JAPLA 研究会資料　20014/3/15

# J-OpenGL による 3D グラフィックスーその 11
# 太鼓の膜振動－極座標で Bessel 関数を 3D 表示する

西川　利男

## はじめに

今年１月の JAPLA 例会では、中野先生からのユニークな年賀状に刺激されて、私なりに連分数と Bessel 関数について報告した。[1] これまで Bessel 関数など名前だけは聞いていたが、まさに特殊関数であり、自分には縁もないと思っていた。それが Bessel 関数とは太鼓の皮、つまり２次元の膜の振動を表す関数であり、それを J の plot ルーチンにより、グラフに描いてやっとイメージがつかめた。実はそのとき、OpenGL により３Ｄグラフィックスを描くつもりでいたが、残念ながら時間切れになってしまった。

これが元でその後、OpenGL で滑らかな曲面を描くというテーマに取り組んだ。その基本技術として、OpenGL での Bezier グラフィックスを取り上げたが、これは別稿とした。考えてみたら、太鼓の膜振動のグラフィックスには Bezier 処理は必要ないことが分かった。しかしながら、別の問題がいろいろあった。

２次元の膜の振動の理解にはニュートンの運動方程式ではなく、ラグランジュの解析力学が必要であり、かつ直交座標ではなく極座標で行わなくてはならない。一方、OpenGL プログラミングでも極座標での扱いは私にとっては初めての経験である。

## １．１次元の波、２次元の波、３次元の波－ニュートン力学から解析力学へ

はじめに物理の問題として少し考えてみる。[2],[3]

---

| １次元の波 | ２次元の波 | ３次元の波 |
|---|---|---|
| （ひもや糸の波） | （太鼓の皮の波） | （電磁波、量子力学の波など） |
| ニュートンの運動方程式 | ラグランジュの解析力学 | |

---

１次元の波とは、微小な質点が次々とつながって互いに力が作用する質点系の問題として、ニュートンの運動方程式を立てて、これから得られる２階の微分方程式を解いて、波の形 u(x, t)が求められる。ここで微分方程式は時刻 t を変数とするものと位置 x を変数とするものと、うまく分けられて（変数分離）、解が得られる。

ところが、２次元の質点系となるとそう簡単にはいかない。たとえば、太陽と地球のような２つの質点系の力学では、直交座標 x, y によるニュートンの運動方程式では変数分離は行えず、式は複雑になり手に負えなくなってしまう。

ラグランジュの運動方程式では、直交座標 x, y ではなく極座標 r, $\theta$ を用いることで変数分離が可能になる。かつ任意の座標を用いてより汎用的な力学となっている。

これが解析力学であり２次元、３次元の波の扱いが容易になる。さらにこの発展としてハミルトニアン力学は量子力学、宇宙論などへと現代の力学になった。

太鼓の膜振動はこのようにして解くことが出来て、ここに Bessel 関数が登場する。

[1] 西川利男「J を使った連分数の計算と Bessel 関数」JAPLA 研究会資料 2014/1/18
[2] スレーター、フランク、井上健訳「理論物理学入門」p.180-191, 岩波書店(1963).
[3] 竹内薫「ゼロから始める物理１，２，３」講談社(2003).

## ２．円形の膜振動の運動方程式とベッセル関数

円形の膜振動の運動方程式は時刻 $t$ と位置を極座標 $r, \theta$ で表したとき、つぎのようになる。[2]

$$\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial u}{\partial r})+\frac{1}{r^2}\frac{\partial^2 u}{\partial \theta^2}=\frac{1}{v^2}\frac{\partial^2 u}{\partial t^2}$$

このとき、波の関数を

$$u=R(r)\Theta(\theta)T(t)$$

とすると、変数分離ができて解くことが出来る。そして$R$については変数 r を x に置き換え、少し式変換をすると、つぎのベッセルの微分方程式となる。

$$\frac{1}{x}\frac{d}{dx}(x\frac{dR}{dx})+(1-\frac{m^2}{x^2})R=0$$

この解は

$$R=const\times J_m(x)$$

$m$次のベッセル関数として得られる。

したがって、膜の振動は次のようになる。

$$u=A(t)\cdot J_m(r)\cdot\cos(m\theta)$$

なお、ベッセル関数は前回の報告[1]にもあるように次のべき級数で求められる。

## ３．Ｊでベッセル関数の値を求める

ベッセル関数は、ガンマ関数を含んだ多項式展開の式として、いろいろな数学書に次のように与えられている。[1]

$$J_\nu(z)=(\frac{z}{2})^\nu\sum_{n=0}^{\infty}\frac{(-1)^n(z/2)^{2n}}{n!\Gamma(\nu+n+1)}$$

