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

ЗВІТ

ПРО ЛАБОРАТОРНУ РОБОТУ №5

ТЕМА: «ОСВІТЛЕННЯ ТА ЗАТІНЕННЯ»

Виконала:
Студентка групи ІК-12
Макарчук Ольга

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

Київ 2024

Лабораторна робота №5

Тема: освітлення та затінення

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

Хід виконання роботи

Файл index.html:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>WebGL Demo</title>
    <script
      src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"
      integrity="sha512-
zhHQR0/H5SEBL3Wn6yYSaTTZej12z0hVZKOv3TwCUXT1z5qeqGcXJLLrbERYRScEDDpYIJhPC1fk31gqR78
3iQ=="
      crossorigin="anonymous"
      defer
    ></script>
    <script src="webgl-demo.js" type="module"></script>
  </head>

  <body>
    <canvas id="glcanvas" width="640" height="480"></canvas>
  </body>
</html>
```

Файл main.js:

```
import { initBuffers } from "./init-buffers.js";
import { drawScene } from "./draw-scene.js";

let cubeRotation = 0.0; // Початковий кут обертання куба
let deltaTime = 0; // Інтервал часу між кадрами

main(); // Виклик головної функції

function main() {
  const canvas = document.querySelector("#glcanvas"); // Отримання посилання на
елемент canvas
  const gl = canvas.getContext("webgl"); // Отримання контексту WebGL для малювання

  if (gl === null) {
    alert(
      "Unable to initialize WebGL. Your browser or machine may not support it."
```

```
    ); // Повідомлення у разі невдалої ініціалізації WebGL
    return;
  }

  gl.clearColor(0.0, 0.0, 0.0, 1.0); // Встановлення кольору очищення канвасу
  gl.clear(gl.COLOR_BUFFER_BIT); // Очистка канвасу

  // Вихідний код вершинного шейдера
  const vsSource = `
  attribute vec4 aVertexPosition;
  attribute vec3 aVertexNormal;
  attribute vec2 aTextureCoord;

  uniform mat4 uNormalMatrix;
  uniform mat4 uModelViewMatrix;
  uniform mat4 uProjectionMatrix;

  varying highp vec2 vTextureCoord;
  varying highp vec3 vLighting;

  void main(void) {
    gl_Position = uProjectionMatrix * uModelViewMatrix * aVertexPosition;
    vTextureCoord = aTextureCoord;

    // Застосування ефекту освітлення

    highp vec3 ambientLight = vec3(0.3, 0.3, 0.3);
    highp vec3 directionalLightColor = vec3(1, 1, 1);
    highp vec3 directionalVector = normalize(vec3(0.85, 0.8, 0.75));

    highp vec4 transformedNormal = uNormalMatrix * vec4(aVertexNormal, 1.0);

    highp float directional = max(dot(transformedNormal.xyz, directionalVector),
0.0);
    vLighting = ambientLight + (directionalLightColor * directional);
  }
`;

  // Вихідний код фрагментного шейдера
  const fsSource = `
  varying highp vec2 vTextureCoord;
  varying highp vec3 vLighting;

  uniform sampler2D uSampler;

  void main(void) {
    highp vec4 texelColor = texture2D(uSampler, vTextureCoord);

    gl_FragColor = vec4(texelColor.rgb * vLighting, texelColor.a);
  }
`;
```

```
  // Ініціалізація програми шейдерів
  const shaderProgram = initShaderProgram(gl, vsSource, fsSource);

  // Об'єкт з інформацією про програму шейдерів
  const programInfo = {
    program: shaderProgram,
    attribLocations: {
      vertexPosition: gl.getAttribLocation(shaderProgram, "aVertexPosition"),
      vertexNormal: gl.getAttribLocation(shaderProgram, "aVertexNormal"),
      textureCoord: gl.getAttribLocation(shaderProgram, "aTextureCoord"),
    },
    uniformLocations: {
      projectionMatrix: gl.getUniformLocation(
        shaderProgram,
        "uProjectionMatrix"
      ),
      modelViewMatrix: gl.getUniformLocation(shaderProgram, "uModelViewMatrix"),
      normalMatrix: gl.getUniformLocation(shaderProgram, "uNormalMatrix"),
      uSampler: gl.getUniformLocation(shaderProgram, "uSampler"),
    },
  };

