Brack Turner
CS411 AS3

1a. WebGL is a rendering system using HTML context for browsers and web applications. OpenGL is used generally in desktop applications. OpenGL ES is the subset stripped down version of OpenGL and doesn't contain glbegin and glend commands.

1b. The color buffer and depth buffer contain the components values of the render target. The stencil buffer is more of a per-pixel masking buffer that changes the values in opengl to your discretion

1c. 4fv takes a vector array with float values as an argument while 1f takes one float value directly as an argument. 4fv is just a vector version of vertexattrib*

1d. The vertex shader performs operations on the vertices and generally performs transformations on input vertices. The fragment shader operations dictate the color, contrast, brightness, etc. of a given pixel. The fragment is the collection of data and values produced by the rasterizer.

1e. Webgl can only draw points, lines, and triangles. More complex structures and polygons are made by combining these objects.

1f. gl.viewport() takes 4 arguments. The first two specify the lower left corner, then the last two specify the width and height of the viewport; which is the drawable area in your OpenGl view. For WebGL purposes we use the gl.viewport(x_offset,y_offset, …,...) instead.

2a. Attribute variables define the vertex data for each vertex (positio). Uniform variables globally pass the same data to all vertices. Varying variables contain data passed from the vertex shader(vertex color)

2b. gl_Position is predefined in the vertex shader. Nothing will render without it because the data from the vertex shader is passed to the fragment shader to make the position data visible.

2c. gl_FragColor must be assigned in the fragment shader for rendering to occur. IF left undefined, it will remain undefined for the rest of the pipeline. That is why it is a principal variable.

2d. Vertex color can be passed by interleaving with the vertex coordinates through the shaders or using a representing array with the coordinate information

2e. gl.getAttribLocation(gl.program, 'a_Postition');

2f. gl.vertexAttrib1f(a_Position, 5.0);

2g. Create shader objects, load and compile the shaders, create the shader program as .js, attach the shaders to the program, link the program to the GPU, then utilize. By doing this at runtime, it means shaders don't have to be precompiled and makes loading shaders much simpler for different GPU architectures.

```
2h. Var shader = gl.createShader(type);
2i. Attribute vec4 a_Position;
    Uniform mat4 u_xformMatrix;
    Void main() {
        gl_Position = u_xformMAtrix * a_Position;
}
//Shader applies matrix to position attribute
Var xformMAtrix = new Float32Array([
1.0, 0.0, 1.0,
0.0, 1.0, 0.0,
0.0, 0.0, 1.0]);
Var u_xformMAtrix = gl.getUniformLocation(gl.program, 'u_xformMatrix');
gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix);
// define matrix and pass the translation matrix to the vshader
```

3a. Array_buffer specifies that the buffer object has vertex data while the element_array_buffer specifies the index values pointing to the vertex data. Element_array_buffer is more efficient in storage because it contains pointer values and is better storage wise for more complicated objects with many vertices

3b. Create the buffer to allocate memory for the buffer. Bind the buffer by connecting it to the array buffer. Allocate storage and initialize the object with data. Assign the buffer object to an attribute variable. Finally enable the attribute variable as an array. ACtivates the connection in the vertex shader

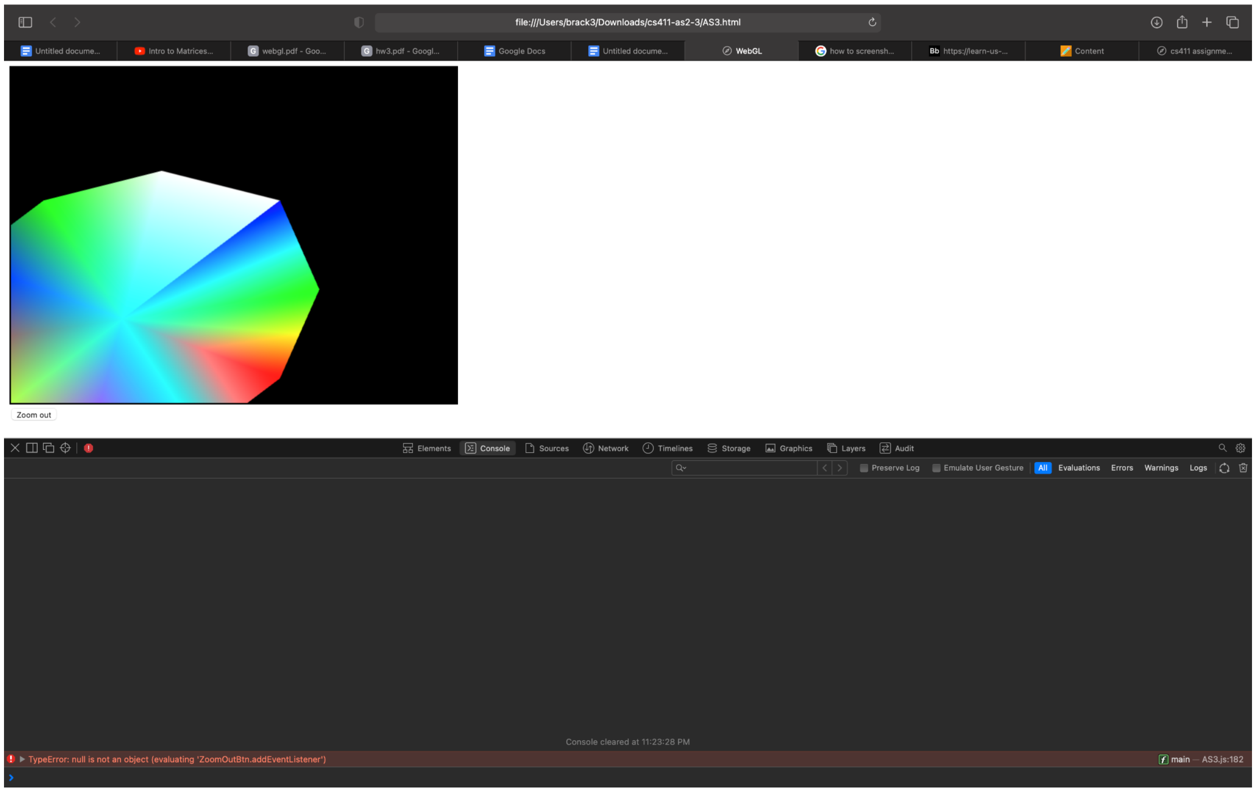
3c. Location: a pointer the the shader's attribute variable
Size: the number of elements per vertex
Type: the data type in the buffer object
Normalized: specifies whether the vertex is fixed point: true or false
Stride: specifies the byte offset between consecutive vertex attributes
Offset: specifies the offset in bytes to the first component of the first vertex attribute in the given array

3d. gl.enableVertexAttribArray & gl.disableVertexAttribArray(vertexBuffer)
3e. gl.deleteBuffer(vertexBuffer)

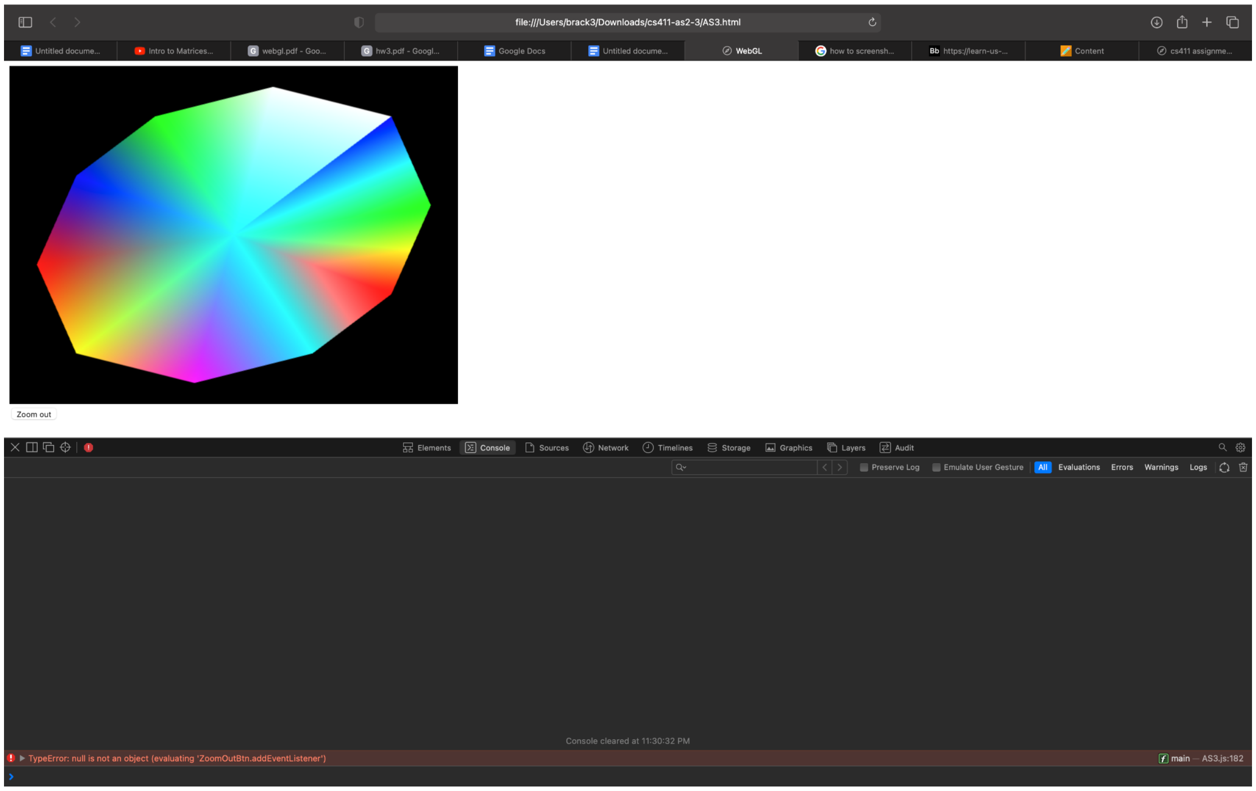
```
3f. Var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
    gl.vertexAttribPointer(a_Position, 2, gl.Float, false, FSIZE, 0);
    gl.enableVertexAttribArray(a_Position);
    // position
   Var a_Color = gl.getAttribLocation(gl.program, 'a_Color');
   gl.vertexAttribPointer(a_Color, 3, gl.Float, false, FSIZE, FSIZE);
   gl.enableVertexAttribArray(a_Color);
```

