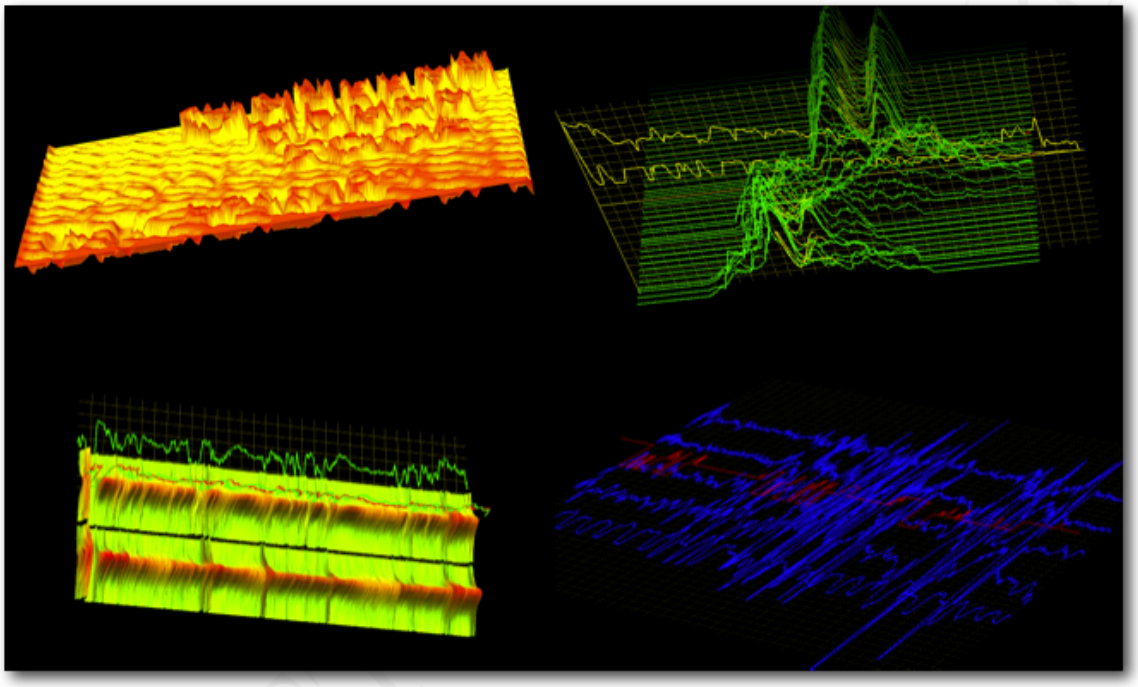
A UNIX tool for graphical rendering of time-series

Mark Renfrew Titchener

Space Odyssey: SpOd



SpOd short for Space Odyssey, is a simple to use interactive 3D visualisation tool whose inspiration in part is the graphical imagery in the classic movie by the same name, and based on the book by Arthur C. Clarke. A reason for including SpOd and associated tools in with the documentation here on *PERTECS* is because it in fact is the core graphical display suite that is used in *PERTECS* .

Written in C, SpOd mostly exploits the capabilities of OpenGL and GLUT [**?**][**?**] which gives it a high level of portability. In its latest incarnation we have ported these tools to the Raspberry pi. The tools have been developed very much with UNIX (OS-X), shell, and tools paradigm [**?**] in mind. The UNIX shell encourages creative use of "tools" rather than a reliance on self-contained "applications". Commands typically invoke combinations of filters and pipes that rely on the use of standard I/O and redirection.

SpOd evolved in the context of reviewing medically derived time series but is equally suited for time-series generally. An early emphasis was on using Deterministic Information Theory methods [**?**] to quantify and compute patient state from EEG (electro-encephologram)/EOG (electro-

oculargram) time series. A number of the tools are devoted to computing information/complexity series from time series, results that are more often required to be interpretted visually. SpOd was developed to simplify visualisation. A typical SpOd data file may contain one or more graphical object prescriptions; static or dynamic graphs, surfaces, with optional position and text markers. One of SpOd 's strengths is its ability to handle points, graphs, surfaces in a form that allows for interactive dynamical visual interpretation.

SpOd accepts data as standard input (or as a file) comprising lists and arrays of numbers. Data for each graphical object is preceded by prescriptive headers and display directives. Optional modifiers further drive the rendering of these prescriptions. A virtual 3D volume may be explored interactively, including roll and spin actions, lighting adjustments, and animated or dynamical scanning of trace sequences (polygraph traces), or graphs. SpOd is capable of rendering well over $10,000$ individual graphs sequentially in a matter of seconds, or to instantly switch between data sets, viewing combinations and so on.

While optomised primarily to achieve high efficiency and bandwidth requirements, other benefits have accrued from using simple text for the input data interface. The syntax is by and large humanly readable for good reason. This has turned out to be beneficial in handling both computation and visualization. Data processing operations are often arrived through a sequence of steps of stages. By rendering the intermediate results in a format suitable for visual exploration, the development of novel signal processing algorithms may be readily visualised step by step. Taking this further, we have explored forms of in-file documentation and notation, including embedding shell procedures to give recursive execution and visualisation of all processing steps. Thus graphical files may, with appropriate formatting, facilitate edited, execution, processed, as well as visualisation.

SpOd incorporates several basic filter functions for extracting and/or translate graphical objects. This paradigm is well suited to the research setting, where often analysis and the development of novel data reduction, control, or other processing algorithms is achieved through experimental iteration of trials.

SpOd also supports real-time data streaming options which have been the precursor to *PERTECS* i/o formats. Several gui tools have been created to facilitate the creation of headers for streaming data series to SpOd , but are now obsolete by the functionality supported by *PERTECS* .

The SpOd executable is generally copied to the `/usr/local/bin directory`. This it may be accessed from the terminal command line. Typical forms of invocation are:

`cat filename | spod` for the display of a file
or alternatively

```
spod filename
spod -f filename
cat filename | spod -s -rate 100 # plays file containing columnated data,
at a rate (Hz)
spod -l graphno -f filename | spod # extracts nth (=graphno) from file and
displays
```

In the last of the above examples SpOd is used first to extract a specific numbered graph from a set of graphs and then the prescription is passed to a second instantiation of SpOd for display. As with any UNIX process more than one instance of SpOd may be running at any time, allowing multiple and simultaneous displays of graphical objects in separate windows.

The following pages provide help to get one started.

SpOd simultaneously supports a range of basic graphing formats, that include surfaces, static

graphs, animated traces and graph sequences. It simultaneously accepts up to 10 surfaces and some 400,000 graphs/traces. I have successfully loaded over a Gbyte of input data corresponding to polygraph traces with a total of 150,000,000 data points.... and surfaces with 2,000,000 vertices. Surfaces and graphs may be displayed individually or in combination or as animated graph sequences. By selecting appropriate combinations of surfaces, static, dynamic or animated graphs and traces and effects one may creative powerful interactive visualisation environments. The real power of SpOd derives from the modern graphical power of OpenGL and the support hardware that is standard in modern PC's. All SpOd does is simply translate lists of data into a form required by OpenGL, to give you the power of the graphical environment.

Basic help is available by typing ?, when an active SpOd session is running. The help menu is returned via std-error to the terminal window.

The graphical object types supported and respective prescriptive formats are illustrated by example. Graphical objects fall in to basic categories, including, i) simple 2D graphs, ii) partially specified 3D graphs, iii) fully specified 3D graphs, iv) simple 3D surfaces, and v) fully specified 3D surfaces) The graphs of i), ii), and iii) may be prescribed as static displays or dynamic traces (very useful for looking at long data series).

What is meant here by static and dynamic is clarified in the examples. A static graph is a plot of points y_i versus x_i say, but displayed within a 3D viewing space. The space may be rotated/swung dynamically about the origin for viewing convenience. A dynamic animated trace or a set of points y_i versus x_i say, displayed within a viewing window that can be moved flexibly across the x-domain (letting x represent time for example). Whereas plotting a series of 500 points fits comfortably into a single display frame, a 1,000,000 point trace will not. In these situations it is useful to allow the graph to extend beyond the viewing space and then flexibly move the viewing window to the left or right along the trace. This may be done either manually, under mouse control or flexibly as a timed animation or replay, or both. Different animation modes are available depending on the kind of object being used and the desired effect.

