Using Software Design Patterns for Temporal Upscaling Compatibility

Project Plan Presentation

Owen Newberry | ASE Capstone

Problem Domain

Modern GPUs have AI-focused hardware (DLSS, FSR, frame generation), but most games don't take full advantage of these tools.

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 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?

Features and Reqs. Overview

Number of features: 4

Number of requirements: 18

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: 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

Development Roadmap

Sprint 1 (4 weeks)

- 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 (Weeks 2-4):

- Engine selection and installation
- Set up project structure and version control
- Begin fixed-timestep simulation loop
- Initial interpolation system prototype

Sprint 2 (6 weeks)

Week 1: Fixed-Timestep Completion

- Complete fixed-timestep simulation loop
- Finalize simulation/render separation
- Polish interpolation system for visual state
- Validate motion vector stability

Week 2: Workload Budgeting System

- Implement per-frame time budget manager
- Create task queue with priority system
- Add time-slicing for expensive operations
- Build budget logging and monitoring

Sprint 2 (continued)

Week 3: Temporal Coherence & Motion Vectors

- Implement single-writer motion authority pattern
- Create motion vector generation system
- Convert test effects to time-based updates
- Validate motion vector consistency across frames

Week 4: System Integration

- Integrate fixed-timestep, budgeting, and temporal systems
- Create test scenes for pattern validation
- Build baseline performance metrics
- Document API and usage patterns

Sprint 2 (continued)

Week 5: DLSS/FSR Integration

- Integrate DLSS and/or FSR into prototype
- Configure upscaling pipeline settings
- Test all patterns with upscaling enabled
- Measure upscaling performance impact

Week 6: Validation & Final Testing

- Validate visual quality and temporal stability
- Test edge cases and stress scenarios
- Generate performance comparison reports
- Final documentation and sprint retrospective