LunarG 2025 Ecosystem Survey Report

March 2025

Executive Summary

Methodology:

- The purpose of the ecosystem survey is to help guide provide priorities to LunarG while developing the Vulkan SDK and many of the Vulkan Ecosystem developer tools.
- It was attempted to reach as many Vulkan developers as possible -- both SDK users and non-SDK users. The survey was advertised on X, Reddit, LinkedIn, the Khronos Vulkan slack channel, Vulkan Discord, and sent directly to 13,000+ recipients of the LunarG Vulkan SDK mailing list. It was amplified by Khronos on their X account and newsletter mailings as well.
- All comments from open-ended questions are included in this report, regardless if they are repeated. This helps you see the frequency of certain types of feedback.

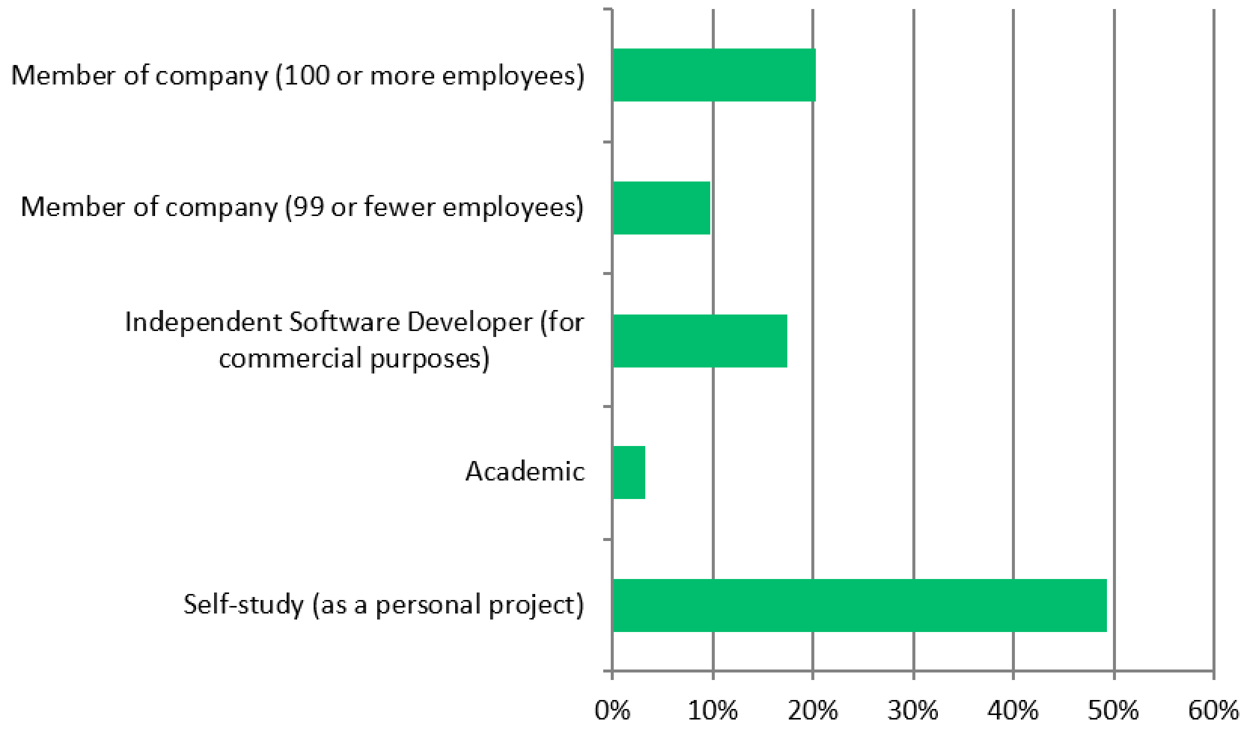
Some Highlights:

1. There were 279 respondents.
2. 47% of the respondents use Vulkan for commercial purposes. 52% of the respondents were self-studying Vulkan as part of a personal project or an Academic environment (non-commercial).
3. 72% of the respondents are regular, advanced, or expert Vulkan developers. 28% are basic or beginner developers. Hence the feedback is coming from a more experienced population.
4. This was the first year that we asked respondents which region (Americas, Europe, Asia) they resided. 60% of the respondents resided in the European region.
5. WebGPU is becoming an important API for the future
6. ARM platforms (both Linux and Windows) will be important development environments in the future.
7. slang (as a language and as a compiler) had a significant jump in popularity
    a. 2024: 3% of survey respondents. 2025: ~27% of respondents
8. Validation Layer Themes:
    a. Improve error messages
    b. Improve coverage
    c. Improve performance
9. Still strong demand for improved Vulkan documentation and tutorials
10. Continued concern about complexity and verboseness of the Vulkan API

LunarG Actions for the year to come:

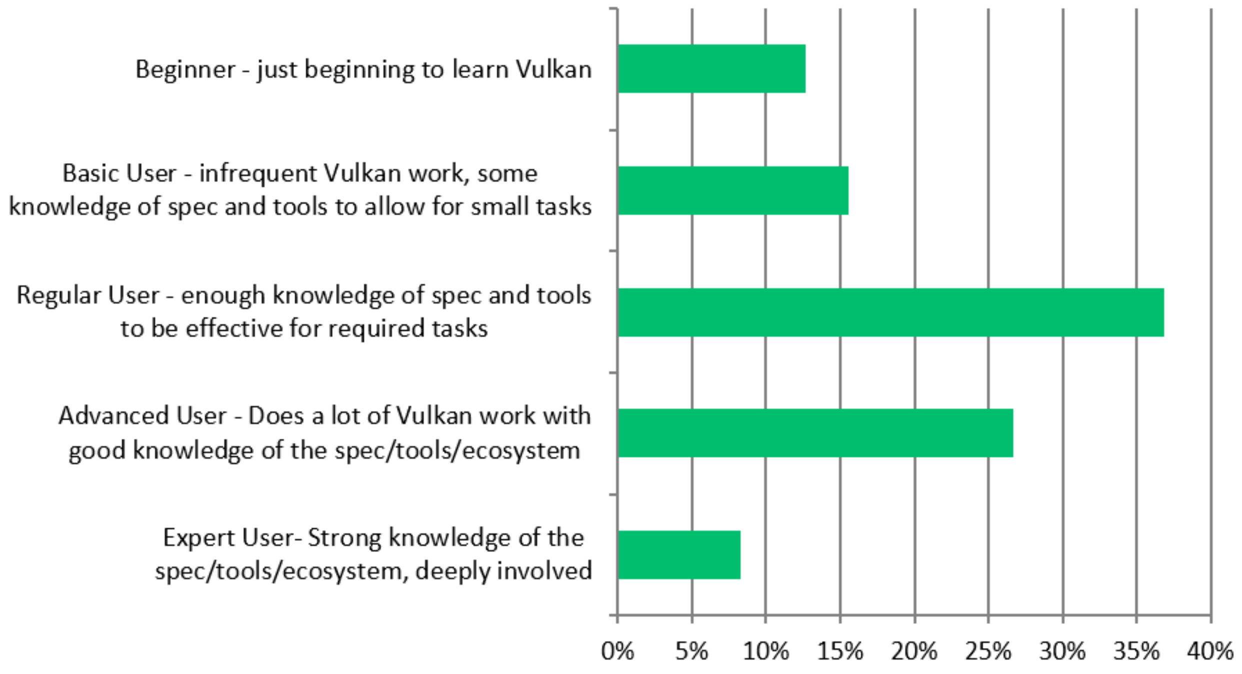
1. Deprecate the Ubuntu Packages. They will discontinue being updated by June of 2025. LunarG is looking for ways to reduce the workload for each SDK release. It has been our suspicion that the Ubuntu packages are nice to have but not critical. The handful of users that indicated the Ubuntu packages were critical, cited the need to rework their CI environment which relies upon package installation. It is very plausible to change these CI environments to install the Linux tarball instead.
    a. See if the freed resources enables a Linux on ARM pre-built tarball SDK
2. Continue validation layer focus on improving GPU-AV.
    a. Getting more people to enable GPU-AV for more complete validation coverage
    b. Performance
    c. More VUID checks requiring validation on the GPU
    d. Descriptor Indexing and Buffer Device Address are popular and LunarG will continue to focus on GPU-AV validating them well.
    e. Ray Tracing/Query is so popular. This is currently WIP to do a better job of validation.
    f. Mesh Shading probably should get more attention
3. Continue improvements to validation layer error messages
    a. Note that with the 1.4.309.0 SDK, some more significant improvements were made. (See the validation layer error messages document and SDK release notes for detail)
4. Enhance vkconfig and Vulkan Loader to enable adding ICD to the list of available ICDs on the system (e.g.: Lavapipe enablement for C.I. purpose)
5. Time permitting: Are there ways to get a Vulkan on Metal solution that can be Vulkan conformant and keep up with Vulkan evolution?
    a. It is known that the current Vulkan on Metal solution (MoltenVK) hasn't been able to keep up with Vulkan as it progresses. For example, its support has not moved beyond Vulkan 1.2 and it hasn't become a conformant implementation. With 30% of the population indicating it is a "must have", ~35% indicating it would be nice to have, the ecosystem has a gap in providing a good Vulkan on Metal solution.
    b. LunarG is investigating ways to help improve this situation.
6. Time permitting: Investigate VK_LAYER_live_introspection as a tool for integration with vkconfig/SDK

What type of Vulkan Developer are you?



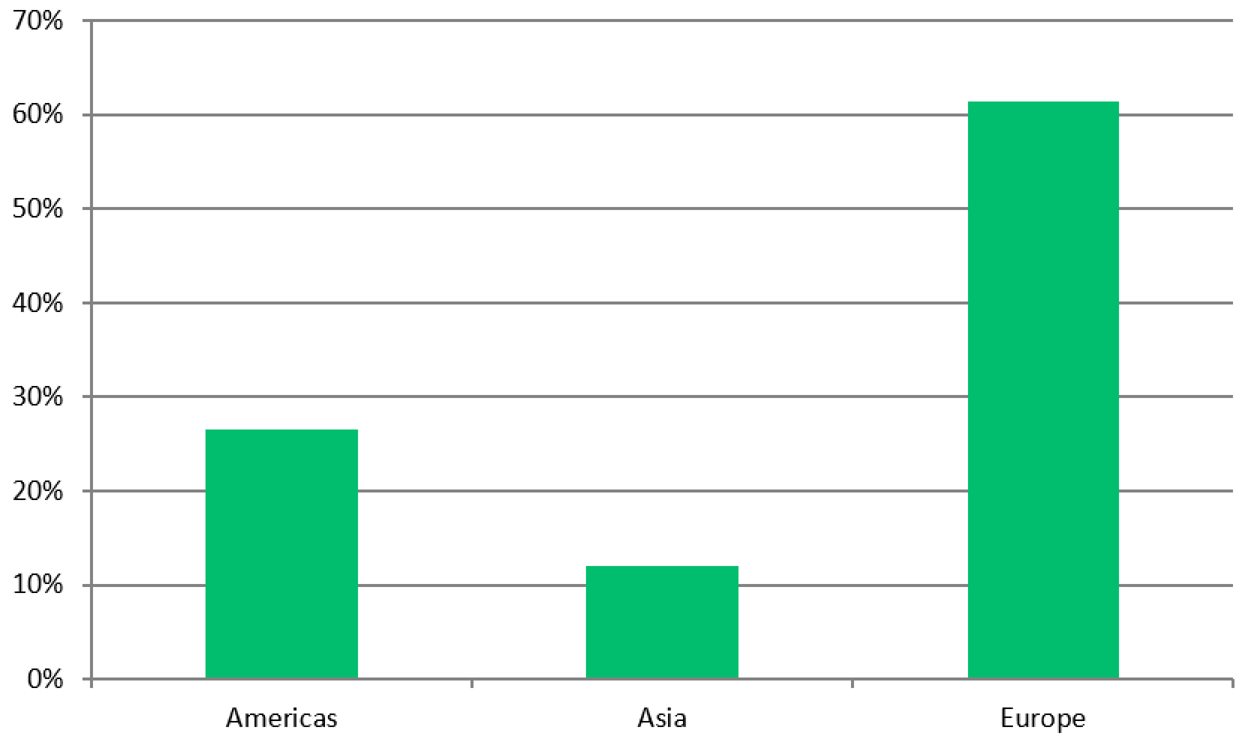
52.5% are self-study or academic
47.5% are using Vulkan for commercial purposes

How experienced of a Vulkan Developer are you?

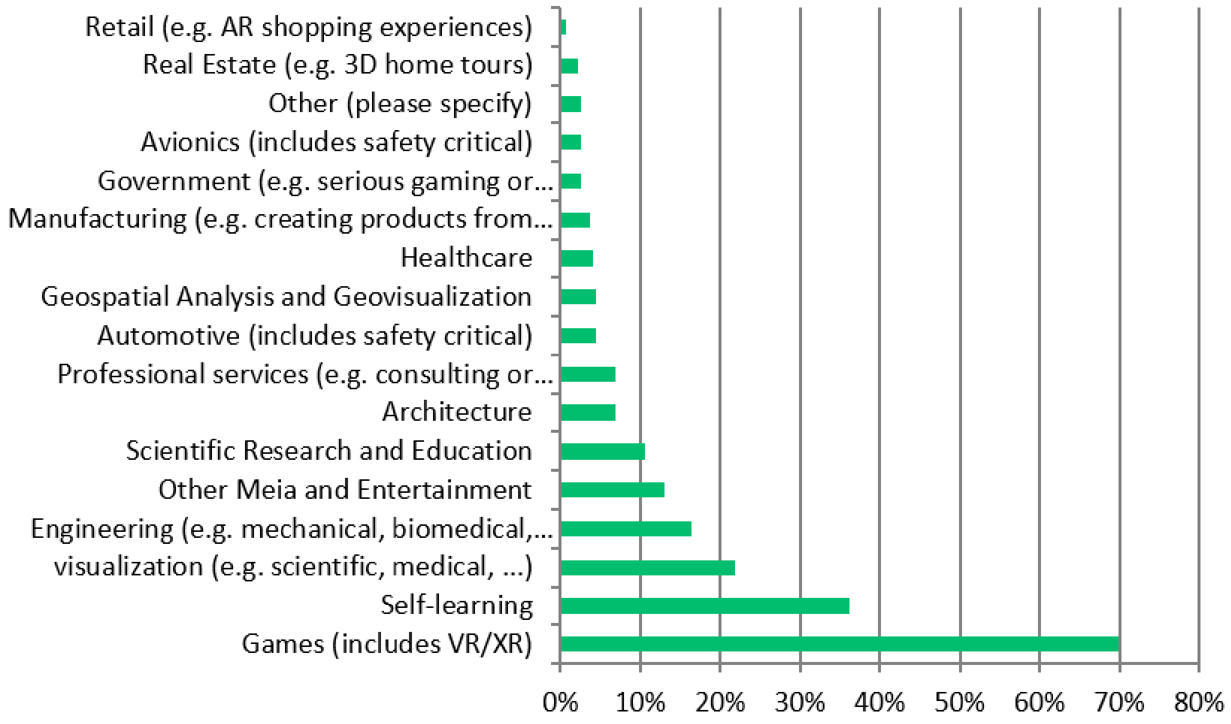


Of the population overall, 72% of the respondents had regular, advanced, or expert experience with the Vulkan API. The survey respondents who were doing Vulkan development for commercial purposes had more advanced or expert level experience with the Vulkan API.

In which Region do you reside?

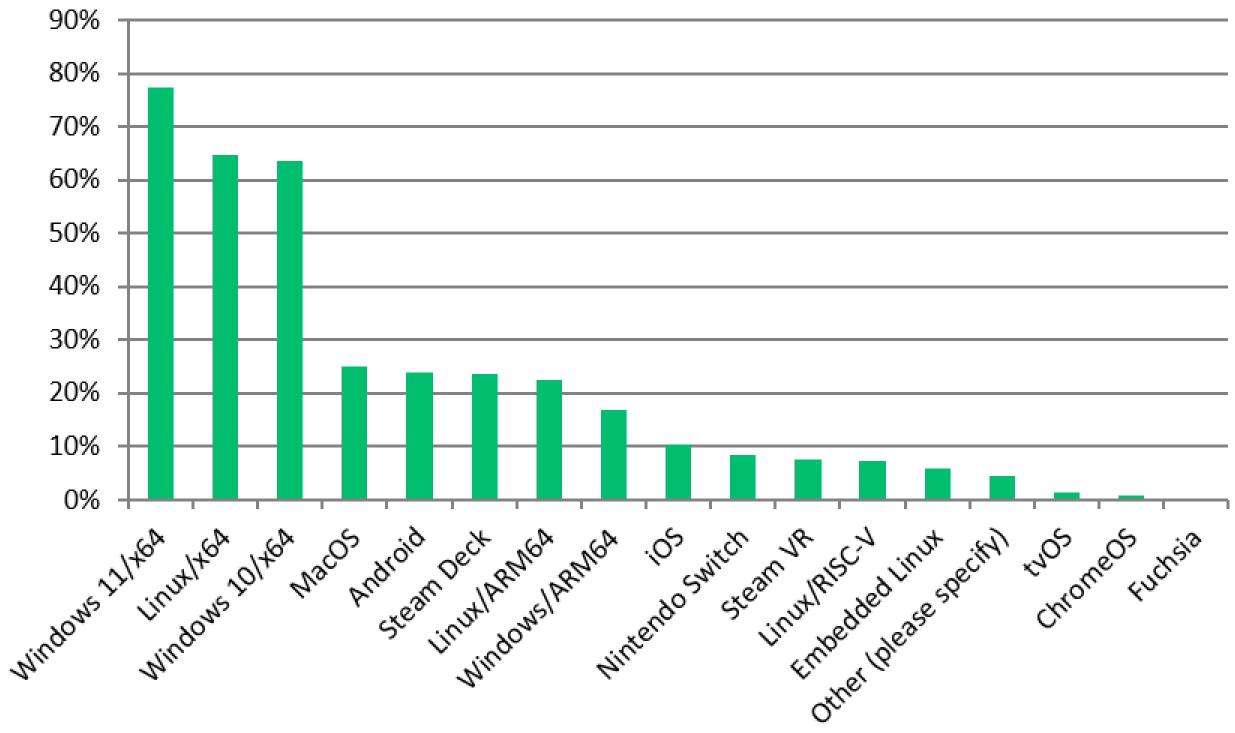


Your Vulkan development is for what type of industry (select all that apply)

