

Démonstrateur de

L-Systèmes

Réalisé par :

Rémi DUCCESCHI et Thomas NOGUER

Proposé et encadré par :

Sébastien AUPETIT

Polytech Tours 2012-2013

Rapport de projet de réalité virtuelle

Introduction Plan

[I. Description L-Système 5](#_Toc354567055)

[Description générale 5](#_Toc354567056)

[Déterminisme 5](#_Toc354567057)

[Dépendance du contexte 5](#_Toc354567058)

[II. Parser et générateur 6](#_Toc354567059)

[les fichiers de règles 6](#_Toc354567060)

[la génération des symboles 6](#_Toc354567061)

[III. Tortue 7](#_Toc354567062)

[Généralités 7](#_Toc354567063)

[Tube 7](#_Toc354567064)

[Tree 8](#_Toc354567065)

[Avancées 9](#_Toc354567066)

[IV. Configuration de moteur 3D 11](#_Toc354567067)

[lumière 11](#_Toc354567068)

[caméra 11](#_Toc354567069)

[inputs 12](#_Toc354567070)

[V. Archi/Evolutivité 12](#_Toc354567071)

[modèle MVC 12](#_Toc354567072)

[M- Grammaires 12](#_Toc354567073)

[V - GUI-JME 12](#_Toc354567074)

[C – Controller 12](#_Toc354567075)

[Nouvelles Interprétation 12](#_Toc354567076)

[Gestion des paramètres 13](#_Toc354567077)

[VI. Gestion de projet 14](#_Toc354567078)

[Git 14](#_Toc354567079)

[Gantt 14](#_Toc354567080)

# Description L-Système

Notre projet repose sur la visualisation d’interprétations de L-Systèmes. Dans cette partie, nous allons décrire ce qu’est une grammaire L-Système.

## Description générale

Les grammaires L-Systèmes sont des spécialisations des grammaires formelles. Elles ont été inventées par le biologiste hongrois Aristid Lindenmayer (et sont parfois appelées Lindenmayer systèmes). Ces grammaires ont la particularité d’utiliser un développement parallèle lors de l’application de règles sur un mot : lors de l’utilisation en production, on utilise à chaque étape l’ensemble des règles applicables sur le mot alors que dans les grammaires de Chomsky, on n’applique qu’une règle par étape. Cela permet de simuler des comportements complexes que l’on peut retrouver dans la nature, comme la division cellulaire (voir ci-dessous) ou la pousse d’arbres.

Si l’on veut simuler la division cellulaire à l’aide d’une telle grammaire, on pourrait partir d’un axiome "O" (qui représente une cellule) et utiliser une règle qui change tous les "O" en "OO" (modélisant ainsi la division d'une cellule). On obtiendrait alors :

* Étape 1 : O
* Étape 2 : OO
* Étape 3 : OOOO
* Étape 4 : OOOOOOOO
* …

Cette modélisation se comprend facilement juste en lisant la production de la grammaire. Cela n'est pas le cas de la simulation de pousse d'arbre qui nécessite une interprétation spécifique de la grammaire (appelée interprétation de la tortue) qui permet de représenter les symboles produits dans une visualisation compréhensible par l'homme. C'est le but de notre projet.

De par leurs spécificités, les grammaires L-Systèmes sont généralement utilisées en production, à l'inverse des grammaires de Chomsky qui sont plutôt utilisées en vérification. L'application des L-Systèmes dans la réalité virtuelle permet de générer des environnements réalistes et détaillés en insérant des arbres générés à l'aide d'une grammaire. L'intérêt étant ici de disposer d'une grande quantité d'arbres différents sans pour autant avoir à modéliser tous les arbres directement. Ceux-ci seront automatiquement générés.

Afin de pouvoir utiliser pleinement une grammaire L-Système, il faut pouvoir la définir. Une telle grammaire est composée de :

* Un ensemble de symboles utilisables (appelé "V") qui seront tous les symboles pouvant apparaître dans les mots produits par la grammaire. Cet ensemble est généralement coupé en deux sous-parties :
  + L'ensemble des symboles constants qui ne peuvent pas être modifiés par une règle de réécriture
  + L'ensemble des symboles modifiables qui ont au moins une règle de réécriture qui leur sont associée.
* Un axiome qui comporte un unique symbole faisant partie des symboles utilisables (généralement constant)
* Une liste de règles de réécritures qui permet de définir l'évolution des mots lors de l'application de ces règles sur l'axiome.
* Éventuellement un angle qui définit l'angle entre les branches lors de l'interprétation de la tortue.

