Computer Graphics Assignment

Link to the GitHub Repository: -

https://github.com/ShubhamGupta986725/ComputerGraphicsAssignment

1. Creation of the Frog Model: -

All the models were made in Blender, as well as the animations. This was one of the most time intensive tasks, as to do this part of the project correctly was essential to make sure the later parts don't mess up.

2. Setting up the project: -

To setup the project, we have used Vite. Vite is a local development server, and to run the project, download the files, and in the directory, open the terminal, and type the following command: -

- npm run dev

Do make sure that when you run this command, you have NodeJS installed. If you don't, you can install it from NodeJS's home page.

We first initialised the project, and that gave us the overall outline for our project, including the HTML, CSS, JS and JSON files.
In the JS file, we will also add orbital controls, which allow us to move the web page using the mouse, and also zoom in and out with the scroll wheel.

```
import * as THREE from 'three';
import { OrbitControls } from
'three/addons/controls/OrbitControls.js';
```

We also add a camera, a scene and a renderer, an ambient light and a point light which is done using the following: -

```
const scene = new THREE.Scene();

const camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000 );

const renderer = new THREE.WebGLRenderer({
    canvas: document.querySelector('#bg'),
});

renderer.setPixelRatio( window.devicePixelRatio );
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.setClearColor(0xffffff, 1);
camera.position.setZ(50);
camera.position.setY(50);
const pointLight = new THREE.PointLight(0xffffff);
pointLight.position.set(30,30,30);

const ambientLight = new THREE.AmbientLight(0xffffff);
ambientLight.position.set(30,30,30);
scene.add(pointLight, ambientLight);
```

Then, we make an animate function, which is continuously called to update and the webpage on the basis of the orbit controls.

```
const controls = new OrbitControls(camera, renderer.domElement);

function animate(){
    requestAnimationFrame(animate);
    renderer.render(scene, camera);
    controls.update();
}
animate();
```

3. Loading the Model: -

The model of the frog, once exported to the GLB format, the model can easily be loaded onto the web page using the GLTF Loader object. To do this, we first have to import GLTF Loader in the following way

```
import { GLTFLoader } from 'three/addons/loaders/GLTFLoader.js';
const frog = new GLTFLoader();
frog.load( 'FrogFinal.glb', function ( gltf ) {
    var model = gltf.scene;
    scaleModel(model);

    var origModel = model.clone();
    scene.add( model );
}, undefined, function ( e ) {
    console.error( e );
});
```

After loading, the webpage will look like this: -

4. The Shaders: -

For this project, we have made a simple shader to accompany the point and ambient lights. ThreeJS automatically adds a default shader to a model when point and ambient lights are added. The extra shaders are added using the following code: -

```
function vertexShader() {
    return `
        varying vec3 vUv;

        void main() {
            vUv = position;

            vec4 modelViewPosition = modelViewMatrix *
vec4(position, 1.0);
            gl_Position = projectionMatrix * modelViewPosition;
        }
        `
  }
```

```
function fragmentShader() {
    return `
        uniform vec3 colorA;
        uniform vec3 colorB;
        varying vec3 vUv;

        void main() {
            gl_FragColor = vec4(mix(colorA, colorB, vUv.z), 1.0);
        }
        `
}

let material =  new THREE.ShaderMaterial({
    fragmentShader: fragmentShader(),
    vertexShader: vertexShader(),
  })
```

This material is then added to the model. The issue is that the model uses a skin on top of the shader material, so even after invoking the shader material, the model's appearance doesn't change.

