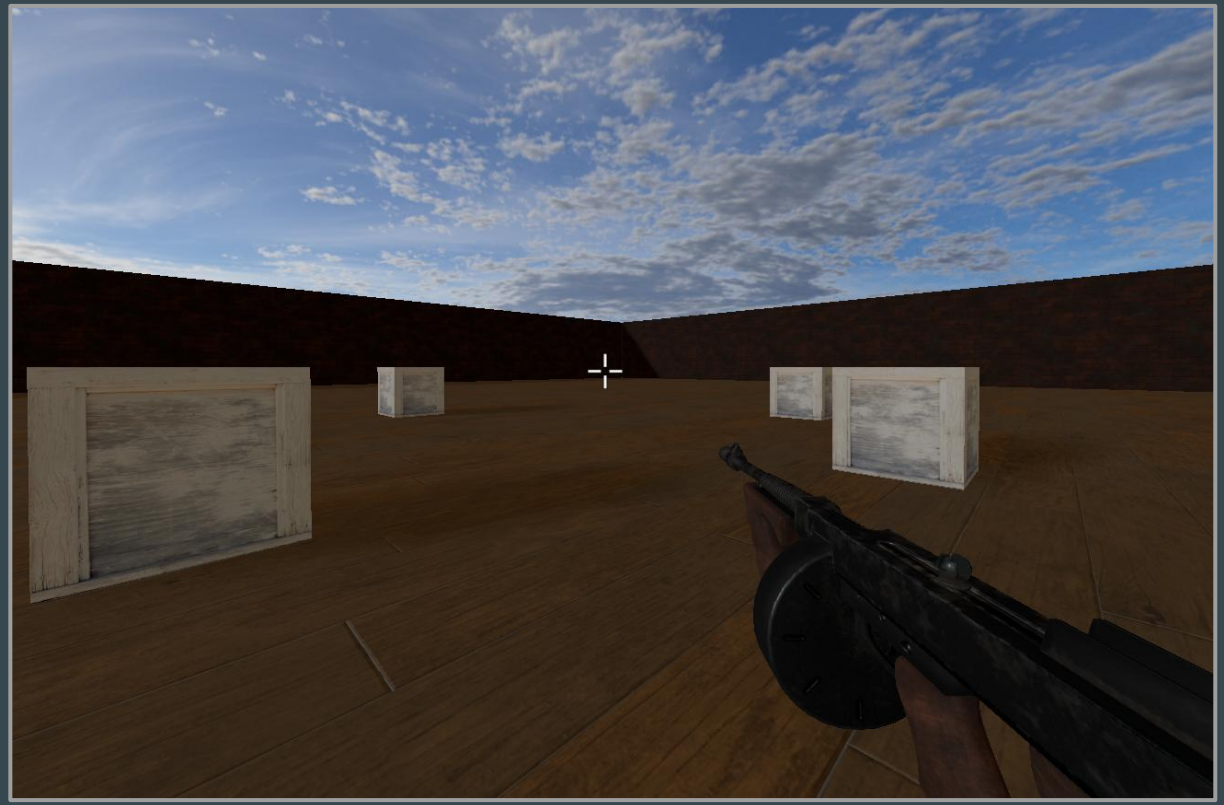
FPS Game Demo

Kendall Fischer

Goals for the project

- Create a first person camera view similar to that of a first person shooter
- Have a gun model follow the camera
- Build a basic scene with some objects in it



First Person Camera

- Controls camera movement
- Simulates step motion using sin()
- Does intersection tests with objects in the scene and does a lookAt()

```
class FirstPersonCamera {
    constructor(camera, objects) {
        this.camera = camera;
        this.input = new InputController();
        this.rotation = new THREE.Quaternion();
        this.translation = new THREE.Vector3(0, 2, 0);
        this.phi = 0;
        this.phiSpeed = 8;
        this.theta = 0;
        this.thetaSpeed = 5;
        this.headBobActive = false;
        this.headBobTimer = 0;
        this.objects = objects;
    }

    update(timeElapsedS) {
        this.updateRotation(timeElapsedS);
        this.updateCamera(timeElapsedS);
        this.updateTranslation(timeElapsedS);
        this.updateHeadBob(timeElapsedS);
        this.input.update(timeElapsedS);
    }
```

The Scene

- Initialization broken up into separate functions
- Enabled physically based rendering
- Attached gun to the camera

```
function init() {
    initializeRenderer();
    initializeLights();
    initializeScene();
    initializeFPSCamera();
    clock = new THREE.Clock();
    raf();
    onWindo
}
```

```
function initializeRenderer() {
    threejs = new THREE.WebGLRenderer();
    threejs.shadowMap.enabled = true;
    threejs.shadowMap.type = THREE.PCFSoftShadowMap;
    threejs.setPixelRatio(window.devicePixelRatio);
    threejs.setSize(window.innerWidth, window.innerHeight);
    threejs.physicallyCorrectLights = true;
```

```
const gltfLoader = new GLTFLoader();
gltfLoader.load( url: './Models/thompson_submachine_gun/scene.gltf', onLoad: function( gltf ) {
    gun = gltf.scene;
    gun.position.set(.5, -.5, -.8);
    gun.rotation.y = Math.PI /2;
    camera.add(gun);
}, onProgress: undefined, onError: function ( error ) {
    console.error( error );
} );
```

Textures

- From freepbr.com
- loadMaterial function loads each part of the texture into a material

```
function loadMaterial(name, tiling) {
    const mapLoader = new THREE.TextureLoader();
    const maxAnisotropy = threejs.capabilities.getMaxAnisotropy();

    const metalMap = mapLoader.load( url: 'Textures/freepbr/' + name + 'metallic.png');
    metalMap.anisotropy = maxAnisotropy;
    metalMap.wrapS = THREE.RepeatWrapping;
    metalMap.wrapT = THREE.RepeatWrapping;
    metalMap.repeat.set(tiling, tiling);

    const albedo = mapLoader.load( url: 'Textures/freepbr/' + name + 'albedo.png');
    albedo.anisotropy = maxAnisotropy;
    albedo.wrapS = THREE.RepeatWrapping;
    albedo.wrapT = THREE.RepeatWrapping;
    albedo.repeat.set(tiling, tiling);
    albedo.encoding = THREE.sRGBEncoding;

    const normalMap = mapLoader.load( url: 'Textures/freepbr/' + name + 'normal.png');
    normalMap.anisotropy = maxAnisotropy;
    normalMap.wrapS = THREE.RepeatWrapping;
    normalMap.wrapT = THREE.RepeatWrapping;
    normalMap.repeat.set(tiling, tiling);

    const roughnessMap = mapLoader.load( url: 'Textures/freepbr/' + name + 'roughness.png');
    roughnessMap.anisotropy = maxAnisotropy;
    roughnessMap.wrapS = THREE.RepeatWrapping;
    roughnessMap.wrapT = THREE.RepeatWrapping;
    roughnessMap.repeat.set(tiling, tiling);

    const material = new THREE.MeshStandardMaterial({
        metalnessMap: metalMap,
        map: albedo,
        normalMap: normalMap,
        roughnessMap: roughnessMap,
    });

    return material;
}
```

Demo