LunarG Vulkan Ecosystem & SDK Survey Results

Karen Ghavam, LunarG
Survey Administered - December 2019
Results Reported - February 2020 - Version 1

Executive Summary

This report provides the results from the LunarG ecosystem survey completed in December of 2019.

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 June of 2018 (third 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+ recipients of the LunarG LunarXchange Vulkan SDK mailing list.
3. Results were shared with the Khronos Vulkan WG since many of the comments were regarding items directly impacted by Vulkan WG deliverables and decisions. Feedback from key ecosystem contributors in the working group was incorporated into this report.

Key points:

1. Vulkan developers are becoming more experienced, which is resulting in feedback and suggestions that are more knowledgeable, constructive, and helpful than in previous surveys.
2. Approximately 50% of the respondents learned of the survey through the LunarG LunarXchange Vulkan SDK mailing list and the other ~50% learned of the survey through other forms of social media.
3. There were 349 respondents.
4. The respondents were a healthy mix of personal use/self study and folks developing Vulkan for commercial purposes (56.4% self-study vs. 43.6% commercial developers) . Respondents were also a good mix of developers for Linux, Windows, Android, and iOS/macOS.
5. Although the Vulkan SDK is desktop focused, Android developers were represented in the survey -- quite a few use the SDK and others have responded after learning about the survey through social media. Of the 100 total Android based respondents, slightly

more than 50% were commercial developers, slightly less than 50% were self-study. None were academics.

6. The survey respondents overall gave a distinctly favorable score on the quality of the ecosystem.
7. The top categories (in order) for inhibitors that may prevent productive and effective Vulkan development are:
    a. Complexity of the API
    b. Driver quality and inconsistency
    c. Tutorials/samples/documentation
    d. Tool deficiencies
8. The percentage of respondents developing for a macOS/iOS environment was higher than the number of respondents developing for Android (34.79% vs. % 29.24)
9. When asked about desired validation layer improvements, the top categories (in order) were:
    a. Improved error reporting
    b. Coverage
    c. Performance
    d. Synchronization validation

List of Survey Questions

Here are the list of questions posed in the survey.

Question 1: How did you find this survey?

Question 2: How would you rate the overall quality of the Vulkan ecosystem today?

Question 3: What type of Vulkan developer are you?

Question 4: How long have you been working with the Vulkan API?

Question 5: What methods have you used to learn the Vulkan API?

Question 6: Your Vulkan development is for what type of industry?

Question 7: Your development is for what type of use case?

Question 8: How much Vulkan development have you done?

Question 9: What is the target of your Vulkan development?

Question 10: What is your Vulkan development environment?

Question 11: How would you rate the completeness of validation layer coverage?

Question 12: How often do you see false error reports (error reported when it shouldn't have been)?

Question 13: How and when do you update your validation layers?

Question 14: Which of these Validation Layer objects do you use?

Question 15: For the Vulkan Layers listed below, indicate their usefulness.

Question 16: For the Vulkan Tools listed below, indicate their usefulness.

Question 17: Which Vendor Independent tool do you use for API capture and analysis?

Question 18: For the Vulkan shader-tool-chain tools listed below, indicate their usefulness.

Question 19: What is your front-end for creating SPIR-V?

Question 20: If you use one of the shader-tool-chain tools in library format, what is your primary build source?

Question 21: What features, tools, and/or improvements would you like to see added to the LunarG Vulkan SDK? (open ended question)

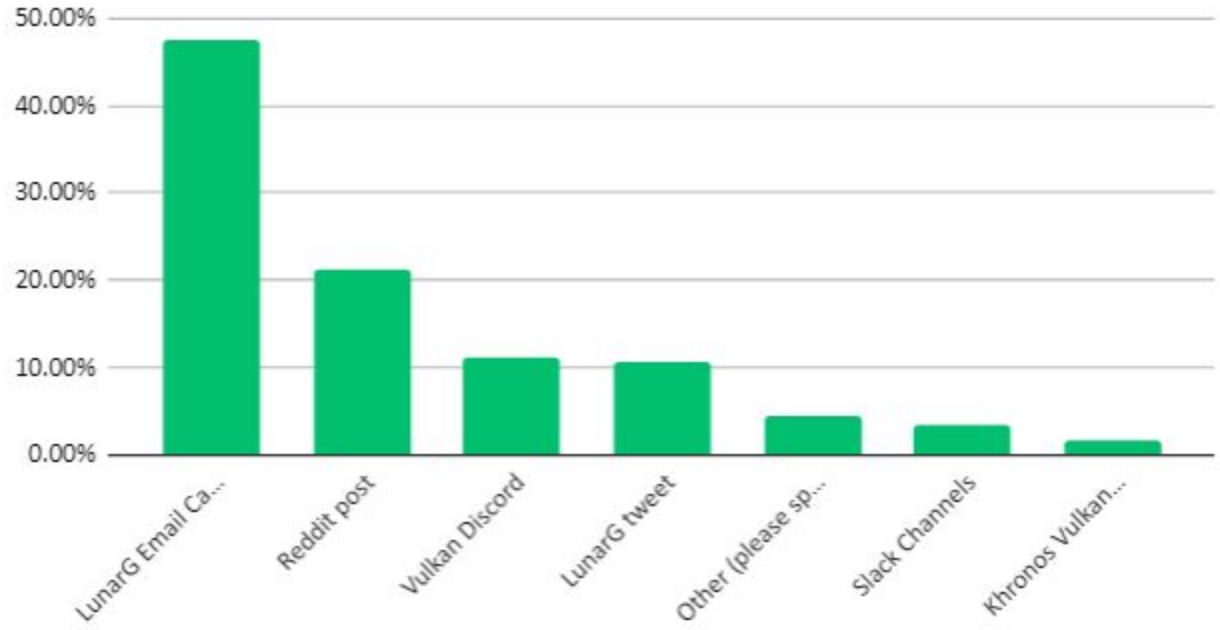
Question 22: How could the validation layers be improved? (open ended question)

Question 23: What inhibits you from effectively and efficiently developing Vulkan applications? (open ended question)

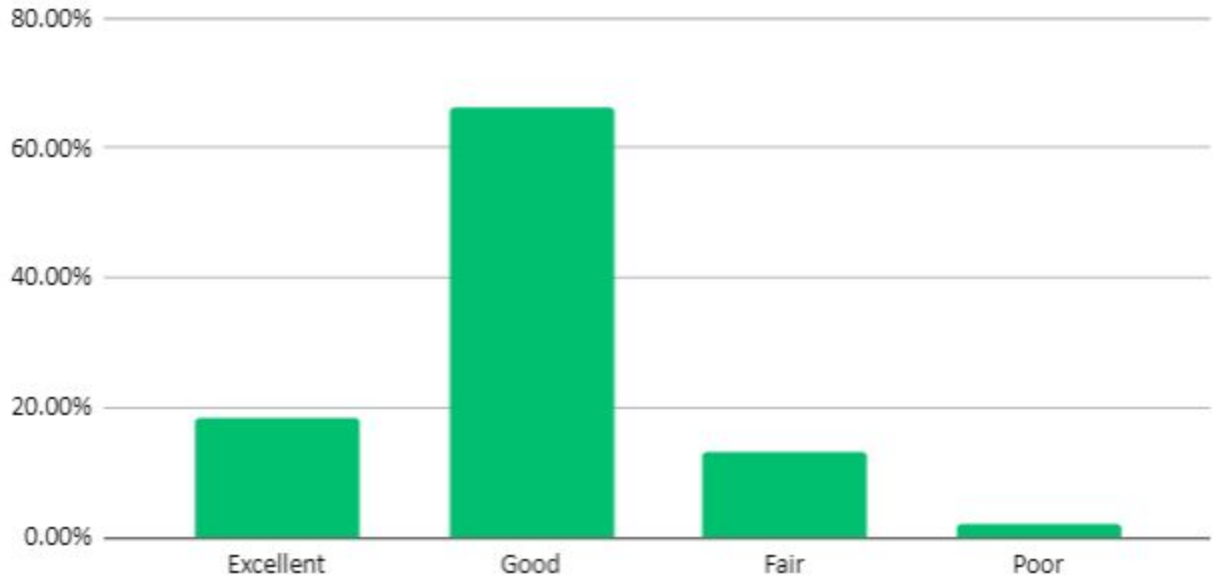
Survey Results and Analysis

In the remainder of this document, you will find each question asked, a bar chart or summary of answers received from the survey respondents, and our thoughts about the survey feedback.

Question 1: How did you find this survey?

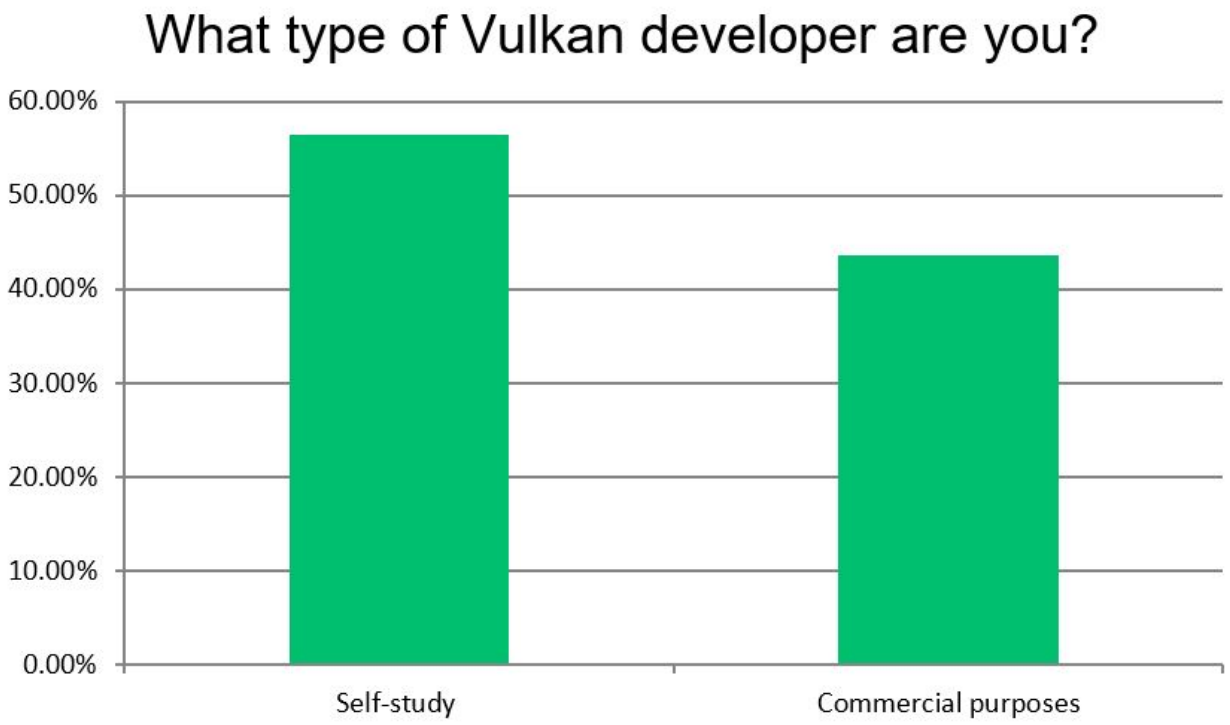


Question 2: How would you rate the overall quality of the Vulkan ecosystem today?



