JAPLA 研究会資料　2009/9/26

Ｊの gl3-OpenGL によるグラフィックスーその１

西川　利男

０．はじめに

先月、JAPLA の蓼科合宿で「花のグラフィックスーチューリップ、朝顔」を発表した[1]。これを機会に３Ｄグラフィックスへの興味を大いに刺激され、さらにその必要性を感じた。

Ｊには３Ｄグラフィックス処理のための gl3 なるプリミティブ命令群があるが、ここでは OpenGL ライクの書式でプログラミングを行なう。そのチュートリアルとして、Ｊの[Studio]-[Lab]の中に、OpenGL Programming の紹介があるが、英文でもあり必ずしも分かりやすいとは言えない。今回、二、三の OpenGL の本[2]なども参考にしつつ、簡単な３Ｄグラフィックスの処理を例として、OpenGL そのもの、および gl3-OpenGL の基本のプログラミングについて解説する。なお、gl3 は、これまで筆者が多用してきた２次元用の gl2 とはかなり違う、ほとんど別物であることも分かった。

１．OpenGL とは

OpenGL(=Open Graphic Library)とは元来、米国シリコングラフィックス社の３次元グラフィックスライブラリ(IRIS.GL)を機種を問わず使用できるようにした仕様で、今や実質的には、３Ｄグラフィックスの標準の地位をしめつつある。技術的にはＣ言語をベースとして、２Ｄ、３Ｄのグラフィックスライブラリを使用するプログラミング書式である。

Ｊの OpenGL にはこの書式にならって、次の３種類の命令群がある。

・glTranslate, glRotate など…OpenGL と同じ書式
・glaRC, glaFont など……Ｊの独自のもの。
・gluSphere, gluCylinder など…基本の図形などユーティリティ
・GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES など…グローバル名詞

本来の OpenGL では例えば関数の呼び出しでＣ言語の書式のため、引数などカッコでくくるところが、Ｊの OpenGL-gl3 ではスペースで区切るなど当然の違いもある.
しかし、ほとんどの命令は本来の OpenGL の書式のままで使われる。

なお、これらの命令群は、system¥main¥gl3 の中に 11!:3000+n のプリミティブとして定義されている。

２．OpenGL の考え方ーモデリングとレンダーリング

OpenGL では基本的に次の２つのステップで考える

・モデリング(modeling)　物体の３次元の位置座標値をどう求めるか。
・レンダーリング(rendering)　コンピュータの画面上にどう投影して表示するか。

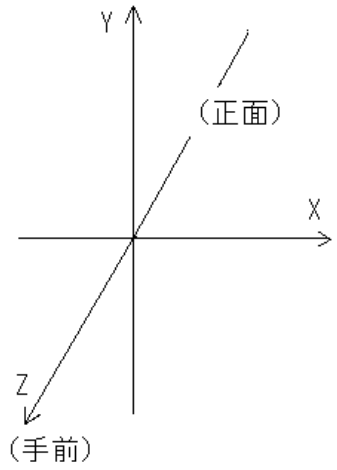
---

[1] 西川利男「Ｊによる花のグラフィックスー１(チューリップ), ２(朝顔)」
JAPLA 研究会資料, 2009/8/5 (夏の蓼科合宿).
[2] 酒井幸市「OpenGL でつくる３次元 CG とアニメーション」森北出版(2008).
ポール・マーツ、松田晃一ら訳「OpenGL の神髄」ピアソン・エデュケーション (2007).