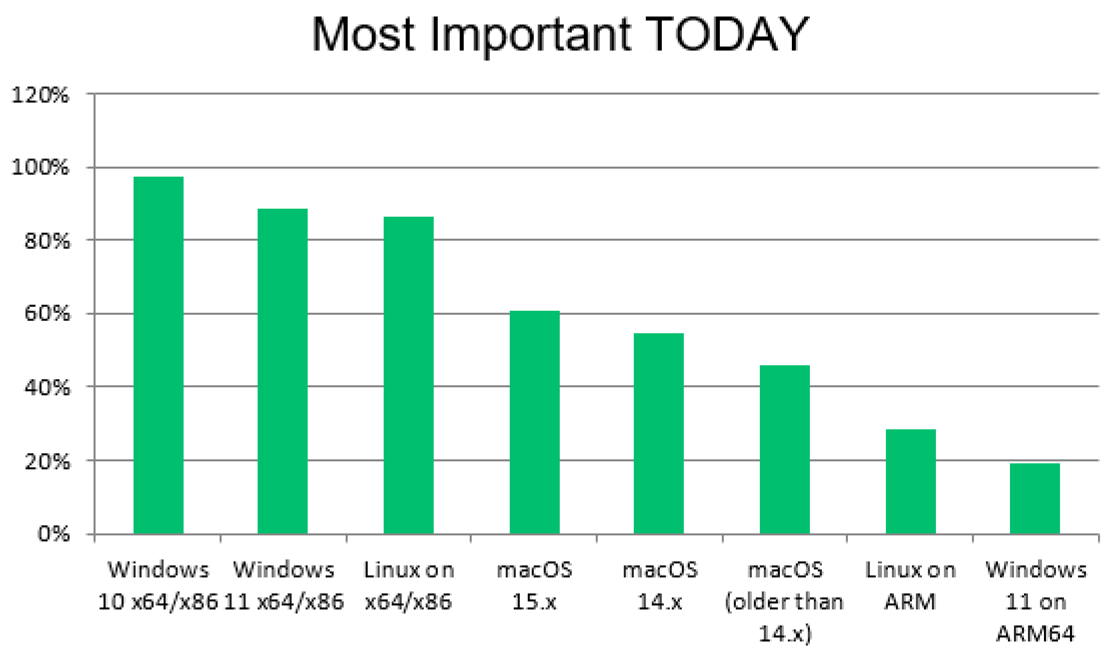


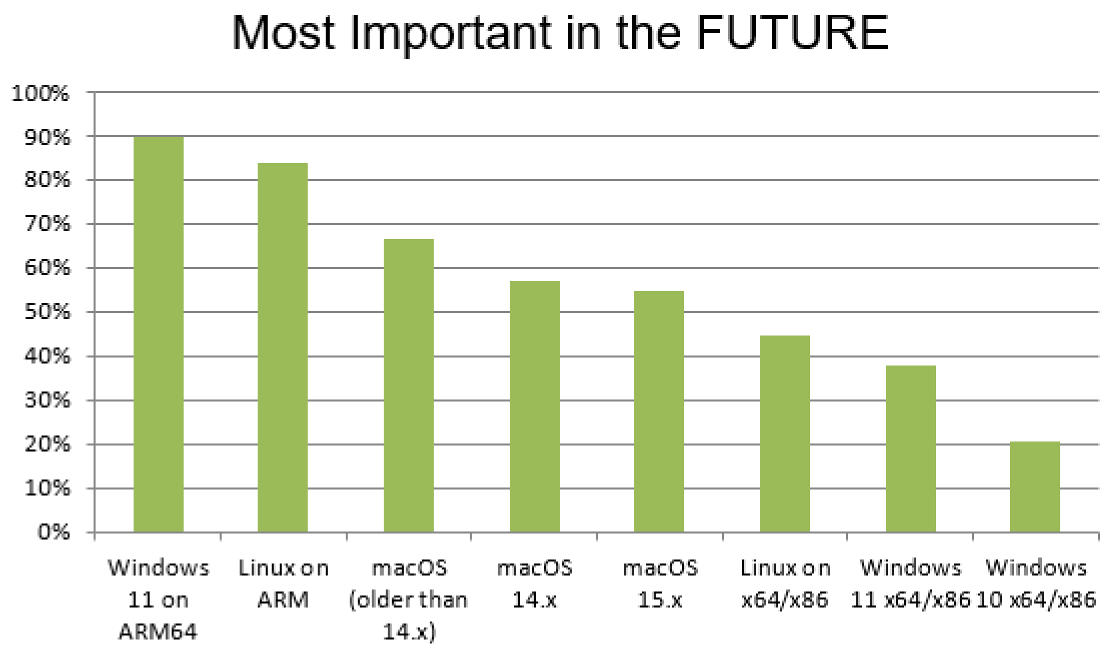
This question is asked to see if Vulkan is expanding into other industries. Compared with the data from 2 years ago, there are no significant shifts.

What are the targets of your Vulkan application? (check all that apply)



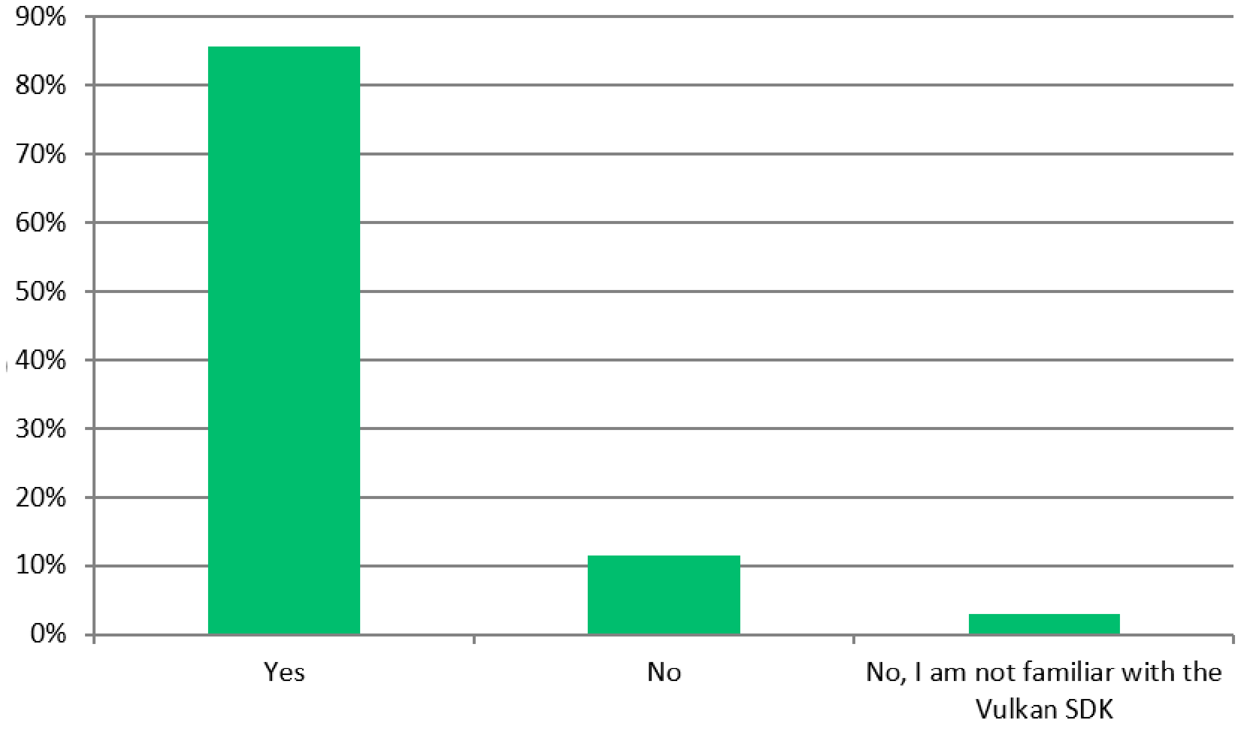
Indicate the importance of the listed development environments (skip environments that are not important)



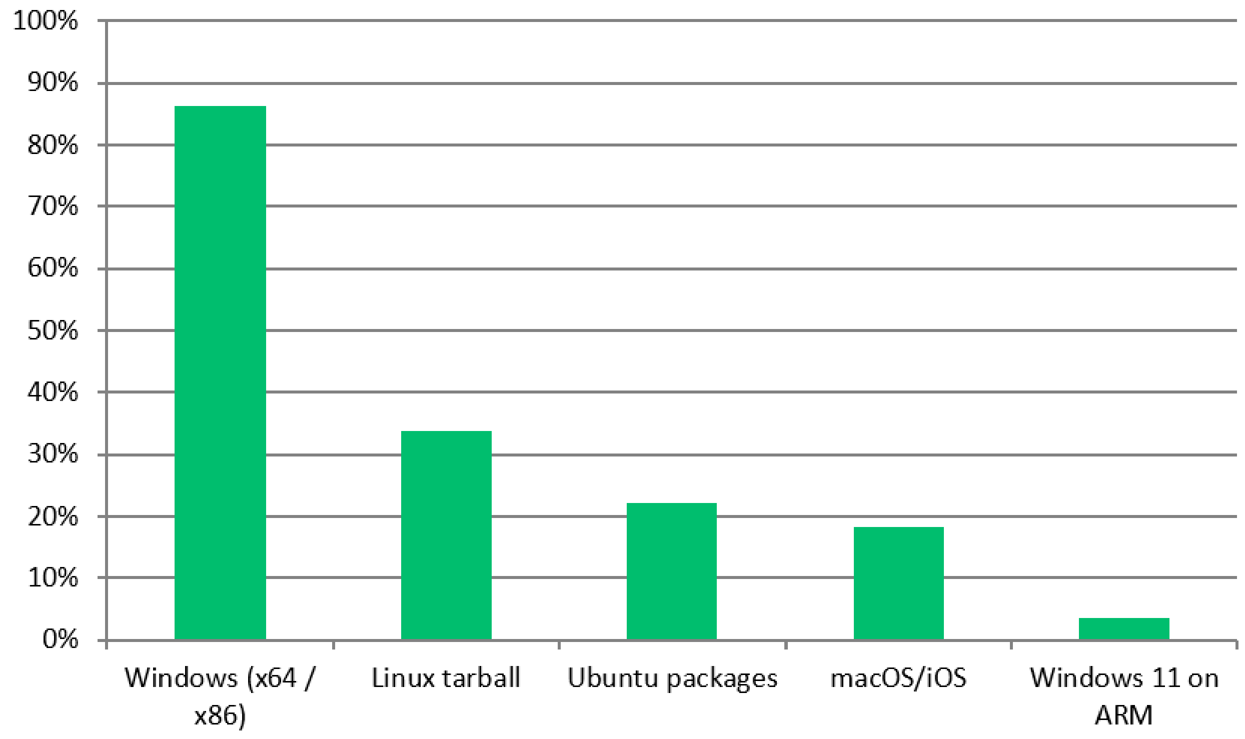


Windows on ARM for both Linux and Windows become more important than x64/x86. Interesting…

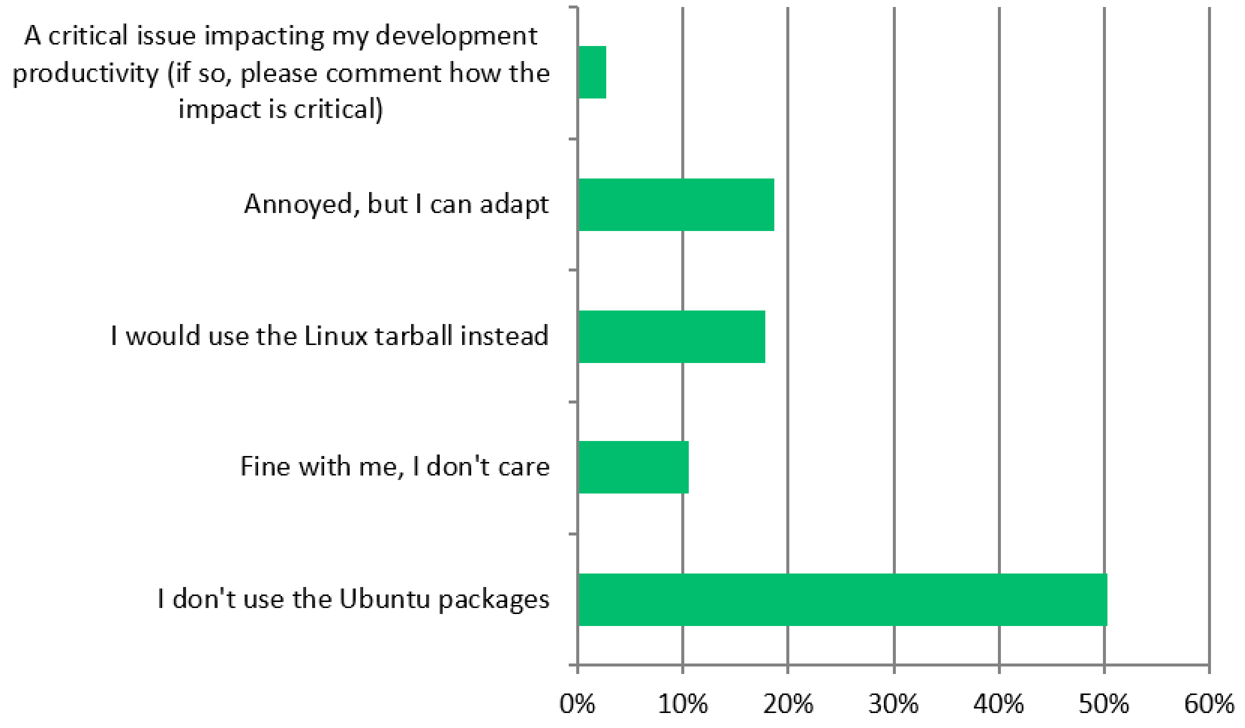
Do you use the Vulkan SDK?



Which of the following Vulkan SDKs do you use? (check all that apply)



If the LunarG Ubuntu packages for SDK releases were removed, what would be your response?



LunarG is looking for ways to reduce the workload for each SDK release. It has been our suspicion that the Ubuntu packages are nice to have but not critical. The handful of users that indicated the Ubuntu packages were critical, cited the need to rework their CI environment which relies upon package installation. It is very plausible to change these CI environments to install the Linux tarball instead.

As such, LunarG will be deprecating the Ubuntu packages in the future.

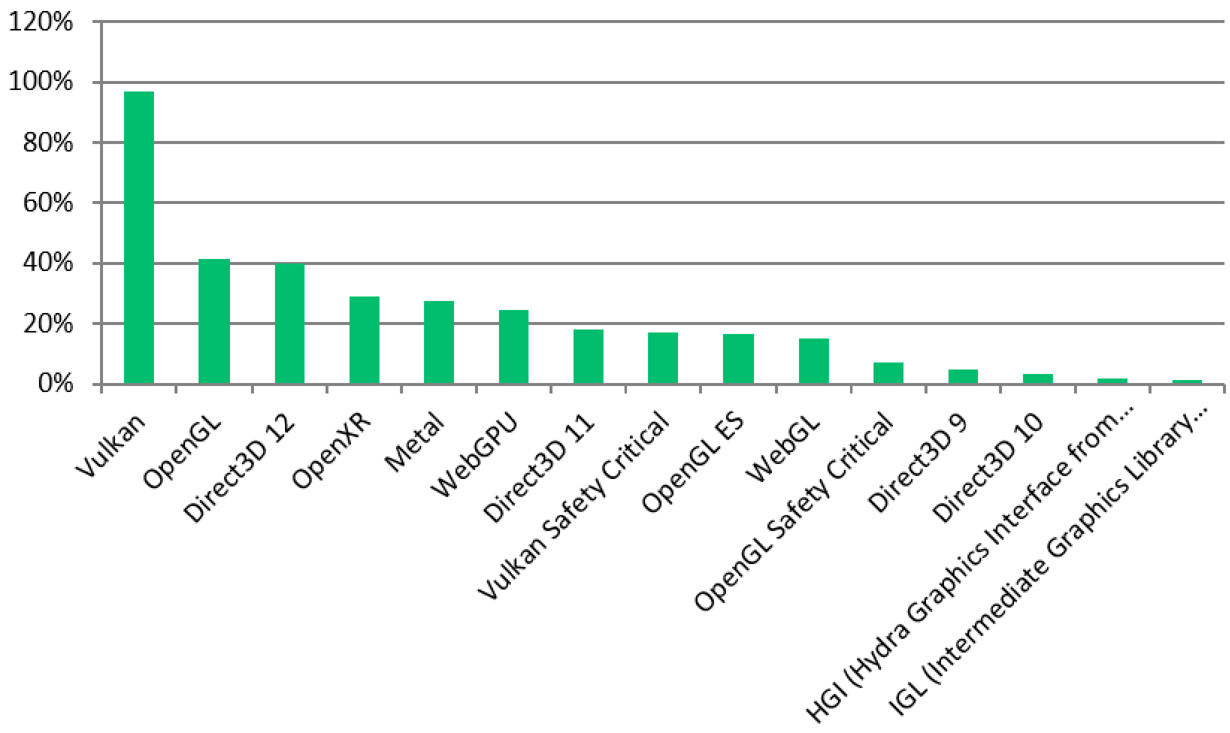
What suggestions do you have for the Vulkan SDK? (open-ended)

1. New SDK versions
    a. Arm version please.
    b. Nice to have Ubuntu ARM packages as well.
    c. Vulkan sdk for linux arm
2. Installation
    a. Provide an offline installer.
    b. I would appreciate an info that if installing on mac and not in global system wide install, it wont work really
        i. LunarG comment: The SDK works fine without global installation. The two draw backs are you have to run setup-env.sh from the terminal to set environment variables to point to the SDK files, and you don't get a system-wide ICD installed. Some developers will prefer one approach over the other. We could attempt to put this information into a message during installation to make it more obvious.
    c. Make it easy to pull the tarball from CI or scripts (no redirect on download page etc)
    d. Regarding question 8 and how to obtain the SDK, while I don't use ubuntu packages, I do rely on the SDK being available in a package manager.
3. IDE
    a. I would love to have first party plugins for GLSL for VS Code and maybe Rust Rover. The plugins existing today suck immensely. Would be great if those included #include syntax
    b. Not having a single suitable interactive development environment having a c++ toolchain along with vulkan SDK and tools built in.
    c. Please please do the GLSL plugins!!!
    d. We need better extensions in VSCode for GLSL development.
4. SDK content additions
    a. Include Renderdoc

