Vulkan Syncronization Notes

[https://github.com/KhronosGroup/Vulkan-Docs/wiki/Synchronization-Examples]

General

A 'vkCmdPipelineBarrier' covers all resources globally.

```
const VkMemoryBarrier mem_barrier = {
    .sType         = VK_STRUCTURE_TYPE_MEMORY_BARRIER,
    .pNext         = NULL,
    .srcAccessMask = VkAccessFlagBits,
    .dstAccessMask = VkAccessFlagBits,
};

const VkImageMemoryBarrier img_barrier = {
    .sType               = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER,
    .pNext               = NULL,
    .srcAccessMask       = VkAccessFlagBits,
    .dstAccessMask       = VkAccessFlagBits,
    .oldLayout           = VkImageLayout,
    .newLayout           = VkImageLayout,
    .srcQueueFamilyIndex = (uint32_t)VK_QUEUE_FAMILY_IGNORED or a queue index,
    .dstQueueFamilyIndex = (uint32_t)VK_QUEUE_FAMILY_IGNORED or a queue index,
    .image               = VkImage,
    .subresourceRange = {
        .aspectMask     = VkImageAspectFlagBits,
        .baseMipLevel   = uint32_t,
        .levelCount     = uint32_t,
        .baseArrayLayer = uint32_t,
        .layerCount     = uint32_t,
    }
};

const VkSubpassDependency sub_pass_dep = {
    .srcSubpass      = uint32_t,
    .dstSubpass      = uint32_t,
    .srcStageMask    = VkPipelineStageFlagBits,
    .dstStageMask    = VkPipelineStageFlagBits,
    .srcAccessMask   = VkAccessFlagBits,
    .dstAccessMask   = VkAccessFlagBits,
    .dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT,
};

'VK_DEPENDENCY_BY_REGION_BIT' means that you only read / write pixels you own in a single sa
```

```
This would not work if you had a blur shader that read outside of that area.

const VkBufferMemoryBarrier buf_mem_barrier = {
    .sType               = VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER,
    .pNext               = NULL,
    .srcAccessMask       = VkAccessFlagBits,
    .dstAccessMask       = VkAccessFlagBits,
    .srcQueueFamilyIndex = uint32_t,
    .dstQueueFamilyIndex = uint32_t,
    .buffer              = VkBuffer,
    .offset              = VkDeviceSize,
    .size                = VkDeviceSize,
};
```

Compute -> Compute

Write -> Read

Using a "vkCmdPipelineBarrier" inbetween two 'vkCmdDispatch' protect against read after write.

'VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT' for both src and dst stage. 'VK_ACCESS_SHADER_WRITE_BIT' for srcAccessMask 'VK_ACCESS_SHADER_READ_BIT' for dstAccessMask

Read -> Write

A simpler execution dependency solves the write after read issue. 'vkCmd-PipelineBarrier' with no memory barrier.

Compute -> Graphics

Write (C) -> Read (G)

Same as Compute -> Compute for this situation.

Write (C) -> Read (G) -> Read (C)

Nearly the same but adds more flags thus batching rather than using two barriers. Or the 'dstStageMask' and 'dstAccessMask' so that it includes both of the following ops.

Image based writes and reads

Use 'VkImageMemoryBarrier' with a correct layout transition. This is only if the Draw wants to view it as an image other wise mem_bariier is fine.

Graphics -> Compute

Use a 'VkImageMemoryBarrier' with the correct stages and access masks. Super simple.

Graphics -> Graphics

Use a 'VkSubpassDependency' if you can. "VkAttachmentDescription" does implicit layout transitions.

Otherwise use a 'VkImageMemoryBarrier'.

WAR Hazard

You would normally only need an execution dep but since you need a layout transition then you need an 'VkImageMemoryBarrier' with an empty 'srcAccessMask'.

WAW Hazard (RP -> RP reusing the same depth buffer)

Always needs a memory dep. Needs use of 'VK_SUBPASS_EXTERNAL' because of the automatic transition. Use pipeline stages with the fragments tests as those use the depth buffer. .dependencyFlags should be 0.

Memory Transfer

Staging buffer -> Device Local Memory

Start with a 'vkCmdCopyBuffer'. Then:

If the queue is unified a normal 'vkCmdPipelineBarrier' is fine.

Otherwise: A 'VkBufferMemoryBarrier' is needed. End and submit to the queue. (Must have a semaphore the gfx queue waints on) Begin commands Another 'VkBufferMemoryBarrier' with the correct dst Access. End cmd and submit to gfx queue.

> Images have the same story except they can use 'VkImageMemoryBarrier' as they also have the family queue indices field. (unified uses: VK_QUEUE_FAMILY_IGNORED)

External Deps

Variables: $first_subpass_that_uses_attachments: 0 $last_subpass_that_uses_attachments : #num_subpasses - 1 $first_need_for_layout: VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPU $last_need_for_layout: VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT;

// When to do layout transitions: dependencies[0].srcSubpass = VK_SUBPASS_EXTERNAL; dependencies[0].dstSubpass = $first_subpass_that_uses_attachments; dependencies[0].srcStageMask = VK_PIPELINE_STAGE_ALL_COMMANDS_BIT;

dependencies[0].dstStageMask = $first_need_for_layout; dependencies[0].srcAccessMask = VK_ACCESS_MEMORY_READ_BIT; dependencies[0].dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT; dependencies[0].dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT;

dependencies[1].srcSubpass = $last_subpass_that_uses_attachments; dependencies[1].dstSubpass = VK_SUBPASS_EXTERNAL; dependencies[1].srcStageMask = $last_need_for_layout; dependencies[1].dstStageMask = VK_PIPELINE_STAGE_ALL_COMMANDS_BIT; dependencies[1].srcAccessMask = VK_ACCESS_COLOR_ATTACHMENT_READ_BIT | VK_ACCESS_COLOR_ATTACHMENT_WRIT
dependencies[1].dstAccessMask = VK_ACCESS_MEMORY_READ_BIT; dependencies[1].dependencyFlags = VK_DEPENDENCY_BY_REGION_BIT;