  // Ініціалізація буферів вершин
  const buffers = initBuffers(gl);

  // Завантаження текстури
  const texture = loadTexture(gl, "cubetexture.jpg");
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true); // Налаштування орієнтації текстури

  let then = 0; // Попередній час

  // Функція малювання кадрів
  function render(now) {
    now *= 0.001; // Конвертування часу в секунди
    deltaTime = now - then; // Розрахунок інтервалу часу між кадрами
    then = now;

    drawScene(gl, programInfo, buffers, texture, cubeRotation); // Малювання сцени
    cubeRotation += deltaTime; // Оновлення кута обертання куба

    requestAnimationFrame(render); // Запит на малювання наступного кадру
  }
  requestAnimationFrame(render); // Початок циклу малювання кадрів
}

// Функція ініціалізації програми шейдерів
function initShaderProgram(gl, vsSource, fsSource) {
  const vertexShader = loadShader(gl, gl.VERTEX_SHADER, vsSource); // Завантаження
вершинного шейдера
```

```
  const fragmentShader = loadShader(gl, gl.FRAGMENT_SHADER, fsSource); //
Завантаження фрагментного шейдера

  const shaderProgram = gl.createProgram(); // Створення програми шейдерів
  gl.attachShader(shaderProgram, vertexShader); // Приєднання вершинного шейдера
  gl.attachShader(shaderProgram, fragmentShader); // Приєднання фрагментного
шейдера
  gl.linkProgram(shaderProgram); // Збирання програми шейдерів

  if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
    alert(
      `Unable to initialize the shader program: ${gl.getProgramInfoLog(
        shaderProgram
      )}`
    );
    return null;
  }

  return shaderProgram;
}

// Функція завантаження шейдера
function loadShader(gl, type, source) {
  const shader = gl.createShader(type); // Створення шейдера

  gl.shaderSource(shader, source); // Встановлення вихідного коду шейдера

  gl.compileShader(shader); // Компіляція шейдера

  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    alert(
      `An error occurred compiling the shaders: ${gl.getShaderInfoLog(shader)}`
    );
    gl.deleteShader(shader);
    return null;
  }

  return shader;
}

// Функція завантаження текстури
function loadTexture(gl, url) {
  const texture = gl.createTexture(); // Створення текстури
  gl.bindTexture(gl.TEXTURE_2D, texture); // Прив'язка текстури

  const level = 0;
  const internalFormat = gl.RGBA;
  const width = 1;
  const height = 1;
  const border = 0;
  const srcFormat = gl.RGBA;
```

```
  const srcType = gl.UNSIGNED_BYTE;
  const pixel = new Uint8Array([0, 0, 255, 255]);
  gl.texImage2D(
    gl.TEXTURE_2D,
    level,
    internalFormat,
    width,
    height,
    border,
    srcFormat,
    srcType,
    pixel
  ); // Створення пустої текстури

  const image = new Image(); // Створення об'єкту зображення
  image.onload = () => {
    gl.bindTexture(gl.TEXTURE_2D, texture);
    gl.texImage2D(
      gl.TEXTURE_2D,
      level,
      internalFormat,
      srcFormat,
      srcType,
      image
    ); // Завантаження зображення в текстуру

    if (isPowerOf2(image.width) && isPowerOf2(image.height)) {
      gl.generateMipmap(gl.TEXTURE_2D); // Генерація мірмар для текстури
    } else {
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
    }
  };
  image.src = url; // Завантаження зображення

  return texture;
}

// Функція, що перевіряє, чи є число степенем двійки
function isPowerOf2(value) {
  return (value & (value - 1)) === 0;
}
```

Файл init-buffers.js:

```
function initBuffers(gl) {
  // Ініціалізація буферів позицій, нормалей, текстурних координат та індексів
  const positionBuffer = initPositionBuffer(gl);
  const textureCoordBuffer = initTextureBuffer(gl);
```

```
  const indexBuffer = initIndexBuffer(gl);
  const normalBuffer = initNormalBuffer(gl);

  return {
    position: positionBuffer,
    normal: normalBuffer,
    textureCoord: textureCoordBuffer,
    indices: indexBuffer,
  };
}

// Ініціалізація буфера позицій вершин
function initPositionBuffer(gl) {
  const positionBuffer = gl.createBuffer();

  gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

  // Координати вершин куба
  const positions = [
    // Front face
    -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0,
    // Back face
    -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0,
    // Top face
    -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0,
    // Bottom face
    -1.0, -1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0,
    // Right face
    1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0,
    // Left face
    -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0,
  ];

  // Заповнення буфера даними про позиції
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions), gl.STATIC_DRAW);

  return positionBuffer;
}

// Ініціалізація буфера індексів
function initIndexBuffer(gl) {
  const indexBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);

  // Індекси вершин для кожного трикутника куба
  const indices = [
    0, 1, 2, 0, 2, 3, // front
    4, 5, 6, 4, 6, 7, // back
    8, 9, 10, 8, 10, 11, // top
    12, 13, 14, 12, 14, 15, // bottom
    16, 17, 18, 16, 18, 19, // right
```

```
    20, 21, 22, 20, 22, 23, // left
  ];

  // Заповнення буфера індексів
  gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices), gl.STATIC_DRAW);

  return indexBuffer;
}

// Ініціалізація буфера текстурних координат
function initTextureBuffer(gl) {
  const textureCoordBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, textureCoordBuffer);

  // Текстурні координати для кожного куба
  const textureCoordinates = [
    // Front
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
    // Back
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
    // Top
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
    // Bottom
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
    // Right
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
    // Left
    0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
  ];

  // Заповнення буфера даними про текстурні координати
  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoordinates),
gl.STATIC_DRAW);

  return textureCoordBuffer;
}

// Ініціалізація буфера нормалей
function initNormalBuffer(gl) {
  const normalBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, normalBuffer);

  // Нормалі для кожного куба
  const vertexNormals = [
    // Front
    0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0,

        // Back
        0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0,

        // Top
```

```
        0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0,

        // Bottom
        0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0,

        // Right
        1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,

        // Left
        -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0,
      ];

      // Заповнення буфера даними про нормалі вершин
      gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexNormals),
gl.STATIC_DRAW);

      return normalBuffer;
    }

    export { initBuffers };
```

Файл draw-scene.js:

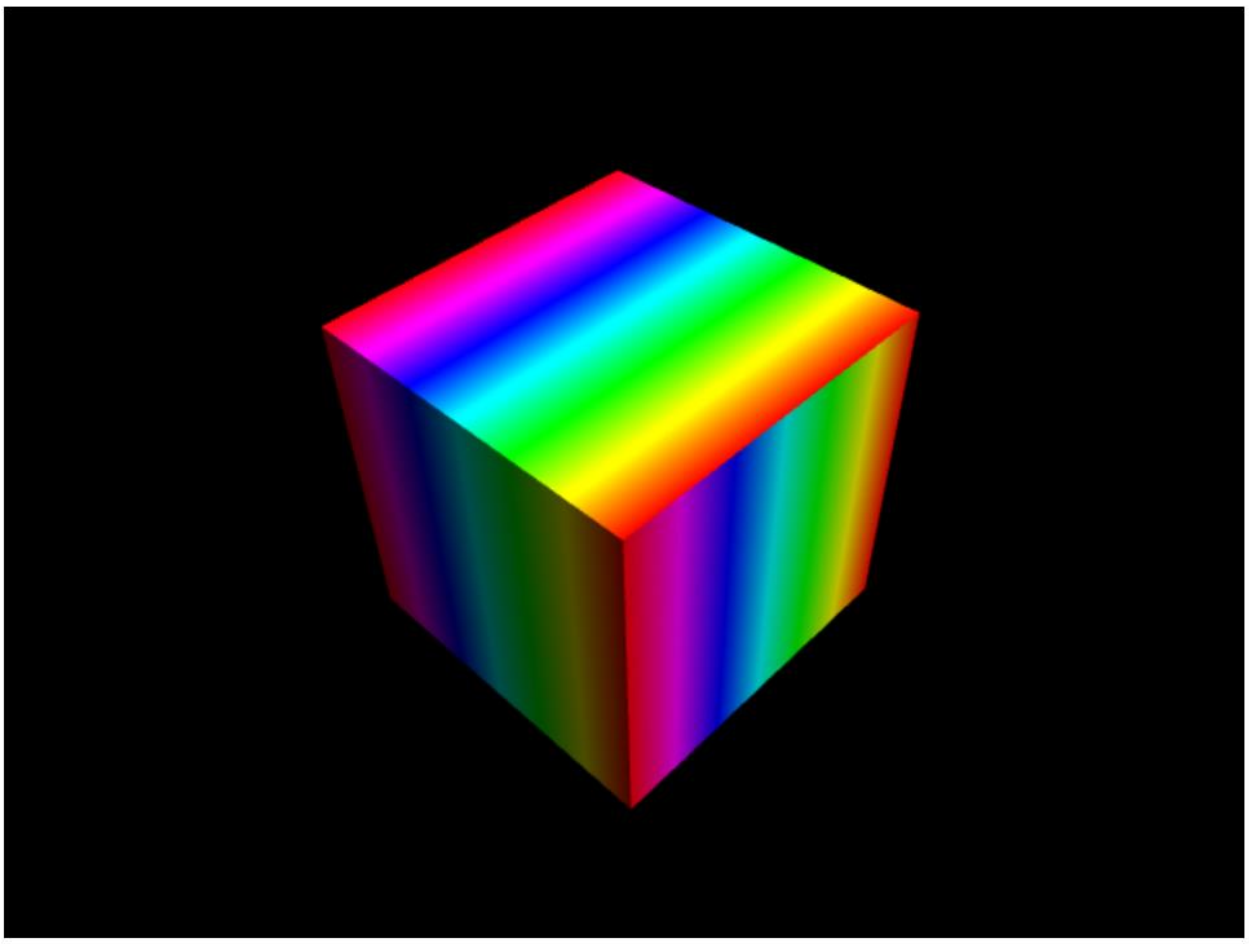
```
function drawScene(gl, programInfo, buffers, texture, cubeRotation) {
  gl.clearColor(0.0, 0.0, 0.0, 1.0); // Встановлення кольору очищення та глибини
канвасу
  gl.clearDepth(1.0); // Встановлення максимальної глибини очищення
  gl.enable(gl.DEPTH_TEST); // Увімкнення тесту глибини
  gl.depthFunc(gl.LEQUAL); // Встановлення правила порівняння глибини

  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT); // Очистка канвасу

  const fieldOfView = (45 * Math.PI) / 180; // Поле огляду камери у радіанах
  const aspect = gl.canvas.clientWidth / gl.canvas.clientHeight; // Співвідношення
сторін канвасу
  const zNear = 0.1; // Мінімальна глибина рендерингу
  const zFar = 100.0; // Максимальна глибина рендерингу
  const projectionMatrix = mat4.create(); // Створення матриці проекції

  mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar); //
Встановлення параметрів проекції