３．OpenGL の座標系と座標値

OpenGL では右手系の座標系を用いる。そして以下のような X 軸, Y 軸, Z 軸の方向を原則とする。つまり正面が X-Y 面で、右が X の正値、上が Y の正値、手前が Z の正値となる。

３次元物体を表す点は、この座標系で(X,Y,Z)として、普通は　_1 から 1 の範囲の小数点数の値で示す。また、角度はラジアンでなく 0 から 360 の度であらわす。



このような座標系でモデルとなる物体の(X,Y,Z)値を決めたのち、移動、回転、スケーリングなどの操作を行う。その後、視点を定めたてコンピュータの画面上に投影して表示する。この後半の操作がレンダーリングと言われる。

４．OpenGL グラフィックスのプログラミングの工程

(1) モデリング(modeling)

物体の幾何学的位置の数値取得であるが、いろいろな方法が取られる。

・数値を並べる
・計算して得る
・ツールを使って得る

まず「数値を並べる」方法で、例えば３次元の３角形の板のモデルを作ってみよう。３角形の上方の頂点 VB は底辺の左の頂点 VA および右の頂点 VC より奥にある。

このとき OpenGL の独特の手法がとられる。

```
VA =: _1  0  0
VB =:  0  1 _1
VC =:  1  0  0
```

とした上で

```
glBegin GL_TRIANGLES
  glVertex VA
  glVertex VB
  glVertex VC
glEnd ''
```

とする。

「計算による」方法では、Jの普通のプログラミングにより、各頂点の(X,Y,Z)を求めればよい。「ツールを使う」方法ではgluSphere, gluCylinderなどそれぞれの書式にしたがってパラメータをきめる。

(2) 物体の移動、回転の操作と行列計算処理

移動、回転、スケーリングにはいわゆるアフィン変換の行列計算が必要だが、OpenGLではあらわに行列計算をプログラムで行なう必要はない。移動、回転、スケーリングの命令が備えられていて、それを使って次のように行なう。

・行列モードの設定（それぞれ場合に応じて使いわける）

モデリングの場合

```
glMatrixMode GL_MODELVIEW
glLoadIdentity ''
```

投影の場合

```
glMatrixMode GL_PROJECTION
glLoadIdenty ''
```

・平行移動　X方向にtX, Y方向にtY, Z方向にtZに物体を移動

```
glTranslate tX, tY, tZ
```

・回転　X軸のまわりにrX°,Y軸のまわりにrY°,Z軸のまわりにrZ° の回転

```
glRotate RXYZ
```

ここで

$$RXYZ = \begin{pmatrix} rX & 1 & 0 & 0 \\ rY & 0 & 1 & 0 \\ rZ & 0 & 0 & 1 \end{pmatrix}$$

・スケーリング　X方向でsX, Y方向でsY, Z方向でsZだけ物体を拡大、縮小

```
glScale sX, sY, sZ
```

(3) 投影と画面表示

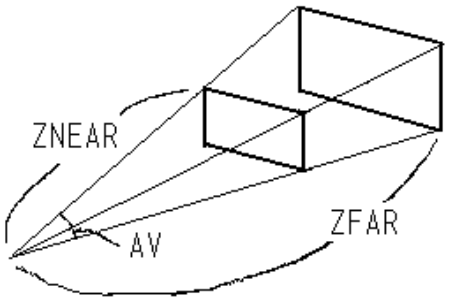
３次元の物体は近くのものは大きく、遠くのものは小さく見える。この機能を行なうのが投影である。

・視点投影の画面表示

```
glPerspective AV, RWH, ZNEAR, ZFAR
```

ここで

```
AV:   Angle of View
RWH:  Ratio of W and H
ZNEAR: Z to Near Point
ZFAR:  Z to Far Point
```



・平行投影の画面表示

```
glOrtho _pX, pX, _pY, pY, _pZ, pZ
```

この場合は各X, Y, Zの直方体で区切られた内部の物体を
無限遠の視点に対応した平行投影で表示する。

５．gl3-OpenGL のＪプログラムの実際(OpGLN0.ijs)

