UNIVERSITY OF NEBRASKA AT KEARNEY

CSIT 422 – Computer Graphics

Tutorial Handout – Three.js

Student:

Igor Felipe Ferreira Ceridório

November 5, 2015

1. Introduction

Three.js is a JavaScript library/API that has the ability to create and display animated 3D computer graphics on a web browser. Three.js uses WebGL, and the idea is to make WebGL easy to use.

The purpose of this document is to guide you through a simple exercise using Three.js, this exercise will teach the basic concepts of structures and rendering principles of this library. So, let's get started!

2. Simple organization of the files

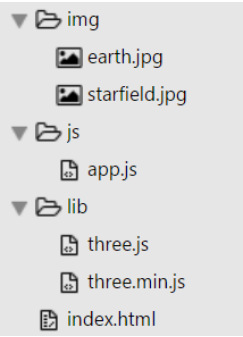
If you already developed anything with HTML and JavaScript you will feel really comfortable with Three.js environment. If you didn't (just like me ☺) don't worry, I think we can get through this.

In order to have more time focusing on the programming, I provided the basic structure on a **.zip** file so we can start with coding right away.

Our little project will be linked to an **index.html**. You don't need to change anything on that file, it basically contains the simple definition of a web page, and also the script reference to the Three.js library we will use, and the call to the JavaScript Start() function that we are going to program together in a few moments.

This **.zip** file also contains three folders inside it: **lib** (contains the **three.min.js,** our Three.js library, this is the same file that is in the full Three.js package I asked you to download), **js** (contains the **app.js** file, which is where we are going to code our tutorial) and **img** (contains two images that will be used in the assignment and extra exercise).

Unzip these files in a folder of you preference, your structure should look like this:

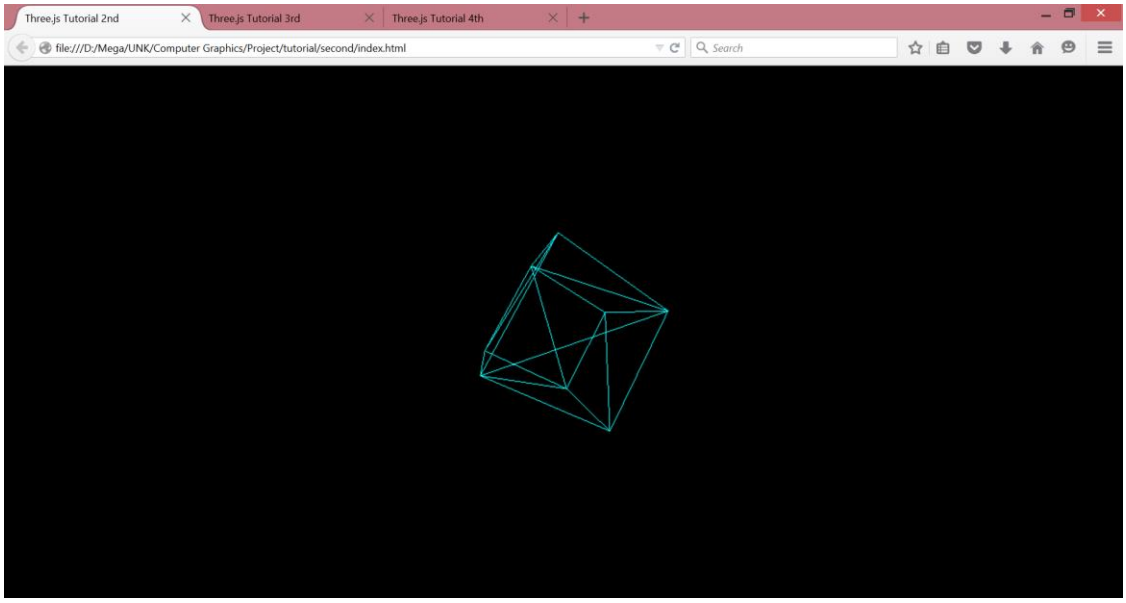


Once you did this, we are ready to start with coding!

3. Rotating cube

We will code a spinning cube centralized in the center of our web browser. More than that, we will add movement to our cube. In the end of this exercise, we will have a nice wire framed cube rotating in the center of our screens, just using Three.js and our web browsers! I kindly ask you if possible to test your code on Mozilla Firefox or Internet Explorer. Google Chrome has some security issues that take some time to solve when dealing with texture (texture that we will use in the assignment later).

In the end of the exercise, you should obtain a result like this:



Let's start!

