Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Computer Graphics(COMP 342) – Lab 4

Submitted To:

Dhiraj Shrestha Sir

Department of Computer Science and Engineering

Submitted By:

Mission Shrestha (54)

CE-2019 Year/Sem: 3rd Year/2nd Sem

Submission Date: 3rd May, 2023

Implementation of:

2D Translation, 2D Rotation, 2D Scaling, 2D Reflection, 2D Shearing

1. Transformations of 2D shapes

Source Code:

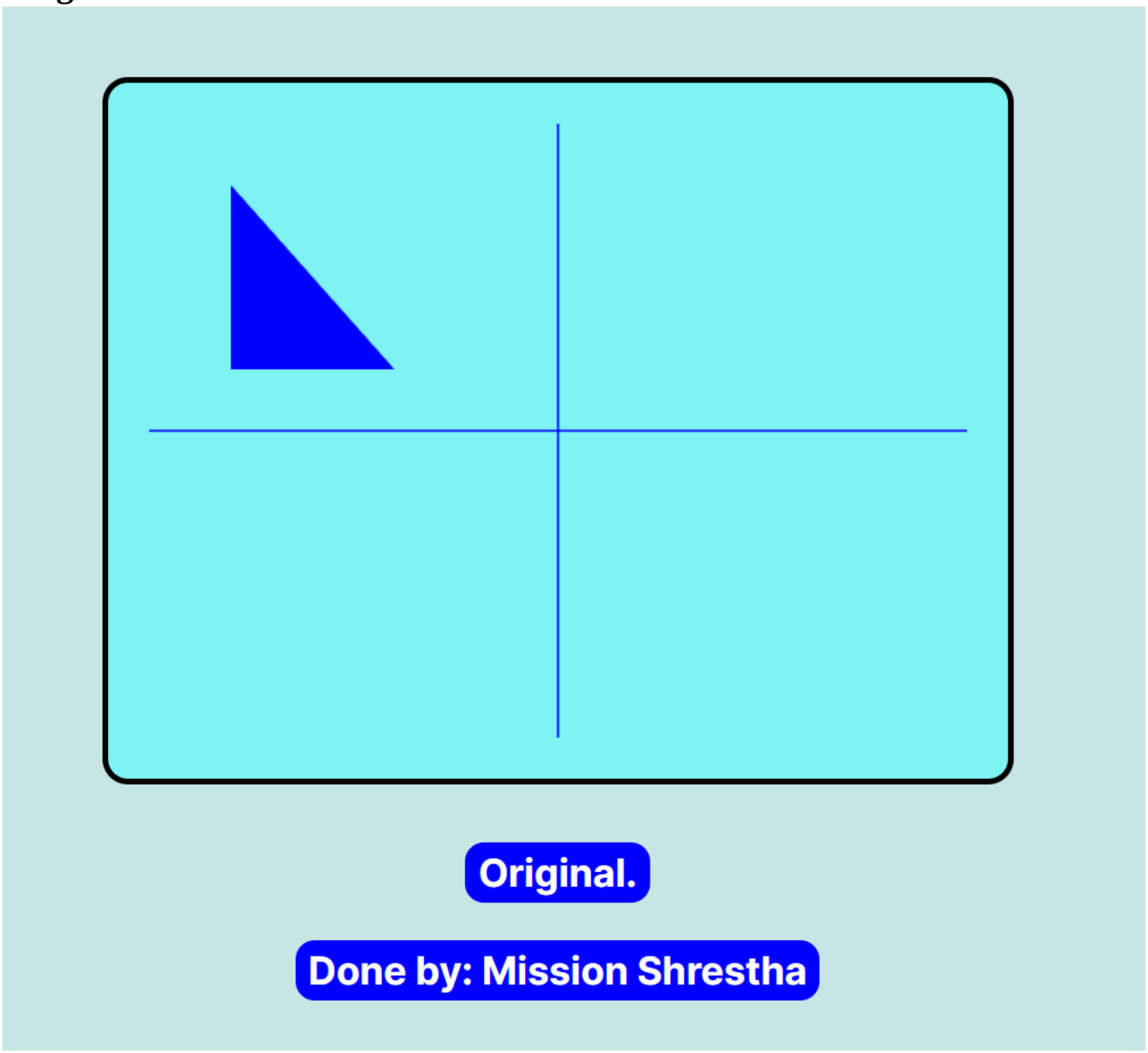
```
const canvas = document.querySelector("#glcanvas");

// Initialize the GL context
const gl = canvas.getContext("webgl");

// Only continue if WebGL is available and working
if (gl === null) {
    alert(
        "Unable to initialize WebGL. Your browser or machine may not support it."
    );
}


// For reflection
const reflectionMatrixAboutYaxis = [
    -1, 0, 0,
    0, 1, 0,
    0, 0, 1
];
const reflectionMatrixAboutXaxis = [
    1, 0, 0,
    0, -1, 0,
    0, 0, 1
];

// For rotation
const angle = -10;
const radianAngle = angle * Math.PI / 180;
const rotationMatrix = [
    Math.cos(radianAngle), -Math.sin(radianAngle), 0,
    Math.sin(radianAngle), Math.cos(radianAngle), 0,
    0, 0, 1,
]

// For scaling
const scalingMatrix = [
    0.4, 0, 0,
    0, 0.4, 0,
    0, 0, 0.4
];

// For translation
const translationMatrix = [
    1, 0, 0.6,
    0, 1, -0.4,
    0, 0, 1
];

// For shearing
const shearingMatrix = [
    1, -0.2, 0,
    0, 1, 0,
    0, 0, 1
];

function draw(vertexData, drawArraysMode, fragmentShaderGLSL = '') {
    const vertexDatas = [
        ...vertexData,
```

```
    ];