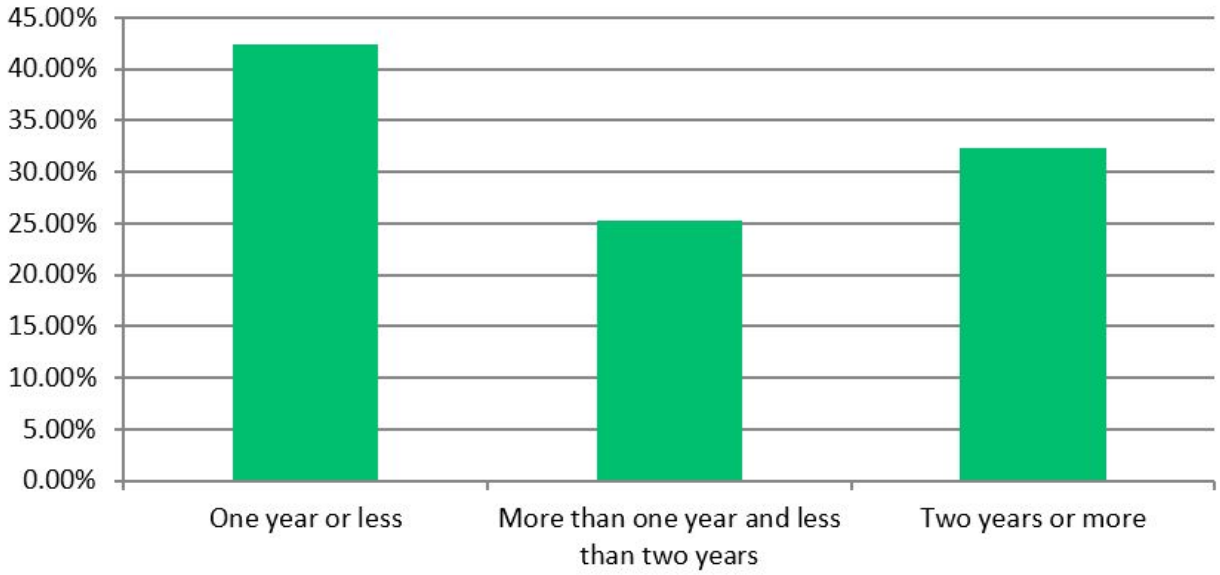
This result is favorable overall for the Vulkan ecosystem and was a bit of a surprise for LunarG. There are still so many opportunities to improve the Vulkan developers experience.

Question 3: What type of Vulkan developer are you?



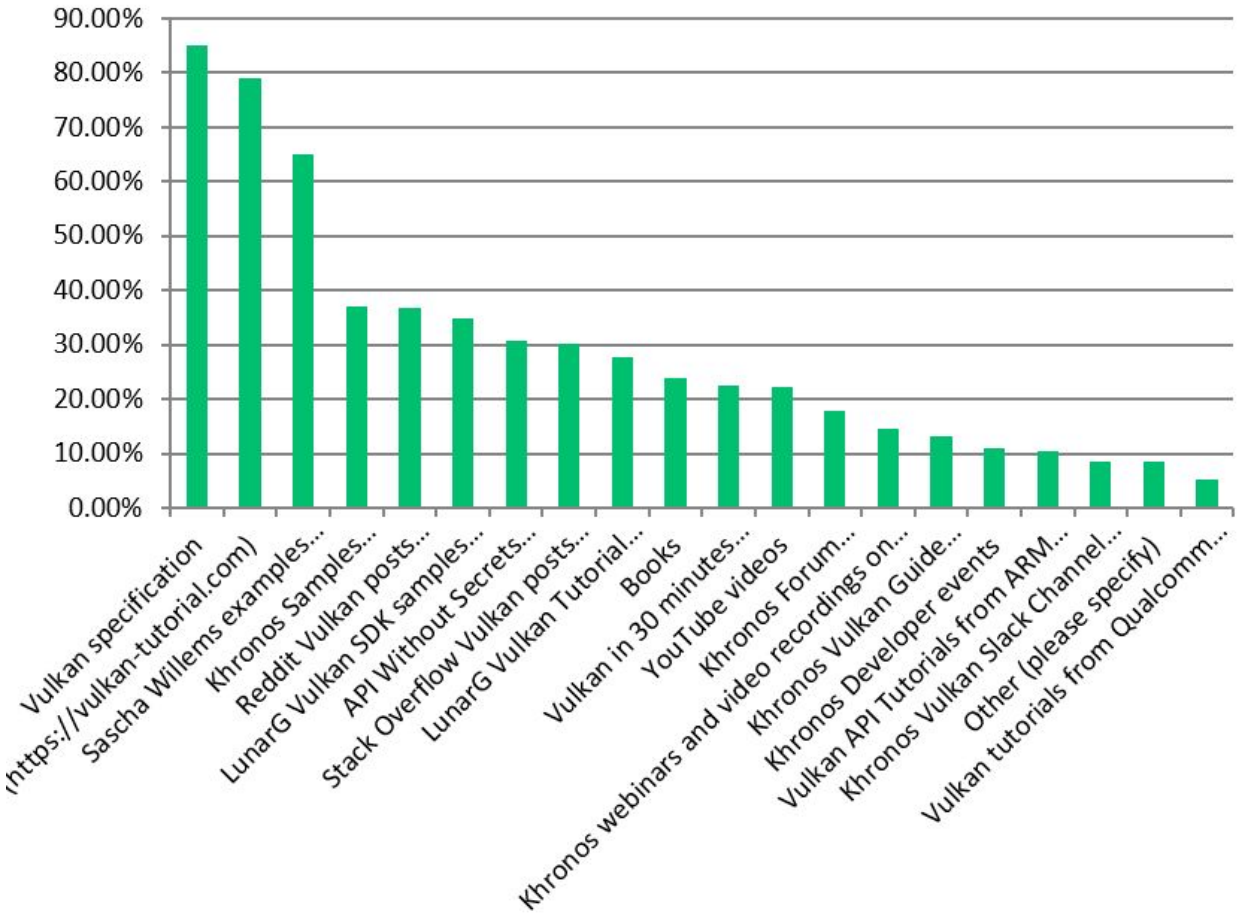
These results show that we have a good mix of survey respondents from both the self-study and commercial purposes categories.

Question 4: How long have you been working with the Vulkan API?



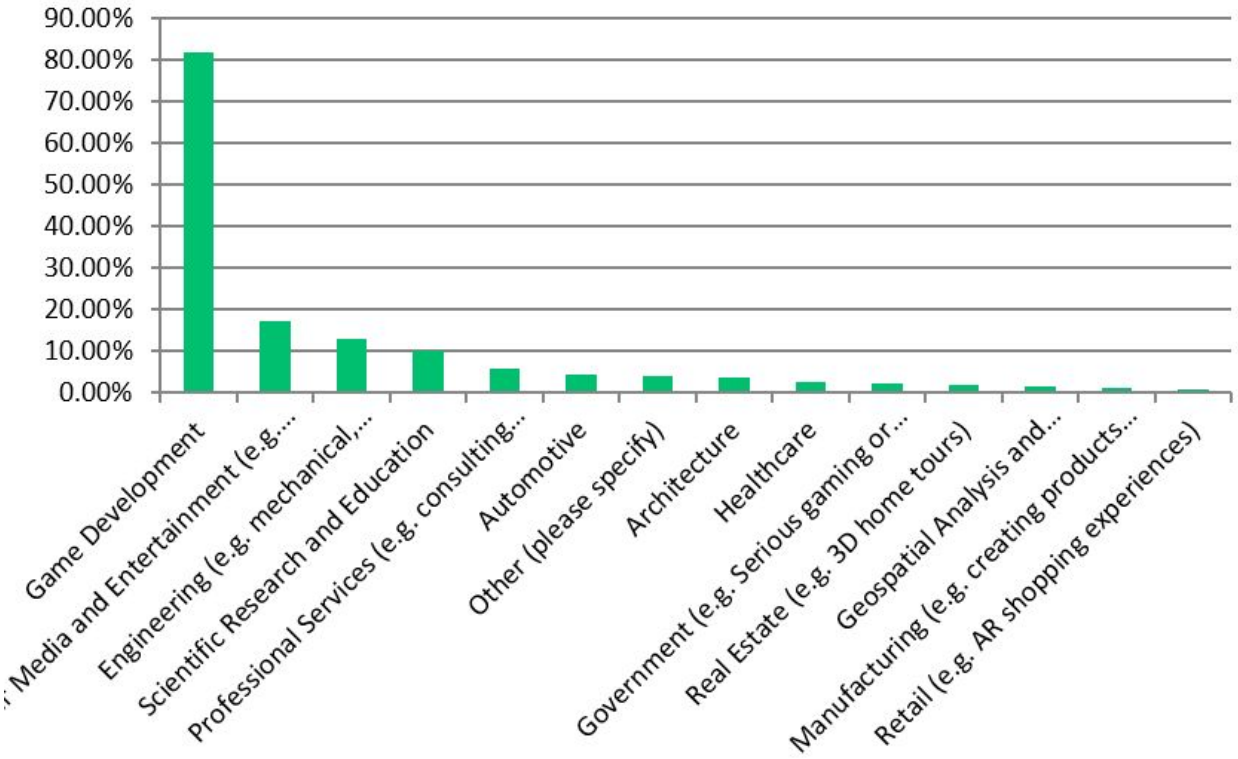
The majority of respondents have been working with Vulkan for more than one year. This improves the quality of the open ended responses later on in the survey. The open ended responses are now showing levels of competence and understanding that enable constructive and informative feedback.

