LunarG 2026 Ecosystem Survey Report

March 2026

Executive Summary

Methodology:

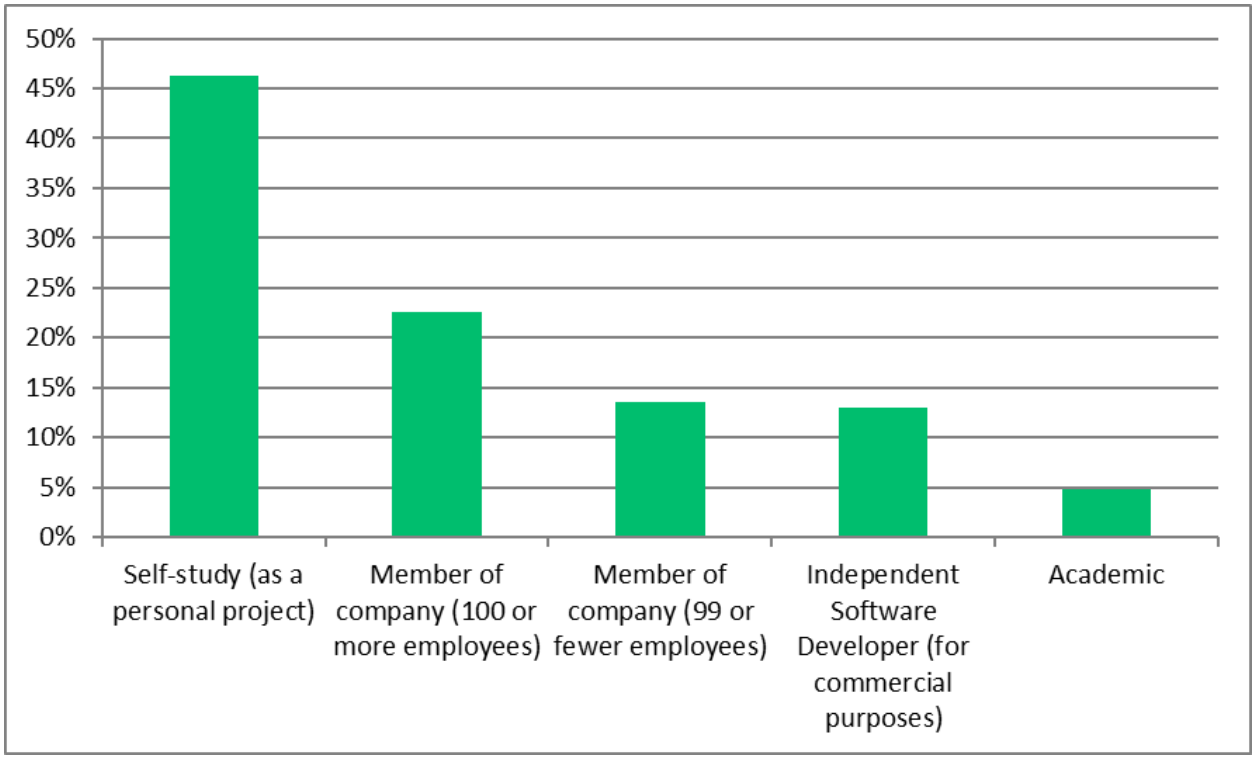
- The purpose of the ecosystem survey is to gauge the Vulkan community's use of and satisfaction with the current Vulkan ecosystem.
- We attempted to reach as many Vulkan developers as possible -- both SDK users and non-SDK users. The survey was advertised on X, Reddit, LinkedIn, the Khronos Vulkan slack channel, Vulkan Discord, and sent directly to 14,000+ 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.
- Editorial comments from LunarG or others in the Vulkan working group are in blue.
- All graphs are shown for the two populations: self-study survey respondents, and commercial developer respondents.
    - All open-ended comments from commercial developers are in red.

Key Takeaways:

1. There were 360 respondents, **almost 30% more than last year.**
2. 49% of the respondents use Vulkan for commercial purposes. 51% of the respondents were self-studying Vulkan as part of a personal project or an academic environment (non-commercial).
3. 71% of the respondents are regular, advanced, or expert Vulkan developers. Hence the feedback is coming from a more experienced population.
4. Still strong demand for improved Vulkan documentation, tutorials, and samples.
5. Continued concern about complexity of the Vulkan API.
6. Concerns about API fragmentation (consistency across platforms), platform stability, and reliability.
    a. LunarG comment: This feedback is being reviewed by the Vulkan WG. The best method to get the Vulkan WG attention and action on this feedback is to submit an issue to Vulkan-docs (https://github.com/KhronosGroup/Vulkan-Docs)
7. slang continues to grow in popularity as a shading language, however GLSL is still the top preferred shading language:
    a. In 2025, 27% of the population favored slang.

    b. In 2026, 42% of the population favored slang.
8. The slang compiler as a tool of choice for generating SPIR-V continues to grow. However the population of users using glslang or glslc is still the largest population at 74%.
    a. In 2025, 24% of the population used the slang compiler.
    b. In 2026, 40% of the population is using the slang compiler.
9. 60% of survey respondents expressed the need for a Vulkan to Metal driver on Apple platforms (Must have or nice to have):
    a. 56% of the population is aware of the KosmicKrisp Mesa driver.
    b. 33% of the population will adopt KosmicKrisp or already have done so. 34% are undecided. KosmicKrisp is converging upon having feature parity with MoltenVK. Once performance tuning is completed it is anticipated the adoption rates will grow significantly.
        i. KosmicKrisp only being supported on OS26 or later is not an inhibitor for adoption.
        ii. KosmicKrisp only supporting native ARM64 builds is not an inhibitor for adoption.
10. Validation layer themes:
    a. Due to significantly decreased comments, error messages from the validation layers have improved.
    b. GPU-AV still needs more maturing from LunarG before we broadcast it as ready for everybody to use every day.
    c. Synchronization validation is in pretty good shape. Still needs integration with GPU-AV to detect synchronization errors occurring during GPU execution.
11. SDK themes:
    a. About 34% of the population is not updating their SDK within a month of release. These users are not benefiting from the ever improving coverage and robustness of the Vulkan validation layer.
    b. Include the llvm-pipe software renderer.
    c. Better management of SDKs (deletion of old versions, auto-update, etc).

