Introduction to Visualization

(and presentation tools)

There are a variety of visualization packages suitable for numerical astrophysics applications, ranging from simple 2-D plotters to professional products that support animation and multi-dimensional exploration. This tutorial will concentrate on `sm`, one of the 2-D varieties, and will touch on a few others. There will also be some brief discussion regarding animations and ray tracing. Above all note that visualization is about graphically conveying numerical information to viewers. As a result, there is a certain artistic element involved that can be hard to master.

A brief word about presentation tools, such as LaTeX, `xfig`, and PowerPoint, is given at the end.

Visualization

sm

`sm` is a proprietary product for which the astronomy department has a license. If it's not already in your search path, add `/local/bin/` to get it. The advantage of `sm` is that you can quickly plot straightforward data yet at the same time you can make very complex scripts to exploit the full potential of the graphing package. The entire manual is available online.[1]

There are 3 ways `sm` is typically used: 1) interactively; 2) semi-interactively with user-defined macros; 3) semi- or non-interactively with user-built scripts. Method 1 is the fastest for simple data that you don't plan to look at again. Method 3 is preferred for complex graphing applications. Method 2 will be discussed only in passing.

Interactive sm

To use `sm` interactively, simply type `sm`. You will be asked to enter a command. Try the following (hit `RETURN` after each line):

```
dev x11
box
relocate 0 0
draw 1 1
```

If all went well an X window popped up after the `device` (shorthand `dev`) command, axes with values ranging from 0 to 1 appeared, and a diagonal line from the lower left corner to the upper right was drawn.

Now in another window create a file called `sq.dat` that looks like this:

```
1 1
2 4
3 9
4 16
5 25
6 36
7 51
8 64
9 81
10 100
```

`sm` interprets whitespace as column delimiters, much like `awk`. In `sm` type:

```
data sq.dat
read {n 1 s 2}
erase
```

[1] `http://www.astro.princeton.edu/~rhl/sm/sm.html`

```
limits n s
box
points n s
```

You should see half a parabola worth of "x"s. The `read` command loaded two vectors, in this example we called them `n` and `s`, with the corresponding columns of the data file. The `limits` command set the plot limits automatically based on the values in `n` and `s`. The `box` command drew the axes (use `xlabel` and `ylabel` to make axis labels) and `points` drew the points. If you don't like the point size, try `expand 2` to make everything double size (you'll need to redraw the points). You can also experiment with the `ptype` command (try `ptype 3 0` for example). *Hint:* If your terminal settings are set correctly, you can use the up and down arrow keys to cycle through the command history. Now try the following:

```
set n2 = n*n
connect n n2
```

The `set` command allows you to make a new vector or modify an existing one. The `connect` command connects points. You should notice the point for N=7 lies off the curve (yes, that was deliberate). You may also notice the curve is a bit jagged (particularly if you replot the curve without doing `points` first). Let's use 100 points instead:

```
erase
box
set i = 1,100
set n2 = i*i
connect i n2
points n s
```

(If you're using `expand 2` you'll notice the axis labels and ticks got bigger). Note how we were able to create the vector `i`. Now try this:

```
ctype red
histogram n s
```

Assuming you're in front of a colour screen you should now have steps drawn in red going up the curve. If you prefer your histograms to have bars that go all the way down, use `barhist 100 n s` instead (the optional numeric value specifies the percentage width of the bars). To go back to drawing in the default colour, type `ctype default` (this is preferable to `ctype white` because the latter won't work very well on white paper, whereas the former automatically knows to switch to a black foreground).

For more help on a particular function, simply type `help` *`function`*. You may also find the `apropos` command useful. And there's always the manual. Some of the most popular commands, in addition to what we've seen so far, are: `define`, `do`, `draw`, `errorbar`, `if`, `ltype`, `lweight`, `macro`, `putlabel`, `relocate`, `sort`, `ticksize`, `window`, and `while`. Also try `help arithmetic` to see what math operators and functions are supported, and `list device` to see what devices are available (e.g. to output everything to an encapsulated postscript file, use `dev postencap myfile.eps` instead of `dev x11`).