0.1 Syntax

Subsequent sections elaborate object instantiation and syntax. Here we summarize the encoding of object properties. A new Graphical object begines with some sort of type prescription

we have 3D surfaces, 3D and 2D graphs.

The syntax is perhaps more easily explained in terms of specific examples.

0.2 Example 1.1 Simple 2D static graphs specify the y- (or alternatively z-) coordinate points only, the x-coordinates are implicit.

For many graphs, a simple list of data points is all that one needs. The points are plotted at uniform increments along the x-axis, for example where the x-coordinate represents time. The list of data points forms a time series $[y_i]$.

Simple graphs are plotted either in the y-x or z-x plane. The x-coordinate is generated from the array index for the time series.

Graphical Object Letter Objects/Specifiers/Qualifiers	
Specifier	Description
s	object: partial surface mesh
S	object: full surface mesh
k	object: kurve 3D
x	object: static graph 3D
y	object: static graph 2D in yx plane
z	object: static graph 2D in zx plane
K	object: animation Kurve sequence 3D
X	object: animation graph sequence 3D
Y	object: animated trace 2D in yx plane
Z	object: animated trace 2D in zx plane
o	specifier: object visibility
b	qualifier: ballmarker e.g. b:r:redball or b:b:blueball
c	qualifier: to be completed colour specifier
O	specifier: Orientation e.g. O 316 9
L	specifier: Light position e.g. L 288 1
C	specifier: surface Colour gradient
I	specifier: Index animation rate and time initiialisation
V	qualifier: View Screen managment e.g. V:f:full screen
v	qualifier: object Visible e.g. v:n:on or v:f:off
w	qualifier: line width e.g. w 0.5
W	specifier: Window position and dimensions xorig yorig xdimen ydimen
p	qualifier: points (line type) e.g. p:.:points or p:-:line
G	specifier: grid brightness
B	specifier: grid bounds
E	specifier: grid origin
M	qualifier: Mode of animation
P	qualifier: Plane Partition parameters for kurves
D	delta-T timed replay
a	qualifier: simultrace set or reset t:n:on or t:f:off
A	qualifier: axis intensity
l	qualifier: label display l:[n]/[f]:label the label is discarded
t	qualifier: text e.g. t:c:"my title":size:x y z
T	qualifier: window Title e.g. T "my title"
#	escape: comment line
Q	escape: quit object descriptors
r	raw data switch r:f:raw_off or r:n:raw_data

Table 1: The syntax is described in the text and with examples

A Graph's header has a generic form. The first letter 'y' (or 'z') of the header indicates that this is a simple static graph. 'y' indicates the graph is to be plotted in the yx plane, whereas 'z' indicates the zx plane is to be used. A further single letter indicates the graph's colour; r=red, g=green, b=blue, m=magenta, c=cyan, y=yellow, w = white, ¡sp¿= black, ('.'=invisible). The last entry on the first line of the header gives the graph label. A graph label can not include white space. For example "mygraph" or "mygraph" are acceptable labels, but "my graph" is not. These prescriptive items are each separated by a colon.

With a simple 2D graph most of the hard work of arranging the graph for view is taken care of for you. The header values must be set to place the graph in the viewing space The header further indicates the size of the data list (e.g. 10001 indicates the list is a 1000×1 points array). Next the x- y-, z- scale parameters are given. Since the x-coordinates are implicit, an x-scale factor of 9 indicates the domain of the graph will be scaled to fit symmetrically in the interval $[-9, +9]$. Similarly the y-scale parameter takes the values in the range $[-1, 1]$ and scales these to the range $[-4, +4]$.

A Comment line is ignored by SpOd , begins with the # character.

For a simple graph in the y-x plane

```
1 #example graph in y-x plane, colour green, note that a comment line begins with the # character.
2 y:g:mygraph: 10000:1: 9:4:0: 0:0:0
3 0.010000 0.019999 0.029996 0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 etc
```

Lets generate an example directly with a shell script. Here I am using awk to run short interpreted bits of C-code. (Regrettably you may not be able to simply cut and paste the following to your terminal window, for it seems that the characters, in particular the single and double quotes, are translated by the web page environment. in principle you can copy the following into a file and execute as a UNIX shell script.)

```
1 #!/bin/sh
2 #example graph in y-x plane, generated by shell script
3 echo "y:g:mygraph: 1000:1: 9:4:0: 0:0:0" | awk '{ printf "%s\n", $0; split($0,an,":");
4 for (i=1; i <=an[4]; i++) { printf "%f ",rand();}
5 }' | spod
```

0.2.1 Example 1-2 Partially specified 3D graphs lists y- and z-coordinate points, the x-coordinates are implicit.

```
1 #example graph in y-x plane, colour green
2 y:g:mygraph: 100:1: 9:4:0: 0:0:0
3 0.010000 0.019999 0.029996 0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 ...
4 0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 0.109778 0.119712 0.129634 ...
```

0.2.2 Example 1-3 Fully specified 3D graph list all x-, y-, z- coordinate points.

```
1 #example graph in y-x plane, colour green
2 y:g:mygraph: 100:1: 9:4:0: 0:0:0
3 0.010000 0.019999 0.029996 0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 ...
4 0.010000 0.019999 0.029996 0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 ...
5 0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 0.109778 0.119712 0.129634 ...
```

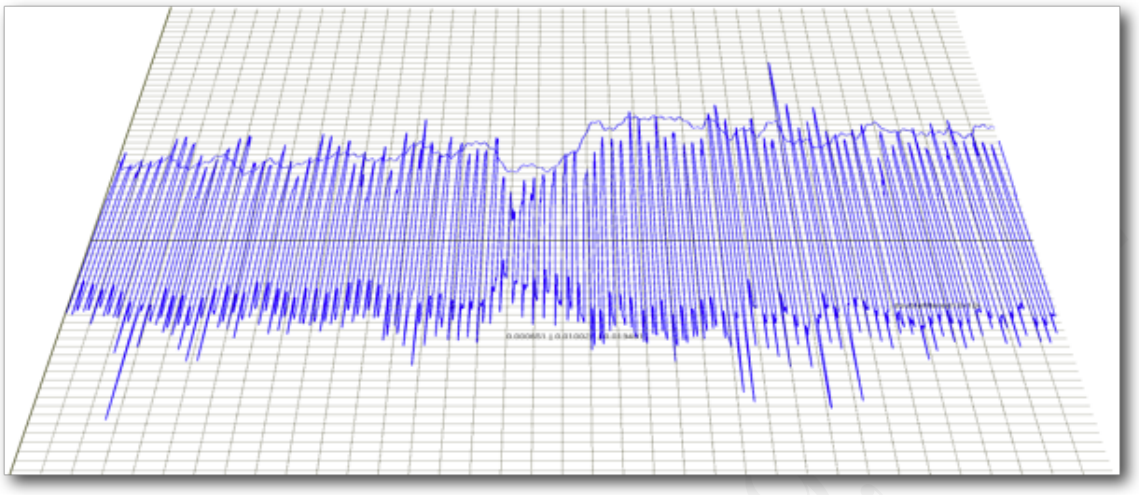


Figure 1:

0.2.3 Example 2-1. Simple surfaces, assumes a regular mesh

```
#example graph in y-x plane, colour green
y:g:mygraph: 100:200: 9:4:0: 0:0:0
0.010000 0.019999 0.029996 0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 ...
0.010000 0.019999 0.029996 0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 ...
0.039989 0.049979 0.059964 0.069943 0.079915 0.089879 0.099833 0.109778 0.119712 0.129634 ...
.
.
.
```

