Display3D via Open Source 3D Software

T. H. Merrett*

McGill University

March 12, 2008

[1]

1. Gnuplot [WK07]

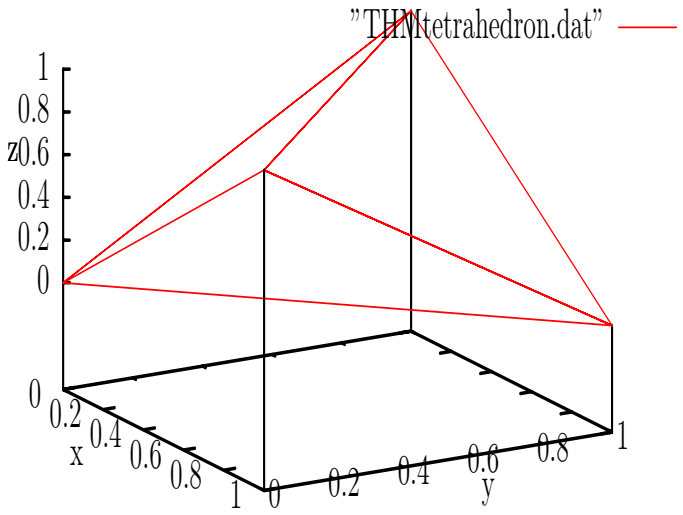
Here is a 3D plot, made from the file

```
# THMtetrahedron.dat
0 0 0
0 1 1

0 0 0
1 0 1

0 0 0
1 1 0

0 1 1
1 0 1

0 1 1
1 1 0

1 0 1
1 1 0
```

using the command

The author gratefully acknowledges support from the taxpayers of Québec and of Canada who have paid his salary and research grants while this work was developed at McGill University, and from his students (who built the implementations and investigated the data structures and algorithms) and their funding agencies.

```
gnuplot> set style data line
gnuplot> splot "THMtetrahedron.dat"
```



The corresponding relation would be, in keeping with the conventions already implemented in **display2D**,

THMtetrahedron

(*x1*	*y1*	*z1*	*x2*	*y2*	*z2*)
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	1	1	1	0	1
0	1	1	1	1	0
1	0	1	1	1	0

and conversion to `gnuplot` format would have to put *x2, y2* and *z2* on a second line, followed by a blank line between edges.

Once splot has drawn the picture, one can interact with it by using one mouse button within the window to change the angle of view (x and z angles), and another mouse button to change the scale (x and y). But many useful interactions are not provided, such as allowing the end-user to chose 3D hidden lines or not.

2. Gnuplot can come closer to Aldat needs with its *vector* mode.

```
gnuplot> splot "THMtetraVect.dat" with vectors nohead
```

draws the tetrahedron using file

```
# THMtetraVect.dat
0 0 0 0  1  1
0 0 0 1  0  1
0 0 0 1  1  0
0 1 1 1 -1  0
0 1 1 1  0 -1
1 0 1 0  1 -1
```

Here the second triple is the vector itself rather than its endpoint. (Matlab uses `quiver3d` in the same way.)

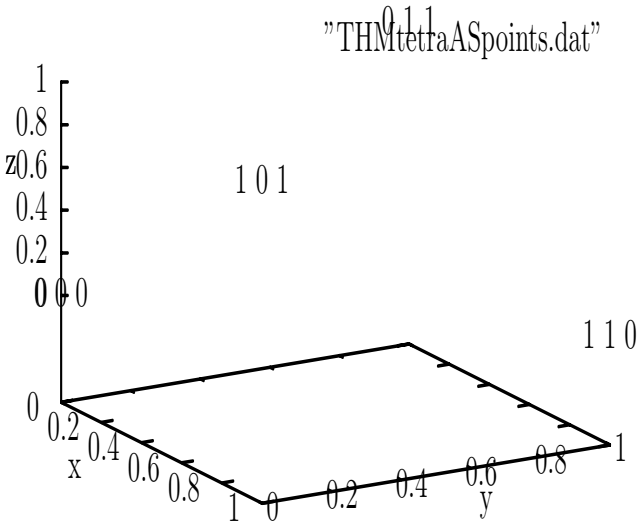
3. Gnuplot allows a column of labels for points.

```
# THMtetraASpoints.dat
0 0 0 "0 0 0"
0 1 1 "0 1 1"
1 0 1 "1 0 1"
1 1 0 "1 1 0"
```

