Vulkan®

Multipass Rendering

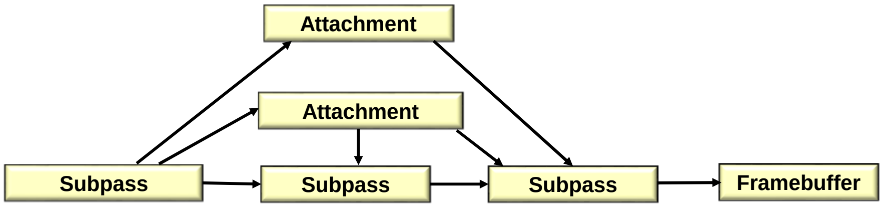
Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

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:



What is an Example of Wanting to do This?

3

There is a process in computer graphics called ***Deferred Rendering***. The idea is that a game-quality fragment shader takes a long time (relatively) to execute, but, with all the 3D scene detail, a lot of the rendered fragments are going to get z-buffered away anyhow. So, why did we invoke the fragment shaders so many times when we didn't need to?

Here's the trick:

Let's create a grossly simple fragment shader that writes out (into multiple framebuffers) each fragment's:

- position (x,y,z)
- normal (nx,ny,nz)
- material color (r,g,b)
- texture coordinates (s,t)

As well as:

- the current light source positions and colors
- the current eye position

When we write these out, the final framebuffers will contain just information for the pixels that *can be seen*. We then make a second pass running the expensive lighting model *just* for those pixels. This known as the ***G-buffer Algorithm***.

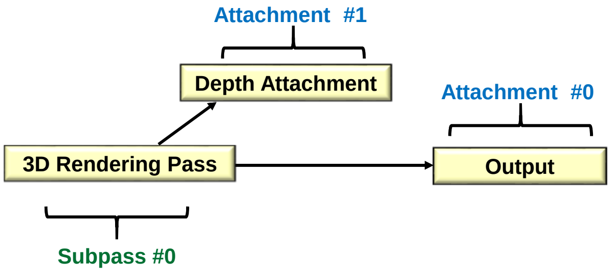
Oregon State University Computer Graphics

