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.

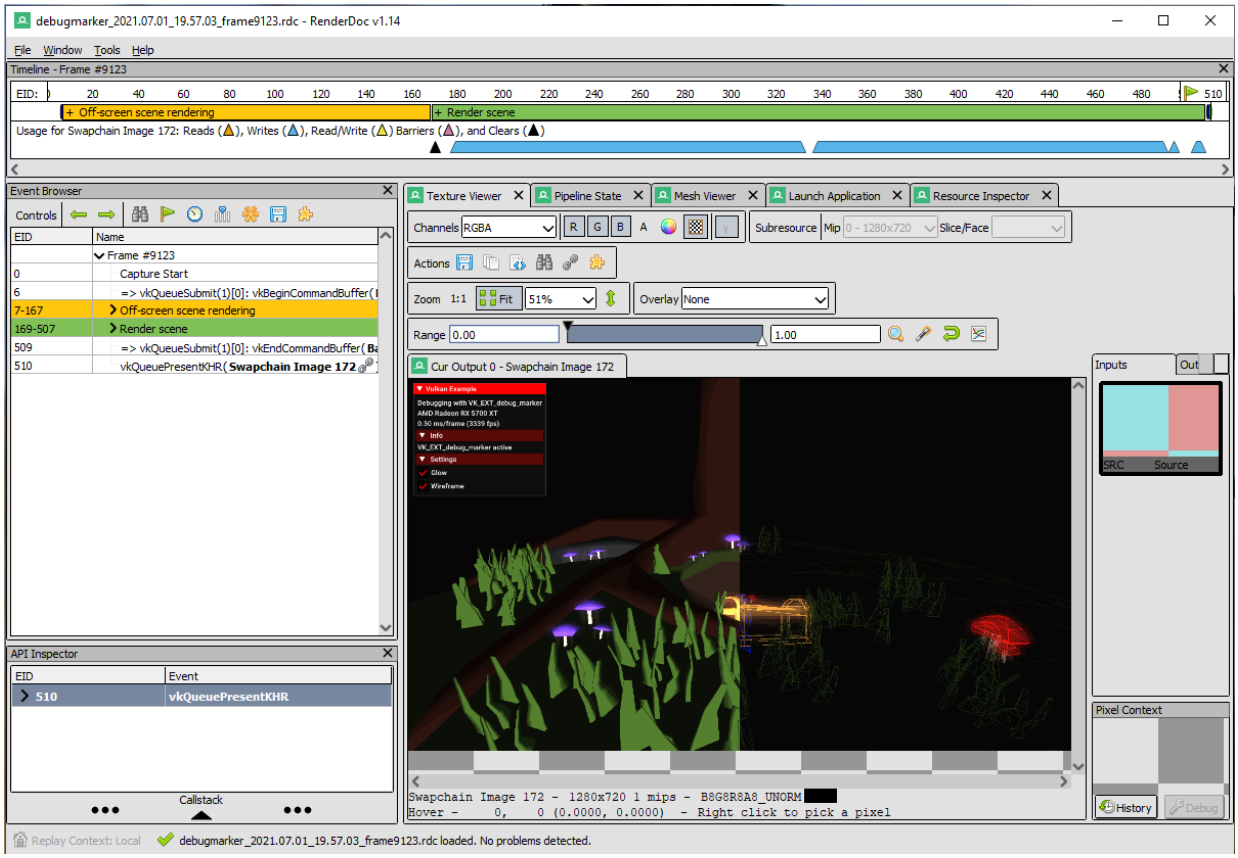
Note that the VK_KHR_shader_non_semantic_info device extension must also be enabled in the Vulkan application using this shader.

