JAPLA 研究会資料 2007/4/28

シェルピンスキーから文字列フラクタルへ

西川　利男

０．はじめに

J 言語は配列処理に加えて、例えば Box(<), Rank('), Power(^:)など多くのユニークな機能を持ち、フラクタル処理プログラミングにとってまたとない強力なツールである。

Reiter の "Fractals Visualization" [1]はそのバイブルとも言うべきものであり、また筆者自身ももう１０年も前になるが、「初めてのフラクタル」[2]の翻訳以来ずっと興味を持ち続けている。先月の JAPLA の例会では、志村正人氏による「Coke on Coke」のフラクタル表示[3]は面白かったが、任意の文字列では現在のところ無理とのことであった。

筆者もこれに刺激されて、それでは任意の文字列のフラクタル表示は出来ないものか、と試みてみたのが今回の発表である。

１．文字列の画像データの作成

任意の文字列でのフラクタル表示を可能にするためには、適当な方法で文字列を画面表示するピクセルの画像データを作ることが必要である。

これを次のようにして行った。

・メモ帳（適当な Windows テキストエディタ）で例えば「JAPLA」のように打ち込んで文字列を表示させる。

・Prt Sc(＝print screen) を押し下げる。これにより文字列を表示した画面は画像データとしてクリップボードにコピーされる。

・ペイントを起動し、ツールバーの[編集]-[貼り付け]を行うと、クリップボードの画像が貼り付けられる。この後、ペイントの操作により文字列部分を切り出し、必要ならば整形する。これを[ファイル]-[名前を付けて保存]により、標準の 24 ビット BMP ファイルとして、例えば「J24.bmp」として保存する。

[1] C.A.Reiter, "Fractals Visualization and J", Iverson Software Inc. (1995)

[2] H.Lauwerier 西川利男訳「初めてのフラクタル」丸善(1996)

[3] 志村正人「 J のグラフィックス Pixel(J6 版)」JAPLA 研究会資料 2007/3/22

[4] 西川利男「 J による DEBUG コマンド・プログラムと BMP ファイルの解析への適用 APL・J シンポジウム資料 2001/12/4

２．ＪによるBMPファイルの読み込み、書き込み、表示などの操作

続いて、先の画像ファイルのデータを元にJで処理を行う。

BMPファイルの構造とＪによる解析については、以前報告したことがある。[4]

BMPファイルの構造は基本的には次のように成っている。

・ヘッダー部　ファイルの大きさ、画像の幅、高さなどの仕様の記述

・データ部　　１ピクセルをRGB値、３バイト（＝24ビット）で表した値の集合

Jで処理するためには、まず次のパッケージを読み込む。

```
require 'system\packages\graphics\bmp.ijs'
```

ここには、BMPのいろいろなユーティリティが整備されているので、これを用いる。

viewbmp, readbmp, writebmp, viewmat など

さらに次のようなJのプログラムを作った。例えば、以下のように実行される。

なお、プログラムリストは最後にまとめてあげた。

・readbmpN … BMPファイルを読み込みJとしての値を得る。readbmpを一部修正した。

・dcp …… 上のreadbmpNを用いてBMPファイルから０，１の配列パターンを得る。0白　1黒　(Display Character Pattern)

```
dcp 'user\J24.bmp'
```

