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

ЗВІТ

ПРО ЛАБОРАТОРНУ РОБОТУ № 3

ТЕМА: «РОБОТА З ПРОЕКЦІЯМИ ТА ТРАНСФОРМАЦІЯМИ»

Виконав: Перевірив:

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

Київ 2024

Мета: отримати практичні навички щодо роботи з проекціями та трансформаціями на основі програмного інтерфейсу WebGL.

Завдання:

1. Створити графічний об’єкт в ортогональній проекції.

2. Представити у перспективній проекції створений у попередньому

завданні графічний об’єкт.

3. Забезпечити використання різних кольорів для різних елементів

графічного об’єкту.

4. Реалізувати для куба анімацію з довільними налаштуваннями.

5. Скласти звіт про виконану роботу.

Хід роботи:  
index.html:

<!DOCTYPE html>

<html>

  <head>

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

    <script type="text/javascript" src="./libs/initShaders.js"></script>

    <script type="text/javascript" src="./libs/MVnew.js"></script>

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

  </head>

  <body>

    <div class="container">

      <div class="inputs">

        <button id="Down">Down</button>

        <button id="Up">Up</button>

        <button id="Left">Left</button>

        <button id="Right">Right</button>

        <button id="Mode">Change mode 🔄</button>

      </div>

      <canvas id="gl-canvas" width="512" height="512">

        Oops ... your browser doesn't support the HTML5 canvas element

      </canvas>

    </div>

  </body>

</html>

styles.css:

@import url("https://fonts.googleapis.com/css2?family=Fredoka:wght@300..700&family=Montserrat&display=swap");

canvas {

  display: flex;

  margin: auto;

}

.inputs {

  display: flex;

  width: fit-content;

  flex-direction: row;

  justify-content: center;

  background-color: #8efd8e;

  border-radius: 15px;

  margin: 10px;

}

button {

  margin: 10px;

  padding: 10px 20px;

  background-color: #1edbfc;

  color: white;

  border: none;

  border-radius: 5px;

  cursor: pointer;

  font-size: 16px;

  transition: background-color 0.3s;

}

button:hover {

  background-color: #4996e8;

}

.container {

  display: flex;

  flex-direction: column;

  align-items: center;

}

a,

button {

  text-decoration: none;

  color: white;

  font-size: 18px;

  font-family: "Fredoka", sans-serif;

  font-weight: 500;

  font-style: normal;

  font-variation-settings: "wdth" 100;

}

main.js:

"use strict";

import { fragmentShader, vertexShader, createShaderProgram } from "./shaders.js";

let projection = "orthogonal";

window.onload = function init() {

  let canvas;

  let gl;

  const numPositions = 36;

  const positionsArray = [];

  const colorsArray = [];

  let near;

  let vertices;

  let far;

  let radius;

  if (projection === "orthogonal") {

    vertices = [

      vec4(-0.5, -0.5, 0.5, 1.0),

      vec4(-0.5, 0.5, 0.5, 1.0),

      vec4(0.5, 0.5, 0.5, 1.0),

      vec4(0.5, -0.5, 0.5, 1.0),

      vec4(-0.5, -0.5, -0.5, 1.0),

      vec4(-0.5, 0.5, -0.5, 1.0),

      vec4(0.5, 0.5, -0.5, 1.0),

      vec4(0.5, -0.5, -0.5, 1.0),

    ];

    near = -1;

    far = 1;

    radius = 0.1;

  } else {

    vertices = [

      vec4(-0.5, -0.5, 1.5, 1.0),

      vec4(-0.5, 0.5, 1.5, 1.0),

      vec4(0.5, 0.5, 1.5, 1.0),

      vec4(0.5, -0.5, 1.5, 1.0),

      vec4(-0.5, -0.5, 0.5, 1.0),

      vec4(-0.5, 0.5, 0.5, 1.0),

      vec4(0.5, 0.5, 0.5, 1.0),

      vec4(0.5, -0.5, 0.5, 1.0),

    ];

    near = 1.2;

    far = 12;

    radius = 4.0;

