The Swap Chain

How We Think of OpenGL Framebuffers

Video

Driver

Update

Refresh

Depth-Buffer

Color Framebuffers

Swap Chains are arranged as a ring buffer

Swap Chains are tightly coupled to the window system.

After creating the Swap Chain in the first place, the process for using the Swap Chain is:

1. Ask the Swap Chain for an image
2. Render into it via the Command Buffer and a Queue
3. Return the image to the Swap Chain for presentation
4. Present the image to the viewer (copy to "front buffer")

What is a Swap Chain?

Because it has the word "chain" in it, let's try to visualize the Swap Chain as a physical chain.

A bicycle chain isn't far off. A bicycle chain goes around and around, each section of the chain taking its turn on the gear teeth, off the gear teeth, on, off, on, off, etc.

Because the Swap Chain is actually a ring buffer, the images in a Swap Chain go around and around too, each image taking its turn being drawn into, being presented, drawn into, being presented etc.

In the same way that bicycle chain links are "re-used", Swap Chain images get re-used too.

This is a pretty good analogy, except that there can be many more images in the ring buffer than are being shown here.
We Need to Find Out What our Display Capabilities Are

VulkanDebug.txt output:

vkGetPhysicalDeviceSurfaceCapabilitiesKHR:

VkSurfaceCapabilitiesKHR vsc = vkGetPhysicalDeviceSurfaceCapabilitiesKHR( PhysicalDevice, Surface );

uint32_t currentExtent = static_cast<uint32_t>( vsc.currentExtent.width );
uint32_t minImageExtent = static_cast<uint32_t>( vsc.minImageExtent.width );
uint32_t maxImageExtent = static_cast<uint32_t>( vsc.maxImageExtent.width );
uint32_t minImageCount = static_cast<uint32_t>( vsc.minImageCount );
uint32_t maxImageCount = static_cast<uint32_t>( vsc.maxImageCount );
uint32_t currentTransform = static_cast<uint32_t>( vsc.currentTransform );
uint32_t supportedCompositeAlpha = static_cast<uint32_t>( vsc.supportedCompositeAlpha );
uint32_t supportedTransforms = static_cast<uint32_t>( vsc.supportedTransforms );
uint32_t supportedRotationModes = static_cast<uint32_t>( vsc.supportedRotationModes );
uint32_t supportedImageUsageFlags = static_cast<uint32_t>( vsc.supportedImageUsageFlags );
uint32_t supportedUsageFlags = static_cast<uint32_t>( vsc.supportedUsageFlags );
uint32_t maxImageArrayLayers = static_cast<uint32_t>( vsc.maxImageArrayLayers );
uint32_t maxImageExtent = static_cast<uint32_t>( vsc.maxImageExtent );
uint32_t minImageExtent = static_cast<uint32_t>( vsc.minImageExtent );

uint32_t supported = VK_TRUE;
float tmeout = 0.0f;
uint32_t result = vkAcquireNextImageKHR( LogicalDevice, Swapchain, In &vsci, PALLOCATOR, Out &nextImageIndex, timeout );

uint32_t nextImageIndex = 0;

uint64_t timeout = UINT64_MAX;

if ( supported == VK_TRUE )
result = vkGetPhysicalDeviceSurfaceSupportKHR( PhysicalDevice, FindQueueFamilyThatDoesGraphics(), Surface, &supported );

VkBool32 supported = VK_TRUE;

uint32_t imageCount = 0;

vkCreateSemaphore( LogicalDevice, In &vsci, PALLOCATOR, Out &imageReadySemaphore );

result = vkCreateSwapchainKHR( LogicalDevice, In &vscci, PALLOCATOR, Out &Swapchain );


We Need to Find Out What our Display Capabilities Are
VkFenceCreateInfo vfci;
vfci.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
vfci.pNext = nullptr;
vfci.flags = 0;

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

VkQueue presentQueue;
vkGetDeviceQueue(LogicalDevice, FindQueueFamilyThatDoesGraphics(), 0, OUT &presentQueue);

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

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

result = vkWaitForFences(LogicalDevice, 1, IN &renderFence, VK_TRUE, UINT64_MAX);

VkPresentInfoKHR vpi;
vpi.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
vpi.pNext = nullptr;
vpi.waitSemaphoreCount = 0;
vpi.pWaitSemaphores = (VkSemaphore *)nullptr;
vpi.swapchainCount = 1;
vpi.pSwapchains = &SwapChain;
vpi.pImageIndices = &nextImageIndex;
vpi.pResults = (VkResult *) nullptr;

result = vkQueue呈KHR(presentQueue, IN &vpi);