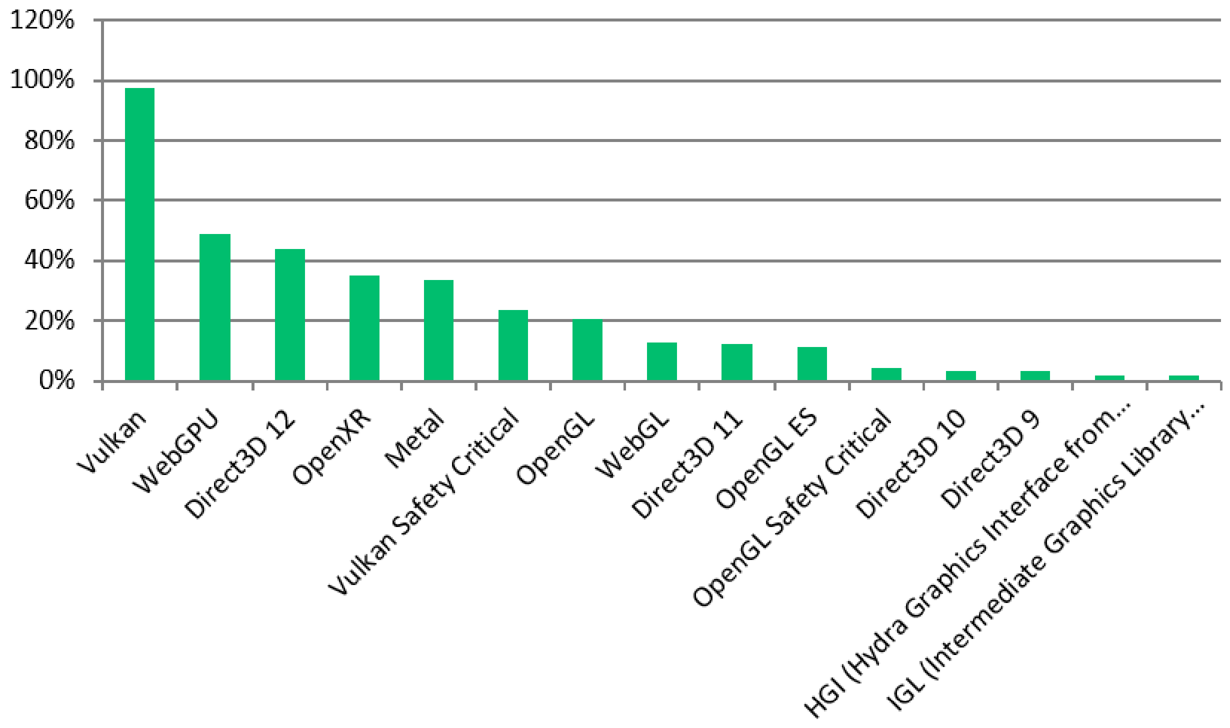
b. Make Rust a first class citizen in the Vulcan SDK. It’s kind of obvious that it is the future at this point. As far as I cab tell, Safety without no performance overhead has become critical.
c. Having the Slang component in the SDK simplifies dev environment setup a lot, but that's quite a lot of lag given how fast they're moving on bug fixes and updates.It'd be nice if there was a "rerun the installer" or "replace this subfolder with that tarball" process to refresh Slang in place.
d. Make installation of Vulkan samples an option.
e. Make it put a shortcut to the Vulkan Configurator on the user's desktop. I see a lot of Vulkan beginners who just don't know about the Vulkan Configurator, and making it more discoverable would help them

5. Make fish-compatible setup-env file
6. SDK is getting better and making our lives easier every year. :)
7. Very happy with the tools provided in the Vulkan SDK.
8. None this year. It's mostly been a pleasure to use.
9. I'm happy :)
10. Thanks for improving Vulkan! It is great!"
11. I appreciate the effort of creating and maintaining the SDK. Many thanks, especially for vkconfig.
12. Have better support for windows 11.
13. You're doing great! Keep with the same pace!
14. Always appreciate the amount of work that is put in the ecosystem and the professional and friendly contact with the Lunar-G people. Especially the last has been very useful to me
15. Please keep it working on older Linux systems (older GLIBC), I cannot use the latest SDKs on all my Linux systems.
16. Thank you for everything you've done. Both as a hobbyist developer for enabling me to do graphics in a significantly more effective and pleasant manner than in OpenGL days and as an user, for setting up the environment so software like dxvk and vkd3d-proton can exist, letting me use Linux for gaming too.
17. More frequent updates, less codebase breaking changes in small releases, or at least with more warning/guides to updating (e.g. vk::DispatchLoaderDynamic -> vk::detail::DispatchLoaderDynamic)
18. Where is chucked vulkan spec?
19. Having a more easier high level framework as well, or at very least guidance on higher level middleware that builds on top of Vulkan.
20. All I want is a tutorial on how to get Vulkan to work on my mac without XCode. I usually use vim and tmux. Would be happy with VSCode. I need to be able to use a make file. There's a lot of settings there.

How important are the following APIs for your development TODAY

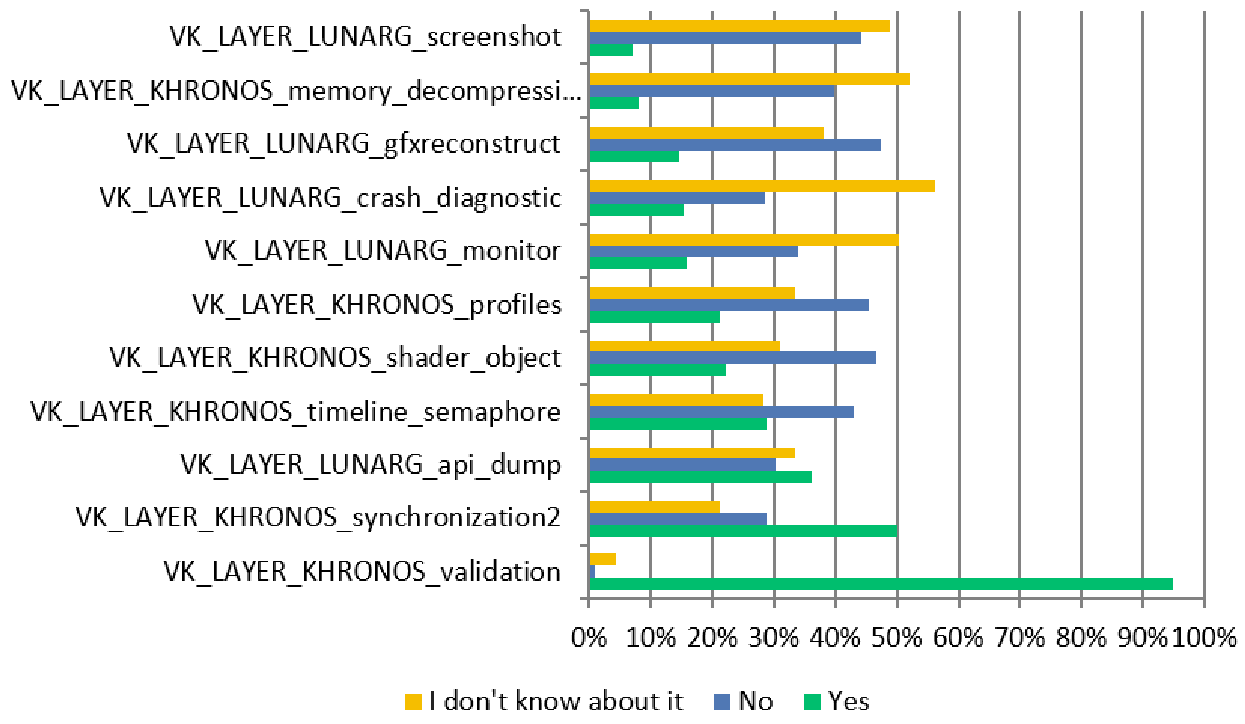


How important are the following APIs for your development IN THE FUTURE

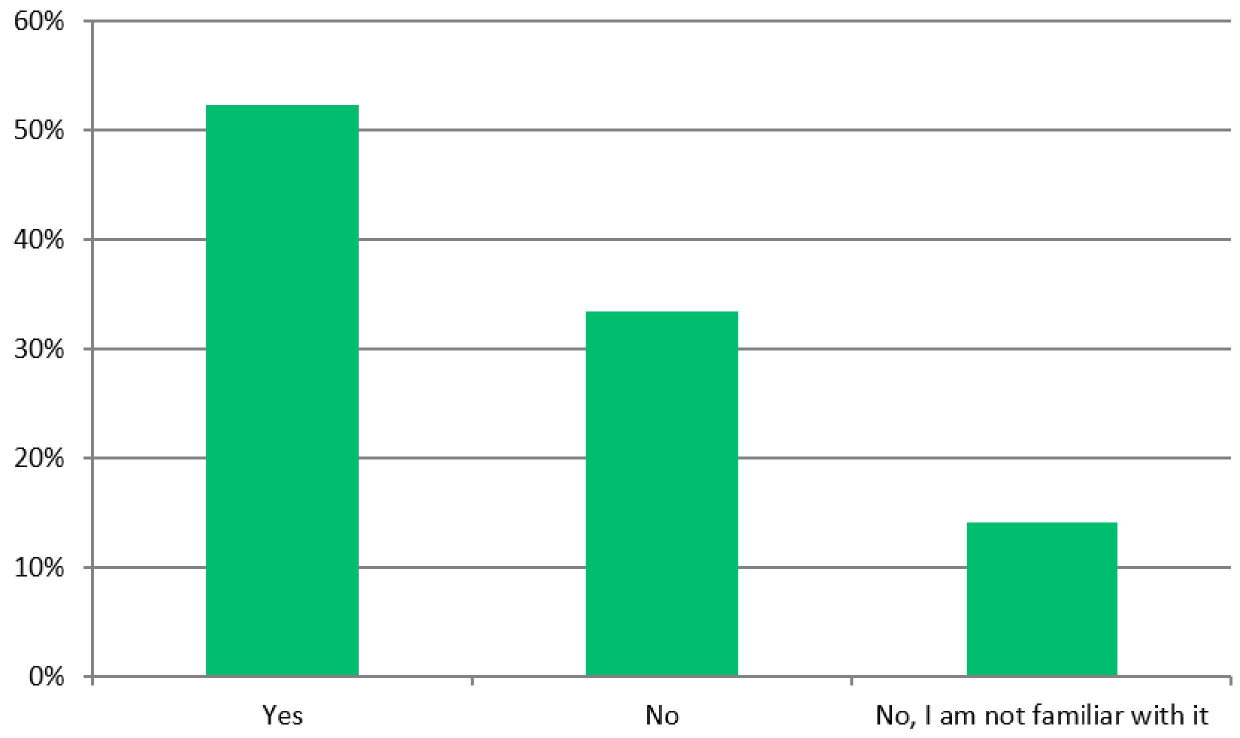


As expected, webGPU will increase in importance in the future.

Which of the following Vulkan layers do you use? (answer choices: yes, no, don't know about it)



Do you use the Vulkan Configurator (vkconfig)?



Vulkan Configurator Open-Ended feedback

All open ended feedback is listed here, regardless if the same comment is said many times.

1. Usability
    a. A bit more documentation on what all the checkboxes on the right mean would be nice (maybe a right-click to go to a detailed description of what it catches), but honestly I think it's fine as is.
        i. LunarG comment: All the settings have tooltips in the application.
    b. Detailed Documentation
    c. The UI is very confusing
    d. I find vkconfig weird to use. When I enable API I can never find where the dump goes to half the time
    e. Make sync validation more discoverable. The current right-hand sidebar of options is great for advanced users but I worry that beginners get overwhelmed by the options and don't know how to navigate them"
2. Happy
    a. none, its great
    b. Working great no requests currently
    c. none, works as expected

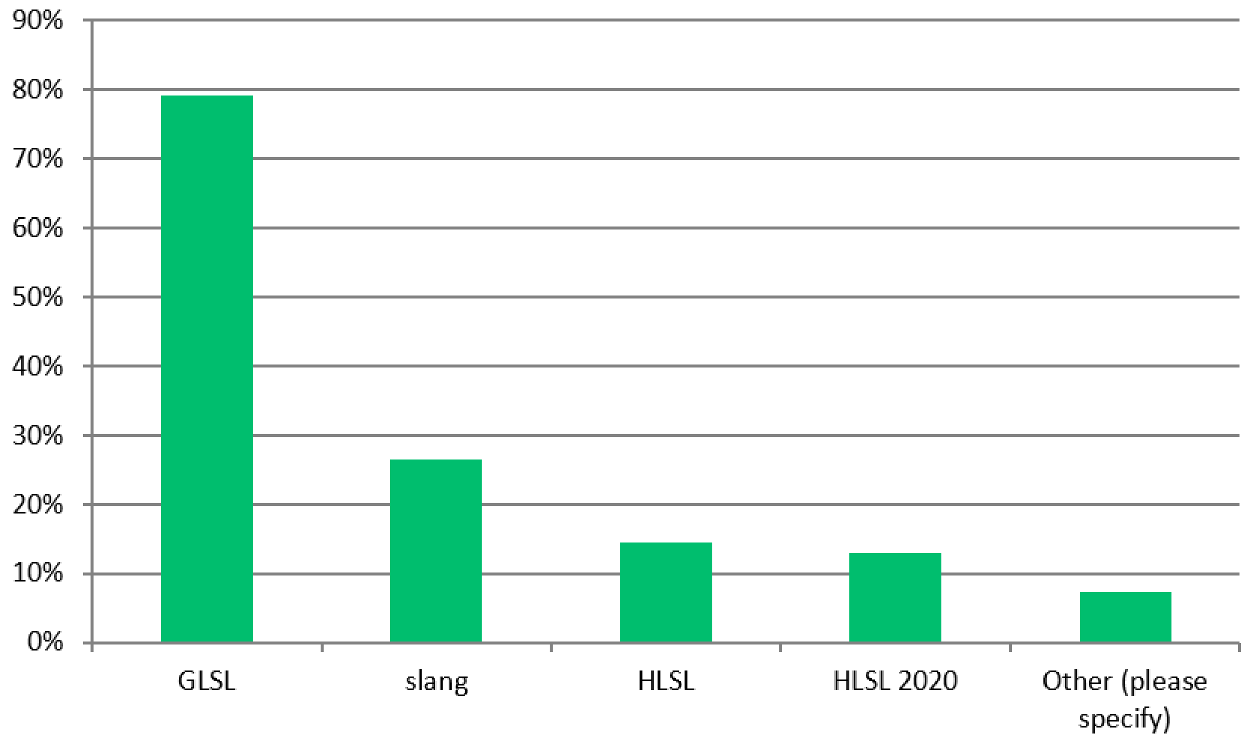
    d. Honestly, it's great.
    e. None. It's performed all of my needs.
    f. None, all previous suggestions have been implemented. <3
    g. still need to get more familiar with vkconfig3, already seeing many usability improvements and not missing anything atm
    h. In-App documentation.

3. Quality/stability
    a. Improve stability (it crashes when I try making a custom profile with synchronization + validation).
    b. I see some errors when I first start it up, every time. ".../Vulkan/RTSSVkLayer64.json is not a valid layer manifest", etc....
        i. LunarG comment: This issue has been fixed in the 1.4.309.0 SDK: https://github.com/LunarG/VulkanTools/issues/2285
    c. If DPI changes (connecting and disconnecting external screen to laptop) vkconfig GUI layout gets broken.
4. Enhancement Requests
    a. Make some data searchable. For example, opening Vulkan Info, and selecting a device, opens up a huge list of properties. It would be great to be able to search it by typing a name.
        i. LunarG comment: We have created a feature request in the github repository for this enhancement.vkconfig: Add searchable diagnostic #2281
    b. Recompile with qt6
        i. LunarG comment: This has been completed and is delivered in SDK 1.4.309.0 for the Linux tarball, Ubuntu packages, Windows on ARM SDK, and macOS SDK. Qt6 support is coming soon for the Windows X64 SDK.
    c. maybe build into vkconfig a one click way to use validation layers from github (latest commit or specific) since there's build artifacts now could maybe even download them
        i. LunarG comment: This may be a nice ease of use enhancement to vkconfig. An issue has been logged in the LunarG/VulkanTools repository as a possible future enhancement. https://github.com/LunarG/VulkanTools/issues/2283
    d. I would like to be able to skip debug breaks for messages with certain IDs or from certain sources, e.g. the Vulkan loader. I don't want to filter these messages, just skip the debug breaks.
        i. LunarG comment: Most likely a change is needed in the validation layer and not vkconfig. A tracking issue has been created in the validation layer repository: https://github.com/KhronosGroup/Vulkan-ValidationLayers/issues/9650

   e. While relatively niche (compared to the more common developers), more tools for IHVs developing ICDs would be useful. Stuff like making it easier to handle the variables for overriding which ICD is chosen.
      i. LunarG comment: This is already under investigation to provide more control over handing Vulkan drivers (e.g. lavapipe). Stay tuned for a future enhancement.

5. Other
   6. I mainly use it to configure applications that I run from IDE/terminal. I start vkconfig set the config minimize it. Some times a day I reconfigure based on needs. Tool does it job for me.
   7. I hate that I need to have vkconfig open to get the GPU debug printf extension to work. This seems to be a never ending issue. End users writing shaders should be able to printf without needing to know intricate Vulkan configuration details.
      a. LunarG comment: You can use environment variables or VK_EXT_layer_settings to configure the validation layer as well, if you don't want to open the Vulkan Configurator. See the SDK documentation: https://vulkan.lunarg.com/doc/sdk/latest/windows/khronos_validation_layer.html and https://vulkan.lunarg.com/doc/view/latest/windows/layer_configuration.html
   8. Save settings while it's not active
      a. LunarG comment: Vulkan Configurator already does this. If you have a situation where that is not working, please submit an issue to VulkanTools on vkconfig.
   9. Add a default profile that enables almost everything
   10. I also do not like VUIDs to be negative numbers. It might be hex number or dec number but negative looks like that the programmer mistakenly wrote %i instead of %u in the formatting string, or something like that. Not too serious problem. But it might be easy improvement.
      i. LunarG comment: This is an issue for the validation layer.
   11. I'm too new to this to offer any sort of useful suggestion.
   12. ability to emulate different devices with different drivers (preferably different vendors and models), operating system as well
   13. Specific overrides, like adding only a single layer on top of what was specified in a program (e.g. adding VK_LAYER_shader_object to a program for a platform that doesn’t support shader objects)
   14. Make vkconfig more discoverable, perhaps with a desktop shortcut
      a. LunarG comment. After SDK installation, the user is given the option to start vkconfig. It is also in the Windows start menu.

