MASTER EN SCIENCES INFORMATIQUES

Génération d'images fractales et implémentation en langage objet

Heinen, Wolfgang

Award date:
1992

Awarding institution:
Universite de Namur

Génération d'images fractales et implémentation en langage objet

Wolfgang HEINEN

Promoteur :
Monsieur J. FICHEFET

Mémoire présenté en vue
de l'obtention du titre
de Licencié et Maître
en Informatique

Année académique 1991 - 1992

Ici, je veux remercier Mr. Fichefet qui a bien voulu assurer la direction de mon mémoire. C'est son aide constructive qui m'a permis d'aboutir au présent résultat.

Je remercie également Mr. Lecharlier et Mme DeBaenst qui m'a aidé dans mes recherches bibliographiques.

Table des matières

6. L'environnement objet WINDOWS
   6.1. La programmation orientée objet
   6.2. Don't call me, I'll call you !
   6.3. Le premier programme sous Windows
   6.4. L'éditeur d'images fractales

0. Introduction

Benoit Mandelbrot définit grosso-modo une fractale comme une figure géométrique dotée de la propriété d'invariance d'échelle, en ce sens que ses parties, aussi petites soient-elles, sont similaires ou quasi-similaires à sa totalité.

Ainsi, un segment de droite a toujours la même apparence, que la droite qui le porte, quelque soit sa taille. Un cercle, par contre, devient une droite, si on effectue des rapprochements successifs vers une partie de celui-ci, il change donc.

On peut comparer une fractale à une côte : elle devient de plus en plus irrégulière, au fur et à mesure qu'on s'en approche, mais, d'une manière générale, la côte reste semblable à elle-même. Il n'y a que la droite et des formes comme les côtes qui vérifient cette propriété.

Ainsi, dans le plan, les fractales sont des formes qui possèdent la propriété d'auto-similitude et qui ne sont pas des droites (voir figure 0.1.).

Il faut cependant insister sur le fait que la définition précédente n'est pas universelle et dépend dans une certaine mesure du point de vue selon lequel on considère les fractales. Ainsi, certains diront-ils qu'une fractale est un objet géométrique possédant une "dimension fractionnaire". D'autres, et ce sera notre cas, considéront les fractales comme des attracteurs de "systèmes dynamiques particuliers".

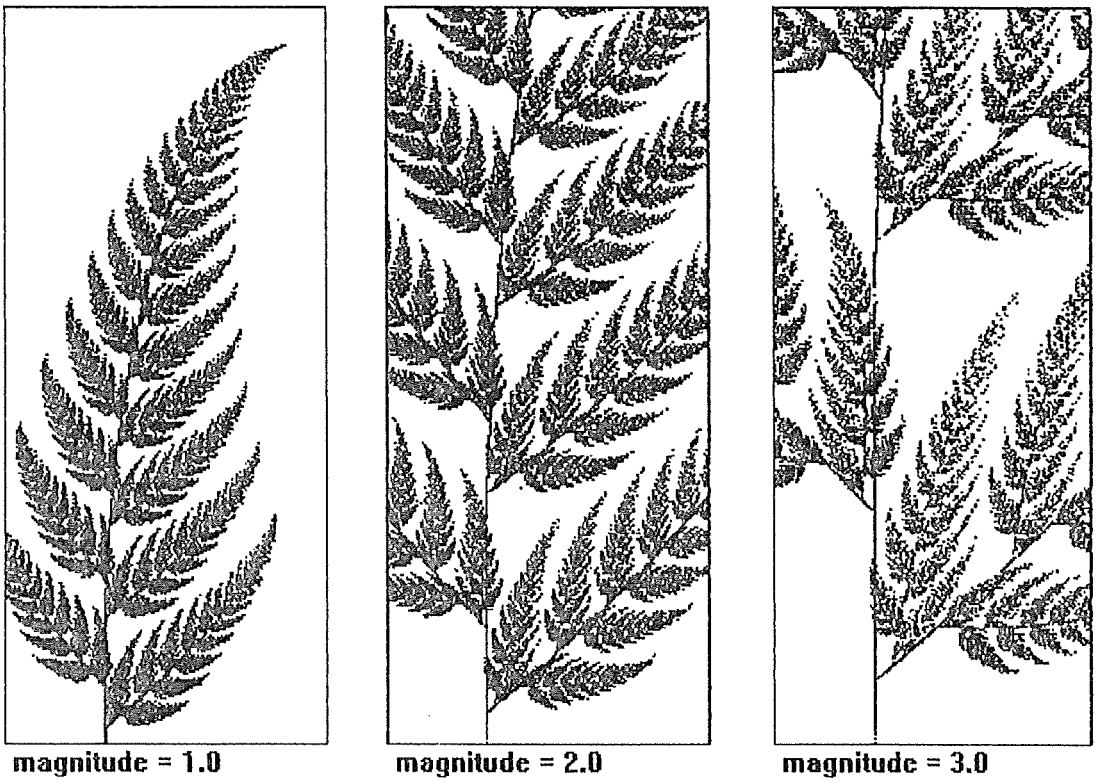


Figure 0.1.(a,b,c)

Une chose est de créer des images qui ressemblent aux objets réels, une tout autre démarche est de trouver les règles à partir desquelles sont construits ces objets. Ainsi, une fougère (fern en anglais) réelle possède des embranchements successifs, mais peut-on formuler la règle qui permet de dire que cette fougère doit croître de telle façon et ne pas d'une autre manière ?

La fougère de la figure 0.1.a. a été créée par ordinateur, et semble représenter parfaitement l'objet réel. Pour ce faire, on exige une sorte de "gymnastique" de la part de l'ordinateur :
ce sont des "transformations affines" qui forment les règles déterminant la construction de la fougère.

C'est au premier chapitre, que sera décrit l'espace mathématique où vivent les fractales. On y trouvera la signification d'une transformation affine et on déterminera sur quoi elle agit pour former l'image fractale. Afin de permettre une génération de ces images, des algorithmes avec leurs propriétés sont proposés au second chapitre.

Afin d'élargir l'horizon défini par les fractales, on peut considérer des structures fractales qui contiennent et qui agissent sur d'autres objets fractals. Le chapitre 3 développe cette idée selon une conceptualisation et une méthodologie qui, à notre connaissance, sont originales. On y trouvera des algorithmes qui sont dérivés des principes du deuxième chapitre. Le quatrième chapitre nous donne la possibilité de colorer une image fractale. On y abordera la théorie des mesures qui donne la définition et la justification mathématique.

Une image fractale se distingue d'un objet géométrique ordinaire par sa complexité. La génération d'une image fractale n'est donc pas aussi immédiate que celle d'un triangle ... et pourtant, on aimerait bien de parler d'un objet géométrique possédant d'une part des propriétés communes à toutes les figures et d'autre part des propriétés particulières. Le langage objet présenté au chapitre 5 permet une traduction immédiate de ces besoins en programmation. On discutera ensuite des nombreuses actions devenues possibles grâce à cette nouvelle philosophie de programmation.

Le sixième et dernier chapitre concerne l'interface orienté objet 'Windows'. Lors de la programmation d'une application Windows, on est confronté de nouveau à une approche objet. On va élargir ainsi la programmation objet à celle d'une interface objet et on y retrouvera des principes similaires.

1. L'espace où vivent les fractales

Dans ce chapitre, le monde dans lequel vivent les fractales va être construit au fur et à mesure. Le point de départ sera un espace métrique 'ordinaire'.
La géométrie fractale est basée sur de simples espaces géométriques. Un tel espace est noté X. C'est l'espace dans lequel on va dessiner les fractales. Jusqu' ici, on peut considérer une fractale comme un simple sous-ensemble d'un espace.

1.1. Espaces Métriques

Definition 1 :
Un espace métrique est un couple (X,d) formé d'un ensemble X d'éléments appelés **points** et d'une fonction à valeurs réelles
$d : X \times X \rightarrow \mathbb{R}$ qui mesure la distance de tous les couples de points x et y de X et vérifie les axiomes suivants :

i. $d(x,y) = d(y,x) \quad \forall x,y \in X$ (axiome de symétrie)
ii $0 < d(x,y) < \infty \quad \forall x,y \in X$ et si $x \neq y$
iii. $d(x,x) = 0 \quad \forall x \in X$
iv. $d(x,y) < d(x,z) + d(z,x) \quad \forall x,y,z \in X$ (inégalité triangulaire)

Cette fonction d est appelée une **métrique** de l'ensemble X.

exemples : $d(x,y) = \sqrt{(x1 - y1)^2 + (x2 - y2)^2}$ est la métrique Euclidienne dans $X = \mathbb{R}^2$
$d(x,y) = |x1 - y1| + |x2 - y2|$, la métrique de Manhattan dans $X=\mathbb{R}^2$
$d(x,y) = |xy|$ n'est pas une métrique dans $X = \mathbb{R}$

Définition 2 :
Deux métriques d_1 et d_2 définies sur un espace X sont des **métriques équivalentes** si il existe deux constantes $0<c_1<c_2<\infty$ telles que :

$$c_1 d_1(x,y) < d_2(x,y) < c_2 d_1(x,y) \quad \forall (x,y) \in X \times X$$

Le fait d'avoir deux métriques équivalentes implique que ces deux mesures donnent la même notion de proximité, en ce sens que si deux points sont proches pour l'une ils le restent pour l'autre métrique.

On peut donc envisager des déformations de l'espace qui laissent les métriques équivalentes.

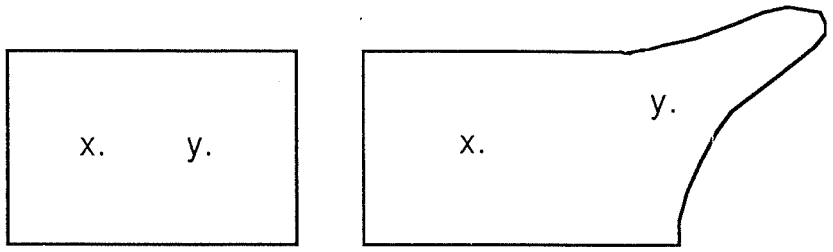


Figure 1.1 : Les deux métriques peuvent être équivalentes si la déformation ne mène pas à des ruptures, à des extensions à l'infini, ...

Définition 3 :
Deux espaces métriques (X_1,d_1) et (X_2,d_2) sont **des espaces métriques équivalents** si il existe une fontion $h : X_1 \rightarrow X_2$ biunivoque, telle que la métrique d_3 définie dans X_1 par :

$$d_3(x,y) = d_2(h(x) , h(y)) \quad \forall (x,y) \in X_1$$

est équivalente à d_1.

On peut donc respecter l'équivalence des espaces en compressant ou en étirant ces espaces.

1.2. Suites de Cauchy

La géométie fractale se fonde sur la description, la classification et l'analyse de sous-ensembles d'espaces métriques (X,d). Les espaces métriques sont généralement d'un caractère 'simpliste' du point de vue géométrique; les sous-ensembles par contre peuvent être très 'complexes'. Un certain nombre de propriétés des sous-ensembles d'espaces métriques apparaissent encore et encore. (exemple : ouverture, fermeture,...)

Ce qui importe pour la conception des fractales, c'est l'existence de propriétés qui restent invariantes pour des espaces métriques équivalents. Parmi ces propriétés figurent celles d'être fermé, borné, complet et compact. On y ajoutera plus tard la dimension fractale d'un sous-ensemble de X.

Ce paragaphe est consacré aux espaces métriques complets.

Définition 1 :
Une séquence $\{x_n\}_{n=1}^{\infty}$ de points d'un espace métrique (X,d) est appelée **suite de Cauchy**, si pour tout $\varepsilon > 0$, il existe un entier N>0 tel que :

$$d(x_n,x_m) < \varepsilon \quad \text{pout tous } n,m>N$$

En d'autres mots, plus on progresse dans la séquence, plus proches sont les points de la suite. Remarquons que cela ne veut pas dire nécessairement que la suite converge vers un point:

Définition 2 :

Une suite de points $\{x_n\}_{n=1}^{\infty}$ d'un espace métrique (X,d) est **convergente** vers un point $x \in X$ (le point limite) si, pour tout $\varepsilon > 0$, il existe un entier $N>0$ tel que

$$d(x_n,x) < \varepsilon \text{ pour tout } n > N.$$

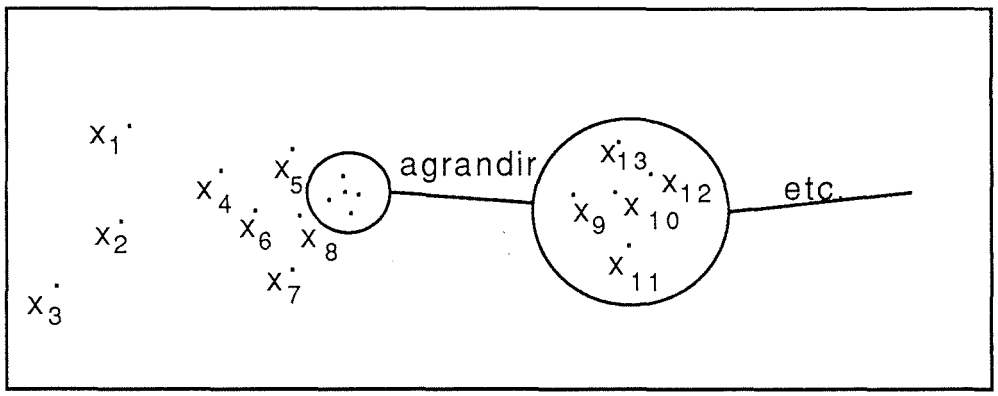


Figure 1.2. : Cette image représente des agrandissements successifs d'une suite de Cauchy dont les points se rapprochent de plus en plus.

On démontre que, si une suite de points $\{x_n\}$ d'un espace métrique (X,d) converge vers un point $x \in X$, alors $\{x_n\}$ est une suite de Cauchy.

Définition 3 :

Un espace métrique (X,d) est **complet** si toute suite de Cauchy $\{x_n\}_{n=1}^{\infty}$ de X converge vers un point limite $x \in X$.

Définition 4 :

Si $S \subset X$ est un sous-ensemble d'un espace métrique (X,d), on dit qu'un point $x \in S$ est un **point limite de S** s'il existe une suite $\{x_n\}_{n=1}^{\infty}$ de points $x_n \in S - \{x\}$ qui converge vers x. La **fermeture** de S, notée $\bar{S}$, est définie comme étant $\bar{S} = S \cup \{\text{points limites de S}\}$. S est **fermé** si il contient tous ses points limites.

Définition 5 :

Soit $S \subset X$ est un sous-ensemble d'un espace métrique (X,d). S est **compact**, si toute suite infinie contient une sous-suite convergente vers un point limite de S. S est **borné** s'il existe un point $a \in X$ et un nombre réel $r > 0$ tels que :

Pour tout $x \in S$: $d(a,x) < r$.

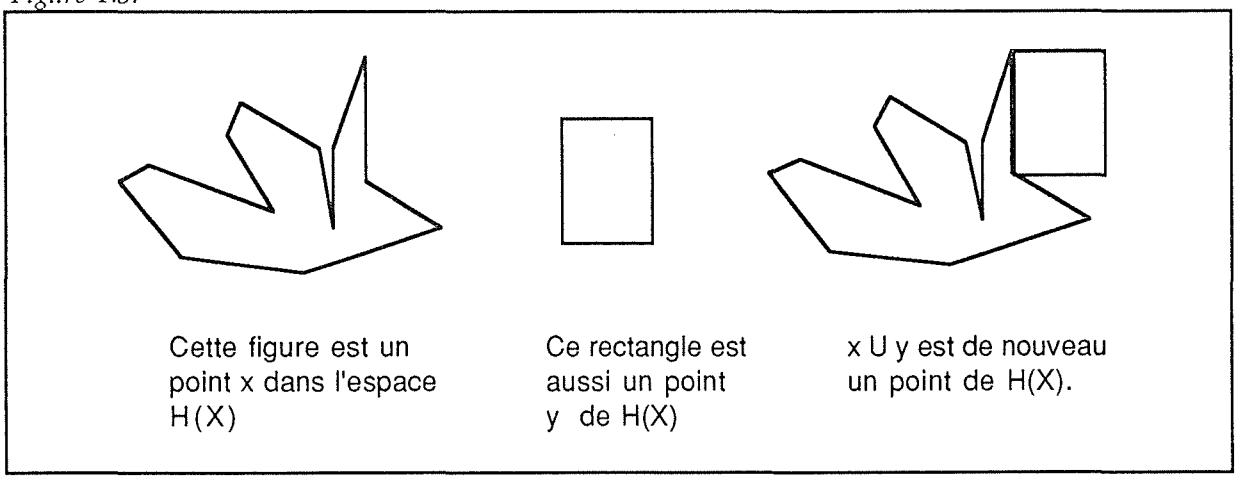
Un Théorème établit une définition plus maniable lorsqu'on travaille dans un espace complet: Soit (X,d) un espace métrique complet.
Si $S \subset X$, alors S est **compact** ssi S est fermé et borné.

1.3. L'espace métrique (H(X),h(d))

C'est l'espace où vivent les fractales ! On part du niveau le plus bas, c'est à dire d'un espace métrique (X,d) qui sert comme 'papier' sur lequel on a dessiné des images qui sont des sous-ensembles de l'espace. L'espace (X,d) peut être par exemple l'ensemble $\mathbb{R}^2$ muni de la métrique Euclidienne. Pour travailler sur ces sous-ensembles, on introduit l'espace H(X) :

Définition 1 :
Soit (X,d) un espace métrique. Alors H(X) est l'espace dont les points sont les ensembles compacts non vides de X.

Figure 1.3.



Définition 2 :
Soient (X,d) un espace métrique complet, $x \in X$, et $B \in H(X)$. Le nombre d(x,B) défini par:

$$d(x,B) = \text{Min} \{ d(x,y) : y \in B \}$$

est appelé la distance de x à l'ensemble B.

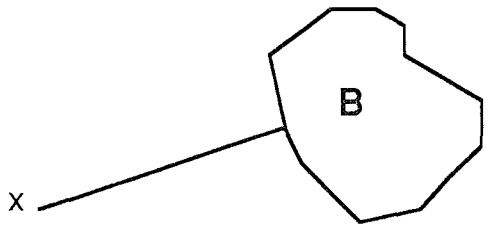


Figure 1.4.

L'existence de ce nombre d(x,B) est assurée par le fait que $B \in H(X)$ est compact et non vide. Afin de la montrer, considérons la fonction $f : B \to \mathbf{R}$ définie par : $f(y) = d(x,y)$ pour tout $y \in B$.

Rappelons d'abord la définition d'une application continue $g : X \to X'$ d'un espace métrique (X,d) dans un espace métrique (X,d').

On dit que g est **continue au point** $a \in X$ ssi

$\forall\, \varepsilon > 0$, il existe un $\delta > 0$ tel que

$d'(g(x),g(a)) < \varepsilon$, $\forall\, x$ tel que $d(x,a) < \delta$.

On dit que g est **continue** si elle est continue en tout point de X. On peut démontrer qu'une condition nécessaire et suffisante pour que g soit continue au point $a \in X$ est que, pour toute suite $\{x_n\}$ de X qui converge vers a, la suite correspondante $\{g(x_n)\}$ converge vers g(a).

Rappelons encore que si $X'=\mathbb{R}$ et si g est continue, on dit que g est une **fonction continue**.

On peut vérifier à partir de la définition de la métrique, que f est continue, si on la considère comme une application de l'espace métrique (B,d) dans l'espace ($\mathbb{R}$,Métrique Euclidienne). Soit donc P défini par $P = \text{Inf}\ \{ f(y) : y \in B\}$ avec la définition habituelle :

si il n'existe pas de y' tel que $y' < f(y)$: $\forall y \in B$, alors $P = -\infty$

sinon

$$\text{Inf}\ \{ f(y) : y \in B\} = \text{Max}\ \{ x \in \mathbb{R} : x < f(s), \forall\, s \in B \}$$

Comme $f(y)>0$ pour tout y, il suit que P est fini. Il suffit donc de se concentrer uniquement sur $P = \text{Max}\ \{ x \in \mathbb{R} : x < f(s), \forall\, s \in B \}$ et de remarquer que l'existence d'une valeur pour l'infimum revient à dire qu' il existe un point $y'' \in B$ tel que $f(y'') = P$.

On prend enfin une suite infinie $\{y_n\}_{n=1}^{\infty}$ de B telle que $| f(y_n) - P | < 1/n$.

C'est ici que la compacité de B intervient :

on sait que la suite infinie $\{y_n\}_{n=1}^{\infty}$ contient une sous-suite convergente dans B qui est indexée par n_1 . En d'autres mots, il existe $y'' \in B$ tel que $\lim_{1 \to \infty} \{y_{n1}\} = y''$.

Prenant la limite sur n_1 et tenant compte de la continuité de f on passe de :
$| f(y_{n1}) - P | < 1/n_1$ à $f(y'') - P = 0$,

i.e. $f(y'') = P$ et en d'autre mots encore : il existe un y'', $d(x,y'') = \text{Inf}\ \{ d(x,y) : y \in B \}$ ce qui prouve bien qu'il existe une valeur minimale pour $\{ d(x,y) : y \in B \}$.

Pour définir l'espace des fractales H(X) ou 'les points sont des ensembles de X', il s'impose maintenant de parler de la distance entre deux ensembles :

Définiton 3 :
Soit (X,d) un espace métrique complet et soient A,B ∈ H(X). On définit

$$d(A,B) = \text{Max}\{ d(x,B) : x \in A\}.$$

comme étant la **distance de l'ensemble** A ∈ H(X) à l'**ensemble** B ∈ H(X).

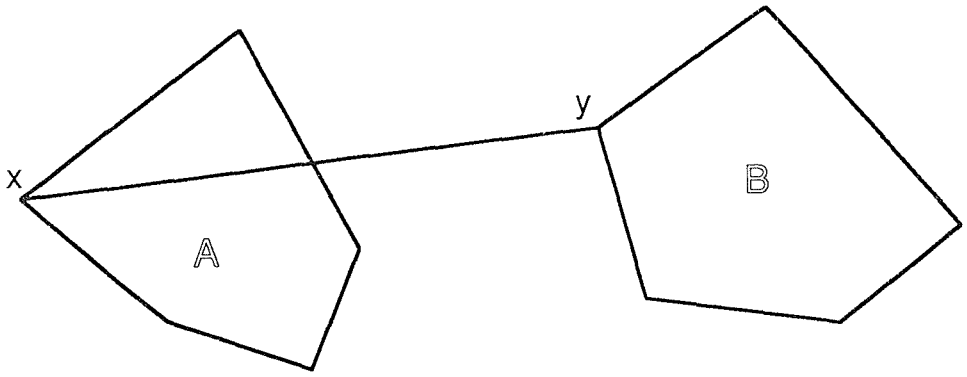


Figure 1.5.

De nouveau, on peut montrer que cette définition a un sens et en particulier, qu'il existe x' ∈ A et y' ∈ B tels que d(A,B) = d (x',y').

On pourrait évidemment penser à adopter cette distance comme métrique pour l'espace H(X). Rein n'assure cependant que d(A,B) = d(B,A), en effet :

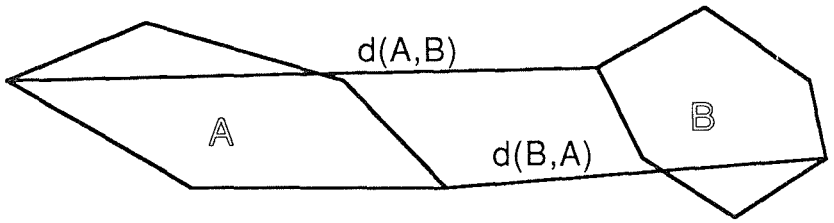


Figure 1.6. : d(A,B) > d(B,A)

L'astuce consiste à définir la distance de Hausdorff entre deux points A et B de H(X) comme:

$$h(A,B) = \text{Max}\ \{ d(A,B)\ ,\ d(B,A)\ \}$$

Cette fois-ci on peut sans beaucoup de réflexion montrer que h est une métrique sur H(X).

Arrivé en ce point de l'exposé, il est sans doute encore trop tôt pour fournir une définition exacte aux fractales. Contentons-nous momentanément de mentionner que l'espace (H(X),h) est l'espace des fractales et que sa complétude est une des propriétés fondamentales :

Théorème 1 :

Soit (X,d) un espace métrique complet. Alors (H(X),h(d)) est aussi un espace métrique complet. On peut alors dire que si $\{ A_n \in H(X) \}$ est une suite de Cauchy, alors

$$A = \lim_{n \to \infty} \{A_n\} \in H(X)$$

où $A = \{ x \in X$ tel qu'il existe une suite de Cauchy $\{ x_n \in A_n \}$ qui converge vers $X \}$

Ce théorème est crucial car il explique pourquoi (H(X),h(d)) EST l'espace des fractales. On le montrera ultérieurement de façon formelle, mais on peut déjà justifier intuitivement pas mal de choses :
Considérons la figure 1.7. : on voit que les ensembles se 'ressemblent' de plus en plus avec n croissant, i.e. la distance entre A_{n-1} et A_n devient arbitrairement petite avec n croissant. Or cela veut dire que les ensembles A_0 à A_n forment un suite de Cauchy. Par le théorème, cette suite converge vers une 'image finale' qui, elle aussi, est incluse dans l'espace de Haussdorff.

En regardant l'image, on se rend compte qu'il existe certaines règles qui permettent de passer d'une image à l'autre. Chaque fois, une image entière est transformée et copiée plusieurs fois pour donner l'image suivante. En admettant que ce passage est une 'transformation', il va être possible de montrer dans le chapitre suivant qu'elle admet -sous certaines conditions- un point fixe dans l'espace de Haussdorff.

Ce processus itératif aboutit donc à un POINT fixe dans H(X), en d'autre mots à une image fixe dans X : L'IMAGE FRACTALE. On va donc trouver l'espace des fractales !

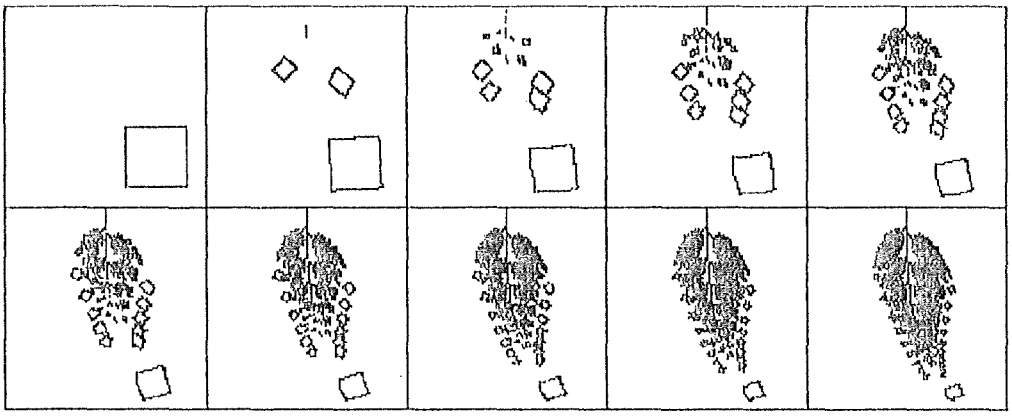


figure 1.7.

1.4. Les transformations affines

1.4.1. Transformations sur des espaces métriques ordinaires

Dans la géométrie fractale, les sous-ensembles qui sont générés par de simples transformations géométriques de l'espace vers lui-même sont considérées. Plus concrètement, on va s'occuper ici de transformations affines dans $\mathbb{R}^2$, qui sont exprimées par une matrice 2x2 et un 2-vecteur. Plus formellement :

Définition 1 :
Soit (X,d) un espace métrique. Une **transformation** de X est une application f: X -> X, qui associe à chaque point $x \in X$ un et un seul point $f(x) \in X$.
Si S C X, alors

$$f(S) = \{ f(x) : x \in S \}.$$

Définition 2 :
Une transformation w: $\mathbb{R}^2$ -> $\mathbb{R}^2$ de la forme

$$w(x_1,x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f)$$

où a,b,c,d,e et f sont des nombres réels,
est appelée une **transformation affine**.

Souvent, on donne une écriture équivalente sous forme matricielle :

$$W(x) = Ax + t,$$

avec $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ et $t = \begin{pmatrix} e \\ f \end{pmatrix}$

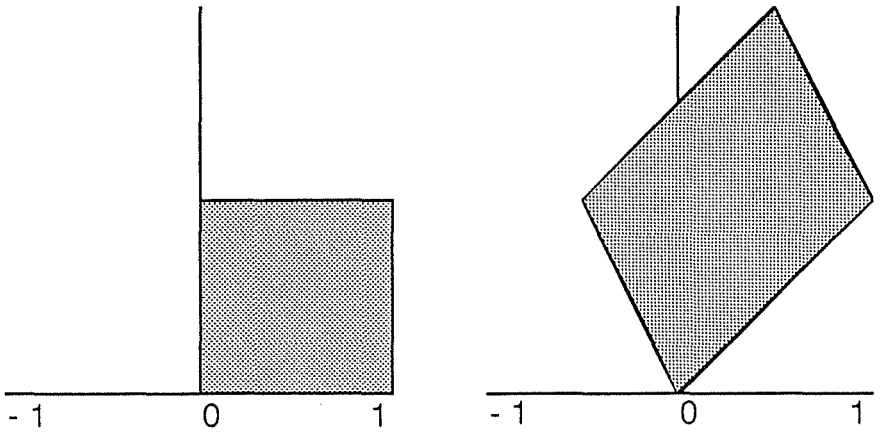


Figure 1.8.

Quelques propriétés sont essentielles pour la compréhension des effets des transformations. Considèrons par exemple la transformation de la figure 1.8. exprimée par : x -> Ax.
C'est une transformation linéaire dans $\mathbb{R}^2$ qui copie un parallélogramme avec un coin sur l'origine dans un autre parallélogramme dont l'origine est de nouveau un coin.

Finalement, une transformation affine n'est rien d'autre qu'une transformation linéaire suivie d'une translation réalisée par le vecteur t.

1.4.2. Le changement de coordonnées

En décrivant des transformations d'espaces, on fait référence aux systèmes de coordonnées sous-jacents à ces espaces. Ceci présente un moyen de localiser les points dans cet espace. Par exemple, l'espace $X = [0,1] \times [0,1]$: un carré qui donne une collection de points avec leur coordonnées (x_1, x_2) restreints par $0 \leq x_1 \leq 1$ et $0 \leq x_2 \leq 1$. On peut donc considérer l'espace avec ses points ou bien le système de coordonnées. Il est clair que le système de coordonnées sous-jacent forme lui-même un sous ensemble d'un espace métrique. Cet espace métrique est noté Xc. Généralement, on ne fait pas la différence entre un point de X et de sa coordonnée dans Xc.

Théorème 1 :
Soient X un espace et $Xc \supset X$ un espace de coordonnées pour X.
Supposons que le changement de coordonnées est réalisé par la transformation $\Gamma : Xc \to Xc$, où Γ est inversible de X vers $\Gamma(X)$.
Notons encore x, les coordonnées du point $x \in X$ avant le changement de coordonnées et x' le point obtenu après le changement de coordonnées, i.e. $x' = \Gamma(x)$.

Soient finalement $f : X \to X$ une transformation de l'espace X, $x \to f(x)$ la formule de f exprimée dans les coordonnées de départ et $x' \to f'(x')$ la formule de f exprimée dans les nouvelles coordonnées.
Alors :

$$f(x) = (\Gamma^{-1} \circ f' \circ \Gamma) (x)$$
$$f'(x') = (\Gamma \circ f \circ \Gamma^{-1}) (x') \quad ^{*)}$$

*) la notation $f \circ g (x)$ est équivalente à $f(g(x))$

Définition 3 :
Soit f : X->X une transformation d'un espace métrique. Un point $x_f \in X$ tel que $f(x_f) = x_f$ est appelé un **point fixe** de la transformation.

En d'autre mots, un point fixe est un point qui reste inchangé lors d'une transformation.

Définition 4 :
Une transformation f : X->X d'un espace métrique (X,d) est dite **contractive** si il existe une constante $0 \leq s < 1$ telle que :

$$d(f(x),f(y)) \leq s\, d(x,y) \quad ,\forall\, x,y \in X.$$

Un tel nombre **s** est nommé **facteur de contractivité** de f.
On remarquera qu'une transformation contractive est continue. Il résulte en effet de la définition 4 que si $\{x_n\}$ converge vers x, alors $\{f(x_n)\}$ converge vers f(x).

Théorème 2 :
Soit f : X->X une transformation contractive d'un espace métrique complet (X,d). Alors f possède exactement **un** point fixe $x_f \in X$ et en plus, pour tout $x \in X$, la suite $\{ f^n(x), n=1,2... \}$ converge vers x_f. En d'autres mots :

$$\lim_{n\to\infty} f^n(x) = x_f \text{ pour tout } x \in X$$

où $f^0(x) = x$, $f^1(x) = f(x)$ et $f^{p+1}(x) = f(f^p(x))$ pour p = 1,2,...

Ce théorème est la base de la théorie des fractales, il va justifier dans le paragraphe suivant la notion d'attracteurs. Le théorème mérite d'être démontré pour bien suivre la construction des résultats :

*** montrons que $\{ f^n(x), n=0,1,2... \}$ est une suite de Cauchy :**

si m > n :

$d(f^n(x),f^m(x)) = d(f^n(x), f^n(f^{m-n}(x)))$, en posant que $(f^{m-n}(x)) = y$:
$= d(f^n(x), f^n(y))$ et en posant $f^{n-1}(x) = x'$
$f^{n-1}(y) = y'$
$= d(f(x'), f(y'))$ en appliquant la définition de contractivité :
$\leq s\, d(x',y')$ en appliquant n fois la définition de contactivité :
$\leq s^n d(x,y)$
$= s^n d(x, f^{m-n}(x))$

si m < n on trouve de la même façon :
$d(f^n(x),f^m(x)) \leq s^m\, d(f^{n-m}(x), x)$

ce qui donne ensemble :
$d(f^n(x),f^m(x)) \leq \min (s^m , s^n)\ d(f^{|n-m|}(x) , x)$

Ce qui reste à voir est que le dernier terme du membre de droite reste borné pour m et n croissants. Si on regarde $d(f^k(x) , x)$ pour un k quelconque (en appliquant k fois l'inégalité triangulaire) :

$$\begin{aligned} d(x , f^k(x)) &\leq d(x , f^1(x)) + d(f^1(x) , f^2(x)) + ... + d(f^{k-1}(x) , f^k(x)) \\ &\leq (1 + s + ... + s^{k-1})\ d(x , f(x)) \\ &= \sum_{i=1}^{k-1} s^i\ d(x , f(x)) \leq \sum_{i=1}^{\infty} s^i\ d(x , f(x)) \\ &= (1 - s)^{-1}\ d(x , f(x)) \end{aligned}$$

ce qui permet de prouver que $\{ f^n(x), n=1,2... \}$ est une suite de Cauchy car :

$$\begin{aligned} d(f^n(x),f^m(x)) &\leq \min (s^m , s^n)\ (1 - s)^{-1}\ d(x , f(x)) \\ &\leq \varepsilon \text{ lorsque m et n sont suffisamment grands} \end{aligned}$$

De la définiton d'une suite de Cauchy dans un espace métrique complet on sait que cette suite possède un point limite, noté par $x_f = \lim (f^n(x))$.

*** montrons que x_f est un point fixe pour f :**

$f(x_f) = f (\lim (f^n(x)))$ comme toute transformation contractive est continue :
$= \lim (f^{n+1}(x)) = x_f$

*** il reste à montrer que le point fixe est unique :**

Supposons qu'il existe deux point fixes x_f et y_f de f. Alors
$d(x_f , y_f) = d(f(x_f) , f(y_f))$ car ce sont des points fixes
$\leq s\ d(x_f , y_f)$ puisque f est contractive

Cela veut dire que $(1 - s)\ d(x_f , y_f) \leq 0$. Comme s est positif et plus petit que 1 et la distance est positive, on a que $d(x_f , y_f) = 0$, donc $x_f = y_f$.

1.4.3. Transformations contractives dans l'espace des fractales

Donnons ici un petit résumé de la situation :
On est dans un espace métrique (X,d) et on lui associe l'espace (H(X),h(d)) qui est l'espace des ensembles compacts non-vides avec la métrique de Hausdorff h(d). On a supposé jusqu' ici que les fractales sont des ensembles d'espaces métriques simples comme ($\mathbb{R}^2$,Euclidien). Si l'on définit maintenant une fractale, on la voit comme un point fixe d'une transformation de (H(X),h(d)) !

Elargissons d'abord la notion de contractivité sur H(X) (lemme 1), pour passer à la notion de contractivité d'un ensemble de transformations (lemme 2)

Lemme 1 :
Soit w : X->X une transformation contractive d'un espace métrique (X,d) avec facteur de contractivité s. Alors w : H(X) -> H(X) définie par :

$$w(B) = \{ w(x) : x \in B \} \text{ pour tout } B \in H(X)$$

est une transformation contractive de (H(X),h(d)) avec facteur de contractivité s.

Lemme 2 :
Soient (X,d) un espace métrique et $\{ w_n : n=1,2,3...N \}$ un ensemble de transformations contractives de (H(X),h(d)) de facteurs de contractivité s_i respectifs. Si W : H(X) -> H(X) est défini par :

$$W(B) = w_1(B) \cup w_2(B) \cup ... \cup w_N(B) \text{ pour tout } B \in H(X)$$

alors W est une transformation contractive avec facteur de contractivité

$$s = \max \{ s_i : i=1,2...N \}$$

Ces lemmes ne seront pas démontrés ici, car ces résultats sont intuitivement une dérivation logique du concept de base. Introduisons plutôt la notion de IFS et d'ATTRACTEUR qui découlent des lemmes:

Définition 1 :
Un **IFS (iterated function system)** est formé d'un espace métrique complet (X,d) et d'un ensemble de transformations contractives w_n : X->X, de facteurs respectifs de contractivité s_n. On le note par : $\{ X; w_n, n=1,2,3...N \}$ et son facteur de contractivité est $s = \max \{ s_n , n=1,2,...N \}$.

Voici le théorème résumant l'ensemble des résultats obtenus :

Théorème 1 :
Soit { X, w_n, n=1,2,...N } un IFS avec facteur de contractivité s. Alors la transformation W : H(X)->H(X) définie par :

$$W(B) = \bigcup_{n=1}^{N} w_n(B)$$

est une transformation contractive de l'espace métrique complet (H(X),h(d)) avec facteur de contractivité s, autrement dit :

$$h(W(B), W(C)) \leq s\, h(B, C)$$

pour tout B,C $\in$ H(X). Son point fixe unique, A $\in$ H(X) vérifie :

$$A = W(A) = \bigcup_{n=1}^{N} w_n(A)$$

et est obtenu par : $A = \lim_{n\to\infty} W^n(B)$ pour tout B $\in$ H(X).

Par définition, l'**attracteur** de l'IFS du théorème 1 est le point fixe de la transformation W.

2. Algorithmes

Deux algorithmes pour générer des images d'attracteurs d'un IFS sont proposés ici. Ce sont l'algorithme déterministe et l'algorithme-chaos (Random Iteration Algorithm).
L'algorithme déterministe est basé sur le calcul direct d'une séquence d'ensembles $\{ A_n = W^n(A) \}$ à partir d'un ensemble initial A_0. L'algorithme-chaos se base sur une suite de choix aléatoires de transformations dans un ensemble de transformations fixées à priori et sur la construction parallèle d'une suite de points convergent vers un attracteur.
Les IFS considérés sont de la forme $\{ \mathbb{R}^2 ; w_n , n=1,2,3...N \}$. Voici le code pour un attracteur qui est le triangle de Sierpinski:

w	a	b	c	d	e	f	p
1	0.5	0	0	0.5	1	1	0.33
2	0.5	0	0	0.5	1	50	0.33
3	0.5	0	0	0.5	50	50	0.34

Remarquons que les probabilités n'interviennent pas dans l'algorithme déterministe, mais ils jouent un rôle important pour l'algorithme-chaos, en particulier pour le coloriage des attracteurs.

2.1. L'algorithme déterministe

Soit $\{ X ; w_n , n=1,2,3...N \}$ un IFS. On choisit un ensemble compact de départ $A_0 \in \mathbb{R}^2$ et on calcule successivement $A_n = W^n(A)$ en se rapportant à :

$$A_{n+1} = \bigcup_{i=1}^{N} w_i (A_n) \text{ pour } n=1,2,3,...$$

En d'autres termes, une suite $\{ A_n , n=1,2,3...N \} \in H(X)$ sera construite. Par le théorème 1.4.2. , la suite converge vers l'attracteur de l'IFS dans la métrique de Hausdorff.

Le programme conceptuel suivant donne les étapes à suivre. L'algoritme exécutable a été implémenté en Pascal (Annexe A):

```
initialiser 2 bitmap        b1(dimx,dimy)
                            et b2(dimx,dimy)
initialiser les transfromations affines :
                    a,b,c,d,e,f(nt)
                    nt = nombre de transformations
initialiser l'ensemble de départ (carré,...) dans b1
```

```
pour n=1 à nombre_de_reproductions
      pour i=1 à dimx faire et
      pour j=1 à dimy faire :
            appliquer W à An pour former An+1 i.e.
            si b1(i,j) = 1 alors
                  b2( a(1)*i + b(1)*j + e(1) , c(1)*i + d(1)*j + f(1) ) = 1
                  b2( a(2)*i + b(2)*j + e(2) , c(2)*i + d(2)*j + f(2) ) = 1
                  ...
                  b2( a(nt)*i + b(nt)*j + e(nt) , c(nt)*i + d(nt)*j + f(nt) ) = 1
            fin si
      fin j
      fin i

      dessiner b2
      copier b2 dans b1
fin n
```

Voici les étapes successives générées par ALG_DET.PAS :

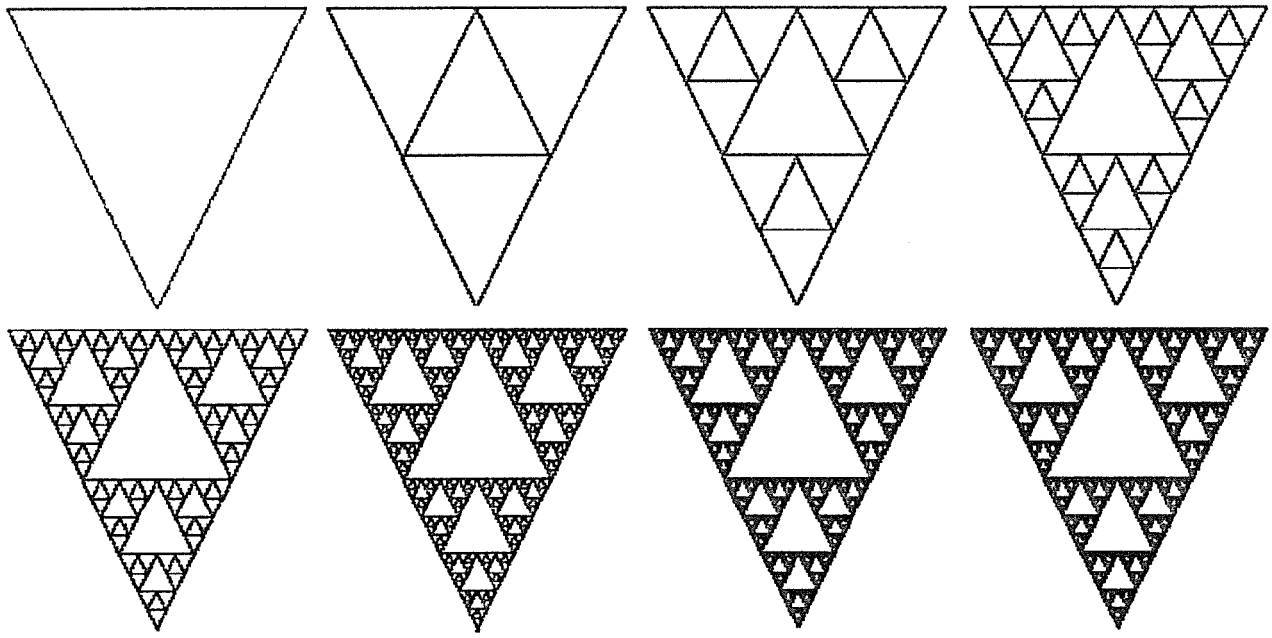
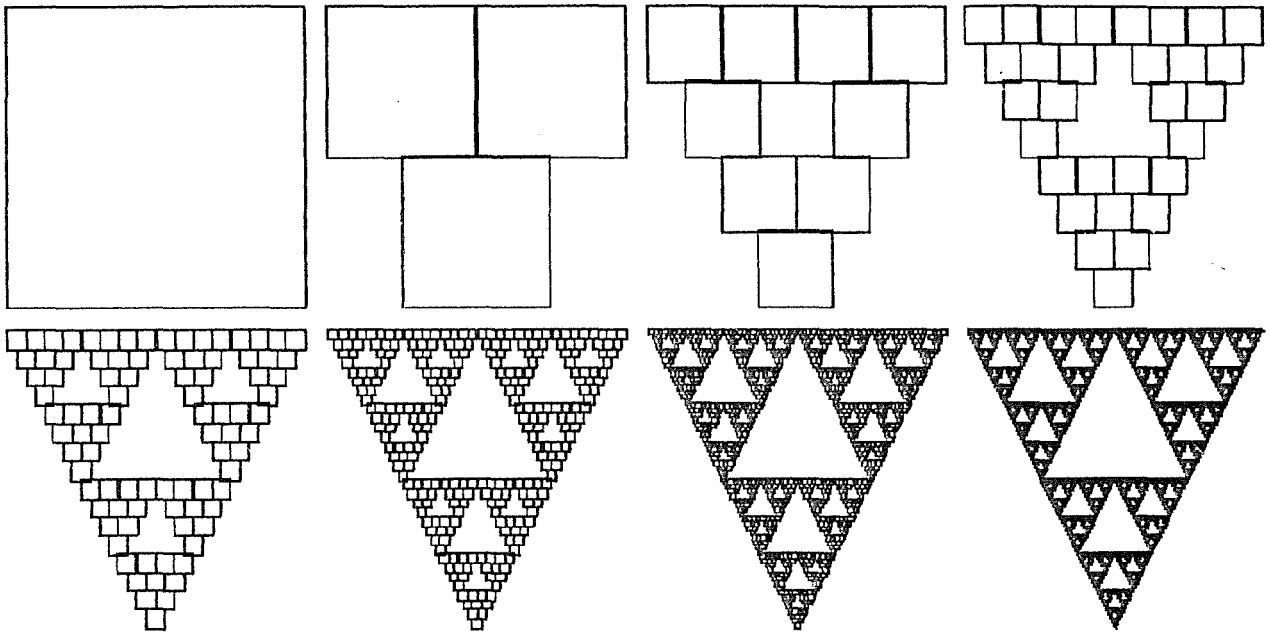


Figure 2.1. : Le programme commence par le tracé d'un triangle dans le bitmap de départ b1. Une observation intéressante (à première vue même inquiétante) est que cette figure de départ n'a auccune influence sur l'image finale du triangle de Sierpinski. On pourrait commencer avec une autre figure de départ pour aboutir au même résultat *(Figure 2.2.)* :



2.2. L'algorithme-chaos

Soient $\{ X ; w_n , n=1,2,3...N \}$ un IFS et des probabilités $p_i > 0$ associées aux w_i , i=1 à N, avec comme d'habitude $\sum_{i=1}^{N} p_i = 1$. On choisit $x_0 \in X$ et indépendamment $x_n \in \{ w_1(xn-1), w_2(xn-1),.. w_N(xn-1) \}$ pour n=1,2,3 ... numits où la probabilité de choisir $x_n = w_i(x_{n-1})$ est p_i.