Example 2-2. fully specified surfaces

```
#try this script
awk 'BEGIN{printf "S:surface: 41:21: 4:4:4: 0:0:0\n"
for (i=-10; i <=10; i++) {
    for (x=-20; x <=20; x++) printf "%f ", sin(x/20 * 3.14159/2);
    printf "\n";
}
for (i=-10; i <=10; i++) {
    for (x=-20; x <=20; x++) { radius = sqrt(1 - sin(x/20 * 3.14159/2)*sin(x/20 * 3.14159/2));
         printf "%f ",radius * sin(i/10 * 3.14159);}
    printf "\n";
}
for (i=-10; i <=10; i++) {
   for (x=-20; x <=20; x++) {
       radius = sqrt(1 - sin(x/20 * 3.14159/2)*sin(x/20 * 3.14159/2));
       y= radius * sin(i/10 * 3.14159);
       printf "%f ",sqrt (radius*radius - y*y);
    }
    printf "\n";
}

}' | spod
.
.
.
```

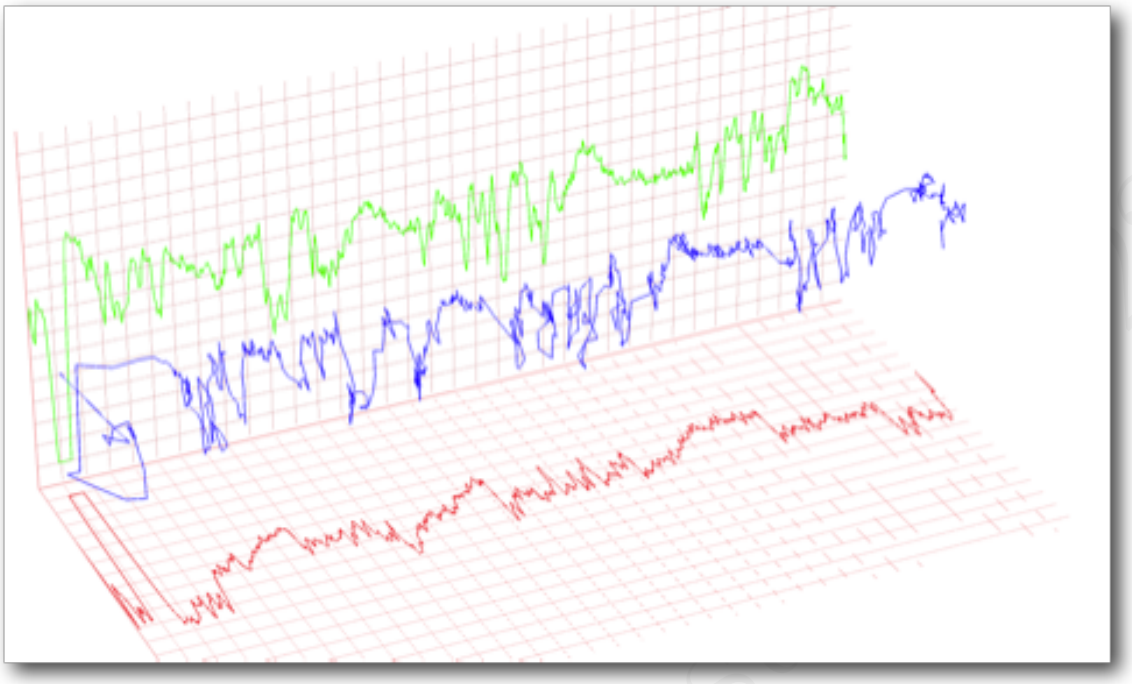


Figure 2:

The following illustrations have been generated using SpOd .

The next are a mixture of graphical results illustrating some of the possible visualisation effects that can be produced using SpOd .

0.3 Some examples

SpOd provides an elementary set of features. The visualisation effects that are possible using the available set of options is largely open to the imagination of the user. By providing some examples here we may stimulate you to think about how to best present information that is visually appealing and effective in communication to others the patterns and structures in your data.

We begin with simple graph drawing examples. Most of these use simple UNIX shell scripts to generate the data on the fly. If you want to understand the data format then simply save the data to a file or display it on the screen instead of piping to SpOd .

Now these UNIX scripts are sensitive to syntax. The scripts that appear here ought to be able to be copied and pasted to your terminal window but be aware that some browsers interfere with the simple act of copying or pasting even text data. You have been warned...

By providing you with the script you have the opportunity to dissect the code to see exactly what is being created

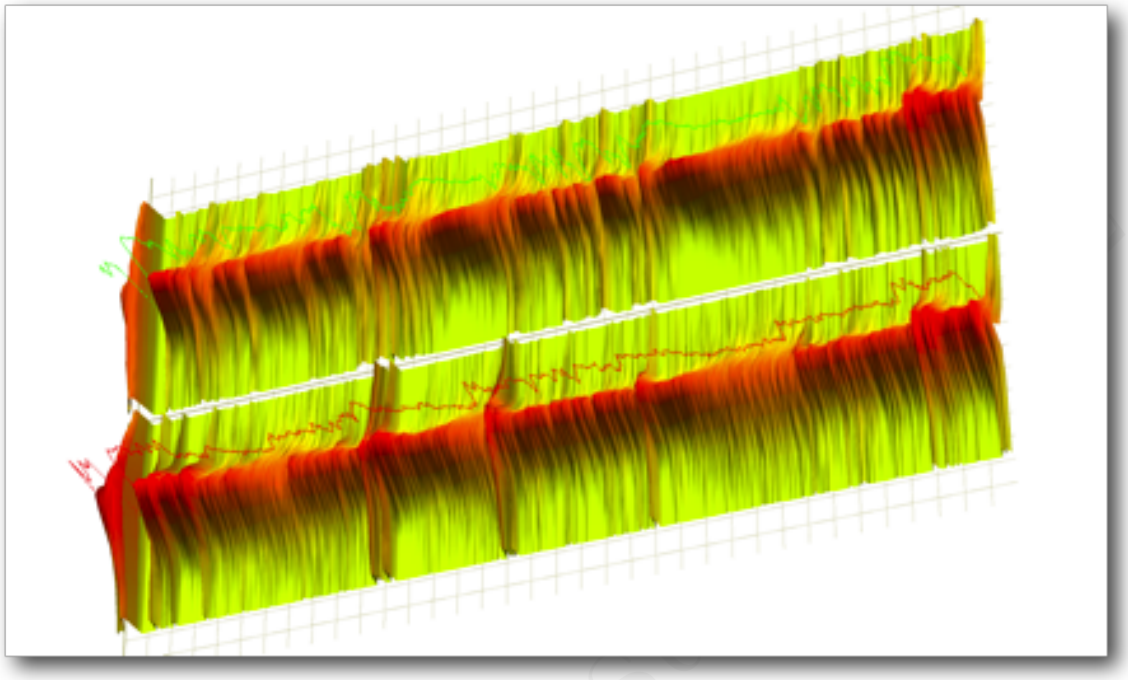


Figure 3: The entropy surfaces produced from an EEG and EOG time series respectively, from a recorded polysomnography study.

0.3.1 Example 1.1: A simple graph

```
1 echo "y:g:mygraph 1000 1 9 4 0 0 0 0" | awk '{
2 printf "%s\n", $0;
3 split($0,an," ");
4 for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
5 }' | spod
```

This script generates a 1000 random values in the range of [0,1] and graphs them as a static graph in the y-x plane, which is the default viewing plane.

The left mouse button can be used to rotate the viewing angle.

