Grapesでフラクタル

Grapes は非常に優れたソフトである．簡単なスクリプト言語をもっており，ローカル変数を使える．よって再帰呼び出しを用いれば 20 行足らず（ただしヒルベルト曲線を除く）で基本的なフラクタル図形を描くことができる．しかも出力はベクトル形式のデータに対応しているため滑らかな図の印刷が可能である．またフラクタルの描画の様子は見ていて飽きない．

以下のレポートはフラクタルを体系的に述べたものではない．また Grapes がフラクタルを作図するために適しているというわけでもない．付属品としてついているスクリプト言語でどの程度までできるのか興味本位で試したものに過ぎない．

1　木

まず最も簡単な，二分木構造である．

```
//main
X:=(0,0) //基準の座標
z:=0 //ネストの深さ
k:=0.5
Call(fractal,X,z)
end

//fractal
if z<10 then
P := X
Q := X+k^z*(cos(Pi/3),sin(Pi/3))
R := X +k^z*(-cos(Pi/3),sin(Pi/3))
overdraw
X := R
call(fractal,Q,z+1)
call(fractal,X,z+1)
endif
end
```

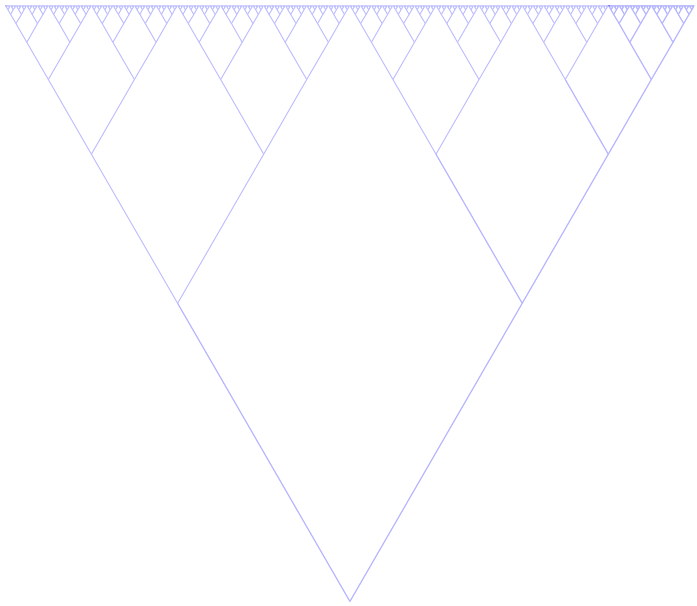

図 1

Grapes のスクリプトで注意しなければならないことの一つに図形の描き方がある．PQ という線分を描きたければ，あらかじめ Grapes 本体で線分は引いておき，それをプログラム上で”draw”あるいは”overdraw”という形で描くのである．その描いた線分を残しておきたい場合は本体の方で残像を指定しておくのである．これは普通のプログラムの図形の描き方とは大きく違うのでとまどう．任意の図形をその場で描くというイメージではない．また，余分な線や点がかきこまれることも多々ある．

もうひとつ注意しなければならないのはローカル変数が 4 個しかないことである．x, y, z, w 以外は全てグローバル変数である．うっかりローカルのつもりで書くと失敗する．

もうすこし木らしく
したのが図 2 である.
木というよりカリフラワーに近い.

```
//main
X:=(0,0) //基準の座標
z:=0 //ネストの深さ
k:=0.553
s:=Pi/6
Call(fractal,X,z,0)
end

//fractal
if z<8 then
P := X
Q := X+k^z*(sin(w+s),cos(w+s))
R := X +k^z*(sin(w-s),cos(w-s))
overdraw
X := R
call(fractal,Q,z+1,w+s)
call(fractal,X,z+1,w-s)
endif
end
```

