JAPLA 研究会資料 2021/3/25

Ｊによる正方形フラクタルのグラフィックス

西川　利男

はじめに

今やかなり以前になるが、「初めてのフラクタル」[1] なる翻訳書を出版したことがある。私がつくばの研究所を定年退職後の仕事として、無我夢中でやった記憶がある。今、あらためて取り出し、読み直してみている。同書にはBASICによるプログラムも付いていている。なお、私の訳書には３次元トリーのフラクタルはあるが、正方形フラクタルはない。文献[2]にその記載があったので、同時に参照した。

[1] H. ラウヴェリエール、西川利男訳「初めてのフラクタル
ー数学とプログラミング」、丸善(1996).

[2] 佃勉「フラクタルの世界」山海堂(1988).

課題は正方形群を次々と縮小して、フラクタル図形として描くグラフィックスである。両書ともBASICのプログラムが付いているから、これをＪのコードで書き直せばたやすいと最初は考えた。しかし、やってみるとBASICの入り組んだループ構造に振り回され、うまくいかなかった。結局、原理から理解し直して、Ｊで一からプログラミングすることになった。そのために、処理の道筋もわかり、すっきりしたものになった。

１．正方形フラクタルの原理

処理過程を次のように分けて考えると分かりやすい。

① スタートとして、まず画面の中央に大きく正方形を描く。… 0次の正方形

② 次に、今描かれた正方形の４つの頂点の一つ（右上）を取り上げ
これをを中心とし辺の長さを縮小して正方形を描く。… 1次の正方形群

③ つづいて、次の頂点（左上）を中心として、縮小した正方形を描く。

④ さらに、３番目の頂点（左下）を中心として、縮小した正方形を描く。

⑤ 最後に、４番目の頂点（右下）を中心として、縮小した正方形を描く。
これらは…… 2次の正方形群 を成す。

このように、正方形の中心をもとに、次数として分類した。

つまり各次数の正方形の数は

0次の正方形は、1個

1次の正方形群は、4個

2次の正方形群は、16個

3次の正方形群は、64個

と増えていく。

２．Ｊによるプログラミングを見直してみる－

本題に入る前に、ここでプログラミング作成の姿勢について、私なりの考えを少し述べさせていただきたいと思う。

Ｊのプログラムに限ったことではないが、プログラムを短くコンパクトにすることに何のメリットもない。

これまで、Ｊでは関数型言語の特徴から、例えば、ループなしで処理が出来るとかBASICなど他の言語よりＪのコーディングが短いことを誇りにしてきた。

私としては、Ｊのコーディングでも必要に応じて、ループ構造、if then else構造もとり入れて、BASIC流に、分かりやすくしたら良いと思う。

これまで、私はＪのループ構造として、もっぱら while. do. else. end. 構文だけを用いてきた。

Ｊのループ構造にforループ構文もあるが、あまり使いこなしていなかった。
forループ構造をつぎのようにBASIC流に対応させた構文として、読みやすくしてみた。
繰り返しインデックスも便利に利用できる。今回ここで積極的に使用してみた。

使い方を以下に示してみる。

```
NB. How_to_use of J for_loop
wr =: 1!:2&2

NB. for_loop of J = 0 ... N-1 ====
test0 =: 3 : 0
N =. y.
for_J. (i. N) do.
    wr (":J), ' ', (J#'A')
end.
'** end **'
)
NB. for_loop of J = 1 ... N ====
test1 =: 3 : 0
N =. y.
for_K. (>: i. N) do.
   wr  (":K), ' ', (K#'A')
end.
'** end **'
)
```

0から4までの繰り返しループ

```
   test0 5
0
1 A
2 AA
3 AAA
4 AAAA
** end **
```

1から5までのの繰り返しループ

```
   test1 5
1 A
2 AA
3 AAA
4 AAAA
5 AAAAA
** end **
```

２重ループはつぎのようになる

```
loop =: 3 : 0
'N M' =. y.
for_I. i. N do.
  for_J. i. M do.
    wr I, J
  end.
end.
'*** end ***'
)
   loop 2 4
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
*** end ***
```