The code

```
gnuplot> set style data point
gnuplot> splot "THMtetraASpoints.dat" with labels
```

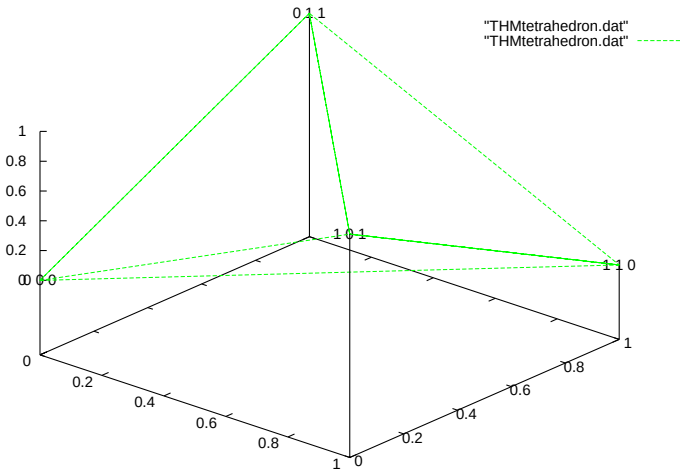
produces an untidy result (I'm sure an experienced `gnuplot` user could clean it up) but gives the idea.



I think there is a way to plot the points symbolically (a red + is the default if we had omitted the phrase `with labels`) *and* show the labels. **Display2D**, using `xfig`, has this ability. Adapt the following.

4. Since `with labels` overrides line-drawing, we can show both lines and labelled points only by code such as

```
gnuplot> set style data line
gnuplot> splot "THMtetrahedron.dat" with labels, "THMtetrahedron.dat"
```



5. We could in principle support the triangle facility of **display2D**, by generating

```
# THMtetraAStri.dat
0 0 0
0 1 1
1 0 1
0 0 0

0 0 0
1 0 1
1 1 0
0 0 0

0 0 0
1 1 0
0 1 1
0 0 0

1 0 1
0 1 1
1 1 0
1 0 1
```

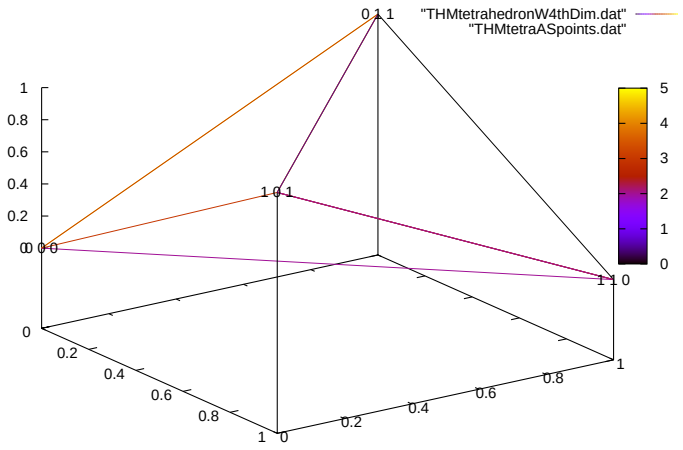
6. Display2D can plot different points and lines in different colours, specified in the relation. Here's the closest I've come with `gnuplot`, using file

```
# THMtetrahedronW4thDim.dat
0 0 0 5
0 1 1 5

0 0 0 3
1 0 1 3

0 0 0 2
1 1 0 2

0 1 1 4
1 0 1 4

0 1 1 0
1 1 0 0

1 0 1 1
1 1 0 1
```

