
MARCH 5-9
2007
SAN FRANCISCO
TAKE CONTROL
MOSCONE CENTER
www.gdconf.com
CMP


# Saints Row Scheduler

- Discussion Areas:
  - Concepts/Philosophy
  - Architecture
  - Performance Topics


# Scheduler Concepts/Philosophy

- What is a “Scheduler”?

  Control flow mechanism, similar to function call or thread dispatch.

  Used to manage dispatch of independently schedulable entities or “jobs” across multiple threads.

  Many, many valid designs.

  Commonly platform/hardware specific.


# Scheduler Concepts/Philosophy

- What is a “Job”?

  Independently schedulable entity, without sequential or data dependencies on other “ready” jobs.

  Generally non-blocking. No waiting for I/O, D3D device, other asynchronous events.

  For our purposes, amounts to a function pointer/data block pair.

  Decomposing an application’s processing into jobs is the bulk of the work in making an application multiprocessor-ready. Outside the scope of this discussion.


# Scheduler Concepts/Philosophy

- Design Criteria

  Simple as possible, intuitive as possible.

  Configurable by application, as flexible as possible.

  High-performance/low overhead, allowing fine job granularity.

  Mechanisms to handle preemptive events gracefully.


# Scheduler Concepts/Philosophy

- General Philosophy

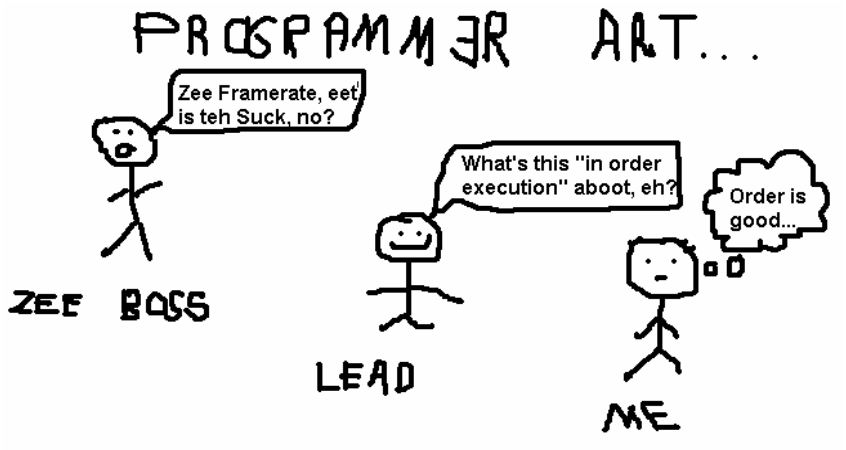
  Keep wires hanging out.

  Create and use building blocks.

  Avoid advanced language constructs.

# Architectural Block Diagrams

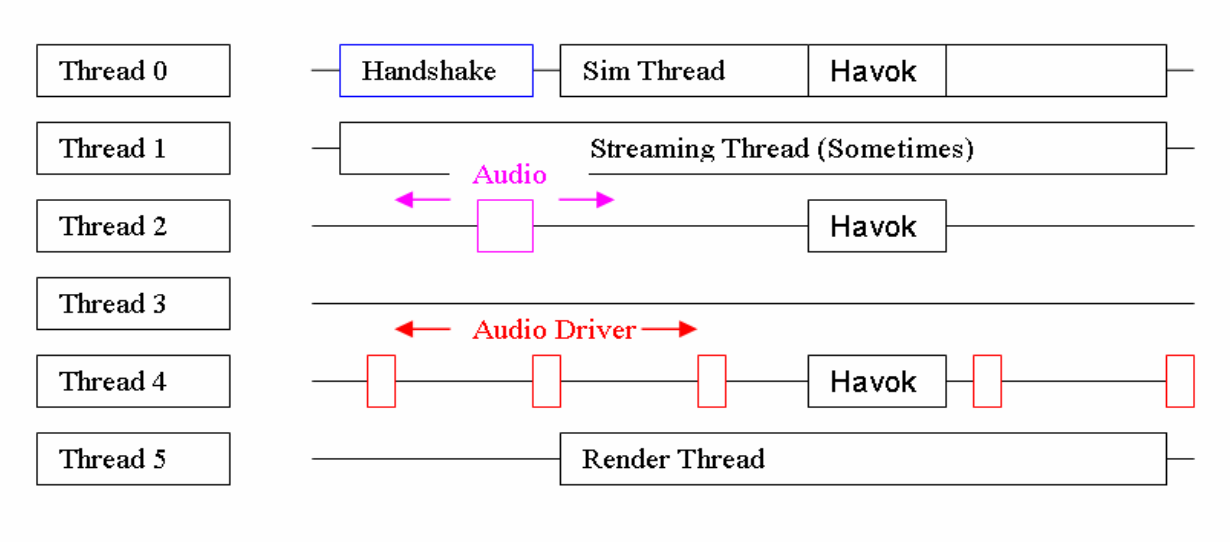
In the beginning, there were Alpha kits, then there were Betas…




- Saints Row derived from a single-threaded application.
- Six hardware threads, Woo Hoo!