5. Coding the motion of frog (Arrow keys and Arrow Keys + Shift): -

To code the translation and rotation functions for the frog, we can just use increment the x and z position or rotation of the frog for the time duration the key is pressed. This is done in the following functions: -

```
function moveForward(model,origModel){
    model.position.x += 0.5;
    origModel.position.x += 0.5;
    origX += 0.5;
```

```
}
function moveBackward(model, origModel){
    model.position.x -= 0.5;
    origModel.position.x -= 0.5;
    origX -= 0.5;
}
function moveLeft(model, origModel){
    model.position.z -= 0.5;
    origModel.position.z -= 0.5;
    origZ -= 0.5;
}
function moveRight(model, origModel){
    model.position.z += 0.5;
    origModel.position.z += 0.5;
    origZ += 0.5;

}
function rotateForward(model, origModel){
    model.rotation.z -= 0.2;
    origModel.rotation.z -= 0.2;
    origZRot -= 0.2;
}
function rotateBackward(model, origModel){
    model.rotation.z += 0.2;
    origModel.rotation.z += 0.2;
    origZRot += 0.2;
}
function rotateLeft(model, origModel){
    model.rotation.x -= 0.2;
    origModel.rotation.x -= 0.2;
    origXRot -= 0.2;
}
function rotateRight(model, origModel){
    model.rotation.x += 0.2;
    origModel.rotation.x += 0.2;
    origXRot += 0.2;
}
```

Here, origModel and origValues are used to find the position of the frog, and then will be used in other animations, like to extend the front and back legs, along with turning the head and jumping, as well as swimming.

To detect for key presses, we add an event listener, which listens for ‘keydown’. This is implemented as follows: -

```
document.addEventListener('keydown', event => {
    if(event.shiftKey && event.key === 'ArrowLeft'){
        rotateLeft(model, origModel);
    }
    if(event.shiftKey && event.key === 'ArrowUp'){
        rotateForward(model, origModel);
    }
    if(event.shiftKey && event.key === 'ArrowRight'){
        rotateRight(model, origModel);
    }
    if(event.shiftKey && event.key === 'ArrowDown'){
        rotateBackward(model, origModel);
    }
    if(event.key === 'ArrowLeft' && !event.shiftKey){
        moveLeft(model, origModel);
    }
    if(event.key === 'ArrowUp' && !event.shiftKey){
        moveForward(model, origModel);
    }
    if(event.key === 'ArrowRight' && !event.shiftKey){
        moveRight(model, origModel);
    }
    if(event.key === 'ArrowDown' && !event.shiftKey){
        moveBackward(model, origModel);
    }
});
```

6. Coding the frog to look left and right (‘a’ and ‘d’): -

To code the frog to look left and right, we can just rotate the model’s head left and right, based on which key is pressed. This is done using the following code

```
function lookLeft(model){
    model.children[1].rotation.x = -2 +
origXRot/(scaleFactor*scaleFactor);
    model.children[1].rotation.z = -1 +
origZRot/(scaleFactor*scaleFactor);
}

function lookRight(model){
    model.children[1].rotation.z = -2 +
origZRot/(scaleFactor*scaleFactor);
    model.children[1].rotation.x = -1 +
origXRot/(scaleFactor*scaleFactor);
}
```

To detect the key presses ‘a’ and ‘d’, we make another document event listener, which listens to keypress, and this is done using the following code

```
document.addEventListener('keypress', event => {
        if (event.key === 'a') {
            // Make frog turn left
            lookLeft(model);
        }
        if (event.key === 'd') {
            // Make frog turn right
            lookRight(model);
        }
});
```

It is also important that once the keys ‘a’ and ‘d’ are lifted, the model returns to the original state. This is done using the final document event listener, which detects keyups.

```
document.addEventListener('keyup', event => {
if (event.key === 'a') {
            scene.remove(model);
            model = origModel.clone();
            scene.add(model);
        }
if (event.key === 'd') {
            scene.remove(model);
            model = origModel.clone();
            scene.add(model);
        }
    });
```