    const buffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexDatas), gl.STATIC_DRAW);

    const vertexShader = gl.createShader(gl.VERTEX_SHADER);
    gl.shaderSource(
        vertexShader,
        `
      attribute vec3 position;
      void main() {
          gl_Position = vec4(position, 1);
          gl_PointSize = 4.0;
      }
  `
    );
    gl.compileShader(vertexShader);

    const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
    if (fragmentShaderGLSL == '') {
        gl.shaderSource(fragmentShader,
            `void main(){
        gl_FragColor = vec4(0, 0, 1, 1);
    }`
        );
    } else {
        gl.shaderSource(fragmentShader, fragmentShaderGLSL);
    }

    gl.compileShader(fragmentShader);

    const program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);

    gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexDatas), gl.STATIC_DRAW);

    const positionLocation = gl.getAttribLocation(program, `position`);
    gl.enableVertexAttribArray(positionLocation);
    gl.vertexAttribPointer(positionLocation, 3, gl.FLOAT, false, 0, 0);
    gl.useProgram(program);

    if (drawArraysMode == 'line') {
        gl.drawArrays(gl.LINES, 0, vertexDatas.length);
    }
    else if (drawArraysMode == 'triangle') {
        gl.drawArrays(gl.TRIANGLES, 0, vertexDatas.length);
    }
}

function displayAxis() {
    let tempVertexData = [];
    tempVertexData.push(1, 0, 0, -1, 0, 0, 0, 1, 0, 0, -1, 0);
    draw(tempVertexData, 'line');
}

// const triangleData = [
//     0.0, 0, 1,
//     0, 1, 1,
//     1, 0, 1,
// ];
const triangleData = [
    -0.8, 0.8, 1,
```

