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: Перевірив:

студент групи ІМ-11 доц.каф.ІПІ   
Царик М.М. Родіонов.П.Ю.

Київ 2024

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.

Хід роботи:

index.html:

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

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

    <link rel="stylesheet" href="styles.css" />

    <title>Lab1</title>

  </head>

  <body>

    <canvas id="glcanvas" width="850" height="500"></canvas>

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

    <script src="main.js" type="module"></script>

  </body>

</html>

styles.css

canvas {

  display: block;

  margin: auto;

}

main.js

import { createBuffers } from "./buffers.js";

import { drawScene } from "./scene.js";

import { createShaderProgram, VSHADER\_SOURCE, FSHADER\_SOURCE } from "./shaders.js";

window.onload = function () {

  main();

};

function main() {

  // Отримуємо посилання на canvas

  const canvas = document.querySelector("#glcanvas");

  // Отримуємо контекст WebGL

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

  if (!gl) {

    // Якщо контекст не існує, виводимо повідомлення про помилку

    alert("Під час ініціалізації WebGL сталася помилка");

    return;

  }

  // Створюємо інформацію про програму шейдерів

  const programInfo = createProgramInfo(gl);

  // Створюємо буфери

  const buffers = createBuffers(gl);

  let previousTime = 0;

  let rotationAngle = 0.0;

  let elapsedTime = 0;

  function render(currentTime) {

    currentTime = currentTime \* 0.001;

    elapsedTime = currentTime - previousTime;

    previousTime = currentTime;

    drawScene(gl, programInfo, buffers, rotationAngle);

    rotationAngle -= elapsedTime;

    requestAnimationFrame(render);

  }

  requestAnimationFrame(render);

}

// Функція для створення інформації про програму шейдерів

function createProgramInfo(gl) {

  // Створюємо програму шейдерів

  const shaderProgram = createShaderProgram(gl, VSHADER\_SOURCE, FSHADER\_SOURCE);

  const programInfo = {

    program: shaderProgram,

    attribLocations: {

      vertexPosition: gl.getAttribLocation(shaderProgram, "aVertexPosition"),

      vertexColor: gl.getAttribLocation(shaderProgram, "aVertexColor"),

    },

    uniformLocations: {

      projectionMatrix: gl.getUniformLocation(shaderProgram, "uProjectionMatrix"),

      modelViewMatrix: gl.getUniformLocation(shaderProgram, "uModelViewMatrix"),

    },

  };

  return programInfo;

}

scene.js

function drawScene(gl, programInfo, buffers, squareRotation) {

  clearBuffers(gl);

  setupProjectionMatrix(gl, programInfo, gl.canvas.clientWidth, gl.canvas.clientHeight);

  drawRotatingSquare(gl, programInfo, buffers.square, squareRotation);

  drawMovingSquare(gl, programInfo, buffers.diamond);

  drawTriangle(gl, programInfo, buffers.triangle1);

  drawTriangle(gl, programInfo, buffers.triangle2);

}

// Функція для очищення буферів

function clearBuffers(gl) {

  gl.clearColor(0.2, 0.2, 0.2, 1.0);

  gl.clearDepth(1.0);

  gl.enable(gl.DEPTH\_TEST);

  gl.depthFunc(gl.LEQUAL);

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

}

// Функція для налаштування матриці проекції

function setupProjectionMatrix(gl, programInfo, width, height) {

  const fieldOfViewRadians = (-55 \* Math.PI) / 180;

  const aspectRatio = width / height;

  const nearPlane = 0.1;

  const farPlane = 100.0;

  const projectionMatrix = mat4.create();

  mat4.perspective(projectionMatrix, fieldOfViewRadians, aspectRatio, nearPlane, farPlane);

  gl.uniformMatrix4fv(programInfo.uniformLocations.projectionMatrix, false, projectionMatrix);

