Optimizing ~~i965~~ for the Future

Kenneth Graunke
Intel Visual Technologies Team
& The Mesa Community

Driver CPU Overhead

- Graphics is always trying to push the limits
  - Time spent by the driver is time wasted for the app

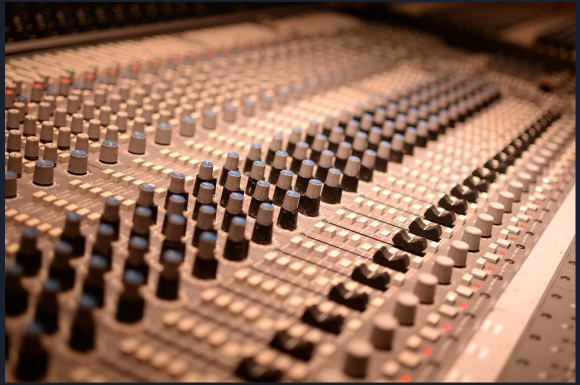
- In the spotlight lately
  - Vulkan has raised the bar (but lots of apps still using OpenGL...)
  - VR is a race against time, with no time to waste
  - Intel CPUs & integrated GPUs share a power envelope
    (Less CPU ⇒ More GPU watts)

- Draw time state upload has always been a volcanic hot path

State Upload:
A Comparison

OpenGL: a mutable state machine

- A million different knobs…
  - Vertex buffers & elements
  - Index buffers & primitive restart
  - Shaders
  - Image/buffer bindings
  - Samplers
  - Clipping, scissoring, viewports
  - Rasterization
  - Stream output
  - Tessellation
  - Multisampling
  - Blending
  - Color, depth, stencil buffers
  - Depth and stencil testing
  - Uniforms
  - Conditional rendering & queries
  - Topology
- GL context is mutable and continually in flux
- Applications dial in the settings they want…
- Draw, rinse, repeat…

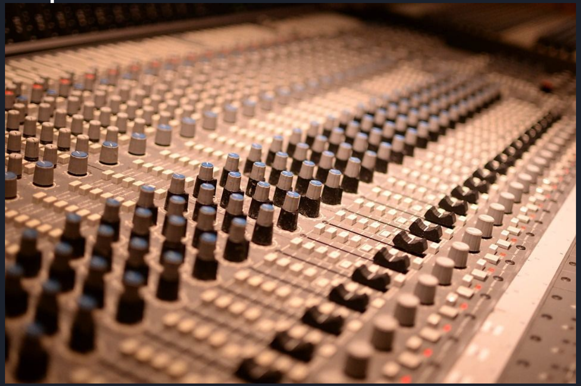


#1: State Streaming

- Translate on the fly… directly and efficiently
  - Track what state is dirty (which knobs were turned)…only emit what's needed
  - Applications try to minimize state changes, drivers track at a fine granularity

- "Not worth reusing state"
  - In theory, every draw could have brand new state
  - There is a cost…access context memory for cache lookup…miss…re-access…
  - Draw time becomes utterly volcanic

- i965 follows this approach



#2: Pre-baked Pipelines (Vulkan)

- Create immutable “pipeline objects” for each kind of object in the scene
  - Specify most of the state up-front, bake the GPU commands at creation
  - A bit of dynamic state remains
- Bind a pipeline, draw, repeat
  - Dirt cheap—submit pre-baked commands, no translation, discovery, etc.
- Fantastic if your app is set up for it… simple, efficient
  - But monolithic pipelines can be a challenge for very dynamic/mutable APIs
  - Basically the opposite model from the million-knob mutable context

#3: Gallium—Mesa’s Hybrid Model

- The model used by most Mesa drivers (notably not i965)
- Combines both state streaming and pre-baking

Gallium: CSOs

- Gallium uses “Constant State Objects” or CSOs
  - Immutable objects capturing part of the GPU state (say, blend state)
  - Cached for reuse across multiple draws
  - Drivers can associate their own state with a CSO
    (*create()* + *bind()* hooks… plus *set()* for dynamic state)

- Essentially a “pipeline in pieces”

