Keeping your staging buffer fixed size!

Presented at the Khronos Vulkanised 2023 Conference

Erfan Ahmadi
Lead Software Engineer
e.ahmadi@devsh.eu
What is a “staging” buffer

● General Truth
  ○ Helps with transferring data from CPU to GPU and vice versa.
  ○ Helps consumer of the data get it in the most convenient (closest) location for read access.
  ○ All staging buffers need to be **HOST_VISIBLE**.

● Our design choice
  ○ For uploading resources to GPU, it is usually used to transfer data to **DEVICE_LOCAL** memory which is not **HOST_VISIBLE**, Because **HOST_VISIBLE** & **DEVICE_LOCAL** is a very limited resource if present at all.

● UMA/Mobile Note: Obviously if you don't incur any penalties from your final buffer living over **HOST_VISIBLE** and **DEVICE_LOCAL**, you can stream straight into that, but Vulkan image layouts are not standardized so you still need to “stage” images
Our system in Nabla

- Deterministic and No memory spikes or growing/shrinking
- Push arbitrary sized image copies across with arbitrary row and slice alignments
- Taking advantage of device properties
  - optimalBufferCopyRowPitchAlignment, optimalBufferCopyOffsetAlignment
- Respects the specification and physical device limits.
  - minImageTransferGranularity, nonCoherentAtomSize
- color format conversion while streaming
- * transient data streaming buffer
- * auto-submission on overflow
Our system in Nabla
Transient Data Streaming Buffer

- Combine general purpose allocator with **a Buffer and Its Memory** (our staging buffer)
- Deallocations are deferred using a deferred event handler.
  - Event can hold onto a refcounted object until they are processed.
  - Command Buffers track refcounted resources they use.
- CullFrees function.
- Thread-Safe
  - We may use a Circular Buffer if there was only one timeline
Overview of Upload Loop

while (!uploadFinished)
{
    size = min(neededSizeToFinishTransferInOneSubmit, maximumPossibleAllocationSize);
allocation = transientDataStreamingBuffer.allocate(size, alignment);
if (allocation)
{
    cpuCopy(allocation.ptr, userData, uploadSize);
    commandBuffer.copyBuffer(...); or commandBuffer.copyBufferToImage(...);
    transientDataStreamingBuffer.requestDeferredDeallocation(allocation, fence, commandBuffer);
}
else
{
    queue.submit(submitInfo);
    device.waitForFence(fence);
    transientDataStreamingBuffer.cullFrees(/*heuristics*/); // or use VK_KHR_timeline_semaphore
    device.resetFence(fence);
    commandBuffer.begin(ONE_TIME_SUBMIT_BIT);
}
}
Auto Submission on Overflow

- A single resource copy may finalize in multiple queue submissions.
- User does not need to care about how many submissions are necessary
  - They tell us what `SubmitInfo` they would use assuming no overflow
  - The final submission is not our responsibility, but we return a modified `SubmitInfo` to take the below into account
  - Waiting for any semaphores must happen on the first submission only.
  - Signalling any semaphores must happen on the last submission only (User’s responsibility).
- The command buffer:
  - Needs to be “resettable” because it will need to reset in the case of implicit auto submission.
  - Needs to **begin** with ONE_TIME_SUBMIT because the recorded commands are invalid for reuse.
Future Improvements using **VK_KHR_synchronization2**

- With *binary* semaphores/fences, the deferred event handler will suffer a lost wake up if a fence latching a deferred free is reset before it is polled.

- With **VK_KHR_timeline_semaphore** we know the event (upload or download a chunk of data) has happened by using *vkWaitSemaphores* and our reference value when submitted.

  (and using VkTimelineSemaphoreSubmitInfo when submitting)
Nabla Property Pools = GPU ECS

- Scattered Streaming Writes/Reads
Questions?