Results of 2020 Vulkan Ecosystem & SDK Survey

Karen Ghavam, LunarG
January 2021

Table of Contents

Executive Summary

This report provides the results from the LunarG ecosystem survey completed in November of 2020.

Methodology:

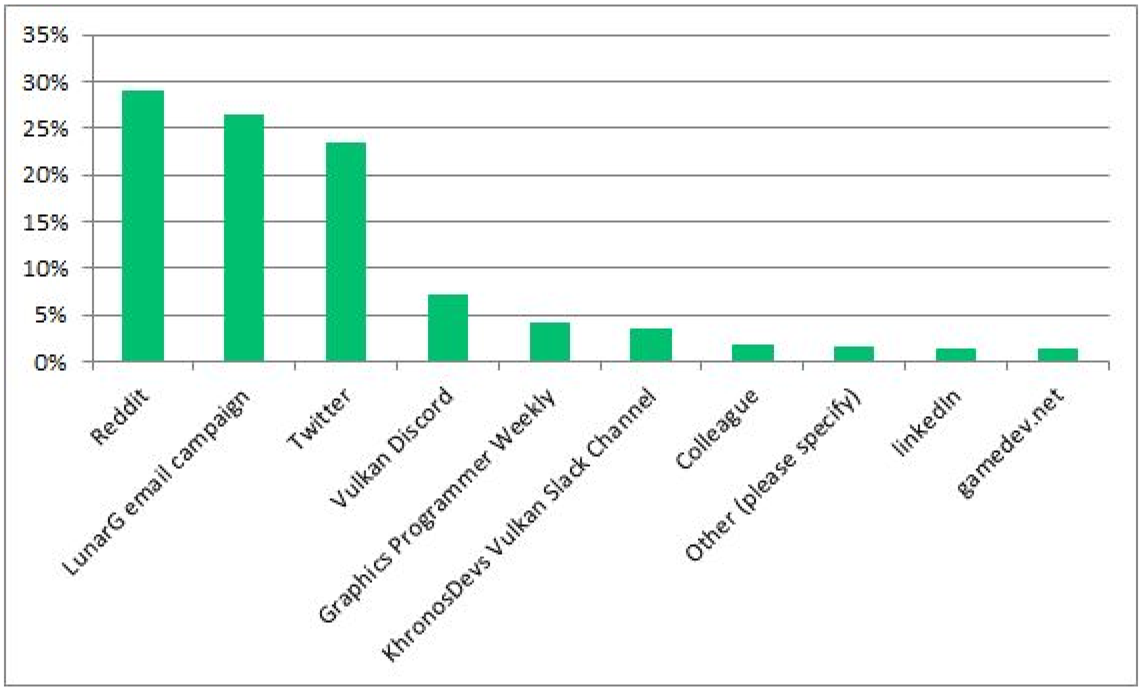
1. LunarG developed this Vulkan ecosystem survey to gauge the Vulkan community's use of and satisfaction with the current Vulkan ecosystem. This is a follow-up survey to the previous LunarG survey completed in December 2019 (fourth Vulkan ecosystem survey by LunarG).
2. LunarG attempted to reach as many Vulkan developers as possible -- both SDK users and non-SDK users. LunarG advertised the survey on Twitter, Reddit, LinkedIn, the Khronos DevRel slack channel, and sent it directly to 13,000+ members of the LunarG LunarXchange Vulkan SDK mailing list.
1. 59% of respondents are hobbyists or developing for non-commercial reasons. 40% of the respondents are commercial developers. Throughout this report, if the results for commercial developers only vs. the total population revealed a difference, it is shown.

Key points:

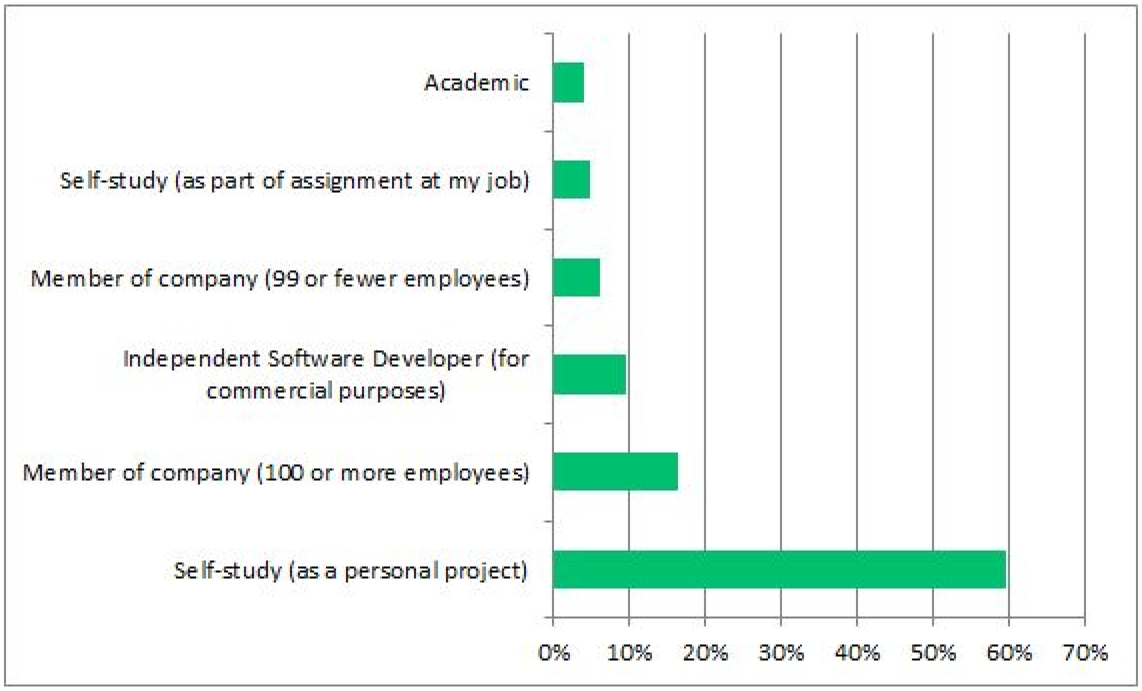
1. Vulkan developers have become more experienced, which is resulting in feedback and suggestions that are more knowledgeable, constructive, and helpful than in previous surveys.
2. There were 364 respondents (slightly more than last year).
3. Respondents were a good mix of developers for Linux, Windows, Android, iOS/macOS, and Google Stadia.
4. Although the Vulkan SDK is desktop focused, some Android developers were also represented in the survey.
    a. There were 101 total Android-based respondents (28% of total respondents),
        i. 51% were commercial developers, 46% were hobbyists
        ii. 56% had released their work or were planning to do so
5. The survey respondents overall gave a distinctly favorable score on the quality of the ecosystem. This is the same as last year.
6. Our investment in synchronization validation in the last year pleased many folks in the ecosystem. It was the "top hitter" in being a very useful validation layer object, and people want more…

The last chapter of this report is "Key Actions for LunarG." This chapter lists all of the identified ecosystem improvement requests in LunarG's control that will be prioritized and worked on in the coming year.

Where did you hear about this survey?



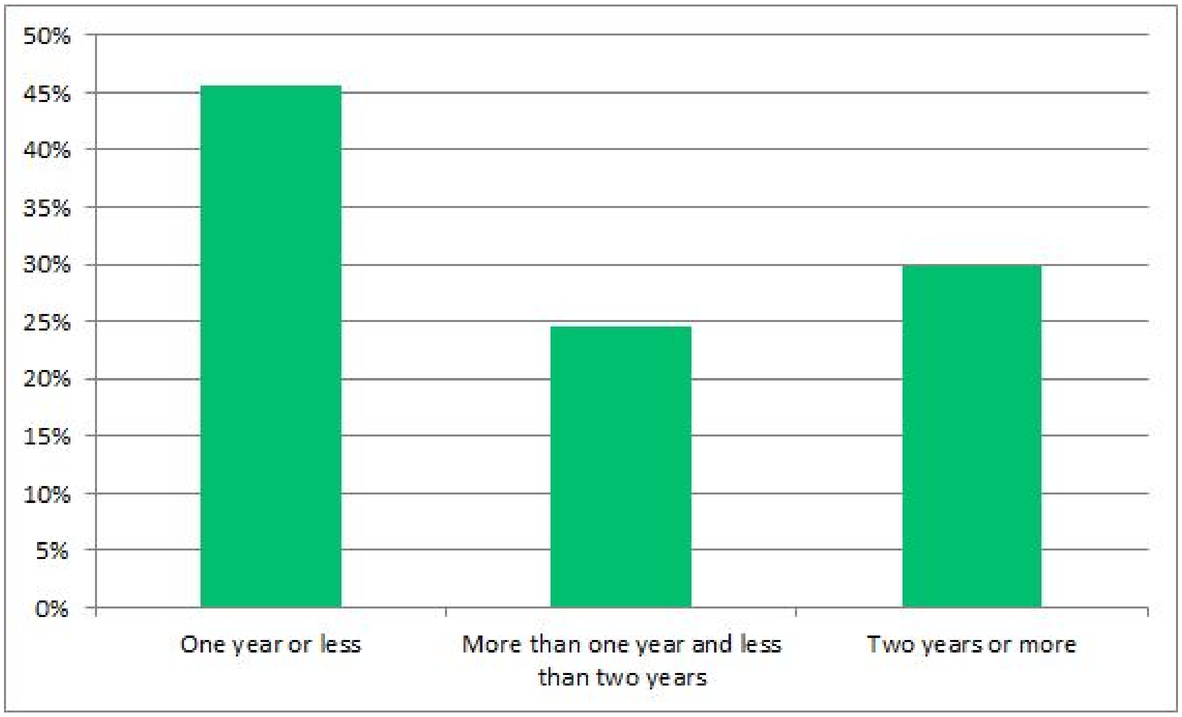
What type of Vulkan developer are you?



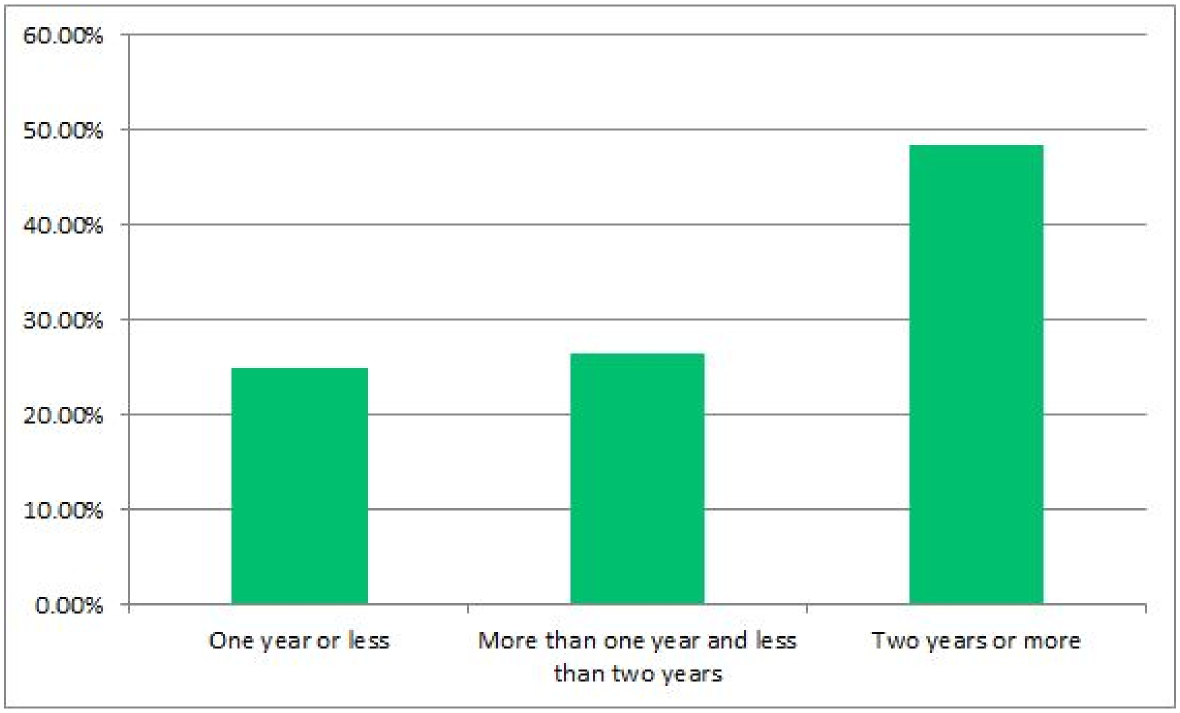
Note: 59% of respondents are hobbyists. 40% of respondents are commercial developers. Through this report, if resulting data for commercial developers only was different than the overall population, those differences are shown.

How long have you been developing with the Vulkan API?

