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

ЗВІТ
ПРО ЛАБОРАТОРНУ РОБОТУ №5
ТЕМА: « ОСВІТЛЕННЯ ТА ЗАТІНЕННЯ »

Виконав:
Студент групи ІП-22
Підпанюк В.А.

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

Київ 2024

Лабораторна робота №5

ОСВІТЛЕННЯ ТА ЗАТІНЕННЯ

Мета: отримати практичні навички щодо роботи з освітленням графічної сцени на основі програмного інтерфейсу WebGL.

Завдання:

1. Створити програму WebGL та використати фоновий колір.
2. Створити та розмістити у графічний сцені об’єкт довільної форми у перспективній проекції.
3. Реалізувати освітлення графічної сцени за допомогою моделі віддзеркалення Фонга
4. Внести зміни в освітлення сцени.
5. Скласти звіт про виконану роботу.

1. Створення сцени із фоновим кольором

У даному пункті створюємо канвас та змінюємо його колір (рис. 1.1).

рис. 1.1. Канвас зі зміненим кольором фону

2. Створення фігури на сцені

У даному пункті створюємо на сцені фігуру у перспективній проекції для подальших пунктів завдання (рис. 2.1).

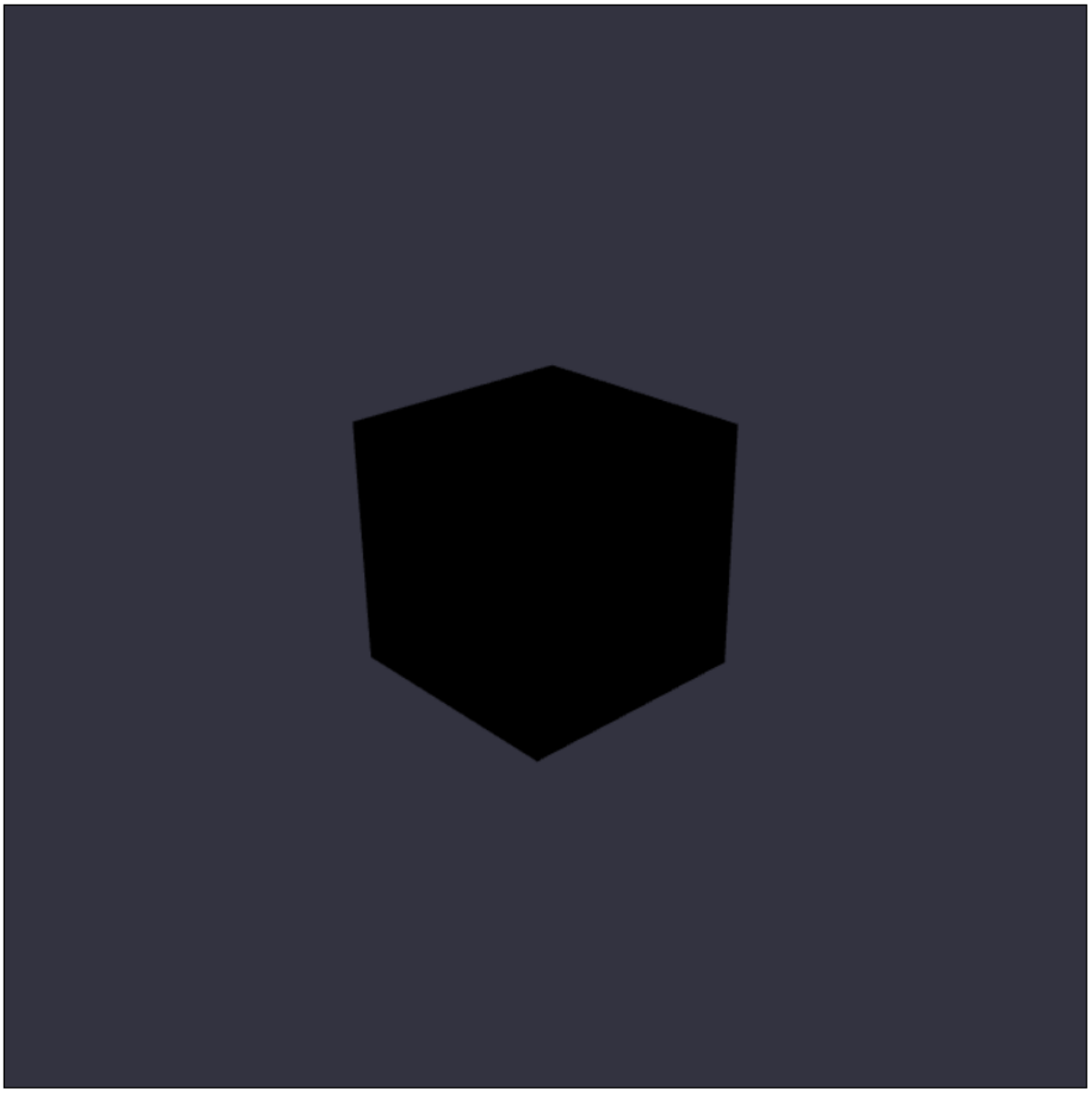

рис. 2.1. Створена фігура

3. Створення джерела світла

У даному пункті ми створюємо на сцені джерело світла та редагуємо його позицію для освітлення нашого об’єкту (рис. 3.1).