ここで、ガンマ関数はＪのプリミティブとして備えられている。したがって、Ｊでは上の式より多項べき級数の和として、直ちに求められる。つまりＪの環境ではベッセル関数の値は数表によらずとも計算により得られる。

Ｊによるベッセル関数の定義を示す。

```
Jn =: 3 : 0"(0)
:
N =. x.
R =. i. 60
J =. ((_1)^R) % ((! R)*(! <: N + R + 1))
x =. (0.5 * y.)^(N + 2 * R)
Jx =. J * x
+/ Jx
)
```

実行してみよう。

```
x = 0, 1, 2, ... ,9  に対する 0 次のベッセル関数の値 J0(x)はつぎのようになる。
   0 Jn i.10
1 0.765198 0.223891 _0.260052 _0.39715 _0.177597 0.150645 0.300079 0.171651
_0.0903336
x = 0, 1, 2, ... ,9  に対する 1 次のベッセル関数の値 J1(x)はつぎのようになる。
   1 Jn i.10
0 0.440051 0.576725 0.339059 _0.0660433 _0.327579 _0.276684 _0.00468282
0.234636 0.245312
```

## ４．J-OpenGL による 3D グラフィックス・プログラミング

Ｊの Windows の OpenGL の環境の構築方法については、前にも紹介して来た[4]。

[4] 西川利男「J-OpenGL による 3D グラフィックスーその 10
メビウスの帯へ向けてーJ-OpenGL をどう理解するか」JAPLA 研究会資料 2013/6/15

プログラミングといっても、OpenGL の決まった書式に従って体裁を整えればよい。

項目として挙げると、次のとおりである。

・require 'gl3'　　J OpenGL ライブラリを有効にする
・Windows を parent a として、グラフィック環境 child g とするフォームの生成
・プログラムの起動 a_run
・文字入力コマンド a_g_char
・画像射影条件　　a_g_size
・図形の作成　　a_g_paint

したがって、実際のプログラムは、上の図形の作成 a_g_paint の中にあって、表示する図形のための頂点座標(X, Y, Z)を作成準備することがＪのプログラミングの主な仕事となる。あとは OpenGL の書式で頂点座標(X, Y, Z)をつなぐだけでよい。

### ４．１　平面上の点の極形式データの生成

まず、平面のある範囲の点を極座標で値の組(r, th)として生成する。

とりあえず、次のような小さなデータでやってみる。

```
RA0 =: 0 + 0.5 * i.10              NB. Radius 0, 0.5, .. 4.5
TH0 =: 0 30 60 90                  NB. Angles 0 to 90
   RA0
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5
   TH0
0 30 60 90
   RA_TH =: { RA0 ; TH0
   RA_TH
+-----+------+------+------+
|0 0  |0 30  |0 60  |0 90  |
+-----+------+------+------+
|0.5 0|0.5 30|0.5 60|0.5 90|
+-----+------+------+------+
|1 0  |1 30  |1 60  |1 90  |
+-----+------+------+------+
|1.5 0|1.5 30|1.5 60|1.5 90|
+-----+------+------+------+
|2 0  |2 30  |2 60  |2 90  |
+-----+------+------+------+
|2.5 0|2.5 30|2.5 60|2.5 90|
+-----+------+------+------+
|3 0  |3 30  |3 60  |3 90  |
+-----+------+------+------+
|3.5 0|3.5 30|3.5 60|3.5 90|
+-----+------+------+------+
|4 0  |4 30  |4 60  |4 90  |
+-----+------+------+------+
|4.5 0|4.5 30|4.5 60|4.5 90|
+-----+------+------+------+
```

