

[**Three-Dimensional Visualization and Animation**](https://fenix.tecnico.ulisboa.pt/disciplinas/AVT351795/2017-2018/1-semestre)

1st Semester 2017/2018



3º Assignment

Project Report

Group 13:

78414 Ricardo Silva

78516 João Loureiro

# Introduction

For this third assignment we migrated our Micromachines application from OpenGL to WebGL. We did everything in WebGL without using the Three.js framework.

We started by migrating and implementing the features we already had in the OpenGL application starting with the basic: the cameras, lights, object movement and collisions.  
We then implemented the techniques we already had: the transparent objects, fog, particle system, 2D lens flare and billboards.

After that we implemented the features specific to this assignment:

1. Bump-Mapping
2. Planar Shadows
3. Planar Reflections
4. Animation
5. Stereo Vision
6. Accelerometer

As explained in the previous report:

* To create the track objects’ position we wrote a script to create a file with the specified positions which in then parsed in our application;
* To texture the table we implemented multi texturing using two textures that are mixed according to a black and red mask;

The link to the sigma machine is the following: <http://web.ist.utl.pt/~ist178414/AVT-WebGL/>  
Or use the following QR code:



# Bump-Mapping

We implemented bump-mapping on the table’s textures to simulate the bumps in the textures and in order for them to look more realistic.

## Method

The first thing we needed to do was calculate the tangents, we do this when we load the models. Next we load the two normal maps the same way we load all textures.

We then pass both the tangents and the normal maps to the shaders where the rest of the work is done.

## Results

|  |  |
| --- | --- |
|  |  |
| Fig. 1 – Before bump-mapping | Fig. 2 – After bump-mapping |

# Planar Shadows

FillerFillerFillerFiller FillerFillerFillerFiller FillerFillerFillerFiller

## Method

FillerFillerFillerFillerFillerFillerFillerFiller

## Results

|  |  |
| --- | --- |
|  |  |
| Fig. 3 – Shadows orthogonal camera | Fig. 4 – Shadows car perspective camera |

# Planar Reflection

We implemented the planar reflection in the rear-view mirror inside the car, using a stencil.

## Method

To accomplish this goal, we used an extra camera located inside the car and oriented towards the road behind the car. First we render the scene as usual, then we render the mirror with the stencil testing active and finally we render the scene using the backwards camera. By using this technique, we are able to create the illusion of a reflection in the rear-view mirror.

## Results

|  |  |
| --- | --- |
|  |  |
| Fig. 5 – View from inside the car | Fig. 6 – Another view from inside the car |

# Animation

We did a pre-defined animation in order for us to be able to run it in a mobile phone.

## Method

To record the animation we did two laps while recording both the car position and angle, which we then save to a file.

To run the animation we simply set a flag so that the position and angle of the car is set from the ones read from the file instead of doing the car update function.

## Results

|  |  |
| --- | --- |
|  |  |
| Fig. 7 – Animation playing code | Fig. 8 – Recording code |

# Stereo Vision

In order for the animation to work with the google cardboard we implemented two stereo cameras.

## Method

The stereo camera works by dividing the screen in two, one side for each eye. Basically we do one lookAt and one frustum for each side.

Also we alternate between two stereo cameras randomly, one from the outside of the car and one from the inside.

## Results

|  |  |
| --- | --- |
|  |  |
| Fig. 9 – Stereo vision from outside the car | Fig. 10 – Stereo vision from inside the car |

# Accelerometer

To change the camera orientation while the animation is running on a phone we used an accelerometer.

## Method

To do this we listen to changes to the device orientation and run a handler if it does, using the addEventListener.

The handler is a function we created that gets the orientation of the device in the three axes and if the change is greater than the specified number, we update the accelerometer with the new values for the axes and use them to draw the scene.

## Results

|  |
| --- |
|  |
| Fig. 11 – handleOrientationEvent function |
|  |
|  |
| Fig. 12 – GameManager updateGyro function |

# Extras

FillerFillerFillerFiller FillerFillerFillerFiller FillerFillerFillerFiller

## Weather

To add to the atmosphere of the game we change the colour of the sky depending on the current time of the day and weather using the clearing colour of WebGL.

## Scoring System and finishing Line

Our scoring system consists of tracking the time a player takes to complete a lap around the track. The best time and the current time are displayed on the top left corner of the screen. To complete the system, we added a finishing line, using a texture, to the place where the game starts and finishes counting the time.

|  |
| --- |
|  |
| Fig. 13 – Finishing Line |

## Custom .obj and .mtl loader //TODO TODO TODO

To load .obj models we wrote a custom loader that parses .obj files and the corresponding .mtl files and creates the meshes in our application. The loader retrieves all the necessary elements (vertices, normals, texture coordinates and material) and then creates the VBOs and VAOs necessary to draw the meshes.

# Conclusion

We completed the project successfully by migrating all the objects/effects, plus some additional effects to complement the game, from OpenGL to WebGL and implementing the six new requirements. During the development of this project we acquired new skills and had the opportunity to learn more about WebGL.