KATHMANDU UNIVERSITY

DHULIKHEL, KAVRE

COMP-342

Lab Report:”4”

Date: 2023/05/03

Submitted By:

Saugat Poudel

Group: CE

Roll No: 63

Submitted To:

Dr Dhiraj Shrestha

Department of Computer and Science and Engineering

LAB -4

1. **Write a Program to implement:**
   - a. **2D Translation**
   - b. **2D Rotation**
   - c. **2D Scaling**
   - d. **2D Reflection**
   - e. **2D Shearing**

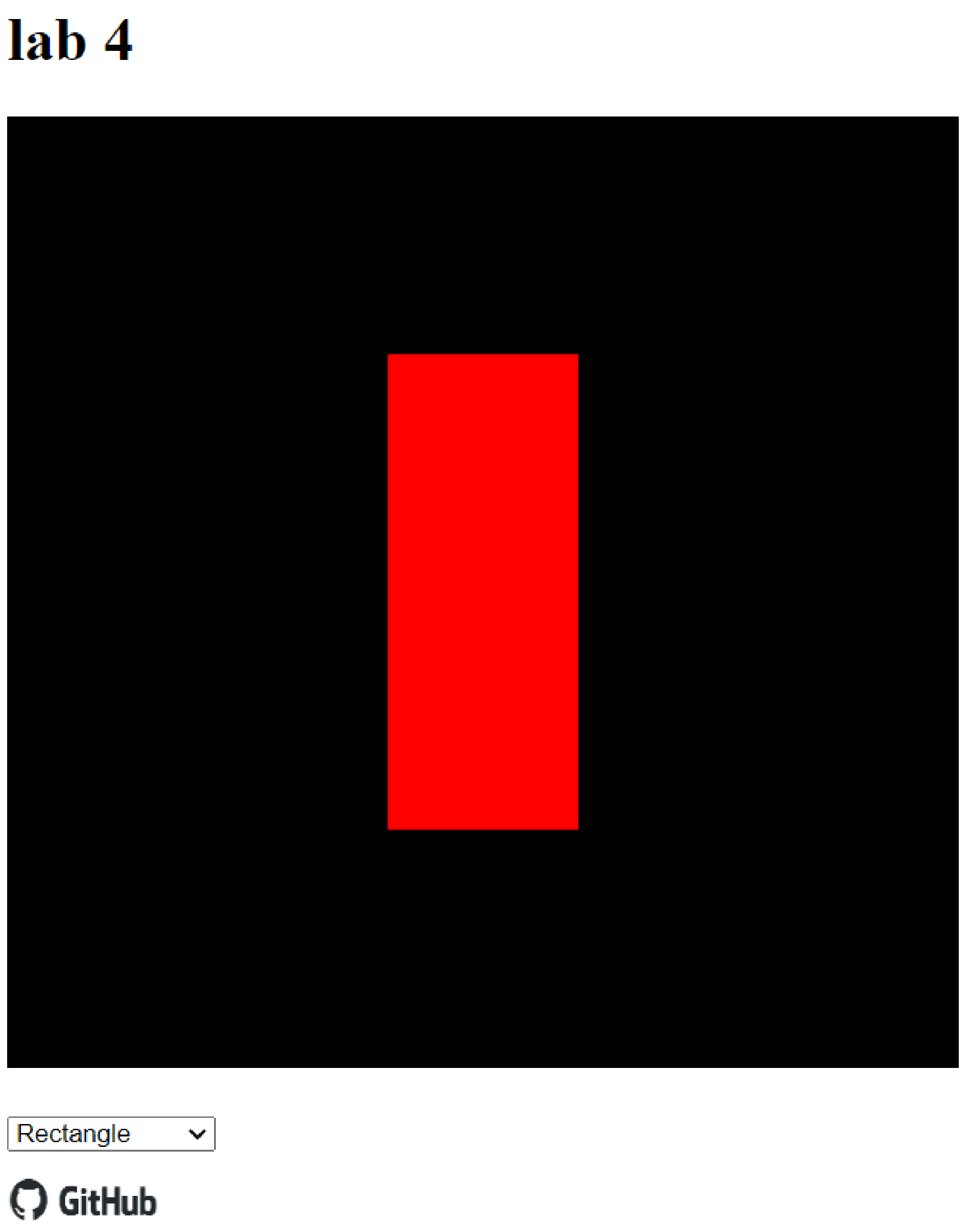
GlL setup to draw rectangle

```
function createrectangle(vertices,fragCode = `
  void main(void) {
    gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
  }
`){
    var vertexBuffer = gl.createBuffer();
  // Bind the buffer to ARRAY_BUFFER
  gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);

  // Copy the vertices data to the buffer
  gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);

// Create the fragment shader
var fragShader = gl.createShader(gl.FRAGMENT_SHADER);

// Define the fragment shader code


// Attach the code to the shader
gl.shaderSource(fragShader, fragCode);

// Compile the shader
gl.compileShader(fragShader);

// Create the shader program
var shaderProgram = gl.createProgram();

// Attach the vertex shader and fragment shader to the program
gl.attachShader(shaderProgram, vertShader);
gl.attachShader(shaderProgram, fragShader);

// Link the program
gl.linkProgram(shaderProgram);

// Use the program
gl.useProgram(shaderProgram);

// Enable the vertex attribute array
var coord = gl.getAttribLocation(shaderProgram, "coordinates");
gl.enableVertexAttribArray(coord);
// Specify how to read the buffer data
gl.vertexAttribPointer(coord, 2, gl.FLOAT, false, 0, 0);

// Draw the square
gl.drawArrays(gl.TRIANGLE_FAN, 0, 4)
}
```