}

// Функція для малювання трикутника

function drawTriangle(gl, programInfo, triangleBuffers) {

  const modelViewMatrix = mat4.create();

  mat4.translate(modelViewMatrix, modelViewMatrix, [2.5, 0.0, -6.0]);

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

  gl.vertexAttribPointer(programInfo.attribLocations.vertexPosition, 3, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition);

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

  gl.vertexAttribPointer(programInfo.attribLocations.vertexColor, 4, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(programInfo.attribLocations.vertexColor);

  gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix, false, modelViewMatrix);

  const vertexCount = 3;

  gl.drawArrays(gl.TRIANGLE\_STRIP, 0, vertexCount);

}

// Функція для малювання обертаючого квадрата

function drawRotatingSquare(gl, programInfo, squareBuffer, squareRotation) {

  const modelViewMatrix = mat4.create();

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

  mat4.rotate(modelViewMatrix, modelViewMatrix, squareRotation, [0, 0, 1]);

  setPositionAttribute(gl, squareBuffer.position, programInfo);

  setColorAttribute(gl, squareBuffer.color, programInfo);

  gl.useProgram(programInfo.program);

  gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix, false, modelViewMatrix);

  const offset = 0;

  const vertexCount = 4;

  gl.drawArrays(gl.TRIANGLE\_STRIP, offset, vertexCount);

}

// Функція для малювання рухомого квадрата

function drawMovingSquare(gl, programInfo, squareBuffer) {

  const modelViewMatrix2 = mat4.create();

  mat4.translate(modelViewMatrix2, modelViewMatrix2, [-2.5, -0.0, -6.0]);

  const translateY = Math.sin(Date.now() \* 0.005) \* 0.9;

  mat4.translate(modelViewMatrix2, modelViewMatrix2, [0.0, translateY, 0.0]);

  setPositionAttribute(gl, squareBuffer.position, programInfo);

  setColorAttribute(gl, squareBuffer.color, programInfo);

  gl.useProgram(programInfo.program);

  gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix, false, modelViewMatrix2);

  {

    const vertexCount = 5;

    gl.drawArrays(gl.TRIANGLE\_FAN, 0, vertexCount);

  }

}

// Функція для встановлення атрибуту позиції

function setPositionAttribute(gl, buffers, programInfo) {

  const numComponents = 2;

  const type = gl.FLOAT;

  const normalize = false;

  const stride = 0;

  const offset = 0;

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

  gl.vertexAttribPointer(

    programInfo.attribLocations.vertexPosition,

    numComponents,

    type,

    normalize,

    stride,

    offset

  );

  gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition);

}

// Функція для встановлення атрибуту кольору

function setColorAttribute(gl, buffers, programInfo) {

  const numComponents = 4;

  const type = gl.FLOAT;

  const normalize = false;

  const stride = 0;

  const offset = 0;

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

  gl.vertexAttribPointer(

    programInfo.attribLocations.vertexColor,

    numComponents,

    type,

    normalize,

    stride,

    offset

  );

  gl.enableVertexAttribArray(programInfo.attribLocations.vertexColor);

}

export { drawScene };

buffers.js

// Функція для створення буферів

function createBuffers(gl) {

  return {

    triangle1: createBuffer(

      gl,

      [-1.0, 0.8, 0.0, -1.0, -1.0, 0.0, 1.5, 0.8, 0.0],

      [1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0]

    ),

    triangle2: createBuffer(

      gl,

      [1.5, 0.8, 0.0, -1.0, -1.0, 0.0, 1.5, -1.0, 0.0],

      [0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0]

    ),

    square: createBuffer(

      gl,

      [0.8, 0.8, -0.8, 0.8, 0.8, -0.8, -0.8, -0.8],

      [0.0, 0.0, 1.0, 1.0, 0.0, 0.5, 1.0, 1.0, 0.0, 0.75, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0]

    ),

    diamond: createBuffer(

      gl,

      [-0.3, 0.1, 0.3, 0.1, 0.6, 0.35, 0.0, 0.9, -0.6, 0.35],

      [

        1.0, 1.0, 1.0, 1.0, 0.7, 0.7, 1.0, 1.0, 0.9, 0.9, 1.0, 1.0, 0.3, 0.3, 1.0, 1.0, 0.9, 0.9,

        1.0, 1.0,

      ]

    ),

  };

}

// Функція для створення буфера

function createBuffer(gl, positions, colors) {

  const positionBuffer = gl.createBuffer();

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

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

  const colorBuffer = gl.createBuffer();

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

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

  return { position: positionBuffer, color: colorBuffer };

}

export { createBuffers };

shaders.js

// Вершинний шейдер

const VSHADER\_SOURCE = `

attribute vec4 aVertexPosition;

attribute vec4 aVertexColor;

uniform mat4 uModelViewMatrix;

uniform mat4 uProjectionMatrix;

varying lowp vec4 vColor;

void main(void){

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

  vColor = aVertexColor;

}

`;

// Фрагментний шейдер

const FSHADER\_SOURCE = `

varying lowp vec4 vColor;

void main(void){

  gl\_FragColor = vColor;

}

`;

// Функція для створення програми шейдерів

function createShaderProgram(gl, VSHADER\_SOURCE, FSHADER\_SOURCE) {

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

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

  const shaderProgram = gl.createProgram();

  gl.attachShader(shaderProgram, vertexShader);

  gl.attachShader(shaderProgram, fragmentShader);

  gl.linkProgram(shaderProgram);

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

    console.error("Під час ініціалізації програми шейдерів виникла помилка");

    return null;

  }

  return shaderProgram;

}

// Функція для завантаження шейдера

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("Під час компіляції шейдера виникла помилка");

    gl.deleteShader(shader);

    return null;