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

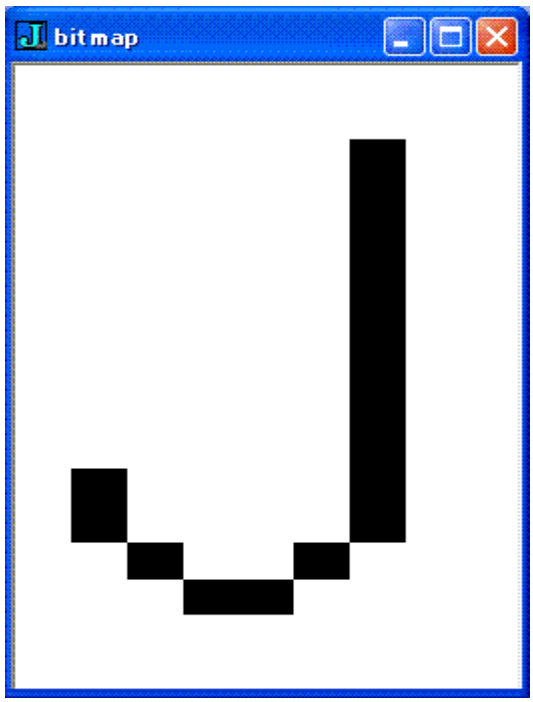
・z2inf … 0 の部分を _(inf) にして見やすくする。

z2inf dcp 'user\J24.bmp'

```
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ 1 _ _
_ _ _ _ _ _ 1 _ _
_ 1 _ _ _ _ 1 _ _
_ 1 _ _ _ _ 1 _ _
_ _ 1 _ _ 1 _ _ _
_ _ _ 1 1 _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
```

・viewmat … ウィンドウズ・グラフィックスとして見る。

viewmat -. dcp 'J24.mp'　　(-. による白黒反転が必要)



３．自己複写(Selfcopy)によるフラクタル生成

・copy01 … box 手法を利用して

０の部分はコピーしない

１の部分は元の配列をコピーする

結果はボックスで囲まれた配列となる。

```
copy01=: 3 : 0 (L:0)
:
if. 1=y. do. x. end.
if. 0=y. do. ($x.)$0 end.
)
```

例えば、次のように実行される。

```
   m1 =: 2 2$1 0 1 1
   m1
1 0
1 1
   m2 =: m1 copy01 <"(0) m1
   m2
+---+---+
|1 0|0 0|
|1 1|0 0|
+---+---+
|1 0|1 0|
|1 1|1 1|
+---+---+
```

しかしながら、次に進むためにはこのボックスをはずすことが必要である。この処理がなかなか困難であった。いろいろ試行錯誤の末、次のようにして出来た。（付録を参照）

```
   m3 =: ,./ ,./ >m2
   m3
1 0 0 0
1 1 0 0
1 0 1 0
1 1 1 1
```

これを含めたのが次の selfcopy の定義である。

```
selfcopy =: 3 : 0
r =. y. (copy01 <"(0)) y.
```

```
,./ ,./ >r
)
```

４．シェルピンスキーのフラクタル

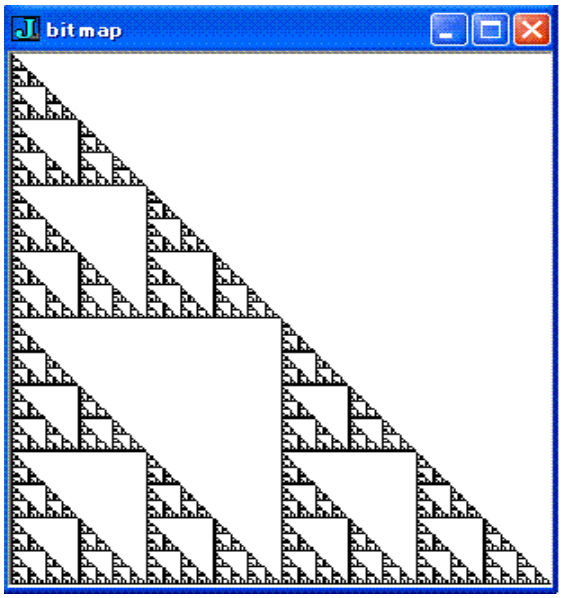
先の値m1を用いて次のように行う。

```
   selfcopy^:2 m1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0
1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0
1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

さて、このようにして得た配列をウィンドウズ・グラフィックスで見てみよう。