Main.js

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

const canvasHeight = canvas.height;
const canvasWidth = canvas.width;
const gl = canvas.getContext("webgl");

if (!gl) {
  throw new Error(
    "Unable to load WebGL. Your computer or browser maynot support it"
  );
}

var vertexBuffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, null);

var vertCode =
  "attribute vec3 coordinates;" +
  "void main(void)" +
  "{" +
  "gl_Position = vec4(coordinates, 1.0);" +
  "gl_PointSize = 10.0;" +
  "}";

var vertShader = gl.createShader(gl.VERTEX_SHADER);

gl.shaderSource(vertShader, vertCode);

gl.compileShader(vertShader);

// gl.clearColor()
// gl.clear()
```

Initial rectangle



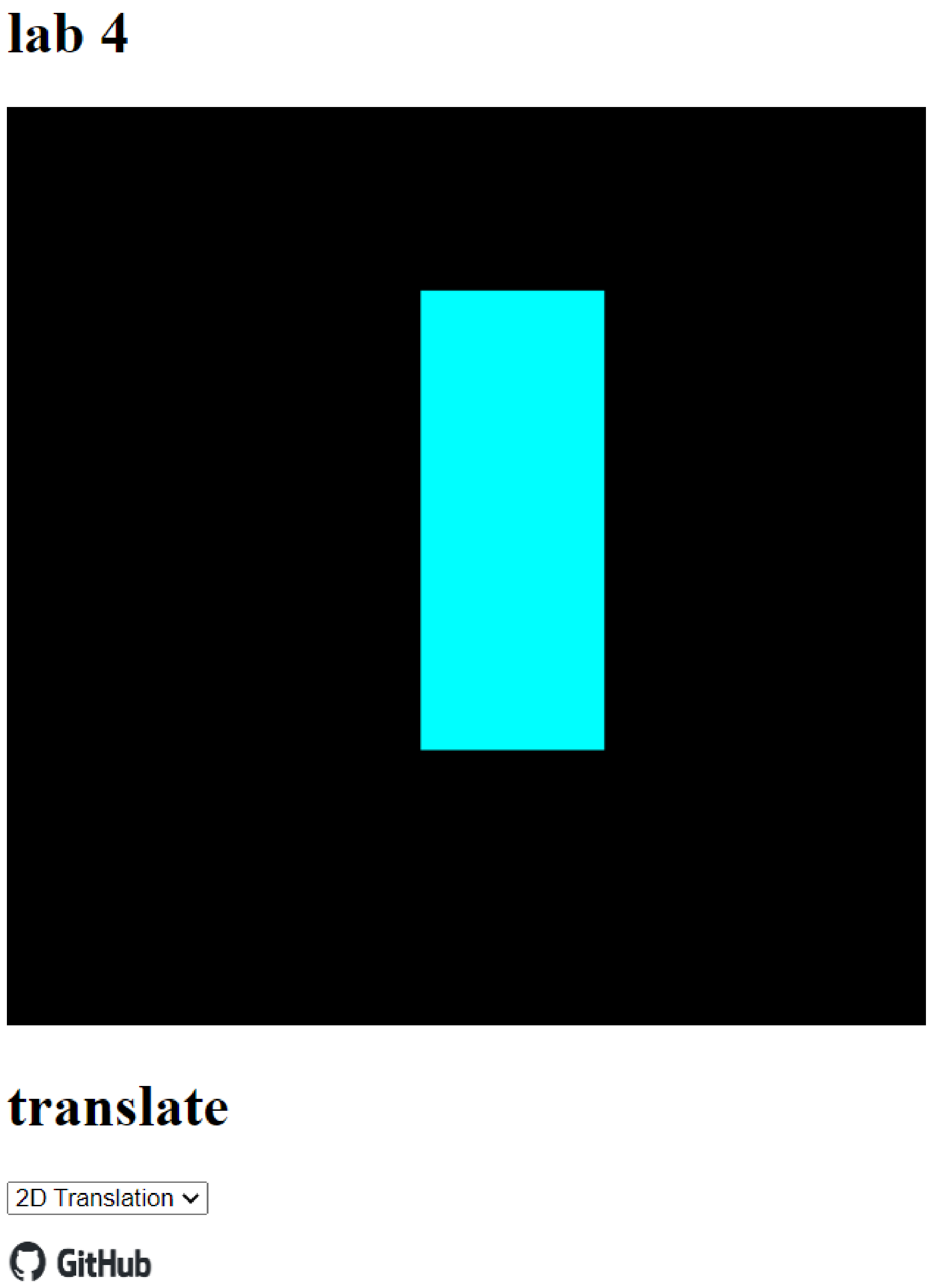
lab 4

a) 2D Translation

```
var initial_vertices = new Float32Array([

    -0.2, -0.5,  // Bottom-left corner
    +0.2, -0.5,  // Bottom-right corner
    +0.2, +0.5,  // Top-right corner
    -0.2, +0.5   // Top-left corner
]);
```

```
function translaterectangle(vertice,translation) {
    let translated_Vertices = [];

    for (let i = 0; i < vertice.length; i = i + 2) {
      translated_Vertices.push(vertice[i] + translation[0]);
      translated_Vertices.push(vertice[i + 1] + translation[1]);
    }
    return translated_Vertices;
  }

