

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 № 3 з дисципліни
«Комп'ютерна графіка та обробка зображень»

Тема: «Відтворення кольорів в OpenGL з використанням білінійної
інтерполяції»

Виконав(ла)

ІП-15 Мешков 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 2.1	4
Завдання 2.2	6
Завдання 2.3	8
Завдання 2.4	10
Завдання 2.5	12
ВИСНОВКИ.....	20
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	21

ВСТУП

У світі комп'ютерної графіки інструменти, які дозволяють створювати візуальні ефекти, є надзвичайно важливими. Одним із таких інструментів є бібліотека Three.js, яка надає зручний інтерфейс для роботи з WebGL, що в свою чергу є інструментом для роботи з 3D-графікою в браузері, використовуючи мову програмування JavaScript.

Технології:







Three.js - це бібліотека JavaScript, яка дозволяє створювати 3D-графіку у браузері з використанням WebGL. Вона забезпечує зручний API для створення та управління сценами, об'єктами, матеріалами та освітленням, що робить процес розробки графічних додатків більш доступним.

JavaScript - це мова програмування, яка використовується для написання клієнтської частини браузерних додатків. Вона має широкі можливості для маніпулювання DOM, обробки подій користувача та виконання різних операцій, що робить її ідеальним вибором для розробки веб-додатків.

ХІД РОБОТИ

Завдання 2.1

Підібрати шляхом змішування складових компонент моделі RGB наступні кольори для граней куба: бузковий, лимонний, коричневий, морської хвилі, вишневий, салатний.

<u>Бузковий</u>	Lilac	#C8A2C8	
<u>Лимонний</u>	Lemon	#FDE910	
<u>Брунатний</u>	Brown	#993300	
<u>Морської хвилі</u>	Aqua	#00FFFF	
<u>Вишневий</u>	Cerise dark	#640023	
<u>Салатовий, Шартрез</u>	Chartreuse	#7FFF00	

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 { 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);
5

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

const loader = new GLTFLoader();
const geometry = new THREE.BoxGeometry();

const materials = [
  new THREE.MeshBasicMaterial({ color: 0xC8A2C8 }), // Lilac
  new THREE.MeshBasicMaterial({ color: 0xFDE910 }), // Lemon
  new THREE.MeshBasicMaterial({ color: 0x993300 }), // Brown
  new THREE.MeshBasicMaterial({ color: 0x00ffff }), // Aqua
  new THREE.MeshBasicMaterial({ color: 0x640023 }), // Cerise dark
  new THREE.MeshBasicMaterial({ color: 0x7FFF00 }), // Chartreuse
];