```
viewmat -. selfcopy^:3 m1
```

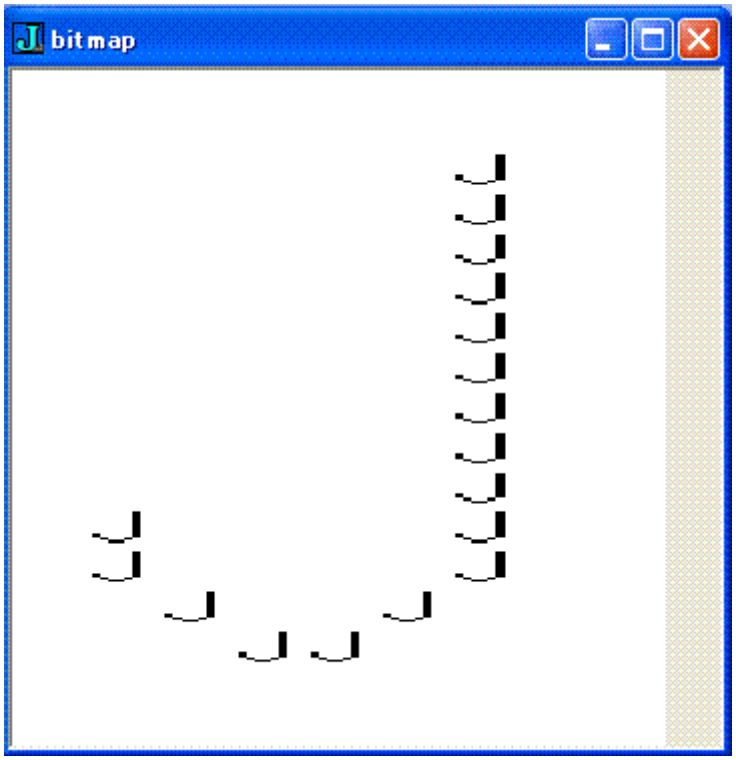
やり方は異なるがReiterの本にあるのと同じシェルピンスキーのフラクタル図である。



５．文字列のフラクタル

５．１　文字Ｊのフラクタル

viewmat -. selfcopy dcp 'user¥J24.bmp'



他の文字列でもやってみよう。

viewmat -. selfcopy dcp 'user¥TN24.bmp'

viewmat -. selfcopy dcp 'user¥Niskaw.bmp'

bitmap
bitmap

付録　どうやって、ボックスをはずすか？

```
   B =: (i.2 3);(6+i.2 3)
   C =: (12+i.2 3);(18+i.2 3)
   B
+-----+-------+
|0 1 2|6  7  8|
|3 4 5|9 10 11|
+-----+-------+
   C
+--------+--------+
|12 13 14|18 19 20|
|15 16 17|21 22 23|
+--------+--------+
   D =: B,. C
   D
+-------+--------+
|0 1 2  |12 13 14|
|3 4 5  |15 16 17|
+-------+--------+
|6  7  8|18 19 20|
|9 10 11|21 22 23|
+-------+--------+
```

このボックスをどうやってはずすか？ … 望むようにはずすＪのプリミティブはない！

⇒ すべて(スカラーまで)はずした後、多ランク(多次元)の配列を連結する。

```
   E =: >D
   E
 0  1  2
 3  4  5

12 13 14
15 16 17



 6  7  8
 9 10 11
```

```
18 19 20
21 22 23
   $E
2 2 2 3
   E0 =: 0{E
   E1 =: 1{E
   E0
 0  1  2
 3  4  5

12 13 14
15 16 17
   E1
 6  7  8
 9 10 11

18 19 20
21 22 23
```

ランク２の連結 Append(,)でまず１ステップできた。

```
   E0 ,"(2) E1
 0  1  2
 3  4  5
 6  7  8
 9 10 11

12 13 14
15 16 17
18 19 20
21 22 23
```

同じ操作は Stitch(,.)（ランクを１下げた Append(,)に等しい）でも行うことができる。

