### 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.

![Semaphore Diagram](image)

#### Creating a Semaphore

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

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

#### 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.

```cpp
#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

VkFence fence;
result = vkCreateFence( LogicalDevice, &vfci, PALLOCATOR, &fence );
// returns right away:
result = vkGetFenceStatus( LogicalDevice, fence );
// result = VK_SUCCESS means it has signaled
// result = VK_NOT_READY means it has not signaled
// blocks:
result = vkWaitForFences( LogicalDevice, 1, &fence, VK_TRUE, UINT64_MAX );
```

#### Semaphores Example during the Render Loop

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

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

uint32_t nextImageIndex;
vkAcquireNextImageKHR( LogicalDevice, SwapChain, UINT64_MAX, imageReadySemaphore, VK_NULL_HANDLE, &nextImageIndex );

VkPipelineStageFlags waitAtBottom = 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;
    vsi.pWaitDstStageMask = &waitAtBottom;
    vsi.commandBufferCount = 1;
    vsi.pCommandBuffers = &CommandBuffers[nextImageIndex];
    vsi.signalSemaphoreCount = 0;
    vsi.pSignalSemaphores = nullptr;

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

Could be an array of semaphores.
Fence Example

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 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 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

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);

if (result = VK_EVENT_SET: signaled
    // result = VK_EVENT_RESET: not signaled

Note: the host cannot block waiting for an event

Controlling Events from the Device

result = vkCmdSetEvent(CommandBuffer, IN event, pipelineStageBits);

result = vkCmdResetEvent(CommandBuffer, IN event, pipelineStageBits);

result = vkCmdWaitEvents(CommandBuffer, 1, &event,

Where signaled, where wait
for the signal

Memory barriers get executed

Note: the device cannot test for an event

result = VK_PIPELINE_STAGE TopOfPipe_BIT

result = vkCmdWaitEvents(CommandBuffer, 1, &event, pipelineStageBits,

result = vkCmdWaitEvents(CommandBuffer, 1, &event, pipelineStageBits,

Could be an array of events