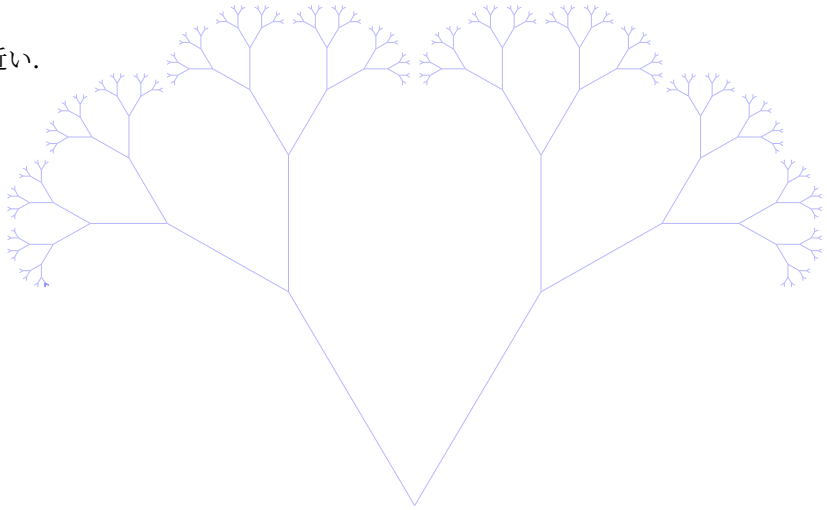

図 2

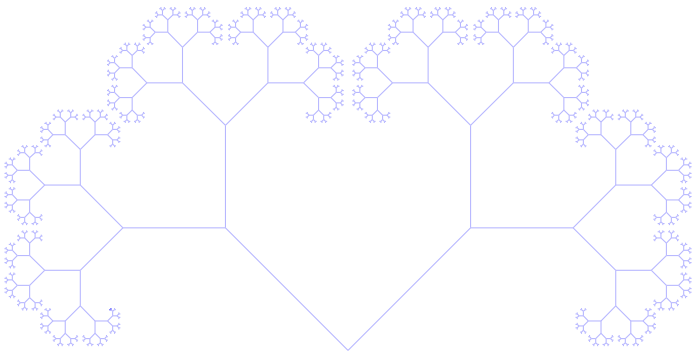

図 3

ほとんど，プログラムの構造は変わっていない．図 1 では単純に左右の枝を $\pm 30°$ 横に振っただけであったが，それを元の枝の角度に応じて枝分かれするようにしただけである．すこし枝の間に余裕ができるので，枝の長さの比を 0.553 まで増やしてみた．枝を 3 本にして，中心の枝は長く左右は短くすると杉の木のようにできるが，残念ながらローカル変数が足りないのでうまくいかない．再帰にこだわらなければいいのだが，今回は再帰呼び出しのみでやろうと思う．

枝の角度をもう少し広げると枝がぶつからなくなるので，k を 0.59 まで増やすことができる．それでは，この角度のままで $k = \frac{1}{\sqrt{2}}$ まで増やすとどうなるかというと，当然枝同士がぶつかることになるのだが，この枝の先端を線分で結ぶと次の C 曲線となる．

