Interactive Graphics

Warplanes

Giuseppe Gabriele Sirico, Bruno Cataldi

1810153, 2026604

July 2022

Contents

This is the report of the project for the exam of Interactive Graphics.

1 About the project

We have chosen to develop a game for our project, and in order to not shift the focus from the main goal (that is, the graphic part) we have kept things simple. The game is an endless run wave-type game, where the user controls a plane and tries to shoot down the enemies without getting hit.

2 Environment used

For the most part, the environment used was an advanced library of WebGl, three.js. The main function of this library is to provide easier access to complex operations of WebGl, so it can be thought as a sort of template for 3D web graphics. In our project it was used for handling the camera, the lighting of the scene, to build the models of the plane and also to apply the textures.

In conjunction with WebGl and three.js we also used html with an attached css to properly link the pages running the JavaScript scripts, model the UI and also for some human interaction, like choosing what plane would the user prefer to use.

3 Libraries, tools and models

3.1 Libraries

In conjunction with trhee.module.js (which is the standard module offered by three.js), we also used other two libraries from it, Water.js and GLTFloader.js.

3.1.1 Water.js

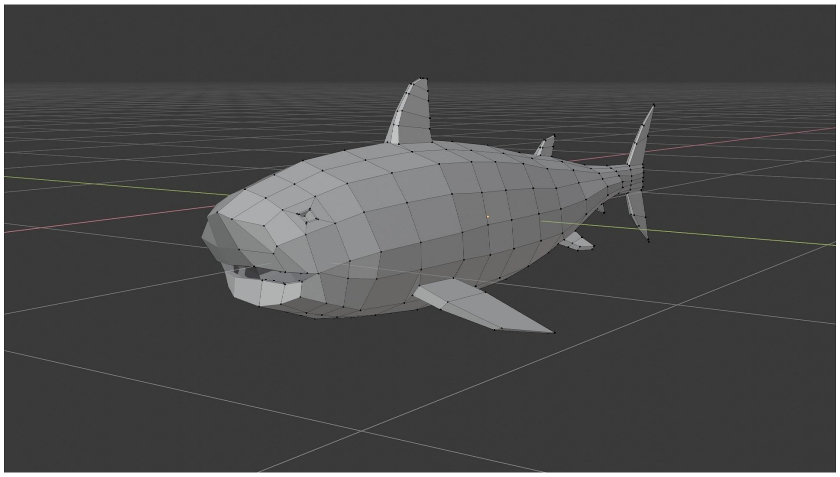
This library was used to give to a geometric plane the effect of moving water, by providing to it a texture corresponding to the normals of a body of water used as a material for the plane itself. This texture was taken from one of the example offered by the site threejs.org

3.1.2 GLTFloader.js

This script was imported in order to load into the project external model and render them into the scene. In the project, this was used to create the background. Low-poly models were used, in order to not slow down the program and to not draw too much attention to them (and instead focus on the main part of the project).

3.2 Models

This is the list of the models that were not made with WebGL, but rather were imported in the project



taken from CGtrader and made by the user ”karthic2k1999”

3.2.1 Island

taken from CGtrader and made by the user ”naren-sirigere”

3.2.2 Mill

taken from CGtrader and made by the user ”lowpolylab”

Each one of these model is free and has Royalty Free License.

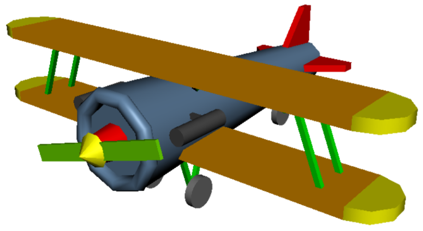
4 Technical aspect

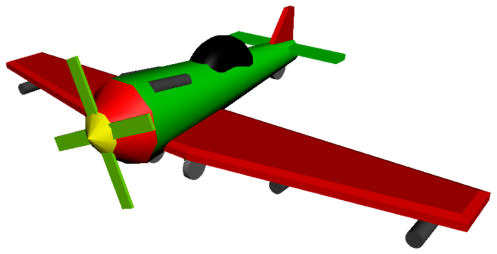
4.1 Planes Models

In the game are present two different models of plane, that are used both for the player's plane and the enemies planes. The two models have been constructed by combining different primitives provided by Three.js. This primitives are common 3D shapes that are generated with a bunch of parameters used to adjust the different properties of the particular shape. In general each of the two planes is a hierarchical model that is constituted by the following elements:

- A **Central Body**, represented as a cylinder, to which all the other parts are attached.
- **Wings** connected to the central body, that has a different shape according to the type of the plane.
- **Guns**, modelled as cylinders, are connected to the central body in the case of the first plane and to the wings in the case of the second plane.
- **Spoilers** are connected to the central body and placed in the tail of the plane.
- The **Tip of the plane** has been modelled different for each type of plane: it is constituted by several elements connected among them and then connected to the central body.
- The **Propeller** is modelled as a parallelepiped placed in the tip of the plane.
- **Poles and Wheels**: some poles are added to the plane model for an aesthetic purpose. They are used to connect the **wheels** and the **guns** to the plane.

The planes are first initialized as a Three.js Object3D() and then each part is gradually modelled and connected to its father of the model's three structure; furthermore, each part is a Three.js Mesh with a particular geometry and a particular material. In the figure below are shown the two models of the planes before the textures are applied.