３．Ｊによる正方形フラクタルのプログラム

フラクタル図形の描画はWindowsのgl2グラフィックスを用いて行った。

このような問題では解析幾何としての座標の計算とコンピュータ画面上のグラフィックスのピクセル値と２つの計算を行わなくてはならない。

一般にコンピュータ画面では文章の横書きに合わせて、画像のピクセル値も左上から右下へ増える数値で示される。幸いにJ4のgl2では左下を(0, 0)とし、右上を(1000, 1000) となるよう設定できるので、このようにした。

一方、解析幾何学では座標(x, y)のとり方は中央を原点(0,0)とし、右方向を x > 0、上方向をy > 0とする。その変換調整が必要であるが、次の関数adj で行った。

```
adj =: 500&+@(250&*)
```

３．０　0次の正方形

まず、スタートとなる0次の正方形を次のようにして描く。

中心を(0, 0)として、一辺の長さ2の正方形の４つの頂点座標値は　(1, 1), (_1, 1), (_1, _1), (1, _1)となるので、これに最初の座標値を付加して、関数adjを用いてgl2により閉じた正方形が描かれる。

```
gllines adj (1, 1), (_1, 1), (_1, _1), (1, _1), (1, 1)
```

この操作を一般的に行うため、つぎのような関数squareを定義した。

```
square =: 3 : 0
'x0 y0 a' =. y.
a2 =. a % 2
p1 =. (x0 + a2), (y0 + a2)
p2 =. (x0 - a2), (y0 + a2)
p3 =. (x0 - a2), (y0 - a2)
p4 =. (x0 + a2), (y0 - a2)
SQ =: p1;p2;p3;p4
)
```

これは、Windowsグラフィックスの[Run0]ボタンを押すことで実行する。

```
fract_Run0_button=: 3 : 0
glrgb 0 0 0
glpen 1 0
XY0 =. square 0, 0, 2
XY0DA =: XY0  NB. Global Value
XY0 =. XY0, (0{XY0)
gllines adj ;XY0
glshow ''
)
```

３．１ １次の正方形フラクタル

次に１次の正方形フラクタルを描く。

これには、0 次の正方形４つの頂点座標値をそれぞれ中心として、一辺の長さ１（縮小率 0.5）で正方形を４つ描く。

```
fract_Run1_button=: 3 : 0
XY1 =. square 1,1,1
XY1 =. XY1, (0{XY1)
gllines adj ;XY1
XY2 =. square _1, 1, 1
XY2 =. XY2, (0{XY2)
gllines adj ;XY2
XY3 =. square _1, _1, 1
XY3 =. XY3, (0{XY3)
gllines adj ;XY3
XY4 =. square 1, _1, 1
XY4 =. XY4, (0{XY4)
gllines adj ;XY4
glshow ''
)
```

これに、先に紹介した J の for ループ構文を用いてみよう。

ここでのポイントは先の関数 fract_Run0_button を実行したとき、0 次の正方形の４つの頂点位置をグローバル値 XY0DA とし、それをここでループ構造のインデックスとして呼び出し使用していることである。

また同様にして、グローバル値 XY1DA の 16 個の値を次の関数使用に準備している。

```
fract_Run1_button=: 3 : 0
XY1DA =: ''
for_J. (i. 4) do.
  XYXY =. square (; J{XY0DA), 1
  XY1DA =: XY1DA, XYXY  NB. Global Value
  gllines adj ; XYXY, ({. XYXY)
end.
glshow ''
)
```

[Run1]ボタンを押すことで実行する。なお、比較のためにループを使わないで、直接描くプログラムは[Run10]ボタンで実行できる。最後のリスティングに示した。

３．２ ２次の正方形フラクタル

２次の正方形フラクタルでは、もうお分かりのように先の１次の16個の正方形の頂点座標、グローバル値XY1DAを利用して、forループを用いて描いている。

```
fract_Run2_button=: 3 : 0
for_K. (i.16) do.
  XY2DA =: ''
  XYXY =. square (; K{XY1DA), 0.5
  XY2DA =: XY2DA, XYXY
  gllines adj ; XYXY, ({. XYXY)
end.
glshow ''
)
```

実行はボタン[Run2]を押すことで行う。ループによらない直接実行は[Run20]ボタンで行える。当然のこととして、Run20のＪのコーディングは大変な長いコーディングになっている。

