Porting DOOM to Vulkan

SIGGRAPH 2016

Axel Gneiting

id Software

Agenda

- Demo & short idTech 6 overview
- Porting to Vulkan
  - Shaders, pipelines & states
  - Descriptor Sets
  - Multithreading
  - Image layouts & barriers
  - Memory & synchronization
  - Asynchronous compute
- Results & Future Work

DOOM

123 FPS
8.25ms
Vulkan 1.0.8
NVIDIA
GTX TITAN X
VRAM 12288 MB
Driver 368.81.0.0
CODEX ENTRY ACQUIRED
MONSTERS
Imp
[TAB] Press to View
Press [E]
+25
104
∞

Video

idTech 6

- PC OpenGL & Vulkan, PS4, Xbox One
- DOOM and future id Software titles
- 60+ Hz on all Platforms
- Shader syntax similar to HLSL
  - Translated to PSSL/HLSL/GLSL at build time

CPU

- Parallel command buffer generation
  - Split up into several “contexts” per frame
  - Each contexts owns command buffer
  - For each context we run multiple jobs to fill CB
  - Last job in frame submits command buffers to GPU
- OpenGL runs sequential on one thread
  - Some scene preparation work is still in jobs

GPU

- Clustered forward shading with some deferred
- Same shader for most of the geometry
  - Same set of textures too (virtual texturing)
  - Very few state changes
- Extensive post process
  - DoF, Temporal AA, SSDO, motion blur, etc.
- Lots of asynchronous compute
  - DXT encode, particles & post processing

Porting to Vulkan

- Started 2015 with an early version
  - Wrote most of the Vulkan backend code
  - Got first triangle rendering
- Picked it up in late March 2016 again
- Was mostly running at game launch
  - RenderDoc helps, even better now!
- Small issues delaying release ☹
  - Driver issues
  - Swap chain surprisingly hard to get right

Porting to Vulkan

- Validation layers were unreliable back then
- Lots of false errors
- Had to write some validation code ourselves
- Validation layers much better now
- Still good to have own validation for debugging

Shaders

- Already had GLSL translator
  - But OpenGL was binding by name
  - Vulkan uses binding IDs at pipeline creation
- Using AMD extensions if available
  - Variant for all shaders
  - AMD_shader_ballot & AMD_gcn_shader

Shaders

- Normalized clip space is upside down
  - Shader generator adds `gl_Position.y = -gl_Position.y` at end of every vertex program
  - Can we please have an extension that fixes this?
  - Platform differences are a waste of time
- Z range is good: [0,1] ☺

Pipelines & States

- Abstraction layer still old style API like
- Need to emulate stateful API & track states
- Hash table for pipelines, render passes & frame buffer states
  - Way smaller perf overhead than thought
- Dynamic state for scissor/viewport/stencil and depth bias
- Only ~350 total graphics pipelines for entire game

Pipelines & States

- Pipeline creation expensive
  - Lookup misses unacceptable at runtime
  - Some pipelines take 100+ ms to compile
- Solution
  - Play game and serialize states to disk
  - On startup launch jobs to compile pipelines
  - Fairly robust, missed pipelines would just cause stalls for player

Descriptor Sets

- No deletion of Vulkan objects while playing
  - Geometry statically loaded
  - Textures virtualized
- Got away with a descriptor hash table
- One big descriptor set for each combination
- Complete table flush if a Vulkan handle gets deleted
  - Level load & unload, etc.
- About 3-4k descriptor sets usually

Descriptor Sets

- Dynamic uniforms written to ring buffer
- Thread safe allocation from ring with atomics
  - 256 byte align allocations for simplicity
- Bound with UNIFORM_BUFFER_DYNAMIC
  - Offset set as vkCmdBindDescriptorSets parameter
- Also used UNIFORM_BUFFER_DYNAMIC for skinning data
  - Baked range problematic
  - Got away with 64kB range for everything
  - Alternative would have been way more descriptor sets

