Synchronization

Remember the Overall Block Diagram?

Application

Instance

Physical Device

Logical Device

Fence

Command Buffer

Semaphore

Semaphores

- Indicates that a batch of commands has been processed from a queue. Basically announces "I am finished!".
- You create one and give it to a Vulkan function which sets it. Later on, you tell another Vulkan function to wait for this semaphore to be signaled.
- 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

Try to Use that Something

Semaphore
Creating a Semaphore

```c
VkSemaphoreCreateInfo vsci;
vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
vsci.pNext = nullptr;
vsci.flags = 0;

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

This doesn't actually do anything with the semaphore – it just sets it up.

Semaphores Example during the Render Loop

```c
VkSemaphore imageReadySemaphore;
VkSemaphoreCreateInfo vsci;
vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
vsci.pNext = nullptr;
vsci.flags = 0;

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

uint32_t nextImageIndex;

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

VkPipelineStageFlags waitAtBottomOfPipe = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;
VkSubmitInfo vsi;

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

vsi.waitSemaphoreCount = 1;
vsi.pWaitSemaphores = &imageReadySemaphore;

// https://www.khronos.org/registry/vulkan/specs/1.2-headers/vulkan.h

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

Fences

- Used to synchronize CPU-GPU tasks.
- Used when the host needs to wait for the device to complete something big.
- Announces that queue-submitted work is finished.
- You can un-signal, signal, test or block-while-waiting.

```c
#define VK_FENCE_CREATE_UNSIGNALED_BIT 0

VkFenceCreateInfo vfci;

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

vfci.flags = VK_FENCE_CREATE_UNSIGNALED_BIT; // = 0
// VK_FENCE_CREATE_SIGNALED_BIT is only other option

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

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

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

// blocks the host from executing:
result = vkWaitForFences( LogicalDevice, IN fence, IN waitForAll, timeout );
```

Fences

- Set the fence
- Wait on the fence(s)
- Un-signal, signal, test or block-while-waiting
- Blocks the host from executing
Fence Example

```cpp
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 ); // don't present the image until done rendering
```

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

```cpp
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

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

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

Controlling Events from the Device

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

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

Memory barriers get executed after events have been signaled