School of Computing

FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Procedural Generation of a Stochastic Forest

Billy Whitaker

Submitted in accordance with the requirements for the degree of
<Name of Degree> MEng Computer Science with High-Performance Graphics and Games Engineering

<Session> 2022/23

<Module code and name> *COMP3931 Individual Project*

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Final Report	*PDF file*	*Uploaded to Minerva (27/04/2023)*
Video	*MP4 file in repository called Video_Demo.mp4*	*Sent to supervisor and assessor (27/04/2023)* *https://gitlab.com/sc20bw/Procedural-Forest-Modelling*
Link to online code repository	*URL*	*Sent to supervisor and assessor (27/04/2023)* *https://gitlab.com/sc20bw/Procedural-Forest-Modelling*
User manuals	*ReadMe Text File in repository*	*Sent to client and supervisor (27/04/2023)* *https://gitlab.com/sc20bw/Procedural-Forest-Modelling*

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) Billy Whitaker

Summary

In this project I will explore the techniques of rendering a 3D forest and then implementing them myself with my own piece of software. This will create beautiful random forests that could be used in a wide variety of ways. All trees in the forest should be random, unique and believable to the user. To do this I will write my code using c++ and use the OpenGL libraries to render 3D objects. The user will also be able to move around the environment going through the forest and being able to look over the forest.

Acknowledgements

First, I would like to acknowledge Dr Hamish Carr, my supervisor for this project, for helping me technically throughout the project and helping me figure out that this would be the problem I would like to tackle in my project.

I would then like to acknowledge my parents and my girlfriend for allowing me to chat about my project to them for the last year even though they most likely did not understand a thing I was saying.

Table of Contents

Chapter 1
Introduction

1.1 Introduction

Software for rendering procedurally generated trees and forests can be used in a wide variety of ways, varying from scientific modelling and experimentation to game design. In this project I wish to create a piece of software that could be the basis for either one of these scenarios. So, from my final product with a little bit of rewriting someone could purpose the tree algorithm for their own intensions. This will give the software more options in the future.

1.2 Aims and Objectives

The aims of this project are:

- Produce graphical models of trees that are unique from each other.
- Have at least three different tree types.
- Software can then produce a forest from these trees that can hold up to 500 trees, while only having a linear time increase.
- Have a UI that allows the user to change the number of trees, types of trees in the forest and allows them to re-run the program making the forest look different each time

Chapter 2
Background Research and Design

2.1 Background Research

2.1.1 L-Systems

The first method for modelling plant structures graphically is **L-systems** (Lindenmayer Systems) introduced in [2]. These models can be created in a 2D environment and a 3D environment. The basis of L-Systems is about rewriting, where you are given a symbol and use this to produce something else. In Algorithmic Beauty of Plants (ABOP) [1] this is demonstrated in 2D where you have an **initiator** (a regular sided polygon) and a **generator** (a set of connected lines with no cycle). Then using the initiator each line is replaced by the generator to produce a new shape, like in Figure 1.

2.1.1.1 Turtle Interpretation of Strings

The main method of L-Systems I will use in this project is that of L-Systems using character strings as the initiator and 3D graphical meshes as the generators. I will also use a **turtle** that in 2D space can be described with variables x, y and a where x and y are the Cartesian coordinates of the turtle and “a” represents the angle that the turtle is facing. The syntax of the grammar defines how the turtle will move around the space, or another way to look at it is the syntax describes how the variables of the turtle will change over time. In [1] the syntax for 2D turtle interpretation is defined as:

F - which tells the turtle to travel a length of d in the direction it faces and draw a line from its initial position and its final position.

f - which tells the turtle to travel a length of d in the direction it faces without drawing a line.

+ - which tells the turtle to turn the direction it is looking left by δ.

- - which tells the turtle to turn the direction it is looking right by δ.

An example string for this, that would draw a square in front of and to the left of where the turtle is initially looking at, would be (where δ = 90 degrees):

F-F-F-F

We can then introduce multiple generators for our string this requires first introducing a variable n which determines how many times every line in a shape should be redrawn as the generator. So, if we imagine an algorithm for reading the strings of the initiator and generators to be recursive, then when a character representing a generator is read the algorithm draws the shape of the initiator instead of a normal line and does this n amount of times. On the nth call of a generator the algorithm just draws a line instead of calling another generator. To illustrate an example of this, in ABOP they show multiple quadratic Koch islands, found in [3], with varying values of n:

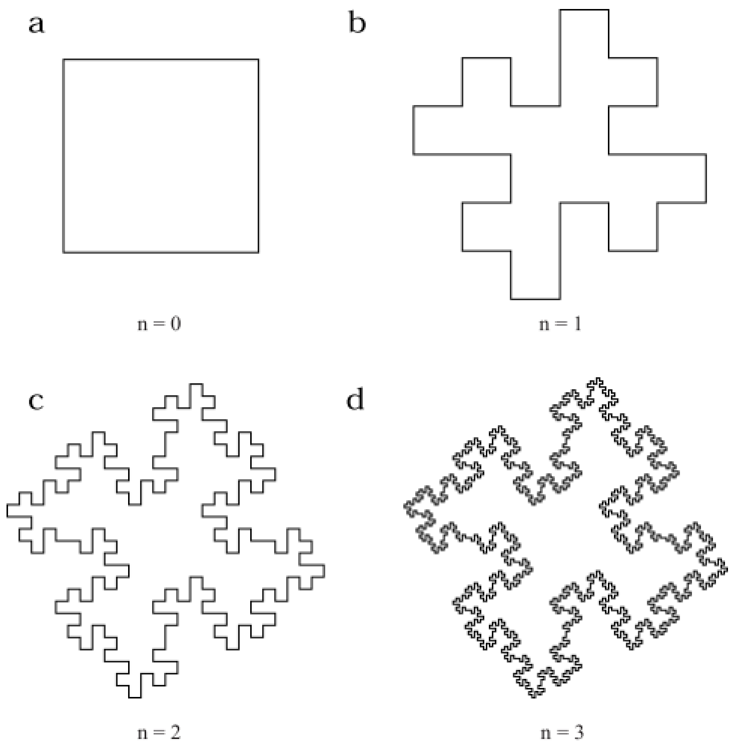


Figure 1: Quadratic Koch islands found in [1]

The strings used to create this Koch islands can be expressed as:

Initiator: F–F–F–F

Generator(s): F → F–F+F+FF-F-F+F

Where each F is replaced by the generator F → F–F+F+FF-F-F+F n times.

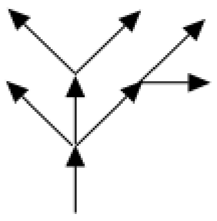
2.1.1.2 Branching Structures

Now that basic turtle interpretations of strings for L-systems has been explained, we can introduce the basis of tree like structures which is branching. To add branching to our syntax shown in section 2.1.1.1 we add the characters "[" and "]". Also, we need to introduce a stack to the algorithm that holds previous positions and directions of the turtle.

[- push the current position and direction of the turtle to the stack

] - change current position and direction of the turtle to that of the top of the stack and then pop the top of the stack

So now if our algorithm reads a [it will create a branch from the original branch. An example of this can be shown by:



F[+F][-F[-F]F]F[+F][-F]

Figure 2: Bracketed tree structure with string representation

2.1.1.3 Stochastic L-Systems

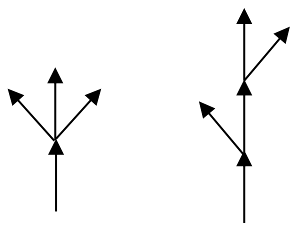
For all previously described L-System and tree strings there has been no variance for an individual string, so if we put multiple in the same environment they would be identical. This could obviously look very wrong when drawing a forest if the trees are all identical. So, to address this problem we can introduce stochastic or random characters/strings to our strings each with their own probability to be called. To demonstrate this let's introduce an L-System with n = 1 and δ = 60 degrees:

FA

A → [+F][-F]F with probability of 50%

A → [+F]F[-F]F with probability of 50%

This L-system has 50/50 chance to create these trees:



2.1.1.4 Parametric L-Systems

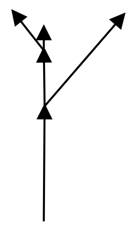
The trees in our previous examples have an issue where the length of the branches will always be the same. This causes our strings to only be able to produce certain shapes and trees unless the length of each branch is extremely small. But having extremely small branches causes another problem, that if we want to produce a long line in our string it will take a massive string causing concerns about storage, memory and how long it will take the algorithm to complete. Luckily there is a solution to all these problems, where we give the creator of the strings the ability to change the values of a variable as the string is being read. So, we introduce two new characters to our syntax “(“ and “)”. These parentheses are used to cut off the string from editing the turtle’s position so that it can read an expression between the two parentheses. Then the values produced by these expressions can be used to change the length of the branches (variable d in the examples) drawn from this point or to change any other important variables.

For the expressions between the parentheses, we can introduce another alphabet, that we’ll represent with V, independent from the syntax of our strings. This alphabet contains characters that represent float values that can be used in these expressions so that we don’t need to produce a calculator to evaluate the expressions. A character to introduce to the alphabet V in every L-System would be “d” representing the current length of the branches. To illustrate the use of these parametric L-systems we’ll add another character to V being H, where H = 0.5. Using this we can produce an L-system, where n=1 and δ=45 degrees, like:

F

F → F[-F]F(d*H)[+F]F(d*H)

Which would produce:

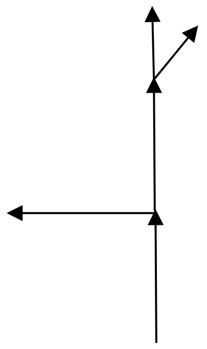


If using these parametric L-Systems we wished to alter the values of multiple variables at the same time we could add a comma between each expression. An example L-System, where n=1, δ=90 and introducing a new character to V being δ representing the value of δ at the time the expression is read, could be:

F

F → F[-F]F[+F(d*H,δ*H)]F

Which would produce:



2.1.1.5 3D L-systems

When using L-Systems in 3D environments we need to add a few new variables first we add z so that we now have x, y and z representing the turtle’s position in 3D space. Then we introduce three angle variables, one that controls which way the turtle looks left and right, one that controls the pitch up and down of the turtle and one that controls the roll of the turtle. Then to change these variables we need to introduce 3D rotation matrices, which can be read more upon here [4], but are:

$$\mathbf{R_U}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_L}(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

$$\mathbf{R_H}(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

Figure 3: 3D rotation matrices

Using these rotation matrices we can create a new syntax for our L-Systems in 3D Space:

F – Same as in 2D

f - Same as in 2D

+ - Turn left by and δ, using rotation matrix $R_U(\delta)$

- - Turn right by and δ, using rotation matrix $R_U(-\delta)$

& - Pitch down by and δ, using rotation matrix $R_L(\delta)$

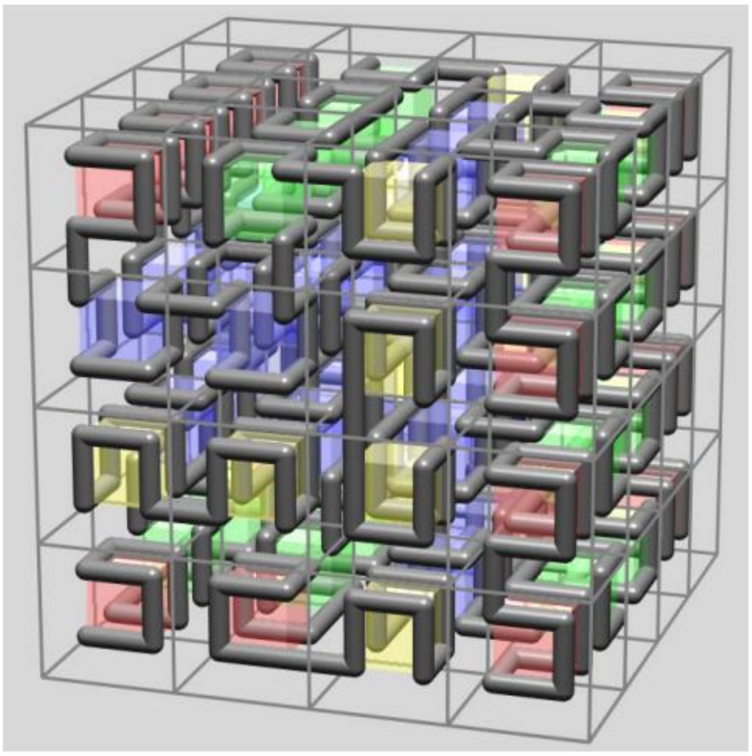
^ - Pitch up by and δ, using rotation matrix $R_L(-\delta)$

\ - Roll left by and δ, using rotation matrix $R_H(\delta)$

/ - Roll right by and δ, using rotation matrix $R_H(-\delta)$

| - Turn around, using rotation matrix R_U(180 degrees)

An example of L-System using this syntax is presented in ABOP and looks like this:



```
n=2,  δ=90°
A
A → B-F+CFC+F-D&F∧D-F+&&CFC+F+B//
B → A&F∧CFB∧F∧D∧∧-F-D∧|F∧B|FC∧F∧A//
C → |D∧|F∧B-F+C∧F∧A&&FA&F∧C+F+B∧F∧D//
D → |CFB-F+B|FA&F∧A&&FB-F+B|FC//
```

Figure 4: 3D structure produced by L-Systems

2.1.2 Self-Thinning Process

An environment where a forest can populate has a finite amount of resources, such as light, water and nutrients, no matter how large it is. The spread of these resources are what determines how plants are distributed around the environment. The amount of resources a single plant takes is known as **the yield of the plant.** So, the total **yield of the forest** must be larger than the sum of the yields of the plants in the forest. Therefore the **density** of the forest determines the survival chance of any new plant introduced to the environment. As each new plant is introduced to the environment, the mean yield of the plants decreases causing the probability of a new plant surviving decreasing. In A Model of Interference Within Plant Monocultures [6] Empirical equations have been formed to model these density dependencies. Mean yield per plant can be described as:

$$\overline{w} = \overline{w}_m(1 + aN)^{-b}$$

Where $\overline{w}$ is the mean yield per plant, $\overline{w}_m$ is the mean yield of isolated plants grown in the same habitat for the same time, N is the Density of plants and “a” and “b” are fitted parameters. Density-dependent mortality due to **self-thinning** can be described as:

$$N = \frac{N_i}{(1+mN_i)}$$

Where N_i is the initial density and m is the maximum density that the habitat can support. The parameter m changes over time as plants grow according to the relationship

$$w = cm^k$$

Where c and k are fitted parameters, k usually having a value of approximately 1.5. This relationship describes the limit of biomass and density as no combination of density and biomass can be larger than it.

Now if we look at an environment where we have a single plant, the size of the plant depends on its age, genotype, the resources of the environment and actions of herbivores and pathogens. The area of resources that the plant can take from is limited to a finite area, therefore any plant outside this area will not affect the growth of this plant, this area can be termed as **the ecological neighbourhood area**, which we will represent with q. Now that our single plant has taken the resources from this area it needs, any new plant introduced into this area will have only the amount of resources left by the first plant. This area can be termed as the **zone of influence**, which we will represent with z. Therefore z ≤ q and the yield of an induvial plant, w, is a function of z, q and the yield it would have had with no other plants around. Yield of a plant isn't directly proportional to z, as some other factors such as physiology will affect the proportion of how much resources a plant needs. This efficiency of resource utilisation can be represented as r. We can model this with the following equation:

$$w = w_m({}^{z}/_{q})^r$$

Using this we can model the mean yield per plant by:

$$\overline{w} = \sum_{i=1}^{N} w_{mi}({}^{z}/_{q})^r$$

With these equations we can model how plants may grow in different habitats with different distributions of resources.

2.2 Design

2.2.1 Tree Design

The first thing I planned for the implementation of the project was my syntax of the strings to be interpreted by the turtle, this is what I figured out:

F – Move forward and draw cylinder length of d

f - Move forward without drawing

+ - Turn left by δ, using rotation matrix $R_U(δ)$

- - Turn right by δ, using rotation matrix $R_U(-δ)$

& - Pitch down by δ, using rotation matrix $R_L(δ)$

^ - Pitch up by δ, using rotation matrix $R_L(-δ)$

| - Roll left by δ, using rotation matrix $R_H(δ)$

/ - Roll right by δ, using rotation matrix $R_H(-δ)$

[push the current position and direction of the turtle to the stack

] change current position and direction of the turtle to that of the top of the stack and then pop the top of the stack

(- beginning of an expression

) – end of an expression

Digit – Code for initiator or same as F if n is equal to 0

L – If n equals 0 draw a leaf and don't move turtle, else do nothing

This has a few differences and additions to the syntax introduced in section 2.1.1.5. First, I have removed the rotate 180 degrees as there isn't a need to turn back on a tree branch itself and if this was needed it can still be done with multiple +'s in the string. Then I have changed the symbol for roll left by δ, using rotation matrix $R_H(\delta)$ from "\" to "|", as I will be writing the program in c++ and "\" in a string is recognised as an operator so would cause an error. Finally, I plan on storing all initiators into an array of strings. Therefore, I have added all integers to the syntax which shall represent the index of the next string (initiator) that should be interpreted by the turtle. An example L-System of this could be:

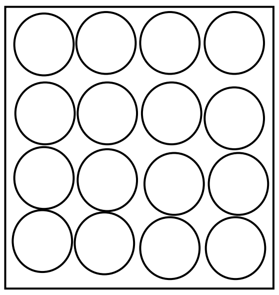
0: F[+1][-1]0

1: F[/1L]F[|1L]1

The turtle would interpret this by starting with string 0 reading the code systematically, executing instructions corresponding to each character until it reaches a digit. If n is equal to 0 then the code just interprets the digit like an F, but if n > 0 then the function that interprets a string will re call itself and interpret the next string that corresponds to the digit that was read.

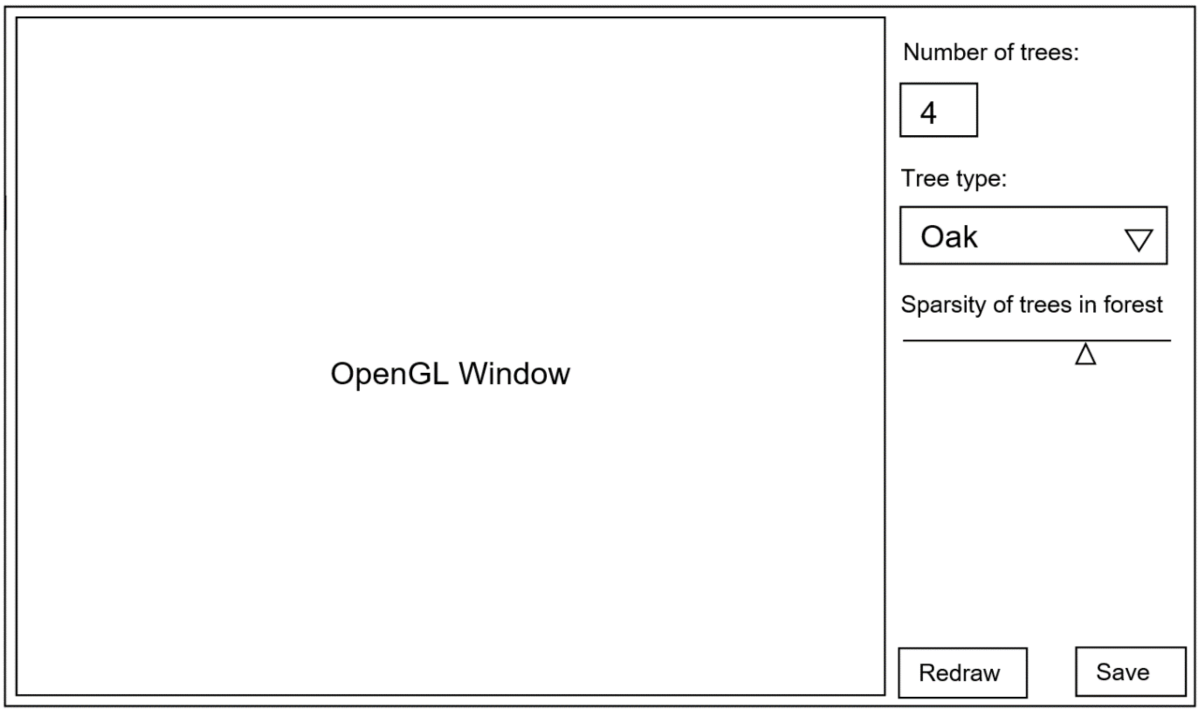
2.2.2 Forest Design

For the design of the forest and more specifically the distribution of trees in the area the program should take in the number of trees wanted in the forest and determine the length of the square area the trees will take up. Then the square will be split up into ecological neighbourhood areas which can be imagined like this, though will be invisible to the user:



Each ecological neighbourhood area would then be given arbitrary resources values either a value for available water, light, nutrients or one whole value for the area. Then randomly assign a point coordinate for a tree if the ecological neighbourhood area has enough resources for the trees genotype to create a full sized tree, then use all these resources and create full grown tree. If the area it is placed in does not have enough resources to create a fully grown tree but enough for a sapling to survive then take up the remaining resources of that area and create a tree of equivalent size. If an area doesn't have enough resources for a sapling to survive find another point to create the tree in the forest. Repeat this till either the given number of trees is reached or till each zone does not have enough resources for a sapling to survive.