３．３ ３次の正方形フラクタル

さらに、3次の正方形フラクタルは、64回のforループを用いて描いている。
プログラムは次のとおりである。

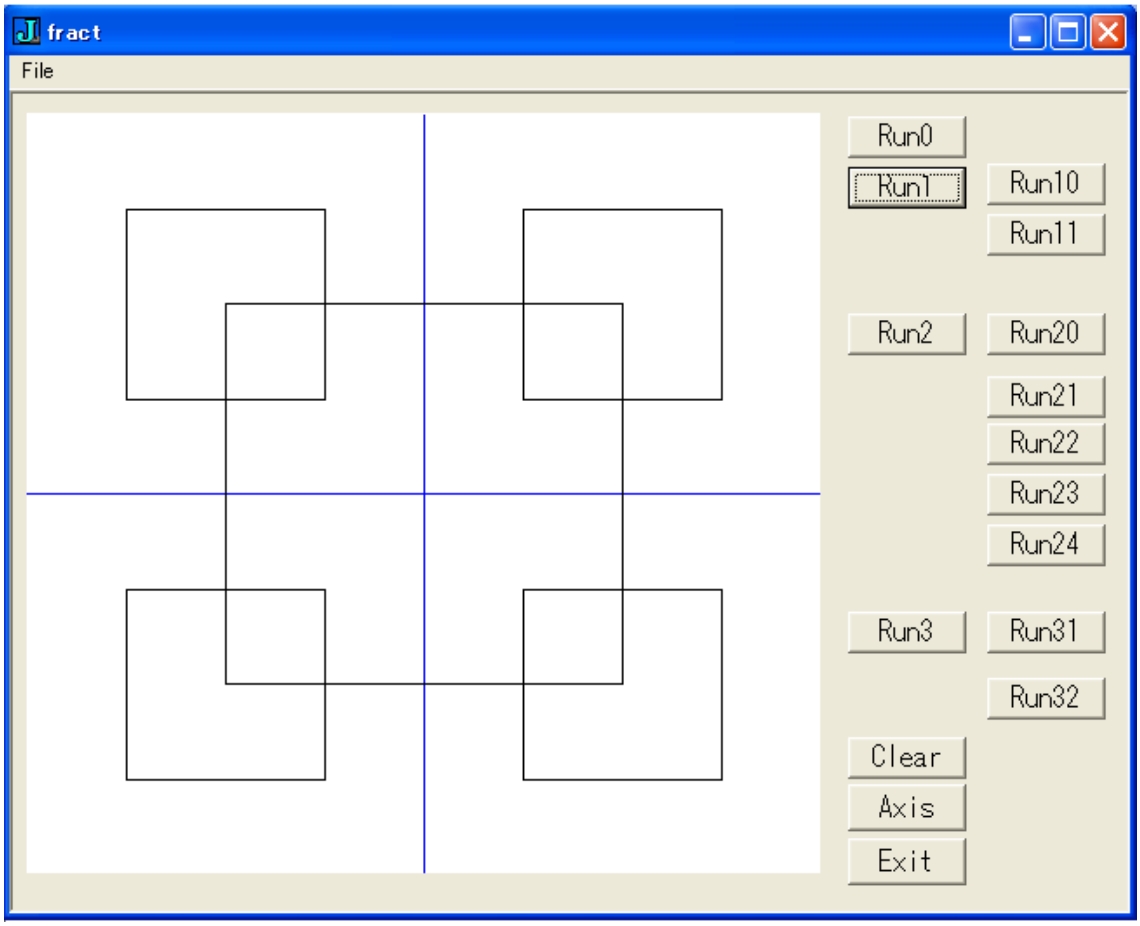
```
fract_Run3_button=: 3 : 0
XY3DA =: ''
for_L. (i.64) do.
  XYXY =. square (; L{XY2DA), 0.25
  XY3DA =: XY3DA, XYXY
  gllines adj ; XYXY, ({. XYXY)
end.
glshow ''
)
```

