J のグラフィックス（C.Reiter 編）(J6 版)

SHIMURA Masato
JCD02773@nifty.ne.jp

2011 年 11 月 25 日

目次

概要

C.Reiter の名著 [Fractal Visualization and J] は第 3 版 (2008) が出版されており、J にも Fractal のチュートリアルが入っている。これらで C.Reiter は J のキャンバスや描画関数を発展させている。

1 C.Reiter **のグラフィックス** (1)(dwin)

C.Reiter は J のグラフィックスに携わっているようだ。dwin は C.Reiter による J の addon 版である Studio/Lab の Graphics/dwin2 を試してみよう。
Script は/addons/graphics/fvj3/dwin2.ijs にあるのでロードする

1.1 キャンバス

フリーサイズ キャンバスはフリーサイズで、大きさは最初に **dwin** で指定する。

- 1000 × 1000 のキャンバス 1000 号でも、キャンバス 1 号、2 号でも指定できる。

```
0 0 1000 1000 dwin 'Object window'
_1 _1 1 1 dwin ''
```

- サイズは左引数 (x) で与える。右引数の文字列はキャンバスの外枠に表示される文字で、null('') ならばサイズが表示される。
- キャンバスは必ず先に出しておかなければならない。
- 特段オブジェクトを意識しなくてもキャンバスは数枚同時に表示できる
- 重ね描きができる

- キャンバスは白地
  ```
  0 0 1000 1000  dwin '1000 window'
  ```
- **始点**は左下、**終点**は右上である。(gdxxx と同じで、1000 号キャンバスでは J6 でなく J4,J5 に戻っている)

キャンバスの Script　　C.Reiter によるキャンバスの定義

```
dwin=: 3 : 0           NB. pixel sized window drawing
0 0 1 1 dwin y
:
'a c b d'=.x        NB. note non-alphabetic order
SC=:WIN_WH&*@((-&(a,d)) %((b-a),c-d)"1)"1   NB. global definition
sz=.":WIN_WH
PN=:y,';   Window bounds are x: ',(":a),' to ',(":b),'   y: ',(":c),' to ',":d
nx_WIN ''
z=.'pc ',WIN_nam,' closeok;pn "',PN,'";cc g isigraph;setxywhx g 2 2 ',sz,';pas 2 2'
wd z,';pshow;'
)
```

1.2 描画関数

J の描画関数に *dxxx* を付ける

color .

```
NB. -----------------------------------------------
dfillcolor=: 3 : 0     NB. Start Fill Color
wd 'psel ',WIN_nam
if. _1 e.y do. glbrushnull ''
   else. glrgb y
   glbrush''
end.
)
```

dline .

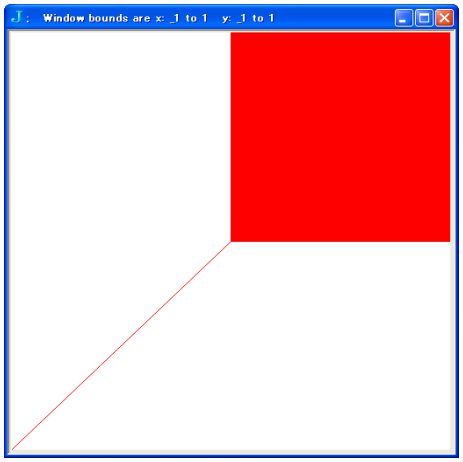
```
NB. -----------------------------------------------
NB.    255 0 0 dline 0 0 ,0 1,: 1 1
dline=: 3 : 0 "1 2
0 0 0 dline y
:
Y=.x:^:_1 SC 2{."1 y
wd 'psel ',WIN_nam
```

```
glrgb x
glpen 1 0
gllines ,Y
glpaint ''
)
```

- 次のキャンバスは *2* 号。 *(0,0)* は中心になる
  `_1 _1 1 1 dwin 'Object window'`
- 次は左下 → から右上となる
  `255 0 0 dline _1 _1,:1 1`
- データは縦 *2* 列で与える

dpoly ポリゴンを描く

```
NB. ------------------------------------------------
NB. e.g.:   255 0 0 dpoly 0 0,100 0,:100 100
dpoly=: 3 : 0 "1 2
0 0 0 dpoly y
:
Y=.x:^:_1 SC 2{."1 y
wd 'psel ',WIN_nam
dfillcolor x
glpolygon ,Y
glpaint''
)
```