## ４．２　ベッセル関数値の挿入

上のボックスデータRA_TH内の最初の要素、距離rに対してベッセル関数J0の値を求めて、これを各ボックス内の最後の要素として追加してこれをzの値とする。

以下の定義で行う。

```
enter_J0 =: 3 : 0
'r th' =. y.
r, th, (0&Jn r)
)
```

```
   enter_J0 L:0 RA_TH
```

| | | | |
|---|---|---|---|
| 0 0 1 | 0 30 1 | 0 60 1 | 0 90 1 |
| 0.5 0 0.93847 | 0.5 30 0.93847 | 0.5 60 0.93847 | 0.5 90 0.93847 |
| 1 0 0.765198 | 1 30 0.765198 | 1 60 0.765198 | 1 90 0.765198 |
| 1.5 0 0.511828 | 1.5 30 0.511828 | 1.5 60 0.511828 | 1.5 90 0.511828 |
| 2 0 0.223891 | 2 30 0.223891 | 2 60 0.223891 | 2 90 0.223891 |
| 2.5 0 _0.0483838 | 2.5 30 _0.0483838 | 2.5 60 _0.0483838 | 2.5 90 _0.0483838 |
| 3 0 _0.260052 | 3 30 _0.260052 | 3 60 _0.260052 | 3 90 _0.260052 |
| 3.5 0 _0.380128 | 3.5 30 _0.380128 | 3.5 60 _0.380128 | 3.5 90 _0.380128 |
| 4 0 _0.39715 | 4 30 _0.39715 | 4 60 _0.39715 | 4 90 _0.39715 |
| 4.5 0 _0.320543 | 4.5 30 _0.320543 | 4.5 60 _0.320543 | 4.5 90 _0.320543 |

## ４．３　極形式から直交形式へ

OpenGLでは頂点座標は直交形式(x, y, z)で行われる。そのため極形式(r, th, z)から直交形式(x, y, z)への変換が必要である。以下の定義で行う。

```
polar2xy =: 3 : 0
'r th z' =. y.
(r * cosd th), (r * sind th), z
)
```

実行結果は４列のボックス配列になるが、最後の列ボックスは削除した。

```
   }:"(1)  polar2xy L:0 enter_J0 L:0 RA_TH
```

| | | |
|---|---|---|
| 0 0 1 | 0 0 1 | 0 0 1 |
| 0.5 0 0.93847 | 0.433013 0.25 0.93847 | 0.25 0.433013 0.93847 |
| 1 0 0.765198 | 0.866025 0.5 0.765198 | 0.5 0.866025 0.765198 |
| 1.5 0 0.511828 | 1.29904 0.75 0.511828 | 0.75 1.29904 0.511828 |
| 2 0 0.223891 | 1.73205 1 0.223891 | 1 1.73205 0.223891 |
| 2.5 0 _0.0483838 | 2.16506 1.25 _0.0483838 | 1.25 2.16506 _0.0483838 |
| 3 0 _0.260052 | 2.59808 1.5 _0.260052 | 1.5 2.59808 _0.260052 |
| 3.5 0 _0.380128 | 3.03109 1.75 _0.380128 | 1.75 3.03109 _0.380128 |
| 4 0 _0.39715 | 3.4641 2 _0.39715 | 2 3.4641 _0.39715 |
| 4.5 0 _0.320543 | 3.89711 2.25 _0.320543 | 2.25 3.89711 _0.320543 |

## ４．４　ベッセル線群の描画

上のようにしてボックス配列として出来たベッセル値から頂点群 VJ0 を作る。
ここでは、実際に合わせてもっと多くの(r, th)値について行った。

```
RA =: 0 + 0.1 * i.120
TH =: 30 * i.12
R_T =: { RA ; TH
VXYJ0 =: enter_J0 R_T
VJ0 =: polar2xy L:0 VXYJ0
```