If you leave out the "| spod " the script will display on the terminal the text that is being used to create the graphical object.

```
1 y:g:mygraph 1000 1 9 4 0 0 0 0
2 0.000008 0.131538 0.755605 0.458650 0.532767 ...
```

For an explanation of the format have a look at the papers:

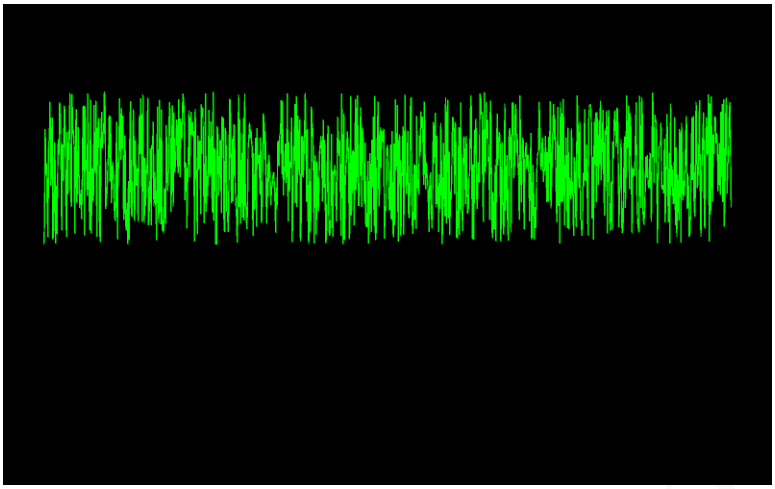


Figure 4:

0.3.2 Example 1.2: now lets offset the graph wrt the x-axis, activate the display reference grid.

Note that the graph is also displaced in the z direction slightly to bring the graph in front of the grid using the z-offset 0.1. You may rotate the image to see this side on.

```
1 echo "G 0.7 0.0 0.5 9.0 5.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0
2 y:g:mygraph 1000 1 9 4 0 0 -2 .1" | awk '{cnt++;
3 printf "%s\n", $0;
4 if (cnt ==2) {
5 split($0,an," ");
6 for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
7 }
8 }' | spod
```

0.3.3 Example 1.3: ten times the data !

```
1  echo "G 0.7 0.0 0.5 9.0 5.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0
2 y:g:mygraph 10000 1 9 4 0 0 -2 .2" | awk '{cnt++;
3 printf "%s\n", $0;
4 if (cnt==2) {
5     split($0,an," ");
6     for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
7 }
8 }' | spod
```

Its clear that not all the points can be displayed at a time without the display looking distinctly crowded. We have several options. One is to display the points only, by adding the graphics qualifyier "p:.:points". p indicates that it is a plotting qualifier, the period indicates plot mode with points, and this is followed by a mandatory text label string. The colon characters are delimiting characters in the qualifier.

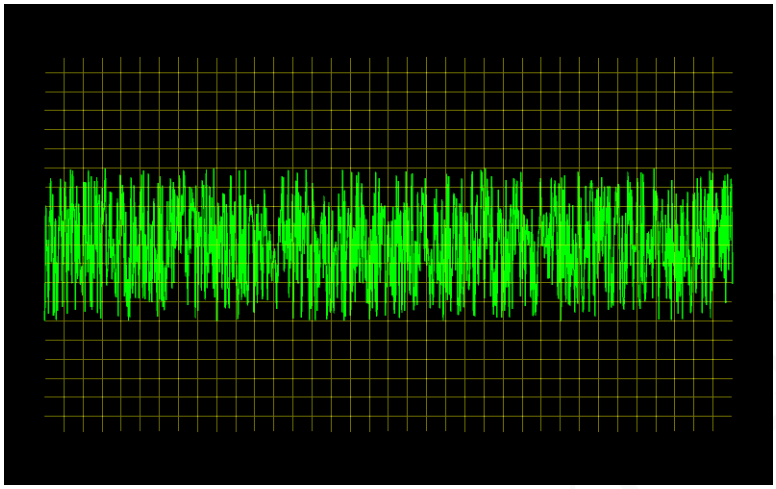


Figure 5: The grid brightness may be controlled as on an oscilloscope through a combination of ways. The "G" settings which determine the initial "load" state are explained elsewhere in the documentation. Interactively, the shift-1 and shift-2 keys increase and decrease the brightness respectively, and also in conjunction with the shift-mouse-left button. (The grid in the z-x plane is similarly controlled by shift-3 (brighten) and shift-4 (darken) keys. and shift-5 toggles the major axes on/off.)

```
echo "G 0.7 0.0 0.5  9.0 5.0 5.0  0.0 0.0 0.0  1.0 1.0 0.0
y:g:mygraph 10000 1 9 4 0 0 -2 .2" | awk '{cnt++;
printf "%s\n", $0;
if (cnt==2) {
    split($0,an," ");
    for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
    printf "\np:.:points\n";
}
}' | spod
```

An alternative is to display a window onto the data, to create what is in effect a dynamic trace or polygraph style display. This is achieved by capitalising the Y in the 'Y:g:mygraph' object specifier. Use the right-mouse button to scroll the display horizontally. The left-mouse button will rotate the graph view.

```
echo "G 0.7 0.0 0.5  9.0 5.0 5.0  0.0 0.0 0.0  1.0 1.0 0.0
Y:g:mygraph 100000 1 9 4 0 0 -2 .2" | awk '{cnt++;
printf "%s\n", $0;
if (cnt==2) {
    split($0,an," ");
    for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
}
}' | spod
```

It is perfectly feasible to have upwards of 10,000,000 data points in a graph using this format and to be able to scroll through all the data in a jiffy.

One may also animate the display. Try typing the 'g' key, (even repeatedly). 'G' will reverse the animation effect. Try typing 'h', or even 'H'.

```
\subsection{Example 1.4:  Multiple graphs and traces}
```

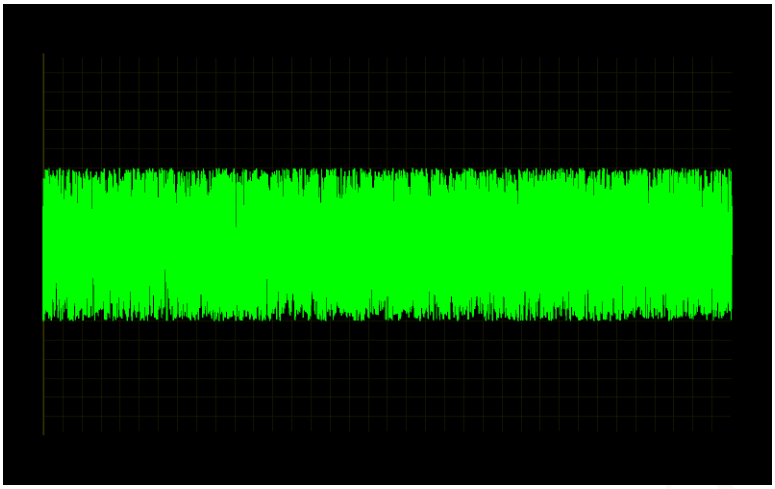


Figure 6:

```
echo "G 0.7 0.0 0.5  9.0 5.0 5.0  0.0 0.0 0.0  0 0 10
C 1 1 0 1 1 1 1
I -1.0 0.000004 0.017500 0.0 -1.0
L 12 61 148 210 176 340
M 0 0 1 10
" | awk '{cnt++; print $0
if (cnt==5) {
   numbpoints=1000
   printf "\ny:g:graph1 %d 1 9 1 0 0 2 .2\n", numbpoints;
   for (i=1; i <=numbpoints; i++) { printf "%f ",sin(i*2*3.1415926/100)}
   printf "\nv:n:visible_on";
   printf "\nb:r:redball";

   printf "\nY:r:trace2 %d 1 9 1 0 0 -2 .2\n", 250*numbpoints;
   for (i=1; i <=250*numbpoints; i++) { printf "%f ",cos(i*2*3.1415926/100)}
   printf "\nv:n:visible_on";
   printf "\nb:g:redball";

}
}' | spod
```

The 'l' (lowercase 'ell') key will toggle the display of the graph labels and the index counters. The index counters run in the range of $[0, 1]$ and indicate the data positions with respect to the object arrays for the left-edge, centre-point, right-edge of the window respectively. When there are a few graphs only the number keys '1', '2', '3', etc may be used to toggle the graph for display. The position of the label corresponds to the number key of the graph by the same label.

The ball markers are invoked by object qualifiers of the form for example 'b:r:redball'. The third field is a mandatory label, but can be any sort of string label desired.

You may wish to experiment with the above script to work out which line gives rise to which display feature or features. If one wishes to capture a particular view, or set of display modes, then the environment settings may be output to the terminal window using the shift-O (output) key.

In principle the environment variables may be included with a set of graphics objects to preset a given view. This is discussed in more detail later.

