VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

ENGR489 Project: Preliminary Report

Bryony Gatehouse

Supervisor: Zohar Levi

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

This project creates a game environment that shows CO_2 levels when a tree is added to a scene. A tree model based on a state of the art paper will be used to supply high quality realism.

Contents

Chapter 1

Introduction

Trees are critical to sustaining life on earth, providing the oxygen we breathe. They also supply materials that have been incorporated into our everyday lives. Apart from these functions, they are also an inherent part of a scene and thus are important when creating a realistic scene for a movie, game or urban planning.

This project's objective is to create a 3D game that emphasises the importance of trees in maintaining the O_2 levels in the environment. The tree model will be based on a state of the art paper that provides both a technological and theoretical challenge. Unfortunately, recent work describes their method at a high level, without going into the low-level details. One challenge of this project is to bridge, and implement, the gap between a classic L-system [1] that generates a simple tree structure and a state of the art realistic tree model [2]. The former is typically used as a foundation for recent work.

1.1 Overview of Project

The focus of this project will be implementing the paper [2] to create a realistic tree model which can be used in a 3D environment. Work will also be done on the CO_2 generation of a tree, based off the generated tree model.

Work done in Trimester 1 included exploring the game engine Godot, as a popular game engine with a shallow learning curve, and closely following the steps for building an L-system from [1] and implementing their examples. Trimester 2 will focus on implementing [2] and incorporating the tree model into a game environment.

The project will be evaluated against the results demonstrated in the paper [2].

Project Objectives:

- Implement an L-system to generate a tree model, following [1].
- Extend the program, following the paper [2], to generate a more realistic tree model.
- Introduce an equation to calculate the CO_2 absorption of the tree model.
- Create a game environment with the CO_2 level changing when a tree model is added.

Chapter 2

Background

This project will focus on creating a realistic tree based on a state of the art paper. [2] extends the basic L-system tree generator, so an understanding of the L-system is important.

This chapter will explain the steps of tree generation using the L-system and include some extra material on making the models more realistic. There will also be a short introduction to [2].

2.1 Introduction to Modeling Trees with the L-system

2.1.1 L-system

The Lindenmayer system - or L-system for short - was originally designed as a mathematical theory to model plant geometry [1]. The L-system is based on the idea of rewriting. Given a list of rules or actions, it replaces - or rewrites - certain sections with new rules, extending the list.

Given a string (or word) built from the two letters a and b, where the letters can occur multiple times in a string, rewriting rules can be attached to each letter. The rule $a \rightarrow ab$ states that every a must be replaced with the string ab, and the rule $b \rightarrow a$ states that every b will be replaced with an a. Note that, if an a is replaced with an ab, the b is not then replaced with an a.

The table below shows the progression of the string b using the predefined rules. The third row shows the second rewriting step, where the letter a is rewritten as ab and the letter b is rewritten as a creating the string aba.

b	a				
a	ab				
ab	ab	a			
aba	ab	a	ab		
$abaab$	ab	a	ab	ab	a

2.1.2 Turtle interpretation of strings

The letters of the L-system could be used to represent geometric structures [1]. A turtle sitting at position (x, y) and facing direction α can be controlled by a string of letters and symbols generated using the L-system. Its position and direction (or angle) is the *state* of the turtle. The step length d and the angle increment δ should be decided beforehand. The turtle can then respond to the following symbols and letters by changing its state, and thus moving around:

F — Move forwards by the length d to $x' = x + d\cos\alpha$ and $y' = y + d\sin\alpha$. A line is drawn between the original position and the new position.

f — Move forwards by the length d without drawing a line.

$+$ — Turn left by angle δ, changing the state to $(x, y, \alpha + \delta)$.

$-$ — Turn right by angle δ, changing the state to $(x, y, \alpha - \delta)$.

2.1.3 Branching structures

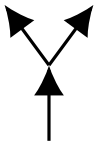
A branch can be represented using a directed edge [1]. A tree is made up of branches, or directed edges, and nodes. The base node of the structure is the root of the tree. Each node following can be a branching node with two branches continuing from it. A terminal node, at the end of the branches, has no following branches.