The code

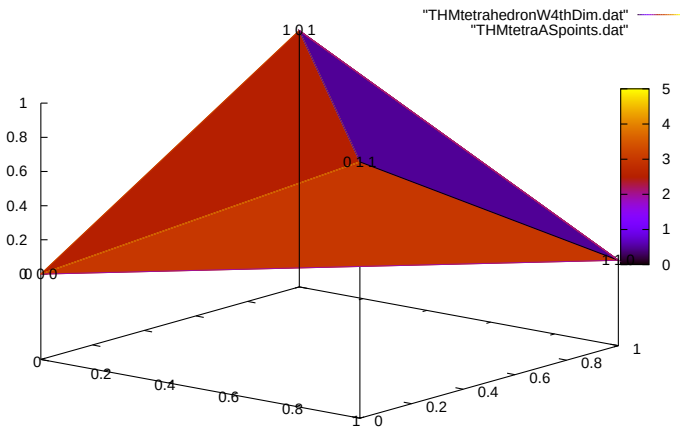
```
gnuplot> splot "THMtetrahedronW4thDim.dat" lc pal, "THMtetraASpoints.dat" with labels
```

gives (`pal` abbreviates `palette`)



To get coloured surfaces, we can use the same file and code, first turning on

```
gnuplot> set pm3d
```

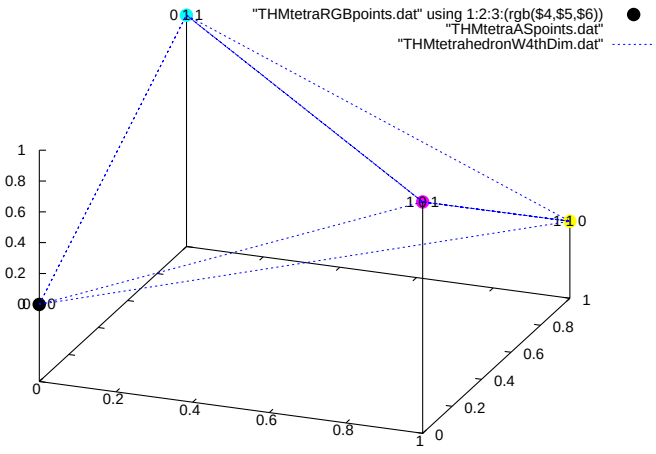


7. Here's an improvement, using the previous two files and

```
# THMtetraRGBpoints.dat
#x y z r   g   b
 0 0 0 0   0   0
 0 1 1 0   255 255
 1 0 1 255 0   255
 1 1 0 255 255 0
```

Note how we can calculate an rgb value from the 4th, 5th and 6th columns in the file, specify this rgb as a fourth column for the plot, and pick it up with `rgb var` (or `rgb variable`) after the `lc` (or `linecolor`) specification. Note also the further specifications for `pt` or `pointtype` 7 (a filled circle), and `ps` or `pointsize` 2.

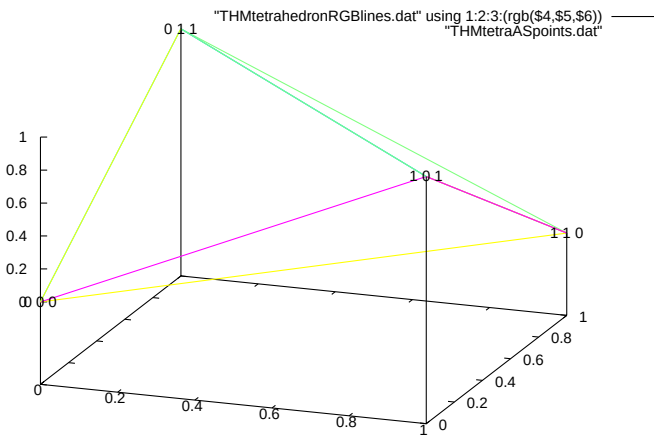
```
gnuplot> rgb(r,g,b) = int(r)*65536 + int(g)*256 + int(b)
gnuplot> splot "THMtetraRGBpoints.dat" using 1:2:3:(rgb($4,$5,$6))
                  with points pt 7 ps 2 lc rgb var, \
> "THMtetraASpoints.dat" with labels, "THMtetrahedronW4thDim.dat"
```



