TER L3 Informatique

Faire pousser des arbres en programmant

(Interprétation graphique d'une grammaire L-Système)

Sous la responsabilité
de
Jean-Florent Raymond

Réalisé par

Nicolas Théron	nicolas.theron@etu.umontpellier.fr
Clément Daumet	clement.daumet@etu.umontpellier.fr
Dorine Tabary	dorine.tabary@etu.umontpellier.fr

Université Montpellier 2
Faculte des sciences, section informatique
2 Place Eugène Bataillon, 34090 Montpellier

semestre 6, année 2015-2016

Introduction

Le sujet :

Le travail à effectuer se divisait en plusieurs étapes :
- compréhension du principe du L-system [1] grâce à la lecture du livre *The Algorithmic Beauty of Plants* ;
- écrire un programme qui génère des mots à partir d'un ensemble de règles lues dans un fichier ;
- produire une image qui représente une plante en deux dimensions ;

Nous pouvions aussi aller plus loin selon ces directions :
- génération et dessin d'objets 3D ;
- réaliser des L-systems stochastiques ;

Le travail réalisé

Pendant les quatre mois impartis, nous avons donc :
- fait tout ce qu'il nous était demandé de faire ;
- enrichi les possibilités d'arbre de base (base multiple, longueur, largeur) ;
- étayé les règles de construction (contexte sensitif, gravité, constante).

L'environnement utilisé

C++ comme langage de programmation principal

Ce langage a été choisi pour sa puissance de calcul et pour pouvoir manipuler explicitement des registres et des adresses mémoires. Il permet ainsi de comprendre au mieux le fonctionnement des algorithmes utilisés.

Python comme lien avec Blender

Nous avons aussi choisi Python, langage de programmation objet interprété car il facilite le travail sur les chaînes lors du transfert d'informations vers le logiciel de rendu 3D Blender [2] (logiciel libre, gratuit et correspondant à nos besoins).

Le GIT, comme outil de communication

Enfin, l'environnement de dépôt git a été utilisé pour communiquer et travailler en groupe.

1. Lindenmayer system
2. Blender est un logiciel libre de modélisation, d'animation et de rendu en 3D

Table des matières

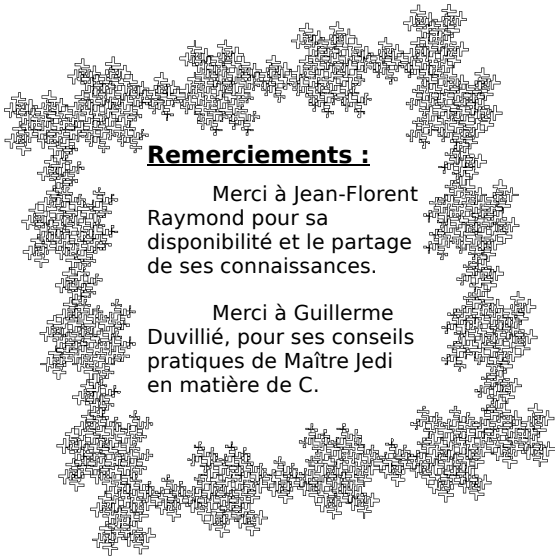
1 -Présentation **3**
- 1.1 -Un peu d'histoire . . . 3
- 1.2 -Fonctionnement . . . 3
- 1.3 -Les domaines associés . . . 5
- 1.4 -Intérêts . . . 5
- 1.5 -Les commandes . . . 6

2 -Le projet **7**
- 2.1 -Etapes de création des L-Systems. . . . 7
- 2.2 -Les options rajoutées . . . 8
- 2.3 -La troisième dimension . . . 10

3 -Conclusion **12**

References **13**

Annexe **14**




```
1  angleXY=90.
2  angleZ=0.
3  largeur=1.
4  longueur=10.
5  XYXYXYX+XYXYXYX+XYXYXYX+XYXYXYX.
6  X=FX+FX+FXFY-FY-.
7  Y=+FX+FXFY-FY-FY.
```

Générateur du cadre

Chapitre 1-Présentation

1.1-Un peu d'histoire

Les L-Systems [1] ont été créés par le biologiste Aristid Lindenmayer (1925-1989) afin d'étudier la croissance des végétaux.
Par la suite, Przemysław Prusinkiewicz a étendu le domaine aux trois dimensions et a permis ainsi de les visualiser.
Les L-Systems sont donc des modèles grammaticaux qui peuvent décrire de façon compacte la complexité d'une figure.

1.2-Fonctionnement

Un L-System est un système destiné à représenter des plantes. Chaque plante est considérée comme une répétition de modèles simples.
Il suffit dès lors, de trouver la façon dont ces modèles sont répétés pour pouvoir déduire la règle qui régit ces plantes.

1.2.1-Grammaire associée

-Théorie :

Un L-system est une grammaire formelle notée $\{V, S, \omega, P\}$.
Pour plus de détails, cliquez ici.

Exemple :

TABLE 1.1 – Le générateur

axiome de Base :	FAA
Règles :	A=F+B B= F-A

TABLE 1.2 – Chaînes générées

Itération	Chaîne sortante
0	FAA
1	FF+B F+B
2	FF+F-AF+F-A
3	FF+F-F+B F+F-F+B
4	FF+F-F+F-AF+F-F+F-A

1. Lindenmayer system

1.2.2-La méthode de la tortue

Exemple : la courbe quadratique de Koch (type 1)

FIGURE 1.1 – Koch 0

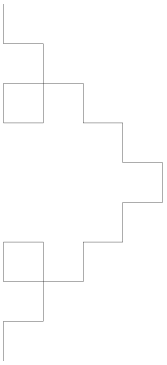


FIGURE 1.2 – Koch 1

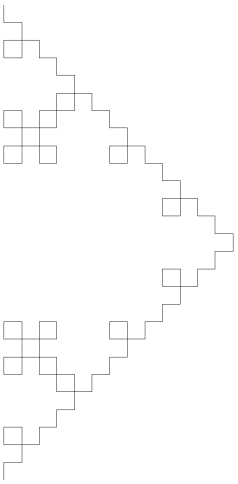


FIGURE 1.3 – Koch 2

TABLE 1.3 – Le générateur

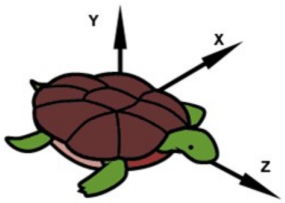
axiome de Base :	F
Règles :	F= F+F-F-F+F

Pour plus de détails, cliquez ici.

