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

ЗВІТ

ПРО ЛАБОРАТОРНУ РОБОТУ №1

ТЕМА: «ОСНОВИ СТВОРЕННЯ НАЙПРОСТІШОЇ WEBGL-ПРОГРАМИ»

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

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

Київ 2023

ХІД РОБОТИ

1. Створити програму WebGL:

- створити документ HTML з елементом Canvas;

- налаштувати Viewport та встановити довільний колір екрану;

- створити контекст WebGL за допомогою «setupWebGL» та подію «windowonload».

Лістинг 1 - Програмний код у файлі HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Лабораторна 1</title>
    <script src="script.js" defer></script>
</head>
<body>
    <canvas id="myCanvas" width="400" height="400"></canvas>

</body>
</html>
```

Лістинг 2 - Програмний код у файлі JavaScript

```
function setupWebGL(canvasId) {
    let canvas = document.getElementById(canvasId);
    let gl = canvas.getContext('webgl');
    if (!gl) {
        console.error('WebGL не підтримується, перевірте ваш браузер.');
        return null;
    }
    gl.viewport(0, 0, canvas.width, canvas.height);
    gl.clearColor(0.2, 0.4, 0.6, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
    return gl;
}

function windowOnLoad() {
    let gl = setupWebGL('myCanvas');
    if (!gl) return;

}
window.onload = windowOnLoad;
```

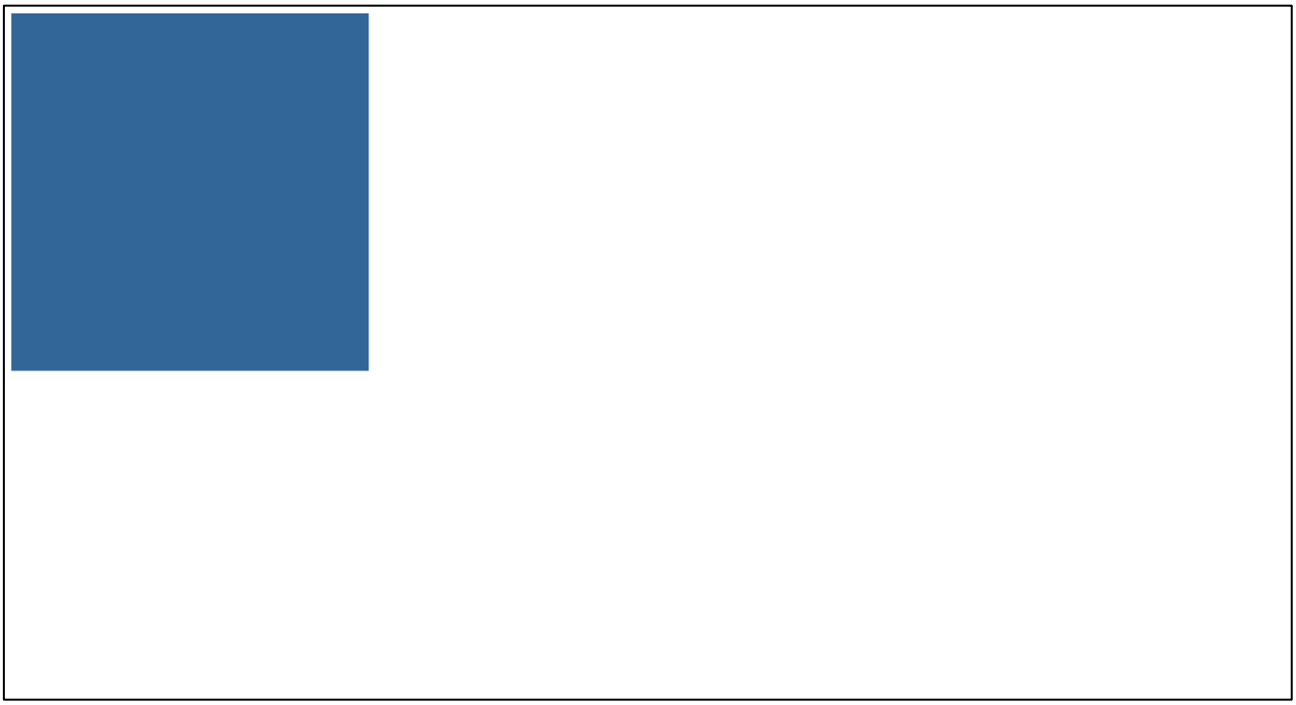

Рисунок 1 – Результат виконання першого завдання

2. Виконати рендеринг кольорового трикутника:

- створити фрагментний шейдер;

- створити вершинний шейдер;

- налаштувати буфер вершин з відповідним покажчиком на атрибут

для створення трикутника, кожна вершина якого має відмінний від інших вершин колір.

Лістинг 3 - Програмний код у файлі JavaScript