We can do something for lines, using file

```
# THMtetrahedronRGBlines.dat
#x y z r   g   b
 0 0 0 0   255 255
 0 1 1 0   255 255

 0 0 0 255 0   255
 1 0 1 255 0   255

 0 0 0 255 255 0
 1 1 0 255 255 0

 0 1 1 127 127 255
 1 0 1 127 127 255

 0 1 1 127 255 127
 1 1 0 127 255 127

 1 0 1 255 127 255
 1 1 0 255 127 255
```

```
splot "THMtetrahedronRGBlines.dat" using 1:2:3:(rgb($4,$5,$6)) with lines lc rgb var,\
> "THMtetraASpoints.dat" with labels
```



8. Gnuplot makes it easier to support plotting functions than MATLAB, because it generates grids automatically from the file format.

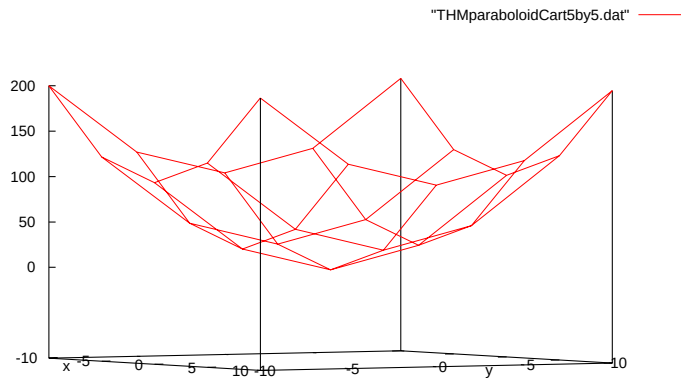
```
# THMparaboloidCart5by5.dat
# x   y   z
 -10 -10 200
 -10  -5 125
 -10   0 100
 -10   5 125
 -10  10 200

  -5 -10 125
  -5  -5  50
  -5   0  25
  -5   5  50
  -5  10 125

   0 -10 100
   0  -5  25
   0   0   0
   0   5  25
   0  10 100

   5 -10 125
   5  -5  50
  10   0 100
  10   5 125
  10  10 200
```

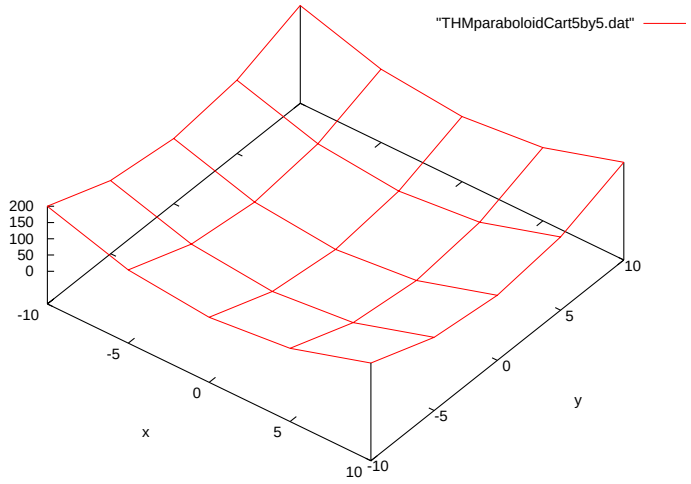
Here are two views of this plot (which is the paraboloid $x^2 + y^2$), given respectively by

```
gnuplot> set xlabel "x"
gnuplot> set ylabel "y"
gnuplot> set view 87,59
gnuplot> splot "THMparaboloidCart5by5.dat"
```

and by

```
gnuplot> set view 21,38
gnuplot> splot "THMparaboloidCart5by5.dat"
```