TABLE 1.4 – Chaînes générées

Itération	Chaîne sortante
0	F
1	F +F -F -F +F
2	F+F-F-F+F+F+F-F-F+F-F+F-F-F+F -F+F-F-F+F+F+F-F-F+F

En théorie :



La chaîne de caractères obtenue a une interprétation graphique, en deux ou trois dimensions.
En deux dimensions, on imagine qu'une main tient un crayon qui se déplace sur la feuille selon des instructions : "monte d'un cran, puis tourne de 20 ◦ à gauche, déplace-toi deux fois de un cran, mémorise ta position et avance encore d'un cran, lève-toi puis repose-toi sur la position mémorisée" et ainsi de suite...
On introduit donc des symboles variants ∈ V, ou constants ∈ S, pour permettre de guider la main. Plusieurs d'entre eux ont été normalisés. Ils font partie de ce qu'on appelle la "Turtle interpretation" [2].
Cette tortue est la main qui tient le crayon.
Les signes couramment utilisés sont les suivants :

2. Ce nom vient de la "tortue" du langage de programmation Logo fonctionnant sur le même principe.

— F : se déplacer d'un pas unitaire ($\in$ V) ;
— + : tourner à gauche d'angle α ($\in$ S) ;
— - : tourner à droite d'un angle α ($\in$ S) ;
— $\backslash$: pivoter vers le bas d'un angle α ($\in$ S)
— / : pivoter vers le haut d'un angle α ($\in$ S) ;
— $\sqsubset$: sauvegarder la position courante ($\in$ S) ;
— $\sqsupset$: restaurer la dernière position sauvée ($\in$ S).

1.3-Les domaines associés

Ce projet s'inscrit au contingent de plusieurs domaines informatiques.
A la base, un L-system est une grammaire formelle [3].
De même, la modélisation demande des connaissances en algorithme des graphes (avec de nombreux parcours d'arbre au sens littéral), en système d'exploitation (afin de pouvoir récupérer des données dans un fichier).
Les options rajoutées s'imprègnent aussi d'autres domaines informatiques tels que l'intelligence artificielle.
Enfin, la programmation en soit, nécessite des connaissances en des langages comme le C++ et le Python et de pouvoir s'adapter à de nouveaux types de logiciels comme Blender.

1.4-Intérêts

Actuellement, les intérêts des L-Systems sont multiples et variés. Leur étude permet des avancées dans plusieurs domaines de recherche comme :

— en biologie, par l'étude du développement et de la prolifération des plantes ou des bactéries au niveau microscopique ;
— en écologie, avec l'acquisition de connaissances, l'étude et l'analyse de données, la représentation ou la description de processus biologiques complexes du monde végétal. A terme, les L-Systems permettront de :
  — comprendre les différentes échelles spatiales et temporelles dans les modèles en fonction de la question biologique initiale ;
  — fournir des outils d'aide à la décision notamment pour l'étude des scenarii de perturbation (activités humaines, changement climatique, etc...) ;
  — pouvoir gérer et analyser un afflux de données massives ;
— en graphisme, toujours dans la conception d'illustrations avec l'utilisation de schémas prédictibles ;
— en intelligence artificielle [4].

3. Une grammaire est un formalisme permettant de définir une syntaxe et donc un langage formel (ensemble de mots admissibles sur un alphabet donné). .
4. C'est d'ailleurs l'une des premières applications pratiques de la vie artificielle.

1.5-Les commandes

Afin, d'appréhender l'étendue des possibilités offertes par ce système, l'utilisation parallèle de plusieurs interfaces est préférée.

1.5.1-Le fichier générateur

Le fichier générateur, de type texte consiste en une fiche descriptive de l'arbre comprenant :

- l'angle de référence dans le plan XY ;
- l'angle de référence dans le plan YZ ;
- la largeur de base du tronc ;
- la longueur d'une branche ;
- le contexte ;
- le ou les axiome(s) de base ;
- les règles de construction spécifiques à l'arbre.

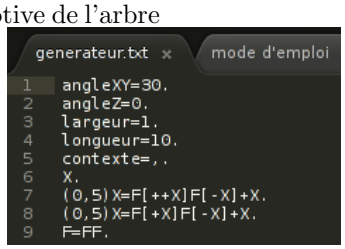


FIGURE 1.4 – Générateur

1.5.2-Le terminal de commande

Le terminal de commande est utilisé afin de :

- voir grandir l'arbre itération par itération ;
- créer l'arbre directement en proposant d'entrer dans le terminal les caractéristiques de l'arbre que l'on trouve dans le fichier texte "générateur".

1.5.3-L'add-on pour Blender.

Pour utiliser Blender, un add-on [5] a été spécialement programmé afin d'interpréter plus facilement les arbres générés en trois dimensions.

Cet add-on offre trois intéractions à l'utilisateur :

- un path ;
- un itérateur ;
- le bouton d'ajout.

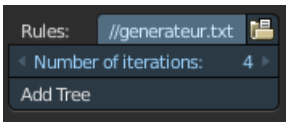


FIGURE 1.5 – aperçu de l'add-on dans Blender

Le "path" ouvre un gestionnaire de fichiers permettant de spécifier le fichier de règles de construction que l'on souhaite utiliser pour générer notre arbre.

Le second permet de choisir le nombre d'itérations de la règle choisie afin de former l'arbre.

Le dernier lance un appel système sur le programme avec les paramètres spécifiés. Un fichier ".obj" est généré de façon à correspondre à l'arbre demandé. L'add-on importe automatiquement le fichier.obj dans la scène.

5. Programme destiné à rajouter des possibilités supplémentaires à un autre programme.

Chapitre 2-Le projet

2.1-Etapes de création des L-Systems.

La création d'arbres se divise en plusieurs étapes :

1. lecture d'un fichier texte, générateur d'un type arbre, et enregistrement des données caractéristiques de cet arbre ;
2. faire pousser l'arbre un nombre de fois spécifié par l'utilisateur (que ce soit à travers un nombre ou itération par itération) ;
3. créer le visuel de l'arbre en 2D sur un fichier post script ;
4. créer le visuel de l'arbre en 3D grâce à Blender.

