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

ЗВІТ

ПРО ЛАБОРАТОРНУ РОБОТУ №4

ТЕМА: «ПЕРСПЕКТИВА ТА ТРАНСФОРМАЦІЯ КАМЕРИ»

Виконала:
Студентка групи ІК-12
Макарчук Ольга

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

Київ 2024

Лабораторна робота №4

Тема: перспектива та трансформація камери

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

Хід виконання роботи

Файл index.html:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Laba #3</title>

    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="canvas-container">
        <canvas width="600" height="600" id="canvas"></canvas>
    </div>

    <script src="libs/glMatrix-0.9.5.min.js"></script>
    <script src="js/main.js" type="module"></script>
</body>
</html>
```

Файл style.css:

```css
* {
    margin: 0;
    padding: 0;
}

.canvas-container {
    width: 1000px;
    height: 600px;
    margin: 50px auto;
    display: flex;
    justify-content: center;
    align-items: center;
}
```

Файл main.js:

```
import { initializeWebGlContext, getShadersProgramText, getTriangleData, getCubeData }
from './helpers.js';
import { initializeShaders } from './shaders-helpers.js';
import { initializeTriangleBuffers, initializeCubeBuffers } from './buffers.js';

function main() {
    const canvas = document.getElementById('canvas');
    const glContext = initializeWebGlContext(canvas);

    const { vertexShader, fragmentShader } = getShadersProgramText();
    const { triangleVertex, triangleFace } = getTriangleData();
    const { cubeVertices, cubeFaces } = getCubeData();

    const shaderProgram = initializeShaders(glContext, vertexShader, fragmentShader);

    const programInfo = {
        attribLocations: {
            aPosition: glContext.getAttribLocation(shaderProgram, 'a_Position'),
            aColor: glContext.getAttribLocation(shaderProgram, 'a_Color')
        },
        uniformLocations: {
            uPmatrix: glContext.getUniformLocation(shaderProgram, 'u_Pmatrix'),
            uMmatrix: glContext.getUniformLocation(shaderProgram, 'u_Mmatrix'),
            uVmatrix: glContext.getUniformLocation(shaderProgram, 'u_Vmatrix')
        }
    };

    glContext.enableVertexAttribArray(programInfo.attribLocations.aPosition);
    glContext.enableVertexAttribArray(programInfo.attribLocations.aColor);

    const { triangleVertexBuffer, triangleFaceBuffer } =
initializeTriangleBuffers(glContext, triangleVertex, triangleFace);
    const { cubeVertexBuffer, cubeFaceBuffer } = initializeCubeBuffers(glContext,
cubeVertices, cubeFaces);

    const PROJMATRIX = mat4.perspective(35, canvas.width / canvas.height, 1, 100);
    let VIEWMATRIX = mat4.create();
    let MODELMATRIX = mat4.create();
    let VIEWMATRIX_eye = mat4.create();
    glContext.enable(glContext.DEPTH_TEST);
    const animate = function (time) {

        glContext.clear(glContext.COLOR_BUFFER_BIT | glContext.DEPTH_BUFFER_BIT);

        mat4.identity(MODELMATRIX);
        mat4.identity(VIEWMATRIX);
        mat4.identity(VIEWMATRIX_eye);

        mat4.translate(VIEWMATRIX_eye, [0.0, 0.0, 5.0]);
```

```
        let eye = vec3.create([0.0, 5.0, 5.0]);
        eye = mat4.multiplyVec3(VIEWMATRIX_eye, eye);
        const center = vec3.create([0.0, 0.0, 0.0]);
        const up = vec3.create([0.0, 1.0, 0.0]);
        VIEWMATRIX = mat4.lookAt(eye, center, up);

        glContext.clearColor(.5, .4, .5, 1.0);
        glContext.clear(glContext.COLOR_BUFFER_BIT);
        glContext.viewport(0, 0, canvas.width, canvas.height);

        mat4.scale(MODELMATRIX, [1.0, 1.0, 1.0]);
        mat4.rotateY(MODELMATRIX, rotationAngle);

