Interactive Graphics - Homework 1

A.Y. 2018/2019

Marco Costa 1691388

28 April 2019

1 Introduction

This project consists in the implementation of a textured cube in a 3D space, using two projections (orthographic and perspective). Moreover, it's necessary to apply some transformations and two shading models (Gouraud and Phong). Some parameters, for projections and cube transformations, can be modified at runtime through some sliders and buttons in HTML index page.

2 Solution

2.1 Point 1

In the first point we have to implement a viewer position. It's possible to do it through the *eye* variable. To assign a value to this variable, we need two parameters, *theta* and *phi*, which are the angles. After that, it's possible to compute *modelViewMatrix* using *lookAt* function, which in turn uses three variables: *eye* (seen before), *at* and *up*. The viewing volume we can use two variables, *far* and *near*. The viewer position instead is given by the combination of *theta* and *phi* values. To determine, instead, the viewing volume I use zNear and zFar variables. For this point, I used a perspective projection. The variables *far*, *near*, *theta* and *phi* can be modified at runtime trought dedicated slider. The advantage is that the user can choose the viewer position simply modifying the values of these variables.

```
//Camera
eye = vec3(radius * Math.sin(phi), radius * Math.sin(theta), radius * Math.cos(phi));
```

```
//Camera position
var eye;
const at = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);
```

2.2 Point 2

In order to solve this point, I had to implement two transformations: *Scaling* and *Translation*. For the first transformation, I had to implement an "uniform" transformation; in this way the scaling factor *scaling* must be equal for each axis. I set the default value to 0.5, but it can be modified trought a slider. To perform this transformation it's necessary to use scaling matrix, a 4x4 diagonal matrix.

$$\begin{bmatrix} scaling & 0 & 0 & 0 \\ 0 & scaling & 0 & 0 \\ 0 & 0 & scaling & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling is the scaling factor.

For the second transformation, instead, we have three variables: *translX*, *translY*, *translZ*. In this way, it's possible to perform the traslation for each axis modifying the three variables independently trought dedicated sliders in order to translate the cube on the different axis. We have to use the translation Matrix:

$$\begin{bmatrix} 1 & 0 & 0 & translX \\ 0 & 1 & 0 & translY \\ 0 & 0 & 1 & translZ \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
    var translationMatrix = [
        1.0, 0.0, 0.0, 0.0,
        0.0, 1.0, 0.0, 0.0,
        0.0, 0.0, 1.0, 0.0,
        translX, translY, translZ, 1.0
    ];

    var scalingMatrix = [
        scaling, 0.0, 0.0, 0.0,
        0.0, scaling, 0.0, 0.0,
        0.0, 0.0, scaling, 0.0,
        0.0, 0.0, 0.0, 1.0
    ];
```

2.3 Point 3

In the third point we have to add another projection (orthographic). It's possible to do it simply creating another projection matrix, using the dedicated function *ortho* of MV library.

2.4 Point 4

In order to split the window vertically into two parts we can use the *gl.scissor()* function, that permits to modify only a specific area of the canvas. To activate the function, it's necessary to activate *gl.enable(gl.SCISSOR_ TEST)* mask.

```
//Divide canvas in two parts
const width = gl.canvas.width;
const height = gl.canvas.height;
const displayWidth = gl.canvas.clientWidth;
const displayHeight = gl.canvas.clientHeight;
const dispWidth = displayWidth / 2;
const dispHeight = displayHeight;
const aspect = dispWidth/dispHeight;

gl.enable(gl.SCISSOR_TEST);

//Render left part
gl.clearColor(0.1, 0.1, 0.1, 1);
gl.scissor(0,  0, width/2, height);
gl.viewport(0,  0, width/2, height);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.drawArrays(gl.TRIANGLES, 0, numVertices);

//Render right part
projectionMatrix = perspective(fovy, aspect, near, far);
gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));

gl.clearColor(0.2, 0.2, 0.2, 1);
gl.scissor(width/2, 0, width/2, height);
gl.viewport(width/2, 0, width/2, height);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.drawArrays(gl.TRIANGLES, 0, numVertices);
```

2.5 Point 5

To implement this point, I had to introduce a light source. It's necessary to define the position of light source, the ambient light and so on. Then it's necessary to define the material property about the absorption and the reflection of the light.

```
//light
var lightPosition = vec4(2.0, 2.0, 2.0, 0.0);
var lightAmbient = vec4(0.1, 0.1, 0.1, 1.0);
var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0);

//material
var materialAmbient = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse = vec4( 0.8, 0.4, 0.2, 1.0);
var materialSpecular = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 20.0;
```

After that it computes the interaction between the matter and the light in this instructions:

```
var ambientProduct = mult(lightAmbient, materialAmbient);
var diffuseProduct = mult(lightDiffuse, materialDiffuse);
var specularProduct = mult(lightSpecular, materialSpecular);
```

Finally i had to implementat a normal for each vertex of the triangle of the cube's faces. To do this I simply modified the *quad()* function as follow:

```
function quad(a, b, c, d) {
    var t1 = subtract(vertices[b], vertices[a]);
    var t2 = subtract(vertices[c], vertices[b]);
    var normal = cross(t1, t2);
    var normal = vec3(normal);

    pointsArray.push(vertices[a]);
    texCoordsArray.push(texCoord[0]);
    normalsArray.push(normal);

    pointsArray.push(vertices[b]);
    texCoordsArray.push(texCoord[1]);
    normalsArray.push(normal);

    pointsArray.push(vertices[c]);
    texCoordsArray.push(texCoord[2]);
    normalsArray.push(normal);

    pointsArray.push(vertices[a]);
    texCoordsArray.push(texCoord[0]);
    normalsArray.push(normal);

    pointsArray.push(vertices[c]);
    texCoordsArray.push(texCoord[2]);
    normalsArray.push(normal);

    pointsArray.push(vertices[d]);
    texCoordsArray.push(texCoord[3]);
    normalsArray.push(normal);
}
```

2.6 Point 6

2.7 Point 7