Von Koch, Mandelbrot et fractales…

Dès que l'on observe la nature (côtes rocheuses, surface d'un morceau d'ambre, cristallisations, colloïdes, turbulences atmosphériques, groupements d'étoiles, structure des bronches …) on s'aperçoit que les « braves fonctions » (dérivables jusqu'à un ordre suffisant pour ne pas donner de cauchemars à un lycéen) ou que les « braves objets géométriques » (cercles, droites…) ne sont pas des modèles satisfaisants pour décrire l'aspect « chaotique » de ces observations.

La baie de Derby en Australie

Benoit Mandelbrot fournit avec ses courbes fractales un modèle intéressant. Ces objets sont « auto-similaires ».

L'exemple d'une côte rocheuse :

On peut trouver dans le « Petit Robert » : la côte des la France mesure 3200km.

Cette affirmation est fort douteuse. En choisissant une méthode de mesure :

- Se promener le long de la côte avec un compas d'ouverture **ε** et compter le nombre de pas
- Recouvrir la côte avec un ruban de largeur **ε** et diviser la surface de ce ruban par **ε**
- Recouvrir la côte avec des disques de rayon **ε**

Il se trouve que la longueur L(**ε**) trouvée croit indéfiniment lorsque **ε** diminue.

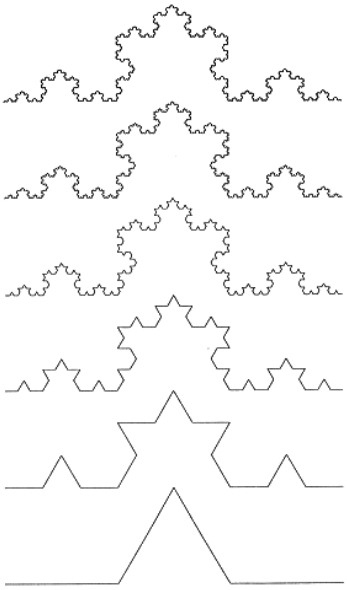
On comprend bien que, selon **ε** , on va enjamber les montagnes, contourner les rochers, les galets , les grains de sable, les atomes …). Mais il ne faut pas se servir d'une carte pour réaliser cette expérience, la carte étant déjà un modèle « trop sympathique » d'un point de vue différentiel.

Un modèle : la courbe de Niels Von Koch (1870-1924)

A partir d'un premier segment de droite, on modifie récursivement chaque segment:

1. on divise le segment de droite en trois segments de longueurs égales,

2. on construit un triangle équilatéral ayant pour base le segment médian de la première étape,
3. on supprime le segment de droite qui était la base du triangle de la deuxième étape.



La courbe de Von Koch est la limite des courbes obtenues, lorsqu'on répète indéfiniment les étapes précédentes. Elle a une longueur infinie parce qu'à chaque fois qu'on applique les modifications ci-dessus sur chaque segment de droite, la longueur totale est multipliée par 4/3.

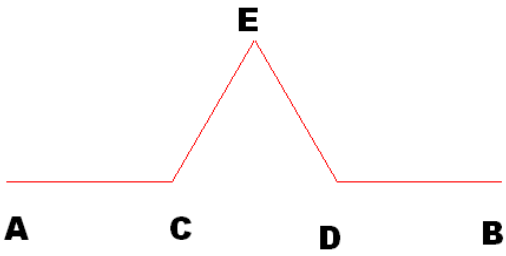
La surface délimitée par la courbe est cependant finie. Si on a choisi l'unité d'aire de telle sorte que le triangle construit à la première itération soit d'aire 1, alors l'aire de chacun des quatre triangles construits lors de la seconde itération est 1/9 : on a donc augmenté l'aire totale de 4/9. La surface totale est : $\sum_{n=0}^{\infty}(\frac{4}{9})^n = \frac{1}{1-\frac{4}{9}} = \frac{9}{5}$.

