced in a rather crude way, by allowing

```
VARIABLE = VARIABLE '.' VARIABLE
```

Thus in `Math.abs(...)` `Math` has been defined as a variable (`@define var Math;`).

On the other hand, in

```
web_file: Data_Input_Stream;
```

the class `DataInputStream` is defined as a type (`@define type DataInputStream;`).

No class (so far) has appeared in both rôles, so no conflict has arisen. (An earlier version, with `class` and `object` tokens, proved too complicated.)

4 Implementation

For the most part Web to Java is if anything simpler than Web to C. One apparent difficulty is the lack of a pre-processor in Java, since `web2c` leaves a good deal of work to `cpp`. This means that more must be done in the change file, which is probably a Good Thing.

The 3 main issues which arise are:

- The absence of `goto`s in Java;
- The lack of `typedef`s in Java; and
- Input/Output.

These are discussed in the following 3 subsections.

4.1 *Removing `goto`’s*

Java has no `goto`; Perhaps in compensation, it allows `break` and `continue` statements to carry a *label*, as for example in `break lab21` or `continue lab3`. The corresponding labels must appear at the beginning of the loop in question. (A `break` label can also be attached to a `switch` statement, but we make no use of that.)

We implement this as follows: `goto <num>` or `goto +<num>` are converted to `break lab`, as for example

$$\texttt{goto 6} \mapsto \texttt{break lab6}, \quad \texttt{goto +17} \mapsto \texttt{break lab17}.$$

On the other hand `goto -<num>` is converted to `continue lab`, as for example

$$\texttt{goto -7} \mapsto \texttt{continue lab7}.$$

Finally, `goto 0` is converted to `break`:

$$\texttt{goto 0} \mapsto \texttt{break}.$$

Labels of the form ±`<num>:` are commented out:

$$\texttt{-20:} \mapsto \texttt{/* lab20 */}.$$

As we shall see, this allows old labels to be left.

In practice, new labels are needed in virtually all cases. Here is a simple example

```
while n<>2 do begin
  ... goto done ...
end;
done: ...
```

Here `done` is defined at the beginning of the web file:

```
@d done=30 {go here to exit a loop}
```

We alter this (in the change file, of course) to

```
@d Done=30 {go here to exit a loop}
@d done==+Done
```

and we modify the loop above to

```
Done: while n<>2 do begin
  ... goto done ...
end;
done: ...
```

This translates in the Pascal file to

```
30: while n<>2 do begin
  ... goto +30 ...
end;
-30: ...
```

```
@x
@<Start translation of command |o| and |goto| the appropriate label to
  finish the job@>;
fin_set: @<Finish a command that either sets or puts a character, then
    |goto move_right| or |done|@>;
fin_rule: @<Finish a command that either sets or puts a rule, then
    |goto move_right| or |done|@>;
move_right: @<Finish a command that sets |h:=h+q|, then |goto done|@>;
show_state: @<Show the values of |ss|, |h|, |v|, |w|, |x|, |y|, |z|,
  |hh|, and |vv|; then |goto done|@>;
done: if showing then print_ln(’ ’);
@y
Done: loop begin
show_state: loop begin
move_right: loop begin
fin_rule: loop begin
fin_set: loop begin
@<Start translation of command |o| and |goto| the appropriate label to
  finish the job@>;
break end; @<Finish a command that either sets or puts a character, then
    |goto move_right| or |done|@>;
break end; @<Finish a command that either sets or puts a rule, then
    |goto move_right| or |done|@>;
break end; @<Finish a command that sets |h:=h+q|, then |goto done|@>;
break end; @<Show the values of |ss|, |h|, |v|, |w|, |x|, |y|, |z|,
  |hh|, and |vv|; then |goto done|@>;
break end; if showing then print_ln(’ ’);
@z
```

Figure 1. Translation of targets of `goto`s in `dvitype.ch`

and this in turn translates into the Java code

```
lab30: while n<>2 do begin
  ... break lab30 ...
end;
/* lab30: */ ...
```

It will be seen that in this case only one line needed to be changed, by the addition of the label `Done:`.