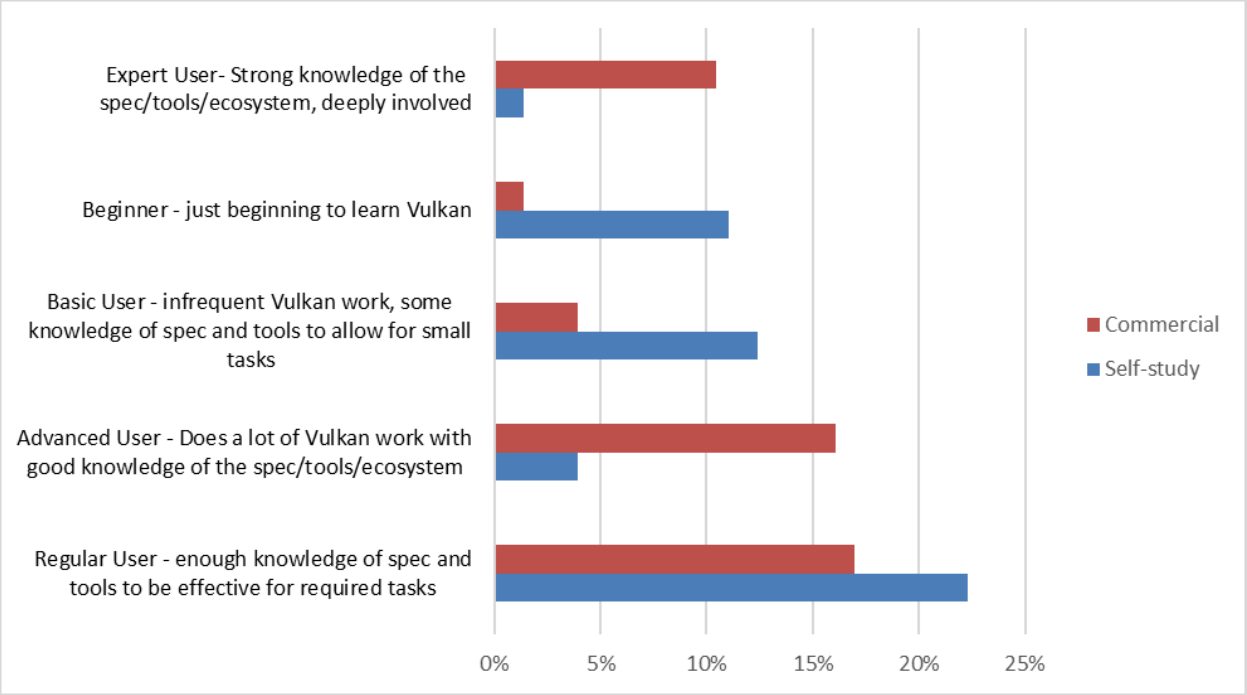
What type of Vulkan Developer are you?



Answered: 355

51% are self-study or academic
49% are using Vulkan for commercial purposes

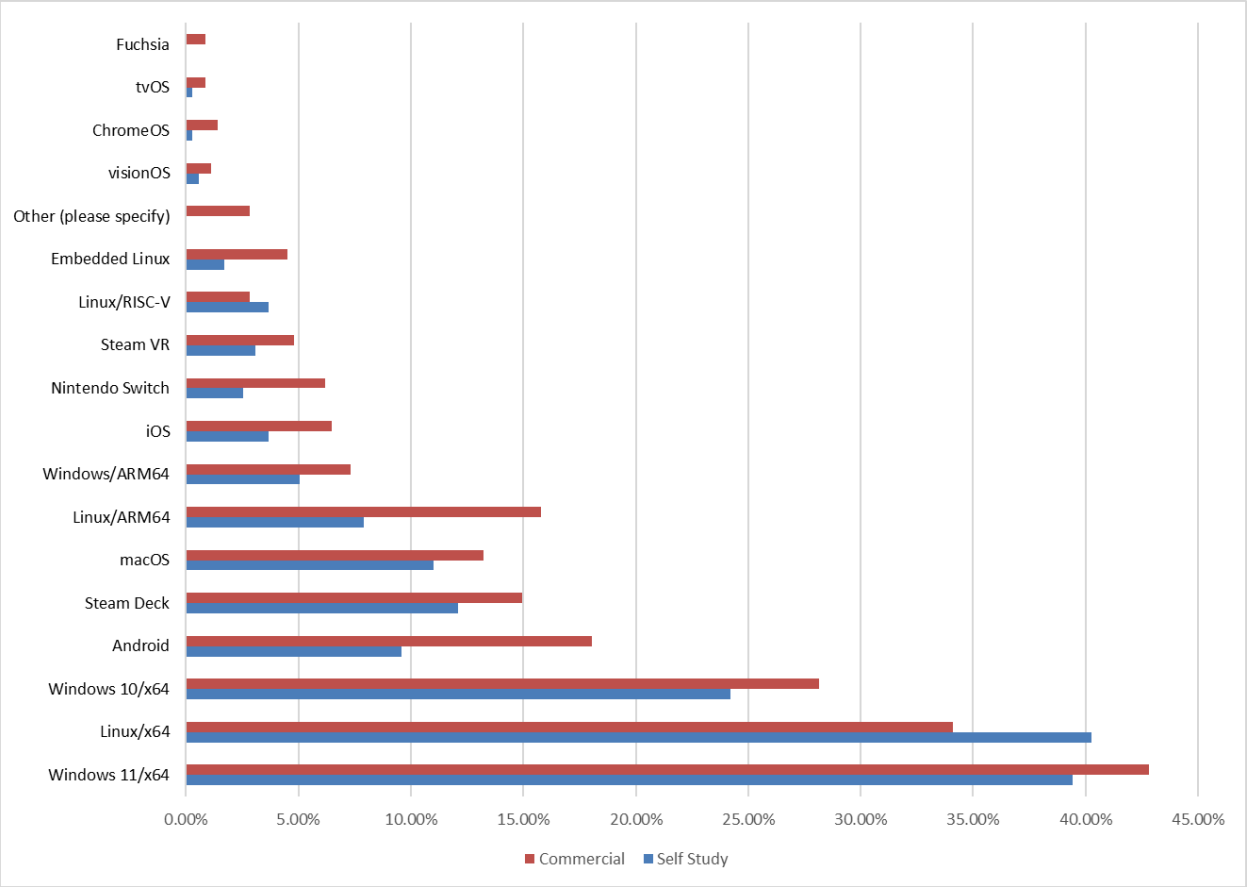
How experienced of a Vulkan Developer are you?



Answered: 354

Of the population overall, 71% 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.

