The PracTEX Journal, 2010, No. 1
Article revision 2010/05/22

Tools for creating LaTeX-integrated graphics and animations under GNU/Linux

Francesc Suñol

Email francesc@fa.upc.edu
Website http://dfa.upc.es/personals/francesc/
Address Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract This paper describes how to easily create graphics and animations that can be included in LaTeX documents. This article discusses three kinds of figures: plots, schematics, and pictures. The tools presented here can quickly generate plots, and are based on simple gnuplot and bash scripts that display the final result on the screen. Ipe is an excellent program to deal with complex figures and schematics, and the animate package is used to make a series of figures change over time to simulate a movie. All the programs used in this article are free software.

1 Introduction

It's quite common to see documents with figures that do not preserve a consistent style. Graphics that contain words or symbols must match the text in the journal or document in order to give a pleasant appearance. TeX was not originally designed for graphical work, but fortunately today we have many tools that can generate high quality figures. These generated figures can be integrated into our documents in a style consistent with the surrounding text. To give a few examples, PSTricks, PGF/Ti*k*Z, and METAPOSTare extraordinary tools that do this job well. Unfortunately, if one needs to plot a large number of points, these tools can run out of memory. In a similar way, if one needs to draw a complex figure with these tools, the task can become difficult. Other approaches are based on the removal of figure labels (nicely described by J. Levine [1]), directly hacking the `ps` or `eps` files, and rewriting the labels with the help of LaTeX packages like `psfrag` or `overpic`. I think these methods may be appropriate in some cases, but often this process can be slow and tedious.

In this article, the method proposed to automate this process is based on the use of simple bash scripts. All the tools described here are free software, and have been tested under Debian GNU/Linux. They should work in other Linux distributions as well. If you are using Windows or Mac you can try similar methods, but I think they may be more difficult to set up.

When writing scientific papers, one has to deal with at least three types of figures: plots, schematics, and pictures. Sometimes it is useful to make these figures appear to change over time (animations), for example in slideshows, or simply for teaching purposes. In the next sections I will present the tools I have developed for creating these *static* and *dynamic* plots, sketches, and pictures.

2 Static figures

In this section I explain step-by-step how to generate graphics files that do not change over time. I reserve the name "dynamic graphics" for those that simulate videos, that is, movies or animations, and these will be described in Section 3.

The objective of the current section is to create figures both in `eps` and `pdf` formats (in order to run `latex` or `pdflatex`) that can be included in documents using the `graphicx` package in the usual way: including the line

```
\usepackage{graphicx}
```

in the preamble, and

```
\begin{figure}[!h]
\centering
\includegraphics[options]{your-figure}
\caption{A caption.}
\label{your-label}
\end{figure}
```

in the document.

2.1 Plots

The best way I have found to quickly create plots is using gnuplot. Gnuplot is a very powerful program that is versatile and portable, and allows you to visualize

mathematical functions or data. I am not going to give a detailed description of how to use this program; if you are interested in the commands and screen terminals available please see the gnuplot manual by T. Williams et. al. [2].

One problem that often occurs is that after spending some time customizing your plot, when you see it on the screen the conversion to LaTeX (using `latex` or `pslatex` terminals) gives results that can be disappointing: the plot legends do not have the correct spacing, the font sizes are not appropriate, and labels are often put in the wrong place or even outside the plot. How do you deal with these problems?

I think the easiest solution is to have a file with some gnuplot commands in it, and then run a bash script in the command prompt that displays the resulting final plot on the screen. With this method you can modify the gnuplot file, run the script, and see the modifications in the plot instantaneously. If the result is not as desired, just correct the gnuplot file and rerun the script.

