Com S 336
Fall 2023
Homework 1

Here are a few exercises to try out some of the fundamentals we've looked at so far:

- get comfortable with JavaScript
- carefully read the sample code and know what each line of code is doing
- understand interpolation

None of them is very long in terms of lines of code (e.g. a couple dozen lines at most), but you are bound to get stuck somewhere, so start early and talk to me, or the TAs, or post on Piazza, as needed!

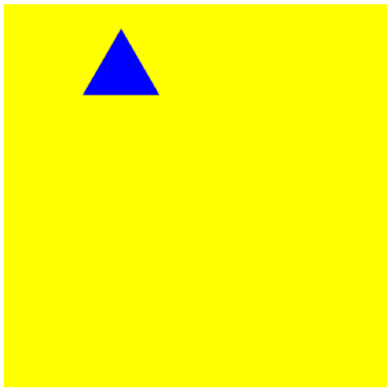
Please submit ONE zip archive on Canvas, containing the files indicated at the beginning of each problem. When working from an existing example, do not unnecessarily reformat or otherwise modify the code that does not need to be changed (so that we can diff your submission against the original). All referenced sample code can be found on

https://stevekautz.com/cs336f23/examples/

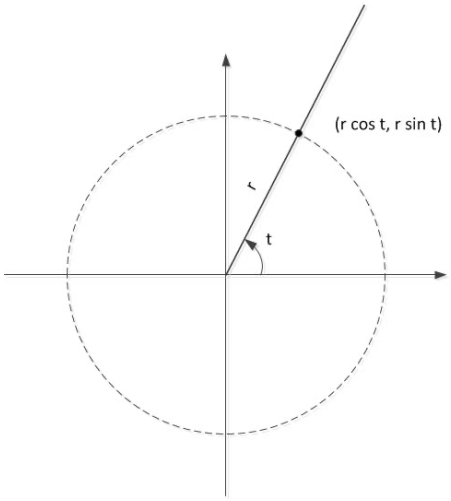
1. (*Please turn in two new files named* ***`problem1.html`*** *and* ***`problem1.js`****.)*

Modify **`GL_example1a_with_animation`** so that :

- instead of a red unit square on a cyan background, it draws a blue equilateral triangle with sides of length 1 on a yellow background
- instead of moving back and forth, it moves so that its *center* travels in a circle of radius 0.8 at a rate of one degree per frame
- the page has a text box for entering a scale factor (this should scale the triangle, *not* the radius)



This is pretty straightforward, you'll just need another uniform variable for the y shift (or use a vec2). To calculate the position, use the sine and cosine functions as pictured below. Trig functions are available in JavaScript as **`Math.cos()`**, **`Math.sin()`**, and of course you might also need **`Math.PI`**. (Remember the JS trig functions expect radian measure! If you wish, you can use the handy function **`toRadians`** in **`util/cs336util.js`**.)



The html code for a text box looks like:

```
Scale: <input id="scaleBox" type="text" value="1.0" size=4/>
```

where **`value`** is the default text and **`size`** is the width. To get the value, use the built-in JS function **`parseFloat`**, e.g.,

```
let scale = parseFloat(document.getElementById("scaleBox").value);
```

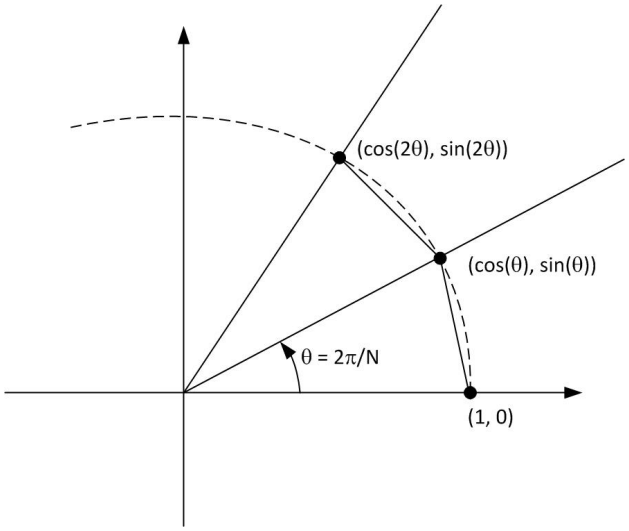
See the sample code **`examples/intro/text_box.html`** for an example using a text box and attaching a handler.

Remember: you don't want to reload new vertices when the scale changes! Just use a multiplier in the vertex shader. (As a warm-up, try modifying the vertex shader for **`GL_example1a_with_animation`** to make the square one-quarter it's initial size.)

2. (*Please turn in two new files named* **`problem2.html`** *and* **`problem2.js`**.)

Modify **`GL_Example1a`** to draw a regular N-gon instead of a square. It is not hard to compute the coordinates in a loop using the sine and cosine functions as pictured below. If you have not used JS arrays before, note you can append elements using the **`push()`** method (like a Java ArrayList). Trig functions are available in JS as **`Math.cos()`**, **`Math.sin()`**, and you might also need **`Math.PI`**. Scale the radius down to 0.8 so it's all visible. N should be selectable via an html text box.