4.2 Planes and other Objects Classes

All the code implemented for the planes and other objects in the game is present in the *mscript.js* file. In particular it contains the following 4 classes: **Plane()**, **Plane1()**, **Bullet()** and **Heart()** that are used in the main script to load the respective models in the game. Each class builds the 3D model and gives some properties and functions to the object. The two planes have a different structure from a 3D model point of view, but they share the same properties and functions. Given the fact that the two models are used both for the player's plane and the enemies planes, they have 2 different kind of properties: the one used for the enemies and the one used for the player. The enemies are equipped with the following properties and functions:

- Two arrays that contain the bullets
- A different bullet velocity for each plane
- A different bullet position for each plane, according to where the guns are situated
- A boolean variable to control if a plane has been hit (initially set to false)
- A shoot() function that uses the array of bullets to shoot them against the player
- A destroy() function that will make the plane collapse once it has been hit
- Two functions that will make the plane fly according to a different movement pattern

The player's plane is equipped with the following properties and functions:

- A number of lives that will make the player loose when they reach value 0
- A different bullet position for each plane, according to where the guns are situated

- A boolean variable to control if the player has been hit
- A removelife() function to remove a life when the player has been hit
- A destroy() function that will make the player's plane collapse once he has lost all his lives

The Bullet() class create the model of the bullets that are used by all the planes while the Heart() class create a heart shape to represent the lives of the player; the heart shape is created using Three.js Shape() that make a particular extruded geometry from a path shape. The hearts have a function fall() that make them collapse when the plane of the player is hit.

4.3 Game Mechanisms Management

All the code implementing the functionalities that will be discussed in this paragraph is contained in the *script.js* file.

4.3.1 Enemies Planes

The enemies planes are added in the scene thanks to the *generatePlane()* function: this function chooses a random plane model to be generated and then spawns it in a fixed area of the scene outside the playable area. The game has an increasing grade of difficulty because the *generatePlane()* function is called in decreasing intervals of time in order to gradually generate more enemies with the passing of the time. This has been done exploiting the *setInterval()* function.

4.3.2 Shooting & Collisions

The shooting part of the game is mainly managed by two functions:

- **generateBullet()** function: each time this function is calle, it creates the models of the bullets and places them in the actual position of the plane's guns.

- **collision(bullet, plane)** function: this function is used to check if a plane has been hit by a bullet. It takes in input a bullet and a plane and checks if the bullet position is in a certain range with respect to the plane position. In the case this condition is verified the function turn the hit variable of the plane to true making the plane collapse.

The function mentioned above are used for the bullets of the player's plane and for their collision with the enemies plane. Two similar functions (*generateEnemyBullet()* and *player-Collision()*) has been created in order to manage the bullets of the enemies and their collision with the player.

4.4 Animations

The animation process of the scene is handled by the functions *render()* and *animate()* together with the *requestAnimationFrame()* function. Moreover in order to render all the scene of the game with Threejs, a WebGLRenderer is created to take as input the scene and the camera.

4.4.1 Planes and other objects

Once generated, the enemies planes has their propeller rotating and they move directly against the player with a random pattern while shooting their bullets; this animations are managed by the *movePlanePattern()* and *shoot()* functions of the plane. The animation that make an enemy plane collapse when it is hit by the player's bullet is handled by the *destroy()* function; in particular to simulate a realistic collapsing animation, this function make the plane draw a parabolic curve towards the sea while it is rotating along its body.

To give a more realistic and smooth movement effect to the player when he is controlling his plane, it has been enriched with an animation that make the plane recline on the basis of the direction to which the player is bringing his plane. Moreover when the player lose a life a heart will drop from the left-high part of the screen thanks to its *fall()* function.
All the function mentioned above are called in the *animate()* function.

4.5 Background

As mentioned before, the background is kept simple, in order to not clutter the screen with too many objects. It's composed of 2 parts:

- The sea
- The models

As for the sea, it is made from a geometric plane, created with the three.js function planeGeometry() and given the size of 10000, 10000. After this, through the function water() (imported from Water.js) to the plane was assigned the material composed of a texture, waternormals.jpg, that was applied repeatedly (with the method RepeatWrapping), a sun with color: 0xffffff, a color:0x001e0f and a value of distortionScale of 3.7. This element is then loaded into the scene, and with respect to the time has the uniforms of the material changed, in order to make the water behave like an open sea.

The models are instead loaded with the GLTFloader() function, and after some adjustment like a resize and orientation to make them fit better with the scene, they are gradually moved south. This is done to give the impression to the user that the plane is actually moving forward, while in reality is the background that it's gradually moving backward. In order to better convey this illusion, the shark is actually moving a little bit faster than the two island, and it does so with a repeating motion, that makes it jump in and out of the water (similarly to how dolphin swim).

4.6 Texture

There are several texture applied to the objects in the project. One of them has already been spoken about, and is waternormals.jpg.

As for the others, they are applied to the two models of the plane. For the first model, it has a simple texture applied to the core of it (metal.jpg) to make it resemble a crude metal plane. On the wings instead, it has a color texture and a bump texture of bricks, that contribute to its simple but crude aspect.

In line with the first plane, the second model also has a simple and crude aesthetic, with the same iron on the core. However, instead of having a texture on the wing, this time the wings are kept textureless, and a color and bump texture of rust is applied to the spoilers and the tip of the core. This was done because due to the size of the wings of the second model, a texture applied to them wouldn't look as good in our opinion.

5 User interactions

Following this there is a rudimentary game manual, detailing how the user can interact

5.1 Game Controls

The user interaction consist in controlling the plane to avoid enemies' bullets and in shooting against them to survive as long as possible. The player can control his plane with arrow keys and can shoot with space key. The management of the user's input has been handled with two event listeners that are appointed to manage respectively the press and the release of a keyboard key. The user has also the possibility to pause and resume the game by pressing the key *p*.

5.2 Menu/HTML pages

The main menu page consists in a HTML page from which the user can start a new game by clicking the Play button or can first select the plane he wants to play with and then play the game. The main menu contains also a script that make some planes fly in the background. The selection of the planes gives the player the possibility to look at the planes models and choose the one he prefers to play.