        glContext.bindBuffer(glContext.ARRAY_BUFFER, triangleVertexBuffer);
        glContext.uniformMatrix4fv(programInfo.uniformLocations.uPmatrix, false,
PROJMATRIX);
        glContext.uniformMatrix4fv(programInfo.uniformLocations.uMmatrix, false,
MODELMATRIX);
        glContext.uniformMatrix4fv(programInfo.uniformLocations.uVmatrix, false,
VIEWMATRIX);
        glContext.vertexAttribPointer(programInfo.attribLocations.aPosition, 3,
glContext.FLOAT, false, 24, 0);
        glContext.vertexAttribPointer(programInfo.attribLocations.aColor, 3,
glContext.FLOAT, false, 24, 12);

        glContext.bindBuffer(glContext.ELEMENT_ARRAY_BUFFER, triangleFaceBuffer);
        glContext.drawElements(glContext.TRIANGLES, 36, glContext.UNSIGNED_SHORT, 0);

        //кришка
        mat4.identity(MODELMATRIX);
        mat4.translate(MODELMATRIX, [0, 1, 0]);
        mat4.rotateY(MODELMATRIX, rotationAngle,0);

        if (isLidOpen) {
            mat4.translate(MODELMATRIX, [0, 0.55, 0]); // Відкрита позиція кришки
            mat4.rotateX(MODELMATRIX, Math.PI / 7); // Додатковий нахил для відкритої
кришки
        }
        glContext.bindBuffer(glContext.ARRAY_BUFFER, cubeVertexBuffer);
        glContext.uniformMatrix4fv(programInfo.uniformLocations.uPmatrix, false,
PROJMATRIX);
        glContext.uniformMatrix4fv(programInfo.uniformLocations.uMmatrix, false,
MODELMATRIX);
        glContext.uniformMatrix4fv(programInfo.uniformLocations.uVmatrix, false,
VIEWMATRIX);
        glContext.vertexAttribPointer(programInfo.attribLocations.aPosition, 3,
glContext.FLOAT, false, 24, 0);
        glContext.vertexAttribPointer(programInfo.attribLocations.aColor, 3,
glContext.FLOAT, false, 24, 12);
```

```
        glContext.bindBuffer(glContext.ELEMENT_ARRAY_BUFFER, cubeFaceBuffer);
        glContext.drawElements(glContext.TRIANGLES, 36, glContext.UNSIGNED_SHORT, 0);
        glContext.flush();
        window.requestAnimationFrame(animate);
    }
    animate(0);
}

window.addEventListener('load', () => {
    main();
});

let isLidOpen = false;
let rotationAngle = 0;
function handleKeyPress(event) {
    if (event.key === "ArrowUp") {
        isLidOpen = true;
    } else if (event.key === "ArrowDown") {
        isLidOpen = false;
    }else if (event.key === "ArrowLeft") {
        rotationAngle += Math.PI / 36; // Збільшуємо кут обертання вліво
    } else if (event.key === "ArrowRight") {
        rotationAngle -= Math.PI / 36; // Зменшуємо кут обертання вправо
    }
}
document.addEventListener('keydown', handleKeyPress);
```

Файл shalders-helper.js:

```
export function initializeShaders(glContext, vertexShader, fragmentShader) {
    const getFragmentShaderResult = getShader(glContext, 'vs', vertexShader);
    const getVertexShaderResult = getShader(glContext, 'fs', fragmentShader);

    const shaderProgram = glContext.createProgram();
    glContext.attachShader(shaderProgram, getFragmentShaderResult);
    glContext.attachShader(shaderProgram, getVertexShaderResult);
    glContext.linkProgram(shaderProgram);

    glContext.useProgram(shaderProgram);

    return shaderProgram;
}

function getShader(glContext, id, str) {
    const shader = (id === 'vs') ? glContext.createShader(glContext.VERTEX_SHADER) :
        (id === 'fs') ? glContext.createShader(glContext.FRAGMENT_SHADER) :
            null;

    glContext.shaderSource(shader, str);
```

```
    glContext.compileShader(shader);

    if (!glContext.getShaderParameter(shader, glContext.COMPILE_STATUS)) {
        console.error(glContext.getShaderInfoLog(shader));
        return null;
    }

    return shader;
}
```

Файл helpers.js:

```
export function initializeWebGlContext(canvas) {
    let glContext = null;

    try {
        glContext = canvas.getContext('webgl' || 'experimental-webgl');
    } catch (error) {
        console.error(error);
        return;
    }
    if (!glContext) {
        console.error('Unable to initialize WebGL. Your browser may not support it.');
        return;
    }
    return glContext;
}

export function getShadersProgramText() {
    const fragmentShader = `
        precision mediump float;
        uniform vec4 u_FragColor;
        varying vec3 v_Color;

        void main() {
            gl_FragColor = vec4(v_Color, 1.0);
        }
    `;

    const vertexShader = `
        attribute vec3 a_Position;
        attribute vec3 a_Color;
        uniform mat4 u_Pmatrix;
        uniform mat4 u_Mmatrix;
        uniform mat4 u_Vmatrix;
        varying vec3 v_Color;

        void main() {
            v_Color = a_Color;
            gl_Position = u_Pmatrix * u_Vmatrix * u_Mmatrix * vec4(a_Position, 1.0);
        }
```

```
    `;
    return { fragmentShader, vertexShader };
}

export function getTriangleData() {
    const triangleVertex = [
        -1, -1, -1, 1, 1, 0,
        1, -1, -1, 1, 1, 0,
        1, 1, -1, 1, 1, 0,
        -1, 1, -1, 1, 1, 0,

        -1, -1, 1, 1, 0, 0,
        1, -1, 1, 1, 0, 0,
        1, 1, 1, 1, 0, 0,
        -1, 1, 1, 1, 0, 0,

        -1, -1, -1, 0, 1, 1,
        -1, 1, -1, 0, 1, 1,
        -1, 1, 1, 0, 1, 1,
        -1, -1, 1, 0, 1, 1,

        1, -1, -1, 1, 0, 1,
        1, 1, -1, 1, 0, 1,
        1, 1, 1, 1, 0, 1,
        1, -1, 1, 1, 0, 1,

        -1, -1, -1, 1, 0, 1,
        -1, -1, 1, 1, 0, 1,
        1, -1, 1, 1, 0, 1,
        1, -1, -1, 1, 0, 1,

        -1, 1, -1, 0, 1, 0,
        -1, 1, 1, 0, 1, 0,
        1, 1, 1, 0, 1, 0,
        1, 1, -1, 0, 1, 0
    ];

    const triangleFace = [
        0, 1, 2,
        0, 2, 3,

        4, 5, 6,
        4, 6, 7,

        8, 9, 10,
        8, 10, 11,

        12, 13, 14,
        12, 14, 15,

        16, 17, 18,
```

```
        16, 18, 19,

        20, 21, 22,
        20, 22, 23
    ];

    return { triangleVertex, triangleFace };
}

export function getCubeData() {
    const cubeVertices = [
        -1, -0.2, -1, 1, 1, 0,
        1, -0.2, -1, 1, 1, 0,
        1, 0.2, -1, 1, 1, 0,
        -1, 0.2, -1, 1, 1, 0,

        -1, -0.2, 1, 1, 0, 0,
        1, -0.2, 1, 1, 0, 0,
        1, 0.2, 1, 1, 0, 0,
        -1, 0.2, 1, 1, 0, 0,

        -1, -0.2, -1, 0, 1, 1,
        -1, 0.2, -1, 0, 1, 1,
        -1, 0.2, 1, 0, 1, 1,
        -1, -0.2, 1, 0, 1, 1,

        1, -0.2, -1, 1, 0, 1,
        1, 0.2, -1, 1, 0, 1,
        1, 0.2, 1, 1, 0, 1,
        1, -0.2, 1, 1, 0, 1,

        -1, -0.2, -1, 1, 0, 1,
        -1, -0.2, 1, 1, 0, 1,
        1, -0.2, 1, 1, 0, 1,
        1, -0.2, -1, 1, 0, 1,

