Bringing Fire to the Volcano

AKA: Rendering Galaxy on Fire 3: Manticore with Vulkan on Mobile Devices

2017-05-25 / Vulkanised! / Deep Silver FISHLABS / Johannes Kuhlmann
Why are you here?
Road to Vulkan
Starting Point

Gameplay

Render Pipeline

- Uniforms
- Culling
- Sorting
- ...

OpenGLRenderer

MetalRenderer

VulkanRenderer
**DescriptorSet Bindings**

**Uniforms**
- 1 DescriptorSet per shader
- use `VK_DESCRIPTOR_TYPE_UNIFORM_BUFFER_DYNAMIC`
- 1 big buffer
  - for all objects
  - for all in-flight frames

**Textures**
- our API accepts arbitrary shaders/textures combination
- tried updating DescriptorSets
  - too slow
  - so... cache‘em
  - `{Shader, Textures}`->`DescriptorSet`
Pipeline State Management

Pipeline State

- combines
  - vertex layout
  - shader
  - render target
  - raster/blend state
- have to be managed somehow
- take long to be compiled

Solution

- cache‘em
  - hash all inputs
  - if not compiled yet, compile async
  - on load, pre-warm cache
Don’t forget the Asset Pipeline

Precompile Shaders

- we use
  - Google’s shaderc
  - GLSL -> SPIR-V
  - can share GLSL code that way

Shader Reflection

- bring your own
- SPIRV-Cross
- just assume?
What did we learn on the road?
You have to pay respect

- Maximum allocation count
- Memory alignment requirements
- API version
- Texture compression formats
- Framebuffer formats
- Composite alpha flags
Validity doesn’t imply correctness

Super useful, use them!
Load validation layers in order
Validation layers WIP

Device specific
May go missing due to Play Libs
It’s easy to lose your device

VK_ERROR_DEVICE_LOST

Super difficult to debug

Causes include, but not limited to

- garbage uniforms
- unbound textures
- synchronization issues
Drivers have issues, too

E.g. Adreno

- DescriptorSet binding order
- dynamic pipeline state

Devices behave differently

- can/cannot map buffer twice
- render sub pass dependencies
- texture depth == 0
Vulkan can run on Android 6

Cannot depend on libvulkan.so

- load dynamically
- may use Vulkan wrapper

But 6 != 6

- API version can be 0.0.1
- can get VK_ERROR_INCOMPATIBLE_DRIVER
You have to handle it (duh)

Surface is destroyed and re-created

Must re-create swapchain and its framebuffers
Some Numbers
Lines of Code

LOCs

- OpenGL: 3108
- Metal: 3493
- Vulkan: 4401
Timing Drawcalls

Generating ~1400 draw calls [ms]

<table>
<thead>
<tr>
<th>Device</th>
<th>OpenGL ES 3</th>
<th>Vulkan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nexus 5X</td>
<td>71.4</td>
<td>26.4</td>
</tr>
<tr>
<td>Galaxy S7</td>
<td>32.1</td>
<td>11.3</td>
</tr>
<tr>
<td>Google Pixel XL</td>
<td>33.1</td>
<td>8.2</td>
</tr>
</tbody>
</table>
Summary

If slow, cache it!

Vulkan does indeed improve performance

Expect work from fixing for different drivers/devices
Thank you!

Questions?

j.kuhlmann@dsfishlabs.com / @j66k
class Renderer { public:
    // -- Lifecycle handling --
    virtual void initialize() = 0;
    virtual void createContext(RenderContextState&, RenderTargetDescriptor const&, bool debug) = 0;
    virtual void destroyContext() = 0;

    // -- Per-frame rendering --
    virtual ParameterBuffer* getParameterBuffer() = 0;
    virtual void execute(const RenderQueue&) = 0;

    // -- Render pipeline states --
    virtual HardwareResourceHandle getPipeline(Mesh const&, RenderState const&, Shader const&, RenderTarget const*, bool sync);
    virtual std::size_t removeUnusedPipelines(std::size_t minUnusedFrames);

    // -- Resources --
    virtual Texture* newTexture(const TextureDescriptor&, task::Task* parentTask = 0) = 0;
    virtual void destroyTexture(Texture*) = 0;
    virtual Mesh* newMesh(const MeshDescriptor&, task::Task* parentTask = 0) = 0;
    virtual void destroyMesh(Mesh* mesh) = 0;
    virtual Shader* newShader(const ShaderDescriptor&, task::Task* parentTask = 0) = 0;
    virtual void destroyShader(Shader*) = 0;
    virtual RenderState* newRenderState(const RenderStateDescriptor&) = 0;
    virtual void destroyRenderState(RenderState*) = 0;

    // -- Dynamic meshes --
    virtual MappedBufferHandle* mapDynamicMeshIndexBuffer(Mesh&, RenderBuffer& frameContext) = 0;
    virtual void unmmapDynamicMeshIndexBuffer(MappedBufferHandle*, std::size_t indexCount) = 0;
    virtual MappedBufferHandle* mapDynamicMeshVertexBuffer(Mesh&, std::size_t vertexBufferIndex, RenderBuffer& frameContext) = 0;
    virtual void unmmapDynamicMeshVertexBuffer(MappedBufferHandle*, std::size_t vertexCount, Aabb const&) = 0;

    // -- Render targets --
    virtual void createRenderTarget(RenderTarget*) = 0;
    virtual void destroyRenderTarget(RenderTarget*) = 0;
    virtual void resizeRenderTarget(RenderTarget*) = 0;
    virtual void copyRenderTargetToMemory(RenderTarget*) = 0;
};
Synchronization

Acquire

Semaphores

Abyss Command Buffer

Scaleform Command Buffer

Pre-presentation Command Buffer

Present

Abyss Command Buffer

RenderPass 0

RenderPass 1

RenderPass ...

Fence
Implementation: Dynamic Meshes

• Buffer size *= maxInFlightFrames
• Cycle through ranges of buffer per frame
• “stream” mode, must write every frame
• Note: cannot map buffer twice; so write indices/vertices after each other
Debugging Tips

• Of course, validation layers
• Tools: Renderdoc
• vkDeviceWaitIdle()
• Check result after each function, print error