On construit donc une suite $\{ x_n : n=1,2,... \} \in X$ qui converge vers l'attracteur de l'IFS. Voici la version théorique de l'algorithme-chaos (la version Pascal se trouve en annexe B) :

```
initialiser les transformations affines :
        a,b,c,d,e,f(nt)
        nt = nombre de transformations
        et leurs probabilités associées

choisir un point (x,y) de départ

pour n=1 à nombre_iter
    choisir une transfromation au hasard = tr (en respect des probabilités)
    déterminer le nouveau point :
        newx = a(tr)*x + b(tr)*y + e(tr)
        newy = c(tr)*x + d(tr)*y + f(tr)
    x = newx , y = newy
    dessiner le point (x,y)
fin n
```

Figure 2.3. : Le résultat de l'algorithme-chaos pour des nombres d'itération croissantes. Le 'point dansant' balaye au fur et a mesure l'image pour former l'attracteur de l'IFS.

2.3. La dépendance continue des fractales par rapport à leurs paramètres

Théorème 1 :
Soit (X,d) un espace métrique. Soit aussi { X ; w_n , n=1,2,3...N } un IFS de contractivité s. Si w_n dépend continument d'un paramètre p ∈ P, où P est un espace métrique compact, alors l'attracteur A(p) ∈ H(X) dépend continument de p ∈ P en respectant la métrique de Hausdorff.

En d'autres mots, des petites modifications des paramètres vont conduire à de petits changements de l'attracteur. Ce fait est important car il implique qu'on peut contrôler continument l'attracteur d'un IFS en ajustant les paramètres dans les transformations. De plus, le passage d'un attracteur à un autre se fait sans ruptures, ce qui permet de gérer facilement des animations, par exemple.

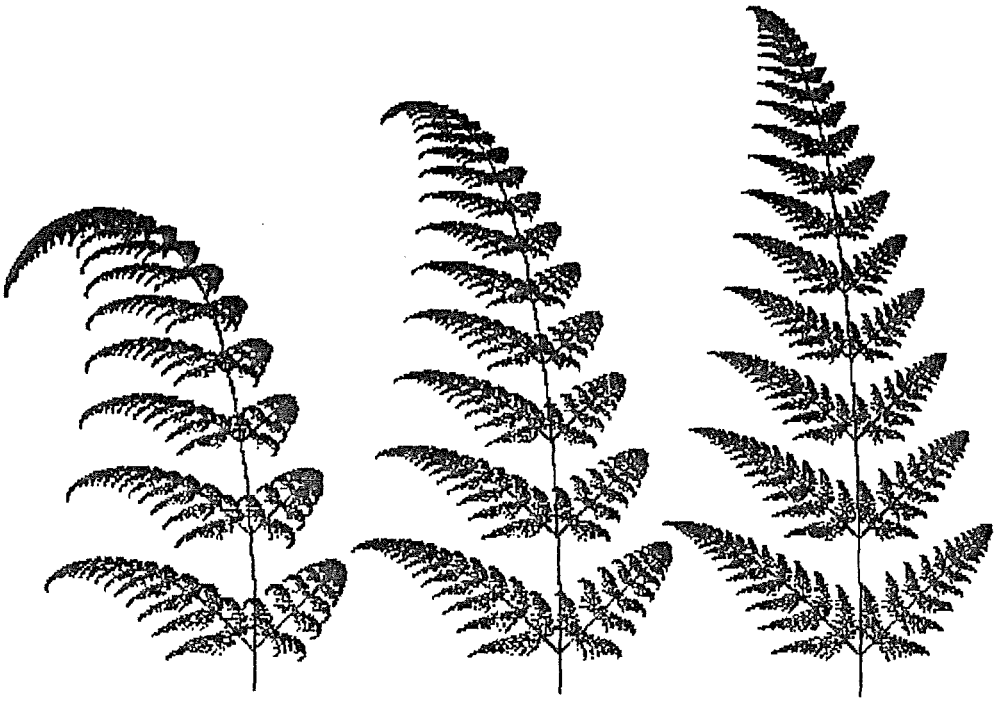


Figure 2.4. : Voici trois modifications d'une seule transformation d'un IFS.
(Code d'un farn en trois dimensions)

Mais cela veut dire aussi qu'on peut envisager le problème inverse : Etant donné un ensemble E, trouver l' IFS pour lequel E est l'attracteur. La réalisation se fait en ajustant les paramètres des transformations (programme SATFRACT) et à l'aide du théorème de collage :

Théorème de collage :
Soit (X,d) un espace métrique complet. Soit $L \in H(X)$ donné et soit $\varepsilon > 0$.
On choisit un IFS $\{ X ; w_n , n=1,2,3...N \}$ avec facteur de contractivité $0 \leq s < 1$ tel que :

$$h\left(L , \bigcup_{i=1}^{N} w_i (L) \right) < \varepsilon$$, où h(d) est la distance de Hausdorff. Alors

$$\mathbf{h (L , A) \leq \varepsilon / (1-s)}$$, où A est l'attracteur du IFS.

preuve :
Il faut montrer que si on considère une transformation contractive
F : H(X) -> H(X) avec son point fixe A de H(X) , alors pour tout autre point L de H(X) on a :

$$h(L , A) \leq d(L , F(L)) \; 1/(1 - s)$$

si l'on identifie F avec l'union des w_i on obtient le résultat annoncé par le théorème. Le travail consiste alors simplement à trouver les w_i qui recouvrent bien (i.e. à ε près) l'image à approximer.

On sait :

$h(L , A) = h(L , \lim F^n(L))$ car A est un point fixe de F

$= \lim (h(L , F^n(L)))$ car h est continue

$\leq \lim \sum_{m=1}^{n} h(F^{m-1}(L) , F^n(L))$ par l'inégalité triangulaire

$\leq \lim \; h(L , F(L)) \; (1 + s + ... + s^{n-1})$ voir preuve théorème 2 / Ch. 2.2.

$\leq (1 - s)^{-1} \; h(L , F(L))$ pour tout L de H(X)

3. Les META - IFS

3.1. Introduction

Lorsqu'on passe à l'implémentation des IFS par l'algorithme déterministe ou l'algorithme chaos, on se rend compte que la théorie fractale comme elle a été définie ici ne convient pas toujours à la modélisation souhaitée. Certains objets sont reproduits parfaitement par les IFS, l'exemple classique étant celui des fougères.

Mais si on veut représenter des objets de nature plus complexe, comme une branche d'un arbre qui contient des feuilles, il semble difficile de modéliser ceci par un IFS si l'on insiste sur le fait qu'une feuille est un concept indépendant de celui de branche. Une feuille devrait être modélisée par un autre ensemble de transformations que la branche qui la contient.

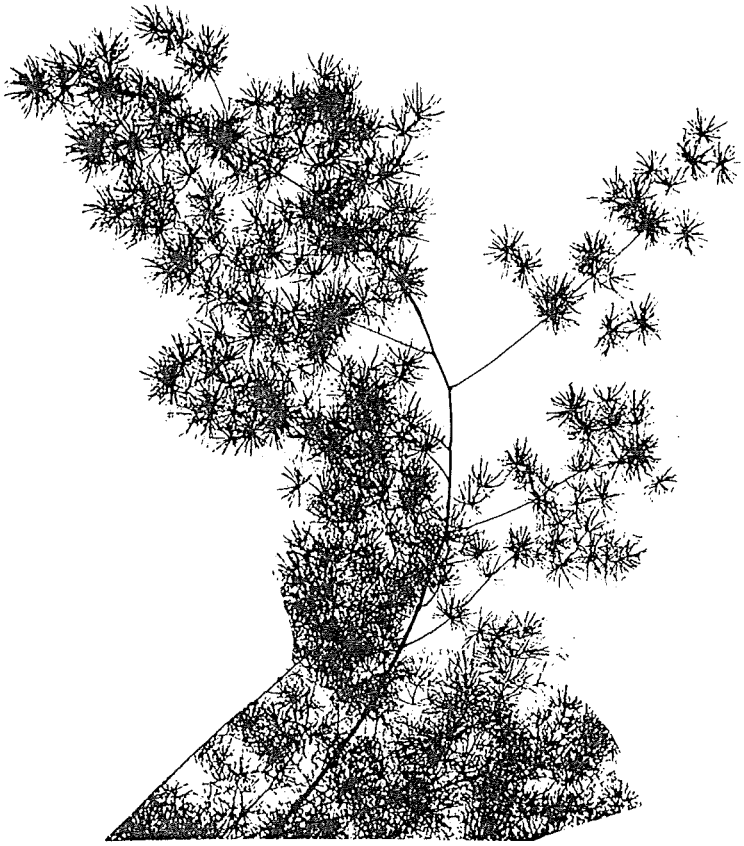


Figure 3.1. : En effet, il semble exister dans le monde réel une sorte d'hiérarchie entre différents objets. Ainsi, la croissance d'une feuille est contrainte par la croissance de la branche et le nombre de feuilles subsistantes dépend du nombre d'embranchements.

Ultérieurement, on pourra comparer l'image d'un arbre au modèle obtenu par un Méta-IFS.

En formalisant d'une autre manière encore, on pourrait penser à créer un IFS de type 'branche' généré par un ensemble de transformations qui déplace un autre objet (la feuille) qu'il contient suivant ses transformations. Dans cette perspective, on parlera plutôt de Méta-IFS et on dira en général qu'un 'Méta-IFS' est formé d'un ensemble de transformations et d'un ensemble d'IFS. Les

transformations du Méta-IFS agissent sur les attracteurs des IFS pour former l'attracteur du Méta-IFS.

Cette idée vaut la peine d'être développée plus formellement :

3.2. Construction des Méta-IFS

Définissons donc en premier lieu un **ensemble de m IFS** par :

$$\{ X , \{ w_n^1 , n=1,2,...N_1 \} , \{ w_n^2 , n=1,2,...N_2 \} , ... \{ w_n^m , n=1,2,...N_m \} \}$$

Les attracteurs correspondants sont donnés par :

$$A_i = \bigcup_{n=1}^{N_i} w_n^i (A_i) \quad \text{pour } 1 \le i \le m$$

Définition 1 :
Un **Méta-IFS** est formé d'un ensemble de transformations $\{ w^*_n , n=1,2,...N^* \}$ et d'un ensemble d'IFS. La notation explicite est donc de la forme suivante :

$$\boxed{\{ X , \{ w_n^1 , n=1,2,...N_1 \} , ... \{ w_n^m , n=1,2,...N_m \} , \{ w^*_n \quad n=1,... N^* \} \}}$$

Définition 2 :
A est un **attracteur dominé** s'il existe un $j \in \{ N_1 , N_2 , ..N_m \}$ tel que :

$$A = \bigcup_{n=1}^{N_j} w_n^j (A)$$

Dans la formalisation de W*, il faut tenir compte que cette transformation n'agit que sur un ensemble limité de points de H(X). Ce sont toujours des ensembles de (X,d) qui sont des transformations successives des attracteurs dominés.

Ainsi : $W^* : H(X) \dashrightarrow H(X)$

$$W^*(B) = \bigcup_{n=1}^{N^*} w^*_n (B)$$

pour tout B tel qu'il existe un i : $B = (W^*)^i (\bigcup_{n=1}^{m} A_n)$

et ceci pour tout ensemble $\{ A_1 , A_2 , ... A_m \}$

Cela implique que le théorème 1 du paragraphe 1.4.3. reste applicable mutatis mutandis car le résultat reste certainement valable sur un domaine plus restreint s'il l'est pour H(X) tout entier. On peut donc dire qu'il existe un point fixe unique $A \in H(X)$ qui vérifie :

$$A = W^*(A) = \bigcup_{n=1}^{N^*} w^*_n(A)$$

$$\text{où } A = \lim_{n \to \infty} (W^*)^n(B) \quad \text{pour tout } \{ A_1, A_2, \ldots A_m \} \text{ tq } B = \bigcup_{n=1}^{m} A_n$$

Définition 3 :
Un **Méta-Attracteur** est dès lors le point fixe unique du Méta-IFS

A première vue, ce résultat devrait choquer, le Méta-Attracteur est indépendant de l'Attracteur dominé ! D'autre part, ceci était prévisible, car on vient de montrer précédemment que l'attracteur d'un IFS est indépendant de l'ensemble de départ. Il importe donc peu pour la convergence, qu'il y ait un attracteur dominé comme structure initiale ou pas : Méta-Attracteur et Attracteur sont deux mêmes objets !

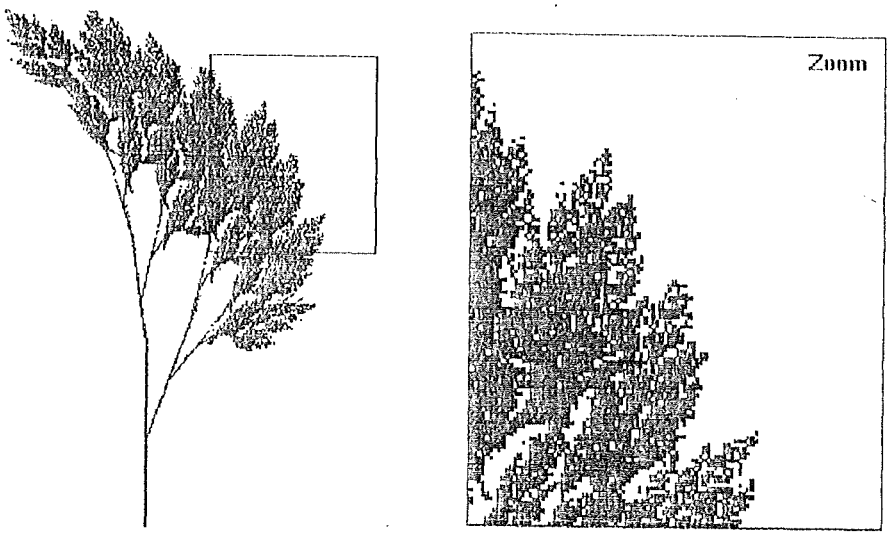
D'un point de vue pratique, on devrait observer que les attracteurs dominés vont être de plus en plus contractés pour être reduit à un point. Mais cela ne correspond pas aux idées précisées auparavant. Si l'on se rappelle l'exemple de la branche, il est clair qu'on exige qu'un nombre limité d'embranchements. A ce moment là, on ne représentera pas les Méta-Attracteurs, mais plutôt des reproductions successives par les transformations. Ce principe est comparable à celui de l'algorithme déterministe.

3.3. Algorithmes

3.3.1. Algorithme déterministe

Si on se base sur l'algorithme déterministe présenté au 2ème chapitre, il peut sembler que l'implémentation de ce concept revient à considérer l'algorithme déterministe qui agit sur un tableau contenant comme ensemble de départ non seulement un carré, mais une ou plusieures images d'attracteurs d'IFS. Pratiquement, ce n'est pas ce qu'on veut obtenir puisque du fait que les attracteurs dominés sont mémorisés de façon fixe, on perd complètement la propriété que l'image contenue dans ce tableau est un attracteur.

Figure 3.2. : Un Zoom par exemple conduit ainsi à une perte de précision :



Au contraire, l'algorithme associé aux Méta-IFS doit générer toutes les combinaisons de transformations possibles pour agir comme un changement de coordonnées sur les IFS y contenus et demander, pour toute configuration obtenue, à l'IFS de se dessiner sur le domaine transformé. On établira donc les attracteurs dominés sur différents domaines qui sont construits selon la structure du Méta-IFS.

Cette implémentation nécessite une technique beaucoup plus subtile :

- ☞ Implémentation de l'algorithme chaos (ou déterministe) avec un changement de coordonnées pour la production des IFS.
- ☞ Modification de l'algorithme déterministe en forme récursive, afin de générer les changements de domaine aux IFS de la part du Méta-IFS.
- ☞ Programmation en langage Objet pour faciliter l'interaction entre Méta-IFS et IFS

Dérivons l'algorithme à partir du concept mathématique :
Les itérées successives de la transformation contractive forment donc une suite de Cauchy et le programme doit être capable de les reproduire. Ne considérons pour des raisons de lisibilité qu'un seul attracteur dominé.

Voici l'expression d'une étape **i** quelconque pour un seul attracteur dominé **j** :

$$W^{*i}(A_j) = \left(\bigcup_{n=1}^{N^*} w^*_n \right)^i (A_j)$$

Présenté sous cette forme, la formule ne permet pas encore de dessiner un Méta-IFS. Exprimons le fait que A_j est un attracteur :

$$W^{*i}(A_j) = \left(\bigcup_{n=1}^{N^*} w^*_n \right)^i \left(\lim_{q \to \infty} \bigcup_{m=1}^{N_j} (w^j_m)^q (B) \right) \quad \forall B \in H(X)$$

Pour enfin pouvoir se ramener au dessin d'un point, il suffit de remarquer que $w^j_n(B) = \{ w^j_m(x) \ , \ x \in B \}$ pour tout B de H(X). Ceci permet de voir le problème sous la forme :

1. Appliquer pour tout attracteur dominé j ⇨ 2. Appliquer récursivement i fois toutes les transformations du Méta-IFS à ⇨ 3. Appliquer une infinité de fois une transformation choisie au hasard parmi 1..Nj à ⇨ 4. Dessiner un point.

Voici les différents algorithmes conceptuels nécessaires aux différentes étapes :

1. Dessiner le Méta-IFS pour tout attracteur dominé :

```
initialiser le nombre de reproductions : p
initialiser les transfromations affines du Méta-IFS :
      W* : l'ensemble des transformations du Méta-IFS
      N* : le nombre de transformations
initialiser les attracteurs dominés : W1 à Wm et N1 à Nj

pour  j = 1 à m faire :
      choisir Wj
      exécuter Dessiner-Réc ( p , Wj )
```

2. Dessiner le Méta-IFS récursivement pour un attracteur :

Appel initial :

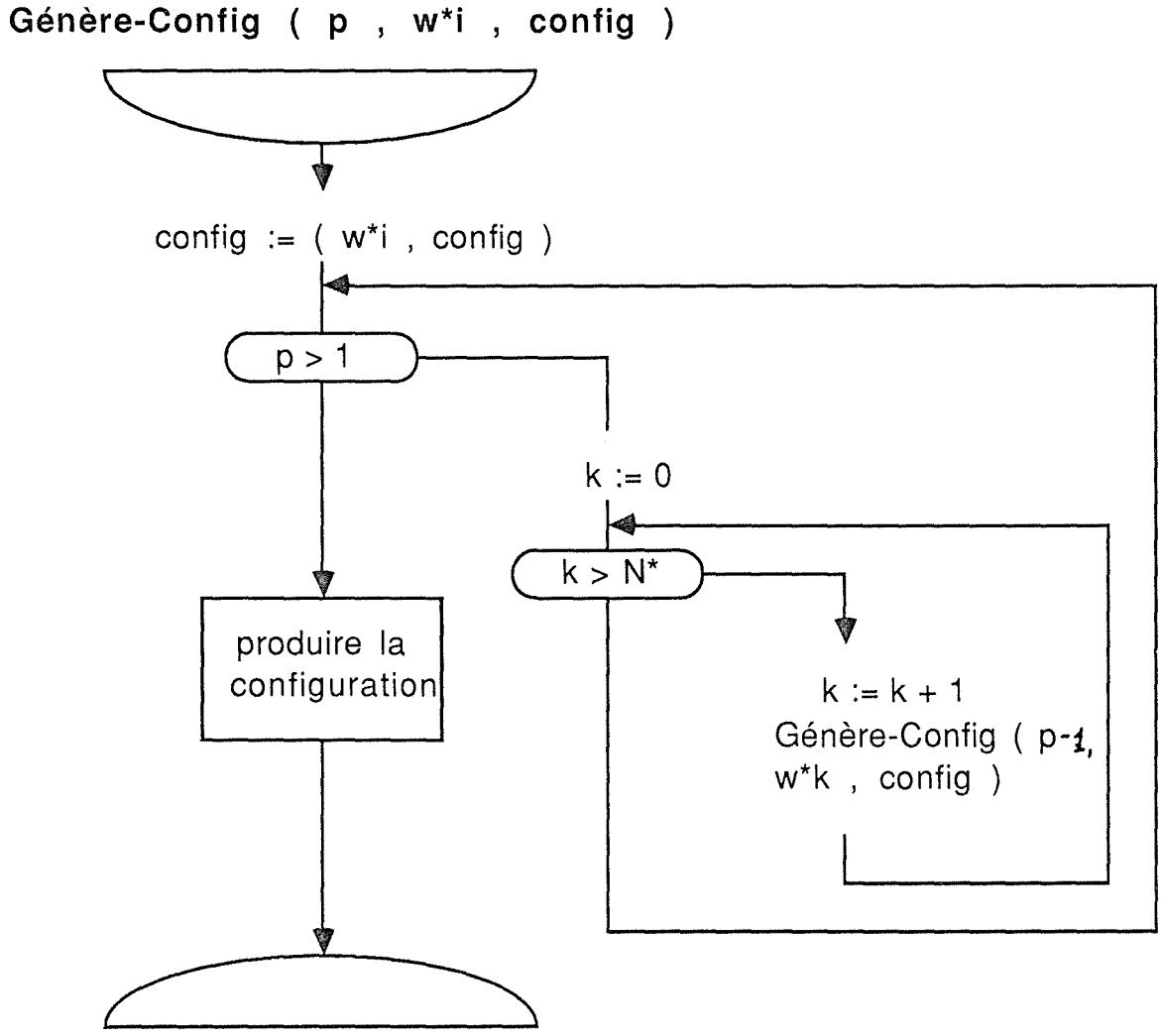
procédure : Dessiner-Réc (p , Wj)

initialiser la configuration à produire : config = (Wj)

pour i = 1 à N* faire :
exécuter Génère-Config (p , w^*_i , config)

Corps :

Comme mentionné, la génération des combinaisons se fait de manière récursive. Il semble être préférable de représenter cet algorithme sous forme d'organigramme :



Si l'on considère un exemple numérique :

soit la configuration : $(w^*_1 (w^*_4 (w^*_1 (w^*_2 (w^*_1 (\{ w^1_1, w^1_2, w^1_3 \}))))))$

ce qui correspond aux différentes valeurs des variables :

$j = 1$	$p = 5$
$Nj = 3$	$i_1 = 1$
$N^* = 4$	$i_2 = 2$
	$i_3 = 1$
	$i_4 = 4$
	$i_5 = 1$

On peut trouver d'ici, qu'une façon de réaliser concrètement l'algorithme est de faire subir des changements de repère successifs à l'IFS $\{ w^1_1, w^1_2, w^1_3 \}$ par les transformations w^*_1, w^*_2, w^*_1, w^*_4, puis w^*_1 (config = (w^*_i, config)). L'IFS est alors dessiné (produire la configuration) sur le domaine obtenu.

Donnons ici une image fractale obtenue par un Méta-IFS, généré par l'algorithme déterminsite, l'image réelle (figure 3.3.b.) est un Pin parasol.

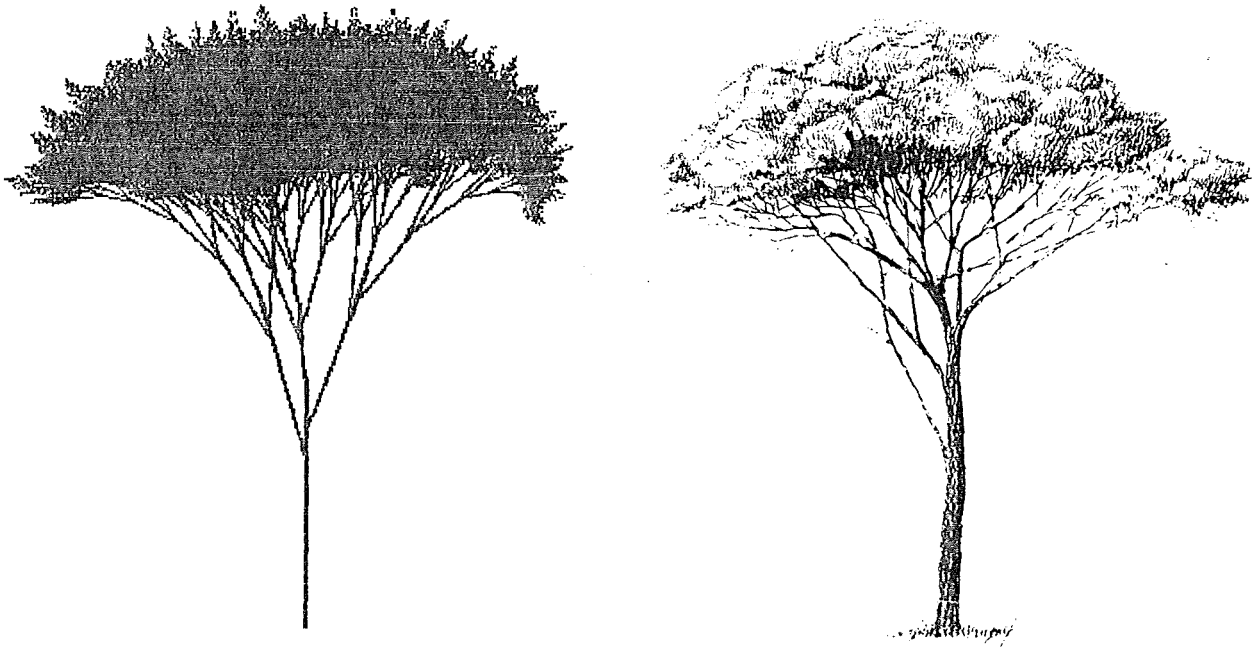


Figure 3.3. (a) Les transformations du Méta-IFS forment la structure de l'arbre et les attracteurs dominés sont deux feuilles. (b) Image réelle d'un '*Pinus Pinea L.* '

3.3.2. Algorithme-chaos

Si l'on considère les deux algorithmes pour la génération des IFS, on constate que l'algorithme déterministe établit, à partir d'une image initiale, un carré par exemple, les itérées successives de la suite convergente vers l'attracteur, alors que l'algorithme chaos dessine immédiatement l'attracteur.

On avait déja mentionné qu'on considère uniquement des itérées des Méta-IFS et non l'attracteur de celui-ci. Il semble donc que le principe chaos ne convienne pas à la construction des Méta-IFS ...

Cependant, il y a moyen d'implémenter cette idée avec un procédé de dessin surprenant : l'algorithme présenté ci-dessous semble "mélanger" les deux ensembles de transformations; les points appartenantsà la structure du Méta-IFS semblent apparaitre en même temps que ceux des IFS dominés et, en plus, TOUS les IFS se déssinent simultanément.

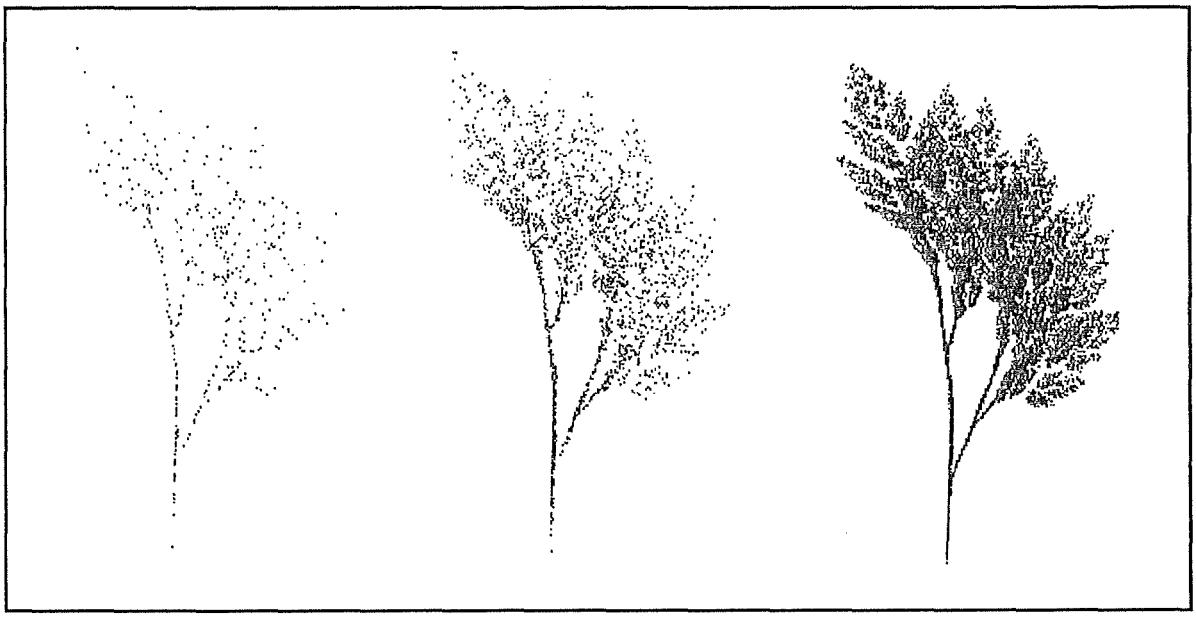


Figure 3.4 : Le principe correspond miraculeusement à l'idée chaos : l'image (l'itérée de la suite) se produit par un point dansant qui peut changer de couleur en passant d'un attracteur dominé à un autre.

Soient donc $\{ X , \{ w^1_n , n=1,2,...N_1 \} , ... \{ w^m_n , n=1,2,...N_m \} , w^*_n \quad n=1,... N^* \}$ le

Méta-IFS et $p^*_i \quad i=1,... N^*$ les probabilités associées aux transformations w^*_i .
L'efficacité du principe chaos consiste dans sa simplicité :

```
initialiser le nombre de reproductions : npro
déterminer le nombre d'itérations
initialiser les transfromations affines du Méta-IFS :
      W* : l'ensemble des transformations du Méta-IFS
      N* : le nombre de transformations
initialiser les attracteurs dominés : W1 à Wm et N1 à Nj

pour  j = 1 à nombre_itérations faire :
      choisir un IFS-dominé, Wj
      demander à l'IFS de générer un point au hasard
            résultat : (ptx,pty) et sa couleur coul

      pour i = 1 à npro
            choisir une transformation numéro tr du Méta-IFS (de W*) au hasard
            déterminer le nouveau point :
                  newx = a(tr)*ptx + b(tr)*pty + e(tr)
                  newy = c(tr)*ptx + d(tr)*pty + f(tr)
            ptx = newx, pty = newy
            dessiner le point (ptx,pty) avec la couleur coul
      fin i

fin j
```

Ce qui est tout à fait frappant dans ce principe est que l'algorithme chaos présenté au point 2.2 ne peut générer que le point fixe de la suite lorsqu'on veut dessiner un simple IFS, alors qu'il est devenu possible, pour les Méta-IFS, de dessiner une étape quelconque de cette suite. Le lecteur devrait en savoir la raison.

Remarquons ici que le dessin établi par l'algorithme chaos n'est pas influencé par l'ordre dans lequel on a inséré les IFS-dominés alors que l'algorithme déterministe génère tous les copiages du premier IFS-dominé avant de passer au second. De plus, il est préférable de ne pas colorer les IFS-dominés s'ils se superposent dans l'image finale et qu'on veut utiliser le principe chaos, car le point dansant passant aux différents IFS peut produire plusieurs changements de couleur successifs pour un même pixel de l'image finale et produire ainsi des interférences. Il est alors préférable d'employer l'algorithme déterministe.

La réalisation concrète nécessite une implémentation dans laquelle tout IFS-dominé possède un point dansant et que, lorsque le Méta-IFS le lui demande, il soit capable d'appliquer une transformation au hasard à ce point. Le Méta-IFS prend alors ce point comme point de départ et y applique ses transformations.

Dans cette formalisation, on voit déjà qu'il est absolument nécessaire de posséder un moyen informatique qui permet de parler d'objets qui détiennent des données et des procédures ou méthodes qui permettent de travailler dessus. Les objets doivent être capables de s'envoyer mutuellement des messages.

Mais ceci correspond justement aux principes de base des langages à objets :

4. Mesures sur les fractales

L'algorithme-chaos introduit précédemment est un moyen de calculer l'attracteur d'un IFS dans $\mathbb{R}^2$. Pour ce faire, il nécessite un ensemble de probabilités associées aux IFS.

4.1. Le principe

Définition 1 :
Un **IFS probabiliste** est un IFS $\{ X, w_n, n=1,2,...N \}$ auquel est associé un ensemble de nombres $\{ p_n, n=1,2,...N \}$ tel que :
$p_1 + p_2 + p_3 + ...p_N = 1$ et $p_i > 0$, $i=1,2,...N$
La probabilité p_i est associée à la transformation w_i. La notation complète d'un IFS probabiliste est alors

$$\{ X, w_n, n=1,2,...N ; p_1, p_2, p_3 ..., p_N \}$$

Considérons l'IFS donné par la table suivante :

w	a	b	c	d	e	f	p
1	0.5	0	0	0.5	1	1	0.1
2	0.5	0	0	0.5	50	1	0.2
3	0.5	0	0	0.5	1	50	0.3
4	0.5	0	0	0.5	50	50	0.4

L'attracteur est un carré. Si le nombre d'itérations est suffisamment grand, l'image résultante par l'algorithme chaos sera un carré rempli. En d'autres mots, tous les points du carré ont été visités par le point dansant de l'algorithme-chaos. Comme les domaines des transformations ne se coupent pas, on peut prédire qu'après 10.000 itérations :

- le nombre de points calculés par w_1 = 1000,
- le nombre de points calculés par w_2 = 2000,
- le nombre de points calculés par w_3 = 3000,
- le nombre de points calculés par w_4 = 4000.

Au fur et à mesure, le carré se remplit et devient dense. Mais ce qu'on observe au début est que les points 'tombent' sur l'image avec une fréquence différente selon les endroits. Cela suggère une question intéressante : Peut-on dire qu'un IFS probabiliste engendre un attracteur avec une densité ou une texture unique ? Malheureusement on perd la vue lorsque le nombre d'itérations croît.

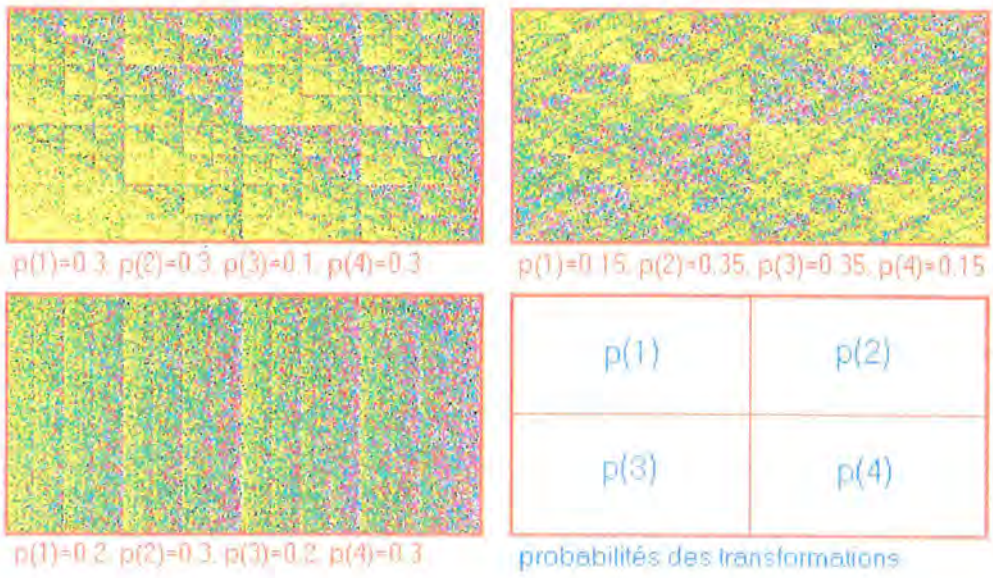


Figure 4.1. : Les mesures vont résoudre ce problème, car, en général, une mesure peut indiquer des distributions de "poids" dans les espaces métriques et on obtiendra pour le carré de prodigieuses textures comme celle ci-dessus.

La théorie des mesures est un domaine qui est trop complexe pour être développé ici, on se limitera plutôt à une compréhension intuitive. En considérant l'exemple de la table précédente, on dirait que w_4 a plus de "poids" que w_1.

Décrivons une **mesure sur l'attracteur** A d'un IFS probabiliste donné dans un espace métrique complet (X,d) :
Une mesure μ donne un poids aux ensembles de X. Ainsi, on pourrait dire que $\mu(A)=1$, $\mu(\text{vide})=0$ et $\mu(X)=1$, ce qui implique que le poids de l'espace entier est égal au poids de l'attracteur d'un IFS, oubien encore que le poids est localisé sur l'attracteur.

Si l'on prend une boule fermée B de l'espace X, on peut calculer $\mu(B)$ le poids de cette boule par :

Appliquer l'algorithme-chaos à l'IFS probabiliste, pour produire une séquence de points $\{z_n\}$.

Soit : N(B,n) = nombre de points dans $\{z_1,...z_n\} \in B$, alors :

$$\mu(B) = \lim_{n\to\infty} \{ N(B,n)/(n+1) \}$$

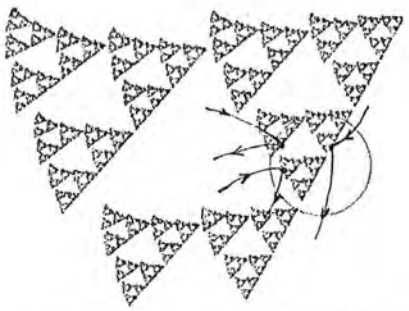


Figure 4.2. : Le poids de la boule B est donc la proportion des points que l'algorithme chaos produit dans B au moyen de l'IFS probabilisé utilisé.

Il serait sans doute trop complexe de donner une définition détaillée d'un sous-ensemble de Borel de X. En gros, on peut dire que les **sous-ensembles de Borel** sont des ensembles qui possèdent un poids et qui incluent les sous-ensembles compacts non-vides de X. Plus formellement et avec plus de précautions, le théorème d'**ELTON** nous renseigne que :

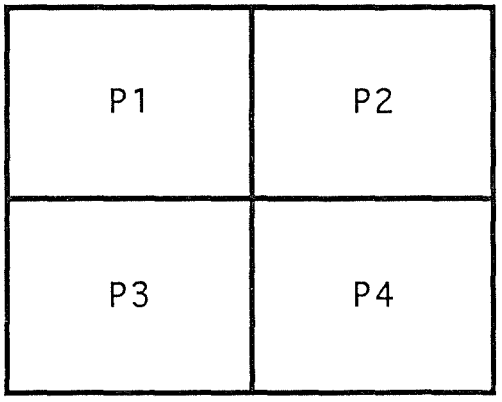
> Théorème 1 :
>
> Soit B un sous-ensemble de Borel de X et soit μ(frontière de B) = 0. Soit aussi N(B,n) = nombre de points dans $\{ x_0, x_1 ... x_n \} \in B$, alors avec probabilité 1, on a :
>
> $$\mu(B) = \lim_{n \to \infty} \{ N(B,n)/(n+1) \},$$
>
> pour tout point de départ x_0.

Ce qui veut dire que le poids de B est la proportion de points qui ont été produits par l'algorithme et qui sont tombés dans B.

Examinons l'exemple précédent et considérons les 4 transformations comme des copiages du carré dans son quadrant supérieur gauche, droite, inférieur gauche et droite. Un comptage donne 1003 points tombant dans le premier, 1994 dans le second, 3005 dans le troisième et 3998 dans le dernier quadrant. Prenons les mesures de ces quatres boules qui sont des rectangles :

u(w1(carré)) = 1003/10000 = 0.1
u(w2(carré)) = 1994/10000 = 0.2
u(w3(carré)) = 3005/10000 = 0.3
u(w4(carré)) = 3998/10000 = 0.4,

ce sont les probabilités des transformations (voir aussi *figure 4.3.a.*).



P1P1	P1P2	P2P1	P2P2
P1P3	P1P4	P2P3	P2P4
P3P1	P3P2	P4P1	P4P2
P3P3	P3P4	P4P3	P4P4

Figure 4.3.a et b. : Si on découpe encore plus finement (*figure 4.3.b.*) on obtient le produit des probabilités des transformations qui mènent à cette boule, car le poids de la cellule précédente est distribué parmi les cellules plus fines. La distribution du poids peut donc être déterminée dans des échelles de plus en plus fines.

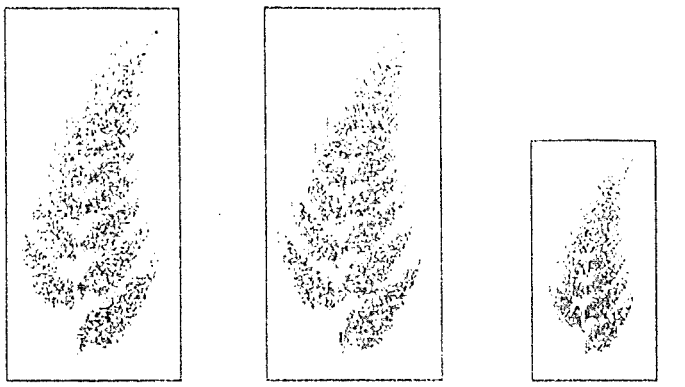


Figure 4.4.a.b. et c.

Concrètement, on peut donc continuer à raffiner pour arriver à un PIXEL de l'écran. Il faut simplement savoir combien de points de l'algorithme arrivent dans la boule, i.e. combien de fois on imprime le même pixel. Cela peut se réaliser par exemple par une matrice d'entiers (compt(x,y)) dont chaque élément est un compteur pour un pixel de l'éran. Par le théorème d'Elton, si numits est suffisamment grand, cela devrait être une bonne approximation de la mesure du pixel.
En conséquence, on va associer des couleurs aux différentes valeurs des compteurs. Si on veut représenter une figure sur un fond noir, il est logique d'attacher des couleurs d'intensités croissantes (du gris foncé jusqu'au blanc clair, par exemple) pour des valeurs du compteur croissantes. La couleur peut être obtenue comme suit :

couleur(i,j) = facteur * compt(i,j)/numits (1)

Ci-dessus l'image résultante qui montre que la mesure est invariante par rapport au nombre d'itérations, pourvu que ce nombre est suffisamment élevée (*Figure 4.4.a. et b*).

Un Zoom (magnitude) sur une fractale est un multiplicateur pour les coordonnées d'un point (x,y). Considérons l'exemple :

Si on double la taille de l'attracteur (magnitude=2), la résolution d' un pixel est augmentée d'un facteur 4, le nombre de points sera donc réparti sur 4 pixels au lieu d'un seul. Pour que les couleurs de l'image restent stables, il faut donc multiplier le membre de droite de l'équation (1) par 4, en général donc par le carré du multiplicateur :

couleur(i,j) = facteur * compt(i,j)/numits * mag^2

4.2. Algorithme

Voici l'algorithme conceptuel :

```
initialiser les transformations affines :
                a,b,c,d,e,f(nt),
                nt = nombre de transformations
                et leurs probabilités associées

initialiser le compteur : compt(dimx,dimy)

initialiser     mag (le zoom)
                numits (nombre d'itérations)
                facteur ('densité' des couleurs)

choisir un point de départ

pour n=1 à numits
    choisir une transformation au hasard = tr (en respect des probabilités)
    déterminer le nouveau point :
        newx = a(tr)*x + b(tr)*y + e(tr)
        newy = c(tr)*x + d(tr)*y + f(tr)
    x = newx , y = newy
    i = mag*x , j = mag*y
    si (i,j) ∈ (1..dimx,1..dimy) alors
        compt(i,j) = compt(i,j) + 1
    fin si
fin n

pour n1=1 à dimx
    pour n2 = 1 à dimy
        si compt(n1,n2)<>0 alors
            coul = compt(n1,n2) * facteur * mag^2 / numits
            dessiner le point (n1,n2) de couleur coul
        fin si
    fin n2
fin n1
```

Si on modifie les paramètres mag, p() et numits, le programme permet de vérifier les propriétés suivantes (voir aussi en annexe C) :

a) Une fois que les couleurs et le facteur ont été déterminés, l'image reste stable par rapport au nombre d'itérations, pourvu que le nombre d'itérations soit suffisamment grand (supérieur à 15000 pour la fenêtre donnée).

b) Les images dépendent continument du code IFS, y inclus les probabilités, ce qui permet de contrôler les images interactivement (voir aussi *Figure 4.5.*).