3.1 Creating the scene, the camera and the renderer

The first thing to do in our project is to set up the scene, the camera and the renderer we will use. The scene place everything that we want to display with the renderer. It contains objects, lights and the cameras. In your file **app.js**, under `function Start(){` type:

```
// Creating the scene
var scene = new THREE.Scene(), WIDTH = window.innerWidth, HEIGHT =
window.innerHeight;

// Creating the camera in perspective
var camera = new THREE.PerspectiveCamera(100, WIDTH / HEIGHT, 0.1, 1000);

// Creating the rendering WebGL object
var renderer = new THREE.WebGLRenderer();
```

There are two main types of renderers available in Three.js library. **CanvasRenderer** (older renderer with slower performance) and **WebGLRenderer** (newest renderer with performance improved). We will use **WebGLRenderer** here.

The next step right now is to configure the renderer dimensions. This can be done with:

```
// Configuring rendering dimensions
renderer.setSize(WIDTH, HEIGHT);
```

Now we need to “attach” our recently created renderer to our web page. We will link the renderer to the **body** element of our **index.html** file.

```
// Inserting the renderer to the DOM
document.getElementsByTagName('body')[0].appendChild(renderer.domElement);
```

Our scene, camera and renderer are now ready to be used.

3.2 Creating the 3D object and adding it to the scene

Our next step is to create our cube, and add it to our scene. This is done in three simple steps: we create the object, the material of what it is going to be made of, and then we mesh the object with its material. After that, we just add the object to the scene.

```
// Creating the cube
var geometry = new THREE.BoxGeometry(10, 10, 10);

// Creating the material of the object, defining its exhibition of wireframe
var material = new THREE.MeshBasicMaterial({color: 0xfffff, wireframe: true});

// Adding the created material to the created object
var cube = new THREE.Mesh(geometry, material);

// Adding the object to the scene
scene.add(cube);
```

3.3 Defining the camera position and what scene and camera will be used

We are almost done with our project. We just need to define a couple things left. Since Three.js renderer is executed just in the end of the file, unfortunately we cannot see our project running yet.

We will now define the camera position and set up which scene and camera our renderer will use, since one project can have multiple scenes and cameras inside it.

```
// Defining the position of the camera
camera.position.z = 20;

// Defining the scene and the camera that will be used
renderer.render(scene, camera);
```

3.4 Creating the rendering function

The last step is to code our render function. It will be responsible for rotating the cube and for refreshing the rendering. It is as simple as that:

```
// Function to control the scene renderization
function render(){
	// Defining the scene and the camera that will be used
	renderer.render(scene, camera);

	// Altering axis values to execute some movement
	cube.rotation.z += 0.01;
	cube.rotation.y += 0.01;
	cube.rotation.x += 0.01;

	// Executing recursively
	requestAnimationFrame(render);
}
```

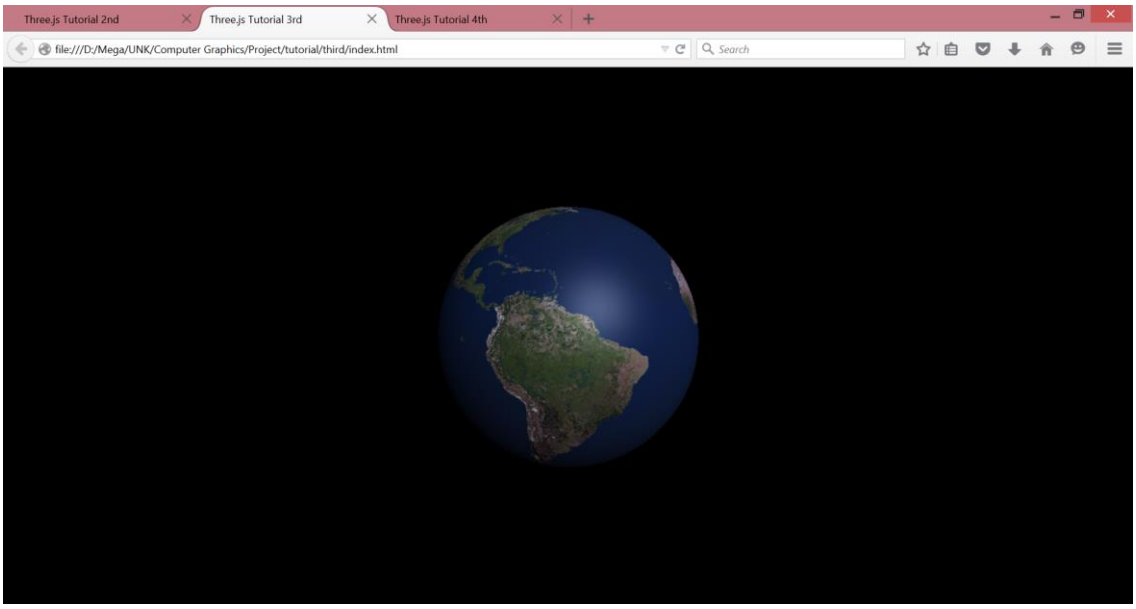
We cannot forget to call our render function:

```
// Executing the rendering function
render();
```

Now just open **index.html** with your web browser and you should see your spinning wireframed cube in the center of the screen.