mjb – March 3, 2020

Back in Our Single-pass Days

4

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



Attachment #1

Depth Attachment

Attachment #0

3D Rendering Pass

Output

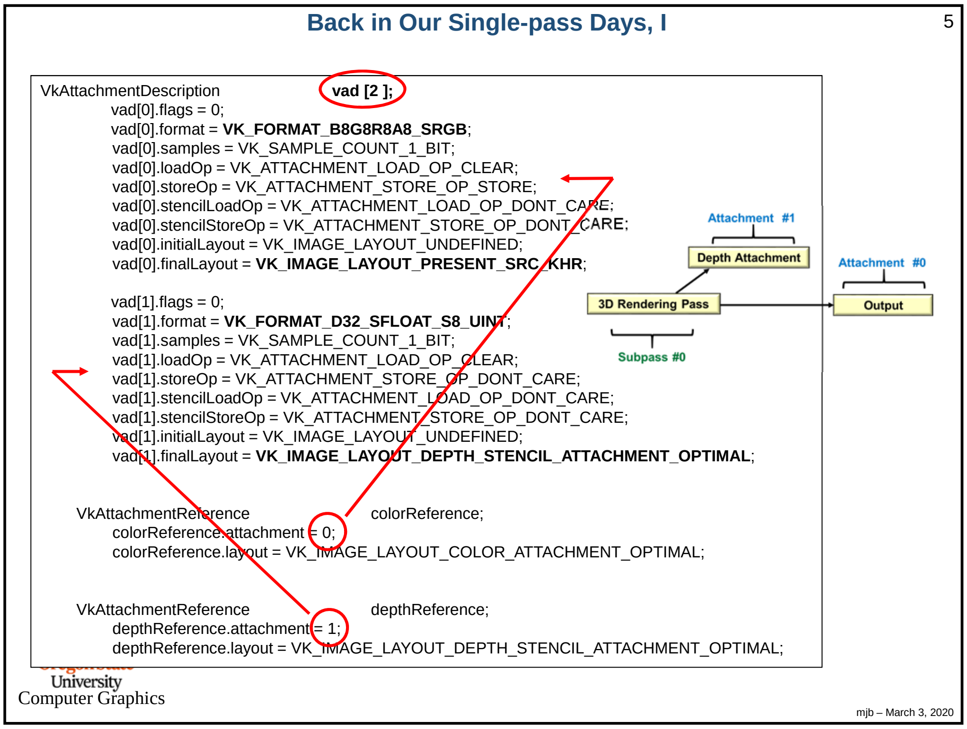
Subpass #0

Here comes a quick reminder of how we did that.

Afterwards, we will extend it.

Oregon State University Computer Graphics

mjb – March 3, 2020



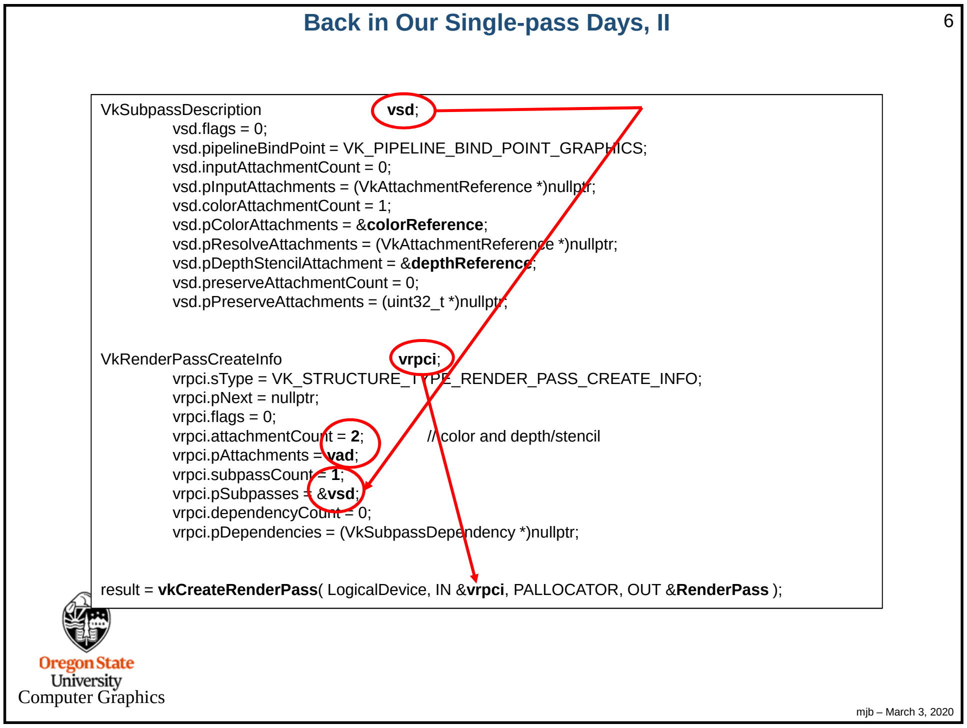
Back in Our Single-pass Days, I

5

```
VkAttachmentDescription                 vad [2 ];
            vad[0].flags = 0;
            vad[0].format = VK_FORMAT_B8G8R8A8_SRGB;
            vad[0].samples = VK_SAMPLE_COUNT_1_BIT;
            vad[0].loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
            vad[0].storeOp = VK_ATTACHMENT_STORE_OP_STORE;
            vad[0].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
            vad[0].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
            vad[0].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
            vad[0].finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;

            vad[1].flags = 0;
            vad[1].format = VK_FORMAT_D32_SFLOAT_S8_UINT;
            vad[1].samples = VK_SAMPLE_COUNT_1_BIT;
            vad[1].loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
            vad[1].storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
            vad[1].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
            vad[1].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
            vad[1].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
            vad[1].finalLayout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;


      VkAttachmentReference                 colorReference;
            colorReference.attachment = 0;
            colorReference.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;


      VkAttachmentReference                 depthReference;
            depthReference.attachment = 1;
            depthReference.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;
```

