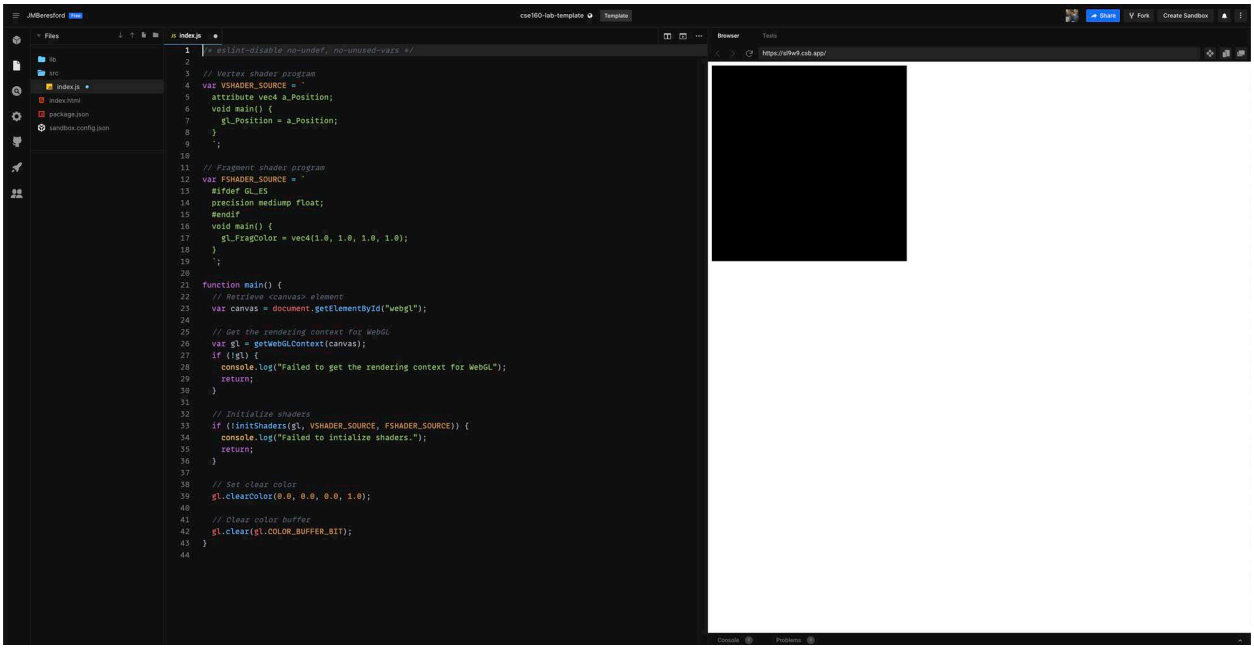
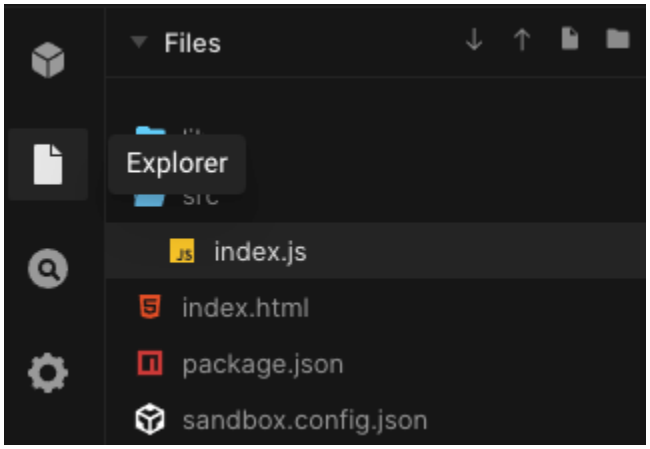
Lab Activity 1

The Lab Environment

To start the programming lab go to: https://codesandbox.io/s/cse160-lab-template-dmj4l

You should be met with some code and a black canvas in a webpage:





Notice the file explorer in the left tab menu:

If not already open, navigate to the `src/index.js` file.

You are able to modify the code in the editor and see the changes on the preview once you save the file; let's verify this by changing the canvas color from black to dark gray:

```
// Set clear color
// gl.clearColor(0.0, 0.0, 0.0, 1.0);
gl.clearColor(0.2, 0.2, 0.2, 1.0);
```

Change this line and save the file to see the canvas color change.