Debug Printf messages in RenderDoc

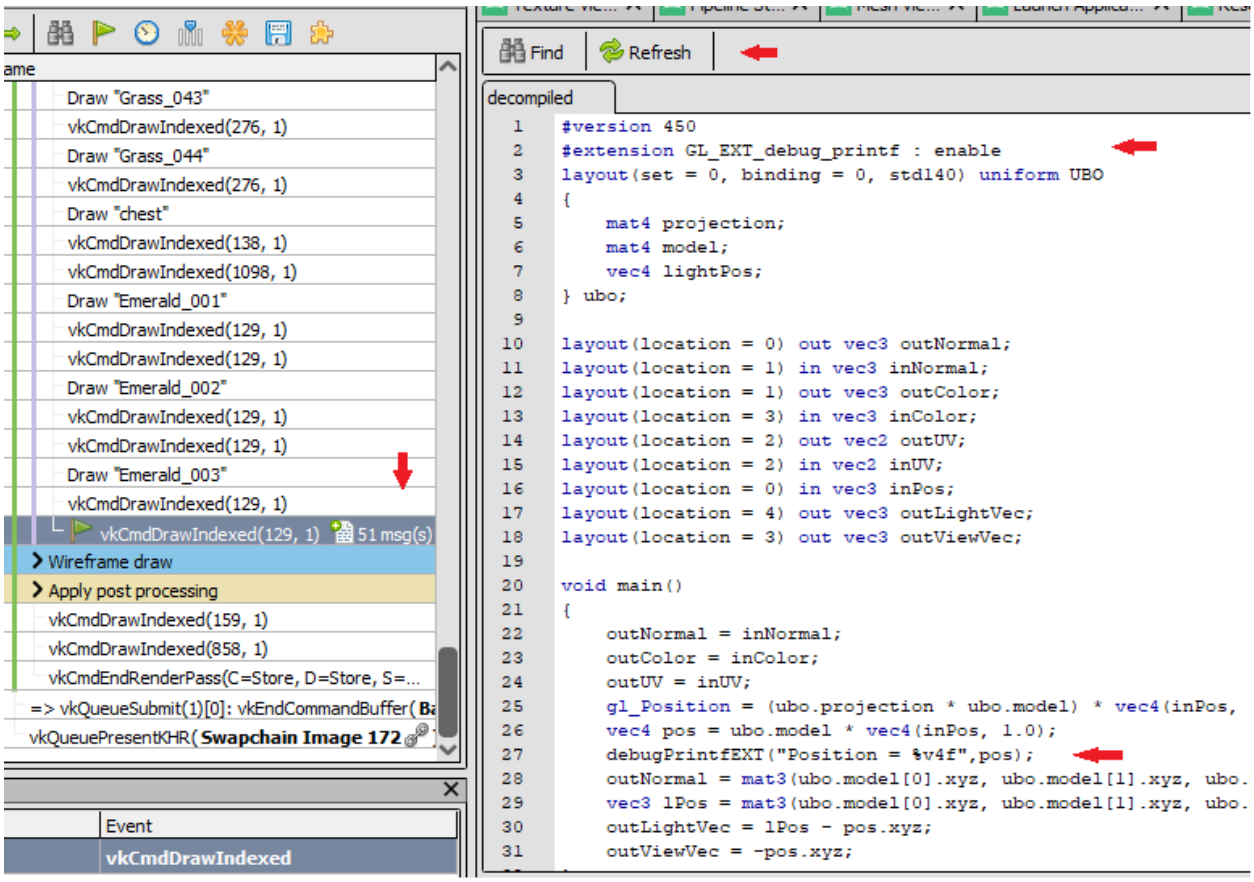
As of RenderDoc release 1.14, Debug Printf statements can be added to shaders, and debug printf messages will be received and logged in the Event Browser window.

