JAPLA 研究会資料 2017/5/20

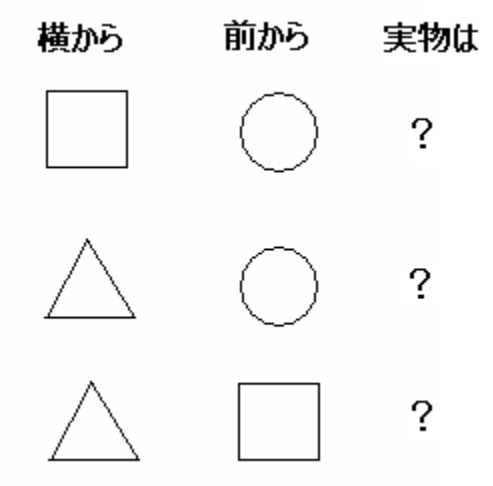
J-OpenGL による 3D-グラフィックスーその 13
ー３次元空間での錯視ー

西川　利男

はじめに

私がボランティアとして出ている日本科学未来館で、現在（5/20 まで）「錯視」というテーマをやっている。その中で、杉原厚吉先生による３次元空間での錯視のデモが好評で、来館者を楽しませてくれている。それに触発されて、３次元空間の幾何学の問題を J-OpenGL でプログラミングしてみた。

そこで、小手調べとしてあなたへの質問だが、前から見た形と横から見た形を元に、実際はどんな物体だか、頭に描くことができるだろうか？



１．３次元空間内の射影幾何学とは

世の中にあるすべての物体は、３次元空間内でその形が存在する。しかし、人間の目を通して網膜に写し出された像はあくまで２次元の平面である。それを視覚神経を通して、脳の働きにより奥行きを持った３次元の物体として認識できる。つまり、目で見てその形が分かるとは、目（＝センサー）と脳（＝データ処理）との協調作業である。

ユークリッド幾何学とは２次元平面の幾何学である。現実世界の３次元の物体が、2 次元の平面上にどう映し出されるのかを扱うのが射影幾何学である。しかし、ここで直ちに非ユークリッド幾何学なる数学者の jargon（＝専門用語）を持ち出すのは私は賛同できない。

２．OpenGL グラフィックスー現代の幾何学の道具として

数学の伝統的幾何学は、紙とえんぴつとを前提として発展してきた。しかし、現代の幾何

学はコンピュータ画面の上で行われる。

ところが、現実の３次元空間内の物体を実際に目で見たように描画するには、通常の２次元のグラフィックスだけでは不十分である。コンピュータのディスプレイ画面という制約のため、３次元内の物体が２次元の平面にどう映し出されるのか、つまり射影幾何学が必要になる。　難しい数学によらず、これにこたえるのが OpenGL グラフィックスである。

これまで J-OpenGL について、何回か報告してきたが、単に目を引く図形を画面上に描くというだけでなく、今回は３次元空間内の物体をしっかり観察するという意味での OpenGL の活用を行ってみた。

３．OpenGL を J の哲学から解する

OpenGL グラフィックスでは、球、円柱など定型的図形は glSphere, glCylinder などによりパラメータを入れるだけで容易に描ける。

しかし、現実世界の物体は数学的計算で得られる立方体や円柱、円錐などだけではなく、もっと自由自在な、例えば花びら、魚の形、おむすび、プロペラ…などいろいろある。　したがって真に役に立つツールとしては、これらに対応しなくてはならない。

一般の複雑な図形では、立体面をメッシュに区切って、つまり描画プリミティブと呼ぶ小さな図形片を貼り付けることにより、これを実現している。そして、glVertex で頂点座標を指定して、次のようなキーワードを持つ構文で立体面の描画を行っている。

GL_LINES

GL_TRIANGLES

GL_QUADS

なお、私の好きな J の哲学として名詞と動詞という考え方がある。

名詞...値、データ、オブジェクト

動詞...関数、処理、操作、メソッド、アプリ

この流儀では、OpenGL の哲学は「座標値なる名詞を、いろいろな動詞で操作する」となる。

名詞...頂点座標(X, Y, Z)

図形座標(X0, Y0, Z0);(X1, Y1, Z1); … (Xn, Yn, Zn)

