Results of the February 2023
LunarG Vulkan Ecosystem & SDK Survey

Report for the Public Vulkan Community

Executive Summary

This report provides the results from the LunarG ecosystem survey completed in February of 2023.

Methodology

1. LunarG and Khronos developer relations staff developed this Vulkan ecosystem survey to gauge the Vulkan community's use of and satisfaction with the current Vulkan ecosystem.
2. We attempted to reach as many Vulkan developers as possible -- both SDK users and non-SDK users. The survey was promoted on Twitter, Reddit, LinkedIn, the Khronos Vulkan slack channel, Vulkan Discord, and sent directly to 13,000+ recipients of the LunarG Vulkan SDK mailing list. Khronos helped to amplify the survey through their twitter account and newsletter mailings.
3. All comments from open-ended questions were included in this report, regardless if they were repeated. This helps you see the frequency of certain types of feedback.

Highlights

1. There were 275 respondents.
2. Target platforms for applications are (in order of usage) Windows desktop 10, Desktop Linux, Windows desktop 11, Android, macOS, Linux/ARM64, SteamDeck, and Windows/ARM64.
3. Some year over year insights:
    a. Compared to the previous year's survey, the number of folks who have released their Vulkan application for public use has increased from 28% to 36%.
    b. The top 8 use case categories have remained the same for multiple years.
    c. Windows, Linux, Android, and MoltenVK remain the top 4 target platforms (with the same ordering).
    d. glslangValidator (glsl->SPIR-V) or shaderc (glsls->SPIR-V) remains the most used front end from which to generate SPIR-V.
    e. Validation Layer coverage had a significant improvement from "medium coverage" to "high coverage". This is good positive progress.
4. Validation Layer improvement themes were:
    a. Continue to increase validation layer coverage
    b. Error messages are very verbose and could be formatted better for easier reading
    c. Interpreting errors (finding root cause of my error) is difficult
    d. Improve the performance
5. There was some vocal open-ended feedback about macOS and MoltenVK and getting that platform up to par with the other platforms.
6. Multiple comments/concerns about the shader toolchain (needs more maintenance and enhancement). Should also note that 60% of the population uses glslangValidator vs. 20% of the population uses DXC.
7. In regards to developer tasks, the top items that were the easiest to perform with the existing toolset were
    a. Debugging visual artifacts
    b. Creating Vulkan conformant application code
    c. Debugging crashes
    d. Debugging shader issues

8. In regards to developer tasks, the most difficult tasks to perform with the existing tool sets were
    a. Identifying driver bugs
    b. Debugging layer issues
    c. Debugging Vulkan installation and configuration issues

Potential actions and future priorities

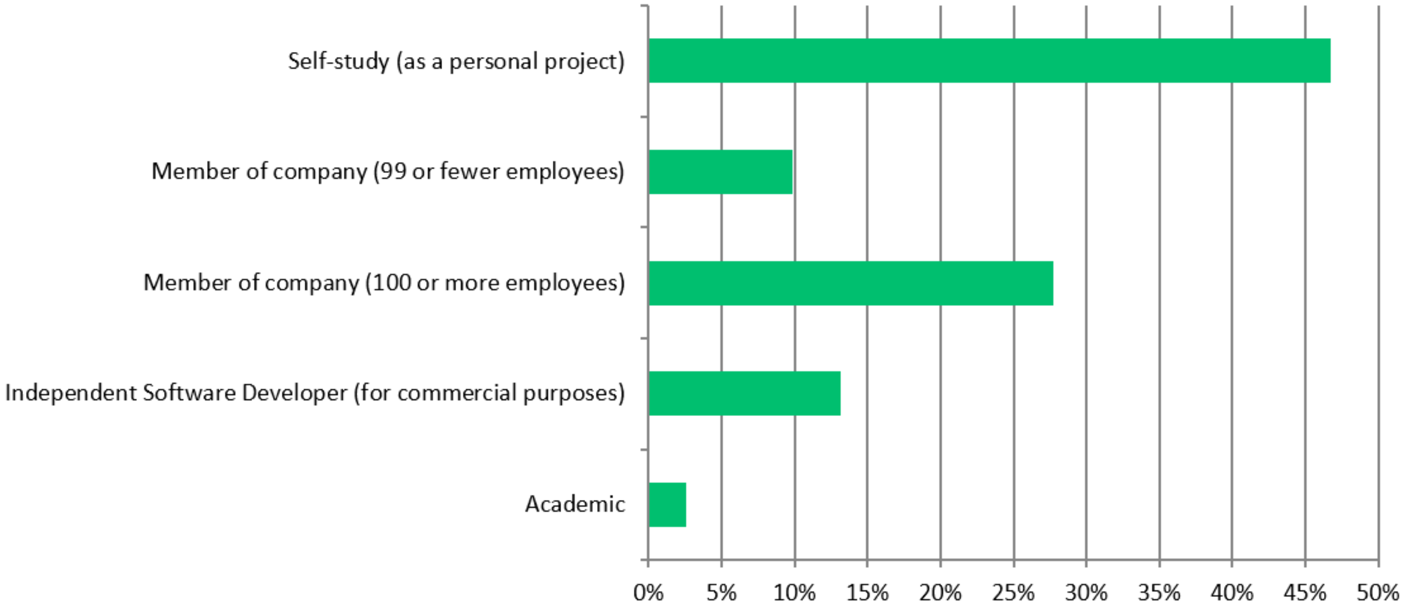
Feedback from this survey indicates that future action would be welcome in the following areas:

1. Validation layers
    a. Continue focus on validation layer performance initiative
    b. Continue github issue responsiveness and filling out coverage
    c. Would be nice if time permits: create tools to help developers debug their errors
    d. GPU-AV may need some focused attention for more functionality and addressing bugs
        i. LunarG Note: The refactor currently in progress to improve performance will touch much of the GPU-AV code. After that refactor is complete, additional functionality and bug fixing can occur.
    e. Investigate methods to create more readable validation layer error messages.
2. Evaluate potential SDK additions
    a. Windows 11 support (add to CI environments)
    b. GLFW
    c. iOS as a target (create Vulkan Loader for iOS) and fuller validation layer support
3. Vulkan Configurator
    a. As we design the next major revision on the Vulkan Configurator consider
        i. Ways to assist developers with layer issues
        ii. Ways to assist in debugging Vulkan installation and configuration issues
        iii. Improve the UI for more simplicity
4. Raise awareness in the Khronos WG for the need for more contributors and developers to quicken issue resolution and enhancements
    a. MoltenVK
    b. DXC and glslang
    c. RenderDoc

Survey Questions and Answers

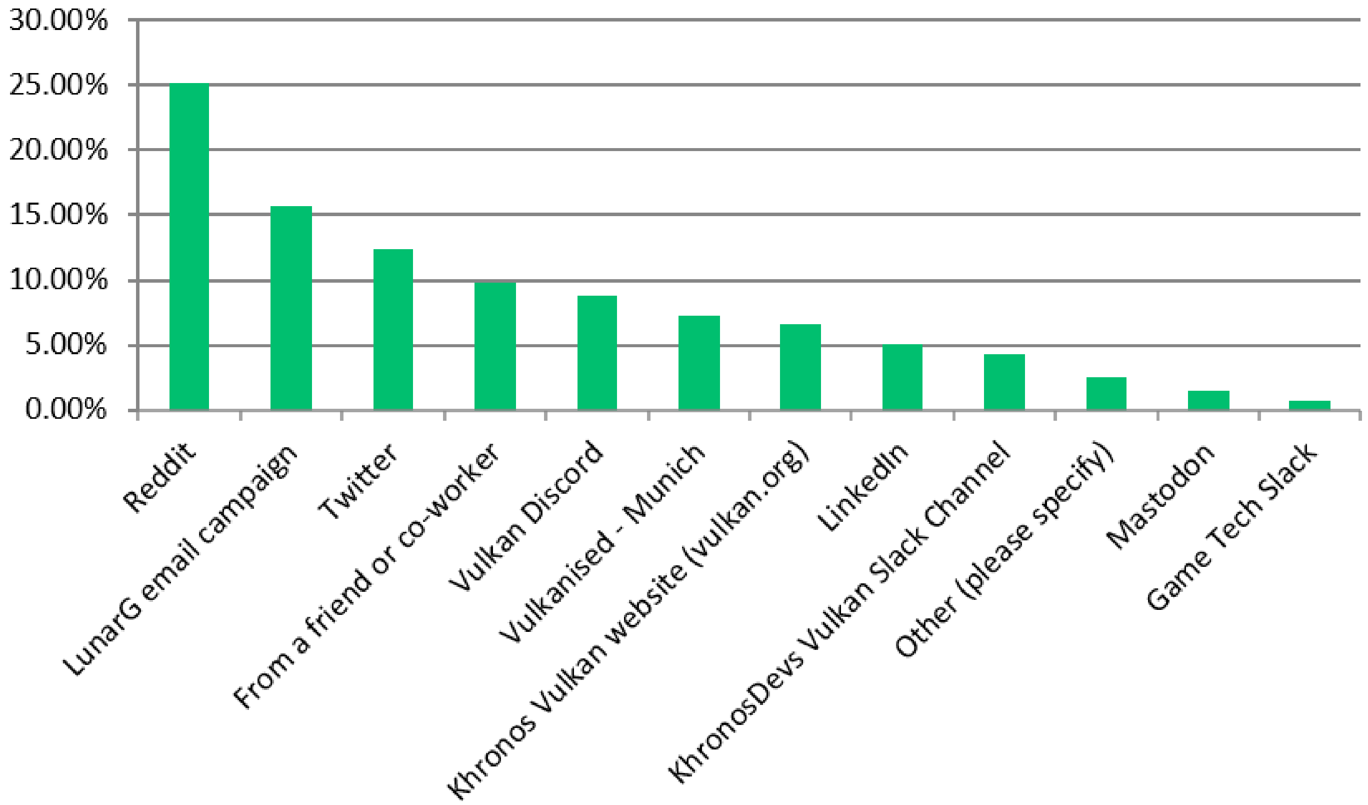
This section of the document lists each survey question and provides the feedback from the respondents.

What type of Vulkan developer are you?



50.7% of the respondents use Vulkan for commercial purposes. 49.3% of the respondents were self-studying Vulkan as part of a personal project or an Academic environment.

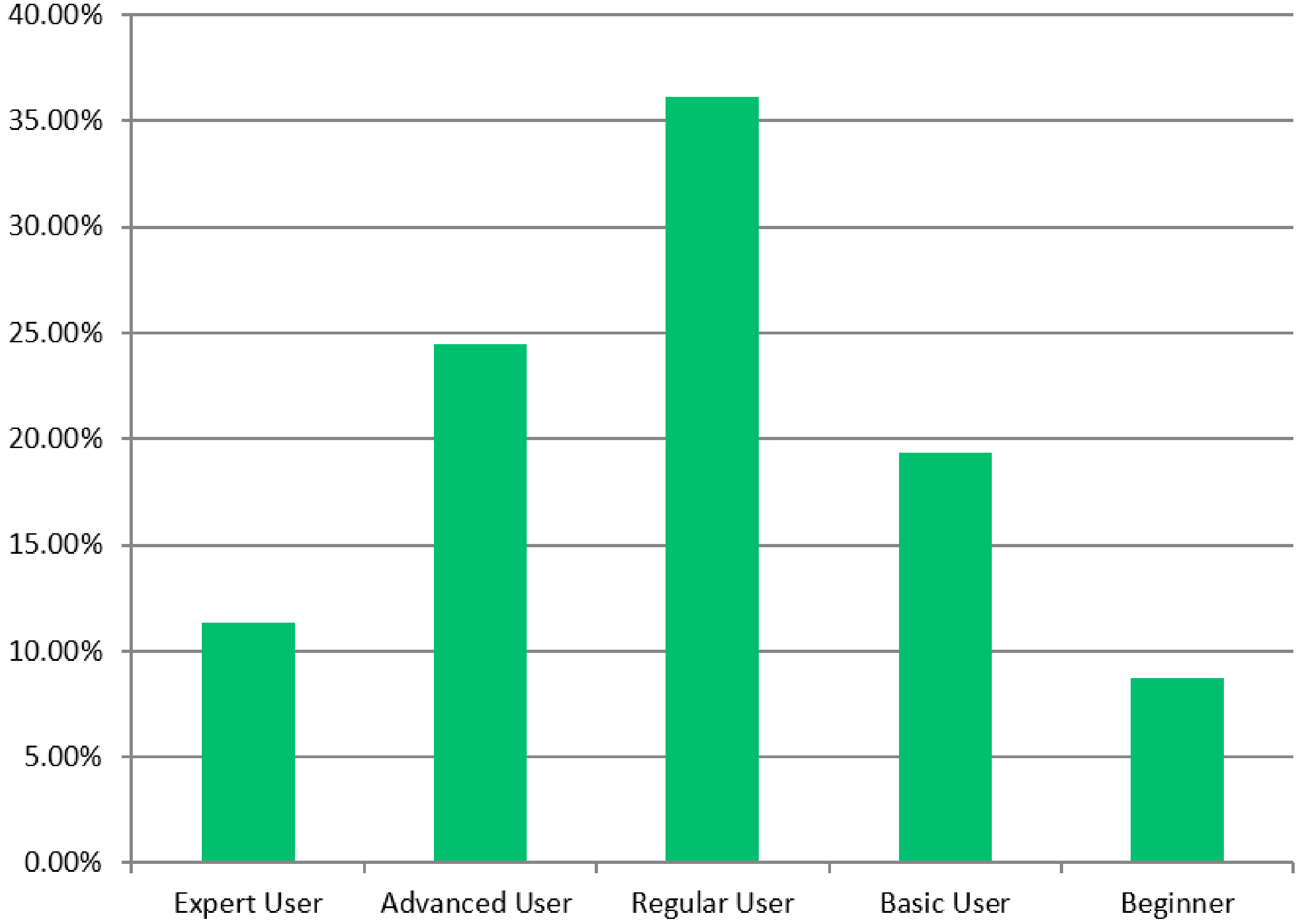
Where did you hear about this survey?



Items in the “Other” category included:

1. WeChat Group: Khronos China
2. Khronos newsletter
3. Khronos email campaign
4. forum
5. ON1

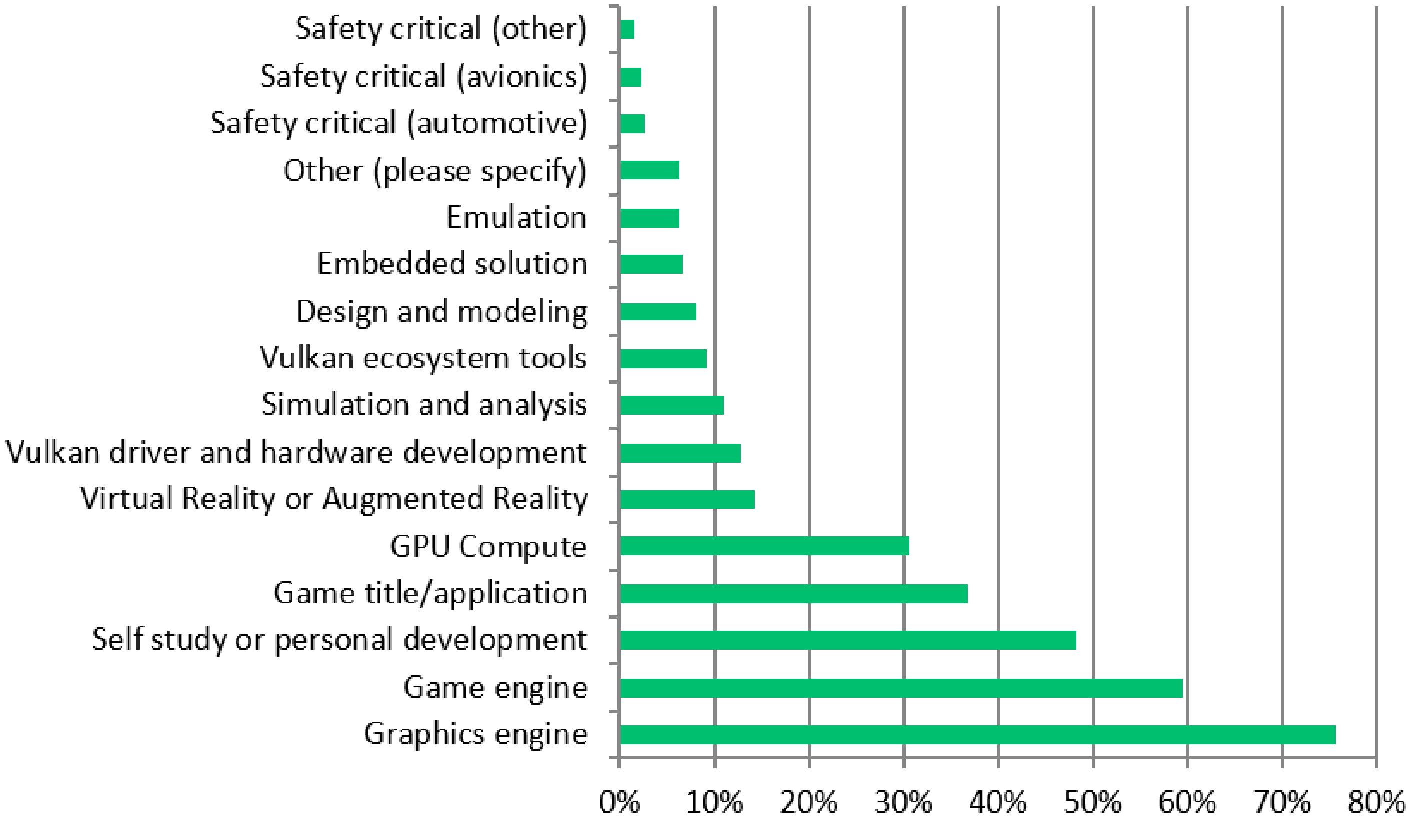
How experienced of a Vulkan Developer are you?



