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-22
Підпанюк В.А.

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

Київ 2024

Лабораторна робота №1

ОСНОВИ СТВОРЕННЯ НАЙПРОСТІШОЇ WEBGL-ПРОГРАМИ

Мета: отримати практичні навички програмування WebGL-програм, які дозволяють створювати графічні об’єкти та анімації.

Завдання:

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

1. Створити програму WebGL

Результати створення html коду з блоком canvas, налаштування в’юпорта, вибір кольору заливки та створення події windowonload (рис. 1.1).

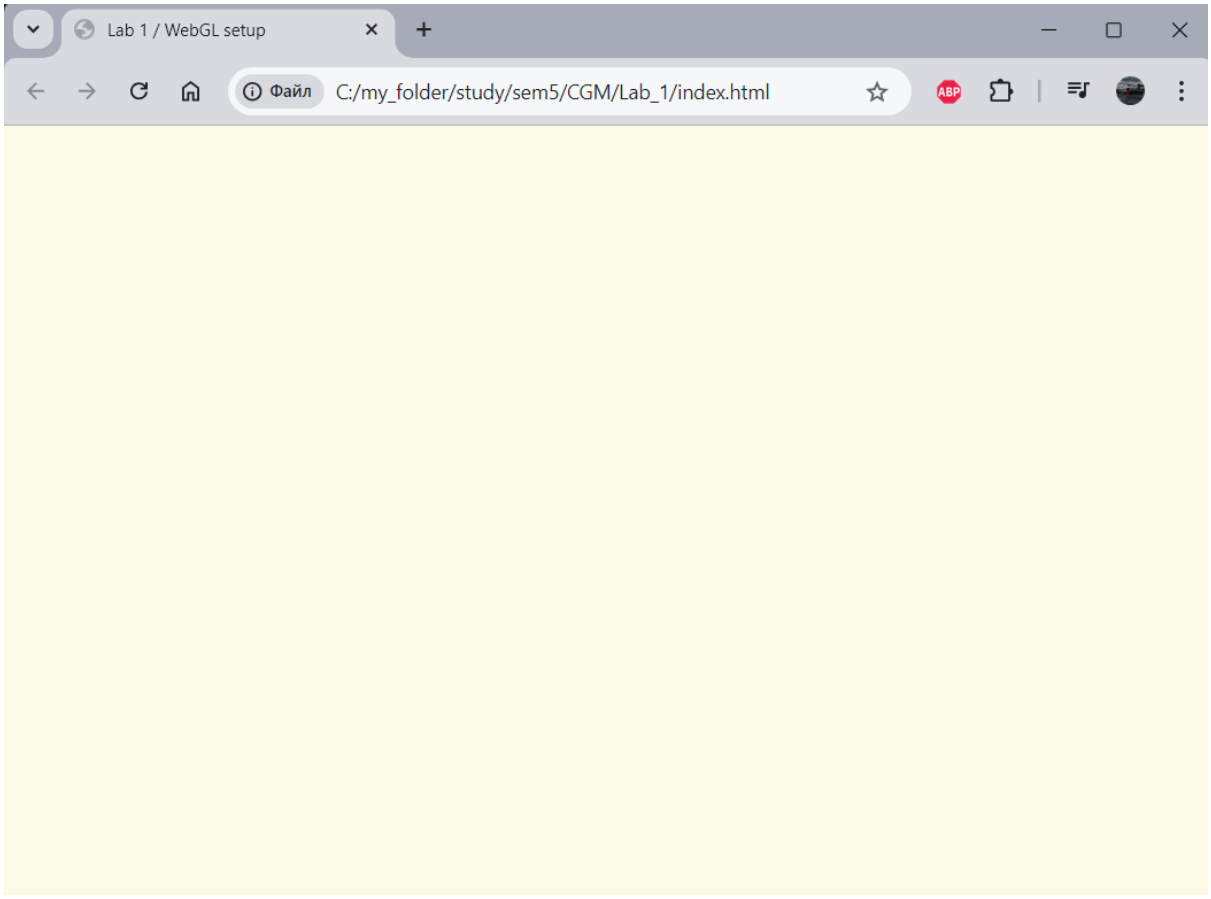

рис. 1.1. Демонстрація виконання першого пункту завдання

2. Виконати рендеринг кольорового трикутника

Результат створення вершинного та фрагментного шейдерів, додавання буферу вершин трикутника та їх кольорів та додавання самого трикутника (рис. 2.1).