More often, 2 lines need to be changed. The following is a typical case.

```
reswitch: case ch of
  'x': ... goto reswitch ...
endcases;
```

At the beginning of the web file we read

```
@d reswitch=21 {go here to start a case
                        statement again}
```

We change this to

```
@d Reswitch=21 {go here to start a case
                        statement again}
@d reswitch==-Reswitch
@d break==goto 0
```

Now we alter the switch statement to

```
Reswitch: loop begin case ch of
  'x': ... goto reswitch ...
endcases; break end;
```

In the Pascal code this becomes

```
31: loop begin case ch of
  'x': ... goto -31 ...
endcases; goto 0 end;
```

which `web2java` converts to

```
lab31: while (true) { switch(ch) {
  case 'x': ... continue lab31 ...
} break; }
```

In practice all `goto` statements in the 'classic' web files can be dealt with in this way, introducing an appropriate loop if necessary. One can imagine cases where this would be very difficult, if not impossible. Fortunately these do not arise in practice. In fact virtually all `goto` statements can be eliminated as above, by following a very small number of practical rules.

Figure 1 is a fairly complicated case, from `dvitype.ch`. Recall that the material in `dvitype.web` between `@x` and `@y` is replaced by the material between `@y` and `@z`.

```
function signed_pair:integer; {returns the next two bytes, signed}
var a,@!b:eight_bits;
begin a:=0; b:=0;
try begin a:=dvi_file.readByte; b:=dvi_file.readUnsignedByte; end;
catch (ex: IOException) EOF_dvi_file:=true;
if EOF_dvi_file then signed_pair:=0
else begin cur_loc:=cur_loc+2; signed_pair:=a*256+b; end;
end;
```

Figure 2. Mapping of ordinary input from file in Java

4.2 Type definitions

There are no `typedef`s in Java. In theory one could replace `typedef`s by class definitions, but that would add considerable complication to the code. Instead we simply change them to substitutions (as though in C changing `typedef`s to `#define`'s).

So for example we make the change

```
@x
@<Types...@>=
@!ASCII_code=0..255;
@y
@d ASCII_code==0..255
@z
```

Later `web2java` will replace this range `0..255` by an appropriate type (currently `int`). This entails some changes in `web2java.y`, to allow ranges for procedure and function parameters, as eg in

```
procedure p(x:0..255);
```

Presently all ranges are replaced by `int`, since Java is rather strict about type conversion, and requires casting where C does not.

4.3 Input/Output

On the whole, Java I/O is closer to Pascal syntax than is C. Thus

```
write_ln(term_out, 'value is ', v);
```

in Pascal becomes

```
System.out.println("value is " + v);
```

in Java.

However, we follow `web2c` in leaving this translation to the post-processor — in our case the script `javafix.perl`.

This Perl script looks for 'words' starting with `jT`, and deals only with these. Thus the translation above is implemented through the definitions

```
@d term_out == System.out
@d write_ln == jT_print_ln
```

The only unusual feature of Java I/O is that most I/O statements must be contained in a `try` statement, which must be followed by a `catch` statement to catch any I/O 'errors'. However, this is perfectly straightforward, as may be seen in figure 2, which shows an I/O function from `dvitype.ch`.

For simplicity, `reset` and `rewrite` are defined in the 'TEX library class' `TeXlib.java`. Thus

```
reset(web_file);
```

is replaced in `tangle.ch` by

```
web_file:=TeXlib.reset(web_name);
```

5 Conclusions

Writing a Java change file is a straightforward exercise, following the lines suggested above.

Whether it is time well-spent is another matter! As we have said, the resulting programs are painfully slow; and the viability of the exercise must depend on the development of true Java compilers.

5.1 Project details

All the material in the project is in the public domain, and can be retrieved by anonymous FTP from `ftp://ftp.maths.tcd.ie/pub/TeX/javaTeX`. (It is hoped later to submit the programs to the CTANs.)

Any suggestions and contributions are very welcome.

A mailing list has been set up. To join this, email `majordomo@maths.tcd.ie` with the message (not heading)

```
subscribe javatex
```

The material forming the project is also available through the mailing-list. You can obtain this file, for example, by emailing `majordomo@maths.tcd.ie` with the message

```
get javatex javaTeX.tex
```

To obtain a list of available files, send the message

```
index javatex
```

For information on the working of `majordomo` send the message

```
help
```

to `majordomo@maths.tcd.ie`.

VI Malcolm's Gleanings

Malcolm Clark

0.2 Ligature trivia

In the *TUGboat* **17**(2), Haralambous and Plaice, taking a slight diversion present a whole galaxy of glyphs which could find a role in their Ω system. In passing it is intriguing to read the following article by Richard Kinch and speculate on the rivalries which must underlie that contribution. One of Haralambous and Plaice's offerings are the "french ligatures for st and ct". I was a little startled, since I hadn't realised that these *were French*. I'd seen them in books from 1800 in English, and, although I've since not been able to confirm it, I think that Eric Gill's Joanna had them too.

