Texel Density Balancer – ROI Variant (Day 2.5)

Rameel Malik

August 12, 2025

1 Introduction

This document explains the **ROI (Region of Interest) Variant** of our Texel Density Balancer prototype. The goal of this variant is to keep a circular region of the screen sharp (full texel density) while reducing texel density in the periphery to save performance. This mimics foveated rendering techniques.

2 Concept Overview

In Day 2, we applied a uniform Level-of-Detail (LOD) bias across the entire screen. In Day 2.5, we:

- Track a region of interest (ROI) using the mouse cursor or fixed center.
- Use two texture samplers: one sharp (LOD bias = 0) and one biased (LOD bias > 0).
- Blend between sharp and biased textures based on distance from the ROI center using a smooth transition (feather).

3 Key Changes from Day 2

3.1 Two Texture Samplers

We now create two OpenGL samplers:

```
// Sharp sampler
glGenSamplers(1, &sampHi);
glSamplerParameteri(sampHi, GL_TEXTURE_MIN_FILTER,
    GL_LINEAR_MIPMAP_LINEAR);
glSamplerParameteri(sampHi, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    ;
glSamplerParameterf(sampHi, GL_TEXTURE_LOD_BIAS, 0.0f);

// Biased sampler (periphery)
glGenSamplers(1, &sampLo);
glSamplerParameteri(sampLo, GL_TEXTURE_MIN_FILTER,
    GL_LINEAR_MIPMAP_LINEAR);
glSamplerParameteri(sampLo, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    ;
glSamplerParameterf(sampLo, GL_TEXTURE_LOD_BIAS, lodBias);
```

The `sampHi` sampler is used inside the ROI for sharp rendering. The `sampLo` sampler applies a positive LOD bias to fetch lower-resolution mipmap levels in the periphery.

3.2 Fragment Shader ROI Blend

The fragment shader receives both samplers and blends them based on distance from the ROI center:

```
float d = distance(gl_FragCoord.xy, roiCenter);
// w = 0 inside sharp zone, 1 outside sharp+feather
float w = smoothstep(roiRadius, roiRadius + roiFeather, d);

vec4 sharp = texture(texHi, vUV);
vec4 soft  = texture(texLo, vUV);
fragColor = mix(sharp, soft, w);
```

Here:

- `roiRadius` = fully sharp distance from center.
- `roiFeather` = transition width.
- `w` = blend weight (smoothstep ensures a smooth gradient).

3.3 Mouse Tracking and ROI Position

We track mouse position with:

```
double mx=0.0, my=0.0;
glfwGetCursorPos(window, &mx, &my);
```

```
roiX = static_cast<float>(mx);
roiY = static_cast<float>(fbH - my); // Flip Y
```

GLFW's `glfwGetCursorPos` gives coordinates with the origin at the **top-left**. OpenGL's `gl_FragCoord` uses the **bottom-left** as origin. We fix this mismatch by flipping Y:

$$roiY = framebufferHeight - mouseY$$

3.4 Y-Flip Explanation

Without flipping Y, the ROI would follow the mouse **vertically inverted**. Flipping ensures the fragment shader's coordinate system matches the input from GLFW:

- Top-left (GLFW) → (0,0)
- Bottom-left (OpenGL) → (0,0)

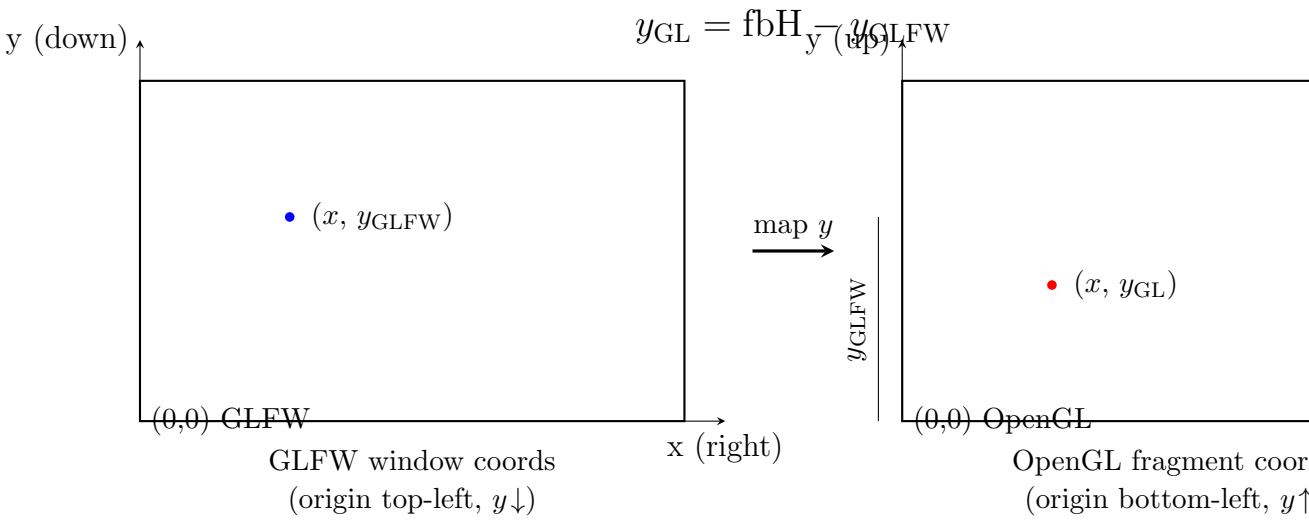

Figure 1: Coordinate origin mismatch and Y-flip: GLFW gives mouse Y from the top-left; the fragment shader uses bottom-left. Convert with $y_{GL} = fbH - y_{GLFW}$.

4 Pipeline Summary

1. Load texture and generate mipmaps.

2. Create two samplers: sharp and biased.
3. Pass ROI parameters (center, radius, feather) as uniforms to shader.
4. In fragment shader:
   (a) Compute pixel distance from ROI center.
   (b) Compute blend factor with `smoothstep`.
   (c) Sample both textures (different LOD bias) and blend.

5 Behavior vs. Day 2

- Day 2: uniform bias over whole screen.
- Day 2.5: bias applied only outside ROI, sharpness preserved in center.
- Limitation: Blur stops increasing after highest mip level (hardware limit).

6 Expected Output

- A sharp region follows the mouse cursor.
- Outside this region, textures are progressively more blurred depending on `lodBias`.
- Increasing `roiFeather` makes the transition softer.

7 Conclusion

This ROI variant demonstrates how foveated rendering concepts can be implemented with minimal code changes in OpenGL by leveraging:

- Multiple texture samplers with different LOD biases.
- Distance-based blending in the fragment shader.
- Mouse tracking with coordinate system correction.