I suffered plenty of setbacks *en route* to a working set of circular macros. Sometimes the results of faulty macros were interesting in their own right. Take a look at the best such mistake in Figure 9.

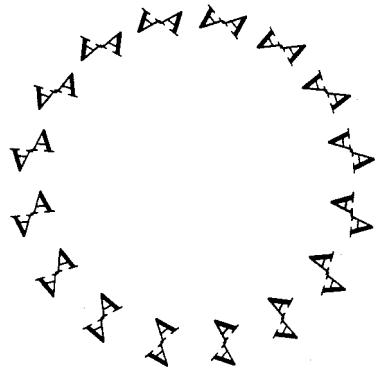

Figure 9. Mistake.

If you try this stuff yourself, note that circular typesetting may throw your previewer and device driver for a loop (apt?). You have been warned.

◇ Alan Hoenig
17 Bay Avenue
Huntington, NY 11743
(516) 385-0736

Graphics

On the Implementation of Graphics into TEX

Gerhard Berendt

1 Abstract

The problem of implementing more complex pictures than are provided by the LaTeX `picture` environment into a typical PC version of TEX is discussed. In the first part of the article (Sections 2 and 3) a solution is presented which circumvents the usual limitation of the restricted main memory of TEX and respects the moderate hash size of the PC versions of TEX. This solution remains, however, totally within the frame of TEX. In the second part (Sections 4 to 7) a solution to the problem is given which makes use of PostScript within the TEX environment.

2 Introduction

While TEX is a very powerful tool for producing mathematical and technical texts, it has its well-known deficiencies as far as the implementation of graphics is concerned. The problem is twofold:

- The hash size of about 3000 for a typical PC version of TEX limits the complexity of macro packages which implement graphics. It is, e.g., impossible to add the rather comfortable PICTEX macro[1] package to LaTeX because of an overflow of the hash size. In order not to surpass the given hash size, it is therefore necessary to use a more moderate graphics macro package, if the LaTeX environment is obligatory. Our solution to this problem will be presented in the next section.
- Another more subtle problem results from the fact that even a picture of only moderate complexity — if it is not produced by characters of special fonts (as is the philosophy in LaTeX) — might overflow the main memory of TEX. It is then impossible to compile a page which contains this picture. The only way out of this difficulty is to compile text and picture separately and either to combine the two `dvi` files afterwards or to print text and picture in two runs.

In the first part of this article, we present a compromise solution to both problems which:

- enables the user to produce texts plus included pictures of moderate complexity; and
- needs nothing but LaTeX running on a PC together with a small graphics macro package, a parameter file extraction program and (optionally) another utility program which automates the creation of the picture input.

Our solution relies neither on special output devices or files (e.g. laser printers or PostScript files) nor on drawing programs or special picture formats. Instead, the pictures are drawn within the LaTeX `picture` environment which is enriched by a few graphics macros from the extended `epic` style.

[1] M.J. Wichura, *TUGboat* 9, no. (2), p. 193, 1988

3 Graphics via LaTeX techniques

In the following, we use the rather small Enhanced Picture Environment package,[2] which is given by the epic style format, together with the bezier style format[3] for drawing curves, and finally a few additional macros for producing gray tones within certain parts of the picture. As the bezier style and the epic style are public domain files, we concentrate on the additional macros, which are included in the file neubild.sty.[4]

3.1 The picture input

There are two main macros within this style file, \varbild and \VARBILD. The macro \varbild has 7 parameters, 3 of which are optional; the complete macro goes as follows:

```
\varbild[#1](#2,#3)(#4,#5)#6#7
```

#1 is optional; its possible values are <l>, <r>, <c> or <v>, the default being <v>.
The values of #1 have the following meanings:

<l>: The picture is located at the left margin of the text, and the text flows around the picture on its right side.

<r>: The picture is located at the right margin of the text, and the text flows around the picture on its left side.

<c>: The picture is centered, and there is no text neither at the right nor at the left of the picture.

<v>: <v>=<r> if the pagenumber is odd, else <v>=<l>.

#2 and #3 denote the width and the height of the picture in multiples of \unitlength. The default \unitlength provided by neubild.sty is 1 mm.

#4 and #5 are again optional; the defaults are #4=#5=0. They denote the origin's translation in the coordinate system of the LaTeX picture environment, again as multiples of \unitlength.

#6 denotes the contents of the picture environment (without the control sequences \begin{picture} and \end{picture}).

