Christopher Walen
CAP 4720, Fall 2019
Prof. Pattanaik
12/10/2019

Procedural Scene Project

Purpose:

The purpose of the project is to procedurally or otherwise randomly generate scenes that the user could see and walk through using the First Person Camera controls, or Flight Controls (as the First Person Camera controls support both methods of control). Of the four biomes that were proposed, three made it into the final submission - forest, mountains (previously caves), and island. All models and textures were created by myself for the project, with a focus on low-polygon and vertex count to ensure quick loading, efficient run-time, and a reduced file-size footprint.

How to Use:

To experience the Procedural Scene, host a local server to the directory that contains "Generator.html", then open your local IP at the port you're hosting in your web browser and navigate to the Generator.html file, opening it in your browser. Upon loading you will see one of the three biomes, which will be procedurally re-generated upon every refresh of the page. You can fly around the area to observe the scene from different angles using the First Person controls. As this project uses THREE.js as its primary source of rendering and model handling, the First Person Camera uses the THREE.FirstPersonCamera() controls, which are as follows:

Key/Mouse Button	Action
W	Move forward from camera vector
S	Move backward from camera vector
A	Move perpendicular to the left from camera vector
D	Move perpendicular to the right from camera vector
R	Move along the camera's up vector
F	Move along the camera's negative up vector
Left Mouse	Move forward from camera vector
Right Mouse	Move backward from camera vector

Production:

The customizable aspect of the Procedural Scene was not able to make it into the project due to time constraints, and a severe impact on performance that occured upon a single change to the settings through the dat.gui functions. This has been simulated however, by supplying the generation with 6 different sun light colors, up to three models and textures for each object type in each scene, with over 392 possible positions in the smaller biomes (this is to ensure that there is little overlapping of the models). Each model may be randomly rotated and shifted further from these positions, ensuring that no two runs of the program give you the same scene (within a reasonable amount of time).

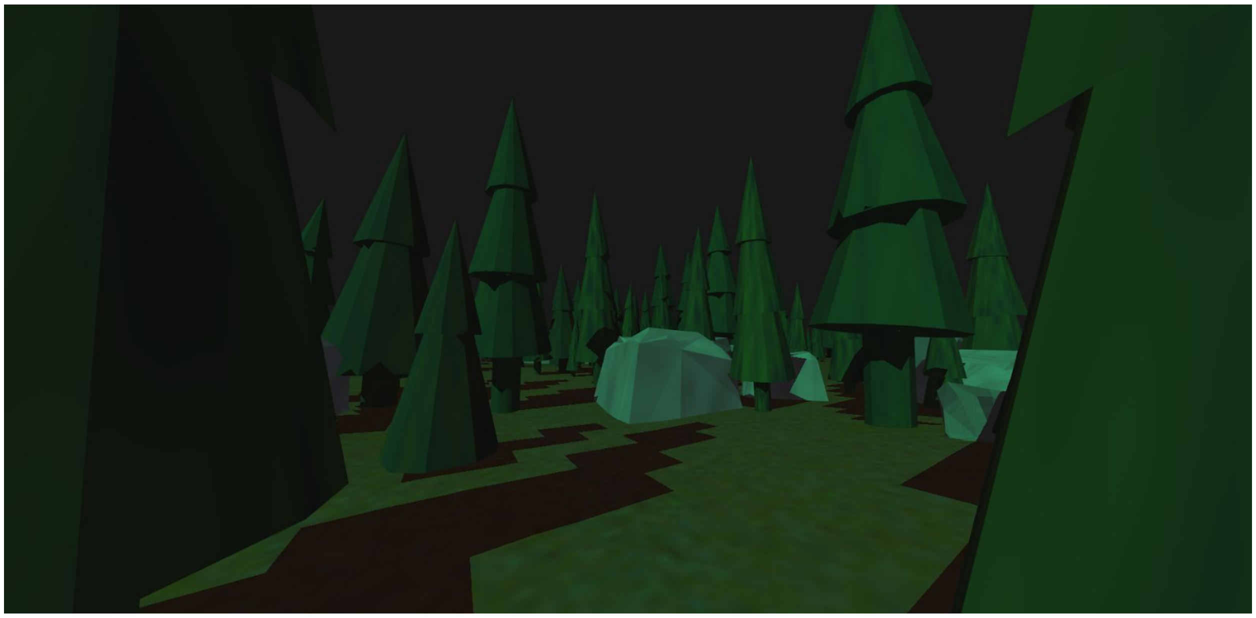
This is achieved mainly with crafty usage of JavaScript's Math.random(), as most procedural generation is. The algorithm is not very special in this version of the Procedural Scene, as I could not find a quick way to grab the y-values from the floor models at runtime, and Blender gave me trouble when trying to generate normal maps for use as height maps. The plan in this regard was to look at the alpha value from the height map, looping through each pixel in the height map corresponding with the vertices on the floor model, and placing the terrain models at the correct height so that they sat flush with the floor model. This unfortunately had to be faked, by extending the models of the terrain objects so that they had vertices below the (0,0,0) origin, and by keeping the areas of each floor model relatively flat where generation would occur. The two combined allows for a visual that appears as if all of the terrain objects are indeed connected with the ground model, but in fact are separate.

