ADARSH SINGH
05/04/2025

Homework 3 - Colors

1. **Fill in the blanks below to convert colors between the RGB, HSV, and CMY models:**

	Red	Green	Blue	Hue°	Saturation%	Value%	Cyan	Magenta	Yellow
(a)	0	0	0	__	0	0	1	1	1
(b)	255	255	0	60	100	100	0	0	1
(c)	0	0	127.5	240	100	50	1	1	0.5
(d)	255	255	255	__	0	100	0	0	0
(e)	0	255	255	__	100	100	1	0	0
(f)	255	0	0	0	100	100	0	1	1
(g)	255	255	255	180	0	100	0	0	0
(h)	127.5	0	0	300	50	0	0.5	1	1

2. **Given a color c_1 = RGB(25,127,76), which color c_2 can you add to get a color pink? Define (using the RGB model) the pink you are aiming and then the color you added.**

3. **Sketch the color models for RGB (cube) and HSV (cone). In both model sketches, mark (with color name) the position of each of the following colors: black, white, gray, red, cyan, and yellow.**

4. **Given the color c_1 = RGB(255,255,0), if you decrease the "saturation" as defined in HSV until fully desaturated, describe the range of RGB colors you will pass through, as well as the final result.**

5. **Linearly interpolate the color halfway between red and blue in each of the following color models. (Requires a separate calculation for each subproblem, not just converting the resulting color.)**

   (a) RGB

   (b) HSV

   (c) CMY

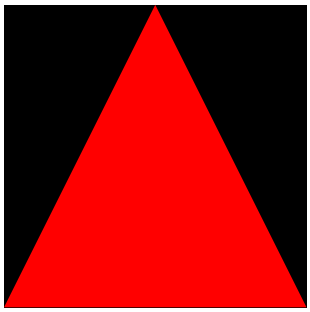
6. **Considering the results of Question 5. What can be concluded about linear interpolation in different color spaces?**

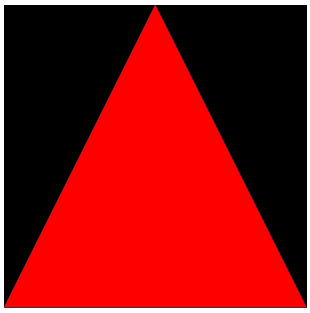
7. **Consider the following javascript snippet to setup a vertex buffer and the the pair of vertex/fragment shaders defined in GLSL:**

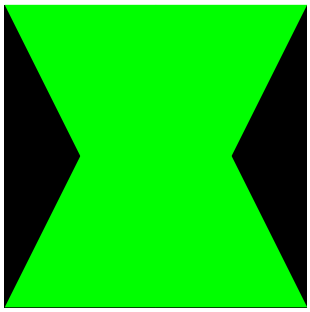
```
//  === VERTEX SHADER
attribute vec2 a_Position;
attribute vec3 a_Color;

varying vec3 v_Color;
main() {
    v_Color = a_Color;
```

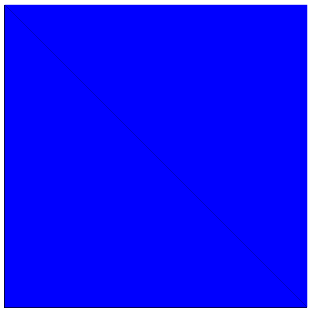
```
    gl_Position = vec4(a_Position, 0.0, 1.0);
}

//  === FRAGMENT SHADER
varying vec3 v_Color;
main() {
    gl_FragColor = vec4(v_Color, 1.0);
}

// === VERTEX BUFFER SETUP IN JAVASCRIPT
gl.clearColor(0.0, 0.0, 0.0, 1.0);
gl.clear(gl.COLOR_BUFFER_BIT);

let vertices = new Float32Array([ ___ ]);

let a_Position = gl.getAttribLocation(gl.program, "a_Position");
gl.vertexAttribPointer(a_Position, ___, gl.FLOAT, false, ___, ___ );
gl.enableVertexAttribArray(a_Position);

let a_Color = gl.getAttribLocation(gl.program, "a_Color");
gl.vertexAttribPointer(a_Color, ___, gl.FLOAT, false, ___, ___ );
gl.enableVertexAttribArray(a_Color);

gl.bufferData(gl.ARRAY_BUFFER, vertices, gl.STATIC_DRAW);
gl.drawArrays(gl.TRIANGLES, 0, ___ );
```