2.1.1-Lecture du fichier texte

Toutes les données caractéristiques de l'arbre sont enregistrées dans des variables spécifiques.
Le fichier doit obéir à des règles précises de forme. Les variables booléennes correspondant à chacunes des caractéristiques de l'arbre sont initialisées à "Faux" et deviennent "Vrai" après le passage de la tête de lecture et le remplissage de la variable correspondante.
Cliquez ici pour voir l'algorithme correspondant.

2.1.2-Arbre initial : le DOL-System

Le DOL-system ou Deterministic 0-context System est par définition déterministe. Ainsi, il n'offre qu'une seule évolution possible depuis l'axiome à la génération choisie. Comme une cause n'engendre qu'un seul effet, une variable ne peut subir qu'un seul type de transformation, toujours identique.
Il n'y a de ce fait qu'une seule règle par variable.
Cliquez ici pour voir des exemples.

Les principales classes utilisées :

— la classe noeud représente une branche de l'arbre et est définie par les variables spécifiques aux branches (inclinaisons, largeur, nom) ;
— la classe arbre réunit tous les noeuds propres à un arbre et regroupe les spécificités générales sauvegardées à partir du générateur.

L'utilisation du retour sur trace

Le retour sur trace (appelé aussi backtracking) est un algorithme qui permet de revenir en arrière. Dans notre cas, il permet d'accéder aux caractéristiques d'une précédente branche à partir de laquelle sera construite d'autres branches. Pour des raisons d'économie de mémoire, les appels récursifs ont été préférés à l'utilisation d'une pile.

Exemple avec la chaîne : [+F[F[F]−F]]-F

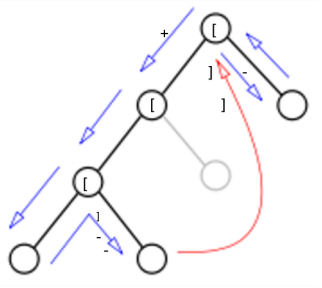


FIGURE 2.1 – BackTrack

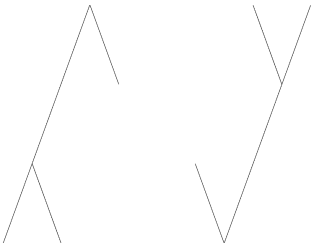


FIGURE 2.2 – Arbre inversé

FIGURE 2.3 – Arbre final

Explications

Chaque crochet, symbolisant une nouvelle branche appelle l' algorithme de backtracking. Les variables sont passées en paramètres et modifiées (telles que inclinaisonXY).

Pour un supplément de détails, cliquez ici.

2.1.3-Modélisation de l'arbre en deux dimensions

Afin de modéliser l'arbre sur un fichier post script, on réalise d'abord un premier parcours en profondeur de l'arbre afin de connaître ses dimensions . Ensuite, on réalise un second parcours de l'arbre à partir de la racine pour transformer et lier chaque noeud en des données visibles.
La commande dans le terminal "evince arbre.ps &" permet ainsi de voir littéralement, itération par itération pousser l'arbre.

2.2-Les options rajoutées

2.2.1-Axiomes multiples

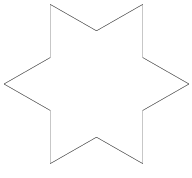


FIGURE 2.4 – Flocon de Koch

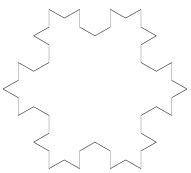


FIGURE 2.5 – i=1 Flocon

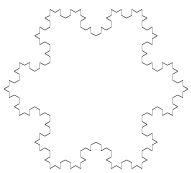


FIGURE 2.6 – i=2 Flocon

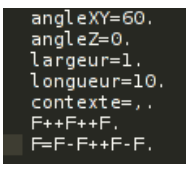


FIGURE 2.7 – Générateur

L'astuce, pour la création d'axiomes multiples consiste à définir à la base les axiomes multiples comme dérivés d'un axiome atomique virtuel.
Exemple : A=F++F++F (avec A, axiome unique de base.)

Cliquez ici pour voir d'autres exemples.

2.2.2-Les SOL-Systems

```
1  angleXY=30.
2  angleZ=0.
3  largeur=1.
4  longueur=10.
5  contexte=,.
6  X.
7  (0,5)X=F[++X]F[-X][--X]+X.
8  (0,5)X=F[+X]F[--X][-X]+X.
9  F=FF.
```

FIGURE 2.8 – Générateur

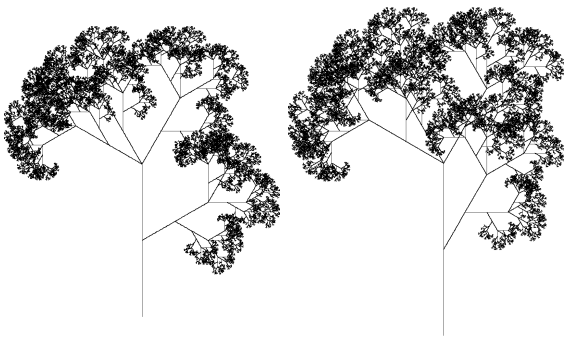


FIGURE 2.9 – i = 7 : Le premier arbre

FIGURE 2.10 – i = 7 : Le second arbre

Définition :

Les SOL-Systems ou Stochastic 0-context System fait appel aux probabilités. Il est aussi appelé système non-déterministe. Contrairement au DOL-system, il est possible de déterminer plusieurs transformations pour un symbole. Chaque possibilité est pondérée pour pouvoir donner la priorité à certaines transformations par rapport à d'autres.

La classe probabilité :

Elle a comme attributs un string et un double.
S'il n'y a pas de probabilité indiquée, la valeur par défaut est de 1 (100%).
La classe arbre possède un vecteur des probabilités.

Cliquez ici pour plus de détails.

2.2.3- IL-System ou encore (k, l)-System

Les deux systèmes précédents (des OL-Systems) ne peuvent pas simuler l'intéraction de parties d'une plante car ils sont context-free [1]. Un L-System context-sensitive résout ce problème en prenant en compte ce qui précède ou succède à une partie, c'est-à-dire un symbole. Un tel système est appelé IL-System ou encore (k, l)-System, le contexte de gauche est un "mot" de longueur k et celui de droite un "mot" de longueur l.