#7 finally is a correction parameter; it denotes the number of lines by which TeX's calculation for the picture depth should be shortened. #7 can be positive, zero or negative and does not have any meaning with the option #1=<c> (but must be set to zero in this case).

The macro \VARBILD has 8 parameters, 3 of which are optional. Here, instead of including picture commands explicitly as with argument #6 above, we pass filenames so that the picture may be laid out on a separate run of TeX. Two filenames are needed for each picture: one for a LaTeX 'driver' file which places the picture in the proper position on a page, the other for a file containing picture commands.

The complete macro goes as follows:

```
\VARBILD[#1](#2,#3)(#4,#5)#6#7#8
```

The parameters #1 through #5 have the same meaning as in the macro \varbild; again, #1, #4 and #5 are optional.

#6 has the same meaning as the parameter #7 in \varbild.

#7 is a string in DOS-format; it denotes the name (without extension) of a ⟨*LaTeX driver*⟩ file, which becomes the source file of the picture in question on a separate run of TeX (since \VARBILD only provides an empty frame of the correct picture dimensions within the text file).

#8 finally — again a string like #7 — denotes the name of a ⟨*picture source*⟩ file which will contain the picture commands called by the corresponding ⟨*LaTeX driver*⟩.

The macro \varbild is used whenever all the commands of a picture shall be enclosed within the text file, while the macro \VARBILD creates an empty frame of the chosen dimensions within the text file and, in addition, writes to a parameter file the information needed to build pure picture files, each of which will be placed at the correct position on its respective page. If the \VARBILD alternative is chosen, the text file must start with the line

```
\immediate\openout\bilder = ⟨parameter file⟩
```

and its last line must be

```
\closeout\bilder
```

After compilation of the main text file, which will also generate the parameter file, the picture files

[2] Copyright (©) Sunil Podar, Dept. of Computer Science, SUNY at Stony Brook, NY 11794, U.S.A.

[3] Copyright (©) 1985 by Leslie Lamport

[4] As the macro package was developed in the context of a German book project, the names of the macros are given in German, which we hope will not give rise to irritations.

have to be created with the program MAKEPIC via the command line

MAKEPIC ⟨*parameter file*⟩

This program produces for each line of the parameter file (*i.e.* for each occurrence of \VARBILD in the source file) a ⟨*LaTeX driver*⟩ file with the name given by argument #7 of \VARBILD. This file contains nothing but the picture environment at the correct place.[5] It can therefore be compiled as usual and be printed onto the appropriate page separately. If the placement of the picture is not quite accurate, it is possible to move the picture by applying the program MAKEPIC once more to the parameter file, this time using the option of moving the coordinate system in question.

3.2 Picture creation and implementation

While it is rather easy to construct pictures within the picture environment of LaTeX as long as there are only lines, vectors, circles and text involved, it is very cumbersome to introduce picture input which contains Bézier functions or other complex features. Therefore, the MAKEPIC program creates a LaTeX file with only the frame of the picture, while the actual picture input is contained in an input file, the name of which is given by the parameter #8 in the \VARBILD macro. The whole picture input — apart from the \begin{picture} and \end{picture} lines — is thus expected as the contents of a ⟨*picture source*⟩ file. This file may be created manually or via the additional utility program PICTPLUS. This program asks either for a function $x = x(t), y = y(t)$ or for an area to be shadowed, and will in return produce a LaTeX input file, which can be inserted into the ⟨*LaTeX driver*⟩ file in place of the line \input ⟨*picture source*⟩ or can be left within the working directory as the ⟨*picture source*⟩ file itself.

Thus, the whole procedure for including a picture of moderate complexity into a LaTeX text file by help of the \VARBILD goes like this:

1. Insert the line

   \immediate\openout\bilder=⟨*parameter file*⟩

   at the very beginning, and the line

   \closeout\bilder

   as the last line into your LaTeX source file.
2. Introduce the option neubild into your document style.
3. Write the source file and insert a \VARBILD macro at the beginning of each paragraph where a picture should be placed. Be sure that the text that will flow around the picture frame does not contain any \par macro (this does not apply if the option #1=<c> is chosen). It is advisable to write the whole paragraph which surrounds a picture in \sloppy mode, since the \textwidth is reduced within this paragraph whenever you produce a picture which does not cover the whole width of the page.
4. Compile the source file and thereby create the parameter file automatically because of step 1.
5. Run the program MAKEPIC on the parameter file to create the ⟨*LaTeX driver*⟩ files for each occurrence of a \VARBILD.
6. Create the ⟨*picture source*⟩ files for each of the ⟨*LaTeX driver*⟩ files either manually or via the program PICTPLUS. Insert these files into the ⟨*LaTeX driver*⟩ files in exchange for the line \input ⟨*picture source*⟩ or put these \input files into your working directory.
7. Compile the ⟨*LaTeX driver*⟩ files separately.
8. Print the main dvi file first and then the ⟨*LaTeX driver*⟩ dvi files separately onto those printer output pages where the corresponding empty frames of the pictures have been produced. If for a certain picture there are slight deviations from the correct placement, compile the ⟨*LaTeX driver*⟩ file in question once again, this time using the option of moving the coordinate system appropriately, or use the facilities of the MAKEPIC program to shift the picture slightly.

While the whole procedure might look a little bit complex, one should bear in mind that — apart from the two auxiliary programs MAKEPIC and PICTPLUS — there are no requirements necessary in addition to the pure TeX mechanism. In the second part of this paper a more elegant solution to the problem, using PostScript files, will be presented.

4 Implementation of graphics via PostScript

With the advent of software driven PostScript interpreters like FREEDOM OF PRESS it has become reasonable to print TeX files via PostScript on a multitude of cheap printers. Of course, printing time is much longer in PostScript than it is with a TeX driver like PCDOT or PTIJET. It is therefore worthwhile to switch to the PostScript scheme only if the results are inconvenient or insufficient otherwise. This is the case if even only moderately

[5] It is, of course, also possible to write the picture source file by hand, using the information from the parameter file.

complex pictures are to be included into TEX or LATEX text files, because on the one hand there exists only a very limited variety of picture elements within TEX or LATEX, and — which is much worse — on the other hand the compilation of the source file may fail due to memory restrictions of TEX.[6] Therefore, if time is not an important factor it might be convenient to create a `dvi` file from the source text together with a PostScript file for each of the pictures to be included and then to combine these files into one PostScript file, for instance, via the `PTIPS` driver. `PTIPS` enables the user to insert any ordinary PostScript file at an arbitrary position into the main file by help of the TEX `\special` command. In the following, a scheme is developed to use this feature in LATEX files which contain pictures of moderate complexity.

5 Preparation of the main text file

In order to prepare the main text file for the inclusion of pictures, a method similar to the one given in the first part of this paper is used. Again, the option `neubild` has to be added to the documentstyle of the main file, and a parameter file has to be opened by the line

`\immediate\openout\bilder=`⟨*parameter file*⟩

and closed by the line

`\closeout\bilder.`

The file `neubild.sty` contains, in addition to the macros already mentioned, the macro `\varpsbild` which writes an entry into the parameter file and sets an empty frame of the desired size and position within the main file. In addition to this, `\varpsbild` puts a mark equivalent to the line

`\special{ps:bildname.ps}`

at the correct position in the main text `dvi` file after compilation. This mark enables the `PTIPS` driver to insert the PostScript file `bildname.ps` at this position as it creates the PostScript file of the main text.

The macro `\varpsbild` is given as

`\varpsbild[#1](#2,#3)(#4,#5)#6#7`

where the parameters `#1` to `#5` have the same meaning as the corresponding parameters within the macro `\VARBILD`, while the parameters `#6` and `#7` have the following meaning:

`#6` is mandatory; it is the name of the PostScript file which contains the picture elements to be put into the empty frame.

`#7` is mandatory; it has the same meaning as in the macro `\varbild` (it denotes the number of baselines by which TEX must underestimate the surrounding text to place the picture properly).

6 Preparation of pictures to be included

In principle, any PostScript picture file can be created manually by help of any appropriate editor using the PostScript language. Anybody who has a fluent knowledge of the PostScript language will certainly prefer this way to other possibilities. Since, however, many users of TEX are not so experienced in PostScript, we developed a small PASCAL program which can be used interactively to input the most common elements of the LATEX `picture` environment and output a complete PostScript file ready for insertion into the main text PostScript file. This program, TEX2PS, is menu-driven and allows construction of objects ⟨*line*⟩, ⟨*vector*⟩, ⟨*curve*⟩, ⟨*ellipse*⟩ and ⟨*text*⟩ in any desired combination, solid or dashed, with or without shading. Moreover, multiple constructs of those objects can be performed in any order and PostScript program lines can be added at will manually. Thus, the elements of any picture which can be created within a slightly extended picture environment of LATEX (as, e.g., in the `epic` style) can be input to TEX2PS and be transferred to the corresponding PostScript description. By means of the procedure described above it is then possible to produce a PostScript file which combines the main text with any number of pictures of that sort. The combined file can be printed either directly by a PostScript printer or via a software interpreter like `FREEDOM OF PRESS` or similar program on many non-PostScript printers.

