

J-OpenGLによるベジエ(Bezier)グラフィックスーその2 立体空間内で滑らかな曲面を描く

西川 利男

1. ベジエ(Bezier)グラフィックスと J-OpenGL

J-OpenGLとベジエ(Bezier)グラフィックスの考え方と基本の技術として平面上で滑らかな曲線を描くプログラミングについては別項[1]で述べた。

ここでは、立体空間内で滑らかな曲面を描くプログラミングを扱う。

Jのシステム内には、前と同様、デモとプログラムが載せられている。

J4 では [Studio]-[Demos]-[OpenGL Lab Demo]-[gldemo]-[surface]

J6 では [Studio]-[Demos]-[OpenGL simple]-[surface]

実は、このデモ・プログラムはOpenGLの実際上の標準規格書であり、文法書でもある文献[3]「OpenGLプログラミングガイド」のp.448-449のプログラム例の同じデータを使ったものである。文献はC言語のコーディングであるが、J-[Studio]はもちろんJのコードで書かれているので、OpenGLのコマンドをそれぞれ比較されたらよい。

ここでは、文献[2]のCのプログラム例を参考に、私なりにJでわかり易く、機能をいくつか追加したプログラムで説明する。

2. OpenGLでのベジエ処理処理

ベジエ(Bezier)の計算の原理は、別項[1]で述べたが、実際上はJ-OpenGLで行う限り、Jであらわに計算する必要はない。OpenGLのコマンドとして備えられているので、その仕様に合わせたデータを用意するだけである。ただ、この仕様が仲々ややこしい。

最初に用意する頂点の値を制御値(Control Value)という。頂点の個数の(X, Y, Z)の値が必要である。

そして最終的に滑らかな曲線や曲面を描く頂点の座標値を評価値(Evaluate Value)と言う。そのための計算がベジエ(Bezier)の計算であり、そのプログラムルーチンは評価関数、エバリュエータ(Evaluator)と呼ぶ。

大切なポイントはベジエ(Bezier)処理の内挿かつ近似とは、評価値の最初と最後の点だけは元となる制御値の点を通るが、評価値の途中の点は制御値の点を通らない。制御値からのばらつきが全体として最小になるような値となる。つまり、これが滑らかな曲線や曲面になるのである。この点はスプライン近似などとは異なる。

なお、これとは別の処理にNURBS(Non-Uniform Rational B-Spline)法があるが、これもOpenGLでサポートされている。

[1] 西川利男「J-OpenGLによるベジエ(Bezier)グラフィックスーその1

平面上で滑らかな曲線を描く」JAPLA研究会資料 2014/3/15

[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).

[3] M. Woo, J. Neider, T. Davis「OpenGLプログラミングガイド第2版」

アジソン・ウエスレイ(1997).

3. J-OpenGL のプログラム

プログラムは次のような構成になっている。
 OpenGL 機能の取り込み (J4 の場合)
 require 'gl3'
 OpenGL 機能の取り込み (J6 の場合)
 require 'jzopengl'
 coinsert 'jgl3'
 ウィンドウ・フォームの作成 A
 プログラムの起動 a_run
 OpenGL の取り込み (J4 の場合) glarc''
 OpenGL の取り込み (J6 の場合) ogl=: '' conew' jzopengl'
 インスタンス ogl として行う
 画面の大きさや投影法の設定 a_size
 文字コマンド a_char
 画像の描画 a_paint
 ベジエの計算、描画など

4. ベジエ処理の J のプログラム

ベジエの計算、描画などの J のプログラムは画像の描画 a_paint の中で書かれる。

4. 1 制御値の設定

「ちょっと凹んだ亀の甲羅」をイメージしてもらいたい。

4 x 4 = 16 点の (X, Y, Z) の値を最初の制御値とした。

これは、出来るだけ簡単で、特徴が現れる図形パターンの値を、文献[2] p.49 の値を元に手直しして作ったものある。

```
VP0 := noun define NB. plane = X-Z frame, height = Y-axis
```

```
1 0 1
-0.3 0 1
-0.3 0 1
1 0 1
```