2.2.3 User interface Design



OpenGL Window – Shows the generation of the forest and allows the user to move around the forest in this window

Number of trees – Allows the user to change the number of trees that will be in the next forest when redrawn

Tree type – Choose the type of trees in the next forest when redrawn

Sparsity of trees in forest – Allows the user to edit how close trees will be to one another on average

Redraw – Draw a new different looking forest with the current values in the UI

Save – Save the meshes of the current forest so it can be generated at a later time without re doing the algorithm to create it

Chapter 3
Methods

3.1 3D-Camera and Lighting

3.1.1 Arc Ball Camera

The camera my program uses that allows the user to travel around the forest is based on the arc ball camera. The camera is defined by a structure called camControl with boolean values; cameraActive, actionZoomIn, actionZoomOut, moveLeft, moveRight, moveUp, moveDown, fast, and float values phi, theta, x, y, z, speed, lastX and lastY:

cameraActive – True if the user presses space and allows the user to control the camera

actionZoomIn – True if the user is holding the W key and zooms the camera in or moves the camera forwards

actionZoomOut – True if the user is holding the S key and zooms the camera out or moves the camera backwards

moveLeft – True if the user is holding the A key and moves the camera left

moveRight – True if the user is holding the D key and moves the camera right

moveUp – True if the user is holding the E key and moves the camera up

moveDown – True if the user is holding the Q key and moves the camera down

fast – True if the user is pressing left-shift and makes the camera move faster

phi – holds the value of the angle the camera is looking in the x axis

theta – holds the value of the angle the camera is looking in the y axis

x – holds the x position of the camera

y – holds the y position of the camera

z – holds the z position of the camera

speed – holds the speed at which the camera is moving

lastX – holds the value of the x coordinate of the mouse before it was moved

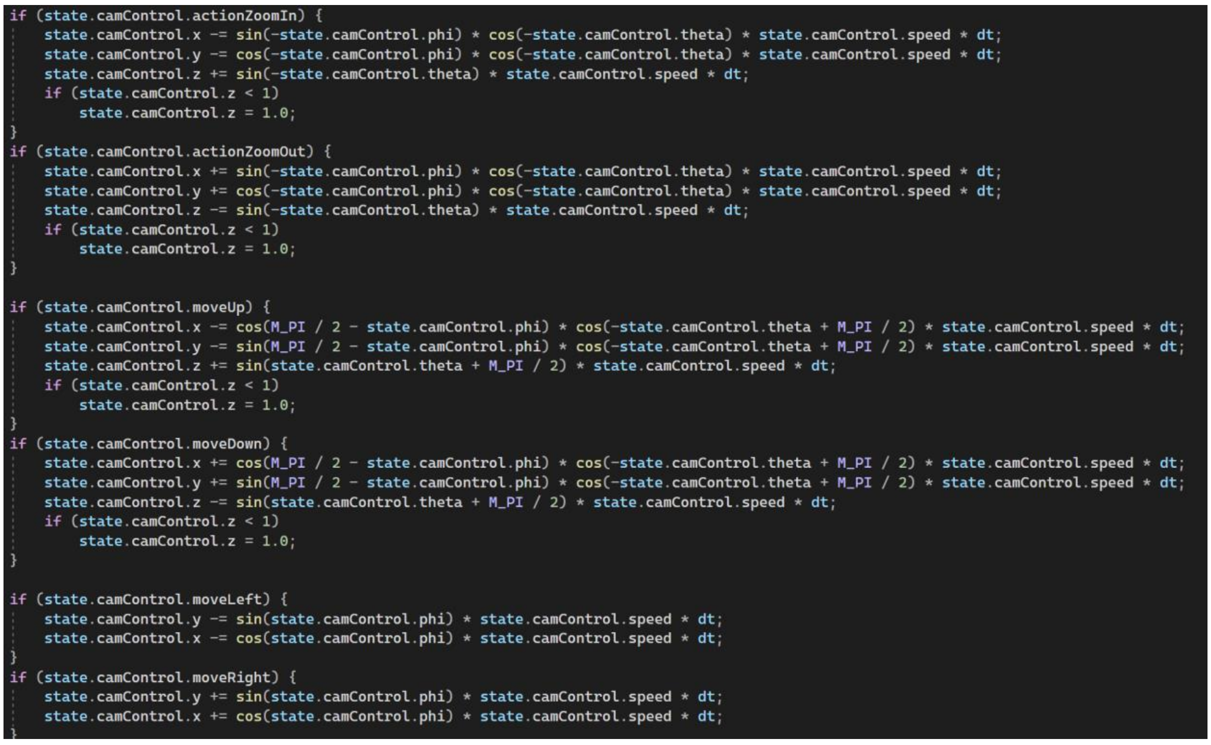
lastY – holds the value of the y coordinate of the mouse before it was moved

3.1.1.1 Camera Motion

The camera rotates by using a glfw_callback_motion_() which are provided with the glfw library and is called when the users mouse is moved. Then if the cameraActive variable is True the software calculates the change of x and y from what lastX and lastY was to where the mouse is currently located on screen using parameters, aX and aY, of glfw_callback_motion_(). These differences in x and y are stored in local variables dx and dy, then using these values we take dx multiplied by the mouses sensitivity away from phi and add dy multiplied by the mouses sensitivity to theta.

3.1.1.2 Camera Movement

The camera moves around the environment using glfw_callback_key_() making the Booleans presented earlier True if the corresponding key is pressed or held. Then in the main game loop selection statements are used to change the values of x, y and z depending on which buttons are being held. To calculate the changes of x, y and z the software uses trig with phi and theta for the direction the camera is looking at, these equations look like this for each button press for movement:



```
if (state.camControl.actionZoomIn) {
    state.camControl.x -= sin(-state.camControl.phi) * cos(-state.camControl.theta) * state.camControl.speed * dt;
    state.camControl.y -= cos(-state.camControl.phi) * cos(-state.camControl.theta) * state.camControl.speed * dt;
    state.camControl.z += sin(-state.camControl.theta) * state.camControl.speed * dt;
    if (state.camControl.z < 1)
        state.camControl.z = 1.0;
}
if (state.camControl.actionZoomOut) {
    state.camControl.x += sin(-state.camControl.phi) * cos(-state.camControl.theta) * state.camControl.speed * dt;
    state.camControl.y += cos(-state.camControl.phi) * cos(-state.camControl.theta) * state.camControl.speed * dt;
    state.camControl.z -= sin(-state.camControl.theta) * state.camControl.speed * dt;
    if (state.camControl.z < 1)
        state.camControl.z = 1.0;
}

if (state.camControl.moveUp) {
    state.camControl.x -= cos(M_PI / 2 - state.camControl.phi) * cos(-state.camControl.theta + M_PI / 2) * state.camControl.speed * dt;
    state.camControl.y -= sin(M_PI / 2 - state.camControl.phi) * cos(-state.camControl.theta + M_PI / 2) * state.camControl.speed * dt;
    state.camControl.z += sin(state.camControl.theta + M_PI / 2) * state.camControl.speed * dt;
    if (state.camControl.z < 1)
        state.camControl.z = 1.0;
}
if (state.camControl.moveDown) {
    state.camControl.x += cos(M_PI / 2 - state.camControl.phi) * cos(-state.camControl.theta + M_PI / 2) * state.camControl.speed * dt;
    state.camControl.y += sin(M_PI / 2 - state.camControl.phi) * cos(-state.camControl.theta + M_PI / 2) * state.camControl.speed * dt;
    state.camControl.z -= sin(state.camControl.theta + M_PI / 2) * state.camControl.speed * dt;
    if (state.camControl.z < 1)
        state.camControl.z = 1.0;
}

if (state.camControl.moveLeft) {
    state.camControl.y -= sin(state.camControl.phi) * state.camControl.speed * dt;
    state.camControl.x -= cos(state.camControl.phi) * state.camControl.speed * dt;
}
if (state.camControl.moveRight) {
    state.camControl.y += sin(state.camControl.phi) * state.camControl.speed * dt;
    state.camControl.x += cos(state.camControl.phi) * state.camControl.speed * dt;
```

Figure 5: Equations for the change of x, y and z for each key press

3.1.1.3 Camera Variables Used in Projection

Now to truly understand the camera we need to understand that the camera never really moves but that all the objects in the scene are moved around the camera depending on the user’s input. So, the software needs to create a transformation matrix for where the objects should be placed around the camera each game loop.

The first thing the software does to create this matrix is make two rotation matrices, one rotation around the x axis with phi and one ration around the y axis with theta. Then we create a translation matrix with x, y and z values of the camera and times the two rotation matrices and this translation matrix together. Finally, we create a perspective projection using the field of view and height and width of the window and times this with the other matrices. Then this matrix will be used in the vertex shader for all meshes in the scene to place them in the correct location around the camera by multiplying the vector positions of the meshes by this matrix.

3.1.2 Blinn-Phong Lighting

The lighting technique for the forest is based on Blinn-Phong lighting [11]. First the program defines the value of the lights position and its colour, in my software each forest only has one source of light. These are then sent to the fragment shader along with a Boolean variable that’s value represents if an object has a texture or not. The fragment shader is how OpenGL calculates the values for the colour of objects in which the light around the objects have to be taken into account.

In Blinn-Phong lighting we calculate three values that represent the light. Ambient lighting, which can be imagined as the intensity or the brightness of light globally. Diffuse lighting can be imagined as the direction of the light and its proximity to an object, this is what causes one side of an object to be shaded and different objects to be clearer as they are closer to a light. Finally, there is specular lighting, which is used to show the reflection of light in an object, this has not been implemented in my software as trees are not obviously reflective.

Ambient light is easily calculated using the light colour already passed to the fragment shader and then multiplying it by a constant value being the intensity of the light, in our model we use an intensity of 0.5.

Diffuse lighting gets a bit more difficult requiring the normals of an object, which are passed through the vertex shader, as in section 3.2.3. Then we calculate the direction of the light by taking the position of the fragment away from the lights position and dot product [12] this direction with the normal of the fragment. Then to create the diffuse light colour we times the value obtained by the dot product with the lights colour like with ambient lighting. Then for each fragment of every object we add the ambient and diffuse values together, then times their sum by the colour of the object and if the object is textured we times them by the textures values.

