MASTER EN SCIENCES INFORMATIQUES

Traitement des images fractales

Hilgers, Irène

Award date:
1991

Awarding institution:
Universite de Namur

TRAITEMENT DES IMAGES FRACTALES

IRENE HILGERS

Promoteur : J. FICHEFET

Septembre 1991

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
rue de Bruxelles 61, B-5000 NAMUR
Tél. 081-72.41.11
Télex 59222 facnam-b
Téléfax 081-23.03.91

Traitement des images fractales

HILGERS IRENE

Résumé

La géométrie fractale, introduite par B. Mandelbrot, permet de décrire la structure des objets naturels. En se basant sur leurs propriétés fractales, on parvient à les représenter graphiquement.
Ce travail étudie deux algorithmes, appelées algorithme "IFS" et "systèmes de Lindenmayer" et les applique, pour dessiner des plantes.

Abstract

The fractal geometry, introduced by B. Mandelbrot, permits to describe the structure of natural objects. In taking their fractal properties into account, these objects can be represented graphically.
This work studies two algorithms, the IFS theory and Lindenmayer systems, and applies them to draw plants.

Mémoire de licence et maîtrise en Informatique

Septembre 1991

Promoteur : J. FICHEFET

Au terme de ce mémoire, qu'il me soit permise de remercier Monsieur le Professeur J. FICHEFET pour avoir accepté de promouvoir ce travail.
J'adresse aussi mes plus vifs remerciements à Madame DE BAENST-VANDENBROUCKE pour sa disponibilité.
Enfin, que toute personne ayant directement ou indirectement contribué à la réalisation de ce mémoire soit assurée de mon entière gratitude.

Introduction

La description des structures naturelles à l'aide de la géométrie euclidienne s'avère souvent impossible à faire, puisque ces structures sont irrégulières, rugueuses, fragmentées, ou les trois à la fois. De ce fait, leur représentation graphique à l'aide d'un ordinateur s'est révélée trop compliquée et trop complexe.

Au milieu des années septante, B. Mandelbrot a introduit la géométrie fractale pour décrire des objets fragmentés, irréguliers et scalants, c'est-à-dire semblables à toutes les échelles : exactement les propriétés vérifiées par les objets naturels. De plus, en considérant ces objets comme des fractales, on parvient à les représenter graphiquement.

Dans ce mémoire, nous présentons deux méthodes, appelées l'algorithme "IFS" et "système de Lindenmayer", qui permettent de dessiner des fractales à l'écran. Comme objet naturel, nous avons choisi les plantes, dont la caractéristique principale est la structure ramifiée, qui peut être fractale. Cette caractéristique est également observable dans le réseau fluvial, les voies respiratoires, le réseau de vaisseaux sanguins...

Les deux méthodes étudiées permettent de dessiner un objet, l'une à partir d'une série de transformations affines, l'autre à partir d'un ensemble de productions, et de stocker seulement ces derniers, ce qui permet de gagner de la place en mémoire. Cette description courte d'une image compliquée ouvre des nouvelles voies dans le domaine de la transmission des images de télévision ou de celles engendrées par les ordinateurs. Par ailleurs la reproduction réaliste de la croissance et du développement d'une plante facilite la prévoyance de son comportement et de sa production en fruits ou en fleurs.

Le premier chapitre explique les concepts importants de la géométrie fractale et la compare avec la géométrie euclidienne. Le deuxième chapitre étudie les plantes, tandis que les chapitres suivants introduisent les deux méthodes pour dessiner les fractales. Le cinquième chapitre contient une analyse critique des deux méthodes.

L'implémentation de l'algorithme, basé sur les systèmes de Lindenmayer, se trouve en annexe.

1er Chapitre : La géométrie fractale

1. Introduction

"Pourquoi la géométrie est-elle souvent perçue comme une discipline "froide" et "sèche" ? L'une des raisons est son incapacité à décrire la forme d'un nuage, d'une montagne, d'un littoral ou d'un arbre. Les nuages ne sont pas des sphères, les montagnes ne sont pas des cônes, les côtes ne sont pas des cercles, l'écorce d'un arbre n'est pas lisse et la foudre ne se propage pas en ligne droite".

Dans cette citation, trouvée dans le livre "The Fractal Geometry of Nature" (1982), l'auteur, le mathématicien français Benoît Mandelbrot exprime l'idée qui est à la base de toutes ses recherches : la géométrie euclidienne classique, avec ses figures lisses et régulières, est incapable de donner une représentation fidèle du monde qui nous entoure. Beaucoup de structures naturelles sont trop irrégulières, trop rugueuses, trop fragmentées, ou les trois à la fois, pour être décrites par la géométrie habituelle. De sorte que si l'on veut analyser mathématiquement ces structures - au lieu de les considérer comme amorphes ou trop complexes pour être décrites -, il faut utiliser une "nouvelle" géométrie.

2. Fondement historique

D'une certaine manière, cette nouvelle géométrie existait depuis un siècle : dans une lettre de Cantor à Dedekind (20 juin 1877), l'auteur remettait en cause certains fondements de la géométrie euclidienne, notamment la notion de dimension. En 1890, Peano annonçait l'existence de courbes capables de remplir un carré. D'autres courbes apparentées à la précédente ont été engendrées par Cantor (1884), Von Koch (1904), etc. Elles sont des "figures intermédiaires" entre points et lignes, lignes et surfaces, ou surfaces et volumes. On les avait classés dans le domaine des mathématiques pures, abstraites et dépourvues de toute application réaliste.

Mandelbrot a montré que ces courbes historiques permettent de définir et de construire des formes géométriques très proches des formes naturelles, par exemple, certaines approximants de la courbe de Peano fournissent un modèle

géométrique de réseau fluvial [réf. 7] et il les a baptisé "fractales", du latin fractus : brisé, fragmenté.

3. Les Fractales

Les fractales réussissent là où la géométrie euclidienne échoue, à cause d'une combinaison - a priori contradictoire - d'extrême irrégularité et de profonde symétrie. L'irrégularité se voit tout de suite : rien n'est lisse dans les fractales; tout n'est que lignes brisées, surfaces rugueuses et fragmentées, formes disposées dans le plus complet désordre (apparent). La symétrie est moins évidente et ne doit pas être comprise dans le sens habituel (symétrie par rapport à un point, un axe, un plan, etc), mais dans le sens de l'harmonie des parties par rapport au tout. Cela signifie que si l'on examine les formes fractales sous un grossissement de plus en plus puissant, certains détails changent, mais la structure générale reste la même.

Selon Mandelbrot, il s'agit d'une propriété fondamentale des structures naturelles : en observant un arbre, on constate que toute partie de cet arbre ressemble à l'arbre tout entier. De la même façon, le sommet d'une montagne est analogue à la montagne complète, sauf qu'il est à une échelle plus petite. Autrement dit, le monde qui nous entoure est entièrement dominé par des structures dont les parties sont harmonieuses au tout.

En termes scientifiques, on dit que ces structures possèdent une invariance d'échelle - puisqu'elles sont semblables à toutes les échelles - ou encore qu'elles sont "scalantes" (néologisme forgé par Mandelbrot d'après l'anglais scaling, qui signifie graduation, échelle). Cette symétrie profonde confère aux fractales, en dépit de leur irrégularité, une structure très fortement organisée.

1°) Formes intermédiaires

Sur le plan mathématique, les fractales ont des propriétés assez curieuses. La géométrie euclidienne est construite selon une rigoureuse hiérarchie : le point, la droite, le plan, l'espace. De sorte que les objets géométriques autres que des points appartiennent nécessairement à l'une des catégories suivantes : lignes (droites, brisées, courbes...), surfaces ou volumes. Les fractales se comportent comme si elles n'avaient aucun respect de cette hiérarchie : certaines sont, en un sens, intermédiaires entre un point et une ligne, d'autres sont intermédiaires entre une ligne et une surface, d'autres entre une surface et un volume.

Il existe beaucoup d'exemples dans la littérature pour montrer que cette notion de "formes intermédiaires" est tout le contraire d'une abstraction mathématique : si on choisit des échelles de mesure de plus en plus petites, la longueur d'une côte tend vers l'infini. Ce comportement est tout à fait différent de ce qui se passe lorsqu'on mesure une ligne ordinaire : le résultat de la mesure de la longueur d'un cercle de manière approchée par un polygone régulier va tendre vers une limite finie qui est la longueur exacte du cercle donnée par la formule $2 \Pi R$ (où R est le rayon du cercle).

Toute ligne ordinaire se comporterait de la même manière, c'est-à-dire que des mesures approchées de plus en plus précise tendraient vers une limite finie, la longueur exacte de la ligne. Tandis que la côte se comporte comme si elle avait une longueur infinie, bien qu'elle soit limitée quant à son extension spatiale. En ce sens, on peut affirmer que la côte est "plus qu'une ligne". Mais elle est aussi "moins qu'une surface", puisqu'elle sépare la terre de la mer. Par conséquent, le contour d'un littoral est une forme intermédiaire entre une ligne et une surface.

De la même façon, une forme naturelle peut être intermédiaire entre une surface et un volume : le poumon a une surface tellement plissée (extrêmement grande), tout en restant contenue à l'intérieur d'un volume fini.

On peut conclure de ces deux exemples :

- Il y a un lien direct entre l'irrégularité d'une forme fractale, le fait qu'on ne puisse pas la mesurer et la propriété d'invariance d'échelle. Ainsi, ce qui différencie la côte d'une ligne ordinaire, c'est qu'elle présente des irrégularités à toutes les échelles, et par conséquent en nombre infini. Mais ces irrégularités sont toutes bâties, grosso modo, sur le même modèle : ce sont des baies et des péninsules, à l'intérieur desquelles se trouvent des baies et des péninsules semblables, mais plus petites, qui, elles-mêmes...etc. Si cette "cascade" s'arrêtait très rapidement, par exemple au deuxième ou troisième ordre, la côte pourrait être décrite par une ligne géométrique classique : elle n'aurait qu'un nombre fini d'irrégularités et sa longueur serait clairement définie. Mais la forme ne serait plus scalante, puisqu'à partir d'une certaine échelle on ne retrouverait plus la structure générale de départ.

- Les fractales remettent en cause la notion usuelle de dimension. La géométrie euclidienne attribue en effet la dimension 1 à un ligne, la dimension 2 à une surface et la dimension 3 à un volume. La question qui se pose est de savoir

quelle dimension attribuer à une forme intermédiaire entre une ligne et une surface et entre une surface et un volume.

2°) Les dimensions d'un objet fractal

Une fractale possède à la fois une dimension entière correspondant à la notion classique, et une dimension non nécessairement entière définie plus loin.

1. Dimension topologique

La première dimension est appelée la "dimension topologique" et intuitivement, elle reflète la manière dont les éléments d'une forme sont "connectés" entre eux. En fait, elle correspond à la définition rigoureuse de la dimension usuelle, et ne peut donc prendre que des valeurs entières. En ce qui concerne la dimension topologique, les formes fractales ne diffèrent en rien des formes ordinaires.

2. Dimension fractale : Approche intuitive

La deuxième dimension est liée à la notion de "mesure de contenu", au sens le plus large. La géométrie euclidienne ne connaît que trois type de contenus : la longueur, l'aire, le volume. De plus, il y a une correspondance entre ces trois types de contenus et la dimension topologique : un point, de dimension 0, a aussi une longueur, une surface et un volume nuls. Une ligne (non fractale), dont la dimension topologique est 1, possède une longueur non nulle (et finie si la ligne est limitée dans l'espace) mais une aire et un volume nuls. Une surface, de la même façon, possède une aire non nulle, mais un volume nul, et une longueur infinie. Enfin, un volume possède un volume fini, mais son aire et sa longueur sont infinis. Il ne s'agit pas, bien entendu, de l'aire extérieure qui est finie, mais de l'aire d'une surface qui remplirait totalement le volume.

Il est ainsi possible d'associer à chaque forme, en plus de la dimension topologique, une "dimension de contenu", qui est par définition égale à 0 pour un point, à 1 pour une longueur, à 2 pour une aire, à 3 pour un volume. Ainsi définie, cette "dimension de contenu" s'identifie dans le cas des objets géométriques ordinaires à la dimension topologique.

Une ligne fractale, de dimension topologique égale à 1, se comporte comme si sa "dimension de contenu" est supérieure à 1, puisqu'elle a une longueur infinie, mais en même temps inférieure à 2, puisqu'elle a une aire nulle. Et que, de la même façon, une surface fractale, de volume nul, peut avoir une aire

infinie. Cette situation suggère fortement que les formes fractales peuvent avoir une "dimension de contenu" qui n'est pas égale à leur dimension topologique, et qui n'est pas nécessairement entière. Enfin, la "dimension de contenu", à l'opposé de la dimension topologique, varie continûment. Sa valeur reflète le degré d'irrégularité, ou de rugosité, de la forme considérée. Comme les fractales sont plus irrégulières que les formes ordinaires, elles ont une "dimension de contenu" supérieure à leur dimension topologique. Sur le plan mathématique, on peut d'ailleurs les définir par cette dernière propriété.

Plus une fractale est irrégulière, plus sa "dimension de contenu" - ou dimension fractale, selon la terminologie de Mandelbrot - est élevée. Cette dimension est encore appelée dimension de Hausdorff-Besicovitch.

3. Dimension fractale : Définition mathématique

1ère définition : Commençons avec le cas non fractal où la dimension possède une valeur entière. Dans ce cas, elle indique si ce que l'on peut mesurer est un contenu de type longueur, aire, ou volume. Ceci peut s'exprimer dans une formule mathématique simple et identique pour les trois cas.

Pour mesurer un contenu, qu'il s'agisse d'une longueur, d'une aire ou d'un volume, on choisit un contenu unité, et on regarde combien de fois l'unité est elle-même contenue dans la grandeur mesurée. Le résultat est une mesure approchée à l'unité près. La mesure exacte est obtenue en faisant décroître la taille de l'unité : si la mesure approchée tend vers une limite finie lorsque la taille de l'unité tend vers zéro, cette limite finie constitue la mesure exacte.

Ainsi, pour mesurer une longueur, on prend comme unité un segment de droite de longueur connue u, et l'on détermine le nombre de fois que ce segment est contenu dans la longueur à mesurer. Si on appelle N(u) ce nombre, la mesure approchée à l'unité u près est égale au produit N(u) * u. La mesure exacte est la limite de N(u) * u, lorsque u tend vers zéro. On obtient une formule analogue dans le cas d'une surface. L'unité est alors un carré de côté u, donc d'aire u^2, et la mesure approchée est N(u) * u^2, où N(u) désigne cette fois le nombre minimum de carrés unités qu'il faudrait utiliser pour recouvrir complètement la surface. La mesure exacte est la limite de N(u) * u^2, lorsque u tend vers zéro. Il est facile de voir que, de la même façon, la mesure exacte d'un volume peut être définie comme la limite de N(u) * u^3 lorsque u tend vers zéro et que N(u) désigne le nombre de fois qu'un cube unité d'arête u est contenu dans le volume à mesurer.

On voit que, dans les trois cas, la mesure exacte s'exprime quasiment par la même formule, la seule chose qui change étant l'exposant de l'unité u : 1 pour une longueur, 2 pour une aire, 3 pour un volume. Si on appelle d cet exposant, la formule devient alors identique dans les trois cas : la mesure exacte d'une longueur, d'une surface ou d'un volume est la limite de $N(u) * u^d$ lorsque u tend vers zéro, d valant respectivement 1, 2 ou 3. Cet exposant d n'est autre que la "dimension de contenu" de la forme étudiée.

On remarque que du point de vue purement mathématique, l'expression $N(u) * u^d$ conserve un sens même si d n'est pas égal à 1, 2 ou 3, mais a une valeur intermédiaire. En effet, les fonctions exponentielles permettent de définir la valeur de u^d pour n'importe quelle valeur non entière de d. Par conséquent, la limite de $N(u) * u^d$ peut aussi être calculée même si d n'est pas un entier.

La mesure de la longueur d'une côte s'exprime par exemple mathématiquement ainsi : lorsque d est égal à 1, la limite de $N(u) * u^d$, u tendant vers zéro, est pour la côte égale à l'infini; mais d'autre part, pour d égal à 2, la limite est nulle (puisque la côte a une aire nulle). Pour une certaine valeur de d, appelé D, comprise entre 1 et 2, la limite a une valeur qui soit à la fois finie et différente de zéro.

On peut dire que le "contenu" de la côte est "trop grand" pour pouvoir être mesuré dans la dimension 1, mais "trop petit" pour être mesuré dans la dimension 2. En revanche, dans la dimension D comprise entre 1 et 2, la mesure est possible.

Ceci est assez analogue à ce qui se passe pour une surface ordinaire : elle a en effet un contenu trop important pour que sa longueur ait une valeur finie, mais trop faible pour avoir un volume non nul.

La "dimension de contenu" d'une forme quelconque (qu'elle soit fractale ou classique) peut donc se définir, en tout généralité, comme étant la dimension, pour laquelle le contenu de cette forme peut être mesuré. La dimension d'une forme fractale vaut $1 < D < 2$, quand la dimension topologique est égal à 1. Elle vaut $2 < D < 3$, pour une dimension topologique égale à 2 et elle peut être $3 < D < 4$ pour une dimension topologique égale à 3.

2ième définition : Une autre manière de "voir" la dimension fractale est étroitement liée à la propriété d'invariance d'échelle. Considérons un objet à 1 dimension, une droite par exemple, on peut le diviser en N parties identiques, dont chacune est réduite d'un facteur

$$r = \frac{1}{N}$$

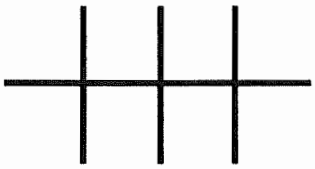
de l'objet entier. De la même manière, un objet à deux dimensions, par exemple, un carré dans un plan, peut être divisé en N parties identiques, dont chacune est réduite d'un facteur

$$r = \frac{1}{\sqrt{N}}$$

de l'objet entier. Pour un objet à trois dimensions, on trouve un facteur

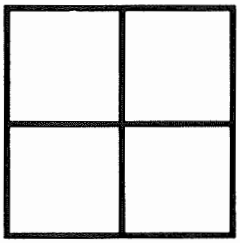
$$r = \frac{1}{\sqrt[3]{N}}$$

Représentation graphique



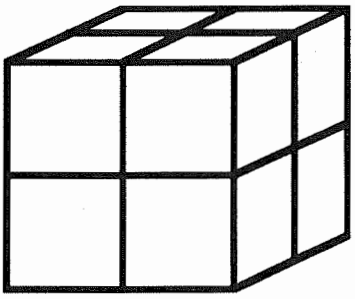
1-D N parties, réduites dun facteur $r = \frac{1}{N}$

$N\,r = 1$



2-D N parties, réduites d'un facteur $r = \frac{1}{\sqrt{N}}$

$N\,r^2 = 1$



3-D N parties, réduites d'un facteur $r = \frac{1}{\sqrt[3]{N}}$

$N\,r^3 = 1$

La propriété d'invariance d'échelle permet directement de généraliser ce facteur. Un objet scalant à D dimensions peut être divisé en N copies plus petites, dont chacune est réduite d'un facteur

$$r = \frac{1}{\sqrt[D]{N}}$$

de l'objet,

ou

$$N = \frac{1}{r^D}$$

Inversément, étant donné un objet scalant de N parties réduites d'un rapport r de l'objet entier, sa dimension fractale est donnée par :

$$D = \frac{\log N}{\log \frac{1}{r}}$$

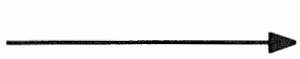
3°) Définition

Selon Mandelbrot, "un objet fractal est un objet, dont la dimension fractale (aussi appelée dimension de Hausdorff-Besicovitch) est strictement plus grande que la dimension topologique" et aussi "un objet dont la dimension de Hausdorff-Besicovitch n'est pas un nombre entier".

4. Exemples

Une ligne fractale simple se construit aisément : il faut prendre un générateur et le reproduire en changeant d'échelle sur chacun des éléments du générateur.

