МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ЗВІТ

ПРО ЛАБОРАТОРНУ РОБОТУ №6

ТЕМА: «ЗАТІНЕННЯ ПОВЕРХОНЬ»

Виконав:
Студент групи ІП-15
Мєшков А.І.

Перевірив:
доц. каф. ІПІ
Родіонов П.Ю.

Київ 2023

ХІД РОБОТИ

1. Написати комп’ютерну програму, що рисує фігуру довільної форми.

2. Застосувати до фігури затінення.

3. Написати коментарі до логічних блоків програмного коду.

4. Скласти та завантажити на платформу Google Classroom звіт про виконану лабораторну роботу.

Лістинг 1 - Програмний код у файлі HTML

```
<!DOCTYPE <!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <title>Лабораторна робота №5</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <style>
        body {
            margin: 0;
            background:  #001eff;
        }

        canvas {
            width: 100%;
            height: 100%;
        }
    </style>
    <script src="script.js" defer></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-
matrix/2.4.0/gl-matrix.js"></script>
</head>
<body>

    <canvas id="myCanvas"></canvas>

</body>
</html>
```

Лістинг 2 - Програмний код у файлі JavaScript

```
// Get the canvas element and its WebGL rendering context
const canvas = document.getElementById('myCanvas');
const gl = canvas.getContext('webgl');

// Check if WebGL is supported
if (!gl) {
    throw new Error('WebGL not supported');
}

// Define vertex data for a cube
const vertexData = [

    0.5, 0.5, 0.5,
    0.5, -0.5, 0.5,
    -0.5, 0.5, 0.5,
```

```
    -0.5, 0.5, 0.5,
    0.5, -0.5, 0.5,
    -0.5, -0.5, 0.5,

    -0.5, 0.5, 0.5,
    -0.5, -0.5, 0.5,
    -0.5, 0.5, -0.5,
    -0.5, 0.5, -0.5,
    -0.5, -0.5, 0.5,
    -0.5, -0.5, -0.5,

    -0.5, 0.5, -0.5,
    -0.5, -0.5, -0.5,
    0.5, 0.5, -0.5,
    0.5, 0.5, -0.5,
    -0.5, -0.5, -0.5,
    0.5, -0.5, -0.5,

    0.5, 0.5, -0.5,
    0.5, -0.5, -0.5,
    0.5, 0.5, 0.5,
    0.5, 0.5, 0.5,
    0.5, -0.5, -0.5,
    0.5, -0.5, 0.5,

    0.5, 0.5, 0.5,
    0.5, 0.5, -0.5,
    -0.5, 0.5, 0.5,
    -0.5, 0.5, 0.5,
    0.5, 0.5, -0.5,
    -0.5, 0.5, -0.5,

    0.5, -0.5, 0.5,
    0.5, -0.5, -0.5,
    -0.5, -0.5, 0.5,
    -0.5, -0.5, 0.5,
    0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5,
];

// Function to repeat a pattern 'n' times
function repeat(n, pattern) {
    return [...Array(n)].reduce(sum => sum.concat(pattern), []);
}

// Generate normal data for the cube
const normalData = [
    ...repeat(6, [0, 0, 1]),
    ...repeat(6, [-1, 0, 0]),
    ...repeat(6, [0, 0, -1]),
    ...repeat(6, [1, 0, 0]),
    ...repeat(6, [0, 1, 0]),
    ...repeat(6, [0, -1, 0]),
];

// Create buffers and fill them with vertex and normal data
const positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexData),
gl.STATIC_DRAW);

const normalBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, normalBuffer);
```

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(normalData),
gl.STATIC_DRAW);

