CS_189 LaTeX and Linux: Graphics Inclusion and Color

(1) Robert S. Laramee

Computer Science Department
School of Physical Sciences
Swansea University

March 11, 2008

Overview of Lecture

On Graphics Inclusion

- LaTeX is not a graphics generation program. The business of creating graphics is a complex subject all by itself.
- But LaTeX, like other word processing software, has features to let the author include graphics and pictures into a document as well as perform some simple operations on those images.
- LaTeX also has some packages that let authors create simple diagrams and line drawings.
- The main topic of this lecture is on how to include graphics from other sources into a .tex document.

Terminology: Postscript

A page description language (PDL) developed by Adobe Systems. PostScript is primarily a language for printing documents on laser printers. PostScript is the standard for desktop publishing because it is supported by imagesetters, the very high-resolution printers used by service bureaus to produce camera-ready copy.

PostScript is an object-oriented language. It treats images, including fonts, as collections of geometrical objects rather than as bit maps.

The principal advantage of object-oriented (vector) graphics over bit-mapped graphics is that object-oriented images take advantage of high-resolution output devices whereas bit-mapped images do not. A PostScript drawing looks much better when printed on a 600-dpi printer than on a 300-dpi printer.

From `www.webopedia.com`

The Graphics Packages

There are two graphics packages:

- \usepackage{graphics} The 'standard' graphics package.
- \usepackage{graphicx} The 'extended' or 'enhanced' graphics package.

The two differ only in the format of optional arguments for the commands defined. The command names, and the mandatory arguments are the same for the two packages.

Tip: We recommend using the graphicx package on Linux.

Graphics Package Options

The graphics packages have some other options for controlling how many of the features to enable:

draft suppress all 'special' features. i.e., graphics files are not included (but they are still read for size) just the filename is printed in a box of the correct size.

final The opposite of `draft`. Useful to over-ride a global `draft` option specified in the `\documentclass` command.

hiresbb Look for size specifications in %%HiResBoundingBox lines rather than standard %%BoundingBox lines.

demo Instead of inserting an image file `\includegraphics` draws a 150 pt by 100 pt rectangle (unless other dimensions are specified manually).

Rotation

```
\rotatebox{angle}{text}                          % graphics
\rotatebox[key value list]{angle}{text}   % graphicx
```

This puts *text* in a box, like \mbox, but rotates the box through *angle* degrees:

Example

```
\rotatebox{15}{like this}
```

like this

The standard version always rotates around the reference point of the box, but the key value list version takes the following keys:

```
|origin| = {label}
|x|      = {dimen}
|y|      = {dimen}
|units|  = {number}
```

Rotation

Thus you may specify both x and y, which give the coordinate of the centre of rotation relative to the reference point of the box, eg., `[x=2mm, y=5mm]`.

Alternatively, for the most common points, one may use `origin` with a *label* containing one or two of the following: **lrctbB** (**B** denotes the baseline).

For example, compare default rotation of 180° ...Like This... to effects gained by using `origin` key:
`[origin = c]` rotates about **c**enter of box ...Like This...
`[origin = tr]` rotates about **t**op **r**ight-hand corner: ...Like This...

The `units` key allows a change from the default units of degrees anti-clockwise. Give the number of units in one full anti-clockwise rotation.

Scaling: By a Scale Factor

`\scalebox{h-scale}{v-scale}{text}`

For example:
`[units = -360]` specifies degrees clockwise.
`[units = 6.283185]` specifies radians.

Like `\mbox` but scales the *text*. If *v-scale* is not specified it defaults to *h-scale*. If it is specified the text is distorted as the horizontal and vertical stretches are different, Like This.

`\reflectbox{text}`

An abbreviation for `\scalebox{-1}{1}{text}`:

Example

`\reflectbox{reflected text}` normal text
ʇxǝʇ pǝʇɔǝlɟǝɹ normal text

Scaling: To a Requested Size

`\resizebox{h-length}{v-length}{text}`

Scales *text* so that the width is *h-length*. If ! is used as either length argument, the other argument is used to determine a scale factor that is used in both directions. Normally *v-length* refers to the height of the box, but in the star form, it refers to the 'height + depth'.

As normal for LaTeX 2_ε box length arguments, `\height`, `\width`, `\totalheight`, `\depth` may be used to refer to the original size of the box.

Scaling: To a Requested Size (Continued)

Example

```
\resizebox{1in}{\height}{Some text}
```

Some text

```
\resizebox{1in}{!}{Some text}
```

Some text

Recall: \depth is the distance from baseline to the bottom (of a box).

