

Strategies for including graphics in L^AT_EX documents

Klaus H^oppner
Nieder-Ramst^adter Str. 47
64283 Darmstadt
Germany
klaus.hoepner@gmx.de

Abstract

This talk presents strategies for including graphics into L^AT_EX documents. It shows the usage of the standard graphics packages of L^AT_EX as well as an introduction to different graphics formats. Some external tools for converting graphics formats are discussed.

Overview of graphics formats

In general, there exist two kinds of graphics formats: vector and bitmap graphics. For bitmaps, there exist different flavors: no compression (which can make your files truly huge, dependent on resolution and color depth, so I won't cover them from here on), compression methods which completely preserve the image quality while reducing the data size, and "lossy" compression methods which cause a consequent reduction in image quality.

So let's go more into detail:

Vector graphics are set up by drawing or filling geometrical objects such as lines, B^ezier curves, polygons, circles and so on. The properties of these objects are stored mathematically. Vector graphics are in general device independent. It is easy to scale or rotate them without loss of quality, since the job of rasterizing them into actual pixels is done by the printer or printer driver.

Bitmaps without lossy compression store the image information as pixels, each pixel of a given color. In principle, the quality of a bitmap becomes better with increased resolution

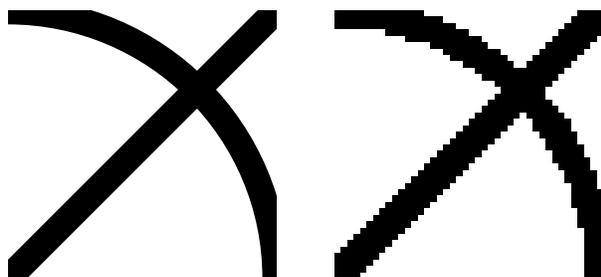


Figure 1: Zoomed view into a sample image as vector graphics (left) and bitmap (right).

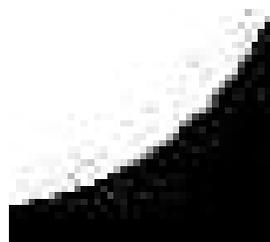


Figure 2: A low quality JPEG image showing some artifacts at the transition between black and white.

and color depth (e. g. GIF files use a color depth of 8 bits, leading to 256 different indexed colors while a bitmap with 24 bit color depth can have about 16 million colors). Scaling and rotating bitmap images will yield a loss of quality, and printing bitmaps to a device with a different resolution can produce bad results. Fig. 1 shows the difference between a scaled image as vector and bitmap graphics.

Bitmaps with lossy compression use the fact that the human eye is fairly good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. For this reason, components in the high frequency region can be reduced, leading to smaller file sizes. This works well for photographs that usually contain smooth transitions in color, but for graphics with a sharp border, artifacts can occur, as shown in fig. 2. The most prominent graphics format using lossy compression is JPEG.

Graphics formats in practice

There exist very many graphics formats, so I will concentrate on a few of those most often used:

EPS is the encapsulated PostScript format. It is mostly used for vector graphics but can also contain bitmaps.

PNG is the portable network graphics format. It was introduced due to the problem that Unisys claimed a patent for the compression algorithm used in GIF format. For this reason, it is often used nowadays on web pages. PNG is a bitmap format that supports compression both with and without loss of image quality.

JPEG is a bitmap format with lossy compression and is often used for photographs (e.g. most digital cameras produce JPEG files).

TIFF is a bitmap format sometimes used for high quality pictures—in part because it supports the CMYK color space important especially for commercial printing.

Now the question is: What format shall I use for what purpose? Though there is no one true answer to this question, my advice is as follows:

1. For drawings (e.g. technical drawings or data plots) use vector graphics. It gives you maximum freedom to manipulate the image when including it into a document where you often need to scale the image to fit into your layout. Additionally, it is independent of the output device, and thus you can zoom into the image in your document viewer without seeing single pixels.

Drawing tools offered by TÉX distributions— notably PSTricks and METAPOST— can usually produce EPS output natively. Most vector drawing programs like xFIG and Corel Draw also offer export functionality for producing EPS output (though sometimes buggy).

2. If you are stuck with bitmaps, use PNG for images with sharp color transitions, such as black and white boundaries.
3. For photographs, you can use JPEG in most cases, since the quality loss by compression is normally imperceptible when printed. On most devices, a resolution of 100 to 200 dpi will be sufficient (remember that screen resolution is normally about 75 to 100 dpi, and color printers claim to have high resolutions but dither color prints, so you will hardly notice the difference compared to JPEGs with higher resolution).

The LÁTÉX graphics package

Since the introduction of LÁTÉX 2 ϵ , the `graphics` bundle is part of the standard package set accompanying the LÁTÉX base distribution [1]. It consists of two style files, `graphics.sty` and `graphicx.sty`. While `graphics.sty` requires the use of `\scalebox` and `\rotatebox` for scaling or rotating graphics, the extended style `graphicx.sty` supports scaling and rotating using the `keyval` package, which pro-

vides a convenient interface for specifying parameters. In general, there is no reason not to always use `graphicx.sty`.