  }

  const vertexColors = [

    vec4(0.5, 0.0, 0.5, 1),

    vec4(1.0, 0.5, 0.0, 1),

    vec4(1.0, 1.0, 0.0, 1),

    vec4(0.0, 0.3, 1.0, 1),

    vec4(0.5, 0.0, 0.5, 1),

    vec4(1.0, 0.5, 0.0, 1),

    vec4(1.0, 1.0, 0.0, 1),

    vec4(0.0, 0.3, 1.0, 1),

  ];

  let theta = 0.0; // Кут повороту навколо осі Y

  let phi = 0.0; // Кут повороту навколо осі X

  const dr = (5.0 \* Math.PI) / 180.0; // Крок зміни угла

  const left = -1.0;

  const right = 1.0;

  const top = 1.0;

  const bottom = -1.0;

  const fovy = 45.0; //Кут нахилу

  let aspect; // Співвідношення ширини канвасу до його висоти

  let modelViewMatrix, projectionMatrix;

  let modelViewMatrixLoc, projectionMatrixLoc;

  let eye; // Положення камери

  const at = vec3(0.0, 0.0, 0.0); //Точка куди дивиться камера

  const up = vec3(0.0, 1.0, 0.0); // Напрямок "вгору" для камери

  const downButton = document.getElementById("Down");

  const upButton = document.getElementById("Up");

  const leftButton = document.getElementById("Left");

  const rightButton = document.getElementById("Right");

  const modeButton = document.getElementById("Mode");

  canvas = document.getElementById("gl-canvas");

  gl = canvas.getContext("webgl2");

  if (!gl) alert("WebGL 2.0 isn't available");

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

  aspect = canvas.width / canvas.height; //???

  gl.clearColor(1.0, 1.0, 1.0, 1.0);

  gl.enable(gl.DEPTH\_TEST);

  gl.clearColor(0.8, 0.8, 0.8, 0.87);

  const program = createShaderProgram(gl, vertexShader, fragmentShader);

  gl.useProgram(program);

  colorCube();

  const cBuffer = gl.createBuffer();

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

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

  const colorLoc = gl.getAttribLocation(program, "vColor");

  gl.vertexAttribPointer(colorLoc, 4, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(colorLoc);

  const vBuffer = gl.createBuffer();

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

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

  const positionLoc = gl.getAttribLocation(program, "vPosition");

  gl.vertexAttribPointer(positionLoc, 4, gl.FLOAT, false, 0, 0);

  gl.enableVertexAttribArray(positionLoc);

  modelViewMatrixLoc = gl.getUniformLocation(program, "uModelViewMatrix");

  projectionMatrixLoc = gl.getUniformLocation(program, "uProjectionMatrix");

  let interval1, interval2, interval3, interval4;

  downButton.onmousedown = function () {

    interval1 = setInterval(function () {

      theta += dr;

    }, 50);

  };

  downButton.onmouseup = function () {

    clearInterval(interval1);

  };

  upButton.onmousedown = function () {

    interval2 = setInterval(function () {

      theta -= dr;

    }, 50);

  };

  upButton.onmouseup = function () {

    clearInterval(interval2);

  };

  leftButton.onmousedown = function () {

    interval3 = setInterval(function () {

      phi += dr;

    }, 50);

  };

  leftButton.onmouseup = function () {

    clearInterval(interval3);

  };

  rightButton.onmousedown = function () {

    interval4 = setInterval(function () {

      phi -= dr;

    }, 50);

  };

  rightButton.onmouseup = function () {

    clearInterval(interval4);

  };

  modeButton.onclick = function () {

    toggleProjection();

