

[**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 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. The normals are encoded in the normal maps and are defined in tangent space.

In the vertex shader we used the normal, tangents and light directions to calculate the TBN vectors and build the matrix to transform from Eye Space to Tangent Space. Next, we transform the light directions and half vectors to tangent space. Then in the fragment shader we proceed as usual to compute the lighting of the fragments.

## Results

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

# Planar Shadows

To improve the visual appeal and realism of the application we needed to implement shadows. To achieve this goal we used planar shadows using the plane of the table as the target plane for the shadows.

## Method

To draw the shadows, we begin by drawing the table with the lights on. Then we activate the stencil mechanism and draw the table in the stencil buffer. The we use the shadow matrix computed with the Blinn Technique presented in the class and draw the shadows of the objects in the stencil buffer where the stencil was filled by the table. Lastly, we draw the table with all the lights off using the mask created by the stencil to draw only in the places where both the table and the shadows were filled in the stencil buffer. By using this technique, we achieve great results that guarantee that the shadows are only drawn in the table and that the shadow color is exactly the color of the table if there were no lights present.

## 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 the stencil mechanism.

## 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 defined an animation for the car for us to be able to demo the application in a mobile phone without requiring user input to steer the car.

## Method

To record the animation we complete two laps around the track while recording both the car positions and angles every couple frames. We then output all the saved positions and angle to a file.

To run the animation we replace the user input and car updating method by setting the position and angle of the car directly using the values from the file.

## Results

|  |  |
| --- | --- |
|  |  |
| Fig. 7 – Animation recording | Fig. 8 – Animation playing |

# Stereo Vision

With the objective to run the application in a virtual reality setting we created stereo cameras that are able to render the application in a way that’s compatible with Google cardboard glasses.

## Method

The stereo camera works by dividing the screen vertically in half, one side for each eye. Then we invoke the lookAt() and Frustum() methods once 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 the accelerometer of the mobile device.

## Method

To do this we rely on the ‘deviceorientation’ event to capture changes in the orientation of the mobile device. We defined a handler for the event using the addEventListener() method.

The handler is a function that gets the orientation of the device in the three axes and updates our application with the angles obtained. We then use the angles to rotate our camera during the rendering of the scene.

## Results

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

# Extras

## 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

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 required techniques. During the development of this project we acquired new skills and had the opportunity to learn how to use WebGL and JavaScript to bring interactive graphical applications to the web.