先の「３次元空間内にある三角形の板を表示して、大きさを変えたり、回転したりしてみる」という問題のＪプログラムを作り、コーディングに沿って解説する。

```
NB. OpGLN0.ijs
NB. test for gl3 & openGL

require 'gl3'

A=: 0 : 0
pc a closeok;
xywh 0 0 200 200;cc g isigraph ws_clipchildren ws_clipsiblings rightmove
bottommove;
pas 0 0;
rem form end;
)

run=: a_run
a_run=: 3 : 0
wd A
glaRC ''
R =: 0 0 0
S =: 1
glaFont 'arial 30'
glaUseFontBitmaps 0 32 26 32
wd 'pshow;ptop'
)
```

これらはウィンドウの設定である。次の命令により OpenGL が起動される。

```
glaRC ''
```

同時に isigraph に以下の設定が必要である。

```
ws_clipchildren ws_clipsiblings
```

起動とともに、以下の２つのイベントは、自動的に実行される。

```
a_g_paint
```

では、モデルの作成が行なわれる。

```
a_g_size
```

により、投影表示がなされる。

```
NB. display the model picture ========================================
a_g_paint =: verb define
glClearColor 1 1 1 0
glClear GL_COLOR_BUFFER_BIT
```

```
drawtriangle ''         NB. ３角形を描く
drawtext''              NB. 回転の角度の表示
glaSwapBuffers ''       NB. 画像表示を更新する
)
NB. モデルとしての２つの三角形の頂点の値データ
VA =: _1  0  0
VB =:  0  1 _1
VC =:  1  0  0
VD =:  0 _1  0
NB. 頂点を結合して三角形を作る
drawtriangle =: verb define
glMatrixMode GL_MODELVIEW
glLoadIdentity ''
glTranslate 0 0 _1              NB. 移動
glRotate R ,. 3 3 $ 1 0 0 0     NB. 回転
glScale S, S, S                 NB. スケーリング
glPolygonMode GL_FRONT_AND_BACK, GL_LINE      NB. 輪郭線のみを描く
NB. glPolygonMode GL_FRONT_AND_BACK, GL_FILL  NB. 輪郭の内部を塗りつぶす
glBegin GL_TRIANGLES            NB. 三角形を作る
  glColor 1 0 0 0               NB. 赤
  glVertex VA
  glVertex VB
  glVertex VC
  glColor 0 0 1 0               NB. 青
  glVertex VA
  glVertex VD
  glVertex VC
glEnd ''
)
```

以下は投影、画面表示の設定であり、平行投影で行なっている。

```
NB. project the picture on the screen ====================
a_g_size =: verb define
glViewport 0 0  , glqwh ''
glMatrixMode GL_PROJECTION
glLoadIdentity ''
glOrtho _2.5 2.5 _2.5 2.5 _2.5 2.5
)
```

キー入力により、回転、サイズの変更を行なう。
その結果はglaSwapBuffers ''に更新される。

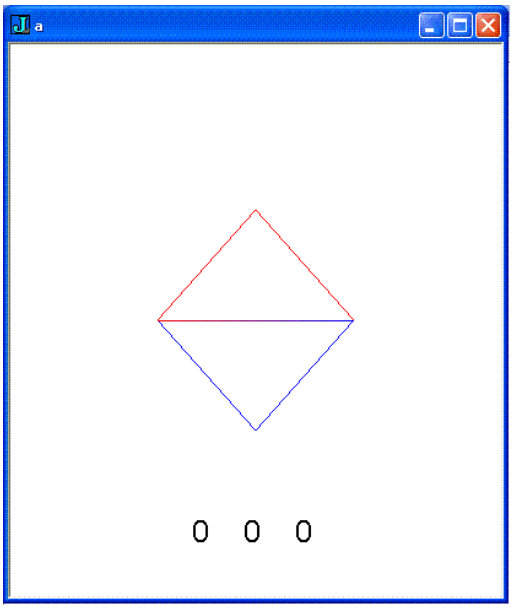
```
NB. key-in x, y, z, X, Y, Z for rotation ================
NB.         L for larger, S for smaller
a_g_char =: verb define
R =: 360 | R + 5 * 'xyz' = 0 { sysdata
```

```
R =: 360 | R - 5 * 'XYZ' = 0 { sysdata
S =: S + 0.5 * 'L' = 0 { sysdata
S =: S - 0.5 * 'S' = 0 { sysdata
glpaintx''
)
X軸, Y軸, Z軸のまわりの回転の角度の値を表示する。
NB. indicate rotated angle values x, y, z in degree ============
drawtext =: verb define
glMatrixMode GL_MODELVIEW
glLoadIdentity ''
glColor 0 0 0 0
glRasterPos _1 _2 0
glCallLists 5 ": R
)
```