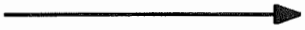
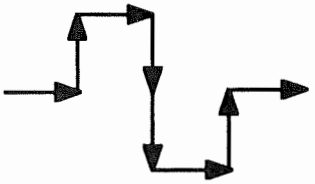
1°) Courbes de Von Koch

1)  remplacé par 

$N = 4, r = \frac{1}{3}$

$D = \frac{\log(4)}{\log(3)} = 1.2618$

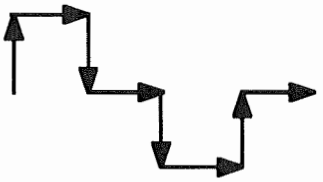
voir figure 1.1

2)  remplacé par 

$N = 8, r = \frac{1}{4}$

$D = \frac{\log(8)}{\log(4)} = 1.5$

voir figure 1.2

3)  remplacé par 

$N = 8, r = \frac{1}{4}$

$D = \frac{\log(8)}{\log(4)} = 1.5$

voir figure 1.3

2°) Explications

Sur les trois exemples, on voit bien la propriété d'invariance d'échelle des fractales. Une autre caractéristique des fractales est : à une dimension fractale ne correspond pas une et une seule figure et la dimension fractale ne dépend pas de l'échelle d'observation.

En fait, la première courbe exactement scalante de Von Koch peut être considérée comme un modèle rudimentaire d'une côte. Mais les deux fractales diffèrent d'un aspect significatif : la courbe de Von Koch, à différents

agrandissements, a toujours la même forme, tandis que si on examine une côte sous un agrandissement de plus en plus puissant, certains détails changent, mais la structure générale reste la même. Dans certains livres [réf. 3], ces objets sont appelés auto-semblables statistiquement, où la notion "auto-semblable" est un autre mot pour scalant.

On a constaté [réf. 3], que la dimension fractale d'une côte réelle est comprise entre 1.15 et 1.25, et est donc proche de la dimension fractale

$$\frac{\log (4)}{\log (3)} = 1.2618$$

de la courbe de Von Koch.

La propriété que des objets auto-semblables statistiquement diffèrent en détails à différentes échelles, est une caractéristique centrale des fractales naturelles. Une autre propriété des fractales naturelles est l'existence d'une plus petite et d'une plus grande échelle de mesure.

5. Comparaison entre la géométrie euclidienne et la géométrie fractale

Géométrie euclidienne	**Géométrie fractale**
* traditionnelle (>2000 ans)	* monstres modernes (≅ 10 ans)
* basée sur une grandeur caractéristique ou échelle	* pas de grandeur spécifique ou échelle
* convient à des objets produits par l'homme	* appropriée pour des formes naturelles
* décrite par des formules	* algorithmes récursifs

Ce tableau résume les différences majeures entre la géométrie euclidienne et la géométrie fractale.

- Les fractales sont décidément des inventions modernes. Bien qu'on puisse trouver des ancêtres des fractales à la fin du siècle précédé, leur utilité n'a été découverte par les scientifiques que dans le courant des dix dernières années.

- Tandis que les formes euclidiennes ont une ou plusieurs grandeurs caractéristiques ou longueurs (par exemple : le rayon d'une sphère, le côté d'un

cube), les fractales ne possèdent pas de grandeurs caractéristiques. Les formes fractales sont dites scalantes.

- La géométrie euclidienne fournit des descriptions précises des objets faits par l'homme, mais ne convient pas pour les formes naturelles, parce qu'elle donne des descriptions lourdes et incorrectes. Les fractales fournissent une description excellente de beaucoup de formes naturelles et ont donné lieu à un engouement naturel pour l'imagerie par ordinateur.

- Finalement, les formes euclidiennes sont décrites par des formules algébriques simples (par exemple : $r^2 = x^2 + y^2$ défini un cercle de rayon r), alors que les fractales, en général, sont le résultat d' algorithmes qui sont souvent récursifs et idéalement adaptés à l'ordinateur.

2ième Chapitre : Les plantes

Dans ce mémoire, nous nous intéressons en particulier aux végétaux, des objets naturels, dont la caractéristique principale est la structure ramifiée.

Ce chapitre explique la morphologie d'une plante et les facteurs écologiques qui peuvent influencer sa croissance. Ensuite, il étudie les propriétés fractales et se termine avec les différentes méthodes pour dessiner les plantes à l'ordinateur.

1. Caractéristiques principales

Commençons avec les caractéristiques principales : les végétaux sont des êtres vivants qui sont presque tous fixés à un substrat, par exemple le sol. Ils n'ont pas de système nerveux et sont donc en apparence insensibles et immobiles. Ils n'ingèrent pas d'aliments solides; ils absorbent, par imbition et osmose, des substances gazeuses ou dissoutes dans l'eau.

La plupart des végétaux contiennent de la chlorophylle qui leur permet de transformer des substances minérales comme le gaz carbonique ou l'eau en substances organiques (en glucides). De tels végétaux sont dits autotrophes (autos = "soi-même"; trophê = "nourriture") parce qu'ils fabriquent eux-mêmes leurs aliments.

2. Morphologie d'une plante

Du point de vue de sa morphologie externe, un végétal typique est constitué de trois organes différents : tige, feuilles, racines.

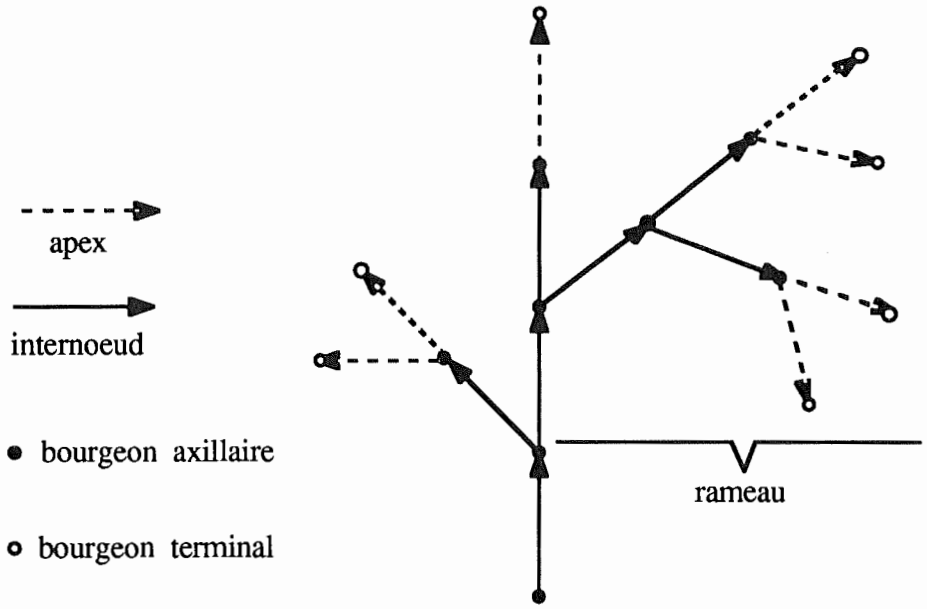
1°) Tige

La tige se présente comme un axe mince dont un bourgeon, dit terminal, occupe le sommet. Le long de la tige, des renflements appelés noeuds portent les feuilles; l'espace compris entre deux renflements constitue un internoeud. Cette dernière notion a son importance, en particulier, si on envisage la croissance de la plante. A l'aisselle de chaque feuille, on peut observer un autre bourgeon, appelé bourgeon axillaire.

Le bourgeon terminal est le moteur de la croissance en long de la tige. Des bourgeons terminaux déterminent aussi la croissance des tiges secondaires (ou rameaux), développées à partir de certains bourgeons axillaires. En tout cas, le système de ramification de plantes herbacées est, en général, simple. Ceci provient également de la faible durée de vie des parties aériennes.

Avec les tiges ligneuses comme celles des arbustes et, bien entendu, des arbres, le système de ramification est beaucoup plus important. Il se développe sur des années. Tandis que les arbres possèdent une tige principale, l'arbuste est un ensemble touffu de tiges qui jaillit du sol.

Représentation graphique



2°) Les feuilles et la tige

L'étude de la disposition des feuilles le long de la tige constitue à elle seule une discipline spécialisée, la phyllotaxie. Cette disposition ne se fait pas au hasard. Elle obéit à certaines règles.

On peut définir deux grands types de dispositions foliaires. La disposition verticillée correspond au fait que deux ou plusieurs feuilles sont insérées simultanément à chaque noeud de la tige. Pour chaque noeud, l'ensemble des feuilles présentes correspond à un verticille foliaire. Si on considère un ensemble de verticilles, on constate que chaque feuille à une disposition alternée de l'un à

l'autre. Dans le cas le plus général, le verticille comporte seulement deux feuilles opposées.

Dans la disposition alterne, chaque noeud ne porte qu'une seule feuille. Chez les Monocotylédones, végétaux où la graine n'a qu'un cotylédon (feuille qui naît sur l'axe du graine), les feuilles successives sont souvent implantées en opposition. Sur un rameau de Lin, on trouve un autre disposition : les insertions foliaires s'ordonnent selon une hélice s'enroulant à droite ou à gauche à partir de la base.

Chez une plante adulte, la disposition des feuilles est caractérisée. Au cours du développement, cependant, ou de la croissance des rameaux, on peut rencontrer des dispositions particulières. Quel que soit le mode de distribution des feuilles sur le rameau âgé, les deux premières feuilles sont en général opposées. De la même façon, la disposition des feuilles primordiales de la tige principale peut être particulière. Chez certaines Dicotylédones, ces feuilles primordiales sont souvent verticillées, alors que les feuilles suivantes sont en disposition alterne.

3°) Tiges souterraines et rhizomes

Chez des plantes vivaces, les rhizomes sont des parties de tiges souterraines épaissies et jouant un rôle de réserve des substances nutritives. Ils présentent généralement des parties renflées et des parties étrécies.

Sur la face supérieure, au niveau des parties renflées, se trouvent les cicatrices d'anciennes tiges aériennes. La croissance souterraine du rhizome se fait grâce à un ou plusieurs bourgeons axillaires et donne une structure ramifiée. Suivant l'orientation de la croissance, on distingue des rhizomes à croissance plagiotrope (rhizomes dits horizontaux) et des rhizomes à croissance orthotrope (rhizomes verticaux). Quelle que soit l'orientation de sa croissance, il n'émerge jamais du sol.

A chaque saison végétative, et à partir du bourgeon terminal, le rhizome ne produit, normalement, qu'une seule tige aérienne. Lorsque celle-ci meurt, le rhizome persiste, subit une période de dormance hivernale et, au-delà, se remet à croître.

Pour être complet, il faut citer qu'il existe des plantes qui possèdent des tubercules (exemple: Pomme de terre), tandis que d'autres ont un bulbe (exemple : Tulipe).

3. Les végétaux dans leur environnement

1°) Facteurs écologiques

Les végétaux qui croissent et se reproduisent dans un environnement déterminé sont sous la dépendance de nombreux facteurs écologiques. Pour ne considérer que les milieux terrestres, dans lesquels les végétaux supérieurs sont particulièrement bien représentés, on peut distinguer au moins :

- des facteurs climatiques et atmosphériques (lumière, température, humidité de l'air...) liés au vent et aux perturbations atmosphériques.

- des facteurs édaphiques, c'est-à-dire liés au sol. Les uns sont physiques (texture, structure commandant la compacité et l'aération du sol, stabilité, hydratation, température...); les autres sont chimiques (pH, teneur en calcium, en chlorures, en nitrates, carence en certains éléments...).

- des facteurs topographiques entraînant des modifications climatiques, atmosphériques et édaphiques.

- des facteurs biotiques, autrement dit, propres à l'environnement biologique. Il s'agit soit d'interactions entre végétaux : influence de la microflore du sol (bactéries et champignons); parasitisme (entre végétaux supérieurs; ou maladies cryptogamiques provoquées par des champignons); phénomènes de compétition. Il s'agit aussi d'interactions entre végétaux et animaux : destruction par des herbivores (animal qui se nourrit d'herbes, de feuilles); pollinisation par les insectes...

2°) Accommodation

Dans le cours du temps, l'environnement d'un végétal ne reste jamais constant. Si les variations des conditions écologiques sont de faible amplitude, il les supporte sans que son phénotype, l'ensemble des manifestations morphologiques d'un génotype, soit apparemment modifié. Au contraire, pour des variations plus importantes, l'organisme végétal réagit, s'adapte.

Le développement et la différenciation du végétal sont donc intimement liés à son environnement. On dit qu'il y a accommodation.

Un exemple typique sont les variations d'éclairement. Elles font apparaître des accommodats, des modifications morphologiques, non transmises aux descendants, portant sur le port, la surface et l'anatomie des feuilles. Sur un même arbre, les feuilles sont différentes selon qu'elles sont situées à l'extérieur ou à l'intérieur de sa couronne.

Un autre exemple est la compétition entre végétaux. Qu'ils appartiennent à la même espèce ou à des espèces différentes, les individus occupant une même station sont en compétition pour les sources d'énergie et de matières premières. Chez les végétaux supérieurs, cette concurrence se crée, dans les peuplements très serrés, pour la lumière, pour l'eau (quand la réserve du sol devient insuffisante) et pour le éléments minéraux assimilables.

Les interactions entre les végétaux peuvent aussi prendre un autre aspect. Les substances émises par les racines de certaines espèces peuvent, à distance, avoir des effets favorables ou défavorables sur la germination et la croissance d'autres espèces. Ainsi, l'émission d'acides aminés par les racines de Légumineuses (famille de plantes angiospermes dicotylédones, comprenant des arbres, des arbustes ou des herbes dont le fruit est une gousse) favorise la croissance des Graminées (famille de plantes phanérogames angiospermes) en améliorant leur nutrition azotée.

4. Propriétés fractales

Après avoir décrit la morphologie des plantes et les facteurs ayant une influence sur la croissance, on va considérer maintenant les propriétés fractales des plantes.

Dans son livre "The Fractal Geometry of Nature" (1983), Mandelbrot appelle les arbres des fractales non-uniformes, où "non-uniforme" veut dire que la dimension fractale D et / ou la dimension topologique D_T prennent différentes valeurs pour différentes parties de cet ensemble.

La partie aérienne et la partie souterraine d'un végétal sont des objets fractals. Elles possèdent comme tout objet naturel une plus grande et une plus petite échelle d'observation. Leur caractéristique principale est la structure ramifiée qu'on retrouve également dans le réseau fluvial, les voies respiratoires, le réseau de vaisseaux sanguins...

On constate en observant la partie aérienne d'un végétal, que toute partie de ce végétal ressemble au végétal tout entier. La dimension fractale mesure en quelque sorte le degré de ramification, c'est-à-dire la manière dont il remplit l'espace dans lequel il se trouve (ici : l'espace à 3 dimensions). Moins le végétal possède de ramifications, plus sa dimension fractale se rapproche de 1. A la limite où aucune "branche" ne pousse sur la tige, sa dimension fractale est égale à 1. Inversement, lorsque le végétal est très touffu, autrement dit lorsqu'il tend à occuper tout l'espace dans lequel il se trouve, sa dimension fractale tend vers 3. Pour un végétal fractal idéal, la dimension fractale reste la même selon que l'on considère le végétal tout entier ou bien l'une de ses parties les plus petites. Cela signifie qu'il existe, en fait, autant de "trous" ou d'espaces vides de grande taille que de petite taille.

5. Modélisation

Il y a différentes méthodes pour dessiner des plantes à l'ordinateur [réf. 4]. Les méthodes étudiées ici se basent sur les propriétés fractales des plantes.

Elles s'appellent :
- **L'algorithme "IFS"** ("Iterated function systems")
- **Systèmes de Lindenmayer** (ou systèmes L).

La première méthode est un outil pratique pour produire des images comprenant des nuages et de la fumée, des horizons, la mer, des structures ramifiées comme des arbres et des fougères, et de simples objets artificiels.

La deuxième méthode, qui est basée sur l'interprétation de chaînes de caractères, a été créée pour modeler la géométrie des plantes et permet également de générer des courbes fractales classiques, par exemple la courbe de Von Koch.

D'un point de vue strict, aucune figure obtenue avec ces deux méthodes n'est fractale, parce que les figures consistent en un nombre fini de lignes. Ceci à comme conséquence que la dimension fractale est égale à la dimension topologique. Les figures sont à considérer comme des approximations des objets fractals.

3ième Chapitre : L'algorithme " IFS"

L'algorithme "IFS", en se basant sur la propriété d'invariance d'échelle, permet de dessiner des objets fractals. Il utilise le concept de transformation affine pour exprimer des relations entre différentes parties d'un objet géométrique.

1. Transformation affine

1°) Définition

Une transformation affine W est définie par :

$$W : R^2 \rightarrow R^2 :$$

$$(x, y) \rightarrow W(x, y) = (ax + by + e, cx + dy + f)$$

où a, b, c, d, e et f sont des nombres réels.

Elle transforme des points du plan en des nouveaux points du plan.

Elle peut également s'exprimer sous forme de matrice :

$$W\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ax + by + e \\ cx + dy + f \end{bmatrix}$$

Il s'agit d'une transformation linéaire, représentée par la matrice 2*2, suivie d'une translation.

Exemple :

$W : R^2 \rightarrow R^2 :$

$$(x, y) \rightarrow W(x,y) = (0.5x + 0.25y + 1, 0.25x + 0.5y + 0.5)$$

ou sous forme de matrice,

$$W\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0.25 \\ 0.25 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1 \\ 0.5 \end{bmatrix}$$

Une application de cette transformation est montrée sur la figure 3.1. La transformation est appliquée à la grande figure et a comme résultat la petite figure. On constate que les yeux sont plus proches dans la figure

résultante : la transformation est contractive, en ce sens qu'elle rapproche toujours 2 points.

2°) Contractivité

Etant donné une transformation affine W : R^2 -> R^2, on peut toujours trouver un nombre non négatif s tel que

$$||W(\vec{x}) - W(\vec{y})|| \leq s \cdot |\vec{x} - \vec{y}| \quad \forall \ \vec{x}, \vec{y}$$

où

$$||\vec{x}|| = \sqrt{x_1^2 + x_2^2}$$

Le plus petit nombre s pour lequel ceci est vrai est appelé la constante de Lipschitz de W.

La transformation affine est dite contractive si s < 1, elle est dite symétrique lorsque

$$||W(\vec{x}) - W(\vec{y})|| = ||\vec{x} - \vec{y}|| \quad \forall \ \vec{x}, \vec{y}$$

et elle est dite expansive si s > 1.

Les transformations affines utilisées dans ce mémoire sont toutes contractives.

3°) Attracteur

En appliquant continûment une transformation affine contractive à une figure, par exemple un rectangle, on obtient des rectangles de plus en plus petits. A la limite, on obtient un point unique (à cause de la contractivité), qui est appelé *point fixe*. On remarquera que le point fixe d'une transformation affine contractive de R^2 dans R^2 n'est rien d'autre que la solution (unique) de l'équation :

$$W\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Ce point a comme propriété qu'il n'est pas déplacé quand on lui applique la transformation. Dans le cas d'une transformation affine unique, ce point est

l'attracteur du "système de fonctions itérées" (en anglais : Iterated Function System); c'est la fractale définie par le système.

2. Le système de fonctions itérées : IFS

Un IFS est un ensemble de transformations affines contractives. Cet ensemble définit une image fractale unique, appelée l'attracteur du IFS.

1°) Définition des notion d'IFS et d'attracteur pour le cas particulier de R^2

Un IFS est un ensemble fini $\{W_1, W_2, ..., W_N\}$ de transformations affines contractives W_i: R^2 -> R^2, de constantes de contractions $s_i < 1$ (i = 1, ..., N). La constante

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

est appelée la constante de contraction de l'IFS.

