Lesson 3: Texture and Interaction

Beatriz Sofia Mesquita Gonçalves, 115367

December 2022

Student	Beatriz Sofia Mesquita Gonçalves, 115367
Professors	Paulo Dias and Beatriz Sousa Santos

Introduction

This assignment was developed in the context of the Information Visualization course of the Data Science Master's program at the University of Aveiro. To run the code, create a local server, e.g. live server and run in localhost the pages.
This third delivery consists of a brief texture and interaction in *three.js*. The **outline** of this assignment is:

- Texture;
- Combining texture and Illumination;
- Keyboard Interaction.

1 Using a texture in a plane

We start by modifying the rotating cube example [1] to show a plane (PlaneGeometry). We used the Material's map attribute to apply the Lena.jpg image as the plane texture.

To read the image, we use:

```
// Load the texture
    const texloader = new THREE.TextureLoader();
    const tex = texloader.load("./images/earth_surface_2048.jpg");
```

Listing 1: Read the image

We used a *MeshBasicMaterial*, which is a material that applies the texture directly to the geometry without any lighting calculations. We get the following output:

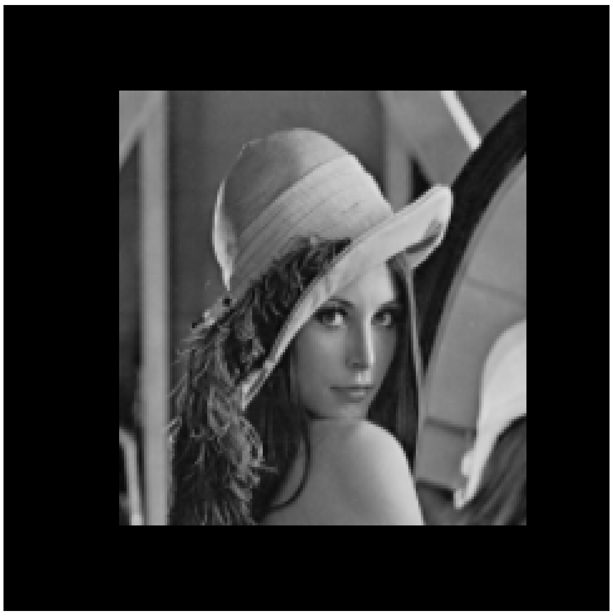

Figure 1: Texture in a plane.

FILE: using*atextureinaplane.html*

Then, we try to modify the size of the plane. If we modify the size of the plane, the texture will also be scaled to fit the new size of the plane. For example, if we increase the size of the plane, the texture will appear smaller relative to the plane, and if we decrease the size of the plane, the texture will appear larger relative to the plane.

To solve this, we can set the *wrapS* and *wrapT* properties of a texture to *THREE.ClampToEdgeWrapping*:

```
// Set the texture's wrapS and wrapT properties to ClampToEdgeWrapping
      tex.wrapS = THREE.ClampToEdgeWrapping;
      tex.wrapT = THREE.ClampToEdgeWrapping;
```

Listing 2: wrapS and wrapT properties

The *wrapS* and *wrapT* properties of a texture determine how the texture is repeated when it is applied to a surface that is larger than the texture itself. By default, the *wrapS* and *wrapT* properties are set to *THREE.RepeatWrapping*, which means that the texture is repeated in both the horizontal (*wrapS*) and vertical (*wrapT*) directions to cover the surface.

Setting the *wrapS* and *wrapT* properties to *THREE.ClampToEdgeWrapping* changes this behavior. With *THREE.ClampToEdgeWrapping*, the texture is not repeated and instead the edge pixels of the texture are used to fill in the areas that extend beyond the texture's original dimensions. [2] [5] [4]

And we get a figure of different size: (of course, the difference is best seen when you run both codes)