1. chaque partie se développe indépendamment des autres parties.

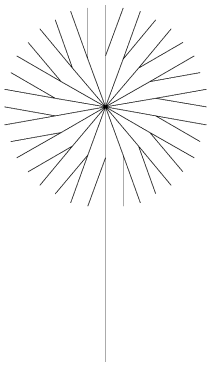


FIGURE 2.11 – Pissenlit

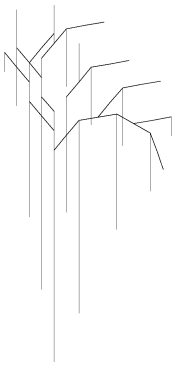


FIGURE 2.12 – Saule pleureur

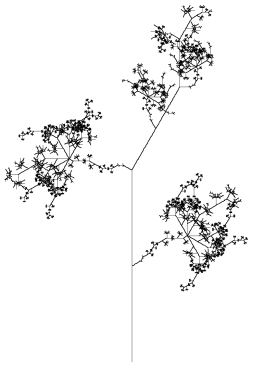


FIGURE 2.13 – Arbre à fleurs

La classe contexte :

Elle a comme attributs :
- un string 'règles anciennes' (la suite de lettres qui changera) ;
- un string ' règles nouvelles' (la suite de lettres à insérer) ;
- un entier correspondant à la position du caractère à remplacer par les règles nouvelles ;
- un booléen nommé gravité. S'il est activé, la valeur de l'inclinaisonXY de la branche et de tous ses descendants est de 180 (vers le bas).

La classe arbre possède un vecteur des contextes.
En cas de conflit, le premier contexte est prioritaire sur les suivants.
Cliquez ici pour plus de détails.

2.3-La troisième dimension

2.3.1-Rajout de la largeur et de la profondeur.

La largeur :

Concernant la largeur, visible qu'en 3D, nous avons d'abord opté pour une définition automatique en fonction du nombre de branches.
L'utilisateur n'entrait qu'une seule valeur correspondant à celle du tronc de base. Cette largeur était automatiquement divisée en fonction du nombre de branches.
Cependant, il était impossible avec cette méthode de faire pousser des arbres comme le baobab, le bamboo, ou le saule pleureur.
C'est pour cela que nous avons choisi une définition manuelle du même type que les inclinaisons ou la largeur. Les commandes dans les règles de construction "*"et "∼" permettent de faire varier la largeur des branches.

La 3D nous ouvre ses branches avec la profondeur :

Dès le début, chaque noeud de l'arbre excepté le 1^{er} était décrit en coordonnées sphériques. Nous utilisions aussi des fonctions capables de passer des

coordonnées polaires à cartésiennes et vice versa.

Pour rajouter la 3^{eme} dimension, nous avons dû modifier les ordonnées comme suit :

— x=r $\sin\varphi\cos\theta$;
— y=r $\sin\varphi\sin\theta$;
— z=r $\cos\varphi$.

avec :

— φ correspond à inclinaisonXY ;
— θ correspond à inclinaisonZ ;
— r correspond à la longueur d'une branche.

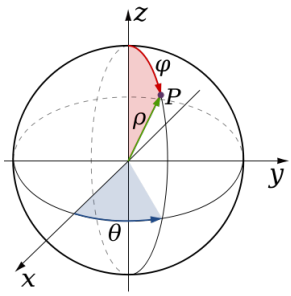


FIGURE 2.14 – point en coordonnées sphériques

2.3.2-Code avec Blender Python

Les choix d'exportation

Le format wavefront a été choisi comme format d'exportation pour son adaptabilité et son interopérabilité [2]. Wavefront permet ainsi d'exporter facilement un ou plusieurs objets 3D, qu'ils soient composés de vertices [3], de segments ou de faces, et ce accompagnés des matérieux pour ces faces.

L'arbre est donc exportable :

— sous forme d'arêtes (on a donc la forme générale de l'arbre) ;
— avec des contours solides (on peut ainsi tenir compte des variations de la largeur du tronc).

L'exportation des arbres se fait par un parcours en profondeur. Les coordonnées de chaque noeud sont exportées et correspondent aux spécificités de l'arbre.

Blender Python

Le Blender Python (.bpy) écrit en Python inclut les bibliothèques spécifiques à Blender. Python permet ainsi de créer un add-on pour Blender grâce à l'importation de l'arbre dans une scène de rendu.

Dès la mise en place de l'add-on, le moteur de rendu est modifié avec par exemple le rajout d'un panneau permettant à l'utilisateur de choisir son générateur ainsi que le nombre d'itérations à effectuer.

Cliquez ici pour voir plus de détails.

2. capacité que possède un système à fonctionner avec d'autres systèmes et ce sans restriction d'accès ou de mise en œuvre
3. Point d'intersection entre deux ou plusieurs segments dans une construction 3D.

Cliquez sur la vidéo ci dessus pour un aperçu.
(nécessite la vidéo dans le même répertoire ; dépend des lecteurs de pdf)

Chapitre 3-Conclusion

Bilan de nos travaux

Etudier un sujet aussi étendu que les L-Systems est passionnant. Nous avons appris beaucoup de choses dans de vastes domaines allant d'une expérience pratique de modélisation aux différentes étapes de la création d'arbres, en passant par des exercices de programmation en C++/Blender/Python. Il ne faut pas oublier non plus, l'étude théorique des différents algorithmes pouvant répondre aux nombreux problèmes qui se sont posés à nous.
Il était très agréable de voir les idées théoriques se concrétiser en 3D.

