Lesson 3 - Texture and Interaction

Course: Information Visualization - 44156

André Fernandes (97977) 50.0%, Gonçalo Machado (98359) 50.0%

27/12/2023

Class - TP2

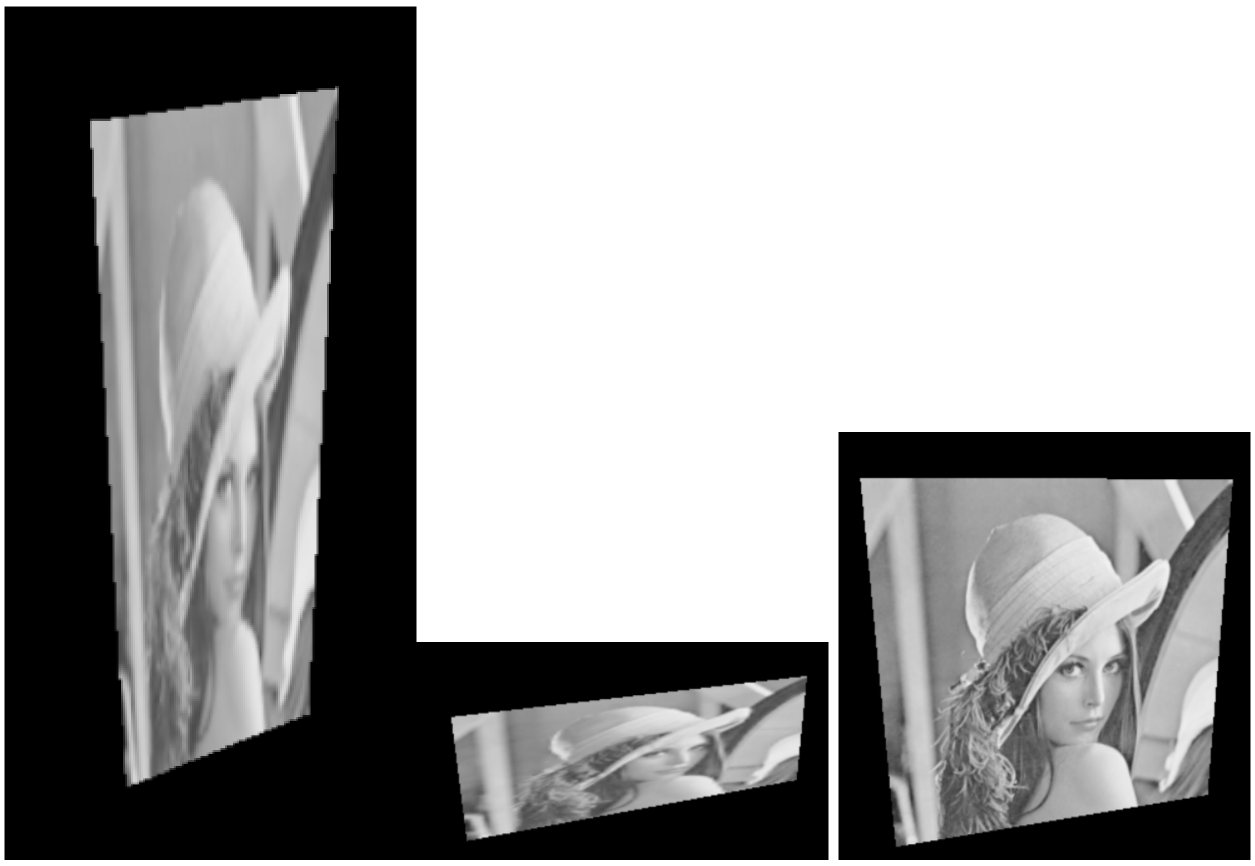
Developed *software* can be found **here**.

Introduction

In the Lesson 3 of the Computer Graphics module, the focus will be on textures and interactions. Here, we'll understand what are textures, how to apply them to models and how they interact with light. We will also use some interaction to change the scene and the models within, while creating a representation of the earth and moon, their interaction with each other and their interaction in a light that simulates the sun.

