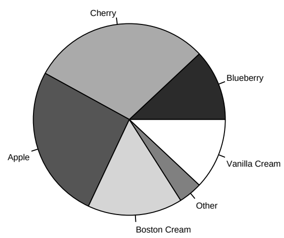
# Psfig/TEX 1.2 Users Guide

Trevor Darrell
trevor@grasp.cis.upenn.edu
Computer and Information Science
University of Pennsylvania

## 1 Introduction

The TEX typesetting system is a powerful tool in the preparation of the written word, yet when the time came to add figures or pictures to a document, one traditionally had to resort to tedious manual paste-up. With the advent of the PostScript page description language, which allows the 'nesting' of environments and is rapidly becoming a *de facto* standard, it is possible to merge graphics directly into a document. Psfig/TEX is a macro package for TEX that facilitates the inclusion of arbitrary PostScript figures into TEX documents. Figures are automatically scaled and positioned on the page, and the proper amount of space is reserved. Some possible figures include:



Custom characters such as '∞' and '◕' may be created and used freely throughout a document.

## 2 Simple figures

To include a PostScript figure with psfig, first load the psfig macros at the beginning of your document with

```
\input{psfig}
```

then, when you wish to include a figure, invoke the macro

```
\psfig{figure=input}
```

where *input* is the name of a PostScript file. Psfig will automatically position the figure at the current point on the page, and reserve the proper amount of space in TEX so that it doesn't conflict with any other objects on the page.

For example, if we have a file called 'piechart.ps' which contains the PostScript code to draw the chart in the introduction, we could use the commands

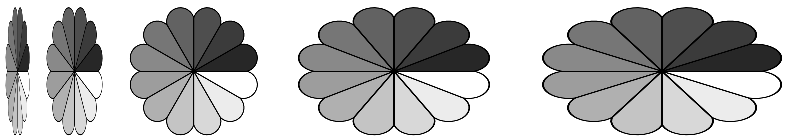
```
\par
\centerline{\psfig{figure=piechart.ps}}
\par
```

to include it as a centered paragraph. Since no mention of size was made in the above example, psfig draws the figure at its natural size (as if it was printed directly on a PostScript printer.) The pie's natural size is several inches across, which is a little large; the pie in the introduction was produced with:

```
\centerline{\psfig{figure=piechart.ps,height=1.5in}}
```

The `height` option specifies how tall the figure should be on the page. Since no `width` was specified, the figure was scaled equally in both dimensions. By listing both a `height` and a `width`, figures can be scaled disproportionately, with interesting results.

For example:



was produced with:

```
\centerline{\hbox{
\psfig{figure=rosette.ps,height=.8in,width=.15in}
\psfig{figure=rosette.ps,height=.8in,width=.35in}
\psfig{figure=rosette.ps,height=.8in}
\psfig{figure=rosette.ps,height=.8in,width=1.2in}
\psfig{figure=rosette.ps,height=.8in,width=1.5in}
}}
```

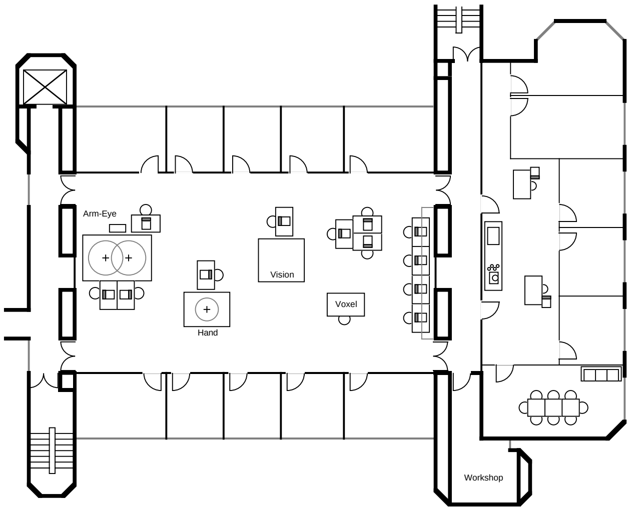



Figure 1: A figure produced by MacDraw

# 3 Macintosh files

Since the Macintosh drawing applications produce PostScript, they can be used to create figures. MacDraw is a recommended program for creating figures, since it produces code that is independent of scaling (as opposed to MacPaint, which produces a bitmap of the figure.) Many other drawing applications exist; hopefully they will also behave in the manner described below.

