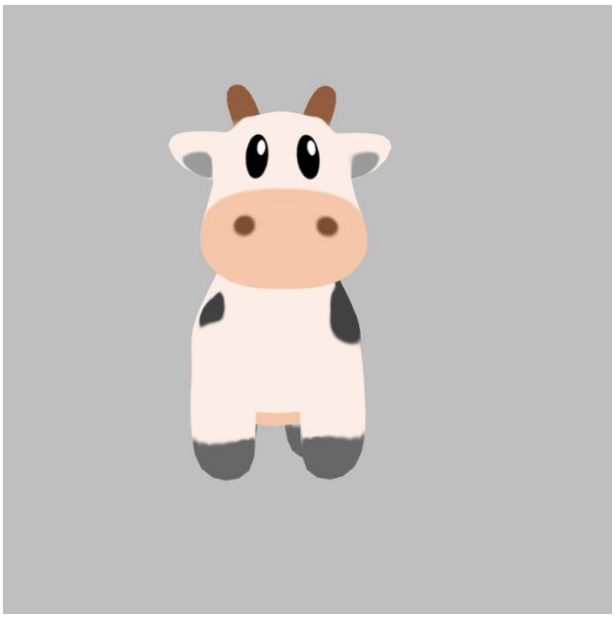
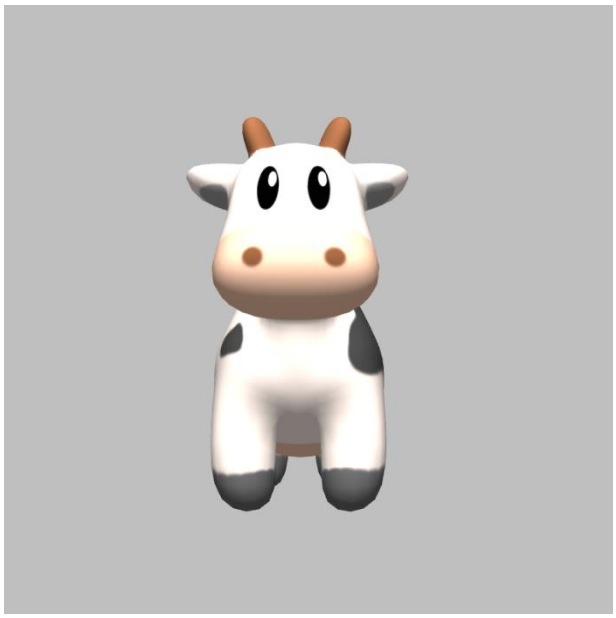
Computer Graphics 2024/25 — Lab 4

Name (ID) : Erfan Rafieioskouei – 240842587

A1 — Texture setup and rendering

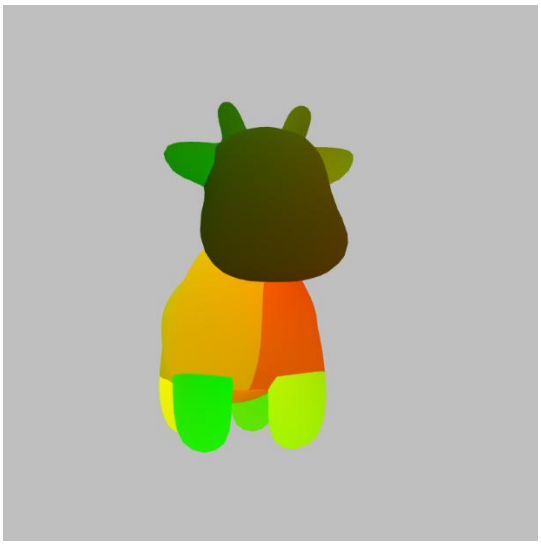




Differences between the models:

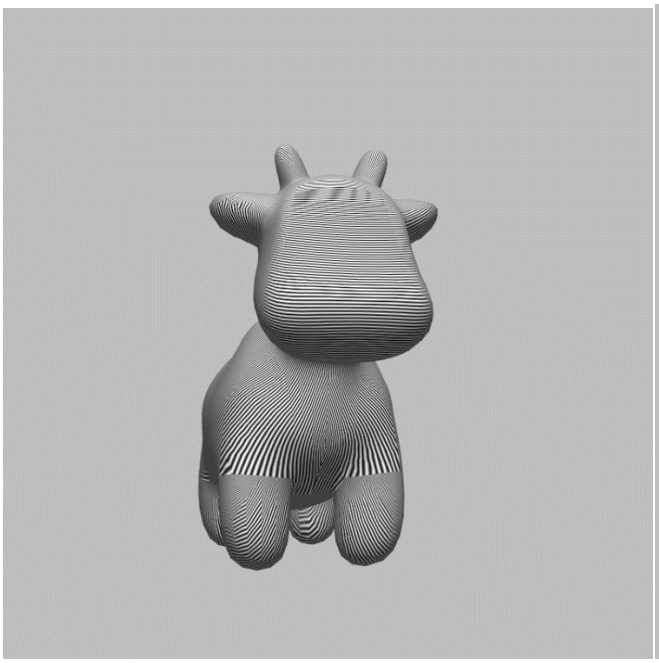
- **Lit Model (without texture):**
  Displays lighting effects on a uniform color, showing the object's shape and depth but without any texture detail.
- **Textured Model (without lighting):**
  Applies the texture directly, displaying the texture patterns but lacking depth cues and appearing flat.
- **Textured and Lit Model:**
  Combines both texture and lighting, showing detailed textures with realistic shading and highlights, enhancing the object's three-dimensional appearance.

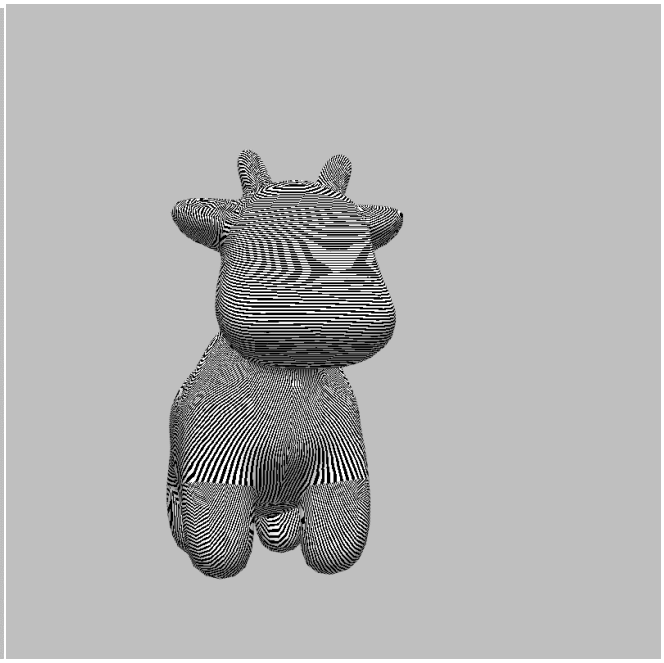
A2 — Texture coordinates



The head appears dark because, in that area, both **map.s** and **map.t** are close to 0, resulting in low red and green values. This means the head is mapped to the lower-left corner of the texture image where the coordinates are minimal.

