From OpenGL to Vulkan™

Khronos Munich Chapter Meeting 2016/04/08

Sascha Willems

www.saschawillems.de / @SaschaWillems2

Introduction

- Started using OpenGL around 2000 (as a hobby)
  - Initially with Delphi, but moved to C++
  - Helped founding the DelphiGL OpenGL Wiki
  - Still maintain the OpenGL Pascal Header translations
- Not a professional 3D developer
- Maintainer of the OpenGL/ES Hardware Databases
- **Member of the Khronos Vulkan Advisory Panel**
- **Vulkan launch day contributions**
  - Open Source Vulkan examples (C++)
  - Vulkan hardware database

Vulkan Hardware Database

GPUinfo.org

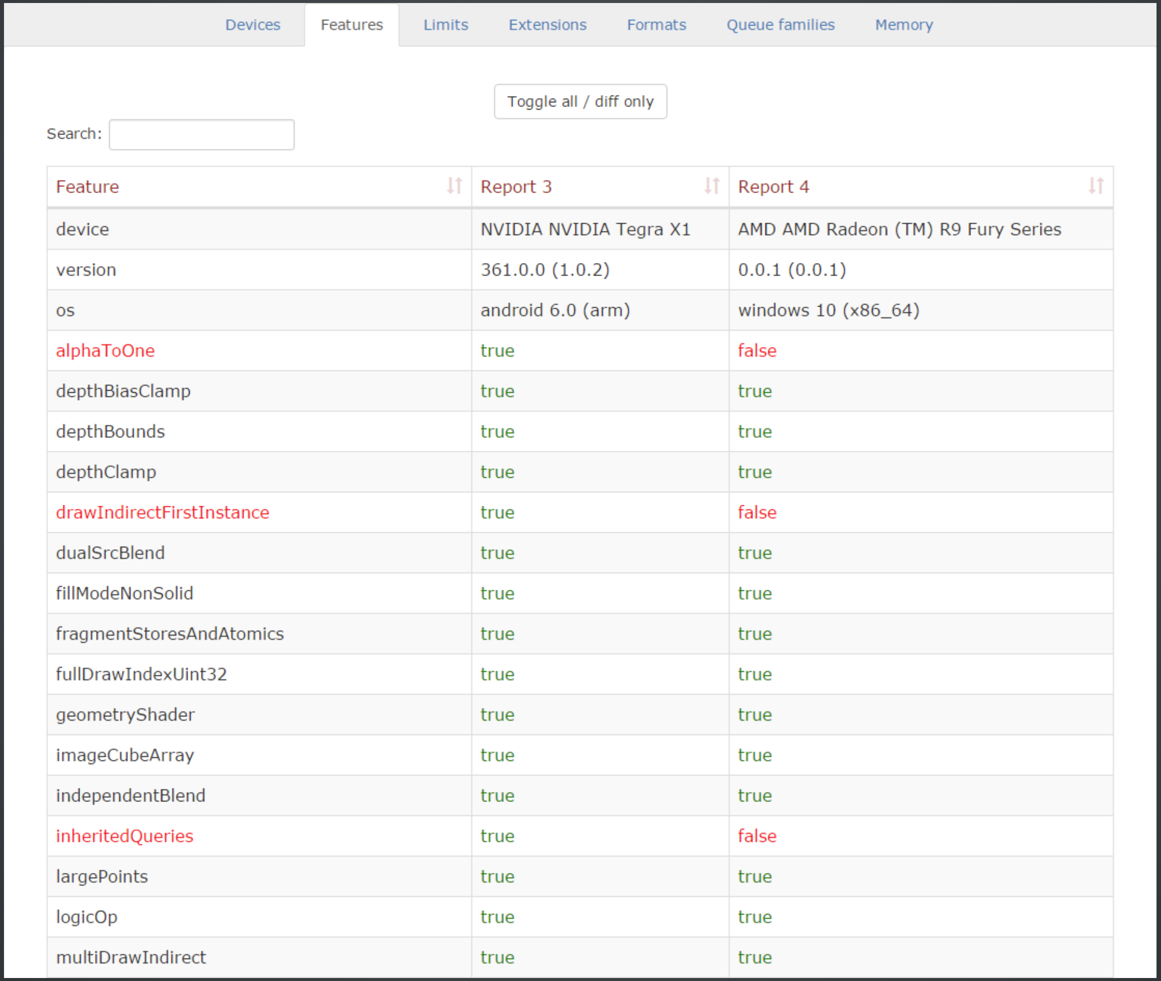
Vulkan Hardware Database

- Open Source [1]
  - Client for Linux, Windows and Android (C++, Qt)
  - Online data base (PHP, MySQL)
- Contains all implementation info available to the API
  - Features and limits
  - Format information (incl. flags)
  - Queue families
  - Memory properties
  - And more...
- Global statistics (extensions, formats)
- Compare implementations
- Over 200 reports (and counting)