Definitions shown on the survey:

- Expert User- Strong knowledge of the spec/tools/ecosystem, deeply involved
- Advanced User - Does a lot of Vulkan work with good knowledge of the spec/tools/ecosystem
- Regular User - enough knowledge of spec and tools to be effective for required tasks
- Basic User - infrequent Vulkan work, some knowledge of spec and tools to allow for small tasks
- Beginner - just beginning to learn Vulkan

Your development is for what type of use case? Check all that apply:

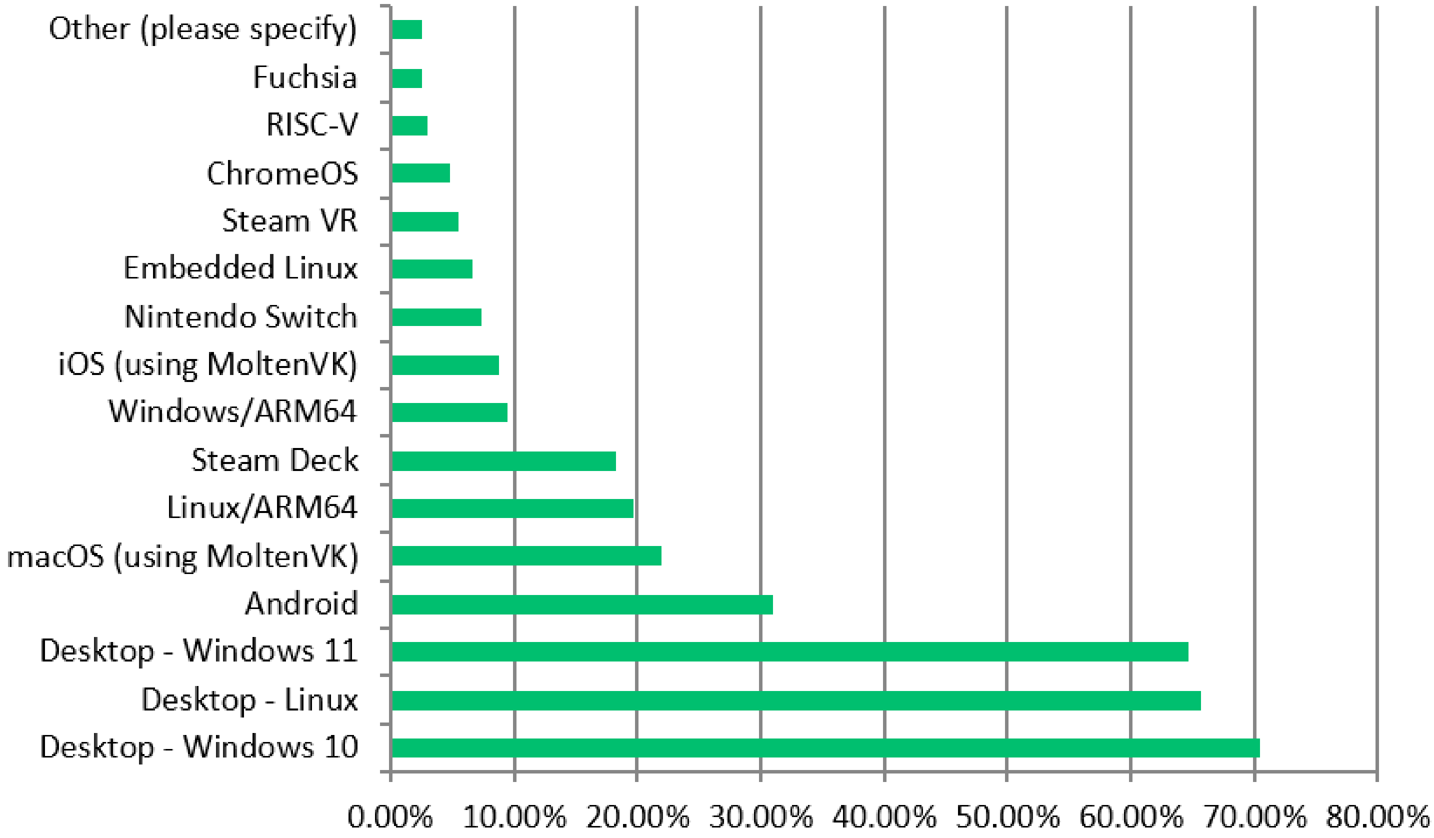


LunarG Note: Year over year, the top 8 categories have remained the same.

Items in the "Other" category:

1. GUI system
2. Safety critical (medical), Safety critical (dental)
3. VJ app
4. medicine
5. Path Tracing
6. Wayland compositor
7. virtualization
8. Light field
9. D3D translation
10. Desktop rendering (in the future)
11. Education
12. Research
13. Ray Tracing Physics
14. Scientific visualization

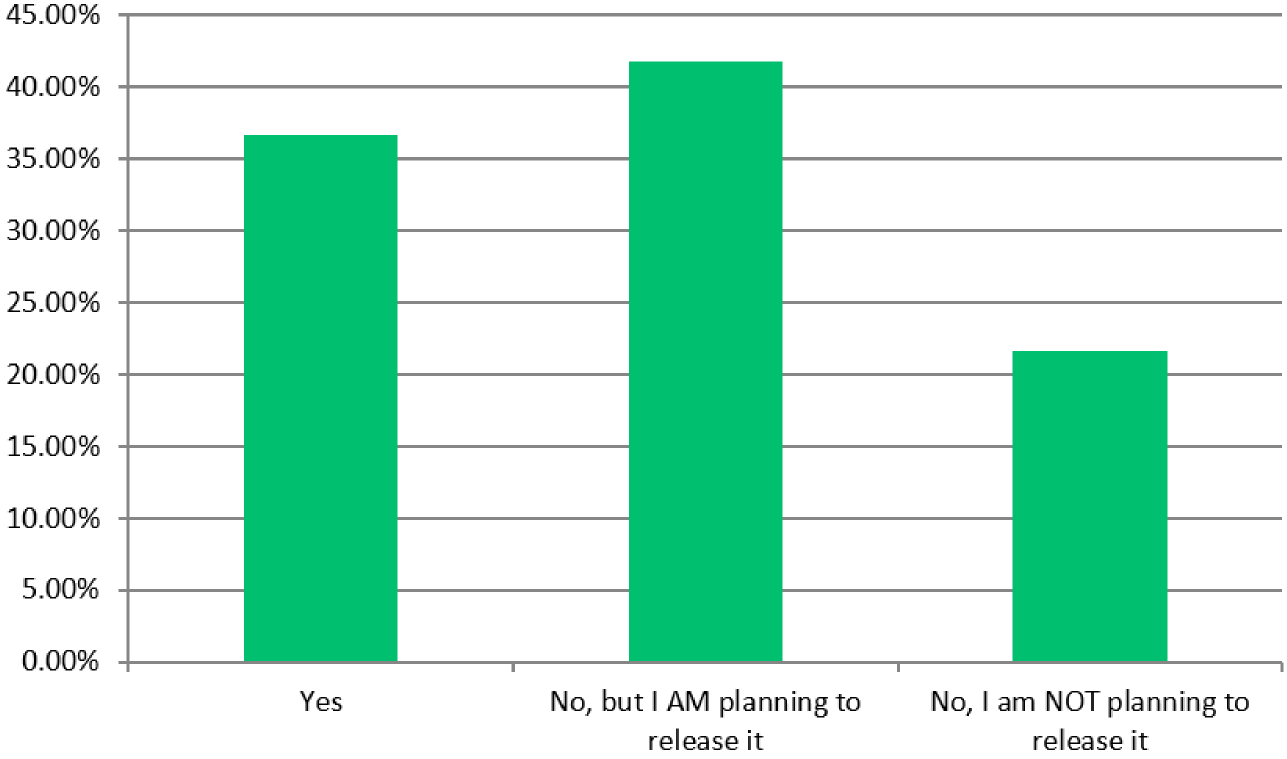
What is the target of your Vulkan application? Check all that apply:



LunarG notes:

1. **As suspected, Windows 11 is growing as a target for applications and as such we should probably add Windows 11 configurations to our CI environments.**
2. **Windows, Linux, Android, and MoltenVK remain the top 4 target platforms in the same ordering.**

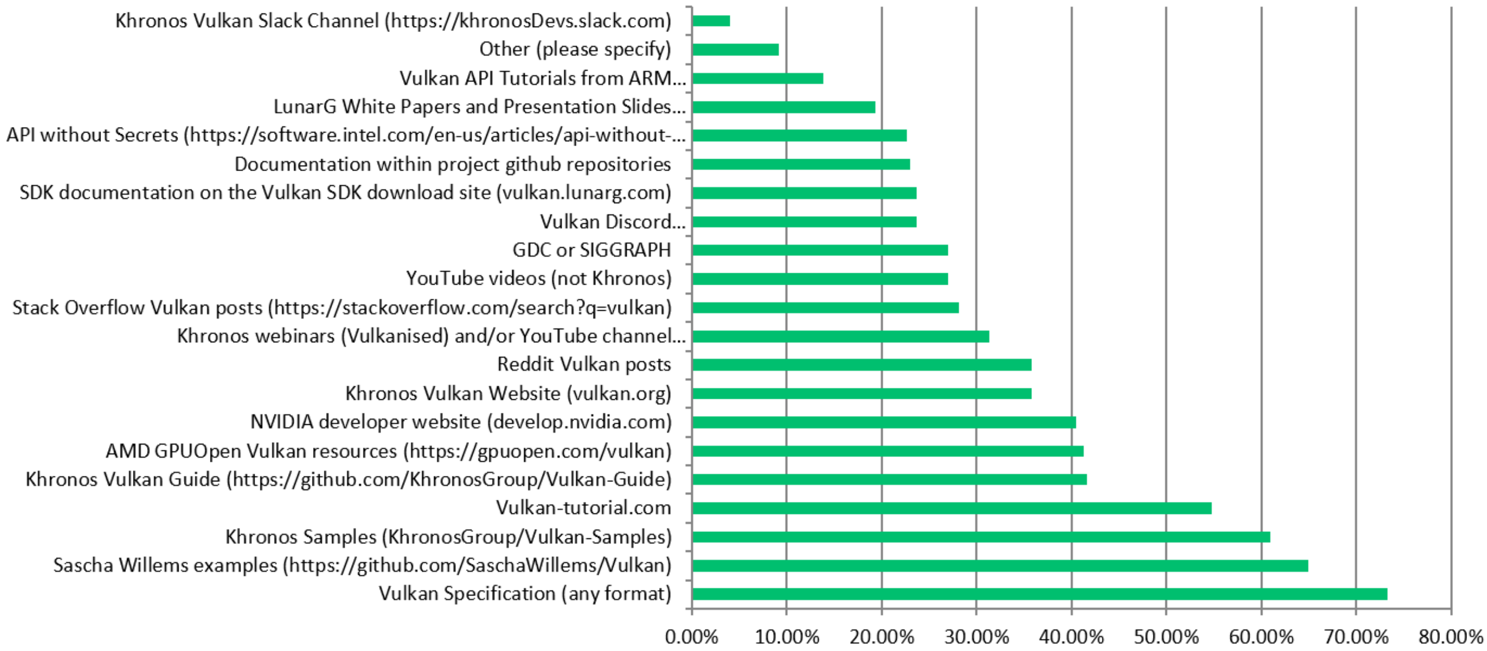
Have you released your Vulkan development project for public use?



LunarG Note:

1. **Compared to the previous year's survey's, the number of folks who have released their Vulkan application for public use continues to grow (25% to 28% to 36%).**

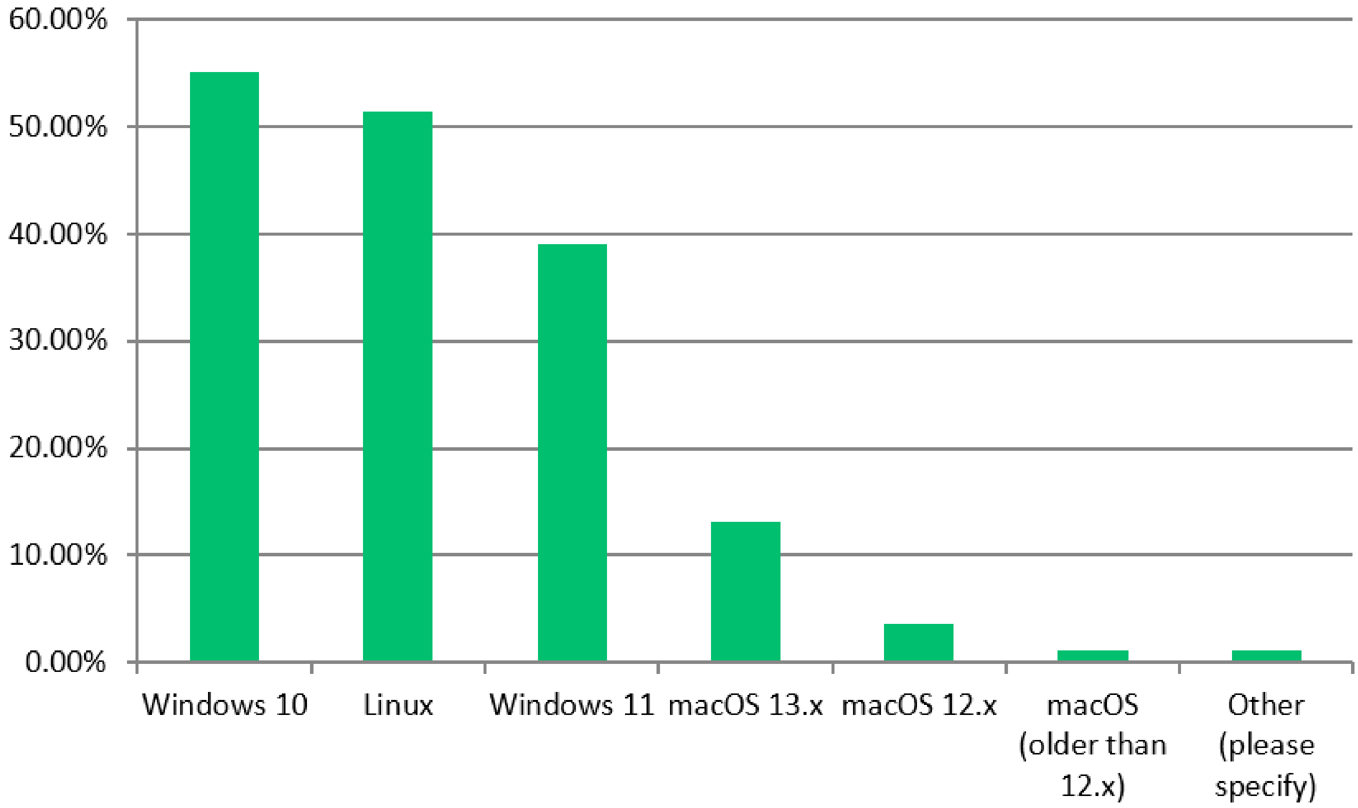
Which resources did you use to learn the Vulkan API? Check all that apply:



Other responses:

1. https://vkguide.dev/
2. NVidia Vulkan samples (nvpro-samples)
3. Lunarg sample
4. books
5. Any information
6. Godot
7. Vulkan Programming Guide by Graham Sellers
8. This lecture series: https://www.youtube.com/watch?v=tLwbj9qys18
9. Qt Examples about Vulkan
10. https://vkguide.dev/
11. vkguide.com
12. vkguide.dev
13. vkguide.dev website
14. taichi
15. Too long story
16. Vulkan Validation Layers
17. Games Tech Slack
18. books
19. vkdev.guide
20. https://www.youtube.com/@BrendanGalea
21. vkguide.dev
22. I don't know
23. Vkguide.dev

What is your development environment? Check all that apply:



What Vulkan Samples would be most useful for Khronos to add in the future?

