Thought process for developement

- I recycled most of the libraries and components from previous assignments
- I made a few changes to the main program based on how I normally write modern js

Theory

1.
    - a.
        - OpenGL: FOSS, cross-platform, cross-language graphics library
        - OpenGL ES: Variant of OpenGL designed for embedded systems
        - WebGL: JavaScript api that interfaces with OpenGL es in browsers
    - b.
        - color:
        - depth:
        - stencil:
    - c.
        - `vertexAttrib1f`: takes a single `float` as an argument
        - `vertexAttrib4fv`: takes a vector containing 4 `floats` as arguments
    - d. between vertex and fragement shaders there is geometric shape assembly and rasterization
        - geometric shape assembly: geometric shape created from verticies transformed by vertex shader
        - fragment: pixel + depth
        - rasterization: geometric shape is rasterized (converted into fragments)
    - e. points, lines, triangles
    - f. setting a viewport specifies the size of the drawing buffer. It can be done like so:

```
gl.viewport(0,0, gl.viewportWidth, gl.viewportHeight);
```

    you can also do an orthographic transformations and translation+scale if you would prefer to use a different coordinate system, but this isn't needed if you're ok with the [-1, 1] scale

2.
    - a.
        - attribute: different for each vertex
        - uniform:
            - vertex: same for all verticies
            - fragment: global data passed from js program
        - varying: comes from vertex shader
    - b. gl_Position: it defines the position of the respective vertex
    - c. gl_FragColor: it defines the color to be used to render the fragments

- d. via a varying variable
    - declare it in both vertex and fragment shader programs
    - in the vertex shader program give it a value
    - use it in the fragment shader program
- e. `gl.getAttribLocation()`, `gl.getUniformLocation()`

```
let a_Position = gl.getAttribLocation(gl.program, "a_Position");
let u_ModelMatrix = gl.getUniformLocation(gl.program, "u_ModelMatrix");
```

- f. `gl.uniform*()`, `gl.vertexAttrib*()`
- g. compiling it once at runtime gives better performance than interpreting it throughout the program, espescially considering in most applications there are hundreds/thousands of verticies that need to pass through it and communication with the gpu has overhead.
    - create shader objects
    - load and compile shaders
    - create shader program
    - attach shaders to program
    - link program to GPU
    - use program
- h. `gl.drawArrays()`
- i. vertex shader program could look something like this

```
attribute mat4 a_Position;
uniform mat4 u_TransMat;
void main(){
    gl_Position = a_Position * u_TransMat;
}
```

`u_TransMat` can be set from javascript similar to the following

```
const u_TransMat = gl.getUniformLocation(gl.program, "u_TransMat");
gl.uniformMatrix4fv(u_TransMat, false, new Float32Array([...]));
```

3.
- a.
    - ELEMENT_ARRAY_BUFFER: buffer contains indicies that point to vertex data
    - ARRAY_BUFFER: buffer contains vertex data
    - ARRAY_BUFFER is more efficient as all the items are in the same memory location
        - this reduces probability of a cache miss
        - no need for deference operation
        - all data can be transfered from ram/cache/gpu at same time reducing number of operations
- b.
    1. `gl.createBuffer`: allocate memory for buffer
    2. `gl.bindBuffer`: make buffer active

3. `gl.bufferData`: allocate storage and initialize
4. `gl.vertexAttribPointer`: assign it an attribute variable
5. `gl.enableVertexAttribArray`: enable attribute variable as array

- c.

```
// (index: number, size: number, type: number, normalized: boolean,
stride: number, offset: number): void
gl.vertexAttribPointer(
    index, // attribute pointer to assign
    size, // number of values needed
    type, // datatype of each element (ie - `gl.FLOAT`)
    normalized, //
    stride, // number of bytes per vertex
    offset, // number of bytes into vertex before desired data
)
```

- d. `gl.enableVertexAttribArray`, `gl.disableVertexAttribArray`
- e. `gl.deleteBuffer`
- f. `gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, 5 * buffer.BYTES_PER_ELEMENT, 2 * buffer.BYTES_PER_ELEMENT);`