JAPLA 研究会資料 2013/3/23

Ｊによる３進法の処理－その２
３進数によるフラクタル・グラフィックス
OOP プログラムの例として

西川　利男

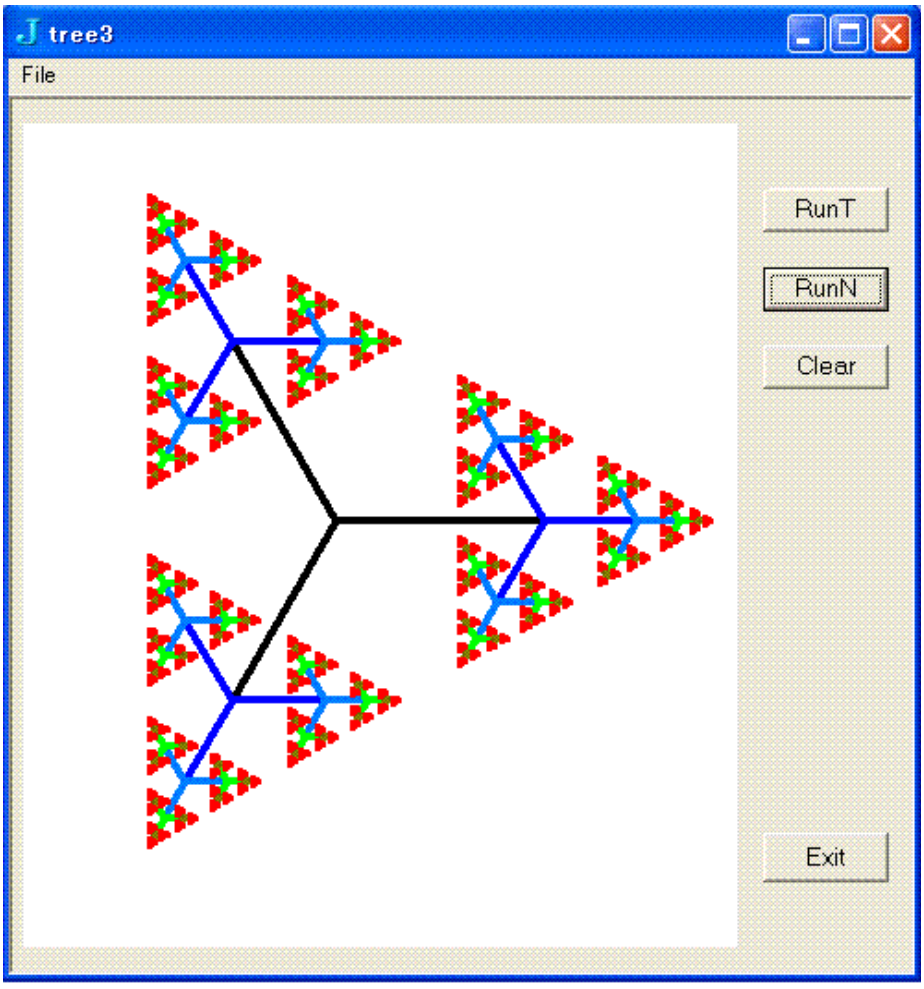
はじめに

今年の大学センター試験(1/21)に、３進数に関する BASIC によるプログラミングの問題が出された。単なる知識ではなく、真に理解するにはプログラミングも含めて３進法を取り上げたのは非常によかったと思う。

しかしながら、受験生にとってだけでなく、３進法にはどういう価値があるのか、どう使われるのか、これまでの数学の教科書ではあきらかにされていない。

それでは３進法にはどういう利点があるか。ON-OFF-中立からなる３通信方式、空間のトリー構造など、さまざま世界が広がることと思う。そのひとつを見てみよう。

ここでは、３進法を利用した以下のようなフラクタル・グラフィックスを紹介する。さらに、そのプログラム作成について、OOP(オブジェクト指向)の例としても検討する。



１．３進のフラクタル

筆者が数年前に訳したハンス・ラウヴェリエールの著「初めてのフラクタル」[1]に、「３進」の木というのがある。そこにはBASICのプログラムも添えられている。

［1］西川利男「初めてのフラクタル」p.10-12, p.187(プログラム)丸善(1996).

