

## 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

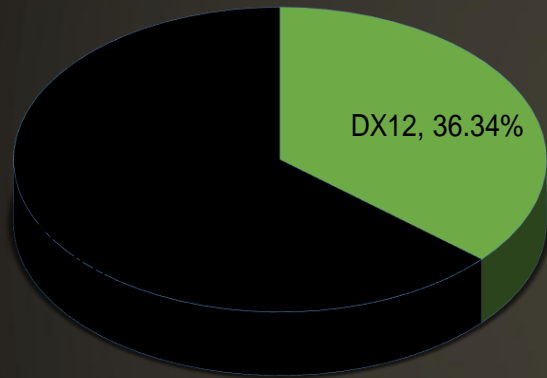
- Vulkan Status Update, Dan Ginsburg
- Vulkan – Care and Feeding, Dean Sekulic
- Debugging with Vulkan Renderdoc, Baldur Karlsson
- Q&A

# Vulkan Status on Desktop

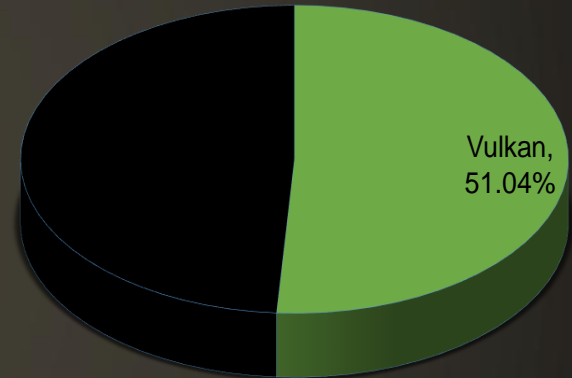
- Vulkan 1.0 has shipped
  - Windows 7, 8, 10
    - NVIDIA – GeForce 600-series+ (Kepler, Maxwell)
    - AMD – HD 7700+ (GCN 1.0, 1.1, 1.2)
    - Intel – Skylake (Beta)
  - Linux
    - NVIDIA – same GPUs as Windows
    - AMD – unreleased
    - Intel OTC – Broadwell, Skylake