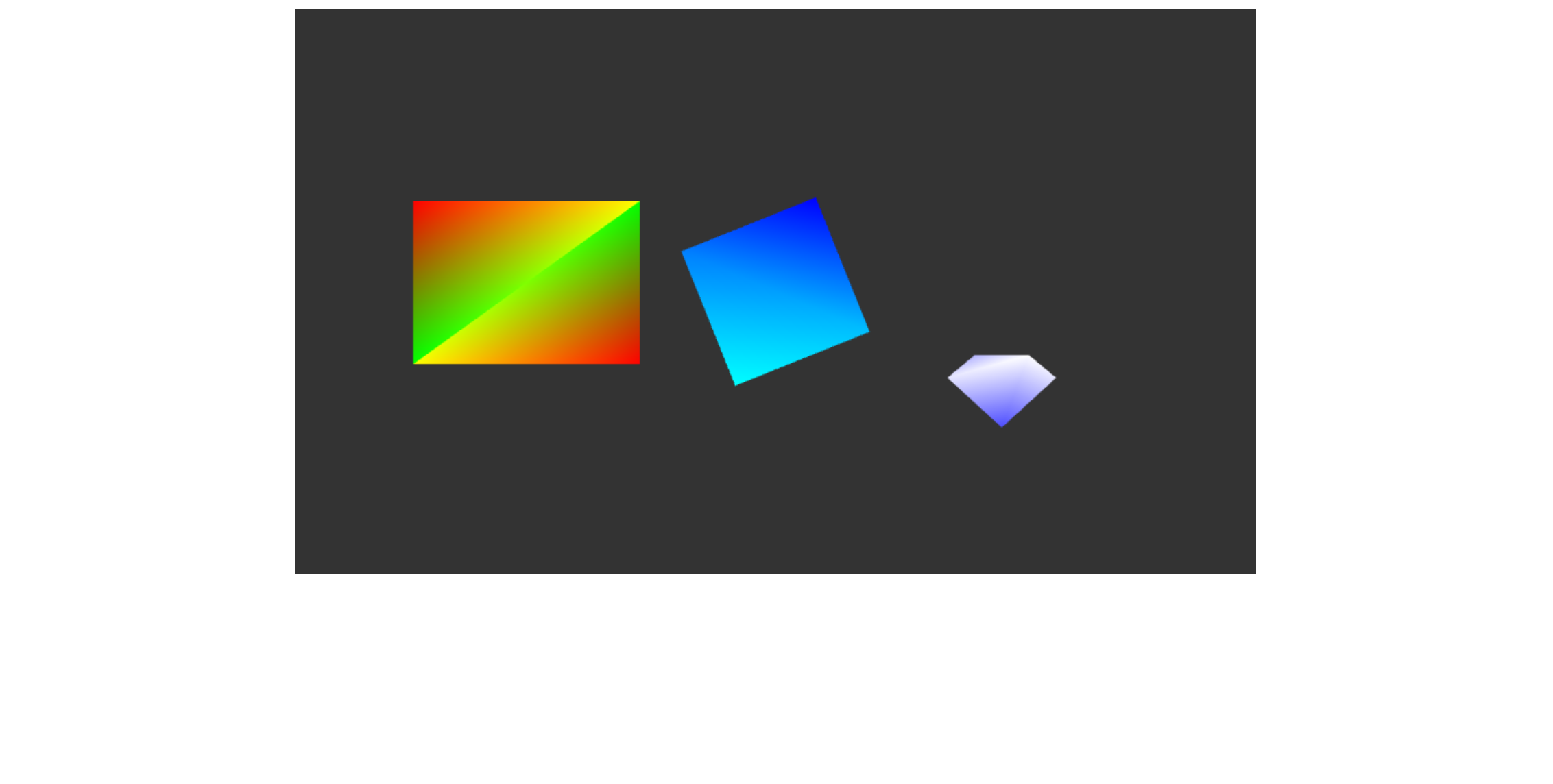
  }