  var translatedVertices = translaterectangle(initial_vertices, [0.1, 0.1]);
  tarr=new Float32Array(translatedVertices)
```

Output



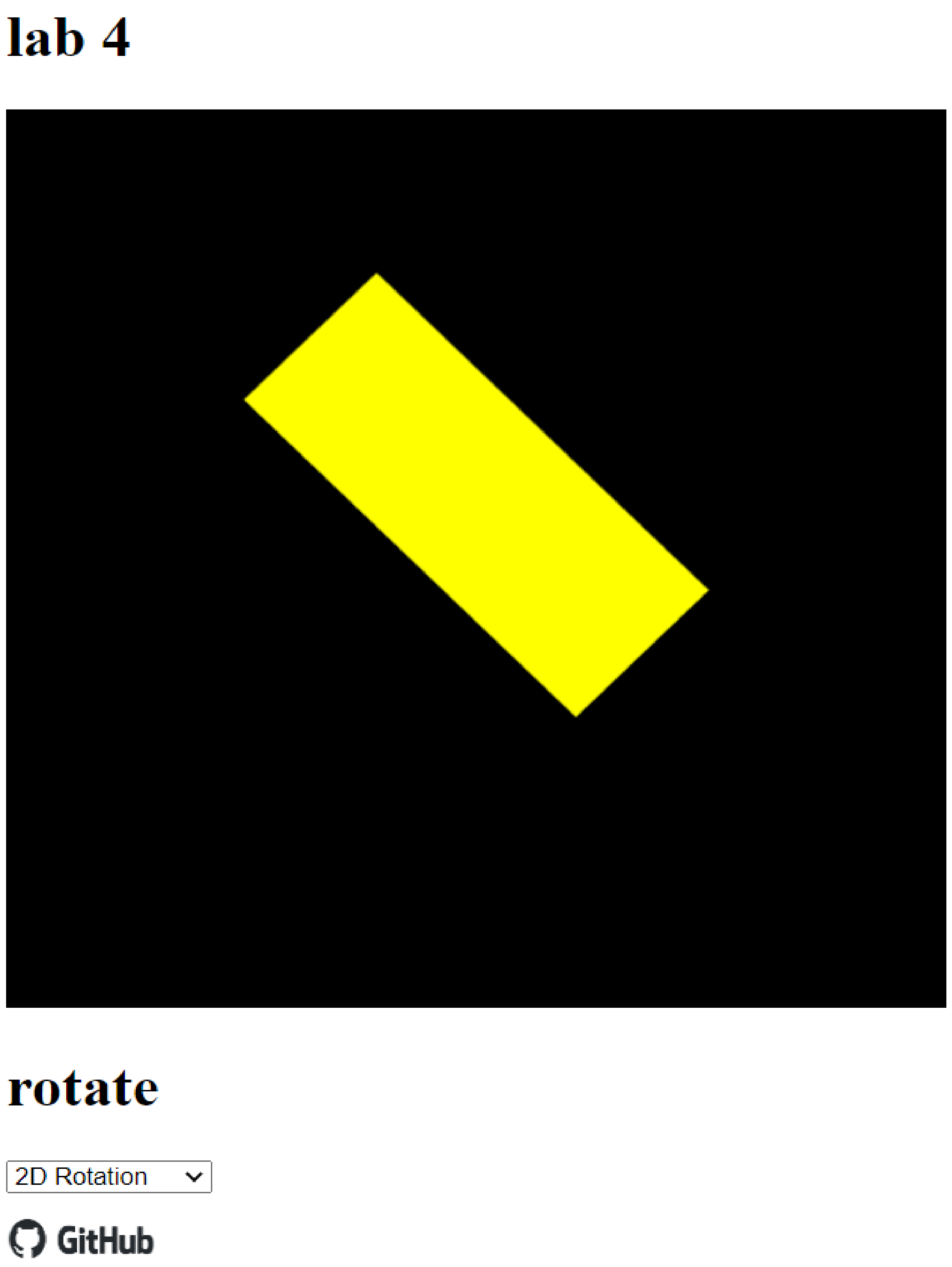
lab 4

translate

b)2D Rotation

```
 //rotation

function rotateRectangle(vertice, angle) {
    let rotated_Vertices = [];

    for (let i = 0; i < vertice.length; i = i + 2) {
      let x = vertice[i];
      let y = vertice[i + 1];

      // Rotate the point (x, y) around the origin (0, 0)
      let rotatedX = x * Math.cos(angle) - y * Math.sin(angle);
      let rotatedY = x * Math.sin(angle) + y * Math.cos(angle);