Une règle de réécriture est composée de deux parties :

* La partie gauche contient le symbole qui sera modifié par l'application de la règle ;
* La partie droite contient l'ensemble des symboles qui viendront remplacer celui de la partie gauche.

En plus de cela, le résultat de la production d'une grammaire dépend beaucoup du nombre d'itération que l'on souhaite appliquer. Selon le nombre d'étapes, le mot final pourra être plus ou moins long et complexe, et donc l'arbre plus ou moins détaillé.

Une règle ne peut modifier qu'un seul symbole. La partie gauche ne doit donc contenir qu'un seul symbole à modifier. Il est possible de supprimer un symbole en mettant en partie droite le symbole "ε".

Il existe plusieurs manières d'appliquer une règle à un symbole, et selon la manière, on peut modifier le type de la grammaire. Il existe deux catégories de grammaires : le déterminisme et la dépendance du contexte. Tous les L-Systèmes apparaissent dans ces deux catégories. Ainsi, une grammaire peut être déterministe et indépendante du contexte, ou déterministe et dépendante du contexte… On identifie ainsi 4 types différents de grammaires :

* DOL : déterministe et indépendante du contexte ;
* SOL : stochastique et indépendante du contexte ;
* DIL : déterministe et dépendante du contexte ;
* SIL : stochastique et dépendante du contexte.

## Déterminisme

Le déterminisme permet de créer des grammaires qui seront constantes dans la production à partir d'un axiome donné, ou au contraire qui pourront générer différents résultats avec la même configuration de base. Les grammaires stochastiques permettent ainsi de générer des arbres différents à chaque fois, sans pour autant changer de grammaires. Cela est particulièrement pratique dans un jeu vidéo par exemple, ou un grand nombre d'arbres peuvent être générés lors de la création d'un niveau.

Dans les grammaires déterministes, il ne peut y avoir au maximum qu'une seule règle de réécriture pour un symbole donné. Ainsi, à chaque fois que le symbole est rencontré, il est toujours remplacé par la même séquence. Une grammaire déterministe est noté "D-L"

## Dépendance du contexte

# Parser et générateur

## les fichiers de règles

## la génération des symboles

# Tortue

## Généralités

Les interprétations de la tortue peuvent être très variées et différentes les unes des autres. Il est possible de mettre en place une interprétation de la tortue pour dessiner de simples lignes ou bien une interprétation avec un personnage qui effectue différents pas de danse selon les symboles à interpréter. Il est possible de tout faire.

Cependant, toutes les interprétations de la tortue ont des points communs. En effet, elles sont toutes des interprétations et cherchent à représenter d'une manière ou d'une autre une chaîne de symboles. C'est pour cette raison et d'autres qu'une tortue mère a été mise en place. Chaque interprétation de la tortue existante et future se doit d'hériter de cette classe mère qui regroupe toutes les caractéristiques communes des interprétations de la tortue. En plus de la liste des symboles et des mécanismes qui tournent autour, les tortues mises en place ont une construction commune du graphe de scène. Après la création d'une entité dans la scène, on rajoute celle-ci dans le graphe de scène afin de permettre au moteur 3D de l’afficher à l’écran. Pour les deux interprétations de la tortues existantes (Tube, Tree) le graphe de scène est construit de cette façon : lorsque l'on ajoute un tube ou une branche dans le graphe de scène, on créé un nouveau nœud qui se situe au bout de l'entité ajoutée. De cette façon, le prochain objet qui sera ajouté le sera au bout de l'objet précédent. Ce système permet aussi de mettre en place le système de sauvegarde de position et de restauration. Lorsque l'on souhaite sauvegarder la position courante on sauvegarde le nœud courant dans une pile. Pour restaurer la position il suffit de définir le nœud courant comme étant le premier nœud de la pile de sauvegarde.

Néanmoins, cette architecturation du graphe de scène, bien que commune aux deux interprétations existantes, peut ne pas convenir à toutes les interprétations possible. Il faudra modifier ce système et en ajouter une autre le cas échéant.

## Tube

TubeTurle est l'interprétation de la tortue de base. Elle est capable de représenter toutes les listes de symboles qui ne contiennent que les symboles de base. Elle est très simple et permet un rapide coup d'œil du résultat de la grammaire. Cette interprétation n'est capable de dessiner que des cylindres lorsqu'elle rencontre un symbole FORWARD, tous les autres symboles ne font que modifier la position ou la direction de dessin des prochains cylindres. Par conséquent, si une chaîne ne contient aucun symbole FORWARD, il n'y aura aucun résultat visible à l'écran.

