MAX–PLANCK–INSTITUT FÜR AERONOMIE

D–37191 Katlenburg-Lindau, Federal Republic of Germany

Graphics and Colour with LaTeX

Patrick W. Daly

1998 June 4

[This document is also available in PDF and HTML form.]

Contents

1 Introduction

The advent of PostScript drivers for TeX has offered the opportunity to expand the capabilities of LaTeX to include a number of very useful features, such as the importation, scaling, and rotation of external graphics files, and colour. However, since each driver tended to have its own packages and user interface commands, the resulting LaTeX document files threatened to become tied to one particular driver.

With LaTeX 2$_\varepsilon$, a standardized set of commands is made available for all drivers by means of the packages `graphics` and `color`. The specialized coding to translate the user commands into instructions for the specific drivers is placed in certain `.def` files which are loaded by options to these packages. Thus one only needs to change the option to switch to another driver; the main text remains unchanged.

The commands defined by these packages are the building blocks for other packages that either emulate the older driver-specific ones or provide a more comfortable syntax for these functions. As long as these other packages are based on `graphics` and `color`, they should be equally compatible with all the supported drivers.

2 Importing external graphics

The problem is to have a graphics file produced by some other program included in the document, most likely as a figure. It may be necessary to scale it or to rotate it by 90°. One wants to do by computer what used to be done with scissors and glue.

There are two packages available for importing and manipulating external graphics files: the more basic `graphics` and the more extended `graphicx`. They both have identical functionality, differing only in syntax.

In addition to the driver names, there are some other options that may be invoked when the packages `graphics` or `graphicx` are loaded:

`draft` does not import but places a framed box where the graphic would appear, with the file name printed inside; this speeds up the processing considerably when one is only working on the text;

`final` counteracts `draft`; needed when the `draft` option has been issued globally in `\documentclass`;

`hidescale`
leaves blank space where scaled text should be;

`hiderotate`
leaves blank space where rotated text should be; this and the option `hidescale` are useful if the previewer cannot handle scaling or rotation;

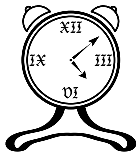
`hiresbb`
: look for bounding box values in `%%HiResBoundingBox` instead of the the normal `%%BoundingBox` line.

2.1 Importing with the graphics package

The basic importing command with the `graphics` package is

`\includegraphics[`*llx,lly*`][`*urx,ury*`]{`*file_name*`}`

where *llx, lly* are the coordinates of the lower left corner, and *urx, ury* those of the upper right corner of the *bounding box* containing the part of the picture that is to be included. In other words, they say where the scissors are to be applied. Units may be specified (like `[3cm, 2in]`) but if they are omitted, big points (`bp`, 72 per inch, 28.3464... per cm) are assumed. If only one optional argument is given, it is the upper right corner, and the lower left is assumed to be `[0,0]`.



If no bounding box coordinates are given, the driver will obtain them some other way, depending on the type of graphics file. For example, for the very common *encapsulated PostScript* files with extension `.eps`, the bounding box information is extracted from the graphics file itself. The figure at the right is stored in such a file and is included simply with the command

`\includegraphics{clock.eps}`

With `\includegraphics*`, the figure is clipped, so that any drawing outside the specified bounding box is suppressed. This is useful if only part of a figure is to be reproduced. It is also vital with some perverse figures that paint the whole page white!

2.2 Scaling

The graphics file in the above commands is transferred to the document in its original size. In order to rescale it, there are two commands available:

`\scalebox{`*h_scale*`}[`*v_scale*`]{`*text*`}`

which applies horizontal and vertical scale factors to the contents *text*; if *v_scale* is omitted, it is the same as *h_scale*;

`\resizebox{`*h_length*`}{`*v_length*`}{`*text*`}`

adjusts the figure to fit into the specified horizontal and vertical sizes; if either length is given as `!`, the one scale factor is used for both dimensions. A *-form allows *v_length* to refer to the height + depth of the box, rather than just to the height. In both cases, the contents *text* may be an `\includegraphics` command, but it may also be any arbitrary text.

2.3 Reflection

The contents of a box may be reflected horizontally with

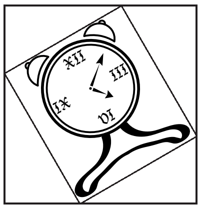
`\reflectbox{`*text*`}`

2.4 Rotation

Rotation of a box about the left-hand end of its baseline is done with

`\rotatebox{`*angle*`}{`*text*`}`

where *angle* is in degrees, and the rotation is counterclockwise.