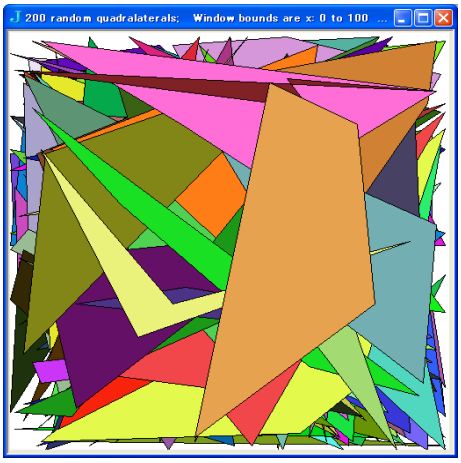

```
  _1 _1 1 1 dwin ''
255 0 0 dline _1 _1 ,0 0 ,0.7 0.5,:1 1
255 0 0 dpoly sq
```

- データは縦型で与える。
  `]sq=:#:0 1 3 2` NB. 2 **進法で表す**

```
0 0
0 1
1 1
```

```
1 0

   255 0 0 dline _1 1,:1 _1
   0 255 0 dpoly sq
```

- 簡易動画にもなる。動く万華鏡




```
0 0 100 100 dwin '200 random quadralaterals'
 (?200 3$256) dpoly ?200 4 2$100
```

4×2 のマトリクスが *200* 個　*200*

color は　256×2 = 1600 万色のテーブルを乱数で *200* 個作成

pixel　*dwin* は *pixel* も表示できる

```
NB. ------------------------------------------------
NB. e.g.:   0 255 0 dpixel 0 0,100 0,:100 100
dpixel=: 3 : 0    NB. Show pixel
0 0 0 dpixel y
:
wd 'psel ',WIN_nam
glrgb x
Y=.x:^:_1 SC 2{."1 y
glpixel Y
glpaint ''
)

 0 0 200 200 dwin''
   255 0 0 dpixel  ? 1100 2 $ 100
```

pixel は小さいのでたくさん打たないと視認できない

2　C.Reiter の Fractal Visualization の基本テクニック

2.1　カラパレットの作成

RGB の構成　０－７を *2* 進法で表示し、１を *255* に変換すると基本色が現れる。（１で表示する方法もある）

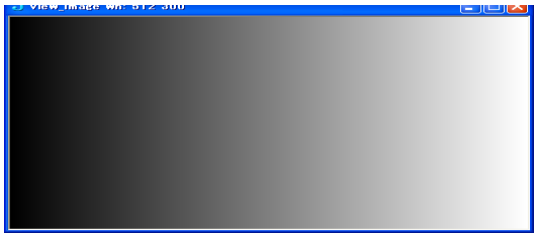
```
   (#:i.8){0 255
-------------------
  0   0   0
  0   0 255
  0 255   0
  0 255 255
255   0   0
255   0 255
255 255   0
255 255 255
```

0	0	0	Black	
0	0	255	Blue	B
0	255	0	Green	G
0	255	255	Cyan	
255	0	0	Red	R
255	0	255	Magenta	
255	255	0	Yellow	
255	255	255	White	

白黒 256 階調　.

`p0=:3#"0  i.256  NB. color table` *0-255 (i.256)* を諧調として *RGB3* 列に同じものをコピーする *(3 # ”0)*

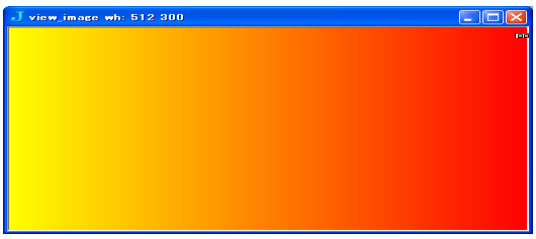
```
p0=:3#"0  i.256  NB. gray scale  palette
table=: 300 # ,:2#i.256 NB. 512*300
```



```
 view_image p0;table
512 300 NB. matrix size

 5# ,: 2#i.5  NB. mini table
0 0 1 1 2 2 3 3 4 4
0 0 1 1 2 2 3 3 4 4
0 0 1 1 2 2 3 3 4 4
0 0 1 1 2 2 3 3 4 4
0 0 1 1 2 2 3 3 4 4
```

