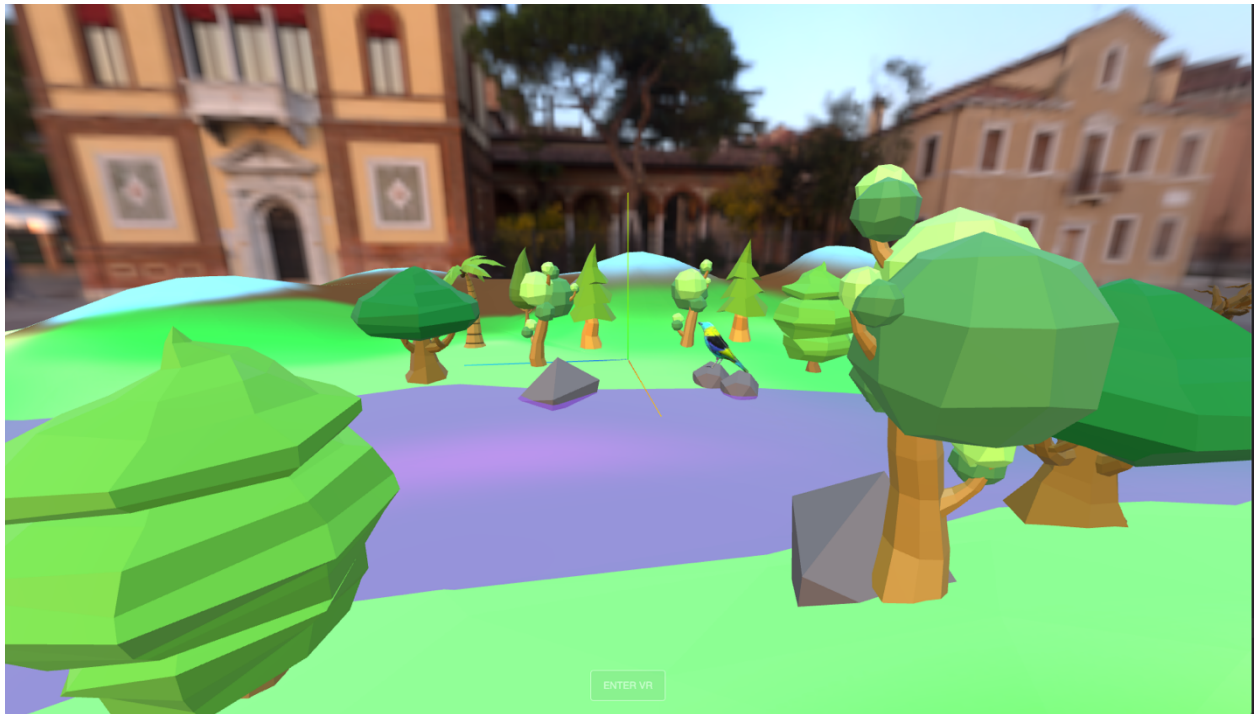
Week 3 and 4 – VR, Controllers, Grabbing



Let us start by adding the VR functionality. Go to:

https://threejs.org/manual/#en/how-to-create-vr-content

To test out the VR without glasses, you will need an emulator. The new, official one came out a while ago:

Immersive Web Emulator (for Chrome)

Install the emulator and try that you are able to enter the VR mode in your browser.

We will follow the example from:

https://github.com/mrdoob/three.js/blob/master/examples/webxr_xr_dragging.html

Before adding the controllers, please tidy up your code. After every step, check that your code runs and no errors are printed to the console.

1. **Add visible Controllers**

First add and initialize some variables in the global scope. Add these after the imports.

Initializing a variable in the global scope means declaring a variable outside of any function or block, making it accessible from anywhere within your JavaScript code. When you initialize a variable in the global scope, it has a global lifetime and can be accessed and modified from any part of the code. This can be useful for variables that need to be accessed by multiple functions or scripts.

We also will need a new module, that will create the controller models, so add that too.

```
	import { XRControllerModelFactory } from
'three/addons/webxr/XRControllerModelFactory.js';

	//////////////
	//Initialize variables
	//////////////

	let controller1, controller2;
	let controllerGrip1, controllerGrip2;

	let raycaster;

	const intersected = [];
	const tempMatrix = new THREE.Matrix4();

	let group;
```

Then after you have created your scene and set up the renderer, make an init function for the VR.

```
	initVR();

	function initVR() {
	  document.body.appendChild(VRButton.createButton(renderer));
	  renderer.xr.enabled = true;
	}
```

Now lets add some code (line 132-167) to the initVR function from the:
https://github.com/mrdoob/three.js/blob/master/examples/webxr_xr_dragging.html

Study:
https://threejs.org/docs/index.html?q=webxr#api/en/renderers/webxr/WebXRManager