So the first step is to load the `graphicx` style file after the `\documentclass` statement:

```
\usepackage{graphicx}
```

In fact, the TÉX compiler doesn't know anything about graphics, and including them is done by the DVI driver. So the `graphicx` package has to do two things:

1. find the bounding box of the image (this can be troublesome when you have e.g. an EPS file created by an application that wrote a wrong `BoundingBox` comment—in this case, it can be helpful to put the `\includegraphics` command into an `\fbox` to find out what `graphicx` *thinks* about the bounding box);
2. produce the appropriate `\special` for the output driver; thus, the usage of the `graphics` bundle is driver dependent.

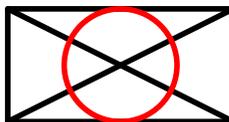
Nowadays, there are two main workflows for producing documents: using `latex` to produce a DVI file and then `dvips` for converting it to PostScript, and using `pdflatex` to produce a PDF file. Most modern TÉX systems are configured to automatically check whether you are using `latex` or `pdflatex` and producing `dvips \specials` in the first case and the appropriate `\pdfimage` commands in the second case. So if you are using one of the above workflows, you shouldn't need to specify your output backend explicitly. If you are using another backend you have to specify it as an option, e.g.

```
\usepackage[dvipsone]{graphicx}
```

(for the Y&Y `dvipsone` driver), but be aware that other backends often don't support scaling or rotating. For example, DVI previewers like `xdvi` or `windvi` try to interpret the `dvips` specials, but rotations may not be displayed properly in DVI preview.

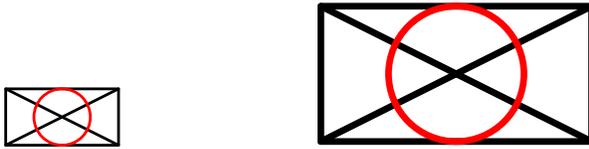
After the package is loaded, to include an image simply use:

```
\includegraphics{sample}
```



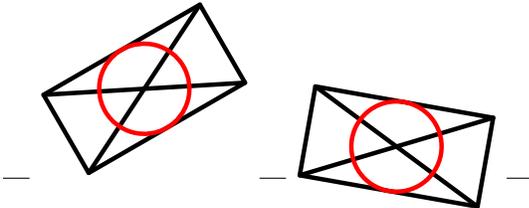
Please notice that no extension for the file was given. The explanation why will follow later. In the case of using `\includegraphics` without options the image is included at its natural size, as shown above. When using the `graphicx` style, you can scale your image by a factor:

```
\includegraphics[scale=0.5]{sample}
\includegraphics[scale=1.2]{sample}
```



Another option supports rotating an image:

```
\includegraphics[angle=30]{sample}
\includegraphics[angle=-10]{sample}
```



Positive numbers lead to counterclockwise rotation, negative numbers to clockwise rotation. The origin for the rotation is the lower left corner of the image, so in the clockwise rotation above the result has not only a height but also a depth below the baseline (as shown by the rules).

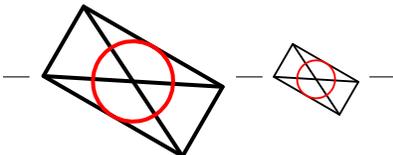
Images can not only be scaled by a given factor, you can specify a height and/or width for the resulting image instead:

```
\includegraphics[width=2cm]{sample}
\includegraphics[height=1.5cm]{sample}
```



`height` gives the height above the baseline. If your image has a depth, you can use `totalheight` instead, i.e. the sum of height and depth will be scaled to the given length.

```
\includegraphics[angle=-30,height=1cm]{sample}
\includegraphics[angle=-30,
totalheight=1cm]{sample}
```



You can specify both `width` and `height`. In this case your image may be scaled differently in horizontal and vertical direction, unless you use the `keepaspectratio` option:

```
\includegraphics[width=1.5cm,height=1.5cm]{sample}
\includegraphics[width=1.5cm,height=1.5cm,
keepaspectratio]{sample}
```

Source	Target	Tool
latex+dvips		
EPS		directly supported
PNG	EPS	ImageMagick/netpbm
JPEG	EPS	ImageMagick/netpbm
TIFF	EPS	ImageMagick/netpbm/tif2eps
pdflatex		
PDF		directly supported
EPS	PDF	epstopdf
PNG		directly supported
JPEG		directly supported
TIFF	PNG	ImageMagick/netpbm
TIFF	PDF	tif2eps+epstopdf

Table 1: Conversion of graphics formats supported by `latex+dvips` and `pdflatex`.



Please notice that usage of `angle` and `width` or `height` is sensitive to the order in which the options are given. Specifying the angle first means that your image is rotated first and then the rotated image is scaled to the desired width or height, while specifying a width or height first will first scale the natural image and rotate it afterwards.

