Content-

-plane with grid lines

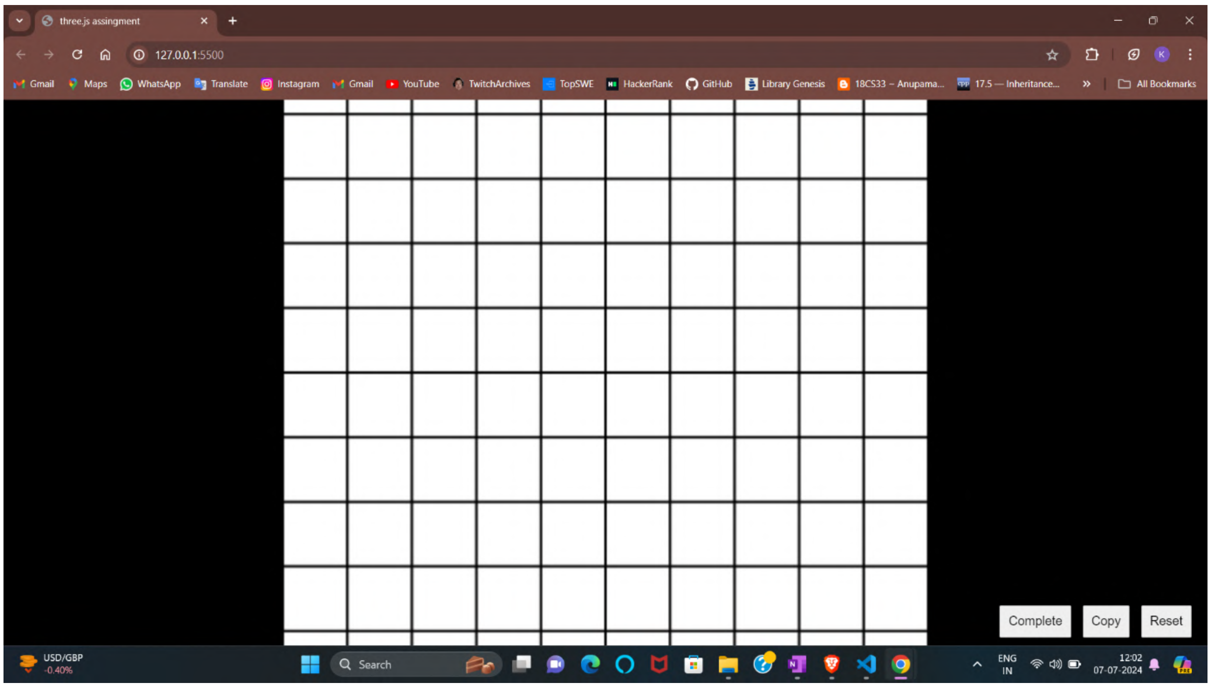
-placing vertices

-copy functionality

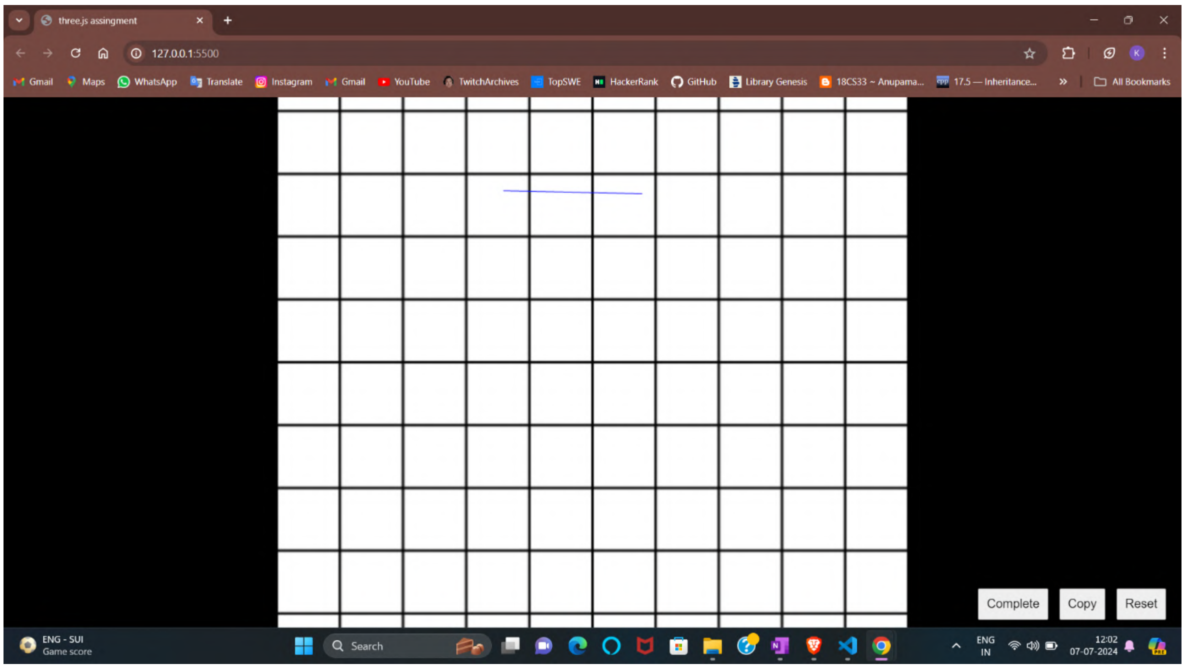
-reset functionality

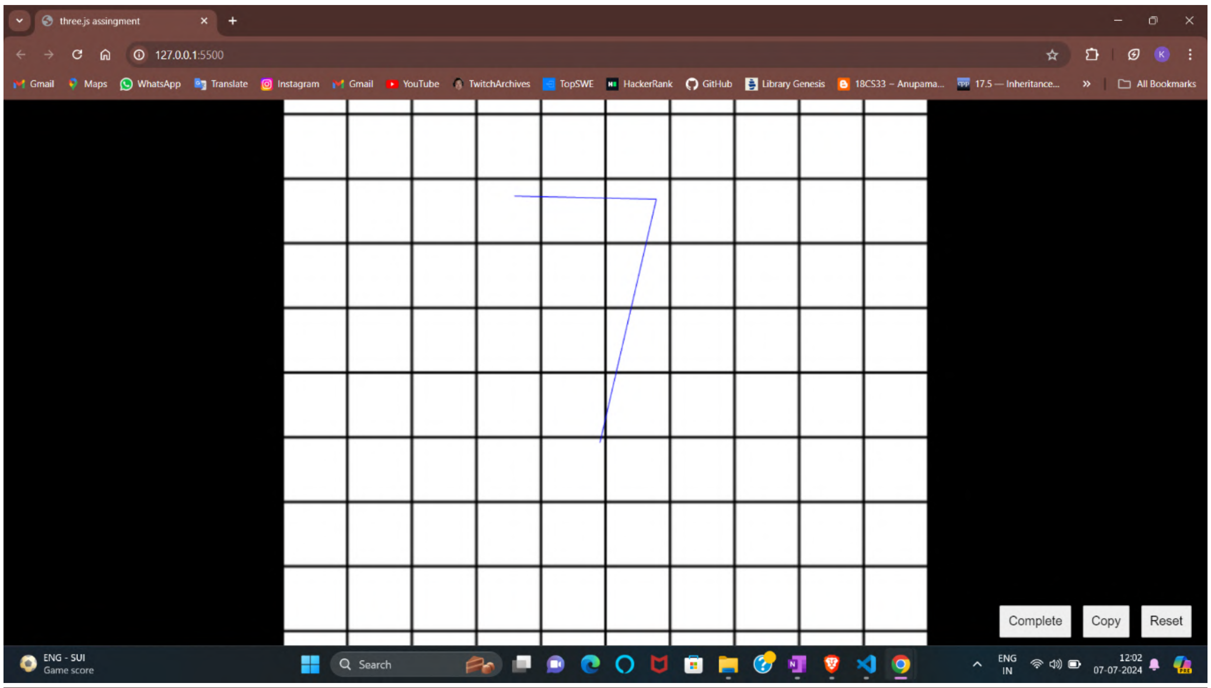
-code

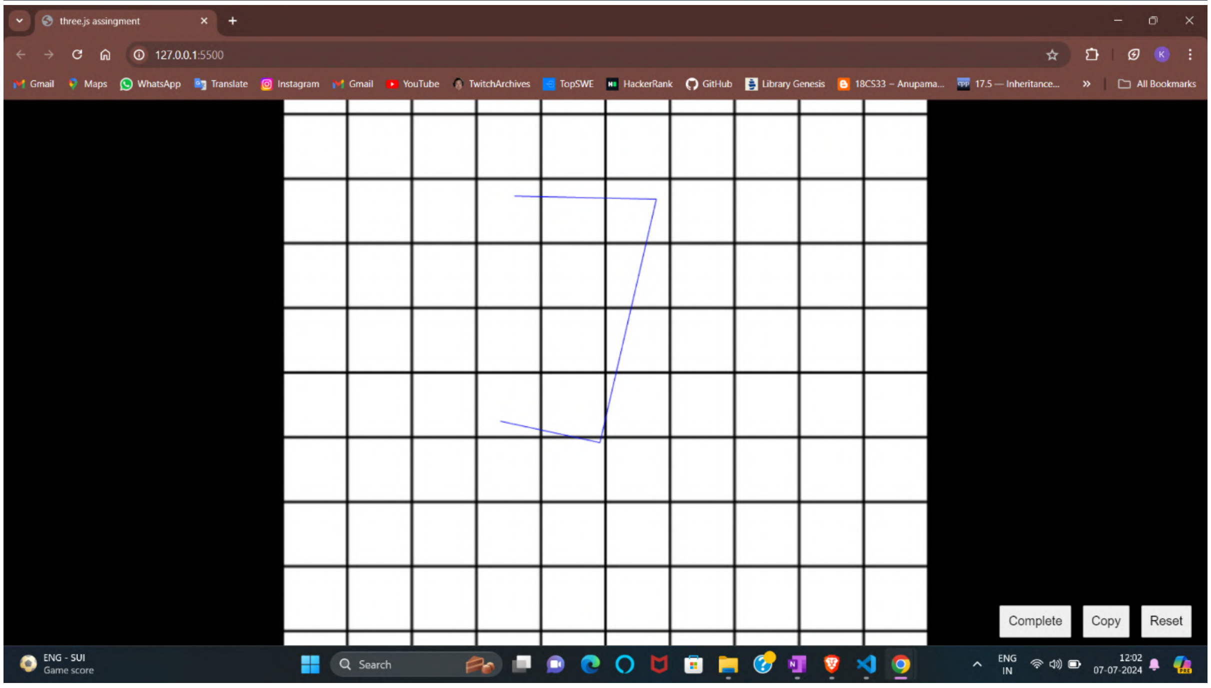
#1 Plane with grid lines

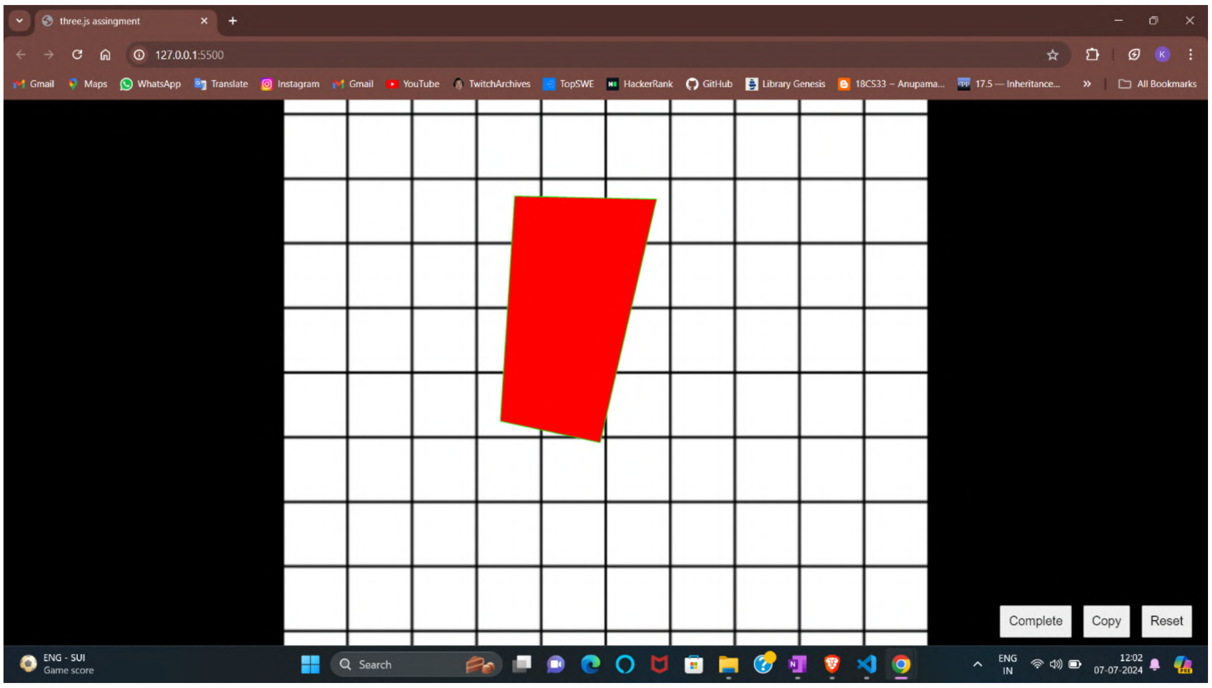


#2 Placing vertices using mouse

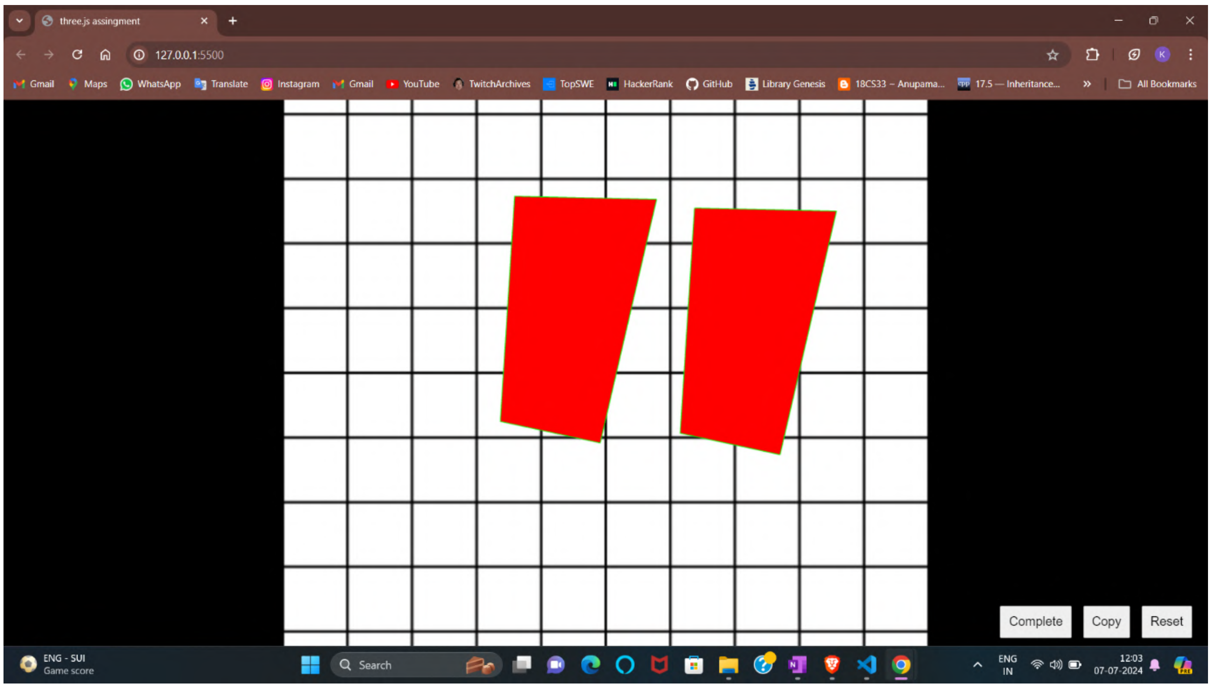


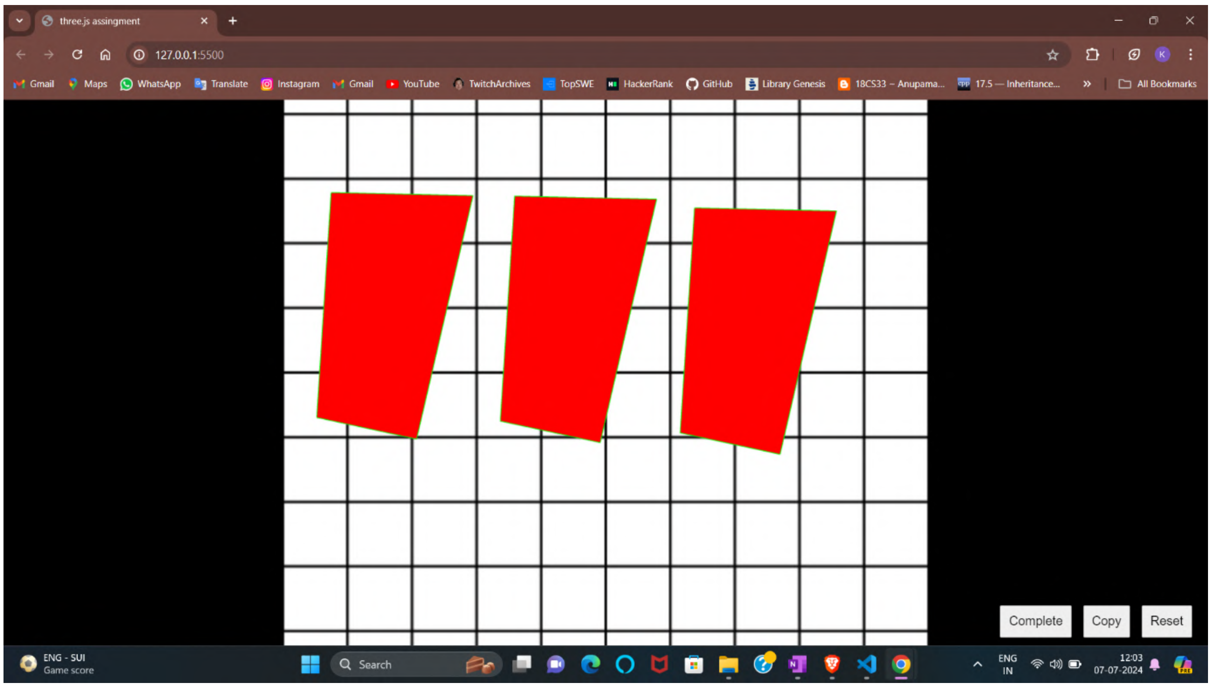

three.js assingment
127.0.0.1:5500
Complete
Copy
Reset


three.js assingment
127.0.0.1:5500
Complete
Copy
Reset

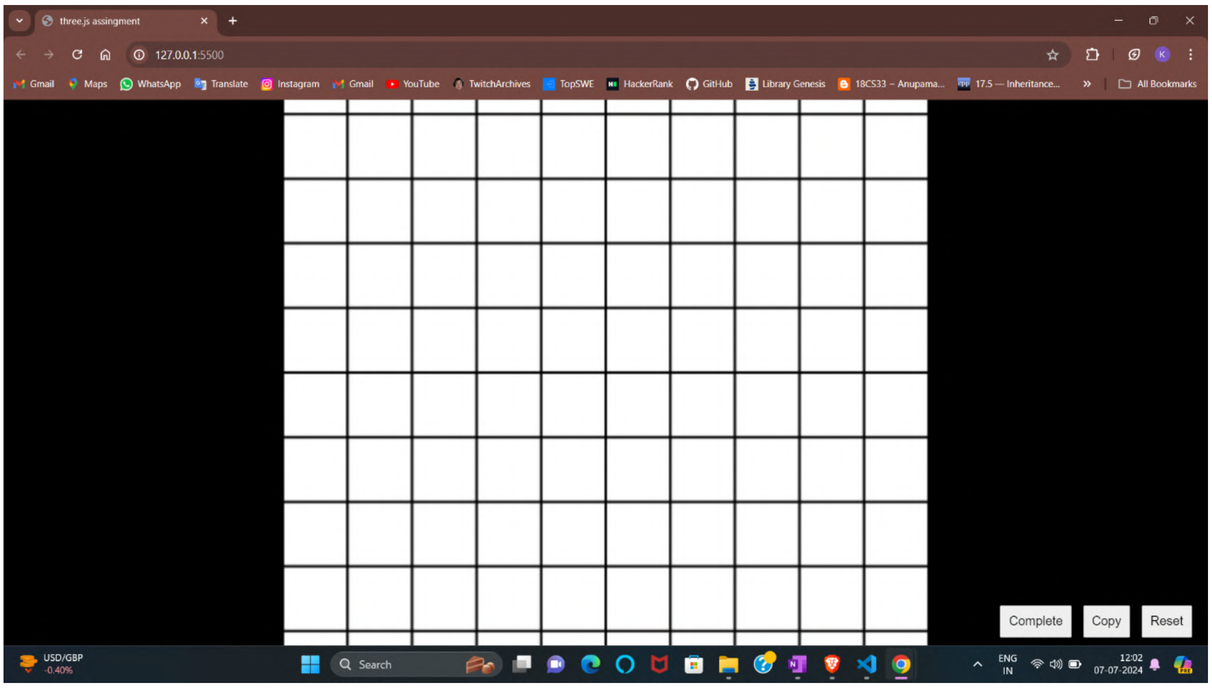


#3 Copy functionality





#4 Reset functionality



#5 code

App.js

```
const scene = new THREE.Scene();

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

const renderer = new THREE.WebGLRenderer({ antialias: true });
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);


const planeGeometry = new THREE.PlaneGeometry(20, 20);

const size = 512;
const divisions = 10;
const canvas = document.createElement('canvas');
