Practical Development for Vulkan

Dan Ginsburg, Valve
Baldur Karlsson, Unity
Dean Sekulic, Croteam
Session Overview

- Vulkan Status Update, Dan Ginsburg
- Vulkan – Care and Feeding, Dean Sekulic
- Debugging with Vulkan Renderdoc, Baldur Karlsson
- Q&A
Session Overview

- Vulkan Status Update, Dan Ginsburg
- Vulkan – Care and Feeding, Dean Sekulic
- Debugging with Vulkan Renderdoc, Baldur Karlsson
- Q&A
Vulkan Status on Desktop

- Vulkan 1.0 has shipped
  - Windows 7, 8, 10
    - NVIDIA – GeForce 600-series+ (Kepler, Maxwell)
    - AMD – HD 7700+ (GCN 1.0, 1.1, 1.2)
    - Intel – Skylake (Beta)
  - Linux
    - NVIDIA – same GPUs as Windows
    - AMD – unreleased
    - Intel OTC – Broadwell, Skylake
Steam Survey Data

DX12 Support

Steam Hardware Survey, Feb 2016
Vulkan Adoption

- Vulkan Steam Overlay complete
- Linux
  - Vulkan Loader included in Steam Linux runtime
  - SteamOS 2.64 – Vulkan NVIDIA
  - Working with Linux distros (Canonical, RedHat) to include Vulkan Loader
- Windows
  - IHV Driver installers including VulkanRT installer
Vulkan Source 2

- Dota 2 running on Vulkan
  - Seeded to all GPU vendors
  - Up on NVIDIA, AMD, and Intel
- Scaleform
  - Autodesk working on Vulkan support
  - Will release as soon as this is integrated
Vulkan Status

- **Wide platform support**
  - Larger share of the desktop than DX12
- **Drivers rolling out quickly**
  - NVIDIA already released WHQL non-beta Vulkan drivers
- **Tools**
  - LunarG SDK
    - Loader
    - Validation Layers
    - vktrace
    - Samples
  - RenderDoc
  - glslang
  - All Open Source on github
HLSSL Translation

- HLSL translation continues to be impediment for developers from DirectX
- Work has begun on HLSL -> SPIR-V
  - Part of glslang project – hlsl-frontend branch
  - Leverages existing SPIR-V code generation
HLSL -> SPIR-V – Get Involved!

- Develop
  - https://github.com/KhronosGroup/glslang/commits/hlsl-frontend
- Discuss
  - https://github.com/KhronosGroup/glslang/issues/200
Session Overview

- Vulkan Status Update, Dan Ginsburg
- Vulkan – Care and Feeding, Dean Sekulic
- Debugging with Vulkan Renderdoc, Baldur Karlsson
- Q&A
VULKAN – care and feeding

Dean Sekulić
graphics programmer
And now...

Conclusion!

• Vulkan IS hard to code for

• Not for everyone
  – small projects or not-CPU bound - go with OpenGL (ES)
    (but lots of objects on screen -> Vulkan!)
  – quick prototyping is faster with OpenGL

• Vulkan is fast and portable! what more do you want?
  – so answer is YES - go for it, we did!
Conclusion (2)

• You can "just port" your engine right away (like we did)
  – will get speedups and less stuttering
  – do proper way later or...

• ... right from the start!
  – change paradigm! forget about state machine - it's so 90's!
  – code gfx part of engine around Vulkan, do a wrapper for OpenGL (ES) and/or Direct3D 11

• And now, onto some problems we stumbled upon along the way...
The Talos Principle GFX design

• We ported The Talos Principle to Vulkan as proof of concept
  – took us 4 men/months
    • at least a month of that time was because of "hitting a moving target“
      (lots of API changes, because of work-in-progress state)

• Our gfx wrapper
  – API agnostic (of course!)
  – all old-fashion functions are inlined!
  – at Draw time, first call
    [D3D9|D3D11|OGL|VLK...]:UpdateStates()
Handling Pipeline state objects