To illustrate this, we have scaled the previous clock figure to a height of 2 cm and then rotated it by 30°, using the commands

```
\rotatebox{30}{\resizebox{!}{2cm}{%
  \includegraphics{clock.eps}}}
```

For demonstration purposes, we have added framed boxes around the figure before and after the rotation. It is the inner (tilted) box that is 2 cm high, while the overall outer box is somewhat higher and broader due to its inclined contents. Without these frames the figure seems to have extraneous space above and to the left, something that can be very puzzling even to experienced users. (The frames would not normally appear with the above commands.)

2.5 Importing, scaling, rotating with the `graphicx` package

If one selects the `graphicx` rather than the `graphics` package, a different interface is available for both importing and rotation, one making use of keys and values:

`\includegraphics[`*key*`=`*value*`,...]{`*file_name*`}`

The keys are of two types: those that take a numerical value, and those that are flags with the values `true` or `false`. Simply giving the name of a flag without a value is equivalent to setting it to `true`. Possible keys and their values are:

`scale=` *number*; enters the number by which the figure size should be magnified over its natural size;

`width=` *length*; specifies the width to which the figure should be scaled to; if `height` not given, it is scaled with the same factor as the width;

`height=` *length*; specifies the height to which the figure should be scaled to; if `width` is not given, it is scaled with the same factor as the height;

`totalheight=` *length*; like `height` but specifies the height plus depth of the figure; should always be used in place of `height` if the figure has been rotated;

`keepaspectratio` (=`true`/`false`); if both `height` and `width` are specified, this flag ensures that the original height/width ratio remains unchanged; the figure will not exceed either of the given dimensions;

`angle=` *number*; the angle by which the figure is to be rotated counterclockwise, in degrees; any `height` or `width` specifications coming before this key are also rotated, so that the height becomes the width, while the width becomes either the height (positive angle) or depth (negative angle);

`origin=` *loc*; determines the point about which the rotation occurs; default is `bl` for bottom left corner; also possible are `c` for center, `t` for top, `r` for right, and `B` for baseline; any sensible combination, such as `tr`, is allowed;

`draft` (=`true`/`false`); like the `draft` package option but applied to the one graphics file; the figure is not imported, but rather a framed box of the correct size is printed containing the name of the file;

`clip` (=`true`/`false`); suppresses the printing of any graphic outside the bounding box;

`bb=` *llx lly urx ury*; enters the coordinates of the bounding box manually, if they are missing or incorrect in the graphics file, or to be deliberately altered; the specifications are four lengths separated by blanks; units may be given, but if omitted, big points (`bp`) are assumed;

`viewport=` *llx lly urx ury*; specifies the bounding box but relative to the lower left corner of the existing one; useful for correcting the bounding box, or (with `clip`) to select only a portion of the whole figure;

`trim=` *dllx dlly durx dury*; reduces the existing bounding box by the amounts specified;

`hiresbb` (=`true`/`false`); like the `hiresbb` package option but applied to the one graphics file; reads bounding box information from the `%%HiResBoundingBox` line in the graphics file.

The keys are all optional; they are included as needed. Their order is not important other than that `angle` can change previous `height` and `width` meanings. The sets of key/values are separated from each other by commas.

With the key/value syntax, the tilted, scaled graphic on page 4 is produced with

```
\includegraphics[height=2cm,angle=30]{clock.eps}
```

For compatibility with the `graphics` package, there also exists an `\includegraphics*` version that clips the imported figure; this is equivalent to including the key `clip`.

With the `graphicx` package, the `\rotatebox` command is similarly redefined to accept the optional key `origin`.

3 Additional graphics packages

3.1 The `epsfig` package

Sebastian Rahtz has provided a package `epsfig` that not only updates the earlier (2.09) version, but also re-implements Rokicki's `epsf` package by means of the `graphics` commands. This is helpful for users who are accustomed to those syntaxes. For `epsf`, this is

`\epsfysize=`*y_size* or `\epsfxsize=`*x_size*
`\epsf[`*llx lly urx ury*`]{`*file_name*`}`

The `epsfig` package also defines an importing command that makes use of the regular keys and values to enter its parameters:

`\epsfig{file=`*file_name*`,`*key*`=`*value*`,...}`

For compatibility with some older versions, there is a `\psfig` command which is synonymous with the above.

The `epsfig` package is included in the bundle of `graphics` packages and drivers.