A few remarks should be made concerning the program TEX2PS. Though it is meant to be used without referring to the PostScript language, a slight knowledge of the PostScript graphic elements is recommended (of course, you might get the scheme by trial and error after a while). The assignments *'Draw'* (=*'stroke'*), *'Fill'* and *'Eofill'* have their origin in the PostScript language and do exactly what they would do in a PostScript setting[7]; however, they are enclosed between *'gsave'* and *'grestore'* lines. It is therefore possible to shade an area first and then to draw its border lines without repeating the whole pattern. The option *'Link'* is not a PostScript operator; it is used to construct a continuous path of equal or different elements (for instance to be shaded afterwards).

[6] Section 1.

[7] e.g. *The PostScript Language Reference Manual* by Adobe Systems Inc., Addison-Wesley, 1985

Finally, it should once more be emphasized that — if you have a good knowledge of the PostScript language — it is generally much more efficient to create the picture PostScript file by directly editing the picture rather than using the automated but necessarily clumsy version which is provided by the program TEX2PS.

7 The picture implementation

The method described above yields the following steps of procedure:

1. Start your main LaTeX text file with the line
   `\immediate\openout\bilder=`⟨*name of parameter file*⟩
   and close it by the line
   `\closeout\bilder.`
2. Introduce the option `neubild` into your document style.
3. Write your text file and introduce the line
   `\varpsbild...`
   with the appropriate parameters as explained above at any place where you want to insert a picture.
4. Compile the text file to the corresponding `dvi` file. This also produces the parameter file.
5. Create all the pictures in a PostScript setting either manually or by help of the program TEX2PS, taking into account the correct size and name of every picture (the parameter file contains these parameters for each picture involved).
6. Convert the main `dvi` file by help of the `PTIPS` driver, using all of the PostScript picture files, to the final PostScript file.
7. Print the final PostScript file either on a PostScript printer or via a soft interpreter like `FREEDOM OF PRESS`.

8 Conclusion

The two procedures described above are certainly not the most elegant ones for implementing graphics in TeX. As has been shown, the first method, however, has the advantage of not using any graphics input apart from that which is admissible in the LaTeX `picture` environment, and it is completely driver-independent. A somewhat similar approach to this problem is, for instance, given by M. Ballantyne and collaborators[8]; their method, however, is at the moment not applicable within a LaTeX environment and, moreover, does not seem to work very well if the pictures are to be surrounded by text passages. On the other hand, that method can also be used to include complex tables into a TeX file.

We have not discussed the various methods which use graphics input from different drawing programs to be included into TeX source files. These methods depend heavily on the output format of the drawing programs (e.g. whether or not they are pixel oriented) as well as the ability of TeX drivers to implement the different graphic formats (usually by means of `\special` commands).

The second procedure allows the insertion of much more complex pictures into LaTeX text files at the price of using part of the PostScript machinery. We feel that it might be a good compromise if the time factor does not have first priority and the pictures to be inserted into the text are of moderate complexity.

A diskette containing the files used in these approaches can be ordered from the author. Please, enclose an empty diskette and DM 5,- for postage.

◇ Gerhard Berendt
Institut für Mathematik I
Freie Universität Berlin
Arnimallee 2-6
1000 Berlin 33
Germany
`berendt@fubinf.uucp`

[8] M. Ballantyne et al., *TUGboat* 10, no. 2, p. 164, 1989

Including Macintosh Graphics in LaTeX Documents

Len Schwer

Abstract

The basics of including Macintosh graphics in LaTeX documents are discussed for the person who is inexperienced at doing so. Because there is no universal way to incorporate such graphics, other than with scissors and glue, this article tries to be as general as possible, but ultimately references specific software and hardware, *e.g.* ArborText's DVIPS, Trevor Darrell's `psfig` macros, and an Apple LaserWriter+. The reader is assumed to have some knowledge of