Question 5: What methods have you used to learn the Vulkan API? Check all that apply:



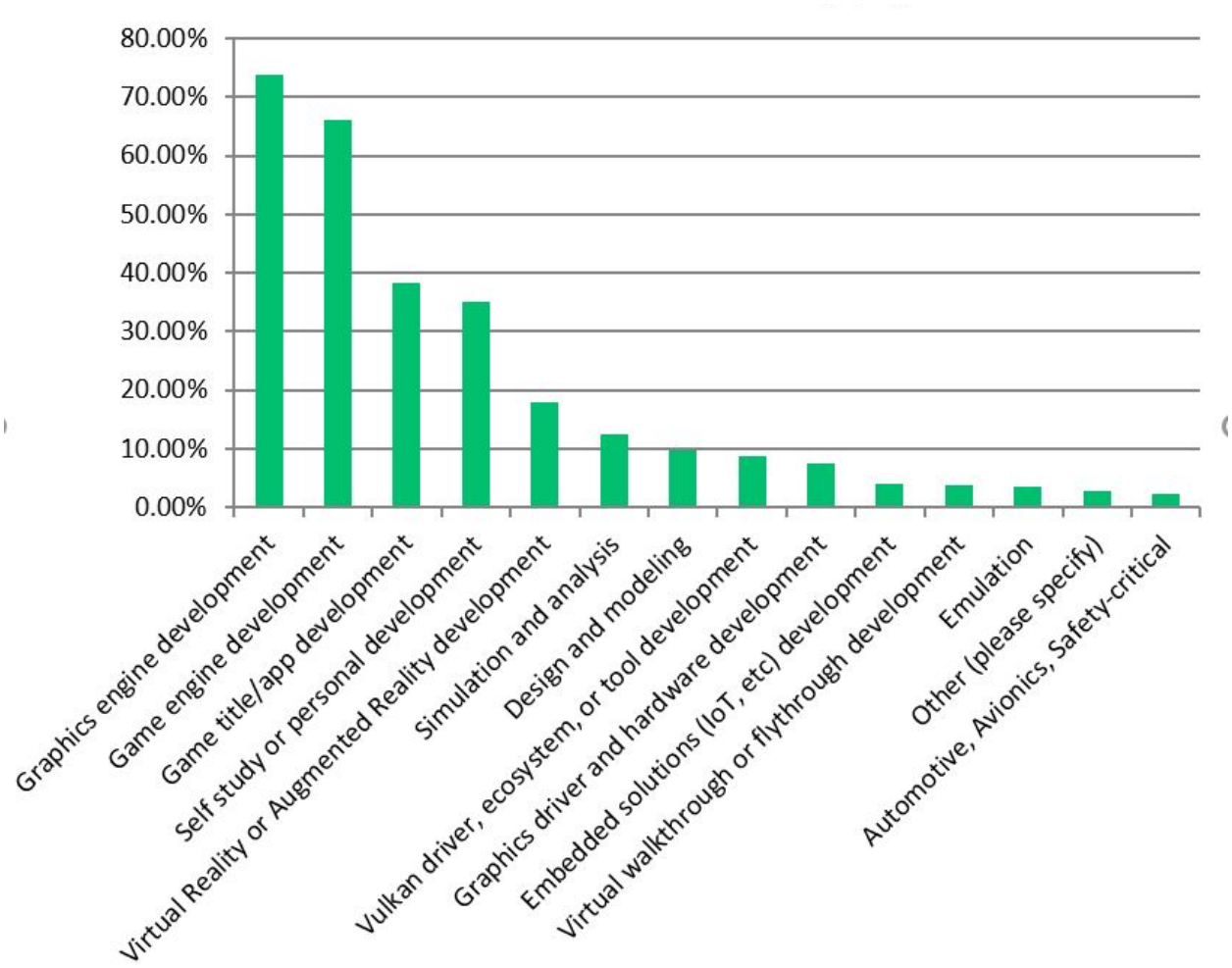
The Khronos Samples repository has only been released for a couple of months and already it is approaching the Sascha Willems samples in popularity. The LunarG Vulkan samples are still pretty popular as a learning tool. With the creation of the Khronos Samples repository, LunarG needs to determine what to do with the LunarG samples repository in the long term.

Question 6: Your Vulkan development is for what type of industry? Check all that apply.

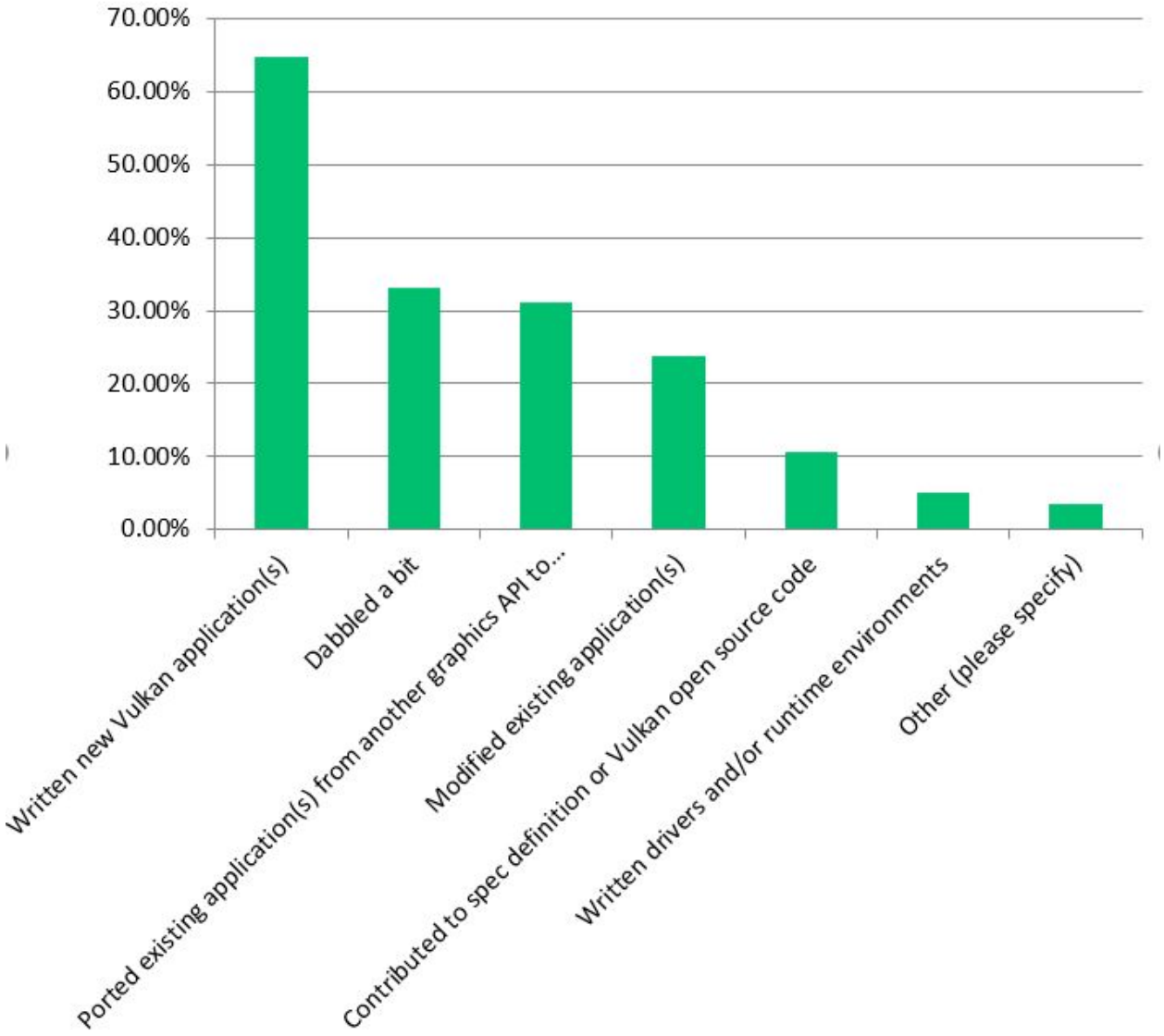


This question is asked to see if Vulkan adoption will expand beyond the gaming industry. The results this year are still very similar to previous year results.

