Synchronization

Where Synchronization Fits in the Overall Block Diagram

Creating a Semaphore

Synchronization Example during the Render Loop

Semaphores

• Used to synchronize work executing on different queues within 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

Try to Use that Something

Semaphore

• This doesn’t actually do anything with the semaphore – it just sets it up

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 );

uint32_t nextImageIndex;
vkAcquireNextImageKHR( LogicalDevice, IN SwapChain, IN UINT64_MAX, IN semaphore, IN VK_NULL_HANDLE, OUT &nextImageIndex );

VkPipelineStageFlags waitAtBottom = VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT;

VkSubmitInfo vsi;

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

You do this to wait for an image to be ready to be rendered into
Fences

• Used when the host needs to wait for the device to complete something big
• 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

#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, 1, IN &fence, waitForAll, timeout );
// waitForAll = VK_TRUE:   wait for all fences in the list
// waitForAll = 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 = 0xffffffffffffff (= 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

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

• 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

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

Controlling Events from the Device

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.