You should load the new vertex data into the buffer *only* when a new value of N is selected, not every frame. See the sample code **`examples/intro/text_box.html`** for an example using a text box and attaching a handler.



Note: Remember that the argument to the constructor **`Float32Array`** or **`Uint16Array`** is a JavaScript array, not a sequence of numbers. That is, we can write

```
var myVertices = new Float32Array([1.0, 2.0, 3.0, 4.0]);
```

but NOT

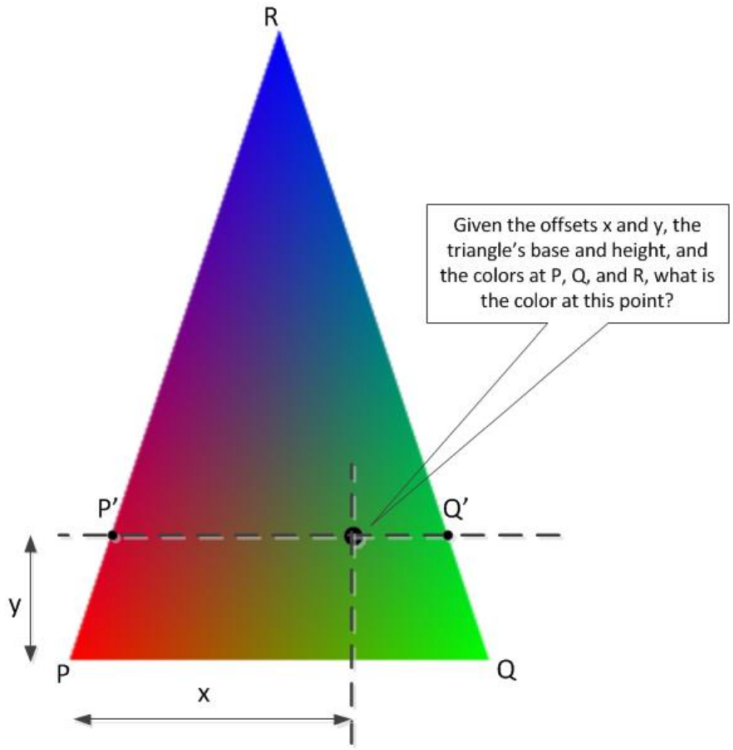
```
var myVertices = new Float32Array(1.0, 2.0, 3.0, 4.0);
```

To fill in the circle, you can use any strategy you want but the "obvious" one is list the coordinates counterclockwise and put (0, 0) at the beginning of the buffer and use TRIANGLE_FAN.

3. (*Please turn in your modified version of* **`color_interpolator.js`**.)

Write a JavaScript function findRGB that performs interpolation of colors associated with the corners of an isosceles triangle (simulating what is done by the rasterizer in GL_example2). That is, given the *base* and *height* of the triangle, *colors* for the three corners, and an integer x and y *offset* within the triangle, find the interpolated values for red, green, and blue at (x, y). (Don't worry about alpha here, assume it's always 1.0.)

The exact signature for the function is in the file **`color_interpolator.js`** along with a definition for a simple type representing an RGBA color.



There are several ways you could do this. One is to use a Fahrenheit-to-Celsius conversion to find interpolated values at P' and Q' along the vertical legs of the triangle, and then do it again to interpolate horizontally between those two values. Note you'll have to separately interpolate values for red, green, and blue. A slicker and more general solution would be to use *barycentric coordinates*, this is optional. If you are interested, maybe see

https://codeplea.com/triangular-interpolation to get started.

4. (*Please turn in the two new files* **`problem4.html`** *and* **`problem4.js`**.)

We have seen in GL_example1 how to use a uniform variable in a shader to shift a figure to the left or right, and we have seen in GL_example2 how to use varying variables to have colors associated with vertices interpolated across a triangle. Based on these examples, create files problem4.js and problem4.html as follows:

a) Draw a colored triangle, similar to the figure above, shifted to the left side of the canvas and draw a solid colored square shifted to the right side. You'll need a different shader for each figure. The one on the left can adapt the shader from **`GL_example2_varying_variables`** and the one on the right can adapt the shader from **`GL_example1a_with_uniform_color`** (you'll need to be able to set the color from your JS code).

b) Add a mouse handler that, when the mouse is clicked on the left triangle, the handler will ***use your function from problem 3*** to calculate what color would be at that pixel, and the application then sets the right-hand square to be that color. (A rudimentary color-picker!) Please note you are not trying to "sample" from the framebuffer to see what color is there – you're just using the function from #3 to **simulate** what the rasterizer would have done to interpolate the color.

For simplicity you may hard-code the canvas size as 400x400, the triangle coordinates as (-.5, -.75), (.5, -.75), and (0, .75), and the left shift amount as -0.5. Then the canvas coordinates for your triangle would always be (0, 50), (200, 50), and (100, 350).

If the mouse is clicked outside the triangle, you can ignore the click.

See **`examples/intro/mouse.html`** for how to get the mouse coordinates.