Final Exam

This is a preview of the published version of the quiz

Started: May 19 at 8:50am

Quiz Instructions

Question 1 6 pts

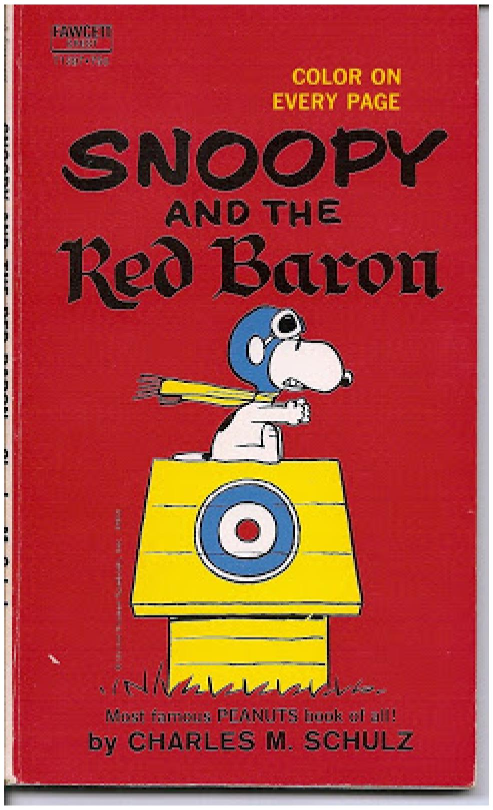
(Question 1) In this class, we've discussed numerous coordinate systems. For this question, match the **best** answer with the associated range of values for a particular coordinate system.

Below, the notation of $[a, b]$ indicates the range of values from a to b.

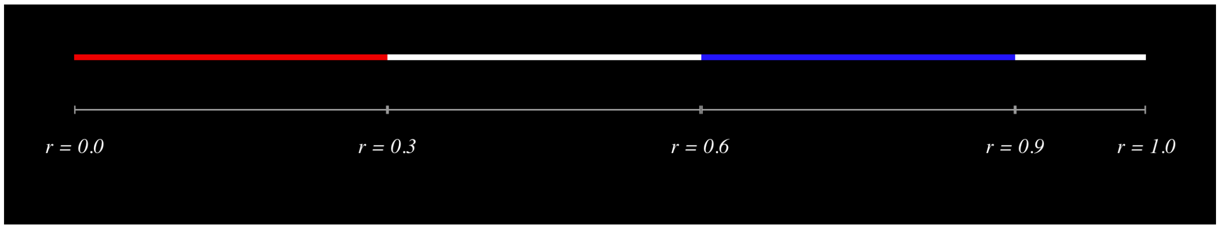
Values ranging from [-1,1] in each dimension	[Choose]
Values ranging from [0, 1] as distances from the eye	[Choose]
Values ranging from [0, 1] in the s, and t dimensions	[Choose]
Values that go from [-infinity,infinity] in any dimension	[Choose]
Values that go from [near, far] along the -z axis	[Choose]
Values that go from [0, width-1] and [0, height-1]	[Choose]

Question 2 12 pts

(Question 2) The latest project from the game studio you're working at is a contract form the Schulz family chronicling everyone's favorite canine, Snoopy, in his heroic adventure to save the world from tyranny. You're now responsible for rendering Snoopy's insignia on his trustworthy Sopworth Camel fighter (doghouse) in his battles with the Red Baron, as chronicled in the classic



The Art Department provided the following specification for the insignia



You are to write the *fragment* shader to generate the procedural texture representing the insignia. The associated vertex shader provides texture coordinates that range from $[-1, 1]$ in the *s* and *t* dimensions. The demanding artists require that the center of the insignia (the red dot) is centered at $(0, 0)$, and extend the full one unit to maximally cover the polygon you're given.