On démontre que :

Si A_0 est un sous-ensemble compact (càd borné et fermé) de R^2 et si :

$$A_1 = \bigcup_{i=1}^{N} W_i(A_0)$$

$$A_2 = \bigcup_{i=1}^{N} W_i(A_1)$$

$$\vdots$$

$$A_n = \bigcup_{i=1}^{N} W_i(A_{n-1})$$

alors la suite $\{A_n : n=0, 1, 2, 3,...\}$ converge vers un sous-ensemble A compact unique de R^2, lequel est appelé l'attracteur ou image fractale de l'IFS et est tel que

$$A = \bigcup_{i=1}^{N} W_i(A)$$

Il convient de noter que la convergence visée ci-dessus est celle au sens de Hausdorff, c'est-à-dire telle que :

$$\lim_{n \to \infty} h(A_n, A) = 0,$$

où :

$$h(A_n, A) = \max (d(A_n, A), d(A, A_n))$$

avec:

$$d(A_n, A) = \max_{z \in A_n} \min_{t \in A} |z - t|$$

$$d(A, A_n) = \max_{t \in A} \min_{z \in A_n} |z - t|$$

2°) Condition de contractivité moyenne

Soient N transformations affines W_1, W_2, W_3 ...W_N avec $W_i : R^2 \to R^2$ pour i=1,...N, et les probabilités p_1, p_2, ... p_N (où p_i est la probabilité que W_i soit sélectionnée), avec $p_i > 0$ pour i = 1, ...N et $p_1 + p_2 + ... + p_N = 1$.

Si s_n dénote la constante de Lipschitz de W_n pour n = 1...N, alors on dit que le code IFS obéit à la condition de contractivité moyenne si

$$s_1^{p_1} \cdot s_2^{p_2} \cdots s_N^{p_N} < 1$$

Un code IFS est un IFS $\{ W_n, p_n : n = 1, 2, ...N \}$ tel que la condition de contractivité moyenne est vérifiée.

Exemple :

$$W_1\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 8 \\ 8 \end{bmatrix}$$

$$W_2\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 96 \\ 16 \end{bmatrix}$$

$$W_3\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 96 \\ 60 \end{bmatrix}$$

Ceci est l'ensemble de transformations affines contractives du "triangle de Sierpinski". Les 3 points fixes sont (16,16), (192,32) et (192, 120) respectivement, et constituent les 3 sommets du triangle.

Son code IFS est donné par :

0.5	0	0	0.5	8	8	0.33
0.5	0	0	0.5	96	16	0.33
0.5	0	0	0.5	96	60	0.34

Sur la figure 3.2, on voit que l'attracteur, l'image fractale associée au IFS, obéit

$$\text{Attracteur} = W_1(\text{Attracteur}) \cup W_2(\text{Attracteur}) \cup W_3(\text{Attracteur})$$

c'est-à-dire qu'il est l'union de 3 copies réduites de lui-même.

En général, un IFS peut contenir un nombre quelconque, N, de transformations affines contractives W_1, W_2, W_3 ...W_N . En conséquence, il existe un attracteur unique qui satisfait

$$\text{Attracteur} = W_1(\text{Attracteur}) \cup W_2(\text{Attracteur}) \cup \ldots W_N(\text{Attracteur})$$

3. Algorithme

Un algorithme permettant de dessiner l'attracteur d'un ensemble de transformations affines a été découvert par M. Barnsley. Commençons avec le "jeu du chaos", qui explique l'idée de base de l'algorithme.

1°) "Jeu du chaos"

Considérons le code IFS du "triangle de Sierpinski" :

0.5	0	0	0.5	8	8	0.33
0.5	0	0	0.5	96	16	0.33
0.5	0	0	0.5	96	60	0.34

qui correspond aux trois transformations affines suivantes :

$$W_1\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 8 \\ 8 \end{bmatrix}$$

$$W_2\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 96 \\ 16 \end{bmatrix}$$

$$W_3\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 96 \\ 60 \end{bmatrix}$$

Les coordonnées des points fixes de ces transformations sont (16,16), (192,32), (192, 120) respectivement. Dans ce cas particulier, on peut vérifier que le résultat de l'application d'une de ces 3 transformations à un point (x, y) dans le plan euclidien est le point à mi-chemin entre (x, y) et le point fixe de la transformation. A l'aide d'un dé qui montre les numéros 1, 2 et 3 sur ses faces, on peut maintenant jouer le "jeu du chaos" : il faut dessiner les 3 points fixes sur une feuille de papier et les appeler 1, 2 et 3. Ensuite, on choisit un $4^{ième}$ point n'importe où sur la feuille, qui est appelé (x_0, y_0). On lance le dé et on dessine un point appelé (x_1, y_1) et situé à mi-chemin entre (x_0, y_0) et le point dont le numéro est donné par le dé. En fait, on a appliqué la transformation sélectionnée à l'aide du dé au point (x_0, y_0). On relance le dé et on dessine le point suivant (x_2 , y_2) à mi-chemin entre (x_1, y_1) et le point dont le numéro est donné par le dé. Ce processus est répété pour obtenir un grand nombre de points : (x_0, y_0), (x_1, y_1), (x_2 , y_2), (x_3 , y_3), (x_4 , y_4), (x_5 , y_5) ...

Un résultat de base de la théorie des systèmes de fonctions itérées est la convergence de cette séquence de points avec 100% de probabilité vers l'attracteur de l'IFS. En plus, si (x_0, y_0) est actuellement un des points fixes, tous les points de la séquence se trouvent sur l'attracteur.

La même méthode peut être utilisée pour obtenir l'image d'un attracteur correspondant à un code IFS.

2°) Algorithme (Random Iteration Algorithm)

En général, un IFS peut contenir N transformations affines $W_1, W_2, W_3 ...W_N$, dont chacune a une certaine probabilité d'être sélectionnée. Les N probabilités $p_1, p_2, ... p_N$ doivent vérifier l'équation suivante : $p_1 + p_2 + ... + p_N = 1$ et $p_i > 0$ pour i = 1, 2, ...N. p_i est la probabilité avec laquelle la transformation i est sélectionnée. En théorie, les valeurs données n'influencent pas le résultat, mais en pratique elles ont une influence sur le temps mis pour dessiner la fractale.

L'algorithme

(1) x = 0; y = 0.

(2) Choisir comme valeur de k une réalisation aléatoire d'une variable aléatoire possédant la distribution $\{p_j; j = 1, ...,N\}$

(3) Appliquer la transformation W_k au point (x, y) pour obtenir

(new x, new y),

$$\begin{bmatrix} \text{new x} \\ \text{new y} \end{bmatrix} = W_k \begin{bmatrix} x \\ y \end{bmatrix}$$

(4) x = new x; y = new y.

(5) dessiner (x, y).

(6) retourner en (2).

Il existe deux manières pour choisir le point initial de l'itération :

- soit on le choisit au hasard. Dans ce cas, on ne dessine pas les premiers points de l'itération.

- soit on prend le point fixe d'une transformation affine, déterminé par calcul.

4. Détermination d'une transformation affine

Pour illustrer la détermination d'une transformation affine $W : R^2 \rightarrow R^2$, considérons la copie de 2 feuilles, une grande et une petite, comme montré sur la figure 3.3 . On veut trouver les nombres réels a, b, c, d, e et f pour que la transformation

$$W\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ax + by + e \\ cx + dy + f \end{bmatrix}$$

ait la propriété de changer la grande feuille en une feuille approximativent égale à la petite feuille.

On commence par introduire les axes de coordonnées x et y et par déterminer 3 coordonnées de la grande feuille : (α_1, α_2), (β_1, β_2) et (v_1, v_2). On

dessine les points correspondants sur la petite feuille et on détermine également leurs coordonnées $(\widetilde{\alpha_1}, \widetilde{\alpha_2})$, $(\widetilde{\beta_1}, \widetilde{\beta_2})$ et $(\widetilde{\nu_1}, \widetilde{\nu_2})$.

Alors a, b et e sont obtenus en résolvant les 3 équations :

$$\alpha_1 a + \alpha_2 b + e = \widetilde{\alpha_1}$$

$$\beta_1 a + \beta_2 b + e = \widetilde{\beta_1}$$

$$\nu_1 a + \nu_2 b + e = \widetilde{\nu_1}$$

tandis que c, d et f satisfont :

$$\alpha_1 c + \alpha_2 d + f = \widetilde{\alpha_2}$$

$$\beta_1 c + \beta_2 d + f = \widetilde{\beta_2}$$

$$\nu_1 c + \nu_2 d + f = \widetilde{\nu_2}$$

5. Détermination du code IFS

La détermination du code IFS se fait par essai-erreur. Il existe des logiciels pour faciliter cette détermination. Le logiciel "Desktop Fractal Design System" [réf. 11] réalisé par M. Barnsley, et qui permet soit d'afficher une fractale correspondant à un code IFS, soit d'étudier les différents effets sur un attracteur en ajoutant, en modifiant, en retirant une transformation d'un ensemble de transformations. Pour être complet, on doit encore signaler que le logiciel permet également de dessiner l'ensemble de Mandelbrot et les ensembles de Julia correspondants.

Chaque ensemble de points provenant de "l'application d'une transformation affine" à la figure entière est entouré par un polygone, qu'on peut rendre visible ou invisible. Un polygone est l'image du contour de la "fenêtre attracteur" sous la transformation.

Il existe des fonctions agissant sur les polygones. On peut agrandir ou réduire la grandeur d'un polygone en changeant les valeurs des variables a, b, c, d, e et f. On peut également tourner un polygone sélectionné, ajouter ou enlever

des polygones, c'est-à-dire des transformations. Il existe aussi une fonction qui permet de changer la valeur d'une des variables a, b, c, d, e et f d'une transformation. Toutes ces fonctions modifient bien entendu l'attracteur.

En regardant un code IFS fait d'une seule transformation, on peut déterminer et modifier son point fixe.

Pour déterminer le code IFS d'une figure, je n'ai pas réellement trouver de méthode. Une façon serait de se baser sur des codes IFS existants, pour essayer de dériver des nouveaux attracteurs à l'aide des fonctions définies plus haut.

Exemple :

0.33	0	0	0.33	84	28	0.16
0.33	0	0	0.33	84	56	0.16
0.33	0	0	0.33	102	34	0.17
0.33	0	0	0.33	102	50	0.17
0.33	0	0	0.33	66	34	0.17
0.33	0	0	0.33	66	50	0.17

Ce code IFS, qui a comme attracteur une fleur, a été dérivé du "triangle de Sierpinski".

Une autre façon, proposée dans la réf. 3, serait de commencer avec l'image dont on veut déterminer le code IFS, sous forme digitale. On y applique une transformation affine W_1 et le résultat est dessiné à l'écran. L'utilisateur adapte interactivement les nombres a, b, c, d, e et f pour que le résultat de cette transformation affine se trouve sur une partie de l'image initiale. Ensuite, on refait la même chose avec une deuxième transformation. On continue jusqu'à ce que l'attracteur corresponde à l'image initiale.

4ième Chapitre : Systèmes de Lindenmayer

1. Introduction

En 1968, Aristid Lindenmayer, directeur du groupe de biologie théorique à l'université d'Utrecht, développe un algorithme pour simuler la croissance d'organismes filamenteux : les algues microscopiques par exemple. A partir d'un caractère choisi au départ dans un "alphabet", il produit des séquences de caractères de plus en plus longues en appliquant de façon itérative un certain nombre de règles, appelées productions. Un exemple est donné ci-dessous :

caractère de départ : A

production 1 : $A \rightarrow I\ [\ L_0\ ]\ A$

production 2 : $A \rightarrow I\ [\ L_0\ ]\ B$

production 3 : $B \rightarrow I\ [\ I\ F_0\ ]\ B$

production 4 : $L_i \rightarrow L_{i+1} \quad i \geq 0$

production 5 : $F_i \rightarrow F_{i+1} \quad i \geq 0$

A chaque étape, toutes les lettres de la séquence produite à l'étape précédente sont remplacées par les suites de caractères spécifiées par les productions.

Ce type d'algorithmes avaient déjà été utilisé par des linguistes pour décrire la syntaxe des langages naturels. Ceux de Lindenmayer, les systèmes L, présentent deux originalités fondamentales : les règles opèrent simultanément sur tous les caractères de la séquence, et elles sont capables de décrire des structures arborescentes car elles contiennent des symboles spécifiant des ramifications. On constate en outre qu'un petit nombre de règles relativement compactes suffisent pour saisir la complexité de plantes très diverses. A partir des années 1970, Lindenmayer et d'autres affinent les systèmes L en vue de modéliser la croissance des plantes. Les premiers décrivaient des structures arborescentes, ainsi que les interactions internes entre leurs différentes parties, liées à la circulation de nutriments, d'hormones, etc. Les plus récents intègrent les règles relatives à l'ordre

et à la probabilité d'apparition des ramifications et organes de la plante, et à la croissance des fleurs et feuilles.

Les premiers dessins automatisés de plantes sont réalisés sur table traçante, en 1974, par D. Frijters et Lindenmayer, et par P. Hogeweg et B. Hesper. Et surtout, vers 1984, Alvy Ray Smith montre qu'il est possible d'utiliser les systèmes L pour la synthèse d'images de végétaux par ordinateur.

Dans l'exemple ci-dessus, l'apex A, ou sommet, commence par produire (production 1) une suite d'internoeuds I (portion de tige comprise entre deux groupes de feuilles ou de fleurs) et de feuilles L_i. Puis, à un certain moment (non spécifié ici), l'apex passe de l'état végétatif A à l'état de floraison B (production 2), et produit des fleurs F_i (production 3). Les productions 4 et 5 interdisent notamment aux feuilles de se transformer en fleurs et vice-versa. L'indice i indique l'âge des fleurs et des feuilles. Les accolades symbolisent les branchements dans la structure.

En 1985, P. Prusinkiewicz eut l'idée de transcrire les systèmes L sous une forme telle que ces suites de caractères soient interprétées comme des déplacements d'un curseur sur un écran d'ordinateur. Une extension de cette méthode permet d'engendrer des figures tridimensionnelles complexes. Outre la modélisation des plantes herbacées, les systèmes L permettent également la représentation graphique de courbes classiques comme par exemple la courbe de Von Koch.

Dans ce mémoire, je me suis surtout basée sur le travail de P. Prusinkiewicz [réf. 6].

2. Systèmes L

Cette partie du chapitre va essayer de donner une définition formelle de quelques systèmes L.

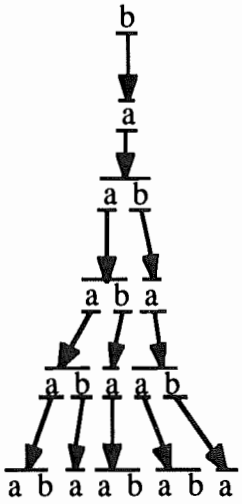
1°) Systèmes OL

Les systèmes OL sont des systèmes L qui sont indépendants du contexte des caractères à remplacer ("context-free").

Exemple

Considérons des chaînes faites de 2 caractères a et b. A chaque caractère est associé une règle. La règle a → ab signifie que la lettre a est remplacée par la chaîne ab, et la règle b → a signifie que la lettre b est remplacée par a. Chaque processus commence avec une chaîne distinguée appelée *axiome*. Si l'axiome est composé de la seule lettre b, alors, à la première dérivation, l'axiome b est remplacé par a en utilisant b → a. A la deuxième dérivation, a est remplacé par ab en utilisant a → ab. La chaîne ab consiste en 2 lettres, qui sont toutes les deux remplacées simultanément à la dérivation suivante. On obtient ainsi la chaîne aba et on continue les dérivations en remplaçant a par ab et b par a.

Représentation graphique



...

Définition : V dénote un alphabet, V* l'ensemble des mots de l'alphabet V et V+ l'ensemble des mots non-vides de V*. Une *chaîne système OL* est un triplet G = < V, w, P > où V est l'alphabet, w ∈ V+ est un mot non-vide appelé *axiome* et P c V * V* est un *ensemble* fini de *productions*.

Une production (a, χ) ∈ P est écrite a -> χ où a et χ sont appelés respectivement le *prédécesseur* et le *successeur*. On suppose que pour toute lettre a ∈ V, il existe au moins un mot χ ∈ V* tel que a → χ. Si aucune

production n'est explicitement spécifiée pour une lettre a ∈ V, la production identité a→a est supposée appartenir à l'ensemble de production P.

Soit $\mu = a_1 \ldots a_m$ un mot arbitraire de V^*. On dit que le mot $\gamma=\chi_1\chi_2\ldots\chi_m$ est *directement déductible* de μ et on écrit $\mu \Rightarrow \gamma$ si et seulement si $a_i \rightarrow \chi_i$; $\forall$ i = 1...m.

Un mot γ est *déductible de l'axiome* w, *en n dérivations* , s'il existe une séquence de mots $\mu_0, \mu_1, \mu_2, \ldots \mu_n$ telle que $\mu_0 = w$, $\mu_n = \gamma$ et $\mu_0 \Rightarrow \mu_1 \Rightarrow \mu_2 \ldots \Rightarrow \mu_n$.

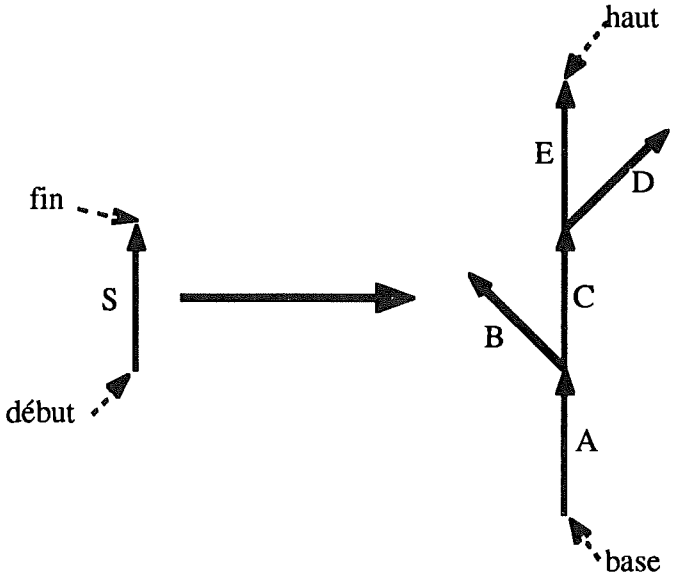
1. Systèmes DOL

Un *système OL* est *déterministe* (noté *système DOL*) si et seulement si $\forall$ a ∈ V, il y a exactement un $\chi \in V^*$ tel que $a \rightarrow \chi$.

2. Systèmes OL arborescents

Une plante peut être considérée mathématiquement comme une arborescence : les arcs forment des chemins entre un sommet spécifique, appelé racine au base, et les sommets terminaux. Pour représenter une plante au moyen de système L, on pourrait imaginer des productions qui remplacent un arc prédécesseur par une arborescence successeur telle que le sommet initial du prédécesseur est identifié avec la base du successeur et le sommet terminal avec le haut du successeur.

Représentation graphique



Définition : Un *système OL arborescent* G est spécifié par trois composants : un ensemble d'étiquettes d'arcs V, une arborescence initiale w avec des étiquettes de V, et un ensemble de productions P.

On notera que cette définition ne spécifie pas la structure de données pour représenter les arborescences. Aussi, pour représenter un système OL arborescent, Lindenmayer a-t-il introduit les accolades [ ], qui permettent de délimiter une branche.

Les dérivations des systèmes OL avec accolades procèdent comme les systèmes OL sans accolades. Les accolades sont remplacées par elles-mêmes.

3. Systèmes OL stochastiques

A partir d'un système DOL, on peut seulement engendrer une chaîne de caractères, ce qui veut dire que toutes les plantes générées par ce système sont identiques. Pour obtenir différents exemplaires d'une même espèce, il est nécessaire d'introduire des variations qui conservent les aspects généraux d'une plante mais qui changent ses détails. Les variations peuvent être obtenues en randomisant le système L, ce qui affecte la topologie et la géométrie de la plante.

Définition : Un *système OL stochastique* est un quadruplet $G_\pi = < V, w, P, \pi >$. L'alphabet V, l'axiome w et l'ensemble de productions P sont définis comme dans un système OL. La fonction $\pi : P \rightarrow (0, 1]$, appelée *distribution de probabilité* , admet que pour chaque lettre $a \in V$, la somme des probabilités de toutes les productions ayant a comme prédécesseur est égale à 1.