• Quick’n’simple way
  – load/cache/create them per request, do hashing, binning, sorting, fast search
    (we use linear search with more frequently used sorted to begining of array is fast enough)
  – not so optimal, could produce stuttering in frame rate
    • yes, use PSO caches!
  – CreatePipeline is more expensive than vkCreateShader!
    (shader is actually optimized during pipeline creation)

• Don't forget to destroy some when frame-buffer is destroyed
  (to keep count to minimum for faster searches!)
Descriptor sets

• We're currently emulating old bind model
  – several "predefined" layouts

• Do everything like for PSOs (create, hash, bin, sort...)
  (much faster to create them than PSOs, no caching needed here)
  – we update update only once, at creation time
  – later on, just bind

• Don't forget to destroy some when a texture that DS references is destroyed
  (same as for PSOs, to keep count to minimum)
Memory management

- Rewrote memory management code four times! :(
  (personal record for rewrites)
  – critical for performance

- Don't use Vulkan vkAllocateMemory() for every object
  (Vulkan is not designed this way!)

- Have your own memory manager - it's a must!
  – watch out for fragmentation!
    (should have some form of anti-fragmentation system)

- But don't have just one big pool of device memory
  (otherwise, OS will have hard time swapping it, if/when needed!)
  – have several small(er) pools
  – we also have separate pools for linear and optimal allocations!
    (if you forget about VkPhysicalDeviceLimits.bufferImageGranularity, hell awaits!)
Memory management (2)

- Basically, there are 3 types of memory...
- Device, host (mappable) and shared
  - have several pools for each of these types
  - preallocate and do additional allocation as needed
- Keep host memory mapped all the time
  - but be careful with CPU-GPU sync!
- If system (also) has shared memory
  (device mappable; like VkPhysicalDeviceType.VK_PHYSICAL_DEVICE_TYPE_INTEGRATED_GPU)
  forget about all the copies and updates; just keep it mapped all the time and write/copy into that memory directly
- Watch out!
  - there could be some hidden internal memory allocations
    (host allocations exposed, but not driver’s device memory allocations!)
  - using host allocations call-back might hurt performance (so use them only for debugging!)
Uploading resources

• Not simple as with older gfx APIs :(

• Have a staging resource class that has own command buffers, fences and buffers used as source for transfer from host to device (video) memory
  – have one command pool for all that (or one per thread)
  – allocate command buffer when needed for upload, begin, add copy/update commands, submit and free CB when its fence is done
  – don't wait end of frame to submit and/or free those
    (it might fill up your whole host memory when reloading a lot of textures!)
  – submit from other threads, have a spin-lock to avoid calling vkQueueSubmit() concurrently!

• Also reduce number of submits that you have per resource
  – there's a certain amount of “call”-overhead attached to it (regardless of command buffer size!)
  – don't use one staging resource per each mipmap of each face of a texture!
  – use one for all mipmaps and all faces!
Destroying resources

• We're lazy, no fences per resource :)

• We just wait for some frames to pass
  – put them at "delay" array or list
    (some frames > swap-chains_count+1, to be on the safe side)

• But be careful not to overflow memory
  – new resources are uploaded before old have been really destroyed! (the same applies to reusing resources)
Barriers

• Critically important for cross-vendor correctness
  – and you are now required to insert them yourself!

• Read-after-Write access (shadow maps, for example)
  – definitely requires a barrier
    • otherwise you’ll get artifacts, but only sometimes/somewhere (Yikes!)

• if you have too many per frame (say, >20) - batch them!