Cette interprétation est historiquement la première et aussi la plus simple des deux. Elle a été conçue pour permettre une représentation la plus simple possible. Les objets dessinés sont donc des lignes 3D : des cylindres ou tubes. La conception de cette interprétation a permis de mettre en avant l'architecturation du graphe de scène expliqué plus haut. L'interprétation de la tortue est capable de réaliser les actions suivantes selon les symboles à interpréter :

* Dessiner un tube à la position courante
* Tourner d'un angle fixe selon un des 3 axes
* Sauvegarder la position courante
* Restaurer la dernière position sauvée

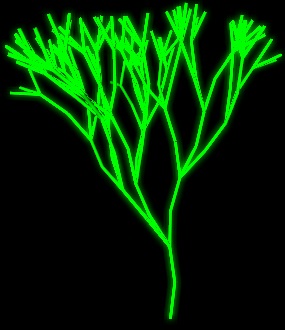
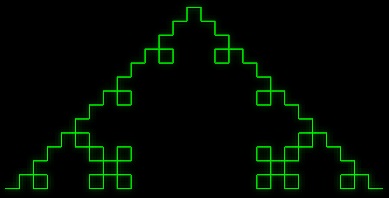
Pour réaliser l'action de dessin d'un tube il suffit de créer un nouvel objet et de l'attacher au nœud courant. Il faut aussi créer un nouveau nœud fils et l'attacher au nœud courant en prenant soin de le déplacer de la taille du tube. De cette façon, le nœud fils est positionné au bout du tube dessiné. Le nœud fils devient le nouveau nœud courant.

Pour réaliser l'action de rotation selon un axe il suffit d'effectuer une rotation du nœud courant selon l'axe souhaité.

La sauvegarde la position courante utilise une pile de nœud. A chaque appel de l'action de sauvegarde on ajoute le nœud courant dans la pile de sauvegarde.

L'action de restauration de la position utilise la même pile de nœud que l'action de sauvegarde. On dépile le dernier nœud ajouté dans la pile et on le considère comme nouveau nœud courant.

Ces simples actions permettent d'obtenir des rendus variés :



## Tree

L’interprétation de la tortue Tree Turtle utilise les mêmes mécanismes que décris précédemment pour Tube Turtle. Elle rajoute de nouvelles spécificités liées à la représentation d’arbres. Le tronc d’un arbre est plus mince et long que ses branches. Afin de simuler ce comportement deux paramètres ont été rajoutés :

* Width reduction
* Length reduction

Ces deux paramètres sont respectivement les pourcentages de réductions de la longueur et de l’épaisseur d’un tube fils par rapport à son père. Un tube est fils de son père lorsqu’il est situé après celui-ci.

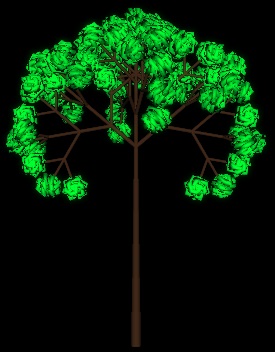
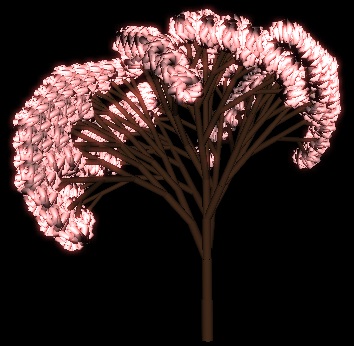
Sur l’image ci-dessus on peut voir une branche père (à gauche) avec sa branche fille (à droite). La branche fille est moins large et longue que sa branche père.

Cette interprétation est aussi capable de représenter le symbole feuille (interprétation 21). Les feuilles sont des représentés avec des tords afin de simuler un amas de feuilles.

Contrairement aux tubes, lorsqu’une feuille est ajoutée dans le graphe de scène on choisit de ne pas déplacer le nœud fils. En effet, un arbre possède des feuilles sur des branches uniquement et ne peut pas avoir de branches qui poussent sur des feuilles.

Ci-dessous le même screenshot que précédemment avec une branche père et une branche fils avec en plus une feuille.

Cette nouvelle interprétation permet d’obtenir des résultats d’arbres intéressant :



## Avancées

Une évolution importante et particulièrement intéressante de l’application actuelle serait de pouvoir utiliser des meshs à la place des objets de base qui sont utilisés actuellement. Deux possibilités sont envisageables. La première serait de permettre à l’utilisateur d’utiliser ses propres meshs en les plaçant dans un dossier par exemple. Cette solution est la plus compliquée à mettre en place mais permettrait d’avoir un rendu très varié et de pouvoir, par exemple, représenter tout type d’arbre avec l’interprétation Tree Turtle.