  };

  render();

  function colorCube() {

    quad(1, 0, 3, 2, vertexColors[1], vertexColors[0], vertexColors[3], vertexColors[2]);

    quad(2, 3, 7, 6, vertexColors[2], vertexColors[3], vertexColors[7], vertexColors[6]);

    quad(3, 0, 4, 7, vertexColors[3], vertexColors[0], vertexColors[4], vertexColors[7]);

    quad(6, 5, 1, 2, vertexColors[6], vertexColors[5], vertexColors[1], vertexColors[2]);

    quad(4, 5, 6, 7, vertexColors[4], vertexColors[5], vertexColors[6], vertexColors[7]);

    quad(5, 4, 0, 1, vertexColors[5], vertexColors[4], vertexColors[0], vertexColors[1]);

  }

  function quad(a, b, c, d, colorA, colorB, colorC, colorD) {

    positionsArray.push(vertices[a]);

    colorsArray.push(colorA);

    positionsArray.push(vertices[b]);

    colorsArray.push(colorB);

    positionsArray.push(vertices[c]);

    colorsArray.push(colorC);

    positionsArray.push(vertices[a]);

    colorsArray.push(colorA);

    positionsArray.push(vertices[c]);

    colorsArray.push(colorC);

    positionsArray.push(vertices[d]);

    colorsArray.push(colorD);

  }

  function toggleProjection() {

    projection = projection === "orthogonal" ? "perspective" : "orthogonal";

    init();

  }

  function render() {

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

    eye = vec3(radius \* Math.sin(phi), radius \* Math.sin(theta), radius \* Math.cos(phi));

    modelViewMatrix = lookAt(eye, at, up);

    if (projection === "orthogonal") {

      projectionMatrix = ortho(left, right, bottom, top, near, far);

    } else {

      projectionMatrix = perspective(fovy, aspect, near, far);

    }

    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(modelViewMatrix));

    gl.uniformMatrix4fv(projectionMatrixLoc, false, flatten(projectionMatrix));

    gl.drawArrays(gl.TRIANGLES, 0, numPositions);

    requestAnimationFrame(render);

  }

};

shaders.js:

const vertexShader = `

attribute vec4 vPosition;

attribute vec4 vColor;

varying vec4 fColor;

uniform mat4 uModelViewMatrix;

uniform mat4 uProjectionMatrix;

void main() {

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

    fColor = vColor;

}

`;

const fragmentShader = `

precision mediump float;

varying vec4 fColor;

void main() {

    gl\_FragColor = fColor;

}

`;

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);

  // Перевірка на успішність лінкування

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

    console.error("An error occurred while initializing the shader program");

    return null;

  }

  return shaderProgram;

}

function loadShader(gl, type, source) {

  const shader = gl.createShader(type);

  gl.shaderSource(shader, source); //Вказуємо джерело

  gl.compileShader(shader); // Компілюємо

  // Перевірка на успішність компіляції

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

    console.error("An error occurred while compiling a shader");

    gl.deleteShader(shader);

    return null;

  }

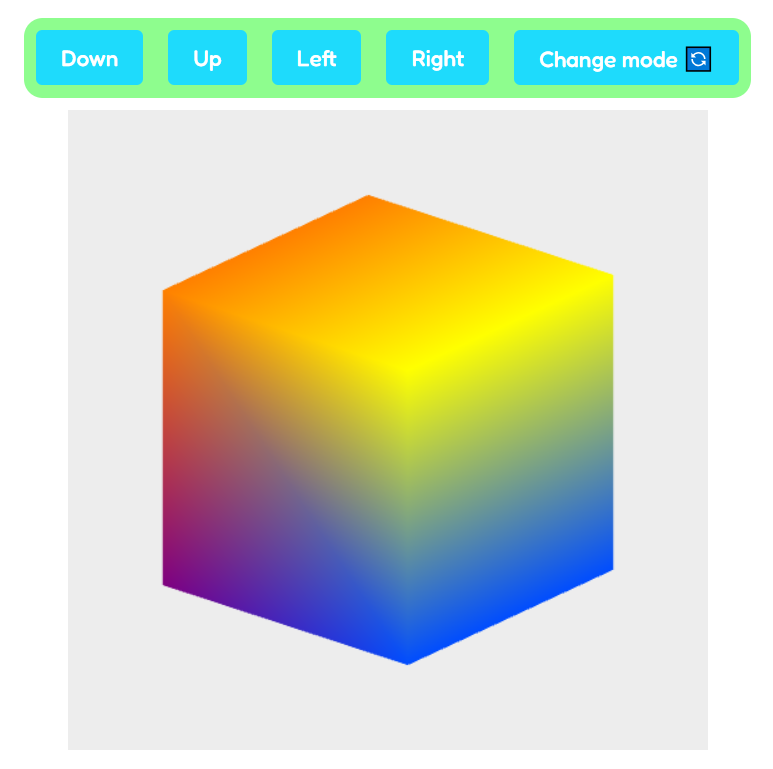
  return shader;

