# Міністерство освіти і науки України

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

# Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 8 з дисципліни

«Комп’ютерна графіка та мультимедіа»

# „ПРОЕКЦІЙНІ ТІНІ ТА КОНВЕЄР ВІЗУАЛІЗАЦІЇ”

**Виконав(ла)**

*ІП-11 Калашніков А.Є.*

(шифр, прізвище, ім'я, по батькові)

# Перевірив

доц. каф. ІПІ, *Родіонов П.Ю.*

(посада, прізвище, ім'я, по батькові)

Київ 2024

**Завдання.**

1. Створити графічну сцену, що містить три об’єкти: великий

прямокутник (земля); два прямокутники меншого розміру, що

розташовуються на великому прямокутнику.

2. Додати у графічну сцену джерело освітлення.

3. Застосувати буфер глибини з функцією перевірки глибини для

рисування тіней лише у випадках, коли це необхідно.

4. Зробити тіні напівпрозорими для забезпечення більшого реалізму

графічної сцени.

5. Скласти звіт про виконану роботу.

Lab8.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>WebGL Lab 8</title>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-min.js"></script>

    <script src="lab8.js"></script>

</head>

<body>

    <canvas id="glCanvas" width="840" height="680"></canvas>

</body>

</html>

Lab8.js(1)

document.addEventListener("DOMContentLoaded", function () {

const canvas = document.getElementById('glCanvas');

let gl = canvas.getContext('webgl');

if (!gl) {

console.error('WebGL not supported, falling back on experimental-webgl');

gl = canvas.getContext('experimental-webgl');

}

if (!gl) {

alert('Your browser does not support WebGL');

return;

}

// Define shader source codes

const vertexShaderSource = `

attribute vec3 a\_Vertex;

attribute vec3 a\_Color;

uniform mat4 u\_Scene\_transform;

uniform mat4 u\_Camera\_model\_transform;

uniform mat4 u\_Light\_transform;

varying vec4 v\_Vertex\_camera\_space;

varying vec4 v\_Vertex\_shadow\_map;

varying vec3 v\_Color;

void main() {

v\_Vertex\_shadow\_map = u\_Light\_transform \* vec4(a\_Vertex, 1.0);

v\_Vertex\_camera\_space = u\_Camera\_model\_transform \* vec4(a\_Vertex, 1.0);

gl\_Position = u\_Scene\_transform \* vec4(a\_Vertex, 1.0);

v\_Color = a\_Color;

}

`;

const fragmentShaderSource = `

precision mediump float;

uniform sampler2D u\_Shadow\_map;

uniform vec3 u\_Light\_color;

varying vec4 v\_Vertex\_camera\_space;

varying vec4 v\_Vertex\_shadow\_map;

varying vec3 v\_Color;

bool in\_shadow(vec4 vertex\_relative\_to\_light, sampler2D shadow\_map) {

vec3 ndc = vertex\_relative\_to\_light.xyz / vertex\_relative\_to\_light.w;

vec3 percentages = ndc \* 0.5 + 0.5;

vec4 shadow\_map\_color = texture2D(shadow\_map, percentages.xy);

float shadow\_map\_distance = shadow\_map\_color.r;

return percentages.z > shadow\_map\_distance + 0.005;

}

void main() {

vec3 light\_effect = u\_Light\_color;

if (in\_shadow(v\_Vertex\_shadow\_map, u\_Shadow\_map)) {

light\_effect \*= 0.4; // Make the shadow semi-transparent

}

gl\_FragColor = vec4(v\_Color \* light\_effect, 1.0);

}

`;

// Compile shaders

function compileShader(gl, source, type) {

const shader = gl.createShader(type);

gl.shaderSource(shader, source);

gl.compileShader(shader);

if (!gl.getShaderParameter(shader, gl.COMPILE\_STATUS)) {

console.error('An error occurred compiling the shaders: ' + gl.getShaderInfoLog(shader));

gl.deleteShader(shader);

return null;

}

return shader;

}

const vertexShader = compileShader(gl, vertexShaderSource, gl.VERTEX\_SHADER);

const fragmentShader = compileShader(gl, fragmentShaderSource, gl.FRAGMENT\_SHADER);

// Link shaders to create program

const shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertexShader);

gl.attachShader(shaderProgram, fragmentShader);

gl.linkProgram(shaderProgram);

if (!gl.getProgramParameter(shaderProgram, gl.LINK\_STATUS)) {

console.error('Unable to initialize the shader program: ' + gl.getProgramInfoLog(shaderProgram));

