Vulkan on Android: Gotchas and best practices

f.garnier@samsung.com    inae13.kim@samsung.com
Vulkan on Android: Gotchas and best practices

• Pipeline barriers
  • Introduction / examples
  • Case study
• Loading times improvement
  • Pipeline Cache
  • Pipeline Derivatives
Galaxy GameDev

• Promote the use of Vulkan on Android
• Support studios remotely & on-site with issues on our devices
• Help game studios port their games to Vulkan
Disclaimers!

Results from Galaxy S7 and Galaxy S8 devices using Arm Mali & Qualcomm Adreno GPUs

From the following games:

- Blade 2 Mobile by Action Square
- AxE by NexonRed
- Final Fantasy 15 by Square Enix
- Lineage II Revolution by Netmarble Games

Results are dependent on the game content and GPU driver
Feature support and optimization advice differs between GPUs
Pipeline barriers
introduction

void vkCmdPipelineBarrier(
    ...
    VkPipelineStageFlags srcStageMask,
    VkPipelineStageFlags dstStageMask,
    ...
);

typedef struct VkImageMemoryBarrier {
    ...
    VkAccessFlags srcAccessMask;
    VkAccessFlags dstAccessMask;
    VkImageLayout oldLayout;
    VkImageLayout newLayout;
    ...
} VkImageMemoryBarrier;
Pipeline barriers

Copying data to an image

- Images in Vulkan are created with a specific tiling arrangement
  - Linear tiling
  - Optimal tiling

- Images also require a valid memory layout
  - Layout is specific to how the image will be used
  - Need a barrier to transition to the correct layout prior to using it

- Only two layouts can be used when creating the image in Vulkan
  - `VK_IMAGE_LAYOUT_UNDEFINED`
  - `VK_IMAGE_LAYOUT_PREINITIALIZED`
Copying data to an image

- Transition my image from an UNDEFINED layout to a TRANSFER layout
- Wait for the pipeline stage in `srcStageMask` to complete
- When that completes, unblock the stage in `dstStageMask`
- Add memory barriers to perform layout transition

```c
{
    srcStageMask = TOP_OF_PIPE_BIT
    dstStageMask = TRANSFER_BIT
    srcAccessMask = 0
    dstAccessMask = TRANSFER_WRITE_BIT
    oldLayout = UNDEFINED
    newLayout = TRANSFER_DST_OPTIMAL
}
```
Copying data to an image

• Need another pipeline barrier prior to sampling the texture
  • Perform layout transition to correct layout
  • Wait for transfer to complete before sampling
  • Ensure transfer writes are complete before shader reads happen

{  
srcStageMask = TRANSFER_BIT  
dstStageMask = FRAGMENT_SHADER_BIT  
srcAccessMask = TRANSFER_WRITE_BIT  
dstAccessMask = SHADER_READ_BIT  
oldLayout = TRANSFER_DST_OPTIMAL  
newLayout=SHADER_READ_ONLY_OPTIMAL 
}
Rendering to an image & presenting

• Images also need to be transitioned to the correct layout before presentation

• Preferable to transition as part of the render pass if possible
  • Can specify an image layout to use per subpass and a final layout
  • Final layout is what the image transitions to at the end of the render pass

• Why not use a subpass dependency for the previous case?
  • Copy command can only be called outside of a render pass instance
Rendering to an image & presenting

typedef struct VkAttachmentReference {
    ...
    VkImageLayout layout;
} VkAttachmentReference;

typedef struct VkAttachmentDescription {
    ...
    VkImageLayout initialLayout;
    VkImageLayout finalLayout;
} VkAttachmentDescription;