RGB の 256 諧調　　*RGB* を *256* 階調にすると *1600* 万色となる

- *Green(0 255 0)* の双方の *0* を *i.256* にすると Green → White の諧調パレットになる
  `view_image  ((i.256),.255,.i.256);table`
- *Green* の *255* を *i.256* にすると Green → Black の諧調パレットになる
  `view_image  (0,.(i.256),.0);table`
- Yellow → Red
  RGB の *G* の *256* 階調を反転させると Yellow → Red の *UP*、反転させないと Red → Yellow の *Down* の諧調になる




```
view_image  (255,.(|. i.256),.0);table
```

- Blue → Green の諧調

```
view_image  (0,.(i.256),.|. i.256);table
```

2.2 ポリゴンとライン

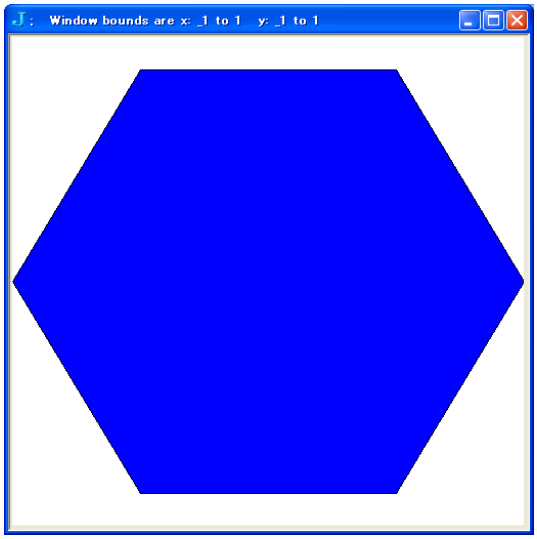
dwin 上で多角形をポリゴンとラインで描く。タートルグラフィックスは極座標を用いているので、進む距離と方向を示せば描けるが、ポリゴンやラインは各頂点の *x*,*y* 座標を順に示して連結する。

ポリゴン . 多角形の座標

```
calc_polygon 6
```

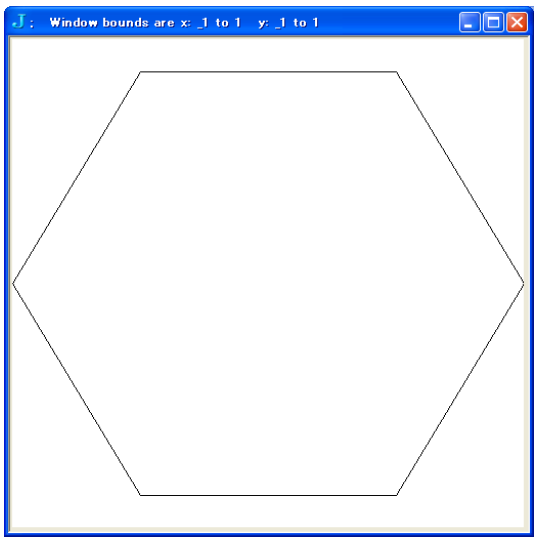
```
   1          0
 0.5   0.866025
_0.5   0.866025
  _1 1.22465e_16
_0.5  _0.866025
 0.5  _0.866025
```

```
NB. draw triangle square pentagon .........
calc_polygon=: 3 : '+. r. 2p1 * (i.y) %y' NB. 2pi % n

calc_polygont=: +.@r.@ *&2p1&(i.@] % ])   NB. tacit
  NB. calc_polygon 6
```




```
 0 0 255  draw_shape 6

draw_shape=: 4 : 0
tmp0=. calc_polygon y
tmp1=.find_maxmin tmp0
(1 3 0 2 {;tmp1) dwin ''
x dpoly tmp0
)
```

ライン ラインで多角形を描く場合に、ポリゴンは終点から始点への自動連結機能が入っているが、ラインのデータは終点から始点への連結を付加えなければならない。




```
draw_line=: 3 : 0
NB. 255 0 255 draw_figure 6
tmp0=.({. tmp0),~ tmp0=. calc_polygon y
tmp1=.find_maxmin tmp0
(1 3 0 2 {;tmp1) dwin ''
0 0 0  dline tmp0
)
```

