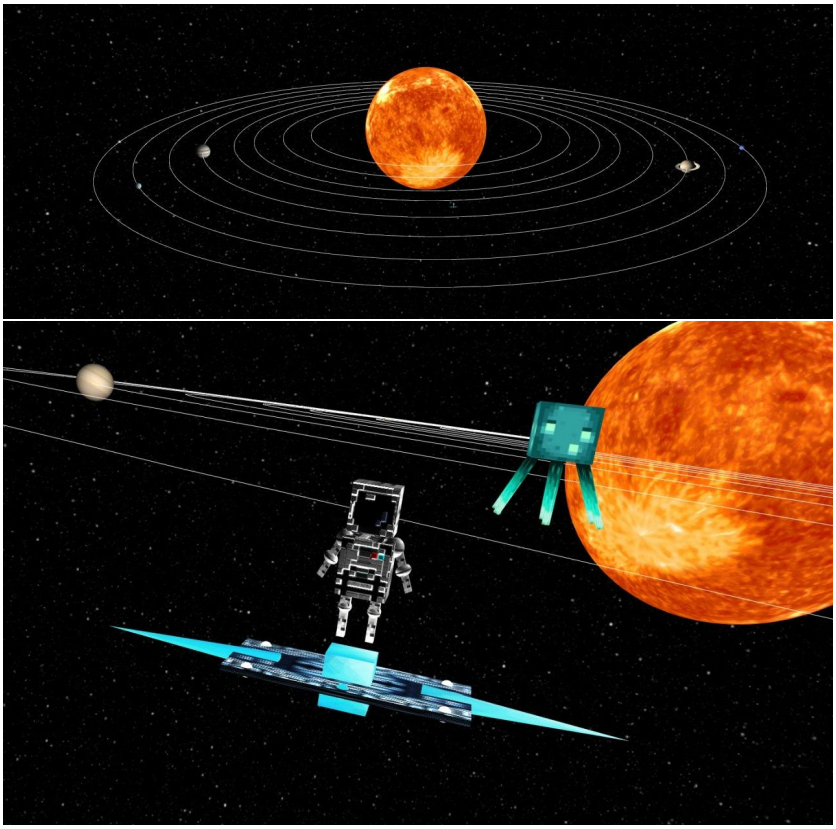
Team Rocket final project

Rosanna Greco 1760815, Sandro Materazzi 1965527

1 Introduction

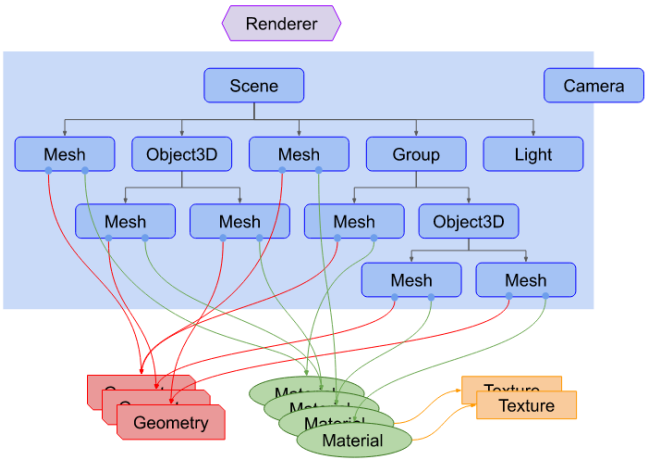
The purpose of the project is to create a realistic model of the Solar System and to give the users the possibility to explore it adding three different modalities, in one of which they can control the 3D model of an astronaut and see a smaller model of the system, observing different animations.



2 Description of the used environment

The project has been developed using `ThreeJs`, an advanced library used as an extension of `WebGL` which allows us to describe data in three dimensions. With this tool it is possible to define new classes and compose the scene in an abstract and easy way.

In fact, `WebGL` only draws points, lines and triangles, while, using `ThreeJS`, it is possible to draw more complex shapes. The graphs are formed explicitly representing the relationship between the various components of the scene.