動詞...図形の描画、回転、移動、合成、…　など

４．J-OpenGL プログラムのポイント

OpenGL プログラムのポイントは、頂点座標(X, Y, Z)の生成にある。今回は、立体図形として、はじめにあげたような、いろいろと変形した３種の円錐を用いた。

(1) 普通の直円錐(regular cone)

(2) 半分に切った直円錐(half cone) ＝ たてに切った筒の形

(3) 底面が傾いた円錐 (distorted cone) ＝ スリッパ形

プログラム全体のコードは最後に示すが、ポイントとなる部分を示す。

NB. generate interval values from(f) to(t) with (n) points =======

```
stepn =: 3 : 0
'f t n' =. y.
f + (t-f) * (i. >: n) % n
)
```

普通の円錐(regular cone)について、図形片の頂点座標を後述のJコードにより計算すると、たとえばつぎのように、座標値(x, y, z)の組として得られる。

```
   NX =: 6
   NY =: 6
   NZ =: 4
   NT =: 180
   4j1": L:0 P_XYZ
```

2.0 0.0 2.0	1.7 1.0 2.0	1.0 1.7 2.0	0.0 2.0 2.0	_1.0 1.7 2.0	_1.7 1.0 2.0	_2.0 0.0 2.0
1.8 0.0 1.8	1.5 0.9 1.8	0.9 1.5 1.8	0.0 1.8 1.8	_0.9 1.5 1.8	_1.5 0.9 1.8	_1.8 0.0 1.8
1.5 0.0 1.5	1.3 0.7 1.5	0.8 1.3 1.5	0.0 1.5 1.5	_0.7 1.3 1.5	_1.3 0.7 1.5	_1.5 0.0 1.5
1.3 0.0 1.3	1.1 0.6 1.3	0.6 1.1 1.3	0.0 1.3 1.3	_0.6 1.1 1.3	_1.1 0.6 1.3	_1.3 0.0 1.3

上のような座標値(x, y, z)の組P_XYZから、４つの頂点の座標値をもとに作った四角形の小片で敷き詰めて、立体全体を描く。その描画の動詞部分は次の通りである。

```
glBegin GL_QUADS
 i =. 0
 while. i < (NZ-1)
  do.
    j =. 0
    while. j < NX
      do.
       glVertex L:0 ( (i, j); ((i+1), j); ((i+1), j+1); (<i, j+1) ){P_XYZ
       end.
      j =. j + 1
      end.
  j =. 0
  i =. i + 1
```

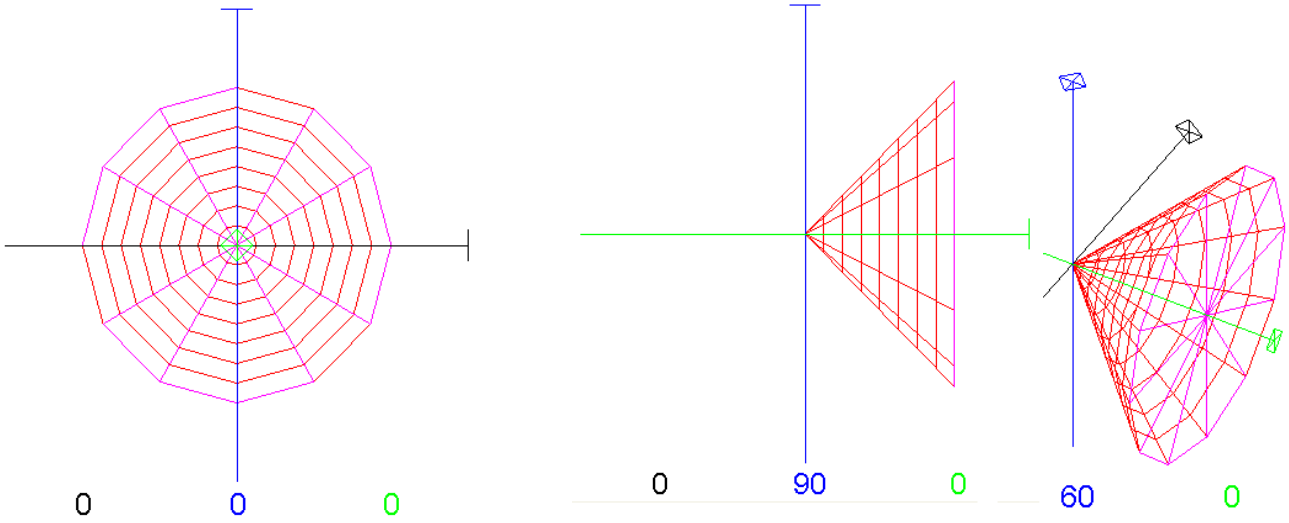
end.

