OpenGL Efficiency: AZDO

Cass Everitt
OpenGL Engineer, NVIDIA
GDC, San Francisco, March 2014
AZDO?

- Approaching Zero Driver Overhead
Why do you care about driver overhead?

• Because \textbf{driver overhead} == \textbf{cost}

• Costs
  - CPU cycles from app
  - CPU cache from app
  - power / battery
  - GPU throughput
OpenGL Fallacy: Old and Inefficient

- Immediate Mode
- Fixed Function
- Display Lists
- Evaluators
- Ancient crufty stuff
- Feedback
- Selectors
- Selection
OpenGL Reality: Modern & Efficient

- Bindless ARB
- Multi-Draw Indirect GL4.3
- Texture Arrays GL3.0
- Buffer Storage GL4.4
- SSBO GL4.3
- UBO GL3.1
Plus, OpenGL has all the features

- Compute
- Geometry Shaders
- Tessellation
- Sparse Textures
- Image Load/Store
Classic OpenGL Model

Direct Drawing Commands
(via the command fifo)

CPU

GPU

Memory
- indirect draw
- buffer object
- buffer object
- texture object
- buffer object
- buffer object
- buffer object
- render target
- buffer object
Classic Model Pros / Cons

• Pro
  - Very stable - 20+ year old code still “just works”
  - Simple
    - driver handles hazards, sync, allocation
  - Empowered the GPU revolution
  - Many classes of applications well served

• Cons
  - Demanding apps are not so well served
    - Intense games, VR
  - Doesn’t scale with high scene complexity
  - Threading model
  - Hardware abstraction showing age
Aspirational Goal

• Can we address the cons within the framework of the existing API?
  - That is, can we fix the cons without tossing the pros?

• Good question!
  
  - As it turns out, Smart People in Khronos have actually been working on this question for a while now
  - And they’ve developed an efficient, modern OpenGL that
    - Gives amazing perf improvements, and lives within the existing framework

• And here’s what it looks like...
Efficient OpenGL Model

CPU → Memory → GPU

indirect draw
buffer object
indirect object
buffer object
texture object
buffer object
indirect object
buffer object
texture object
buffer object
indirect object
buffer object
render target
buffer object
CPU and GPU decoupled
CPU Writes Memory - multi-threaded (no API)!

Memory

- indirect draw
- buffer object
- indirect draw
- buffer object
- texture object
- buffer object
- indirect draw
- buffer object
- texture object
- buffer object
- buffer object
- render target
- buffer object

CPU

CPU

CPU

GPU
And/Or GPU Writes Memory

Still no API - the magic of communicating through memory...
GPU Reads Commands from Memory

Minimal CPU / driver involvement...
Results

• Integer multiple speedups $\sim 5x - \sim 15x$
  - This is not a typo
  - On driver limited cases, obviously

• Works TODAY on existing drivers!
  - Mostly GL4.2+
  - Extensions are at least EXT
Bonuses

- Enables scalable multi-threading with no new API
  - Cores just write to memory

- Enables GPU Work Creation
  - Compute job or similar
    - Builds buffers, constructs MDI commands

- Does not require a new object model

- Does not require breaking existing applications
Results

• **Integer multiple speedups ~5x - ~15x**
  - This is not a typo
  - On driver limited cases, obviously

• **Works TODAY on existing drivers!**
  - Mostly GL4.2+
  - Extensions are at least EXT
Results

- **Integer multiple speedups ~5x - ~15x**
  - This is not a typo
  - On driver limited cases, obviously

- **Works TODAY on existing drivers!**
  - Mostly GL4.2+
  - Extensions are at least EXT
Results

- **Integer multiple speedups ~5x - ~15x**
  - This is not a typo
  - On driver limited cases, obviously

- **Works TODAY on existing drivers!**
  - Mostly GL4.2+
  - Extensions are at least EXT