      rotated_Vertices.push(rotatedX);
      rotated_Vertices.push(rotatedY);
    }
    return rotated_Vertices;
  }
  var rotatedVertices = rotateRectangle(tarr, Math.PI / 4);
  var rotatedArr = new Float32Array(rotatedVertices)
```

Output:



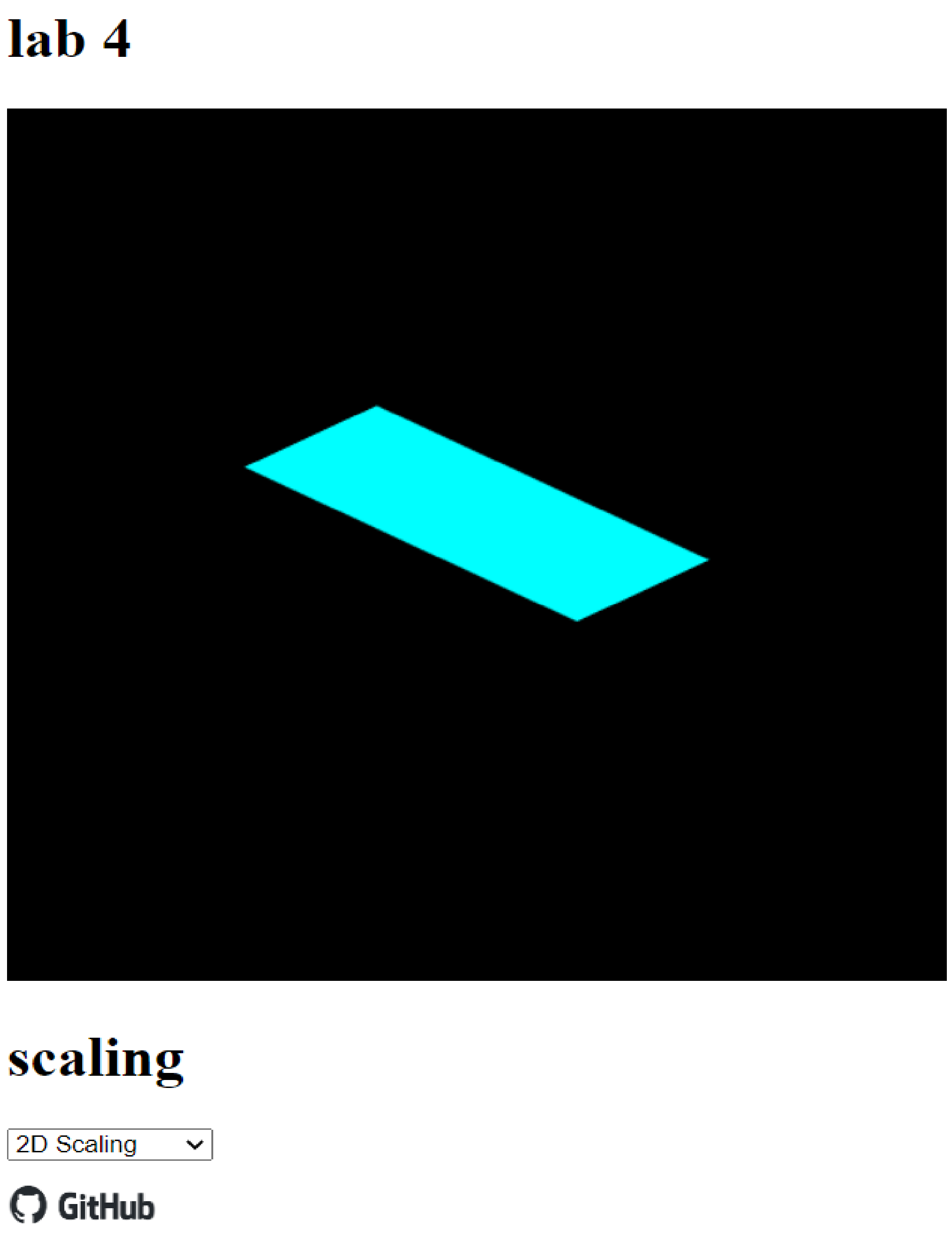
lab 4

rotate

c) 2D Scaling

