id Tech 5 Challenges

From Texture Virtualization to Massive Parallelization

J.M.P. van Waveren
senior programmer
id Software
- GPU virtual texturing, a couple of interesting issues
- How virtual texturing got us to a parallel job system
- Widespread use of the job system throughout the engine
- Getting the jobs back onto the (GP) GPU
Virtual Texturing

• Unique, very large virtual textures key to id tech 5 rendering
• Full description beyond the scope of this talk
Virtual Texturing
Virtual Texturing
Virtual Texturing

Texture Pyramid with Sparse Page Residency

Physical Page Texture

Quad-tree of Sparse Texture Pyramid
Virtual Texturing
Virtual Texturing
Virtual Texturing

Very Large = 128k x 128k texels (1024 pages on a side)
Virtual Texturing

Very Large = 128k x 128k texels (1024 pages on a side)
Virtual Texturing

Very Large = 128k x 128k texels (1024 pages on a side)
Virtual Texturing

Very Large = 128k x 128k texels (1024 pages on a side)
Virtual Texturing

A few interesting issues...
- Texture filtering
- Thrashing due to physical memory oversubscription
- LOD transitions under high latency
Virtual Texturing - Filtering

- We tried no filtering at all
- We tried bilinear filtering without borders
- Bilinear filtering with border works well
- Trilinear filtering reasonably but still expensive
- Anisotropic filtering possible via TXD (texgrad)
  - 4-texel border necessary (max aniso = 4)
  - TEX with implicit derivs ok too (on some hardware)
Virtual Texturing - Thrashing

- Sometimes you need more physical pages than you have
- With conventional virtual memory, you must thrash
- With virtual texturing, you can globally adjust feedback LOD bias until working set fits

32 x 32 pages

1024 Physical Pages

8x8 pages

64 Physical Pages
Virtual Texturing – LOD Snap

• Latency between first need and availability can be high
  – Especially if optical disk read required (>100 msec seek!)
• Visible snap happens when magnified texture changes LOD
• If we used trilinear filtering, blending in detail would be easy
• Instead continuously update physical pages with blended data
Virtual Texturing – LOD Snap

- Upsample coarse page immediately
- Then blend in finer data when available
Virtual Texturing - Management

- Analysis tells us what pages we need
- We fetch what we can

- But this is a real-time app... so no blocking allowed
- Cache handles hits, schedules misses to load in background
- Resident pages managed independent of disk cache
- Physical pages organized as quad-tree per virtual texture
- Linked lists for free, LRU, and locked pages
Virtual Texturing - Feedback

• Feedback Analysis
  – Gen ~breadth-first quad-tree order w/ priority

Color Buffer

Feedback Buffer

![Diagram showing breadth-first quad-tree order with priority levels 0 to 3]
Virtual Texturing - Transcode

- Transcode
  - diffuse, specular, bump and cover/alpha
  - specular block scale stored in bump
- Typically 2-6kB input, 40kB output
- Unmap, Transcode, and Map all happen in parallel on platforms that can directly write texture memory

Transcode pipelined to block or row level to reduce memory profile.
Virtual Texturing - Pipeline

• Compute intensive complex system with dependencies that we want to run in parallel on all the different platforms
Game Engine Situation Today

• Logical GPU Architecture Stable
  – DX9 == nirvana for conventional hardware graphics
  – programmable stages, fixed topology

• CPU Architectures all over the map
  – Fast single core model definitely dead
  – Homogenous / Symmetric processors (PC, XBox)
    • big cores w/ cache, 1-2 hardware threads / core
    • some have complicated out-of-order processing
  – Heterogeneous processors (Cell)
    • 1-2 big cores
    • multiple small in-order cores w/ local memory & DMA controller
  – Streaming processors / GPGPU (NVIDIA / AMD GPUs, Intel Larrabee)
    • many cores
    • CUDA / OpenCL

• Challenge: one engine to efficiently harness them all
What's the big deal?