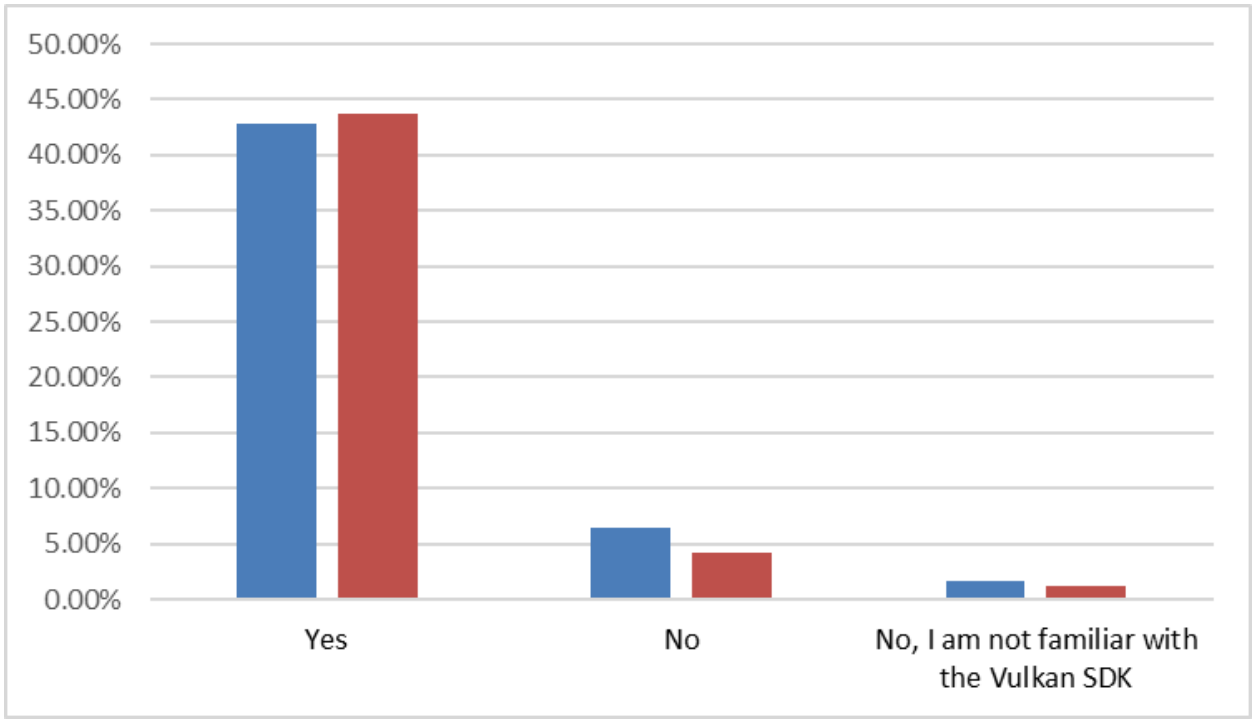
What are the targets of your Vulkan application? (check all that apply)



Answered: 355

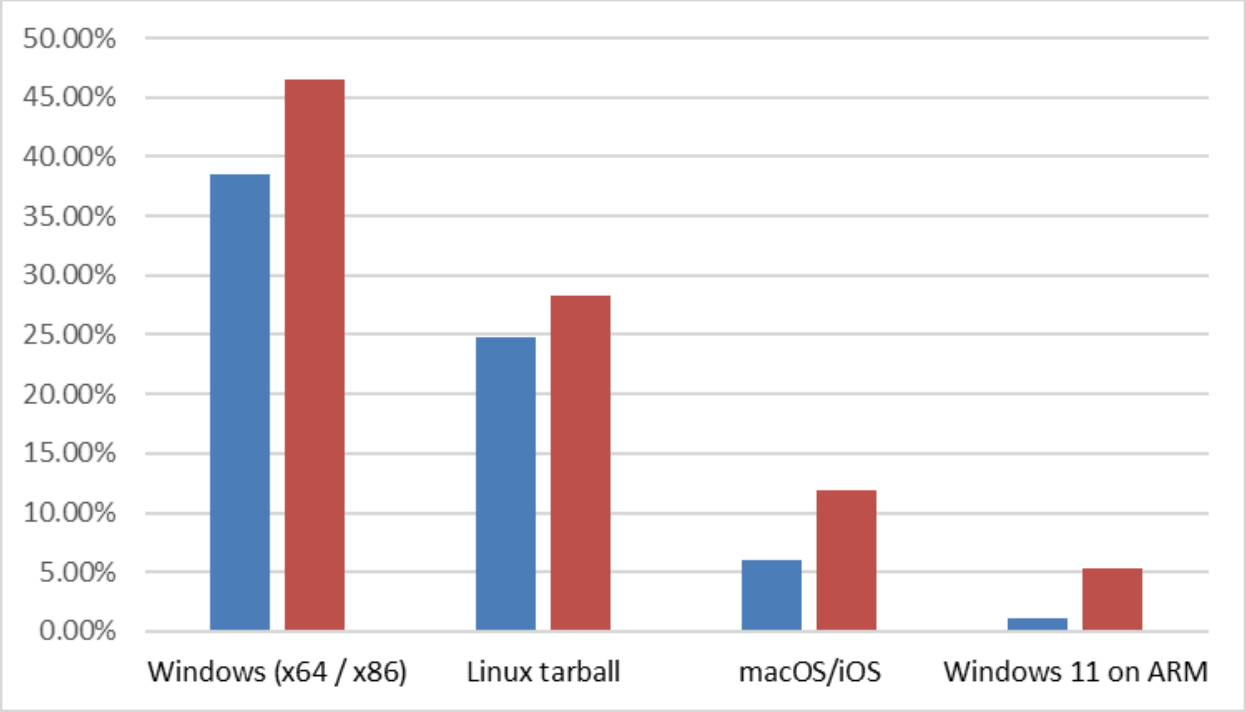
Windows 10 still remains a popular target. This must be due to the fact that businesses can still get support through several "Extended Security Update" (ESU) paths

Do you use the Vulkan SDK?