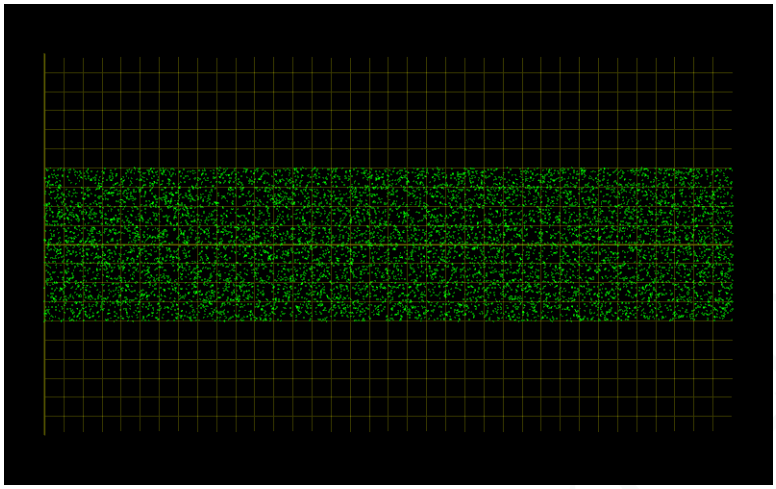


Figure 7:

0.4 Graphing and plotting in 2D & 3D

In reality SpOd implicitly provides a 3D visualisation experience even when you are only using 2 of the available dimensions. Placing multiple graphs in a 3D space can offer advantages if you are trying to relate events in one set against those in another. Part of the 3D experience is also about animation. Being able to 'play' a graph sequence over a range of speeds can help to high light events in ways that might otherwise allow them to be missed. The eye has a way of inferring structure where the effects in the data stream are in fact subtle, or masked by noise or other processes.

This section provides examples where the graphing exploits the 3D space. In examples-1 we used only the y-x plane to graph in 2D. Changing the y (or Y) to a z (or Z) in the graphics object header translates the 2D graphing plane to the z-x plane. (note also the use of the z-scale and z-offset parameters in the object header.) The z-direction is by default into the screen, but a simple rotation of the space using the mouse will expose the plane.

0.4.1 Example 2.1: pair of graphs in 3D

```
1 echo "O 206 316
2 L 35 258 148 210 180 333
3 G 1 0.6 0.6  9.0 5.0 5.0  0.0 0.0 0.0  .3 0 0
4 C 1 1 0 1 1 1 1
5 M 0 0 1 10
6 I -1 0.00010 0.017500 0.050 0.000000
7 " | awk '{cnt++; print $0
8 if (cnt==5) {
9    numbpoints=1000
10   printf "\ny:b:sine %d 1 9 2 0 0 0 0\n", numbpoints;
11   for (i=1; i <=numbpoints; i++) { printf "%f ",sin(i*2*3.1415926/100)}
12   printf "\nv:n:visible_on";
13   printf "\nb:r:redball\nw 1.3";
14
15   printf "\nz:m:cosine %d 1 9 0 2 0 0 0\n", numbpoints;
16   for (i=1; i <=numbpoints; i++) { printf "%f ",cos(i*2*3.1415926/100)}
```

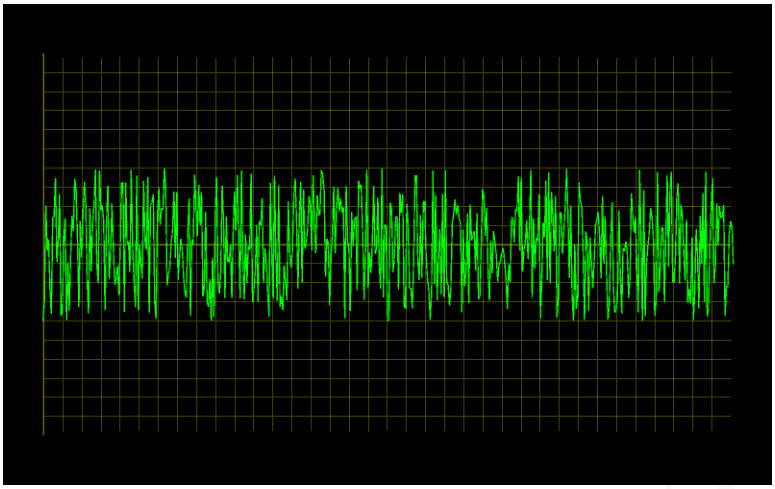


Figure 8:

```
17    printf "\nv:n:visible_on";
18    printf "\nb:g:redball\nw 1.3";
19
20 }
21 }' | spod
```

Hit the 'j' (jump) key to jump to the start of the animation sequence.

The 'j' and 'J' keys provide support for jumping to preset positions or views within a given data set. For more information on this see the extended list of instructions on the graphic object specification.

Use the space bar to halt the animation. 'g' or 'G' are also effective in controlling the scan speed of the animation. 't' or 'T' may be used to control the turning speed, and 'b' allows both the animated trace to be combined with panning. Hitting 'b' more than once also initiates tumbling of the graphics objects.

You may need to experiment with the combination of these keys to understand their full effect, and the order in which they may be combined in order to obtain desired effects. Of course you may simply resort to using the left-mouse button to rotate the objects manually.

The adaptation of the above script results in a pair of animated traces in 3D. Here the 'k' is exchanged for a 'K', and the number of points used to render the curves is increased to 100,000.

```
1 echo "O 206 316
2 L 35 258 148 210 180 333
3 G 1 0.6 0.6  9.0 5.0 5.0  0.0 0.0 0.0  .3 0 0
4 C 1 1 0 1 1 1 1
5 M 0 0 1 10
6 I -1 0.000010 0.017500 0.050 0.000000
7 " | awk '{cnt++; print $0
8 if (cnt==5) {
9    numbpoints=100000
10   printf "\nY:b:sine %d 1 9 2 0 0 0 0\n", numbpoints;
11   for (i=1; i <=numbpoints; i++) { printf "%f ",sin(i*2*3.1415926/100)}
12   printf "\nv:n:visible_on";
13   printf "\nb:r:redball\nw 1.3";
```

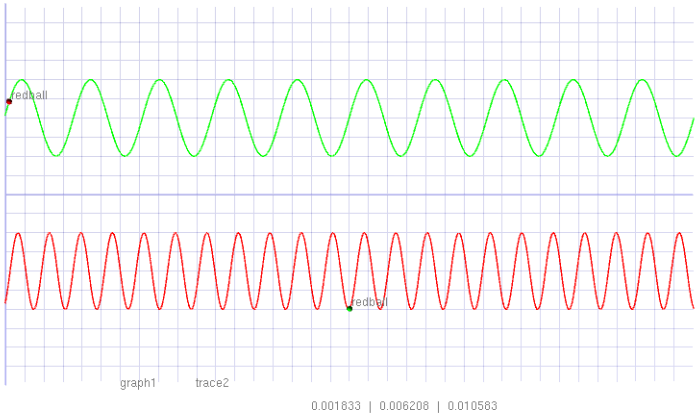


Figure 9:

```
14
15    printf "\nZ:m:cosine %d 1 9 0 2 0 0 0\n", numbpoints;
16    for (i=1; i <=numbpoints; i++) { printf "%f ",cos(i*2*3.1415926/100)}
17    printf "\nv:n:visible_on";
18    printf "\nb:g:redball\nw 1.3";
19
20 }
21 }' | spod
```

0.4.2 Example 2.2: A 3D time trajectory

Using the 'k' (or 'K') instead of 'y','Y' or 'z','Z' graphical specifiers allows one to plot the y versus z against time as a 3D kurve, either as a static or dynamic trace (depending on whether the lower case or upper case specifier is used).

```
1 echo "O 206 316
2 L 35 258 148 210 180 333
3 G 1 0.6 0.6  9.0 5.0 5.0  0.0 0.0 0.0  0 0 10
4 C 1 1 0 1 1 1 1
5 M 0 0 1 10
6 I -1 0.001 0.000001 0.05 0
7 " | awk '{cnt++; print $0
8 if (cnt==5) {
9    numbpoints=1000
10   printf "\nk:b:sin_versus_cos %d 1 9 2 2 0 0 0\n", numbpoints;
11   for (i=1; i <=numbpoints; i++) { printf "%f ",sin(i*2*3.1415926/100)}
12   for (i=1; i <=numbpoints; i++) { printf "%f ",cos(i*2*3.1415926/100)}
13        printf "\nv:n:visible_on";
14   printf "\nb:r:redball\nw 1.3";
15 }
16 }' | spod
```