Question 7: Your development is for what type of use case? Check all that apply:

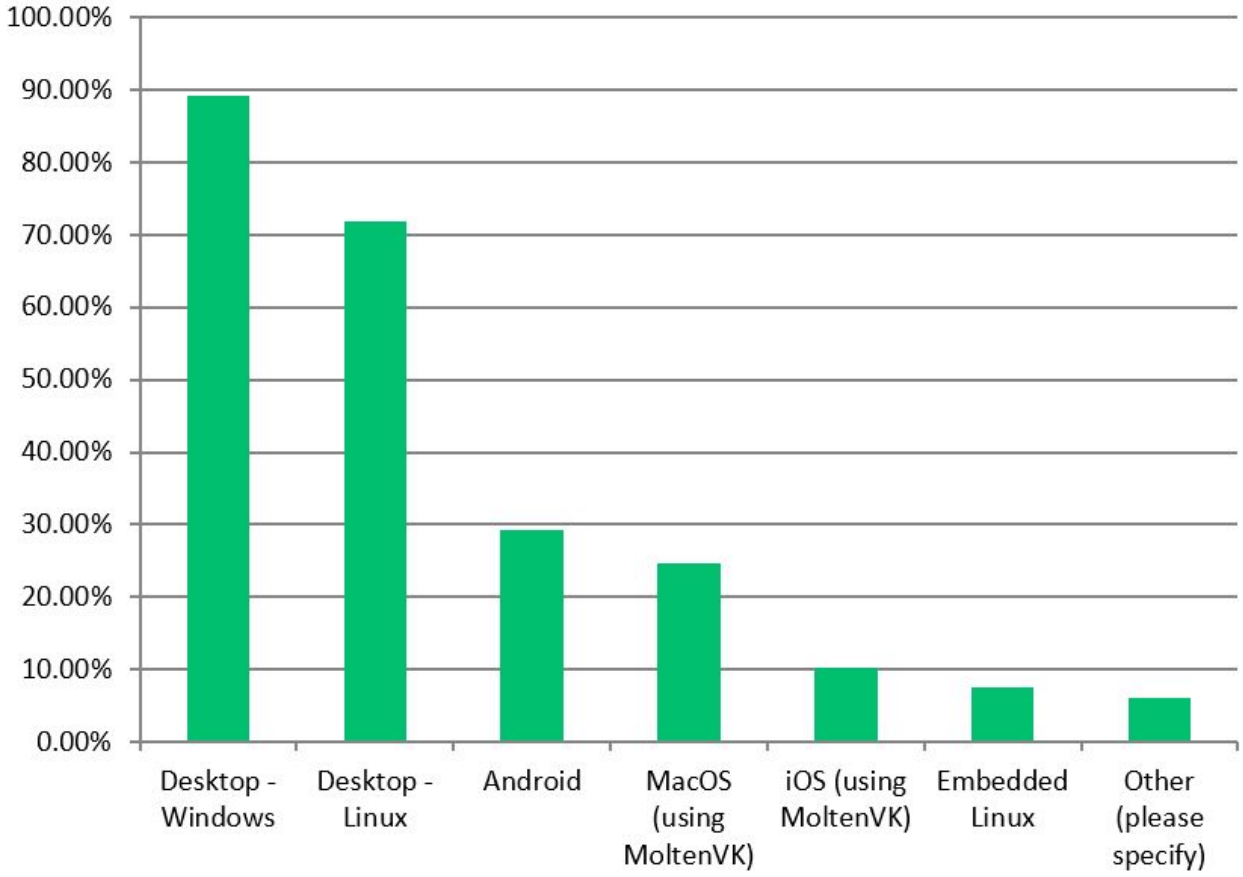


Question 8: How much Vulkan development have you done? Check all that apply:



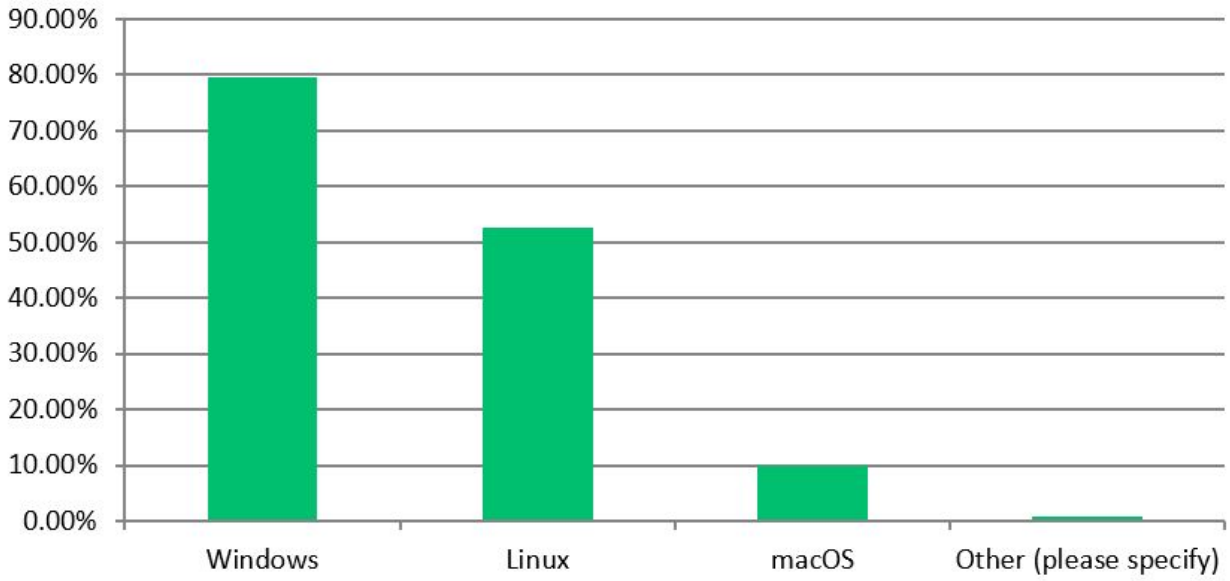
There is an increase in the percentage of individuals who have written new Vulkan applications. Previous surveys were running around 40%. It has increased to 65%.

Question 9: What is the target of your Vulkan development? Check all that apply:

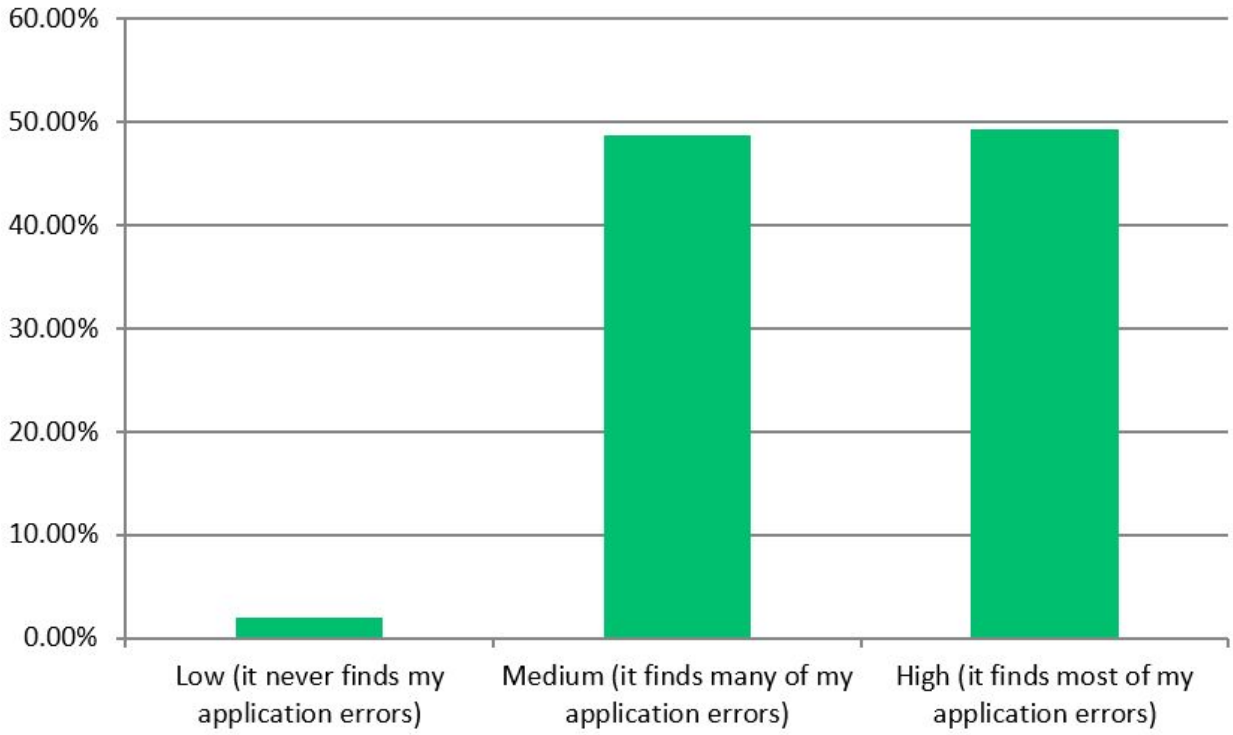


Note: Our SDK was intended for desktop applications (however there are folks who took the survey and are not SDK users).It is evident that there are some Android developers using the SDK. It should be noted that many Android developers may be developing on Linux or Windows and then later testing on Android.

Question 10: What is your Vulkan development environment? Check all that apply:

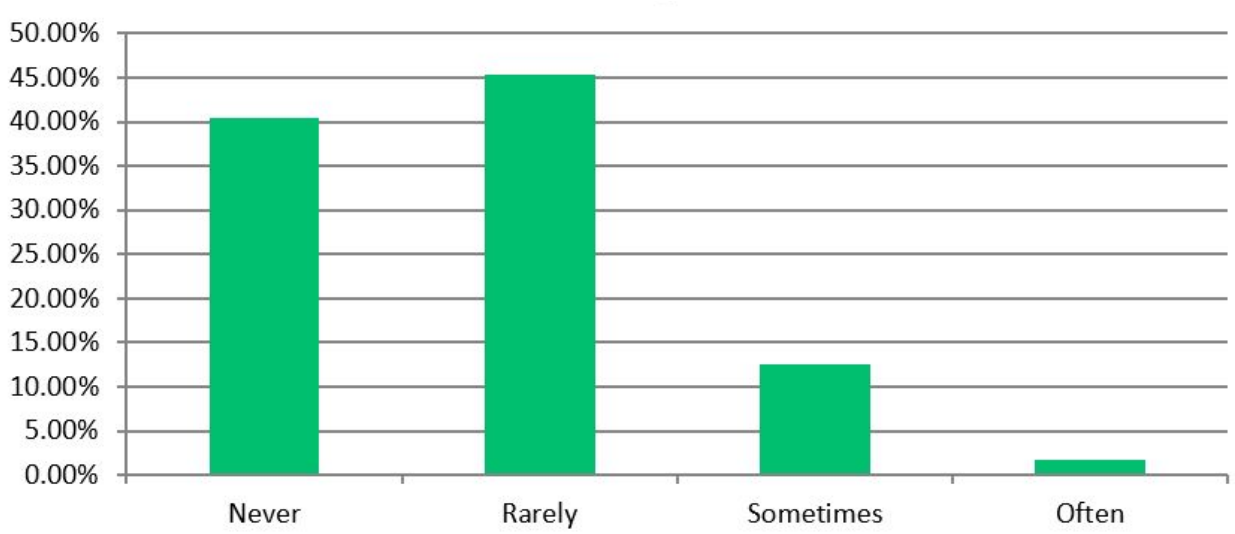


Question 11: How would you rate the completeness of validation layer coverage?