You are presented with the following framework for your shader:

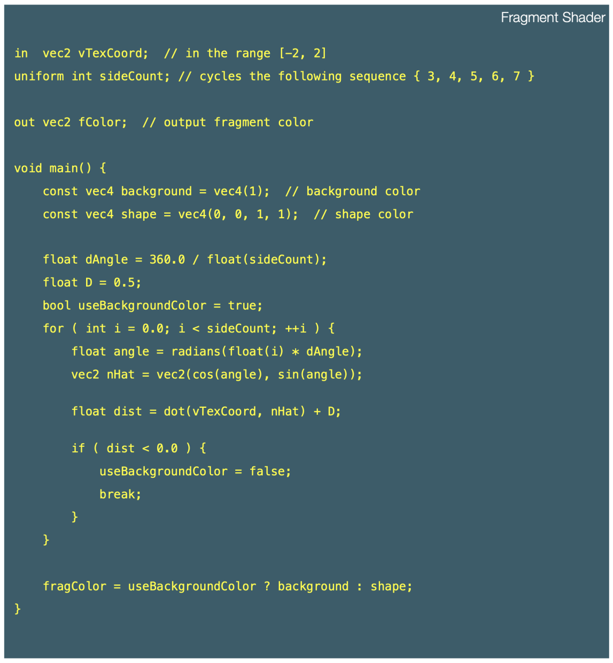
```
in vec2 vTexCoord;  // values between [-1,1] in (s,t)

out vec4 fColor;    // the final fragment color

void main()
{
    // write the code for the fragment shader, and submit that as your answer
}
```

One final consideration: the insignia should be compatible with any colors that lies underneath the insignia. You consider this, and realize that statement means that any fragments that aren't part of the insignia should be rendered transparent. Be sure to include that in your implementation.

(Question 3) Consider the following shader:



Fragment Shader

```
in  vec2 vTexCoord;  // in the range [-2, 2]
uniform int sideCount; // cycles the following sequence { 3, 4, 5, 6, 7 }

out vec2 fColor;  // output fragment color

void main() {
    const vec4 background = vec4(1);  // background color
    const vec4 shape = vec4(0, 0, 1, 1);  // shape color

    float dAngle = 360.0 / float(sideCount);
    float D = 0.5;
    bool useBackgroundColor = true;
    for ( int i = 0.0; i < sideCount; ++i ) {
        float angle = radians(float(i) * dAngle);
        vec2 nHat = vec2(cos(angle), sin(angle));

        float dist = dot(vTexCoord, nHat) + D;

        if ( dist < 0.0 ) {
            useBackgroundColor = false;
            break;
        }
    }

    fragColor = useBackgroundColor ? background : shape;
}
```

This fragment shader is used to shade a square, whose vertices include texture coordinate values that range from $[-2, 2]$ in both the s and t directions. Additionally, the shader variable `sideCount` iterates through the list: { 3, 4, 5, 6, 7 }, updating (with values wrapping around) once every thirty frames.

Question 3

4 pts

(Question 3.a) For the above shader, describe what is rendered across the execution of the application

