The Next Generation of PhyreEngine™

Matt Swoboda
Principal Engineer
Sony Computer Entertainment Europe R&D

History

History

- PhyreEngine 2.X
  - PlayStation®3 as primary target
    - Heavy SPU usage; Supported multicore ports of SPU work
    - Assumed RSX level for graphics
  - Works on PC, portable to other platforms
    - Shipped example DX9 implementation
    - PSP®(PlayStation®Portable) version available
- Used in > 45 released titles

PhyreEngine 3

PhyreEngine 3

- Targets NGP, PS3™, PC
  - Support wider variety of hardware
    - CPU and GPU
- Greatly enhanced tool set
- Scripting support

Ported Most Frequently Used Phyre 2 Components

- Text
- LOD
- Animation
- Physics
  - Bullet
  - Havok™
  - NVIDIA® PhysX®
- Occluders
- Dynamic Geometry
- Profiling
- Scene
- SPU Post Processing

New object model

- Data oriented
  - Entity/component model
  - Target data parallelism
- Serialize data straight to target's class layout
  - Automatic packing and fix up for serialization
  - Different memory-mapped binary file per target

Scripting at Core

- Members and methods from object model
- Isolated context for each script
  - Enables parallelism
- Runtime script scheduler in most recent version

Upgraded Renderer

- Best features ported from Phyre 2
  - SPU dynamic geometry, post processing, deferred renderer etc.
- Added ubershader support
  - Simplifies asset creation
- Split renderer into separate thread[s]
  - Multithreaded submission to single render thread

Improved DCC Tool Integration

- COLLADA™
  - Take data from wide range of supporting tools
- Direct support for Maya® and 3ds MAX®
  - Our own supported exporters – much improved over standard issue COLLADA exporters
- Our ubershader works as a shader node
  - Same look & features as in game

Simplified Tool Set

- Asset Processor
  - Inputs:
    - COLLADA™, textures, CgFX, scripts + more
  - Outputs platform specific cluster file
    - Plus cache of pre-processed cluster for reuse
- Level Editor
  - Build levels for use in the game engine

Phyre Level Editor

- Place assets from DCC tool
  - Meshes, lights, collision data
- Gameplay elements
  - Triggers
  - AI / Navigation mesh
  - Entity and component editing

Entity Templates

- Create entity templates in Level Editor
- E.g. a soldier:
  - AI / Nav component
  - Renderable component
  - Physics component
  - Trigger interactions component

Phyre Level Editor

- Run your game application in the editor
  - Communicates via TCP
  - Hosts your game with Game Edit library
  - Game Edit passes Phyre object model info to editor
    - Additionally include game specific data
- Connect to PS3 and NGP
  - On target preview and editing

Demo

The All-New Phyre 3 Game Template

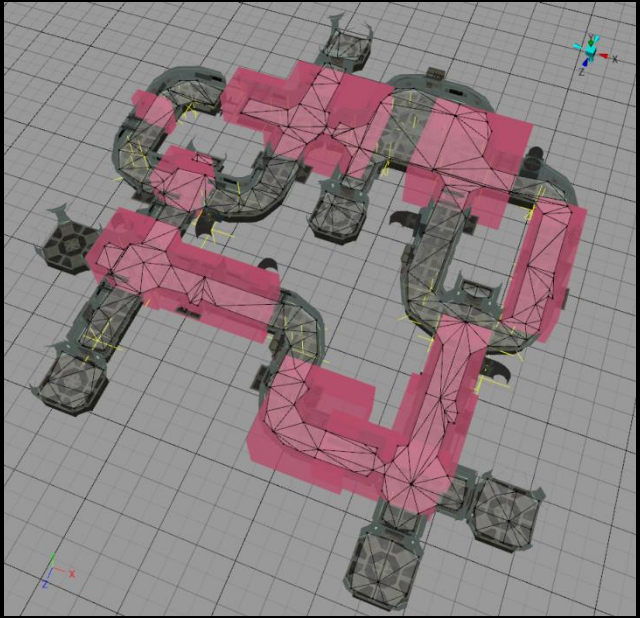
Building The Demo

- Assets built in DCC tool
  - Level chunks
    - Shaders, lights, collision mesh in DCC tool
  - Characters + animations
- Imported into Phyre Level Editor
  - Chunks snapped together to form a level
  - Add occluder geometry
  - Edit lighting
  - Add triggers, doors, spawn points

