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

ЗВІТ

ПРО ЛАБОРАТОРНУ РОБОТУ №4

ТЕМА: « ПЕРСПЕКТИВА ТА ТРАНСФОРМАЦІЯ КАМЕРИ »

Виконав:
Студент групи ІП-22
Підпанюк В.А.

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

Київ 2024

Лабораторна робота №4

ПЕРСПЕКТИВА ТА ТРАНСФОРМАЦІЯ КАМЕРИ

Мета: поглибити теоретичні знання та практичні навички щодо роботи з режимами перегляду у WebGL.

Завдання:

1. Створити ящик з кришкою. Ящик і кришка мають однакові координати; тільки кришка є масштабованою версією ящика.
2. Застосувати довільні кольори.
3. Застосувати окремі шейдерні програми для ящика та кришки. Застосувати до них перспективну проекцію.
4. При кожному натисканні клавіш зі стрілками вліво та вправо ящик разом із кришкою має повертатися вздовж осі Y.
5. Після кожного натискання клавіш вгору та вниз відкриватиметься та закриватиметься лише кришка.

1. Створення графічного об’єкта

У даному пункті створюємо кубічну коробку та кришку для неї за заданими у завданні параметрами (рис. 1.1).

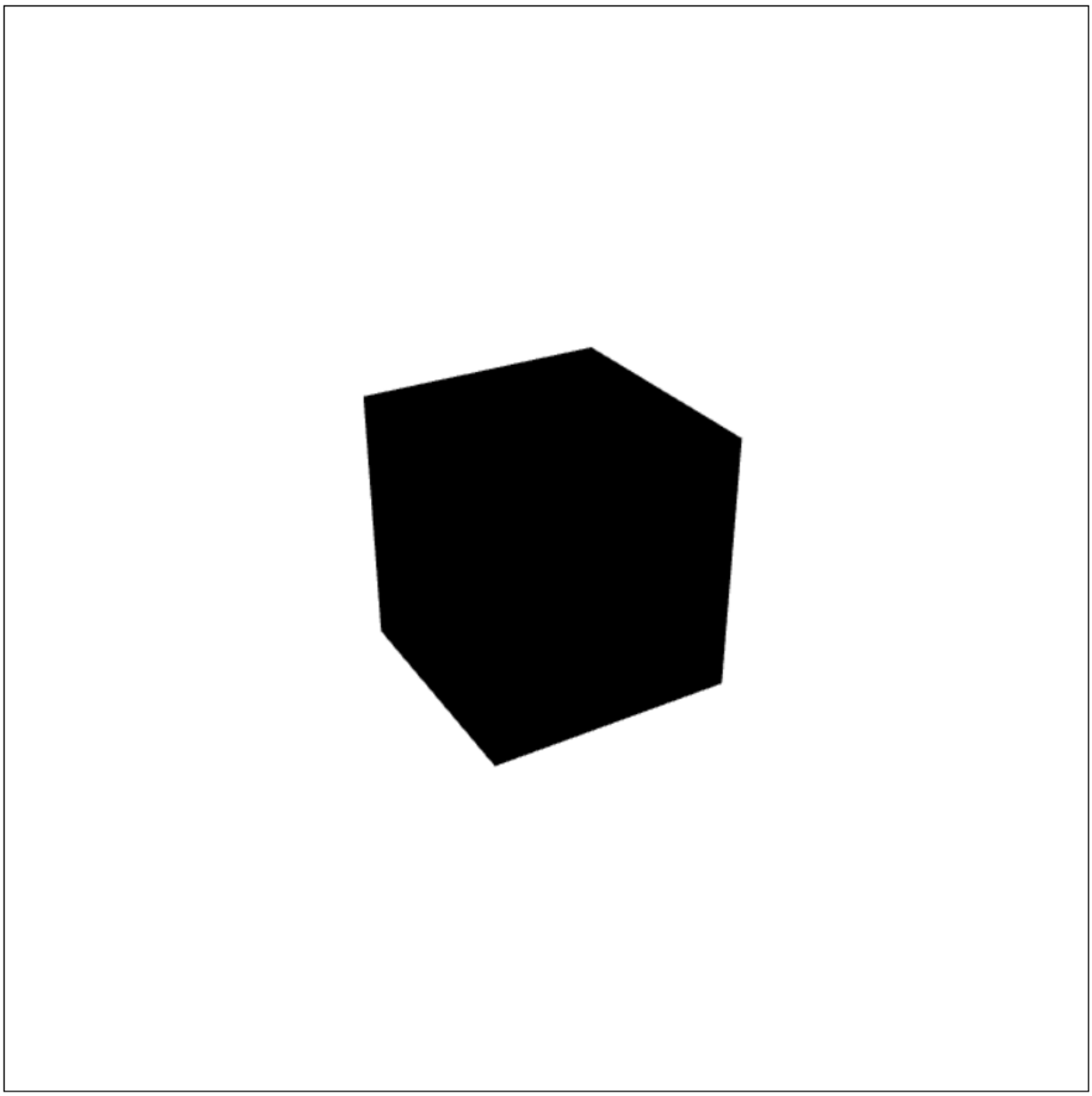

рис. 1.1. Створена кубічна коробка з кришкою

2-3. Застосування кольорів

У даному пункті застосовуємо кольори до коробки та кришки, та створюємо перспективну проекцію (рис. 2-3.1).