By an amazing coincidence, the topic of these ligatures cropped up on the `typo-l` discussion list and this stimulated me to pursue a little research, from which I concluded that Sweynheym & Pannartz' cut roman typeface in about 1467 which had the ct, and a long-st ligature. I'll ignore the gothic faces which had as many ligatures as scribes were used to using in hand lettering. Sweynheym & Pannartz were German and worked in Italy. Like good type design, it's all in the details.

0.3 TUGboat

I usually manage to make some veiled comment about the timliness of *TUGboat*, but this time I am pleased to direct your attention to a *TUGboat* Web page. Try out: http://www.halcyon.com/clcook/tugclndr.htm

0.4 The classic fonts

Another discussion on the `typo-l` list involved the origin of the 'classic' fonts which were found on the first Adobe *i.e.* POSTSCRIPT laser printers. (Note I am using 'classic' in the same sense that a Citroën 2CV might be described as 'classic'—dangerous firetrap thought it undoubtedly was.) Andrew Boag pointed out that the IBM 4250 600dpi electro-erosion printer shown at DRUPA in 1982 had Monotype hand-edited bitmap fonts which included Courier, Helvetica (sub-licenced from Linotype), Palatino and Times NR.

David Lemon added that the basic LaserWriter fonts were selected in discussions between Adobe and Apple. A representative from the Adobe Type group tried to keep it 'reasonable', but , Steve Jobs had to be talked out of including ITC Gorilla, and ITC American Typewriter almost became the "typewriter" design, until Adobe pointed out that it is not monospaced. When the other fonts were added some time later, Jobs had his revenge in the form of ITC Avant Garde. Still, he was and may still be one of the most gifted computer designers of the last 20 years: he gets to screw up in chosing fonts.

0.5 Slip between cup and lip

A while back I wrote about the proposed math handling of HTML. When the standard was published, the maths had disappeared. Nevertheless, Dave Raggett does describe some of the markup and the hopes in his book *HTML3, electronic publishing on the World Wide Web*, Dave Raggett, Jenny Lam & Ian Alexander, Addison-Wesley. I enjoyed the book's informal style, though I wouldn't wish to be seen in socks like those.

0.6 TEX lauded at Seybold

I exaggerate slightly. Conrad Taylor, writing for the Seybold Report on Publishing Systems *What has wysiwyg done for us?*, suggests that "most of the H & J (hyphenation and justification) algorithms in desktop publishing programs are lamentable, particularly when compared with TEX". We knew that, but a small accolade to that man.

0.7 TB-L

Tim Berners-Lee, creator of the World Wide Web, was recently made a Distinguished Fellow of the British Computer Society. I was surprised that he is **not** seven feet tall.

After his presentation Tim gave a talk on "The World Wide Web—Past, Present and Future". A transcript of the presentation is available as http://www.bcs.org.uk/news/timbl.htm

I was able to ask TB-L a little about the incorporation of HTML into the Web. I asked why he chose this way of expressing content (rather hoping he would endorse the wisdom of separating form and content), but apparently it was much more simple than that. People at Cern were accustomed to those little angle brackets, so he gave them something with which they were familiar, and which was simple. Thank goodness Cern wasn't using Wordperfect!

VII TUG'96 — fun and profit in Dubna

Robin Fairbairns

Travel

I was instructed by the UK TEX Users' Group committee to attend TUG'96 to represent the group. On a flight that cost less than half as much as BA's cheapest, the trip to Moscow was pretty simple. Russian immigration was tedious, but more ominous was the requirement to fill in a form declaring how much foreign currency (and Roubles) we were carrying, as well as the amount of ammunition and general weaponry, not to mention religious icons and the like. One has grown used to being asked whether one is carrying weapons at the *beginning* of a flight ...

Eventually the whole of the TUG board was being led by Irina Makhovaya of CyrTUG across the airport car park to find the minibus which was to take us to Dubna. If the ones we encountered were typical, Russian roads are pretty awful. The forests we drove through were predominantly birch and pine (the proportions vary). The birches have startlingly bright white bark, which catches the sun; when mixed with the pines' dark bark, the effect is really striking. We kept passing really fancy signs for villages, but then saw almost nothing of the villages from the road; the sign for Dubna is a massive steel thing with DUBNA in 3-D steel letters supported on wires.

Checking into the hotel was simple; we were given little chits for each day's breakfast, and room keys with convenient bottle-openers attached. The meal in the hotel wasn't exactly exciting, but the CyrTUG contingent invited us for tea (in the Russian style) afterwards; we had arrived, and were feeling welcome already.

