LaTeX to PDF

Introduction

Making Portable Document Format (PDF) files from LaTeX source is a little tricky, because the PDF file must incorporate not only the images for any figures, but also the font glyphs (or at least, partial fonts) for anything outside the standard handful of fonts in the basic PostScript set. In particular, this means most mathematical characters, Greek letters, and the like — even if they're in the normal Symbol font.

If this isn't done right, you get the kind of mess frequently downloaded from the Web: documents with missing characters, or wrong characters. The latter problem comes from differences in font **encoding** between the system the assembled the PDF file and the one on which it's displayed. This is why all but the basic fonts really **must** be embedded in the PDF file, even though this makes the file bigger. Furthermore, the fonts that are embedded should be outline (Type 1) fonts, so that they'll look “clean” wherever they are displayed or printed.

Though there are a number of HOWTO documents and Web pages devoted to this problem, I haven't found a single page that combines all the necessary information in one place. Hence, this document.

How to proceed

What to do depends on what your raw materials (fonts and images) are like, and on exactly what products you require. Are your figures mostly PostScript vector graphics, or mostly rasterized images? Do you need *just* a PDF file, or do you also want a PostScript version to be printed? Do you need a DVI file for other reasons? Do you need to have hypertext links in the final PDF file?

Here's an overview of the general possibilities:

Traditional method

Traditionally, you converted your LaTeX source file to a DVI file, which could then be converted to PostScript with `dvips`. This, in turn, can be converted to a PDF file by `ps2pdf`:

```
           latex            dvips            ps2pdf
text.tex -------> text.dvi -------> text.ps --------> text.pdf
```

This requires **all** the graphics to be EPS files. But that's not a major problem, as raster graphics can be converted to EPS. Furthermore, the scalability of vector graphics means clean-looking figures at all resolutions. And vector PS is usually very compact.

But what about using photographs, which are usually saved as JPEGs? This really isn't a problem, because the `jpeg2ps` command (from Debian's `jpeg2ps` package) wraps JPEG images in an EPS header.

The main drawback is the large number of conversions; there are many places to make mistakes. Nevertheless, it *can* be done reliably [see details].

dvipdfm method

If you **don't** need PostScript output, you can save a step by going directly from DVI to PDF format by using `dvipdfm`:

```
           latex           dvipdfm
text.tex -------> text.dvi -------> text.pdf
```

Once again, the figures **must** be Encapsulated PS. So you have compact, scalable graphics — with one less step. [See details.]

Of course, this still produces a DVI file as an intermediate. Do you really need that? If not, there's `pdflatex`:

Pdflatex method

The `pdflatex` program produces a PDF file *directly* from the LaTeX source:

```
          pdflatex
text.tex --------> text.pdf
```

That looks pretty painless; but there's a catch. While the previous methods employ EPS exclusively as the graphics format, `pdflatex` won't accept EPS directly at all: you have to convert all the graphics to JPEG, PNG, or PDF (!) before compiling.

That isn't as bad as it sounds, because EPS can be “wrapped” with PDF headers to become PDF and still have scalable, vector graphics. And JPEG is a compact format for photographs, while PNG is a very compact way to store images with sharp outlines without introducing compression artifacts. (See details.)

Summary

So it mainly comes down to the products you need, if you're willing to do a little file-conversion work to get the images to the right format:

Need *just* PDF?
: Use `pdflatex`.

Need both DVI and PDF?
: Use `dvipdfm`.

Need both PS and PDF?
: Use the traditional method (`dvips` plus `ps2pdf`).

Finally, there's yet another way to proceed, if you want both PDF and DVI.

Each of these methods requires some care in interfacing the different steps, so that everything proceeds smoothly. Some special incantations are required in the LaTeX source, as well as the correct options to each subsequent processing step. The best strategy is to build a `makefile` with the correct options.

Here are the details.

Traditional method (details)

1. Convert *all* the figures to EPS.

If you have raw, *un*encapsulated PS, see here.

If you have JPEG images, use `jpeg2ps`.

If you have PBM/PGM/PPM/PNM or PNG raster images, see here for details, and here for a summary.

2. Tell LaTeX what to expect.

You have to tell the `latex` program to add instructions for the post-processors. So your preamble should look like this:

```
\documentclass[dvips,...]{article}      % the dvips is essential
\usepackage[dvips]{graphicx}            % to include images
\usepackage{pslatex}        % to use PostScript fonts

% redefine float-placement parameters here

\begin{document}
```

You might choose some other style than `article`. The important items here are the `dvips` options, and the `pslatex` package, which uses the standard PostScript outline fonts, instead of Computer Modern. See `/usr/share/doc/texmf/latex/pslatex`

`/00readme.txt` for more information on `pslatex`.