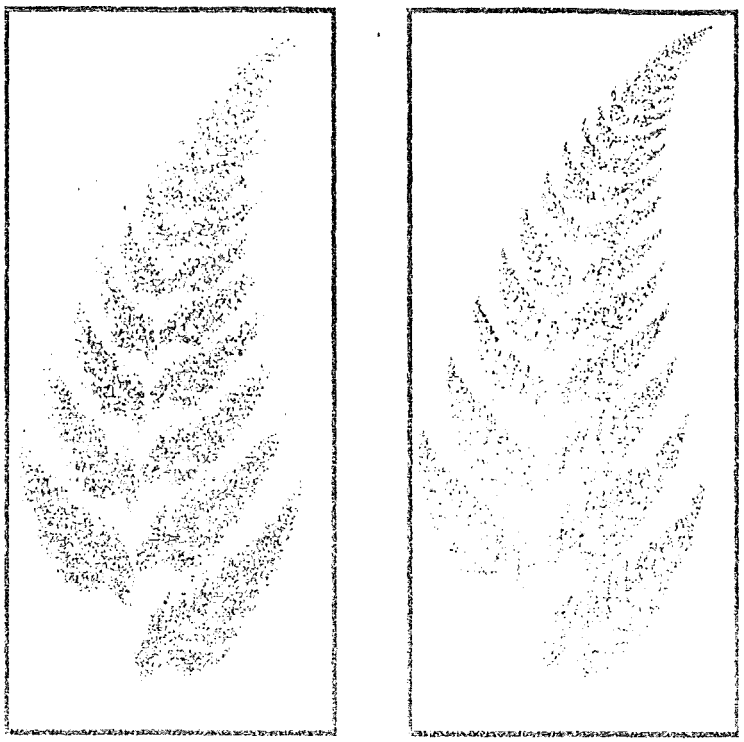


Figure 4.5. : Voici le résultat du coloriage d'un farn. L'ensemble des couleurs n'a pas été changé. Les seuls paramètres changés sont les probabilités des transformations.

c) Les images varient avec la taille du zoom choisi sans perdre leur précision si le nombre d'itérations est suffisamment grand, mais les couleurs restent inchangées.

5. Le concept OBJET

5.1. Introduction

L'"approche objet" se caractérise par l'utilisation de briques de base, ou éléments fondamentaux, qui représentent des abstractions d'objets du monde réel et qui se combinent pour définir des entités de plus haut niveau. Dans le jargon informatique, les briques de base sont aussi appelées objets.

La notion d'objet néanmoins est utilisée dans deux contextes différents : dans la description d'interfaces et en lien avec la programmation orientée objet.

Une interface orientée objet est un environnement qui permet la manipulation d'objets en utilisant des icones et une souris. Le concept 'objet" est à prendre au sens intuitif du terme, il fait référence à des composants de l'information contenus dans un interface-écran. L'utilisateur voit un objet comme une unité qui peut être manipulée par des actions définies. Par exemple, cette unité-objet peut être sélectionnée par la souris, déplacée ou effacée. Comme exemples de ce genre d'objets on peut citer des icones, des menus qui peuvent être sélectionnés ou bien des fenêtres qui peuvent être ouvertes, fermées, déplacées, réduites, etc. Remarquons que l'implémentation d'interfaces orientés-objet ne nécessite pas un langage objet.

On pourrait croire que cette idée est loin d'être une nouvelle invention, puisque qu'elle a été la base essentielle du Design par ordinateur (programmes CAD). L'aspect révolutionnaire de l'approche "interface objet" est que ce genre d'interface a remplacé presque entièrement les interfaces orientés commande comme environnement d'alternative (exemple : le windows sous DOS).

Par **programmation orientée objet**, on entend les langages de programmation orientés vers la manipulation d'objets. Les langages objets sont généralement dérivés d'un langage 'conventionnel' et il est théoriquement toujours possible de programmer sans nécessairement passer par la notion d'objet. Néanmoins la notion d'objet offre de nouvelles capacités qui définissent finalement une nouvelle philosophie en programmation.

5.2. La programmation orientée objet

5.2.1. Les Classes

Une classe peut être vue comme la description d'une famille d'objets ayant une même structure et un même comportement. Elle regroupe un

ensemble de données et un ensemble de fonctions. Chaque classe contient donc :

- une composante statique : les données, constituées de champs nommés, qui possèdent une valeur. Les champs caractérisent l'état des objets pendant l'exécution du programme.

- une composante dynamique : les procédures ou fonctions, appelées méthodes, qui représentent le comportement commun des objets appartenant à la classe. Les méthodes manipulent les champs des objets et caractérisent les actions qui peuvent être effectuées par les objets (ne pas lire 'sur les objets' !).

En prenant l'exemple de la gestion d'un magasin, on peut illustrer cette nouvelle définition : Une première classe d'objets, ARTICLE, décrit un article du stock. La classe définit 3 champs et quatre méthodes.

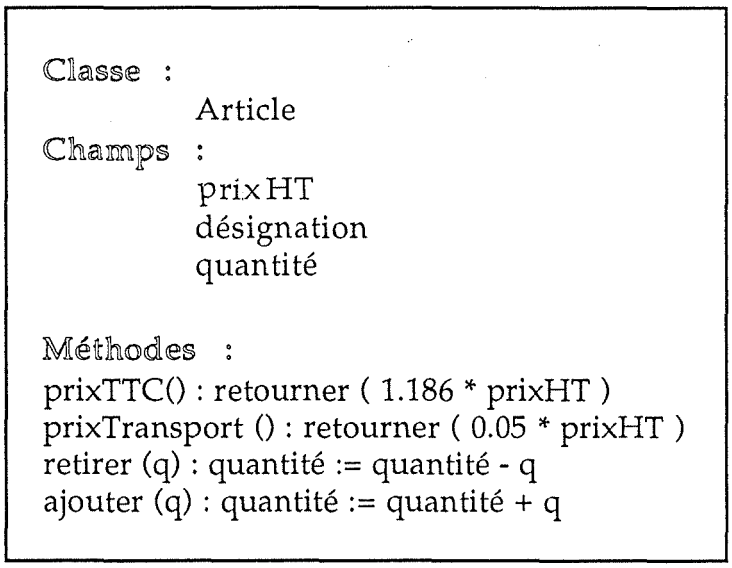


Figure 5.1. : Définition d'une classe

Comme on le voit, chaque méthode est définie par un nom qui est le **sélecteur** de la méthode. L'ensemble des couples sélecteur-méthode forme le **dictionnaire des méthodes**. Il est important de voir que le sélecteur nomme une méthode et que celle-ci se présente comme un programme qui prend la forme d'une procédure ou d'une fonction.

5.2.2. Les instances

Une **instance d'une classe** est un objet particulier qui est créé en respectant les plans de construction donnés par sa classe. La classe est donc une sorte de pochoir pour la création d'instances; elle permet de reproduire autant d'exemplaires que nécessaire. (On parle aussi de moulage)
Il faut préciser que ce moulage est partiel, car il est inutile de dupliquer l'ensemble des méthodes - les méthodes sont donc conservées par la classe.

Il convient donc de se poser le problème du partage efficace de connaissances. La classe doit être considérée comme un réservoir de connaissances à partir duquel il est possible de définir d'autres classes plus spécifiques complétant les connaissances de leur mère (=la superclasse). Les connaissances les plus générales sont ainsi mises en commun dans les classes qui sont ensuite spécialisées en sous-classes successsives, contenant des connaissances de plus en plus spécifiques. La spécialisation d'une classe peut être réalisée selon deux techniques :

1. L'enrichissement : la sous-classe est dotée de nouvelles variables ou de nouvelles méthodes représentant les caractéristiques propres au sous-ensemble d'objets ainsi décrit. Le problème de la chemise est ainsi réglé, on va associer à la classe Article une sous-classe Chemise aves les variables supplémentaires qui permettent de définir complètement les chemises. (figure 5.3.) Par héritage, la classe Chemise contient 5 variables d'instance, désignation, prixHT, quantité, taille, type_de_manche, et 4 méthodes.

2. La substitution : elle permet de donner une nouvelle définition à une méthode héritée, lorsque celle-ci est inadéquate pour l'ensemble des objets décrits par la classe. Par exemple, en créant une sous-classe pour les articles de luxe, il devient possible de définir une nouvelle méthode pour le calcul du prixTTC (figure 5.3.). La sous-classe Article_de_luxe contient les mêmes informations que la classe Article mais la méthode prixTTC masque celle qui est héritée.

Classe :	Classe :
Chemise	Article_de_luxe
Superclasse :	Superclasse :
Article	Article
Variables d'instance :	Variables d'instance :
taille	
coloris	
Méthodes :	Méthodes :
	prixTTC(): retourne(1.33*prixHT)

Figure 5.3. : Définition des sous-classes

La **relation d'héritage** lie une classe à une superclasse. La représentation de cette relation forme le graphe d'héritage (voir figure 5.4.).

Remarquons que la classe la plus générale est appelée Objet. Cette classe est prédéfinie et elle détient le comportement commun à tous les objets : comment imprimer un objet, comment trouver la classe d'appartenance d'un objet, ...

De plus, les instances d'une même classe possèdent toutes les mêmes champs (avec des valeurs différentes bien entendu !) , ce qui a pour conséquence que la liste des champs reste détenue par la classe, tandis que les instances possèdent les valeurs.

L'ensemble des couples associant une variable d'instance à sa valeur forme le dictionnaire des variables, qui est donc divisé en deux parties. L'une contient les noms des variables, est détenue par la classe et constitue le modèle utilisé pour créer les instances par moulage. L'autre contient les valeurs appartenant à chaque instance.

5.2.3. L'envoi de messages

Un objet est vu comme une entité indépendante dont la structure est connue de lui seul, et ne peut pas, en principe, agir directement sur un autre objet. Il doit obligatoirement utiliser une des méthodes appartenent à l'interface de cet autre objet et lui envoyer un message de demande d'exécution de la méthode en question. Le message doit être vu comme une **requête que l'objet appelé doit satisfaire.**

Un message entre objets devrait avoir la forme :

send(objet-receveur, sélecteur-méthode, arguments)

ou bien pour l'exemple :
send(x,"retirer",5) ce qui veut dire que pour l'instance X de la classe Article on retirera 5 éléments.

Lorsque la méthode passe un résultat, celui-ci est retourné à l'expéditeur du message. On dit qu'il y a transmission avec retour :

Pour l'exemple : prix <- send(y, "prixTTC")

5.2.4. L'héritage

Il est clair que, pour le moment, la gestion de stock suggérée par les exemples précédents n'est ni très pratique, ni économique, pour implanter des spécificités propres à chacune des catégories. En effet, la classe Article devrait alors regrouper l'ensemble des variables et méthodes nécessaires pour caractériser et manipuler TOUTES les catégories d'articles possibles. En plus, seule une partie de cet ensemble sera effectivement nécessaire pour une instance donnée.
Par exemple, une variable Type_de_manche (= courtes ou longues) est nécessaire pour l'enregistrement d'une chemise, mais elle est superflue pour un aspirateur ! De même, la méthode prixTTC ne convient pas pour tous les articles - les produits de luxe ont un taux TVA plus élevé que des produits de consommation.

La structuration en classes et sous-classes entraîne une modularité qui sera notamment importante pour la composition d'objets fractals : la description de l'univers est divisée en parties indépendantes. La modification de l'une d'elles n'entraîne qu'un minimum de modifications des autres. Plus précisément, chaque sous-arbre regroupe les objets partageant les caractéristiques de sa racine. La modification de cette racine n'a d'effets que sur ses sous-classes.

5.2.5. L'héritage multiple

L' héritage, tel qu'il a été considéré jusqu'à présent, est dit simple : une classe ne posséde qu'une seule superclasse directe et la relation d'héritage est représentée par une arborescense. Ce principe n'est pas satisfaisant, car deux classes appartenant à des branches différentes de l'arbre peuvent très bien avoir des propriétés communes. De nouveau, ce problème peut être résolu en dotant ces classes de variables communes, mais cela entrainerait des duplications d'informations. Une solution bien plus en rapport avec la philosophie objet consiste à donner à une classe la possibilité d'hériter directement de plusieurs autres classes.

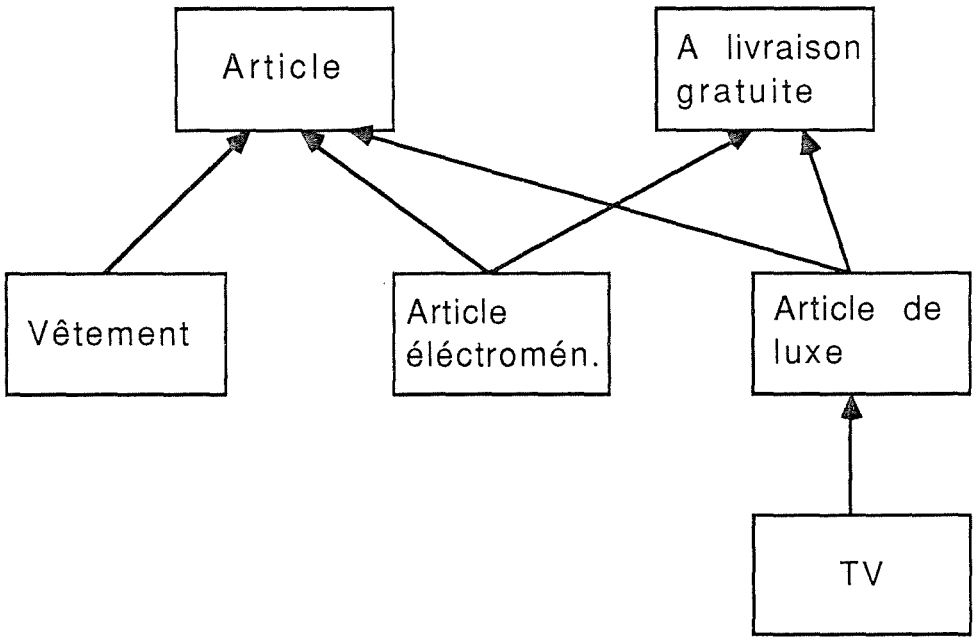


Figure 5.4. : Graphe d'héritage - héritage multiple

Dans cette perspective, une classe possède plusieurs superclasses directes, et l'héritage est dit multiple. L'ensemble des classes n'est plus structurée en arborescence mais forme un graphe orienté sans circuit. Ainsi une classe hérite de l'**union** des variables et des méthodes de ses superclasses.

L'avantage de l'héritage multiple est d'accroître encore la modularité des programmes. Il permet généralement un gain de place par mise en commun d'informations, programmes et données, qu'il faudrait sinon dupliquer dans plusieurs classes.

L'héritage repose sur deux composantes. L'une est statique : c'est la relation qui lie un objet à celui ou ceux dont il hérite et qui est représentée par le graphe d'héritage. L'autre est dynamique : c'est le processus qui réalise l'héritage en donnant aux objets l'accès aux informations dont ils héritent.

5.2.6. La réalisation d'héritage :

L'héritage simple

Comme chaque objet possède un seul père, le graphe d'héritage est un arbre et la fermeture transitive de la relation d'héritage sur l'arbre d'héritage d'un objet est une relation d'ordre total. Ainsi, les superclasses de la classe Téléviseur sont ordonnées de la façon suivante :

Téléviseur, ArticleDeLuxe, Article, Objet.

Les classes sont examinées dans l'ordre établi, de la plus spécifique à la plus générale, en s'arrêtant à la première qui possède la propriété cherchée : La première occurence de la propriété cherchée masque les autres. L'ordre du parcours de la hiérarchie est bien fixé puisqu'il est établi grâce à une relation d'ordre total.

Par exemple, si une instance de la classe Téléviseur reçoit un message prixTTC, la recherche commence dans la classe d'instance, Téléviseur, et se poursuit successivement dans les classes ArticleDeLuxe, Article et Objet, s'il y a lieu.

L'héritage multiple

Un objet peut dans ce cas-ci avoir plusieurs pères. Le graphe d'héritage d'un objet n'est donc plus un arbre et la fermeture transitive de la relation d'héritage n'est plus qu'une relation d'ordre partiel. Certains objets ne sont plus comparables et le parcours peut effectivement donner lieu à des ambiguités. Chaque langage de programmation définit ses propres solutions.

5.3. Les fractales et les objets

5.3.1. Introduction

L'essai présenté ici à pour but de faire plus qu'une simple implémentation des fractales en langage objet. Une première partie du travail informatique consiste en une implémentation structurée des IFS.

Le point de départ est la génération de l'algorithme déterministe et de l'algorithme-chaos par l'approche objet : on définira une classe nommée 'IFS' dans laquelle on va définir les instances qui y sont associées. La figure 2.3. peut ainsi être l'attracteur généré par la méthode 'Dessiner' de l'instance 'farn' appartenant à la classe 'IFS'. On peut ainsi obtenir un vocabulaire d'instructions ressemblant presque au langage naturel.

Une première mise en oeuvre de l'héritage sera la complexification de la classe des fractales : On définit la classe 'IFS_coloré' qui à première vue est un enrichissement de la classe 'IFS'.

Mais on verra que l'héritage des classes à des conséquences plus importantes : la génération naturelle et souple d'un Méta-IFS est devenue possible ! Grâce aux objets et à l'héritage des classes, on va pouvoir PROGRAMMER des objets fractals composés en se servant simplement des méthodes définies :

Une branche est un Méta-IFS
Une feuille est un fractal du type IFS-chaos
Une branche est composée de feuilles
Dessiner une branche
Montrer les objets détenus par la branche
Agrandir une partie de la feuille
Colorer la feuille ...

5.3.2. Construction des classes

Dans ce paragraphe, on trouvera une justification de la modularisation choisie. Les classes sont présentées sans que leur contenus soient décrits en détail. Ci-après donc une manière de classifier le monde des fractales :

A. La classe IFS

Lorsqu'on passe du programme pascal aux objets, la toute première idée est de représenter un IFS par une classe et de remplacer les appels de procédure par des instances comme "Triangle_Sierpinski", "Farn" ou "Feuille". Ce sont les objets qui appartiennent à la classe IFS.

La définition de la classe va contenir, suivant le principe du paragraphe 2.2., comme champs de données :

- les transformations $\{ w_n , n=1,2,3...N \}$
- les probabilités associées $\{ p_n, n=1,2,...N \}$ tel que :

$$p_1 + p_2 + p_3 + ...p_N = 1 \text{ et } p_i > 0,\ i=1,2,...N$$

et momentanément, sa seule méthode permet de dessiner l'attracteur :

- Dessiner() qui à ce stade-ci est un simple copiage du programme pascal.

On verra plus tard que cette encapsulation offre de nombreux avantages. De nombreuses méthodes seront ajoutées, et parmi les nouvelles ajoutées figure un "point_dansant" qui est ignoré de l'extérieur mais qui permet d'accroître considérablement les performances des algorithmes précédents.

B. Héritage : la classe Point_H

Passons en revue ce qu'on a établi au chapitre 1 : On a défini l'espace métrique de Hausdorff dont tout point est un ensemble compact non vide de l'espace métrique sous-jacent X. Plus précisément, toute image qui est un rectangle, une ligne ou un autre objet est un point de H(X).

Par le Théorème 1, on venait de trouver que l'attracteur d'un IFS est aussi un point de H(X). Il semble donc être naturel de définir une classe "Point_H" qui détient des données et des méthodes qui définissent des actions "générales" sur les dessins ou points de Hausdorff.

La classe IFS est bien une sous-classe de Point_H, car on peut dire que l'IFS (ou plutôt l'attracteur de l'IFS) **EST** un point de H(X).

Si on examine un logiciel de dessin comme *Corel Draw* ou *Mac Draw* , on constate qu'il existe un ensemble d'"objets" prédéfinis. Les plus classiques sont les lignes, les cercles, ...

Un tel logiciel de dessin vectoriel est constitué de sorte que tout objet appartenant à l'écran peut être sélectionné, déplacé, changé ou effacé indépendamment. Un moyen de chaînage est donc sous-jacent à ce principe. Par conséquent, un champs détenu par la classe Point_H est un pointeur vers l'image suivante et les méthodes doivent permettre d'agir sur ce pointeur.

Une notion commune à toute figure est celle de "domaine", c.à.d. l'endroit dans lequel la figure doit se reproduire. Le domaine est donc repris dans la classe Point_H et les méthodes asociées y sont ajoutées aussi. Voici un premier graphe d'héritage :

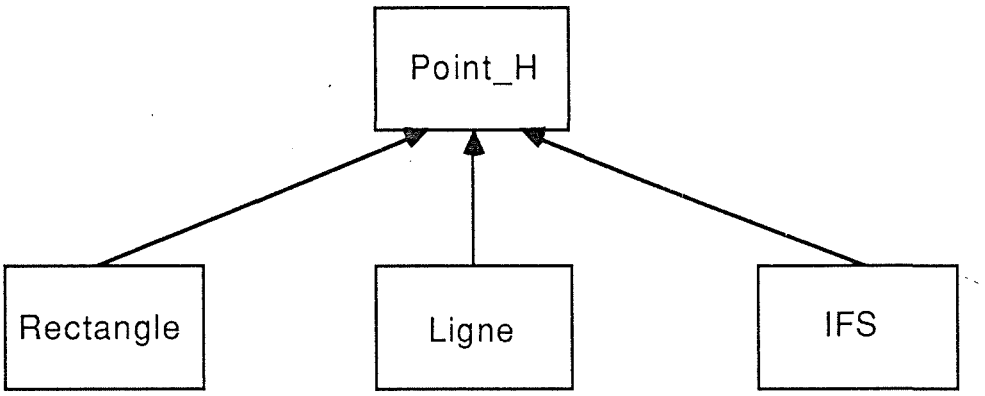


Figure 5.5. : Afin de rendre la hiérarchie établie plus riche en structures, il est intéressant de définir d'autres objets (non-fractals) comme une ligne et un rectangle. Ces objets sont des points de H(X)

C. Héritage : la classe IFS_coloré

Dans le chapitre concernant les mesures sur les fractales, on a proposé un algorithme qui permet de colorer un IFS.

Du programme (voir aussi Annexe) on peut constater qu'un IFS coloré EST un IFS auquel on associe un compteur pour la détermination des couleurs. Il s'agit donc d'une spécialisation de la classe mère IFS : la classe IFS_coloré contient une nouvelle définition à la méthode héritée Dessiner() :

- Dessiner() qui va se baser sur les connaissances héritées de la superclasse ou classe mère (les transformations et probabilités)

L'enrichissement en variables d'instance consiste en :

- bitmap qui est une matrice contenant les compteurs des couleurs
- couleurs, le tableau des couleurs définies

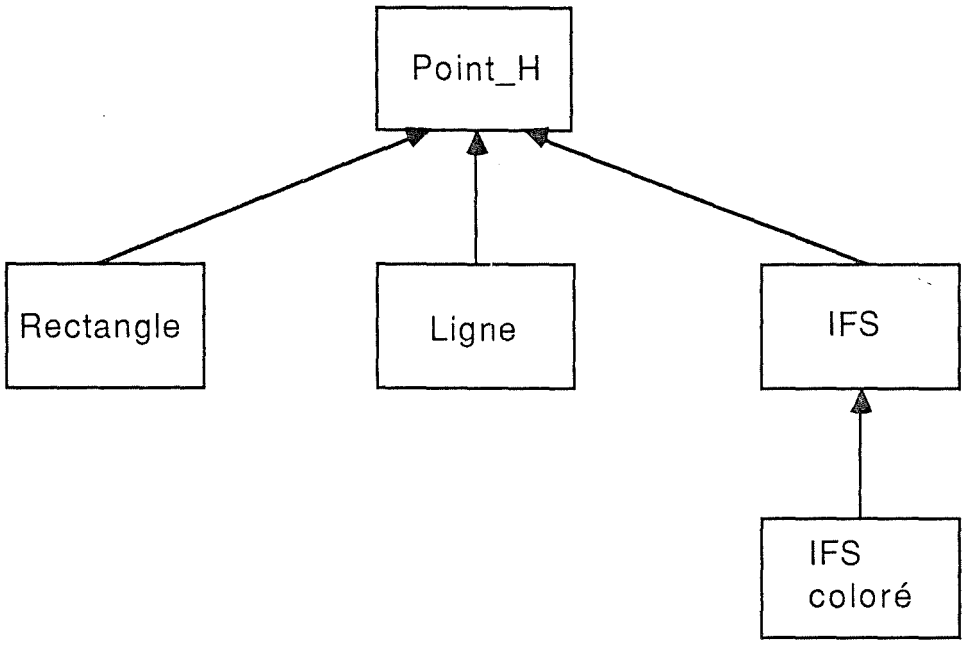


Figure 5.6. : La hiérarchie est complétée en développant le fait qu'un IFS coloré est un IFS.

D. La classe Méta-IFS

Considérons la définition d'un Méta-IFS :
$\{ X , \{ w_n^1 , n=1,2,...N_1 \} , ... \{ w_n^m , n=1,2,...N_m \} , w^*_n \quad n=1,... N^* \}$, il est clair qu'il faut considérer que les m IFS dominés font partie de la classe IFS. Par suite, un Méta-IFS à comme données un ensemble de transformations $\{ w^*_n , n=1,2,...N^* \}$ et il doit être capable d'enchaîner les objets IFS pour former une liste des attracteurs dominés. Cela est possible du fait que la classe IFS est une sous-classe de Point_H et que dans cette dernière classe se trouvent les méthodes qui permettent d'enchainer l'objet.

C'est ainsi que les algorithmes associés aux Méta-IFS seront compris dans les méthodes dessiner_det et dessiner_chaos.

C'est la méthode exécutant l'algorithme déterministe (voir point 3.3.1.) du Méta-IFS qui va générer les messages demandant à l'IFS, étant un Point de Hausdorff, de modifier son domaine suite à des transformations de sa part. L'algorithme exige alors à un certain moment de l'IFS qu'il se dessine sur son domaine que l'IFS vient de modifer suite à ces demandes préalables.

L'algorithme chaos ne va pas demander de changements de domaine, mais, comme on a précisé au point 3.3.2., il aura besoin de la génération d'un point au hasard de la part d'un IFS dominé. Ces propriétés nécessitent des mécanismes plus complexes qui sont exposés au point suivant.

Remarque : Afin de rendre les algorithmes plus puissants, il semble être intéressant de ne pas simplement insérer un IFS, mais de permettre à des objets non-fractals d'être reproduits par le Méta-IFS. Il faut dès lors que tous les objets appartenants aux sous-classes de Point_H soient capables de se reproduire sur un domaine et de générer un point (qui fait partie de leur image) au hasard. Voici la hierarchie résultante qui à été implémentée concrètement :

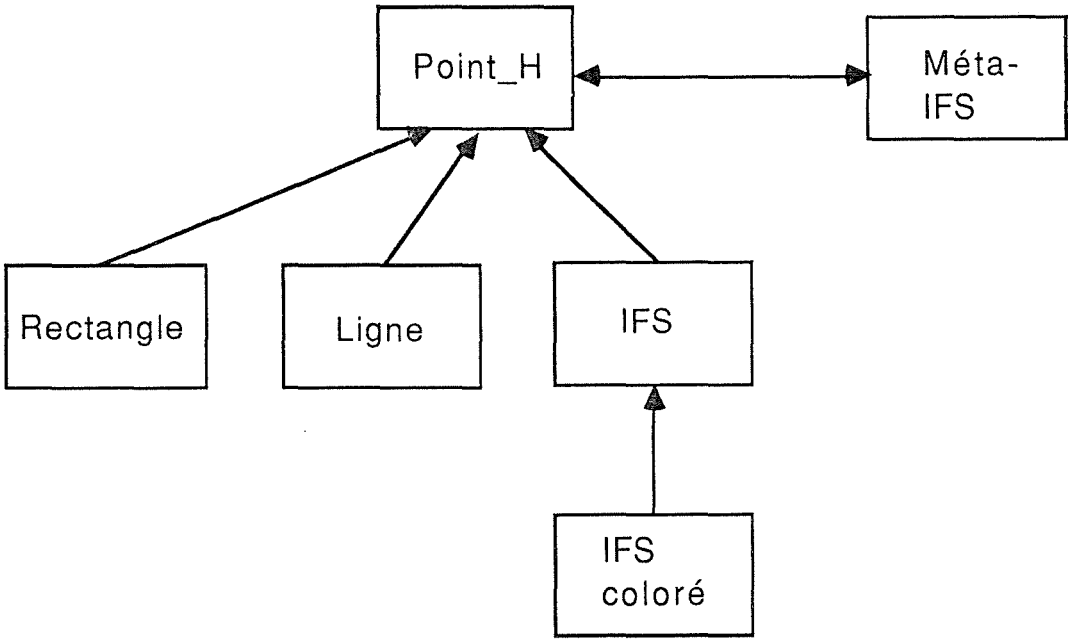


Figure 5.7. : Ceci est une possibilité de découpe en classes, elle a été choisie car elle semble être la plus cohérente avec la théorie.

On pourrait envisager d'autres classifications :

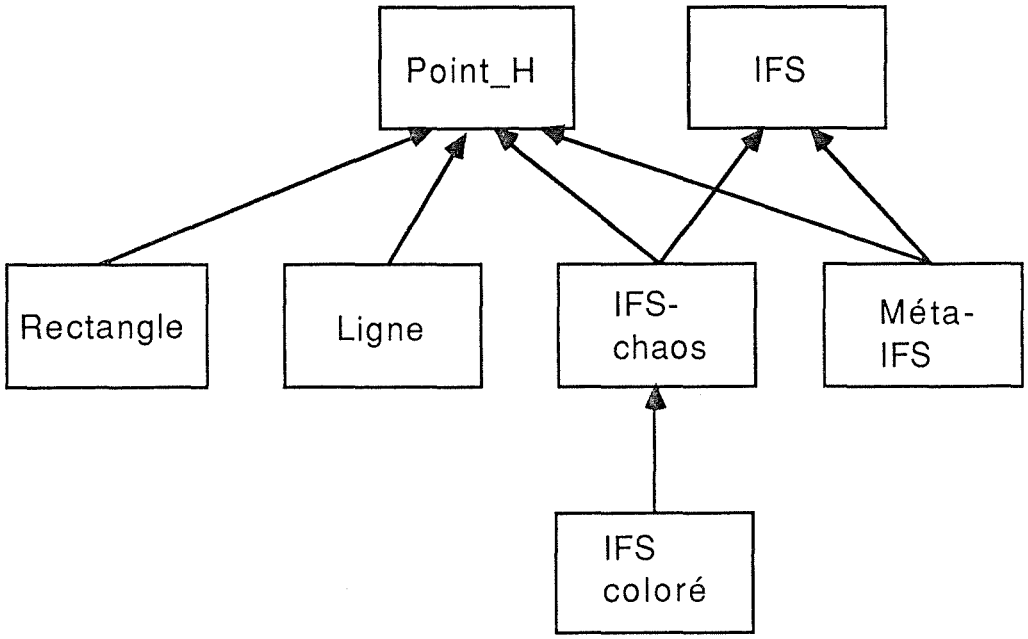


Figure 5.8. : Une classification qui fait intervenir l'héritage multiple

5.3.3. Implémentation C++

A. Le langage C++

Donnons ici quelques notions non encore mentionnées dans les points précédents mais qui font partie du langage objet C++. Il est clair qu'une présentation exhaustive n'est pas envisageable dans ce contexte-ci. Néanmons, il existe quelques concepts dont la description s'impose pour comprendre l'architecture détaillée des classes.

On peut dire qu'une classe C++ contient deux attributs ou membres, à savoir l' ensemble des données et l'ensemble des méthodes.
La déclaration des membres-données est equivalente à celle utilisée en C.
Les membres-méthodes qui prennent plusieures lignes de code sont définies généralement en dehors de la classe, on trouve alors la déclaration de leurs sélecteurs à l'intérieur de cette classe.

La protection d'informations est un mécanisme pour restreindre l'accès de l'utilisateur à la représentation interne d'un type de classe. Cette abstraction est spécifiée par des sections **public, protected** et **private** de la classe. Des membres qui sont déclarés à l'intérieur de la section "public" deviennent des membres publiques, ceux déclarés dans la section protected sont des membres protégés et ceux déclarés dans la section private forment les membres privés :

* Un **membre "public"** est accessible de tout endroit du programme. Une classe qui met l'accent sur la protection de l'information limite les membres publics aux méthodes.

* Un **membre "protégé"**d'une classe se comporte comme un membre "public" à toute sous-classe et il est vu comme un membre "privé" pour tout le reste du programme.

* Un **membre"privé"** peut être référencé uniquement par les méthodes de la classe même et par les classes amies. (Les méthodes d'une classe qui est déclarée amie d'une seconde, peuvent accéder aux champs non-publics.) Une classe qui renforce la protection de l'information déclare toutes ces champs comme membres privés.

Considérons le problème suivant : La classe Point_H contient une méthode virtuelle Dessiner(). On a précisé avant que tout objet de Point_H peut être inséré dans une liste détenue par le Méta-IFS. Il est clair que le Méta-IFS, ignorant la nature exacte de ses éléments, doit pouvoir demander à un objet de se dessiner. Par conséquent, la classe Point_H doit contenir une fonction qui permet de dessiner toute figure, or elle ignore la nature de ses sous-classes.

C++ offre un remède à ce genre de problèmes : la fonction virtuelle. Une fonction ou méthode est virtuelle lorsque des détails d'implémentation actuels sont dépendants du type de la classe et ne sont pas connus à ce temps là.

La fonction virtuelle est une sorte de "panneau" qui renvoie l'appel vers ses sous-classes.

* Si une définition est donnée, elle sert comme méthode par défaut pour une sous classe si cette sous classe décide de ne pas procurer sa propre définition de la fonction.

* Si une fonction est purement virtuelle, la sous classe dérivée DOIT définir la méthode.

B. Description des classes

Classe : Point_H		
Champs :		
PROTECTED :	Suivant	int
	(originex, originey)	(float, float)
	(axeix, axeiy)	(float, float)
	(axejx, axejy)	(float, float)
Méthodes :		
PUBLIC :	Modifier_Domaine	void
	Transmettre_Domaine	void
	Tronquer	int
	Sélectionné	int
	Dessiner	void
	Génère_Point	void
	Modifier_Elem_Suivant	void
	ElemSuivant	Point_H*
PROTECTED :	Point_H	void

Figure 5.9.

CHAMPS :

* Suivant est un pointeur vers un objet appartenant à la classe Point_H. Ce champs permet de chaîner les figures et il est utilisé par le Méta-IFS pour établir la liste des attracteurs dominés.

* (originex,originey), (axeix,axeiy), (axejx,axejy) forment le repère (ou domaine) d'une figure qui est donc défini par une origine et deux axes. Il est absolument nécessaire de l'exprimer sous forme réelle pour ne pas obtenir une perte de précision. Ces champs sont en mode *protected* afin d'être utilisés par des sous-classes.

METHODES :

* Point_H () est le constructeur de la classe Point_H qui est mis en *protected* (!), car en soi, l'existence d'un objet Point_H n'appartenant à aucune des sous-classes n'a pas de sens. Il faut qu'il soit du type IFS, ligne,... C'est ainsi que le constructeur va être utilisé exclusivement par les constructeurs des sous-classes.

Les méthodes associées au repère :

* Modifier_Domaine (norx : float, nory : float, naix : float, naiy : float, najx : float, najy : float)
  modifie le domaine d'une figure. Le nouveau repère est indiqué par les paramètres contenus dans le sélecteur de la méthode. Remarquons qu'il est possible de faire des tests de validité à ce moment.

* Transmettre_Domaine (norx : float, nory : float, naix : float, naiy : float,najx : float, najy : float)
  La valeur actuelle du domaine est transmise aux paramètres.

* Tronquer () effectue une troncature du domaine actuel par rapport au domaine initial (i.e. (0,0),(100,0),(0,100)). Toute coordonnée de l'origine ou d'un des axes dépassant une de ces valeurs est tronquée à celle-ci.
  Valeurs de retour : 0 : si aucune troncature n'a eu lieu,
  1 : si le domaine a été tronqué.

* Sélectionné (x : int, y : int) vérifie si le point de coordonnées (x,y) se trouve à l'intérieur du domaine de la figure.
  Valeurs de retour : 1 si la détection a eu lieu
  0 sinon

* Dessiner () est une méthode virtuelle, on renvoie la tâche aux sous-classes qui doivent contenir une méthode du même nom. De la définition du constructeur, on est assuré que l'objet fait partie d'une des sous-classes de Point_H.

* Génère_Point () est une seconde méthode virtuelle qui renvoie la tâche de génération d'un point aux sous-classes.

Les actions sur le pointeur :

* Modifer_Elem_suivant (Element : Point_H&) fait pointer Suivant vers l'objet de type Point_H passé en paramètre.

* ElemSuivant () a comme valeur de retour : Suivant

Classe : Ligne	Superclasse : Point_H	
Champs :		
PRIVATE :	couleur	int
	type	char
Méthodes :		
PUBLIC :	Ligne	void
	Dessiner	void
	Génère_Point	void
	Clone	Ligne*

Figure 5.10.

CHAMPS :

* couleur est la couleur de la ligne. Cette valeur n'est pas du même type pour toutes les figures et, par conséquent, ne peut être reprise par la classe Point_H.

* type représente le type de la ligne ('H' pour horizontal, 'V' pour vertical)

METHODES :

* Ligne (rouge : int, vert : int, bleu : int) est le constructeur de la classe qui établit la couleur à partir des quantités en rouge, vert et bleu (valeurs de 0 à 255).

* Dessiner () dessine l'objet ligne de couleur et de type précisé par les champs sur le domaine donné par la classe-mère Point_H.

* Génère_Point (x,y : float, coul : int) génère un point (x,y) faisant partie de la ligne.

* Clone () produit un clone de l'objet, sa valeur de retour est un pointeur vers le clone.

Classe : Rectangle	Superclasse : Point_H	
Champs :		
PRIVATE :	couleur	int
Méthodes :		
PUBLIC :	Rectangle	void
	Dessiner	void
	Génère_Point	void
	Clone	Rectangle*

Figure 5.11.

CHAMPS :

* couleur est la couleur du rectangle.

METHODES :

* Rectangle (rouge : int, vert : int, bleu : int) est le constructeur de la classe rectangle qui établit la couleur à partir des quantités en rouge, vert et bleu
(valeurs de 0 à 255).

* Dessiner () dessine l'objet rectangle de couleur et de type précisé par les champs sur le domaine donné par la classe-mère Point_H.

* Génère_Point (x,y : float, coul : int) génère un point (x,y) faisant partie du rectangle.

* Clone () produit un clone de l'objet rectangle, sa valeur de retour est un pointeur vers le clone.

Classe : IFS		Superclasse : Point_H
Champs :		
PRIVATE :	couleur	int
	ntrans	int
	ta,tb,tc,td,te,tf(*)	float
	probab(*)	float
	précision	int
	qualité	int
	point_dansant	(float,float)
Méthodes :		
PUBLIC :	ifs	void
	Dessiner	void
	Génère_Point	void
	Progresser_Trans	int
	Animation	bool
	Zoom	int
	Encadrer	void
	Clone	ifs*

Figure 5.12.

CHAMPS :

* couleur est la couleur de l'attracteur obtenue à partir des taux en rouge, bleu et vert.
* ntrans : le nombre de transformations
* ta,tb,tc,td,te,tf(*) : les tableaux qui contiennent les transformations
* probab(*) est un tableau qui détient les probabilités
* précision représente la densité de l'image, sa définition exacte est développée lors de la description de la méthode Dessiner()

* qualité est un paramètre qui détermine la qualité des transformations. Les différentes valeurs possibles sont :
  1 : l'IFS est de bonne qualité, la génération de l'attracteur ne produit pas de points dépassant le domaine. Aucun test de validité n'est nécessaire.

2 : l'IFS est de qualité non spécifiée ou inconnue, un test déterminant le comportement des transformations est exigé de la part de l'objet. L'IFS est alors classifié selon 1 ou 3.
3 : l'IFS est de mauvaise qualité : les transformations risquent de générer des points dépassant le domaine. Par conséquent, à chaque itération, un test de validité est ajouté. Ceci réduit les performances de l'algorithme d'environ 15 %.

* point_dansant est donc le point dansant utilisé par l'algorithme chaos. Sa présence dans la partie champs est justifiée dans la méthode Génère_Point() et dans le constructeur.

METHODES :

* IFS (ntrans : int, a,b,c,d,e,f(*) : float, préc : int, qual : int, rouge : int, vert : int, bleu : int) est le constructeur de la classe IFS. Les valeurs des sept premiers paramètres sont bien connues.
précision est un paramètre qui détermine la densité de l'image de l'attracteur. Faisant varier la précision de 1 à 10, on obtient des attracteurs de plus en plus denses, c'est le nombre d'itérations qui dépend continument de ce paramètre
(voir aussi Dessiner()).
qualité a été décrit dans les champs. Il n'est toutefois pas toujours préférable de laisser la détermination de la qualité de l'IFS à l'objet lui-même (qualité = 2), car il se peut que l'objet détecte un dépassement de repère d'une des transformations, mais que, néanmoins, l'attracteur reste bien dans le domaine d'admissibilité (voir figure 5.13.).

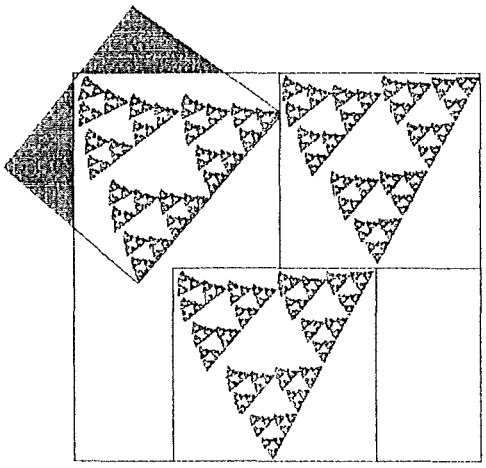


Figure 5.13. : L'objet détecte un dépassement de domaine

Remarquons ici que le point dansant est ignoré de l'extérieur. Lorsqu'on considère le programme pascal de l'algorithme chaos se trouvant en annexe, on voit qu'il est nécessaire, avant de passer à la génération de l'attracteur, d'effectuer quelques itérations pour déterminer le point de départ qui se trouve à l'intérieur de l'attracteur.

En ne considérant plus ce point comme une variable locale à la méthode Dessiner(), on peut effectuer cette partie "initialisation" dans le **constructeur** de la classe. On obtient ainsi un premier gain en efficacité, car si on demande plusieurs fois à l'objet de se dessiner, la méthode prend simplement le point dansant, **sachant qu'il se trouve à tout moment dans l'attracteur.**

* Dessiner () : Comme mentionné avant, toute figure doit être capable de se dessiner sur un domaine quelconque. On peut dériver la procédure finale en considérant les étapes successives :

  La formule fractale à été définie sur un domaine (0,0), (100,0), (0,100) et tous les points sont générés à l'intérieur de celui-ci. Le changement de repère peut être vu de la façon suivante :

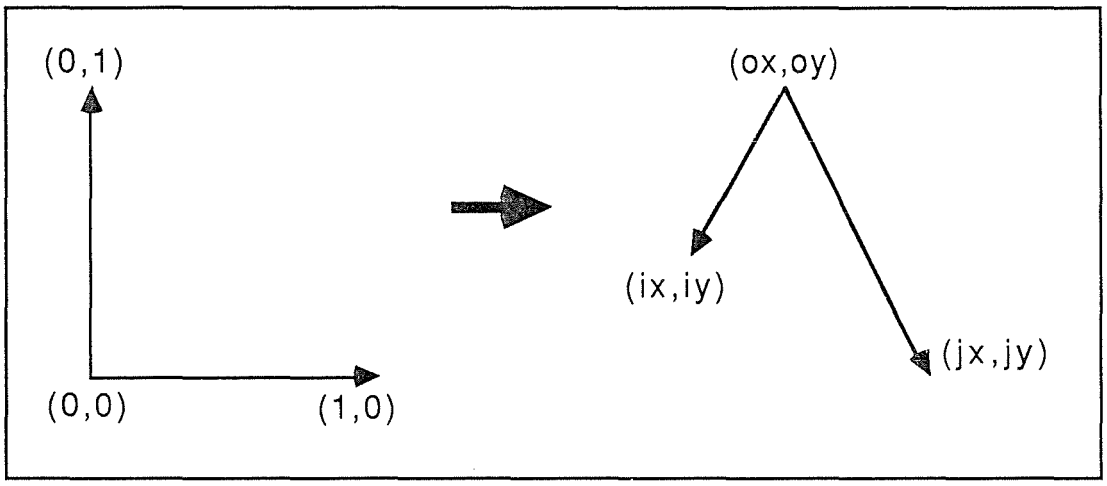


Figure 5.14. : Le changement de repère

  * Il est clair qu'on ne peut pas agir sur le point dansant pour effectuer le changement de domaine, car la formule fractale ramène toujours le point vers le domaine 100x100.
  * Il faut donc définir à chaque itération un nouveau point (px,py) qui est obtenu à partir du point dansant (x,y) par le changement de coordonnées qui se refère à la figure 5.14. :

```
px = ( ix - ox ) * x  +  ( jx - ox ) * y  +  ox
py = ( iy - oy ) * y  +  ( jy - oy ) * y  +  oy
```

PAS1 : En considérant de noueau le programme pascal de l'algorithme chaos, on voit que le nombre d'itérations a été fixé pour une taille donnée de l'image.

PAS2 : Les changements de domaine nécessitent un changement du nombre d'itérations. Le nombre d'itérations doit s'adapter à la taille du repère, ou bien, pour être plus précis, au rapport entre l'aire occupée par le domaine intitial et celle du nouveau domaine.

Le calcul de l'aire d'un domaine quelconque a la forme suivante :

$$aire = \| i \| * \| j \| * \sin(\alpha)$$

les différents composantes de la formule peuvent être obtenus par :

$$\| i \| = \sqrt{(ix - ox)^2 + (iy - oy)^2}$$
$$\| j \| = \sqrt{(jx - ox)^2 + (jy - oy)^2}$$

et

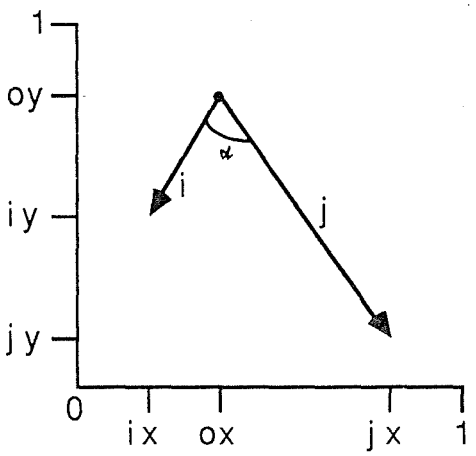


Figure 5.15. : Calcul d'une surface déterminée par deux vecteurs

$$< i , j > = (ix - ox)(jx - ox) + (iy - oy)(jy - oy)$$

$$\text{et} \quad \alpha = \arccos\left(\frac{< i , j >}{\| i \| * \| j \|}\right)$$

ayant obtenu l'aire du nouveau repére, on déduit le nombre d'itérations (*newiter*) à partir de l'ancienne valeur *iter* par :

newiter = iter * aire2 / aire1

où aire2 est l'aire occupée par le nouveau domaine
aire1 est l'aire du repère 100x100, donc aire1 = 10000.

PAS3 : On peut remplacer le paramètre 'nombre d'itérations' par une quantité qui indique simplement la **densité** de l'image. Le nombre d'itérations est alors pris comme un multiple de cette valeur et il est adapté au domaine :

newiter = densité * 1000 * aire2 / aire1

En conclusion, on peut retenir que, chaque fois qu'un objet va être dessiné, la méthode doit déterminer le nombre d'itérations nécessaires pour satisfaire la précision exigée. Ce nombre étant variable du fait qu'il peut être changé par les méthodes de la classe-mère Point_H.

* Génère_Point (x : float, y : int, coul : int) : Du fait que l'objet possède le point dansant dans sa partie champs, la génération d'un point au hasard est élémentaire : il suffit de choisir une transformation au hasard, de l'appliquer à son point dansant et de passer le résultat en paramètre.

Remarque : C'est cette méthode qui est déclenchée une multitude de fois lors de la génération des attracteurs dominés d'un Méta-IFS par l'algorithme chaos (voir aussi paragraphe 3.3.2.). Il est impossible d'implémenter l'algorithme chaos associé au Méta-IFS de manière efficace si l'objet IFS ne possède pas le point dansant dans son champs, le lecteur devrait en savoir la raison.

D'autres fonctions deviennent possibles : On peut réaliser une animation qui fait passer continument l'image d'un attracteur vers celle d'un second. Le Théorème 1 du point 2.3. a établi cette dépendance continue des fractales en rapport à leurs paramètres. Un programme pascal (F_ARCANI.PAS) montre cette propriété.

Mais, au lieu d'établir cet effet par des appels de procédures successifs, il est beaucoup plus naturel de résoudre ceci par l'approche objets. On va simplement demander à un objet fractal de s'"animer" vers un second. Ce sont les deux objets qui vont interagir et produire ainsi l'animation du premier. Le principe est basé sur le fait que tout IFS possède une méthode

* Progresser_Trans (a,b,c,d,e,f : float, ntrans : int) . Les paramètres sont des valeurs d'une transformation quelconque et ntrans est le numéro de la transformation à laquelle les paramètres sont sensés de "se rapprocher". Ainsi, la méthode compare chaque composante de sa transformation au paramètre correspondant et décide de quel taux cette valeur est modifiée :

```
bool Progresser_Trans ( a,b,c,d,e,f : float ; ntrans : int )

res = FALSE

si  ta(ntrans) > ( a + 0.01 ) alors   a := a + 0.01
                                      res := TRUE
si  ta(ntrans) < ( a - 0.01 ) alors   a := a - 0.01
                                      res := TRUE

si  tb(ntrans) > ( b + 0.01 ) alors   b := b + 0.01
                                      res := TRUE
...

si  tf(ntrans) < ( f - 0.01 ) alors    f := f - 0.01
                                      res := TRUE
return  res
```

Sa valeur de retour est donc de type boléen qui indique si une progression a eu lieu.

Pour réaliser une animation, la méthode d'un IFS

* Animation (Ptrifs : IFS*) demande, pour toutes ses transformations, à un second IFS de faire progresser les transformations du premier (en les mettant en paramètre de Progresser_Trans) tant que ceci reste possible
(valeur de retour TRUE). L'algorithme peut se présenter sous cette forme :

```
bool Animation ( Ptrifs : IFS* )

si ( Ptrifs.ntrans() = ntrans )

   répéter
      fin := TRUE

      pour i = 1 à ntrans faire
         si ( Ptrifs.Progresser_Trans(a,b,c,d,e,f,i) )
            fin := FALSE
            Dessiner()
         fin si
      fin i

   jusqu'à fin
   return TRUE

sinon
   return FALSE
fin si
```

ntrans() est une méthode de la classe IFS qui fournit le nombre de transformations.

* Zoom (pt1x, pt1y, pt2x, pt2y : int) est une méthode qui se sert du fait qu'une image fractale ne perd pas de précision lors d'un agrandissement. En principe, l'algorithme chaos est appliqué pour générer un point (toujours compris dans le domaine 100x100). Ceci peut être réalisé par l'objet qui envoie un message Génère_Point(x,y,c) vers lui-même. Néanmoins, pour des raisons d'efficacité, il est préférable de recopier cette génération d'un point dans la méthode même.
Ayant donc obtenu un point au hasard de l'attracteur, la méthode vérifie ensuite si ce point se trouve dans une fenêtre du zoom déterminée par deux points passés en paramètre. Si c'est le cas, le point subit deux changements de coordonnées :
- un premier réalise l'agrandissement de la fenêtre zoom sur 100x100 et
- le deuxième le transforme sur le domaine actuel de l'IFS.

Il est important de voir que la sélection du zoom se fait sur 100x100 et non sur le domaine actuel de l'objet IFS. Des programmes utilisant cette méthode doivent en tenir compte. (voir aussi listing de l'"Editeur d'IFS" en annexe)
Valeurs de retour : 0 : exécution normale
-1 : fenêtre zoom non-valide
-2 : le nombre maximum d'itérations fixé à 64000 est atteint

* Encadrer () réalise l'encadrement du domaine et
* Clone () retourne un pointeur vers la duplication de l'objet.

Classe : IFS_coloré	Superclasse : IFS	
Champs :		
PRIVATE :	brillance	int
	ncouleurs	int
	couleur(*)	int
	color_scaling	float
	bitmap(100,100)	int
Méthodes :		
PUBLIC :	IFS_coloré	void
	Dessiner	void
	Zoom	int

Figure 5.16.

CHAMPS :

Un IFS_coloré est une spécialisation d'un IFS qui possède des champs supplémentaires concernant le coloriage :

* brillance est un paramètre qui ajuste la clarté de l'image. Des changements de valeur conduisent à des changements dans le choix des couleurs prédéfinies.
  Ainsi, la figure 4.5.a. représente une fougère de brillance = 5, alors que si on augmente ce paramètre, des nuances en jaune et magenta vont dominer l'image. Cela donne toujours le même TYPE de coloriage - càd. clair aux pointes, plus foncé vers l'intérieur - et il n'est pas à confondre avec la figure 4.5.b où le type de coloriage a changé avec une nouvelle distribution des probabilités.

* ncouleurs est le nombre de couleurs que possède l'objet.

* couleur (*) est un tableau qui contient l'ensemble des couleurs définies pour la génération de l'image. On suppose ici que les couleurs sont rangées selon leur luminosité. L'ensemble des couleurs utilisées pour établir la figure 4.5. est vert, vert-clair, magenta, magenta-clair, jaune.

* bitmap (100,100) est une matrice d'entiers dont chaque élément réalise le comptage des points générés dans le domaine 100x100.

* color_scaling : un facteur qui détermine le passage entre le compteur du pixel et sa couleur. La raison exacte d'existence de ce champs est donnée dans la méthode Dessiner(). Color_scaling est calculé à partir de :

  numits = précision * 1000 , car aire2 = 10000 = aire1
  color_scaling = brillance * 1000/numits

  et par après, on obtient la couleur d'un pixel (i,j) par :

  coleur(i,j) = color_scaling * bitmap(i,j)

  Il faut être attentif a ne pas conclure de la première formule que précision et brillance sont inverses l'un de l'autre ! Ils ont des effets tout à fait différents, le lecteur devrait voir les nuances.

METHODES :

* IFS_coloré (ntrans : int; a,b,c,d,e,f : float; précision, qualité, brillance, ncouleurs : int; couleurs(*) : int)

  est le constructeur de la classe. Une bonne valeur de brillance est aux environs de 5 à 20, toutes les initialisations des autres paramètres ont déjà été discutées précédemment.

* Dessiner () est une redéfinition de la méthode de la classe IFS, le principe d'un tel algorithme avec coloriage a été discuté au chapitre 4. Afin de gagner en efficacité lors d'un tel dessin, on propose d'établir une seule fois le bitmap si les paramètres restent constants.

  Le problème est que l'IFS coloré est incapable de dire si un paramètre comme par exemple 'précision' a changé entretemps, car il appartient à sa superclasse IFS, et, il se pourrait que cette classe détienne une méthode qui le modifie. Une solution peu efficace et contre la 'philosophie objet' serait de dupliquer le code, il y a moyen de faire mieux :

  Un test peut être obtenu en recalculant le color_scaling avec les valeurs actuelles des champs et en le comparant au champ color_scaling associé au bitmap actuel. Ainsi, la modification d'un paramètre est détectée. De plus, si une méthode modifie le bitmap, elle doit agir sur le paramètre color_scaling associé (elle peut le mettre à 0 pour forcer un nouveau calcul du bitmap).

  La méthode qui s'occupe de dessiner l'IFS coloré teste donc d'abord si le bitmap est 'up-to-date', si ce n'est pas le cas, l'algorithme chaos est appliqué pour ainsi faire un comptage des points générés selon le principe établi précédemment.

  Ensuite, il s'agit de dessiner le bitmap sur un domaine déterminé par la classe Point_H. Remarquons ici, que si la taille du repère dépasse 100x100, on obtient une perte de précision. On peut remédier à ce problème en chosissant un bitmap de taille supérieure.

Le Zoom par contre ne perd pas de son efficacité :

* Zoom (pt1x, pt1y, pt2x, pt2y : int) : Le zoom sur un IFS coloré suit exactement le même principe que pour l'IFS. La seule chose qui change est que le point se situant dans la fenêtre du zoom n'est plus dessiné sur le domaine, mais il est ajouté dans un bitmap qui définit ainsi un "domaine" 100x100.
  Le bitmap étant établi, il reste à dessiner celui-ci sur le domaine dont les valeurs sont détenues par la classe Point_H.

  Comme le bitmap est utilisé pour générer le zoom, il n'est plus à jour. Par conséquent, color_scaling est mis à 0.

Classe :	Méta-IFS	
Champs :		
PRIVATE :	ntrans	int
	ta,tb,tc,td,te,tf(*)	float
	probab(*)	float
	(ox,oy),(ix,iy),(jx,jy)	float
	premier	Point_H*
	dernier	Point_H*
Méthodes :		
PUBLIC :	Méta-IFS	void
	Insérer	int
	Extraire	void
	Avant_Plan	void
	Appliquer_Tans	int
	Elément_Séléctionné	int
	Montrer_Composition	void
	Dessiner_Dét	void
	Dessiner_Chaos	void
	Transformer	void
PRIVATE :	Génère_Trans	void

Figure 5.17. : La classe Méta-IFS avec une méthode locale ou privée

CHAMPS :

Les champs d'un Méta-IFS ressemblent à ceux d'un IFS, ils contiennent à côté des valeurs usuelles, deux pointeurs qui permettent l'accès à la liste :

* premier est le pointeur vers le premier élément de la liste, et
* dernier est le pointeur vers le dernier élément.

METHODES :

* Méta_IFS (ntrans : int; a,b,c,d,e,f :float) : Le constructeur - tous les paramètres devraient être familiers.
* Insérer (figure : Point_H&) réalise l'insertion d'un objet appartenant à la classe Point_H à la fin de la liste détenue par le Méta-IFS.
  Valeurs de retour : 1 si l'insertion a eu lieu, 0 sinon.
* Extraire (figure : Point_H&) extrait un objet de cette liste.
* Avant_Plan (figure : Point_H&) met un élément de la liste précisé par figure à la fin. Ainsi on obtient un effet 'avant plan' lors de la production de l'image par l'algorithme chaos.
  L'implémentation est élémentaire : La méthode génère successivement les messages Extraire et Insérer vers lui-même.
* Appliquer_Trans (numéro : int, figure : Point_H&) applique au domaine d'une figure quelconque une transformaion de numéro donné. La réalisation est également élémentaire : L'objet Méta-IFS demande à la figure de transmettre son domaine actuel. (Transmettre_Domaine), qui est alors modifié par la transformation. Le Méta-IFS demande une acceptation des nouvelles valeurs de la part de la figure par le message Modifier_Domaine.
  Valeurs de retour : 1 si la transformation a eu lieu, 0 sinon.
* Element_Sélectionné (x,y : int) teste si la liste du Méta-IFS contient un élément dont le domaine comprend le point (x,y). De nouveau ceci se réalise par un dialogue entre Méta-IFS et les figures de la liste.
* Transformer (ox,oy,ix,iy,jx,jy : float) transforme le domaine du Méta-IFS selon le principe décrit por la classe Point_H.
* Monter_Composition () a pour effet d'afficher les éléments de la liste qui se produisent selon deux changements de repère : Pour chaque élément, le Méta-IFS demande son domaine associé et il lui impose le changement de coordonnées déterminé par son propre repère.
* Dessiner_dét (npro : int) : Cette méthode reprend l'idée développée au point 3.3.1. Ci-après se trouve une version de l'algorithme qui se base sur l'optique "objets" :
* Génère_Trans(npro : int, ntr : int, repère) : Les paramètres sont le nombre de productions et le numéro de la transformation à appliquer au repère fourni. Ci-après se trouve une version de l'algorithme qui se base sur l'optique "objets" :

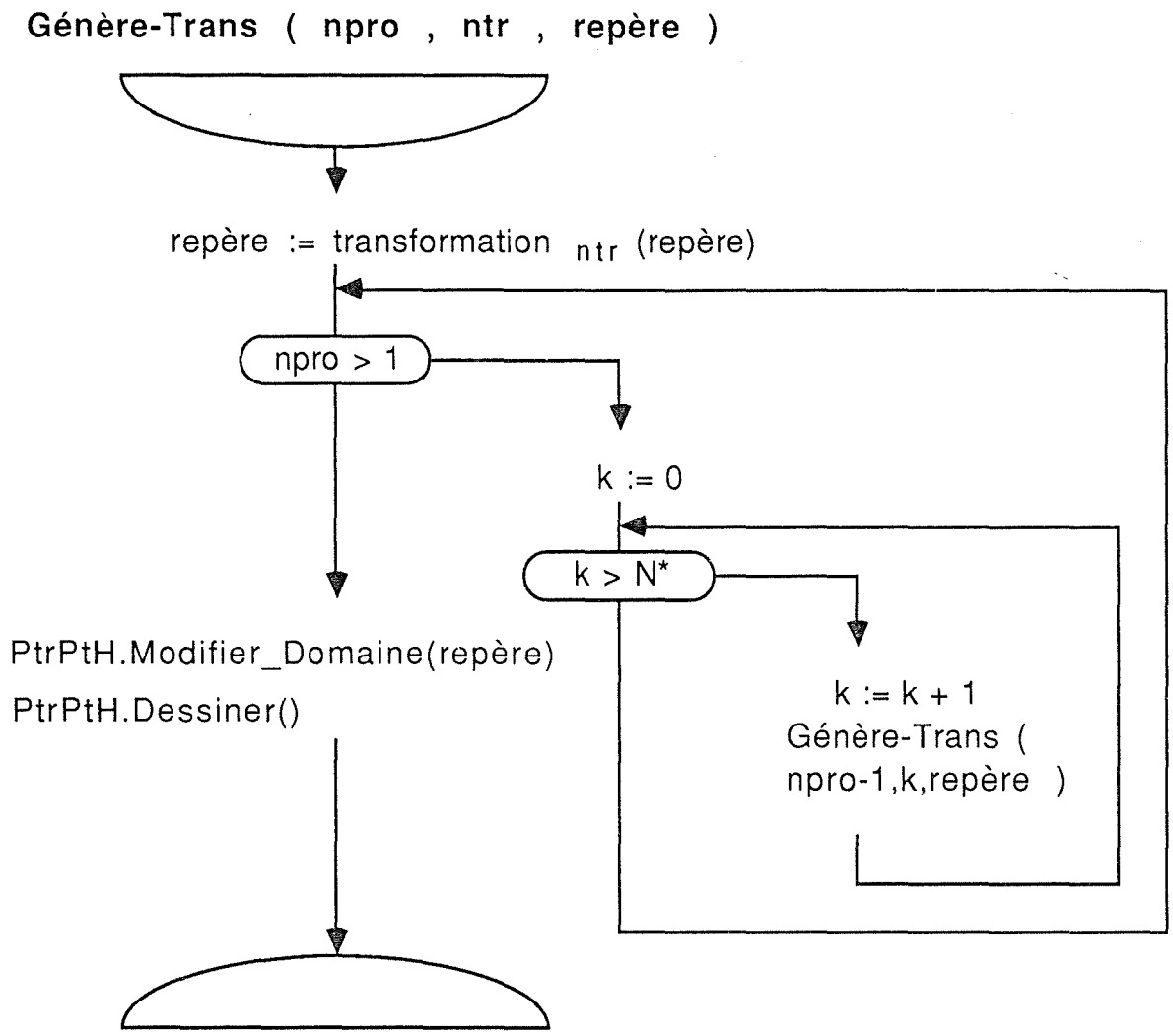


Figure 5.18. : PtrPtH est un pointeur vers un élément de la liste. Ainsi, Modifer_Domaine et Dessiner sont les méthodes propres aux éléments.

Pour compléter l'analyse du principe, présentons ici l'algorithme obtenu en éliminant la récursivité (Figure 5.19.).

Elimination des paramètres passés par valeur :

- npro par des instructions inverses
- k, repére par une pile

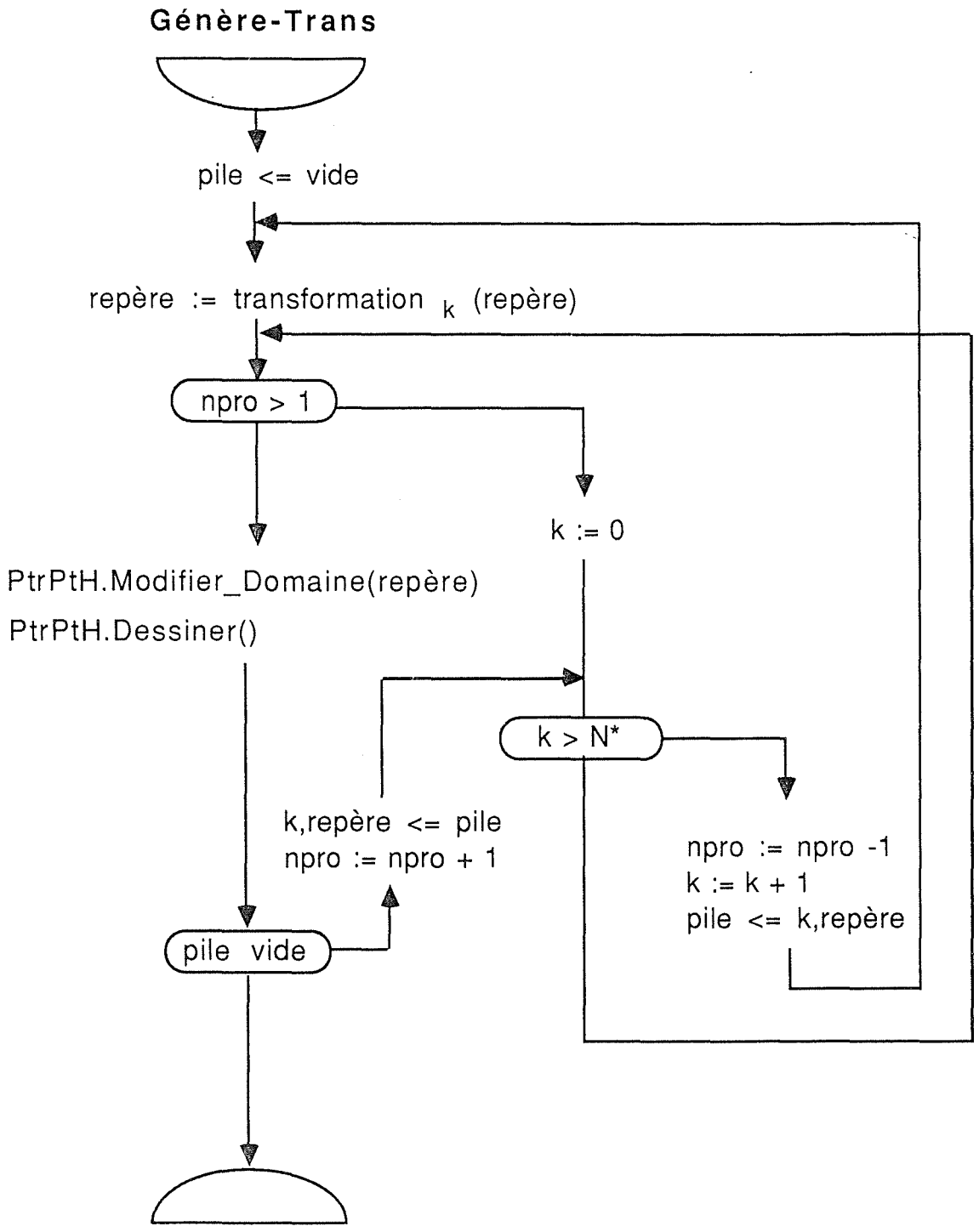


Figure 5.19. : Algorithme non-récursif.

6. L'environnement objet WINDOWS

L'environnement WINDOWS, qui a été introduit par Microsoft en 1985, est devenu l'interface graphique le plus populaire pour le MS-DOS. Plusieurs millions de licences sont vendus mondialement et une grande partie des nouvelles applications sont implémentées sous Windows.
Sans lister tous les points forts du Windows (car il y en a beaucoup), il faut préciser quelques raisons pour lesquelles j'ai choisi de programmer Windows :

* Les applications sous Windows ont une apparence similaire. Elles sont ainsi souvent plus faciles à comprendre que des programmes conventionnels sous DOS.
* La tendance actuelle pour l'implémentation d'applications est sans aucun doute l'environnement Windows. Si on ouvre une revue de software, on peut en faire le compte !
* Les applications C++ sont sans perte de performances exécutables sous Windows.
* La programmation Windows est d'un certain point de vue une programmation orientée objet :

6.1. La programmation d'un environnement objet

Lorsqu'on programme sous Windows, on est réellement confronté à un type de programmation orientée objet. L'objet le plus élémentaire qui existe sous Windows est la fenêtre, elle donne d'ailleurs le nom à Windows.

Les fenêtres sont des objets rectangulaires de l'écran. Une fenêtre reçoit des informations de la part de l'utilisateur par le clavier ou via la souris et elle imprime des outputs graphiques sur sa surface.

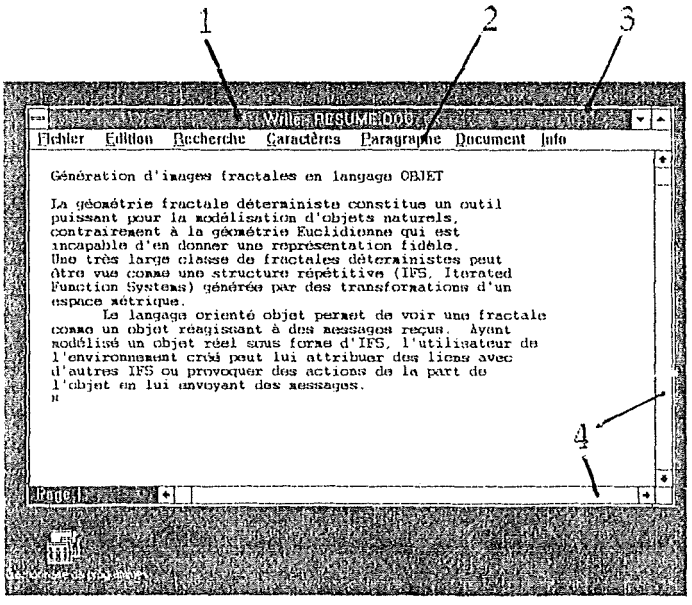


Figure 6.1.: Une application fenêtre est généralement composée d'une barre d'état qui contient le titre de l'application (1), d'une barre de menu (2), d'un cadre qui forme le bord de la fenêtre (3) et éventuellement de barres de déplacement, si le contenu d'une fenêtre dépasse la taille de l'espace d'affichage (4).

Des fenêtres auxiliaires sont des boîtes de dialogue (figure 6.2.) qui contiennent à leur tour des sous-fenêtres (des bouttons, des champs d'entrée de texte, des listes,...).

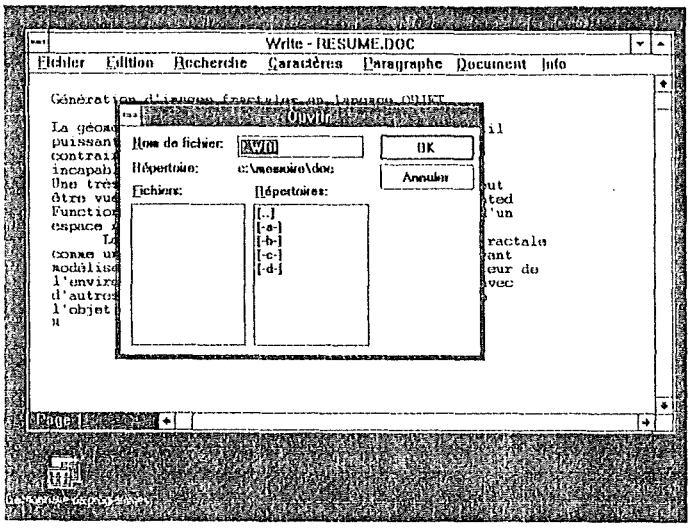


Figure 6.2.

L'utilisateur voit ces fenêtres comme objets sur l'écran et interagit directement avec ces objets en appuyant par exemple sur des boutons. D'un point de vue programmation, on peut dire la même chose : étant concepteur, on interagit avec la fenêtre par des messages. Par exemple, on indique à une boîte de dialogue qu'elle doit contenir un boutton 'OK' et un bouton 'Cancel' par les messages :

```
DEFPUSHBUTTON "OK" IDOK, 20, 168, 40, 14, WS_GROUP
DEFPUSHBUTTON "Cancel" IDCANCEL, 80,168, 40, 14, WS_GROUP
```

Gérer ces messages est la tâche principale dans la programmation sous Windows.

6.2. Don't call me, I 'll call you !

Une chose tout à fait frappante d'un tel type d'applications est que l'application s'adapte aux changements de la taille de la fenêtre. Comment cette application a-t-elle pu savoir que sa fenêtre a été changée ?!
Il se trouve que ceci est la question cruciale pour la compréhension d'architectures utilisées par des interfaces graphiques. Si un utilisateur change la taille de la fenêtre, alors Windows envoie un message **vers** le programme qui indique la nouvelle taille. Le programme peut ajuster alors ses contenus pour s'y adapter.

Normalement, ceci devrait sembler **TRES** étrange - on vient quand-même de parler de programmes et non d'un système de courrier électronique ! Comment est-il possible qu'un O.S. puisse envoyer "tout-à-coup" des messages à un programme et l'appeler ainsi ? Est-ce que le programme "ne perd pas le contrôle" ? ...

C'est C. Petzold qui pense que : "If at first you find Windows programming to be difficult, awkward, bizarrely convuled, and filled with alien concepts, rest assured this is a normal reaction. You are not alone."

Quand Windows envoie un message, il fait appel à une fonction du programme. Ses paramètres permettent de décrire le message en particulier. Cette fonction qui se trouve dans toutes les applications Windows, est la "window procedure".

La procédure - window

On est certes habitué à l'idée qu'un programme fait un appel à l'O.S., pour l'ouverture d'un fichier par exemple. Ce qui est nouveau ici est l'idée qu'un O.S. peut appeler l'application. Ceci est fondamental dans l'architecture orientée objet de Windows. Toute fenêtre qu'un programme crée possède sa procédure-window associée. Windows envoie un message vers une fenêtre en appelant la procédure-window. La procédure effectue alors un certain travail et renvoie alors le contrôle à Windows.

Souvent, ces messages informent la fenêtre d'un input de l'utilisateur par le clavier ou par la souris. C'est ainsi qu'une fenêtre de type "bouton-OK" sait qu'elle a été sélectionnée.

Comme il a été dit auparavant, un objet est formé de données et de méthodes. Une fenêtre est un objet : sa méthode est la procédure-window et ses données sont retenues par la procédure.

6.3. Le premier programme sous Windows

Il semble que, dans tous les ouvrages d'introduction à la programmation, on trouve un premier programme du type :

```
#inlude <stdio.h>
main()
{
  printf('Bonjour');
}
```

Si on essaie d'écrire un programme similaire sous Windows, il convient d'implémenter ... 72 lignes de code ! Le listing du programme se trouve ci-après :

```
/************************************************************************
Programme                              Bonjour
************************************************************************/

#include <windows.h>

long FAR PASCAL WndProc (HWND, WORD, WORD, LONG);

/*** point d'entrée du programme : ***/
int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow )
{
    /*** le nom de l'application ***/
    static char szAppName[] =  "Bonjour";
    WNDCLASS                   wndClass;
    MSG                        msg;
    HWND                       hwnd;

    /*** si auccune autre copie du programme existe : ***/
    if (!hPrevInstance)
    {
        /*** définir la fenêtre ***/
        wndClass.style         = CS_HREDRAW | CS_VREDRAW ;
        wndClass.lpfnWndProc   = WndProc;
        wndClass.cbClsExtra    = 0;
        wndClass.cbWndExtra    = 0;
        wndClass.hInstance     = hInstance;
        wndClass.hIcon         = LoadIcon(hInstance,"IDI_BON");
        wndClass.hCursor       = LoadCursor(NULL, IDC_ARROW);
        wndClass.hbrBackground = GetStockObject(WHITE_BRUSH);
        wndClass.lpszMenuName  = NULL;
        wndClass.lpszClassName = szAppName;

        RegisterClass(&wndClass) ;
    }

    /*** créer la fenêtre ***/
    hwnd = CreateWindow(szAppName,
                        "Le Programme BONJOUR",
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        NULL,
                        NULL,
                        hInstance,
                        NULL);

    /*** comment on imprime la fenêtre ***/
    ShowWindow(hwnd, nCmdShow);
    /*** garnir la fenêtre i.e.      ***/
    /*** générer le message WM_PAINT ***/
    UpdateWindow(hwnd);

    /*** boucle des messages ***/
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return msg.wParam;
}
```

```
/*** traitement des messages  ***/
long FAR PASCAL WndProc (HWND hwnd, WORD Message,
                         WORD wParam, LONG lParam)
{
    HDC                 hdc;
    PAINTSTRUCT         ps;
    RECT                rect;

    switch(Message)
        {
        case WM_PAINT:
             /*** dessiner la fenêtre ***/
             hdc = BeginPaint (hwnd, &ps);
             GetClientRect (hwnd, &rect);
             DrawText (hdc, "Bonjour",-1,&rect,DT_SINGLELINE | DT_CENTER );
             EndPaint (hwnd, &ps);
             return 0;

        case WM_DESTROY:
             /*** fermer la fenêtre ***/
             PostQuitMessage(0);
             return 0;

        }
        return DefWindowProc(hwnd,Message,wParam,lParam);
}
```

Au lieu de se poser la question de savoir pourquoi le programme est si compliqué, il faudrait plutôt se demander pourquoi l'autre programme est si simple. Par exemple, pourquoi le premier programme ne doit-il pas indiquer la position où le texte va être écrit ?

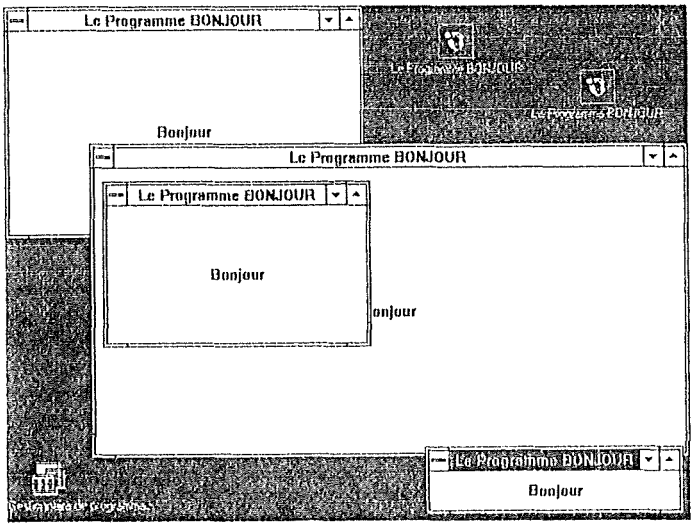


Figure 6.3. : Mais d'autre part, comment réaliser plusieurs exécutions du premier programme sans que les outputs interfèrent. Remarquons aussi qu'on peut faire un grand nombre d'opérations sur la fenêtre.

Sans entrer dans les détails de la programmation Windows (Cela prendrait 944 pages), on peut dire que tout programme possède une boucle qui attend des messages provenant d'une file d'attente gérée par Windows :

- GetMessage prend un message de la file d'attente
- TranslateMessage le traduit et
- DispatchMessage remet la main à Windows qui envoie alors un message vers la procédure-window WndProc. WndProc s'occupe alors du traitement par le programme et renvoie en terminaison le contrôle à Windows qui est en train de servir l'appel du DispatchMessage. Windows retourne à son tour le contrôle au programme et la boucle GetMessage continue.

Avec ce petit aperçu du fonctionnement d'un tel environnement, il devrait être possible de suivre les idées principales du code de l'application **'Editeur d'images fractales'** qui se trouve en annexe.

6.4. L'Editeur d'images fractales

Voici un bref aperçu des fonctions réalisables par l'application :

POPUP Edition

MENUITEM Informations : Informations sur le programme

MENUITEM Reset 100x100 : Remise à zéro de la première zone de dessin (en haut à gauche de l'écran).

MENUITEM Reset 400x400 : Remise à zéro de la deuxième zone de l'écran.

MENUITEM Reset écran : Mise à zéro des deux zones

MENUITEM Quit : Quitter l'éditeur en confirmant

POPUP Dimension

MENUITEM Ouvrir BMP : Chargement d'un bitmap dans la deuxième zone. Cette partie a été implémentée en collaboration avec Thierry Reniers.

MENUITEM Box Counting : Appliquer l'algorithme 'Box-Couting' à une zone de l'écran.

POPUP code-IFS

MENUITEM Nouveau IFS : Introduction d'un nouveau IFS.

MENUITEM Modifier IFS : Modification de l'IFS courant.

MENUITEM Choisir IFS : Choix d'un objet parmi les IFS prédéfinis (qui peut être modifié eventuellement)

MENUITEM Changer Couleurs : Changement de la couleur de l'IFS actuel en agissant sur les taux en rouge, bleu et vert.

MENUITEM Dessiner le fractal : Génération de l'attracteur sur un domaine 100x100 ou bien sur un domaine à déterminer. Pour ce faire, il s'agit de fixer l'origine, l'axei et l'axej sur la deuxième zone de l'écran.

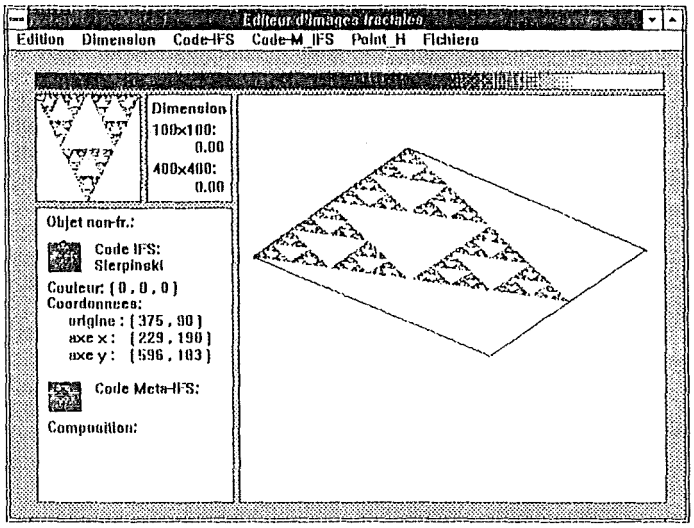


Figure 6.4. : Dessiner sur 100x100 ne nécessite pas de précisions supplémentaires. Si on souhaite l'affichage sur un domaine, le programme vérifie un dépassement de la zone de dessin. Les coordonnées du repère transformé ainsi que la couleur de l'IFS sont affichés au fur et à mesure qu'ils sont établis.

MENUITEM Zoom sur l'attracteur : Il faut choisir dans la première zone la fenêtre du zoom. Le premier point est pris en enfonçant le bouton de la souris, le deuxième est déterminé en le relâchant. Le zoom est alors reproduit sur l'entièreté de la deuxième fenêtre.

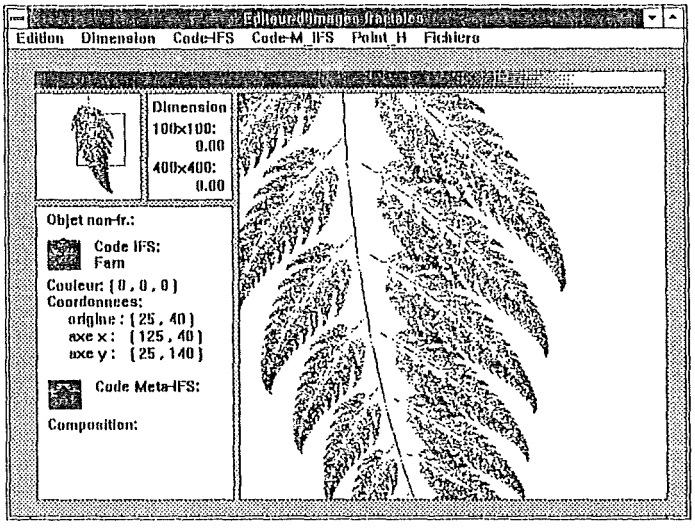


Figure 6.5. : Du fait que le zoom est affiché sur un domaine 400x400, l'algorithme nécessite un nombre très élevé d'itérations. Afin d'obtenir une image nette, la taille de la fenêtre zoom devrait dépasser le quart de la première zone de l'écran.

POPUP Code-Méta_IFS

MENUITEM Nouveau Méta-IFS : Introduction d'un Méta-IFS.

MENUITEM Modifier Méta-IFS : Modification du Méta-IFS courant.

MENUITEM Choisir Méta-IFS : Choix d'un objet parmi les Méta-IFS prédéfinis.

MENUITEM Insérer Pt_H : Insertion d'une figure non fractale dans la liste constituant le Méta-IFS.

MENUITEM Insérer IFS : Insertion d'un IFS dans l'ensemble des IFS dominés du Méta-IFS courant.

MENUITEM Appliquer tr. à non-fractal / IFS : On applique une transformation du Méta-IFS à une figure (ou un IFS), ce qui veut dire qu'on déplace le domaine de la figure selon cette transformation. En insérant ensuite cet élément dans la liste, on peut obtenir une composition dont les éléments sont bien distingués. Un exemple (figure 6.6. première zone de dessin) montre l'avantage d'un tel traitement préalable à l'insertion : La première transformation a été appliquée à l'objet non-fractal 'ligne' avant l'insertion, et les deux autres transformations ont été effectuées sur les deux objets de type feuille qui ont été insérés ensuite.

MENUITEM Montrer les composants : C'est ici que le Méta-IFS montre la liste des figures qu'il contient actuellement.

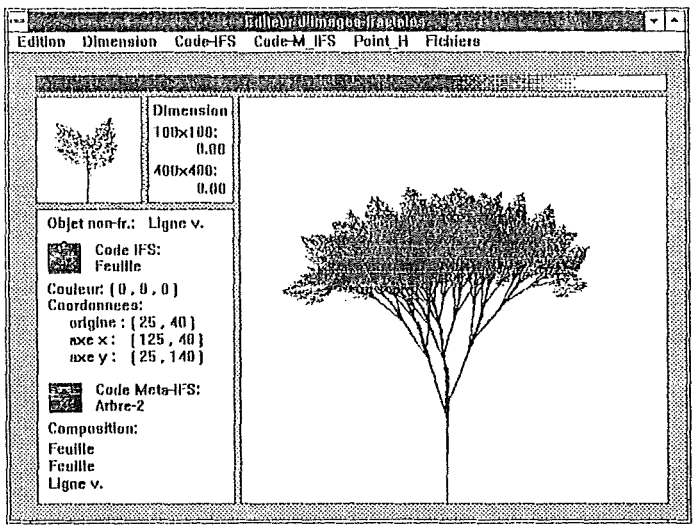


Figure 6.6. : La composition du Méta-IFS (l'objet non-fractal et les attracteurs dominés) est affichée en indiquant les noms des objets. Elle peut être visualisée graphiquement dans la première zone de l'écran en sélectionnant 'Montrer les composants' du menu 'Code-Méta_IFS'.

MENUITEM Dessiner le fractal : Génération de l'attracteur du Méta-IFS sur un domaine 400x400 ou bien sur un domaine à déterminer. De nouveau, il s'agit de fixer l'origine, l'axei et l'axej sur la deuxième zone de l'écran.

MENUITEM Zoom sur l'attracteur : Un zoom est fait sur l'attracteur.

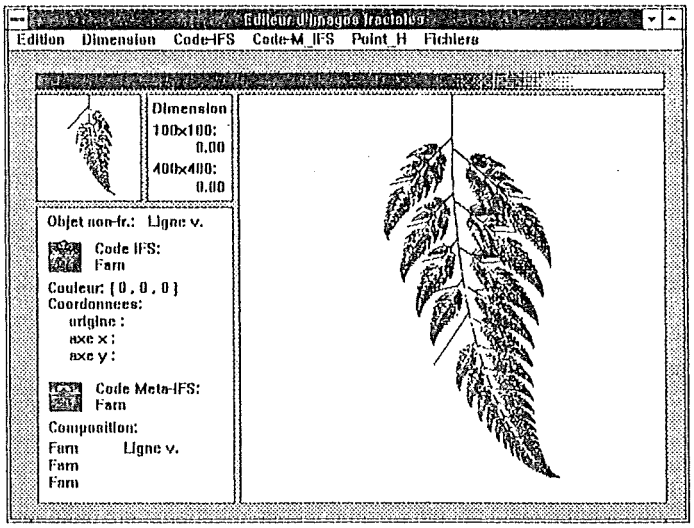


Figure 6.7. : Le developpement d'"anomalies" dans une formule fractale peut être simulé. Voici la démarche suivie :
* Méta-IFS : Farn
* IFS : Farn (coloré)
* Appliquer tr. 0 à l'IFS Farn
* Insérer IFS dans Méta-IFS
* Appliquer tr 1 à l'IFS Farn
* Insérer IFS dans Méta-IFS
* Appliquer tr 3 à l'IFS Farn
* Insérer IFS dans Méta-IFS
* Figure : Ligne verticale
* Insérer Figure dans Méta-IFS
* Appliquer tr 1 à l'IFS Farn
* On peut alors dessiner le Méta-IFS avec un nombre de reproductions variable pour suivre la "progression" de la ligne.

POPUP Figures

MENUITEM Choisir objet : Choix d'une figure non-fractale.

MENUITEM Dessiner Figure : Afficher l'objet.

MENUITEM Changer attributs : Changements de la couleur.

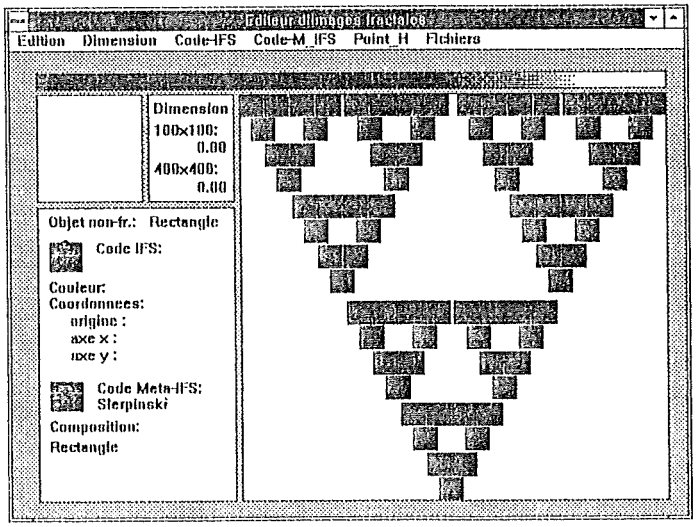


Figure 6.8. : La généralisation du Méta-IFS aux figures quelconques permet de simuler l'algorithme déterministe. Le dessin est équivalent à la figure 2.2.

Voici quelques formules fractales qui sont utilisées par les programmes et en particulier par l'Editeur. Ces tableaux peuvent s'avérer très utiles lors d'un choix 'Appliquer transformation à un IFS/non-fractal' :

Triangle de Sierpinski :

ntr.	a	b	c	d	e	f	p
0	0.49	0	0	0.49	0	0	0.3
1	0.49	0	0	0.49	51	0	0.2
2	0.49	0	0	0.49	25	51	0.5

Rectangle :

ntr.	a	b	c	d	e	f	p
0	0.49	0	0	0.49	0	0	0.3
1	0.49	0	0	0.49	51	0	0.2
2	0.49	0	0	0.49	0	51	0.2
3	0.49	0	0	0.49	51	51	0.2

Farn :

ntr.	a	b	c	d	e	f	p
0	-0.15	0.28	0.26	0.24	58	0.44	0.09
1	0.85	0.04	-0.04	0.85	8	18	0.8
2	0.2	-0.21	0.18	0.22	40	1.6	0.09
3	0	0	0	0.16	50	0	0.02

Arbre - 1 :

ntr.	a	b	c	d	e	f	p
0	0.01	0	0	0.45	48	55	0.1
1	0.55	-0.24	0.24	0.65	46	-1	0.4
2	0.6	0.05	-0.05	0.6	14.8	20	0.5

Arbre - 2 :

ntr.	a	b	c	d	e	f	p
0	0.01	0	0	0.45	48	55	0.1
1	0.55	-0.24	0.24	0.65	46	-1	0.3
2	0.7	0.2	-0.2	0.7	-5.5	19	0.3
3	0.6	0.05	-0.05	0.6	14.8	20	0.3

Feuille :

ntr.	a	b	c	d	e	f	p
0	0.44	0.32	-0.07	0.61	0	46	0.4
1	-0.82	0.16	-0.16	0.81	88	10	0.6

Annexes

Annexe A : Algorithme déterministe implémenté en PASCAL

Annexe B : Algorithme chaos implémenté en PASCAL

Annexe C : Algorithme chaos avec coloriage implémenté en PASCAL

Annexe D : Une première implémentation objet d'un IFS : la classe Random-Fractal avec constructeur et méthode Dessiner().

Annexe E : Listing du programme C++ contenant les classes qui définissent les fractales.

Annexe F : Listing de l'application objet 'Editeur d'images fractales' :

- FR.H contient les types prédéfinis
- FR.RC est le fichier des ressources qui donne une définition des menus et des Dialogues utilisés par le programme.
- FR.CPP est le programme principal utilisant les classes de WINFRACT pour la production des fractales.

Annexe G : Description des programmes fournis.

Annexe A :


```
{*************************************************************************}
 program algorithme_det;
{*************************************************************************}

{  Un exemple d'algorithme deterministe

   VARIABLES:
   s       : (B) matrice 100x100 contenant la transformation des pixels
             contenus dans t
   t       : (B) matrice 100x100 qui sert de 'bitmap'
   a..f    : (R) tableaux qui contiennent les transformations
   inc     : (B) le nombre de copiages (i.e. applications des transformations)
   i,j,dec : (B) compteurs de boucle, variables auxiliaires
   aux     : (C) variable auxiliaire
   gm,gp   : (I) pour l'intialisation du graphe


   USES:
   crt,dos,graph
}

uses crt,dos,graph;

var s           : array[1..100,1..100] of byte;
    t           : array[1..100,1..100] of byte;
    a,b,c,d,e,f : array[1..4] of real;

    gm,gp       : integer;
    inc,i,j,dec : byte;
    aux         : char;

begin
      {**** initialiser le code-IFS ****}
      a[1]:=0.5; b[1]:=  0;     e[1]:= 1;
      c[1]:=  0; d[1]:=0.5;     f[1]:= 1;
      a[2]:=0.5; b[2]:=  0;     e[2]:=50;
      c[2]:=  0; d[2]:=0.5;     f[2]:= 1;
      a[3]:=0.5; b[3]:=  0;     e[3]:=25;
      c[3]:=  0; d[3]:=0.5;     f[3]:=50;

      {**** initialiser ensemble de départ (carré) ****}
      for i:=1 to 100 do for j:=1 to 100 do
          begin t[i,j]:=0; s[i,j]:=0; end;
      dec :=1;
      for i:=1 to 50 do begin
          t[i,1]:=1;     t[i,dec]:=1;     t[i,dec+1]:=1;
          t[101-i,1]:=1; t[101-i,dec]:=1; t[101-i,dec+1]:=1;
          dec := dec+2;
      end;

      {**** initialiser graph ****}
      gp := detect;
      initgraph(gp,gm,'');
      if graphresult< grOK then
         halt(1);

  {*** appliquer 5 fois ***}
  for inc:=0 to 5 do begin

      {**** appliquer transformation affine -> s ****}
      for i:=1 to 100 do begin
          for j:=1 to 100 do begin

              if t[i,j]<>0 then begin
                 s[trunc(a[1]*i+b[1]*j+e[1]),trunc(c[1]*i+d[1]*j+f[1])] := 1;
                 s[trunc(a[2]*i+b[2]*j+e[2]),trunc(c[2]*i+d[2]*j+f[2])] := 1;
```

```
                    s[trunc(a[3]*i+b[3]*j+e[3]),trunc(c[3]*i+d[3]*j+f[3])] := 1;
                end;
            end;
        end;

        {**** transferer resultat de s en t et dessiner t ****}
        for i:=1 to 100 do begin
            for j:=1 to 100 do begin
                t[i,j] := s[i,j];
                s[i,j] := 0;
                if t[i,j]<>0 then
                    putpixel(inc*100+i,j,white);
            end;
        end;

 end;
 read(aux);
 closegraph;
end.
```

Annexe B :


```
{*************************************************************************}
 program algorithme_random;
{*************************************************************************}

{  Un exemple d'algorithme-chaos

   VARIABLES:
   a..f     : (R) tableaux qui contiennent les transformations
   p        : (R) tableau qui contient les probibilités associées aux
                  transformations
   x,y      : (R) les coordonnées du point actuel
   nx,ny    : (R) les coordonnées du point suivant
   i,t,aux,gm,gp sont des variables auxiliaires

   LABEL:
   1        : jump lors de la génération des probabilités

   USES:
   crt,dos,graph

}

uses crt,dos,graph;
label 1;

var a,b,c,d,e,f,p    : array[1..4] of real;
    aux              : char;
    gm,gp,i,t        : integer;
    nx,ny,x,y,trans  : real;

begin
      {*** initialiser le code-IFS pour un FARN ***}
      a[1]:=-0.15; b[1]:= 0.28;    e[1]:= 0;
      c[1]:= 0.26; d[1]:= 0.24;    f[1]:=20;    p[1]:=0.09;
      a[2]:= 0.85; b[2]:= 0.04;    e[2]:= 0;
      c[2]:=-0.04; d[2]:= 0.85;    f[2]:=72;    p[2]:=0.80;
      a[3]:= 0.20; b[3]:=-0.21;    e[3]:= 0;
      c[3]:= 0.18; d[3]:= 0.22;    f[3]:=72;    p[3]:=0.09;
      a[4]:= 0.00; b[4]:= 0.00;    e[4]:= 0;
      c[4]:= 0.00; d[4]:= 0.16;    f[4]:= 0;    p[4]:=0.02;

      {*** initialiser graph ***}
      gp := detect;
      initgraph(gp,gm,'');
      if graphresult< grOK then
         halt(1);
      randomize;

      {**** point de départ ****}
      y:=10;x:=10;

      {**** appliquer les trnasformations ****}
      for i:=1 to 20000 do begin

         {**** dessiner le pixel ****}
         if i>10 then
         putpixel(trunc(x)+200,480-trunc(y),10);

         {**** choisir une des transformations ****}
         trans := random;
         if trans <= p[1]              then begin t:=1; goto 1; end;
         if trans <= (p[1]+p[2])       then begin t:=2; goto 1; end;
         if trans <= (p[1]+p[2]+p[3]) then begin t:=3; goto 1; end
                                       else t:=4;

         {**** appliquer la transformation affine ****}
```

```
1:          nx := (a[t]*(x-10)+b[t]*y+e[t]+10);
            ny := (c[t]*(x-10)+d[t]*y+f[t]);
            x:=nx;
            y:=ny;

        end;
 readln(aux);
 closegraph;
end.
```

```
{*****************************************************************************}
 program algorithme_random_couleurs;
{*****************************************************************************}

{  Un exemple d'algorithme-chaos avec bitmap-couleurs

   VARIABLES:
   a..f          : (R) tableaux qui contiennent les transformations
   p             : (R) tableau qui contient les probibilités associées aux
                       transformations
   bitmap        : (B) le bitmap contenant le nombre de visites du pixel
   color1        : (B) tableau qui est le set de couleurs
   mag           : (R) le zoom sur l'attracteur ( 1 = taille normale )
   x,y           : (R) les coordonnées du point actuel
   nx,ny         : (R) les coordonnées du point suivant
   numits        : (I) le nombre d'itérations
   col           : (I) la couleur (élément de colori) à imprimer
   factor        : (R) le facteur "d'éclairage" de l'image
   scaling,i,j,t,aux,gm,gp sont des variables auxiliaires

   LABEL:
   1       : jump lors de la génération des probabilités

   USES:
   crt,dos,graph
}

uses crt,dos,graph;

label 1;

var a,b,c,d,e,f,p              : array[1..4] of real;
    bitmap                     : array[1..160,1..380] of byte;
    color1                     : array [1..6] of byte;
    mag,factor,x,y,nx,ny       : real;
    numits,col                 : integer;

    {*** variables auxiliaires ***}
    aux : string; gm,gp,dec : integer; t,i,j,ni,nj : integer;
    scaling,trans : real;

begin
      {*** changer le facteur de colorage permet de          ***
       *** passer d'une image 'claire' à une image 'sombre' ***}
      factor := 7000;

      {*** changer magnitude ou nombre d'itérations n'a  ***
       *** auccun effet sur le colorage !                ***}
      numits := 10000;
      mag := 1.1;

      {*** initialiser set de couleurs ***}
      color1[1]:=02; color1[2]:=02; color1[3]:=03;
      color1[4]:=10; color1[5]:=11; color1[6]:=14;

      {*** initialiser le code-IFS pour un FARN ***}
      a[1]:=-0.15; b[1]:= 0.28;    e[1]:= 0;
      c[1]:= 0.26; d[1]:= 0.24;    f[1]:= 4;    p[1]:=0.09;
      a[2]:= 0.85; b[2]:= 0.04;    e[2]:= 0;
      c[2]:=-0.04; d[2]:= 0.85;    f[2]:=14;    p[2]:=0.80;
      a[3]:= 0.20; b[3]:=-0.21;    e[3]:= 0;
      c[3]:= 0.18; d[3]:= 0.22;    f[3]:=14;    p[3]:=0.09;
      a[4]:= 0.00; b[4]:= 0.00;    e[4]:= 0;
      c[4]:= 0.00; d[4]:= 0.16;    f[4]:= 0;    p[4]:=0.02;

      {**** initialiser graph ****}
```

```
      gp := detect;
      initgraph(gp,gm,'');
      if graphresult< grOK then
         halt(1);
      randomize;

      {**** initialiser le bitmap ****}
      for i:=1 to 160 do for j:=1 to 380 do
          bitmap[i,j]:=0;

      {**** point de départ ****}
      y:=10;x:=10;

      {**** appliquer les transfarmations ****}
      for i:=1 to numits do begin

         {**** choisir une des transformations ****}
         trans := random;
         if trans <= p[1]           then begin t:=1; goto 1; end;
         if trans <= p[1]+p[2]      then begin t:=2; goto 1; end;
         if trans <= p[1]+p[2]+p[3] then begin t:=3; goto 1; end;
         t:=4;


         {**** appliquer transformation affine      ****}
         {**** et ajouter le point obtenu au bitmap ****}
1:       nx := (a[t]*(x-10)+b[t]*y+e[t]+10);
         ny := (c[t]*(x-10)+d[t]*y+f[t]);
         ni := 50+trunc(mag*nx);
         nj := trunc(mag*ny);
         if ((ni>0)and(ni<=160))and((nj>0)and(nj<=380))
            then
            bitmap[ni,nj]:=bitmap[ni,nj]+1;
         x:=nx;
         y:=ny;
      end;
      {*** imprimer le bitmap avec les  ***}
      {*** trois sets de couleurs       ***}
      scaling := factor*mag*mag/numits;
      for i:=1 to 160 do
           for j:=1 to 380 do
               if (bitmap[i,j]>0) then begin
                  col := trunc (bitmap[i,j]*scaling);
                  if col>6 then col:=6;
                  putpixel(160+i,400-j,color1[col]);
 end;
 readln(aux);
 closegraph;
end.
```

Annexe D :

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <time.h>

class Random_Fractal
{
 private :
    int ntrans;                    // le nombre de transformations de l'objet
    float ta[10],tb[10],tc[10],    // les
          td[10],te[10],tf[10];    // transformations
    float probab[10];              // avec leurs probabilités
 public :
    Random_Fractal(int,float*, float*, float*,
                   float*, float*, float*, float*);   // le constructeur
    void Dessiner(int);                               // dessiner sur écran

};

/*****************************************************************
 * Constructeur de la classe                                     *
 *****************************************************************/
 Random_Fractal::Random_Fractal(int nt, float *a,float *b,float *c,
                                        float *d,float *e,float *f,
                                        float *p)

 {
  for (int j = 0; j < nt; ++j)
       {
       probab[j] = p[j];
       ta[j] = a[j];
       tb[j] = b[j];
       tc[j] = c[j];
       td[j] = d[j];
       te[j] = e[j];
       tf[j] = f[j];
       }
  ntrans = nt;
 };

/*****************************************************************
 * fonction qui génère le 'random iteration algorithm'           *
 *****************************************************************/
 void Random_Fractal::Dessiner(int numits)

/* un premier algorithme-chaos
   VARIABLES:
   numt      : (I) le numéro courant de la transformation
   x,y       : (R) les coordonnées du point actuel
   newx,newy : (R) les coordonnées du point suivant
   et des variables auxiliaires
*/
{
  float x=0, y=0, newx=0, newy=0;
  register int i;
  int numt,rnd;
  float proba;

  randomize();
  for (i=1; i <= numits; ++i)
       {
       /*** dessiner le pixel ***/
       if (i>10)
```

```
        putpixel((int)x,(int)y,10);

      /*** choix d'une des transformations ***/
      rnd = random(100);
      numt=0;
      proba=probab[0]*100.0;
      while (rnd>proba)
          {
          ++numt;
          proba += probab[numt]*100.0;
          }

      /*** appliquer la transformation ***/
      newx = ta[numt]*x + tb[numt]*y + te[numt] ;
      newy = tc[numt]*x + td[numt]*y + tf[numt] ;
      x = newx;
      y = newy;
      }
};


main()
{
 /*** code d'un farn ***/
 float fa[4] = {-0.15, 0.85,  0.2,    0},
       fb[4] = { 0.28, 0.04,-0.21,    0},
       fc[4] = { 0.26,-0.04, 0.18,    0},
       fd[4] = { 0.24, 0.85, 0.22, 0.16},
       fe[4] = {    0,    0,    0,    0},
       ff[4] = {   11,   40,   40,    0},
       fp[4] = { 0.06, 0.85, 0.06, 0.03};

 /*** code du traingle de Sierpinski  ***/
 float sa[3] = { 0.50, 0.50, 0.50},
       sb[3] = {    0,    0,    0},
       sc[3] = {    0,    0,    0},
       sd[3] = { 0.50, 0.50, 0.50},
       se[3] = {    0,  320,  160},
       sf[3] = {    0,    0,  240},
       sp[3] = { 0.20, 0.30, 0.50};

 /*** initialiser le mode graphique ***/
 int graphdriver = DETECT , graphmode;
 initgraph (&graphdriver,&graphmode,"c:\\borlandc\\bgi");

 /*** construire les deux objets Fractales ***/
 Random_Fractal Farn(4,fa,fb,fc,fd,fe,ff,fp);
 Random_Fractal Sierpinski(3,sa,sb,sc,sd,se,sf,sp);


 /*** dessiner les objets ***/
 Farn.Dessiner( 5000);
 Sierpinski.Dessiner(15000);
}
```

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

class Point_H;

/*****************************************************************************/
/******************** LA CLASSE DEFINISSANT LE META-IFS **********************/
/*****************************************************************************/
 class Meta_ifs
{

 public :
   Meta_ifs();                  // constructeur - initialisation des pointeurs
   Meta_ifs(int,float*,float*   // constructeur - initialisation des pointeurs
            ,float*,float*      // et de plusieures transformations du
            ,float*,float*);    // méta-ifs.
   Meta_ifs(int,float*,float*   // constructeur - initialisation des pointeurs
            ,float*,float*      // et de plusieures transformations du
            ,float*,float*      // méta-ifs.
            ,float*,int);
   int  Inserer(Point_H&);      // insertion d'un élément (point de Hausdorff)
   void Extraire(Point_H&);     // extraction d'un élément de la liste
   void Avant_Plan(Point_H&);   // mettre l'objet en avant-plan
   void Montrer_Composition(); // montrer la composition du méta-ifs
   void Dessiner(int);          // dessiner le méta-IFS - principe déterministe
   void Dessiner_Chaos(int);    // dessiner le méta-IFS - principe chaos
   int  Applic_Trans(int, Point_H&); // on applique une transformation du
                                     // meta-ifs au domaine du point de H.
   void Transformer(float,float,float,float,float,float);
   int  Element_Selectionne(int,int);
   Point_H* PtrDernier()        // donner le dernier élément de la liste
         {return Dernier;};     //

 protected :
   Point_H *Premier; // le pointeur vers le premier élément du méta-ifs et
   Point_H *Dernier; // le pointeur vers le dernier élément de la liste

 private :
   int ntrans_m;                          // le nombre de transformations
   float ta_m[5],tb_m[5],tc_m[5],         // et les transformations
         td_m[5],te_m[5],tf_m[5],         //
         pr_m[5];                         //
   float orx,ory,                         // le systéme de coordonnées relatif,
         aix,aiy,                         // càd. l'origine, l'axe i et j.
         ajx,ajy;                         //
                                          //
   int npro;                              // variable globale
   int precision;                         //
   Point_H* PtrFigure;                    //
   void Genere_Trans(int);                // génération récursive des transf.

};
```

```
/*****************************************************************************/
/*************** LA CLASSE DEFINISSANT UN POINT DE H(X) ***********************/
/*****************************************************************************/
 class Point_H
{

 protected :

   Point_H();                   // constructeur à valeurs par
                                // défaut
   float originex,originey,     // le systéme de coordonnées relatif,
         axeix,axeiy,           // càd. l'origine, l'axe i et j.
         axejx,axejy;           //

 private :
   Point_H *Suivant;            // le pointeur vers l'élément suivant

 public :

   int  Selectionne(int,int);
   int  Modifier_Domaine(float,float,float,
                         float,float,float);   // choix du domaine
   void Transmettre_Domaine(float&,float&,float&,
                            float&,float&,float&);
   void Tronquer();
   void ModifElemSuivant(Point_H* Element)     // faire pointer 'Suivant' sur
                        {Suivant = Element;};  // le nouvel élément
   Point_H* ElemSuivant() {return Suivant;};   // donner l'élément suivant de
                                               // la liste
   virtual void Dessiner() = 0;                // dessiner un élément de la
                                               // liste (redéfinie)
   virtual void Genere_PtD(float&,float&,int&)=0;// générer un point au hasard
};                                               // (redéfinie)

/*****************************************************************************/
/************** LA CLASSE DEFINISSANT UN OBJET  << LIGNE >>  ******************/
/*****************************************************************************/
class Ligne : public Point_H
{
 private :
   int couleur;                           // la couleur associée
   char type;                             // les deux points extrêmes du segment
 public :                                 //
   Ligne(int,char);                       // constructeur de l'objet
   void Dessiner();                       // dessiner la ligne
   void Genere_PtD(float&,float&,int&); // génération d'un point de la ligne
   Ligne* Clone();                        // produire une copie
};

/*****************************************************************************/
/************ LA CLASSE DEFINISSANT UN OBJET  << RECTANGLE >>  ***************/
/*****************************************************************************/
class Rect : public Point_H
{
 private :
   int couleur;                           // la couleur du rectangle
                                          //
 public :                                 //
   Rect(int);                             // son constructeur
   void Dessiner();                       // dessiner le rectangle
   void Genere_PtD(float&,float&,int&); // génération d'un point du rectangle
   Rect* Clone();                         // produire une copie
};
```

```
/*************************************************************************/
/************ LA CLASSE DEFINISSANT UN OBJET  << FRACTAL >>  **************/
/*************************************************************************/
class ifs : public Point_H
{
 protected :
    int ntrans;                     // le nombre de transformations de l'objet
    float ta[10],tb[10],tc[10],     // et les
          td[10],te[10],tf[10];     // transformations
    float probab[10];               // avec leurs probabilités
    float ptx,pty;                  // le point dançant
    int precision;                  // (1 à 10) la précision du dessin
    int qualite;                    // 1  : bonne qualité des transformations
                                    // 2  : qualité non-connue (à tester)
                                    // >2 : mauvaise q. (dépassement du domaine)
 public :                           //
    int couleur;                    // sa couleur
    ifs (int,float*, float*, float*, float*,
         float*, float*, float*,int,int,int); // le constructeur
    void Dessiner();                          // dessiner avec domaine
    void Genere_PtD(float&,float&,int&);      // générer un pt de l'attracteur
    void Encadrer();                          // dessiner le domaine
    int  Zoom(int,int,int,int);               // zoom dessiné sur le dom.
    ifs* Clone();                             // produire une copie

};

/*************************************************************************/
/********** LA CLASSE DEFINISSANT UN OBJET  << FRACTAL COLORE >>  **********/
/*************************************************************************/
class ifs_Couleur : public ifs
{
 private :                // ENRICHISSEMENT de ifs :
    int brillance;        // clareté de l'image, 1(sombre) à 20(image claire)
    int ncouleurs;        // le nombre de nuances de couleurs
    int couleur[15];      // l'ensemble des couleurs utilisées
    int color_scaling;    // indicateur si le tableau à été initialisé
    int bitmap[100][100]; // tableau contenant les pixels colorés

 public :
    ifs_Couleur   (int,float*, float*, float*,          // le constructeur
                   float*, float*, float*, float*,      // avec les 3 para-
                   int, int, int, int*);                // metres en plus
    void Dessiner();                                    // dessiner en couleur
    void Mirroir(char);                                 // effets spéciaux

 private :
    void Bmp_Init();                                    // initialisation BMP
};

/*=========================================================================*/
/*================= METHODES DE LA CLASSE  << META_IFS >>  ================*/
/*=========================================================================*/

/*-----------------------------------------------------*/
/*------  Le constructeur de la classe Méta-ifs  ------*/
/*-----------------------------------------------------*/
Meta_ifs::Meta_ifs() // constructeur de la liste vide
{
  Premier = 0;
  Dernier = 0;
  ntrans_m = 1;
  ta_m[0]=1;
  tb_m[0]=0;
  tc_m[0]=0;
  td_m[0]=1;
  te_m[0]=0;
  orx = 0;
  ory = 0;
  aix = 100;
  aiy = 0;
  ajx = 0;
  ajy = 100;
};

/*-----------------------------------------------------*/
/*------  Le constructeur de la classe Méta-ifs  ------*/
/*-----------------------------------------------------*/
Meta_ifs::Meta_ifs(int nt, float *a,float *b,float *c,
                   float *d,float *e,float *f)
{
  Premier = 0;
  Dernier = 0;
  for (int j = 0; j < nt; ++j)
      {
      ta_m[j] = a[j];
      tb_m[j] = b[j];
      tc_m[j] = c[j];
      td_m[j] = d[j];
      te_m[j] = e[j];
      tf_m[j] = f[j];
      pr_m[j] = 1/(nt*1.0);
      }
  ntrans_m = nt;
  orx = 0;
  ory = 0;
  aix = 100;
  aiy = 0;
  ajx = 0;
  ajy = 100;
  precision = 1;
};
/*-----------------------------------------------------*/
/*------  Le constructeur de la classe Méta-ifs  ------*/
/*-----------------------------------------------------*/
Meta_ifs::Meta_ifs(int nt, float *a,float *b,float *c,
                   float *d,float *e,float *f,float *p,int prec)
{
  Premier = 0;
  Dernier = 0;
  for (int j = 0; j < nt; ++j)
      {
      ta_m[j] = a[j];
      tb_m[j] = b[j];
      tc_m[j] = c[j];
      td_m[j] = d[j];
      te_m[j] = e[j];
      tf_m[j] = f[j];
      pr_m[j] = p[j];
      }
  ntrans_m = nt;
  orx = 0;
  ory = 0;
  aix = 100;
  aiy = 0;
  ajx = 0;
  ajy = 100;
  precision = prec;
};
```

```
/*-------------------------------------------------------*/
/*------  Modification du domaine d'un méta-ifs  ------*/
/*-------------------------------------------------------*/
void Meta_ifs::Transformer (float ox,float oy,float ix,
                            float iy,float jx,float jy)
{
  orx = ox;
  ory = oy;
  aix = ix;
  aiy = iy;
  ajx = jx;
  ajy = jy;
};

/*-------------------------------------------------------*/
/*--  Méthode qui montre la composition du méta-ifs  --*/
/*-------------------------------------------------------*/
void Meta_ifs::Montrer_Composition()
{
  Point_H* PtrFigure;
  float ox,oy,ix,iy,jx,jy;

  for(PtrFigure=Premier; PtrFigure!=0; PtrFigure=(*PtrFigure).ElemSuivant())
    {

    PtrFigure->Transmettre_Domaine(ox,oy,ix,iy,jx,jy);
    PtrFigure->Modifier_Domaine(
                      (aix-orx)*ox/100.0 + (ajx-orx)*oy/100.0 + orx,
                      (aiy-ory)*ox/100.0 + (ajy-ory)*oy/100.0 + ory,
                      (aix-orx)*ix/100.0 + (ajx-orx)*iy/100.0 + orx,
                      (aiy-ory)*ix/100.0 + (ajy-ory)*iy/100.0 + ory,
                      (aix-orx)*jx/100.0 + (ajx-orx)*jy/100.0 + orx,
                      (aiy-ory)*jx/100.0 + (ajy-ory)*jy/100.0 + ory);
    PtrFigure->Dessiner();
    PtrFigure->Modifier_Domaine(ox,oy,ix,iy,jx,jy);
    };
};

/*-------------------------------------------------------*/
/*-----  Méthode qui dessine le méta-ifs/chaos  -------*/
/*-------------------------------------------------------*/
void Meta_ifs::Dessiner_Chaos(int np)
{
  float newx,newy,x,y,px,py,rnd,proba;
  unsigned long int numits;
  float normei,normej, psij,aired,airec,angle,numit;
  int numt,nbelts;
  register int i,j,c;

  /*** calcul de l'aire occupée par le domaine ***/
  normei = sqrt ( (aix-orx)*(aix-orx) + (aiy-ory)*(aiy-ory) );
  normej = sqrt ( (ajx-orx)*(ajx-orx) + (ajy-ory)*(ajy-ory) );
  psij   = (aix-orx)*(ajx-orx) + (aiy-ory)*(ajy-ory);
  if ( ((normei*normej)!=0))
     {
     if( ((psij/(normei*normej))<1)&&
         ((psij/(normei*normej))>-1)   )
         {
         angle =  acos (psij/(normei*normej));
         }
     }
  else
     angle = 0;

  aired =  (normei*normej*sin(angle));

  /*** pour déterminer le nombre d'itérations ***/
  /*** necessaires pour la précision demandée ***/
  airec = 10000.0;

  numit = precision * (aired/airec);
  if (numit<64)
     numits = (int) (numit * 1000);
  else
     numits = 64000;
  randomize();


  /*** pour chaque élément de la liste : ***/
  nbelts=0;
  for (PtrFigure = Premier; PtrFigure != 0;
       PtrFigure = PtrFigure->ElemSuivant())
       {
       PtrFigure->Tronquer();
       nbelts += 1;
       };
  if (nbelts!=0) numits = numits/(nbelts*1.0);

  randomize();
  /*** appliquer l'algorithme ***/
  for (j = 0; j < numits; j++)
      {
      for (PtrFigure = Premier; PtrFigure != 0;
           PtrFigure = PtrFigure->ElemSuivant())
           {
           PtrFigure->Genere_PtD(x,y,c);

           for (i=1; i <= np; ++i)
               {
               /*** choix d'une des transformations ***/
               rnd = random(100);
               numt=0;
               proba=pr_m[0]*100.0;
               while (rnd>proba)
                     {
                     ++numt;
                     proba += pr_m[numt]*100.0;
                     }

               /*** appliquer la transformation ***/
               newx = ta_m[numt]*x + tb_m[numt]*y + te_m[numt] ;
               newy = tc_m[numt]*x + td_m[numt]*y + tf_m[numt] ;
               x = newx;
               y = newy;
               };

           if ((x>=0)&&(x<=100)&&(y>=0)&&(y<=100))
              {
              px = (aix-orx)*x/100.0 + (ajx-orx)*y/100.0 + orx;
              py = (aiy-ory)*x/100.0 + (ajy-ory)*y/100.0 + ory;
              putpixel((int)px,(int)py,c);
              };
           };
      };
};

/*-------------------------------------------------------*/
/*-------  Algorithme récursif pour générer les  ------*/
/*-------  transformations du méta-ifs           ------*/
/*-------------------------------------------------------*/
void Meta_ifs::Genere_Trans(int notr)
  {
```

```
    float ox,oy,ix,iy,jx,jy;

    PtrFigure->Transmettre_Domaine(ox,oy,ix,iy,jx,jy);
    /*** appliquer la transformation 'notr' ***/
    PtrFigure->Modifier_Domaine(ta_m[notr]*ox + tb_m[notr]*oy + te_m[notr],
                                tc_m[notr]*ox + td_m[notr]*oy + tf_m[notr],
                                ta_m[notr]*ix + tb_m[notr]*iy + te_m[notr],
                                tc_m[notr]*ix + td_m[notr]*iy + tf_m[notr],
                                ta_m[notr]*jx + tb_m[notr]*jy + te_m[notr],
                                tc_m[notr]*jx + td_m[notr]*jy + tf_m[notr]);

    if (npro>1)
       {
       /*** sauver les valeurs initiales ***/
       PtrFigure->Transmettre_Domaine(ox,oy,ix,iy,jx,jy);
       npro -= 1;

       /*** appels récursifs ***/
       for(int j=0;j<ntrans_m;j++)
           {
           Genere_Trans(j);
           /*** rétablir les valeurs de départ ***/
           PtrFigure->Modifier_Domaine(ox,oy,ix,iy,jx,jy);
           };
       npro += 1;
       }
    else
       {
       PtrFigure->Tronquer();
       PtrFigure->Transmettre_Domaine(ox,oy,ix,iy,jx,jy);
       /*** changement de coordonnées du domaine ***/
       PtrFigure->Modifier_Domaine(
                     (aix-orx)*ox/100.0 + (ajx-orx)*oy/100.0 + orx,
                     (aiy-ory)*ox/100.0 + (ajy-ory)*oy/100.0 + ory,
                     (aix-orx)*ix/100.0 + (ajx-orx)*iy/100.0 + orx,
                     (aiy-ory)*ix/100.0 + (ajy-ory)*iy/100.0 + ory,
                     (aix-orx)*jx/100.0 + (ajx-orx)*jy/100.0 + orx,
                     (aiy-ory)*jx/100.0 + (ajy-ory)*jy/100.0 + ory);
       /*** dessiner sur domaine ***/
       PtrFigure->Dessiner();
       PtrFigure->Modifier_Domaine(ox,oy,ix,iy,jx,jy);
       };
  };

/*-------------------------------------------------------*/
/*--------  Méthode qui dessine le méta-ifs  ----------*/
/*-------------------------------------------------------*/
void Meta_ifs::Dessiner(int np)
{
  float ox,oy,ix,iy,jx,jy;
  npro = np;

  /*** pour chaque élément de la liste : ***/
  for(PtrFigure = Premier; PtrFigure != 0; PtrFigure = PtrFigure->ElemSuivant())
    {
    PtrFigure->Transmettre_Domaine(ox,oy,ix,iy,jx,jy);
    PtrFigure->Tronquer();

    /*** appliquer l'algorithme récursif ***/
    for (int j = 0; j < ntrans_m; j++)
        {
        Genere_Trans(j);
        PtrFigure->Modifier_Domaine(ox,oy,ix,iy,jx,jy);
        };
    /*** simple liste - pas de transformations ***/
    if (ntrans_m==0)
        {
        PtrFigure->Modifier_Domaine(
                     (aix-orx)*ox/100.0 + (ajx-orx)*oy/100.0 + orx,
                     (aiy-ory)*ox/100.0 + (ajy-ory)*oy/100.0 + ory,
                     (aix-orx)*ix/100.0 + (ajx-orx)*iy/100.0 + orx,
                     (aiy-ory)*ix/100.0 + (ajy-ory)*iy/100.0 + ory,
                     (aix-orx)*jx/100.0 + (ajx-orx)*jy/100.0 + orx,
                     (aiy-ory)*jx/100.0 + (ajy-ory)*jy/100.0 + ory);
        PtrFigure->Dessiner();
        PtrFigure->Modifier_Domaine(ox,oy,ix,iy,jx,jy);
        };
    };
};

/*-------------------------------------------------------*/
/*------  Insertion d'un élément dans la liste  -------*/
/*-------------------------------------------------------*/
int Meta_ifs::Inserer(Point_H& Figure)
{
if ( ((Figure).ElemSuivant()==0)&&((&Figure)!=PtrDernier()) )
   {
   if (!Premier)                              // si la liste est vide :
      Premier = &Figure;                      // l'élément est le 1er de la liste
   else                                       // sinon :
      Dernier->ModifElemSuivant(&Figure);     // le dernier élément de la liste
                                              // pointe sur l'élément inséré.
   Dernier = &Figure;                         // l'élément inséré est le nouveau
   return 1;                                  // dernier élément
   }
else
   {
   return 0;
   };
};

/*-------------------------------------------------------*/
/*------- Extraction d'un élément de la liste  --------*/
/*-------------------------------------------------------*/
void Meta_ifs::Extraire(Point_H& Figure)
{
  Point_H* AncPtrFigure=Premier;
  Point_H* Null=0;

  for(PtrFigure = Premier; PtrFigure != 0; PtrFigure = PtrFigure->ElemSuivant())
    {
    if ((PtrFigure)==(&Figure))
       {
       if ((PtrFigure) == Premier)
          /*** faire pointer Premier vers ***/
          /*** le 2ème élément           ***/
          Premier=PtrFigure->ElemSuivant();
       else
          /*** faire pointer le précédent sur le suivant ***/
          AncPtrFigure->ModifElemSuivant(PtrFigure->ElemSuivant());
       };
    AncPtrFigure = PtrFigure;
    };
  Figure.ModifElemSuivant(Null);
};

/*-------------------------------------------------------*/
/*---  Mise en avant-plan d'un élément de la liste  ---*/
/*-------------------------------------------------------*/
void Meta_ifs::Avant_Plan(Point_H& Figure)
{
  Extraire(Figure);
  Inserer(Figure);        // le test n'est pas nécessaire !
};
```

```
/*-------------------------------------------------------*/
/*-- Appliquer une transformation a un point de H(X) --*/
/*-------------------------------------------------------*/
int Meta_ifs::Applic_Trans(int num,Point_H& fig)
{
float ox,oy,ix,iy,jx,jy;

if (num<ntrans_m)
  {
  fig.Transmettre_Domaine(ox,oy,ix,iy,jx,jy);
  fig.Modifier_Domaine(ta_m[num]*ox + tb_m[num]*oy + te_m[num],
                       tc_m[num]*ox + td_m[num]*oy + tf_m[num],
                       ta_m[num]*ix + tb_m[num]*iy + te_m[num],
                       tc_m[num]*ix + td_m[num]*iy + tf_m[num],
                  --   ta_m[num]*jx + tb_m[num]*jy + te_m[num],
                       tc_m[num]*jx + td_m[num]*jy + tf_m[num]);
  return 1;
  }
else
  return 0;
};

/*-------------------------------------------------------*/
/*- Indiquer quel élément du méta-ifs est sélectionné -*/
/*-------------------------------------------------------*/
int Meta_ifs::Element_Selectionne(int ptx,int pty)
{
  int trouve=0;

  /*** pour chaque élément de la liste : ***/
  for(PtrFigure = Premier; PtrFigure != 0; PtrFigure = PtrFigure->ElemSuivant())
    {
    trouve += 1;
    if (PtrFigure->Selectionne(ptx,pty))
       return trouve;
    };
  return 0;
};

/*=======================================================================*/
/*================= METHODES DE LA CLASSE  << POINT_H >>  ===============*/
/*=======================================================================*/

/*-------------------------------------------------------*/
/*------  Le constructeur de la classe Point_H  -------*/
/*-------------------------------------------------------*/
Point_H::Point_H() // constructeur à valeurs par défaut
{
  Suivant = 0;
  originex = 0;
  originey = 0;
  axeix = 100;
  axeiy = 0;
  axejx = 0;
  axejy = 100;
};

/*-------------------------------------------------------*/
/*---------  vérification si un pt appartient  --------*/
/*---------  au domaine du point de H(X)       --------*/
/*-------------------------------------------------------*/
int Point_H::Selectionne (int ptx, int pty)
{
  int minx,miny,maxx,maxy;

  minx=originex;
  if (minx>axeix) minx=axeix;
  if (minx>axejx) minx=axejx;
  miny=originey;
  if (miny>axeiy) miny=axeiy;
  if (miny>axejy) miny=axejy;
  maxx=originex;
  if (maxx<axeix) maxx=axeix;
  if (maxx<axejx) maxx=axejx;
  maxy=originey;
  if (maxy<axeiy) maxy=axeiy;
  if (maxy<axejy) maxy=axejy;

  if((ptx>minx)&&
     (ptx<maxx)&&
     (pty>miny)&&
     (pty<maxy))
     return 1;
  else
     return 0;
};

/*-------------------------------------------------------*/
/*---- Modification du domaine d'un point de H(X)  ----*/
/*-------------------------------------------------------*/
int Point_H::Modifier_Domaine (float ox, float oy, float ix,
                               float iy, float jx, float jy)
{
/*
  if ((ox>=0)&&(oy>=0)&&
      (ix>=0)&&(iy>=0)&&
      (jx>=0)&&(jy>=0)&&
      (ox<1300)&&(oy<1000)&&
      (ix<1300)&&(iy<1000)&&
      (jx<1300)&&(jy<1000))
     {
*/
     originex = ox;
     originey = oy;
     axeix = ix;
     axeiy = iy;
     axejx = jx;
     axejy = jy;
     return 1;
/*
     }
  else
     {
     return 0;
     };
*/
};

/*-------------------------------------------------------*/
/*---- Transmission du domaine d'un point de H(X)  ----*/
/*-------------------------------------------------------*/
void Point_H::Transmettre_Domaine (float &ox,float &oy,float &ix,
                                   float &iy,float &jx,float &jy)
{
  ox = originex;
  oy = originey;
  ix = axeix;
  iy = axeiy;
  jx = axejx;
  jy = axejy;
};
```

```
/*-----------------------------------------------------*/
/*----  Troncature du domaine d'un point de H(X)  -----*/
/*-----------------------------------------------------*/
void Point_H::Tronquer()
{
    /*** troncature sur 100x100 ***/
    if (originex < 0)   originex = 0;
    if (originex > 100) originex = 100;
    if (originey < 0)   originey = 0;
    if (originey > 100) originey = 100;
    if (axeix < 0)      axeix = 0;
    if (axeix > 100)    axeix = 100;
    if (axeiy < 0)      axeiy = 0;
    if (axeiy > 100)    axeiy = 100;
    if (axejx < 0)      axejx = 0;
    if (axejx > 100)    axejx = 100;
    if (axejy < 0)      axejy = 0;
    if (axejy > 100)    axejy = 100;
};

/*=====================================================================================*/
/*================== METHODES DE LA CLASSE  << LIGNE >>  ==================*/
/*=====================================================================================*/

/*-----------------------------------------------------*/
/*------- Le constructeur de la classe Ligne  ---------*/
/*-----------------------------------------------------*/
Ligne::Ligne(int c,char tp)
{
  couleur = c;
  type = tp;
};

/*-----------------------------------------------------*/
/*---------------  Dessiner une Ligne  ----------------*/
/*-----------------------------------------------------*/
void Ligne::Dessiner()
{
 setcolor(couleur);
 if (type=='V')
    {
    /*** une ligne verticale sur le domaine ***/
    line(originex+(axeix-originex)/2,originey+(axeiy-originey)/2,
         axejx+(axeix-originex)/2,axejy+(axeiy-originey)/2);
    }
 else
    {
    line(originex+(axejx-originex)/2,originey+(axejy-originey)/2,
         axeix+(axejx-originex)/2,axeiy+(axejy-originey)/2);

    }
};

/*-----------------------------------------------------*/
/*----------  Etablir un clone de la ligne  -----------*/
/*-----------------------------------------------------*/
 Ligne* Ligne::Clone()

 {
 Ligne *clone;

 clone = new Ligne (couleur,type);
 return clone;
 };
```

```
/*-----------------------------------------------------*/
/*------ fonction qui génère un point / origine  ------*/
/*-----------------------------------------------------*/
void Ligne::Genere_PtD(float &pointx,float &pointy,int &coul)

{
  float ptx,pty,rnd;

  rnd = random(1000)/10.0;
  if (type=='V')
     {
     ptx = 50;
     pty = rnd;
     }
  else
     {
     ptx = rnd;
     pty = 50;
     };
  pointx = (axeix-originex)*ptx/100.0 + (axejx-originex)*pty/100.0 + originex;
  pointy = (axeiy-originey)*ptx/100.0 + (axejy-originey)*pty/100.0 + originey;
  coul = couleur;
};

/*=====================================================================================*/
/*================ METHODES DE LA CLASSE  << RECTANGLE >>  ================*/
/*=====================================================================================*/

/*-----------------------------------------------------*/
/*-------  Le constructeur de la classe Rect  ---------*/
/*-----------------------------------------------------*/
Rect::Rect(int c)
{
  couleur = c;
};

/*-----------------------------------------------------*/
/*-------------  Dessiner un rectangle  ---------------*/
/*-----------------------------------------------------*/
void Rect::Dessiner()
{
 setcolor(couleur);
 /*** dessiner le rectangle ***/
 line (originex,originey,axeix,axeiy);
 line (originex,originey,axejx,axejy);
 line (axejx,axejy,axejx+axeix-originex,(axejy+axeiy-originey));
 line (axeix,axeiy,axeix+axejx-originex,(axeiy+axejy-originey));
};

/*-----------------------------------------------------*/
/*----------  Etablir un clone du rectangle  ----------*/
/*-----------------------------------------------------*/
 Rect* Rect::Clone()

 {
 Rect *clone;

 clone = new Rect(couleur);
 return clone;
 };

/*-----------------------------------------------------*/
/*------ fonction qui génère un point / origine  ------*/
/*-----------------------------------------------------*/
void Rect::Genere_PtD(float &pointx,float &pointy,int &coul)
```

```
{
  float ptx,pty,rnd;
  int xy;

  rnd = random(1000)/10.0;
  xy = random(4);
  if (xy==1)
     {
     ptx = rnd;
     pty = 1;
     };
  if (xy==2)
     {
     pty = rnd;
     ptx = 100;
     };
  if (xy==3)
     {
     ptx = rnd;
     pty = 100;
     };
  if (xy==4)
     {
     ptx = 1;
     pty = rnd;
     };
  pointx = (axeix-originex)*ptx/100.0 + (axejx-originex)*pty/100.0 + originex;
  pointy = (axeiy-originey)*ptx/100.0 + (axejy-originey)*pty/100.0 + originey;
  coul = couleur;
};


/*=========================================================================*/
/*================ METHODES DE LA CLASSE  << FRACTAL >>  ==================*/
/*=========================================================================*/

/*---------------------------------------------------------*/
/*--------  Le constructeur de la classe ifs  ---------*/
/*---------------------------------------------------------*/
 ifs::ifs (int nt, float *a,float *b,float *c, float *d,float *e,float *f,
           float *p, int prec,int qual,int col)
 {
  int i,numt;
  float newx,newy,proba,rnd;

  for (int j = 0; j < nt; ++j)
      {
      probab[j] = p[j];
      ta[j] = a[j];
      tb[j] = b[j];
      tc[j] = c[j];
      td[j] = d[j];
      te[j] = e[j];
      tf[j] = f[j];
      }
  ntrans = nt;
  precision = prec;
  couleur = col;

  if (qual==2)
        {
        /*** test sur la qualité des transformations ***/
        /*** -> si leur domaine reste dans 100x100   ***/
        qual=1;
        for (i=0;i<ntrans;i++)
                {
                newx =                          te[i] ;
                newy =                          tf[i] ;
                if ((newx<0)||(newx>100)||(newy<0)||(newy>100)) qual = 3;
                newx = ta[i]*100 + tb[i]*100 + te[i] ;
                newy = tc[i]*100 + td[i]*100 + tf[i] ;
                if ((newx<0)||(newx>100)||(newy<0)||(newy>100)) qual = 3;
                newx =             tb[i]*100 + te[i] ;
                newy =             td[i]*100 + tf[i] ;
                if ((newx<0)||(newx>100)||(newy<0)||(newy>100)) qual = 3;
                newx = ta[i]*100             + te[i] ;
                newy = tc[i]*100             + tf[i] ;
                if ((newx<0)||(newx>100)||(newy<0)||(newy>100)) qual = 3;
                };
        };
  qualite = qual;

  ptx=50;
  pty=50;
  /*** déterminier le point dançant ***/
  randomize();
  for (i=1; i <= 15; ++i)
      {
      /*** choix d'une des transformations ***/
      rnd = random(100);
      numt=0;
      proba=p[0]*100.0;
      while (rnd>proba)
           {
           ++numt;
           proba += p[numt]*100.0;
           }
      /*** appliquer la transformation ***/
      newx = a[numt]*ptx + b[numt]*pty + e[numt] ;
      newy = c[numt]*ptx + d[numt]*pty + f[numt] ;
      if ((newx>=0)&&(newx<=100)&&(newy>=0)&&(newy<=100))
         {
         ptx = newx;
         pty = newy;
         };
      };
 };

/*---------------------------------------------------------*/
/*----------  Etablir un clone de l'ifs  -------------*/
/*---------------------------------------------------------*/
 ifs* ifs::Clone()

 {
 ifs *clone;

 clone = new ifs (ntrans,ta,tb,tc,td,te,tf,
                  probab,precision,qualite,15);
 clone->couleur = couleur;
 return clone;
 };

/*---------------------------------------------------------*/
/*----------  Afficher le domaine d'un ifs  -----------*/
/*---------------------------------------------------------*/
void ifs::Encadrer()
{
 setcolor(WHITE);
 /*** dessiner le rectangle ***/
 line (originex,originey,axeix,axeiy);
 line (originex,originey,axejx,axejy);
 line (axejx,axejy,axejx+axeix-originex,(axejy+axeiy-originey));
 line (axeix,axeiy,axeix+axejx-originex,(axeiy+axejy-originey));
};
```

```
/*-------------------------------------------------------*/
/*-- fonction qui génère un point dançant au hasard  --*/
/*-------------------------------------------------------*/
void ifs::Genere_PtD(float &pointx,float &pointy,int &coul)

{
  int numt,rnd;
  float newx,newy,proba;


  /*** choix d'une des transformations ***/
  rnd = random(100);
  numt=0;
  proba=probab[0]*100.0;
  while (rnd>proba)-
     {
     ++numt;
     proba += probab[numt]*100.0;
     };
  /*** appliquer la transformation ***/
  newx = ta[numt]*ptx + tb[numt]*pty + te[numt] ;
  newy = tc[numt]*ptx + td[numt]*pty + tf[numt] ;
  if ((newx>=0)&&(newx<=100)&&(newy>=0)&&(newy<=100))
     {
     ptx = newx;
     pty = newy;
     };
  pointx = (axeix-originex)*ptx/100.0 + (axejx-originex)*pty/100.0 + originex;
  pointy = (axeiy-originey)*ptx/100.0 + (axejy-originey)*pty/100.0 + originey;
  coul = couleur;
};


/*-------------------------------------------------------*/
/*--  génèration de l'algorithme-chaos avec domaine  --*/
/*-------------------------------------------------------*/
 void ifs::Dessiner()

/* algorithme-chaos
   VARIABLES:
   numt      : (I) le numéro courant de la transformation
   x,y,px,py : (R) les coordonnées du point actuel
   newx,newy : (R) les coordonnées du point suivant
   et des variables auxiliaires
*/
{
  float px,py, newx=50, newy=50;
  register int i;
  int numt,rnd;
  float angle;
  float proba;
  float numit;
  unsigned long int numits;
  float normei,normej, psij,aired,airec;

  /*** calcul de l'aire occupée par le domaine ***/
  normei = sqrt ( (axeix-originex)*(axeix-originex)
                + (axeiy-originey)*(axeiy-originey) );
  normej = sqrt ( (axejx-originex)*(axejx-originex)
                + (axejy-originey)*(axejy-originey) );
  psij   = (axeix-originex)*(axejx-originex)
           + (axeiy-originey)*(axejy-originey);
  if ( ((normei*normej)!=0))
     {
     if( ((psij/(normei*normej))<1)&&
         ((psij/(normei*normej))>-1)   )
          {
          angle =  acos (psij/(normei*normej));
          }
     }
  else
     angle = 0;

  aired =  (normei*normej*sin(angle));

  /*** pour déterminer le nombre d'itérations ***/
  /*** necessaires pour la précision demandée ***/
  airec = 10000.0;

  numit = precision * (aired/airec);
  if (numit<64)
     numits = (int) (numit * 1000);
  else
     numits = 64000;
  if (numits<10) numits=10;

  randomize();

  if (numits>0)
     {

     px = (axeix-originex)*ptx/100.0 + (axejx-originex)*pty/100.0 + originex;
     py = (axeiy-originey)*ptx/100.0 + (axejy-originey)*pty/100.0 + originey;

     if (qualite>1)
        {
        for (i=1; i <= numits; ++i)
             {
             /*** dessiner le pixel ***/
             putpixel((int)px,(int)py,couleur);
             /*** choix d'une des transformations ***/
             rnd = random(100);
             numt=0;
             proba=probab[0]*100.0;
             while (rnd>proba)
                   {
                   ++numt;
                   proba += probab[numt]*100.0;
                   }

             /*** appliquer la transformation ***/
             newx = ta[numt]*ptx + tb[numt]*pty + te[numt] ;
             newy = tc[numt]*ptx + td[numt]*pty + tf[numt] ;
             if ((newx>=0)&&(newx<=100)&&(newy>=0)&&(newy<=100))
                {
                ptx = newx;
                pty = newy;
                px = (axeix-originex)*ptx/100.0 + (axejx-originex)*pty/100.0 + or
iginex;
                py = (axeiy-originey)*ptx/100.0 + (axejy-originey)*pty/100.0 + or
iginey;
                };
             };
           }
     else
        {
        for (i=1; i <= numits; ++i)
             {
             /*** dessiner le pixel ***/
             putpixel((int)px,(int)py,couleur);
             /*** choix d'une des transformations ***/
             rnd = random(100);
```

```
            numt=0;
            proba=probab[0]*100.0;
            while (rnd>proba)
                  {
                  ++numt;
                  proba += probab[numt]*100.0;
                  }

            /*** appliquer la transformation ***/
            newx = ta[numt]*ptx + tb[numt]*pty + te[numt] ;
            newy = tc[numt]*ptx + td[numt]*pty + tf[numt] ;
            ptx = newx;
            pty = newy;
            px = (axeix-originex)*ptx/100.0 + (axejx-originex)*pty/100.0 + origi
nex;
            py = (axeiy-originey)*ptx/100.0 + (axejy-originey)*pty/100.0 + origi
ney;
            };
         };
      }
};

/*-------------------------------------------------------*/
/*------  'random iteration algorithm' avec zoom  -----*/
/*-------------------------------------------------------*/
 int ifs::Zoom(int pt1x,int pt1y,int pt2x,int pt2y)

/* un premier algorithme-chaos
   VARIABLES:
   numt      : (I) le numéro courant de la transformation
   x,y,px,py : (R) les coordonnées du point actuel
   newx,newy : (R) les coordonnées du point suivant
   et des variables auxiliaires
*/
{
  int res;
  float px,py, newx=originex, newy=originey;
  register int i;
  int numt,rnd;
  float angle;
  float proba;
  float numit;
  unsigned long int numits;
  float normei,normej, psij,aired,airec;

  /*** Test si le domaine choisi est valide ***/
  if (pt1x<0    || pt1x>100 ||
      pt1y<0    || pt1y>100 ||
      pt2x<pt1x || pt2x>100 ||
      pt2y<pt1y || pt2y>100    )
      {return -1;};

  /*** calcul de l'aire occupée par le domaine ***/
  normei = sqrt ( (axeix-originex)*(axeix-originex)
                + (axeiy-originey)*(axeiy-originey) );
  normej = sqrt ( (axejx-originex)*(axejx-originex)
                + (axejy-originey)*(axejy-originey) );
  psij   = (axeix-originex)*(axejx-originex)
         + (axeiy-originey)*(axejy-originey);
  if ( ((normei*normej)!=0))
     {
     if( ((psij/(normei*normej))<1)&&
         ((psij/(normei*normej))>-1)   )
         {
         angle =  acos (psij/(normei*normej));
         }
     }
  else
     angle = 0;

  aired =  (normei*normej*sin(angle));

  /*** pour déterminer le nombre d'itérations ***/
  /*** necessaires pour la précision demandée ***/
  airec = 10000.0;

  numit = precision * (aired/airec);
  if (numit<64)
     numits = (int) (numit * 1000);
  else
     numits = 64000;

  randomize();

  if (numits>0)
     {
     px = (axeix-originex)*ptx/100.0 + (axejx-originex)*pty/100.0 + originex;
     py = (axeiy-originey)*ptx/100.0 + (axejy-originey)*pty/100.0 + originey;

     for (i=1; i <= numits; ++i)
        {
        /*** choix d'une des transformations ***/
        rnd = random(100);
        numt=0;
        proba=probab[0]*100.0;
        while ((rnd>proba)&&(numt<ntrans))
          {
          ++numt;
          proba += probab[numt]*100.0;
          }

        /*** appliquer la transformation ***/
        newx = ta[numt]*ptx + tb[numt]*pty + te[numt] ;
        newy = tc[numt]*ptx + td[numt]*pty + tf[numt] ;
        ptx = newx;
        pty = newy;
        /*** si le point se trouve dans le rectangle ***/
        if (newx>pt1x && newx<pt2x && newy>pt1y && newy<pt2y)
           {
           /*** nouvelles coordonnées du point dans 100x100 ***/
           /*** suivant le zoom choisi                      ***/
           newx = (newx-pt1x)/((pt2x-pt1x)*1.0);
           newy = (newy-pt1y)/((pt2y-pt1y)*1.0);
           /*** calculer les coordonnées pour le domaine de l'IFS ***/
           /*** i.e. faire encore un changement de coordonnées    ***/
           px = (axeix-originex)*newx + (axejx-originex)*newy + originex;
           py = (axeiy-originey)*newx + (axejy-originey)*newy + originey;
           /*** dessiner le pixel ***/
           putpixel((int)px,(int)py,couleur);
           };
        }
     }
     if (numits==64000) {return -2;}
     else return 0;
};
```

```
/*=========================================================================*/
/*=============== METHODES DE LA CLASSE  << IFS_COLORE >>  ================*/
/*=========================================================================*/

/*-------------------------------------------------------*/
/*----------  Le constructeur de la classe  -------------*/
/*-------------------------------------------------------*/
 ifs_Couleur::ifs_Couleur(int nt, float *a,float *b,float *c,
                          float *d,float *e,float *f,float *p,
                          int brill, int prec, int ncoul, int *coul)

 : ifs(nt,a,b,c,d,e,f,p,prec,1,15)

 {
  for (int j = 0; j < ncoul; ++j)
      {
      couleur[j] = coul[j];
      };
  brillance = brill;
  ncouleurs = ncoul;

  /*** calcul du bitmap coloré ***/
  Bmp_Init();

 };


/*-------------------------------------------------------*/
/*----------     Initialiser le bitmap      -------------*/
/*-------------------------------------------------------*/
 void ifs_Couleur::Bmp_Init()

{
  register int i,j,px,py;
  unsigned long int nits;
  int newx=50, newy=50;
  int numt,rnd;
  float proba;

  nits = precision * 1000;

  for (i=0; i < 100; ++i)
      {
      for (j=0; j < 100; ++j)
          {
          bitmap[i][j] = 0;
          };
      };

  randomize();

  for (i=1; i <= nits; ++i)
      {
      /*** choix d'une des transformations ***/
      rnd = random(100);
      numt=0;
      proba=probab[0]*100.0;
      while ((rnd>proba)&&(numt<ntrans))
          {
          ++numt;
          proba += probab[numt]*100.0;
          };

      /*** appliquer la transformation ***/
      newx = (int) (ta[numt]*ptx + tb[numt]*pty + te[numt]) ;
      newy = (int) (tc[numt]*ptx + td[numt]*pty + tf[numt]) ;

      if ((newx >= 0) && (newx < 100) && (newy >= 0) && (newy < 100)
          && (bitmap[newx][newy]<150))
          {
          bitmap[newx][newy] += 1;
          };
      ptx = newx;
      pty = newy;
      };

  color_scaling = brillance / (precision *1.0);
};

/*-------------------------------------------------------*/
/*----  génèration du 'random iteration algorithm' ----*/
/*----  avec domaine et couleurs                   ----*/
/*-------------------------------------------------------*/
 void ifs_Couleur::Dessiner()

/* un premier algorithme-chaos
   VARIABLES:
   numt      : (I) le numéro courant de la transformation
   x,y,px,py : (R) les coordonnées du point actuel
   newx,newy : (R) les coordonnées du point suivant
   et des variables auxiliaires
*/
{
  register int i,j,px,py;
  int coul,numt,rnd;
  float scaling,proba;
  float normei,normej, psij,aired;
  float angle;

  scaling = brillance / (precision*1.0);

if ((int)(scaling*1000) != (int)(color_scaling*1000))
  {
  /*** calcul du bitmap coloré ***/
  Bmp_Init();
  };

  /*** calcul de l'aire occupée par le domaine ***/
  normei = sqrt ( (axeix-originex)*(axeix-originex)
                  + (axeiy-originey)*(axeiy-originey) );
  normej = sqrt ( (axejx-originex)*(axejx-originex)
                  + (axejy-originey)*(axejy-originey) );
  psij   = (axeix-originex)*(axejx-originex)
           + (axeiy-originey)*(axejy-originey);
  if ( ((normei*normej)!=0))
     {
     if( ((psij/(normei*normej))<1)&&
         ((psij/(normei*normej))>-1)   )
         {
         angle =  acos (psij/(normei*normej));
         }
     }
  else
     angle = 0;

  aired =  (normei*normej*sin(angle));

  if (aired>50)
      {
      for (i=0; i < 100; ++i)
          {
          for (j=0; j < 100; ++j)
```

```
            {
            if (bitmap[i][j] > 0)
               {
               coul = bitmap[i][j]*color_scaling;
               if (coul >= ncouleurs) {coul = ncouleurs - 1;};
               px = (axeix-originex)*i/100.0 + (axejx-originex)*j/100.0
                                             + originex;
               py = (axeiy-originey)*i/100.0 + (axejy-originey)*j/100.0
                                             + originey;
               putpixel((int)px,(int)py,couleur[coul]);
               };
            };
         };
      };
};


/*-------------------------------------------------------*/
/*--  retournement horizontal et vertical de l'image --*/
/*-------------------------------------------------------*/
void ifs_Couleur::Mirroir(char type)

{
   if (type == 'H')
      Modifier_Domaine(originex,100-originey,axeix,100-axeiy,axejx,100-axejy);
   else
      Modifier_Domaine(100-originex,originey,100-axeix,axeiy,100-axejx,axejy);
};

/*************************************************************************/
/*********************   LE  PROGRAMME  PRINCIPAL   **********************/
/*************************************************************************/
main()
{
 /*** code du triangle de Sierpinski  ***/
 float sa[3] = { 0.49, 0.49, 0.49},
       sb[3] = {    0,    0,    0},
       sc[3] = {    0,    0,    0},
       sd[3] = { 0.49, 0.49, 0.49},
       se[3] = {    0,   51,   25},
       sf[3] = {    0,    0,   51},
       sp[3] = {  0.3,  0.2,  0.5};


 /*** code d'un farn ***/
 float fa[4] = {-0.15, 0.85, 0.20,    0},
       fb[4] = { 0.28, 0.04,-0.21,    0},
       fc[4] = { 0.26,-0.04, 0.18,    0},
       fd[4] = { 0.24, 0.85, 0.22, 0.16},
       fe[4] = {   58,    8,   40,   50},
       ff[4] = { 0.44,   18, 1.60,    0},
       fp[4] = { 0.09,  0.8, 0.09, 0.02};
 int   coulf[6] = { 02, 02, 03, 10, 11, 14 };

 /*** code d'un arbre ***/
 float aa[4] = {0.01,  0.55,  0.7, 0.6},
       ab[4] = { 0.0, -0.24, 0.20,0.05},
       ac[4] = { 0.0,  0.24,-0.20,-0.05},
       ad[4] = {0.45,  0.65,  0.7, 0.6},
       ae[4] = {  48,    46, -5.5,14.8},
       af[4] = {  55,    -1,   19,  20},
       ap[4] = { 0.05,   0.3,  0.35, 0.3};
```

```
/*** code d'une feuille ***/
float la[2] = { 0.44,-0.82},
      lb[2] = { 0.32, 0.16},
      lc[2] = {-0.07,-0.16},
      ld[2] = { 0.61, 0.81},
      le[2] = {    0,   88},
      lf[2] = {   46,   10},
      lp[2] = {  0.4,  0.6};

float x,y;
int c;

 /*** code d'un arbre 1 IDD_IFS5 ***/
   aa[0]=0.01; aa[1]= 0.55; aa[2]=  0.7;
   ab[0]= 0.0; ab[1]=-0.24; ab[2]= 0.10;
   ac[0]= 0.0; ac[1]= 0.24; ac[2]=-0.10;
   ad[0]=0.45; ad[1]= 0.65; ad[2]=  0.7;
   ae[0]=  50; ae[1]=   47; ae[2]=  5.5;
   af[0]=  55; af[1]=    9; af[2]=   10;
   ap[0]= 0.1; ap[1]=  0.4; ap[2]=  0.5;

  /*** initialiser le mode graphique ***/
  int graphdriver = DETECT , graphmode;
  initgraph (&graphdriver,&graphmode,"c:\\borlandc\\bgi");

  /*** construire les objets ***/
  Ligne ligne(BROWN,'V');
  Rect rectangle2(YELLOW);
  Rect rectangle3(LIGHTBLUE);
  ifs feuille2(2,la,lb,lc,ld,le,lf,lp,1,1,YELLOW);
  ifs feuille3(2,la,lb,lc,ld,le,lf,lp,1,1,GREEN);
  ifs farn(4,fa,fb,fc,fd,fe,ff,fp,2,1,YELLOW);

  /*** créer les méta-IFS ***/
  Meta_ifs sier(3,aa,ab,ac,ad,ae,af,ap,2);

  sier.Inserer(feuille2);
 /* sier.Inserer(feuille3);
  *** et dessiner le méta-IFS ***/
  sier.Transformer(0,0,200,0,0,200);
  sier.Dessiner(3);

  sier.Transformer(200,0,250,0,200,50);
  sier.Dessiner_Chaos(3);

  /*
  ifs_Couleur farnc(4,fa,fb,fc,fd,fe,ff,fp,5,5,6,coulf);
  farnc.Dessiner();
  farnc.Mirroir('H');
  farnc.Dessiner();
  */

  /*** agrandir et déplacer la feuille ***
  farn.Modifier_Domaine(100,200,300,150,200,50);
  farn.Dessiner();
  farn.Encadrer();
  *** déplacer le domaine pour dessiner le zoom ***/
  feuille3.Modifier_Domaine(0,400,150,400,0,250);
  feuille3.Zoom(50,50,100,100);
  feuille3.Encadrer();

  /*** applicquer des transformations de l'arbre ***/
  /*** aux objets qui y seront insérés           ***/
  Meta_ifs arbre(4,aa,ab,ac,ad,ae,af);
```

```
  arbre.Applic_Trans(0,ligne);
  arbre.Applic_Trans(1,rectangle2);
  arbre.Applic_Trans(2,rectangle3);

  /*** insérer les objets dans le méta-IFS ***/
  arbre.Inserer(ligne);
  arbre.Inserer(rectangle3);
  arbre.Transformer(150,0,250,0,150,100);
  arbre.Extraire(rectangle2);
  arbre.Dessiner(2);
  arbre.Transformer(50,0,250,0,50,100);
  arbre.Extraire(ligne);
  arbre.Dessiner(4);

  getch();
  closegraph();
}
```