４．正方形フラクタルの実行例

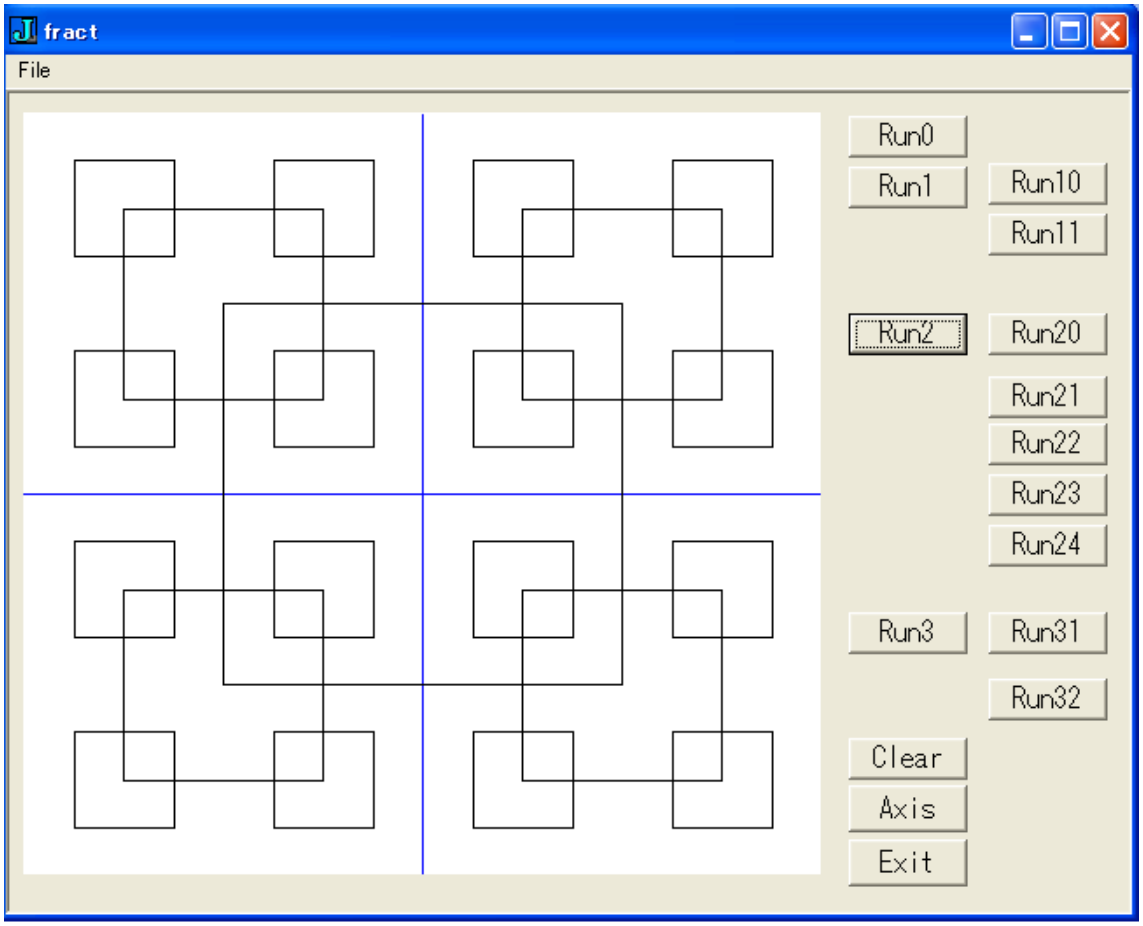
プログラムの実行はＪを起動したijx画面より、load nfractsq.ijsをロードしてrun ’’ によりウィンドウグラフィックスの画面が現れる。