        -1, 0.2, -1, 0, 1, 0,
        -1, 0.2, 1, 0, 1, 0,
        1, 0.2, 1, 0, 1, 0,
        1, 0.2, -1, 0, 1, 0
    ];

    const cubeFaces = [
        0, 1, 2,
        0, 2, 3,

        4, 5, 6,
        4, 6, 7,

        8, 9, 10,
        8, 10, 11,
```

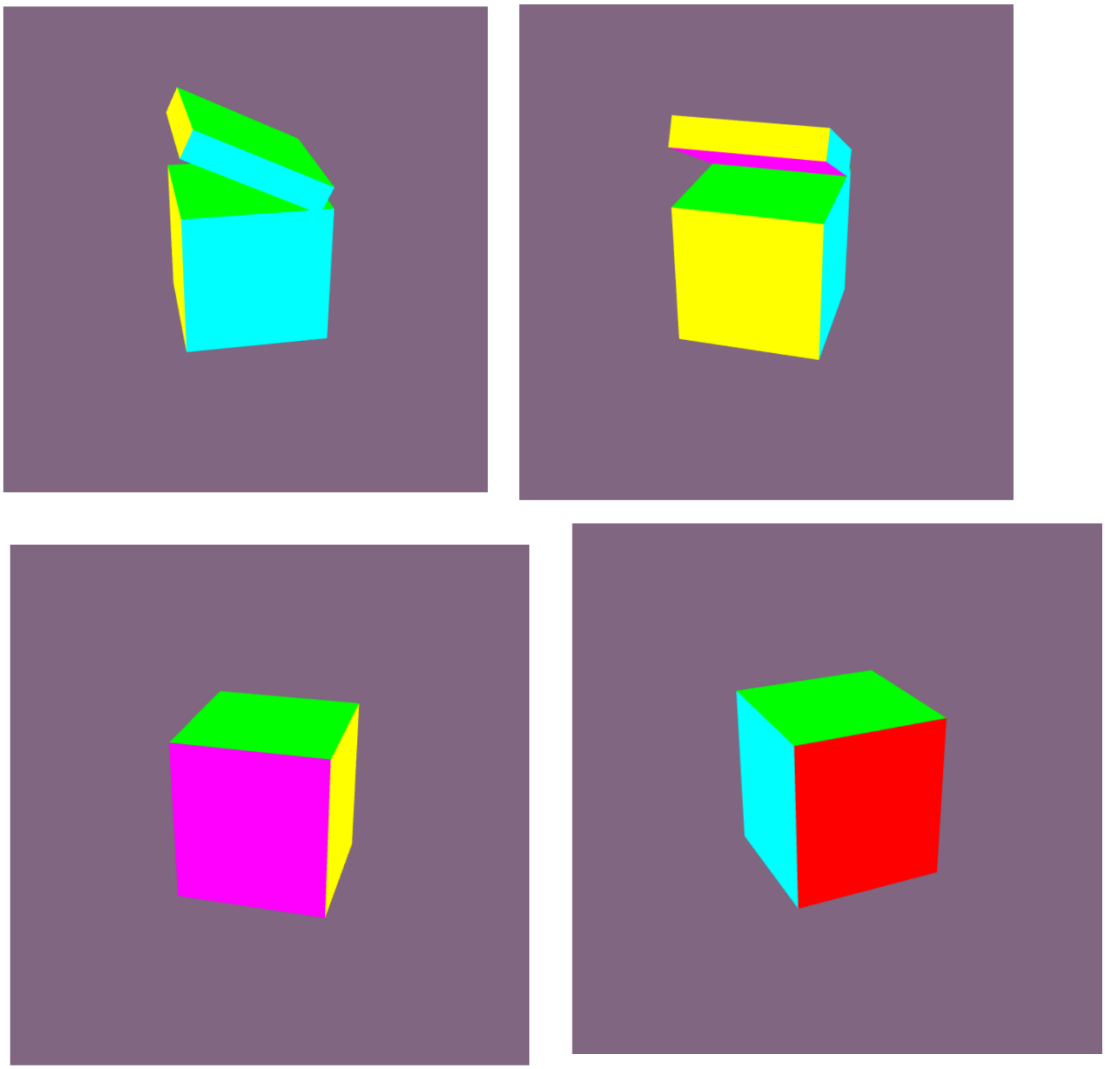
```
        12, 13, 14,
        12, 14, 15,

        16, 17, 18,
        16, 18, 19,

        20, 21, 22,
        20, 22, 23
    ];

    return { cubeVertices, cubeFaces };
}
```

Результат:



Висновок

У даній лабораторній роботі, я поглибила теоретичні знання та практичні навички щодо роботи з режимами перегляду у WebGL. Також було застосовано на практиці роботу з проекціями та трансформаціями на основі програмного інтерфейсу WebGL.