Une dérivation $\mu \Rightarrow \gamma$ est appelée *dérivation stochastique* si pour chaque occurrence de la lettre a dans le mot μ la probabilité d'appliquer la production p, dont le prédécesseur est a, est égale à π (p). Ainsi, des productions différentes ayant le même prédécesseur peuvent être appliquées à des occurrences de la même lettre dans une dérivation.

2°) Systèmes L contexte-sensitifs

1. Définition

Les productions vues jusqu'ici étaient applicables sans tenir compte du contexte, c'est-à-dire des caractères voisins du prédécesseur. Cependant, le choix d'une production applicable peut dépendre du contexte. Ceci peut être utile dans

la simulation d'interaction entre des parts d'une plante, par exemple le flux d'hormones ou de nourritures. Différentes extensions des systèmes L ont été proposées :

– ainsi, des systèmes 2L utilisent des productions de la forme $a_l < a > a_r \rightarrow \chi$, où la lettre a (appelée prédécesseur strict) est "remplacée" par le mot χ si et seulement si a est précédée par la lettre a_l et suivie par a_r. Ainsi, les lettres a_l et a_r forment le contexte "gauche" et le contexte "droite" de a dans cette production.

– ainsi encore, dans des systèmes 1L, les productions sont de la forme $a_l < a \rightarrow \chi$, ou $a > a_r \rightarrow \chi$.

Dans la spécification d'un système L contexte-sensitif, on peut se définir des "symboles ignorés", c'est-à-dire des symboles qui doivent être considérés comme non-existants quand on vérifie le contexte.

Systèmes OL, systèmes 1L et systèmes 2L appartiennent à une classe plus large de systèmes IL, appelés également systèmes (k, l). Dans un système (k, l), le contexte "gauche" est un mot de longueur k et le contexte "droite" est un mot de longueur l.

Dans un système L, des productions ayant différentes longueurs de contexte peuvent coexister. En plus, des productions dépendantes du contexte ont une priorité sur des productions indépendantes du contexte ("context-free"), qui ont le même prédécesseur. Si on ne peut pas appliquer une production à une lettre, elle est remplacée par elle-même.

2. Systèmes L arborescents

L'extension aux systèmes L arborescents demande de vérifier les arcs voisins de l'arc à remplacer. Un prédécesseur d'une production p consiste en trois composants : un chemin l formant le contexte "gauche", un arc S, appelé le prédécesseur strict et une arborescence r constituant le contexte "droite". L'asymétrie entre le contexte "gauche" et le contexte "droite" reflète le fait qu'il y a seulement un chemin partant de la racine vers l'arc à remplacer, tandis qu'il peut avoir plusieurs chemins de l'arc vers les sommets terminaux. Une occurrence d'un arc S dans un arbre T est remplacée par une arborescence s'il existe une production p dont le contexte "gauche" l est un chemin dans T se terminant au

sommet initial de l'arc S et le contexte "droite" r est une arborescence de T commençant au sommet terminal de S.

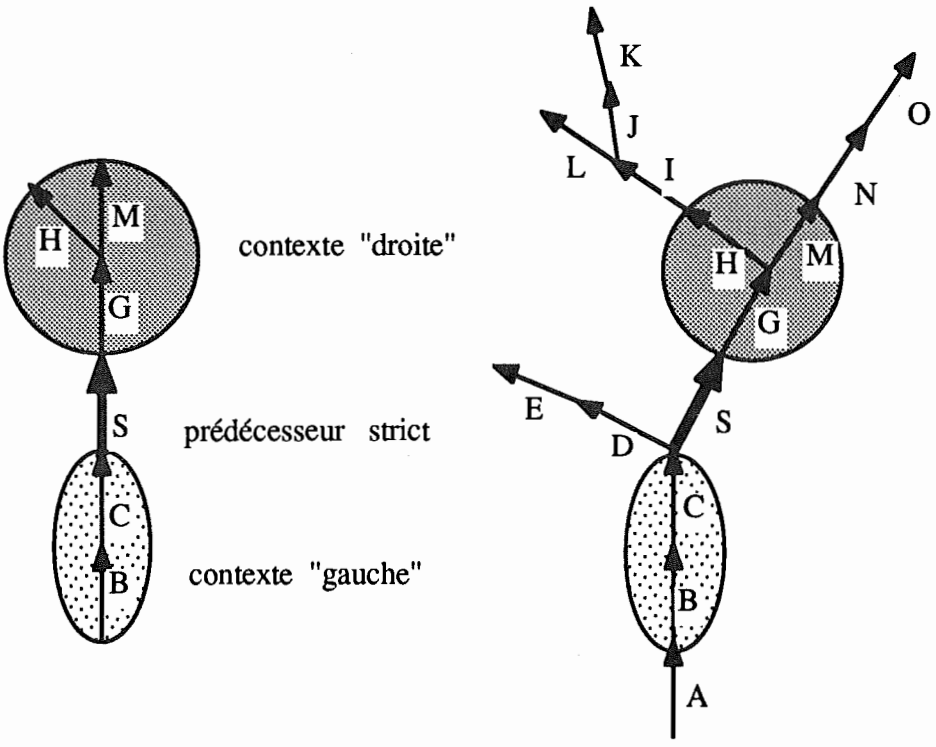
L'introduction du contexte dans des systèmes L avec accolades est plus difficile, parce que les chaînes de caractères ne conservent pas le voisinage des segments. Par conséquent, il peut arriver que la procédure vérifiant le contexte doive sauter des symboles représentant des branches.

L'exemple suivant indique qu'une production avec prédécesseur
BC < S > G [H] M peut être appliquée au symbole S dans la chaîne

A B C [DE] [**S** G [HI [J K] L]M N O]

qui implique de sauter les symboles [DE] dans la recherche du contexte "gauche" et I [JK] L dans la recherche du contexte "droite".

Représentation graphique



Exemple

Le contexte "gauche" peut être utilisé pour simuler des signaux de contrôle se propageant de la racine vers les noeuds terminaux de la plante, tandis que le contexte "droite" peut représenter des signaux qui se propagent des noeuds terminaux vers la racine.

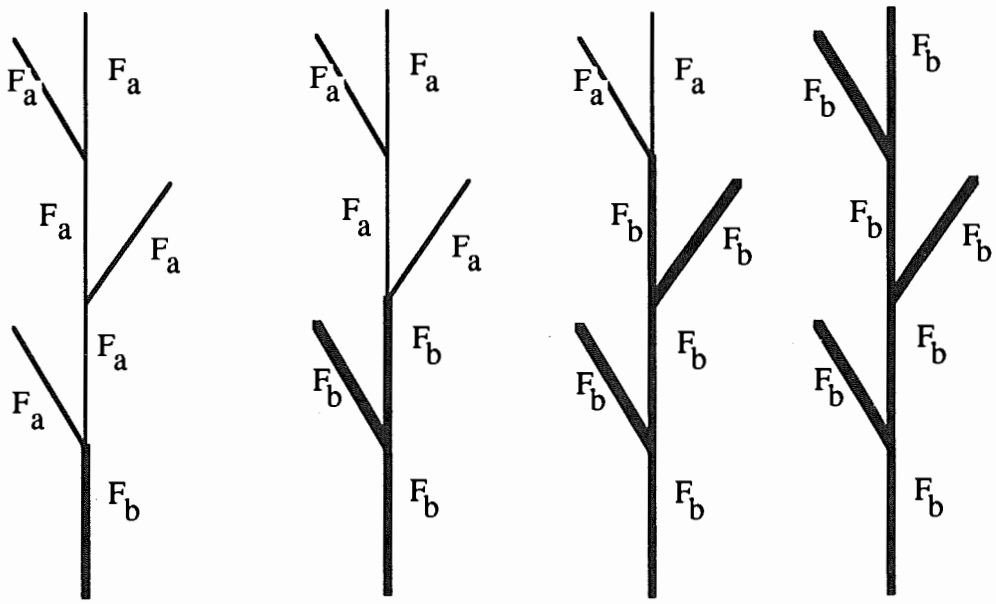
Les deux systèmes 1L suivants permettent de représenter la simulation de la propagation des signaux dans une structure ramifiée.

symboles ignorés : + -

axiome : F_b [+F_a] F_a [-F_a] F_a [+F_a] F_a

production : F_b < F_a -> F_b

Le symbole F_b représente un segment déjà atteint par le signal, tandis que F_a représente un segment pas encore atteint. Les images suivantes représentent des étapes consécutives de la propagation du signal.

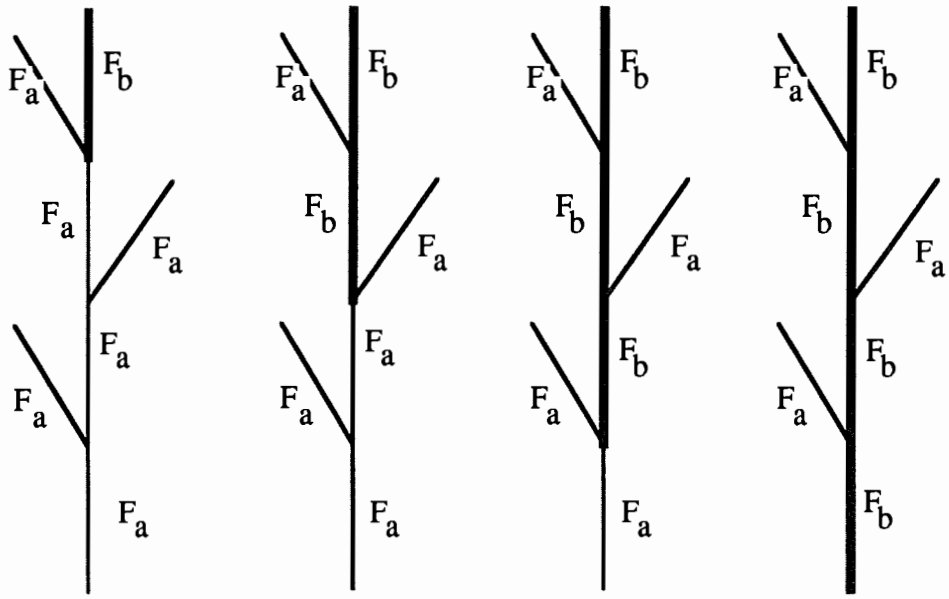


Le système 1L suivant simule la propagation d'un signal d'un noeud terminal vers la racine.

symboles ignorés : + -

axiome : F_a [+F_a] F_a [-F_a] F_a [+F_a] F_b

production : F_a > F_b -> F_b



3. L'interprétation graphique des chaînes

Il existe plusieurs interprétations graphiques d'une chaîne de caractères. Une interprétation, introduite par Abelson et diSessa [réf.12], utilisée par Prusinkiewicz, se base sur la notion de *tortue* de LOGO.

L'idée de base est la suivante : un *état* de la tortue est défini comme un triplet (x, y, α), où les coordonnées cartésiennes (x, y) représentent sa *position* et l'angle α son *orientation* .

Etant donné le *pas* d et *l'accroissement angulaire* δ, la tortue peut répondre à des commandes représentées par les symboles suivants :

F Avancer d'un pas d. Le nouvel état de la tortue est (x', y', α) où $x' = x + d.\cos \alpha$ et $y' = y + d.\sin \alpha$. Un segment de droite est tracé entre les points (x, y) et (x', y').

f Avancer d'un pas d sans tracer de segment de droite.

+ Tourner à droite d'un angle δ. Le nouvel état de la tortue est (x, y, $\alpha + \delta$). On suppose ici que l'orientation positive des angles est dans le sens des aiguilles d'une montre.

- Tourner à gauche d'un angle δ. Son état suivant est (x, y, $\alpha - \delta$).

[Mettre l'état courant de la tortue sur une pile. L'information sauvée contient la position et l'orientation de la tortue, mais peut également contenir la couleur et la largeur des segments de droite dessinés, ou toute autre caractéristique.

] Prendre un état de la tortue dans la pile, qui devient l'état courant. La position de la tortue change, mais aucun segment de droite n'est tiré.

Tous les autres symboles sont ignorés par la tortue, auxquels cas elle garde son état courant.

Si γ est une chaîne de caractères désignant des commandes, si (x_0, y_0, α_0) est l'état initial de la tortue et si d et δ sont des paramètres dont les valeurs sont fixées, on appelle *interprétation de la chaîne* γ par la tortue l'image (c'est-à-dire l'ensemble des segments de droite) que trace la tortue, au départ de son état initial, en réponse aux commandes de la chaîne γ.

Exemples

1°) La figure 4.1 représente 4 approximations d'une "île de Von Koch". Les figures sont obtenues en interprétant des chaînes générées par le système DOL suivant :

variation de l'angle : $\delta = 90°$

axiome : F + F + F + F

production : F $\rightarrow$ F + F - F - FF + F + F - F

Les images correspondent aux chaînes obtenues en 0, 1, 2, et 3 dérivations. d est diminué 4 fois entre les images subséquentes.

2°) L'exemple suivant est un système DOL contenant des accolades. La figure résultante se trouve sur la figure 4.2 (page 40).

nombre de dérivations : n = 4

variation de l'angle : $\delta = 22.5°$

axiome : F

production : F $\rightarrow$ F F + [+ F - F - F] - [- F + F + F]

3°) Un exemple d'un système L stochastique est donné par :

nombre de dérivations : n = 5

variation de l'angle : $\delta = 30°$

axiome : F

productions :

$$p_1 : F \xrightarrow{0.33} F[+F]F[-F]F$$

$$p_2 : F \xrightarrow{0.33} F[+F]F$$

$$p_3 : F \xrightarrow{0.34} F[-F]F$$

Au-dessus du symbole $\rightarrow$ se trouve la probabilité d'appliquer la production. Des exemples de structures ramifiées générées par ce système L sont montrés à la figure 4. 3. Ils ressemblent à des exemplaires différents d'une même espèce.

4°) Le dernier exemple est un système L contexte-sensitif. La figure résultante est à la figure 4.4 (page 41).

nombre de dérivations : n = 25

variation de l'angle : $\delta = 22.5°$

axiome : F1F1F1

symboles ignorés : + - F

productions :

$0 < 0 > 0 \rightarrow 1$

$0 < 0 > 1 \rightarrow 1[-F1F1]$

$0 < 1 > 0 \rightarrow 1$

$0 < 1 > 1 \rightarrow 1$

$1 < 0 > 0 \rightarrow 0$

$1 < 0 > 1 \rightarrow 1F1$

$1 < 1 > 0 \rightarrow 1$

$1 < 1 > 1 \rightarrow F0$

$* < + > * \rightarrow -$

$* < - > * \rightarrow +$

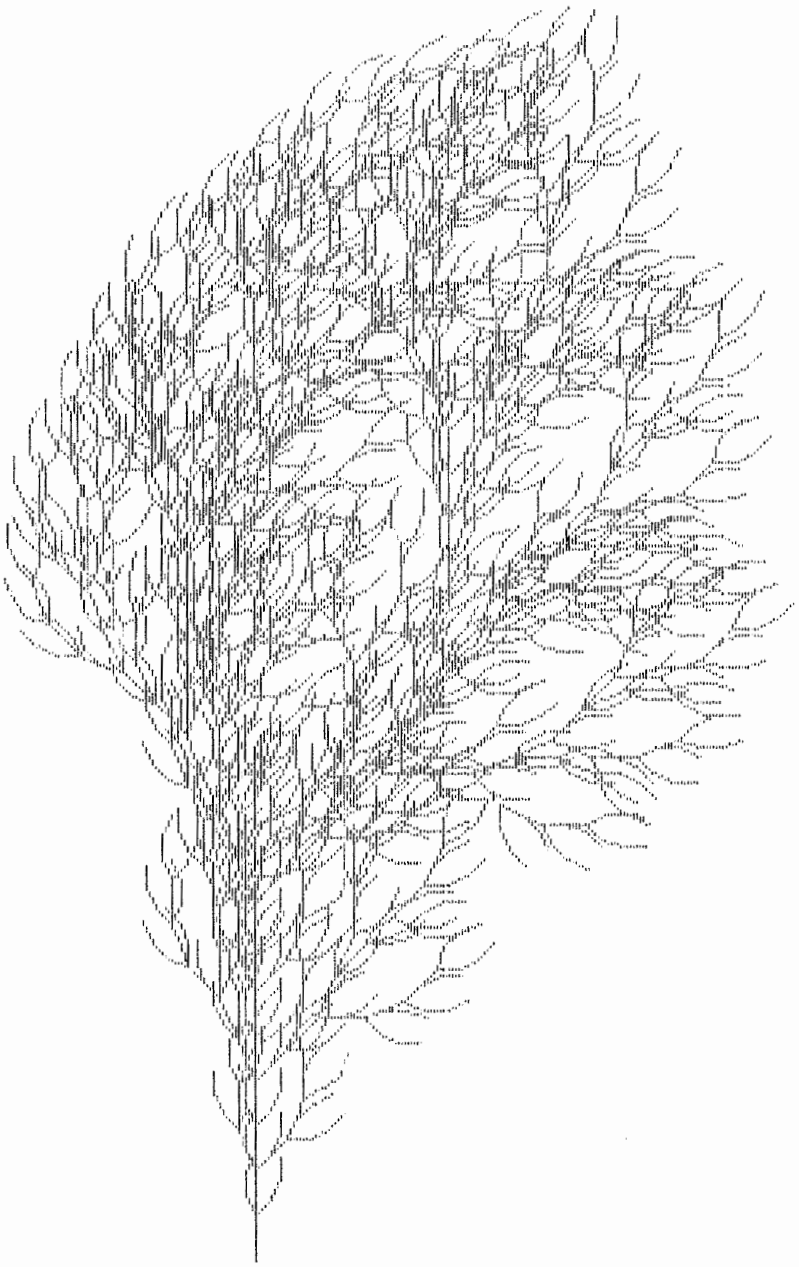


figure 4.2

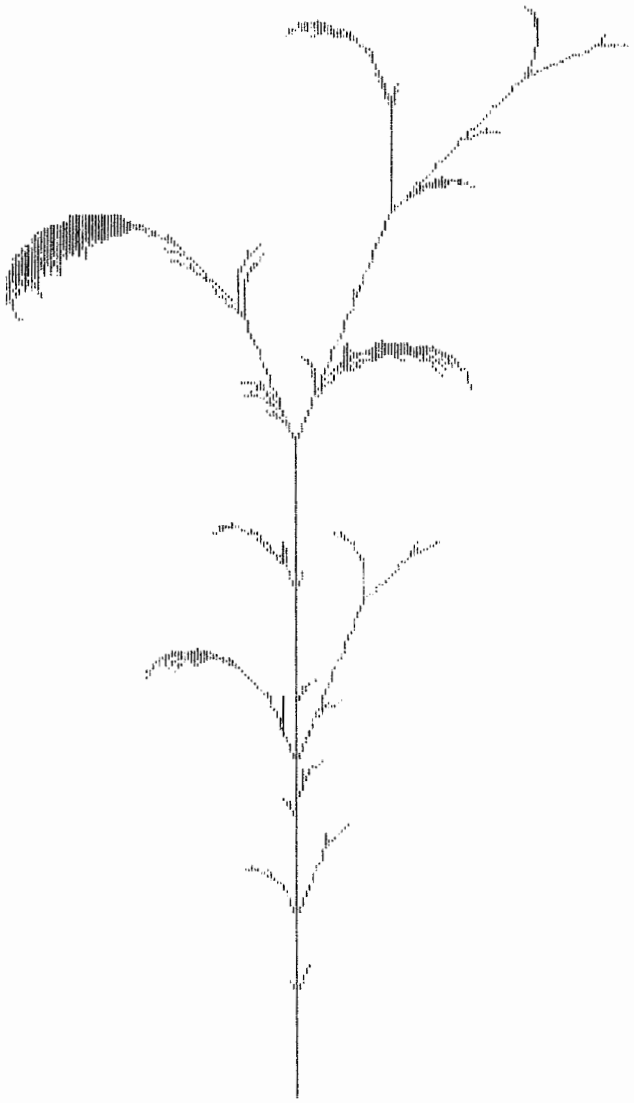


figure 4.4

4. Fonction de croissance

1°) Définition

On peut distinguer entre des productions qui spécifient la structure ramifiée et celles qui décrivent l'allongement des segments de la plante.