Answered: 355

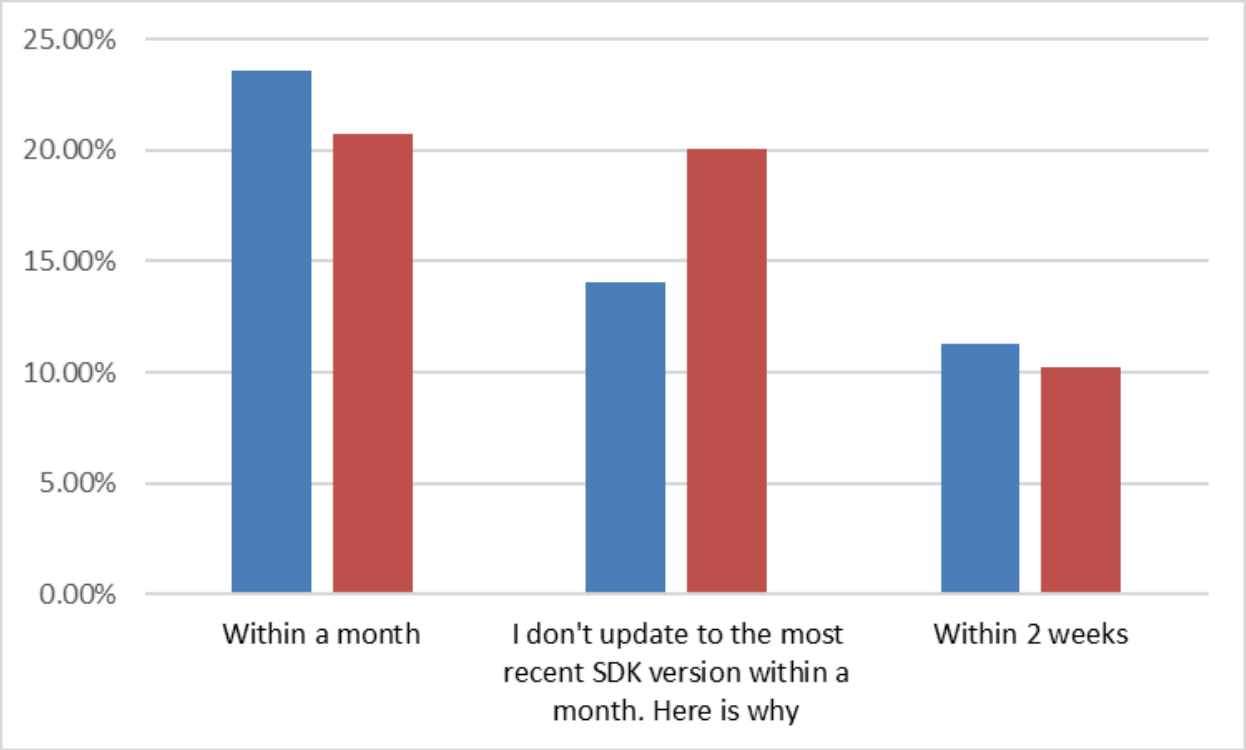
Which of the following Vulkan SDKs do you use? (check all that apply)



Answered: 286

Growth in usage of the tarball is due to Ubuntu packages being discontinued. Last year 34% were using the tarball. This year 53% were using the tarball.
Small growth in use of the Windows on ARM SDK. Last year 4%. This year 6%. Overall, more commercial developers were using the Vulkan SDK compared to self-study developers.

When do you update to the most recent SDK version?



Answered: 284

The purpose of this question was to gauge how many users are benefiting by getting more recent versions of the Vulkan validation layer by updating their SDK on a regular basis. The survey results show that 66% of the population is updating within a month. This is good. However there are still about 34% of people not updating the SDK on a regular basis.

LunarG comment: There are many comments below from users implying they don't believe there is a benefit in updating to a newer version unless there is a new feature or Vulkan API version needed. However the validation layers are continuously improving in bug fixes, and more importantly in validating more of the Vulkan API. If users don't update, the validation of their API usage doesn't improve as well.

LunarG comment: If you don't want to be developing for 1.4, the newer SDK is backwards compatible with previous Vulkan versions. You won't be receiving the validation layer improvements if you stay on an older SDK version.

Why I don't update within a month (commercial developers responses in red):

1. Only update when there are new features, versions, or bug fixes needed version
    a. There's usually not a reason too unless new features are required.
    b. Latest features also can't be used due to hardware support requirements, reducing need to update regularly."

c. I update if there are features I want.
d. I update to the most recent version once there are some new interesting features.
e. going higher then Vulkan 1.3 will cut off customers.
f. I only update whenever I find useful features.
g. depends on changes / fixes (specs / docs, headers, tools, compilers / libs) and what projects I work on (new extensions?), but big releases / updating a couple of times per year is usually good enough
h. No need for the most recent SDK, and we need all the machines to run the exact same version.
i. "LWJGL version used is usually behind VK by a good amount, so newer version isn't strictly needed.
j. I don’t always require the latest features.
k. All depends on if a new extension that I want to use comes out (e.g. EXT_descriptor_heap, EXT_mutable_descriptor, EXT_mesh_shader, etc...)
l. "There rarely are feature updates worth the update.
m. I update when I see a reason to.
n. Currently my project is in development before release, so I update once every 3 months or so. Right now the focus is on feature progress.
o. Only update as needed for new features/fixes.
p. I only update when needed (bug fix, new feature...)
q. My use of the SDK is quite limited and usually run home-built validation layers from source, so I only update the SDK when I need a new version for a repro or someone else's project.
r. When I don't have new features I want to use I usually don't bother since it takes a bit of time to restructure
s. I update maybe once a year unless specific updates affect me.
t. I often don't have the need to. I only do so when something exciting happens that requires an update to work with (new tooling, new extension, ...)
u. We have a dependency update cycle. The frequency of the updates depends on how we can handle the other updates. Since the previous versions do not necessarily bring new things we desperately need each time, and since all versions we use are stable, updating is a low-priority task.
v. I only update once I find features I'd like to use in the changelog
w. I upgrade only when I see a specific purpose for the upgrade, for instance that it solves a particular problem I've had.
x. We just randomly update it these days whenever there's a bugfix we need.
y. Unless there are bugs, upgrading SDKs is considered a low priority thing. And there are too many high priority things going on, so this falls on the way side.
z. We update on a need basis.
aa. I update on an as-needed basis.
bb. I update when I start a new project or notice issues with the old version
cc. when needed

dd. I update when I get a weird error that I can't fix.
ee. when I need to I update

2. I don't need any new features
    a. If it ain't broke, don't fix it.
    b. I update when I expect an explicit benefit, mostly the currently installed version does all I need.
    c. dont usually depend on latest features (waiting for descriptor heap though)
    d. Is generally not needed
    e. No reason to update until there is something I need.
    f. My app is using Vulkan 1.1, no need for latest functionalities. But nice to have the debug tools !"
    g. I'm not always using the newest features or need the latest bugfixes from the Vulkan SDK in order to develop, so I'm able to rely on the stability of older releases and update more infrequently.
    h. The current one works for me and updating isn't a priority
    i. Because it rarely matters
    j. There isn't really any need to for what I'm doing
    k. I see no reason why
    l. Haven’t seen the need yet.
    m. if it works, why would I risk it?
    n. Why update? It could break the tutorial.
    o. I only use the SDK because its the only way to get validation layers, and headers required to compile. Since those don't change very fast, I just don't pay attention to when the SDK gets an update.
    p. I don't really need the most recent features
    q. Usually I don't need the most recent version of the SDK so I update from time to time.
    r. I don't see any features i need right now, despite how nice descriptor heaps sounds like, as they aren't widely supportet yet.
    s. I install the SDK when I need it and isn't installed, or if the Vulkan spec updates (eg. 1.3 -> 1.4)
    t. I usually use Rust bindings generated from the spec, so keeping the SDK up to date isn't that important.
3. Updating can cause some instability and cost
    a. LunarG comment: Updating the SDK will likely cause new validation errors. This is due to the validation layers constantly improving in coverage and bug fixes. Validating your application with the newer validation errors and addressing the issues will improve the quality of your Vulkan application.
    b. Updates are usually prone to breakage.
    c. "Updating has a chance to break existing things which can be time wasted just to get to the same point.

   d. Real product with many software dependencies, updating software happens infrequently
   e. Often new false positives or even crashes in new versions of the validation layers. I'm reducing the overhead of working around those, making github issues or providing fixes by updating once or maybe twice a year.
   f. Maintaining stability of the development environment.
   g. Sometimes there are false positives.
   h. The sdk releases often, I dont have the time.
   i. I usually wait for SDK stability confirmation before upgrading, as validation layers, tooling changes, or subtle SPIR-V behavior differences can affect correctness and benchmarking results."
   j. Small patches might have to be applied to either my source code (vulkan-hpp C++ modules are still experimental and can introduce breaking change) or to the sdk (some CMake files to locate tools such as the slang compiler might not work as expected).
   k. There had been incompatibilities with MoltenVK on macOS. Basically automated tests on a virtual GPU failed because of out of memory errors.
   l. We are an enormous title who operate at scale with full vendorization of our toolchain and requisite QA, etc.
   m. Last time I updated too soon, some bug in the custom memory manager used by the SDK resulted in my program crashing and me wasting too much time trying to debug it. So now I update only when I have time to spend on any potential breakage.
   n. I dislike regular updates. Especially as stuff might randomly break and for compatibility concerns
   o. risk of team down time due to unknown bugs or slight changes in behaviour
   p. SDK updates almost always generate a pile of validation issues that need addressing. That takes time and needs to be scheduled.
   q. Also, one of the macOS updates broke the rendering that resulted in a missed delivery. I know it was it because in a later update the rendering got back to normal without me changing the code."
4. Lazy or I forget
   a. Lazy and don't see any benefits
   b. Because I sometimes forget to check
   c. Lazy
   d. Too much corporate trouble
   e. I forget I can update it
   f. I just forget that it updates and don't see a need unless there's major changes
   g. Don't have time to follow that closely
   h. On Windows, I have to manually go to the website to download and install it. Which is too much effort to do on a regular basis.
   i. I don't know when a new version releases.

   j. Lazy
5. Installation process issues or installation cost
   a. Updating in a build environment (i.e. CI builds) is a pain and not always wanted.
   b. The SDK updating process requires updating cached CMake variables which is kinda annoying, and I'm lazy.t
   c. Working on an internal offline network, so it requires quite a bit of work to update.
   d. Installer takes 30 mins on macos
   e. Updating on my private setup (Arch Linux) is even more of a pain due to manually setting environment variables."
   f. Overhead of tracking releases and updating build systems with the new packages.
   g. The update process is also somewhat lengthy.
   h. If it's not automatic (snap, flathub, etc) then I'll only update as necessary
   i. Updating on Windows is very manual.
   j. Dealing withe tarball is a hassle. I miss the apt install option…
   k. "I stopped chasing updates after the APT repository got deprecated.
6. Other
   a. It seems most mobile are still using 1.3.0
   b. Have not worked with platforms that the SDK supports in the last month.
   c. I'm not the person on my team responsible for updating the SDK, so I'm not sure how often we update.
   d. Very rarely need to update SDK because get the Vulkan headers, volk and tools by other methods"
   e. Still using the old deb packages
   f. i've updated it time by time. sometimes as fast as the new version is uploaded sometimes i didnt update for 4 monthes
   g. I'm not working with vulkan regularly. Sometimes there are multiple months in between touching vulkan related code.
   h. I use SPIR-V tools to eg. compile HLSL to SPIR-V then legalize it and alter it (eg. merge/split textures and samplers programatically) which then gets consumed by a proprietary NVIDIA compiler for Nintendo Switch's proprietary graphics API, so I'm not in a hurry to upgrade often.
   i. "I work on performance-critical and non-graphics Vulkan compute workloads.
   j. We build SPIR-V tools, Shader Compilers and Validation Layers from source
   k. "Every 1 year - 6 months.
   l. "The work I do is more sporadic than once a month.
   m. But for Windows and Linux I do update regularly.
   n. For MacOS we also build MoltenVK from source which does not require an updated SDK as much."
   o. Android support.
   p. "- IT services
   q. - Also use other Vulkan bindings, plus vk.xml etc directly"

r. I mostly use Vulkan via the Rust ash crate that us directly generated from the xml, and the associated tooling doesn't change fast enough to warrant frequent updates.
s. I'm maintaining GTK for macOS, and it doesn't change that often.
t. When I update the SDK depends from the state of my tasks, in general I prefer to avoid to change the SDK version in the middle of a task, I prefer to do it between two tasks.
u. Due to vk.xml registry changes sometimes breaking auto-generated vulkan zig language bindings. I update SDK only when binding generator fixes it. And as these changes are only in nightly build, usually are ~6 months to a year behind.
v. The API is stabilizing, it's not as important to keep up to the minute updates.
w. I installed it once and never updated it on my Windows machine. I installed the development libraries via the standard package manager on my Linux machine and it updates automatically when I update everything else.

What suggestions do you have for the SDK?

Feedback (red text is from commercial developers):

1. Satisfied, works fine
    a. Seems good to me. Updating the SDK has been zero friction. Maybe some people want to consume it through Nuget or vcpkg? I am not one of those people though.
    b. None; so far so good!
    c. I love the precompiled static libraries, they make starting up projects incredibly easy and quick.
    d. Keep up the good work, thanks for all the help!
    e. Quite frankly I'm fully satisfied with the current state of the SDK. I don't think there is much more to bring to it, except of course support for new extensions and validation layers when they appear.
    f. Nothing
    g. So far, it fulfills my needs. I like the idea of having spirv-tools included with latetst VVL and others.
    h. Keep up the good work
    i. it's ok
2. New features/content
    a. More modern optimization examples
    b. Bundling of llvm-pipe
    c. Bundle llvm-pipe
    d. Bundle llvm-pipe
    e. For Windows on ARM and Windows in general, I would appreciate software device (lavapipe) with SDK to be installed.

    i. LunarG comment: Adding llvm-pipe is on our radar for future addition to the SDK.
f. Could add more tools for spvasm e.g. a linter
g. Maybe add KTX libraries to the package?
    i. LunarG comment: Created an internal tracking issue for future consideration.
h. I realize it's a bit out of scope, but I'd love a precompiled version of jolt physics for getting up and running with simulation projects quickly. I do realize that's kinda out of the direct rendering scope though.
i. I need DXC compiler for Android, currently I cant build using CMake.
    i. LunarG comment: The Vulkan SDK is for the desktop and not for Android. You could submit an issue to the DXC project (https://github.com/microsoft/DirectXShaderCompiler) asking for periodic Android releases.
j. The Vulkan SDK is excellent for graphics development, but non-graphics, compute-heavy use cases still face gaps in tooling and validation clarity.
    i. LunarG comment: What gaps in tooling would be needed? Vulkan compute is the Vulkan API and is validated and supported by the developer tools just like graphics pipelines. Unfortunately this individual did not leave an email address for us to probe more.
k. More first-class support for compute-only workflows, including improved validation diagnostics, SPIR-V execution correctness checks, and performance analysis tools that do not assume graphics pipelines, would significantly benefit emerging Vulkan applications in infrastructure and high-performance compute domains.
    i. LunarG comment: There is some shader debugging capability in RenderDoc. Does this not satisfy the request for SPIR-V execution correctness checks? What would be the improved validation diagnostics? Validation validates the API and doesn't validate differently for compute or graphics. Unfortunately this individual did not leave an email address for us to probe more.
l. Please add linux/arm
m. Provide ARM64 binaries in Linux tarball
    i. LunarG comment: Thank you. This request is gaining traction. We removed the Ubuntu packages in 2025 due to workloads increasing and having a fixed budget. The current solution is for the user to download the x86_64 tarball and build it for ARM using the included vulkansdk script.
n. Bring back the ubuntu packages!
o. pls no tarballs :cry:
p. Work with the rust ash project to provide up-to-date rust bindings
q. Probably, as Im using clspv as compiler, might be nice to add other external shader compilers beyong glsl/slangc.

   r. Provide a template library so people can have a default behavior on the functions instead of nothing
3. Size
   a. "Reduce its size
   b. Also, too much stuff i dont need, i just want updated headers and validation layers, not all these other unnecessary libraries."
   c. Keep it leaner, it's a bit of a heavy install.
   d. Keep them separate to other third party libraries. Make the SDK lighter.
   e. LunarG comment: The above comments don't specify which SDK (Windows, Linux, or mac). On Windows and Mac third party libraries and many other SDK components are already optional packages. On Linux, everything is in the tarball.
4. Quality, stability, reliability
   a. "I ask again, only put production ready stuff in there. (Remove beta quality KK)
      i. LunarG comment: This is currently an optional package and can be de-selected.
   b. Better testing before release.
   c. I would suggest fewer, more tested releases, with more clear version distinctions.
5. Auto update, SDK version management, installation
   a. Some kind of manager to deal with all the SDKs installed, it's a bit cumbersome to have to also go out and get the SDK every single time to update, which is why i also do not update that frequently. It would be nice to have some application that can auto update the sdk, and manage the installs.
   b. Is there a auto-update, e.g. a bit like Java's SDK on Windows?
   c. "Thorough replacement & cleanup of older SDK versions.
   d. I only just learned of the vkconfig update mechanism - I would have been asking for exactly that otherwise."
   e. maybe a "check for new update" and "update" button in maintenance tool
      i. LunarG comment: The Vulkan Configurator checks for a newer Vulkan SDK and will install it for you if desired.
   f. Easier update.
   g. Auto-delete previous version.
   h. Release newer versions faster
   i. Publish to a repository (snap,etc) to encourage frequent updates
   j. snap support?
   k. I'd be happy to have some kind of Linux installer. it's not hard to do myself. but it's just a little extra work
   l. I would like a minimal zip file, or tar ball, for all platforms, that contains only what is needed to compile a Vulkan application. This would be for easier packaging of Vulkan as a dependency.
      i. LunarG comment: The ability to do this already exists on Windows and macOS. In the getting started guide for Windows it states "There is an option to only copy the SDK files and not perform any operations to the

registry such as setting up new layers, creating shortcuts, and adjustments to the system path. For the copy only option, append copy_only=1 to the end of the command line installer executable." On macOS, you can omit the “System Global” optional package, or leave it out of the command line installer list of packages to include.

m. Loss of ubuntu packages is disappointing.
n. "As mentioned, the file setup and distribution with the Windows SDK.
o. This may be a skill issue on my side, but the best and most robust way to build an disribute is using vcpkg for dependency managemen (including spirv_cross, shaderc) and load vulkan fptrs directly. I would like to use the vulkan configurator, but it seems my dependency chain breaks
p. I found the installation guide for Linux to be a bit confusing, and just copied everything into /usr/local. I had issues with my first program not finding the validation layers (VK_ERROR_LAYER_NOT_PRESENT when creating the instance), but after troubleshooting, it just worked and hasn't caused issues yet.
q. Linux desktop needs an ARM64 SDK.

6. Usability
    a. Would be nice to be able to target the same SDK on Android, rather than it being delivered with Android SDK/NDK.
        i. LunarG comment: Yes, we hear this frequently. This is a pretty big project to enable this and also is stepping into the boundaries of Google's responsibilities.
    b. I had trouble finding documentation & practice on idiomatically enabling extended validation via VK_EXT_layer_settings."
        i. LunarG comment: The SDK documentation contains the following documentation on how to enable layers using VK_EXT_layer_settings. See https://vulkan.lunarg.com/doc/sdk/latest/windows/layer_configuration.html
    c. "The installer should be able to ""change"" the setup - i.e. install 3rd Party parts after the SDK was installed, via GUI. If you forget something, you'll have to first un- and then reinstall the entire SDK, which is quite annoying.
        i. LunarG comment: Both the Windows and macOS SDK’s come with a maintenance tool that will allow you to modify your installation. It is not necessary to uninstall and then reinstall. The maintenance tool is documented in the Getting Started guide. You can find the maintenancetool.exe in the SDK install directory.
    d. Remove the installer on OSX, let me just untar a folder.
        i. LunarG comment: We intentionally moved away from a DMG/tarball/zip so that we could track downloads of optional components. We continue to need this data and do not intend to go back to a DMG/tarball/zip. Additionally, there is an uninstall.sh script if you installed the “System Global” package, and as no other system changes are made, you can just delete the folder with no side effects.

