

Source-level Shader Debugging in Vulkan with RenderDoc

Jeremy Hayes, LunarG, Inc.
Greg Fischer, LunarG, Inc.
October 27, 2022



/ WEBINARS
& MEETUPS



Shader debugging in Vulkan is here!

The screenshot shows the Vulkan Profiler interface with the "Debugging Shader Module 372..." tab selected. The main window displays the assembly code for the `deferred.frag` shader. A specific line of code, `float3 normal = textureNormal.Sample(samplerNormal, inUV).rgb;`, is highlighted with a red rectangle, indicating it is the current focus of the debugger.

```
90         fragcolor *= shadowFactor;
91     }
92     return fragcolor;
93 }
94
95 float4 main([[vk::location(0)]] float2 inUV : TEXCOORD0) : SV_TARGET
96 {
97     // Get G-Buffer values
98     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    return float4(fragcolor, 1.0);
```

Below the code editor, there are three tables providing analysis and resources:

Constants & Resources		Accessed Resources	
Name	Register(s)	Type	Value
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		Variable Values		High-level Variables	
Name	Register(s)	Type	Value	Name	Value
fragPos	_414.xyz	float3	-0.13428, -1.42383, 0.46558	inUV	_401.xy
					float2 0.50977, 0.37431

Callstack			
main			

Source-level Shader Debugging in Vulkan with RenderDoc

Jeremy Hayes, LunarG, Inc.

Senior Graphics Software Engineer with 20 years of graphics experience (3Dlabs, Intel, LunarG).

Greg Fischer, LunarG, Inc.

Senior Graphics Software Engineer with 38 years of compiler experience across numerous graphics and compute platforms.

Slides are available at:

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

Agenda

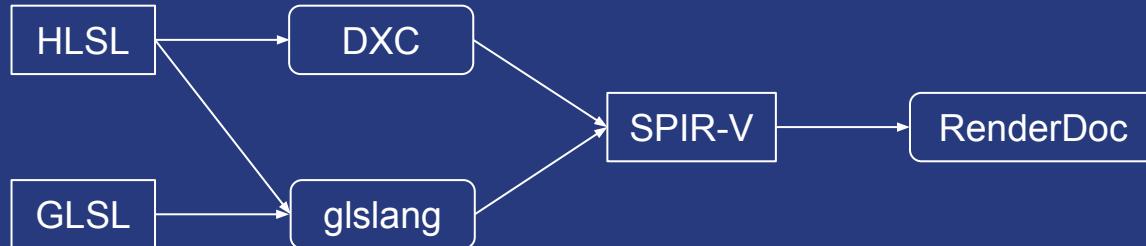
- What is NonSemantic.Shader.DebugInfo.100
- How to generate debug information
- Debug Instruction classifications
- Status of compiler toolchain support
- How to debug shaders in RenderDoc

NonSemantic.Shader.DebugInfo.100

- What?
 - Non-semantic SPIR-V extension (OpExtension “SPV_KHR_non_semantic_info”)
 - NonSemantic instructions can be safely ignored by drivers
 - Inspired and derived from DWARF debugging standard and OpenCL.DebugInfo.100
 - Authored by Baldur Karlsson from RenderDoc
- Why?
 - Information useful to debugging is lost during compilation and optimization
 - This extension preserves the lost information
 - Allows debugging of optimized shaders

NonSemantic.Shader.DebugInfo.100

- How?
 - `%result_id = OpExtInstImport "NonSemantic.Shader.DebugInfo.100"`
 - Annotates locations, scopes, types, variables, values, and procedures
 - DXC and glslang compiler support
 - SPIRV-Tools support
 - Optimized shaders can be debugged in addition to unoptimized shaders



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-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.
- `-fspv-debug=vulkan-with-source` instructs the compiler to embed the source string in the `DebugSource` instruction. RenderDoc reads the high-level source from this instruction.