以下、名詞としての頂点座標を入れ換えることで、いろいろな立体に対応できる。

５．いろいろな円錐立体を J-OpenGL グラフィックスによりさまざまな方向から観察する

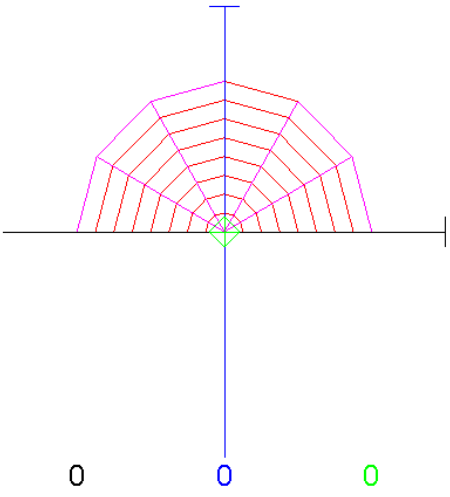
５． １ 普通の直円錐(regular cone)

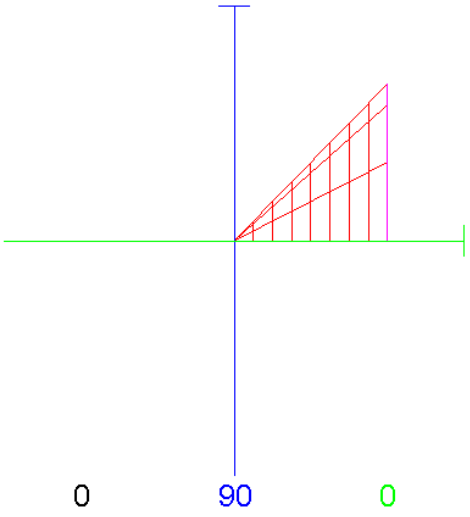
```
NB. regular cone ==========================
P_X =: cosd stepn 0, NT, NX
P_Y =: sind stepn 0, NT, NY
P_Z =: 2 - 0.25*i.NZ
P_XY =: P_X */L:0 <"(1) (P_X ,. P_Y)
P_XYZ0 =: P_Z */L:0 <"(1) (P_X,.P_Y)
P_XYZ1 =: P_XYZ0 ,"(1 0) L:0 P_Z
P_XYZ =: |: <"(1) > P_XYZ1
```

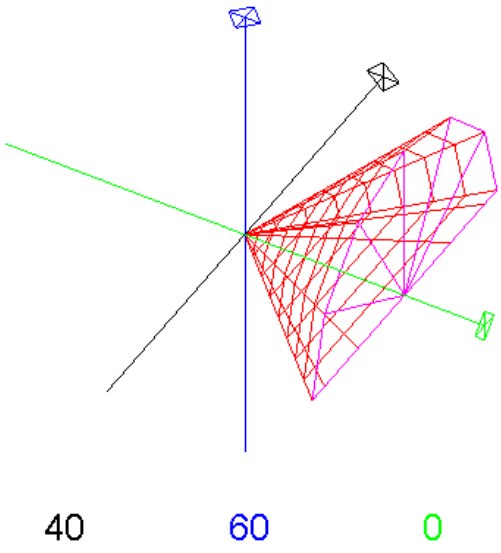


５．２ 半分に切った直円錐(half cone) ＝ たてに切った筒の半分の形

```
NB. half cone ==============================
Q_X =: cosd stepn 0, NT, NX
Q_Y =: (sind stepn 0, (-:NT), (-:NY)), (-:NY)#0
Q_Z =: 2 - 0.25*i.NZ
Q_XY =: Q_X */L:0 <"(1) (Q_X ,. Q_Y)
Q_XYZ0 =: Q_Z */L:0 <"(1) (Q_X,.Q_Y)
Q_XYZ1 =: Q_XYZ0 ,"(1 0) L:0 Q_Z
Q_XYZ =: |: <"(1) > Q_XYZ1
```