A3 — Texture filtering





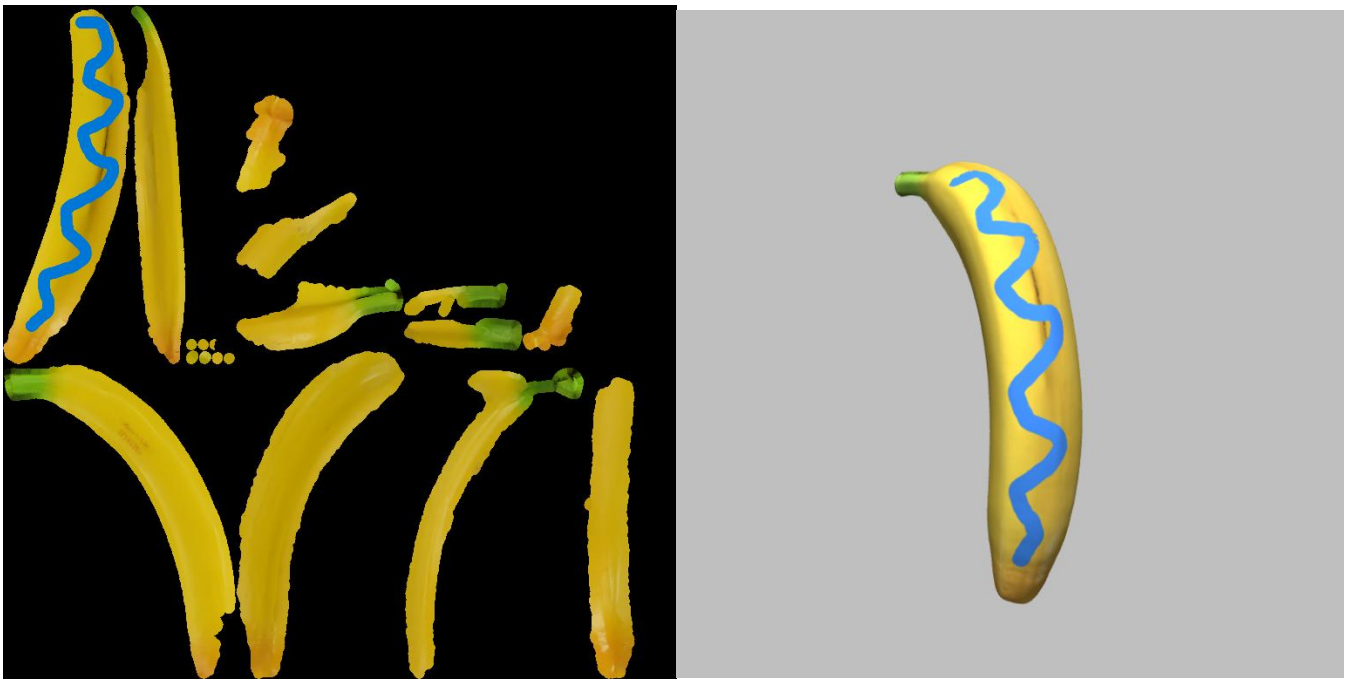
When you change the texture filter settings in the setup_texture() function to:

```
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
```

and then rotate or move the object, you'll notice severe aliasing artifacts, especially on areas like the back of the head. These artifacts manifest as pixelated textures and jagged edges.

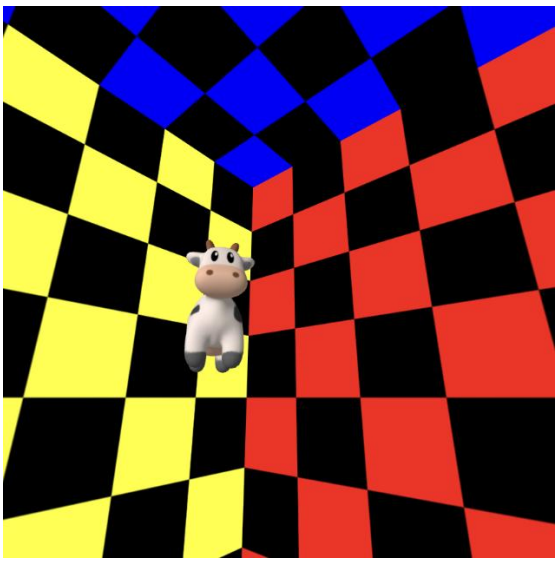
This occurs because gl.NEAREST uses nearest-neighbor interpolation, which selects the closest texel without any smoothing. When the texture is minified (when the object is far away) or magnified (when the object is close), this leads to abrupt changes between texels, causing visible blockiness and aliasing.

A4 — Texture editing



The effects on the textures could be seen in the rendering model.

