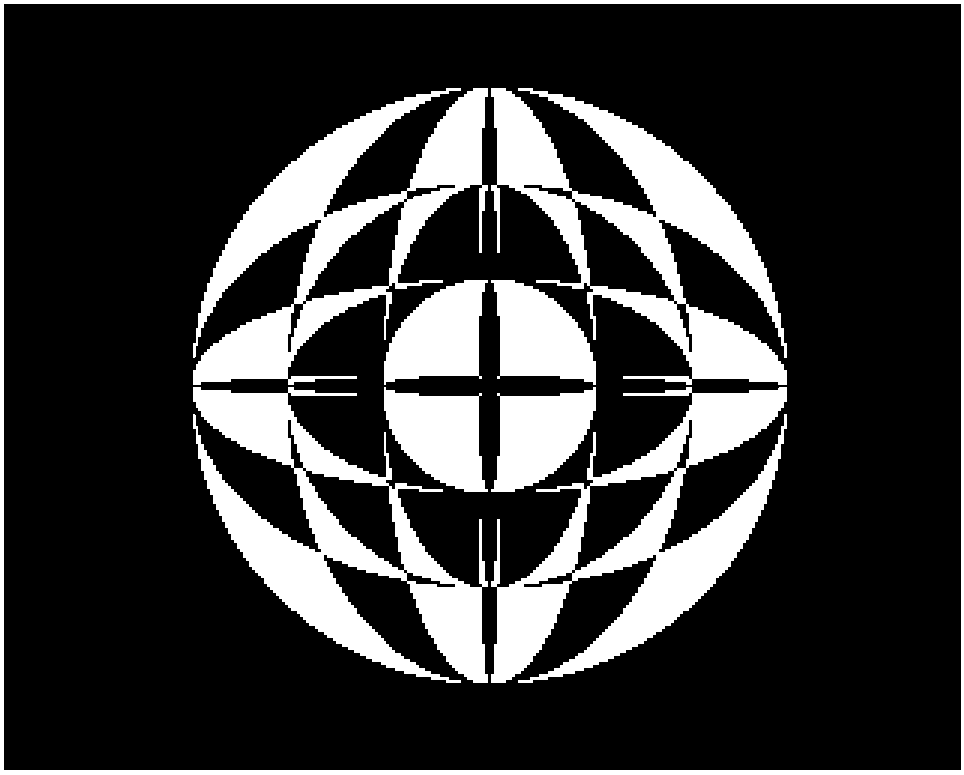
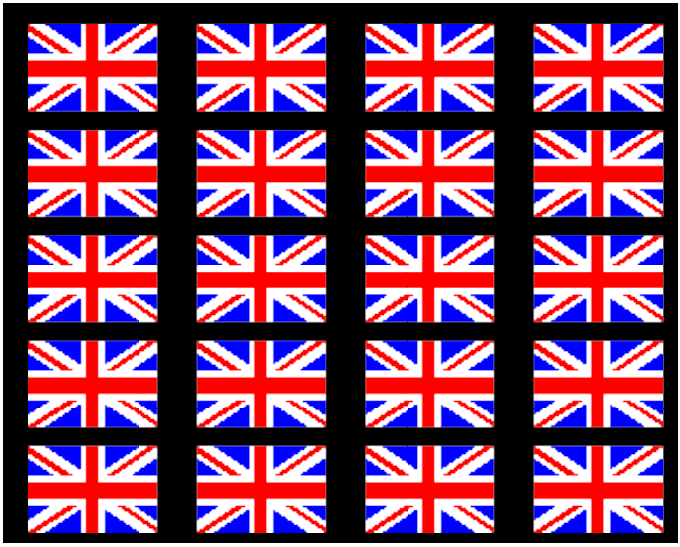
VISUAL DISPLAYS



UNION JACK
by David Stonebanks



This program produces a whole series of representations of the Union Jack with various aspect ratios. The display is colourful and eye-catching, and gives a good demonstration of the quality of Electron graphics.