```
function setupWebGL(canvasId) {
    let canvas = document.getElementById(canvasId);
    let gl = canvas.getContext('webgl');
    if (!gl) {
        console.error('WebGL не підтримується, перевірте ваш браузер.');
        return null;
    }
    gl.clearColor(0.2, 0.4, 0.6, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
    return gl;
}

function createShader(gl, type, source) {
    let shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        console.error('Shader compilation failed:',
gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
```

```
    }
    return shader;
}

function createProgram(gl, vertexShader, fragmentShader) {
    let program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);
    if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
        console.error('Program linking failed:',
gl.getProgramInfoLog(program));
        gl.deleteProgram(program);
        return null;
    }
    return program;
}

function windowOnLoad() {
    let gl = setupWebGL('myCanvas');
    if (!gl) return;

    let vsSource = `
        attribute vec4 coordinates;
        attribute vec4 vertexColor;
        varying lowp vec4 varyingColor;
        void main(void) {
            gl_Position = coordinates;
            varyingColor = vertexColor;
        }`;

    let fsSource = `
        varying lowp vec4 varyingColor;
        void main(void) {
            gl_FragColor = varyingColor;
        }`;

    let vertexShader = createShader(gl, gl.VERTEX_SHADER, vsSource);
    let fragmentShader = createShader(gl, gl.FRAGMENT_SHADER, fsSource);

    let shaderProgram = createProgram(gl, vertexShader, fragmentShader);
    gl.useProgram(shaderProgram);

    let vertices = new Float32Array([
        0.0, 1.0, 1.0, 0.0, 0.0,
        -1.0, -1.0, 0.0, 1.0, 0.0,
        1.0, -1.0, 0.0, 0.0, 1.0
    ]);

    let vertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

    let coord = gl.getAttribLocation(shaderProgram, "coordinates");
    gl.vertexAttribPointer(coord, 2, gl.FLOAT, false, 20, 0);
    gl.enableVertexAttribArray(coord);

    let color = gl.getAttribLocation(shaderProgram, "vertexColor");
    gl.vertexAttribPointer(color, 3, gl.FLOAT, false, 20, 8);
    gl.enableVertexAttribArray(color);

    gl.drawArrays(gl.TRIANGLES, 0, 3);
}
```

```
window.onload = windowOnLoad;
```

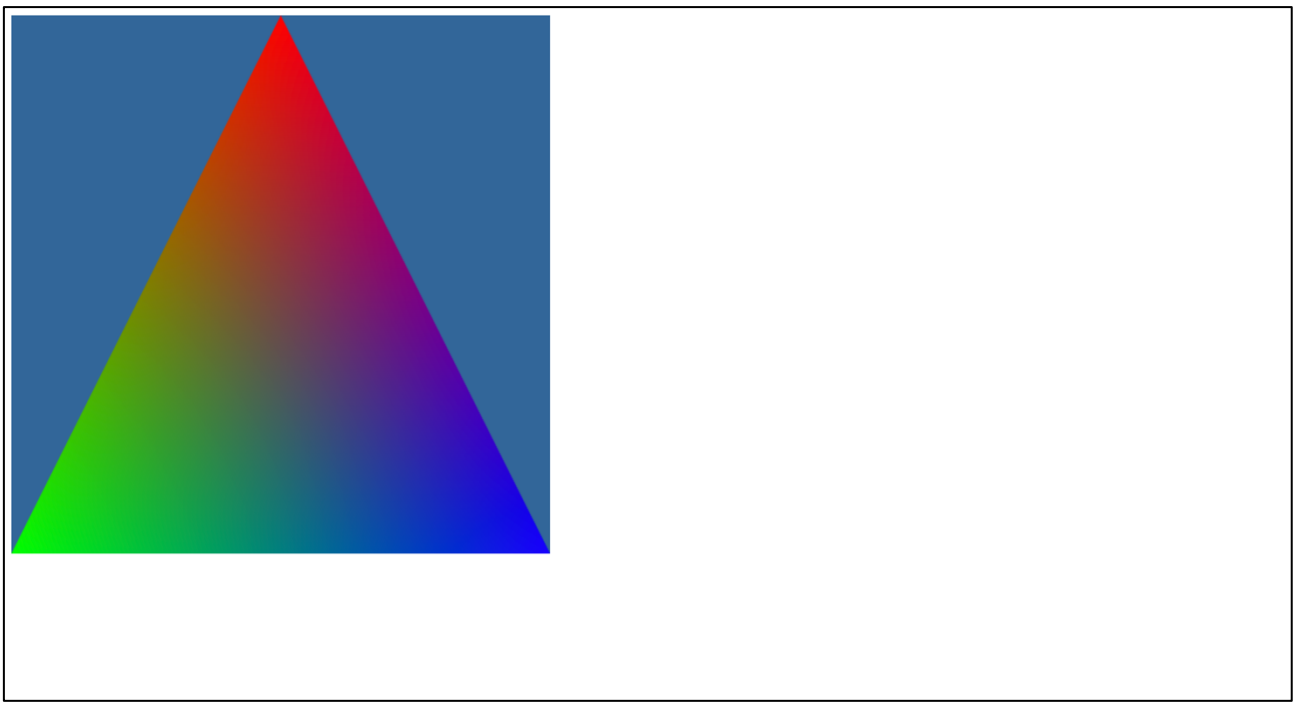

Рисунок 2 – Результат виконання другого завдання

3. Обертання фігури:

- додати другий трикутник та утворити прямокутник;

- розмістити квадрат в центрі екрана та організувати його обертання навколо власного центру за допомогою функції «RequestAnimationFrame».

Лістинг 4 - Програмний код у файлі JavaScript

```
function setupWebGL(canvasId) {
    let canvas = document.getElementById(canvasId);
    let gl = canvas.getContext('webgl');
    if (!gl) {
        console.error('WebGL не підтримується, перевірте ваш браузер.');
        return null;
    }
    return gl;
}

function createShader(gl, type, source) {
    let shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
        console.error('Shader compilation failed:',
gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }
    return shader;
}

function createProgram(gl, vertexShader, fragmentShader) {
    let program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
```

```
    gl.linkProgram(program);
    if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
        console.error('Program linking failed:',
gl.getProgramInfoLog(program));
        gl.deleteProgram(program);
        return null;
    }
    return program;
}

function windowOnLoad() {
    let gl;
    let shaderProgram;
    let triangleRotation = 0.0;
    gl = setupWebGL('myCanvas');
    if (!gl) return;

    let vsSource = `
        attribute vec4 coordinates;
        attribute vec4 vertexColor;
        varying lowp vec4 varyingColor;
        void main(void) {
            gl_Position = coordinates;
            varyingColor = vertexColor;
        }`;

    let fsSource = `
        varying lowp vec4 varyingColor;
        void main(void) {
            gl_FragColor = varyingColor;
        }`;

    let vertexShader = createShader(gl, gl.VERTEX_SHADER, vsSource);
    let fragmentShader = createShader(gl, gl.FRAGMENT_SHADER, fsSource);

    shaderProgram = createProgram(gl, vertexShader, fragmentShader);
    gl.useProgram(shaderProgram);


    let vertices = new Float32Array([
        -0.5,  0.5,  1.0, 0.0, 0.0,
        -0.5, -0.5,  0.0, 1.0, 0.0,
         0.5, -0.5,  0.0, 0.0, 1.0,

        -0.5,  0.5,  0.0, 0.0, 1.0,
         0.5,  0.5,  0.0, 1.0, 0.0,
         0.5, -0.5,  1.0, 0.0, 0.0      ]);

    let vertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

    let coord = gl.getAttribLocation(shaderProgram, "coordinates");
    gl.vertexAttribPointer(coord, 2, gl.FLOAT, false, 20, 0);
    gl.enableVertexAttribArray(coord);

    let color = gl.getAttribLocation(shaderProgram, "vertexColor");
    gl.vertexAttribPointer(color, 3, gl.FLOAT, false, 20, 8);
    gl.enableVertexAttribArray(color);

    function rotateVertex(x, y, cosTheta, sinTheta) {
        let newX = cosTheta * x - sinTheta * y;
        let newY = sinTheta * x + cosTheta * y;
```