６．gl3-OpenGL のＪプログラムの実行例

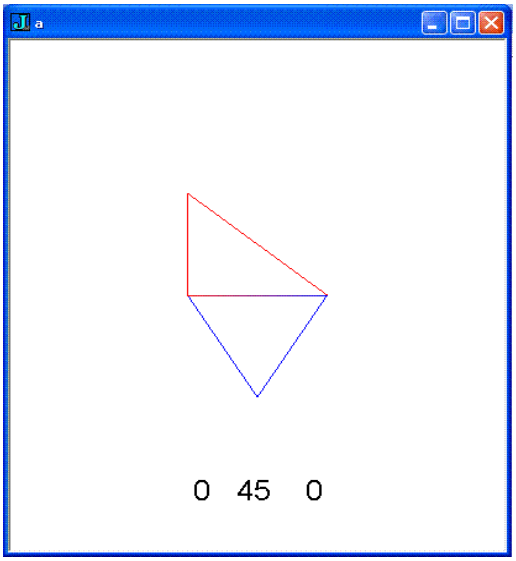
実行すると、右図のように２つの３角形から成る板を表示する。

なお、a_g_paint で
glPolygonMode GL_FRONT_AND_BACK, GL_FILL
とすると、内部を塗りつぶした３角形を描く。



ためしに、キーボードから'Y'を次々と打ち込むと、Y軸のまわりに右向きに回転し、45° では右図のようになる。

つまり、初めの２つの３角形板をななめに見たので、下の３角形の下の頂点の位置は変わらないのに、上の３角形の上の頂点は移動し、奥にあることがわかる。



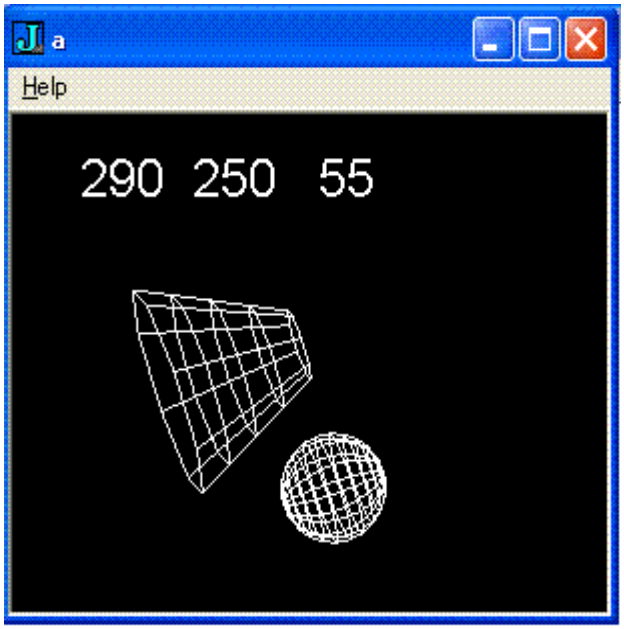
７．モデル・ツールと glCallList, glNewList の利用(OpGLN1.ijs)

いくつかのモデル物体についてはあらかじめ定義されている。また、まとめて処理を行う glCallList, glNewList の機能を利用するコンパクトにコーディングできる。

