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 We Think of OpenGL Framebuffers

Video
Driver
Front
Back
Double-buffered
Color Framebuffers

Update
Refresh

Swap Chain
Depth-Buff

What is a Swap Chain?

Swap Chains are tightly coupled with 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 it 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')

What is a Swap Chain?

Because it has the word "chain" in it, let's try to visualize the Swap Chain as a physical bicycle 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.
We Need to Find Out What our Display Capabilities Are

We Need to Find Out What our Display Capabilities Are

Creating the Swap Chain

Creating the Swap Chain

Creating the Swap Chain Images and Image Views

Rendering into the Swap Chain, 1

Creating the Swap Chain Images and Image Views

Rendering into the Swap Chain, 1

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

Creating the Swap Chain

Rendering into the Swap Chain, 1

We Need to Find Out What our Display Capabilities Are

We Need to Find Out What our Display Capabilities Are

Creating a Swap Chain

Creating a Swap Chain

Creating the Swap Chain Images and Image Views

Rendering into the Swap Chain, 1

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