What is your preferred shading language? Check all that apply

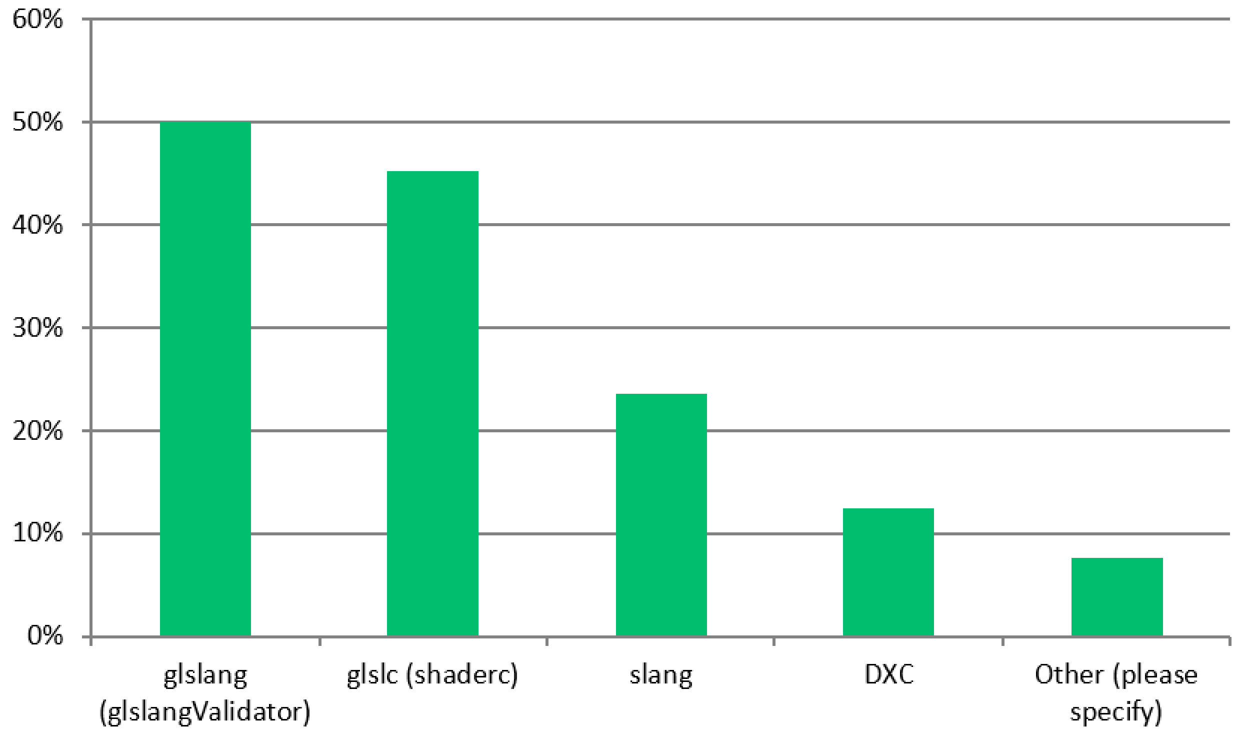


Compared to last year, slang has made a significant jump in shader language preference. In 2024 it was preferred by about 3% of the population vs. ~27% today.

Other, commercial developers:

1. NZSL
2. MLIR
3. zig
4. A subset of Rust itself
5. custom language
6. Nabla HLSL STL
7. Nabla HLSL STL
8. WGSL
9. Blender GLSL
10. We want to transition to slang
11. vcc
12. My custom language that compiles to SPIR-V
13. rust-gpu
14. Ruamoko (work in progress, not ready for others)
15. Vcc
16. Rust (rust-gpu)

What is your tool of choice for generating SPIR-V? Check all that apply



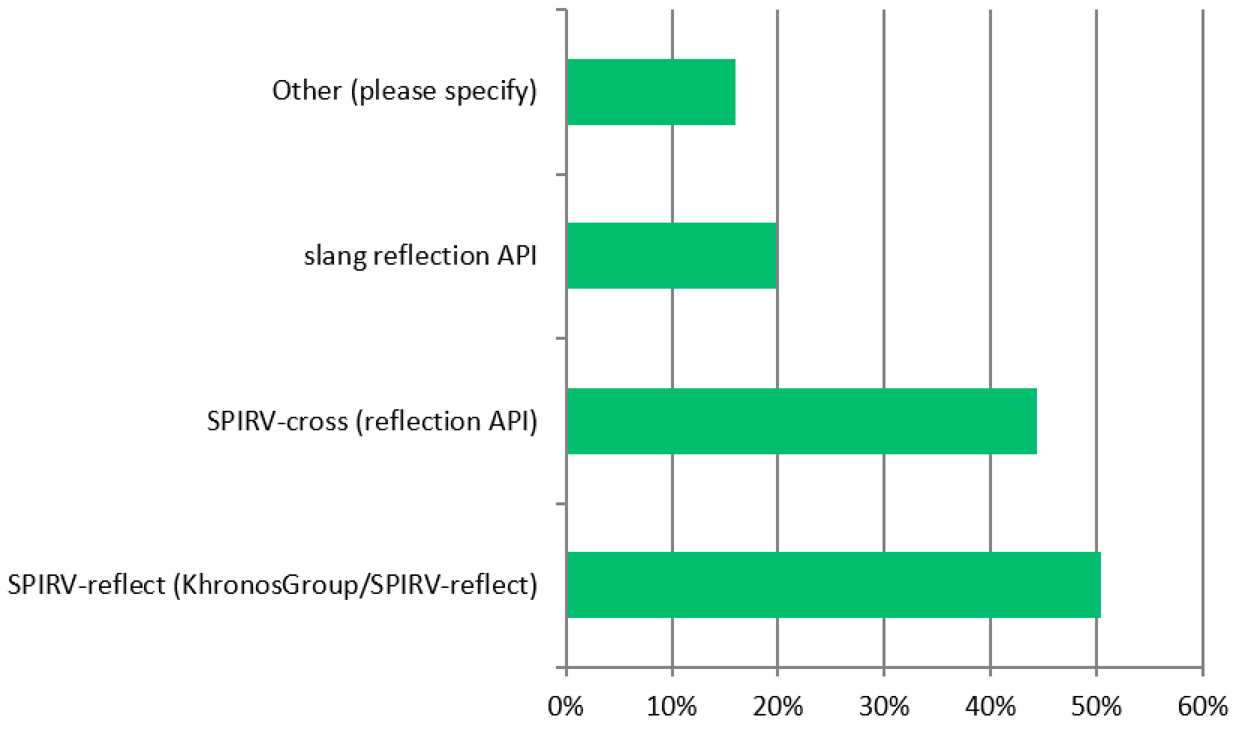
Slang made a significant jump in the preferred tool for generating spir-v. In 2024 it was at about 3% overall. This survey is showing it at about 27%.

Other, commercial developers:

1. NZSL
2. proprietary translation tool
3. MLIR
4. naga
5. Custom
6. Naga
7. custom generator
8. Clang
9. Clang
10. naga (note: NAGA is a shader translator and validator. Part of the wgpu project)
11. vcc
12. Naga (Rust)
13. qfcc (quakeforge, wip)

14. Vcc
15. embedded shaderc into my app
16. rust-gpu

What is your preferred tool for SPIR-V reflection? Check all that apply



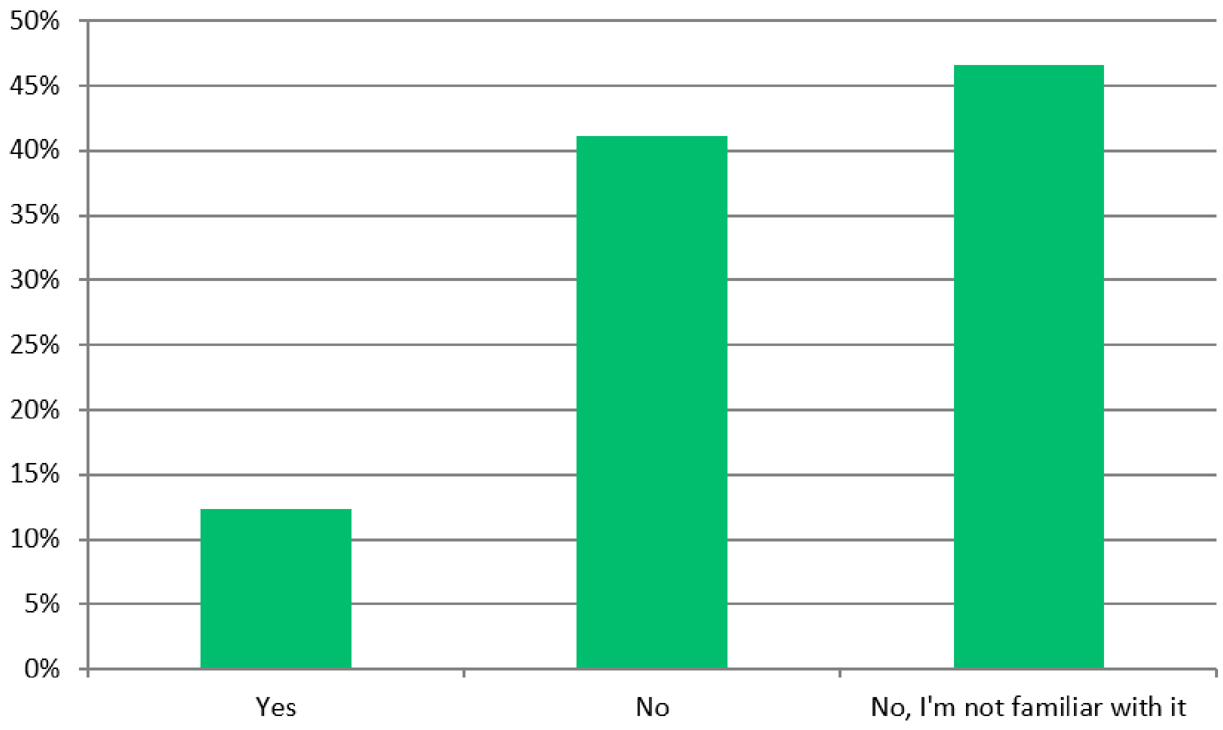
There are multiple reflection tools available. The purpose of this question was to see if there was an obvious tool that wasn't being used. Evidently not.

Other category:

1. NZSL
2. I don't use SPIR-V reflection
3. N/A
4. I don't use reflection
5. RenderDoc
6. SPIRV-reflect sucks
7. Naga
8. not needed
9. none.
10. unfortunately not used yet.
11. spirv-tools

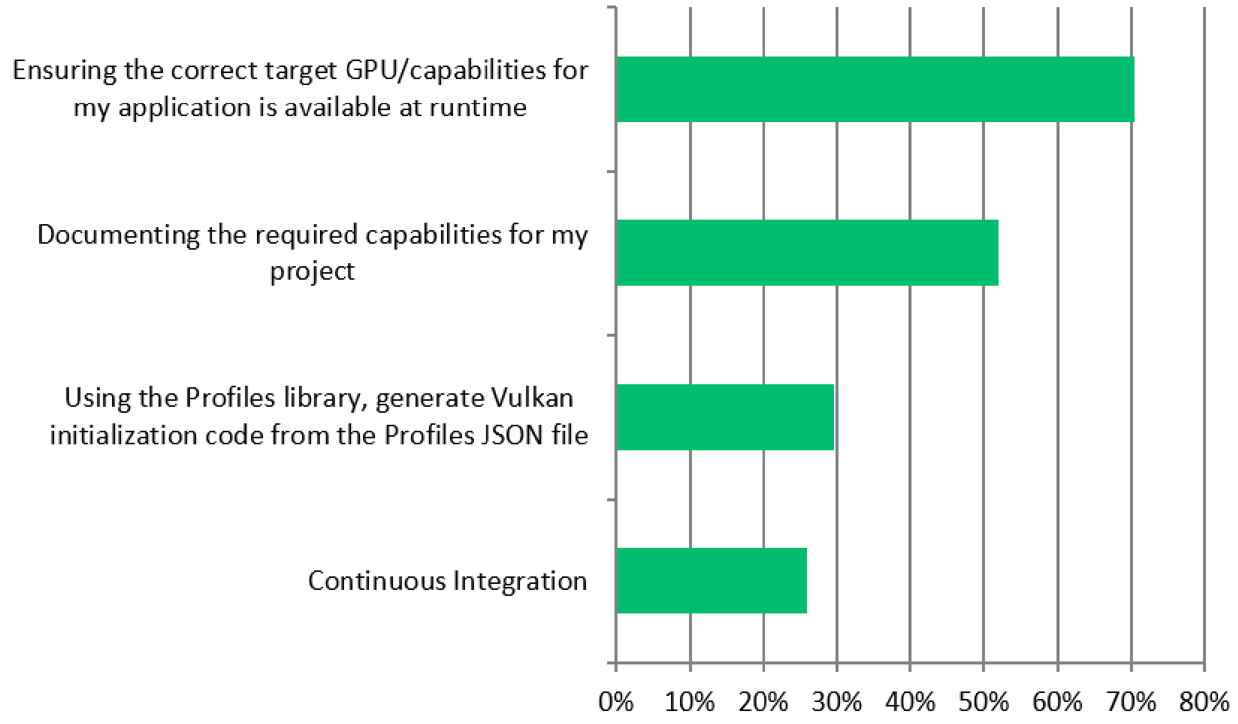
12. The available SPIR-V reflection tools are either too cumbersome, complicated, or lack features I need, so I wrote my own.
13. We want to transition so slang (but not active yet)
14. rspirv (rust crate)
15. Hardcoded
16. https://docs.rs/spirq/latest/spirq/
17. I roll my own
18. parsing the bytecode in C
19. Custom library
20. spirv-reflect
21. I don't use reflection for now

Do you use the Vulkan Profiles toolset?



Similar rates of usage as the previous year.

If you are using the Vulkan Profiles toolset, what are you using them for? (check all that apply)



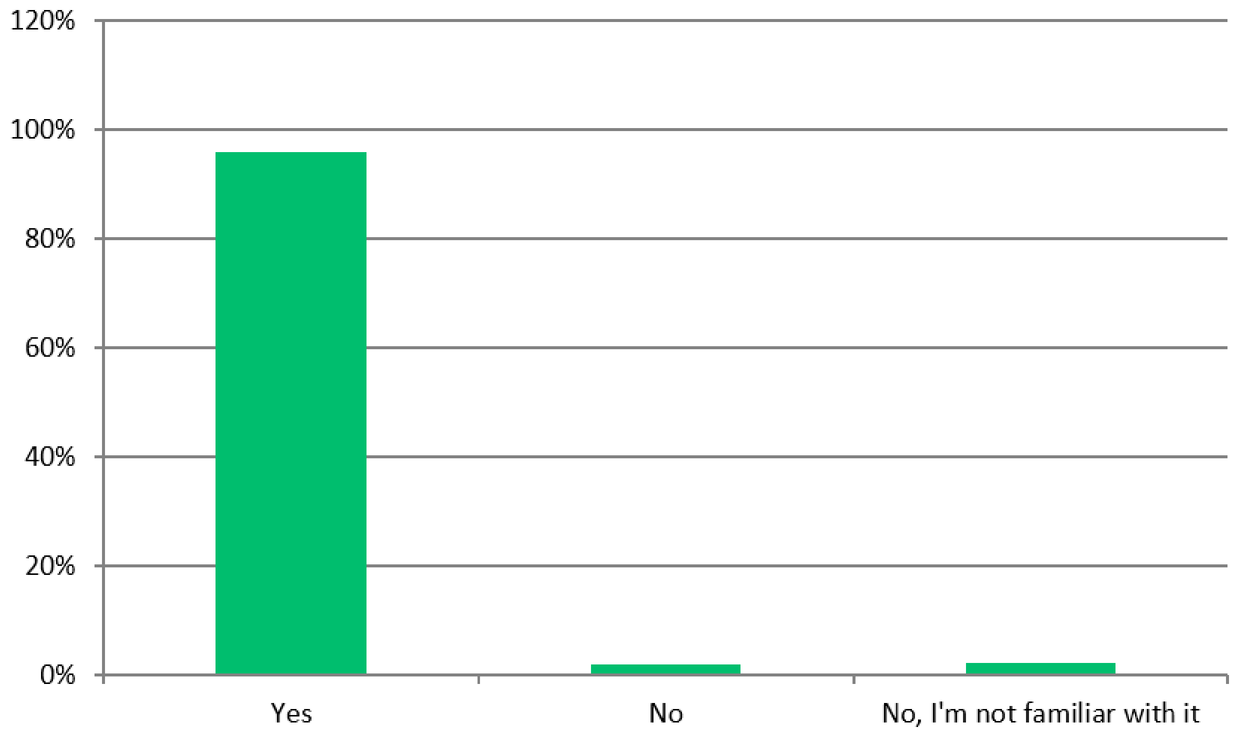
There were no "other" usages of the Profiles toolset listed by survey respondents. So the Profiles toolset is being used for the expected use cases.

When using the Vulkan Profiles toolset, what are the inhibitors for you to use them easily or effectively?

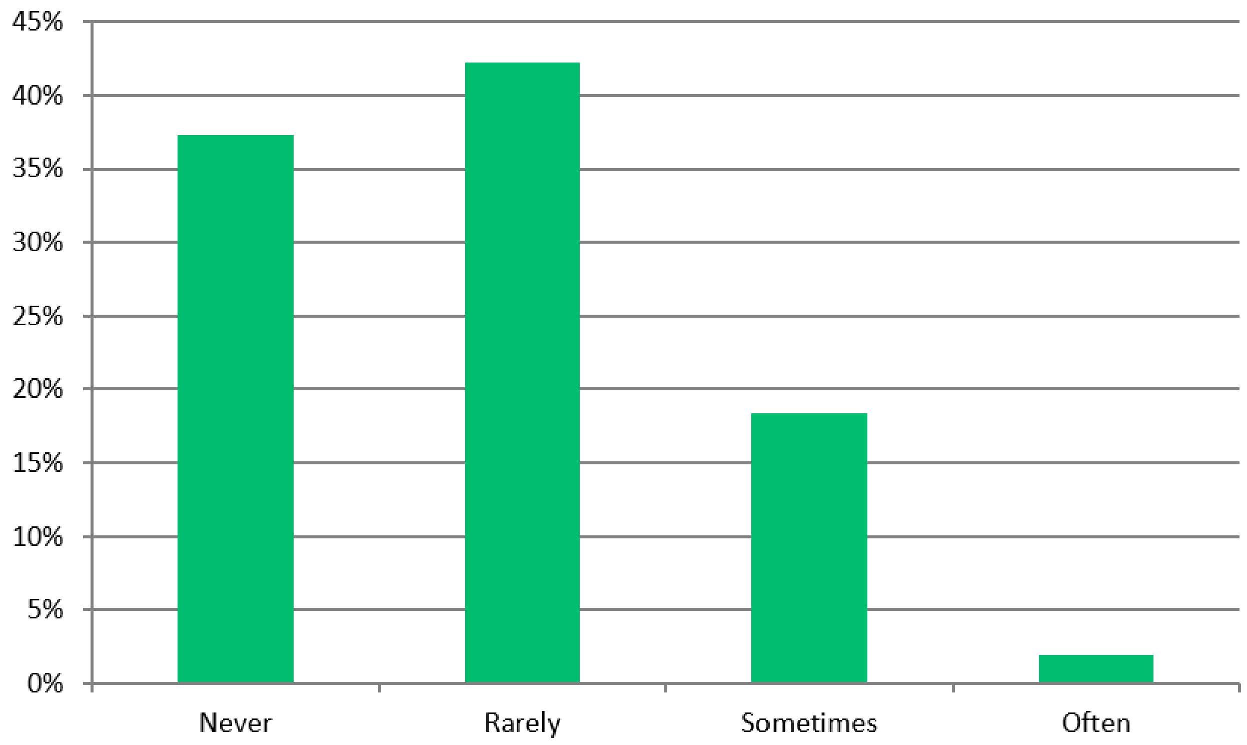
1. Can't scrape large amounts of GPU reports from gpuinfo.org
    a. LunarG comment: There is an API to query gpuinfo.org documented here: https://github.com/SaschaWillems/vulkan.gpuinfo.org/blob/master/docs/api.md
2. No changes needed that I could think off
3. Build environment must support the instance extensions to have the profile included in the binary.
    a. LunarG comment: To support this would require some significant changes to the Vulkan Loader. This was evaluated and determined that it would not be pursued because it would be a breaking change (can't break compatibility until a 2.0 release).