ボタンRun0 に続き、ボタンRun1を押した状態は次のようである。

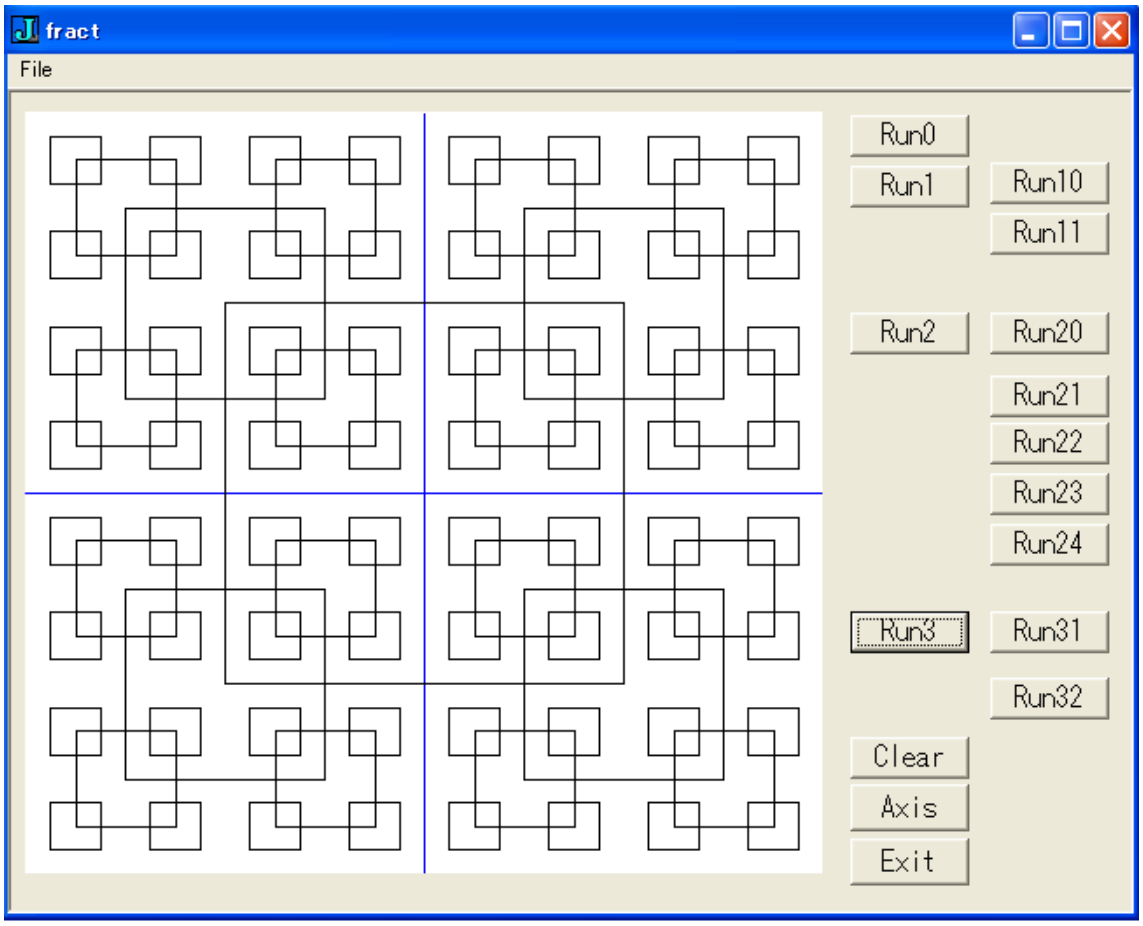
0次正方形と１次正方形フラクタルまでのグラフィックス表示の様子である。



つぎに続いて、ボタンRun2を押すと、2次正方形フラクタルまでのグラフィックス画面が表示される。



さらに、続けて　ボタン Run3 を押すと 3 次正方形フラクタルまでのグラフィックス画面が表示される。



５．おわりに

Ｊで正方形フラクタル図形のグラフィックス表示のプログラムを構築した。

BASIC のプログラムがあるので、それをＪでプログラミングし直せば容易にできると最初は考えた。一ヶ月以上、いじくりまわしたが、結局混沌としてしまってダメだった。俗な言葉で言えば、「他人のふんどしで、すもうはとれない」ということだ。

原理をしっかりと理解して、それをじっくりとていねいにＪでプログラムを作ることである。スマートな例えば再帰的プログラムなどは危険である。ループを使って確実に積み上げていくことである。プログラミングとはエレガントなプログラムより、ださい工学的な積み上げ方式の実用のプログラムがやはり大切である。

