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

- Update
- Back
- Front
- Depth
- Refresh

How Vulkan Thinks of Framebuffers – the Swap Chain

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:

1. VulkanDebug.txt output:

- Update
- Back
- Front
- Depth
- Refresh

- VulkanDebug.txt output:

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 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:

1. VulkanDebug.txt output:

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 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”)
Creating a Swap Chain

1. **Creating the Swap Chain Images and Image Views**
   ```
   VkSwapchainCreateInfoKHR vscci;
   vscci.clipped = VK_TRUE;
   vscci.oldSwapchain = VK_NULL_HANDLE;
   vscci.pQueueFamilyIndices = (const uint32_t *)nullptr;
   vscci.queueFamilyIndexCount = 0;
   vscci.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
   vscci.imageArrayLayers = 1;
   vscci.compositeAlpha = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;
   vscci.imageUsage = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
   vscci.imageExtent.height = surfaceRes.height;
   vscci.imageExtent.width = surfaceRes.width;
   vscci.imageColorSpace = VK_COLORSPACE_SRGB_NONLINEAR_KHR;
   vscci.imageFormat = VK_FORMAT_B8G8R8A8_UNORM;
   vscci.minImageCount = 2;
   vscci.maxImageCount = 3;
   vscci.maxImageArrayLayers = 4;
   vscci.minImageArrayLayers = 0;
   vscci.imageExtent.aspectRatio = VK_IMAGE_ASPECT_COLOR_BIT;
   vscci.imageExtent.layerCount = 1;
   vscci.imageExtent.baseArrayLayer = 0;
   vscci.imageExtent.levelCount = 1;
   vscci.imageColorSpace = VK_COLORSPACE_SRGB_NONLINEAR_KHR;
   vscci.imageFormat = VK_FORMAT_B8G8R8A8_UNORM;
   vscci.viewType = VK_IMAGE_VIEW_TYPE_2D;
   vscci.flags = 0;
   vscci.pNext = nullptr;
   vscci.sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR;
   
   result = vkCreateSwapchainKHR(LogicalDevice, IN &vscci, PALLOCATOR, OUT &SwapChain);
   ```

2. **Rendering into the Swap Chain, I**
   ```
   VkSemaphoreCreateInfo vsci;
   vsci.pNext = nullptr;
   vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
   
   result = vkCreateSemaphore(LogicalDevice, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore);
   ```

3. **Rendering into the Swap Chain, II**
   ```
   VkPresentInfoKHR vpi;
   vpi.pImageIndices = &nextImageIndex;
   vpi.pSwapchains = &SwapChain;
   vpi.swapchainCount = 1;
   vpi.pWaitSemaphores = (VkSemaphore *)nullptr;
   vpi.waitSemaphoreCount = 0;
   vpi.pNext = nullptr;
   vpi.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
   vpi.pResults = (VkResult *) nullptr;
   
   result = vkQueuePresentKHR(presentQueue, IN &vpi);
   ```

4. **Rendering into the Swap Chain, III**
   ```
   VkFenceCreateInfo vfci;
   vfci.flags = 0;
   vfci.pNext = nullptr;
   vfci.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
   
   result = vkCreateFence(LogicalDevice, IN &vfci, PALLOCATOR, OUT &renderFence);
   ```

Creating a Swap Chain

1. **Creating the Swap Chain Images and Image Views**
   ```
   VkSemaphoreCreateInfo vsci;
   vsci.pNext = nullptr;
   vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
   
   result = vkCreateSemaphore(LogicalDevice, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore);
   ```

2. **Rendering into the Swap Chain, I**
   ```
   VkPresentInfoKHR vpi;
   vpi.pImageIndices = &nextImageIndex;
   vpi.pSwapchains = &SwapChain;
   vpi.swapchainCount = 1;
   vpi.pWaitSemaphores = (VkSemaphore *)nullptr;
   vpi.waitSemaphoreCount = 0;
   vpi.pNext = nullptr;
   vpi.sType = VK_STRUCTURE_TYPE_PRESENT_INFO_KHR;
   vpi.pResults = (VkResult *) nullptr;
   
   result = vkQueuePresentKHR(presentQueue, IN &vpi);
   ```

3. **Rendering into the Swap Chain, II**
   ```
   VkFenceCreateInfo vfci;
   vfci.flags = 0;
   vfci.pNext = nullptr;
   vfci.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
   
   result = vkCreateFence(LogicalDevice, IN &vfci, PALLOCATOR, OUT &renderFence);
   ```