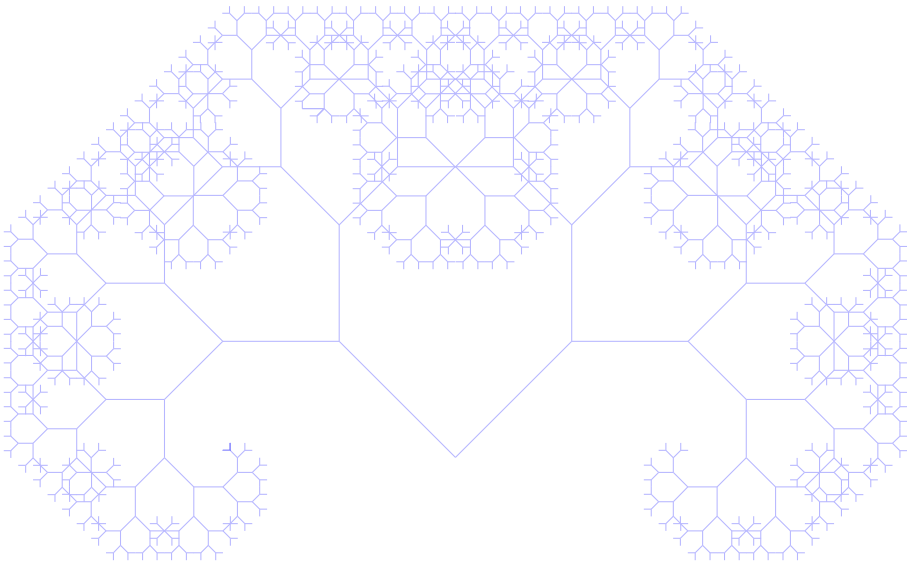

図 4

2　C 曲線

C 曲線をかくには，ローカル変数の不足を感じる，せめてもう一つあればと思うところである．ネストの深さを測る変数がローカル変数でとれないので，安直ではあるが，線分の長さがある程度短くなったらサブルーチンを抜けるようにしてある．アルゴリズムとしては難しくないが，ローカル変数のやり繰りがやや困難である．

```
//main
X:=(-1,0) //基準の座標
Y:=(1,0)
Call(fractal,X,Y)
end

//fractal
if (x-z)^2+(y-w)^2<0.002 then
P := X
Q :=Y
overdraw
```

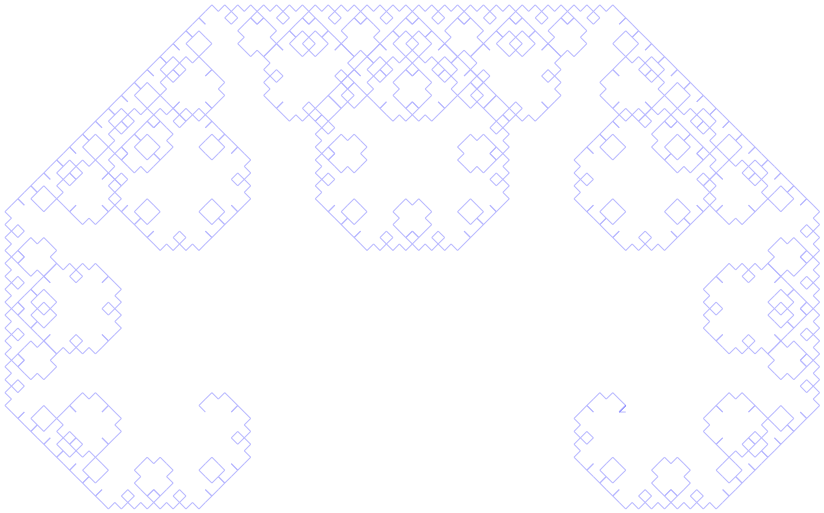

図 5

```
else
Call(fractal,X,X+((z-x)-(w-y),(z-x)+(w-y))/2)
Call(fractal,X+((z-x)-(w-y),(z-x)+(w-y))/2,Y)
endif
end
```

3　ドラゴンカーブ

C 曲線とドラゴンカーブはアルゴリズムはほとんど同じである．たった一箇所，座標をひっくり返しただけである．C 曲線でも同様であるが，これ以上細かくすると，3000 ステップをこえてしまうので，残像が前のほうから消えていってしまう．これは Grapes の仕様なので仕様がない（日本語が変？）．

```
//main
X:=(-1,0) //基準の座標
Y:=(1,0)
Call(fractal,X,Y)
end

//fractal
if (x-z)^2+(y-w)^2<0.002 then
P := X
Q := Y
overdraw
else
```

