Lesson 2 - Projections, lighting and transformations

Artur Romão (nMec 98470), Eva Bartolomeu (nMec 98513)
Information Visualization, 2023 (MSc Informatics Engineering, University of Aveiro)

Camera models

First, we copied the first exercise from lesson 1, and made the requested changes to the statement. We change the wireframe property to true on the material and remove the rotation from the cube.
Then we changed the camera type to Orthographic Camera (previously it was Perspective Camera). In addition, we pass the following parameters in the camera: left: -3; right: 3; top: 3 * (window.innerHeight / window.innerWidth); bottom: -3 * (window.innerHeight / window.innerWidth); near: 0.1; far: 1000.
PerspectiveCamera and OrthographicCamera are both types of cameras that are used to create an image of a 3D scene from a single viewpoint.
A PerspectiveCamera introduces perspective distortion to create the illusion of depth and distance in the scene, while an OrthographicCamera does not introduce this distortion.
Taking into account the aforementioned definitions, we can say that the results obtained were as expected, since the resulting cube with the Perspective Camera gives the idea of depth and distance in the scene, whereas with the Orthographic Camera, it does not give us that idea.

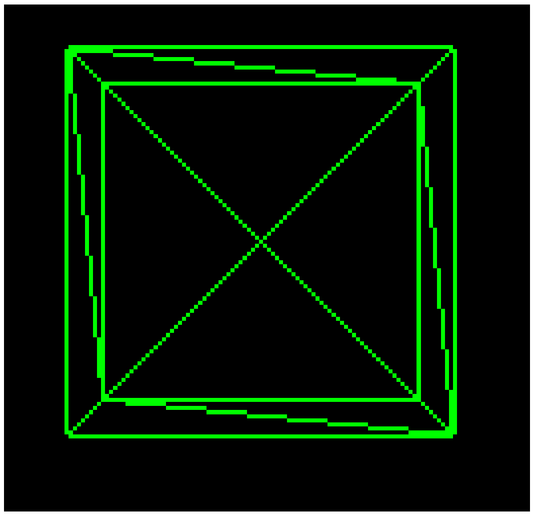


Figure 1 - Result with Perspective Camera.

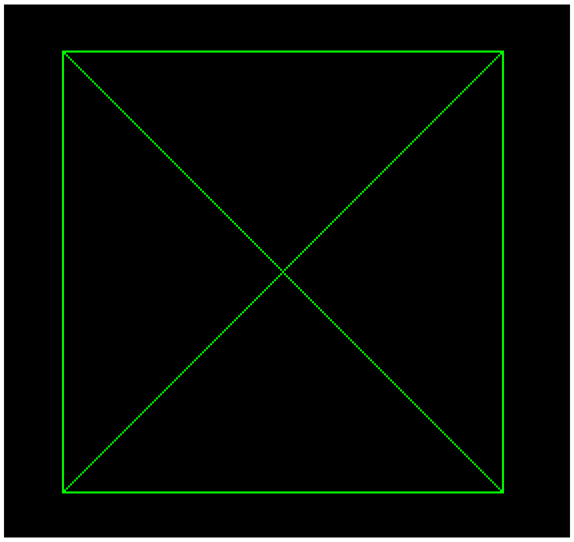


Figure 2 - Result with Orthographic Camera.

Orbit control

There are several controls in Three.js that allow you to interact with a 3D scene (for example: OrbitControls, TrackballControls, FlyControls, FirstPersonControls...).
OrbitControls allows you to rotate, pan and zoom a center point using mouse or touch input.

TrackballControls lets you rotate and zoom using the mouse and pan using the keyboard.
FlyControls lets you fly around the scene using the keyboard.

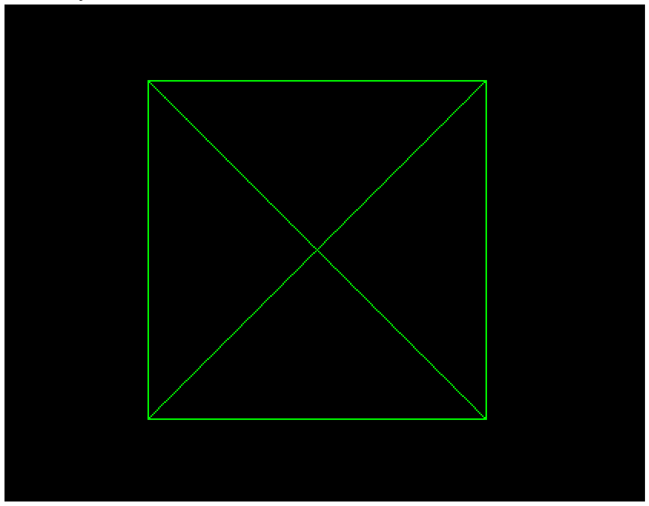


Figure 3 - Result with OrbitControls.

