Sarun Jeyamenan
Matricola: 2036344

Project report

Description of the environment

For this project I've used the three.js library. This is a library used to create and display animated 3D computer graphics in a web browser using WebGL. It has good documentation online and simplifies the usage of WebGl. First I did was to create a scene with a cube then added light and shadows. Later I changed the cube to a sphere and played with different textures. I opted for a neptune texture and that is how I got the theme of space and explorations which this project is showing in a minimalistic way.

Skybox

To give a three dimensional background look, I created a skybox. I downloaded the images from this website https://opengameart.org/content/skiingpenguins-skybox-pack which gives 6 images for different views of a terrain. The six textures are respectively added as the top, front, back, right, left and bottom of a box in three js.

```
scene.background= New
THREE.CubeTextureLoader().setPath("skybox_img/").load(["mystic_l
f.jpg","mystic_rt.jpg","mystic_up.jpg","mystic_dn.jpg","mystic_f
t.jpg","mystic_bk.jpg"]);
```

The cube texture is added as the scene background. To explore the rendered world I added a thee.js extension for the camera, Orbitcontrols. This lets the user control the camera position with the mouse by holding down the mouse and moving the cursor around. In addition let the user zoom in and out with the mousewheel.

Sarun Jeyamenan
Matricola: 2036344

GLTF loader

For the model I explored at the website Mixamo, which is a library with premade 3D-models. I picked a model called “Vanguard By T. Choonyung” and downloaded the model in glb format. To load the model to the scene I used a three js GLTF loader to load the model and added it to the scene.

```
const loader = new GLTFLoader().load( 'models/Soldier.glb',
function ( gltf ) {
        loaded_model = gltf.scene;
        loaded_model.castShadow = true;
        loaded_model.traverse(function (node) {
            if (node.isMesh) {
              node.castShadow = true;
              node.receiveShadow = true;
            }
        });
```

Also to make the model cast shadows, I needed to set the attribute castShadow = true for the model.scene. However it did not work, and after some searching on google I found that only the mesh object of the 3D model can cast shadow, by traversing through the child nodes and checking if it is a mesh object, then set the castShadow = true and receiveShadow = true. Somehow it still didn't work so I did not focus on it.

Animation

The model is a complex model with multiple nodes and to exploit the advantages of the scenegraph to make an animation, I needed to be able to control each element of the model. Since the model was a glb file it was not easy to understand how the model was built up. I found a function from https://r105.threejsfundamentals.org/threejs/lessons/threejs-load-gltf.html that could print the scene graph of the model and see the object name of each node in the model.

Sarun Jeyamenan
Matricola: 2036344

```
Scene [Group]                                                    app.js:90
  └Character [Object3D]
    ├mixamorigHips [Bone]
    │ ├mixamorigSpine [Bone]
    │ │ └mixamorigSpine1 [Bone]
    │ │   └mixamorigSpine2 [Bone]
    │ │     ├mixamorigNeck [Bone]
    │ │     │ └mixamorigHead [Bone]
    │ │     │   └mixamorigHeadTop_End [Object3D]
    │ │     ├mixamorigLeftShoulder [Bone]
    │ │     │ └mixamorigLeftArm [Bone]
    │ │     │   └mixamorigLeftForeArm [Bone]
    │ │     │     └mixamorigLeftHand [Bone]
    │ │     │       ├mixamorigLeftHandThumb1 [Bone]
    │ │     │       │ └mixamorigLeftHandThumb2 [Bone]
    │ │     │       │   └mixamorigLeftHandThumb3 [Object3D]
    │ │     │       │     └mixamorigLeftHandThumb4 [Object3D]
    │ │     │       ├mixamorigLeftHandIndex1 [Bone]
    │ │     │       │ └mixamorigLeftHandIndex2 [Bone]
    │ │     │       │   └mixamorigLeftHandIndex3 [Bone]
    │ │     │       │     └mixamorigLeftHandIndex4 [Object3D]
    │ │     │       ├mixamorigLeftHandMiddle1 [Bone]
    │ │     │       │ └mixamorigLeftHandMiddle2 [Bone]
```

After printing the scenegraph, I had to access each object to do the animation on. By using the object name from the scene graph I could save each object globally. The model was downloaded in a T-pose, but for the animation it was not a good starting point for the arm so I rotated the arm so the modal had a neutral position with the arm downward on the sides.

```
leftarm = loaded_model.getObjectByName('mixamorigLeftArm');
lefForeArm = loaded_model.getObjectByName('mixamorigLeftForeArm');
rightarm = loaded_model.getObjectByName('mixamorigRightArm');
```

After being able to get control over each node, I tested the different rotations on each node to see the local xyz - orientation of each node and find the animation interval for each node. 3 theta list was made where 2 of the list was the start and end values for the theta. The third was the theta that was used by the node and changed incrementally during the rendering. I made a function to control the speed and the change of direction when the interval boundaries were met. With further development I would have added more animations for the hand and foot for a less robotic walking animation and make it more human like animation.

Sarun Jeyamenan
Matricola: 2036344

Character Control

To make the model walk from user interactions I added event listeners for Keydown and Keyup. A function was made for each event where the keyboard keys W,A,S,D were catched if they were pressed.

```
function onKeyDown(event) {
    switch (event.keyCode) {
      case 87: // w
        keys.forward = true;
        break;
      case 65: // a
        keys.left = true;
        break;
      case 83: // s
        keys.backward = true;
        break;
      case 68: // d
        keys.right = true;
        break;
    }
}
```

Then in the animate function I have to make the model translate accordingly to the keys pressed. To move the model forward, a unit vector is multiplied with the model Quaternion which is the local rotation matrix, then normalized. Then the x and z position is increased and updates the model positions. For the rotations a unit vector is set and a quaternion matrix is made from a given angle from the unit vector. This matrix is then multiplied with the model matrix and updated.

To make the walking animation only run when walking controls are pressed I added a boolean check, where if the keys are pressed the animation will run increment else the animation stops. I noticed that the animation was going slower so I turned off the anti-aliasing which resulted in a less smoother surface edge on the sphere.

Sarun Jeyamenan
Matricola: 2036344

This project was done by only me and not Sven Thorkildsen. The github was made before he chose to not take the course.

Resources

THREE.js: https://cdn.jsdelivr.net/npm/three@0.118/build/three.module.js

Orbitcontrol:
https://cdn.jsdelivr.net/npm/three@0.118/examples/jsm/controls/OrbitControls.js

GLTFLoader:
https://cdn.jsdelivr.net/npm/three@0.118.1/examples/jsm/loaders/GLTFLoader.js

Character model:
https://www.mixamo.com/#/?page=1&type=Character

Animation of GLFT model resource:
https://r105.threejsfundamentals.org/threejs/lessons/threejs-load-gltf.html

Three.js docs:
https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene

Skybox link:
https://opengameart.org/content/skiingpenguins-skybox-pack

Neptune image:
https://www.solarsystemscope.com/textures/download/2k_neptune.jpg