

Using Vulkan Debug Printf

Tony Barbour, LunarG August 2021

Introduction

Debugging Vulkan shaders, especially compute shaders, can be very difficult to do even with the aid of a powerful debugging tool like RenderDoc. Debug Printf is a recent Vulkan feature that allows developers to debug their shaders by inserting Debug Print statements. This feature is now supported within RenderDoc in a way that allows for per-invocation inspection of values in a shader. This article describes how to instrument your GLSL or HLSL shaders with Debug Printf and how to inspect and debug with them in RenderDoc, using vkconfig, or with environment variables.

Using Debug Printf in GLSL Shaders

To use Debug Printf in GLSL shaders, you need to enable the GL_EXT_debug_printf extension. Then add debugPrintfEXT calls at the locations in your shader where you want to print messages and/or values

Here is a very simple example:

```
#version 450
#extension GL_EXT_debug_printf : enable
void main() {
  float myfloat = 3.1415f;
   debugPrintfEXT("My float is %f", myfloat);
}
```

Then use glslangValidator to generate SPIR-V to use in vkCreateShaderModule. "glslangvalidator --target-env vulkan1.2 -x -e main -o shader.vert.spv shader.vert" would be one example of compiling shader.vert

Note that every time this shader is executed, "My float is 3.141500" will be printed. If this were in a vertex shader and a triangle was drawn, it would be printed 3 times.

Note also that the VK_KHR_shader_non_semantic_info device extension must be enabled in the Vulkan application using this shader.



Using Debug Printf in HLSL Shaders

In HLSL, debug printf can be invoked as follows:

```
void main() {
float myfloat = 3.1415;
printf("My float is %f", myfloat);
}
```

```
Use glslangValidator or dxc to generate SPIR-V for this shader.
For instance:
glslangValidator.exe -D --target-env vulkan1.2 -e main -x -o shader.vert.spvx shader.vert
dxc.exe -spirv -E main -T ps_6_0 -fspv-target-env=vulkan1.2 shader.vert -Fo shader.vert.spv
```

Note that the VK_KHR_shader_non_semantic_info device extension must also be enabled in the Vulkan application using this shader.

Using Debug Printf in SPIR-V Shaders

Normally, developers will use a high-level language like HLSL or GLSL to generate SPIR-V. However, in some cases, developers may wish to insert Debug Printfs directly into SPIR-V

To execute debug printfs in a SPIR-V shader, a developer will need the following two instructions specified:

OpExtension "SPV_KHR_non_semantic_info" %N0 = OpExtInstImport NonSemantic.DebugPrintf

Debug printf operations can then be specified in any function with the following instruction: %NN = OpExtInst %void %N0 1 %N1 %N2 %N3 ... where:

N0 is the result id of the OpExtInstImport NonSemantic.DebugPrintf

1 is the opcode of the DebugPrintf instruction in NonSemantic.DebugPrintf

- N1 is the result of an OpString containing the format for the debug printf
- N2, N3, ... are result ids of scalar and vector values to be printed

NN is the result id of the debug printf operation. This value is undefined.

Note that the VK_KHR_shader_non_semantic_info device extension must also be enabled in the Vulkan application using this shader.



Debug Printf messages in RenderDoc

As of RenderDoc release 1.14, Debug Printf statements can be added to shaders, and debug printf messages will be received and logged in the Event Browser window.

Using the debugmarker sample from Sascha Willems' Vulkan samples repository:

1. Capture a frame:





- 2. Edit the shader:
 - a. Add "#extension GL_EXT_debug_printf : enable" to beginning of shader
 - b. Add "debugPrintfEXT("Position = %v4f", pos);" to shader after pos definition
 - c. Hit Refresh

