Pipeline Barriers

Oregon State University
Mike Bailey
mjb@cs.oregonstate.edu

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

From the Command Buffer Notes:
These are the Commands that could be entered into the Command Buffer, I

- `vkCmdBeginQuery(commandBuffer, flags);`
- `vkCmdBeginRenderPass(commandBuffer, const contents);`
- `vkCmdBindDescriptorSets(commandBuffer, pDynamicOffsets);`
- `vkCmdBindIndexBuffer(commandBuffer, indexType);`
- `vkCmdBindPipeline(commandBuffer, pipeline);`
- `vkCmdBindVertexBuffers(commandBuffer, firstBinding, bindingCount, const pOffsets);`
- `vkCmdBlitImage(commandBuffer, filter);`
- `vkCmdClearAttachments(commandBuffer, attachmentCount, const pRects);`
- `vkCmdClearColorImage(commandBuffer, pRanges);`
- `vkCmdClearDepthStencilImage(commandBuffer, pRanges);`
- `vkCmdCopyBuffer(commandBuffer, pRegions);`
- `vkCmdCopyBufferToImage(commandBuffer, pRanges);`
- `vkCmdCopyImage(commandBuffer, pRegions);`
- `vkCmdCopyImageToBuffer(commandBuffer, pRegions);`
- `vkCmdDebugMarkerBeginEXT(commandBuffer, pMarkerInfo);`
- `vkCmdDebugMarkerEndEXT(commandBuffer);`
- `vkCmdDebugMarkerInsertEXT(commandBuffer, pMarkerInfo);`
- `vkCmdDispatch(commandBuffer, groupCountX, groupCountY, groupCountZ);`
- `vkCmdDispatchIndirect(commandBuffer, offset);`
- `vkCmdDraw(commandBuffer, vertexCount, instanceCount, firstVertex, firstInstance);`
- `vkCmdDrawIndexed(commandBuffer, indexCount, instanceCount, firstIndex, int32_t vertexOffset, firstInstance);`
- `vkCmdDrawIndexedIndirect(commandBuffer, stride);`
- `vkCmdDrawIndexedIndirectCountAMD(commandBuffer, stride);`
- `vkCmdDrawIndirect(commandBuffer, stride);`
- `vkCmdDrawIndirectCountAMD(commandBuffer, stride);`
- `vkCmdEndQuery(commandBuffer, query);`
- `vkCmdEndRenderPass(commandBuffer);`
- `vkCmdExecuteCommands(commandBuffer, commandBufferCount, const pCommandBuffers);`
From the Command Buffer Notes:

These are the Commands that could be entered into the Command Buffer, II

vkCmdFillBuffer(commandBuffer, dstBuffer, dstOffset, size, data);
vkCmdNextSubpass(commandBuffer, contents);
vkCmdPipelineBarrier(commandBuffer, srcStageMask, dstStageMask, memoryBarrierCount, pMemoryBarriers, bufferMemoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers);
vkCmdProcessedCommandNVX(commandBuffer, pProcessedCommandsInfo);
vkCmdPushConstant(commandBuffer, layout, stageFlags, dependencyFlags, memoryBarrierCount, pBufferMemoryBarriers, imageMemoryBarrierCount, pImageMemoryBarriers);
vkCmdPushDescriptorSetWithTemplateKHR(commandBuffer, descriptorUpdateTemplateInfo, layout, set, pValues);
vkCmdResolveImage(commandBuffer, srcImage, srcImageLayout, dstImage, dstImageLayout, regionCount, pRegions);
vkCmdSetBlendConstants(commandBuffer, blendConstants[4]);
vkCmdSetDepthBounds(commandBuffer, minDepthBounds, maxDepthBounds);
vkCmdSetDeviceMask_KHR(commandBuffer, deviceMask);
vkCmdSetDiscardRectangleEXT(commandBuffer, firstDiscardRectangle, discardRectangleCount, pDiscardRectangles);
vkCmdSetEvent(commandBuffer, event, stageMask);
vkCmdSetStencilCompareMask(commandBuffer, faceMask, compareMask);
vkCmdSetStencilReference(commandBuffer, faceMask, reference);
vkCmdSetStencilWriteMask(commandBuffer, faceMask, writeMask);
vkCmdSetViewport(commandBuffer, firstViewport, viewportCount, pViewports);
vkCmdSetViewportWScalingNV(commandBuffer, firstViewport, viewportCount, pViewportWScalings);
vkCmdSetViewportRectanglesNV(commandBuffer, rectCount, pRectangles);
vkCmdSetLineWidth(commandBuffer, lineWidth);
vkCmdSetScissor(commandBuffer, firstScissor, scissorCount, pScissors);
vkCmdSetStencilMask(commandBuffer, stencilMask);
vkCmdSetStencilRelativeConstant(commandBuffer, stencilRelativeConstant);
vkCmdSetStencilTestEnable(commandBuffer, stencilTestEnable);
vkCmdUpdateImageLayout(commandBuffer, image, layout);
vkCmdUpdateImageLayoutNVX(commandBuffer, image, layout);
vkCmdViewport(commandBuffer, x, y, width, height);
vkCmdViewportsNVX(commandBuffer, viewportCount, pViewports);
vkCmdWriteTimestamp(commandBuffer, pipelineStage, queryPool, query);

We want all of these commands to be able to run “flat-out”, but, if we do that, surely there will be race conditions!