If you have no figures, you can omit the `\usepackage[dvips]{graphicx}` line. If you have neither figures nor tables (nor any other floats), you can omit the section of the preamble in which the float-placement parameters are given more reasonable values.

3. Include the figures in the `*.tex` file.

To insert a figure (contained in a file named `circle1.eps`) into the text, put something like

```
\begin{figure}[htp]
\resizebox{\textwidth}{!}{
        \includegraphics[width=\textwidth]{circle1}
        }
\caption{circle1}
\end{figure}
```

where you want the figure to appear in the text.

The `begin` – `end` pair delimits a `figure` environment, which uses the `\includegraphics` operator that was defined when the `graphicx` package was included in the preamble. Note the `[htp]` argument to `\begin{figure}`: that usually puts the figure where you want it, provided you have added the necessary parameter changes to the preamble.

Here, the graphic contained in the file `circle1.eps` is inserted in the document; you don't have to explicitly add the filename extension. The `width=` argument tells the `graphicx` package how to scale the figure (in this example, to match the width of the text). See the local documentation for the `graphicx` package for details on how to scale figures; the command

```
gv /usr/share/doc/texmf/latex/graphics/grfguide.ps.gz
```

should display it on a Debian system.

There is one variation on this theme that needs to be considered. I recently found myself with a figure too long to fit on the page, when its width was set to `\textwidth` as it is in the example above. I could shrink the image enough to fit by using `.7\textwidth` as the arguments of both the `\resizebox` and the `\includegraphics` commands; but the figure then appeared with its left edge flush with the left margin of the text in the caption.

The solution is to put a `\centering` command immediately after the `\begin{figure}` and before the `\resizebox` command.

Oddly enough, I got the same result by using just `\center` instead of `\centering`, which ought to be illegal. Apparently something is creating `\center` as an alias for `\centering` somewhere.

4. Build a `makefile` with the proper options.

It's easiest to get everything done correctly if you set up a makefile. Here's an example, which assumes your LaTeX source file is named `text.tex`:

```
# makefile for dvips

text.dvi: text.tex
        latex text.tex

text.ps: text.dvi
        dvips -Ppdf -G0 text.dvi
        # the -G0 fixes the ligature problem.

text.pdf: text.ps
        ps2pdf text.ps

PDF: text.pdf
        xpdf text.pdf
```

Don't forget that the indented lines in makefiles must be indented with a <TAB> character, not spaces.

Now you can just say `make` on the command line to make the DVI file, or `make text.ps` to create the PostScript version. When everything is working properly, you can say `make text.pdf` to make the PDF version; I've added a second target at the end of the makefile that lets you say just `make PDF` to do the same thing, and let you view the result with `xpdf`.

The `-Ppdf` option to `dvips` is required to make it include Type 1 (scalable) fonts in the PS file, so that the PDF comes out right.

The only thing tricky here is the `-G0` option to `dvips`. That works around a bug in versions of `dvips` earlier than 5.90; so it's needed in Debian woody, which has version 5.86e. The typical symptom of the bug is ligatures printing as some incorrect character: the `fi` ligature, for example, shows up as a £ sign. If `-G0` doesn't work for you, try `-G1`.

For the manual on `dvips`, use `texdoc dvips` or `info dvips`.

For the manual on `ps2pdf`, use a browser to display `/usr/share/doc/gs/Ps2pdf.htm`.

This PDF file is displayed correctly by `xpdf` at all magnifications. However, `acroread 5.0.8` has font problems: at magnifications below 600%, it displays a gray

rectangle for some of the smaller math characters — including, oddly enough, the period following the displayed equation. (I suppose it takes this to be a decimal point rather than a text period.) As `xpdf` displays the PDF correctly, and it prints correctly when sent to a PostScript printer from *both* `xpdf` and `acroread`, the problem is clearly in `acroread`, not the PDF file itself.

`dvipdfm` method (details)

1. Convert *all* the figures to EPS.

See the traditional method for how to do this.

2. Tell LaTeX what to expect.

The LaTex file needs almost the same preamble as for the traditional method:

```
\documentclass[dvipdfm,...]{article}    % the dvipdfm is essential
\usepackage[dvips]{graphicx}            % to include images
\usepackage{pslatex}        % to use PostScript fonts

\usepackage{mathptm}    % to use math fonts
\usepackage{mathptmx}   % to use math fonts

% redefine float-placement parameters here

\begin{document}
```

Of course, we now specify `dvipdfm` among the options to the `\documentclass` operator. But we still tell the `graphicx` package to prepare for the `dvips` operator, even though we don't use it.