⇒ 🛱 🟲 🕑 🕅 券 🗒 🔁		nd 😤 Defrech 👝			
ame					
Draw "Grass_043"		decompiled			
vkCmdDrawIndexed(276, 1)	1	#version 450			
Draw "Grass_044"		<pre>#extension GL_EXT_debug_printf : enable</pre>			
vkCmdDrawIndexed(276, 1)	3	layout(set = 0, binding = 0, std140) uniform UBO			
Draw "chest"	4	t mate projection:			
vkCmdDrawIndexed(138, 1)	6	mat4 projection, mat4 model:			
vkCmdDrawIndexed(1098, 1)	7	vec4 lightPos;			
Draw "Emerald 001"	8	} ubo;			
vkCmdDrawIndexed(129, 1)	9				
vkCmdDrawIndexed(129, 1)	10	layout(location = 0) out vec3 outNormal;			
Draw "Emerald 002"	12	layout(location = 1) in vec3 inNormal;			
vkCmdDrawIndexed(129, 1)	13	<pre>layout(location = 3) in vec3 inColor;</pre>			
vkCmdDrawIndexed(129, 1)	14	layout(location = 2) out vec2 outUV;			
Draw "Emerald 003"	15	<pre>layout(location = 2) in vec2 inUV;</pre>			
vkCmdDrawIndexed(129_1)	16	<pre>layout(location = 0) in vec3 inPos;</pre>			
	17	<pre>layout(location = 4) out vec3 outLightVec;</pre>			
Wireframe draw	18	layout(location = 3) out vec3 outViewVec;			
And a set of	20	void main()			
 Apply post processing Apply post processing 	21	{			
VkCmdDrawIndexed(159, 1)	22	outNormal = inNormal;			
vkCmdDrawIndexed(858, 1)	23	<pre>outColor = inColor;</pre>			
vkCmdEndRenderPass(C=Store, D=Store, S=	24	outUV = inUV;			
=> vkQueueSubmit(1)[0]: vkEndCommandBuffer(Ba	25	<pre>gl_Position = (ubo.projection * ubo.model) * vec4(inPos,</pre>			
vkQueuePresentKHR (Swapchain Image 172 🖉 🖳		<pre>vec4 pos = ubo.model * vec4(inPos, 1.0); debugDrintfFVT("Desition = &u(f" pos);</pre>			
	28	outNormal = mat3(ubo model[0] xvz_ubo model[1] xvz_ubo			
X	29	<pre>vec3 1Pos = mat3(ubo.model[0].xyz, ubo.model[1].xyz, ubo.</pre>			
Event	30	outLightVec = 1Pos - pos.xyz;			
vkCmdDrawIndexed	31	outViewVec = -pos.xyz;			

vkCmdDrawIndexed in question now has 51 messages.