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

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

animate();
```

Результат виконання завдань:

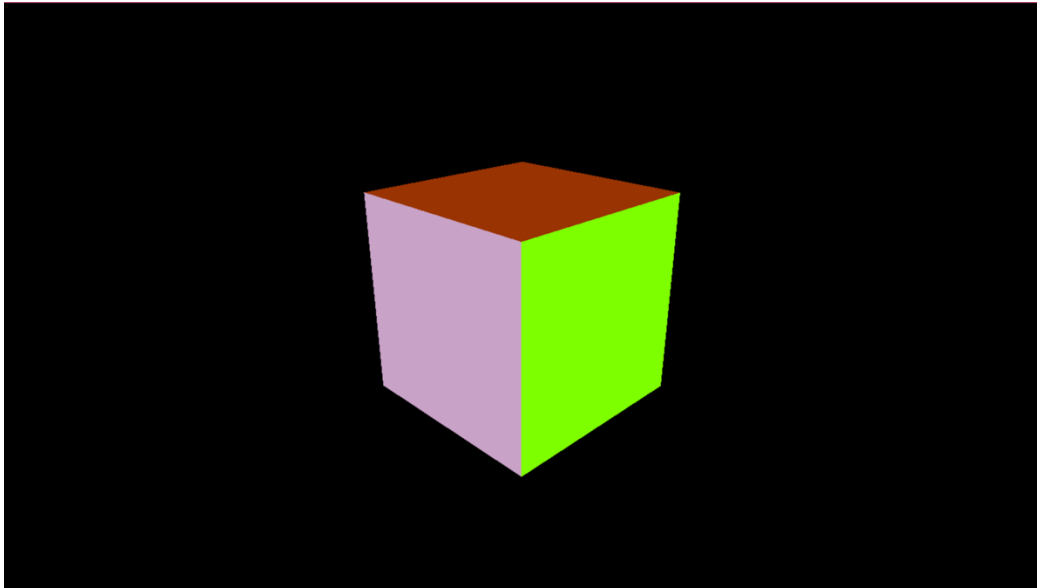


Рисунок 2.1 — Результат виконання програми

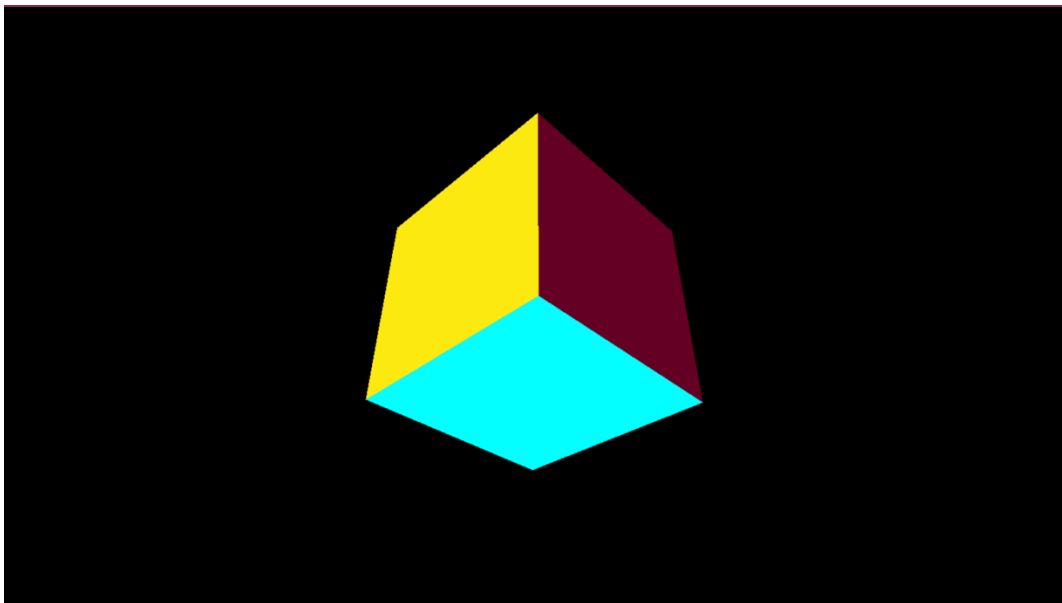


Рисунок 2.2 — Результат виконання програми

Завдання 2.2

Верхні вершини куба зафарбувати в червоний колір, нижні — в жовтий. Досягти плавного переходу кольорів від червоного до жовтого на бічних гранях.

Лістинг програмного коду:

Task2.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 geometry = new THREE.BoxGeometry();

const gradient = new THREE.ShaderMaterial({
  uniforms: {
    color1: {
      value: new THREE.Color("yellow")
    },
    color2: {
      value: new THREE.Color("red")
    }
  },
  vertexShader: `
    varying vec2 vUv;

    void main() {
      vUv = uv;
      gl_Position = projectionMatrix * modelViewMatrix * vec4(position,1.0);
    }
  `,
  fragmentShader: `
    uniform vec3 color1;
    uniform vec3 color2;

    varying vec2 vUv;

    void main() {

      gl_FragColor = vec4(mix(color1, color2, vUv.y), 1.0);
    }
  `
});

const materials = [
  gradient,
  gradient,
  new THREE.MeshBasicMaterial({ color: 0xFF0000 }),

```

```

    new THREE.MeshBasicMaterial({ color: 0xFFFF00 }),
    gradient,
    gradient
  ];

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

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

  animate();

```

Результат виконання завдань:

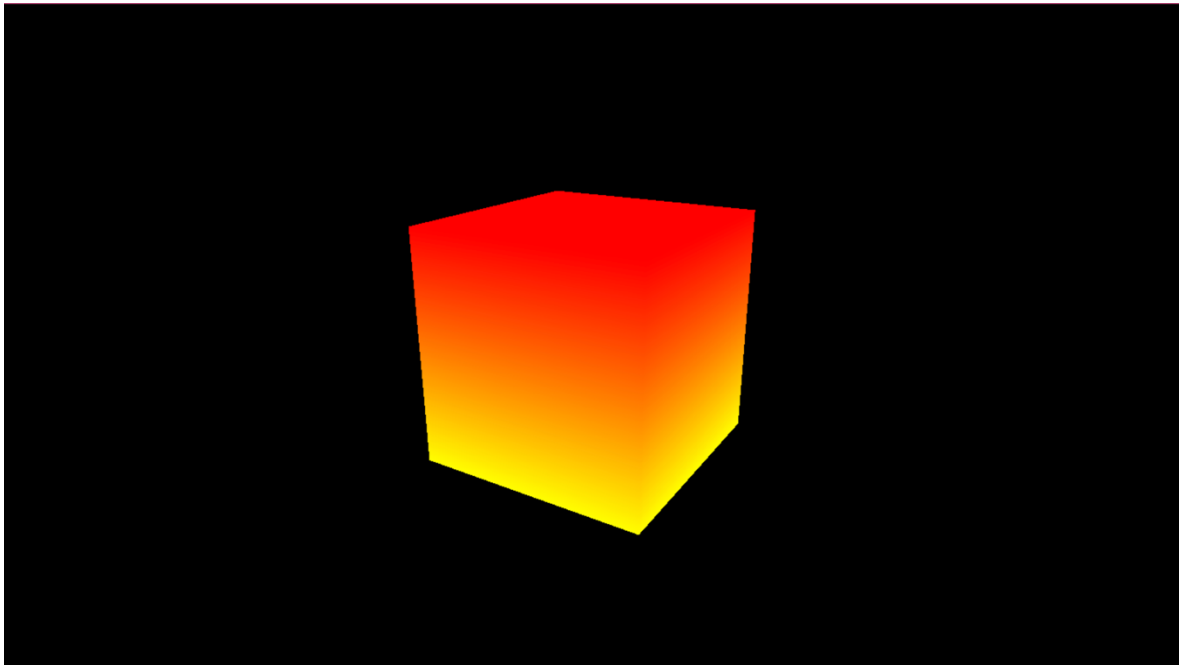


Рисунок 2.3 — Результат виконання програми

Завдання 2.3

Запрограмувати зміну кольорів всіх вершин з невеликим кроком зі зміною параметра Angle.

Лістинг програмного коду:

Task3.js

```

import * as THREE from 'three';
import { OrbitControls } from 'three/examples/jsm/controls/OrbitControls.js';
import { GLTFLoader } from 'three/examples/jsm/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 geometry = new THREE.BoxGeometry();

const gradient = new THREE.ShaderMaterial({
  uniforms: {
    color1: {
      value: new THREE.Color("rgb(255,255,0)")
    },
    color2: {
      value: new THREE.Color("rgb(255,0,0)")
    },
  },
  vertexShader: `
    varying vec2 vUv;

    void main() {
      vUv = uv;
      gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }

  `,
  fragmentShader: `
    uniform vec3 color1;
    uniform vec3 color2;
    varying vec2 vUv;
    void main() {
      gl_FragColor = vec4(mix(color1, color2, vUv.y), 1.0);
    }

  `,
});

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

let angle = 0;
function animate() {
  requestAnimationFrame(animate);
  angle += 0.01;

  gradient.uniforms.color1.value.setRGB(

```

```

    Math.sin(angle) * 0.5 + 0.5,
    Math.cos(angle) * 0.5 + 0.5,
    0.5
  );
  gradient.uniforms.color2.value.setRGB(
    Math.cos(angle) * 0.5 + 0.5,
    Math.sin(angle) * 0.5 + 0.5,
    0.5
  );

  renderer.render(scene, camera);
}

animate();

