МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт

З лабораторної роботи № 7 з дисципліни
«Комп’ютерна графіка та обробка зображень»

Тема: «Проектування у тривимірному просторі»

Виконав(ла) *ІП-15 Мєшков Андрій* _____

(шифр, прізвище, ім'я, по батькові)

Перевірив *Щебланін Ю. М.* _____

(шифр, прізвище, ім'я, по батькові)

Київ 2024

ЗМІСТ

ВСТУП

У сучасному світі комп’ютерна графіка та обробка зображень стали невід’ємною частиною багатьох галузей, включаючи інженерію, архітектуру, дизайн та розробку відеоігор. Лабораторна робота № 7 присвячена проектуванню у тривимірному просторі з використанням бібліотеки Three.js. Метою роботи є набуття практичних навичок створення та налаштування тривимірних об'єктів, роботи зі світлом та камерами, а також застосування різних проекцій для візуалізації сцен.

В процесі виконання роботи було розглянуто декілька завдань, кожне з яких спрямоване на вивчення конкретних аспектів тривимірного проектування. Завдання включали створення об'єктів, налаштування матеріалів та текстур, додавання світлових ефектів, а також використання перспективних проекцій для надання сцени реалістичності.

ХІД РОБОТИ

Завдання 7.1

Додавання бліків.

Лістинг програмного коду:

Index.html

```html
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Lab 7</title>
        <style>
            * {
                box-sizing: border-box;
            }
            body {
                margin: 0;
                padding: 0;
            }
        </style>
    </head>
    <body>
        <script type="module" src="./task2.js"></script>
    </body>
</html>
```

Task1.js

```javascript
import * as THREE from 'three';
import { OrbitControls } from 'three/addons/controls/OrbitControls.js';
import { TeapotGeometry } from 'three/addons/geometries/TeapotGeometry.js';

// Color constants
const COLORS = {
    C_White: new THREE.Color(1.0, 1.0, 1.0),
    C_Black: new THREE.Color(0.0, 0.0, 0.0),
    C_Grey: new THREE.Color(0.5, 0.5, 0.5),
    C_DarcGrey: new THREE.Color(0.2, 0.2, 0.2),
    C_Red: new THREE.Color(1.0, 0.0, 0.0),
    C_Green: new THREE.Color(0.0, 1.0, 0.0),
    C_Blue: new THREE.Color(0.0, 0.0, 1.0),
    C_DarcBlue: new THREE.Color(0.0, 0.0, 0.5),
    C_Cyan: new THREE.Color(0.0, 1.0, 1.0),
    C_Magenta: new THREE.Color(1.0, 0.0, 1.0),
    C_Yellow: new THREE.Color(1.0, 1.0, 0.0),
    C_Orange: new THREE.Color(0.1, 0.5, 0.0),
    C_Lemon: new THREE.Color(0.8, 1.0, 0.0),
    C_Brown: new THREE.Color(0.5, 0.3, 0.0),
    C_Navy: new THREE.Color(0.0, 0.4, 0.8),
```

```
    C_Aqua: new THREE.Color(0.4, 0.7, 1.0),
    C_Cherry: new THREE.Color(1.0, 0.0, 0.5)
};

const scene = new THREE.Scene();

// Add shininess and specular
const geometry = new TeapotGeometry(2);
const material = new THREE.MeshPhongMaterial({
    color: COLORS.C_Cherry,
    shininess: 50,
    specular: COLORS.C_White
});
const teapot = new THREE.Mesh(geometry, material);
scene.add(teapot);

const axesHelper = new THREE.AxesHelper(5);
scene.add(axesHelper);

const light = new THREE.DirectionalLight(0xffffff, 1);
light.position.set(1, 1, 1).normalize();
scene.add(light);

const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

const aspectRatio = window.innerWidth / window.innerHeight;
const cameras = [];

const camera1 = new THREE.OrthographicCamera(-aspectRatio * 5, aspectRatio * 5, 5, -5, 0.1,
100);
camera1.position.set(0, 0, 5);
camera1.lookAt(0, 0, 0);
cameras.push(camera1);

const camera2 = new THREE.OrthographicCamera(-aspectRatio * 5, aspectRatio * 5, 5, -5, 0.1,
100);
camera2.position.set(0, 5, 0);
camera2.lookAt(0, 0, 0);
camera2.up.set(0, 0, -1);
cameras.push(camera2);

const camera3 = new THREE.OrthographicCamera(-aspectRatio * 5, aspectRatio * 5, 5, -5, 0.1,
100);
camera3.position.set(5, 0, 0);
camera3.lookAt(0, 0, 0);
cameras.push(camera3);

const camera4 = new THREE.OrthographicCamera(-aspectRatio * 5, aspectRatio * 5, 5, -5, 0.1,
100);
camera4.position.set(5, 5, 5);
camera4.lookAt(0, 0, 0);
```

```
cameras.push(camera4);

function renderView(camera, x, y, width, height) {
    const windowWidth = window.innerWidth;
    const windowHeight = window.innerHeight;

    renderer.setViewport(x, y, width, height);
    renderer.setScissor(x, y, width, height);
    renderer.setScissorTest(true);

    camera.aspect = width / height;
    camera.updateProjectionMatrix();

    renderer.render(scene, camera);
}

function animate() {
    requestAnimationFrame(animate);

    renderer.clear();

    const width = window.innerWidth / 2;
    const height = window.innerHeight / 2;

    renderView(cameras[0], 0, height, width, height);
    renderView(cameras[1], 0, 0, width, height);
    renderView(cameras[2], width, height, width, height);
    renderView(cameras[3], width, 0, width, height);
}

animate();
```

