Porting Source 2 to Vulkan

Dan Ginsburg
Valve
Summary

- Source 2 Overview
- Porting to Vulkan
  - Shaders and Pipelines
  - Command Buffers
  - Memory Management
  - Descriptor Sets
Source 2 Overview
Source 2

- OpenGL, DX9, DX11, Vulkan
- Windows, Linux, Mac
- Dota 2 Reborn
Source 2 Rendering

- DX11-like rendersystem abstraction
- Multithreaded
  - DX9/GL: software command buffers
  - DX11: deferred contexts
  - Single submission thread
Source 2 Rendering (GL)
Source 2 Rendering (GL)
Source 2 Vulkan Port

- Started with GL and DX11 renderer
  - DX11 deferred contexts mapped well to Vulkan command buffers
  - Leveraged GLSL shader conversion
Shaders and Pipelines
Porting Shaders to Vulkan

• HLSL -> GLSL
  – See: *Moving Your Games to OpenGL, Steam Dev Days 2014*

• GLSL -> SPIR-V
  – Descriptor set layout qualifiers to GLSL
  – Open source glslang SPIR-V backend
  – SPVremapper for compression
  – [https://github.com/KhronosGroup/glslang](https://github.com/KhronosGroup/glslang)
Pipeline State Objects (PSOs)

• Each thread caches pipeline state
• Global pipeline manager
  – PSO Map
  – Pending and current
    • Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
  - Reduces mutexing
Pipeline State Objects (PSOs)

• Each thread caches pipeline state
• Global pipeline manager
  – PSO Map
  – Pending and current
    • Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
  - Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
- Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
    - Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
- Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
- Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
- Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
- Reduces mutexing
Pipeline State Objects (PSOs)

- Each thread caches pipeline state
- Global pipeline manager
  - PSO Map
  - Pending and current
- Reduces mutexing
Command Buffers
Command Buffers

- Used where DX11 deferred contexts were used
- Each thread builds command buffer
- Single thread performs submission to queue
Command Buffers

- Used where DX11 deferred contexts were used
- Each thread builds command buffer
- Single thread performs submission to queue
Command Buffers

- Used where DX11 deferred contexts were used
- Each thread builds command buffer
- Single thread performs submission to queue
Command Buffers

- Used where DX11 deferred contexts were used
- Each thread builds command buffer
- Single thread performs submission to queue
Command Buffers

- Recycled within per-thread pools

Thread 0
CommandPool

Thread 1
CommandPool

Thread 2
CommandPool

Thread0
CmdBuf  CmdBuf

Thread1
CmdBuf  CmdBuf

Thread2
CmdBuf  CmdBuf  CmdBuf

Submit
Submit1  Submit2

Main

Check Fences
Command Buffers

- Recycled within per-thread pools
Command Buffers

- Recycled within per-thread pools
Command Buffers

- Recycled within per-thread pools
Command Buffers

- Recycled within per-thread pools

Thread 0 CommandPool
Thread 1 CommandPool
Thread 2 CommandPool

Thread0  CmdBuf  CmdBuf
Thread1  CmdBuf  CmdBuf
Thread2  CmdBuf  CmdBuf  CmdBuf
Submit   Submit1  Submit2
Main     Check Fences
Command Buffers

- Recycled within per-thread pools
Command Buffers

- Recycled within per-thread pools

Thread 0 CommandPool
Thread 1 CommandPool
Thread 2 CommandPool

Submit 1
Submit 2
Check Fences
Command Buffers

- Recycled within per-thread pools
Command Buffers

- Recycled within per-thread pools
Command Buffers

- Recycled within per-thread pools
Command Buffer Performance

- Submit in batches
  - `vkQueueSubmit` has cost on Windows
  - Faster to group submissions together
- Minimize number of command buffers
- Minimize memory referenced per command buffer
- Use `VK_CMD_BUFFER_OPTIMIZE_ONE_TIME_SUBMIT_BIT`
  - Optimize for one-time submission
Memory Management
General Strategies

- Pool resources together
  - Reduces memory reference count
- Use per-thread pools to reduce contention
- Recycle dynamic pools on frame boundaries
## Resources

<table>
<thead>
<tr>
<th>Static Resources</th>
<th>Dynamic Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Pools</td>
<td>Per-Thread Pools</td>
</tr>
<tr>
<td>Device Only</td>
<td>Host Visible (Persistently Mapped)</td>
</tr>
<tr>
<td>Textures/Render Targets</td>
<td>VB/IB/CBs</td>
</tr>
<tr>
<td>• 128MB Pools</td>
<td>• 8MB Pools</td>
</tr>
<tr>
<td>VB/IB/CBs</td>
<td></td>
</tr>
<tr>
<td>• 8MB Pools</td>
<td></td>
</tr>
</tbody>
</table>
Dynamic Vertex/Index Buffers

• To update:
  – Grab new offset from per-thread pool
  – memcpy into pool
  – Bind VBs with: vkCmdBindVertexBuffers(..,buffer,offset)
  – Bind IBs with: vkCmdBindIndexBuffer(..,buffer,offset,..)
• Recycle pools when last GPU fence of frame retires
Dynamic Uniform Buffers

- Differences from VB/IBs:
  - UBOs are bound via descriptors
  - Use dynamic UBOs to avoid vkUpdateDescriptorSets
  - Pass UBO offset to vkCmdBindDescriptorSets
Dynamic Textures

• Staged in persistently mapped buffers
  – Recycled per-frame
• Copy with vkCmdCopyBufferToImage
Descriptor Sets
Descriptor Set - Ideal

- Allocate and bake descriptor sets up front
- Group sets by update frequency
- Only update changed sets
Descriptor Set - Reality

- Difficult to bake descriptors with DX11-like abstraction
- Our approach
  - Pre-allocate descriptor sets with fixed slots
  - Only bind to used slots
  - Update descriptors each draw
Summary

- Source 2 Overview
- Porting to Vulkan
  - Shaders and Pipelines
  - Command Buffers
  - Memory Management
  - Descriptor Sets
Questions?

- **dang@valvesoftware.com**
- **Khronos BOF:**
  - Wed. August 12\(^{th}\), 5:30-7:30
  - JW Marriott LA Live in the Platinum Ballroom Salon F-I