```

Результат виконання завдань:

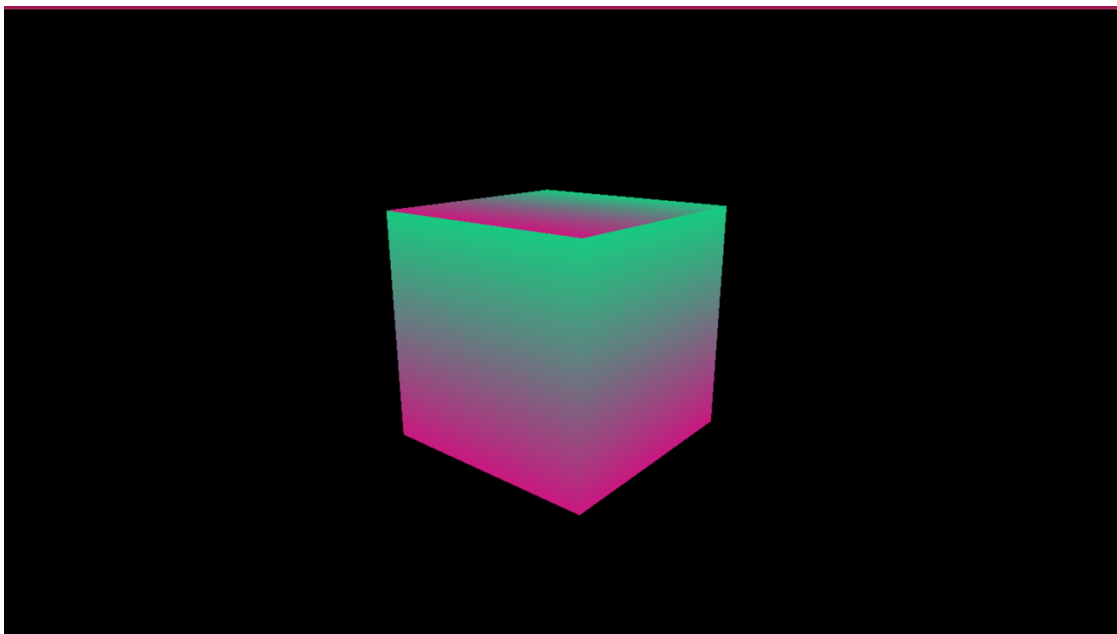


Рисунок 2.4 — Результат виконання програми

Завдання 2.4

Порівняти моделювання освітлення призматичної та циліндричної поверхонь.

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 prismGeometry = new THREE.CylinderGeometry(0, 0.5, 1, 40, 1);
const prismMaterial = new THREE.MeshPhongMaterial({ color: 0x7FFF00});
const prism = new THREE.Mesh(prismGeometry, prismMaterial);
prism.position.x = -0.7;
scene.add(prism);

const cylinderGeometry = new THREE.CylinderGeometry(0.5, 0.5, 1, 40);
const cylinderMaterial = new THREE.MeshPhongMaterial({ color: 0xC8A2C8});
const cylinder = new THREE.Mesh(cylinderGeometry, cylinderMaterial);
cylinder.position.x = 0.7;
scene.add(cylinder);

const ambientLight = new THREE.AmbientLight(0xffffff, 0.7);
scene.add(ambientLight);

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

animate();

```

Результат виконання завдань:

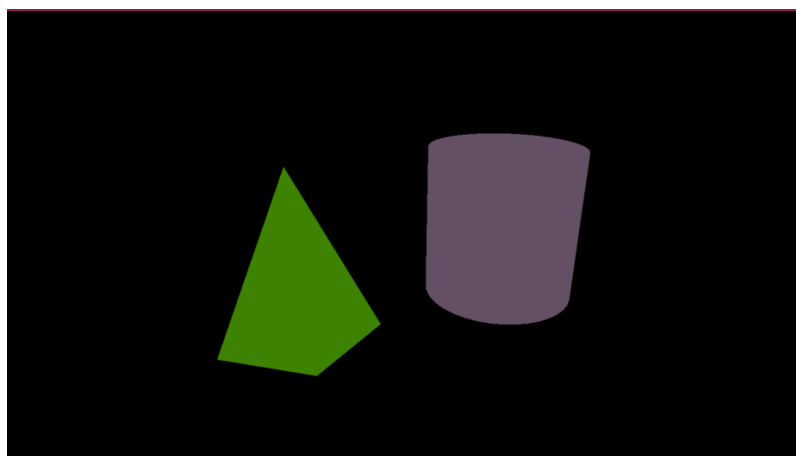


Рисунок 2.4 — Результат виконання програми(Освітлення однакове)

Завдання 2.5

Скопіювати проект із завдання 2.4 в нову папку. Замінити процедуру по черзі процедурами з прикладів 2.2, 2.3, 2.4, 2.5.

Лістинг програмного коду:

Task5.2.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 gradient = new THREE.ShaderMaterial({
  uniforms: {
    color1: {
      value: new THREE.Color("yellow")
    },
    color2: {
      value: new THREE.Color("red")
    }
  },
  vertexShader: `
    varying vec2 vUv;

    void main() {
      vUv = uv;
      gl_Position = projectionMatrix * modelViewMatrix * vec4(position,1.0);
    }
  `,
  fragmentShader: `
    uniform vec3 color1;
    uniform vec3 color2;

    varying vec2 vUv;

    void main() {
      gl_FragColor = vec4(mix(color1, color2, vUv.y), 1.0);
    }
  `
});
```

```

    });

const materials = [
  gradient,
  new THREE.MeshBasicMaterial({ color: 0xFF0000 }),
  new THREE.MeshBasicMaterial({ color: 0xFFFF00 }),
  gradient,
  gradient,
  gradient
];

const cylinderGeometry = new THREE.CylinderGeometry(0.5, 0.5, 1, 40);
const cylinder = new THREE.Mesh(cylinderGeometry, materials);
scene.add(cylinder);

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

animate();

```

Результат виконання завдань:

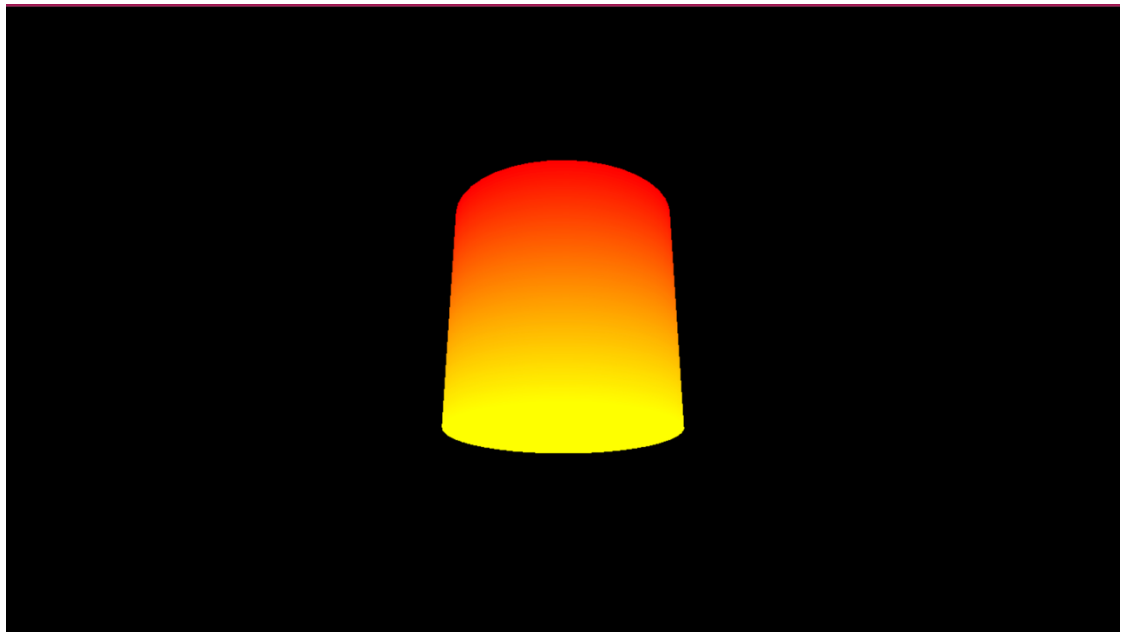


Рисунок 2.5 — Результат виконання програми

Лістинг програмного коду:

Task5.3.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 gradient = new THREE.ShaderMaterial({
  uniforms: {
    color1: {
      value: new THREE.Color("rgb(255,255,0)")
    },
    color2: {
      value: new THREE.Color("rgb(255,0,0)")
    },
  },
  vertexShader: `
    varying vec2 vUv;

    void main() {
      vUv = uv;
      gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }

  `,
  fragmentShader: `
    uniform vec3 color1;
    uniform vec3 color2;
    varying vec2 vUv;
    void main() {
      gl_FragColor = vec4(mix(color1, color2, vUv.y), 1.0);
    }

  `,
});

const cylinderGeometry = new THREE.CylinderGeometry(0.5, 0.5, 1, 40);
const cylinder = new THREE.Mesh(cylinderGeometry, gradient);
scene.add(cylinder);

let angle = 0;
function animate() {
  requestAnimationFrame(animate);
  angle += 0.01;

  gradient.uniforms.color1.value.setRGB(
    Math.sin(angle) * 0.5 + 0.5,

```

```
    Math.cos(angle) * 0.5 + 0.5,  
    0.5  
  );  
  gradient.uniforms.color2.value.setRGB(  
    Math.cos(angle) * 0.5 + 0.5,  
    Math.sin(angle) * 0.5 + 0.5,  
    0.5  
  );  
  
  renderer.render(scene, camera);  
}  
  
animate();
```

Результат виконання завдань:



Рисунок 2.6 — Результат виконання програми

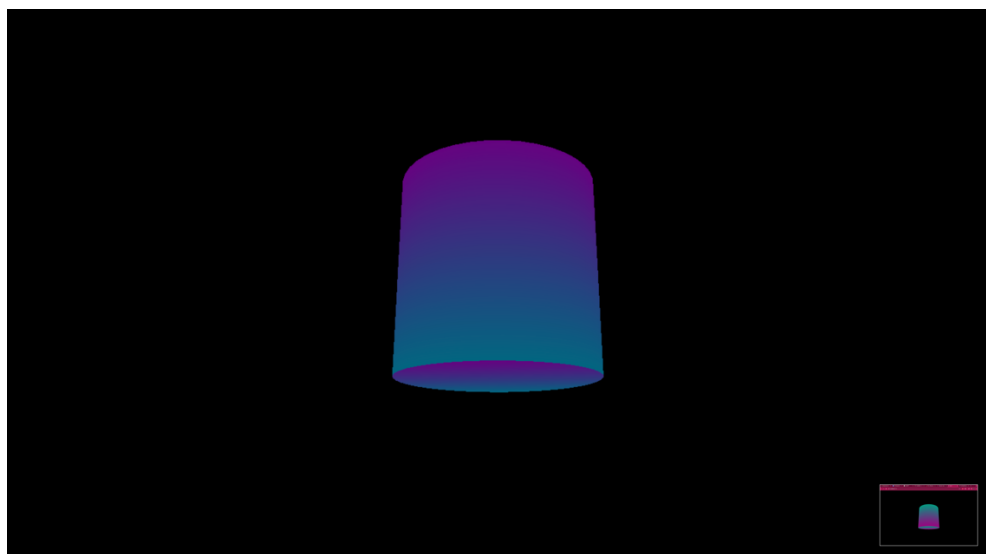


Рисунок 2.7 — Результат виконання програми

Лістинг програмного коду:

Task5.4.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 gradient = new THREE.ShaderMaterial({
  uniforms: {
    color1: {
      value: new THREE.Color("rgb(255,0,0)")
    },
    color2: {
      value: new THREE.Color("rgb(255,255,0)")
    },
  },
  vertexShader: `
    varying vec2 vUv;

    void main() {
      vUv = uv;
      gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }
  `,
  fragmentShader: `
    uniform vec3 color1;
    uniform vec3 color2;
    varying vec2 vUv;
    void main() {
      gl_FragColor = vec4(mix(color1, color2, vUv.y), 1.0);
    }
  `,
});

const coneGeometry = new THREE.ConeGeometry(0.5, 1, 64, 1, true);
const cone1 = new THREE.Mesh(coneGeometry, gradient);
```



```

cone1.rotation.x = Math.PI;
cone1.position.y = -0.5;

const cone2 = new THREE.Mesh(coneGeometry, gradient);
cone2.position.y = 0.5;

scene.add(cone1);
scene.add(cone2);

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