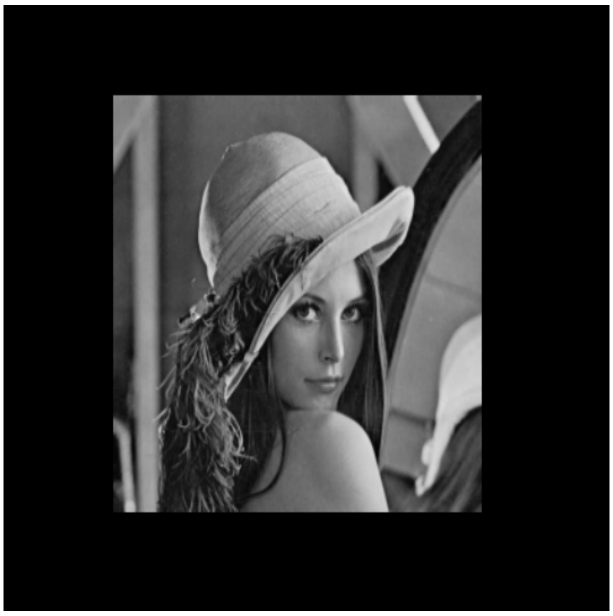

Figure 2: Modify size of the plane

FILE: using*atextureinaplanechangedimensions.html*

2 Texture on a cube

We then use the lena.jpg image as the texture for the cube. Here's our output for a rotating cube:

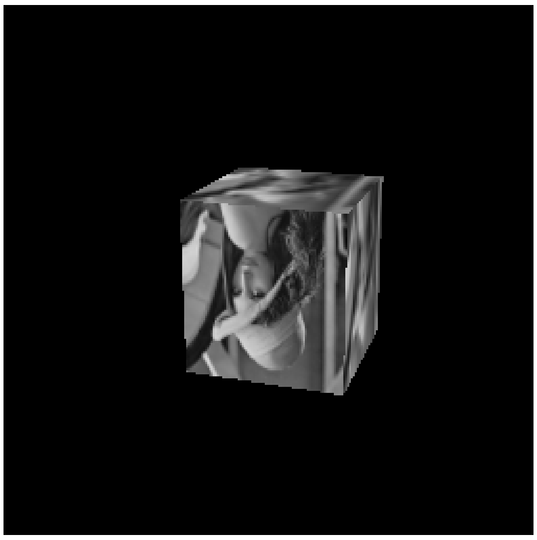

Figure 3: lena.png as the texture for the rotating cube

FILE: textureonacube.html

By default, the texture will be mapped to the cube in a way that the texture's x-axis (horizontal direction) is aligned with the cube's x-axis, and the texture's y-axis (vertical direction) is aligned with the cube's y-axis. This means that the top face of the cube will have the top part of the texture, the bottom face will have the bottom part of the texture, the front face will have the left part of the texture, and the back face will have the right part of the texture.

We can modify this default behavior by setting the *wrapS* and *wrapT* properties of the texture that we talked before, or by using a different material that allows you to specify the UV coordinates for the texture mapping. [2] [5] [3]

We modify the program to map a different image to each face of the cube (use the images Im1.jpg, Im2.jpg... Im6.jpg). To use several texture, we need to aggregate all the textures in a materials variable (var materials = []) using the push command.

Basically, to map a different image to each face of the cube, we can create a *MeshBasicMaterial* for each face and specify the texture for each material. Then, we can create a Mesh object using a *BoxGeometry* and an array of materials, with one material for each face of the cube. This way we get, for the rotating cube:

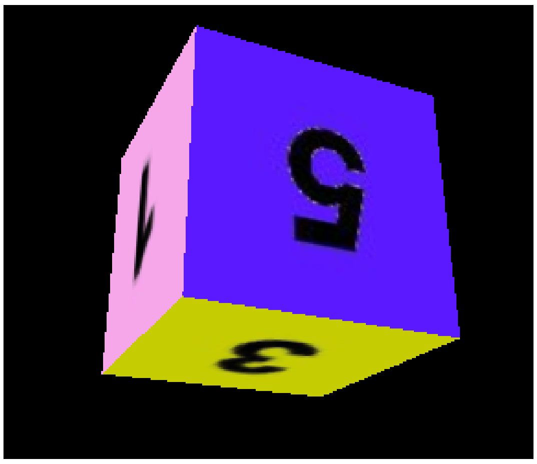

Figure 4: Cube with 6 different images.