Note there are blanks in this code highlighted in red. For each of the images below, fill the blanks of the code that would generate the output images:

(a) 

(b) 

(c) 

(d) 

2. $C_1 = RGB(25, 127, 76)$

Target Pink = $RGB(255, 192, 203)$

$\rightarrow C_2 = RGB(230, 65, 127)$

3. • RGB Cube:
   - Black = (0,0,0) → origin color
   - White = (1,1,1) → far diagonal corner
   - Gray = (0.5, 0.5, 0.5) → center
   - Red = (1,0,0)
   - Cyan = (0,1,1)
   - Yellow = (1,1,0)

• HSV Cone:
   - Black = bottom tip
   - White = top center
   - Gray = vertical center axis
   - Red = around 0° on edge
   - Cyan = 180°
   - Yellow = 60°

4. Starting from RGB(255, 255, 0) – bright yellow.

As saturation decreases in HSV, the color moves toward white.

You'll pass through light yellow shades (pastel yellow) until it becomes white at full desaturation.

5. a) RGB:

Red = (255, 0, 0), Blue = (0, 0, 255)

→ Midpoint: RGB(127.5, 0, 127.5) (Purple)

b) HSV:

Red = (0°, 100%, 100%), Blue = (240°, 100%, 100%)

→ Midpoint hue = 120°

Result: HSV(120°, 100%, 100%) = Green

c) CMY:

Red = (0,1,1), Blue = (1,1,0) → Midpoint: CMY(0.5, 1, 0.5)

6 In RGB, it blends colors directly, producing visually smooth mixes.
In HSV, interpolation can jump hues (e.g. red to blue gives green).
In CMY, results may not match human perception due to subtractive mixing.
— Interpolation is model-dependent and affects color appearance.

7. a)

```
let vertices = new Float32Array([0, 0.5, 1, 0, 0,  -0.5, -0.5, 0, 1, 0,  0.5, -0.5, 0, 0, 1]);
gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 20, 0);
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, 20, 8);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

b)

```
let vertices = new Float32Array([-0.5, 0.5, 1, 0, 0,  -0.5, -0.5, 0, 1, 0,  0.5, -0.5, 0, 0, 1]);
gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 20, 0);
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, 20, 8);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

c)

```
let vertices = new Float32Array([-0.5, 0.5, 1, 0, 0,  0.5, 0.5, 0, 1, 0,  0, -0.5, 0, 0, 1]);
gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 20, 0);
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, 20, 8);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

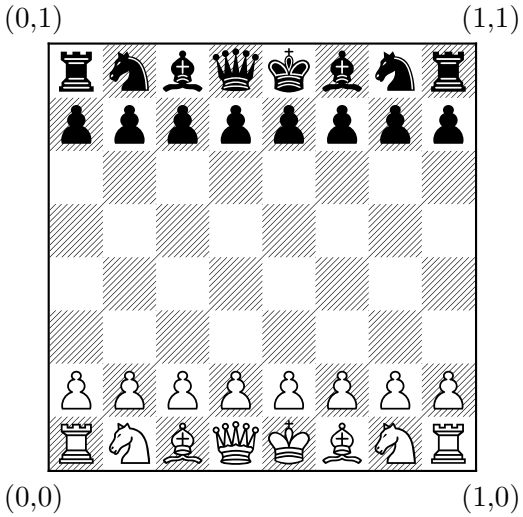
d)

```
let vertices = new Float32Array([-0.5, -0.5, 1, 0, 1,  0.5, -0.5, 0, 1, 1,  0, 0.5, 1, 1, 0]);
gl.vertexAttribPointer(a_Position, 2, gl.FLOAT, false, 20, 0);
gl.vertexAttribPointer(a_Color, 3, gl.FLOAT, false, 20, 8);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

