From PostScript to PDF with epstopdf, pdftricks, pst-pdf, auto-pst-pdf, pst2pdf, and more

Herbert Voß

1 Introduction

For a few years now the pdfTEX engine has been the default engine for LATEX. It can create PDF output directly, which implies that images must be in one of these formats: PDF, JPEG, or PNG. The EPS or PostScript formats cannot be used for images with PDF output, whereas they worked with no problem in "traditional" TEX usage (Figure 1):

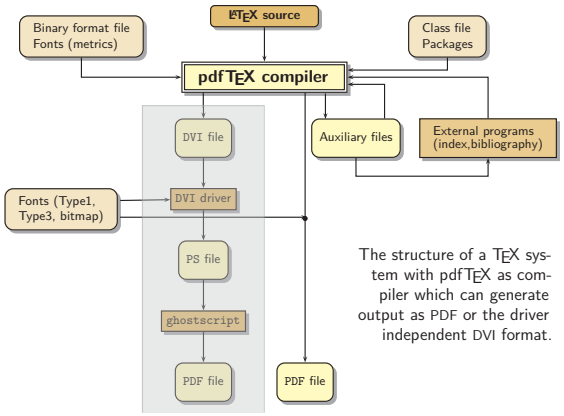


Figure 1: The traditional way of creating PDF output, via DVI.

There are still several reasons to use this traditional way of creating PDF output, namely the sequence `latex` → `dvips` → `ps2pdf`. Using pdfLATEX is only possible when the PostScript related code is handled before the pdfLATEX run. Thus, several packages and/or scripts have been developed to support EPS images, or general PostScript-related code, in a document which is compiled at least one time with pdfLATEX: `pst-pdf`, `auto-pst-pdf`, `pstool`, `epstopdf`, `pst2pdf`, `pdftricks`, ... All have the same general goal, but each works in a different way.

We also show some examples of the reverse: using non-PostScript image files with `latex`.

2 Using EPS images with pdflatex

For `eps`, perhaps the simplest approach is to use the `epstopdf` package, as follows:

```
\documentclass{article}
  \usepackage{graphicx}
  \usepackage{epstopdf}
\begin{document}
  \includegraphics{demo.eps} \end{document}
```

It is necessary to enable execution of external commands in the `pdflatex` run, as shown next.

```
pdflatex -shell-escape test0.tex
```

2.1 Using zipped EPS images with latex

`latex` cannot read compressed files directly, in addition to the image itself, say `demo.eps.gz`, we need a second file for the bounding box: `demo.eps.bb`. The `.bb` file should contain a single line, the EPS bounding box comment:

```
%%BoundingBox: 102 93 203 122
```

Then we can run include `demo.eps.gz` as usual:

```
\documentclass{article}
  \usepackage{graphicx}
\begin{document}
  \includegraphics{demo}
\end{document}
```

```
latex test0.tex
```

3 Using other image types with pdflatex

3.1 Using GIF images with pdflatex

Since pdfTEX doesn't read GIF files directly, this example uses the `convert` utility from ImageMagick to convert GIF to PNG. Any conversion program could be used. (The `ifplatform` package can be used to run different programs on different platforms.) We load the `epstopdf` package to enable the conversions, even though we aren't using EPS images.

```
\documentclass{article}
  \usepackage{graphicx}
  \usepackage{epstopdf}
  \epstopdfDeclareGraphicsRule
    {.gif}{png}{.png}
    {convert gif:#1 png:\OutputFile}
  \AppendGraphicsExtensions{.gif}
\begin{document}
  \includegraphics{knuth-tex.gif}
\end{document}
```

As before, we must enable shell escapes:

```
pdflatex -shell-escape test0.tex
```

3.2 Using PNG and GIF images with latex

Analogously, we can use PNG and GIF images with `latex`, via conversion to EPS. This shows conversion without involving `epstopdf`.

```
\documentclass{article}
  \usepackage{graphicx}
```

```
  \DeclareGraphicsRule
    {.png}{eps}{.bb}
    {‘convert #1 eps:-}
\makeatletter % more complex method for
              % programs other than convert:
  \let\Saved@Gin@base\Gin@base
  \let\Gin@base\relax
  \DeclareGraphicsRule{.gif}{eps}{.bb}
    {‘convert #1 \Gin@base.eps &&
     cat \Gin@base.eps}
  \let\Gin@base\Saved@Gin@base
\makeatother
  \usepackage{grfext}
  \AppendGraphicsExtensions*{.png,.gif}
\begin{document}
  \includegraphics{lion}\qquad
  \includegraphics{knuth-tex}
\end{document}
```

Again, we need `.bb` files for the images:

```
knuth-tex.bb  knuth-tex.gif
lion.bb       lion.png
```

And shell escapes enabled:

```
latex -shell-escape testimgfmts.tex
```

4 Using PSTricks with pdflatex

Let's turn our attention now to some of the methods for using PSTricks packages specifically with `pdflatex`.

4.1 pdftricks

First, the `pdftricks` package. With this, you demarcate the preamble that should be used for the intermediate run with the `psinputs` environment:

```
\documentclass{article}
\usepackage{pdftricks}
\begin{psinputs}% preamble for latex runs!
  \usepackage{pst-node}
  \usepackage{graphicx}
\end{psinputs}
```

And then the usual:

```
pdflatex -shell-escape testpdftricks
```

We can thus use PSTricks packages together with EPS images, and as usual for `pdflatex` also JPEG, PNG, and PDF images.

