Vulkan Synchronization Validation Quick Start Guide

John Zulauf, LunarG
Available as Alpha Release with SDK 1.2.148.1
August 2020

Introduction

Synchronization Validation is intended to identify resource access conflicts due to missing or incorrect synchronization operations between actions (Draw, Copy, Dispatch, Blit) reading or writing the same regions of memory.

The following quick start should enable initial testing for those familiar with Vulkan synchronization and debugging validation issues. Prior to enabling Synchronization Validation, assure that the default set of Validation checks run cleanly. It is assumed that quick start readers are familiar with both Vulkan Synchronization and using/configuring Vulkan Validation.

Running Synchronization Validation

The simplest way to run synchronization validation and debug issues is to:

- Enable Synchronization Validation using Vulkan Configurator (vkconfig).
- Create a debug callback with `vkCreateDebugUtilsMessengerEXT` with `VK_DEBUG_REPORT_ERROR_BIT_EXT` set.
- Set a breakpoint in the debug callback and run your application in the debugger.
- The hazards will be reported when a `vkCmd`... command with a hazard is recorded.

Synchronization Validation Messages

All synchronization error messages begin with `SYNC-<hazard name>`. The message body is constructed:

```
<cmd name>: Hazard <hazard name> <command specific details> Access info (<...>)
```

Command specific details typically include the specifics of the access within the current command. The `Access info` is common to all Synchronization Validation error messages.

Field	Description
usage	The stage/access of the current command
prior_usage	The stage/access of the previous (hazarded) use
read_barrier	For read usage, the list of stages with execution barriers between prior_usage and usage
write_barrier	For write usage, the list of stage/access (in usage format) with memory barriers between prior_usage and usage
command	The command that performed prior_usage
seq_no	The zero based index of command within the command buffer
reset_no	the reset count of the command buffer command is recorded to

Frequently Found Issues

- Assuming Pipeline stages are logically extended with respect to memory access barriers. Specifying the vertex shader stage in a barrier will not apply to all subsequent shader stages read/write access.
- Invalid stage/access pairs (specifying a pipeline stage for which a given access is not valid) that yield no barrier.
- Relying on implicit subpass dependencies with VK_SUBPASS_EXTERNAL when memory barriers are needed.
- Missing memory dependencies with Image Layout Transitions from pipeline barrier or renderpass Begin/Next/End operations.
- Missing stage/access scopes for load operations, noting that color and depth/stencil are done by different stage/access.

Debugging Tips

- Read and write barriers in the error message can help identify the synchronization operation (either subpass dependency or pipeline barrier) with insufficient or incorrect destination stage/access masks (second scope).
- `Access info read_barrier` and `write_barrier` values of 0, reflect the absence of any barrier and can indicate an insufficient or incorrect source mask (first scope)
- Insert additional barriers with stage/access `VK_PIPELINE_STAGE_ALL_COMMANDS_BIT`, `VK_ACCESS_MEMORY_READ_BIT|VK_ACCESS_MEMORY_WRITE_BIT` for both `src*Mask` and `dst*Mask` fields to locate missing barriers. If the inserted barrier *resolves* a hazard, the conflicting access *happens-before* the inserted barrier. (Be sure to delete later.)

Synchronization blogs/articles

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

Keeping your GPU fed without getting bitten
https://www.youtube.com/watch?v=oF7vOTTaAh4

Yet another blog explaining Vulkan synchronization
http://themaister.net/blog/2019/08/14/yet-another-blog-explaining-vulkan-synchronization/