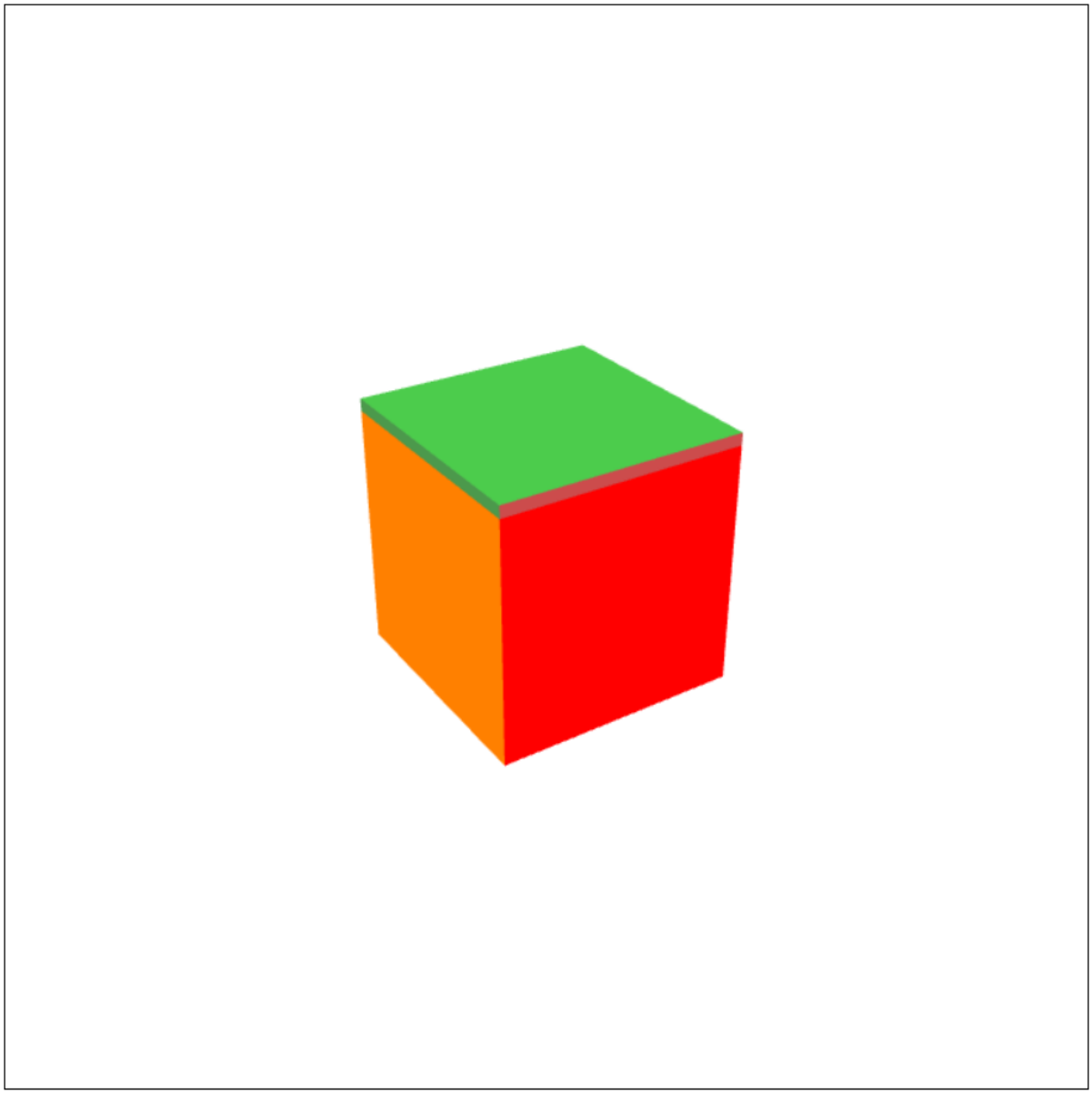

рис. 2-3.1. Коробка та кришка із застосованими кольорами

4. Анімація обертання

У даному пункті створимо анімацію обертання коробки із кришкою, що відбувається при натисканні клавіш “ArrowLeft” та “ArrowRight” (рис. 4.1).