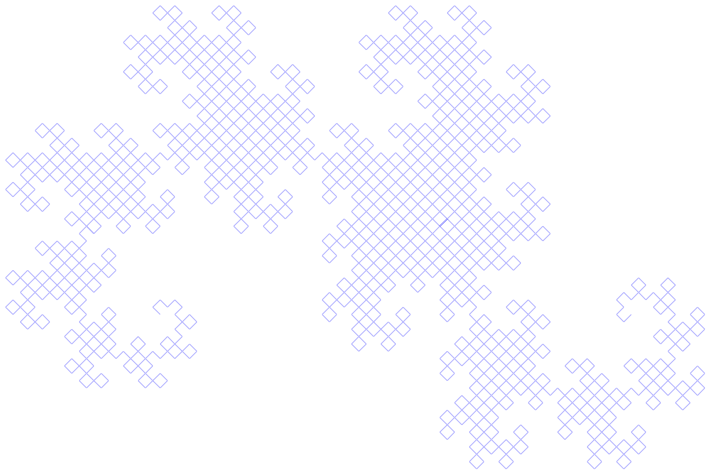

図 6

```
Call(fractal,X,X+((z-x)-(w-y),(z-x)+(w-y))/2)
Call(fractal,Y,X+((z-x)-(w-y),(z-x)+(w-y))/2)//ここが入れ替わっただけである．
endif
end
```

このように C 曲線のプログラムを流用すると簡単だが，切れ切れに描いてしまうので，一筆書きで描いていくのを見る楽しみはない．もし一筆にしたかったら少し改変する必要がある．

4　コッホ曲線

コッホ曲線は若干複雑であるが，考え方に大きな変化は無い．

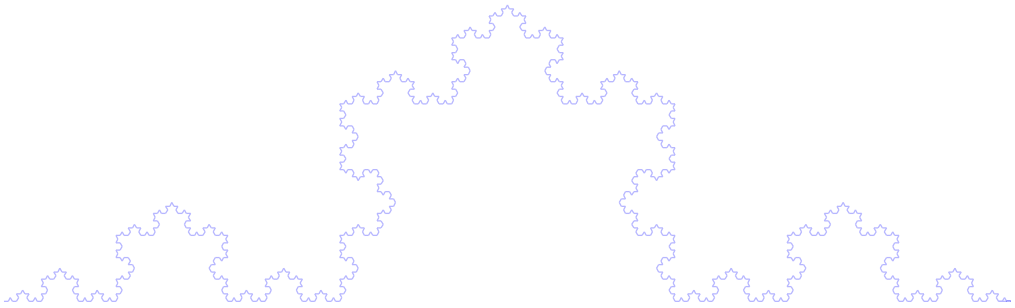

図 7

```
//main
X:=(-1,0) //基準の座標
Y:=(1,0)
Call(fractal,X,Y)
end

//fractal
if (x-z)^2+(y-w)^2<0.0001 then
P := X
Q := Y
overdraw
else
Call(fractal,X,X+(Y-X)/3)
Call(fractal,X+(Y-X)/3,X+(1/(2*Sqrt(3)))*(Sqrt(3)*(z-x)-(w-y),(z-x)+Sqrt(3)*(w-y)))
Call(fractal,X+(1/(2*Sqrt(3)))*(Sqrt(3)*(z-x)-(w-y),(z-x)+Sqrt(3)*(w-y)),X+2*(Y-X)/3)
Call(fractal,X+2*(Y-X)/3,Y)
endif
end
```

5　高木関数

高木関数はプログラムにするとかえってわかりにくくなるが，理屈はかんたんである．

```
//main
s:=(.5)^9
for p:=-.25  to 2.25 step s
q:=0
Call(takagi,p,1)
draw
Q:=P//点でなく線分にするため
next
end

//takagi
if y<s then
end
else
q:=q+abs(x-y)
Call(takagi,abs(x-y),y/2)
```

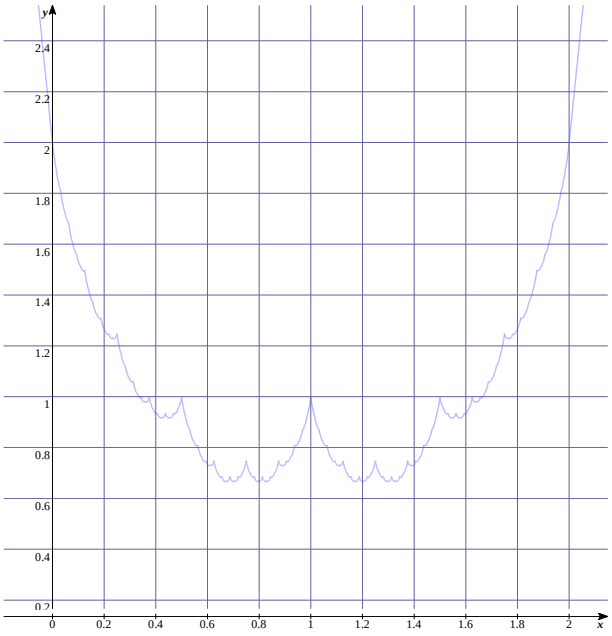

図 8

```
end
```

6　シェルピンスキーのカーペットとガスケット

```
//main
X:=(0,0) //基準の座標
z:=0
Call(carpet,X,z+1)
end

//carpet
if z>3 then
P:= X+((1/3)^z,(1/3)^z)
Q:= X-((1/3)^z,(1/3)^z)
overdraw
else
Call(carpet,X+((1/3)^z,(1/3)^z),z+1)
Call(carpet,X-((1/3)^z,(1/3)^z),z+1)
Call(carpet,X+(-(1/3)^z,(1/3)^z),z+1)
Call(carpet,X-(-(1/3)^z,(1/3)^z),z+1)
Call(carpet,X+(0,(1/3)^z),z+1)
Call(carpet,X-(0,(1/3)^z),z+1)
Call(carpet,X+(-(1/3)^z,0),z+1)
Call(carpet,X-(-(1/3)^z,0),z+1)
endif
end
```