# Steam Survey Data

## DX12 Support



## Vulkan 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

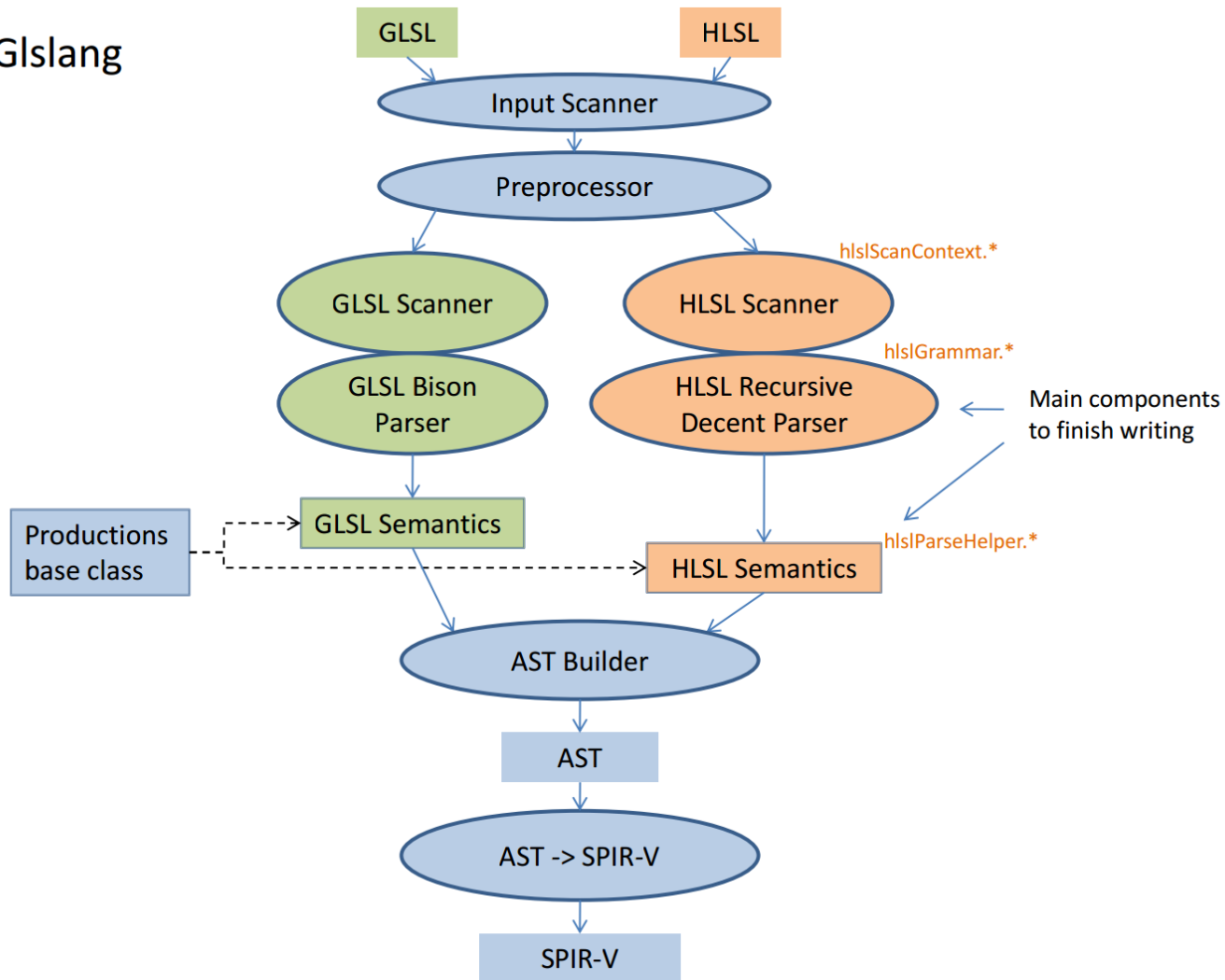


# HLSL Translation

- 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

Glslang



# 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 beginning 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" everything
    - 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

Pipeline State   Mesh Output   Texture Viewer   **Capture Executable**   ▾ ×

Program

Executable Path  ...

Working Directory  ...

Command-line Arguments

Capture Options

Allow Fullscreen    Allow VSync    Seconds Delay    Collect Callstacks    Only Drawcall stacks    Create Debug Device

Hook Into Children    Save All Initials    Ref All Resources    Capture All Cmd Lists    Verify Map() Writes    Auto start

Actions

Queue Capture of Frame  

Save   Load   Capture   Close

# In-game overlay

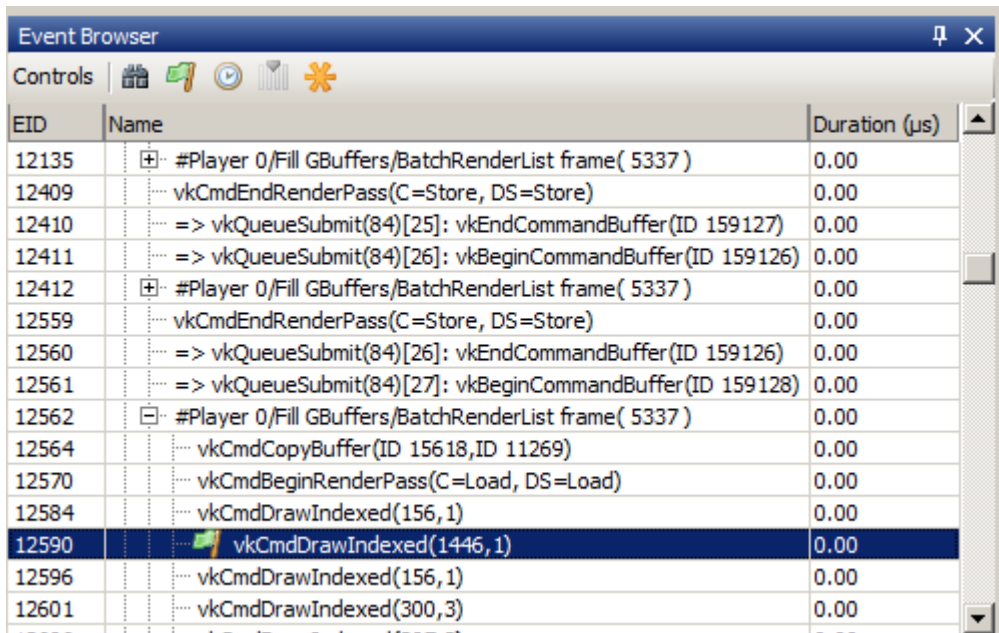
Vulkan. F12, PrtScrn to capture. Frame: 5256. 83.25 ms (73.03 .. 116.04) (12. FPS)  
0 Captures saved.

fps: 12 ping: 0ms loss: 0%

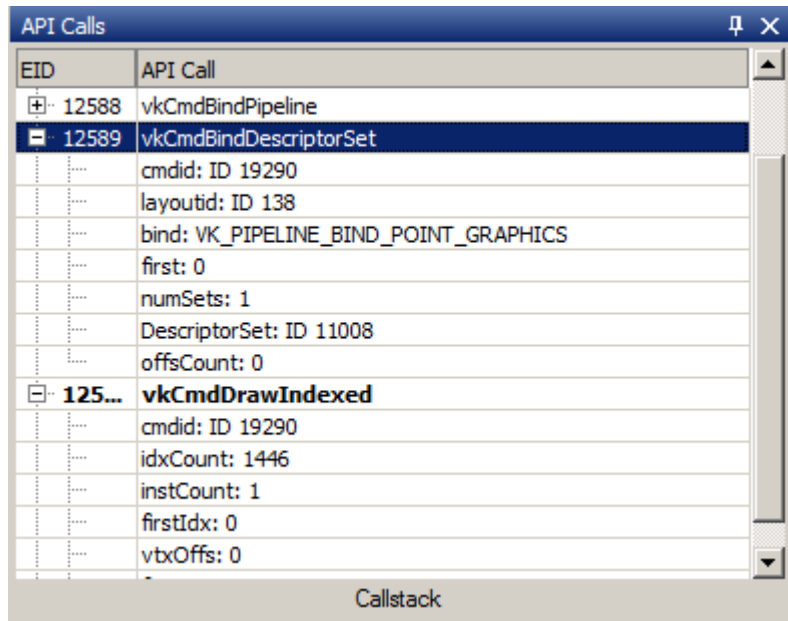
- Demo Options
- HERO
- Refresh
- Free Spells
- Invulnerability
- Level Up
- Level to Max
- SPAWN
- Enemy
- Level Up Enemy
- Remove Spawns
- CUSTOMIZE
- Equip New Items
- Select New Hero
- Pause
- QUIT



# Event Browser & API Inspector



EID	Name	Duration (µs)
12135	#Player 0/Fill GBuffers/BatchRenderList frame( 5337)	0.00
12409	vkCmdEndRenderPass(C=Store, DS=Store)	0.00
12410	=> vkQueueSubmit(84)[25]: vkEndCommandBuffer(ID 159127)	0.00
12411	=> vkQueueSubmit(84)[26]: vkBeginCommandBuffer(ID 159126)	0.00
12412	#Player 0/Fill GBuffers/BatchRenderList frame( 5337)	0.00
12559	vkCmdEndRenderPass(C=Store, DS=Store)	0.00
12560	=> vkQueueSubmit(84)[26]: vkEndCommandBuffer(ID 159126)	0.00
12561	=> vkQueueSubmit(84)[27]: vkBeginCommandBuffer(ID 159128)	0.00
12562	#Player 0/Fill GBuffers/BatchRenderList frame( 5337)	0.00
12564	vkCmdCopyBuffer(ID 15618, ID 11269)	0.00
12570	vkCmdBeginRenderPass(C=Load, DS=Load)	0.00
12584	vkCmdDrawIndexed(156, 1)	0.00
12590	vkCmdDrawIndexed(1446, 1)	0.00
12596	vkCmdDrawIndexed(156, 1)	0.00
12601	vkCmdDrawIndexed(300, 3)	0.00



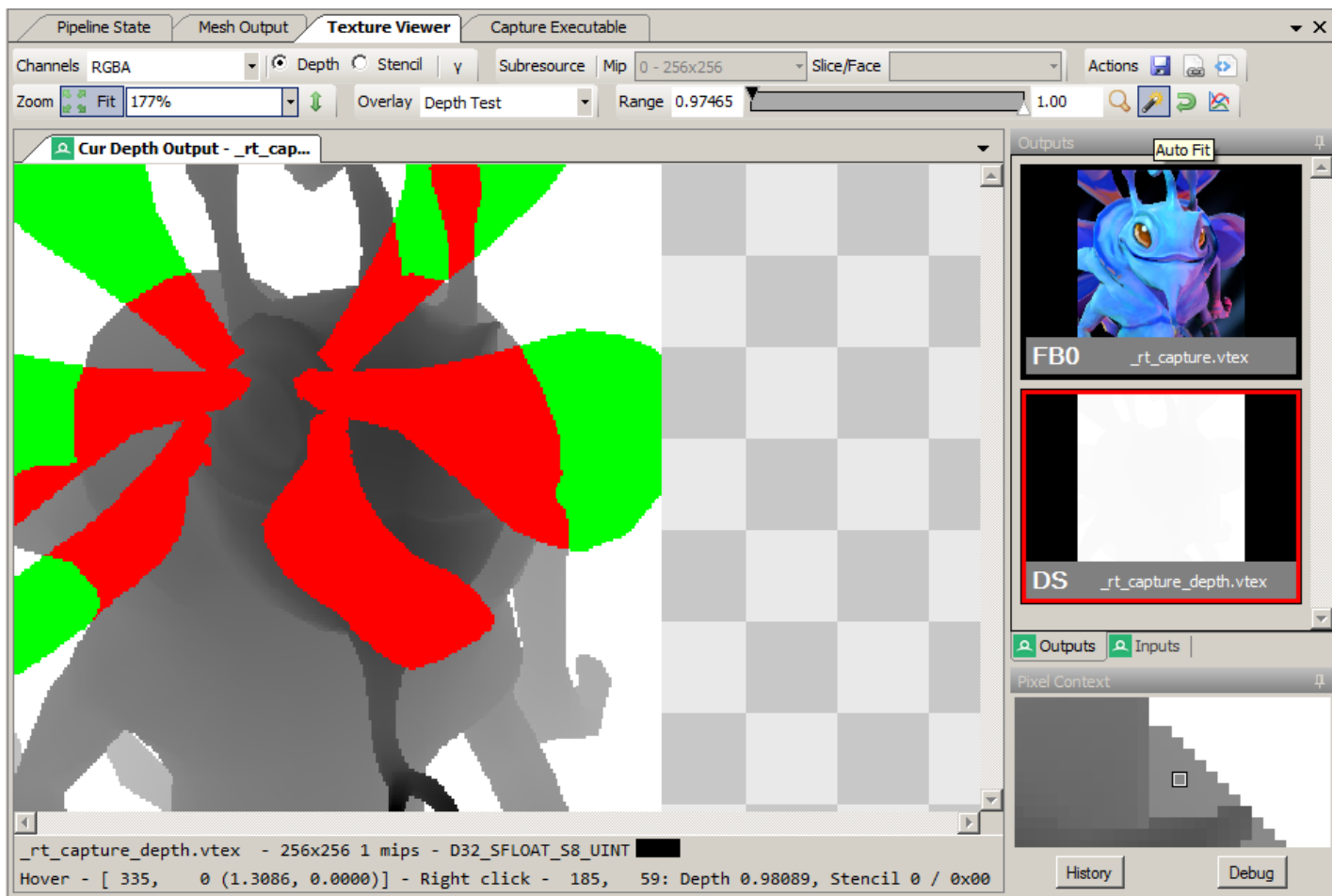
EID	API Call
12588	vkCmdBindPipeline
12589	vkCmdBindDescriptorSet
...	cmdid: ID 19290
...	layoutid: ID 138
...	bind: VK_PIPELINE_BIND_POINT_GRAPHICS
...	first: 0
...	numSets: 1
...	DescriptorSet: ID 11008
...	offsCount: 0
125...	vkCmdDrawIndexed
...	cmdid: ID 19290
...	idxCount: 1446
...	instCount: 1
...	firstIdx: 0
...	vtxOffs: 0

