CS 175 Project Write-up

Tingda Wang and Meng Dong

December 2018

1 Introduction

In this project, we constructed a scene of grass and trees, with panoramic city view rendered using skybox as background. The basic structure of our code is presented in Figure 1.

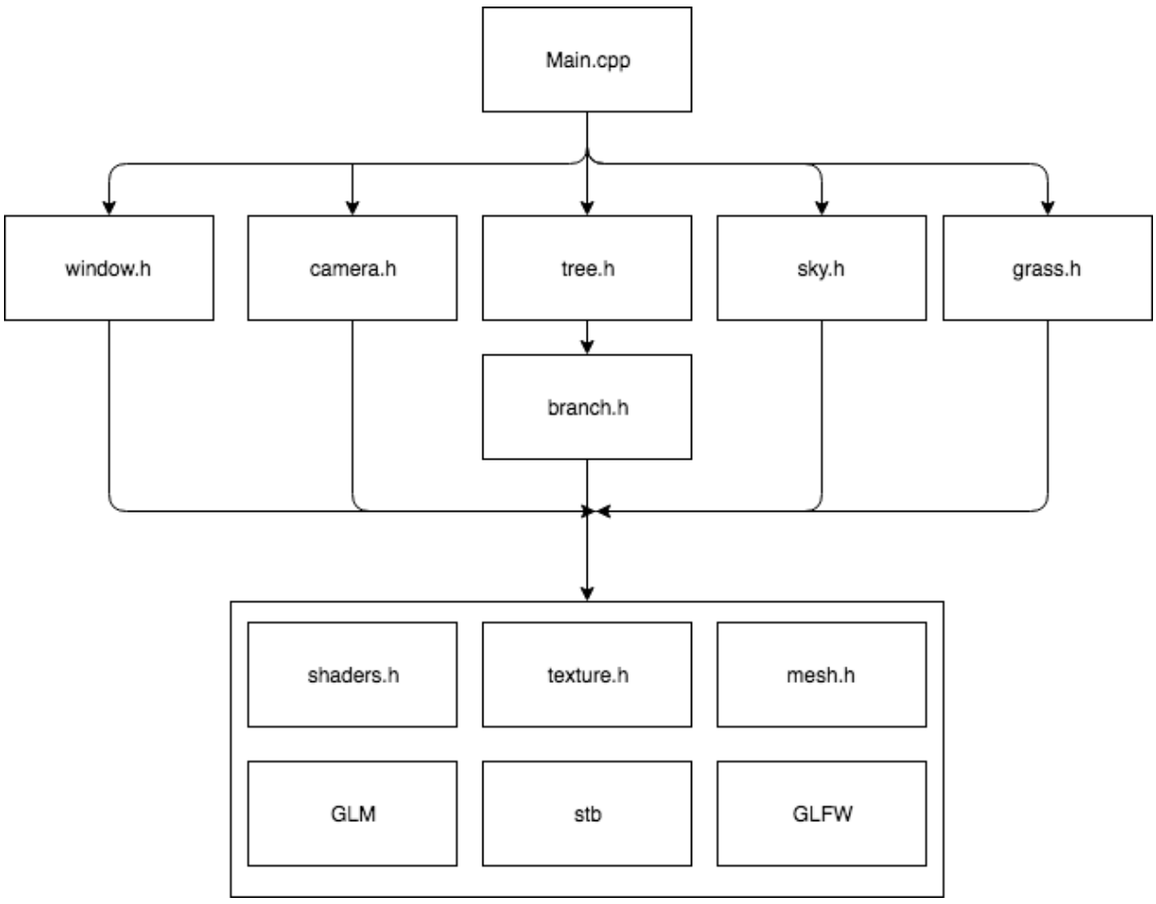


Figure 1: Structure of Code Base.

1.1 Infrastructure

In this project, we wanted to build something on our own, instead of using the infrastructure that we built in class. Therefore, we tried to use different packages such as OpenGL3, Glew, GLFW, GLM, instead of GLUT.