今回は、これを元にしてＪでプログラムを開発した。なお、Ｊでプログラミングすることで、もっと３進数の原理に基づいた理解を得ることができる。

「３進」の木とは、

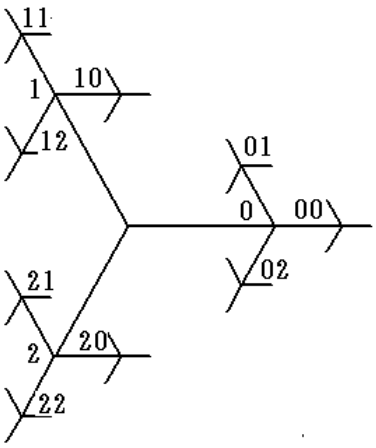
平面の一点から３方向に伸びる枝に対して
そのそれぞれを次の点としてさらに３方向に伸びる枝を
……

と次々と展開して行くと現れるフラクタル構造の木である。

３進数の値を極座標で

0をx軸の正の方向
1をx軸から120度
2をx軸から240度

のベクトルに対応させる。そして桁が増える毎に一定の比率で延長する。



２．Ｊによる３進の木

上のような考え方で、これを次々に行うときれいなフラクタル図形が描かれる。このようなグラフィックスをＪでプログラムしてみようと思う。

別稿で述べたようにＪでは3進数への変換は次のようにプリミティブ#.と #:とを使ってごく簡単に行える。

```
  3 3 3 #: 10          NB. 10進数10を3桁の3進数に変換する
1 0 1
  3 #.^:(_1) 10        NB. 桁数を指定しなくても、変換できる
1 0 1
  3 #. 1 0 1           NB. 3進数1 0 1を10進数に変換する
10
```

それぞれ、つぎのように名前を付けて定義した。

```
tfd =: 3 : '3 #. ^:(_1) y'  NB. ternary from decimal
```

```
dft =: 3 : '3 #. y'          NB. decimal from ternary
```

3進の木の原理は次のポイントにある。
ある点P(X, Y)があるとき、次の点Q(X', Y')は下の式で表される。

$$X' = X + A\text{^}K * \cos(\frac{2\pi}{3} \cdot T)$$

$$Y' = Y + A\text{^}K * \sin(\frac{2\pi}{3} \cdot T)$$

上の式で、次数Kに応じた縮小率A(＝ 0.4)を係数として、3進の桁T(＝ 0, 1, 2のいずれか)で決まるベクトルの点を、次々と結んで延長して行くことで、3進フラクタルの木が描かれる。

計算した数値のようすを記してみる。

```
   testree 2
P: 2
T: 0 0
A: 0.45
==========================================
M=0
------------------------------------------
N: 0 T: 0 0
==========================================
M=1
------------------------------------------
N: 0 T: 0 0
F(0) = 0.00000,     X: 0.45000 Y: 0.00000
------------------------------------------
N: 1 T: 0 1
F(0) = 2.09440,     X:_0.22500 Y: 0.38971
------------------------------------------
N: 2 T: 0 2
F(0) = 4.18879,     X:_0.22500 Y:_0.38971
==========================================
M=2
------------------------------------------
N: 0 T: 0 0
F(0) = 0.00000,     X: 0.45000 Y: 0.00000
F(1) = 0.00000,     X: 0.65250 Y: 0.00000
------------------------------------------
N: 1 T: 0 1
F(0) = 2.09440,     X:_0.22500 Y: 0.38971
F(1) = 0.00000,     X:_0.02250 Y: 0.38971
------------------------------------------
N: 2 T: 0 2
F(0) = 4.18879,     X:_0.22500 Y:_0.38971
F(1) = 0.00000,     X:_0.02250 Y:_0.38971
```

```
-----------------------------------------
N: 3 T: 1 0
F(0) = 0.00000,     X: 0.45000 Y: 0.00000
F(1) = 2.09440,     X: 0.34875 Y: 0.17537
-----------------------------------------
N: 4 T: 1 1
F(0) = 2.09440,     X:_0.22500 Y: 0.38971
F(1) = 2.09440,     X:_0.32625 Y: 0.56508
-----------------------------------------
N: 5 T: 1 2
F(0) = 4.18879,     X:_0.22500 Y:_0.38971
F(1) = 2.09440,     X:_0.32625 Y:_0.21434
-----------------------------------------
N: 6 T: 2 0
F(0) = 0.00000,     X: 0.45000 Y: 0.00000
F(1) = 4.18879,     X: 0.34875 Y:_0.17537
-----------------------------------------
N: 7 T: 2 1
F(0) = 2.09440,     X:_0.22500 Y: 0.38971
F(1) = 4.18879,     X:_0.32625 Y: 0.21434
-----------------------------------------
N: 8 T: 2 2
F(0) = 4.18879,     X:_0.22500 Y:_0.38971
F(1) = 4.18879,     X:_0.32625 Y:_0.56508
*** end ***
```

