Pipeline Barriers: A case of Gate-ing and Wait-ing

From the Command Buffer Notes:
These are the Commands that can be entered into the Command Buffer, I

- vkCmdExecuteCommands(commandBuffer, commandBufferCount, pCommandBuffers)
- vkCmdDrawIndirectCountAMD(commandBuffer, stride)
- vkCmdDrawIndirect(commandBuffer, stride)
- vkCmdDrawIndexedIndirect(commandBuffer, stride)
- vkCmdDrawIndexed(commandBuffer, indexCount, instanceCount, firstIndex, vertexOffset, firstInstance)
- vkCmdDispatch(commandBuffer, groupCountX, groupCountY, groupCountZ)
- vkCmdDebugMarkerEndEXT(commandBuffer)
- vkCmdCopyQueryPoolResults(commandBuffer, flags)
- vkCmdCopyBufferToImage(commandBuffer, pRegions)
- vkCmdCopyBuffer(commandBuffer, pRegions)
- vkCmdClearDepthStencilImage(commandBuffer, pRanges)
- vkCmdClearColorImage(commandBuffer, pRanges)
- vkCmdClearAttachments(commandBuffer, attachmentCount, pRects)
- vkCmdBindVertexBuffers(commandBuffer, firstBinding, bindingCount, pOffsets)
- vkCmdBindPipeline(commandBuffer, pipeline)
- vkCmdBindDescriptorSets(commandBuffer, pDynamicOffsets)
- vkCmdBeginRenderPass(commandBuffer, pContents)
- vkCmdBeginQuery(commandBuffer, flags)

From the Command Buffer Notes:
These are the Commands that can be entered into the Command Buffer, II

- vkCmdSetStencilCompareMask(commandBuffer, faceMask, compareMask)
- vkCmdWriteTimestamp(commandBuffer, pipelineStage, queryPool, query)
- vkCmdWaitEvents(commandBuffer, eventCount, pEvents, dstStageMask, srcStageMask, memoryBarrierCount, pMemoryBarriers, bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers)
- vkCmdSetViewport(commandBuffer, firstViewport, viewportCount, pViewports)
- vkCmdSetStencilWriteMask(commandBuffer, faceMask, writeMask)
- vkCmdSetStencilReference(commandBuffer, faceMask, reference)
- vkCmdSetScissor(commandBuffer, firstScissor, scissorCount, pScissors)
- vkCmdSetLineWidth(commandBuffer, lineWidth)
- vkCmdSetEvent(commandBuffer, event, stageMask)
- vkCmdSetDiscardRectangleEXT(commandBuffer, firstDiscardRectangle, discardRectangleCount, pDiscardRectangles)
- vkCmdSetDeviceMaskKHX(commandBuffer, deviceMask)
- vkCmdSetViewportWScalingNV(commandBuffer, firstViewport, viewportCount, pViewportWScalings)
- vkCmdSetDepthBounds(commandBuffer, minDepthBounds, maxDepthBounds)
- vkCmdReserveSpaceForCommandsNVX(commandBuffer, pReserveSpaceInfo)
- vkCmdPushDescriptorSetWithTemplateKHR(commandBuffer, descriptorUpdateTemplate, layout, set, pData)
- vkCmdPushDescriptorSetKHR(commandBuffer, pipelineBindPoint, layout, set, descriptorWriteCount, pDescriptorWrites)
- vkCmdFillBuffer(commandBuffer, dstBuffer, dstOffset, size, data)
- vkCmdSetDepthBias(commandBuffer, depthBiasConstantFactor, depthBiasClamp, depthBiasSlopeFactor)
- vkCmdResolveImage(commandBuffer, srcImage, srcImageLayout, dstImage, dstImageLayout, regionCount, pRegions)
- vkCmdResetEvent(commandBuffer, event, stageMask)
- vkCmdPipelineBarrier(commandBuffer, srcStageMask, dstStageMask, dependencyFlags, memoryBarrierCount, pMemoryBarriers, bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers)
- vkCmdNextSubpass(commandBuffer, contents)

A Pipeline Barrier is a way to establish a memory dependency between commands that were submitted before the barrier and commands that are submitted after the barrier.