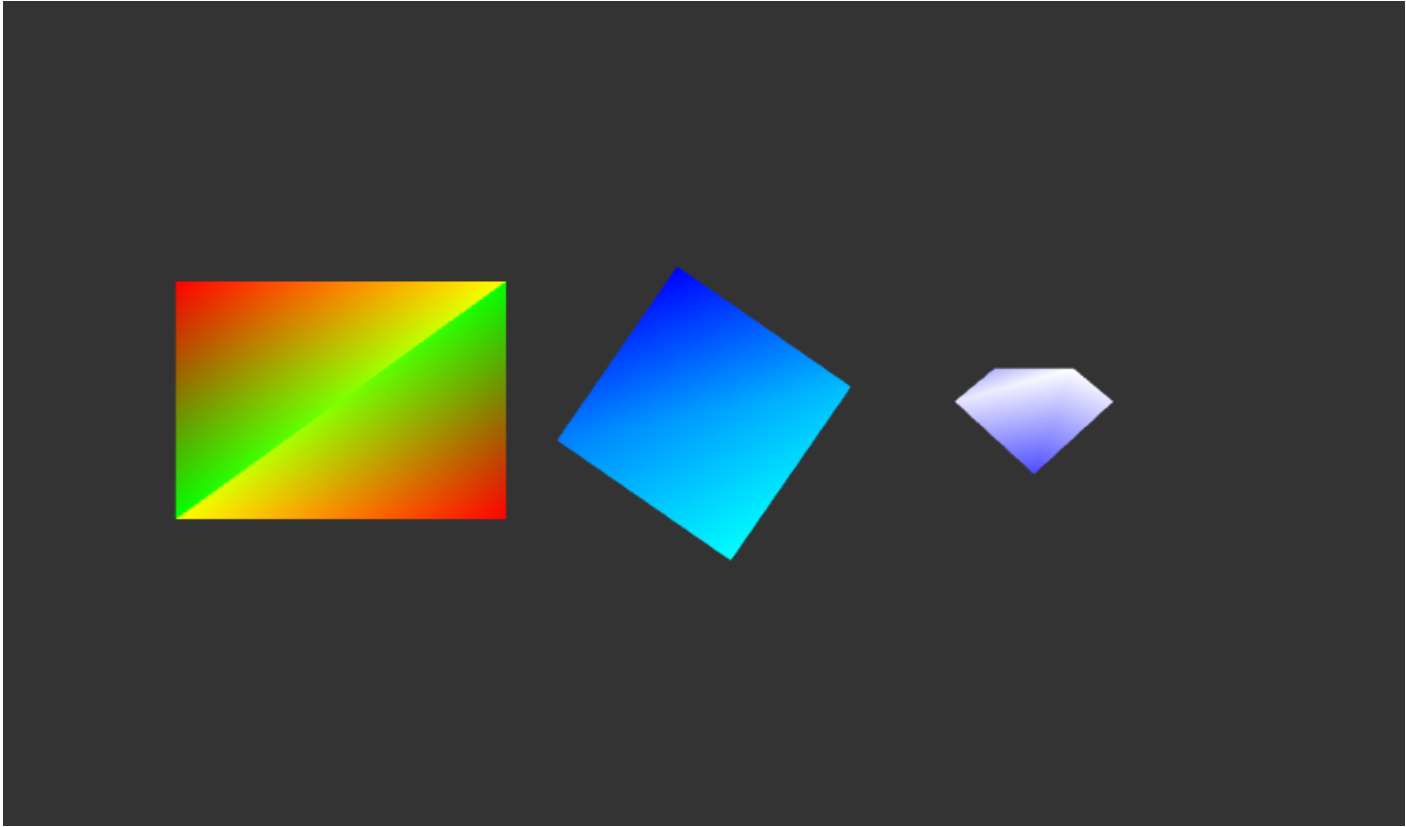
  return shader;