3.1 - Using a texture in a plane

For the first exercise we got to use as a base, once again, the rotating cube but now, as a material, we've used a material's map attribute to apply the Lena.jpg image as the plane texture. To read the image we have used given code.
Before all of this, we should have change the cube to a plane geometry.
In Figure 1 are represented 3 different results were the size of the plane was changed, the THREE.PlaneGeometry parameters.



(a) THREE.PlaneGeometry parameters (1, 3)
(b) THREE.PlaneGeometry parameters (3, 1)
(c) THREE.PlaneGeometry parameters (3, 3)

Figure 1: Results of 3.1 - Using a texture in a plane

As it possible to see, in Figure 1a we've used as parameters (1, 3) and the image got stretched vertically; in Figure 1b the parameters were (3, 1) and now, the image got stretched horizontally; finally, using (3, 3) as parameters, in Figure 1c, the image is represented without distortion, since the original image is a square.

NOTE: the angle of the plane is because of an animation that is rotating the plane.

3.2 - Texture on a cube

In this exercise, which is based on the previous, we were asked to change the geometry back to a cube and see how the image was mapped.

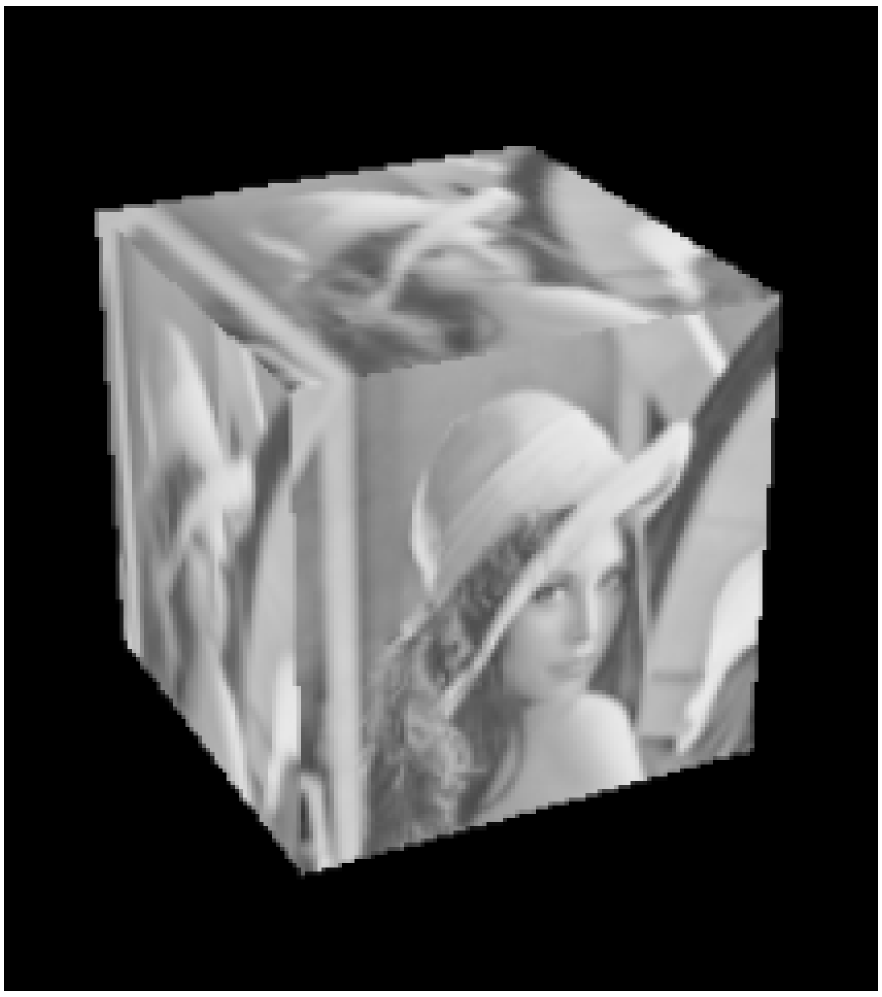


Figure 2: Mapping lena.png to texture of cube

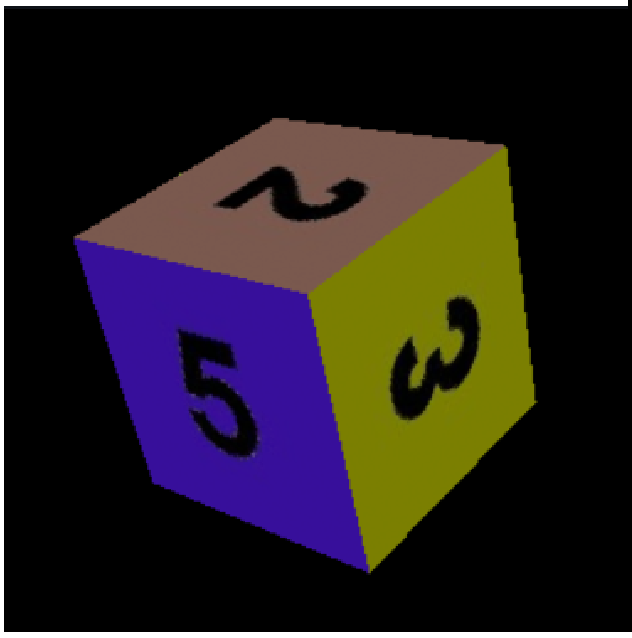
As we can see in Figure 2, the image was mapped to all six faces of the cube.

After, we were asked to map different images for each face of the cube as texture.

To achieve that, it is necessary to aggregate all the textures in a materials variable using the push command:

```
const materials = [];
for (var i = 1; i <= 6; i++){
    materials.push(new THREE.MeshBasicMaterial({ map: new
                        THREE.TextureLoader().load('images/Im'+i+'.jpg') }));
}
```

The results are represented in Figure 3.


(a) Expected result.

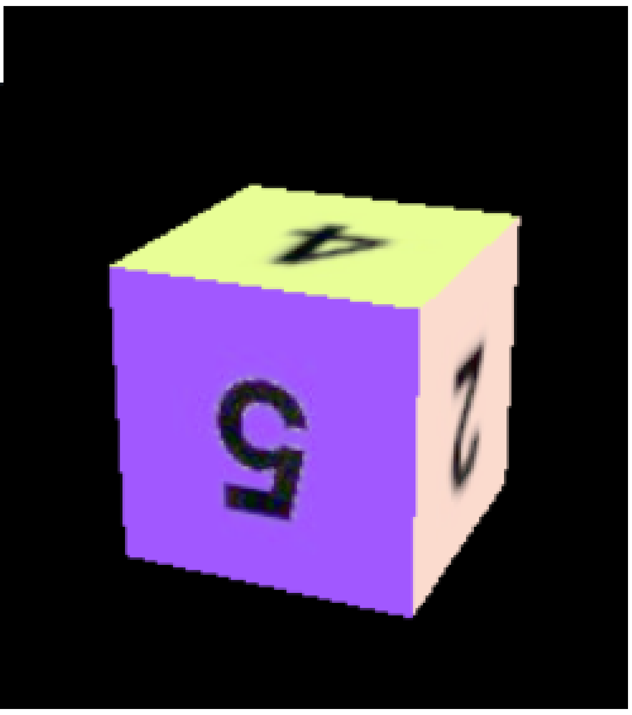

(b) Result gotten.

Figure 3: Results of 3.2 - Texture on a cube

3.3 - Texture and Lighting

Different from the first two exercises, this one asks for a creation of a new program with the goal of visualizing a sphere of radius 1 using 32 segments in width and height.

After the sphere creation, consulting the work done in Lesson 2, the next step was apply the earth_surface_2048.jpg as texture to the sphere. For that, the strategy followed was the same as the Exercise 3.1.

Then, we were told to change the material type to MeshPhongMaterial. This material type requires light so, the next step was adding an ambient light with the value 0x333333 and a directional light with direction (1,0,0) and with the value 0xfffff representing the sun, following the steps used in Lesson 2.

Besides the result obtain in Figure 4, we had to implement an animation. This animation consists on a rotation around the y axis of 0.0025 rad and, before that, a fixed rotation on the Z axis of 0.41 rad.

```
sphere.rotation.y += 0.0025;
sphere.rotation.z = 0.41;
```

Note that the rotation parameter in the Z axis only as an = instead of += because this is a single rotation, that is, is fixed and don't change over time like the Y axis one, that represents the Earth rotation.

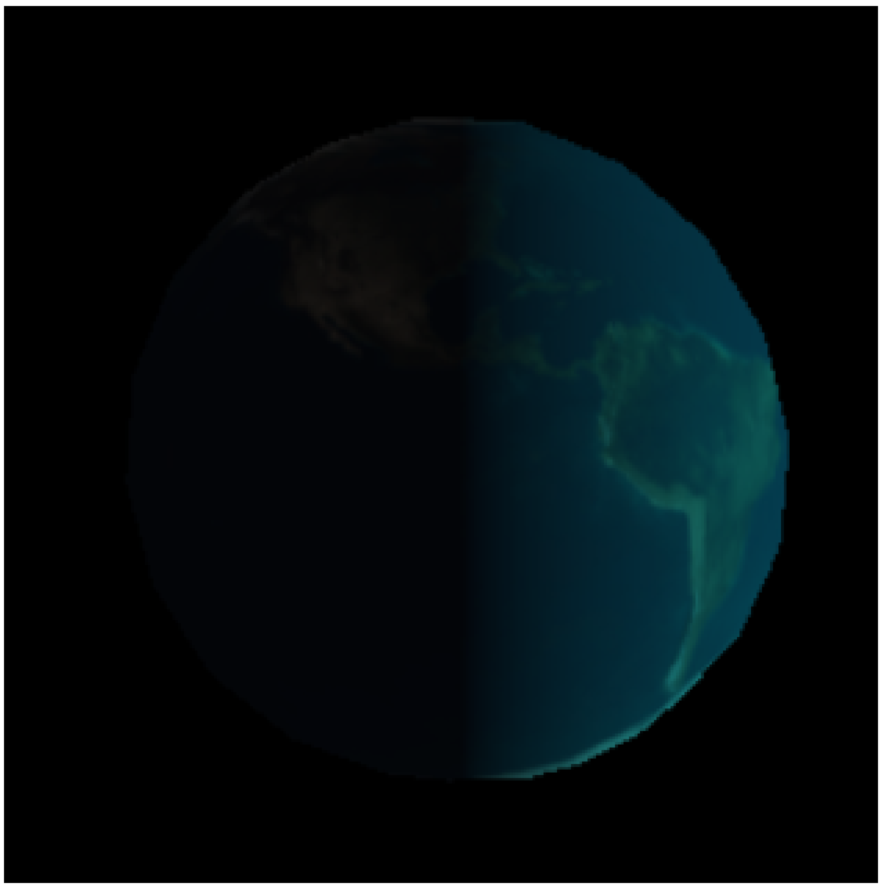


Figure 4: Results of 3.3 - Texture and Lighting

3.4 - Interaction

In this exercise, the only task was to add an event that responded to a keydown (pressing a keyboard key) event, that is, adding the possibility to our program know that we were pressing keys and giving it the possibility to read them/know what key were we pressing.

For that, we just implemented in the code before the following given code:

```
document.addEventListener("keydown", onDocumentKeyDown, false);

function onDocumentKeyDown(event){
    // Get the key code of the pressed key
    var keyCode = event.which;
    console.log("Key  " + keyCode);
}
```

3.5 - Lighting activation

Now, with the possibility to interact with our program by pressing keyboard keys, we were asked to modify the previews code so that there was the possibility to turn on/off the directional light via the L key or the possibility to turn the material of the sphere from MeshPhongMaterial back to MeshBasicMaterial and vice-versa. We chose to make both options possible, with the option of turning the directional light on/off via the L key and the option of turning the material of the sphere to MeshPhongMaterial/MeshBasicMaterial via the K key.

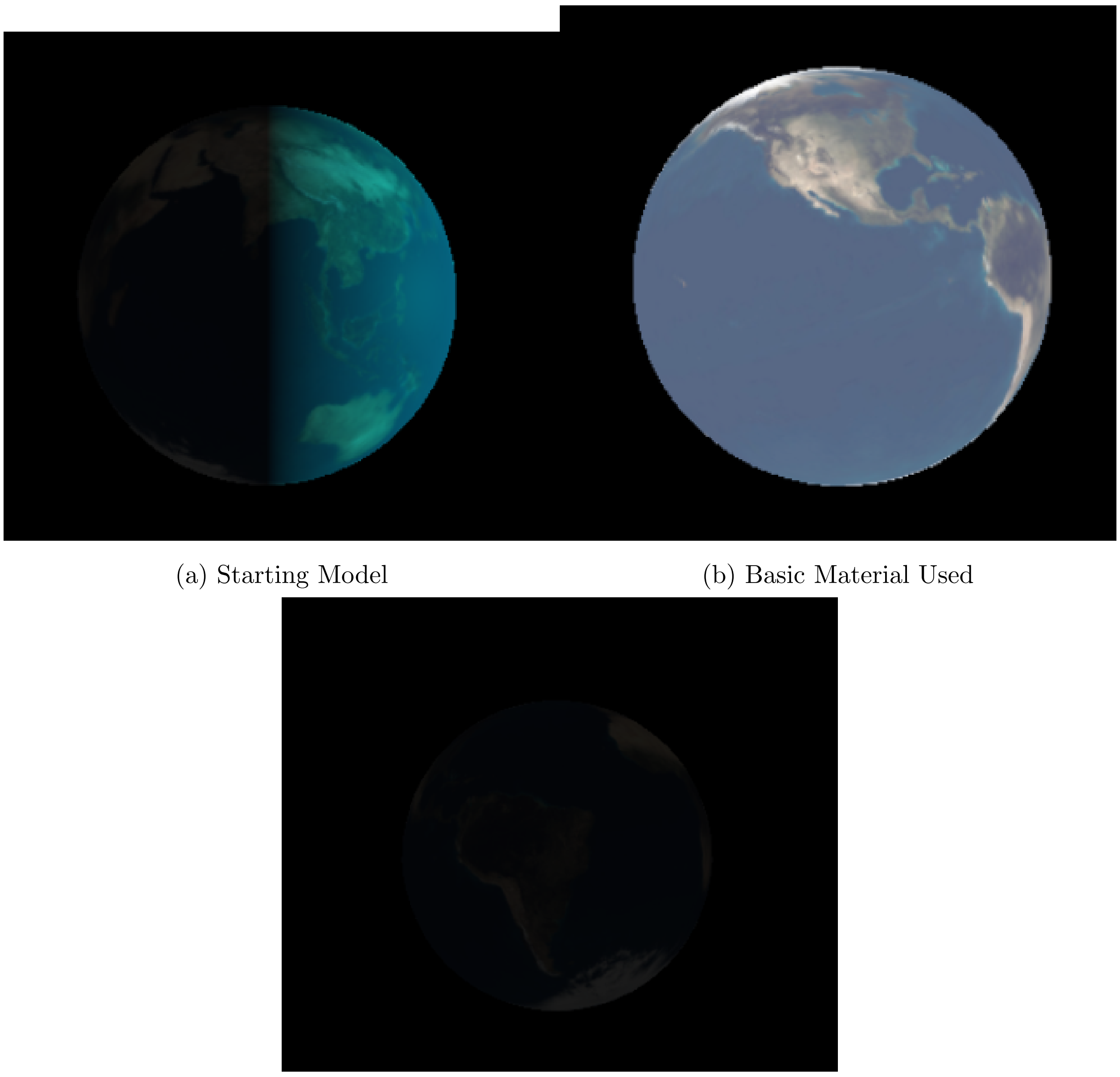
To do this, in the function onDocumentKeyDown, we created a switch case using the keyCode as the expression, and with the following cases:

```
case 76:
    if (light_on){
        scene.remove(directionalLight);
        light_on = false;
        console.log('Directional Light Off');
    }else{
        scene.add(directionalLight);
        light_on = true;
        console.log('Directional Light On');
    }
    break;
// k key
case 75:
    if (phong_mesh){
        sphere.material = basic_material;
        phong_mesh = false;
        console.log('Using Basic Material');
    }else{
        sphere.material = phong_material;
        phong_mesh = true;
        console.log('Using Phong Material');
    }
    break;
```