3. Click on msg(s) to see Debug Printf output per draw:

Event Browser		×	Tex	tu X		ineli X		×
Controls 🦛 •	→ 船 🕨 🕥 🛝 👾 🗐 🌧		Chale mercans for ELL 0220 + MC/m2/2 mile deared (20, 1)					
EID	Jame	~	andee messages nom <u>Liz wyse</u> - vecimeranizaeee(125, 1)					
315	Draw "Grass_043"		Filter					
316	vkCmdDrawIndexed(276, 1)		Show messages from: Vertex Fragment					
317	Draw "Grass_044"		Eilter e		a have			
318	vkCmdDrawIndexed(276, 1)		L'itter i	nessag	etext		- Price	<u> </u>
319	Draw "chest"		Debug	G	io to	Location ^	Message	^
320	vkCmdDrawIndexed(138, 1)		8		酳	Idx 4025	Position = 0.212795, 0.593742, -5.542615, 1.000000	
321	vkCmdDrawIndexed(1098, 1)		ß	í –	品品	Tel:: 40.25	Desitive = 0.212705_0_5027425_542615_1_000000	
322	Draw "Emerald_001"				32	10x 4025	Postor = 0.212795, 0.595742, -5.542015, 1.00000	-
323	vkCmdDrawIndexed(129, 1)				88	Idx 4026	Position = 0.149710, 0.571184, -5.552277, 1.000000	
324	vkCmdDrawIndexed(129, 1)		6		鶕	Idx 4026	Position = 0.149710, 0.571184, -5.552277, 1.000000	
325	Draw "Emerald_002"		1	1	商品	Idx 4027	Position = 0.205543, 0.590240, -5.553230, 1.000000	
326	vkCmdDrawIndexed(129, 1)		13	í	23	Idv: 4027	Berting = 0.205542.0.500240 .5 552220.1.000000	
327	VKCmdDrawIndexed(129, 1)		0		33	10x 4027	Position = 0.205545, 0.590240, -5.555250, 1.000000	
320			l l		曲	Idx 4028	Position = 0.194664, 0.592898, -5.486473, 1.000000	
330	L b v/cmdDrawIndexed(120, 1) A 51 mcs(c)		1		êñ.	Idx 4028	Position = 0.194664, 0.592898, -5.486473, 1.000000	
332-489	> Wireframe draw		J		鶕	Idx 4029	Position = 0.147017, 0.565112, -5.551942, 1.000000	
492-494	Apply post processing		8		鶕	Idx 4029	Position = 0.147017, 0.565112, -5.551942, 1.000000	
504	vkCmdDrawIndexed(159, 1)		3	í –	88	Tdy 4029	Position = 0 147017 0 565112 -5 551942 1 000000	-
506	vkCmdDrawIndexed(858, 1)		6	1	33	102.5		-
507	vkCmdEndRenderPass(C=Store, D=Store, S=				88	Idx 4030	Position = 0.267668, 0.620570, -5.513439, 1.000000	-
509	=> vkQueueSubmit(1)[0]: vkEndCommandBuffer(Ba		1		68	Idx 4030	Position = 0.267668, 0.620570, -5.513439, 1.000000	
1510	VKQueuePresentKHR(Swapchain Image 172@*)	\sim	J		鶕	Idx 4031	Position = 0.224599, 0.597462, -5.560269, 1.000000	
API Inspector		×	1		酳	Idx 4031	Position = 0.224599, 0.597462, -5.560269, 1.000000	
EID	Event		6		88	Idx 4032	Position = 0.224572, 0.598206, -5.546965, 1.000000	
> 330	vkCmdDrawIndexed		P]	鶕	Idx 4032	Position = 0.224572, 0.598206, -5.546965, 1.000000	
			1		88	Idx 4033	Position = 0.194930, 0.585436, -5.619911, 1.000000	
			J]	鶕	Idx 4033	Position = 0.194930, 0.585436, -5.619911, 1.000000	
			1		ê₿	Idx 4034	Position = 0.267833, 0.615958, -5.595908, 1.000000	
L			J		曲	Idx 4034	Position = 0.267833, 0.615958, -5.595908, 1.000000	
•	Calistack		1		鹛	Idx 4035	Position = 0.212838, 0.592539, -5.564141, 1.000000	~



Debug Printf messages from Validation Layers via VkConfig (Vulkan Configurator)

Here's an example of adding a Debug Printf statement to the shader in the vkcube demo (from the Vulkan-Tools repository), and then using VkConfig to enable Debug Printf, launch vkcube, and see the Debug Printf output.

- 1. Add Debug Printf to the vkcube demo:
 - a. Add VK_KHR_shader_non_semantic_info to cube's CreateDevice
 - b. Add extension and debugPrintfEXT call to the shader
 - c. Use glslangvalidator to compile the new shader
 - d. (Offscreen) Rebuild vkcube





- 2. Configure VkConfig to enable Debug Printf
 - a. Set Shader Printf Preset
 - b. Set the executable path to the vkcube demo and add --c 1 to the command line to render one frame
 - c. Click the "Launch" button

