1

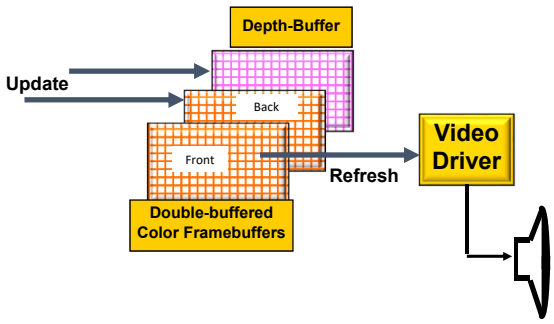
The Swap Chain

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

Oregon State University
Computer Graphics

2

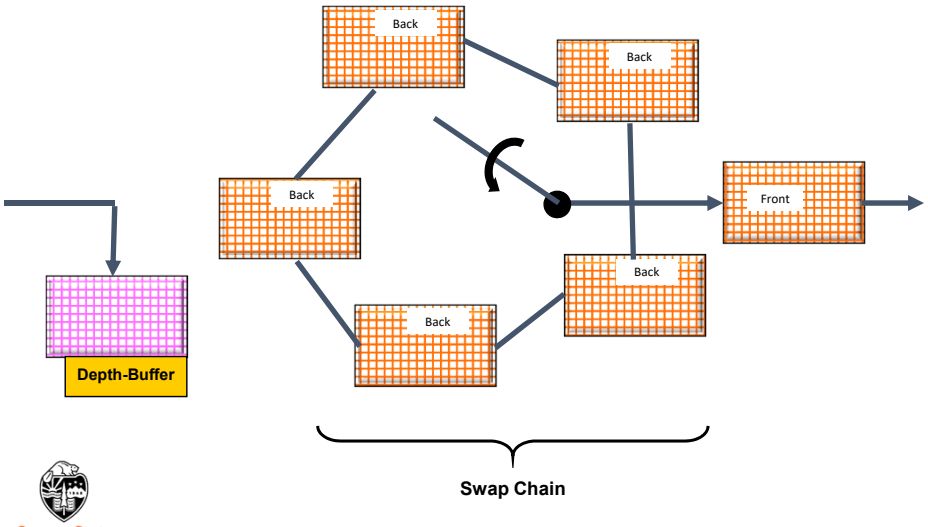
How We Think of OpenGL Framebuffers



Oregon State University
Computer Graphics

Vulkan Thinks of it as a Ring Buffer

3



Oregon State University Computer Graphics

mjb – September 18, 2018

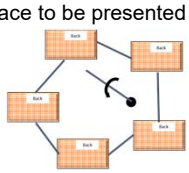
What is a Swap Chain?

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. The 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")

Oregon State University Computer Graphics

mjb – September 18, 2018

What is a Swap Chain?

5

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.

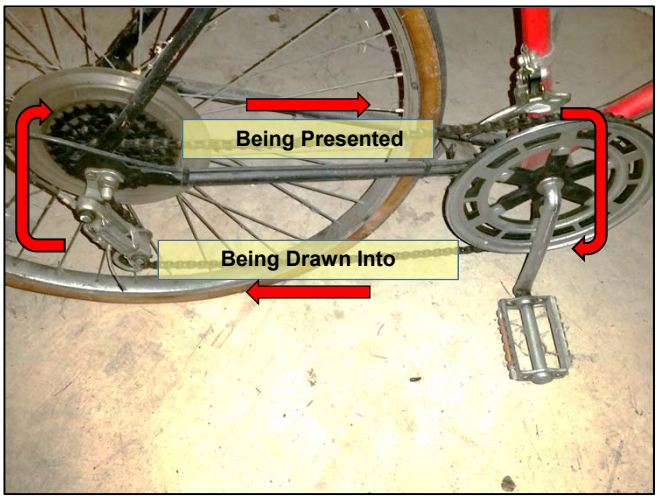
In the same way that bicycle chain links are “re-used”, Swap Chain images get re-used too.

Oregon State University Computer Graphics

mjb – September 18, 2018

What is a Swap Chain?

6



This is a pretty good analogy, except that there can be many more images in the ring buffer than are being shown here.

Oregon State University Computer Graphics

mjb – September 18, 2018

We Need to Find Out What our Display Capabilities Are


```
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 );

        . . .
```

Oregon State University
Computer Graphics

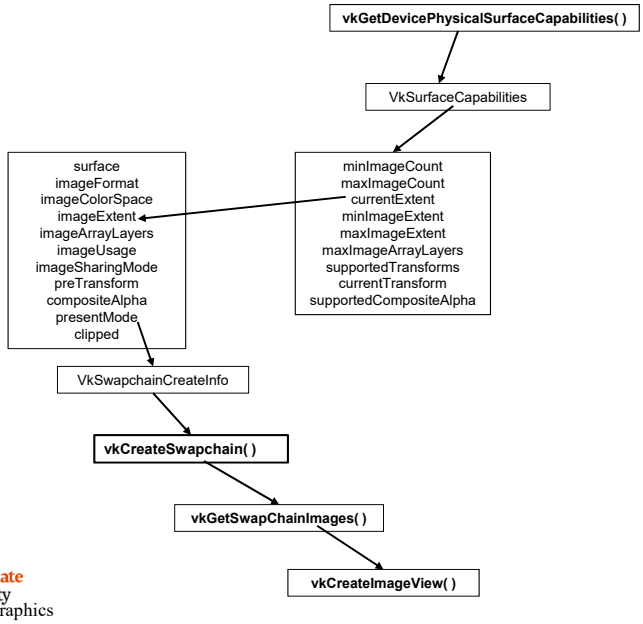
We Need to Find Out What our Display Capabilities Are

VulkanDebug.txt output:

```
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

** 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 )
```

Oregon State University
Computer Graphics

Creating a Swap Chain



Creating a Swap Chain


```
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 );
```

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 ] );
}
```


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, IN &vsci, PALLOCATOR, OUT &imageReadySemaphore );

uint32_t  nextImageIndex;
uint64_t  tmeout = 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 ] );
```

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 );

    . . .

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
```


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 );
```