3.2 The `lscape` package

Another extra package in the `graphics` bundle is `lscape`, by David Carlisle. This defines a `landscape` environment that prints its contents rotated 90°on a page for itself. Head and footlines remain as normal. This is intended primarily for inserting figures that are in landscape mode, that is, wider than they are high.

3.3 The `rotating` package

The `rotating` package by Sebastian Rahtz and Leonor Barroca tries to make the interface for rotation somewhat simpler. It defines

`\begin{sideways}` *text* `\end{sideways}`
`\begin{turn}{`*angle*`}` *text* `\end{turn}`
`\begin{rotate}{`*angle*`}` *text* `\end{rotate}`
`\turnbox{`*angle*`}{`*text*`}`

where `sideways` rotates *text* by 90°, `turn` by an arbitrary angle. The environment `rotate` and command `turnbox` are equivalent: they rotate but in a box of zero size, so that the contents overlap the surroundings.

This package is not part of the `graphics` bundle and must be obtained separately.

4 Troubleshooting

4.1 How importation works

In order to understand better what can go wrong when a graphics file is imported, and to know what to do about, it is important to realize how the interplay between LaTeX and the driver program functions.

LaTeX has no idea what is in the graphics file; for it, the figure is simply a box of a given height, width, and depth, as indeed are all the characters that LaTeX processes. The information on the graphic's natural size is somehow obtained, by reading `%%BoundingBox` line in the graphics file itself, or in some other file, or through the optional entries in the `\includegraphics` command or equivalent. After scaling and rotating, LaTeX knows the final size that it must reserve in the output text for the figure.

What is then written to the `.dvi` file is the name of the graphics file and information on how it should be transformed. Just how this information is coded depends on the graphics driver selected. When the printer driver program processes the `.dvi` file, it interprets these special instructions, reads in the specified graphics file, performs the transformations, and places the result where LaTeX has said it should go. The end result is that the area inside the designated bounding box coincides with the box that LaTeX has reserved for it. If the bounding box information is incorrect, the figure is obviously going to be misplaced.

The most common type of graphics file to be imported is *encapsulated PostScript*. Files adhering to this standard are intended to be included within other PostScript files, and so must not contain certain PostScript commands that reset the whole graphics page. Most importantly, it must contain a comment line (ignored by PostScript itself) of the form:

`%%BoundingBox:` *llx lly urx ury*

giving the coordinates of the lower left and upper right corners of the bounding box. The units are never specified, always being big points.

Having pointed out how the importation takes place, we can now discuss what can go wrong along this chain of processes.

4.2 Problems with importation

The most common problems encountered when importing encapsulated PostScript files are listed here.

No bounding box. If the bounding box information is totally missing from the graphics file, LaTeX issues the error message

```
! LaTeX Error: Cannot determine size of graphic
                                in ... (no BoundingBox).
```

The solution is to determine the bounding box coordinates somehow (see next point) and to include them, either in the `\includegraphics` command, or by editing the graphics file itself. However, if there really is no bounding box information in the file, it is unlikely to conform to the encapsulated standard and will cause other problems.

The placement is incorrect. Both the LaTeX and driver process without error messages, but the figure is either displaced from the expected position, or is far too small.

Most likely the bounding box information is incorrect. Many applications that produce PostScript files are too lazy to calculate the true bounding box, or they think they are generating a whole page with a figure somewhere in the middle. In either case, the bounding box corresponds to the full page even though the printed figure occupies only a portion of it.

Find the true bounding box by one of the following methods:

1. Print the figure, mark the lower left and upper right corners of the box containing the figure, and measure their distances from the left and bottom edges. Enter these distances in the `\includegraphics` command, or edit the PostScript file. In the latter case, convert to big points.

   Difficulties with this are that some encapsulated PostScript files cannot be printed on their own, and that the left and bottom edges of the paper need not be the exact lines from which the printer really measures.

2. Include the figure in a short LaTeX file such as

   ```
   \setlength{\fboxsep}{-\fboxrule}
   \fbox{\includegraphics{test.eps}}
   ```

   and print the output. The apparent bounding box appears as a framed box. Measure the true bounding box relative to the left and bottom edges of the apparent one, and enter the values in the `\includegraphics` command with the `viewport=` key. If necessary, scale the figure down to fit on the page, but then remember to increase the measurements by the same scale factor.

3. Use the GhostView program to fix up the bounding box, either automatically or manually. This is the most convenient method if you have this utility for viewing and manipulating PostScript files.

