Combining TEX and PostScript

Vladimir Batagelj
University of Ljubljana, Department of Mathematics
Jadranska 19, 61 111 Ljubljana
Slovenia

WWW: http://www.uni-lj.si/~vlado/vlado.html

Abstract

PostScript is becoming a de facto standard as a device independent page description language. By embedding PostScript elements in TEX we can extend the use of TEX to new areas of application.

In the first part of the paper we give some general information about PostScript and its features. In the rest of the paper we present some of our own experiences and solutions in combining TEX and PostScript:

- dictionaries, prolog files and how to save a lot of space with PostScript figures produced in *CorelDRAW*, *Mathematica*, ...;
- writing TEX-PostScript macros, case: drawing graphs (combinatorics) in TEX; PostScript error handling mechanism, an application in function graph drawing macro.

Version: March 1995 – updated version of the paper presented at EuroTEX'94, Sobieszewo, Poland, 26-30. Sep 1994.

Key words: PostScript, TEX, inclusion of graphics, dictionaries, macros, error handling.

Math. Subj. Class. (1991): 68 U 15, 68 - 01, 68 N 15

Introduction

Pictures, figures and color are often important elements of a document. They are foreign concepts to TEX which is essentially based on arranging and glueing of boxes.

In his *A Survey of TEX and Graphics* [6, p. 275-276] S. Rahtz discusses six approaches for producing graphics in TEX. The first five are based on the TEX system and therefore preserve device independence, but they are inflexible in those cases where a picture has to be transformed (scaled, rotated).

The sixth approach is based on the use of the TEX command `\special` with which we can include in the DVI file commands for a selected output device driver. By doing this we lose device independence; but, in the case of PostScript, and considering its graphical power and the availability of printers and previewers, this little adultery seems worthwhile. In this paper we shall take a closer look at this approach.

In the first part of the paper we give a short introduction to basic ideas and capabilities of PostScript, thus making the paper self-contained. In the rest of the paper we present some of our own experiences and solutions on PCs in combining TEX and PostScript.

PostScript

What is PostScript? PostScript is a graphics programming language for describing, in a device-independent manner, text and other graphical objects and how they are placed on the page or screen.

It was developed in 1985 by Adobe Systems in a joint project with Apple Computer on the development of the Apple LaserWriter. This version is known as PostScript Level 1.

Although PostScript was initially designed as an interface between picture production and text formatting programs on one side, and printers on the other side, it evolved into a general interface language between (application) programs and display devices. Its main extensions, by different users, were:

- introduction of colors, improvements of pattern filling and halftones;
- support for composite fonts (Japanese and other Eastern alphabets);
- representation and communication of information in some computer systems – Display PostScript (NeXT, Silicon Graphics).

At the first PostScript Conference in 1990, PostScript Level 2 was announced which integrated these features into a new version of the PostScript language.

PostScript programs and their execution. A PostScript program is a text (ASCII) file. Usually it is produced by some other graphics or text formatting program (Word, Word Perfect, *CorelDRAW*, *Mathematica*, ...), but it can be also prepared and maintained by a user and any text editor.

To obtain from a document described in TEX on *file*.`tex` its PostScript description on *file*.`ps`, we first produce, as usual, the corresponding DVI file *file*.`dvi` and translate it using some DVI-to-PS program (DVIPS, DVI2PS, DVITOPS, ...) into PostScript.

The simplest way to display the results of a PostScript program on *file*.`ps` is to send it to a PostScript printer (`copy` *file*.`ps lpt:` or `print` *file*.`ps`).

PostScript programs are either interpreted by an interpreter built into a display device (i.e., laser printer) or by a software interpreter in the user's computer. The most widespread software PostScript interpreter is Ghostscript (Aladdin Enterprises and Free Software Foundation). Ghostscript 3.12 (September 1994) implements PostScript Level 2. Ghostscript enables us to preview PostScript documents on the screen and to print them on several nonPostScript printers.

Basic PostScript programming

Syntax. PostScript program starts with

`%!PS`

followed by the description of page(s). PostScript recognizes, besides a printable subset of the ASCII character set, also characters *space*, *tab* and *newline* (CR or LF or CR LF).

Some PostScript printers use CTRL-D as an indicator of end-of-job. For this reason some application programs insert CTRL-D at the beginning of PostScript files, which is often a source of problems when we are trying to include such files in our documents.

The content of the line from `%` till the end of line is a comment.

PostScript is a stack-based language and uses a postfix (reverse Polish) notation for commands

$p_1\ p_2\ \ldots p_n\ cmd$

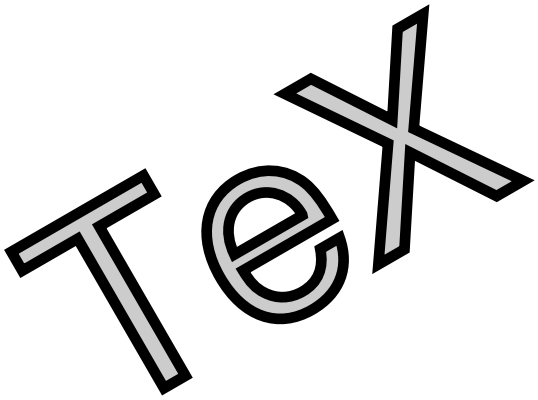
The interpreter puts the arguments p_1, p_2, ..., p_n on the stack and leaves the results of command *cmd* on it.