その部分だけのプログラムを示す。

```
a_g_paint =: verb define
glClearColor 0 0 0 0
glClear GL_COLOR_BUFFER_BIT + GL_DEPTH_BUFFER_BIT
glEnable GL_DEPTH_TEST
glMatrixMode GL_MODELVIEW
glLoadIdentity''
glTranslate 0 0 _10
glRotate R ,. 3 3 $ 1 0 0 0
glCallList WIRES                    NB. NewList の呼び出し
drawtext ''
glaSwapBuffers ''
)

wires=: verb define                 NB. NewList の定義
obj=.gluNewQuadric''
glNewList WIRES,GL_COMPILE
 gluQuadricDrawStyle obj,GLU_LINE
 gluSphere obj, 3 12 12             NB. 球: radius 3, nSlice 12, nStack 12
 glTranslate 10 0 0
 gluCylinder obj, 2 6 8 8 4         NB. 円錐台: rBottom 2, rTop 6, Height 8,
glEndList''                         NB.          nSlice 4, nStack 4
gluDeleteQuadric obj
)
```



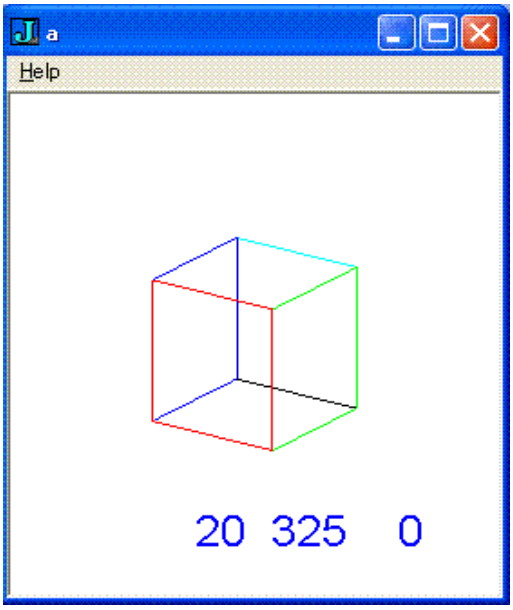
８．立方体（サイコロ）の回転と陰線処理の例(OpGLN2.ijs)

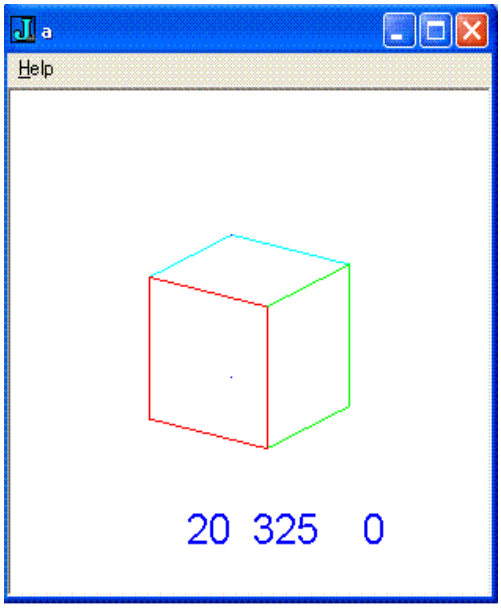
```
NB. make cubic as a model ==================================
NB. Vertex Values
P =: ] ;._2 (0 : 0)
      0.5   0.5   0.5
     _0.5   0.5   0.5
     _0.5  _0.5   0.5
      0.5  _0.5   0.5
      0.5   0.5  _0.5
     _0.5   0.5  _0.5
     _0.5  _0.5  _0.5
      0.5  _0.5  _0.5
)
P =: ". P
P =: 1.5 * P

NB. Draw Vertex
draw =: verb define
glMatrixMode GL_MODELVIEW
glLoadIdentity ''
glTranslate 0 0 _1
glRotate R ,. 3 3 $ 1 0 0 0
if. LS = 0
  do.
    glPolygonMode GL_FRONT, GL_LINE   NB. Paint line
  else.
    glPolygonMode GL_FRONT_AND_BACK, GL_FILL   NB. Paint full
end.
glPolygonMode GL_BACK, Hid{GL_LINE, GL_POINT   NB. Hidden
glBegin GL_QUADS
NB. (_X)-Y plane Back Face
  glColor 1 1 0 0
  glVertex 4{P
  glVertex 7{P
  glVertex 6{P
  glVertex 5{P
NB. (_Z)-X plane Top Face
  glColor 0 1 1 0
  glVertex 0{P
  glVertex 4{P
  glVertex 5{P
  glVertex 1{P
NB. X-Z plane    Bottom Face
  glColor 0 0 0 0
  glVertex 2{P
```

```
  glVertex 6{P
  glVertex 7{P
  glVertex 3{P
NB. Z-Y plane    Left Face
  glColor 0 0 1 0
  glVertex 1{P
  glVertex 5{P
  glVertex 6{P
  glVertex 2{P
NB. (_Z)-Y plane Right Face
  glColor 0 1 0 0
  glVertex 0{P
  glVertex 3{P
  glVertex 7{P
  glVertex 4{P
NB. X-Y plane    Front Face
  glColor 1 0 0 0
  glVertex 0{P
  glVertex 1{P
  glVertex 2{P
  glVertex 3{P
glEnd ''
)
```