AI / Navigation

- High level soldier behavior scripted in Lua
  - Communicates with Phyre object model
- Soldiers use navigation component
  - Uses Detour to move to target
  - Based on pre-generated Recast nav mesh
  - Includes DetourCrowd to avoid collision

Navigation Mesh



Physics

- Physics meshes imported from DCC
  - Rigid body setup
- Choice of Bullet, Havok, PhysX
  - Can switch at compile time

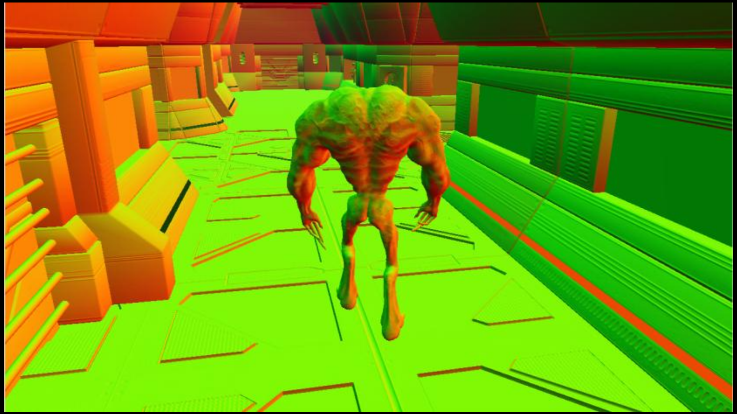
Rendering Overview

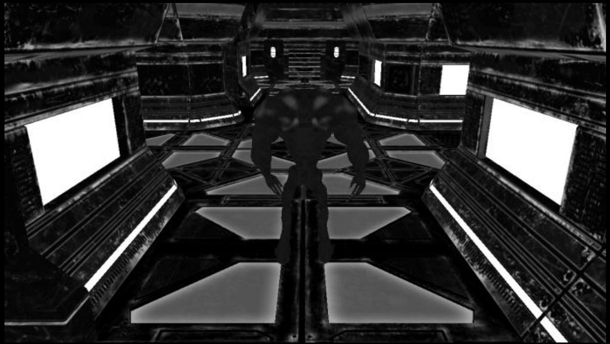
- Good performance
  - Full 720p – 1280x720 or half 1080p – 960x1080
  - 60 hz framerate usually
  - Load is roughly evenly balanced between RSX and SPU
- Full deferred renderer on SPU
  - Large number of point and spot lights; 4 shadow casters
- Post processing on RSX/SPU
  - Glow, motion blur, MLAA (no MSAA)

Rendering Overview

- GBuffers:
  - 24 bit color + 8 bit specular gloss - in main memory
  - 8 bit view space normal x & y + 16 bit view space linear depth - in main memory
  - 24 bit depth, 8 bit stencil - in VRAM
    - Stencil is used for object ID for motion blur
  - Shadow buffer : 32 bits split between N lights - in main memory
    - Used 4 lights, 8 bit per light here
- Velocity buffer reconstructed in post

Composition: GBuffers





Composition: Lighting

Composition: Lit + Tone Mapped + MLAA

Composition: Motion Blur + Glow

Composition: Particles & Transparencies

Composition: Final + HUD + Spot Effects

The Anatomy of a Frame

RSX

- Rasterize GBuffers
- Rasterize shadow maps
- Generate glow buffer
- Apply MLAA
- Render particles & transparencies
- Motion blur and composite

- One frame of latency

SPU

- Visibility & Occlusion Culling (geometry & lights)
- Animation & Skinning
- Particle simulation / sort
- MLAA pre-process
- Deferred lighting + tone map

Deferred Lighting on SPU

- First presented at GDC09
  - Now an increasingly popular approach in technically advanced titles
- Handles point, spot and directional lights
  - Specular supported on all lights
- Tile-based classification approach
- Lights culled to per-tile frusta
  - Get min+max depth in tile to build frustum
  - 32 x 16 pixel tiles
  - Cull cones for spots, spheres for points
- Multiple shadows supported
  - Screen space shadow buffer generated on RSX
  - Pack multiple shadows into RGBA8