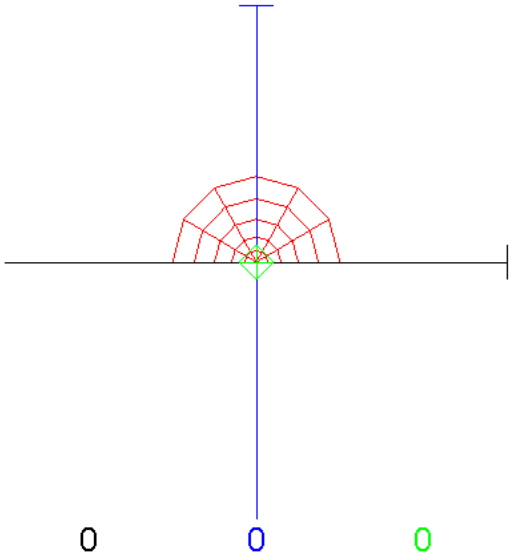


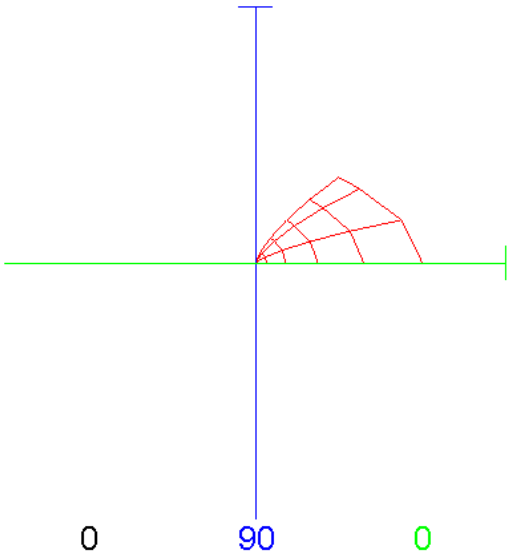




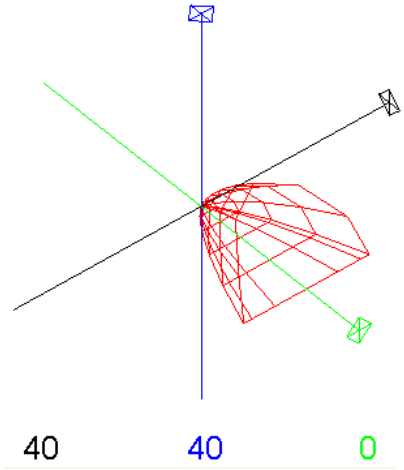
５．３ 底面が傾いた円錐(distorted cone) ＝ スリッパ形

```
NB. distorted cone ========================
R_X =: cosd stepn 0, NT, NX
R_Y =: (sind stepn 0, (-:NT), (-:NY)), (-:NY)#0
R_Z =: -: 2 - 0.25*i.NZ
R_XY =: R_X */L:0 <"(1) (R_X ,. R_Y)
R_XYZ0 =: R_Z */L:0 <"(1) (R_X,.R_Y)
R_XYZ1 =: R_XYZ0 ,"(1 0) L:0 R_Z
R_XYZ =: |: <"(1) > R_XYZ1
```