// Function to create and compile the shader program
function shaderProgram() {
    const vertexShader = gl.createShader(gl.VERTEX_SHADER);
    gl.shaderSource(vertexShader, `
        precision mediump float;
        attribute vec3 position;
        attribute vec3 normal;
        varying vec3 normalInterp;
        varying vec3 vertPos;

        uniform float Ka;   // Ambient reflection coefficient
        uniform float Kd;   // Diffuse reflection coefficient
        uniform float Ks;   // Specular reflection coefficient
        uniform float shininessVal ; // Shininess
        // Material color
        uniform vec3 ambientColor;
        uniform vec3 diffuseColor;
        uniform vec3 specularColor;
        uniform vec3 lightPos; // Light position
        varying vec4 color; //color

        uniform mat4 matrix;
        uniform mat4 normalMatrix;

        void main() {
            vec4 vertPos4 = matrix * vec4(position, 1.0);
            vertPos = vec3(vertPos4) / vertPos4.w;
            normalInterp = vec3(normalMatrix * vec4(normal, 0));
            gl_Position = matrix * vec4(position, 1);

            vec3 N = normalize(normalInterp);
            vec3 L = normalize(lightPos - vertPos);
            // Lambert's cosine law
            float lambertian = max(dot(N, L), 0.0);
            float specular = 0.0;
            if(lambertian > 0.0) {
              vec3 R = reflect(-L, N);      // Reflected light vector
              vec3 V = normalize(-vertPos); // Vector to viewer
              // Compute the specular term
              float specAngle = max(dot(R, V), 0.0);
              specular = pow(specAngle, shininessVal);
            }


            color = vec4(Ka * ambientColor +
                Kd * lambertian * diffuseColor +
                Ks * specular * specularColor, 1.0);

        }
    `);
    gl.compileShader(vertexShader);

    if (!gl.getShaderParameter(vertexShader, gl.COMPILE_STATUS)) {
        console.error(gl.getShaderInfoLog(vertexShader));
        throw new Error('Failed to compile vertex shader');
    }

    const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
    gl.shaderSource(fragmentShader, `
        precision mediump float;
```

```
        varying vec4 color;

        void main() {
            gl_FragColor = color;
        }
    `);
    gl.compileShader(fragmentShader);

    if (!gl.getShaderParameter(fragmentShader, gl.COMPILE_STATUS)) {
        console.error(gl.getShaderInfoLog(fragmentShader));
        throw new Error('Failed to compile fragment shader');
    }

    const program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);

    gl.linkProgram(program);

    if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
        console.error(gl.getProgramInfoLog(program));
        throw new Error('Failed to link program');
    }

    const positionLocation = gl.getAttribLocation(program, 'position');
    gl.enableVertexAttribArray(positionLocation);
    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
    gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 0, 0);

    const normalLocation = gl.getAttribLocation(program, 'normal');
    gl.enableVertexAttribArray(normalLocation);
    gl.bindBuffer(gl.ARRAY_BUFFER, normalBuffer);
    gl.vertexAttribPointer(normalLocation, 3, gl.FLOAT, false, 0, 0);

    gl.useProgram(program);
    gl.enable(gl.DEPTH_TEST);

    return {
        program,
        uniformLocations: {
            matrix: gl.getUniformLocation(program, 'matrix'),
            normalMatrix: gl.getUniformLocation(program, 'normalMatrix'),
        },
    };
}

// Initialize the shader program and get uniform locations
const { program, uniformLocations } = shaderProgram();

// Initialize matrices for transformations
const modelMatrix = mat4.create();
const viewMatrix = mat4.create();
const projectionMatrix = mat4.create();
const mvMatrix = mat4.create();
const mvpMatrix = mat4.create();
const normalMatrix = mat4.create();

// Set perspective projection matrix
mat4.perspective(projectionMatrix,
    75 * Math.PI / 180,
    canvas.width / canvas.height,
    1e-4,
    1e4
);
```

```
// Translate and invert the view matrix
mat4.translate(viewMatrix, viewMatrix, [0, 0.1, 2]);
mat4.invert(viewMatrix, viewMatrix);

// Function to render the scene
function render() {
    // requestAnimationFrame(render);

     // Rotate the model matrix
    mat4.rotateY(modelMatrix, modelMatrix, 0.9);
    mat4.rotateX(modelMatrix, modelMatrix, 0.5);