B1 — Skyboxes

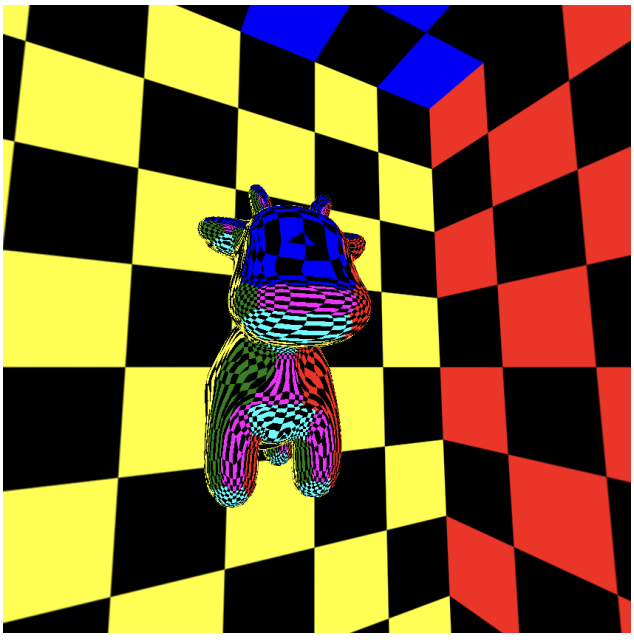


The intersection was desirable and the cow object collided with the skybox as it can be seen in the picture given.

By disabling depth testing before rendering the skybox with gl.disable(gl.DEPTH_TEST) , the skybox is drawn without considering depth values.
This means the skybox will always render behind all other objects in the scene, regardless of their positions. As a result, moving the main object further away and increasing its size will no longer cause it to intersect visibly with the skybox.

Re-enabling depth testing before rendering the main model using gl.enable(gl.DEPTH_TEST) is crucial.
Depth testing ensures that pixels closer to the camera obscure those that are further away. Without re-enabling it, the main model would render incorrectly, with surfaces potentially appearing out of order or showing visual artifacts. Enabling depth testing for the main model maintains the correct depth relationships within the object, allowing for proper 3D rendering.

B2 — Reflection mapping



The ambient, diffuse, specular, and reflection terms are combined by adding their weighted contributions to compute the final color. Each term influences the shading effect:

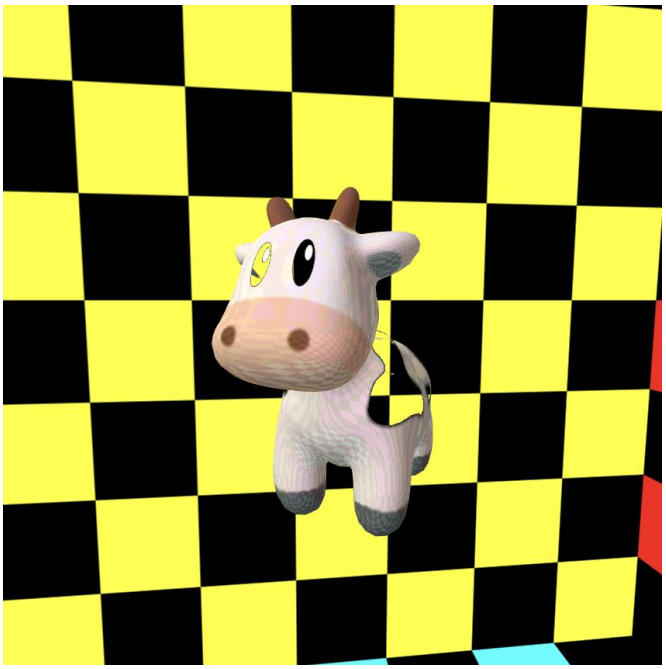
- **Ambient**: Base color under all lighting.
- **Diffuse**: Shading based on light direction and surface normal.
- **Specular**: Highlights where light reflects directly to the viewer.
- **Reflection**: Environmental reflection from the cube map.

C1 — RGB transformations

To invert the RGB components of the texture sample while keeping the alpha unchanged, we can subtract the RGB values from 1.0 after sampling:

```
// object colour
       vec4 material_colour = texture2D(texture, map);
       material_colour.rgb = vec3(1.0) - material_colour.rgb;
```

C2 — Alpha transformations



In the fragment shader, we perform an alpha transformation to make the dark parts of the texture transparent. This is achieved by measuring the overall darkness of the texture color using the length() function on the RGB vector.
The length() function calculates the Euclidean length of the vector, which gives us a measure of the intensity of the color.

```
// Measure darkness
        float darkness = length(material_colour.rgb);

        // Set alpha based on darkness
        if (darkness < 0.5) { // Adjust the threshold as needed
            material_colour.a = 0.0; // Fully transparent
        } else {
            material_colour.a = 1.0; // Fully opaque
        }
```

This approach allows us to selectively make the darker parts of the texture transparent, which can be useful for various visual effects.

