Mobilizing Call of Duty: Bringing a Blockbuster Title to Android

Reboot Develop Red 2019

Benjamin Mitchell
GameDev Engineer, Samsung Electronics
Galaxy GameDev: since 2016
Bringing Call of Duty to Mobile

- Call of Duty Mobile
- Adaptive Performance
- Vulkan Optimizations
Call of Duty Mobile
Call of Duty Mobile
Call of Duty Mobile
Call of Duty Mobile
Call of Duty Mobile

SAMSUNG

unity

Vulkan. Adaptive Performance
Adaptive Performance
Mobile Limitation

• Temperature
  – No active cooling
  – Performance Throttling

• Power
  – Limitation of Battery

• Performance
  – Hardware Fragmentation

• Optimization is out of the Developer’s control.
Adaptive Performance

• Gather Data
  - Device Performance
  - Thermal Trends

• Adapt Performance
  - Adjust Quality Settings
  - Modify performance requirements

Galaxy Game SDK
Power Manager

Power Budget Control

Default Core Frequencies

With Frequency Cap
Power Manager

Frame Time Tracking

Bottleneck Detection

Frame Time
CPU Time
GPU Time

Target FPS
CPU
GPU

Frame Time
CPU Time
GPU Time

Target FPS
CPU
GPU

1.0 ms
1.0 ms
1.0 ms
Adaptive Performance Quality Scaling

- Quality Scale Factors
  - Shadow/Rendering Distance
  - Texture LoD
  - Animation LoD

- Target Frame Rate
Power Manager + Dynamic Quality Scaling

Frame Rate

- Adaptive Performance
- Without Adaptive Performance
Adaptive Performance is now available

Adaptive Performance

- Initialize on Startup: checked
- Enable Logging: unchecked
- Frequency of Stats logged: 500
- Enable Automatic Performance: checked

Loaders
- Samsung Android Loader
- Simulator Loader
Adaptive Performance is now available
Adaptive Performance in Call of Duty Mobile
Adaptive Performance in Call of Duty Mobile

- Quality Scale Factors
  - Shadow Distance
  - Foliage LoD
  - Animation LoD
  - Target Frame Rate

- Power Management
  - Bottleneck Detection
  - CPU/GPU Power Budget Control
## Adaptive Performance

<table>
<thead>
<tr>
<th>Feature</th>
<th>Parameter</th>
<th>Scaling Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shadow Distance</td>
<td>80 ~ 50</td>
<td>CPU 0~10% offloading</td>
</tr>
<tr>
<td>Foliage LOD</td>
<td>1 ~ 0.8</td>
<td>GPU 0~5% offloading</td>
</tr>
<tr>
<td>Animation LOD</td>
<td>0 ~ 2</td>
<td>CPU 0<del>3% offloading, GPU 0</del>1% offloading</td>
</tr>
<tr>
<td>Target Frame Rate</td>
<td>57 ~ 60</td>
<td>CPU 0<del>5% offloading, GPU 0</del>5% offloading</td>
</tr>
</tbody>
</table>

- **Shadow Distance (80 ~ 50)**
- **Foliage LOD (1.0 ~ 0.8)**
Adaptive Performance

- Adaptive Performance vs Normal Condition

<table>
<thead>
<tr>
<th>Feature</th>
<th>ADP</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPS</td>
<td>58</td>
<td>54</td>
</tr>
<tr>
<td>Stability (%)</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>CPU (%)</td>
<td>11.37</td>
<td>11.69</td>
</tr>
<tr>
<td>GPU (%)</td>
<td>80.08</td>
<td>87.85</td>
</tr>
</tbody>
</table>

![Graph showing FPS comparison between ADP and Normal conditions]
Adaptive Performance

- Can save power and manage thermal better

<table>
<thead>
<tr>
<th>Feature</th>
<th>ADP</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPS</td>
<td>58</td>
<td>54</td>
</tr>
<tr>
<td>Stability (%)</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>CPU (%)</td>
<td>11.37</td>
<td>11.69</td>
</tr>
<tr>
<td>GPU (%)</td>
<td>80.08</td>
<td>87.85</td>
</tr>
</tbody>
</table>
General Vulkan Optimizations

- Pipeline Barriers
  - Ensure most optimal stage masks
  - Batch them!
- Load/Store Ops
  - You don’t always have to care!
  - LoadOpLoad and StoreOpStore
- Cache Stuff
  - PSOs
  - Descriptor Sets
General Vulkan Optimizations

- **Pipeline Barriers**
  - Ensure most optimal stage masks
  - Batch them!
- **Load/Store Ops**
  - You don’t always have to care!
  - LoadOpLoad and StoreOpStore
- **Cache Stuff**
  - PSOs
  - Descriptor Sets
General Vulkan Optimizations

- **Pipeline Barriers**
  - Ensure optimal use of stage masks
  - This is NOT optimal

![Diagram showing pipeline barriers]