The Scenario

1. All traffic lights start out green (“we want all of these commands to be able to run flat-out!”)
2. There are special sensors at all intersections that get triggered when the first car in a group enters that intersection
3. There are special connections from those sensors to the traffic lights so that when a sensor gets triggered, an arbitrary traffic light can be turned red
4. When an entire group of cars completely makes it through an intersection, that sensor can use that same connection to turn the light back to green
5. The src cars get released first, immediately followed by the trigger/connection rules, immediately followed by the dst cars

We want all of these commands to be able to run “flat-out”, but, if we do that, surely there will be race conditions!

The Scenario

1. All traffic lights start out green (“we want all of these commands to be able to run flat-out!”)
2. There are special sensors at all intersections that get triggered when the first car in a group enters that intersection
3. There are special connections from those sensors to the traffic lights so that when a sensor gets triggered, an arbitrary traffic light can be turned red
4. When an entire group of cars completely makes it through an intersection, that sensor can use that same connection to turn the light back to green
5. The src cars get released first, immediately followed by the trigger/connection rules, immediately followed by the dst cars
**vkCmdPipelineBarrier() Function Call**

A **Pipeline Barrier** is a way to establish a memory dependency between commands that were submitted before the barrier and commands that are submitted after the barrier.

```c
vkCmdPipelineBarrier( commandBuffer,
    srcStageMask,   // Guarantee that this pipeline stage has completely finished with one
    dstStageMask,   // ... allowing this pipeline stage to proceed with the next set of data
    VK_DEPENDENCY_BY_REGION_BIT,
    memoryBarrierCount,  // Defines what data we will be blocking/un-blocking on
    pMemoryBarriers,
    bufferMemoryBarrierCount,  // Defines what data we will be blocking/un-blocking on
    pBufferMemoryBarriers,
    imageMemoryBarrierCount,  // Defines what data we will be blocking/un-blocking on
    pImageMemoryBarriers
);
```

**Pipeline Stage Flags – Where in the Pipeline is this Memory being Accessed?**

- 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_FRAGMENT_SHADER_BIT
- VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
- VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
- VK_PIPELINE_STAGE_TRANSFER_BIT
- VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
- VK_PIPELINE_STAGE_HOST_BIT
- VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
- VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

*Note: the concept of an in-order pipeline is accurate, but really the src and dst triggering action only depends on the name of the street where you are right now.*
Pipeline Stages

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_FRAGMENT_SHADER_BIT
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_PIPELINE_STAGE_TRANSFER_BIT
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
VK_PIPELINE_STAGE_HOST_BIT
VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

Memory Barrier

VkMemoryBarrier vmb;
    vmb.sType = VK_STRUCTURE_TYPE_MEMORY_BARRIER;
    vmb.pNext = nullptr;
    vmb.srcAccessMask = ??;
    vmb.dstAccessMask = ??
Access Flags –
What Operation are you Interested in this Memory for?

- VK_ACCESS_INDIRECT_COMMAND_READ_BIT
- VK_ACCESS_INDEX_READ_BIT
- VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
- VK_ACCESS_UNIFORM_READ_BIT
- VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
- VK_ACCESS_SHADER_READ_BIT
- VK_ACCESS_SHADER_WRITE_BIT
- VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
- VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
- VK_ACCESS_TRANSFER_READ_BIT
- VK_ACCESS_TRANSFER_WRITE_BIT
- VK_ACCESS_HOST_READ_BIT
- VK_ACCESS_HOST_WRITE_BIT
- VK_ACCESS_MEMORY_READ_BIT
- VK_ACCESS_MEMORY_WRITE_BIT

Pipeline Stage Flags and Access Flags Together

- 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_FRAGMENT_SHADER_BIT
- VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
- VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
- VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
- VK_PIPELINE_STAGE_TRANSFER_BIT
- VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
- VK_PIPELINE_STAGE_HOST_BIT
- VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
- VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

- VK_ACCESS INDIRECT COMMAND_READ_BIT
- VK_ACCESS_INDEX_READ_BIT
- VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
- VK_ACCESS_UNIFORM_READ_BIT
- VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
- VK_ACCESS_SHADER_READ_BIT
- VK_ACCESS_SHADER_WRITE_BIT
- VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
- VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
- VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
- VK_ACCESS_TRANSFER_READ_BIT
- VK_ACCESS_TRANSFER_WRITE_BIT
- VK_ACCESS_HOST_READ_BIT
- VK_ACCESS_HOST_WRITE_BIT
- VK_ACCESS_MEMORY_READ_BIT
- VK_ACCESS_MEMORY_WRITE_BIT
Example: Don’t overwrite a texture image until we are done using it