Note: underlined items already exist, *italicized items are WIP*

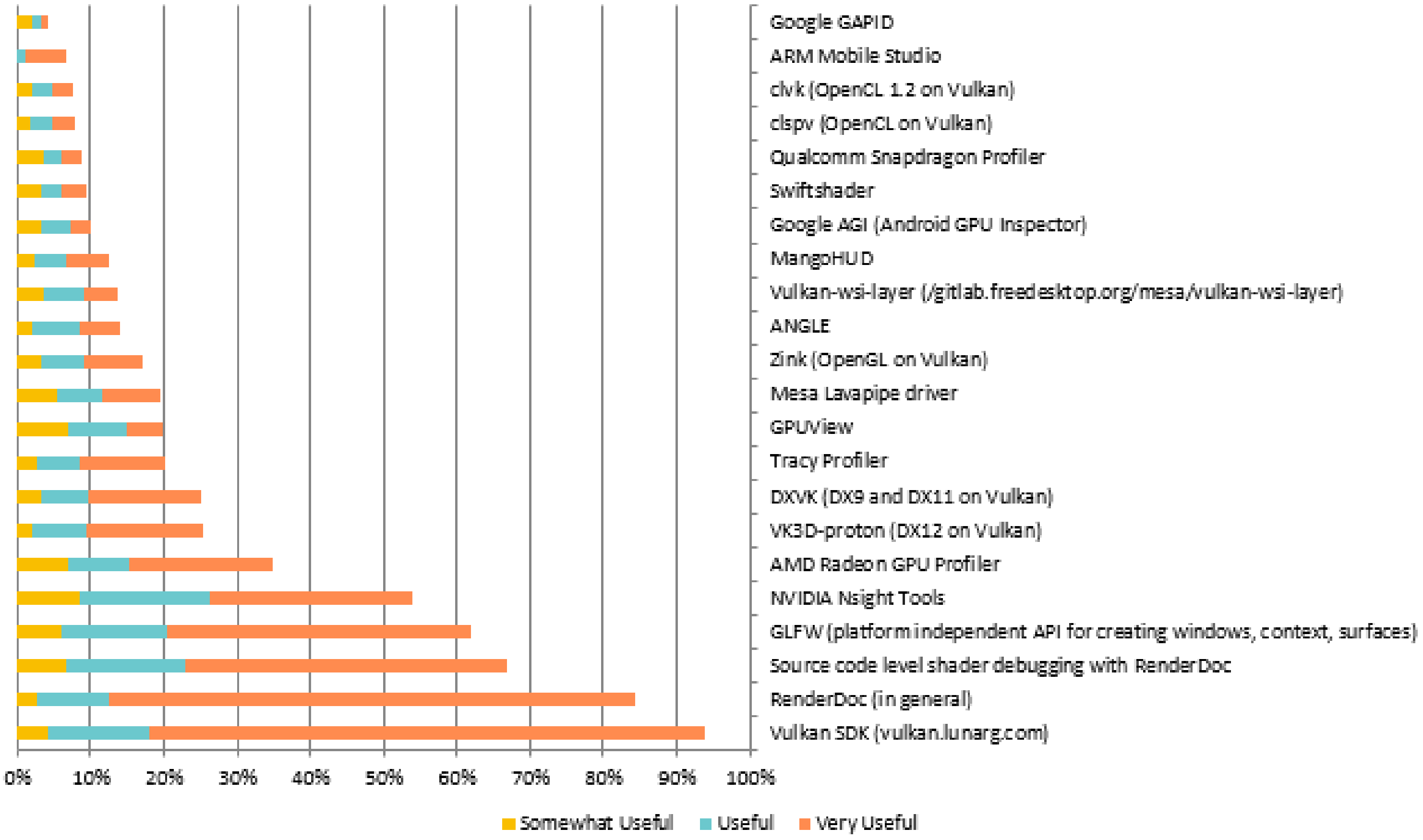
1. Samples focused on performance and soundness best practices.
2. *How to use timestamp queries properly.*
3. A sample on VK_EXT_descriptor_buffer
4. Some wrong or bad usage samples of Vulkan are also useful. Learning from failures maybe better than learning from success. Comparing the wrong or bad usages with the correct or good usages will help to master the correct the usages.
5. Examples about how to detect the performance bottle neck and how to fix them and get the improved results.
6. Frame pacing / present timing
7. Ray Tracing, Pipeline cache, Mesh Shading
8. Vulkan SC

9. I have never used Vulkan samples as learning resources, but maybe:
    a. How to use VK_EXT_descriptor_buffer properly
    b. How to expand ""render pass"" over multiple command buffers with VK_EXT_dynamic_rendering
10. Multi Copy Engine Synchronization with Main Rendering
11. Vulkan Samples using VMA, VulkanHpp and best practices additional to the ""Raw""-Vulkan samples.
12. More samples about timeline synchronization. etc."
13. macos example using headers from the sdk + loader + VK_EXT_metal_objects + VK_MVK_moltenvk extensions. (the latter one is written as extension but actually isn't ... apparently?)
14. Compute shaders
15. Video decoding and encoding, "bindless" drawing
16. More specific DirectX->Vulkan conversion samples.
17. video encode/decode
18. *mesh shader*
19. Handling Renderpasses and Subpasses
20. Bindless Descriptorsets
21. Performance oriented samples for desktop apps
22. OpenGL-Vulkan interop
23. Signed Distance Field Based Global Illumination
24. Swapchain recreation for embedded systems
25. Not sure
26. Deferred renderer, Gpu based rendering and async compute
27. More rtx features/samples
28. VK_EXT_descriptor_buffer
29. Subgroup operations (I don't think there was a sample for this)
30. Sparse images and sampler feedback
31. YUV (with and without the sampler conversion ext)
32. In general more tutorials on Vulkan compute
33. I think examples that showcase popular extensions would be beneficial, especially for beginners
34. Use of video encoding and decoding
35. VK_EXT_DESCRIPTOR_BUFFER
36. Better video samples that don't require Nvidia proprietary dlls
37. An abstraction that leverages SM6.6 style bindless
38. Vulkan SC, Vulkan Video
39. VK_EXT_mesh_shader with HLSL!
40. using VK_GOOGLE_hlsl_functionality1
41. using VK_GOOGLE_user_type
42. Sparse resources
43. More synchronisation examples and use cases for performance enhancements (other than synchronisation for necessity) - outlining where synch could degrade instead of enhance.
44. Best practices
45. *OpenXR + Vulkan + OpenCL integration*

46. A larger example application using vulkan_raii.hpp that combines all of the current samples and shows how to e.g. render multiple textured models with varying shaders/descriptor sets in a basic render loop (with swapchain, image acquisition, etc.), preferably in as compact code as possible and with minimal references to external files (like the confusing vk::su namespace). Such an example would be very useful as a reference for beginners who haven't yet fully figured out how all of the different parts of the API fit together, and would also serve as a great sanity check for more experienced developers who want to make sure they are using various features in the way the API designers intended. It would be extra useful if the sample included solutions to some common problems such as how to handle window resizing.
47. Don't know.
48. Proper SI usage.
49. Android OpenXR interop
50. Texture and Mesh Streaming
51. Ray tracing, gpu compute/NN acceleration
52. Samples (or even better: one big sample) that show(s) how to design an efficient and (easily) expandable Vulkan renderer.
53. Something about raytracing for both real-time and offline rendering.
54. Instanced rendering
55. Multipass rendering
56. Usage of HDR
57. Usage of Gbuffer for deferred rendering
58. Setup of piepeline for advanced lighting of many objects
59. Timeline semaphores galore
60. *Swapchain modes and synchronization*
61. Windows resize handling best practices
62. Compiling code using shader_c with #include support or an example using dxc for HLSL raytracing on windows.
63. *Mesh shaders*
64. Lossless GPU (de)compression (E.g. VK_NV_memory_decompression)
65. Anything and everything. all of the time 😂 maybe some optimisations examples would be nice
66. Terrain rendering, more detailed shadow mapping, frustum culling, how to pass array of sampler2D to shader
67. Updating examples with primary usage of newest extensions.
68. Mesh shaders, using VMA efficiently
69. read write texture
70. Synchronization
71. Sample showing how to bind different resources at different frequencies per-frame using different descriptor-set layouts as described in this post: https://developer.nvidia.com/vulkan-shader-resource-binding
72. If there alreafy exists one I couldn't find it.
73. Specifically going from 1 textured square to 2 I found surprisingly difficult and confusing.
74. Gaming
75. FreeSync and minimal latency app - how to render with minimal frame latency, possibly starting shortly before frame deadline; using the FreeSync for low frame latency
76. How to develop minimalist Vulkan application. (The API is incredibly complicated)
77. sample about using different queues, and when using more queues actually hurt performance

78. sample about performance difference between, normal descriptor use, using a mindless model, and using a bindless model using VK_EXT_descriptor_buffer
79. sample about 8/16 bit int usage in shader, not only floats"
80. Idk
81. VK_EXT_descriptor_buffer
82. Suprise me!
83. Complete app able to display data driven scenes that illustrates performant techniques that would scale well in real-world use. Right now it's difficult to surmise from to samples which alternative features are worth optimizing for, and which wouldn't provide a discernible benefit.
84. basic usage examples of:
85. task shaders, mesh shaders
86. An overview of a good practices how to structure a framegraph-based renderer
87. Perfomance and optimisation
88. Video decoding/encoding
89. I haven’t seen all that are currently published, so I don’t know what could be missing and added yet.
90. More glslang doc and exemples
91. *More stuff with Vulkan hpp please!*
92. How to manage VkImage Layout in runtime.
93. More samples about multithreading
94. How to instal it use in ON1 plugins for photoshop
95. Recommendations for what features to use and not use with revisions (like 1.0 to 1.1 etc).

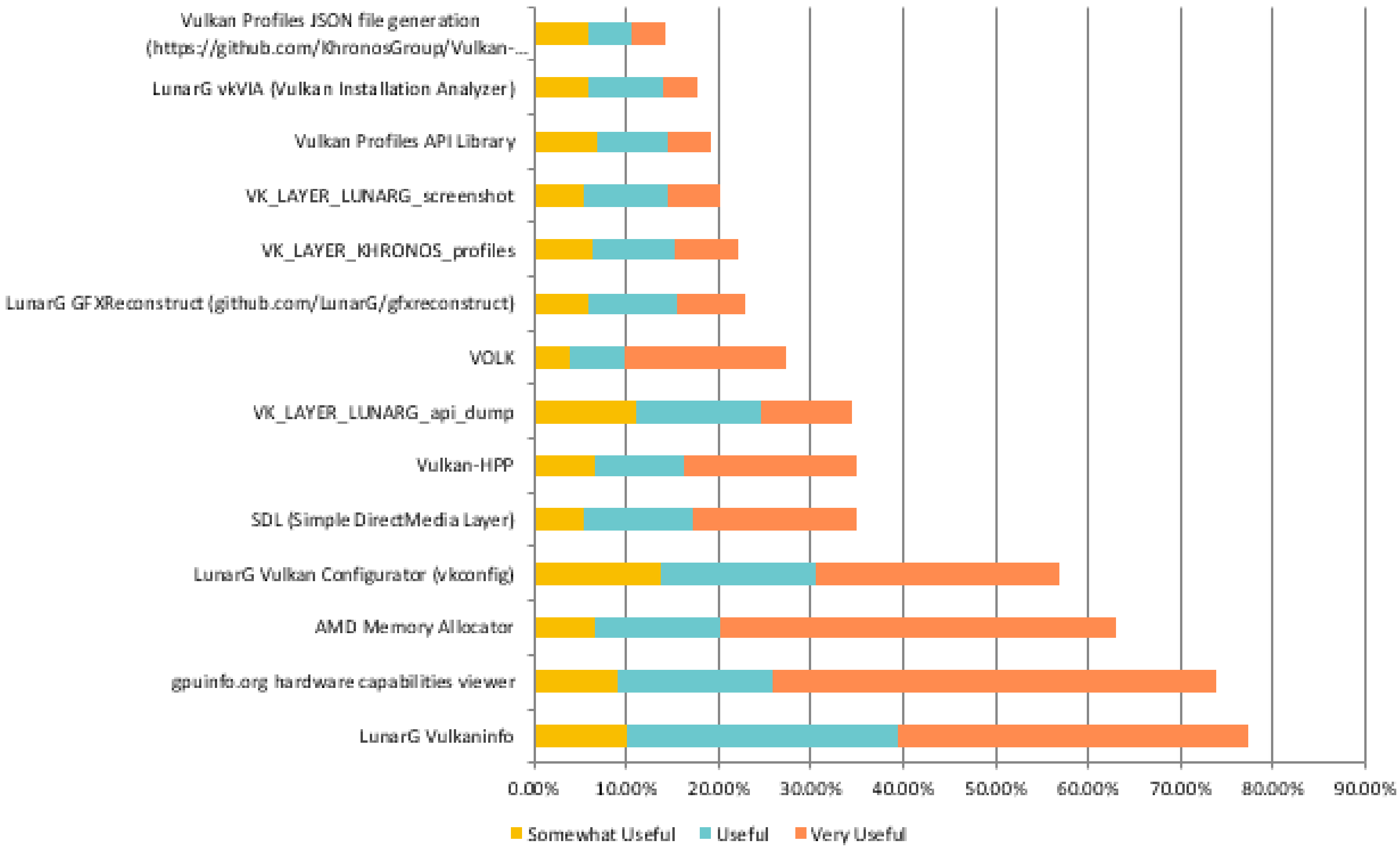
Indicate the usefulness of the following tools and/or layers:



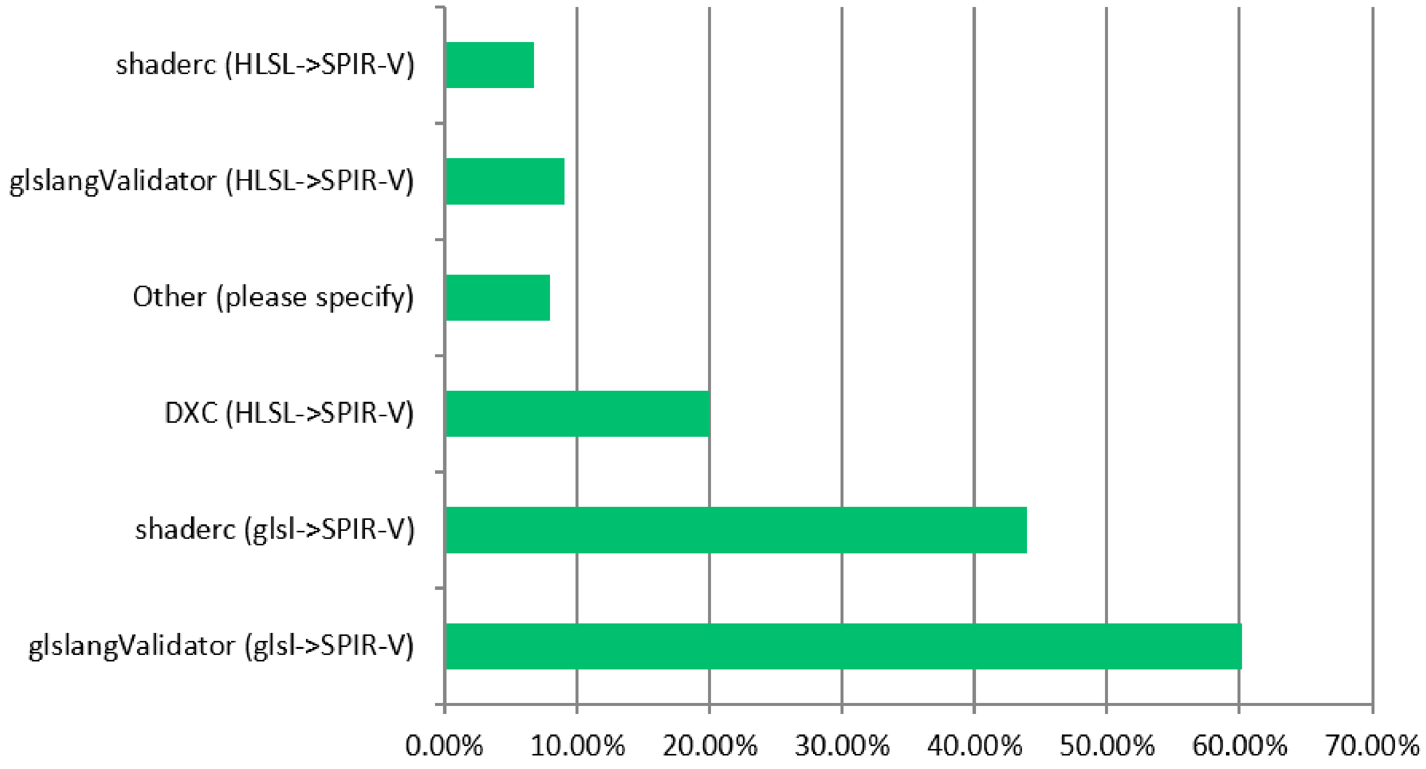
LunarG note: It is satisfying that the Vulkan SDK is the top hitter. It is also satisfying that the source code level shader debugging with RenderDoc was a top hit. Especially since the pieces for that functionality (glslang, spirv-tools, DXC changes, renderDoc changes) just released last fall. That is proof that it was a highly needed feature.

We may need to consider adding GLFW (in addition to SDL) to the SDK.

