Porting Roblox to Vulkan

Arseny Kapoulkine
@zeuxcg
What is Roblox?

- Online multiplayer game creation platform
- All content is user generated
- Windows, macOS, iOS, Android, Xbox One
- 50M+ MAU, 1.5M+ CCU
Jailbreak

- Cops and robbers
- Two primary developers (1 artist, 1 programmer)
- Peak 130K CCU in February 2018

You can rob banks after you escape, but you'll need a keycard!
Why Vulkan?

- Lots of performance challenges on Android
- Need maximum performance without tweaking content
- Need modern* GAPI features for current/future rendering projects
- Long term desire to discontinue OpenGL
It's going to be easy, right?

- D3D9, D3D11, GL2/3, GLES2/3, Metal, GNM*...
- What's one more API to support?
- Drivers should be pretty good since API is so much simpler...
I have not failed. I've just found 10,000 ways that won't work.
— Thomas A. Edison
Shaders

- Shared & familiar shading language, would like to keep it

```cpp
LightingVertexOutput ParticleLightingVS(Appdata IN)
{
    LightingVertexOutput OUT;

    float2 uv = getLightingUV(IN.lightuv, IN.disp);

    OUT.HPosition = float4(uv.x * 2 - 1, 1 - uv.y * 2, 0, 1);
    OUT.LightPosition = lgridPrepareSample(IN.worldPos);

    #ifdef DX9
    OUT.HPosition.xy += CB1.AtlasParams.xy * float2(-1, 1);
    #endif

    return OUT;
}
```
Shader toolchain: OpenGL

- github.com/Thekla/hlslparser: HLSL -> GLSL
- github.com/aras-p/glsl-optimizer: GLSL -> GLSL
Shader toolchain: Vulkan

- [github.com/Thekla/hlslparser](https://github.com/Thekla/hlslparser): HLSL -> GLSL
- [glslang](https://github.com/KhronosGroup/glslang): GLSL -> SPIRV
Shader toolchain: Vulkan, take 2

- [github.com/Thekla/hlslparser](https://github.com/Thekla/hlslparser): HLSL -> GLSL
- [github.com/aras-p/glsl-optimizer](https://github.com/aras-p/glsl-optimizer): GLSL -> GLSL
- [glslang](https://www.khronos.org/collada/glslang): GLSL -> SPIRV
Shader toolchain: Vulkan, actual take 2

- github.com/Thekla/hlslparser: HLSL $\rightarrow$ GLSL
- github.com/aras-p/glsl-optimizer: GLSL $\rightarrow$ GLSL
- `std::regex`: replace uniform variables with UBOs
- glslang: GLSL $\rightarrow$ SPIRV
Shader toolchain: Vulkan, future

- glslang / DXC: HLSL -> SPIRV
- spirv-opt: SPIRV -> SPIRV (-Os)
- spirv-opt: SPIRV -> SPIRV (strip-debug for OpName)
- spirv-val is essential!
API

- Shared & convenient interface, would like to keep it

```cpp
PassClear passClear;
passClear.mask = Framebuffer::Mask_Color0;

ctx->beginPass(fb, 0, Framebuffer::Mask_Color0, &passClear);

ctx->bindProgram(program.get());
ctx->bindBuffer(0, &globalData, sizeof(globalData));
ctx->bindBuffer(1, &params, sizeof(params));
ctx->bindTexture(0, lightMap, SamplerState::Filter_Linear);

ctx->draw(geometry, Geometry::Primitive_Triangles, 0, count);

ctx->endPass();
```
Implementation - dumb parts

- Lazily create and cache everything
- Serialize pipeline cache to disk to minimize stalls
- Dynamic descriptor sets (allocate, update, bind)
- Generic memory allocator, predates AMD's

github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator
Implementation - smart* parts

- Defer resource destruction/reuse until frame is executed
- No layout tracking, transition on `beginPass / endPass` boundaries*
- Avoid descriptor allocation/updates (dynamic descriptors)
- Carefully optimize everything

github.com/zeux/volk
"It's hard to beat the driver"