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_FRAGMENT_SHADER_BIT
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_PIPELINE_STAGE_TRANSFER_BIT
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
VK_PIPELINE_STAGE_HOST_BIT
VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
VK_PIPELINE_STAGE_ALL_COMMANDS_BIT

VK_ACCESS_INDIRECT_COMMAND_READ_BIT
VK_ACCESS_INDEX_READ_BIT
VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
VK_ACCESS_UNIFORM_READ_BIT
VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
VK_ACCESS_SHADER_READ_BIT
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
VK_ACCESS_TRANSFER_READ_BIT
VK_ACCESS_HOST_READ_BIT
VK_ACCESS_MEMORY_READ_BIT

VK_ACCESS_SHADER_WRITE_BIT
VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
VK_ACCESS_TRANSFER_WRITE_BIT
VK_ACCESS_HOST_WRITE_BIT
VK_ACCESS_MEMORY_WRITE_BIT

src cars
vertex shader
early fragment tests
late fragment tests
geometry shader
fragment shader
color attachment output
compute shader
transfer
bottom of pipe
host
all graphics
all commands

dst cars
vertex shader
early fragment tests
late fragment tests
game
fragment shader
color attachment output
compute shader
transfer
d bottom of pipe
d host
d all graphics
d all commands

The Scenario
Example: Don’t read a buffer back to the host until a shader is done writing

```
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_TESSSELLATION_CONTROL_SHADER_BIT
VK_PIPELINE_STAGE_TESSSELLATION_EVALUATION_SHADER_BIT
VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT
VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT
VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT
VK_PIPELINE_STAGE_TRANSFER_BIT
VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT
VK_PIPELINE_STAGE_HOST_BIT
VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT
VK_PIPELINE_STAGE_ALL_COMMANDS_BIT
```

```
src
```

```
dst
```

```
VK_ACCESS_INDIRECT_COMMAND_READ_BIT
VK_ACCESS_INDEX_READ_BIT
VK_ACCESS_VERTEX_ATTRIBUTE_READ_BIT
VK_ACCESS_UNIFORM_READ_BIT
VK_ACCESS_INPUT_ATTACHMENT_READ_BIT
VK_ACCESS_SHADER_READ_BIT
VK_ACCESS_SHADER_WRITE_BIT
VK_ACCESS_COLOR_ATTACHMENT_READ_BIT
VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_READ_BIT
VK_ACCESS_DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
VK_ACCESS_TRANSFER_READ_BIT
VK_ACCESS_TRANSFER_WRITE_BIT
VK_ACCESS_HOST_READ_BIT
VK_ACCESS_HOST_WRITE_BIT
VK_ACCESS_MEMORY_READ_BIT
VK_ACCESS_MEMORY_WRITE_BIT
```

```
src
```

```
dst
```

The Scenario
Buffer Memory Barrier

```cpp
VkBufferMemoryBarrier vbmb;
vbmb.sType = VK_STRUCTURE_TYPE_BUFFER_MEMORY_BARRIER;
vbmb.pNext = nullptr;
vbmb.srcAccessMask = ??;
vbmb.dstAccessMask = ??;
vbmb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vbmb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vbmb.buffer = ??;
vbmb.offset = 0;
vbmb.size = VK_WHOLE_SIZE;
```

Image Memory Barrier

```cpp
VkImageSubresourceRange visr;
visr.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
visr.baseMipLevel = 0;
visr.levelCount = 1;
visr.baseArrayLayer = 0;
visr.layerCount = 1;
```

```cpp
VkImageMemoryBarrier vimb;
vimb.sType = VK_STRUCTURE_TYPE_IMAGE_MEMORY_BARRIER;
vimb.pNext = nullptr;
vimb.srcAccessMask = ??;
vimb.dstAccessMask = ??;
vimb.oldLayout = ??;
vimb.newLayout = ??;
vimb.srcQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vimb.dstQueueFamilyIndex = VK_QUEUE_FAMILY_IGNORED;
vimb.image = ??;
vimb.subresourceRange = visr;
```
**VkImageLayout – How an Image gets Laid Out in Memory depends on How it will be Used**

<table>
<thead>
<tr>
<th>VkImageLayout</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_IMAGE_LAYOUT_UNDEFINED</td>
<td>Used as a color attachment</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_GENERAL</td>
<td></td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL</td>
<td>Used as a color attachment</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL</td>
<td>Used as a color attachment</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL</td>
<td>Used as a color attachment</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL</td>
<td>Read into a shader as a texture</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL</td>
<td>Copy from</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUTTRANSFER_DST_OPTIMAL</td>
<td>Copy to</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_PRESENT_SRC_KHR</td>
<td>Show image to viewer</td>
</tr>
<tr>
<td>VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR</td>
<td></td>
</tr>
</tbody>
</table>

One use of `vkCmdPipelineBarrier()` is to simply change the layout of an image.