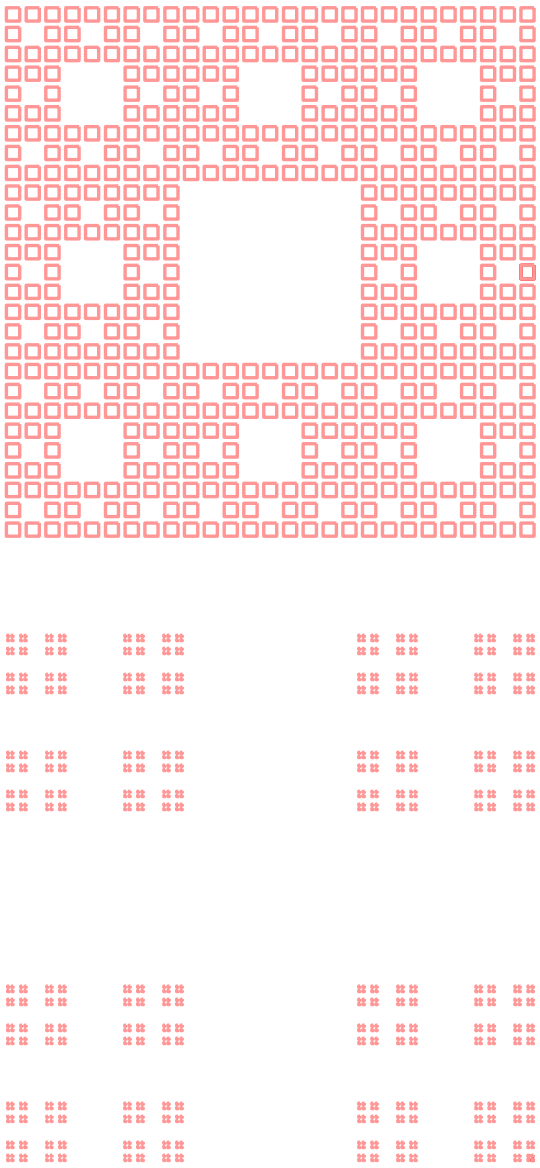

図 9

いくら再帰呼び出しといってもさすがに同じような文を 8 つも並べては格好が悪い．ここは For〜Next にすべきであろう．これ以上細かくすると 3000 ステップを超えて一部が残像として残らない．8 つのうち後ろの 4 行を除くと，カントールの塵という図形になる（図 9 下）

シェルピンスキーのカーペットというのは四角であるが，三角形にしたのがガスケットと呼ばれる図形である．考え方は同じである．

```
//main
X:=(0,0) //基準の座標
z:=0
Call(carpet,X,z+1)
end

//carpet
if z>7 then
P:=X+2*(0,(1/2)^z)
Q:=X-2*((1/2)^z*sqrt(3)/2,(1/2)^z/2)
R:=X+2*((1/2)^z*sqrt(3)/2,-(1/2)^z/2)
overdraw
else
Call(carpet,X+(0,(1/2)^z),z+1)
Call(carpet,X-((1/2)^z*sqrt(3)/2,(1/2)^z/2),z+1)
Call(carpet,X+((1/2)^z*sqrt(3)/2,-(1/2)^z/2),z+1)
endif
end
```

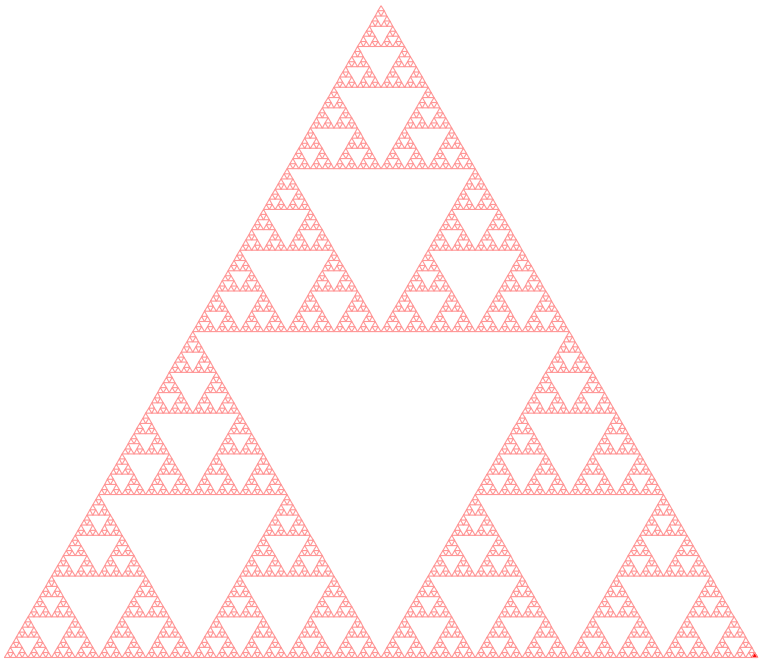

図 10

7　マンデルブロー集合

再帰呼び出しを無理矢理使ったのでわかりにくくなってしまった．ネストの深さに制限があるためこれ以上正確なものが描けない．よって再帰を使わないほうが妥当であろう．余談ではあるが，私はマンデルブロー集合を見るとカブトガニを連想する．

```
 //main
 for a:=-2 to 0.6 step 0.02
 for b:=0 to 1.5 step 0.02
Call(Mandelbrot,a,b,0)
 next b
 next a
end

//Mandelbrot
if z>18 then
overdraw
end
else
if x*x+y*y>4 then end
endif
Call(Mandelbrot,x^2-y^2+a,2*x*y+b,z+1)
endif
end
```

