La Sapienza University of Rome. Department of Computer, Control, and Management Engineering

Interactive Graphics Final Project

Jean Pierre Richa, Simone Faricelli

December 1, 2018

1 Introduction

The project is about creating a game we named FishIt see figure 1.1, which is an interactive game made as close to reality as possible, it simulates a fisherman driving a boat inside a world consisting of everything that we can find in a real world fishing environment, such as the sky, sun, light-house, boat, rod, water, and fishes. The aim from the game is trying to catch as much fishes as possible. Each fish caught increases the total score by 1.
We used rigid 3D objects, which we later animated using javascript such as the boat, and the rod, which can both be controlled by the user using the keyboard, and the fishes, which move randomly in the water. Furthermore, we included a hierarchical structure, which we are going to see in details when we discuss the implementation. The hierarchical structure was applied on several objects such as the boat and the rod. The fishes move randomly and when the rod is thrown in the water, we wait for the bait to collide with a fish and give the user a counter specifying the time within which he should pull the rod to gain points, otherwise the fish will escape and he will not gain a new point.

Figure 1.1: Top view for the whole environment.

2 Implementation

In order to have a clear view about the implementation, this section will be split into several part, where each part will hold details about the implementation of each object. The areas that will be discussed in the following subsections are: the creation of each object, the placement in the environment, whether or not the object is controllable and how it is controllable, and other details specific to each object.

2.1 Sky

The first object to consider in this project is the sky, because we need the environment to look as real as possible. The sky is nothing but a *Cube*, enlightened in a certain way and scaled by a value big enough to make it look continuous from the inside (See Figure 2.1). After that, the turbidity, luminance, and other characteristics were specified for the sky to fit this specific environment. The sky is clear and fixed, and cannot be controlled by the user. Finally, the above-mentioned characteristics were chosen to make the daytime look like if it's near the sunset.

Figure 2.1: External view of the Sky Box and the Sun Sphere.

2.2 Sun and Light

An important part of this project is the simulation of the sun as a light source. In order to have it, as for the sky, we added to the environment a huge *Sphere*, positioning it really far away from the center of the scene to make it as realistic as possible. After that, two light source were added. The first one is a *Directional Light*, which better simulates the sunlight when positioned in the same place of the sun sphere and directed at the scene, while the second one is an *Ambient Light* in order to simulate the diffuse light all over the scene, created by the refraction of the sun light through the atmosphere. For better visualization check figure 2.2. Moreover, another important issue to deal with when it comes to using lights in a scene is shadowing, so *castShadow* function for the light was set to true in order to get the objects in the environment rendered into the shadow map. **See also the Code section**.

```
light2 = new THREE.AmbientLight();
scene.add(light2);

light = new THREE.DirectionalLight();
light.position.set(x, y, z);
light.castShadow = true;
scene.add(light);
```

Figure 2.2: The sun viewed from the lake.

2.3 Hierarchy

A necessary feature is building a hierarchical structure. The structure is obtained through creating nodes in a hierarchical way where each node has a sibling and/or a child. The child is usually the node following the current one, and the sibling is the node parallel or next to the current one. The whole structure have to be aligned along a specific axis based on its desired position, because as we will see, sometimes an object should be below, next, or above the other one. First of all, and in order to have a clean implementation, we need to start by the central object, which is the lighthouse since the other moving objects will be turning around it. After creating all the nodes specifying whether it's a sibling or a child of another node, and specifying the location of the node using the parent's location, we will obtain all the parts connected to each other. For more clarification **See the Code section**.

Code

```
function initNodes(Id) {
    switch (Id) {
  case boatId:
            t1 = t.add(new THREE.Vector3(270, -6, 600));
            figure [boatId] = createNode(t1, boata, null, rod1Id);
            break;
```

As shown above, this is the case of creating the node for the boat, which is rendered using the function *boata()*, and has the handle of the rod(rod1) as a child.