For the Vulkan ecosystem tools and layers (also INCLUDED in the LunarG Vulkan SDK) listed below, indicate their usefulness:



What is your front end for creating SPIR-V? Check all that apply:

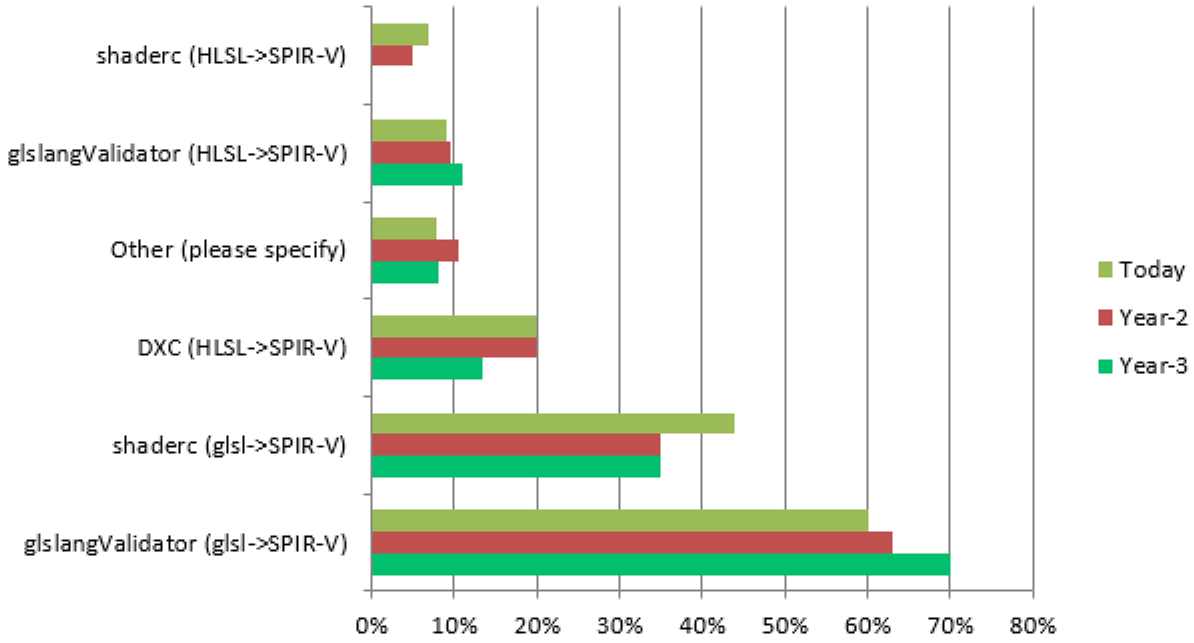


Other category:

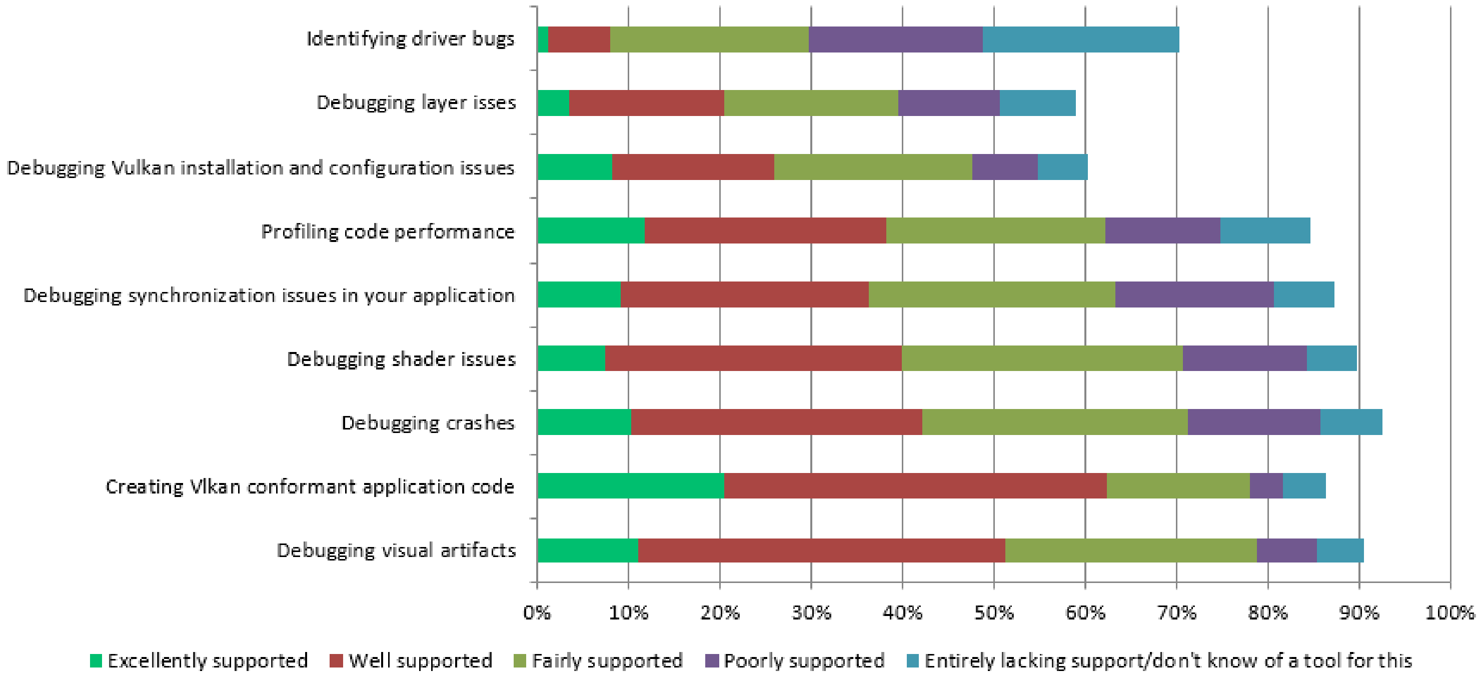
1. naga https://github.com/gfx-rs/naga
2. Would like an easier to use api than shaderc as it can be very confusing at times
3. i don't know
4. glslc
5. Slang API (slang -> SPIR-V)
6. I am the author of: https://github.com/hugobros3/shady
7. Rust-Gpu and custom code generator
8. naga
9. ShaderWriter (C++->SPIR-V)
10. ON1
11. slangc (Slang -> SPIR-V)
12. Custom library using glslang (glsl->SPIR-V)
13. Tint (WGSL -> SPIRV)
14. ANGLE (glsl->SPIR-V)
15. spirv-tools for linking
16. Somtimes handwritten, or bytecode generated from C
17. We use our own SPIR-V generation code.
18. Rust-gpu
19. I wan't to remove GLSL at all form my project, after intergation DXC compiler for Android
20. for Unix/Linux/windows I'm use DXC compiler only. We have no any reason to support legacy GLSL at all.
21. Own compiler based on libshaderc

22. Custom DXIL/DXBC->SPIR-V translation
23. Custom Shader Pipeline

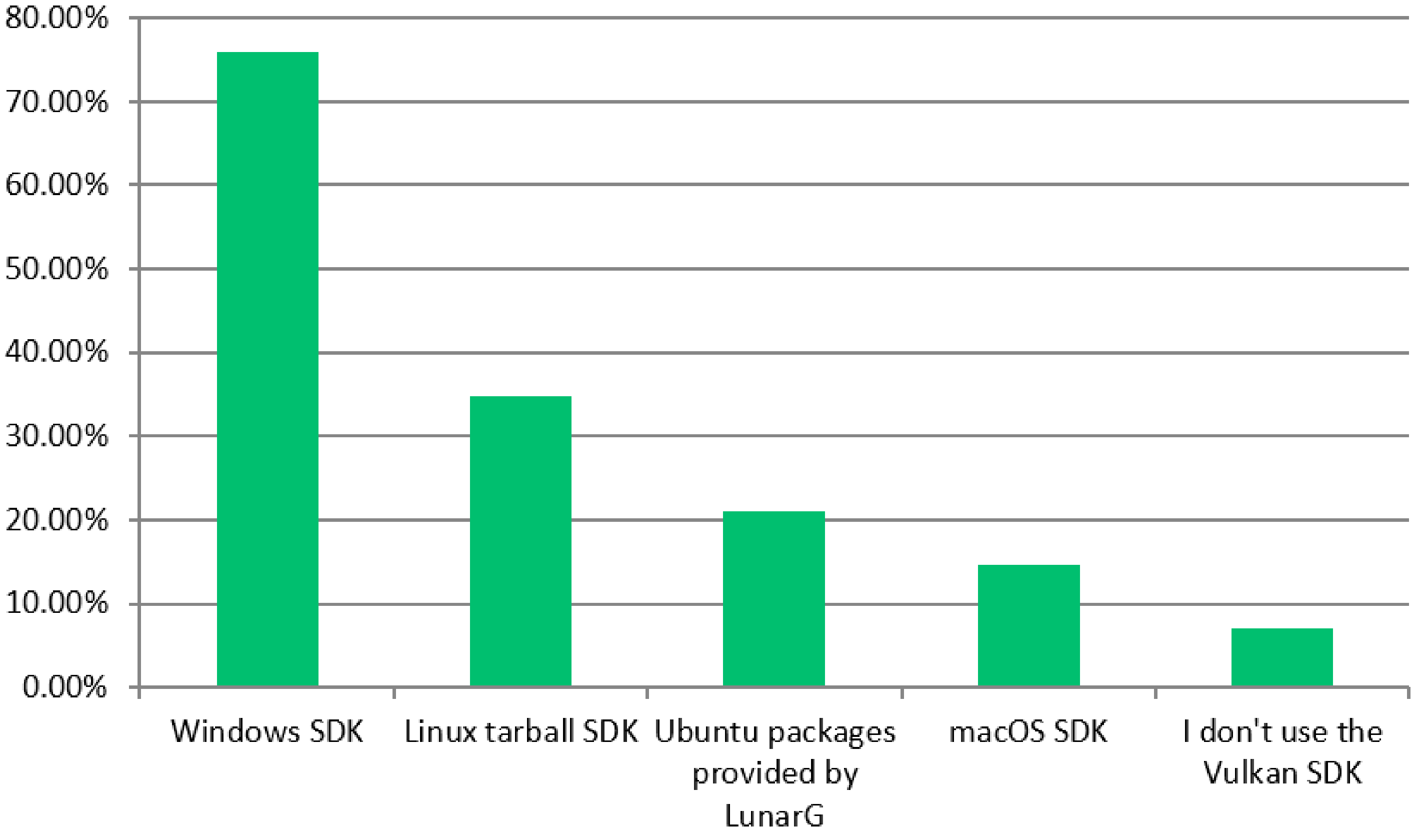
LunarG Note: Here is the trend over the last few years: glslangValidator had a slight drop, but it appears that those users may have moved to shaderc instead.



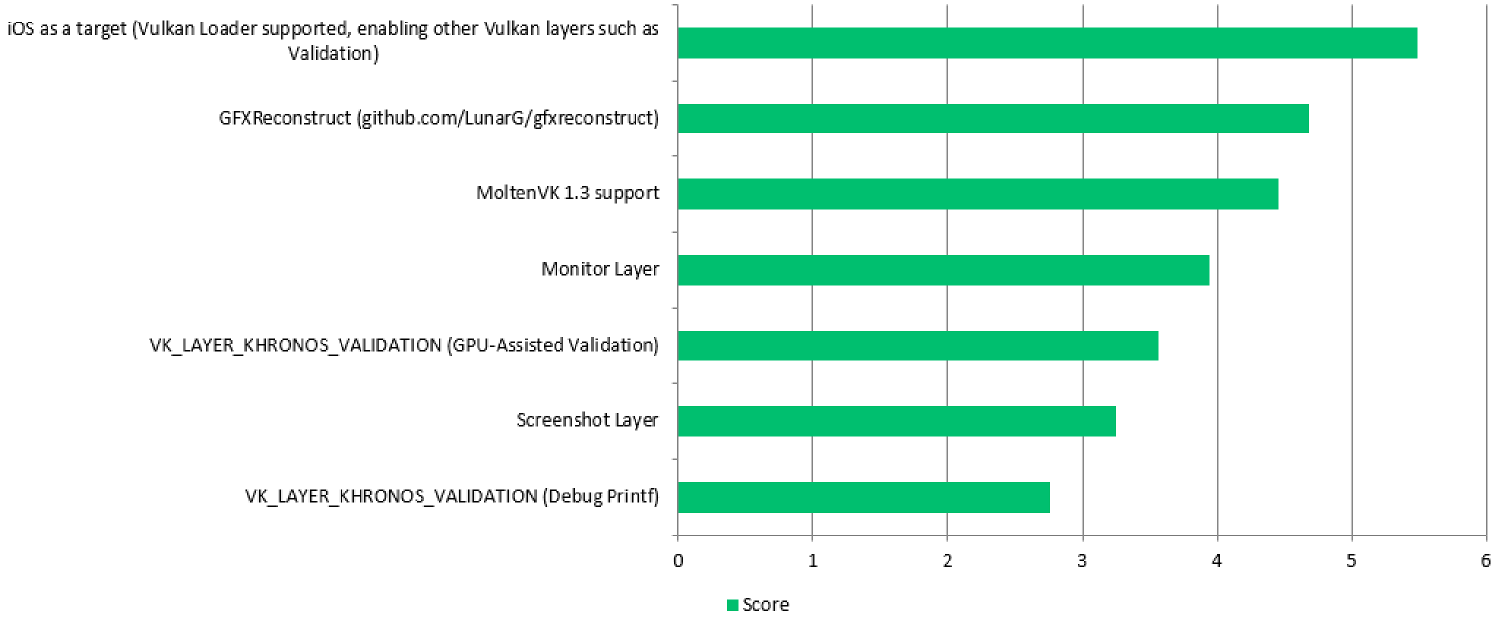
With the ecosystem tools and layers you are using, rate the ease of performing the following tasks:



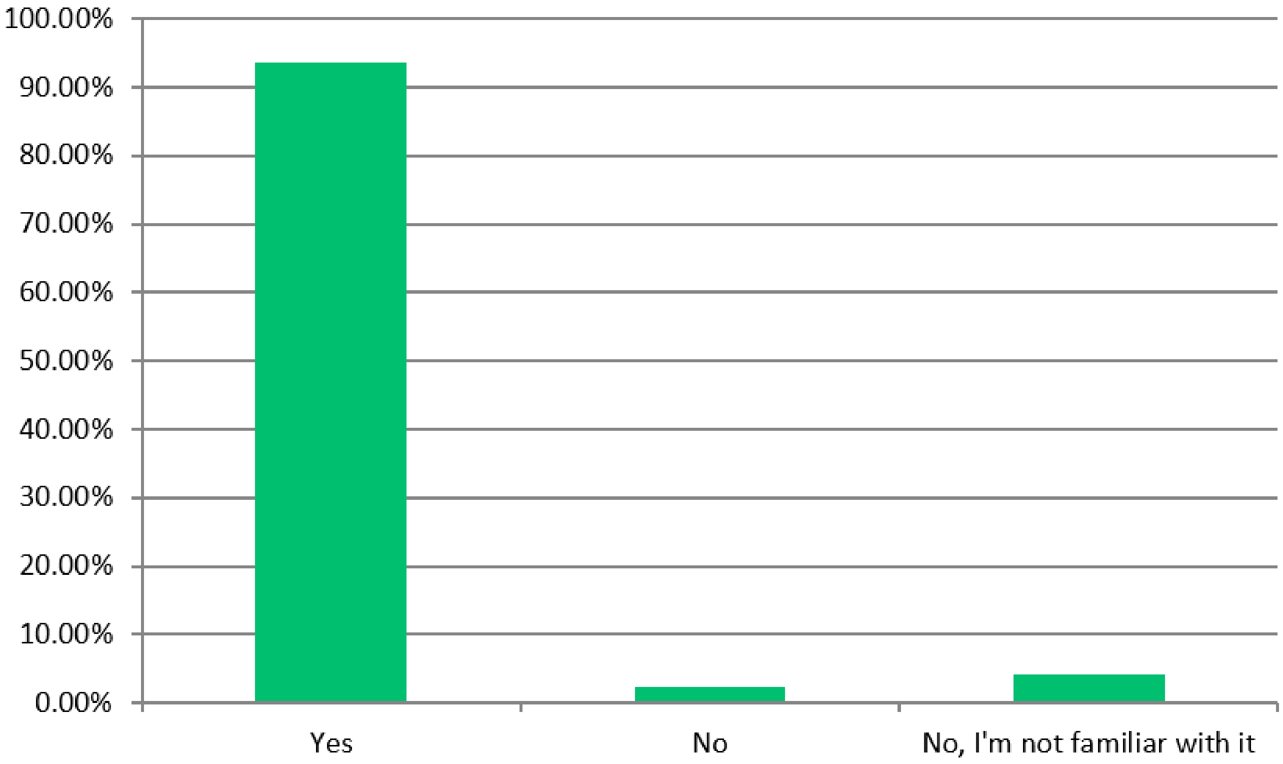
Which Vulkan SDK do you use? Check all that apply:



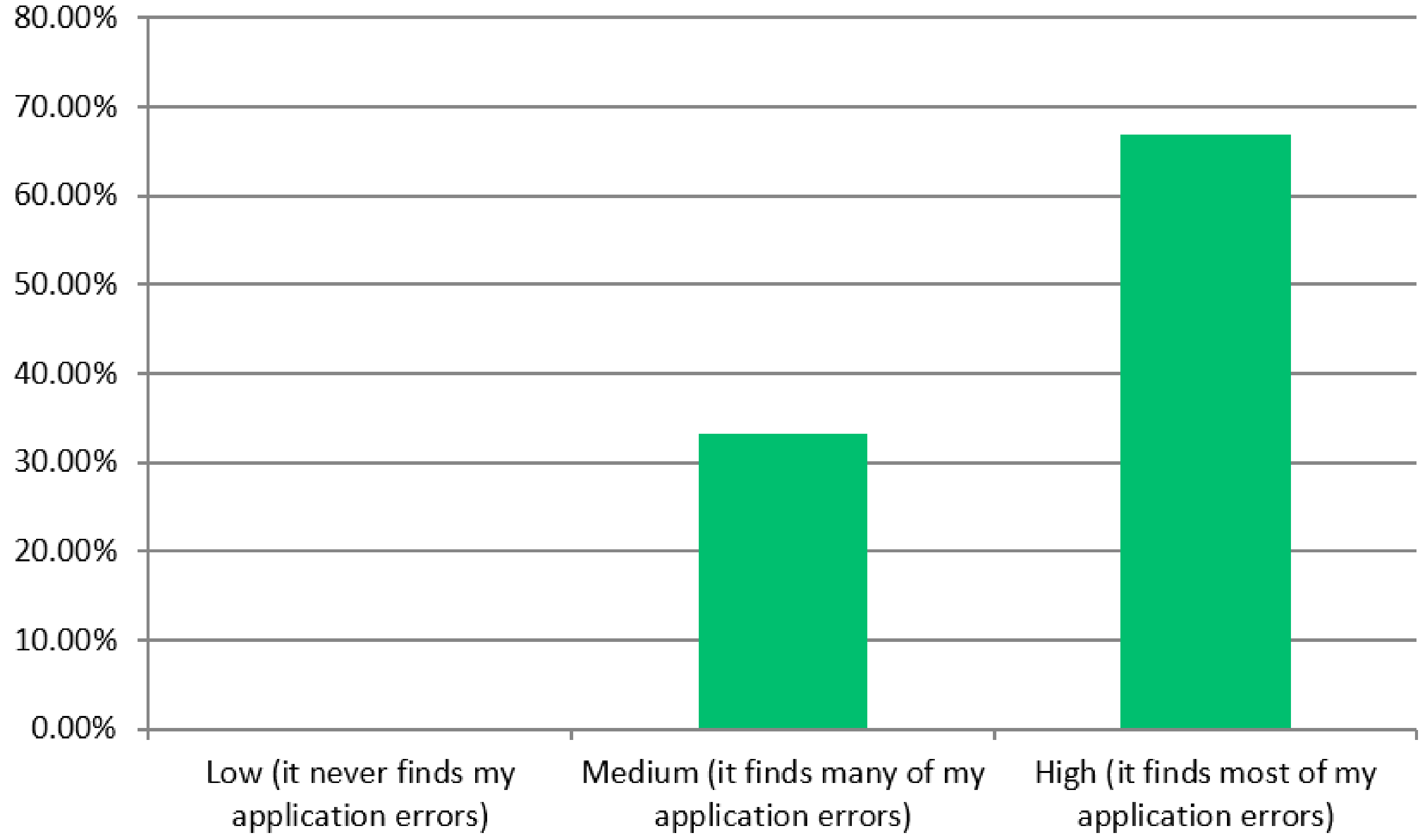
If you use the macOS SDK, rank the importance of the following layers/tools that could be added to the SDK. (1 is most important)



