Combining TEX and PostScript*

Vladimir Batagelj

University of Ljubljana, Department of Mathematics
Jadranska 19, 61 111 Ljubljana
Slovenia
vladimir.batagelj@uni-lj.si/vlado.html
http://www.uni-lj.si/vlado/vlado.html

March 1995

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.

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

1 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.

2 PostScript

2.1 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.

*Version: March 1995 — updated version of the paper presented at EuroTEX'94, Sobieszewo, Poland, 26-30. Sep 1994.
Math. Subj. Class. (1991): 68 U 15, 68 - 01, 68 N 15

2.2 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.

3 Basic PostScript programming

3.1 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.

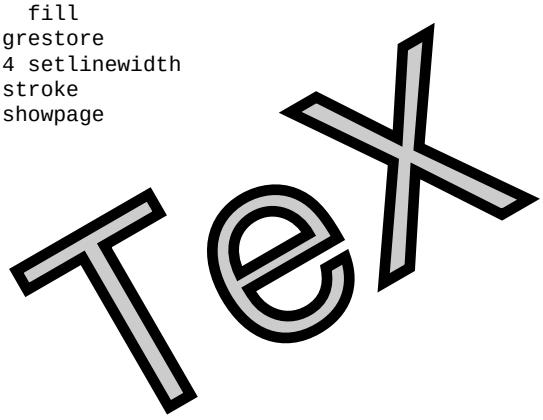
PostScript [1, 12, 2, 3, 13] is a powerful programming language which besides general programming elements: data types (integer, real, boolean, string, array, dictionary, file), control statements (`if`, `ifelse`, `loop`, `for`, `exit`, `exec`), arithmetic operations and functions (`add`, `sub`, `mul`, `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`).

3.2 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 $\times$ 11 inch = 612 $\times$ 792 pt; A4 = 21 $\times$ 29.7 cm = 595 $\times$ 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.

3.3 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.

3.4 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.

3.5 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.

3.6 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.

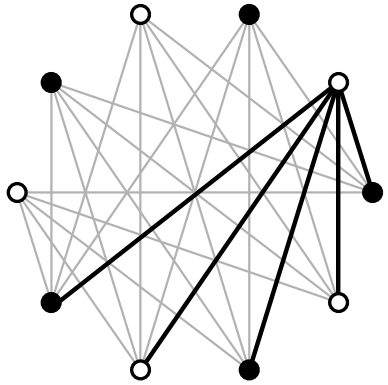


Figure 1: *Graph*

```
%!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
 151 360 249  40 l  151 360 329 101 l
 151 360 360 200 l
  71 299  71 101 l   71 299 151  40 l
  71 299 249  40 l   71 299 329 101 l
  71 299 360 200 l
  40 200  71 101 l   40 200 151  40 l
  40 200 249  40 l   40 200 329 101 l
  40 200 360 200 l
 0 setgray 4 setlinewidth
 329 299  77 101 l  329 299 151  40 l
 329 299 249  40 l  329 299 329 101 l
 329 299 360 200 l
 8 radius 1 pointcolor 3 setlinewidth
  40 200 p  151  40 p  329 101 p
 329 299 p  151 360 p
 0 pointcolor
 360 200 p   71 299 p   71 101 p
 249  40 p  249 360 p
showpage
end
```

3.7 DSC — Document Structuring Conventions

PostScript program lines beginning with `%%` are special comments intended for programs (previewers, utilities) which process PostScript programs. The rules determining the structure and meaning of these comments are known as DSC — Document Structuring Conventions. A typical PostScript program structure is

```
%!PS-Adobe-3.0
%%Pages: 27
```
. . . DSC comments
```
%%EndComments
%%BeginProlog
```
. . . definitions of commands . . .
```
%%EndProlog
%%BeginSetup
...
%%EndSetup
%%Page: 1 1
%%BeginPageSetup
...
%%EndPageSetup
...
%%Trailer
%%EOF
```

The first line `%!PS-Adobe-3.0` tells us that the program conforms to DSC – version 3.0. We can omit unused DSC comments.

An example of a previewer using DSC comments is GsView (by Russell Lang) [10] which by using `%%Page:` comments allows us to see or print the selected page(s).

