

# A Journey in the Solar System

Alessandro Scire', Fabrizio Tropeano, Federico Arcangeli

September 24, 2018

# 1 Introduction

The project **A Journey in the Solar System** is a 3D representation of the Solar System focused on user interaction.

The main model is characterized by the presence of the planets, displayed in a realistic way taking into account their morphology and their motion along the orbits around the sun.

The roto-traslation motion and the planets dimensions are quantified proportionally with respect to the real ones (except, obviously for the dimension of the sun).

Furthermore, the scene comprehends a space ship, namely the ‘**Death Star**’, with which the user can explore the system with a first person perspective, and shoot around with its moving gun.

## 2 Environment

The whole project is realized with the **Three.js**,<sup>1</sup> that is a cross-browser JavaScript library used to create and display animated 3D computer graphics in a web browser<sup>2</sup>. We chose it because of its suitability for the purposes of our project, thanks to the available high level APIs for the construction and the animation of the objects in the scene, the richer presence of standard materials and the more manageable inclusion of either predefined or custom textures with respect to the WebGL library.

## 3 Libraries, Tools and Models

As support of the development step we included the following external libraries:

- **OBJLoader**, for loading the external 3D rendered model of the ‘DeathStar’
- **MTLLoader**, for loading the external materials of the ‘DeathStar’ object
- **THREEx**, including the listeners of the keyboard states, useful for managing the user controls

---

<sup>1</sup><https://threejs.org/>

<sup>2</sup><https://en.wikipedia.org/wiki/Three.js>

- **OrbitControls**, the API for making the whole scene user-interactive by means of rotation, scrolling and zooming actions.
- **DAT.GUI**, for the visualization of the user commands in a GUI.

The **Death Star** object is a 3D rendered model downloaded from an external source<sup>3</sup>. The definitions of the model and materials are included in the *death-star.obj* and *death-star.mtl* files respectively.

## 4 Technical Aspects

We used two different perspective cameras, we will call them *primary* and *secondary*. The primary camera is the one you see when you load the HTML page for the first time, it has a Field Of View (FOV) of 60 degrees, an aspect ratio proportional to the dimension of the browser's window, the near is set to 0.1 and the far plane to 3000. We used these parameters since we had a large setting to represent, and in this way we could capture all the planets and their orbits. The secondary camera uses the same parameters, but it is placed on top of the spaceship, in order to follow every movement.

### 4.1 Hierarchical Model

The hierarchical model of the Death Star is built in the following way: we first created the arm that comes out from the ship that is the parent; it is realized as a simple parallelepiped. Then we attached the arm's gun to the end of it, as a child. We wanted to use a particular shape for the gun. To do so we used a lathed object, which is a 3D model whose vertex geometry is produced by rotating a set of points around an axis. The name is derived from the real world lathe, since it is used to craft objects in the same way.

The Death Star is a 3D object created in Blender. We used **ObjLoader** to import the render inside the scene and then we used **MtlLoader** to apply the materials to the object.

### 4.2 Lights and Textures

Since we are representing a semi-realistic version of the Solar System, the light obviously comes from the Sun. Our star is created as a sphere with a **MeshBasicMaterial**

---

<sup>3</sup><https://clara.io/view/e5fb057b-8c6c-487e-8000-2c278c5cf4f0>

which is not affected by lights, since it should emit light itself. We placed a `PointLight` inside it, which gets emitted from a single point in all directions.

The planets are created as spheres, and for them we used a `MeshLambertMaterial` in order to replicate a non-shiny surface.

The Lambertian model is often used for diffuse reflection. This technique causes the triangles of a 3D mesh to reflect light equally in all directions when rendered. This effect gives to the planets the most realistic aspect when they're hit by the sunlight. The shading is calculated using a Gouraud shading model, which gives great performances at the expense of the graphical accuracy. In this case we can tolerate a graphical loss since we are going to draw many bodies on the scene and we want to keep a nice and smooth rendering.

We applied to every planet a unique texture, in order to make them recognizable by the user and appealing to watch while rotating around the Sun.

For what concerns the spaceship we decided to use a basic material, so the user can observe the textures and the animations of the object in every point of the scene.

### 4.3 Animation

The first animation that is presented to the user is the motion of the planets around the Sun. Each planet is a sphere with a certain radius, an orbit radius and a different revolution speed.

At every frame we increment a small delta in order to change the position of each planet, and then we compute the next position in the orbit by the means of cosine and sine functions. We simplified the orbits from ellipsis to circles, still keeping the scaled distances between the planets.

The Death Star has different animations. The first is when the arm comes out from the ship: after pressing a button the arm is translated on the X-axis until it reaches the predefined position. At this point the gun starts moving. The gun swings on the right covering an angle of  $45^\circ$ ; when it reaches the right-end position it performs a rotation of  $90^\circ$  on the left and from now on the gun will cover angles of 90 degrees from one direction to the other.

The spaceship is able to shoot projectiles from the gun. Bullets come from the gun and are shot in the direction the gun is facing in a particular frame. To shoot in a particular direction we drew a target point at the far end of the scene. This point has the same y and z of the gun, while the x is computed using the tangent of the angle described by the gun, and placed far from it. Now that we know where the target is we can add the bullet to the scene and update its position at every frame,

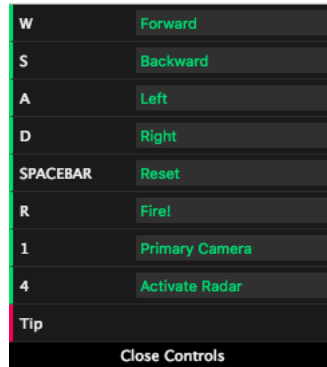
until it reaches the target and gets removed from the actual scene.

## 5 User Interaction

There are some user interactions in the project.

### 5.1 Legend

Our project has a friendly user interface, as can be seen in the following picture:



W	Forward
S	Backward
A	Left
D	Right
SPACEBAR	Reset
R	Fire!
1	Primary Camera
4	Activate Radar
Tip	
Close Controls	

Figure 1: The legend shows the available commands to the user.

This legend helps the user to get a deep knowledge of his potentiality and suggests him some hidden interaction such as the action of clicking on a planet to discover its name that otherwise would have been hard to find out.

### 5.2 Spaceship Movement

The user can control the *DeathStar* using the common configuration *WASD* on the keyboard. To be more precise, the user can move the spaceship through the X-axis using AD, the Y-axis using WS and through the Z-axis using the up and down arrow keys.

Moreover, the user can teleport the deathstar to its original position by clicking on *R*.

### 5.3 Cameras

The user is able to switch from the primary camera to the secondary one by clicking on *1* or *2*, respectively. Moreover, he can switch to the secondary camera by clicking

on *4*. In addition, this will actually activate the gun and it will start a star wars soundtrack. Once the gun is completely out, it will start to rotate of 90 degrees from left to right and, by clicking on *spacebar*, the user can shoot in the direction pointed by the gun. Of course, the user can decide to retire the spaceship's weapon by clicking again on *4*.

For what concern the primary camera, OrbitControls are included and they allow the user to control the environment by mouse clicking and by zooming in and out.