From the Command Buffer Notes:
These are the Commands that can be entered into the Command Buffer, III

- vkCmdPipelineBarrier(commandBuffer, srcStageMask, dstStageMask, dependencyFlags, memoryBarrierCount, pMemoryBarriers, bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers)

Potential Memory Race Conditions that Pipeline Barriers can Prevent

1. Write-then-Read (WtR) – the memory read in one operation hasn’t yet finished before another operation starts overwriting that memory
2. Read-then-Write (RtW) – the memory write in one operation starts overwriting the memory that another operation’s read needs to use
3. Write-then-Write (WtW) – two operations start overwriting the same memory and the end result is non-deterministic

Note: there is no problem with Read-then-Read (RtR) as no data has been changed

viKmPipelineBarrier | Function Call

The Scenario
The Scenario

1. The cross-streets are named after pipeline stages
2. All traffic lights start out green
3. There are special sensors at all intersections that will know when the first car in the src group enters that intersection
4. There are connections from those sensors to the traffic lights so that when the first car in the src group enters its intersection, the proper dst traffic light can be turned back to green
5. When the last car in the src group enters that intersection, the proper dst traffic light can be turned green
6. The Vulkan command pipeline ordering is this: (1) the src cars get released, (2) the dst cars get released (which end up being stopped by a red light somewhere), (3) the last car in the src group enters its intersection, the proper dst traffic light can be turned back to green

Pipeline Stages and what Access Operations can Happen There

Pipeline Stage Masks – Where in the Pipeline is this Memory Data being Generated or Consumed?

Access Masks – What are you Interested in Generating or Consuming this Memory for?
Example: Don’t read a buffer back to the host until a shader is done writing it.

Example: Be sure we are done writing an output image before using it.

### VkImageLayout – How an Image gets Laid Out in Memory

- **Uniform**
  - `VK_IMAGE_LAYOUT_GENERAL`
  - `VK_IMAGE_LAYOUT_PRESENT_SRC_KHR`
  - `VK_IMAGE_LAYOUT_PREINITIALIZED`
  - `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`
  - `VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL`
  - `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`
  - `VK_IMAGE_LAYOUT_UNDEFINED`

- **Transfer**
  - `VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL`
  - `VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL`
  - `VK_IMAGE_LAYOUT_UNDEFINED`

- **Shader Read**
  - `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`

- **Shader Write**
  - `VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL`

- **Vertex Attribute**
  - `VK_IMAGE_LAYOUT_VERTEX_ATTRIBUTE_READ_BIT`

- **Index**
  - `VK_IMAGE_LAYOUT_INDEX_READ_BIT`

- **Indirect Command**
  - `VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL`

- **Stage**
  - `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`
  - `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT`
  - `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT`
  - `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT`
  - `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
  - `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT`
  - `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
  - `VK_PIPELINE_STAGE_VERTEX_SHADER_BIT`
  - `VK_PIPELINE_STAGE_VERTEX_INPUT_BIT`
  - `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT`
  - `VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT`

- **Access**
  - `VK_ACCESS_MEMORY_WRITE_BIT`
  - `VK_ACCESS_MEMORY_READ_BIT`
  - `VK_ACCESS_HOST_WRITE_BIT`
  - `VK_ACCESS_HOST_READ_BIT`
  - `VK_ACCESS_TRANSFER_READ_BIT`
  - `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT`
  - `VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT`
  - `VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT`
  - `VK_ACCESS_COLOR_ATTACHMENT_READ_BIT`
  - `VK_ACCESS_SHADER_READ_BIT`
  - `VK_ACCESS_INPUT_ATTACHMENT_READ_BIT`
  - `VK_ACCESS_UNIFORM_READ_BIT`
  - `VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT`
  - `VK_ACCESS_INDEX_READ_BIT`
  - `VK_ACCESS_INDIRECT_COMMAND_READ_BIT`

- **Command**
  - `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`
  - `VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT`
  - `VK_PIPELINE_STAGE_HOST_BIT`
  - `VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT`
  - `VK_PIPELINE_STAGE_TRANSFER_BIT`
  - `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`
  - `VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT`
  - `VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT`
  - `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT`
  - `VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT`
  - `VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT`
  - `VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT`
  - `VK_PIPELINE_STAGE_VERTEX_SHADER_BIT`
  - `VK_PIPELINE_STAGE_VERTEX_INPUT_BIT`
  - `VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT`
  - `VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT`