#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>2D Polygon Creator</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div id="canvas-container"></div>
    <button id="complete-btn">Complete</button>
    <button id="copy-btn">Copy</button>
    <button id="reset-btn">Reset</button>

    <script
src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></scri
pt>
    <script src="script.js"></script>
</body>
</html>
```

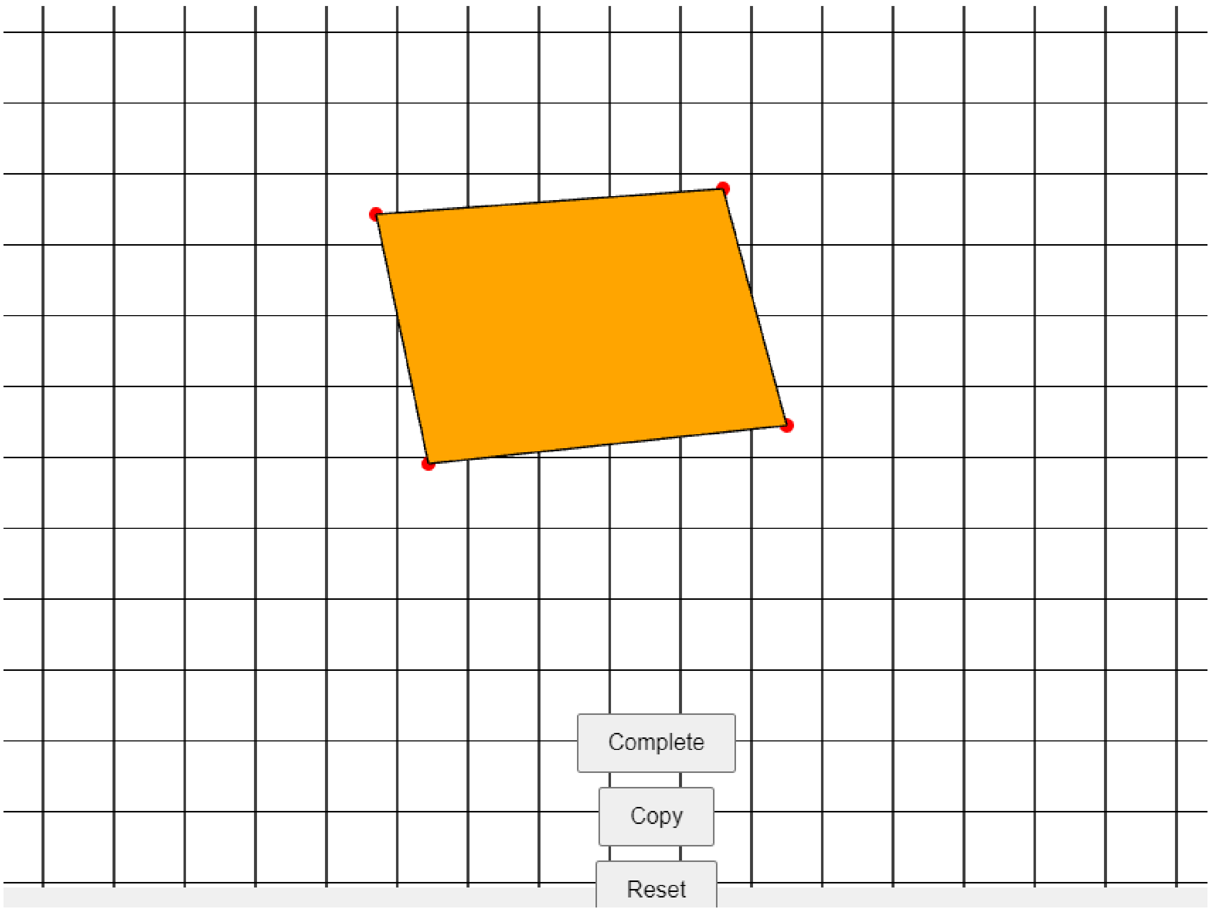
#styles.css

```css
body {
    margin: 0;
    overflow: hidden;
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    height: 100vh;
    background-color: #f0f0f0;
}

#canvas-container {
    width: 100%;
    height: 80%;
}

button {
    margin: 5px;
    padding: 10px 20px;
    font-size: 16px;
    cursor: pointer;
}
```

script.js