For this to be represented in an L-system, a system to return to a prior state must be implemented. This is represented by strings with brackets:

[— Push the current state to a stack.

] — Pop a state from the stack and make it the current state.

An example is the string *F[+F][-F]*. The turtle moves forwards on space, making the trunk of the tree. This state is saved to a stack then a branch is generated leading to the left of the trunk. The saved state, the node at the top of the trunk, is popped from the stack to replace the current state. Note that no line is drawn between these two states. This state is saved again, and a branch is generated leaning towards the right of the trunk.



$F[+F][-F]$

2.1.4 3D L-systems

Adapting the L-system to 3D can be achieved by representing the turtle's orientation in space as three vectors: $\left[\vec{H} \quad \vec{L} \quad \vec{U}\right]$ [1].

The original L-system rotated the turtle around the z-axis, so it stays flat against the x- and y-axis. The vector U represents this rotation and continues to use the $+$ and $-$ symbols to represent its rotation. The x-axis rotation is represented by the vector L and the vector H is the rotation around the y-axis. Thus, the rotations can be represented by the traditional rotation matrices. Using these matrices, the following commands can be used in code to control the turtle's orientation in space:

$+$ — Turn left by angle δ, using rotational matrix $R_U(\delta)$

$-$ — Turn right by angle δ, using rotational matrix $R_U(-\delta)$

& — Pitch down by angle δ, using rotational matrix $R_L(\delta)$

$\wedge$ — Pitch down by angle δ, using rotational matrix $R_L(-\delta)$

\ — Roll left by angle δ, using rotational matrix $R_H(\delta)$

/ — Roll right by angle δ, using rotational matrix $R_H(-\delta)$

| — Turn around, using rotational matrix $R_U(180^\circ)$

2.1.5 Stochastic L-system

Unfortunately, all plants created using the normal L-system are identical. Variation can be introduced by including probabilities into each rewriting step. This method is called the Stochastic L-system [1].

For a Stochastic L-system, a letter can have multiple rewriting rules, each with a probability of occurring. The probabilities for a letter should add to 1, asserting that a rule is picked for each rewriting step. As seen in the table below, the string F has $\frac{1}{3}$ probability of being rewritten as any of the three rules: p_1, p_2, or p_3.

$$
\begin{aligned}
w: &\quad F \\
p_1: &\quad F \xrightarrow{.33} F[+F]F[-F]F \\
p_2: &\quad F \xrightarrow{.33} F[+F]F \\
p_3: &\quad F \xrightarrow{.34} F[-F]F
\end{aligned}
$$

2.1.6 Context-sensitive L-system

A different extension to the basic L-system is the Context-sensitive L-system [1]. Instead of having rewriting rules based on a single letter in the string, the Context-sensitive L-system considers the surrounding letters. The rule $0 < 0 > 1 \rightarrow 1[1]$ states that *1[1]* will replace *0* only if it's neighbours are *0* to the left and *1* to the right. An example of a table of rules is below.

It is assumed that there are invisible *1*s at the end of every string, so the string *0* will use the rule $101 \rightarrow 11$. With a bracketed system, the brackets are considered invisible to letters outside the brackets. Thus, a string like *1[0]* will use the rule $111 \rightarrow 0$ for the *1*, ignoring the brackets, and the bracketed *0* will use the rule $101 \rightarrow 11$ using the invisible *1* at the end, and the visible *1* before the brackets [3].