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 (~70% increase)

Debug instruction classifications

- 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`

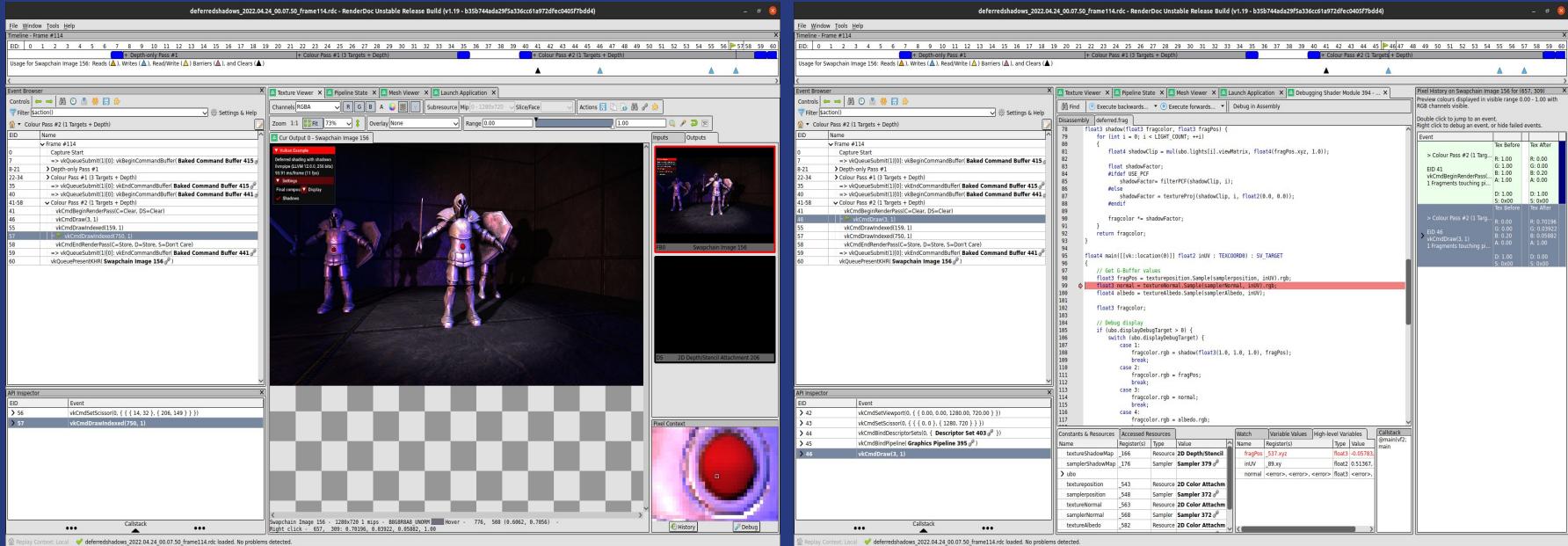
Debug instruction classifications

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

Status

- Supported in the newest LunarG Vulkan SDK (1.3.231.1)
- Valve shaders (thanks to Dan Ginsburg)
- Sascha Willems samples
 - 79 samples
 - 281 HLSL/GLSL Shaders
 - 127 vertex shaders
 - 131 fragment shaders
 - 3 geometry shaders
 - 10 compute shaders
 - 5 tessellation control shaders
 - 5 tessellation evaluation shaders
- Ray tracing not supported
- Mesh shading not supported

Demo



Future work

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

Thanks!

- Dan Ginsburg (shaders)
- Baldur Karlsson (NonSemantic.shader.debuginfo.100, RenderDoc)
- Sascha Willems (shaders)
- Google (OpenCL.debuginfo.100, DXC, SPIRV-Tools)

Questions or presentation feedback?

Contact Jeremy Hayes: jeremy@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/>



Thanks Q&A?

Jeremy Hayes
LunarG, Inc.

Greg Fischer
LunarG, Inc.



WEBINARS
& MEETUPS