University
Computer Graphics

mjb – March 3, 2020



Back in Our Single-pass Days, II

6

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


VkRenderPassCreateInfo                 vrpci;
            vrpci.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
            vrpci.pNext = nullptr;
            vrpci.flags = 0;
            vrpci.attachmentCount = 2;            // color and depth/stencil
            vrpci.pAttachments = vad;
            vrpci.subpassCount = 1;
            vrpci.pSubpasses = &vsd;
            vrpci.dependencyCount = 0;
            vrpci.pDependencies = (VkSubpassDependency *)nullptr;


result = vkCreateRenderPass( LogicalDevice, IN &vrpci, PALLOCATOR, OUT &RenderPass );
```

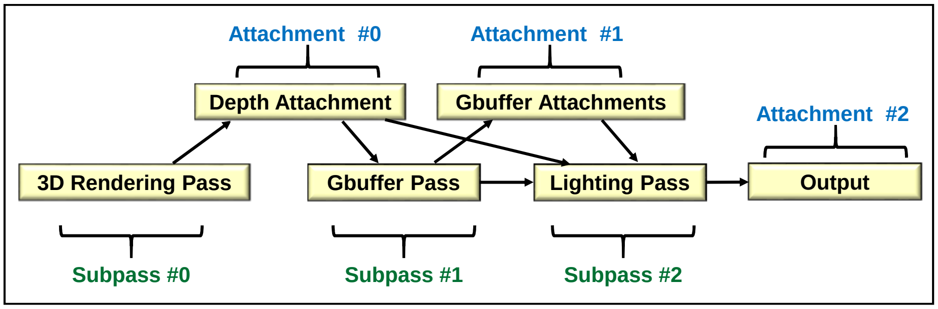
Oregon State
University
Computer Graphics

mjb – March 3, 2020

Multipass Rendering

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 Gbuffer operations.



Attachment #0

Attachment #1

Depth Attachment

Gbuffer Attachments

Attachment #2

3D Rendering Pass

Gbuffer Pass

Lighting Pass

Output

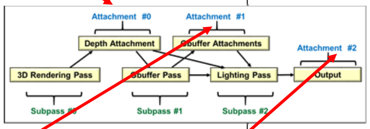
Subpass #0

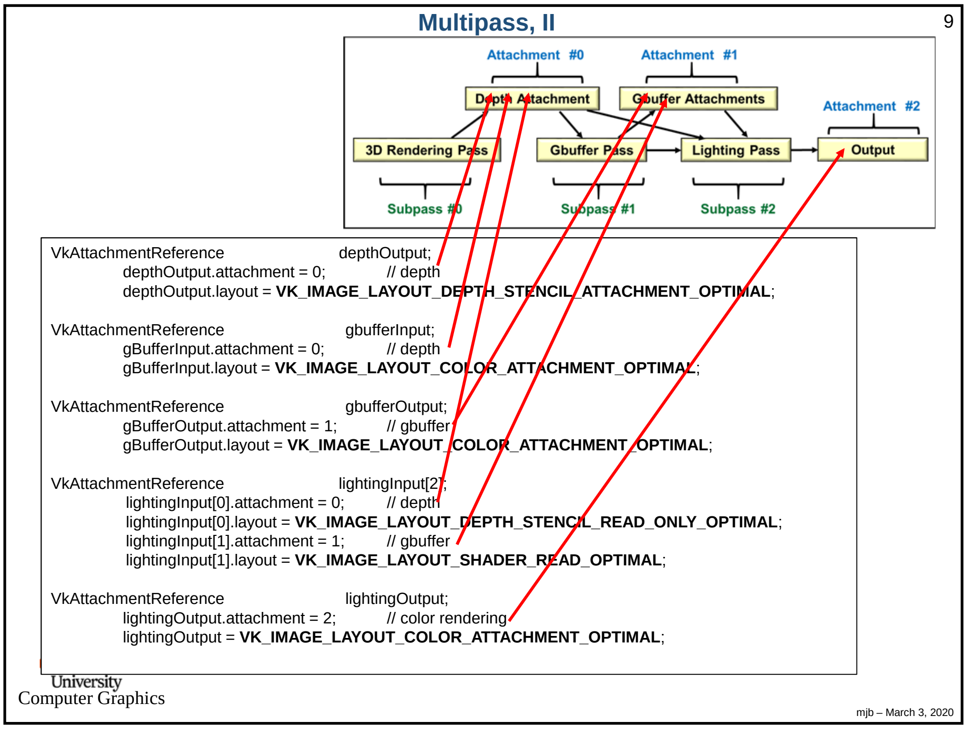
Subpass #1

Subpass #2

Multipass, I