[1] https://github.com/SaschaWillems/VulkanCapsViewer

Vulkan Hardware Database

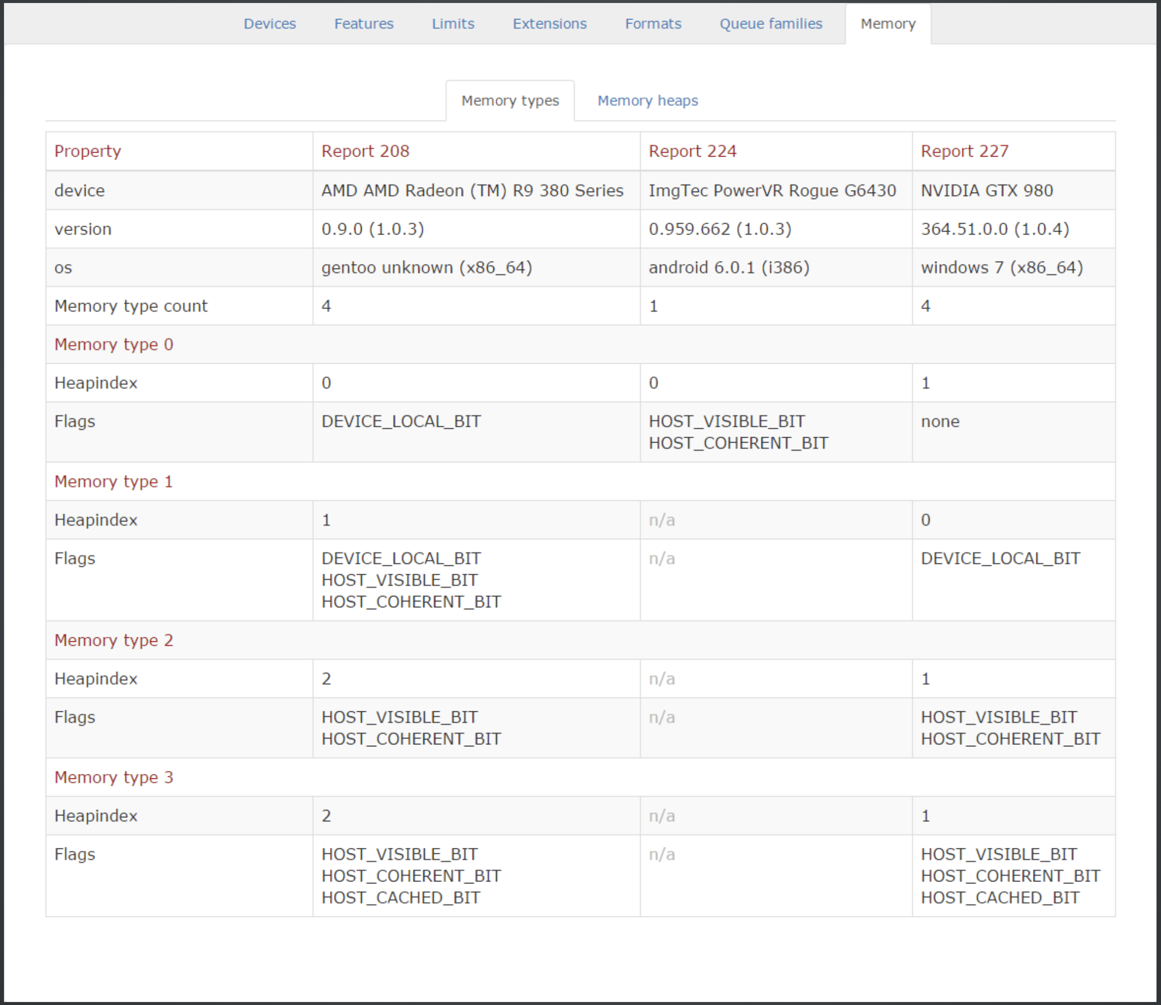
Comparing features



Feature	Report 3	Report 4
device	NVIDIA NVIDIA Tegra X1	AMD AMD Radeon (TM) R9 Fury Series
version	361.0.0 (1.0.2)	0.0.1 (0.0.1)
os	android 6.0 (arm)	windows 10 (x86_64)
alphaToOne	true	false
depthBiasClamp	true	true
depthBounds	true	true
depthClamp	true	true
drawIndirectFirstInstance	true	false
dualSrcBlend	true	true
fillModeNonSolid	true	true
fragmentStoresAndAtomics	true	true
fullDrawIndexUint32	true	true
geometryShader	true	true
imageCubeArray	true	true
independentBlend	true	true
inheritedQueries	true	false
largePoints	true	true
logicOp	true	true
multiDrawIndirect	true	true

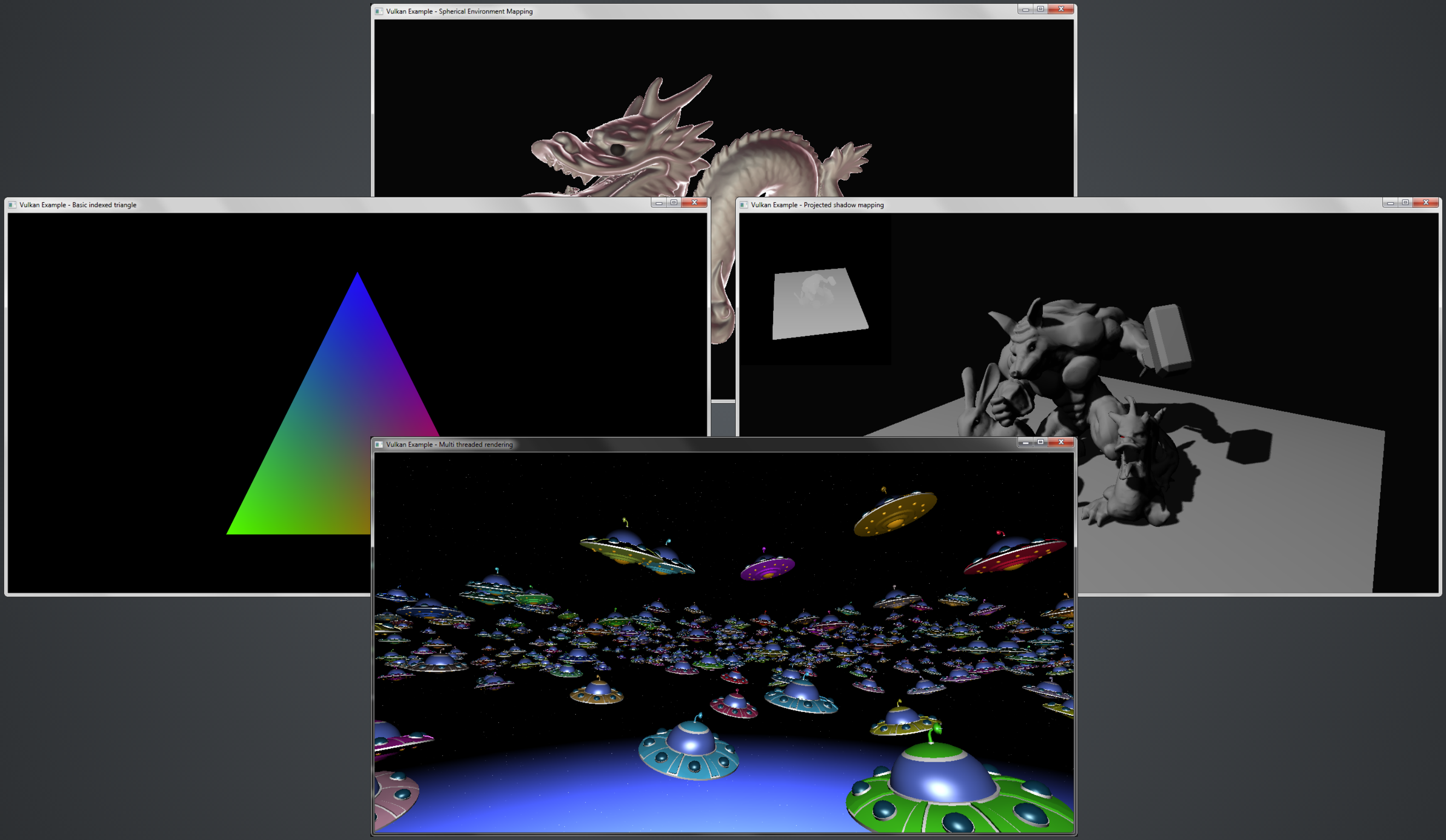
Vulkan Hardware Database

Comparing memory types



Property	Report 208	Report 224	Report 227
device	AMD AMD Radeon (TM) R9 380 Series	ImgTec PowerVR Rogue G6430	NVIDIA GTX 980
version	0.9.0 (1.0.3)	0.959.662 (1.0.3)	364.51.0.0 (1.0.4)
os	gentoo unknown (x86_64)	android 6.0.1 (i386)	windows 7 (x86_64)
Memory type count	4	1	4
Memory type 0			
Heapindex	0	0	1
Flags	DEVICE_LOCAL_BIT	HOST_VISIBLE_BIT HOST_COHERENT_BIT	none
Memory type 1			
Heapindex	1	n/a	0
Flags	DEVICE_LOCAL_BIT HOST_VISIBLE_BIT HOST_COHERENT_BIT	n/a	DEVICE_LOCAL_BIT
Memory type 2			
Heapindex	2	n/a	1
Flags	HOST_VISIBLE_BIT HOST_COHERENT_BIT	n/a	HOST_VISIBLE_BIT HOST_COHERENT_BIT
Memory type 3			
Heapindex	2	n/a	1
Flags	HOST_VISIBLE_BIT HOST_COHERENT_BIT HOST_CACHED_BIT	n/a	HOST_VISIBLE_BIT HOST_COHERENT_BIT HOST_CACHED_BIT

Vulkan examples


Vulkan Example - Spherical Environment Mapping
Vulkan Example - Basic indexed triangle
Vulkan Example - Projected shadow mapping
Vulkan Example - Multi threaded rendering

https://github.com/SaschaWillems/Vulkan

Vulkan Examples

- Demonstrating Vulkan functionality and techniques
- From an explicit "Hello World" triangle...
  - Loading meshes
  - Using pipelines
  - Multi sampling
  - Deferred rendering (MRT)
  - Shadow mapping
  - Different shader stages (compute, tessellation, etc.)
- ...to multi threaded command buffer generation
- Examples try to concentrate on one single thing
- Around 30 examples with more to come

Vulkan Examples

- Open Source [1]
- MIT license
- C++11 (and some C++14 features)
- Intented as a starting point for Vulkan development
- Tried to comment as much as possible
- If you prefer learning from source rather than tutorials
- Not a framework, abstraction only where necessary
  - Base class
  - Swap chain
- Working on Linux, Android and Windows
  - Running on different vendors (AMD, NVIDIA, Intel*)
  - Support for different compilers (via CMAKE)

[1] https://github.com/SaschaWillems/Vulkan

From OpenGL® to Vulkan™

An explicit journey

Coming from OpenGL

- Steep learning curve
  - Not so steep if you did AZDO or DX12
  - But people still use immediate GL...
- Entirely new API
  - More code to get things done (it's explicit)
  - More responsibility
  - Need to implement stuff OpenGL hides
  - New concepts to learn
- Some features missing yet (e.g. transform feedback)
- Vulkan is not for everyone
- OpenGL won't go away ;)

What do I get?

- A clean and modern new API
  - Same across desktop and mobile
- Better performance! [1]
  - Single threaded (lower driver overhead)
  - Multi threading!
- Graphics AND compute (mandatory)
- SPIR-V (no more GLSL compiler woes)
- Validation layers
- Better platform abstraction
- SDK from LunarG [2]

[1] If done right ;)

[2] http://vulkan.lunarg.com

Window System Integration

- Replacement for the OpenGL render context [1]
- Platform specific surface
  - Enable instance extension (VK_KHR_platform_SURFACE_EXTENSION_NAME)
  - Request with (vkCreateplatformSurfaceKHR())
- Swapchain
  - Decoupled from platform
  - Manages images and memory
  - Usually two images (present and render)
- Adding new platforms only a few lines of code!

[1] But much cleaner

SPIR-V for shaders

- Binary intermediate shader representation
- Vulkan core takes shaders as SPIR-V only
  - No more GLSL shaders [1]
- Source can (and mostly will) be GLSL ("front-end")
  - Use glslangvalidator (SDK) to convert to SPIR-V
  - Will also check shaders against current glsl specs
- No need to compile shaders anymore
  - No more glsl compiler woes amongst different IHVs
  - Faster loading times
- Multiple shader entry points possible [2]

[1] Though e.g. NVIDIA has extension to directly load GLSL

[2] May not be implemented everywhere (yet)

No more state machine

- Global state machine replaced by Pipeline State Objects
  - Forces you to layout your render pipeline upfront
  - More work (and planing) for you
  - Much easier (to optimize) for the driver (=faster)
  - Some states still dynamic (line width, depth bias)
- OpenGL

```
// Anywhere you want
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE/GL_FILL);
```