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

1. Capture a frame:

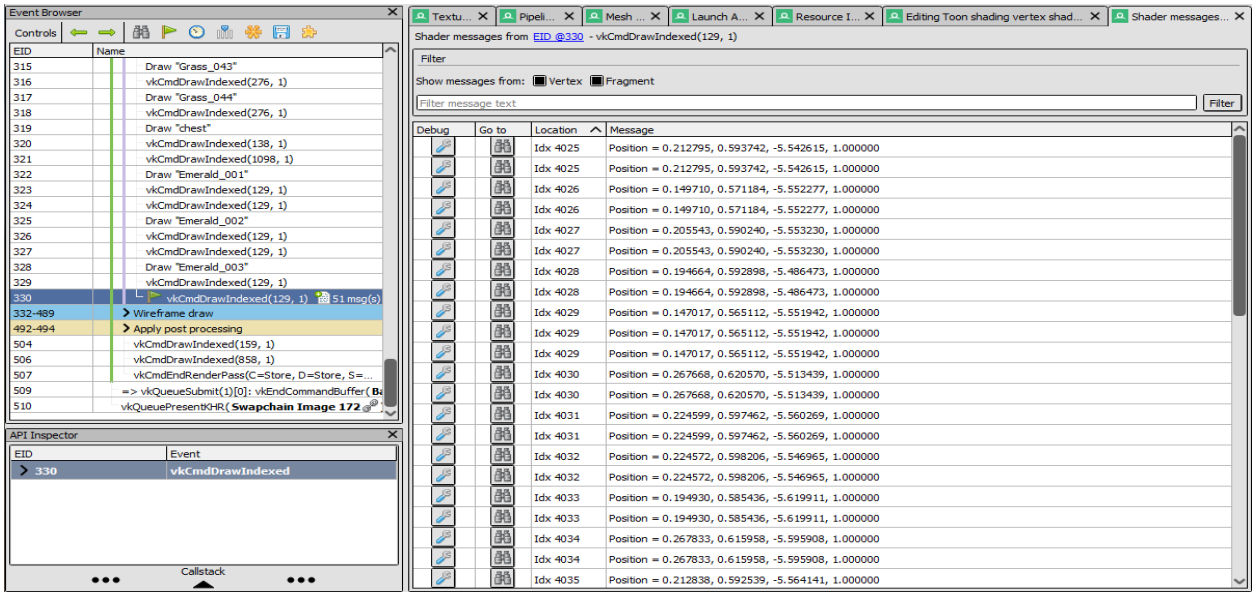


2. Edit the shader:
    a. Add “#extension GL_EXT_debug_printf : enable” to beginning of shader
    b. Add “debugPrintfEXT(“Position = %v4f”, pos);” to shader after pos definition
    c. Hit Refresh



vkCmdDrawIndexed in question now has 51 messages.