このように画面上で、
x, X, y, Y, z, Z のよ
それぞれX軸、Y軸、Z軸
回転し、対応する視点
面が現れる。錯視とは



キーコマンドとして、
うに打ち込むことで、
などの周りに図形が
から、いろいろな画
これに他ならない。

Jのプログラム・リスト

```
NB. OpGLN13.ijs
NB.  J-OpenGL その13 JAPLA 2017/5/20
NB. 2017/3/27, 5/20
NB. JOB=0 普通の円錐
NB. JOB=1 円錐の半分
NB. JOB=2 スリッパ形、円錐
NB. imported frm OpGLN2.ijs
NB. Cube Rotation, Solid/Wired, Hidden Line
NB. 酒井幸市「OpenGLでつくる３次元ＣＧ」,p.29-31
NB.   app3D1.ccp に相当のＪプログラム
NB. run '' => color cube
NB. run 0, 1 => line, hid cube; run 0, 0 => line, unhidden

require 'gl3'
require 'trig'

A=: 0 : 0
pc a closeok;
menupop "&Help";
menu help "&Help" "" "" "";
menupopz;
xywh 0 0 200 172;cc g isigraph ws_clipchildren ws_clipsiblings rightmove bottommove;
pas 0 0;
rem form end;
)

run=: a_run
a_run=: 3 : 0
if. 1 = # y.
 do.
   JOB =: y.
 else.
   JOB =: 0
 end.
```

```
wd :: ] 'psel a;pclose'
wd A
glaRC ''
R =: 0 0 0
glaFont 'arial 30'
NB. letter chacters enable 2017/5/13
glaUseFontBitmaps 0 32 96 32  NB. letter character, A, B, C,..a, b, c,.., {, |, }, ~
NB. glaUseFontBitmaps 0 32 26 32  NB. only numeric character, 0,.. 9
wd 'pshow;ptop'
)

NB. display the model picture ========================================
a_g_paint =: verb define
glClearColor 1 1 1 0
glClear GL_COLOR_BUFFER_BIT
if. JOB = 2
  do.
    drawtt ''
  else. drawp ''
end.
draw_xyz_axis ''
drawtext''
glaSwapBuffers ''
)

NB. cone as testing body ============================================
stepn =: 3 : 0
'f t n' =. y.
f + (t-f) * (i. >: n) % n
)

NX =: 12
NY =: 12
NZ =: 9
```

```
NT =: 360

NB. regular cone ==========================
pxyz =: 3 : 0
P_X =: cosd stepn 0, NT, NX
P_Y =: sind stepn 0, NT, NY
P_Z =: 2 - 0.25*i.NZ
P_XY =: P_X */L:0 <"(1) (P_X ,. P_Y)
P_XYZ0 =: P_Z */L:0 <"(1) (P_X,.P_Y)
P_XYZ1 =: P_XYZ0 ,"(1 0) L:0 P_Z
P_XYZ =: |: <"(1) > P_XYZ1
)
pxyz ''

NB. half cone ==============================
Q_X =: cosd stepn 0, NT, NX
Q_Y =: (sind stepn 0, (-:NT), (-:NY)), (-:NY)#0
Q_Z =: 2 - 0.25*i.NZ
Q_XY =: Q_X */L:0 <"(1) (Q_X ,. Q_Y)
Q_XYZ0 =: Q_Z */L:0 <"(1) (Q_X,.Q_Y)
Q_XYZ1 =: Q_XYZ0 ,"(1 0) L:0 Q_Z
Q_XYZ =: |: <"(1) > Q_XYZ1

NB. new version for distorted cone ==========================
NXt =: 24
NYt =: 24
txyz =: 3 : 0
R_X =: cosd stepn 0, NT, NX
R_Y =: (sind stepn 0, (-:NT), (-:NY)), (-:NY)#0
R_Z =: 2 - 2 * sind stepn 0, 180, NX
R_XY =: R_X */L:0 <"(1) (R_X ,. R_Y)

RR =: (1 + *: cosd stepn 0, 180, 6), (6#2)

T_XYZ0 =: -: (0{R_Z) * L:0 (R_X ,. R_Y) ,"(1 0) RR
T_XYZ1 =: -: (1{R_Z) * L:0 (R_X ,. R_Y) ,"(1 0) (1.75%2) * RR
```

```
T_XYZ2 =: -: (2{R_Z) * L:0 (R_X ,. R_Y) ,"(1 0) (1.5%2) * RR
T_XYZ3 =: -: (3{R_Z) * L:0 (R_X ,. R_Y) ,"(1 0) (1.25%2) * RR
T_XYZ4 =: -: (4{R_Z) * L:0 (R_X ,. R_Y) ,"(1 0) (1%2) * RR
T_XYZN =: T_XYZ0;T_XYZ1;T_XYZ2;T_XYZ3;T_XYZ4
T_XYZNN =: T_XYZN, <(13,3)$0
R_XYZ =: <"(1) >T_XYZNN
)
txyz ''

