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:
Студент групи ІП-22
Підпанюк В.А.

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

Київ 2024

Лабораторна робота №3

РОБОТА З ПРОЕКЦІЯМИ ТА ТРАНСФОРМАЦІЯМИ

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.

1. Створення об’єкта

У даному пункті ми створюємо первинну фігуру в ортогональній проекції, для виконання роботи було обрано куб (рис. 1.1).

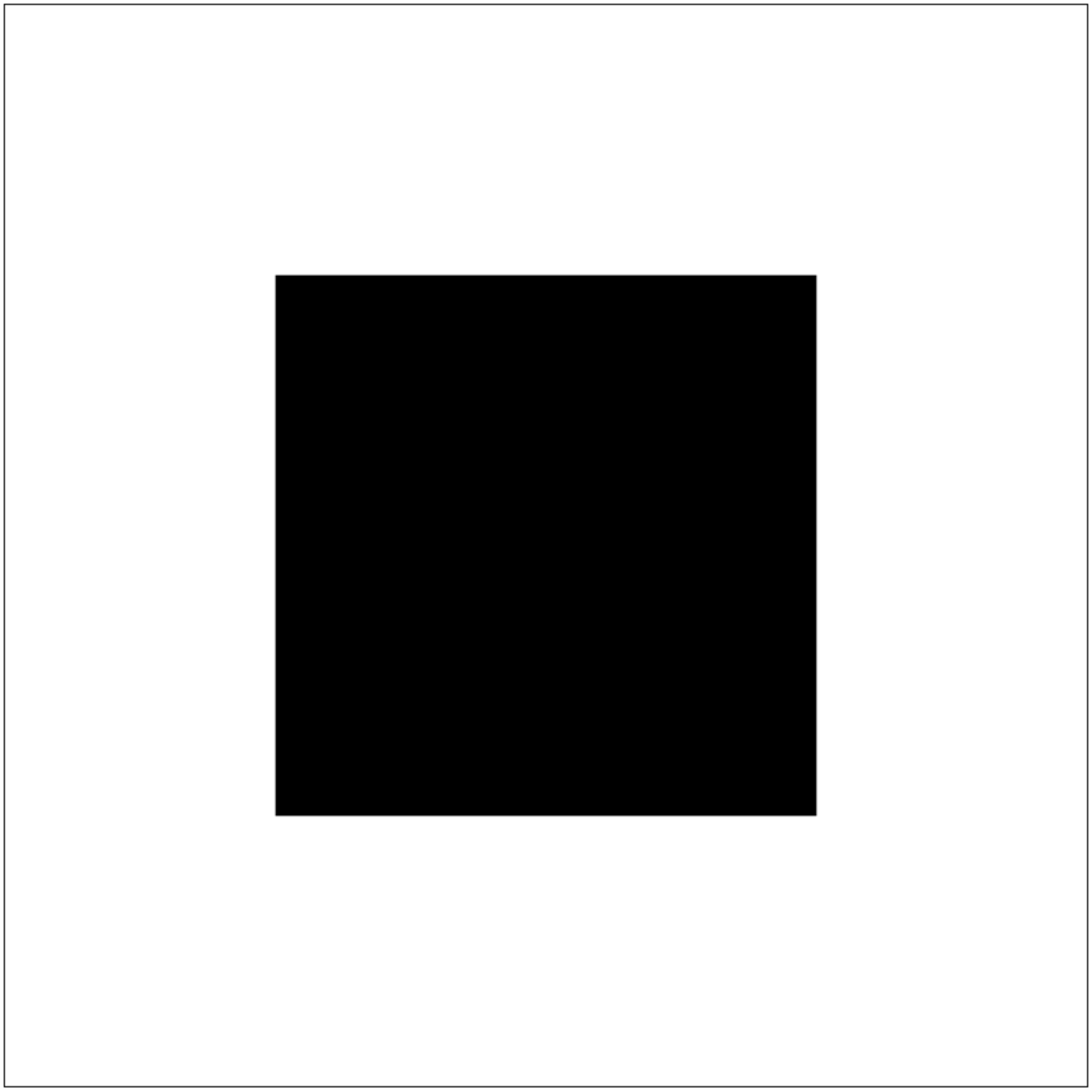

рис. 1.1. Ортогональна проекція куба

2. Представлення у перспективній проекції

У даному пункті ми повертаємо попередньо створену фігуру за одною з осей для демонстрації перспективної проекції даної фігури (рис. 2.1).