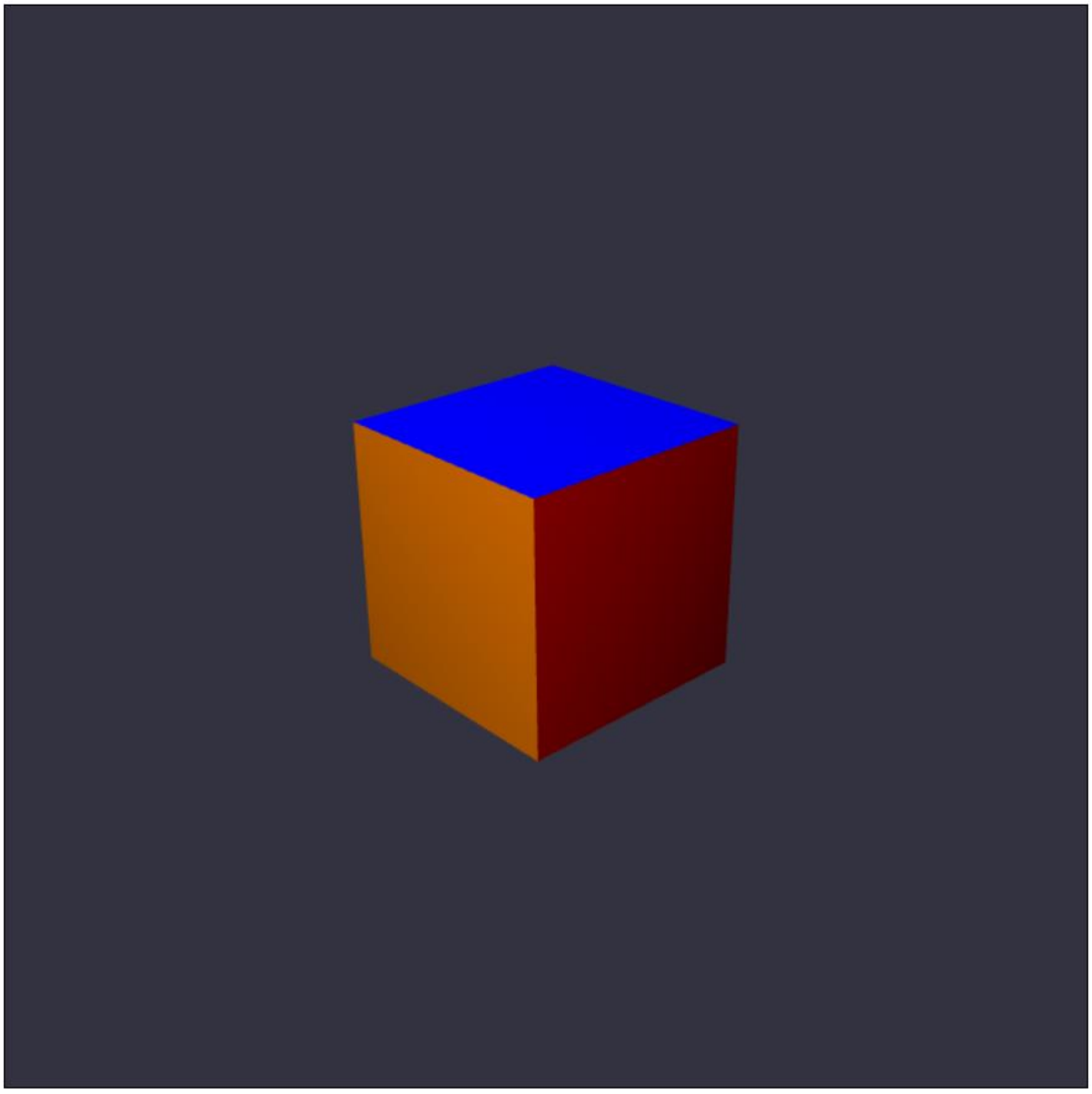

рис. 3.1. Куб під освітленням

Висновок: У ході даної роботи ми створили програму WebGL, яка відображає тривимірний об'єкт у перспективній проекції з використанням фонових кольорів. Для об'єкта було реалізовано освітлення сцени за моделлю віддзеркалення Фонга, що включає амбієнтне, дифузне та дзеркальне освітлення. Ми також внесли зміни в параметри освітлення сцени, демонструючи, як змінюються візуальні характеристики об'єкта під впливом різних умов освітлення. Це дало змогу поглибити розуміння роботи з графічним програмуванням та реалізації моделей освітлення.

ПРОГРАМНИЙ КОД

HTML:

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Lab 5 / Light and shadowing</title>
<style>
                body, html { margin: 0; padding: 0; overflow: hidden; }
                canvas { display: block; border: 1px solid black; margin: 3%;}
                </style>
</head>

<body>
<canvas id = "mycanvas" width = "700" height = "700"></canvas>
<script src="https://cdn.jsdelivr.net/npm/gl-matrix@2.8.1/dist/gl-matrix-
min.js"></script>
<script src="scripts/light.js"></script>
   </body>
</html>
```

JS:

```
const canvas = document.getElementById('mycanvas'); const gl =
canvas.getContext('webgl');
if (!gl) { alert("WebGL не підтримується в цьому браузері"); }
const vertexShaderSource = ` attribute vec4 a_position; attribute vec4 a_color;
attribute vec3 a_normal;
uniform mat4 u_matrix; uniform mat4 u_normalMatrix;
varying vec4 v_color; varying vec3 v_normal; varying vec3 v_position;
void main() { gl_Position = u_matrix * a_position; v_color = a_color; v_normal
= mat3(u_normalMatrix) * a_normal; v_position = vec3(u_matrix * a_position);
} `;
const fragmentShaderSource = `precision mediump float;
varying vec4 v_color; varying vec3 v_normal; varying vec3 v_position;
```

```
uniform vec3 u_lightPosition; uniform vec3 u_lightColor; uniform vec3
u_ambientColor; uniform vec3 u_cameraPosition;
void main() { vec3 normal = normalize(v_normal); vec3 lightDir =
normalize(u_lightPosition - v_position);
// Амбієнтне освітлення
vec3 ambient = u_ambientColor * v_color.rgb;

// Дифузне освітлення
float diff = max(dot(normal, lightDir), 0.0);
vec3 diffuse = diff * u_lightColor * v_color.rgb;

// Дзеркальне освітлення
vec3 viewDir = normalize(u_cameraPosition - v_position);
vec3 reflectDir = reflect(-lightDir, normal);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 16.0);
vec3 specular = spec * u_lightColor;

gl_FragColor = vec4(ambient + diffuse + specular, v_color.a);

} `;
function createShader(gl, type, source) { const shader = gl.createShader(type);
gl.shaderSource(shader, source); gl.compileShader(shader); if
(!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
console.error(gl.getShaderInfoLog(shader)); gl.deleteShader(shader); } return
shader; }
function createProgram(gl, vertexShaderSource, fragmentShaderSource) { const
vertexShader = createShader(gl, gl.VERTEX_SHADER, vertexShaderSource);
const fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
fragmentShaderSource); 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)); gl.deleteProgram(program); }
return program; }
const program = createProgram(gl, vertexShaderSource,
fragmentShaderSource);
const uLightPosition = gl.getUniformLocation(program, 'u_lightPosition');
const uLightColor = gl.getUniformLocation(program, 'u_lightColor'); const
```

```
uAmbientColor = gl.getUniformLocation(program, 'u_ambientColor'); const
uCameraPosition = gl.getUniformLocation(program, 'u_cameraPosition'); const
uNormalMatrix = gl.getUniformLocation(program, 'u_normalMatrix');
const positions = [ -0.2, -0.2, 0.2, 0.2, -0.2, 0.2, 0.2, 0.2, 0.2, -0.2, -0.2, 0.2, 0.2,
0.2, 0.2, -0.2, 0.2, 0.2,
-0.2, -0.2, -0.2,
 0.2, -0.2, -0.2,
 0.2,  0.2, -0.2,
-0.2, -0.2, -0.2,
 0.2,  0.2, -0.2,
-0.2,  0.2, -0.2,

