Introduction to the Crash Diagnostic Layer

Jeremy Gebben
Senior Graphics Software Engineer
LunarG, Inc
About me

- At LunarG for the past 4 years
  - Validation and the synchronization2 emulation layer
- Ex Kernel Mode Driver dev
  - Early / mid Android era
  - GPU hangs roll down hill
- Ex Graphics Software Architect
  - “Hey HW team, why can’t we have nice things?”
- Lots of non-GPU embedded experience
  - Can you debug with LEDs?
Overview of Crash Diagnostic Layer

- Provides ‘glue code’ for debugging VK_ERROR_DEVICE_LOST
- New addition to the Vulkan SDK
  - Alpha quality!!!
  - Windows and Linux currently supported
- Works on many devices (that support debug extensions)
- Lightweight (~5% perf hit)
What can CDL do?

- Track forward progress of queue submission and command buffer processing
- Interpret fault information from the driver
- Manipulate the command stream
  - Add checkpoints, for command buffer forward progress
  - Add pipeline barriers
- Dump state to the filesystem in YAML format
- CANNOT debug within a shader invocation
## Extension support

<table>
<thead>
<tr>
<th>Extension</th>
<th>AMD</th>
<th>ARM</th>
<th>Intel</th>
<th>NVidia</th>
<th>Qualcomm</th>
<th>Samsung</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_AMD_buffer_marker</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>VK_AMD_device_coherent_memory</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>VK_NV_device_diagnostic_checkpoints</td>
<td></td>
<td>✅</td>
<td>✅</td>
<td></td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>VK_EXT_device_fault</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td></td>
<td>✅</td>
<td>✅</td>
</tr>
<tr>
<td>VK_EXT_device_address_binding_report</td>
<td>✅</td>
<td>✅</td>
<td>✅</td>
<td></td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>
What happens when a GPU crashes? (user view)

- Error dialog from app, driver, or OS
- Application just vanishes
- Screen goes black momentarily
- Screen goes black forever
- X session gets logged out
- Kernel panic / BSOD
- Device becomes unresponsive and very warm
What happens when a GPU crashes? (app view)

- Do something
- Vulkan returns DEVICE_LOST error
  - Restart rendering
  - Exit Nicely
  - Crash
What happens when a GPU crashes? (driver view)

- Timeout waiting for something (hang)
- GPU Error/Fault interrupt
- Identify “guilty” application
- Return DEVICE_LOST to any Vulkan calls
- Reset GPU hardware
- Continue processing work from “innocent” applications
Why is GPU crash debugging so hard?

- Pre-Vulkan graphics APIs didn’t consider crashing possible
  - GPU crash == driver or HW bug! Driver must validate EVERYTHING
  - Full screen games -> No concurrent use of the GPU, no fault recovery features
Why is GPU crash debugging so hard?

- Massive concurrency
  - How do you single step through 1 million fragment shader invocations?
  - How much state do you save after a crash?
  - Some problems go away when debugging
Why is GPU crash debugging so hard?

- Intellectual property
  - For some GPUs, hardware information is not publicly available
  - Large architecture differences between different GPU designs
  - Debug features aren’t high priority
How to use CDL

- Get the new SDK
- Start vkconfig
- Choose the Crash Diagnostic configuration
- Crash something
- Look at dump files
  - Linux: ~/cdl/…
  - Windows: %USERPROFILE%\cdl\…
- File Issues!
00:00:00.008 CDL INFO: Version 1.3.289 enabled. Start time tag: 2024-07-03-102527
00:00:00.008 CDL INFO: Begin Watchdog: 30000ms
00:00:00.076 CDL WARNING: No VK_AMD_device_coherent_memory extension, results may not be as accurate as possible.
00:00:00.076 CDL WARNING: No VK_EXT_device_fault extension, vendor-specific crash dumps will not be available.
00:00:00.076 CDL WARNING: No VK_EXT_device_address_binding_report extension, DeviceAddress information will not be available.
00:00:32.236 CDL INFO: Completed sequence number has impossible value: -1 submitted: 4700 VkQueue: 0x00000291204AD320[], VkSemaphore: 0x00000291208C6E70[]
00:00:32.237 CDL INFO: Completed sequence number has impossible value: -1 submitted: 0 VkQueue: 0x00000291206072C0[], VkSemaphore: 0x00000291208C6210[]
00:00:32.237 CDL ERROR: Device error encountered and log being recorded
   Output written to: "C:\Users\jgebb\cdl\2024-07-03-102527\cdl_dump.yaml"
Forward progress

- Evidence that the GPU is still processing work
- In the driver
  - Getting ‘work complete’ interrupts
  - Value of a counter changing in a register or memory counter
  - Lack of fault interrupts
- In an application
  - Various Vulkan wait calls completing
    - But… `vkDeviceWaitIdle()` and `vkQueueWaitIdle()` don’t ever time out
  - Timeline semaphore or fence state changing
Watchdog timer

- Monitors application activity and triggers a dump if application appears “stuck”
- Assumption: a non-stuck application will periodically submit new work to the GPU
- Reasons to turn off
  - If using a debugger, the watchdog may fire because the application is stopped
  - Some drivers have their own watchdog timer
  - Non-standard use cases like long running compute jobs
Submission state tracking

Split up submissions and write a per-queue timeline semaphore after every CB
Command Buffer checkpoints

- Reminder: multiple commands can be executing at the same time!
- Counters that track progress within a command buffer
- Write values somewhere after ‘interesting’ commands
- Written at the TOP_OF_PIPE and BOTTOM_OF_PIPE pipeline stages.
  - TOP_OF_PIPE - command has started executing
  - BOTTOM_OF_PIPE - command has finished execution
Command Buffer checkpoints (VK_AMD_buffer_marker)

- `vkCmdBeginRendering()`
- `vkCmdBindDescriptorSets()`
- `vkCmdBindVertexBuffer()`
- `vkCmdBindPipeline()`
- `vkCmdDraw()`
- `vkCmdEndRendering()`

- `vkCmdWriteBufferMarkerAMD(TOP_OF_PIPE, id)`
- `vkCmdWriteBufferMarkerAMD(BOTTOM_OF_PIPE, id)`
- `vkCmdDraw()`

- Writes arbitrary values to a buffer when the pipeline stage is reached by the command
- Requires VK_AMD_device_coherent_memory for accurate reporting during a crash
- But values for completed command buffers are always written
Command Buffer checkpoints
(VK_NV_device_diagnostic_checkpoints)

- A single command writes both the TOP_OF_PIPE and BOTTOM_OF_PIPE values
- App can call vkGetQueueCheckpointDataNV() to retrieve checkpoint info
- Checkpoints in a crashing CB are usually more accurate
- But checkpoints for completed CBs are not reported
CDL checkpoint output

- # Command:
  id: 17
  checkpointValue: 0x00000012
  name: vkCmdBeginDebugUtilsLabelEXT
  state: COMPLETED
  Labels:
  - Render Mesh
  Parameters: (...) 
  message: "'>>>>>>>>>>>>>> LAST COMPLETE COMMAND ^^^^^^^^^^^^^^^^^'"

(more commands)

- # Command:
  id: 24
  checkpointValue: 0x00000019
  name: vkCmdDrawIndexed
  state: INCOMPLETE
  labels:
  - Render Mesh
  parameters:
    indexCount: 8511627
    instanceCount: 1
    firstIndex: 0
    vertexOffset: 0
    firstInstance: 0
    internalState:
      pipeline: {}
      descriptorSets: []
  message: "'^^^^^^^^^^^^^^^ LAST STARTED COMMAND ^^^^^^^^^^^^^^^'"
GPU faults

● GPU Device Addresses are usually virtual memory
  ○ Most modern GPUs have some sort of MMU
  ○ Page faults are generated for invalid memory accesses

● VK_EXT_device_fault
  ○ Provides details about GPU page faults
  ○ Faulting address range, type of memory access (read, write, execute)
  ○ Can provide vendor specific fault information

● VK_EXT_device_address_binding_report
  ○ Provides notifications about changes to the GPU address space
  ○ Includes both user-visible objects (eg. buffer, image) and internal driver objects
CDL Device Fault output - buffer overrun

DeviceFaultInfo:
  description: GPU fault
  faultAddressRanges:
    - type: Invalid Read
      begin: 0x000000035330A600
      end: 0x000000035330AFFF
  priorAddressRecord:
    begin: 0x00000003531B4D00
    end: 0x000000035330A600
    type: VkDeviceMemory
    handle: 0x000001CDA3359F10[]
  currentlyBound: true
DeviceFaultInfo:
  description: GPU fault
  faultAddressRanges:
    - type: Invalid Read
      begin: 0x00000003531B4D00
      end: 0x00000003531B4DFF
  matchingAddressRecords:
    begin: 0x000000035330A600
    end: 0x000000035330A600
    type: VkDeviceMemory
    handle: 0x000001CDA3359F10[]
    currentlyBound: false
CDL Device Fault - bad address

DeviceFaultInfo:
  description: GPU fault
  faultAddressRanges:
    - type: Invalid Read
      begin: 0x00000BADDEADB000
      end: 0x00000BADDEADBFFF
  priorAddressRecord:
    begin: 0x00000003531B4D00
    end: 0x000000035330A600
    type: VkDeviceMemory
    handle: 0x000001CDA3359F10[]
    currentlyBound: true
Sync after commands

- Insert a pipeline barrier after each checkpoint
  - srcStageMask = dstStageMask = ALL_COMMANDS
  - srcAccessMask = MEMORY_WRITE, dstAccessMask = MEMORY_READ
- This limits how many commands can execute in parallel
- In one sample trace, this reduces the number of running commands from ~180 to 1
- This will make some GPU crashes stop reproducing, which likely means the application is missing synchronization
- Currently only works with dynamic rendering
Sync after commands

vkCmdBeginRendering()
vkCmdBindDescriptorSets()
vkCmdBindVertexBuffer()
vkCmdBindVertexBuffer()  
vkCmdDraw()  
vkCmdEndRendering()  

Deep copy pRenderingInfo

vkCmdDraw()  
vkCmdSetCheckpointNV(id)  
vkCmdEndRendering()  
vkCmdPipelineBarrier()  
vkCmdBeginRendering(saved_rendering_info)
Debug utils

- CDL supports VK_EXT_debug_utils and VK_EXT_debug_marker
- Object names are printed in the dump file
- Command labels are printed for every command
- Log messages can be sent to VK_EXT_debug_utils or VK_EXT_debug_report message callbacks
Thank you!

- Demo at the Khronos Networking Reception
- Included in the 1.3.290 SDK
- Code & Issues at [https://github.com/LunarG/CrashDiagnosticLayer](https://github.com/LunarG/CrashDiagnosticLayer)
- Thank you to the Google Stadia [Graphics Flight Recorder](https://github.com/LunarG/CrashDiagnosticLayer)