Using PostScript with TEX

Alec Dunn, University of Sydney*

Several articles in TUGBOAT have demonstrated methods of drawing simple graphics using dots and horizontal or vertical rules. LaTeX provides a complex `\picture` environment which uses special fonts, containing arcs and diagonal lines, to draw more elaborate figures. These methods have the advantage of portability because they use only facilities common to all TEX implementations and most dvi processors. But they have the disadvantages of limited scope, difficulty of use, TEX memory limitations, and lack of an interface to other graphical systems. A more versatile approach is to link a graphical language, such as PostScript, into TEX, using the `\special` command. A recent TUGBOAT article [1] describes this approach, and several other sites have used this method.

This article describes work on these lines at the University of Sydney. Some small improvements on the methods of [1] make the TEX–PostScript interface simpler and more foolproof for users. Also, the use of this system with a general-purpose graphics package and with a specially-written Macintosh PostScript generator are discussed.

1 The PostScript language

This section briefly describes the features of PostScript relevant to its usage with TEX, for the benefit of readers unfamiliar with the language.

PostScript is a language for programming two-dimensional graphical and typesetting operations. It is independent of any brand of printer, and it has been implemented on several models of laser printers and typesetters. Unlike TEX, PostScript is a proprietary product, owned by Adobe Systems Incorporated. The following small example demonstrates the operation of PostScript:

```
newpath 113 92 moveto
116 96 lineto 113 100 lineto
110 96 lineto closepath stroke
```

To understand this example you should know that commands apply to the numbers preceding them, and you can safely ignore the **newpath** and **stroke** commands. The example code draws a diamond shape (◇) starting from coordinates (113, 92) and with sides of length 5 units.

Note that we haven't specified what the units are. PostScript lets you define and re-define your own units, so the diamond could be drawn at any size by suitable definition of the units. This is one of the most important features of PostScript (for present purposes) — everything, including text, is perfectly scalable. Similarly, the entire coordinate system can be shifted at any time, so the point (113, 92) can be placed anywhere on the page.

Other valuable features of PostScript are demonstrated by the example. Nowhere in an ordinary PostScript program is there any reference to the printing hardware: the example will produce the same results on a 300 dots/inch laser printer and on a phototypesetter; raster conversion is performed in the printer. And the code is entirely in visible ASCII characters and so is fully portable and communicable between different computers and operating systems. PostScript is just as portable as TEX.

2 \special in TEX

The natural way to combine TEX and PostScript is to use the TEX `\special` command to pass PostScript code to the dvi processor (TEX itself has no use for PostScript, since TEX is only concerned with placing objects on pages, not with actually imaging those objects).

To obtain a PostScript graphic, using our system, the TEX user puts a command of the form

*Comments should be sent to A Dunn, School of Electrical Engineering, University of Sydney, NSW 2006, Australia, or via ACSNET to alecd@facet.ee.su.oz.

```
\special{PF filename height width}
```

into his or her TEX file at the point at which the lower left corner of the graphic is to be placed (which may be inside a float). The PF is just a keyword, chosen by us, to distinguish this kind of \special command from any others which may be used (using a keyword for this purpose is recommended in *The TEXBook* page 228). The PostScript code is in file *filename* — we don't insert the PostScript code itself because it tends to be verbose and the \special command uses TEX memory. The *height* and *width* arguments define a bounding box for the graphic. (Historical reasons compelled the unfortunate choice of *height*, *width* order instead of PostScript's x, y order). An example of this form of \special command is:

```
\special{PF diamond 4mm 3mm}
```

which is how the diamond example above was drawn.

The \special command occupies no height or width (since TEX can't interpret its contents), so in practice it must be supplemented by glue commands, for example:

```
\hbox to width
   {\vrule height height width 0pt
    \special{PF filename height width}\hfil}
```

Of course, this can be simplified for users by a suitable macro.

This is the full extent of TEX's role — most of the work in combining TEX and PostScript is performed by the dvi processor. In this case the processor, called Dvi/PS, was written by us for Vax/VMS machines and is proprietary to the University of Sydney. If you have the source code to a different dvi processor you may be able to adapt it to handle the \special command (naturally, it must be driving a PostScript device!).

3 \special in Dvi/PS

When Dvi/PS encounters a \special command, it already knows the coordinates of the lower left corner of the graphic (by the same mechanisms by which it knows where text is to be placed); the PF keyword and the *filename*, *height*, and *width* arguments follow in the dvi file. Dvi/PS then scans the PostScript file to find its bounding box, which, if the file conforms to the Adobe structuring conventions [2], will be given in a specially-formatted comment line.

Knowing the bounding box of the graphic in its own PostScript coordinate space, and the desired location and bounding box in the page's space, Dvi/PS computes a suitable transformation matrix and sends this to the printer before sending the contents of the PostScript file. This relieves the user from having to know anything about the size of the PostScript graphic, or about its coordinate system — the graphic will always appear where the user asked for it and at the size he or she asked for.

It is possible, even likely, that the user-specified bounding box and the graphic's PostScript bounding box will have different aspect ratios. PostScript allows different scale factors horizontally and vertically, so the graphic could be fitted exactly to the user's bounding box. But most users don't want their graphics distorted in this way, so Dvi/PS computes only one scale factor, according to the limiting dimension (horizontal or vertical), and applies that to both dimensions, adjusting the origin transformation so that the graphic will be centered in the non-limiting dimension.

An example may make the whole process clearer. Suppose the TEX file contains

```
\special{PF cat 180pt 200pt}
```

so the user is asking for a graphic of 200pt×180pt (in normal x, y order). Also suppose the PostScript file cat has a bounding box of (110, 50) to (190, 140), after conversion to TEX points. The PostScript width is 80 pt and height 90 pt. The limiting dimension is the vertical, allowing a scale factor of 2×, which leaves 40 pt of white space to be taken up in the horizontal. So Dvi/PS emits PostScript code to shift the origin horizontally by $-110+40$ pt and vertically by -50 pt and then to magnify by 2. The graphic, as printed, will be 180 pt high and 160 pt wide, horizontally centered in its bounding box (see Figure 1).

If either dimension is given as zero in the \special command Dvi/PS ignores the corresponding PostScript dimension in its scaling calculation and doesn't center the graphic, leaving it left- or bottom-justified. Combinations of zero and non-zero arguments, in suitable macros, give most of the facilities users want.

It is not an error for the PostScript code to draw outside of its supposed bounding box, since users

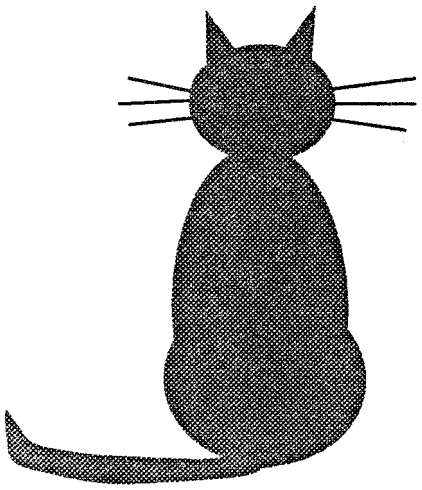

Figure 1: cat

may want to achieve special effects this way. Dvi/PS doesn't produce code to clip the image, nor does it draw the bounding box onto the page.

A PostScript file to be used in a `\special` command should not depend on another such file, since it is not known in what order Dvi/PS will process them. Also, text should use only native PostScript fonts, not downloaded fonts, which Dvi/PS may not yet have loaded. In practice these are not serious limitations.

4 Error handling

Of course, errors are possible: the PostScript file may be missing, or it may not conform to the structuring conventions and so not contain a bounding box specification. Dvi/PS copes with such problems by leaving white space if the file is missing, or not performing any coordinate transformation if the bounding box is unspecified.

But it is impractical for Dvi/PS to attempt to control errors in the PostScript file itself — once Dvi/PS sends the PostScript file to the printer it effectively hands over all control to that file. For the purposes of this system we can define a *well-behaved* PostScript file as one which leaves the printer in the same state as it found it, except for the coordinate system (which Dvi/PS always restores) and marks added to the page (which was the purpose of sending the file). For example, a file which executes the PostScript **showpage** command (which prints and ejects the page) is not well-behaved.

Unfortunately, few PostScript files are well-behaved! Most software with PostScript output aims to produce a complete specification of the final hard copy, which is the purpose for which PostScript was conceived, but here we are using it for "graphical procedures". This problem is the major limitation in using PostScript with TEX — to get well-behaved PostScript has required writing our own software.

5 Sources of PostScript code

You can write PostScript code yourself with any text editor, but this is impractical for graphics of any complexity. At the School of Electrical Engineering we have extended our general-purpose graphics package so that it can produce a well-behaved PostScript file instead of driving a graphics device. Most of the graphically-oriented software written in the School in the last five years can now be used together with TEX, and several theses and reports have been printed almost entirely without cutting and pasting.

Graphics produced by applying programs to data are very useful in engineering, but we also need to be able to just *draw* and have the drawing translated into PostScript. *MacDraw* on the Macintosh is ideal for simple engineering drawing, and *MacPaint* for freehand drawing. But it is tricky to make the Macintosh produce a PostScript file, and almost impossible to convert that file into well-behaved PostScript, so we have written a program, *PostScript from Mac*, which can convert *MacDraw* and *MacPaint* files directly into PostScript files suitable for inclusion in TEX documents. (The **cat** example, above, was drawn with *MacDraw* and converted by *PostScript from Mac* into a PostScript file of about 1 Kbyte.)

PostScript from Mac has proved surprisingly popular with the academic staff of the School and has revealed an unexpected demand for a means of *easily* including good quality graphics into TEX documents.

References

[1] H. Varian and J. Sterken, "MacDraw Pictures in TEX Documents", *TugBoat*, vol 7 no 1.

[2] *PostScript Language Reference Manual*, Addison-Wesley, Reading, Mass. Appendix C, pp 263ff.