The scene is the most important component, which contains all the objects we have to define. Each object is composed by two parts: the geometry (*i.e* vertices and triangles) and the material (with all its properties). The textures are considered part of the material.

The camera is not considered part of the scene, while the lights are handled as objects.

We pass our scene and our camera to a renderer, which use them to draw the part of the scene that is inside the *frustrum* of the defined camera on the canvas.

3 Libraries and models

- `ThreeJS`
- `THREEx.JS`

4 Solar System structure

The creation of the Solar System started from the biggest part of the model: the Galaxy background, a very big sphere that has an internal texture to represent the Universe. To make it visible a perspective camera is used with a big value on the far plane.
Next was created a light source at the center of the model with a big sphere around it representing the Sun, to make the light go through it is made of `THREE.MeshBasicMaterial`, while all the other planets are made of `THREE.MeshPhongMaterial`, so that bump maps and shadow could be there.

Then there are two different representations: a real one, in which there are true proportions in distances and orbits, and a simplified one, where the orbits are only circles and all planets go at the same speed. This choice was made because exploring the real system is nearly impossible due to the large distances between each planets and the velocity of each orbit. In fact

the exploration leads to hours of pain trying to find a single planet. So the simplified one has all the orbits very near and visible, also there are circles that highlight the trajectory of each planet.
To make this possible the orbit of each planet change based on the modality chosen by the user, so for the simplified one each planet has a circular orbit, in the other has an elliptical orbit with different speed based on the position of the planet, like it is in the reality.

Also a planet focus mode was added, in this mode the camera focuses a planet and follows it through the trajectory, this allows to watch each planet without struggling too much. So the planets focus mode uses the real orbits of the system, while the exploration one uses the simplified orbits(the exploration mode controls an astronaut that wanders in the galaxy).
The last mode is the free mode, in which the camera position is controlled and it grants the possibility to navigate in the Solar System. The free mode is not bounded with any type of orbits so both the real and simplified one can be explored, when it is activated the camera is unlocked and can go everywhere even over the Galaxy sphere.

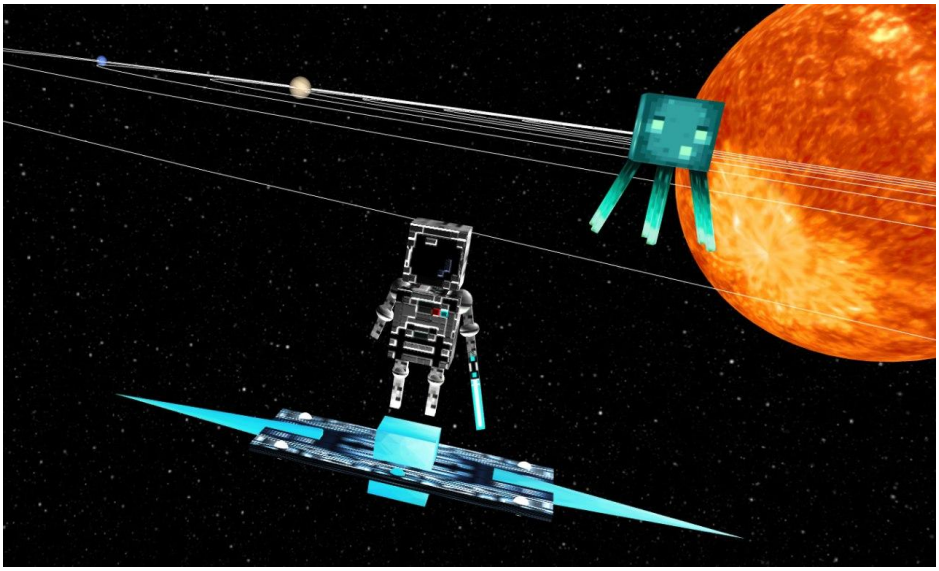
So to make it clear we have three different modes to watch the Solar System that is built in two different ways:

- **Planet focus mode**: the camera is locked near a single planet and follows it. It is bounded with the real orbit distances;
- **Exploration mode**: the camera is locked behind an astronaut that can be controlled and follows him. It is bounded with the simplified orbit distances;
- **Free Camera mode**: the camera position is controllable and does not have any kind of limit. It has no bound with any orbit but has the last type of orbits that has been used.