The important addition here is to call two math packages to get the special characters needed for setting equations. Without them, glyphs for integral signs and certain other math symbols will not get included in the final PDF file.

3. Include the figures in the `*.tex` file.

Now the calls to `\includegraphics` **must** have the filename extensions:

```
\begin{figure}[htp]
\resizebox{\textwidth}{!}{
        \includegraphics[width=\textwidth]{circle1.eps}
        }
\caption{circle1}
\end{figure}
```

Again, don't forget the `[htp]` argument to `\begin{figure}`.

4. Build a `makefile` with the proper options.

Here's a sample makefile:

```
# makefile for dvipdfm

text.dvi: text.tex
        latex text.tex

text.pdf: text.dvi
        dvipdfm text.dvi

PDF: text.pdf
        xpdf text.pdf
```

Notice that this is simpler than the makefile for the traditional method. And we avoid the hassle with `dvips`. Once again, don't forget to use tabs instead of spaces for indentation in makefiles.

To display the manual for `dvipdfm`, use `mozilla file:///usr/share/doc/dvipdfm/dvipdfm.pdf.gz` on a Debian system.

Again there are font problems with `acroread 5.0.8`; but now they occur only at magnifications at and below 50%, where the grayed-out glyphs would hardly have been legible anyway. Again, the PDF file prints correctly, even when sent from `acroread`.

`pdflatex` method (details)

An even simpler makefile is possible if we use `pdflatex` to compile the input file. Now, however, there is one big change:

1. Convert *all* the figures to *non*-EPS form.

As `pdflatex` doesn't use EPS figures, we have to convert them all to either JPEG, PNG, or PDF form. It makes no sense to convert EPS to PNG, because that changes scalable graphics to bulky rasterised form. JPEG would be even worse, because of its lossy compression. The way to go is to convert EPS to PDF:

```
epstopdf image.eps
```

which creates a file named `image.pdf` by default. (This has the happy side-effect of compressing the actual PostScript text of the original EPS file; if that file were longer than about 2 kB, the compression would more than offset the added PDF headers, and the PDF file could be smaller than its EPS parent.)

2. Tell LaTeX what to expect.

The preamble to the LaTeX source file now looks much like the one used in the traditional method:

```
\documentclass[pdftex,...]{article}     % the pdftex is essential
\usepackage[dvips]{graphicx}            % to include images
\usepackage{pslatex}        % to use PostScript fonts

% redefine float-placement parameters here

\begin{document}
```

Note that the package name to put in the argument list is `pdftex`, not `pdflatex`.

3. Include the figures in the `*.tex` file.

Just as in the traditional method, we can omit the filename suffix:

```
\begin{figure}[htp]
\resizebox{\textwidth}{!}{
        \includegraphics[width=\textwidth]{circle1}
        }
\caption{circle1}
\end{figure}
```

But now, what's automatically found is the file `circle1.pdf`, not `circle1.eps`.

4. Build a `makefile` with the proper options.

The makefile is now exceedingly simple:

```
# makefile for pdflatex


text.pdf: text.tex
        pdflatex text.tex

PDF: text.pdf
        xpdf text.pdf
```

As before, `make PDF` makes and displays the PDF file. Once again, `xpdf` displays the result correctly, and `acroread 5.0.8` has font problems at 50% magnification and less. Both can print correctly.

To read the manual for pdftex/pdflatex, do

```
gv /usr/share/doc/texmf/pdftex/base/pdftexman.pdf.gz
```

Checking the PDF file

Of course, you will want to verify that the PDF file you produce displays correctly in both `xpdf` and `acroread`, and that it prints correctly when sent from either of them to your printer.

It's useful to run `pdffonts text.pdf` to make sure the final result really has the proper fonts embedded. You should see **yes** in the column headed **emb** for every font, *except* for the Basic 14 Adobe PostScript fonts (like Times-Roman and Symbol).

Conclusion

Because of the font problems with Acroread 5, it seems best to use either `dvipdfm` or `pdflatex` to make PDF files from LaTeX source. The former can't handle bold math fonts gracefully; and besides, pdflatex allows JPEGs to be used without encapsulation, accepts PNG raster graphics, and compresses the PostScript text of EPS files (though it does bulk up the latter a few hundred bytes with the required PDF wrapper.)

The best general choice seems to be `pdflatex`.

Back to the . . .
main LaTeX page

or the **LaTeX formatting page**

or the **figures-in-LaTeX page**

or the **website overview page**