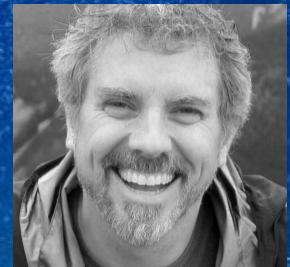


Source-level Shader Debugging in Vulkan with RenderDoc

Greg Fischer
LunarG, Inc.



Presented at the Khronos Vulkanised 2023 Conference

LUNAR G

Who is this guy?

- glslang moderator, developer, maintainer
- spirv-opt developer, maintainer
- dxc developer, maintainer(?)
- GPU-Assisted Validation developer, maintainer
- DebugPrintf developer, maintainer

Shader source debugging in Vulkan is here!

The screenshot shows a debugger interface for Vulkan shader source code. The main window displays the shader code for 'deferred.frag'.

```
90     fragcolor *= shadowFactor;
91 }
92 return fragcolor;
93 }
94
95 float4 main([[vk::location(0)]] float2 inUV : TEXCOORD0) : SV_TARGET
96 {
    // Get G-Buffer values
    float3 fragPos = textureposition.Sample(samplerposition, inUV).rgb;
99    float3 normal = textureNormal.Sample(samplerNormal, inUV).rgb;
100   float4 albedo = textureAlbedo.Sample(samplerAlbedo, inUV);
101
102  float3 fragcolor;
103
104  // Debug display
105  if (ubo.displayDebugTarget > 0) {
106      switch (ubo.displayDebugTarget) {
107          case 1:
108              fragcolor.rgb = shadow(float3(1.0, 1.0, 1.0), fragPos);
109              break;
110          case 2:
111              fragcolor.rgb = fragPos;
112              break;
113          case 3:
114              fragcolor.rgb = normal;
115              break;
116          case 4:
117              fragcolor.rgb = albedo.rgb;
118              break;
119          case 5:
120              fragcolor.rgb = albedo.aaa;
121              break;
122      }
123  }
124
125  return float4(fragcolor, 1.0);
```

Below the code editor are three tables:

Accessed Resources		
Name	Register(s)	Type
textureposition	_6	Resource 2D Color Attachment 334
samplerposition	_7	Sampler Sampler 350
textureNormal	_8	Resource 2D Color Attachment 338
samplerNormal	_9	Sampler Sampler 350
textureAlbedo	_10	Resource 2D Color Attachment 342
samplerAlbedo	_11	Sampler Sampler 350
textureShadowMap	_12	Resource 2D Depth/Stencil Attachment 353

Watch		
Name	Register(s)	Type
fragPos	_414.xyz	float3
inUV	_401.xy	float2

Variable Values		
Name	Register(s)	Type
fragPos	_414.xyz	float3
inUV	_401.xy	float2

High-level Variables		
Name	Register(s)	Type
main		

On the right, there is a 'Callstack' pane showing the current call stack entry: 'main'.

Agenda

- What is RenderDoc?
- What is NonSemantic.Shader.DebugInfo.100
- How to generate source-level debug information
- DebugInfo Instructions
- Status of compiler toolchain support
- Demo: How to debug shaders in RenderDoc

What is RenderDoc?

- Graphics Debugger for Vulkan and other APIs
- renderdoc.org
- Frame capture and debugging
- Shader debugging
 - Previously only SPIR-V disassembly
 - Now supports GLSL/HLSL source-level debugging
 - Source line stepping
 - Local variable names, values, scope-based display
 - Currently supported: fragment, vertex, compute shaders

NonSemantic.Shader.DebugInfo.100

- What?
 - Authored by Baldur Karlsson from RenderDoc
 - Non-semantic SPIR-V extended instruction set
 - (OpExtInstImport “NonSemantic.Shader.DebugInfo.100”)
 - NonSemantic instructions can be safely ignored by drivers
 - Derived from OpenCL.DebugInfo.100 and DWARF debugging standard
- Why?
 - Information useful to debugging is lost during compilation and optimization
 - Function Inlining
 - Local Variable Store and Load Elimination
 - This extension preserves the lost information
 - Allows source-level debugging of optimized shaders, especially legalized HLSL