Elle a été introduite comme exemple de "courbe continue sans tangente, obtenue par une construction géométrique élémentaire", quoiqu'en pensait Hermite (dans une correspondance avec Stieljes « je me détourne avec effroi et horreur de cette plaie lamentable des fonctions continues sans dérivée »)…

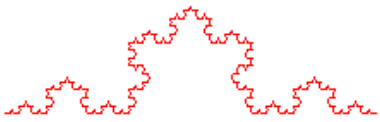
On peut aussi remarquer que la trace sur le segment initial est l'ensemble de Cantor.

<u>Dessin avec ordinateur :</u>

La définition de la courbe de Von Koch amène naturellement à une programmation récursive. A partir de deux points de départ A et B, on calcule les points intermédiaires $C = \frac{2A+B}{3}$, $D = \frac{A+2B}{3}$ et $E = C + (D - C)\frac{1+i\sqrt{3}}{2}$. Puis on relance la procédure au niveau inférieur avec les quatre segments. Au niveau le plus bas, on ne touche pas au segment. Le code est extrêmement compact.



```
> VK:=proc(n,A,B)
  local C,D,E;
  if n=0 then RETURN (A)
  else
    C:=evalc((2*A+B)/3);
    D:=(A+2*B)/3;
    E:=C+(D-C)*(1+I*rac3)/2;
    VK (n-1,A,C),VK(n-1,C,E),VK(n-1,E,D),VK(n-1,D,B);
  fi
  end:
> rac3:=evalf(sqrt(3)):
  with(plots): complexplot([VK(4,0,1),1],axes=none,scaling=constrained);
```



Ou bien,

```
> VK:=proc(n,A,B)
  local C,D,E;
  if n=0 then RETURN (A)
  else
    C:=evalc((2*A+B)/3);
    D:=(A+2*B)/3;
    E:=C+(D-C)*(1+I*rac3)/2;
    VK (n-1,A,C),VK(n-1,C,E),VK(n-1,E,D),VK(n-1,D,B);
  fi
  end:
> rac3:=evalf(sqrt(3)):
  with(plots): complexplot([VK(4,0,1),1],axes=none,scaling=constrained);
```

Avec un langage récursif et une tortue, on peut utiliser la chaine des déplacements

« A G(-60°) A D(+120°) A ».

```
def koch(T,l,n):
    if n<=0:
        T.avance(l)
    else:
        koch(T,l/3,n-1)
        T.tourne_gauche(60)
        koch(T,l/3,n-1)
        T.tourne_droite(120)
        koch(T,l/3,n-1)
        T.tourne_gauche(60)
        koch(T,l/3,n-1)
```

Avec PYTHON (après implémentation d'une tortue):

```
#! /usr/bin/python
# -*- coding:utf-8 -*-
from Tkinter import *
from math import radians,cos,sin,sqrt
class tortue:
    def
__init__(self,canvas,x=0,y=0,angle=0,crayon=1,coul
eur='black',epaisseur=1):
        self.canvas=canvas
        self.crayon=crayon
        self.x=x
        self.y=y
        self.angle=angle
        self.couleur=couleur
        self.epaisseur=epaisseur
    def __repr__(self):
        return "canvas : %s\ncrayon :%d\nx      :
%d\ny     : %d\nangle  : %d"
%(self.canvas,self.crayon,self.x,self.y,self.angle)
    def avance(self,l):
        X,Y=self.x+l*cos(radians(self.angle)),self.y-
l*sin(radians(self.angle))
        if self.crayon:

self.canvas.create_line(self.x,self.y,X,Y,fill=self.co
uleur,width=self.epaisseur)
        self.x,self.y=X,Y
    def tourne_gauche(self,theta):
        self.angle+=theta
    def tourne_droite(self,theta):
        self.tourne_gauche(-theta)
    def pose_crayon(self):
        self.crayon=1
    def leve_crayon(self):
        self.crayon=0

def koch(T,l,n):
    if n<=0:
        T.avance(l)
    else:
        koch(T,l/3,n-1)
        T.tourne_gauche(60)
        koch(T,l/3,n-1)
        T.tourne_droite(120)
        koch(T,l/3,n-1)
        T.tourne_gauche(60)
        koch(T,l/3,n-1)

if __name__=='__main__':
    root=Tk()

can=Canvas(root,height=400,width=1000,bg='white
')
    can.pack()
    T=tortue(can)
    T.y=150
    koch(T,300,5)
    root.mainloop()
```