• strategies
  – change early
    • better performance, might require high-level changes
  – change "on-time"
    • might cause GPU stall
Occlusion culling

- we use OC for high level rendering work decisions
- our pipeline looks like this
  - visibility system (trivial rejection and acceptance) -> distance culling -> frustum culling -> occlusion culling(!)
  - animations -> bones’ transformations -> material modifiers -> render (could be several batches!)
- old API ordering:
  - begin query -> draw -> end query -> swap-buffers -> get query result
- vulkan ordering:
  - reset query? (must be outside of render pass, so this might not be the place for it!) ->
  - begin query -> draw -> end query -> swap-buffers -> get query result -> reset query now (one by one)
- query reset might be executed by GPU too late, after get result in next frame == wrong result!
  - might require extra frame delay!
(cannot reset query outside of render pass, nor directly via CPU!)
- track which queries were tested (begun/end) in a given pool, to avoid infinite waits inside the driver!
Thank you

- special thanks goes to all the great folks at nVidia, AMD, Valve, LunarG, Intel and Baldur (RenderDoc!)

- and Alen (our CTO) who started all this by being really (pro)active on Vulkan Advisory Panel, while I was busy on other fronts

- also all friends and colleagues at Croteam who helped me with this port and gave me courage with their kind words ("You're never gonna finish this", "Vulkan 'till retirement", "Vulkan programmer work is never done", "Drop it while you're young... oh sorry, you're not young anymore"...;)

- and of course, LunarG's Vulkan Validation Layer!
Wasn't enough time for...

- Round-robin buffers
- what if you ran out of memory?
- other queue for copying resources to device memory (haven't experimented with that yet)
- you (usually?) can't have staging linear image for mipmapped (cube-)textures used as source because these are not supported
  - just use vkCmdCopyBufferToImage() and that's it!
- Validation layers
- Tools!
  - RenderDoc is great
  - but we need something that can look inside vkQueueSubmit() - IHVs?!
- Further in time
  - "unwrapping" everthing
    - needs lots of recode in high-level rendering path
  - real MT renderer, not wrapper: command buffers on other threads!
  - precreate PSOs (for each loaded material, i.e. high-level shader)
  - precreate command buffers! (geometry plus material, for whole models)
RenderDoc for Vulkan

Baldur Karlsson (@baldurk)
Brief History

Started as Hobby/spare-time project, mid 2012

Born out of need, other debuggers weren’t working

First public release, early 2014 with source

Initially D3D11 support, then OpenGL, now...
RenderDoc Vulkan support

Began work late 2015

Developed with help from Unity and LunarG

Launched simultaneously with Vulkan 1.0 Spec

Available on Windows bundled in SDK

Linux is coming Real Soon Now™
Initial Capture screen
In-game overlay
Event Browser & API Inspector

Markers from
VK_EXT_debug_marker
Pipeline State
### Pipeline State

#### Resources

<table>
<thead>
<tr>
<th>Set</th>
<th>Binding</th>
<th>Type</th>
<th>Resource</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>2D</td>
<td>Image &amp; Sampler</td>
<td>Texture2D 83319</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2D</td>
<td>Image &amp; Sampler</td>
<td>Texture2D 83303</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2D</td>
<td>Image &amp; Sampler</td>
<td>Texture3D 83275</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2D</td>
<td>Image &amp; Sampler</td>
<td>Sampler 83304</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2D</td>
<td>Image &amp; Sampler</td>
<td>Texture2D 83309</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2D</td>
<td>Image &amp; Sampler</td>
<td>Texture3D 83275</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2D</td>
<td>Image &amp; Sampler</td>
<td>Sampler 20200</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>2D</td>
<td>Image &amp; Sampler</td>
<td>wellknown_white_1x1.vtex</td>
</tr>
</tbody>
</table>

#### Uniform Buffers

<table>
<thead>
<tr>
<th>Set</th>
<th>Binding</th>
<th>Buffer</th>
<th>Byte Range</th>
<th>Size</th>
<th>Go</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
<td>ViewConstant</td>
<td>16384 - 122356</td>
<td>39 Variables, 672 bytes</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>33</td>
<td>_clouds</td>
<td>22694 - 122304</td>
<td>31 Variables, 256 bytes</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>DirectXParams_1</td>
<td>129280 - 122972</td>
<td>23 Variables, 512 bytes</td>
<td></td>
</tr>
</tbody>
</table>
### Mesh View