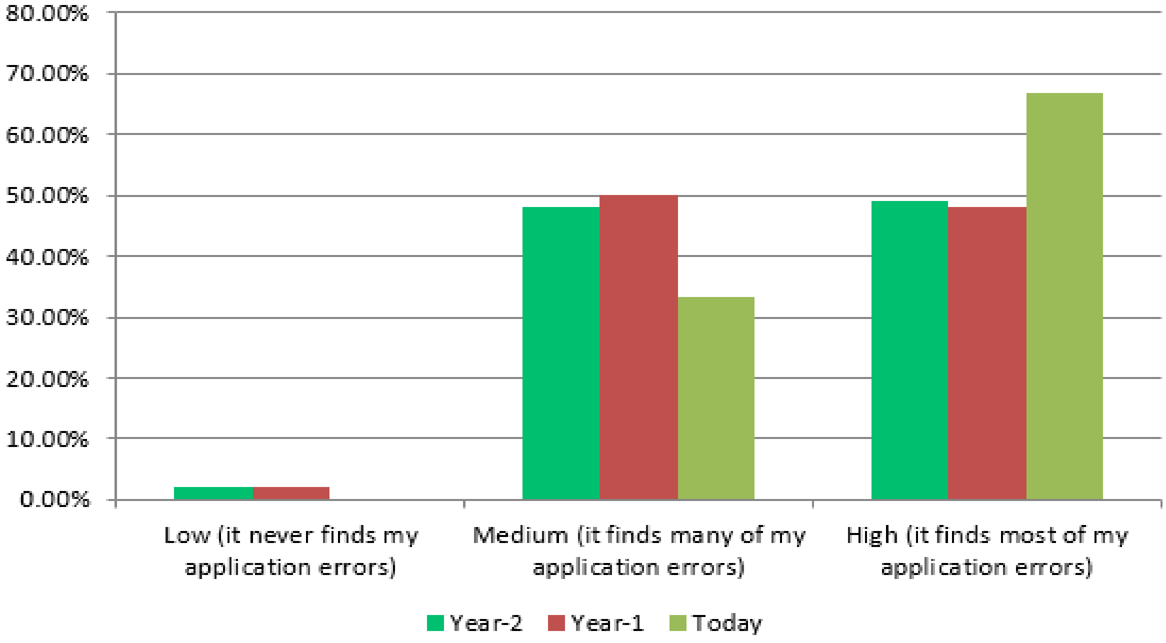
Do you use the Khronos Vulkan Validation Layer (VK_LAYER_KHRONOS_validation)?



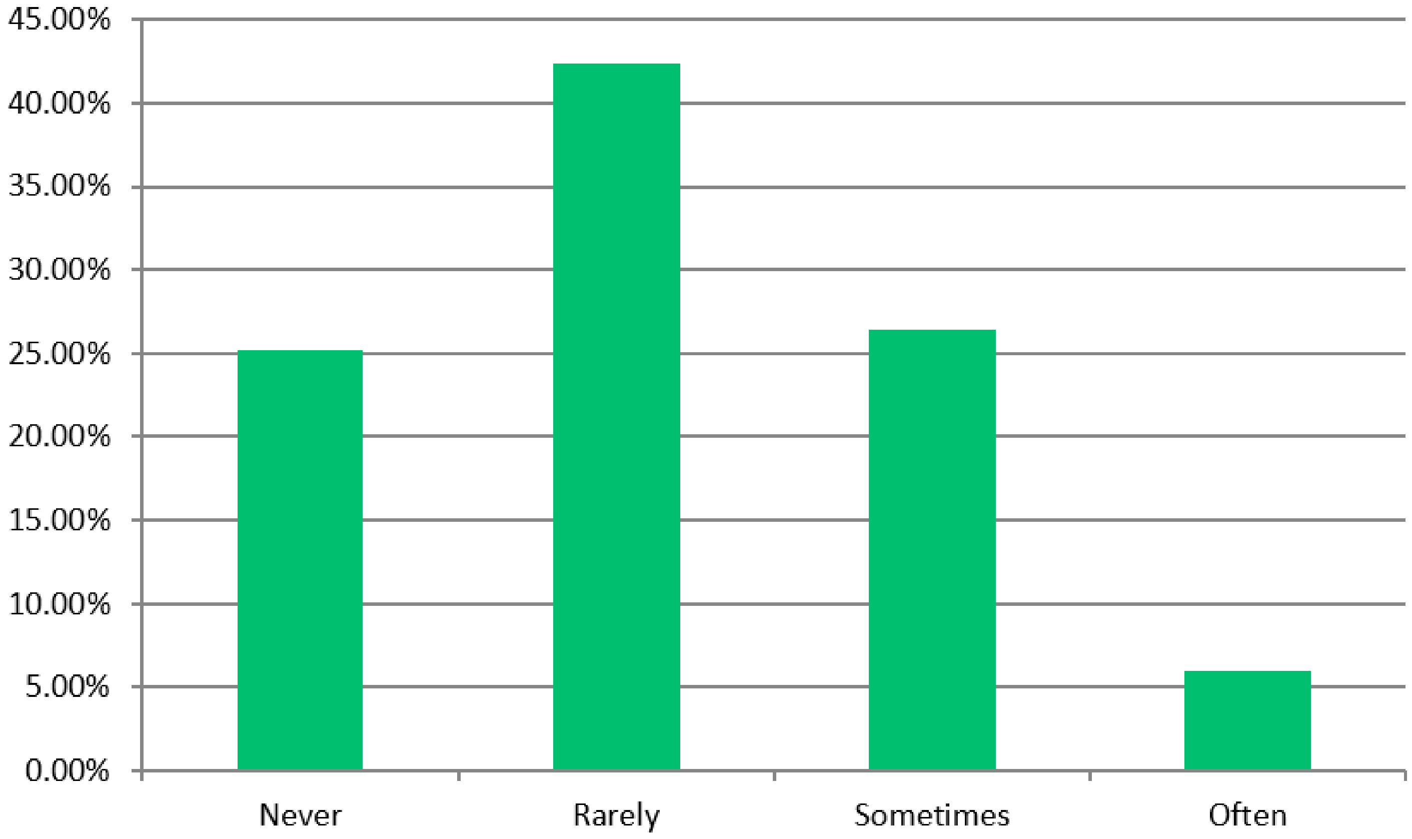
How would you rate the completeness of validation layer coverage?



LunarG Note: Comparing this question over the last few years, we are making good progress:



How often does the performance of the validation layers inhibit effective use of them?

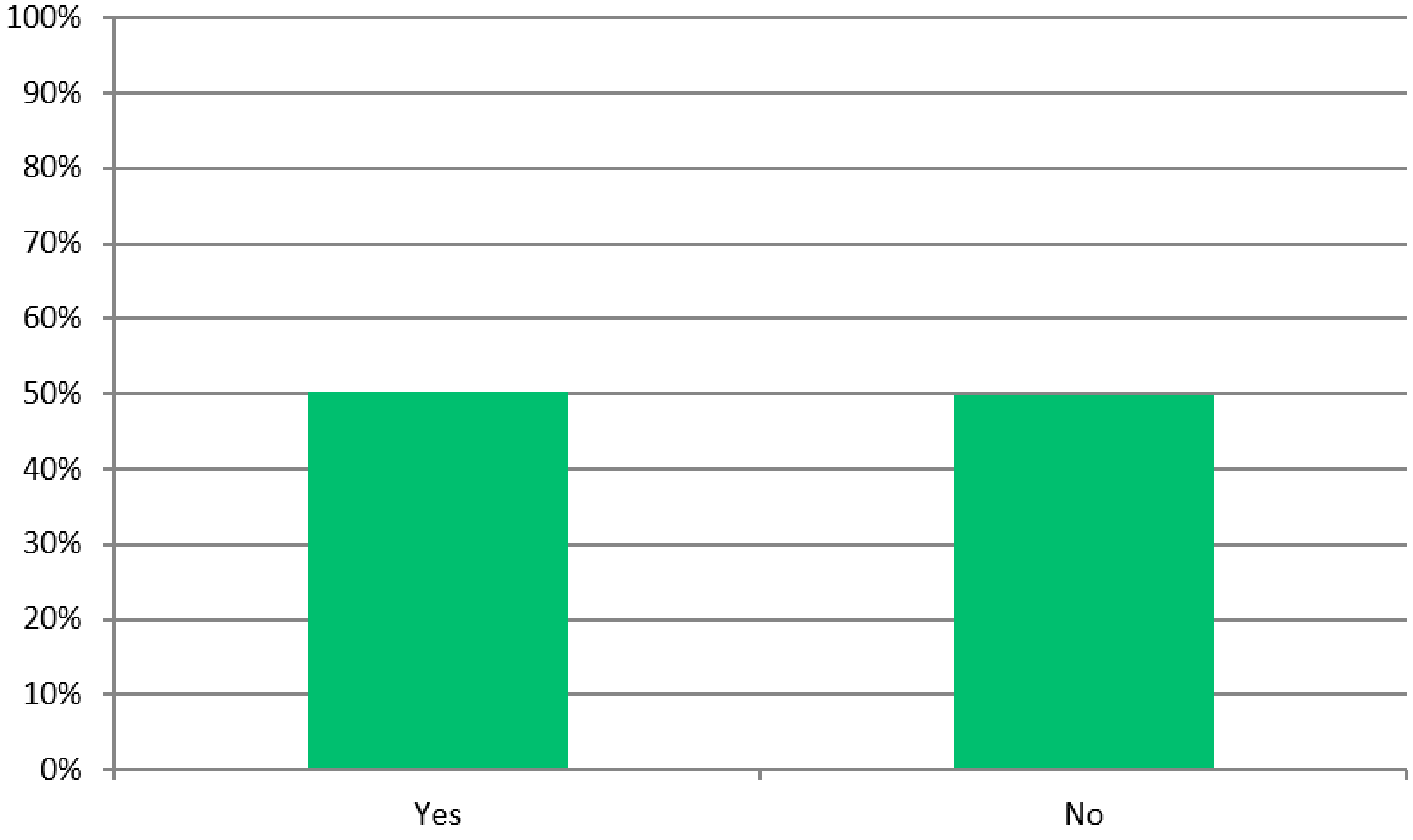


If you specified often, could you indicate an application or use case for us?

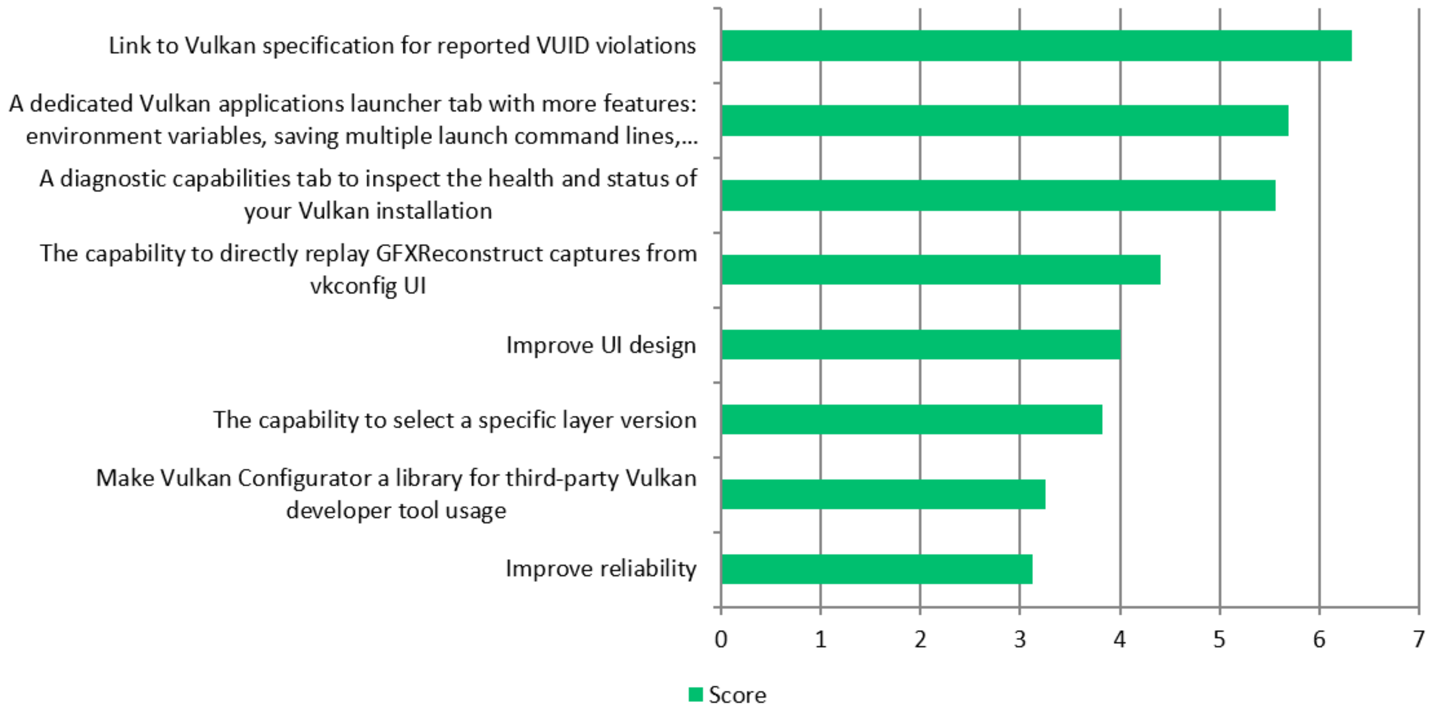
1. I can only run the validation for a few seconds of frames. It has very low performance due to log writing.
2. Just anything, it slows down a lot
3. It substantially reduces performance but only when validation layers are enabled. Obviously in production environment they are not.
4. Very slow
5. render subpasses (was implemented by coworker, can't specify more detailed...)
6. Very useful for debugging and developing, but not for production
7. Found multiple memory leaks over the years, which made us lose a lot of time thinking it was our application that was leaking memory.
8. modern steam games with gpu validation
9. GPU validation...
10. Running our games and/or game editor.

LunarG note: LunarG was expecting more respondents to indicate “Often.” However the folks in the often category are most likely using bindless features with large numbers of descriptors where there are significant performance issues (currently being worked on to make improvements) and the larger population wasn’t using bindless techniques.

Are you familiar with the Vulkan Configurator (vkconfig)?



Rank the following vkconfig improvement areas (1 is most important)



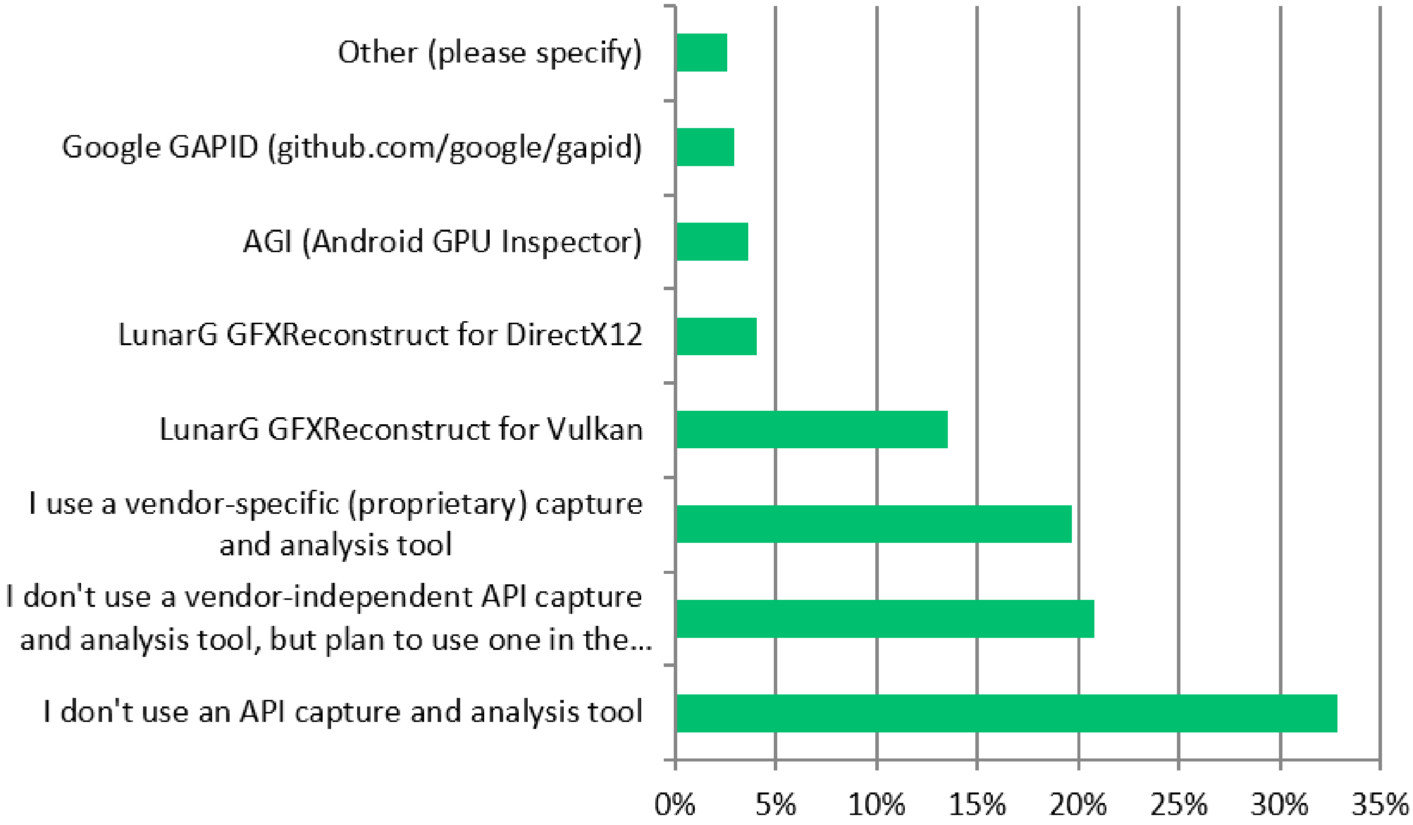
.