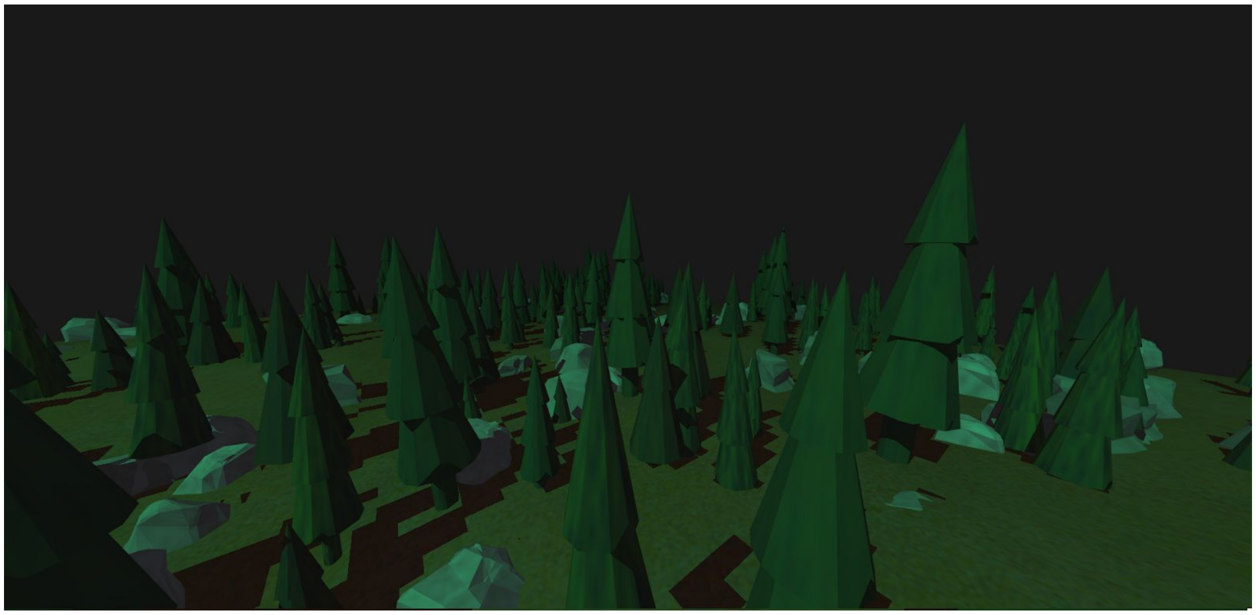
When levels are created, first we create the necessary components such as the THREE.scene and THREE.renderer, and define some default values for generation. After we define the lighting, the level generation takes place by first looping through all of the allowed positions that an object can be placed in in the 3d space, then checks against the probability that we described earlier. If we succeed, then we move on to calling the correct biome's object placement function, feeding it the position of the 'cell' we're currently at. After we finish this loop, we then place the floor model at the scene's default position, and set up the camera. Once we've set everything up, we call the render() function until the page is closed.