Including Graphics Files

Include a graphics file using:

```
\includegraphics[llx, lly][urx, ury]{file}  % graphics
\includegraphics[key val list]{file}        % graphicx
```

If * is present, then the graphic is 'clipped' to the size specified. If * is omitted, then any part of the graphic that is outside the specified 'bounding box' will over-print the surrounding text.

If the optional arguments are omitted, then the size of the graphic will be determined by reading an external file as described below.

- *file* should not include a filename extention. LaTeX will automatically search for a file format it can recognize.
- `pdflatex` will automatically recognize .jpeg files.
- `latex` will automatically recognize .eps (extended postscript) files.

Including Graphics Files: graphics vs graphicx

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

If [urx, ury] is present, then it should specify the coordinates of the top right corner of the image, as a pair of TEX dimensions. If the units are omitted they default to `pt` (point). So `[1in,1in]` and `[72,72]` are equivalent. If only one optional argument appears, the lower left corner of the image is assumed to be at `[0,0]`. Otherwise `[llx, lly]` may be used to specify the coordinates of this point.

`\includegraphics[key val list]{file}`

Here the star form is just for compatibility with the standard version. It just adds `clip` to the list of keys specified.

The allowed keys are listed below.

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

Including Graphics Files: graphicx Options

width=length Required width. The graphic is scaled to this width.

height=length Required height. The graphic is scaled to this height.

totalheight=length Specify the total height (height + depth) of the figure. This will differ from the 'height' if rotation has occurred. In particular if the figure has been rotated by -90° then it will have zero height but large depth.

keepaspectratio (=true|false) Boolean valued key like 'clip'. If set to true then specifying both 'width' and 'height' (or 'totalheight') does not distort the figure but scales such that neither of the specified dimensions is *exceeded*.

angle=number Rotation angle.

origin Origin for rotation. See `\rotatebox`.

Including Graphics Files: graphicx Options (continued)

draft (=true|false) a boolean valued key, like 'clip'. Locally switches to draft mode.

clip (=true|false) Either 'true' or 'false' (or no value, which is equivalent to 'true'). Clip the graphic to the bounding box.

bb=llx lly urx ury The argument should be four dimensions, separated by spaces. These denote the 'Bounding Box' of the printed region within the file.

viewport The `viewport` key takes four arguments, just like `bb`. However in this case the values are taken relative to the origin specified by the bounding box in the file. So to 'view' the 1in square in the bottom left hand corner of the area specified by the bounding box, use the argument `viewport=0 0 72 72`.

Including Graphics Files: graphicx Options (continued)

trim Similar to viewport, but here the four lengths specify the amount to remove or add to each side. `trim= 1 2 3 4` 'crops' the picture by 1bp at the left, 2bp at the bottom, 3bp on the right and 4bp at the top.

hiresbb (=true|false) Boolean valued key. If set to `true` (just specifying `hiresbb` is equivalent to `hiresbb=true`) then TEX will look for `%%HiResBoundingBox` lines rather than `%%BoundingBox`. It may be set to `false` to overrule a default setting of `true` set by the `hiresbb` package option.

Including Graphics Files: Notes on graphicx Parameters

- For the parameters specifying the original size (i.e,, the bounding box, trim and viewport keys) the units can be omitted, in which case bp (i.e., PostScript points) are assumed.
- Several parameters specify the original size of the image. This size needs to be specified in the case that the file can not be read by TEX, or it contains an incorrect size 'BoundingBox' specification.
- **Tip:** This problem can be solved also by using the gimp to save your files in the correct format, especially postscript (.ps) and encapsulated postscript (.eps) file formats.
- Parameters are read left-to-right, so `[angle=90, height=1in]` means rotate by 90 degrees, and then scale to a height of 1 inch. `[height=1in, angle=90]` would result in a final *width* of 1 inch.

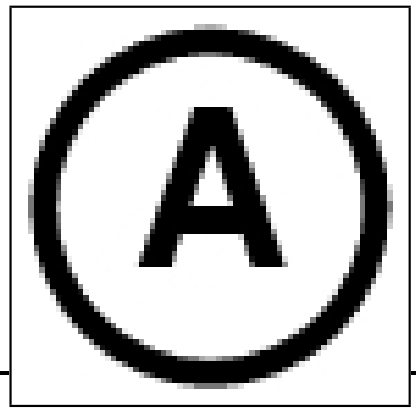
Including Graphics Files: Notes on graphicx Parameters

- TEX leaves the space specified either in the file, or in the optional arguments. If any part of the image is actually outside this area, it will by default overprint the surrounding text. If the star form is used, or `clip` specified, any part of the image outside this area will not be printed.

Including Graphics Files Example: No Options

Example

```
left--- \fbox{ \includegraphics{a} } ---right
```

left—  —right

Note: The graphic is stored in a file called `a.jpg` and processed using `pdflatex`.