Try also the following script:

```
1 echo "O 206 316
```

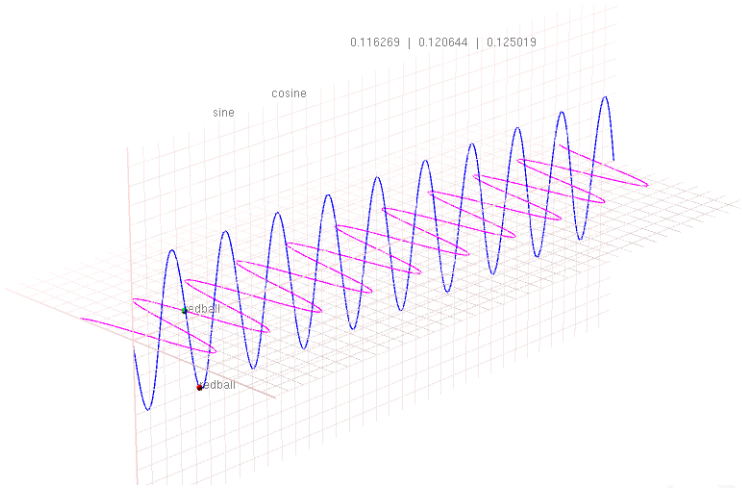


Figure 10:

```
2 L 35 258 148 210 180 333
3 G 1 0.6 0.6  9.0 5.0 5.0  0.0 0.0 0.0  0 0 10
4 C 1 1 0 1 1 1 1
5 M 0 0 1 10
6 I -1 0.000010 0.132500 0.050 0.0
7 " | awk '{cnt++; print $0
8 if (cnt==5) {
9    points=100000
10   printf "\nK:m:slinky %d 1 9 .2 .2 0 0 0\n", points;
11   for (i=1; i <=points; i++) {printf "%f ",exp(i/10000) * sin(i*2*3.1415926/100) + 10*cos(i
         *2*3.1415926/1000)}
12   for (i=1; i <=points; i++) {printf "%f ",exp(i/10000) * cos(i*2*3.1415926/100) + 10*sin(i
         *2*3.1415926/1000)}
13        printf "\nv:n:visible_on";
14   printf "\nb:r:redball";
15 }
16 }' | spod
```

One may achieve useful, even quite spectacular graphical effects using trajectory style plots.

0.4.3 Example 2.3: Graphs that are not a function of time!

Up till now all of our example graphs have assumed data points that are implicitly a function of time (where time is mapped to the x-axis). The assumption is that the data points are to be mapped uniformly along the x-axis which is fine for a large number of applications. But this may not always the case. It may be that one would like to explicitly plot data comprised of fully specified (x,y,z) coordinate values. This is done using the 'x' graphics object specifier. ('X' is reserved for an animation function dealt with in due course)

In the case of the fully specified graphs the list of x-coordinates are followed by the list of y-coordinates and then finally the z-coordinates. Some care may need to be taken to ensure sensible scaling into the fixed viewing space.

```
1 echo "G 0.7 0.5 0.5  9.0 5.0 5.0  0.0 0.0 0.0  1.0 1.0 0.0
```

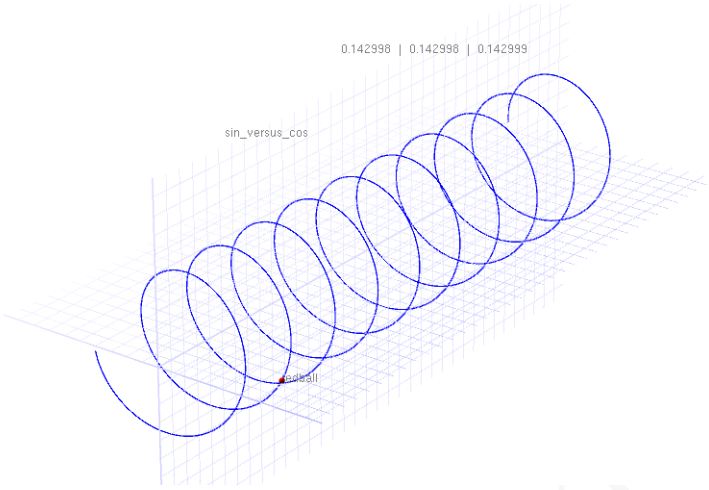


Figure 11:

```
 2  x:g:mygraph 100 1 8 8 8 -4 -4 -4" | awk '{cnt++;
 3 printf "%s\n", $0;
 4 if (cnt==2) {
 5     split($0,an," ");
 6     for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
 7     for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
 8     for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
 9     printf "\nb:r:red\n";
10 }
11 }' | spod
```

10000 random points in a box:

```
1 echo "G 0.7 0.5 0.5  9.0 5.0 5.0  0.0 0.0 0.0  1.0 1.0 0.0
2 x:g:mygraph 10000 1 8 8 8 -4 -4 -4" | awk '{cnt++;
3 printf "%s\n", $0;
4 if (cnt==2) {
5     split($0,an," ");
6     for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
7     for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
8     for (i=1; i <=an[2]; i++) { printf "%f ",rand();}
9     printf "\np:.:points\nw:1.0:weight";
10 }
11 }' | spod
```

0.4.4 Example 2.4: graph animation

SpOd allows you to load thousands of graphs for display as an animated sequence. to date I have successfully loaded several hundred thousand graphs each containing hundreds of points.

The following script was developed to demonstrate the "sort" function. The first part of the script sets up a cluster of 400 points and saves this to a file /tmp/focus

```
1 awk 'BEGIN{
2    printf "x:y:graph %d 1 10 10 10 -5 -5 -5\n",  numb = 400;
```

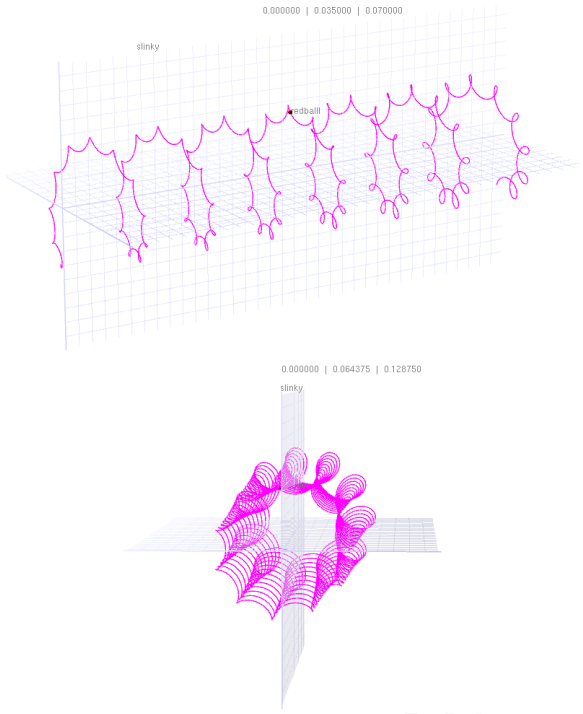


Figure 12:

```
3    for (i=1; i <= numb; i++) printf "%f ", rand(); printf "\n";
4    for (i=1; i <= numb; i++) printf "%f ", rand(); printf "\n";
5    for (i=1; i <= numb; i++) printf "%f ", rand();
6    printf "\np:.:points\nw 2\n";}' > /tmp/focus
```

The next part of the script re-computes a set of points, then computes a connecting graph for each pair of points. Because the format for graphs is largely free-format text, one may put the whole of a graph on a single line, then sort the lines (graphs) into an order based on a particular field. In this example the graphs are sorted with respect to the coordinate in the 15th field.

There are a total of nearly 80,000 graphs generated by the following script. Each begins with the 'X' object specifier, that SpOd interprets to mean that these should be played in sequence. The animated 'X' graphs should always be the last graphs in a file.