```
VkAttachmentDescription                vad [ 3 ];
            vad[0].flags = 0;
            vad[0].format = VK_FORMAT_D32_SFLOAT_S8_UINT;
            vad[0].samples = VK_SAMPLE_COUNT_1_BIT;
            vad[0].loadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
            vad[0].storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
            vad[0].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
            vad[0].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
            vad[0].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
            vad[0].finalLayout = VK_IMAGE_LAYOUT_UNDEFINED;

            vad[1].flags = 0;
            vad[1].format = VK_FORMAT_R32G32B32A32_UINT;
            vad[1].samples = VK_SAMPLE_COUNT_1_BIT;
            vad[1].loadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
            vad[1].storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
            vad[1].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
            vad[1].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
            vad[1].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
            vad[1].finalLayout = VK_IMAGE_LAYOUT_UNDEFINED;

            vad[2].flags = 0;
            vad[2].format = VK_FORMAT_R8G8B8A8_SRGB;
            vad[2].samples = VK_SAMPLE_COUNT_1_BIT;
            vad[2].loadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
            vad[2].storeOp = VK_ATTACHMENT_STORE_OP_STORE;
            vad[2].stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE;
            vad20].stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE;
            vad[2].initialLayout = VK_IMAGE_LAYOUT_UNDEFINED;
            vad[2].finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC;
```





Multipass, II

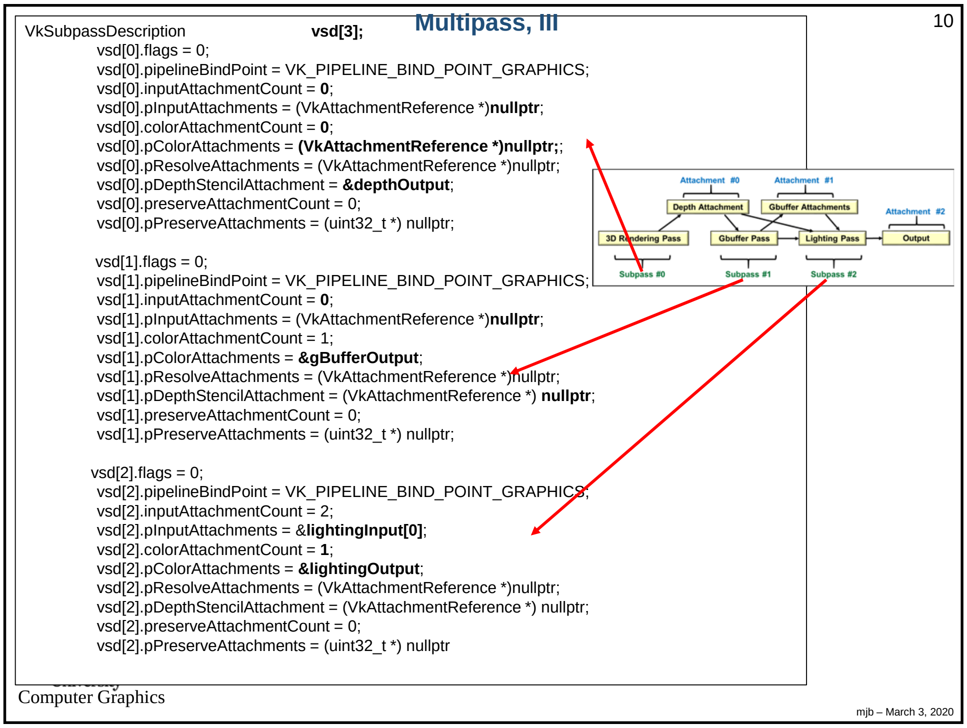
9

Attachment #0
Attachment #1
Attachment #2
Depth Attachment
Gbuffer Attachments
3D Rendering Pass
Gbuffer Pass
Lighting Pass
Output
Subpass #0
Subpass #1
Subpass #2