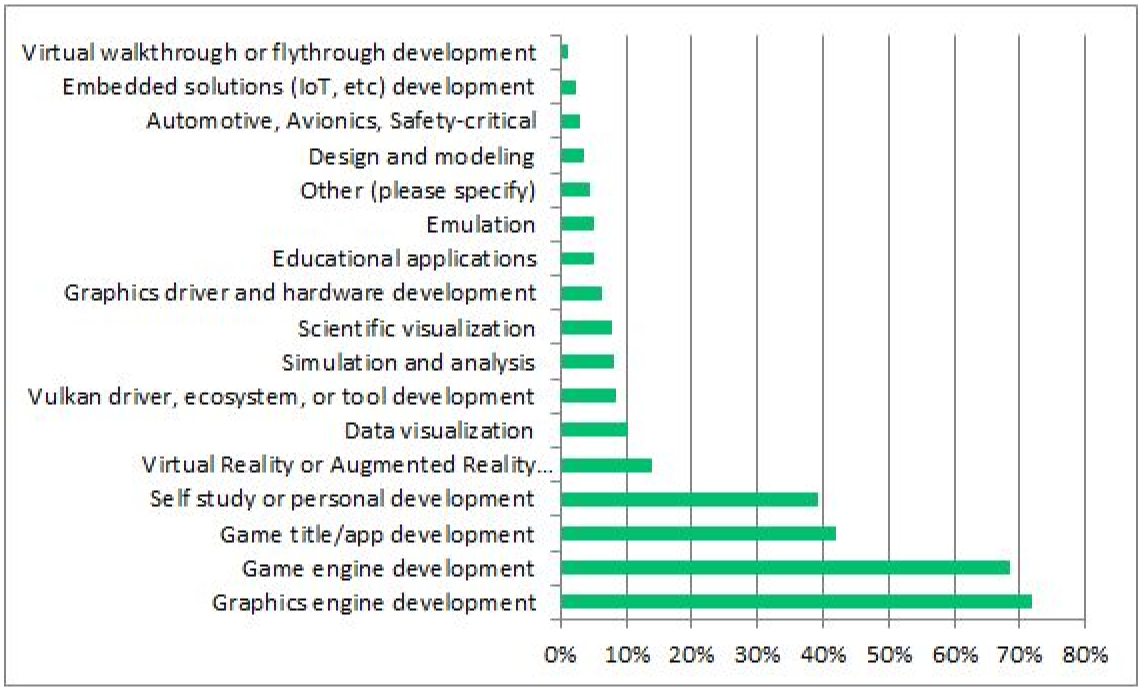
All respondents:



Commercial respondents only (more years of experience):

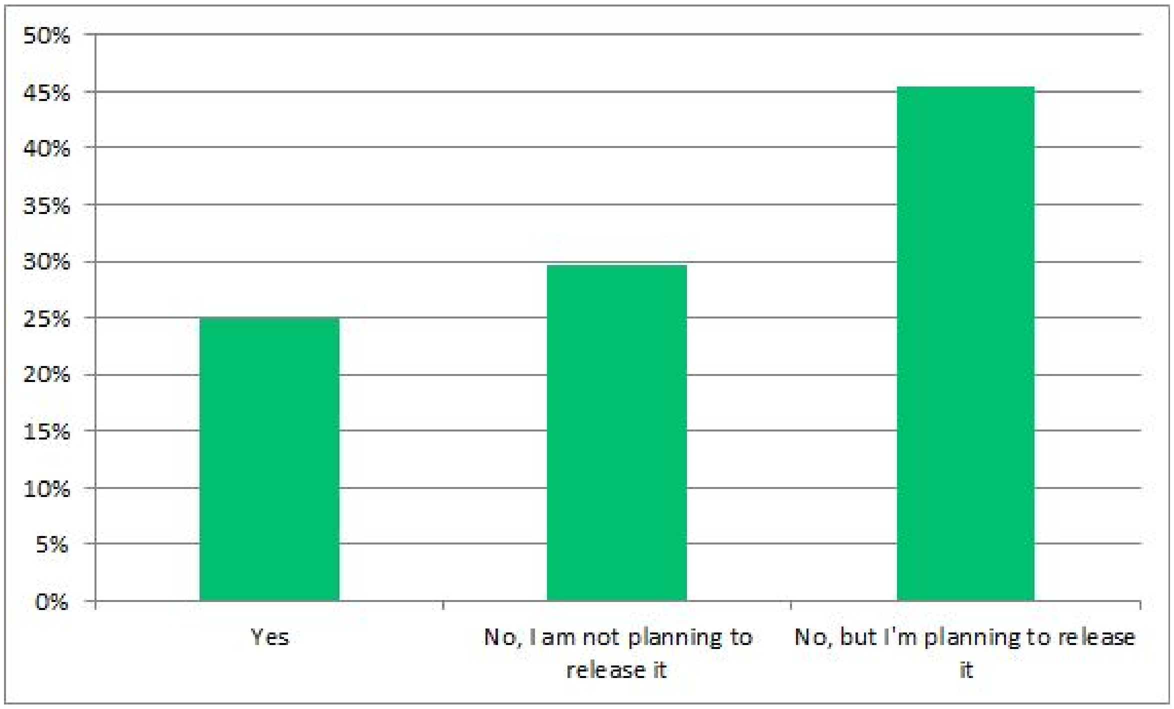


Your development is for what type of use case?

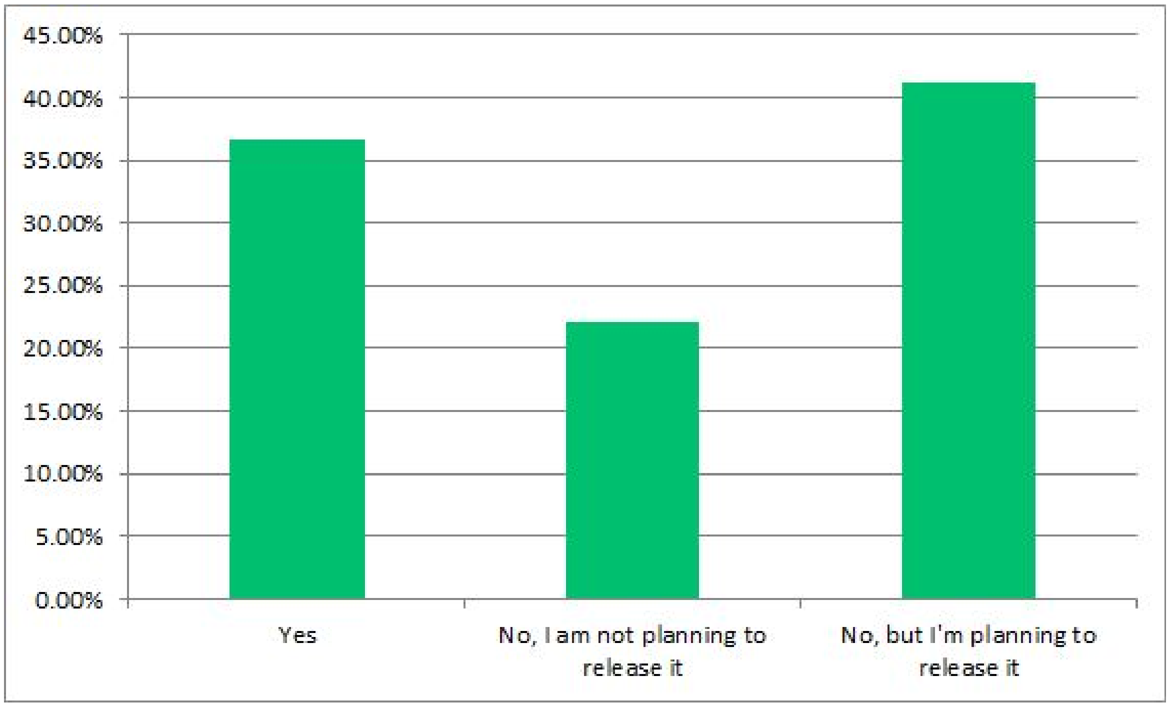


Have you released your Vulkan development project for public use?

All respondents:

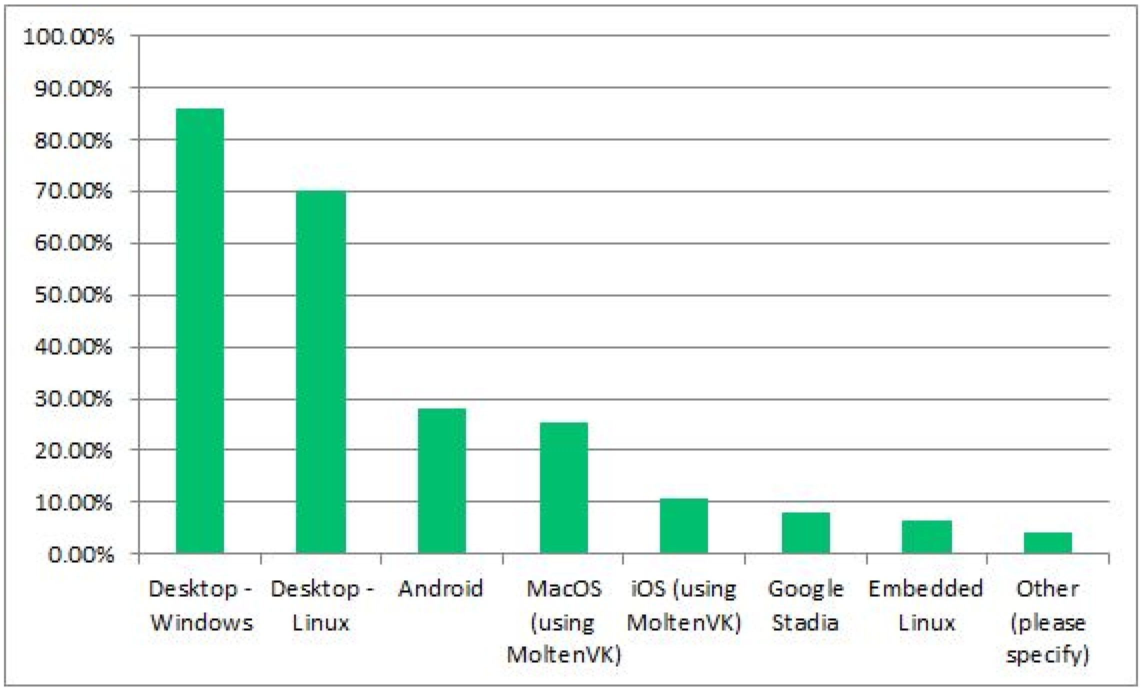


Commercial developer respondents only:

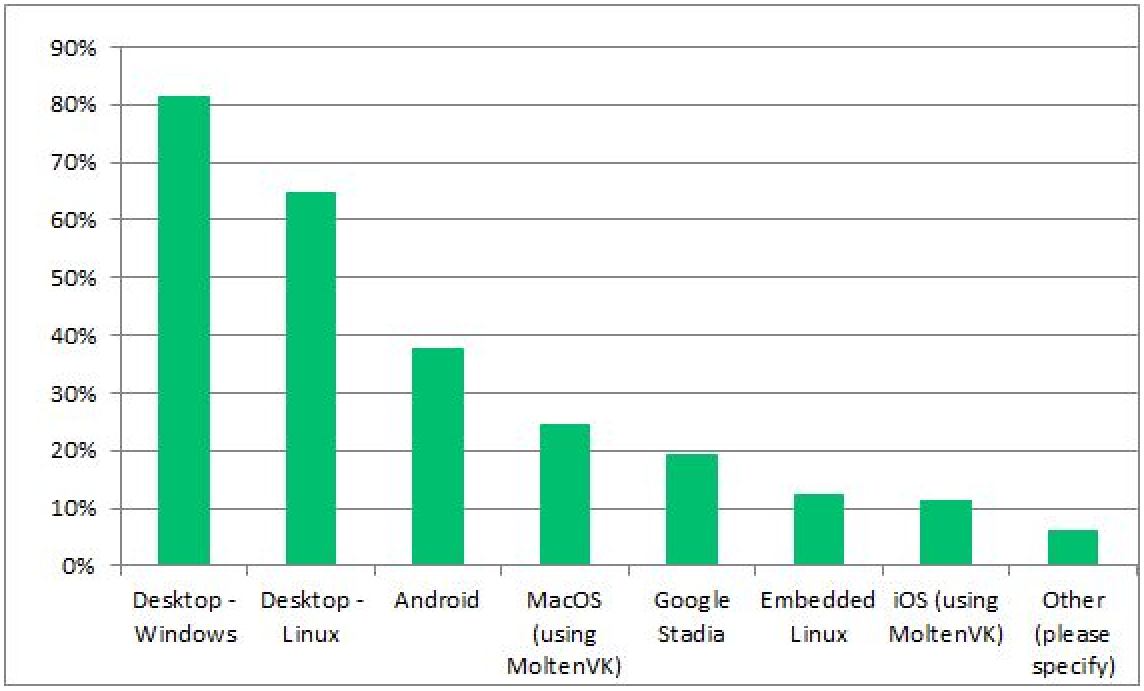


What is the target of your Vulkan application? Check all that apply

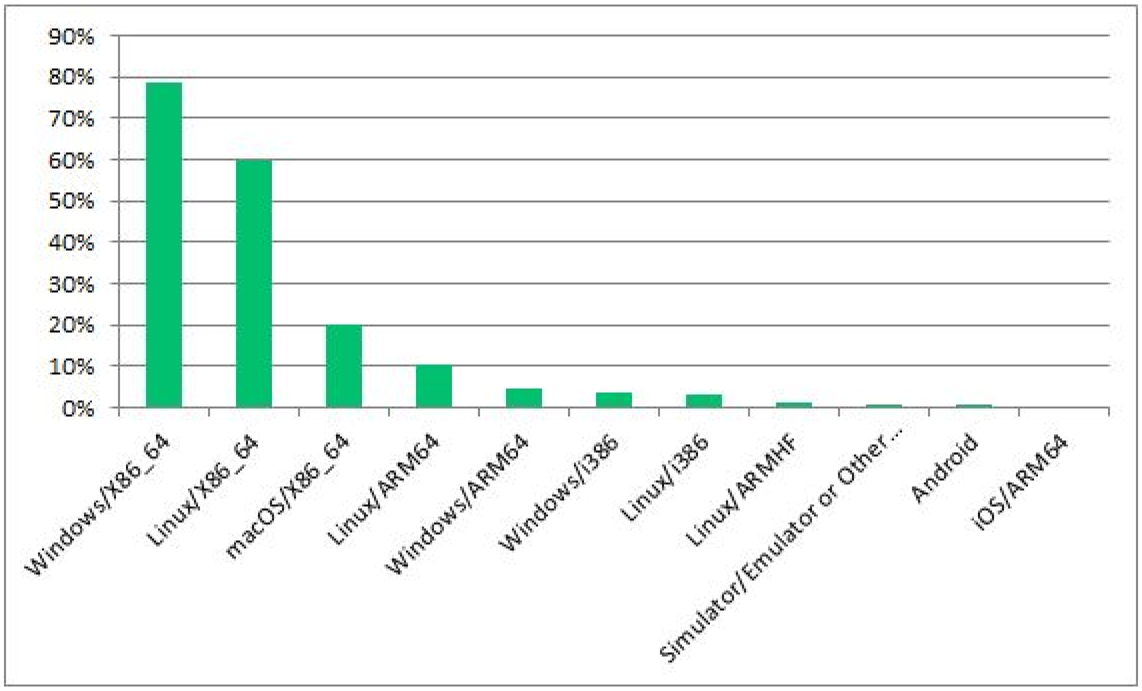
All respondents:



Commercial developers only:

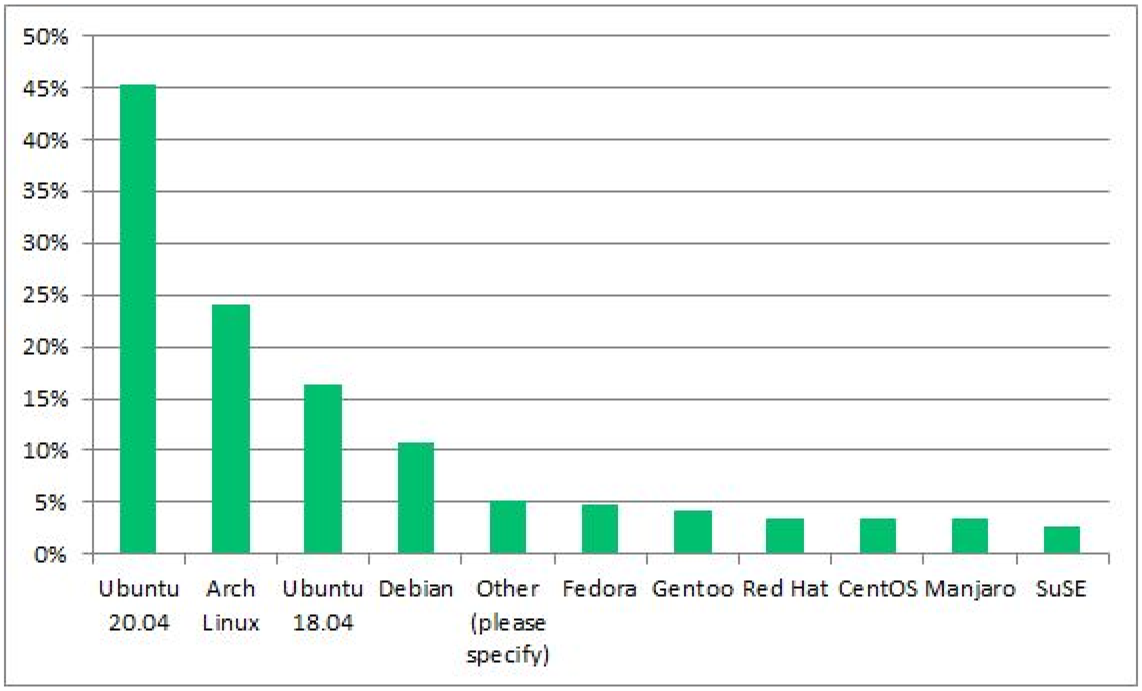


What operating systems/architectures do you use for your development environment? Check all that apply:

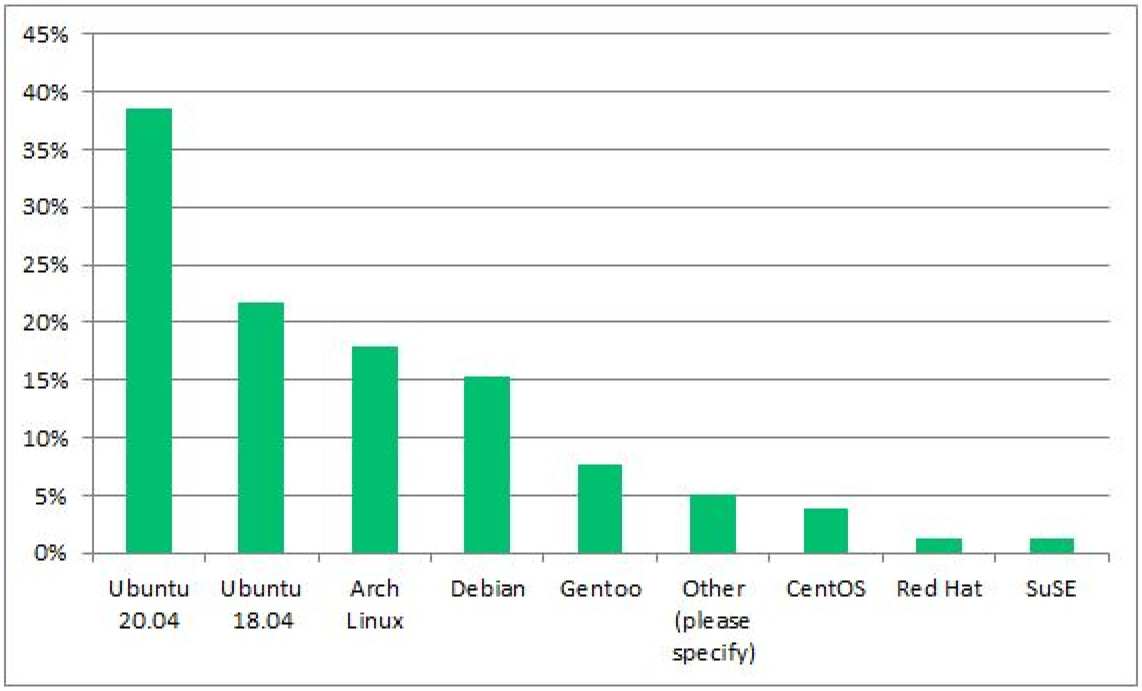


What is the primary Linux distribution you use for development?

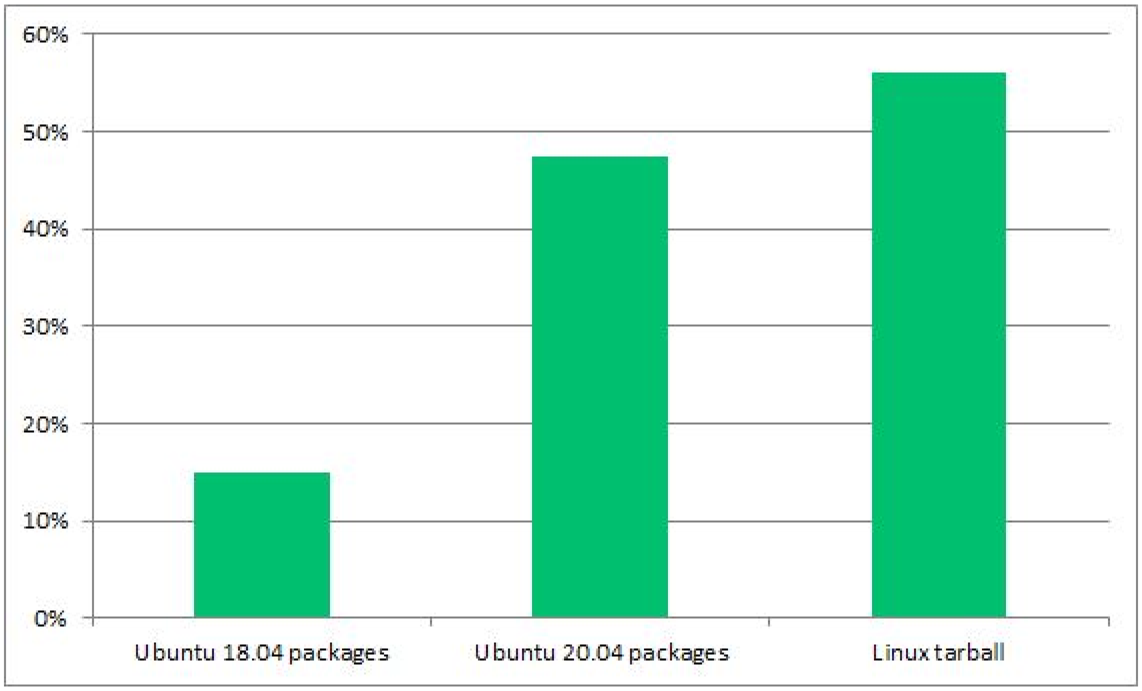
All respondents:



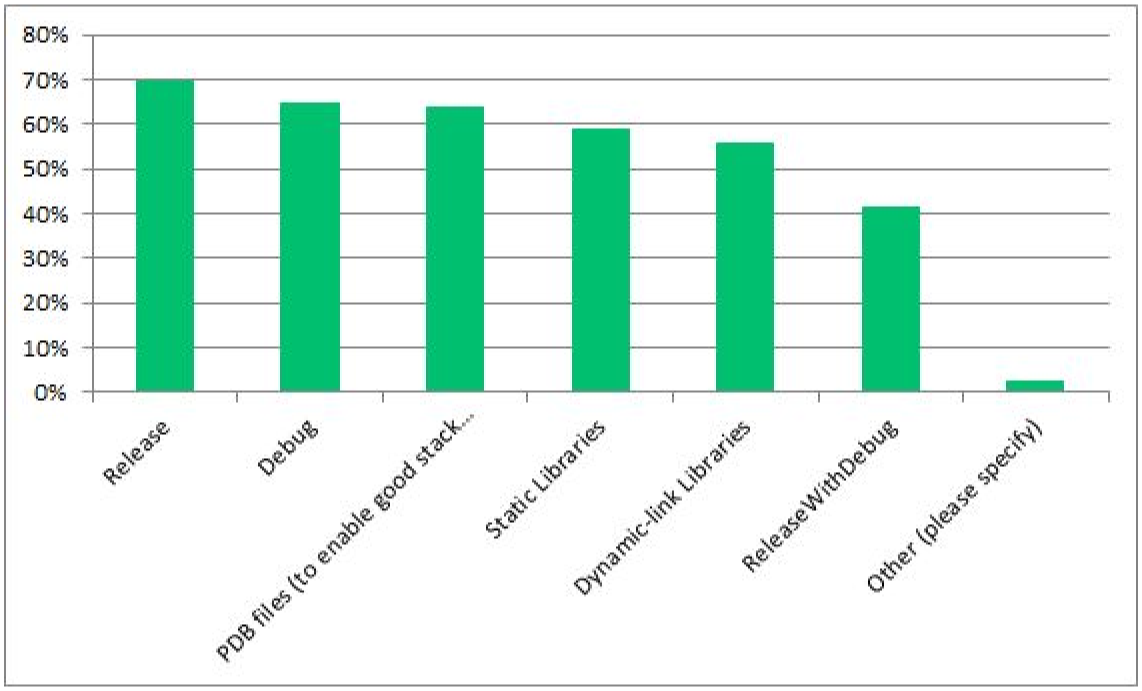
Commercial developers only:



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

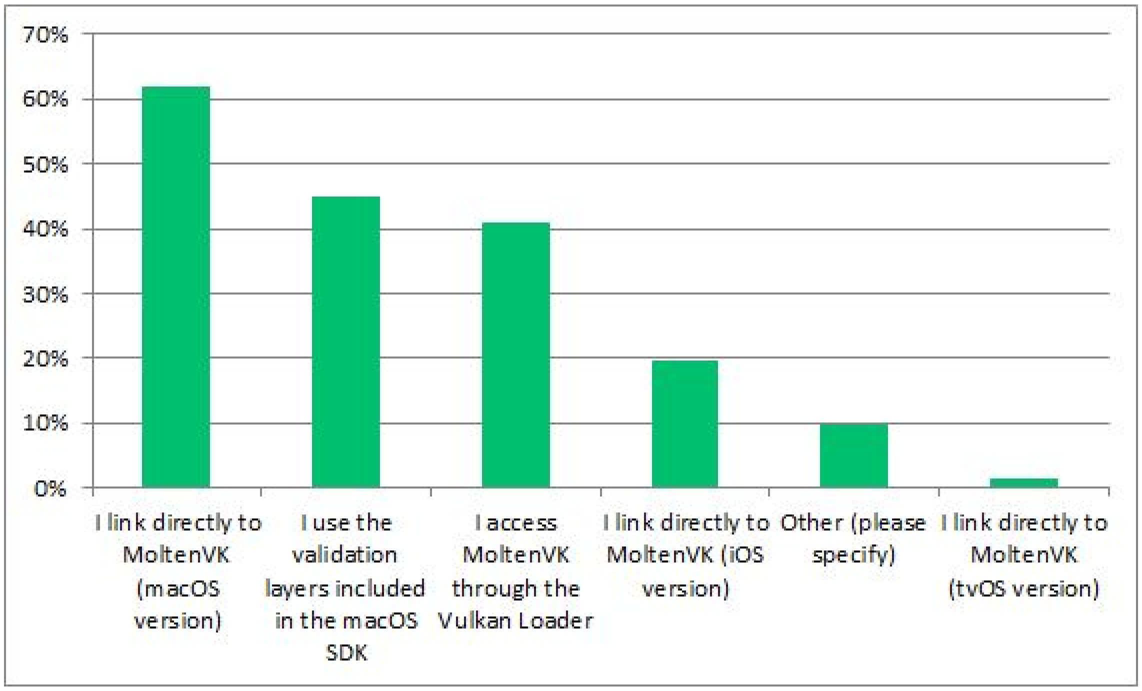


What files or library types do you require for your Windows development environment? Check all that apply:



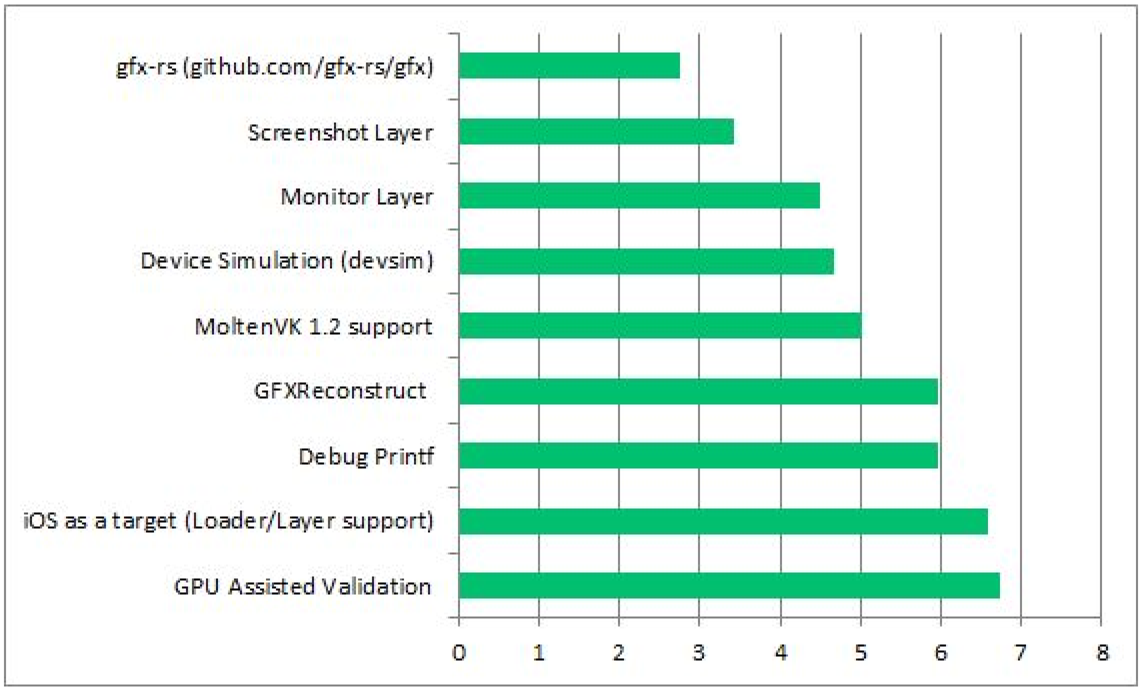
Note: We haven't been including debug and pdb files due to an unacceptable size increase to the Windows SDK. In the future we will be creating separate, selectable packages for various build types.

When doing Vulkan development for macOS, iOS, and/or tvOS, check all that apply.



Note: There are still situations where MoltenVK provides extensions/functionality that can only be accessed by linking directly to MoltenVK instead of using the Vulkan Loader. That may be driving some of the usage of MoltenVK without the loader and validation layers. It is also more likely that developers are doing their validation on a Windows or Linux platform and then using MoltenVK as a driver for their solution on macOS and hence link directly to MoltenVK.

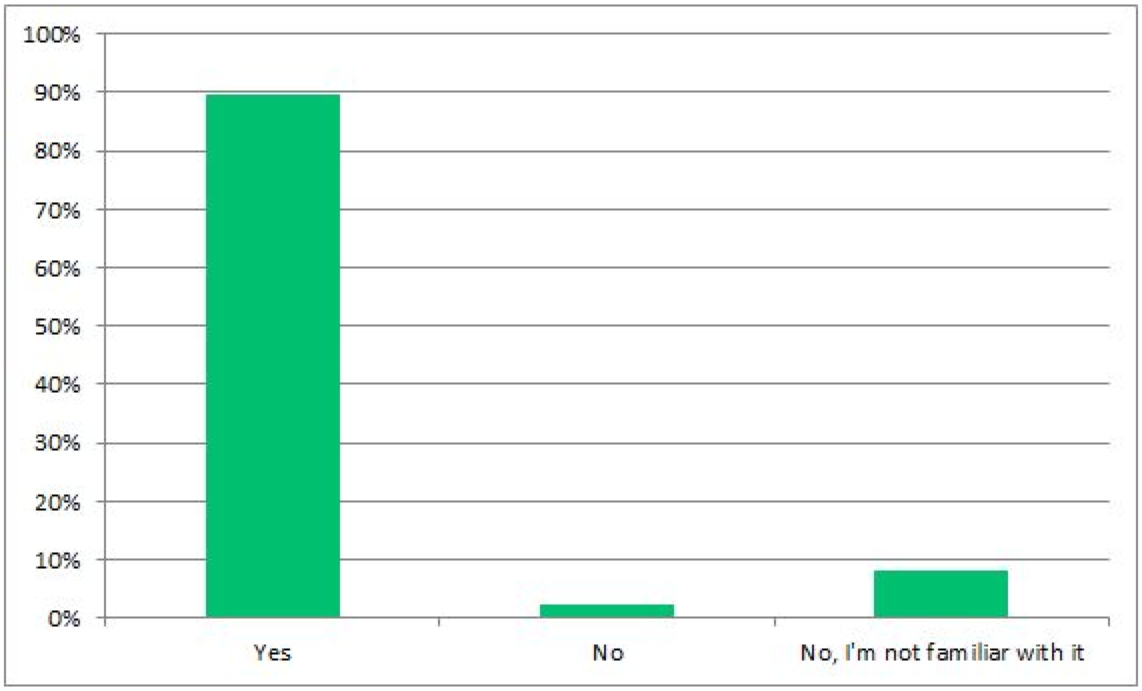
Rank the importance of the following layers/tools in the macOS SDK? (1 is most important, 9 is least important)



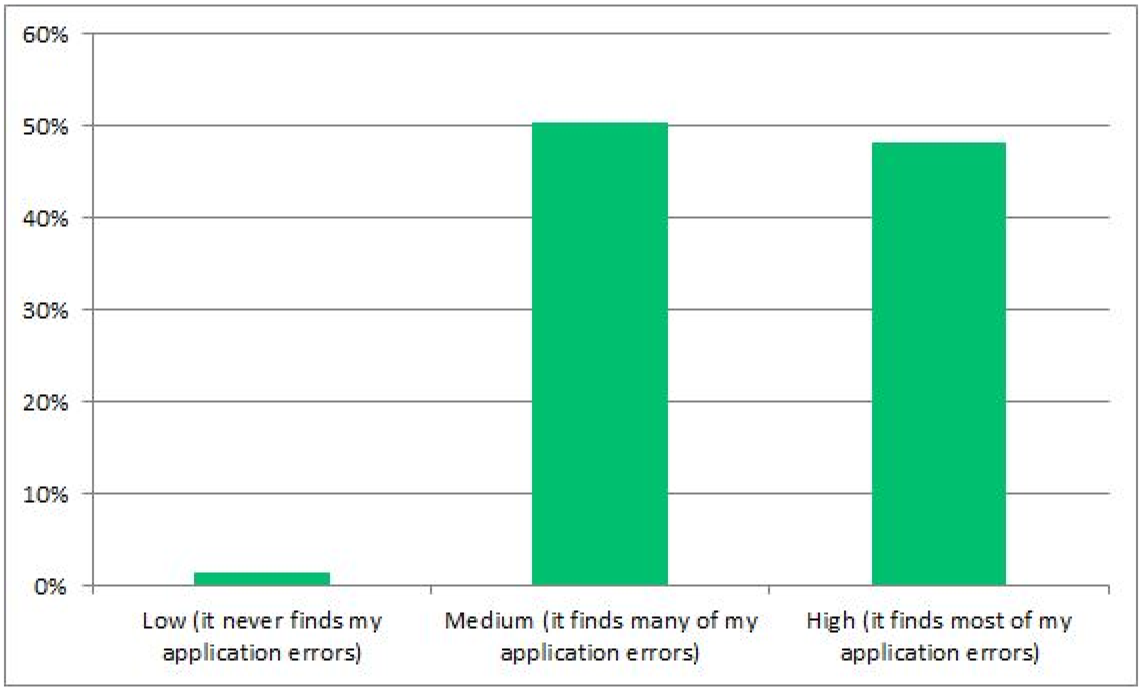
Note: If you only include commercial developers in the responses, the outcome is the same.

GPU-AV and Debug Printf require Vulkan 1.1 + fragmentStoresAndAtomics, vertexPipelineStoresAndAtomics, shaderint64. MoltenVK is at Vulkan 1.1 and supports the Atomics. Adding support for shaderint64 is feasible which could enable future support of GPU-AV with MoltenVK.

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

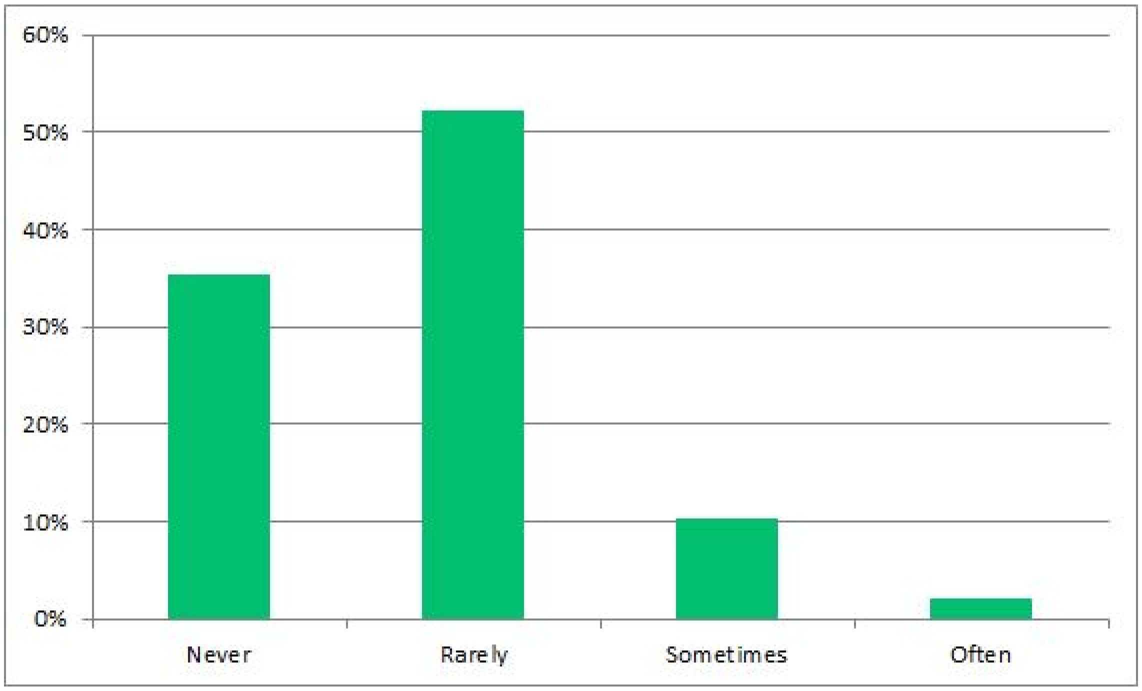


How would you rate the completeness of validation layer coverage?



