Homework 1

Interactive Graphics

Alessandro Giannetti 1592006

2019-04-28

1 Introduction

This report describes the process of building a simple 3D interactive graphics project using HTML, JavaScript and WebGL. The application began with the blue face of the cube. In the next section, we will describe how the solution to various tasks is implemented.

2 Solution

the first feature to implement is to add the *viewer position, a projection and all must be controllable through buttons/sliders.*

In order to create buttons and sliders we simply add the corresponding HTML tag (`button, input type="range"`), taking care to add a unique id inside the tag. In this way it will be possible to find the value in the JS file through the function: `document.getElementById("ID").onclick`.
After this, we want to create the ModelView and the Projection matrices. This is done in the JS file because the main advantage is that computing the transformations for every vertex is a huge waste of time and computation. The Model-view matrix was implemented using the `lookAt` and `Perspective` functions defined in the MV.js file.

For the second point we have to *implement scaling and translation.*

Like the first point, we have to implement in the js file the functions "scalem" and "translate" defined in the MV file. We also add sliders in the HTML file and binding them like the previous point in the JS, for controlling scaling and translation. Translation, and Scaling of the cube were implemented by building Homogenous transformation matrices using the simple translate, and scalem JavaScript functions that are found in the MV.js file and multiplying set matrices to the Model-view matrix using the mult function inside the JavaScript application.These transformations are obtained when we multiply the matrices in the HTML file in a specific order to not have some side effects in the transformations.

After this, it's asked to *separate the canvas into two parts*, wherein the first part there is the cube that has a *perspective projection* and the second has an *orthographic projection.*

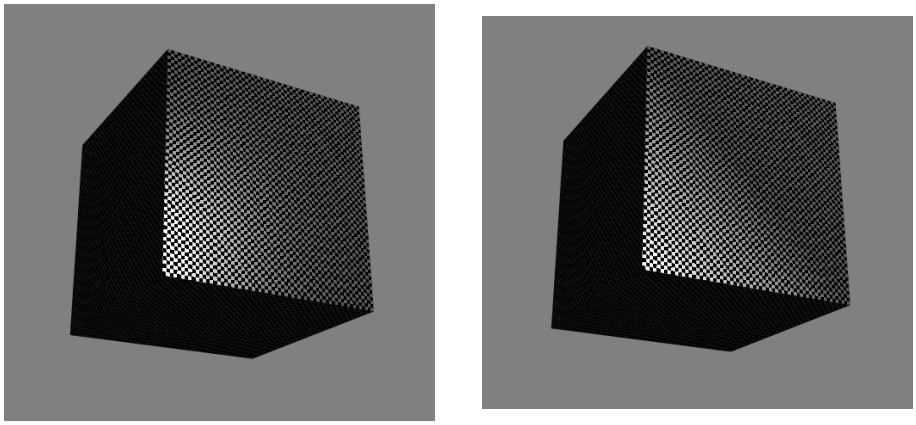
We divided the canvas through the function "viewport" and "scissor". I created a general "renderscene" where it creates the divided scene and all the common features that the 2 cubes have, like the point of view or the translation matrix. After that, we define the width, height, displayWidth and displayHeight that we will use to cut the canvas. For this reason, we define 2 blocks, one on the left, the other on the right. In this way, in the left block, we will define the perspective projection, in the right block the orthogonal projection, using the "ortho" function defined in the MV file. Moreover, two sliders were introduced to control the viewing volume, specifically the location of the Near and Far planes. The same 2 sliders work for both kinds of projections.

In the next task is required to *implement a light source, with the relative Phong and Gouraud visualization.*

We have to define the normal for each vertex by calculating the product of 2 vectors that lie on a face and normalizing it. The main difference between both models is that the Gouraud shading computes the color between each vertex and interpolates it while the Phong shading model interpolates the normal across the triangle and compute the color per fragment (or pixel). For change it in runtime, we defined a button that changes the shading.

In the last task we have to *add a procedural texture.*

For the material, we define in the JS, the values that specify the properties of shine, diffuse and specular. In our cube, we define the properties of the Obsidian. The last task is to add texture to the cube. I defined a checkboard cube through a checks variable, after that we define two variables that define the checkboard pattern and we configure the texture. After that, we pass all this to the HTML file and compile it in the fragment shader.



(a) Phong Shading

(b) Gouraud Shading

Figure 1: The difference between the Gouraud Shading and the Phong Shading model is very clear when we bring the cube closer to the light source.

3 Conclusion

This solution has the main advantage of being flexible and efficient. It correctly and efficiently implements different kinds of shading models, different kinds of projections, and allows one to having full control over the position, orientation, and the point of view of a 3D cube in a simple and efficient manner. The application was improved introducing a CSS stylesheet that organizes the elements on the webpage and gives them a more modern look.