Immovable graphic. It does not shift, nor scale nor rotate, no matter what is specified. In this case, it violates the encapsulated PostScript rules and contains some global plotting commands. Graphics produced by word processing programs are notorious for this. Often the offending command is `setpagedevice`.

There is little that can be done to correct this, other than trying to regenerate the graphics file with an option for encapsulated PostScript. Judicious editing can remove the troublesome lines, but this could result in the file becoming totally unreadable.

5 Configuring graphics importation

Although the graphics syntax has been standardized, and most of the driver-specific coding hidden in the `.def` files, there are still a number of items that must be set up for any particular installation and operating system. These are most conveniently placed in the local configuration file `graphics.cfg` which is read in if it is present.

5.1 Default driver

The choice of driver option must always be specified, but a local default can be established by placing the line

`\ExecuteOptions{`*driver*`}`

into the configuration file, where *driver* is one of the allowed driver options. This may be overridden by any explicit driver option specified in the document file itself.

The rest of the configuring commands in this section can be issued either in the `graphics.cfg` file, or in the document.

5.2 Search path for graphics files

One can specify the directories where LaTeX is to look for graphics files with

`\graphicspath{`*dir_list*`}`

where *dir_list* is a list of directory names, separated by commas. The syntax of the local operating system must be used. Without this command, LaTeX searches for graphics files in the same directories as for all other TeX files.

5.3 Default extensions

A list of default extensions for the graphics files can be defined with

`\DeclareGraphicsExtensions{`*ext_list*`}`

This means that only the root name of the file must be given and LaTeX will attempt to find it by attaching all the possible extensions. For PostScript drivers, the *ext_list* is usually set to `.eps,.ps`. At our installation, we also include the non-standard extension `.psc`. Note that the above command does not add to the list of extensions but rewrites it anew; if you wish to add to the list, you must include all the allowed extensions in the one declaration.

5.4 Graphics types

Defining the extensions is only part of the task: one must also associate each extension with a graphics type so the LaTeX knows how to process it. PostScript recognizes only one type, `eps`, encapsulated PostScript, but there do exist other types such as `bmp` and `pcx` for other drivers. For the non-standard `.psc` extension above, we must also give

```
\DeclareGraphicsRule{.psc}{eps}{}{}
```

to inform LaTeX that this extension belongs to type `eps`. The other two (empty) arguments specify that the bounding box information is to be read from the file itself, and that no other program needs to be applied to the file.

5.5 Compressing graphics files

Since PostScript files are often extremely large, it makes sense to try to compress them with either the `zip` or `gzip` programs. In such a case, the `.eps` file is replaced by a file with extension `.zip`, or `.eps.gz`, or `.eps-gz`. Two problems now arise: first, LaTeX cannot read such files to obtain the bounding box information, and secondly, the driver needs to unpack such a file to include it in the final output. This can be accomplished with, for example,

```
\DeclareGraphicsRule{.eps.gz}{eps}{.eps.bb}{`gunzip -c #1}
```

which establishes the graphics type as `eps`, with the bounding box information in the file of the same name and extension `.eps.bb`, and that the operating system command `gunzip -c` must be applied to the file (represented as `#1`). The single quote ` is required to indicate a system command. The `%%BoundingBox` line of the original file must be copied and stored in the `.eps.bb` file.

Such decompression rules are system dependent and thus need to be configured for the local installation. For example, under the VMS operating system, the `gzip` program produces files with extension `.eps-gz` and decompression is performed with `gzip -d` rather than with `gunzip`. The corresponding rule becomes

```
\DeclareGraphicsRule{.eps-gz}{eps}{.bb}{`gzip -d -c #1}
```

6 Colours

The `color` package recognizes the options:

`monochrome`
: to convert all colour commands to black and white, for previewers that cannot handle colour;

`dvipsnames`
: makes the `named` colour model of `dvips` (Section 7) available to other drivers;

`nodvipsnames`
: disables the `named` model for `dvips`, to save memory;

`usenames`
: loads all the `named` colours as defined ones; again, see Section 7 for details.

A local configuration file `color.cfg` can be set up in the same way as for the `graphics` package. The default driver option is specified in exactly the same way as in Section 5.

Colours are specified either by a defined name, or by the form

`[`*model*`]{`*specs*`}`

where *model* is one of `rgb` (red, green, blue), `cmyk` (cyan, magenta, yellow, black), `gray`, or `named`. The *specs* is a list of numbers from 0 to 1 giving the strengths of the components in the model. Thus `[rgb]{1,0,0}` defines red, `[cmyk]{0,0,1,0}` yellow. The `gray` model takes only one number. The `named` model accesses colours by internal names that were originally built into the `dvips` driver, but which may now be used by some other drivers too. This model is described in Section 7.

