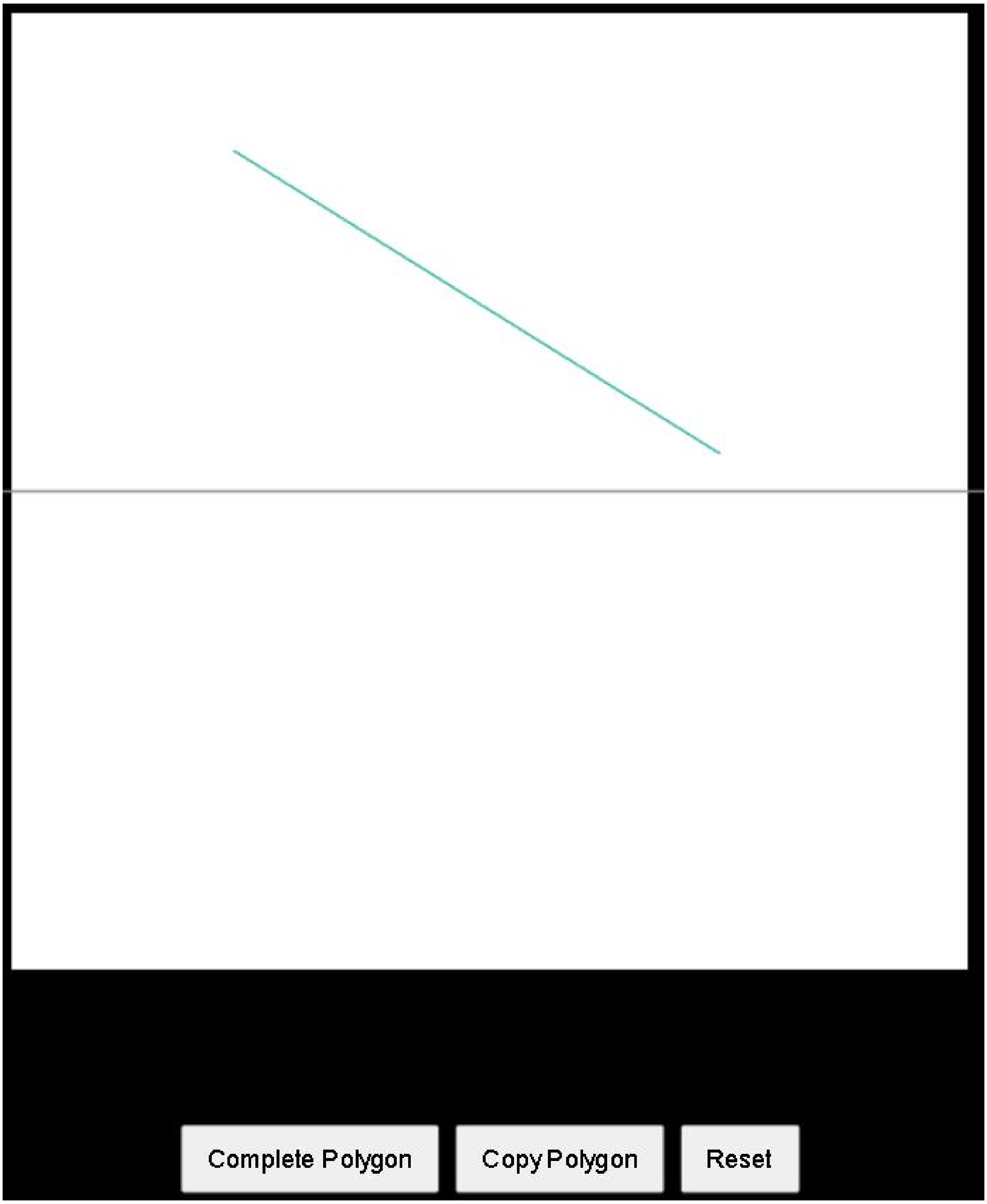
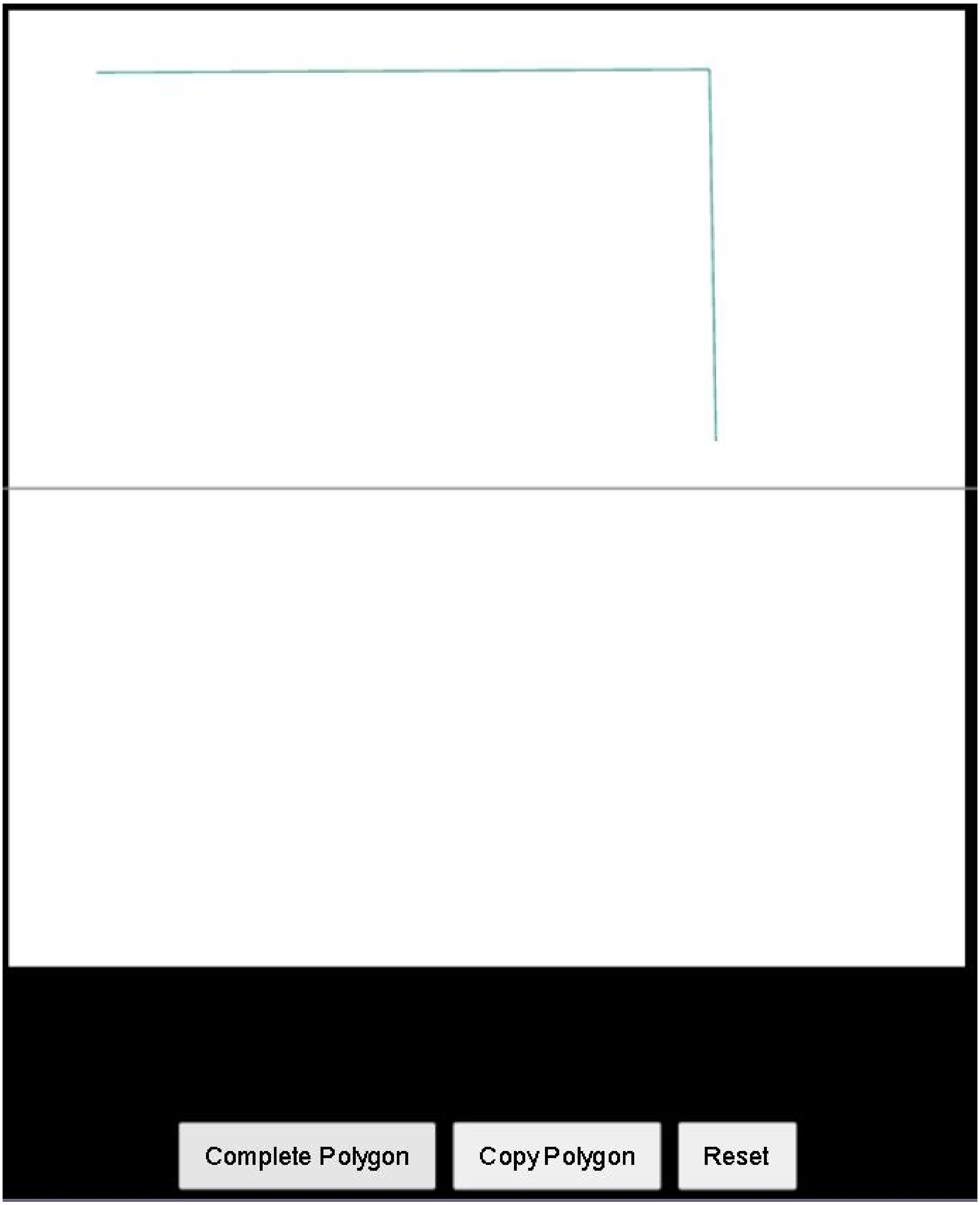
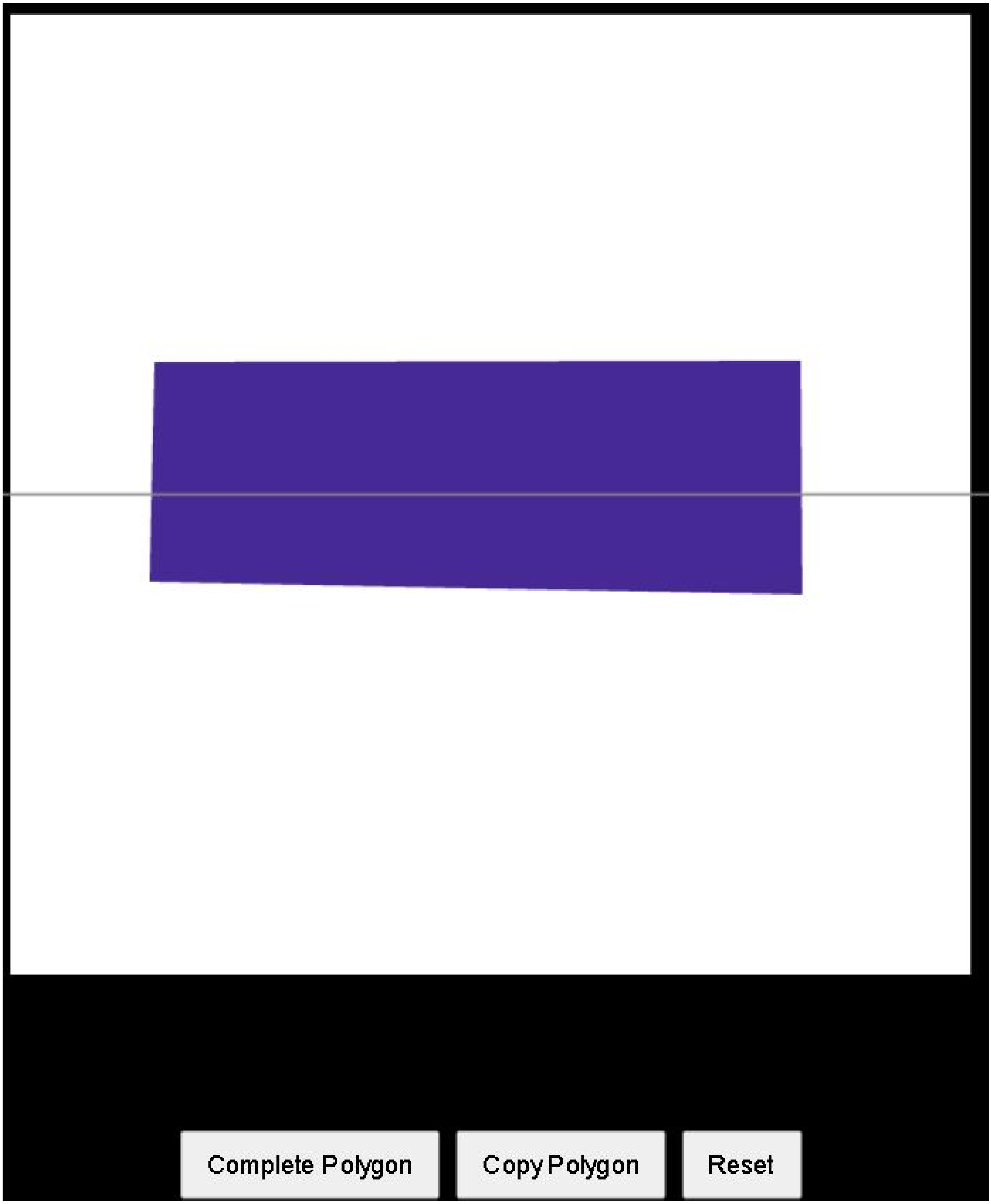
Two Vertices

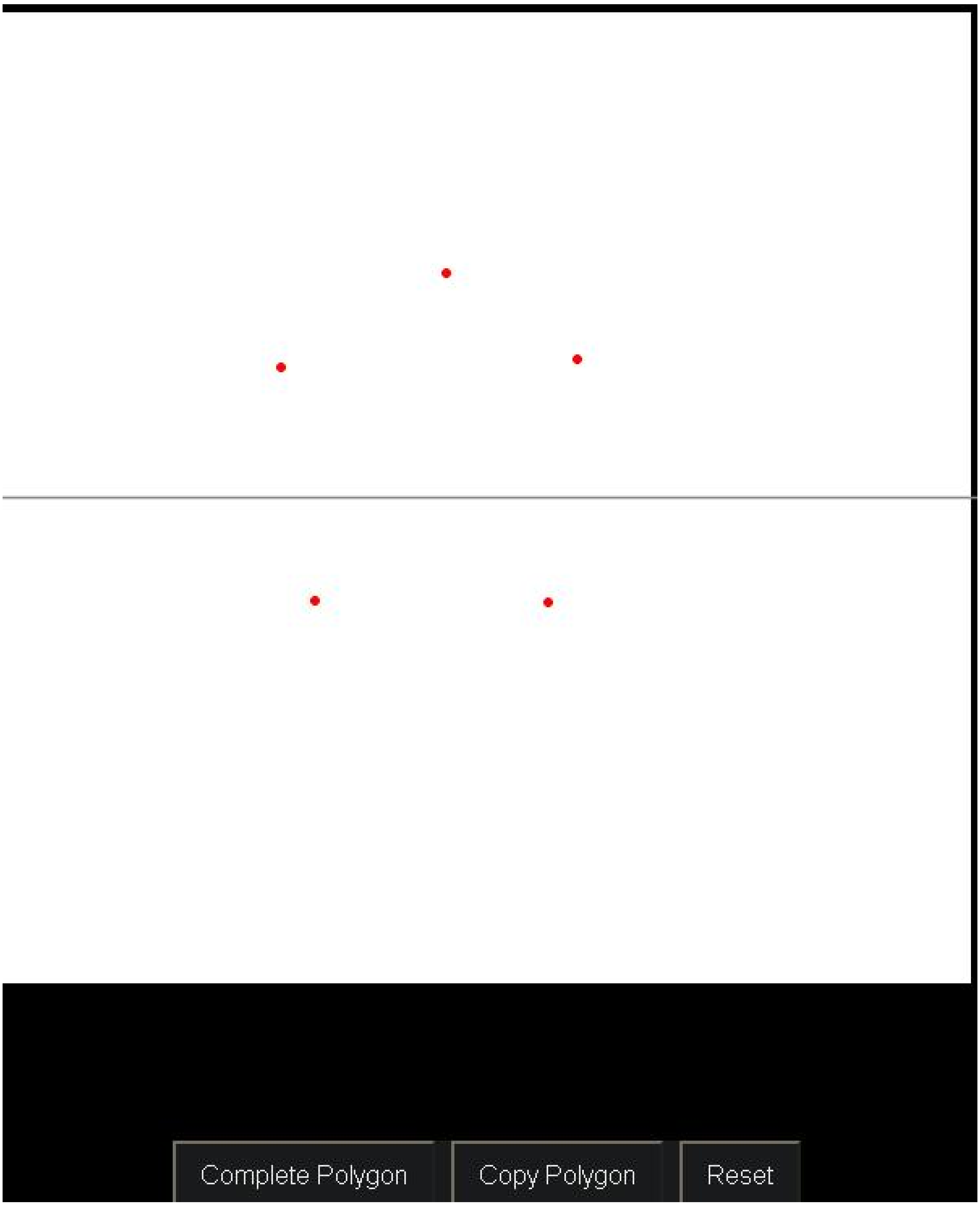


Three vertices

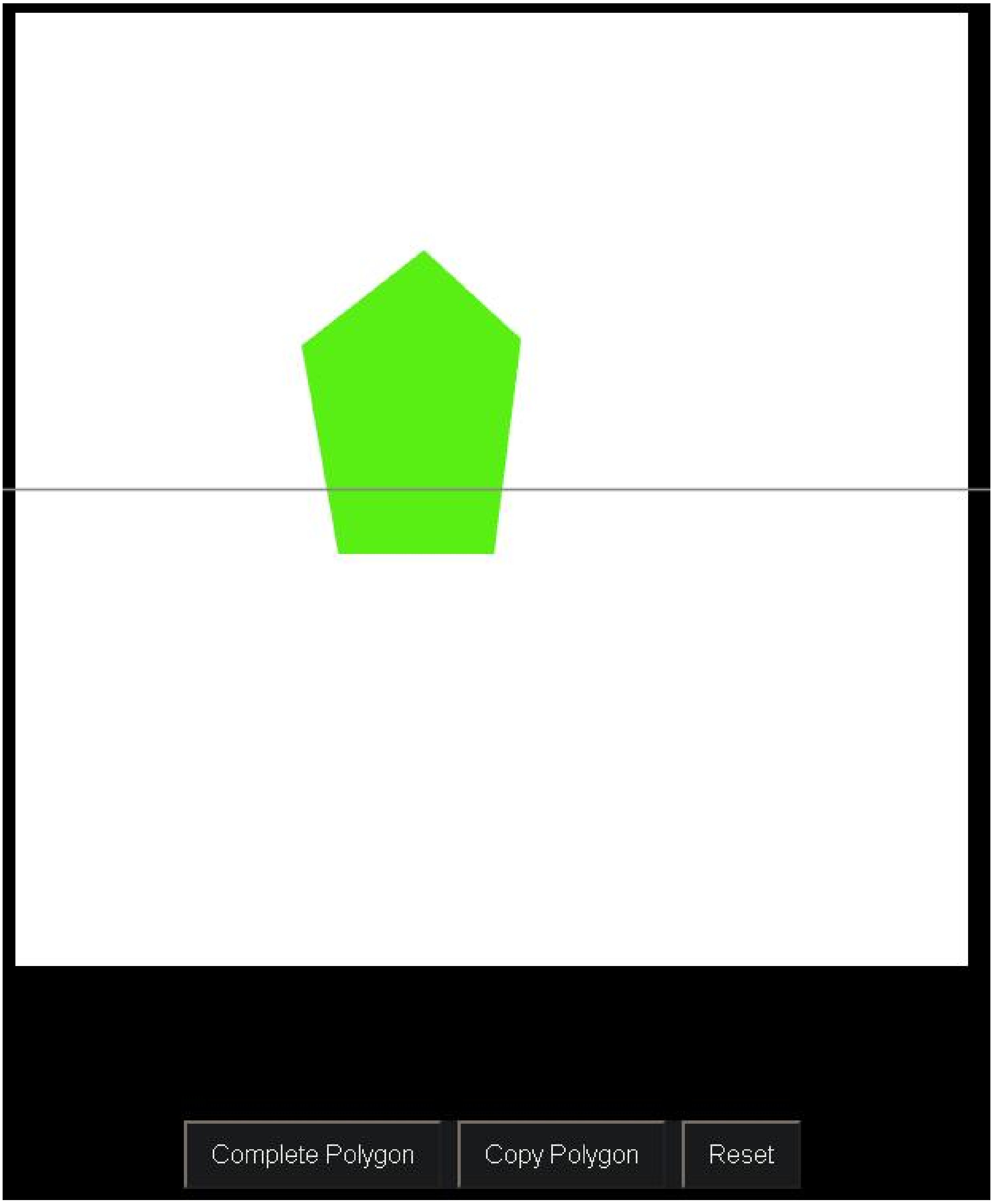


Four vertices




Complete Polygon
Copy Polygon
Reset

Pentagon



```html
<!-- ** index.html  ** -->

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Three.js Polygon Drawing</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></script>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="canvas-container"></div>
    <div class="controls">
        <button id="complete-btn">Complete Polygon</button>
        <button id="copy-btn">Copy Polygon</button>
        <button id="reset-btn">Reset</button>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

```css
/* ** style.css ** */

body {
    margin: 0;
    overflow: hidden;
    font-family: Arial, sans-serif;
}

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

.controls {
    position: absolute;
    bottom: 20px;
    left: 50%;
    transform: translateX(-50%);
    display: flex;
    gap: 10px;
}

button {
    padding: 10px 15px;
    font-size: 16px;
    cursor: pointer;
}
```

```javascript
// ** script.js ** //

let scene, camera, renderer, raycaster, mouse;
let ground, gridHelper;
let polygons = [];
let currentVertices = [];
let currentPolygon = null;
let isCopying = false;
let copiedPolygon = null;
let polygonCopies = []; // Track placed polygon copies

class Polygon {
    constructor(vertices, filled) {
        this.vertices = vertices;
        this.color = Math.random() * 0xffffff;
        this.filled = filled;

        if (this.filled) {
            this.mesh = this.createMesh();
            scene.add(this.mesh);
        } else {
            this.line = this.createLine();
            scene.add(this.line);
        }
    }

    createMesh() {
        const shape = new THREE.Shape();
        shape.moveTo(this.vertices[0].x, this.vertices[0].y);
        for (let i = 1; i < this.vertices.length; i++) {
            shape.lineTo(this.vertices[i].x, this.vertices[i].y);
        }
        shape.lineTo(this.vertices[0].x, this.vertices[0].y); // Close the shape

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

    createLine() {
        const points = this.vertices.map(v => new THREE.Vector3(v.x, v.y, 0));
        const geometry = new THREE.BufferGeometry().setFromPoints(points);
        const material = new THREE.LineBasicMaterial({ color: this.color });
        return new THREE.Line(geometry, material);
    }

    copy() {
        return new Polygon(this.vertices.map(v => ({ ...v })), this.filled);
    }

    setPosition(x, y) {
        if (this.mesh) {
            this.mesh.position.set(x, y, 0);
        }
        if (this.line) {
            this.line.position.set(x, y, 0);
        }
    }

    remove() {
        if (this.mesh) {
            scene.remove(this.mesh);

