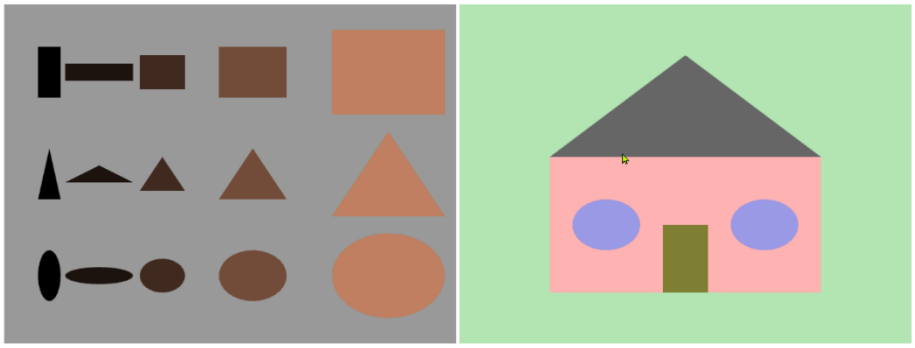
CSS 452: Programming Assignment #2
Drawing and Primitive API

Due time: Please refer to our course web-site

Objective
In this programming assignment we will continue to familiarize ourselves with the programming environment, JavaScript programming language, our IDE, and, take a first look at building an API by developing a vertical slice through the existing engine. You will build and extend drawing functionality to support drawing with different primitive types 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, to verify our understanding of the GPU interface, and, to begin getting use to developing API functionality as opposed to developing actual user programs.

WARNING: 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 integrate your changes into an existing system, provide a stable and well-defined interface for given programs. This provides you with a flavor for the kinds of work we do in this class. In the next little while, programming assignments will involve non-trivial game engine modifications consisting of 10's of files and hundreds and thousands of lines of code. Lots of work, but fun work!

Assignment Specification:
Here is screen shot of the results from the assignment (yes!! There are two instances of drawings!):



You must modify and extend the engine to support three different primitive drawing functions:

- drawRectangle(color, at, size)
- drawTriangle(color, at, size)
- drawCircle(color, at, size)

Here are the default size/position of each of the primitives:

- Rectangle: as given in the game engine
- Triangle: (0, 0.5), (-0.5, -0.5), and (0.5, -0.5)
- Circle: Center at (0, 0), radius of 0.5

Where in each case,

- **color**: a 4-float array with: [r, g, b, a] color of the primitive
- **at**: a 2-float array with [x, y] center position of the primitive
- **size**: 2-float array with [width, height] size of the primitive

Note, the above functions are provided to you as the API where you do not have the option to re-defined or change the above functions names or the parameters passed.

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.

Please refer to the provided source code files:

- **index.html**: You can use this file as is, or modify as you see fit
- **my_game.js**: Please ***do not*** modify this file. This code computes and generates the image on the left. This is the *general* test case, verifying the correctness of the defined functions with different combinations of parameters.
- **my_home.js**: This is the "*let's do something useful*" test case, computes and draws the nice little house to the right. Here are the dimension and location of the primitives in my home:
    - Home body: Rectangle size: 1.2x0.8, at (0.0, -0.3)
    - Roof: Triangle size: 1.2x0.6, at (0.0, 0.4)
    - Left window: Circle radius: 0.3, at (-0.35, -0.3)
    - Right window: Circle radius: 0.3, at (0.35, -0.3)
    - Door: Rectangle size: 0.2x0.4, at (0.0, -0.5)

  You are welcome to modify to ***add*** to this test case but not removing anything. ***Hint:*** interesting additions will earn you extra credits!

In this assignment you must work with these provided end-user program source code files and develop the proper support.

Hints

- 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).

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

- Approach: the following are the steps I followed in my own implementation
    - **Draw with any buffer:** simple_shader.js: Analyzed activate() function, parameterize the gl.bindBuffer() to connect to a buffer that is passed in as a parameter
    - **drawSquare() function**: core.js: define drawRectangle() function and active the shader by passing in the square vertex buffer
    - **Test**: to make sure I can draw a square
    - **Draw position/size**: modify simple_vertex.glsl to support size and position (refer to the following), modify simple_shader::activate() to support the size/position.
    - **Test**: to make sure I can draw multiple squares at different position and size
    - **Triangle:**
        - vertes_buffer.js: Studied the setup and duplicate the pattern for triangle vertices
        - core.js: define drawTriangle() function

        - test: make sure triangle is ok
    - **Circle:** Repeat the Triangle process

    Notice the multiple testing steps, the strategy of splitting the assignment into small parts, implement the simplest part first, test to ensure proper functioning before continuing.

- **Simple_Vertex.glsl:** modification, this is rather intimidating and involved. The following is my 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 forward the change to fragment shader
          gl_Position = vec4(aVertexPosition, 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: changes 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.

- **Circle definition:** take a read of the followings:
    - 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);
}
```

    - More about TRIANGLE_FAN: take a read of
        - https://developer.mozilla.org/en-US/docs/Web/API/WebGLRenderingContext/drawArrays or
        - https://www.tutorialspoint.com/webgl/webgl_drawing_a_model.htm

---

Credit Distribution

Here is how the credits are distributed in this assignment:

1.	Size and Position of rectangle: as per-API specification		25%
	a. At least 5 instance of Rectangle	10%	
	b. No overlaps with different colors	10%	
	c. At least 1 with Width > Height	10%	
	d. At least 1 with Height > Width	10%	
	e. At least three distinct sizes	10%	

2.	Support Triangles as per-API specification		25%
	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 as per-API specification		25%
	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.	Support my_game.js (no modification) and my_home.js		15%
	a. Drawing in the second canvas b. Proper background color c. Include all provided shapes d. All shapes with correct size/position/color	5% 5% 10% 10%	
5.	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 8% towards your final grade for this class.

Creativity and Extra Credits: Support other shapes! Create interesting scene!