Three-Dimensional Vizualization and Animation
Technical Report - Assignment III.

Marfeychuk, Mykhaylo
ist194039

Skalický, Matyáš
ist194904

Introduction

The following document describes the work on the task 3, the development of the frogger game in the ThreeJS framework.

1 Cameras and stereo effect

The application contains several cameras. The player can switch the camera using the keys 1 to 5. All of the cameras also show information about the game except for the stereo mode, for which we do not show any information on the screen to make the screen less cluttered.

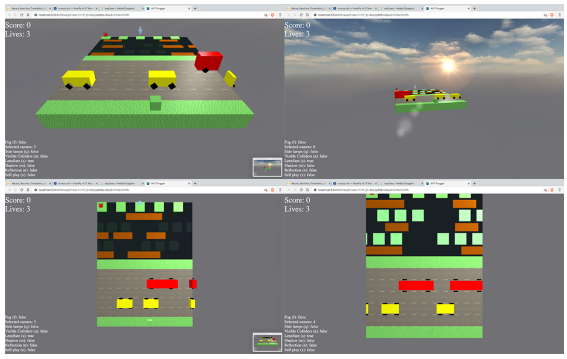

Figure 1: From top left: perspective following camera, rotating perspective camera, top perspective camera and top orthogonal camera.

The stereo viewing was implemented based on the module provided on the ThreeJS demo accessible at `https://threejs.org/examples/webgl_effects_stereo.html`.

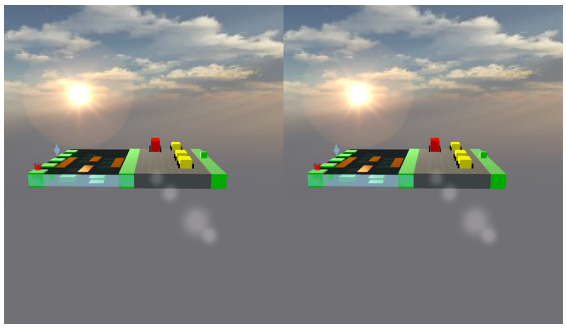

Figure 2: Stereo perspective rotating camera.

2 Object movement and collision detection

Each frame, we first check the collisions, update the game objects positions and then draw them to the screen.

Collision detection is done by checking the collision of the player and all of the collidable game objects each frame. ThreeJS makes this very easy as we can easily precalculate the bounding box for a given geometry and then easily check for collisions between them.

3 Lights

The directional light can be seen on the Figure 1. The 8 point lights in the scene can be toggled with the usage of q key. We used the built-in ThreeJS light classes.

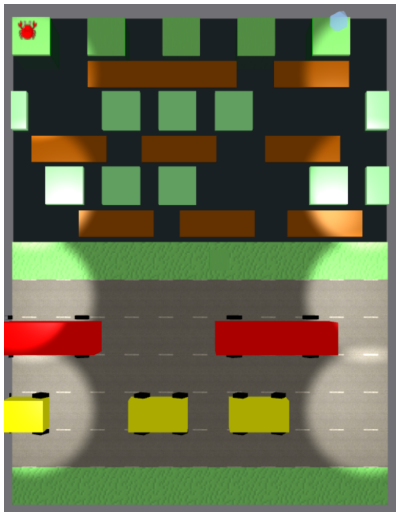

Figure 3: Pointlights activated.

4 Scoring system

Scene contains the current score on the top of the screen as shown on the Figure 4. The score is increased when the player reaches the target crystal on the other side of the river and the game also speeds up to get harder.

Figure 4: Scoring.

5 Fog

Creating fog in ThreeJS is done simply by using the THREE.Fog() class. User can toggle the fog on and off using the f key. The result is shown on the Figure 5.

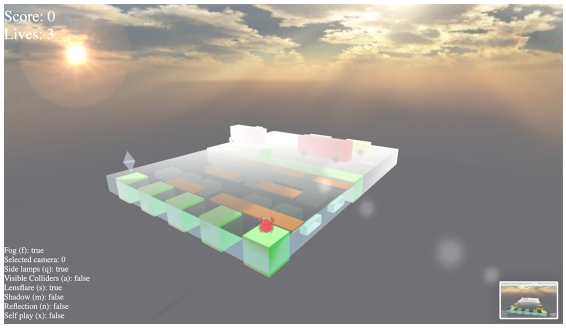

Figure 5: Fog.