₩ Vulkan Configurator 2.4.0 <active></active>		- 0	\times
Tools Help			
Vulkan Layers Management		Validation Settings	
O Layers Fully Controlled by the Vu	Ikan Applications	Edit Layers Find Layers.	
Overriding Layers by the Vulkan 0	Configurator	VK LAYER KHRONOS validation	^
Apply only to the Vulkan App	lications List Edit Application	s Shader Printf Preset	~
Continue Overriding Layers of	> Validation Areas		
		✓ Debug Action	
Vulkan Layers Configurations		Log Message	
O API dump		Debug Output	
O Frame Capture		Break	
 Portability 		✓ Log Filename	
 Synchronization 	stdout		
 Validation 		✓ Message Severity	
		✓ Info	
		✓ Warning	
		✓ performance	
Vulkan Application Launcher		Error	
		Debug	
 Executable Path 	C:\Users\tony\Vulkan-Tools\build\cube\Debug\vkcube.exe ~	··· Duplicated Messages Limit	10
Working Directory	C:\Users\tony\Vulkan-Tools\build\cube\Debug	Mute Message VUIDs	+
Command-line Arguments	c 1	✓ Disables	
Output Log	C:\Users\tony\VulkanSDK\vkcube.txt	✓ Thread Safety	
		Stateless Parameter	
Clear log at launch Clear	Vulkan Loader Messages: none 🗸 Laur	ch Object Lifetime	
- User-Defined Layers Paths fr - User-Defined Layers Paths fr	om VK_LAYEK_PAIH environment Variable: None Tom Vulkan Configurator: None	▲ Core	
 Global `vk_layer_settings.t> C:\Users\tonv\AppData\Loca 	Handle Wrapping		
- Available Layers:	Disable Shaders		
- VK_LAYER_RENDERDOC_Captu	Command Buffer State		
 VK_LAYER_LUNARG_override VK LAYER LUNARG api dump 	Image Layout	~	

3. See the Debug Printf output in Launcher window:

Vulkan Application Launcher						
✓ Executable Path	C:\Users\tony\Vulkan-Tools\build\cube\Debug\vkcube.exe					
Working Directory	C:\Users\tony\Vulkan-Tools\build\cube\Debug					
Command-line Arguments	c 1					
Output Log	C:\Users\tony\VulkanSDK\vkcube.txt					
☑ Clear log at launch Clear ✓ Ulkan Loader Messages: none ✓ Launch						
Settings File: Found at C:\Users\tony\AppData\Local\LunarG\vkconfig\override\vk_layer_settings.txt specified by VkConfig application override. Current Enables: VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT. Current Disables: VK_VALIDATION_FEATURE_DISABLE_OBJECT_LIFETIMES_EXT, VK_VALIDATION_FEATURE_DISABLE_CORE_CHECKS_EXT, VK_VALIDATION_FEATURE_DISABLE_THREAD_SAFETY_EXT, VK_VALIDATION_FEATURE_DISABLE_API_PARAMETERS_EXT, VK_VALIDATION_FEATURE_DISABLE_UNIQUE_HANDLES_EXT.						
Objects: 1 [0] 0x1cd8ff7b820, type: 1, name: NULL						
Selected GPU 0: GeForce GTX 660 Ti, type: 2						
texcoord = 0.000000, 1.0000000, 0.0000000, 0.000000						
Process terminated						



Debug Printf messages from Validation Layers via Environment Variables

With validation layers installed or available, you can set environment variables that will enable the display of any Debug Printf messages that your program generates. Setting the following environment variables and then running your program should send any Debug Printf messages it generates to stdout:

- Set VK_LAYER_PATH to directory containing KHRONOS_validation layer (i.e. /VulkanSDK/<SDK version>/bin)
- Set VK_INSTANCE_LAYERS to VK_LAYER_KHRONOS_validation
- Set VK_LAYER_ENABLES to VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT
- Set VK_LAYER_DISABLES to VK_VALIDATION_FEATURE_DISABLE_ALL_EXT
- Set DEBUG_PRINTF_TO_STDOUT to true
- Run your program

Debug Printf Format String

The format string for this implementation of Debug Printf is more restricted than the traditional printf format string.

Format for specifier is "%"precision <d, i, o, u, x, X, a, A, e, E, f, F, g, G, ul, lu, or lx>

Format for vector specifier is "%"precision"v" [2, 3, or 4] [specifiers list above]

- The vector value separator is ", "
- "%%" will print as "%"
- No length modifiers. Everything except ul, lu, and lx is 32 bits, and ul and lx values are printed in hex
- No strings or characters allowed
- No flags or width specifications allowed
- No error checking for invalid format strings is done.



