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 № 2 з дисципліни
«Комп’ютерна графіка та обробка зображень»

Тема: «Створення мінімальної 3D-програми з використанням графічної бібліотеки OpenGL»

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 3
ХІД РОБОТИ 4
Завдання 1.1 4
Завдання 1.2 6
Завдання 1.3 9
Завдання 1.4 11
ВИСНОВКИ 15
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 16

ВСТУП

В сучасному світі візуалізація об'єктів та створення тривимірних сцен є ключовим елементом в багатьох галузях, починаючи від ігрової індустрії до віртуальної реальності та наукових досліджень. У цій лабораторній роботі ми зосередимося на створенні мінімальної 3D-програми з використанням графічної бібліотеки OpenGL, яка є однією з найпоширеніших технологій для реалізації графіки у реальному часі.

Метою цієї лабораторної роботи є ознайомлення з базовими концепціями тривимірної графіки та реалізація простої 3D-програми з використанням бібліотеки OpenGL. Ми будемо розглядати основні елементи створення тривимірних об'єктів, їх текстурування та відображення на екрані.

У даній лабораторній роботі ми використовували такі технології:

1. Three.js: Це відкрита бібліотека для роботи з тривимірною графікою у веб-проектах, яка спрощує роботу з WebGL.

2. JavaScript: Мова програмування, що використовується для реалізації логіки програми та взаємодії з графічними елементами.

3. HTML/CSS: Використовується для створення користувацького інтерфейсу та відображення 3D-сцени у веб-браузері.

ХІД РОБОТИ

Завдання 1.1

Доповнити процедуру виведенням верхньої та нижньої граней куба.

Лістинг програмного коду:

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Lab 2</title>
    <style>
      * {
        box-sizing: border-box;
      }
      body {
        margin: 0;
        padding: 0;
      }
    </style>
  </head>
  <body>
    <script type="module" src="./task1.js"></script>
  </body>
</html>
```

task1.js

```
import * as THREE from 'three'; // Import the main Three.js library

import { OrbitControls } from 'three/addons/controls/OrbitControls.js'; // Import
OrbitControls for camera manipulation
import { GLTFLoader } from 'three/addons/loaders/GLTFLoader.js'; // Import GLTFLoader for
loading 3D models

// Create a new Three.js scene
const scene = new THREE.Scene();

// Create a perspective camera with a field of view of 50, aspect ratio based on window
size, and near/far clipping planes
const camera = new THREE.PerspectiveCamera(50, window.innerWidth / window.innerHeight, 0.1,
1000);
camera.position.z = 3; // Position the camera 3 units away from the origin along the z-axis

// Create a WebGL renderer and set its size to fill the window
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement); // Append the renderer's DOM element to the
document body
```

```
// Create OrbitControls to allow mouse interaction with the camera
const controls = new OrbitControls(camera, renderer.domElement);

// Create a GLTFLoader to load 3D models (not used in this specific code snippet)
const loader = new GLTFLoader();

const geometry = new THREE.BoxGeometry(); // Create a box geometry (cube)

// Create an array of materials to apply to the faces of the cube
const materials = [
    new THREE.MeshBasicMaterial({ color: 0x00ff00 }), // Green color material
    new THREE.MeshBasicMaterial({ color: 0xffff00 }), // Yellow color material
    new THREE.MeshBasicMaterial({ color: 0xff00ff }), // Magenta color material
    new THREE.MeshBasicMaterial({ color: 0xff00ff }), // Magenta color material
    new THREE.MeshBasicMaterial({ color: 0xff0000 }), // Red color material
    new THREE.MeshBasicMaterial({ color: 0x0000ff }), // Blue color material
];

const wireframeMaterial = new THREE.MeshBasicMaterial({
    color: 0x000000, // Black color for the wireframe
    wireframe: true, // Enable wireframe rendering
    opacity: 0.9, // Set the opacity of the wireframe
    transparent: true, // Enable transparency
});

// Create a mesh with box geometry and the array of materials
const cube = new THREE.Mesh(geometry, materials);
cube.add(new THREE.Mesh(geometry, wireframeMaterial)); // Add wireframe to the cube for
better visualization
scene.add(cube); // Add the cube to the scene