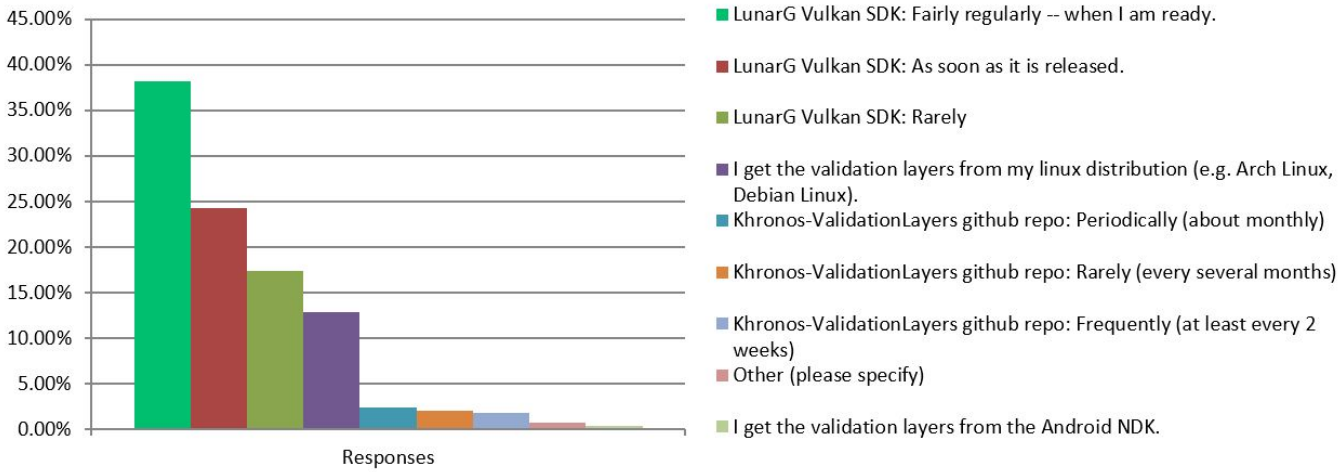
At LunarG, we know there are many gaps still remaining in Validation Layer coverage. We focus on filling those gaps found by developers first (because those gaps are use cases actually being commonly used). Since the previous survey in 2018, the validation layer coverage satisfaction level of “high” has increased from 43% to almost 50%.

Question 12: How often do you see false error reports (error reported when it shouldn't have been)?



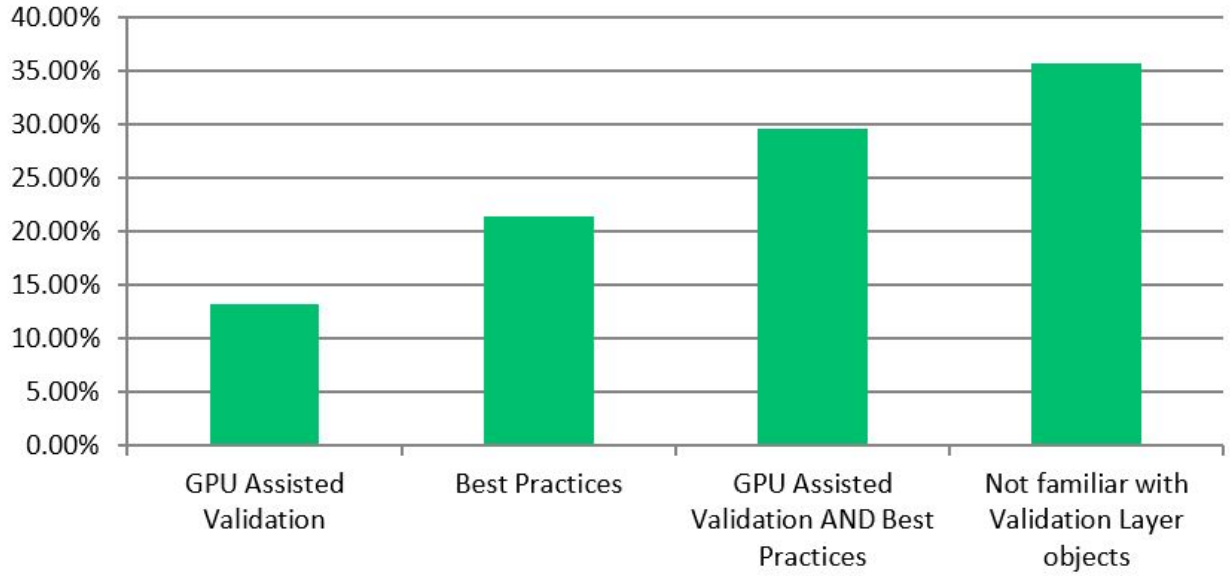
We do know that the validation layers are not perfect and have some defects. However these results show they are effective at accurate reporting. There are no significant changes in these percentages from the survey taken in 2018.

Question 13: How and when do you update your validation layers?



This chart makes it clear that people use the SDK to get their validation layers updated. Most people are updating fairly regularly. But there is a population of about 18% who rarely update their SDK version. There is a small population of folks who prefer to build themselves from the repositories, but the SDK still remains the more popular update mechanism.

Question 14: Which of these Validation Layer objects do you use?

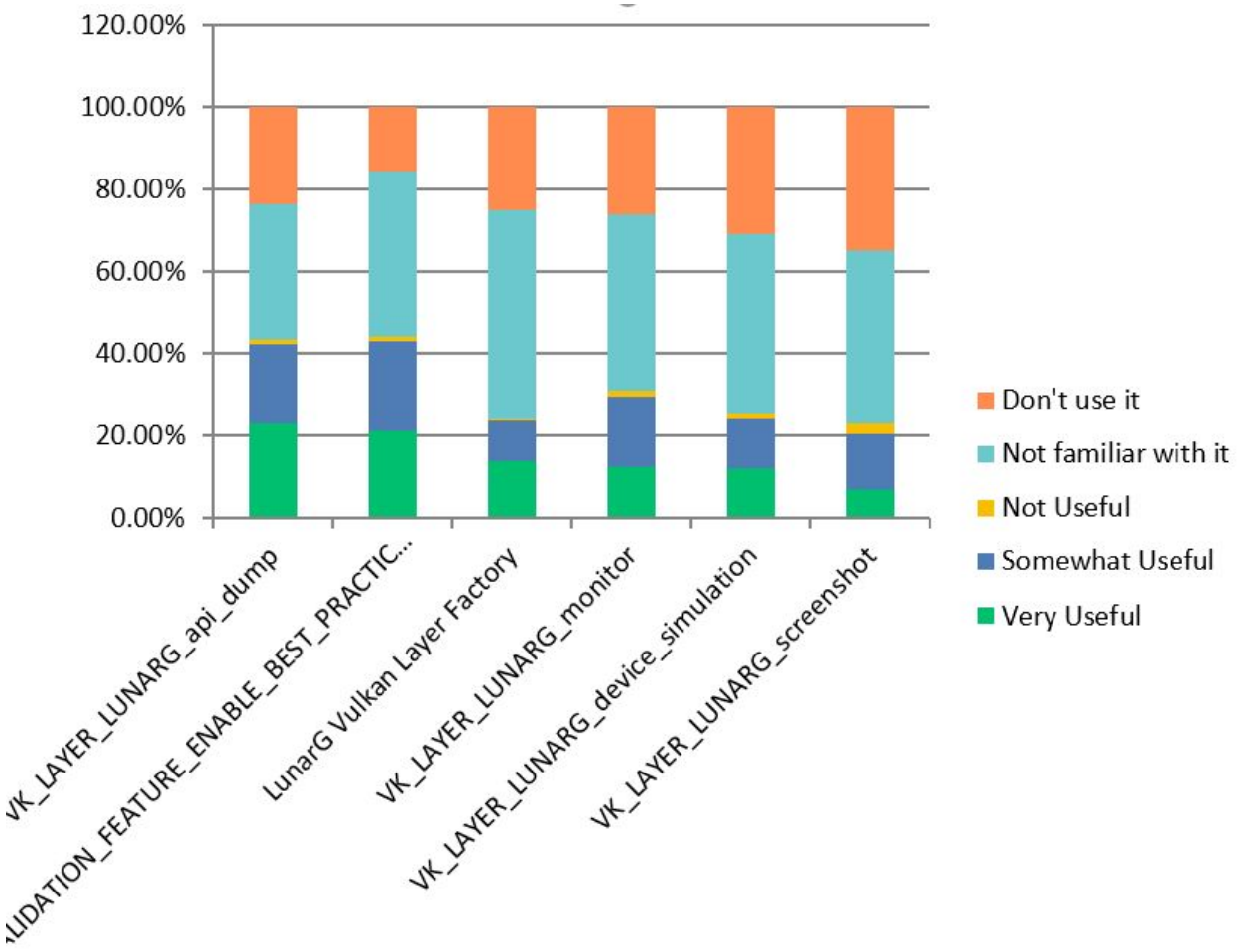


Both the GPU Assisted Validation and the Best Practices validation are useful tools. However this chart shows there are many people who are unaware of these validation layer objects. We need to improve visibility of these objects.

Future releases of the Vulkan Configurator will make the additional objects visible and easy to enable/disable.