Callstack

Markers from  
VK\_EXT\_debug\_marker



# Texture Viewer



# Pipeline State

Pipeline State | Mesh Output | Texture Viewer | Capture Executable

Display Controls | Show Disabled Items | Show Empty Items | Export

VTX → VS → TCS → TES → GS → Rasterizer → FS → FB → CS

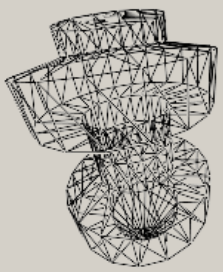
Attributes

Index	Name	Location	Binding	Format	Offset	Go
0	VS_INPUT_gl_vPositionOs	0	0	R32G32B32_SFLOAT	0	⇨
1	VS_INPUT_gl_vNormalOs	1	0	R8G8B8A8_UNORM	12	⇨
2	VS_INPUT_gl_vTangentUOs_fTangentVSign	2	2	R32G32B32A32_SFLOAT	0	⇨
3	VS_INPUT_gl_vTexCoord	3	0	R16G16_SFLOAT	16	⇨
4	VS_INPUT_gl_vBlendIndices	4	0	R8G8B8A8_UINT	20	⇨
5	VS_INPUT_gl_vBlendWeight	5	0	R8G8B8A8_UNORM	24	⇨
6	VS_INPUT_gl_vTransformTextureUV	6	1	R16G16_SFLOAT	0	⇨

Buffers


Slot	Buffer	Rate	Offset	Stride	Byte Len	Go
Index	StaticPool_IB	Index	7494144	2	8388608	⇨
0	StaticPool_VB	Vertex	2425120	28	8388608	⇨
1	StaticPool_VB	Instance	13892	4	8388608	⇨
2	StaticPool_VB	Instance	0	0	8388608	⇨

Mesh View



Primitive Topology

TriangleList



# Pipeline State

**Pipeline State** Mesh Output Texture Viewer Capture Executable

Display Controls Show Disabled Items Show Empty Items Export

VTX → VS → TCS → TES → GS → RS → **FS** → FB → CS

Shader: hero.vfx\_ps

Resources

Set	Binding	Type	Resource	Contents	
0	3: g_tMasks2	2D Image&Sampler	Texture2D 83319	512x256	Bi
0	4: g_tColor	2D Image&Sampler	Texture2D 83303	512x256	Bi
0	5: g_tFresnelColorW...	3D Image&Sampler	Texture3D 83275	32x32x4	R
		Sampler	Sampler 85304	UVW: CLAMP_EDGE	M
0	6: g_tShadowDepth...	2D Image&Sampler	dotasunlightshadowdepthtext...	1024x1024	D
		Sampler	Sampler 26200	UV: CLAMP_BORDER, W: WRAP int(1...	M
0	7: g_tClouds	2D Image&Sampler	wellknown_white_1x1.vtex	1x1	R
		Sampler	Sampler 33101	UVW: WRAP	M

Uniform Buffers

Set	Binding	Buffer	Byte Range	Size	Go
0	32: PerViewConstant...	StaticPool_CB	16384 - 17056	39 Variables, 672 bytes	→
0	33: _Globals_	StaticPool_CB	226048 - 226304	31 Variables, 256 bytes	→
0	34: DotaGlobalParam...	StaticPool_CB	129280 - 129792	23 Variables, 512 bytes	→

**Fragment UBO 0 [0]** StaticPool\_CB

Name	Value	Type
g_matWorldToProjec...	{ {1.54894, 0.10759, -0.96566, ...	flo...
g_matProjectionToW...	{ {0.04816, 0.16962, 1.31348E-0...	flo...
g_matWorldToView	{ {0.27312, 0.01897, 0.96179, 0...	flo...
g_matViewToProjection	{ {5.67128, 0.00, 0.00, 0.00}, ...	flo...
g_vInvProjRow3	0.00, 0.00, -0.249, 0.25	float4
g_vClipPlane0	0.00, 0.00, 0.00, 0.00	float4
g_flToneMapScalarL...	1.00	float
g_flLightMapScalar	1.00	float
g_flEnvMapScalar	1.00	float
g_flToneMapScalarG...	1.00	float
g_vCameraPositionWs	176.71, -45.00, 135.41	float3
g_flViewportMinZ	0.00	float
g_vCameraDirWs	-0.96179, 0.27307, 0.01972	float3
g_flViewportMaxZ	1.00	float
g_vCameraUpDirWs	0.01897, -0.00539, 0.99981	float3
g_flTime	190.2649	float
g_vDepthPsToVsConv...	4000.00, 996.00, -1000.00	float3
g_flNearPlane	4.00	float
g_flFarPlane	1000.00	float
g_flLightBinnerFar...	4096.00	float
g_vInvViewportSize	0.00391, 0.00391	float2
g_vViewportToGBuff...	1.00, 1.00	float2
g_vMorphTextureAtl...	1024.00, 512.00	float2
g_vInvGBufferSize	0.00391, 0.00391, 0.00098, 0.00391	float4
g_vViewportOffset	0.00, 0.00	float2
g_vViewportSize	256.00, 256.00	float2
g_vRenderTargetSize	256.00, 256.00	float2
g_vFogColor	0.00, 0.00, 0.00	float3
g_flNegFogStartOve...	NaN	float
g_flInvFogRange	Infinity	float
g_flFogMaxDensity	0.00	float
g_flFogExponent	0.00	float
g_flMod2xIdentity	0.50	float
g_bRoughnessParams	1.00, 1.00	float2

