Vision Arch
Documentation

Create the project

Create and setup the project

```
npm create vite@latest my-3js-app --template vanilla
cd my-3js-app
```

Setup the Scene

Before we get to the reflections we need to setup a basic room with:

- Walls, Floor, Roof
- Lighting
- A Window on one or two walls
- Realistic Sky

Install required libraries

```
npm install dat.gui postprocessing realism-effects three three-csg-ts
```

Paste the following in your index.html file in the root directory.

Index.htm

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Three.js Example</title>

  <style>
    /* Import Google Fonts */
    @import
url('https://fonts.googleapis.com/css2?family=Nova+Square&display=swap');
    @import
url('https://fonts.googleapis.com/css2?family=Quicksand&display=swap');
```

```css
/* General reset */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

body,
html {
  height: 100%;
  font-family: 'Nova Square', sans-serif;
  background-color: #f0f4f8;
  /* Light background */
  font-size: large;
  /* letter-spacing: 3ch; */
}

/* Styling for screens */
.screen {
  display: none;
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
}


.active {
  display: flex;
}

/* Menu styling */
#menuScreen {
  position: relative;
  /* Necessary for video layering */
  background: transparent;
  /* Make background transparent */
```

```
    color: white;
    text-align: center;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    transition: background 0.3s ease;
  }

  #menuScreen h1 {
    font-size: 3em;
    margin-bottom: 90px;
    animation: fadeIn 1s ease forwards;
    z-index: 2;
    /* Ensure content stays above the video */
    color: black;
    /* Set font color to black */
  }

  /* Button styling */
  button {
    font-size: 1.3em;
    padding: 20px;
    margin: 20px 0;
    /* Increased vertical margin */
    border: none;
    cursor: pointer;
    background-color: #000000;
    color: #E3E2DE;
    font-weight: 600;
    box-shadow: 0px 4px 10px rgba(0, 0, 0, 0.1);
    transition: transform 0.2s, box-shadow 0.2s;
    width: 295px;
    /* Fixed width for equal button sizes */
    animation: fadeInUp 1.2s ease forwards;
    z-index: 2;
    /* Ensure buttons stay above the video */
    font-family: 'Quicksand';
    font-size: large;
    position: relative;
    overflow: hidden;
```

```
    letter-spacing: 0.1ch;
  }

  /* Button hover effect */
  .btn-4 {
    border: 1px solid;
    position: relative;
    overflow: hidden;
  }

  .btn-4 span {
    z-index: 20;
  }

  .btn-4:after {
    background: #fff;
    content: "";
    height: 155px;
    left: -75px;
    opacity: 0.2;
    position: absolute;
    top: -50px;
    transform: rotate(35deg);
    transition: all 1200ms cubic-bezier(0.19, 1, 0.22, 1);
    width: 50px;
    z-index: -10;
  }

  .btn-4:hover:after {
    left: 120%;
    transition: all 1200ms cubic-bezier(0.19, 1, 0.22, 1);
  }

  button:hover {
    transform: translateY(-5px);
    box-shadow: 0px 8px 20px rgba(0, 0, 0, 0.2);
  }

  /* Hide exit button by default */
  #exitButton {
    display: none;
```

```
  position: absolute;
  top: 10px;
  left: 10px;
  padding: 10px;
  background-color: red;
  color: white;
  border-radius: 5px;
  cursor: pointer;
  z-index: 2;
}

/* Show exit button when gameScreen is active */
#gameScreen.active #exitButton {
  display: block;
}

/* Background video styling */
#backgroundVideo {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  object-fit: cover;
  opacity: 0.2;
  /* Set opacity for translucency */
  z-index: 1;
  /* Video should be behind content */
}

/* Animations */
@keyframes fadeIn {
  0% {
    opacity: 0;
  }

  100% {
    opacity: 1;
  }
}
```

```
    @keyframes fadeInUp {
      0% {
        opacity: 0;
        transform: translateY(20px);
      }

      100% {
        opacity: 1;
        transform: translateY(0);
      }
    }
  </style>
</head>

