Synchronization

Remember the Overall Block Diagram?

Application

Instance

Physical Device

Logical Device

Semaphore

Command Buffer

Semaphores

- Used to control readiness of resources within one queue or across different queues belonging to the same logical device
- You create them, and give them to a Vulkan function which sets them. Later on, you tell a Vulkan function to wait on this particular semaphore
- You don’t end up setting, resetting, or checking the semaphore yourself
- Semaphores must be initialized (“created”) before they can be used

Ask for Something

The program continues

Try to Use the Something

Semaphore
Creating a Semaphore

```c
// VkSemaphoreCreateInfo
vk::SemaphoreCreateInfo vsci;

vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
// vsci.pNext = nullptr;

// VkSemaphore correspondence
result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &semaphore );
```

Semaphores Example during the Render Loop

```c
// VkSemaphore imageReadySemaphore;

vk::SemaphoreCreateInfo vsci;

vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
// vsci.pNext = nullptr;

// VkSemaphore correspondence
result = vkCreateSemaphore( LogicalDevice, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore );

// imageReadySemaphore;
uint32_t nextImageIndex;

vkAcquireNextImageKHR( LogicalDevice, IN SwapChain, IN UINT64_MAX, 
// imageReadySemaphore, 
IN VK_NULL_HANDLE, OUT &nextImageIndex );

// VkPipelineStageFlags waitAtBottom = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;

vk::SubmitInfo vsi;

vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
// vsci.pNext = nullptr;

// VkSubmitInfo correspondence
vsi.waitSemaphoreCount = 1;

// vsci.pWaitSemaphores = &imageReadySemaphore; 

vsi.pWaitDstStageMask = &waitAtBottom;

// VkSubmitInfo correspondence
vsi.commandBufferCount = 1;

// vsi.pCommandBuffers = &CommandBuffers[nextImageIndex];

vsi.signalSemaphoreCount = 0;

// vsi.pSignalSemaphores = (VkSemaphore) nullptr;

result = vkQueueSubmit( presentQueue, 1, IN &vsi, IN renderFence );
```

Fences

- Used to synchronize the application with commands submitted to a queue
- Announces that queue-submitted work is finished
- Much finer control than semaphores
- You can un-signal, signal, test or block-while-waiting

```c
#define VK_FENCE_CREATE_UNSIGNALED_BIT 0

vk::FenceCreateInfo vfci;

vfci.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
// vfci.pNext = nullptr;

// VkFenceCreateInfo correspondence
vfci.flags = VK_FENCE_CREATE_UNSIGNALED_BIT; // = 0

vkFenceCreateInfo fence;
result = vkCreateFence( LogicalDevice, IN &vfci, PALLOCATOR, OUT &fence );

// returns right away:
result = vkGetFenceStatus( LogicalDevice, IN fence );

// result = VK_SUCCESS means it has signaled
// result = VK_NOT_READY means it has not signaled

// blocks:
result = vkWaitForFences( LogicalDevice, 1, IN fence, waitForAll, timeout );

// = VK_TRUE: wait for all fences in the list
// = VK_FALSE: wait for any one fence in the list
// timeout is a uint64_t timeout in nanoseconds  (could be 0, which means to return immediately)

// timeout can be up to UINT64_MAX = 2^64 - 1 nanoseconds (i.e. 580+ years)

// result = VK_SUCCESS means it returned because a fence (or all fences) signaled
// result = VK_TIMEOUT means it returned because the timeout was exceeded
```

---

9/17/2019
Fence Example

```c
VkFence renderFence;
vkCreateFence( LogicalDevice, &vfci, PALLOCATOR, OUT &renderFence);

VkPipelineStageFlags waitAtBottom = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;
VkQueue presentQueue;
vkGetDeviceQueue( LogicalDevice, FindQueueFamilyThatDoesGraphics( ), 0, OUT &presentQueue );

VkSubmitInfo vsi;
vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
vsi.pNext = nullptr;
vsi.waitSemaphoreCount = 1;
vsi.pWaitSemaphores = &imageReadySemaphore;
vsi.pWaitDstStageMask = &waitAtBottom;
vsi.commandBufferCount = 1;
vsi.pCommandBuffers = &CommandBuffers[nextImageIndex];
vsi.signalSemaphoreCount = 0;
vsi.pSignalSemaphores = (VkSemaphore) nullptr;

result = vkQueueSubmit( presentQueue, 1, IN &vsi, IN renderFence);
...
result = vkWaitForFences( LogicalDevice, 1, IN &renderFence, VK_TRUE, UINT64_MAX);
...
result = vkQueuePresentKHR( presentQueue, IN &vpi);
```

Events

- Events provide even finer-grained synchronization
- Events are a primitive that can be signaled by the host or the device
- Can even signal at one place in the pipeline and wait for it at another place in the pipeline
- Signaling in the pipeline means "signal me as the last piece of this draw command passes that point in the pipeline".
- You can signal, un-signal, or test from a vk function or from a vkCmd function
- Can wait from a vkCmd function

Controlling Events from the Host

```c
VkEventCreateInfo veci;
veci.sType = VK_STRUCTURE_TYPE_EVENT_CREATE_INFO;
veci.pNext = nullptr;
veci.flags = 0;
VkEvent event;
result = vkCreateEvent( LogicalDevice, IN &veci, PALLOCATOR, OUT &event);
result = vkSetEvent( LogicalDevice, IN event);
result = vkResetEvent( LogicalDevice, IN event);
result = vkGetEventStatus( LogicalDevice, IN event);
// result = VK_EVENT_SET: signaled
// result = VK_EVENT_RESET: not signaled
```

Note: the host cannot block waiting for an event, but it can test for it

Controlling Events from the Device

```c
result = vkCmdSetEvent( CommandBuffer, IN event, pipelineStageBits);
result = vkCmdResetEvent( CommandBuffer, IN event, pipelineStageBits);
result = vkCmdWaitEvents( CommandBuffer, 1, &event, srcPipelineStageBits, dstPipelineStageBits, memoryBarrierCount, pMemoryBarriers, bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers );
```

Note: the device cannot test for an event, but it can block

Memory barriers get executed after events have been signaled

Could be an array of events

Where signaled, where wait for the signal