As we can see in Figure 5b, when using a MeshBasicMaterial, the sphere does not interact with light, so we see the original image used. On Figure 5c, we can see that with the directional light turned off, the sphere is barely visible, and does not become invisible due to the ambient light.

It was also requested the possibility to increase/decrease the light intensity using the + and - keys. For that, the logic was the same:

```
// + key
case 107:
    directionalLight.intensity += 0.1;
    console.log('Increased directional light intensity to ' + directionalLigh
    break;
// - key
case 109:
    if (directionalLight.intensity >= 0){
        directionalLight.intensity -= 0.1;
        console.log('Reduced directional light intensity to ' + directionalLi
    }else{
        console.log('Directional light intensity cannot be reduced further');
```



(a) Starting Model (b) Basic Material Used

(c) Directional Light Off

Figure 5: Different methods used

```
    }
    break;
```

Now, instead of adding/removing something from the scene, we just incremented the intensity parameter of the light. We also checked if the intensity was bigger than 0 before decreasing the intensity, since a negative intensity makes no sense and is no different than a intensity with value 0.

To note that the ASCII code for the 'L' key is 76, for the 'K' key is 75, for the '+' key is 107 and for the '-' key is 109.

3.6 - Modify position and rotation

In the exercise 3.6, the goal was the same as the previews exercise, related to interaction.

This time, when the arrow keys were pressed, left and right, the rotation speed around the yy axis would increase/decrease and, the when up/down keys were pressed, the inclination of the model around the zz axis would change.

Once again, we added new cases on the function of Exercise 3.4:

```
// left arrow key
case 37:
    rotation_speed -= 0.0005;
    console.log('Increased rotation speed to ' + rotation_speed.toFixed(4));
    break;
// right arrow key
case 39:
    rotation_speed += 0.0005;
    console.log('Increased rotation speed to ' + rotation_speed.toFixed(4));
    break;
// up arrow key
case 38:
    sphere.rotation.z += 0.01;
    console.log('Increased sphere inclination to ' + sphere.rotation.z.toFixe
    break;
// down arrow key
case 40:
    sphere.rotation.z -= 0.01;
    console.log('Increased sphere inclination to ' + sphere.rotation.z.toFixe
    break;
```

The code was also changed to use the rotation_speed variable in the animation function.

So, when the user pressed the arrow keys, he could interact with rotation parameters of the sphere.

3.7 - Concatenation of transformations / addition of the Moon

For the finally exercise, we had to add a new model, the Moon, with the same logic of the Earth, adding a new sphere with a specifc texture, the moon_1024.jpg. Its parameters are the following:

```
DISTANCE_FROM_EARTH = 356400;
PERIOD = 28;
INCLINATION = 0.089;
SIZE_IN_EARTHS = 1 / 3.7;
EARTH_RADIUS = 6371;
```

As it occurs in real life, the Moon rotates around the Earth and around itself. To get the rotation around the Earth we need to create the Moon as a child of the Earth so that the moon gets influenced by the Earth's transformations. This is an automatic process because the transformation matrices of the two objects are multiplied.