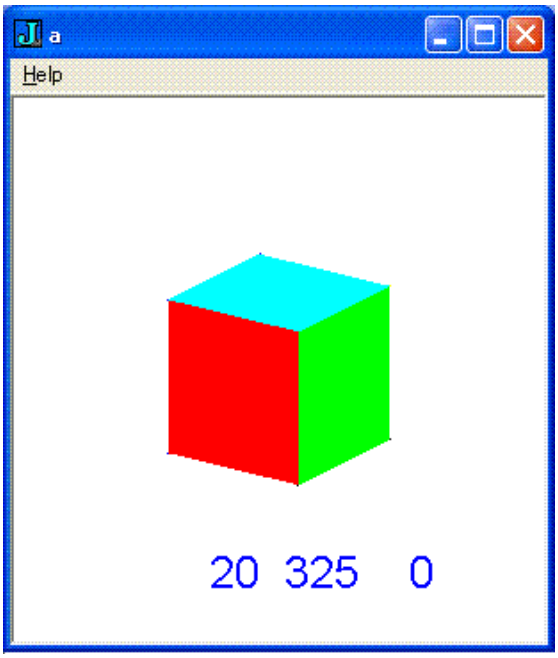
OpenGL での陰線処理については、種々の手法があり仲々難しい。ここでは最も簡単に、「最初の位置で、各軸に対して面を作る頂点の連結を目に見えるときは右ねじの向き、見えないときは左ねじの向きとして、BACK の面だけをを GL_POINT と指定する」という方法を取った。(酒井幸市の書 p.31-35)　また、このように線だけの表示は Wired Frame と呼ばれる。左図は陰線処理をしない表示、右図は陰線処理を行った表示である。





３次元 OpenGL の表示法にはこれに対し、次の Solid 法がある。これはフラグ LS の値により、次のように切り替えて行なう。

```
if. LS = 0
  do.
    glPolygonMode GL_FRONT, GL_LINE   NB. Paint line
  else.
    glPolygonMode GL_FRONT_AND_BACK, GL_FILL   NB. Paint full
end.
```



陰線処理を行うかどうかの選択は、キーボードからの'h'文字の入力で切り替ええる。同様に、Wired と Solid の切り替えは's'で行なう。

これは、回転の場合の'x/X', 'y/Y', 'z/Z'などのキー入力方式と同じであるが、OpenGL ではボタンによらず、このようにするのが普通である。

９．正８面体の回転と陰線処理の例(OpGLN3.ijs)