At its most basic level, the script should do the following:

1. *gnuplot file.gp* (redirecting the output to `pslatex terminal`).
2. *latex file.tex* (compilation of the source)
3. *dvips file.dvi* (conversion to `eps`)
4. *epstool file.eps* (creation of the bounding box)
5. *epstopdf file.eps* (conversion to `pdf`, so you can run `pdflatex`)
6. *rm auxiliary-files* (to remove auxiliary files except the original gnuplot file and the recently created `file.eps` and `file.pdf`)
7. *xpdf file.pdf* (to display the result in the screen)

I wrote a small bash script[1] (`gp2epspdf.bash`) that performs this task. You can download this script from the site where this article appears, or from my website. The script requires `latex`, `dvips`, `gnuplot`, `epstool`, `epstopdf` and `xpdf`. Assuming that you have a working LaTeX installation, you'll have `latex`, `dvips`, and probably `epstopdf` and `xpdf`. If you don't have these programs, you can install them in Debian (or any other Debian-based distribution) just by typing

```
apt-get install gnuplot epstool epstopdf xpdf
```

1. I called the script `gp2epspdf`, because it converts a gnuplot file (I usually add the extension `gp` to gnuplot files), to both an `eps` file and a `pdf` file.

at the command prompt.

Let's look at an example. Using the text editor of your choice create a file (with name `plot.gp`, for example[2]) containing the following text:

plot.gp

```
set size 0.75,0.65
set key 6.3,27 Left reverse samplen 1 spacing 1.2
set border 31 lw 0.5
set xlabel '$t$ (s)'
set ylabel '$y$ (m)'
set xrange [-2:6]
set yrange [-10:30]
plot -x**2+20 w p ps 1.3 pt 1 title '$y(t)=-2t^2+20$'
```

Once the file is created (and it has read permissions), just type

```
./gp2epspdf.bash plot.gp
```

in the bash prompt, and instantaneously a new window will open with Figure 1 in it. Quick and easy!

In order to see all the line styles and point types available, create a file with the word `test` in it (let's call this file `all.gp`),

all.gp

```
test
```

and then type

```
./gp2epspdf.bash all.gp
```

A new window will appear with the available options. In addition to these options, in gnuplot version 4.2 or later you can define your own line colors with the linecolor option.

If you find you are using this `gp2epspdf` script often, you can create an alias in your `.bashrc` file. The script is very simple, and you can easily modify it to fit your needs.

2. The name and the extension of the file are not relevant.

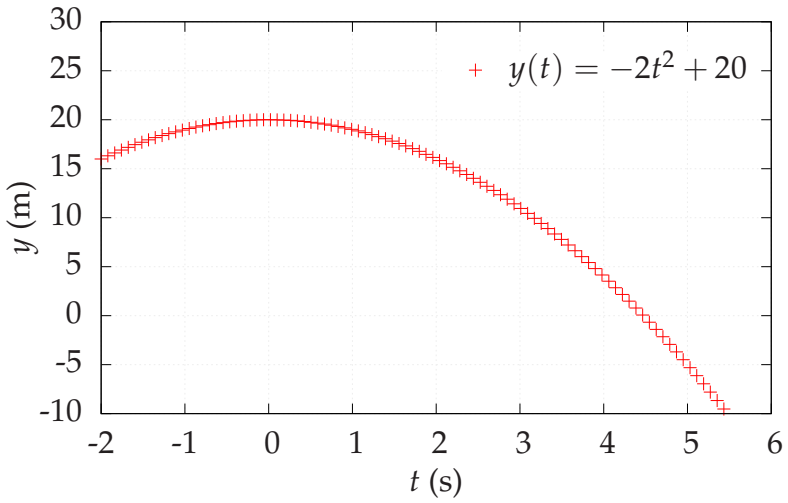

Figure 1: Example of a static plot.

2.2 Sketches

The creation of schematics or sketches usually involves two main approaches: on one hand you have the option to write the figure drawing commands using METAPOST, PSTricks, PGF/Ti*k*Z or others. On the other hand, you can have a visual idea of how the sketch should look, and draw it directly to paper. Personally, I am the kind of person who prefers the second option for complex figures, and I prefer to use the program Ipe in this case.