Assisted by another open source repository [1], we rebuilt the infrastructure modules for using modern OpenGL, namely the Window, Camera, Shaders, Texture and Mesh modules. Constrained by the short time frame, we limited our efforts to focus on thinking and including what is necessary to create a natural landscape simulation. Then we created our scene above the new infrastructure.

1.2 How to Run

This project was implemented on macOS with Glew, OpenGL 3, GLM. We did not test this on other machines or OS. To run, `cd` into `src` and `make`. Then run `./main`

Following operations are available:

- R: Toggle the rotation of world view

[1] https://github.com/erickTornero/Computer-Graphics/tree/master/Camera

- Right/Left Arrow: Change the rotation speed
- W: Zoom in
- S: Zoom out
- A: Move camera to the left
- D: Move camera to the right
- L: Toggle leaves
- G: Grow trees upward
- B: Grow trees downward
- M: Adding more trees (Three instead of one for now)

2 Tree

With pre-defined height and width, trees are generated by DFS at each height level. We first tried to render the branches then the leaves. Each tree object is created with initialization of its depth, height, branch texture, leaf texture, x-coordinate, z-coordinate and a float defining how strong it is.

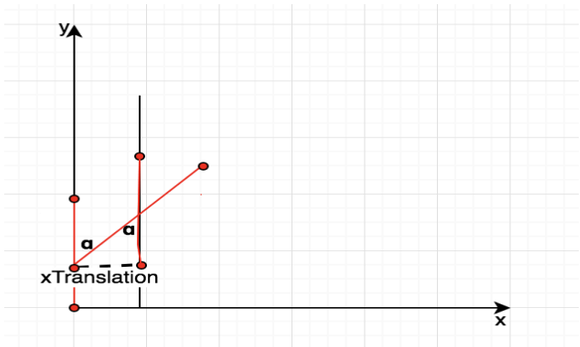
2.1 Branch

The branches are rendered as combination of cylinders. The key challenge here is to place the new branch at the correct location with respect to its parent branch. We pass a model matrix to shader and in this case, this model matrix controls the location of the branches.

The following three variables are required when constructing the model matrix for a new branch. First one is the angle to rotate, which determines the angle between new branch and its parent branch. Second is the translation on y-axis, which changes the distance from the start point of parent branch to the start point of our new branch along the axis of the original branch. The third one is the length scale, specifically, how long the new branch will be. We preserve both the model matrix with and without being scaled for each branch object. The child branch retrieves model matrix without being scaled from its parent and apply transformations, then the scaled model matrix is being passed to shaders.

Furthermore, during the translation phase before rotation, we also need to specify an x-translation so that the start point of new branch is on the axis of its parent after the rotation. As indicated in Figure 2, we set this x-translation to $sin(\alpha) \times L$, where L is length scale factor for the new branch. After the above steps, we add another rotation about y-axis for 3D effect. Figure 3 shows several examples of trees having branches rendered of above described method.