Question 4

6 pts

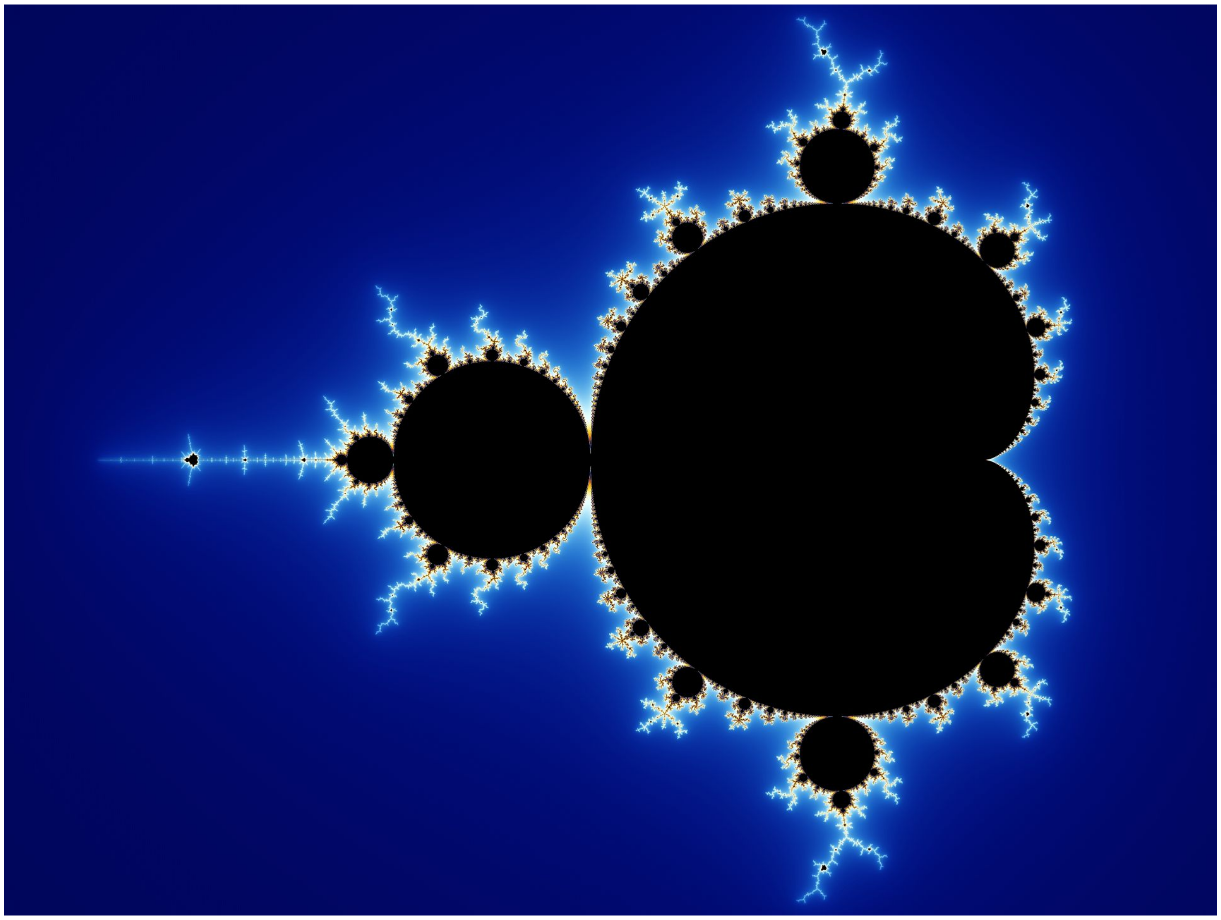
(Question 3.b) For a frame of your choosing, draw a diagram of the rendered shape, demonstrating how the shader determines which color to shade a fragment. Please indicate which value of `sideCount` you're considering. (***Hint**:* choosing one of the smaller values for `sideCount` will result in less work.)

Upload Choose a File

Question 5

14 pts

(Question 4) The **Mandelbrot set** **(https://en.wikipedia.org/wiki/Mandelbrot_set)** is the set of points in the complex (imaginary) plane that when iterated through the equation $z_{n+1} = z_n^2 + c$ doesn't diverge to infinity.



While it might not seem so, rendering a Mandelbrot set is relatively straightforward to implement using a fragment shader — mostly it is just a single `while` loop.   The points that are in the Mandelbrot set have a special property: they all lie within a circle of radius two centered at the origin of the complex plane, and that every iteration value, z_n, is less than two units for the origin. Put another way, the *magnitude* (i.e., length) of the vector from the origin to the point z_n is less than two.

Your job is to implement said fragment shader using the following information and functions.

To begin with, in the equation $z_{n+1} = z_n^2 + c$

- c is an imaginary number often written as $x + iy$, which can be nicely represented in GLSL's `vec2` type
- and the initial value of z_n, denoted as z_0, equals zero.

