The Swap Chain

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

How OpenGL Thinks of Framebuffers

How Vulkan thinks of Framebuffers – the Swap Chain

What is a 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 images 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 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")

We Need to Find Out What our Display Capabilities Are

VulkanDebug.txt output for an Nvidia A6000:

```

The Swap Chain

We Need to Find Out What our Display Capabilities Are

VulkanDebug.txt output for an Nvidia A6000:

```

12/29/2022
Here's What the Vulkan Spec Has to Say About Present Modes, I

Creating a Swap Chain

```
void DeviceRequestSurfaceCapabilities()
{
  VkSurfaceCapabilitiesKHR vsc;
  vkGetDevicePhysicalSurfaceCapabilitiesKHR( 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 = 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 );

// present views for the double-buffering:
PresentImageViews = new VkImageView[imageCount];
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] );
}
```

Creating a Swap Chain

```
VkSemaphoreCreateInfo vsci;
vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
vsci.pNext = nullptr;

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

uint32_t nextImageIndex;

uint64_t timeout = UINT64_MAX;

vkAcquireNextImageKHR( LogicalDevice, IN SwapChain, IN timeout, IN imageReadySemaphore, IN VK_NULL_HANDLE, OUT &nextImageIndex );
```

Rendering into the Swap Chain, I

```
vSemaphoreCreateEvent = vkCreateSemaphoreKHR( LogicalDevice, INakovSwapChain, OUT &imageReadySemaphore );

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

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

vkCmdBindPipeline( CommandBuffers[nextImageIndex], VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipeline );
```

Here's What the Vulkan Spec Has to Say About Present Modes, II

Creating a Swap Chain

```
VkSurfaceCapabilitiesKHR vsc;
vkGetDevicePhysicalSurfaceCapabilitiesKHR( PhysicalDevice, Surface, OUT &vsc );
VkExtent2D surfaceRes = vsc.currentExtent;
```
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;
  vsi.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 = vkQueuePresentKHR(presentQueue, IN &vpi);

Rendering into the Swap Chain, II
Rendering into the Swap Chain, III