2.3 北斎の井筒万字

北斎のデザインした井筒万字を描く。最初にポリゴンの下準備

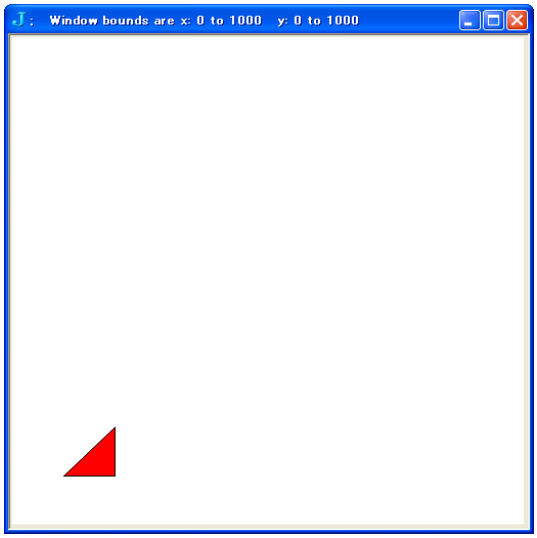
ポリゴン　ポリゴンの定義法。方眼紙を用意し、マスターピースを手書きして、角の *xy* 座標を得る

画面　画面は左下から右上へ

```
0 0  1000 1000 dwin ''
 255 0 0 dpoly 100 100 ,200 100,: 200 200
```

x	y
100	100
200	100
200	200

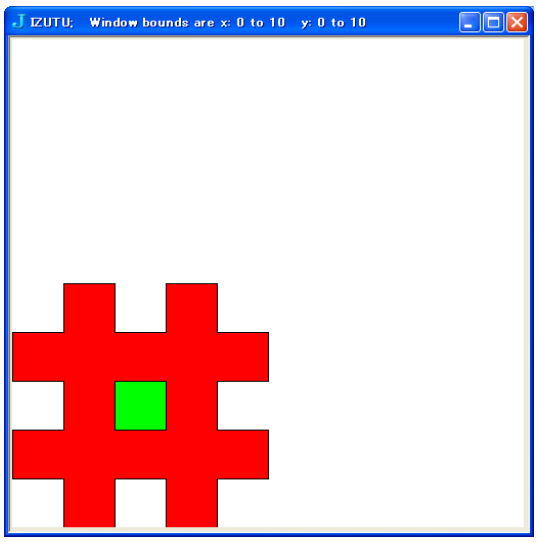
左下が始点で左回り。ポリゴンは終点から始点へは自動連結してくれる（ラインはしない）



北斎の新形小紋帳に描かれている井筒万字を描く

北斎の井筒万字　書の始点 *(左上 (0 4))* から外延のポイントを左回りに *(x,y)* を筆順になぞる。中の正方形は別のポリゴンで描く

最初のピース　２種類のポリゴンによる寄せ木細工またはジグゾウパズル



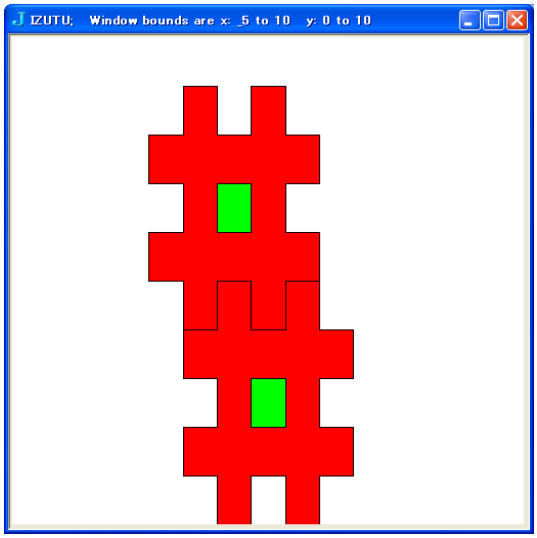

```
0 255 0 dpoly IM2
255 0 0 dpoly  IM
0 255 0 dpoly IM2
```

NB. **井筒組の万字**

```
IM0=:0 4,0 3,1 3,1 2,0 2,0 1,1 1,:1 0
IM1=:2 0,2 1,3 1,3 0,4 0,4 1,:5 1
IM2=:5 2,4 2,4 3,5 3,5 4,4 4,:4 5
IM3=:3 5,3 4,2 4,2 5,1 5,:1 4
IM=: IM0,IM1,IM2,IM3
IM2=: 2 3,2 2,3 2,: 3 3
```

組み木　上に一個積む *(*基準ポイント（*0 4*）からの移動量 *(_1 4)* を加える*)*

ボックスを用いて一度に描画すると効率的である




```
 _5 0   20 20 dwin 'IZUTU'

255 0 0 dpoly L:0 IM + "1 L:0 (0 0;_1 4)
0  255 0 dpoly L:0 IM2 + "1 L:0 (0 0;_1 4)
```