}

// Use the program

gl.useProgram(shaderProgram);

// Get attribute and uniform locations

const vertexPositionAttribute = gl.getAttribLocation(shaderProgram, 'a\_Vertex');

const vertexColorAttribute = gl.getAttribLocation(shaderProgram, 'a\_Color');

gl.enableVertexAttribArray(vertexPositionAttribute);

gl.enableVertexAttribArray(vertexColorAttribute);

const sceneTransformUniform = gl.getUniformLocation(shaderProgram, 'u\_Scene\_transform');

const cameraModelTransformUniform = gl.getUniformLocation(shaderProgram, 'u\_Camera\_model\_transform');

const lightTransformUniform = gl.getUniformLocation(shaderProgram, 'u\_Light\_transform');

const shadowMapUniform = gl.getUniformLocation(shaderProgram, 'u\_Shadow\_map');

const lightColorUniform = gl.getUniformLocation(shaderProgram, 'u\_Light\_color');

// Create buffers

const vertices = new Float32Array([

// Ground rectangle (light gray)

-2.0, -0.5, -2.0, 0.7, 0.7, 0.7,

2.0, -0.5, -2.0, 0.7, 0.7, 0.7,

2.0, -0.5, 2.0, 0.7, 0.7, 0.7,

-2.0, -0.5, 2.0, 0.7, 0.7, 0.7,

// Smaller rectangle 1 (green)

1.5, -0.5, +0.7, 0.0, 1.0, 0.0,

-1.5, -0.5, -0.3, 0.0, 1.0, 0.0,

-1.5, 0.5, -2.2, 0.0, 1.0, 0.0,

1.5, 0.5, -0.7, 0.0, 1.0, 0.0,

// Smaller rectangle 2 (blue)

0.5, -0.5, 1.5, 0.0, 0.0, 1.0,

1.3, -0.5, 1.5, 0.0, 0.0, 1.0,

1.3, 0.3, 1.5, 0.0, 0.0, 1.0,

0.5, 0.3, 1.5, 0.0, 0.0, 1.0,

]);

const indices = new Uint16Array([

0, 1, 2, 0, 2, 3,

4, 5, 6, 4, 6, 7,

8, 9, 10, 8, 10, 11,

]);

const vertexBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

gl.bufferData(gl.ARRAY\_BUFFER, vertices, gl.STATIC\_DRAW);

const indexBuffer = gl.createBuffer();

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, indexBuffer);

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, indices, gl.STATIC\_DRAW);

// Set up the scene transformations

const projectionMatrix = mat4.create();

mat4.perspective(projectionMatrix, Math.PI / 4, canvas.width / canvas.height, 0.1, 100.0);

const viewMatrix = mat4.create();

mat4.lookAt(viewMatrix, [4, 4, 5], [0, 0, 0], [0, 1, 0]);

const modelMatrix = mat4.create();

const sceneTransform = mat4.create();

mat4.multiply(sceneTransform, projectionMatrix, viewMatrix);

mat4.multiply(sceneTransform, sceneTransform, modelMatrix);

// Light transformations

const lightPosition = [1.30, -0.00, -0.00];

const lightColor = [1, 1, 1];

const lightProjectionMatrix = mat4.create();

mat4.perspective(lightProjectionMatrix, Math.PI / 4, 1.0, 2.1, 100.0);

const lightViewMatrix = mat4.create();

mat4.lookAt(lightViewMatrix, lightPosition, [-8.0, -4.5, -1.0], [0.0, 1.0, 0.0]);

const lightTransform = mat4.create();

mat4.multiply(lightTransform, lightProjectionMatrix, lightViewMatrix);

// Set the uniform values

gl.uniformMatrix4fv(sceneTransformUniform, false, sceneTransform);

gl.uniformMatrix4fv(cameraModelTransformUniform, false, viewMatrix);

gl.uniformMatrix4fv(lightTransformUniform, false, lightTransform);

gl.uniform3fv(lightColorUniform, lightColor);

// Set the shadow map texture

const shadowMapSize = 1024;