We see that Aldat need only generate the relation

$$\{(-10), (-5), (0), (5), (10)\}$$

take the Cartesian product with itself and then actualize

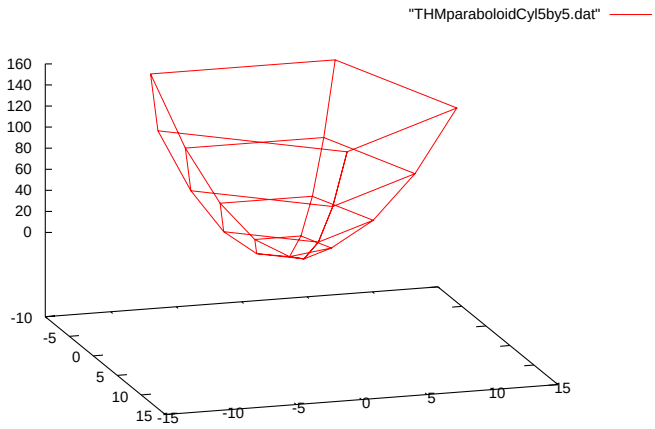
let z **be** x**2 + y**2;

to generate the required relation. The **display3D** mechanism would sort and insert blank lines to produce the file needed by `gnuplot`.

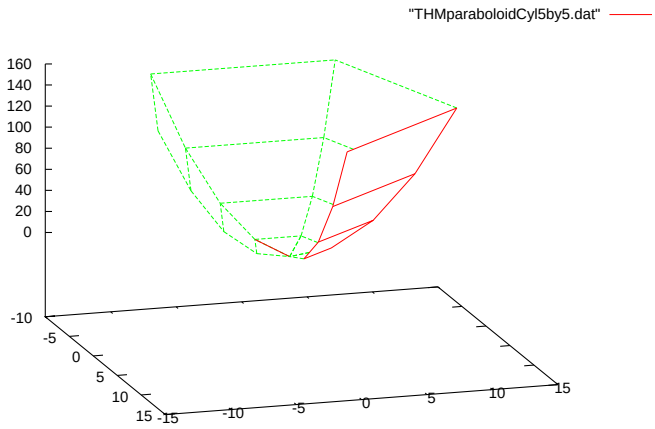
9. We can do it in cylindrical coordinates, using file

```
# THMparaboloidCyl5by5.dat
# t   z   r
  0   0   0
  0   9   3
  0  36   6
  0  81   9
  0 144  12

 72   0   0
 72   9   3
 72  36   6
 72  81   9
 72 144  12

144   0   0
144   9   3
144  36   6
144  81   9
144 144  12

216   0   0
216   9   3
216  36   6
216  81   9
216 144  12

288   0   0
288   9   3
288  36   6
288  81   9
288 144  12

  0   0   0
  0   9   3
  0  36   6
  0  81   9
  0 144  12
```

```
gnuplot> set mapping cyl
gnuplot> set angles deg
gnuplot> splot "THMparaboloidCyl5by5.dat"
```



Note that the file must close the figure by repeating the data for 0 degrees. If this is omitted, the paraboloid does not close between 288 and 0 degrees. (I also enabled hidden line removal below by writing `set hidden`.)



To build this file in Aldat, we need only generate the r values, 0, 3, .. 12, compute the corresponding $z = r^2$, and then take the Cartesian product with the five angles.

10. Fourth dimensions: animation Gnuplot, in its bizarre way, supports a number of language capabilities which we can use for animation.

Subroutines and looping/recursion support the simplest kind of animation. Here is a "subroutine" to rotate the point of view of a 3D plot of a tetrahedron.

```
# THMrotaTetra.dem
set view 60,v2,1,1
splot "THMtetrahedron.dat"
v2 = v2 + 1
if (v2<346) reread
```

It is invoked by

```
v2 = 1
load "THMrotaTetra.dem"
```

But how can we animate the plot itself, changing the internal structure with time? For instance, we might want to alternate between the two tetrahedra that share three vertices.

```
# THMtestUsing.dat
# 0--3 tetrahedron 1
# x  y  z
  0  1  1
  1  0  1
  1  1  0
  0  0  0
# 4--7 tetrahedron 2
# x  y  z
  0  1  1
  1  0  1
  1  1  0
1.3 1.3 1.3
```