The meeting location — JINR

The Joint Institute for Nuclear Research, where the meeting was to be held, is 'pretty near' to the hotel, says Michel Goossens. Actually, by Westerners' standards, it's a fair walk. But the centre of Dubna is spaciously laid out in the middle of the forest, and the walk is quite pleasant. The contrast between the (relatively) modern buildings and the beautiful trees and flowers was striking indeed.

There were 40th anniversary celebrations going on, so that there were any number of brash posters around the town, and the hotel was teeming with Russian athletes (outnumbering the TEXies by a significant proportion: the town-centre hotel that we were in was too expensive for many of the Russian delegates).

At the entrance to the Institute is a gate-house, manned by soldiers. On that first day, they were actually touting their automatic rifles, but later on they accepted that we would meekly do what we were instructed to do. They checked our passports, and lengthily, laboriously, checked that we were indeed in the list of people to be allowed in without benefit of official passes. We were taken to the LCTA[7] building where the meeting was to happen, and climbed to the third floor (of seven) for our first TUG board meeting. After formally opening business, and agreeing an agenda, most of us were dragged off to Vladimir Korenkov's office to be interviewed by local TV. I had determined to be brief (I didn't have much of a voice at the time), but Michel and Korenkov spoke for what seemed an unconscionably long time (in Russian). How much of this interminable interview made it to air I don't know: I would like to think my small contribution made it, because I was wearing my research group's T-shirt ...

For lunch, we went to the Institute's canteen, and learnt how the other half live: the food wasn't terribly classy, but it was extremely cheap (when the conference proper started, we had a separate area with a fixed menu: we weren't told how much this cost).

Dubna is on the banks of the Volga (it's actually near the confluence of the Dubna river with the Volga, and takes its name from the smaller river). On the Friday evening we had dinner in the hotel and wandered off to watch the sun setting over the river; we then discovered the intriguing Pelikan bar — a plastic tent with a refrigerator and a rather loud hi-fi system; it was the nearest place we discovered that sold Russian (as opposed to imported) beer.

[7]Laboratory of Computer Techniques and Automation

The Conference

Our Saturday was spent mostly indoors, in board meetings, eating, and variously milling around. Then, in the early evening registration started, and we received our pack of conference stuff, including a TUG'96 mug, with that curious accordion-playing Russian lion on it. No T-shirts on offer, though: we're told that Russians don't go for them, and so they didn't occur to the organising committee.

Once registration was well under way, there was a little tea-party. Russians say (we're told) "you can have too much Vodka; you can't have enough tea". The Russian method of preparing tea grows on one — I might even feel moved to use it myself: the Samovar is a much simpler object than I'd ever imagined, and can easily be substituted by a simple kettle. After tea, we continue the board meeting by the Volga, amid crowds of people enjoying the evening sunshine.

On Sunday morning, we have the "opening ceremony" of the conference. This consists of a series of grand speeches (many of them in Russian, translated for us ignorant foreigners). We had an intriguing history (in English) of the institute, which was curiously old-fashioned in tone "the biggest ... the first ... the highest flux ... ". The fuel rods in one of their reactors stay there for 10 years!

In this abbreviated account, only a few of the more striking papers get covered ...

- Yannis Haralambous gave us one of his perorations, this time on ligatures in Arabic. Yannis is *so* erudite, but when he goes off on one of these tacks about the æsthetics of typesetting it's hard to keep up with him (though I find him totally fascinating).
- Karel Piska (who says he's a sort of "collector of alphabets") spoke about the usage of Cyrillic alphabets, the paths through which they came to it, and the problems that arise from their slightly different treatment of the alphabet itself.
- Sergei Znamenskii discussed the issues arising from the different standard encodings in use for Cyrillic throughout Russia. This was a repeating issue for the conference; how does one exchange information between systems, given that there are all these different encodings in use. The normal ones are KOI-8 (the '8' being pronounced in Russian as 'vosyem': a standard from the 'old days'), ISO 8859-5, and two Microsoft code pages. Reading a file written in one code as if it were in another can often produce a *partly* sensible result, but it's obviously never going to be acceptable.
- Jörg Knappen spoke on his latest version of the DC fonts, pushing his message of stability in advance of the release of the final 'EC' version[8]. He also discussed the text companion fonts, which contain symbols whose appearance should reasonably change to match the surrounding text font. Jörg requested samples for additional symbols for these TC fonts.
- Jörg also presented Fukui Rei's paper on the new TIPA phonetic fonts: an impressive piece of work, with accompanying macros that permit straightforward use. Rei's encoding has been adopted as a LaTeX standard (T3).
- Olga Lapko spoke about the encoding problems that confront the font-designer who aims to support Cyrillic; the background to the problem is the vast range of languages that Cyrillic covers, another aspect of the subject that Karel Piska was addressing.

