BitSquid Tech
Benefits of a data-driven renderer
Tobias Persson
GDC 2011
Agenda

• An introduction to BitSquid
• Key design principles of BitSquid Tech
• Benefits of having a data-driven rendering pipe
BitSquid

- Core team consists of me (rendering) and Niklas Frykholm (system)
- Based in Stockholm, Sweden
- Founded in September 2009 after GRINs unfortunate bankruptcy
Ambitious Goal

• To develop a new high-end game engine for licensing
• Cross-platform: PS3, X360, PC/DX11 (+ future “console” HW)
• We call it *BitSquid Tech*
Fatshark collaboration

• Impossible to build game technology without close collaboration with end-users (i.e. game developers)

• Fatshark is an independent mid-sized game developer [Lead and Gold, BCR2, Hamilton]

• BitSquid and Fatshark shares office space

• Fatshark are our crash test dummies
Products running BSTech

- **Stone Giant** [BitSquid / Fatshark] - DX11/tessellation tech-demo
- **Hamilton’s Great Adventure** [Fatshark] - 3rd person puzzle game to be released on PSN and Steam
- Two external developers working on unannounced projects
Stone Giant Demo
Key design principles of BitSquid Tech
Fast Content Iterations

• Being able to iterate fast over content is key to create great games

• In BSTech content is everything from low-level engine configuration files to high-level art assets

• Support hot-reloading of all content
Multi-core

- All bulk workload run through our job-system
  - Mixture of task & data-parallel jobs
- Data oriented design
  - Heavy focus on memory-layout of input/output data
  - Easy DMA to coprocessors (SPU/GPGPU)
Tools

• No mega-editor™
• Instead: multiple tools designed for specific purposes
• Much easier to extend and add new tools
• Gathered in a central launcher called “tool center”
Tools cont.

• All visualization using “real” engine

• Avoids strong coupling to engine by forcing all communication over TCP/IP
  • Boot engine with tool slave script
  • Tool sends windows handle to engine, engine creates child window with swap-chain
  • Write tools in the language you prefer
Animation State Machine Editor
Particle Editor
Console Focus

• Hard to get all departments to test their content on console HW
  • Two options: 1. Make PC runtime suck or 2. Make console testing easy

• All tools run on console, examples:
  • Mirroring of level editor
  • Simultaneous tweaking of lighting / material properties
Level Editor mirroring

PC

PS3

Content from unannounced project
Courtesy of Fatshark
Benefits of a data-driven renderer
Definition

• What is a data-driven renderer?
  • Shaders, resource creation/manipulation and flow of the rendering pipe is defined entirely in data
  • Hot-reloadable - for minimal iteration times
Motivation

- Multiple projects with different needs
  - Projects targeting 60Hz will have a completely different rendering pipe than those targeting 30Hz
  - Artistic style: Photorealism vs Toon-shading, Full 3D vs 2.5D, etc.
Early prototype of a 60Hz rendering pipe (Running on PS3)

- Baked AO
- HDR but no light-adaption
- Simplified post processing

Content from "The Fight: Lights Out"
Courtesy of Coldwood Interactive and SCEE
Motivation cont.

• Flexibility: Easy debugging and experimentation
• Scalability: Targeting a wide range of HW requires a rendering pipe that scales
• Game context aware rendering pipe – e.g. stop rendering sun shadows when indoors, simplify rendering in split-screen, etc.
• High-level render pipe code not performance critical
.. is very different from ..
Hamilton’s Great Adventure (PS3)
Courtesy of Fatshark
Implementation

- The `render_config` file:
  - JSON* configuration file describing the entire rendering pipe
  - Supports hot-reloading

Overview of render_config

Global Resource Set

Layers Configuration

Resource Generator

Local Resource Set

Viewport
Global Resource Set

- Specifies GPU resources to be allocated at start-up
- Mainly render targets (all global RTs except swap-chain)
- Resources identified by name

```json
global_resources = [
    { name = "depth_stencil_buffer" type="render_target" depends_on = "back_buffer" w_scale=1 h_scale=1 format="D24S8" hint_needs_clearing = true },
    { name = "albedo" type="render_target" depends_on = "back_buffer" w_scale = 1 h_scale = 1 format = "RGB8888" hint_needs_clearing = false },
    { name = "normal" type="render_target" depends_on = "back_buffer" w_scale = 1 h_scale = 1 format = "RGB8888" hint_needs_clearing = false },
    { name = "depth" type="render_target" depends_on = "back_buffer" w_scale = 1 h_scale = 1 format = "R32F" hint_needs_clearing = false },
    { name = "mask" type="render_target" depends_on = "back_buffer" w_scale = 1 h_scale = 1 format = "RGB8888" hint_needs_clearing = false },
    { name = "light_accumulation" type="render_target" depends_on = "back_buffer" w_scale = 1 h_scale = 1 format = "R16G16B16A16F" hint_needs_clearing = true },
    { name = "self_illumination" type="render_target" depends_on = "back_buffer" w_scale = 1 h_scale = 1 format = "RGB88888" hint_needs_clearing = true }
]```
Layers Configuration

- Global Resource Set
- Resource Generators
- Local Resource Set
- Viewport

Diagram:
- Global Resource Set -> Layers Configuration
- Layers Configuration -> Resource Generators
- Local Resource Set -> Viewport
- Viewport -> Layers Configuration
Layers Configuration

- Defines the draw order of the visible batches in a game world
- Layers are processed in the order they are declared
- Shader system points out which layer to render in
Layer Breakdown

- Name used for referencing from shader system
- Destination render targets (& DST) for the layer batches
- Depth sorting: front-to-back / back-to-front
- Optional Resource Generator* to run

* More on Resource Generators in the coming slides
Resource Generators

- Global Resource Set
- Layers Configuration
- Local Resource Set
- Viewport
- Resource Generators

Diagram showing the relationship between these components.
Resource Generators

• Minimalistic framework for manipulating GPU resources

• Used for post processing, deferred shading, shadow maps, procedural texture effects, debug rendering, and much more..

• Simple design - just a queue of Modifiers that knows when in the frame to run
Resource Generators

Example of a few *Modifiers*

- Fullscreen Pass
- Deferred Shading
- Shadow Mapping
- Compute Shader [DX11]
- SPU Job [PS3]
- Branch
Viewport

- Ties everything together
- Specifies which layer configuration to use
- Local resource set - resources unique for each instance of a viewport (useful for stuff like current adapted luminance)
- GP programmer renders a game world by calling
  - `render_world(world, camera, viewport)`
Questions?

tobias.persson@bitsquid.se

http://www.bitsquid.se

http://bitsquid.blogspot.com