This method allows a tree to be grown using the context of the nodes. Using the table of rules from below, the following table can be generated, showing the first eleven rewriting steps for the string *1*:

$0 < 0 > 0$	$\rightarrow$	0
$0 < 0 > 1$	$\rightarrow$	1[1]
$0 < 1 > 0$	$\rightarrow$	1
$0 < 1 > 1$	$\rightarrow$	1
$1 < 0 > 0$	$\rightarrow$	0
$1 < 0 > 1$	$\rightarrow$	11
$1 < 1 > 0$	$\rightarrow$	1
$1 < 1 > 1$	$\rightarrow$	0

n	L(n)
0	1
1	0
2	11
3	00
4	01[1]
5	111[0]
6	000[11]
7	001[1][10]
8	01[1]1[0][111]
9	111[0]0[11][000]
10	001[11]11[10][001[1]]

When working with a turtle interpretation, specific letters are ignored when checking the context. Typically F, $+$ and $-$, with extra rules being included to reverse the angle changes each rewriting step:

$$* < - > * \rightarrow +$$
$$* < + > * \rightarrow -$$

The $*$ character can represent any letter. Thus the rules mean: and $+$ should be replaced with an $-$, and any $-$ replaced with an $+$.

2.2 Introduction to Self-organising Tree Models

The modeling method described in the paper [2] is split into five parts:

1. The bud's growth direction and ability to grow is decided using the availability or quality of the space surrounding it. Space Colonization presents a bud with any number of areas to grow into. Any areas which overlap with another bud's areas are removed. The average of the leftover areas is the growth direction. If a bud is left with no areas, then it can no grow anymore. Shadow Propagation calculates the amount of shadow from each bud, and the amount of shadow on each bud. The bud then grows towards the light.

2. Whether the bud produces new shoots and the length of the shoot is calculated using the environmental input. The Borchert-Honda model calculates the light exposure of each bud and the light resource of the whole tree. The resource is then distributed and used to calculate the fate of each bud. The Priority mode also calculates the light exposure but uses weights based on the bud's position in the tree in the resource calculations.

3. The angle of a new shoot is calculated as a weighted sum of three vectors. The default orientation is the angle calculated in part 1. The optimal growth direction is the tendency to grow towards the light. Finally, the tropism vector is the tendency to bend under gravity.

4. A branch will be shed if the ratio of the total amount of light compared to the branch size falls beneath a threshold, to reduce resource distribution.

5. Each leaf contributes an initial diameter width to the tree, which adds up as we traverse back to the trunk. So the trunk is the added sum of all leaves.

2.3 Summary

The original L-system details how to create an expanded string using rewriting rules. The turtle interpretation translates each letter from the string into actions for a turtle, changing the string into a geometric shape. Finally, saving states on a stack allows the turtle to return to the trunk after drawing a branch, creating a tree model. Introducing two more vectors can produce 3D tree models. The Stochastic L-system uses probability to make the model more unique, while the Context-sensitive L-system changes the rules based on the node's surroundings. The paper [2] expands the L-system tree model using various methods based on the environment of the tree.

Chapter 3

Work Done

3.1 Godot

Starting with no graphics background, the first step in this project was catching up on graphics fundamentals. This was done using the game engine Godot, which was chosen as an introduction to game engines, as it has a shallow learning curve and is popular. It can quickly create a game environment with terrain and can respond to user input.

3.1.1 First Person Player

The basics of the Godot interface were learnt through a 5hr tutorial on YouTube [4] which lead the viewer through the steps to create a simple 3D game involving a rolling ball, enemies, and coin collection. The important things learnt from the tutorial was the ability to move a camera around a scene, attaching a script to an object, and kinematic bodies.

Each Node can have a single script attached. A script contains Godot code which includes the ability to call internal functions and alert other Nodes of an event. A Kinematic Body is an object which interacts with other objects following the laws of physics [5], like falling and rolling with gravity. Combining this knowledge can lead to a movable object. The Kinematic body allows it to react naturally to the ground and other obstacles, while the script can include code to move the object based on the user inputs:

```
func _physics_process(delta):
    if Input.is_action_pressed("ui_right"): velocity.x = SPEED
    elif Input.is_action_pressed("ui_left"): velocity.x = -SPEED
```

The function `_physics_process` is called every time the engine draws a frame. `Input` is an object which holds any user input made since the last frame. If the right key (`ui-right`) or left key (`ui-left`) is pressed, the x variable of the velocity vector is edited, moving right or left.

Finally, without a camera, a 3D scene would appear as a black screen during run time. Connecting the camera to the Kinematic body will cause it to move with the body as the user moves the body. Thus, the combination of this knowledge is a first-person player!