Sans langage récursif, il faut gérer soit même la liste des points.

```
> VK(0,[0,0],[1,0]);
> VK(1,[0,0],[1,0]);
```

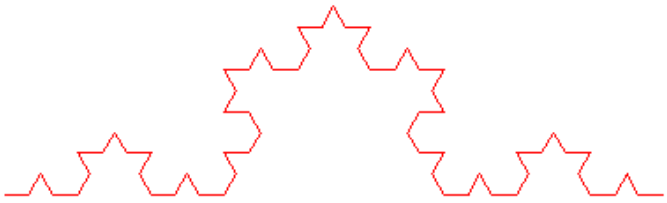
$$[0,0],[1,0]$$

$$[0,0],\left[\frac{1}{3},0\right],\left[\frac{1}{3},0\right],\left[\frac{1}{2},.2886751347\right],\left[\frac{1}{2},.2886751347\right],\left[\frac{2}{3},0\right],\left[\frac{2}{3},0\right],[1,0]$$

```
> VK(2,[0,0],[1,0]);
```

$$[0,0],\left[\frac{1}{9},0\right],\left[\frac{1}{9},0\right],\left[\frac{1}{6},.09622504490\right],\left[\frac{1}{6},.09622504490\right],\left[\frac{2}{9},0\right],\left[\frac{2}{9},0\right],\left[\frac{1}{3},0\right],\left[\frac{1}{3},0\right],\left[\frac{7}{18},.09622504489\right],\left[\frac{7}{18},.09622504489\right],[.3333333333,.1924500899],$$
$$[.3333333333,.1924500899],\left[\frac{4}{9},.1924500898\right],\left[\frac{4}{9},.1924500898\right],\left[\frac{1}{2},.2886751347\right],\left[\frac{1}{2},.2886751347\right],\left[\frac{5}{9},.1924500898\right],\left[\frac{5}{9},.1924500898\right],$$
$$[.6666666667,.1924500899],[.6666666667,.1924500899],\left[\frac{11}{18},.09622504489\right],\left[\frac{11}{18},.09622504489\right],\left[\frac{2}{3},0\right],\left[\frac{2}{3},0\right],\left[\frac{7}{9},0\right],\left[\frac{7}{9},0\right],\left[\frac{5}{6},.09622504490\right],$$
$$\left[\frac{5}{6},.09622504490\right],\left[\frac{8}{9},0\right],\left[\frac{8}{9},0\right],[1,0]$$

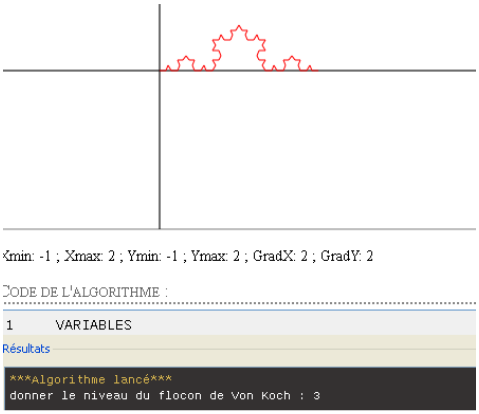
```
VK_nonrecursif:=proc(n,AA,BB)
local L,M,i,    m,k,xdeb, ydeb, xfin, yfin, A,B,C,D,E;
L:=[AA,BB];
for i from 1 to n do
k:=nops(L); M:=[];
for m from 1 to k-1 do
xdeb:=L[m][1]; ydeb:=L[m][2];
xfin:=L[m+1][1]; yfin:=L[m+1][2];
A:=L[m];
C:=[(2*xdeb+xfin)/3,(2*ydeb+yfin)/3];
E:=[(3*xdeb+3*xfin-rac3*(yfin-ydeb))/6,(3*ydeb+3*yfin+rac3*(xfin-xdeb))/6];
D:=[(xdeb+2*xfin)/3,(ydeb+2*yfin)/3];
M:=[op(M),A,C,E,D];
od;
M:=[op(M),[1,0]];
L:=M;
od;
RETURN (L)
end:
> rac3:=evalf(sqrt(3)):
plot([VK_nonrecursif(3,[0,0],[1,0])],axes=none,scaling=constrained);
```