```
        return [newX, newY];
    }

    function drawTriangles() {
        gl.drawArrays(gl.TRIANGLES, 0, 6);
    }

    function animateTriangles() {
        triangleRotation += 0.01;

        gl.clear(gl.COLOR_BUFFER_BIT);

        let cosTheta = Math.cos(triangleRotation);
        let sinTheta = Math.sin(triangleRotation);

        let vertices = new Float32Array([
            ...rotateVertex(-0.5,  0.5, cosTheta, sinTheta),1.0, 0.0, 0.0,
            ...rotateVertex(-0.5,  -0.5, cosTheta, sinTheta),0.0, 1.0,
0.0,
            ...rotateVertex(0.5,  -0.5, cosTheta, sinTheta),0.0, 0.0, 1.0,

            ...rotateVertex(-0.5,  0.5, cosTheta, sinTheta),0.0, 1.0, 1.0,
            ...rotateVertex(0.5,  0.5, cosTheta, sinTheta),1.0, 1.0, 0.0,
            ...rotateVertex(0.5,  -0.5, cosTheta, sinTheta), 1.0, 0.0, 1.0
        ]);


        gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
        drawTriangles();

        requestAnimationFrame(animateTriangles);
    }

    requestAnimationFrame(animateTriangles);
}

window.onload = windowOnLoad;
```

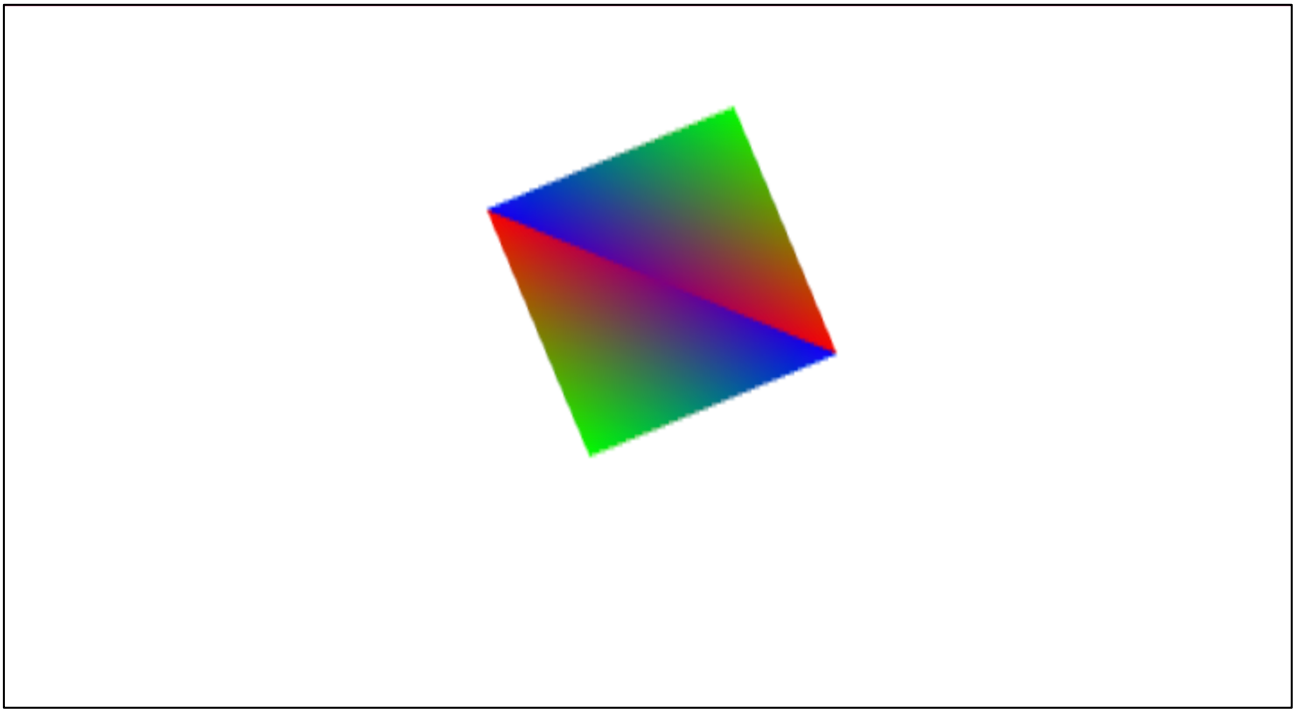

Рисунок 3 – Результат виконання третього завдання

4. Створити довільну графічну фігуру за допомогою режима gl.TRIANGLE_FAN та налаштувати її рух вниз та вгору.

Лістинг 5 - Програмний код у файлі JavaScript