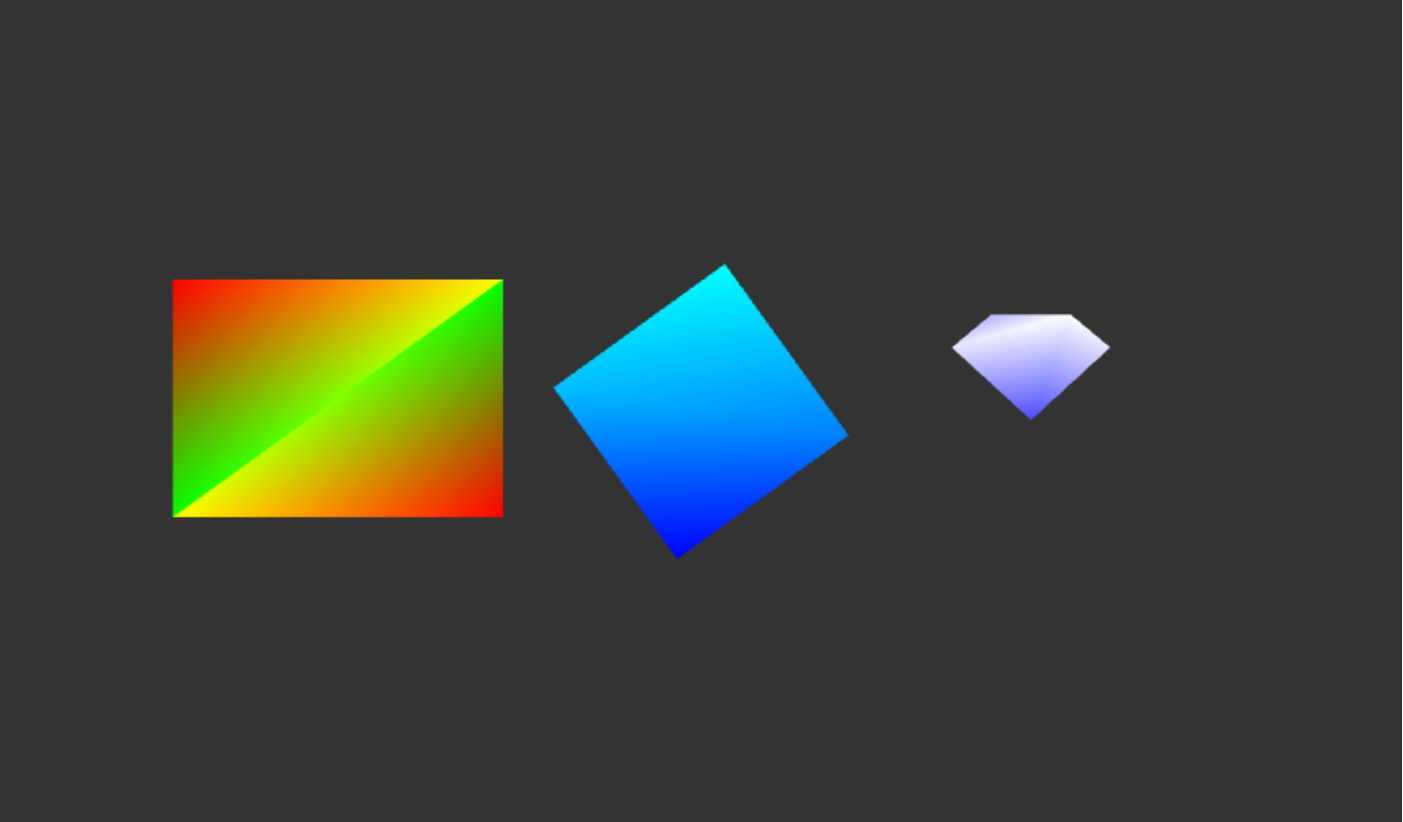
}

export { createShaderProgram, loadShader, VSHADER\_SOURCE, FSHADER\_SOURCE };

Результати виконання:

Скріншот 1 – Перша фігура статична, друга обертається, третя йде вниз

Скріншот 2 – Перша фігура статична, друга обертається, третя йде вгору

Скріншот 3- Перша фігура статична, друга обертається, третя вгорі

Висновки:

Отримати практичні навички програмування WebGL-програм допомогло виконання чотирьох завдань.

1. Під час першого труднощів не виникло, бо більшість матеріалів було пояснено у теоретичних відомостях, що дозволило зрозуміти базові принципи графічних конвеєрів WebGl.
2. У другому завданні основні труднощі викликав етап ознайомлення з API WebGl, він має ширший функціонал в порівнянні з іншими контекстами канвасу, такими як «2d» , але в той же час потребує більший об’єм коду та операцій перед тим як отримати результат.
3. Під час виконання третього завдання виникли труднощі з поворотом квадрату. Проблема була в тому , що разом з ним оберталися и раніше створені трикутники. Це траплялося через те, що я не створював окремий буфер під кожну фігуру, а складав всі координати вершин в один масив. Проблема вирішилася впровадженням окремих буферів для кожного об’єкта.
4. Четверте завдання дозволило вивчити основи переміщення фігур у просторі та створення власних форм, що буде корисно в подальщій роботі з WebGl.

Список використаних джерел:

1. WebGL 2D Rotation by WebGLFundamentals: <https://webglfundamentals.org/webgl/lessons/webgl-2d-rotation.html>
2. WebGL Tutorials by Indigo Code: <https://www.youtube.com/playlist?list=PLjcVFFANLS5zH_PeKC6I8p0Pt1hzph_rt>
3. WebGL by Konstantin Bondarenko: <https://www.youtube.com/playlist?list=PLzt2B3kMUwK_5qn-4ehAFiAnXsWbJb_jj>
4. WebGL Study Note 01: Rotating Triangle by David Guan: <https://www.youtube.com/watch?v=J9sgGpuJ1WA&t=196s>