Avec Algobox, c'est assez fastidieux, les fonctions étant limitées sur les listes (il faut gérer la longueur des listes et la recopie).

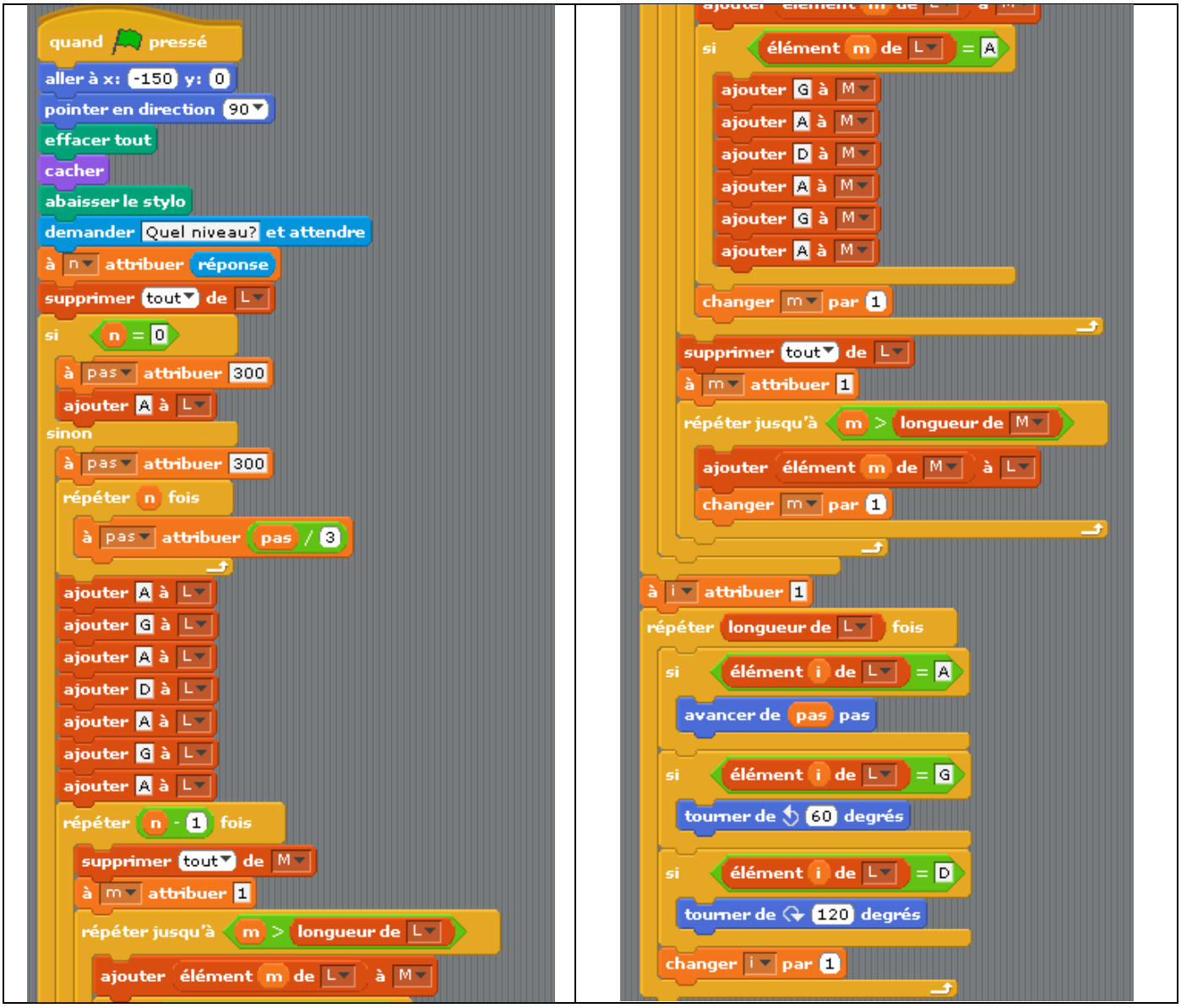
```
1   VARIABLES
2     n EST_DU_TYPE NOMBRE
3     L_abs EST_DU_TYPE LISTE
4     L_ord EST_DU_TYPE LISTE
5     i EST_DU_TYPE NOMBRE
6     k EST_DU_TYPE NOMBRE
7     longueur_L EST_DU_TYPE NOMBRE
8     M_abs EST_DU_TYPE LISTE
9     M_ord EST_DU_TYPE LISTE
10    j EST_DU_TYPE NOMBRE
11    m EST_DU_TYPE NOMBRE
12    x_deb EST_DU_TYPE NOMBRE
13    y_deb EST_DU_TYPE NOMBRE
14    x_fin EST_DU_TYPE NOMBRE
15    y_fin EST_DU_TYPE NOMBRE
16    longueur_M EST_DU_TYPE NOMBRE
17    x_suiv EST_DU_TYPE NOMBRE
18    y_suiv EST_DU_TYPE NOMBRE
19  DEBUT_ALGORITHME
20    AFFICHER "donner le niveau du flocon de
Von Koch : "
21    LIRE n
22    AFFICHER n
23    L_abs[1] PREND_LA_VALEUR 0
24    L_ord[1] PREND_LA_VALEUR 0
25    L_abs[2] PREND_LA_VALEUR 1
26    L_ord[2] PREND_LA_VALEUR 0
27    longueur_L PREND_LA_VALEUR 2
28    POUR i ALLANT_DE 1 A n
29      DEBUT_POUR
30      longueur_M PREND_LA_VALEUR 0
31      POUR m ALLANT_DE 1 A longueur_L - 1
32        DEBUT_POUR
33        x_deb PREND_LA_VALEUR L_abs[m]
```