# Initial Threading Layout

- Standard Sim/Render Split
  - Frame time varies, ~20 to ~50ms, usually ~33ms.
  - Audio driver uses ~2.3ms out of every 5ms.
  - When streaming active, uses entire thread.
  - Audio executes every ~30ms.




# Problems and Concerns

Scheduling a large block of jobs from one of the main threads can monopolize the job threads. Consequence of “first-come, first-served” order.

Some threads have reduced bandwidth.

- Thread 4 with DSP/Audio Driver only ~65%.
- Thread 1 can be dedicated to streaming.
- Both audio and streaming are preemptive.

Combined- and Pre-pass rendering both lengthy, monolithic operations.

Havok issues

- Uses its own threading utility.
- Must have thread memory allocated per-thread.
- Often goes serial.


# Scheduler Design Refinement

- First-come, First-served Order

  Using FIFO job Q results in “natural” job processing order.

  Problem – scheduling a large block of jobs from one of the main threads can monopolize the job threads.
  - Finer job granularity no help.
  - Adding priority to jobs no help.

  Solution: Add additional FIFO job Q’s and “job bias”.
  - Job threads biased towards serving sim-type or render-type jobs.
  - Dynamically configurable – can change bias algorithms and thread/job types on the fly.
  - Ensures each of main threads gets some job thread time.


# HandlingThreading Special Cases:

- Split up job threads between Sim and Render jobs.
  - Sim gets 0, 2, and 1 when not streaming.
    - Havok only runs on 0, 1 and 2.
  - Render gets 3, 5, and what's left of 4 after audio processing.
    - Also gets 1 and 2 during render intensive portions of frame.
- Combined and PrePass:
  - Run prepass on main rendering thread, "nail" combined pass job to thread 3.
    - Top priority job.
    - Thread 3 free of other processing threads.

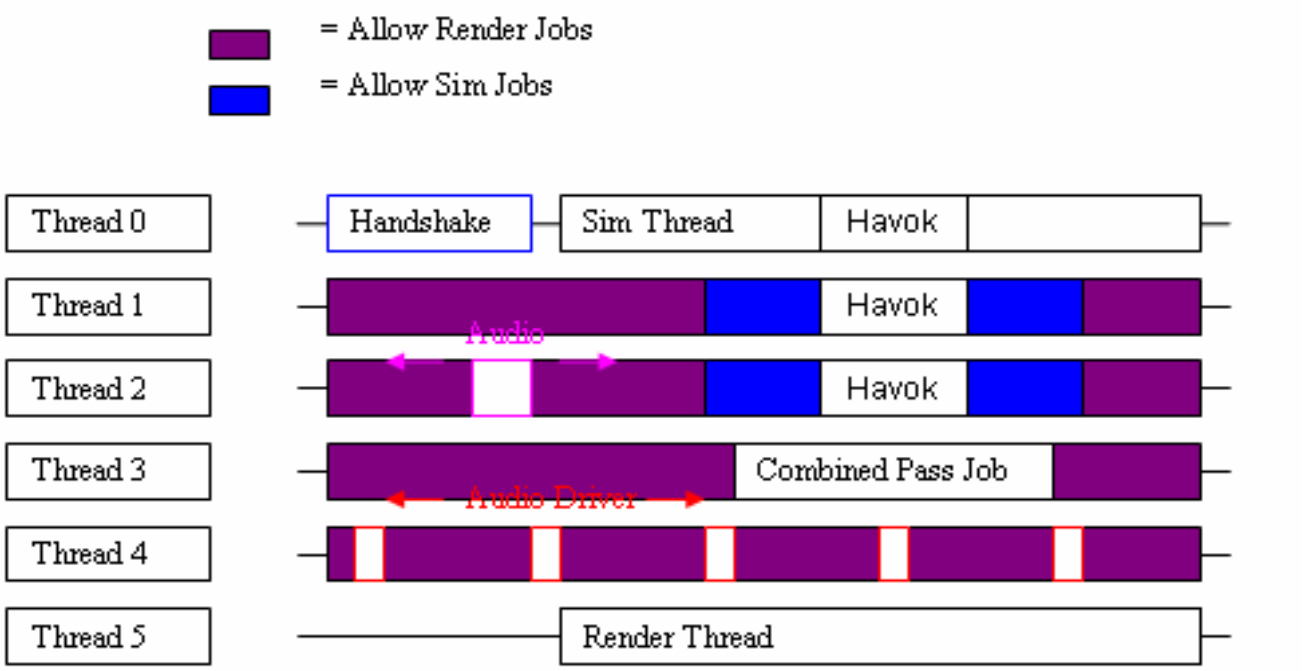

# HandlingThreading Special Cases:

- Havok
  - Comes with its own threading utilities.
    - No dynamic control
    - Each thread performing Havok processing needs Havok thread memory.
  - Executes ~half of processing serially.