# Shader View

The screenshot shows a software interface for viewing shader disassembly. The window title is "hero.vfx\_ps". The "Disassembly" tab is active, displaying the following code:

```
148 float3* H_qw6t8a1 = GLSL.std.450::Normalize({104} + {96} + {97});
149 float3 {132} = GLSL.std.450::Normalize(float3(PS_INPUT_gl_vPositionWs_flLinearDepth.xyz) - PerViewConstantBuffer_t_120.g_vCam
150 float3* H_t3ywhi1 = 0.0f.xxx - {132};
151 float {142} = GLSL.std.450::FClamp(Dot(H_t3ywhi1, H_qw6t8a1), 0.0f, 1.0f);
152 float4* vMasks = ImageSampleImplicitLod(g_tMasks1, float2(PS_INPUT_gl_vBaseTexCoord_vDetailTexCoord.xy));
153 float* flMetalnessMask = GLSL.std.450::FMax(vMasks.z, Globals_161.g_flMetalnessBlendToFull);
154 float {187} = GLSL.std.450::FMax(vFresnelTerms.z, flMetalnessMask);
155 float4* vMasks1 = ImageSampleImplicitLod(g_tMasks2, float2(PS_INPUT_gl_vBaseTexCoord_vDetailTexCoord.xy));
156 float {204} = GLSL.std.450::FMax(vMasks1.x, Globals_161.g_flSpecularBlendToFull);
157 float {214} = GLSL.std.450::FMax(vMasks1.y, Globals_161.g_flEnvMapBlendToFull);
158 float4* vColorTexel = ImageSampleImplicitLod(g_tColor, float2(PS_INPUT_gl_vBaseTexCoord_vDetailTexCoord.xy));
159 float4 {253} = ImageSampleImplicitLod(g_tFresnelColorWarp3D, GLSL.std.450::Pow(float3(vColorTexel.xyz), 0.4545f.xxx));
160 float {255} = vFresnelTerms.y * vMasks.y;
161 float3* vAlbedo1 = float3(PS_INPUT_gl_vVertexColor.xyz) * GLSL.std.450::FMix(float3(vColorTexel.xyz), float3({253}.xyz), floa
162 if(vColorTexel.w < Globals_161.g_flAlphaTestReference) {
163     Kill();
164 }
165 float {313} = PS_INPUT_gl_vPositionWs_flLinearDepth.x * DotaGlobalParams_t_289.g_matShadowWorldToTexture[0].x + PS_INPUT_gl_v
166 float {317} = {313} + PS_INPUT_gl_vPositionWs_flLinearDepth.z * DotaGlobalParams_t_289.g_matShadowWorldToTexture[2].x;
167 float {345} = PS_INPUT_gl_vPositionWs_flLinearDepth.x * DotaGlobalParams_t_289.g_matShadowWorldToTexture[0].y + PS_INPUT_gl_v
168 float {349} = {345} + PS_INPUT_gl_vPositionWs_flLinearDepth.z * DotaGlobalParams_t_289.g_matShadowWorldToTexture[2].y;
169 float {376} = PS_INPUT_gl_vPositionWs_flLinearDepth.x * DotaGlobalParams_t_289.g_matShadowWorldToTexture[0].z + PS_INPUT_gl_v
```

Below the disassembly, there are two signature tables:

**Input Signature**

Name	I..	Reg	Type	SysValue	Mask	Used
PS_INPUT_gl_vBaseTexCoord...		0	float4	None	RGBA	RGBA
PS_INPUT_gl_vNormalWs		1	float3	None	RGB	RGB
PS_INPUT_gl_vTangentJWs_f...		2	float4	None	RGBA	RGBA
PS_INPUT_gl_vVertexColor		3	float4	None	RGBA	RGBA
PS_INPUT_gl_vPositionWs_fl...		4	float4	None	RGBA	RGBA
PS_INPUT_gl_vRimColor_fFog		5	float4	None	RGBA	RGBA
PS_INPUT_gl_vLightAtten		6	float4	None	RGBA	RGBA

**Output Signature**

Name	Index	Reg	Type	SysValue	Mask	Unused
PS_OUTPUT_gl_vDiffuse		0	float4	None	RGBA	RGBA
PS_OUTPUT_gl_fDepth		1	float4	None	RGBA	RGBA

# Mesh View

Pipeline State **Mesh Output** Texture Viewer Capture Executable

Controls Sync Views Highlight Vertices Row Offset 0 Instance 0

VS Input	VS Output	GS/DS Output	Preview		
VTX	IDX	VS_INPUT_gl_vTangentUOs...	VS_INPUT_gl_vTexCoord	V...	
0	434	00	1.00	0.02687 0.604	0
1	435	00	1.00	0.03111 0.61572	0
2	409	00	1.00	0.04007 0.60547	0
3	409	00	1.00	0.04007 0.60547	0
4	408	00	1.00	0.03702 0.59424	0
5	433	00	1.00	0.02687 0.604	0
6	408	00	1.00	0.03702 0.59424	0
7	410	00	1.00	0.01666 0.5752	0
8	433	00	1.00	0.02687 0.604	0
9	409	00	1.00	0.04007 0.60547	0
10	407	00	1.00	0.05991 0.58936	0
11	408	00	1.00	0.03702 0.59424	0
12	408	00	1.00	0.03702 0.59424	0
13	403	00	1.00	0.01677 0.56006	0
14	410	00	1.00	0.01666 0.5752	0
15	410	00	1.00	0.01666 0.5752	0
16	403	00	1.00	0.01677 0.56006	0
17	411	00	1.00	0.01205 0.56494	0
18	407	00	1.00	0.05991 0.58936	0
19	406	00	1.00	0.05722 0.57861	0
20	408	00	1.00	0.03702 0.59424	0
21	407	00	1.00	0.05991 0.58936	0

VS Input VS Output GS/DS Output

Arcball Only this draw Solid Shading Secondary Wi

# Mesh View

Pipeline State **Mesh Output** Texture Viewer Capture Executable

Controls Sync Views Highlight Vertices Row Offset 0 Instance 0

VS Input VS Output GS/DS Output

VTX	IDX	gl_PerVertex.gl_Position			
0	434	8.45487	-66.90381	181.3914	184.6658
1	435	0.61601	-63.81451	182.1429	185.4143
2	409	7.15961	-68.56622	182.8131	186.0819
3	409	7.15961	-68.56622	182.8131	186.0819
4	408	13.01096	-73.36017	182.037	185.3088
5	433	8.45487	-66.90381	181.3914	184.6658
6	408	13.01096	-73.36017	182.037	185.3088
7	410	0.81311	-73.87152	179.2874	182.5702
8	433	8.45487	-66.90381	181.3914	184.6658
9	409	7.15961	-68.56622	182.8131	186.0819
10	407	20.26167	-81.75378	184.1182	187.3817
11	408	13.01096	-73.36017	182.037	185.3088
12	408	13.01096	-73.36017	182.037	185.3088
13	403	-2.28547	-80.29852	179.492	182.774
14	410	0.81311	-73.87152	179.2874	182.5702
15	410	0.81311	-73.87152	179.2874	182.5702
16	403	-2.28547	-80.29852	179.492	182.774
17	411	-8.6893	-76.82861	178.7941	182.0789
18	407	20.26167	-81.75378	184.1182	187.3817
19	406	25.02759	-85.25342	183.1219	186.3894
20	408	13.01096	-73.36017	182.037	185.3088
21	407	20.26167	-81.75378	184.1182	187.3817

Preview

VS Input VS Output GS/DS Output

WASD Show whole pass Solid Shading Secondary Wi

# 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](mailto: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!