**SRC**
- `VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT`

**DST**
- `VK_PIPELINE_STAGE_VERTEX_SHADER_BIT`
General Vulkan Optimizations

- **Pipeline Barriers**
  - Ensure optimal use of stage masks
  - This IS optimal

```
SRC VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT
...
Fragment Shader
...
DST VK_PIPELINE_STAGE_FRAGMENT_SHADER_BIT
```
General Vulkan Optimizations

- Pipeline Barriers

  — Batch them!

```c
vkCmdPipelineBarrier(..., 1, &barrier);
vkCmdPipelineBarrier(..., 1, &barrier);
vkCmdPipelineBarrier(..., 1, &barrier);
vkCmdPipelineBarrier(..., 1, &barrier);
vkCmdPipelineBarrier(..., 1, &barrier);
vkCmdPipelineBarrier(..., 5, &barriers.getData());
```
General Vulkan Optimizations

• Pipeline Barriers
  • Ensure most optimal stage masks
  • Batch them!
• Load/Store Ops
  • You don’t always have to care!
  • LoadOpLoad and StoreOpStore
• Cache Stuff
  • PSOs
  • Descriptor Sets
General Vulkan Optimizations

- **Load/Store Ops**

  - You don’t always have to care!

```
colorAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_LOAD
colorAttachment.storeOp = VK_ATTACHMENT_STORE_OP_STORE
```

```
colorAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE
colorAttachment.storeOp = VK_ATTACHMENT_STORE_OP_DONT_CARE
```
General Vulkan Optimizations

- **Pipeline Barriers**
  - Ensure most optimal stage masks
  - Batch them!

- **Load/Store Ops**
  - You don’t always have to care!
  - LoadOpLoad and StoreOpStore

- **Cache Stuff**
  - Use VKPipelineCache
  - Descriptor Sets
General Vulkan Optimizations

- Cache Stuff
  - Use VkPipelineCache
  - Descriptor Sets

1st Run

```c
vkCreateGraphicsPipelines(device, VK_NULL_HANDLE, ...);
```

2nd Run

```c
vkCreateGraphicsPipelines(device, VK_NULL_HANDLE, ...);
```

1st Run

```c
vkCreateGraphicsPipelines(device, pipelineCache, ...);
```

2nd Run

```c
vkCreateGraphicsPipelines(device, pipelineCache, ...);
```
General Vulkan Optimizations

- **Cache Stuff**
  - Use VkPipelineCache
  - **Descriptor Sets**

1st Frame

```cpp
vkUpdateDescriptorSets(device, …);
```

All remaining Frames

```cpp
vkUpdateDescriptorSets(device, …);
```

1st Frame

```cpp
vkUpdateDescriptorSets(device, …);

someDescriptorMapemplace(resourceKey, &descriptorSet);
```

All remaining Frames

```cpp
descriptorSet = someDescriptorMap.find(resourceKey);
```
General Vulkan Optimizations

- Pipeline Barriers
  - Ensure most optimal stage masks
  - Batch them!
- Load/Store Ops
  - You don't always have to care!
  - LoadOpLoad and StoreOpStore
- Cache Stuff
  - 5-10% CPU time
  - Descriptor Sets

*Up to 40% performance*

*Up to 50% bandwidth*
General Vulkan Optimizations

- WaitForFence
  - Try to avoid it!
  - Use a pool of Command Buffers
- RenderPasses
  - Minimize switching
  - Combine passes!
- Threads
  - Put stuff on other threads.
General Vulkan Optimizations

• **WaitForFence**
  • Try to avoid it!
  • Use a pool of Command Buffers
• **RenderPasses**
  • Minimize switching
  • Combine passes!
• **Threads**
  • Put stuff on other threads.
General Vulkan Optimizations

- **WaitForFences**
  - Try to avoid it!
  - Make use of pools of command buffers

```c
if(!fencePool.empty())
    fence = fencePool.pop_back();
else
    fence = vkCreateFence(..., &fenceToUse);
result = vkGetFenceStatus(device, fence);
if(result == VK_STATUS_NOT_READY)
    fencePool.push_back(fence);
```
General Vulkan Optimizations

• **WaitForFence**
  - Try to avoid it!
  - Use a pool of Command Buffers

• **RenderPasses**
  - Minimize switching
  - Combine passes!

• **Threads**
  - Put stuff on other threads.
General Vulkan Optimizations

- **RenderPasses**
  - Minimize switching

```
vkCmdBeginRenderPass(commandBuffer, &beginInfo, contents);
DrawObjects1();
vkCmdEndRenderPass(commandBuffer);

vkCmdBeginRenderPass(commandBuffer, &beginInfo, contents);
DrawObjects2();
vkCmdEndRenderPass(commandBuffer);

vkCmdBeginRenderPass(commandBuffer, &beginInfo, contents);
DrawObjects1();
DrawObjects2();
vkCmdEndRenderPass(commandBuffer);
```
General Vulkan Optimizations

- **WaitForFence**
  - Try to avoid it!
- **Use a pool of Command Buffers**
- **RenderPasses**
  - Minimize switching
  - Combine passes!
- **Threads**
  - Put stuff on other threads.