  - Solutions:
  - Dedicate three threads to Havok.
    - Allocate thread memory only for those threads.
  - Call Havok timestep from our own scheduler.
    - Allows threading control, add or remove threads on per-frame basis.
    - Performance identical to Havok threading utility.
  - Break out and splat serial portions ourselves.


- Flow – around.
- Want to make use of thread 4, but it has a high-priority thread scheduling intermittently.
  - Pre-empts and runs for 2.3ms.
  - Will cause six thread “dead spot” if pre-empts and blocks completion of a job batch.
- XAudio supplies frame start and end callbacks.
  - Tied to scheduler, allows currently executing job to complete, then terminates job thread.
  - Reactivates job thread on frame end.
  - User specifies whether job is “short” or not.
  - Same mechanism used for audio thread.
- General problem – preemption can catch application code in a blocking state.
  - Critical Sections.
  - See Lockless options.

- Final Thread Layout
  - Havok moved to top 3 threads.
  - Combined pass fixed to thread 3
  - Render jobs allowed during most of frame.
  - Sim jobs allowed during intensive sim processing.
    - Actual sim window more complicated.
  - Jobs may schedule during main thread idle time.

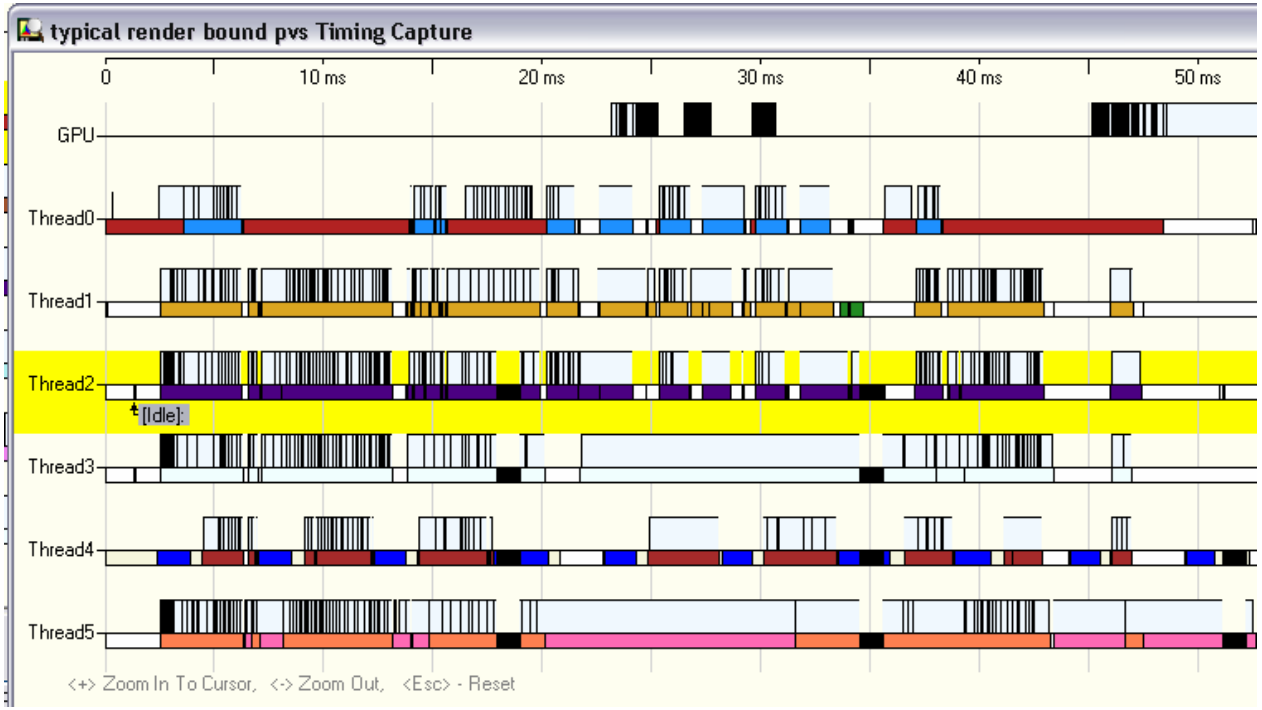



# Only thing left – splat the application!

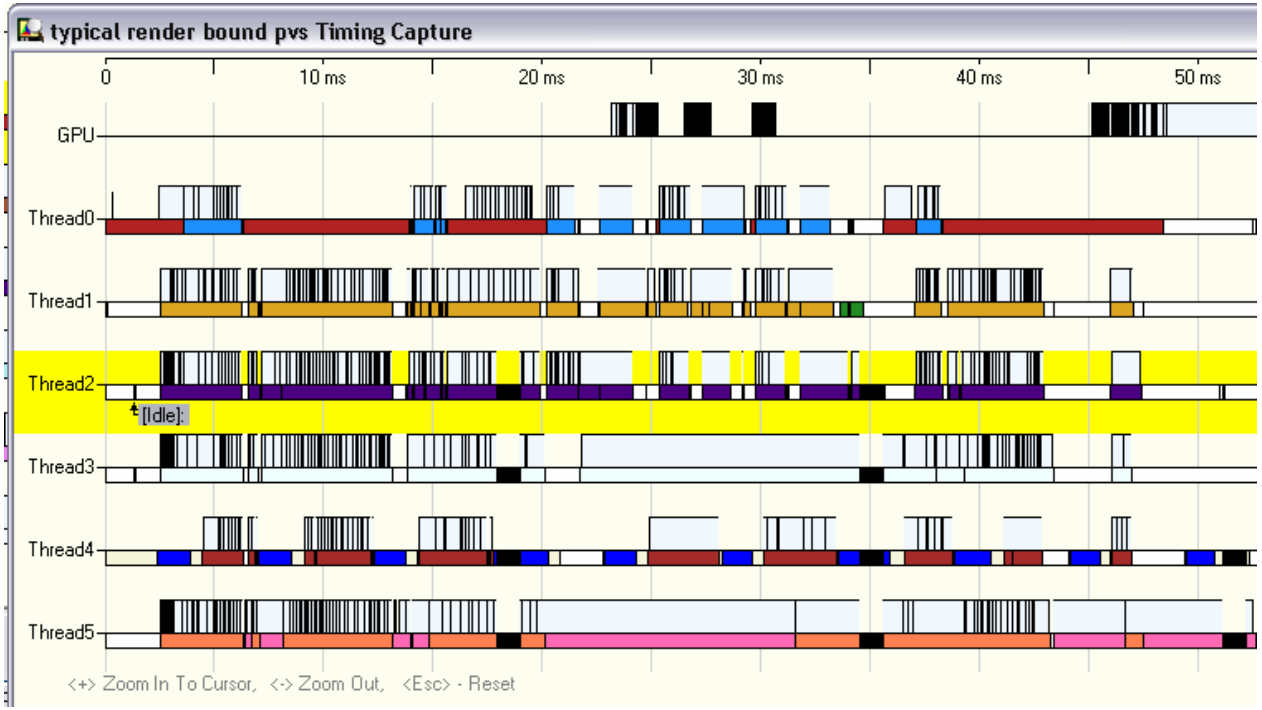
- The hard part.
  - Out of scope for us.
- While splatting - Optimal job size is function of scheduler overhead.