Annexe F :

```
/*********************/
/* types prédéfinis */
/*********************/

#define IDM_GUI 101

#define IDM_ABO 201
#define IDM_RE1 202
#define IDM_RE2 203
#define IDM_RE3 204
        --
#define IDM_LOI1 301
#define IDM_SAI1 302
#define IDM_LOB1 303
#define IDM_SAB1 304
#define IDM_SET1 305
#define IDM_CHI1 306
#define IDM_CHC1 307
#define IDM_DRA1 308
#define IDM_ZOO1 309
#define IDM_SEL1 310
#define IDM_MOD1 311

#define IDM_LOI2 401
#define IDM_SAI2 402
#define IDM_LOB2 403
#define IDM_SAB2 404
#define IDM_SET2 405
#define IDM_DRA2 406
#define IDM_ZOO2 407
#define IDM_SEA2 408
#define IDM_INS2 409
#define IDM_MCO2 410
#define IDM_APP2 411
#define IDM_INH2 412
#define IDM_APH2 413
#define IDM_MOD2 414

#define IDM_LOB3 501
#define IDM_AJD3 502
#define IDM_BOX3 503
#define IDM_BO13 504

#define IDM_PTH1 601
#define IDM_PTH2 602
#define IDM_PTH3 603
#define IDM_PTH4 604
#define IDM_PTH  605
#define IDM_DRA3 607
#define IDM_ATT3 608
```

```
#define IDD_FNAME 0x10
#define IDD_FPATH 0x11
#define IDD_FLIST 0x12
#define IDD_TRANA 0x13
#define IDD_TRANB 0x14
#define IDD_TRANC 0x15
#define IDD_TRAND 0x16
#define IDD_TRANE 0x17
#define IDD_TRANF 0x18
#define IDD_PROBA 0x19
#define IDD_FRNAM 0x20
#define IDD_NEXT  0x21
#define IDD_NTRAN 0x22
#define IDD_DENSI 0x23

#define IDD_WHITE 10
#define IDD_BLUE  11
#define IDD_GREEN 12
#define IDD_CYAN  13
#define IDD_RED   14
#define IDD_MAGEN 15
#define IDD_YELLO 16
#define IDD_GRAY  17
#define IDD_SHGRA 110
#define IDD_SHGRE 111
#define IDD_SHRED 112

#define IDD_IFS1 20
#define IDD_IFS2 21
#define IDD_IFS3 22
#define IDD_IFS4 23
#define IDD_IFS5 24
#define IDD_IFS6 25

#define IDD_PAINT 3

#define IDD_DI1 40
#define IDD_DI2 41
#define IDD_DM1 42
#define IDD_DM2 43
#define IDD_MD  44
#define IDD_MC  45
#define IDD_END 46
```

```
#include <windows.h>
#include "FR.H"

#define SBS_VERT_TAB (SBS_VERT | WS_TABSTOP)

IDI_FRACTALS ICON FRACTALS.ICO
IDI_METFRACT ICON MFRACT.ICO
IDI_F_LOA    ICON FRAC_LOA.ICO
IDI_F_SAV    ICON FRAC_SAV.ICO
IDI_F_COL    ICON FRAC_COL.ICO
IDI_FIGURE   ICON FRAC_FIG.ICO

scan BITMAP scan.bmp

FRACTALS MENU
{
  POPUP "Edition"
  {       MENUITEM "Informations"           ,IDM_ABO
          MENUITEM "Reset 100x100"          ,IDM_RE1
          MENUITEM "Reset 400x400"          ,IDM_RE2
          MENUITEM "Reset Ecran"            ,IDM_RE3
          MENUITEM "Quit"                   ,IDM_QUI
  }
  POPUP "Dimension"
  {
          MENUITEM "Afficher SCAN.BMP"      ,IDM_LOB3
          POPUP "Box Counting"
          {       MENUITEM "100x100"        ,IDM_BO13
                  MENUITEM "400x400"        ,IDM_BOX3
          }
  }
  POPUP "Code-IFS"
  {
          MENUITEM "Nouveau IFS"            ,IDM_SET1
          MENUITEM "Modifier IFS"           ,IDM_MOD1, GRAYED
          MENUITEM "Choisir IFS"            ,IDM_SEL1
          MENUITEM SEPARATOR
          MENUITEM "Changer Couleurs  "     ,IDM_CHC1, GRAYED
          MENUITEM "Dessiner l'attracteur"  ,IDM_DRA1, GRAYED
          MENUITEM "Zoom sur l'attracteur"  ,IDM_ZOO1, GRAYED
  }
  POPUP "Code-M_IFS"
  {
          MENUITEM "Nouveau Meta-IFS"       ,IDM_SET2
          MENUITEM "Modifier Meta-IFS"      ,IDM_MOD2, GRAYED
          MENUITEM "Choisir Meta-IFS"       ,IDM_SEA2
          MENUITEM SEPARATOR
          MENUITEM "Inserer figure"         ,IDM_INH2, GRAYED
          MENUITEM "Inserer IFS"            ,IDM_INS2, GRAYED
          MENUITEM SEPARATOR
          POPUP    "Appl. transformation a"
          {
            MENUITEM "figure"               ,IDM_APH2, GRAYED
            MENUITEM "IFS"                  ,IDM_APP2, GRAYED
          }
          MENUITEM "Montrer les composants",IDM_MCO2, GRAYED
          MENUITEM "Dessiner l'attracteur"  ,IDM_DRA2, GRAYED
          MENUITEM "Zoom sur l'attracteur"  ,IDM_ZOO2, GRAYED
  }
  POPUP "Figure"
  {
          POPUP "Choisir objet"
          {
            MENUITEM "Ligne verticale"      ,IDM_PTH1
            MENUITEM "Ligne horizontale"    ,IDM_PTH2
            MENUITEM "Triangle"             ,IDM_PTH3
            MENUITEM "Rectangle"            ,IDM_PTH4
            MENUITEM "Cercle"               ,IDM_PTH, GRAYED
            MENUITEM "ARectangle"           ,IDM_PTH, GRAYED
            MENUITEM "Polygone"             ,IDM_PTH, GRAYED
            MENUITEM "Cercle + Texture"     ,IDM_PTH, GRAYED
            MENUITEM "Rectangle + Texture"  ,IDM_PTH, GRAYED
            MENUITEM "ARectangle + Texture",IDM_PTH, GRAYED
            MENUITEM "Polygone + Texture"   ,IDM_PTH, GRAYED
          }
          MENUITEM SEPARATOR
          MENUITEM "Dessiner "              ,IDM_DRA3, GRAYED
          MENUITEM "Changer attributs"      ,IDM_ATT3, GRAYED
  }
  POPUP "Fichiers"
  {
          MENUITEM "Ouvrir IFS"             ,IDM_LOI1, GRAYED
          MENUITEM "Ouvrir Meta-IFS"        ,IDM_LOI2, GRAYED
          MENUITEM "Enregistrer IFS"        ,IDM_SAI1, GRAYED
          MENUITEM "Enregistrer Meta-IFS"   ,IDM_SAI2, GRAYED
          MENUITEM SEPARATOR
          MENUITEM "Ouvrir BMP"             ,IDM_LOB2, GRAYED
          MENUITEM "Enregistrer BMP"        ,IDM_SAB2, GRAYED
  }
}

#define TABGRP (WS_TABSTOP | WS_GROUP)

Aboutbox DIALOG 10,10,108,100
   STYLE WS_POPUP | WS_DLGFRAME
   {
   LTEXT  "Fractal Editor"                -1, 32,  8,100,  8
   ICON   "IDI_FRACTALS"                  -1,  8,  8,  0,  0
   LTEXT  "Un exemple de traitement"      -1,  8, 32,100,  8
   LTEXT  "de l'environnement Fractal"    -1,  8, 42,100,  8
   LTEXT  "en C++."                       -1,  8, 52,100,  8
   LTEXT  "Wolfgang Heinen, 1991"         -1,  8, 62,100,  8
   DEFPUSHBUTTON "OK"                   IDOK, 35, 80, 30, 12, WS_GROUP
   }


Colorbox DIALOG 10,10,124,158
   STYLE WS_POPUP | WS_DLGFRAME
   {
   CTEXT   ""                        IDD_PAINT, 99,134, 15, 15
   CONTROL "Rouge",-1,"static",       SS_CENTER, 10,  4, 24,  8
   CONTROL "",10,"scrollbar",    SBS_VERT_TAB, 10, 16, 24,100
   CONTROL "",13,"static",           SS_CENTER, 10,120, 24,  8
   CONTROL "Vert",-1,"static",       SS_CENTER, 50,  4, 24,  8
   CONTROL "",11,"scrollbar",    SBS_VERT_TAB, 50, 16, 24,100
   CONTROL "",14,"static",           SS_CENTER, 50,120, 24,  8
   CONTROL "Bleu",-1,"static",       SS_CENTER, 90,  4, 24,  8
   CONTROL "",12,"scrollbar",    SBS_VERT_TAB, 90, 16, 24,100
   CONTROL "",15,"static",           SS_CENTER, 90,120, 24,  8
   DEFPUSHBUTTON "Accepter la couleur"  IDOK, 10,134, 84, 15, WS_GROUP
   }

SelectIFS DIALOG 10,10,102,140
   STYLE WS_POPUP | WS_DLGFRAME
```

```
    {
    LTEXT          "Selection IFS"          -1, 32,  8,140,  8
    ICON           "IDI_FRACTALS"           -1,  8,  8,  0,  0
    GROUPBOX       "IFS predefinis"         -1,  8, 32, 84, 78
     RADIOBUTTON   "Triangle Sierp."  IDD_IFS1, 16, 44, 60, 12, TABGRP
     RADIOBUTTON   "Rectangle"        IDD_IFS2, 16, 54, 60, 12
     RADIOBUTTON   "Farn"             IDD_IFS3, 16, 64, 60, 12
     RADIOBUTTON   "Feuille"          IDD_IFS4, 16, 74, 60, 12
     RADIOBUTTON   "Arbre1"           IDD_IFS5, 16, 84, 60, 12
     RADIOBUTTON   "Arbre2"           IDD_IFS6, 16, 94, 60, 12
    DEFPUSHBUTTON  "OK"                  IDOK,  8,118, 35, 14, WS_GROUP
    PUSHBUTTON     "Annuler"         IDCANCEL, 56,118, 35, 14, WS_GROUP
    }

SelMetIFS DIALOG 10,10,102,130
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT          "Selection Meta-IFS"     -1, 32,  8,140,  8
    ICON           "IDI_METFRACT"           -1,  8,  8,  0,  0
    GROUPBOX       "IFS predefinis"         -1,  8, 32, 84, 68
     RADIOBUTTON   "Triangle Sierp."  IDD_IFS1, 16, 44, 60, 12, TABGRP
     RADIOBUTTON   "Farn"             IDD_IFS3, 16, 54, 60, 12
     RADIOBUTTON   "Arbre1"           IDD_IFS5, 16, 64, 60, 12
     RADIOBUTTON   "Arbre2"           IDD_IFS6, 16, 74, 60, 12
     RADIOBUTTON   "Liste d'IFS"      IDD_IFS2, 16, 84, 60, 12
    DEFPUSHBUTTON  "OK"                  IDOK,  8,108, 35, 14, WS_GROUP
    PUSHBUTTON     "Annuler"         IDCANCEL, 56,108, 35, 14, WS_GROUP
    }

SelAppIFS DIALOG 10,10,102,76
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT          "Appliquer transf."      -1, 32,  8,140,  8
    LTEXT          "du Meta-IFS"            -1, 32, 18,140,  8
    ICON           "IDI_F_COL"              -1,  8,  8,  0,  0
    LTEXT          "Numero transf. :"       -1,  8, 34, 84, 12
    EDITTEXT                         IDD_FNAME, 70, 34, 20, 12, ES_AUTOHSCROLL
    PUSHBUTTON     "Appliquer"           IDOK,  8, 53, 35, 14, WS_GROUP
    PUSHBUTTON     "Annuler"         IDCANCEL, 56, 53, 35, 14, WS_GROUP
    }

DrawFract DIALOG 10, 10, 120,123
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT          "Dessiner l'attracteur", -1, 30,  8,126, 12
    LTEXT          "domaine",               -1, 70, 58, 72, 12
    LTEXT          "(origine-axex-axey)",   -1, 26, 68, 72, 12
    ICON           "IDI_F_COL"              -1,  8,  8,  0,  0
    GROUPBOX       "Domaine"                -1,  8, 32,102, 48
     RADIOBUTTON   "100 x 100"       IDD_DI1, 16, 44, 50, 12
     RADIOBUTTON   "Selectionner"    IDD_DI2, 16, 56, 50, 12
    CHECKBOX       "Encadrer Domaine", IDD_END,  8, 83,102, 12
    PUSHBUTTON     "Dessiner"            IDOK,  8,103, 40, 14, WS_GROUP
    PUSHBUTTON     "Annuler"         IDCANCEL, 70,103, 40, 14, WS_GROUP
        }

DrawMeta DIALOG 10, 10, 120,160
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT        "Dessiner l'attracteur",  -1, 30,  8,126, 12
    CHECKBOX     "Principe deterministe", IDD_MD,8,29,102, 12
    CHECKBOX     "Principe chaos",     IDD_MC,  8, 40,102, 12
    LTEXT        "domaine",                -1, 70, 78, 72, 12
    LTEXT        "(origine-axex-axey)",    -1, 26, 88, 72, 12
    ICON         "IDI_F_COL"               -1,  8,  8,  0,  0
    GROUPBOX     "Domaine"                 -1,  8, 52,102, 48
     RADIOBUTTON "100 x 100"         IDD_DM1, 16, 64, 50, 12
     RADIOBUTTON "Selectionner"      IDD_DM2, 16, 76, 50, 12
    LTEXT        "Nb reproductions :",     -1,  8,118,100, 12
    EDITTEXT                        IDD_FNAME, 73,118, 15, 12, ES_AUTOHSCROLL
    PUSHBUTTON   "Dessiner"              IDOK,  8,138, 40, 14, WS_GROUP
    PUSHBUTTON   "Annuler"           IDCANCEL, 70,138, 40, 14, WS_GROUP
    CHECKBOX     "Encadrer Domaine",  IDD_END,  8,103,102, 12
    }

NewCodeRan DIALOG 10,  7, 140, 206
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT          "Nouveau code-IFS",       -1, 52, 12,126, 12
    ICON           "IDI_FRACTALS"            -1,  8,  8,  0,  0
    LTEXT          "Nom : "                  -1,  8, 40,100, 12
    EDITTEXT                         IDD_FRNAM, 32, 40,100, 12, ES_AUTOHSCROLL

    LTEXT          "Transformations     :"   -1,  8, 55,160, 12
    PUSHBUTTON     "OK-tr"           IDD_NEXT,103, 84, 29, 12
    LTEXT          "a"                       -1,  4, 70, 15, 12
    LTEXT          "b"                       -1, 36, 70, 15, 12
    LTEXT          "c"                       -1,  4, 84, 15, 12
    LTEXT          "d"                       -1, 36, 84, 15, 12
    LTEXT          "e"                       -1, 68, 70, 15, 12
    LTEXT          "f"                       -1, 68, 84, 15, 12
    LTEXT          "p"                       -1,105, 70, 15, 12
    LTEXT          "Densite :"               -1, 68,150, 30, 12

    EDITTEXT                        IDD_TRANA, 11, 70, 20, 12
    EDITTEXT                        IDD_TRANB, 43, 70, 20, 12
    EDITTEXT                        IDD_TRANC, 11, 84, 20, 12
    EDITTEXT                        IDD_TRAND, 43, 84, 20, 12
    EDITTEXT                        IDD_TRANE, 75, 70, 20, 12
    EDITTEXT                        IDD_TRANF, 75, 84, 20, 12
    EDITTEXT                        IDD_PROBA,112, 70, 20, 12
    EDITTEXT                        IDD_DENSI,112,150, 20, 12

    GROUPBOX       "Mono-couleur"            -1,  4,100, 54, 95
     RADIOBUTTON   "Noir"            IDD_WHITE,  8,110, 40, 12, TABGRP
     RADIOBUTTON   "Bleu"             IDD_BLUE,  8,120, 40, 12
     RADIOBUTTON   "Vert"            IDD_GREEN,  8,130, 40, 12
     RADIOBUTTON   "Cyan "            IDD_CYAN,  8,140, 40, 12
     RADIOBUTTON   "Rouge"             IDD_RED,  8,150, 40, 12
     RADIOBUTTON   "Magenta"         IDD_MAGEN,  8,160, 40, 12
     RADIOBUTTON   "Jaune"           IDD_YELLO,  8,170, 40, 12
     RADIOBUTTON   "Gris"             IDD_GRAY,  8,180, 40, 12
    GROUPBOX       "Multi-couleur"           -1, 68,100, 64, 46, WS_GROUP
     RADIOBUTTON   "Gris ombre"      IDD_SHGRA, 72,110, 55, 12, TABGRP
     RADIOBUTTON   "Vert ombre"      IDD_SHGRE, 72,120, 55, 12
     RADIOBUTTON   "Rouge ombre"     IDD_SHRED, 72,130, 55, 12

    DEFPUSHBUTTON  "Accepter Code"       IDOK, 68,167, 65, 12, WS_GROUP
    PUSHBUTTON     "Annuler"         IDCANCEL, 68,182, 65, 12, WS_GROUP
    }

ModCodeRan DIALOG 10,  7, 140, 206
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT          "Modifier code-IFS",      -1, 52, 12,126, 12
    ICON           "IDI_FRACTALS"            -1,  8,  8,  0,  0
    LTEXT          "Nom : "                  -1,  8, 40,100, 12
    EDITTEXT                         IDD_FRNAM, 32, 40,100, 12, ES_AUTOHSCROLL
```

```
    LTEXT          "Transformations     :"  -1,  8, 55,160, 12
    PUSHBUTTON     "OK-tr"              IDD_NEXT,103, 84, 29, 12
    LTEXT          "a"                        -1,  4, 70, 15, 12
    LTEXT          "b"                        -1, 36, 70, 15, 12
    LTEXT          "c"                        -1,  4, 84, 15, 12
    LTEXT          "d"                        -1, 36, 84, 15, 12
    LTEXT          "e"                        -1, 68, 70, 15, 12
    LTEXT          "f"                        -1, 68, 84, 15, 12
    LTEXT          "p"                        -1,105, 70, 15, 12
    LTEXT          "Densite :"                -1, 68,150, 30, 12

    EDITTEXT                          IDD_TRANA, 11, 70, 20, 12
    EDITTEXT                          IDD_TRANB, 43, 70, 20, 12
    EDITTEXT                          IDD_TRANC, 11, 84, 20, 12
    EDITTEXT                          IDD_TRAND, 43, 84, 20, 12
    EDITTEXT                          IDD_TRANE, 75, 70, 20, 12
    EDITTEXT                          IDD_TRANF, 75, 84, 20, 12
    EDITTEXT                          IDD_PROBA,112, 70, 20, 12
    EDITTEXT                          IDD_DENSI,112,150, 20, 12

    GROUPBOX       "Mono-couleur"             -1,  4,100, 54, 95
     RADIOBUTTON   "Noir"             IDD_WHITE,  8,110, 40, 12, TABGRP
     RADIOBUTTON   "Bleu"              IDD_BLUE,  8,120, 40, 12
     RADIOBUTTON   "Vert"             IDD_GREEN,  8,130, 40, 12
     RADIOBUTTON   "Cyan "             IDD_CYAN,  8,140, 40, 12
     RADIOBUTTON   "Rouge"              IDD_RED,  8,150, 40, 12
     RADIOBUTTON   "Magenta"          IDD_MAGEN,  8,160, 40, 12
     RADIOBUTTON   "Jaune"            IDD_YELLO,  8,170, 40, 12
     RADIOBUTTON   "Gris"              IDD_GRAY,  8,180, 40, 12
    GROUPBOX       "Multi-couleur"            -1, 68,100, 64, 46, WS_GROUP
     RADIOBUTTON   "Gris ombre"       IDD_SHGRA, 72,110, 55, 12, TABGRP
     RADIOBUTTON   "Vert ombre"       IDD_SHGRE, 72,120, 55, 12
     RADIOBUTTON   "Rouge ombre"      IDD_SHRED, 72,130, 55, 12

    DEFPUSHBUTTON  "Accepter Code"        IDOK, 68,167, 65, 12, WS_GROUP
    PUSHBUTTON     "Annuler"           IDCANCEL, 68,182, 65, 12, WS_GROUP
    }

NewCodeMet DIALOG 10, 10, 140, 141
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT          "Nouveau Code Meta-IFS", -1, 42, 12,126, 12
    ICON           "IDI_METFRACT"             -1,  8,  8,  0,  0
    LTEXT          "Nom : "                   -1,  8, 40,100, 12
    EDITTEXT                          IDD_FRNAM, 32, 40,100, 12, ES_AUTOHSCROLL

    LTEXT          "Transformations     :"  -1,  8, 55,160, 12
    PUSHBUTTON     "OK-tr"              IDD_NEXT,103, 84, 29, 12
    LTEXT          "a"                        -1,  4, 70, 15, 12
    LTEXT          "b"                        -1, 36, 70, 15, 12
    LTEXT          "c"                        -1,  4, 84, 15, 12
    LTEXT          "d"                        -1, 36, 84, 15, 12
    LTEXT          "e"                        -1, 68, 70, 15, 12
    LTEXT          "f"                        -1, 68, 84, 15, 12
    LTEXT          "p"                        -1,105, 70, 15, 12
    LTEXT          "Densite :"                -1,  8,100, 30, 12

    EDITTEXT                          IDD_TRANA, 11, 70, 20, 12
    EDITTEXT                          IDD_TRANB, 43, 70, 20, 12
    EDITTEXT                          IDD_TRANC, 11, 84, 20, 12
    EDITTEXT                          IDD_TRAND, 43, 84, 20, 12
    EDITTEXT                          IDD_TRANE, 75, 70, 20, 12
    EDITTEXT                          IDD_TRANF, 75, 84, 20, 12
    EDITTEXT                          IDD_PROBA,112, 70, 20, 12
    EDITTEXT                          IDD_DENSI, 43,100, 20, 12

    DEFPUSHBUTTON  "Accepter Code"        IDOK,  4,120, 60, 12, WS_GROUP
    PUSHBUTTON     "Annuler"           IDCANCEL, 73,120, 60, 12, WS_GROUP
    }

ModCodeMet DIALOG 10, 10, 140, 141
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT          "Modifier Code Meta-IFS",-1, 42, 12,126, 12
    ICON           "IDI_METFRACT"             -1,  8,  8,  0,  0
    LTEXT          "Nom : "                   -1,  8, 40,100, 12
    EDITTEXT                          IDD_FRNAM, 32, 40,100, 12, ES_AUTOHSCROLL

    LTEXT          "Transformations     :"  -1,  8, 55,160, 12
    PUSHBUTTON     "OK-tr"              IDD_NEXT,103, 84, 29, 12
    LTEXT          "a"                        -1,  4, 70, 15, 12
    LTEXT          "b"                        -1, 36, 70, 15, 12
    LTEXT          "c"                        -1,  4, 84, 15, 12
    LTEXT          "d"                        -1, 36, 84, 15, 12
    LTEXT          "e"                        -1, 68, 70, 15, 12
    LTEXT          "f"                        -1, 68, 84, 15, 12
    LTEXT          "p"                        -1,105, 70, 15, 12
    LTEXT          "Densite :"                -1,  8,100, 30, 12

    EDITTEXT                          IDD_TRANA, 11, 70, 20, 12
    EDITTEXT                          IDD_TRANB, 43, 70, 20, 12
    EDITTEXT                          IDD_TRANC, 11, 84, 20, 12
    EDITTEXT                          IDD_TRAND, 43, 84, 20, 12
    EDITTEXT                          IDD_TRANE, 75, 70, 20, 12
    EDITTEXT                          IDD_TRANF, 75, 84, 20, 12
    EDITTEXT                          IDD_PROBA,112, 70, 20, 12
    EDITTEXT                          IDD_DENSI, 43,100, 20, 12

    DEFPUSHBUTTON  "Accepter Code"        IDOK,  4,120, 60, 12, WS_GROUP
    PUSHBUTTON     "Annuler"           IDCANCEL, 73,120, 60, 12, WS_GROUP
    }

ColorFig DIALOG 10,10,140,200
    STYLE WS_POPUP | WS_DLGFRAME
    {
    LTEXT          "Modifier Couleurs"      ,-1, 42, 12,126, 12
    ICON           "IDI_FIGURE"               -1,  8,  8,  0,  0
    GROUPBOX       "Couleur Surface"          -1,  4, 16+15, 67,165
     CTEXT    ""                      IDD_PAINT, 10,134+40, 55, 15
     CONTROL "Rou.",-1,"static",     SS_CENTER, 10,  4+40, 15,  8
     CONTROL "",10,"scrollbar",  SBS_VERT_TAB, 10, 16+40, 15,100
     CONTROL "",13,"static",         SS_CENTER, 10,120+40, 15,  8
     CONTROL "Vert",-1,"static",     SS_CENTER, 30,  4+40, 15,  8
     CONTROL "",11,"scrollbar",  SBS_VERT_TAB, 30, 16+40, 15,100
     CONTROL "",14,"static",         SS_CENTER, 30,120+40, 15,  8
     CONTROL "Bleu",-1,"static",     SS_CENTER, 50,  4+40, 15,  8
     CONTROL "",12,"scrollbar",  SBS_VERT_TAB, 50, 16+40, 15,100
     CONTROL "",15,"static",         SS_CENTER, 50,120+40, 15,  8
    GROUPBOX       "Couleur Bord"             -1, 80, 16+15, 54, 95
     RADIOBUTTON   "Noir"                     50, 86, 26+15, 40, 12, TABGRP
     RADIOBUTTON   "Bleu"                     51, 86, 36+15, 40, 12
     RADIOBUTTON   "Vert"                     52, 86, 46+15, 40, 12
     RADIOBUTTON   "Cyan "                    53, 86, 56+15, 40, 12
     RADIOBUTTON   "Rouge"                    54, 86, 66+15, 40, 12
     RADIOBUTTON   "Magenta"                  55, 86, 76+15, 40, 12
     RADIOBUTTON   "Jaune"                    56, 86, 86+15, 40, 12
     RADIOBUTTON   "Gris"                     57, 86, 96+15, 40, 12
    DEFPUSHBUTTON  "OK"                   IDOK, 80,143+05, 54, 12, WS_GROUP
    PUSHBUTTON     "Annuler"           IDCANCEL, 80,158+05, 54, 12, WS_GROUP
    }
```

```
LoadIFS1 DIALOG 10,10,180,120
   STYLE WS_POPUP | WS_DLGFRAME
   {
   LTEXT          "Ouvrir fichier:",       -1,  52,  4, 126,12
   ICON           "IDI_F_LOA"              -1,   8,  8,  0,  0
   EDITTEXT                           IDD_FNAME,  52, 15,116, 12, ES_AUTOHSCROLL
   LTEXT          "Fichiers dans",         -1,  52, 40, 90, 12
   LTEXT          "",                 IDD_FPATH,  94, 40,144, 12
   LISTBOX                            IDD_FLIST,  52, 54,116, 58, WS_TABSTOP |
                                                                  WS_VSCROLL
   DEFPUSHBUTTON "Ouvrir"                  IDOK,   8, 54, 40, 14, WS_GROUP
   PUSHBUTTON    "Annuler"              IDCANCEL,   8, 74, 40, 14, WS_GROUP
   }

SaveIFS1 DIALOG 10, 10, 160, 54
   STYLE WS_POPUP | WS_DLGFRAME
   {
   LTEXT          "Sauvgarde IFS:",        -1, 52,  4,126, 12
   ICON           "IDI_F_SAV"              -1,  8,  8,  0,  0
   EDITTEXT                           IDD_FNAME, 52, 15,100, 12, ES_AUTOHSCROLL
   DEFPUSHBUTTON "OK"                      IDOK, 20, 35, 40, 14, WS_GROUP
   PUSHBUTTON    "Annuler"              IDCANCEL, 90, 35, 40, 14, WS_GROUP
   }
```

```
/*************************************************************************/
/***      E D I T E U R   D ' I M A G E S   F R A C T A L E S      ***/
/*************************************************************************/

#include <string.h>

#include <windows.h>
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

#include "winfract.h"
#include "fr.h"

long FAR PASCAL _export WndProc (HWND, WORD, WORD, LONG) ;

/*************************************************************************/
/* Variables globales :                                                  */
/*                                                                       */

   char szAppName[] = "Fractals" ;   // nom de l'application
   HANDLE hInst;                     // Handle pour les icones
/*                                                                       */
/* résultats des 'Message-boxes' :                                       */
/*                                                                       */
   DWORD IFSColor = RGB(0,0,0);   // Couleur du IFS
   short color[3]={0,0,0};
   short IFSSelec = 1;            // Numéro de l'IFS sélectionné
   short MIFSSele = 1;            // Numéro du Meta-IFS sélectionné
   short IFSInsSe = 1;            // Numéro  - IFS à insérer au Meta-IFS
   short IFSAppSe = 1;            // Numéro  - IFS pour appliquer une transf.
   short IFSDraw  = 1;            // Mode  - dessiner (-1,1,2) IFS
   short MIFSDraw = 1;            // Mode  - dessiner le Meta-IFS
   short MIFSTyDr = 1;            // Type d'algorithme pour le Meta-IFS
   short EncDom   = 1;            // Encadrer le domaine
   int nprod;
/*                                                                       */
/* pointeurs vers les objets :                                           */
/*                                                                       */
   short numifs = 0;
   ifs      *ObjetIFS1=0;
   Meta_ifs *ObjetMIFS1=0;
   Polygone    *pol=0;
/*                                                                       */
/* chaines de caractères contenants les états affichés à la fenêtre      */
/*                                                                       */
   char Dim100[15]="-          ",
        Dim400[15]="-          ",
        NomPTH[15]="           ",
        NomIFS[15]="           ",
        ori[20]="",
        axi[20]="",
        axj[20]="",
        NomMIFS[15]="          ",
        Comp[6][15]={"","","","","",""};

/*                                                                       */
/* variables auxiliaires :                                               */
/*                                                                       */
   float tra[4]={0,0,0,0},
         trb[4]={0,0,0,0},
         trc[4]={0,0,0,0},
         trd[4]={0,0,0,0},
         tre[4]={0,0,0,0},
         trf[4]={0,0,0,0},
         trp[4]={0,0,0,0};
   int   nt = 0;
   float tma[4]={0,0,0,0},
         tmb[4]={0,0,0,0},
         tmc[4]={0,0,0,0},
         tmd[4]={0,0,0,0},
         tme[4]={0,0,0,0},
         tmf[4]={0,0,0,0},
         tmp[4]={0,0,0,0};
   int   mt = 0;
/*                                                                       */
/*************************************************************************/


HBITMAP hBitmap;
HBRUSH hBrush;


 int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, int nCmdShow )

/*************************************************************************/
/* définition/création de la fenêtre                                     */
{
    WNDCLASS    wndClass;
    MSG         msg;
    HWND        hwnd;

    hBitmap = LoadBitmap(hInstance,"scan");
    hBrush = CreatePatternBrush(hBitmap);

    if ( !hPrevInstance )
    {
        wndClass.style          = CS_HREDRAW | CS_VREDRAW ;
        wndClass.lpfnWndProc    = WndProc;
        wndClass.cbClsExtra     = 0;
        wndClass.cbWndExtra     = 0;
        wndClass.hInstance      = hInstance;
        wndClass.hIcon          = LoadIcon(hInstance,"IDI_FRACTALS");
        wndClass.hCursor        = LoadCursor(hInstance, IDC_ARROW );
        wndClass.hbrBackground  = GetStockObject(BLACK_BRUSH);
        wndClass.lpszMenuName   = szAppName;
        wndClass.lpszClassName  = szAppName;

        RegisterClass(&wndClass) ;
    }
    hInst = hInstance;

    hwnd = CreateWindow(szAppName,"Editeur d'images fractales",
                        WS_OVERLAPPEDWINDOW,0,0,640,
                        480,NULL,NULL,hInstance,NULL);

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg );
        DispatchMessage(&msg );
    }
    return msg.wParam;
}
```

```
// ------------------------------
   void DisableMenus(HMENU hMenu)
// ------------------------------
{
   EnableMenuItem(hMenu,IDM_ABO,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_RE1,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_RE2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_RE3,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_QUI,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_LOB3,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_BO13,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_BOX3,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_SET1,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_MOD1,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_SEL1,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_CHC1,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_DRA1,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_ZOO1,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_SET2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_MOD2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_SEA2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_INH2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_INS2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_APH2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_APP2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_MCO2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_DRA2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_PTH1,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_PTH2,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_PTH3,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_PTH4,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_DRA3,MF_GRAYED);
   EnableMenuItem(hMenu,IDM_ATT3,MF_GRAYED);
};

// ------------------------------
   void EnableMenus(HMENU hMenu)
// ------------------------------
{
   EnableMenuItem(hMenu,IDM_ABO,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_RE1,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_RE2,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_RE3,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_QUI,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_LOB3,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_BO13,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_BOX3,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_SET1,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_SEL1,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_SET2,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_SEA2,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_PTH1,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_PTH2,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_PTH3,MF_ENABLED);
   EnableMenuItem(hMenu,IDM_PTH4,MF_ENABLED);
   if (ObjetIFS1!=0)
      {
      EnableMenuItem(hMenu,IDM_MOD1,MF_ENABLED);
      EnableMenuItem(hMenu,IDM_CHC1,MF_ENABLED);
      EnableMenuItem(hMenu,IDM_DRA1,MF_ENABLED);
      EnableMenuItem(hMenu,IDM_ZOO1,MF_ENABLED);
      };
   if (pol!=0)
      {
      EnableMenuItem(hMenu,IDM_DRA3,MF_ENABLED);
      EnableMenuItem(hMenu,IDM_ATT3,MF_ENABLED);
      };
   if (ObjetMIFS1!=0)
      {
      EnableMenuItem(hMenu,IDM_MOD2,MF_ENABLED);
      if (ObjetIFS1 != 0)
         {
         EnableMenuItem(hMenu,IDM_INS2,MF_ENABLED);
         EnableMenuItem(hMenu,IDM_APP2,MF_ENABLED);
         };
      if (pol != 0)
         {
         EnableMenuItem(hMenu,IDM_INH2,MF_ENABLED);
         EnableMenuItem(hMenu,IDM_APH2,MF_ENABLED);
         };
      if ((ObjetMIFS1->PtrDernier()) != 0)
         {
         EnableMenuItem(hMenu,IDM_MCO2,MF_ENABLED);
         EnableMenuItem(hMenu,IDM_DRA2,MF_ENABLED);
         };
      };
};

// ------------------------------
   void AfficherDonnees(HDC hdc)
// ------------------------------
// mise à jour de l'écran
{
 LOGPEN lPen = {PS_NULL,0,RGB(0,0,0)};
 HPEN   hPen;
 int ii,i,j;

 /*** effacer affichage composition ***/
 hPen = CreatePenIndirect(&lPen);
 hPen = SelectObject(hdc,hPen);
 Rectangle(hdc, 80,169,208,184);
 Rectangle(hdc, 80,205,208,222);
 Rectangle(hdc, 80,320,208,337);
 Rectangle(hdc,113,242,208,259);
 Rectangle(hdc,113,258,208,275);
 Rectangle(hdc,113,274,208,291);
 Rectangle(hdc,30,360,200,416);
 DeleteObject(SelectObject(hdc,hPen));

 /*** regarnir la fenêtre ***/
 TextOut(hdc, 80,169,NomPTH,strlen(NomPTH));
 TextOut(hdc, 80,205,NomIFS,strlen(NomIFS));
 TextOut(hdc, 80,320,NomMIFS,strlen(NomMIFS));
 TextOut(hdc,113,242,ori,strlen(ori));
 TextOut(hdc,113,258,axi,strlen(axi));
 TextOut(hdc,113,274,axj,strlen(axj));
 TextOut(hdc,135,81,Dim100,5);
 TextOut(hdc,135,118,Dim400,5);

 for (ii=0;ii<numifs;ii++)
     {
     if (ii>2)
        {i=71;j=3;}
     else
        {i=0;j=0;};

     TextOut(hdc,35+i,360+16*(ii-j),Comp[ii],strlen(Comp[ii]));
     };
};
```

```
// ------------------------------------------
   void PaintWindow (HWND hCtrlBlk,short *color)
// ------------------------------------------
// impression d'un rectangle coloré pour le
// choix des couleurs.
{
 HDC hdc;
 HBRUSH hBrush;
 RECT rect;

 InvalidateRect(hCtrlBlk,NULL,TRUE);
 UpdateWindow(hCtrlBlk);

 hdc = GetDC(hCtrlBlk);
 GetClientRect(hCtrlBlk,&rect);
 hBrush = CreateSolidBrush(RGB(color[0],color[1],color[2]));
 hBrush = SelectObject(hdc,hBrush);
 Rectangle(hdc,rect.left,rect.top,rect.right,rect.bottom);
 DeleteObject(SelectObject(hdc,hBrush));
 ReleaseDC(hCtrlBlk,hdc);
};

// ------------------------------------------------------
   BOOL FAR PASCAL _export Colorbox(HWND hDlg, WORD message,
                                    WORD wParam, LONG lParam)
// ------------------------------------------------------
// Dialogue : choix des couleurs
{
    HWND         hwndParent, hCtrl;
    static HWND  hCtrlBlk;
    short        nCtrlID, nIndex,nID;

    switch (message)
    {
        case WM_INITDIALOG :
             for (nCtrlID = 10; nCtrlID < 13; nCtrlID++)
                 {
                 hCtrl = GetDlgItem (hDlg,nCtrlID);
                 SetScrollRange(hCtrl,SB_CTL,0,255,FALSE);
                 SetScrollPos  (hCtrl,SB_CTL,color[nCtrlID-10],FALSE);
                 }
             hCtrlBlk = GetDlgItem(hDlg,IDD_PAINT);
             return FALSE;

        case WM_VSCROLL :
           hCtrl = HIWORD(lParam);
           nCtrlID = GetWindowWord(hCtrl,GWW_ID);
           nIndex = nCtrlID - 10;
           hwndParent = GetParent(hDlg);

           switch(wParam)
             {
             case SB_PAGEDOWN :
                  color[nIndex] += 15;
             case SB_LINEDOWN :
                  if (color[nIndex]<255)
                     color[nIndex] = color[nIndex]+1;
                  else
                     color[nIndex] = 255;
                  break;
             case SB_PAGEUP :
                  color[nIndex] -= 15;
             case SB_LINEUP :
                  if (color[nIndex]>0)
                     color[nIndex] = color[nIndex]-1;
                  else
                     color[nIndex] = 0;
                  break;
             case SB_TOP :
                  color[nIndex] = 0;
                  break;
             case SB_BOTTOM :
                  color[nIndex] = 255;
                  break;
             case SB_THUMBPOSITION :
             case SB_THUMBTRACK :
                  color[nIndex] = LOWORD(lParam);
                  break;
             default :
                  return FALSE;
             }
           SetScrollPos(hCtrl,SB_CTL,color[nIndex],TRUE);
           SetDlgItemInt(hDlg,nCtrlID+3,color[nIndex],FALSE);
           PaintWindow(hCtrlBlk,color);
           return TRUE;
        case WM_COMMAND :
           switch(wParam)
             {
             case IDOK :
                  IFSColor = RGB(color[0],color[1],color[2]);
                  ObjetIFS1->couleur = IFSColor;
                  EndDialog(hDlg,TRUE);
                  return TRUE;
             }
             break;
    }
    return FALSE;
};

// ------------------------------------------------------
   BOOL FAR PASCAL _export LoadIFS1 (HWND hDlg, WORD message,
                                     WORD wParam, LONG lParam)
// ------------------------------------------------------
// Dialogue : charger les données d'un IFS - pas implémenté
{
    /**** informations about the program ****/
    switch (message)
    {
        case WM_INITDIALOG :
             return TRUE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDOK :
             case IDCANCEL :
                  EndDialog (hDlg,FALSE);
                  return TRUE;
             }
             break;
    }
    return FALSE;
};

// ------------------------------------------------------
   BOOL FAR PASCAL _export SaveIFS1 (HWND hDlg, WORD message,
                                     WORD wParam, LONG lParam)
// ------------------------------------------------------
// Dialogue : Sauvgarde des données d'un IFS - non implémenté
{
```

```
    switch (message)
    {
        case WM_INITDIALOG :
             return TRUE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDOK :
             case IDCANCEL :
                  EndDialog (hDlg,FALSE);
                  return TRUE;
             }
             break;
    }
    return FALSE;
};

// ------------------------------------------------------------
   BOOL FAR PASCAL _export Aboutbox (HWND hDlg, WORD message,
                                     WORD wParam, LONG lParam)
// ------------------------------------------------------------
// Dialogue : Imformations sur le programme
{
    /**** informations about the program ****/
    switch (message)
    {
        case WM_INITDIALOG :
             return TRUE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDOK :
             case IDCANCEL :
                  EndDialog (hDlg,TRUE);
                  return TRUE;
             }
             break;
    }
    return FALSE;
};

// ------------------------------------------------------------
   BOOL FAR PASCAL _export SelMetIFS (HWND hDlg, WORD message,
                                      WORD wParam, LONG lParam)
// ------------------------------------------------------------
// Dialogue : choix d'un Meta-IFS prédéfini
{
    static short aux;
    switch (message)
    {
        case WM_INITDIALOG :
             return FALSE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDD_IFS1:
             case IDD_IFS2:
             case IDD_IFS3:
             case IDD_IFS5:
             case IDD_IFS6:
                  aux = wParam-19;
                  CheckRadioButton(hDlg,IDD_IFS1,IDD_IFS6,wParam);
                  return FALSE;
```

```
case IDOK:
     MIFSSele = aux;
     numifs=0;
     switch(MIFSSele)
           {
           case 1:
              sprintf(NomMIFS,"Sierpinski\0");
              /*** code du traingle de Sierpinski IDD_IFS1 ***/
              tma[0]=0.49;tma[1]=0.49;tma[2]=0.49;
              tmb[0]=   0;tmb[1]=   0;tmb[2]=   0;
              tmc[0]=   0;tmc[1]=   0;tmc[2]=   0;
              tmd[0]=0.49;tmd[1]=0.49;tmd[2]=0.49;
              tme[0]=   0;tme[1]=  51;tme[2]=  25;
              tmf[0]=   0;tmf[1]=   0;tmf[2]=  51;
              tmp[0]= 0.3;tmp[1]= 0.3;tmp[2]= 0.3;
              mt=3;
              break;
           case 2:
              sprintf(NomMIFS,"Liste IFS \0");
              /*** code d'une liste d'IFS ***/
              mt=0;
              break;
           case 3:
              sprintf(NomMIFS,"Farn      \0");
              /*** code d'un farn IDD_IFS3 ***/
              tma[0]=-0.15;tma[1]= 0.85;tma[2]= 0.20;tma[3]=    0;
              tmb[0]= 0.28;tmb[1]= 0.04;tmb[2]=-0.21;tmb[3]=    0;
              tmc[0]= 0.26;tmc[1]=-0.04;tmc[2]= 0.18;tmc[3]=    0;
              tmd[0]= 0.24;tmd[1]= 0.85;tmd[2]= 0.22;tmd[3]= 0.16;
              tme[0]=   58;tme[1]=  7.5;tme[2]=   40;tme[3]=   50;
              tmf[0]= 0.44;tmf[1]=   17;tmf[2]= 1.60;tmf[3]=    0;
              tmp[0]= 0.18;tmp[1]=  0.6;tmp[2]= 0.18;tmp[4]= 0.04;
              mt=4;
              break;
           case 5:
              sprintf(NomMIFS,"Arbre-1   \0");
              /*** code d'un arbre 1 IDD_IFS5 ***/
              tma[0]=0.01;tma[1]= 0.55;tma[2]=  0.7;
              tmb[0]= 0.0;tmb[1]=-0.24;tmb[2]= 0.10;
              tmc[0]= 0.0;tmc[1]= 0.24;tmc[2]=-0.10;
              tmd[0]=0.45;tmd[1]= 0.65;tmd[2]=  0.7;
              tme[0]=  50;tme[1]=   47;tme[2]=  5.5;
              tmf[0]=  55;tmf[1]=    9;tmf[2]=   10;
              tmp[0]=0.04;tmp[1]=  0.5;tmp[2]= 0.46;

              mt=3;
              break;
           default:
              sprintf(NomMIFS,"Arbre-2   \0");
              /*** code d'un arbre 2 IDD_IFS6 ***/
              tma[0]=0.01;tma[1]= 0.55;tma[2]=  0.7;tma[3]=  0.6;
              tmb[0]= 0.0;tmb[1]=-0.24;tmb[2]= 0.20;tmb[3]= 0.05;
              tmc[0]= 0.0;tmc[1]= 0.24;tmc[2]=-0.20;tmc[3]=-0.05;
              tmd[0]=0.45;tmd[1]= 0.65;tmd[2]=  0.7;tmd[3]=  0.6;
              tme[0]=  48;tme[1]=   46;tme[2]= -5.5;tme[3]= 14.8;
              tmf[0]=  55;tmf[1]=   -1;tmf[2]=   19;tmf[3]=   20;
              tmp[0]=0.04;tmp[1]=  0.3;tmp[2]=  0.3;tmp[3]= 0.26;
              mt=4;
              break;
           };

     /*** initialiser le Meta IFS ***/
     delete ObjetMIFS1;
     ObjetMIFS1=new Meta_ifs(mt,tma,tmb,tmc,tmd,tme,tmf,tmp,1);
     EndDialog (hDlg,TRUE);
     return TRUE;
```

```
            case IDCANCEL:
                 MIFSSele = -1;
                 EndDialog (hDlg,FALSE);
                 return FALSE;
            }
            break;
    }
    return FALSE;
};

// ------------------------------------------------------------
   BOOL FAR PASCAL _export SelAppIFS (HWND hDlg, WORD message,
                                      WORD wParam, LONG lParam)
// ------------------------------------------------------------
// Dialogue : Application d'une transformation d'un Meta-IFS à
//            l'IFS courant
{
    static short aux;
    static char res[10]="1";
    switch (message)
    {
        case WM_INITDIALOG :
             SetDlgItemText(hDlg,IDD_FNAME,res);
             return FALSE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDD_FNAME:
                  GetDlgItemText(hDlg,IDD_FNAME,res,5);
                  IFSAppSe = atoi(res);
                  return FALSE;

             case IDOK:
                  EndDialog (hDlg,TRUE);
                  return TRUE;

             case IDCANCEL:
                  IFSAppSe = -1;
                  EndDialog (hDlg,FALSE);
                  return FALSE;
             }
             break;
    }
    return FALSE;
};

// ------------------------------------------------------------
   BOOL FAR PASCAL _export SelectIFS (HWND hDlg, WORD message,
                                      WORD wParam, LONG lParam)
// ------------------------------------------------------------
// Dialogue : choisir un IFS prédéfini
{
    static short aux;
    switch (message)
    {
        case WM_INITDIALOG :
             return FALSE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDD_IFS1:
             case IDD_IFS2:
             case IDD_IFS3:
             case IDD_IFS4:
             case IDD_IFS5:
```

```
             case IDD_IFS6:
                  aux = wParam - 19;
                  CheckRadioButton(hDlg,IDD_IFS1,IDD_IFS6,wParam);
                  return FALSE;

             case IDOK:
                  IFSSelec = aux;
                  switch(IFSSelec)
                       {
                       case 1:
                          sprintf(NomIFS,"Sierpinski\0");
                          /*** code du traingle de Sierpinski IDD_IFS1 ***/
                          tra[0]=0.49;tra[1]=0.49;tra[2]=0.49;
                          trb[0]=   0;trb[1]=   0;trb[2]=   0;
                          trc[0]=   0;trc[1]=   0;trc[2]=   0;
                          trd[0]=0.49;trd[1]=0.49;trd[2]=0.49;
                          tre[0]=   0;tre[1]=  51;tre[2]=  25;
                          trf[0]=   0;trf[1]=   0;trf[2]=  51;
                          trp[0]= 0.33;trp[1]= 0.34;trp[2]= 0.33;
                          nt=3;
                          break;
                       case 2:
                          sprintf(NomIFS,"Rectangle \0");
                          /*** code d'un rectangle IDD_IFS2 ***/
                          tra[0]=0.49;tra[1]=0.49;tra[2]=0.49;tra[3]=0.49;
                          trb[0]=   0;trb[1]=   0;trb[2]=   0;trb[3]=   0;
                          trc[0]=   0;trc[1]=   0;trc[2]=   0;trc[3]=   0;
                          trd[0]=0.49;trd[1]=0.49;trd[2]=0.49;trd[3]=0.49;
                          tre[0]=   0;tre[1]=  51;tre[2]=   0;tre[3]=  51;
                          trf[0]=   0;trf[1]=   0;trf[2]=  51;trf[3]=  51;
                          trp[0]= 0.3;trp[1]= 0.2;trp[2]= 0.2;trp[3]=0.3;
                          nt=4;
                          break;
                       case 3:
                          sprintf(NomIFS,"Farn      \0");
                          /*** code d'un farn IDD_IFS3 ***/
                          tra[0]=-0.15;tra[1]= 0.85;tra[2]= 0.20;tra[3]=    0;
                          trb[0]= 0.28;trb[1]= 0.04;trb[2]=-0.21;trb[3]=    0;
                          trc[0]= 0.26;trc[1]=-0.04;trc[2]= 0.18;trc[3]=    0;
                          trd[0]= 0.24;trd[1]= 0.85;trd[2]= 0.22;trd[3]= 0.16;
                          tre[0]=   58;tre[1]=    7.5;tre[2]=   40;tre[3]=   50

                          trf[0]= 0.44;trf[1]=   17;trf[2]= 1.60;trf[3]=    0;
                          trp[0]= 0.09;trp[1]=  0.8;trp[2]= 0.09;trp[3]= 0.02;
                          nt=4;
                          break;
                       case 4:
                          sprintf(NomIFS,"Feuille   \0");
                          /*** code d'une feuille IDD_IFS4 ***/
                          tra[0]= 0.44;tra[1]=-0.82;
                          trb[0]= 0.32;trb[1]= 0.16;
                          trc[0]=-0.07;trc[1]=-0.16;
                          trd[0]= 0.61;trd[1]= 0.81;
                          tre[0]=    0;tre[1]=   88;
                          trf[0]=   42;trf[1]=   10;
                          trp[0]=  0.3;trp[1]=  0.7;
                          nt=2;
                          break;
                       case 5:
                          sprintf(NomIFS,"Arbre-1   \0");
                          /*** code d'un arbre 1 IDD_IFS5 ***/
                          tra[0]=0.01;tra[1]= 0.55;tra[2]=  0.7;
                          trb[0]= 0.0;trb[1]=-0.24;trb[2]= 0.10;
                          trc[0]= 0.0;trc[1]= 0.24;trc[2]=-0.10;
                          trd[0]=0.45;trd[1]= 0.65;trd[2]=  0.7;
                          tre[0]=  50;tre[1]=   47;tre[2]=  5.5;
                          trf[0]=  55;trf[1]=    9;trf[2]=   10;
```

```
                    trp[0]= 0.1;trp[1]=  0.4;trp[2]=  0.5;
                    nt=3;
                    break;
                default:
                    sprintf(NomIFS,"Arbre-2   \0");
                    /*** code d'un arbre 2 IDD_IFS6 ***/
                    tra[0]=0.01;tra[1]= 0.55;tra[2]=  0.7;tra[3]=  0.6;
                    trb[0]= 0.0;trb[1]=-0.24;trb[2]= 0.20;trb[3]= 0.05;
                    trc[0]= 0.0;trc[1]= 0.24;trc[2]=-0.20;trc[3]=-0.05;
                    trd[0]=0.45;trd[1]= 0.65;trd[2]=  0.7;trd[3]=  0.6;
                    tre[0]=  48;tre[1]=   46;tre[2]= -5.5;tre[3]= 14.8;
                    trf[0]=  55;trf[1]=   -1;trf[2]=   19;trf[3]=   20;
                    trp[0]=0.05;trp[1]=  0.3;trp[2]= 0.35;trp[3]=  0.3;
                    nt=4;
                    break;
                }
            delete ObjetIFS1;
            ObjetIFS1 = new ifs(nt,tra,trb,trc,trd,tre,trf,trp,1,1,0,0,0);

            EndDialog(hDlg,TRUE);
            return TRUE;

        case IDCANCEL:
            IFSSelec = -1;
            EndDialog(hDlg,FALSE);
            return FALSE;
        }
        break;
    }
    return FALSE;


-------------------------------------------------------------
BOOL FAR PASCAL _export DrawFract (HWND hDlg, WORD message,
                                   WORD wParam, LONG lParam)
-------------------------------------------------------------
Dialogue : Dessiner un IFS sur 100x100 ou domaine à choisir

    static short aux,aux2;
    switch (message)
    {
        case WM_INITDIALOG :
            return FALSE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDD_DI1:
                  aux = 1;
                  CheckRadioButton(hDlg,IDD_DI1,IDD_DI2,wParam);
                  return FALSE;
             case IDD_DI2:
                  aux = 2;
                  CheckRadioButton(hDlg,IDD_DI1,IDD_DI2,wParam);
                  return FALSE;
             case IDD_END:
                  CheckDlgButton(hDlg,wParam,
                                 IsDlgButtonChecked(hDlg,wParam) ? 0 : 1);
                  return FALSE;
             case IDOK:
                  IFSDraw = aux;
                  aux2 = IsDlgButtonChecked(hDlg,IDD_END);
                  if (aux2==0) EncDom = aux2;
                  else EncDom = 1;
                  EndDialog (hDlg,TRUE);
                  return TRUE;
             case IDCANCEL:
                  EncDom = 1;
                  IFSDraw = -1;
                  EndDialog (hDlg,FALSE);
                  return FALSE;
             }
             break;
        }
        return FALSE;
    };

    // -------------------------------------------------------------
       BOOL FAR PASCAL _export DrawMeta (HWND hDlg, WORD message,
                                         WORD wParam, LONG lParam)
    // -------------------------------------------------------------
    // Dialogue : Dessiner un Meta-IFS
    {
        static short aux,aux2=1,aux3;
        static char res[10]="1";
        switch (message)
        {
            case WM_INITDIALOG :
                 SetDlgItemText(hDlg,IDD_FNAME,res);
                 return FALSE;

            case WM_COMMAND :
               switch(wParam)
                 {
                 case IDD_MD:
                      aux2 = 1;
                      CheckDlgButton(hDlg,wParam,
                                     IsDlgButtonChecked(hDlg,wParam) ? 0 : 1);
                      return FALSE;

                 case IDD_MC:
                      aux2 = 2;
                      CheckDlgButton(hDlg,wParam,
                                     IsDlgButtonChecked(hDlg,wParam) ? 0 : 1);
                      return FALSE;
                 case IDD_END:
                      CheckDlgButton(hDlg,wParam,
                                     IsDlgButtonChecked(hDlg,wParam) ? 0 : 1);
                      return FALSE;

                 case IDD_DM1:
                      aux = 1;
                      CheckRadioButton(hDlg,IDD_DM1,IDD_DM2,wParam);
                      return FALSE;

                 case IDD_DM2:
                      aux = 2;
                      CheckRadioButton(hDlg,IDD_DM1,IDD_DM2,wParam);
                      return FALSE;

                 case IDD_FNAME:
                      GetDlgItemText(hDlg,IDD_FNAME,res,5);
                      nprod = atoi(res);
                      return FALSE;

                 case IDOK:
                      MIFSDraw = aux;
                      MIFSTyDr = aux2;
                      aux3 = IsDlgButtonChecked(hDlg,IDD_END);
                      if (aux3==0) EncDom = aux3;
                      else EncDom = 1;
                      EndDialog (hDlg,TRUE);
                      return TRUE;
```

```
            case IDCANCEL:
                 EncDom = 1;
                 MIFSDraw = -1;
                 EndDialog (hDlg,FALSE);
                 return FALSE;
            }
            break;
    }
    return FALSE;
};

// ------------------------------------------------------------
   BOOL FAR PASCAL _export NewCodeRan(HWND hDlg, WORD Message,
                                      WORD wParam, LONG lParam)
// ------------------------------------------------------------
// Dialogue : introduire le code d'un nouveau IFS
{
    static int ntrans,densite;
    static float a[8],b[8],c[8],d[8],e[8],f[8],p[8],sompro;
    static char geta[5],getb[5],getc[5],getd[5],gete[5],getf[5],getp[5],
                dens[5]="1",nom[15]="Ifs1\0",reset[4]="0.0";
    static short col[3]={0,0,0};

    switch(Message)
    {
        case WM_INITDIALOG :
             ntrans = 0;
             SetDlgItemText(hDlg,IDD_TRANA, reset);
             SetDlgItemText(hDlg,IDD_TRANB, reset);
             SetDlgItemText(hDlg,IDD_TRANC, reset);
             SetDlgItemText(hDlg,IDD_TRAND, reset);
             SetDlgItemText(hDlg,IDD_TRANE, reset);
             SetDlgItemText(hDlg,IDD_TRANF, reset);
             SetDlgItemText(hDlg,IDD_PROBA, reset);
             SetDlgItemText(hDlg,IDD_FRNAM, nom);
             SetDlgItemText(hDlg,IDD_DENSI, dens);
             return FALSE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDD_FRNAM :
                  GetDlgItemText(hDlg,IDD_FRNAM,nom,15);
                  return FALSE;
             case IDD_DENSI :
                  GetDlgItemText(hDlg,IDD_DENSI,dens,3);
                  return FALSE;
             case IDD_TRANA :
             case IDD_TRANB :
             case IDD_TRANC :
             case IDD_TRAND :
             case IDD_TRANE :
             case IDD_TRANF :
             case IDD_PROBA :
                  return FALSE;
             case IDD_NEXT  :
                  if (ntrans<7)
                     {
                     GetDlgItemText(hDlg,IDD_TRANA, geta,8);
                     GetDlgItemText(hDlg,IDD_TRANB, getb,8);
                     GetDlgItemText(hDlg,IDD_TRANC, getc,8);
                     GetDlgItemText(hDlg,IDD_TRAND, getd,8);
                     GetDlgItemText(hDlg,IDD_TRANE, gete,8);
                     GetDlgItemText(hDlg,IDD_TRANF, getf,8);
                     GetDlgItemText(hDlg,IDD_PROBA, getp,8);
                     a[ntrans]=atof(geta);
                     b[ntrans]=atof(getb);
                     c[ntrans]=atof(getc);
                     d[ntrans]=atof(getd);
                     e[ntrans]=atof(gete);
                     f[ntrans]=atof(getf);
                     p[ntrans]=atof(getp);
                     if (p[ntrans]>1) p[ntrans]=1;
                     if (p[ntrans]<0) p[ntrans]=0;
                     ntrans += 1;

                     SetDlgItemText(hDlg,IDD_TRANA, reset);
                     SetDlgItemText(hDlg,IDD_TRANB, reset);
                     SetDlgItemText(hDlg,IDD_TRANC, reset);
                     SetDlgItemText(hDlg,IDD_TRAND, reset);
                     SetDlgItemText(hDlg,IDD_TRANE, reset);
                     SetDlgItemText(hDlg,IDD_TRANF, reset);
                     SetDlgItemText(hDlg,IDD_PROBA, reset);
                     };
                  return FALSE;

        case IDD_WHITE :
             CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
             col[0]=0;
             col[1]=0;
             col[2]=0;
             return FALSE;
        case IDD_BLUE  :
             CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
             col[0]=0;
             col[1]=0;
             col[2]=255;
             return FALSE;
        case IDD_GREEN :
             CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
             col[0]=0;
             col[1]=255;
             col[2]=0;
             return FALSE;
        case IDD_CYAN  :
             CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
             col[0]=0;
             col[1]=255;
             col[2]=255;
             return FALSE;
        case IDD_RED   :
             CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
             col[0]=255;
             col[1]=0;
             col[2]=0;
             return FALSE;
        case IDD_MAGEN :
             CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
             col[0]=255;
             col[1]=0;
             col[2]=255;
             return FALSE;
        case IDD_YELLO :
             CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
             col[0]=255;
             col[1]=255;
             col[2]=0;
             return FALSE;
        case IDD_GRAY  :
             CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
```

```
                col[0]=127;
                col[1]=127;
                col[2]=127;
                return FALSE;
           case IDD_SHGRA :
           case IDD_SHGRE :
           case IDD_SHRED :
                CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
                return FALSE;

           case IDOK :
                densite=atoi(dens);
                color[0]=col[0];
                color[1]=col[1];
                color[2]=col[2];
                sprintf(NomIFS,nom);
                sompro=0;
                nt = ntrans;
                for (int i=0;i<nt;i++)
                    {
                    tra[i]=a[i];
                    trb[i]=b[i];
                    trc[i]=c[i];
                    trd[i]=d[i];
                    tre[i]=e[i];
                    trf[i]=f[i];
                    trp[i]=p[i];
                    sompro += trp[i];
                    };
                if (sompro>1)
                   {
                   for (int j=0;j<nt;j++)
                       trp[i]=1/(nt*1.0);
                   }
                else
                   trp[nt-1] = 1 - (sompro-trp[nt-1]);
                if (nt==0) {nt=1;trp[0]=1;};
                delete ObjetIFS1;
                ObjetIFS1 = new ifs(nt,tra,trb,trc,trd,tre,trf,trp,
                                    densite,2,color[0],color[1],color[2]);
                EndDialog(hDlg,TRUE);
                return TRUE;
           case IDCANCEL :
                EndDialog(hDlg,FALSE);
                return FALSE;
           }
           break;
    }
    return FALSE;
};

// ------------------------------------------------------------
   BOOL FAR PASCAL _export ModCodeRan(HWND hDlg, WORD Message,
                                      WORD wParam, LONG lParam)
// ------------------------------------------------------------
// Dialogue : introduire modification du code d'un IFS
{
    static int ntrans,densite;
    static float a[8],b[8],c[8],d[8],e[8],f[8],p[8],sompro;
    static char geta[5],getb[5],getc[5],getd[5],gete[5],getf[5],getp[5],
                dens[5]="1",nom[15]="Ifs1\0",reset[6]="0.0";
    static short col[3]={0,0,0};

    switch(Message)
    {
```

```
case WM_INITDIALOG :
     ntrans = 0;
     sprintf(reset,"%.2f",tra[0]);
     SetDlgItemText(hDlg,IDD_TRANA, reset);
     sprintf(reset,"%.2f",trb[0]);
     SetDlgItemText(hDlg,IDD_TRANB, reset);
     sprintf(reset,"%.2f",trc[0]);
     SetDlgItemText(hDlg,IDD_TRANC, reset);
     sprintf(reset,"%.2f",trd[0]);
     SetDlgItemText(hDlg,IDD_TRAND, reset);
     sprintf(reset,"%.2f",tre[0]);
     SetDlgItemText(hDlg,IDD_TRANE, reset);
     sprintf(reset,"%.2f",trf[0]);
     SetDlgItemText(hDlg,IDD_TRANF, reset);
     sprintf(reset,"%.2f",trp[0]);
     SetDlgItemText(hDlg,IDD_PROBA, reset);
     SetDlgItemText(hDlg,IDD_FRNAM, NomIFS);
     SetDlgItemText(hDlg,IDD_DENSI, dens);
     for (int i=0;i<nt;i++)
              {
              a[i]=tra[i];
              b[i]=trb[i];
              c[i]=trc[i];
              d[i]=trd[i];
              e[i]=tre[i];
              f[i]=trf[i];
              p[i]=trp[i];
              };
     col[0]=color[0];
     col[1]=color[1];
     col[2]=color[2];
     return FALSE;

case WM_COMMAND :
   switch(wParam)
     {
     case IDD_FRNAM :
          GetDlgItemText(hDlg,IDD_FRNAM,nom,15);
          return FALSE;
     case IDD_DENSI :
          GetDlgItemText(hDlg,IDD_DENSI,dens,3);
          return FALSE;
     case IDD_TRANA :
     case IDD_TRANB :
     case IDD_TRANC :
     case IDD_TRAND :
     case IDD_TRANE :
     case IDD_TRANF :
     case IDD_PROBA :
          return FALSE;
     case IDD_NEXT  :
          if (ntrans<7)
             {
             GetDlgItemText(hDlg,IDD_TRANA, geta,8);
             GetDlgItemText(hDlg,IDD_TRANB, getb,8);
             GetDlgItemText(hDlg,IDD_TRANC, getc,8);
             GetDlgItemText(hDlg,IDD_TRAND, getd,8);
             GetDlgItemText(hDlg,IDD_TRANE, gete,8);
             GetDlgItemText(hDlg,IDD_TRANF, getf,8);
             GetDlgItemText(hDlg,IDD_PROBA, getp,8);

             a[ntrans]=atof(geta);
             b[ntrans]=atof(getb);
             c[ntrans]=atof(getc);
             d[ntrans]=atof(getd);
             e[ntrans]=atof(gete);
```

```
            f[ntrans]=atof(getf);
            p[ntrans]=atof(getp);
            if (p[ntrans]>1) p[ntrans]=1;
            if (p[ntrans]<0) p[ntrans]=0;

            if (ntrans<(nt-1))
               {
               ntrans += 1;
               sprintf(reset,"%.2f",tra[ntrans]);
               SetDlgItemText(hDlg,IDD_TRANA, reset);
               sprintf(reset,"%.2f",trb[ntrans]);
               SetDlgItemText(hDlg,IDD_TRANB, reset);
               sprintf(reset,"%.2f",trc[ntrans]);
               SetDlgItemText(hDlg,IDD_TRANC, reset);
               sprintf(reset,"%.2f",trd[ntrans]);
               SetDlgItemText(hDlg,IDD_TRAND, reset);
               sprintf(reset,"%.2f",tre[ntrans]);
               SetDlgItemText(hDlg,IDD_TRANE, reset);
               sprintf(reset,"%.2f",trf[ntrans]);
               SetDlgItemText(hDlg,IDD_TRANF, reset);
               sprintf(reset,"%.2f",trp[ntrans]);
               SetDlgItemText(hDlg,IDD_PROBA, reset);
               }
             else
               {
               sprintf(reset,"-");
               SetDlgItemText(hDlg,IDD_TRANA, reset);
               SetDlgItemText(hDlg,IDD_TRANB, reset);
               SetDlgItemText(hDlg,IDD_TRANC, reset);
               SetDlgItemText(hDlg,IDD_TRAND, reset);
               SetDlgItemText(hDlg,IDD_TRANE, reset);
               SetDlgItemText(hDlg,IDD_TRANF, reset);
               SetDlgItemText(hDlg,IDD_PROBA, reset);
               };
            };
         return FALSE;

      case IDD_WHITE :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         col[0]=0;
         col[1]=0;
         col[2]=0;
         return FALSE;
      case IDD_BLUE  :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         col[0]=0;
         col[1]=0;
         col[2]=255;
         return FALSE;
      case IDD_GREEN :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         col[0]=0;
         col[1]=255;
         col[2]=0;
         return FALSE;
      case IDD_CYAN  :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         col[0]=0;
         col[1]=255;
         col[2]=255;
         return FALSE;
      case IDD_RED   :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         col[0]=255;
         col[1]=0;
         col[2]=0;
         return FALSE;
      case IDD_MAGEN :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         col[0]=255;
         col[1]=0;
         col[2]=255;
         return FALSE;
      case IDD_YELLO :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         col[0]=255;
         col[1]=255;
         col[2]=0;
         return FALSE;
      case IDD_GRAY  :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         col[0]=127;
         col[1]=127;
         col[2]=127;
         return FALSE;
      case IDD_SHGRA :
      case IDD_SHGRE :
      case IDD_SHRED :
         CheckRadioButton(hDlg,IDD_WHITE,IDD_GRAY, wParam);
         return FALSE;
      case IDOK :
         densite=atoi(dens);
         sprintf(NomIFS,nom);
         sompro=0;
         for (int i=0;i<nt;i++)
             {
             tra[i]=a[i];
             trb[i]=b[i];
             trc[i]=c[i];
             trd[i]=d[i];
             tre[i]=e[i];
             trf[i]=f[i];
             trp[i]=p[i];
             sompro += trp[i];
             };
         if (sompro>1)
            {
            for (int j=0;j<nt;j++)
                trp[i]=1/(nt*1.0);
            }
         else
            trp[nt-1] = 1 - (sompro-trp[nt-1]);
         color[0]=col[0];
         color[1]=col[1];
         color[2]=col[2];
         delete ObjetIFS1;
         ObjetIFS1 = new ifs(nt,tra,trb,trc,trd,tre,trf,trp,
                             densite,2,color[0],color[1],color[2]);
         EndDialog(hDlg,TRUE);
         return TRUE;
      case IDCANCEL :
         EndDialog(hDlg,FALSE);
         return FALSE;
      }
      break;
}
return FALSE;
```

```
// ------------------------------------------------------------
   BOOL FAR PASCAL _export NewCodeMet(HWND hDlg, WORD Message,
                                      WORD wParam, LONG lParam)
// ------------------------------------------------------------
// Dialogue : introduire le code d'un nouveau Meta-IFS
{
    static int ntrans,densite;
    static float a[8],b[8],c[8],d[8],e[8],f[8],p[8],sompro;
    static char geta[5],getb[5],getc[5],getd[5],gete[5],getf[5],getp[5],
                dens[5]="1",nom[15]="MIfs1\0",reset[4]="0.0";

    switch(Message)
    {
        case WM_INITDIALOG :
             ntrans = 0;
             SetDlgItemText(hDlg,IDD_TRANA, reset);
             SetDlgItemText(hDlg,IDD_TRANB, reset);
             SetDlgItemText(hDlg,IDD_TRANC, reset);
             SetDlgItemText(hDlg,IDD_TRAND, reset);
             SetDlgItemText(hDlg,IDD_TRANE, reset);
             SetDlgItemText(hDlg,IDD_TRANF, reset);
             SetDlgItemText(hDlg,IDD_PROBA, reset);
             SetDlgItemText(hDlg,IDD_FRNAM, nom);
             SetDlgItemText(hDlg,IDD_DENSI, dens);
             return FALSE;

        case WM_COMMAND :
           switch(wParam)
             {
             case IDD_FRNAM :
                  GetDlgItemText(hDlg,IDD_FRNAM,nom,15);
                  return FALSE;
             case IDD_DENSI :
                  GetDlgItemText(hDlg,IDD_DENSI,dens,3);
                  return FALSE;
             case IDD_TRANA :
             case IDD_TRANB :
             case IDD_TRANC :
             case IDD_TRAND :
             case IDD_TRANE :
             case IDD_TRANF :
             case IDD_PROBA :
                  return FALSE;
             case IDD_NEXT  :
                  if (ntrans<7)
                     {
                     GetDlgItemText(hDlg,IDD_TRANA, geta,8);
                     GetDlgItemText(hDlg,IDD_TRANB, getb,8);
                     GetDlgItemText(hDlg,IDD_TRANC, getc,8);
                     GetDlgItemText(hDlg,IDD_TRAND, getd,8);
                     GetDlgItemText(hDlg,IDD_TRANE, gete,8);
                     GetDlgItemText(hDlg,IDD_TRANF, getf,8);
                     GetDlgItemText(hDlg,IDD_PROBA, getp,8);

                     a[ntrans]=atof(geta);
                     b[ntrans]=atof(getb);
                     c[ntrans]=atof(getc);
                     d[ntrans]=atof(getd);
                     e[ntrans]=atof(gete);
                     f[ntrans]=atof(getf);
                     p[ntrans]=atof(getp);
                     if (p[ntrans]>1) p[ntrans]=1;
                     if (p[ntrans]<0) p[ntrans]=0;
                     ntrans += 1;
                     SetDlgItemText(hDlg,IDD_TRANA, reset);
                     SetDlgItemText(hDlg,IDD_TRANB, reset);
                     SetDlgItemText(hDlg,IDD_TRANC, reset);
                     SetDlgItemText(hDlg,IDD_TRAND, reset);
                     SetDlgItemText(hDlg,IDD_TRANE, reset);
                     SetDlgItemText(hDlg,IDD_TRANF, reset);
                     SetDlgItemText(hDlg,IDD_PROBA, reset);
                     };
                  return FALSE;

             case IDOK :
                  densite=atoi(dens);
                  sprintf(NomMIFS,nom);
                  sompro=0;
                  for (int i=0;i<ntrans;i++)
                      {
                      tma[i]=a[i];
                      tmb[i]=b[i];
                      tmc[i]=c[i];
                      tmd[i]=d[i];
                      tme[i]=e[i];
                      tmf[i]=f[i];
                      tmp[i]=p[i];
                      sompro += tmp[i];
                      };
                  if (sompro>1)
                     {
                     for (int j=0;j<ntrans;j++)
                         tmp[i]=1/(ntrans*1.0);
                     }
                  else
                     tmp[ntrans-1] = 1 - (sompro-tmp[ntrans-1]);
                  mt=ntrans;
                  if (mt==0) {mt=1;tmp[0]=1;};
                  numifs=0;
                  delete ObjetMIFS1;
                  ObjetMIFS1 = new Meta_ifs(mt,tma,tmb,tmc,tmd,tme,tmf,tmp,densi
te);
                  EndDialog(hDlg,TRUE);
                  return TRUE;
             case IDCANCEL :
                  EndDialog(hDlg,FALSE);
                  return FALSE;
             }
             break;
    }
    return FALSE;
};

// ------------------------------------------------------------
   BOOL FAR PASCAL _export ModCodeMet(HWND hDlg, WORD Message,
                                      WORD wParam, LONG lParam)
// ------------------------------------------------------------
// Dialogue : introduire modification du code d'un Meta-IFS
{
    static int ntrans,densite;
    static float a[8],b[8],c[8],d[8],e[8],f[8],p[8],sompro;
    static char geta[5],getb[5],getc[5],getd[5],gete[5],getf[5],getp[5],
                dens[5]="1",nom[15]="MIfs1\0",reset[4]="0.0";

    switch(Message)
    {
        case WM_INITDIALOG :
             ntrans = 0;
```

```
        if (ntrans<(mt-1))
           {
           ntrans += 1;
           sprintf(reset,"%.2f",tma[ntrans]);
           SetDlgItemText(hDlg,IDD_TRANA, reset);
           sprintf(reset,"%.2f",tmb[ntrans]);
           SetDlgItemText(hDlg,IDD_TRANB, reset);
           sprintf(reset,"%.2f",tmc[ntrans]);
           SetDlgItemText(hDlg,IDD_TRANC, reset);
           sprintf(reset,"%.2f",tmd[ntrans]);
           SetDlgItemText(hDlg,IDD_TRAND, reset);
           sprintf(reset,"%.2f",tme[ntrans]);
           SetDlgItemText(hDlg,IDD_TRANE, reset);
           sprintf(reset,"%.2f",tmf[ntrans]);
        --  SetDlgItemText(hDlg,IDD_TRANF, reset);
           sprintf(reset,"%.2f",tmp[ntrans]);
           SetDlgItemText(hDlg,IDD_PROBA, reset);
           }
         else
           {
           sprintf(reset,"-");
           SetDlgItemText(hDlg,IDD_TRANA, reset);
           SetDlgItemText(hDlg,IDD_TRANB, reset);
           SetDlgItemText(hDlg,IDD_TRANC, reset);
           SetDlgItemText(hDlg,IDD_TRAND, reset);
           SetDlgItemText(hDlg,IDD_TRANE, reset);
           SetDlgItemText(hDlg,IDD_TRANF, reset);
           SetDlgItemText(hDlg,IDD_PROBA, reset);
           };
        };
     return FALSE;

case IDOK :
     densite=atoi(dens);
     sprintf(NomMIFS,nom);
     sompro=0;
     for (int i=0;i<ntrans;i++)
         {
         tma[i]=a[i];
         tmb[i]=b[i];
         tmc[i]=c[i];
         tmd[i]=d[i];
         tme[i]=e[i];
         tmf[i]=f[i];
         tmp[i]=p[i];
         sompro += tmp[i];
         };
     if (sompro>1)
        {
        for (int j=0;j<ntrans;j++)
            tmp[i]=1/(ntrans*1.0);
        }
     else
        tmp[ntrans-1] = 1 - (sompro-tmp[ntrans-1]);
     mt=ntrans;
     numifs=0;
     delete ObjetMIFS1;
     ObjetMIFS1 = new Meta_ifs(mt,tma,tmb,tmc,tmd,tme,tmf,tmp,densi

     EndDialog(hDlg,TRUE);
     return TRUE;
case IDCANCEL :
     EndDialog(hDlg,FALSE);
     return FALSE;
}
break;
}
```

```
     sprintf(reset,"%.2f",tma[0]);
     SetDlgItemText(hDlg,IDD_TRANA, reset);
     sprintf(reset,"%.2f",tmb[0]);
     SetDlgItemText(hDlg,IDD_TRANB, reset);
     sprintf(reset,"%.2f",tmc[0]);
     SetDlgItemText(hDlg,IDD_TRANC, reset);
     sprintf(reset,"%.2f",tmd[0]);
     SetDlgItemText(hDlg,IDD_TRAND, reset);
     sprintf(reset,"%.2f",tme[0]);
     SetDlgItemText(hDlg,IDD_TRANE, reset);
     sprintf(reset,"%.2f",tmp[0]);
     SetDlgItemText(hDlg,IDD_PROBA, reset);
     sprintf(reset,"%.2f",tmf[0]);
     SetDlgItemText(hDlg,IDD_TRANF, reset);
     SetDlgItemText(hDlg,IDD_DENSI, dens);
     SetDlgItemText(hDlg,IDD_FRNAM, NomMIFS);
     for (int i=0;i<mt;i++)
             {
             a[i]=tma[i];
             b[i]=tmb[i];
             c[i]=tmc[i];
             d[i]=tmd[i];
             e[i]=tme[i];
             f[i]=tmf[i];
             p[i]=tmp[i];
             };
     return FALSE;

case WM_COMMAND :
   switch(wParam)
     {
     case IDD_FRNAM :
          GetDlgItemText(hDlg,IDD_FRNAM,nom,15);
          return FALSE;
     case IDD_DENSI :
          GetDlgItemText(hDlg,IDD_DENSI,dens,3);
          return FALSE;
     case IDD_TRANA :
     case IDD_TRANB :
     case IDD_TRANC :
     case IDD_TRAND :
     case IDD_TRANE :
     case IDD_TRANF :
     case IDD_PROBA :
          return FALSE;
     case IDD_NEXT  :
          if (ntrans<7)
             {
             GetDlgItemText(hDlg,IDD_TRANA, geta,8);
             GetDlgItemText(hDlg,IDD_TRANB, getb,8);
             GetDlgItemText(hDlg,IDD_TRANC, getc,8);
             GetDlgItemText(hDlg,IDD_TRAND, getd,8);
             GetDlgItemText(hDlg,IDD_TRANE, gete,8);
             GetDlgItemText(hDlg,IDD_TRANF, getf,8);
             GetDlgItemText(hDlg,IDD_PROBA, getp,8);

             a[ntrans]=atof(geta);
             b[ntrans]=atof(getb);
             c[ntrans]=atof(getc);
             d[ntrans]=atof(getd);
             e[ntrans]=atof(gete);
             f[ntrans]=atof(getf);
             p[ntrans]=atof(getp);
             if (p[ntrans]>1) p[ntrans]=1;
             if (p[ntrans]<0) p[ntrans]=0;
```

```
 ---------------------------------------------------------------------
 void DrawBitmap(HDC hdc,HBITMAP hBitmap,short xstart,short ystart)
 ---------------------------------------------------------------------
 Fonction : Dessiner le Bitmap

BITMAP bm;
HDC hdcmem;
DWORD dwsize;
POINT ptsize,ptorg;

hdcmem = CreateCompatibleDC(hdc);
SelectObject(hdcmem,hBitmap);
SetMapMode(hdcmem,GetMapMode(hdc));
GetObject(hBitmap,sizeof(BITMAP),(LPSTR)&bm);

ptsize.x = bm.bmWidth;
ptsize.y = bm.bmHeight;
if (ptsize.x>398) ptsize.x = 398;
if (ptsize.y>378) ptsize.y = 378;
DPtoLP(hdc,&ptsize,1);

ptorg.x = 0;
ptorg.y = 0;
DPtoLP(hdcmem,&ptorg,1);

BitBlt(hdc,xstart,ystart,ptsize.x,ptsize.y,hdcmem,ptorg.x,ptorg.y,SRCCOPY);

DeleteDC(hdcmem);


 ---------------------------------------------------
  float Dimension(HDC hdc,int sx,int sy,int fx,int fy)
 ---------------------------------------------------
 Fonction : Dimension d'une partie de l'écran

int imin=1000,
    imax=0,
    jmin=1000,
    jmax=0,
    i,j;
float cote=0,count=0,dim=0;
unsigned long int rgbcolor;
HPEN hPen;
HBRUSH hBrush;
LOGPEN lPen = {PS_NULL,0,RGB(0,0,0)};

hPen = CreatePenIndirect(&lPen);
hPen = SelectObject(hdc,hPen);

for (i=sx; i < fx; i++)
    {
    /*** contrôle écran ***/
    if (i<(216+66))
       {
       hBrush = CreateSolidBrush(RGB(i-26,0,0));
       }
    else
       {
       if (i<(216+66+255))
          {
          hBrush = CreateSolidBrush(RGB(255,i-282,0));
          }
       else
          {
          hBrush = CreateSolidBrush(RGB(255,255,(i*2)-537));
          };
       };
       hBrush = SelectObject(hdc,hBrush);
       Rectangle(hdc,i-1,18,i+2,36);
       DeleteObject(SelectObject(hdc,hBrush));

       /*** BOX counting ***/
       for (j=sy; j < fy; j++)
           {
           rgbcolor = GetPixel(hdc,i,j);
           if (rgbcolor != RGB(255,255,255))
              {
              SetPixel(hdc,i,j,RGB(0,0,255));
              count = count+1;

              if (i<imin) imin = i;
              if (i>imax) imax = i;
              if (j<jmin) jmin = j;
              if (j>jmax) jmax = j;

              };
           };
       };

  DeleteObject(SelectObject(hdc,hPen));

  if ((imin !=1000)&&(jmin !=1000))
     {
     if ((imax-imin)>(jmax-jmin))
        {
        cote = imax-imin+1;
        }
     else
        {
        cote = jmax-jmin+1;
        };

     dim = log(count)/log(cote);
     return dim;
     }
  else
     {
     return -1;
     }
};

// ------------------------------------------
   void PaintWinFig (HWND hCtrlBlk,short *color)
// ------------------------------------------
// impression d'un rectangle coloré pour le
// choix des couleurs.
{
 HDC hdc;
 HBRUSH hBrush;
 RECT rect;

 InvalidateRect(hCtrlBlk,NULL,TRUE);
 UpdateWindow(hCtrlBlk);

 hdc = GetDC(hCtrlBlk);
 GetClientRect(hCtrlBlk,&rect);
 hBrush = CreateSolidBrush(RGB(color[0],color[1],color[2]));
 hBrush = SelectObject(hdc,hBrush);
 Rectangle(hdc,rect.left,rect.top,rect.right,rect.bottom);
 DeleteObject(SelectObject(hdc,hBrush));
 ReleaseDC(hCtrlBlk,hdc);
};
```

```
        case WM_COMMAND :
           switch(wParam)
             {
             case 50 :
                  CheckRadioButton(hDlg,50,57, wParam);
                  col[0]=0;
                  col[1]=0;
                  col[2]=0;
                  return FALSE;
             case 51 :
                  CheckRadioButton(hDlg,50,57, wParam);
                  col[0]=0;
                  col[1]=0;
                  col[2]=255;
                  return FALSE;
             case 52 :
                  CheckRadioButton(hDlg,50,57, wParam);
                  col[0]=0;
                  col[1]=255;
                  col[2]=0;
                  return FALSE;
             case 53 :
                  CheckRadioButton(hDlg,50,57, wParam);
                  col[0]=0;
                  col[1]=255;
                  col[2]=255;
                  return FALSE;
             case 54 :
                  CheckRadioButton(hDlg,50,57, wParam);
                  col[0]=255;
                  col[1]=0;
                  col[2]=0;
                  return FALSE;
             case 55 :
                  CheckRadioButton(hDlg,50,57, wParam);
                  col[0]=255;
                  col[1]=0;
                  col[2]=255;
                  return FALSE;
             case 56 :
                  CheckRadioButton(hDlg,50,57, wParam);
                  col[0]=255;
                  col[1]=255;
                  col[2]=0;
                  return FALSE;
             case 57 :
                  CheckRadioButton(hDlg,50,57, wParam);
                  col[0]=127;
                  col[1]=127;
                  col[2]=127;
                  return FALSE;
             case IDOK :
                  pol->Changer_Coul(col[0],col[1],col[2],
                                    color[0],color[1],color[2]);
                  EndDialog(hDlg,TRUE);
                  return TRUE;
             case IDCANCEL:
                  EndDialog(hDlg,FALSE);
                  return FALSE;
             }
             break;
    }
    return FALSE;
};
```

```
// ------------------------------------------------------
   BOOL FAR PASCAL _export ColorFig(HWND hDlg, WORD message,
                                    WORD wParam, LONG lParam)
// ------------------------------------------------------
// Dialogue : choix des couleurs
{
    HWND         hwndParent, hCtrl;
    static HWND  hCtrlBlk;
    static col[3]={0,0,0};
    short        nCtrlID, nIndex,nID;

    switch (message)
    {
        case WM_INITDIALOG :
             for (nCtrlID = 10; nCtrlID < 13; nCtrlID++)
                 {
                 hCtrl = GetDlgItem (hDlg,nCtrlID);
                 SetScrollRange(hCtrl,SB_CTL,0,255,FALSE);
                 SetScrollPos  (hCtrl,SB_CTL,color[nCtrlID-10],FALSE);
                 }
             hCtrlBlk = GetDlgItem(hDlg,IDD_PAINT);
             PaintWinFig(hCtrlBlk,color);
             return FALSE;

        case WM_VSCROLL :
           hCtrl = HIWORD(lParam);
           nCtrlID = GetWindowWord(hCtrl,GWW_ID);
           nIndex = nCtrlID - 10;
           hwndParent = GetParent(hDlg);

           switch(wParam)
             {
             case SB_PAGEDOWN :
                  color[nIndex] += 15;
             case SB_LINEDOWN :
                  if (color[nIndex]<255)
                     color[nIndex] = color[nIndex]+1;
                  else
                     color[nIndex] = 255;
                  break;
             case SB_PAGEUP :
                  color[nIndex] -= 15;
             case SB_LINEUP :
                  if (color[nIndex]>0)
                     color[nIndex] = color[nIndex]-1;
                  else
                     color[nIndex] = 0;
                  break;
             case SB_TOP :
                  color[nIndex] = 0;
                  break;
             case SB_BOTTOM :
                  color[nIndex] = 255;
                  break;
             case SB_THUMBPOSITION :
             case SB_THUMBTRACK :
                  color[nIndex] = LOWORD(lParam);
                  break;
             default :
                  return FALSE;
             }
           SetScrollPos(hCtrl,SB_CTL,color[nIndex],TRUE);
           SetDlgItemInt(hDlg,nCtrlID+3,color[nIndex],FALSE);
           PaintWinFig(hCtrlBlk,color);
           return TRUE;
```

```
/*************************************************************************/

 long FAR PASCAL _export WndProc (HWND hwnd, WORD Message,
                                  WORD wParam, LONG lParam)

/*************************************************************************/
/* programme principal : le traitment des messages                       */
{
    static FARPROC lpfnewcode,lpfnewmet ,lpfncolors,lpfnabout ,lpfnloadi1,
                   lpfnsavei1,lpfndrawfr,lpfndrawme,lpfnselect,lpfnselmet,
                   lpfmodcode,lpfnappifs,lpfmodmet ,lpfncolfig ;
    static HANDLE hInstance;
    HDC         hdc;
    HMENU       hMenu;
    PAINTSTRUCT ps;
    RECT        rect;
    static POINT corner1,corner2,corner3,corner4;
    static LOGPEN lPen = {PS_NULL,0,RGB(0,0,0)};
    static HPEN   hPen;
    static HBRUSH hBrush;
    static HICON hIcon1,hIcon2,hIcon3;
    static int paintmode;

    // variables auxiliaires :
    int res,quit,i,j,k;
    POINT tab[8];
    POINT auxp;
    static float dim;
    static char szString[60]=" ";
    HBITMAP hbmp;

    switch(Message)
    {
        case WM_PAINT:
             hdc = BeginPaint (hwnd, &ps);
             SetClassWord(hwnd,GCW_HBRBACKGROUND,
             CreateSolidBrush(RGB(79,160,96)));
             Rectangle(hdc,25,40,125,140);
             Rectangle(hdc,130,40,210,140);
                TextOut(hdc,135,45,"Dimension",9);
                TextOut(hdc,135,65,"100x100:",8);
                TextOut(hdc,135,101,"400x400:",8);
             Rectangle(hdc,215,40,615,420);
             Rectangle(hdc,25,17,615,36);
             Rectangle(hdc,25,145,210,420);
             DrawIcon(hdc,35,154,hIcon3);
             TextOut(hdc,80,154,"Figure:",7);
             DrawIcon(hdc,35,190,hIcon1);
             TextOut(hdc,80,190,"Code IFS:",9);
             TextOut(hdc,35,226,"Coordonnees:",12);
             TextOut(hdc,55,242,"origine :",9);
             TextOut(hdc,55,258,"axe x :",7);
             TextOut(hdc,55,274,"axe y :",7);
             DrawIcon(hdc,35,305,hIcon2);
             TextOut(hdc,80,305,"Code Meta-IFS:",14);
             TextOut(hdc,35,341,"Composition:",12);

             hPen = CreatePenIndirect(&lPen);
             hPen = SelectObject(hdc,hPen);

             for (i=0;i<256;i++)
                 {
                 hBrush = CreateSolidBrush(RGB(i,0,0));
                 hBrush = SelectObject(hdc,hBrush);
                 Rectangle(hdc,25+i,18,25+i+3,36);
                 DeleteObject(SelectObject(hdc,hBrush));
                 };
             for (i=0;i<256;i++)
                 {
                 hBrush = CreateSolidBrush(RGB(255,i,0));
                 hBrush = SelectObject(hdc,hBrush);
                 Rectangle(hdc,255+25+i,18,255+25+i+3,36);
                 DeleteObject(SelectObject(hdc,hBrush));
                 };
             for (i=0;i<77;i=i+1)
                 {
                 hBrush = CreateSolidBrush(RGB(255,255,i*3));
                 hBrush = SelectObject(hdc,hBrush);
                 Rectangle(hdc,511+25+i,18,511+25+i+3,36);
                 DeleteObject(SelectObject(hdc,hBrush));
                 };

             DeleteObject(SelectObject(hdc,hPen));

             /*** Données affichées ***/
             AfficherDonnees(hdc);

             EndPaint (hwnd, &ps);
             return 0;

    case WM_CREATE:
        SetClassWord(hwnd,GCW_HBRBACKGROUND,
                     CreateSolidBrush(RGB(79,160,96)));
        hInstance = ((LPCREATESTRUCT) lParam)->hInstance;
        lpfnewcode = MakeProcInstance ((FARPROC)NewCodeRan,hInstance);
        lpfmodcode = MakeProcInstance ((FARPROC)ModCodeRan,hInstance);
        lpfnewmet  = MakeProcInstance ((FARPROC)NewCodeMet,hInstance);
        lpfmodmet  = MakeProcInstance ((FARPROC)ModCodeMet,hInstance);
        lpfncolors = MakeProcInstance ((FARPROC)Colorbox,hInstance);
        lpfnabout  = MakeProcInstance ((FARPROC)Aboutbox,hInstance);
        lpfnloadi1 = MakeProcInstance ((FARPROC)LoadIFS1,hInstance);
        lpfnsavei1 = MakeProcInstance ((FARPROC)SaveIFS1,hInstance);
        lpfndrawfr = MakeProcInstance ((FARPROC)DrawFract,hInstance);
        lpfndrawme = MakeProcInstance ((FARPROC)DrawMeta,hInstance);
        lpfnselect = MakeProcInstance ((FARPROC)SelectIFS,hInstance);
        lpfnselmet = MakeProcInstance ((FARPROC)SelMetIFS,hInstance);
        lpfnappifs = MakeProcInstance ((FARPROC)SelAppIFS,hInstance);
        lpfncolfig = MakeProcInstance ((FARPROC)ColorFig ,hInstance);
        paintmode = 0;
        hIcon1 = LoadIcon(hInst,"IDI_FRACTALS");
        hIcon2 = LoadIcon(hInst,"IDI_METFRACT");
        hIcon3 = LoadIcon(hInst,"IDI_FIGURE");
        ObjetIFS1  = 0;
        ObjetMIFS1 = 0;
        pol        = 0;
        return 0;

    case WM_LBUTTONDOWN :

         hMenu = GetMenu(hwnd);
         switch (paintmode)
            {
            case 1:  // zoom sélectionné
                corner1 = MAKEPOINT (lParam);
                if((corner1.x<25) || (corner1.x>125) ||
                   (corner1.y<40) || (corner1.y>140))
                   {
                   MessageBox(hwnd,"Fenetre-Zoom hors du domaine"
                                   ,"Zoom sur IFS"
                                   ,MB_ICONSTOP);
                   ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                   EnableMenus(hMenu);
                   paintmode=0;
                   }
```

```
                else
                   {
                   hdc = GetDC(hwnd);
                   Rectangle(hdc,corner1.x,corner1.y,corner1.x+2,corner1.y+2
;
                   ReleaseDC(hwnd,hdc);
                   };
                break;
             }
          return 0;

       case WM_LBUTTONUP :
          hMenu = GetMenu(hwnd);
          switch (paintmode)
             {
             case 1:
                /*** une sélection zoom à été activée ***/
                /*** 2ème coordonnée du zoom ***/
                corner2 = MAKEPOINT (lParam);
                if((corner2.x>25) && (corner2.x<125) &&
                   (corner2.y>40) && (corner2.y<140))
                   {
                   /*** dessiner le domaine choisi ***/
                   hdc = GetDC (hwnd);
                   MoveTo(hdc,corner1.x,corner1.y);
                   LineTo(hdc,corner1.x,corner2.y);
                   LineTo(hdc,corner2.x,corner2.y);
                   LineTo(hdc,corner2.x,corner1.y);
                   LineTo(hdc,corner1.x,corner1.y);
                   ReleaseDC (hwnd,hdc);

                   /*** coordonnées par rapp. à 100x100 **/
                   corner1.x -= 25;
                   corner1.y -= 40;
                   corner2.x -= 25;
                   corner2.y -= 40;

                   /*** réécrire les coordonnées 'proprement' ***/
                   if (corner1.x > corner2.x) { auxp.x = corner1.x;
                                                corner1.x = corner2.x;
                                                corner2.x = auxp.x;     }
                   if (corner1.y > corner2.y) { auxp.y = corner1.y;
                                                corner1.y = corner2.y;
                                                corner2.y = auxp.y;     }
                   if ( ((corner2.x-corner1.x)>14) &&
                        ((corner2.y-corner1.y)>14)   )
                      {
                      /*** dessiner le zoom sur l'objet sélectionné ***/
                      SetCursor (LoadCursor (NULL,IDC_WAIT));
                      ObjetIFS1->Modifier_Domaine(215,40,615,40,215,420);
                      res = ObjetIFS1->Zoom(hwnd,corner1.x,corner1.y,corner2
.x,corner2.y);
                      ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                      if (res==-2)
                         {
                         MessageBox(hwnd,"Le nombre maximum d'iterations qui
 est fixe a 64.000 est atteint.  Pour avoir une image plus dense, choisir une fe
netre-Zoom de taille superieure"
                                    ,"Dessiner zoom sur IFS"
                                    ,MB_ICONINFORMATION);
                         };
                      }
                   else
                      {
                      MessageBox(hwnd,"La Taille de la fenetre-Zoom est trop
 petite.           (min 15 pixels)"
                                 ,"Zoom sur IFS"
                                 ,MB_ICONSTOP);
                      CbjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                      };
                   SetCursor (LoadCursor (NULL,IDC_ARROW));
                   }
                else
                   {
                   MessageBox(hwnd,"Fenetre-Zoom hors du domaine"
                                 ,"Zoom sur IFS"
                                 ,MB_ICONSTOP);
                   ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                };
                EnableMenus(hMenu);
                paintmode = 0;
                break;

             case 2:
             case 5:
                /*** une sélection pour l'origine d'un repère ***/
                /*** est activée (2 pour IFS, 5 pour MIFS)    ***/
                corner1 = MAKEPOINT (lParam);
                if((corner1.x<215) || (corner1.x>615) ||
                   (corner1.y<40) || (corner1.y>420))
                   {
                   MessageBox(hwnd,"Point non valide, debordement du domaine

                                 ,"Dessiner - Origine"
                                 ,MB_ICONSTOP);
                   if (paintmode==2)
                      ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                   EnableMenus(hMenu);
                   paintmode=0;
                   }
                else
                   {
                   hdc = GetDC(hwnd);
                   if (EncDom==1)
                      Rectangle(hdc,corner1.x  ,corner1.y,
                                    corner1.x+2,corner1.y+2);
                   if (paintmode==2)
                      {
                      sprintf(ori,"( %i , %i )      ",corner1.x,corner1.y);
                      AfficherDonnees(hdc);
                      };
                   paintmode += 1; // aller vers l'axe i
                   ReleaseDC(hwnd,hdc);
                   };
                break;

             case 3:
             case 6:
                /*** une sélection pour l'axe i d'un repère ***/
                /*** est activée                            ***/
                corner2 = MAKEPOINT (lParam);
                if((corner2.x<215) || (corner2.x>615) ||
                   (corner2.y<40) || (corner2.y>420))
                   {
                   MessageBox(hwnd,"Point non valide, debordement du domaine

                                 ,"Dessiner - Axe X"
                                 ,MB_ICONSTOP);
                   if (paintmode==2)
                      ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                   EnableMenus(hMenu);
                   paintmode=0;
                   }
```

```
            else
               {
               hdc = GetDC(hwnd);
               if (EncDom == 1)
                  {
                  Rectangle(hdc,corner2.x  ,corner2.y,
                                corner2.x+2,corner2.y+2);
                  MoveTo(hdc,corner1.x,corner1.y);
                  LineTo(hdc,corner2.x,corner2.y);
                  };
               if (paintmode==3)
                  {
                  sprintf(axi,"( %i , %i )     ",corner2.x,corner2.y);
                  AfficherDonnees(hdc);
                  };
               ReleaseDC(hwnd,hdc);
               paintmode += 1; // aller vers l'axe j
               };
            break;

         case 4:
         case 7:
            /*** une sélection pour l'axe j d'un repère ***/
            /*** est activée                            ***/
            corner3 = MAKEPOINT (lParam);
            if((corner3.x<215) || (corner3.x>615) ||
               (corner3.y<40) || (corner3.y>420))
               {
               MessageBox(hwnd,"Point non valide, debordement du domaine

                               ,"Dessiner - Axe Y"
                               ,MB_ICONSTOP);
               }
            else
               {
               hdc = GetDC(hwnd);
               if (EncDom == 1)
                  {
                  Rectangle(hdc,corner3.x  ,corner3.y
                                ,corner3.x+2,corner3.y+2);
                  MoveTo(hdc,corner1.x,corner1.y);
                  LineTo(hdc,corner3.x,corner3.y);
                  };
               corner4.x = corner3.x + corner2.x - corner1.x;
               corner4.y = corner3.y + corner2.y - corner1.y;

               if((corner4.x<215) || (corner4.x>615) ||
                  (corner4.y<40) || (corner4.y>420))
                  {
                  MessageBox(hwnd,"Point non valide, IFS deborde le doma
ine"

                                  ,"Dessiner sur domaine"
                                  ,MB_ICONSTOP);
                  }
               else
                  {
                  if (EncDom == 1)
                     {
                     MoveTo(hdc,corner3.x,corner3.y);
                     LineTo(hdc,corner4.x,corner4.y);
                     MoveTo(hdc,corner2.x,corner2.y);
                     LineTo(hdc,corner4.x,corner4.y);
                     };
                  SetCursor (LoadCursor (NULL,IDC_WAIT));
                  if (paintmode==4)
                     {
                     sprintf(axj,"( %i , %i )     ",corner3.x,corner3.y)
```

```
                     AfficherDonnees(hdc);
                     ObjetIFS1->Modifier_Domaine(corner1.x,corner1.y,
                                                 corner2.x,corner2.y,
                                                 corner3.x,corner3.y);
                     ObjetIFS1->Dessiner(hwnd);
                     ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                     }
                  else
                     {
                     ObjetMIFS1->Transformer(corner1.x,corner1.y,
                                             corner2.x,corner2.y,
                                             corner3.x,corner3.y);
                     if (MIFSTyDr==1)
                        ObjetMIFS1->Dessiner(hwnd,nprod);
                     else
                        ObjetMIFS1->Dessiner_Chaos(hwnd,nprod);
                     };
                  };
               SetCursor (LoadCursor (NULL,IDC_ARROW));
               ReleaseDC(hwnd,hdc);
               };
            if (paintmode==2)
               ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
            EnableMenus(hMenu);
            paintmode=0;
            break;
         }
      return 0;

   case WM_COMMAND:

      hMenu = GetMenu(hwnd);
      switch (wParam)
         {
/*** nouveau IFS ***/
         case IDM_SET1:
            if (DialogBox (hInstance, "NewCodeRan", hwnd, lpfnewcode))
               {
               hdc = GetDC(hwnd);
               sprintf(ori,"");
               sprintf(axi,"");
               sprintf(axj,"");
               AfficherDonnees(hdc);
               EnableMenus(hMenu);
               ReleaseDC(hwnd,hdc);
               };
            return 0;
/*** modifier IFS ***/
         case IDM_MOD1:
            if (DialogBox (hInstance, "ModCodeRan", hwnd, lpfmodcode))
               {
               hdc = GetDC(hwnd);
               sprintf(ori,"");
               sprintf(axi,"");
               sprintf(axj,"");
               AfficherDonnees(hdc);
               ReleaseDC(hwnd,hdc);
               };
            return 0;

/*** choix IFS ***/
         case IDM_SEL1:
            if (DialogBox (hInstance, "SelectIFS", hwnd, lpfnselect))
```

```
                    {
                    hdc = GetDC(hwnd);
                    sprintf(ori,"");
                    sprintf(axi,"");
                    sprintf(axj,"");
                    AfficherDonnees(hdc);
                    EnableMenus(hMenu);
                    ReleaseDC(hwnd,hdc);
                    };
                 return 0;

/*** charger IFS ***/
            case IDM_LOI1:
                 if (DialogBox (hInstance, "LoadIFS1", hwnd, lpfnloadi1))
                    InvalidateRect(hwnd,NULL,TRUE);
                 return 0;

/*** sauver IFS ***/
            case IDM_SAI1:
                 if (DialogBox (hInstance, "SaveIFS1", hwnd, lpfnsavei1))
                    InvalidateRect(hwnd,NULL,TRUE);
                 return 0;

/*** changer couleurs ***/
            case IDM_CHC1:
                 DialogBox (hInstance, "Colorbox", hwnd, lpfncolors);
                 return 0;

/*** dessiner IFS ***/
            case IDM_DRA1:
                 if (DialogBox (hInstance, "DrawFract", hwnd, lpfndrawfr))
                 switch(IFSDraw)
                    {
                    case 1:
                       ObjetIFS1->Modifier_Domaine(25,40,125,40,25,140);
                       hdc=GetDC(hwnd);
                       sprintf(ori,"( %i , %i )     ",25,40);
                       sprintf(axi,"( %i , %i )     ",125,40);
                       sprintf(axj,"( %i , %i )     ",25,140);
                       AfficherDonnees(hdc);
                       ReleaseDC(hwnd,hdc);
                       SetCursor (LoadCursor (NULL,IDC_WAIT));
                       ObjetIFS1->Dessiner(hwnd);
                       SetCursor (LoadCursor (NULL,IDC_ARROW));
                       ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                       break;
                    case 2:
                       DisableMenus(hMenu);
                       paintmode=2;
                       SetCursor (LoadCursor (NULL,IDC_CROSS));
                       break;
                    }
                 return 0;

/*** zoom IFS ***/
            case IDM_ZOO1:
                 /*** activate zoom selection ***/
                 if (paintmode != 1)
                    {
                    DisableMenus(hMenu);
                    paintmode = 1;
                       hdc=GetDC(hwnd);
                       Rectangle(hdc,25,40,125,140);
                       sprintf(ori,"( %i , %i )     ",25,40);
                       sprintf(axi,"( %i , %i )     ",125,40);
                       sprintf(axj,"( %i , %i )     ",25,140);
                       AfficherDonnees(hdc);
                       ReleaseDC(hwnd,hdc);
                       ObjetIFS1->Modifier_Domaine(25,40,125,40,25,140);
                       SetCursor (LoadCursor (NULL,IDC_WAIT));
                       ObjetIFS1->Dessiner(hwnd);
                       ObjetIFS1->Modifier_Domaine(0,0,100,0,0,100);
                    SetCursor (LoadCursor (NULL,IDC_CROSS));
                    };
                 return 0;

/*** nouveau Meta-IFS ***/
            case IDM_SET2:
                 if (DialogBox (hInstance, "NewCodeMet", hwnd, lpfnewmet))
                    {
                    EnableMenus(hMenu);
                    hdc = GetDC(hwnd);
                    AfficherDonnees(hdc);
                    ReleaseDC(hwnd,hdc);
                    };
                 return 0;

/*** modifier Meta-IFS ***/
            case IDM_MOD2:
                 if (DialogBox (hInstance, "ModCodeMet", hwnd, lpfmodmet))
                    {
                    hdc = GetDC(hwnd);
                    AfficherDonnees(hdc);
                    ReleaseDC(hwnd,hdc);
                    };
                 return 0;

/*** insérer IFS dans M-IFS ***/
            case IDM_INS2:
                 hdc=GetDC(hwnd);
                 if (numifs<6)
                    {
                    if (ObjetMIFS1->Inserer((*ObjetIFS1)))
                       {
                       EnableMenuItem(hMenu,IDM_MCO2,MF_ENABLED);
                       EnableMenuItem(hMenu,IDM_DRA2,MF_ENABLED);
                       ObjetIFS1 = ObjetIFS1->Clone();
                       sprintf(Comp[numifs],NomIFS);
                       numifs += 1;
                       AfficherDonnees(hdc);
                       };
                    };
                 ReleaseDC(hwnd,hdc);
                 return 0;

/*** insérer pt_h dans M-IFS ***/
            case IDM_INH2:
                 hdc=GetDC(hwnd);
                 if (numifs<6)
                    {
                    if (ObjetMIFS1->Inserer((*pol)))
                       {
                       EnableMenuItem(hMenu,IDM_MCO2,MF_ENABLED);
                       EnableMenuItem(hMenu,IDM_DRA2,MF_ENABLED);
                       pol = pol->Clone();
                       sprintf(Comp[numifs],NomPTH);
                       numifs += 1;
                       AfficherDonnees(hdc);
                       };
                    };
                 ReleaseDC(hwnd,hdc);
                 return 0;
```

```
*** appliquer transf du M-IFS à IFS ***/
              case IDM_APP2:
                   if (DialogBox (hInstance, "SelAppIFS", hwnd, lpfnappifs))
                      ObjetMIFS1->Applic_Trans(IFSAppSe,(*ObjetIFS1));
                   return 0;

*** appliquer transf du M-IFS à PTH ***/
              case IDM_APH2:
                   if (DialogBox (hInstance, "SelAppIFS", hwnd, lpfnappifs))
                      ObjetMIFS1->Applic_Trans(IFSAppSe,(*pol));
                   return 0;

*** montrer composition ***/
              case IDM_MCO2:
                   hdc = GetDC(hwnd);
                   Rectangle(hdc,25,40,125,140);
                   ReleaseDC(hwnd,hdc);
                   ObjetMIFS1->Transformer(25,40,125,40,25,140);
                   SetCursor (LoadCursor (NULL,IDC_WAIT));
                   ObjetMIFS1->Montrer_Composition(hwnd);
                   SetCursor (LoadCursor (NULL,IDC_ARROW));
                   return 0;

*** choix Meta-IFS ***/
              case IDM_SEA2:
                   if (DialogBox (hInstance, "SelMetIFS", hwnd, lpfnselmet))
                      {
                      hdc = GetDC(hwnd);
                      AfficherDonnees(hdc);
                      EnableMenus(hMenu);
                      ReleaseDC(hwnd,hdc);
                      };
                   return 0;

*** dessiner M-IFS ***/
              case IDM_DRA2:
                   if (DialogBox (hInstance, "DrawMeta", hwnd, lpfndrawme))
                   {
                   switch(MIFSDraw)
                      {
                      case 1:
                         ObjetMIFS1->Transformer(215,40,615,40,215,420);
                         SetCursor (LoadCursor (NULL,IDC_WAIT));
                         if (MIFSTyDr==1)
                            ObjetMIFS1->Dessiner(hwnd,nprod);
                         else
                            ObjetMIFS1->Dessiner_Chaos(hwnd,nprod);

                         SetCursor (LoadCursor (NULL,IDC_ARROW));
                         break;
                      case 2:
                         DisableMenus(hMenu);
                         paintmode=5;
                         SetCursor (LoadCursor (NULL,IDC_CROSS));
                         break;
                      }
                   };
                   return 0;

*** zoom Meta-IFS ***/
              case IDM_ZOO2:
                   /*** activate zoom selection ***
                   if (zoom != 1)
                      {
                      SetCursor (LoadCursor (NULL,IDC_CROSS));
                      zoom = 1;
                      }   ***/
                   ...

/*** sélection Figure ***/
                   case IDM_PTH1:
                        tab[0].x=50;tab[0].y=0;
                        tab[1].x=50;tab[1].y=100;
                        delete pol;
                        pol = new Polygone(0,0,0,0,0,0,2,tab);
                        sprintf(NomPTH,"Ligne v.  \0");
                        hdc=GetDC(hwnd);
                        EnableMenus(hMenu);
                        AfficherDonnees(hdc);
                        ReleaseDC(hwnd,hdc);
                        return 0;
                   case IDM_PTH2:
                        tab[0].x=0;  tab[0].y=50;
                        tab[1].x=100;tab[1].y=50;
                        sprintf(NomPTH,"Ligne h.  \0");
                        delete pol;
                        pol = new Polygone(0,0,0,0,0,0,2,tab);
                        hdc=GetDC(hwnd);
                        EnableMenus(hMenu);
                        AfficherDonnees(hdc);
                        ReleaseDC(hwnd,hdc);
                        return 0;
                   case IDM_PTH3:
                        tab[0].x=0  ;tab[0].y=0;
                        tab[1].x=100;tab[1].y=0;
                        tab[2].x=50 ;tab[2].y=100;
                        sprintf(NomPTH,"Triangle\0");
                        delete pol;
                        pol = new Polygone(0,0,0,0,0,0,3,tab);
                        hdc=GetDC(hwnd);
                        EnableMenus(hMenu);
                        AfficherDonnees(hdc);
                        ReleaseDC(hwnd,hdc);
                        return 0;
                   case IDM_PTH4:
                        tab[0].x=0   ;tab[0].y=0;
                        tab[1].x=100 ;tab[1].y=0;
                        tab[2].x=100 ;tab[2].y=100;
                        tab[3].x=0   ;tab[3].y=100;
                        sprintf(NomPTH,"Rectangle\0");
                        delete pol;
                        pol = new Polygone(0,0,0,0,0,0,4,tab);
                        hdc=GetDC(hwnd);
                        EnableMenus(hMenu);
                        AfficherDonnees(hdc);
                        ReleaseDC(hwnd,hdc);
                        return 0;

/*** changement couleurs ***/
                   case IDM_ATT3:
                        DialogBox (hInstance, "ColorFig", hwnd, lpfncolfig);
                        return 0;

/*** dessiner figure ***/
                   case IDM_DRA3:
                        pol->Modifier_Domaine(25,40,125,40,25,140);
                        pol->Dessiner(hwnd);
                        pol->Modifier_Domaine(0,0,100,0,0,100);
                        return 0;

/*** informations ***/
                   case IDM_ABO:
                        DialogBox (hInstance, "Aboutbox", hwnd, lpfnabout);
                        return 0;
```

```
/*** reset 100x100 ***/
                case IDM_RE1:
                     hdc=GetDC(hwnd);
                     Rectangle(hdc,25,40,125,140);
                     ReleaseDC(hwnd,hdc);
                     return 0;

/*** reset 400x400 ***/
                case IDM_RE2:
                     hdc=GetDC(hwnd);
                     Rectangle(hdc,215,40,615,420);
                     ReleaseDC(hwnd,hdc);
                     return 0;

/*** reset écran ***/
                case IDM_RE3:
                     InvalidateRect(hwnd,NULL,TRUE);
                     return 0;

/*** charger/dessiner bitmap ***/
                case IDM_LOB3:
                     hdc=GetDC(hwnd);
                     DrawBitmap(hdc,hBitmap,216,41);
                     ReleaseDC(hwnd,hdc);
                     return 0;

/*** déterminer dimension 400x400 ***/
                case IDM_BOX3:
                     hdc=GetDC(hwnd);
                     SetCursor (LoadCursor (NULL,IDC_WAIT));
                     Rectangle(hdc,216,17,615,36);
                     TextOut(hdc,230,18,"Algorithme 'BOX-Counting' applique au d
omaine 400x400 ",53);
                     dim=Dimension(hdc,216,41,614,419);
                     if (dim!=-1)
                        {
                        sprintf(szString,"Dimension de l'image fractale : %f",di
m);
                        sprintf(Dim400,"%f1.3",dim);
                        AfficherDonnees(hdc);
                        MessageBox(hwnd,szString,"",MB_ICONINFORMATION);
                        }
                     else
                        {
                        MessageBox(hwnd,"Pas de figure sur le domaine 400x400","
",MB_ICONEXCLAMATION);
                        };
                     SetCursor (LoadCursor (NULL,IDC_ARROW));
                     ReleaseDC(hwnd,hdc);
                     return 0;

/*** déterminer dimension 100x100 ***/
                case IDM_BO13:
                     hdc=GetDC(hwnd);
                     SetCursor (LoadCursor (NULL,IDC_WAIT));
                     Rectangle(hdc,25,17,124,36);
                     TextOut(hdc,30,18,"Box-Counting",12);
                     dim=Dimension(hdc,26,41,124,139);
                     if (dim!=-1)
                        {
                        sprintf(szString,"Dimension de l'image fractale : %f",di
m);
                        sprintf(Dim100,"%f1.3",dim);
                        AfficherDonnees(hdc);
                        MessageBox(hwnd,szString,"",MB_ICONINFORMATION);
                        }
                     else
                        {
                        MessageBox(hwnd,"Pas de figure sur le domaine 100x100","
",MB_ICONEXCLAMATION);
                        };
                     SetCursor (LoadCursor (NULL,IDC_ARROW));
                     ReleaseDC(hwnd,hdc);
                     return 0;

/*** quit éditeur ***/
                case IDM_QUI:
                     SendMessage(hwnd, WM_CLOSE, 0, 0L);
                     return 0;
                }
                break;

        case WM_CLOSE:
             quit = MessageBox(hwnd,"Ceci termine l'editeur d'IFS  -  Have a ni
ce day !"
                                    ,"Quitter Editeur"
                                    ,MB_OKCANCEL);
             if (quit==IDOK)
                SendMessage(hwnd, WM_DESTROY, 0, 0L);
             return 0;

        case WM_DESTROY:
             PostQuitMessage(0);
             return 0;
        }
        return DefWindowProc(hwnd,Message,wParam,lParam);
}
```