  const modelViewMatrix = mat4.create(); // Створення матриці моделі-огляду

  mat4.translate(
    modelViewMatrix,
    modelViewMatrix,
    [-0.0, 0.0, -6.0]
  ); // Зсув моделі вглиб сцени
```

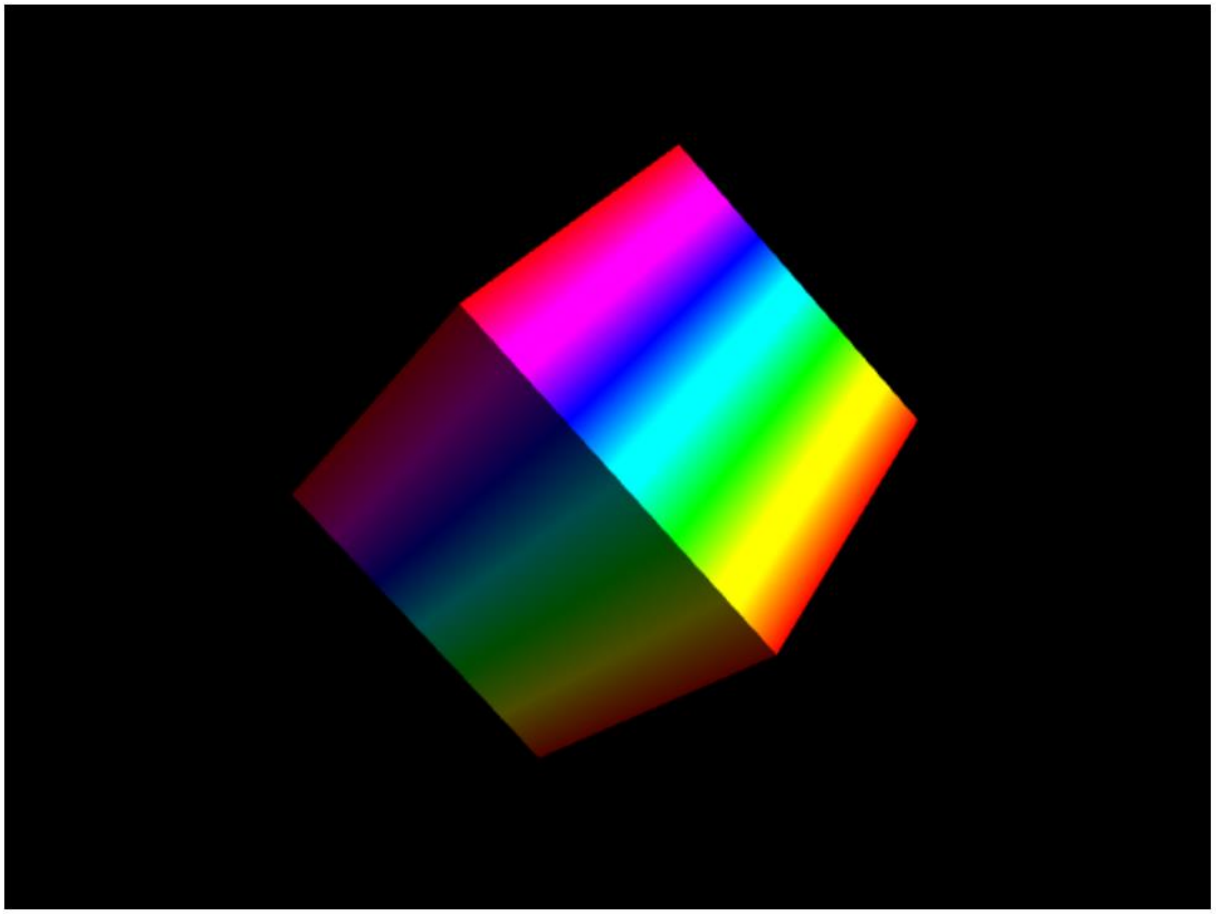
```
  mat4.rotate(
    modelViewMatrix,
    modelViewMatrix,
    cubeRotation,
    [0, 0, 1]
  ); // Обертання моделі навколо власної осі по осі Z
  mat4.rotate(
    modelViewMatrix,
    modelViewMatrix,
    cubeRotation * 0.7,
    [0, 1, 0]
  ); // Обертання моделі навколо власної осі по осі Y
  mat4.rotate(
    modelViewMatrix,
    modelViewMatrix,
    cubeRotation * 0.3,
    [1, 0, 0]
  ); // Обертання моделі навколо власної осі по осі X