  - Set some “acceptable” criteria such as “5% overhead or less”, then measure the per-job scheduling time.
  - For Saints Row, optimal size somewhere around 250-500 microseconds.
- Not desireable for jobs to take much longer.

# Saints Row PIX timeline

## About 90% CPU usage



# Detail Discussion, Job flow




# Scheduler Internals

- Saints Row version, all protection by critical sections or spinlocks.
- Six job threads sitting on each of the hw threads.
- Inserting jobs into job Q activates any idle, matching job thread.
  - Events, not semaphores, for flexibility.
- Job spawning thread may suspend and wait for dispatching event.
  - Suspending thread specifies what job types may run on his hw thread.
- On completion, jobs fire either events (event triggers) or schedule more jobs (scheduling triggers).


# Performance

- Single-job queueing.
  - More efficient to move block of jobs to job queues, more flexible to move one by one.
  - Job queue is a significant overhead source due to thread contention.
  - If using critical section protection, probably should block queue.


# Performance

- Lockless
  - Lockless structures (stacks, queues) perform significantly better than critical section protected structures.
  - LIFO –
    - SLists, GPGems 6 stack
    - Fairly straightforward.
  - FIFO –
    - Michael's floating node.
    - Fober's reinsertion.
  - Be Careful


# Performance

- Profiling
  - PIX
  - DmNotify Threadswitch
  - Realistic test cases
    - samples


GameDevelopers
Conference
07
TAKE
CONTROL
March 5-9, 2007 in
San Francisco
CMP

# Questions?