Memorial University of Newfoundland

COMP-6909-001: Fundamentals of Computer Graphics

PROJECT REPORT

Final Project: Car Driving Game

Group

Dang Tran

Leonardo Eras

Professor

Oscar Meruvia-Pastor

Submission Date

09/04/2025

Introduce

Car Driving Game is a 3D game made solely by WebGL2, with no external library usage. It is a game where the user can control and drive the car around the scene to collect coins and avoid being hit or running out of fuel. The control is quite simple, which allows the user to steer the car, heading up and down to collect or avoid obstacles on their path. Below are some characteristics and features of this game.

Features

3D model loader: we implemented a custom GLTF 2.0 model loader which load the textures, vertices, normals, and materials of a model from a GLTF file, with a corresponding bin file.

Physical-based Rendering: we use metallic, oclussion, and roughness textures and factors loaded from the models in the fragment shader for a more realistic look.

Mesh Storage: Each mesh in the model is loaded into a Vertices Array Object (VAO) and stored in the Mesh Storage. Since the scene contains hundreds of trees, each shares the same mesh with different transformations, we improve memory storage by only storing the original mesh. This also reduces the model loading time since we don't have to load a mesh multiple times.

Prefab Storage: Some objects like trees, coins, and fuel tanks are the same with some shared transformation and some different transformation, we load one instance of them and stored it in the Prefab Storage. Each time a new tree or coin is added, we clone an instance, containing the mesh information and shared transformation, and add a new specific transformation for the cloned instance.

GPU Instancing: To optimize calls and data pass to GPU, we utilized GPU instancing features of WebGL2 to only pass the VAO once and pass all the transformations of objects that use the same VAO as a buffer. The GPU will process each as an instance and we can dramatically reduce the nnumber of call to draw function of WebGL.

Box Collision: The car can collect coins and fuel tanks, or loses when crashing onto trees or clouds. We implemented collision using the Axis-aligned bounding box technique. The game automatically generated a bounding box for each physical mesh. We manually added a "physic" attribute in the GLTF files. Those with "physic" is "true" will be considered a physical mesh. There's an option to visualize the bounding box on the screen using the switch on the menu.

Environment effect

Skybox: we created a sphere skybox with a panorama effect using a sphere object with an emissive texture and loaded with depth mask disabled.

Day and Night: we implemented a day-night cycle, reflected by the strength of the ambient light. At night, the ambient night is weak and there are fog effect on the scene. The ambient and diffuse value of the ambient light are adjusted based on the day-night time.

Fog: we blended a simple fog effect at the end of the fragment shader, which can be adjusted by the user. The intensity of fog effect is changed based on the day-night time.

Emissive light: The car have front and back lights, which are created using emissive mapping and can be turn on and off. The lights are useful for user to navigate in fog and dark conditions. There's also and option to change the color of the light from white to blue.

Movement Control

The game is designed to be event-driven. It processes and re-renders each time a new movement is passed to the movement queue. Upon receiving the movement object, each object will extract its transformations and apply them. By default, the screen will keep rotating around the scene, and the car will automatically move on its path, creating a cycle. We achieved this by generating a movement object for the camera and the car every 100ms and push that to the queue. We used a single perspective camera that moves independent from the car and can be controlled by the user through the keyboard.

User Manual

Deploy and Run

We developed the game using ReactJS one-page application. Noted that NodeJS is required to run the application. First, simply run “npm install” in the code folder to install missing dependencies if the node_modules folder is not provided. After that, run “npm start” to start the application. The application is available at port 3000 by default. Models needed for the game are already included in the code folder.

Play the game

The Car: user will control the car using “A”,”W”,”S”,”D” keyboards. Pressing “W” and “S” will make the car heading up and down, respectively. Pressing "A” and “D” will make the car steer and flip to the left and right side. The car can only head up and down for a maximum of 45 degrees.

The Fuel: The car will consume fuel upon moving. When the fuel is depleted, the user will lose and have to restart the game. User can increase the fuel by collecting fuel tanks, which are generated randomly every 5 seconds at a randomly rate. Each fuel tank collected will increase the fuel bar by 5.

The Coins: The user will try to collect as many coins as possible to increase their score. Each coin give the user 1 score. Coins are generated randomly every 5 seconds at a random rate.

The Night: There is a day-night indicator in the menu panel. At night, the scene will turn dark, and fog will appear until the next morning. In heavy fog conditions, the user may want to turn on the car’s lights to navigate and locate the car’s position. However turning on the lights will consume more fuel, so the user should turn it off in the morning to keep a balance.

