Project- 3D Rendering, animation, camera setup and Texture

Pranav Kumar
IMT2018020
Pranav.Kumar@iiitb.org

Ellanti Bharath Sai
IMT2018022
Bharath.Sai@iiitb.ac.in

Udith Sai Manda
IMT2018081
Udith.Sai@iiitb.ac.in

***Abstract*—This document is a brief report of how 3D rendering done with a scene, Textures and camera setup using three.js library.**

I. Introduction

This Project is combination of all the learned features in this course using Three.js. Previously we have implemented every features using matrices and rendered objects using WebGL. In this project, we implemented all the features using the function in three.js. Three.js is a cross-browser JavaScript library and application programming interface (API) used to create and display animated 3D computer graphics in a web browser using WebGL. Three.js allows the creation of graphical processing unit (GPU)-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins. This is possible due to the advent of WebGL. High-level libraries such as Three.js or GLGE, SceneJS, PhiloGL, or a number of other libraries make it possible to author complex 3D computer animations that display in the browser without the effort required for a traditional standalone application or a plugin. We imported Some models from other websites as .GLTF or .obj files and rendered using three.js library functions. We even used various other properties of three.js like implementing camera, lighting effect, textures.. etc. We even applied the scene-graph property of three.js for rending of objects.

II. Concept behind Project

A. Loading imported Models

We imported models from other websites in .GLTF and .obj format. Then loaded those images using loading function available in three.js. After importing those each models, we will be taking them as an object and give attributes like position, rotation, size...etc. We used sketch fab website to download models.

B. Camera Setup

We used perspective camera available in three.js. We just need to create a camera object using the camera function in three.js and pass the attributes to the camera object. We put a default camera in space so that whole scene and all objects in the scene are visible. We applied orbit controls on default camera so that we can move around easily using mouse.

To create a avatar camera we place a camera at the head of avatar and we update the position of this camera as position of avatar. This makes the camera to to move along with avatar and get the vision of avatar. When avatar is attached to the group of moving objects then we update the position of camera as coordinates of group mesh so that camera follows the path set by group leader.

To create a drone camera we initially placed a camera at some height from origin. We then update the camera position in different directions based on user input. To move up or down we increase or decrease the Y value and similarly for other operations. We make the camera to look at down that is Y direction.

C. Lighting

After importing Lighting and setting up an camera, lighting is the next step which gives natural feel of object. In three.js, there are many type of lights already available in the library. We just need to import that as a function and give attributes like position and target. There are 6 types of lights available. Those are Ambient Light, Hemispherical light, Directional Light, Point light, Spot Light, Rectangular Light.

In this project, we have 4 types of point lights, 2 types of spot lights and one directional light. Point Lights are used as street lights which gives light to the whole view. Spot lights are used to focus on moving objects and as a torch light for moving objects. Directional Light is an extra light that we just used to make the screen more view-able.

D. Scene and Animation

We used some pre defined 3d objects to construct a scene. There is a human object as avatar which enters the scene on some key events. We can control the movement of Avatar by clicking arrow to keys to move in different directions. There are 3 animal objects one cat and two dogs which move one group in a predefined circular path. This group has cat as leader and we compute path on this cat object. Other objects in group follow their before object. We create scene graph to implement all these transformations. We first create a scene and add a group mesh object to the scene. This group mesh can be assumed as a single mesh that contains all the three objects. Whatever transformation we applied to group mesh applies on all the objects in the group. This group mesh has two children one is cat mesh and other is group2 mesh. This group2 mesh contains all the remaining objects except cat mesh. Similarly group2 has two children group3 mesh and dog1 mesh. This

group3 mesh contains all the objects except cat mesh and dog1 mesh. As we have only three objects group3 mesh contains only dog3 mesh. This way we can define our circular path on group mesh and it is followed by group2 mesh and cat mesh. Group2 mesh is followed by group3 mesh and dog1 mesh. Similarly group3 mesh is followed by dog2 mesh. This we can compute paths for group of objects easily. To compute a circular path we increase angle from 0 to 360 with some delta and multiply this angle with radius we get the arc length. So we change the coordinates as arc length and thus we got circular path. We also have a static windmill and three cubes arranged vertically each on top of other.

We create a human object and not add this directly to scene. We add this to scene only when there is key event representing to render avatar. Along with this we add lights to the scene object and default camera to scene object. We then call render function on scene object. This makes to render all the children of scene object to render.

For attaching the human to the moving group of animals, we remove human object from scene object and add to group3 mesh which we created initially. This makes the human to follow group path. To make the human turn around while attached we rotate the human around y-axis by some adding/ subtracting delta from initial value based on left or right rotation. To make the user jump up while attached, we increase the y value using some delta to certain value and decrease to original value. For jump down we first decrease Y value and then increase it.