- Vulkan

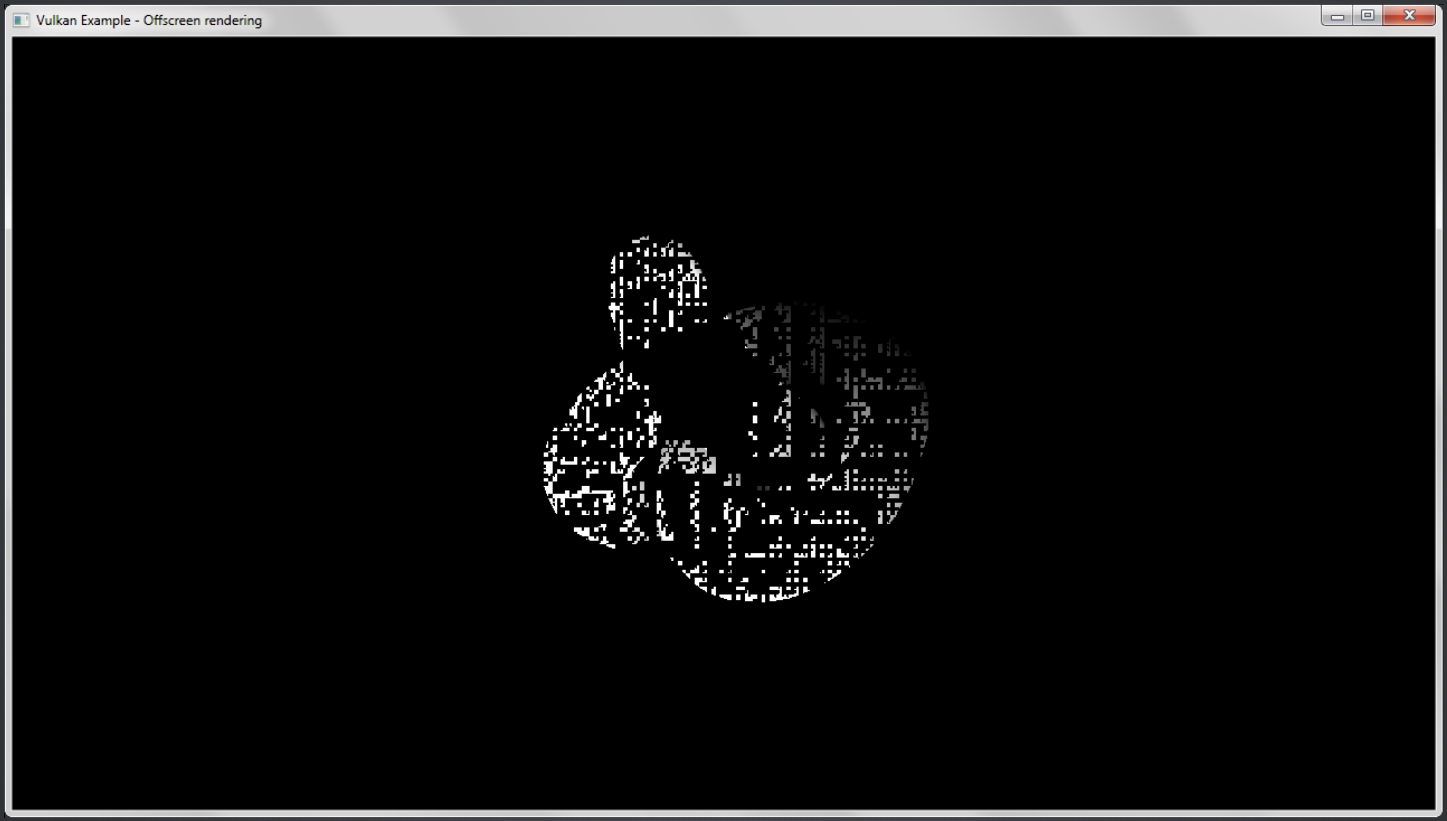
```
// Create two pipelines
vkCreateGraphicsPipelines(...&pipelineCreateInfo, ..., &pipelines.solid);
rasterizationState.polygonMode = VK_POLYGON_MODE_LINE;
vkCreateGraphicsPipelines(...&pipelineCreateInfo, ..., &pipelines.wireFrame);
// Binding depending on user setting
vkBeginCommandBuffer();
vkCmdBindPipeline(...&pipelines.active...);
vkEndCommandBuffer();
// On user toggle : need to rebuild cmd buffer
```

No more state machine

- Describing what (and how) you want to draw upfront
  - Shader attribute bindings (locations and format) part of the pipeline
  - Shader resources bound using descriptor sets
    - No more glUniform*i/u/b/whatever
    - Samplers (and images)
    - Uniform block objects
    - Requires proper descriptor pool setup!
- Render passes
  - Store references to attachments to be rendered to
  - Load and store ops for attachments
  - Can have mulitple sub passes
  - Resolve and preserve attachments (sub passes)
    - E.g. MSAA, MRT