3.8 EPS – Encapsulated PostScript

Encapsulated PostScript format is intended to allow one to import already prepared parts of a picture into a document. The EPS file should contain only one page and shouldn't use operators that would perturb the graphics environment of the surrounding PostScript. It usually starts with a line

`%!PS-Adobe-3.0 EPSF- 3.0`

and one of the following lines should be

`%%BoundingBox:` $ll_x\ ll_y\ ur_x\ ur_y$

which defines the bounding box of the picture in the file.

4 PostScript and TEX

4.1 DVIPS and EPSF

The most popular DVI-to-PS program on PCs is DVIPS [15]. It was written by Tomas Rokicki of Radical Eye Software. DVIPS comes with two files `epsf.tex` and `epsf.sty` which contain TEX macros to include an Encapsulated PostScript graphic. It works by finding the bounding box comment, calculating the correct scale values, and inserting a `vbox` of the appropriate size at the current position in the TEX document.

Program DVIPS recognizes several forms of `\special` command. For example:

`\special{"` $cmds$`}` — includes PostScript commands in place. The user is responsible for providing space for such literal graphics. The $cmds$ are enclosed in a PostScript `save`/`restore` pair.
`\special{ps:` $cmds$`}` — inline PostScript commands — not enclosed in a `save`/`restore` pair.
`\special{header=`$file$`.pro}` — includes the contents of a prolog file $file$`.pro` in `userdict`.

4.2 PostScript-TEX packages

By the end of eighties the first attempts to combine TEX and PostScript were being made (PSLATEX(L.A. Carr), colors (F. King), pspic (K.K. Thorup), psfig (T. Darell [5]), . . .) and recently some excellent packages have been produced (changebar (J. Braams), rotating (S. Rahtz, L. Barroca), psboxit (J. Maillot, T. Sheffler), pspicture (D. Carlisle), epsfig (S. Rahtz), geom (S. Levy [11]), foiltex (J.L. Hafner [8]), seminar, PsTricks (T. Van Zandt [18, 16, 17]), PSNFSS (S. Rahtz)). In the new LATEX there are some PostScript based packages: pict2e, graphics and color [9].

4.3 Epsfig

The most popular package for inclusion of EPS figures in TEX is epsfig (S. Rahtz, based on epsf and on T. Darell's psfig). It extracts the bounding box information from the file and positions the figure according to the user's wishes on the page. Its main macro has the following parameters:

`\epsfig{file=`*file*`,height=`h`,width=`w`,%`
`  clip=,rotate=`a`,silent=,%`
`  bbllx=`lx`,bblly=`ly`,bburx=`ux`,bbury=`uy`}`

5 PostScript, TEX and other programs

5.1 CorelDraw 3

After we enter *CorelDRAW* we can first determine the dimensions of the picture by selecting in menu `File/Page Setup...` the option `Custom`. This enables fields for user's settings. First, if necessary, we change the units of measurement — for example to `millimeters`. Afterwards we enter the selected width and height of the picture and confirm our decision with a click on the `OK` button. Then we draw a picture. For possible future changes we save it in *CorelDRAW* format (options `File/Save As...` and `File/Save`) on *file*`.cdr`.

To get the picture in EPS format we enter the menu `File/Export`. In the pop-up submenu `List Files of Type:` we select the option `Encapsulated PostScript, *.EPS`. Then we move in the directory submenu to the directory where we keep the pictures, enter the name of the file, and confirm our selections with `OK`. A new dialog box `Export EPS` appears in which we select the option `Image Header/None` and confirm it with `OK`. The picture is saved in EPS format in the file *file*`.eps`. The bounding box of the picture is determined by *CorelDRAW* from the picture. Such pictures can be easily included into our TEX document:

```
\documentstyle[11pt,epsfig]{article}
\begin{document}
...
\begin{figure} \begin{center}
\centerline{\epsfig{figure=encormix.eps,width=70mm}}}
 \caption{Clip-art from \CD \label{CD}}
\end{center} \end{figure}
...
\end{document}
```

or in LATEX2e [9]

```
\documentclass[11pt]{article}
\usepackage[dvips]{graphics}
\begin{document}

...

\begin{figure} \begin{center}
 \resizebox{70mm}{!}{\includegraphics{graph.eps}}
 \caption{Clip-art from \CD \label{CD}}
\end{center} \end{figure}

...

\end{document}
```

In Figure 2 a clip-art composition, made in *CorelDRAW* in few minutes, is presented.