これらの頂点値を使って、OpenGL 命令でベッセル線群の図形を描く。

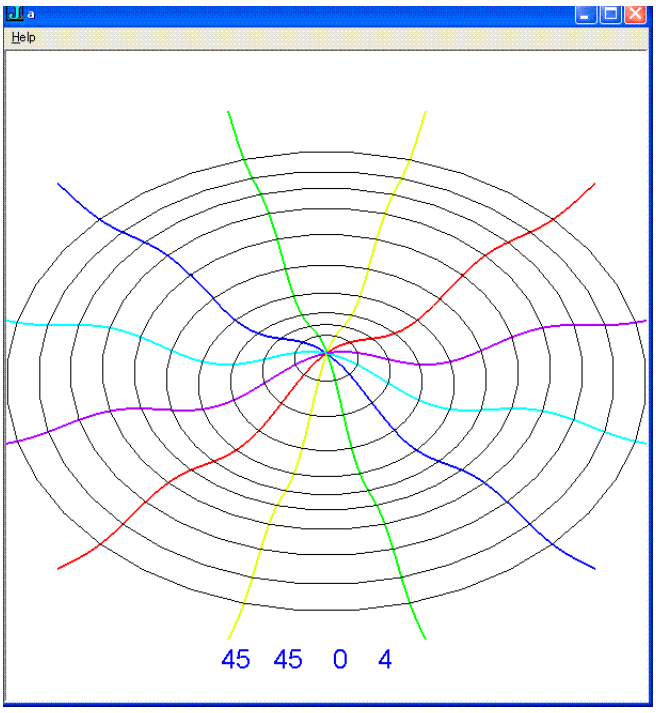
```
dColor =: 1 0 0 1;0.9 1 0 1;0 1 0 1;0 0 1 1;0 1 1 1;0.7 0 1 1

drawSurface =: 3 : 0
i =. 0
while. i < }. $ y.
  do.
    glBegin GL_LINE_STRIP
    glColor > (6|i){dColor
    glVertex (L:0) i {"(1) y.
    glEnd ''
    i =. i + 1
  end.
)

drawBes =: 3 : 0
glClearColor 1 1 1 1
glClear GL_COLOR_BUFFER_BIT + GL_DEPTH_BUFFER_BIT
glColor 0 0 1 1
glMatrixMode GL_MODELVIEW     NB. enable rotate
glLoadIdentity''              NB. enable rotate
glRotate R ,. 3 3 $ 1 0 0 0   NB. enable rotate
glLineWidth 2
drawSurface VJ0               NB. draw J0 curve
glLineWidth 1
drawScaledCircle SJ0          NB. draw scaled circle
)
```

プログラム drawBes が図形の描画を行い、サブプログラムとして drawSurface を呼び、また、　dColor による色表示を行っている。なお、図形を極座標のメッシュで表すため、ある動径に対して同心円を描く定義 drawScaledCircle を用いた。