// Define the animation loop
function animate() {
    requestAnimationFrame(animate); // Request the next frame
    renderer.render(scene, camera); // Render the scene from the perspective of the camera
}

// Start the animation loop
animate();
```

Результат виконання завдань:

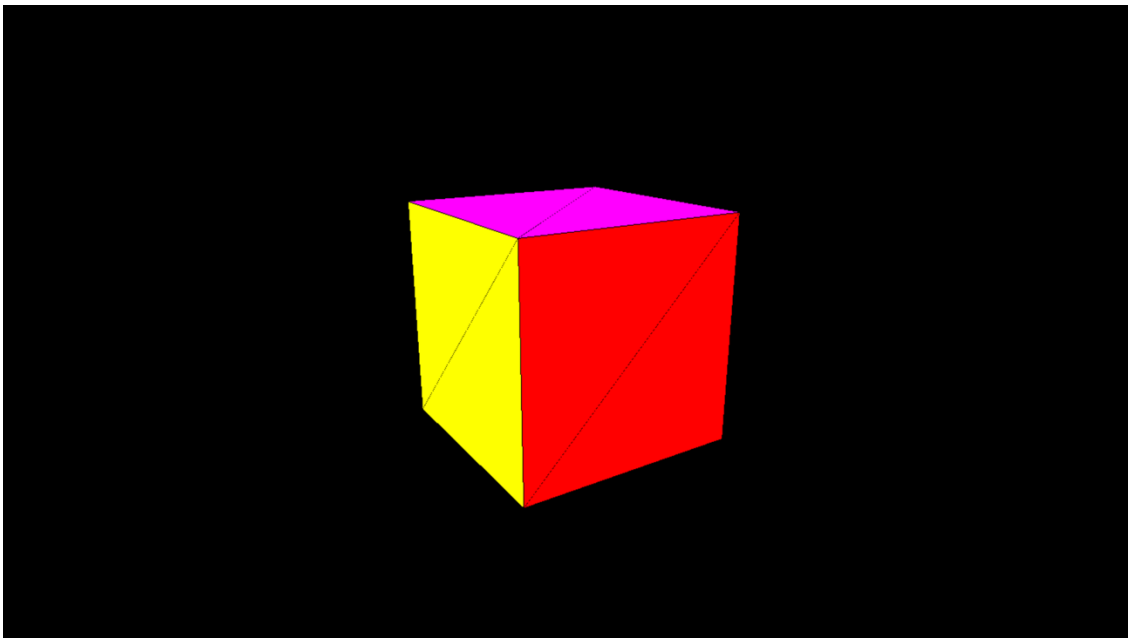

Рисунок 1.1 — Результат виконання програми

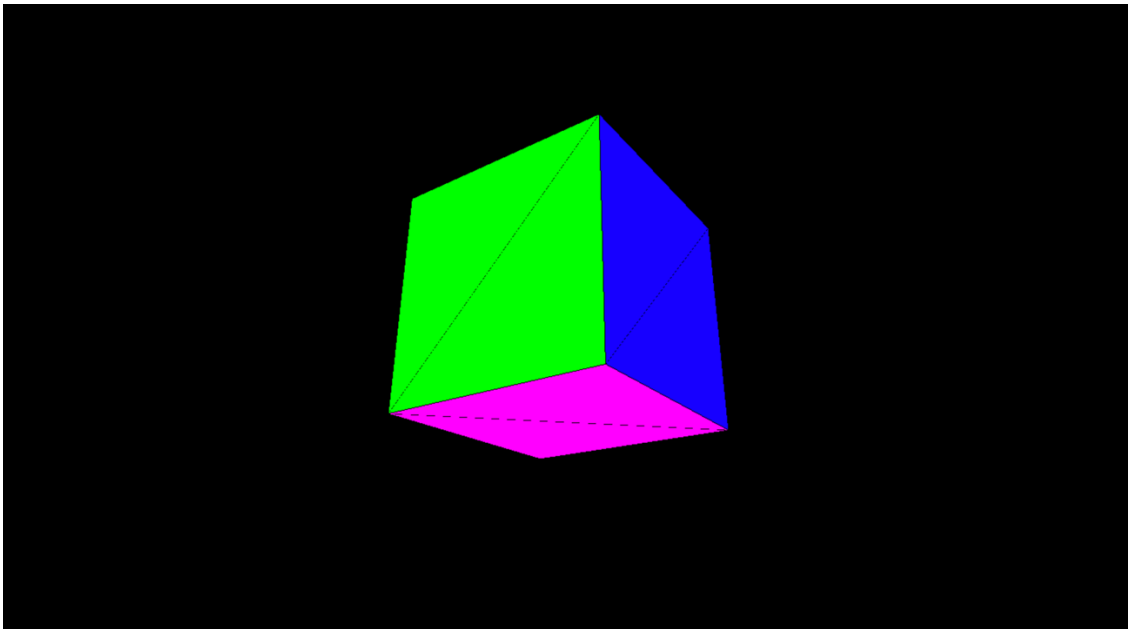

Рисунок 1.2 — Результат виконання програми

Завдання 1.2

Доповнити процедуру виведенням нижньої грані призми (правильного n-кутника).

Лістинг програмного коду:

task2.js

