Interactive Graphics final project

Grossi Manuel

1. Task Description

The theme of the project was the decision of the students, as long if fit a given set of requirements:

- Use at least one hierarchical model
- Use lights and textures
- Enable some kind of user interactions
- Implement custom animations exploiting the hierarchical structure of the models.

This project may be implemented using standard WebGL or more advanced libraries, such as ThreeJS or Babylon, if approved.

2. Proposed solution

My solution consisted of implementing a version in the well-known genre of games, known as the Endless running. In this type of game, the user controls the position of a moving character so to avoid incoming obstacles. This way, the character continues moving, and thus increasing the score. The game generally only stops when the character collides with one of the said obstacles, thus explaining the endless nature of its name. This genre received wide acclaim over the past decade, especially in the mobile ecosystem. Some of its most successful franchises are: Subway surfers, Temple Run and Jetpack joyride.

Figure 1: Temple Run (left) and Jetpack joyride (middle)

My variation of this game is called **Vespa traffic race** and, in the same spirit, consists of a boy on the historic vespa whizzing through traffic and avoiding obstacles (cars).

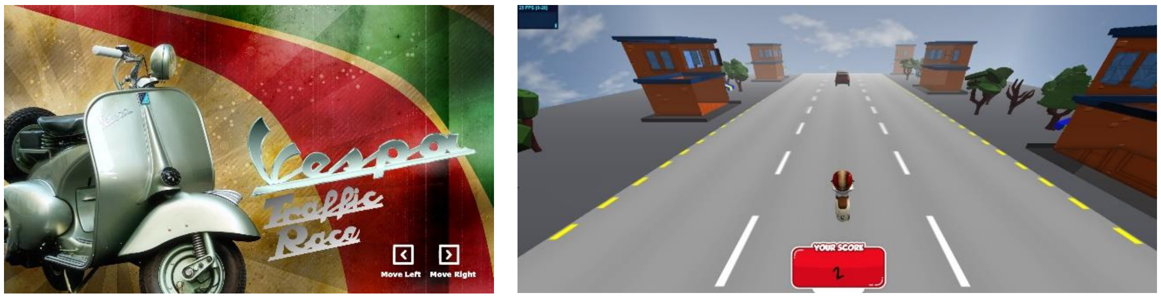


Figure 2: Front cover and gameplay of Vespa traffic race

As in the precedent examples, the Vespa traffic race runs continuously, only stopping once the main character hits one of the obstacles. Furthermore, to avoid user disengagement throughout the gameplay, the game's speed improves over time, so to keep the game entertaining and challenging.

3. Key features and design choices

In the following subsections, we describe some details of the implementation of Vespa traffic race.

Frameworks

The whole application was built exclusively around ThreeJS. As we later see in detail, all the transitions and physical properties were completely accomplished by JavaScript, without relying on external libraries, such as TweenJS.

World

Like the mentioned examples, this game takes place in a flat road, so, to achieve this effect, the implementation revolved around a plane surface. Essentially, the camera and the hero remained still along the z-axis, and the road below them scroll along the z-axis, thus giving the effect of running on a road. To prevent game repetition or excessive computation, the obstacles appear and disappear based on their location on the screen. Once these disappear, they are fed a pool to be reused later in the game. Furthermore, a sky-box complete the scenario of the game. The cube has a fixed size like the flat road and can show different backgrounds depending on the provided path. For deployment, however, this option is set fixed, but it is easily adjustable by using the createPathStrings function in the src/world.js file.

Main character

The only interactive model used in this application is the hero, a guy over on the vespa whose objective is to reach the end of this endless road. It consists of a ready-made model that was modified for this specific task. It is a hierarchical model with a total of 3 sub-elements.