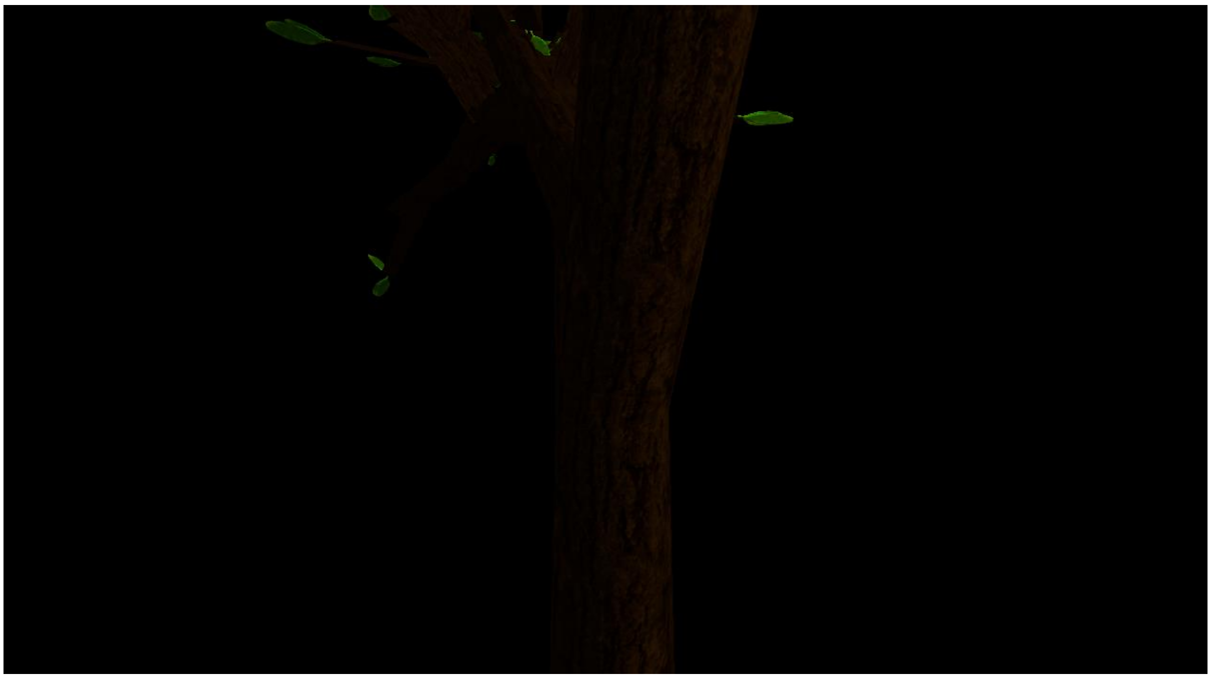


Figure 6: Demonstration of light on a tree’s branches

3.2 3-D Tree Generation

The algorithm to create the tree is a recursive algorithm that reads a string, which causes the turtle to move in a given way and possibly draw a branch or leaf mesh.

3.2.1 Turtle Structure

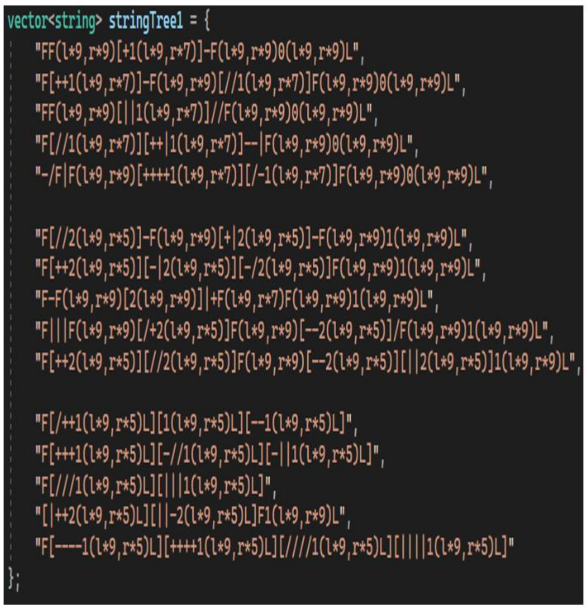
In the software the turtle is defined using a structure called Turtle. This structure has variables, x, y, z, angleX, angleY, angleZ, length, radius1 and radius2. Variables x, y and z refer to the turtles coordinates in 3-D space, angleX, angleY and angleZ refer to the turtle’s rotation around the axis in the variable name. Length, radius1 and radius2 are used for drawing the meshes that the turtle creates, see section 3.2.3.

Variables x, y and z of the turtles are initialised using vectors obtained from the forest algorithm, see section 3.3. Variables angleX, angleY and angleZ are all initialised as zero as the turtle has not rotated yet. Variables length, radius1 and radius2 are initialised by the user, before the forest algorithm is called, using the user interface, see section 3.4.

3.2.2 Tree Algorithm and Strings

3.2.2.1 Tree Strings

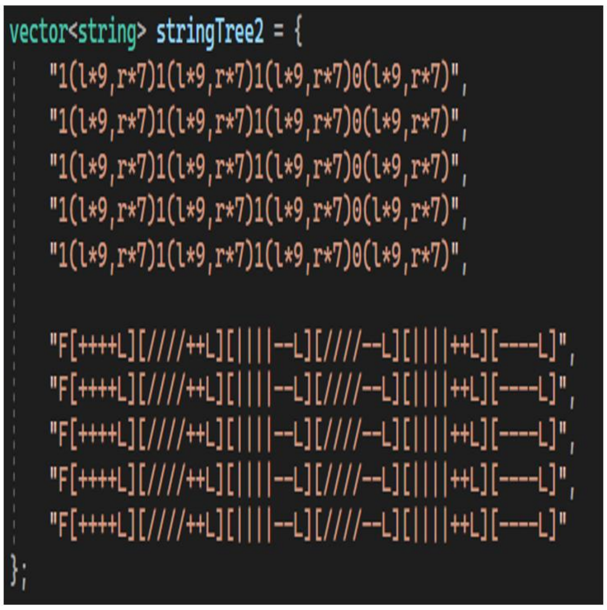
Two tree strings are hard coded in as vectors of strings for the different initiators and stochastic versions of each initiator. These are stringTree1, used for tree type 1 and 2 and stringTree2, used by tree type 3. These look like this in the code:



```
vector<string> stringTree1 = {
    "FF(l*9,r*9)[+1(l*9,r*7)]-F(l*9,r*9)0(l*9,r*9)L",
    "F[++1(l*9,r*7)]-F(l*9,r*9)[//1(l*9,r*7)]F(l*9,r*9)0(l*9,r*9)L",
    "FF(l*9,r*9)[||1(l*9,r*7)]//F(l*9,r*9)0(l*9,r*9)L",
    "F[//1(l*9,r*7)][++|1(l*9,r*7)]--|F(l*9,r*9)0(l*9,r*9)L",
    "-/F|F(l*9,r*9)[++++1(l*9,r*7)][/-1(l*9,r*7)]F(l*9,r*9)0(l*9,r*9)L",

    "F[//2(l*9,r*5)]-F(l*9,r*9)[+|2(l*9,r*5)]-F(l*9,r*9)1(l*9,r*9)L",
    "F[++2(l*9,r*5)][-|2(l*9,r*5)][-/2(l*9,r*5)]F(l*9,r*9)1(l*9,r*9)L",
    "F-F(l*9,r*9)[2(l*9,r*9)]|+F(l*9,r*7)F(l*9,r*9)1(l*9,r*9)L",
    "F|||F(l*9,r*9)[/+2(l*9,r*5)]F(l*9,r*9)[--2(l*9,r*5)]/F(l*9,r*9)1(l*9,r*9)L",
    "F[++2(l*9,r*5)][//2(l*9,r*5)]F(l*9,r*9)[--2(l*9,r*5)][||2(l*9,r*5)]1(l*9,r*9)L",

    "F[/++1(l*9,r*5)L][1(l*9,r*5)L][--1(l*9,r*5)L]",
    "F[+++1(l*9,r*5)L][-//1(l*9,r*5)L][-||1(l*9,r*5)L]",
    "F[///1(l*9,r*5)L][|||1(l*9,r*5)L]",
    "[|++2(l*9,r*5)L][||-2(l*9,r*5)L]F1(l*9,r*9)L",
    "F[----1(l*9,r*5)L][++++1(l*9,r*5)L][////1(l*9,r*5)L][||||1(l*9,r*5)L]"
};
```



```
vector<string> stringTree2 = {
    "1(l*9,r*7)1(l*9,r*7)1(l*9,r*7)0(l*9,r*7)",
    "1(l*9,r*7)1(l*9,r*7)1(l*9,r*7)0(l*9,r*7)",
    "1(l*9,r*7)1(l*9,r*7)1(l*9,r*7)0(l*9,r*7)",
    "1(l*9,r*7)1(l*9,r*7)1(l*9,r*7)0(l*9,r*7)",
    "1(l*9,r*7)1(l*9,r*7)1(l*9,r*7)0(l*9,r*7)",

    "F[++++L][////++L][||||--L][////--L][||||++L][----L]",
    "F[++++L][////++L][||||--L][////--L][||||++L][----L]",
    "F[++++L][////++L][||||--L][////--L][||||++L][----L]",
    "F[++++L][////++L][||||--L][////--L][||||++L][----L]",
    "F[++++L][////++L][||||--L][////--L][||||++L][----L]"
};
```

Figure 7: String representations for each tree type.

Each initiator has five different variables it can take on making each tree look different, each initiator is represented by a space between the next string in the vector in Figure 7. Therefore, say the code is told to draw a mesh with initiator 0, when 0 is read from a string, then one of the first five strings will be picked.

3.2.2.2 String Interpretation Algorithm

In the code the algorithm that interprets the strings of a tree is a function called drawTree(). This function has parameters for the initial turtle values, the vector of strings, the string to be interpreted, the current fractal of the string, the current mesh of the tree (initially an empty structure) and variables for dimensions of the tree's leaves. A lot of these parameters are used in functions called by this algorithm.

The algorithm loops through the string Bstring, passed to the function, and evaluates each character respectively. It does this using a selection statement where given a character it performs a certain task. Most characters perform the tasks mentioned in the design (section 2.2.1). Specifically, character F calls the function drawBranch() explained in section 3.2.3.1, it also checks if the character after it is a bracket, if it is then it will call the function evaluate_expression(), see section 3.2.4, before they call drawBranch(). This is similar to if the code reads digit as a character and n, a parameter to drawTree() holding the current fractal, is equal to zero. Though if n is not equal to zero the code evaluates an expression if a bracket follows and then calls drawTree() again using one of the initiators associated with the value of the digit being read. Character L calls the function drawLeaf(), see section 3.2.3.2, but does not evaluate any expression.

3.2.3 Meshes

The building blocks of the scene and the tree specifically are branches and leaves, these are known as meshes. In this code a mesh is a structure which holds the vertex positions of the triangles that make a mesh, the colour of the corresponding triangle, the normals of the corresponding triangle and the texture coordinates of the corresponding triangle.

3.2.3.1 Branch Meshes

The meshes for the branches are truncated cones and are created with the make_cylinder() function. In this function the mesh is created by going in a circle around a point and drawing rectangles and then capping them to create a prism. It does this by first deciding how many sides the prism will have, where a prism with more sides looks more cylindrical but will take more processing time to create. In my program this ranges from sixty-four sections for the trunk of a tree with five initial fractals to two for the end branches of all trees. So, for each mesh we draw each section (figure 8) iteratively using a for loop increasing the angle around the circle till the program gets back to where it was. The variables radius1 and radius2 are used to determine how close the bottom and top of the truncated cone are to the centre of the cone. Each section is made of four triangles and look like this, where one section is highlighted in read with a triangle reflective to the one on top on the bottom:

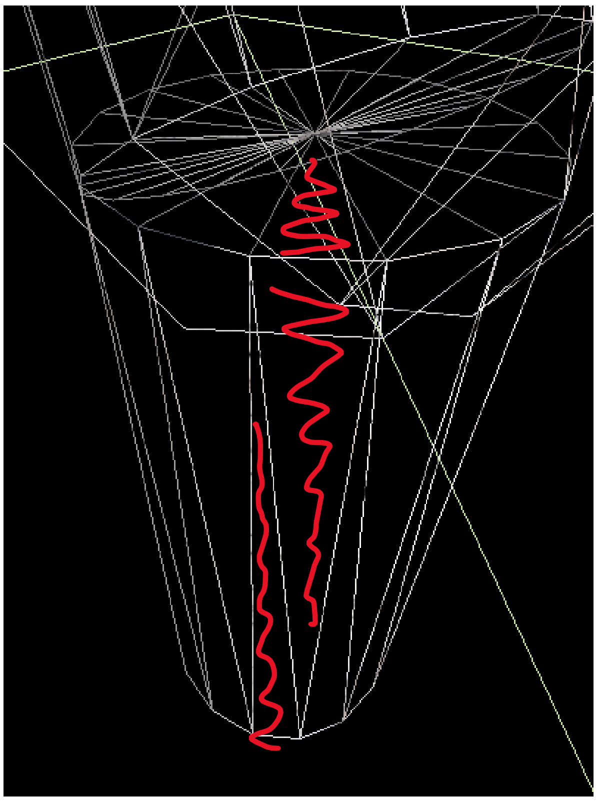


Figure 8: Example of a Branch Texture with Shading of a Single Section

3.2.3.1.1 Mesh Normals

The normals for the meshes are calculated for each triangle and are required for the lighting to work correctly. They are calculated by cross producting [12] the length of two adjacent sides (in vector form) for each tringle.

3.2.3.1.2 Textures

The texture coordinates are how the program maps the corners of the image used for the texture to the relevant corner of the vertex. This is therefore represented as 2D vectors and are between {0.0, 0.0} and {1.0, 1.0} where the first set of coordinates represents the bottom left corner of the image and the other set of coordinates represents the top right corner of the image.

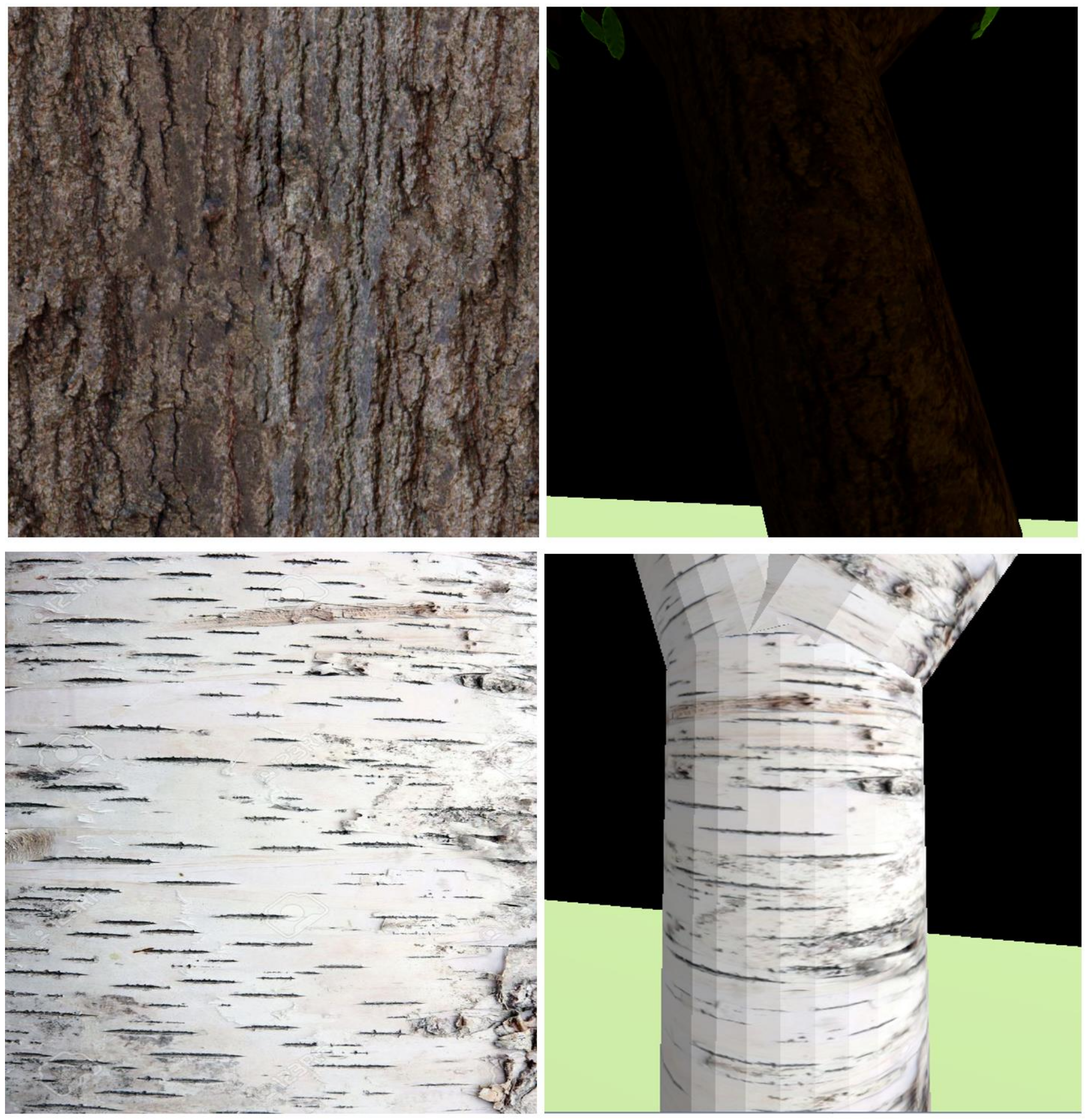
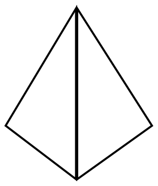


Figure 9: Bark Textures and a Demonstration of them on a Tree

Each image that is used to do the textures for the trees is completly wrapped around the sides of the meshes. The normal wood texture can be found at [7] and the birch tree texture can be found at [8].

3.2.3.2 Leaf Meshes

The leaf meshes work very similar to the branch meshes but since they are 2D they have caps at the top. At first, they were geometric shapes like this:



But since this was too primitive I changed it so that the leaves are squares where the centre of one of the edges of the square is found at the centre of the end of a branch. Then a texture of a leaf was placed on the surfaces of the square with transparent values. For tree type 3 this was a bit different as the leaf textures had branches so these leaves come out of the trunk of the tree. These can be seen here, the images used for the texture on the left can be found at [9] and the one on the right can be found at [10]:

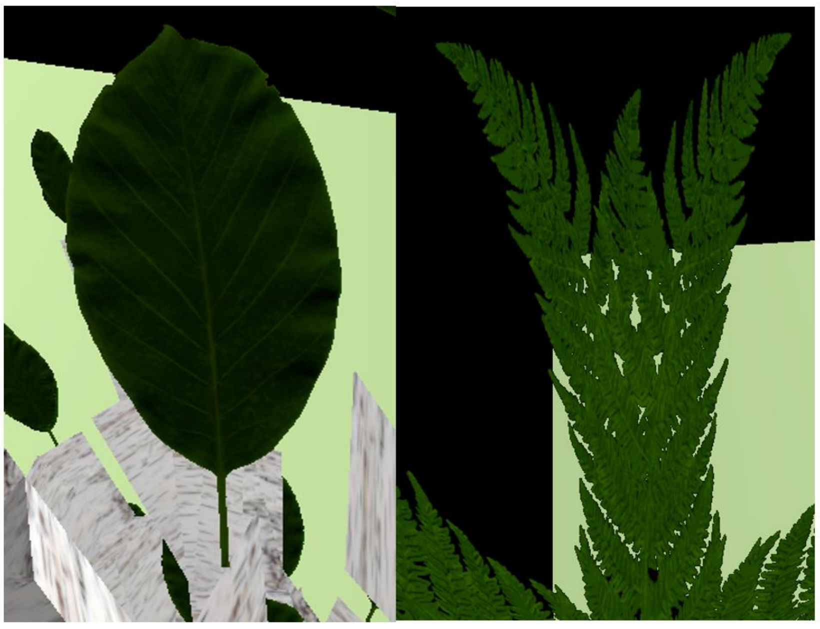
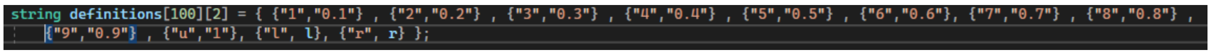


Figure 10: Examples of the Two Different Leaf Textures

3.2.4 Expression Evaluation

When the string interpreter reads an open bracket after a digit or F then it reads an expression in the form of “SymbolOperatorSymbol” and evaluates it, once it has evaluated it, if the next character of the string is a comma it evaluates another expression in the same form. It does this till the next character of the string is a closed bracket and continues with the algorithm as shown in section 3.2.2.2 . The evaluation of the expression is done in a function called evaluate_expression(), which requires two parameters, the expression in the form of a string and a 2D array of symbols and their values. This 2D array is predefined in the code and looks like this in the code:



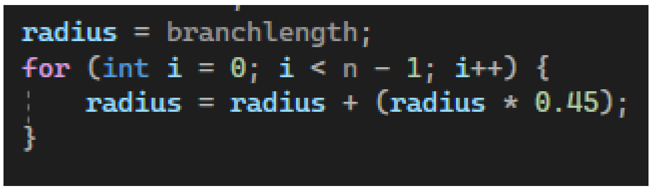
```
string definitions[100][2] = { {"1","0.1"} , {"2","0.2"} , {"3","0.3"} , {"4","0.4"} , {"5","0.5"} , {"6","0.6"}, {"7","0.7"} , {"8","0.8"} ,
    {"9","0.9"} , {"u","1"}, {"l", l}, {"r", r} };
```

The symbols l and r take on the current values of the branch length and radius1. The function evaluate_expression() works by first reading the first symbol of the expression and placing its corresponding value in an array. The operator is then read and a selection statement is used depending on the function. For an example we'll say the "*" operator for multiplication is read, the next symbol is read and its value is placed in the same array as the other. Then both values in the array are multiplied together and then returned by the function.

3.3 Forest Algorithm

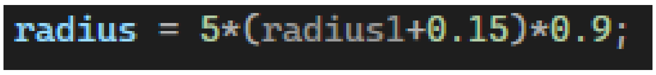
Due to time constraints I was unable to have the array of points of the ground plane to have separate resource values meaning the forest algorithm had to be changed from what was in the plan. Initially in my code the trees were spread around the environment using an imbedded for loop where the initial x and z position of the next tree are increased with each respective loop. This was very primitive and didn't reflect what a forest really looks like, even with random increases to the values. So, to make the forest look more realistic I changed the algorithm so that each tree has its own ecological neighbourhood area so that no other tree can grow in this area. Then the plane is restricted to an area where only N+1 number of trees can be made in it where no tree invades another's ecological neighbourhood area.

To do this the program first gives the trees a maximum radius based on the number of fractals, initial branch length and initial value of radius1 depending on its tree type. The calculations for these radiuses are shown here:



```
radius = branchlength;
for (int i = 0; i < n - 1; i++) {
    radius = radius + (radius * 0.45);
}
```

Figure 11: Calculation for the radius of tree types 1 and 2



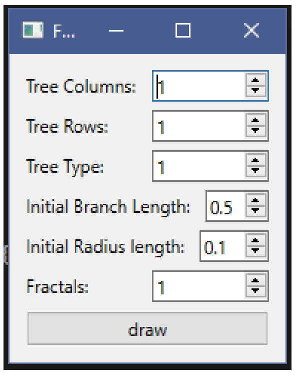
```
radius = 5*(radius1+0.15)*0.9;
```

Figure 12: Calculation for the radius of tree type 3

This radius is then used to calculate the length of the square plane that the trees of the forest will be located in. Then to find the initial positions of each tree in the forest the algorithm randomly finds a point in the square plain and adds it to a vector of positions. Then another point is randomly picked and is checked if it intercepts the radius of the other trees in the vector, this is repeated till all trees in the loop have an initial position on the plane. The vector holding these initial positions can be seen as a stack and in a for loop they are popped off and used to change the position of the turtle before it creates a new tree.

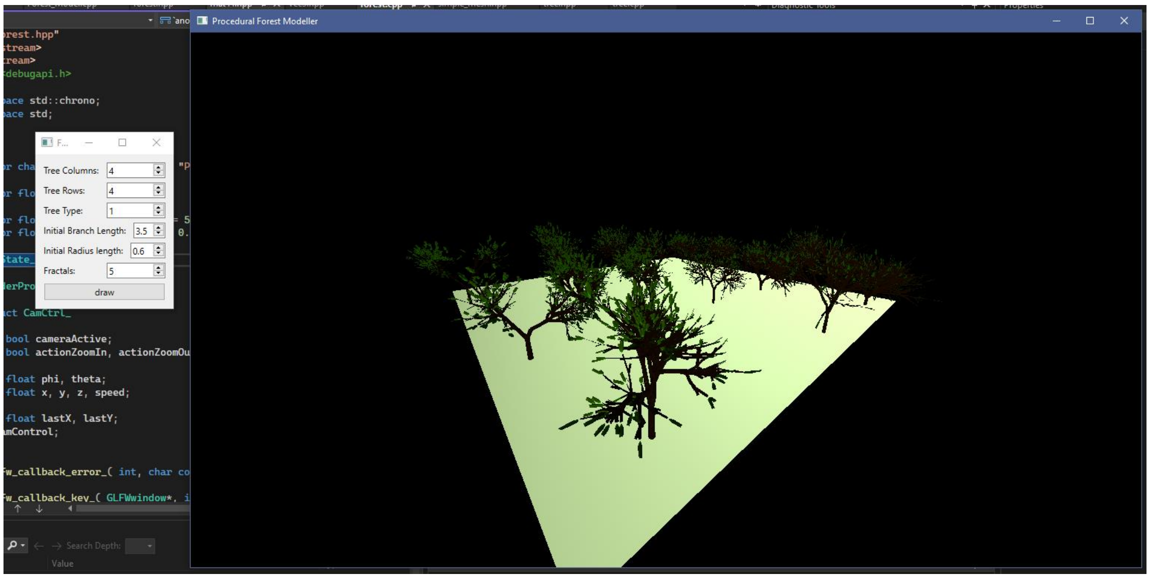
3.4 User Interface

Initially in the design the OpenGL window(s) were supposed to be part of the operating system but during the implementation I found that it would be easier to make the user interface separate with the library I used, that library being Qt, and that it worked better for my program. The user interface for my program looks like this:



Tree Columns:	1
Tree Rows:	1
Tree Type:	1
Initial Branch Length:	0.5
Initial Radius length:	0.1
Fractals:	1

Figure 13: Example of the User Interface of My Program



Tree Columns:	4
Tree Rows:	4
Tree Type:	1
Initial Branch Length:	3.5
Initial Radius length:	0.6
Fractals:	5

```
orest.hpp"
stream>
ream>
<debugapi.h>

pace std::chrono;
pace std;

ct CamCtrl_

 bool cameraActive;
 bool actionZoomIn, actionZoomOu

 float phi, theta;
 float x, y, z, speed;

 float lastX, lastY;
amControl;

fw_callback_error_( int, char co

fw_callback_key_( GLFWwindow*, i
```

Figure 14: Example of the User Interface of My Program with the OpenGL Window

As stated, the user interface is coded with the Qt library using, QLabels for any text, QSpinBoxes for the variables the user can change and a QPushButton that calls the function that opens an OpenGL window and draws a forest using the current variables in the user interface.

3.5 Limitations of Implementation

Certain aspects of my implementation were limited and could be much better, this is down to two reasons either lack of research to begin with or lack of time by the end of the project. The three main aspects I will talk about that I find to be limited in my implementation is the finitivity of the tree strings, the time it takes to create a tree/branch mesh and the forest algorithm in general.

3.5.1 Retrospective

3.5.1.1 Tree Modelling Algorithm

Even though each tree in the program is randomly made, since the strings for a tree is hard coded into the program there is only a finite number of ways a tree can look. This can look particularly bad at the bases of the trees since there are only five ways the base of the tree can look.

If I was going to do this project again with the knowledge I have at the end of the project, I would like to make every tree unique other than extremely rare circumstances. I could do this by, instead of their being a hard coded string in the code, have an algorithm that randomly produces a string with validation so that it doesn’t cause crashes and sticks to a trees genotype. Another way to do this is instead of the tree creation algorithm taking in a tree string and being recursive, the for loop is much larger, the number of branches in a string are specifically defined, and every loop randomly picks a character from the language and performs the instructions associated to it till the number of branches are made.

3.5.1.2 Mesh Creation Speed

The reason the program can take a while to display anything in the OpenGL window is because of the number of branches per tree, especially with higher fractals and because of the efficiency of the algorithm that creates meshes. Since we can’t really reduce the number of branches, as that is just the number of branches in a tree, then the best thing to do is make the mesh creation algorithm more efficient.

The way to make the algorithm more efficient is best done by making the for loop parallel and I would specifically aim to make it parallel on the user’s graphics card using cuda or OpenCL. The algorithm is already kind of made for this so wouldn’t be too hard to fix this, the reason this was not done in this project is because I lacked the knowledge of parallel processing, early on in the project.

3.5.1.3 Forest Algorithm

This mainly came down to the fact that when I got to the task of writing the forest algorithm, I realised there would not be enough time to code the algorithm how I had planned in the design; where I would've liked to have had the plane have random resource values that effect the growth of the trees in an area and how many trees are in an area for the forest. This might change the user interface where the user doesn't input how many trees they would like in the environment but rather the dimensions of the environment.

Chapter 4
Results

4.1 Forest Algorithm Speed Tests

All these tests are made on a Ryzen 5 3500XT which has 6 cores, 12 threads and a frequency of 3.8GHz. Tree types 1 and 2 are put together for the speed test as they are modelled the same with different textures so should take the same amount of time to create their meshes on average.

4.1.1 Serial Tree Type 1 and 2

number of trees	1 Fractal	2 Fractals	3 Fractals	4 Fractals	5 Fractals
1	0.006577	0.020265	0.032285	0.188623	1.129110
10	0.047136	0.127012	0.447067	2.199520	12.92400
25	0.106780	0.350443	1.213420	4.746660	34.14270
50	0.221944	0.705223	2.478290	9.885770	70.72940
100	0.456809	1.339920	4.658310	18.84750	151.1410
200	0.879733	2.626590	9.654250	36.04720	302.0540
400	1.799410	5.404090	20.05640	76.05210	665.0700

4.1.2 Serial Tree Type 3

number of trees	1 Fractal	2 Fractals	3 Fractals	4 Fractals	5 Fractals
1	0.004990	0.006218	0.008387	0.012220	0.013796
10	0.028316	0.046999	0.071003	0.093810	0.126125
25	0.067705	0.114163	0.171167	0.231870	0.310087
50	0.134297	0.229070	0.339792	0.459903	0.618205
100	0.274646	0.457603	0.681556	0.918517	1.224030
200	0.532270	0.910052	1.347220	1.808250	2.435330
400	1.06003	1.819710	2.679680	3.618780	4.886810

4.1.3 Parallel Tree Type 1 and 2

number of trees	1 Fractal	2 Fractals	3 Fractals	4 Fractals	5 Fractals
1	0.012079	0.019695	0.067875	0.672898	1.621310
10	0.103499	0.329338	0.748573	6.439182	13.23220
25	0.307634	0.948857	1.534110	15.21480	50.18840
50	0.637971	1.827040	4.013310	32.44790	175.8920
100	1.234620	3.673090	8.208320	60.41330	361.1120
200	2.520970	7.916650	15.89200	123.5380	744.7350
400	5.004250	15.51560	29.74430	250.1160	1501.350

4.1.4 Parallel Tree Type 3

number of trees	1 Fractal	2 Fractals	3 Fractals	4 Fractals	5 Fractals
1	0.005252	0.008080	0.011002	0.014480	0.018879
10	0.061305	0.102060	0.165359	0.226666	0.295867
25	0.177273	0.312540	0.440841	0.588271	0.787675
50	0.360669	0.586010	0.864932	1.154120	1.595990
100	0.750374	1.243210	1.828880	2.459560	3.319160
200	1.489940	2.580500	3.702730	4.972210	6.736240
400	3.032820	5.212580	7.446810	9.957830	13.53680

4.1.5 Observations

The first observation that is clear from these tables is that making the forest algorithm parallel makes it less efficient even though it is linear. This is likely because the algorithm was not made to be parallel to begin with and something I have tried to add near the end of the project, so the problem is not obvious and could be within the drawTree() function as there are for loops within the function meaning they could be imbedded for loops which cause parallel problems. The second observation is that tree type 3 takes much less time to be created, this is because it requires less branch meshes to be created per tree.

4.2 Visualisations of the Forest and Trees

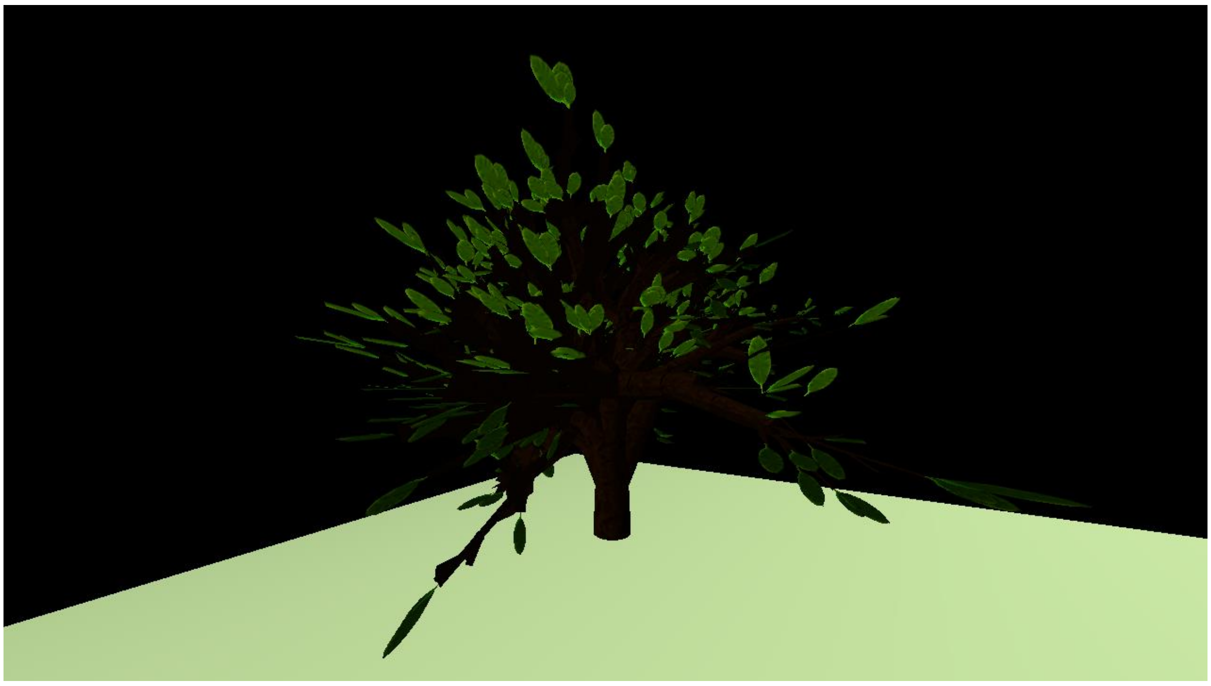


Figure 15: Example of Tree Type 1 with 5 Fractals, Branch Length of 2.5 and Radius of 0.7

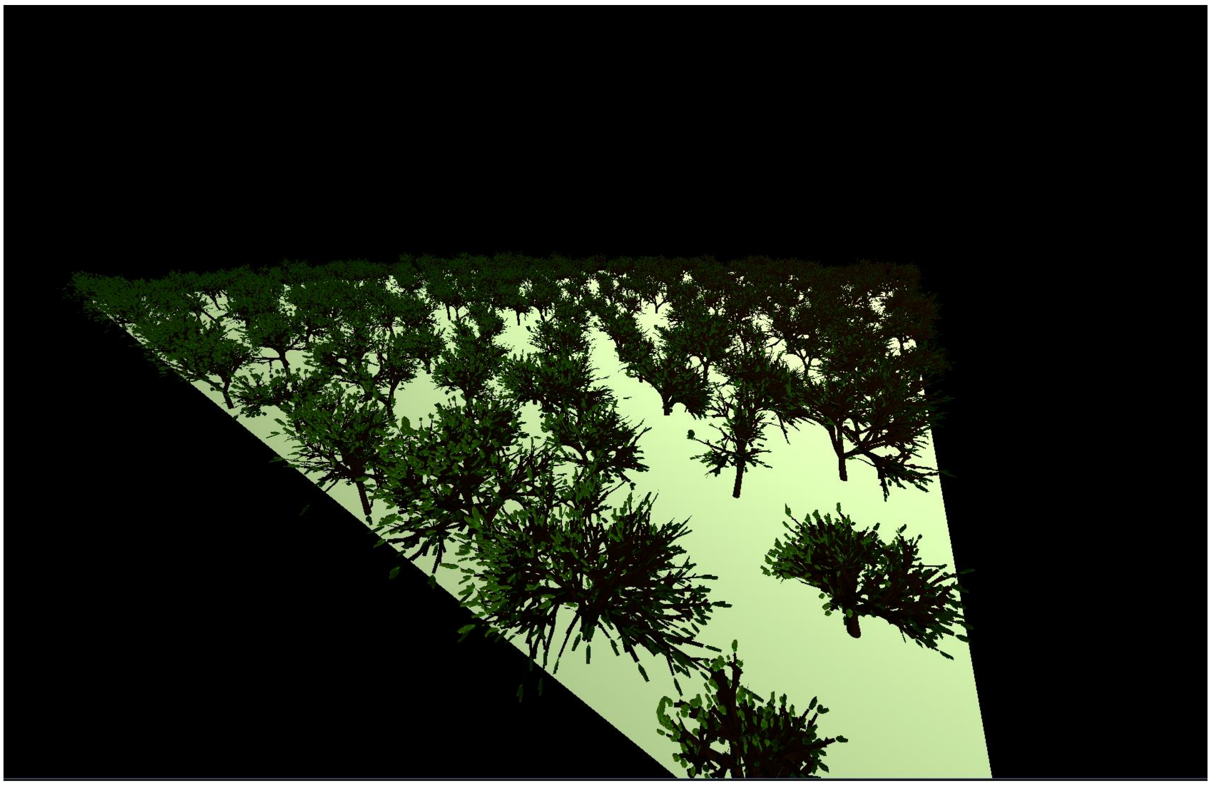


Figure 16: Forest of 400 Trees of Type1, with 5 Fractals, Branch Length of 2.5 and Radius of 0.7

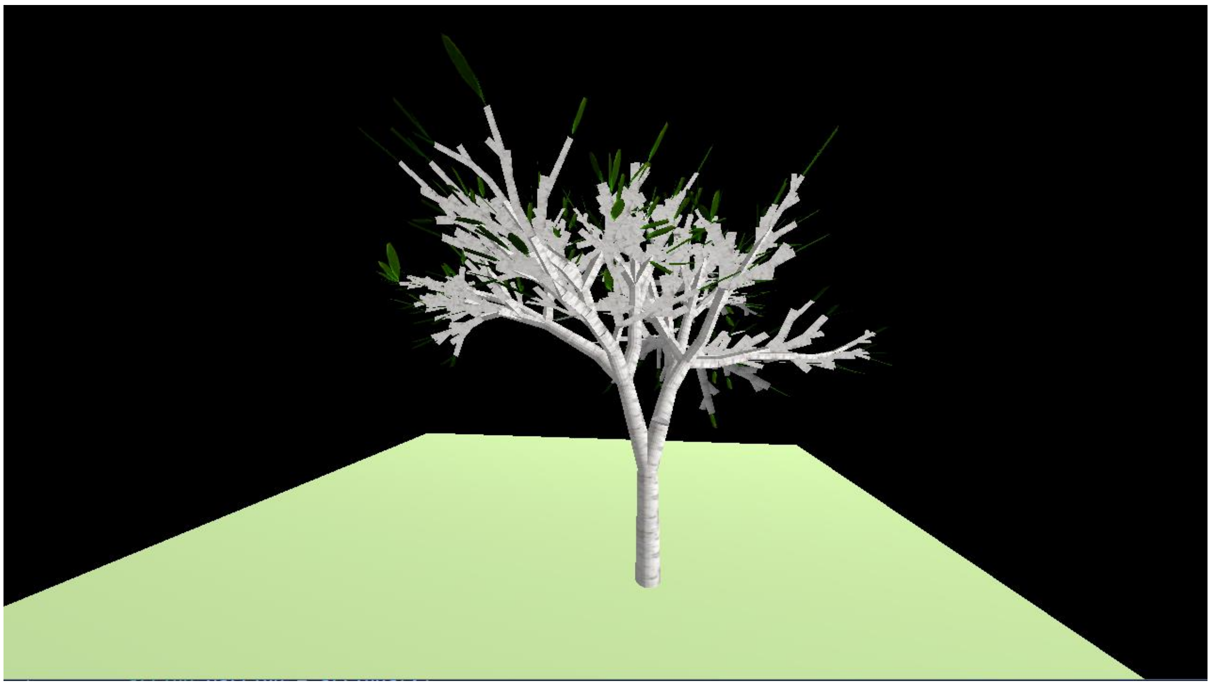


Figure 17: Example of Tree Type 2 with 5 Fractals, Branch Length of 2.5 and Radius of 0.7

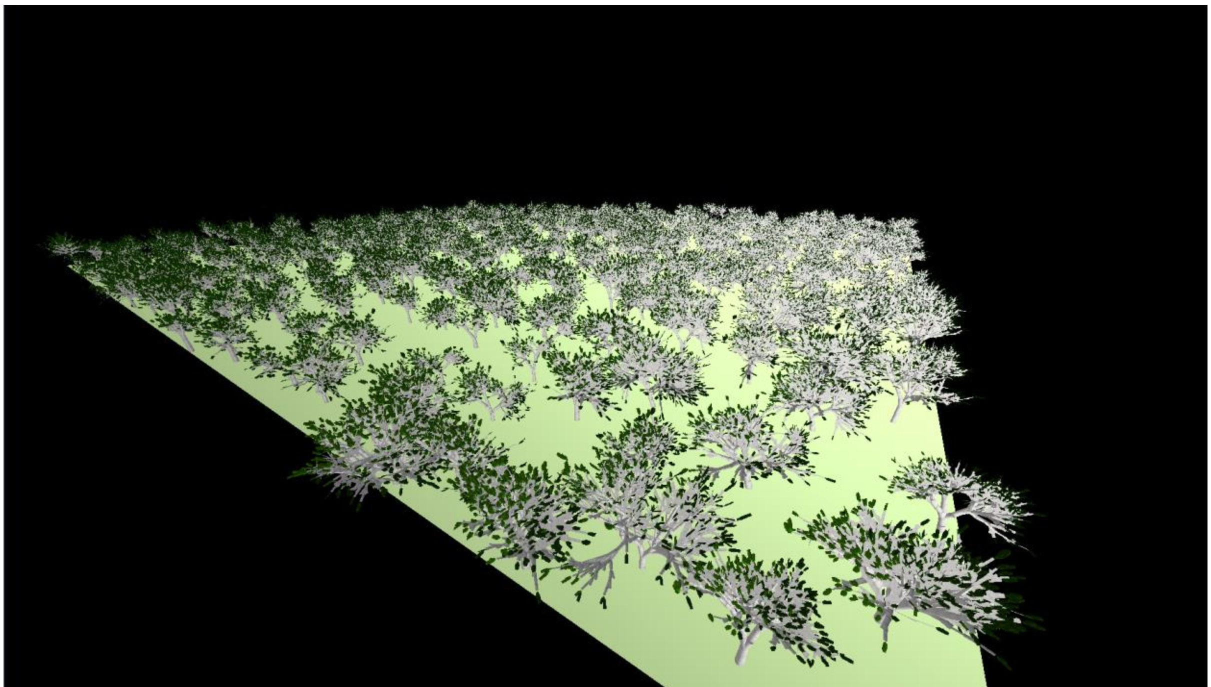


Figure 18: Forest of 400 Trees of Type 2, with 5 Fractals, Branch Length of 2.5 and Radius of 0.7

