




1




# The Swap Chain



**Oregon State University**  
Mike Bailey  
mjb@cs.oregonstate.edu



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/)

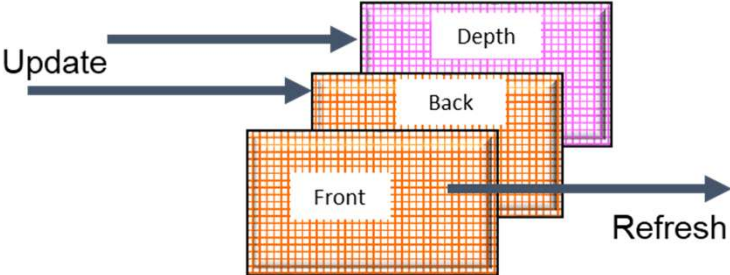


**Oregon State University**  
Computer Graphics


SwapChain.pptx mjb - December 21, 2022

2

## How OpenGL Thinks of Framebuffers

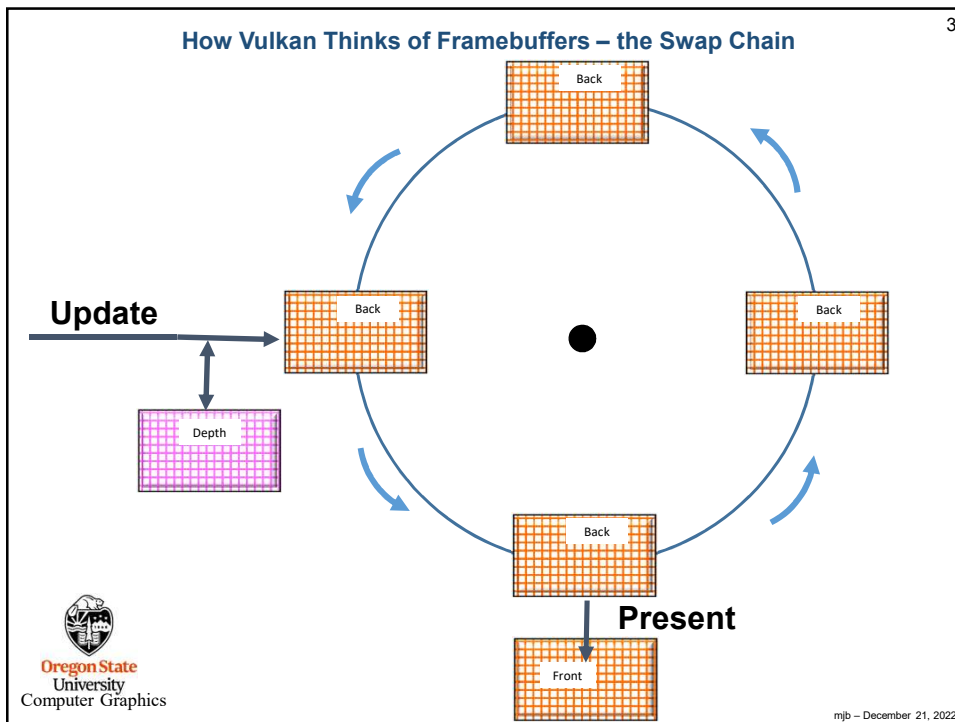


The diagram illustrates the flow of data in OpenGL framebuffers. It shows three overlapping buffers: a purple 'Depth' buffer at the top, an orange 'Back' buffer in the middle, and an orange 'Front' buffer at the bottom. Two arrows labeled 'Update' point from the left to the 'Depth' and 'Back' buffers. A single arrow labeled 'Refresh' points from the 'Front' buffer to the right.



**Oregon State University**  
Computer Graphics

mjb - December 21, 2022



### What is a Swap Chain?

4

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")