FILE: textureonacube*multipleimages.html*

It asked to use *OrbitControls* to control the position and orientation of the cube, like in the 2nd assignment. However, this work was done at the same time as the previous one and the problem persists. The page to import *OrbitControls* does not exist, so we cannot get an output. However, the code for this was done anyway, in the FILE: textureonacube*multipleimagesorbitcontrol.html*.

3 Texture and Lighting

In this section, we are asked to create a program to visualize a sphere of radius 1 (use 32 segments in width and height). Apply the planisphere texture (earth*surface*2048.*jpg*)*tothesphere*.

We view the model with a fixed rotation on the Z axis (0.41 rad) and with an animation (a rotation around the y axis of 0.0025 rad). We then add lighting to the scene, use a *MeshPhongMaterial* type material with the texture (instead of a *MeshBasicMaterial*) and add an ambient light with the value 0x333333 and a directional light with direction (1,0,0) and with the value 0xfffff representing the sun.

We get as a result:

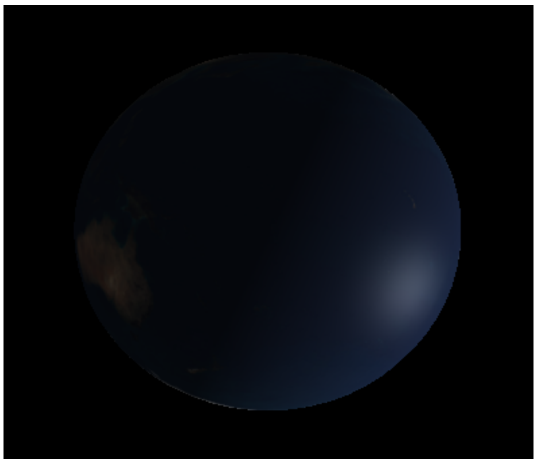

Figure 5: Planet Earth on rotation.

FILE: textureandlighting.html

4 Interaction

We added the following code to respond to the keydown event:

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

Listing 3: Keydown event

And use the following function to see which key was pressed on the console:

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

Listing 4: Keydown event function

FILE: Interaction.html

5 Lighting activation

We modify the code to allow turning on/off the directional light via the **L** key. We also add the possibility to increase/decrease the light intensity using the + and **-** keys. We do this by two methods:

1) Removing the light from the scene

```
// Add event listener for keydown events
      document.addEventListener("keydown", onDocumentKeyDown, false);

      function onDocumentKeyDown(event) {
            // Get the key code of the pressed key
            var keyCode = event.which;

            // Toggle the visibility of the directional light if the L key was pressed
            if (keyCode === 76) {
                if (directionalLight.parent) {
                    scene.remove(directionalLight);
                } else {
                    scene.add(directionalLight);
                }
            }

            // Check for the + key
            if (keyCode == 107) {
                // Increase the light intensity by 0.1
                directionalLight.intensity += 0.1;
            }
            // Check for the - key
            else if (keyCode == 109) {
                // Decrease the light intensity by 0.1
                directionalLight.intensity -= 0.1;
            }
        }
```

Listing 5: Method: Removing Light from the scene

FILE: lighting$activationremovingand + -intensity.html$(**check here the interactions with the output**).
2) Changing the material to a *MeshBasicMaterial*

```
// Add event listener for keydown events
            document.addEventListener("keydown", onDocumentKeyDown, false);

            function onDocumentKeyDown(event) {
            // Get the key code of the pressed key
            var keyCode = event.which;

            // Toggle the visibility of the directional light if the L key was pressed
            if (keyCode === 76) {
              directionalLight.visible = !directionalLight.visible;
            }
          }
```