NonSemantic.Shader.DebugInfo.100 - Instructions

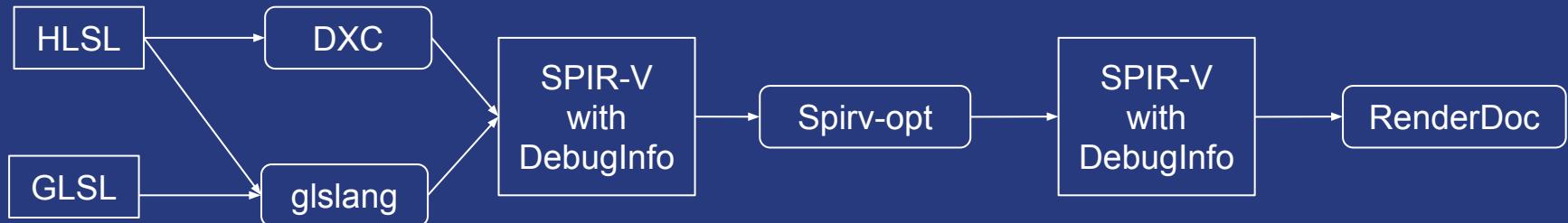
- Location
 - Source: DebugSource
 - Line / Column: DebugLine / DebugNoLine
- Types
 - int / floats / bool: DebugTypeBasic
 - vectors: DebugTypeVector
 - matrix: DebugTypeMatrix
 - arrays: DebugTypeArray
 - structure: DebugTypeComposite / DebugTypeMember
 - functions: DebugTypeFunction
- Variables
 - globals: DebugGlobalVariable
 - locals: DebugLocalVariable, DebugDeclare, DebugValue

NonSemantic.Shader.DebugInfo.100 - Instructions (cont'd)

- Procedures
 - `DebugTypeFunction`, `DebugFunction`
- Scope
 - `DebugScope` / `DebugNoScope`
 - Compilation unit: `DebugCompilationUnit`
 - Function: `DebugFunction`
 - Structure: `DebugTypeComposite`
 - Everything else: `DebugLexicalScope`

NonSemantic.Shader.DebugInfo.100

- How?
 - Annotates locations, scopes, types, variables, values, and procedures
 - DXC and glslang compiler support
 - SPIRV-Tools support
 - Optimization, Validation, Disassembly, Assembly
 - Optimization preserves DebugInfo



Generating Debug Information - DXC

```
\path\to\dxc.exe -spirv -fspv-target-env=vulkan1.3  
    -T <target-profile> -E <entry-point>  
    -fspv-extension=SPV_KHR_non_semantic_info  
    -fspv-debug=vulkan-with-source  
    <hlsl-src-file> -Fo <spirv-bin-file>
```

- `-fspv-debug=vulkan-with-source` instructs the compiler to generate DebugInfo instructions and embed the source string in the DebugSource instruction. RenderDoc reads the high-level source from this instruction.
- `-fspv-extension=SPV_KHR_non_semantic_info` instructs the compiler to use the `SPV_KHR_non_semantic_info` extension which is required to use non semantic extended instruction sets. Not required for Vulkan 1.3.

Generating Debug Information - glslang

HLSL

```
\path\to\glslangValidator.exe -e main -gVS -D -o <spirv-bin-file> <hlsl-src-file>
```

GLSL

```
\path\to\glslangValidator.exe -e main -gVS -V -o <spirv-bin-file> <glsl-src-file>
```