Up to 10% Performance
1-5% Performance
General Vulkan Optimizations

• Threads

Original

Thread 1

Task 1  Task 2  Task 3  Task 1

Multi-Threaded

Thread 1

Task 1  Task 2  Task 1  Task 2

Thread 2

Task 3  Task 3
General Vulkan Optimizations

- **WaitForFence**
  - Try to avoid it!
- **Use a pool of Command Buffers**
- **RenderPasses**
  - Minimize switching
  - Combine passes!
- **Threads**
  - Put stuff on other threads.

- Up to 10% Performance
- 1-5% Performance
- Up to 30% Performance
Vulkan Optimizations in Call of Duty Mobile

- Single Scratch Buffer
- Low Priority Destroy Thread
- Descriptor Set Batching/Caching
- RenderPass Load/Store
- Threaded Present
- VB, IB Binding Optimization
- Direct Buffer Access
Vulkan Optimizations

- Single Scratch Buffer
- Low Priority Destroy Thread
- Descriptor Set Batching/Caching
- RenderPass Load/Store
- Threaded Present
- VB, IB Binding Optimization
- Direct Buffer Access
Vulkan Optimizations

- **Single Scratch Buffer**
  - Ring buffer for all needs
  - Usually recreated only during first load

![Diagram](image)
Vulkan Optimizations

- Single Scratch Buffer
- **Low Priority Destroy Thread**
- Descriptor Set Batching/Caching
- RenderPass Load/Store
- Threaded Present
- VB, IB Binding Optimization
- Direct Buffer Access
Vulkan Optimizations

- **Low Priority Destroy Thread**

  - Render thread
    - Delayed Destroy
    - Delayed Destroy
    - Delayed Destroy
    - Garbage Collect

  - Delayed Destroy

  - Destroy thread
    - Wait
    - vkDestroy***
    - vkDestroy***
    - vkDestroy***
Vulkan Optimizations

- Single Scratch Buffer
- Low Priority Destroy Thread
- **Descriptor Set Batching/Caching**
- RenderPass Load/Store
- Threaded Present
- VB, IB Binding Optimization
- Direct Buffer Access
Vulkan Optimizations

• Descriptor Set Batching/Caching

— Combine separate Update calls into one
Vulkan Optimizations

- **Descriptor Set Batching/Caching**
  - Cache and reuse sets with same resources

Allocate and update Descriptor Sets before each draw call

Reuse Descriptors Sets from cache
Vulkan Optimizations

- **Descriptor Set Batching/Caching**
  
  — Cache and reuse sets with same resources

Allocate and update Descriptor Sets before each draw call

Reuse Descriptors Sets from cache
Vulkan Optimizations

- **Descriptor Set Batching/Caching**

  — Cache and reuse sets with same resources

Allocate and update Descriptor Sets before each draw call

Reuse Descriptors Sets from cache
Vulkan Optimizations

- Single Scratch Buffer
- Low Priority Destroy Thread
- Descriptor Set Batching/Caching
- **RenderPass Load/Store**
- Threaded Present
- VB, IB Binding Optimization
- Direct Buffer Access
Vulkan Optimizations

- **RenderPass Load/Store**

  When render target is being fully redrawn there is no need to load previous data.
Vulkan Optimizations

- Single Scratch Buffer
- Low Priority Destroy Thread
- Descriptor Set Batching/Caching
- RenderPass Load/Store
- Threaded Present
- VB, IB Binding Optimization
- Direct Buffer Access
Vulkan Optimizations

- Threaded Present

Original

Threaded present
Vulkan Optimizations

- Single Scratch Buffer
- Low Priority Destroy Thread
- Descriptor Set Batching/Caching
- RenderPass Load/Store
- Threaded Present
- **VB, IB Binding Optimization**
- Direct Buffer Access
Vulkan Optimizations

• VB, IB Binding Optimization

As buffers are same there is no need to bind them every draw call.
Vulkan Optimizations

- Single Scratch Buffer
- Low Priority Destroy Thread
- Descriptor Set Batching/Caching
- RenderPass Load/Store
- Threaded Present
- VB, IB Binding Optimization
- Direct Buffer Access
Vulkan Optimizations

- **Direct Buffer Access**

  Original

  Main thread: Upload (vertex/index/uniform data) → Memory → Copy → Device buffer

  Render thread: Memory → Copy → Device buffer

  Direct access

  Main thread: Upload (vertex/index/uniform data) → Device buffer

  Render thread: Device buffer
Top 3 Takeaways

1: Use Adaptive Performance!

gamedev@samsung.com
Top 3 Takeaways

1: Use Adaptive Performance!
2: Check your barriers!

gamedev@samsung.com
Top 3 Takeaways

1: Use Adaptive Performance!
2: Check your barriers!
3: Check your load/store ops!

gamedev@samsung.com
Top 3 Takeaways

1: Use Adaptive Performance!
2: Check your barriers!
3: Check your load/store ops!
3.5: Thread stuff!

gamedev@samsung.com
Thank You!

gamedev@samsung.com

benjamin.m@samsung.com