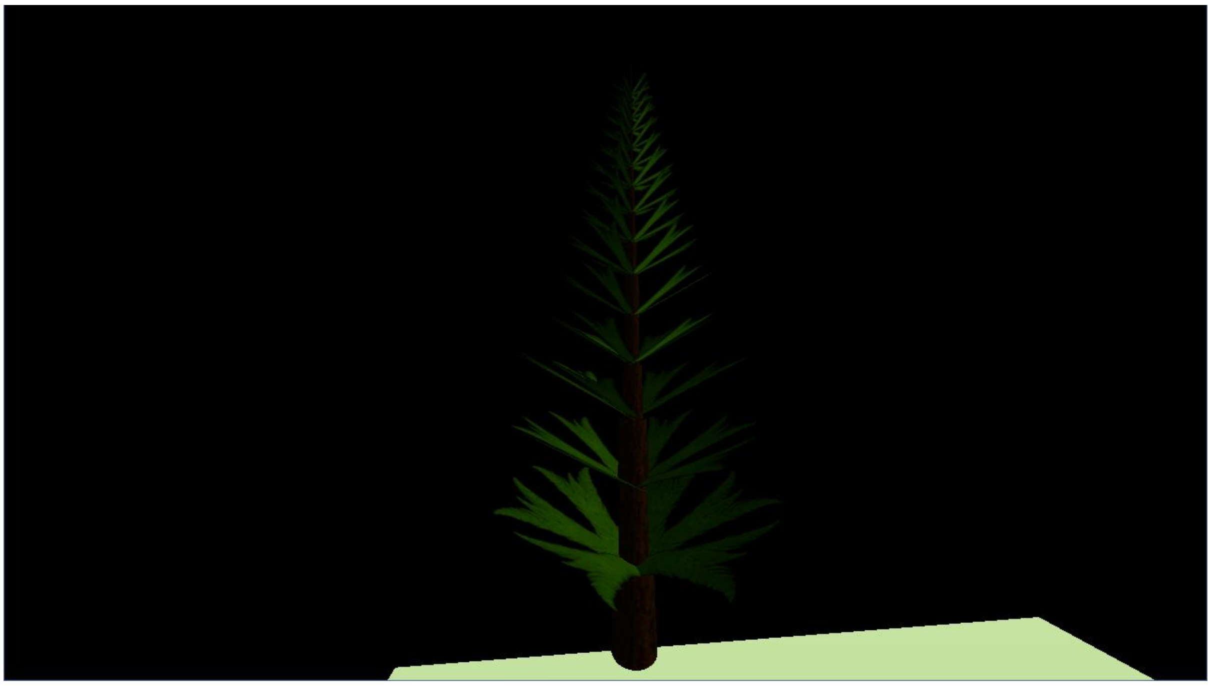


Figure 19: Example of Tree Type 1 with 5 Fractals, Branch Length of 2.5 and Radius of 0.7

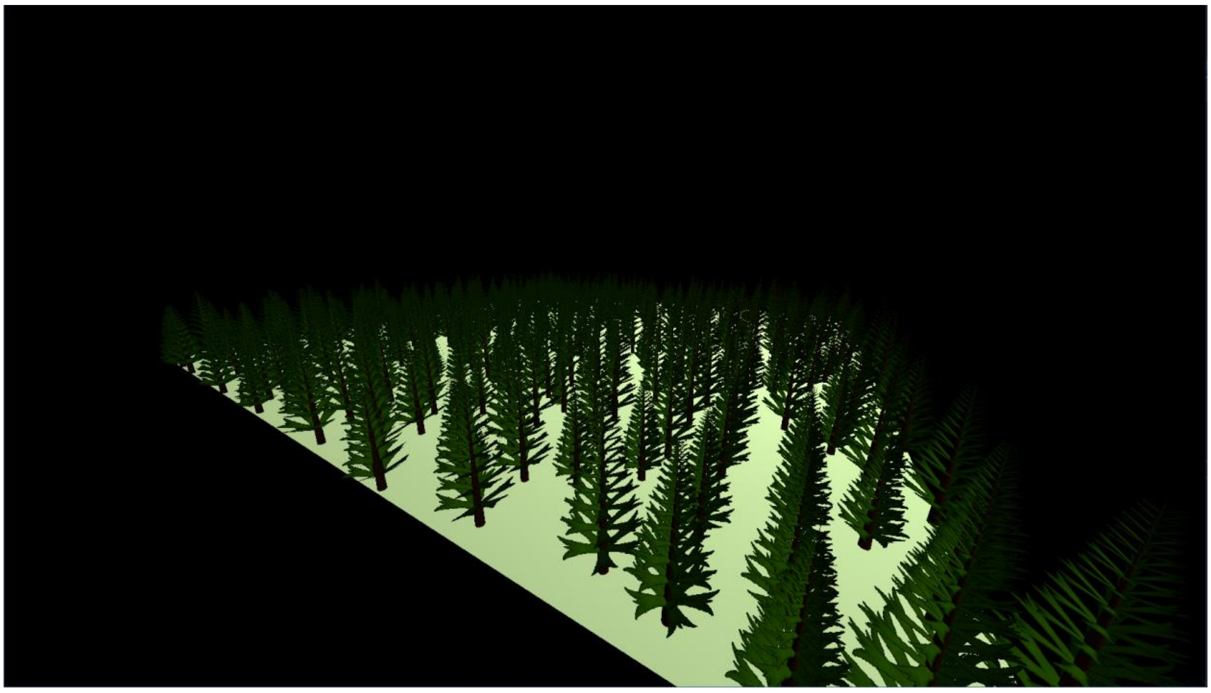


Figure 20: Forest of 400 Trees of Type 3, with 5 Fractals, Branch Length of 2.5 and Radius of 0.7

Chapter 5
Discussion

5.1 Conclusions

For this project I had four main objectives; to create a piece of software that randomly generates a tree, to have this piece of software be able to create at least three distinctly different tree types, then for the software to be able to render multiple of these in a forest and finally for the user to be able to interact with the software with a user interface I programmed. Throughout this project I have described how I have successfully tackled each of these objectives with visual evidence. So, in conclusion this project was successful, and I have achieved everything I had set out to do.

5.2 Ideas for future work

This project has made me fall in love with 3D graphics and procedural modelling and I can't wait to work with these further and have already thought of future plans.

The idea I would work on first is adding different plant types to the forest such as flowers, bushes and grass all being procedurally generated. I would then like to work on the terrain making it not just flat but with curves and then have the forest algorithm be able to make forests on these new planes.

I would also tackle the limitations my software currently has which I talked about in section 3.5.1, possibly rewriting the software in a more efficient manner.

I could also improve the user interface allowing the user to be able to interact with more options that change the forest, even possibly allowing the user to write their own strings in the user interface for trees.

List of References

[1] Przemyslaw P. et al. 1990 The Algorithmic Beauty of Plants. Electronic version. New York: Springer-Verlag.

[2] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology, 18:280–315, 1968.

[3] B. B. Mandelbrot. The fractal geometry of nature. W. H. Freeman, San Francisco, 1982.

[4] Wolfram. Rotation. [Online]. [Accessed 20 January 2023]. Available at: https://mathworld.wolfram.com/RotationMatrix.html

[5] Oliver D. et al. Realistic Modelling and Rendering of Plant Ecosystems. [Online]. [Accessed 25 October 2022]. Available at: https://dl.acm.org/doi/pdf/10.1145/280814.280898

[6] Firbank L.G. A Model of Interference Within Plant Monocultures. [Online]. [Accessed 25 October 2022]. Available at: https://www.sciencedirect.com/science/article/pii/S0022519385802691

[7] Opengameart. Tiling Bark Textures. [Online]. [Accessed 17 February 2023]. Available at: https://lpc.opengameart.org/node/10052

[8] Google. Birch Tree Art. [Online]. [Accessed 17 February 2023]. Available at: https://www.pinterest.co.uk/pin/391672498820505414/

[9] Divizia C. Walnut Tree Leaf Transparent. [Online]. [Accessed 23 February 2023]. Available at: https://www.vecteezy.com/png/8541808-walnut-tree-leaf-transparent-png

[10] Arnold K. Fern Leaf Clipart Illustration. [Online]. [Accessed 14 April 2023]. Available at: https://www.publicdomainpictures.net/en/view-image.php?image=449931&picture=fern-leaf-clipart-illustration

[11] de Vries J. Basic Lighting. [Online]. [Accessed 14 February 2023]. Available at: https://learnopengl.com/Lighting/Basic-Lighting

[12] Math is fun. Dot Product . [Online]. [Accessed 14 February 2023]. Available at: https://www.mathsisfun.com/algebra/vectors-dot-product.html

Appendix A
Self-appraisal

A.1 Critical self-evaluation

As stated in the conclusion of the project I believe I was successful in obtaining all my main objects for the project. This does not mean it was perfect. My main mistake for this project was a lack of research before I started implementing the software. The subjects I believe I could have researched more thoroughly are the forest algorithm, GUI libraries as Qt wasn't optimal and took a long time to integrate and machine learning for the tree algorithm.

A.2 Personal reflection and lessons learned

My main take away from this is if I do a project like this again put more time into the plan and research to save time and improve the quality of the end project. I have also learned a lot of useful techniques that can be used in many scenarios in computer science even those that are unrelated to this topic. These can be to do with branched structures and certain 3D techniques that I have demonstrated in this report.

Looking back on this project I have enjoyed it loads, some moments have been stressful because of errors but mostly enjoyable. This project hasn't just taught me more complex ideas in computer science but also made me love coding again especially when it comes to graphics.

A.3 Legal, social, ethical and professional issues

A.3.1 Legal issues

There could be possible copyright issues with my project as I have used external material from the internet, but I have referenced all material that is not my own work so this should not be a problem

A.3.2 Social issues

My code is only written in English as this is the only language I am fluent in and probably would have lacked the time to add this so any none English users may not be able to use my software. Though there is limited amount of text and the UI isn't too complex so a user may be able to figure it out and the main purpose of the software is graphical which is universal.

A.3.3 Ethical issues

The largest ethical problem with my work comes down to the environmental cost to running it as it is computer program it will use energy and since it is graphical software then it will require lots of energy and some could argue that the software is pointlessly damaging the environment.

A.3.4 Professional issues

Since the software's implementation isn't completely up to date with industry standards then, if anyone used this in a professional manner they may run into issues that could cost them a lot of time and therefore possibly money to fix this.

Appendix B
External Materials

B.1 Figures and Images

Figures 1,2,3,4: http://algorithmicbotany.org/papers/abop/abop.pdf

Figure 9: https://lpc.opengameart.org/node/10052 and https://www.pinterest.co.uk/pin/391672498820505414/

Figure 10: https://www.vecteezy.com/png/8541808-walnut-tree-leaf-transparent-png and https://www.publicdomainpictures.net/en/view-image.php?image=449931&picture=fern-leaf-clipart-illustration

B.2 Code Libraries

OpenGL library: https://www.opengl.org/

Qt Library: https://www.qt.io/