Scripting sm

The commands in the example above could all be placed in a file, say `sq.sm`, and read in like this: `sm inp_new sq.sm`. This is useful if you plan to look at data of this nature over and over. It's particularly useful if you first want to look at the output on the screen and then later print to a file. Note however there were a few `erase`s above, and these cause the plots to flash by. Here's a way to handle such cases:

```
define ps 0 # use 0 for screen output, 1 for file output
macro pause {
    define dum ('c') # default is to continue
    while {1} { # keep looping until a valid response is given
        define dum ? {Type c to continue or q to quit}
        if ('$dum' == 'q') {quit}
        if ('$dum' == 'c') {return}
    }
}
macro next {
    if ($ps) {
        hardcopy
    } else { # NOTE: the else MUST be on the same line as } and {
        pause
        erase
    }
}
#
# execution begins here...
if ($ps) {
    dev postencap myfile.eps
} else {
    dev x11
}
```

Here we've used the `define` command to create scalars (as opposed to vectors). Note that the value of a scalar can be obtained by prepending a dollar sign: `$ps`. If the lines above are added at the start of the script, you could replace all `erase` calls with our user-defined macro `next`. The user would then be prompted to press `RETURN` before the next plot is drawn. Notice that if `ps` is defined then a `hardcopy` command is issued instead, which sends a print request to a file (or printer). Alternatively you could use the `window` command to put more than one plot on the screen at once, but there's a limit to how many plots you can cram in this way.

Commonly used macros can be stored in a file by themselves and read in with the command `macro read` *`macrofile`* before doing anything else. This is how method 2 of using `sm` works.

`sm` can be about as frustrating as `(t)csh` at times as far as remembering the correct syntax, perhaps more so. Trial and error is your best bet. As a general rule of thumb, an expression used as an argument to a `define` command should be in ( ) and any string value should be quoted `''` just to be safe.

Other Graphing Packages

The following table lists some other common Unix graphing packages with their most useful features.

Package	**Features**
`PGPLOT/pgperl`	callable plotting libraries for C or FORTRAN
`gnuplot`	interactive function plotting; 3-D wire & surface plots
IDL	"Interactive Data Language"—heavyweight analysis & plotting package
`geomview`	interactive geometry viewer; real-time rotations
NCAR Graphics	scientific visualization with emphasis on supercomputing

Consult man or web pages for more information on these utilties.

Animation

Animation under Unix remains an awkward process; PCs (running Windows) & Macs are much better at this. Still, it can be done, even without buying expensive software. Generally the easiest way is to create a series of *frames* that you want to string together to form a movie. These frames could be produced by a drawing package (like `POV-Ray`, described below) and/or a plotting package (for example, in `sm` you could output to device `blackgif` to create a GIF image of a plot). The frames should be numbered consecutively and left-padded with zeroes, e.g. `frame001.gif`, `frame002.gif`, etc. for easy manipulation. The file format is largely dictated by the movie-making package you choose. For example, `mpeg_encode` accepts a handful of formats and has a provision to convert from an unsupported format to a supported one. Conversely, `ppm2fli` supports only PPM format ("portable pixmap") and `gifsicle` supports only GIFs. If you need to keep the frames as well as the movie, the best strategy is to use compressed formats for the frames. Otherwise, use PPM for maximum flexibility. The `Netpbm` package contains *lots* of graphics conversion and processing utilities. These are available on the department machines (e.g. `man pbm pgm ppm pnm` to get listings of most of the utilities). Also check out:

```
http://netpbm.sourceforge.net/
```

for the latest source. Another popular conversion tool is `ImageMagick`:

```
http://www.imagemagick.org/
```

(in particular, check out the `convert` command, which comes standard these days in many Linux implementations.) You can use `xv` or `display` to view graphics files in a window.