```
const scene = new THREE.Scene();
const camera = new THREE.OrthographicCamera(window.innerWidth / -2,
window.innerWidth / 2, window.innerHeight / 2, window.innerHeight / -2, 1,
1000);
const renderer = new THREE.WebGLRenderer({ antialias: true });

renderer.setSize(window.innerWidth, window.innerHeight);
document.getElementById('canvas-container').appendChild(renderer.domElement);

camera.position.z = 5;

const planeGeometry = new THREE.PlaneGeometry(window.innerWidth,
window.innerHeight);
const planeMaterial = new THREE.MeshBasicMaterial({ color: 0xffffff });
const plane = new THREE.Mesh(planeGeometry, planeMaterial);
scene.add(plane);


const gridSpacing = 50;
const gridHelper = new THREE.GridHelper(window.innerWidth, window.innerWidth /
gridSpacing, 0x000000, 0x000000);
gridHelper.rotation.x = Math.PI / 2;
scene.add(gridHelper);

let vertices = [];
let polygons = [];
let currentPolygon = null;


class Polygon {
    constructor(vertices) {
        this.vertices = vertices;
        this.color = 0xffa500;
        this.createMesh();
    }

    createMesh() {
        const shape = new THREE.Shape();
        this.vertices.forEach((vertex, index) => {
            if (index === 0) {
                shape.moveTo(vertex.x, vertex.y);
            } else {
                shape.lineTo(vertex.x, vertex.y);
            }
        });
        shape.lineTo(this.vertices[0].x, this.vertices[0].y);

        const geometry = new THREE.ShapeGeometry(shape);
        const material = new THREE.MeshBasicMaterial({ color: this.color,
side: THREE.DoubleSide });
        this.mesh = new THREE.Mesh(geometry, material);

        const lineGeometry = new
THREE.BufferGeometry().setFromPoints(this.vertices.map(v => new
THREE.Vector3(v.x, v.y, 0)));
        const lineMaterial = new THREE.LineBasicMaterial({ color: 0x000000 });
        this.edges = new THREE.LineLoop(lineGeometry, lineMaterial);

        scene.add(this.mesh);
        scene.add(this.edges);
    }

    copy() {
        const newVertices = this.vertices.map(v => ({ x: v.x, y: v.y }));
        return new Polygon(newVertices);
    }

    moveTo(x, y) {
        const dx = x - this.vertices[0].x;
        const dy = y - this.vertices[0].y;
        this.vertices.forEach(v => {
            v.x += dx;
            v.y += dy;
        });
        this.mesh.position.set(dx, dy, 0);
        this.edges.position.set(dx, dy, 0);
    }
}

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

window.addEventListener('resize', () => {
    renderer.setSize(window.innerWidth, window.innerHeight);
    camera.left = window.innerWidth / -2;
    camera.right = window.innerWidth / 2;
    camera.top = window.innerHeight / 2;
    camera.bottom = window.innerHeight / -2;
    camera.updateProjectionMatrix();

    const gridHelperSize = window.innerWidth / gridSpacing;
    gridHelper.scale.set(gridHelperSize, gridHelperSize, 1);
});

renderer.domElement.addEventListener('click', event => {
    if (!currentPolygon) {
        const x = event.clientX - window.innerWidth / 2;
        const y = -(event.clientY - window.innerHeight / 2);
        vertices.push({ x, y });

        const pointGeometry = new THREE.CircleGeometry(5, 32);
        const pointMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000
});
        const point = new THREE.Mesh(pointGeometry, pointMaterial);
        point.position.set(x, y, 0);
        scene.add(point);
    }
});


document.getElementById('complete-btn').addEventListener('click', () => {
    if (vertices.length > 2) {
        currentPolygon = new Polygon(vertices);
        polygons.push(currentPolygon);
        vertices = [];
    }
});


document.getElementById('copy-btn').addEventListener('click', () => {
    if (currentPolygon) {

        const copiedPolygon = currentPolygon.copy();

        document.addEventListener('mousemove', onMouseMove);
        document.addEventListener('click', onMouseClick);

        function onMouseMove(event) {
            const x = event.clientX - window.innerWidth / 2;
            const y = -(event.clientY - window.innerHeight / 2);
            copiedPolygon.moveTo(x, y);
        }

        function onMouseClick() {
            document.removeEventListener('mousemove', onMouseMove);
            document.removeEventListener('click', onMouseClick);
            polygons.push(copiedPolygon);
        }
    }
});

document.getElementById('reset-btn').addEventListener('click', () => {
    polygons.forEach(p => {
        scene.remove(p.mesh);
        scene.remove(p.edges);
    });
    polygons = [];
    currentPolygon = null;
    vertices = [];
    scene.children.forEach(child => {
        if (child.isMesh && child !== plane) {
            scene.remove(child);
        }
    });
});
```

Image:


Complete
Copy
Reset