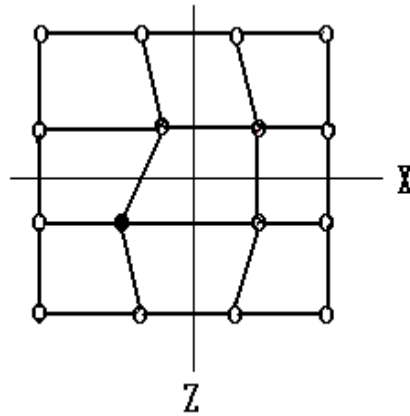
```
1 0 0.3
-0.3 -1.3 0.3
-0.3 1 0.3
1 0 0.3
```

```
1 0 -0.3
-0.3 1.3 -0.3
-0.3 1.3 -0.3
1 0 -0.3
```

```
1 0 1
-0.3 0 1
-0.3 0 1
1 0 1
```

Y 方向から見下ろすと
 面上に乗っかっていて、

OpenGL の座標系は、X 軸は右手を向き、Y 軸は上向きで、Z 軸は手前を正方向とする右手系である。上の制御値の頂点データは、X-Z 面上の 4 角の縁のワクの中で 3 点は山となり、1 点だけ谷として、「ちょっと凹んだ亀の甲羅」となっている。



X-Z 面を上方

白い点は X-Z

黒い点は X-Z 面より下にあり、
 灰色の点は X-Z 面の上方にある

4. 2 J-OpenGL のベジエ操作プログラム

以下がベジエ操作のプログラムであるが、コメントとともにポイント部分を説明する。

```
surface:=verb define
x=. y.
x=. ' ', ((x=LF)#i.#x)} x
points=: SIZE * ".x
```

NB. 文字列データを値へ

```

glClearColor 1 1 1 1
glClear GL_COLOR_BUFFER_BIT + GL_DEPTH_BUFFER_BIT
glColor 0 0 0 1
glMatrixMode GL_MODELVIEW
glLoadIdentity
glMap2 GL_MAP2_VERTEX_3, 0 1 3 4 0 1 12 4, points
glEnable GL_MAP2_VERTEX_3
glEnable GL_AUTO_NORMAL
glMapGrid2 STEPS, 0 1, STEPS, 0 1
glEnable GL_DEPTH_TEST
glShadeModel GL_FLAT
glRotate R, . 3 3 $ 1 0 0 0
glEvalMesh2 (FILL{GL_LINE, GL_FILL}, 0, STEPS, 0, STEPS
)

```

NB. 色は黒

NB. 以下に説明

NB. 以下に説明

NB. STEPS による粗密

最もベジエの設定をおこなう大切なコマンドで、引数は次の意味をもっている。

```

glMap2 target,
      ul, u2, ustride, uorder
      vl, v2, vstride, vorder
      points

```

target は GL_MAP2_VERTEX と指定する
u1=0 から u2 =1 の範囲、
ustride=3 X, Y, Z の 3 つの値、
uorder=4 4 つの点
v1=0 から v2 =1 の範囲
vstride=12 全体では 3x4=12 個の値
vorder=4 4 次の配列
制御点の座標値

以上の設定で、次のようになる。

```

glMap2 GL_MAP2_VERTEX 3, 0 1 3 4 0 1 12 4, points

```

次のコマンドにより target に対してベジエの計算処理を実行する。

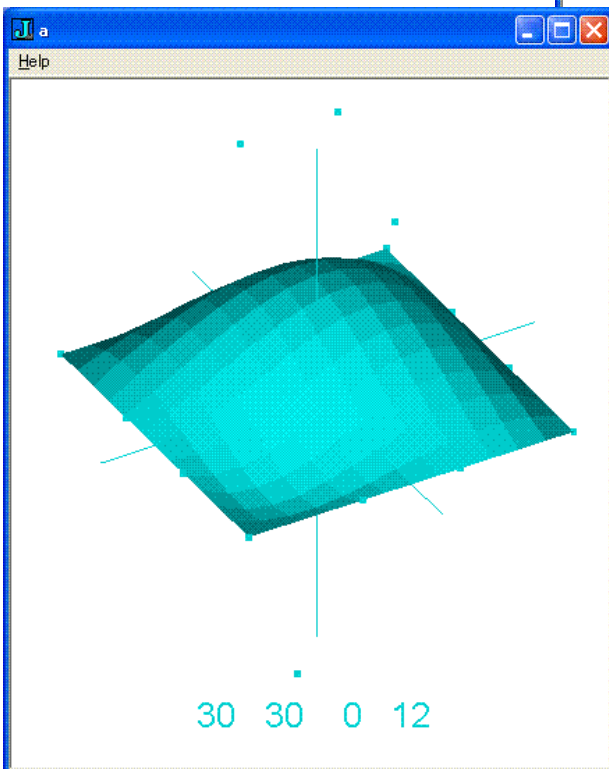
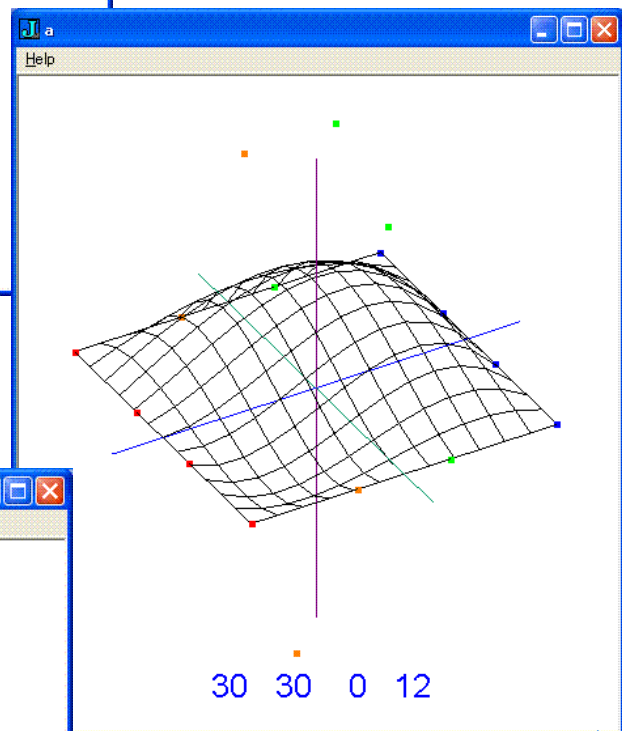
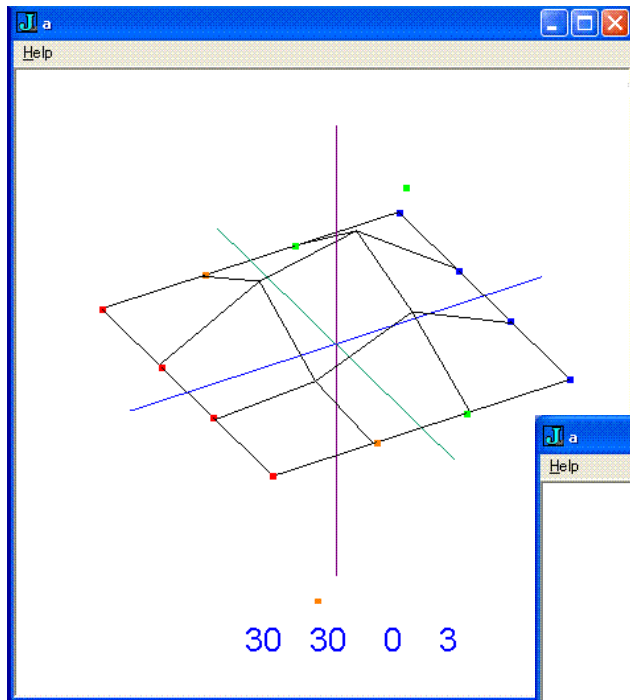
```

glEnable GL_MAP2_VERTEX_3

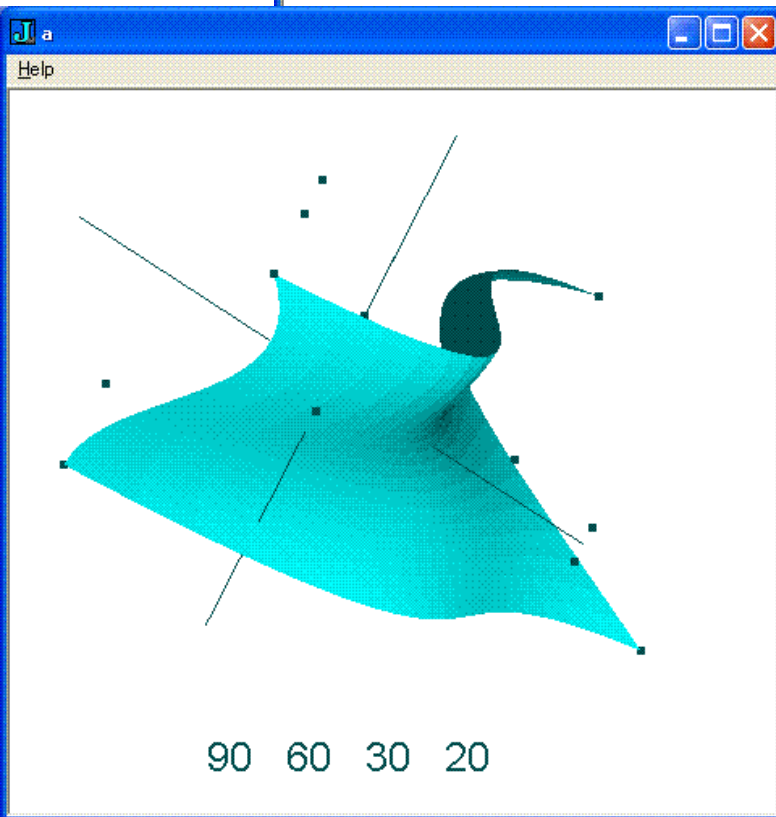
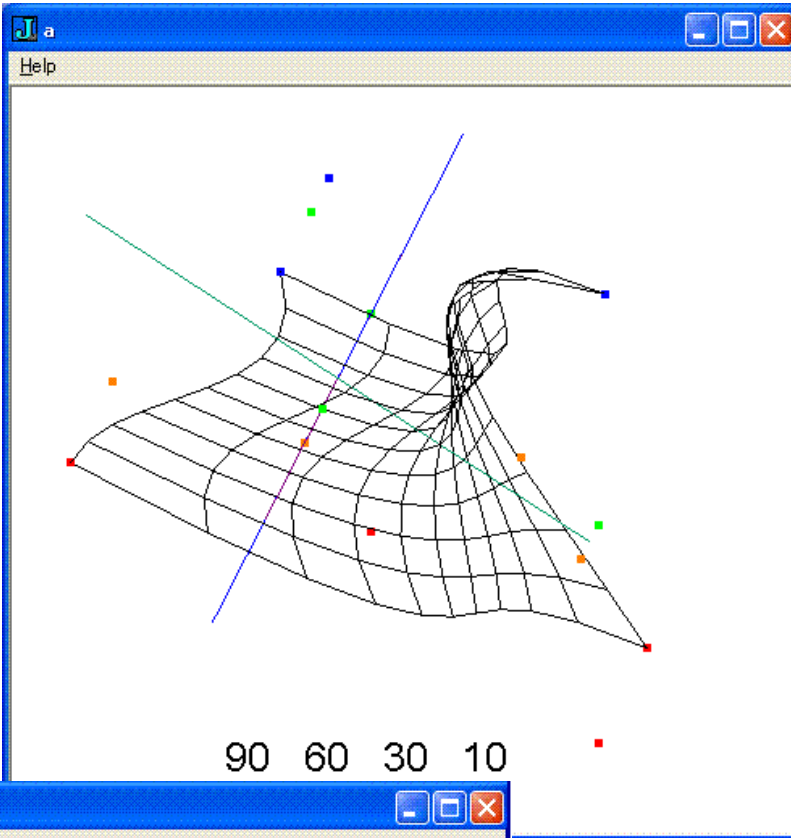
```

作成した動詞 plotctrlpt では各制御点の位置を色分けしてプロットで示した。
なお、いくつか追加の機能のために動詞を定義したが、リスティングを見ていただきたい。

5. プログラムの実際



6. J-Studio-Demo のデータによる実行



NB. OpenGL_Bezier2.ijs

NB. imported from opengl/opgl_surface.ijs

NB. imported from Bessel.ijs

SIZE =: 1

require 'gl3'

A=: noun define

pc a closeok;

menupop "&Help";

menu help "&Help" "" "" "";

menupopz;

xywh 0 0 300 300;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

wā 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

if. (JOB = 0) +. (JOB = 1) do. STEP =: 3 end.

glFont 'arial 30'

glUseFontBitmaps 0 32 26 32

wā 'pshow;top'

)

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
glOrtho _5 5, _5 5, _5 5
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. surface VP1
    plotctrlpt VP1
  case. 1 do. surface VP0
    plotctrlpt VP0
  case. 2 do. surface b2dpts
    plotctrlpt b2dpts
  end.
end.
drawtext ''
glSwapBuffers ''
)

plotctrlpt =: 3 : 0
x =. y.
points =. > char_to_val x
CTRLV =. points
NB. CTRLV =. 16 3$points
CTRLC =. 1 0 0 1;1 0.5 0 1;0 1 0 1;0 0 1 1
glPointSize 5
i =. 0
while. i < 16
do.
  glBegin GL_POINTS
  glColor > (4|i){CTRLC
  glVertex i{CTRLV
  glEnd''
  i =. i + 1
end.