Do you have additional suggestions for the improvement of the Vulkan Configurator?

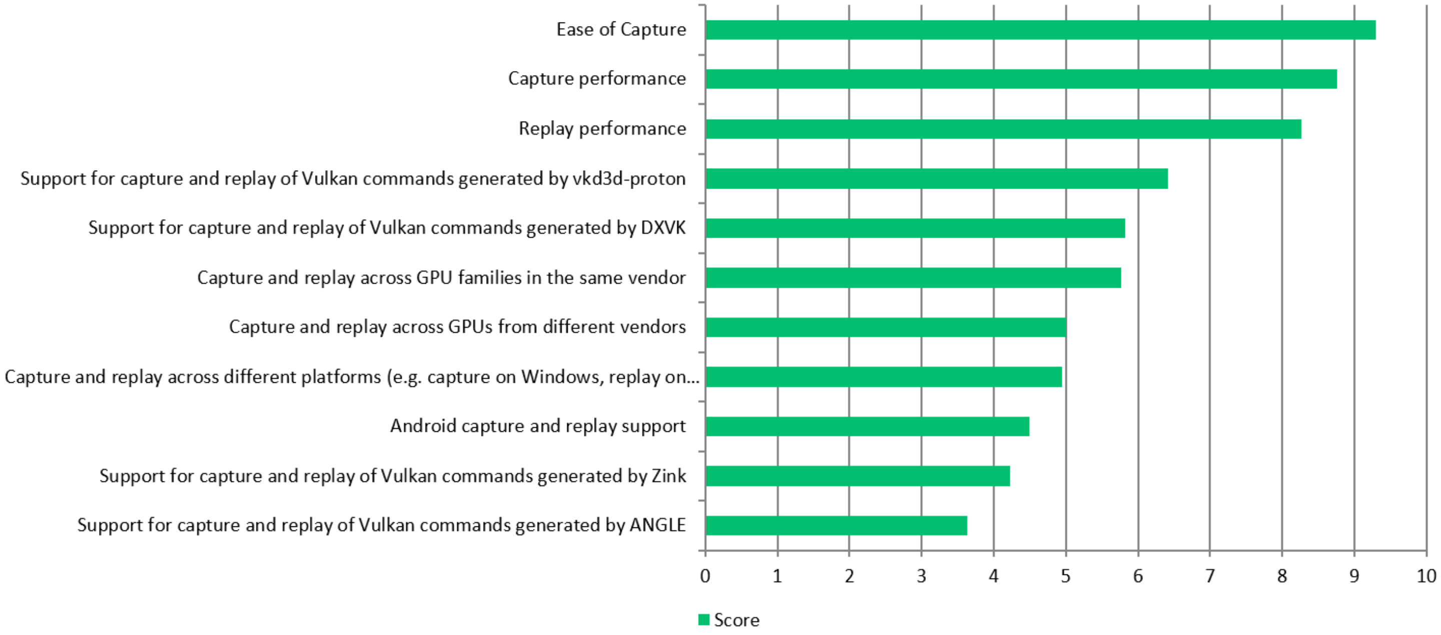
1. Memory tracking for cpu and gpu
    a. **LunarG Note: This wouldn’t really be a feature for the Vulkan Configurator. But could be a new layer.**
2. It does not always look well on 4K displays.
3. I would also like to see a possibility to disable particular features and extensions, override Vulkan version, capabilities and limits - for debugging and testing purposes. Something like devsim did and something like profiles can do, but in more user friendly way. Just to ""click"" on something and disable it.
    a. **LunarG Note: This sounds like a request for a GUI editor for Vulkan Profiles.**
4. I would also like to change the order of enumerated PhysicalDevices, or disable some in vkconfig.
    a. **LunarG Note: This is being developed for the next major revision of the Vulkan Configurator.**
5. I only use it to change layers and their settings when launching applications elsewhere, so I don't care for the application launcher and built in log view. I don't need any more features, and would like to keep the interface simple.
    a. **LunarG Note: The next major release of the Vulkan Configurator is going to take a tab based approach to simplify the UI and should improve this experience for the user.**
6. A .desktop file to make it easier to launch on Linux might be helpful.

7. Maybe not directly Vulkan related. . .but we need much better support for input (keyboard, gamepads, controls in general), and a professional grade support for the different hardware platforms. I am actively moving from D3D12 to Vulkan given the cuts Microsoft just made to the Xbox division.
8. N/A
9. In previous versions it could somehow "revive" itself if it couldn't start for whatever unknown reason. Now it's no longer working and we sometimes can't use the latest version of the configurators.
10. No idea i don't use it
11. Add more info to the listings of physical device features/limits.
12. Make it clearer which applicatons are currently being affected while the window is open (maybe show a list of running vulkan processes?)

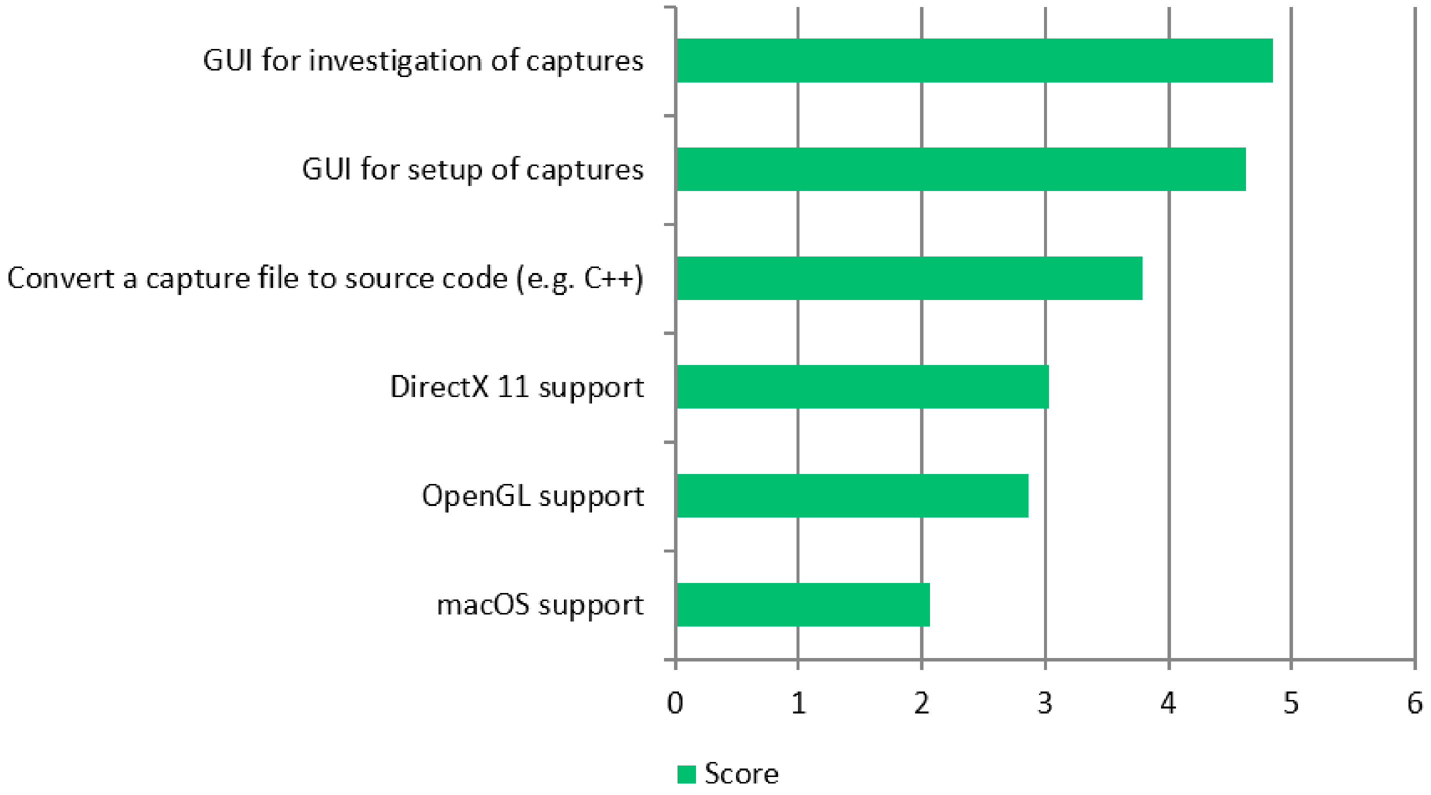
Which vendor-independent tool do you use for multi-frame API capture and analysis? Check all that apply:



Rank the following GFXReconstruct improvement areas (1 is most important)



Rank the following possible new GFXReconstruct features in terms of usefulness for your projects

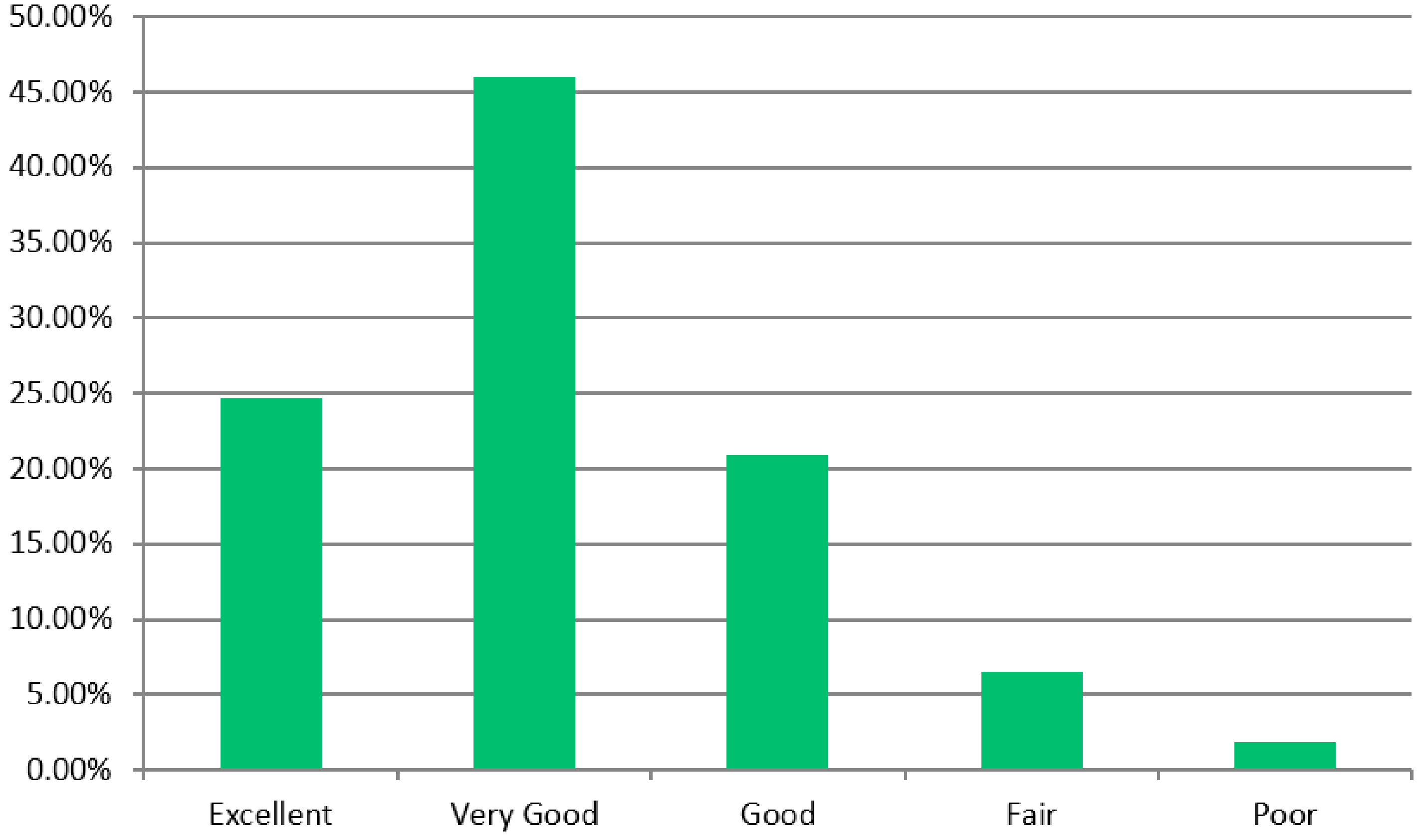


What improvements or enhancements would you like to have added to GFXReconstruct?

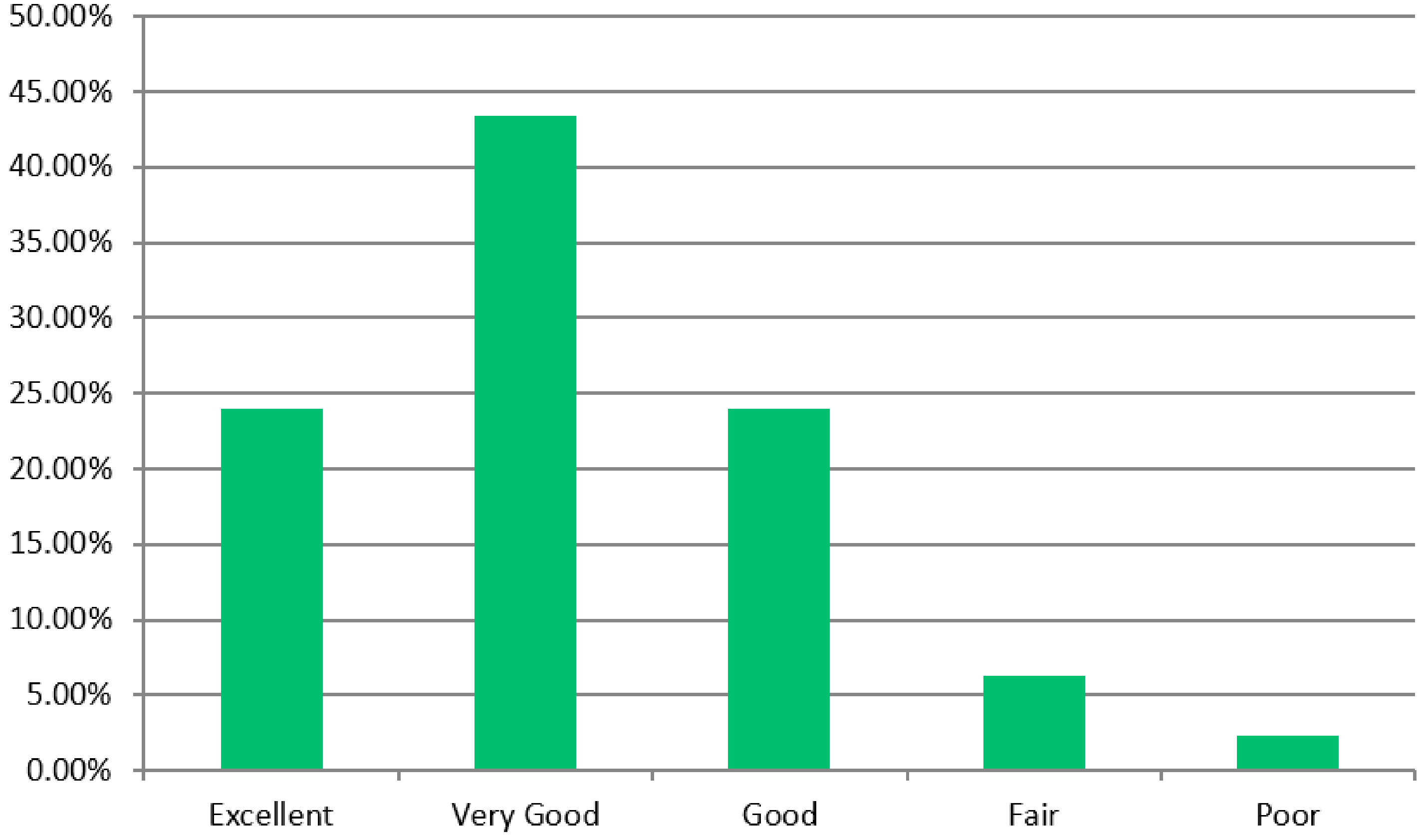
1. A possibility to inspect and modify the content of the capture. Either a visual editor, some text export/import like RenderDoc has with XML format, or just C++ code generation.
2. Convert a capture file to source code (e.g. C++)
3. After replay, giving some metric data, such as, timing, GPU counter and so on, about each API or some user defined actions.
4. Ability to modify existing capture (or with GUI tool or with export/import option)
5. please use C not C++ for source code capture conversion
6. deferred capture
7. GUI for capture investigation (especially if even limited editing capabilities would be supported) would be wonderful and help in multi-frame issues investigation. And even limited ability to edit the capture would be great to quickly test theories why something doesn't work, since in RenderDoc it's possible to edit only shaders.
8. Better ways to diagnose missing extensions on replay.
9. Better integration/documentation with device-sim layers to control feature support during captures
10. Dump captured resources like textures
11. multithreaded replay

12. Performance is poor for some games that cause a lot of page-guard exceptions inserted by gfx-reconstruct while VisualStudio debugger is attached. I'm not aware of any workaround, it makes debugging capture issues harder.