Ipe [3] is an easy-to-use drawing editor with a friendly graphical user interface. It contains a variety of basic geometry primitives like lines, splines, polygons, circles... and has snapping and grouping options, scaling and rotating features, and many more advantages. The figures can be saved in `pdf`, `eps`, or `xml` formats, and text entry is done by using LaTeX source code, which makes it easy to enter mathematical expressions. Another important point is that Ipe can accept so-called *Ipelets*, which are user-created plugins that extend the functionality of the program. O. Cheong [4] and J. Hlaveck [5] wrote intuitive manuals for learning and using Ipe.

An example of a sketch created by Ipe is shown in Figure 2.

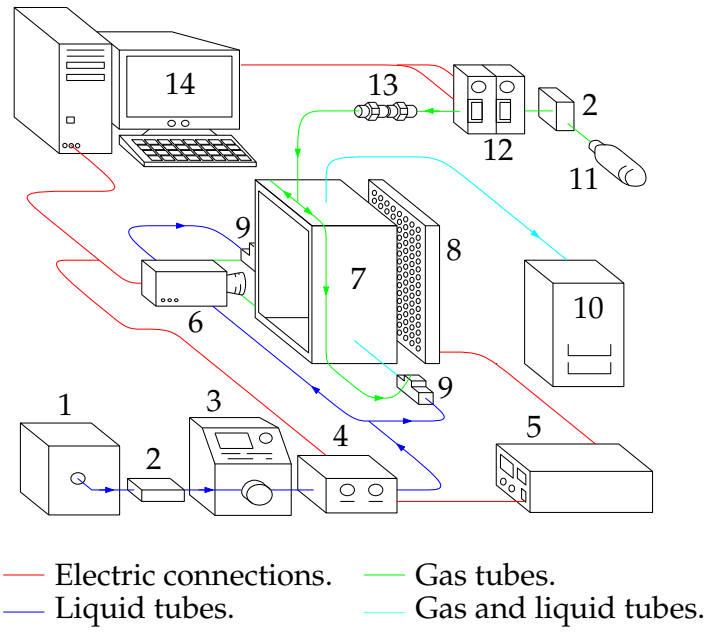

Figure 2: Example of a figure created with Ipe.

2.3 Pictures

Pictures are commonly used in documents. The one point I want to make here is that there are several tools in GNU/Linux to edit pictures. The most popular program is The Gimp, which has a wide range of capabilities. Another image editing software that fits my purposes perfectly is Imagemagick [6]. Imagemagick is a collection of programs to edit images, convert between file formats (more than one hundred formats supported), and similar functions. It is typically run from the command line. For example, if you have a `bmp` image and you need a `pdf`, just type

```
convert file.bmp file.pdf
```

To include labels (or whatever you want) in the pictures, you can import the figure to Ipe, place your labels, and save the file. Once again...Quick and easy!

An example of a picture containing labels is shown in Figure 3.

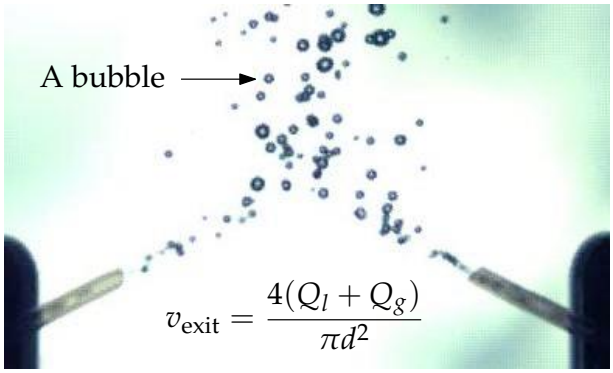

Figure 3: Example of a picture with some labels in it.

3 Dynamic figures: animations

Here I describe how to generate animations in LaTeX (plots, sketches, and movies) using the `animate` package [7]. It's important to note that these animations currently only work with Acrobat Reader 6 or later, and javascript should be enabled. All of the figures created in this section can be included in documents using the `animate` package by including the line

```
\usepackage{animate}
```

in the preamble, and

```
\begin{figure}[!h]
\centering
\animategraphics[options]{fps}{basename}{first}{last}
\caption{A caption.}
\label{your-label}
\end{figure}
```

in the document. The final `pdf` can be viewed in Adobe Reader on all supported platforms.

