Source-level Shader Debugging in Vulkan With RenderDoc

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

Introduction

Source-level debugging is a powerful tool that can aid shader development. This capability is now available to Vulkan shaders using RenderDoc, the DirectX Shader Compiler (DXC), and the glslangValidator shader compiler. This document describes how to generate the necessary SPIR-V instructions which enable source-level debugging.

Generating debug information

Source-level debugging in SPIR-V requires an extension called NonSemantic.Shader.DebugInfo.100. This extension is supported by DXC and glslangValidator. The following stages are currently supported: vertex, geometry, tessellation control and evaluation, fragment, and compute. Shaders optimized by `spirv-opt` may also be debugged.

Using DXC

DXC can be used to generate debug information within SPIR-V from HLSL using the following command line:

```
\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>
```

The command-line arguments relevant to debug information are described below.

- `-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. This is not required for Vulkan1.3.
- `-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.

Using glslangValidator

glslangValidator can also be used to generate debug information within SPIR-V from HLSL. In addition, glslangValidator can also generate debug information from GLSL. To generate debug information from HLSL, use the following command line.

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

The command-line arguments relevant to debug information are described below.

- `-gVS` instructs the compiler to embed the source string in the `DebugSource` instruction (similar to the `-fspv-debug=vulkan-with-source` argument in DXC).
- `-Od` disables HLSL legalization and optimization (similar to the `-fcgl` argument in DXC).
- `-D` tells the compiler that the source is HLSL.

To generate debug information from GLSL, the command line is very similar.

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

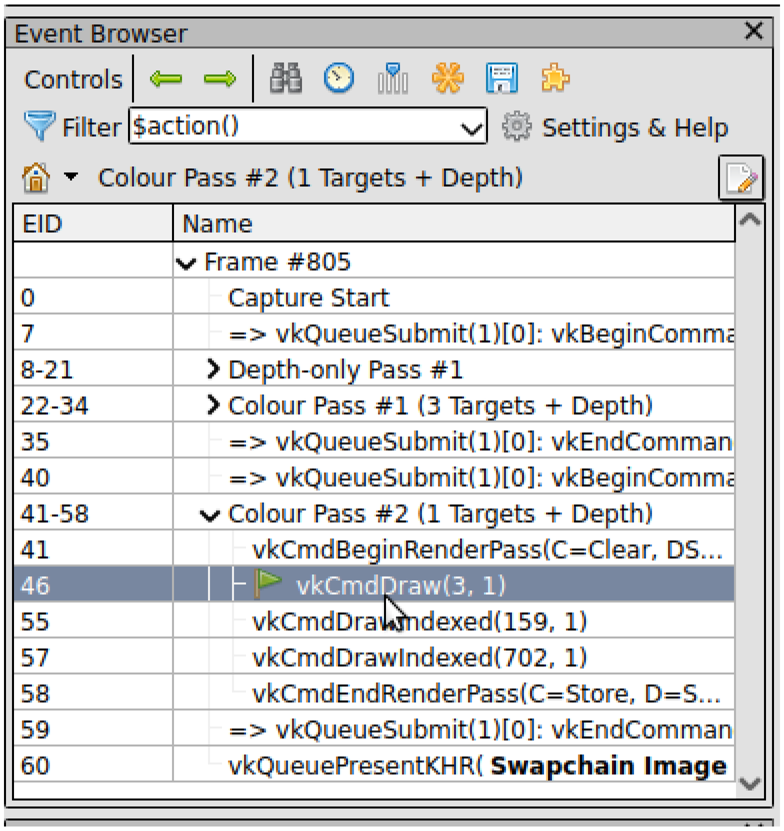
The only difference is that the `-D` argument is replaced with `-V`.

Debugging with RenderDoc

SPIR-V shaders containing debug information can be debugged interactively using RenderDoc. The latest stable builds (v1.22) of Renderdoc support this functionality. These builds can be located on the RenderDoc website.