This means that the regardless of the value of, $z_1 = c$. To generate z_2, and every z_{n+1} after, we need to square the value of z_n and add c. You are provided with a GLSL function (which you can call in your fragment shader) that squares complex numbers, whose implementation is

```
vec2 complexSquare( vec2 z )
{
    return vec2( z.x*z.x - z.y*z.y, 2*z.x*z.y );
}
```

which means we can write the essential equation for this computation as

```
    z = complexSquare(z) + c
```

You iterate the above equation until one of two alternatives happens (you'll check these every iteration):

1. the magnitude of z is greater than two, at which point you know the iteration is going to diverge, and so you stop iterating
2. your loop reaches the maximum number of iterations

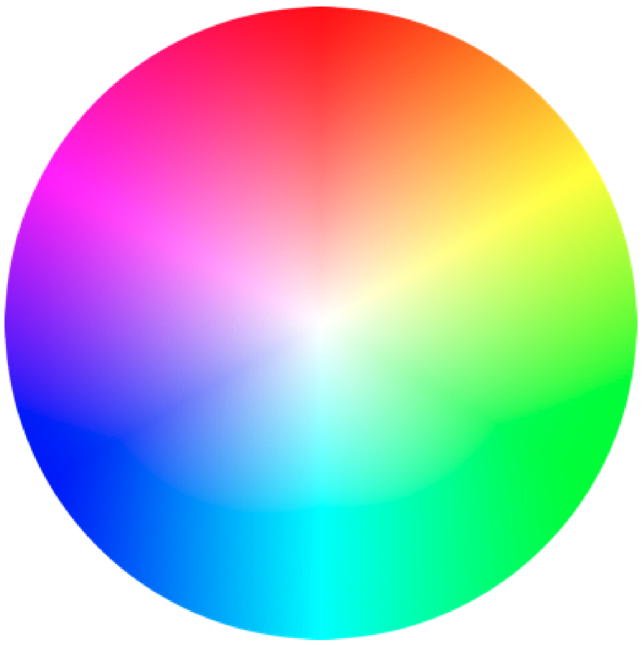
Let's say you wish to recreate the above image. For points that are in the Mandelbrot set (which means they satisfy point 2. above), you shade the fragment black.

For the points that diverge (those satisfying point 1.) you might compute the ratio of completed iterations to the maximum number of iterations, and use that value to look up a color in a texture (as they did in the image above), or perhaps just modulate a color value. For example, you might multiply the color blue by your ratio, and you'd get a reasonable approximation to the above image.

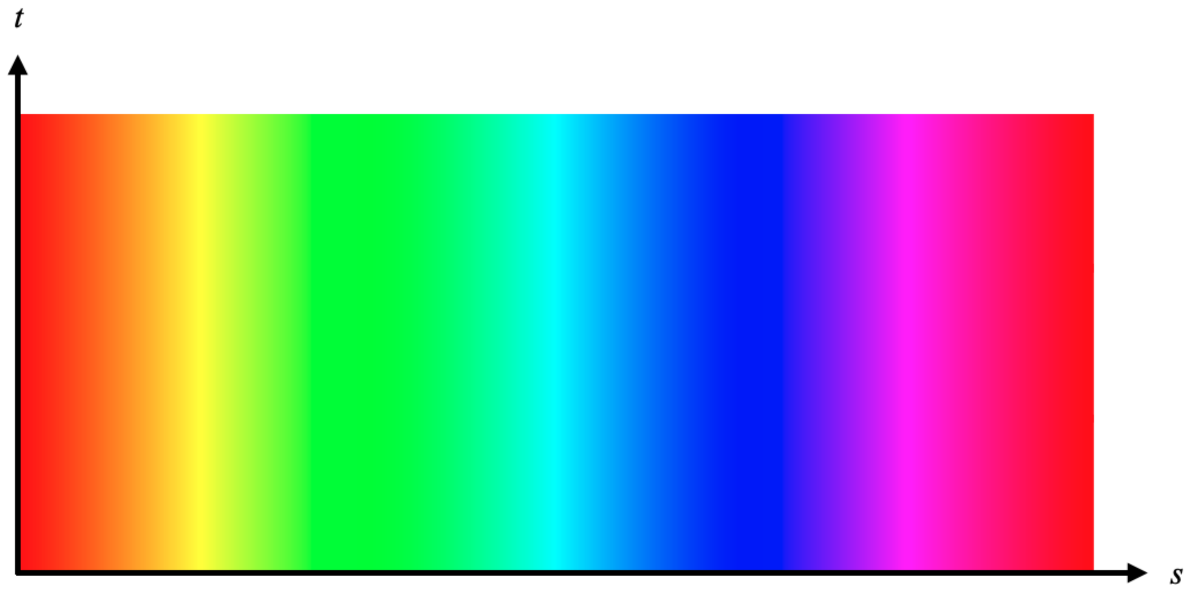
Please submit your shader source as the answer to this question.