<body>
  <!-- Background video -->

  <!-- Menu Screen -->
  <div id="menuScreen" class="screen active">
    <h1 style="letter-spacing: 1.5ch;">Realistic Reflections</h1>
    <button id="uploadButton" class="btn-4"><span> Just a
button</span></button>
    <button id="exploreButton" class="btn-4"><span>EXPLORE SAMPLE
HOUSE</span></button>
  </div>

  <!-- Game Screen (Cube) -->
  <div id="gameScreen" class="screen">
    <div id="exitButton">EXIT</div>
    <div id="colorPickerContainer" style="display:none;">
      <input type="color" id="floorColorInput" />
    </div>
  </div>

  <script type="module" src="/main.js"></script>
</body>
<script>
  document.addEventListener('DOMContentLoaded', function () {
    const video = document.getElementById('backgroundVideo');
    video.playbackRate = 0.83; // Adjust this value to control speed (0.5 is
half speed)
```

```
  });
</script>

</html>
```

The following is to be pasted in main.js in the root directory.

Import the required libraries

```
import * as THREE from 'three';
import { PointerLockControls } from
'three/examples/jsm/controls/PointerLockControls.js';
import { GLTFLoader } from 'three/examples/jsm/loaders/GLTFLoader.js';
import { RGBELoader } from 'three/examples/jsm/loaders/RGBELoader.js';
import { CSG } from 'three-csg-ts';
```

Initialise the buttons

```
const exitButton = document.getElementById('exitButton');
const menuScreen = document.getElementById('menuScreen');
const gameScreen = document.getElementById('gameScreen');
const uploadButton = document.getElementById('uploadButton');
const exploreButton = document.getElementById('exploreButton');
```

Declare all the global variables we will be using

```
let scene, camera, renderer, animationId, controls;
let cubeCamera, cubeRenderTarget;
const moveSpeed = 0.1;
const movementState = {
    moveForward: false,
    moveBackward: false,
    moveLeft: false,
    moveRight: false
};
```

The init function:

- **Cleanup:** Checks if a renderer already exists, disposes of it, and removes it from the DOM.
- **Scene Creation:** Creates a new `THREE.Scene` to hold all 3D objects.
- **Camera Setup:** Creates a `THREE.PerspectiveCamera` with a field of view of 75 degrees, aspect ratio based on the window size, and a near and far clipping plane. Positions the camera 5 units back and 2 units up.
- **Renderer Creation:** Creates a `THREE.WebGLRenderer` with anti-aliasing enabled. Sets the renderer size to match the window dimensions and enables shadow mapping. Appends the renderer's DOM element to the `gameScreen` element.
- **Pointer Lock Controls:** Initializes `PointerLockControls` to allow first-person camera movement. Listens for clicks to lock the pointer.
- **Background and Environment:** Sets the background color and loads an HDR environment map to provide realistic lighting and reflections.
- **Cubemap Setup:** Creates a cubemap to capture reflections from the environment.
- **Floor Addition:** Adds a floor to the scene, using the cubemap as a reflective texture.
- **Model Loading:** Loads 3D models into the scene.
- **Lighting Setup:** Adds lighting to the scene, likely using directional, ambient, or point lights.
- **Animation Loop:** Starts the animation loop to render the scene continuously.

2. Helper Objects and Rendering Settings:

- **Grid and Axes Helpers:** Adds a grid and axes helper to the scene for visualization.
- **Output Encoding and Tone Mapping:** Configures the renderer's output encoding and tone mapping to improve image quality and appearance.

3. Event Listeners:

- **Keyboard Input:** Sets up event listeners for keydown and keyup events to handle user input.

The key things to notice here are the initialisation of the cubeCamera that we will be using for reflections and the arguements to it which are (near, far , object that creates the map and texture).

The init Function

