Computer Graphics Assignment 4

Jishnu V Kumar : IMT2018033
Khaveesh N : IMT2018036
Vikram Adithya : IMT2018085

18 May 2021

1 Introduction

The problem statement given was to implement a Virtual World with animated elements, lighting and texture. The user can join the world as an avatar and move around the world. The user can also pilot a drone to fly around the world.

2 Implementation Details

2.1 Scene Graph

The scene graph was divided into 2 parts. For the rendering, we used the scene graph functionality provided by Three.js. All the lights and cameras were added to this scene graph. We used our own implementation of hierarchy for doing the animation for moving objects. Each moving object maintains a list of attached objects. When an object's move function is called, it recursively calls it's children's move function. Each object implements only the movements with respect to it's parent in its own move function.

2.2 Texture Mapping

Texture mapping was done by modifying the UV mapping values of a mesh. For spherical mapping, a bounding sphere is calculated for the mesh. Using the radius, and centre of the sphere, UV coordinates can be found using 3D geometry and trigonometry. The UV mapping is done at a vertex level. The texture mapping in the interior of the faces are interpolated from the vertices.

Changing the texture image was done by maintaining two three.js texture objects per mesh. When we want to change the image, we can replace the texture objects connected to the three.js mesh.

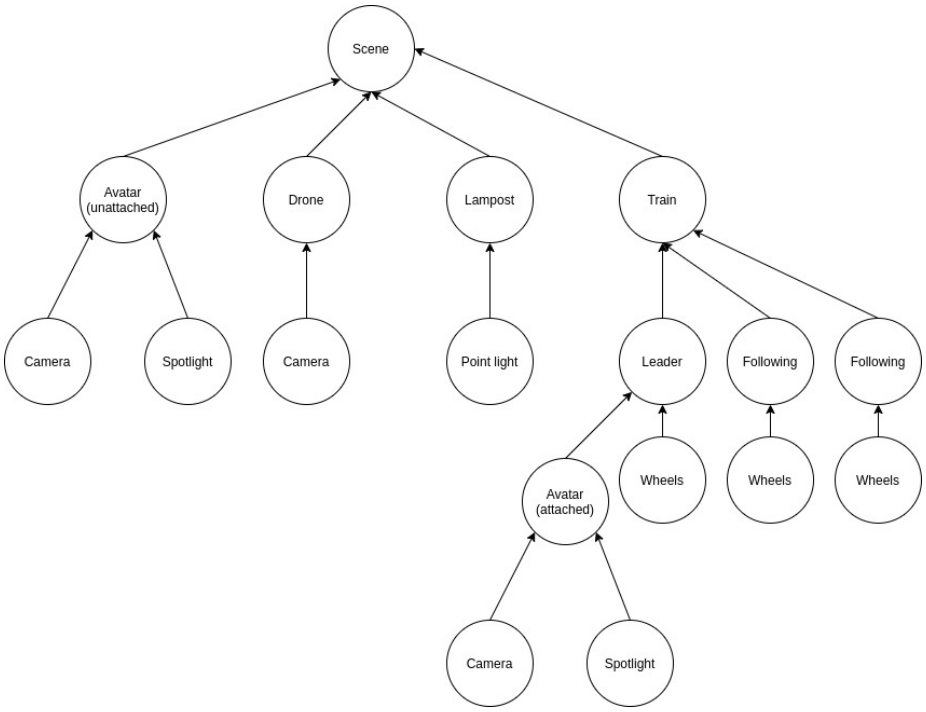

Figure 1: Scene Graph

2.3 Avatar and Drone

For the Avatar, we used the pointerLockControl functionality of three.js to allow the avatar to look around with the mouse. The WASD keys are used to move the Avatar around with respect to the horizontal direction it is facing.

The drone motion was implemented by modifying the three.js vectors for position and rotation associated with the mesh.

In both cases, the camera was attached to the mesh in the scene graph, so camera always follows the mesh.

2.4 Moving Objects (Leader and followers)

The lead object (Train engine) and it's followers (compartments) move along a fixed path which we created from a combination of curves added to CurvePath.

The axis rotation of an object is calculated by taking the cross product of a unit vector along z axis and the tangent at that point.

The angle of rotation is calculated by taking the cos inverse of the dot product of the above two vectors.

We then assign the new values of the quaternions of the object by using setFromAxisAngle(axis, angle). Thus allowing the object to rotate by the nec-

essary amount required while travelling the path.

For the movement of the lead object we use path.getPoint() to get the point the object must traverse to by passing a parameter, where 0 returns the initial point of the path and 1 the final point. By incrementing the parameter passed we can set the next position of the leader.

The position of the children/following objects are set to points behind the leader by passing a slightly lower value than the one held by leader, thus maintaining a set distance and speed throughout.

Though it might seem like each object is moving individually, we felt that this would be more suitable since when we utilise a dynamic path, we can make it such that the position of the lead object is used to build the path followed by the rest.

2.5 Collision Detection

Collision Detection is implemented using bounding boxes. We first compute the bounding box of the avatar. Then, we compute the bounding box of all the static objects in the scene and check if it intersects with the avatar's bounding box.

Since the dynamic objects in the scene, i.e the trains, are implemented as a grouping of individual components, we need to get an array of meshes of each component. Then, detection can be implemented as it is done for static objects.