JAPLA 研究会資料　20014/3/15

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

西川　利男

はじめに

コンピュータ・グラフィックスで滑らかな曲線を描くには、いろいろな方法がある。

(1) 直接、点を結ぶ方法（従来からの方法）
・複数の点の座標を次々と結ぶ。点の数が少なければ折れ線になる。
・滑らかな線にするには、区間を分割して点を多くして結べば、滑らかになる。
Ｊのグラフィックスとしては、plot ルーチンなどで行う。

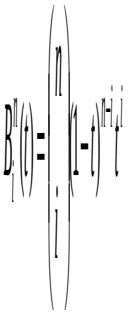
数式で計算できる数学の関数の表示などでは、これで十分である。しかし、物理的な測定データなどを元として滑らかな曲線を描きたいときには、これでは出来ない。

(2) 計算により滑らかな曲線を描く方法－１
与えられた点の座標からそれらを通る内挿点を求めて、つないで曲線にする。
これにはスプライン法などがある。

(3) 計算により滑らかな曲線を描く方法－２
複数の点－これを制御点(Control Point)と呼ぶ－を与えてにこれらに最も近接する曲線の点の座標を計算で生成して(エバリュエータ、評価という)曲線を描く。
これがベジエ(Bezier)法である。近似点の座標 $P(t)$は、次の式で計算される。

$$\boldsymbol{P}(t)=\sum_{i=0}^{\infty}\boldsymbol{B}_i^n(t)\boldsymbol{P}_i$$

ここで重み係数 Bはバーンスタイン基底関数と呼ばれる



同様の別の方法に NURBS(Non-Uniform Rational B-Spline)がある。

OpenGL では、もちろん J-OpenGL でも、計算ルーチンを含めて両方の方法がコマンドレベルでサポートされている。

本稿では、２次元の平面上で Bezier 法による滑らかな曲線を描く場合の J-OpenGL のプログラミングを、マウスによる操作実験を含めて、丁寧に説明する。３次元空間内の Bezier 曲面については「その２」として別項で述べる。

[1] 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).
[2] M. Woo, J. Neider, T. Davis「OpenGL プログラミングガイド第２版」
アジソン・ウェスレイ(1997).

１．J-OpenGL とは

OpenGL とは２次元、３次元のグラフィックスを行う汎用の仕様である。C++言語によりプログラムすることが多いが、その仕様は言語とは独立で、Ｊでもこれをサポートして、同じコマンド体系で行うことができる。したがって、ここではＪというより BASIC や C などふつうの言語感覚でプログラムすることになる。

Ｊのシステムには、[Studio]というデモないしオンライン・チュートリアルがあるが、その中にベジエのプログラム例がある。

J4　では　[Studio]-[Demos]-[OpenGL Lab Demo]-[gldemo]-[curve]
J6　では　[Studio]-[Demos]-[OpenGL simple]-[curve]

[Run] ではそのまま実行するが、[View]ではソース・プログラムを見ることができる。それぞれのプログラム・コードは稿末にあげた。

ソース・プログラムはファイル名をきめて、自分のところにコピーして、納得するまで、コードを修正したりして、実行しつつ、理解するのが良い。

なお、J-OpenGL プログラミングの詳細は、以下の報告を参照のこと。

[3] 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 をどう理解するかー」JAPLA 研究会資料　20013/6/15

今回は、上の J-demo の curve プログラムを元に、マウス操作などを追加したもので説明する。

J4 では、直接プログラムするが、J6 では、OOP でクラスファイルを利用する。そのため、いくつかの違いがあるが、OpenGL としては同じである。

ここでは、基本のポイントを箇条書きで記す。

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
```

上のプログラム a を起動すると、フォーム A に従ったウィンドウ・画面が開き、その画面上で、a_size, a_char, a_paint が実行される。つまり、決められた投影法で図形が表示される。

その後、文字入力、マウス入力などが行われると、それに対応する処理はイベント・ドリブンで即実行される。

２．OpenGL での図形描画の特徴

従来のグラフィックスでは draw_dot, draw_line などの描画命令で、座標位置を指定して図形を描くが、OpenGL では異なっている。

次のような書式で、図形の種類 Polygon と、(X, Y, Z)として、頂点座標を指示する。