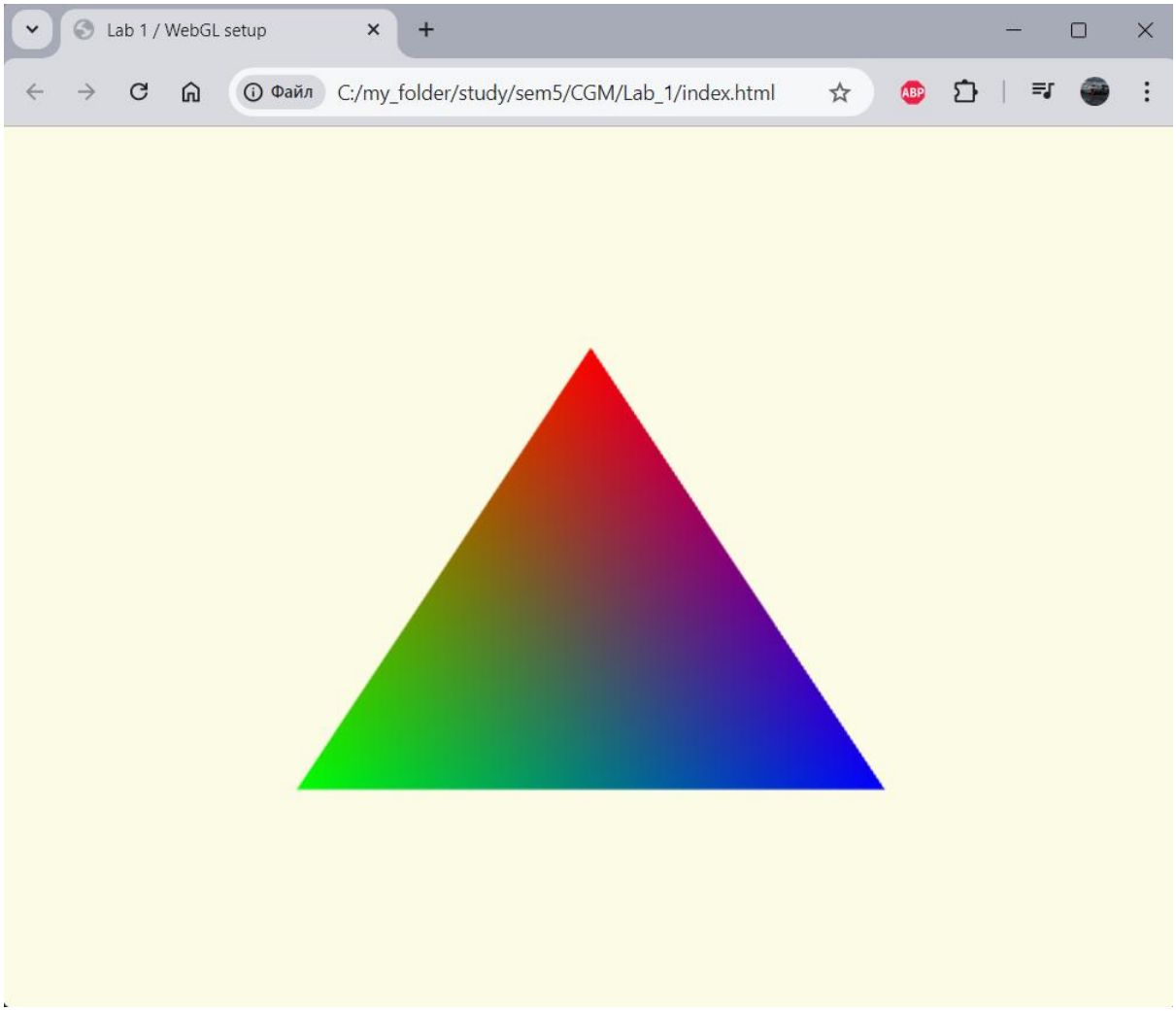

рис. 2.1. Демонстрація виконання другого пункту завдання

3. Обертання фігури

Результат додавання другого трикутника, моделювання з наявних трикутників квадрату та обертання його у центрі (рис. 3.1).

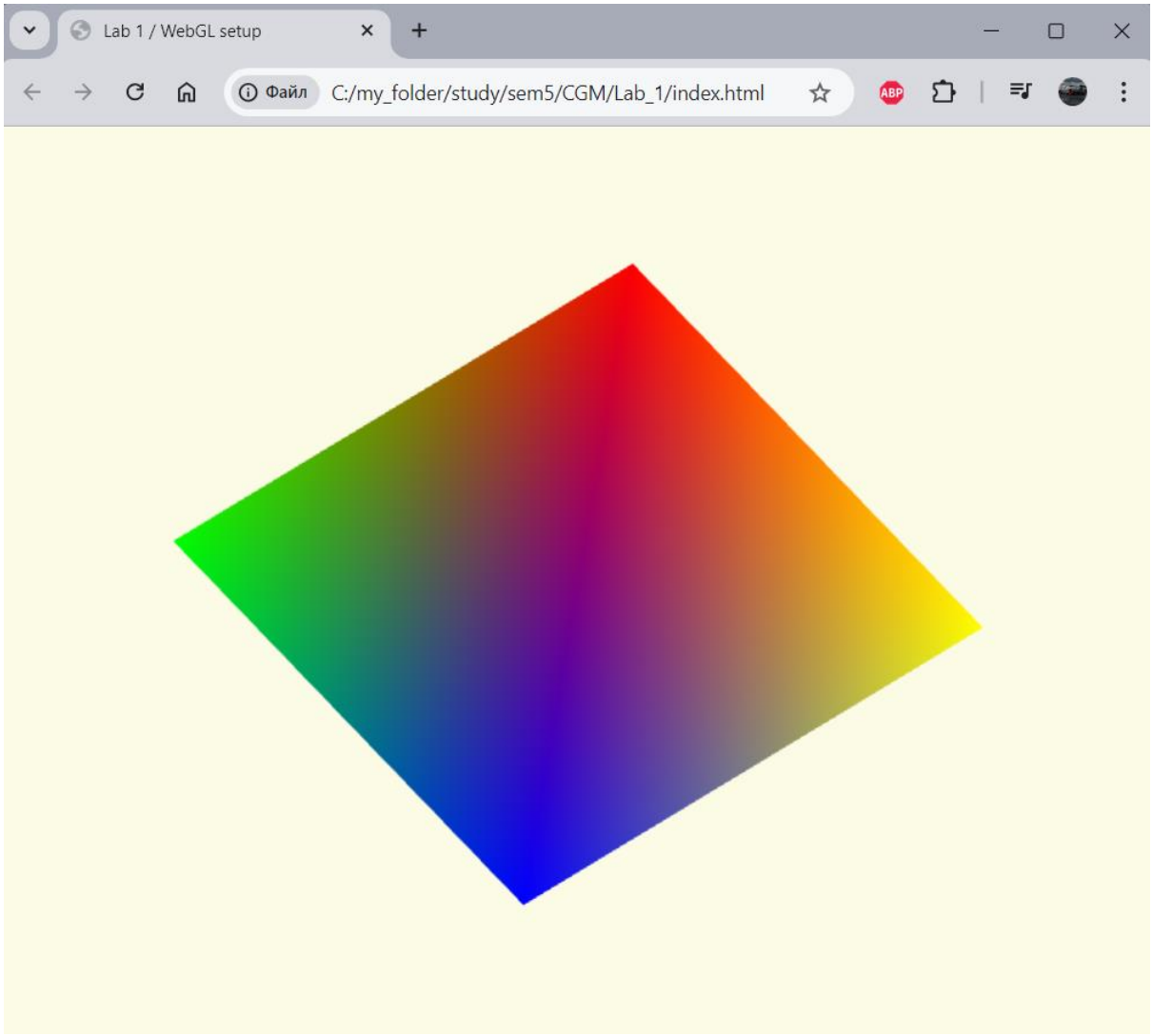

рис. 3.1. Демонстрація виконання третього пункту завдання

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

Результат додавання фігури (кола) за допомогою gl.TRIANGLE_FAN (рис. 4.1).