drawtt =: 3 : 0
glMatrixMode GL_MODELVIEW
glLoadIdentity ''
glTranslate 0 0 _1
glRotate R ,. 3 3 $ 1 0 0 0
glPolygonMode GL_FRONT_AND_BACK, GL_LINE   NB. Paint line

glColor 1 0 0 0
glBegin GL_QUADS
  i =. 0
  while. i < 5
    do.
  j =. 0
  while. j < NX
    do.
      glVertex L:0 ( (i, j); ((i+1), j); ((i+1), j+1); (<i, j+1) ) {R_XYZ
      j =. j + 1
    end.
  j =. 0
  i =. i + 1
  end.
glEnd ''
)

draw_xyz_axis =: 3 : 0  NB. draw x, y, z axes
glMatrixMode GL_MODELVIEW
glLoadIdentity ''
```

```
glTranslate 0 0 _1
glRotate R ,. 3 3 $ 1 0 0 0
glBegin GL_LINES
NB. X-axis Black
  glColor 0 0 0 0
  glVertex L:0 ((_3, 0, 0);(3, 0, 0))

  glVertex L:0 ((3, 0, _0.2);(3, 0, 0.2))
  glVertex L:0 ((3, _0.2, 0);(3, 0.2, 0))
  glVertex L:0 ((3, 0, _0.2);(3, _0.2, 0))
  glVertex L:0 ((3, 0, 0.2);(3, 0.2, 0))
  glVertex L:0 ((3, 0.2, 0);(3, 0, _0.2))
  glVertex L:0 ((3, _0.2, 0);(3, 0, 0.2))
NB, Y-axis Blue
  glColor 0 0 1 0
  glVertex L:0 ((0, _3, 0);(0, 3, 0))

  glVertex L:0 ((0, 3, _0.2);(0, 3, 0.2))
  glVertex L:0 ((_0.2, 3, 0);(0.2, 3, 0))
  glVertex L:0 ((0, 3, _0.2);(_0.2, 3, 0))
  glVertex L:0 ((0, 3, 0.2);(0.2, 3, 0))
  glVertex L:0 ((0.2, 3, 0);(0, 3, _0.2))
  glVertex L:0 ((_0.2, 3, 0);(0, 3, 0.2))
NB. Z-axis Green
  glColor 0 1 0 0
  glVertex L:0 ((0, 0, _3);(0, 0, 3))

  glVertex L:0 ((0, _0.2, 3);(0, 0.2, 3))
  glVertex L:0 ((_0.2, 0, 3);(0.2, 0, 3))
  glVertex L:0 ((0, _0.2, 3);(_0.2, 0, 3))
  glVertex L:0 ((0, 0.2, 3);(0.2, 0, 3))
  glVertex L:0 ((0.2, 0, 3);(0, _0.2, 3))
  glVertex L:0 ((_0.2, 0, 3);(0, 0.2, 3))
glEnd ''
)
```

```
NB. project the picture on the screen ===================
a_g_size =: verb define
wh =. glqwh ''
glViewport 0 0, wh
glMatrixMode GL_PROJECTION
glLoadIdentity ''
glOrtho _3.5 3.5 _3.5 3.5 _3.5 3.5
NB. gluPerspective 60, (%/wh), 1 30
)

NB. key-in x, y, z, X, Y, Z for rotation ================
a_g_char =: verb define
k =. 0 { sysdata
R =: 360 | R + 5 * 'xyz' = 0 { sysdata
R =: 360 | R - 5 * 'XYZ' = 0 { sysdata
NB. LS =: ('s' = k) { LS, -. LS
NB. Hid =: ('h' = k) { Hid, -. Hid
glpaintx''
)



NB. indicate rotated angle values x, y, z in degree ============
drawtext =: verb define
glMatrixMode GL_MODELVIEW
glLoadIdentity ''
glColor 0 0 0 0
glRasterPos _2.1 _3.4 0
NB. glCallLists 5 ": R
glCallLists (3 ": (0{R))
glColor 0 0 1 0
glRasterPos _0.3 _3.4 0
glCallLists (3 ": (1{R))
glColor 0 1 0 0
glRasterPos 1.7 _3.4 0
glCallLists (3 ": (2{R))
```

```
NB. letter chacters enable 2017/5/13
glColor 0 0 0 0
glRasterPos _2 _2.9 0
glCallLists (' x')
glColor 0 0 1 0
glRasterPos _0.2 _2.9 0
glCallLists (' y')
glColor 0 1 0 0
glRasterPos 1.8 _2.9 0
glCallLists (' z')
)



a_help_button =: verb define
wd 'mb OpenGL *Press keys, x/X, y/Y, z/Z rotate around each axis.'
wd 'setfocus g'
)
```