```
 function scalerectangle(vertices, scale) {

  let scaledVertices = [];

  for (let i = 0; i < vertices.length; i = i + 2) {
    scaledVertices.push(vertices[i] * scale[0]);
    scaledVertices.push(vertices[i + 1] * scale[1]);
  }

  return new Float32Array(scaledVertices);
}

var scaledVertices = scalerectangle(rotatedArr, [1, 0.5]);
scaled = new Float32Array(scaledVertices);
```

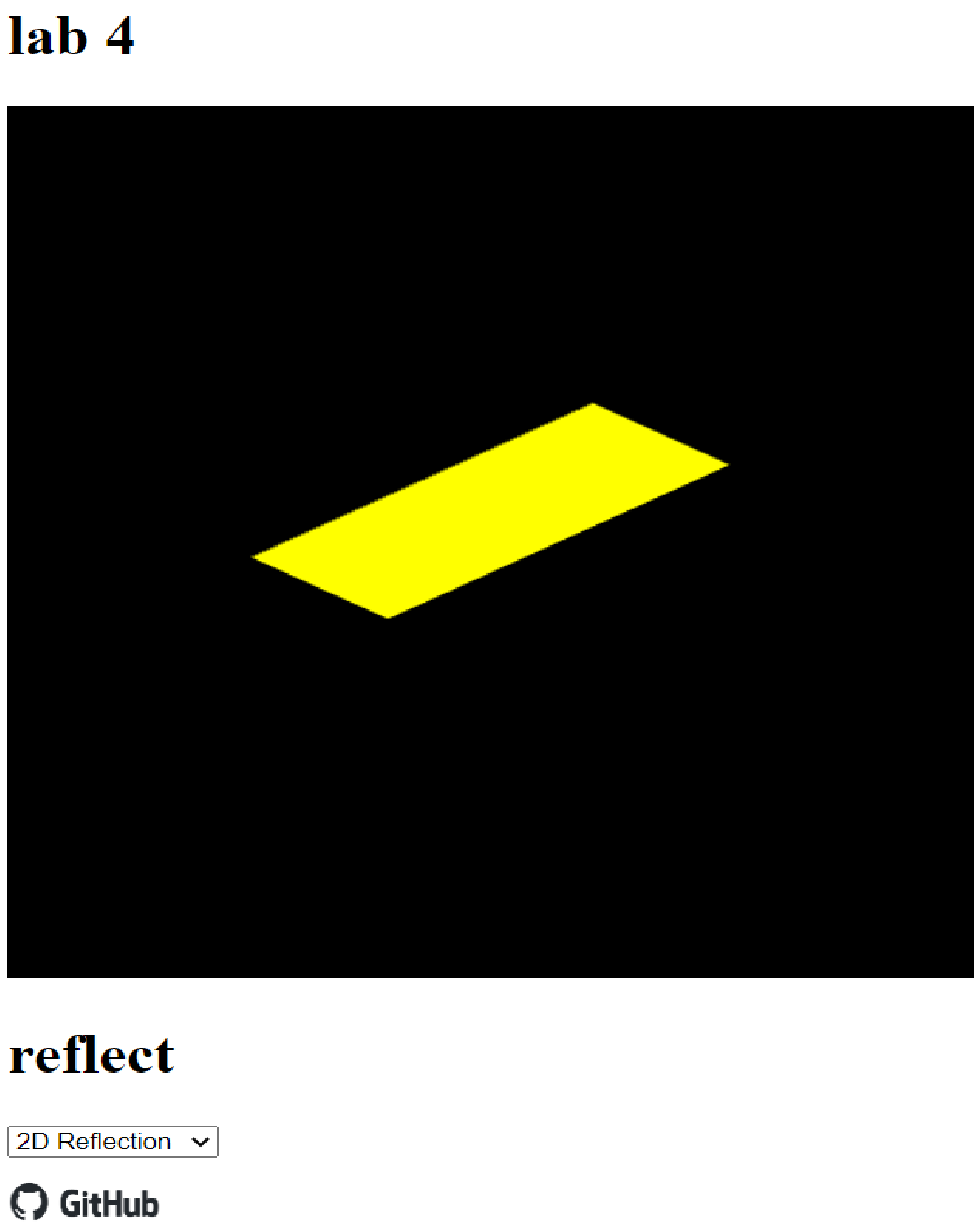
Output:



d) 2D Reflection