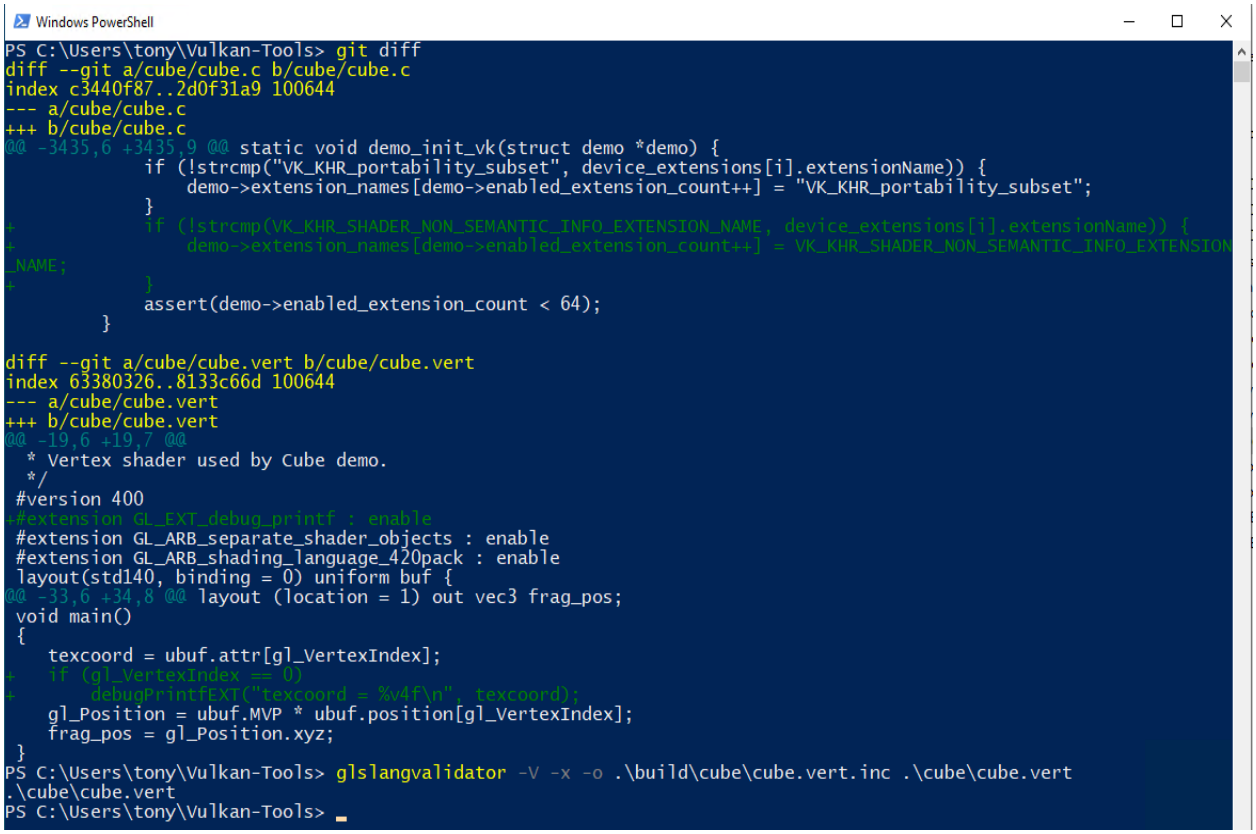
3. Click on msg(s) to see Debug Printf output per draw:



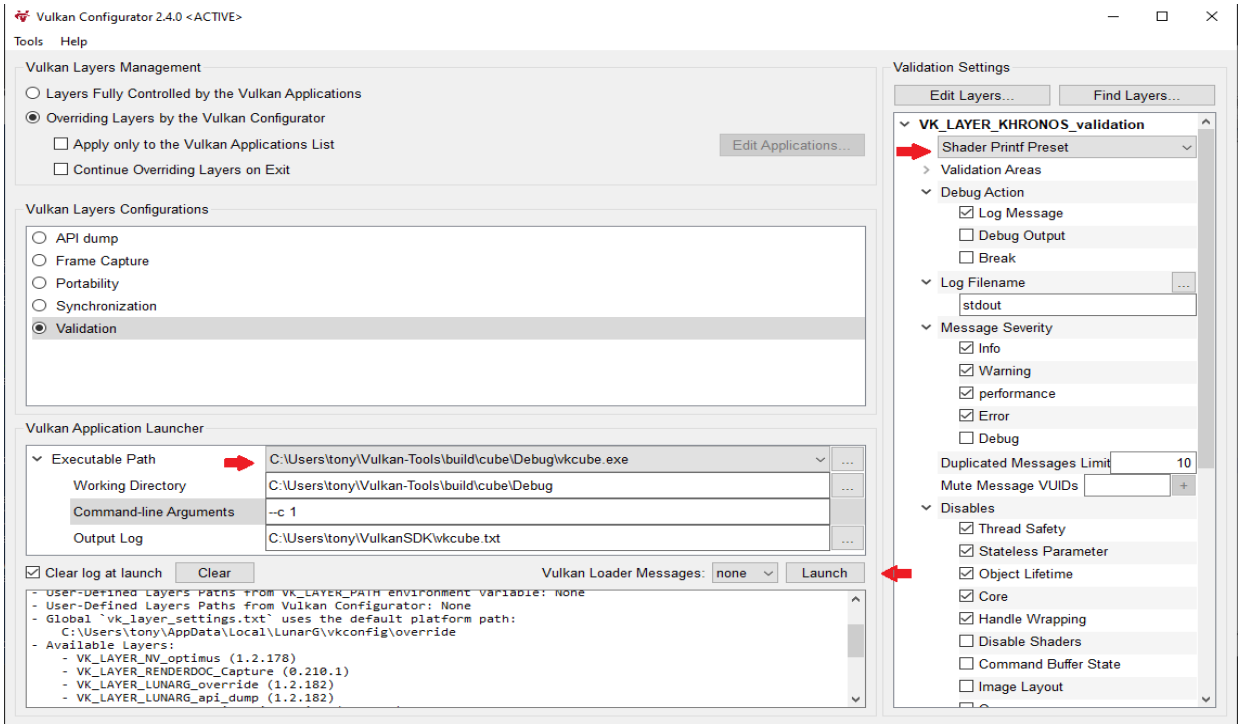
Debug Printf messages from Validation Layers via VkConfig (Vulkan Configurator)