Pressing the spacebar during the running of the program will freeze the display at the end of the frame in progress. Pressing the spacebar again will release it.

The flags are drawn by the procedure PROCunionjack, which calls two further procedures PROC box and PROCdiag. The program lines 170-370 sequence through the range of sizes and shapes of the flag, and the multi-flag display. On each occasion the height (vsize) and width (hsize) of the flag are set before calling PROCunionjack. Note the use of VDU29 in lines 160 and 340 to change the graphics origin (see User Guide p.113), and so print the flag off-centre.

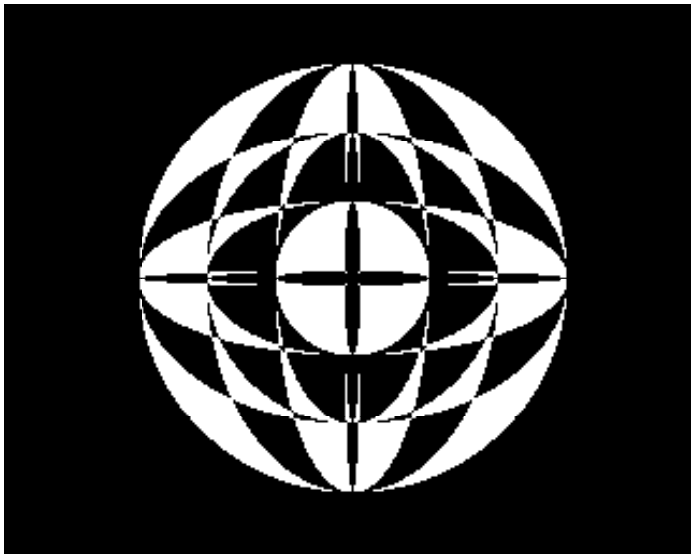
```
10 REM Union Jack
20 REM by D.Stonebanks
30 REM BEEBUG
```

```
  40 REM VERSION P 1.0
  50 :
 100 ON ERROR ON ERROR OFF:MODE 6:REPORT:PRINT" at
line ";ERL:END
 110 REPEAT
 120 MODE 1
 130 VDU23,1,0;0;0;0;:REM delete cursor
 140 black=0:red=1:blue=2:white=3
 150 VDU19,blue,4;0;
 160 VDU29,640;512;:REM set graphics
 170 FOR hsize=5 TO 60 STEP 10
 180 vsize=hsize:REM square
 190 PROCunionjack
 200 PROCdelay(100)
 210 NEXT hsize
 220 CLS
 230 FOR hsize=5 TO 60 STEP 10
 240 FOR vsize=5 TO 60 STEP 10
 250 PROCunionjack
 260 PROCdelay(100)
 270 NEXT vsize
 280 CLS
 290 NEXT hsize
 300 CLS
 310 hsize=10:vsize=10
 320 FOR X=160TO1200 STEP 320
 330 FOR Y=100 TO 900 STEP 200
 340 VDU29,X;Y;:REM move graphics origin
 350 PROCunionjack
 360 NEXT Y
 370 NEXT X
 380 PROCdelay(200)
 390 UNTIL FALSE
 400 END
 410 :
1000 DEFPROCunionjack
1010 PROCbox(blue,12,8)
1020 PROCdiag(white,9,8,-12,-6,12,8,-12,-8)
1030 PROCdiag(white,12,8,-12,-8,12,6,-9,-8)
1040 PROCdiag(white,-9,8,-12,8,12,-6,12,-8)
1050 PROCdiag(white,-12,8,-12,6,12,-8,9,-8)
```

```
 1060 PROCdiag(red,10.5,8,0,1,12,8,0,0)
 1070 PROCdiag(red,0,0,-12,-8,0,-1,-10.5,-8)
 1080 PROCdiag(red,0,0,-12,8,0,-1,-12,7)
 1090 PROCdiag(red,0,0,12,-8,0,1,12,-7)
 1100 PROCbox(white,2,8)
 1110 PROCbox(white,12,2.5)
 1120 PROCbox(red,1,8)
 1130 PROCbox(red,12,1.5)
 1140 IF INKEY-99 THEN REPEAT UNTIL NOT INKEY-99:RE
PEAT UNTIL INKEY-99
 1150 ENDPROC
 1155 :
 1160 DEF PROCbox(colour,halfx,halfy)
 1170 GCOL0,colour
 1180 MOVEhalfx*hsize,halfy*vsize:MOVE-halfx*hsize,
halfy*vsize
 1190 PLOT85,halfx*hsize,-halfy*vsize:PLOT85,-halfx
*hsize,-halfy*vsize
 1200 ENDPROC
 1205 :
 1210 DEFPROCdiag(colour,x1,y1,x2,y2,x3,y3,x4,y4)
 1220 GCOL0,colour
 1230 MOVEx1*hsize,y1*vsize:MOVEx2*hsize,y2*vsize
 1240 PLOT85,x3*hsize,y3*vsize:PLOT85,x4*hsize,y4*v
size
 1250 ENDPROC
 1260 DEFPROCdelay(t)
 1270 T=TIME+t:REPEAT UNTIL TIME>T
 1280 ENDPROC
```

