Performance Analysis and Frame Debugging with FrameRetrace
Mark Janes, September 20, 2017
mark.a.janes@intel.com
About me:

- Working on Linux platforms since 2004, with a background on embedded devices.
- Joined Mesa in 2014, working on performance tools and automation.
GPU Performance Analysis Workflow

- Investigate system bottlenecks first
  - `top`, `gputop`, `rapl`
  - 100% GPU utilization with lower CPU utilization indicates a GPU-bound workload
  - TDP limited workloads cause GPU clock rate to fall.
  - `MESA_DEBUG=perf`
GPU Performance Analysis Workflow

- CPU Bound workloads have traditional tools
  - perf, callgrind, cachegrind, sysprof
- GPU performance analysis has a sparse landscape of Linux tools
  - AMD GPU PerfStudio, Nvidia Linux Graphics Debugger, QApiTrace
  - Leverage GPU hardware counters to quantify the cost of asynchronous GPU operations.
  - Live experimentation to see the effect on performance.
  - Deeply investigate a graphics workload.
GPU Tools stumbling blocks

- Generally hardware-specific
- Mostly closed source
- Linux support is an afterthought
- Tracing/retracing not reliable
- Low numbers of users
- Mesa support for GPU performance counters
FrameRetrace: frame analysis based on ApiTrace

- Widely used and high quality trace/retrace
- https://github.com/janesma/apitrace
- Cross-platform: Linux and Windows
- Upstream GPU Counter support in Mesa and Kernel for Haswell and later.
- Leveraged by Intel Mesa team to identify and fix several performance issues in i965.
FrameRetrace: frame analysis based on ApiTrace

- GPU Metrics for each render
- Render target visualization and experiments
- Api log
- Batch disassembly
- Shader analysis, live editing, and assembly
- Uniform constant display and live editing
- Render experiments
Demo
Other features

• Windows support provides important leverage for open source driver teams seeking to find Mesa performance gaps.

• Proposed features:
  – Display and modify GL State
  – Display texture state, with mip clamp experiment
  – Display geometry mesh
  – Depth buffer visualization
  – Overdraw / hotspot rendertarget visualization
  – UI improvements
  – Support for more hardware
Caveats

• **Currently a one-person side project, with help**
  - Thanks to Laura Ekstrand, Robert Bragg, Lionel Landerwelin, Eero Taminen, Pekka Jylhä-Ollila

• **Experiments require intricate state tracking**

• **Some workloads do not have single-frame run loops**
Questions?