**Project 8: Simple Demo Scene**

Benjamin Carter and Josh Canode

Grand Canyon University

CST-305: Computer Graphics

Professor Ricardo Citro

November 19th, 2023

**Project Description and Main Idea**

When deciding on what simulation to design, we settled on making a realistic forest simulator. As one of us was also talking a differential equations class at the same time, we wanted to create a useful simulator modeling something in the real world. So, this project is a forest simulator. It demonstrates the life cycle of a forest, with tree growth, wildfires, burned trees, and then the cycle repeating again.

**Video Here:** [**https://codingcando.com/fileShare/file?code=TDAX62YKEW**](https://codingcando.com/fileShare/file?code=TDAX62YKEW)

[(If above doesn’t work, click me)](https://mygcuedu6961-my.sharepoint.com/:v:/g/personal/bcarter44_my_gcu_edu/EcoKg01SOVpBp8UNkGK0RC0BzII-QxfLv7WPFytuGMGSlg?e=6xwcXt)

**Why is it interesting to watch? Challenging to design? Tricky to program?**

Using a random number generator, the forest expands and contracts in different ways each time the program is run. This gives something “new” to see each time the program is run. Also, as fires quickly spread from one tree to another, one can watch as fires sweep across, moving in a chaotic pattern. This simulation is mesmerizing as it captivates a user to see if the forest fire will wreck havoc to the forest, or if the forest can withstand the fires and grow back quick enough.

To program this, we implemented aspects from mathematical graphs to model state changes in a forest, which was tricky to design and program. As this program uses many mathematical concepts, it was difficult fitting them all together to get them to produce the desired result.

**Theoretical background**

A forest is composed of trees and open spaces. If a forest was divided into a grid system, there would be cells with a tree and then cells that are empty. Out of the cells that have a tree, the tree can either be alive, burning, or burned. This leaves four states per cell: an empty cell, an alive tree, a burning tree, and a burned tree. This way, the main portion of the program deals with triggering “state changes”, or calculating when a cell in one state changes into another states. This can be then represented as a mathematical graph:

**Forest State Graph**

A black background with white ovals

Description automatically generated

In the simulation, it runs a simulation script for each cell. That simulation script through a random number generator will call the appropriate state change.

**Mathematical concepts**

**Differential Equations**

From the forest state graph, the simulator models a system of differential equations. The quality of any state depends on the quantity and percentages of the previous state and the transition phase respectively. This makes the following differential equation:

Then, each value is supplied with the forest settings struct (see ahead).

**Random If with FPS adjustment**

Central in this project is random decisions based off of percentages. However, one cannot just pass in the original percent, as the function can be called many times per second. If one calculates a 5% chance 20 times a second, that is equivalent to 100% per second (with it only happening at least once, independently drawn, so probabilities add). But, if the same was done on a machine running at 5ps, this then is equivalent to 25%. This means that there are differing output percentages when the frames per second changes. So, to make a random if with FPS adjustment, all one needs to do is divide the original percentage by fps.

A computer screen shot of a program code

Description automatically generated

**Programming implementation**

***See the index.html page in the Doxygen folder or the PDF for more.***

**Trees Class**

The core of the program is the **Tree** class. This includes tree properties data, information about neighbor data, and functions for drawing, simulating, and managing lightning. Top of Form

There are four states a tree can be in, noTree, alive, burning, or burned. Each tree state has a set of functions to manage the state. There are also methods that switch from one state to another when certain criteria have been met, for example when a tree is completely done burning it transitions from burning to burned which changes the color and several other attributes of the tree.

The Tree class also has a draw loop that is called in main to draw the trees for each frame. It also draws lightning strikes.

**ForestAnimationSettings**

Simulation and Animation setting are stored in a struct ForestAnimationSettings.

This keeps track of critical info such as how the fire spreads, tree growth and more.