寄せ木（又はタイル）　　ボックスを用いて一度に計算するためのポイント表と差分表を作成する

_3	16	1	17	5	18	9	19
_2	12	2	13	6	14	10	15
_1	8	3	9	7	10	11	11
0	4	4	5	8	6	12	7

左下がマスターピースで上と右に貼り付けていく。

原点は（*0,4)*

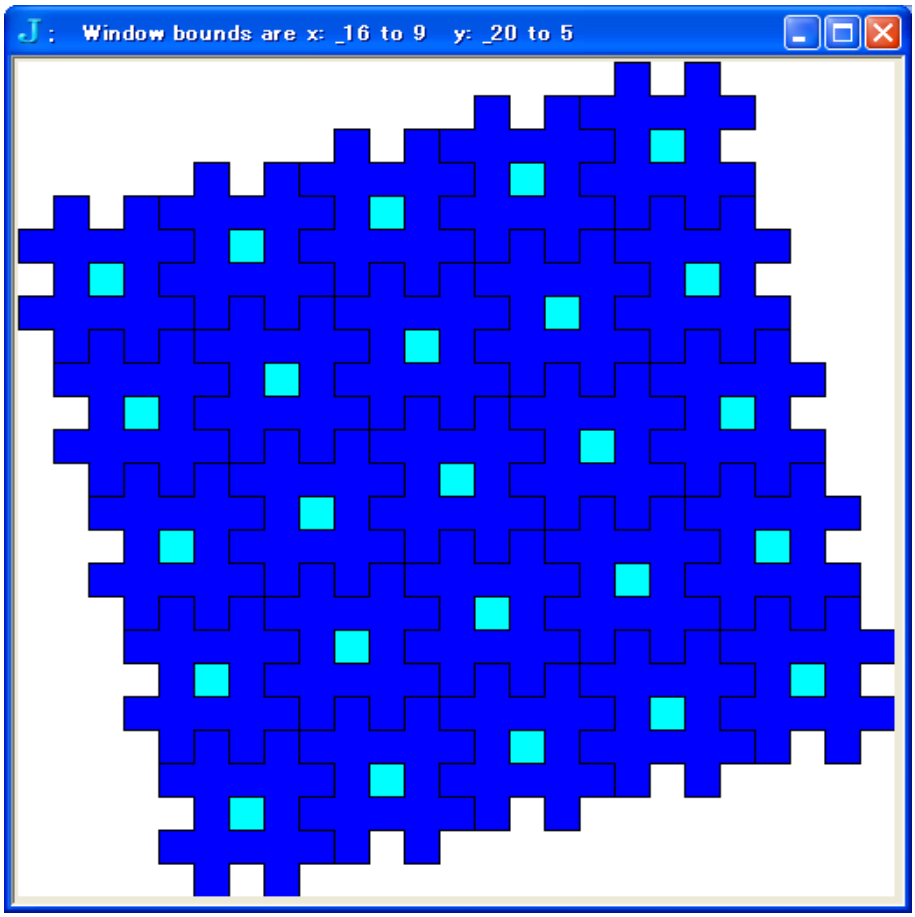
16 **ピース**　.

IM_16

_3	12	1	13	5	14	9	15
_2	8	2	9	6	10	10	11
_1	4	3	5	7	6	11	7
0	0	4	1	8	2	12	3

差分表はポイント表から原点 *(0,4)* を差し引く。

```
_5 0   20 20 dwin 'IZUTU'
 255 0 0 dpoly L:0 IM + "1 L:0 IM_16
0  255 0 dpoly L:0 IM2 + "1 L:0 IM_16
```



3 図形の線形変換 Homogeneous Coordinates

図形の伸縮、回転、移動は次のように行える。3×3 のマトリクスを用いる場合は ***Homogeneous Coordinates*** と呼ばれる。本来 *3* 次元の回転であるが *z* 軸を固定した場合に相当し *2* 次平面で作用する。

- 葉書を串で突き刺して回した場合に相当
- 計算は内積演算　`mp=: +/ . *`
- ポリゴンやラインの *x,y* のデータには *z* 軸にあたる列に *1* を加えて計算してグラフィックスに渡す前に落とす

$$a =: \begin{matrix} 100 & 100 & 1 \\ 200 & 100 & 1 \\ 200 & 200 & 1 \end{matrix}$$