３．３進の木のグラフィックスーgl2直接プログラム(J602版)

J602版のgl2プログラムの基本については箇条書きで要点のみを記す。

① 引数パラメータx. → x 、y. → y 手動で行う以外に次の方法が可能である。
[Editor]-[Configure]-[Parameters]の以下のオプション
Permit x. y. names in explicit definitions をクリックして有効にする。

② require 'gl2' に加えて
coinsert 'jgl2' が必要

③ グラフの表示 glpaint ''

④ 表示の座標範囲 左上(0, 0) ー 右下(200, 200) 物理的ピクセル値 (可変)

⑤ グラフィックス命令

gllines	…… 線を連結する
glrect	…… 長方形を描く
glpolygon	…… 多角形を描く
glellipse	…… 楕円、円を描く
glcolor	…… 色を指定して変える
glpen	…… ペンの太さを指定して変える

など

なお、グラフィックスのためのウィンドウズ・フォームの作成は
[Editor]-[Form Editor]から指示に従って、[isigraph]を選択して行う。

3進の木・グラフィックスのプログラム(tree3.ijs)のリストの詳細は稿末に記した。

４．３進の木のグラフィックスーgl2-OOPプログラム(J602版)

ところでこのようなグラフィックス処理の実際は、かなり込み入っていて、相当のプログラム量になる。このようなときに、利用するだけであれば、OOP(オブジェクト指向)という手法が、最近の流れである。

別項でOOPの基本の考えについて述べたが、ちょうど適当なOOPの実際例として、クラス・プログラムと利用プログラムの両方にわたってプログラム作成のポイントを説明する。

４．１　クラス・プログラム　ptree3.ijs

クラス・プログラムの作成は、ふつうは[File]-[New Class]をクリックして、その指示(Wizard)に従って行うのが良い。通常のプログラム・スクリプトのサブ・フォルダー上に、pで始まるファイル名でを作成する。ここでは、ptree3.ijsとした。

ここで、クラス・ファイルと実行ファイルの２つが作られ、最初の記述部分も自動的に作られる。これを元に、自分の目的に合うように、修正・拡張してゆけば良い。

クラス・プログラムの最初は、次の宣言から始まる。

```
coclass 'ptree3'

create=: 3 : 0
0
)
```

coclassは、これを実行するスクリプト・プログラム（インスタンス）から呼ばれるプログラム（クラス）であることを宣言する。createは最初に自動的に実行する。今回は何もせず、インスタンスに制御を移してそこから実行するようにした。これ以下は、直接プログラムのグラフィックス処理をそのまま、そっくり記述してある。

４．２　実行スクリプト・プログラム tree3oop.ijs

```
path =: 7 {. 1!:43 ''
load path, '¥system¥j602-user¥user¥classes¥ptree3.ijs' …… ①
run =: 3 : 0 …… ②
ins =: 'd1' conew 'ptree3'
run__ins y
)
ooprun =: 3 : 0 …… ③
ins =: '' conew 'ptree3'
'** OOP loaded OK **', LF, '** ins created **', LF, 'try: tern__ins 20'
)
```

① クラス・プログラムをロケールにロードして、使えるよう準備をおこなう。

② runを実行すると、conewによりインスタンスinsが作られる。そしてrun__insを実行できる。これはクラス・プログラムに在る3進の木のグラフィックス処理プログラムrunに他ならない。この引数yの値は'd1'を通してクラス・プログラムに渡される。これは3進の木の次数となる。

③ ooprun '' により、クラス・プログラムのロード状態をテストする。

結論として、「ユーザはこの簡単な実行スクリプト・プログラム tree3oop.ijs を書くだけで、3進の木・フラクタル・グラフィックスを楽しむ」ことが出来るのである。