Multithreading

- Mostly straight forward port from consoles
- Image layouts problematic (more soon)
- Double buffered CBs per context
- Read/write locks for state hash tables
    - Never blocks if no state misses

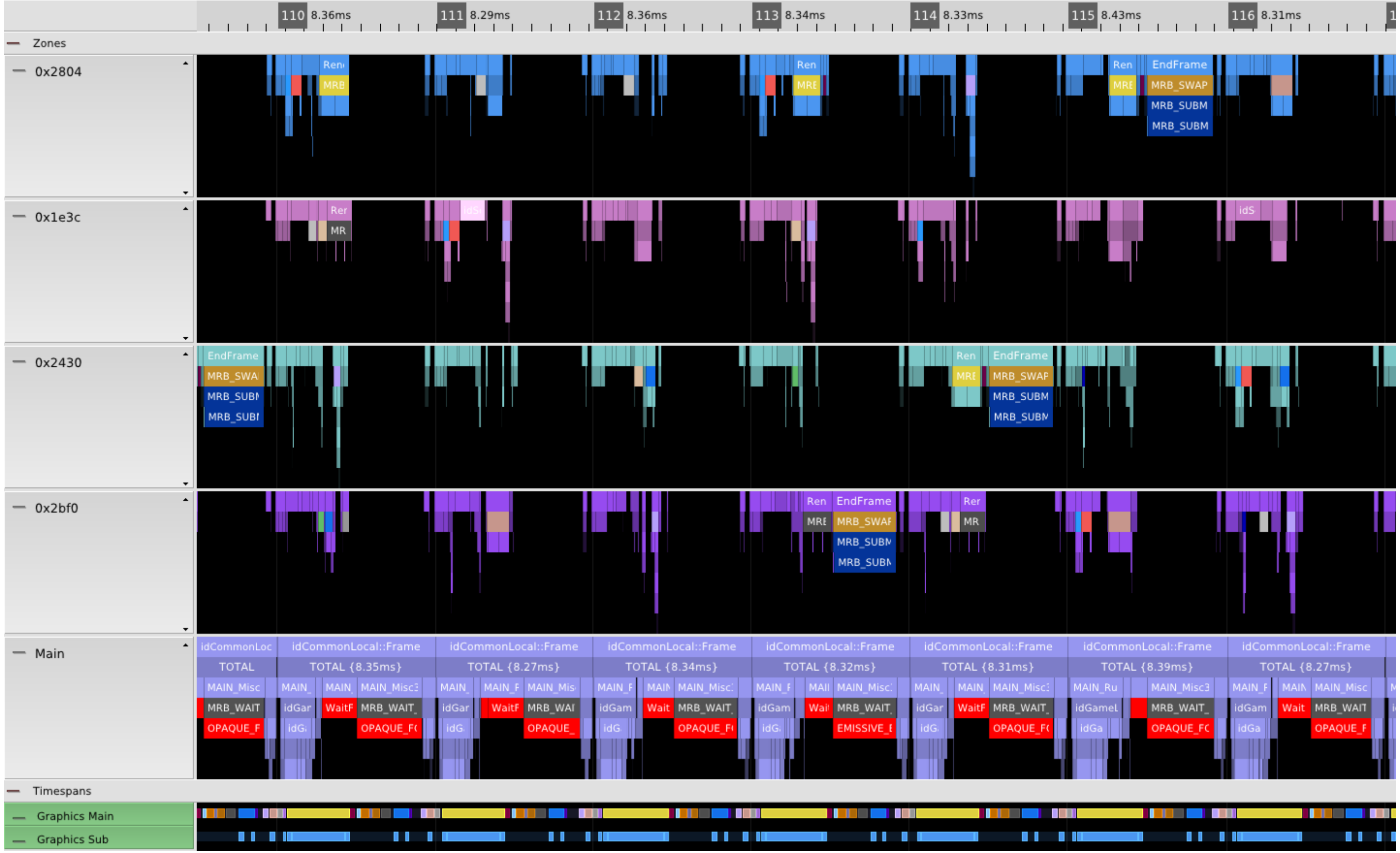

110 8.36ms
111 8.29ms
112 8.36ms
113 8.34ms
114 8.33ms
115 8.43ms
116 8.31ms
Zones
0x2804
0x1e3c
0x2430
0x2bf0
Main
EndFrame
MRB_SWAP
MRB_SUBM
MRB_SUBM
idCommonLocal::Frame
TOTAL {8.35ms}
TOTAL {8.27ms}
TOTAL {8.34ms}
TOTAL {8.32ms}
TOTAL {8.31ms}
TOTAL {8.39ms}
TOTAL {8.27ms}
MAIN_Misc
MRB_WAIT_
OPAQUE_F
EMISSIVE_E
Timespans
Graphics Main
Graphics Sub

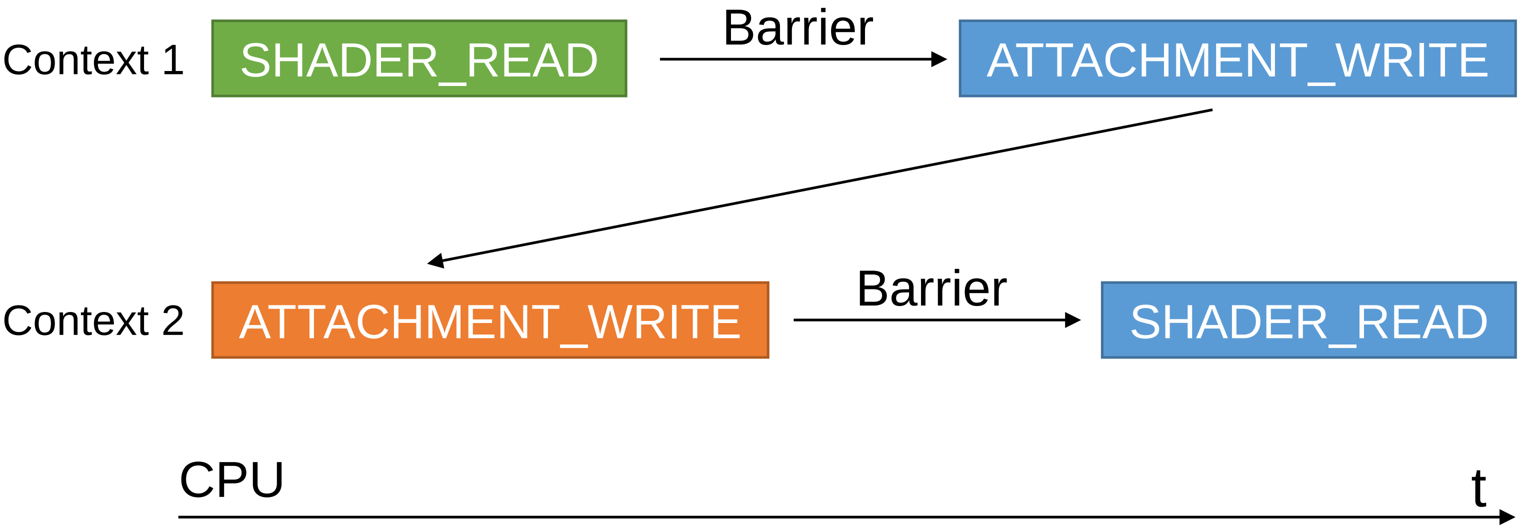
Image layouts & barriers

- Image layouts were a big headache
  - 25+ barriers per frame
  - Hundreds of layout changes
- Combining as many barriers as possible
- Knowing last image state difficult
  - We only specify the new state in code
- But parallelism makes complete automatic tracking impossible

Image layouts & barriers

- Automatic tracking inside each context / CB
- Not many images used across CBs
- Start of frame: Set state for start of CB to fix up missing tracking
- End of frame:
  - Go over transitions & determine initial next frame state
  - Validate image transitions
- No vkCmdSetEvent/vkCmdWaitEvents right now