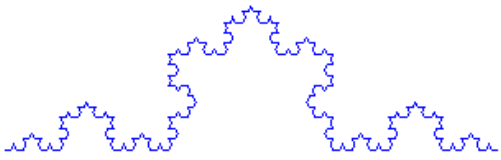
```
34        y_deb PREND_LA_VALEUR L_ord[m]
35        x_fin PREND_LA_VALEUR L_abs[m+1]
36        y_fin PREND_LA_VALEUR L_ord[m+1]
37        M_abs[longueur_M +1]
PREND_LA_VALEUR L_abs[m]
38        M_ord[longueur_M +1]
PREND_LA_VALEUR L_ord[m]
39        M_abs[longueur_M+2]
PREND_LA_VALEUR (2*x_deb+x_fin)/3
40        M_ord[longueur_M +2]
PREND_LA_VALEUR (2*y_deb+y_fin)/3
41        M_abs[longueur_M+3]
PREND_LA_VALEUR (3*x_deb+3*x_fin-
sqrt(3)*(y_fin-y_deb))/6
42        M_ord[longueur_M+3]
PREND_LA_VALEUR
(3*y_deb+3*y_fin+sqrt(3)*(x_fin-x_deb))/6
43        M_abs[longueur_M+4]
PREND_LA_VALEUR (x_deb+2*x_fin)/3
44        M_ord[longueur_M+4]
PREND_LA_VALEUR (y_deb+2*y_fin)/3
45        longueur_M PREND_LA_VALEUR
longueur_M+4
46        FIN_POUR
47        longueur_M PREND_LA_VALEUR
longueur_M+1
48        M_abs[longueur_M] PREND_LA_VALEUR 1
49        M_ord[longueur_M] PREND_LA_VALEUR
0
50        POUR m ALLANT_DE 1 A longueur_M
51         DEBUT_POUR
52         L_abs[m] PREND_LA_VALEUR M_abs[m]
53         L_ord[m] PREND_LA_VALEUR M_ord[m]
54         FIN_POUR
55        longueur_L PREND_LA_VALEUR
longueur_M
56        FIN_POUR
57       POUR m ALLANT_DE 1 A longueur_L-1
58        DEBUT_POUR
59        x_deb PREND_LA_VALEUR L_abs[m]
60        y_deb PREND_LA_VALEUR L_ord[m]
61       x_suiv PREND_LA_VALEUR L_abs[m+1]
62        y_suiv PREND_LA_VALEUR L_ord[m+1]
63        TRACER_SEGMENT (x_deb,y_deb)-
>(x_suiv,y_suiv)
64        FIN_POUR
65    FIN_ALGORITHME
```



Avec Scratch, on a bien la tortue, mais pas la récursivité.
La gestion des listes est cependant prévue.

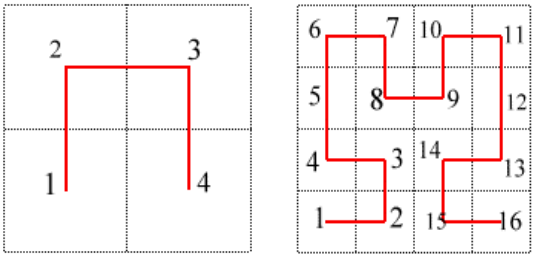
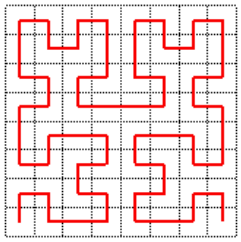
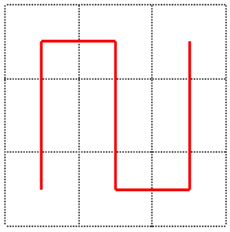
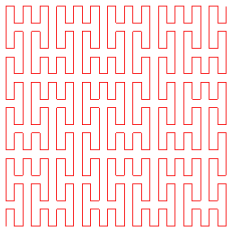
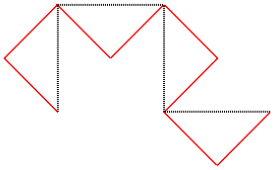
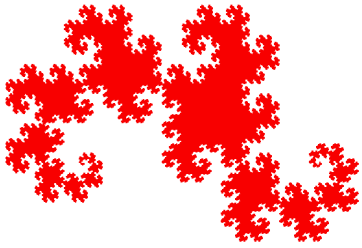
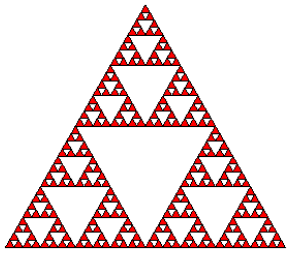
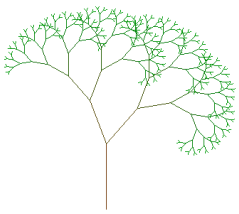
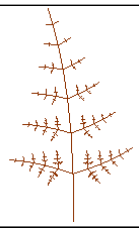
L :=[A] niveau 0
L :=[AGADA] niveau 1
L :=[AGADA G AGADA D AGADA] niveau 2
L :=[AGADA G AGADA D AGADA G AGADA G AGADA D AGADA D AGADA G AGADA D AGADA] niveau 3


quand pressé
aller à x: -150 y: 0
pointer en direction 90
effacer tout
cacher
abaisser le stylo
demander Quel niveau? et attendre
à n attribuer réponse
supprimer tout de L
si n = 0
à pas attribuer 300
ajouter A à L
sinon
à pas attribuer 300
répéter n fois
à pas attribuer pas / 3
ajouter A à L
ajouter G à L
ajouter A à L
ajouter D à L
ajouter A à L
ajouter G à L
ajouter A à L
répéter n - 1 fois
supprimer tout de M
à m attribuer 1
répéter jusqu'à m > longueur de L
ajouter élément m de L à M
si élément m de L = A
ajouter G à M
ajouter A à M
ajouter D à M
ajouter A à M
ajouter G à M
ajouter A à M
changer m par 1
supprimer tout de L
à m attribuer 1
répéter jusqu'à m > longueur de M
ajouter élément m de M à L
changer m par 1
à i attribuer 1
répéter longueur de L fois
si élément i de L = A
avancer de pas pas
si élément i de L = G
tourner de 60 degrés
si élément i de L = D
tourner de 120 degrés
changer i par 1



Autres programmations récursives et fractales à l'ordinateur :

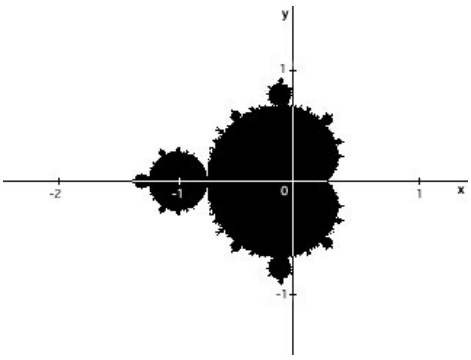
L'ensemble de Cantor	`> cantor:=proc(A,B,n)`
	`if n=0 then [A,B]`
	`else`
	`cantor(A, (2*A+B)/3, n-1),cantor((A+2*B)/3, B, n-1)`
	`fi`
	`end:`
	
	`> plot([cantor([0,0],[1,0],3)],axes=none,color=black);`
		– – – – – – – –

La courbe de Hilbert		
La courbe de Peano		
Le dragon	$S_1 = D$ $S_2 = DDG$ $S_3 = DDG\ DDG\ G$ 	
Le triangle de Sierpinski		
Arbres, en incluant une petite graine aléatoire		

Les mathématiciens ont donné des dimensions non entières aux fractales (ln(4)/ln(2)=1,26.. pour Von Koch ; ln(2)/ln(3)=0,63.. pour Cantor,..).

Sur une courbe « normale » on se repère avec un réel, dans un carré avec deux. Mais sur la courbe de Von Koch, ça ne marche pas. De plus les courbes de Hilbert et de Peano remplissent le carré…

L'ensemble de Mandelbrot :



Pour tout complexe C , on s'intéresse aux itérés successifs par la récurrence

$$\begin{cases} z_{n+1} = {z_n}^2 + C \\ z_0 = 0 \end{cases}$$

L'ensemble de Mandelbrot est l'ensemble des complexes pour lesquels la suite des itérés reste bornée. Cet ensemble est connexe. Sa frontière a un aspect fractal.

Pour reproduire l'ensemble de Mandelbrot, on associe à C des valeurs du plan complexe. On considère généralement la portion du plan complexe ayant comme partie réelle, les valeurs entre -2.5 et 1.5 et comme partie imaginaire, les valeurs entre -1.5 et 1.5. Pour chaque valeur de C, on obtient une suite dont les modules peuvent converger ou diverger.

En pratique, on considère que la suite des modules converge si les 30 premiers modules sont inférieurs à 2. Lorsque la suite des modules converge, on colorie en noir le point de la grille. Après avoir considéré tous les points de la grille, on obtient un ensemble de points noircis : l'ensemble de Mandelbrot.

En couleur : on attribue au point la couleur qui correspond au nombre d'étapes.

```
> mandelbrot:=proc(a,b)
  local x,y,xi,yi,n;
  x:=a;
  y:=b;
  for n from 0 to 30 while evalf(x^2+y^2) < 4 do
  xi:=evalf(x^2-y^2+a);
  yi:=evalf(2*x*y+b);
  x:=xi;
  y:=yi;
  od;
  RETURN(n);
  end:

  plot3d(0,(-2.5)..(1),(-1.5)..(1.5),orientation=[-90,0],style=patchnogrid,
  scaling=constrained,axes=none,numpoints=10000,color=mandelbrot,axes=framed);
```