```

Результат виконання завдань:

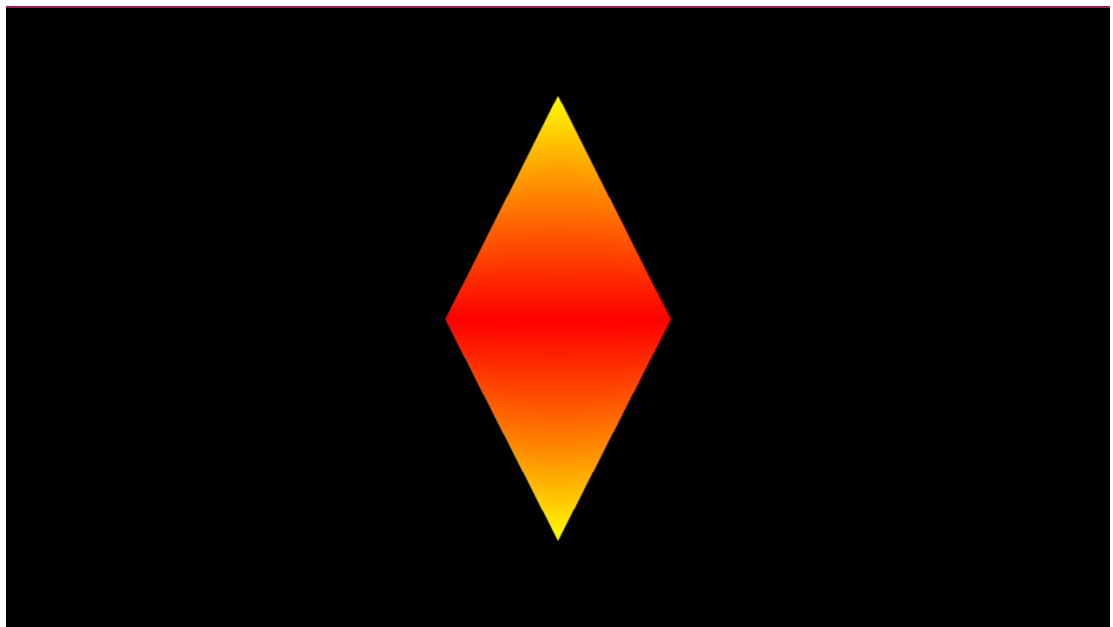


Рисунок 2.8 — Результат виконання програми

Лістинг програмного коду:

Task5.5.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 gradient = new THREE.ShaderMaterial({
  uniforms: {
    color1: {
      value: new THREE.Color("rgb(255,0,0)")

    },
    color2: {
      value: new THREE.Color("rgb(255,255,0)")
    },
  },
  vertexShader: `
    varying vec2 vUv;

    void main() {
      vUv = uv;
      gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }

  `,
  fragmentShader: `
    uniform vec3 color1;
    uniform vec3 color2;
    varying vec2 vUv;
    void main() {
      gl_FragColor = vec4(mix(color1, color2, vUv.y), 1.0);
    }