Deferred Lighting on SPU

- Reads tiled input buffers from main memory
  - No read back from RSX required
  - Reading back multiple Gbuffers gets **expensive**
  - De-tile on SPU
- MSAA optimisation
  - Not used here..
- Tone map in place
- Also supports fog, color correction in place & more..
  - Add value by rolling in more operations on same buffers for free/cheap
  - Don’t waste RSX time on them
- **Result: balanced SPU/RSX renderer**

Morphological Antialiasing

- Post process for antialiasing edges [1]
- We have a GPU version for PC
  - ~16 ms on RSX – too slow
- PlayStation®EDGE Post – SPU-only
  - Fast
  - Potential latency – executes on SPU after final color buffer rendered

Morphological Antialiasing

- Our version: pre-process on SPU, final render on RSX
  - Pick input available early, in main memory already – low latency
  - Support tiled inputs
- Generate edge buffer and edge length buffer on SPU
  - Fast linear processes
- Generate tile classification buffer on SPU
  - Find regions with edges requiring MLAA
  - Point sprite list, 8x8 pixel tiles
- Render final MLAA pass on RSX
  - Low latency

Morphological Antialiasing

- Make best use of RSX and SPU together
  - **Perform each task on the best unit for the job**
- Rolled pre-process into SPU deferred lighting
  - Uses final output buffer for best possible quality
  - Improved performance
    - Edge detect and tile classify rolled in with lighting code: don’t have to DMA in & de-tile again

Morphological Antialiasing

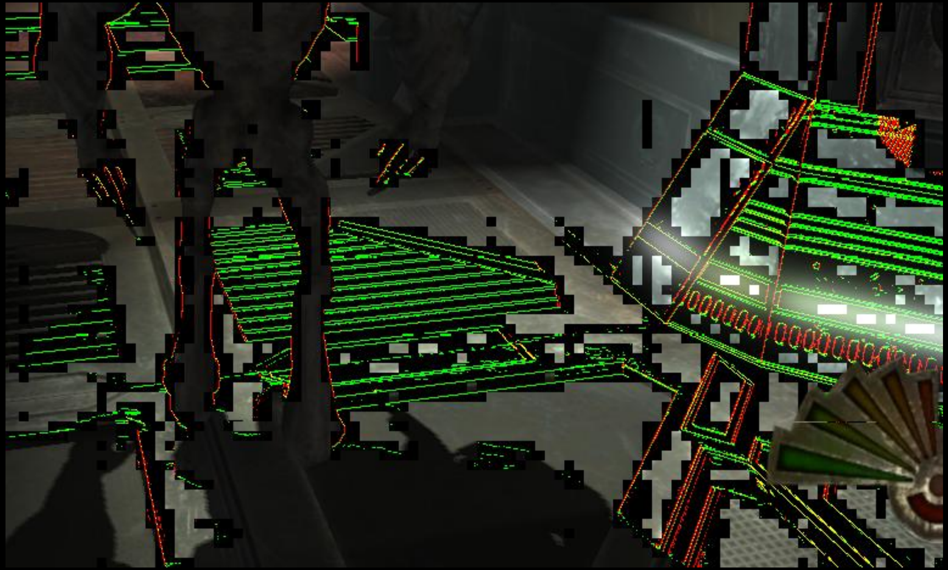
- Without MLAA (zoomed):

Morphological Antialiasing

- With MLAA (zoomed):

Morphological Antialiasing

- MLAA tiles:



Particle System

- Motivation: modern, high quality GPU-style particle system on SPU
- Emit from and attract to skinned meshes
  - Smooth transition from mesh to particles
- Fluid-esque movement
- Depth sorting

Particle Simulation

- Skinned meshes
  - Particle positions pre-calculated
  - Spread particles by triangle area with barycentric coordinates
  - Apply skin calculations to particles on SPU
- Curl noise [2]
  - Approximates the look of fluid simulation procedurally
  - Back-ported to SPU from my GPU version
  - Optimisations for SPU: parallelize, reduce table lookups