Результати виконання програми:

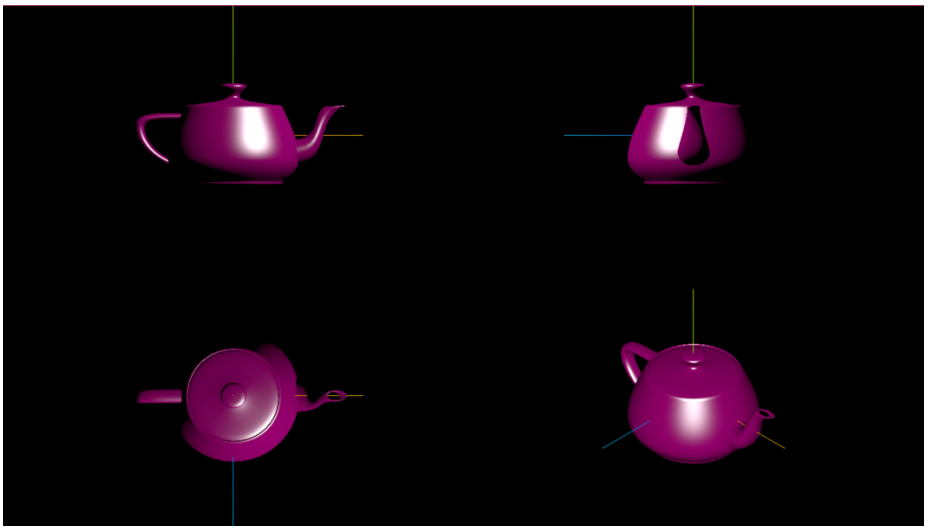

Рисунок 7.1 — Результат виконання програми

Завдання 7.2

Перспективна триточкова проекція.

Лістинг програмного коду:

Task2.js

```
import * as THREE from 'three';
import { OrbitControls } from 'three/addons/controls/OrbitControls.js';

// Create the scene
const scene = new THREE.Scene();

// Create a cube with different colored faces
const geometry = new THREE.BoxGeometry(2,2,2);
const materials = [
    new THREE.MeshBasicMaterial({ color: 0x00ff00 }), // green
    new THREE.MeshBasicMaterial({ color: 0xffff00 }), // yellow
    new THREE.MeshBasicMaterial({ color: 0xff00ff }), // magenta
    new THREE.MeshBasicMaterial({ color: 0xff00ff }), // magenta
    new THREE.MeshBasicMaterial({ color: 0xff0000 }), // red
    new THREE.MeshBasicMaterial({ color: 0x0000ff }), // blue
];

const cube = new THREE.Mesh(geometry, materials);
scene.add(cube);

// Add coordinate axes to the scene
const axesHelper = new THREE.AxesHelper(5); // 5 units long
scene.add(axesHelper);

// Custom perspective projection matrix
const matrix = new Float32Array([
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, -1/5, // Adjust the perspective projection matrix
    0, 0, 0, 1
]);

// Create the renderer
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

const aspectRatio = window.innerWidth / window.innerHeight;
const cameras = [];

// Function to create a camera with perspective projection
function createCamera(fov, aspect, near, far, x, y, z) {
    const camera = new THREE.PerspectiveCamera(fov, aspect, near, far);
    camera.position.set(x, y, z);
    camera.lookAt(0, 0, 0);
    camera.projectionMatrix.elements = matrix;
```

```
    return camera;
}

// Camera 1: Front view (XOY)
const camera1 = createCamera(75, aspectRatio, 0.1, 100, 0, 0, 5);
cameras.push(camera1);

// Camera 2: Top view (XOZ)
const camera2 = createCamera(75, aspectRatio, 0.1, 100, 0, 5, 0);
camera2.up.set(0, 0, -1);
cameras.push(camera2);

// Camera 3: Side view (YOZ)
const camera3 = createCamera(75, aspectRatio, 0.1, 100, 5, 0, 0);
cameras.push(camera3);

// Camera 4: Isometric view
const camera4 = createCamera(75, aspectRatio, 0.1, 100, 5, 5, 5);
cameras.push(camera4);

function renderView(camera, x, y, width, height) {
    const windowWidth = window.innerWidth;
    const windowHeight = window.innerHeight;

    renderer.setViewport(x, y, width, height);
    renderer.setScissor(x, y, width, height);
    renderer.setScissorTest(true);

    camera.aspect = width / height;
    camera.updateProjectionMatrix();

    renderer.render(scene, camera);
}

function animate() {
    requestAnimationFrame(animate);

    // Clear the entire canvas
    renderer.clear();

    // Calculate viewports and render each view
    const width = window.innerWidth / 2;
    const height = window.innerHeight / 2;

    // Top-left: Front view (XOY)
    renderView(cameras[0], 0, height, width, height);

    // Bottom-left: Top view (XOZ)
    renderView(cameras[1], 0, 0, width, height);

    // Top-right: Side view (YOZ)
    renderView(cameras[2], width, height, width, height);
```

```
    // Bottom-right: Isometric view
    renderView(cameras[3], width, 0, width, height);
}

animate();
```

Результати виконання програми:

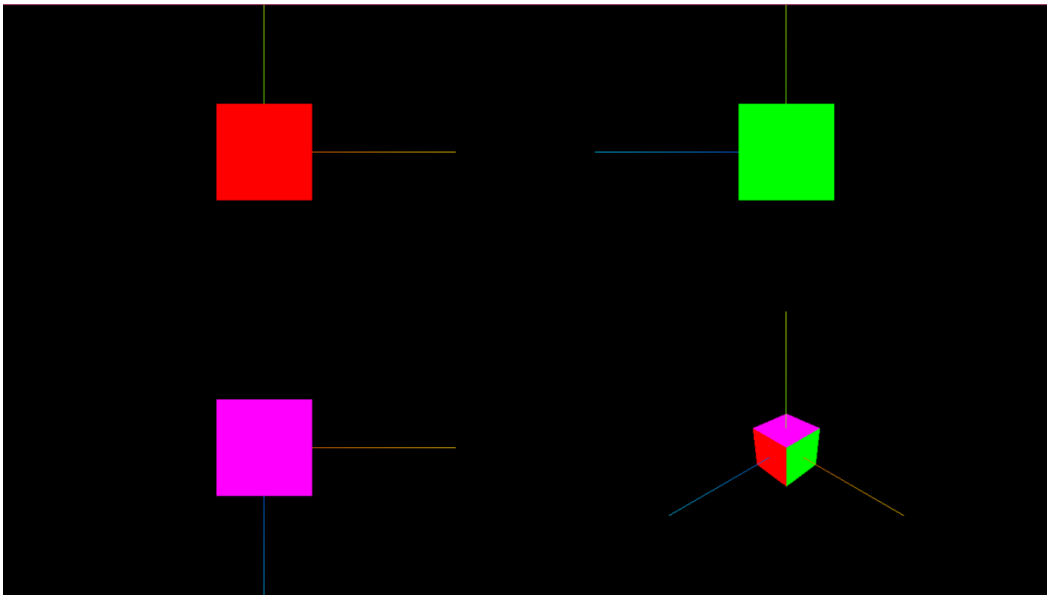

Рисунок 5.2 — Результат виконання програми

Завдання 7.3

Матриця перспективної триточкової проекції з врахуванням параметрів об’єму видимості.