// Create a renderbuffer for the depth attachment

const shadowDepthRenderbuffer = gl.createRenderbuffer();

gl.bindRenderbuffer(gl.RENDERBUFFER, shadowDepthRenderbuffer);

gl.renderbufferStorage(gl.RENDERBUFFER, gl.DEPTH\_COMPONENT16, shadowMapSize, shadowMapSize);

// Create the framebuffer and attach the renderbuffer

const shadowFramebuffer = gl.createFramebuffer();

gl.bindFramebuffer(gl.FRAMEBUFFER, shadowFramebuffer);

gl.framebufferRenderbuffer(gl.FRAMEBUFFER, gl.DEPTH\_ATTACHMENT, gl.RENDERBUFFER, shadowDepthRenderbuffer);

const shadowFramebufferStatus = gl.checkFramebufferStatus(gl.FRAMEBUFFER);

if (shadowFramebufferStatus !== gl.FRAMEBUFFER\_COMPLETE) {

console.error('Framebuffer not complete: ' + shadowFramebufferStatus.toString());

// Handle the error or try alternative approaches

}

gl.bindFramebuffer(gl.FRAMEBUFFER, null);

// Function to render the shadow map

function renderShadowMap() {

gl.bindFramebuffer(gl.FRAMEBUFFER, shadowFramebuffer);

gl.viewport(0, 0, shadowMapSize, shadowMapSize);

gl.clear(gl.DEPTH\_BUFFER\_BIT);

// Disable the vertex color attribute for the shadow map rendering pass

gl.disableVertexAttribArray(vertexColorAttribute);

// Set the light transform for shadow map rendering

gl.uniformMatrix4fv(lightTransformUniform, false, lightTransform);

gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 6 \* Float32Array.BYTES\_PER\_ELEMENT, 0);

gl.enableVertexAttribArray(vertexPositionAttribute);

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, indexBuffer);

gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED\_SHORT, 0);

gl.bindFramebuffer(gl.FRAMEBUFFER, null);

}

// Function to render the scene

function renderScene() {

gl.viewport(0, 0, canvas.width, canvas.height);

gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

// Set the scene transform for camera rendering

gl.uniformMatrix4fv(sceneTransformUniform, false, sceneTransform);

gl.uniformMatrix4fv(cameraModelTransformUniform, false, viewMatrix);

gl.uniform1i(shadowMapUniform, 0);

gl.bindBuffer(gl.ARRAY\_BUFFER, vertexBuffer);

gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 6 \* Float32Array.BYTES\_PER\_ELEMENT, 0);

gl.enableVertexAttribArray(vertexPositionAttribute);

gl.vertexAttribPointer(vertexColorAttribute, 3, gl.FLOAT, false, 6 \* Float32Array.BYTES\_PER\_ELEMENT, 3 \* Float32Array.BYTES\_PER\_ELEMENT);

gl.enableVertexAttribArray(vertexColorAttribute);

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, indexBuffer);

gl.drawElements(gl.TRIANGLES, indices.length, gl.UNSIGNED\_SHORT, 0);

}

function render() {

renderShadowMap();

renderScene();

requestAnimationFrame(render);

}

// Enable depth testing

gl.enable(gl.DEPTH\_TEST);

// Set clear color to black, fully opaque

gl.clearColor(0.0, 0.0, 0.0, 1.0);

// Start rendering

render();

});

Lab8.js (2)

window.onload = function() {

    const canvas = document.getElementById('glCanvas');

    const gl = canvas.getContext('webgl');

    if (!gl) {

        console.error('Unable to initialize WebGL. Your browser may not support it.');

        return;

    }

    // Vertex shader program for shadow map

    const vsShadowSource = `

        attribute vec4 aVertexPosition;

        uniform mat4 uLightMatrix;

        void main(void) {

            gl\_Position = uLightMatrix \* aVertexPosition;

        }

    `;

    // Fragment shader program for shadow map

    const fsShadowSource = `

        precision highp float;

        void main(void) {

            gl\_FragColor = vec4(gl\_FragCoord.z, gl\_FragCoord.z, gl\_FragCoord.z, 1.0);

        }

    `;

    // Vertex shader program for scene

    const vsSceneSource = `

        attribute vec4 aVertexPosition;

        attribute vec4 aVertexColor;

        uniform mat4 uModelViewMatrix;

        uniform mat4 uProjectionMatrix;

        uniform mat4 uLightMatrix;

        varying lowp vec4 vColor;

        varying vec4 vShadowCoord;

        void main(void) {

            gl\_Position = uProjectionMatrix \* uModelViewMatrix \* aVertexPosition;

            vColor = aVertexColor;

            vShadowCoord = uLightMatrix \* aVertexPosition;