La deuxième possibilité serait de proposer un échantillon de meshs différents à l’utilisateur. Cette solution serait plus simple à mettre en place que la première sous réserve de trouver un bon échantillon de meshs.

Enfin, l’idéale serait d’avoir les deux. Un échantillon de meshs à proposer à l’utilisateur et la possibilité d’ajouter ses propres meshs dans la liste.

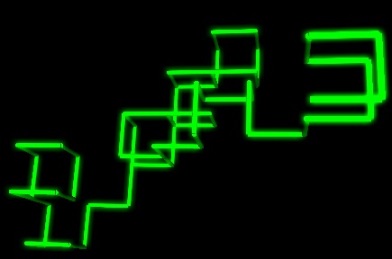
*Une autre évolution de l’application serait d’un niveau plus technique. Lorsque l’on choisit d’afficher une chaîne très longue, dans le cas d’un nombre d’itération important ou d’une grammaire très verbeuse,*

*Pb de nombres d’objets*

# Configuration de moteur 3D

## lumière

La lumière est une composante indispensable d’une scène 3D, elle permet de mettre en avant les reliefs et de voir les objets tout simplement.

 Le cas qui nous intéresse est assez spécifique. Nous ne connaissons pas à l’avance la disposition des objets et leur nombre. Il faut positionner un système de lumière qui permette d’éclairer la scène comme nécessaire dans toutes les situations. C’est avec ces contraintes à l’esprit que nous avons décidé de positionner une lumière directionnelle dans la direction de la caméra. Cette direction est mise à jour à chaque fois que la caméra se déplace. De cette façon la scène est toujours éclairée selon la position de la caméra et peu importe sa position. Même si la caméra est très éloignée des objets de la scène, ils seront éclairés par la lumière si la caméra regarde ces objets.

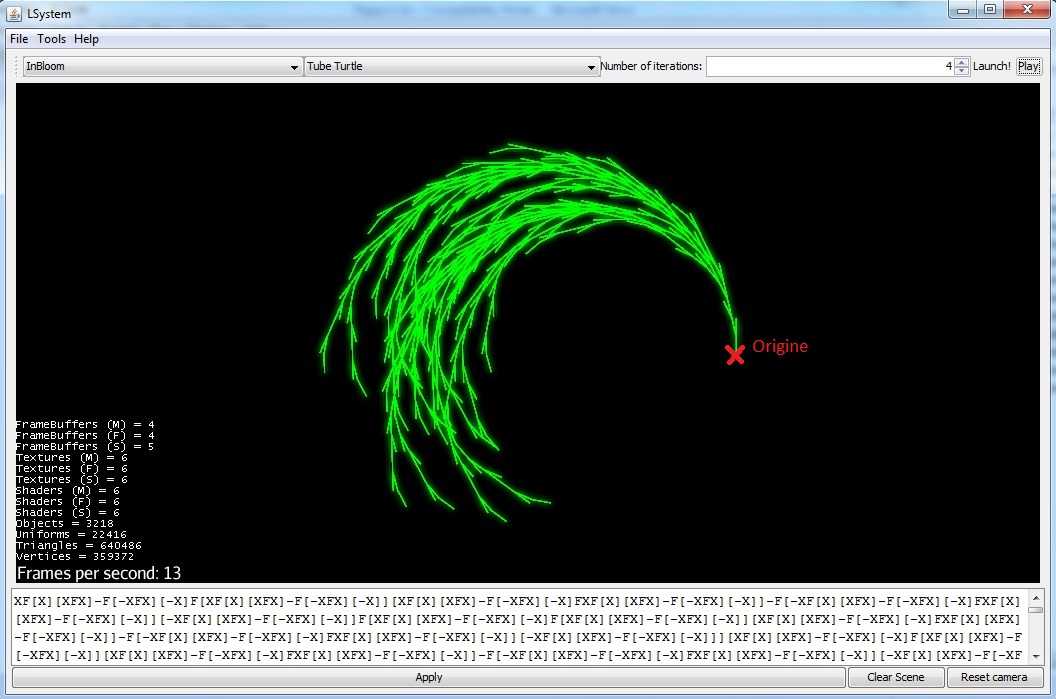
## caméra

Le problème de la position initiale de la caméra est aussi lié aux mêmes contraintes que celles décrites pour la lumière. On ne connait pas le nombre d’objets et leur positionnement avant qu’ils soient affichés à l’écran.