-0.2,  0.2,  0.2,
 0.2,  0.2,  0.2,
 0.2,  0.2, -0.2,
-0.2,  0.2,  0.2,
 0.2,  0.2, -0.2,
-0.2,  0.2, -0.2,

-0.2, -0.2,  0.2,
 0.2, -0.2,  0.2,
 0.2, -0.2, -0.2,
-0.2, -0.2,  0.2,
 0.2, -0.2, -0.2,
-0.2, -0.2, -0.2,

-0.2, -0.2,  0.2,
-0.2,  0.2,  0.2,
-0.2,  0.2, -0.2,
-0.2, -0.2,  0.2,
-0.2,  0.2, -0.2,
-0.2, -0.2, -0.2,

 0.2, -0.2,  0.2,
 0.2,  0.2,  0.2,
 0.2,  0.2, -0.2,
 0.2, -0.2,  0.2,
 0.2,  0.2, -0.2,
```

```
 0.2, -0.2, -0.2,

];
const colors = [ [1.0, 0.0, 0.0, 1.0], [0.0, 1.0, 0.0, 1.0], [0.0, 0.0, 1.0, 1.0], [1.0,
1.0, 0.0, 1.0], [1.0, 0.5, 0.0, 1.0], [0.2, 0.2, 0.2, 1.0], ].flatMap(color =>
Array(6).fill(color).flat());
const normals = [
0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,

0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1,

0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,

0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0,

-1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0, -1, 0, 0,

1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,

];
const positionBuffer = gl.createBuffer(); const colorBuffer = gl.createBuffer();
const normalBuffer = gl.createBuffer(); gl.bindBuffer(gl.ARRAY_BUFFER,
normalBuffer); gl.bufferData(gl.ARRAY_BUFFER, new
Float32Array(normals), gl.STATIC_DRAW);
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions),
gl.STATIC_DRAW);
gl.bindBuffer(gl.ARRAY_BUFFER, colorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors),
gl.STATIC_DRAW);
const projectionMatrix = mat4.create(); const fieldOfViewRadians = Math.PI /
4; const aspect = canvas.width / canvas.height;
mat4.perspective(projectionMatrix, fieldOfViewRadians, aspect, 0.1, 100.0);
function drawScene() { gl.clearColor(1.0, 1.0, 1.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
gl.enable(gl.DEPTH_TEST);
```

```
const modelMatrix = mat4.create();
mat4.translate(modelMatrix, modelMatrix, [0, 0, -2]);
mat4.rotateY(modelMatrix, modelMatrix, Math.PI / 4);
mat4.rotateX(modelMatrix, modelMatrix, Math.PI / 10);
mat4.rotateZ(modelMatrix, modelMatrix, Math.PI / 10);

gl.clearColor(0.2, 0.2, 0.25, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

gl.uniform3fv(uLightPosition, [-1, 2, 5]);
gl.uniform3fv(uLightColor, [1.0, 1.0, 1.0]);
gl.uniform3fv(uAmbientColor, [0.2, 0.2, 0.2]);
gl.uniform3fv(uCameraPosition, [0, 0, 5]);

const normalMatrix = mat4.create();
mat4.invert(normalMatrix, modelMatrix);
mat4.transpose(normalMatrix, normalMatrix);
gl.uniformMatrix4fv(uNormalMatrix, false, normalMatrix);



const mvpMatrix = mat4.create();
mat4.multiply(mvpMatrix, projectionMatrix, modelMatrix);

gl.useProgram(program);
setAttribute(positionBuffer, 'a_position', 3, program);
setAttribute(colorBuffer, 'a_color', 4, program);
setAttribute(normalBuffer, 'a_normal', 3, program);
gl.uniformMatrix4fv(gl.getUniformLocation(program, 'u_matrix'), false,
mvpMatrix);

gl.drawArrays(gl.TRIANGLES, 0, positions.length / 3);
requestAnimationFrame(drawScene);

}
function setAttribute(buffer, attribute, size, program) {
gl.bindBuffer(gl.ARRAY_BUFFER, buffer); const location =
gl.getAttribLocation(program, attribute); gl.enableVertexAttribArray(location);
gl.vertexAttribPointer(location, size, gl.FLOAT, false, 0, 0); }
```

```
requestAnimationFrame(drawScene);
```