        }

    `;

    // Fragment shader program for scene

    const fsSceneSource = `

        precision lowp float;

        precision lowp int;

        varying lowp vec4 vColor;

        varying vec4 vShadowCoord;

        uniform sampler2D uShadowMap;

        uniform vec3 uLightPosition;

        uniform mat4 uViewMatrix; // Add viewMatrix uniform

        void main(void) {

            vec3 shadowCoord = vShadowCoord.xyz / vShadowCoord.w;

            shadowCoord = shadowCoord \* 0.5 + 0.5;

            float shadow = 0.0;

            float distance = texture2D(uShadowMap, shadowCoord.xy).r;

            if (shadowCoord.z > distance) {

                shadow = 0.5;

            }

            gl\_FragColor = vec4(vColor.rgb \* (1.0 - shadow), vColor.a);

        }

    `;

    let rotationX = 0;

    let rotationY = 0;

    let cameraPosition = [5, 5, 5];

    let cameraTarget = [0, 0, 0];

    const shadowShaderProgram = initShaderProgram(gl, vsShadowSource, fsShadowSource);

    const sceneShaderProgram = initShaderProgram(gl, vsSceneSource, fsSceneSource);

    const shadowProgramInfo = {

        program: shadowShaderProgram,

        attribLocations: {

            vertexPosition: gl.getAttribLocation(shadowShaderProgram, 'aVertexPosition'),

        },

        uniformLocations: {

            lightMatrix: gl.getUniformLocation(shadowShaderProgram, 'uLightMatrix'),

        },

    };

    const sceneProgramInfo = {

        program: sceneShaderProgram,

        attribLocations: {

            vertexPosition: gl.getAttribLocation(sceneShaderProgram, 'aVertexPosition'),

            vertexColor: gl.getAttribLocation(sceneShaderProgram, 'aVertexColor'),

        },

        uniformLocations: {

            projectionMatrix: gl.getUniformLocation(sceneShaderProgram, 'uProjectionMatrix'),

            modelViewMatrix: gl.getUniformLocation(sceneShaderProgram, 'uModelViewMatrix'),

            lightMatrix: gl.getUniformLocation(sceneShaderProgram, 'uLightMatrix'),

            shadowMap: gl.getUniformLocation(sceneShaderProgram, 'uShadowMap'),

        },

    };

    const buffers = initBuffers(gl);

    const shadowFramebuffer = initFramebuffer(gl);

    drawScene(gl, shadowFramebuffer, shadowProgramInfo, sceneProgramInfo, buffers);

    function initShaderProgram(gl, vsSource, fsSource) {

        const vertexShader = loadShader(gl, gl.VERTEX\_SHADER, vsSource);

        const fragmentShader = loadShader(gl, gl.FRAGMENT\_SHADER, fsSource);

        const shaderProgram = gl.createProgram();

        gl.attachShader(shaderProgram, vertexShader);

        gl.attachShader(shaderProgram, fragmentShader);

        gl.linkProgram(shaderProgram);

        if (!gl.getProgramParameter(shaderProgram, gl.LINK\_STATUS)) {

            console.error('Unable to initialize the shader program: ' + gl.getProgramInfoLog(shaderProgram));

            return null;

        }

        return shaderProgram;

    }

    function loadShader(gl, type, source) {

        const shader = gl.createShader(type);

        gl.shaderSource(shader, source);

        gl.compileShader(shader);

        if (!gl.getShaderParameter(shader, gl.COMPILE\_STATUS)) {

            console.error('An error occurred compiling the shaders: ' + gl.getShaderInfoLog(shader));

            gl.deleteShader(shader);

            return null;

        }

        return shader;

    }

    function initBuffers(gl) {

        const positionBuffer = gl.createBuffer();

        gl.bindBuffer(gl.ARRAY\_BUFFER, positionBuffer);

        const positions = [

            // Ground

            -3.0, -2.5, 0.0,

             2.0, -2.5, 0.0,

             2.0,  1.5, 0.0,

            -3.0,  1.5, 0.0,

            // Smaller rectangle 1

            -1.5, -0.5, 0.1,

            -0.3, -0.5, 0.1,

            -0.3, 0.1, 0.1,

            -1.5, 0.1, 0.1,

        ];

