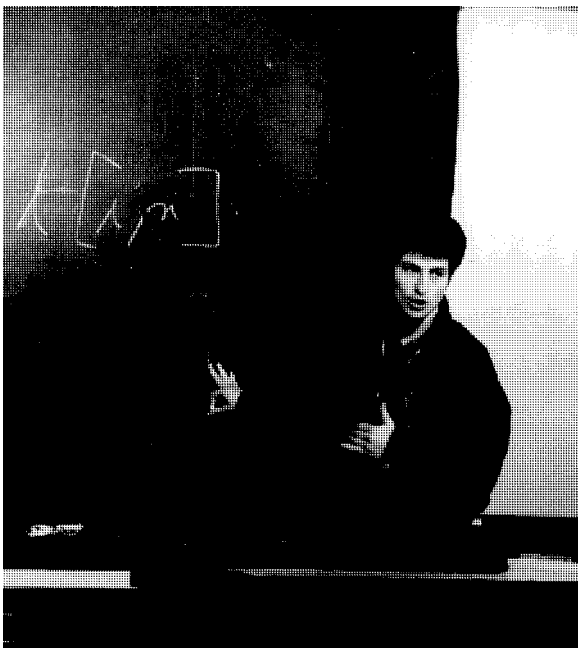
du pli aux fractales

par Olivier Ruis et Damien Schneider, DEUG A1, Université d'Aix-Marseille II

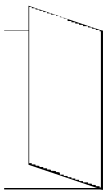
chercheur : Christian Mauduit

Nous remercions notre tuteur L. Beddou (D.E.A. d'informatique) et I. Perez (DEUG MASS 1), qui travaillent sur le même projet, pour leur collaboration.



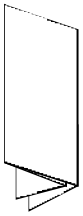
[NDLR : ce sujet ressemble au précédent (page 105), mais la façon de le traiter est assez différente et les mathématiques y restent bien cachées ; au fait, quels étaient les sujets proposés aux élèves, dans un cas comme dans l'autre ?]

Prenez une feuille de papier et pliez-la en deux (1ère itération) :

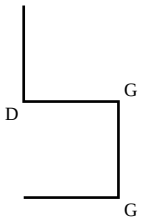


Il s'agit ici de ce qu'on appellera un « pli à gauche ».

Repliez-la une seconde fois, sans déplier le premier pli, et dans le même sens (2ème itération) :



Ensuite, dépliez la feuille en laissant un angle de 90° entre les "faces" ainsi obtenues ; en regardant la feuille de profil, vous obtenez ce motif :



Le premier pli, en bas à droite, est un ***pli à gauche***, ainsi que le suivant immédiatement au dessus. Le troisième pli est ***à droite***.

Il existe une infinité de possibilités dans le cadre de ce genre d'étude : changement du nombre et de l'orientation des plis à chaque itération, puis des angles lors du dépli.

Notre étude s'arrêtera à ce cas du pli systématique en deux à gauche et à droite, et du dépli avec un angle de 90° entre les faces.

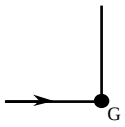
étude préliminaire

Nous allons étudier les premières itérations du raisonnement.

— pli systématique à gauche à chaque itération :

• 1ère itération : 1 seul pli. Nous écrirons la succession des plis par une série de *G* (pli à gauche) et de *D* (pli à droite).

Motif :

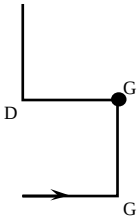


• 2ème itération : 3 plis.

GGD

Le pli de l'itération est celui qui est situé au milieu de la totalité de la série.

Motif :

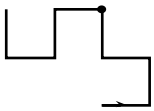


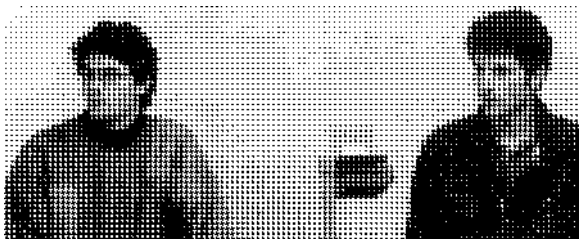
• 3ème itération : 7 plis.

GGDGGDD

La série en gras est la même que la totalité de la série à l'itération précédente. Elle est située à gauche du pli de l'itération.