3.2 L-system

3.2.1 Java

To understand the concepts behind the L-system, the initial programs were written in Java. The first step was creating a basic L-system, expanding to a tree model, then implementing

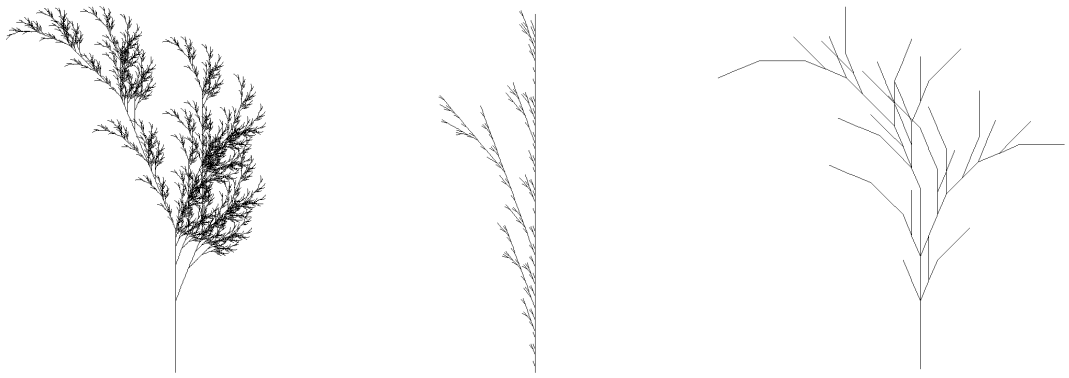
the stochastic and context-sensitive systems [1].

The basic L-system is rewriting a string using given rules. To implement this, the string is split into characters, allowing the program to loop through each letter, replacing it if there is a rule connected. The rules are saved in a map with the letter as the key, allowing for a quick lookup if the current letter has a rule. The string to replace the letter with is saved as the value. This is repeated for the specified number of rewriting steps.

A tree model is built from the string produced from the basic L-system. The letters with actions are F, $+$, $-$, [, and], any other letters are ignored during the build process. These relate to actions to move the 'turtle' around. The turtle is represented as a State object, which holds the current position (x, y) and the angle (δ). Every action updates the turtle. When the turtle moves forwards, a line is created from the original position to the new position. This can be saved or drawn onto the canvas. A problem that arose during this process was the x-axis pointing down, requiring the angle to be flipped (shifted 180°) before calculating the new y value.

The Stochastic L-system requires changes to the rewriting method. Instead of one string, it saves multiple strings in an ArrayList as the value of the Map. Thus, when rewriting a letter, a random string can be picked from the ArrayList to replace the letter. The current approach considers all strings as equally possible.

The Context-sensitive L-system saves the rewriting rules in an object which holds the letter, the neighbours, and the replacement string. During the rewrite method, a separate string is created, excluding all ignored letter, allowing the search for neighbours to be quicker. If the letter is not ignored, its neighbours are found from the separate string and the correct rule is discovered based on the neighbours. If the letter is ignored, it is still checked against the rules as it may match an angle rule ($* + * \rightarrow -$ or $* - * \rightarrow +$), else it doesn't change.



(a) Normal L-system (b) Stochastic L-system (c) Context Sensitive L-system

Figure 3.1: Tree models built in Java. L-systems can be found in the Appendix [6].

3.2.2 Godot

I attempted to replicate the Normal L-system program in Godot's programming language, GDScript [5]. This involved using the Dictionary structure, which is similar to the Java Map collection type, to keep the rules in for quick lookup. The method I had come up with copied a line object and edited it to be between the two points for that branch segment. However, creating a new object each time (or copying) is very expensive. Thus, when asked to reproduce the tree model in Figure 3.1a it took a long time and produced a lot of errors.

After some research, the correct approach should be to write a plugin that draws a tree at a low level.

Chapter 4

Future Plan

4.1 Progress Compared to Original Plan

The original plan [7] laid out in the project proposal expected a tree model built by the L-system to be completed by week 11. However, the current version is only represented by lines in Java.