The gl_FrontFacing test is used to discard any fragments that are facing away from the camera. This helps to eliminate artifacts such as checker/flicker effects that can occur due to depth issues.

```
 // Discard backwards-facing fragments
        if(!gl_FrontFacing) {
            discard;
        }
```

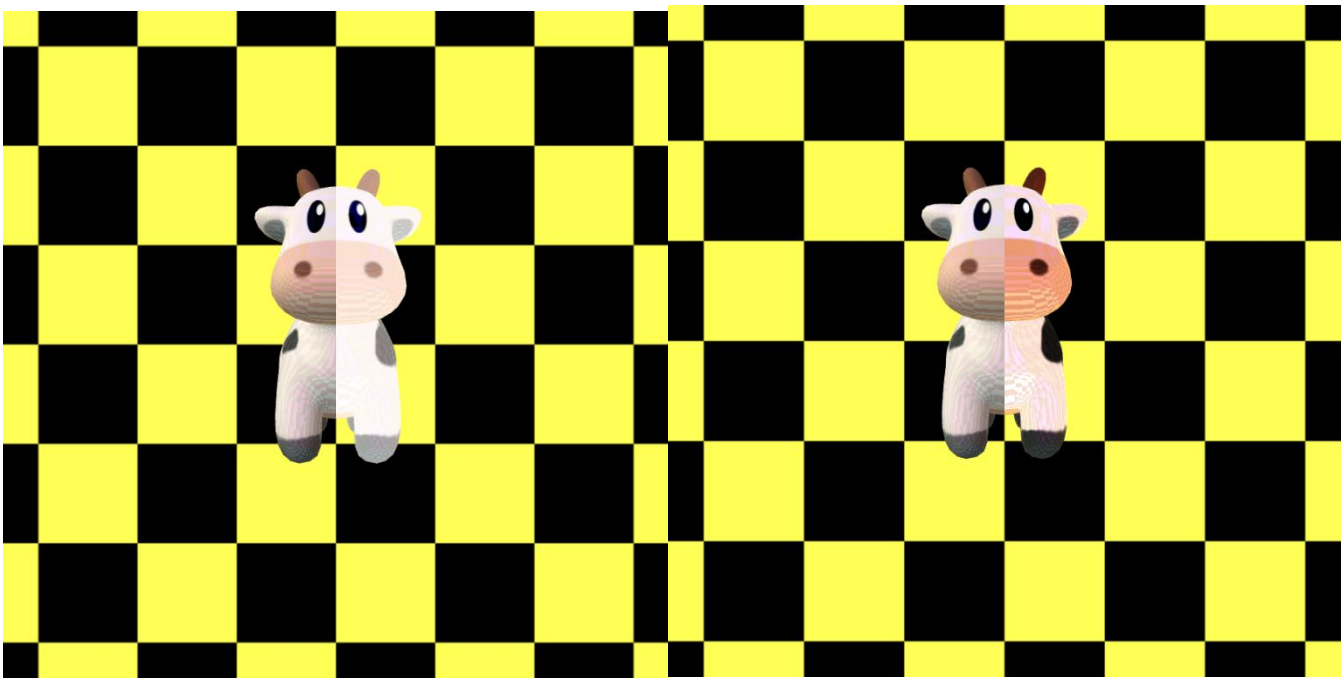
In this test:

- gl_FrontFacing is a built-in GLSL variable that is true if the fragment is part of a front-facing primitive, and false otherwise.
- If gl_FrontFacing is false, the fragment is discarded using the discard statement.