Particle Sorting

- "Bucket and bitonic"
- Bitonic sort implementation from Phyre 2/3
  - Fastest brute force sort on SPU
  - Requires all elements in memory at once
- Bucket sort
  - Place particles into buckets by depth
  - FIFO per bucket - handles arbitrary particle count
- Bucket sort first, then bitonic sort per bucket

PhyreEngine on our next generation portable entertainment system (codename: NGP)

NGP Overview

- Multi-core ARM® mobile CPU
- GPU: SGX543MP4+ chipset
- Tile-based deferred renderer (TBDR)
- Flexible and programmable shaders [3]
- **Different to typical desktop GPU architecture**
  - Requires some mental adjustment..

Porting Phyre to NGP

- Phyre 2 was designed for PC / home console
  - Assumed desktop GPU, fast CPU clocks
  - Many techniques targeted at SPU or fast multi-core CPUs
- More scalability required
  - Greater abstraction between hardware targets

Porting Phyre to NGP

- Initial port was not hard
  - Unit tests and harness ported in a day
  - Core rendering features in 2 weeks
- PC version a better starting point than PS3
  - No SPU code to worry about, more GPU-oriented

Porting Phyre to NGP

- CPU code ported easily from PC version
  - ARM CPU is kind – handles scalar, branch-heavy code well
- Multithreading essential
- Scene traversal, animation, visibility, occlusion culling moved to CPU
  - Plain PC multi-threaded versions ported well

Porting Phyre to NGP

- GPU shaders initially ported from PS3/PC
  - Compilation all offline
  - We have a common front-end using an ubershader model
- Moved skinning to GPU
- Forward renderer with cascaded shadows up and running quickly
  - Few changes needed from PS3/PC shaders
- Our initial port used the same shaders as PC version
  - But **most** need tuning specifically for NGP for performance

Meshes & Textures

- Vaguely similar resolution meshes to PS3, but..
  - Vertices must be written to memory until fragment stage – every vertex costs memory to render on TBDR
  - No Edge-style culling support
- Reduce skinned meshes and skeletons from PS3
  - Animation and skinning was trivial on SPU
- Optimise vertex format
  - Split streams by use: separate streams needed for shadow passes from those only needed for color passes

Meshes & Textures

- Make use of texture compression
- LOD aggressively
  - Meshes, textures and shaders
  - Different LOD blending scheme per platform?
- **So far – similar bottlenecks to PS3 version**
  - Our long fragment shaders are the limiting factor

Rendering

- First approach: forward renderer
  - All shading in one pass
    - (+ shadow generation)
- Ubershader selects best shader for context per instance
  - Affecting lights, shadow casters, affecting cascades
- Generated long fragment shaders
  - Hard to use many lights efficiently
- Now adding a deferred rendering solution alongside forward renderer
  - Targeting feature parity with other platforms

Deferred Rendering

- We have a working deferred rendering solution for NGP
  - Targeted for our next release
- Reduced Gbuffer format
  - Similar to what we used for SPU deferred rendering on PS3
  - Consider which Gbuffer channels you really need
- Light pre-pass implementation available too
  - Preferred to avoid it because of the two geometry passes – as on PS3

Summary

- PhyreEngine 3 is available **now**
  - **Free** for all PS3/NGP licensees
  - Beta download from PS3/NGP Devnet today
  - Full release due in April

PhyreEngine 2.7 also available on PSP®

Thanks

- Contributors & Assistance:
  - PhyreEngine Team
  - SCE R&D
    - Steven Tovey, Colin Hughes, Sebastien Schertenleib
  - SCE WWS ATG
    - Nicolas Serres, Simon Brown, Tobias Berghoff, Matteo Scapuzzi, Jan Althaus et al.

References

- [1] Intel Labs: *Morphological Antialiasing* (2009)
- [2] R. Bridson: Curl noise for procedural fluid flow (2009)
- [3] http://www.imgtec.com/news/Release/index.asp?NewsID=428
- [4] http://www.scribd.com/doc/27337782/Powervr-Sgx-Series5xt-Ip-Core-Family-1-0