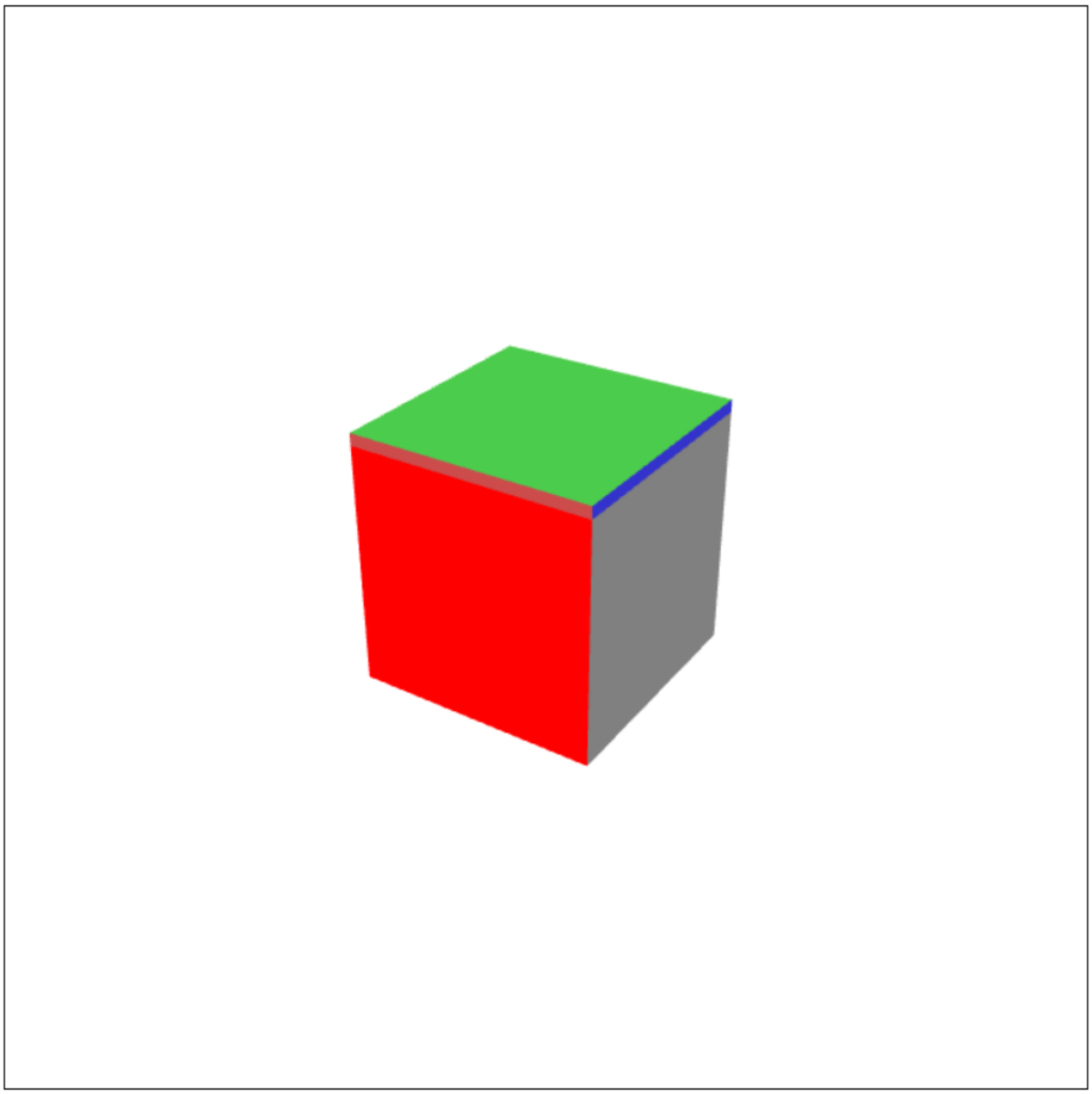

рис. 4.1. Куб повернутий навколо своєї осі

5. Анімація відкривання кришки

У даному пункті ми створюємо анімацію відкривання кришки, що відбувається при натисканні клавіш “ArrowUp” та “ArrowDown” (рис. 5.1).