(100 100,200 100,: 200 200),.1

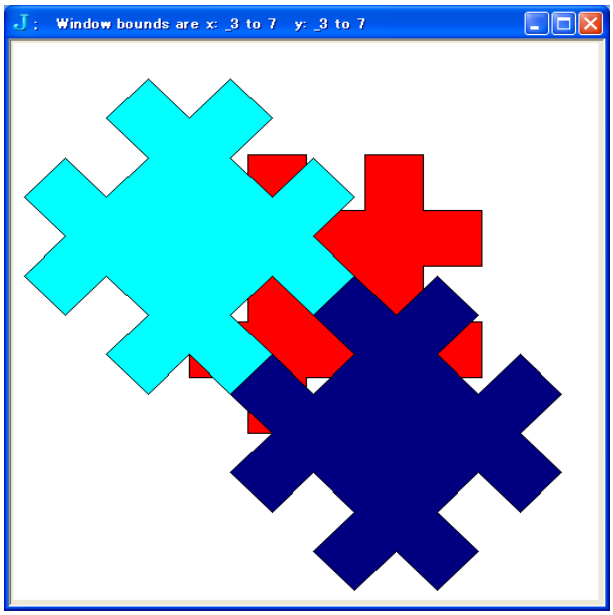
Script　同時変換の *Script*

```
elongm=: 3 :  '(y,1)* =i.3'
rotm=: (cos, sin,0:),(-@sin,cos,0:),: 0: ,0:,1:
transm=: 3 : '(=i.2), y,1'
mp=: +/ . *   NB. inner products
```

回転 .

$$(x,y,1)_{new} = (x,y,1)\begin{pmatrix} \cos(t) & \sin(t) & 0 \\ -\sin(t) & \cos(t) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
   rotm 1r4p1
 0.707107 0.707107 0
_0.707107 0.707107 0
        0        0 1
```



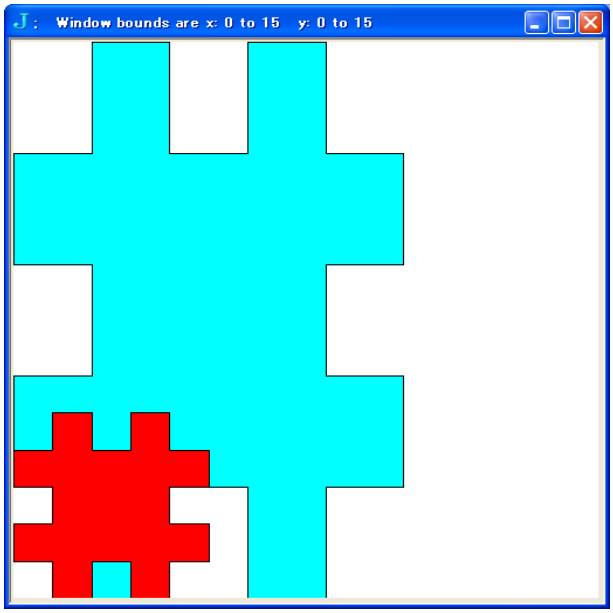

```
 _3 _3 7 7 dwin ''
255 0 0 dpoly IM
0 255 255 dpoly }:"1 (IM,.1) mp rotm 1r4p1
0 0 128 dpoly }:"1 (IM,.1) mp rotm _1r4p1
```

1r4p1 は $\frac{1}{4}\pi$

伸縮　横に 2 倍、縦に 3 倍に伸ばす。縮小はマイナスで

$$(x,y,1)_{new} = (x,y,1)\begin{pmatrix} r & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
   elongm 2 3
2 0 0
0 3 0
0 0 1
```

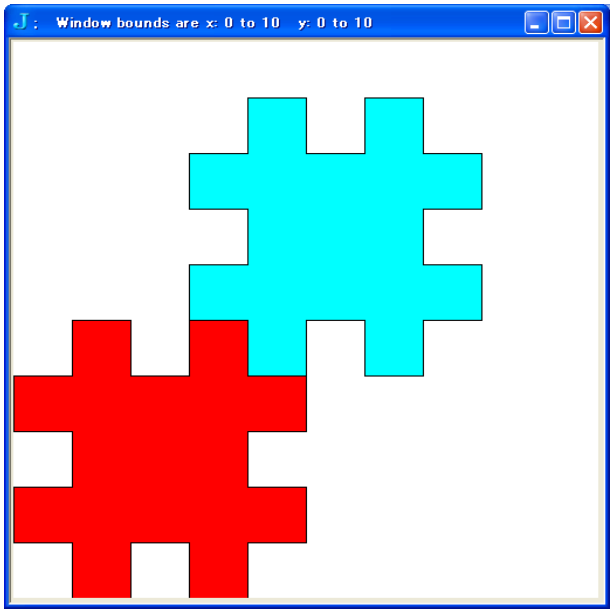



```
 0 0 15 15 dwin ''
0 255 255 dpoly }:"1 (IM,.1) mp elongm 2 3
255 0 0 dpoly IM
```