<table>
<thead>
<tr>
<th>VTX</th>
<th>IDX</th>
<th>VS_INPUT.gl_v.TangentUOs</th>
<th>VS_INPUT.gl_vTexCoord</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>434</td>
<td>1.00</td>
<td>0.02687 0.604</td>
</tr>
<tr>
<td>1</td>
<td>435</td>
<td>1.00</td>
<td>0.03111 0.61572 0</td>
</tr>
<tr>
<td>2</td>
<td>409</td>
<td>1.00</td>
<td>0.04087 0.60547 0</td>
</tr>
<tr>
<td>3</td>
<td>409</td>
<td>1.00</td>
<td>0.04087 0.60547 0</td>
</tr>
<tr>
<td>4</td>
<td>408</td>
<td>1.00</td>
<td>0.03702 0.59424 0</td>
</tr>
<tr>
<td>5</td>
<td>433</td>
<td>1.00</td>
<td>0.02687 0.604</td>
</tr>
<tr>
<td>6</td>
<td>408</td>
<td>1.00</td>
<td>0.03702 0.59424 0</td>
</tr>
<tr>
<td>7</td>
<td>410</td>
<td>1.00</td>
<td>0.01665 0.5752</td>
</tr>
<tr>
<td>8</td>
<td>433</td>
<td>1.00</td>
<td>0.02687 0.604</td>
</tr>
<tr>
<td>9</td>
<td>409</td>
<td>1.00</td>
<td>0.04087 0.60547 0</td>
</tr>
<tr>
<td>10</td>
<td>407</td>
<td>1.00</td>
<td>0.05991 0.56936 0</td>
</tr>
<tr>
<td>11</td>
<td>406</td>
<td>1.00</td>
<td>0.03702 0.59424 0</td>
</tr>
<tr>
<td>12</td>
<td>408</td>
<td>1.00</td>
<td>0.03702 0.59424 0</td>
</tr>
<tr>
<td>13</td>
<td>403</td>
<td>1.00</td>
<td>0.01677 0.56006 0</td>
</tr>
<tr>
<td>14</td>
<td>410</td>
<td>1.00</td>
<td>0.01665 0.5752</td>
</tr>
<tr>
<td>15</td>
<td>411</td>
<td>1.00</td>
<td>0.01665 0.5752</td>
</tr>
<tr>
<td>16</td>
<td>407</td>
<td>1.00</td>
<td>0.01205 0.56494 0</td>
</tr>
<tr>
<td>17</td>
<td>406</td>
<td>1.00</td>
<td>0.05722 0.57861 0</td>
</tr>
<tr>
<td>18</td>
<td>403</td>
<td>1.00</td>
<td>0.03702 0.59424 0</td>
</tr>
<tr>
<td>19</td>
<td>410</td>
<td>1.00</td>
<td>0.01677 0.56006 0</td>
</tr>
<tr>
<td>20</td>
<td>408</td>
<td>1.00</td>
<td>0.05991 0.56936 0</td>
</tr>
<tr>
<td>21</td>
<td>407</td>
<td>1.00</td>
<td>0.05991 0.56936 0</td>
</tr>
</tbody>
</table>
### Mesh View

The image displays a software interface for a mesh view editor. The interface includes a table with columns for VTX, IDX, `gl_PerVertex.gl_Position`, and two numerical columns. The table shows coordinates for various vertices, with values ranging from negative to positive numbers. The mesh viewer on the right side of the interface shows a 3D model with a checkerboard background, indicating a perspective view of the mesh.