Supported graphics formats

To make things a bit more complicated, `latex` with `dvips` and `pdflatex` support different graphics formats:

- `latex+dvips`: EPS
- `pdflatex`: PDF, PNG, JPEG, MPS

Table 1 shows ways to convert the standard graphics formats to supported formats. In particular, converting EPS graphics used with `latex+dvips` to PDF for `pdflatex` workflow is quite easy; just run the `epstopdf` Perl script, which uses Ghostscript to convert EPS to PDF.

This also explains why it is generally best to give the file names in `\includegraphics` commands without extensions. In this case the `graphics` package looks for a supported graphics format automatically. So if you have an image both as EPS and (e.g.) PDF, you can use both the `latex+dvips` and `pdflatex` workflows without changing your source.

One other useful special case: including the output of METAPOST is also easy; although it is technically an EPS file, it uses only a small set of com-

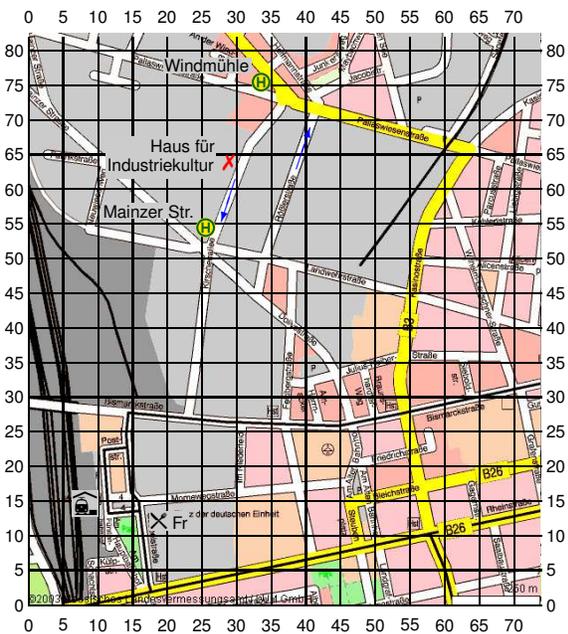


Figure 3: A map with additional marks produced with `overpic`

mands. So `pdflatex` can support the inclusion of METAPOST output directly. The only thing you have to do is to change the file extension of the output file to `.mps`.

Tools for image conversion

There exist several tools for conversion of graphics formats, both free and commercial. Besides free GUI-based tools like Gimp on Unix systems there are two command line tools available for Unix and Windows: ImageMagick [2] and netpbm [3].

ImageMagick can convert images directly, e.g. by typing

```
convert sample.gif sample.png
```

while netpbm uses the pnm format as intermediate format:

```
giftopnm sample.gif | pnmtopng -> sample.png
```

Another nice tool is `tif2eps` by Bogusław Jankowski et al. [4] which uses Ghostscript to convert a TIFF file to EPS, e.g.

```
gs -- tif2eps.ps sample.tif sample.esp -rh
```

which produces a RLE compressed and hex encoded EPSfile. In my experience EPS files produced with `tif2eps` are smaller than those produced by ImageMagick. Additionally it supports CMYK TIFF files smoothly.

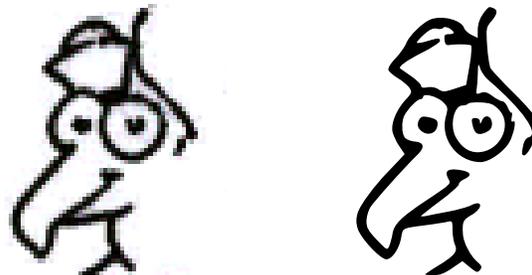


Figure 4: Zoomed view: bitmap (left) converted to vector graphics (right)

Additional tools

There are many other helpful tools. I will mention two I use quite often.

`overpic` is a \LaTeX package written by Rolf Niepraschk [5]. It includes an image into a \LaTeX picture environment, giving you the opportunity to add new elements into the image with normal \LaTeX picture commands. Fig. 3 shows a map overlaid with symbols and text at some points. The source code for this picture looks like

```
\usepackage[abs]{overpic}
...
\begin{document}
\begin{overpic}[grid,tics=5]{map}
\put(32,74){\includegraphics[scale=.3]{busstop.mps}}
\put(32,77){\llap{\scriptsize
\colorbox{back}{Windm\''uhle}}}
\put(28,63){\small\textcolor{red}{%
\ding{55}}}
...
\put(17.5,11){\scriptsize\colorbox{back}%
{\Pisymbol{ftsy}{65} Fr}}
\put(6.3,13){\colorbox{back}%
{\Pisymbol{ftsy}{68}}}
\put(29.8,61.4){\color{blue}\vector(-1,-3){2}}
\put(38.6,63){\color{blue}\vector(1,3){2}}
\end{overpic}
\end{document}
```

`potrace` is a tool to convert a pure black and white bitmap to vector graphics [6]. Fig. 4 shows a sample bitmap converted to a vector image.

References

- [1] CTAN:macros/latex/required/graphics
- [2] <http://www.imagemagick.org>
- [3] <http://netpbm.sourceforge.net>
- [4] CTAN:support/pstools/tif2eps
- [5] CTAN:macros/latex/contrib/overpic
- [6] <http://potrace.sourceforge.net>