Listing 6: Method: Changing the material to a MeshBasicMaterial

FILE: Lighting$activationmodifying.html$

6 Modify position and rotation

We use the arrow keys [left and right] to increase/decrease the rotation speed around the yy axis axes and the Up/Down keys to increase/decrease the inclination of the model around the zz axis, using:

```
 // Add event listener for keydown events
      document.addEventListener("keydown", onDocumentKeyDown, false);

      function onDocumentKeyDown(event) {
        // Get the key code of the pressed key
        var keyCode = event.keyCode || event.code;

        // Check for the L key
        if (keyCode == 76 || keyCode == "KeyL") {
          // Toggle the directional light on/off
          if (directionalLight.intensity > 0) {
            directionalLight.intensity = 0;
          } else {
            directionalLight.intensity = 1;
          }
        }
        // Check for the + key
        else if (keyCode == 187 || keyCode == "Equal") {
          // Increase the intensity of the directional light
          directionalLight.intensity += 0.1;
        }
        // Check for the - key
        else if (keyCode == 189 || keyCode == "Minus") {
          // Decrease the intensity of the directional light
          directionalLight.intensity -= 0.1;
        }
        // Check for the left arrow key
        else if (keyCode == 37 || keyCode == "ArrowLeft") {
          // Decrease the rotation speed around the y axis
          rotationSpeed -= 0.0005;
        }
        // Check for the right arrow key
        else if (keyCode == 39 || keyCode == "ArrowRight") {
          // Increase the rotation speed around the y axis
          rotationSpeed += 0.0005;
        }
        // Check for the Up arrow key
        else if (keyCode == 38 || keyCode == "ArrowUp") {
          // Increase the inclination of the model around the z axis
          inclination += 0.0005;
        }
        // Check for the Down arrow key
        else if (keyCode == 40 || keyCode == "ArrowDown") {
          // Decrease the inclination of the model around the z axis
          inclination -= 0.0005;
        }
```

Listing 7: Modify position and rotation

FILE: Modify position and rotation.html (**check here the interactions with the output**).
Here's a summary of what the code does:

- It adds an event listener for keydown events on the document;
- It defines the *onDocumentKeyDown* function to handle the keydown events;
- In the *onDocumentKeyDown* function, it checks the key code of the pressed key and performs the appropriate action based on the key that was pressed.

7 Concatenation of transformations / addition of the moon

We add a new model to represent the moon using the texture moon1024.*jpg*, *consideringthegivenconstants* :

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

Listing 8: Given constants

To allow the moon to rotate around the earth, we have to create the moon as a child of the earth to be influenced by the earth's transformations (transformations applied to earth are automatically applied to the moon, multiplying the transformation matrices of the two objects) . For this, the moon model must be added (add) to the earth model:

```
 // Load the textures
      const texloader = new THREE.TextureLoader();
      const texEarth = texloader.load("./images/earth_surface_2048.jpg");
      const texMoon = texloader.load("./images/moon_1024.jpg");

      // Create a Phong material for the earth with the earth texture
      const materialEarth = new THREE.MeshPhongMaterial({map: texEarth});

      // Create a sphere geometry for the earth
      const geometryEarth = new THREE.SphereGeometry(1, 32, 32);

      // Create a sphere mesh using the geometry and material for the earth
      const earth = new THREE.Mesh(geometryEarth, materialEarth);

      // Add the earth to the scene
      scene.add(earth);

      // Create a Phong material for the moon with the moon texture
      const materialMoon = new THREE.MeshPhongMaterial({map: texMoon});

      // Create a sphere geometry for the moon
      const geometryMoon = new THREE.SphereGeometry(SIZE_IN_EARTHS, 32, 32);

      // Create a sphere mesh using the geometry and material for the moon
      const moon = new THREE.Mesh(geometryMoon, materialMoon);

      // Add the moon to the earth as a child object
      earth.add(moon);
```

Listing 9: Adding the moon

The considered transformations to initialize the moon in the correct position and apply the correct animation were:

```
// Set the initial position of the moon relative to the earth
    var distance = DISTANCE_FROM_EARTH / EARTH_RADIUS;
    moon.position.set(Math.sqrt(distance / 2), 0,-Math.sqrt(distance / 2));

    // Rotate the moon so it shows its moon-face toward earth
    moon.rotation.y = Math.PI;
    moon.rotation.x = INCLINATION;

    function animate() {
      requestAnimationFrame(animate);

      // Animate the earth
      earth.rotation.y += rotationSpeed;

      // Animate the moon
      moon.rotation.y += (earth.rotation.y / PERIOD);

      // Render the scene
      renderer.render(scene, camera);
    }

    animate();
```