The `using` clause (see Note 7) already permits something like projection. It also turns out to support selection, albeit weirdly.

```
# THMtestUsing.dem
beg = 4*(ct%2)
fin = beg + 3
set xrange [0:2]
set yrange [0:2]
set zrange [0:2]
splot "THMtestUsing.dat" using 1:2:(beg<=$0 && $0<=fin ? $3 : 1/0) with points
       pt 7 ps 2
ct = ct + 1
pause 0.1
if (ct<100) reread
reset
```

This is invoked by setting `ct = 0` and then `load "THMtestUsing.dem"`. The key is the use of column 0: this counts the (non-comment) rows in the file. So the conditional expression tests whether the row number is in the range `beg` to `fin`. If it is not, the code tells `splot` to show the undefined quantity `0/1`, which `splot` ignores. Thus it plots only the rows for which the range condition is true. *Ad-hoc*, isn't it?

(I used `set xrange`, etc. so that the axis ranges do not jump around between the different plots. This has side-effects: the next plot will have its axis ranges fixed at these values unless

they are reset.)

11. Now let's do someting more ambitious. We'll use Aldat to create a four-dimensional simplex (the 4D triangle or tetrahedron: 5 vertices are all connected by 10 edges), to rotate it in one of our three dimensions and the fourth, and to draw a perspective projection of it into a three-dimensional "display" from the point of view of an "eye", located at positions d and e, respectively along the fourth dimension. Aldat creates a relation with ten tuples for each degree of rotation, from 0 to 89, and `gnuplot` animates the drawing of this relation (which we call *display3D* because it suggests the **display3D** interface we should build into Aldat).

Here is the Aldat code.

```
<< simplex4D.a THM 080310>>
domain w, x, y, z real;
relation simplex4D(w,x,y,z) <- {(0.0,1.0,1.0,1.0),(1.0,0.0,1.0,1.0),
                                (1.0,1.0,0.0,1.0),(1.0,1.0,1.0,0.0),
                                (0.190983,0.190983,0.190983,0.190983)};
let w2 be w;
let x2 be x;
let y2 be y;
let z2 be z;
simpEdges4D <- where z + 10*(y + 10*(x + 10*w)) < z2 + 10*(y2 + 10*(x2 + 10*w2))
                in (simplex4D ijoin [w2,x2,y2,z2] in simplex4D);

domain t real;
relation angle(t);
relation maxa(t) <- {(89.0)};
comp post:add:angle() is
{ let tmin1 be (red min of t) - 1.0;
  let t be tmin1;
  update angle add [t] in [tmin1] where tmin1>=0 in angle
};
update angle add maxa;

let tRad be t*3.14159/180.0;

let w' be w*cos(tRad) - x*sin(tRad);        << rotate in  wx  plane >>
let x' be w*sin(tRad) + x*cos(tRad);        << rotate in  wx  plane >>

let w2' be w2*cos(tRad) - x2*sin(tRad);     << rotate in  wx  plane >>
let x2' be w2*sin(tRad) + x2*cos(tRad);     << rotate in  wx  plane >>

let e be 10.0;        << position of "eye" along  w  axis in 4D >>
let d be 5.0;         << position of "display" along  w  axis in 4D >>

let p be x'*(e-d)/(e-w');
let q be y*(e-d)/(e-w');
let r be z*(e-d)/(e-w');
```

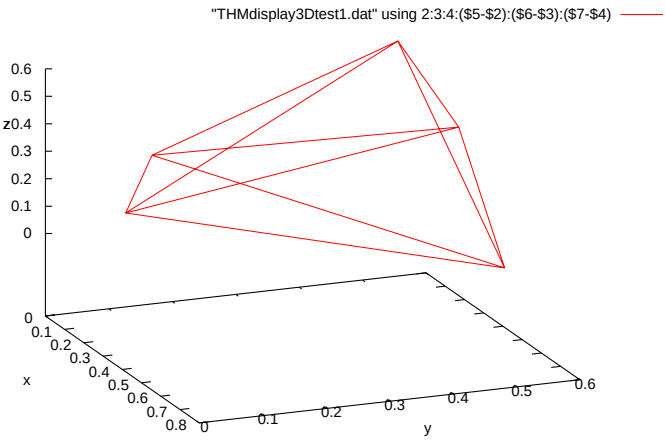
```
let p2 be x2'*(e-d)/(e-w2');
let q2 be y2*(e-d)/(e-w2');
let r2 be z2*(e-d)/(e-w2');

display3D <- [t,p,q,r,p2,q2,r2] in [t,w',x',y,z,w2',x2',y2,z2,e,d] in
             (simpEdges4D ijoin [t,tRad] in angle); << to be passed to gnuplot>>
```

