GPU-Aware Game Architecture

Validating Patterns for Temporal Upscaling Compatibility

Project Plan Presentation

Owen Newberry | ASE Capstone

Problem Domain

The Challenge

Modern GPUs have AI-focused hardware (DLSS, FSR, frame generation), but **most developers don't know how to write code that actually works well with it**.

Student and indie projects constantly break temporal reconstruction:

- Frame-rate-coupled game logic
- Inconsistent motion vectors
- Multiple systems fighting over transforms
- Frame-based instead of time-based effects

Result: Ghosting, smearing, stuttering, frame-time spikes

The Solution

Research Paper: How GPU advancements (especially Blackwell) are reshaping software architecture in game development

Code Project: Apply research findings by implementing a prototype using validated upscaler-compatible patterns

Research Foundation

Scholarly Question

"How have recent advancements in computer hardware, particularly GPUs and their integration of AI-focused architectures, reshaped the software engineering principles, design patterns, and development workflows of modern video games?"

Research Areas

GPU Evolution

- Blackwell vs. Ada Lovelace architecture
- Transition from graphics-only to AI processors
- Neural rendering and frame generation

Case Studies

- Unreal Engine 5 (Lumen, Nanite)
- Unity's ECS/DOTS framework
- DLSS/FSR technical implementation

Pattern Analysis

- What changed in development workflows?

Feature 1: Fixed-Timestep Simulation

Purpose: Decouple simulation from rendering for deterministic behavior

- Separate simulation tick from render frame
- Produce deterministic simulation output
- Interpolate visual state between ticks
- Validate motion vector stability

Feature 2: Workload Budgeting

Purpose: Prevent frame-time spikes from AI/physics computation

- Implement per-frame AI time budgets
- Time-slice expensive AI operations
- Prioritize AI tasks by importance
- Defer non-critical work under load
- Log budget usage and overruns
- Measure frame-time stability improvement

Feature 3: Temporal Coherence

Purpose: Maintain consistent motion data for upscaling

- Implement single-writer motion authority
- Keep motion vectors coherent across frames
- Convert effects to time-based updates
- Produce deterministic visual output

Feature 4: Performance Measurement

Purpose: Collect quantitative data for validation

- Collect frame-time metrics
- Measure CPU/GPU utilization
- Track memory usage and allocation patterns
- Track 1% low and 0.1% low FPS
- Export performance data for analysis
- Generate automated performance reports

Feature 5: DLSS/FSR Integration

Purpose: Test patterns with actual upscaling technology

- Integrate DLSS and/or FSR
- Test all patterns with upscaling enabled
- Measure upscaling performance impact
- Validate visual quality and temporal stability

Feature 6: Game Prototype

Purpose: Realistic stress-test context

- Create confined test environment
- Implement player movement and interaction
- Add AI entity with variable workload
- Include environmental effects for testing

Development Roadmap

Sprint 1 (4 weeks) — Research Paper

- Week 1: GPU evolution literature review + start writing
- Week 2: Case studies (UE5, Unity, DLSS/FSR) + methodology
- Week 3: Blackwell deep dive + findings/analysis
- Week 4: Complete writing, revision, finalization

Concurrent: Engine selection, installation, planning

Sprint 2 (6 weeks) — Code Implementation

Phase 1: Core Patterns (Weeks 1-3)

- Implement fixed-timestep simulation
- Add workload budgeting system
- Build temporal coherence framework
- Integrate performance measurement

Phase 2: DLSS/FSR Integration (Weeks 4-5)

- Enable DLSS and/or FSR
- Test patterns with upscaling active
- Measure performance metrics

Phase 3: Validation & Documentation (Week 6)