Figure 2: Illustration of Translation and Rotation for New Branch



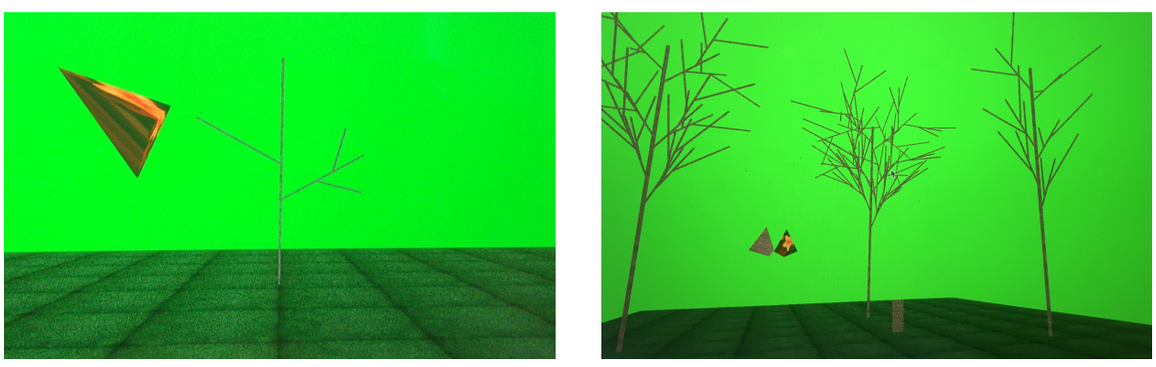


Figure 3: Tree Branches of Various Depth and Width generated by DFS

2.2 Leaves

The leaves are created using the same method as branches. Since in our current implementation, the leaves share the same transformations with branches, we added a boolean flag in the Branch class to indicate whether the created object is a leaf or a branch. During rendering, the boolean flag is checked and meshes are textured according to the flag. Currently, we generate leaves for branches at top 2 depth level.

We first placed the mesh creation function inside rendering loop, which severely hurt the performance of our program. Optimization was realized by storing a vector of meshes at the same time as building the tree structures, which happens before the actual rendering.

3 Skybox

We also implemented a skybox to represent the sky as a texture. We essentially created a large cube to encapsulate the enter scene, which is given a texture for each of it's six sides. We initially ran into a lot of trouble having the skybox not block other objects and to orient correctly. However, in the end, we were able to render the texture successfully such that the sky looks like a seemless and endless expanse of space.

An extension we wanted to try was environment mapping which would come in handy with water and other reflective surfaces, where the sky would reflect onto the water.

The skybox texture was created by 'The Mighty Pete' at http://www.petesoasis.com which is distributed it under the GNU General Public License version 2.

4 Grass & other features

To improve upon the landscape we added grass which is randomly generated as a texture on a transparent cube. We also implemented special shaders so that the pixels that are transparent are discarded instead of shaded. We also created a generation function to randomly generate grass each time the program starts.

Figure 4. shows our final scene with all the elements described in section 2, 3, 4 included.

Figure 4: Adding Leaves, Grass and Background

5 Other Experiments

We also wanted to implement other features such as water, which was something we spent a lot of time on. However, due to time constraints, we could not get a working version of a water simulation in time. We were only able to create a reflective and refractive surface but none of the water physics. Further, to host the body of water, we would need a terrain implementation, which we planned on implementing using heightmaps. This would likely be extremely time consuming and we just couldn't have everything done in time, although we did do various experiments to try to implement both features.

Additionally, we also experimented with particle systems and smoke/fire that was discussed in class. However, while we had working versions in 2D, the visuals were not optimal in 3D. Further, smoke and fire don't go well with the landscape scene. So we ended up discarding the idea. However, in the future, we want to try applying particle systems on simulating falling leaves.

6 Discussion

In previous sections we described our methods of rendering a scene consisting of trees and grass, also our attempts to include water and more complicated terrains. Additional effects closely related to our scene but

we didn't have time to try are tree growth animation and simulation of leave falling process. Overall we enjoyed working on this project although it only reflects a fraction of what we've learned through the entire semester.

In terms of the generation of trees, we noticed several limitations in our current strategy. Firstly regarding the appearance, we used simple cylinder of same radius at top and bottom to render all the branches, which impeded representation of the curves for individual trunks or branches. Similar problem applies to leaves too. We tried simple shapes like triangles, diamond for leaves and finally used combinations of triangles in different directions to increase the density. However, none of them look realistic enough. The potential improvement is replacing those simple 2D shapes with finer 3D models

Secondly about the ability of creating complex tree structures, our current implementation only allows fixed number of branches at each height and depth level. The degrees rotated from the parent branch and the length of each branch is hard-coded with limited variance depending on the specific height and depth. A structured format describing tree specifications is desired. After that, another question of interest is how to determine angles and length for each branch in a more principled way given this specification.