```
const canvas = document.getElementById("myCanvas");
const gl = canvas.getContext("webgl");

if (!gl) {
    console.error("Unable to initialize WebGL. Your browser may not
support it.");
}

const vertices = [
    0.0, 0.0,
    -0.5, -0.5,
    0.5, -0.5,
    0.5, 0.5,
    -0.5, 0.5
];

const vertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);

const numVertices = 5;

const vsSource = `
    attribute vec2 coordinates;
    void main(void) {
        gl_Position = vec4(coordinates, 0.0, 1.0);
    }
`;

const fsSource = `
    void main(void) {
        gl_FragColor = vec4(1.0, 0.0, 1.0, 1.0);
    }
`;

const vertexShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertexShader, vsSource);
gl.compileShader(vertexShader);

const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragmentShader, fsSource);
gl.compileShader(fragmentShader);

const shaderProgram = gl.createProgram();
gl.attachShader(shaderProgram, vertexShader);
gl.attachShader(shaderProgram, fragmentShader);
gl.linkProgram(shaderProgram);
gl.useProgram(shaderProgram);

const coord = gl.getAttribLocation(shaderProgram, "coordinates");

gl.enableVertexAttribArray(coord);

gl.vertexAttribPointer(coord, 2, gl.FLOAT, false, 0, 0);
```

```
let yTranslate = 0.0;
let direction = 1;

function render() {

    gl.clear(gl.COLOR_BUFFER_BIT);
    gl.clearColor(0.2, 0.4, 0.6, 1.0);
    yTranslate += 0.01 * direction;

    if (yTranslate > 0.5 || yTranslate < -0.5) {
        direction *= -1;
    }

    const vertices = [
        0.0, yTranslate,
        -0.5, -0.5 + yTranslate,
        0.5, -0.5 + yTranslate,
        0.5, 0.5 + yTranslate,
        -0.5, 0.5 + yTranslate
    ];

    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
gl.STATIC_DRAW);
    gl.drawArrays(gl.TRIANGLE_FAN, 0, numVertices);

    requestAnimationFrame(render);
}

requestAnimationFrame(render);
```

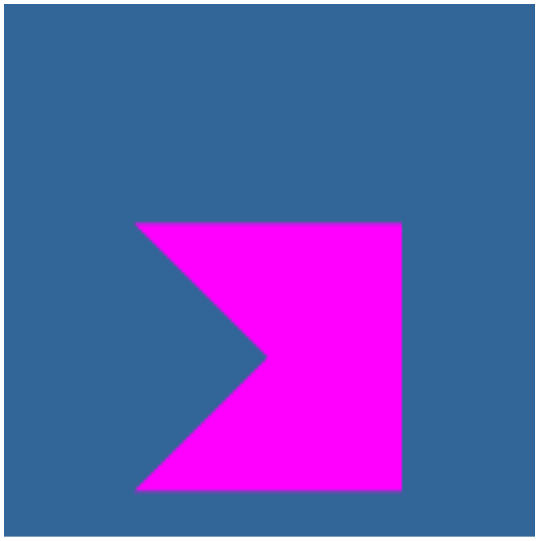

Рисунок 4 – Результат виконання четвертого завдання(рух у низ)

Рисунок 5 – Результат виконання четвертого завдання(рух у гору)

ВИСНОВОК

Мета даної роботи була створити програму WebGL та виконати рендеринг графічних об'єктів з використанням WebGL API. Давайте оцінимо виконану роботу, порівняємо отримані результати з теоретичними положеннями та очікуваними результатами.

- Створення програми WebGL: Було успішно створено HTML-документ з елементом Canvas та налаштовано viewport. Контекст WebGL було створено і готово використовувати для рендерингу.
- Рендеринг кольорового трикутника: Фрагментний і вершинний шейдери були створені, і вони працюють правильно для обробки вершин і пікселів. Був створений буфер вершин для рендерингу трикутника з різними кольорами вершин.
- Обертання фігури: Додатковий трикутник був успішно доданий до сцени. Фігура була розміщені в центрі екрана, і була реалізована її обертання навколо власного центру за допомогою RequestAnimationFrame.
- Створення графічної фігури з режимом TRIANGLE_FAN: Була створена графічна фігура за допомогою режиму gl.TRIANGLE_FAN, і ця фігура успішно рухалася вниз та вгору.

Загалом, робота була виконана успішно, і всі поставлені завдання були досягнуті. Отримані результати відповідають теоретичним положенням та очікуваним результатам. Даний проект є гарним вступом до програмування на WebGL та створення веб-графіки за допомогою цієї технології.