Cette distinction peut être observée dans le système L suivant :

nombre de dérivations : n = 7

variation de l'angle : $\delta = 20°$

axiome : X

productions : $X \rightarrow F [+X] F [-X] + X$

$F \rightarrow FF$

L'image résultante se trouve à la figure 4.5.

La première production décrit la structure ramifiée, tandis que la deuxième production $F \rightarrow FF$ décrit l'allongement de segments représenté par la suite du symbole F. Le nombre d'occurrences de la lettre F dans un chaîne χ_n provenant d'une lettre F est doublé dans chaque étape de dérivation en sorte que l'élongation est exponentielle, avec longueur (χ_n) = 2^n.

Une fonction qui décrit le nombre de symboles dans un mot en fonction du nombre de dérivations est appelée une *fonction de croissance* . Pour un système DOL, elle est indépendante de l'ordre des lettres dans les productions et mots dérivés. Ainsi, la relation entre le nombre d'occurrences de lettres dans des mots μ et γ, tel que $\mu \Rightarrow \gamma$, peut être exprimée à l'aide d'une matrice.

Soit G = < V, w, P >, un système DOL avec V = { a_1, a_2, ... a_m }. Construisons une matrice carrée Q_{m*m}, où q_{ij} est égal au nombre d'occurrences de la lettre a_j dans le successeur de la production dont le prédécesseur est a_i. Soit a^k_i, le nombre d'occurrences de la lettre a_i dans le mot χ généré par G en k dérivations.

On obtient l'équation suivante :

$$\left[a_1^k\ a_2^k\ \ldots\ a_m^k\right]\begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \vdots & & & \vdots \\ q_{m1} & q_{m2} & \cdots & q_{mm} \end{bmatrix} = \left[a_1^{k+1}\ a_2^{k+1}\ \ldots\ a_m^{k+1}\right]$$

Cette notation matricielle est utile dans l'analyse de la fonction de croissance.

Exemple :

Pour le système L suivant

axiome : a

productions : $a \rightarrow a\ b$

$b \rightarrow a$

la relation entre les nombres d'occurrences des lettres a et b dans deux mots consécutifs est donnée par :

$$\left[a^k\ b^k\right]\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \left[a^{k+1}\ b^{k+1}\right]$$

c'est-à-dire :

$$a^{k+1} = a^k + b^k = a^k + a^{k-1} \quad \text{pour } k = 1, 2, 3 \ldots$$

De l'axiome, il résulte $a^0 = 1$ et $b^0 = 0$. Ainsi, le nombre d'occurrences de la lettre a dans les chaînes générées par le système L croît conformément à la série de Fibonacci : 1, 1, 2, 3, 5, 8, ...

Il existe également des fonctions de croissances polynomiales et des combinaisons de fonctions polynomiales et exponentielles. Cependant, en pratique, il est difficile de trouver des systèmes L ayant des fonctions de croissance déterminées.

Dans la section suivante, on va présenter une extension des systèmes L qui évite ce problème en permettant l'inclusion explicite des fonctions de croissance dans les spécifications.

2°) Systèmes L paramétriques

Bien que les systèmes L vus jusqu'ici permettent de générer une variété d'objets intéressants, leur capacité de modélisation est limitée, puisque toutes les lignes tracées sont des multiples entiers d'un segment-unité.

Une approximation rationnelle de la longueur des lignes est une solution limitée, parce que le pas-unité doit être le plus petit dénominateur commun de toutes les longueurs de segments de la structure modelée. En conséquence, la représentation d'un simple modèle de plante, tel qu'un internoeud, peut nécessiter un grand nombre de symboles. Ceci est également valable pour les angles.

Un autre problème est la simulation du développement d'une structure végétale. Il existe des structures où on doit appliquer différentes productions en fonction des différentes étapes du développement.

Pour résoudre ces problèmes, Lindenmayer à proposé l'association de paramètres numériques aux symboles.

1. Systèmes OL paramétriques

Un *système L paramétrique* opère sur des *mots paramétriques* , lesquels constituent des chaînes de *modules* consistant en *lettres* avec des paramètres associés. Les lettres appartiennent à un *alphabet* V et les paramètres appartiennent à l'ensemble de nombres réels $\Re$. Un module avec lettre $A \in V$ et paramètres $a_1, a_2, \ldots a_n \in \Re$ est noté $A(a_1, a_2, \ldots a_n)$. Chaque module appartient à l'ensemble $M = V * \Re^*$, où $\Re^*$ est l'ensemble de toutes les séquences finies de paramètres. L'ensemble de toutes les chaînes de modules et l'ensemble de toutes les chaînes non-vides sont notés $M^* = (V \times \Re^*)^*$ et $M^+ = (V \times \Re^*)^+$, respectivement. Les paramètres actuels réels qui apparaissent dans les mots correspondent aux paramètres formels utilisés dans la spécification des productions du système L. Si Σ est l'ensemble des paramètres formels, alors $C(\Sigma)$ dénote une *expression logique* avec des paramètres de Σ et $E(\Sigma)$ une *expression arithmétique* . Les expressions consistent en des paramètres formels et des constantes numériques, combinés en utilisant des opérateurs arithmétiques +, -, *, /; l'opérateur d'exponentiation $\wedge$; les opérateurs relationnels <, >, =; les opérateurs logiques !, &, I (not, and, or) et les parenthèses.

Les ensembles des expressions logiques et arithmétiques avec des paramètres de Σ sont notés $C(\Sigma)$ et $E(\Sigma)$.

Définition : Un *système OL paramétrique* est défini comme un quadruplet $G = < V, \Sigma, w, P >$, où

- V est l'*alphabet* du système,

- Σ est l'ensemble des *paramètres formels* ,

- $w \in (V \times \Re^*)^+$ est un mot paramétrique non-vide appelé l'*axiome*,

- $P \subset (V \times \Sigma^*) \times C(\Sigma) \times (V \times E(\Sigma))^*$ est un *ensemble fini de productions* .

Les symboles : et $\rightarrow$ sont utilisés pour séparer les trois composants d'une production : le *prédécesseur* , la *condition* et le *successeur* .

Par exemple, une production avec prédécesseur A(t), condition t > 5 et successeur B(t+1) C D(t * 2, t - 2) est écrite comme

$$A(t) : t > 5 \rightarrow B(t + 1)\ C\ D(t * 2, t-2).$$

Un successeur d'une production remplace un module dans un mot paramétrique, si les conditions suivantes sont vérifiées :

- la lettre dans le module et la lettre du prédécesseur de la production sont identiques,

- le nombre de paramètres actuels dans le module est égal au nombre de paramètres formels dans le prédécesseur de la production, et

- la condition est *vraie* si les valeurs des paramètres actuels remplacent les paramètres formels dans la production.

Une production peut être appliquée à un module, créant une chaîne de modules spécifiée par le successeur de la production. Les valeurs des paramètres actuels remplacent les paramètres formels selon leur position.

Par exemple, la production donnée plus haut s'applique au module A(9), parce que la lettre A dans le module est identique à la lettre du prédécesseur de la production, il y a un paramètre actuel dans le module A(9) et un paramètre formel dans le prédécesseur A(t), et l'expression

logique t > 5 est vérifiée pour t = 9. Le résultat de l'application de cette production est un mot paramétrique B(10) C D(18,7).

Etant donné un mot paramétrique $\mu = A_1 A_2 \ldots A_m$, on dit que le mot $\gamma = \chi_1\chi_2\cdots\chi_m$ est directement déductible de μ et on écrit $\mu \Rightarrow \gamma$ si et seulement si $A_i \rightarrow \chi_i$; $\forall$ i= 1...m. Un mot paramétrique γ est *déductible* de l'axiome w, en n dérivations, s'il existe une séquence de mots $\mu_0, \mu_1, \mu_2, \ldots \mu_n$ tels que $\mu_0 = w$, $\mu_n = \gamma$ et $\mu_0 \Rightarrow \mu_1 \Rightarrow \mu_2 \ldots \Rightarrow \mu_n$.

Un exemple d'un système L paramétrique est donné par :

axiome : B(2) A(4,4)

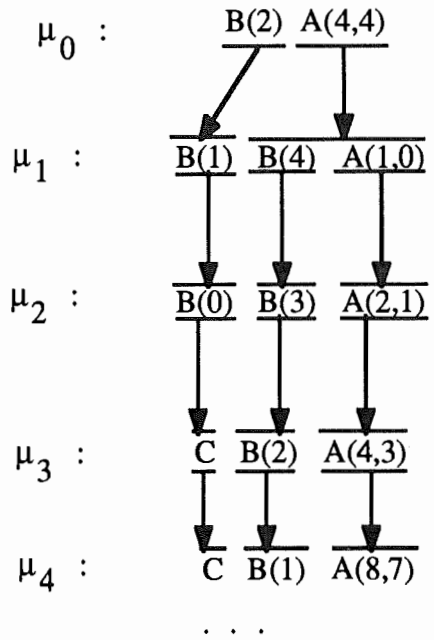
productions : A(x,y) : y <= 3 → A(x*2,x+y)

A(x,y) : y > 3 → B(x) A(x/y,0)

B(x) : x < 1 → C

B(x) : x >= 1 → B(x-1)

Comme pour les systèmes L non paramétriques, on suppose qu'un module est remplacé par lui-même si aucune production n'est trouvée dans l'ensemble P. Les mots obtenus dans les premiers étapes de dérivation sont montrés ci-dessous.



2. Systèmes 2L paramétriques

Les productions des systèmes OL paramétriques vues jusqu'ici ne dépendent pas de contexte, c'est-à-dire elles sont applicables sans tenir compte des modules voisins du module à remplacer. Une extension est nécessaire pour modéliser l'échange d'information entre des modules voisins. Si les productions incluent le contexte, chaque composant d'un prédécesseur (le contexte "gauche", le prédécesseur strict et le contexte "droite") est un mot paramétrique avec les lettres de l'alphabet V et des paramètres formels de l'ensemble Σ et qui peuvent tous apparaître dans la condition et le successeur.

Un exemple d'une production est donné par :

$A(x) < B(y) > C(z) : x + y + z > 10 \rightarrow E((x + y) / 2)\ F((y + z)/2)$

Elle peut être appliquée au module B(5) qui apparaît dans le mot paramétrique : ...A(4) B(5) C(6)..., parce que la séquence de lettres A, B, C dans la production et le mot paramétrique sont les mêmes, le nombre de paramètres formels et de paramètres actuels coïncident, et la condition 4+5+6>10 est vérifiée. Le module B(5) va être remplacé par E(4.5)F(5.5). Evidemment, les modules A(4) et C(6) sont remplacés par d'autres productions dans la même étape de dérivation.

3. Interprétation des mots paramétriques par la tortue

Si un ou plusieurs paramètres sont associés à un symbole interprété par la tortue, la valeur du premier paramètre contrôle l'état de la tortue. Si le symbole n'est pas suivi d'un paramètre, des valeurs par défaut spécifiées en-dehors du système L sont utilisées, tout comme dans le cas non-paramétrique. A titre d'exemple, on pourrait prendre comme ensemble de symboles de base affectés par des paramètres :

F(a) Avancer d'un pas a > 0. Le nouvel état de la tortue est (x', y', α) où $x' = x+a.\cos \alpha$ et $y' = y + a.\sin \alpha$. Un segment de droite est tracé entre les points (x, y) et (x', y').

f(a) Avancer d'un pas a sans tracer de segment de droite.

+(a) Tourner d'un angle a. Si a est positif, la tortue tourne à droite. Sinon pour a négatif, elle tourne à gauche.

On constate que le symbole + est utilisé comme une lettre de l'alphabet V et comme un opérateur dans les expressions arithmétiques. Sa signification dépend du contexte.

Exemple

Le système L suivant est un exemple d'un système L paramétrique qui engendre différentes feuilles selon la valeur attribuée aux variables D et R.

axiome : A(0)

productions : A(d) : d > 0 → A(d-1)

A(d) : d = 0 → F(1) [+A(D)] [-A(D)] F(1) A(0)

F(a) : * → F(a*R)

Pour D = 0, R = 2.00 et le nombre de dérivations n = 10, on obtient la feuille montrée à la figure 4.6. Le symbole *, utilisé comme condition dans la dernière production, signifie, qu'il n'y a pas de condition à vérifier pour appliquer cette production.

5. Algorithme

Après avoir introduit quelques-uns des concepts fondamentaux des systèmes L, revenons au but de ce mémoire, à savoir la représentation d' objets fractals à l'aide d'un ordinateur.

Tout algorithme qui permet de dessiner des objets, en se basant sur leurs propriétés fractales , passe obligatoirement par les étapes suivantes :

- lire les paramètres et les productions

- générer une chaîne de caractères

- interpréter une chaîne de caractères

Nous nous limiterons, ci-dessous, à des explicitations concernant les systèmes L non-paramétriques déterministes.

1°) Lecture des paramètres et des productions

Parmi les paramètres à introduire se trouve évidemment la variation de l'angle δ. Un paramètre intéressant est également le nombre de dérivations qui

permet de simuler le développement d'un objet. Pour certaines fractales, il est nécessaire d'avoir la possibilité d'indiquer des "symboles ignorés", c'est-à-dire considérés comme non-existants quand on vérifie le contexte. Normalement, ces symboles représentent des informations géométriques qui sont sans rapport d'un point de vue d'interaction entre les composants de la structure modelée. Il faut évidemment introduire l'axiome et les productions.

La grandeur du pas d'avancement d de la tortue influence seulement la grandeur de la figure. En plus, tous les segments sont des multiples entiers du segment-unité.

Dans le but de créer une "bibliothèque" d'objets fractals, une bonne méthode consiste à introduire les paramètres, l'axiome et les productions à l'aide d'un fichier.

2°) Génération d'une chaîne de caractères

La deuxième étape de l'algorithme est la "création" d'une chaîne de caractères en partant d'un axiome et en utilisant un ensemble de productions. Cette "génération" d'une chaîne, dont la longueur dépend du nombre de dérivations, doit se baser sur l'application parallèle des productions, c'est-à-dire le remplacement simultané de toutes les caractères d'une chaîne par leurs successeurs.

Si le système L peut contenir des accolades et peut avoir des productions dépendant du contexte, on doit tenir compte des facteurs suivants dans la recherche de la production applicable :

- la sous-chaîne considérée doit s'identifier à un prédécesseur strict d'une production et,

- en tenant compte des "symboles ignorés" et des accolades, ses sous-chaînes "voisines" doivent correspondre aux contextes de la production.

3°) Interprétation d'une chaîne de caractères

La chaîne de caractères créée est interprétée par la tortue. Selon les objets, elle doit obéir à certaines commandes.

Dans le cas des plantes, les commandes indispensables sont :

[Mettre l'état courant de la tortue sur une pile.

] Prendre un état de la tortue dans la pile. Cet état devient l'état courant.

F Avancer d'un pas de grandeur déterminée.

+ Tourner à droite d'un angle δ .

- Tourner à gauche d'un angle δ .

On commence avec l'initialisation de la position et de la direction de la tortue. L'étape suivante est la lecture des symboles de la chaîne, en commençant à gauche, et l'exécution des actions sous-jacentes.

6. Un exemple d'Implémentation

L'implémentation de l'algorithme, réalisée à l'occasion de ce mémoire, est basée sur le langage de programmation C. Pour l'implémentation, je me suis surtout basée sur le programme se trouvant à la fin du livre "Lindenmayer Systems, Fractals and Plants" de P. Prusinkiewicz et J. Hanan.

Le langage de programmation C permet une implémentation efficace de cet algorithme à cause de ses facilités de manipulation des caractères et de gestion de la place en mémoire. La facilité de manipulation des caractères est nécessaire à la recherche de la production à appliquer dans la génération de la chaîne. Elle facilite la vérification du contexte en tenant compte des "symboles ignorés" et des branchements. Ce langage permet en outre une programmation proche de la machine, ce qui facilite également la génération de la chaîne de caractères.

Le programme, que nous avons mis au point et qui est orienté vers l'utilisation d'un PC 386 SX, s'appelle "pfg". Il dessine à l'écran des images à partir des chaînes de caractères, définies par des systèmes L, qui peuvent avoir des productions dépendant du contexte et contenir des accolades. Il est fait de deux modules dont le premier, appelé "generate.C", contient la lecture du fichier de données et la génération d'une chaîne de caractères, tandis que le deuxième, appelé "interpret.C", s'occupe de l'interprétation d'une chaîne de caractères. Ce

programme utilise également deux fichiers, appelés "generate.h" et "interpret.h", qui contiennent la déclaration des constantes et la définition des types de données.

On peut demander l'exécution du programme "pfg" en lui passant directement le nom du fichier. Ceci se fait par la commande "pfg nom_du_fichier".

1°) Introduction des données

Les données, c'est-à-dire les paramètres, l'axiome et les productions, sont introduites par un fichier. Les paramètres sont la longueur de dérivation, le "facteur d'angle", le "facteur d'échelle" et les "symboles ignorés". Le "facteur d'angle" détermine la variation d'angle δ, qui est donnée par : $\delta = 360° / (\text{facteur d'angle})$. Puisque ce facteur a été spécifié comme une variable entière, toutes les variations d'angles ne sont pas possibles. Il permet de calculer une et une seule fois les valeurs de $d * \sin\alpha$ et $d * \cos\alpha$ où d est le pas de la tortue et où α a comme valeur tous les multiples de δ . Le "facteur d'échelle" détermine la grandeur de l'image résultante. En principe, 100 dénote une image sur tout l'écran et 0 dénote une image réduite à un pixel. Cependant, l'image est toujours dessinée de telle façon que la longueur d'un segment soit égale à un nombre entier de pixels.

Un exemple de fichier est donné ci-dessous :

```
derivation length : 25
angle factor : 16
scale factor : 100
axiom : F1F1F1
ignore : + - F
0 < 0 > 0 --> 1
0 < 0 > 1 --> 1[-F1F1]
0 < 1 > 0 --> 1
0 < 1 > 1 --> 1
1 < 0 > 0 --> 0
1 < 0 > 1 --> 1F1
1 < 1 > 0 --> 1
1 < 1 > 1 --> F0
* < + > * --> -
* < - > * --> +
```

La figure résultante se trouve à la figure 4.4.

Les séparateurs < et > doivent se trouver dans chaque production. Les sous-chaînes vides sont dénotées par le symbole *. Tous les composants d'une production, y compris le prédécesseur strict, peuvent contenir plusieurs lettres.

Par exemple, AB < CDE > FG --> A [B] A est une production valide qui remplace la sous-chaîne CDE par A [B] A si le contexte "gauche" et le contexte "droite" du prédécesseur sont vérifiés.

La longueur d'un contexte peut varier d'une production à l'autre à l'intérieur d'un ensemble de productions. Les accolades peuvent seulement se trouver dans les successeurs.

2°) Fichiers Include

Outre les fichiers "header" que le système C met à la disposition de l'utilisateur, le programme inclut deux fichiers.

1. "generate.h"

Les différentes constantes déclarées dans ce fichier sont MAXPROD, le nombre maximum de productions, MAXAXIOM, la longueur maximum de l'axiome, MAXSTR, la longueur maximum de la chaîne de caractères, MAXCHARS, le nombre de caractères ASCII, et MAXIGNORE, le nombre maximum de "symboles ignorés". Elles sont indispensables pour allouer la place en mémoire pour les différentes variables utilisées.

Deux types de données sont également définis dans cette partie. Le premier, appelé "Parameter", permet de déclarer des variables contenant tous les paramètres d'un fichier, c'est-à-dire son nom, la longueur de dérivation, le "facteur d'angle" et le "facteur d'échelle".

Le deuxième type de données, appelé "Production", contient les longueurs et les pointeurs vers les premiers caractères : du contexte "gauche", du prédécesseur strict, du contexte "droite" et du successeur de la production.

2. "interpret.h"

Ce fichier contient la déclaration des constantes et la définition des types de données qui sont en rapport avec la représentation graphique de la chaîne.