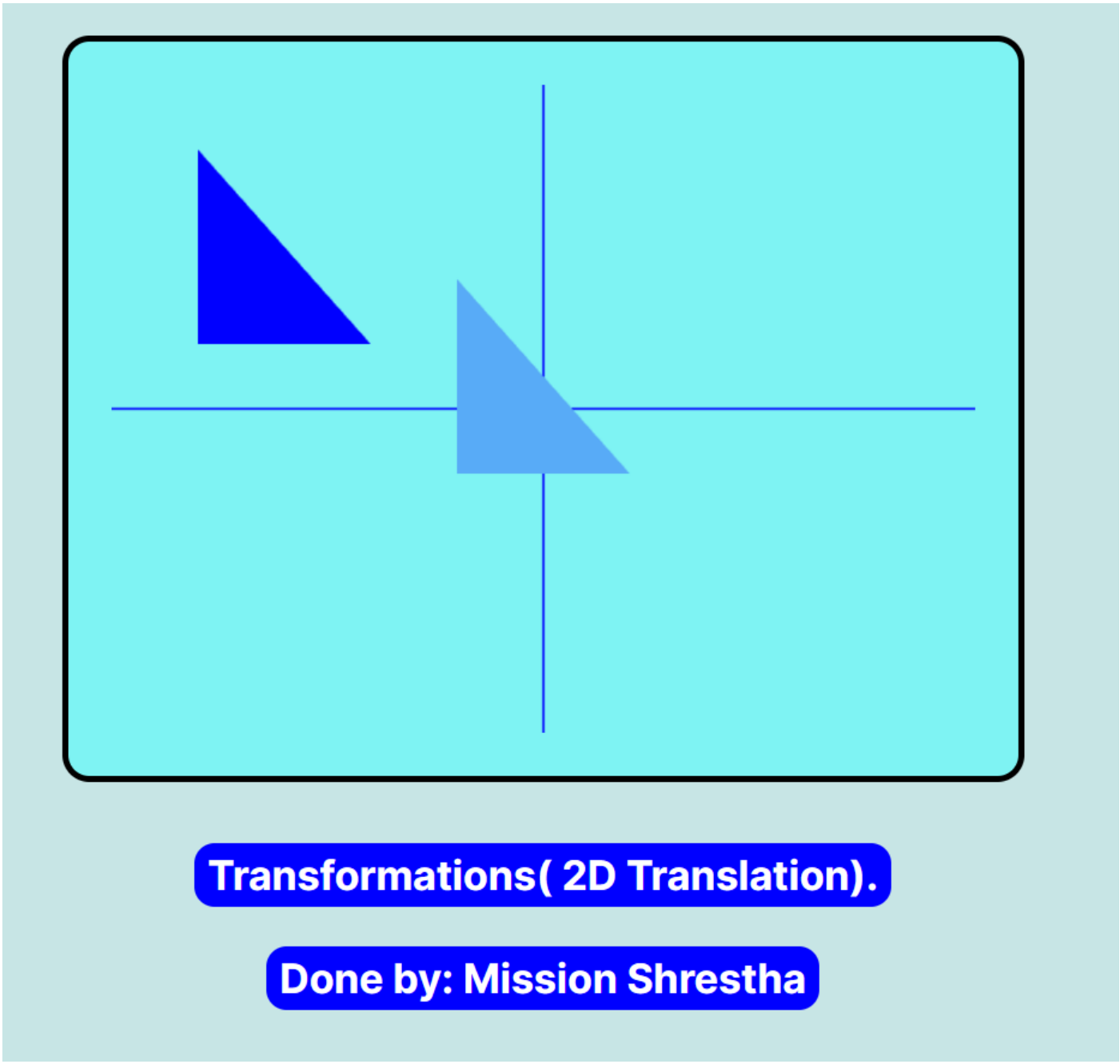
```
    -0.8, 0.2, 1,
    -0.4, 0.2, 1,
];

function drawInitialTriangle() {
    let vertexData = [
        ...triangleData,
    ];
    draw(vertexData, 'triangle');
}



function twoDTransformation(transformationMatrix, drawArrayMode, fragmentShaderGLSL) {
    let tempVertexData = [];
    tempVertexData.push(...matrixMultiplication(transformationMatrix, triangleData.slice(0, 3)));
    tempVertexData.push(...matrixMultiplication(transformationMatrix, triangleData.slice(3, 6)));
    tempVertexData.push(...matrixMultiplication(transformationMatrix, triangleData.slice(6, 9)));
    draw(tempVertexData, drawArrayMode, fragmentShaderGLSL);
}

function matrixMultiplication(transformerMatrix, vertices) {
    let result = [];
    let [a11, a12, a13, a21, a22, a23, a31, a32, a33] = transformerMatrix; // 3 * 3
    let [b1, b2, b3] = vertices; // 3*1
    let c1 = a11 * b1 + a12 * b2 + a13 * b3;
    let c2 = a21 * b1 + a22 * b2 + a23 * b3;
    let c3 = a31 * b1 + a32 * b2 + a33 * b3;
    result.push(...[c1, c2, c3]);
    return result;
}

displayAxis();
drawInitialTriangle();

// twoDTransformation(translationMatrix, 'triangle', `void main(){gl_FragColor = vec4(0, 0, 0.3, 0.3);}`);
// twoDTransformation(rotationMatrix, 'triangle', `void main(){gl_FragColor = vec4(0, 0, 0.3, 0.3);}`);
// twoDTransformation(scalingMatrix, 'triangle', `void main(){gl_FragColor = vec4(0, 0, 0.3, 0.3);}`);
// twoDTransformation(reflectionMatrixAboutYaxis, 'triangle', `void main(){gl_FragColor = vec4(0, 0, 0.3, 0.3);}`);
// twoDTransformation(reflectionMatrixAboutXaxis, 'triangle', `void main(){gl_FragColor = vec4(0, 0, 0.3, 0.3);}`);
// twoDTransformation(shearingMatrix, 'triangle', `void main(){gl_FragColor = vec4(0, 0, 0.3, 0.3);}`);
```

Output:

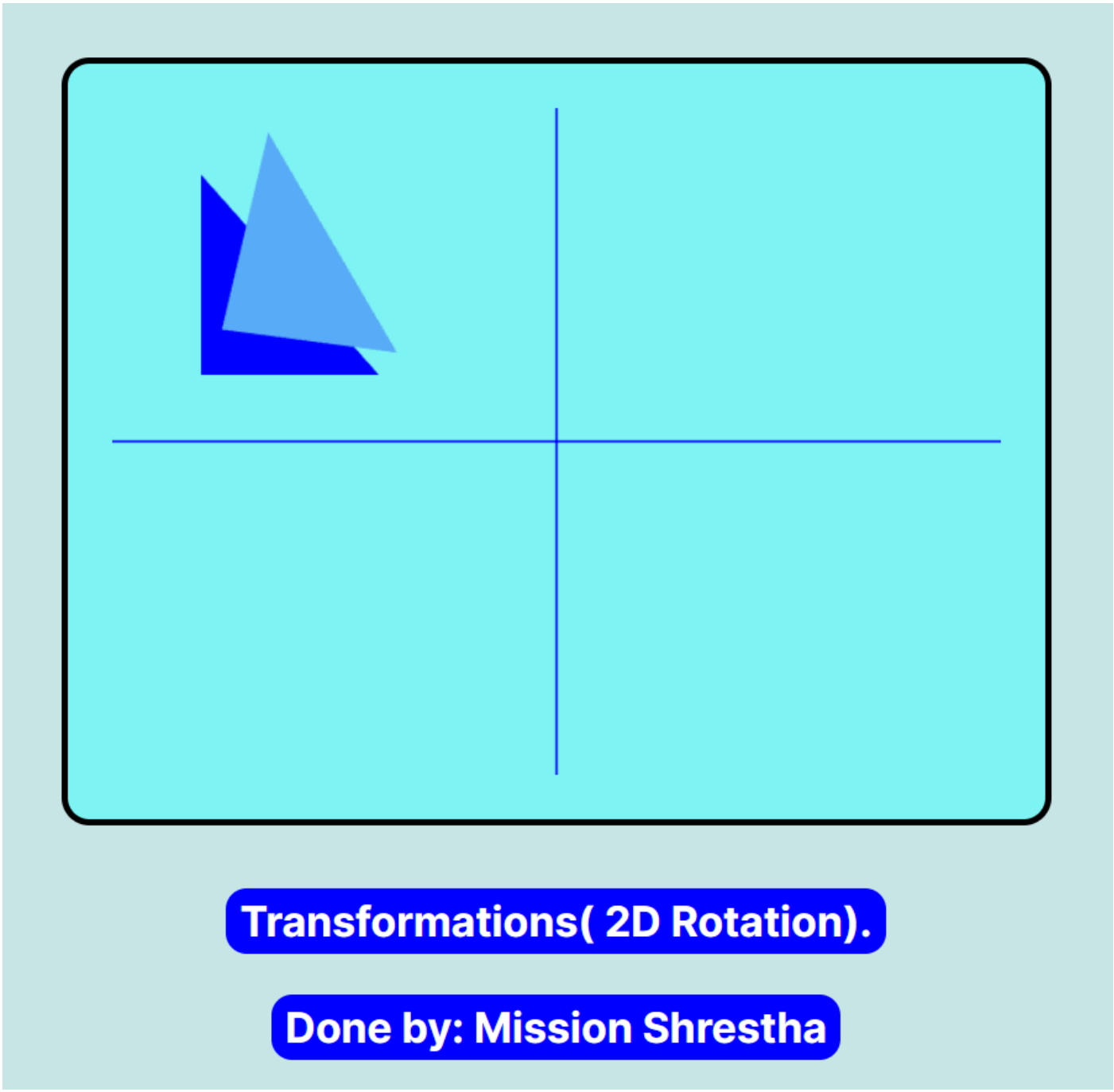
Original:



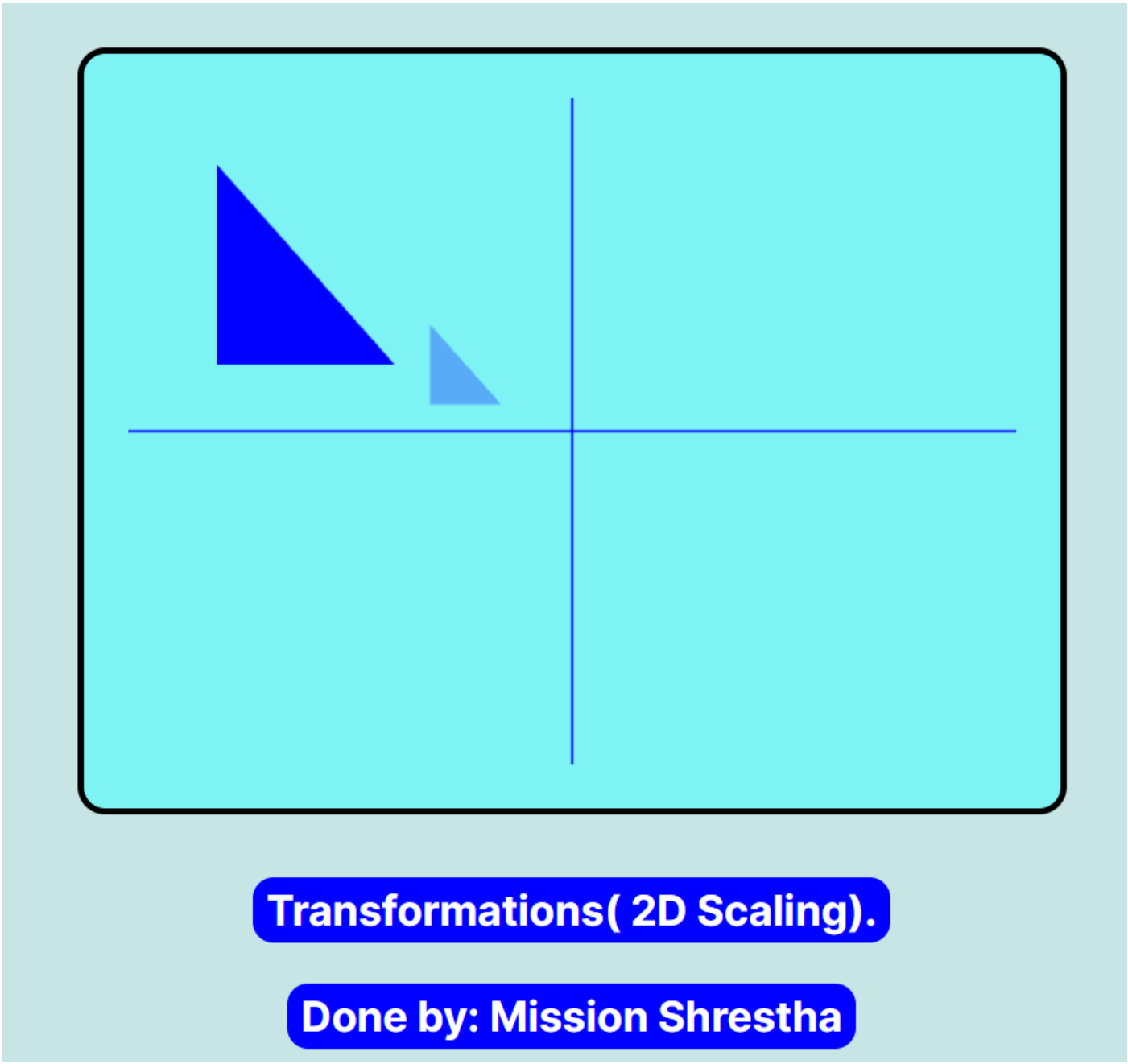
a. 2D Translation



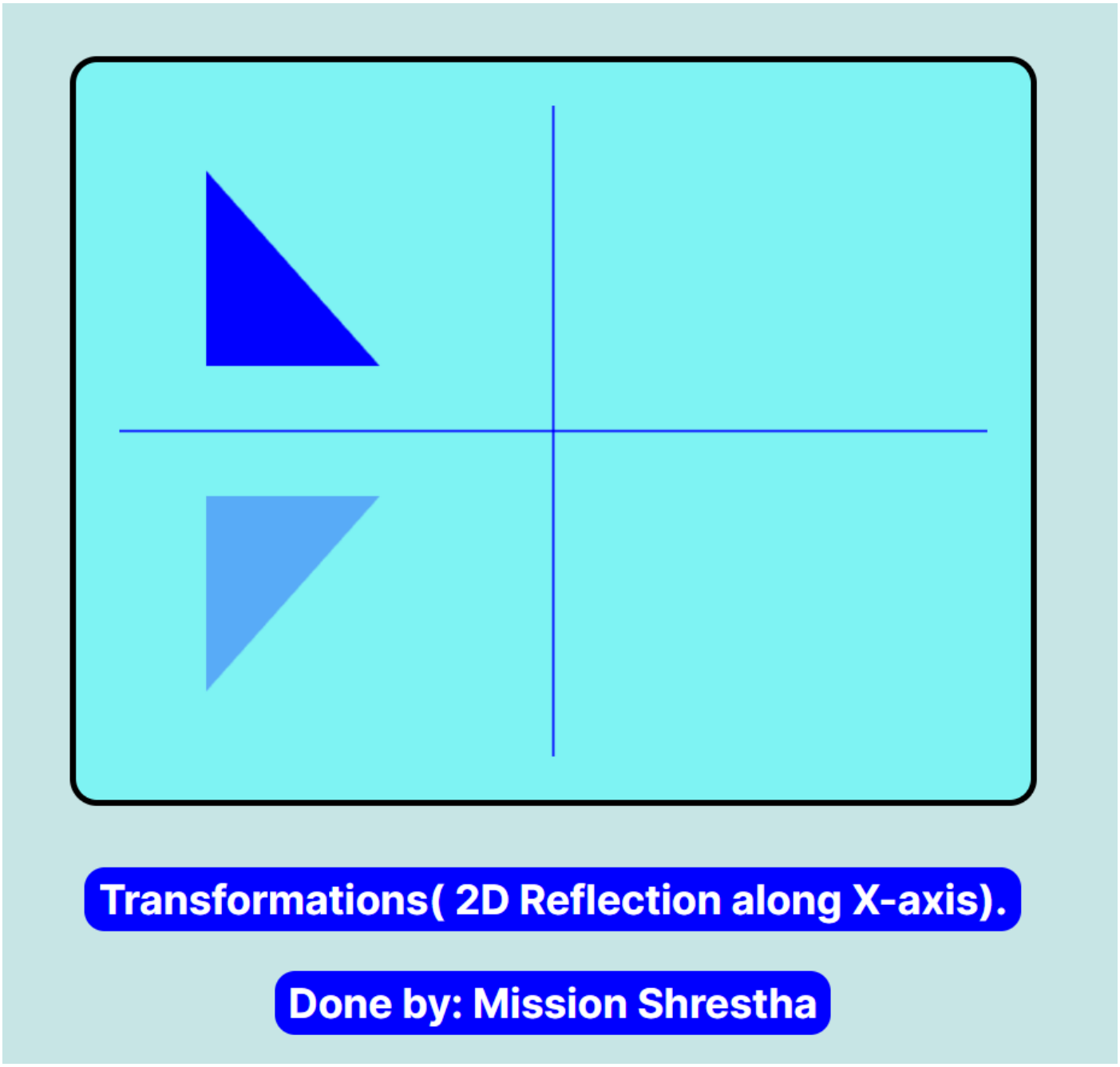
b. 2D Rotation