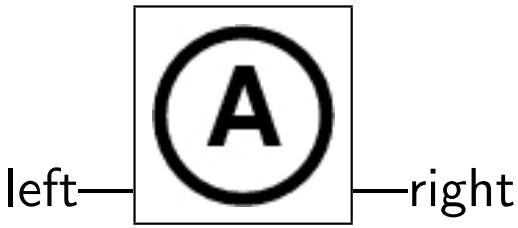
Including Graphics Files Example: Scaling

Syntax:

```
\scalebox{0.5}{\includegraphics{a}}    % graphics
\includegraphics[scale=.05]{a}         % graphicx
```

Example

```
left---\fbox{ \includegraphics[scale=0.5]{a} }---right
```



Including Graphics Files Example: Bounding Box

Syntax:

```
\includegraphics[10, 10][50, 50]{a}    % graphic
\includegraphics[bb = 10 10 50 50]{a}  % graphicx
```

Example

```
left---\fbox{ \includegraphics[bb = 10 10 50 50]{a} }---right
```

left—  —right

Note: The actual size of the `a.jpg` image is 66 by 66 points.

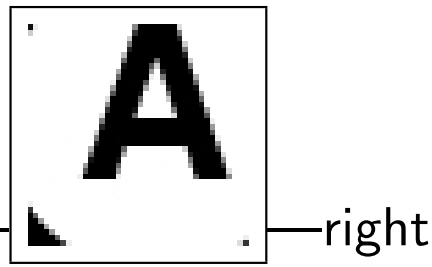
Including Graphics Files Example: Clipping

Syntax:

```
\includegraphics*[10, 10][50, 50]{a}        % graphics
\includegraphics[bb= 10 10 50 50, clip]{a}  % graphicx
```

Example

```
left---\fbox{\includegraphics[bb= 10 10 50 50,clip]{a}}---right
```



Other Commands: \graphicspath

\graphicspath{directory-list}

This optional declaration may be used to specify a list of directories in which to search for graphics files. A list of directories, each in a {} group (even if there is only one in the list). For example:

\graphicspath{{eps/}{tiff/}}

would cause the system to look in the subdirectories eps and tiff of the current directory.

The default setting of this path is \input@path that is: graphics files will be found wherever TEX files are found.

Common Problems when Importing Graphics (1)

```
! LaTeX Error: Unknown graphics extension: .eps
```

This type of error can occur when `latex` or `pdflatex` does not recognize the type of image you are importing.

Rule of Thumb:

- `pdflatex` recognizes JPEG (.jpg) files and PNG (.png) files.
- `latex` recognizes Postscript (.ps) files and Encapsulated Postscript (.eps) files.

Solution:

- `convert` will transform just about any graphic image in any format to any other format:
  `%> convert fileName.ps fileName.jpg`

Common Problems when Importing Graphics (2)

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

This is a common error when processing your files using `latex`.

Rule of Thumb:

- High quality software like `xfig` and the `gimp` will produce postscript files containing explicit bounding box information.

Solution:

- Open your graphic file using the `gimp` and save it in postscript (.ps) or encapsulated postcript (.eps) format (Pay attention to the file name extension.)
- The `gimp` is a free, open source, feature-rich image processing program very comparable to Adobe Photoshop.

Color

Color support is built around the idea of a system of *Color Models*. The Color models supported by a driver vary, but typically include

rgb Red Green Blue: A comma separated list of three numbers between 0 and 1, giving the components of the color.

cmyk Cyan Magenta Yellow [K]Black [1] : A comma separated list of four numbers between 0 and 1, giving the components of the color according to the additive model used in most printers.

gray Grey scale: a single number between 0 and 1.

[1] The 'K' in CMYK stands for key, as in four-color printing, cyan, magenta, and yellow printing plates are carefully keyed or aligned with the key line of the black key plate. Some sources suggest that the K in CMYK comes from the last letter in black. –www.wikipedia.org

Color: Color Models

named Colors accessed by name, e.g. 'JungleGreen'. Not all drivers support this model. The names must either be 'known' to the driver or added using commands described in —color.dtx—. Some drivers support an extended form of the named model in which an 'intensity' of the color may also be specified, so 'JungleGreen, 0.5' would denote that color at half strength.

Note that the **named** model is really just given as an example of a color model that takes names rather than a numeric specification. Other options may be provided locally that provide different color models, eg **pantone** (An industry standard set of colors), **x11** (Color names from the X Window System), etc. The standard distribution does not currently have such models.

