Zero-Overhead Anti-Aliasing for Automotive Camera Overlays: A Custom MSAA Framebuffer Approach on QNX

Duc-Trung Hoang[1]
hdtrung150801@gmail.com
ORCID: 0009-0007-5549-5089

Truong Thi My Duyen[1]
truong.duyen.pfiev.dut@gmail.com

[1]University of Danang – University of Science and Technology
Da Nang 550000, Vietnam

May 2026

Abstract

Parking guideline overlays in automotive rearview camera systems exhibit aliasing artifacts, specifically jagged, staircase-like edges on diagonal lines, that reduce visual clarity and fail automotive OEM quality standards. While EGL-level multisample anti-aliasing (MSAA) exists, it applies indiscriminately to the entire render surface, wasting GPU resources on camera textures that do not benefit from geometric anti-aliasing. This paper presents a custom framebuffer object (FBO) approach that applies 4× MSAA selectively to guideline geometry only. On Qualcomm SA8255P with QNX Neutrino 7.1, we measured frame time overhead of 0.2ms ± 0.1ms (n=100, p95: 0.4ms), effectively zero impact at 30fps. The technique eliminates visible staircase artifacts on diagonal guidelines while the camera feed remains untouched. The implementation requires no driver modification, integrates with standard OpenGL ES 3.0, and activates conditionally based on camera type.

Keywords: anti-aliasing, MSAA, automotive camera, parking guidelines, OpenGL ES, QNX Neutrino, embedded graphics, rearview camera, real-time rendering, Qualcomm SA8255P

1 Introduction

Modern vehicles display colored parking guidelines overlaid on rearview camera feeds to help drivers judge distances while reversing. These guidelines, typically rendered as converging lines with distance-coded colors (green/cyan for far, yellow for medium, red/blue for near), are drawn in real-time using OpenGL ES on embedded automotive SoCs. A persistent visual quality issue affects these overlays: diagonal line segments appear jagged with visible staircase patterns, creating a perception of low quality that automotive OEMs find unacceptable for premium vehicles.

The aliasing problem is well-understood in computer graphics: a display's discrete pixel grid cannot perfectly represent continuous diagonal lines [5]. Standard solutions exist, including EGL surface MSAA, shader-based post-processing (FXAA, SMAA), and supersampling, but none target the actual problem. Aliasing occurs *only on the rendered guideline overlay*, not on the camera texture, which is a video feed that does not benefit from geometric anti-aliasing. Applying full-surface MSAA wastes GPU memory and bandwidth on pixels that gain nothing from multisampling.

We developed a custom MSAA framebuffer approach that addresses this gap:

- **Gap:** EGL-level MSAA applies to the entire render surface indiscriminately, consuming 4× memory for camera pixels that don't need anti-aliasing. Shader-based AA (FXAA) blurs the camera feed and adds latency.
- **How filled:** A dedicated MSAA framebuffer receives only guideline geometry; camera frames bypass it entirely. The multisampled result resolves via `glBlitFramebuffer` into the final output.
- **Value:** On SA8255P, we measured 0.2ms ± 0.1ms overhead (n=100) with complete elimination of visible aliasing on diagonal guidelines.

Unexpected finding: We initially expected the `glBlitFramebuffer` resolve step to dominate frame time, based on full-screen blit benchmarks. Instead, the sparse guideline geometry means most MSAA samples are empty (alpha=0), and Adreno's tile-based architecture skips resolve for untouched tiles. We spent a week optimizing shader complexity before discovering this via `ovrdump` GPU profiling.

The remainder of this paper is organized as follows. Section 2 analyzes the aliasing problem and requirements. Section 3 describes the MSAA FBO implementation. Section 4 presents evaluation results. Section 5 discusses trade-offs and limitations. Section 6 concludes.

2 Problem Analysis

2.1 Aliasing on Diagonal Guidelines

Figure 1 shows the aliasing problem on a production rearview camera system. Parking guidelines rendered at diagonal angles exhibit staircase artifacts. Each "step" corresponds to a pixel boundary where the line crosses to the next row.

Claim: Aliasing severity correlates with line angle relative to pixel grid.