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(positions), gl.STATIC\_DRAW);

        const faceColors = [

            [0.0, 1.0, 0.0, 1.0],    // Ground: green

            [1.0, 0.0, 0.0, 1.0],    // Smaller rectangle: red

        ];

        var colors = [];

        for (var j = 0; j < faceColors.length; ++j) {

            const c = faceColors[j];

            colors = colors.concat(c, c, c, c);

        }

        const colorBuffer = gl.createBuffer();

        gl.bindBuffer(gl.ARRAY\_BUFFER, colorBuffer);

        gl.bufferData(gl.ARRAY\_BUFFER, new Float32Array(colors), gl.STATIC\_DRAW);

        const indexBuffer = gl.createBuffer();

        gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, indexBuffer);

        const indices = [

            0, 1, 2,      0, 2, 3,    // ground

            4, 5, 6,      4, 6, 7,    // smaller rectangle

        ];

        gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, new Uint16Array(indices), gl.STATIC\_DRAW);

        return {

            position: positionBuffer,

            color: colorBuffer,

            indices: indexBuffer,

        };

    }

    function initFramebuffer(gl) {

        const framebuffer = gl.createFramebuffer();

        gl.bindFramebuffer(gl.FRAMEBUFFER, framebuffer);

        const texture = gl.createTexture();

        gl.bindTexture(gl.TEXTURE\_2D, texture);

        gl.texImage2D(gl.TEXTURE\_2D, 0, gl.DEPTH\_COMPONENT16, canvas.width, canvas.height, 0, gl.DEPTH\_COMPONENT, gl.UNSIGNED\_SHORT, null);

        gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MIN\_FILTER, gl.NEAREST);

        gl.texParameteri(gl.TEXTURE\_2D, gl.TEXTURE\_MAG\_FILTER, gl.NEAREST);

        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.framebufferTexture2D(gl.FRAMEBUFFER, gl.DEPTH\_ATTACHMENT, gl.TEXTURE\_2D, texture, 0);

        gl.bindTexture(gl.TEXTURE\_2D, null);

        gl.bindFramebuffer(gl.FRAMEBUFFER, null);

        return {

            framebuffer: framebuffer,

            texture: texture,

        };

    }

    function drawScene(gl, shadowFramebuffer, shadowProgramInfo, sceneProgramInfo, buffers) {

        const fieldOfView = 45 \* Math.PI / 180;

        const aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;

        const zNear = 0.1;

        const zFar = 100.0;

        const lightPosition = [0.5, 0.0, 0.0];

        const lightMatrix = mat4.create();

        mat4.perspective(lightMatrix, fieldOfView, aspect, zNear, zFar);

        mat4.lookAt(lightMatrix, lightPosition, [0.3, 0.1, 0.3], [0, 1, 0]);

        // Render to shadow map

        gl.bindFramebuffer(gl.FRAMEBUFFER, shadowFramebuffer.framebuffer);

        gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

        gl.clear(gl.DEPTH\_BUFFER\_BIT);

        gl.useProgram(shadowProgramInfo.program);

        gl.bindBuffer(gl.ARRAY\_BUFFER, buffers.position);

        gl.vertexAttribPointer(

            shadowProgramInfo.attribLocations.vertexPosition,

            3, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(shadowProgramInfo.attribLocations.vertexPosition);

        gl.uniformMatrix4fv(

            shadowProgramInfo.uniformLocations.lightMatrix,

            false, lightMatrix);

        gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, buffers.indices);

        gl.drawElements(gl.TRIANGLES, 12, gl.UNSIGNED\_SHORT, 0);

        // Render scene

        gl.bindFramebuffer(gl.FRAMEBUFFER, null);

        gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

        gl.clear(gl.COLOR\_BUFFER\_BIT | gl.DEPTH\_BUFFER\_BIT);

        const projectionMatrix = mat4.create();

        mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar);

        const viewMatrix = mat4.create();

        gl.uniformMatrix4fv(

            sceneProgramInfo.uniformLocations.viewMatrix,

            false, viewMatrix);

        mat4.lookAt(viewMatrix, cameraPosition, cameraTarget, [10, 10, 0]); // Update lookAt to use cameraPosition and cameraTarget

        const modelViewMatrix = mat4.create();

