The Swap Chain

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License
What is a Swap Chain?

In Vulkan, the Swap Chain manages one or more image objects that form a sequence of images that can be drawn into and presented to the user for viewing.

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
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 link in the chain taking its turn on the gear teeth, off the gear teeth, on, off, on, 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.
What is a Swap Chain?
Creating a Swap Chain

vkGetDevicePhysicalSurfaceCapabilities()

VkSurfaceCapabilities

VkSwapchainCreateInfo

surface
imageFormat
imageColorSpace
imageExtent
imageArrayLayers
imageUsage
imageSharingMode
preTransform
compositeAlpha
presentMode
clipped

minImageCount
maxImageCount
currentExtent
minImageExtent
maxImageExtent
maxImageArrayLayers
supportedTransforms
currentTransform
supportedCompositeAlpha

vkCreateSwapchain()

vkGetSwapChainImages()

vkCreateImageView()
VkSurfaceCapabilitiesKHR vsc;
vkGetPhysicalDeviceSurfaceCapabilitiesKHR( PhysicalDevice, Surface, OUT &vsc );
VkExtent2D surfaceRes = vsc.currentExtent;

VkSwapchainCreateInfoKHR vscci;
    vscci.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
    vscci.pNext = nullptr;
    vscci.flags = 0;
    vscci.surface = Surface;
    vscci.minImageCount = 2;  // double buffering
    vscci.imageFormat = VK_FORMAT_B8G8R8A8_UNORM;
    vscci.imageColorSpace = VK_COLORSPACE_SRGB_NONLINEAR_KHR;
    vscci.imageExtent.width = surfaceRes.width;
    vscci.imageExtent.height = surfaceRes.height;
    vscci.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
    vscci.preTransform = VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR;
    vscci.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;
    vscci.imageArrayLayers = 1;
    vscci.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
    vscci.queueFamilyIndexCount = 0;
    vscci.pQueueFamilyIndices = (const uint32_t *)nullptr;
    vscci.presentMode = VK_PRESENT_MODE_MAILBOX_KHR;
    vscci.oldSwapchain = VK_NULL_HANDLE;
    vscci.clipped = true;

result = vkCreateSwapchainKHR( LogicalDevice, IN &vscci, PALLOCATOR, OUT &SwapChain );
uint32_t imageCount;
result = vkGetSwapchainImagesKHR( LogicalDevice, IN SwapChain, OUT &imageCount, (VkImage *)nullptr );
PresentImages = new VkImage[ imageCount ];
result = vkGetSwapchainImagesKHR( LogicalDevice, SwapChain, OUT &imageCount, PresentImages );

// present views for the double-buffering:

PresentImageViews = new VkImageView[ imageCount ]; // better be 2
for( unsigned int i = 0; i < imageCount; i++ )
{
    VkImageViewCreateInfo vivci;
    vivci.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
    vivci.pNext = nullptr;
    vivci.flags = 0;
    vivci.viewType = VK_IMAGE_VIEW_TYPE_2D;
    vivci.format = VK_FORMAT_B8G8R8A8_UNORM;
    vivci.components.r = VK_COMPONENT_SWIZZLE_R;
    vivci.components.g = VK_COMPONENT_SWIZZLE_G;
    vivci.components.b = VK_COMPONENT_SWIZZLE_B;
    vivci.components.a = VK_COMPONENT_SWIZZLE_A;
    vivci.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
    vivci.subresourceRange.baseMipLevel = 0;
    vivci.subresourceRange.levelCount = 1;
    vivci.subresourceRange.baseArrayLayer = 0;
    vivci.subresourceRange.layerCount = 1;
    vivci.image = PresentImages[ i ];

    result = vkCreateImageView( LogicalDevice, IN &vivci, PALLOCATOR, OUT &PresentImageViews[ i ] );
}
uint32_t nextImageIndex;
uint64_t timeout = UINT64_MAX;

vkAcquireNextImageKHR( LogicalDevice, IN SwapChain, IN timeout, IN VK_NULL_HANDLE,
                      IN VK_NULL_HANDLE, OUT &nextImageIndex ); // semaphore, fence

... result = vkBeginCommandBuffer( CommandBuffers[nextImageIndex], IN &vcbbi );

... vkCmdBeginRenderPass( CommandBuffers[nextImageIndex], IN &vrpbi,
                         IN VK_SUBPASS_CONTENTS_INLINE );

... vkCmdEndRenderPass( CommandBuffers[nextImageIndex] );
vkEndCommandBuffer( CommandBuffers[nextImageIndex] );
Rendering into the Swap Chain, II

```cpp
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;

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

.
.

VkSubmitInfo vsi;
    vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
    vsi.pNext = nullptr;
    vsi.waitSemaphoreCount = 0;
    vsi.pWaitSemaphores = &SemaphoreImageAvailable;
    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
```
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 = vkQueuePresentKHR( presentQueue, IN &vpi );