The index counter is now used to index through the graph sequence, and the window-width determines how many of the graphs will be in the view at any given instant.

```
1 head -4 /tmp/focus | tail -3  | awk '{ cnt++;
2 if (cnt==1) n=split($0,xn," ");
3 else if (cnt==2) n=split($0,yn," ");
4 else if (cnt==3) {
5    n=split($0,zn," ");
6    for (i=1; i<=n; i++) {
7        for (j=1; j<i; j++) {
8            printf "X:g:graph_%d 2 1 10 10 10 -5 -5 -5 %f %f %f %f %f %f w .3\n",cntr++, xn[i], xn
                 [j],yn[i],yn[j],zn[i],zn[j];
```

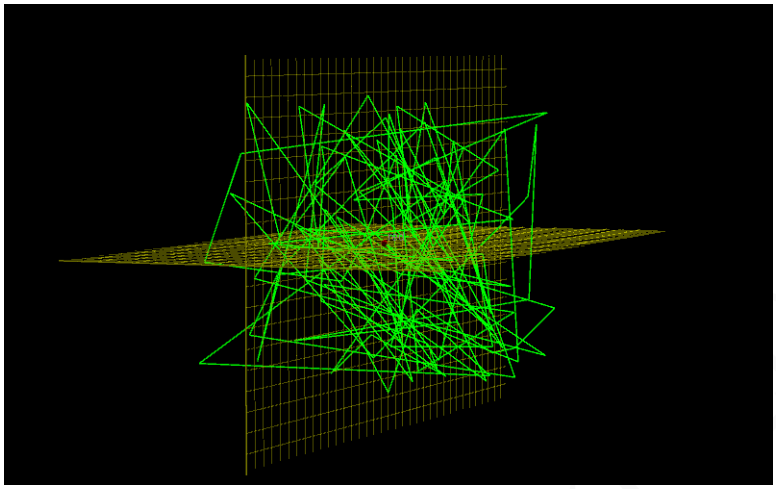


Figure 13:

```
 9          }
10      }
11 }
12 }' | sort -t " " -k 15  >> /tmp/focus
```

Then we display the resultant file

```
1 echo "O 0 128
2 L 32 13 0 0 0 0
3 C 1.000000 1.000000 0.000000 1.000000 0.000000 0.000000 0.000000
4 G 0.000000 0.000000 0.000000  9.000000 4.000000 6.000000  0.000000 0.000000 0.000000  1.000000
      1.000000 0.000000
5 M 1 0 0 10.000000
6 P 0.000000 5.000000 5.000000 5.000000
7 I -0.967501 0.000020 0.010000 0.100000 0.000000
8 `cat /tmp/focus`" | spod
```

The graph animation may be used in a variety of ways. For example one may simply display traces as if on an oscilloscope. or the graphs may be displaced in the y- or z-plane to give separation.

For example here are the graphs obtained by slicing a surface into its constituent layers. The surface is derived from the entropy of the logistic map which is also used as an example in the next section in relation to rendering surfaces.

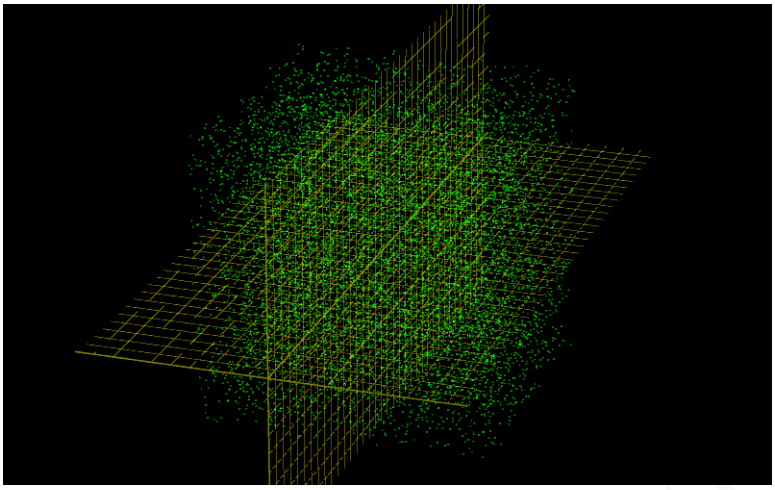


Figure 14:

0.4.5 Example 2.5: Histogram

```
awk -v "n=30" 'BEGIN{
ox=-15; oy=-5;sep=.5; width=.5
printf "T:histogram_display\n\n";
printf "W:0:10:1000:400\n\n";
printf "O: 270:0\n";
printf "L: 180:0: 259:162: 300:40\n";
printf "C: 1.000000:1.000000:0.000000: 1.000000:0.000000:0.000000: 0.000000\n";
printf "G: 0.000000:0.600000:0.000000: %d:0:%d: 0.000000:0.000000:0.000000:
      1.000000:1.000000:0.000000\n",-ox,-oy+1;
printf "M: 0:0:0: 10.000000\n";
printf "P 0.000000 5.000000 5.000000 5.000000\n\n";

printf "I: -1:0:.01333:0:-1\n\n";

printf "S:b:surface18: %d:2 : 1:0:.05: %g:-.1:%d\n",4*n,ox+sep/2,oy;
for (i=1; i <= n; i++) a[i]=rand()+.1;
for (i=1; i <=n; i++) printf "%d %d %g %g ", i-1,i-1, i-sep, i-sep;
printf "\n";
for (i=1; i <=n; i++) printf "%d %d %g %g ", i-1,i-1, i-sep, i-sep;
printf "\n\n";
for (i=1; i <=n; i++) printf "0 0 0 0 ";
printf "\n";
for (i=1; i <=n; i++) printf "0 0 0 0 ";
printf "\n\n";
for (i=1; i <=n; i++) printf "0 0 0 0 ";
printf "\n";
for (i=1; i <=n; i++) printf "0 %g %g 0 ", 100*a[i], 100*a[i];
printf "\nv:n:on\n\n";

printf "S:c:surface18: %d:2 : 1:0:.05: %g:-.3:%g\n",4*n,ox+sep/2-.175,oy;
for (i=1; i <= n; i++) a[i]=rand()+.1;
for (i=1; i <=n; i++) printf "%d %d %g %g ", i-1,i-1, i-sep, i-sep;
printf "\n";
for (i=1; i <=n; i++) printf "%d %d %g %g ", i-1,i-1, i-sep, i-sep;
printf "\n\n";
for (i=1; i <=n; i++) printf "0 0 0 0 ";
printf "\n";
```

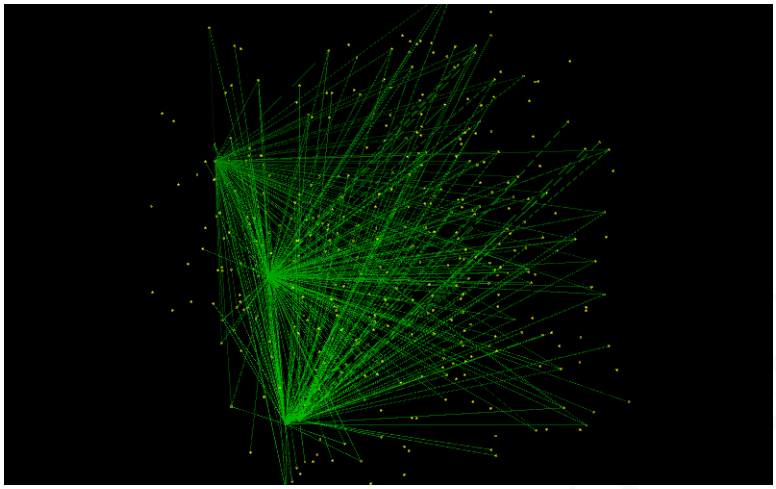


Figure 15:

```
37 for (i=1; i <=n; i++) printf "0 0 0 0 ";
38 printf "\n\n";
39 for (i=1; i <=n; i++) printf "0 0 0 0 ";
40 printf "\n";
41 for (i=1; i <=n; i++) printf "0 %g %g 0 ", 100*a[i], 100*a[i];
42 printf "\nv:n:on\n\n";
43
44 for (i=1; i <=n; i++) printf "t:w:%d:%g:-.2:%g\n", i, ox-.7+i,oy-.6;
45 printf "\n";
46 for (i=0; i <=10; i++) printf "t:w:%d:%g:-.2:%g\n", i, ox-.5,oy-.1 + i;
47
48 printf "X:y:t: 2:3 : 1:1:1 : 0:-.1:0\n%d %d 0 0 %d %d\n",ox,-ox,oy,oy;
49 printf "w:2.0";
50 printf "X:y:t: 2:3 : 1:1:1 : 0:-.1:0\n%d %d 0 0 %d %d\n",ox,ox,-oy,oy;
51 printf "w:2.0";
52 }'
```

0.5 SpOd commands

SpOd accepts pre-formatted data, essentially lists or arrays of numbers interspersed with header information and viewing parameters. It operates then as an interactive viewer. There is nothing especially sophisticated about the way objects are rendered. What this viewer does do, is provide a simple and convenient way to create from lists and arrays of data, basic graphical objects, such as surfaces, graphs, traces, and animated objects within a 3D viewing space.

