Systèmes de Lindenmayer

Valvassori Moïse

27 juin 2001

Résumé

Il existe un programme en langage SCHEME qui génère des plantes sur des ordinateurs. En voici une documentation. Au passage, j'espère donner quelques rappels sur les systèmes de Lindenmayer.

Table des matières

1 L-Systèmes

Les L-Systems ont été inventé en 1968 par Aristid Lindenmayer.

Il existe plusieurs classes de L-system, les plus simples sont les systèmes déterministes de contexte libre (DOL).

Formellement, un L-system est composé de trois éléments :

– Un alphabet :Σ
– Un mot initial : w
– D'un ensemble de règles de productions : P

Les règles de productions définissent les transformations que va subir le système.

1.1 DOL System

Les DOL sont les

Exemple du flocon de Koch :

$$w : F--F--F$$
$$F \longrightarrow F+F--F+F$$

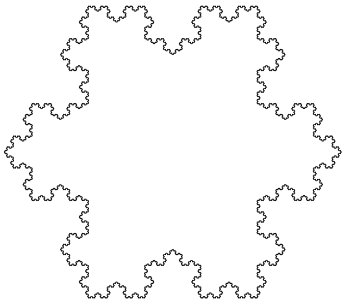

FIG. 1 – Flocon de Koch.

Autre exemple :

$$w : F$$
$$F \longrightarrow FF-[-F+F+F]+[+F-F-F]$$

(voir figure 2 sur la page suivante)

2 Commandes

2.1 Aides

Ce programme contient une aide en ligne. La commande pour voir l'aide est :

```
(lsys-help 'key)
```

`key` est le nom de la rubrique que l'on souhaite consulter.

```
(lsys-help 'list)
```

donne la liste des rubriques disponibles.

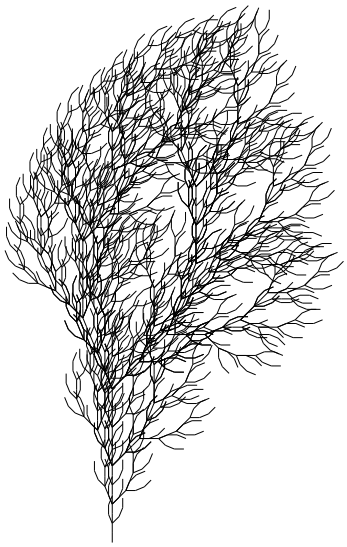

FIG. 2 – Un arbre.

2.2 Format de description

Les systèmes sont décrits dans une liste d'association. Les champs de cette liste sont :

`type`
: le type du système.

  `dol`
  : Système déterministe contexte libre.

  `sol`
  : Système stochastique contexte libre

`name`
: Le nom du L-system

`author`
: l'auteur du L-system

`description`
: une descriton du système

`w`
: Mot initial

`prod`
: Liste d'association qui contient les règles de productions.
  La syntaxe de ces regles est varialbe selon le type du système.

`angle`
: Incrément de l'angle

Exemple :

```
(define adolsys '((type dol)
  (name "A simple L-System")
  (author "Valvassori Moïse (typed)")
```

```
  (description "The L-sys given in page 3 of\n  \"The Algoritmic beauty of
  (w (b))
  (prod ((a (a + b))
     (b (a))))))
```

2.3 Génération des L-Systèmes

`(lsys-gen system n)`

sort la chaîne à la génération `n`

`(lsys-generate-turtle-string lsys n turtle)`

génère l'arbre.

2.4 Tortue

Les L-systems ne sont qu'un systéme formel. Pour visualiser les arbres, on a besoin de donner un sens aux symboles et d'interpréter cette sémantique.

2.4.1 Table d'interface

Pour donner un sens à un L-system, on utilise une table de correspondance entre un symbole et son sens. Par exemple, le symbole `F` signifie «*avance*».

Pour faire le lien entre ce symbole et son sens, on utilise une liste d'association symbole-sémantique.

Voici un exemple d'une table d'association pour un L-System en deux dimensions.

```
(define lsys-turtle-ass '((F (turtle 'move-forward))
  (f (turtle 'jump-forward))
  (+ (turtle 'turn-right))
  (- (turtle 'turn-left))
  ([ (turtle 'push))
  (] (turtle 'pop))))
```

2.4.2 Description d'une tortue

ToDO Structure de description de la tortue :

`lsys2turtle`

lambda d'entrée de la tortue

ass

Liste d'association des symboles (voir 2.4.1)

...

Exemple :

```
(define simpleturtle '((lsys2turtle ,turtle-string-to-turtle)
       (ass ,lsys-turtle-ass)
       (constructor make-turtle)
       (segment-length 5)
       (segment-width 0.1)
       (initial-angle 0)
       (output-file "/tmp/a.ps")
       ))
```

2.4.3 Interface de la tortue

Une tortue est une «*lambda*» qui recoit des commandes graphiques et les interprètes. On peut imaginer des tortues qui trace les graphiques sur une écran, d'autres qui génère directement les images.

Exemple d'un squelette de tortue :

```
(define (make-turtle angle-inc line-length)
  (let ((turtle-x 100)
(turtle-y 300)
(turtle-angle (/ PI 2))
(turtle-segment-length line-length)
(turtle-angle-step angle-inc)
(stack '())
(out (open-output-file "/tmp/toto.ps")))
       ;;;; la valeur de retour  ;;;;
    (lambda (mesg)
      (case mesg
((move-forward)
 (turtle-move-forward))
((jump-forward)
 (turtle-jump-forward))
((turn-left)
 (turtle-turn-left))
((turn-right)
 (turtle-turn-right))
((push)
 (turtle-push))
((pop)
 (turtle-pop))
((close)
 (close-output-port out)))
      )))
```

2.4.4 Tortue 3D

ToDO Il est peut être possible de calculer de façon différencielle les coordonnées du vecteur déplacement aprés chaque rotation :

$$\frac{dR_U(\alpha)}{d\alpha} = \begin{pmatrix} x(\cos\alpha - 1) - y\sin\alpha \\ x\sin\alpha + y(\cos\alpha - 1) \\ 0 \end{pmatrix}$$

$$\frac{dR_L(\alpha)}{d\alpha} = \begin{pmatrix} z\sin\alpha + x(\cos\alpha - 1) \\ 0 \\ z(\cos\alpha - 1) - x\sin\alpha \end{pmatrix}$$

$$\frac{dR_H(\alpha)}{d\alpha} = \begin{pmatrix} 0 \\ z\sin\alpha + y(\cos\alpha - 1) \\ z(\cos\alpha - 1) - y\sin\alpha \end{pmatrix}$$

$$\frac{\partial R_U(\alpha)}{\partial\alpha} = \begin{pmatrix} \alpha x\cos\alpha - \alpha y\sin\alpha \\ \alpha y\cos\alpha + \alpha x\sin\alpha \\ \alpha z \end{pmatrix}$$

$$\frac{\partial R_L(\alpha)}{\partial\alpha} = \begin{pmatrix} \alpha x\cos\alpha + \alpha z\sin\alpha \\ \alpha y \\ \alpha z\cos\alpha - \alpha x\sin\alpha \end{pmatrix}$$

$$\frac{\partial R_H(\alpha)}{\partial\alpha} = \begin{pmatrix} \alpha x \\ \alpha y\cos\alpha + \alpha z\sin\alpha \\ \alpha z\cos\alpha - \alpha y\sin\alpha \end{pmatrix}$$

⚠ À vérifier...