PostScript [1, 12, 2, 3, 13] is a powerful programming language which besides general programming elements: data types (integer, real, boolean, string, array, dictionary, file), control statements (`if`, `ifelse`, `loop`, `for`, `exit`, `exec`), arithmetic operations and functions (`add`, `sub`, `mul`, `div`, `idiv`, `mod`, `abs`, `neg`, `ceiling`, `floor`, `round`, `truncate`, `sqrt`, `exp`, `ln`, `log`, `sin`, `cos`, `atan`, `rand`, `srand`, `rrand`), operations and functions on other data types, conversion operators, stack commands (`dup`, `exch`, `pop`, `copy`, `roll`), environment commands(`save`, `restore`, `gsave`, `grestore`); contains also many specific graphics commands: coordinate system changing commands (`rotate`, `scale`, `translate`, `transform`), path drawing commands (`moveto`, `rmoveto`, `lineto`, `rlineto`, `curveto`, `arc`, `charpath`, `newpath`, `closepath`), attribute setting commands (`setgray`, `setcmykcolor`, `setrgbcolor`, `setlinewidth`), font commands (`findfont`, `scalefont`, `setfont`), displaying commands (`clip`, `stroke`, `fill`, `show`, `showpage`).

PostScript's coordinate system. PostScript's own coordinate system is based on units called points (72 pt = 1 inch). It has the origin (0,0) in the lower left corner (letter = 8.5 × 11 inch = 612 × 792 pt; A4 = 21 × 29.7 cm = 595 × 842 pt). The content of the page is composed of page elements – parts of pictures or text. Each page element is determined by a set of paths (lines, arcs, curves) and their properties which are realized after the application of some displaying command. Characters are also treated as pictures, but supported by a special set of very efficient commands.

Example: Simple program.

```
%!PS
/Helvetica findfont 100 scalefont setfont
40 0 moveto
30 rotate
(TeX) false charpath
gsave
  0.8 setgray
  fill
grestore
4 setlinewidth
stroke
showpage
```



The first line of the program declares that this is a PostScript program. In the second line we set the Helvetica font at size 100pt as the current font. Then we move to the point (40,0) and rotate the coordinate system through 30 degrees. In the next line we transform the text `TeX` into its outline. The command `gsave` saves the current graphic environment. We fill the interior of the outline with 0.8 gray (1 is white, 0 is black) and restore the graphical environment. Now we set the line width to 4pt and draw the outline. It has to be emphasized that path drawing and attribute setting commands create only descriptions of paths which are not realized on the page until some displaying command is issued. The command `showpage` at the end of the page requires that the interpreter display the page.

Dictionaries. An important concept in PostScript is the notion of a dictionary. It consists of (*key, value*) pairs, which are in some sense the PostScript equivalent of the concept of a variable. The *value* is stored under the name */key* into the current dictionary by the command

/key value `def`

There is a stack of active dictionaries which determine the current context. There are always two permanent dictionaries `systemdict` and `userdict` (and `globaldict`), but the user can introduce his own dictionaries.

A new dictionary of size n (number of entries) is created by the command

n `dict`

and saved in the current dictionary under the name */D* by the command

/D n `dict def`

It is opened for use by the command

D `begin`

and closed by the matching command

`end`

Although dictionaries allow us to use variables in a way similar to normal programming languages, this is not in the 'spirit' of PostScript – try to do the job on the stack.

Besides data, we can store in a dictionary also procedures. Dictionaries are usually used to prepare libraries for special tasks.

User defined commands. User defined commands (procedures) are, in PostScript, a special kind of array enclosed in braces `{ }` – executable arrays. Usually we define a procedure *proc* by storing its body `{` *cmds* `}` into a current dictionary

/proc `{` *cmds* `}` `def`

The following two commands define the usual units

```
/inch { 72 mul } def
/mm { 2.835 mul } def
```

The command `11 mm` puts on the stack values 11 and 2.835, `mul`tiply them and returns their product (11 mm expressed in pts) on the stack.

Example: Drawing graphs. This example demonstrates the use of a dictionary for the simple task of drawing (combinatorial) graphs. The dictionary `Graph` contains two quantities:

`pr` – radius of a point;
`pc` – color of the interior of a point;

and four commands

r `radius` – defines/changes `pr`;
c `pointcolor` – defines/changes `pc`;
x y `p` – draws a point at (x, y);
x_1 y_1 x_2 y_2 `l` – draws a line connecting (x_1, y_1) and (x_2, y_2).

The `p` and `l` commands in the description of the graph were obtained by the *Mathematica* based system Vega [14]. The resulting graph is presented in Figure 1. Note that all lines are drawn before points.

```
%!PS
%%BoundingBox: 30 30 370 370
/Graph 6 dict def
Graph begin
 /radius {/pr exch def} def
 /pointcolor {/pc exch def} def
 /p { pr 0 360 arc
  gsave pc setgray fill grestore
  stroke } def
 /l { moveto lineto stroke } def
end
Graph begin
 0.7 setgray 2 setlinewidth
 249 360  71 101 l  249 360 151  40 l
 249 360 249  40 l  249 360 329 101 l
 249 360 360 200 l
 151 360  71 101 l  151 360 151  40 l
```