```
import * as THREE from 'three'; // Import the main Three.js library
import { OrbitControls } from 'three/addons/controls/OrbitControls.js'; // Import
OrbitControls for camera manipulation
import { GLTFLoader } from 'three/addons/loaders/GLTFLoader.js'; // Import GLTFLoader for
loading 3D models

// Create a new Three.js scene
const scene = new THREE.Scene();
scene.background = new THREE.Color(0x000000); // Set the background color of the scene to
black

// Create a perspective camera with a field of view of 50, aspect ratio based on window
size, and near/far clipping planes
const camera = new THREE.PerspectiveCamera(50, window.innerWidth / window.innerHeight, 0.1,
1000);
camera.position.z = 3; // Position the camera 3 units away from the origin along the z-axis

// Create a WebGL renderer and set its size to fill the window
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement); // Append the renderer's DOM element to the
document body

// Create OrbitControls to allow mouse interaction with the camera
const controls = new OrbitControls(camera, renderer.domElement);

// Create a GLTFLoader to load 3D models (not used in this specific code snippet)
const loader = new GLTFLoader();

// Parameters for the hexagon
const radius = 0.5; // Radius of the hexagon base
const height = 1; // Height of the hexagon

// Create geometry for a hexagonal cylinder (hexagon prism)
const geometry = new THREE.CylinderGeometry(radius, radius, height, 6);

// Create an array of materials for the hexagon faces
const materials = [
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
    new THREE.MeshBasicMaterial({ color: 0xff00ff }), // Magenta color material for a
specific face
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
];

// Create a wireframe material for the hexagon
```

```
const wireframeMaterial = new THREE.MeshBasicMaterial({
  color: 0x000000, // Black color for the wireframe
  wireframe: true, // Enable wireframe rendering
  opacity: 0.9, // Set the opacity of the wireframe
  transparent: true, // Enable transparency
});

// Create the hexagon mesh with the geometry and materials
const hexagon = new THREE.Mesh(geometry, materials);

// Add a wireframe to the hexagon for better visual representation
hexagon.add(new THREE.Mesh(geometry, wireframeMaterial));

// Add the hexagon to the scene
scene.add(hexagon);

// Define the animation loop
function animate() {
  requestAnimationFrame(animate); // Request the next frame
  renderer.render(scene, camera); // Render the scene from the perspective of the camera
}