```
VkAttachmentReference                  depthOutput;
            depthOutput.attachment = 0;          // depth
            depthOutput.layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL;

VkAttachmentReference                   gbufferInput;
            gBufferInput.attachment = 0;          // depth
            gBufferInput.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;

VkAttachmentReference                   gbufferOutput;
            gBufferOutput.attachment = 1;         // gbuffer
            gBufferOutput.layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;

VkAttachmentReference                  lightingInput[2];
             lightingInput[0].attachment = 0;     // depth
             lightingInput[0].layout = VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL;
             lightingInput[1].attachment = 1;     // gbuffer
             lightingInput[1].layout = VK_IMAGE_LAYOUT_SHADER_READ_OPTIMAL;

VkAttachmentReference                   lightingOutput;
            lightingOutput.attachment = 2;        // color rendering
            lightingOutput = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL;
```

University
Computer Graphics

mjb – March 3, 2020



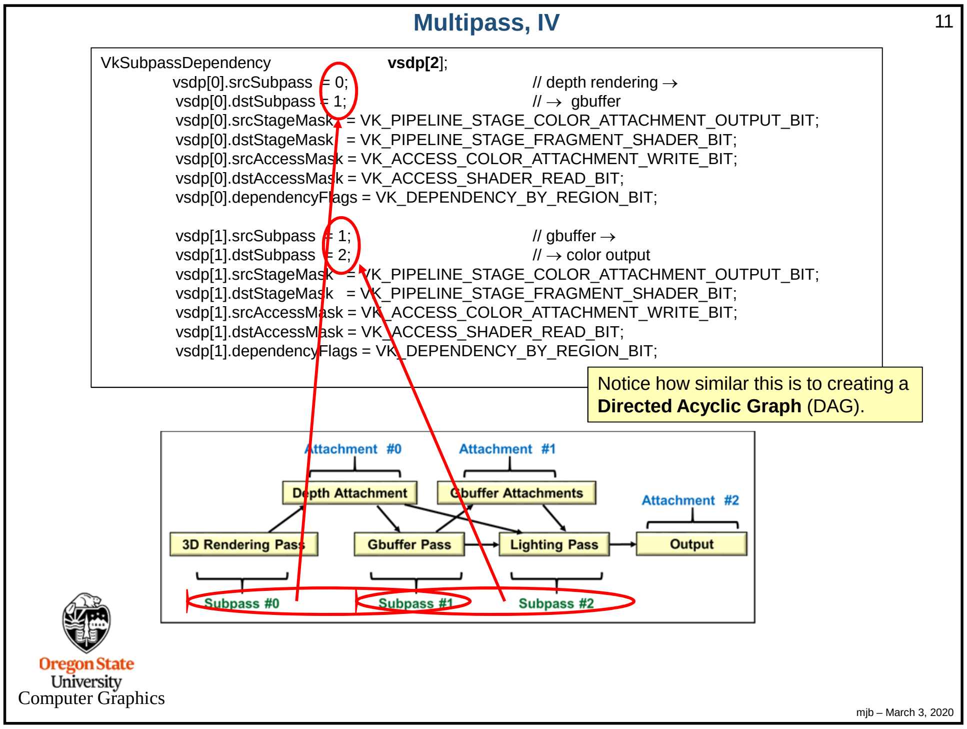
Multipass, III

10