This test is necessary because:

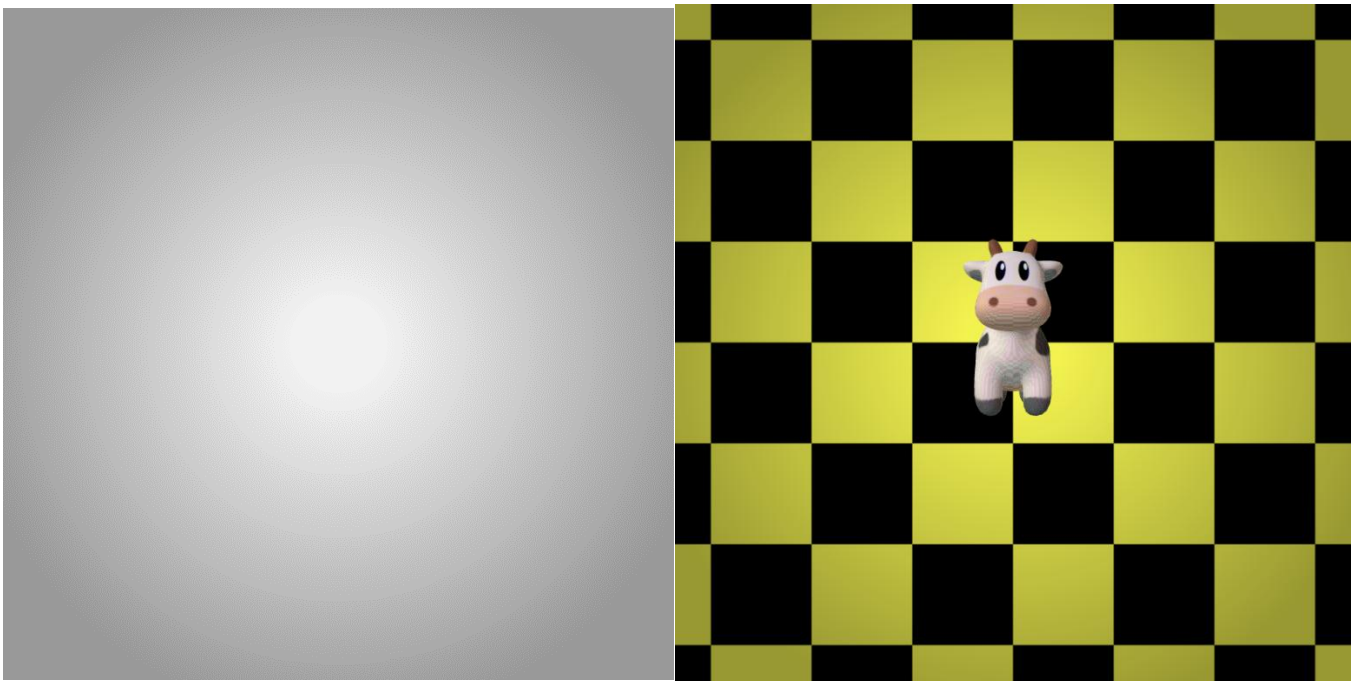
- Backwards-facing fragments can cause visual artifacts due to depth conflicts, especially when rendering complex scenes with multiple overlapping objects.
- By discarding these fragments, we ensure that only the front-facing fragments are rendered, which helps to maintain the visual integrity of the scene.

C3 — Gamma transformations



1. **Gamma Transformation Effect**:
    - **C3i.png (Gamma = 0.5)**:
        - The gamma transformation with an exponent of 0.5 brightens the colors. This is because the gamma value less than 1.0 increases the intensity of the colors.
        - **Effect on Black and White Colors**:
            - Black colors remain black since any power of zero is still zero.
            - White colors remain white since any power of one is still one.
            - Mid-tone colors become brighter, making the overall image appear lighter on the right half of the canvas.
    - **C3ii.png (Gamma = 2.0)**:
        - The gamma transformation with an exponent of 2.0 darkens the colors. This is because the gamma value greater than 1.0 decreases the intensity of the colors.
        - **Effect on Black and White Colors**:
            - Black colors remain black since any power of zero is still zero.
            - White colors remain white since any power of one is still one.
            - Mid-tone colors become darker, making the overall image appear darker on the right half of the canvas.
2. **Visual Comparison**:
    - By splitting the canvas into two halves, with the left half showing the original colors and the right half showing the gamma-transformed colors, it is easier to observe the effect of gamma correction.
    - This comparison highlights how gamma correction can be used to adjust the brightness and contrast of an image.