Before including a Macintosh figure in your TEX document, it must be created on the Macintosh and moved to the host on which TEX is running (assuming the two are not the same nor share filesystems.)

As of this writing, it is possible to convince MacDraw to place a PostScript description of a figure in a file rather than sending it directly to a LaserWriter by hitting clover-F *immediately* after clicking 'OK' on the print dialog box. The file is always called 'PostScript' and is left in the folder MacDraw was started in. Use a communications program to move the file over to the host on which TEX is running.

MacDraw creates a output file in the form of 'QuickDraw' calls, which are interpreted as a set of PostScript procedures. These procedures are defined in what we call the 'Macintosh prolog', which must be prepended to any Macintosh file before being sent to the printer. By using clover-K rather than clover-F in the key sequence above both the prolog and figure will be captured in the PostScript file. The prolog can be extracted by hand-editing out the portion of the clover-K file also present in the clover-F file.

Some versions of MacDraw and the Macintosh LaserWriter driver do not include a bounding box comment in the PostScript files they produce; figures created by these systems will have to have their bounding box added manually (see section 6.)

There is a `prolog` option in the psfig macro to specify a file that should be prepended to the figure. The name of the prolog is, of course, site dependent; we have used '/usr/lib/ps/mac.pro'. For example, if you had a file 'frog.mac' that contained the macintosh code to draw a floor-plan (figure 1).

```
\psfig{figure=lab.ps,prolog=mac.pro}}
```

If there are many such figures, it is probable that the repeated inclusion of the mac.pro file will cause a significant increase in transmission and printing time. A better method is to load the mac.pro file once, concatenating it before the entire document (using `psprint -m`, on our systems,) and ommiting the prolog clause from each psfig command.

# 4 Other sources of figures

Any program that produces PostScript as output can be used for psfig figures as long as it adheres to the bounding box comment convention (see below). For instance, the *ph* program was used to convert a bitmap image of the author into PostScript form and included as a figure at left.

# 5 Draft figures and Silent mode

Normally, psfig will print advisory messages to remind you that it is including figures as TeX processes a document. This behavior can be disabled with `\pssilent`, and re-enabled with `\psnoisy`.

Some PostScript figures can take quite a long time to transmit and print; for these figures a draft mode is available to speed printing intermediate versions of the document. A figure printed in draft mode will occupy the proper space as far as TeX is concerned but no PostScript code will be executed to print it. If psfig is not in silent mode, it will also print the name of the PostScript file in the space reserved for the figure. No `\special`'s are used in draft mode; thus any dvi processor can be used.

The macro `\psdraft` will switch into draft mode, and all subsequent psfig macros will produce draft figures until reaching the macro `\psfull`, which switches out of draft mode.

## 6 Bounding boxes

To properly translate and scale a figure psfig must know its 'natural' position on the page; this information is present in what is called the *bounding box* of a PostScript program. The bounding box is a outer limit to the marks created by a program, and is specified as four coordinates of a rectangle: the lower-left $x$ coordinate (bbllx), the lower-left $y$ coordinate (bblly), the upper-right $x$ coordinate (bburx), and the upper-right $y$ coordinate (bbury). Adobe has defined a convention whereby the bounding box of a program is contained in a 'bounding box comment'. [1] This comment, which must be present in any file to be used as a psfig figure, is a line of the form

`%%BoundingBox:` *bbllx bblly bburx bbury*

All values are in PostScript points, relative to the *default* transformation matrix. The only mandatory PostScript convention is that the first line of the file should begin with the characters '%!' (a '%' begins a comment in PostScript.) A good place for the bounding box comment is as the second line of the file.

The bounding box values may be specified directly on the psfig macro, using clauses `bbllx=`*bbllx*,`bblly=`*bblly*,..., in which case the file is not searched for the bounding box.

