CSI 4130 – Assignment 4 – Group 2

Dylan Toh, 300411757, [dtoh062@uottawa.ca](mailto:dtoh062@uottawa.ca)

Samuel Pierre-Louis, 300211427, [spier033@uottawa.ca](mailto:spier033@uottawa.ca)

Submitted to Prof. Jochen Lang  
Due April 8th, 2023

As stated, the theme for this project is “Winter Wonderland”. We have made a low-poly snowy town with a forest, roads, houses, and a train with tracks.

The main goal at the start of the assignment was to get practice with creating mildly animated scenes in three.js. We wanted to load models with different textures and make them move. We wanted to load some skeletal animation. We wanted to use a skybox of some sort. We wanted to make a camera that could alternate between manual controls and a pre-defined path. We wanted to use splines in some way to get more familiar with them, and for the same reason we wanted create snow particles. We wanted the user to have some control over the lighting and other features.

There were a few ideas we considered but decided against implementing. Namely, we were curious about doing procedural bump mapping via a custom shader. We learned about the ShaderMaterial class provided by three.js, but we didn’t use it. Instead, we used randomized generation for the snowflake and outer trees, though it isn’t WebGL-oriented.

We’re happy to have achieved most of these goals. Because we used sets in the project, we had to learn about how multiple meshes can be stored as a scene in one file as well as how three.js learns these models. We discovered some interesting ways of loading the meshes and we made good use of instancing to limit the amount of data being sent to the GPU every frame. A starry night cubemap texture was used to model the skybox. We implemented a custom camera controller for WASDEQ + Mouse inputs (next page) and we’ve made the camera and train paths using Catmull-Rom splines. Additionally, we utilized an FBX Loader to import a character's skeletal animation, enabling them to dance, which adds to the scene with a playful and amusing atmosphere. Finally, Dylan implemented the snow using points and randomized velocity, giving a somewhat natural and chaotic feel to the snowfall.

As for the UI, we have reused the colors + azimuth/elevation approach for the lighting, added some options to control the speeds of the camera and train, and have added visualisation for the lighting and the Catmull-Rom paths.

# Setup

The application expects to be run in a web server in the same folder as “index.html”. For the list of packages to install, please check the “setup.md” file included in the zip archive.

# Controls

The control scheme is that of a regular first-person view. The scheme is as follows

|  |  |
| --- | --- |
| **INPUT** | **DESCRIPTION** |
| W / S | Forward / Backward |
| A / D | Left / Right |
| Q / E | Down / Up |
| Shift | Quadruple Movement Speed |
| Hold LMB/RMB + Drag | Pan/tilt camera in the motion of the mouse |

These controls are disabled when the camera is on the “On Tracks” mode.

## UI

### Lighting

The colour of the ambient and directional lights can be modified along with the direction of the directional light.

The origin of the directional light can be shown or hidden by toggling the “See light” option.

### Camera

The linear and rotation speed of the camera. Linear speed governs speed in both camera modes. The rotation speed is a measurement of how many pixels the mouse must be dragged to give a 180-degree rotation.

The camera mode can be toggled between “Manual” and “On Tracks” mode by pressing the “Toggle Camera Mode” button, and the path the camera follows can be visualised by toggling the “See path” option.

### Train

The speed of the train can be controlled just like the camera, and its path can be visualised in the same way.