Motif :





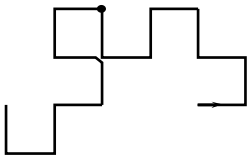
• 4ème itération : 15 plis.

GGDGGDD*GGGDDGDD*

La série en italique est l'"antisymétrique" par rapport au pli de l'itération, de la série en gras.

C'est-à-dire : chaque pli de de la première série a son correspondant dans la seconde, mais avec un changement d'orientation. [NDLR : ceci quand on regarde deux lettres situées à des emplacements symétriques par rapport au G central, dit "pli de l'itération".]

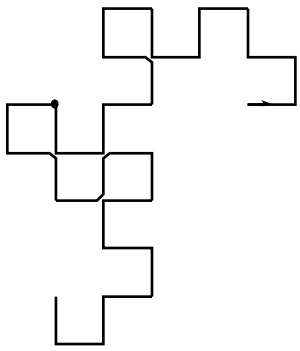
Motif :



• 5ème itération : 31 plis.

GGDGGDDGGGDDGDD*GGGDGGDDD GGDDGDD*

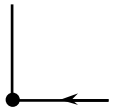
Motif :



— pli systématique à droite à chaque itération :

• 1ère itération : 1 seul pli.

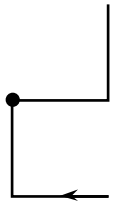
Motif :



• 2ème itération : 3 plis.

DD*G*

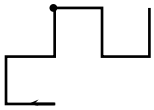
Motif :



• 3ème itération : 7 plis.

DDGD*DGG*

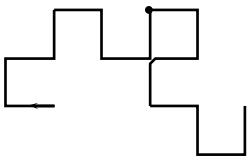
Motif :



• 4ème itération : 15 plis.

DDGDDGGD*DDGGDGG*

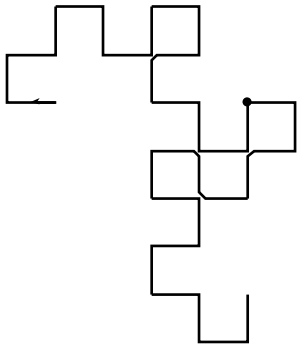
Motif :



• 5ème itération : 31 plis.

DDGDDGGDDDGGDGGD*DDGDDGGG DDGGDGG*

Motif :



Par rapport aux séries, on peut faire les mêmes remarques que pour le pli systématique à gauche.

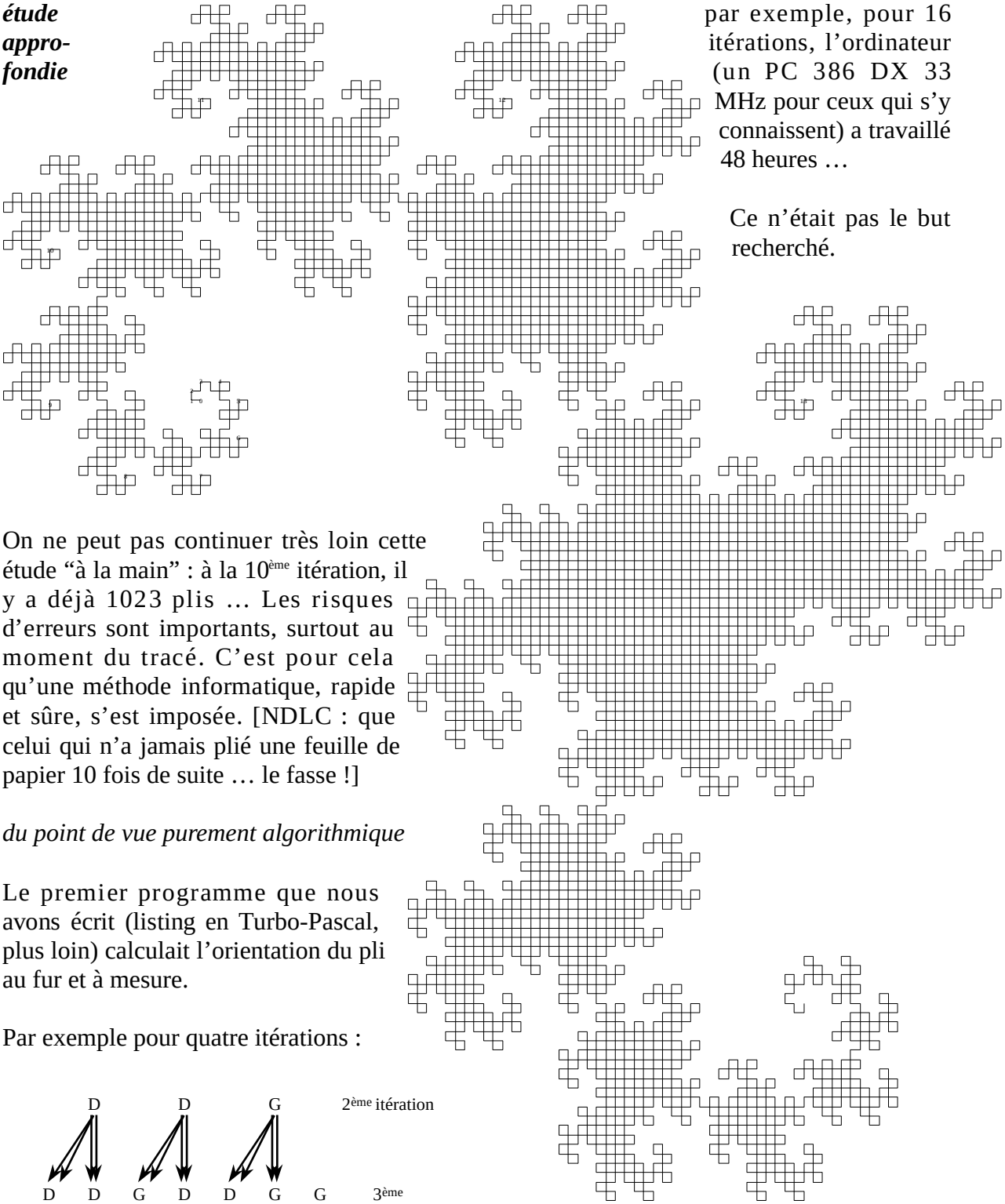
— comparaison & similitudes :