The Goal

In this week's lab section we are going to draw a picture with some triangles. Specifically, we are going to draw this:

To do this we will take a single right isosceles triangle, and repeatedly draw it with transformation matrices that move, rotate and scale it across the canvas.

The book has a good example of getting just one triangle on the screen using a transformation matrix (code listing 4.2 on page 122)

The Process

1. Setting Up - This section should look familiar from your last assignment.

    a. Define your triangle's vertices:

    ```
    const vertices = new Float32Array([v1.x, v1.y, v2.x, v2.y, v3.x, v3.y]);
    ```

    (**replace the v1,v2,v3 placeholders with actual floating point values** that you want your triangle to use, e.g. [`-0.5, -0.5, 0.5, -0.5, -0.5, 0.5`])

    b. Create a buffer to store the vertex data:

    ```
    const vertexBuffer = gl.createBuffer();
    if (!vertexBuffer) {
      console.log("Failed to create the buffer object");
    }
    ```

    c. Bind the buffer to your webGL program and write the vertex data into it:

    ```
    // Bind the buffer object to target
    gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
    // Write data into the buffer object
    gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
    ```

    d. Grab the location of the position attribute variable in your vertex shader program:

    ```
    const aPosPtr = gl.getAttribLocation(gl.program, "aPosition");

    if (aPosPtr < 0) {
      console.error("Could not find aPosition ptr");
    }
    ```

    e. Setup the attribute parameters and enable it:

    ```
    gl.vertexAttribPointer(aPosPtr, 2, gl.FLOAT, false, 0, 0);
    gl.enableVertexAttribArray(aPosPtr);
    ```

    f. Create a matrix to be used later on in our function:

    ```
    const M = new Matrix4();
    ```

2. A Single Triangle using a Matrix - You haven't used the Matrix4 class in js, nor the mat4 type in GLSL before, since we did use matrices in Assignment1.

    a. Let's create a function that draws a spaceship: This function will take in our webGL context and a matrix to use for transformations.

    ```
    function drawSpaceship(gl, matrix)
    ```

    b. In this function we are going to modify our transformation matrix, so we need to grab the location of that from the vertex shader:

    ```
    function drawSpaceship(gl, matrix) {
      const uModelMatrixPtr = gl.getUniformLocation(gl.program, "uModelMatrix");
    ```

        i. We also need to make sure our vertex shader has this matrix defined, and is using it so change this at the top of the file:

        ```
        // Vertex shader program
        const VSHADER_SOURCE = `
          attribute vec2 aPosition;
          uniform mat4 uModelMatrix;
          void main() {
            gl_Position = uModelMatrix * vec4(aPosition, 0.0, 1.0);
          }
          `;
        ```

    c. Let's create another matrix that we can safely modify **inside our function**:

    ```
    const M1 = new Matrix4();
    ```

    d. Before we get crazy, let's make sure we haven't made a mistake yet and get a triangle on screen.

        i. Draw a triangle using our matrix which is currently an identity matrix (**do this inside the drawSpaceship function**):

        ```
        gl.uniformMatrix4fv(uModelMatrixPtr, false, M1.elements);
        gl.drawArrays(gl.TRIANGLES, 0, 3);
        ```

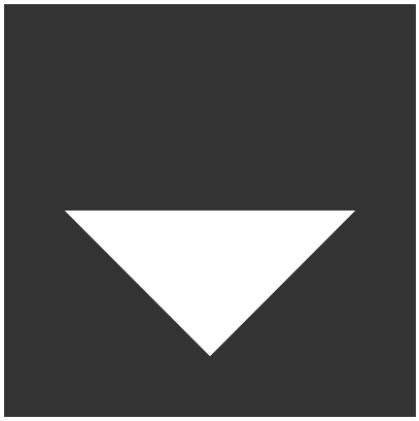
ii. You should get something like this, depending on how you defined your vertices:

3. The Spaceship - Same triangle, drawn with different matrices

   a. Now we need to draw more triangles to create the spaceship. To do this we will want to update our transformation matrix and redraw our triangle. Try rotating your triangle 45 degrees about the z-axis:

```
M1.rotate(45, 0, 0, 1);
gl.uniformMatrix4fv(uModelMatrixPtr, false, M1.elements);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

      i. You should see something like this (note this will be different if your first triangle was different than the one above):



b. The general way you will want to transform your triangle is as such:

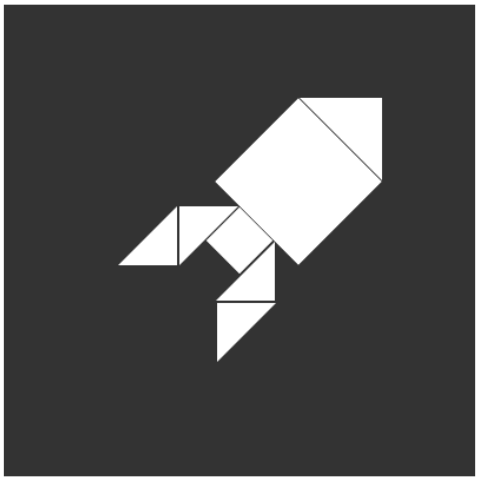
   i. Reset the matrix back to the original we sent into the function

   ii. Remember that in the matrix multiplication our operations are performed right to left, so if we want to scale and rotate *before* we translate (which we do), we want something like this:

```
M1.set(matrix);
M1.translate(0, 0, 0);
M1.rotate(45, 0, 0, 1);
M1.scale(1, 1, 1);
gl.uniformMatrix4fv(uModelMatrixPtr, false, M1.elements);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

   iii. Doing the above sequence should draw the same exact triangle as in our rotation step earlier. Note we can change the values of any of these operations now to reflect what transformation we want.

c. After positioning all of your triangles (9 by the reference image), you should have something like this:

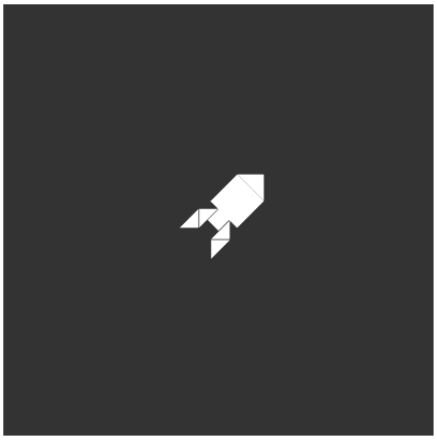


4. Many spaceships for the price of one

   a. Now that we have a function that draws a spaceship given a transformation matrix, we can draw whole spaceships all over our canvas with one matrix each!

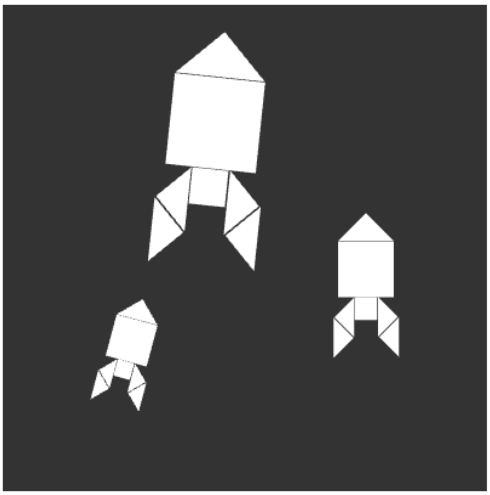
b. Try making your spaceship smaller by altering the matrix you passed into your `drawSpaceship()` function:

```
const M = new Matrix4();

M.setTranslate(0, 0, 0);
M.scale(0.35, 0.35, 0.35);
M.rotate(0, 0, 0, 1);

drawSpaceship(gl, M);
```



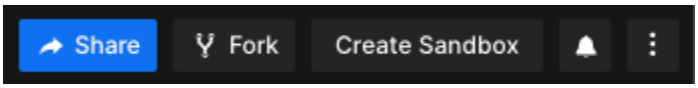
c. Now make a whole fleet of spaceships!

```
const M = new Matrix4();

M.setTranslate(-0.5, -0.5, 0);
M.scale(0.35, 0.35, 0.35);
M.rotate(30, 0, 0, 1);

drawSpaceship(gl, M);

M.setTranslate(0.5, -0.25, 0);
M.scale(0.46, 0.46, 0.46);
M.rotate(45, 0, 0, 1);

drawSpaceship(gl, M);

M.setTranslate(-0.15, 0.25, 0);
M.scale(0.75, 0.75, 0.75);
M.rotate(39, 0, 0, 1);

drawSpaceship(gl, M);
```



Submission

To submit this lab assignment, make sure you submit your ***own*** codesandbox link for the lab. This means that you will need to fork the sandbox that you and your group worked on together such that each individual has their own. The fork button is on the top right corner of the page:



Then, you will submit the following on canvas:

- Link to your sandbox (**click on the blue share button and click copy link at bottom**)
- List of groupmates names