Question 15: For the Vulkan Layers listed below, indicate their usefulness:

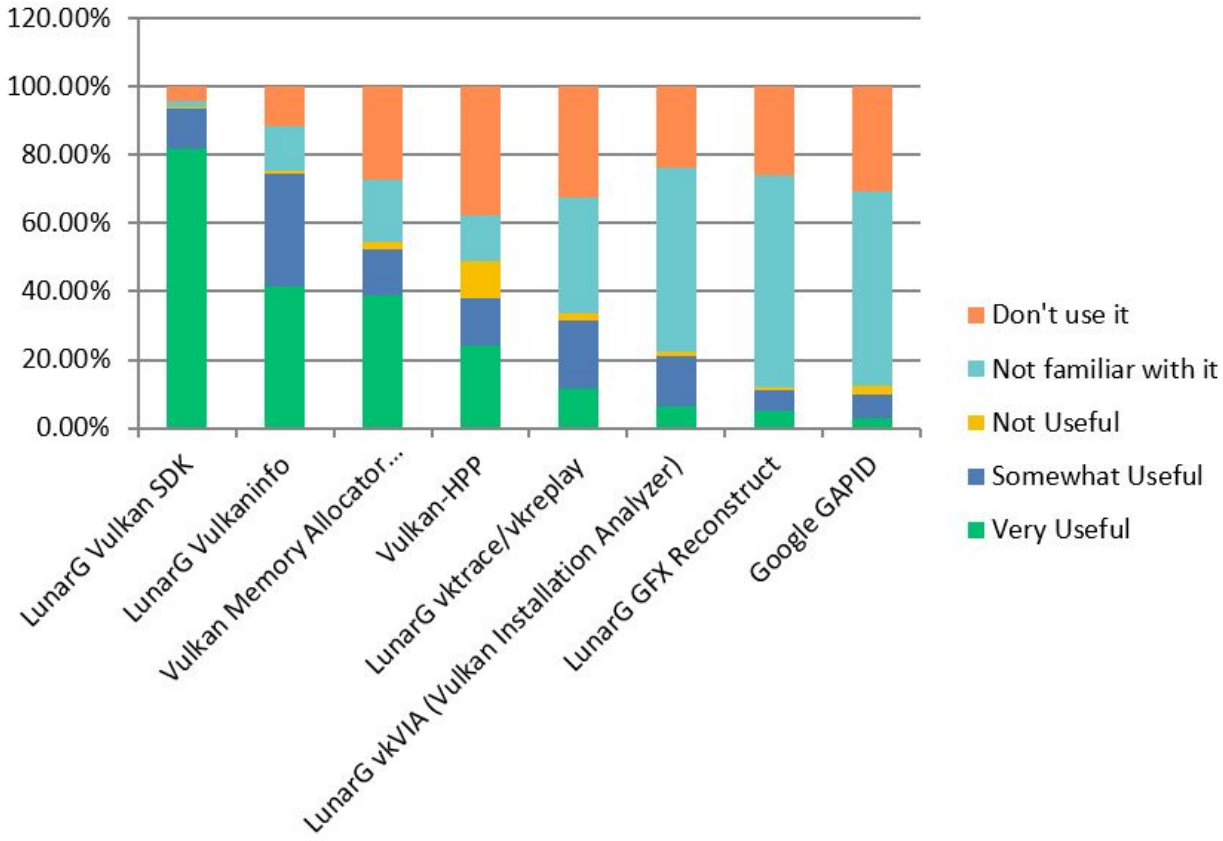
The



The purpose of this question was to get a view of the usefulness of some of the additional layers beyond the validation layers. Key conclusions:

1. Many of the folks are not familiar with the additional layers/objects.
2. Screenshot is lowest on the list. The motivation for this layer was to aid in graphics regression testing being done by LunarG and its use case for the broader audience is limited.
3. The device simulation layer hasn't been updated with more capabilities for over a year. An update would probably help increase its usefulness.
4. The Best Practices was just released as an object of the validation layers in the last 6 months. It is useful. More people need to be aware of this object. (The GPU Assisted Validation object should also have been a choice in this list).

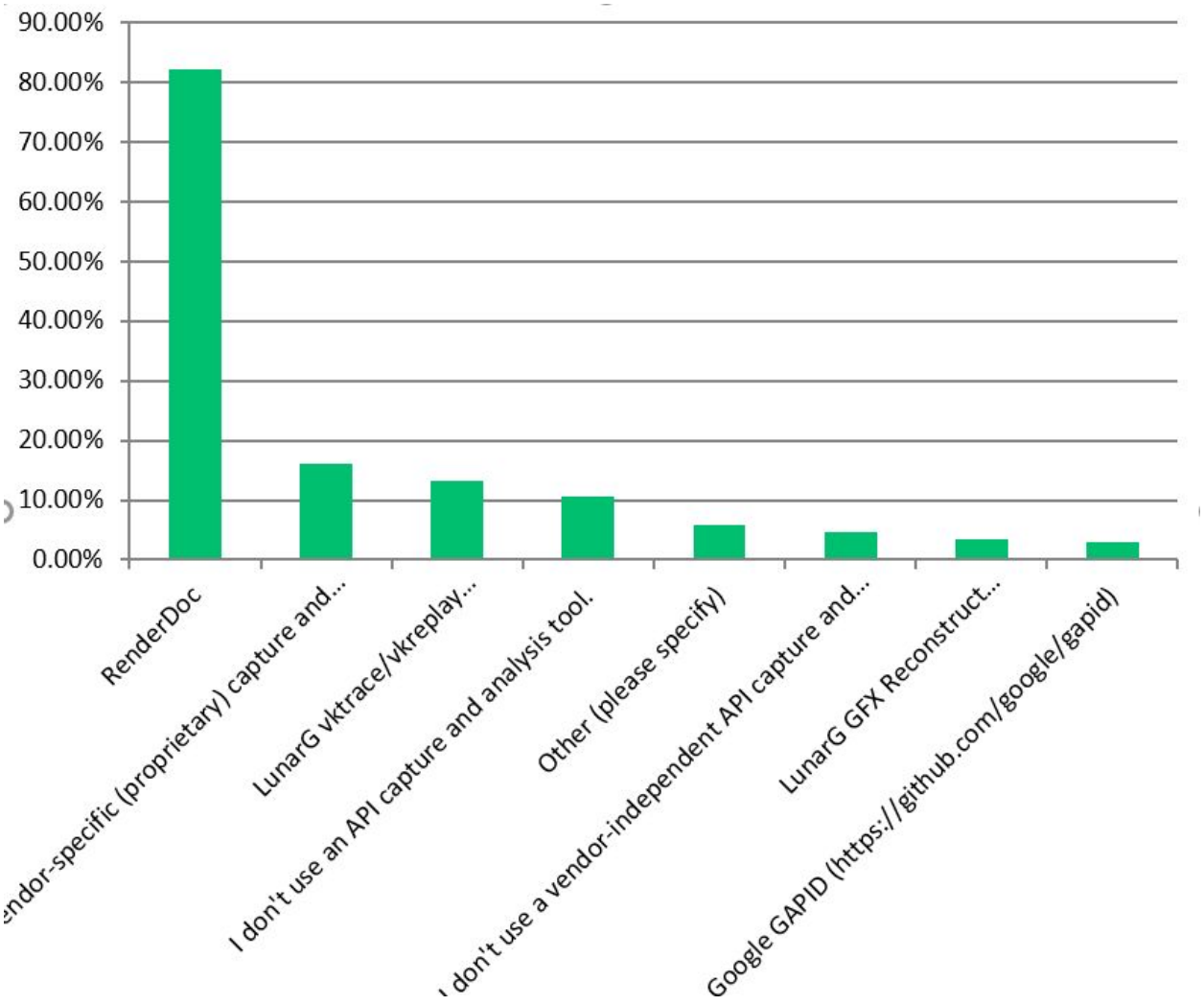
Question 16: For the Vulkan Tools listed below, indicate their usefulness:



Conclusions:

1. The Vulkan Memory Allocator is quite popular and is now being considered as an addition to the SDK
2. vkVIA being lower on the chart is not that surprising. It's output is awkward. But it is a useful tool to analyze what is wrong with your Vulkan installation. Once your Vulkan installation is good, there isn't much need to use it anymore.
3. GFX Reconstruct will replace vktrace/vkreplay (very soon). It is not yet part of the SDK and is in Beta mode. However it is already showing increased quality, reliability, and functionality over vktrace/vkreplay.

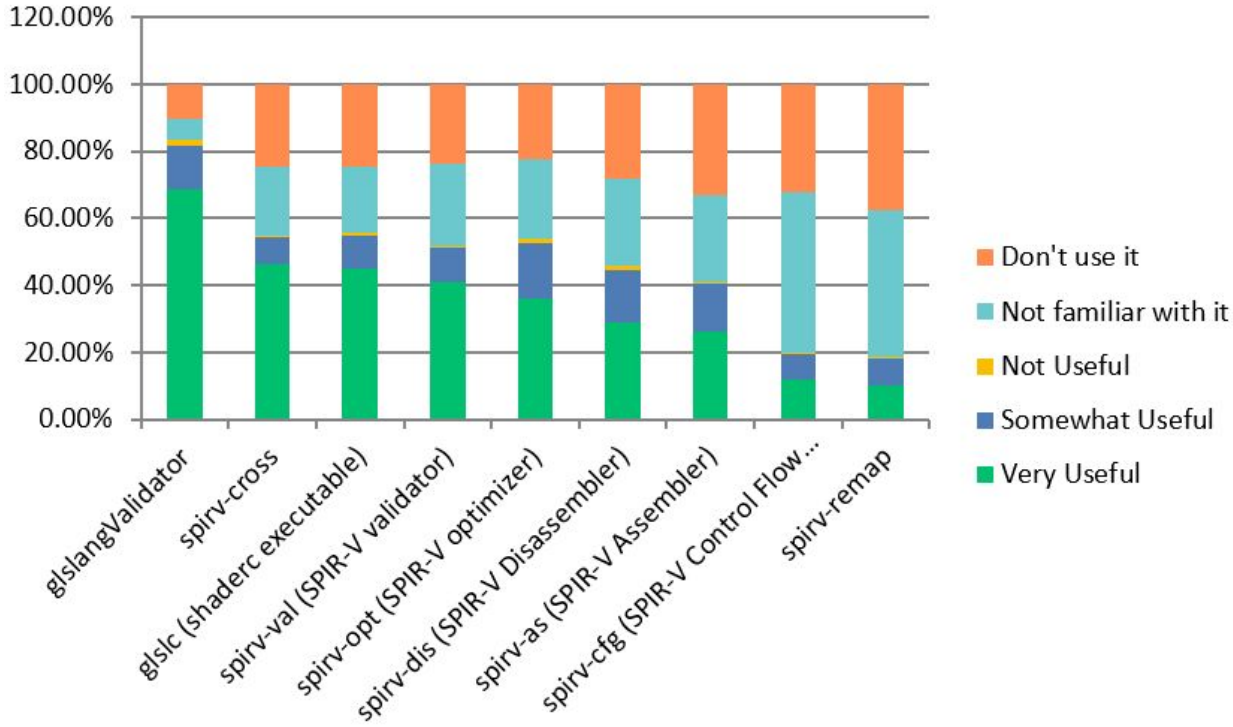
Question 17: Which Vendor Independent tool do you use for API capture and analysis? Check all that apply:



The items listed in the other category were:

1. Custom in-engine built tools
2. Metal frame capture
3. No good, stable options for macOS that I know of.
4. CodeXL
5. Intel GPA
6. internal

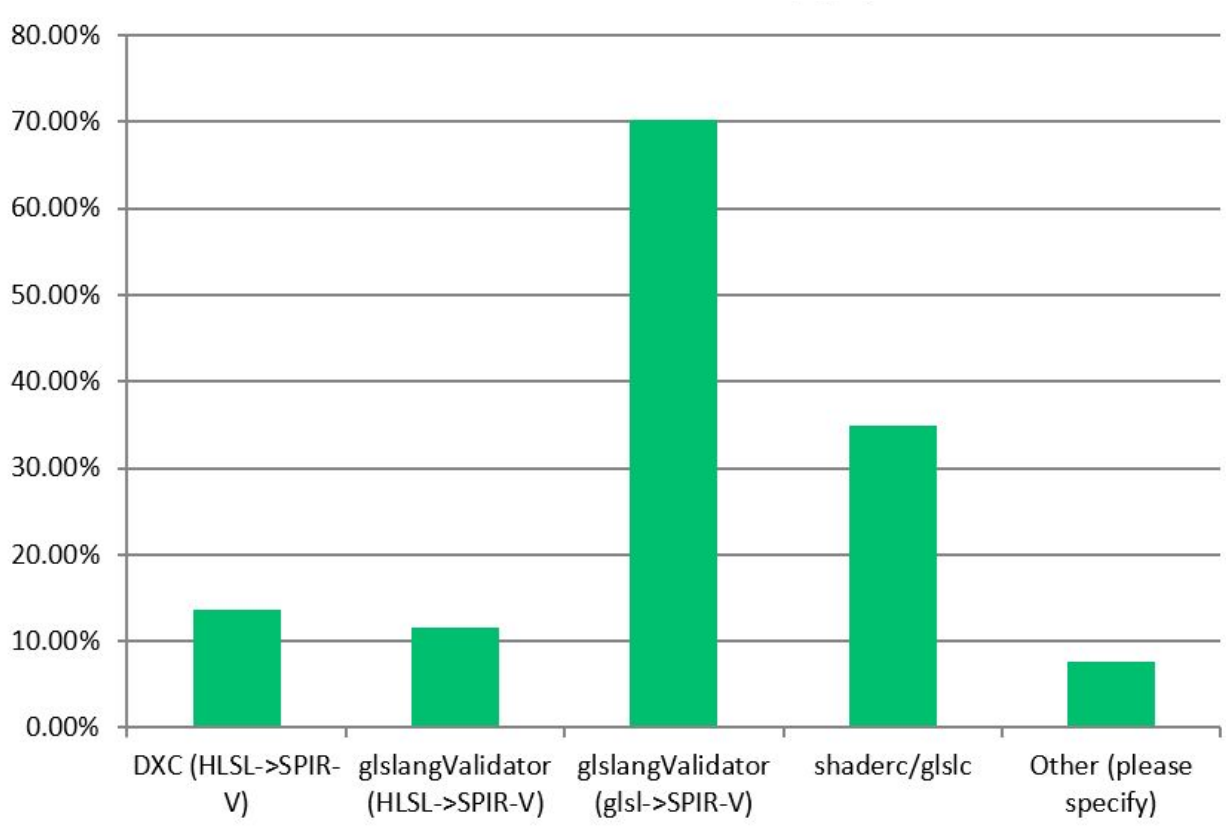
Question 18: For the Vulkan shader-tool-chain tools listed below, indicate their usefulness:



This question was asked because we have incomplete and inconsistent inclusion of the shader tool chain tools in the SDK across the platforms. The items in this list were inconsistently being included across the SDKs. I was probing to confirm popularity and need.
Conclusions:

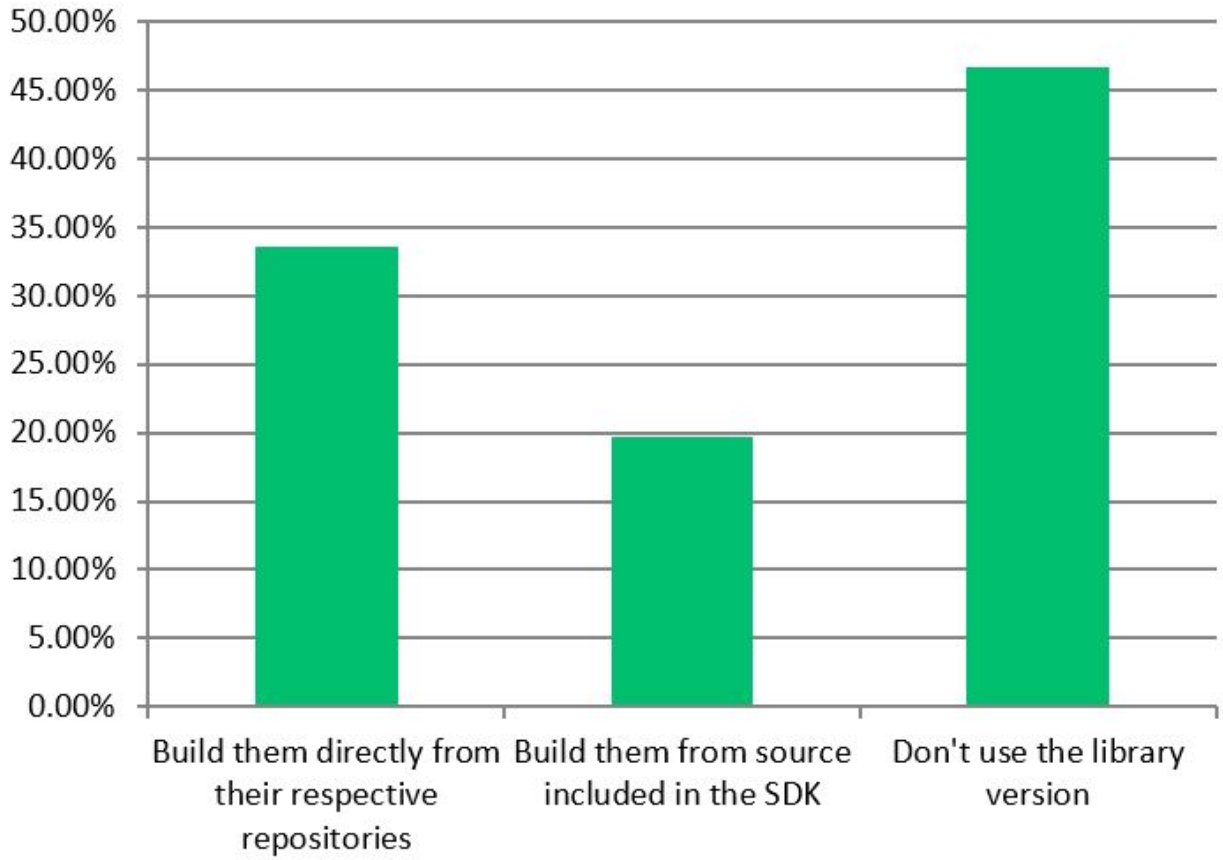
1. This chart, and additional open ended feedback indicates that all of these tools should be included in all SDKs.
2. DXC should have been included in the list.

Question 19: What is your front-end for creating SPIR-V? Check all that apply:



Note that we don't have two options for shaderc (HLSL->SPIR-V and glsl->SPIR-V) and should have. So we don't know how many people in the shaderc bucket are coming from HLSL or glsl. Results of this question and feedback from open-ended questions means that LunarG will investigate getting DXC added to a future SDK.

Question 20: If you use one of the shader-tool-chain tools in library format, what is your primary build source?



The original question was: For the Windows SDK, the SPIR-V and glslang executables (commands) are included, but not the software libraries (e.g. SPIRV-Tools-shared.dll, OGLCompiler.lib). If you use the software library, please indicate your primary build source.

Today, the SDK only includes the executables for the shader-tool-chain tools and not the API libraries. The primary reason for this is Visual Studio binary compatibility issues across Visual Studio versions and the potential for needing to build multiple versions of each library and the complexity that brings to the SDK.

The outcome of this question indicates that we need to find a way to make these libraries easily accessible to SDK users, either by making the build of them in the SDK extremely easy, or to build and include them in the SDK. Responses on open ended questions also confirmed that SDK users were looking for this capability.