// Start the animation loop
animate();
```

Результат виконання завдань:

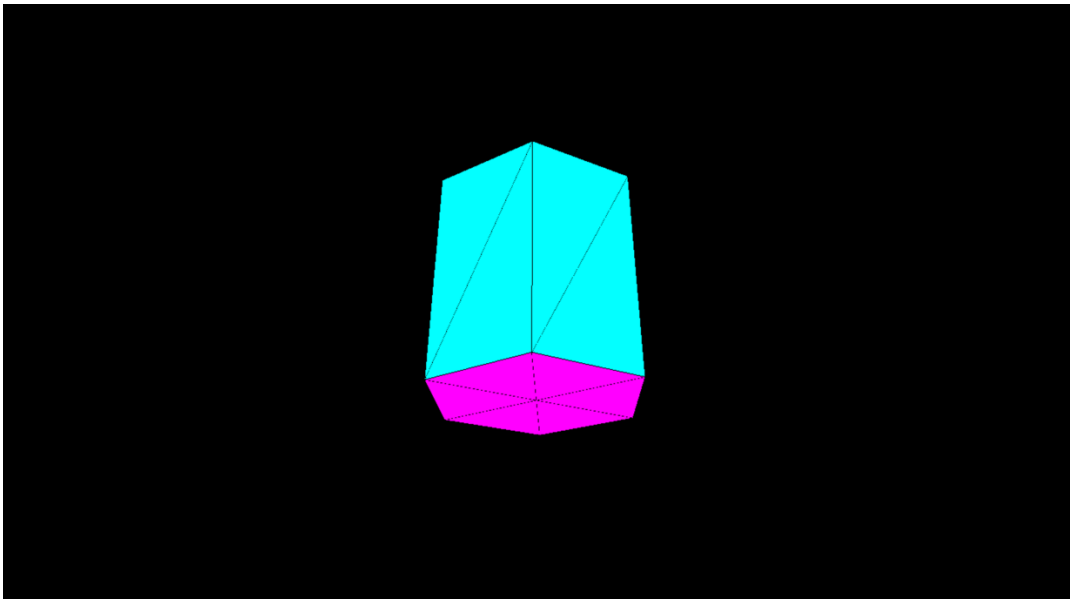

Рисунок 1.3 — Результат виконання програми

Завдання 1.3

Доповнити процедуру виведенням основи піраміди (правильного n-кутника).

Лістинг програмного коду:

task3.js

```
import * as THREE from 'three'; // Import the main Three.js library

import { OrbitControls } from 'three/addons/controls/OrbitControls.js'; // Import
OrbitControls for camera manipulation
import { GLTFLoader } from 'three/addons/loaders/GLTFLoader.js'; // Import GLTFLoader for
loading 3D models

// Create a new Three.js scene
const scene = new THREE.Scene();

// Create a perspective camera with a field of view of 50, aspect ratio based on window
size, and near/far clipping planes
const camera = new THREE.PerspectiveCamera(50, window.innerWidth / window.innerHeight, 0.1,
1000);
camera.position.z = 3; // Position the camera 3 units away from the origin along the z-axis

// Create a WebGL renderer and set its size to fill the window
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement); // Append the renderer's DOM element to the
document body

// Create OrbitControls to allow mouse interaction with the camera
const controls = new OrbitControls(camera, renderer.domElement);

// Create a GLTFLoader to load 3D models (not used in this specific code snippet)
const loader = new GLTFLoader();

// Parameters for the pyramid
const radius = 0.5; // Radius of the base of the pyramid
const height = 1; // Height of the pyramid
const sides = 5; // Number of sides of the base polygon (pentagon)

// Create geometry for a pyramid (cylinder with 0 radius at the top)
const geometry = new THREE.CylinderGeometry(0, radius, height, sides);

// Create an array of materials to apply to the faces of the pyramid
const materials = [
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
    new THREE.MeshBasicMaterial({ color: 0xff00ff }), // Magenta color material
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
```

```
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
    new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Cyan color material
];

// Create a wireframe material for the pyramid
const wireframeMaterial = new THREE.MeshBasicMaterial({
    color: 0x000000, // Black color for the wireframe
    wireframe: true, // Enable wireframe rendering
    opacity: 0.9, // Set the opacity of the wireframe
    transparent: true, // Enable transparency
});

// Create a mesh with the pyramid geometry and the array of materials
const pyramid = new THREE.Mesh(geometry, materials);
pyramid.add(new THREE.Mesh(geometry, wireframeMaterial)); // Add wireframe to the pyramid
for better visualization
scene.add(pyramid); // Add the pyramid to the scene

// Define the animation loop
function animate() {
    requestAnimationFrame(animate); // Request the next frame
    renderer.render(scene, camera); // Render the scene from the perspective of the camera
}