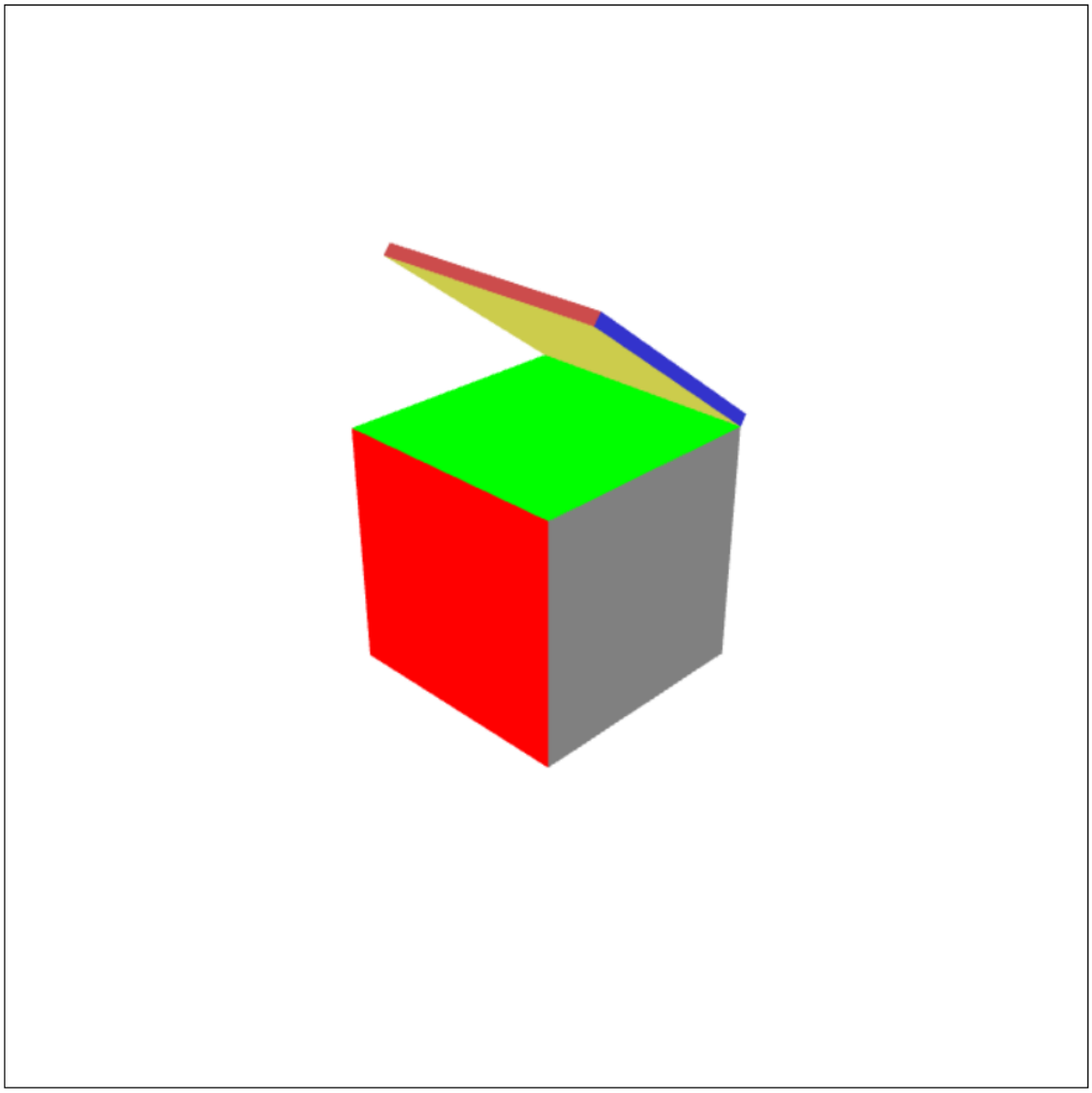

рис. 5.1. Коробка із кришкою в момент відкриття

Висновок: У ході виконання даної роботи ми створили просту 3D-сцену в WebGL, що складається з ящика з кришкою. Ящик і кришка мають однакові координати, використовуючи окремі шейдерні програми і кольори. Застосувавши перспективну проекцію, ми реалізували механізми обертання ящика та відкриття/закриття кришки за допомогою клавіш, що дозволило отримати інтерактивний графічний інтерфейс. Це підвищило наше розуміння графічного програмування та роботи з WebGL.

ПРОГРАМНИЙ КОД

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 boxVertexShaderSource = `
   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 boxFragmentShaderSource = `
    precision mediump float;
    varying vec4 v_color;

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

const lidVertexShaderSource = `
    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 lidFragmentShaderSource = `
    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, vertexShaderSource, fragmentShaderSource) {