// ————————————————————-// // Required Layout Table for Access from Shaders // // ————————————————————-// // (1) Storage Image // VK_IMAGE_LAYOUT_GENERAL // // (2) Sampled Image: // VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL // VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_GENERAL // // (3) Combined Image Sampler: // VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL // VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_GENERAL // // (4) Input Attachment: // VK_IMAGE_LAYOUT_DEPTH_READ_ONLY_STENCIL_ATTACHMENT_OPTIMAL // VK_IMAGE_LAYOUT_DEPTH_ATTACHMENT_STENCIL_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL // VK_IMAGE_LAYOUT_GENERAL

// https://www.khronos.org/registry/vulkan/specs/1.1-extensions/html/vkspec.html#synchronization-access-types-supported

Access flag	Supported pipeline stages
VK_ACCESS_INDIRECT_COMMAND_READ_BIT	VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
VK_ACCESS_INDEX_READ_BIT	VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT	VK_PIPELINE_STAGE_VERTEX_INPUT_BIT
VK_ACCESS_UNIFORM_READ_BIT	VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV, VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV,VK_PIPELI VK_PIPELINE_STAGE_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT,VK_PIPE or VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT

Access flag	Supported pipeline stages
VK_ACCESS_SHADER_READ_BIT	VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV, VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV,VK_PIPELI VK_PIPELINE_STAGE_VERTEX_SHADER_BIT,VK_PIPELIN or VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_ACCESS_SHADER_WRITE_BIT	VK_PIPELINE_STAGE_TASK_SHADER_BIT_NV, VK_PIPELINE_STAGE_MESH_SHADER_BIT_NV, VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR, VK_PIPELINE_STAGE_VERTEX_SHADER_BIT, VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT,VK_PIPE or VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_ACCESS_INPUT_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BI
VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT	VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BI
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT, or VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT	VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT, or VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_ACCESS_TRANSFER_READ_BIT	VK_PIPELINE_STAGE_TRANSFER_BIT
VK_ACCESS_TRANSFER_WRITE_BIT	VK_PIPELINE_STAGE_TRANSFER_BIT
VK_ACCESS_HOST_READ_BIT	VK_PIPELINE_STAGE_HOST_BIT
VK_ACCESS_HOST_WRITE_BIT	VK_PIPELINE_STAGE_HOST_BIT
VK_ACCESS_MEMORY_READ_BIT	Any
VK_ACCESS_MEMORY_WRITE_BIT	Any
VK_ACCESS_COLOR_ATTACHMENT_READ_NONCOHERENT_BIT_EXT	VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BI
VK_ACCESS_COMMAND_PREPROCESS_READ_BIT_NV	VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV
VK_ACCESS_COMMAND_PREPROCESS_WRITE_BIT_NV	VK_PIPELINE_STAGE_COMMAND_PREPROCESS_BIT_NV
VK_ACCESS_CONDITIONAL_RENDERING_READ_BIT_EXT	VK_PIPELINE_STAGE_CONDITIONAL_RENDERING_BIT_EX
VK_ACCESS_SHADING_RATE_IMAGE_READ_BIT_NV	VK_PIPELINE_STAGE_SHADING_RATE_IMAGE_BIT_NV
VK_ACCESS_TRANSFORM_FEEDBACK_WRITE_BIT_EXT	VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_WRITE_BIT_EXT	VK_PIPELINE_STAGE_TRANSFORM_FEEDBACK_BIT_EXT
VK_ACCESS_TRANSFORM_FEEDBACK_COUNTER_READ_BIT_EXT	VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT
VK_ACCESS_ACCELERATION_STRUCTURE_READ_BIT_KHR	VK_PIPELINE_STAGE_RAY_TRACING_SHADER_BIT_KHR, or VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUIL
VK_ACCESS_ACCELERATION_STRUCTURE_WRITE_BIT_KHR	VK_PIPELINE_STAGE_ACCELERATION_STRUCTURE_BUIL
VK_ACCESS_FRAGMENT_DENSITY_MAP_READ_BIT_EXT	VK_PIPELINE_STAGE_FRAGMENT_DENSITY_PROCESS_BI

Vulkan Sync In English

There is no synchronization across different Command Buffers when submitted to the same Queue. The queue treats all Command Buffers submitted as one big linear list of commands. This has the implication that any synchronization performed in a single Command Buffer will be seen by all other submitted Command Buffers in the same Queue.

Commands

- Commands should be treated as if they execute out-of-order if no explicit synchronization is performed. This includes across Command Buffers and calls to `vkQueueSubmit`.
- All commands go through a set of *VK_PIPELINE_STAGE*s, these pipeline states are what get synchronized rather than individual commands. All commands go through a `TOP_OF_PIPE` (GPU parsing the command) and `BOTTOM_OF_PIPE` stage (command is done working).

Execution Barriers

Pipeline Barriers

These are created by the `vkCmdPipelineBarrier` function, splitting the command stream into two halves.

References

Yet another blog explaining Vulkan synchronization