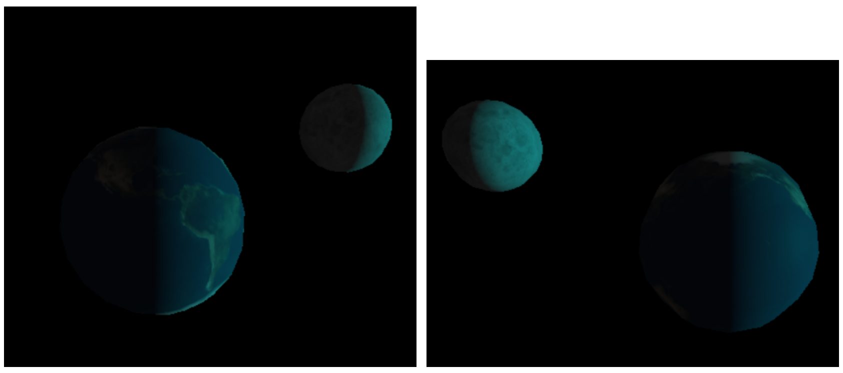
To initialize the Moon in its correct position we added the following code lines:

```
var distance = DISTANCE_FROM_EARTH / EARTH_RADIUS;
moon.position.set(Math.sqrt(distance / 2), 0,-Math.sqrt(distance / 2));
```

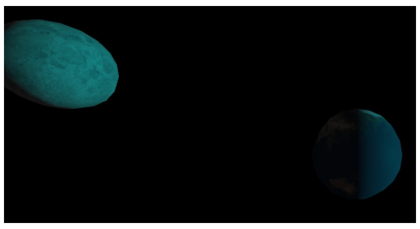
Then, its animation is obtained by the following:

```
// Rotate the moon so it shows its moon-face toward earth
moon.rotation.y = Math.PI;
moon.rotation.x = INCLINATION;
// For animation
moon.rotation.y += (earth.rotation.y / PERIOD);
```

The result obtained are illustrated in three different angles, in Figure 6, so it is possible to have a representation of the Moon's movement, its rotation, around the Earth.



(a) One angle. (b) Different angle.



(c) Another angle.

Figure 6: Results of 3.7 - Concatenation of transformations / addition of the Moon

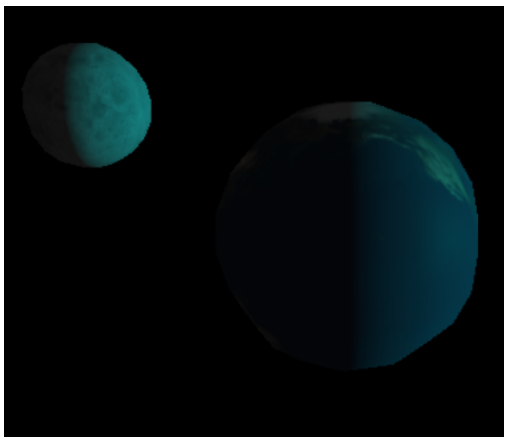
As an addiction to the exercise, we added a new interaction that allow the user change the texture applied on the sphere object that represents the Earth.

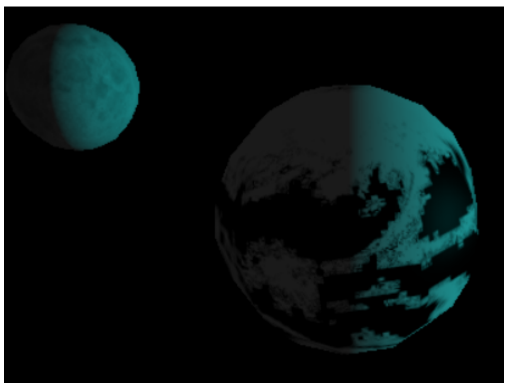
For that, it was necessary to add a new condition to the that when 'C' key is pressed, a function is called and the texture change is performed:

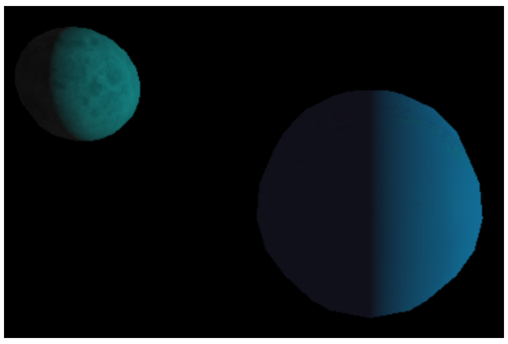
```
var texloader = new THREE.TextureLoader();
var earth_tex = texloader.load("../images/earth_surface_2048.jpg");
const earth_textures = [
    "../images/earth_surface_2048.jpg",
    "../images/earth_clouds_1024.png",
    "../images/earth_normal_2048.jpg",
    "../images/earth_specular_2048.jpg",
    "../images/earth_atmos_2048.jpg",
  ];
```

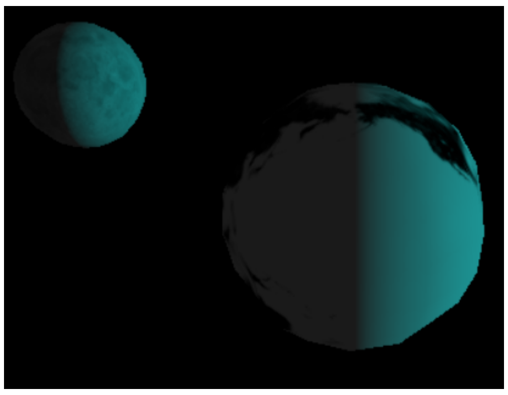
```
...

function onDocumentKeyDown(event){

    ...

    case 67:
    // Increment the texture index (loop back to the start if needed)
    current_texture_index = (current_texture_index + 1) % earth_textures.leng

    const next_texture_path = earth_textures[current_texture_index];
    const next_texture = texloader.load(next_texture_path);
    console.log("Earth Texture: " + next_texture_path);

    // Update both materials map with the new texture
    basic_material.map = next_texture;
    phong_material.map = next_texture;

    basic_material.needsUpdate = true;
    phong_material.needsUpdate = true;

    break;

}
```

The first step is increment the variable that is the image index and, if this value is equal to the number of images available minus 1 (because index starts in 0), the operation returns a 0 and the variable is reset, that is, it is attributed the 0 value. After, we get the image path from the earth_textures list using an variable as index. Then, we set both materials map as the texture, and we update the materials.
All texture can be checked in Figure 7.


(a) 1st Texture: earth_atmos_2048.


(b) 2nd Texture: earth_clouds_1024.


(c) 3rd Texture: earth_normal_2048.


(d) 4th Texture: earth_specular_2048.

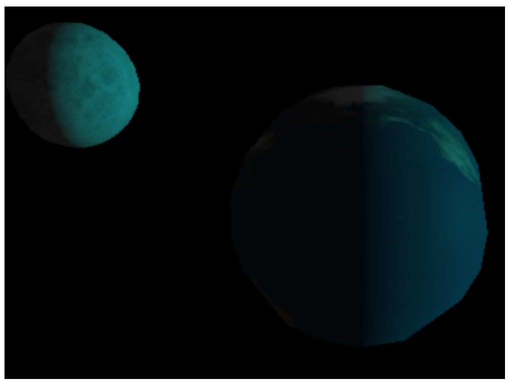

(e) 5th Texture: earth_surface_2048.

Figure 7: Results of 3.7 - Extra Work.

Conclusion

To sum up the Lesson 3 - Texture and Interaction, covering textures, lighting, and interactive elements, the exercises explored applying textures to different geometries, incorporating lighting and introducing user interaction through keyboard input. Overall, the lesson provided a hands-on experience in creating visually compelling and interactive 3D scenes by combining textures, lighting, and user-driven transformations.

Finally, it is possible to say that the exercises were completed with success as the results were as expected.