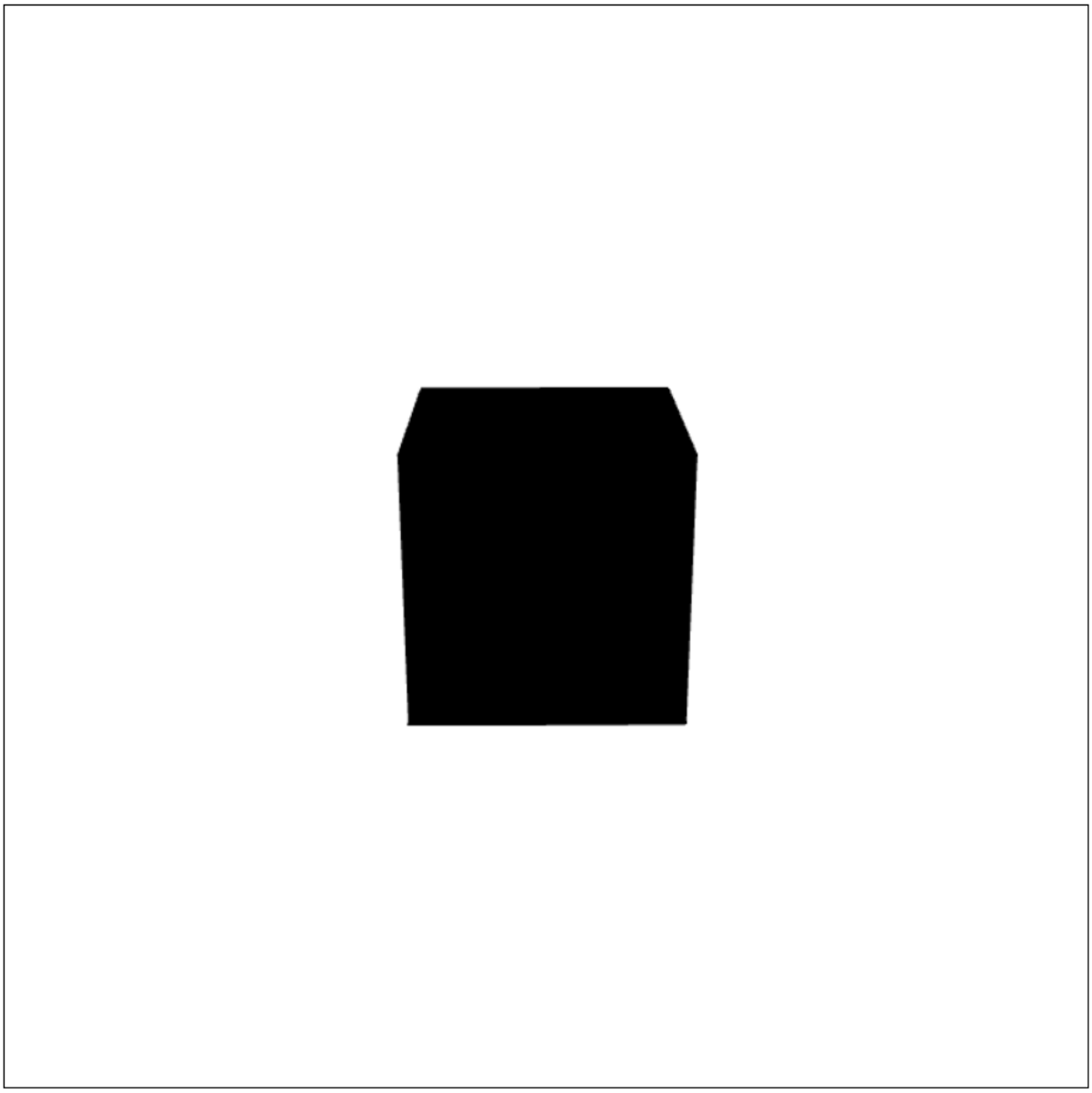

рис. 2.1. Перспективна проекція куба

3. Розфарбування графічного об’єкта

У даному пункті ми надаємо частинам кубу різних кольорів для того щоб чітко розрізняти у якому положенні він знаходиться (рис. 3.1).