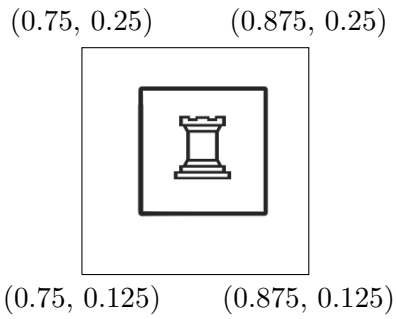
ADARSH SINGH
05/04/2025

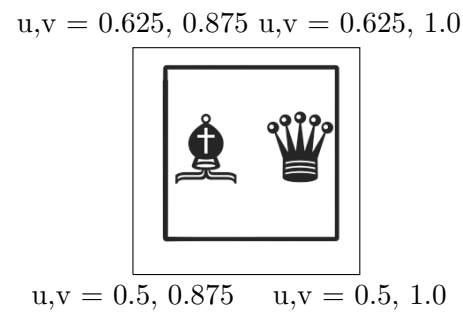
Homework 3 - Textures

1. **Considering the following chessboard texture with (u,v) coordinates:**

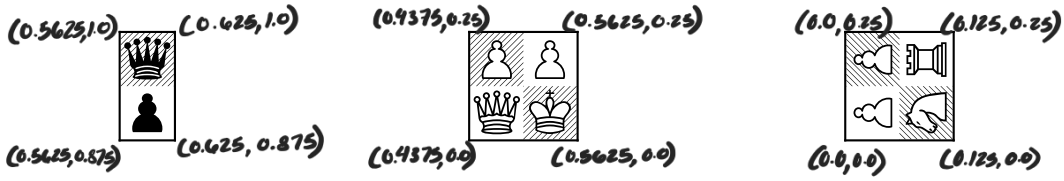


   (a) **Draw the approximate mapping on each quad if they were textured using the above image.**





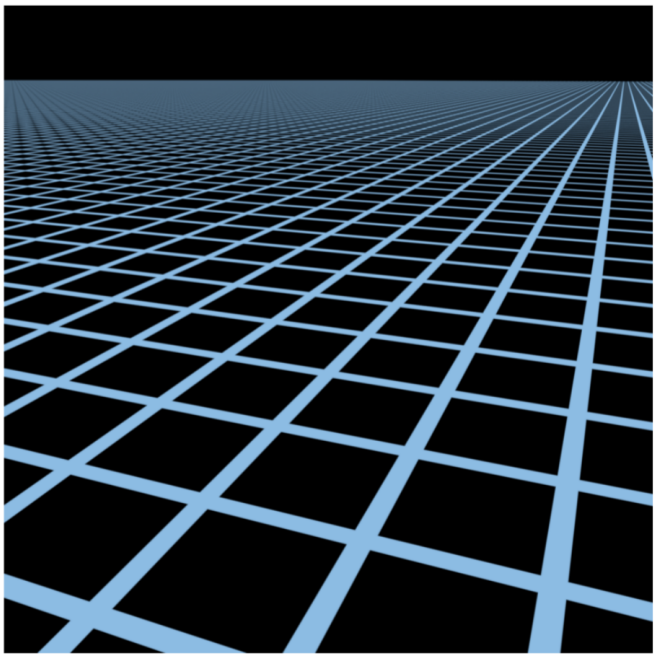
   (b) **Label the (u,v) texture coordinates for each of the 4 vertices that would result in the image below (just write them next to the corner of each square).**

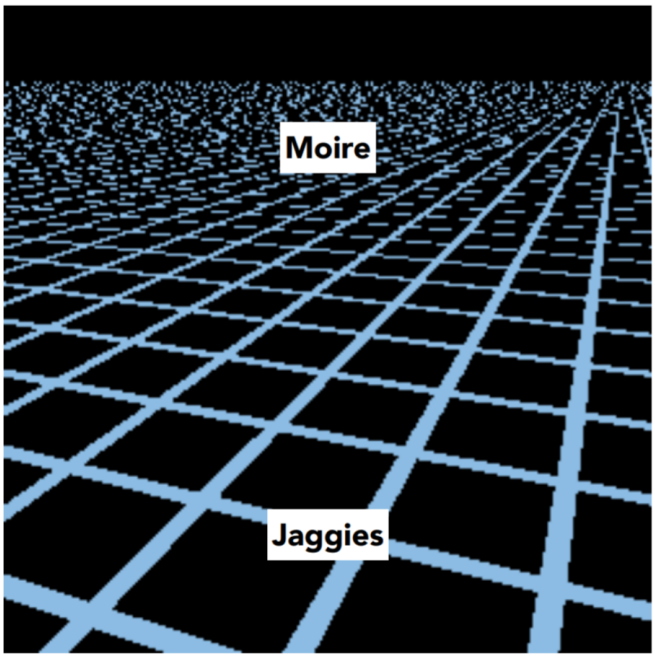


2. **Describe how and when one could use Spherical Parameterization to map a texture onto a 3D model.**

