“La Sapienza” University of Rome
Faculty of information engineering, information technology and statistics
Department of informatics, automation and control engineering
"ANTONIO RUBERTI"
Degree program: Artificial Intelligence and Robotics

HOMEWORK №1

Student: OREL, Egor
Matricola: 1836231
Cloud storage containing the solution: https://yadi.sk/d/szn3cptgBZW2cw

Teacher: SCHAERF, Marco

Rome, 2019

Exercise: to apply following actions to the given cube:

1. Add the viewer position (your choice), a projection (your choice) and compute the ModelView and Projection matrices in the Javascript application. The viewer position and viewing volume should be controllable with buttons, sliders or menus.

2. Include a scaling (uniform, all parameters have the same value) and a translation Matrix and control them with sliders.

3. Define an orthographic projection with the planes near and far controlled by sliders.

4. Split the window vertically into two parts. One shows the ortographic projection defined above, the second uses a perspective projection. The slider for near and far should work for both projections. Points 5 to 7 use the splitted window with two different projections

5. Introduce a light source, replace the colors by the properties of the material (your choice) and assign to each vertex a normal.

6. Implement both the Gouraud and the Phong shading models, with a button switching between them.

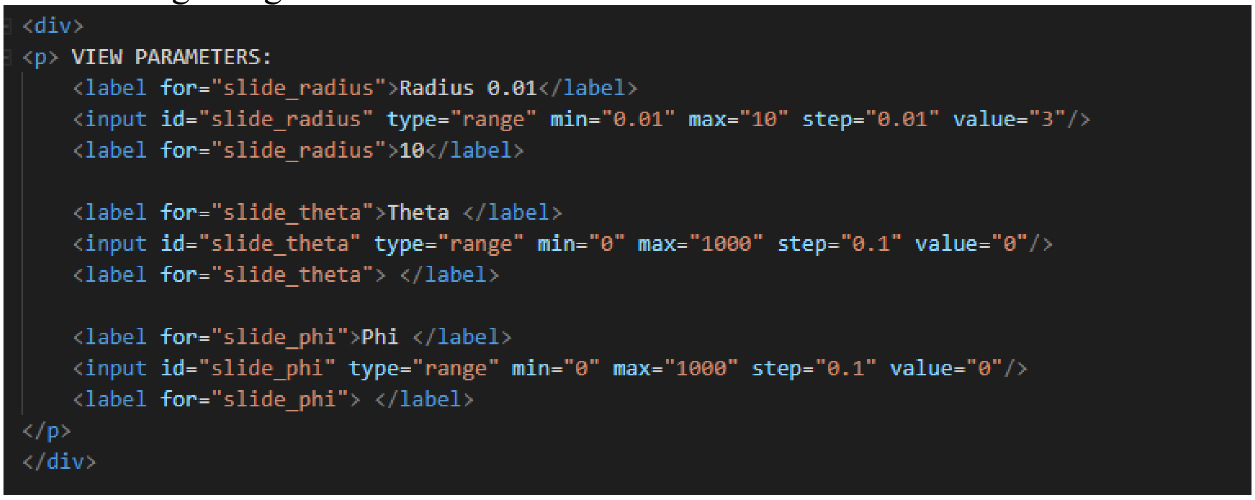
7. Add a procedural texture (your choice) on each face, with the pixel color a combination of the color computed using the lighting model and the texture

Results

All results will be given according to numbers of exercises and divided on two parts: about important exchanges in *.html-file and about *.js-file

1, 2)

<HTML> Sliders and buttons of blocks "View Parameters", "Sliding" and "Translating along axes" were introduced.

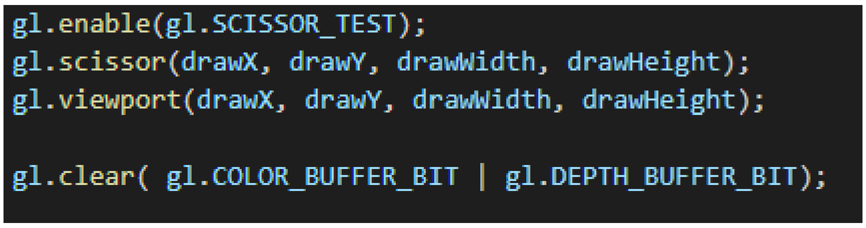


```
<div>
<p> VIEW PARAMETERS:
    <label for="slide_radius">Radius 0.01</label>
    <input id="slide_radius" type="range" min="0.01" max="10" step="0.01" value="3"/>
    <label for="slide_radius">10</label>

    <label for="slide_theta">Theta </label>
    <input id="slide_theta" type="range" min="0" max="1000" step="0.1" value="0"/>
    <label for="slide_theta"> </label>

    <label for="slide_phi">Phi </label>
    <input id="slide_phi" type="range" min="0" max="1000" step="0.1" value="0"/>
    <label for="slide_phi"> </label>
</p>
</div>
```

<JS> The value of each slider was used for building the ModelView matrix and Projections. Improvements are made inside of the render() function.