６．さらにおまけに　－　円のフラクタル

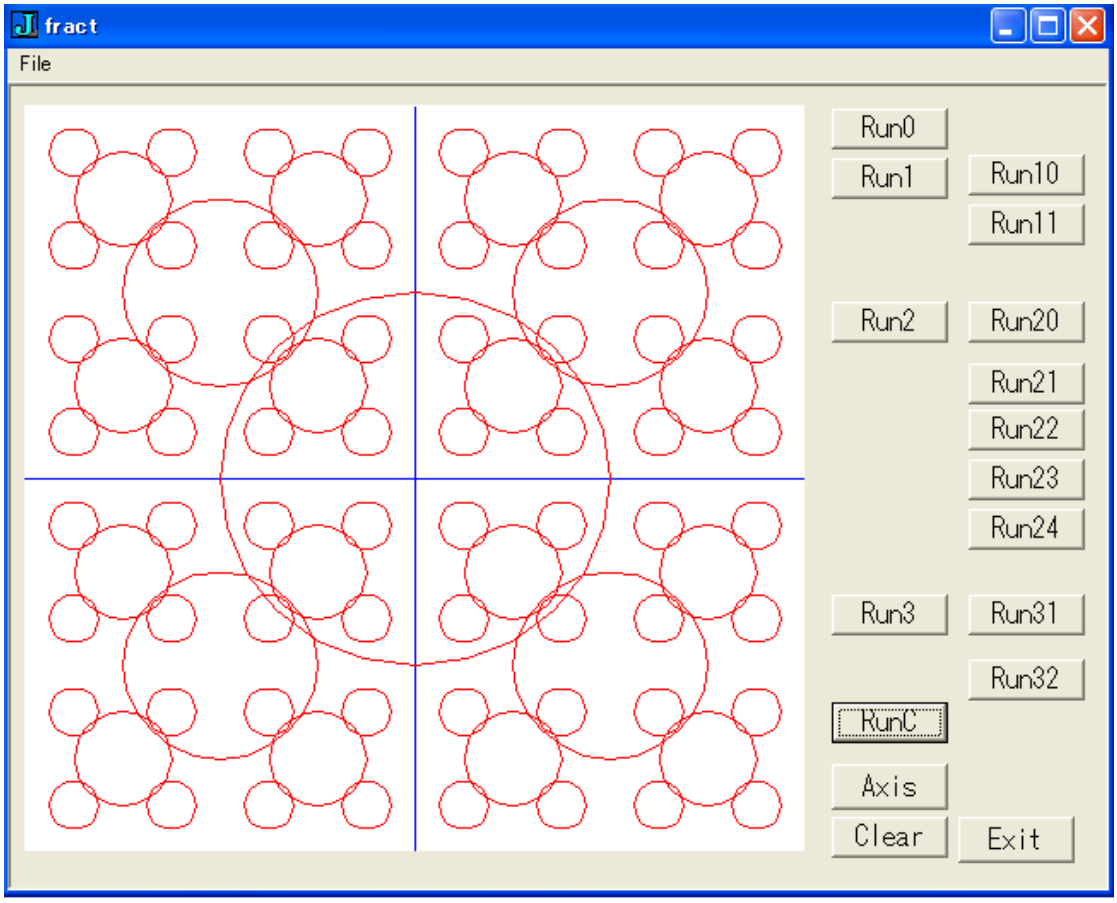
同様の発想で、多数の円から成るフラクタルをおまけとして、描いてみた。
円を描くには、glellipseがあるが、これでは円の内部を塗りつぶしてしまう。
従って、円の周囲の座標を細かく多角形で近似して描いた。計算を行う関数circleの定義は次のようになる。

```
circle =: 3 : 0
'X Y A' =. y.
, (X + A*(cos (i.25)*(1r12p1)) ) ,. (Y + A*(sin (i.25)*(1r12p1)) )
)
```

0次、１次、２次、３次の円フラクタルを描く関数定義は以下のようである。

```
fract_RunC_button=: 3 : 0
glrgb 255 0 0
glpen 1, 0

NB. 0 order circle ================================
gllines adj , circle 0, 0, 1

NB. 1 order circles ===============================
for_J. (i. 4) do.
  gllines adj , circle , (; J{XY0DA), 0.5
end.

NB. 2 order circles ===============================
for_K. (i. 16) do.
  gllines adj , circle , (; K{XY1DA), 0.25
end.

NB. 3 order circles ===============================
for_L. (i. 64) do.
  gllines adj , circle , (; L{XY2DA), 0.125
end.
```

円フラクタルの実行は、まず正方形を描いてから、ボタン RunC を押せば、３次までの円フラクタルが重ねて描かれる。各次数の正方形の頂点のグローバル値を利用しているからである。円フラクタルだけを描くには、一度ボタン Clear により消してから、ボタン RunC を押せばよい。



