Pipeline Barriers:
A case of Gate-ing and Wait-ing

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 can be entered into the Command Buffer, I

```cpp
// Vulkan commands
vkCmdBeginQuery(commandBuffer, flags);
vkCmdBeginRenderPass(commandBuffer, const contents);
vkCmdBindDescriptorSets(commandBuffer, pDynamicOffsets);
vkCmdBindIndirectBuffer(commandBuffer, indexType);
vkCmdBindPipeline(commandBuffer, pipeline);
vkCmdBindVertexBuffers(commandBuffer, firstBinding, bindingCount, const pOffsets);
vkCmdBlitImage(commandBuffer, image, image, filter);
vkCmdClearAttachments(commandBuffer, attachmentCount, const pRects);
vkCmdClearColorImage(commandBuffer, image, range);
vkCmdClearDepthStencilImage(commandBuffer, range);
vkCmdCopyBuffer(commandBuffer, pRegions);
vkCmdCopyBufferToImage(commandBuffer, pRegions);
vkCmdCopyImageToBuffer(commandBuffer, pRegions);
vkCmdCopyQueryPoolResults(commandBuffer, flags);
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);
vkCmdExecuteCommands(commandBuffer, commandBufferCount, const pCommandBuffers);
```
From the Command Buffer Notes:
These are the Commands that can be entered into the Command Buffer, II

- vkCmdFillBuffer
- vkCmdNextSubpass
- vkCmdPipelineBarrier
- vkCmdProcessCommandNV
- vkCmdPushConstants
- vkCmdPushDescriptorSetKHR
- vkCmdPushDescriptorSetWithTemplateKHR
- vkCmdReserveSpaceForCommandsNV
- vkCmdResetEvent
- vkCmdResetQueryPool
- vkCmdResolveImage
- vkCmdSetBlendConstants
- vkCmdSetDepthBias
- vkCmdSetDepthBounds
- vkCmdSetDeviceMaskKHX
- vkCmdSetDiscardRectangleEXT
- vkCmdSetEvent
- vkCmdSetLineWidth
- vkCmdSetScissor
- vkCmdSetStencilCompareMask
- vkCmdSetStencilReference
- vkCmdSetStencilWriteMask
- vkCmdSetViewport
- vkCmdSetViewportWScalingNV
- vkCmdUpdateBuffer
- vkCmdWaitEvents
- vkCmdWriteTimestamp
- vkCmdWriteTimestampNV
- vkCmdWriteTimestampNV

We don’t any one of these commands to have to wait on a previous command unless you say so. In general, we want all of these commands to be able to run “flat-out”.

But, if we do that, surely there will be nasty race conditions!

Potential Memory Race Conditions that Pipeline Barriers can Prevent

1. Write-then-Read (WtR) – the memory write in one operation starts overwriting the memory that another operation’s read needs to use
2. Read-then-Write (RtW) – the memory read in one operation hasn’t yet finished before another operation starts overwriting that memory
3. Write-then-Write (WtW) – two operations start overwriting the same memory and the end result is non-deterministic

Note: there is no problem with Read-then-Read (RtR) as no data has been changed
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.

vkCmdPipelineBarrier( commandBuffer,
    srcStageMask, dstStageMask,
    VK_DEPENDENCY_BY_REGION_BIT,
    memoryBarrierCount, pMemoryBarriers,
    bufferMemoryBarrierCount, pBufferMemoryBarriers,
    imageMemoryBarrierCount, pImageMemoryBarriers
    );

The Scenario

1. The cross-streets are named after pipeline stages
2. All traffic lights start out green ("we want all of these commands to be able to run flat-out")
3. There are special sensors at all intersections that will know when the first car in the src group enters that intersection
4. There are connections from those sensors to the traffic lights so that when the first car in the src group enters its intersection, the dst traffic light will be turned red
5. When the last car in the src group completely makes it through its intersection, the dst traffic light can be turned back to green
6. The Vulkan command pipeline ordering is this: (1) the src cars get released, (2) the pipeline barrier is invoked (which turns some lights red), (3) the dst cars get released (which end up being stopped by a red light somewhere)
Pipeline Stage Masks –
Where in the Pipeline is this Memory Data being Generated or Consumed?

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

Pipeline Stages

Vertex Shader
Primitive Assembly
Tessellation Control Shader
Tessellation Primitive Generator
Tessellation Evaluation Shader
Primitive Assembly
Geometry Shader
Primitive Assembly
Rasterizer
Fragment Shader
Access Masks –
What are you Interested in Generating or Consuming 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 Stages and what Access Operations can Happen There
## Access Operations and what Pipeline Stages they can be used In

<table>
<thead>
<tr>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
<th>Stage Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_PIPELINE_STAGE_TOP_OF_PIPE_BIT</td>
<td>VK_PIPELINE_STAGE_DRAW_INDIRECT_BIT</td>
<td>VK_PIPELINE_STAGE_VERTEX_INPUT_BIT</td>
<td>VK_PIPELINE_STAGE_VERTEX_SHADER_BIT</td>
<td>VK_PIPELINE_STAGE_TESSELLATION_CONTROL_SHADER_BIT</td>
<td>VK_PIPELINE_STAGE_TESSELLATION_EVALUATION_SHADER_BIT</td>
<td>VK_PIPELINE_STAGE_GEOMETRY_SHADER_BIT</td>
<td>VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT</td>
<td>VK_PIPELINE_STAGE_EARLY_FRAGMENT_TESTS_BIT</td>
<td>VK_PIPELINE_STAGE_LATE_FRAGMENT_TESTS_BIT</td>
<td>VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT</td>
<td>VK_PIPELINE_STAGE_COMPUTE_SHADER_BIT</td>
<td>VK_PIPELINE_STAGE_TRANSFER_BIT</td>
<td>VK_PIPELINE_STAGE_BOTTOM_OF_PIPE_BIT</td>
<td>VK_PIPELINE_STAGE_HOST_BIT</td>
<td>VK_PIPELINE_STAGE_ALL_GRAPHICS_BIT</td>
<td>VK_PIPELINE_STAGE_ALL_COMMANDS_BIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Access types
- **src**: Indicates source access type
- **dst**: Indicates destination access type
- **dst (no access setting needed)**: Indicates no access setting needed

### Example: Be sure we are done writing an output image before using it for something else

- **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**
The Scenario

src cars are generating the image

dst cars are doing something with that image

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

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_FRAGMENT_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

Access types

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
The Scenario

src cars

TOP_OF_PIPE STREET

BERTEX_INPUT STREET

VERTEX_SHADER STREET

COLOR_ATTACHMENT_OUTPUT STREET

BOTTOM_OF_PIPE STREET

dst cars

VERTEX_SHADER STREET

FRAGMENT_SHADER STREET

TRANSFER_BIT STREET

VK_IMAGE_LAYOUT_UNDEFINED

VK_IMAGE_LAYOUT_GENERAL

VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL

VK_IMAGE_LAYOUT_DEPTH_STENCIL_ATTACHMENT_OPTIMAL

VK_IMAGE_LAYOUT_DEPTH_STENCIL_READ_ONLY_OPTIMAL

VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL

VK_IMAGE_LAYOUT_TRANSFER_SRC_OPTIMAL

VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL

VK_IMAGE_LAYOUT_PREINITIALIZED

VK_IMAGE_LAYOUT_PRESENT_SRC_KHR

VK_IMAGE_LAYOUT_SHARED_PRESENT_KHR

Here, the use of vkCmdPipelineBarrier( ) is to simply change the layout of an image

Used as a color attachment

Read into a shader as a texture

Copy from

Copy to

Show image to viewer

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