ELLIPTO
by S. Wilkinson

This is a relatively short program which produces varied patterns by plotting a succession of filled ellipses of random size. The ellipses have the same centre, so the pattern is gradually built up using so-called 'exclusive or' plotting (achievedwith the

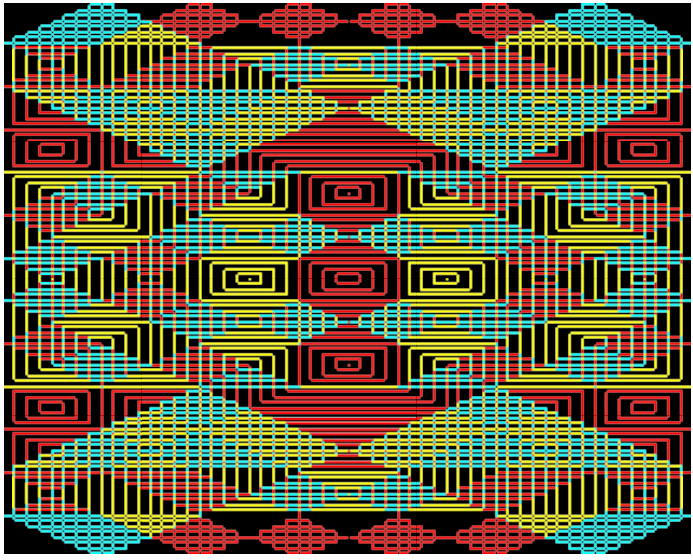


GCOL 3,1 statement in line 160 -- see User Guide p.153). At the start of the program you can choose between modes 0 and 4 for the display.

```
   10 REM Ellipto
   20 REM by S.Wilkinson
   30 REM BEEBUG
   40 REM VERSION P 1.0
   50 ON ERROR GOTO 290
   60 MODE 6
   70 PRINT
   80 REPEAT:CLS
   90 PRINT'''TAB(12);"E L L I P T O"''TAB(12);"by
S.Wilkinson"
  100 INPUTTAB(0,15)"Mode 0 or Mode 4 :"M%
  110 UNTIL M%=4 OR M%=0
  120 MODE M%
  130 IF M%=0 M%=2
  140 VDU23,1,0;0;0;0;
  150 VDU29,640;512;
  160 GCOL3,1
  170 C%=RND(7)*16+16
  180 FORA=16 TO 512 STEP C%
  190 FORB=16 TO 512 STEP C%
```

```
200 L%=-A
210 MOVEL%,0:DRAW-L%,0
220 FOR Y%=4 TO B STEP4
230 X%=A/B*SQR(B*B-Y%*Y%)/M%:X%=X%*M%
240 MOVE X%,Y%:DRAW -X%,Y%
250 MOVE X%,-Y%:DRAW -X%,-Y%
260 NEXT Y%
270 NEXT,
280 A=GET:CLS:GOTO170
290 ON ERROR OFF
300 MODE6:IF ERR=17 END
310 REPORT:PRINT" at line ";ERL
320 END
```