For the WebGl portion I was only able to generate a 10 2D triangle polygon with the triangle fan function. I then interleaved the color data into the vertex data so that each specific vertex had a different color.

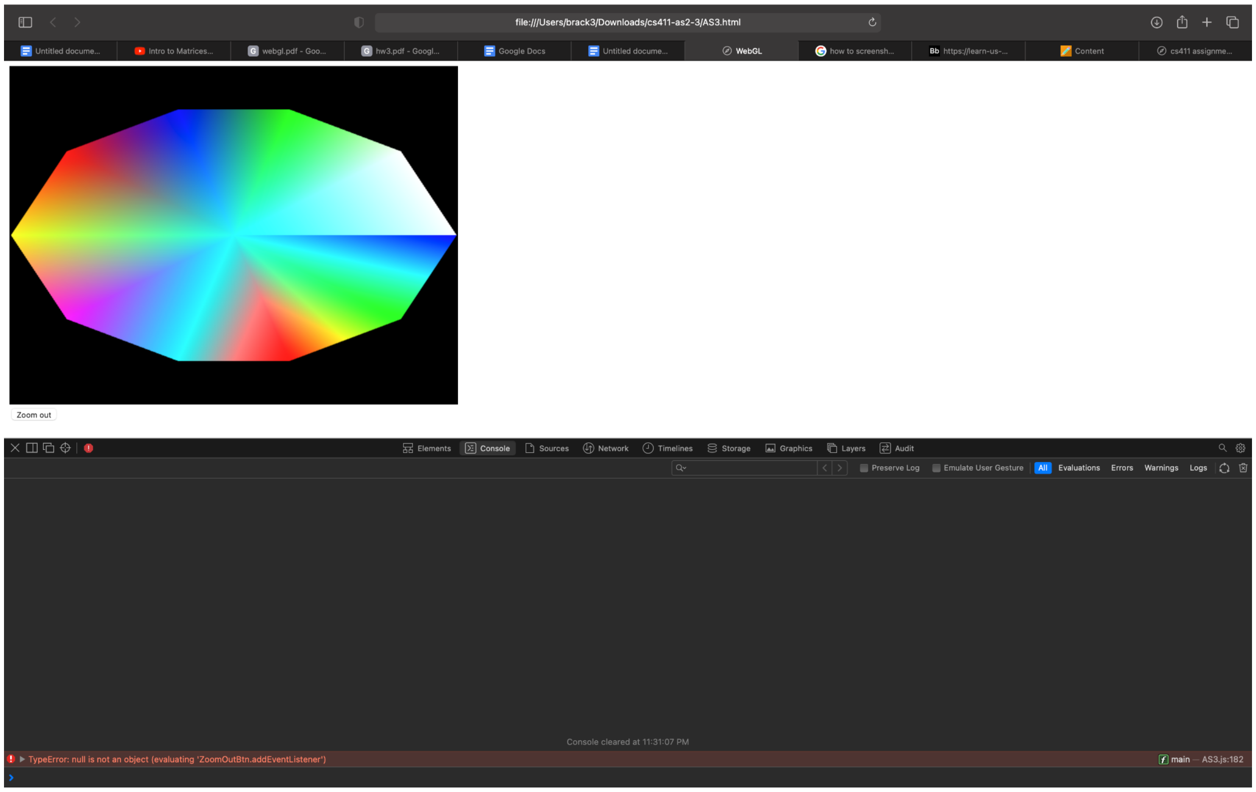
I was not able to figure out how to toggle the rotation matrix or the translation matrix using buttons. However I was able to apply the transformation matrices on the coordinate positions. Understanding the naming conventions and buffer object data is still a difficult challenge for me to alter the colors from RGB to BGR.



```
Console cleared at 11:23:28 PM
TypeError: null is not an object (evaluating 'ZoomOutBtn.addEventListener')          main — AS3.js:182
```



```
Console cleared at 11:30:32 PM
TypeError: null is not an object (evaluating 'ZoomOutBtn.addEventListener')          main — AS3.js:182
```



```
Console cleared at 11:31:07 PM
TypeError: null is not an object (evaluating 'ZoomOutBtn.addEventListener')          main — AS3.js:182
```