(***Hint***: testing your implementation at **Shadertoy.com (https://www.shadertoy.com/new)** will give you a lot of confidence that you're on the right track).

(Question 5) As the star programmer at 🐷🍬 R Us (Pigments — get it?), you're tasked with implementing their new color picker, which is modeled off of what's commonly called a *color wheel*.



You also (wisely) decide that using a gl.TRIANGLE_FAN would make a good choice for rendering the color wheel. Further, you realize that while you could generate the RGB values necessary, that using a texture map like the following actually makes things simpler



and that you merely need to generate useful texture coordinates for the perimeter vertices.

Question 6

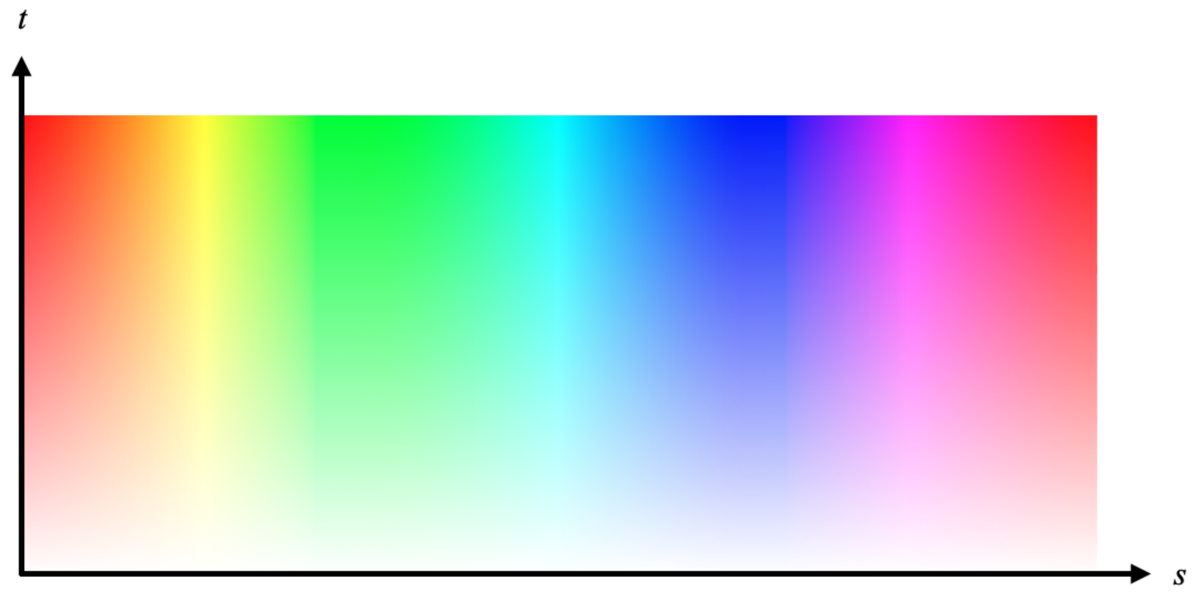
4 pts

(Question 5.a) Suggest a method to generate suitable texture coordinates that would help you shade the color wheel using the above texture.

Question 7

1 pts

(Question 5.b) Would a texture like the texture below would simplify the problem?



- ○ True
- ○ False

Question 8

3 pts

(Question 5.c) Explain your decision.

Not saved Submit Quiz