4. No formal specification like Vulkan's vk.xml, this makes generating a Rust wrapper have to deal with parsing C, or manual as I currently do it.
5. How restrictive the extension capabilities are
6. Invest more in the Vulkan Profiles Library
7. More investment in the Vulkan Profiles Library, such as formal specification with a .xml to match that of Vulkan.

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

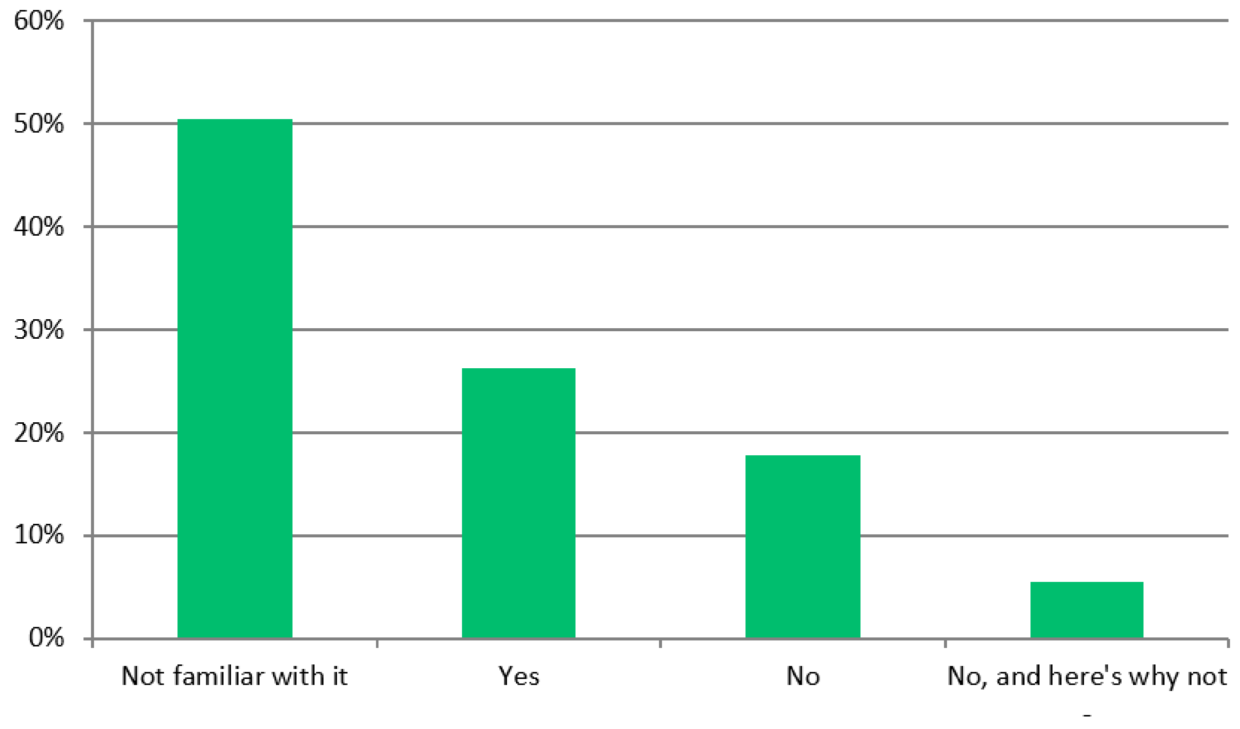


How often does the performance of the Validation Layers inhibit effective use of them?



LunarG comment: Similar to last year. Those who are impacted by the performance of the validation layer are commercial developers and the performance impact is not always there. Most likely when using GPU-AV and synchronization validation is when the performance hit is observed.

Do you use GPU Assisted Validation (GPU-AV, GPU-Assisted, VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT)

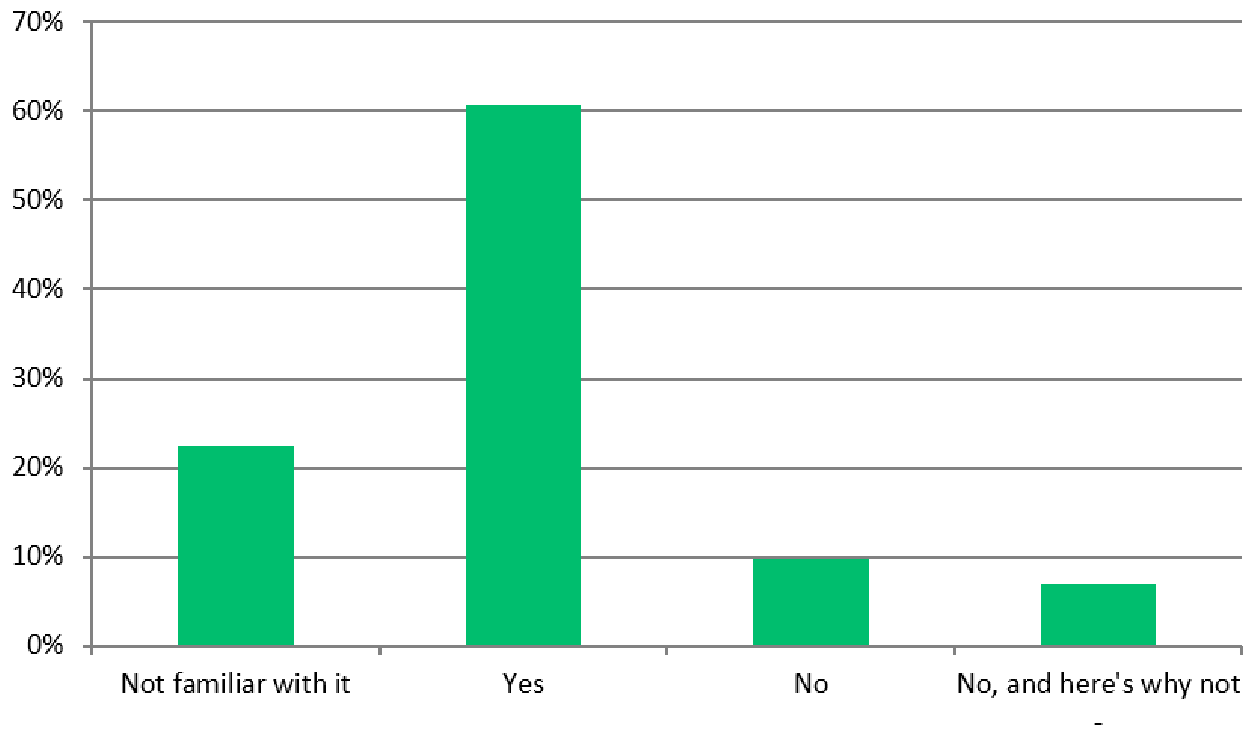


Comments about why they are not using GPU-AV:

1. Didn't try yet
2. but I intend too very soon
3. Going to start after this conference
4. I've had a couple issues with it causing crashing and it causes an unacceptable performance impact. Often tools like renderdoc or NVIDIA Aftermath are enough for my use cases
5. -
6. Far too slow
7. Tried to have our test suite run with it, it was too slow.
8. every few releases it will crash or blackscreens my app, when it does work frame times are horrible so I tend to avoid it unless something is really wrong
9. Last time I used it, it reported false positives and introduced GPU crashes that were not present without it
10. Last time I checked it wasn't checking out of bounds accesses to storage buffers
11. Causes the Nvidia driver to crash - but i haven't tested with 572.16

LunarG comment: With more validation moving to the GPU, for developers to get a fuller validation coverage, they really need to be enabling GPU-AV. GPU-AV is still under active development within LunarG and we are eager for you to submit issues. Our goal is to improve coverage and performance to make it usable most of the time.

Do you use Synchronization Validation (VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT)



"Here's why not" comments:

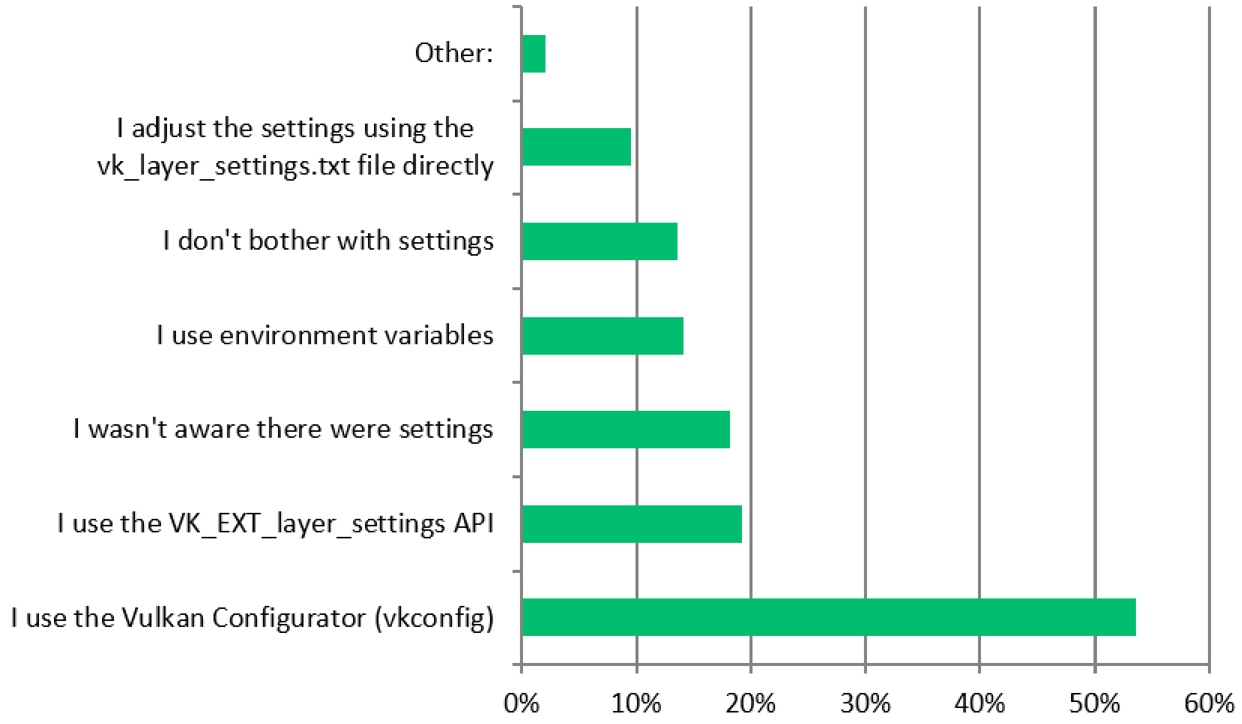
1. I'm scared of what it might tell us. Ignorance is bliss. Also, using it on Android/iOS through a MAUI project is cumbersome.
2. I usually don't have large sets of command buffers which need intricate synchronizing
3. It doesn't work for timeline semaphores.
    a. LunarG comment: Synchronization validation for timeline semaphores was completed in 2024. Is there an issue with it or is this statement made because the user had not yet updated to a version of the SDK or validation layer that provides support?
4. It's incomplete. Has too many false positives. Reporting issues results in several months of waiting before a non-fix is made.
    a. LunarG comment: Somewhere last year we started to follow a policy of false-positive free behavior. Some functionality was disabled that causes false-positives, some things were fixed, some false-positive friendly features need to be turned on manually and are marked as heuristic. We are pretty serious about keeping it this way and currently we are not aware about active

false-positives (but there could be some). Please open issues because that's something we care about.

   b. LunarG comment: Regression defects are given the highest priority before working on new validation. If any regression is detected, please submit a github issue indicating it was a regression.

5. I find that it reports on many things that I see as perfectly valid and work across NVIDIA and AMD without issues
   a. LunarG comment: Yes, if you know how specific hardware works it might be valid for that hardware but still be a violation of the Vulkan spec and the validation checks against the spec. It's fine to violate spec in production environments when you know what you are doing. It is also possible that for timing purposes you are getting lucky.
6. My application skips synchronization on purpose and this layer will complain

How do you adjust settings for the Validation Layers? (check all that apply)



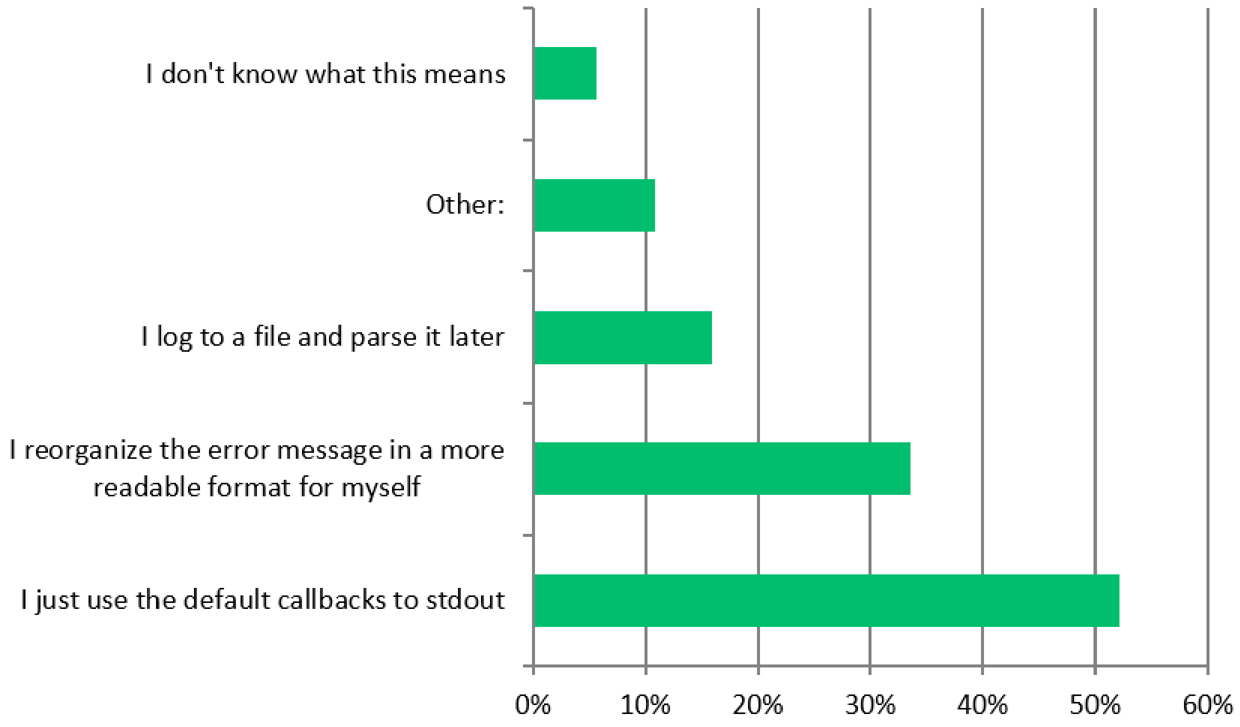
Other comments:

1. or VkInstanceCreateInfo
2. Ignore false warnings from code
3. I will use VK_EXT_layer_settings inside Vulkan code in the future
4. I haven't had a need to, but I'm aware of the vkconfig if I need to

LunarG Take-aways:

1. We still need to support the vk_layer_setting.txt file
2. Glad to know that vkconfig is the true majority
3. People do use settings mostly (This is good!)
4. There are still a good number of people (about 25%) who don't use settings (or not aware of them) so the "out-of-the-box" needs to still be a good experience and we may need to do more promotion of the value of using settings.

Do you parse the messages in your own callback? (Check all that apply)



Other comments:

1. I regex out the verbose debug message text, though it seems to frequently change and break.
    a. LunarG comment: There is now a JSON format that they can turn on (as of the 1.4.309.0 SDK) and using that, likely no need to do a regex and can ensure the JSON schema will not break in the future
2. I pass them to my logging framework.
3. Own logging system with partial parsing
4. I run them through a custom logging system.

5. I have my own logging utilities, but I don't parse anything, aside from the occasional string search to remove redundant errors.
6. I use the log callback to rouge the log messages into my own logger, which both prints them to a file and to stdout
7. I forward the messages to our logging system
8. Basic Windows OutputDebugStringA callback
9. I don't parse the messages, just read them. I do check for absence of validation-warnings/errors in unit/functional tests though
10. We record the messages in our app's log.
11. I filter messages.
12. I gather the output and feed it into my regression testing system
13. set breakpoints etc.
14. Redirect between stdout and OutputDebugString as appropriate (Windows)
15. In unit tests, they are logged in a file, when the application is running, they are both written to stdout and to a log file.
16. Yes. To ignore false positives.
17. I ignore certain errors instead of terminating my App
18. I log it as formatted output so I can see what the message, messageId and object(s) are (etc.)
19. I use trasnlation to log file(for All Platforms) + stderr for Desktop Unix and OutputDebugString Windows
20. simply print to stderr and exit(1)
21. I log to stdout and raise SIGTRAP or debug break

LunarG comment: Wow. Over half of the people use the default format. A change was just made in the validation layers that will be delivered in the 1.4.309.0 SDK that makes improvements to this format.

How could the validation layers be improved? (Open-ended)

All open ended feedback is listed here, regardless if the same comment is said many times.