Question 21: What features, tools, and/or improvements would you like to see added to the LunarG Vulkan SDK? (Open Ended)

There were many constructive and informative comments from the survey respondents. From the comments the following conclusions can be made:

1. The shader tool chain libraries in addition to the offline executables need to be added to the SDK.
    a. This is work in progress and will be in a future SDK.
    b. For the Android ecosystem, shader tooling is available from the Android NDK. Currently the NDK contains source for online shader compilation, but not precompiled libraries. Google proposes to deliver precompiled libraries in some future version of the NDK, in .aar form, for common Android ABIs. For more on .aar files, see https://developer.android.com/studio/projects/android-library.
2. Add SPIR-V reflection to the SDK
    a. It is recommended that reflection occur on a SPIR-V module directly.
    b. Work is underway to migrate Hai Nguyen's SPIRV-Reflect project (at https://github.com/chaoticbob/SPIRV-Reflect) to the SPIRV-Tools GitHub project at https://github.com/KhronosGroup/SPIRV-Tools. This will enable issue resolution and feature enhancements..
    c. Additionally, use of SPIRV-Reflect should also offload uses of Glslang for SPIR-V reflection; glslang reflection is based on the high level language, not based on SPIR-V.
    d. Once SPIRV-Reflect is available it will be included in a future SDK.
3. There is an interest in the SDK also including the Android builds of the validation layers.
    a. This is under evaluation and will be in a future SDK.
4. There is interest in having DXC included in the SDK.
    a. This is being investigated.
5. Folks are interested in more complicated samples.
    a. The Khronos working group has now started a KhronosSamples repository and project. There will be continuous improvements and additions made to this repository over time.
6. There are gaps in tools for which folks would still like to see solutions:
    a. Visual profiling. There is still a need for a cross platform performance profiling tool. There are hardware vendor specific profiling tools available. This is an opportunity that hasn't been resolved yet.
    b. Synchronization validation and/or synchronization analysis tools

        i. It should be noted that LunarG is currently working on creating synchronization validation in the validation layers. It is a large and complicated project and should begin releasing portions beginning this spring.
    c. Shader debugging - We recognize there is a desire for but lack of vendor-independent debugging tools for Vulkan shaders.
7. Specification
    a. There was a request to include the specification with the SDK. The SDK already builds a specification for each SDK release.
    b. All the SDK documentation (including the specification builds) is online at vulkan.lunarg.com. Go to the “docs” page and select your SDK version. All available documentation for that selected SDK can be found.

Question 22: How could the validation layers be improved? (Open Ended)

Many useful and constructive comments about the validation layers. I would classify the comments into the following categories:

1. Validation error reporting improvements
2. Better validation layer coverage
3. Synchronization Validation
4. Improve the performance of validation layers
5. Incompatibility of Validation Layers and MoltenVK extensions

Validation Layer error reporting improvements:

Most of the comments were in regards to error messages and how they could be improved for better understanding. Awareness of this is key as new errors are written to create the best possible error message string.

We are also aware of the general usability of validation layer error messages:

1. VUIDs are not visible in any of the versions of the specification
2. When the validation layers emit an error, a link into the single-file html version is also provided along with the VUID. The link is referencing a non-visible VUID tag in the html spec.
    a. The time to load the single-file HTML version is very long, limiting the usefulness of this link in the error message.
    b. Linking to the chunked version of the specification is impractical because VUIDs are not specified by chapter.

    c. It is possible to generate per-VUID links to the PDF version of the spec if the VUID's were visible or marked as PDF destinations
3. After opening the link and waiting for the specification to load, the specific VUID text being referenced is not obvious or apparent, as the VUID text is not visible.

We would prefer to have the VUIDs visible in all versions of the specification. There are three versions of the specification available currently, none of which can be searched for a VUID:

1. HTML single-file specification
2. HTML chunked specification
3. PDF full specification

The source code of the single-file HTML specification can be searched, but this is non-intuitive, and as the HTML spec is very large load times make this painful. If the existing VUID tags were visible in all versions of the specification, each of the versions would be searchable including the PDF version, which loads very quickly. Spec links currently provided in all validation error messages could be changed to point to the PDF (or even chunked) version of the spec, greatly improving the user experience.

In short, Enhancing the spec by making the VUID tags visible would not increase size or complexity, and would be very useful to end-users looking for context related to problems reported by the validation layers.

We are working with the Vulkan Working Group to see if such an enhancement could be made to the specifications.

Better validation layer coverage

We focus first on resolving issues found by developers (tagged as "incomplete" issues in the github project). Then we focus on other major initiatives such as synchronization validation, GPU-Assisted validation, and moderating pull requests from the community. If you find an area being missed by validation, please be sure to submit a github issue (and it will get tagged as "incomplete"). This will help elevate it's priority and visibility. There is a lot of validation layer work to be done and we love contributions! See CONTRIBUTING.md for more information on how to contribute.

For the last year we have also followed the Vulkan working group policy that new extensions cannot be released without corresponding validation layer implementations. This is helping to increase day-one coverage and to prevent the gaps in validation from growing over time.

Provide Synchronization Validation

A comprehensive synchronization validation solution is in progress. It will be rolled out in phases. Dedicated resources have been working on the implementation for 8 months. It is an immensely difficult problem to implement and we anticipate some of the first code drops to be available in the Q2 timeframe.

Improve the performance of validation layers

There was some performance profiling and improvements made in the last year resulting in a 2X improvement in performance. As well, an improved subresource tracking container has been implemented that is common to the image layout tracking and synchronization validation. It is less intensive regarding new/free operations, supports range based for operations, and suitable to be used to replace the slow (profiler hot-spot) global image layout maps.

In the future, we will be delivering more controls for enabling/disabling portions of validation layer checks to help with performance. These will be made visible in the Vulkan Configurator (vkconfig) for easy access.

We acknowledge that performance needs to be continuously measured and improved and that we are not done. If you have a specific workload that is showing unacceptable performance, sending us a profile in a github issue is very helpful.

Incompatibility of Validation Layers and MoltenVK extensions

MoltenVK provides extensions to access Metal features (such as allowing artifact interop between Vulkan and Metal, or support for passing MSL directly to vkCreateShaderModule within layers) that are not included as extensions in the Vulkan specification resulting in the inability for the Validation Layers to execute properly when these MoltenVK extensions are used by applications. The Portability group, MoltenVK maintainer, and Vulkan Working Group are working to determine a solution to resolve this incompatibility. This MoltenVK issue can be used to track it's progress.

Question 23: What inhibits you from effectively and efficiently developing Vulkan applications? (Open Ended)

The open ended responses can be categorized into the following categories:

1. API Complexity (and I don't have the time)
    a. Yep. It's complicated. Vulkan puts more work and responsibility into the application. Not every developer will want to make that extra investment, but those that do so correctly can find power and performance improvements.
    b. The complexity of the API is what is driving the request for more documentation, samples, and tutorials.
2. Driver quality and inconsistencies between GPU vendors
    a. LunarG does testing of the validation layers and various ecosystem tools across multiple GPUs and will find defects. We are sure to report these issues to the hardware vendors.
    b. As you find issues, submitting issues to the hardware vendor is the best method to help improve the quality of the drivers.
3. More documentation, samples, and tutorials are needed
    a. In the last year, the Vulkan Guide was developed by contributions from Vulkan Working Group members. This guide is proving helpful and will continue to be maintained and evolved by the Vulkan Working Group.
    b. The Khronos working group has now started a KhronosSamples repository and project. There will be continuous improvements and additions made to this repository over time.
4. Deficiencies in tools
    a. There are multiple requests for more debugging tools, profilers, and shader debuggers. While hardware vendors are providing tools unique for their hardware platform, there are still requests for tools that work across hardware platforms.
    b. These deficiencies are well known and over time the developers in the ecosystem will find more cycles to bring more tools to the community.
    c. Due to the complexity and verboseness of the API, there are requests for wrappers that allow developers to more efficiently get started with Vulkan.
5. Shader Tool Chain
    a. SPIR-V to DXIL translation
        i. This sounds interesting and could have a wide audience. However, we know of no projects to do this.
        ii. There is an indirect path already: SPIRV-Cross to translate SPIR-V to HLSL, then DXC to translate HLSL to DXIL. However, we think that's not what was meant by the feedback.

b. SPIR-V optimization: fine-grain control
    i. We believe this is already directly supported. The spirv-opt options are:
        1. The input file
        2. The output file
        3. A list of transformations, in order.
    ii. For fine grain control of optimizations, list exactly the transforms you want to do, in order.
    iii. Additionally, the spirv-opt tool has pre-baked “recipes” for size or “speed” optimization, as -Os and -O options. You can also create your own recipes. See the help for spirv-opt.
    iv. The C++ API to the optimizer has the same ability to specify individual transformations (or “passes”) in sequence. Instantiate a “Pass manager”, then schedule individual passes, then run the scheduled passes over a given module. See the API description at https://github.com/KhronosGroup/SPIRV-Tools/blob/master/include/spirv-tools/optimizer.hpp#L37

Closing

Vulkan is now four years old. Considering the entire lifespan of a graphics API standard, Vulkan is still very young. There are basic core deliverables that are an absolute must to be delivered (SDK, Vulkan Loader, Vulkan Validation Layers, …). These core deliverables still require active maintenance and enhancements and much of the LunarG resource is focused on ensuring these core elements are meeting ecosystem requirements. Over time, more cycles will become available to work on additional ecosystem tools such as profilers, debuggers, and abstraction wrappers.

The very constructive and informed feedback to this survey has resulted in a heightened focus from the Vulkan working group and ecosystem contributors to fix issues and provide solutions.

While using the Validation Layers and Vulkan Loader, if you find bugs, issues, and gaps in the layers, please submit your issues to the Vulkan Validation Layer repository and Vulkan Loader repository. This brings visibility to the active developers for these projects to enable continuous improvements.

If you find errors and improvements to the Vulkan SDK, you can submit issues at the Vulkan SDK download site.

If you have additional feedback about the survey itself, please contact us at info@lunarg.com.

Acknowledgements

Thanks to those of you who took the time to complete our survey, sharing your Vulkan experiences and providing valuable feedback!

Thanks to the Vulkan Working Group for in depth review and feedback of the ecosystem survey results.

Special thanks to Google and David Neto’s team for reviewing the shader tool chain comments and providing answers where necessary.