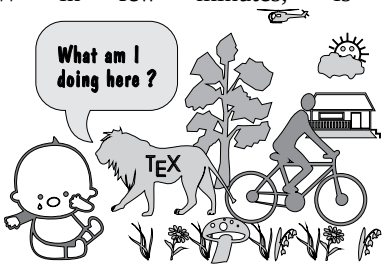


Figure 2: *Clip-art from CorelDRAW*

When we have several *CorelDRAW* figures to be included in our text we can achieve substantial savings both in disk space (15K per file) and processing time if we notice that all *CorelDRAW* EPS files have large parts that are identical. The structure of an EPS file produced by *CorelDRAW* 3 is as follows

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 7 6 191 139
%%Creator: CorelDRAW!
%%Title: TESTA.EPS
%%CreationDate: Sat May 14 21:20:40 1994
%%DocumentFonts: AvantGarde-Book
%%DocumentProcessColors: Cyan Magenta Yel...
%%EndComments
%%BeginProlog
/AutoFlatness false def
% -------------- POSTSCRIPT PROLOG FOR CO...
% Copyright 1992 Corel Corporation.  All ...
/wCorelDict 300 dict def wCorelDict begin...

...

%%EndProlog
%%BeginSetup

...

%%Trailer
 end
```

The contents of the file after the `%%BeginProlog` until the `%%EndProlog` is the same for all *CorelDRAW* EPS files. It starts with a dictionary

```
/wCorelDict 300 dict def wCorelDict begin
```

which is ended by the `end` in the trailer at the end of the file.

We can save the constant contents between the `%%BeginProlog` and the `%%EndProlog` as a header file `corel3.pro` (15K), which is read only once by a style file `corel3.sty`

```
\long\def\ifundefined#1#2#3{\expandafter
 \ifx\csname#1\endcsname\relax#2\else#3\fi}
\ifundefined{Corel3DrawSTY}
 {\def\Corel3DrawSTY{}}{\endinput}
\immediate\write16{Document Style Option %
 'CorelDraw 3' ver 1.0 / 14-May-94 / VB}
\special{header=corel3.pro}
```

We have to insert in `corel3.pro` an `end` a line before the `%%EndProlog`.

Then we can in each *CorelDRAW* EPS file delete the contents between the `%%BeginProlog` and the `%%EndProlog` and replace it with a command `wCorelDict begin` thus obtaining a shortened file *file*`.cps`. This can be done manually using an editor or by a simple program.

We can still view the shortened files by Ghostscript requesting

`gs corel3.pro` *file*`.cps showpage.ps`

On CTANs we can get also a PostScript version of standard TEX fonts. If we register (some of) them with ATM (Adobe Type Manager) they become available to *CorelDRAW* and we can use them for labels in our pictures (see Figure 3).

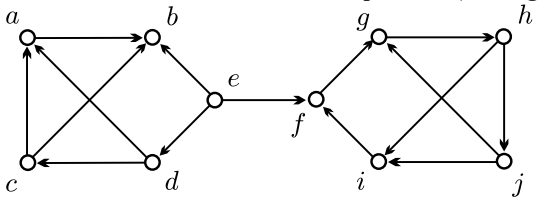


Figure 3: *Picture from CorelDRAW with* `cmmi10` *labels*

5.2 CorelDraw 5

Essentially the same applies as for *CorelDRAW* 3. There are some differences in the preparation of CPS files. We extract (once) from a *CorelDRAW* 5 EPS file the segment from `%%BeginProlog` till `%%EndProlog` (including) to the file `corel5.pro` (21.4K). In this file we insert after `wCorel5dict begin` the command

```
/AutoFlatness true def
```

We produce a CPS file by deleting from the corresponding EPS file the segment from the first `%%BeginSetup` till `%%EndProlog` (including).

For unknown reasons in *CorelDRAW* 5 the origin of coordinate system is placed at position around $(2200, 2200)$. To preview the *CorelDRAW* 5 CPS files with Ghostscript we write on the file `cd5.ps` the lines

```
%!PS
-2200 -2200 translate
```

Now we can require

`gs cd5.ps corel5.pro` *file*`.cps showpage.ps`

5.3 Mathematica

Using *Mathematica* we draw a picture and save it with command `Display["`*file*`",`*picture*`]` to the file *file*. For example:

```
d:\>math
In[1] := <<Graphics‘Polyhedra‘
In[2] := Show[Graphics3D[Dodecahedron[],
 ColorOutput->GrayLevel,Boxed->False]]
In[3] := Display["dodec.ps",%]
In[4] := Exit
```

The saved *file* contains a PostScript description of a picture. After we exit *Mathematica*, we have still to transform the saved picture into EPS format. This can be done with the DOS command `eps` *file EPSfile*. In our example

```
d:\>eps dodec.ps dodec
```

The command `eps` is determined by a batch file `eps.bat` which contains a call to *Mathematica* program `rasterps`

```
rasterps -format eps -file %2.eps %1 %3 %4 %5
```

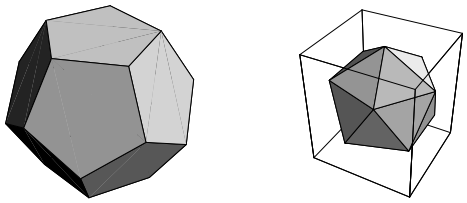


Figure 4: *Pictures from Mathematica*

This program inserts at the beginning of the *file* some DSC comments and the *Mathematica* prolog.

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 138 138
%%Creator: Mathematica
%%CreationDate: Thu Aug  4 03:01:20 1994
%%EndComments
%%BeginPreview: 200 200 1 200

...

} bind def
%!
%%Creator: Mathematica
%%AspectRatio: 1.00154
MathPictureStart
%% Graphics3D
/Courier findfont 10  scalefont  setfont
```

...picture

```
% End of Graphics
MathPictureEnd
end
showpage
```

thus producing an EPS file.

Again we can input *Mathematica* prolog (24K) only once and insert at the begining of the *file* a short header

```
%!PS-Adobe-2.0 EPSF-2.0
%%BoundingBox: 0 0 276 276
%%Creator: Mathematica
% *** Partial EPS form produced by PS4TeX
%%CreationDate: ...
%%EndComments
Mathdict begin
```

and add as a last line `end showpage`. For unknown reasons the bounding box provided by `rasterps` is only one half of the correct one.

For preparing `cps` and `mps` files, a short program `PS4TEX` was written. Its latest version for the PC is available in self-extracting format by anonymous FTP from

```
uek.uni-lj.si:pub/vlado/tex/ps4tex*.exe
```

6 TEX-PostScript macros

As already said, in principle PostScript programs are generated by other programs which are also responsible for their correctness. Different approaches to this problem were used in the existing TEX-PostScript packages (see, for example, Van Zandt's PSTricks and Carlisle's pspicture).

Although PostScript is a complete programming language, the generating program should perform all computations and other processing that it can. Nevertheless, since TEX lacks (trigonometric and other) functions, we sometimes leave PostScript to do the job.

6.1 Error mechanism in PostScript

The dictionary `systemdict` contains as its entries two dictionaries related to error handling. The `errordict` contains built-in procedures for all possible error types and a procedure `handleerror`. On an error the PostScript interpreter executes the corresponding error procedure which saves the information about the error in the second dictionary `$error` and executes the command `stop`.

We can catch a `stop` inside the commands context $cmds$ by the command

{ $cmds$ } `stopped`

which, when an error occurs, pops the corresponding part of execution stack and returns a boolean value `true`.

The command `stop` transfers control to the innermost `stopped` context. The main interpreter loop is always such a context and invokes the standard `handleerror` procedure as a part of error recovery. The user can replace this procedure by his or her own procedure.

6.2 Example: Functions

The following simple PostScript program draws (see Figure 5) a function $f(x) = 200 \sin 2x$.

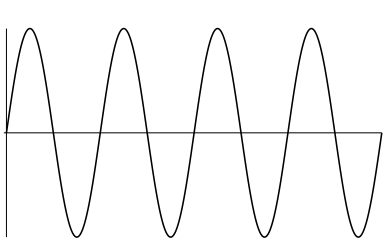


Figure 5: *Function*

```
%!PS
%%BoundingBox: -5 90 720 510
0 300 moveto
0 1 720 { %     from step to
  dup     %     x x
  2 mul   %     2*x
  sin     %     sin(2x)
  200 mul %     200 sin(2x)
  300 add %     300 +
  lineto  %
} for
3 setlinewidth stroke
-5 300 moveto 720 300 lineto
0 100 moveto 0 500 lineto
1 setlinewidth stroke
showpage
```

But, if we try to draw, by inserting a `sqrt` after the `sin`, a function $f(x) = 200\sqrt{\sin 2x}$ the error `rangecheck` occurs. We can overcome the problem (see Figure 6) by intercepting the error by the command `stopped` as shown in the improved version of the program

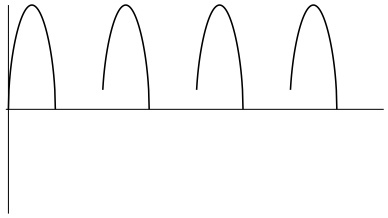


Figure 6: *Function*

```
%!PS
%%BoundingBox: -5 90 720 510
/up true def
0 300 moveto
0 1 720 {
  { dup 2 mul sin sqrt 200 mul
    300 add } stopped
  { pop /up true def }
  { up
    { moveto /up false def }
    { lineto }
    ifelse }
  ifelse }
for
3 setlinewidth stroke
-5 300 moveto 720 300 lineto
0 100 moveto 0 500 lineto
1 setlinewidth stroke
showpage
```

There is still a lot of work to convert this solution to a macro for drawing functions, but the details go beyond the scope of this paper.

7 Conclusion

Combining TEX and PostScript returns TEX into the competition with other typesetting systems. It will be a big challenge for the TEX community in the following years to provide the support for this approach.

References

[1] Adobe Systems Inc.: PostScript Language, Reference Manual. Second Edition. Addison-Wesley, Reading, MA, 1990.

[2] Adobe Systems Inc.: PostScript Language, Tutorial and Cookbook. Addison-Wesley, Reading, MA, 1985.

[3] Adobe Systems Inc., Reid G.C.: PostScript Language, Program Design. Addison-Wesley, Reading, MA, 1988.

[4] Aladdin Enterprises: Ghostscript, `use.doc`. 30 Sep 1994. `ftp.cs.wisc.edu:pub/ghost`, 1994.

[5] Darrell T.: Psfig/TEX 1.10 User's Guide. `whitechapel.mit.media.edu:pub`, 1994.

[6] Goossens M., Mittelbach F., Samarin A.: The LATEX Companion. Addison-Wesley, Reading, MA, 1994.

[7] Goossens M., Van Herwijnen E.: Scientific Text Processing. International Journal of Modern Physics C **3**(1992)3, 479-546.

[8] Hafner J.L.: Making Foils Using FoilTEX. 21 aug 1992. IBM Almaden Research Center, San Jose, CA.

[9] Lamport L.: A Document Preparation System LATEX. User's Guide and Reference Manual. Second Edition. Addison-Wesley, Reading, MA, 1994.

[10] Lang R.: Gsview, `readme.doc`. 9 Dec 1994. `ftp.cs.wisc.edu:pub/ghost/rjl`, 1994.

[11] Levy S.: The `geom` style for LATEX. Geometry center, University of Minnesota, `geom.umn.edu`, July 1992.

[12] McGilton H., Campione M.: PostScript by Example. Addison-Wesley, Reading, MA, 1992.

[13] Monsarrat J.: — PostScript — Answers to Questions. The `comp.lang.postscript` FAQ v2.2 (12-26-1993). `wilma.cs.brown.edu:pub`, 1993.

[14] Pisanski T.: Vega 0.3 alpha release, 1994. `http://vegaj.mat.uni-lj.si/vega03/`.

[15] Rokicki T.: DVIPS: A TEX Driver. Manual, v5.521. `labrea.stanford.edu:pub`, 1994.

[16] Van Zandt T.: PSTricks, PostScript macros for Generic TEX. User's Guide. Version 0.93a, `princeton.edu:pub/tvz`, 12 Mar 1993.

[17] PSTricks et Seminar (D. Girou, ed.). Cahiers GUTenberg 16, 1994. (in French)

[18] Van Zandt T.: seminar.sty, A LATEX style for slides and notes. User's Guide. Version 0.93, `princeton.edu:pub/tvz`, 16 Feb 1993.

[19] Walsh N.: Making TEX Work. O'Reilly, Sebastopol, CA, 1994.

See also:

```
http://www.cs.wisc.edu/~ghost/index.html
http://www.cs.indiana.edu/docproject/
     programming/postscript/postscript.html
http://www.adobe.com/
http://www.ucc.ie/info/TeX/tug/
http://www.tex.ac.uk/UKTUG/home.html
http://www.stat.wisc.edu/latex.html
http://info.desy.de/UCO/latex2e.html
```

8 Acknowledgements

Supported in part by the Ministry of Science and Technology of Slovenia.