typedef struct VkSubpassDependency {
    uint32_t srcSubpass;
    uint32_t dstSubpass;
    VkPipelineStageFlags srcStageMask;
    VkPipelineStageFlags dstStageMask;
    VkAccessFlags srcAccessMask;
    VkAccessFlags dstAccessMask;
    VkDependencyFlags dependencyFlags;
} VkSubpassDependency;
• Transition at the beginning of the renderpass instance may happen out of order
  • Need to make sure presentation engine is done reading the image
  • Subpass dependencies allow us to express the execution & memory dependencies we need
  • Implicit subpass dependencies exist but not suitable for this use case

```c
{
  srcSubpass = VK_SUBPASS_EXTERNAL
  dstSubpass = firstSubpass
  srcStageMask = TOP_OF_PIPE_BIT
  dstStageMask = ALL_COMMANDS_BIT
  srcAccessMask = 0
  dstAccessMask = INPUT_ATTACHMENT_READ_BIT | COLOR_ATTACHMENT_READ_BIT | COLOR_ATTACHMENT_WRITE_BIT | DEPTH_STENCIL_ATTACHMENT_READ_BIT | DEPTH_STENCIL_ATTACHMENT_WRITE_BIT
  VkDependencyFlags = 0
}
```
Rendering to an image & presenting

- Transition the image when it can be rendered to
  - i.e. when made available by semaphore
- Not just limited to synchronizing with presentation engine
  - Render passes can be used for off-screen rendering
    - Next one depends on previous one
  - Stage masks and access flags need to be set accordingly

{  
  srcSubpass = VK_SUBPASS_EXTERNAL  
  dstSubpass = 0  
  srcStageMask = COLOR_ATTACHMENT_OUTPUT_BIT  
  dstStageMask = COLOR_ATTACHMENT_OUTPUT_BIT  
  srcAccessMask = 0  
  dstAccessMask = COLOR_ATTACHMENT_READ_BIT | COLOR_ATTACHMENT_WRITE_BIT  
  VkDependencyFlags = 0
};
Rendering to an image & presenting

• Need a final layout transition to prepare for presentation..
  • Images need to be in \textit{PRESENT\_SRC} layout for presentation
• An implicit post-render pass dependency also exists
  • We don’t need to explicit define this
  • Defines that the transition happens after all work is done aka bottom of pipe

```c
{
    srcSubpass = lastSubpass
    dstSubpass = VK\_SUBPASS\_EXTERNAL
    srcStageMask = ALL\_COMMANDS\_BIT
    dstStageMask = BOTTOM\_OF\_PIPE\_BIT
    srcAccessMask = INPUT\_ATTACHMENT\_READ\_BIT | COLOR\_ATTACHMENT\_READ\_BIT | COLOR\_ATTACHMENT\_WRITE\_BIT | DEPTH\_STENCIL\_ATTACHMENT\_READ\_BIT | DEPTH\_STENCIL\_ATTACHMENT\_WRITE\_BIT
    dstAccessMask = 0
    VkDependencyFlags = 0
};
```
Rendering to an image & presenting

• But we can still define our own post-renderpass dependency
  • Depends on how the image will be accessed and at which stage
• We can change to a specific stage and apply specific access masks

```c
{
    srcSubpass = 0
    dstSubpass = VK_SUBPASS_EXTERNAL
    srcStageMask = COLOR_ATTACHMENT_BIT
    dstStageMask = BOTTOM_OF_PIPE_BIT
    srcAccessMask = COLOR_ATTACHMENT_READ_BIT | COLOR_ATTACHMENT_WRITE_BIT
    dstAccessMask = 0
    VkDependencyFlags = 0
};
```
Case study & best practices
Lineage II Revolution

• Transitioning image to a readable state.. but using wrong stages
  • srcStageMask = FRAGMENT_SHADER_BIT
  • dstStageMask = VERTEX_SHADER_BIT | VERTEX_INPUT_BIT

Vertex   Wait!!   Fragment
...       ...       ...       ...  