1. More/better/timely coverage:
    a. better & more complete validation layer support
    b. adding more VUID coverage
    c. More checks, obviously.
    d. Occasionally the lack of validation for some parts of the API.
    e. GPU assisted validation doesn't check all access hazards.
    f. More video validation layers
    g. More validation for VK 1.3 and 1.4 features and extensions.

        i. LunarG comment: Validation for 1.3 and 1.4 is complete. New extensions are validated shortly after public release.
    h. More ray tracing validation coverage
        i. LunarG comment: All of the CPU based validation for ray tracing is complete. There is a lot of validation for ray tracing that must be done on the GPU. Here is a tracking issue for that work: https://github.com/KhronosGroup/Vulkan-ValidationLayers/issues/9446
    i. Timeline semaphores and queue family transfers
        i. LunarG comment: Synchronization validation of timeline semaphores was completed as of the 1.3.296.0 SDK
    j. The validation layer's don't work with timeline semaphores.
        i. LunarG comment: Synchronization validation of timeline semaphores was completed as of the 1.3.296.0 SDK
    k. Better support for descriptor indexing and descriptor buffers.
    l. New extensions are not that important for me, but completing the basis is.
    m. have validation-layer coverage for new features as early as possible
    n. Maybe more validation coverage? Validation layers are the best tool of all, probably. Improving them is very useful!
    o. better sync and oob access validation when buffer device addresses are involved
    p. Fewer false positives. These practically force you to use gigabarriers for everything if you use bindless, otherwise you get worthless GPU-AV.
    q. Synchronization validation does not catch some errors.
    r. GPU assisted validation needs some love or at least refresh the ecosystem's memory on what checks are implemented and what checks are missing.
        i. LunarG comment: Agreed. And GPU-AV validation is currently a top priority at LunarG.
    s. Acceleration structure, shader binding table should be able to detect that they are not created correctly
    t. More synchronization validation
    u. Focus on HLSL and GPU assisted validation since most things move to GPU driven
    v. Debugger support for descriptor buffers

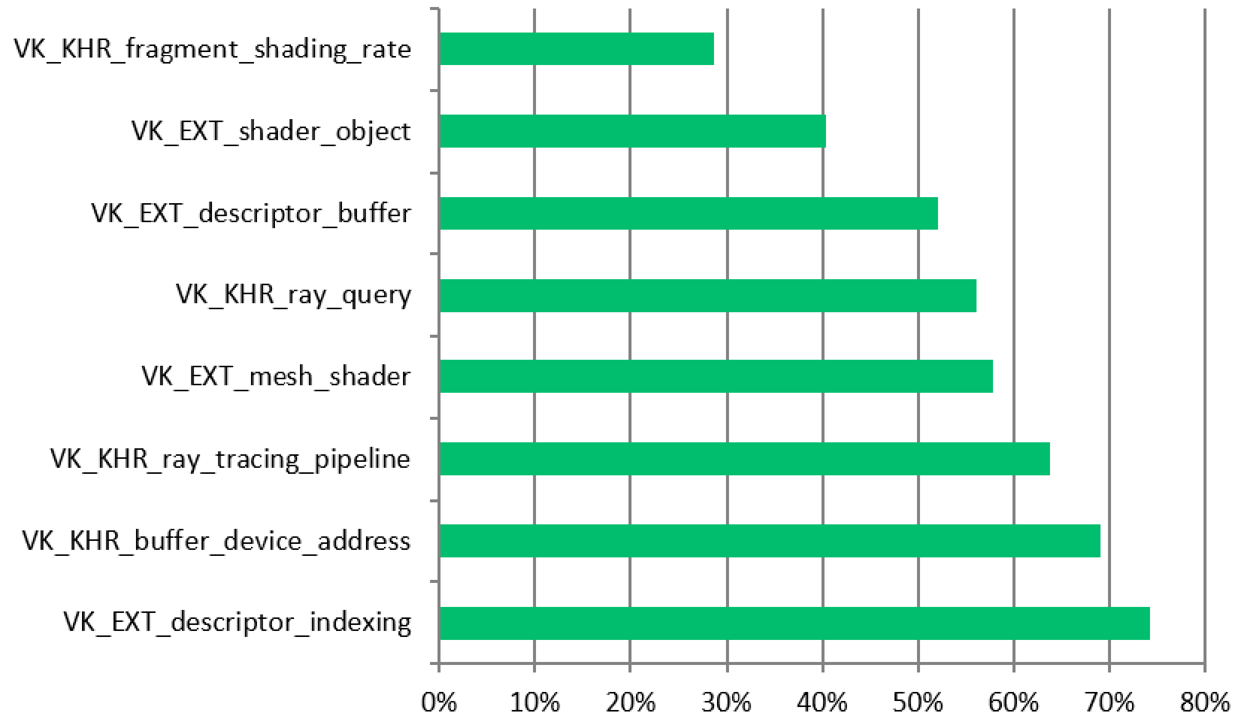
2. Performance:
    a. When I enable debugPrintfEXT, the fps of my app drops a lot. I hope that the fps will not drop even if I enable debugPrintfEXT.
        i. LunarG comment: The 1.4.309.0 SDK should be faster now since we now only do work if the app has printf. If the user is not wrapping their debugPrintfEXT with a condition to only print once, and they are printing a value for a fragment shader that is running a million times, that will be slow as you are trying to print a million times
    b. Faster GPU Assisted Validation

   c. Less performance overhead.
   d. Improved performance
   e. perf
   f. more performance
   g. just being faster in general
   h. improve performance.
   i. Performance of the validations layers
   j. 1000x synchronization validation layer.
3. Error messages:
   a. The information provided should be more precise
   b. The validation layers should be more consistent and provide better information for debugging.
   c. provide a mory readable and clean format output.
   d. Make errors more readable?
   e. Clearer more user-friendly messages
   f. Continue improving error messages.
   g. I think they're fine, though to someone relatively new, it seems verbose and probably too "spec"-y. I think I've gotten used to parsing it in my head and understanding where I messed up.
   h. If their error handling were more readable…
   i. Improved formatting of error messages.
   j. reasonable format for printing and maybe structure (eg, json)
   k. Better readability, or maybe a UI you can use to "inspect" the errors better
   l. Cleaner error messages
   m. Readability of error messages could be improved
   n. More human-readable, maybe suggesting fixes in the registry or validation message itself
   o. Validation errors are hard to read
      i. LunarG comment: In response to all the comments above, the 1.4.309.0 SDK will have a cleaner format. There is also an option added to report JSON and then the user can format how they find it best for them.
   p. Let the user specify enough information to report errors sooner (e.g. in cases where there is just one command buffer being recorded per frame and it runs sequentially). Getting an error when you submit and not being able to trace where it came from sucks. Alternatively, allow the user to place breadcrumbs and report the most recent breadcrumb executed in the message."
      i. LunarG comment: At the 2025 Vulkanised that took place in February 2025, Spencer Fricke gave a presentation about debugging your GPU workflow. The presentation is here: https://www.vulkan.org/user/pages/09.events/vulkanised-2025/T39-Spencer-Fricke-LunarG.pdf Starting around slide 40 you can find specific information about vkCmdBeginDebugUtilsLabelEXT.

   q. I do not understand how to parse synchronization layer messages and correlate all the data they provide with the API dump output.
   r. sync validation messages read like sacred texts you need to journey up to the wise one to decypher, generally finding out which cmds are involved is more painful than it should be
   s. More sync error info
   t. Better validation error output for sync errors, they feel hard to track down
      i. LunarG comment: With the 1.4.309.0 SDK, improvements have been made to the synchronization validation error messages to make them human readable.
   u. Debug message from VK_KHR_DEBUG_UTILS_EXTENSION_NAME was hard to read, maybe it should be multiline.
   v. The feedback from the validation layer is still confusing and not accurate enough. At the end of last year, we spent a month finding a trivial error just because it occasionally showed up in the validation layers.
4. Best Practices
   a. More performance checks
   b. would be interesting if developer had received "tips" what can be improved in API calling for optimal usage.
5. Configuration
   a. Better documentation for the validation layers and how to work with them.
      i. LunarG comment: How to configure the validation layers is fully documented here::
         1. https://github.com/KhronosGroup/Vulkan-ValidationLayers/blob/main/docs/settings.md
      ii. Your comment made me realize that the same documentation should be available in the SDK documentation and currently is not there. This will be fixed in the 1.4.309.0 SDK
   b. Somehow more aggressive defaults without the use of environment variables or vkconfig
      i. LunarG Comment: As of the 1.4.309.0 SDK, useful warnings that were not on by default will now be enabled by default.
   c. It is very complex to obtain and install for mobile device emulators
   d. more stuff enabled ootb,
   e. Make the debug printf layer work well without vkconfig, eg by exposing all options through extensions/flags in the API that an engine can directly work with.
      i. LunarG comment: There is now a `VK_LAYER_PRINTF_ONLY_PRESET` environment variable that can be used to quickly turn it on without vkconfig (also can be set through the API with VK_EXT_layer_settings)
   f. I have found 3 different ways to configure validation layers and none of them seem to work with rust/vulkanalia.
   g. more modern interface, simpler integratrion

   h. The BestPractices-specialuse-extension warning for debug_utils is shown on a debug build. It should only show on a release build otherwise people will simply filter it out which defeats the purpose and adds extra work.
   i. If it cannot be determined whether an application is running in release or debug I do not think this is a proper validation
   j. better integration with IDEs
6. Quality/bugs
   a. Focus on validation layers, make them perfect
   b. improving GPU-AV.
   c. Just less bugs in the validation layers. I've reported some but haven't got around to others. Few but frustrating.
   d. Fix all erros,
   e. Today I see 240 issue form github
   f. We have still a lof of error for Vulkan Validation Layers
   g. Debug printf often crashes inside the validation layers"
   h. One is definitely printf not appearing consistently.
   i. This layer is an all or nothing deal and if it doesn't even support basic but essential vulkan 1.0 core functionality (like qfot) then i cant use it. I would like to use it though because i have a bug that looks like sync, but the layer is throwing a lot of false positives and i have to modify my code to not use the stuff i mentioned. I dont want to validate modified code, i want to validate the code i have to release to my users.
7. Diagnosing failures
   a. use label names if available. provide hint/example how to fix them online.
   b. Better debugging of situations where my code is spec-compliant but not quite working
   c. Not much, maybe I would like a stack trace from a validation error but i think i can do it on my own
8. Invest more in Validation layers
9. I'm 100% supporting to tie feature-development with CTS and validation-development.
10. The validation layers have helped immensely and a great resource! For debugging applications and the new GPU tools will come help greatly.
11. they are great, keep it up!
12. Keeping up the good work
13. debugPrintf is also really useful for my use case.
14. The documentation links, but this was just fixed!!! So no comment at this time
15. Just thinking out loud: Resource tracking using pre-given strategies.
16. Validation Layers could help in spotting usage of undefined/discarded images and buffers."

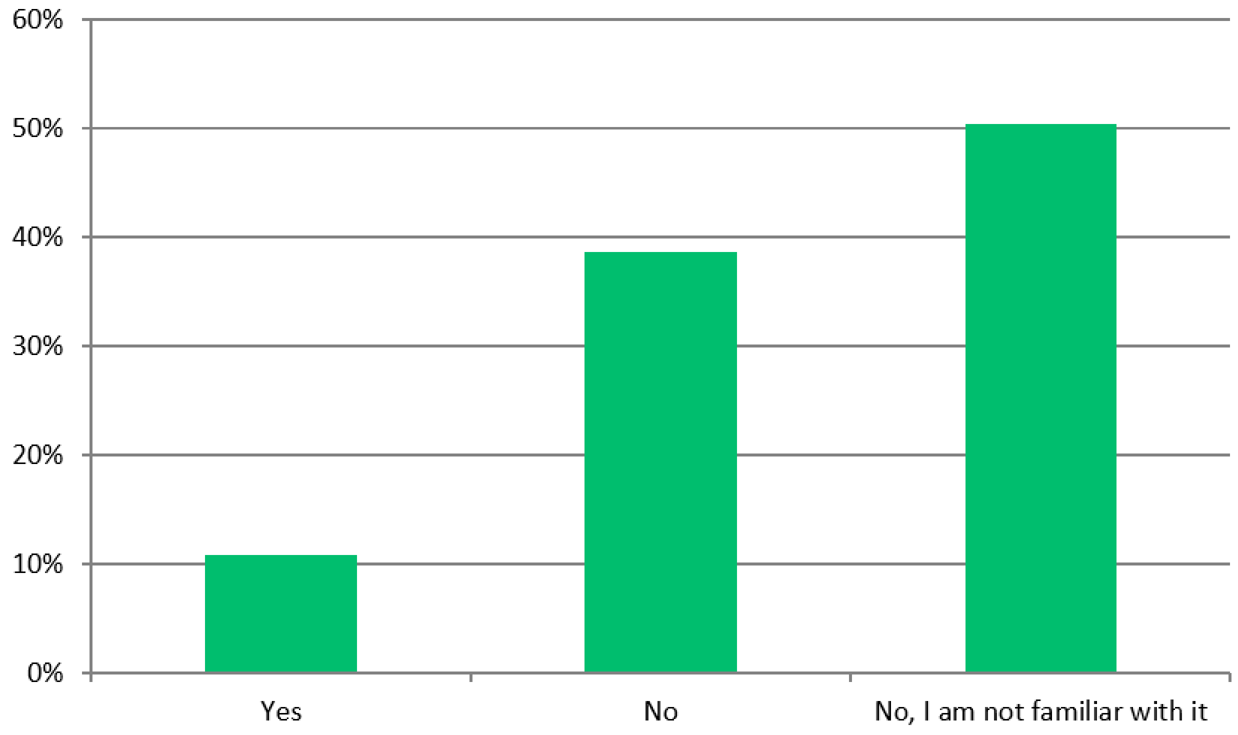
Please indicate which extensions you currently use or plan to use (check all that apply)



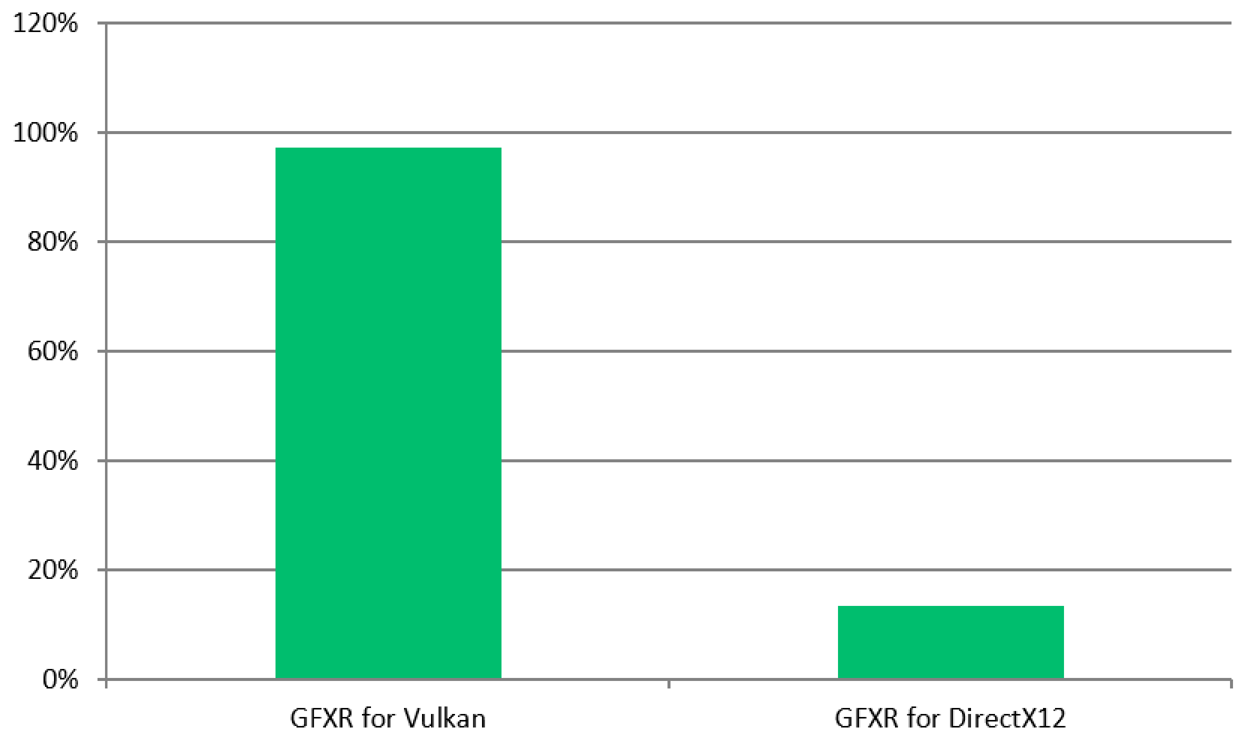
LunarG Take-aways:

1. Descriptor Indexing and Buffer Device Address remain popular and LunarG will continue to focus on GPU-AV validating them well.
2. Surprised that Ray Tracing/Query is so popular. This is currently WIP to do a better job of validation.
3. Shader Object didn't suddenly become "the way" to do things yet.
4. Mesh Shading probably should get more attention
5. Descriptor Buffers is more popular than expected.

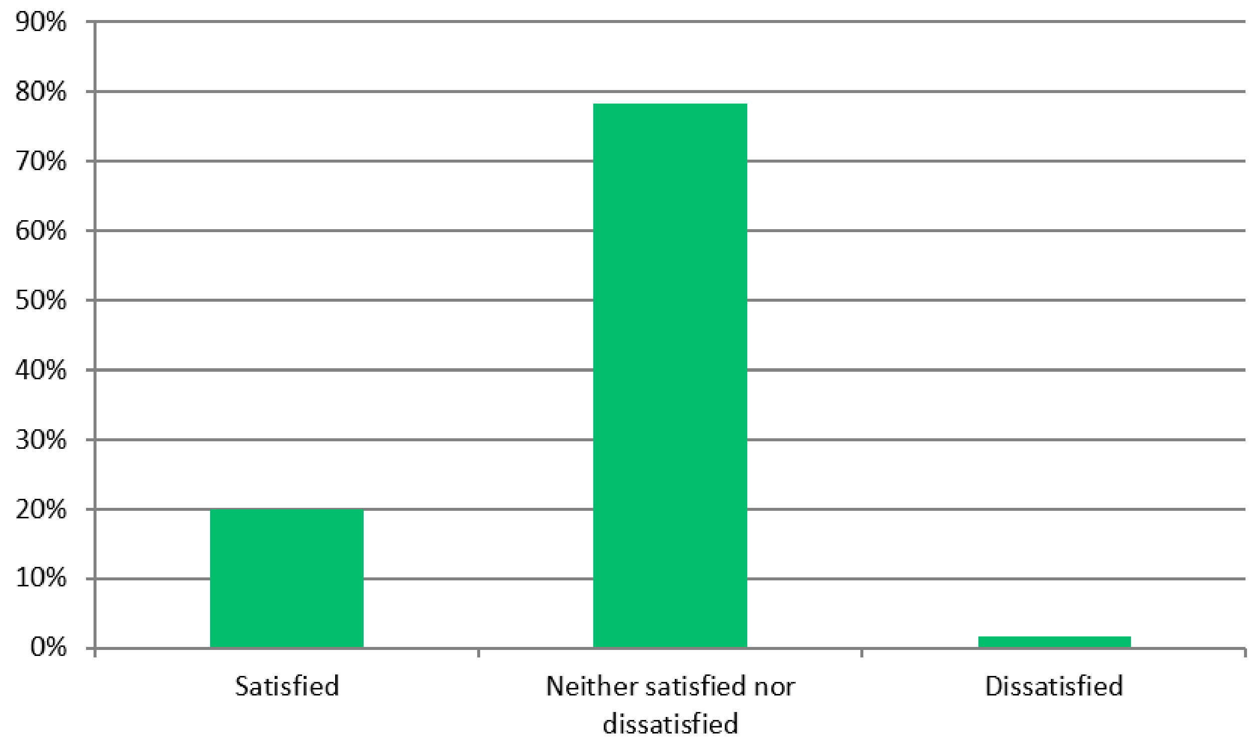
Do you use GFXReconstruct?