```
function reflectrectangle(vertices) {

   let reflectedVertices = [];

   for (let i = 0; i < vertices.length; i = i + 2) {
     reflectedVertices.push(vertices[i] * -1);  // flip x-coordinate
     reflectedVertices.push(vertices[i + 1]);   // keep y-coordinate
   }

   return new Float32Array(reflectedVertices);
 }

 var reflectedVertices = reflectrectangle(scaled);
```

Output:



lab 4

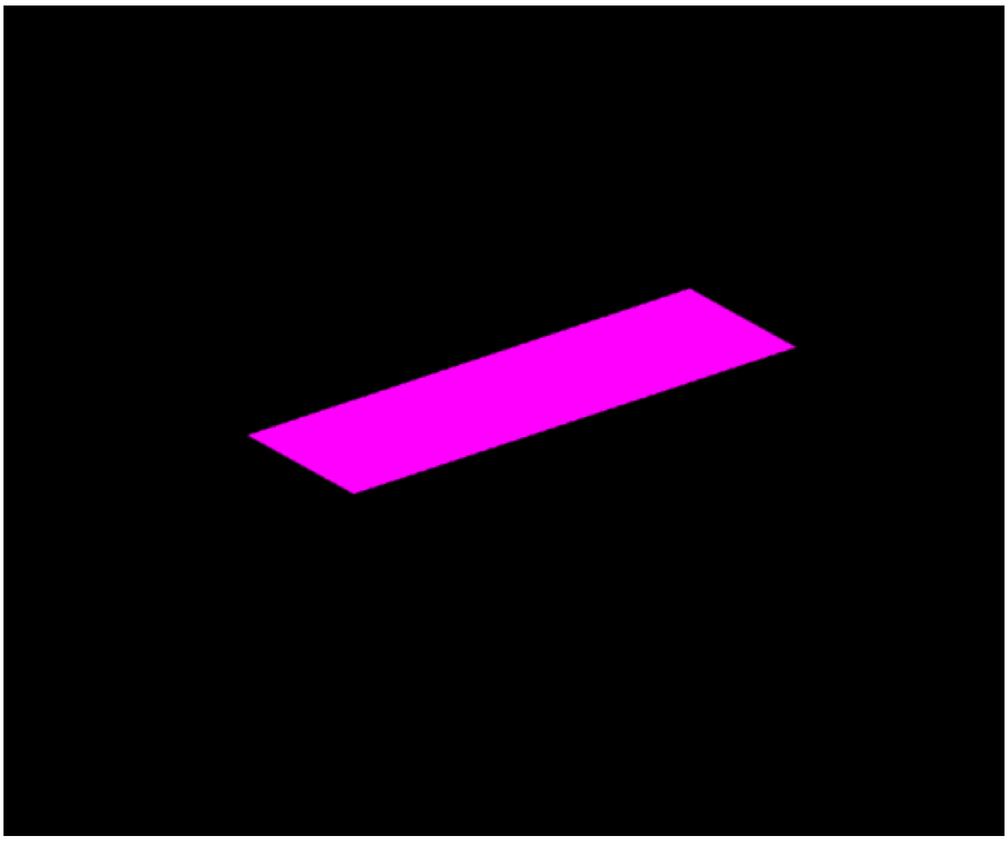
reflect

e) 2D Shearing

```
function shearrectangleX(vertices, shear) {
  let shearedVertices = [];

  for (let i = 0; i < vertices.length; i = i + 2) {
    shearedVertices.push(vertices[i] + vertices[i + 1] * shear[0]);
    shearedVertices.push(vertices[i + 1]);
  }

  return new Float32Array(shearedVertices);
}

var shearedVertices = shearrectangleX(reflectedVertices, [0.5]);
sheared = new Float32Array(shearedVertices);
```

Output:

lab 4



shearing

2D shearing

GitHub

Conclusion:
Hence, an initial rectangle was drawn with the initial vertices and accordingly it was reflected, rotated, translated, scaled and sheared. All these transformation was performed using WebGL library.