mjb – December 21, 2022

Oregon State University Computer Graphics

## We Need to Find Out What our Display Capabilities Are

5

```

VkSurfaceCapabilitiesKHR vsc;
vkGetPhysicalDeviceSurfaceCapabilitiesKHR( PhysicalDevice, Surface, OUT &vsc );
VkExtent2D surfaceRes = vsc.currentExtent;
fprintf( FpDebug, "\nvkGetPhysicalDeviceSurfaceCapabilitiesKHR:\n" );

...

VkBool32 supported;
result = vkGetPhysicalDeviceSurfaceSupportKHR( PhysicalDevice, FindQueueFamilyThatDoesGraphics( ), Surface, &supported );
if( supported == VK_TRUE )
    fprintf( FpDebug, "*** This Surface is supported by the Graphics Queue **\n" );

uint32_t formatCount;
vkGetPhysicalDeviceSurfaceFormatsKHR( PhysicalDevice, Surface, &formatCount, (VkSurfaceFormatKHR *) nullptr );
VkSurfaceFormatKHR * surfaceFormats = new VkSurfaceFormatKHR[ formatCount ];
vkGetPhysicalDeviceSurfaceFormatsKHR( PhysicalDevice, Surface, &formatCount, surfaceFormats );
fprintf( FpDebug, "\nFound %d Surface Formats:\n", formatCount )

...

uint32_t presentModeCount;
vkGetPhysicalDeviceSurfacePresentModesKHR( PhysicalDevice, Surface, &presentModeCount, (VkPresentModeKHR *) nullptr );
VkPresentModeKHR * presentModes = new VkPresentModeKHR[ presentModeCount ];
vkGetPhysicalDeviceSurfacePresentModesKHR( PhysicalDevice, Surface, &presentModeCount, presentModes );
fprintf( FpDebug, "\nFound %d Present Modes:\n", presentModeCount );

...

```

## We Need to Find Out What our Display Capabilities Are

6

### VulkanDebug.txt output for an Nvidia A6000:

\*\*\*\*\* Init08Swapchain \*\*\*\*\*

#### vkGetPhysicalDeviceSurfaceCapabilitiesKHR:

```

minImageCount = 2 ; maxImageCount = 8
currentExtent = 1024 x 1024
minImageExtent = 1024 x 1024
maxImageExtent = 1024 x 1024
maxImageArrayLayers = 1
supportedTransforms = 0x0001
currentTransform = 0x0001
supportedCompositeAlpha = 0x0001
supportedUsageFlags = 0x009f

```

#### vkGetPhysicalDeviceSurfaceSupportKHR:

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

#### Found 3 Surface Formats:

```

0: 44      0 VK_COLOR_SPACE_SRGB_NONLINEAR_KHR
1: 50      0 VK_COLOR_SPACE_SRGB_NONLINEAR_KHR
2: 64      0 VK_COLOR_SPACE_SRGB_NONLINEAR_KHR

```

#### Found 4 Present Modes:

```

0: 2 VK_PRESENT_MODE_FIFO_KHR
1: 3 VK_PRESENT_MODE_FIFO_RELAXED_KHR
2: 1 VK_PRESENT_MODE_MAILBOX_KHR
3: 0 VK_PRESENT_MODE_IMMEDIATE_KHR

```

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

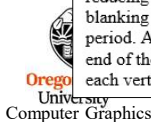
7

VK\_PRESENT\_MODE\_IMMEDIATE\_KHR specifies that the presentation engine does not wait for a vertical blanking period to update the current image, meaning this mode **may** result in visible tearing. No internal queuing of presentation requests is needed, as the requests are applied immediately.

VK\_PRESENT\_MODE\_MAILBOX\_KHR specifies that the presentation engine waits for the next vertical blanking period to update the current image. Tearing **cannot** be observed. An internal single-entry queue is used to hold pending presentation requests. If the queue is full when a new presentation request is received, the new request replaces the existing entry, and any images associated with the prior entry become available for re-use by the application. One request is removed from the queue and processed during each vertical blanking period in which the queue is non-empty.

VK\_PRESENT\_MODE\_FIFO\_KHR specifies that the presentation engine waits for the next vertical blanking period to update the current image. Tearing **cannot** be observed. An internal queue is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during each vertical blanking period in which the queue is non-empty. This is the only value of `presentMode` that is **required** to be supported.

VK\_PRESENT\_MODE\_FIFO\_RELAXED\_KHR specifies that the presentation engine generally waits for the next vertical blanking period to update the current image. If a vertical blanking period has already passed since the last update of the current image then the presentation engine does not wait for another vertical blanking period for the update, meaning this mode **may** result in visible tearing in this case. This mode is useful for reducing visual stutter with an application that will mostly present a new image before the next vertical blanking period, but may occasionally be late, and present a new image just after the next vertical blanking period. An internal queue is used to hold pending presentation requests. New requests are appended to the end of the queue, and one request is removed from the beginning of the queue and processed during or after each vertical blanking period in which the queue is non-empty.