- Drivers work almost entirely with CSO objects

Gallium: State Tracking

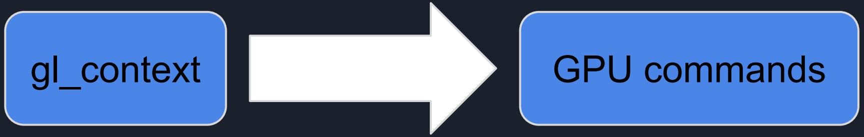
- Adapts a mutable API (GL) to the immutable Gallium world (CSOs)

- The Mesa *state tracker* looks at the mutable GL context, does dirty tracking, and ideally “rediscovers” cached CSOs for that state
    - “Hey, it looks like we’re drawing barrels again...”
    - If no hits, make new CSOs via *create()*...either way, *bind()*
    - Look familiar? *st/mesa* is actually a state streaming Mesa classic driver

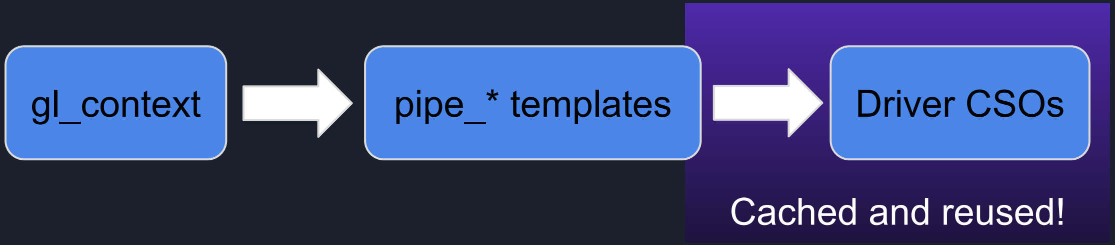
- Can distill state for the driver
    - Figure out Y-flipping parity, or ignore blending options on integer RTs…
    - This can increase CSO cache hits & simplify life for drivers

An Extra Layer?

Classic (State Streaming)



Gallium



Let’s look at **i965**...

i965 CPU usage

- We knew it could be better
  - Code is pretty efficient, but bad tracking means it executes too often
  - Most of our workloads were GPU bound, so we'd mostly focused there

- Remained a constant source of criticism
  - Various Intel teams
  - Twitter shaming from Vulkan fans
  - The last straw…data showing i965 was getting obliterated by radeonsi. (But this was actually constructive!)

- I decided to do something about it.

A (Worst) Case Study

- Say an application…binds a new texture
  (or really does anything to any texture…or VBOs for that matter…)

- i965 reacts: "_NEW_TEXTURE"?!
  - For each texture and storage image bound in any shader stage…
    - Retranslate SURFACE_STATE from scratch
    - Retranslate SAMPLER_STATE from scratch
    - Build new binding tables
  - Trigger any state-dependent shader program changes

- State reuse would help a ton…but that's actually hard
  - For surprising reasons

Memory MisManagement

- In the bad old days… one virtual GPU address space for all processes
  - Tell the kernel what buffers you have…it places them
  - Give it a list of pointers to patch up when it “relocates” buffers

- Intel GPUs save the last known GPU state in a “hardware context”
  - Back-to-back batches can inherit state instead of re-emitting commands
  - This includes pointers…to un-patched addresses.
  - Basically can’t inherit any state involving pointers… like SURFACE_STATE

- A lot of state uses a *base address* + offset to minimize pointers
  - But this means that all state must live in a single buffer
  - Need to re-emit due to lifetime problems

Modern Memory Management

- Modern hardware doesn’t need relocations
    - Gen8+ has 256TB of VMA… per-process
    - *Softpin* (Kernel 4.5+) allows userspace to assign virtual addresses
- Just assign addresses up front and never change them
    - Allows pre-baking or inheriting state involving pointers
- Can create 4GB “memory zones” for each base address
    - Use as many buffers as you want… no lifetime problems
    - Makes reusing state a ton easier

Architectural Overhaul, Please!

- Clearly need to save/reuse state
  - A pretty fundamental rework of the state upload code
  - No real infrastructure for this in the classic world
  - Need to modernize memory management

- Prototyping in the production driver was miserable
  - How to do it incrementally?
  - Need to handle every corner case right away
  - Enterprise kernel support makes modernizing miserable
  - Working on Gen11+ while thinking about Gen4+ is getting harder

- I realized…that Gallium solves these problems

WAT

In the past...