e. Make SDK easier to install offline.
    i. LunarG comment: The Windows SDK install can be done offline if you ONLY want the core package and none of the optional packages. It is currently not a priority to redesign the SDK install for offline installations of all the packages.
f. You can set up debugger path substitutions for the loader and validation layers, (e.g. using gdb's set substitute-path command in ~/.config/gdb/gdbinit) but this is trickier now the source is not included. The git repositories required and the tags to checkout are listed in the release notes, but the path is usually something like /vulkan-sdk/1.4.328.0/source/Vulkan-ValidationLayers/ and you have to guess what this is for when you have an incomplete stack trace. Perhaps this could be better documented. (To be honest, the main reason I use the SDK is for a more up to date validation layer than my distro provides, and the distro provides both a debug symbol server and separate debug/source packages, so not having the source show up in the debugger is never a problem with their builds.)
    i. LunarG comment: Many years back it was decided NOT to ship source with the SDK. This originally was for size reasons, but also was due to the fact that a library will have dependencies on many other libraries as well. So all of those libraries would also need to be built debuggable. It gets really huge. The SDK purpose is to USE the SDK components, but not to DEBUG into the SDK components. This is why Release libraries with debug info are delivered. Your best solution is to go build the validation layers from the github repository as debug.
g. A symbol server for the loader (and possibly validation layers) would be incredibly helpful on windows. If not a symbol server, at least a way to download every loader binary that has been distributed - either in the SDK or by drivers.
h. Maybe ship sources for libraries included in the SDK so they can be compiled how we want (e.g. with ASan enabled)."
    i. LunarG comment: It is intentional that the SDK does not ship source. It is also intentional that the SDK is not a tool for building all of the tool repositories included in the SDK. If you want the source, the shipped config.json clarifies all of the source you would need to download.
i. Additionally, the Windows version spreads its files in far too many locations. Searching for files and uninstalling the SDK is very difficult. Concentrate files in a few common locations (like AppData), and make sure the uninstaller actually cleans up leftovers.
    i. LunarG comment: The windows SDK comes with a maintenance tool that will allow you to modify your installation. It is not necessary to uninstall and then reinstall. The maintenance tool is documented in the Getting Started guide. You can find the maintenancetool.exe in the SDK install directory.

j. I would also suggest changing the default Vulkan SDK version installation path to NOT be in the root C: system folder (put it beneath Program Files etc. instead)."
    i. LunarG comment: During installation time you can change the default path. It isn't hard coded.
k. Expose more of the libraries shipped with the SDK from find_package(Vulkan) in CMake.
    i. LunarG comment: There is an open LunarXchange issue for this: https://vulkan.lunarg.com/issue/view/6598781b5df1125b58afbb3c We acknowledge the value of this and are hoping our limited resources can get to this in the next year.
l. prepending bin to PATH on windows is a questionable decision and has caused a lot of wasted time
    i. LunarG comment: We have created a tracking issue where the user can select or de-select updating of PATH with the SDK location.
m. Tutorial books. Real step by step instructions with accompanying Youtube video.
n. A windows application to compile glsl to spir_v with a user friendly interface
o. Fix vulkan_profiles.hpp
    i. LunarG comment: After more probing it was determined that the issue to be fixed is https://github.com/KhronosGroup/Vulkan-Profiles/issues/734
p. make easier way to get vulkan sdk version (now I need to parse vulkan_core.h)
q. update the vulkan configurator, and if possible the gpuinfo site to be faster to navigate
    i. Comment from Sascha Willems: An initiative is currently in progress to improve the performance. You can read more about it from Sascha at this Reddit post: https://www.reddit.com/r/vulkan/comments/1rtdo2v/improving_performance_of_the_vulkan_hardware/
r. C++ headers are ugly, it had to be only C headers in the SDK and C++ wrapper as a separate repo.
s. Slow down on the newest visual studio version adoption.
t. Make it as clear as possible what the recommended set of extensions is to target a specific platform (e.g. desktop) with a certain level of hardware backward compatibilty.
    i. LunarG comment: There is a good whitepaper on how to use the Vulkan Profiles toolset here: https://www.lunarg.com/wp-content/uploads/2024/04/The-Vulkan-Profiles-Tools-LunarG-Christophe-Riccio-04-11-2024.pdf

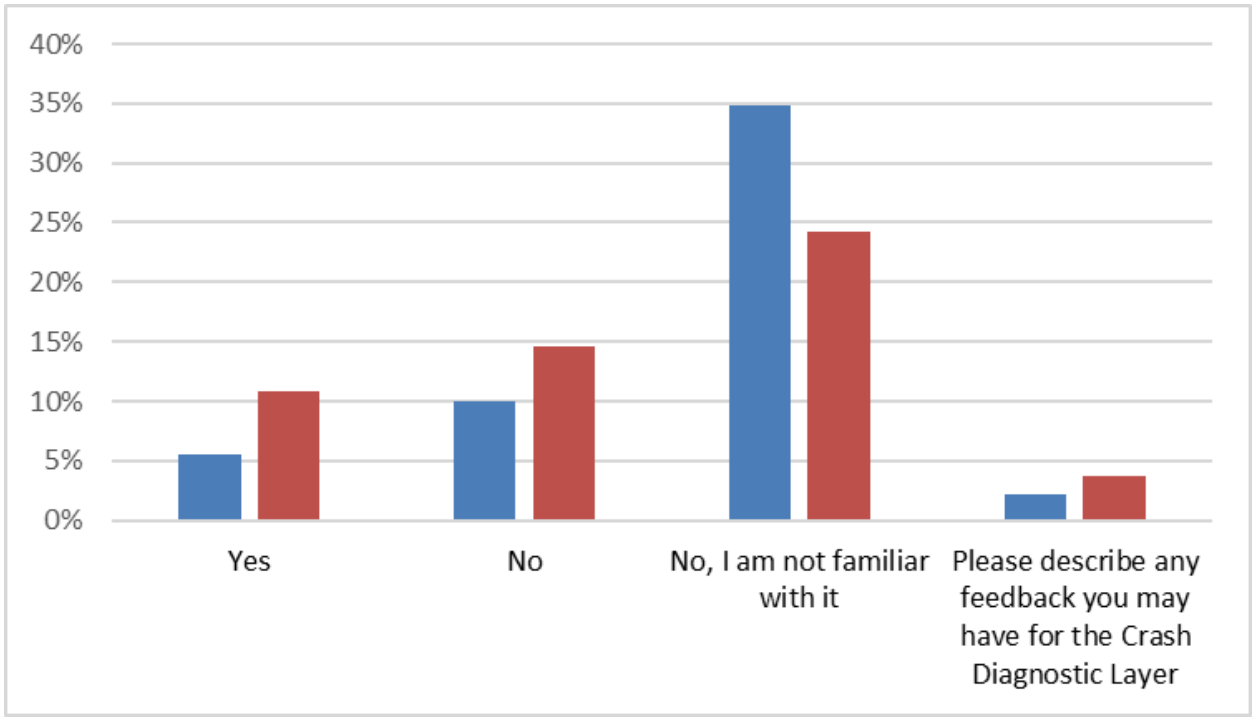
7. Not an SDK issue
    a. Keep adding samples.
        i. LunarG comment: LunarG does NOT include the samples in the SDK and they are managed by the Vulkan WG samples project. This feedback has been given to them.