５．３進の木のグラフィックスの実行例

プログラムの実行は、直接版でもOOP版でも、同じである。

OOP版であれば、上のtree3oop.ijsを作成、実行してから、例えば

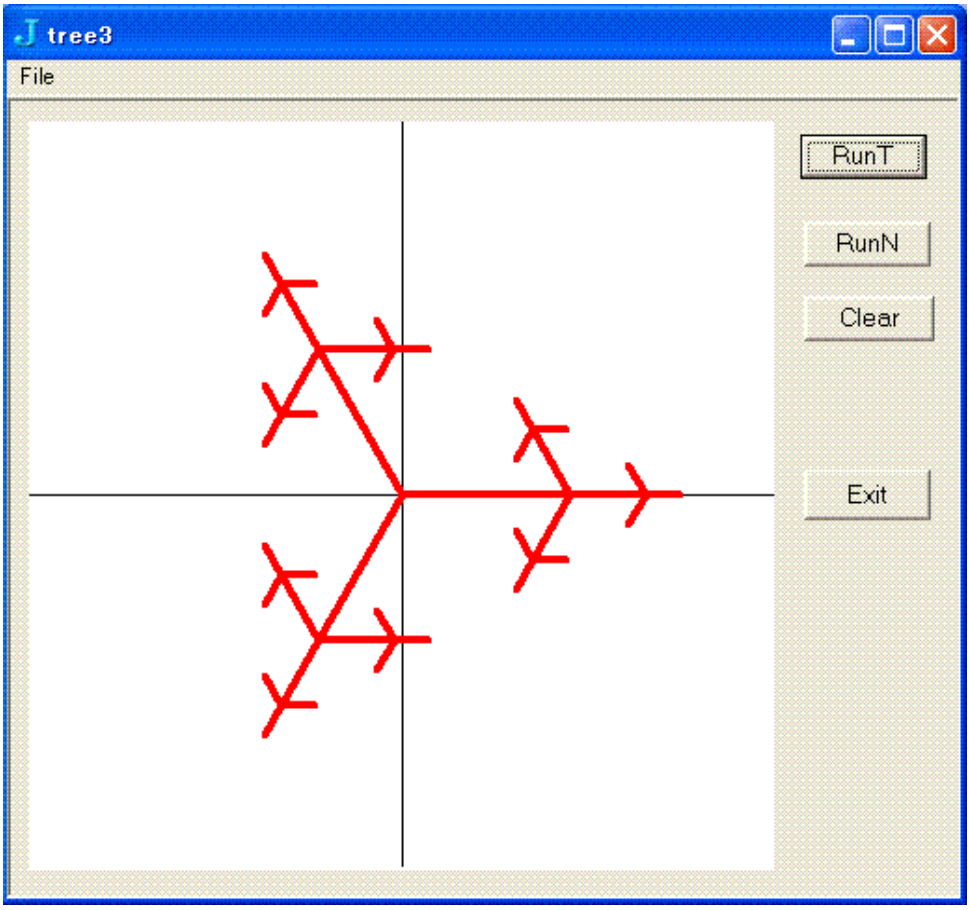
```
run 3
```

と打ち込むと、下のような画面となり、ボタン[RunT]を押すと3次の3進の木・フラクタルが表示される。なお

```
run ''
```

では、デフォルトとして、6次としてある。

また、ボタン[RunN]では、ボタンを押すごとに次々と高次の3進の木が延長拡大して行くのを観察できる。第１ページにあげた絵はこの例である。



Jのプログラム・リスト

```
tfd =: 3 : '3 #. ^:(_1) y'  NB. ternary from decimal
dft =: 3 : '3 #. y'         NB. decimal from ternary

NB. decimal to ternary ==============================
NB.  tern 50 => 1 2 1 2
tern =: 3 : 0
N =. y
M =. N
T =. ''
i =. 0
while. i < (3 ^. N)   NB. repeat while less than log N based on 3
  do.
    wr 'M: ', ": M1 =. <. M % 3    NB. integer divide by 3
    wr 'T: ', ": T =. (3|M), T     NB. residue by 3, appended as upper digit
    if. M1 = 0 do. '** tern:', (":T) return. end.
    rd 1
    M =. M1
    i=. i + 1
  end.
)

NB. ternary to decimal ==============================
NB.  tern_to_dec 1 2 1 2 => 50
tern_to_dec =: 3 : '+/ y * 3 ^ |. i. #y'

NB. gyakuten of ternary notation ====================
NB.    gyakuten 100
NB. 1 0 2 0 1
NB. 1 0 2 0 1
NB. 1
gyakuten =: 3 : 0
wr 'N: ', (":y), ', T: ', ": T =. 3 #.(^:_1) y
'gyakuten: ', ": TT =. |. T
)

gyaku =: 3 : 0"(0)
T =. 3 #.(^:_1) y
TT =. |. T
T -: TT
)

NB.    gyakutensuu 50
NB. 0 1 2 4 8 10 13 16 20 23 26 28 40
gyakutensuu =: 3 : '(gyaku i. y.)#(i. y.)'
```