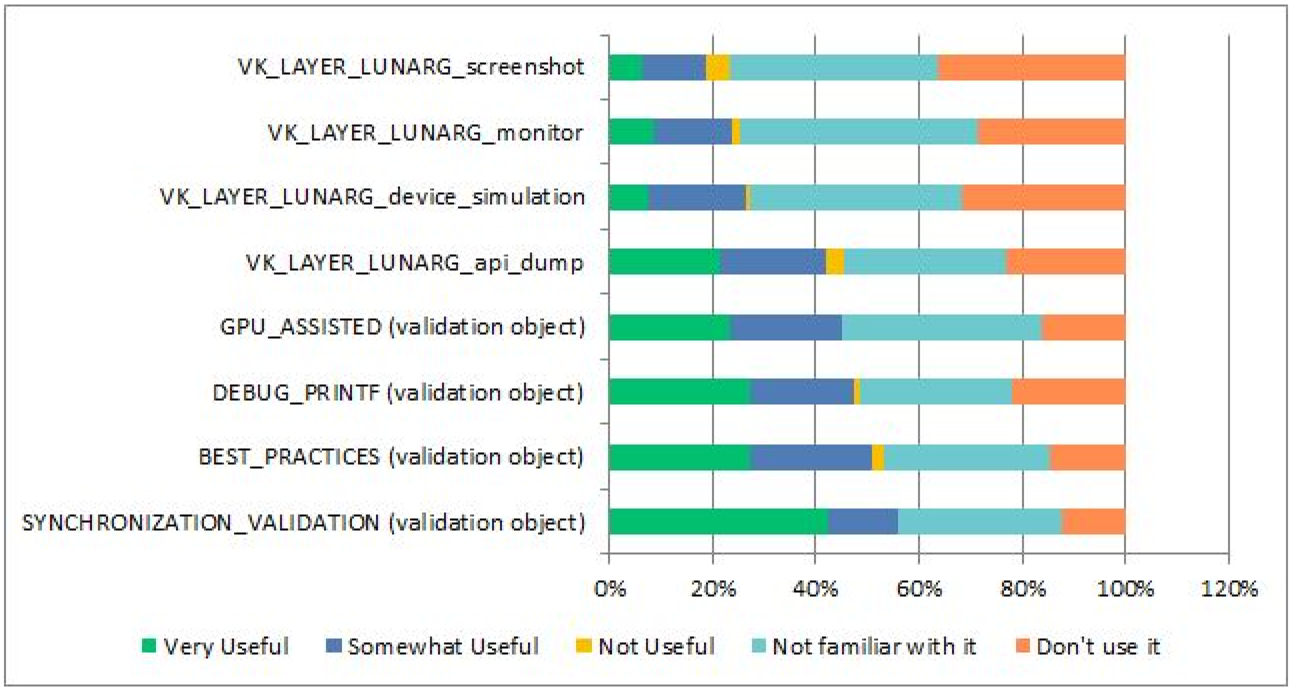
If you only include commercial developers in the answers, there is no change in the results.

How often do you see false error reports (error reported when it shouldn't have been)?



Note: If you only include responses from commercial developers, there is no change in the results.

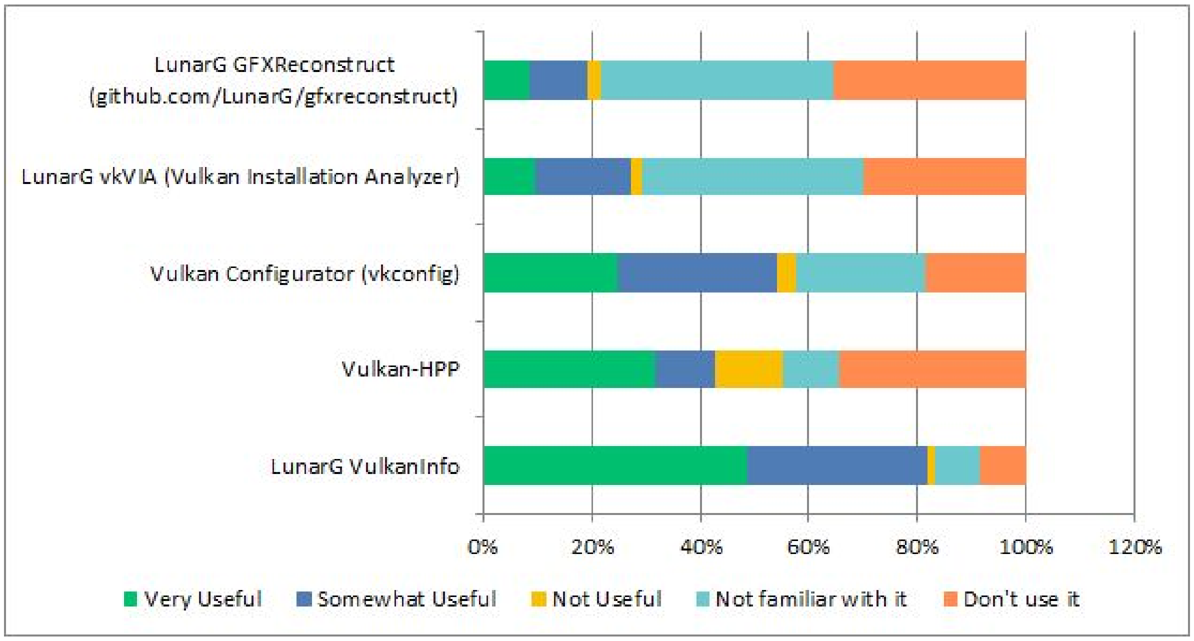
For the Vulkan layers or Validation Layer objects listed below, indicate their usefulness:



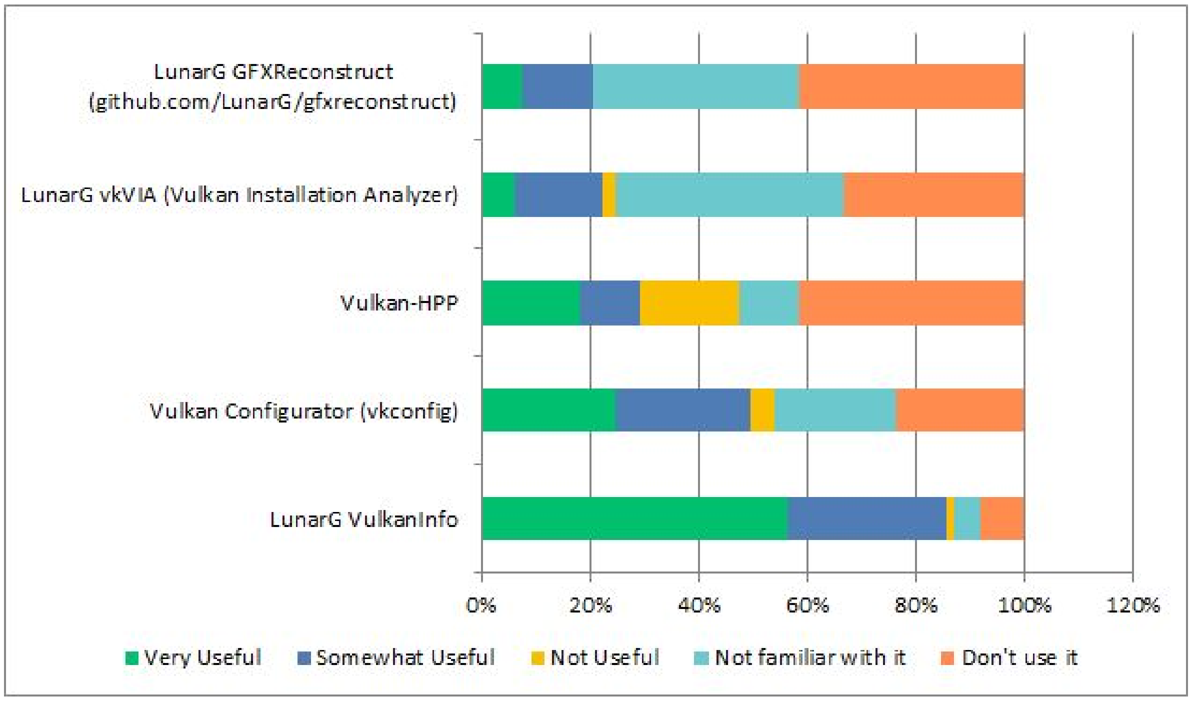
Note: If you only include commercial developers in the responses, the outcome is the same. We are very pleased with the usefulness of the synchronization validation. The investment there has been significant and very useful for Vulkan developers.

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

All responses:

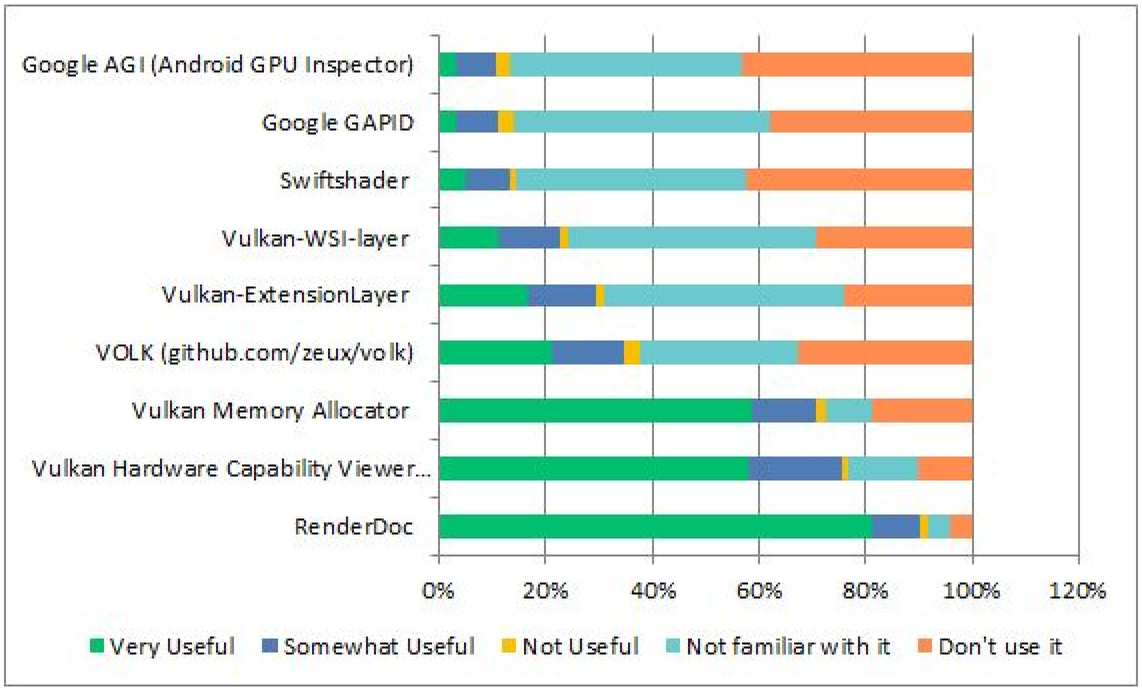


Commercial developers only:



For the Vulkan ecosystem tools/layers (not included in the SDK) that are listed below, indicate their usefulness:

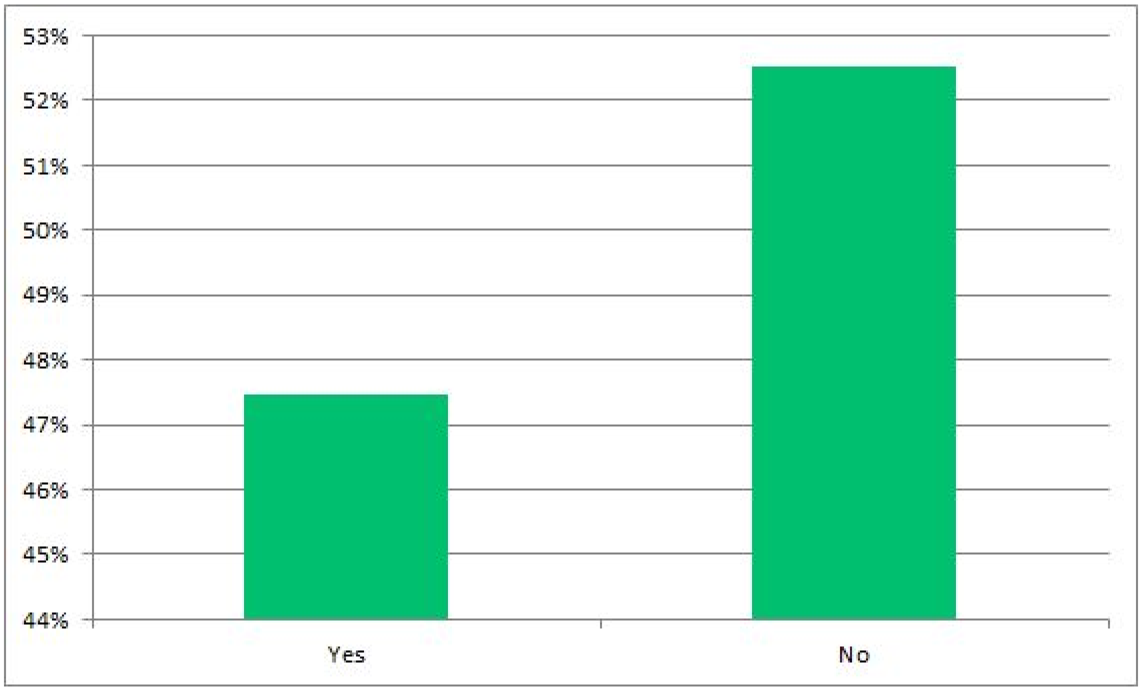
All responses:



Commercial developers only: No significant difference

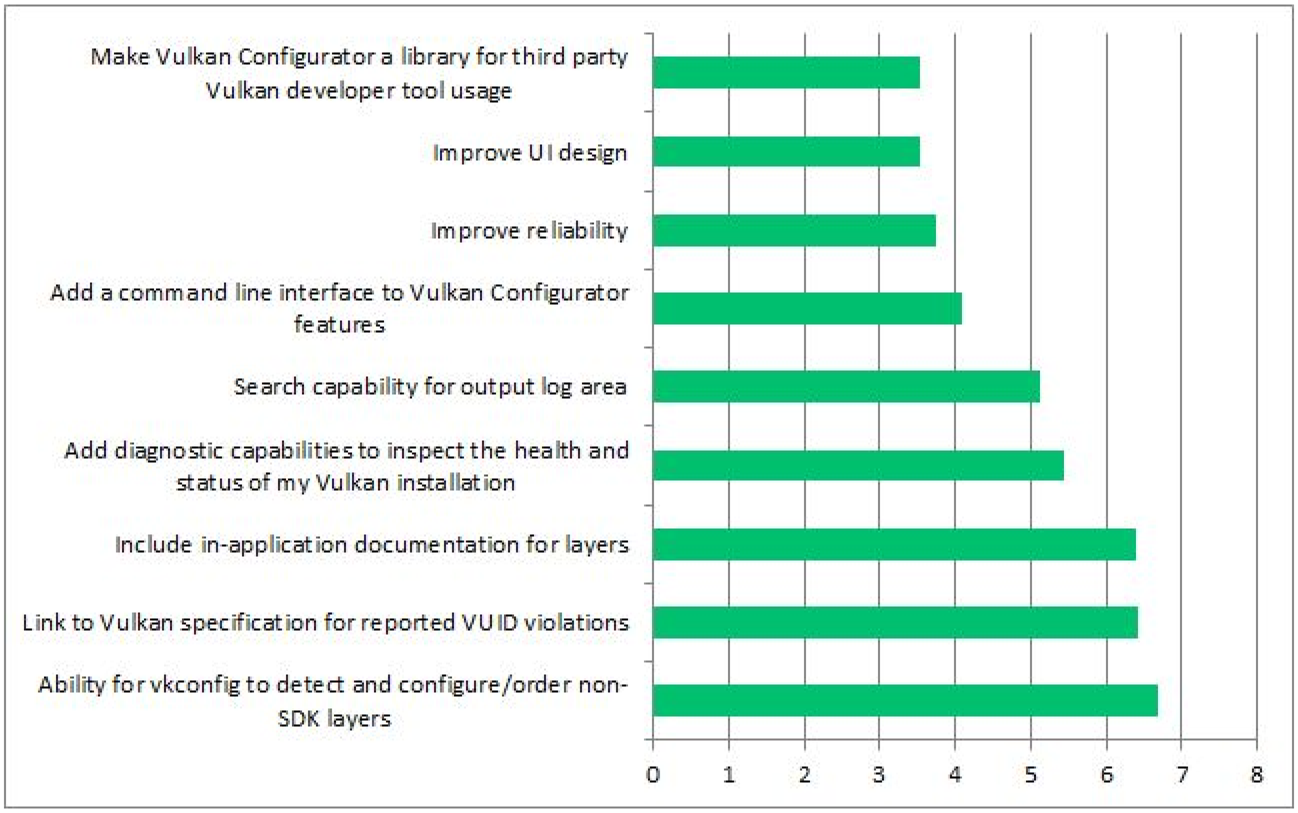
.

Are you familiar with the Vulkan Configurator (vkconfig)?



Note: Although awareness of the Vulkan Configurator is at 47.5%, this is high considering it was just released with more robust support in the summer of 2020. We anticipate the usage of this tool to continue to increase in the next year due to its usefulness and planned continued enhancements. More information about the Vulkan Configurator is documented in the SDK.

Rank the following vkconfig improvement areas from 1 to 9 (1 is most important, 9 is least important)



So it turns out that the top requested feature to the Vulkan Configurator is already LunarG's top priority and is actively being developed. The solution will rely on JSON for layer developers to expose the layer settings.

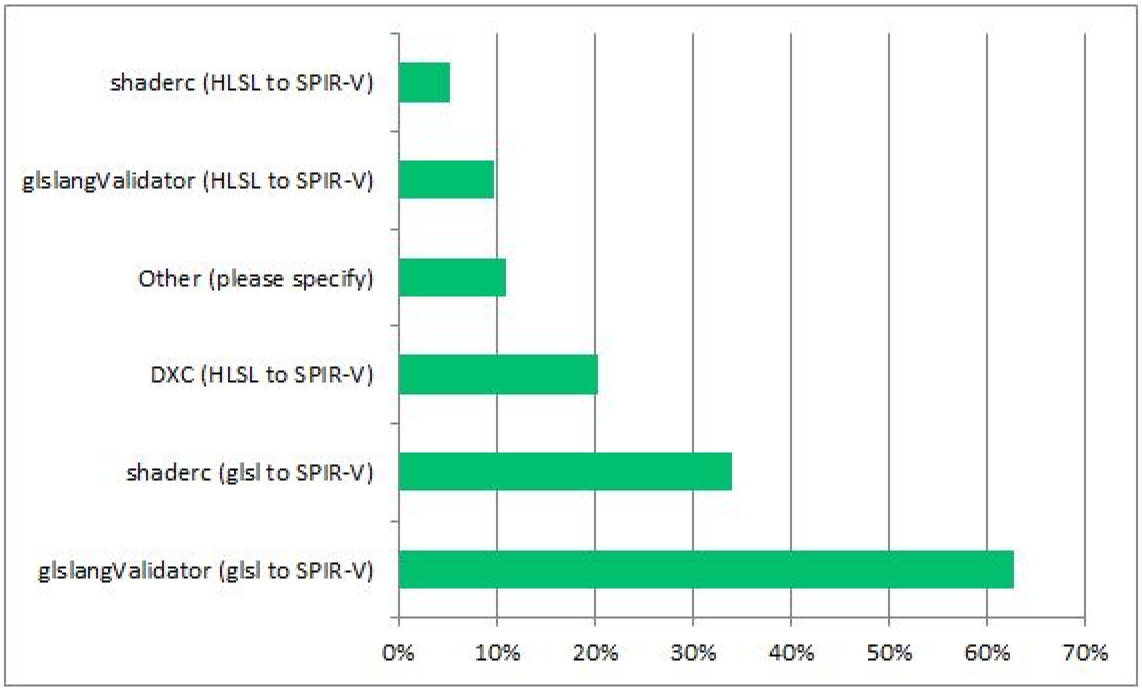
As of the December SDK release, Vulkan Configurator provides a command line interface to override layers. This feature could be particularly useful for continuous integration testing.

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

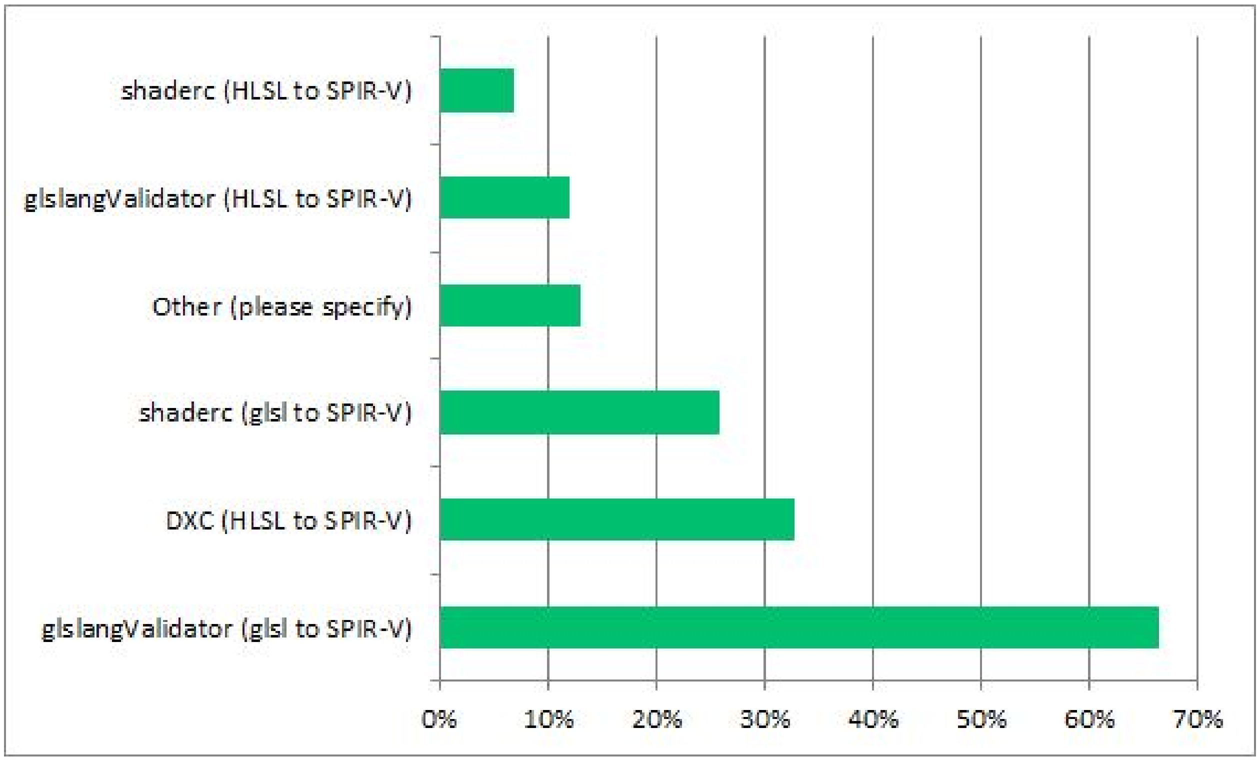
Open ended feedback indicated that the API to enable layers is quite confusing and that the Vulkan Configurator was able to resolve this issue by interpreting the layer settings for the user. The comments in the survey also emphasized the need for better in-application documentation of the settings. Work has already begun to add some of these in-application documentation.