The following are the images of the frog looking left and right: -

7. Extending the front and rear legs (‘w’ and ‘s’): -

This and the 8th point were hardest to implement because of the complexity of the implementation. To extend the model’s front and read legs, we have transformed each component of the legs. We also have to keep in mind the frogs orientation, both rotation and location wise. This is done using the following Code

```
function extendFrontLegs(model){
```

```
    model.children[0].children[0].rotation.x = 0.3 +
origXRot/(scaleFactor*scaleFactor);
    model.children[0].children[0].rotation.y = 0.3;
    model.children[0].children[0].rotation.z = -0.3 +
origZRot/(scaleFactor*scaleFactor);
    model.children[0].children[0].position.x = -0.0017 +
origX/(scaleFactor*scaleFactor);
    model.children[0].children[0].position.y = 0.0018;
    model.children[0].children[0].position.z = 0.003 +
origX/(scaleFactor*scaleFactor);

    model.children[0].children[1].rotation.x = 0.3 +
origXRot/(scaleFactor*scaleFactor);
    model.children[0].children[1].rotation.y = -0.3;
    model.children[0].children[1].rotation.z = -0.3 +
origZRot/(scaleFactor*scaleFactor);
    model.children[0].children[1].position.x = -0.0017 +
origX/(scaleFactor*scaleFactor);
    model.children[0].children[1].position.y = 0.0018;
    model.children[0].children[1].position.z = -0.003 +
origX/(scaleFactor*scaleFactor);

    model.children[10].position.x = 0.0045 +
origX/(scaleFactor*scaleFactor);
    model.children[10].position.y = -0.0015;
    model.children[10].position.z = -0.0045 +
origZ/(scaleFactor*scaleFactor);
    model.children[10].rotation.x = 0 + origX/(scaleFactor*scaleFactor);
    model.children[10].rotation.y = 2.5;
    model.children[10].rotation.z = 1.5 +
origZRot/(scaleFactor*scaleFactor);

    model.children[11].position.x = 0.004 +
origX/(scaleFactor*scaleFactor);
    model.children[11].position.y = -0.0012;
    model.children[11].position.z = 0.005 +
origZ/(scaleFactor*scaleFactor);
    model.children[11].rotation.x = 0 + origX/(scaleFactor*scaleFactor);
    model.children[11].rotation.y = 3.5;
```

```
    model.children[11].rotation.z = 4.5 +
origZRot/(scaleFactor*scaleFactor);
}

function extendRearLegs(model){
    model.children[9].children[0].rotation.x = -0.3 +
origXRot/(scaleFactor*scaleFactor);
    model.children[9].children[0].rotation.y = 0.5;
    model.children[9].children[0].rotation.z = 0.3 +
origZRot/(scaleFactor*scaleFactor);
    model.children[9].children[0].position.x = 0.004 +
origX/(scaleFactor*scaleFactor);
    model.children[9].children[0].position.y = 0.004;
    model.children[9].children[0].position.z = -0.009 +
origZ/(scaleFactor*scaleFactor);

    model.children[9].children[0].children[0].rotation.x = 3 +
origXRot/(scaleFactor*scaleFactor);
    model.children[9].children[0].children[0].rotation.y = 0.5;
    model.children[9].children[0].children[0].rotation.z = 5.5 +
origZRot/(scaleFactor*scaleFactor);
    model.children[9].children[0].children[0].position.x = 0.007 +
origX/(scaleFactor*scaleFactor);
    model.children[9].children[0].children[0].position.y = -0.0004;
    model.children[9].children[0].children[0].position.z = -0.004 +
origZ/(scaleFactor*scaleFactor);



    model.children[9].children[1].rotation.x = 0.3 +
origXRot/(scaleFactor*scaleFactor);
    model.children[9].children[1].rotation.y = -0.5;
    model.children[9].children[1].rotation.z = -0.3 +
origZRot/(scaleFactor*scaleFactor);
    model.children[9].children[1].position.x = -0.005 +
origX/(scaleFactor*scaleFactor);
    model.children[9].children[1].position.y = 0.005;
    model.children[9].children[1].position.z = -0.008 +
origZ/(scaleFactor*scaleFactor);
```

```
    model.children[9].children[1].children[0].rotation.x = 2.5 +
origXRot/(scaleFactor*scaleFactor);
    model.children[9].children[1].children[0].rotation.y = 3;
    model.children[9].children[1].children[0].rotation.z = 2.5 +
origZRot/(scaleFactor*scaleFactor);
    model.children[9].children[1].children[0].position.x = 0.005 +
origX/(scaleFactor*scaleFactor);
    model.children[9].children[1].children[0].position.y = 0.0025;
    model.children[9].children[1].children[0].position.z = 0.003 +
origZ/(scaleFactor*scaleFactor);
}
```