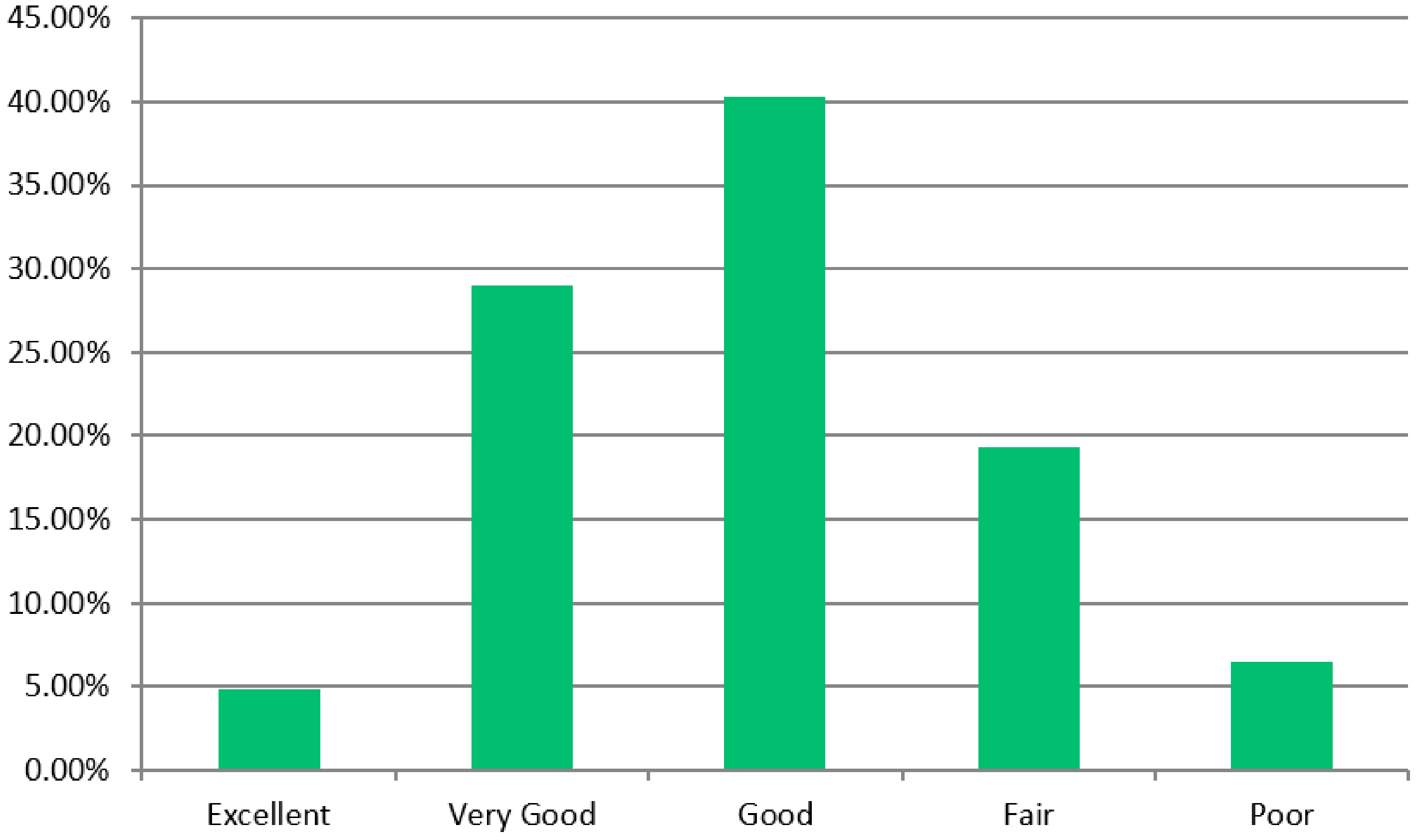
For creating a Windows application, how would you rate your development experience and the overall quality of the Vulkan ecosystem tools?



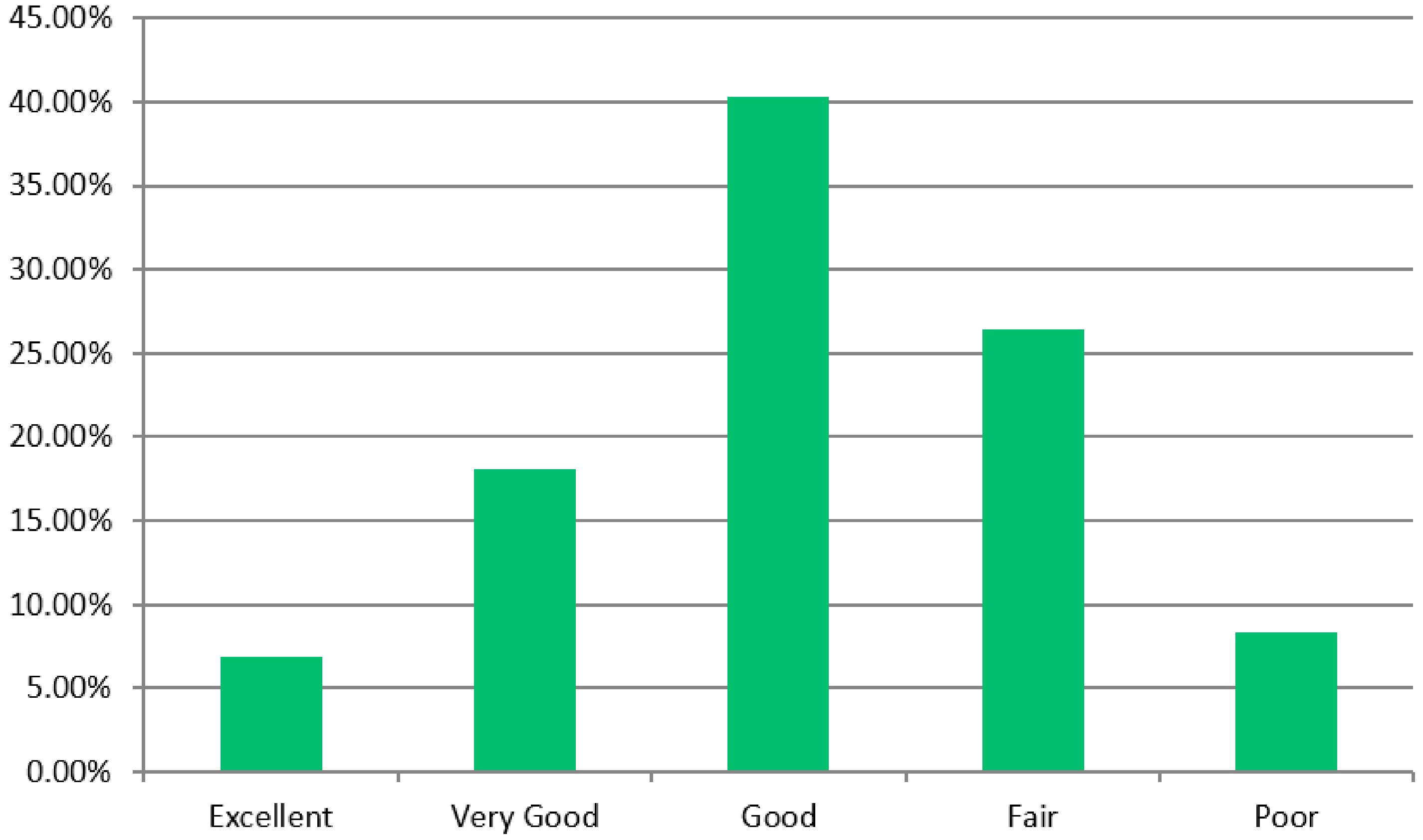
For creating a Linux application, how would you rate your development experience and the overall quality of the Vulkan ecosystem tools?



For creating a macOS/iOS application, how would you rate your development experience and the overall quality of the Vulkan ecosystem tools?



For creating an Android application, how would you rate your development experience and the overall quality of the Vulkan ecosystem tools?



Open-Ended Feedback

There were multiple questions in the survey asking for open-ended feedback. All of that feedback is compiled here, sorted by topic/category. All feedback is included, regardless if it is repeated multiple times. This way it is possible to see where there were multiple similar comments about an issue or topic. All feedback has been recorded below exactly as it was submitted to the survey tool.

Improving the Validation Layers

1. Messages:
    a. Clearer messages
    b. Make the output a bit more readable (D3D's error style is great - it goes to the point)
    c. I'd like a standard formatting tool to enable easy output of nice rich text with colors and hyperlinks. Not sure exactly how to do this in windows console, but if there was a small header-only library that does this, I'd use it for sure.
    d. make the output format more easy to read

    e. Put log writing in a separate thread, in a compressed bitstream, then decode it once the profiling is complete for a readable text file.
    f. better search engine on Vulkan's website from the error message/number. Some warnings don't have a number/ID and they are difficult to search for on the internet
        i. **LunarG note: Yes, in the specification there are some cases where the spec says it is invalid, but is missing a proper VU (but currently validated anyway). This makes it impossible to link back to the VUID in the specification when an error is reported. About two years ago there were about 80 known cases of UNASSIGNED VUIDs. LunarG is slowly removing them from the specification (replacing them with actual VUIDs) and today the count is around 25 UNASSIGNED VUIDs. Over time these UNASSIGNED VUIDs will be eliminated from the specification. There is a better communication channel with the working group now so future cases will be fixed right away.**
    g. Better message to find issues faster (for example, the ability to name function calls to find them back immediately). It could be implemented in the .pNext field for example. We could have a vkSetValidationLayoutContext with some debug info that we could put before API calls. This way, it would be immediate to find the issue and then resolve it.
    h. error messages are very obscure.
    i. add more descriptive messages like where to look. Beginners have a very hard time translating a validation error message to changes in the code.
    j. More user-friendly output messages,
    k. Some errors are spelled very general, when it is a clear problem underneath -> make it more fine granular
    l. Sometimes they are a bit hard to read in the console. The information is useful but it would be nice to have an option for simplified output/short version
2. Performance
    a. Performance
    b. Improving performance of some layers
    c. Improve performance.
    d. Do not cause too high overhead on Android 11 armv8 64bit.
    e. Performance
    f. Better performance
    g. Frame times certainly go down a decent amount. It would be super helpful to ship an app with validation layers enabled to write to a dump file. Release crashes would be much easier to spot.
3. Coverage
    a. Report too coarse synchronization (performance bottleneck)
    b. More GPU-Assisted validation for GPU-driven functionality (e.g. functionality using Buffer Device Addresses)
    c. More Raytracing stuff
    d. Better coverage
    e. By improving the coverage of the validation layer to catch more cases that are invalid on some GPU's but internally ignored on other GPU's.
    f. More checks,
    g. Add more performance issue detection
    h. coverage. Overall I'm mostly satisfied with debug layers

i. catch more mistakes in the code.
j. More of everything would be nice.
k. There are flags and values that are completely ignored on my GPU, but also by this layer. Leading to crashes on other vendors' devices.
l. More validation for EXT and NV extensions
m. I know it's hard, but GPU assisted debugging when using dynamic descriptor indexing and the like. As specially when using an image in the wrong layout via a bindless setup. That would be perfect!
    i. LunarG Note: Descriptor index validation is the top performance bottleneck for the validation layers and a refactor of this validation is WIP that identifies the actual descriptors that are used and only validate them rather than validating all descriptors. This requires some tricky instrumentation in the shaders sent to the GPU to identify the used descriptors and reporting that back to the validation layers. Once this refactor is done we expect that the performance will greatly improve and in addition, much new validation will be added that just can't be done today on the CPU.
n. fix deadlocks that happen when using timeline semaphores in a multi-threaded context, when you wait on that semaphore from a different queue
o. Improve the synchronization validation layers. They produce too many false positives right now.
p. Analyse the chance of data races (aka wrong or missing pipeline barriers)

4. Shader related
    a. Add tools for shader code runtime validation
    b. Shader Debugging is still difficult, especially Extensions.
        i. **LunarG note: As of fall 2022, source level shader debugging is available in RenderDoc. At the Munich Vulkanise event that took place February 2023, Greg Fischer made a presentation on this very useful and valuable tool. You can see the presentation here. You can also watch a video of the presentation that is available on the Khronos website here.**
    c. Some way of debugging pagefaults/hangs inside a specific shader. Ex. Automatically inserting printf-like instructions to save last position of each thread (enabled for specified shader (using shaderID for example, that user know that hangs).
    d. More shader-instrumentation level debugging.
    e. spirv-val has some holes, and I occasionally had broken shaders crash inside mesa without getting caught
5. Tools to help me find my error
    a. Is it possible to output what is the cause when an exception is thrown in VkLayer_khronos_validation.dll with 0xC0000005 ?
    b. The error messages must be improved to say what to fix instead of just saying what went wrong and a link to the (already complicated) spec. Something more akin to the D3D12 debug layers would be much more beneficial (= clear error messages that clearly explain what went wrong + how to fix the problem in most cases)
    c. More detail on the problem such as line numbers or variable names.
    d. If possible, point to the line number in the application that causes the validation layer error
    e. Debug symbols, so it is easier to debug why an error happens
    f. maybe more detailed explanation of possible solutions to e.g. synchronization hazards.
    g. More detailed explanation of the issue and give some guide to fix the issue or find more resource about the issue.

- h. tell us if data submitted is malformed to the current configuration provided
- i. Debugging layer that would show content of buffers, images, etc.
- j. would it be possible to collect callstacks to creation/update of objects that lead to validation errors?
- k. While I am not sure whether this can be fixed, sometimes, when you make some mistake and get hit with the VK_ERROR_DEVICE_LOST, you don't get much useful information on what could be going wrong.
- l. After VK_ERROR_DEVICE_LOST -> Dumping/printing last positions of threads in spv/glsl.
- m. You're basically left on your own, when a device lost error occurs. This is where I find validation layer lacking.

6. Enablement
   - a. Make it easier to enable synchronization validation or changing validation features. It's always so difficult to figure out which environment variables or files to set up, and I usually don't want to modify my application's code for it.
     - i. LunarG note: The Vulkan Configurator can do this for you. See more about the Vulkan Configurator that is included with the Vulkan SDK **here**.
   - b. A method to disable/disable validation layers at runtime. For example, in a loop of drawing 1000 objects, I want to disable it before the loop and then enable it after
   - c. It would be nice to be able to enable/disable them during runtime.
     - **i. LunarG note: enabling/disabling validation during the runtime is almost impossible. Every validation check depends on all prior state, which requires state to be captured for every prior API call. For example, it is possible an API call made in frame 10 modifies state that affects an API call made in frame 100.**
   - d. Would be nice to turn them on and off without having to recreate the instance.
   - e. Or maybe some kind of toggleable low overhead mode that doesn't actively check errors, just doing the minimum to track state.
7. Bugs
   - a. The GPU-assisted checks sometimes break applications or report what seems to be incorrect errors.
   - b. less bugs.
   - c. Support for newer Vulkan features is often very buggy to the point where the usefulness of validation layers is limited; e.g. public Github issue #4497.
8. Thanks
   - a. All is great!
   - b. Keep doing what you're doing! I have no specific feedback other than they have always been very useful.
   - c. The improvements over the years are great.
   - d. I think current layer development path is good.
9. Miscellaneous
   - a. Add support for Vulkan SC
     - i. LunarG note: See **VulkanSC-ValidationLayers** where validation layers for Vulkan Safety Critical are being developed.
   - b. Make them usable on macOS. I know in theory it's possible, but in practice it's not. It requires using the loader but MVK doesn't offer it's full feature set when it's used through the loader.
     - **i. Khronos/LunarG note: The Vulkan Portability solution VK_KHR_portability_subset extension) is currently provisional until MoltenVK has completed analysis of all CTS failures and determined their solutions. Before this extension can be**

removed from provisional status, MoltenVK will be modified to not have exclusive MoltenVK functionality that is not accessible via standard Vulkan interfaces.

c. Issues with external objects (opengl interop) are hard to debug.

d. The problem is not so much application errors, but driver issues. It would be great if validation layers also checked for known driver issues and warned about them. This would require collaborations with IHVs to disclose and document these problems, but would facilitate users life's enormously.

e. It currently does not seem to (at least correctly) check for NULL pointers in in many structs, instead just crashing when NULL is provided for a required pointer. Example: VkPipelineColorBlendStateCreateInfo.pAttachments

f. I don't t know. I have already done errors's fixing using Vulkan Validation layers for the last tasks. A lot of errors had a synchronization problem. I will report any problems using github, may be I will create a pull request.

g. Renderdoc does not plan to support Raytracing in near future

h. Synchronization and update at similar cadence to the release of new features (i.e. hold feature release in the SDK until VVL coverage happens).

   i. LunarG note: By design, the Khronos Vulkan WG does not delay extension releases until the validation layer development is also completed. This enables folks who want access to the new features ASAP. Holding SDK releases until the full validation is completed would be very complicated in that SDK versions are by Vulkan header release which will include APIs and extensions that are still under development for the validation layers. As such, Khronos has created tracking issues for the various APIs so that users can determine which SDK version will have the validation layer support as well. For example, see: Task list for VK_EXT_surface_maintenance1 and VK_EXT_swapchain_maintenance1 release #2006

i. Avoid use of extensions and function pointers (getProcAddress) to get the validation layers operating

j. Maybe examples of code that fails the validation and the same code adapted to pass the validation?

k. Unsure.

l. Best practices / gpu vendor recommendations

Windows Developer Issues

1. performance tuning is hard
2. profiling code performance
3. Profiling performance can be hit or miss depending on vendor tool availability
4. Dont rely on environment variables. We're programmers, not system managers.
   a. LunarG note: The Vulkan Configurator can do this for you. See more about the Vulkan Configurator that is included with the Vulkan SDK here.
5. While the LunarSDK is already quite good improvment versus OpenGL lack of SDK, it still fails a bit short of the DirectXTK + PIX + VS HLSL development experience.
6. It's kind of annoying how the SDK installs to the root of the C drive.