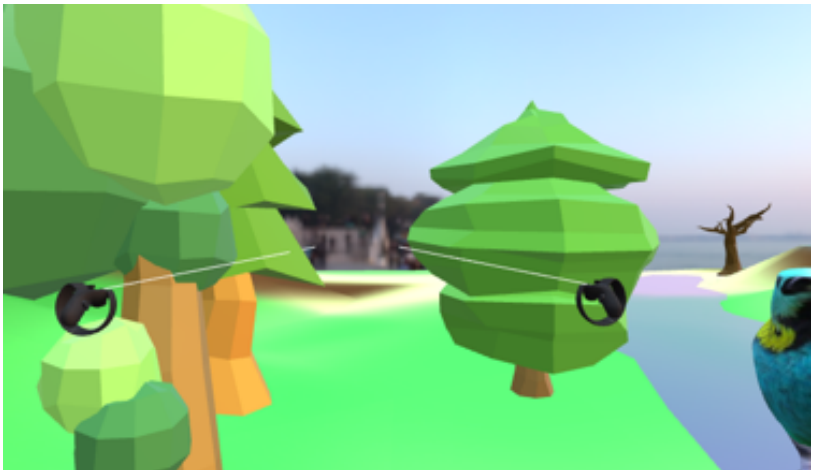
Create also two empty functions for the selectend and selectstart functions. These will not do anything yet, but you need them since you have created EventListeners that trigger the function call.

```
	function onSelectStart(event) {}
```

```
    function onSelectEnd(event) {}
```

I will explain the code for you. Feel free to add some comments.

You should now see your controllers in the world as you move them, but not interact with anything yet. Test your scene at this stage. Test it with the WebXR API and on oculus.



2. **Attach your own model to one of the controllers instead of the default**

You can get the model used in the class from OMA workspace, folder named Gundy.

Lets add this model instead of the default controller. For the return of the assignment, you **should pick your own model to use instead**.

```
const loader = new GLTFLoader().setPath(basePath);
loader.load('gundy/scene.gltf', async function (gltf) {
  gltf.scene.scale.set(0.0003, 0.0003, 0.0003);
  let mymodel = gltf.scene;
  mymodel.rotation.y = THREE.MathUtils.degToRad(180);
  mymodel.rotation.x = THREE.MathUtils.degToRad(-36.5);
  mymodel.position.set(0, 0.01, 0);
  controllerGrip2.add(mymodel);
});
```

Study raycasting
https://davidlyons.dev/threejs-intro/#slide-114

Now if all is working, lets add some more code to actually grab something in the scene. I will explain and comment the code as we go on. Now, copy from the example the functions:

```
function onSelectStart,
function onSelectEnd,
function getIntersections,
function intersectObjects,
function cleanIntersected()
```

In the getIntersections function, change the raycaster to detect also child objects in the scene.

```
    return raycaster.intersectObjects(group.children, true);
```

Add also the lines 279-282 to your animationloop.

```
renderer.setAnimationLoop(function () {
  cleanIntersected();

  intersectObjects(controller1);
  intersectObjects(controller2);

  controls.update();

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

Our code doesn't still work. This is because the raycaster is going through an object3D group, but we haven't added the scene to it. Therefore it is empty and nothing to raycast to. Lets add first create a group, then add the group to our scene. Then add all of our gltf models to the group.

Globally:

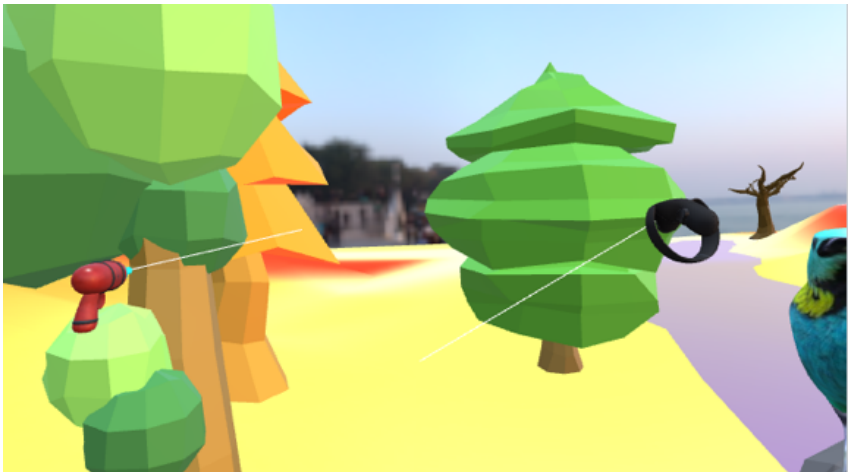
```
group = new THREE.Group();
scene.add(group);
```

Inside our gltf loader:

```
//scene.add(landscape);
group.add(landscape);
```

Now test your scene!! You should have everything working and you can now grab and mode models in your scene. The problem is that you can also grab the earth and the water. You can see the raycast hitting the landscape and coloring it red.

3. **Exclude the landscape from the code**

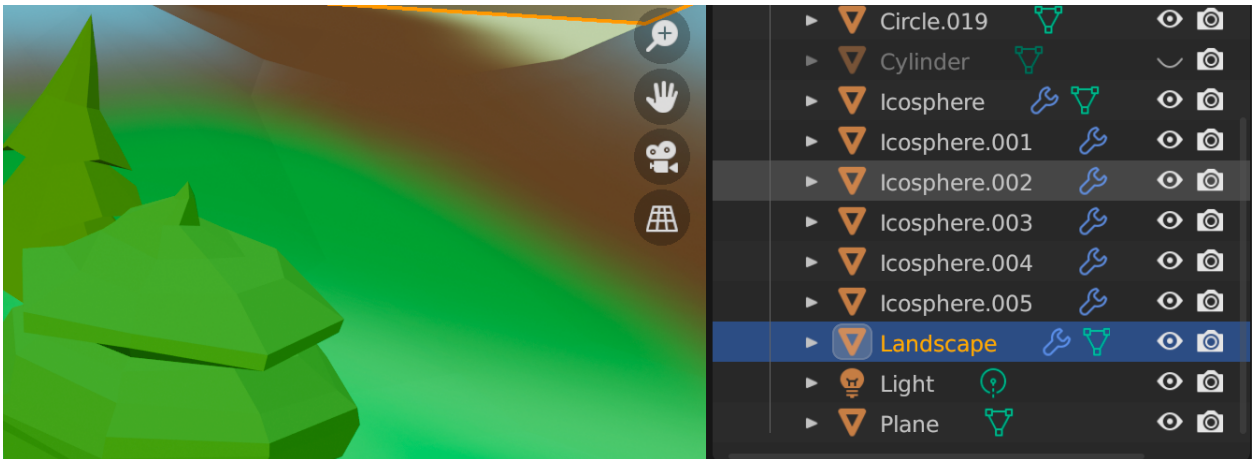


You have two options to deal with this.

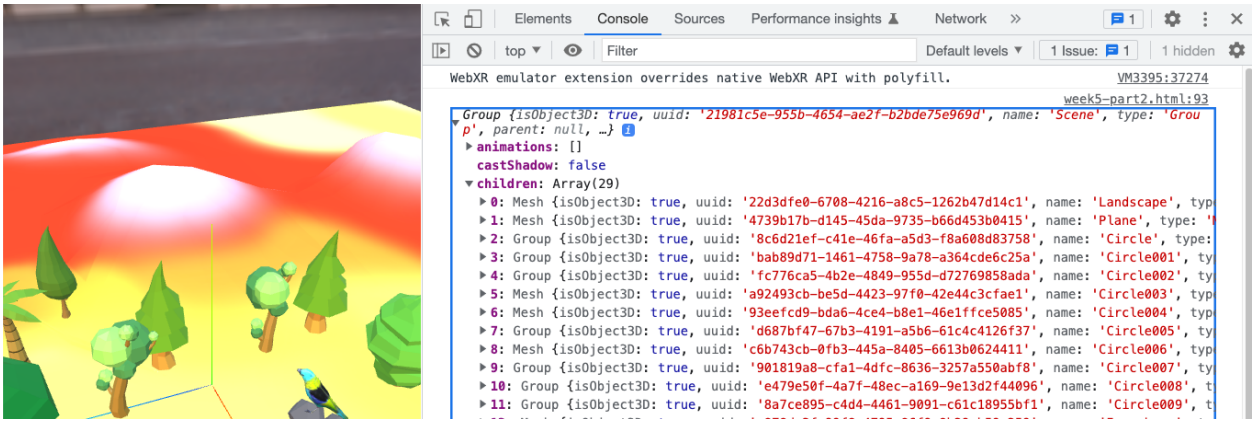
1. Export your world from Blender in two separate files. One with the models you want to interact and one with the rest
    a. Then add the one gltf file you want to interact with to the raycaster group and the other to the scene

```
scene.add(gltfNoInteractions);
group.add(gltfInteractions);
```

2. In the code, create an array where you exclude some objects from your scene based on the name you have given in Blender



If you console.log your imported scene, you can study the objects. And the names. As you can see the names of the meshes in Blender are included in the gltf scene. But not always..



Then simply:

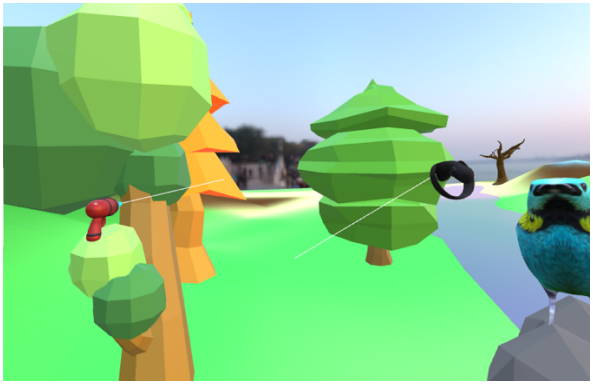
```
const arr = ['Landscape', 'Plane'];
```

And in the selectStart and intersectObjects functions. Exclude the objects in the array.