移動　*x,y=(3 4)* に移動

$$(x,y,1)_{new} = (x,y,1)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{pmatrix}$$

```
    transm 3 4
1 0 0
0 1 0
3 4 1
```

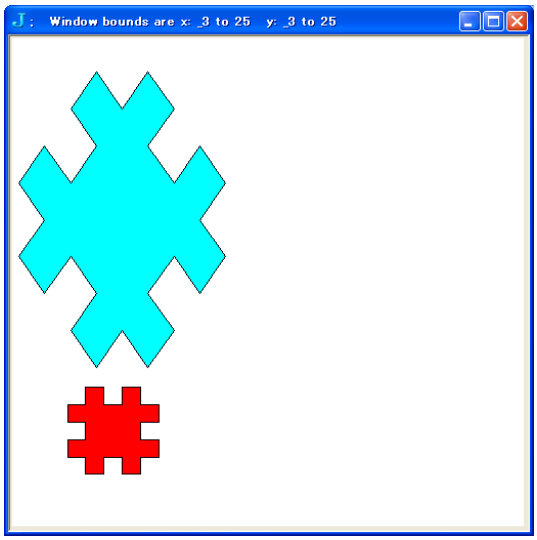



```
0 0 10 10 dwin ''
255 0 0 dpoly IM
0 255 255 dpoly a=. }:"1 (IM,.1)mp transm 3 4
```

全部まとめて　全部纏めて一つの変換マトリクスにすることもできる

```
 a=.  (rotm 1r4p1) mp (elongm 2 3) mp transm 3 4
 1.41421 2.12132 0
_1.41421 2.12132 0
```

3 4 1




```
 _3 _3 25 25 dwin ''
255 0 0 dpoly IM
0 255 255 dpoly a1=. }:"1 (IM,.1)mp a
```

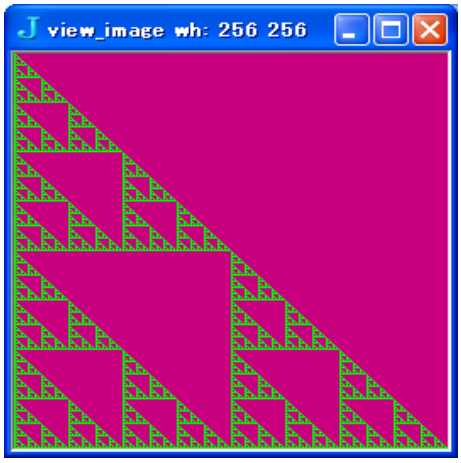
4 C.Reiter のグラフィックス (2)-ピクセルで描く

同じく *C.Reiter* の *addon* に入っている *raster6* の *vwin* を用いて *view_image* と *vpixel* で描く
フラクタルはピクセルを多用する。
Studio/Lab の *Graphics/raster6* を試してみよう。*Script* は*/addons/graphics/fvj3/raster6.ijs* にあるのでロードする

vwin .

- 窓は次で出てくる
  0 0 2 2 vwin ”

view_image




```
p=. 200 0 128,: 0 255 0 NB. color palette
b=. (,,.~)^:8 ,1        NB. f(x) and iteration

   view_image p;b
256 256        NB. output size

NB. Sierpinski
   ] b=. (,,.~)^:2 ,1
1 0 0 0
1 1 0 0
1 0 1 0
1 1 1 1
```

- *view_image* を用いる場合はキャンバスを最初に出しておく必要は無い
- *view_image* ではキャンバスのサイズは自動判別

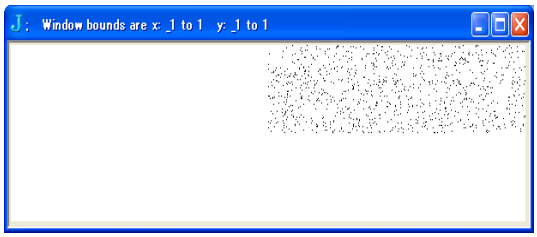
Script .

```
vwin=: 3 : 0
0 0 1 1 vwin y
:
'a c b d'=.x
SCvra=:<.@:((-&(a,c))"1 %(((b-a),d-c)%|.0.999999*|.VRAWH)"1)
if. a. e.~ {. y do.
sz=.":VRAWH
PN=:y,';   Window bounds are x: ',(":a),' to ',(":b),'   y: ',(":c),' to ',":d
nx_WIN ''
wd 'pc ',WIN_nam,' closeok;pn "',PN,'";cc g isigraph;setxywhx g 2 2 ',sz,';pas 2 2'
vclear ''
wd ';pshow;'
else. vclear '' end.
''
)
```

4.1 描画関数

キャンバスは *vwin dwin* どちらでも良い

vpixel .