SQUARE DANCE
by Martin Richards



This program displays a series of explanding and contracting rectangles which change colour during the course of execution. The actual form of the

patterns is random so that re-runs will display different pattenrs. Some patterns repeat after a while, whilst others appear to keep on changing, giving an interesting display of coloured graphics.

The commands GCOL and VDU19 (see User Guide p.97 and 102) are used to good effect in producing the many colours generated. Some interesting fringe patterns are also produced if you watch the output on a monochrome TV, especially if you try it in MODE 1.

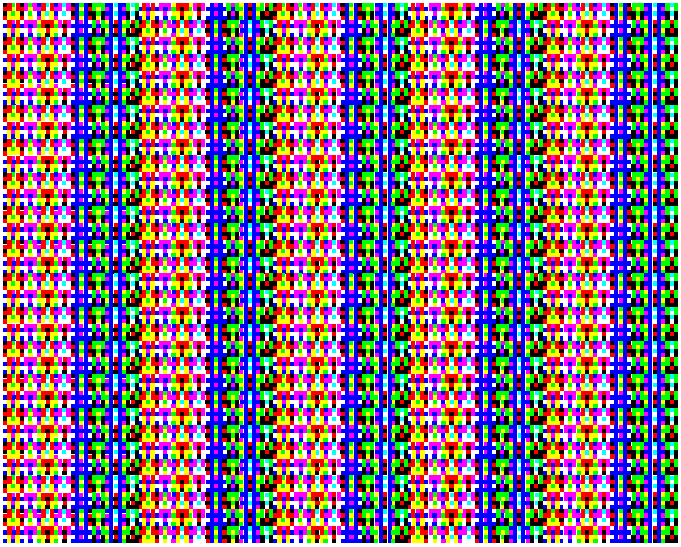
Program analysis

Line 80	Gets rid of the cursor.
Line 90	Puts the graphics origin at the centre of the screen.
Line 100	Selects the initial height and width of the rectangle.
Line 110	Selects the speed at which the sides are to change.
Line 120	Selects the initial colour.
Line 130 to 200	Main Loop (press ESCAPE to stop).
Line 140 to 160	Draws any particular rectangle.
Line 170	Chooses the next width and height.
Line 180	If rectangle is too wide then change direction and colour.
Line 190	If rectangle is too high then change direction and logical colour.

```
 10  REM Square Dance
 20  REM by M.Richards
 30  REM BEEBUG
 40  REM VERSION P 1.0
 50 :
 60  ON ERROR GOTO 210
 70  MODE 1
 80  VDU23,1,0;0;0;0;
 90  VDU29,640;510;
100  X=0:Y=0
110  DX=RND(50):DY=RND(50)
120  C=RND(3)
130  REPEAT
140  GCOL 3,C
150  MOVEX,Y:DRAW-X,Y:DRAW-X,-Y
160  DRAWX,-Y:DRAWX,Y
```

```
  170  X=X+DX:Y=Y+DY
  180  IFABS(X)>640 THEN DX=-DX :C=RND(3)
  190  IFABS(Y)>510 THEN DY=-DY:VDU 19,RND(3),RND(7
);0;
  200  UNTIL TRUE=FALSE
  210  ON ERROR OFF:MODE 6
  220  REPORT :PRINT" @ line ";ERL
  230  END
```