`mpeg_encode`

The `mpeg_encode` utility allows you to take individual frames and make them into an MPEG movie, a popular though lossy animation format. You could then use `mpeg_play` (or any variety of other MPEG viewers) to watch the movie. The player is available on the department Suns. The encoder can be found at:

```
http://bmrc.berkeley.edu/frame/research/mpeg/
```

Also try out `mencoder` and `mplayer`, a more modern MPEG encoder and player set, available here:

```
http://www.mplayerhq.hu/homepage/design7/news.html
```

`ppm2fli`

This utility makes a FLI animation that can be viewed with `xanim` (available on the department machines). FLI is not nearly as compressed as MPEG, but the playback quality is far superior. Only PPM format frames are supported. The encoder is available at:

```
http://vento.pi.tu-berlin.de/fli.html
```

`gifsicle`

The `gifsicle` utility creates an animated GIF from individual GIF frames. This is one way to make an embedded animation on a webpage. It's available on the department Linux machines, or from:

```
http://www.lcdf.org/ eddietwo/gifsicle/
```

GIF anims are more compressed than FLIs, but not as much as MPEGs.

Other encoders

There may be other free encoders out there—surf the web to find them. Otherwise there are a variety of packages available for purchase that can make very fancy animations with titles, fades, etc., but most are designed for Windows/Mac environments.

Also, I have designed my own suite of animation scripts that handles a variety of formats and supports simple titles, fades, and so on.

Ray Tracers

To make really high quality animations (or stills) that consist of more than dots or line drawings, you need a ray tracer. A ray tracer allows you to construct a world seen from a specific viewpoint, using complicated geometrical shapes and carefully arranged light sources. A very easy-to-use public domain package is the Persistence of Vision Ray Tracer, `POV-Ray`:

    http://www.povray.org/

(be sure to check out the Hall of Fame pictures!) `POV-Ray` accepts text-format descriptions of worlds that can be quickly transformed into high-quality images.

Presentation Tools

Although not visualization methods *per se*, presentation tools allow you to convey scientific concepts more easily by incorporating graphics and equations into text.

LaTeX

LaTeX (or, more specifically, LaTeX 2_ε) is a powerful text formatting language. Much like code, a LaTeX document contains text commands that are compiled into the finished product (typically a postscript file). The language is best illustrated by example:

```
% this is a comment
\documentclass[12pt]{article}
\begin{document}

\section{Introduction}

Latex (or \LaTeX) is a powerful text formatting language.
You can write in \textit{italics} or \textbf{bold face}.  You can
\underline{underline} and $\overline{\mbox{overline}}$.
Best of all, you can do equations:
\begin{equation}
  E = mc^2
\end{equation}
or arrays of equations:
\begin{eqnarray}
  \dot{x} & = & z \\
  \dot{z} & = & x + z
\end{eqnarray}
Even integrals!
\begin{equation}
  M = \int_x \int_y \int_z \rho(x,y,z)\, dz\,dy\,dx
\end{equation}

\subsection{Here's a subsection}

Here's a table:

\begin{table}[h]
\begin{tabular}{l|c|r}
Left & Center & Right \\
\hline
1 & 2 & 3 \\
4 & 5 & 6
\end{tabular}
\end{table}

Here's how you start a new paragraph (just leave a blank line).
Notice how the line is indented.

\subsubsection*{Here's a subsubsection without a number}

Here's how you might include a figure:

\begin{figure}
  \resizebox{\textwidth}{!}{\includegraphics{myfigure.eps}}
  \caption{Caption goes here.}
\end{figure}

(you'll need a \usepackage{graphics} command in the preamble).

\subsubsection*{And here's a numbered list:}
\begin{enumerate}
\item Item 1.
\item Item 2.
\item etc.
\end{enumerate}

\end{document}
```

As you can see, LaTeX contains powerful equation typesetting tools and supports both embedded tables and figures. Naturally, the document you are currently reading was written in LaTeX (and converted to HTML for the course website using the `latex2html` utility).