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.

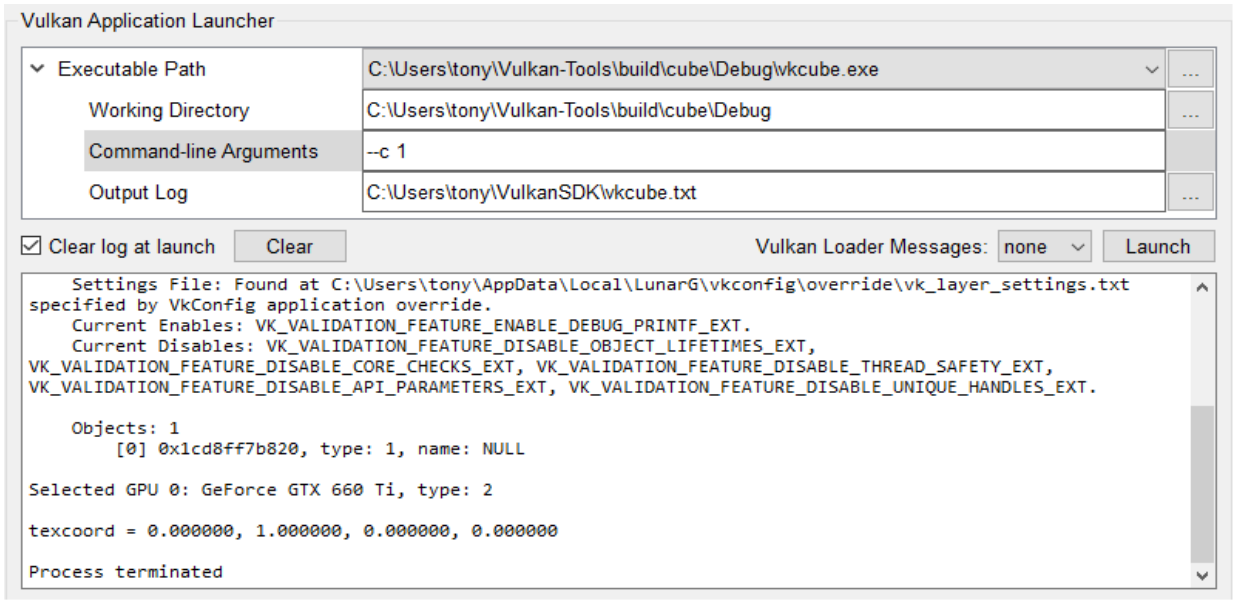
1. Add Debug Printf to the vkcube demo:
    a. Add VK_KHR_shader_non_semantic_info to cube's CreateDevice
    b. Add extension and debugPrintfEXT call to the shader
    c. Use glslangvalidator to compile the new shader
    d. (Offscreen) Rebuild vkcube



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



3. See the Debug Printf output in Launcher window:



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]

