Multipass Rendering

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License

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

Attachment

Attachment

Framebuffer

Subpass

Attachment

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.

Back in Our Single-pass Days, I

```
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;
```

Back in Our Single-pass Days, II

```
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 );
```

The following code is written to create a renderpass for rendering a depth map and then loading that into an image that will be used for some image processing.

MultisamplePass

vkCreateSampler

Sampler

ColorAttachment

Attachment #0

DepthAttachment

Attachment #1

Attachment #2

Output

Notice how close this resembles a Directed Acyclic Graph (DAG) data structure: nodes connected by arrows that point in one direction.

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 some image processing on the outcome.
Multispass Algorithm to Render and then Image Process

Original

No Noise

Noise

VkAttachmentReference outputReference;

VkAttachmentReference depthReference;

VkAttachmentReference colorReference;

Multipass Algorithm to Render and then Image Process

result = vkCreateRenderPass(logicalDevice, &vrpci, PALLOCATOR, OUT &RenderPass);

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].srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_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

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