En passant d'un cas à l'autre, on remarque que l'on obtient exactement la même succession de plis, à part l'orientation qui est exactement opposée. Sur le motif, cela se traduit par une symétrie d'axe vertical.

Le nombre de plis à la $n^{ème}$ itération est $2^n - 1$. Ou encore :

2×(nombre de plis à l'itération précédente)+1

étude approfondie



par exemple, pour 16 itérations, l'ordinateur (un PC 386 DX 33 MHz pour ceux qui s'y connaissent) a travaillé 48 heures …

Ce n'était pas le but recherché.

On ne peut pas continuer très loin cette étude "à la main" : à la 10ème itération, il y a déjà 1023 plis … Les risques d'erreurs sont importants, surtout au moment du tracé. C'est pour cela qu'une méthode informatique, rapide et sûre, s'est imposée. [NDLC : que celui qui n'a jamais plié une feuille de papier 10 fois de suite … le fasse !]

du point de vue purement algorithmique

Le premier programme que nous avons écrit (listing en Turbo-Pascal, plus loin) calculait l'orientation du pli au fur et à mesure.

Par exemple pour quatre itérations :

Sans entrer dans les détails, il faut savoir que ce genre d'algorithme doit recalculer, pour chaque pli, toutes les itérations précédentes, d'où une perte de temps considérable :

Nous avons alors eu l'idée d'un autre algorithme qui calcule une fois pour toutes l'orientation des plis (listing en langage C, plus loin aussi). Le principe est très simple et utilise les propriétés des séries énoncées dans l'étude préliminaire [NDLR : ce sont des propriétés établies, ou seulement des observations ?] ; il suffit alors de remplir un tableau

en binaire. Nous avons choisi arbitrairement 0 pour un pli à Gauche, et 1 pour un pli à Droite. Le gain de temps est alors impressionnant : pour 21 itérations, l'ordinateur a travaillé 10 minutes. Le problème n'était alors plus le temps mais la mémoire disponible : à chaque itération supplémentaire, la taille du tableau doublait, nous limitant à une quarantaine d'itérations. Nous travaillons en ce moment à une alternative entre les deux algorithmes et à une généralisation à n'importe quel angle (non plus seulement 90°).

du pli aux fractales

Voir page précédente le motif obtenu pour 13 itérations, avec un pli systématique à droite : il y a 8191 plis (les couleurs de chaque segment ont été choisies aléatoirement par l'ordinateur). C'est une fractale, un objet très irrégulier et complexe, obtenu la plupart du temps à partir d'algorithmes mathématiques très simples (comme c'est le cas ici), et qui possède certaines propriétés : par exemple, quelle que soit l'échelle d'observation, on retrouve toujours le même motif de base.