## ５．ベッセル関数－3D グラフィックスの実際

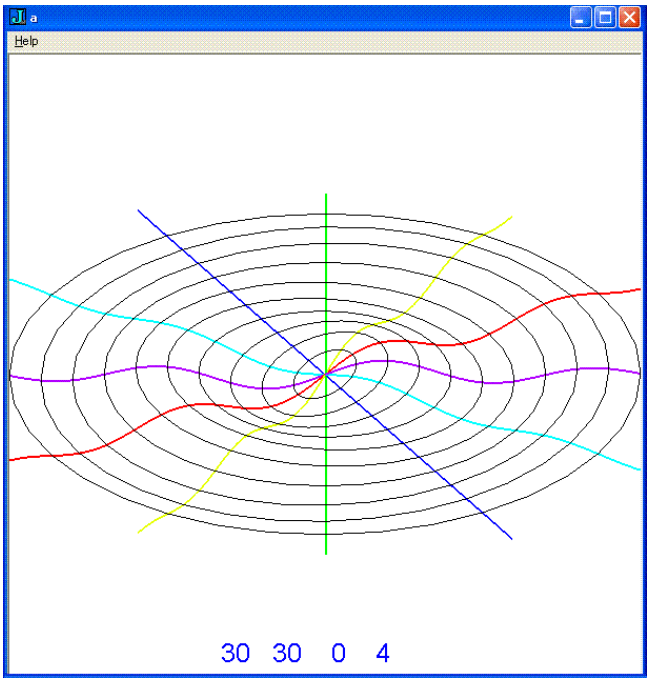
### ５．１　０次のベッセル関数の 3D グラフィックスの実行

run 0 または run 1 として実行。キーコマンド x, X, y, Y, z, Z によりさまざまな方向から視点を変えて見ることができる。




### ５．２　１次のベッセル関数の 3D グラフィックスの実行

run 2 として実行。原点(0, 0)を通り動径の逆方向では、ベッセル関数の正負が変わり、凹凸が上下、逆になることが見られるであろう。



```
NB. OpGLN_Bessel.ijs 2013/2/12
NB. imported from opengl¥opgl_surface.ijs
NB. imported from Bessel.ijs

NB. Bessel Function by T.N 2014/1/3 ===============

require 'trig'

NB. e.g.
NB.  J0 i. 5 => 1 0.765198 0.223891 _0.260052 _0.39715

J0 =: 3 : 0"(0)
r =. i. 60
J =. ((_1)^r) % (*: ! r)
x =. (0.5 * y.)^(2*r)
Jx =. J * x
+/ Jx
)

J1 =: 3 : 0"(0)
r =. i. 60
J =. ((_1)^r) % ((! r)*(! >:r))
x =. (0.5 * y.)^(1 + 2*r)
Jx =. J * x
+/ Jx
)

NB. e.g. 0&Jn i.5 => 1 0.765198 0.223891 _0.260052 _0.39715
Jn =: 3 : 0"(0)
:
N =. x.
r =. i. 60
J =. ((_1)^r) % ((! r)*(! <: N + r + 1))
x =. (0.5 * y.)^(N + 2 * r)
Jx =. J * x
+/ Jx
)

In =: 3 : 0"(0)
:
N =. x.
M =. i. 100
J =. 1 % ((! M)*(! <: N + M + 1))
x =. (0.5 * y.)^(N + 2 * M)
Jx =. J * x
+/ Jx
)

NB. Bessel Calculation / Revised 2013/2/10
=======================================
RA0 =: 0 + 0.5 * i.10              NB. Radius 0, 0.5, .. 4.5
TH0 =: 0 30 60 90 120 150          NB. Angles 0 to 150
R_T0 =: { RA0 ; TH0

RA =: 0 + 0.1 * i.120              NB. Radius 0, 0.1, .. 11.9
TH =: 30 * i.12                    NB. Angles 0 to 330
R_T =:  { RA ; TH
```

```
NB. Bessel J0 values calculated, then enter as Z values
enter_J0 =: 3 : 0
'r th' =. y.
r, th, (J0 r)
)
```

NB. VJ0 = (X, Y, Z)
VXYJ0 =: enter_J0 L:0 R_T  NB. Radius, Angle, J0

```
NB. Bessel J0 values calculated, then enter as Z values 2013/2/16
enter_J1 =: 3 : 0
'r th' =. y.
r, th, (cosd th)*(J1 r)
)
```

VXYJ1 =: enter_J1 L:0 R_T

```
NB. convert polar 'r th Z' to rectangular 'X Y Z' ====================


polar2xy =: 3 : 0
'r th z' =. y.
(r * cosd th), (r * sind th), z
)
```

