Vulkan Synchronization

1. Note：

* Synchronization of accessing resources is the responsibility of application in Vulkan.
* There are two type synchronization. There are distinct implicit and explicit synchronization. Implicit synchronization guarantee is basic rule about the order that commands in a command buffer are executed. Explicit synchronization mechanisms are used to let developer design their synchronization for themselves.
* Synchronization scope can be limited to just operations executing in specific pipeline stages. (pipeline barrier)

1. Execution model：

We can divide commands to four parts as follow:

|  |  |
| --- | --- |
| Command types |  |
| Drawing and dispatching | draw, dispatch, clear, copy, query/timestamp operations, begin/end subpass operations. |
| Set states | bind pipelines/descriptor sets/and buffers  set dynamic state/push constants/set renderpass/subpass state. |
| Perform synchronizations | set/wait events, pipeline barrier, render pass/subpass dependencies. |
| Actions | alter framebuffer attachments, read/write buffer or image memory, or write to query pools. |

1. Synchronization Type：

Implicit Synchronization guarantees：

* *Submit order*：

Explicit Synchronization mechanisms：

* *Fence*：Use while host wait device. (CPU wait GPU)
* *Semaphores*：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 Barriers*：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 Passes*：Provide synchronization when we execute command in a render pass. (Subpasses are executed asynchronically.)

1. Execution and Memory Dependencies