// Start the animation loop
animate();
```

Результат виконання завдань:

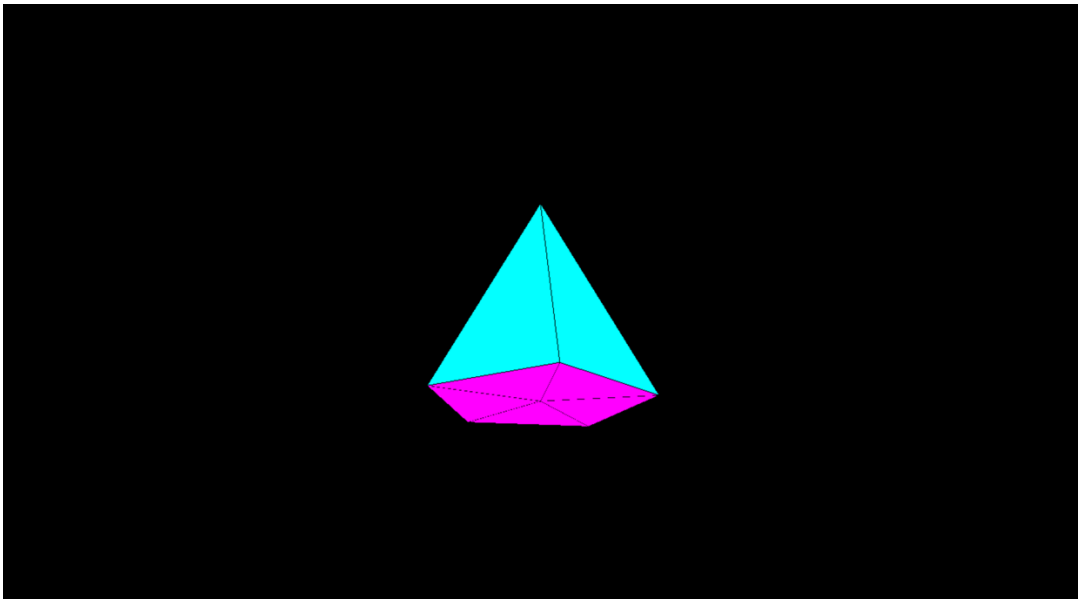

Рисунок 1.4 — Результат виконання програми

Завдання 1.4

а) Доповнити процедуру командою для виведення граней у вигляді набору точок

б) Доповнити процедуру командою для виведення граней у вигляді каркасу

в) Викликати процедуру двічі для побудови суміщеної точкової та суцільної моделі многогранника.

г) Викликати процедуру двічі для побудови суміщеної точкової та каркасної моделі многогранника.

д) Викликати процедуру двічі для побудови суцільної моделі многогранника з наведеними ребрами.

Лістинг програмного коду:

task4.js

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

import { OrbitControls } from 'three/addons/controls/OrbitControls.js';
import { GLTFLoader } from 'three/addons/loaders/GLTFLoader.js';

const scene = new THREE.Scene();

const camera = new THREE.PerspectiveCamera(50, window.innerWidth / window.innerHeight, 0.1,
1000);
camera.position.z = 3;

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

const controls = new OrbitControls( camera, renderer.domElement );
const loader = new GLTFLoader();

const radius = 0.5;
const height = 1;
const sides = 6;

const geometry = new THREE.CylinderGeometry(0, radius, height, sides);

const materials = [
  new THREE.MeshBasicMaterial({ color: 0x00ffff, opacity: 0.5, transparent: true }),
  new THREE.MeshBasicMaterial({ color: 0x00ffff, opacity: 0.5, transparent: true }),
```

```
  new THREE.MeshBasicMaterial({ color: 0x00ffff, opacity: 0.5, transparent: true }),
  new THREE.MeshBasicMaterial({ color: 0x00ffff, opacity: 0.5, transparent: true }),
  new THREE.MeshBasicMaterial({ color: 0x00ffff, opacity: 0.5, transparent: true }),
  new THREE.MeshBasicMaterial({ color: 0x00ffff, opacity: 0.5, transparent: true }),
];

const wireframeMaterial = new THREE.MeshBasicMaterial({
  color: 0xff00ff,
  wireframe: true,
  opacity: 0,
});


// Task A: Show only points
const pointsMaterialA = new THREE.PointsMaterial({
    color: 0xffff00,
    size: 0.09,
});
const pointsA = new THREE.Points(geometry, pointsMaterialA);
scene.add(pointsA);

// Task B: Show wireframe only
const resultB = new THREE.Mesh(geometry, wireframeMaterial);
scene.add(resultB);

// Task C: Show solid geometry with points
const pointsMaterialC = new THREE.PointsMaterial({
    color: 0xffff00,
    size: 0.09,
});
const pointsC = new THREE.Points(geometry, pointsMaterialC);
const resultC = new THREE.Mesh(geometry, materials);
resultC.add(pointsC);
scene.add(resultC);

// Task D: Show solid geometry with wireframe
const wireframeD = new THREE.Mesh(geometry, wireframeMaterial);
wireframeD.material.wireframe = true;
const resultD = new THREE.Mesh(geometry, materials);
resultD.add(wireframeD);
scene.add(resultD);

// Task E: Show solid geometry with wireframe and points
const wireframeE = new THREE.Mesh(geometry, wireframeMaterial);
wireframeE.material.wireframe = true;
const pointsMaterialE = new THREE.PointsMaterial({
    color: 0xffff00,
    size: 0.09,
});
const pointsE = new THREE.Points(geometry, pointsMaterialE);
const resultE = new THREE.Mesh(geometry, materials);
resultE.add(wireframeE, pointsE);
scene.add(resultE);
```

```
function animate() {
  requestAnimationFrame(animate);
  renderer.render(scene, camera);
}

animate();
```