Améliorations possibles

Nous pouvons continuer ce projet en de nombreuses façons comme :
- une meilleure gestion de la mémoire. Malgré nos efforts, il subsiste une légère perte de mémoire visible grâce à "valgrind" ;
- la colorisation des arbres 2D et 3D ;
- la création d'une application avec une interface graphique personnalisée ;
- la possibilité d'afficher en 2D et en 3D plusieurs arbres différents côte à côte ;
- modéliser les différents types de reproduction d'arbres à travers les graines ;
- étudier la concurrence entre ces différents arbres avec par exemple des algorithmes distribués (comme l'algorithme de Naimi-Trehel) ;

Bibliographie

[1] http://marief.soler.free.fr/Monsite/lsystem.html,

[2] algorithmicBeautyOfPlants, http://algorithmicbotany.org/papers/abop/abop.pdf

[3] http://www.kevs3d.co.uk/dev/lsystems/,

[4] http://amap.cirad.fr/fr/axe3.php,

[5] http://www.futura-sciences.com/magazines/high-tech/infos/dossiers/d/technologie-vie-artificielle-vers-avenir-1438/page/10/,

[6] https://fr.wikipedia.org/wiki/L-Syst%C3%A8me

[7] http://c.developpez.com/

[8] https://www.blender.org/api/blender_python_api_2_77_0/

[9] https://fr.wikipedia.org/wiki/Liste_de_fractales_par_dimension_de_Hausdorff

Annexes

Grammaire des L-Systems

Un L-System est une grammaire formelle qui comprend :
- un alphabet V : l'ensemble des variables du L-System. V^* est l'ensemble des « mots » que l'on peut construire avec les symboles de V, et V^+, l'ensemble des mots contenant au moins un symbole ;
- un ensemble de valeurs constantes S. Certains de ces symboles sont communs à tous les L-Systems (plus d'informations avec la Turtle interpretation) ;
- un axiome de départ ω choisi parmi V^+, c'est-à-dire l'état initial ;
- un ensemble de règles, noté P, de reproduction des symboles de V.

Un L-System est donc noté $\{V, S, \omega, P\}$.

Cliquez ici pour retourner aux explications.

Koch et d'autres fractales

Courbe de Koch, de Sierpinski, et fractales

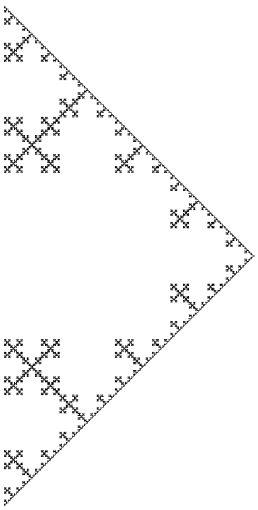


FIGURE 3.1 – Courbe de Koch après 7 itérations

```
1  angleXY=90.
2  angleZ=0.
3  largeur=1.
4  longueur=10.
5  contexte=,.
6  F.
7  F=F+F-F-F+F.
```

FIGURE 3.2 – Générateur

Pour retourner aux explications, cliquez ici

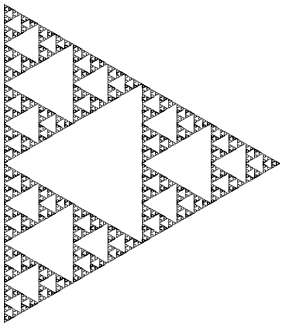


FIGURE 3.3 – Courbe de Sierpiński après 8 itérations

```
1  angleXY=90.
2  angleZ=0.
3  largeur=1.
4  longueur=10.
5  contexte=,.
6  F.
7  F=F+F-F-F+F.
```

FIGURE 3.4 – Générateur

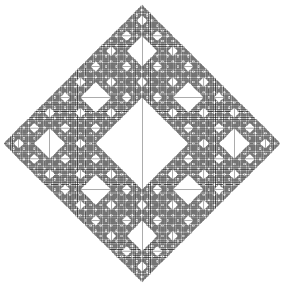


FIGURE 3.5 – Fractale après 4 itérations

```
angleXY=90.
angleZ=0.
largeur=1.
longueur=10.
contexte=,.
F.
F=F+F-F-F-G+F+F+F-F.
G=GGG.
```

FIGURE 3.6 – Générateur

Lecture de fichiers

Le manuel d'utilisation

Le fichier générateur, de type texte consiste en une fiche descriptive de l'arbre qui doit obéir à de nombreuses règles expliquées ci-dessous avec leur ligne correspondante.

```
generateur.txt ×   mode d'emploi
1  angleXY=30.
2  angleZ=0.
3  largeur=1.
4  longueur=10.
5  contexte=,.
6  X.
7  (0,5)X=F[++X]F[-X]+X.
8  (0,5)X=F[+X]F[-X]+X.
9  F=FF.
```

FIGURE 3.7 – Générateur

1. ligne correspondant à l'angle de référence dans le plan XY. Le degré doit obligatoirement être précisé après "angleXY=" ;
2. ligne correspondant à l'angle de référence dans le plan Z. Le degré doit obligatoirement être précisé après "angleZ=" ;
3. ligne correspondant à la largeur du tronc, visible qu'en 3D. La largeur doit être précisée après "lar-

geur=" ;

4. ligne correspondant à la longueur d'une branche. La longueur doit obligatoirement être précisée après "longueur=". C'est la seule variable non modifiable par la suite ;
5. ligne correspondant au contexte. S'il est vide. La ligne 5 doit être "contexte=," (voir au chapitre correspondant pour plus de précisions) ;
6. ligne correspondant à ou aux axiome(s) de base ;
7. et plus : lignes correspondant aux règles de construction spécifiques à l'arbre (voir aussi au chapitre correspondant).

Toutes les valeurs de variables entrées doivent être un nombre, et finir par un ".".

Fragment de code :

```
  bool angleXYFait = false;
  bool angleZFait = false;

while(getline(file, line))//accède ligne par ligne aux donnees
{
  b = false;//est utilisé plus tard pour initialiser chaque caractéristique stochastique.
  for(unsigned int k = 0; k < line.size(); ++k)//accède à chaque lettre de la ligne.
  {
    if(!angleXYFait)//permet de remplir une par une les spécificités de l'arbre.
    {
      if((k >= 8) && (line[k] != '.'))// récupère les données juste apres le "=" correspondant à l'angleXY.
      {
        //L'objectif est de modifier la valeur de angleXY de la classe arbre.
        //L'angle est initialisé à 0.
        //on accède à la valeur de chaque chiffre en enlevant 48 du caractère ascii correspondant.
        //Le premier chiffre noté c1 correspond à 0*10+c1. On notera le nombre obtenu n1.
        //le second chiffre noté c2 correspond à n1*10+c2. On notera le nombre obtenu n2, et ainsi de suite.
        setAngleXY(getAngleXY()*10+line[k] - 48); //On récupère donc le nombre caractéristique de l'angle XY.
      }
      if(line[k] == '.')//Dans le cas où la ligne est finie.
      {
        angleXYFait = true;//l'angleXYFait est initialisé à vrai, on peut donc passer à l'angleZ.
        cout << "angleXY=" << getAngleXY() << endl;
      }
    }
    else
    {
      if(!angleZFait)//dans le cas où l'angleZ n'est pas fait.
      {
        if((k >= 7) && (line[k] != '.'))// récupère les données juste apres le "=" correspondant à l'angleXY.
          {
            setAngleZ(getAngleZ()*10+line[k] - 48);//on récupère le nombre correspondant à l'angle Z.
          }
        if(line[k] == '.')//la ligne correspondant à l'angle XY est finie
        {
          angleZFait = true;
          cout << "angleZ=" << getAngleZ() << endl;//l'angleZFait est initialisé à vrai, on peut donc passer à l'angleZ.
        }
      }
      else
      {
        if(!poidsArbreFait)
        {
        // cout << "line[" << k << "]=" << line[k] << endl;
          if((k>=8)&&(line[k] != '.'))// recupere la premiere donnee correspondant Á  l'angleXYXY
          {
            setPoids(getPoids()*10+line[k] - 48);
```