```
glBegin(GL_POLYGON)
    glVertex (X0, Y0, Z0)
    glVertex(X1, Y1, Z1)
    glVertex(X2, Y2, Z2)
     ---
glEnd
```

すると、別に定めた投影法により図形が描かれる。

このように、OpenGL の命令コマンドは gl で始まるキーワードになっている。

OpenGL では、座標位置を与えて図形を作成する（レンダーリング）処理と、投影法に従って図形を画面上に表示する処理とを分けている。

つまり、同じ図形であっても、投影法(視点、見方)によっては、実際の見え方はいろいろは変わることになる。OpenGL ではシステム内で、これを自動的に行っている。

３．ベジエの処理（計算と図示）

３．１　まず、ベジエ処理のための制御点を設定する。

４つの制御点(V0, V1, V2, V3)の座標値 X, Y, Z を決める。

なお、OpenGL では、座標の取り方は右手系で、

- 座標の原点は　0, 0, 0
- X の正方向は　右
- Y の正方向は　上
- Z の正方向は　手前

となっている。

ここでは次のようにする。

```
V0: _4 _4 0
V1: _2  4 0
V2:  2 _4 0
V3:  4  4 0
```

これをまとめて

```
CTRLP =: _4 _4 0 _2 4 0 2 _4 0 4 4 0
```

３．２　ベジエ処理のプログラムは次のようになる。

```
bezier=:verb define
ctrlpoints=: y.
glMap1 GL_MAP1_VERTEX_3, 0, 1, 3, 4, ctrlpoints
glEnable GL_MAP1_VERTEX_3
glColor 0 0 0 0
glBegin GL_LINE_STRIP
 glEvalCoord1"0 (i.30)%30
glEnd''
)
```

これを行ごとに見て行こう。

glMap1 GL_MAP1_VERTEX_3, 0, 1, 3, 4, ctrlpoints　NB. (1)

glEnable GL_MAP1_VERTEX_3　　NB. (2)

(1)行はベジエ計算のための設定を行う。

```
glMap1 target, u1, u2, ustride, uorder, ctrlpoints
```

- target は GL_MAP1_VERTEX_3　１次のマップを３つの頂点座標 X,Y,Z で作る。
- u1, u2 は 0 から 1 の間で計算する
- ustride は間隔 3 で行う
- uorder は制御点 4 個でおこなう
- ctrlpoints は実際の制御点の値 12 個を入れる

(2)行では、計算ルーチンを有効にする

```
glColor 0 0 0 0            NB. (3)
```
(3)行は、ベジエ曲線の色は黒にする
```
glBegin GL_LINE_STRIP      NB. (4)
 glEvalCoord1"0 (i.30)%30  NB. (5)
glEnd''                    NB. (6)
```
(4),(5),(6)の行ではベジエ計算と線の連結を行う
先の条件に従い、glEvalCoord1 の関数により、0, 1/30, 2/30, .., 29/30 の３０個の評価値(Evaluate Value)の座標を生成し、(4)行の GL_LINE_STRIP により線分として glBegin と glEnd で繰り返してつなぐ。

次はベジエ計算とは関係ないが、制御点４つをプロットするものである。
```
plotctrlpt =: verb define
ctrlpoints=: y.
glPointSize 5
glColor 1 0 0 1
glBegin GL_POINTS
 glVertex 4 3$ctrlpoints
glEnd''
)
```

４．マウス操作により、制御点を移動させて、ベジエ曲線の変化をみる

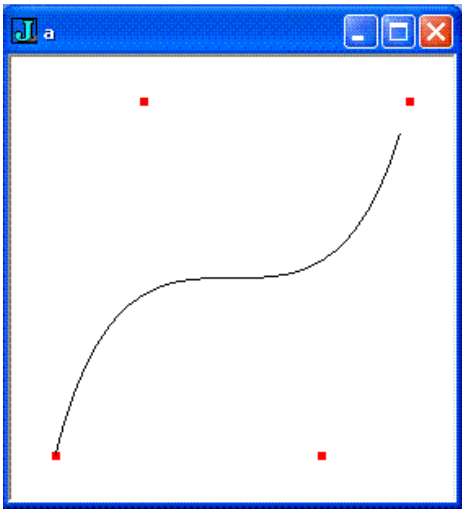
これは J-demo にはないが、ベジエ処理をより良く理解するために、今回作ったものである。また、これは OpenGL とは別の require 'gl2'で可能な機能である。

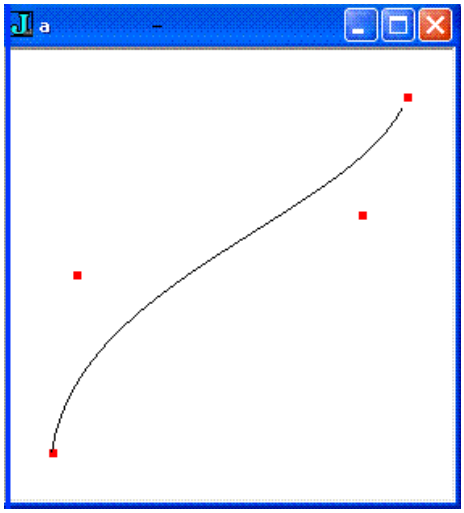
マウスの左ボタンの操作は、次のコマンドで行われる。
```
a_g_mbldown=: 3 : 0
d=. ". sysdata           NB. get sysdata with mouse action
x=.(0{d) * 1000 % (2{d)  NB. convert x pixel value
y=.(1{d) * 1000 % (3{d)  NB. convert y pixel value
X0 =: _5 + x % 100       NB. start graphic X0 value
Y0 =: _5 + y % 100       NB. start graphic Y0 value
)
a_g_mblup=: 3 : 0
  (途中省略)
X                      NB. end graphic X value
Y                      NB. end graphic Y value
bez_change ''          NB. renew bezier calc. and drawing
)