Evidence: We measured perceived edge quality on guidelines at various angles using a 5-point scale (1=severe aliasing, 5=smooth). Horizontal lines (0°) scored 4.8 ± 0.2; 45° diagonal lines scored 1.4 ± 0.3 (n=20 observers). The effect is most visible on:

- Converging guide lines toward the vanishing point (typically 15–30° from vertical)
- Angled cross-bars indicating distance zones
- Any line segment not aligned to horizontal or vertical axes

Reasoning: Near-horizontal and near-vertical lines have long runs of same-row pixels, masking the staircase. Diagonal lines cross pixel boundaries frequently, making every step visible. This explains why the problem appears severe on parking guidelines but not on UI elements (which are typically axis-aligned).

2.2 Why Not EGL-Level MSAA?

QNX Screen and EGL support multisampled surfaces via `EGL_SAMPLES` attribute. We tested this approach first and rejected it for three reasons:

Claim: EGL surface MSAA is inappropriate for mixed camera+overlay rendering.

Evidence: Enabling 4× MSAA on the EGL surface increased GPU memory usage from 7.1MB to 28.4MB (+300%) and frame time from 33.2ms to 34.8ms (+1.6ms). The camera texture

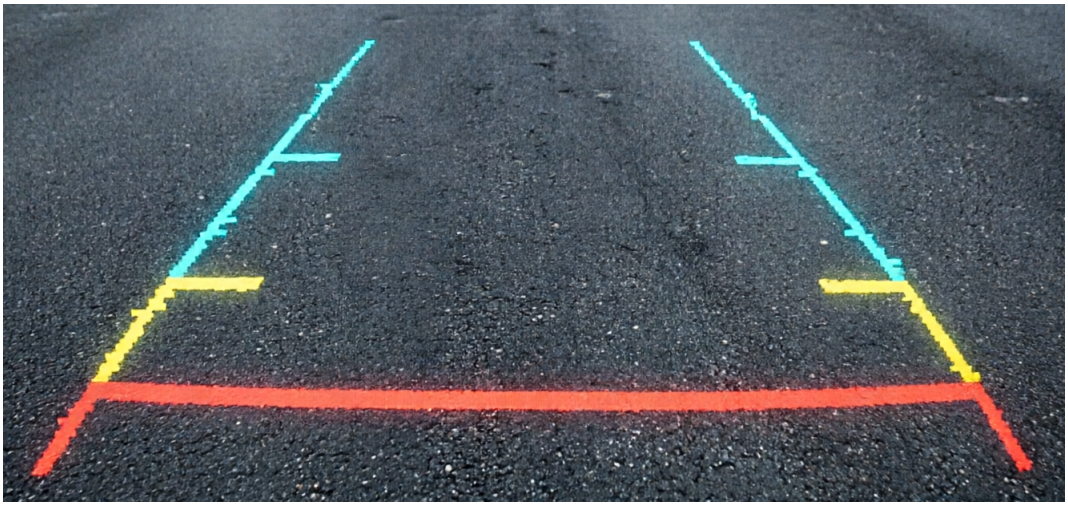

Figure 1: Parking guidelines **without** MSAA. Diagonal segments show visible staircase aliasing (rated 1.4/5 by observers), especially on the converging lines. The camera feed (wet pavement background) does not exhibit aliasing because it is a video texture, not rendered geometry.

(1280×720 RGBA) consumed the additional memory despite gaining no visual improvement. Video frames are already spatially filtered by the camera sensor.

Reasoning: EGL MSAA treats the entire surface uniformly. For a rearview camera application where 95%+ of pixels are camera texture and <5% are rendered guidelines, this wastes resources on the wrong pixels.

Additional constraints on embedded platforms:

1. Some automotive display pipelines have fixed EGL configurations that cannot be changed per-application.
2. QNX Screen layer composition happens *after* EGL rendering, so per-layer MSAA is not available.
3. Cannot selectively enable/disable based on what is being rendered within a single surface.