Лістинг програмного коду:

Task3.js

```
import * as THREE from 'three';

const scene = new THREE.Scene();

const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1,
1000);
camera.position.set(-1, -2, 10);

const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

const geometry = new THREE.BoxGeometry(2, 2, 2);

const materials = [
```

```
    new THREE.MeshBasicMaterial({ color: 0x00ff00 }), // green
    new THREE.MeshBasicMaterial({ color: 0xffff00 }), // yellow
    new THREE.MeshBasicMaterial({ color: 0xff00ff }), // magenta
    new THREE.MeshBasicMaterial({ color: 0xff00ff }), // magenta
    new THREE.MeshBasicMaterial({ color: 0xff0000 }), // red
    new THREE.MeshBasicMaterial({ color: 0x0000ff }), // blue
];

const cube = new THREE.Mesh(geometry, materials);
scene.add(cube);

cube.rotation.x = 0.6;
cube.rotation.y = 0.6;

const pointLight = new THREE.PointLight(0xffffff, 1, 100);
pointLight.position.set(5, 5, 10);
scene.add(pointLight);

function animate() {
    requestAnimationFrame(animate);

    renderer.render(scene, camera);
}

animate();

window.addEventListener('resize', () => {
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize(window.innerWidth, window.innerHeight);
});
```

Результати виконання програми:

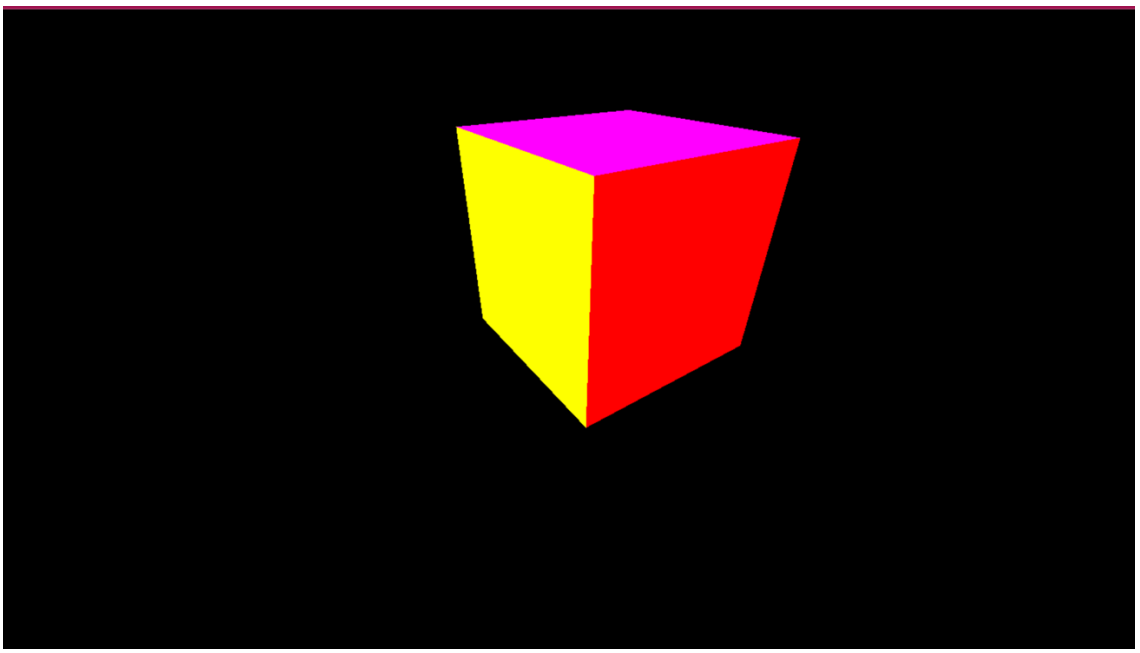

Рисунок 5.3 — Результат виконання програми

ВИСНОВКИ

Виконання лабораторної роботи № 7 дозволило поглибити знання та навички у сфері тривимірного проектування з використанням бібліотеки Three.js. В ході роботи були створені та налаштовані різні тривимірні об'єкти, застосовані різні види матеріалів та текстур, додані світлові ефекти, а також використані різні проекції для візуалізації сцен.

Основні висновки, які можна зробити на основі виконаної роботи:

1. Three.js є потужним інструментом для створення тривимірної графіки в веб-браузерах, що дозволяє легко реалізувати складні графічні сцени.

2. Використання різних видів камер та проекцій дає можливість більш гнучко керувати відображенням тривимірних об'єктів, що є важливим для досягнення реалістичності та точності у візуалізації.

3. Додавання світлових ефектів значно покращує вигляд сцени, надаючи їй об'ємність та динамічність.

4. Практичне застосування знань з комп’ютерної графіки дозволяє глибше зрозуміти теоретичні концепції та підходи, що використовуються у даній галузі.

Загалом, лабораторна робота сприяла розвитку практичних навичок, необхідних для подальшого вивчення та професійного застосування технологій тривимірного проектування та комп’ютерної графіки.

.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Three.js [Електронний ресурс] — Режим доступу: https://threejs.org/docs/