FIGURE 3.8 – initialisation des angles XY et Z

Pour retourner aux explications, cliquez ici.

Annexes sur les DOL systèmes

```
angleXY=20.
angleZ=0.
largeur=1.
longueur=10.
contexte=,.
F.
F=F[+F][-F[-F]F[+F][-F].
```

FIGURE 3.9 – Générateur

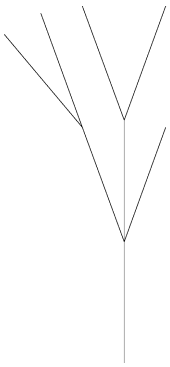


FIGURE 3.10 – i=1 Arbre DOL

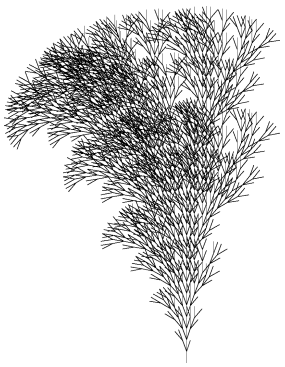


FIGURE 3.11 – i=4 Arbre DOL

Cliquez ici pour retourner à l'explication.

Annexes sur le backtrack

Complément d'explications :

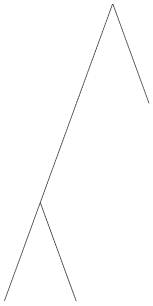


FIGURE 3.12 – Arbre inversé

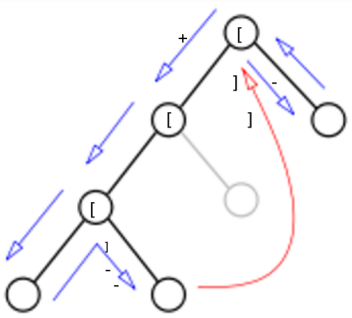


FIGURE 3.13 – BackTrack

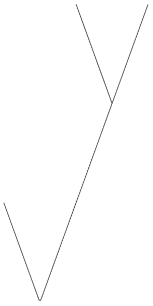


FIGURE 3.14 – Arbre final

A travers le schéma ci-dessus correspondant à [+F [F[F]–F]]-F, le premier crochet [appelle une fonction avec une inclinaison "γ". Le "+" modifie le "γ" qui devient "γ_b", nouvelle inclinaison de la branche dans cette nouvelle fonction. Les crochets suivants [et [appellent des fonctions avec des inclinaisons "γ_r" et "γ_g" identiques à "γ_b" et construisent les branches associées.
Le crochet] permet de quitter la dernière fonction.
On a donc γ_r, qui est modifié par le "-". Une nouvelle branche correspondant est construite.
Les crochets] et] permettent de quitter les deux fonctions précédentes. L'angle est γ, modifié ensuite par le "-". Une dernière branche est construite.

L'algorithme correspondant au backtrack.

Data: string "a", chaîne de caractère à modéliser
Result: affichage graphique pour la modélisation 2D, et l'exportation de la racine pour la modélisation 3D.
creation de la racine, de type "noeud";
inclinaisons des variables initialisées à "0";
while *le curseur pointe une lettre dans "a"* **do**
  if *lettre pointée est '['* **then**
    curseur ← fonction arbrePere;
  else
    les inclinaisons sont modifiées en fonction de la lettre pointée;
  end
end
afficher en 2D l'arbre;
exporter la racine;

Algorithm 1: Algorithme d'amorce du BackTrack

Data: string "a"(chaîne de caractères à modéliser), entier m (position du curseur), noeud *pere (noeud père du noeud qui va être créé), variables inclinaisons.
Result: un entier "m", position du curseur.
while *le curseur pointe une lettre dans "a"* **do**
  if *lettre pointée est ']'* **then**
    retourner "m", position du curseur;
  end
  if *lettre pointée est '['* **then**
    m←arbrePere;
  end
  if *lettre pointée est un symbole du type "+,-,*..."* **then**
    modifier les inclinaisons en fonction du symbole;
  end
  if *lettre pointée appartient à l'alphabet* **then**
    créer un nouveau noeud nommé "enfant" avec les inclinaisons modifiées;
    pere←enfant;
  end
  m←m+1;
end
retourner m;

Algorithm 2: ArbrePere : Algorithme du BackTrack en soit

Pour retourner aux explications, cliquez ici.

Annexes sur les axiomes multiples

Triangle de Sierpiński

Remarque : une fractale est un espace métrique dont la dimension de Hausdorff est strictement supérieure à la dimension topologique.