En agrandissant une partie du motif ci-dessus, nous retrouvons celui de la 5ème itération. Nous avons comparé les deux motifs en les superposant : le motif de la 5ème itération (agrandi) est à l'intérieur des limites de celui de la 13ème : ils se superposent exactement, ils ont la même "forme".

Nous avons voulu savoir ensuite la façon dont se remplissait le motif au fur et à mesure qu'il se dessinait. Nous avons pour cela programmé l'ordinateur pour qu'il change de couleur à partir d'un certain moment. Par exemple sur le motif obtenu pour 21 itérations, avec un pli systématique à droite (à ce stade, le motif possède plus de 2 millions de segments et le dessin sort de l'écran) : le motif se remplit toujours à partir du même motif spiralé, que l'on observe déjà à la 13ème itération. Nous avons pris une partie du motif en double, et nous avons placé les morceaux de manière à bien voir la superposition.

[NDLR : les dessins, en couleurs, ne passant pas dans cette publication en noir et blanc, (voir dessin de la page précédente) nous n'avons gardé que les commentaires des auteurs ; libre au lecteur d'utiliser les programmes pour contrôler par lui-même.]

Le rapport s'arrête ici. Nos recherches continuent, mais nous n'avons pas encore pu confirmer les nouvelles données que nous avons.

annexes : les programmes

listing en Turbo-Pascal

```
program fract;
uses crt,graph3,
  Graph;
VAR
  Graphilot : INTEGER;
  GraphMode : INTEGER;
  CodeErreur : INTEGER;
 c,x1,x2,yl,y2,d,e:integer;
  ch1:char;
  i:longint;
 procedure t(ch2:char);
 var tx,ty:integer;
 begin
 tx:x2;ty:=y2;
  if ch2='D' then begin
    if yl>y2 then   x2:=x2+d
    else if y1<y2 then  x2:=x2-d
    else if x1>x2 then  y2:=y2-d
    else if x1<x2 then  y2:=y2+d
    lineto(x2,y2);
    end;
  if ch2='G' then begin
    if yl>y2 then   x2:=x2-d
    else if y1<y2 then  x2:=x2+d
    else if x1>x2 then  y2:=y2+d
    else if x1<x2 then  y2:=y2-d
    lineto(x2,y2);
    end;
  x1:=tx;
  y1:=ty;
 end;
 function n(m:integer);longint;
 var w,v:longint;
  begin
   v:=0;
   for w:=0 to m do
    begin
   v:=2*v+1;
   end;
   n:=v;
  end;
 Function impaire (i,r:longint):char;
  var a,j:longint;
   ch:char;
   begin
    ch:=' ';
    a:=1;
    for j:=1 to (n(r)) do
     begin
      if i=a then ch:='D';
      a:=a+4;
     end
     if ch<>'D' then ch:='G';
     impaire:=ch;
 end;
 Function paire (k:longint):char;
  var g,b:longint;
 begin
  g:=e; b:=2;
  while (round(i/b) mod 2)=0 do
   begin
    b:=b*2;
    dec(g);
   end;
```

```
   if (round(i/b)mod 2)<>0 then dec(g);
   paire:=impaire(round(i/b),g);
  end;
begin
 randomize;
 clrscr;
 writeln('Entrez le nombre d''itérations');
 readln(e);
 writeln('Distance élémentaire');
 readln(d);
  GraphPilote := Detect;
 InitGraph(GraphPilote,GraphMode,'');
  CodeErreur := GraphResult;
  c:=0;
  x1:=320;y1:=240;x2:=xl;y2:=yl-d;
  line(x1,y1,x2,y2);
  moveto(x2,y2);
 for i:=1 to n(e) do
  begin
     if c=17 then c:=0
  else inc(c);
  setcolor(c);
    if (i mod 2) <>0 then t(impaire(i,e))
     else t(paire(i));
  end;
  chl:=readkey;
  chl:=readkey;
  end.
```