A complete demonstration of RenderDoc is outside the scope of this document. However, we will demonstrate how to access the source debugger within RenderDoc. First, you will need to launch your application and capture a frame (please see the RenderDoc documentation if you do not know how to do this).

Once you have a captured frame, select a draw command from the event browser located on the left-hand side.

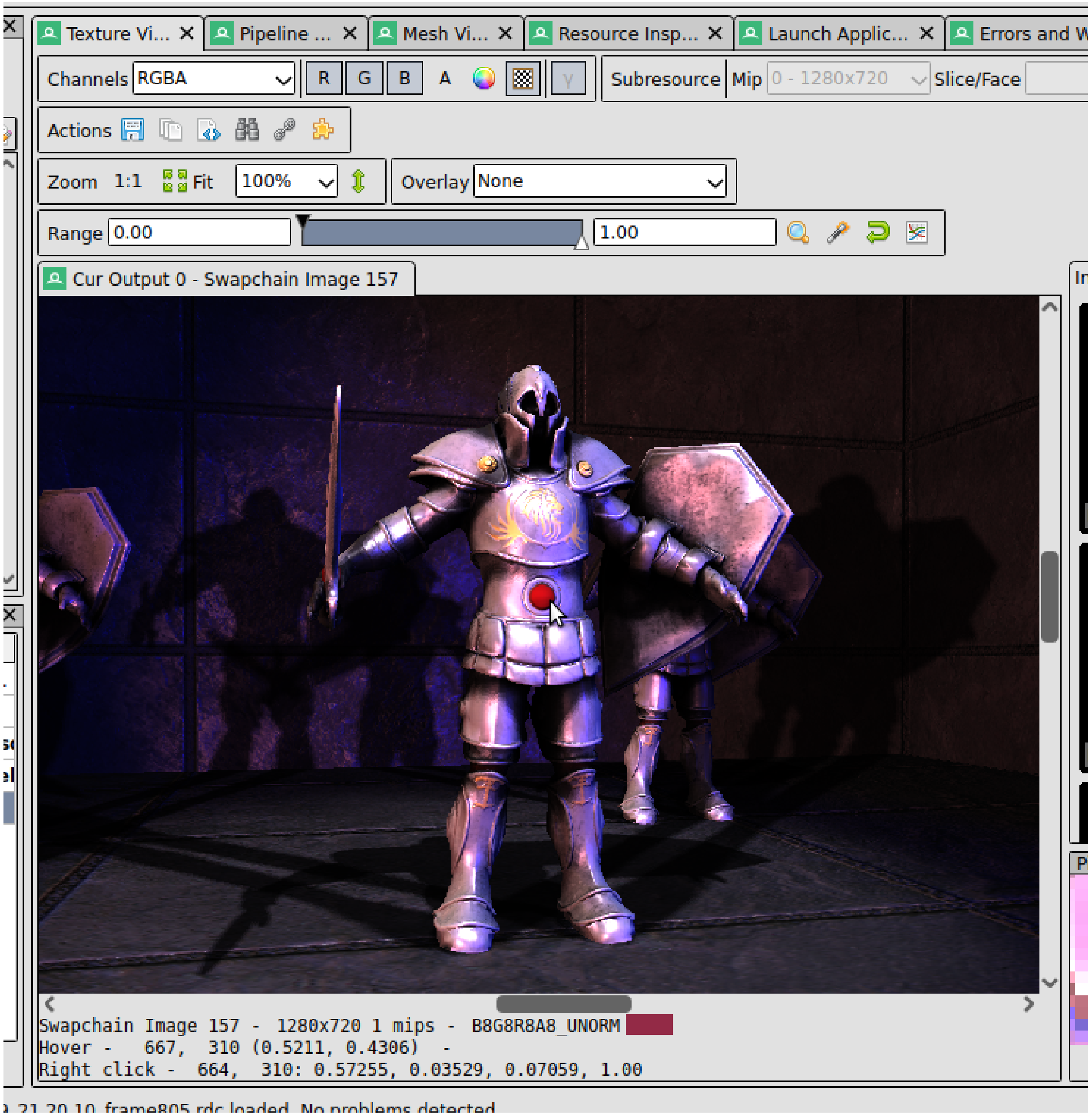


Filter: $action()