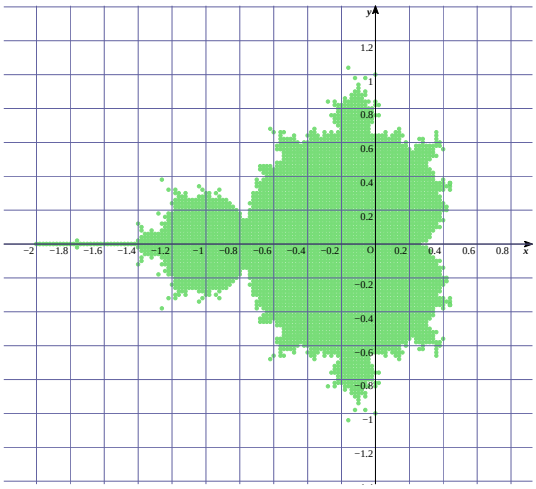

図 11

2 次のマンデルブロー集合がカブトガニなら，3 次のそれはさしずめ亀だろう．

```
//main
 for a:=-2 to 0.6 step 0.02
 for b:=0 to 1.5 step 0.02
Call(Mandelbrot,a,b,0)
 next b
 next a
end

//Mandelbrot
if z>18 then
overdraw
end
else
if x*x+y*y>4 then end
endif
Call(Mandelbrot,x*(x^2-3*y^2)+a,y*(3*x^2-y^2)+b,z+1)
endif
end
```

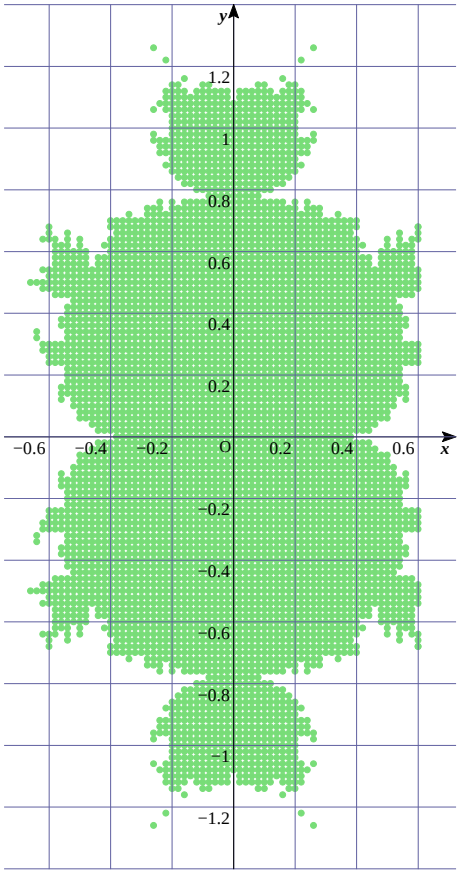

図 12

8　ヒルベルト曲線

ヒルベルト曲線は最も難しい．それは再帰関数が 4 つ必要で（うまくすると一つにまとめることができるとは思うのだが），それをそれぞれが呼び合う形となるからである．したがってスクリプトは長くなり十数行というわけにはいかない．