The relation *display3D*(*t,p,q,r,p2,q2,r2*) is edited to replace Aldat's field delimiters (`ctrl-F`) by spaces and stored as `THMdisplay3D.dat`. Here is the `gnuplot` code to animate it.

```
# THMrotSimp4D.dem                          THM                                  080310
len = 90
beg = 10*(ct%len)
fin = beg + 9
set xrange [0:.8]
set yrange [0:.6]
set zrange [0:.6]
splot "THMdisplay3D.dat" using 2:3:4:($5-$2):($6-$3):(beg<=$0 && $0<=fin ? $7-$4 : 1/0)
       with vectors nohead
ct = ct + 1
if (ct<len) reread
reset
```

It is invoked, as THMtestUsing.dem was, by setting `ct = 0` and then loading it. Note that I have used the `with vectors nohead` option to draw the edges. This means that the last three columns provided must be the differences between the last three columns calculated by Aldat and the first three columns.

I can't show the animation, but here is a snapshot of the start of the animation.



Try this with a tesseract, the four-dimensional hypercube. It has 32 edges, but you should find the resulting 3D projection much more intuitive.

12. Since **display3D** is intended to be an editor, not just an output device, we need a means of input to `gnuplot`. It is hard to update three-dimensional images from a two-dimensional

screen, and `gnuplot` offers no high-level facilities of the sort that `xfig` makes possible for **display2D**. But there is a way to input mouse coordinates from the screen.

```
pause mouse keypress
print "Keystroke ", MOUSE_KEY, " at ", MOUSE_X, " ", MOUSE_Y
```

gives the following values if we press keys `a, b, c` and `d` with the mouse positioned respectively at the lower left, lower right, upper left and upper right corners of the plot window.

```
Keystroke 97 at -0.177972465581977 -0.184316267012314
Keystroke 98 at 1.60518783542039 1.48395061728395
Keystroke 99 at -0.0685152057245081 -0.193827160493827
Keystroke 100 at 1.07692307692308 1.23868312757202
```

Here is a second test, using uppercase

```
Keystroke 65 at -0.170661896243292 -0.196296296296296
Keystroke 66 at 1.60518783542039 1.48395061728395
Keystroke 67 at -0.0685152057245081 -0.193827160493827
Keystroke 68 at 1.07513416815742 1.23662551440329
```

And a third time, with the window moved to another part of the screen, and using keys 1, 2, 3 and 4

```
Keystroke 49 at -0.170661896243292 -0.196296296296296
Keystroke 50 at 1.60769230769231 1.48641975308642
Keystroke 51 at -0.0706618962432916 -0.196296296296296
Keystroke 52 at 1.07513416815742 1.23662551440329
```

(There is the option of using mouse clicks, but even when they are combined with other keys, as claimed for `MOUSE_SHIFT`, `splot` confuses them with mouse operations to rotate or scale the 3D image.)

13. By the way, to make `gnuplot` prepare eps (enhanced Postscript) files for inclusion in LaTeX documents, change terminals temporarily, e.g.:

```
gnuplot> set terminal postscript eps colour
gnuplot> set output "tetra4thDlc.eps"
gnuplot> set view 62,55
gnuplot> splot "THMtetrahedronW4thDim.dat" lc pal, "THMtetraASpoints.dat" with labels
gnuplot> set terminal x11
```

References

[WK07] Thomas Williams and Colin Kelley. gnuplot: An interactive plotting program. gnuplot.sourceforge.net/docs/gnuplot.pdf, 2007.