2.4 3D Objects

The approach for adding the different materials used in this project, is importing non-animated 3D objects such as the rod, boat, lighthouse, and fishes.**See e Code section**. Using the hierarchical structure we can put different parts together, which in this case are the previously listed objects added to the environment scene. The main idea of using the hierarchy is to simplify the relation between the boat and the different parts of the rod (handle, rope, and bait), which were split to make it easier for the collision checking with the fishes. The boat was imported and then coded from scratch to be moved in the environment by the user. The rod can be thrown and pulled by the user also, but the trick here is to tell the user within which time frame the rod should be pulled after colliding with the fish and this was done using a counter that shows on the screen the slot within which the fish is still in collision with the bait. If the rod was pulled in this time frame, by pressing *"SpaceBar"*, a point will be added to the final score. Otherwise, no points will be gained.
The boat can be driven using the keyboard keys:

- W for moving forward,
- S for moving backward,
- W + A for moving left forward,
- W + D for moving right forward,
- S + A for moving left backward,
- S + D for moving right backward.
- The rod can be thrown using "G" and pulled using "F"

Code

```
function boata() {
    loader.load(
        '/models/boat2/scene.gltf',
        function (gltf) {
            boat = gltf.scene;
            scene.add(boat);
        }
```

Figure 2.3: View for all the objects.

2.5 Camera

It is important for the user to see what is going on during the time he is playing, more specifically it is crucial to see everything relevant to getting a higher score, and knowing his/her position in the environment. In order to provide this feature, the camera was placed on the boat, where the view changes based on what the user is doing. For example if the user is driving the boat, the camera will show what's in front of it (see 2.4), but when the driving stops, the camera will show the rod, so the user knows whether it is in the water or not (see 2.5).

Figure 2.4: Camera view while driving the boat

Figure 2.5: Camera view of the rod when the user is not driving the boat.

2.6 Fishes

A 3D model of a goldfish was imported and used, it was added to the scene a Hundred times, each with a different name. The fishes move randomly in the environment at a constant speed. They move randomly, in order to make it more difficult for the user to predict the next direction of the fish, which makes the game harder and the collision more difficult to happen. The fishes were put at a depth of 20 pixels below the water surface (see 2.6), which is their fixed depth.

Figure 2.6: Fishes view from underneath the water.

2.7 Collision

In order for a collision between the bait and a fish to happen, the bait and the fish should be covered by *Boxes* made by *Wire Material* and at the same height on the Y axis, which is 20 pixels below the water surface as previously stated. Because the vertical axis here is Y, since we are using three.js and this is the reference frame orientation in this library. They also should be on the same X, and Z axes points. Boxes were created and coded so as to follow the fishes and the bait movements, so a collision happens when wires of 2 boxes are intersecting.

Boxes code

```
var box = new THREE.Box3().setFromObject(bait);
var cubeGeometry = new THREE.BoxGeometry(box.getSize(center).x,
    box.getSize(center).y, box.getSize(center).z);
var wireMaterial = new THREE.MeshBasicMaterial({
    opacity: 0,
    color: 0xFFFFFF,
    wireframe: true
});
baitBox = new THREE.Mesh(cubeGeometry, wireMaterial);
scene.add(baitBox);
```

3 Conclusions

Finally, a *Homepage* containing the information about the game and its developers has been added at the beginning of the game. So as an FPS counter, in the top right of the page, due to the heavy load of the game. Also background and events audio sounds, were coded and inserted into the game, such as a relaxed whistling or the cast and pull up sounds.

3.1 Launching The Game

For the application to run, we need first to establish a local server, which can be done using python through navigating to the game's directory using the command prompt (terminal shell) and writing: "python -m http.server". After the connection is established, one can open a browser and type "localhost:8000", this will allow the game to load in the browser page.

4 Future Improvements

- Levelling System
- Adding Physics to the Fishing Rod
- Randomize Fish Size and Adjust the Score Based on That
- With a certain Probability, Fishes are Moving to the Bait

5 Libraries, Frameworks and References

Three.js is a cross-browser JavaScript library, based on WebGL, and Application Programming Interface (API) used to create and display animated 3D computer graphics in a web browser. The entire scene and all the libraries such as water, sky, camera controls and loader is made in Three.js

Sketchfab was used to download the raw 3D objects, such as the boat, rod, fishes, and lighthouse.

Blender was used to correct the the reference frames of the objects, the orientation and other minor adjustments