Image layouts & barriers



Memory

- Simple block allocator
  - Split into max 128 MB pieces
  - Try smaller allocation until allocation succeedes
  - Or falls back to system memory if allocations fail in VRAM
  - Resizable images allocated individually
- NVIDIA problematic under pressure (2GB)
  - Lots of fixes in driver by now
  - Use NV_dedicated_allocation if possible

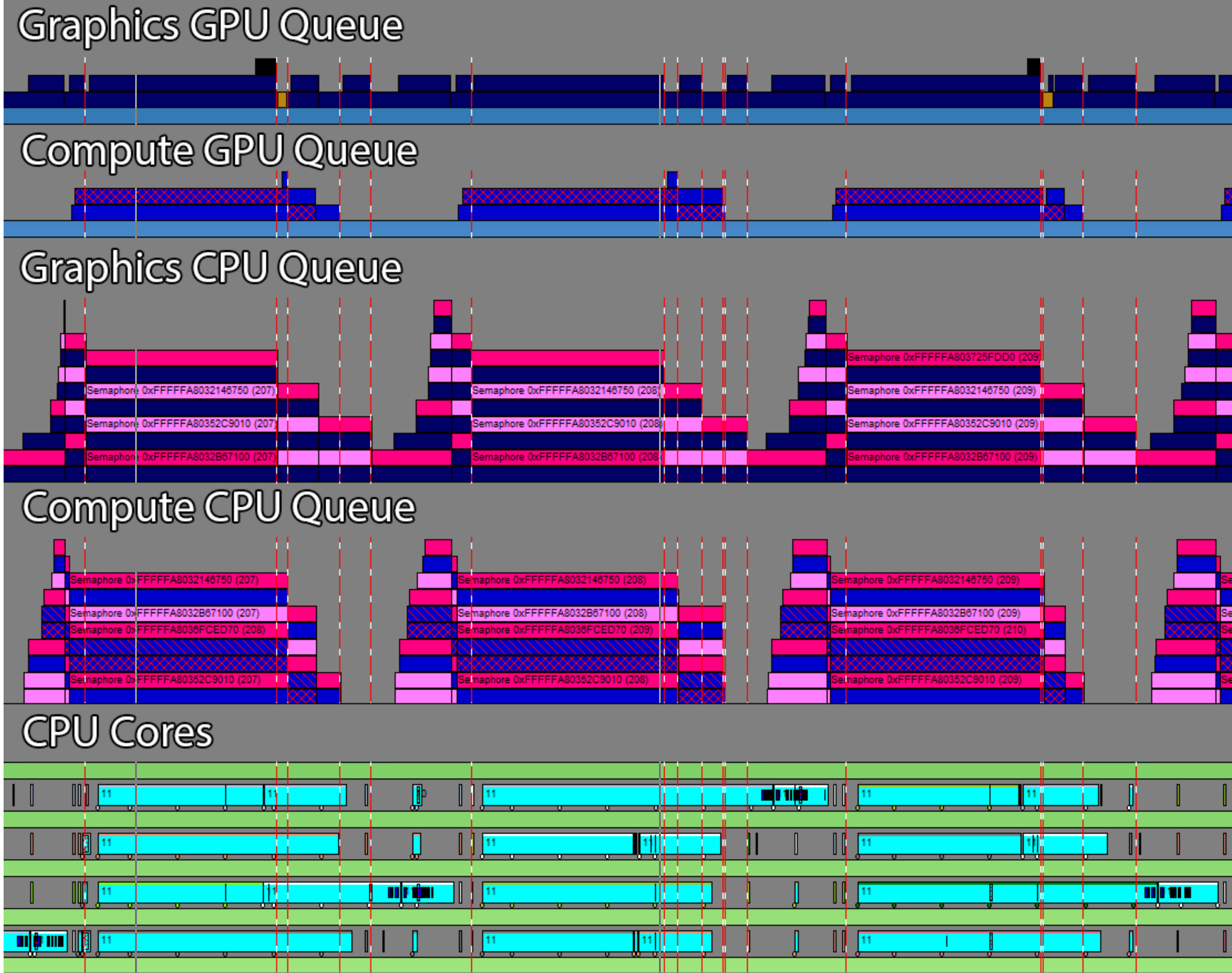
Memory

- All uploads through common manager
- Double buffered host staging memory
- Each staging buffer associated with
  - Command buffer
  - Fence
- If buffer is full, write fence at end of CB and submit
- Wait on fence before reuse
- Flush host visible ranges before graphics submits

Synchronization

- Double buffering everywhere
  - Wait for command buffer fence on CPU
  - Minimizes latency
- GPUView is your friend!
  - Much more useful than with OpenGL/DX11
- Swap chains are tricky
  - Make sure acquire & present always matching
  - Acquire as late as possible (avoids stalls)

- 🟪 Semaphore Wait
- 🟥 Semaphore Signal
- ▨ Present
- 🟦 Work (Submit)
- ▮ API Calls


Graphics GPU Queue
Compute GPU Queue
Graphics CPU Queue
Semaphore 0xFFFFFA803725FDD0 (209)
Semaphore 0xFFFFFA8032146750 (207)
Semaphore 0xFFFFFA8032146750 (208)
Semaphore 0xFFFFFA8032146750 (209)
Semaphore 0xFFFFFA80352C9010 (207)
Semaphore 0xFFFFFA80352C9010 (208)
Semaphore 0xFFFFFA80352C9010 (209)
Semaphore 0xFFFFFA8032B67100 (207)
Semaphore 0xFFFFFA8032B67100 (208
Semaphore 0xFFFFFA8032B67100 (209)
Compute CPU Queue
Semaphore 0xFFFFFA8032146750 (207)
Semaphore 0xFFFFFA8032146750 (208)
Semaphore 0xFFFFFA8032146750 (209)