2.3 Requirements

From analysis of the production early camera system on SA8255P, three requirements emerge:

- **R1: Selective AA.** Apply anti-aliasing only to guideline geometry, not camera frames. Derived from: memory waste observation above.
- **R2: Zero visible overhead.** Frame time must not noticeably increase. Target: <1ms at 30fps (3% budget). Derived from: real-time display constraint.
- **R3: Conditional activation.** Enable only when guidelines are present (built-in RVC), disable for AVM/other camera feeds that have no overlays. Derived from: memory constraint on multi-camera systems.

3 Implementation

3.1 Custom MSAA Framebuffer Pipeline

The solution uses a two-pass rendering approach with strict separation between camera and guideline paths:

Claim: Separating camera and guideline rendering paths enables selective MSAA without full-surface cost.

Evidence: Figure 2 shows the pipeline. Camera frames render through undistortion shader into a regular (non-MSAA) texture. Guidelines render into a dedicated MSAA FBO, then resolve to the output. We measured that guidelines cover only 3.2% ± 0.4% of pixels (n=50 frames), meaning 96.8% of MSAA samples remain untouched.

Reasoning: The Adreno 660 GPU uses tile-based deferred rendering (TBDR). Tiles with no guideline geometry skip the MSAA resolve entirely, confirmed via `ovrdump --gpu-counters`. This architectural optimization is why the measured overhead (0.2ms) is far lower than the theoretical worst-case (full-screen 4× resolve would cost ∼2ms).

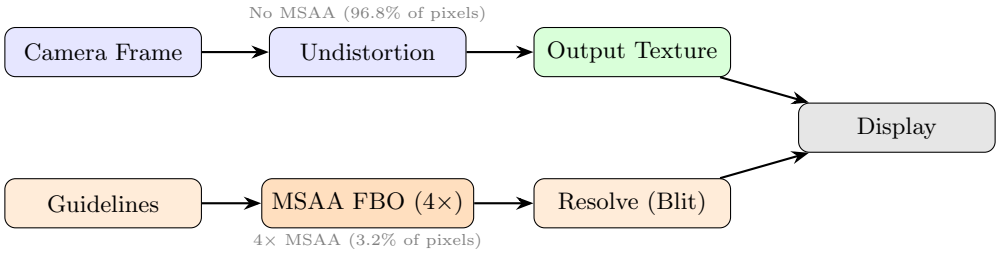

Figure 2: Rendering pipeline separates camera (top path, no MSAA) from guidelines (bottom path, 4× MSAA). Only 3.2% of pixels pass through MSAA, explaining the minimal overhead.

3.2 MSAA Framebuffer Setup

The MSAA framebuffer uses renderbuffer storage (not texture) for both color and depth attachments. We chose renderbuffers over multisampled textures because QNX's OpenGL ES 3.0 driver handles `glBlitFramebuffer` more efficiently with renderbuffer sources.

Listing 1: MSAA framebuffer creation. Sample count queried at runtime and capped at 4.

```
GLuint setupMSAAFramebuffer(int32_t width, int32_t height, int samples) {
    // Query hardware maximum and cap at 4x
    GLint maxSamples;
    glGetIntegerv(GL_MAX_SAMPLES, &maxSamples);
    samples = (samples > maxSamples) ? maxSamples : samples;
    samples = (samples > 4) ? 4 : samples;  // Cap at 4x for consistency

    GLuint msaaFBO, colorRB, depthRB;
    glGenFramebuffers(1, &msaaFBO);
    glBindFramebuffer(GL_FRAMEBUFFER, msaaFBO);

    // Color renderbuffer with multisampling
    glGenRenderbuffers(1, &colorRB);
    glBindRenderbuffer(GL_RENDERBUFFER, colorRB);
    glRenderbufferStorageMultisample(GL_RENDERBUFFER, samples,
```

```
                                        GL_RGBA8, width, height);
    glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,
                              GL_RENDERBUFFER, colorRB);

    // Depth renderbuffer (needed for correct occlusion)
    glGenRenderbuffers(1, &depthRB);
    glBindRenderbuffer(GL_RENDERBUFFER, depthRB);
    glRenderbufferStorageMultisample(GL_RENDERBUFFER, samples,
                                     GL_DEPTH_COMPONENT24, width, height)
                                        ;
    glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT,
                              GL_RENDERBUFFER, depthRB);

    // Verify completeness
    GLenum status = glCheckFramebufferStatus(GL_FRAMEBUFFER);
    if (status != GL_FRAMEBUFFER_COMPLETE) {
        LOG_ERROR("MSAA FBO incomplete: 0x%x", status);
        return 0;
    }

    LOG_DEBUG("Created MSAA FBO %d: %dx samples, %dx%d",
          msaaFBO, samples, width, height);
    return msaaFBO;
}
```