3.1 Plots

In order to make a plot animation, I created a bash script (`plot-animation.bash`[3]) that uses a procedure similar to the one shown in Section 2.1, inside a while loop. Mainly, it does the following:

1. *while* $i \leq N$, *plot function* $+\Delta i$ (using the same procedure as the script `gp2epspdf.bash`, but without displaying each individual plot in the screen).
2. *pdftk created-files.pdf cat output animation.pdf* (Merge all the resulting `pdf` files into one single file with `pdftk`).
3. *rm auxiliary-files* (Remove all the created `pdf` files except the one created by `pdftk`).
4. *xpdf animation.pdf* (Display the final result in the screen).

If you don't have `pdftk`, in Debian you can install it by typing

```
apt-get install pdftk
```

This script will create a single `pdf` with as many pages as the number of frames, and doesn't need an input gnuplot file; you have to edit the script in order to define what are you planning to plot. Of course, you can do the same with the script `gp2epspdf.bash`, if you don't want to have a gnuplot input file. This script is very simple, and once more you can easily modify its contents in order to fit your needs, or just to improve it.

To create the animation, type the following line in the command prompt:

```
./plot-animation.bash
```

An example of an animation created with this script is presented in Figure 4.

If you need to decrease the size of the animation, you can separate the moving parts (data points and/or functions) from the non-moving parts (axis, labels, tics, grid, ...) and use the `timeline` option of the `animate` package.

3. The script `plot-animation.bash` is available along with this article or from my website.

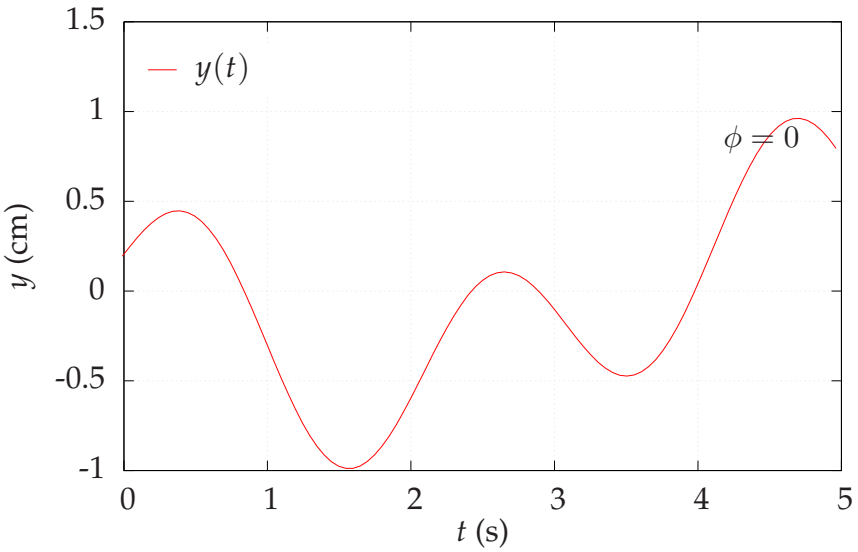

Figure 4: Example of a moving plot (only works with Acrobat Reader 6 or later, and javascript should be enabled!).

3.2 Sketches

Often it's useful to see a series of sketches that appear to change over time. Once again, Ipe can help us to do this job. The animation shown in Figure 5 has been created using Ipe as follows:

- In the first page, I draw all the lines and labels that do not change their position in time.
- In the next page, I draw the pendulum in its first position.
- In the next page, I draw the pendulum in its second position: rotated a little bit with respect to the first position.
- ...
- In the following pages I draw the ball, the spring and the moving labels.

As you can see, this process is completely manual. It works for simple animations, but not for large ones. Currently I am searching for some kind of automation, but until now I haven't found any better way to do this. Fortunately, Ipe is easy to use and it's not difficult to create large animations.