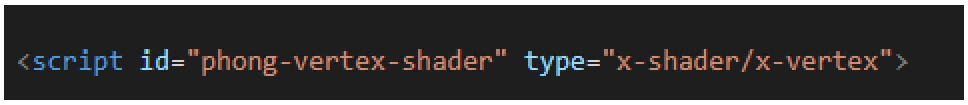
3)
<HTML> Sliders and buttons of blocks “Projection Parameters” were introduced.
<JS> Projection matrices (Orthographic and Perspective) are implemented separately for left and right windows (paragraph 4 of the exercise) using predefined library MV.js.

4)
<HTML> No changes
<JS> Splitting of the window was introduced by cutting the canvas for two parts using gl.scissor() function. It was applied twice inside the render() function as follows at the screenshot:



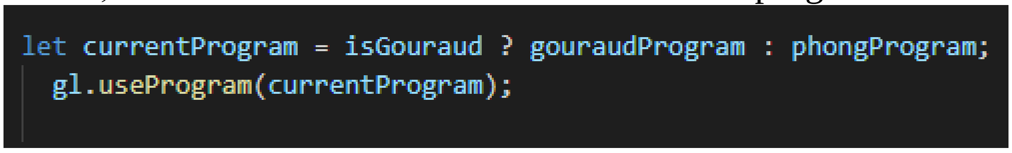
```
gl.enable(gl.SCISSOR_TEST);
gl.scissor(drawX, drawY, drawWidth, drawHeight);
gl.viewport(drawX, drawY, drawWidth, drawHeight);

gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

5, 6)
<HTML> Solving of the problem needs implementation inside of fragment shaders and vertex shaders. Requires multiplication of previous data of the model to new one, which obtained with the introduction of material and light properties, light position. Each of Phong and Gourand light models compile with different vertex and fragment shaders that defined as “phong-vertex-shader”, “phong-fragment-shader” and “gouraud-vertex-shader”, “gouraud-fragment-shader”. A shading button was defined:



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

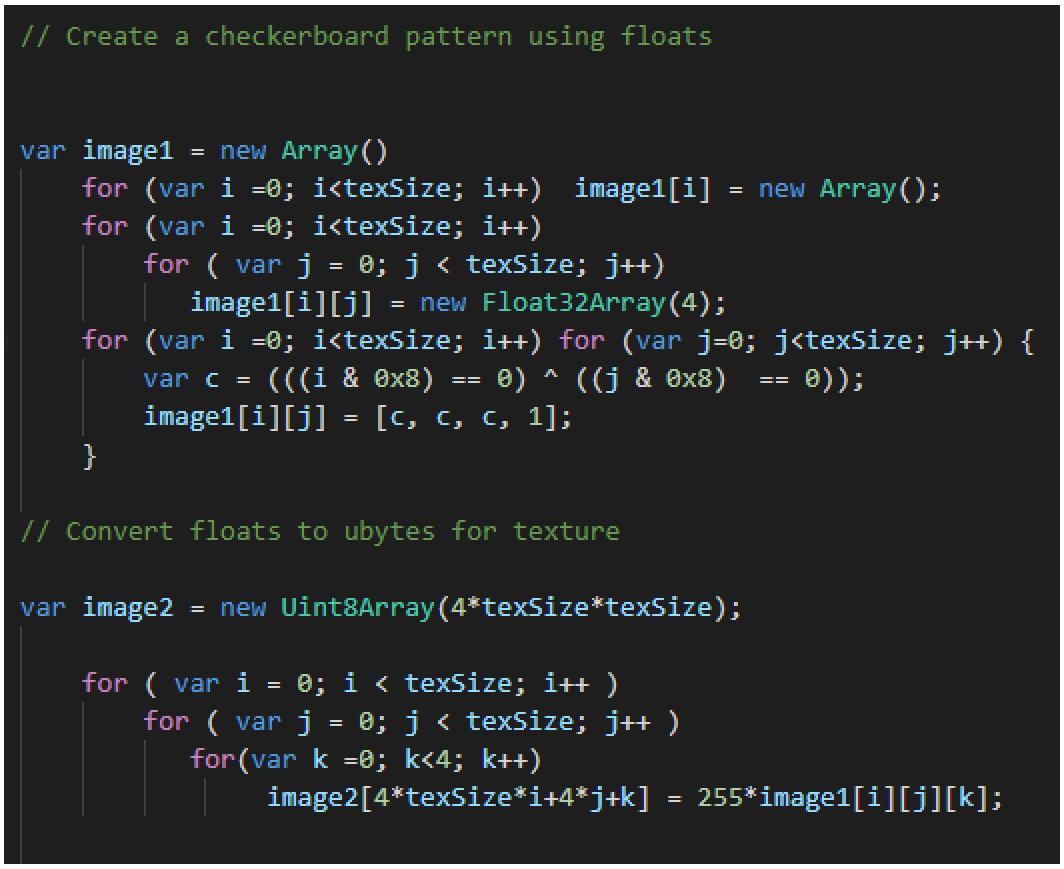
<JS> Since Phong and Gouraud model use different approaches for changing pixel color values, two separate WebGL programs was used in the js script file – phongProgram and gouraudProgram. Then, on each iteration, according to current shading model, WebGL’s method called to set the current program as follows:



```
let currentProgram = isGouraud ? gouraudProgram : phongProgram;
  gl.useProgram(currentProgram);
```

and all assignment of projectionMatrix, modelViewMatrix, material and light characteristics are assigned to variables of current program

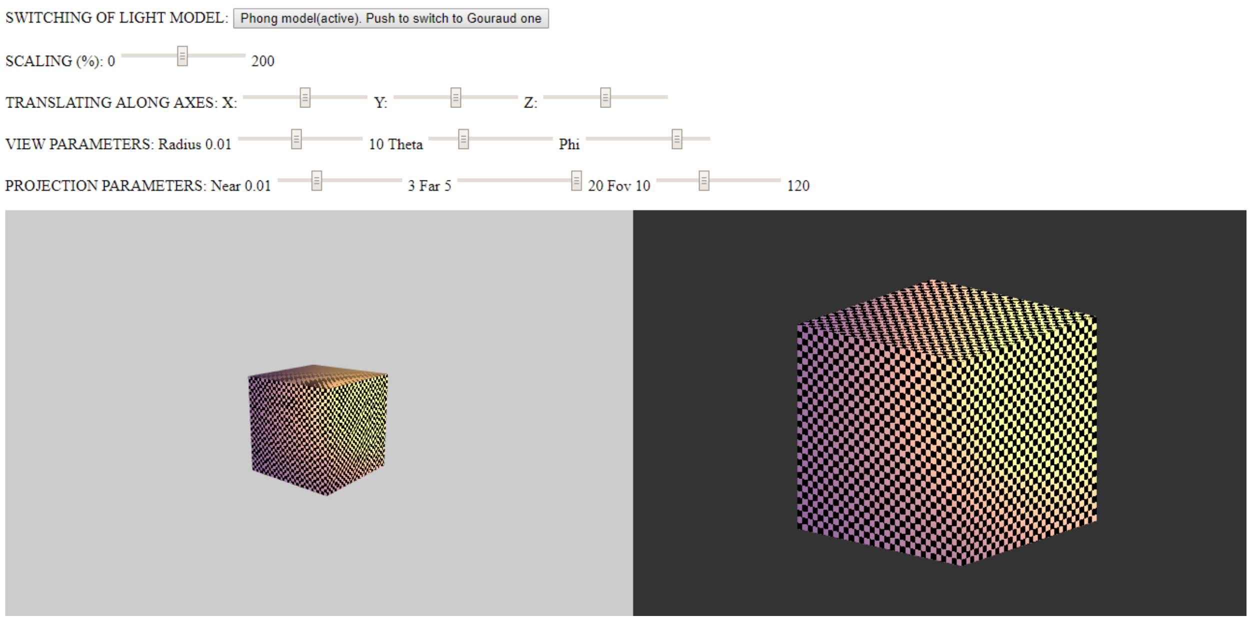
7)
<HTML> For vertex shaders of both Phong and Gourand models calculate data of the mapping from the surface and multiply it to gl_Position. In the fragment shader just was applied a texture 2D sampler.
<JS> One of the most important things to implement for texture mapping is the image. In this homework a checkerboard structure was implemented:



```
// Create a checkerboard pattern using floats


var image1 = new Array()
    for (var i =0; i<texSize; i++)  image1[i] = new Array();
    for (var i =0; i<texSize; i++)
        for ( var j = 0; j < texSize; j++)
           image1[i][j] = new Float32Array(4);
    for (var i =0; i<texSize; i++) for (var j=0; j<texSize; j++) {
        var c = (((i & 0x8) == 0) ^ ((j & 0x8)  == 0));
        image1[i][j] = [c, c, c, 1];
    }

// Convert floats to ubytes for texture

var image2 = new Uint8Array(4*texSize*texSize);

    for ( var i = 0; i < texSize; i++ )
        for ( var j = 0; j < texSize; j++ )
           for(var k =0; k<4; k++)
                image2[4*texSize*i+4*j+k] = 255*image1[i][j][k];
```

Then to push all these pictures to every side of the cube in the quad() function. And configure texture in the render() function.

Screenshot of the final result:



SWITCHING OF LIGHT MODEL: Phong model(active). Push to switch to Gouraud one

SCALING (%): 0 200

TRANSLATING ALONG AXES: X: Y: Z:

VIEW PARAMETERS: Radius 0.01 10 Theta Phi

PROJECTION PARAMETERS: Near 0.01 3 Far 5 20 Fov 10 120