Multipass Rendering

Back in Our Single-pass Days

So far, we've only performed single-pass rendering, within a single Vulkan RenderPass.

Here comes a quick reminder of how we did that.

Afterwards, we will extend that.

Multipass Rendering

Back in Our Single-pass Days, I

Multipass Rendering uses Attachments -- What is a Vulkan Attachment Anyway?

"[An attachment is] an image associated with a renderpass that can be used as the input or output of one or more of its subpasses."

— Vulkan Programming Guide

An attachment can be written to, read from, or both.

For example:

```
Subpass  Attachment
Framebuffer
```

Back in Our Single-pass Days, II

So far, we've only performed single-pass rendering, but within a single Vulkan RenderPass, we can also have several subpasses, each of which is feeding information to the next subpass or subpasses.

In this case, we will look at following up a 3D rendering with some image processing on the outcome.

```
result = vkCreateRenderPass(LogicalDevice, &renderPassCreateInfo, nullptr, &renderPass);
```
Multipass Algorithm to Render and then Image Process

No Noise

Noise

Original	Sharpened	Edge Detected

 VkAttachmentReference outputReference;
 VkAttachmentReference depthReference;

VkSubpassDependency vsdp[1]

vkCreateRenderPass

outputReference.attachment = 2;
depthReference.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
depthReference.attachment = 1;
colorReference.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
colorReference.attachment = 0;

vsdp[0].srcSubpass = 0; // 3D rendering

vrpci.pAttachments = vrpci.attachmentCount = 3; // color, depth/stencil, output
vrpci.flags = 0;
vrpci.pNext = nullptr;
vrpci.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;

vrpci.pSubpasses = vrpci.subpassCount = 2;

vrpci.pDependencies = vrpci.dependencyCount = 1;

vsdp[0].dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT;
vsdp[0].dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
vsdp[0].srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
vsdp[0].dstStageMask = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
vsdp[0].srcStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
vsdp[0].dstSubpass = 1; // image processing

Placing a Pipeline Barrier so an Image is not used before it is Ready

PipelineBarrierBarrier

vkCmdPipelineBarrier

vsd[2].flags = 0;
vsd[2].pPreserveAttachments = (uint32_t *) nullptr;
vsd[2].preserveAttachmentCount = 0;
vsd[2].pDepthStencilAttachment = (VkAttachmentReference *) nullptr;
vsd[2].pResolveAttachments = (VkAttachmentReference *) nullptr;
vsd[2].pColorAttachments = &outputReference;
vsd[2].colorAttachmentCount = 1;
vsd[2].pInputAttachments = (VkAttachmentReference *) nullptr;
vsd[2].inputAttachmentCount = 0;
vsd[2].pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;

vsd[1].flags = 0;
vsd[1].pPreserveAttachments = (uint32_t *) nullptr;
vsd[1].preserveAttachmentCount = 0;
vsd[1].pDepthStencilAttachment = &depthReference;
vsd[1].pColorAttachments = colorReference;
vsd[1].colorAttachmentCount = 1;
vsd[1].pInputAttachments = (VkAttachmentReference *) nullptr;
vsd[1].inputAttachmentCount = 0;
vsd[1].pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;

vsd[0].flags = 0;
vsd[0].pPreserveAttachments = (uint32_t *) nullptr;
vsd[0].preserveAttachmentCount = 0;
vsd[0].pDepthStencilAttachment = &depthReference;
vsd[0].pColorAttachments = colorReference;
vsd[0].colorAttachmentCount = 1;
vsd[0].pInputAttachments = (VkAttachmentReference *) nullptr;
vsd[0].inputAttachmentCount = 0;
vsd[0].pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
vkCmdBeginRenderPass(CommandBuffers[nextImageIndex], IN &vrpbi, IN VK_SUBPASS_CONTENTS_INLINE);
// First subpass is automatically started here
vkCmdBindPipeline(CommandBuffers[nextImageIndex], VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipeline);
vkCmdBindDescriptorSets(CommandBuffers[nextImageIndex], VK_PIPELINE_BIND_POINT_GRAPHICS, GraphicsPipelineLayout, 0, 4, DescriptorSets, 0, (uint32_t *) nullptr);
vkCmdBindVertexBuffers(CommandBuffers[nextImageIndex], 0, 1, vBuffers, offsets);
vkCmdDraw(CommandBuffers[nextImageIndex], vertexCount, instanceCount, firstVertex, firstInstance);
...
vkCmdNextSubpass(CommandBuffers[nextImageIndex], VK_SUBPASS_CONTENTS_INLINE);
// Second subpass is started here – doesn’t need any new drawing vkCmd’s...
...
vkCmdEndRenderPass(CommandBuffers[nextImageIndex]);