Again, to detect key presses and to reset the frog back to the original, we use a similar code as that in part 6. For continuity, it is given below

```
document.addEventListener('keypress', event => {
        if (event.key === 'w') {
            // Extend front legs
            extendFrontLegs(model);
        }
        if (event.key === 's') {
            // Extend rear legs
            extendRearLegs(model);
        }
    });
```

```
document.addEventListener('keyup', event => {
        if (event.key === 'w') {
            scene.remove(model);
            model = origModel.clone();
            scene.add(model);
        }
        if (event.key === 's') {
            scene.remove(model);
            model = origModel.clone();
            scene.add(model);
        }
```

```
    });
```

8. Swimming and Jumping Animations (‘j’ and ‘q’): -

This part was easily the hardest to do, as we needed to make custom animations for the frog to jump and swim. After making the custom animations, we used an AnimationMixer to play the animations. It was also a part where a lot of debugging had to be done, as dealing with multiple models was not the easiest task to do. Finally, this is the code for the jump and swim animations: -

```
function playAnimations(gltf, modelJ){
    mixerJump = new THREE.AnimationMixer( modelJ );
    const clips = gltf.animations;
    clips.forEach(function(clip){
        const action = mixerJump.clipAction( clip );
        action.setLoop(THREE.LoopOnce);
        action.clampWhenFinished = true;
        action.enable = true;
        action.play();
    });
    aniamtionDone = true;
}

function jump(model, origModel){
    scene.remove(model);
    const jumper = new GLTFLoader();
    jumper.load( 'Jump.glb', function ( gltf ) {
        var modelJ = gltf.scene;
        scaleModel(modelJ);
        scene.add( modelJ );
        playAnimations(gltf, modelJ);
        setTimeout(function(){
            scene.remove(modelJ);
```

```
        }
        , 2000);
    });
};

function swim(model, origModel){
    scene.remove(model);
    const jumper = new GLTFLoader();
    jumper.load( 'Swim.glb', function ( gltf ) {
        var modelJ = gltf.scene;
        scaleModel(modelJ);
        scene.add( modelJ );
        playAnimations(gltf, modelJ);
        setTimeout(function(){
            scene.remove(modelJ);
        }
        , 2000);
    });
};
```

Detecting the key presses for ‘q’ and ‘j’ were pretty much the same as before: -

```
document.addEventListener('keypress', event => {
        if (event.key === 'q') {
            // Swim
            swim(model,origModel);
        }
        if (event.key === 'j') {
            // Jump
            jump(model,origModel);
        }
    });
```

However, resetting the model was a bit different, as we needed a timer to make sure the model wasn’t restored as soon as the key was lifted, but rather when the animation was over.

```
document.addEventListener('keyup', event => {
        if (event.key === 'j') {
```

```
            let temp = true;
            while(temp){
                setTimeout(function(){
                    scene.remove(model);
                    model = origModel.clone();
                    scene.add(model);},
                2000);
                temp = false;
            }
        }
    if (event.key === 'q') {
            let temp = true;
            while(temp){
                setTimeout(function(){
                    scene.remove(model);
                    model = origModel.clone();
                    scene.add(model);},
                2000);
                temp = false;
            }
        }
    });
```

9. Supporting Resources: -

For the following project, we have tried to document our thought process and steps we have incorporated in as much detail as possible. Along those lines, we have also made a video folder, which has different operations in a video format. The link to the same is as follows: -

https://drive.google.com/drive/folders/1CWJCiPEeUbxM3Xi2bAUZwYOxAv7Nj5Kc?usp=sharing