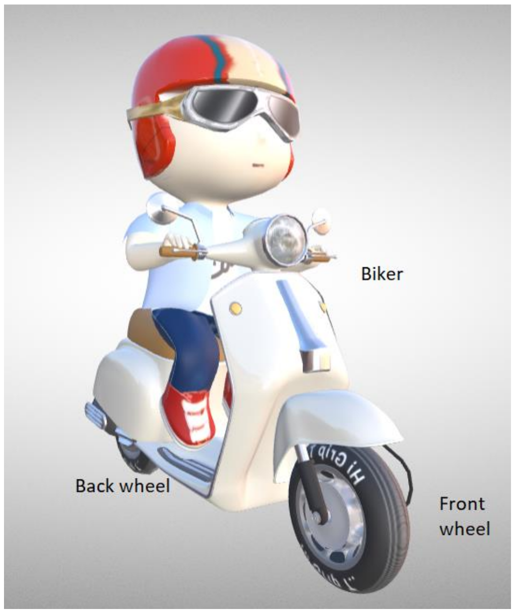
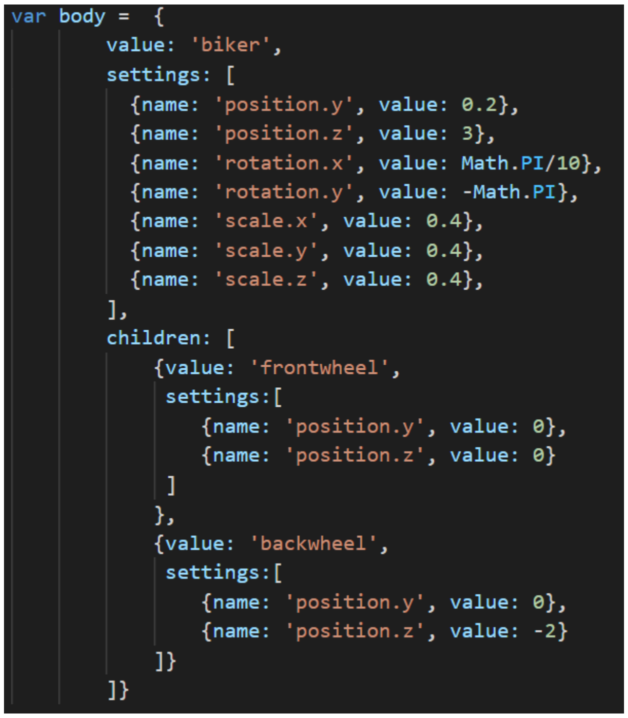


Figure 3: Structure of the hero

Along the y-axis of the model, all parts are recursively dependent, starting by the upper part of the model. Some details of the model, such as the two wheels were kept fixed in their respective locations since these would not be used. Additionally, since each element of the model was detached from one another, the devised solution to build the hierarchical structure based on a recursion method, that not only linked parents and children but also applied the necessary modifications to ensure the correct location of each part of the hero's body.



```
var body =  {
        value: 'biker',
        settings: [
          {name: 'position.y', value: 0.2},
          {name: 'position.z', value: 3},
          {name: 'rotation.x', value: Math.PI/10},
          {name: 'rotation.y', value: -Math.PI},
          {name: 'scale.x', value: 0.4},
          {name: 'scale.y', value: 0.4},
          {name: 'scale.z', value: 0.4},
        ],
        children: [
            {value: 'frontwheel',
             settings:[
                {name: 'position.y', value: 0},
                {name: 'position.z', value: 0}
             ]
            },
            {value: 'backwheel',
             settings:[
                {name: 'position.y', value: 0},
                {name: 'position.z', value: -2}
            ]}
        ]}
```

Figure 4: Body specifications fed to the recursive building function

To make the animation process much easier, the building method also provided a template of the standard position and rotation of each part of the model. This proved useful during animation so as to avoid repetition when specifying the desired target poses. When a particular element of the full pose was not provided, the standard values were used instead.

Obstacles

The obstacles used in this game are simple cars. The cars are divided into two randomly chosen types: one red and one blue. As previously mentioned, these models are only rendered in a small portion of the flat road. To ensure variability in the gameplay, these have minor random changes, in particularly the position on the road. The models are rendered at a regular interval and can also have slight modifications with respect to the number of obstacles per interval.

Animations

In addition to the previously mentioned scrolling plane, the game also has character animations. The mechanics of the game allow two types of movement, two lateral (left and right). For both cases there is a specific animation. To achieve the desired effect, when the user issues a certain game command, it chains the sequence of poses to be interpolated through a certain number of frame updates. Each frame update fills a small gap between the current and the target pose. When these are through, the only animation that remains is the standard vertical flicker of the contact between the hero and the surface, produced by the addition and subtraction of random noise and a damping factor here called gravity.