Render passes

Wrong storeOp in depth attachment description



VK_ATTACHMENT_STORE_OP_DONT_CARE instead of VK_ATTACHMENT_STORE_OP_STORE

Resource management

- Your responsibility now!
- Images and buffers (unlike GL)
- Need to manually allocate (and release!) memory
    - Different memory types depending on implementation
    - Need to find memory type index for usage
- Correct usage flags (VK_BUFFER_USAGE_*)
- For images
    - Layout transitions (VK_IMAGE_LAYOUT_*)
    - Crucial for some GPUs (e.g. AMD)
    - Set for all layers / levels (single barrier with range)

Resource management

- Vertex buffer

```
bufferInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT;
vkCreateBuffer(...);
vkGetBufferMemoryRequirements(...);
// Custom function to find appropriate memory type index
// To upload data, you need to find one with host visible bit set
getMemoryType(...VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT...);
vkAllocateMemory(...);
vkMapMemory(...);
// Copy data
memcpy(..);
vkUnmapMemory(...);
```

- Texture with mip maps

```
// Prepare for transfer
imageMemoryBarrier.oldLayout = VK_IMAGE_LAYOUT_PREINITIALIZED;
imageMemoryBarrier.newLayout = VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL;
imageMemoryBarrier.subresourceRange.levelCount = texture.mipLevelCount;
...
vkCmdPipelineBarrier(...&imageMemoryBarrier);

// Copy mip levels from linear image or (better) buffer
for (uint32_t level = 0; level < texture.mipLevels; ++level) {...}

// Prepare for shader usage
imageMemoryBarrier.oldLayout = VK_IMAGE_LAYOUT_TRANSFER_DST_OPTIMAL;
imageMemoryBarrier.newLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
vkCmdPipelineBarrier(...&imageMemoryBarrier);
```

Drawing stuff

- Render commands recorded in command buffers
  - Similar to NV_command_list (OpenGL)
  - Build once, reuse often
    - Can use secondary command buffers
    - Can be created outside of the main thread!
    - E.g. only add secondary buffer if object is visible
- Example :

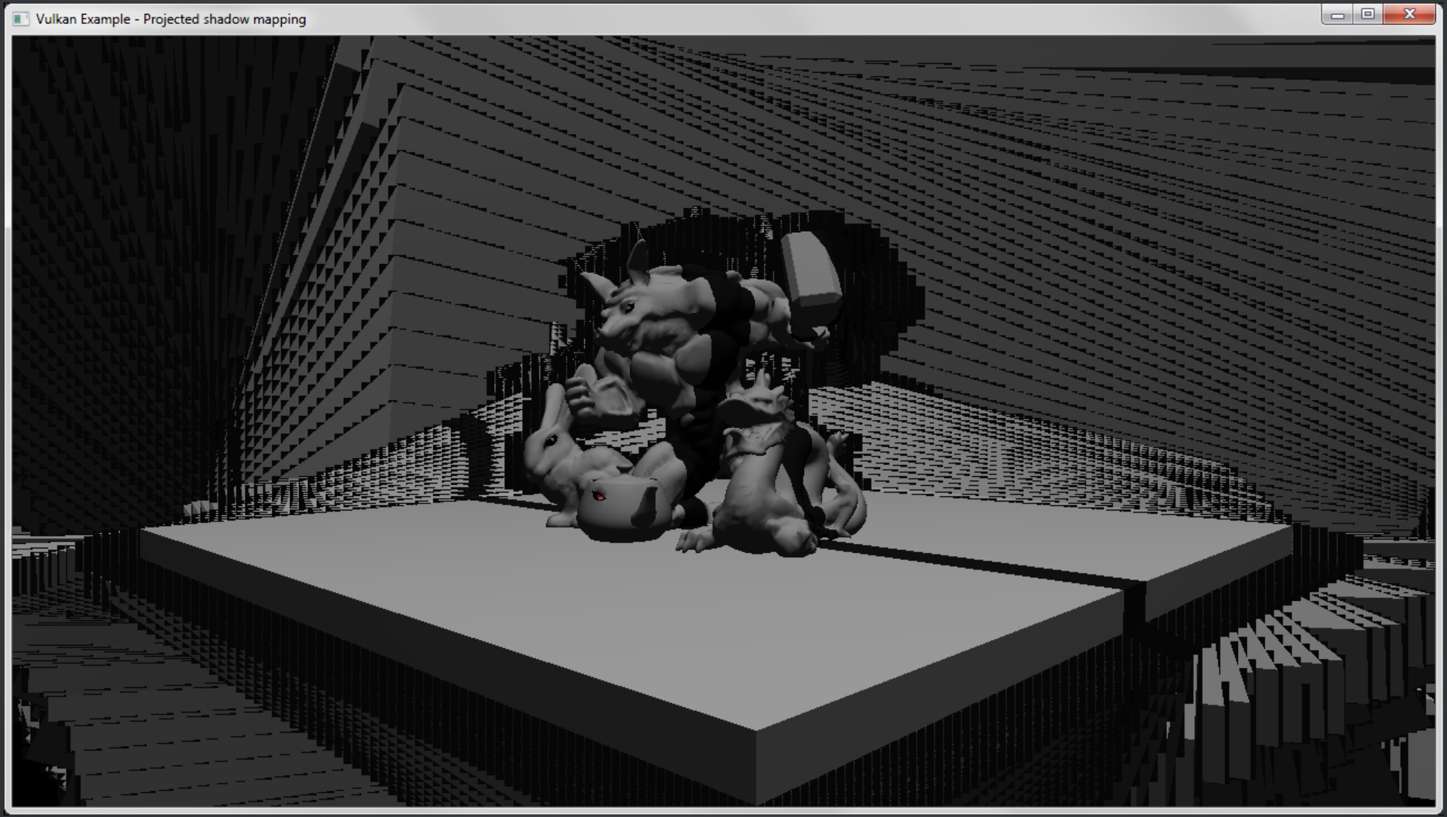
```
vkBeginCommandBuffer(commandBuffer, ...);
vkCmdBeginRenderPass(commandBuffer, ...);
// Dynamic states
vkCmdSetViewport(...);
// Bind descriptor sets (shader attribute binding points)
vkCmdBindDescriptorSets(...pipelineLayout, ... &descriptorSet, ...);
// Bind the rendering pipeline (including the shaders)
vkCmdBindPipeline(...VK_PIPELINE_BIND_POINT_GRAPHICS, pipelines.solid);
// Draw
vkCmdBindVertexBuffers(...);
vkCmdBindIndexBuffer(...);
vkCmdDrawIndexed(...);
vkCmdEndRenderPass(commandBuffer);
...
vkQueueSubmit();
```