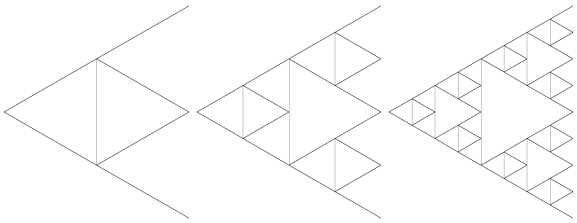


FIGURE 3.15 – Triangle

FIGURE 3.16 – i=1 Triangle

FIGURE 3.17 – i=2 Triangle

```
angleXY=120.
angleZ=0.
largeur=1.
longueur=10.
contexte=,.
F-G-G.
F=F-G+F+G-F.
G=GG.
```

FIGURE 3.18 – Générateur

Fractale de forme multi-hexagonale

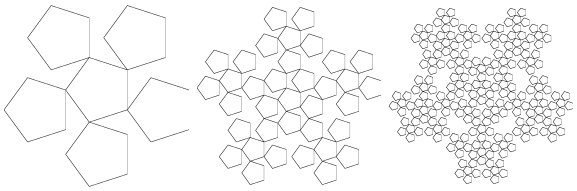


FIGURE 3.19 – Hexagones

FIGURE 3.20 – i=1 Hexagones

FIGURE 3.21 – i=2 Hexagones

```
angleXY=72.
angleZ=0.
largeur=1.
longueur=10.
contexte=,.
F-F-F-F-F.
F=F-F++F+F-F-F.
```

FIGURE 3.22 – Générateur

Fractales à base hexagonale

Pour retourner aux explications, cliquez ici

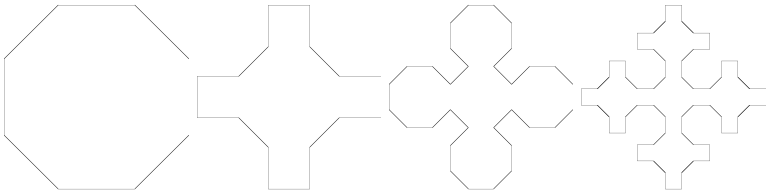


FIGURE 3.23 – i=0 Fractale

FIGURE 3.24 – i=1 Fractale

FIGURE 3.25 – i=2 Fractale

FIGURE 3.26 – i=3 Fractale

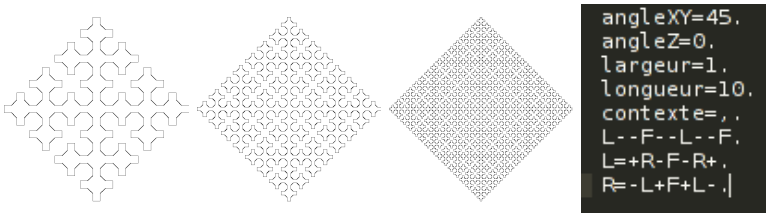


FIGURE 3.27 – i=5 Fractale

FIGURE 3.28 – i=7 Fractale

FIGURE 3.29 – i=9 Fractale

```
angleXY=45.
angleZ=0.
largeur=1.
longueur=10.
contexte=,.
L--F--L--F.
L=+R-F-R+.
R=-L+F+L-.
```

FIGURE 3.30 – Générateur

Annexes sur les SOL-Systems

En pratique :

```
angleXY=30.
angleZ=0.
largeur=1.
longueur=10.
contexte=,.
X.
(0,5)X=F[++X]F[-X][--X]+X.
(0,25)X=F[+X]F[-X]+X.
(0,25)X=F[+X]F[-X]+X.
F=FF.
```

FIGURE 3.31 – Le générateur

```
j = 0 motsP = X -> F[++X]F[-X][--X]+Xde probabilite0.5
j = 1 motsP = X -> F[+X]F[-X]+Xde probabilite0.25
j = 2 motsP = X -> F[+X]F[-X]+Xde probabilite0.25
j = 3 motsP = F -> FFde probabilite1
```

FIGURE 3.32 – Contenu du vecteur de probabilités, attribut de la classe arbre.

Explications :

La classe arbre possède ce vecteur de Probabilités nommé "règlesP", listant les probabilités en fonction du motsP correspondant.
Une fonction random lors de la création de l'arbre donne un nombre entre 0 et 1 pour chaque lettre enregistrée dans étageF.
Dans cet exemple, on commence par le X :

- si random renvoie 0.3, (comme 0.3<0.5), c'est la règle correspondant au cas j=0 qui est utilisée ;
- si random renvoie 0.6,(comme 0.6>0.5) on retranche 0.5 à 0.6 pour obtenir 0.1 et on passe à la règle j=1. Comme 0.1<0.5, c'est la règle correspondant au cas j=1 qui est utilisée ;

— si random renvoie 0.8, (comme 0.8>0.5) on retranche 0.5 à 0.8 pour obtenir 0.3 et on passe à la règle j=1. Comme 0.3>0.25, on retranche pour obtenir 0.05.Comme 0.05<0.25, c'est la règle correspondant au cas j=2 qui est utilisée ;
— si random renvoie un nombre supérieur n'ayant pas d'équivalent. C'est la dernière probabilité de la liste correspondant à la lettre qui est renvoyée.

Si aucune probabilité n'est inscrite, la probabilité est automatiquement initialisée à 1.

Arbres correspondant au générateur après 6 itérations :

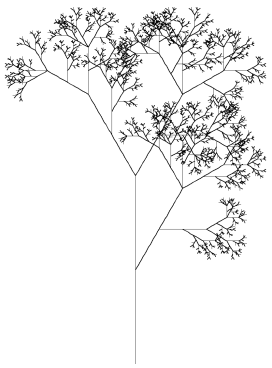


FIGURE 3.33 – Arbre 1

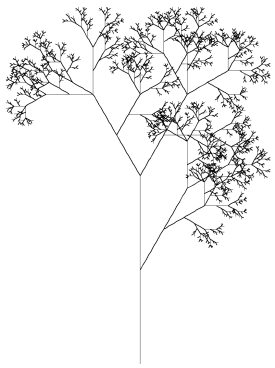


FIGURE 3.34 – Arbre 2

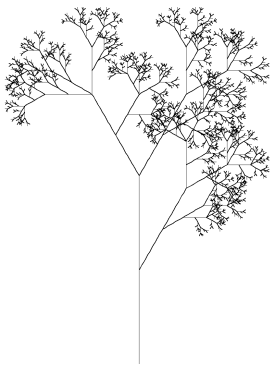


FIGURE 3.35 – Arbre 3

Pour retourner aux explications, cliquez ici.

Annexes sur les contextes liés

En botanique, le système contexte free est appelé acropète [1] ou basipète [2].

En pratique :

```
angleXY=30.
angleZ=0.
largeur=1.
longueur=10.
contexte=F<X=B,AA<A=X.
X.
(0,5)X=F[++X]F[--X]+X.
(0,5)X=F[--X]F[++X]-X.
F=FF.
B=VA[+A][++A][+++A][-A][--A][---A].
A=AA.
```

FIGURE 3.36 – Le générateur

```
_contextes :
l'ancienne regleFX , la nouvelle regles B, la position :2
 l'ancienne regleAAA , la nouvelle regles X, la position :3
```

FIGURE 3.37 – Contenu du vecteur de contextes, attribut de la classe arbre.