```
function initThreeJS() {
    if (renderer) { renderer.dispose();
gameScreen.removeChild(renderer.domElement); }
    scene = new THREE.Scene();

    camera = new THREE.PerspectiveCamera(75, window.innerWidth /
```

```
window.innerHeight, 0.1, 50);
    camera.position.set(0, 2, 5);

    renderer = new THREE.WebGLRenderer({ antialias: true });
    renderer.setSize(window.innerWidth, window.innerHeight);
    renderer.shadowMap.enabled = true;
    renderer.shadowMap.type = THREE.PCFSoftShadowMap;
    gameScreen.appendChild(renderer.domElement);

    controls = new PointerLockControls(camera, renderer.domElement);
    gameScreen.addEventListener('click', () => {
        controls.lock();
    });

    renderer.setClearColor(0xA3A3A3);
    const rgbeLoader = new RGBELoader();
    rgbeLoader.load('./assets/textures/pure_sky.hdr', (texture) => {
    texture.mapping = THREE.EquirectangularReflectionMapping;
    texture.encoding = THREE.sRGBEncoding;

    scene.environment = texture;
    scene.background = texture;

    const cubeRenderTargetSize = 256;
    cubeRenderTarget = new THREE.WebGLCubeRenderTarget(cubeRenderTargetSize,
{
        format: THREE.RGBFormat,
        generateMipmaps: true,
        minFilter: THREE.LinearMipmapLinearFilter,
    });
    cubeCamera = new THREE.CubeCamera(0.1, 50, cubeRenderTarget);
    scene.add(cubeCamera);

    addFloor(cubeRenderTarget.texture);


    loadModels(cubeRenderTarget.texture);
    addLighting();
    animate();
});
    const gridHelper = new THREE.GridHelper(40, 40, 0x0000ff, 0x808080);
```

```
    const axesHelper = new THREE.AxesHelper(5);
    scene.add(axesHelper);

    renderer.outputEncoding = THREE.sRGBEncoding;
    renderer.toneMapping = THREE.NeutralToneMapping;
    renderer.toneMappingExposure = 0.4;

    document.addEventListener('keydown', onKeyDown);
    document.addEventListener('keyup', onKeyUp);
}
```

Load Models:

- Loads a 3D model in GLTF format.
- Scales and positions the model in the scene.
- Configures the model's materials to use environment mapping and adjust their appearance.
- Enables shadow casting and receiving for the model.
- Adds the model to the scene, making it visible.

In Here, Notice the line `child.material.envMap = cubeRenderTarget.texture;`
This basically is setting the env map of the object to the texture created by the cubeCamera.(which is equivalent to any env map that are available on the internet)

Load a gltf/glb model that you want the reflections on

```
function loadModels() {
    const loader = new GLTFLoader();
    loader.load('./assets/models/fridge.glb', (gltf) => {
        const model = gltf.scene;

        const scaleFactor = 0.015;
        model.scale.set(scaleFactor, scaleFactor, scaleFactor);
        model.position.set(0, 0, -9);

        model.traverse((child) => {
            if (child.isMesh) {

                child.material.envMap = cubeRenderTarget.texture;
                child.material.envMapIntensity = 1;
                child.material.roughness = 0.05;
                child.material.metalness = 0.8;

                child.castShadow = true;
                child.receiveShadow = true;
            }
        });


        scene.add(model);
    });
}
```

AddFloor:

- Adds the floor and the texture to it. Also allows to receive shadows.
- Adds the ceiling and the texture to it. Also allows to cast and receive shadows.
- Adds the texture to the walls , and allows to cast and receive shadows.
- Adds a wall with window using CSG operation which basically takes a sold mesh of the wall and converts it into a CSG object (solid object) and also takes the mesh of

the window , converts it to a CSG object , subtracts the window object from the wall to make it look like a empty rectangle in the middle of the wall.

Note here that we have set the env map of the wall to the texture that the cubeCamera gives us , This is not necessary as the walls are not reflective and will not make much deifference. This line can be removed for slightly improved performance .

Adding the floor , ceiling and the wall which includes the window

```
function addFloor() {
    const textureLoader = new THREE.TextureLoader();