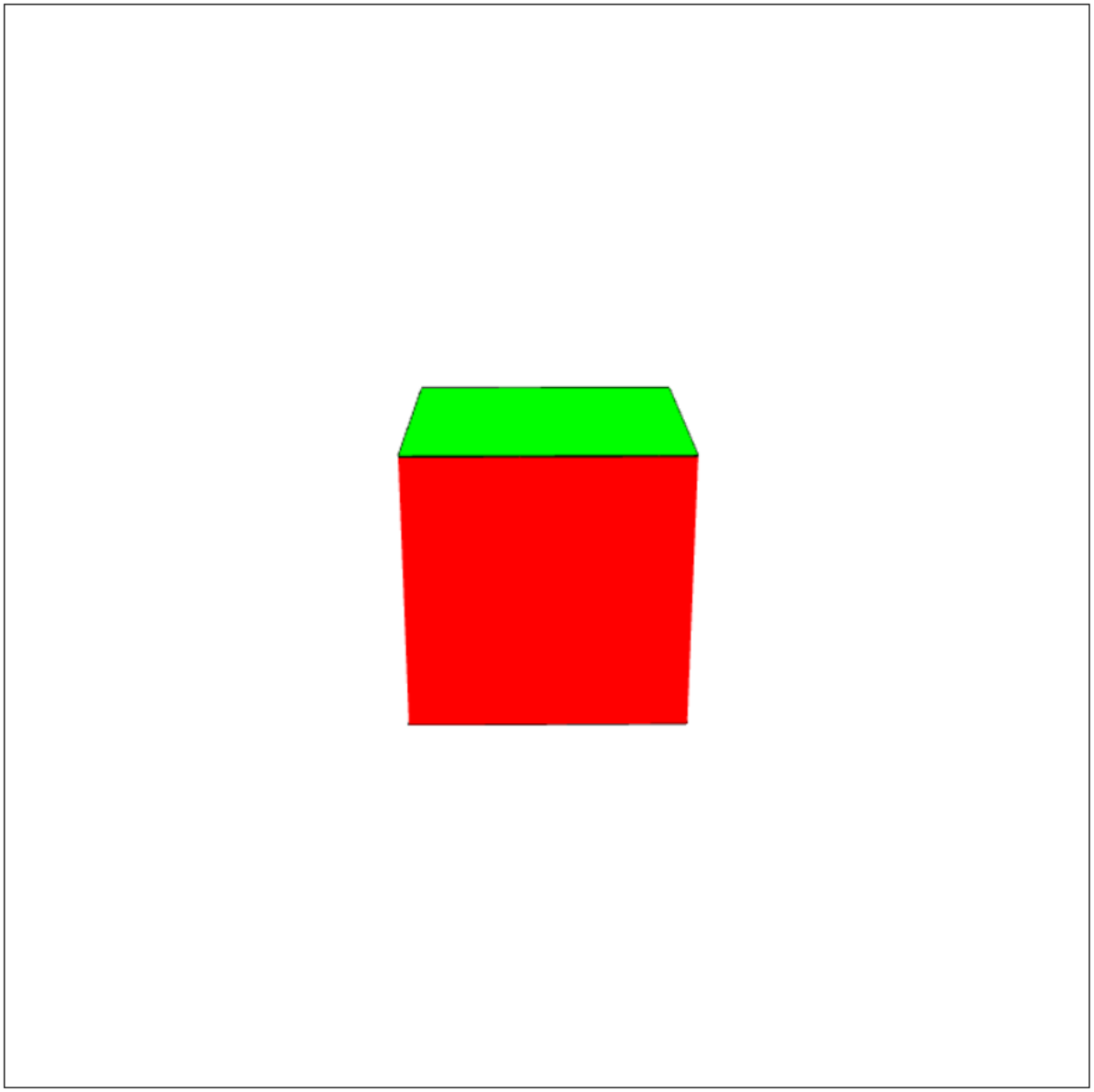

рис. 3.1. Перспективна проекція розфарбованого куба

4. Створення анімації для графічного об’єкта

У даному пункті ми створюємо для куба анімацію обертання навколо осі **У** (рис. 4.1).