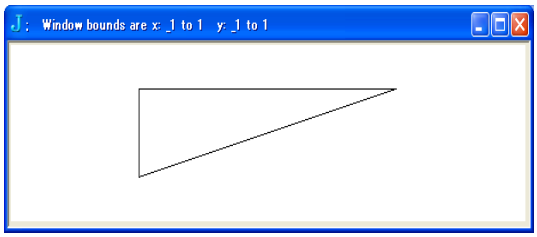


vpixel .

```
   _1 _1 1 1 vwin ''
   vpixel ? 1000 2 $0
vshow ''
```

vpixel には *vshow* ” が必要

vline .



vline .

```
_1 _1 1 1 vwin ''
 vline 0.5*1 1,_1 1,_1 _1 ,: 1 1
 vshow ''
```

4.2 IFS

IFS *Iteration Function System*

IFS という変換と *pixel* を多用した点描のグラフィックスがあり *pixel* で描く。*elongm rotm transm* で構成された変換マトリクスを用い、このマトリクスが決まったときには絶妙な美ができるがマトリクスのパラメー

ターを得るのは難関である。

2 の変換マトリクス　　*C.Reiter* に載っている *2* つの変換マトリクスで多重スパイラルを描く。

$$B0 = \begin{vmatrix} 0.2 & 0 & 0 \\ 0 & 0.2 & 0 \\ 0.4 & 0.8 & 1 \end{vmatrix} \qquad B1 = \begin{vmatrix} 0.904498 & 0.350404 & 0 \\ -0.350404 & 0.904498 & 0 \\ 0.222953 & -0.127451 & 1 \end{vmatrix}$$

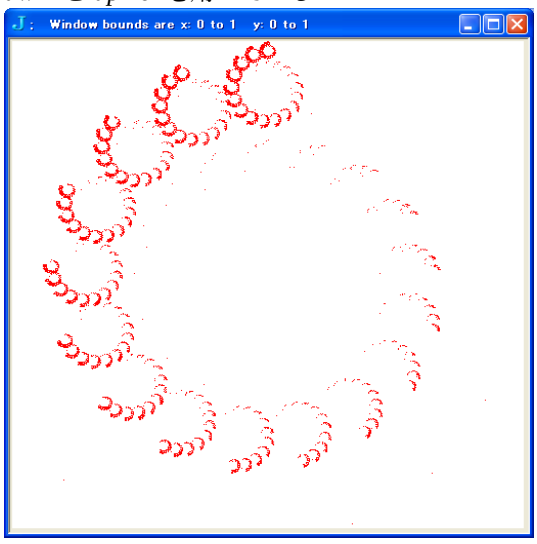
tacit　*3* 行で *IFS* の演算ができる。@. 以下は条件文

```
mp=: +/ . *  NB. Inner Products

b0=: mp&B0
b1=: mp&B1
spiral=: b0`b1@.(?@:2:)
NB. spiral ^:(i.15) 0.1 0.1 1
```

IFS の計算　*tacit* のループを用いる。

```
    spiral ^:(i.4) S=: 0.1 0.1 1
 0.1        0.1 1  NB. S
0.278362 _0.0019608 1  NB. S mp B1
0.455672   0.799608 1  NB. S mp B1 mp B0
0.354922   0.755462 1  NB. S mp B0 mp B0 mp B1
```

描画　*dwin* と *dpixel* を用いている




```
  0 0 1 1 dwin ''
a=.  spiral ^:(i.3000000) 0.1 0.1 1
  255 0 0 dpixel a
```

非力なノートでは *3* 百万回のループによる *300* 万個の点描画が限界である。

References

Crif Reiter [Fractal Visualization and J 3rd Edition] Lulu 2007

J602 J701 はトロントから *DL* 出来ます

`http://www.jsoftware.com`

スクリプトは次から *DL* できます

`http://japla.sakura/ne.jp` の *Workshop NOV 2011*