SCREENPLAY
by I. Bell



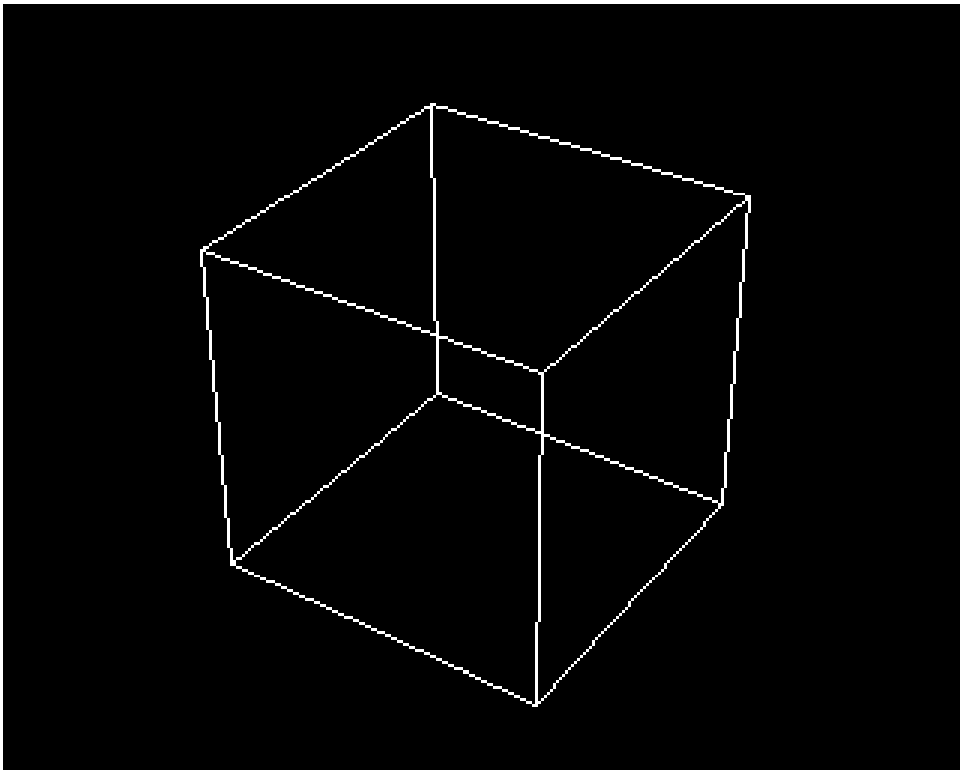
This program shows the speed of the Electron' s graphics and the ease and speed of the Assembler. It runs in mode 2 and uses all 16 colours. To continue drawing each picture simply press the space bar.

This program will run in any mode, and to do this just change the value of mode in line 60, and also set NC to the number of colours available in that mode, for example:

60 MODE 1:NC=4

```
 10  REM Screenplay
 20  REM by I.Bell
 30  REM BEEBUG
 40  REM VERSION P 1.0
 50 :
 60  MODE 2:NC=16
 70  VDU23,1,0;0;0;0;:P%=&2F00
 80  [OPT0
 90  .ST LDX#0:STX&70:LDX#&30:STX&71:LDX#0
100  .HERE LDA&70:STA(&70,X):INC &70:BNE HERE
110  INC &71:LDY &71:CPY#&80:BNE HERE:RTS
120  ]
130  CALL ST
140  REPEAT
150   VDU19,RND(NC)-1,RND(8)-1,0,0,0
160   REPEAT UNTIL GET=32
170  UNTIL FALSE
```

3-D Rotator
by James Hastings



' 3-DRotator' enables shaes to be drawn in three dimensions and then viewed from any angle.

Data statements define the object to be drawn. These statements are easily set up, as they are the X, YZ co-ordinates of the ' corners'of the objects. Once these are entered and the program is run, a front view is displayed in mode 4 graphics. The object may then be rotated about any axis or viewed from nearer or further simply by pressing one of 8 keys. The program will only draw and rotate objectsmade up from straight lines, but there is no apparent limit to the complexity of the ' wire-frame' objects generated.