Semaphore 0xFFFFFA8032B67100 (207)
Semaphore 0xFFFFFA8032B67100 (208)
Semaphore 0xFFFFFA8032B67100 (209)
Semaphore 0xFFFFFA8036FCED70 (208)
Semaphore 0xFFFFFA8036FCED70 (209)
Semaphore 0xFFFFFA8036FCED70 (210)
Semaphore 0xFFFFFA80352C9010 (207)
Semaphore 0xFFFFFA80352C9010 (208)
Semaphore 0xFFFFFA80352C9010 (209)
CPU Cores

Asynchronous Compute

- Useful for leveraging wasted GPU idle time
  - E.g. during shadow & depth pass
- GPU particles & post process
- Post process overlaps with beginning of next frame
  - Present from compute queue on AMD
  - NVIDIA still working on driver support
- Using SHARING_MODE_CONCURRENT for render targets
  - Careful, might be slower

Results

- Very pleased with performance gains
- 60%-70% in some scenes on AMD in GPU limit
  - Faster than OpenGL even without async/intrinsics
- NVIDIA GPU time about the same
- Render CPU limit is mostly gone
  - People reporting 60+ Hz in power saving mode
- Lots of potential

Future Work

- Prepare image barriers & layouts at beginning of frame
- Remove hashes and make high level code aware of states
- Know exactly what pipelines are used in game
- Better use of render passes (sub passes, layout transitions)

Future Work

- Split barriers (vkCmdSetEvent/vkCmdWaitEvents)
- Command buffer reuse (e.g. deferred passes & post process)
- More asynchronous compute
- Asynchronous transfers

Thanks

- Jean Geffroy, Tiago Sousa, Billy Khan & the whole team at id Software
- Baldur Karlsson for RenderDoc
- AMD and NVIDIA for help on Vulkan port
- Make sure to play the game!

We are Hiring

- Various openings across Zenimax Studios !
- Please visit https://jobs.zenimax.com