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.

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

```
57    //Camera position
58    var eye;
59    const at = vec3(0.0, 0.0, 0.0);
60    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:

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

```
274    var translationMatrix = [
275        1.0, 0.0, 0.0, 0.0,
276        0.0, 1.0, 0.0, 0.0,
277        0.0, 0.0, 1.0, 0.0,
278        translX, translY, translZ, 1.0
279    ];
280
281    var scalingMatrix = [
282        scaling, 0.0, 0.0, 0.0,
283        0.0, scaling, 0.0, 0.0,
284        0.0, 0.0, scaling, 0.0,
285        0.0, 0.0, 0.0, 1.0
286    ];
```

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.

```
304    //Divide canvas in two parts
305    const width = gl.canvas.width;
306    const height = gl.canvas.height;
307    const displayWidth = gl.canvas.clientWidth;
308    const displayHeight = gl.canvas.clientHeight;
309    const dispWidth = displayWidth / 2;
310    const dispHeight = displayHeight;
311    const aspect = dispWidth/dispHeight;
312
313    gl.enable(gl.SCISSOR_TEST);
314
315    //Render left part
316    gl.clearColor(0.1, 0.1, 0.1, 1);
317    gl.scissor(0,  0, width/2, height);
318    gl.viewport(0,  0, width/2, height);
319    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
320    gl.drawArrays(gl.TRIANGLES, 0, numVertices);
321
322    //Render right part
323    projectionMatrix = perspective(fovy, aspect, near, far);
324    gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));
325
326    gl.clearColor(0.2, 0.2, 0.2, 1);
327    gl.scissor(width/2, 0, width/2, height);
328    gl.viewport(width/2, 0, width/2, height);
329    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
330    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.

```
62  //light
63  var lightPosition = vec4(2.0, 2.0, 2.0, 0.0);
64  var lightAmbient = vec4(0.1, 0.1, 0.1, 1.0);
65  var lightDiffuse = vec4(1.0, 1.0, 1.0, 1.0);
66  var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0);
67
68  //material
69  var materialAmbient = vec4( 1.0, 0.0, 1.0, 1.0 );
70  var materialDiffuse = vec4( 0.8, 0.4, 0.2, 1.0);
71  var materialSpecular = vec4( 1.0, 0.8, 0.0, 1.0 );
72  var materialShininess = 20.0;
```

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

```
195    var ambientProduct = mult(lightAmbient, materialAmbient);
196    var diffuseProduct = mult(lightDiffuse, materialDiffuse);
197    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:

```
113  function quad(a, b, c, d) {
114      var t1 = subtract(vertices[b], vertices[a]);
115      var t2 = subtract(vertices[c], vertices[b]);
116      var normal = cross(t1, t2);
117      var normal = vec3(normal);
118
119      pointsArray.push(vertices[a]);
120      texCoordsArray.push(texCoord[0]);
121      normalsArray.push(normal);
122
123      pointsArray.push(vertices[b]);
124      texCoordsArray.push(texCoord[1]);
125      normalsArray.push(normal);
126
127      pointsArray.push(vertices[c]);
128      texCoordsArray.push(texCoord[2]);
129      normalsArray.push(normal);
130
131      pointsArray.push(vertices[a]);
132      texCoordsArray.push(texCoord[0]);
133      normalsArray.push(normal);
134
135      pointsArray.push(vertices[c]);
136      texCoordsArray.push(texCoord[2]);
137      normalsArray.push(normal);
138
139      pointsArray.push(vertices[d]);
140      texCoordsArray.push(texCoord[3]);
141      normalsArray.push(normal);
142  }
```

2.6   Point 6

I made a boolean variable *shadingModel* which let to change the shading mo-
del simply clicking on a button. It can be modifiet at runtime thanks to this
function:

```
248      document.getElementById("ButtonShadingModel").onclick = function() {
249          shadingModel = !shadingModel;
250          if (shadingModel) {
251              document.getElementById('currentModel').innerText = "Current Shading Model: Phong";
252          }
253          else {
254              document.getElementById('currentModel').innerText = "Current Shading Model: Gouraud";
255          }
256      };
```

After that, I modified the *vertex-shader* and the *fragment-shader* by inserting
an if. In this way it's possible to change the shading model trought the button.

2.7   Point 7

This last point was quite simple. I simply added the texture in the javascript
file as follow:

```
258      //Texture
259      var tBuffer = gl.createBuffer();
260      gl.bindBuffer(gl.ARRAY_BUFFER, tBuffer);
261      gl.bufferData(gl.ARRAY_BUFFER, flatten(texCoordsArray), gl.STATIC_DRAW);
262
263      var vTexCoord = gl.getAttribLocation(program, "vTexCoord");
264      gl.vertexAttribPointer(vTexCoord, 2, gl.FLOAT, false, 0, 0);
265      gl.enableVertexAttribArray(vTexCoord);
266
267      configureTexture(image2);
```

4

At the end I modified the *vertex-shader* and the *fragment-shader* obtaining the following result:

```html
<script id="vertex-shader" type="x-shader/x-vertex">

    attribute vec4 vPosition;
    varying vec4 fColor;

    uniform mat4 modelViewMatrix;
    uniform mat4 projectionMatrix;
    uniform mat4 scalingMatrix;
    uniform mat4 translationMatrix;

    //light and material
    attribute vec3 vNormal;
    uniform vec4 lightPosition;
    uniform vec4 vAmbientProduct;
    uniform vec4 vDiffuseProduct;
    uniform vec4 vSpecularProduct;
    uniform float vShininess;

    uniform bool shadingModel;
    varying vec3 fN, fL, fE;

    //texture
    attribute vec2 vTexCoord;
    varying vec2 fTexCoord;

    void main() {
        vec3 pos = (modelViewMatrix * translationMatrix * scalingMatrix * vPosition).xyz;
        vec3 L = normalize( lightPosition.xyz - pos );
        vec3 E = -normalize( pos );
        vec3 H = normalize( L + E );
        vec3 N = normalize((translationMatrix*scalingMatrix* vec4(vNormal, 0)).xyz);

        //Switch Model
        if (!shadingModel) {
            vec4 ambient = vAmbientProduct;

            float Kd = max(dot(L, N), 0.0);
            vec4  diffuse = Kd*vDiffuseProduct;

            float Ks = pow(max(dot(N, H), 0.0), vShininess);
            vec4  specular = Ks * vSpecularProduct;

            if( dot(L, N) < 0.0 ) specular = vec4(0.0, 0.0, 0.0, 1.0);

            fColor = ambient + diffuse + specular;
            fColor.a = 1.0;
        }
        else {
            fL = L;
            fN = N;
            fE = E;
        }

        fTexCoord = vTexCoord;
        gl_Position = projectionMatrix * modelViewMatrix * translationMatrix * scalingMatrix * vPosition;
    }
</script>
```

```html
<script id="fragment-shader" type="x-shader/x-fragment">

    precision mediump float;
    varying vec4 fColor;

    uniform vec4 fAmbientProduct;
    uniform vec4 fDiffuseProduct;
    uniform vec4 fSpecularProduct;
    uniform float fShininess;
    uniform bool shadingModel;
    varying vec3 fN, fL, fE;

    varying vec2 fTexCoord;
    uniform sampler2D texture;

    void main(){
        if (!shadingModel) {
            gl_FragColor = fColor * texture2D( texture, fTexCoord );
        }
        else {
            vec4 fColorF;
            vec3 H = normalize(fL + fE);
            vec4 ambient = fAmbientProduct;

            float Kd = max(dot(fL, fN), 0.0);
            vec4 diffuse = Kd * fDiffuseProduct;

            float Ks = pow(max(dot(fN, H), 0.0), fShininess);
            vec4 specular = Ks * fSpecularProduct;

            if (dot(fL, fN) < 0.0) specular = vec4(0.0, 0.0, 0.0, 1.0);

            fColorF = ambient + diffuse + specular;
            fColorF.a = 1.0;

            gl_FragColor = fColorF*texture2D( texture, fTexCoord );
        }
    }
</script>
```

3   Final result

Near:
0.1

Far:
5

Theta:
45

Phi:
45

Scaling:
0.5

Fov:
90

Translate on X:
0

Translate on Y:
0

Translate on Z:
0

Current Shading Model: Phong

Change Shading Model