    // Load floor textures
    const floorGeometry = new THREE.PlaneGeometry(20, 20, 512, 512);
    const baseColor =
textureLoader.load('./assets/textures/floor/granite/basecolour.jpg');
    const displacementMap =
textureLoader.load('./assets/textures/floor/granite/displacement.tiff');
    const normalMap =
textureLoader.load('./assets/textures/floor/granite/normal.png');
    const roughnessMap =
textureLoader.load('./assets/textures/floor/granite/roughness.jpg');
    const metallicMap =
textureLoader.load('./assets/textures/floor/granite/metallic.jpg');

    [baseColor, displacementMap, normalMap, roughnessMap,
metallicMap].forEach((texture) => {
        texture.wrapS = texture.wrapT = THREE.RepeatWrapping;
        texture.repeat.set(4, 4);
    });

    //Create the textured floor material
    const floorMaterial = new THREE.MeshPhysicalMaterial({
        map: baseColor,
        displacementMap: displacementMap,
        normalMap: normalMap,
        roughnessMap: roughnessMap,
        metalnessMap: metallicMap,
        side: THREE.DoubleSide,
        metalness: 0.9,
        roughness: 0.05,
        Opacity:0.8,
```

```
    });


    // Create the floor mesh
    const floor = new THREE.Mesh(floorGeometry, floorMaterial);
    floor.rotation.x = -Math.PI / 2;
    floor.receiveShadow = true;
    scene.add(floor);

    // Wall Material
    const wallDiffuseTexture =
textureLoader.load('./assets/textures/wall/textures/diff.jpg');
    const wallNormalTexture =
textureLoader.load('./assets/textures/wall/textures/nor.jpg');
    const wallGlossinessTexture =
textureLoader.load('./assets/textures/wall/textures/arm.jpg');

    [wallDiffuseTexture, wallNormalTexture,
wallGlossinessTexture].forEach((texture) => {
        texture.wrapS = texture.wrapT = THREE.RepeatWrapping;
        texture.repeat.set(3, 2);
    });

    const wallMaterial = new THREE.MeshStandardMaterial({
        map: wallDiffuseTexture,
        normalMap: wallNormalTexture,
        roughnessMap: wallGlossinessTexture,
        roughness: 0.4,
        metalness: 0.2,
        envMap: cubeRenderTarget.texture,
    });

    // Ceiling
    const ceilingGeometry = new THREE.PlaneGeometry(25, 25);
    const ceiling = new THREE.Mesh(ceilingGeometry, wallMaterial);
    ceiling.rotation.x = Math.PI / 2;
    ceiling.position.set(0, 4, 0);
    ceiling.receiveShadow = true;
    ceiling.castShadow=true;
    scene.add(ceiling);
```

```
    // Wall Geometry
    const wallWidth = 25, wallHeight = 6, wallThickness = 0.1;
    const wallGeometry = new THREE.BoxGeometry(wallWidth, wallHeight,
wallThickness);

    // Window for CSG operation
    const windowWidth = 5, windowHeight = 2;
    const windowGeometry = new THREE.BoxGeometry(windowWidth, windowHeight,
wallThickness + 0.01);

    // Back Wall with CSG window
    const backWall = new THREE.Mesh(wallGeometry, wallMaterial);
    const windowMeshBack = new THREE.Mesh(windowGeometry);
    windowMeshBack.position.set(0, wallHeight / 2 - 1, 10);

    const wallCSG = CSG.fromMesh(backWall);
    const windowCSG = CSG.fromMesh(windowMeshBack);
    const wallWithWindowCSG = wallCSG.subtract(windowCSG);
    const wallWithWindow = CSG.toMesh(wallWithWindowCSG, backWall.matrix,
wallMaterial);

    wallWithWindow.position.set(0, wallHeight / 2 - 1, 10);
    wallWithWindow.castShadow = true;
    wallWithWindow.receiveShadow = true;
    scene.add(wallWithWindow);

    // Left, Right, and Front Walls
    const leftWall = new THREE.Mesh(wallGeometry, wallMaterial);
    leftWall.position.set(-10, wallHeight / 2 - 1, 0);
    leftWall.rotation.y = Math.PI / 2;
    leftWall.castShadow = true;
    leftWall.receiveShadow = true;
    scene.add(leftWall);

    const rightWall = new THREE.Mesh(wallGeometry, wallMaterial);
    rightWall.position.set(10, wallHeight / 2 - 1, 0);
    rightWall.rotation.y = Math.PI / 2;
    rightWall.castShadow = true;
    rightWall.receiveShadow = true;
    scene.add(rightWall);
```

```
    const frontWall = new THREE.Mesh(wallGeometry, wallMaterial);
    frontWall.position.set(0, wallHeight / 2 - 1, -10);
    frontWall.castShadow = true;
    frontWall.receiveShadow = true;
    scene.add(frontWall);

    renderer.render(scene, camera);
}
```

Add Lighting:

- Adds ambient lighting for overall soft light illumination.
- Adds directional light to mimic sunlight.
- Adds point lights in opposite corners of the room to mimic lights(although might not look the best , you can try playing with the placement, intensity and colour of the light).

Add Lighting which includes ambient, directional and point lights.

```
function addLighting() {
    // Ambient light for soft general illumination
    const ambientLight = new THREE.AmbientLight(0x404040, 8); // Soft white light
    scene.add(ambientLight);

    // Single Directional Light positioned from the ceiling
    const directionalLight = new THREE.DirectionalLight(0xffffff, 4); // Color and intensity
    directionalLight.position.set(-10, 5, 15); // Positioned directly above the center of the room
    directionalLight.castShadow = true;

    // Adjust shadow camera to cover the room area
    directionalLight.shadow.camera.left = -40;
    directionalLight.shadow.camera.right = 40;
    directionalLight.shadow.camera.top = 40;
    directionalLight.shadow.camera.bottom = -40;
    directionalLight.shadow.mapSize.width = 2048; // Increased resolution
    directionalLight.shadow.mapSize.height = 2048;
```

```
    scene.add(directionalLight);

    // Optional: Additional Directional Light
    const dL2 = new THREE.DirectionalLight(0xffffff, 4); // Color and
intensity
    dL2.position.set(-2, 5, 10); // Positioned directly above the center of
the room
    dL2.castShadow = true;

    dL2.shadow.camera.left = -12;
    dL2.shadow.camera.right = 12;
    dL2.shadow.camera.top = 7;
    dL2.shadow.camera.bottom = -7;
    dL2.shadow.mapSize.width = 2048;
    dL2.shadow.mapSize.height = 2048;

    // scene.add(dL2); // Add if needed

    // Point Lights - for localized lighting (e.g., lamps, spot
illumination)

    // Point Light 1
    const pointLight1 = new THREE.PointLight(0xffccaa, 3, 20); // Warm
light, high intensity, 20 units range
    pointLight1.position.set(5, 3, 5); // Position above a certain point in
the room
    pointLight1.castShadow = true;
    scene.add(pointLight1);

    // Point Light 2
    const pointLight2 = new THREE.PointLight(0xaaccff, 3, 20); // Cool
light, high intensity, 20 units range
    pointLight2.position.set(-5, 3, -5); // Position in another part of the
room
    pointLight2.castShadow = true;
    scene.add(pointLight2);

}
```

Functions that enable First person like movement

```
function onKeyDown(event) {
    switch (event.code) {
        case 'KeyW':
            movementState.moveForward = true;
            break;
        case 'KeyS':
            movementState.moveBackward = true;
            break;
        case 'KeyA':
            movementState.moveLeft = true;
            break;
        case 'KeyD':
            movementState.moveRight = true;
            break;
    }
}

function onKeyUp(event) {
    switch (event.code) {
        case 'KeyW':
            movementState.moveForward = false;
            break;
        case 'KeyS':
            movementState.moveBackward = false;
            break;
        case 'KeyA':
            movementState.moveLeft = false;
            break;
        case 'KeyD':
            movementState.moveRight = false;
            break;
    }
}

function updateMovement() {
    if (movementState.moveForward) controls.moveForward(moveSpeed);
    if (movementState.moveBackward) controls.moveForward(-moveSpeed);
    if (movementState.moveLeft) controls.moveRight(-moveSpeed);
    if (movementState.moveRight) controls.moveRight(moveSpeed);
```

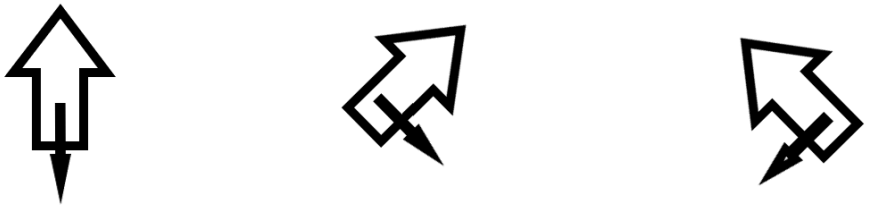
```
}
```

Animate function:

This is where the magic happens, the explanation to what is happening is as follows:

- Every frame the movement is being updated.
- The cubeCamera which is helping us create reflections is copying the coordinates of the camera, essentially where ever you move, the cubeCamera follows.
- The cubeCamera's x rotation (horizontal) is set to the negative value of the camera's x rotation signifying when the main camera turns right , the cubeCamera also turns right , but in the opposite direction of the the main camera (facing backwards direction to the main camera).

Hollow arrow -> Main camera
Solid arrow -> cubeCamera



- By setting the cube camera's y rotation to the negative of the main camera's y rotation, you ensure that the top and bottom faces of the cubemap are captured from the correct perspective relative to the main camera's viewpoint.

The Animate function

```
function animate() {
    requestAnimationFrame(animate);
    updateMovement();

    // Update the main CubeCamera to follow the main camera
    cubeCamera.position.copy(camera.position);
    cubeCamera.rotation.y = -camera.rotation.y;
    cubeCamera.rotation.x = -camera.rotation.x;
    cubeCamera.update(renderer, scene);

    renderer.render(scene, camera);
```

```
}
```

Functions that handle the starting, ending, resizing of the game scene

```
function startGame() {
    menuScreen.classList.remove('active');
    menuScreen.style.display = 'none';
    gameScreen.classList.add('active');
    gameScreen.style.display = 'flex';

    if (!renderer) {
        initThreeJS();
    }
}

function cleanupScene() {
    scene.traverse((object) => {
        if (object.isMesh) {
            object.geometry.dispose();

            if (Array.isArray(object.material)) {
                object.material.forEach((material) => material.dispose());
            } else {
                object.material.dispose();
            }
        }
    });

    while (scene.children.length > 0) {
        const child = scene.children[0];
        scene.remove(child);
    }

    if (renderer) {
        renderer.dispose();
        gameScreen.removeChild(renderer.domElement);
    }

    if (animationId) {
        cancelAnimationFrame(animationId);
```

```
    }

    scene = null;
    camera = null;
    renderer = null;
    controls = null;
}

function exitGame() {
    gameScreen.classList.remove('active');
    gameScreen.style.display = 'none';
    menuScreen.classList.add('active');
    menuScreen.style.display = 'flex';

    cleanupScene();
}

uploadButton.addEventListener('click', () => {
    console.log('Upload clicked');
});

exploreButton.addEventListener('click', startGame);
exitButton.addEventListener('click', exitGame);
```

Paste all of these one by one onto the main.js file in the root directory.
Create a folder named assets inside which there are two more folders namely models which contain the glb model of the fridge and textures which contain the hdr env map of the scene. These files will be shared with the document.

Finally to to run the project

Command to run the project

```
npm run dev
```

Instructions:

You will see two buttons which includes a dummy button and explore sample house button. Please ignore the dummy button and click on the explore sample house button, you should be able to see the room itself with an env mapping and a fridge which has near realistic type reflections.

Disclaimer: This is not portraying the exact reflections on an object, but instead is a easy, light weight workaround to implementing reflections on an object.
You can notice that at critical angles to the object, the reflections are inaccurate and is only accurate up to a certain angle.

If any issues encountered please contact any of the below:
Sai Tarun A : saitarun575@gmail.com
Shreevathsa GP : shreevathsasgp@gmail.com