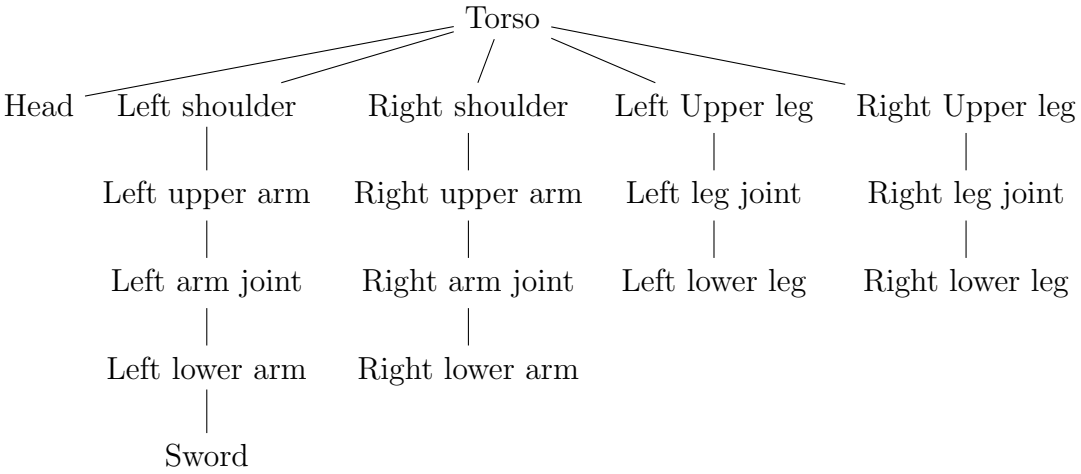
5 Hierarchical structure of the astronaut and its animations

5.1 Hierarchical structure

While in Exploration mode, the user is able to control the 3D model of an astronaut. In this case, the camera follows the torso.

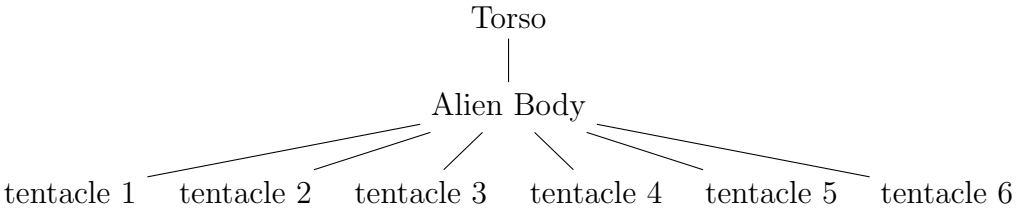


The body of the astronaut is part of a more complex hierarchical structure defined in the file `astronaut.js`, which comprehends the platform and the alien and can be summarized with the graph shown in the next page.

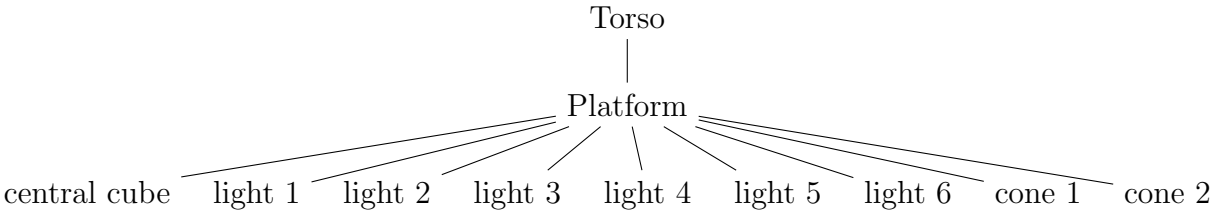


It is composed by geometrical objects defined using `three.js`, in particular cubes and spheres. In order to add a node to the scene using this library, it is sufficient to add it to its parent, using `parent.add(child)`.

The platform and the alien body are structured as follows.