プログラム・リスト
３進の木のグラフィックスーgl2直接プログラム(J602版)

```
NB. tree3.ijs
require 'trig'

COS =: 3 : 'cos rfd y.'
SIN =: 3 : 'sin rfd y.'

require 'gl2'
coinsert 'jgl2'  NB. J602

TREE3=: 0 : 0
pc tree3;
menupop "File";
menu new "&New" "" "" "";
menu open "&Open" "" "" "";
menusep ;
menu exit "&Exit" "" "" "";
menupopz;
xywh 244 192 34 12;cc cancel button;cn "Exit";
xywh 3 6 234 214;cc tree isigraph;
xywh 244 9 34 11;cc RunT button;
xywh 244 28 34 11;cc RunN button;
xywh 244 47 34 11;cc Clear button;
pas 6 6;pcenter;
rem form end;
)

run =: tree3_run
tree3_run=: 3 : 0
RK =: y
if. 0 = #y do. RK =: 4 end.
wd TREE3
NK =: 0
Rd =: 0
Gr =: 0
wd 'pshow;'
)

tree3_close=: 3 : 0
wd'pclose'
)

tree3_cancel_button=: 3 : 0
tree3_close''
```

```
)
adj =: 3 : '250 + 250 * y.'  NB. revised for J602
NB. 2013/2/11 OK
tree3_RunT_button=: 3 : 0
gllines 0 250 500 250  NB. X-axis
gllines 250 0 250 500  NB. Y-axis
glrgb 255 0 0
glpen 4 0
A =. 0.45
KA =. %: %: A
K =. 0
while. K < RK do.
  KK =. K + 1
  PXY =. runpos K
  i =. 0
  while. i < #PXY do.
    PXYi =. i{PXY
    'X Y' =. PXYi
    gllines  adj X, Y, (X + A^KK), (Y)
    gllines  adj X, Y, (X - 0.5*A^KK), (Y + (0.5*(%:3))*A^KK)
    gllines  adj X, Y, (X - 0.5*A^KK), (Y - (0.5*(%:3))*A^KK)
    i =. i + 1
  end.
K =. K + 1
end.
glpaint ''
)
GCol =: 0 0 0;0 0 255;0 125 255;0 255 0;125 125 0;255 0 0;255 125 0;255 255
125

NB. NK is global, when click button, then increase by one ====
tree3_RunN_button=: 3 : 0
glrgb > NK{GCol
glpen 1 0
NB. gllines 500 200 500 800
NB. gllines 100 500 900 500
NB. glrgb Rd, Gr, 255
glpen 4 0
A =. 0.45
KA =. %: %: A
  KK =. NK + 1
  PXY =. runpos NK
  i =. 0
  while. i < #PXY do.
    PXYi =. i{PXY
```

```
    'X Y' =. PXYi
    gllines adj X, Y, (X + A^KK), (Y)
    gllines adj X, Y, (X - 0.5*A^KK), (Y + (0.5*(%:3))*A^KK)
    gllines adj X, Y, (X - 0.5*A^KK), (Y - (0.5*(%:3))*A^KK)
    i =. i + 1
  end.
NK =: NK + 1
Rd =: Rd + 10
Gr =: Gr + 10
glpaint ''
)

tree3_Clear_button=: 3 : 0
glclear ''
glrgb 0 0 0
glpen 1 0
gllines 500 200 500 800
gllines 100 500 900 500
NK =: 0
Rd =: 0
Gr =: 0
glpaint ''
)

t2pos =: 3 : 0
:
A =. x.
M =. {. y.
T =. }. y.
T =. |. (-M) {. T
MM =. i.M
XX =. (COS 120 * T) * (A ^ (>: MM))
YY =. (SIN 120 * T) * (A ^ (>: MM))
X =. +/ XX
Y =. +/ YY
X, Y
)

runpos =: 3 : 0
P =. y.
T =. ((>:P) # 3) #: i. 3 ^ P
PT =. P ,"(0 1) (-P) {."(1) T
0.45 t2pos"(1) PT
)
```

プログラム・リスト
３進の木のグラフィックスーgl2-OOPプログラム(J602版)
ーOOP実行スクリプト

```
NB. tree3oop.ijs

path =: 1!:43 ''
path =: 7 {. path

load path, '¥system¥j602-user¥user¥classes¥ptree3.ijs'

ooprun =: 3 : 0
ins =: '' conew 'ptree3'
'** OOP loaded OK **', LF, '** ins created **', LF, 'try: tern__ins 20'
)

run =: 3 : 0
ins =: 'd1' conew 'ptree3'
run__ins y
)
```

プログラム・リスト
３進の木のグラフィックスーgl2-OOPプログラム(J602版)
ーOOP-Classプログラム

```
coclass 'ptree3'

create=: 3 : 0
0
)

sqrt =: 3 : '%: y'
DA =: 789

==========================================================
NB. imported from tree3.ijs
```

以下tree3.ijs と全く同一であるので、省略した。