Colour Pass #2 (1 Targets + Depth)

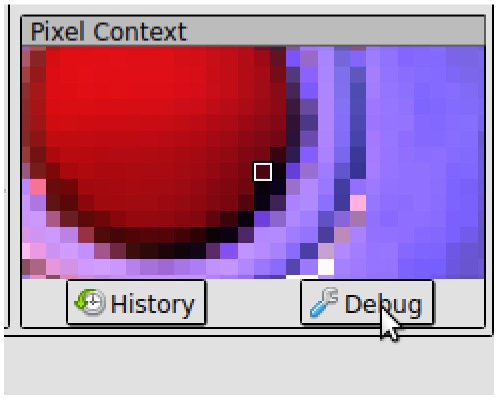
EID	Name
	Frame #805
0	Capture Start
7	=> vkQueueSubmit(1)[0]: vkBeginComma
8-21	Depth-only Pass #1
22-34	Colour Pass #1 (3 Targets + Depth)
35	=> vkQueueSubmit(1)[0]: vkEndComman
40	=> vkQueueSubmit(1)[0]: vkBeginComma
41-58	Colour Pass #2 (1 Targets + Depth)
41	vkCmdBeginRenderPass(C=Clear, DS...
46	vkCmdDraw(3, 1)
55	vkCmdDrawIndexed(159, 1)
57	vkCmdDrawIndexed(702, 1)
58	vkCmdEndRenderPass(C=Store, D=S...
59	=> vkQueueSubmit(1)[0]: vkEndComman
60	vkQueuePresentKHR(**Swapchain Image**

Next, right-click on a pixel in the texture viewer.

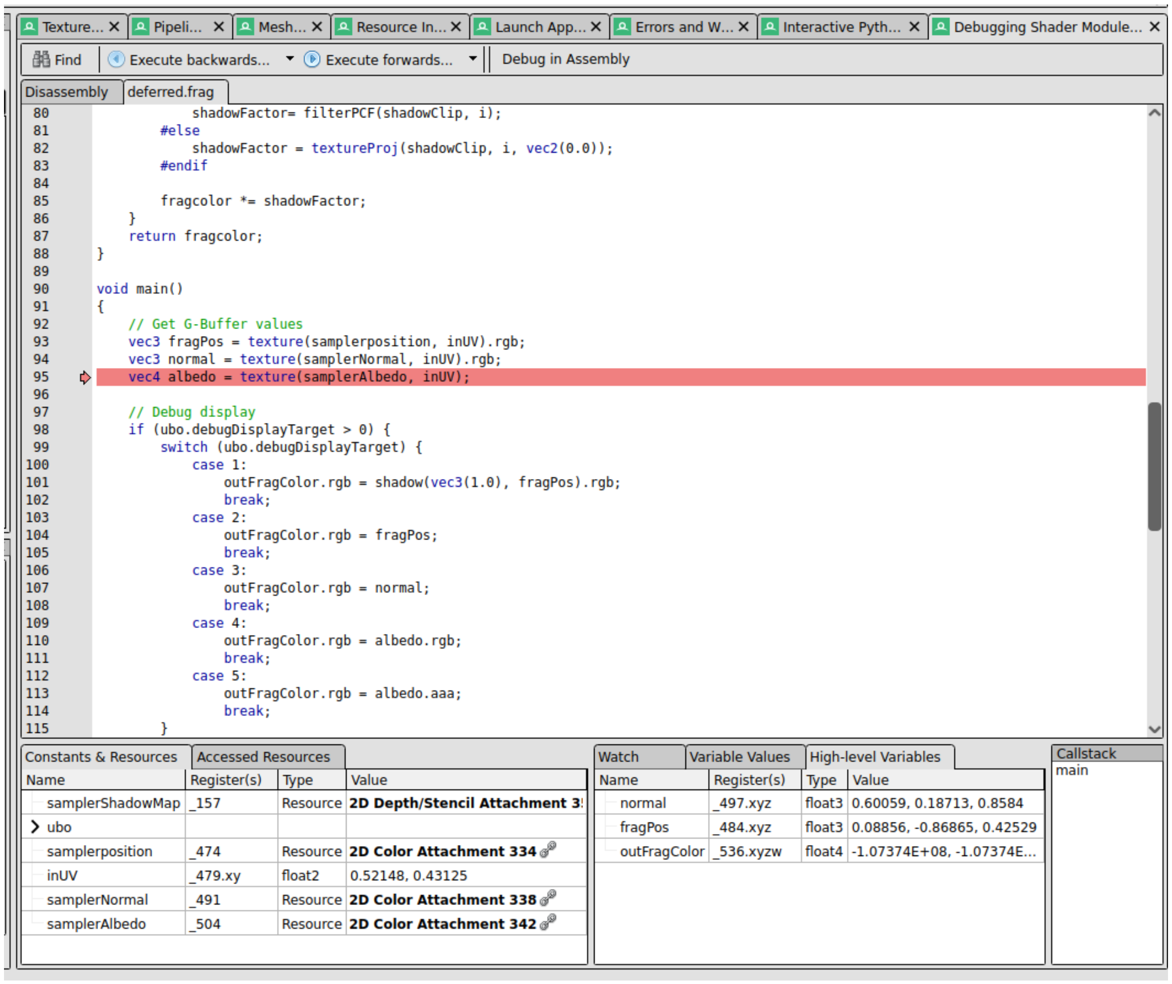


```
Swapchain Image 157 - 1280x720 1 mips - B8G8R8A8_UNORM
Hover -   667,  310 (0.5211, 0.4306)  -
Right click -  664,  310: 0.57255, 0.03529, 0.07059, 1.00
```