They are defined as torso's children. In this way, we can apply the motion even to these elements.



Using this structure, the transformations applied on the torso will be applied also to the other elements of our structure.

On the platform we have defined seven lights: one placed on the cube and six on the little spheres. Another light has been placed inside the alien.

5.2 Textures and materials

In order to define a texture in `three.js`, we have to define a texture loader and then load each texture. Each material is assigned to a color texture and to a bump one. For the astronaut and the platform we used a Phong material in order to see some shadows, while the central cube of the platform, the alien and the small spheres are made of Basic material and have emissive properties.

In our case, we used nine different color textures: two for the alien (one for the body and one for the tentacles), three for the platfom and four on the astronaut (one for the face, one for head, arms and legs and one on the torso).

Some parts of the platform change color during the animation, so we defined two different materials in order to achieve this effect .

5.3 Collision detection

When the 3D model of the astronaut gets too close to the sun, a translation brings it in a further position. In order to achieve this, we implemented a

collision detection using two bounding boxes(one for the astronaut and one for the sun) and the `intersectBox()` function.

5.4 Animations

The animations applied to the defined structure have been implemented in the `animate()` function, which can be found in the `astronaut.js` file, while the controls used to move the astronaut in the environment are handled in the `Solar_System.js` file.

The model can be controlled using `WASD` and can go up and down using the spacebar and `V`.

We have three different animations: one used while the astronaut is standing still and floating, one which makes the character run and one which makes the character attack.

5.4.1 Floating animation

The floating animation is divided in two phases. During the former one the torso translates up, and the platform goes down, while the arms rotate downward. The alien's body translates and rotates and its tentacles rotate downward. In the second phase, the astronaut goes back to it's position, translating downward, while the arms move upward. The platform is affected by a translation in the opposite direction. The alien translates too, while its tentacles rotate upward to complete the movement. In the meanwhile, the light spheres on the platform change their color, switching between the two assigned materials.

The two phases are determined using a counter.

5.4.2 Running animation

When the button `X` is pressed, a boolean flag brings us to a different animation, in which we see the astronaut running. This motion is composed by four different phases, consisting in a series of rotations and translations of the various components, and explores the whole body structure. While the astronaut is running, the alien moves faster and the platform colors change faster.

Let's do a brief description of the above phases:

- During the first phase, the astronaut moves forward the left arm (rotating the left shoulder) and the right upper leg. The right leg joint rotates in order to make the corresponding lower leg move in the opposite direction, simulating the behaviour of a knee. The right upper arm and the left leg joint rotate backwards.

- In the second phase all the elements go back to their initial position.

- The third phase is very similar to the first one, but the left leg and the right arm are moving forward, while the right leg and the left arm move backwards.

- In the last phase, everything goes back to the initial position.

In order to switch to the previous animation, the `C` button shall be pressed.

Before changing the value of each flag, the astronaut goes back to the initial position, so the subsequent motion can start even if we are in the middle of one of the four phases described.

5.4.3 Sword attack animation

The sword (which is defined in the structure described at the beginning of this section) will be affected by the transformations applied to the left arm. It can be extracted only while the astronaut is floating, pressing `G`. Subsequently, pressing `J`, we can see the attack animation. In order to make the torso move without dislocating the platform, it is possible to make the object move in the opposite direction. During this motion, the camera doesn't follows the torso, so we can see the astronaut taking a step forward. The alien follows the astronaut and the left shoulder rotates.

In order to sheathe the sword, it is sufficient to press `H`.

We achieved this effect simply adding and removing the sword from its parent (the left lower arm) according to the value of a boolean flag. For this purpose two function have been defined in the `astronaut.js` file: `createSword()` and `removeSword()`.

6 Audio

In order to add a soundtrack, we created a non-positional audio object, defining an audio listener and a source using `ThreeJs`. This has been done in the

`astronaut.js` file. The loaded sound, in `.ogg` format, which is in the `sound` folder, is a simple arpeggio played by Rosanna.

7 Info overlay

When the application starts running, an overlay shows a list of all the possible interactions. It can be hidden pressing `O`.