4.2 pst-pdf

With the `pst-pdf` package, load PSTricks packages only when not producing PDF:

```
\documentclass{article}
\usepackage{pst-pdf,ifpdf}
\ifpdf\else
  \usepackage{pst-node}
\fi
```

Then there are several steps to the processing:

1. Run `latex` to create a `dvi` file with only the extracted `pspicture` and `postscript` environments. or `\includegraphics` for `eps` images.
2. The `dvi` output then is converted to a PostScript file which itself is of a special format and can only be used for the next step.
3. The `ps` output is converted to a `pdf` file which has one page per extracted image.
4. If needed, run `pdfcrop` to tightly crop.
5. The last `pdflatex` run replaces the `pspicture` and `postscript` environments and `eps` images with created `pdf` images.

For example:

```
latex ptest
dvips -o ptest-pics.ps ptest.dvi
ps2pdf ptest-pics.ps ptest-pics.pdf
#pdfcrop ptest-pics.pdf
#mv ptest-pics-crop.pdf ptest-pics.pdf
pdflatex ptest
```

4.3 auto-pst-pdf

The `auto-pst-pdf` package automates the above process.

```
\documentclass{article}
\usepackage{auto-pst-pdf,ifpdf}
\ifpdf\else
  \usepackage{pst-node}
\fi
```

We need only *one* `pdflatex` run, everything is done inside of the `auto-pst-pdf` package.

```
pdflatex -shell-escape ptest
```

4.4 Option pdf for PSTricks

The `[pdf]` option to `pstricks` works only for `latex`! It loads the package `auto-pst-pdf`.

```
\documentclass{article}
\usepackage[pdf]{pstricks}
\ifpdf\else
  \usepackage{pst-node}
\fi
...
```

As above, we need only one `pdflatex` run:

```
pdflatex -shell-escape ptest
```

5 pst2pdf

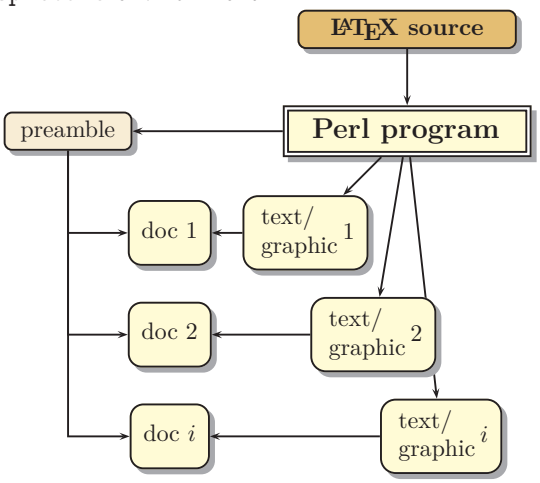
pst2pdf is a Perl script which extracts postscript and pspicture environments from the main document and creates sub-documents with the same preamble as the main document. Thus, any definition will also be valid in the sub-documents.

Both pspicture and postscript environments, can contain arbitrary code *except* that neither can contain a postscript environment. (Thus, only pspicture environments can be nested.) In both cases, only top-level environments will be extracted.

Next, we describe each of pst2pdf's steps in more detail.

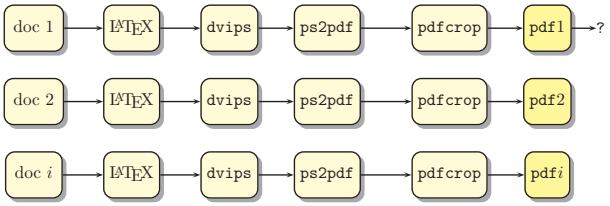
5.1 pst2pdf step 1: create sub-documents

In the first step, the preamble of the main document is saved and then used together with the extracted code snippets to create new sub-documents, each of which contains the code of one outer postscript or pspicture environment.



5.2 pst2pdf step 2

Next, each sub-document is run with the command sequence latex → dvips → ps2pdf → pdfcrop, to create eps and pdf versions of the sub-document, which will by default be cropped to get rid of any surrounding whitespace. The Perl script also supports (on Unix-ish systems) other output formats, e.g., png.



5.3 pst2pdf step 3

The last step is a pdflatex run that replaces all extracted environments with the created image of the sub-documents. An original source:

```
some text...
\begin{postscript}
\includegraphics{image.eps}
\end{postscript}
... more text ...
\begin{pspicture}(3,7)
...
\end{pspicture}
... again some normal text
```

will be modified to:

```
some text...
\includegraphics{image1.pdf}
... more text ...
\includegraphics{image2.pdf}
... again some normal text
```

As shown, the last pdflatex run now uses the original document without any PostScript-related code; everything is inserted as a PDF image.

5.4 Optional arguments for pst2pdf

```
# all image types (png not on Windows):
my @imageType = ("eps","pdf","png");

# directory in which to save the images:
my $imageDir = "images";

# leave empty if not special:
my $Iext = ".png";

my $tempDir = "."; # temporary directory
my $verbose = 1;   # 0/1, logfile
my $clear = 0;     # 0/1, clears temporaries
my $DPI = 75;      # very low value
my $Iscale = 1;    # for \includegraphics
my $noImages = 0;  # 1->create no images
```

◇ Herbert Voß
DANTE e.V.
http://tug.org/PSTricks