4.2 Revisions to Project Timeline

Weeks 11 and 12 will be busy with assignments and lectures. There is still some work to be done on the line version of the tree model, which should be completed by Week 13. Once completed, it should be expanded to create a 3D mesh of the tree model.

Finally, the paper [2] will be split into the parts described in 2 and spread over the second trimester. Most of the work should be completed by week 27, allowing time to write the Final Report and prepare for the Presentation. This should hopefully reduce the workload during the stressful weeks at the end of Trimester 2.

Part 5 of the paper [2], calculating the width of a tree branch, will be implemented first, as it will be useful in creating a realistic 3D mesh of a tree model.

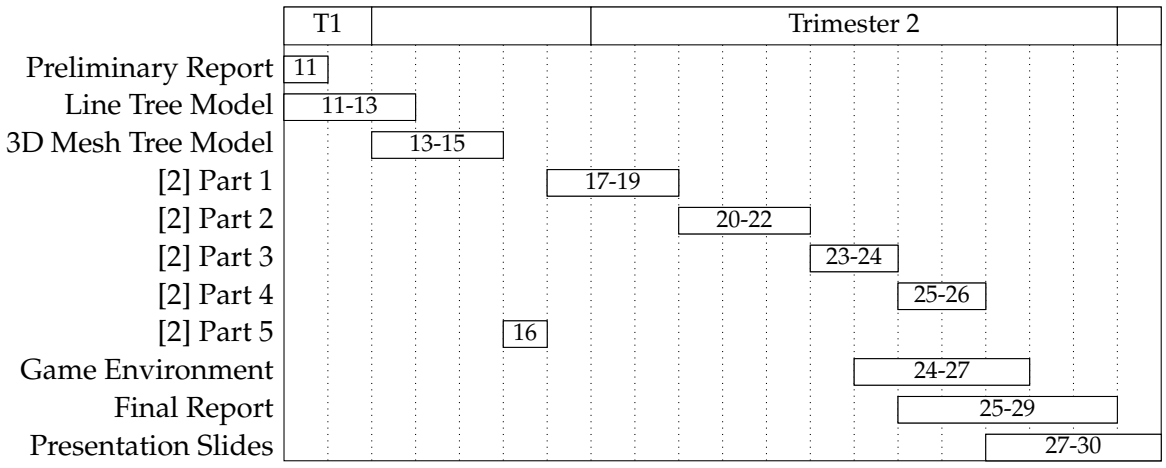


Figure 4.1: Updated Proposed timeline for Project

Feedback

This section highlights any difficulties currently faces, and makes requests for guidance from the examination committee.

I got sick in week 9, preventing me from working on this project or other assignments. This was also the point in the Trimester when the stress starts increasing. Due to this, I was unable to work on the project for two weeks. However, the current project timeline is still feasible based on the work completed in this Trimester.

Bibliography

[1] A. Lindenmayer and P. Prusinkiewicz, *The Algorithmic Beauty of Plants*. Springer New York, Mar. 1990.

[2] W. Palubicki, K. Horel, S. Longay, A. Runions, B. Lane, R. Měch, and P. Prusinkiewicz, "Self-organizing tree models for image synthesis," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 1–10, Jul. 2009.

[3] A. R. Smith, "Plants, fractals, and formal languages," *ACM SIGGRAPH Computer Graphics*, vol. 18, no. 3, pp. 1–10, Jul. 1984.

[4] BornCG, "Godot 3.1: Creating a simple 3d game," YouTube, Mar. 2021. [Online]. Available: https://www.youtube.com/playlist?list=PLda3VoSoc_TSBBOBYwcmlamF1UrjVtccZ

[5] J. Linietsky, A. Manzur, and the Godot Community, "Kinematicbody," *Godot Docs*, 2021. [Online]. Available: https://docs.godotengine.org/en/stable/classes/class_kinematicbody.html

[6] B. Gatehouse, "Tree models and l-systems," In Appendix, May 2021.

[7] ——, "Project proposal," In Appendix, Mar. 2021.