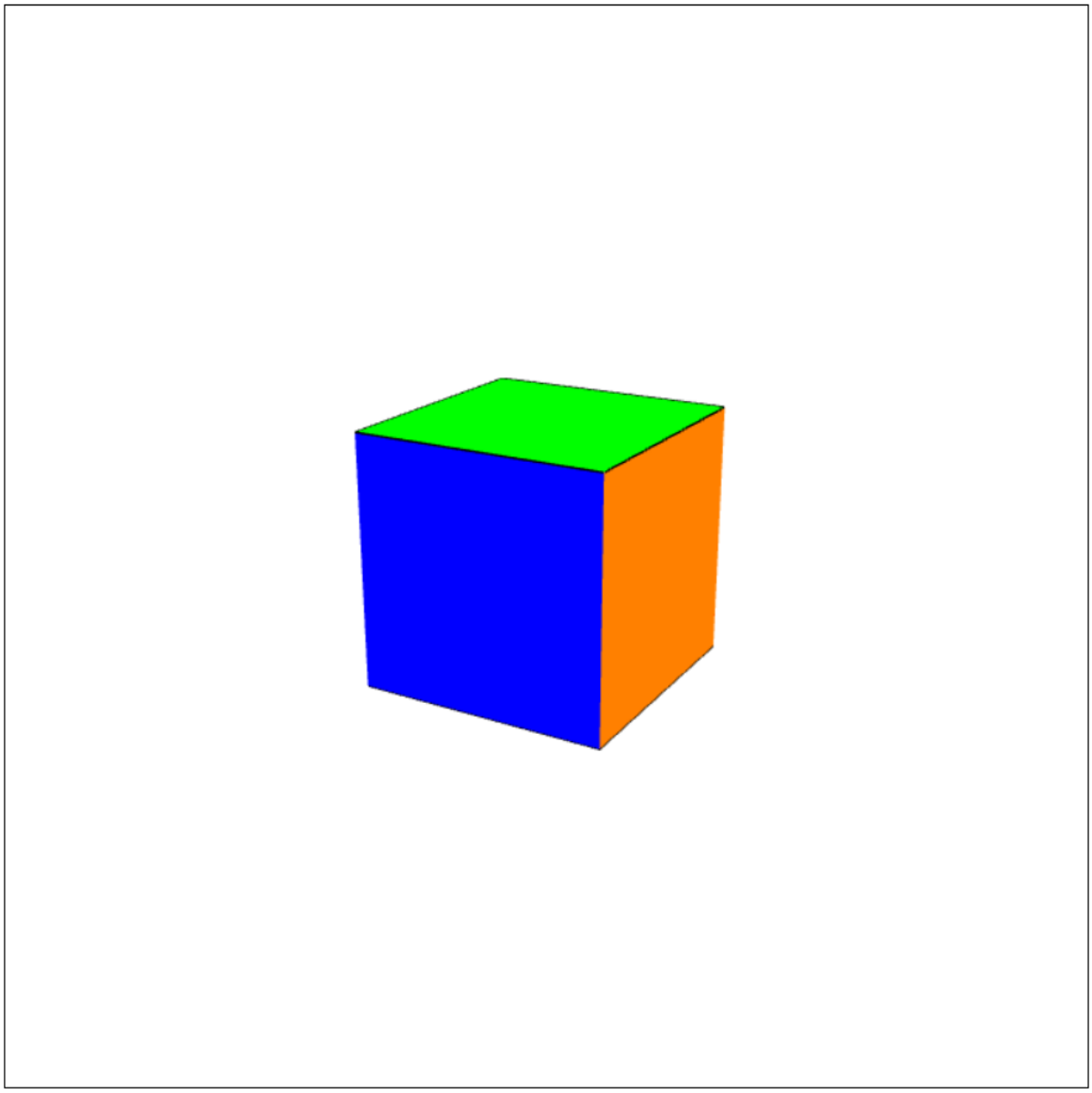

рис. 4.1. Перспективна проекція куба з анімацією обертання

Висновок: У ході виконання даної роботи ми створили графічний об’єкт — куб — у двох проекціях: ортогональній та перспективній. Спочатку об'єкт був побудований в ортогональній проекції, що забезпечило точне представлення розмірів без урахування перспективи. Далі цей же куб був відображений у перспективній проекції, що дозволило отримати більш реалістичне тривимірне зображення об'єкта. Кожна грань куба була зафарбована різним кольором, відповідно до вимог завдання. На завершення ми реалізували анімацію, яка забезпечує плавне обертання куба в просторі, демонструючи різні аспекти його проекції.