3. **In texture mapping, what is the difference between magnification and minification?**

4. **We textured a floor in our world and wish it looked like the left image, instead we got the right image, with speckled Moire patterns in the back and Jaggies in the front.**





   (a) **Explain why these Moire patterns appear and how to fix them.**
   (b) **Explain why these Jaggies patterns appear and how to fix them.**

5. **What is the following fragment shader doing?**

```
varying vec2 uv;

uniform sampler2D tex;
uniform vec4 baseColor;

void main() {
    vec4 texColor = texture2D(tex, uv);
    gl_FragColor = 0.5 * texColor + 0.5 * baseColor;
}
```

2. Spherical Parameterization maps a 2D texture onto a spherical 3D surface using spherical coordinates (θ, ϕ). It is useful when texturing round objects like planets, heads, eyeballs.
Use it when the model is approximately spherical, and you want consistent mapping without severe distortion or stretching.

3. · Magnification happens when a texture is scaled up (fewer texels cover more pixels), often causing blurriness.
· Minification occurs when a texture is scaled down (many texels per pixel), often causing aliasing.
– They differ in how textures are filtered based on whether you're zooming in or out.

4. a) Moire patterns appear due to aliasing when high-frequency texture details are downsampled improperly.
Fix: Use MIP mapping with trilinear filtering to smooth transitions and avoid sampling issues at a distance.
b) Jaggies appear due to aliasing when edges are rendered at low resolution or steep angles.
Fix: Use anti-aliasing techniques like MSAA (Multi-Sample Anti-Aliasing) to smooth the edges and improve visual quality.

5. The fragment shader blends the texture color with a base color equally.
`gl_FragColor = 0.5 * texColor + 0.5 * baseColor;`
– This creates a 50/50 mix of the texture and the color, giving a tinted or blended look.

ADARSH SINGH
05/04/2025

Homework 3 - Readings Ch5

1. What is the difference between using multiple buffers to store vertex data vs. interleaving vertex data in one buffer?

2. Describe the two process that take place between the vertex and fragment shaders.

3. In the context of the fragment shader, describe how varying variables can be used to interpolate data among fragments.

4. In the context of texture mapping, what is a magnification method? Enumerate and explain different methods.

5. In the context of texture mapping, what is a minification method? Enumerate and explain different methods.

1. Using multiple buffers means storing different vertex attributes (e.g., position, color, normal) in separate buffers. This can improve flexibility but may lead to more draw calls and memory fetches.
Interleaving stores all attributes for each vertex in a single buffer (e.g., position → color → normal → position...). This improves cache performance and is more efficient for rendering.

2. 1) Primitive Assembly: The GPU connects vertices into shapes (e.g., triangles or lines) after the vertex shader runs.
2) Rasterization: Converts these shapes into pixels (fragments), determining which screen pixels are covered and generating fragment data like interpolated attributes.

3. Varying variables are passed from the vertex shader to the fragment shader. During rasterization, their values are automatically interpolated across the surface of a primitive, so each fragment gets a smoothly blended value (e.g., color or texture coordinate) based on its position within the triangle.

4 Magnification occurs when a texture is scaled up (fewer texels cover more sparse pixels).
1) Nearest-neighbor filtering: Picks the closest texel. Fast but can look blocky.
2) Linear filtering: Blends nearby texels for smoother results. Preferred for quality.

5. Minification happens when a texture is scaled down (many texels map to one pixel), often causing aliasing.
1) Nearest-neighbor: Chooses one texel. Fast but can alias.
2) Linear filtering: Averages nearby texels. Smoother but still limited.
3) MIP mapping: Uses precomputed, scaled-down versions of the texture.
   - Nearest MIP: Picks the closest MIP level.
   - Trilinear filtering: Blends between two MIP levels for smoother transitions.