RTG1IL Real Time Graphics – WS 2024/25
Lab Report 1

Jan Eberwein, s2410629003
Johannes Eder, s2410629004

Abstract

This project demonstrates the integration of a vignette effect as a post-processing enhancement in a deferred shading pipeline using OpenGL. The vignette effect, which darkens the edges of the screen to draw focus to the center, is implemented in a separate shader pass via a framebuffer object. The implementation includes user controls that enable toggling the effect and adjusting its intensity.

1.1 Assignment Overview

The main objective of this assignment was to implement a custom post-processing effect within an existing OpenGL deferred shading pipeline. In this project, a vignette effect was chosen due to its popularity and its ability to subtly guide the viewer's attention. Key requirements included:

- Implementing a shader-based vignette effect.
- Integrating the effect as an additional rendering pass using a framebuffer object (FBO).
- Providing GUI controls to toggle the effect and adjust parameters such as intensity.

1.2 Implementation Details

1.2.1 Post-Processing Framework

Deferred shading renders scene geometry into a G-buffer where attributes (positions, normals, albedo, etc.) are stored. The lighting pass uses these buffers and renders the lit scene to an off-screen framebuffer. This off-screen rendering (via an FBO) then serves as the input for further post-processing effects. For the vignette effect, the scene is first rendered to a post-processing FBO, and then a full-screen quad is drawn using a custom vignette shader.

Framebuffer Setup

The following code snippet shows the configuration of the post-processing FBO:

Listing 1.1: Setting up the post-processing FBO for the vignette effect

```
unsigned int ppFBO;
glGenFramebuffers(1, &ppFBO);
glBindFramebuffer(GL_FRAMEBUFFER, ppFBO);

unsigned int ppColorBuffer;
glGenTextures(1, &ppColorBuffer);
glBindTexture(GL_TEXTURE_2D, ppColorBuffer);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, SCR_WIDTH, SCR_HEIGHT, 0, GL_RGBA,
     GL_UNSIGNED_BYTE, NULL);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
     ppColorBuffer, 0);

unsigned int rboPP;
glGenRenderbuffers(1, &rboPP);
glBindRenderbuffer(GL_RENDERBUFFER, rboPP);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH24_STENCIL8, SCR_WIDTH, SCR_HEIGHT);
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_STENCIL_ATTACHMENT,
     GL_RENDERBUFFER, rboPP);

if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
    std::cout << "Postprocessing framebuffer not complete!" << std::endl;
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

1.2.2 Vignette Effect Shader

The vignette effect is implemented in the fragment shader. It works by sampling the rendered scene texture and calculating the distance of each fragment from the center of the screen. This distance is used to blend the original color with a darkened version, where the strength of the darkening is adjustable.

Vignette Fragment Shader

The following code is the complete fragment shader for the vignette effect:

Listing 1.2: Vignette Fragment Shader (vignette.fs)

```
#version 330 core
in vec2 TexCoords;
out vec4 FragColor;

uniform sampler2D screenTexture;
uniform bool vignetteOn;
uniform float vignetteStrength; // 0.0 = no effect, 1.0 = full effect

void main()
{
    vec4 color = texture(screenTexture, TexCoords);
    if(vignetteOn)
    {
        // Calculate the normalized distance from the center (0.5, 0.5)
        float dist = distance(TexCoords, vec2(0.5, 0.5));
        // Apply smoothstep to generate a gradual darkening effect from the center
    outward.
        float vignette = 1.0 - smoothstep(0.3, 0.8, dist);
        // Multiply the result to enhance the effect and clamp to [0,1].
        vignette = clamp(vignette * 1.5, 0.0, 1.0);
        // Blend the original color with the darkened color based on vignette
    strength.
        color.rgb = mix(color.rgb, color.rgb * vignette, vignetteStrength);
    }
    FragColor = color;
}
```

Vignette Vertex Shader

The vertex shader simply passes through the texture coordinates to the fragment shader:

Listing 1.3: Vignette Vertex Shader (vignette.vs)

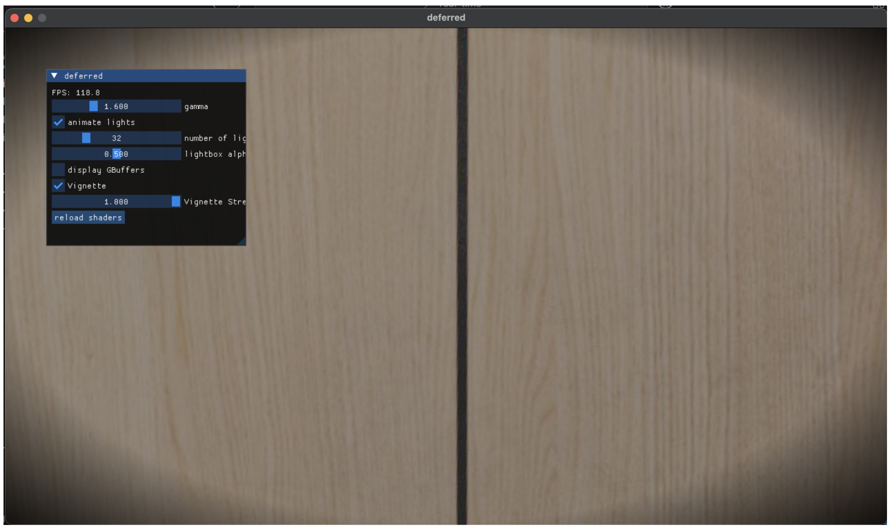
```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoords;

out vec2 TexCoords;

void main()
{
    TexCoords = aTexCoords;
    gl_Position = vec4(aPos, 1.0);
}
```

1.2.3 GUI Integration and Runtime Control

The final integration involves a post-processing pass where the output of the lighting stage (stored in the FBO) is rendered using the vignette shader. Graphical User Interface (GUI) controls allow the user to toggle the vignette effect and adjust its strength. This real-time adjustability is crucial for fine-tuning the visual appearance without recompiling the code.

1.3 Results

The implementation yielded a working scene where the vignette effect can be toggled on and off and its intensity adjusted via a slider. The effect darkens the screen edges in a smooth, customizable manner, drawing focus to the center of the image. Figure 1.1 shows a sample output of the scene with the vignette effect applied.



```
▼ deferred
FPS: 118.8
1.600          gamma
✓ animate lights
32             number of lig
0.500          lightbox alph
  display GBuffers
✓ Vignette
1.000          Vignette Stre
reload shaders
```

Figure 1.1: Screenshot of the OpenGL scene with the vignette post-processing effect applied.

1.4 Conclusion

This project successfully demonstrates the integration of a custom vignette effect into an OpenGL deferred shading pipeline. By using a dedicated post-processing pass and a flexible shader-based implementation, the effect was added without altering the scene's geometry or lighting calculations.