(fin du listing en Turbo-Pascal)

listing en langage C

```
/* Zorglub */
#include<stdlib.h>
#include<conio.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>
#define Nbbits 16
typedef unsigned short mot;
typedef unsigned long entier;
#define CM(n) (n/Nbbits)
#define CB(n) (n%Nbbits)
mot *table;
mot bit[Nbbits];
 entier d;
int x1,y1,x2,y2,d1;
/***********************************************/
/* Creation de la table et remplissage des bits */
/***********************************************/
void initialisation(entier t)
{
 int i;
 for(i=0;i<Nbbits;i++) bit[i]=1<<i;
 if((table=(mot *)malloc(sizeof(mot)*t/Nbbits))==NULL
 {
  fprintf(stderr,"Pas assez de memoire...\n");
  exit(-2);
 }
 for(i=0;i<sizeof(mot)*t/Nbbits;i++) table[i]=0;
}
/*****************************/
/* Teste la valeur d'un bit */
/*****************************/
int test(entier num)
{
 return (table[CM(num] & bit[CB(num)]);
}
/*****************/
/* Allume un bit */
/*****************/
void allume(entier num)
{
 table[CM(num)]=table[CM(num)]|bit[CB(num)];
}
/*****************/
/* Eteint un bit */
/*****************/
void eteint(entier num)
{
 table[CM(num)]& ~bit[CB(num)];
}
/***************/
/* ((a+1)^b)-1 */
/***************/
/* parametres b:nombre d'iterations 1-->oo a:nb de
virages de la 1er iteration */
/* retourne nb virage */
entier nbv(int a,int b)
{
 return (entier) (pow((a+1),b)-1);
}
/**************/
/* parametres n:iteration p:nb iteration de la 1er
ligne, q:nb virages initial */
/* retourne l'espace que l'on doit sauter dans le
tableau final */
/**************/
entier esp(int n,int p,int q)
{
 return (entier)(((nbv(q,n)-bnv(q,p))/(nbv(q,p)+1))+1);
}
/*****************************************/
/* trace d'une ligne dans une direction ch2 */
/*****************************************/
void tt(int ch1)
{int tx,ty;
 tx=x2;ty=y2;
  if (ch==1)
  {if (y1>y2) x2=x2+d1;
    else if (y1<y2) x2=x2-d1;
    else if (x1>x2) y2=y2-d1;
    else if (x1<x2) y2=y2+d1;
    lineto(x2,y2);
    }
    x1=tx;
  y1=ty;
}
int main()
{
 int   GraphDriver; /* The Graphics device driver */
 int   GraphMode;   /* The Graphics mode value    */
 int   ErrorCode;   /* Reports any graphics errors*/
 int b,t;
 int i,j,bool=1;
 int nit;
 char car[255];
 clrscr;
 printf("Nombre d'iterations :");
 scanf("%d",&nit);
 b=nbv(1,nit);
 initialisation(nbv(1,nit));
 b=esp(nit,1,1);
 allume(esp(nit,1,1)-1);
 for(i=2;i<=nit;i++)
 {
  for(t=0;t<nbv(1,nit);t++)
  {gotoxy(2,2); printf(test(t)?"* ":"0 ");}
  d=esp(nit,i,1);
  printf("%d\n",i);
  for(j=0;j<(nbv(1,i)-1);j++)
  {gotoxy(2,3);
   printf("%d\n",j);
   bool=!bool;
 /*   allume(j*2*d+d-1);
   if((nit-i)%2)     */
     {if(bool) allume(j*2*d+d-1);}
 /* else
     {if(!bool) allume(j*2*d+d-1);}*/
  }
 }
/* for(i=0;i<nbv(1,nit);i++) printf(test(i)?"1":"0");
 printf(" %d\n",nbv(1,nit));*/
 gets(car);
 gets(car);
  GraphDriver = DETECT; /* Request auto-detection */
  initgraph( &GraphDriver, &GraphMode, "" );
  ErrorCode = graphresult(); /* Read result of
                                  initialisation */
  if( ErrorCode != grOk ){  /* Error occured during
                                      init */
    printf(" Graphics System Error: %s\n",
                       grapherrormsg( ErrorCode ) );
    exit( 1 );}
  x1=320;y1=240;x2=x1;y2=y1-d;
  line(x1,y1,x2,y2);
  moveto(x2,y2);
  d1=2;
 for(i=0;i<nbv(1,nit);i++)
 {setcolor(random(17));
 if (test(i)) {tt(1);}
 else {tt(0);}}
}
```

(fin du listing en langage C)