4. Planet Earth!

Now that you understood the principles of Three.js let's do another project. This time we do not want to draw a cube in the center of the screen, we want to create the planet Earth! At the end of this exercise, you should be able to accomplish this:



You can modify the code from the last exercise, or create a new project with the same structure, it is up to you. Few things will change related to the previous exercise.

First, you may notice that there is a light focusing over the planet Earth. You should create that light right after attaching the rendering to the **body** of **index.html.**

To create a light do the following:

```
// Creating a light source that will allow MeshPhongMaterial to work
var pointLight = new THREE.PointLight(0xffffff);
pointLight.position.set(100, 100, 250);
scene.add(pointLight);
```

Remember that now you do not have a **BoxGeometry** (cube) anymore, you have a **SphereGeometry** object, and it has three parameters, radius, widthSegments, and heigthSegments.

After creating your new geometry form, you must create the material for it, just like in the first exercise, but this time, our material will be composed by the image of the planet Earth.

To create the material using the image provided in **img** folder:

```
// Creating the material of the object, defining its exhibition of wireframe
var material = new THREE.MeshPhongMaterial();
material.map = THREE.ImageUtils.loadTexture('img/earth.jpg');
```

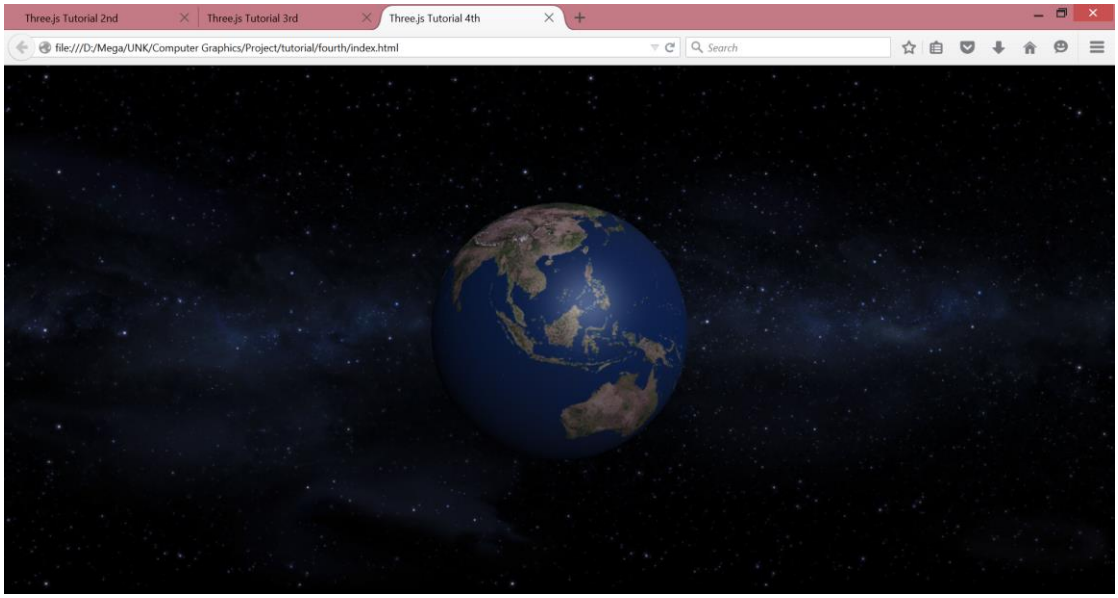
After creating the material just mesh your geometry and material like in the previous exercise.

You are almost done, just do not forget that planet Earth does not rotate in three directions, just one of them would be fine!

5. Adding some stars to the galaxy!

If you already completed the two previous exercises try to add some stars to our scene.

You can do that by applying a background to the scene, there is a background available at **img** folder, but you can put any other background that you want. You should accomplish something like this:



Some hints to accomplish that:

- Think about creating a new texture and new Mesh object to use as background.
- You may need to add another scene and camera as well, think that one scene and camera will handle the background, and the other scene and camera will handle the planet Earth. As stated before, you can have as many scenes and cameras as you want.
- In the render function you must now render both the background and the planet Earth. You should add these lines as well:

```
// Renderer should not automatically clear its output before rendering
renderer.autoClear = false;
renderer.clear();
```

That's it! I hope you liked it. Thank you very much!

Igor Felipe Ferreira Ceridório.