Next, click the debug button in the pixel context window located in the lower-right corner.



A window containing the fragment-shader source should appear. You can step forwards and backward through the statements using the buttons at the top of the debugger window. Variables identifiers and values will be displayed in the lower right corner.

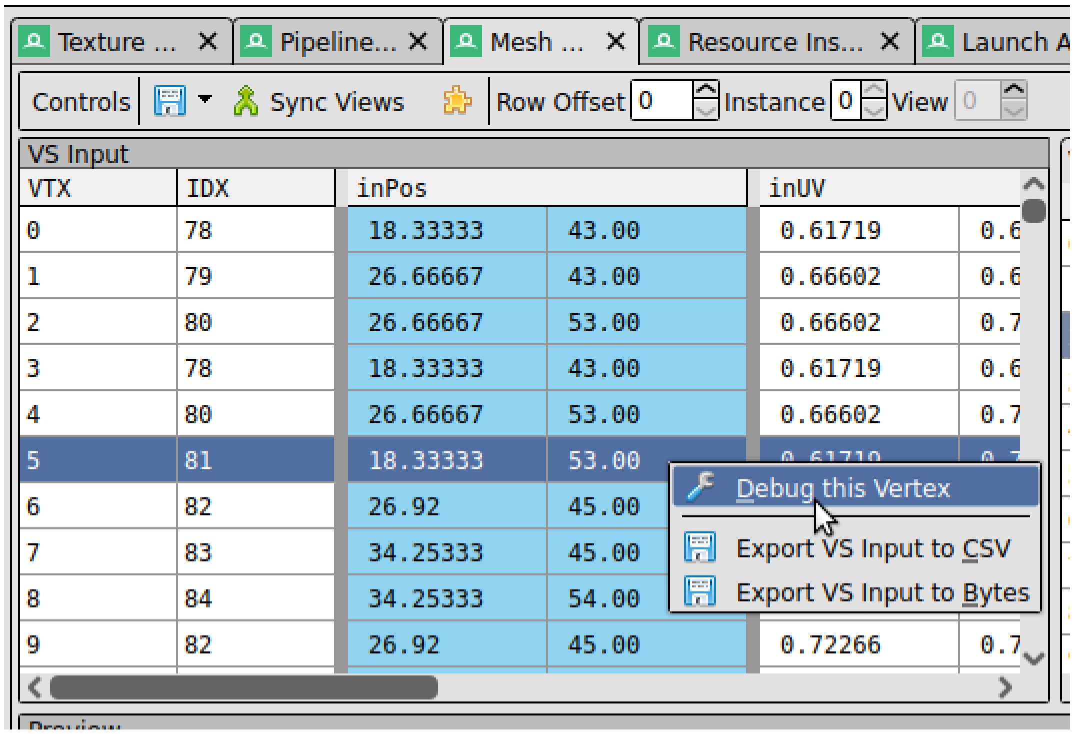


```
        shadowFactor= filterPCF(shadowClip, i);
    #else
        shadowFactor = textureProj(shadowClip, i, vec2(0.0));
    #endif

    fragcolor *= shadowFactor;
  }
  return fragcolor;
}

void main()
{
  // Get G-Buffer values
  vec3 fragPos = texture(samplerposition, inUV).rgb;
  vec3 normal = texture(samplerNormal, inUV).rgb;
  vec4 albedo = texture(samplerAlbedo, inUV);

  // Debug display
  if (ubo.debugDisplayTarget > 0) {
    switch (ubo.debugDisplayTarget) {
      case 1:
        outFragColor.rgb = shadow(vec3(1.0), fragPos).rgb;
        break;
      case 2:
        outFragColor.rgb = fragPos;
        break;
      case 3:
        outFragColor.rgb = normal;
        break;
      case 4:
        outFragColor.rgb = albedo.rgb;
        break;
      case 5:
        outFragColor.rgb = albedo.aaa;
        break;
    }
```

Name	Register(s)	Type	Value
samplerShadowMap	_157	Resource	**2D Depth/Stencil Attachment 3!**
ubo			
samplerposition	_474	Resource	**2D Color Attachment 334**