```
E0 ,. E1
 0  1  2
 3  4  5
 6  7  8
```

```
 9 10 11

12 13 14
15 16 17
18 19 20
21 22 23
```
これを最初からまとめて行うには、次のようにする。
```
   ,. / >D
 0  1  2
 3  4  5
 6  7  8
 9 10 11

12 13 14
15 16 17
18 19 20
21 22 23
   F =: ,. / >D
   $F
2 4 3
```
さらに、Stitch(,.)をもう一回行う。
```
   ,. /F
0  1  2 12 13 14
3  4  5 15 16 17
6  7  8 18 19 20
9 10 11 21 22 23
```
以上のようにして、ボックスをはずした配列を得ることが出来た。

付録　プログラムリスト
```
NB. Fractal Visualization for Character String
NB.   programmed by T.Nishikawa, 2007/4/16
require 'system¥packages¥graphics¥bmp.ijs'

NB. readbmp    - read bitmap file
NB. returns RGB data
NB.   revised by T.N. 2001/11/2
```

```
readbmpN=: 3 : 0
try. dat=. fread < y.
catch. 'file read error' return. end.
if. -. 'BM'-:2{.dat do. 'not a bitmap file' return. end.
toi=. 256&#.@(a.&i.)@(|."1)
bits=. toi 28 29 {dat
if. toi 30 31 32 33{dat do.
  'compressed format not supported' return.
end.
'off shdr cls rws'=. toi (10+i.4 4){dat
pal=. 256 #. a. i. _4 }: ¥ (shdr+14) }. off{.dat
dat=. off}.dat
if. bits=4 do.
  dat=. a. {~ ,16 16#:a.i.dat
end.
if. bits e. 4 8 do.
  pal {~ a.i. |. (rws,cls){.(rws,cls+4|-cls)$dat
elseif. bits=24 do.
NB.  |. (rws,cls) $ 256 #. a. i. _3 [ ¥ dat
NB. revised by T.N.
NB. da...simply RGB numbers disply (3 numbers for one pixel)
  da =. |. a. i. (-4|cls) }."(1) (rws, (cls*3) + (4|cls))$dat
NB. RGB boxed display per pixel
  (rws, cls) $ <"(1) |. "(1) _3 [ ¥ , da
elseif. 1 do.
  'only 4,8 and 24-bit bitmaps supported, this is ',(":bits),'-bit'
end.
)

NB. Display Pattern of 24-BMP(0,1) characters
NB. dcp 'J24.bmp'
dcp =: 3 : 0
> ((0, 0, 0)&-:) (L:0) readbmpN y.
)

Jchar =: dcp 'user¥J24.bmp'
```

```
NB. viewmat -. dcp 'J24.bmp'
NB. z2inf dcp 'J24.bmp'
z2inf =: 3 : 0"(1)
_ ((y.=0)#(i.#y.)) } y.
)
bmp_lib =: 3 : 0
require 'system¥main¥dir.ijs'
dir 'user¥*.bmp'
)

copy01=: 3 : 0 (L:0)
:
if. 1=y. do. x. end.
if. 0=y. do. ($x.)$0 end.
)

m1 =: 2 2$1 0 1 1
NB. m2 =: m1 copy01 <"(0) m1
NB. m3 =: ,./ ,./ >m2

M1 =: 3 2$1 0 1 0 1 1
NB. M2 =: M1 copy01 <"(0) M1
NB. M3 =: ,./ ,./ >M2

selfcopy =: 3 : 0
r =. y. (copy01 <"(0)) y.
,./ ,./ >r
)
NB. Usage examples
NB. viewmat -. selfcopy^:2 m1
NB. viewmat -. selfcopy^:3 m1 for Sierpinski Fractal
NB. viewmat -. selfcopy Jchar
```