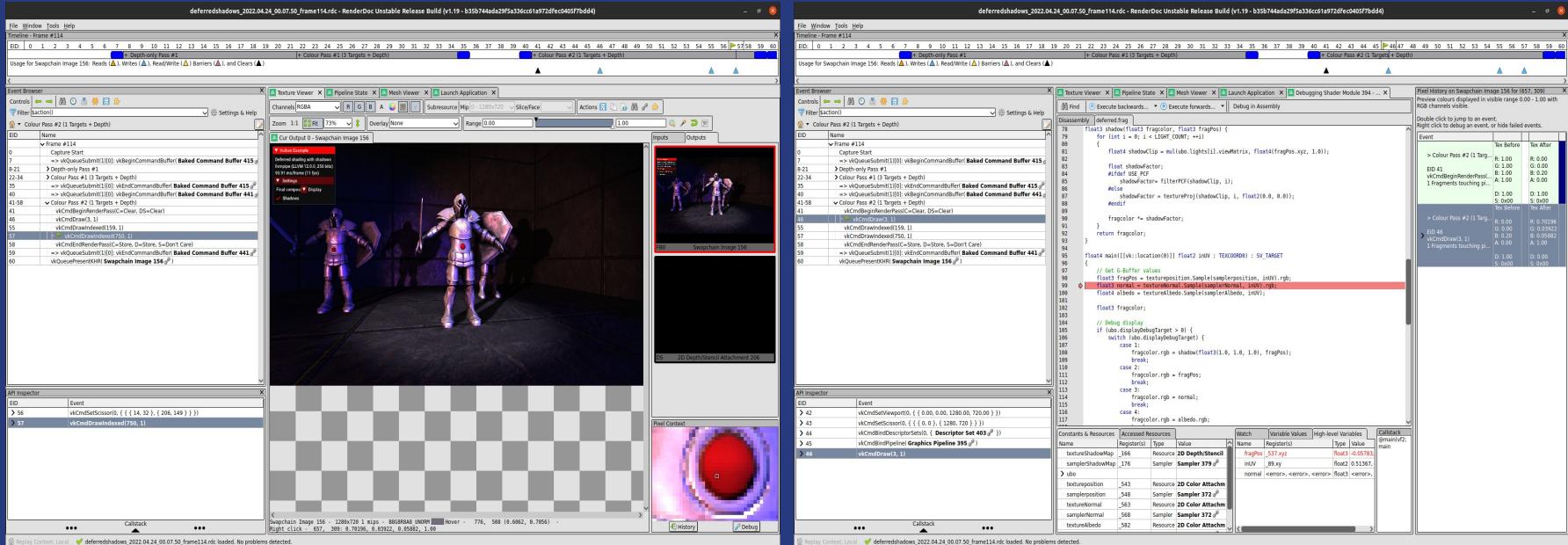
- **-gVS** instructs the compiler to embed the source string in the DebugSource instruction (similar to the `-fspv-debug=vulkan-with-source` argument in DXC).
- **-D** tells the compiler that the source is HLSL.
- **-V** tells the compiler that the source is GLSL under Vulkan semantics.

Debug information will increase the size of your SPIR-V (2-3X increase)

dxc/glslang Status

- Supported as of Vulkan SDK (1.3.231.1) and Renderdoc v1.25
- Valve shaders (thanks to Dan Ginsburg)
- Sascha Willems samples
 - 79 samples
 - 281 HLSL/GLSL Shaders
 - 127 vertex shaders
 - 131 fragment shaders
 - 10 compute shaders
 - 3 geometry shaders
 - 5 tessellation control shaders
 - 5 tessellation evaluation shaders
- Geom, Tesc, Tese untested
- DebugInfo not yet generated for: Ray Tracing, Mesh Shaders

Demo



Future work

- Squash bugs
- DebugBuildIdentifier / DebugStoragePath
 - Store debug information in a separate file

Thanks!

- Jeremy Hayes (LunarG)
 - original Vulkan Webinar presentation, DebugInfo in glslang
- Google
 - OpenCL.Debuginfo.100 support in DXC, SPIRV-Tools
- Baldur Karlsson (Valve)
 - NonSemantic.Shader.Debuginfo.100, RenderDoc
- Dan Ginsburg (Valve)
 - shaders, testing
- Sascha Willems
 - shaders, samples

Questions or presentation feedback?

Greg Fischer: ggreg@lunarg.com

Jeremy Hayes: jhayes@lunarg.com

Report bugs or make feature requests here:

<https://github.com/KhronosGroup/glslang>

<https://github.com/microsoft/DirectXShaderCompiler>

For more information:

<https://www.lunarg.com/news-insights/white-papers/source-level-shader-debugging-in-vulkan-with-renderdoc-feb2023/>





Thanks! Q&A?

Greg Fischer
LunarG, Inc.
greg@lunarg.com



Help Us Improve the
Vulkan SDK and Ecosystem

Share Your Feedback

Take the LunarG annual developer's survey

- Survey results are tabulated
- Shared with the Vulkan Working Group
- Actions are assigned
- Results are reported

Survey closes February 27, 2023



<https://www.surveymonkey.com/r/PVM92RH>