A colour can be defined with

`\definecolor{`*name*`}{`*model*`}{`*specs*`}`

and then the *name* may be used in all the following colour commands. Certain colours are automatically predefined for all drivers: `red`, `green`, `blue`, `yellow`, `cyan`, `magenta`, `black`, `white`.

In the following colour commands, *col_spec* is either the name of a defined colour, like `{blue}`, or `[`*model*`]{`*spec*`}`, like `[rgb]{0,1,0}`.

`\pagecolor` *col_spec* sets the background colour for the current and following pages;

`\color` *col_spec* is a declaration to switch to setting text in the given colour;

`\textcolor` *col_spec*`{`*text*`}` sets the text of its argument in the given colour;

`\colorbox` *col_spec*`{`*text*`}` sets its argument in a box with the given colour as background;

`\fcolorbox` *col_spec1 col_spec2*`{`*text*`}` is like `\colorbox`, with a frame of *col_spec1* around a box of background colour *col_spec2*; the two specifications must either both be defined ones, or both use the same model, which is given only once; for example, `\fcolorbox{red}{green}{Text}` sets 'Text' in the current text colour on a green background with a red frame;

`\normalcolor` switches to the colour that was active at the end of the preamble. Thus placing a `\color` command in the preamble can change the standard colour for the whole document. This is the equivalent to `\normalfont` for font selection.

Normally one would try to define all the colours needed as names for the *col_spec* entries. This simplifies changing the colour definition everywhere should fine tuning be required after the initial printed results are seen. The same colour definition can produce quite different effects on different printers. Even the display on the monitor is no reliable guide as to how the output will appear on paper.

7 The `named` colour model

One very useful colour model is called `named` and is based on the 68 predefined internal colours of the `dvips` PostScript driver. Sample names are `BurntOrange` or `DarkOrchid`. This model can be activated for other drivers with the option `dvipsnames`, in which case one can define colours as, for example

```
\definecolor{titlecol}[named]{DarkOrchid}
```

The colour `titlecol` can then be used as *col_spec* in the various colour commands.

The named colours can be defined with their own names if one invokes the option `usenames`, which effectively declares

```
\definecolor{BurntOrange}[named]{BurntOrange}
```

and so on, for all 68 colours.

It is possible to generate a palette of the named colours by processing the following short LaTeX file and sending the output to the desired printer.

```
\documentclass[12pt,a4paper]{article}
\usepackage[dvips]{color}
\usepackage{multicol}
\pagestyle{empty}
\setlength{\oddsidemargin}{0pt}
\setlength{\textwidth}{16cm}
\setlength{\textheight}{22cm}
\setlength{\parindent}{0pt}
\setlength{\parskip}{0pt}

\begin{document}
\renewcommand*{\DefineNamedColor}[4]{%
   \textcolor[named]{#2}{\rule{7mm}{7mm}}\quad
   \texttt{#2}\strut\\}

\begin{center}\Large Named colours in \texttt{dvipsnam.def}
\end{center}
\begin{multicols}{3}
\input{dvipsnam.def}
\end{multicols}
\end{document}
```

Remember, each printer can reproduce the colours differently, so it is important to test this table with every colour printer that might be used. The output of this job is shown in the next page.

Named colours in `dvipsnam.def`

GreenYellow
Yellow
Goldenrod
Dandelion
Apricot
Peach
Melon
YellowOrange
Orange
BurntOrange
Bittersweet
RedOrange
Mahogany
Maroon
BrickRed
Red
OrangeRed
RubineRed
WildStrawberry
Salmon
CarnationPink
Magenta
VioletRed
Rhodamine
Mulberry
RedViolet
Fuchsia
Lavender
Thistle
Orchid
DarkOrchid
Purple
Plum
Violet
RoyalPurple
BlueViolet
Periwinkle
CadetBlue
CornflowerBlue
MidnightBlue
NavyBlue
RoyalBlue
Blue
Cerulean
Cyan
ProcessBlue
SkyBlue
Turquoise
TealBlue
Aquamarine
BlueGreen
Emerald
JungleGreen
SeaGreen
Green
ForestGreen
PineGreen
LimeGreen
YellowGreen
SpringGreen
OliveGreen
RawSienna
Sepia
Brown
Tan
Gray
Black
White