For example:

float myfloat = 3.1415f; vec4 floatvec = vec4(1.2f, 2.2f, 3.2f, 4.2f); uint64_t bigvar = 0x200000000000001ul; debugPrintfEXT("Here's a float value to 2 decimals %1.2f", myfloat); Would print "Here's a float value to 2 decimals 3.14" debugPrintfEXT("Here's a vector of floats %1.2v4f", floatvec); Would print "Here's a vector of floats 1.20, 2.20, 3.20, 4.20" debugPrintfEXT("Unsigned long as decimal %lu and as hex 0x%lx", bigvar, bigvar); Would print "Unsigned long as decimal 2305843009213693953 and as hex 0x2000000000001"

Debug Printf Settings

The following settings are available for Debug Printf: 1) Printf to stdout, 2) Verbose, and 3) Print buffer size.

😽 Vulkan Configurator 2.4.0 < ACTIVE>		- 🗆 X		
Tools Help				
Vulkan Layers Management	Validation Settings			
O Layers Fully Controlled by the Vull	Edit Layers Find Layers			
Overriding Layers by the Vulkan Co	Validation Areas			
Apply only to the Vulkan Appli	Debug Action			
Continue Overriding Lavers on	✓ Log Message			
	Debug Output			
Vulkan Layers Configurations	Break			
O API dump	O API dump			
⊖ Frame Capture	> Message Severity			
 Portability 	Duplicated Messages 10			
 Synchronization 	Mute Message VUIDs +			
 Validation 	> Disables			
		✓ Enables		
Vulkan Application Launcher		Synchronization		
Vulkan Application Launcher		✓ Debug Printf		
✓ Executable Path	C:\VulkanSDK\1.2.182.0\Bin\vkcube.exe v	GPU-Assisted		
Working Directory	C:\VulkanSDK\1.2.182.0\Bin	Reserve Descriptor Set Bindir		
Command-line Arguments	suppress_popups	Best Practices		
Output Log	C:\Users\tony\VulkanSDK\vkcube.txt	ARM-specific best practices		
Clear log at launch Clear	Vulkan Loader Messages: none V Launch	Verbose		
- VK_LAYER_LUNARG_monitor(- VK_LAYER_LUNARG_screensho Physical Devices: - GeForce GTX 660 Ti (Discr	Printf buffer size (byte 1024 ✓ Check Out of Bounds ✓ Check Draw Indirect Count Buffer			



Printf to stdout

Debug Printf messages can be sent to a registered debug callback or sent to stdout. This can also be enabled by setting the environment variable DEBUG_PRINTF_TO_STDOUT.

Verbose

Debug Printf messages can show just the basic information and messages, or if the verbose option is selected, the messages will contain pipeline stage, shader id, line number, and other information in addition to the resulting string.

Printf buffer size

This setting allows you to specify the size of the per draw buffer, in bytes of device memory, for returning Debug Printf values. The default is 1024 bytes. Each printf will require 32 bytes for header information and an additional four bytes for each 32-bit value being printed and an additional 8 bytes for each 64-bit value. If printfs are truncated due to lack of memory, a warning will be sent to the Vulkan debug callback.

Limitations

- Debug Printf cannot be used at the same time as GPU Assisted Validation.
- Debug Printf consumes a descriptor set. If your application uses every last descriptor set on the GPU, Debug Printf will not work.
- Debug Printf consumes device memory on the GPU. Large or numerous Debug Printf messages can exhaust device memory. See settings above to control buffer size.
- Validation Layers version: 1.2.135.0 or later is required
- Vulkan API version 1.1 or greater is required
- VkPhysicalDevice features: fragmentStoresAndAtomics and vertexPipelineStoresAndAtomics are required
- VK_KHR_shader_non_semantic_info extension supported and enabled
- RenderDoc release 1.14 or later
- When using Debug Printf with a debug callback, it is recommended to disable validation, as the debug level of INFO or DEBUG causes the validation layers to



produce many messages unrelated to Debug Printf, making it difficult to find the desired output.

Other References

Documentation for the GL_EXT_debug_printf extension can be found here.

There is a validation layer test that demonstrates the simple and programmatic use of Debug Printf. It is called "GpuDebugPrintf" and is in <u>vklayertests_gpu.cpp</u> in the Vulkan-ValidationLayers repository.

Document Change Log

July 29, 2021

Added more mentions of the need to enable the VK_KHR_shader_non_semantic_info extension in the application