ПРОГРАМНИЙ КОД

Код для ортогональної проекції:

HTML:

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Lab 2 / User interaction</title>
      <style>
         body, html { margin: 0; padding: 0; overflow: hidden; }
         canvas { display: block; border: 1px solid black; margin: 3%;}
      </style>
   </head>

   <body>
      <canvas id = "mycanvas" width = "700" height = "700"></canvas>
      <script src="https://cdn.jsdelivr.net/npm/gl-matrix@2.8.1/dist/gl-matrix-
min.js"></script>
      <!-- <script src="scripts/perspective.js"></script> -->
      <script src="scripts/ortogonal.js"></script>
   </body>
</html>
```

JS:

```
const canvas = document.getElementById('mycanvas');
const gl = canvas.getContext('webgl');

if (!gl) {
   alert("WebGL не підтримується в цьому браузері");
}

const vertexShaderSource = `
   attribute vec4 a_position;
   uniform mat4 u_matrix;
```

```
    void main() {
        gl_Position = u_matrix * a_position;
    }
`;

const fragmentShaderSource = `
    precision mediump float;
    void main() {
        gl_FragColor = vec4(0, 0, 0, 1); // чорний колір
    }
`;

function createShader(gl, type, source) {
    const shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    const success = gl.getShaderParameter(shader, gl.COMPILE_STATUS);
    if (success) {
        return shader;
    }
    console.log(gl.getShaderInfoLog(shader));
    gl.deleteShader(shader);
}

function createProgram(gl, vertexShader, fragmentShader) {
    const program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);
    const success = gl.getProgramParameter(program, gl.LINK_STATUS);
    if (success) {
        return program;
    }
    console.log(gl.getProgramInfoLog(program));
    gl.deleteProgram(program);
}
```

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

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

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

const positions = [
   -0.5, -0.5,  0.5,
    0.5, -0.5,  0.5,
    0.5,  0.5,  0.5,
   -0.5,  0.5,  0.5,

   -0.5, -0.5, -0.5,
    0.5, -0.5, -0.5,
    0.5,  0.5, -0.5,
   -0.5,  0.5, -0.5
];

gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions),
gl.STATIC_DRAW);

function createOrthographic(left, right, bottom, top, near, far) {
   return [
      2 / (right - left), 0, 0, 0,
      0, 2 / (top - bottom), 0, 0,
      0, 0, 2 / (near - far), 0,
      -(right + left) / (right - left), -(top + bottom) / (top - bottom), -(far + near) /
(far - near), 1,
   ];
}

const orthographicMatrix = createOrthographic(-1, 1, -1, 1, -1, 1);
```

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

   gl.useProgram(program);

   const positionLocation = gl.getAttribLocation(program, "a_position");
   const matrixLocation = gl.getUniformLocation(program, "u_matrix");

   gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);

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

   gl.uniformMatrix4fv(matrixLocation, false, orthographicMatrix);

   gl.drawArrays(gl.TRIANGLE_FAN, 0, 4);
}

gl.clearColor(1, 1, 1, 1);
drawScene();
```

Код для перспективної проекції:

HTML:

```
<!DOCTYPE HTML>
<html>
   <head>
      <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Lab 2 / User interaction</title>
      <style>
         body, html { margin: 0; padding: 0; overflow: hidden; }
         canvas { display: block; border: 1px solid black; margin: 3%;}
      </style>
   </head>
```

```
    <body>
        <canvas id = "mycanvas" width = "700" height = "700"></canvas>
        <script src="https://cdn.jsdelivr.net/npm/gl-matrix@2.8.1/dist/gl-matrix-
min.js"></script>
        <!-- <script src="scripts/perspective.js"></script> -->
        <script src="scripts/perspective.js"></script>
    </body>
</html>
```

JS:

```
const canvas = document.getElementById('mycanvas');
const gl = canvas.getContext('webgl');

if (!gl) {
    alert("WebGL не підтримується в цьому браузері");
}

const vertexShaderSource = `
    attribute vec4 a_position;
    attribute vec4 a_color;
    varying vec4 v_color;
    uniform mat4 u_matrix;

    void main() {
        gl_Position = u_matrix * a_position;
        v_color = a_color;
    }
`;

const fragmentShaderSource = `
    precision mediump float;
    varying vec4 v_color;

    void main() {
        gl_FragColor = v_color;
    }
```

```
`;

function createShader(gl, type, source) {
    const shader = gl.createShader(type);
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    const success = gl.getShaderParameter(shader, gl.COMPILE_STATUS);
    if (success) {
        return shader;
    }
    console.log(gl.getShaderInfoLog(shader));
    gl.deleteShader(shader);
}

function createProgram(gl, vertexShader, fragmentShader) {
    const program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);
    const success = gl.getProgramParameter(program, gl.LINK_STATUS);
    if (success) {
        return program;
    }
    console.log(gl.getProgramInfoLog(program));
    gl.deleteProgram(program);
}