Annexe G :

La diskette ci-jointe contient l'ensemble des programmes établis dans le cadre de ce mémoire. Les 54 fichiers ont une taille d'environ 1.1 MB et ils sont repris dans 5 répertoires.

Un petit programme d'installation (INSTALL.BAT) fournit le copiage des programmes exécutables comprises dans le répertoire EXE.

Les programmes suivants sont exécutables sous DOS :

* S_ALGDET Algorithme Déterministe implémenté en PASCAL avec comme code IFS le triangle de Sierpinski.

* F_ALGRAN Algorithme Random (ou chaos) implémenté en PASCAL avec un farn comme code IFS.

* F3D_ARAN Algorithme Random en 3 dimensions qui dessine un farn.

* T_ARCOUL Algorithme Random avec coloriage implémenté en PASCAL montrant une texture.

* F_ARCOUL Algorithme Random avec coloriage implémenté en PASCAL produisant un farn.

* F_ARCANI Algorithme Random avec coloriage implémenté en PASCAL sur un farn qui montre la possibilité d'une animation.

Les fichiers-sources de ces programmes se trouvent dans le 2ème répertoire TURBO_PA.

* DOSFRACT contient tous les classes implémentées en C++. Il existe deux versions de cette partie objet : une version DOS (DOSFRACT) et une version Windows (WINFRACT). Les fichiers-sources de DOSFRACT se trouvent dans le répertoire CPP/DOS et le listing est en annexe E.

* BONJOUR est l'application Windows décrite précédemment

* FR est l'éditeur d'image fractales. Il utilise des classes définies dans Winfract et son listing est en Annexe F.

Après installation, 'Bonjour' et 'Editeur d'IFS' doivent être incluses dans l'environnement Windows. La démarche à suivre est proposée lors de l'exécution du programme d'installation. Les fichiers-sources de BONJOUR se trouvent au répertoire CPP et ceux de FR en CPP/WINDOWS