A single instance of the viewer may support multiple surfaces (currently up to 10), static or animated graphs and traces, as well as visual markers, in total some 400,000 objects at a time.

Viewing is from a 2D screen, but the working space is a fixed 3D volume, nominally bounded by $(\pm 10, \pm 6, \pm 6)$ units. The object data is assumed to be in readable text floating-point format, but as the data is read into memory, it is scaled and offset to fit the viewing volume. The parameters for scaling and shifting the objects are included as part of the individual object headers.

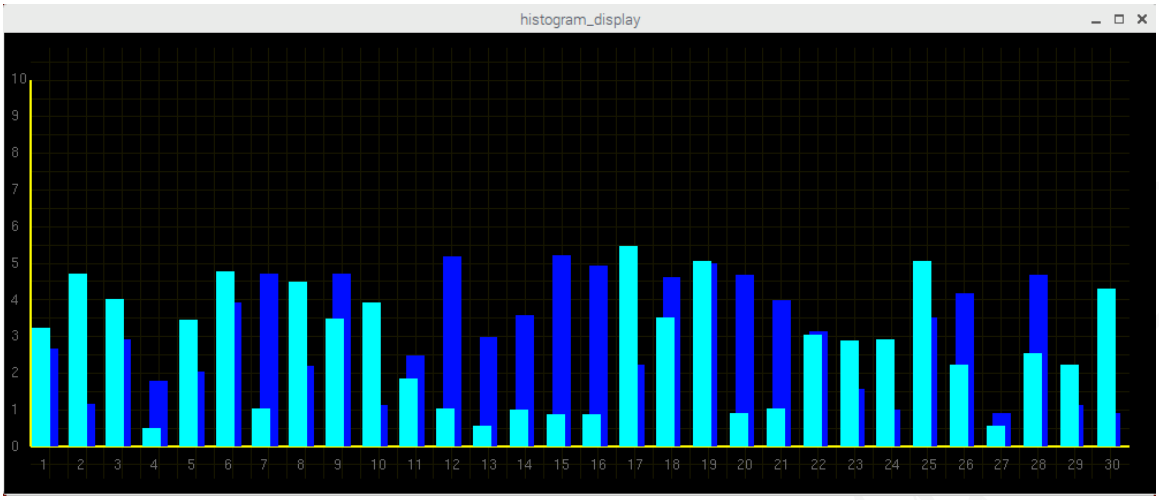


Figure 16: Example of pair of histograms rendered in SpOd as fully specified surfaces.

With the viewing space able to be rotated about a centrally located origin, objects that extend beyond the nominal volume, by even as much as a factor of a hundred, are often still able to be viewed from some angle or other, by suitably rotating into view.

Data files may be generated by any number of means, as lists or arrays from spreadsheets, or output from environments like Matlab or Octave, or from shell scripts. By assuming a text format the graphics files may also be conveniently edited within a conventional text editor. It is sometimes quicker to cut and paste objects together, or edit display parameters than to regenerate object data and header formats from programmed scripts.

Data objects may be stored collectively or separately. The UNIX cat command may be conveniently used to combine objects on the fly for viewing purposes. The object data files may provide a convenient stepping off point for subsequent processing steps. The further processing may be then coded appropriately for viewing. It is often convenient to retain the series of intermediate processing stages creating in effect a visual journal of the research steps.

Multiple and simultaneously instances of the viewer process may also be invoked independently with separate object source files, giving even greater flexibility and opportunity for a comparative display of research results.

The object file format is to be described here: (still coming)

Once the objects are loaded you may flexibly view them by using appropriate key/mouse actions. These are summarised below both in alphabetic order, but also by function.

Alphabetic listing of key commands (note: that since much of the software was developed in relation to sleep studies, there are some commands that are specific to this work.)

Keyboard Letter-Function Assignments			
key	action ($)		
a	Sleep staging :- sensors disconnected/connect	I	toggle simultaneous/animated d
b	toggle surface display mode:	L	toggle static graph-intensity by
	i) triangular tessellations,	O	display sleep staging
	ii) colour gradiant, iii) smoothed	R	REM
d	Sleep staging :- s1	R	REM
e	Sleep staging :- s2	S	Sleep
f	Sleep staging :- nrem	T	tumble forward rate decrease
g	Go, animation : - rate increase or decrease	S	Sleep
h	Animation : - window width increase	T	tumble forward rate decrease
i	Index animation : - toggle scope mode,	< sp >	freeze animated display, reset a
	i) w/out motion	1	toggle select/deselect 1st graphi
	ii) z-motion	2	toggle select/deselect 2nd graph
	iii) y-motion	3	toggle select/deselect 3rd graph
j	Jump to next stored view	4	toggle select/deselect 4th graph
l	labels display/inhibit toggle	5	toggle select/deselect 5th graph
o	Sleep staging markup computed	6	toggle select/deselect 6th graph
r	Sleep staging :- rem sleep	7	toggle select/deselect 4th surfac
s	Sleep staging :- sleep/wake	8	toggle select/deselect 3rd surfac
t	tumble forward rate increase	9	toggle select/deselect 2nd surfac
w	Sleep staging :- wake/sleep	0	toggle select/deselect 1st surfac
A	Sleep staging :- sensors connect /disconnected	!	illuminated scale $z - x$ plane, b
B	BOTH tumble/roll/animation, used with t/T and g/G	@	illuminated scale $z - x$ plane, b
D	Sleep staging :- S1/S2	#	illuminated scale $y - x$ plane, b
E	Sleep staging :- S2/S1	$	illuminated scale $y - x$ plane, b
F	Sleep staging :- NREM	%	toggle major scale axis
G	Animation :- reverse rate increase /forward rate decrease	<esc>	exit display
H	Animation :- window width decrease		
→	surface colour, red/green shift	←	surface colour, green/red shift
↑	light intensity offset increase	↓	light intensity offset decrease
shft →	blue light increase	shft ←	blue light decrease
left btn		right btn	(option + mouse)
		middle btn	(ctrl + mouse)

Table 2: A summary of the interactive key commands to manage the display.

Tools that work with SpOd

0.6 background

SpOd is an interactive visualisation tool for primarily displaying time series or data arrays as graphs and surfaces. Because it is relatively versatile it can also be used to depict histograms and other graphical features as illustrated in some of the following examples. It also offers several filter functions for example, for the extraction of graphical objects from a collection of objects in a given file or stream. Several support tools can be used in conjunction with SpOd to manipulate functional views, e.g. rotation of arrays objects or translation of surface objects to graphical slices. Still other signal processing tools for computing entropy metrics from time series may generate SpOd ready views of the resulting data. Our entropy computation derives from novel string structures that emerge from the T-code construction and render graphs and surfaces in a variety of complexity, information and entropy units.

We introducing some of the computational tools here but also include an introduction to some of the string manipulation tools that set the stage for the entropy, information and complexity tools. This chapter alos introduces a set of unix tools that were developed early in relation to T-codes. This is a diversion from the main thrust of the documentation but is included for the sake of completeness.