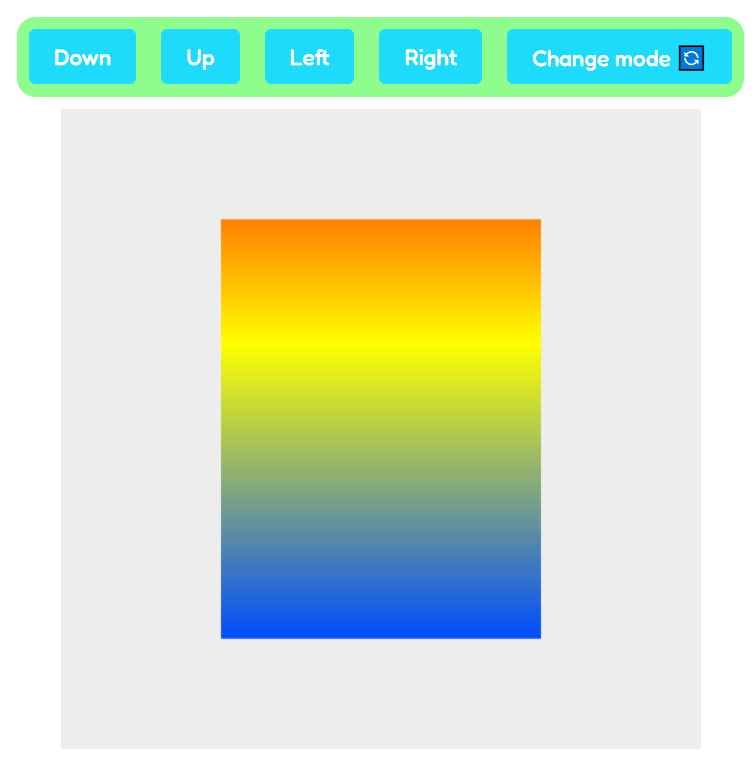
}

export { createShaderProgram, fragmentShader, vertexShader };

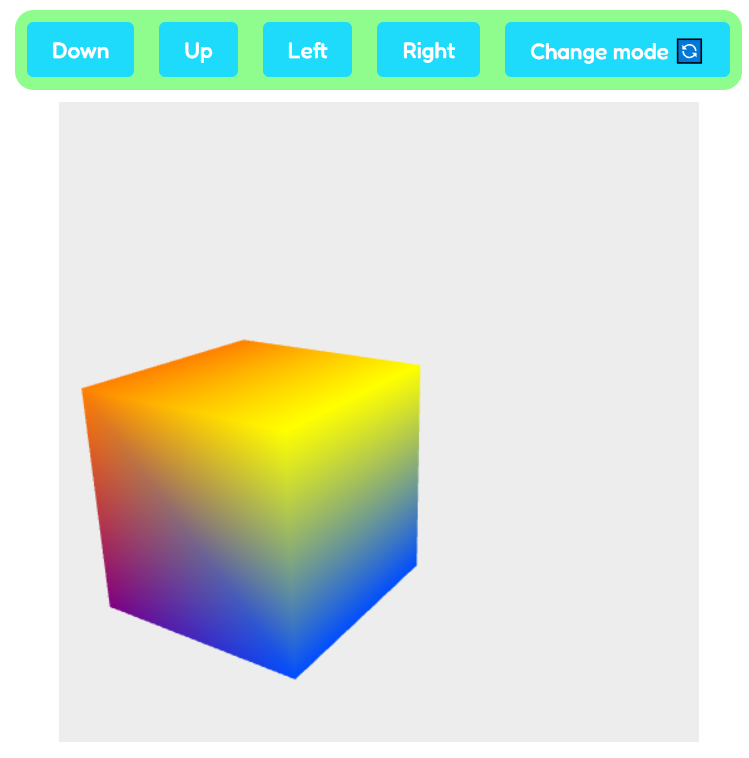
Результати виконання:

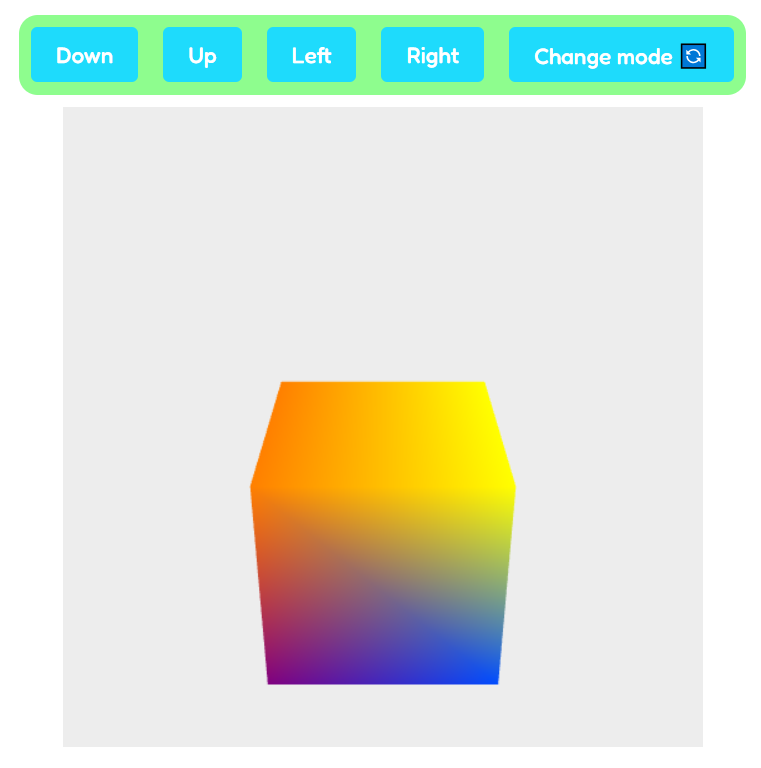
Ортогональна проэкція:





Перспективна проєкція:





Висновки:

Під час виконання лабораторної роботи було успішно створено графічний куб у ортогональній проекції, використовуючи відповідні матриці та кольори для кожної сторони. Куб було перетворено до перспективної проекції, забезпечуючи коректне відображення у тривимірному просторі. Для різних сторін куба були використані різні кольори, що дозволило краще виділити їх. Реалізована анімація обертання куба, що дозволяє користувачеві взаємодіяти з об'єктом та спостерігати його з різних кутів. Загалом, ця робота дозволила закріпити знання отримані раніше (поєднання JS і WebGL, робота з шейдерами та буферами) та навчила працювати з комп’ютерною графікою у тривимірному просторі.

Список використаних джерел:

1. “WebGL Tutorial 02 - Rotating 3D Cube”: <https://www.youtube.com/watch?v=3yLL9ADo-ko>
2. WebGL - Cube Rotation by tutorialspoint: <https://www.tutorialspoint.com/webgl/webgl_cube_rotation.htm>
3. WebGL Course by The University of New Mexico: <https://www.youtube.com/playlist?list=PLAmORN9Zcs_aI30--W7Ckgi3SGa5aAuJE>