Terminology: Driver

A program that controls a device. Every device, whether it be a printer, disk drive, or keyboard, must have a driver program. Many drivers, such as the keyboard driver, come with the operating system. For other devices, you may need to load a new driver when you connect the device to your computer.

A driver acts like a translator between the device and programs that use the device. Each device has its own set of specialized commands that only its driver knows. In contrast, most programs access devices by using generic commands. The driver, therefore, accepts generic commands from a program and then translates them into specialized commands for the device.

From `www.webopedia.com`

Package Options

`\usepackage[options]{color}`

One special option for the `color` package that is `monochrome`. If this option is selected the color commands are all disabled so that they do not generate errors, but do not generate color either. This is useful if previewing with a previewer that can not produce color.

Three other package options control the use of the **named** model. The `dvips` driver (by default) pre-defines 68 color names. The `dvips` option normally makes these names available in the **named** color model. If you do not want these names to be declared in this model (Saving TEX some memory) you may give the `nodvipsnames` option.

Package Options Conitnued

`\usepackage[options]{color}`

Conversely, if you are using another driver, you may wish to add these names to the named model for that driver (especially if you are processing a document originally produced on `dvips`).

In this case you could use the `dvipsnames` option. Lastly the `usenames` option makes all names in the **named** model directly available (more on this coming).

Defining Colors

The colors `black`, `white`, `red`, `green`, `blue`, `cyan`, `magenta`, `yellow` should be predefined, but should you wish to mix your own colors use the `\definecolor` command:

```
\definecolor{name}{model}{color specification}
```

This defines `name` as a color which can be used in later color commands. For example

```
\definecolor{light-blue}{rgb}{0.8, 0.85, 1}
\definecolor{mygrey}{gray}{0.75}
```

Now `light-blue` and `mygrey` may be used in addition to the predefined colors above.

Using predefined colors

The syntax for color changes is designed to mimic font changes. The basic syntax is:

```
\color{name}
```

This is a *declaration*, like `\bfseries` It changes the current color to `name` until the end of the current group or environment.

An alternative command syntax is to use a *command* form that takes the text to be colored as an *argument*. This is similar to the font commands such as `\textbf`:

```
\textcolor{name}{text}
```

So the above is essentially equivalent to:

```
{\color{name} text}
```

orange text

Using Color specifications directly

`\color{model}{specification}`
`\textcolor[model]{specification}{text}`

Normally one would predeclare all the colors used in a package, or in the document preamble, but sometimes it is convenient to directly use a color without naming it first. To achieve this `\color` (and all the other color commands) take an optional argument specifying the model. If this is used then the mandatory argument takes a `color specification` instead of a `name`. For example:
`\color[rgb]{1,0.2,0.3}`
would directly select that color.

This is particularly useful for accessing the **named** model:
`\color[named]{BrickRed}` selects the `dvips` color BrickRed.

Using Color Specifications Directly (Continued)

```
\color{model}{specification}
\textcolor[model]{specification}{text}
```

Rather than repeatedly use `[named]` you may use `\definecolor` to provide convenient aliases:
`\definecolor{myred}{named}{WildStrawberry}` ...
`\color{myred}` ...

Alternatively if you are happy to use the existing names from the **named** model, you may use the `usenames` package option, which effectively calls `\definecolor` on every color in the **named** model, thus allowing `\color{WildStrawberry}` in addition to `\color[named]{WildStrawbery}`.

Page Color

```
\pagecolor{name}
\pagecolor[model]{specification}
```

The background color of the whole page can be set using `\pagecolor`. This takes the same argument forms as `\color` but sets the background color for the current and all subsequent pages.

It is a global declaration, so you need to use `\pagecolor{white}` to get back to normal.

Box Backgrounds

Two commands similar to `\fbox` produce boxes with the backgrounds shaded an appropriate color.

```
\colorbox{name}{text}
\colorbox[model]{specification}text}
\fcolorbox{name1}{name2}{text}
\fcolorbox[model]{specification1}{specification2}{text}
```

The former produces a box colored with *name* like this. The latter is similar but puts a frame of color *name1* around the box colored *name2*. like this.

These commands use the `\fbox` parameters `\fboxrule` and `\fboxsep` to determine the thickness of the rule, and the size of the shaded area.

Acknowledgments

We thank the authors of, *"A Not So Short Guide to LaTeX"*, namely, Tobias Oetiker, Hubert Partl, Irene Hyna and Elisabeth Schlegl, for valuable contributions to this lecture material.

We thank the authors of, *"A Guide to LaTeX Fourth Edition"*, namely, Helmut Kopka and Patrick W. Daly for valuable contributions to this lecture material.

We thank D. P. Carlisle and The LaTeX3 Project for writing "*Packages in the 'graphics' bundle.*"