inUV	_479.xy	float2	0.52148, 0.43125
samplerNormal	_491	Resource	**2D Color Attachment 338**
samplerAlbedo	_504	Resource	**2D Color Attachment 342**

Name	Register(s)	Type	Value
normal	_497.xyz	float3	0.60059, 0.18713, 0.8584
fragPos	_484.xyz	float3	0.08856, -0.86865, 0.42529
outFragColor	_536.xyzw	float4	-1.07374E+08, -1.07374E...

Vertex shaders can also be debugged by right-clicking a vertex in the mesh viewer and selecting "Debug this Vertex."



VTX	IDX	inPos		inUV	
0	78	18.33333	43.00	0.61719	0.6
1	79	26.66667	43.00	0.66602	0.6
2	80	26.66667	53.00	0.66602	0.7
3	78	18.33333	43.00	0.61719	0.6
4	80	26.66667	53.00	0.66602	0.7
5	81	18.33333	53.00	0.61719	0.7
6	82	26.92	45.00		
7	83	34.25333	45.00		
8	84	34.25333	54.00		
9	82	26.92	45.00	0.72266	0.7

Future Work

Currently, all debug information is embedded within the SPIR-V shader. Future updates to DXC and glslangValidator will allow the debug information to be stored in a separate file (similar to a program database or .pdb file in Visual Studio). Support for other stages/shaders is also possible.

References

- DXC - https://github.com/Microsoft/DirectXShaderCompiler
- glslangValidator - https://github.com/KhronosGroup/glslang
- SPIR-V extension - https://htmlpreview.github.io/?https://github.com/KhronosGroup/SPIRV-Registry/blob/main/nonsemantic/NonSemantic.Shader.DebugInfo.100.html
- RenderDoc - https://renderdoc.org/