        }
        if (this.line) {
            scene.remove(this.line);
        }
    }
}

function init() {
    scene = new THREE.Scene();
    camera = new THREE.PerspectiveCamera(
        75,
        window.innerWidth / window.innerHeight,
        0.1,
        1000
    );
    camera.position.set(0, 0, 10);

    renderer = new THREE.WebGLRenderer({ antialias: true });
    renderer.setSize(window.innerWidth, window.innerHeight);
    document.getElementById("canvas-container").appendChild(renderer.domElement);

    raycaster = new THREE.Raycaster();
    mouse = new THREE.Vector2();

    createGround();
    createGridHelper(); // Create grid helper for background grid

    window.addEventListener("resize", onWindowResize);
    window.addEventListener("click", onMouseClick);

    animate();
}

function createGround() {
    const geometry = new THREE.PlaneGeometry(10, 10);
    const material = new THREE.MeshBasicMaterial({
        color: 0xffffff,
        side: THREE.DoubleSide,
    });
    ground = new THREE.Mesh(geometry, material);
    scene.add(ground);
}

function createGridHelper() {
    gridHelper = new THREE.GridHelper(10, 10);
    scene.add(gridHelper);
}

function onMouseClick(event) {
    if (isCopying && copiedPolygon) {
        isCopying = false; // Stop moving the copied polygon with the cursor

        mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
        mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;

        raycaster.setFromCamera(mouse, camera);
        const intersects = raycaster.intersectObject(ground);

        if (intersects.length > 0) {
            const point = intersects[0].point;
            copiedPolygon.setPosition(point.x, point.y); // Place the copy at the clicked position
            polygonCopies.push(copiedPolygon); // Track this placed polygon
            copiedPolygon = null; // Reset copiedPolygon for future copying
        }
    } else {
        // Continue with creating new vertices for the polygon if not copying

        mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
        mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;

        raycaster.setFromCamera(mouse, camera);
        const intersects = raycaster.intersectObject(ground);
        if (intersects.length > 0) {
            const point = intersects[0].point;
            if (currentVertices.length < 20) {  // Change this number to set the new max limit
                currentVertices.push({ x: point.x, y: point.y });
                drawVertex(point);
            }
        }
    }
}


let vertexMeshes = []; // Track all vertex meshes

function drawVertex(point) {
    const geometry = new THREE.CircleGeometry(0.05, 32);
    const material = new THREE.MeshBasicMaterial({ color: 0xff0000 }); // Red color for vertices
    const vertex = new THREE.Mesh(geometry, material);
    vertex.position.set(point.x, point.y, 0);
    scene.add(vertex);
    vertexMeshes.push(vertex); // Add the vertex to the array
}

document.getElementById("complete-btn").addEventListener("click", () => {
    if (currentVertices.length > 1) {
        if (currentVertices.length > 3) {
            // Create a filled polygon
            currentPolygon = new Polygon(currentVertices, true);
        } else {
            // Create just the connected lines
            currentPolygon = new Polygon(currentVertices, false);
        }
        polygons.push(currentPolygon);
        currentVertices = [];
        // Remove the vertex dots after creating the shape
        vertexMeshes.forEach(vertex => scene.remove(vertex));
        vertexMeshes = [];
    }
});

document.getElementById("copy-btn").addEventListener("click", () => {
    if (currentPolygon) {
        copiedPolygon = currentPolygon.copy(); // Create a copy of the current polygon
        isCopying = true; // Indicate that we are copying
    }
});

document.getElementById("reset-btn").addEventListener("click", () => {
    // Remove all polygons and vertices
    polygons.forEach(p => p.remove());
    vertexMeshes.forEach(vertex => scene.remove(vertex)); // Remove all vertex dots
    vertexMeshes = [];
    polygons = [];
    currentPolygon = null;
    currentVertices = [];
    if (copiedPolygon) {
        copiedPolygon.remove();
        copiedPolygon = null;
    }
    polygonCopies.forEach(p => p.remove());
    polygonCopies = [];

});

function animate() {
    requestAnimationFrame(animate);

    if (isCopying && copiedPolygon) {
        mouse.x = (event.clientX / window.innerWidth) * 2 - 1;
        mouse.y = -(event.clientY / window.innerHeight) * 2 + 1;
```

```javascript
        raycaster.setFromCamera(mouse, camera);
        const intersects = raycaster.intersectObject(ground);

        if (intersects.length > 0) {
            const point = intersects[0].point;
            copiedPolygon.setPosition(point.x, point.y); // Move the copy with the cursor
        }
    }

    renderer.render(scene, camera);
}

function onWindowResize() {
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize(window.innerWidth, window.innerHeight);
}

init();
```

Source-code link :- **https://github.com/Rohith1905/Connect-dots**