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

All responses:



Commercial developers respondents only:

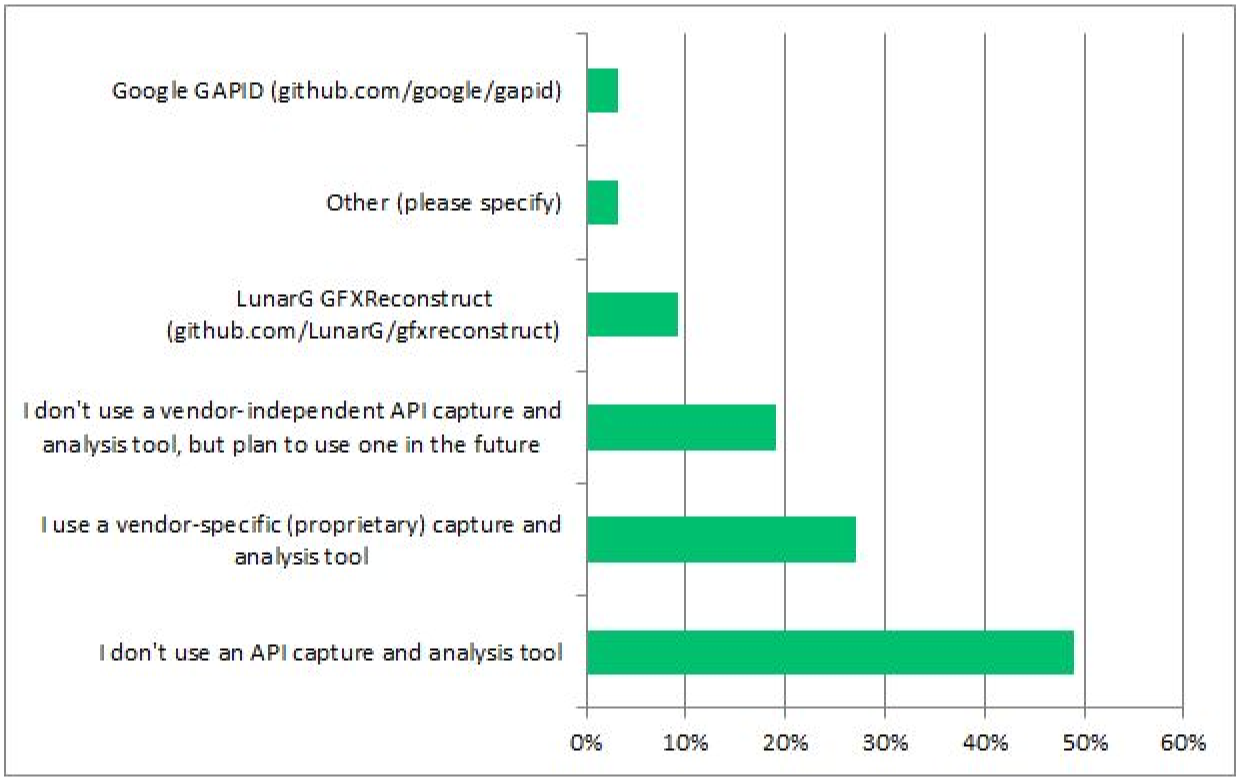


If you use glslangValidator (HLSL to SPIR-V), why do you use this tool instead of using DXC (DirectX Shader Compiler)?

The major themes were:

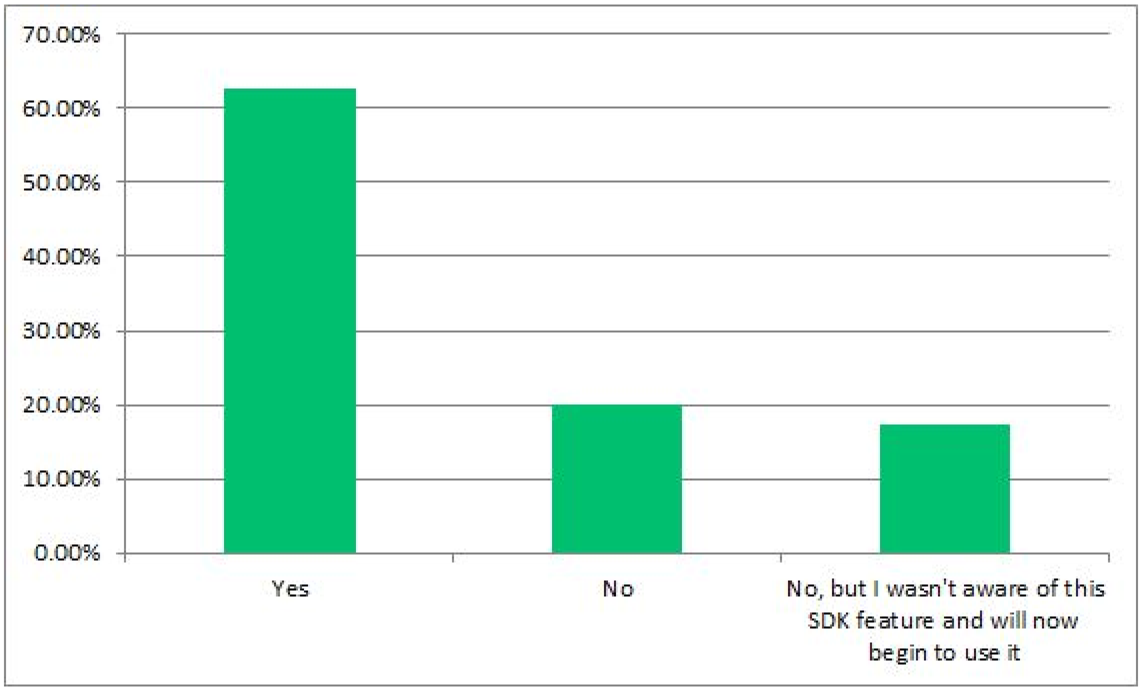
1. glslangValidator is smaller and it is familiar
2. Precedence (DXC was not included in SDKs until the summer of 2020. Note: there were some folks who were not aware that DXC is included in the SDK (Linux, Windows, and macOS)
3. Most of my shaders are GLSL
4. I need a library version (not just offline executable) for DXC

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

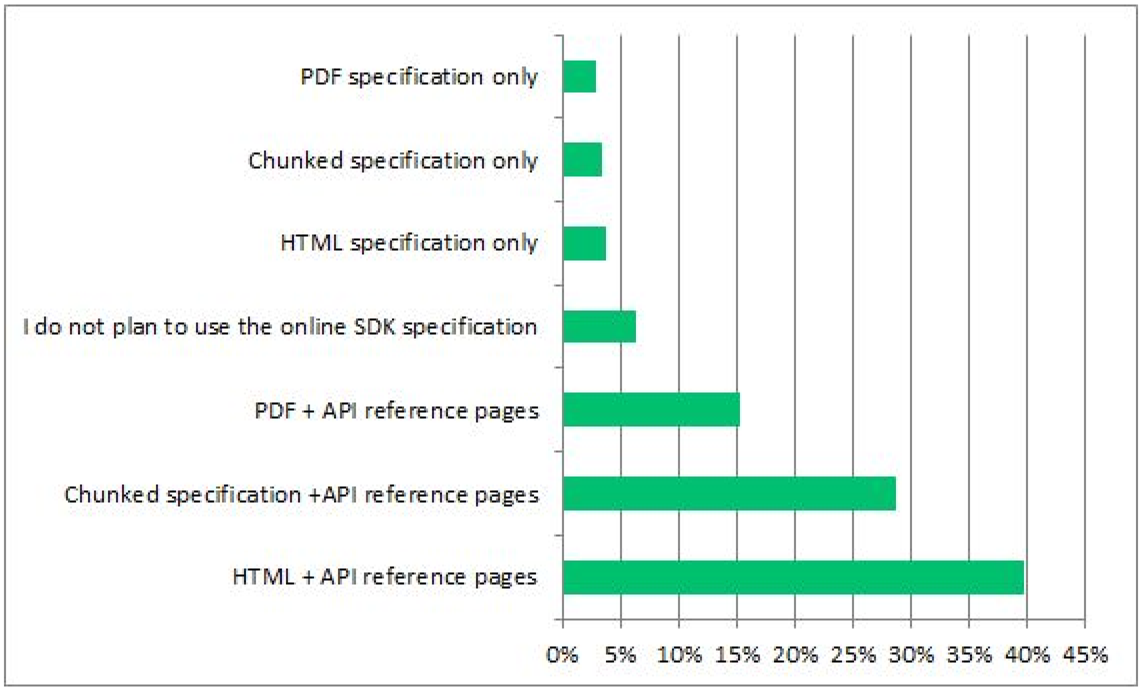


Note: If you only include commercial developers in the responses, the results are almost equivalent.

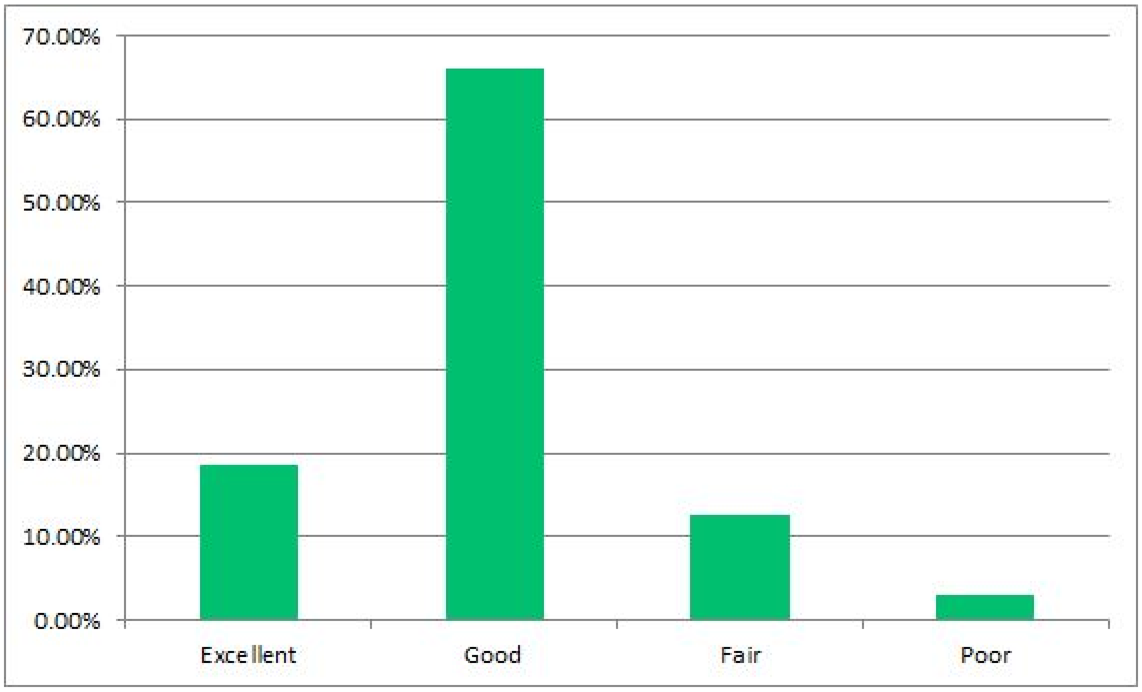
The online SDK documentation (vulkan.lunarg.com/doc/sdk) contains a specification built for the SDK header version. Do you use the online SDK specification?



For the online SDK specification documentation, which is your preferred selection of formats? (Select one)



How would you rate the overall quality of the Vulkan ecosystem today?



Open-Ended Feedback

In the survey there were multiple places to provide some open-ended feedback:

1. What tools would you recommend for the macOS SDK?
2. How could the validation layers be improved?
3. What features, tools, and/or improvements would you like to see added to the LunarG Vulkan SDK (Windows, Linux, and/or macOS)?
4. If your rating of the overall quality of the Vulkan ecosystem is fair or poor, what are your pain points?
5. What inhibits you from effectively and efficiently developing Vulkan applications?

All of the comments were grouped into the following categories and reviewed.

1. API and API Complexity
2. Specification
3. Tutorials/Samples/Developer Documentation
4. Vulkan-HPP
5. macOS
6. Developer tools, ecosystem needs/pain points
7. Validation Layers
8. Shader Tool chain
9. SDK

All of the comments were shared with the Khronos working group and ecosystem project owners. For each topic area the key themes of the feedback is provided below.

API (Feedback was shared with the Vulkan Working Group)

1. Supporting both desktop and mobile under one API added complication.
2. Would love to have a video Encode/Decode solution.
3. The verbosity of the API slows down development
4. It’s a very complex API
5. Reliable extension availability across multiple platforms
6. Would be nice to have extension grouping via feature levels

Specification (Feedback was shared with the Vulkan Working Group)

1. Despite improvements that have been made, the single html page still performs slowly.
2. Better cross-referencing of information would be appreciated
3. Small guide pages for topics like synchronization, required queue family types, etc. would make it easier.

Tutorials, Samples, and Developer Documentation

This topic area had some of the most comments. Developers are wanting good tutorials, good documentation regarding best practices and common design patterns, and good samples. LunarG resources are focused on the development of the Validation-Layers, Vulkan-Loader, Vulkan SDK, GFXReconstruct, Vulkan-Tools, devsim, apidump, and the Vulkan Configurator. However all of the feedback has been shared with the group of engineers contributing to the KhronosGroup/Vulkan-Samples repository and working on ecosystem documentation. They are reviewing the feedback and will take into account this feedback as they continue their work over the next year. Please take a look at KhronosGroup/Vulkan-Samples to see if there are useful samples there and feel free to submit issues or contribute. In addition there is a Trello project where they can vote for proposed samples at https://trello.com/b/1yFqP8ZJ/vulkan-samples-requests.