canvas.width = size;
canvas.height = size;
const context = canvas.getContext('2d');

context.fillStyle = '#FFFFFF';
context.fillRect(0, 0, size, size);

context.strokeStyle = '#000000';
context.lineWidth = 2;

for (let i = 0; i <= divisions; i++) {
    let x = (size / divisions) * i;
    context.moveTo(x, 0);
    context.lineTo(x, size);
    context.moveTo(0, x);
    context.lineTo(size, x);
}
context.stroke();

const gridTexture = new THREE.CanvasTexture(canvas);
const planeMaterial = new THREE.MeshBasicMaterial({ map: gridTexture });

const plane = new THREE.Mesh(planeGeometry, planeMaterial);
plane.position.z = -1;
scene.add(plane);

class Polygon {
    constructor() {
        this.vertices = [];
        this.lines = [];
        this.edgeLines = null;
        this.polygonMesh = null;
    }

    addVertex(x, y) {
        const vertex = new THREE.Vector3(x, y, 0);
        this.vertices.push(vertex);
        if (this.vertices.length > 1) {
            const lineGeometry = new
THREE.BufferGeometry().setFromPoints([this.vertices[this.vertices.length
- 2], vertex]);
            const lineMaterial = new THREE.LineBasicMaterial({ color:
0x0000ff });
            const line = new THREE.Line(lineGeometry, lineMaterial);
            this.lines.push(line);
            scene.add(line);
        }
    }

    completePolygon() {
        if (this.vertices.length < 3) return;

        const shape = new THREE.Shape(this.vertices.map(v => new
THREE.Vector2(v.x, v.y)));
        const geometry = new THREE.ShapeGeometry(shape);
        const material = new THREE.MeshBasicMaterial({ color: 0xff0000,
side: THREE.DoubleSide });
        this.polygonMesh = new THREE.Mesh(geometry, material);
        scene.add(this.polygonMesh);


        this.lines.forEach(line => scene.remove(line));
        this.lines = [];


        const edgeGeometry = new
THREE.BufferGeometry().setFromPoints([...this.vertices,
this.vertices[0]]);
        const edgeMaterial = new THREE.LineBasicMaterial({ color:
0x00ff00 });
        this.edgeLines = new THREE.Line(edgeGeometry, edgeMaterial);
        scene.add(this.edgeLines);
    }
}

class SceneController {
    constructor() {
        this.currentPolygon = new Polygon();
        this.polygons = [];
        this.isCopying = false;
    }

    addVertex(x, y) {
        if (this.isCopying) return;
        this.currentPolygon.addVertex(x, y);
    }

    completePolygon() {
        this.currentPolygon.completePolygon();
        this.polygons.push(this.currentPolygon);
        this.currentPolygon = new Polygon();
    }

    copyPolygon() {
        if (!this.polygons.length) return;

        const lastPolygon = this.polygons[this.polygons.length - 1];
        const clonedGeometry = lastPolygon.polygonMesh.geometry.clone();
        const clonedMaterial = lastPolygon.polygonMesh.material.clone();
        const clonedMesh = new THREE.Mesh(clonedGeometry,
clonedMaterial);
        scene.add(clonedMesh);

        const edgeGeometry = lastPolygon.edgeLines.geometry.clone();
        const edgeMaterial = lastPolygon.edgeLines.material.clone();
        const edgeLines = new THREE.Line(edgeGeometry, edgeMaterial);
        scene.add(edgeLines);

        this.isCopying = true;

        function onMouseMove(event) {
            const rect = renderer.domElement.getBoundingClientRect();
            const x = ((event.clientX - rect.left) / rect.width) * 2 - 1;
            const y = -((event.clientY - rect.top) / rect.height) * 2 +
1;
            clonedMesh.position.set(x * 10, y * 10, 0);
            edgeLines.position.set(x * 10, y * 10, 0);
        }

        window.addEventListener('mousemove', onMouseMove);

        window.addEventListener('click', function placePolygon() {
            window.removeEventListener('mousemove', onMouseMove);
            window.removeEventListener('click', placePolygon);
            sceneController.isCopying = false;
            const newPolygon = new Polygon();
            newPolygon.polygonMesh = clonedMesh;
            newPolygon.edgeLines = edgeLines;
            sceneController.polygons.push(newPolygon);