- Gallium never seemed to solve a problem we had
    - Didn’t magically get us from GL 2.1 to GL 4.5…tons of feature work…
    - Didn’t magically enable new hardware
    - Didn’t solve our driver performance problems at the time
    - Shader compiler story was entirely lacking, or far from viable (TGSI)… didn’t give us a proper GLSL frontend, or a modern SSA-based optimizer
    - None of us cared about implementing more APIs
    - Added abstraction layers that didn’t seem useful

- Massive pile of work
    - Spend over a year rewriting the driver for questionable benefits
    - Certainly not a silver bullet

Time to reconsider?

- Gallium has improved a lot
  - Tons of work on st/mesa efficiency
  - Threading (u_threaded_context)
  - NIR is now a viable option, replacing TGSI
  - Years of polish from the community

- i965 has become more modular thanks to our Vulkan efforts
  - ISL library for surface layout calculations
  - BLORP library for blits and resolves
  - Shader compiler backend

- Still…OMG effort…and would it even pay off?

The Big Science Experiment

- Last November… I decided to try it
  - Started from scratch—using the *noop* driver template, not *ilo*
  - Borrow ideas from our Vulkan driver
  - Focus on the latest hardware & kernels
  - Gain the freedom to experiment

- Keep it on the down low
  - Didn't want a ton of press / peanut gallery
  - Wanted to be able to scrap it if it wasn't panning out
  - Talked to the community on IRC… code in public since January

10 months later...

Introducing *iris_dri.so* (“Iris”)

- The science experiment was a success
    - A new Gallium-based 3D driver for Intel Iris GPUs
    - i965 reimagined for 2018 and rebuilt from the ground up
- Code available now:
    - https://gitlab.freedesktop.org/kwg/mesa/commits/iris
    - Primarily for driver developers… not ready for users yet
    - Zero TGSI was consumed in the development of this driver
- Requirements:
    - Only supports Gen9+ hardware (Skylake)
    - Kernel v4.16+ (could go back to v4.5 if needed)

Driver Status

- Iris is looking reasonably healthy
  - Currently passing 87% of Piglit
  - Can run some applications…others hit bugs

- Missing features
  - Color compression, fast clears, HiZ (critical for performance, not started)
  - Compute shaders & storage images (in progress)
  - Query objects (in progress) & sync objects (sketched)
  - Shader spilling (not started), on-disk shader cache (not started)

- Complete enough for measurements to be “in the right ball park”

Draw Overhead (from Piglit)

Draw calls per second (millions)	**i965**
DrawArrays (1 VBO, 0 UBO, 0) w/ no state change	**1.96 million**
DrawArrays (4 VBO, 0 UBO, 0) w/ no state change	**1.35** (69%)
DrawArrays (16 VBO, 0 UBO, 0) w/ no state change	**0.586** (30%)
DrawArrays (1 VBO, 8 UBO, 8 Tex) w/ 1 tex change	**0.271** (14%)
DrawElements (1 VBO, 0 UBO, 0) w/ no state chg.	**1.91 million**

Draw Overhead (from Piglit)

Draw calls per second (millions)	**i965**	**iris**	
DrawArrays (1 VBO, 0 UBO, 0) w/ no state change	**1.96 million**	**9.11 million**	4.65x
DrawArrays (4 VBO, 0 UBO, 0) w/ no state change	**1.35** (69%)	**9.07** (99%)	6.72x
DrawArrays (16 VBO, 0 UBO, 0) w/ no state change	**0.586** (30%)	**8.89** (97%)	15.2x
DrawArrays (1 VBO, 8 UBO, 8 Tex) w/ 1 tex change	**0.271** (14%)	**0.872** (9%)	3.21x
DrawElements (1 VBO, 0 UBO, 0) w/ no state chg.	**1.91 million**	**7.23 million**	3.79x

- On average 5.45x more draw calls per second

“wow those are quite good numbers”

There’s more: *u_threaded_context*

Draw calls per second (millions)	**i965**	**iris**	
DrawArrays (1 VBO, 0 UBO, 0) w/ no state change	**1.96 million**	**12.70 million**	6.48x
DrawArrays (4 VBO, 0 UBO, 0) w/ no state change	**1.35** (69%)	**12.50** (98%)	9.26x
DrawArrays (16 VBO, 0 UBO, 0) w/ no state change	**0.586** (30%)	**12.20** (97%)	**20.8x**
DrawArrays (1 VBO, 8 UBO, 8 Tex) w/ 1 tex change	**0.271** (14%)	**1.09** (8%)	4.02x
DrawElements (1 VBO, 0 UBO, 0) w/ no state chg.	**1.91 million**	**7.37 million**	3.85x

- But iris u_threaded_context support isn’t stable yet, so…<grain of salt>

Actual Performance?