- Faster CPU dispatch, matching GPU performance
- Does NOT require a mindset change! YMMV

- Desktop: test level @ 5090 draw calls, single core
  - NVidia: DX11 10 ms (w/driver threading: 4.5 ms), Vulkan 3.6 ms
  - AMD: DX11 11.4 ms (w/driver threading: 5.2 ms), Vulkan 5.8 ms
  - Intel: DX11 25 ms, Vulkan 12.8 ms

- Mobile: test level @ 1130 draw calls, single core
  - Qualcomm: GL 9.8 ms, Vulkan 3.8 ms
  - ARM: GL 15.3 ms, Vulkan 5.9 ms
  - PowerVR: GL 29.0 ms*, Vulkan 7.8 ms
Vulkan on Android: stats of doom

- 17% of our Android users can use Vulkan (>2M MAU!)
- Android version stats:
  - 6.0 → 0.8%
  - 7.0 → 82.0%
  - 7.1 → 14.5%
  - 8.0 → 1.7%
  - 8.1 → 1.0%
- Most users will have drivers released months after 1.0 spec
Synchronization 101

- Rule #1: You don't understand barrier semantics
- Rule #2: Your barrier code is wrong
- No validation for synchronization... 2018? please?

github.com/KhronosGroup/Vulkan-Docs/wiki/Synchronization-Examples
github.com/Tobski/simple_vulkan_synchronization
Command buffer management

- `vkFreeCommandBuffers` can be a no-op!
- Use `vkResetCommandPool` or `vkResetCommandBuffer`
- Need NxM `VkCommandPool` objects for multi-threaded rendering
Shader compiler bugs, rollup

- Sampler function arguments don't work (always)
- Local struct variables don't work (sometimes)
- Varying `OpName` names have to match between VS & FS
- Entrypoint `OpName` must be equal to OpEntryPoint name
- Combined image & sampler descriptors have to be combined in SPIRV
- `vkCreateGraphicsPipelines` returns `VK_INCOMPLETE`
- Gaps in UBO binding indices can lead to incorrect rendering
- Simple control flow is miscompiled to always take the branch
Driver bugs, rollup

- VkInstanceCreateInfo::pApplicationInfo=NULL crashes in driver
- Barrier with VK_REMAINING_ARRAY LAYERS crashes in driver
- stencilLoadOp=LOAD only works with depth loadOp=LOAD
- loadOp=DONT_CARE doesn't work with swapchain image
- vkCmdBlitImage ignores layerCount for cubemaps
- vkCmdCopyBufferToTexture mishandles Z offset for 3D textures
- Red and blue channels are swapped in swapchain image
- VK_SUBPASS_EXTERNAL dependencies don't work
- vkCmdBindPipeline disturbs vertex buffer bindings
- Incomplete initial data crashes vkCreatePipelineCache
Driver bugs, what to do?

• Report conformance bugs!
  ○ github.com/KhronosGroup/VK-GL-CTS

• Talk to platform holders / GPU vendors
  ○ Samsung developer.samsung.com/game
  ○ Google issuetracker.google.com

• Prepare to work around bugs

• Prepare to blacklist devices
Driver workarounds & blacklists

uint32_t apiVersion; // spec version; patch version not useful
uint32_t driverVersion; // semver, CL#, binary hash
uint32_t vendorID; // 0x8086
uint32_t deviceID; // not generally useful

- We use a combination of vendorID and Android ABI version*
- For blacklisting we pattern-match device model names
Bug reporting guide

- [github.com/KhronosGroup/](https://github.com/KhronosGroup/)
- Found a bug in your code?
  - Report to Vulkan-LoaderAndValidationLayers
- Found a bug in the driver?
  - Report to VK-GL-CTS
  - Report to GPU vendors
- Found a bug in shader toolchain?
  - Report to glslang / DXC / SPIRV-Tools as appropriate
  - Report missing validation to SPIRV-Tools
Conclusion

- Vulkan is practical today
- Requires high pain tolerance, but gains are worth it
- Help us make Vulkan better!
Thank you!

arseny@roblox.com