Which version of GFXReconstruct do you use? (check all that apply)



How satisfied are you with the reliability and quality of GFXReconstruct?



Comments (Especially if you are dissatisfied):

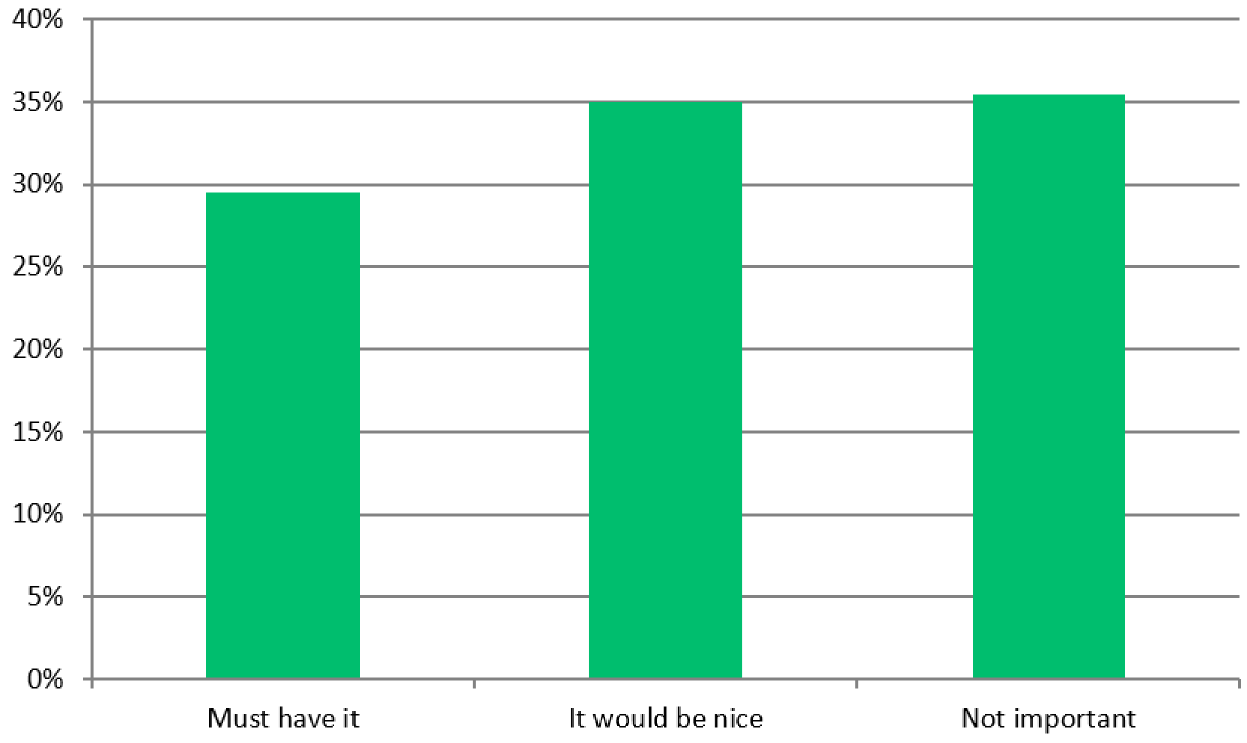
1. it's a great tool but portability between different WSI-backends and gpu-vendors should improve. also there are many options already for a replay and can become tricky to get it right
2. I used it rarely but it was helpful
3. slowly improving
4. Do not use
5. Want to extract buffer and image contents after draws or dispatchés like in Renderdoc for CI purposes

LunarG comment: GFXReconstruct ease of use could certainly improve. GFXReconstruct is actively being worked on as it is becoming the underlying engine for some important profiling and debugging tools and this is continuing to drive quality and feature completeness.

What improvements or enhancements would you like to have added to GFXReconstruct (open ended)?

1. more things should 'just work' with less options
2. The lack of a UI for this tool(s) has dissuaded me from using it, especially because it's not immediately clear what it provides over, for example, RenderDoc or Nsight Graphics.
3. I plan to use it in the future, but never got around to it.
4. Renderdoc like capture API
5. Log call parametrs before executing the actual call so that we can see them when diagnosing driver crashes
6. I think RayTracing/RayQuery needs polishing. Hard to say.
7. Seems reliable. No complains.
8. ability to begin capture after the application has been started. not sure if that's not already a thing.
9. N/A, I don't use GFXR

How important is a Vulkan to Metal translation layer (e.g. MoltenVK) for you?



Open-Ended Feedback

In the survey, there were additional opportunities to provide open-ended feedback via the following questions:

1. If you have used the VK_LAYER_LUNARG_crash_diagnostic layer, please provide any feedback you may have (open ended)
2. What prevents you from being effective and productive while doing your Vulkan development? (open ended)
3. Where would you like to see more investment in the Vulkan developer tools space?
4. Is there anything else you would like to share?

All of this open-ended feedback has been consolidated and grouped into the sections below. Comments were not removed, even if repeated or saying the same thing a different way.

Vulkan on Apple Platforms

1. The current situation with MoltenVK doing their own releases on their github page is confusing. I'm using those for macos but feel like the lunarg sdk releases should be the one and only way to get the sdk.
    a. LunarG comment: The releases done on the MoltenVK repository are the releases pulled into the Vulkan SDK. Users can choose to use the macOS SDK (and get those MoltenVK releases), or to use MoltenVK directly from the repository.
2. "My christmas wishlist: - Ray queries (or even tracing) on macOS"
3. much better support for MoltenVK (1.3 is not available yet)
4. MoltenVK is a constant bottleneck always lagging behind what's available on Windows or even Linux. For that reason i have to make application design decisions that i dont want to make but have to because i require macOS support.
5. I'm worried about the pace of development of MoltenVK, it still doesn't support version 1.3 and I'm worried that it will take much longer to support ray tracing. To the point that I'm considering dropping Vulkan in favour of Metal, because iOS and iPadOS is a very important platform for me.
6. Having to use the Xcode frame profile, since I have to export my VSCode CMake project as an Xcode one, configure it there and then debug it. This repeats when I made a change in the original project…
    a. But that is Apple's fault. I am very thankful for you guys providing MoltenVK.
7. MoltenVK missing extension support for advanced features. Limited Rust SPIR-V library support
8. Support of 1.3 in MoltenVK and ray tracing extensions
9. MoltenVK ray tracing is needed for truly portable modern VK code.
10. MoltenVK ray tracing
11. Please try to maintain and put more into MoltenVK
12. MoltenVK missing ray tracing support is a huge problem.
13. Need more work on MoltenVK
14. Fuller 1.3 support in MoltenVK. Ya'll have done an amazing job here already, and I understand that resources and funding are tight as it is. Also having people comfortable with both Metal and Vulkan I'm sure are not easy to find.

15. MoltenVK needs more investment

LunarG comment: It is known that the current Vulkan on Metal solution (MoltenVK) hasn't been able to keep up with Vulkan as it progresses. For example, its support has not moved beyond Vulkan 1.2 and it hasn't become a conformant implementation. LunarG is investigating ways to help improve this situation.

Crash Diagnostic Layer

1. The layer is easy to use, to configure and provides useful feedback in many cases.
2. I really like that there is such a tool!
3. Useful but not yet the same level of information as e.g. Aftermath
4. I generally haven't found it to be useful. Crashes seem to be somewhere deeper in driver code that isn't picked up.
5. just recently became aware of it, will use it to narrow down future crashes
6. I like it, altough in some cases I did not get much info in the dump (might be a configuration issue or a missing extension on my nvidia 4xxx gpu).
7. Diagnostics are somewhat inconsistent
8. It was functionally unusable the one time I tried it. My SDK is a few versions out of date, so maybe that has since changed.
    a. LunarG comment: For the 3 previous comments, earlier versions of the layer had these issues. They have since been fixed. Try it again!
9. that seems to be newer; we use nvidia only aftermath:-(
10. It caused issues on my driver. Haven't investigated yet.
11. In my case most of the time if I screw something up I end up killing the driver entriely (nvidia) so i dont even try to use this one as it wouldnt catch anything i guess
12. I have not had a device lost since the layer came out

Inhibitors to being productive or effective during Vulkan App Development

1. My Display Miniport Driver isn't developed enough to support a Vulkan client.
2. nothing really or not Vulkan related :)
3. Too many possible configurations to cover all devices
4. Nothing!
5. init a big graphic pipeline， and init lot of struct
6. There's a lot of stuff that needs to be repeated in different ways in different places. An obvious example is pipeline layouts: once in API calls, once more in shader source, yet again in a material system. This is understandable, but since all of the reflection options are limited, it's difficult to just automate the problem away.
7. Drivers on anything but windows.
8. That Vulkan devices can suddenly be lost unreproducibly (VK_ERROR_DEVICE_LOST) even without a programming error on my side.

   a. LunarG comment: Try using the Crash Diagnostic Layer (included in the Vulkan SDK or found at https://github.com/LunarG/CrashDiagnosticLayer

9. AMD Vulkan Compute is a huge problem for us... we currently turn it off on Linux and on Integrated GPUs... we're still trying to figure out how to handle the issues... we would like to expand our server compute offerings but this is basically blocking lot of work.
10. Invest more in AMD Vulkan Compute
11. AMD Vulkan Compute:
    a. - works generally on Windows
    b. - has trouble on AMD Integrated GPUs on Windows
    c. - has trouble on all AMD GPUs on Linux
12. I keep saying it but our big issue is we are having problems with Vulkan Compute on AMD Linux... I think it's driver problems on their end but it blocks lots of our plans.
13. c/c++ itself
14. Limited resources on some less frequently used extensions
15. Shader Objects doesn't support raytracing + I would be happy if Shader Objects will get more performance optimizations.
16. Has monolithic PSO - Metal/Direct3D12(may be LibGNM).
17. Nvidia/amd differences
18. gpu captures & replay, not smooth, specially when blocked by a single capture opened at a time.
19. XCode.
20. Productivity impacts due to experience/knowledge
    a. Lack of theorethical graphics knowledge
    b. My very limited knowledge
    c. Setting up a new project and synchronization
    d. Bugs in my code, mostly. Right now I have a bug where every other frame, one of my buffers (bound through BDA) is just... wrong. My atomic add adds to the wrong place in the buffer. Existing tools like RenderDoc and GPU-Assisted Validation haven't helped, running my code on Android just makes it crash on startup, I don't know where to go from here to debug it
    e. Not getting enough time to work on it..

Shader Compiler, Languages and related Tools

1. slang
    a. Ship with slang, and get perfect glsl to slang mappings as well as more information for developers switching.
    b. Invest more in slang
    c. I would like GLSL updates, or a stronger commitment to GLSL developers with slang. Currently its perfect for HLSL, but subpar for GLSL, which is the opposite of how a Khronos API should be. I get that most large code bases target HLSL,

but being a second class citizen because you didn't historically focus on Microsoft technologies is unfortunate.

   d. "Slang. It's an amazing language but there are compilation bugs, weird-looking SPIR-V generation, and all sorts of problems with reflection (ranging from an incomprehensible API that takes far too long to get to first ~triangle~ ""generated a pipeline layout by reflection"", to an inability to query assigned variable defaults, to all sorts of sharp edges when reflecting user attributes).
   e. GLSL is dated and slang adoption is still annoying.
   f. GLSL doesn't support rich annotations, and Slang's annotations are in theory good, but in practice this part of the Slang project is still pretty limited and buggy.
   g. Need to invest more in shader compilers (e.g. Slang)
   h. I'm the principal developer on SDL's new GPU API. Overall I'm really pleased with the state of Vulkan tooling right now. My only thought is that slang needs a clear path for laying out descriptor bindings. Our shader system requires a particular descriptor set layout and developers who try to use slang with SDL GPU run into this problem. I don't personally use slang but it's a frequent complaint that I see.
   i. I really dislike Slang and fact that HIP/Rocm and Opencl c++ don't compile to vulkan SpIR-V.
   j. It would help to have slangc be listed alongside dxc in the find Vulkan cmake for easier build system integration
   k. Someone really should make Rust bindings to Slang.

2. GLSL
   a. Also GLSL has early support for all the new extensions but the language does not support useful modern features
   b. limitations of GLSL, poor integration of shading languages in the host language (e.g. for structs shared between host & device, need to write the definition twice, for shaders and for the host app)
   c. I need single header shader compilation libraries for runtime so glsl shader code could be loaded directly without needing to recompile them before outside the application
3. DXC/HLSL
   a. DXC bugs and quirks.
   b. I need dxccompiler.so for Android, I still can't compile it.
   c. improve HLSL to spirv translation
   d. I need robust DXC support and bug fixes
   e. More investment needed here:
      i. Clang-HLSL, SPIR-V function calls and pointers
      ii. Clang-HLSL, SpIR-V function calls/pointers, forward progress.
   f. Clang-HLSL not having enough resources
   g. Lackluster development velocity of Clang-HLSL
4. Shading languages being universally terrible.
5. Invest more in SPIRV-cross

6. Uniform shader compiler to compile lot shader language to spir-v
7. No function pointers or goto statements in SPIR-V, and lack of forward progress it guarantees
8. Lack of goto/function pointers in SpIRv, and forward progress it guarantees.
9. multiple shader languages: GLSL / HLSL / Slang
10. Being forced to external tools for compiling shaders.
11. Shader languages (is inhibiting my productivity and effectiveness)
12. SPIR-V cross bugs
13. SPIR-V Cross needs to be rewritten
14. spirv-reflect could have feature parity with spirv-cross as linking spirv-cross just for reflection is a tad excessive
15. Add direct and simple support for compiling shaders from source to the API.
16. the lack of single-source programming environments similar to CUDA.
17. Runtime shader compilation and linking to a library that's required for it. (CMake + FetchContent/CPM). Maybe I just haven't found a simple way to do that yet. Examples?
18. SPIR-V debugging is very cumbersome. https://www.khronos.org/spirv/visualizer/ does not to that demand justice. It would be nice to have more aids, such as basic block grouping and better explanations about various referenced inputs, types, and structs.
19. Invest more in Shader languages
20. Need better shader languages
21. Some more investment on spirv tools like spirv-link would be nice
22. I'd like to see a shading language that integrates better with the host-side language (e.g. reuse struct definitions between shader & host code)
23. what ever glsl or hlsl etc. all the shading language can transform to spir-v use uniform compiler.

Tooling

1. Vulkan development tools have been great and helped solve many issues!
2. Debugging tools are not as useful/robust as dx12 equivalents
3. RenderDoc
    a. Renderdoc not working with newest extensions.
    b. Being unable to use RenderDoc
    c. Validation and debugging tool support for descriptor buffers, especially RenderDoc.
    d. I think last year, I would have said tools like RenderDoc and NSight not catching up, but I think that's rapidly changing.
    e. RenderDoc also needs to support RT, or we need to embrace a better alternative more equipt to handle modern VK code.
    f. Improving RenderDoc (it's not a part of Kronos, but it should be!) to catch up with the latest additions to Vulkan.

- g. Invest more in RenderDoc
- h. I am mostly fine with what is available. Maybe GPU debugging? But RenderDoc gets the job done for the most part.

4. Profilers
    - a. Lack of good cross-vendor profiling tools
    - b. Profilers like RGP and other performancettools (RGA) that are easier to set up and more robust
5. Device Lost
    - a. Diagnosing device lost or fence timeouts.
6. Shader debugging
    - a. lack of good shader/device emulation/debugging
    - b. Better shader debugging tools
    - c. I hope there will be improvements to the shader debugging environment. especially for compute shaders.
    - d. lack of shader/register level profiling tools for older hardware
7. Bindings
    - a. Official Rust bindings for Khronos side projects like KTX would be great.
    - b. More support for Rust bindings,
    - c. Funding/support for https://github.com/ash-rs/ash
    - d. The Vulkano team needs some love.
    - e. modern C++ support
    - f. Official support for safer languages like Rust
    - g. Need Official Rust bindings.
    - h. Also, when creating tools, I think it's a mistake to provide these tools as c++ libraries with no c interfaces. Vulkan is used in a lot of languages, not just C. Having to interop with C++ is extremely annoying, and could be avoided if helper libraries simply were made in plain ol C, or at least provided an option. As it stands we end up having to remake certain things like vkbootstrap in different langauges, even though those can provide value to people generally.
8. GPU debugging (LunarG comment: To do good cross-GPU debuggers, the IHVs would need to expose GPU characteristics to the Vulkan API to enable such tools. This won't happen due to the proprietary nature of IHV GPUs. So GPU specific debuggers will be unique to each IHV).
    - a. Lack of an actual GPU execution debugger like we have on the CPU. Nvidia recently released a beta debugger in this vein for Nsight Graphics, but naturally it only supports their GPUs.
    - b. I want GPU analyzers for linux!
    - c. GPU analyzers for linux
    - d. lack of GPU debuggers
    - e. Invest in GPU analyzers for linux
    - f. Invest more in Debugging tools (the existing ones are awesome, but there's a gap for a real execution debugger).

    g. There should be a single vendor independent debugging tool that should feature all features to debug all kinds of stuff: shader execution, rays, models, PCI and memory usage, basically mix of RenderDoc, NSight and RDP

9. Vulkan Introspection Layer
    a. I found the Vulkan introspection Layer (https://github.com/nyorain/vil) a really convenient tool for real-time debugging.
    b. There is this Vulkan Introspection Layer (https://github.com/nyorain/vil) that is Very effective, as it provides real time debugging capabilities, and it also supports ray tracing (visualizing the acceleration structures). But its not that active in development and it has a few instability issues. Still its very good, Could Khronos help out in some way to develop this tool?
    c. Invest more in Vulkan Introspection Layer maintanance, (and potentially integration into vkconfig) (https://github.com/nyorain/vil)
10. Video and compute debugging tools with no graphics pipeline
11. Debugging applications that use advanced features (descriptor indexing, buffer references and ray tracing) is very hard through the lack of good cross platform tools
12. Most time consuming issue is finding lost resources. When was a resource create, destroyed and handle reused. Could be automated in a layer....
13. Doing compute shader debugging, it can be tedious at times thought debugPrintf helps
14. tooling quality and out-of-dateness on fedora linux
15. Tools and libraries that are largely unrelated to Vulkan but are necessary such as Visual Studio which is frequently being frustrating or figuring out libraries to do model loading
16. Inabillity to emulate different devices and OSes
17. Official (or recommended) tools that provide an abstraction over memory allocation, descriptor allocation, synchronization and device bootstrap.
18. Better debuggers that can help when my code is technically spec-compliant but still buggy. I'd like an emphasis on multi-frame debugging, since a lot of modern rendering techniques use information from previous frames to inform the current frame. RenderDoc's UI could also use a bit of work, I constantly feel like I have to hunt through three or four separate tabs to get an idea of what my renderer is doing. Maybe a more visual representation of the data flow in my frame would help
19. Software emulation of extensions
20. would be nice to have a testing framework for lib developers where you can say ""apidump between these 2 points"" and then compare against a predefined dump of all the functions, structs and their values, extra points for running under mockicd so you don't need drivers or a device for a CI environment
21. more investment in the gpu captures, gpu captures for compute (using renderdoc), never fun with vulkan as far as i've seen
22. Areas where more investment would be useful:
    a. Invest more in game engines like Godot
    b. Invest more in device emulation
    c. profiling tools, examples, good practice examples

d. a cross-vendor/platform frame-debugger (like Renderdoc) or capture-tool with raytracing support and ability to visually debug raytring workloads, visualize Acceleration-structures (like e.g. nsight)
e. Raytrace debugging (nsight sortof covers this on nvidia gpus. Also its partly ongoing in renderdoc, as it now can capture with raytracing enabled which is a huge step)
f. Graphic debuggers.
g. Single-source C++ programming environment similar to CUDA where I don't have to manage the boundary between CPU and GPU (in the process, eliminating the need for a shading language).
h. It would also be awesome if Sascha's device registry (which is already indispensable) supported richer queries and maybe even cross-referenced itself with the Steam hardware survey or something like that so that it would be easier to answer ""what amount of compatibility am I giving up if I require feature X"" type questions.
i. Debug tooling
j. I'm primarily working with headless (no surface) rendering and calculations, profiling tools are difficult, limited or even impossible to use in such scenario.
k. I am missing a tool for showing the content of buffers and images in GPU memory during application debugging. I believe it could be implemented as a layer. That is the reason I asked one of my students at my university to try to make a prototype. He is implementing for me a Vulkan Debugging Tool (https://github.com/Vulkan-FIT/#vulkan-debugging-tool). He is expected to work one more year on it, but the prototype already tells me that it is possible. The idea is to step through the code in Microsoft Visual C++ and see in real time the content of buffers and images of the application being debugged. Feel free to contact me about this tool on peciva@fit.vut.cz .
l. Conformence testing & Validation layers
    i. Both are good, but improvements here would benefit all."