- id Tech 5 does a lot of processing
  - Animation blending – ~2 msec
  - Collision detection – ~4 msec
  - Obstacle avoidance – ~4 msec
  - Transparency sorting – ~2 msec
  - Virtual texturing – ~8 msec
  - Misc processing – ~4 msec
  - Rendering – ~10 msec
  - Audio – ~4 msec

- And at 60 Hz, not much time to do it – 16 msec
- Portable parallel software architecture is required
What Software Architecture?

• OS thread factoring
  – Good for small # of cores
  – Not terribly invasive
  – Complexity grows nonlinearly
  – Load balancing tricky
  – Not a good match for cell SPUs

• Small stand-alone job decomposition
  – Quite invasive rewrite
  – Very scalable
  – Almost required by cell SPUs
  – Good for heterogeneous processors
Job Processing System

- Simplicity key to scalability
  - Job has well defined input and output
  - Independent stateless, no stalls, always completes
  - Jobs added to job lists
  - Multiple job lists
  - Job lists fully independent
  - Simple synchronization of jobs within list through “signal” and “synchronize” tokens
Death by Synchronization

- Synchronization means waiting, waiting destroys parallelism
- Architectural decision: Job processing given 1 frame of latency to complete
  - Results of jobs show up a frame late
  - Requires some algorithm surgery
    - e.g. foliage
  - Rules out some algorithms
    - e.g. screen-space binning of transparency sort
  - But overall, not a bad compromise
id Tech 5 Job Decomposition

• Major parts of id Tech 5 processing factored into jobs
  – Collision detection
  – Animation blend
  – Obstacle avoidance
  – Virtual texturing
  – Transparency processing (foliage, particles)
  – Cloth simulation
  – Water surface simulation
  – Detail model generation (rocks, pebbles etc.)
Collision Detection

• Two phases
  – Query (continuous collision detection CCD)
    • Check sub-model collisions
  – Merge
    • Find the first collision or gather all contacts
• Player physics does not use delayed detection
  – 16 msec extra delay in user feedback undesirable
Animation Blend

• Animation graph or “web” describes valid transitions
• A stack is used to evaluate a blend tree
  – Leaves are decoded source animations
  – Parents are intermediate blend results
• Tree walking generates a command list for the stack
• Most blending happens in local space (parallel)
• Final phase moves everything to model space
Obstacle Avoidance

• One job per character that wants to avoid obstacles
• Construction of job input comes from a scan of Area Awareness System for potential obstacles and their surroundings
Transparency

• Transparency requires sorting and blending: expensive
  – Must be handled separately
• Restrict to particle systems and foliage
• Limited buffer size

• Split into a number of jobs
  – Foliage gather
  – Foliage gen
  – Particle gen
  – Transparency sort and index gen

• Tricky to keep these jobs under SPU limits
Jobs on the (GP) GPU

- We are cautiously optimistic about the job model
  - Anticipate CUDA, OpenCL, Larrabee support
    - Easy to add additional job processing resources
  - But this is new territory…
Jobs on the (GP) GPU

- Not enough jobs to fill SIMD / SIMT lanes
- Code paths of different jobs diverge too much
- Jobs are useful as unit of work (latency tolerant & small memory footprint)
- Data parallelism within jobs needs to be exploited
- Split jobs into many fine grained threads
- Data dependencies in input
- Convergence of output data
- Memory access of the fine grained threads is important
Conclusions

• Virtual texturing + great artists = awesome environments
• id Tech 5 does a lot of work and has to exploit parallelism
• Cell forced us to re-factor engine into jobs
• Latency tolerant computational services model attractive
• Jobs are now running on a variety of processors
• Hopefully soon CUDA, OpenCL, Larrabee support
Virtual Texturing
Virtual Texturing
Questions?

• Please complete the course evaluation at: http://www.siggraph.org/courses_evaluation

• Chance to win a SIGGRAPH’09 mug!
  One winner per course, notified by email in the evening.