Explications :

La classe arbre possède ce vecteur de Contextes nommé "contextes", listant les contextes.

Exemples :	F<X=B	AA<A= X	DA<B>C=FG
Règle Ancienne :	FX	AAA	DABC
Règle Nouvelle :	B	X	FG
Position :	2	3	3
Avant le remplacement :	FX	AAA	DABC
Après le remplacement :	FB	AA X	DAFGC

1. Une croissance acropète qualifie le développement d'organes, que ce soit des feuilles ou de fleurs, qui s'effectue de la base vers le sommet, l'apex. Se dit donc d'un développement se réalisant de la base vers le sommet. Par exemple, la floraison d'un thyrse de Lilas..

2. Basipète qualifie le développement d'organes botaniques comme des feuilles ou fleurs s'effectuant du sommet vers le bas. Se dit d'un développement qui se fait de l'apex vers la base.

Algorithmes :

Data: string "a", chaîne de caractères à modéliser
Result: modifie a en fonction du contexte
a ← rechercheContexte;
Algorithm 3: Amorce de contexte dans createTreeRankByRank

Data: string "a"(chaîne de caractères à modéliser. Il est passé par référence et donc est directement modifiable.), entier "cpt" (pointeur vers la lettre de "a"), le vecteur "contextes" (regroupe les différents contextes.)
Result: vérifie si règle ancienne est facteur de "a". Si c'est le cas, la lettre adéquate est changée par "règle nouvelle".

```
while il existe un contexte non vu dans le vecteur "contextes" do
    if première lettre de règle ancienne equivaut "a[cpt]" then
        cpt++;
        if verifierContextStringVrai then
            insérer dans "a" règle nouvelle;
        end
    end
end
```

Algorithm 4: RechercheContexte

Data: string "a"(chaîne de caractère à modéliser.), entier "cpt" (position de la lettre de "a"), vecteur contextes, les entiers i, j (permettent l'accès à contextes[i][j]).
Result: base passée par référence (si règle ancienne de contextes[i] n'est pas facteur, elle équivaut à -1 sinon elle prend la valeur de la position exacte de la lettre à modifier.)

```
while cpt< taille de "a" do
    if j= taille de règle ancienne then
        retourner Vrai;
    end
    if a[cpt] est une lettre then
        if règle Ancienne [j] équivaut à a[cpt] then
            modifier base en fonction de "position";
            j←j+1;
            cpt←cpt+1;
        else
            retourner Faux;
        end
    else
        if a[cpt] équivaut à "[" then
            cpt←cpt+1;
            if verifierContextString vrai then
                retourner Vrai ;
            else
                recherche le "]" correspondant à ce "]" et modifier cpt en
                 fonction ;
            end
        else
            if a[cpt équivaut à "]" then
                retourner Faux ;
            else
                cpt←cpt+1;
            end
        end
    end
    retourner Faux ;
end
```

Algorithm 5: L'algorithme VérifierContextString

Pour retourner aux explications, cliquez ici.

L'exportation d'arbres à tronc variable

Afin de créer un tronc de largeur spécifié, pour chaque noeud exporté de l'arbre nous calculons quatre points autour et à distance égale. Or pour obtenir ces points, il est nécéssaire de procéder par étapes :

1. calculer un nouveau point p1 entre le noeud et un de ses fils.
2. à partir de p1, calculer les normes entre
   — ce point et le noeud à traiter ;
   — ce point et son père.
3. avec des deux normes, déduire deux points à une distance unitaire du noeud courant ;
4. chercher le produit vectoriel entre ces nouveaux points.

Les deux points à distance unitaire et les coordonnées du produit vectoriel, permettent de donner quatre nouveaux points autour du noeud en cours de traitement car ils se situent sur un plan bissecteur [3] de l'angle formé par le père de ce noeud et le fils choisi.

Le problème de torsions des troncs est évité par le calcul des normes pour un des points de ce noeud avec les quatre points du tronc du noeud précédant. La norme la plus petite definit alors le premier point du tronc à exporter.

Pour retourner aux explications, cliquez ici.

3. qui partage en deux parties égales.

Table des figures

1.1 Koch 0 4
1.2 Koch 1 4
1.3 Koch 2 4
1.4 Générateur 6
1.5 aperçu de l'add-on dans Blender 6

2.1 BackTrack 8
2.2 Arbre inversé 8
2.3 Arbre final 8
2.4 Flocon de Koch 8
2.5 i=1 Flocon 8
2.6 i=2 Flocon 8
2.7 Générateur 8
2.8 Générateur 9
2.9 i = 7 : Le premier arbre 9
2.10 i = 7 : Le second arbre 9
2.11 Pissenlit 10
2.12 Saule pleureur 10
2.13 Arbre à fleurs 10
2.14 point en coordonnées sphériques 11

3.1 Courbe de Koch après 7 itérations 14
3.2 Générateur 14
3.3 Courbe de Sierpiński après 8 itérations 15
3.4 Générateur 15
3.5 Fractale après 4 itérations 15
3.6 Générateur 15
3.7 Générateur 15
3.8 initialisation des angles XY et Z 16
3.9 Générateur 17
3.10 i=1 Arbre DOL 17
3.11 i=4 Arbre DOL 17
3.12 Arbre inversé 17
3.13 BackTrack 17
3.14 Arbre final 17
3.15 Triangle 19
3.16 i=1 Triangle 19
3.17 i=2 Triangle 19
3.18 Générateur 19
3.19 Hexagones 19
3.20 i=1 Hexagones 19
3.21 i=2 Hexagones 19
3.22 Générateur 19
3.23 i=0 Fractale 20

3.24 i=1 Fractale 20
3.25 i=2 Fractale 20
3.26 i=3 Fractale 20
3.27 i=5 Fractale 20
3.28 i=7 Fractale 20
3.29 i=9 Fractale 20
3.30 Générateur 20
3.31 Le générateur 20
3.32 Contenu du vecteur de probabilités, attribut de la classe arbre. . 20
3.33 Arbre 1 21
3.34 Arbre 2 21
3.35 Arbre 3 21
3.36 Le générateur 22
3.37 Contenu du vecteur de contextes, attribut de la classe arbre. . . 22