#### Table Preview

<table>
<thead>
<tr>
<th>VTX</th>
<th>IDX</th>
<th>gl_PerVertex.gl_Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>434</td>
<td>8.45487 -69.08381 181.3914 184.6658</td>
</tr>
<tr>
<td>1</td>
<td>435</td>
<td>0.61601 -63.81541 102.1429 185.4143</td>
</tr>
<tr>
<td>2</td>
<td>409</td>
<td>7.15961 -68.56622 182.8131 186.0019</td>
</tr>
<tr>
<td>3</td>
<td>489</td>
<td>7.15961 -68.56622 182.8131 186.0019</td>
</tr>
<tr>
<td>4</td>
<td>408</td>
<td>13.81096 -73.39017 182.607 185.3888</td>
</tr>
<tr>
<td>5</td>
<td>433</td>
<td>8.45487 -69.08381 181.3914 184.6658</td>
</tr>
<tr>
<td>6</td>
<td>408</td>
<td>13.81096 -73.39017 182.607 185.3888</td>
</tr>
<tr>
<td>7</td>
<td>410</td>
<td>0.81311 -73.87152 179.2874 182.5762</td>
</tr>
<tr>
<td>8</td>
<td>433</td>
<td>8.45487 -69.08381 181.3914 184.6658</td>
</tr>
<tr>
<td>9</td>
<td>409</td>
<td>7.15961 -68.56622 182.8131 186.0019</td>
</tr>
<tr>
<td>10</td>
<td>407</td>
<td>20.26167 -81.75376 164.1182 187.3617</td>
</tr>
<tr>
<td>11</td>
<td>408</td>
<td>13.81096 -73.39017 182.607 185.3888</td>
</tr>
<tr>
<td>12</td>
<td>408</td>
<td>13.81096 -73.39017 182.607 185.3888</td>
</tr>
<tr>
<td>13</td>
<td>403</td>
<td>-2.28547 -80.29852 179.492 182.774</td>
</tr>
<tr>
<td>14</td>
<td>410</td>
<td>0.81311 -73.87152 179.2874 182.5762</td>
</tr>
<tr>
<td>15</td>
<td>410</td>
<td>0.81311 -73.87152 179.2874 182.5762</td>
</tr>
<tr>
<td>16</td>
<td>403</td>
<td>-2.28547 -80.29852 179.492 182.774</td>
</tr>
<tr>
<td>17</td>
<td>411</td>
<td>-8.6893 -76.82861 178.7941 182.0789</td>
</tr>
<tr>
<td>18</td>
<td>407</td>
<td>20.26167 -81.75376 164.1182 187.3617</td>
</tr>
<tr>
<td>19</td>
<td>406</td>
<td>25.00259 -85.25342 183.1219 186.5894</td>
</tr>
<tr>
<td>20</td>
<td>408</td>
<td>13.81096 -73.39017 182.607 185.3888</td>
</tr>
<tr>
<td>21</td>
<td>407</td>
<td>20.26167 -81.75376 164.1182 187.3617</td>
</tr>
</tbody>
</table>
But wait, there’s more!

Timeline bar showing intraframe resource reads and writes

Drawcall microsecond timings

Texture/buffer export to file (.dds/.exr)

Python Shell with access to all underlying data
Conclusion

RenderDoc is available and working now - today

May even already be installed if you have Windows Vulkan SDK

Open source on github - MIT license

Active development & improvement all the time

Talk to me! - I will help fix issues, improve workflows, add features. Always happy to talk
Thank you!

baldurk@baldurk.org
@baldurk
https://github.com/baldurk/renderdoc
Appendix: Coherent persistent maps!

Keeping pointers to mapped memory in Vulkan can be very useful.

Be wary of doing this for coherent memory types when debugging.

Requires a very slow memcmp() on each frame’s submission to detect changes.

Instead either prefer non-coherent types, or call vkFlushMapped... anyway to denote modified regions to the debugger.

Only do this while debugging though!