        });
    }

    reset() {
        this.polygons.forEach(polygon => {
            if (polygon.polygonMesh) scene.remove(polygon.polygonMesh);
            if (polygon.edgeLines) scene.remove(polygon.edgeLines);
        });
        this.polygons = [];
        this.currentPolygon = new Polygon();
    }
}

const sceneController = new SceneController();

window.addEventListener('click', (event) => {
    const rect = renderer.domElement.getBoundingClientRect();
    if (
        event.clientX >= rect.left &&
        event.clientX <= rect.right &&
        event.clientY >= rect.top &&
        event.clientY <= rect.bottom
    ) {
        const x = ((event.clientX - rect.left) / rect.width) * 2 - 1;
        const y = -((event.clientY - rect.top) / rect.height) * 2 + 1;
        sceneController.addVertex(x * 10, y * 10);
    }
});

document.getElementById('completeBtn').addEventListener('click', (event)
=> {
    event.stopPropagation();
    sceneController.completePolygon();
});

document.getElementById('copyBtn').addEventListener('click', (event) => {
    event.stopPropagation();
    sceneController.copyPolygon();
});

document.getElementById('resetBtn').addEventListener('click', (event) =>
{
    event.stopPropagation();
    sceneController.reset();
});

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


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

Index.js

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="style.css">
    <title>three.js assingment</title>
</head>
<body>
    <div id="buttons">
```

```html
        <button id="completeBtn">Complete</button>
        <button id="copyBtn">Copy</button>
        <button id="resetBtn">Reset</button>
    </div>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"><
/script>
    <script src="app.js"></script>
</body>
</html>
```

Style.css

```css
body { margin: 0; }
canvas { display: block; }
#buttons {
  position: absolute;
  bottom: 10px;
  right: 10px;
}
button {
  margin-right: 10px;
  padding: 10px;
  font-size: 16px;
}
```