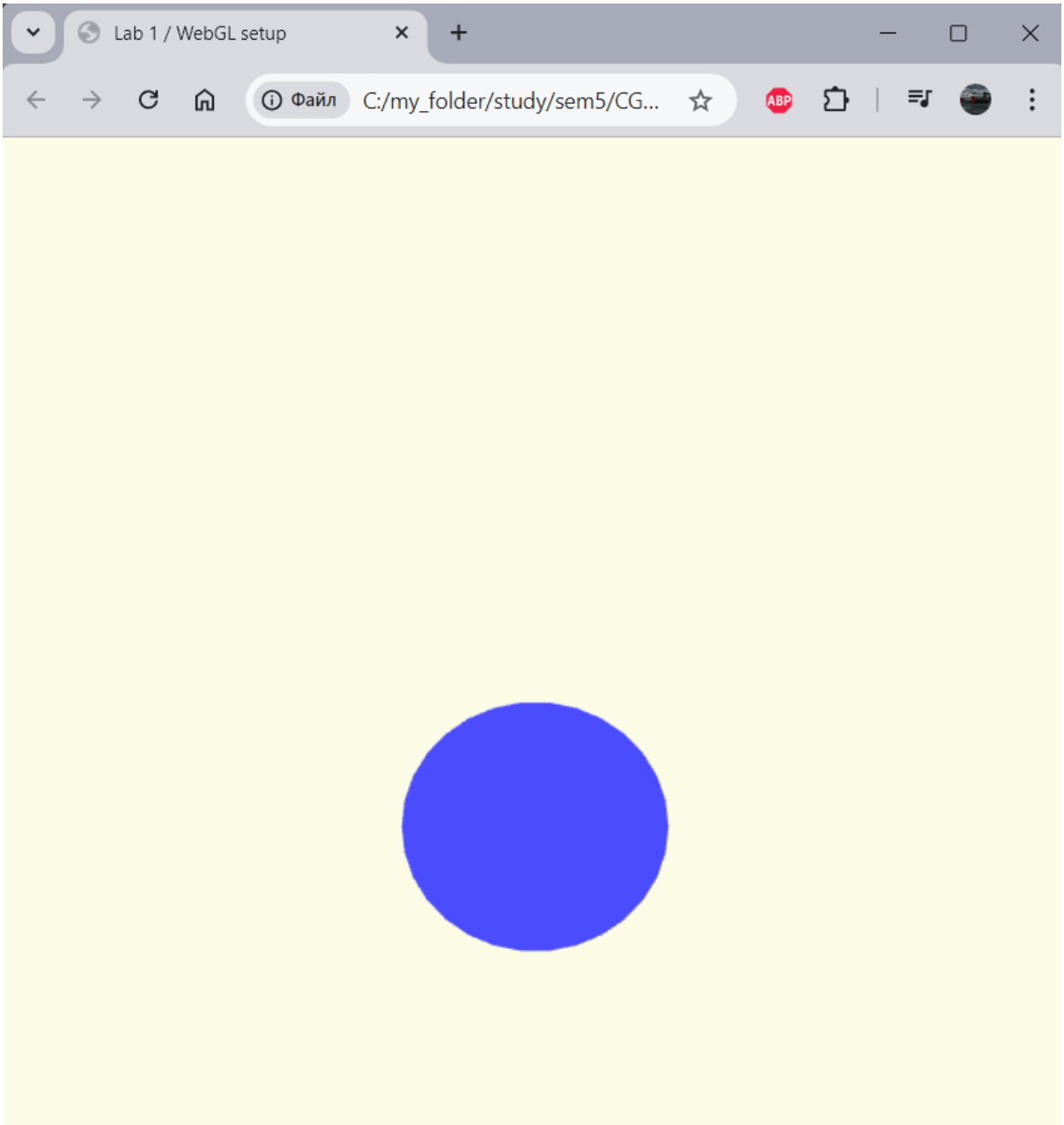

рис. 4.1. Демонстрація виконання 4 пункту завдання

Висновок: під час виконання даної лабораторної роботи ми навчилися створювати графічні фігури в WebGL, налаштовувати кольорові шейдери, організовувати обертання фігур та реалізовувати анімацію, використовуючи режим `gl.TRIANGLE_FAN` для створення кола.