Samples

1. More code samples
2. Synchronization is very hard to understand. Fully working examples with complex cases should be available.
3. update examples to vulkan-hpp.

Tutorial

1. I think it would be useful if the Vulkan Tutorial just used vkconfig for validation layers instead of enabling them in code.

    a. Comment from the Khronos Vulkan Tutorial team: Adding this as an option to the programmatic layer setup is feasible. It will be investigated. See https://github.com/KhronosGroup/Vulkan-Tutorial/issues/60
2. "Vulkan Tutorial" using vkconfig instead of enabling layers in code.
3. More tutorials on intermediate features. Mesh shaders, the different methods for descriptors, etc
4. If you want to learn the basics, the Vulkan Tutorial is great. However, if you want to learn newer or advanced concepts, it's not always easy to find good materials. The specification is huge and hard to read; it'd be nice to have some more guides, tutorials, or materials that make learning easier.
5. Good official tutorials and guides for Modern Vulkan Features. The spec is nice, but a written guide is a lot nicer introduction
6. I think the most important thing is to make Vulkan more accessible. Vulkan can be a bit overwhelming even after completing the tutorial, and it'd be nice to have some more tutorials in a centralized way to get a bit more used to the API. Not only showing how I can make something work, like in a sample, but why should I use it this way. I'd love to see some more informal guides, like how I should do certain things, to make them performant. Videos can be good, and there are some great ones, but if I need something specific, I can't really search if it contains that thing, so I either watch the full 1-hour video or search for another material. But on the positive side, I can see that there are a lot of videos, and there is an effort to make it more accessible.
7. I like the tutorials on youtube. I also took a tutorial on Udemy, by Galea. This stuff is really hard to learn by myself.
8. More tutorials for beginners would have been nice.

Documentation

1. I need documentation and how the many extensions relate
2. Fuller documentation and more beginner friendly tutorials/samples
3. Vulkan is love! In general the documentation and best practices should reflect the latest version
4. I very need spirv-reflect doc.
5. spirv-reflect online doc like docs.vulkan.org
6. I feel Vulkan is a very important project for the gfx community, but I fear big industry players will try to interfere with the viability of it all, just as they did with OpenGL - making the future uncertain for short, and in turn with consequences on the adoption of Vulkan by programmers.
    a. Therefore, thorough, open, user-friendly documentation (whatever the mileage of readers), could well be a powerful safe guard for the future of the API&toolchain. I'm saying this because I've noticed, in almost ten years of Vulkan, the ""official"" documentation effort (beyond just specs) only sprung up relatively recently, it was a bit of a (pleasant) surprise, but also worrying to a certain degree."

7. like there to be more GLSL documentation that's specific to Vulkan. A lot of GLSL documentation is specific to OpenGL and there are things that apply to Vulkan and things that don't, which makes it more confusing. Sascha Willems' examples pertain more to the Vulkan API itself but not the GLSL side of things so much - even though there are some GLSL shaders. There's no GLSL-for-Vulkan documentation is what I think I'm trying to say. :]
8. I feel I lack User guides as part of the Vulkan documentation, per-Vulkan version, as things apparently changed substantially between 1.0, 1.1, ... up to 1.4. Such guides should point clearly into best practices for a given version, and the overall direction/roadmap for Vulkan by Khronos (a bird's eye view). Same goes for extensions and stuff moved to core: I'd need a clear map of things.
9. Not enough documentation is available which explains the vkspec, similar to the notes in the vkspec.
10. not enough documentation
11. The red book is terrible. Beyond Vulkan Tutorial, good reading material is quite rare
12. Good extensive and easy documentation, preferably in tutorial format that is not just specification like but provides a story and takes you to a result through big picture and a complete story/context.
13. Navigating the documentation could be easier. The new documentation website is a good improvement though.
14. Specification load speed
    a. The Vulkan Documentation should be much easier to navigate - it shouldn't take a while to load and be one big huge document. It should be individual separate pages like the registry.
    b. Slow loading specs website
    c. Full online specification loading time is extremely slow. It's not an issue if you need to open the web page just once and keep it opened in your browser, but when you frequently jump to the full spec with the hash to a section through the hyperlinks from other pages, it's a painful experience.
        i. LunarG comment: If you were linking to the full online specification from a validation layer error message, the validation layer error messages now link to an Antora site version of the Vulkan specification and load quickly. This became available with SDK version 1.4.304.1
15. Lack of a central "best practices" store for Vulkan techniques. I am constantly second-guessing myself when implementing basic features like async image uploading.
16. Quality online documentation, the current system is hard to navigate
17. Lack of optimisation recommandation inside the docs
18. Low amounts of fleshed out documentation in comparison to OpenGL (previous toolchain) such as the vulkan docs being very barebones and most developers having to rely on the registry or third-party tutorials/codebases

19. I'm a professional gfx programmer in the Games industry (I work on d3d12), I know I can handle the learning curve for Vulkan fairly well (just starting down that road, for personal projects mostly) - but I fear getting bogged down in outdated tutorials, books and such."

Vulkan API and Specification

1. Happy with Vulkan
    a. Vulkan is pretty tight, yo.
    b. Thank you all for making and maintaining Vulkan. It's the best of both GL and D3D without all the garbage, and I'm elated that it exists.
    c. I am incredibly more productive than when I wrote OpenGL. The first steps are very hard, because of the API's explicitness inducing a lot of verbosity. After becoming more familiar with the API (and introducing abstraction layers in the framework I'm using), the workflow becomes both smooth and mechanical (syntactically similar across functionalities).
    d. Thank you for your hard work!!!
    e. "Thank you for Vulkan!
    f. We will improve our 3D graphics using Vulkan API
    g. i love vulkan, khronos, opengl,..i love you :)
    h. Vulkan forever, OpenGL must die as soon as possible"
    i. I'm super new, but I like vulkan so far.
    j. I can't really pinpoint something, as I have been quite comfortable with Vulkan lately. I think the roster of features that have been added to the core in 1.2 and 1.3 improved Vulkan heavily.
2. Not happy with Vulkan
    a. Vulkan is exactly the same as OpenGL, except with no sensible defaults and fewer features.
    b. Vulkan is garbage and is killing your organization
    c. Khronos prevents me from being productive and effective with my application development
    d. All Vulkan succeeds in doing is ensuring industry regulatory capture from larger developers by adding needless complexity to programming GPUs.
    e. The sooner it is abandoned in favor of separate, simpler, APIs for tiled and desktop class GPUs, the sooner we can get back to actually being productive again."
    f. Abandon the API, there is no fixing it.
    g. It's too late - I've moved to DX12
    h. Release OpenGL 5 and drop the failed Vulkan API.
    i. vulkan is annoying to use.
    j. Improve OpenGL instead.
    k. that's why im currently using OpenGL

3. Verbosity, complexity, and bloat
    a. Verbosity of the API (esp when just trying to explore new extensions/features),
    b. The verbosity & complexity of the API, and runtime pitfalls such as implicit layers.
    c. Legacy bloat in the Vulkan spec
    d. The lack of a standardized mid-level API is a major issue to independent developers who are not dedicated entirely to graphics programming. Vulkan is proving too complex for production use without dedicated engineers for it.
    e. I'dDespite having used most versions OpenGL and DirectX and currently have a reasonable Vulkan render back end, I think Vulkan is still too complex, unnecessarily complex for most developers. An option for automatic synchronization and simplified handling of things like SwapChains could help, even if optimal performance is sacrificed. I understand why one size cannot fit all. With more responsibilities than ever, the graphics programmers job never ends now.
    f. it was difficult to begin with, steep learning curve, but after a while-becomes challenging in a good way, i believe i am productive enough
    g. Lack of experience and know-how. Thinking about how to abstract things is getting easier with more experience.
    h. Image layout transitions are a bummer. For now I'm just transitioning images to IMAGE_LAYOUT_GENERAL for now, and when mobile becomes a higher priority I'll start thinking more about it.
    i. It is a little hard that there are multiple active ways to do similar things.
        i. - descriptor pools / buffers
        ii. - render pass / dynamic rendering
        iii. - pipelines / shader objects
        iv. - timeline / binary semaphores
    j. - storage buffers as function arguments
    k. - templates
    l. I hope vulkan can become more beginner friendly, without cutting back on the amount of boilerplate. I think the amount of granularity in setting up a renderer is great, but sometimes can go unexplained or left to a black box. I hope the docs can be filled out, registry made more friendly to beginners, maybe suggesting fixes or common mistakes made in either the validation layers or registry links, and make the roadmap more aggressive with its feature set.
    m. It is difficult to understand the correct usage of VkSubpassDependency. It's more difficult to understand depending on the situation like MSAA or subpass(input attachment).
    n. it is hard to deduce best practices, many examples for boilerplate but not many for practical uses, modern techniques allow simplified development but they are hard to find.
    o. I'm a beginner so my biggest problem is figuring out the correct way to do things. Knowing about things like synchronization2, the various maintenance extensions,

etc is a bit harder when leaning it all at once. You have to learn the old way, the new way and the reasons for the change..

p. The amount of pNext structures making programming and navigating through the spec hard and unclear. The vulkan API is piling up extension after extension and this problem is becoming worse and worse.

4. Feature requests
    a. Standardization of the VK_NV_ray_tracing_motion_blur extension, so it can be used on non-nvidia hardware
    b. We need multi-level bvh for ray tracing and get more feedback from bvh hits (e.g. diagnostics about how often parts of bvh are hit so we could stream in/out bvh data for massive scenes)
    c. I want Present timing on Windows and Direct Storage extension
    d. Invest more on unifying raytracing specification of vulkan.
    e. I am missing two things in Vulkan API:
        i. (1) - Indices are just 32-bit. It means they cannot be used for the indexing all the vertices in GPU memory. If they would be represented by 64-bit value, it could be much more general solution allowing for using it as pointer to the whole GPU memory. This would provide quite flexibility for CAD applications handling huge models with many parts scattered through the whole gpu memory. This is related to another problem: If indices are scattered through the whole gpu memory, I cannot submit batch of indirect command structs by a single vkCmdDrawIndexedIndirect call, because I have to update index buffer binding before processing of each indirect command struct. I would like to drop the idea of binding index buffer and to have pointer (or VkDeviceAddress) to the first index stored in the indirect command struct. This way, indices might be stored anywhere in gpu memory and I could easily make single vkCmdDrawIndexedIndirect call for one hundred of thousands indirect command structs as opposite of one hundred of thousands pairs of draw calls interleaved with vkCmdBindIndexBuffer calls. Even recording it is slow.
        ii. (2) - It would be so cool if I would not need to bind descriptors, but could access them directly by pointer. Something like the buffer_reference used in GLSL can do for direct memory access (without binding of buffers).
    f. Please make descriptor buffer extension core and focus on forcing vendors to develop drivers for older gpus(for example intel doesnt support vulkan on some not so old mobile gpus on windows)
    g. I would be happy if descriptor Buffer had promoted to API, please and more dynamism, but take into account and first place performance. More low-level control and less high-level abstractions. Developers are happy when they have more control how they managing bytes in memory :)

- h. Timeline semaphores not being able to extend to swapchain synchronization is really a bummer. It means you cannot universally adopt them, which means an extra place for things to go wrong.
- i. Possibly a common abstraction for "work graphs", like a cross-vendor version of VK_AMDX_shader_enqueue

5. Modernization/Deprecation
   - a. Yes. Stop releasing 100s of extensions every single year. I work with Vulkan almost daily and even I can't keep up.
   - b. Also, start deprecating stuff sooner rather than later. Implement the deprecated craft in some layer for backwards compatibility and clean the API."
   - c. I am strongly in favour of a Vulkan 2.0 release that removes all legacy parts of the Vulkan API
   - d. Vulkan is piling up on new features, that are replacing old patterns, and despite its good seeing the api getting more modern, at the same time it is making things more confusing. As an example see descriptor buffers, push descriptors and normal descriptor sets. Official guides still reference different patterns, and is unclear which ones are actually recommended nowadays. I think a Vulkan 2025 guide is needed, illustrating the modern patterns which are recommended to be uses, assuming modern desktop hardware.
   - e. Make Vulkan simpler, deprecate and remove, lots of controls are unnecessary and or could be simplified while leaving the more advanced door open
6. Overall with 1.2 extensions and onward the developer experience has vastly improved. Adding more options to dynamic rendering is always helpful but extending what can be done on the GPU with more functions would be nice (for example providing a list of scissors and changing scissor state with a single draw indirect call would be a nice addition)
7. Direct3D12, Metal has no compatibility for older API, and has success.
8. Timeline semaphore support for swapchains
9. 1. Bad API design
   - a. - Legacy from OpenGL VkFrameBuffer. Direct3D12/Metal has no this abstraction at all.
   - b. - VkPipelineLayout Direct3D12 has no this abstraction at all.
   - c. - GLSL this Is poor choice for Vulkan. HLSL should be first language for Vulkan 1.0 and later, instead of legacy GLSL
   - d. - numeric binding for Vertex Atrributes instead of Semantic names like as in Direct3D12. We can create extension
   - e. VK_EXT_HLSL_semantic, it similar for Direct3D12 D3D12_INPUT_ELEMENT_DESC:
   - f. typedef struct VkVertexInputElementDescEXT {
   - g.   const char*   semanticName;
   - h.   uint32_t   semanticIndex;
   - i.   VkFormat   format;

j. uint32_t inputSlot;
k. uint32_t alignedByteOffset;
l. VkVertexInputRate inputSlotClass;
m. uint32_t instanceDataStepRate;
n. } VkVertexInputElementDesc;
o.
p. // VK_STRUCTURE_TYPE_INPUT_LAYOUT_DESC_EXT
q. typedef struct VkInputLayoutDescEXT {
r. VkStructureType sType;
s. const void* pNext;
t. const VkVertexInputElementDesc* pInputElementDescs;
u. uint32_t numElements;
v. };

10. 2. Vulkan features
    a. - Where Tile Shading for Vulkan on Mobile Platform ? See Metal API
    b. - Where Mesh Shaders for Mobile platform ? see Apple A15 for iPhone 13 and later
    c. - When we will have Work Graph Shaders KHR, core ? https://github.com/KhronosGroup/Vulkan-Docs/blob/main/proposals/VK_AMDX_shader_enqueue.adoc
11. 3 .4 Unnecessary extensions for legacy OpenGL compatibility
    a. VK_EXT_shader_objects - Again OpenGL legacy compatibility ? The modern successful Graphics API
12. VK_EXT_provoking_vertex Again OpenGL legacy compatibility ?
13. I'm still very new to it so mainly just learning the API. In particular, specifics about changing of approaches to solving problems from OpenGL to Vulkan that are more useful than a basic hello triangle. For example, migration from multi draw indirect calls to an equivalent Vulkan solution would be welcome.
14. safer APIs

Android

1. Feature fragmentation across Android devices, especially our need for supporting years old devices without driver updates
2. Driver bugs on some mobile devices
3. More debug tools on mobile platforms
4. Vulkan on Android
    a. Bad Driver support, Developing of OpenGL ES/Vulkan Drivers spend a lot of time for developers, we need to remove OpenGL ES’s native legacy drivers from Android by replacing Angle via Vulkan.

Miscellaneous

1. I’m a little unsure how the arch Linux package works but I assume it uses the tarball or will quickly change so it should be fine
2. Update arch packages to 1.4
3. live debugging (i.e. not frame capture but debugging the live program, like nsight shader debugger, rocgdb)
4. It would be nice if the OpenXR runtime and the VVL could (optionally, of course) ~conspire~ cooperate to mute the absurd amount of validation spam some of the XR runtimes generate. (I would say "make the XR vendors make their runtimes validation-clean, but I don't believe in fairy tales.)
5. Improve the Loader-ICD interface. Vulkan's ability to support extensions is nice, but the D3D UMD is so much easier to implement.
6. better vulkan wayland support
7. More and more extensive/wider integration with various exiting libre/opensource graphics tools and programming tools
8. Our current workaround is to have the server only GPU accelerate on Nvida, which defeats the purpose of Vulkan Compute. It's primary advantage is cross hardware, if you wanted Nvidia compute only why not use CUDA.
9. Continue the good work
10. Cute anime girl representation, a Vulkan-tan
11. more cowbell
12. I found the numbers of Vulkan Video talks disproportionately high at this year's Vulkanised
13. Better support for older Linux distributions (with older GLIBC), e.g. we have to bundle an old glslangvalidator with our app to compile shaders on the fly.
14. "As mentionned, I'm just starting my Vulkan journey, although I do work professionnally with d3d12, and I'm also an OpenGL old timer.
15. The questtion about the important platform for the fututre was missing Linux/RISC-V, I see this one becoming very important in the future.
16. "All I want is OpenCL 1.2 with triangle rasterisation and bytecode kernels.