Vertex   Vertex
Fragment  Fragment
Case study & best practices
Lineage II Revolution

• Transitioning image to a readable state.. but using wrong stages
  • $srcStageMask = COLOR_ATTACHMENT_OUTPUT_BIT$
  • $dstStageMask = FRAGMENT_SHADER_BIT$
Case study & best practices
Lineage II Revolution

• GLES build – 20.1ms frame time

• Vulkan (Incorrect barrier stages) – 24.5ms (+3.4)

• Vulkan (Optimized barrier stages) – 12.4ms (-8.3)
Case study & best practices
Final Fantasy XV

• Original build – 36ms for 1 frame

• Optimized build – 25 ms (-11) for 1 frame
Create Vulkan pipelines, prepare textures, build command buffers, …
Pipeline Cache

- Calling `vkCreateGraphicsPipelines` without a `VkPipelineCache` will be expensive
- `PipelineCache` is highly recommended

```c
VkResult vkCreateGraphicsPipelines(
    ...,
    VkPipelineCache pipelineCache,
    ...);
```
**Blade 2 : benefit of Pipeline cache**

- Loading time greatly reduced when a pipeline is reused
- Table below shows execution times for `vkCreateGraphicsPipeline` calls
  - over 300 pipelines
  - average pipeline creation time is about 30ms

<table>
<thead>
<tr>
<th>Time for <code>vkCreateGraphicsPipeline</code> (sec)</th>
<th>Scene 1</th>
<th>Scene 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using pipeline cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st</td>
<td>5.523</td>
<td>6.110</td>
</tr>
<tr>
<td>2nd</td>
<td>0.204</td>
<td>0.214</td>
</tr>
<tr>
<td>Without pipeline cache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st</td>
<td>9.021</td>
<td>9.462</td>
</tr>
<tr>
<td>2nd</td>
<td>9.034</td>
<td>9.526</td>
</tr>
</tbody>
</table>
Pipeline Derivatives

- Pipeline derivatives can reduce the cost of creation where a child pipeline has some commonality with a parent pipeline.
- Benefit of using pipeline derivatives vary between GPU architectures.
  
  No harm using them, but may not provide any saving.
Pipeline Derivatives

- No Pipeline Derivatives with VK_PIPELINE_CREATE_DISABLE_OPTIMIZATION_BIT
- Base pipeline (parent) with VK_PIPELINE_CREATE_ALLOW_DERIVATIVES_BIT
  - save base pipeline handle(index)
- Child pipeline with VK_PIPELINE_CREATE_DERIVATIVE_BIT
  - load base pipeline handle(index) for re-use it

Typedef structure VkGraphicsPipelineCreateInfo {
  ...
  VkPipelineCreateFlags flags;
  ...
  VkPipeline basePipelineHandle;
  Int32_t basePipelineIndex;
};
Benefit of using Pipeline Derivatives

- This time is sum of vkCreate*Pipelines by sample data
- Creating 24 pipelines on a Galaxy S7 Snapdragon (msec)

<table>
<thead>
<tr>
<th>Creating 24 pipelines (msec)</th>
<th>shader 1</th>
<th>shader2</th>
<th>shader3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using Pipeline Derivatives</td>
<td>10.61</td>
<td>23.64</td>
<td>13.16</td>
</tr>
<tr>
<td>Without Pipeline Derivatives</td>
<td>223.79</td>
<td>950.31</td>
<td>509.78</td>
</tr>
</tbody>
</table>
Pipeline Cache vs Pipeline Derivatives

- **Pipeline Cache**
  - Saved as a file
  - Pipeline cache stores the entire pipeline
  - Reuse cache when same pipeline

- **Pipeline Derivatives**
  - Created and used at runtime
  - Parent pipeline used as a starting point
  - Reuse when same parent
AxE: benefit of using Pipeline Cache and Pipeline Derivatives

- This is total of vkCreateGraphicsPipeline time (msec) in AxE

<table>
<thead>
<tr>
<th>Time (msec)</th>
<th>1st loading</th>
<th>2nd loading</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nothing</td>
<td>3929</td>
<td>4001</td>
</tr>
<tr>
<td>Pipeline Derivatives</td>
<td>2944</td>
<td>2990</td>
</tr>
<tr>
<td>Pipeline Cache</td>
<td>2377</td>
<td>227</td>
</tr>
<tr>
<td>Cache + Derivatives</td>
<td>2326</td>
<td>144</td>
</tr>
</tbody>
</table>
Thank You!

Frederic Garnier, Inae Kim
Galaxy GameDev, Samsung Electronics
http://developer.samsung.com/game
f.garnier@samsung.com
inae13.kim@samsung.com
gamedev@samsung.com