NB. change Bezier curve
NB.  using subprogram between
NB.  global: X0, Y0, X, Y
bez_change =: 3 : 0
XX =. X0 + (_0.2, 0.2)  NB. searching area XX
YY =. Y0 + (_0.5, 0.5)  NB. searching area YY
TCP =. }:"(1) 4 3$CTRLP
i =. 0
while. i < 4
  do.
    test =. , @( (XX & between) ` (YY & between)/. ) i{TCP
    if. 1 = */ test do. TV =. i end.
    i =. i + 1
  end.
TCQ =. (X, Y) TV } TCP
TCQ =. TCQ ,"(1) 0
CTRLP =: , TCQ
glpaintx ''
)
```

プログラムの詳細な説明は行わないが、その概要は次のようである。
制御点の位置でマウス・ダウンを行い、ピクセル位置を得て、グラフィック座標値(X0, Y0)に変換する。この位置から±0.2の範囲でサーチし、どの制御点が選ばれたかを知る。（サブプログラム between を使用）そして、マウスをドラッグして、変えたい位置でマウス・アップする。そこでその位置(X,Y)を、新たな制御点として、ベジエ処理を行い、ベジエ曲線を更新する。

５．プログラムの実行





プログラムのコーディング(J4)

```
NB. OpGLN_Bezier.ijs  2014/2/1 by T.N.
NB. Bezier line freely changed with mouse action
NB. mouse down, move, then mouse up => new Bezier line will be drawn

NB. referred from OpenGL_curve.ijs in [lab] [demo] [opengl symple]

wr =: 1!:2&2

require 'gl3'
require 'gl2'

A=: noun define
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=: verb define
wd A
glaRC''
CTRLP =: _4 _4 0 _2 4 0 2 _4 0 4 4 0
wd 'pshow;ptop'
)

a_g_char =: verb define
R =: 360 | R + 2 * 'xyz' = 0 { sysdata
glpaintx''
)

a_g_size=:verb define
wh=.glqwh''
glViewport 0 0,wh
glMatrixMode GL_PROJECTION
glLoadIdentity''
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
glClearColor 1 1 1 1
glClear GL_COLOR_BUFFER_BIT
plotctrlpt CTRLP  NB. plot Contrl Points
bezier CTRLP      NB. draw Bezier Curve
glaSwapBuffers ''
)


bezier=:verb define
ctrlpoints=: y.
NB. ctrlpoints=: _4 _4 0 _2 4 0 2 _4 0 4 4 0
```

```
glMap1 GL_MAP1_VERTEX_3, 0, 1, 3,4, ctrlpoints
glEnable GL_MAP1_VERTEX_3
glColor 0 0 0 0
glBegin GL_LINE_STRIP
 glEvalCoord1"0 (i.30)%30
glEnd''
)

plotctrlpt =: verb define
ctrlpoints=: y.
glPointSize 5
glColor 1 0 0 1
glBegin GL_POINTS
 glVertex 4 3$ctrlpoints
glEnd''
)