E. Appearance and Texture

Texture mapping is in simple terms is take an image and stretch it over a 3D mesh object. The image used in this case is called as the texture used. Textures can be used to represent material properties like color, roughness, and opacity. It's quite simple to map the texture onto a regular shapes such as cubes or such. But with complex models that have irregular geometry, efficient mapping techniques have to be implemented. In this project, we have used **UV Mapping**, which allows to create a connection between points on the geometry and points on the face. The implementation must be carefully done since the UV map must be coordinated to match the texture to the correct points on the face.

There are two cases in texture mapping where the UV mapping implementation helps.

1) Spherical Mapping: Imagine a model is translucent and a sphere inside made of film and inside the sphere is a point light so that it projects (like a projector) from the sphere in all directions.

To get a point on the sphere, we multiply the points by the inverse of the world matrix for the sphere and then normalize the result. After that there's still the problem of how the texture itself is mapped to the imaginary sphere.

The work around to this problem is to implement the UV coordinates to the geometry of the model. Assuming x,y,z are the normalized coordinates.

$$U = Math.atan2(z, x)/Math.PI * 0.5 - 0.5;$$
$$V = 0.5 - Math.asin(y)/Math.PI;$$

2) Cylindrical Mapping: Like spherical mapping but except here we assume there's tiny cylinder projecting on to the model.

Unlike a sphere, cylinder has orientation but the points of the model can be translated into the orientation of the cylinder then assuming x,y,z are now relative to the cylinder. They are multiplied by the inverse matrix of the matrix that represents the orientation of the cylinder.

We implement the UV coordinates to the geometry of the model as follows. Assuming x,y,z are the normalized coordinates.

$$U = Math.atan2(x, z)/Math.PI * 0.5 + 0.5$$
$$V = y$$

III. Implementation

- Initially we imported some models from sketch fab. We imported grass, dogs, cat, human, street light, wind mill, sun.
- Then we gave attributes like position, size and alignment to prepare a scene.
- We initialized three cameras as mentioned above. Those are default total view camera, human camera and other is drone camera.
- We also included Orbit Controls for default camera to easily look around the model using mouse.
- Then we kept lighting. As mentioned above we used three types of lights. One is point light which is used as street lights. These positions are fixed.
- Then Second light is spot light. One spot light will at a fixed point but the target moves along the moving objects. This light is red colour. The other spot light is moving with moving objects and targets the front view of moving objects. This light is yellow colour. We even used an extra directional light just to make the scene view able.
- Now we have created a scene graph of 2 dogs and cat. They will be moving in lined manner in circular direction.
- When we click "c", we will be able to see man in front of us. Using Arrow keys we can move the man.
- Clicking 'a', the man gets back of the dogs and cat and follow them in same motion.

- Clicking 'u' 'o' while the man is on moving objects, then the man jumps up or down respectively in motion.
- Clicking 'd', the man gets back to initial position out of moving objects.
- When we are in light mode, this happens when we click 'l'. Default it is in light mode. We can ON/OFF different lights,
  - 1- ON/OFF 1st street light
  - 2- ON/OFF 2nd street light
  - 3- ON/OFF 3rd street light
  - 4- ON/OFF 4th street light
  - 5- ON/OFF Focus light on dogs and cat
  - 6- ON/OFF Torch light which is present on animals and points in front of them.
- When we click 'p', We go to camera mode where we can view through different cameras which are implemented.
  - 1 - default view
  - 2 - human view
  - 3 - drone view(can move using arrows and o,u)
- We implement texture mapping on all objects in the scene. Windmill, car and cube are implemented such a way that they can change their texture images using user clicks.
  - 1 - tapping **m** will change the texture image of car and windmill
  - 2 - clicking **k** will change the texture image of all the cubes in the tower and there are three textures associated with it.
  - 3- clicking **h** will change the mapping of the texture of the topmost cube in the tower to Spherical mapping
  - 4- clicking **j** will change the mapping of the texture of the 2nd/the middle cube in the tower to Cylindrical mapping
  - 5- clicking **b** will reset the texture mapping to planar mapping for all the cubes in the tower

IV. Conclusion

Thus we created a scene with different objects and render them with different camera setups, textures and lighting. We first used some basic geometry like cube and cylinder and build upon them by importing real objects in place of them. We performed different actions in the scene using different user inputs. The library we used three.js has provided lot of abstraction and hence avoid us to implement basic functionalities like updating projection matrix, computing vertex shader and fragment shader. We got some functions for above operations to implement.

V. Acknowledgement

We would like to thank Professor Jaya Sreevalsan Nair and Professor TK Srikant for giving us a great head start in this world of Computer Graphics. Next we would also like to thank our amazing TA's especially Amit Tomar who have been teaching us all the things that we have needed for the course and also for the assignments.

References

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