- So… it has less CPU overhead. Most workloads are GPU bound.
- This microbenchmark is basically the ideal case for Gallium
  - Back-to-back draws hitting the CSO cache repeatedly
  - May be overstating the improvement… but, pretty representative, too?
- We need to measure real programs
  - One demo was ~19% faster on Apollolake
  - Many others are basically the same as i965
  - Currently measuring with HiZ/CCS disabled
  - Tons of risk—but the rewards seem worth it

Conclusion

- Debate settled!
    - i965 was the best classic driver, and Iris crushes it in terms of efficiency
    - Gallium is so much nicer to work with than Classic
    - We don’t regret the path we took, but are excited about the future
- Iris is a much better architecture for the future
- Mesa drivers can be fast, efficient, and competitive
    - Iris and RadeonSI have basically debunked the “Mesa is slow” myth

Next Steps

1. Make it work
   - Finish missing features, fix piles of bugs and push towards conformance
   - Test lots and lots of apps
   - Drop Gallium hacks so we can think about upstreaming it

2. Make it fast
   - Add missing performance features (color compression, HiZ, fast clears, …)
   - Use *FrameRetrace* on a whole bunch of apps, identify any gaps with i965

3. Dream about the future

Thank You!

Questions?

Backup

i965: Dirty Tracking

```
_NEW_TEXTURE, _NEW_BUFFERS, _NEW_PROGRAM, …

BRW_NEW_BATCH,
BRW_NEW_{VS,GS,TCS,TES,FS,CS}_PROG_DATA,
BRW_NEW_PRIMITIVE,
BRW_NEW_SURFACES,
BRW_NEW_BINDING_TABLE_POINTERS,
BRW_NEW_INDICES,
BRW_NEW_VERTICES,
BRW_NEW_DEFAULT_TESS_LEVELS,
BRW_NEW_PROGRAM_CACHE,
BRW_NEW_STATE_BASE_ADDRESS,
BRW_NEW_VUE_MAP_GEOM_OUT,
BRW_NEW_TRANSFORM_FEEDBACK,
BRW_NEW_RASTERIZER_DISCARD,
BRW_NEW_NUM_SAMPLES, ...
```

i965: Dirty Tracking

```
_NEW_TEXTURE, _NEW_BUFFERS, _NEW_PROGRAM, ...

BRW_NEW_BATCH,
BRW_NEW_{VS,GS,TCS,TES,FS,CS}_PROG_DATA,
BRW_NEW_PRIMITIVE,
BRW_NEW_SURFACES,
BRW_NEW_BINDING_TABLE_POINTERS,
BRW_NEW_INDICES,
BRW_NEW_VERTICES,
BRW_NEW_DEFAULT_TESS_LEVELS,
BRW_NEW_PROGRAM_CACHE,
BRW_NEW_STATE_BASE_ADDRESS,
BRW_NEW_VUE_MAP_GEOM_OUT,
BRW_NEW_TRANSFORM_FEEDBACK,
BRW_NEW_RASTERIZER_DISCARD,
BRW_NEW_NUM_SAMPLES, ...
```

These are oddly specific…
Bits for every scenario...

i965: Atoms

- Giant list of 70 “tracked state atoms” (dirty bits, function to emit)

```
static const struct brw_tracked_state genX(ps_blend) = {
   .dirty = {
      .mesa = _NEW_BUFFERS | _NEW_COLOR | _NEW_MULTISAMPLE,
      .brw = BRW_NEW_BLORP | BRW_NEW_CONTEXT |
             BRW_NEW_FRAGMENT_PROGRAM,
   },
   .emit = genX(upload_ps_blend)
};
```

- Each draw, walk list of 70 atoms, call function via pointer...
- Atoms may produce data and add dirty flags for later atoms (messy!)
- Plus bunches of ad-hoc stuff