NB. mouse action ==============================
a_g_mbldown=: 3 : 0
d=. ". sysdata
x=.(0{d) * 1000 % (2{d)
y=.(1{d) * 1000 % (3{d)
X0 =: _5 + x % 100   NB. mouse start coord. X
Y0 =: _5 + y % 100   NB. mouse start coord. Y
)

a_g_mmove=: 3 : 0
d=.". sysdata
if. -.4{d do. return. end.
x=.(0{d) * 1000 % (2{d)
y=.(1{d) * 1000 % (3{d)
)

a_g_mblup=: 3 : 0
d=. ". sysdata
x=.(0{d) * 1000 % (2{d)
y=.(1{d) * 1000 % (3{d)
X =: _5 + x % 100   NB. mouse end coord. X
Y =: _5 + y % 100   NB. mouse end coord. Y
bez_change ''
)

NB. change Bezier curve
NB.  global: X0, Y0, X, Y
bez_change =: 3 : 0
XX =. X0 + (_0.2, 0.2)
YY =. Y0 + (_0.5, 0.5)

TCP =. }:"(1) 4 3$CTRLP
i =. 0
while. i < 4
  do.
    test =. , @( (XX & between) ` (YY & between)/. ) i{TCP
    if. 1 = */ test do. TV =. i end.
    i =. i + 1
  end.
NB. wr 'i: ', ": TV
NB. wr TV { TCP
```

```
NB. wr $ TV { TCP
TCQ =. (X, Y) TV } TCP
TCQ =. TCQ ,"(1) 0
CTRLP =: , TCQ
glpaintx ''
)
```

NB. CinC J Phrases p.32

```
NB. 5 CC 5 13 => 1, 15 CC 5 13=> 0
CC =: ({.@] <: []) *. ([ <: {:@])

NB. 5 13 between 10 + i.7 => 1 1 1 1 0 0 0
between =: CC~"(1 0)

NB. test 1st value(4) between 5 and 9, 2nd value(11) between 10 and 13, yes:1
no:0
NB. , @( ((5 9)&between) ` ((10 13)&between)/. ) 4 11 => 0 1
NB. , @( ((5 9)&between) ` ((10 13)&between)/. ) 6 11 => 1 1
NB. , @( ((5 9)&between) ` ((10 13)&between)/. ) 10 11 => 0 1
NB. , @( ((5 9)&between) ` ((10 13)&between)/. ) 10 14 => 0 0
```

J6のソースコード

```
require 'jzopengl'
coinsert 'jgl3'

A=: noun define
pc a;
xywh 0 0 150 150;cc g isigraph rightmove bottommove;
pas 0 0;
rem form end;
)

a_run=: verb define
wd A
R=: 0 0 0
ogl=: ''conew'jzopengl'
a_g_paint''
wd 'pshow'
)

a_g_char=: verb define
R=: 360 | R + 2 * 'xyz' = 0 { sysdata
a_g_paint''
glpaint''
)

a_g_paint=: verb define
wh=. glqwh''
alloc__ogl wh
glViewport 0 0,wh
glMatrixMode GL_PROJECTION
glLoadIdentity''
if. </wh do.
  glOrtho _5 5,(_5 5*%%/wh), _5 5
else.
  glOrtho (_5 5*%/wh), _5 5 _5 5
end.
glClearColor 0 0 0 0
glClear GL_COLOR_BUFFER_BIT
bezier ''
glFlush''
glpixels 0 0,wh,pixels__ogl''
)

bezier=: verb define
ctrlpoints=: _4 _4 0  _2 4 0  2 _4 0  4 4 0
glaMap1 GL_MAP1_VERTEX_3, 0, 1, 3,4, ctrlpoints
glEnable GL_MAP1_VERTEX_3
glColor 1 1 1 1
glBegin GL_LINE_STRIP
glEvalCoord1d"0 (i.30)%30
glEnd''
glPointSize 5
glColor 1 0 0 1
glBegin GL_POINTS
glVertex 4 3$ctrlpoints
glEnd''
)
```

```
a_close=: 3 : 0
destroy__ogl''
wd 'pclose'
)

a_cancel=: a_close
```