How to use the program
The main program starts from line 100 so that earlier lines can be used for data statements defining objects to be developed later. The data statements on lines 7 to 12 define a 3-D cube.

Run the program. You should see a cube, viewed from one side. The cube may now be moved using the following keys:

(a) Cursor keys LEFT and RIGHT rotate around the Y axis.

(b) Cursor keys UP and DOWN rotate around the X axis.

(c) ' RETURN'and the ' :'key (left of RETURN key) rotate around the Z axis.

(d)' <'and ' >'reduce and increase the size of the object.

This provides the ability to view from any point, including actually from within the object itself. The program always draws all the lines of the object concerned as if it were a wire frame.

Creating a new 2-D object
Designing your own shape is very easy. To stat with let us consider how to draw a simple square in 2 dimensions. (The program will also draw and display 2-D objects in a 3-D field of vision.)

Take a piece of paper (graph paper is best), and draw a large cross in the middle. This will represent the X and Y axes upon which we will sketch the object, in this case a square. Now draw a large square with the cross at its centre. Number the corners of the square anti-clockwise, from 1 to 4, starting with the top right corner as 1.

We now need to work out the X, Y and Z co-ordinates of the square. Let' sassume for convenience that the length of each side of the square is 1000 units.

This makes the X and Y co-ordinates of point 1 . . . 500,500. As the object is flat (we are only drawing it in 2-D), the Z co-ordinate will be 0.

So the co-ordinates of point 1 are 500,500,0 (co-ordinates are always given in the order X, Y, Z).

Similarly those of point 2 are -500,500,0

Point 3's are -500,-500,0

Point 4's are 500,-500,0

We can now compose the data statements for the program. These will be inserted into the program on any line numbers up to 990 and will replace those in the program listed here on lines up to 990. The program requires information in the following format.

1. Number of 'corners' following by the X, Y and Z co-ordinate of each 'corner'.

2. Number of lines to be drawn, followed by the 'corner numbers' which they should join.

This may sound complicated but is in fact very straight-forward. The number of corners in our square is obviously 4 and we have worked out the co-ordinates already, so the first data statement will read:

```
70 DATA 4,500,500,500,-500,500,0,-500,-500,0,500,
   -500,0
```

The number of lines is also 4. If you look at the sketch you will see that we have numbered the corners from 1 to 4. The lines of the square go from point 1 to point 2, point 2 to point 3, point 3 to point 4, and point 4 to point 1. This is all that is required for the second data statement, which we can now write.

```
120 DATA 4,1,2,2,3,3,4,4,1
```

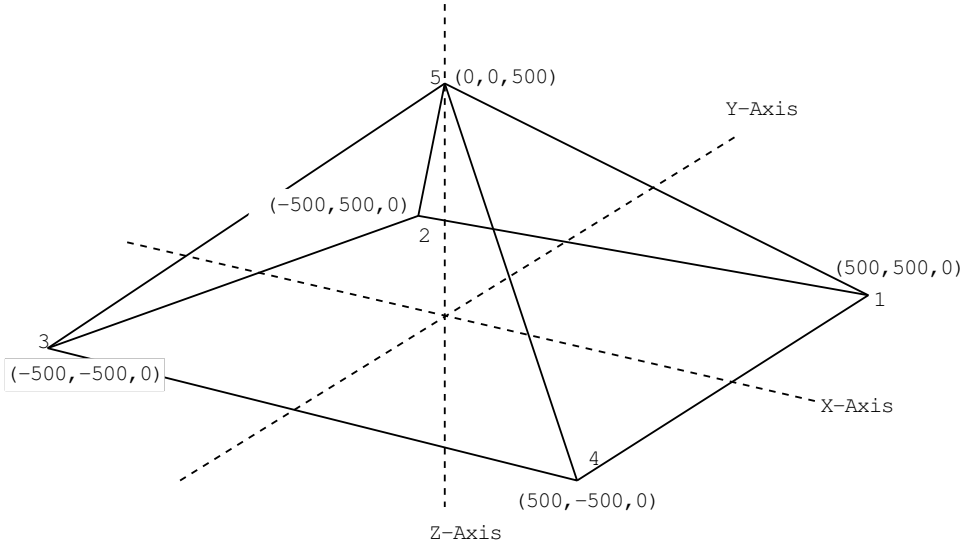
That's all there is to it. These two lines, 70 and 120, should replace the data statements in the program listed below on lines 70, 80 and 120. The actual line numbers are irrelevant as long as they are below 1000. Type them in and run the program, remembering to check that previous data statements, such as line 80, have been removed.