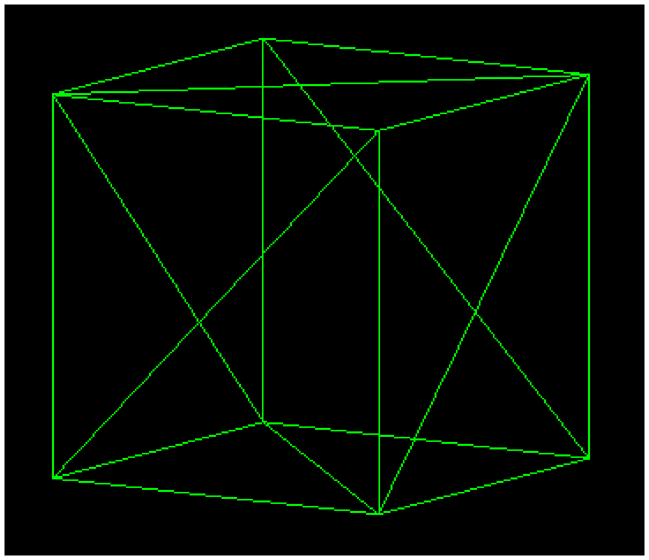


Figure 4 - Result with OrbitControls, after a rotation.

Lighting and materials

In this exercise we added a THREE.DirectionalLight object to the scene, a type of light that shines in a specific direction. We modify the position of the light, placing it at a height of 5, and set the light to white with an intensity of 1.
After these codes, we noticed that there were no changes in the scene, this is because the THREE.MeshBasicMaterial is not affected by light. Then we change the cube's material to the MeshPhongMaterial, where we also define the color, specular and shine.
In order to be able to see the color of the material, we add a THREE.AmbientLight, in white. This object is a light that fills all objects in a scene equally, without a specific direction, with a uniform and diffused light.

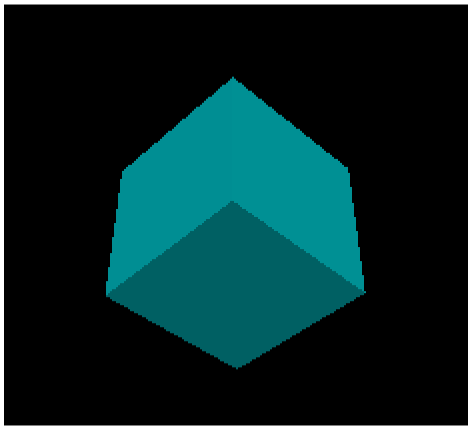


Figure 5 - Result of the Lighting and materials.

Shading

In this exercise we changed the rotating cube simple example to have two spheres rotating instead.
First, we modified the parameters *widthSegments* and *heightSegments*, which represent, respectively, the number of horizontal segments and the number of vertical segments, respectively. The bigger those numbers are, the more perfect the sphere will be.
Then, we added the ambient and directional lights from the previous exercise and applied *MeshPhongMaterial* to both spheres, changed the *flatShading* parameter to true for one of them and to false for the other one. We observed that for *flatShading*: true (sphere on the left), the lights "followed" the rotation of the sphere and didn't stop moving. On the other hand, for *flatShading*: false (sphere on the right), the lights remained still besides the rotation of the sphere.

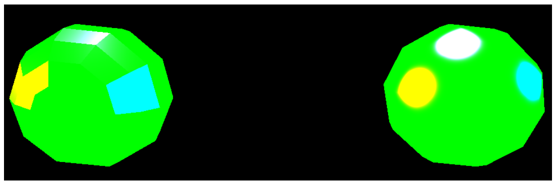


Figure 6 - Result with the left sphere having *flatShading: true* and the right sphere having *flatShading: false*.

As an optional alinea, we were asked to apply a *MeshLambertMaterial* to the first sphere and remove the *specular* and *shininess* attributes (even though they are ignored). By doing this, the shade disappeared and we could not see the reflection of the lights in the sphere.
Then, we were asked to modify the properties of the spheres by selecting some values in a given table and see the effects of the different materials, here is an example with the *emerald MeshPhongMaterial*:

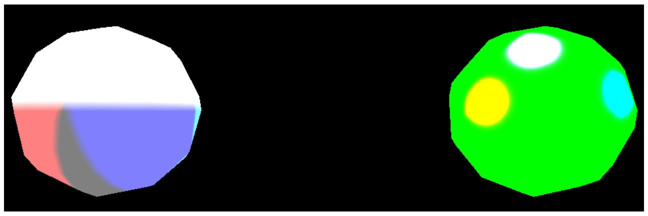


Figure 7 - Result with the left sphere having *emerald MeshPhongMaterial*.

Finally, as another optional alinea, we had to put different lights in different positions pointing to different directions, you can see that in Figure 6.

Transparency

In this exercise we were supposed to change the previous script by adding more spheres or cubes around the original spheres and change the material of the new shapes to receive a parameter *opacity* and a parameter *transparent*. We decided to add three new cubes with *opacity: 0.4* and *transparency: true*. This was the result:

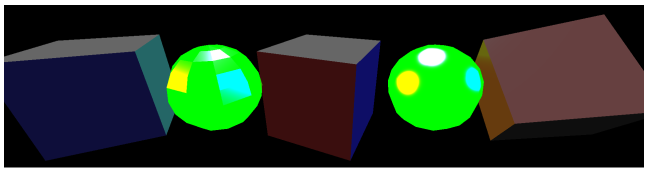


Figure 8 - Result for *opacity: 0.4*.