Vulkan-HPP (Feedback was shared with the Vulkan-HPP developers)

1. There was a statement about shipping HEAD of the Vulkan-HPP in the SDK not being useful.
    a. This was a misconception. The version of vulkan.hpp shipped in the Vulkan SDK comes from KhronosGroup/Vulkan-Headers
2. Vulkan-HPP provided a better coding experience than the standard Vulkan C API. Better promotion of Vulkan-HPP and showcasing it's proper use in official tutorials would be helpful.
3. Concern areas for Vulkan-HPP
    a. Newer versions of Vulkan-HPP in the Vulkan SDK frequently caused a breakage.
    b. The size of Vulkan-HPP

macOS (Feedback was shared with the MoltenVK developers)

1. There were multiple folks with concerns about the need to link directly against MoltenVK to get access to the MoltenVK API and extensions. And that if you wanted to use the Vulkan Loader and Validation Layers you would lose access to the MoltenVK API and extensions.
2. The documentation between the macOS SDK and the KhronosGroup/MoltenVK README documentation in regards to the runtime is inconsistent and needs clarification.
3. Installation and usage of the macOS SDK can be confusing and needs some clarification.
4. People want more Vulkan extension support in MoltenVK (such as ray tracing)
5. There was a request for RenderDoc inclusion in the macOS SDK. Renderdoc is not supported yet in a macOS environment.

Developer tools, ecosystem needs/pain points

1. Ecosystem is fragmented in terms of tools and APIs.
2. A desire for additional bindings to the Vulkan API such as C# and Rust
3. More helper and utility libraries are needed
    a. Such as an EGL-like library for setting up a reasonable swapchain
4. No software Vulkan implementation exists.
    a. This is not true actually. There is a software implementation of Vulkan called “Swiftshader" available at https://github.com/google/swiftshader. In addition Mesa 20.3 supports Vulkan software rendering via VALLIUM and llvmpipe.
5. A continued need for a profiler that can be used across platforms and GPUs
6. Some way (e.g. layer) to enable mocking vulkan calls (for testing boring things like allocators in contrived settings (different memory types, for example)); non-coherent memory type simulation layer, this would help when targeting some android systems
7. An automatic check or a correction for memory alignment in buffer objects.
    a. The validation layers already do checks for correct memory alignment. If you find a case where an incorrect memory alignment is not detected, please submit an issue to the validation layers repository.

Validation Layers

There were many, many comments and LunarG read every one to make sure it was already logged in our github issue tracker or on our TO DO list. One of my favorite comments was: “Honestly, keep on trucking - the validation layer is awesome”. We take pride and work hard to continuously improve this important component of the Vulkan ecosystem. And we do acknowledge there is still room to improve.

Here is a list of some of the comments that have some specific responses that may be useful for developers:

1. Coverage
    a. There was a request for more clear documentation of the gaps in validation layer coverage. This exists today. With each SDK, a coverage report is generated. Here is the html version of this report that is included with the 1.2.162.0 SDK (https://vulkan.lunarg.com/doc/sdk/1.2.162.0/windows/validation_error_database.html). There is also a downloadable CSV version for each SDK version.
    b. There was a request to have all extensions validated. The Vulkan Working Group and LunarG ensure that any EXT or KHR extension is validated by the validation layers as they are released. Vendor specific extensions are not monitored by LunarG or the working group for validation layer coverage. This would be the responsibility of the IHV.
    c. There was a request for more validation inspection of buffers and images. This is done as needed when adding VUID support for particular VUs to the validation layers.
    d. More validation for descriptor indexing was requested. This is logged in the enhancement list for GPU Assisted validation (https://github.com/KhronosGroup/Vulkan-ValidationLayers/projects/3)
2. Error messages
    a. A “friendly debug name” was requested for objects in validation errors. This already exists for object naming in the validation error reporting.
    b. There were numerous comments about how validation errors should be more immediate to help identify a more exact place of the error.
        i. It is very difficult to do this. In Vulkan, it is valid to create 'invalid' objects if they are not used. For this reason, much validation must be deferred until submit-time.
    c. Create a stack trace of the error’s origin (API call) when validation is done at queue submit time.
        i. This is currently tracked as an enhancement to the message reporting in the validation layers error message improvements project (https://github.com/KhronosGroup/Vulkan-ValidationLayers/projects/2)

    d. There was a request to have debugPrintf messages have their own type flags, instead of being bundled with the “info” flag, so they can be enabled without having to enable the other “info” messages. This has been logged as a potential enhancement request in the project for GPU Assisted validation (https://github.com/KhronosGroup/Vulkan-ValidationLayers/projects/3)
    e. There was a request to include the validation layer source code line in the error message. This existed in the past but was intentionally removed due to user feedback. However VUIDs are unique and you can use the VUID in the error message to search the source code.
    f. It was mentioned that in Windows 10, the messages printed to the console can have a weird format and that adding a newline for every sentence would be useful. This is the intention and purpose of using the VK_EXT_Debug_utils extension.
    g. Request: extensions for common debuggers such as Visual Studio that supports validation layer errors being displayed through the debug output window
        i. This is currently supported through the VK_DBG_LAYER_ACTION_DEBUG_OUTPUT layer setting. This setting is defined in the vk_layer_settings.txt file found in the Vulkan SDK in the Config directory. You can also enable this feature using vkconfig (select the “Debug Output” under Debug Action for any of the Validation Vulkan Layer Configurations).

3. Improve the performance of the validation layers.
4. Synchronization Validation
    a. Synchronization validation layers have been incredibly useful for application developers. People want more.
5. Warnings/Best Practices
    a. There were multiple requests for “Best Practice” warnings such as:
        i. Warn on deprecated function usage
        ii. Some insight into common ""mis-patterns"" or unusual behavior
        iii. Some insights into potential performance warnings
    b. The BEST PRACTICES validation layer object covers many of these items to some degree. More information on the BEST PRACTICES can be found here: https://vulkan.lunarg.com/doc/sdk/latest/windows/best_practices.html

Shader Tool Chain

1. Multiple concerns about the shader toolchain (glsl/hlsl) in terms of functionality and quality.
   a. The shader toolchain folks would ask anyone experiencing problems to open/ping issues on github. Here are the related github repositories:
      i. SPIRV-Cross
      ii. Glslang
      iii. SPIRV-tools
      iv. Microsoft DirectXShaderCompiler
      v. SPIRV-reflect
2. Make sure online compilation in addition to offline compilation is supported well.
3. The addition of SPIRV-Reflect is great. It is a perfect example of a good Vulkan tool. It dones one job. It is lightweight. And it does it well.

SDK

This list represents the most commonly requested features or feedback for the Vulkan SDK.

1. SDK is not as feature rich such as Metal Kit, DirectX TK, or game console SDKs
2. Would like more tools to help with performance tuning
3. Folks indicated they wanted DXC in the SDK. As of August, DXC is in the SDK for Windows, Linux, and macOS
4. SDK is a big “blob.” Breaking it into smaller packages and inclusion of all the build types (debug, static, dynamic, PDB, …) would be more useful
5. Linking to the chunked specification would be nice.
6. Scripts to be able to clone and build all of the repositories included in the SDK.
7. Some simple getting started libraries such as vk-bootstrap would be useful.
8. An automatic check or a correction for memory alignment in buffer objects.
9. Add packages for more Linux distributions such as BSD or Arch Linux.

Key Actions for LunarG

This list is a consolidation of the identified enhancements from the survey feedback. It is not prioritized in terms of what features will be added and in what order. Continued planning inside of LunarG throughout the year will make decisions about which of these features will be added.

1. Add more features to the macOS SDK. Time permitting more functionality will be added to the macOS SDK
    a. GPU Assisted Validation (and Debug Printf)
        i. GPU-AV and Debug Printf require Vulkan 1.1 + fragmentStoresAndAtomics, vertexPipelineStoresAndAtomics, shaderint64. MoltenVK is at Vulkan 1.1 and supports the Atomics. Adding support for shaderint64 is feasible which could enable future support of GPU-AV with MoltenVK.
    b. iOS as a target of the SDK
    c. Port to macOS
        i. Devsim
        ii. GFXReconstruct
        iii. Monitor
        iv. Screenshot
2. For the vulkan configurator, add these additional enhancements:
    a. Ability for vkconfig to detect and configure/order non-SDK layers
    b. Link to Vulkan specification for reported VUID violations
    c. Include in-application documentation for layers
    d. Add diagnostic capabilities to inspect the health and status of my Vulkan installation
    e. Search capability for the output log area
    f. Add a command line interface to Vulkan Configurator features
    g. Continued reliability and UI design improvements
    h. Make Vulkan Configurator a library for 3rd party Vulkan developer tool
3. SDK Additions
    a. DXC API library
    b. VOLK
    c. Vulkan Hardware Capability Viewer
    d. Vulkan Memory Allocator
    e. Create separate packages for debug builds and PDB files for Windows SDK components
    f. Linkage to the Chunked specification from the validation layer errors (note: One format needs to be chosen for this and the HTML format currently being used was requested from a higher percentage of users. So most likely this change will not be made).
    g. Include a script to build all the repository versions included in the SDK.

            i. Note: The release notes for each SDK has a list of branches or commits for all the included repositories. In addition, with each SDK, there is a CONFIG.json that is a programmatic way to determine all the branches and commits. You can read about the API to download this information on the SDK download site (see the SDK version query and download API documentation).
            ii. It would be useful to also create a script to read the contents of this config.json and clone and build all of the SDK repository versions.
    h. Include vk-bootstrap in the SDK as a simple getting started example.
    i. For synchronization validation, create a tool to visualize the synchronization. Both as defined by the specification and what is being actualized by the application.
    j. More tools to help with performance tuning
    k. Packaging for additional distributions such as BSD or Arch Linux.
        i. Note: LunarG resources aren't available to do more Linux packaging for the SDK. It was hoped that the ability to query SDK versions and repository versions for the SDK would enable Linux distributors to create Linux packages themselves.

4. macOS and MoltenVK
    a. The documentation between the macOS SDK and the KhronosGroup/MoltenVK README documentation is inconsistent and needs clarification.
    b. Installation and usage of the macOS SDK can be confusing and needs some clarification.
    c. Have the Vulkan Validation layers compatible/available with MoltenVK extensions at the same time.
5. Vulkan Validation Layers
    a. Improve the performance
        i. In the last 3 months, LunarG has started a performance initiative for the validation layers. The first step was to create a performance regression test suite to prevent future impacts to performance as development continues. Some bottleneck areas have also been identified and will receive some tuning in the months to come. If you have a workload that is showing unacceptable performance while using the validation layers, and you are willing to share it with LunarG, let us know. You can contact us at info@lunarg.com
    b. Continue development on the synchronization validation to fill out more valid usage checks!
        i. Focused work at LunarG continues working on synchronization validation. After a complete triage of all CTS failures with synchronization, the synchronization validation is ready to be removed from "alpha quality" to production quality. This is already available on the validation layer github repository and will be included in the next SDK release to be released in the January timeframe.
    c. Continue filling out validation layer VUID checks

        i. Filling out validation layer VUID checks is always occurring at LunarG throughout the year. The github issues that are submitted as missing VUID checks are categorized and "incomplete" and given top priority since they are missing VUID checks found by actual application developers. Once all of them are complete, the work can begin on filling out additional VUID checks not reported by users.

6. Vulkan Loader
    a. Applications need a way to query all the layers (can do this today), but to also know which layers are implicit and which layers are explicit (can't differentiate between them today). Applications are having issues with bad implicit layers causing crashes in their applications and need a way to query the layers for logging purposes, and possibly disable them.
        i. LunarG intends to take this to the Working Group to get an API defined to enable this querying (and potential disabling of layers by an application).
            1. Define a vkEnumarateInstanceLayerProperties2 (or similar) that will list the layers and also indicate which layers are implicit
            2. Define a programmatic way for users to disable layers at create Instance time, most likely as a pNext struct in vkCreateInstance
        ii. Another feature that will be released soon is "safe mode for layer loading" that will be implemented in the loader.
            1. The current plan is to implement a safe mode that only disables layers that were not explicitly enabled, either through vkCreateInstance or environment variables. This means that explicit layers would behave the same as outside safe mode. Implicit layers would still run when they are enabled with an environment variable (either VK_INSTANCE_LAYERS or the layer-specific envar) but would not ever run purely because they are installed to the system. The goal is to eliminate the layers that no one knows are installed to a system. This would allow tools like steam and renderdoc to still work in safe mode (because both enabled their tools through envars) but would disable layers that aren't being used.