Synchronization Objects

- Something rarely used in OpenGL (except compute)
  - Doing them wrong will harm performance
  - Hard to get right
- Fences (heavy!)
  - Synchronize between GPU and CPU (host)
- Barriers and events
  - Synchronize within command buffer
  - E.g. Image layout transitions
- Semaphores
  - Synchronize queue submissions (also across queues)
  - E.g. sync presentation and rendering

Synchronization Objects

Missing post present barrier (strict hardware)



Some tips...

Learned during development

Use staging

- Create buffer with host visibility (transfer source)

```
bufferInfo.usage = VK_BUFFER_USAGE_TRANSFER_SRC_BIT;
...
vkCreateBuffer(...);
vkGetBufferMemoryRequirements(...);
getMemoryType(...VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT...);
vkAllocateMemory(...);
// Map and copy data to buffer
```

- Create device local buffer (transfer dest)

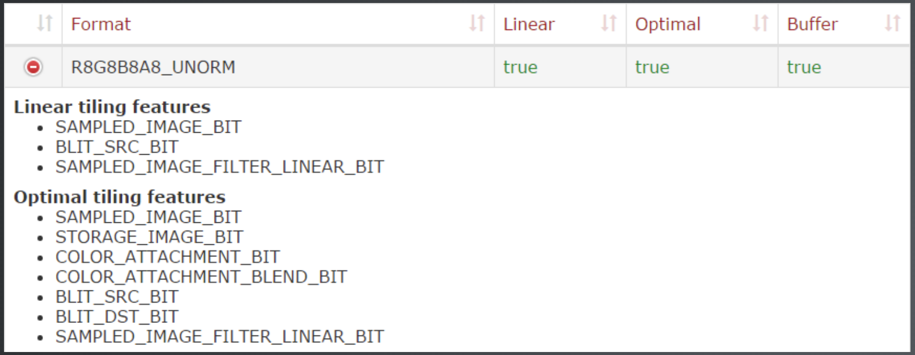
```
bufferInfo.usage = VK_BUFFER_USAGE_VERTEX_BUFFER_BIT | VK_BUFFER_USAGE_TRANSFER_DST_BIT;
// Create an empty buffer with same size as staging buffer
vkCreateBuffer(...);
getMemoryType(...VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT...);
```

- Copy

```
VkBufferCopy copyRegion = {};
copyRegion.size = vertexBufferSize;
...
vkBeginCommandBuffer();
vkCmdCopyBuffer(...©Region);
vkEndCommandBuffer();
// Submit command buffer
// Delete staging buffer
```