    // Combine matrices for MVP transformation
    mat4.multiply(mvMatrix, viewMatrix, modelMatrix);
    mat4.multiply(mvpMatrix, projectionMatrix, mvMatrix);

    // Calculate and set the normal matrix
    mat4.invert(normalMatrix, mvMatrix);
    mat4.transpose(normalMatrix, normalMatrix);

    // Set uniforms and clear the canvas
    gl.uniformMatrix4fv(uniformLocations.normalMatrix, false,
normalMatrix);
    gl.uniformMatrix4fv(uniformLocations.matrix, false, mvpMatrix);
    gl.uniform1f(gl.getUniformLocation(program, 'Ka'), 0.5);
    gl.uniform1f(gl.getUniformLocation(program, 'Kd'), 1.0);
    gl.uniform1f(gl.getUniformLocation(program, 'Ks'), 1.0);
    gl.uniform1f(gl.getUniformLocation(program, 'shininessVal'), 3.0);
    gl.uniform3fv(gl.getUniformLocation(program, 'ambientColor'), [0.2,
0.09, 0]);
    gl.uniform3fv(gl.getUniformLocation(program, 'diffuseColor'), [0.8,
0.4, 0]);
    gl.uniform3fv(gl.getUniformLocation(program, 'specularColor'), [1, 1,
1]);
    gl.uniform3fv(gl.getUniformLocation(program, 'lightPos'), [0.3, 0,
1.5]);

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

    // Draw the cube
    gl.drawArrays(gl.TRIANGLES, 0, vertexData.length / 3);
}

// Start the rendering loop
render();
```

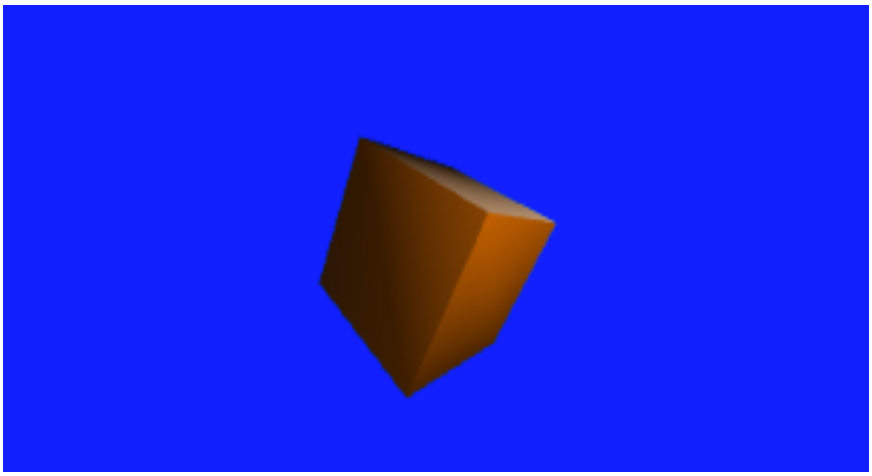

Рисунок 1 – Результат виконання завдання

ВИСНОВОК

У ході виконання лабораторної роботи було досягнуто основної мети - застосування теоретичних знань про освітлення та затінення в WebGL і реалізація затінення Фонга. Основні кроки та результати роботи можуть бути визначені наступним чином:

1. Розробка комп’ютерної програми:
   - Написана програма рисує куб у тривимірному просторі за допомогою WebGL.
   - Використовується GLSL для реалізації затінення Фонга, що дозволяє моделювати реалістичні властивості світла на поверхнях фігури.
2. Застосування затінення:
   - Затінення Фонга впроваджено для кожного вершинного елементу фігури, що дозволяє враховувати освітленість та тіні при відображенні.

Загалом, виконана лабораторна робота сприяла поглибленню знань з області графічного програмування, а саме в застосуванні освітлення та затінення для досягнення більш реалістичного візуального ефекту у віртуальних об'єктах.