



# A Game-play Architecture for Performance

Terrance Cohen  
Insomniac Games

7/6/2007

# Outline

- Preface
- Case study: Resistance GP Optimization
  - Drones
  - Aspects experiment
- Cell Performance – High Level
- Aspects for Performance – a Proposal
- Summary
- *Moderated Discussion*

# Preface

- Qualifications?
  - Performance considerations for game-play
- Not what we're used to
  - Comments encouraged
- Looking to the future
- Many variations possible within the proposed framework \*

# Resistance GP Optimization – Problem Statement

- AI and Move were expensive
- Profile was pretty flat, ie. few hot-spots
- Deferred targeting collision checks – small improvement
- Designer contributions
  - Limited effect
  - Too labor-intensive to be a general-purpose solution

“Massive Monolithic Updates are Bad.”

# Resistance GP Optimization – the Technique

- Decided to implement Drones
  - Very cheap (by comparison) but Convincing
  - No AI or Move
- Abilities
  - Select target
  - Shoot
  - Cover (low or medium, but not full)
  - Die
  - Promote
- Drones could *not* relocate \*

# Resistance GP Optimization – Drone Planning

- Who is a Drone and when
- Desired population
  - 3 “Affinities”: *No*, Medium, High
  - Fill from High Affinity if possible
  - If desired population can’t be filled with High Affinity candidates, use Low Affinity ones
  - Never demote No Affinity candidates
- For Resistance,  $A(\text{moby}) = f(\text{distance})$ 
  - 2 distances fully specify Affinity Zones \*

# Resistance GP Optimization – Drone Planning (cont'd)

- Promotion triggers
  - When is a Drone not a Drone?
  - Damaged by player
  - Zoomed-in on
  - No Affinity (part of Planning)
  - Timed, randomized interval
- Planning updated *continuously*
- Planning data
  - 2 Affinity Zone border distances, Desired %
  - Global (level) planning
  - Optional per-squad planning override

# Resistance GP Optimization – Results

- Drones were ~ 10% of the cost
- Planning worked well
- Squad planning data used rarely
- With planning, affinity, and promotion triggers, Drones are barely noticeable
- A few scenarios still needed to be trimmed
- AI & Move fit in budget

# Proposal: Aspects for Performance

- Idea of Aspects taken from Aspect Oriented Programming
  - Help programmers in the separation of concerns:
  - Breaking down a program into distinct parts that overlap in functionality as little as possible
- Consider the high-level elements of Cell performance

# Cell Performance - High Level

- Weakness
  - High penalties for cache miss and mispredicted branches
- Strength
  - Parallelism!
- Approach:
  - Separate concerns, run them in parallel
  - Goals match AOP!
- These factors are likely to be more prominent in future hardware

# Proposal: Aspects for Performance

- Systems own their data (instead of mobs)
- When a System updates, it tears through its data
- Systems could work in parallel with other systems
- Tighter loops with fewer conditionals and less branching
  - No need to test if a Game Object *has* a particular type of data to update
- Context switches are always expensive!
- Systems operating on local data can operate more efficiently

# Proposal: Aspects for Performance

- Infrastructure for Drones exercise in AOP
- We've been heading in that direction!
  - Components
  - Physics Instances
  - Update Classes (to an extent)
  - Anim Manager
- Tools have moved toward Assets
  - Lends itself well to Aspects

# Proposal: Aspects for Performance

- Ultimately, GameObject is just a container
  - The container is no longer necessary!
- Game Objects would exist only as IDs on Aspect instances
- Query API:
  - `Aspect* GetAspect(AspectType, GameObject ID)`
- *Not* all-or-nothing
  - gradual transition is straightforward

# Aspects for Performance Current Technique

- Components
  - Abstract interface to Game Objects
  - Encourages synchronous processing

# Aspects for Performance Looking Forward

- Aspects
  - Dynamically adding/removing Aspects supported *in a unified way*
  - Systems own their data, processing is cache-friendly, encourages asynch processing
  - No overhead querying for Components

# AOGD

## Thought Experiments

- How granular should aspects be?
- Should they be hierarchical or flat?
- Consider forming a state machine out of Aspects
  - State transitions replace the current state aspect with another
  - State initialized when Aspect added
  - Support for simultaneous states
  - States communicate via shared Blackboard (effectively member data) (like “dynamic pvars”)

# AOGD

## Thought Experiments

- New ways of visualization / analysis
- Cool Aspect Analysis screens:
  - Aspects by owner (Game Object ID)
    - Point at Game Object and see his Aspects
    - Select an aspect and see its properties & values
      - Aspects could implement a DebugDump method
      - To TTY or to Viewport
  - Aspects by System w/ owner
  - Pivot data

# Aspects for Performance

## Misc. Considerations

- Sharing data between Aspects
  - “cross-cutting concerns” in AOP lingo
  - Implement as systems see fit
    - Shared Data uploaded with multiple systems
    - System refers to data owned by other system
- Debugging
  - Effectively the same as before
    - Debugging the affected system
  - Debugger scripting could assist in locating objects within other systems

# Summary

- Drones were a very effective optimization
- Provided a *limited* test of Aspects
- Cell performance demands parallelism
- Aspect oriented game development plays directly into the strengths and weaknesses of the Cell

# Moderated Discussion

- Speak Up