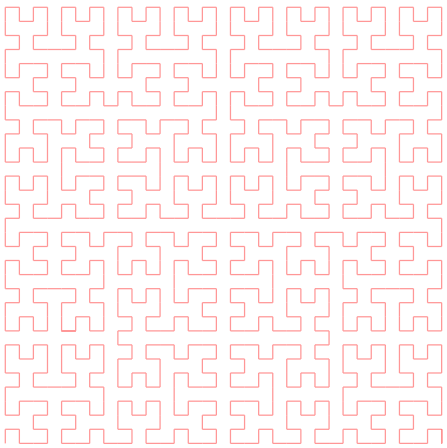

図 13

```
//main
P:=(0,0)
Q:=(0,0)
n:=5
k:=0.5^n
Call(Urd,n)
end

//Ldr
if x>0 then
Call(Dlu,x-1)
Q:=P
P:=P-(k,0)
overdraw
Call(Ldr,x-1)
Q:=P
P:=P-(0,k)
overdraw
Call(Ldr,x-1)
Q:=P
P:=P+(k,0)
overdraw
Call(Urd,x-1)
endif
end

//Urd
if x>0 then
Call(Rul,x-1)
Q:=P
P:=P+(0,k)
overdraw
Call(Urd,x-1)
Q:=P
P:=P+(k,0)
overdraw
Call(Urd,x-1)
Q:=P
P:=P-(0,k)
overdraw
Call(Ldr,x-1)
endif
end

//Rul
if x>0 then
Call(Urd,x-1)
Q:=P
P:=P+(k,0)
overdraw
Call(Rul,x-1)
Q:=P
P:=P+(0,k)
overdraw
Call(Rul,x-1)
Q:=P
P:=P-(k,0)
overdraw
Call(Dlu,x-1)
endif
end

//Dlu
if x>0 then
Call(Ldr,x-1)
Q:=P
P:=P-(0,k)
overdraw
Call(Dlu,x-1)
Q:=P
P:=P-(k,0)
overdraw
Call(Dlu,x-1)
Q:=P
P:=P+(0,k)
overdraw
Call(Rul,x-1)
endif
end
```

若干，ヒルベルト曲線の関数について述べておこう．通常 2 点間をコの字形で移動するにはふた通りあることは自明であるが，一つのヒルベルト曲線ではこのうち一通りしか許していない．例えば前頁の例では下に移動するときは  ，上に移動するときは  ，右に移動するときは  ，左は  という具合である．上下，左右の動きは鏡像になっている．この 4 種類の動きがそれぞれの関数になっているわけである．前述のプログラムは overdraw がたびたび出てきて冗長感がぬぐえない．改良しようとしたが失敗した．

9　ふたたびシェルピンスキーのガスケット

シェルピンスキーのガスケットをヒルベルト曲線風に描いてみよう．

今度は先ほどの冗長プログラムと違って，うまく書けたと思う．

```
//main
P:=(0,0)
Q:=(0,0)
a:=0.5^n
Call(Gascket,n,0)
end

hidescript//Gascket
if x=0 then
Q:=P
P:=P+a*roll(y*Pi/3)
draw
else
Call(Gascket,x-1,y+1-(y mod 2)*2)
Call(Gascket,x-1,y)
Call(Gascket,x-1,y-1+(y mod 2)*2)
endif
end
```

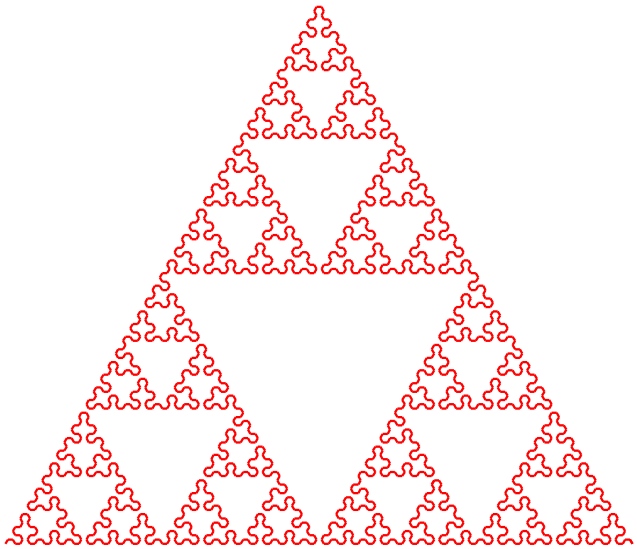

図 14