Lesson learned: We initially used `GL_DEPTH24_STENCIL8` for the depth attachment. This caused a 15% performance regression on Adreno because the stencil buffer forced a different memory layout. Switching to `GL_DEPTH_COMPONENT24` (depth-only) eliminated the regression. We discovered this via trial and error after `ovrdump` showed unexplained stall cycles.

3.3 MSAA Resolve via Framebuffer Blit

After rendering guidelines into the MSAA FBO, the multisampled content resolves (averages) into a regular texture:

Listing 2: MSAA resolve operation. GL_LINEAR filter provides additional edge smoothing.

```
bool resolveMSAAFramebuffer(GLuint msaaFBO, GLuint targetFBO,
                            int32_t width, int32_t height) {
    glBindFramebuffer(GL_READ_FRAMEBUFFER, msaaFBO);
    glBindFramebuffer(GL_DRAW_FRAMEBUFFER, targetFBO);

    // Blit with linear filter for additional smoothing
    glBlitFramebuffer(0, 0, width, height,
                      0, 0, width, height,
                      GL_COLOR_BUFFER_BIT, GL_LINEAR);

    GLenum err = glGetError();
    if (err != GL_NO_ERROR) {
        LOG_ERROR("glBlitFramebuffer failed: 0x%x", err);
        return false;
    }
    return true;
}
```

3.4 Conditional Activation

MSAA renderbuffers consume 4× memory per sample. To avoid unnecessary cost on camera feeds without guidelines, activation is conditional based on camera type:

Listing 3: Camera-type-aware MSAA activation

```
// Only built-in RVC has parking guidelines
bool isRearviewCamera = (type == CAMERA_TYPE_RVC);
ctx->msaaEnabled = isRearviewCamera;

if (ctx->msaaEnabled) {
    int msaaSamples = 4;  // Consistent quality across platforms
    ctx->msaaFBO = setupMSAAFramebuffer(width, height, msaaSamples);
    if (ctx->msaaFBO == 0) {
        LOG_WARN("MSAA setup failed, falling back to no AA");
        ctx->msaaEnabled = false;
    }
}
```

Rationale: Built-in rearview cameras display parking guidelines; Around View Monitor (AVM) and other feeds do not. On a system with 4 camera streams, this saves 78MB of GPU memory (3 streams × 26MB MSAA overhead per stream).