How to draw your own 3-D object

To do this we follow exactly the same process as above, now create an object with depth, eg. a pyramid. As this is drawn as an extension to a square, we will be able to use some of the co-ordinates calculated above.

Take the sketch of the square made earlier and consider the Z axis . This is at right angles to the other two axes and can be thought of as extending from above the paper, through the centre of the cross, to below the piece of paper. When calculating 3-D co-ordinates you have the choice of either imagining the points are not actually on the paper, or attempting to sketch them using perspective.

To continue with the pyramid, consider its apex, which we will define, for convenience, at a height of 500 units. This places it exactly over the intersection of the X and Y axes at a height of 500. Consequently its co-ordinates will be 0,0,500. If we number the apex as ' corner'5, you can see that it will require lines to be drawn joining it to ' corners'1,2,3 and 4. We now have an object with 5 ' corners'and 8 lines. The data statements can therefore be represented as follows:



```
 70  DATA 5,500,500,0,-500,500,0,
     -500,-500,0,500,-500,0,0,0,500
120  DATA 8,1,2,2,3,3,4,4,1,5,1,
     5,2,5,3,5,4
```

Type in these lines instead of all the other lines up to 990 and run the program.

If this doesn' tmake sense to you, compare them with the data statements calculated above for the square. Refer also to the picture of the pyramid accompanying this program in order to see how the axes would pass through it. The co-ordinates of the ' corners' are also indicated.

```
   10 REM 3-D Rotation
   20 REM by J.Hastings
   30 REM BEEBUG
   40 :
   50 REM Points data
   60 :
   70 DATA 8,-500,500,500,500,500,500,500,-500,500,
-500,-500,500
   80 DATA -500,500,-500,500,500,-500,500,-500,-500
,-500,-500,-500
   90 :
  100 REM Lines data
  110 :
  120 DATA 12,1,2,2,3,3,4,4,1,1,5,2,6,3,7,4,8,5,6,6
,7,7,8,8,5
  130 :
 1000 ON ERROR GOTO 1180
 1010 MODE 4
 1020 VDU 29,640;512;:VDU23,1,0;0;0;0;
 1030 *FX 4,1
 1040 PROCassembleCLG
 1050 PROCpoints
 1060 PROClines
 1070 distance%=5000
 1080 diststep%=500
 1090 anglestep=PI/16
 1100 REPEAT
```

```
 1110 PROC2D
 1120 PROCdraw
 1130 PROCupdate
 1140 PROCrotate
 1150 UNTIL FALSE
 1160 END
 1170 :
 1180 REM Error trap
 1190 MODE6
 1200 IF ERR<>17 REPORT: PRINT " at line "; ERL
 1210 *FX 4,0
 1220 END
 1230 :
 1240 DEF PROCassembleCLG
 1250 REM Fast CLG routine
 1260 DIM P% 25
 1270 [ OPT 2
 1280 .clg LDA #0
 1290 LDX #0
 1300 LDY #40
 1310 .loop STA &5800,X
 1320 INX
 1330 BNE loop
 1340 INC loop+2
 1350 DEY