Jのプログラム・リスティング

```
NB. nfractsq.ijs 2021/3/12
wr =: 1!:2&2
require 'numeric'
require 'trig'
require 'gl2'

FRACT=: 0 : 0
pc fract;
menupop "File";
menu new "&New" "" "" "";
menu open "&Open" "" "" "";
menusep ;
menu exit "&Exit" "" "" "";
menupopz;
xywh 4 5 228 194;cc gfract isigraph;
xywh 240 176 34 12;cc ok button;cn "Axis";
xywh 277 190 34 12;cc cancel button;cn "Exit";
xywh 240 190 34 11;cc Clear button;
xywh 240 6 34 11;cc Run0 button;
xywh 240 19 34 11;cc Run1 button;
xywh 280 18 34 11;cc Run10 button;
xywh 280 31 34 11;cc Run11 button;
xywh 240 56 34 11;cc Run2 button;
xywh 280 56 34 11;cc Run20 button;
xywh 280 72 34 11;cc Run21 button;
xywh 280 84 34 11;cc Run22 button;
xywh 280 97 34 11;cc Run23 button;
xywh 280 110 34 11;cc Run24 button;
xywh 240 132 34 11;cc Run3 button;
xywh 280 132 34 11;cc Run31 button;
xywh 280 149 34 11;cc Run32 button;
xywh 240 160 34 11;cc RunC button;
pas 6 6;pcenter;
rem form end;
```

```
)
run =: fract_run
fract_run=: 3 : 0
wd FRACT
NB. initialize form here
wd 'pshow;'
)

fract_close=: 3 : 0
wd'pclose'
)

fract_cancel_button=: 3 : 0
fract_close''
)

fract_Clear_button=: 3 : 0
glclear ''
glshow ''
)

fract_ok_button=: 3 : 0
glrgb 0 0 255
glpen 1 0
gllines 0 500 1000 500
gllines 500 0 500 1000
glshow ''
)

adj =: 500&+@(250&*)

square =: 3 : 0
'x0 y0 a' =. y.
a2 =. a % 2
p1 =. (x0 + a2), (y0 + a2)
p2 =. (x0 - a2), (y0 + a2)
```

```
p3 =. (x0 - a2), (y0 - a2)
p4 =. (x0 + a2), (y0 - a2)
SQ =: p1;p2;p3;p4
)

NB. 0 order square / start ========================
fract_Run0_button=: 3 : 0
glrgb 0 0 0
glpen 1 0
XY0 =. square 0, 0, 2
XY0DA =: XY0
gllines adj ; XY0, (0{XY0)
glshow ''
)

NB. 1 order square fractals =======================
fract_Run1_button=: 3 : 0
XY1DA =: ''
for_J. (i. 4) do.
  XYXY =. square (; J{XY0DA), 1
  XY1DA =: XY1DA, XYXY
  gllines adj ; XYXY, ({. XYXY)
end.
glshow ''
)

fract_Run10_button=: 3 : 0
XY11 =. square 1,1,1
XY11 =. XY11, (0{XY11)
gllines adj ;XY11
XY21 =. square _1, 1, 1
XY21 =. XY21, (0{XY21)
gllines adj ;XY21
XY31 =. square _1, _1, 1
XY31 =. XY31, (0{XY31)
gllines adj ;XY31
```

```
XY41 =. square 1, _1, 1
XY41 =. XY41, (0{XY41)
gllines adj ;XY41
glshow ''
)

NB. 2 order square fractals ======================
fract_Run2_button=: 3 : 0
XY2DA =: ''
for_K. (i.16) do.
  XYXY =. square (; K{XY1DA), 0.5
  XY2DA =: XY2DA, XYXY
  gllines adj ; XYXY, ({. XYXY)
end.
glshow ''
)

NB. 3 order square fractals ======================
fract_Run3_button=: 3 : 0
XY3DA =: ''
for_L. (i.64) do.
  XYXY =. square (; L{XY2DA), 0.25
  XY3DA =: XY3DA, XYXY
  gllines adj ; XYXY, ({. XYXY)
end.
glshow ''
)
```

```
NB. draw circle ========================================
circle =: 3 : 0
'X Y A' =. y.
, (X + A*(cos (i.25)*(1r12p1)) ) ,. (Y + A*(sin (i.25)*(1r12p1)) )
)

fract_RunC_button=: 3 : 0
glrgb 255 0 0
glpen 1, 0

NB. 0 order circle ==================================
gllines adj , circle 0, 0, 1

NB. 1 order circles =================================
for_J. (i. 4) do.
  gllines adj , circle , (; J{XY0DA), 0.5
end.

NB. 2 order circles =================================
for_K. (i. 16) do.
  gllines adj , circle , (; K{XY1DA), 0.25
end.

NB. 3 order circles =================================
for_L. (i. 64) do.
  gllines adj , circle , (; L{XY2DA), 0.125
end.
)
```