# CM20219 - CW 2 - 2020/21

December 13, 2020

## 1 Draw a simple cube

The cube was drawn with side dimensions of 2. Since the default for *scene.add* is to centre the cube at the origin $(0, 0, 0)$, side lengths of 2 gives the corner coordinates $(-1, -1, -1)$ and $(1, 1, 1)$. the *wireframe* parameter simply makes the cube easier to see, and the *color* is blue.

```
//Basic Cube
var boxGeo = new THREE.BoxGeometry(2,2,2);
var boxMat = new THREE.MeshBasicMaterial({color: 0xffffff, wireframe: true});
var cube = new THREE.Mesh(boxGeo,boxMat);
scene.add(cube);
```

Figure 1

## 2 Draw coordinate system axes

The axes are simply drawn using the built-in *AxesHelper* function.

```
var axesHelper = new THREE.AxesHelper(20);
scene.add(axesHelper);
```

Figure 2

## 3 Rotate the cube

Rotation of the cube is performed by adjusting the object (cube) *rotation* property. the *rotateCube*() function is called in *animate*() so that it is constantly updated.

```
function rotateCube(){
        cube.rotation.x -= 0.01;
        cube.rotation.y -= 0.01;
        cube.rotation.z -= 0.01;
}
```

Figure 3

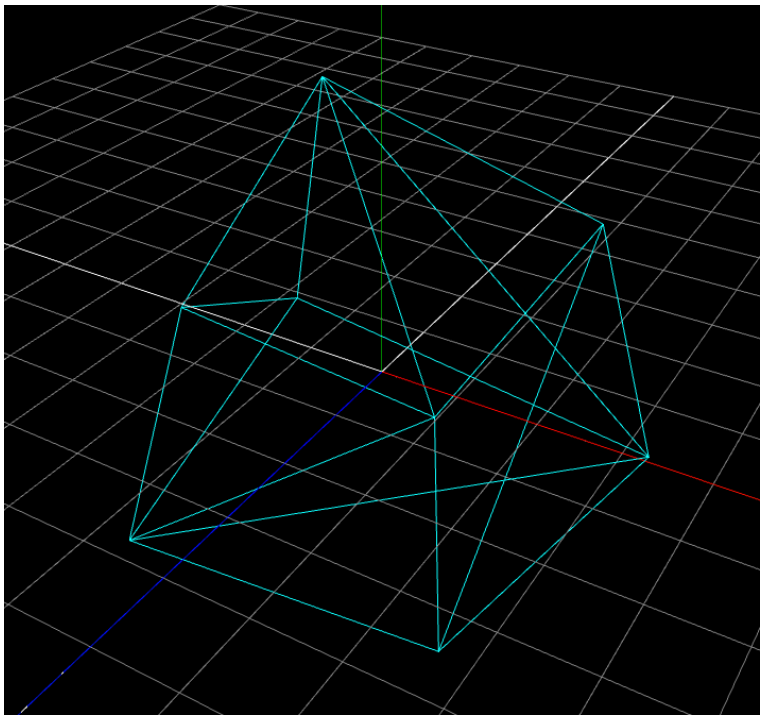

Figure 4

As you can see in Figure 4, the cube now rotates.

# 4 Different Render Modes

Switching render modes is done in the *handleKeyDown*() function, as seen in Figure 5. Each time either $v, e, f$ are pressed, a new *CubeGeometry* is generated. Faces are generated with the same code as requirement 1. The edges are drawn using *EdgeGeometry* and vertices are drawn by using a *PointsMaterial* along with *CubeGeometry* in the mesh. Each keypress starts with *scene.remove*(*cube*), then the changes are made, then the cube is readded to the scene.

```
// Render modes.
case 70: // f = face
    scene.remove(cube);
    var boxGeo = new THREE.CubeGeometry(2,2,2);
    var boxMat = new THREE.MeshBasicMaterial({color: 0xfffff});
    cube = new THREE.Mesh(boxGeo,boxMat);
    scene.add(cube);
    break;

case 69: // e = edge
    scene.remove(cube);
    var boxGeo = new THREE.CubeGeometry(2,2,2);
    var edgeGeo = new THREE.EdgesGeometry(boxGeo);
    var edgeMat = new THREE.LineBasicMaterial( { color: 0xfffff } );
    cube = new THREE.LineSegments(edgeGeo, edgeMat);
    scene.add(cube);
    break;

case 86: // v = vertex
    scene.remove(cube);
    var boxGeo = new THREE.CubeGeometry(2,2,2);
    var vertMat = new THREE.PointsMaterial({color: 0xfffff, size: 0.1});
    cube = new THREE.Points(boxGeo,vertMat);
    scene.add(cube);
    break;
```

Figure 5

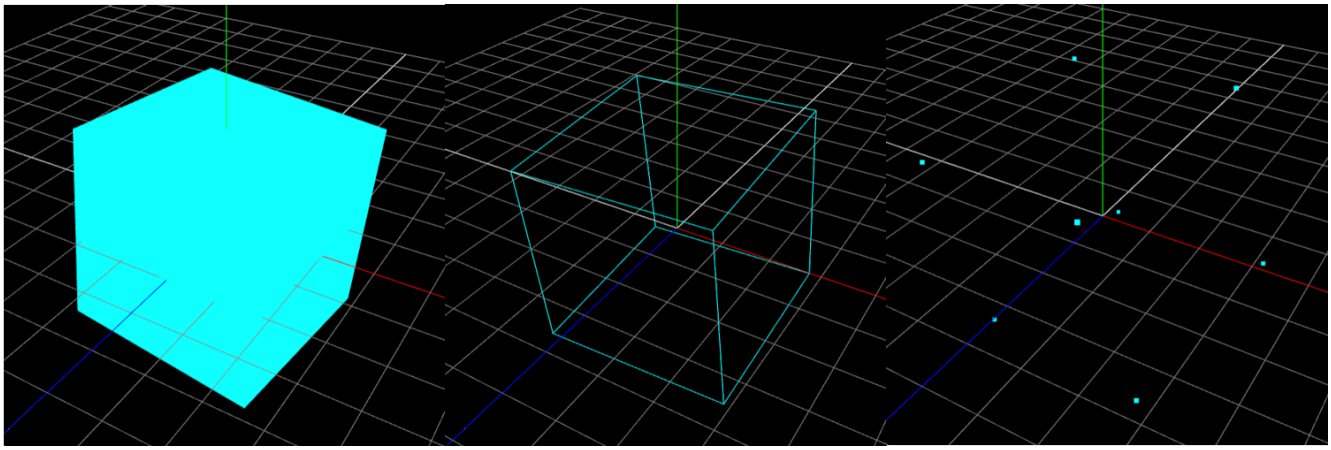

Figure 6

# 5 Translate the Camera

The camera is translated on its local axes using the arrow keys, as seen in Figure 7. Backwards/forwards zooming is handled by the scroll wheel, as seen in Figure 8. In addition to the function shown in Figure 8, A new *EventListener* was added for the mouse wheel, just under the *EventListener* for keypresses.

```
case 38: //Translate Camera UP (UP Arrow Key)
    camera.translateY(1);
    break;

case 40: //Translate Camera DOWN (DOWN Arrow Key)
    camera.translateY(-1);
    break;

case 37: //Translate Camera LEFT (LEFT Arrow Key)
    camera.translateX(-1);
    break;

case 39: //Translate Camera RIGHT (RIGHT Arrow Key)
    camera.translateX(1);
    break;
```

Figure 7

```
//Handle scrolling
function handleScroll(event){
    camera.translateZ(event.deltaY);
}
```

Figure 8