To compile a LaTeX document, just type `latex mydocument.tex`, where `mydocument.tex` is the name of your document. This creates a `dvi` file which can be viewed in a window with `xdvi` or converted to postscript with `dvips` (use `ghostview` or `gv` to view a postscript document in a window). If you want PDF output, use the `convert` utility: `convert mydocument.ps mydocument.pdf` (use `acroread` to view a PDF document in a window).

A nice LaTeX tutorial is available here:

`http://www.ctan.org/tex-archive/info/lshort/english/lshort.pdf?action=/starter/`

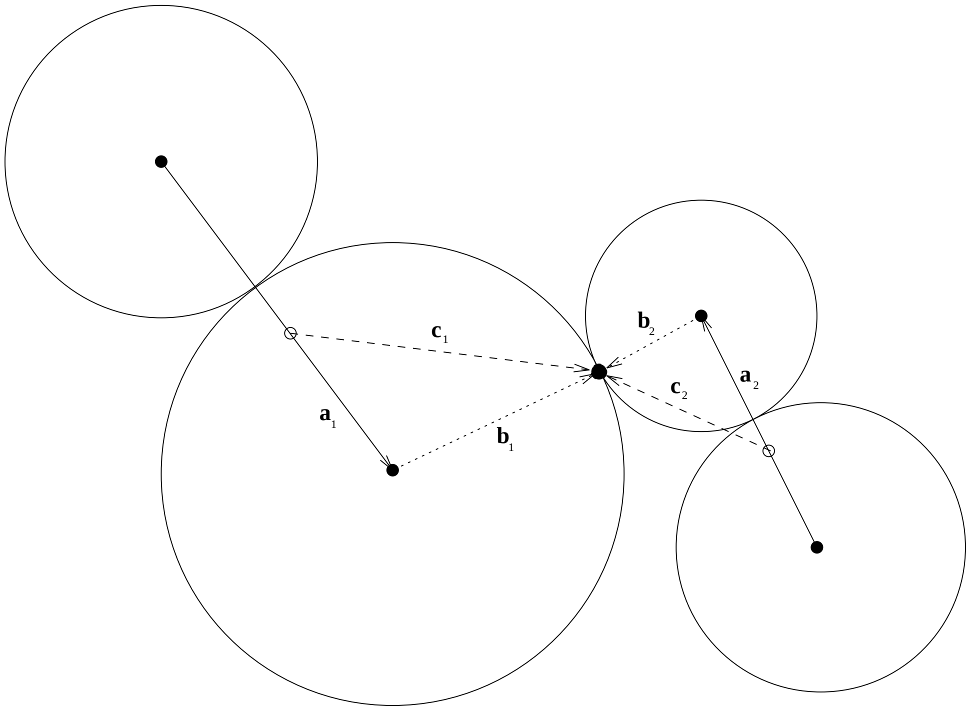
There are also extensive lists of special symbols that you might find useful:

`http://www.ctan.org/tex-archive/info/symbols/comprehensive/symbols-a4.pdf`

(these documents are all part of the Comprehensive TeX Archive Network at `http://www.ctan.org/`).

xfig

`xfig` is a free drawing tool that allows the user to produce precise schematics. It can also be used to overlay text and symbols on image files. A variety of output formats are supported. Here is a sample figure produced with `xfig`:



For more information on `xfig`, surf here:

`http://www.xfig.org/`

PowerPoint

PowerPoint is software for creating and displaying presentations, typically used in conjunction with a video projector. It is the standard these days at scientific conferences (you might also see OpenOffice or PowerBook display tools). Examples of both PowerPoint and OpenOffice presentations are available on the course web page.

One disadvantage of these tools is that it can often be hard to typeset equations (not that you should have too many equations in a presentation anyway...). An easy workaround is to generate the equations in LaTeX first, convert the output to a rendered image, then import it into the presentation.

Another Word About HTML

A web page (HTML) is another good way of presenting your work, and has the advantage that it can be viewed by anyone with a web browser. Refer to the UNIX tutorial for more information about HTML.