glPointSize 1
glColor 0 0 1 1
glBegin GL_LINES
  glVertex _3 0 0
  glVertex _3 0 0
glEnd ''
glColor 0.5 0 0.5 1
glBegin GL_LINES
  glVertex 0 _3 0
  glVertex 0 _3 0
glEnd ''
glColor 0 0.6 0.4 1

```

```
glBegin GL_LINES
  glVertex 0 0 3
  glVertex 0 0 -3
glEnd ''
```

)

NB. OpenGL プログラミングガイド p.448

NB. J Studio Demo / OpenGL surface と同じデータ

b2dpts=: noun define

```
_1.5 _1.5 4
_0.5 _1.5 2
_0.5 _1.5 -1
 1.5 _1.5 -2
_1.5 _0.5 1
_0.5 _0.5 3
 0.5 _0.5 0
 1.5 _0.5 -1
_1.5 0.5 4
_0.5 0.5 0
_0.5 0.5 3
 1.5 0.5 4
_1.5 1.5 2
_0.5 1.5 -2
_0.5 1.5 0
 1.5 1.5 -1
```

)

NB. 酒井幸市「OpenGL でつくる 3次元CG」p.49, 森北出版(2008)

NB. JOB = 1

VP0 =: noun define NB. plane = X-Z frame, height = Y-axis

```
_1 0 1
_0.3 0 1
_0.3 0 1
 1 0 1

_1 0 0.3
_0.3 -1.3 0.3
_0.3 1 0.3
 1 0 0.3

_1 0 -0.3
_0.3 1.3 -0.3
_0.3 1.3 -0.3
 1 0 -0.3

_1 0 -1
_0.3 0 -1
_0.3 0 -1
 1 0 -1
```

)

NB. JOB = 0

VP1 =: noun define NB. plane = X-Y frame, height = Z-axis

```
_1 1 0
_0.3 1 0
```



```

0.3 1 0
1 1 0

_1 0.3 0
_0.3 0.3 _1.3
_0.3 0.3 1
1 0.3 0

_1 _0.3 0
_0.3 _0.3 1.3
_0.3 _0.3 1.3
1 _0.3 0

_1 _1 0
_0.3 _1 0
_0.3 _1 0
1 _1 0
)

```

```

char_to_val =: 3 : 0 NB. referred from nmap
x=. y.
x=. ', ' ((x=LF)#i.#x)} x
points=. SIZE * ".x
DAT =. <"(1) _3 [\ points
)

```

NB. J Original Version

```

=====
surface=:verb define
x=. y.
x=. ', ' ((x=LF)#i.#x)} x
NB. wr x
points=: SIZE * ".x
glClearColor 1 1 1 1
glClear GL_COLOR_BUFFER_BIT + GL_DEPTH_BUFFER_BIT
glColor 0 0 0 1
glMatrixMode GL_MODELVIEW
glLoadIdentity''
glMap2 GL_MAP2_VERTEX_3,0 1 3 4 0 1 12 4, points
glEnable GL_MAP2_VERTEX_3
glEnable GL_AUTO_NORMAL
glMapGrid2 STEPS,0 1,STEPS,0 1
glEnable GL_DEPTH_TEST
glShadeModel GL_FLAT
glRotate R ,. 3 3 $ 1 0 0 0
NB. glRotate 85 1 1 1
glEvalMesh2 (FILL{GL_LINE, GL_FILL}),0,STEPS,0,STEPS
)

light=:verb define
if. FILL do.
glLight GL_LIGHT0, GL_AMBIENT, 0.1 0.1 0.1 1
glLight GL_LIGHT0, GL_DIFFUSE, 0.7 0.7 0.7 1
glLight GL_LIGHT0, GL_SPECULAR, 0.0 0.0 0.0 1

```

```

glEnable GL_LIGHTING
glEnable GL_LIGHT0
glMaterial GL_FRONT, GL_AMBIENT_AND_DIFFUSE, 0 1 1 1
else.
glDisable GL_LIGHTING
end.
)

```

```

a_help_button=: verb define
wd'mb OpenGL *Press keys, a, s: steps, f: toggle line or fill, l: larger.'
wd 'setfocus g'
)

```

```

NB. indicate rotated angle x, y, z in degree =====
drawtext =: verb define
glMatrixMode GL_MODELVIEW
glLoadIdentity ''
glColor 0 0 0 1 NB. color is determined by glMaterial Command
glRasterPos 1.5 4.5 0
glCallLists 5 " : R, STEPS
)

```