CS606: Computer Graphics / Term 2 (2020-21) / Programming Assignment 4

Shashank Reddy Chirra
IMT2018069
IIIT Bangalore
shashank.reddy@iiitb.ac.in

Siddartha Veluvolu
IMT2018071
IIIT Bangalore
siddhartha.veluvolu@iiitb.ac.in

Akash Chunduri
IMT2018005
IIIT Bangalore
akash.chunduri@iiitb.ac.in

***Abstract*—In this assignment, we try to model a 3D scene with animation, camera setup, and textures. Some of the key learning points in this assignment are- Procedural animation of a scene, Modeling a dynamic scene with a Scene Graph and relative transforms, and enabling animation through the scene graph, Generating texture mappings for objects of different shapes, Managing lights and camera, Handling collisions and Basics of a VR application.**

I. Introduction

In this VR animation we generate a scene where a person is exploring a small city like environment which contains a few roads, a few buildings, street lights etc. The road has traffic in the form of busses. The person can also jump on these busses and move / jump on top of them. We also observe changes in the environment based on user input, for example, changes in the season.

The rendering of the scene can be broadly split into 4 main modules, namely Scene and Animation, Appearance and Textures, Lighting and Camera. We will discuss the functionality and core implementation in each of these modules in the following section.

II. Functionality and Implementation

A. Scene

1) Static Objects: Most of the static objects have been created by the intersection of basic geometries in Three.JS such as cube, cuboids and cylinders. Some examples of this are the Advertisment Board, some Buildings and the Street Lights.

The rest of the objects have been rendered using the OBJLoader() and MTLoader() classes, some examples of this are the trees and a few Buildings.

2) Dynamic Objects: Our scene only contains 2 dynamic objects, the user's avatar and the bus. The user's avatar has been generated using the intersection of Three.JS Cylinders and the has been loaded using OBJLoader() and MTLLoader().

B. Animation

1) Avatar: We used a scene graph to animate the avatar. First we created an empty container called body. This acts as the parent to multiple nodes, hand parent, leg parent, torso and head. Leg and Hand parent are parents of nodes called legs and hands respectively, this is because these parent nodes simulate joints in the human body and hence these leg and hand nodes rotate about their parents when the avatar is walking. Thus, complex movements such as walking and jumping results in the flow of computation through the scene graph to compute the positions of all the children with respect to the parent at each level.

The jumping motion is governed by Newtonian equations in order to simulate gravity, the body container is translated in the y direction and the scene graph updates are almost identical to the previous case of walking.

2) Buses: The buses are also translated using a scene graph. In this case, we have an empty container called Bus Controller that acts as the root of the scene graph. In this case we define the concept of a leader and a follower. Where one bus acts as a leader and all the other busses act as followers, that is, they mimic the actions of the leader. The Leader moves from end to end on the main road and takes a U turn at the end of the Main Road and the followers consequently follow the leader. Since the leader has a parent called Bus Controller in practice all the translations / rotations are applied on this container and not the leader. Hence to move the busses along the main road the BusController container is translated in the direction of the road. To take the Uturn, the BusController object is rotated about the y axis. The leader is actually translated slightly to the right in the Bus Controller's local co-ordinares, thus the leader rotates about the an axis located at position of the Bus Controller and the follower busses consequently follow the leader.

3) Bus and Avatar Interaction: The scene has designated bus stops where the avatar can hop onto the roof of the bus if the bus is also close to this bus stop. After the user lands on the rooftop, he can move along the length of the bus and also jump up and down, the user can then get down from the bus at the bus stop. This feature as well has been implemented using scene graphs, as we know the busses and the avatar are already implement using scene graphs, thus to attach the user to the roof of the bus, we add the root node of the avatar (body) as the child of the leader bus. After this all translations / rotations happen in the local co-ordinate system of the bus, thus we can freely walk and jump as we would before. To detach the avatar from the leader bus, we simply remove the body node as a child of the leader bus and add it back to the scene.

C. Lighting

The scene is illuminated by 3 light sources, an ambient light source (to mimic the sky), street lights and a spotlight in the sky. The street lights and the spot light have been implemented using the THREE.SpotLight() object and the ambient light has been implemented using the THREE.Hemisphere() object. The street lights and the spolight can be turned on and off bu the user.

D. Cameras

View have 3 cameras that can be shown in the viewing area. The first is the default view and is present in the sky, this camera can be moved using the mouse via trackball rotations. We can also zoom in and out using the mouse scroll. This camera was implemented using the OrbitalControls module in THREE.js. The second camera is attached to the avatar and gives the avatar's view of the scene. This camera has been attached to the head node of the avatar's scene graph, thus all translations and rotations applied to the avatar flow through the scene graph to the head node and consequently to the camera. The third camera attached to a drone in the sky, that can be controlled by the user. The user can elevate, lower, move forward, move backward and rotate the drone. The user can also increase / decrease the speed of the drone.

Textures

All the objects in the scene are rendered using textures. The meshes created using three.js geometry are mapped to their textures using three.js's THREE.TextureLoader(). For obj files, *MTLloader* is used to load the textures. A subset of objects textures are can be changed with user input by changing the image file path in the THREE.TextureLoader(). In case of OBJ files, the .mtl file used is changed and the object is re-rendered. A few objects textures are cylindrically mapped. This is achieved by manipulating the uv co-ordinates of those objects.

III. Acknowledgment

We would like to thank Prof Jaya and Prof Srikanth for coming up with such an interesting assignment and the TAs especially Amit Tomar for their tutorials and quick responses to doubts we had when attempting this assignment.

References

[1] https://threejsfundamentals.org/threejs/lessons/

[2] https://stackoverflow.com/questions/20774648/three-js-generate-uv-coordinate

[3] https://stackoverflow.com/questions/34958072/programmatically-generate-simple-uv-mapping-for-models

[4] https://threejs.org/docs/api/en/geometries/BoxGeometry

[5] https://threejs.org/docs/api/en/helpers/GridHelper