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

const positions = [
    -0.2, -0.2,  0.2,
     0.2, -0.2,  0.2,
     0.2,  0.2,  0.2,
```

-0.2, -0.2, 0.2,
 0.2, 0.2, 0.2,
-0.2, 0.2, 0.2,

-0.2, -0.2, -0.2,
 0.2, -0.2, -0.2,
 0.2, 0.2, -0.2,
-0.2, -0.2, -0.2,
 0.2, 0.2, -0.2,
-0.2, 0.2, -0.2,

-0.2, 0.2, 0.2,
 0.2, 0.2, 0.2,
 0.2, 0.2, -0.2,
-0.2, 0.2, 0.2,
 0.2, 0.2, -0.2,
-0.2, 0.2, -0.2,

-0.2, -0.2, 0.2,
 0.2, -0.2, 0.2,
 0.2, -0.2, -0.2,
-0.2, -0.2, 0.2,
 0.2, -0.2, -0.2,
-0.2, -0.2, -0.2,

-0.2, -0.2, 0.2,
-0.2, 0.2, 0.2,
-0.2, 0.2, -0.2,
-0.2, -0.2, 0.2,
-0.2, 0.2, -0.2,
-0.2, -0.2, -0.2,

 0.2, -0.2, 0.2,
 0.2, 0.2, 0.2,
 0.2, 0.2, -0.2,
 0.2, -0.2, 0.2,
 0.2, 0.2, -0.2,

```
     0.2, -0.2, -0.2
];

const faceColors = [
    [1.0, 0.0, 0.0, 1.0],
    [0.0, 0.0, 1.0, 1.0],
    [0.0, 1.0, 0.0, 1.0],
    [1.0, 1.0, 0.0, 1.0],
    [1.0, 0.5, 0.0, 1.0],
    [0.5, 0.5, 0.5, 1.0]
].flatMap(color => Array(6).fill(color).flat());


const linePositions = [
    -0.2, -0.2,  0.2,
     0.2, -0.2,  0.2,
     0.2,  0.2,  0.2,
    -0.2,  0.2,  0.2,

    -0.2, -0.2, -0.2,
     0.2, -0.2, -0.2,
     0.2,  0.2, -0.2,
    -0.2,  0.2, -0.2,

    -0.2, -0.2,  0.2,
    -0.2, -0.2, -0.2,
     0.2, -0.2,  0.2,
     0.2, -0.2, -0.2,
     0.2,  0.2,  0.2,
     0.2,  0.2, -0.2,
    -0.2,  0.2,  0.2,
    -0.2,  0.2, -0.2,
];

const edgeColors = new Array(24).fill([0.0, 0.0, 0.0, 1.0]).flat();

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(faceColors),
gl.STATIC_DRAW);

const lineBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, lineBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(linePositions),
gl.STATIC_DRAW);

const lineColorBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, lineColorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(edgeColors),
gl.STATIC_DRAW);

function setAttribute(buffer, attribute, size) {
   gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
   const location = gl.getAttribLocation(program, attribute);
   gl.enableVertexAttribArray(location);
   gl.vertexAttribPointer(location, size, gl.FLOAT, false, 0, 0);
}

const fieldOfViewRadians = Math.PI / 4;
const aspect = canvas.width / canvas.height;
const zNear = 0.1;
const zFar = 100.0;
const projectionMatrix = mat4.create();
mat4.perspective(projectionMatrix, fieldOfViewRadians, aspect, zNear, zFar);

let rotationY = 0;
function drawScene() {
   gl.clearColor(1.0, 1.0, 1.0, 1.0);
   gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
```

```
    gl.enable(gl.DEPTH_TEST);

    rotationY += 0.01;

    const modelMatrix = mat4.create();
    mat4.translate(modelMatrix, modelMatrix, [0, 0, -2]);
    mat4.rotateX(modelMatrix, modelMatrix, Math.PI / 9);
    mat4.rotateY(modelMatrix, modelMatrix, rotationY);

    const mvpMatrix = mat4.create();
    mat4.multiply(mvpMatrix, projectionMatrix, modelMatrix);

    gl.useProgram(program);
    setAttribute(positionBuffer, 'a_position', 3);
    setAttribute(colorBuffer, 'a_color', 4);
    gl.uniformMatrix4fv(gl.getUniformLocation(program, 'u_matrix'), false,
mvpMatrix);
    gl.drawArrays(gl.TRIANGLES, 0, positions.length / 3);

    setAttribute(lineBuffer, 'a_position', 3);
    setAttribute(lineColorBuffer, 'a_color', 4);
    gl.drawArrays(gl.LINES, 0, linePositions.length / 3);

    requestAnimationFrame(drawScene);
}

requestAnimationFrame(drawScene);
```