ПРОГРАМНИЙ КОД

Версія коду для 3-го завдання:

HTML:

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Lab 1 / WebGL setup</title>
      <style>
         body, html { margin: 0; padding: 0; overflow: hidden; }
         canvas { display: block; }
      </style>
   </head>

   <body>
      <canvas id = "mycanvas" width = "600" height = "200"></canvas>
      <script src="https://cdn.jsdelivr.net/npm/gl-matrix@2.8.1/dist/gl-matrix-min.js"></script>
      <script src="scripts/webgl_setup.js"></script>
   </body>
</html>
```

JS:

```
function setupWebGL()
{
  const canvas = document.getElementById("mycanvas");

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

  if (!gl)
  {
```

```
    console.error("WebGL не підтримується вашим браузером.");
    return;
  }

  canvas.width = window.innerWidth;
  canvas.height = window.innerHeight;
  gl.viewport(0, 0, canvas.width, canvas.height);

  gl.clearColor(0.99, 0.99, 0.9, 1.0);
  gl.clear(gl.COLOR_BUFFER_BIT);


  const vertexShaderSource = `
    attribute vec4 aVertexPosition;
    attribute vec4 aVertexColor;
    uniform mat4 uModelViewMatrix;
    varying lowp vec4 vColor;
    void main(void) {
      gl_Position = uModelViewMatrix * aVertexPosition;
      vColor = aVertexColor;
    }
  `;
  const fragmentShaderSource = `
    varying lowp vec4 vColor;
    void main(void) {
      gl_FragColor = vColor;
    }
  `;

  const vertexShader = createShader(gl, gl.VERTEX_SHADER,
vertexShaderSource);
  const fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
fragmentShaderSource);

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

```
  const vertexPositionAttribute = gl.getAttribLocation(shaderProgram,
'aVertexPosition');
  const vertexColorAttribute = gl.getAttribLocation(shaderProgram,
'aVertexColor');
  const uModelViewMatrix = gl.getUniformLocation(shaderProgram,
'uModelViewMatrix');

   // Позиції (x, y, z) та кольори (r, g, b, a) для вершини двох трикутників
(квадрат)
   const vertices = new Float32Array([
    -0.5,  0.5,  0.0,  1.0,  0.0,  0.0,  1.0,
    -0.5, -0.5,  0.0,  0.0,  1.0,  0.0,  1.0,
     0.5, -0.5,  0.0,  0.0,  0.0,  1.0,  1.0,

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

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

  gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 28, 0);
  gl.enableVertexAttribArray(vertexPositionAttribute);

  gl.vertexAttribPointer(vertexColorAttribute, 4, gl.FLOAT, false, 28, 12);
  gl.enableVertexAttribArray(vertexColorAttribute);

  let rotation = 0;

  function drawScene()
  {
    gl.clear(gl.COLOR_BUFFER_BIT);

    const ModelViewMatrix = mat4.create();
    mat4.translate(ModelViewMatrix, ModelViewMatrix, [0.0, 0.0, 0.0]);
    mat4.rotate(ModelViewMatrix, ModelViewMatrix, rotation, [0, 0, 1]);
```

```
    gl.uniformMatrix4fv(uModelViewMatrix, false, ModelViewMatrix);
    gl.drawArrays(gl.TRIANGLES, 0, 6);
    rotation += 0.01;

    requestAnimationFrame(drawScene);
  }

  drawScene();
}

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('Помилка компіляції шейдера:',
gl.getShaderInfoLog(shader));
    gl.deleteShader(shader);
    return null;
  }

  return shader;
}

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

  if (!gl.getProgramParameter(program, gl.LINK_STATUS))
  {
```

```
    console.error('Помилка лінкування програми:',
gl.getProgramInfoLog(program));
    gl.deleteProgram(program);
    return null;
  }

  return program;
}

window.onload = setupWebGL;
```

Версія коду для 4-го завдання:

HTML:

```
<!DOCTYPE HTML>
<html>
    <head>
        <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Lab 1 / WebGL setup</title>
        <style>
            body, html { margin: 0; padding: 0; overflow: hidden; }
            canvas { display: block; }
        </style>
    </head>

    <body>
        <canvas id = "mycanvas" width = "600" height = "200"></canvas>
        <script src="https://cdn.jsdelivr.net/npm/gl-matrix@2.8.1/dist/gl-matrix-
min.js"></script>
        <script src="scripts/webgl_setup2.js"></script>
    </body>
</html>
```

JS:

```
function setupWebGL() {
    const canvas = document.getElementById("mycanvas");
    const gl = canvas.getContext("webgl");
```

```
    if (!gl) {
        console.error("WebGL не підтримується вашим браузером.");
        return;
    }

    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
    gl.viewport(0, 0, canvas.width, canvas.height);

    gl.clearColor(0.99, 0.99, 0.9, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);

    const vertexShaderSource = `
        attribute vec4 aVertexPosition;
        attribute vec4 aVertexColor;
        uniform mat4 uModelViewMatrix;
        varying lowp vec4 vColor;
        void main(void) {
            gl_Position = uModelViewMatrix * aVertexPosition;
            vColor = aVertexColor;
        }
    `;

    const fragmentShaderSource = `
        varying lowp vec4 vColor;
        void main(void) {
            gl_FragColor = vColor;
        }
    `;

    const vertexShader = createShader(gl, gl.VERTEX_SHADER,
vertexShaderSource);
    const fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
fragmentShaderSource);

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

```
    const vertexPositionAttribute = gl.getAttribLocation(shaderProgram,
'aVertexPosition');
    const vertexColorAttribute = gl.getAttribLocation(shaderProgram,
'aVertexColor');
    const uModelViewMatrix = gl.getUniformLocation(shaderProgram,
'uModelViewMatrix');

    const circleVertices = [];
    const segments = 30;
    const radius = 0.25;
    const verticalMovementSpeed = 0.01;

    circleVertices.push(0.0, 0.0, 0.0);

    for (let i = 0; i <= segments; i++) {
        const angle = (i * 2 * Math.PI) / segments;
        const x = radius * Math.cos(angle);
        const y = radius * Math.sin(angle);
        circleVertices.push(x, y, 0.0);
    }

    const circleVertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, circleVertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(circleVertices),
gl.STATIC_DRAW);

    const circleColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, circleColorBuffer);

    const circleColors = new Float32Array((segments + 2) * 4);
    for (let i = 0; i < circleColors.length; i += 4) {
        circleColors[i] = 0.3; // R
        circleColors[i + 1] = 0.3; // G
        circleColors[i + 2] = 1.0; // B
        circleColors[i + 3] = 1.0; // A
    }
```

```
    gl.bufferData(gl.ARRAY_BUFFER, circleColors, gl.STATIC_DRAW);

    let verticalMovement = 0.0;
    let direction = 1;

    function drawScene() {
        gl.clear(gl.COLOR_BUFFER_BIT);

        const circleModelViewMatrix = mat4.create();
        mat4.translate(circleModelViewMatrix, circleModelViewMatrix, [0.0,
verticalMovement, 0.0]);
        gl.uniformMatrix4fv(uModelViewMatrix, false, circleModelViewMatrix);

        gl.bindBuffer(gl.ARRAY_BUFFER, circleVertexBuffer);
        gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(vertexPositionAttribute);

        gl.bindBuffer(gl.ARRAY_BUFFER, circleColorBuffer);
        gl.vertexAttribPointer(vertexColorAttribute, 4, gl.FLOAT, false, 0, 0);
        gl.enableVertexAttribArray(vertexColorAttribute);

        gl.drawArrays(gl.TRIANGLE_FAN, 0, circleVertices.length / 3);

        verticalMovement += verticalMovementSpeed * direction;
        if (verticalMovement > 0.5 || verticalMovement < -0.5) {
            direction *= -1;
        }

        requestAnimationFrame(drawScene);
    }

    drawScene();
}

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('Помилка компіляції шейдера:',
gl.getShaderInfoLog(shader));
        gl.deleteShader(shader);
        return null;
    }

    return shader;
}

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

    if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
        console.error('Помилка лінкування програми:',
gl.getProgramInfoLog(program));
        gl.deleteProgram(program);
        return null;
    }

    return program;
}

window.onload = setupWebGL;
```