6 Planar reflections and shadows

The planar reflection is implemented using a cube camera positioned on the other side of the plane which is then projected to the surface of the water. This effect, where the skybox is projected can be seen in the Figure 6.

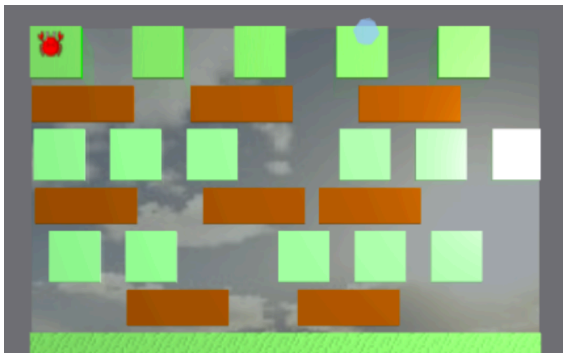

Figure 6: Reflection.

The shadows are created by using the native Three.js framework methods, which already does all the necessary to render the shadows. The framework first renders the models onto a plane, and then renders the real object on top. The shadows can be seen in Figure 7.

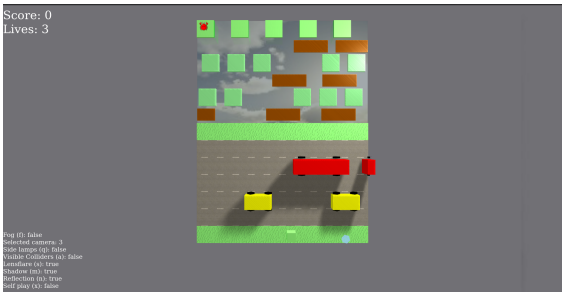

Figure 7: Shadow.

7 Billboard

We used a little crab to demonstrate the usage of billboards in our game. This can be seen for example on the Figure 5.

8 Lens flare

For our demo, we have utilized the ThreeJS Lensflare class, which makes the lens flares very easy. the lens flare is located at the "sun" in the sky and can be seen on the Figure 8.

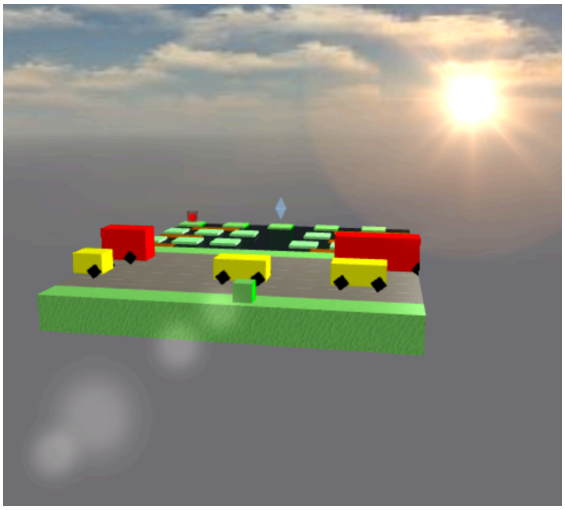

Figure 8: Lens flare.

9 Texturing and skybox

ThreeJS allows the developers to write custom shaders for a material. The skybox is done using a large cube textured with a custom GLSL shader cube map. The skybox can be seen on the Figure 5.

We gave textured the ground with a grass texture as well as the road with the road stripes texture.

10 Other features

We have used the clipping mechanism to remove the objects which come into the scene and also go away.

We have also implemented the autoplay. User can activate the auto-play feature to test the game when using the VR glasses. This is done by pressing the x key.

Conclusion

We have implemented most of the required functionality for the third task of our project. Sadly, due to time contains and also one of our group member missing, we have not implemented the bump-mapping and a particle system. The mobile gyroscope controls were developed, however we ran into a technical problem with hosting. Mobile browser is not able to access the gyro data over http and our hosting does not have the SSL set up.

The testing was difficult as Matyáš's old phone did not run any of the WebGL content, and Mykhaylo's phone was too large to fit into the Google Glass headset.

We have ported most of the functionality from the original C++ game into the modern ThreeJS environment. Development in ThreeJS is incomparably simpler than the low-level C.