Результат виконання завдань:

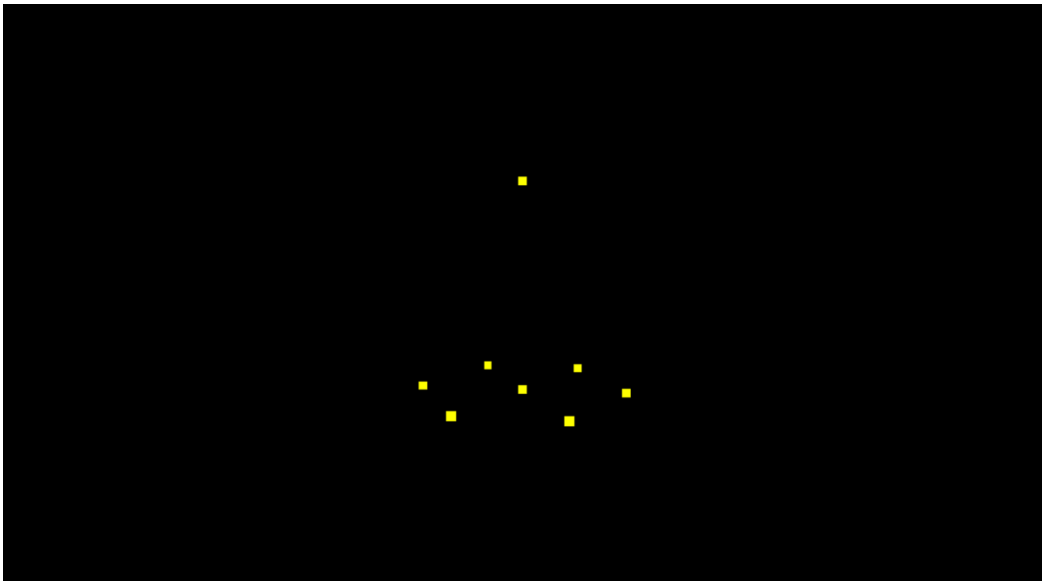

Рисунок 1.5 — Результат виконання програми A

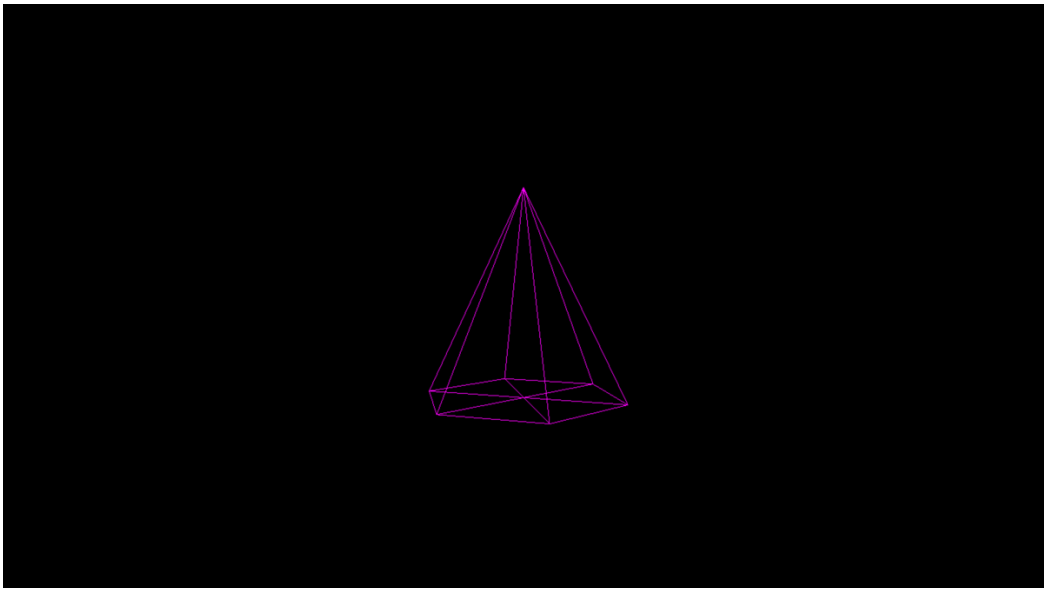

Рисунок 1.6 — Результат виконання програми B

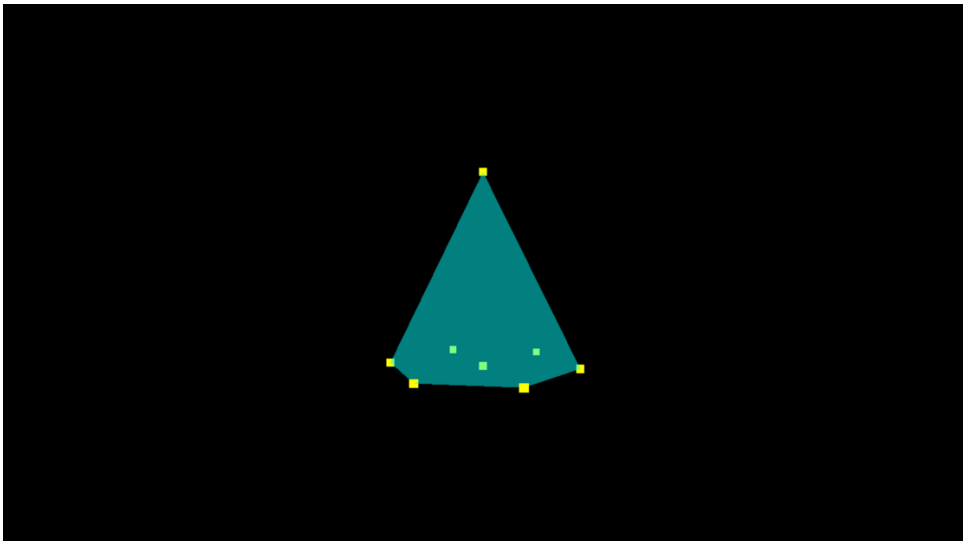

Рисунок 1.7 — Результат виконання програми C

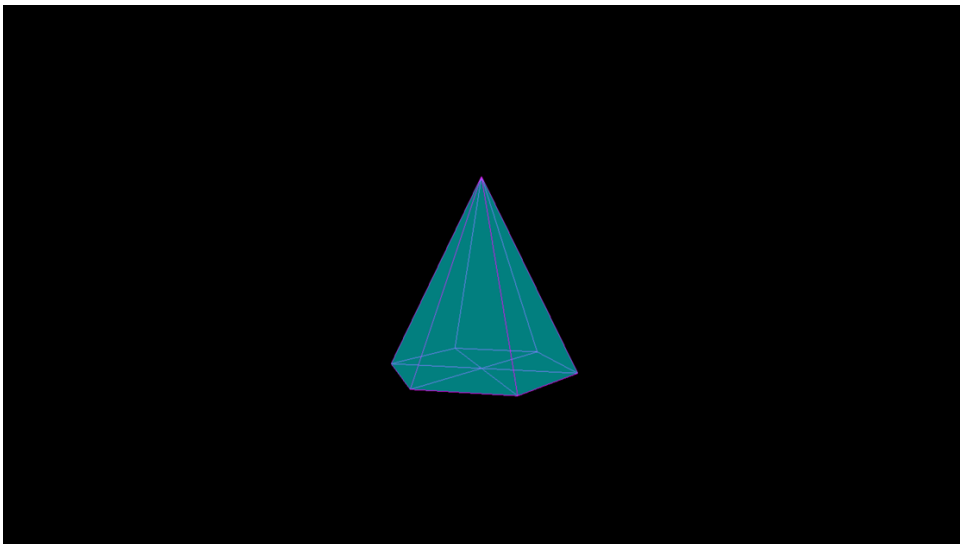

Рисунок 1.8 — Результат виконання програми D

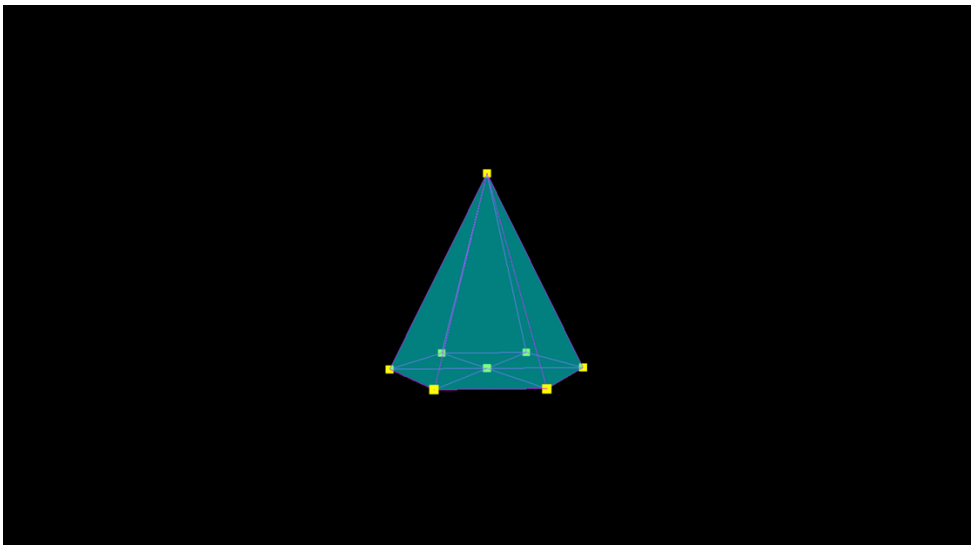

Рисунок 1.9 — Результат виконання програми E

ВИСНОВКИ

У цій лабораторній роботі ми розглянули основні концепції роботи з тривимірною графікою за допомогою бібліотеки Three.js та JavaScript. В ході виконання завдань ми здійснили наступне:

1. Виведення граней у вигляді набору точок: Ми створили точкове представлення геометричного об'єкта за допомогою `THREE.Points` та `THREE.PointsMaterial`.

2. Виведення граней у вигляді каркасу: Реалізували відображення геометричного об'єкта у вигляді каркасу за допомогою `THREE.Mesh` та матеріалу з опцією `wireframe`.

Ця лабораторна робота дозволила нам краще розібратися з основними можливостями бібліотеки Three.js та здійснити практичне застосування різноманітних методів візуалізації тривимірних об'єктів. Вона також сприяла розвитку навичок роботи з графічними елементами та їх взаємодії у веб-просторі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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