La solution adoptée ici est différente. Lorsque l’on dessine un objet à l’écran on regarde si cet objet dépasse le maximum ou le minimum de chaque axe de notre repère. De cette façon, à la fin de la création de la scène nous disposons des coordonnées maximum et minimum sur chaque axe des objets de la scène. Il est très simple avec ces données de calculer le point central de la scène et de centrer la caméra sur ce point.

Il reste néanmoins encore un problème, il faut reculer suffisamment la caméra par rapport à ce point central afin de voir toute la scène à l’écran. Ce problème a été résolu de façon presque empirique, nous prenons le maximum entre la longueur et la largeur de l’écran et on recule la caméra en multipliant cette valeur par un facteur déterminé par expérimentations. Cette solution n’est pas parfaite mais fonctionne avec l’application actuelle.

Ci-dessus une capture écran montrant le positionnement initial de la caméra avec une forme qui prend son origine sur la croix rouge. La caméra était initialement centrée sur ce point.



## inputs

# Archi/Evolutivité

## modèle MVC

### M- Grammaires

### V - GUI-JME

### C – Controller

## Nouvelles Interprétation

La création de nouvelles interprétations de la tortue est relativement simple à faire comme dit précédemment. Il y a quelques points importants cependant à ne pas omettre :

* Il faut rajouter un attribut static int dans la classe Turtle pour chaque nouvelle interprétation de la tortue ajoutée. Il suffit de copier une ligne déjà présente pour une autre interprétation et de prendre un entier non utilisé.
* La classe de la nouvelle interprétation doit hériter de la classe Turtle et redéfinir les fonctions drawScene, checkSymbols, initParameters et setParameters.
* Afin de permettre à la caméra de se positionner correctement il est important d’appeler la fonction updateBoundsCoordinates après l’ajout d’un objet dans la scène dans la fonction drawScene avec les coordonnées de cet objet.

### Gestion des paramètres

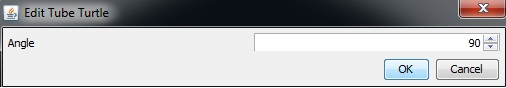
Les paramètres sont une composante importante d’une interprétation de la tortue. On veut, par exemple, pouvoir changer la couleur des tubes, leur longueur, leur largeur pour l’interprétation Tube. Dans une optique d’évolutivité de l’application des paramètres génériques ont été mis en place. Ainsi si une nouvelle interprétation de la tortue est amenée à être mise en place, l’intégration de ses paramètres sera le plus simple possible. Afin d’ajouter un paramètre il suffit d’ajouter deux lignes dans les fonctions initParameters et setParameters de la classe de l’interprétation de la tortue.

Par exemple si l’on veut ajouter un paramètre de type entier pour l’angle il suffit d’ajouter ces lignes :

**parameters.add(new Parameter("Angle", ParameterType.TYPE\_INTEGER, new Integer((int) angle)));**

angle = ((Integer) parameters.get(0).getValue()).floatValue();

La première ligne permet de créer un nouveau paramètre ayant pour nom Angle, de type entier et dont la valeur initiale est celle de la variable angle.

La deuxième ligne permet de récupérer la valeur du paramètre et de l’affecter à la variable angle. Ainsi la valeur sera disponible dans la classe.

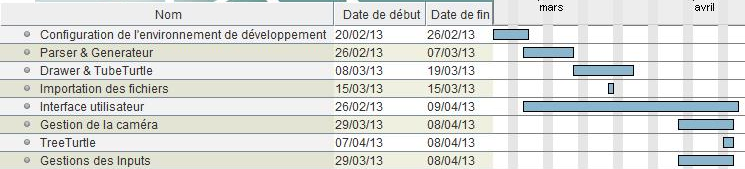
Une fenêtre de paramètres est créée automatiquement. Dans notre cas la fenêtre donne le rendu suivant :

# Gestion de projet

## Git

Afin de gérer nos sources nous avons naturellement choisi d’utiliser Git. Il est simple d’utilisation et nous est familier. De plus, l’utilisation de Git permet également d’utiliser le dépôt GitHub qui est très répandu et permet une distribution de l’application gratuitement et rapidement. Notre application est ainsi disponible et libre d’accès à tous.

## Gantt



Ce diagramme de Gantt présente les fonctionnalités principales qui ont été réalisées au cours de ce projet. Il s’agit d’un diagramme construit avec les dates des commits du dépôt Git.

Conclusion

Annexes