The Swap Chain

Vulkan does not use the idea of a “back buffer”. So, we need a place to render into before moving an image into place for viewing. This is called the Swap Chain.

In essence, the Swap Chain manages one or more image objects that form a sequence of images that can be drawn into and then given to the Surface to be 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 (“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.
What is a Swap Chain?

We Need to Find Out What our Display Capabilities Are

VulkanDebug.txt output:

```
VkSurfaceCapabilitiesKHR vsc;
vkGetPhysicalDeviceSurfaceCapabilitiesKHR( PhysicalDevice, Surface, OUT &vsc );
vkGetPhysicalDeviceSurfaceCapabilitiesKHR:

minImageCount = 2 ; maxImageCount = 8
minImageExtent = 1024 x 1024
maxImageExtent = 1024 x 1024
maxImageArrayLayers = 1
supportedTransforms = 0x0001
currentTransform = 0x0001

supportedCompositeAlpha = 0x0001

** This Surface is supported by the Graphics Queue **

Found 2 Surface Formats:
0: 44 0 ( VK_FORMAT_B8G8R8A8_UNORM, VK_COLOR_SPACE_SRGB_NONLINEAR_KHR )
1: 50 0 ( VK_FORMAT_B8G8R8A8_SRGB, VK_COLOR_SPACE_SRGB_NONLINEAR_KHR )

Found 3 Present Modes:
0: 2 ( VK_PRESENT_MODE_FIFO_KHR )
1: 3 ( VK_PRESENT_MODE_FIFO_RELAXED_KHR )
2: 1 ( VK_PRESENT_MODE_MAILBOX_KHR )
```

We Need to Find Out What our Display Capabilities Are

Creating a Swap Chain

```
vkCreateSwapchain( )
```

SRGB gives more accurate perceived colors
MAILBOX more likely to avoid image tearing
Creating a Swap Chain

 VkSurfaceCapabilitiesKHR vscc;
 vkGetPhysicalDeviceSurfaceCapabilitiesKHR(PhysicalDevice, Surface, OUT &vscc);
 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.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 = VK_TRUE;
 result = vkCreateSwapchainKHR(LogicalDevice, IN &vscci, PALLOCATOR, OUT &SwapChain);

Creating the Swap Chain Images and Image Views

 uint32_t imageCount; // # of display buffers – 2? 3?
 result = vkGetSwapchainImagesKHR(LogicalDevice, IN SwapChain, OUT &imageCount, (VkImage *)nullptr);
 PresentImages = new VkImage[imageCount];
 result = vkGetSwapchainImagesKHR(LogicalDevice, SwapChain, OUT &imageCount, PresentImages);

Rendering into the Swap Chain, I

 VkSemaphoreCreateInfo vsci;
 vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
 vsci.pNext = nullptr;
 vsci.flags = 0;
 VkSemaphore imageReadySemaphore;
 result = vkCreateSemaphore(LogicalDevice, &vsci, PALLOCATOR, OUT &imageReadySemaphore);

Rendering into the Swap Chain, II

 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);

VkQueue presentQueue;
 vkGetDeviceQueue(LogicalDevice, &vfci, PALLOCATOR, OUT &renderFence);
 VkSubmitInfo vsi;
 vsi.sType = VK_STRUCTURE_TYPE_SUBMIT_INFO;
 vsi.pNext = nullptr;
 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
Rendering into the Swap Chain, III

```
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 = vkQueuePresentKHR( presentQueue, IN &vpi );
```