```
        if (!arr.includes(object.name)) {
          //console.log('picked an object');
          object.material.emissive.b = 1;
          controller.attach(object);
        }
```

```
        if (!arr.includes(object.name)) {
          //console.log('picked an object');
          object.material.emissive.r = 1;
        }
```

Now you should be set up and ready to have fun!



4. **Add shadow to your scene, homework/extra**

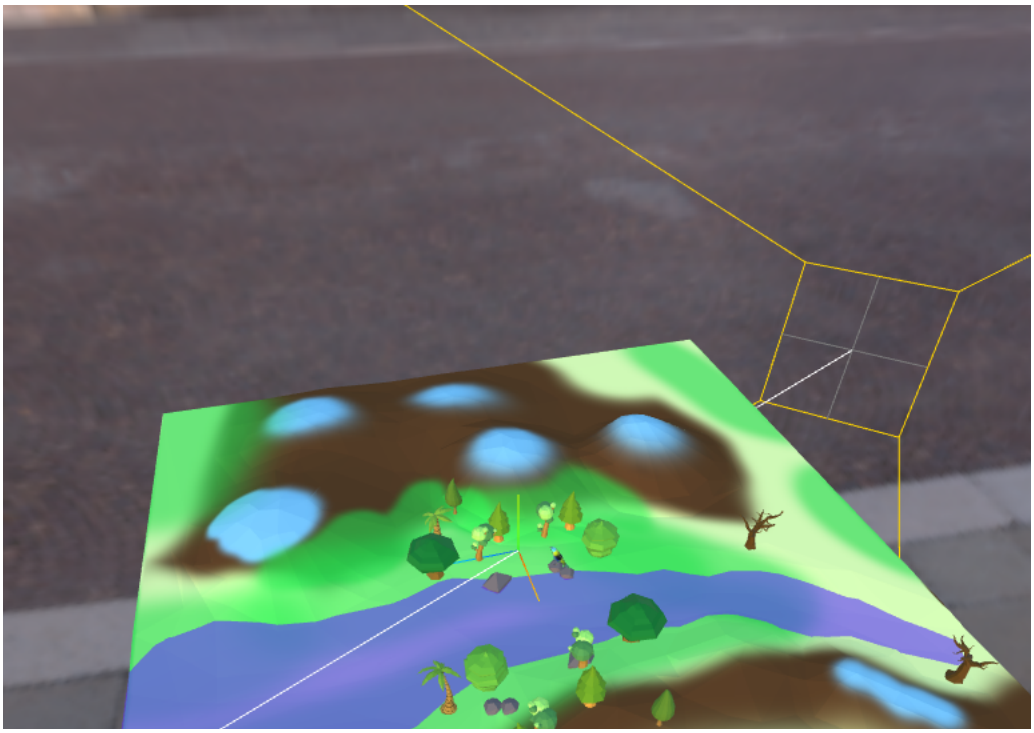
Try to understand the code from the docs. Make sure you have a directional light in your scene. In order to use shadows, they need to be turned on in the renderer, for the light and also the model properties need to be updated to both to cast and receive shadows.

Change the direction of your directional light. Make it shine form an angle.

https://threejs.org/docs/#api/en/lights/shadows/DirectionalLightShadow

Remember to use a helper class here to see when the light is coming from.

```
//Create a helper for the shadow camera (optional)
    const helper = new THREE.CameraHelper(directionalLight.shadow.camera);
    scene.add(helper);
```



Now add shadows to your scene. For the gltf models, you need to go through the full hierarchy to enable both the casting and the receiving.

```
        // before you add your gltf file to the scene
        // go through child nodes and make double sided and
        // recieve/cast shadows
        landscape.traverse(function (node) {
          if (node.material) {
            console.log(gltf.scene);
            node.material.side = THREE.DoubleSide;
            node.castShadow = true;
            node.receiveShadow = true;
          }
        });
```

You might notice that the shadows look horrible. You might want to make the mapsize larger and change the bias. Try different sizes to match your world.

```
      directionalLight.shadow.normalBias = 1;
       directionalLight.shadow.mapSize.set(4096, 4096);
```