        mat4.translate(modelViewMatrix, modelViewMatrix, [-0.0, 0.0, -6.0]);

        gl.useProgram(sceneProgramInfo.program);

        gl.bindBuffer(gl.ARRAY\_BUFFER, buffers.position);

        gl.vertexAttribPointer(

            sceneProgramInfo.attribLocations.vertexPosition,

            3, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(sceneProgramInfo.attribLocations.vertexPosition);

        gl.bindBuffer(gl.ARRAY\_BUFFER, buffers.color);

        gl.vertexAttribPointer(

            sceneProgramInfo.attribLocations.vertexColor,

            4, gl.FLOAT, false, 0, 0);

        gl.enableVertexAttribArray(sceneProgramInfo.attribLocations.vertexColor);

        gl.uniformMatrix4fv(

            sceneProgramInfo.uniformLocations.projectionMatrix,

            false, projectionMatrix);

        gl.uniformMatrix4fv(

            sceneProgramInfo.uniformLocations.viewMatrix, // Pass the viewMatrix to the shader

            false, viewMatrix);

        gl.uniformMatrix4fv(

            sceneProgramInfo.uniformLocations.modelViewMatrix,

            false, modelViewMatrix);

        gl.uniformMatrix4fv(

            sceneProgramInfo.uniformLocations.lightMatrix,

            false, lightMatrix);

        gl.uniform1i(sceneProgramInfo.uniformLocations.shadowMap, 0);

        gl.activeTexture(gl.TEXTURE0);

        gl.bindTexture(gl.TEXTURE\_2D, shadowFramebuffer.texture);

        gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, buffers.indices);

        gl.drawElements(gl.TRIANGLES, 12, gl.UNSIGNED\_SHORT, 0);

    }

    function updateCamera() {

        const speed = 0.1;

        document.addEventListener('keydown', function(event) {

            switch(event.key.toLowerCase()) {

                case 'w':

                    cameraPosition[2] -= speed;

                    break;

                case 'a':

                    cameraPosition[0] -= speed;

                    break;

                case 's':

                    cameraPosition[2] += speed;

                    break;

                case 'd':

                    cameraPosition[0] += speed;

                    break;

            }

        });

    }

    updateCamera();

}

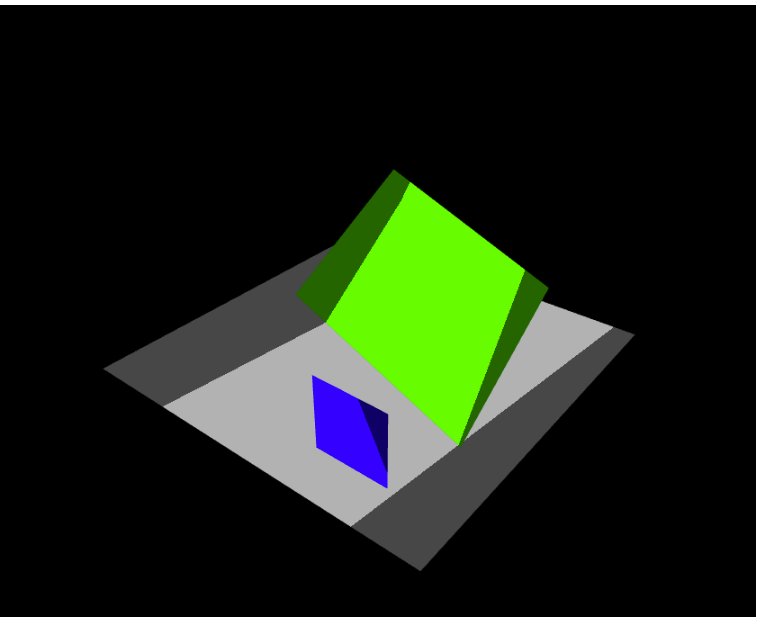


Рис.1 Скріншот виконання

Висновок: виконуючи дану лабораторну роботу, я отримав практичні навички щодо створення тіней за допомогою проекційних матриць. Створено графічну сцену, що містить три об’єкти: великий прямокутник (земля); два прямокутники меншого розміру, що розташовуються на великому прямокутнику.

Додано у графічну сцену джерело освітлення. Застосовано буфер глибини з функцією перевірки глибини для рисування тіней лише у випадках, коли це необхідно. Зроблено тіні напівпрозорими для забезпечення більшого реалізму графічної сцени.