```
VkSubpassDescription                     vsd[3];
            vsd[0].flags = 0;
            vsd[0].pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS;
            vsd[0].inputAttachmentCount = 0;
            vsd[0].pInputAttachments = (VkAttachmentReference *)nullptr;
            vsd[0].colorAttachmentCount = 0;
            vsd[0].pColorAttachments = (VkAttachmentReference *)nullptr;;
            vsd[0].pResolveAttachments = (VkAttachmentReference *)nullptr;
            vsd[0].pDepthStencilAttachment = &depthOutput;
            vsd[0].preserveAttachmentCount = 0;
            vsd[0].pPreserveAttachments = (uint32_t *) nullptr;

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

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

Computer Graphics

mjb – March 3, 2020



Multipass, IV

11

```
VkSubpassDependency                    vsdp[2];
            vsdp[0].srcSubpass = 0;                             // depth rendering →
            vsdp[0].dstSubpass = 1;                             // →  gbuffer
            vsdp[0].srcStageMask  = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
            vsdp[0].dstStageMask   = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
            vsdp[0].srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
            vsdp[0].dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
            vsdp[0].dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT;

            vsdp[1].srcSubpass = 1;                             // gbuffer →
            vsdp[1].dstSubpass = 2;                             // → color output
            vsdp[1].srcStageMask   = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;
            vsdp[1].dstStageMask   = VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT;
            vsdp[1].srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT;
            vsdp[1].dstAccessMask = VK_ACCESS_SHADER_READ_BIT;
            vsdp[1].dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT;
```

Notice how similar this is to creating a **Directed Acyclic Graph** (DAG).