C4 — Vignetting



For this vignetting effect I firs created the function for it in FS code :

```
float vignette(vec2 coord, vec2 resolution) {
  // Calculate the distance of the fragment from the center of the image
  vec2 center = resolution / 2.0;
  float dist = distance(coord, center);


  float vignetteFactor = 1.0 - dist / length(resolution);
  vignetteFactor = clamp(vignetteFactor, 0.6, 0.95);
  return vignetteFactor;
}
```

And to apply the effect I added the last part of code :

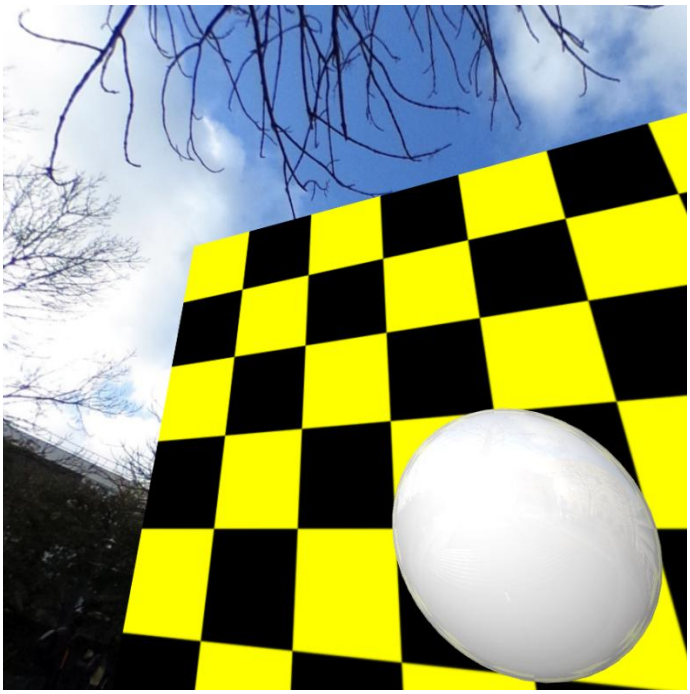
```
// vignette effect
    vec2 resolution = vec2(850,850);
    //gl_FragColor = vec4(1.0);

    // scale the final fragment rgb by vignette value
    gl_FragColor.rgb *= vignette(gl_FragCoord.xy, resolution);
```

This code section implements a vignette effect in a GLSL fragment shader. A vignette effect darkens the edges of an image while keeping the center brighter, creating a subtle fade-out effect that draws attention to the center of the scene.

The vec4(1.0) is for debugging the effect on empty scene.

D1 — Real images



I did the same steps as given in the exercise.

D2 — Reflection mapping revisited

1. **Gamma value (2.2)**:

- I used gamma value of 2.2 which I searched is the standard for display correction
- This compensates for typical display non-linearity

2. **Implementation details**:

```
gl_FragColor = gamma_transform(reflection_colour, 2.2);
```

- Applied to reflection components only
- Preserves alpha channel
- Used in both textured and reflection-only modes

3. **The reason for discrepancies**:

- Real photos vs rendered sphere differences may arise from:
    - Simplified Blinn-Phong lighting model
    - Limited dynamic range in cubemap textures
    - Monitor calibration differences
    - Real-world material complexities not captured in shader