A screenshot of a computer program

Description automatically generated

**Mesh**

The mesh is responsible for the creation of a 3D mesh representing the ground. The mesh is formed by dividing a specified region into a grid, and the shape of the ground is determined by a function. The class contains functions for setting up the mesh, retrieving top points, cube points, and side points, as well as calculating the square width of each division. The mesh setup involves populating arrays with vertices to form triangles for both the top and sides of the ground.Top of Form

A computer generated image of a brown wavy surface

Description automatically generated

**TextWriter**

The TextWriter is used to help with writing all of text which displays the current status of the simulation.

A screenshot of a video game

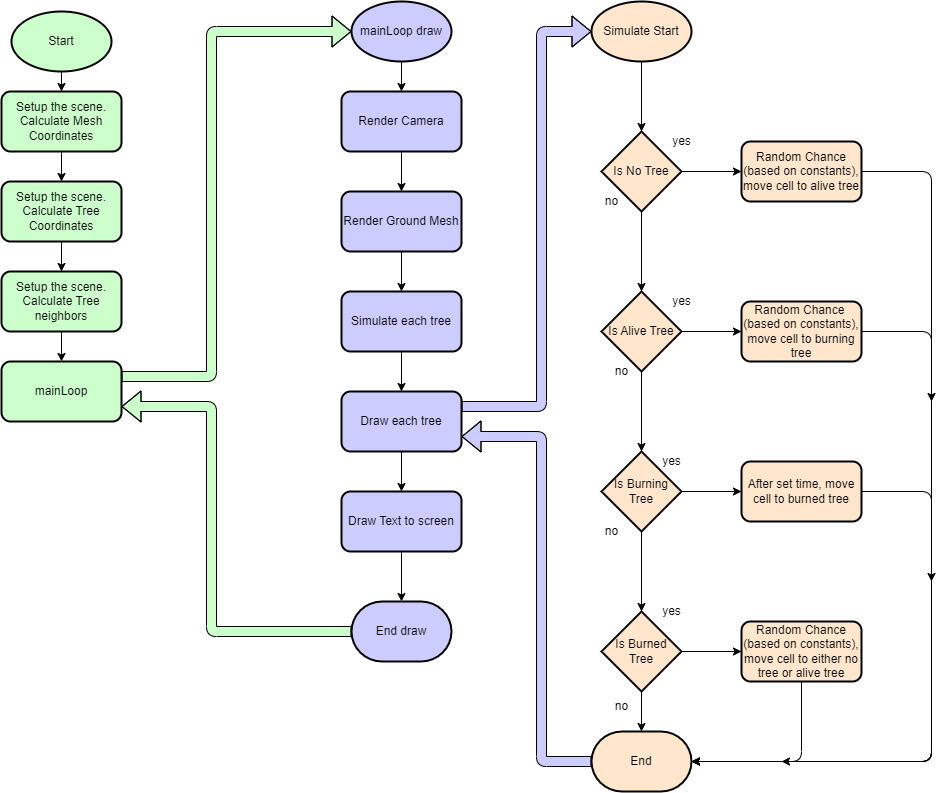
Description automatically generated

**Main**

The **main** function is how the forest lifecycle simulation is run. It defines various parameters such as window size, position, and display mode. The initialization of global variables, including the ground mesh (**groundMesh**), camera position (**camPos**), and tree settings (**forestSettings**) is carried out to set the initial state of the simulation. The program then establishes callback functions for rendering (**frame**), window resizing (**reshape**), animation timing (**timer**), and idle processing (**idleFunction**). The initial state is created in the **setupCalculations** function, where the ground mesh is initialized, and the array of trees (**allTrees**) is created with respective positions and dimensions. This is done first to avoid doing much of the calculation during the frames. The program enters the main loop where the simulation is continuously rendered, animated, and updated based on the defined callbacks.

Top of Form

**Flowchart**

****

**Screenshots**

A screenshot of a computer generated image

Description automatically generated

The forest faces its first fire.

A screenshot of a video game

Description automatically generated

The forest regrows after the fire

A computer screen shot of a computer generated image

Description automatically generated

The forest catches fire again.

A screenshot of a video game

Description automatically generated

Lightning Animation