Queues and Command Buffers

Application

Instance

Physical Device

Logical Device

Command Buffer

Vulkan: Overall Block Diagram

Vulkan Queues and Command Buffers

- Graphics commands are recorded in command buffers, e.g., `vkCmdDoSomething(cmdBuffer, ...);
- You can have as many simultaneous Command Buffers as you want
- Each command buffer can be filled from a different thread
- Command Buffers record our commands, but no work takes place until a Command Buffer is submitted to a Queue
- We don't create Queues — the Logical Device has them already
- Each Queue belongs to a Queue Family
- We don't create Queue Families — the Physical Device already has them

Vulkan: a More Typical (and Simplified) Block Diagram

Querying what Queue Families are Available

Similarly, we can write a function that finds the proper Queue Family

```c
int FindQueueFamilyThatDoesGraphics()
{
    uint32_t count = -1;
    vkGetPhysicalDeviceQueueFamilyProperties(IN PhysicalDevice, &count, OUT (VkQueueFamilyProperties *)nullptr);
    VkQueueFamilyProperties *vqfp = new VkQueueFamilyProperties[count];
    vkGetPhysicalDeviceQueueFamilyProperties(IN PhysicalDevice, &count, OUT vqfp);
    for (unsigned int i = 0; i < count; i++)
    {
        if ((vqfp[i].queueFlags & VK_QUEUE_GRAPHICS_BIT) != 0)
            return i;
    }
    return -1;
}
```

"These are not the Queue Families you're looking for."
Creating a Logical Device Queue Needs to Know Queue Family Information

```c
VkDeviceQueueCreateInfo vdqci[1];
{
    .queueCount = 1;
    .queueFamilyIndex = FindQueueFamilyThatDoesGraphics();
    .queuePriorities = (float *) queuePriorities;
    .flags = 0;
    .pNext = nullptr;
    .sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
}
```

Creating the Command Buffers

```c
VkResult result = vkCreateCommandBuffer(LogicalDevice, IN &vcbbi, PALLOCATOR, OUT &CommandBuffers[0]);
```

Beginning a Command Buffer

```c
vkBeginCommandBuffer(commandBuffer, commandBufferCount, const pCommandBuffers);
```

These are the Commands that could be entered into the Command Buffer, I

```c
vkCmdDraw( commandBuffer, vertexCount, instanceCount, firstVertex, firstInstance );
vkCmdDispatch( commandBuffer, groupCountX, groupCountY, groupCountZ );
vkCmdBlitImage( commandBuffer, filter );
```
These are the Commands that could be entered into the Command Buffer, II

vkCmdWriteTimestamp( commandBuffer, pipelineStage, queryPool, query );
vkCmdWaitEvents( commandBuffer, eventCount, pEvents, srcStageMask, dstStageMask, memoryBarrierCount, pMemoryBarriers, bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers );
vkCmdUpdateBuffer( commandBuffer, dstBuffer, dstOffset, dataSize, pData );
vkCmdSetStencilWriteMask( commandBuffer, faceMask, writeMask );
vkCmdSetScissor( commandBuffer, firstScissor, scissorCount, pScissors );
vkCmdSetLineWidth( commandBuffer, lineWidth );
vkCmdSetDiscardRectangleEXT( commandBuffer, firstDiscardRectangle, discardRectangleCount, pDiscardRectangles );
vkCmdSetDeviceMaskKHX( commandBuffer, deviceMask );
vkCmdSetDepthBounds( commandBuffer, minDepthBounds, maxDepthBounds );
vkCmdReserveSpaceForCommandsNVX( commandBuffer, pReserveSpaceInfo );
vkCmdPushDescriptorSetWithTemplateKHR( commandBuffer, descriptorUpdateTemplate, layout, set, pData );
vkCmdPushDescriptorSetKHR( commandBuffer, pipelineBindPoint, layout, set, descriptorWriteCount, pDescriptorWrites );
vkCmdProcessCommandsNVX( commandBuffer, pProcessCommandsInfo );
vkCmdSetDepthBias( commandBuffer, depthBiasConstantFactor, depthBiasClamp, depthBiasSlopeFactor );
vkCmdSetBlendConstants( commandBuffer, blendConstants[4] );
vkCmdResetEvent( commandBuffer, event, stageMask );

Submitting a Command Buffer to a Queue for Execution

VkExtent2D e2d = { Width, Height };
VkOffset2D o2d = { 0, 0 };

VkClearValue vcv[2];
VkClearDepthStencilValue vcdsv;
VkClearColorValue vccv;
vcv[0].color = vccv;
vccv.float32[0] = 0.0;
vccv.float32[1] = 0.0;
vccv.float32[2] = 1.0;
vccv.float32[3] = 1.0;

vrpbi.pClearValues = vcv;               // used for VK_ATTACHMENT_LOAD_OP_CLEAR
vrpbi.clearValueCount = 2;

vkCmdBeginRenderPass
(vrpbi.pNext = nullptr);
vrpbi.framebuffer = Framebuffers[ nextImageIndex ];
vrpbi.renderPass = RenderPass;

vkCmdBindVertexBuffers
(CommandBuffers[nextImageIndex], 1, &buffers[1], offsets[1]);

vkCmdDrawIndexed
(CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance);

Submitting a Command Buffer to a Queue for Execution

The Entire Submission / Wait / Display Process

1/31/2018