Attachment #0

Attachment #1

Attachment #2

Depth Attachment

Gbuffer Attachments

3D Rendering Pass

Gbuffer Pass

Lighting Pass

Output

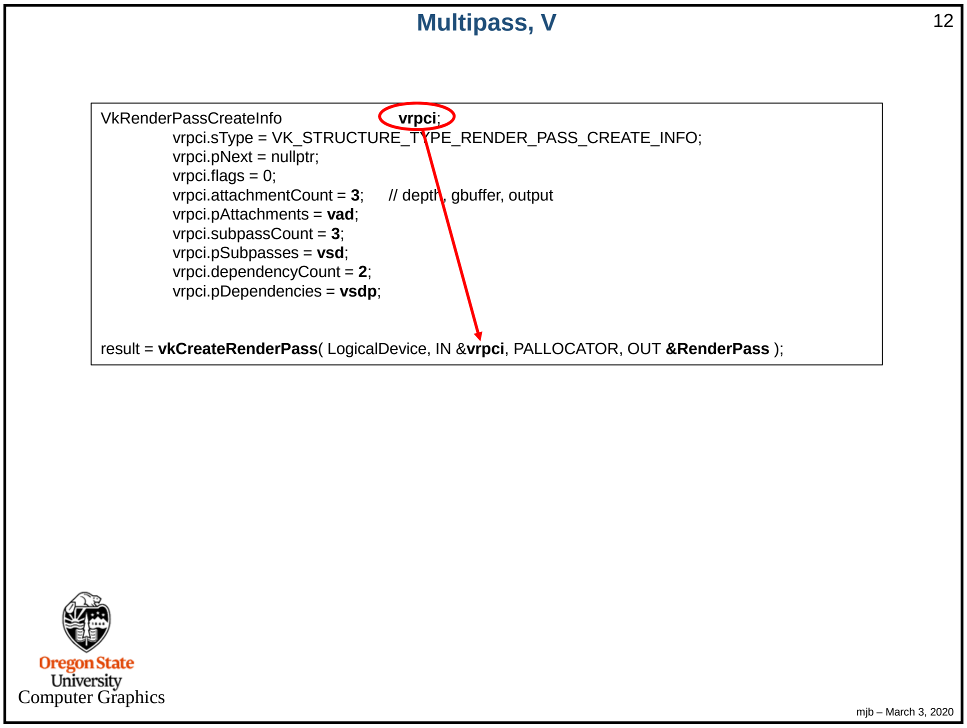
Subpass #0

Subpass #1

Subpass #2

Oregon State
University
Computer Graphics

mjb – March 3, 2020



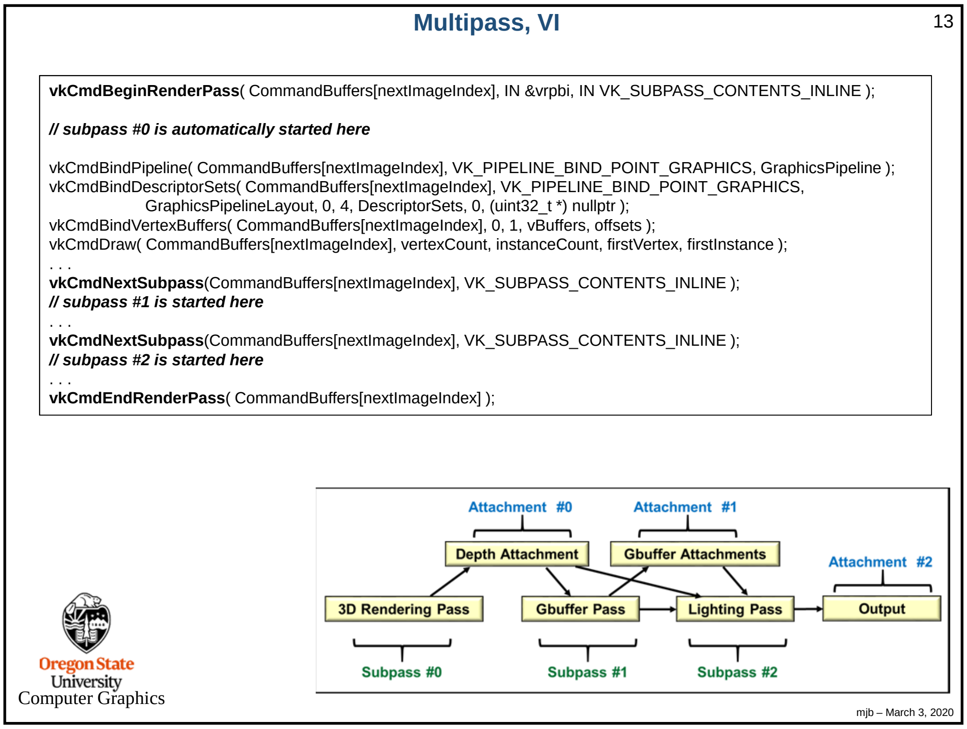
Multipass, V

12

```
VkRenderPassCreateInfo                  vrpci;
            vrpci.sType = VK_STRUCTURE_TYPE_RENDER_PASS_CREATE_INFO;
            vrpci.pNext = nullptr;
            vrpci.flags = 0;
            vrpci.attachmentCount = 3;      // depth, gbuffer, output
            vrpci.pAttachments = vad;
            vrpci.subpassCount = 3;
            vrpci.pSubpasses = vsd;
            vrpci.dependencyCount = 2;
            vrpci.pDependencies = vsdp;


result = vkCreateRenderPass( LogicalDevice, IN &vrpci, PALLOCATOR, OUT &RenderPass );
```

Oregon State
University
Computer Graphics

mjb – March 3, 2020



Multipass, VI

13

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

// subpass #0 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 );
// subpass #1 is started here
. . .
vkCmdNextSubpass(CommandBuffers[nextImageIndex], VK_SUBPASS_CONTENTS_INLINE );
// subpass #2 is started here
. . .
vkCmdEndRenderPass( CommandBuffers[nextImageIndex] );
```

Oregon State University Computer Graphics

mjb – March 3, 2020