  `,
});

const coneGeometry = new THREE.ConeGeometry(0.5, 1, 64, 1, true);

const cone1 = new THREE.Mesh(coneGeometry, gradient);
cone1.rotation.x = Math.PI;
cone1.position.y = 0.49;

const cone2 = new THREE.Mesh(coneGeometry, gradient);
cone2.position.y = -0.49;

scene.add(cone1);
scene.add(cone2);

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

```

Результат виконання завдань:

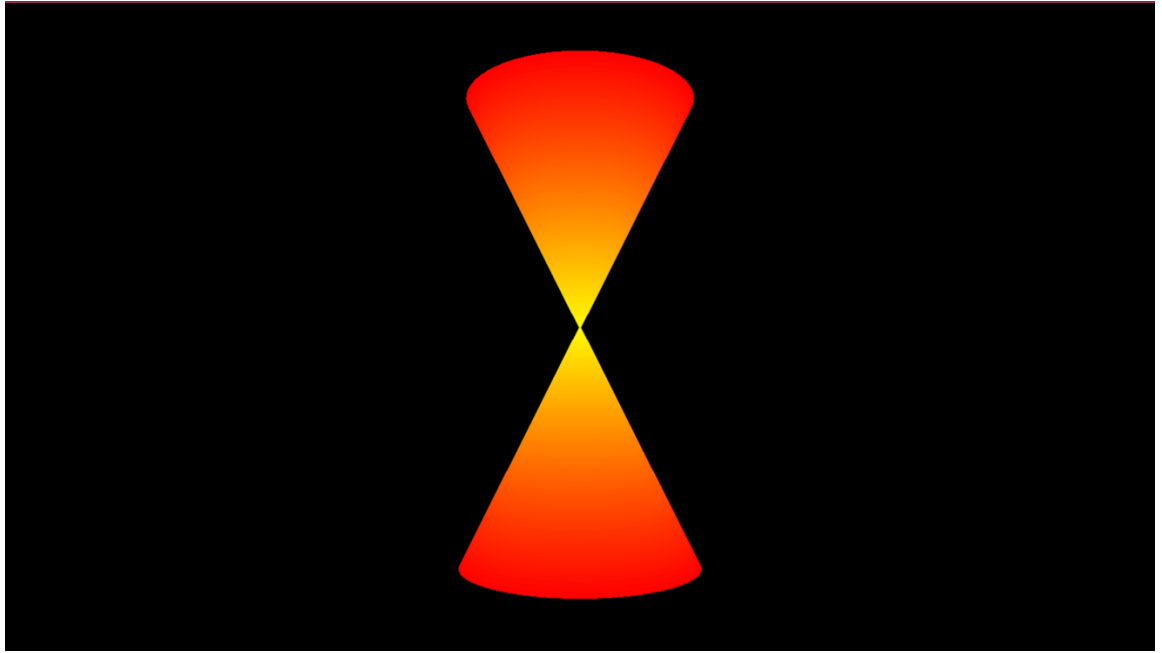


Рисунок 2.9 — Результат виконання програми

ВИСНОВКИ

Під час виконання лабораторної роботи ми активно використовували бібліотеку Three.js для розв'язання різноманітних завдань з малювання геометричних об'єктів у тривимірному просторі у веб-середовищі. Кожне завдання мало свою специфіку та ціль, яка спрямовувалась на розширення функціональності програми та практичне використання основних можливостей бібліотеки Three.js.

Спочатку ми реалізували візуалізацію циліндричної поверхні, де застосували основні концепції Three.js, такі як встановлення світла, кольору та нормалей до поверхні. Подальше дослідження включало аналіз різних прикладів, де ми використовували градієнтну зміну кольору, гіроскоп, пісочний годинник та трансформований циліндр, кожен з яких демонстрував різноманітні аспекти моделювання освітлення у Three.js.

Під час роботи над проектом ми отримали важливі знання та навички у моделюванні освітлення на геометричних об'єктах з використанням бібліотеки Three.js. Ці знання виявляться корисними у подальшій візуалізації складних сцен та вирішенні різноманітних завдань комп'ютерної графіки в веб-розробці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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