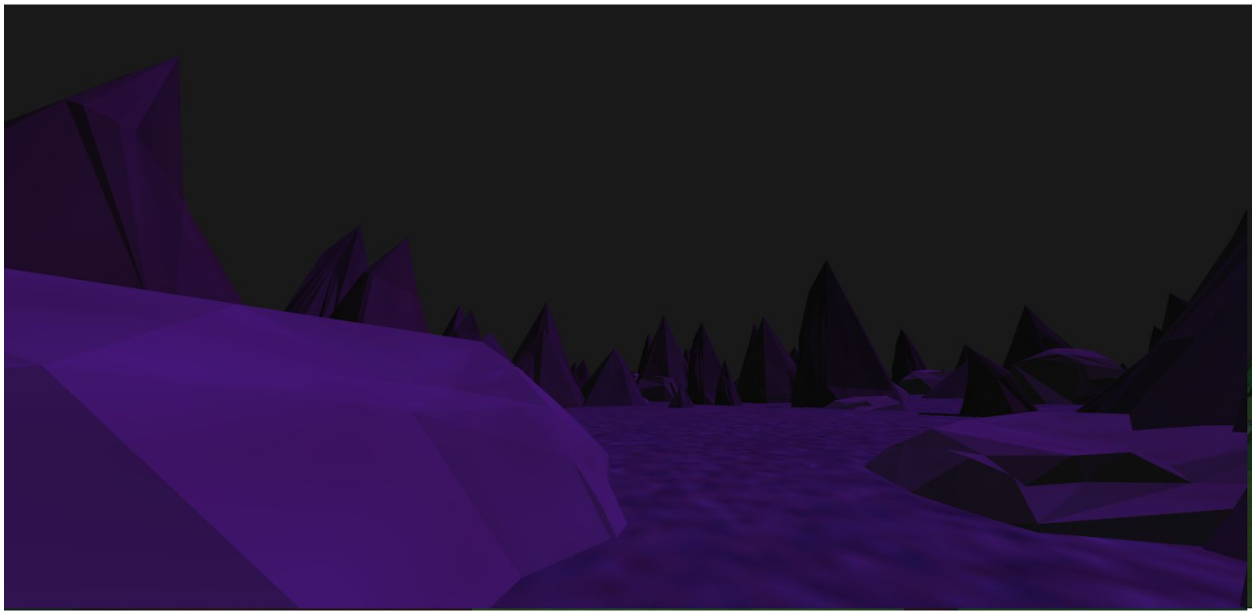
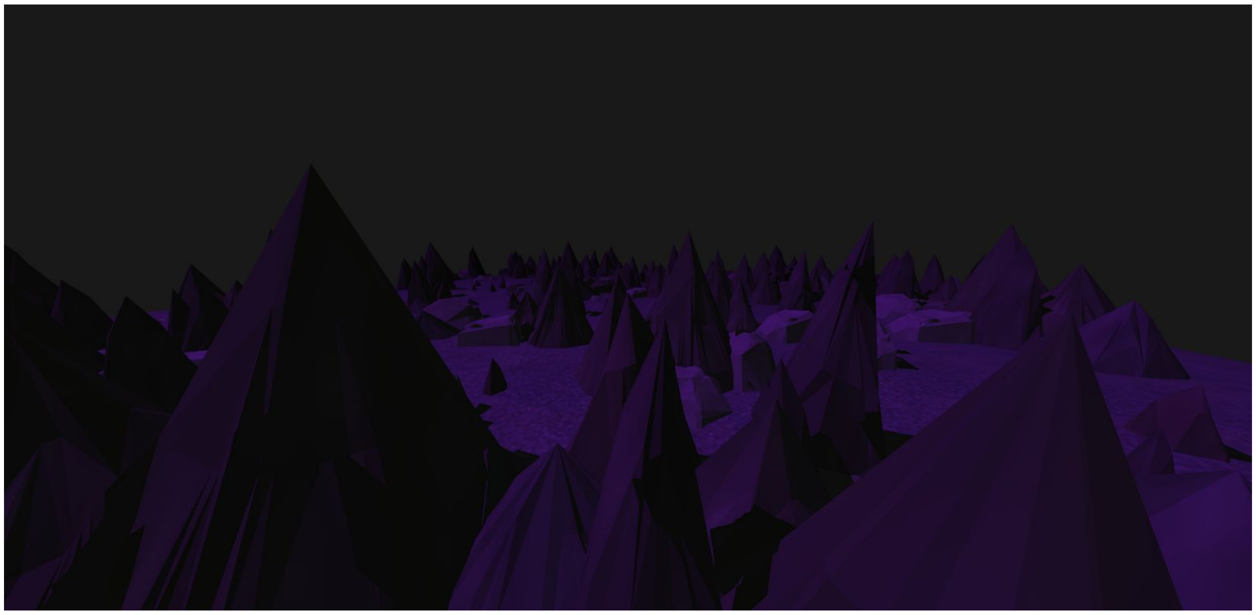
In my proposal I also mentioned that I was going to try to generate the .obj models themselves in the javascript program, though this was on the condition that I found enough research that pointed in the right direction to allow this. After examining several pages of documentation from various sources, as well as examining the .obj files in their text forms, I determined that in order to achieve what I had hoped for would require much further examination and careful algorithm design that would not fit within the time allotted to the project. The algorithms would have to describe the various objects that I would be creating in the 3d-environment, for example an algorithm that describes the shape of a tree and/or simulates a tree's "life" persay to get the correct shape. It was far more time-efficient and productive to create the models by hand, in a generic and efficient way that benefitted the constraints of the program.