D3 — Combined rendering

1. **Environment Mapping:**

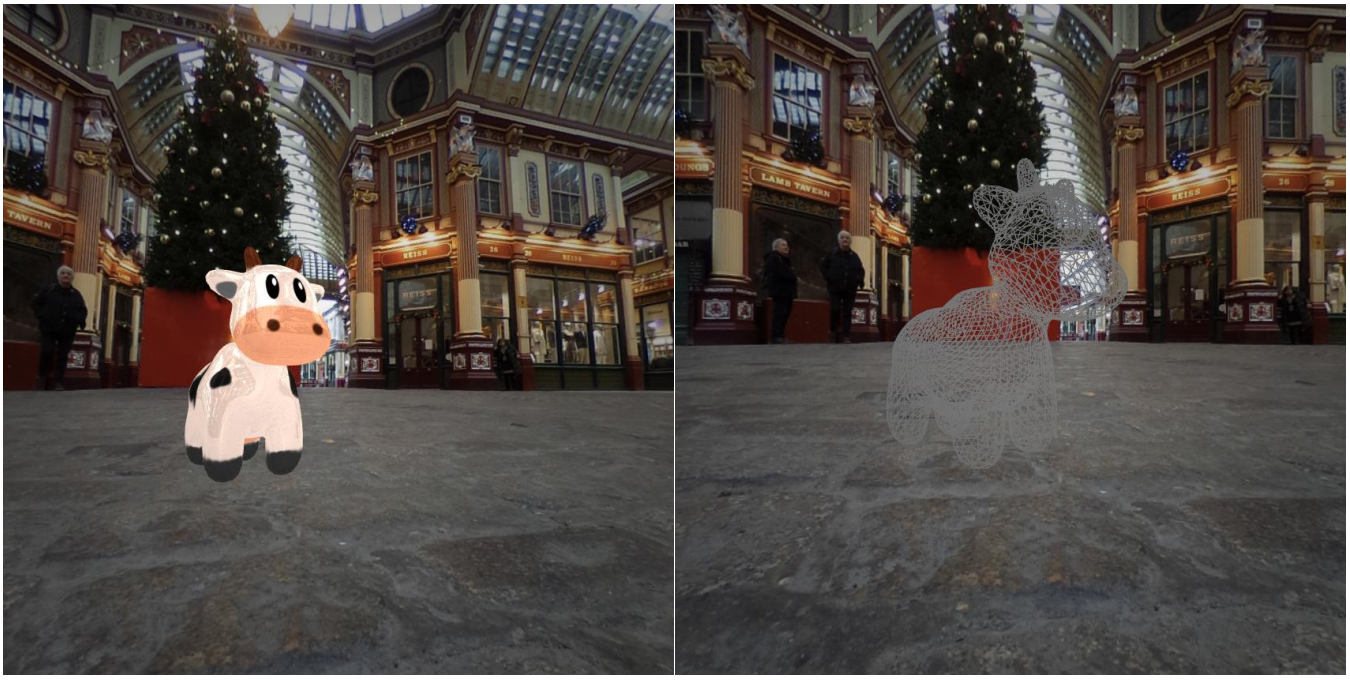
- Reflections show realistic environmental details
- Mouse drag interaction allows full 360° exploration
- Cube map faces properly aligned with seamless transitions

2. Surface Materials

- Blinn-Phong lighting model for basic illumination:
    - Ambient (0.5 weight)
    - Diffuse (0.5 weight)
    - Specular (0.01 weight)
- Reflection contribution (0.1 weight) balanced with direct lighting
- Gamma correction (2.2) improves reflection realism
- Alpha channel preserved for transparency

3. Visual Effects

- Vignette effect adds subtle darkening around edges
- Combined with reflections enhances depth perception
- Helps focus attention on central object



In this implementation, I modified the fragment shader to properly handle wireframe rendering by adding an early exit condition for wireframe mode. The key change was placing the wireframe check at the beginning of the non-skybox branch

of the shader, before any lighting or texture calculations occur. This ensures that when wireframe mode is active (wireframe_mode = true), the shader immediately returns a solid grey color (vec4(0.5, 0.5, 0.5, 1.0)) with the vignette effect applied, bypassing all complex lighting calculations, texture sampling, and gamma corrections. This creates a clean wireframe visualization that clearly shows the mesh structure without being affected by material properties or environmental reflections.

```
void main()
{
    if(render_skybox) {
        gl_FragColor = textureCube(cubemap,vec3(-d.x,d.y,d.z));
    }
    else {
        // Early wireframe check
        if(wireframe_mode) {
            gl_FragColor = vec4(0.5, 0.5, 0.5, 1.0);
            vec2 resolution = vec2(850,850);
            gl_FragColor.rgb *= vignette(gl_FragCoord.xy, resolution);
            return;
        }
```