Listing 10: Transformations for Moon's position

So here's a screenshot of what is happening:

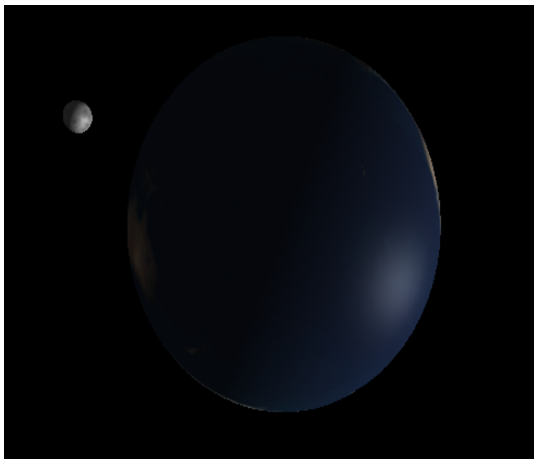

Figure 6: Planet Earth and Moon on rotation.

FILE: Concatenation of transformations addition of the moon.html (**check here the interactions with the output**).

To summarize what we have done in this last section and simplify interpretation: This code displays a 3D scene with an earth and a moon orbiting around it. The earth should be rotating around its own axis and the moon should be orbiting around the earth. The camera should be positioned at a fixed distance from the earth and should have a fixed inclination around the z-axis.

Here are all the steps done:

- It creates a scene, a camera, and a renderer;
- It loads the textures for the earth and moon using the TextureLoader class;
- It creates a Phong material for the earth and a sphere geometry for the earth, and uses these to create a sphere mesh for the earth. It adds the earth to the scene;
- It creates a Phong material for the moon and a sphere geometry for the moon, and uses these to create a sphere mesh for the moon. It adds the moon to the earth as a child object;
- It sets the initial position and orientation of the camera;
- It adds ambient and directional lighting to the scene;

- It sets the initial rotation speed around the y-axis and the inclination around the z-axis;
- It sets the initial position of the moon relative to the earth;
- It rotates the moon so it shows its moon-face toward earth;
- It defines the animate function to update the rotation of the earth and moon, and render the scene;
- It starts the animation loop by calling the animate function;
- It adds an event listener to update the renderer size and camera aspect ratio when the window is resized;
- It adds an event listener to listen for keydown events and respond to the L, +, and - keys to toggle the directional light on and off, and increase or decrease its intensity;
- It adds an event listener to listen for keydown events and respond to the arrow keys to increase or decrease the rotation speed around the y-axis and the inclination around the z-axis.

[8] [6] [7]

References

[1] Rotating cube example. `https://threejs.org/docs/index.html#manual/introduction/Creating-a-scene`. Accessed: 2022-12-23.

[2] Three.js documentation on creating materials. `https://threejs.org/docs/index.html#api/en/materials/Material`. Accessed: 2022-12-23.

[3] Three.js documentation on texture mapping. `https://threejs.org/docs/index.html#api/en/materials/MeshBasicMaterial.map`. Accessed: 2022-12-23.

[4] Three.js documentation on texture wrapping. `https://threejs.org/docs/index.html#api/en/textures/Texture.wrapS`. Accessed: 2022-12-23.

[5] Three.js documentation on textures. `https://threejs.org/docs/index.html#api/en/textures/Texture`. Accessed: 2022-12-23.

[6] three.js examples. `https://threejs.org/examples/`. Accessed: 2022-12-14.

[7] three.js learn. `https://threejs.org/`. Accessed: 2022-12-14.

[8] uainfovis/three.js/lesson01. `https://github.com/pmdjdias/ua_infovis/tree/master/Three.js/Lesson_01`. Accessed: 2022-12-14.