Vulkan Synchronization

1. Execution model：

In Vulkan, the execution order of command follows two principles, implicit synchronization guarantees and explicit synchronization mechanisms. ***Explicit synchronization mechanisms*** are used to synchronize the order between device-host (VkEvent, VkFence), queues (VkSemaphore), and commands within a command buffer (VkPipelineBarrier, VkSubpassDependency). ***Implicit Synchronization Guarantees*** is used to tell us the rule about the called orders of all commands within a command buffer that submitted into a queue.

1. Execution and Memory Dependencies：
   1. Pipeline Stage of Commands

In Vulkan, different kinds of command have different **stages**. We list the stages for several types of command:

|  |  |
| --- | --- |
| Type | Stages |
| primitive shading  (Use vertex, tessellation, geometry, fragment shader) | VK\_PIPELINE\_STAGE\_TOP\_OF\_PIPE\_BIT  VK\_PIPELINE\_STAGE\_DRAW\_INDIRECT\_BIT  VK\_PIPELINE\_STAGE\_VERTEX\_INPUT\_BIT  VK\_PIPELINE\_STAGE\_VERTEX\_SHADER\_BIT  VK\_PIPELINE\_STAGE\_TESSELLATION\_CONTROL\_SHADER\_BIT VK\_PIPELINE\_STAGE\_TESSELLATION\_EVALUATION\_SHADER\_BIT  VK\_PIPELINE\_STAGE\_GEOMETRY\_SHADER\_BIT  VK\_PIPELINE\_STAGE\_TRANSFORM\_FEEDBACK\_BIT\_EXT  VK\_PIPELINE\_STAGE\_SHADING\_RATE\_IMAGE\_BIT\_NV  VK\_PIPELINE\_STAGE\_EARLY\_FRAGMENT\_TESTS\_BIT  VK\_PIPELINE\_STAGE\_FRAGMENT\_SHADER\_BIT  VK\_PIPELINE\_STAGE\_LATE\_FRAGMENT\_TESTS\_BIT  VK\_PIPELINE\_STAGE\_COLOR\_ATTACHMENT\_OUTPUT\_BIT  VK\_PIPELINE\_STAGE\_BOTTOM\_OF\_PIPE\_BIT |
| mesh shading  (use task/mesh and fragment shader) | VK\_PIPELINE\_STAGE\_TOP\_OF\_PIPE\_BIT  VK\_PIPELINE\_STAGE\_DRAW\_INDIRECT\_BIT  VK\_PIPELINE\_STAGE\_TASK\_SHADER\_BIT\_NV  VK\_PIPELINE\_STAGE\_MESH\_SHADER\_BIT\_NV  VK\_PIPELINE\_STAGE\_SHADING\_RATE\_IMAGE\_BIT\_NV  VK\_PIPELINE\_STAGE\_EARLY\_FRAGMENT\_TESTS\_BIT  VK\_PIPELINE\_STAGE\_FRAGMENT\_SHADER\_BIT  VK\_PIPELINE\_STAGE\_LATE\_FRAGMENT\_TESTS\_BIT  VK\_PIPELINE\_STAGE\_COLOR\_ATTACHMENT\_OUTPUT\_BIT  VK\_PIPELINE\_STAGE\_BOTTOM\_OF\_PIPE\_BIT |
| Compute  (computer shader) | VK\_PIPELINE\_STAGE\_TOP\_OF\_PIPE\_BIT  VK\_PIPELINE\_STAGE\_DRAW\_INDIRECT\_BIT  VK\_PIPELINE\_STAGE\_COMPUTE\_SHADER\_BIT  VK\_PIPELINE\_STAGE\_BOTTOM\_OF\_PIPE\_BIT |
| Transfer  (buffer copy) | VK\_PIPELINE\_STAGE\_TOP\_OF\_PIPE\_BIT  VK\_PIPELINE\_STAGE\_TRANSFER\_BIT  VK\_PIPELINE\_STAGE\_BOTTOM\_OF\_PIPE\_BIT |
| command processing  (???) | VK\_PIPELINE\_STAGE\_TOP\_OF\_PIPE\_BIT  VK\_PIPELINE\_STAGE\_COMMAND\_PROCESS\_BIT\_NVX  VK\_PIPELINE\_STAGE\_BOTTOM\_OF\_PIPE\_BIT |

Those stages will be used for specifying synchronization section (pipeline barrier and sub-pass dependency). We will specify

* 1. Memory Access Type
  2. Execution Dependency
  3. Memory Dependency

1. Synchronization Type：

Implicit Synchronization Guarantees：

* ***Submission order*：**

Explicit Synchronization Mechanisms：

* ***Fence*：**Use while host wait device. (CPU wait GPU)
* ***Semaphore*：**Use while some device tasks (queue submit, etc…) need to wait other device task. (GPU wait GPU)
* ***Event*：**Use this mechanism while we want to let device wait host. (GPU wait CPU)
* ***Pipeline Barrier*：**Provide synchronization while device execute commands within a command buffer. It’s one of kind for *synchronization commands*. (synchronization commands in the same command buffer.)
* ***Render Pass*：**Provide synchronization when we execute command in a render pass.  
  (Subpasses are executed asynchronically.)
  1. Fence
  2. Semaphore
  3. Event
  4. Pipeline Barrier
  5. Render Pass

1. Reference

[1] Yet another blog explaining Vulkan synchronization   
<http://themaister.net/blog/2019/08/14/yet-another-blog-explaining-vulkan-synchronization/>

[2] Vulkan's execution model and synchronization   
<https://stackoverflow.com/questions/56838455/vulkans-execution-model-and-sycnhronization>

[3] Vulkan barriers explained  
<https://gpuopen.com/vulkan-barriers-explained/>

[4] Khronos spec   
<https://www.khronos.org/registry/vulkan/specs/1.1-extensions/pdf/vkspec.pdf>

[5] BARRIERS IN VULKAN : THEY ARE NOT THAT DIFFICULT  
<http://cpp-rendering.io/barriers-vulkan-not-difficult/>