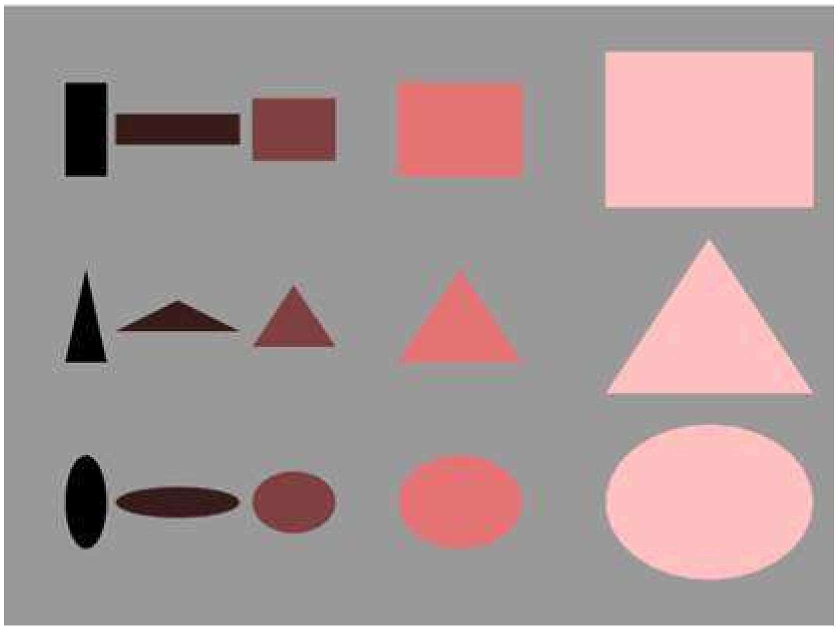
CSS 452: Programming Assignment #1
Working with Primitives

Due time: Please refer to our course web-site

Objective
In this programming assignment we will familiarize ourselves with the programming environment, JavaScript programming language, our IDE, AND, develop a vertical slice through the existing engine building simple drawing functionality from the user program all the way to the GPU. The goals are for us to feel comfortable working with the source code system, to have a better understanding of the Engine source code as provided, and to verify our understanding of the GPU interface.

Though relatively straightforward, dealing with the strangeness of JavaScript, the many different source code files, and the many new technologies can be confusing and intimidating at times. Please do start early and bring questions to class! The number of lines of code you need to write is actually relatively small (probably around 100-lines of code). However, instead of developing your own solution, you have to modify and insert your changes into an existing system. This provides you with a flavor for the kinds of work we do in this class. Very quickly programming assignments will demand you to modify a non-trivial game engine consisting of 10's of files and hundreds and thousands of lines of code.

Assignment Specification:
Here is screen shot of the results from the assignment:



Here are the details of what you need to support:

1. **Triangle:** the simplest shape with an area,
2. **Circle:** arguably the most general convex shape, and
3. **Transformed and instances:** seeing multiple instances of each of the shapes, with different dimensions.

Approach:

- **Web GL uniform variable:** make sure you understand Example 2.6 (Parameterized Fragment Shader), understand how the per-primitive color is passed from the CPU down to the GPU.
- **Duplicate the pattern:** Now, duplicate the above per-primitive color pattern to support ***uOffset*** and ***uScale*** (both vec2). Modify your simple_vs.glsl to work with the ***uOffset*** and ***uScale***:

```
        // Primitive offset and scale
        uniform vec2 uOffset;
        uniform vec2 uScale;

        void main(void) {
                // Convert the vec3 into vec4 for scan conversion and
                // assign to gl_Position to pass the vertex to the fragment
            shader
                gl_Position = vec4(aSquareVertexPosition, 1.0);
                gl_Position.x = gl_Position.x * uScale.x + uOffset.x;
                gl_Position.y = gl_Position.y * uScale.y + uOffset.y;
        }
```

    - Notice the above code change the size of the vertex, and then move it. Think about this, this is how we will support drawing multiple instances of primitives in general! In the next few lectures, we will develop the math tools and architecture support to implement this functionality ***and*** to hide everything from our user.

- Make sure you can draw multiple instances of the square with different dimensions!
- Now, simplify the square support to support a triangle.
    - Make sure you test and can support the drawing of triangle just like you can for the square (multiple instances and at different location with different scales)
- Now, take a read of the followings:
    - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/drawArrays or
    - https://www.tutorialspoint.com/webgl/webgl_drawing_a_model.htm
    - You will need to work with TRIANGLE_FAN to draw the circle. Remember, vertices on a unit circle that is approximated with ***mCircleNumVertices*** is defined by

```
            var delta = (2.0 * Math.PI)/ (mCircleNumVertex-1);
            for (let i = 1; i<=mCircleNumVertex; i++) {
            var angle = (i-1) * delta;
                    let x = 0.5 * Math.cos(angle);
                    let y = 0.5 * Math.sin(angle);
            }
```

Hints:

1. This assignment also serves as a motivation for learning about transformation in Chapter 3, as such, you are ***not*** allowed to use materials from Chapter 3. My implementation is based on Example 2.6 (parameterized fragment shader).

2. Make sure you understand:
    a. **engine/vertex_buffer**: you have to create two other buffers just like what we have done for the square.
    b. **simple_shader**: you need to understand from which buffer you want to load to support the drawing of the shape
    c. **glsl_shaders/simple_vs**: recall that the aVertexPosition is connected to a buffer that contains the vertices of the unit square.

3. You will probably need to define an array to keep track of all the circle vertices.

a. Do a google on: “*javascript array*”. Look at: http://www.w3schools.com/js/js_arrays.asp

i. You can initialize an array by:

```
var myArray = [];  // notice the empty “[]”
```

b. To add to the array:

```
myArray.push(newObj); // add to the end of the array
```

Credit Distribution

Here is how the credits are distributed in this assignment:

1.	Size and Position of rectangle		30%
	a. At least 5 instance of Rectangle b. No overlaps with different colors c. At least 1 with Width > Height d. At least 1 with Height > Width e. At least three distinct sizes	10% 10% 10% 10% 10%	
2.	Support Triangles		30%
	a. At least 5 instances b. No overlaps with different colors c. At least 1 with Width scale > Height scale d. At least 1 with Width scale < Height scale e. At least three distinct sizes	10% 10% 10% 10% 10%	
3.	Support Circles		30%
	a. At least 5 instances b. No overlaps with different colors c. At least 1 with Width scale > Height scale d. At least 1 with Width scale < Height scale e. At least three distinct sizes	10% 10% 10% 10% 10%	
4.	Proper submission		10%
	a. Zip file names with **NO SPACES** b. No extra unused files/folders (E.g., Test folder) c. Styles (project name, variable names, etc.)	10% 10% 10%	

This programming assignment will count 6% towards your final grade for this class.

Creativity and Extra Credits: Support other shapes! But make sure Rectangle, Triangle, and Circles do not overlap. We want to be able to observe the correctness of the shapes.