    const vertexShader = createShader(gl, gl.VERTEX_SHADER,
vertexShaderSource);
    const fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
fragmentShaderSource);
    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 boxProgram = createProgram(gl, boxVertexShaderSource,
boxFragmentShaderSource);
const lidProgram = createProgram(gl, lidVertexShaderSource,
lidFragmentShaderSource);

const boxPositions = [
    -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 boxColors = [
    [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 lidPositions = [
    -0.2, 0.2,  0.2,
     0.2, 0.2,  0.2,
     0.2,  0.22,  0.2,
    -0.2, 0.2,  0.2,
     0.2,  0.22,  0.2,
    -0.2,  0.22,  0.2,

    -0.2, 0.2, -0.2,
     0.2, 0.2, -0.2,
     0.2,  0.22, -0.2,
    -0.2, 0.2, -0.2,
     0.2,  0.22, -0.2,
    -0.2,  0.22, -0.2,

    -0.2,  0.22,  0.2,
     0.2,  0.22,  0.2,
     0.2,  0.22, -0.2,
    -0.2,  0.22,  0.2,
     0.2,  0.22, -0.2,
    -0.2,  0.22, -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,  0.22,  0.2,
    -0.2,  0.22, -0.2,
    -0.2, 0.2,  0.2,
    -0.2,  0.22, -0.2,
    -0.2, 0.2, -0.2,