Monday, and how not to chair a session

Monday was the one appearance I made on the podium. This could have been difficult (I had no voice) but I could make myself understood by using the microphone. The session was thrown into complete confusion because I had an incorrect version of the session timetable (I never found out why), with more time allowed for Yannis Haralambous (on his new fonts) and less allowed for Richard Kinch (on Unicode encoding issues) than either was expecting. So I had to extemporise a short discussion period at the end of the session, on encodings. This discussion didn't really get off the ground; I had noted several questions I wanted to ask of the morning's speakers, but no-one else (apparently) had done so. And when I had asked all my questions and had answers to them, the session (sort of) dried up ...

The afternoon was rounded off by the TUG business meeting, at the start of which Jörg Knappen led a group of Russians out, to discuss the planned LaTeX T2 encoding for Cyrillic. (This was the one major question I had failed to get under way in my extempore discussion session.) Michel's presentation seemed to go down reasonably well with the members, and there was some discussion, but it was curtailed by an urgent summons to the canteen for dinner.

And so back to the hotel, whence to the Pelikan for a beer to drink by the banks of the Volga, finding and skimming stones (there are rather few stones of any sort, let along flat ones, since it's not a sea shore, for all its sandiness). And

[8]Which has since been released

when the sun's set too far to continue, back to the Pelikan to sit with a group of other delegates, drink a final beer, and shout conversations over the loud music and to the accompaniment of a drunken dancer who occasionally lunged at people in a sort of conversational way (and invariably missed).

Sergiev Posad — the *big trip*

Tuesday morning's session was short (*and* I was late for it, having been told the first speaker hadn't arrived). Laurent Siebenman was speaking when I arrived about the concept of using DVI as a document distribution medium, but of course I missed some of his reasoning. I'm sceptical.

Then off to a slightly-hurried early lunch and thence to Sergiev Posad by bus. It's a major centre of the Russian Orthodox Church, including a monastery, a seminary, etc. Our tour took in three 'cathedrals' and a museum which included some beautiful illuminated manuscript books.

Our journey back was far from smooth — a startlingly loud puncture delaayed us for ages. But when we got back to the hotel, we found that a call has gone ahead of us to JINR, and our dinner has been put in huge pots and brought to the hotel. We ate it as a picnic on a bit of meadow land between the hotel and the Volga.

And how do you follow that?

The Wednesday morning started with *two* papers from Kees van der Laan: I never know what to make of this man ...

The first paper is on how to do (what seem to me) trivial graphics in TEX (native) using the good old 'turtle' model. This is fun, but isn't taking us anywhere much: who gets to using TEX when they're still of an age to need turtle graphics? (Native) graphics in TEX *are* a problem, but there are reasonable solutions in a variety of areas, and almost anyone nowadays will go to encapsulated PostScript for anything of any significance.

Kees' second paper develops from one single idea: that of doing METAFONT (or META O T) graphics in three dimensions and imposing a projection transformation at a late stage. This is neat, but he doesn't seem to be thinking of anything but decorative effects.

The rest of the morning (after the coffee break) is taken up with two new PDF-related projects. The first presentation, of TEX2PDF[9] is by Petr Sojka. TEX2PDF provides an alternative output mechanism (as a change file on the sources) which creates PDF output in place of DVI. This is, of course, the *right way* to do a hypertext-ish output; hypertext inevitably suffers from the same problems of context as the colour package, which we're familiar with from David Carlisle's accounts. Knuth spoke at TUG'95 of his surprise that so few people have used TEX as the basis for radical modification; Sojka has taken him at his word, and it was pleasing to hear that Knuth had approved of the work when he visited Brno earlier in 1996.

The second presentation was from Sergei Lesenko, who is the original author of the partial Type 1 font downloading code that appears in the latest alpha-test dvips. Lesenko is developing a dvi2pdf, based on the code of dvips. As I explained above, this isn't *really* the 'right' way to do the job (any more than the established TEX – dvips – repere – Distiller), but one can imagine it 'catching on' more quickly than TEX2PDF, simply because of the world's entirely justifiable reliance on the stability of TEX itself.

Neither of these projects is complete, and neither could be accepted with their present specifications even if complete:

- the precise nature of Sojka and Thanh's new primitives wasn't clear, but what *was* clear didn't please many, and
- Lesenko's `\special` commands don't conform to the HyperTEX conventions, nor does their syntax look like that defined by the TWG on device driver standards.

Nevertheless, I take heart from the fact that substantial new projects continue to spring up in our cosy little TEX world!

In the afternoon we had a trip to a neighbouring town called Kimri, to visit the town museum. The bus again collected us from the institute, but by the time we arrived at the gate, the rain was beginning to be serious, and we had to get out and file through the gate-house so that the bus could be 'checked' (an operation that, naturally, took rather little time ...). The road took us out of Dubna *under* the Moscow-Volga canal, past an enormous statue of Lenin and then along the top of the dam that makes the "man-made (Moscow) sea" which (as I understand it) is on the Volga. By the time we arrived at Kimri, getting from bus to museum was an occasion to sprint through huge puddles.

On the way back to the institute for dinner, the weather is more terrible still. The bus driver agreed to wait (for no longer than 20 minutes) while we ate our meal, which meant that we got a lot less wet than we might otherwise expect.

[9] See the paper elsewhere in this issue

The guards weren't playing: we had to get out and show our passes and passports, filing through the pouring rain, on the way in, and again on the way out. The brilliant arrangement this time was to spend significant time 'checking' the bus, so that the guard house became packed and the (real) workers at the institute couldn't go home from work on their bicycles.

So to the hotel, and to Irina Makhovaya's birthday party in her hotel room. Wow! — food, drink, toasts, all flowing freely until (after three hours) tea arrived with the inevitable Samovar. Lots of tea and then dancing, until (at midnight) someone said we shouldn't have music in the hotel after 11 p.m. So out to dance in the open air to the sound of the tape player in Professor Pankratiev's car. Until about 1.30 a.m., when the military police arrived: they wanted to impound the car and arrest the Professor. A slight dampener on the enthusiasm for dancing (Richard Kinch disappeared into the undergrowth), but the Russian women present pled with the police (in relays). Eventually the word came that we are to move the car about 5 metres back so that it's *this* side of the 'No entry' sign ... and a little while later, Pankratiev was no longer in jeopardy. Eventually the police go away (and Richard reappears!). When I left (after 2 a.m.), the dancing was still in full swing; I've demonstrated to myself that even free-flowing Vodka doesn't make me a competent dancer!

The last day

The conference itself was scheduled to finish on Thursday 1st; the morning session was to be papers (as normal), then a closing ceremony.

Andrei Slepukhin swapped places with Kees van der Laan, so that Yannis Haralambous (who was leaving early) could hear Slepukhin's ideas on multilingual processing. Slepukhin may have a point, but if he does, he seems me to miss it himself; on the whole, I need to read a paper (there wasn't one, even in the pre-prints). Michel Goossens re-did (and still further extended) his talk about LaTeX↔HTML.

Then after coffee Kees van der Laan (again) talked about his BLUe's format; Kees is a polished presenter, but I remain unconvinced. More unconvincing still was Astrelin, who was (as far as one could tell) talking about a C++ class library implementing a few rather trivial graphics functions together with some serious unresolved research issues. Until Pankratiev[10] spoke a little later, there was no context whatever for Astrelin's work; Pankratiev's project is putting together a standard harness for the use of TeX throughout Russia, and Astrelin's work is a small part of it. Between Astrelin and Pankratiev, though, was another piece of solid down-to-earth stuff from Berdnikov's group — not exactly earth-shattering, but plainly addressing real needs of their user community. I was impressed by all three papers that came out of Berdnikov's group.

Then we had our closing ceremony. We had asked for nominations for the Cathy Booth prize, on the basis of "what would affect your work most in the next year". I was sure that either Sojka or Lesenko should get the prize; in the event Lesenko's showing was poor, while Sojka was the clear winner. However, since Sojka had a prize last year, and since the prizes we had on offer were rather infra-dig for him anyway (for example, as president of CS-TUG, he gets that group's complimentary copy of *Baskerville*), he accepted the prize on behalf of his student Han The Thanh. The top prize awarded by TUG was to Berdnikov as representative of his group. Well-deserved, as I said; solid down-to-earth work. CyrTUG gave Michel Goossens a big book (presumably about Russia — all I could see was that it was in Russian).

Then after another rather rapid lunch we went off to the bank of the Volga, just up the path from the hotel, to catch the boat for our picnic trip. Beautiful weather, the sort of boat that plies many a Western river with a lot more passengers, so one could walk around and talk without inconveniencing one's fellows, and a leisurely trip up the Volga to somewhere that wasn't identified (to me, at least), where we turned around and sailed back down again. It's hard to give a clear picture of this trip. I took a lot of photographs of the assembled TeXies; Olga Grineva (of Berdnikov's group) went round and collected signatures from every participant on the back of her copy of the group photograph of the conference.

Two things, off our boat, stick in the mind. The first is the perfectly ordinary idyll of a camp-site in amongst the trees on the banks of the river; people enjoying a holiday, with a canoe and a perfect beach to swim from. The second is the totally incomprehensible old man who was wandering around a vast barge as it chugged along in the opposite direction to us, stopping from time to time at one or the other of the piles of sand on the barge and tipping a shovelful of the sand into the river.

And eventually we had come back to the confluence of the Dubna river with the Volga, where we turned up the Dubna for a short way to a picnic site at Ratmino. A fantastic spread was set out for us — food a-plenty and masses of

[10] Whose talk came as a surprise, as it too hadn't made it to my copy of the programme

booze. We all tuck in, and then realise that they've got huge barbecues running, and we're being offered shashliki — huge metal skewers with giant lumps of lamb on them. Of *course* there was a speech or two, and as a result there were toasts. We only had plastic mugs to toast with, and they don't 'chink' . . . A small group struck up folk singing, others just talked and enjoyed the early evening.

Moscow, and the end of the party

Friday was a trip to Moscow which combined a tour with dropping people off for their travel home. The bus turned off into Sheremetevo (2: the international airport), and we said good bye to the first bunch. Then we went into the centre of the city and met a guide, strikingly dressed to look like a cross between a cartoon bee and a Latin-American slinky dancer. We went to Red Square, and observed the outside of Lenin's mausoleum, of St. Basil's (extraordinary) Cathedral and, of course, the Kremlin. Then back to the bus and a long drive around the city, where we were for ever seeing 'panoramas' of this, that or the other. There were striking things in all this, like the cathedral being rebuilt in the shell of the swimming pool that replaced the original cathedral on the site, and like the beautiful old houses that have survived since the rebuilding of the city after Napoleon's devastation in 1812.

After a stop at a touristy-shop, we climbed the Lenin Hills (the promontory on which Moscow University stands), dropped a few people off, and then stopped for what I would call a *real* panorama of the city. The ski-jumps up here are (strangely) on the itinerary of newly-weds, who climb the rickety iron steps in their finery to look out over city from a better vantage point still. Then on to the war memorial that was dedicated in 1995 in remembrance of the of the 1941–45 war. This is a most astounding piece of architecture, achieving a dignity that one doesn't, somehow, expect. The arrays of fountains were spectacular; the singing in the memorial church was wonderful, the sculpture was striking. I loved it.

But at the end of that little tour, the party started breaking up in earnest, and within the hour there was almost no-one left on the bus.

1 1995/96 Accounts

Accounts for the period 1st August 1995 to 31st August 1996

INCOME	
Membership Income	
1995 TUG Subscriptions	32.50
1996 TUG Subscriptions	3244.00
1995 UKTUG Subscriptions	58.00
1996 UKTUG Subscriptions	2827.00
EmTEX/OzTEX Subs	165.00
EmTEX/OzTEX New Subs	355.00
Handling	255.50
Inv Charges	55.00
CD Services	
4AllTEX CD ROM	1719.37
UNIX CD Rom	477.24
Book sales	
Book sales	405.98
EDMAC book	76.00
Group Meetings	
20/3/96 Meeting	745.00
LATEX3	48012.89
Miscellaneous	
Bursary	95.04
General	404.64
Bank Interest	655.30
High Int A/C Interest	497.96
TOTAL	60081.42

EXPENDITURE

Membership Services	
Discs	445.48
Books	536.12
Baskerville	1622.98
CD Roms	48.40
Member	79.12
UNIX CD Rom	1000.00
Meetings (inc committee)	
1/4/95	245.00
1/6/95	298.70
18/10/95	213.08
25/1/96	120.80
15/2/96	60.21
20/3/96	144.40
9/5/96	90.56
LaTeX3	16318.53
Miscellaneous	
Bank Charges	2.00
General	1155.35
EuroTeX	220.00
Dubna	305.69
T Shirts	218.75
TOTAL	23125.17

Bank Assets	
UKTUG Funds	16756.24
TUG Funds	3244.00
LaTeX3	38005.83

Peter Abbott, Honorary Treasurer

VIII The UK TEX Users' Group

The 1995–96 UKTUG committee

R. Fairbairns	Chair
P. Abbott	Treasurer and Membership Secretary
D. P. Carlisle	Committee Secretary
M. Clark	Meetings Secretary

K. Bazargan; S. P. Q. Rahtz; M. D. Wooding.

Book Discounts for UKTUG members

We have arrangements with Addison-Wesley for their well-known TEX-related publications, and with International Thomson Publishing to supply any of the very excellent O'Reilly & Associates Inc. series of books to members.

The agreed list of books, together with the discounted (at least 20%) price, is distributed occasionally with *Baskerville*, but is always available from the Treasurer, Peter Abbott. The quoted price includes the cost of postage and packing.

We are only allowed to offer this service to **current** members of the UK TEX Users' Group and/or members of TUG. Please send your order and cheque (in UK £) to Peter Abbott (address in *Baskerville* masthead). Make cheques payable to 'UKTUG' please. Books from Addison-Wesley are delivered direct but books from O'Reilly will be routed through UKTUG. *In all cases* please notify Peter Abbott by email, phone, fax or letter when books are delivered. This service is unfortunately not a speedy process

IX Obtaining TEX

edited by Peter Abbott

From the network – CTAN

The UK TEX Archive on `ftp.tex.ac.uk` is part of the CTAN (Comprehensive TEX Archive Network) collaborating network of archives on the Internet organised by the TEX Users Group.

The CTAN archives run an enhanced *ftp* server which supports dynamic compression, uncompression, and archive creation options. Fetch the top-level file `README.archive-features` for information. The server also supports site-defined commands to assist you. Please read `README.site-commands` for a brief overview.

Please report any problems with CTAN archives via email to `ctan@urz.Uni-Heidelberg.de`.

The main directories which make up CTAN are listed below; readers are referred to Graham Williams' *TEX and LATEX Catalogue* which is available from CTAN as `help/Catalogue/catalogue.html`

biblio	bibliography-related files, such as BIBTEX.
digests	back issues of TEX-related periodicals
dviware	contains the various `dvi`-to-whatever filters and drivers
fonts	fonts, both sources and pre-compiled
graphics	utilities and macros related to graphics
help	overviews of the archive and the TEX system
info	files and tutorials which document various aspects of TEX
indexing	utilities and related files for indexing
language	material for typesetting non-English documents
macros	macros packages for TEX and style files
support	programs which can be used in support of TEX
systems	complete system setups, organized by operating system
tools	the various archiving tools used on CTAN
web	contains WEB-related files and utilities

Unix – CD-ROM

GUTenberg and UKTUG, in collaboration with TUG and NTG, have produced a plug-and-play CD-ROM based on Thomas Esser's teTEX distribution. As it uses the ISO 9660 standard, the platform-independent files can, in principle, be read on all operating systems which are compatible with that format.

Unix executables for the following platform/operating system combinations are included: Digital alpha-osf (2.0 and 3.2), Hewlett Packard hpux (9.01 and 10.01), Intel i386 bsdi2.0, freebsd (2.0.5 and 2.1.0) netbsd (1.0 and 1.1), Intel i486 (linux and linuxaout), m68k (linux, linuxoldld, and nextstep3), mips (irix 5.2, 5.3 and ultrix4.4) IBM RS6000 (aix3.2 and aix4.1.1) Sparc Solaris (2.4 and 2.5) and Sunos 4.1.3.

For full details see the article in *Baskerville* 6.2.

The CD is available to members of TEX user groups at £15 and to non-members at £25. Order the disk from Peter Abbott; see the section 'PC and Mac disks' for details.

DOS – CD-ROM

UKTUG distributes the comprehensive 4AllTEX CD-ROM, created by the Dutch TEX Users' Group (NTG), now in its 3rd edition. This costs £25 for 2 CDs, and is for DOS users.

PC and Mac disks

The UKTUG distributes an emTEX kit for PCs, and an OzTEX kit for Macintosh. The cost covers copying and postage costs, together with the shareware fee for OzTEX (and other Mac programs) and Eddi4TEX. Each set costs £30, and is available from Peter Abbott, 1 Eymore Close, Selly Oak, Birmingham B29 4LB. Cheques must be payable to 'UKTUG'. Please note that this service *is available to UKTUG members only.* Each set comes with an installation guide, and (at least) full TEX and METAFONT, a previewer, a PostScript driver, and CM fonts. Two update disks a year will be sent out automatically, with the current version of LATEX 2ε, and other goodies. A subscription service is