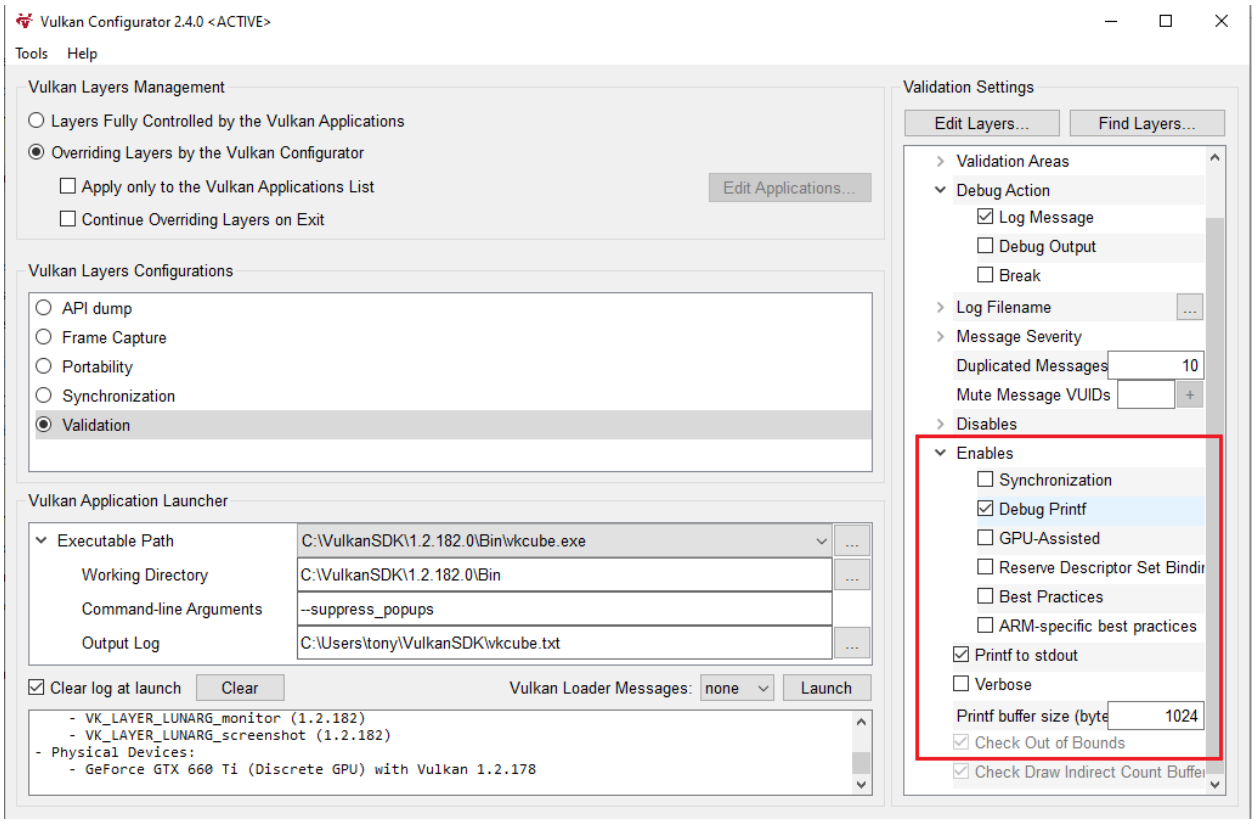
- The vector value separator is ", "
- "%%" will print as "%"
- No length modifiers. Everything except ul, lu, and lx is 32 bits, and ul and lx values are printed in hex
- No strings or characters allowed
- No flags or width specifications allowed
- No error checking for invalid format strings is done.

For example:

```
float myfloat = 3.1415f;
vec4 floatvec = vec4(1.2f, 2.2f, 3.2f, 4.2f);
uint64_t bigvar = 0x2000000000000001ul;
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
0x2000000000000001”
```

Debug Printf Settings

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



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 vklayertests_gpu.cpp 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