Prefer optimal tiling

- For images
- Linear tiling features be very limited



	Format	Linear	Optimal	Buffer
⊖	R8G8B8A8_UNORM	true	true	true

Linear tiling features

- SAMPLED_IMAGE_BIT
- BLIT_SRC_BIT
- SAMPLED_IMAGE_FILTER_LINEAR_BIT

Optimal tiling features

- SAMPLED_IMAGE_BIT
- STORAGE_IMAGE_BIT
- COLOR_ATTACHMENT_BIT
- COLOR_ATTACHMENT_BLEND_BIT
- BLIT_SRC_BIT
- BLIT_DST_BIT
- SAMPLED_IMAGE_FILTER_LINEAR_BIT

- Use staging
  - Check format flags with vkGetPhysicalDeviceFormatProperties
  - Create device local image with optimal tiling

```
imageCreateInfo.tiling = VK_IMAGE_TILING_OPTIMAL;
...
getMemoryType(...VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT...);
```

  - Copy from linear image or (better) buffer

Hardware differences

- Not all GPUs are equal
  - Correct image layout/usage crucial on AMD
  - NVIDIA GPUs ignore image layout
  - Intel (Open Source) also pretty strict
  - Performance on tile based renderers?
- Make sure validation reports no errors
- Store properties of physical device

```
vkGetPhysicalDeviceProperties(physicalDevice, &deviceProperties);
```

  - Limits and features in one place
  - Easier to access than with GL
  - Easy to check at runtime

```
assert(sizeof(pushConstantBlock) <= deviceProps.limits.maxPushConstantsSize);
```

Use the Validation layers

- Save you lots of trouble!
    - No more "why the hell is everythin black"
- Like GL_ARB_debug_output but much better
    - Messages generated by the layers, not the driver
    - Consistent validation across all implementations
- Available if the SDK is installed
    - Layers for memory, threading, images, draw state, etc.
- Performance penalty!
    - Don't enable by default
- Example (Draw State validation layer) :

```
// Missing image memory barrier before first use
ERROR: [DS]Code 6 : Cannot submit cmd buffer using image with layout
VK_IMAGE_LAYOUT_PREINITIALIZED when first use is VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL
```

- Unsure about a validation message?
    - Look (or step) into the layer source

Push Constants

- Small block for easy (and fast) shader data updates
- Spec requires at least 128 bytes (Two 4x4 matrices!)
- Declared in shader

```
layout(push_constant) uniform PushConsts {
        mat4 m;
} pushConsts;
```

Note : Vulkan specific, so use glslangvalidator from SDK to convert!

- Part of the pipeline layout

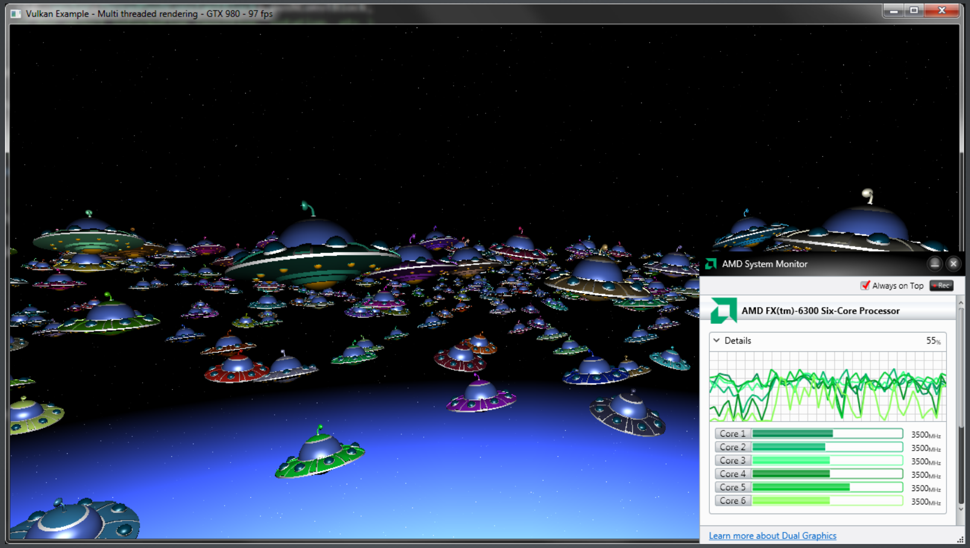
```
VkPushConstantRange pushConstantRange;
pushConstantRange.stageFlags = VK_SHADER_STAGE_VERTEX_BIT;
pushConstantRange.size = sizeof(pushConstantBlock);
pipelineLayoutCreateInfo.pPushConstantRanges = &pushConstantRange;
```

- Update during render pass

```
vkCmdPushConstants(...VK_SHADER_STAGE_VERTEX_BIT...pushConstantBlock.data());
```

- No need to use a uniform block object (and descriptor set)

Multi threading



- Full multi thread support!
- Stream resources, generate render workload
- Great for mobile devices
- More on this by Mathias...

Thanks for listening!

Keep on forging :)