7. Difficult to troubleshoot random VK_ERROR_INITIALIZATION_FAILED from vkCreateDevice. Likely a driver or setup issue but again my game gets blamed for it. #1 crash on R6Extraction PC, we could never fix it…
8. Any layer can intercept any entrypoint, so bugs in layers (like OBS) can cause crashes in my game, so we get blamed for it.
   a. LunarG note: The Vulkan Loader has been enhanced with additional debugging logging and tools to help identify faulty layers. See the LunarG white paper, **The Vulkan Loader and Vulkan Layers: Diagnosing Layer Issues**
9. Debugging synchronization issues
   a. **LunarG note: A Vulkanise event was done in October of 2022 to explain Vulkan synchronization and using the validation layers. The slides are here. This video could help with debugging on synchronization issues.**
10. PIX is still better than Renderdoc :)
11. Only nsight has been useful for debugging ray tracing workloads.
12. Vulkan APIs usually accept large *_DESC structs with many fields. Ideally, these fields would be initialized with designated initializers (C++20), but C++ (somewhat painfully) stipulates that initializers need to be specified in declaration order. MSVC tooling doesn't yet display fields in declaration order in intellisense.
13. The OS itself
14. Driver bugs.
15. Dealing with window events and swapchain resizing
16. No serious pain, but an official and regularly updated chocolatey package would be nice.
17. Not debuggable "access denied" crashes
18. hard to fine ktx texture related tutorials
19. Missing viewing GPU memory during debugging in Visual C++.
20. MSVC's idea of C11 has very little connection to any published standard. MinGW FTW :D
21. cannot update vulkan instance, you have to first DELETE the old, then install the new, even though there is a maintenance application
22. Debugging tools are very poor for compute shaders
   a. **LunarG note: As of fall 2022, source level shader debugging is available in RenderDoc. At the Munich Vulkanise event that took place February 2023, Greg Fischer made a presentation on this very useful and valuable tool. You can see the presentation here. You can also watch a video of the presentation that is available on the Khronos website here.**
23. Better support for tools + Physical device address/buffer_reference. Currently a lot of the vulkan ecosystem revolves around RenderDoc, which is a great tool, but doesn't have good support for physical device address, which is used extensively in our code, which means the new debugging features are completely useless and greatly slows our development.
24. False positive in validation layers.
15. Linux is faster ... Always a Problem for Cross Platform
16. Microsoft's C++ compiler sucks! It's not your fault though :)
17. not directly windows related, but the headers are too big, especially the hpp ones. maybe they could be split into subsets (with forward decls) so I don't have to include everything everywhere?
18. Documentation
19. Vulkan feels too isolated from the Windows applixation programming model. You really have to make a significative effort to create a Windows app with Vulkan that gives you the best of both worlds.

20. Also: why is text rendering so supremely hard to make with current graphics APIs?
21. poor sparse memory performance
22. I don't know anything about it. I just want my photo tools to work.

Linux Developer Issues

1. performance tuning is hard
2. Proprietary drivers (i.e. Nvidia) make everything worse.
3. RenderDoc
    a. renderdoc lacks support for running directly on Wayland and has to be launched with QT_QPA_PLATFORM=xcb instead of using the .desktop file.
    b. Renderdoc laggs behind in support, but it's understandable.
    c. RenderDoc does not support capturing on Wayland currently :(
    d. Better support for tools + Physical device address/buffer_reference. Currently a lot of the vulkan ecosystem revolves around RenderDoc, which is a great tool, but doesn't have good support for physical device address, which is used extensively in our code, which means the new debugging features are completely useless and greatly slows our development.
    e. Lacks RenderDoc
        i. LunarG note: RenderDoc is supported on Linux and available for download at renderdoc.org
4. gpu hang debugging
5. Installing the Vulkan SDK on Linux still doesn't seem as easy as it could be.
6. I currently use the LunarG SDK packages for Ubuntu. It's unclear from the documentation what the process is to safely uninstall these packages and revert to the default libvulkan etc. provided by the system repositories. When I mark the LunarG packages for removal in a tool like Synaptic and preview the changes, it indicates that various other core packages (related to the desktop environment) are "no longer needed and will be removed," leading me to worry that removing the LunarG packages would brick my Ubuntu install. I may be misreading the message from the package manager, but confirmation of the correct way to uninstall the LunarG SDK packages would be nice
7. Vulkan has much better support for wrangling linux-isms than previous APIs, but it's still not easy to support multiple window systems.
8. Always seems to work, rarely any bugs
9. Debugging GPU crashes in Linux is more painful than Windows. Built-in DRED-like functionality that kicks in on TDR for Linux would be helpful.
10. DXC compiler crashes for last verion of Vulkan sdk, I will create issue.
13. Sometimes tooling feels like an afterthought.
14. All windows for all currently active programs are very laggy when moving when running a vulkan application and using VK_PRESENT_MODE_FIFO_KHR
15. would like to see meson support for glsl compiling.
16. would like to see a standard application framework beginners can use to get up and running with hello triangle leading to more complicated model loading and multipass pipeline rendering
17. The main pain point I have is less about the development experience and more the very wide variety of setups that individual users have (proprietary vs. open drivers, x11 vs. wayland, etc.)
18. glslangValidator - is a critical thing which sometimes has bugs. Last time I wanted to write a custom intersection shader, and it didn't compile due to a bug that had already been fixed upstream but not released for more than 2+ months after, so I had to do everything myself. This isn't convenient given

that people (usually) prefer native-to-your-distro installations rather than "make install" from sources if one already had a native package installed. A minor inconvenience, yet - a bug that has been there for some time. I found the cause myself and could have even fixed it myself, but the problem here, I guess, is that whenever such critical bugs occur (like when the users can't compile a shader!), the work on it should start ASAP, and it should be released ASAP. That's just how I'd do it. Perhaps, I am too biased to perfection.

19. There are No ...
20. Vulkan+Linux is a dream :)
21. Documentation
22. I have noticed subtle differences in the past between the Windows and Vulkan SDK. For example, some includes are missing, moved, or have a slightly different name. Although, I haven't done any development on Linux in the past year.
    a. **LunarG note: This comment was submitted to LunarG at vulkan.lunarg.com. LunarG is working on scrubbing all the include files between Windows, Linux, and macOS and working to get consistency.**

Android Developer Issues

1. Driver bugs/quality
    a. Driver quality is low. Support for new Vulkan versions is slow to arrive. Vendors provide weird proprietary tools which don't usually work for me using a desktop Linux for development (I would prefer FOSS, non-vendor specific tools).
    b. Android vulkan drivers are also bad at reporting failures within the driver/compiler.
    c. Numerous driver bugs. Errors and crashes are hard to debug and diagnose and work around.
    d. It's so painful that GPU drivers don't get updated, preventing me from upgrading to ""modern"" Vulkan.
    e. We need to solve the Vulkan version update on Android somehow... userland drivers or something? It's very bleak right now…
    f. Android devices are flakey by nature, and IHV debug tools are way worse than native counterparts.
    g. The driver for my main device somehow only supports VK 1.1 while the GPU could support 1.3
2. Performance
    a. performance tuning is hard
3. Tool enablement/developer experience
    a. Debug layers are hard (impossible?) to enable and tools are broken or have very limited functionality.
    b. The development experience is often much worse than with GLES and certainly way way behind the other mobile platform.
    c. Google's tools -- gradle, Android Studio, etc -- are almost uniformly harder to use than any other game platform (PlayStation, XBox, Windows, Nintendo). This isn't Vulkan's fault though -- Vulkan supports Android as well as could be expected, given Google's environment
    d. Google makes us go mess around github repos with raw samples, doesn't expose Vulkan to Java/Kotlin, lacks any kind of SDK that can compete against Metal Kit/Scene Kit/Metal IO with Swift. Using the AGDK is pretty much a DIY experience made by devs that insist in using C for everything.
    e. Installing the validation layers on an Android device is painful, but it kind of makes sense.

   f. There are a lot of odd android-isms that make developing for it (both app and driver) harder than desktop, but those are mostly in Google's wheelhouse.
   g. Android development is painful in general, but it's rarely because of Vulkan tooling.
   h. Some tools are as a straightforward to use, e.g. RenderDoc.
   i. No simple out-of-application utility to enable/disable layers.
   j. Another stuff is lack of proper support tiled memory in RenderDoc. Lazily allocated memory. In order to use RenderDoc the app should be recompiled to use regular memory.
   k. Recent NDK removed prebuilded Validation Layers so you have to be an expert to build Layers from source code. Doing it by manual never worked for me, so I created my own build system. Here is step by step manual using CMake without tests: https://github.com/Goshido/android-vulkan/blob/master/docs/vulkan-validation-layers.md
      i. **Note from LunarG: With each Vulkan SDK release, the Vulkan-ValidationLayers repository has a corresponding Android build released. See https://github.com/KhronosGroup/Vulkan-ValidationLayers/releases**

4. Shader toolchain
   a. DXC support for Android compatible Spirv codegen is very git-or-miss
   b. The compiler pipeline is terrible. Shaders often produce different results on each vendor, spirv-opt often breaks shaders, or makes them slower.
   c. I want to use DXC compiler for directly compile HLSL shaders instead of old/bad/legacy glslang.
5. Miscellaneous
   a. Android feels second class.
   b. Iteration times, emulation
   c. We're using it mostly with Rust and have good experience there, we're contributing quite a bit to improve ecosystem there as well.

Note from Google: https://github.com/android/ndk-samples/tree/main/hello-vulkan provides some Vulkan samples to help the Vulkan developer on Android.

macOS/iOS Developer Issues

1. RenderDoc on macOS
2. Lacking renderdoc support
3. Honestly MVK is a mess.
   a. In 2022 i've been updating to MVK releases regularly as the app i use it in also needed to be released. NONE of the releases i used were actually production ready. They all either contained memory leaks, freezes or crashes. I currently have to use a modified / custom built mvk version because public releases aren't production ready. I can now only update mvk right after a release of my own app so that i have enough time to do testing myself and fix any new issues introduced in the latest mvk release, that's a horrible way of feeling about a library so vital to my app.
   b. There is/was BETA stuff in public releases (descriptor indexing). This is wasting my time trying to get it to work and then only after spending a lot of time finding out that it's not expected to work and will be changed. It was not clear to me that this is beta stuff (i didnt even enable beta functionality) and it was only made clear to me after reporting issues with it.

c. There is misleading / confusing information preventing me to use mvk effectively. VK_MVK_moltenvk is being called a ""Vulkan extension"" in the user guide, but i found out later that's it's really not an extension and thus i still cant use the loader.

d. Development is slow, there's still no 1.3, i also still cant use sync2 yet. Also it being made by a single developer is scary in terms of sustainability, reliability and correctness.

e. Finally there is too much configuration that needs to be done by default. On windows i have 0 configuration variables for the driver and everything works wonderfully, but on macOS i somehow need to configure mvk in order to get it to do the right thing. Why aren't extra features that only some users need implemented as extensions. Then by default mvk can be correct, and if that's too slow/strict i can enable extensions to make it work differently. I'm guessing this is too much work, but i think that's the vulkan way. Probably this links back to the 1 person is not enough issue?

f. It's really unfortunate really. It was fun to get something up and running using mvk, but currently i cant faithfully tell my customers / employer that the product will keep working on macOS in the future without having to write a metal backend. I realize this is Apple's fault for not providing a real Vulkan driver, but MVK presents itself as alternative and it's just not living up to expectations right now for me.

g. Some of these issues were already present and reported last year. I got the response that lunarg is aware of it and looking into it but have not seen any result yet other than the 1.2 and portability push which are nice. I'm really not sure what's holding mvk back, is it interest from khronos side, or funding maybe?

h. Note from the MoltenVK development team:

   i. One of the challenges of running an open-source project like MoltenVK is that we do not have a list of customers we can reach out to for feedback and priorities. So we very much appreciate receiving feedback like this.

   ii. Crashes and leaks are important to identify and fix, and we encourage everyone to report these to the issues list on the MoltenVK site. We do regularly run the full suite of CTS tests on MoltenVK, but users can sometimes uncover issues that are not covered by the CTS tests, depending on how they are using Vulkan.

   iii. Descriptor indexing is definitely in a beta state, primarily due to the disconnect between how Metal Argument Buffers work relative to Vulkan descriptor sets. With the introduction of Metal 3, we have plans, and a priority, to improve this functionality.

   iv. The private VK_MVK_moltenvk extension is a holdover from the early days of Vulkan and MoltenVK. We will be deprecating this functionality, and removing it as an advertised Vulkan extension. We will be providing other mechanisms for configuring MoltenVK programmatically, and most of the other functions in this private extension have been replaced by the public VK_EXT_metal_objects extension.

   v. Having MoltenVK be a configuration-free driver is definitely desirable. And for the most part, MoltenVK is effective straight out of the box without any configuration applied. The additional configuration arises because of the differences between Vulkan and Metal behavior, and the different priorities app developers put on how the bridging of those Vulkan/Metal differences should be handled. Essentially, MoltenVK configuration exists to ask app developers "based on how you are using Vulkan, do you want to map Vulkan to Metal this way, or that way". We would definitely appreciate feedback on what app developers consider should be

the "default" for each configuration option. If you have suggestions, please submit an issue to the MoltenVK site.

4. MoltenVK's file size is really big.
5. The portability enumeration extension is annoying.
6. MoltenVK updates to deal with iOS changes taking time to make there way into the next Vulkan SDK release. Unsure how to test a pre-release MoltenVK build.
7. It has improved a lot since Richard's new installer, but the iOS support was pretty rough last I checked. (its probably better now)
8. moltenvk can have signficant performance overhead leading ne to consider also writing a Metal backend
9. Sometimes double buffering does not work correctly, the VulkanMemoryAllocator library completely makes the program non-working, this library does not want to allocate memory on macOS, after switching to standard Vulkan functions, everything worked
10. I encountered some Metal driver bugs, esp. wrt subgroup operations. Nothing I can blame Vulkan for!
11. MoltenVK dumps out the spirv-cross'd Metal code when something goes wrong, which is helpful but confusing when trying to track down an error reported by the Apple shader compiler.
12. I have mostly given up on it because it lacks support for modern approaches and writing a separate metal backend ends up making more sense
13. Nothing on your end. Mainly apple being a pain in the butt.

Miscellaneous Comments

1. Keep up the good job!
2. I hope to add more Chinese materials about Vulkan.
3. I am developing an application using Vulkan. I appreciate it very much.
4. Thanks for the job you're doing on Vulkan !
5. Playstation and XBox keep being left out of official Vulkan communications, even if only to acknowledge that there isn't any support there, and on the Switch, NVN plays a major role despite Vulkan support.
6. I really wish there was a way to write vulkan code for web browsers using c/c++ and emscripten
7. Keep up the good work!
8. All in all: good job!
9. A Fixed Function(ala Classic OpenGL) path-tracing layer that sits above Vulkan would be pretty cool. Learning graphics with Vulkan or D3D12 is pretty rough these days for undergrad students. Most just want to use Unreal or Unity, its become pretty challenging to teach the next generation of graphics programmers since simplicity and approachability have taken a back seat to performance and features.
10. Please keep good support for dxvk and vkd3d.
11. no
12. In Linux, Intel has more tolerance than Nvidia.
13. Nope
14. Add an official vcpkg portfile instead of just providing the SDK standalone. This allows for easy updates, independent of system packages. There are currently vulkan packages on vcpkg, but they are badly out of date and there are several conflicting packages.
15. Frame graphs/render graphs are a de-facto standard way to connect a gameplay programmer to a GAPI, yet there are hardly any tutorials even mentioning their existence. Tutorials teach you how to do basic hardcoded passes and barriers and they don't tell you that nobody does this in practice, synchronization is in practice automated with render graphs and here's how they work."

16. no
17. P.S. If you have time please check if I was right:
    https://gist.github.com/vityafx/3de208ba5f81738b7c055a586ac3ecbb
18. Thank you.
19. More Tutorials would be awesome.
20. There just needs to be more clear information and documentation out there.
21. Multi-threading
22. Asynchronous resource manage
23. Multi-ComandBuffer manage
24. For question about layer impact, there is no i don't know option.