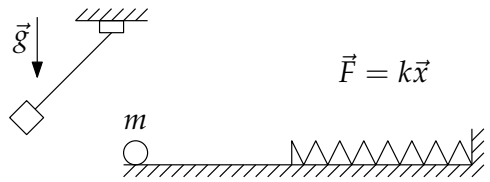

Figure 5: Example of a time-evolving sketch (only works with Acrobat Reader 6 or later, and javascript should be enabled!).

3.3 Movies

Including a movie in a `pdf` file can be done in several ways (e.g. using the `movie15` or the `easymovie` packages), but I prefer to use the `animate` package for two reasons:

1. The movie is embedded in the `pdf` file, so everything is contained in a single file.
2. Adobe Reader plays the animation, without calling any external programs. This is important to make your document truly portable across platforms.

First of all we need to split all the images of the movie file. In the case where we have an `avi` movie, and we want the frames in `jpg` format, this can be done by typing the command

```
mplayer movie.avi -vo jpeg
```

Normal cameras usually have a recording speed of around 30 frames per second, so if you have a movie of one minute duration, this command will create approximately 1800 images. Let's rename the images as `basename1.jpg`, `basename2.jpg`, ... and `basename1800.jpg`. Now, to merge the images in the new animation, include the following lines in your document

```
\begin{figure}[!h]
\centering
\animategraphics[options]{15}{basename}{1}{1800}
\caption{A movie caption.}
\label{movie-label}
\end{figure}
```

and the work is done. In the preceding example I set the fps to 15. The reason for this is that 15 frames per second is the average speed that the human eye can register images, and it's not necessary to force our computer to do more work than needed. The example presented would create a two-minute duration animation, since we are playing at 15 fps. If we want a one-minute animation, we can play at 30 fps. A better solution is to skip all the odd-numbered frames which significantly reduces the size of the resulting `pdf`.

An example of a movie, generated with the steps explained above (using 17 frames inside a loop, playing at 14 fps), is presented in Figure 6.

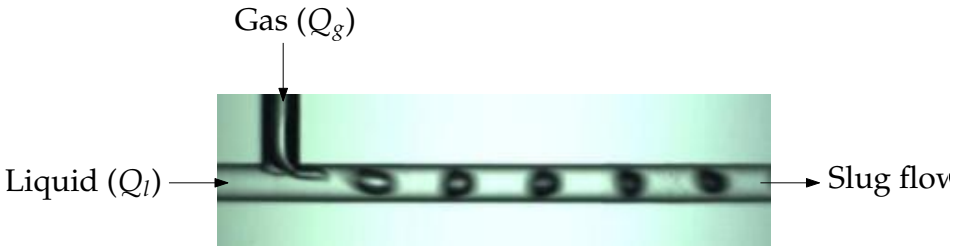

Figure 6: Example of a movie (only works with Acrobat Reader 6 or later, and javascript should be enabled!).

4 Summary

We have learned about some useful tools for generating graphics and animations in GNU/Linux. Three types of graphics have been discussed: plots, sketches, and pictures, both static and changing over time.

The LaTeX user community is continually growing, and along with this growth the facilities and useful packages to make high quality graphics are increasing as well. As a result, you now have to decide which of the many available tools are the best for your purposes — it's your choice.

References

[1] Jenny Levine. *Label replacement in graphics*. The PracTEX Journal 2005-1.
http://tug.org/pracjoun/2005-1/levine

[2] Thomas Williams et. al. *Gnuplot. An interactive plotting program* (2007).
http://www.gnuplot.info/docs/gnuplot.pdf

[3] Ipe home page.
http://tclab.kaist.ac.kr/ipe/

[4] Otfried Cheong. *The Ipe manual* (2007).
http://www.cosy.sbg.ac.at/~held/teaching/wiss_arbeiten/Ipe/Ipe_manual.pdf

[5] Jan Hlavacek. *Ipe — a graphics editor for LaTeX*. The PracTEX Journal 2006-2.
http://www.tug.org/pracjourn/2006-2/hlavacek/

[6] Imagemagick home page.
http://www.imagemagick.org

[7] Alexander Grahn. *The animate package* (2008).
www.ctan.org/tex-archive/macros/latex/contrib/animate/doc/animate.pdf