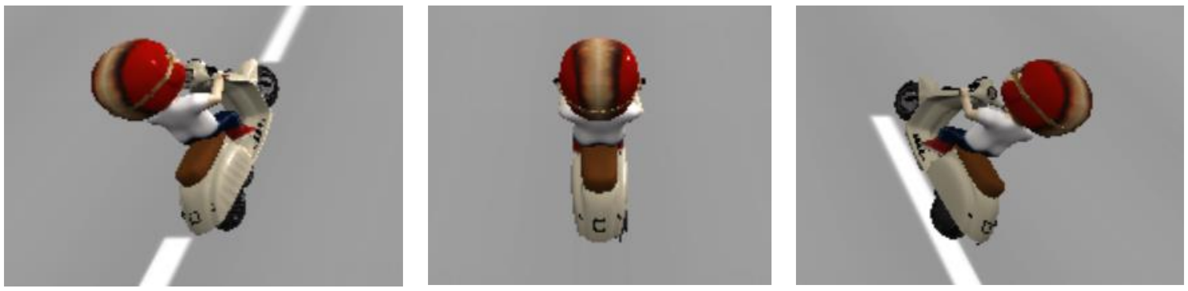


Figure 5: Poses and animations

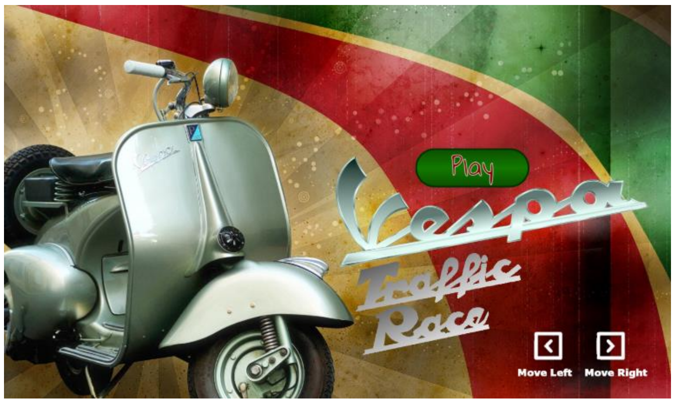
4. Considerations

Taking into account the technical details previously described, the author believes that the project meets the agreed requirements. The overall game experience is satisfying, and it was well-received among friends and colleagues. Some details:

- When the game starts there may be some small clicks of the scene due to the loading of the models
- The game is not very heavy but if you don't have average hardware it could be very slow in terms of fps

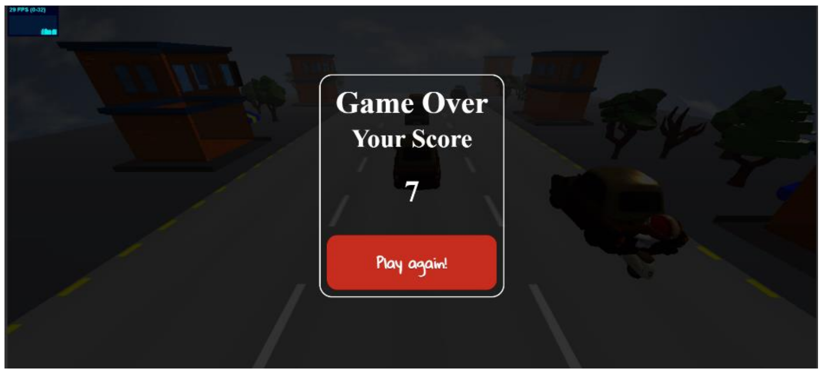
5. User manual

1. Begin by pressing start in the initial page



2. Move the character by pressing ⬅ (left) and ➡ (right) keys to avoid incoming cars

3. If you hit an obstacle, you lose the game. If you wish to play again, click the button "Play again".



References of the models used in the game

Boy and vespa: https://sketchfab.com/3d-models/scooter1-ca94ac7354044f5db4786ef676410675

Cars
Red: https://sketchfab.com/3d-models/british-compact-car-low-poly-model-f1e9c43c61bb492a9b15d84f9a16c0f7

Blue: https://sketchfab.com/3d-models/cartoon-sports-car-0fd87559642a41b7a7924876ad3e9399

Building: https://sketchfab.com/3d-models/building-cartoon-low-poly-e0f9c472e6b8470ba14fd8c600936709

Trees: https://sketchfab.com/3d-models/low-poly-trees-2e70c34af8994852acd4b9ffce596336