     0.2, 0.2,  0.2,
     0.2,  0.22,  0.2,
     0.2,  0.22, -0.2,
     0.2, 0.2,  0.2,
     0.2,  0.22, -0.2,
     0.2, 0.2, -0.2
];

const lidColors = [
    [0.8, 0.3, 0.3, 1.0],
    [0.3, 0.3, 0.8, 1.0],
    [0.3, 0.8, 0.3, 1.0],
    [0.8, 0.8, 0.3, 1.0],
    [0.3, 0.6, 0.3, 1.0],
    [0.2, 0.2, 0.8, 1.0]
].flatMap(color => Array(6).fill(color).flat());

const boxPositionBuffer = gl.createBuffer();
const boxColorBuffer = gl.createBuffer();
const lidPositionBuffer = gl.createBuffer();
const lidColorBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, boxPositionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(boxPositions),
gl.STATIC_DRAW);

gl.bindBuffer(gl.ARRAY_BUFFER, boxColorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(boxColors),
gl.STATIC_DRAW);

gl.bindBuffer(gl.ARRAY_BUFFER, lidPositionBuffer);
```

```
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(lidPositions),
gl.STATIC_DRAW);

gl.bindBuffer(gl.ARRAY_BUFFER, lidColorBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(lidColors),
gl.STATIC_DRAW);

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

let rotationY = 0;
let lidAngle = 0;
let lidOpen = false;

window.addEventListener('keydown', (event) => {
    if (event.key === 'ArrowLeft') {
        rotationY -= 0.05;
    } else if (event.key === 'ArrowRight') {
        rotationY += 0.05;
    } else if (event.key === 'ArrowUp') {
        lidOpen = true;
    } else if (event.key === 'ArrowDown') {
        lidOpen = false;
    }
});

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);

    const modelMatrix = mat4.create();
    mat4.translate(modelMatrix, modelMatrix, [0, 0, -2]);
    mat4.rotateX(modelMatrix, modelMatrix, Math.PI / 6);
    mat4.rotateY(modelMatrix, modelMatrix, Math.PI / 6);
```

```
    mat4.rotateY(modelMatrix, modelMatrix, rotationY);

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

    gl.useProgram(boxProgram);
    setAttribute(boxPositionBuffer, 'a_position', 3, boxProgram);
    setAttribute(boxColorBuffer, 'a_color', 4, boxProgram);
    gl.uniformMatrix4fv(gl.getUniformLocation(boxProgram, 'u_matrix'), false,
mvpMatrix);
    gl.drawArrays(gl.TRIANGLES, 0, boxPositions.length / 3);

    if (lidOpen && lidAngle < Math.PI / 2) {
        lidAngle += 0.02;
    } else if (!lidOpen && lidAngle > 0) {
        lidAngle -= 0.02;
    }

    const lidModelMatrix = mat4.create();
    mat4.translate(lidModelMatrix, lidModelMatrix, [0, 0, -2]);
    mat4.rotateX(lidModelMatrix, lidModelMatrix, Math.PI / 6);
    mat4.rotateY(lidModelMatrix, lidModelMatrix, Math.PI / 6);
    mat4.rotateY(lidModelMatrix, lidModelMatrix, rotationY);
    mat4.translate(lidModelMatrix, lidModelMatrix, [0, 0.2, -0.2]);
    mat4.rotateX(lidModelMatrix, lidModelMatrix, -lidAngle);
    mat4.translate(lidModelMatrix, lidModelMatrix, [0, -0.2, 0.2]);

    const lidMvpMatrix = mat4.create();
    mat4.multiply(lidMvpMatrix, projectionMatrix, lidModelMatrix);

    gl.useProgram(lidProgram);
    setAttribute(lidPositionBuffer, 'a_position', 3, lidProgram);
    setAttribute(lidColorBuffer, 'a_color', 4, lidProgram);
    gl.uniformMatrix4fv(gl.getUniformLocation(lidProgram, 'u_matrix'), false,
lidMvpMatrix);
    gl.drawArrays(gl.TRIANGLES, 0, lidPositions.length / 3);

    requestAnimationFrame(drawScene);
```

```
}

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

requestAnimationFrame(drawScene);
```