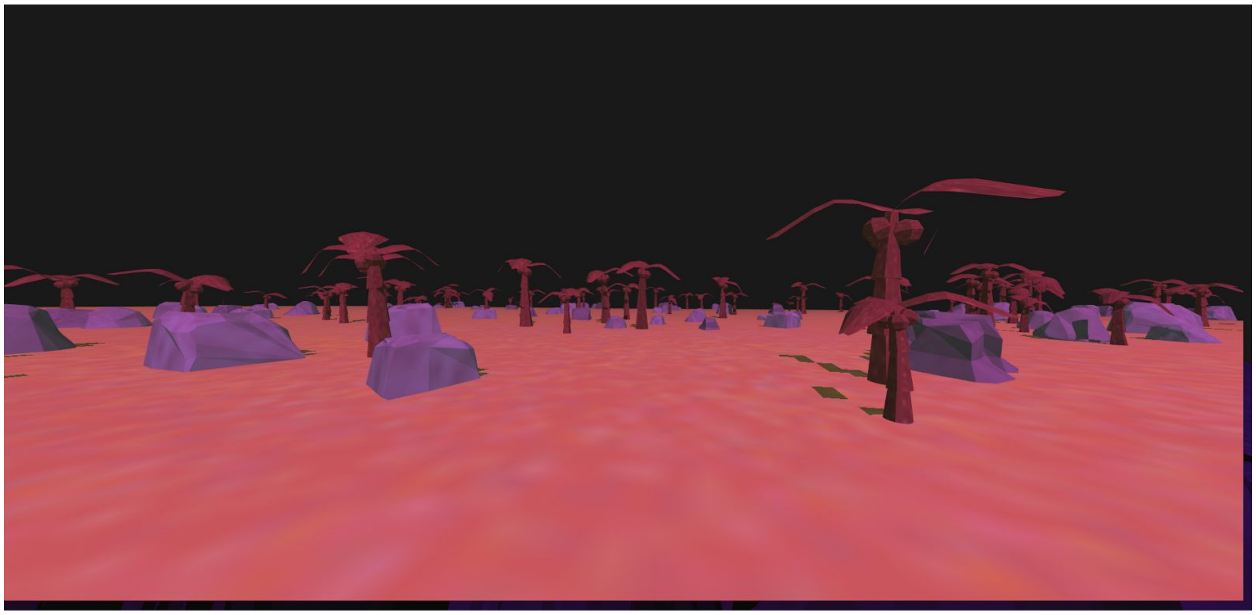
Results:

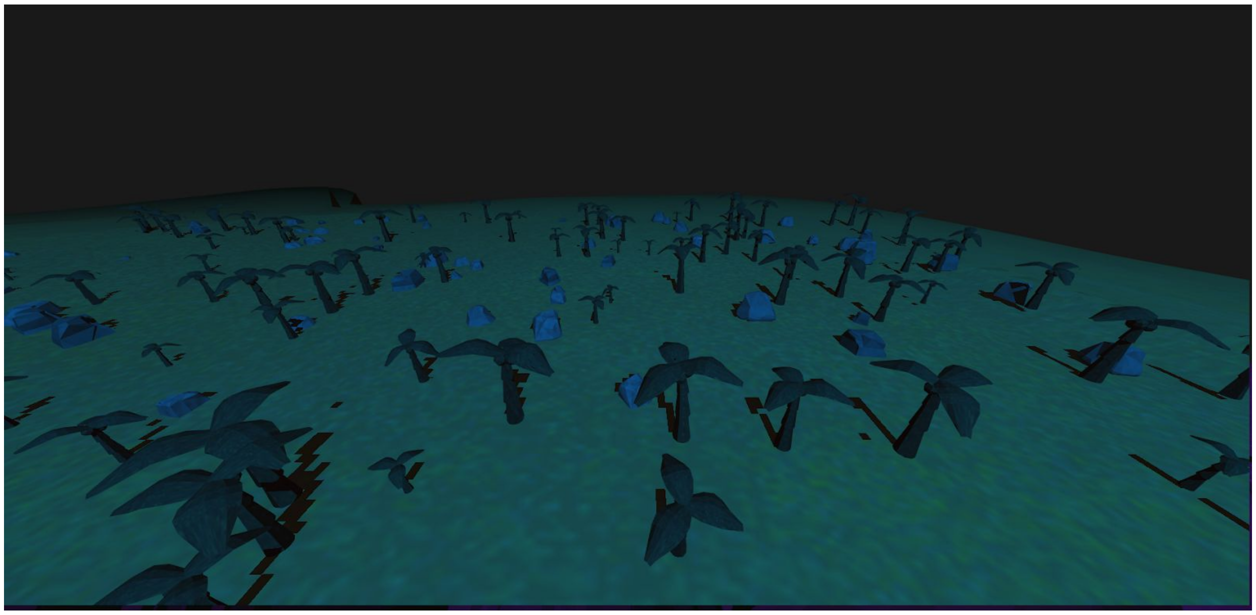
The scenes have at least 50 generated models with their corresponding textures, and THREE.js materials for all of them. In order for the forest, the mountains, and the island, this ends up looking like this:



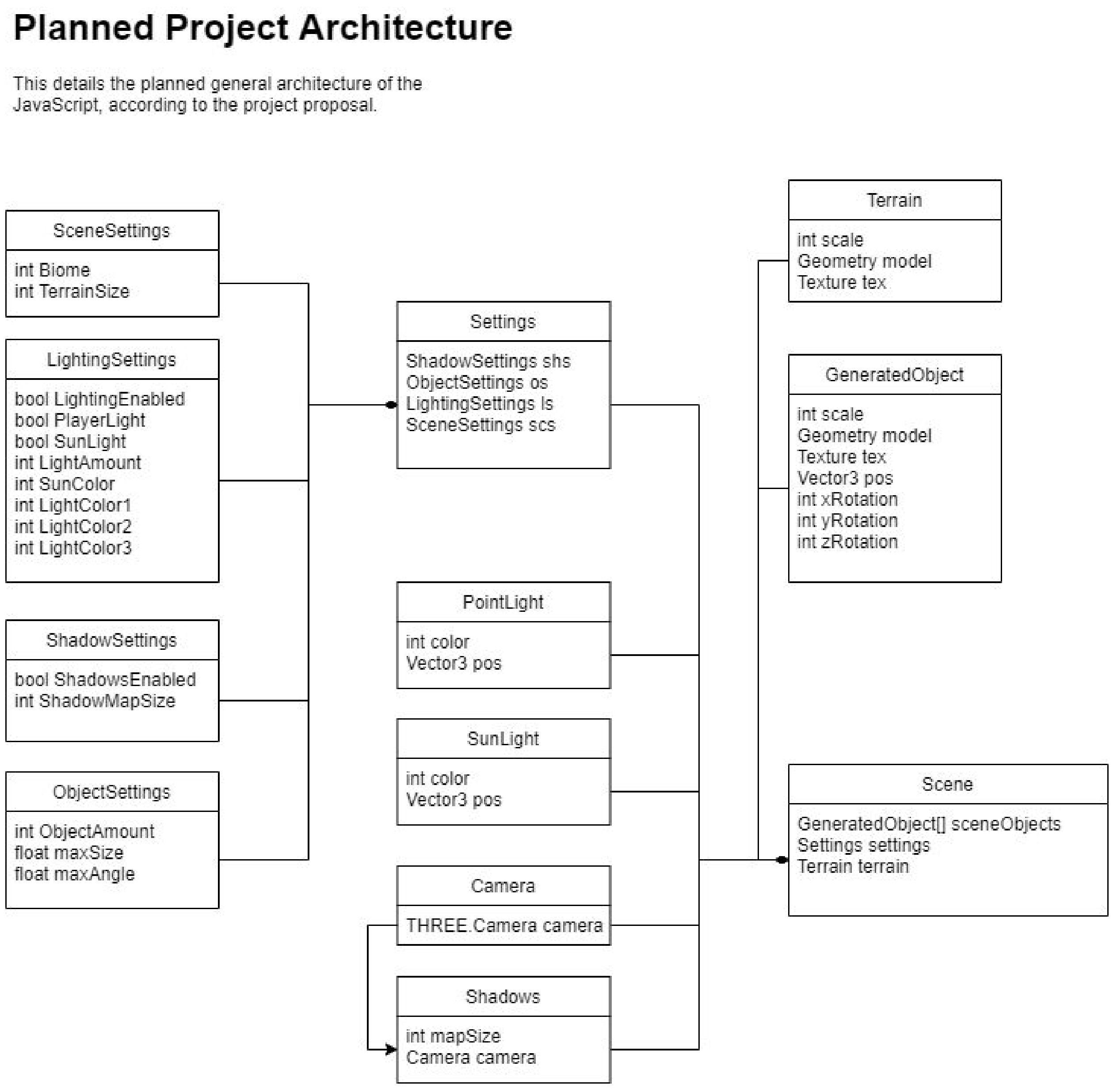




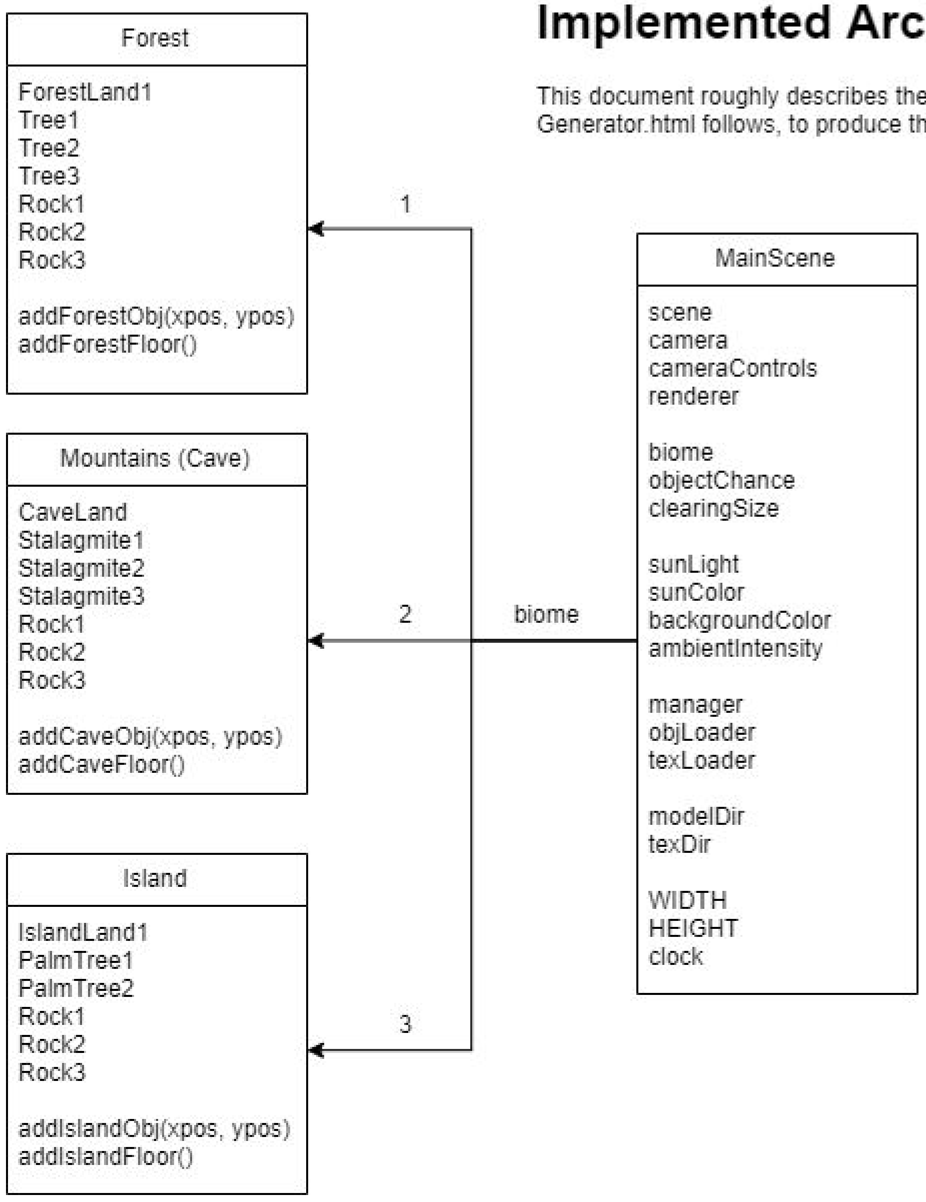


I have also included some brief architecture designs that I created going into the project, and upon satisfactory completion of the generation.



Implemented Architecture

This document roughly describes the structure that the script in Generator.html follows, to produce the Procedural Scene.



Copy of Proposal:

Below is a copy-paste of what was submitted for the project proposal.

Project Proposal: Unseen Landscape
Written By: Christopher Walen

Goal:

To create a procedurally generated 3D environment, that the user can explore and experience new placements and objects on each generation.

Objective:

To demonstrate that procedurally chosen and generated 3d environments can look as good as manually constructed scenes, and can be used for a variety of applications, namely in this case video games.

Personally I am very interested in the field of procedural generation and its applications, so even if this project proposal is rejected I would like to create a project that uses procedural generation in some way.

Planned Resources:

I have two options in mind, with the plan to have a toggleable setting for both:

A. Some simple models created by myself or found online to represent various aspects of terrain - trees, buildings, lampposts, etc. - to then build a scene out of, or

B. Explore aspects of generating models within the program itself that can be stored in exportable .obj files, to then add to the scene to create abstract and/or realistic scenes that the user can explore.

Both of these options would include customization that the user can toggle or modify, including things such as:

- Shadows on/off
- Lights on/off
- 'Player light' i.e. a light that follows the camera's position
- Color pallette for:
  - primary colors
  - secondary colors
  - sky/ground colors
  - tree colors
  - lights
- Scene object scale
- Scene object variety and chance (such as the ratio of trees planted in the generated world)
- Basic scene type (island, forest, cave, city)

If extra time is available, a music track for each type of scene would be provided to enhance the user's immersion into the scene. This is not necessary and may not be in the final product, regardless of which plan is followed.

General Plan:

Some basic research on the generation of .obj files that could be used by the program and/or exported for future use. This would affect the project's track greatly, and as such I propose a main plan granted that the research is fruitful, and a plan in case the research produces no valid results for the time frame.

Main Plan:

To allow two types of scenes to be created: one assembled from premade, publicly available assets such as trees, cars, bushes, pools, stalagmites, etc. to assemble a realistically convincing scene using procedural generation according to the user's defined theme (of which a list would be provided), and another scene where even the models themselves are procedurally generated to allow for more surreal and 'random' environments to be created. This is the ideal scenario, as the latter scene is the one I am more interested in producing.

The primary scene would work as follows:

1. Assemble a list of environments to generate: island, forest, city, cave
2. Assemble a list of objects that would be expected to appear in the list in each environment
3. Describe, write, and test algorithms to place scene objects for each environment
4. Describe, write, and test algorithms to place lighting for each environment based on certain parameters, e.g. time of day
5. Describe, write, and test generation algorithms to run the object placement algorithms for each environment. These would then be called and executed upon the user's decision of a scene type.

The secondary scene would work as follows:

1. Describe, write, and test algorithms to produce various .obj files to act as the various scene objects for primary settings, i.e. island, forest, cave
2. Describe, write, and test algorithms to place the generated objects in the scene
3. Describe, write, and test algorithms to place lighting for the environments
4. Describe, write, and test generation algorithms to place both objects and lights in the environment.

Backup Plan:

To allow for various types of the primary scene to be created, this being the scene that is assembled only from premade, publicly available assets. This would require more memory, but could still produce usable results that would still offer an interesting and engaging environment to explore. Without the requirement to create the secondary scene, more assets can be compiled for the primary scenes, which would allow for greater variety to be present in the created environments, allowing for a much more interesting scene to be presented to the user to explore.