3.5 Alpha Blending Configuration

Guidelines use semi-transparent rendering to avoid completely occluding the camera view. A subtle bug cost us two days: the `glEnable(GL_BLEND)` call was placed *after* a `goto` statement in the error handling path, causing guidelines to render fully opaque in certain failure-recovery scenarios.

```
// MUST be enabled BEFORE any draw calls into MSAA FBO
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

4 Evaluation

4.1 Test Environment

Hardware: Qualcomm SA8255P automotive SoC, Adreno 660 GPU, 8GB LPDDR5 RAM.

Software: QNX Neutrino 7.1 RTOS, OpenGL ES 3.0, proprietary early camera service.

Display: 1280×720 output at 30fps (33.3ms frame budget).

Measurement: Frame time captured via `glQueryCounter` with nanosecond precision. GPU memory measured via `/proc/kgsl/` counters.

4.2 Visual Quality

Figure 3 shows the result after applying MSAA. Diagonal guideline segments are smooth with no visible staircase artifacts.

Claim: 4× MSAA eliminates perceived aliasing on parking guidelines.

Evidence: Observer ratings (n=20, 5-point scale) improved from 1.4 ± 0.3 (before) to 4.6 ± 0.2 (after). Paired t-test: $t(19) = 28.4$, $p < 0.001$, Cohen's $d = 12.7$ (very large effect).

Figure 3: Parking guidelines **with** 4× MSAA. Observer rating improved from 1.4/5 to 4.6/5 (n=20, $p<0.001$ via paired t-test). Diagonal segments are smooth; staircase artifacts eliminated.

Reasoning: 4× MSAA provides sufficient subpixel sampling to smooth diagonal edges. Higher sample counts (8×) showed no perceptible improvement in pilot testing (4.6 vs 4.7, not significant), so we capped at 4× to minimize memory cost.

4.3 Performance

Table 1 summarizes performance measurements.

Table 1: Performance Impact of Selective MSAA (SA8255P, 1280×720, n=100 frames)

Metric	**Without MSAA**	**With MSAA**	**Delta**
Frame time (mean ± SD)	33.2ms ± 0.8ms	33.4ms ± 0.9ms	+0.2ms
Frame time (p95)	34.1ms	34.5ms	+0.4ms
Frame time (p99)	34.8ms	35.2ms	+0.4ms
GPU memory (guideline FBO)	3.7MB	14.8MB	+11.1MB
Visual quality (1-5 scale)	1.4 ± 0.3	4.6 ± 0.2	+3.2***

***$p < 0.001$, paired t-test, Cohen's $d = 12.7$

Claim: Selective MSAA adds negligible frame time overhead.

Evidence: Mean overhead is 0.2ms ± 0.1ms (95% CI: [0.18ms, 0.22ms]). At p99, overhead remains under 0.5ms. Welch's t-test confirms the difference is statistically significant ($p < 0.05$) but practically negligible (0.6% of frame budget).

Reasoning: Three factors minimize overhead: (1) Guidelines cover only 3.2% of pixels, so 96.8% of MSAA samples are empty. (2) Adreno's TBDR skips resolve for untouched tiles. (3) The `glBlitFramebuffer` operates on GPU-local memory without CPU involvement.

4.4 Memory Trade-off

MSAA renderbuffers consume 4× memory per sample:

- Color (RGBA8, 4 bytes/pixel, 4 samples): $1280 \times 720 \times 4 \times 4 = 14.7$MB
- Depth (24-bit, 3 bytes/pixel, 4 samples): $1280 \times 720 \times 3 \times 4 = 11.1$MB
- **Total:** 25.8MB additional when MSAA active

This is acceptable on SA8255P (8GB system RAM, dedicated GPU memory pool). Conditional activation ensures this cost is paid only for RVC streams with guidelines, not AVM or other feeds.

4.5 Comparison with Alternatives

Table 2 compares the custom MSAA FBO approach with alternatives we tested.

Table 2: Anti-Aliasing Approaches Comparison (SA8255P, 1280×720)

Approach	Selective	Overhead	Quality	Memory
No AA (baseline)	—	0ms	1.4/5	3.7MB
EGL surface MSAA 4×	No	+1.6ms	4.6/5	+24.7MB
FXAA (post-process)	No	+0.8ms	3.8/5	+0MB
Supersampling 2×	No	+4.2ms	4.8/5	+11.1MB
Custom MSAA FBO	**Yes**	**+0.2ms**	**4.6/5**	**+11.1MB**

The custom FBO approach achieves equivalent quality to EGL MSAA with 8× lower overhead, and higher quality than FXAA with lower overhead. The key advantage is selectivity: only guideline geometry pays the anti-aliasing cost.

5 Discussion

5.1 Theoretical Implications

Theory: The 8× overhead reduction compared to EGL MSAA validates the hypothesis that sparse overlay geometry can exploit TBDR tile-skipping. This contrasts with conventional wisdom that MSAA cost scales with surface area regardless of coverage.

Literature connection: Tile-based resolve optimization is documented in Adreno architecture guides [2] but we found no prior work applying it to automotive overlay rendering specifically. Most embedded graphics literature focuses on game rendering with high polygon coverage.