NB. Bessel Values: X, Y, Z(=J0)
VJ0 =: polar2xy L:0 VXYJ0  NB. X, Y, Z(=J0)
NB. OpenGL Graphics => testVJ0 VJ0
VJ1 =: polar2xy L:0 VXYJ1

```
NB. Bessel Curve Z values upward
swapYZ =: 3 : '((_1, 1)*(1,2){y.) (2,1) } y.'
WJ0 =: swapYZ L:0 VJ0
WJ1 =: swapYZ L:0 VJ1



NB. Scaled Circle 2013/2/16 ==================================
RA1 =: 1 + i. 10
TH1 =: 10 * i.37
S_RT =: { RA1 ; TH1
SXYJ0 =: enter_J0 L:0 S_RT   NB. for J0
SJ0 =: polar2xy L:0 SXYJ0

TJ0 =: swapYZ L:0 SJ0

NB. RA1 =: 1 + i. 10
NB. TH1 =: 10 * i.37
NB. S_RT =: { RA1 ; TH1
SXYJ1 =: enter_J1 L:0 S_RT   NB. for J1
SJ1 =: polar2xy L:0 SXYJ1

TJ1 =: swapYZ L:0 SJ1


NB. OpGL Graphics ============================================
SIZE =: 0.5
```

```
require 'gl3'

A=: noun define
pc a closeok;
menupop "&Help";
menu help "&Help" "" "" "";
menupopz;
xywh 0 0 300 270;cc g isigraph ws_clipchildren ws_clipsiblings rightmove
bottommove;
pas 0 0;
rem form end;
)

NB. run 0 => JOB = 0, VP1, plane = X-Z frame, height = Z-axis
NB. run 1 => JOB = 1, VP0, plane = X-Y frame, height = Y-axis
NB. run 2 => OpenGLプログラミングガイド p.448, J-Studio/Demo/surface と同じ
NB. key command s => fine, a = > coarse
run =: a_run
a_run=: verb define
wd A
JOB =: y.
if. 0 = # JOB do. JOB =: 0 end.
SEL =: 0    NB. opgl_light, SEL =: 1 nmap
NB. R =: 85 60 0
R =: 0 0 0
SIZE =: 1
glaRC''
FILL =: 0
STEPS =:4
glaFont 'arial 30'
glaUseFontBitmaps 0 32 26 32
wd 'pshow;ptop'
)

a_g_char =: verb define
k=.0{sysdata
R =: 360 | R + 5 * 'xyz' = k
R =: 360 | R - 5 * 'XYZ' = k
SIZE =: ('l'=k) { SIZE, 1.5*SIZE
STEPS =: 100 <. STEPS + 's' = k
STEPS =: 0 >. STEPS - 'a' = k
FILL =: ('f'=k) { FILL,-.FILL
JOB =: ('j'=k) { JOB, 1+JOB
SEL =: ('n'=k) { SEL,-.SEL
glpaintx''
)

a_g_size=:verb define
wh=.glqwh''
glViewport 0 0,wh
glMatrixMode GL_PROJECTION
glLoadIdentity''
NB. glOrtho _5 5,(_5 5*%%/wh), _5 5
NB. glOrtho _5 5, _5 5, _5 5
glOrtho _10 10, _10 10, _10 10
return.
if. </wh do.
 glOrtho _5 5,(_5 5*%%/wh), _5 5
else.
 glOrtho (_5 5*%/wh), _5 5 _5 5
```

```
end.
)

a_g_paint =: verb define
NB. nmap VP1
NB. testmap nmap VP1
NB. test VJ
if. SEL = 0
 do.
  light''
  select. JOB
    case. 0  do. STEPS =: 3
                 surface VP1            NB. surface J original codes version
                 plotctrlpt VP1
    case. 1  do. STEPS =: 3
                 surface VP0
                 plotctrlpt VP0
    case. 2  do. surface b2dpts
                 plotctrlpt b2dpts
    case. 3  do. nsurface nmap VP1       NB. surface T.N programmed codes
version
    case. 4  do. test nmap  VP1
    case. 5  do. testmap nmap    b2dpts
    case. 10 do. nsurface , > VJ44A
    case. 11 do. nsurface , > |."(1) (2&*) L:0 (_0.5&+) L:0 VJ44A
    case. 12 do. nsurface , > JXYZ
    case. 99 do. testVU ''
    case. 101;102 do. drawBes ''
    case. 111;112 do. drawBes ''
NB.    case. 101 do. drawJ0 VJ0  NB. see onto X(right)-Y(up) plane
NB.    case. 102 do. drawJ0 WJ0  NB. see onto X(right)-Z(up) plane, Y <- _Z,
Z <- Y
  end.
 else.
  select. 3 | JOB
    case. 3  do. nmap    VP1       NB. surface T.N programmed codes version
    case. 4  do. nmap    VP0
    case. 5  do. nmap    b2dpts
  end.
end.
NB. plotctrlpt b2dpts
drawtext ''
glaSwapBuffers ''
)




NB. draw Bessel Curve ========================================
NB. drawJ0 VJ0,  drawJ0 WJ0

dColor =: 1 0 0 1;0.9 1 0 1;0 1 0 1;0 0 1 1;0 1 1 1;0.7 0 1 1

drawSurface =: 3 : 0
i =. 0
while. i < }. $ y.
  do.
```

```
    glBegin GL_LINE_STRIP
    glColor > (6|i){dColor
    glVertex (L:0) i {"(1) y.
    glEnd ''
    i =. i + 1
  end.
)

drawScaledCircle =: 3 : 0
i =. 0
while. i < # y.
  do.
    glBegin GL_LINE_STRIP
    glColor 0 0 0 1
    glVertex (L:0) i{ y.
    glEnd ''
    i =. i + 1
  end.
)

NB. draw Bessel curve J0, J1
NB. using 'JOB' as global parameter
drawBes =: 3 : 0
glClearColor 1 1 1 1
glClear GL_COLOR_BUFFER_BIT + GL_DEPTH_BUFFER_BIT
glColor 0 0 1 1
glMatrixMode GL_MODELVIEW     NB. enable rotate
glLoadIdentity''              NB. enable rotate
glRotate R ,. 3 3 $ 1 0 0 0   NB. enable rotate
select. JOB
  case. 101 do.
    glLineWidth 2
    drawSurface VJ0               NB. draw J0 curve
    glLineWidth 1
    drawScaledCircle SJ0          NB. draw Scaled Circle
  case. 102 do.
    glLineWidth 2
    drawSurface WJ0               NB. draw J0 curve Bessel up
    glLineWidth 1
    drawScaledCircle TJ0          NB. draw Scaled Circle
  case. 111 do.
    glLineWidth 2
    drawSurface VJ1               NB. draw J1 curve
    glLineWidth 1
    drawScaledCircle SJ1
  case. 112 do.
    glLineWidth 2
    drawSurface WJ1               NB. draw J1 curve Bessel up
    glLineWidth 1
    drawScaledCircle TJ1
end.
)
```