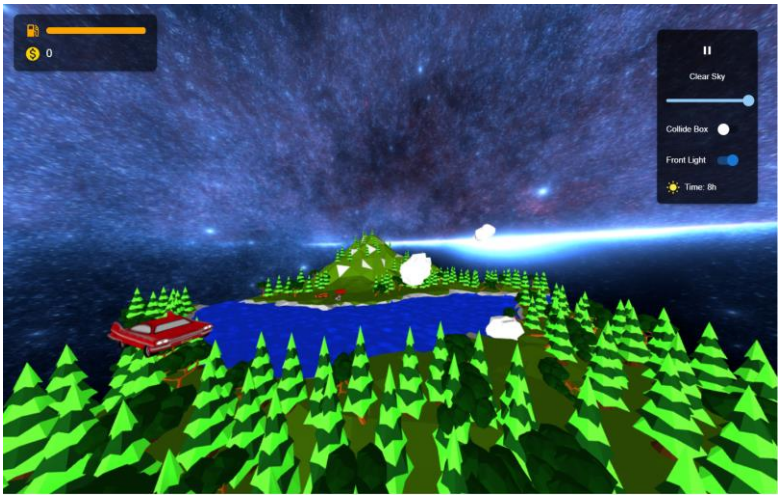
The Menu Panel: On the menu panel, the user can click the stop/resume button to stop or continue the game. Next, there is a slider to adjust the fog level, or sky clarity. When the slider is at minimum value, the fog is heavy and vice versa. The user can also toogle between showing bounding boxes of objects or not with a toogle button. After that, there is another toogle button to turn on or off the lights of the car, and 1 toogle button to change the light color to blue. Finally, there is an indicator showing the current daytime.

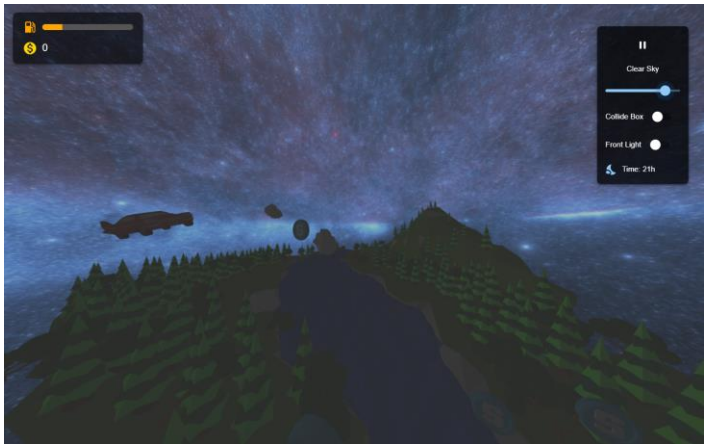
The Camera: The user can zoom in and out using “Q” and “E” keyboards. The user can also rotate the camera along X, Y, Z axis using “X”, “Z”, “C” keyboards respectively. This feature is just for exploring the scene as it will heavily effect the gameplay if the camera is rotated.

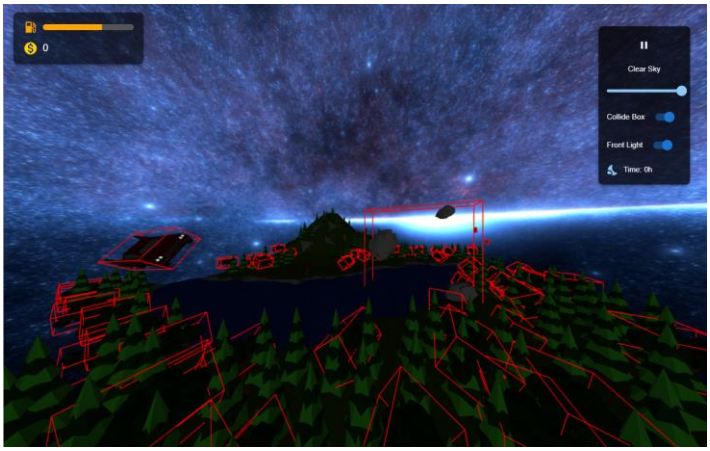
Acknowledgements

All the code submitted is written by ourself, with no usage of any external libraries. Models are collected from free sources online and were modified to serve the project.

Screenshots







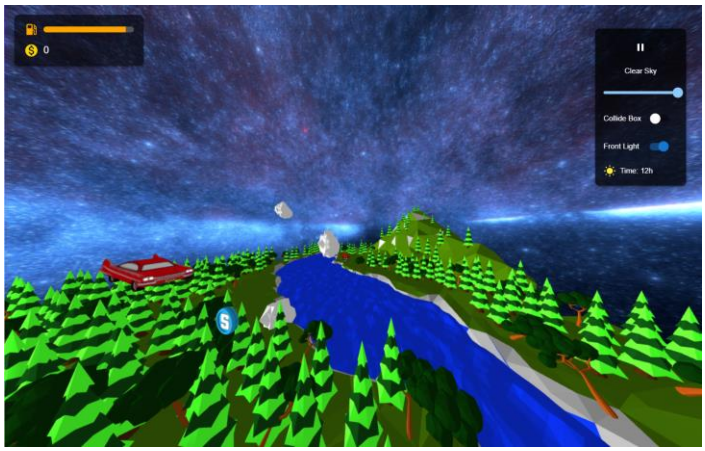


Figure 1: Car at day time

Figure 2: Car at foggy night with light on

Figure 3: Car at night with no fog and light off

Figure 4: Collide boxes are visible

Figure 5: Car is going to collect a coin

Video

A video to introduce this project is available at: https://youtu.be/DDrCW8bC4UA

References & Sources

No external resources except those provided by the instructor were used to produce this assignment.