 1360 BNE loop
 1370 LDA #&58
 1380 STA loop+2
 1390 RTS
 1400 ]
 1410 ENDPROC
 1420 :
 1430 DEF PROCpoints
 1440 REM Dimension point arrays and read in points
  data
 1450 READ points%
 1460 DIM X(points%),Y(points%),Z(points%),X2D(poin
ts%),Y2D(points%)
 1470 FOR count%=1 TO points%
 1480 READ X(count%),Y(count%),Z(count%)
 1490 NEXT count%
```

```
 1500 ENDPROC
 1510 :
 1520 DEF PROClines
 1530 REM Dimension line arrays and read in lines d
ata
 1540 READ lines%
 1550 DIM start%(lines%),end%(lines%)
 1560 FOR count%=1 TO lines%
 1570 READ start%(count%),end%(count%)
 1580 NEXT count%
 1590 ENDPROC
 1600 :
 1610 DEF PROC2D
 1620 REM Convert to 2-D
 1630 FOR count%=1 TO points%
 1640 X2D(count%)=X(count%)*2500/(distance%-Z(count
%))
 1650 Y2D(count%)=Y(count%)*2500/(distance%-Z(count
%))
 1660 NEXT count%
 1670 ENDPROC
 1680 :
 1690 DEF PROCdraw
 1700 CALLclg
 1710 FOR count%=1 TO lines%
 1720 MOVE X2D(start%(count%)),Y2D(start%(count%))
 1730 DRAW X2D(end%(count%)),Y2D(end%(count%))
 1740 NEXT count%
 1750 ENDPROC
 1760 :
 1770 DEF PROCupdate
 1780 phi=0: theta=0: psi=0
 1790 REPEAT
 1800 *FX 15,0
 1810 key=GET
 1820 UNTIL key=13 OR key=58 OR key=44 OR key=46 OR
 key=136 OR key=137 OR key=138 OR key=139
 1830 REM Rotate about X axis ?
 1840 IF key=138 THEN phi=anglestep
 1850 IF key=139 THEN phi=-anglestep
 1860 REM Rotate about Y axis ?
```

```
1870 IF key=136 THEN theta=anglestep
1880 IF key=137 THEN theta=-anglestep
1890 REM Rotate about Z axis ?
1900 IF key=13 THEN psi=-anglestep
1910 IF key=58 THEN psi=anglestep
1920 REM Change viewing distance ?
1930 IF key=44 THEN distance%=distance%+diststep%
1940 IF key=46 THEN distance%=distance%-diststep%
1950 ENDPROC
1960 :
1970 DEF PROCrotate
1980 IF phi<>0 THEN PROCXrotation
1990 IF theta<>0 THEN PROCYrotation
2000 IF psi<>0 THEN PROCZrotation
2010 ENDPROC
2020 :
2030 DEF PROCXrotation
2040 REM Rotate about X axis
2050 Cosphi=COS(phi): Sinphi=SIN(phi)
2060 FOR count%=1 TO points%
2070 Y=Y(count%): Z=Z(count%)
2080 Y(count%)=Y*Cosphi-Z*Sinphi
2090 Z(count%)=Z*Cosphi+Y*Sinphi
2100 NEXT count%
2110 ENDPROC
2120 :
2130 DEF PROCYrotation
2140 REM Rotate about Y axis
2150 Costheta=COS(theta): Sintheta=SIN(theta)
2160 FOR count%=1 TO points%
2170 X=X(count%): Z=Z(count%)
2180 X(count%)=X*Costheta-Z*Sintheta
2190 Z(count%)=Z*Costheta+X*Sintheta
2200 NEXT count%
2210 ENDPROC
2220 :
2230 DEF PROCZrotation
2240 REM Rotate about Z axis
2250 Cospsi=COS(psi): Sinpsi=SIN(psi)
2260 FOR count%=1 TO points%
2270 X=X(count%): Y=Y(count%)
```

```
2280 X(count%)=X*Cospsi-Y*Sinpsi
2290 Y(count%)=Y*Cospsi+X*Sinpsi
2300 NEXT count%
2310 ENDPROC
```