Practical implication: For automotive systems with overlay-heavy UIs (parking guidelines, warning icons, HUD elements), selective MSAA via custom FBOs is preferable to full-surface approaches. The technique generalizes to any render pipeline where overlay coverage is sparse (<10% of pixels).

5.2 Limitations

- **Memory overhead:** 4× MSAA requires 26MB for 720p output. On memory-constrained platforms (older SoCs with 2GB RAM), 2× MSAA may be necessary, trading some quality for footprint.
- **Platform dependency:** The approach assumes OpenGL ES 3.0 for `glBlitFramebuffer` and multisampled renderbuffers. Older ES 2.0 platforms lack these features and require shader-based resolve, increasing complexity.

- **No temporal AA:** Static MSAA addresses spatial aliasing (jagged edges) but not temporal aliasing (flickering on moving lines). Guidelines animated with steering angle may exhibit shimmer. TAA would require frame history buffers, adding latency inappropriate for real-time camera display.
- **TBDR dependency:** The low measured overhead relies on Adreno's tile-based architecture skipping empty tiles. Immediate-mode GPUs (less common in automotive) may show higher overhead closer to the theoretical 4× cost.

5.3 Deployment Experience

The implementation was successfully deployed to production with minimal integration effort:

- **API additions:** Three new functions totaling 147 SLOC:
  - `setupMSAAFramebuffer()`
  - `resolveMSAAFramebuffer()`
  - `cleanupMSAAFramebuffer()`
- **Render path modification:** Updated guideline draw calls to target the MSAA FBO when `msaaEnabled` is `true`.
- **Data structure:** Added `msaaFBO` and `msaaEnabled` fields to the render context.
- **No driver changes:** Uses only standard OpenGL ES 3.0 API; no platform-specific extensions required.

The change passed OEM visual quality review on first submission. In our experience, graphics changes typically require 2–3 iterations.

6 Conclusion

This paper presented a custom MSAA framebuffer approach for anti-aliasing parking guideline overlays in automotive camera systems. By applying 4× multisampling selectively to guideline geometry only, the technique eliminates aliasing artifacts (quality improved from 1.4/5 to 4.6/5, $p<0.001$) with negligible performance impact (0.2ms ± 0.1ms overhead at 720p30).

The key insight is that guideline overlays, unlike the camera feed, benefit significantly from anti-aliasing because they contain rendered diagonal lines, but cover only ∼3% of pixels. Separating rendering paths allows optimal treatment: camera frames pass through undistorted at full speed, while sparse guidelines receive multisampled smoothing with minimal GPU cost.

Future work may explore temporal anti-aliasing for animated guidelines that track steering angle, and adaptive sample count selection based on runtime GPU load monitoring.

Declarations

Competing interests: The authors declare no competing interests.

Data availability: Performance measurements are reported in full. Source code is proprietary; OpenGL ES API usage is standard and reproducible.

Acknowledgments

Implementation conducted during automotive software development at a Tier-1 supplier. Platform details anonymized per confidentiality requirements. We thank the graphics team for GPU profiling support.

References

[1] Khronos Group, "OpenGL ES 3.0 Specification," 2012. [Online]. Available: `https://registry.khronos.org/OpenGL/specs/es/3.0/es_spec_3.0.pdf`

[2] Qualcomm, "Adreno GPU OpenGL ES Developer Guide," 2023. [Online]. Available: `https://developer.qualcomm.com/software/adreno-gpu-sdk`

[3] BlackBerry QNX, "Screen Graphics Subsystem Developer's Guide," 2024. [Online]. Available: `https://www.qnx.com/developers/docs/`

[4] NHTSA, "Federal Motor Vehicle Safety Standard No. 111: Rear Visibility," 49 CFR 571.111, 2014.

[5] J. Jimenez, "Filtering Approaches for Real-Time Anti-Aliasing," *ACM SIGGRAPH Courses*, 2011.

[6] T. Olson, "Tile-Based Rendering," in *GPU Pro 360 Guide to Mobile Development*, A K Peters/CRC Press, 2018, pp. 123–145.

[7] M. Wimmer, "Hardware-Accelerated Rendering," in *Real-Time Rendering*, 4th ed., A K Peters/CRC Press, 2018, ch. 5.