mjb - December 21, 2022

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

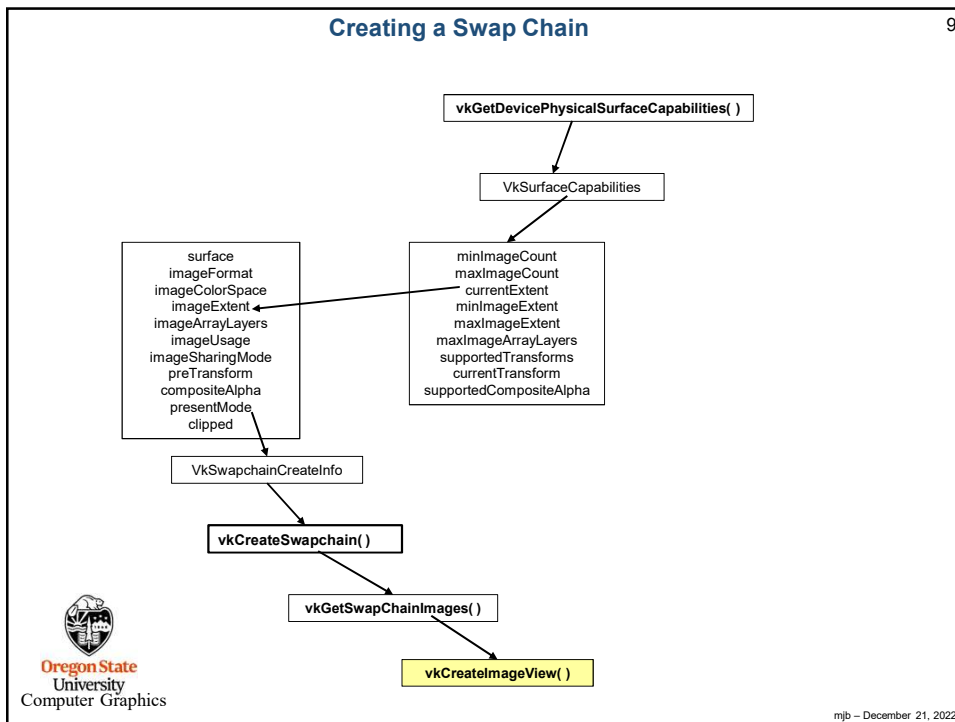
8

VK\_PRESENT\_MODE\_SHARED\_DEMAND\_REFRESH\_KHR specifies that the presentation engine and application have concurrent access to a single image, which is referred to as a *shared presentable image*. The presentation engine is only required to update the current image after a new presentation request is received. Therefore the application **must** make a presentation request whenever an update is required. However, the presentation engine **may** update the current image at any point, meaning this mode **may** result in visible tearing.

VK\_PRESENT\_MODE\_SHARED\_CONTINUOUS\_REFRESH\_KHR specifies that the presentation engine and application have concurrent access to a single image, which is referred to as a *shared presentable image*. The presentation engine periodically updates the current image on its regular refresh cycle. The application is only required to make one initial presentation request, after which the presentation engine **must** update the current image without any need for further presentation requests. The application **can** indicate the image contents have been updated by making a presentation request, but this does not guarantee the timing of when it will be updated. This mode **may** result in visible tearing if rendering to the image is not timed correctly.



mjb - December 21, 2022



### Creating a Swap Chain

10

```

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 = VK_TRUE;

result = vkCreateSwapchainKHR( LogicalDevice, IN &vscci, PALLOCATOR, OUT &SwapChain );
  
```

**Oregon State University Computer Graphics**

mjb - December 21, 2022

### Creating the Swap Chain Images and Image Views

11

```

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 ] );
}

```

022

### Rendering into the Swap Chain, I

12

```

VkSemaphoreCreateInfo vsci;
vsci.sType = VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO;
vsci.pNext = nullptr;
vsci.flags = 0;

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 );
...

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

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

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

vkCmdEndRenderPass( CommandBuffers[ nextImageIndex ] );
vkEndCommandBuffer( CommandBuffers[ nextImageIndex ] );

```

022

Oregon State  
University  
Computer Graphics

mjb - December 21, 2022

## Rendering into the Swap Chain, II

13

```

VkFenceCreateInfo
    vfc.i.sType = VK_STRUCTURE_TYPE_FENCE_CREATE_INFO;
    vfc.i.pNext = nullptr;
    vfc.i.flags = 0;

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

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

...

VkSubmitInfo
    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

```

Or  
University  
Computer Graphics

mjb - December 21, 2022

## Rendering into the Swap Chain, III

14

```

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

VkPresentInfoKHR
    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 );

```

  
Oregon State  
University  
Computer Graphics



mjb - December 21, 2022