Les différents constantes déclarées sont MAXANGLE, la valeur maximum du "facteur d'angle", MAXSCALE, la valeur maximum du "facteur d'échelle", TWO_PI, dont la valeur est égale à 6.2831853, STACK_SIZE, la profondeur

maximum des branches, et LEFT, TOP, RIGHT, BOTTOM, les valeurs qui définissent une fenêtre à l'écran.

Trois nouveaux types de données sont également définis dans ce fichier. Le premier, appelé TURTLE, simule la notion de tortue. Le deuxième type de données, appelé PIXEL, permet de déclarer des variables représentant des points de l'écran. Le troisième type prédéfini, appelé BOX, permet de déclarer une fenêtre.

3°) Module "generate.C"

Le module "generate.C" contient la fonction principale, qui est faite des fonctions "input","Derive" et "interprete" qui coïncident avec les 3 parties de l'algorithme vues plus haut.

Le but de la fonction "input" est de lire les paramètres, l'axiome et les productions dans le fichier, et de les stocker dans des variables appropriées.

La fonction "Derive" crée une chaîne de caractères en partant d'un axiome et en utilisant les productions. La longueur de cette chaîne dépend du nombre de dérivations. L'application parallèle des productions d'un système L est réalisée par le processus suivant : une première chaîne de caractères, contenant au début l'axiome, est parcourue de gauche à droite. Ses sous-chaînes consécutives sont comparées avec les prédécesseurs des productions. Les successeurs appropriés sont ajoutés à la fin d'une seconde chaîne. Après avoir examiné la première chaîne, on copie la deuxième chaîne dans la première chaîne. Ce processus est répété jusqu'à ce que le nombre de dérivations désiré soit atteint.

En fait, pour trouver la production à appliquer, la fonction vérifie, d'une part, que la sous-chaîne courante de la première chaîne correspond à un prédécesseur strict d'une production, èt, d'autre part, que le contexte "gauche" et le contexte "droite" sont vérifiés en tenant compte des "symboles ignorés" et du fait que la chaîne peut contenir des accolades. Dans la vérification du contexte, la fonction concernée cherche le contexte en sautant des branches qui peuvent même contenir des sous-branches. L'ensemble de productions est parcouru dans le même ordre que celui des productions énumérées dans le fichier.

Si on n'arrive pas à allouer de la place en mémoire pour les 2 chaînes ou si la seconde chaîne dépasse sa place allouée, un message d'erreur apparaît et le programme se termine.

La fonction "interprete" se trouve dans le module "interpret.C".

4°) Module "interpret.C"

La fonction "interprete" crée l'environnement graphique et dessine la figure.

En fait, la chaîne de caractères est "interprétée" deux fois. La première fois, l'image résultante n'est pas dessinée, mais permet de trouver la grandeur de la figure. Pour cette première lecture, la position initiale de la tortue est égale à (0, 0), son pas d'avancement égal à 1 et son orientation vers le haut. La grandeur de la figure permet de déterminer la position initiale de la tortue et son pas d'avancement d, en fonction du "facteur d'échelle", pour que la figure soit comprise dans la fenêtre définie par les constantes LEFT, TOP, RIGHT et BOTTOM se trouvant dans le fichier "interpret.h".

La fonction, qui dessine la figure définie par la chaîne de caractères, possède un paramètre, appelé "flag", dont la valeur détermine si on dessine la figure (flag = 1) ou si on détermine seulement sa grandeur (flag = 0).

Les actions effectuées sont associées aux symboles suivants :

+ Tourner à droite d'un angle déterminé par le "facteur d'angle".

- Tourner à gauche d'un angle déterminé par le "facteur d'angle".

| Tourner 180° autour de l'axe y.

[Mettre l'état courant sur une pile.

] Prendre un état dans la pile, qui devient l'état courant.

F Avancer et tirer une ligne.

f Avancer sans tirer de ligne.

A cette liste , j'ai ajouté deux symboles :

{ Commencer un polygone.

} Terminer un polygone.

La figure, un polygone, définie par la sous-chaîne se trouvant entre ces deux symboles, est remplie. Dans cette sous-chaîne, il faut utiliser le symbole f pour avancer et "enregistrer" le point final comme sommet du polygone.

Ce programme permet, comme montre l'exemple suivant, de dessiner des polygones ressemblant à des feuilles végétales.

Exemple

nombre de dérivations : n = 1

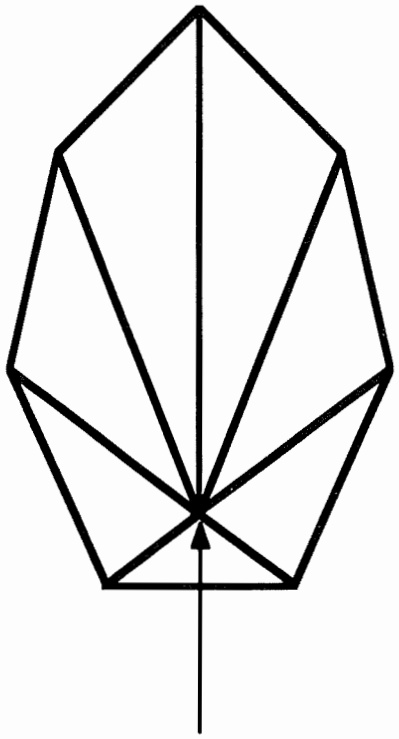
variation de l'angle : $\delta = 20°$

axiome : L

production : L → {[++++f] [++Ff] [+FFf] [FFFFf] [-FFf] [--Ff] [----f] }

L'image résultante se trouve à la figure 4.7.

La figure suivante montre les sommets du polygone et la manière de les obtenir :



point d'origine

Pour définir les différents sommets, dont le nombre est déterminé par le nombre d'accolades, la tortue part toujours du point-milieu et effectue les commandes se trouvant entre les accolades. La dernière commande a pour but, dans ce cas, de toujours avancer d'un pas et "d'enregistrer" le point final comme sommet du polygone.

En augmentant ou diminuant le nombre de sommets, la variation de l'angle δ ou le nombre de commandes entre les accolades, on peut obtenir différents polygones.

Exemple

nombre de dérivations : n = 1

variation de l'angle : $\delta = 30°$

axiome : L

production : L $\rightarrow$ {[+++f] [++Ff] [+FFf] [FFFf] [-FFf] [--Ff] [---f] }

L'image résultante se trouve à la figure 4.8.

Dans ces deux exemples, la variation du nombre de dérivations n'a pas d'influence sur la figure résultante.

D'autres figures faites avec ce programme se trouvent aux pages suivantes: la figure 4. 9 montre une courbe classique, appelée courbe d'Hilbert. Les figures 4.10 et 4.11 sont faites à l'aide d'un système OL arborescent.

Les listings de ce programme, ainsi que les listings de tous les fichiers, se trouvent en annexe.

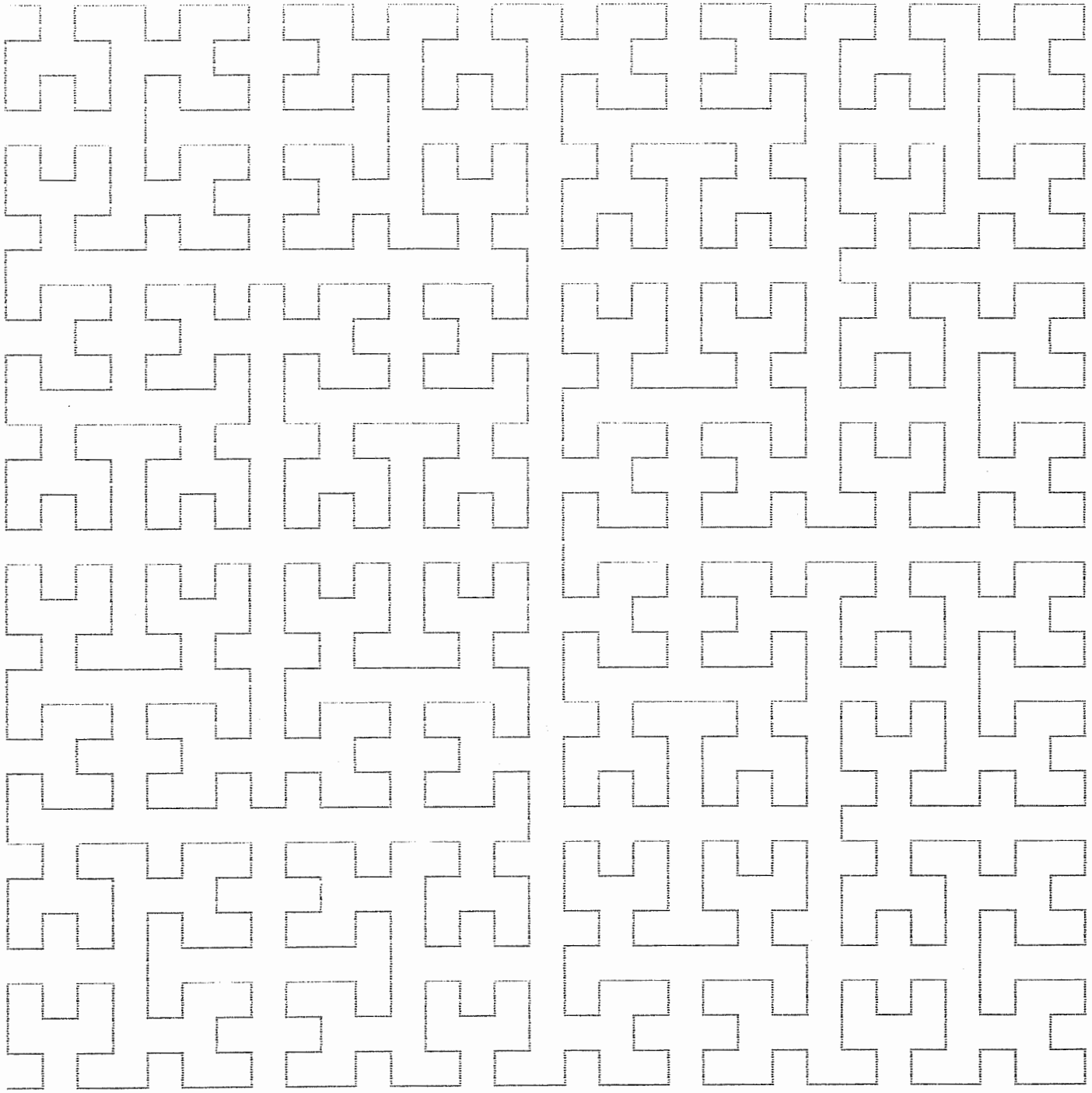


figure 4.9

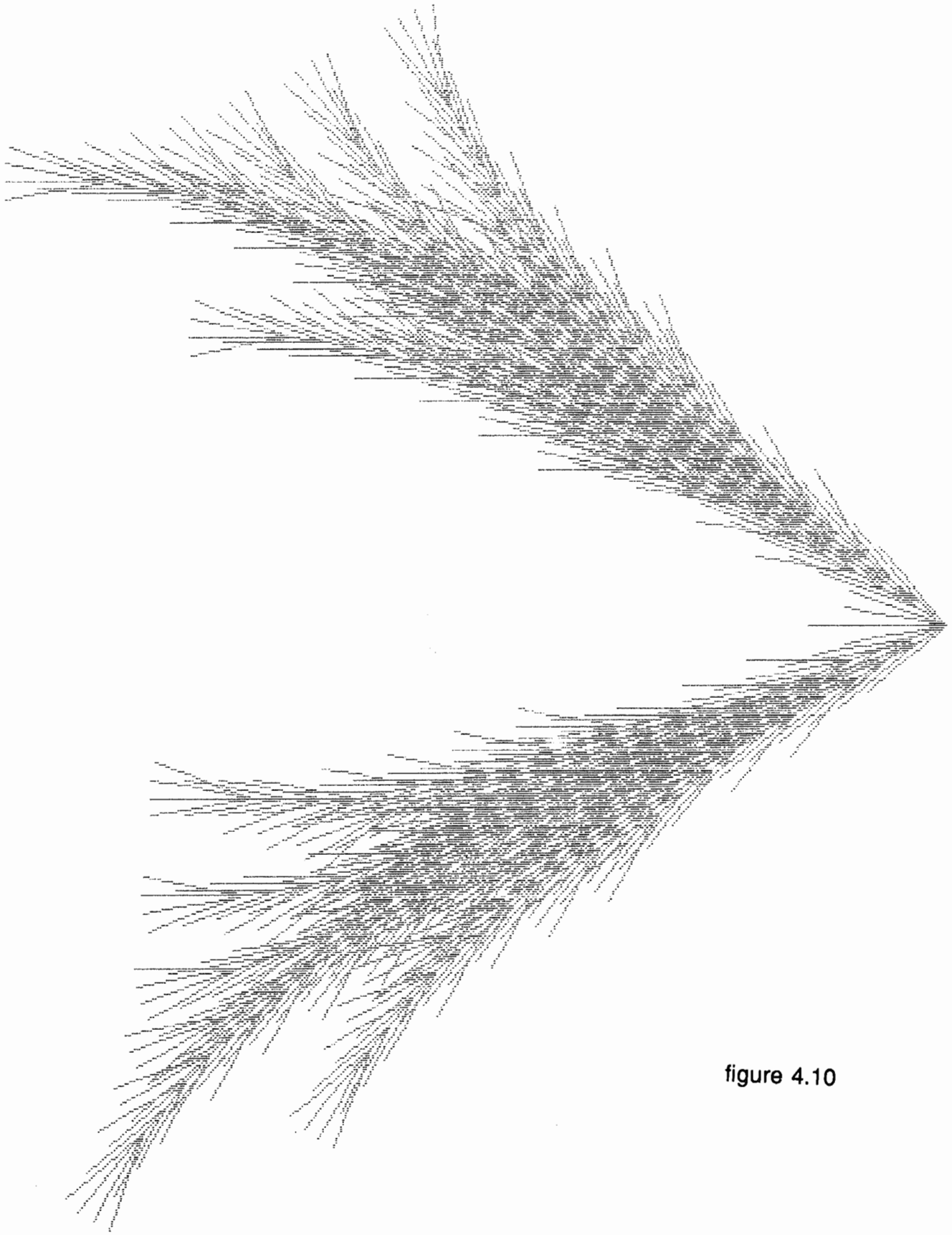


figure 4.10

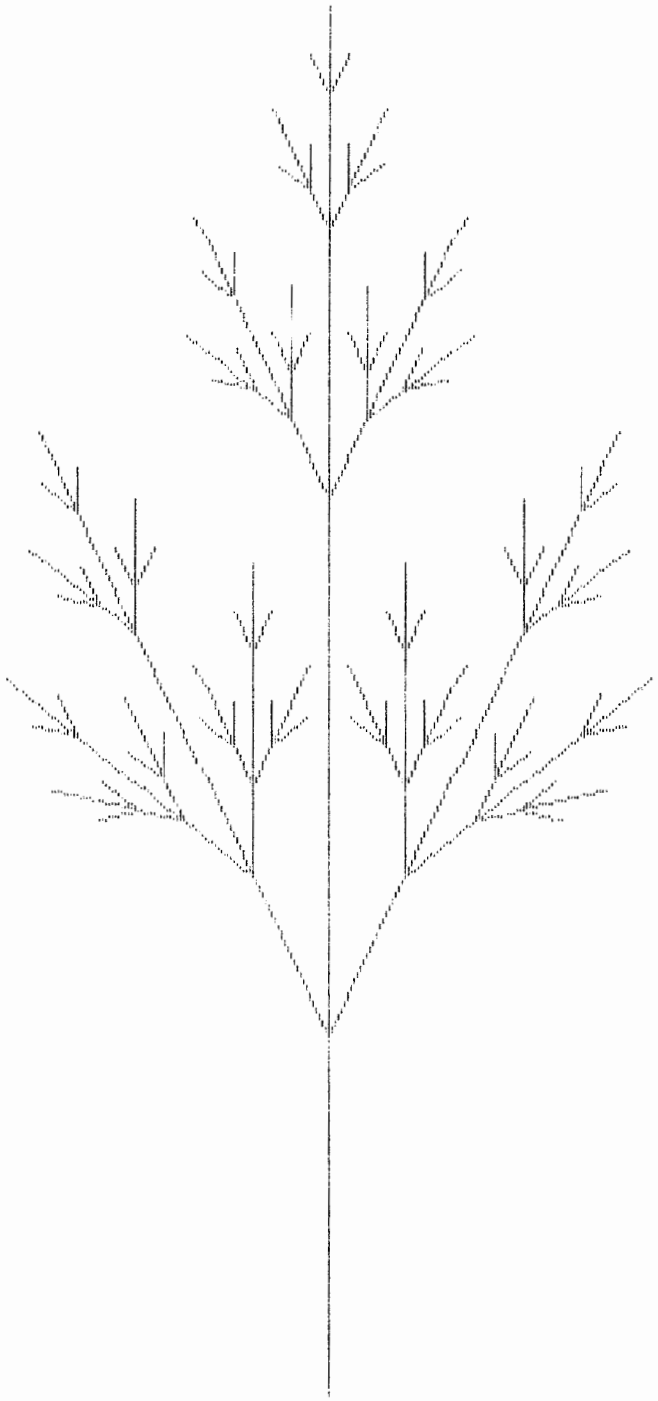


figure 4.11

5ième Chapitre : Analyse des deux approches étudiées

Après avoir introduit les systèmes de Lindenmayer et l'algorithme "IFS" dans les chapitres précédents, il convient maintenant de les analyser et de les comparer. L'analyse proposée ici va surtout considérer des plantes, des objets dont la caractéristique principale est la structure ramifiée. Les systèmes de Lindenmayer et l'algorithme "IFS" permettent de dessiner certaines structures fractales. Rappelons [réf. 17] que, selon une définition intuitive proposée par B. Mandelbrot, une structure fractale est une figure géométrique ou un objet naturel qui combine les caractéristiques suivantes :

- ses parties ont la même forme ou structure que le tout, à ceci près qu'elles sont à une échelle différente et peuvent être légèrement déformées;

- sa forme est, soit extrêmement irrégulière, soit extrêmement interrompue ou fragmentée, quelle que soit l'échelle d'examen;

- elle contient des "éléments distinctifs" dont les échelles sont très variées et couvrent une très large gamme.

Les deux types d'objets, fractales et plantes, ne sont en effet pas sans rapports. Ainsi, l'architecture d'un lilas, montrée à la figure 5. 1, est dans une certaine mesure fractale, puisque la structure arborescente visible sur la tige principale se répète sur les branches latérales. Bien sûr, aucune plante n'est totalement fractale, et certaines ne le sont pas du tout.

Un objet fractal est, selon une définition plus précise due à B. Mandelbrot, un objet dont la dimension de Hausdorff-Besicovitch est strictement plus grande que la dimension topologique. D'un point de vue strict, aucune figure montrée dans ce mémoire n'est fractale, puisque les figures consistent en un nombre fini de lignes et leur dimension de Hausdorff-Besicovitch est alors égale à la dimension topologique. Les figures sont à considérer comme des approximations des objets fractals.

Nous comparons ci-dessous les deux approches étudiées dans ce mémoire sous le quadruple point de vue de leurs domaines d'applications, de leurs possibilités de caractérisation, de représentation et de calcul de la dimension d'un

objet fractal. Nous terminons enfin en énonçant un certain nombre d'avantages et d'inconvénients des programmes utilisés et par mis au point dans le cadre du mémoire.

1. Domaines d'application

1°) Algorithme "IFS"

L'algorithme "IFS", découvert par M. Barnsley, permet de dessiner des objets naturels divers comme les nuages, la fumée, les feuilles, les fougères, mais également des objets artificiels simples, comme par exemple un rectangle, une paire de droites, etc.. Les images obtenues sont faites de points et leur qualité dépend du nombre de points affichés, c'est-à-dire du temps écoulé.

L'algorithme "IFS" permet également de dessiner le fameux *ensemble de Mandelbrot* et les ensembles de Julia. Ils se trouvent dans le plan complexe et sont obtenus en étudiant l'expression $z^2 + c$, où z est un nombre complexe variable et c un nombre complexe fixe. L'ensemble de Mandelbrot est constitué par tous les nombres complexes c pour lesquels $z^2 + c$ reste à distance finie dans le plan complexe, quel que soit le nombre d'itérations, même si ce nombre est infini. Si on adopte la règle opposée, qui consiste à fixer la valeur de c et à faire varier la valeur initiale de z, on définit un ensemble différent que l'on appelle un *ensemble de Julia*.

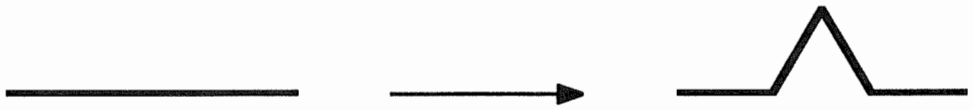
2°) Systèmes L

Les systèmes de Lindenmayer permettent la représentation graphique des plantes. Il ont cependant une portée limitée : leur formalisme est mieux adapté à la description et à la simulation de la croissance de plantes herbacées, plutôt qu'à l'étude des arbres. Cela tient notamment au fait que les facteurs génétiques jouent un rôle prédominant dans le développement des plantes herbacées, alors que celui des arbres est nettement plus conditionné par des facteurs environnementaux (lumière, vent, etc.). Or les systèmes L tiennent compte des signaux éventuels (hormonaux et autres) circulant au sein d'une plante, mais se prêtent difficilement à la simulation de certaines influences extérieures.

Les systèmes L peuvent également décrire et représenter des courbes classiques, comme par exemple la courbe appelée "flocon de neige" de Von Koch, dont la production et les paramètres se trouvent en annexe 2. A chaque étape de

dérivation, tous les segments de droite sont remplacés en utilisant la production représentée graphiquement ci-dessous.

Représentation graphique



La figure résultante est montrée à la figure 5.2.

Les figures obtenues sont faites de segments de droite. Suivant le nombre de symboles interprétables par la tortue, l'image peut être complexe. L'utilisateur des systèmes L peut toujours ajouter des nouveaux symboles et leur donner une interprétation.

Les deux méthodes permettent d'obtenir des figures complexes, à l'aide de quelques transformations affines ou de productions. Leurs espaces de stockage sont petits, ce qui permet de constituer une bibliothèque d'objets fractals.

2. Possibilités de caractérisation d'un objet fractal

Dans la méthode "IFS", une fractale est caractérisée par un ensemble de transformations affines contractives. Tandis qu'un système L déterministe décrit un objet fractal par un axiome, un "facteur d'angle", des" symboles ignorés" et un ensemble de productions.

Les deux méthodes utilisent une description récursive et courte d'un objet fractal pour le dessiner : quelques transformations affines, ou quelques productions, sont seulement nécessaires pour obtenir des figures complexes.

1°) Algorithme "IFS"

Le code IFS peut être vu comme une entité dynamique qui peut être modifiée pour donner lieu à des images apparentées.

Supposons que $\{ W_n, p_n : n = 1,2 \ldots N\}$ est un code IFS avec l'attracteur associé A et que t est une transformation inversible de R^2 vers R^2. Par exemple, t peut être une translation, une rotation, etc. Le code IFS pour A sous t est simplement $\{ t \circ W_n \circ t^{-1}, p_n : n = 1,2 \ldots N\}$. En particulier, ceci signifie que si un code IFS d'un objet est disponible dans une bibliothèque de codes, des images du

même objet d'un point de vue quelconque, ou n'importe quelle copie déformée de cet objet, peuvent être facilement encodées, décodées et visualisées.

L'algorithme "IFS" ne permet pas de simuler ni le développement ni la croissance d'une plante.

2°) Systèmes L

En variant le nombre de dérivations, les systèmes L permettent de reproduire de façon biologiquement réaliste le processus de croissance des plantes.

Dans le cas d'une courbe classique, la variation du nombre de dérivations montre les différentes étapes dans la génération d'une figure.

Les systèmes L stochastiques permettent d'obtenir différents exemplaires d'une même espèce en introduisant des variations qui conservant les aspects généraux d'une plante mais qui changent ses détails.

Toutes les figures obtenues dans la simulation de la croissance d'une plante sont discrètes dans le temps. Les systèmes L ne permettent pas de montrer de façon continue la croissance d'une plante.

Les influences extérieures, comme par exemple les effets de la lumière ou de la gravité, dans la croissance d'une plante peuvent être simuler en tournant légèrement la tortue dans la direction du vecteur de tropisme $\vec{T}$ prédéfini, après avoir tracé chaque segment de droite. A deux dimensions, l'ajustage de l'orientation α est déterminée par la formule

$$\alpha = e\,\vec{F} \times \vec{T}$$

où e est un paramètre qui capte la susceptibilité de courbure d'une axe. Cette formule heuristique a une motivation physique : si $\vec{T}$ est interprété comme une force appliquée au point final du segment $\vec{F}$ et $\vec{F}$ peut tourner autour de son point débutant, le moment de torsion est égal à $\vec{F} \times \vec{T}$.

3. Représentation d'un objet fractal

1°) Algorithme "IFS"

La détermination d'un ensemble de transformations affines peut se faire par essai-erreur. Comme mentionné au chapitre 3, il existe des logiciels qui facilitent cette tâche difficile, en proposant des fonctions qui permettent de changer les coefficients d'une transformation affine.

Une autre possibilité de détermination des codes IFS consiste à partir d'une image numérisée T. Une première transformation affine $W_1(x, y) = (ax + by + e, cx + dy + f)$, dont les coefficients sont initialisés à $a = d = 0.25$ et $b = c = e = f = 0$, est appliquée à l'image T se trouvant à l'écran. Le résultat $W_1(T)$ est une copie de T, de dimension réduite d'un quart, et est affichée à l'écran dans une couleur différente de T. L'utilisateur peut maintenant adapter les coefficients pour que $W_1(T)$ se trouve sur une partie de T. Il est important que $W_1(T)$ soit plus petite que T, pour garantir que W_1 soit une contraction. Une fois $W_1(T)$ bien positionnée, on introduit une nouvelle transformation affine W_2, qui est également appliquée à T. W_2 est interactivement ajustée jusqu'à ce que $W_2(T)$ recouvre un sous-ensemble de T qui n'est pas en $W_1(T)$.

Ceci permet de déterminer un ensemble de transformations affines contractives $\{ W_1, W_2 \ldots W_N\}$ qui vérifient la propriété

$$\tilde{T} = \bigcup_{n=1}^{N} W_n(T)$$

où T est l'image originale. $W_n(T)$ est appelé le $n^{ième}$ carreau du collage. L'indicateur mathématique de la proximité de T et $\tilde{T}$ est la distance de Hausdorff $h(T, \tilde{T})$, définie plus loin. L'expression "visuellement proche", employée ci-dessous, veut dire "$h(T, \tilde{T})$ est petite".

Le théorème du Collage, énoncé plus bas, assure qu'un attracteur A de n'importe quel code IFS $\{ W_n, p_n : n = 1,2 \ldots N\}$, qui utilise cet ensemble de transformations affines, va être "visuellement proche" de T. En outre, si $T = \tilde{T}$, c'est-à-dire $h(T, \tilde{T}) = 0$, alors $A = T$. La méthode fournit des codes IFS tels que la géométrie du modèle sous-tendu ressemble à celle de la cible.

Les énoncés précis qui déterminent la méthode sont les suivants :

La distance de Hausdorff h(A, B) entre deux sous-ensembles fermés et bornés A et B de R^2 est définie par :

$$h(A, B) = \max \left\{ \max_{\vec{x} \in A} \min_{\vec{y} \in B} \left|\left| \vec{x} - \vec{y} \right|\right| ; \max_{\vec{y} \in B} \min_{\vec{x} \in A} \left|\left| \vec{y} - \vec{x} \right|\right| \right\}$$

Le Théorème du Collage traite la distance de Hausdorff entre l'attracteur A d'un "système de fonctions itérées" et l'image originale T. Il peut s'énoncer comme suit :

Théorème du Collage : Soient $\{ W_n, p_n : n = 1,2 \ldots N\}$ un code IFS, $s < 1$ la plus grande constante de Lipshitz des applications contractives du code, et $\varepsilon > 0$ n'importe quel nombre positif. Soit T un sous-ensemble fermé et borné de R^2. Si les W_n ont été choisies de telle façon que

$$h\left(T, \bigcup_{n=1}^{N} W_n(T)\right) < \varepsilon$$

alors

$$h(T, A) < \frac{\varepsilon}{1 - s}$$

où A dénote l'attracteur de l'IFS.

Un code IFS ne permet pas de spécifier explicitement la géométrie d'une figure. Elle est déterminée implicitement par les coefficients des transformations affines.

2°) Systèmes L

Pour modéliser une forme particulière, les systèmes de Lindenmayer essaient de capturer l'essentiel du processus de développement qui mène à cette forme. Pour déterminer les productions, il faut, selon Prusinkiewicz, procéder par étapes en observant la plante. Tout d'abord, il faut déterminer le type de développement de la plante considérée : a-t-elle une tige principale ou seulement des ramifications multiples ? Quelle partie se développe en premier et d'où sortent les premiers fleurs ?, etc. Ensuite, on mesure les angles entre les différentes ramifications. Puis il faut déterminer les proportions de la plante. Enfin, on observe

les caractéristiques des organes de la plante, c'est-à-dire principalement celles de ses feuilles et fleurs. Une aide pour trouver les différentes productions, exprimées en caractères, pourrait être la représentation graphique des productions nécessaires, qui sont ensuite "traduites" en une chaîne de caractères.

Depuis 1987, Lindenmayer cherche à automatiser la détermination des productions à partir, par exemple, d'une image numérisée de la plante. Prusinkiewicz estime que d'ici deux ans le problème sera résolu pour certaines structures simples.

Puisqu'on peut toujours ajouter des symboles dans l'alphabet d'un système L et leur donner une interprétation graphique, on peut symboliser des fleurs ou feuilles à l'aide d'un caractère. Ils sont dessinés quand le caractère prédéfini apparaît dans la chaîne de caractères à interpréter.

4. Dimension fractale

Puisqu'une plante n'est pas strictement scalante, c'est à-dire semblable à toutes les échelles, sa dimension fractale est difficile à mesurer. Ceci est valable pour les figures obtenues avec les deux programmes.

Sauf dans le cas des courbes classiques, qui sont scalantes, la détermination de la dimension fractale est possible à l'aide de la formule

$$D = \frac{\log N}{\log \frac{1}{r}}$$

où r est le facteur de réduction et N le nombre de copies plus petites de l'objet.

Des figures, dont la dimension fractale a été déterminée, se trouvent au chapitre 1.

5. Avantages et inconvénients des programmes utilisés

1°) Systèmes L

Le programme, appelé "pfg" et composé des deux modules "generate.C" et "interpret.C", dessine des images à partir des chaînes de caractères, définies par des systèmes L, qui peuvent avoir des productions dépendant du contexte et contenir des accolades.

On pourrait imaginer un environnement software, contenant entre autre ces deux modules, qui construit et étudie des modèles mathématiques de plantes. Cet environnement pourrait fournir des résultats importants sur le développement et la croissance d'une plante. Il pourrait être vu comme un petit monde, un laboratoire virtuel, indépendant du temps réel, c'est-à-dire permettant d'accélérer ou de ralentir la croissance d'une plante.

L'inconvénient majeur du programme "pfg" est qu'il nécessite beaucoup de place en mémoire, surtout pour la génération de la chaîne de caractères. A cause de cette "consommation de mémoire", on ne peut pas augmenter indéfiniment le nombre de dérivations. Un remède serait d'utiliser les systèmes L paramétriques qui, à l'aide des conditions et paramètres, permettraient d'obtenir des figures plus compliquées. D'autre part, l'implémentation de l'algorithme se complique, parce qu'il faut vérifier les conditions et effectuer le calcul des paramètres actuels.

Un autre inconvénient du programme est son incapacité de montrer simultanément l'image, les paramètres et les productions, de permettre la modification directe d'une production et d'ajuster l'image.

Toutes les plantes faites avec ce programme ont des tiges qui n'ont pas de largeur. Il est cependant possible de remédier à cet inconvénient, puisqu'on peut toujours ajouter des symboles dans l'alphabet d'un système L et leur donner une interprétation graphique.

2°) Algorithme "IFS"

Un grand avantage de ce programme, mis au point par M. Barnsley, est la possibilité d'agir directement sur les coefficients d'une transformation affine et de montrer l'image résultante. Toutes les modifications sont permises, sauf celles qui ont pour effet de faire sortir les points résultants du contour de la "fenêtre attracteur".

Pour déterminer les coefficients d'une transformation affine, on peut seulement procéder soit par essai-erreur soit en partant des codes IFS existants. La détermination des coefficients à partir d'une image numérisée n'est pas encore possible.

Conclusion

Dans ce travail, nous avons étudié deux algorithmes permettant de dessiner une fractale et donc une plante, dont la structure ramifiée possède des caractéristiques fractales. Bien qu'il existe d'autres méthodes pour simuler la croissance et le développement d'une plante, les deux méthodes présentées dans ce mémoire ont comme avantage d'utiliser une définition courte et récursive d'une plante pour la dessiner : les systèmes de Lindenmayer se basent sur un ensemble de productions, tandis que l'algorithme "IFS" utilise une série de transformations affines.

Les algorithmes permettent tous deux une approche orientée objet, thème à la mode, qui rend les logiciels plus fiables, plus souples et plus évolutifs.

Les systèmes de Lindenmayer, en essayant de capter l'essentiel du processus de développement d'une plante, permettent une représentation réaliste de la plante et facilitent l'étude de sa croissance car ils permettent de simuler son développement grâce au paramètre "nombre de dérivations".

L'avantage de l'algorithme "IFS" est la possibilité de dessiner n'importe quel objet naturel et même des objets artificiels.

Dans la détermination d'un ensemble de productions ou d'une série de transformations affines, il reste encore beaucoup à faire du point de vue de la programmation : aucun des deux programmes utilisés n'aident concrètement à leur détermination.

Les plantes présentées dans ce mémoire sont sans rapport avec la réalité. Ce sont des plantes imaginaires. Pour obtenir une représentation réaliste, il conviendrait de faire une étude détaillée des plantes existantes, ce qui nécessiterait donc une approche pluridisciplinaire : botanique, agronomie, mathématiques et informatique.

Les deux algorithmes facilitent la représentation graphique des objets naturels à l'aide d'une définition courte et on pourrait imaginer des logiciels comportant des fonctions, qui permettent une définition facile et simple d'un objet et son inclusion dans des images complexes.

Références :

1. R. Apéry, M. Caveing, J.-P. Desclés, J. Dieudonné, R. Fraissé, F. de Gandt, P. Gochet, J.-M. Lévy-Leblond, M. Loi, B. Mandelbrot, J.-C. Pont, R.Thom
Penser les mathématiques
Coll. Sciences, Le Seuil,1982

2. B. Mandelbrot Extrait de l'article paru dans "La Science & la Vie", juillet 1983

3. H.-O. Peitgen, D. Saupe The Science of Fractal Images
Springer-Verlag

4. Ph. de Reffye, C. Edelin, M. Jaeger La modélisation de la croissance des plantes
La recherche n° 207 (février 89)

5. P. Prusinkiewicz, A. Lindenmayer The Algorithmic Beauty of Plants
Springer-Verlag

6. P. Prusinkiewicz, J. Hanan Lindenmayer Systems, Fractals and Plants.
Springer-Verlag. New York 89

7. B. Mandelbrot Les objets fractales
La Recherche n° 85 (Janvier 78)

8. Le monde végétal Science et Vie (hors série) (mars 78)

9. B. Mandelbrot The Fractal Geometry of Nature (1983)
Freeman San Franscisco

10. Roger Caratini Les plantes
Bordas

11. Michael F. Barnsley The Desktop fractal Design Handbook
Academic Press

12. H. Abelson, A.A. diSessa. Turtle geometry
M.I.T.Press, Cambridge, 1982

13. A.R. Smith. Plants, Fractals, and formal languages.
Computer Graphics Volume 18, Number 3, July 1984

14. P. Prusinkiewicz, A. Lindenmayer, J. Hanan.
Developmental Models of Herbaceous Plants for Computer Imagery Purposes`
Computer Graphics Volume 22, Number 4, August 1988

15. A. Gerschenfeld L'ordinateur cultive son jardin
La Recherche n° 225 (Octobre 90)

16. A. Dewdney Reconstitution d'images
Pour la Science n°153 (Juillet 90)

17. B. Mandelbrot Les objets fractals
Troisième édition, suivie de "Survol du langage fractal".
Flammarion, Paris, 1989

18. M. Barnsley Fractals Everywhere
Academic Press, Boston, 1988

19. H.-O. Peitgen, P. H. Richter The Beauty of Fractals
Springer Verlag Berlin Heidelberg 1986

ANNEXES

Annexe 1

generate.h

```
#define MAXPROD         50  /*maximum number of productions          '
#define MAXAXIOM       100  /*maximum length of axiom                *
#define MAXSTR       15000  /*maximum length of the derived string   */
#define MAXCHARS       256  /*the number of ASCII characters         */
#define MAXIGNORE       50  /*maximum number of ignored symbols      */
/*************************************************************************
/* structure"Parameter" collects various input parameters             */
/*************************************************************************
struct Parameter {
     char *filename;        /*input file name                         */
     int n;                 /*derivation length                       */
     int angle;             /*the angle factor                        */
     int scale;             /*the scaling factor                      */
};
typedef struct Parameter Parameter;
/*************************************************************************
/* "Production" is a structure which contains pointers to the first*/
/* characters and the lengths of : the left context, the strict    */
/* predecessor, the right context and the production successor.    */
/* The actual strings are stored linearly in the array inpStr,     */
/* separated by the '\0' character.                                */
/*************************************************************************/
struct Production {
     char *lCon;            /*the left context                        */
     int lConLen;           /*the length of the left context          */
     char *pred;            /*the strict predecessor                  */
     int predLen;           /*the length of the strict predecessor    */
     char *rCon;            /*the right context                       */
     int rConLen;           /*the length of the right context         */
     char *succ;            /*the successor                           */
     int succLen;           /*the length of the successor             */
     };
typedef struct Production Production;
```

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include "a:generate.h"
/*********************************************************************/
/*      "main" organizes computation.                                */
/*********************************************************************/
main(argc,argv)
int argc;
char *argv[];
{
     char *string,*Derive();
     char axiom[MAXAXIOM];
     static char ignore[MAXCHARS];
     Production prodSet[MAXPROD];
     Parameter p;
     if (argc != 2) {
          printf("Usage: %s filename\n", argv[0]);
          exit(1);
     }
     p.filename = argv[1];
     input(axiom,ignore,prodSet,&p);      /*read the input file  */
     string = Derive(axiom,ignore,prodSet,p.n);/*generate        */
     interpret(string,&p);                    /*interpret       */
     return;
}
/*********************************************************************/
/*  Function "input" reads the input file                            */
/*********************************************************************/
input(axiom,ignore,prodSet,pPtr)
char axiom[];
char ignore[];
Production prodSet[];
Parameter *pPtr;
{
    FILE *fp, *fopen();
    char lcontext[MAXAXIOM],rcontext[MAXAXIOM],predecessor[MAXAXIOM];
    char successor[MAXAXIOM];
    char ignore_buf[MAXCHARS];
    char *inpStr;    /*inpStr will contain the contexts, the strict
                       predecessors and the successors of all
                       productions, separated by null characters*/
    char *malloc();
    int i;
    if ((fp = fopen (pPtr->filename,"r")) == NULL){
         printf("Can't open %s\n",pPtr->filename);
         exit(1);
    }
    if ((inpStr = malloc(MAXSTR)) ==NULL) {
         printf("Can't allocate inpStr\n");
```

```
        exit(1);
    }
    *inpStr++ = '\0';  /*Start inpStr with the null character
              needed to terminate the search for the left context*/
    fscanf(fp,"derivation length:%d\n",&pPtr->n);
    fscanf(fp,"angle factor:%d\n", &pPtr->angle);
    fscanf(fp,"scale factor:%d\n", &pPtr->scale);
    fscanf(fp,"axiom:%s\n",axiom);
    fscanf(fp,"ignore:%s",ignore_buf);
    /* Set flags corresponding to the ignored characters.          *
    for (i=0;i<MAXCHARS;i++)
        ignore[i] = 0;
    for (i=0; ignore_buf[i] !=0;i++) {
         ignore[ignore_buf[i]] = 1;
    }
    /* Read productions and enter them into the "prodSet" structure
    for (i=0; 1;i++,prodSet++) {
       prodSet->pred=inpStr;
       if (fscanf(fp, "%s < %s > %s --> %s",
              lcontext,predecessor,rcontext,successor) == EOF)
           break;
       enter(lcontext, &prodSet->lConLen, &prodSet->lCon, &inpStr);
       enter(predecessor,&prodSet->predLen,&prodSet->pred, &inpStr);
       enter(rcontext,&prodSet->rConLen,&prodSet->rCon, &inpStr);
       enter(successor,&prodSet->succLen,&prodSet->succ, &inpStr);
    };
    prodSet->predLen = 0;
    printf("%d productions read\n", i);
    fclose(fp);
    return;
}
/*****************************************************************/
/* Function "enter" is used to copy "string" to "inpStr" and fill*/
/* the fields in a "prodSet" structure.                          */
/*****************************************************************/
enter(string,lenPtr,prodStrHandle,inpStrHandle)
char *string;
int *lenPtr;
char **prodStrHandle, **inpStrHandle;
{
    if (*string == '*') {
        *lenPtr = 0;
        **inpStrHandle = '\0';
    }
    else {
         *lenPtr = strlen(string);
         strcpy(*inpStrHandle,string);
    }
    *prodStrHandle = *inpStrHandle;
```

```
    *inpStrHandle += *lenPtr + 1;
    return;
}
/*****************************************************************/
/*Given an axiom, a set of productions and a derivation length,  */
/*function "Derive" creates the generated string and returns a   */
/*pointer to it.  The parallel operation of an L-system is       */
/*simulated as follows.  First, string1 (containing the axiom) is*/
/*scanned from the left to the right.  Its consecutive substrings*/
/*are matched with the predecessors of productions.  The         */
/*appropriate successors are appended to the end of string2.     */
/*After all of string1 is scanned, string2 becomes string1.      */
/*The process is repeated until the desired derivation length    */
/*is achieved.                                                   */
 *****************************************************************/
char *Derive(axiom,ignore,prodSet,n)
char axiom[];
char ignore[];
Production prodSet[];
int n;
{
     char *curPtr,*nextPtr,*tempPtr,*limPtr;
     char *string1, *string2;
     int i;
     Production *FindProd();
     if ((string1 = malloc(MAXSTR)) == NULL) {
          printf("pfg: can't allocate string1\n");
          exit(1);
     }
     if ((string2 = malloc(MAXSTR)) == NULL) {
          printf("pfg: can't allocate string2\n");
          exit(1);
     }
     for (i=0;i<MAXSTR;i++)
          *(string1+i) = *(string2+i) = '\0';
     ++string1;     /* start with '\0' for proper context handling*/
     ++string2;
     strcpy(string1,axiom);
     limPtr = string2 + MAXSTR - MAXAXIOM;
     for (i=1;i<=n;i++) {
          printf("Computing derivation step %d\n",i);
          curPtr = string1;
          nextPtr = string2;
          while (*curPtr !='\0'){
              ApplyProd(&curPtr,&nextPtr,FindProd(curPtr,prodSet,ignore));
              if(nextPtr > limPtr) {
                 printf("String too long");
                 exit(1);
              }
```

```
            *nextPtr = '\0';
        }
    tempPtr = string1;
    string1 = string2;
    string2 = tempPtr;
    }
    return(string1);
}
/*****************************************************************/
/*  Copy the successor of the production *prodPtr starting at    */
/*  location *nexthandle.  Update curHandle and nextHandle.      */
/*****************************************************************/
ApplyProd(curHandle,nextHandle,prodPtr)
char **curHandle,**nextHandle;
Production *prodPtr;
{
   if (prodPtr != NULL) {
      strcpy (*nextHandle,prodPtr ->succ);
      *curHandle += prodPtr->predLen;
      *nextHandle += prodPtr->succLen;
   }
   else {
      **nextHandle = **curHandle;
      ++(*nextHandle);
      ++(*curHandle);
   }
   return;
}
/*****************************************************************/
/*Given a pointer to a string and a set of productions, return the    */
/*pointer to the first applicable production or NULL if no            */
/*matching production can be found.  The set of productions is        */
/*scanned in the same order in which productions are listed in        */
/*the input file.                                                     */
/*****************************************************************/
Production *FindProd(curPtr,prodSet,ignore)
char *curPtr;
Production prodSet[];
char ignore[];
{
   while (prodSet->predLen != 0) {
         if (prefix(prodSet->pred,curPtr) ||
             rcondiff(prodSet->rCon,curPtr + prodSet ->predLen,ignore) ||
             lcondiff(prodSet->lCon + prodSet->lConLen - 1,
                                                curPtr - 1,ignore))
             ++prodSet;
         else
             return(prodSet);
   }
   /*Predecessor not found*/
```

```
    return(NULL);
}
/*************************************************************************/
/*  Check whether string s1 is a prefix of string s2.                    */
/*************************************************************************/
prefix(s1,s2)
char *s1,*s2;
{
  while (*s1 != '\0')
     if (*s2++ != *s1++)
        return(1);
  return(0);
}
/*************************************************************************/
/* Check whether string s1 matches s2 as the right context.              */
/* Ignore specified symbols and skip over branches.                      */
/*************************************************************************/
rcondiff(s1,s2,ignore)
char *s1,*s2;
char ignore[];
{
    char *skipright();
    while(1) {
        if (*s1 =='\0')
           return(0);
        if(ignore[*s2])
           s2++;
        else if (*s2 == '[')
            s2 = skipright(s2+1) +1;
        else if (*s1++ != *s2++)
               return(1);
    }
}
/*************************************************************************/
/* Skip over a branch while searching for the right context.  The*/
/* branch may contain subbranches.                                       */
/*************************************************************************/
char *skipright(s)
char *s;
{
   int level = 0;
   while (*s != '\0') {
       switch (*s) {
       case '[':
            ++level;
            break;
       case ']':
           if(level == 0)
              return(s);
           else
```

```
                --level;
            break;
        default:
          break;
        }
        s++;
    }
    return(s);
}
/*********************************************************************/
/*Check whether string s1 matches s2 as the left context.          */
/*Ignore specified symbols and skip over branches.  The parent     */
/*branch belongs to the left context of a child branch.            */
/*********************************************************************/
lcondiff(s1,s2,ignore)
char *s1,*s2;
char ignore[];
{
    char *skipleft();
    while(1) {
        if(*s1 == '\0')
            return(0);
        if(ignore[*s2] || (*s2 == '['))
           s2--;
        else if (*s2 == ']')
            s2 = skipleft(s2);
        else {
            if (*s1-- != *s2--)
               return(1);
        }
    }
}
/*********************************************************************/
/* Skip over a branch while searching for the left context.  The */
/* branch may contain subbranches.                                   */
/*********************************************************************/
char *skipleft(s)
char *s;
{
    int level = 0;
    s--;
    while (*s != '\0') {
       switch(*s) {
           case ']':
              ++level;
              break;
           case'[':
             if(level == 0)
                return(--s);
```

```
            else
               --level;
            break;
         default:
            break;
         }
         s--;
      }
      return(s);
}
```

```
#define MAXANGLE    40        /*maximum value of the angle factor */
#define MAXSCALE    100       /*maximum value of the scale factor */
#define TWO_PI      6.2831853
#define STACK_SIZE  40        /*maximum depth of branches         */
#define MAXNUMBER   100       /*MAXNUMBER/2 : maximum number of
                                      vertex of the filled polygone*/
#define LEFT        10        /* window parameter                 */
#define TOP         15
#define RIGHT       600
#define BOTTOM      450

struct TURTLE {               /* turtle position and orientation  */
  double x;
  double y;
  short int dir;
  };
typedef struct TURTLE TURTLE;

struct PIXEL {                /* turtle position in screen        */
   short int h,v;             /* coordinates                      */
   };
typedef struct PIXEL PIXEL;

struct BOX {                  /* the bounding box of the curve    */
   double xmin,xmax;
   double ymin,ymax;
   };
typedef struct BOX BOX;
```

```
#include <graphics.h>
#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <math.h>
#include "a:generate.h"
#include "a:interpret.h"
/*****************************************************************/
/* Create the graphics environment in which the curve will be    */
/* drawn.  Find the bounding box of the curve, center it and draw*/
/* using the specified parameters.                               */
/*****************************************************************/
interpret(string,pPtr)
char *string;
Parameter *pPtr;
{
  int GraphDriver,GraphMode,ErrorCode,color,couleur;
  short int inc;      /* the step size                           */
  PIXEL start;        /* starting position of the turtle         */
  BOX boundBox;       /* bounding box of the curve               */
     /*Check the value of the angle factor                      */
  if (pPtr->angle > MAXANGLE) {
     printf("Angle factor too big (%d > %d) \n",
               pPtr ->angle,MAXANGLE);
     exit(1);
  }
     /* Compute the bounding rectangle of the curve, assuming    */
     /* that the mouse starts at the point (0,0) and the step    */
     /* size is equal to 1.                                      */
  start.h = start.v = 0;
  inc =1;
  draw(string,&start,inc,pPtr->angle,0,&boundBox);
     /* Given the bounding rectangle and the size factor,        */
     /* center the figure in the window.                         */
  SetDrawParam(&start, &inc,&boundBox,pPtr->scale);
     /* Initialize the drawing environment.                      */
  color=6;
  couleur=1;
  GraphDriver=DETECT;
  initgraph(&GraphDriver,&GraphMode,"..\\BGI");
  ErrorCode=graphresult();
  if (ErrorCode != grOk){
     printf("Graphics System Error:%s\n",grapherrormsg(ErrorCode));
     exit(1);
     }
  setbkcolor(color);
  setcolor(couleur);
  setviewport(LEFT,TOP,RIGHT,BOTTOM,1);
     /*Draw curve.                                              */
```

```
  draw(string,&start,inc,pPtr->angle,1,&boundBox);
  closegraph();
  return;
}
/*****************************************************************/
/*Draw figure defined by the "string".  The parameters have      */
/*the following meanings :                                       */
/*                                                               */
/*string   the string being interpreted                          */
/*startPtr initial position of the turtle                        */
/*inc      step size                                             */
/*angFac   the angle factor                                      */
/*flag     0 - make all lines invisible, 1 - draw.  The value    */
/*         of 0 is used when calculating the bounding rectangle. */
/*boxPtr   Pointer to the structure returning the coordinates    */
/*         of the bounding rectangle.                            */
/*                                                               */
/*The following string symbols are interpreted by the turtle :   */
/*                     +  -  |  [  ]  F  f                       */
/*****************************************************************/
draw(string,startPtr,inc,angFac,flag,boxPtr)
char *string;
PIXEL *startPtr;
short int inc,angFac,flag;
BOX *boxPtr;
{
   double SI[MAXANGLE],CO[MAXANGLE];
   TURTLE tu,stack[STACK_SIZE],*stackPtr,*bottom,*top;
   char c;
   int i,halfangFac;
   double ang = -TWO_PI/4;
   char *str;
   int polygone[MAXNUMBER];
   int k=0;
   int nbpoints=0;
   char figure='n';
   str = string;
   /*  Set stack limits.                                         */
   stackPtr = bottom = stack;
   top = stack + STACK_SIZE - 1;
   /* Precalculate coordinates for turtle steps in all possible  */
   /* directions.                                                */
   for(i=0;i<angFac;i++) {
      SI[i] = inc * sin(ang);
      CO[i] = inc * cos(ang);
      ang +=TWO_PI/angFac;
   };
   halfangFac = angFac/2;    /*needed to interpret the symbol |   */
   angFac--;                 /*more convenient for comparisons    */
```

```
/* Initialize the bounding rectangle and the starting position    */
/* of the turtle for the purpose of bounding rectangle            */
/* calculation.                                                   */
boxPtr->xmin = boxPtr->xmax = tu.x = (double) (startPtr->h) + 0.5;
boxPtr->ymin = boxPtr->ymax = tu.y = (double) (startPtr->v) + 0.5;
tu.dir=0;
/* Move the turtle to an appropriate starting position in         */
/* order to center the final drawing.                             */
if (flag)
    moveto(startPtr->h,startPtr->v);
/* Scan the string and interpret its consecutive symbols.         */
while ((c = *str++) != '\0') {
      switch (c) {
      case'+':        /* Turn right.                              */
         if(tu.dir<angFac)
           ++tu.dir;
         else
            tu.dir=0:
          break;
       case '-':      /* Turn left.                                */
         if(tu.dir>0)
           --tu.dir;
         else
           tu.dir=angFac;
         break;
       case '|':     /* Turn around.                               */
         if(tu.dir>=halfangFac)
            tu.dir-=halfangFac:
         else
            tu.dir+=halfangFac;
         break;
       case '[':     /* Push current state on stack.               */
         if(stackPtr >= top) {
              printf("Too many [\n");
              exit(1);
         }
         TurtleCopy(stackPtr,&tu);
         stackPtr++;
         break;
       case ']':    /* Pop state from the stack.                   */
         if(stackPtr <= bottom) {
              printf("Too many ]\n");
              exit(1);
         }
         --stackPtr;
         TurtleCopy(&tu,stackPtr);
         if(flag)
           moveto((short)(tu.x),(short)(tu.y));
         break:
```

```
case 'F':   /* Move forward and draw a line.                    */
   tu.x += CO[tu.dir];
   tu.y += SI[tu.dir];
   if(flag)
     lineto((short)(tu.x),(short)(tu.y));
   else
     BoxUpdate(&tu,boxPtr);
   break;
case 'f':   /* Move forward without drawing.                    */
   tu.x += CO[tu.dir];
   tu.y += SI[tu.dir];
   if (flag)
      moveto((short)(tu.x), (short)(tu.y));
   else
      BoxUpdate(&tu,boxPtr);
   if ((figure=='y') & (flag))
       {
       ++k;
       if (k== MAXNUMBER)
       {
            outtextxy(10,300,"MAXNUMBER is too little");
            getch();
            closegraph();
            exit(1);
       }
       else
       {
            polygone[k]=getx();
            ++nbpoints;
       }
       ++k;
       if (k==MAXNUMBER)
       {
            outtextxy(10,300,"MAXNUMBER is too little");
            getch();
            closegraph();
            exit(1);
       }
       else polygone[k]=gety();
       }
   break;
case '{':
   for(k=0;k<MAXNUMBER;k++)
       polygone[k]='\0';
   k=0;
   figure='y';
   nbpoints=1;
   polygone[k]=getx();
   ++k;
   polygone[k]=gety();
```

```
            break;
        case'}':
          if (flag)
          {
          setfillstyle(SOLID_FILL,GREEN);
          fillpoly(nbpoints,polygone);
          figure='n';
          }
          break;
        default:    /* room for extensions                              */
            break;
        }
}
if (flag)
    {
         outtextxy(10,400,"STRIKE ANY KEY TO CONTINUE...");
    getch();
    }
return;
}
/*********************************************************************/
/* Function "TurtleCopy" copies structure "fromPtr" to "toPtr".  */
/*********************************************************************/
TurtleCopy(toPtr,fromPtr)
TURTLE *toPtr, *fromPtr;
{
     toPtr->x = fromPtr->x;
     toPtr->y = fromPtr->y;
     toPtr->dir = fromPtr->dir;
     return;
}
/*********************************************************************/
/* Check whether the turtle moves outside the current box. If it */
/* does, adjust the box boundary.                                    */
/*********************************************************************/
BoxUpdate(tuPtr,boxPtr)
TURTLE *tuPtr;
BOX *boxPtr;
{
     if (tuPtr->x < boxPtr->xmin)
        boxPtr->xmin = tuPtr->x;
     if (tuPtr->x > boxPtr->xmax)
        boxPtr->xmax = tuPtr->x;
     if (tuPtr->y < boxPtr->ymin)
        boxPtr->ymin = tuPtr->y;
     if (tuPtr->y > boxPtr->ymax)
        boxPtr->ymax = tuPtr->y;
        return;
}
```

```
/*****************************************************************/
/* Set the starting point and the step increment, given the      */
/* bounding box and the scale factor.                            */
/*****************************************************************/
SetDrawParam(startPtr,incPtr,boxPtr,scale)
PIXEL *startPtr;
short int *incPtr;
BOX *boxPtr;
int scale;
{
   double xscale,yscale,sc;
   xscale = (RIGHT - LEFT)/(boxPtr->xmax - boxPtr->xmin);
   yscale = (BOTTOM - TOP)/(boxPtr->ymax - boxPtr->ymin);
   if(xscale>yscale)
      sc = yscale;
   else
      sc = xscale;
   *incPtr = (int) floor ((double) ((sc *scale)/MAXSCALE));
   startPtr->h=(short)
      (RIGHT - LEFT - *incPtr * (boxPtr->xmin + boxPtr->xmax -1.0))/2;
   startPtr->v = (short)
      (BOTTOM - TOP - *incPtr * (boxPtr->ymin + boxPtr->ymax -1.0))/2;
   return;
}
```

Annexe 2

Cet annexe contient les différents fichiers de données, utilisés par le programme "pfg", décrit dans le chapitre 4 et dont le listing se trouve en annexe 1.

Puisqu'il faut toujours introduire des "symboles ignorés", on peut prendre une lettre, qui n'est pas utilisée dans une production. Ici, j'ai pris la lettre "a".

1°) *figure 4.1*
derivation length : 3
angle factor : 4
scale factor : 100
axiom : F+F+F+F
ignore : a
* < F > * --> F+F-F-FF+F+F-F

2°) *figure 4.2*
derivation length : 4
angle factor : 16
scale factor : 100
axiom : F
ignore : a
* < F > * --> FF+[+F-F-F] - [-F+F+F]

3°) *figure 4.4*
derivation length : 25
angle factor : 16
scale factor : 100
axiom : F1F1F1
ignore : +-F
0 < 0 > 0 --> 1
0 < 0 > 1 --> 1[-F1F1]
0 < 1 > 0 --> 1
0 < 1 > 1 --> 1
1 < 0 > 0 --> 0
1 < 0 > 1 --> 1F1
1 < 1 > 0 --> 1
1 < 1 > 1 --> F0
* < + > * --> -
* < - > * --> +

4°) *figure 4.5*
derivation length : 5
angle factor : 18
scale factor : 100
axiom : X
ignore : a
* < X > * --> F[+X] F[-X] +X

* < F > * --> FF

5°) *figure 4.7*
derivation length : 1
angle factor : 18
scale factor : 100
axiom : L
ignore : a
* < L > * --> {[++++f] [++Ff] [+FFf] [FFFFf] [-FFf] [--Ff] [----f] }

6°) *figure 4.8*
derivation length : 1
angle factor : 12
scale factor : 100
axiom : L
ignore : a
* < L > * --> {[+++f] [++Ff] [+FFf] [FFFf] [-FFf] [--Ff] [---f] }

7°) *figure 4.9*
derivation length : 5
angle factor : 4
scale factor : 100
axiom : +X
ignore : a
* < X > * --> -YF+XFX+FY-
* < Y > * --> +XF-YFY-FX+

8°) *figure 4.10*
derivation length : 10
angle factor : 36
scale factor : 100
axiom : SLFFF
ignore : a
* < S > * --> [+++G] [---G] TS

* < G > * --> +H [-G] L

* < H > * --> - G [+H] L

* < L > * --> [-FFF] [+FFF] F

9°) *figure 4.11*
derivation length : 5
angle factor : 14
scale factor : 100
axiom : X
ignore : a
* < X > * --> F[+X][-X]FX
* < F > * --> FF

10°) *figure 5.2*
derivation length : 4
angle factor : 12
scale factor : 100
axiom : +++F----F----F
ignore : a
* < F > * --> F++F----F++F