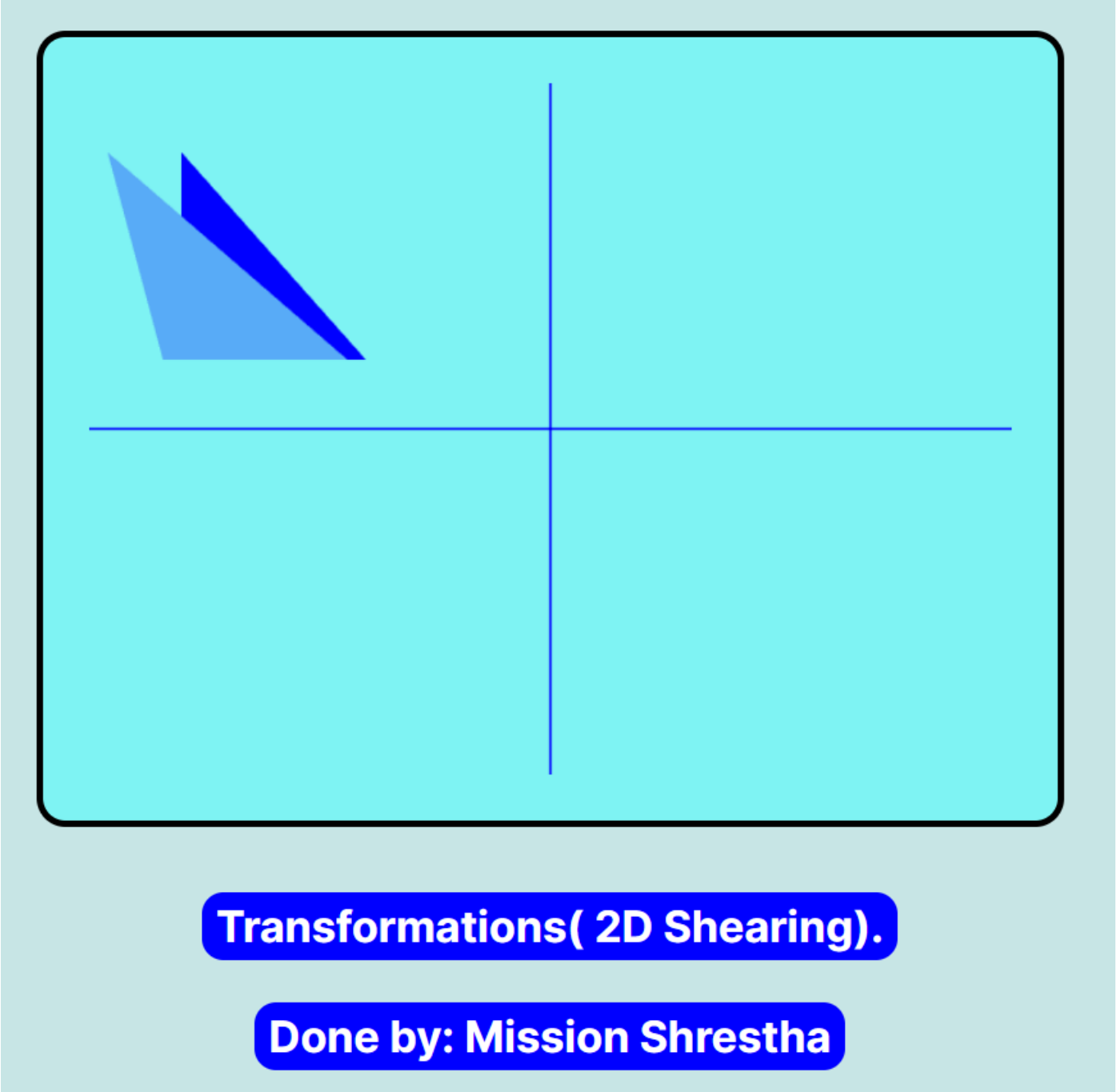
c. 2D Scaling



d. 2D Reflection



e. 2D Shearing



Conclusion:

Hence, Implementation of Transformation (2D Translation, 2D Rotation, 2D Scaling, 2D Reflection, 2D Shearing) on a 2D object (Triangle) was performed