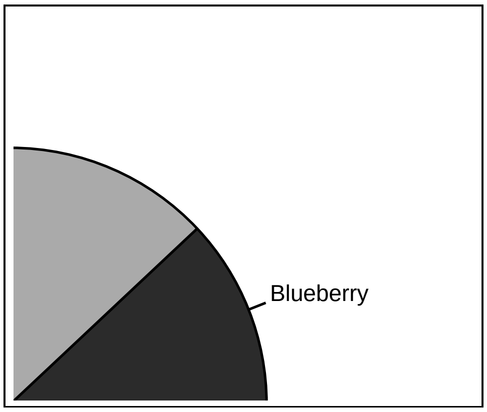
## 7 Reserved size

There are two sizes associated with each psfig figure: the size at which it is to be printed on the page and the size it reserves in TEX. This latter size is appropriately termed the *reserved size*, and is expressed as clauses of the form "`rheight=`*dimen*" and "`rwidth=`*dimen*". If ommited, the reserved size defaults to the real size. Some special effects need to be transparent to TEX and thus have a zero reserved size, such as the grey box enclosing this paragraph.

## 8 Clipping

Normally a PostScript program can be expected to not mark the page outside its bounding box. If this is not the case, or if you want to specify a bounding box so as to isolate part of a larger figure, there is an option that sets the PostScript clip path so that no marks will show up outside the declared bounding box. Currently this is invoked by adding a clause of the form "`clip=`". Here a slice has been taken out of the pie chart in the example by specifying a smaller bounding box and adding the clip option.

[1] See 'Appendix J: PostScript File Structuring Conventions' in *The PostScript Language Reference Manual*



A piece of the pie.

Some PostScript programs use the clipping path to position their output on the page; if a figure is being drawn at its natural size and position despite psfig commands to the contrary, it may need the clip option.

# 9 PostScript environment

The PostScript environment within psfig is fairly robust. All of the usual PostScript operators will operate as desired; in particular the operators 'showpage', 'initgraphics', and 'defaultmatrix' will all behave consistently inside a figure, except that 'showpage' will only do an 'initgraphics' and will *not* print or erase the current page. [2]

It is very possible, however, for a PostScript program to bypass the psfig environment and disrupt a document.

# 10 Caveats

Some versions of LaTeX will fail to properly center a lone figure in a centering environment; a good work-around is to precede the figure with a hard space. On documents with complicated figures, the printer memory allocated to dvips may have to be limited to avoid vmerrors. PostScript files produced from Adobe's ps4014 need to have the clipping boundary set. Finally, the `\psfig` macro will be confused by white space or newlines in its argument.

[2] For more detailed information about PostScript issues in the internals of psfig, see *Psfig – A Ditroff Preprocessor for PostScript Figures* in the USENIX 87 proceedings, or *Bringing troff up to speed* in the July 1987 issue of Unix Review.

## 11 Acknowledgements

Ned Batchelder co-developed the original *troff* version of this program with the author, and was responsible for much of the overall design. Greg Hager provided an initial TEX implementation. The three broken out figures in the introduction were taken from examples in *The PostScript Language Tutorial and Cookbook.* Thanks to Ira Winston and the CIS department for supporting this work.

# A Installation, and Availability

Version 1.2 of psfig/tex is supported for the *dvips* and *dvi2ps* [3] postprocessors. It is available via anonymous ftp from linc.cis.upenn.edu, in the directory `ftp/dist/psfig` as the file `tex.tar.Z`.

The first step to install psfig is to print out a draft copy of this guide simply by running `latex ug.tex`, which presumably you have already done. All *psfig* files mentioned are relative to then `psfig/tex` directory on the distribution.

## A.1 Install psfig.pro

For *dvips*, append the file `dvips/psfig.pro` to the end of the `dvips.pro` file that *dvips* includes each time it processes a document.

For *dvi2ps*, copy `dvi2ps/psfig.pro` into the *dvi2ps* PostScript driver directory (determined by the `PS_DRIVER_DIR` variable in the `local-defs` file of the *dvi2ps* distribution).

## A.2 Install psfig.tex

Copy `dvips/psfig.tex` (for *dvips*), or `dvi2ps/psfig.tex` (for *dvi2ps*), into a directory TeX will search when resolving `\input` commands.

## A.3 Print a full copy of this document

Comment out the first line that contains `\psdraft` from the file `ug.tex`, then `make`. Print `ug.ps`.

[3] Version 1.0BETA by Stephan Bechtolsheim (i5f@l.cc.purdue.edu)