By analyzing the Figure 8, we can denote that the cube faces reflect the color of a light according to the color of the *DirectionalLight* that is falling in the face.
The closer the parameter *opacity* is to 0, the less the cube will reflect the colors of the lights and the darker it will be. Below you can see two examples, the first one using *opacity: 0.2* and the other one using *opacity: 0.8*.

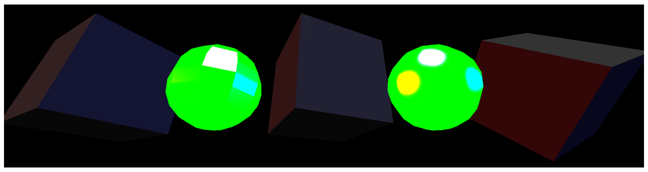


Figure 9 - Result for *opacity: 0.2*.

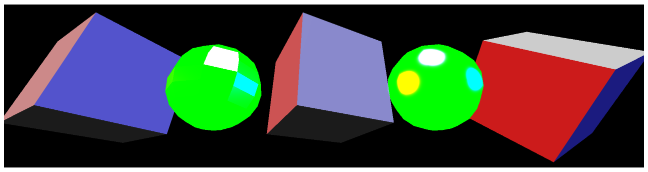


Figure 10 - Result for *opacity: 0.8*.

Transformations (scale and rotation)

In this exercise we created a transparent light blue box and four dark orange spheres using the scale properties, this was the result:

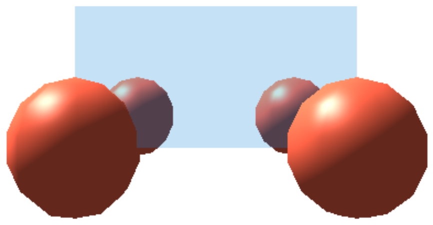


Figure 11 - Frontal view of the created object.

Instead of adding multiple meshes and having an unnecessary big code, we added the box and sphere meshes into a single *THREE.Object3D()* as suggested by the teacher. Below you can see another perspective of the object using Orbit Control:

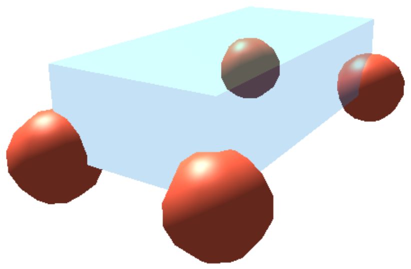
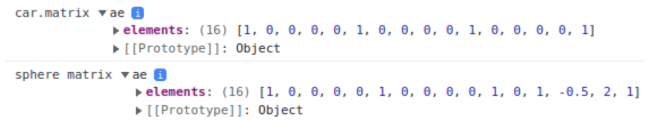


Figure 12 - Another view of the created object.

In the exercise it is also asked to access the transformation matrices, here they are:



```
car.matrix ▾ae
              ▸elements: (16) [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1]
              ▸[[Prototype]]: Object
sphere matrix ▾ae
                ▸elements: (16) [1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, -0.5, 2, 1]
                ▸[[Prototype]]: Object
```

Figure 13 - Transformation matrices of the Objects.

Transformations (rotations)

In this exercise we started by creating an object that represents a coordinate system using three cylinders, a red, a green and a blue one for each axis. All those cylinders belong to a single *THREE.Object3D()* called *"axis"*.
Then, we used the object created in the last scene and changed the spheres to cylinders so that the object would look more like a car.
Then, since the car was an only object, we could move it and we decided to make it go back and forth.

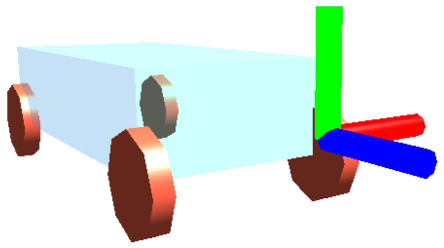


Figure 14 - Result of the final exercise, a car and an object containing the three axis.

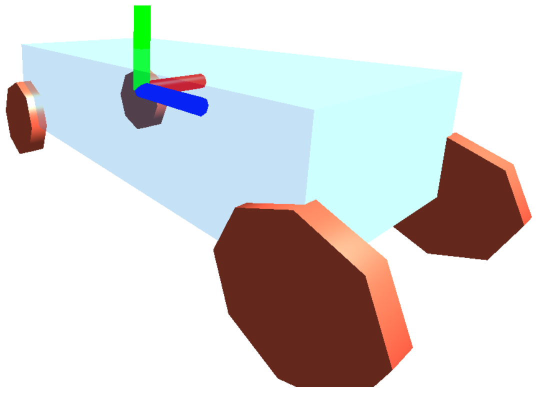


Figure 14 - Result of the final exercise, with the car closer to the camera.