  const normalMatrix = mat4.create(); // Створення матриці нормалей
  mat4.invert(normalMatrix, modelViewMatrix); // Обернення матриці моделі-огляду
  mat4.transpose(normalMatrix, normalMatrix); // Транспонування матриці нормалей

  setPositionAttribute(gl, buffers, programInfo); // Встановлення атрибута позицій
  setTextureAttribute(gl, buffers, programInfo); // Встановлення атрибута текстур
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, buffers.indices); // Прив'язка буфера
індексів
  setNormalAttribute(gl, buffers, programInfo); // Встановлення атрибута нормалей

  gl.useProgram(programInfo.program); // Використання програми шейдерів

  gl.uniformMatrix4fv(
    programInfo.uniformLocations.projectionMatrix,
    false,
    projectionMatrix
  ); // Передача матриці проекції у шейдер
  gl.uniformMatrix4fv(
    programInfo.uniformLocations.modelViewMatrix,
    false,
    modelViewMatrix
  ); // Передача матриці моделі-огляду у шейдер
  gl.uniformMatrix4fv(
    programInfo.uniformLocations.normalMatrix,
    false,
    normalMatrix
  ); // Передача матриці нормалей у шейдер

  gl.activeTexture(gl.TEXTURE0); // Активація текстурного блоку 0
  gl.bindTexture(gl.TEXTURE_2D, texture); // Прив'язка текстури
  gl.uniform1i(programInfo.uniformLocations.uSampler, 0); // Передача текстури у
шейдер
```

```
  {
    const vertexCount = 36; // Кількість вершин
    const type = gl.UNSIGNED_SHORT; // Тип даних індексів
    const offset = 0; // Зміщення
    gl.drawElements(gl.TRIANGLES, vertexCount, type, offset); // Малювання об'єкту
  }
}

function setPositionAttribute(gl, buffers, programInfo) {
  const numComponents = 3; // Кількість компонентів
  const type = gl.FLOAT; // Тип даних
  const normalize = false; // Нормалізація
  const stride = 0; // Крок
  const offset = 0; // Зміщення
  gl.bindBuffer(gl.ARRAY_BUFFER, buffers.position); // Прив'язка буфера позицій
  gl.vertexAttribPointer(
    programInfo.attribLocations.vertexPosition,
    numComponents,
    type,
    normalize,
    stride,
    offset
  ); // Встановлення атрибута позицій
  gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition); //
Увімкнення атрибута позицій
}

function setTextureAttribute(gl, buffers, programInfo) {
  const num = 2; // Кількість компонентів
  const type = gl.FLOAT; // Тип даних
  const normalize = false; // Нормалізація
  const stride = 0; // Крок
  const offset = 0; // Зміщення
  gl.bindBuffer(gl.ARRAY_BUFFER, buffers.textureCoord); // Прив'язка буфера
координат текстур
  gl.vertexAttribPointer(
    programInfo.attribLocations.textureCoord,
    num,
    type,
    normalize,
    stride,
    offset
  ); // Встановлення атрибута координат текстур
  gl.enableVertexAttribArray(programInfo.attribLocations.textureCoord); //
Увімкнення атрибута координат текстур
}

function setNormalAttribute(gl, buffers, programInfo) {
  const numComponents = 3; // Кількість компонентів
  const type = gl.FLOAT; // Тип даних
```

```
  const normalize = false; // Нормалізація
  const stride = 0; // Крок
  const offset = 0; // Зміщення
  gl.bindBuffer(gl.ARRAY_BUFFER, buffers.normal); // Прив'язка буфера нормалей
  gl.vertexAttribPointer(
    programInfo.attribLocations.vertexNormal,
    numComponents,
    type,
    normalize,
    stride,
    offset
  ); // Встановлення атрибута нормалей
  gl.enableVertexAttribArray(programInfo.attribLocations.vertexNormal); //
Увімкнення атрибута нормалей
}

export { drawScene };
```

Результат:





Висновок: Під час лабораторної роботи я отримала практичні навички щодо роботи з освітленням графічної сцени на основі програмного інтерфейсу WebGL.