b. Better handling of setting up pipeline states and synchronization
c. please make a headless swapchain available for integration in WPF and similar use cases.
d. The main documentation PDF has these large sections of validation rules that make reading hard. It should be possible to have a shortened version of the PDF with things stripped that you wouldn't need when learning the API.
    i. LunarG comment: I believe when they say "main documentation PDF", they are referring to the Vulkan specification?
e. Get slang to version its output"
    i. LunarG comment: As of the December 2025 SDK, the slang project has begun versioning their libraries for Linux and macOS. Here is the information in the 1.4.335.0 SDK release notes:
        1. The Slang libraries included in this SDK have received important updates regarding library versioning and ABI stability. Developers integrating Slang into their products should pay close attention to the following distribution requirements
            a. macOS and Linux Versioning: Slang libraries now use versioned filenames (e.g., libslang-compiler.so.0.2025.21). The current major version number is 0, which strictly indicates an unstable public Application Binary Interface (ABI). Downstream tools are linked directly against these fully versioned names.
            b. Windows Versioning: Slang libraries for Windows do not have an explicit version in the filename, but the guidance on ABI stability is the same.
        2. Required Redistribution Guidance:
            a. On all platforms (Windows, macOS, and Linux), downstream users of Slang distributing their products as binaries MUST redistribute the specific Slang libraries they linked against.
            b. It is not the case that a user of your product can just install any recent Slang release and expect it to work with your pre-compiled binary. The dependency on the linked library version is explicit and strict.
f. Better integration of extension function calls
g. Descriptor heap is a very interesting new feature, renderdoc support and driver support would be my two wishes in 2026

8. Unsure what is meant by the feedback
    a. Make it compatible with macOS and iOS and others without glue layers

   b. More web support
   c. shader function pointers
   d. Please make vulkan layer for the web pls pls pls we already have 2 for mac the web is last ! go go go
   e. Easier licenses
   f. Better macOS support
      i. LunarG comment: Unactionable and no email to enable more probing.
   g. Please stable crash diagnostic layers!
      i. LunarG comment: Unactionable and no email to enable more probing. If the crash diagnostic layer is the one in question, please submit a github issue to https://github.com/LunarG/CrashDiagnosticLayer
9. Other
   a. why you won't change it
   b. Maybe recommend updating more strongly.

Do you use the VK_LAYER_LUNARG_crash_diagnostic_layer?



Answered: 322

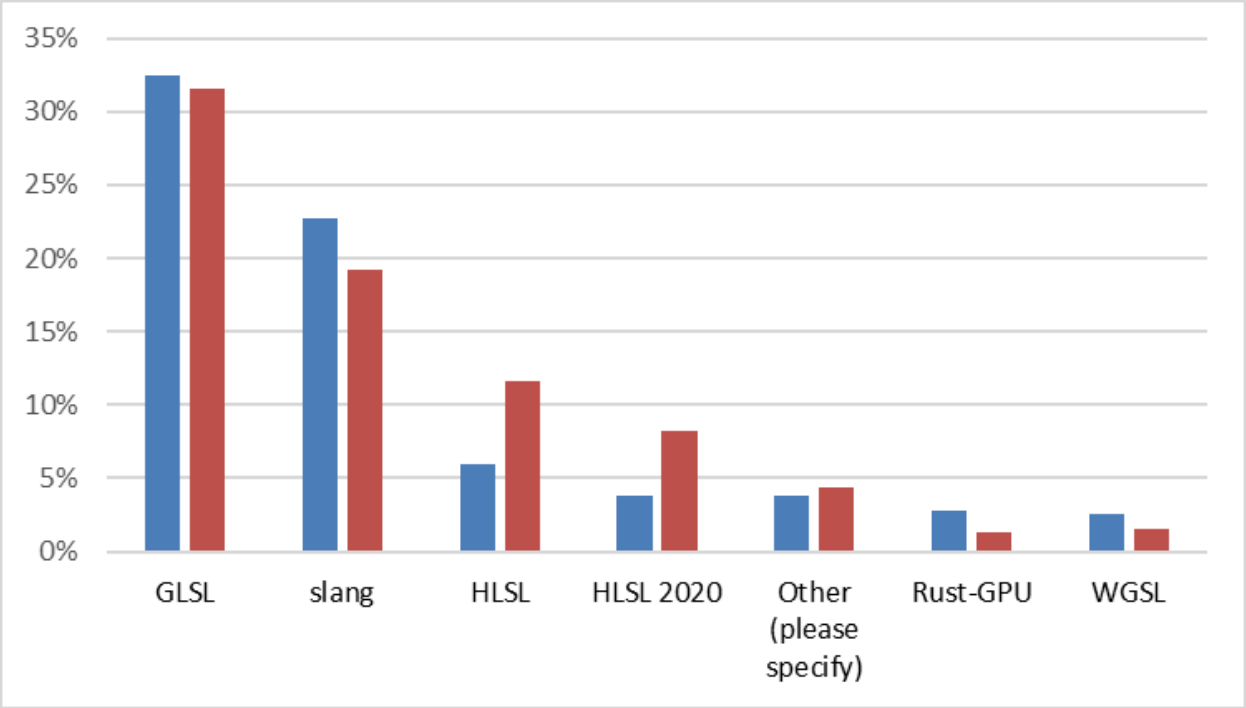
Open ended feedback on the Crash Diagnostic Layer (red text is from commercial developers)

1. That seems useful

2. Hadn't heard of it before.
3. Works well for me.
4. May be I will test this later.
5. It was very useful at the development stage of low-level abstraction layers.
6. I'm not sure if we use that specific layer, but we definitely use the validation layers in general.
7. "If I were to start from scratch, I would absolutely use the crash diagnostics layer. As it stands, we have a custom solution that integrates VK_EXT_device_fault, Nvidia Aftermath (VK_NV_device_diagnostics_config and VK_NV_device_diagnostic_checkpoints) and VK_AMD_buffer_marker for custom annotation of our command stream and crash information recovery.
8. I guess my one wish would be for VK_LAYER_LUNARG_crash_diagnostics to be easily usable on client machines, our solution ships with the game itself and lets us easily diagnose crashes that happen in the field."
9. Might be good to see a blog post talking about the crash diagnostic layer...?
    a. LunarG comment: A presentation was done at the 2025 Vulkanised. The slides can be found here: https://vulkan.org/user/pages/09.events/vulkanised-2025/T40-Jeremy-Gebben-LunarG.pdf The video can be found here: https://www.youtube.com/watch?v=h5Ty-o8_pWE
10. "I primarily work on compute-only Vulkan workloads.
11. Crashes are usually investigated through validation layers, logging, and shader/SPIR-V correctness checks rather than post-mortem crash diagnostics.
12. More compute-focused diagnostics would be valuable."
13. Would be nice for all other vendors to catch up with Nvidia and AMD debug markers or at least the AMD coherent memory extensions so the breadcrumbs would be more reliable.
14. i use nvidia aftermath
15. I have not had the need. But as soon as a situation arrises that other tools cannot help with, then I will use it.
16. When I tried it I didn't help much, or perhaps I was incorrectly interpreting my results. IIRC the issue I was trying to diagnose were pointer dereference crashes inside a shader
    a. LunarG comment: Unfortunately the crash diagnostic layer isn't monitoring for crashes in shaders. This is where an IHV specific tool such as Nvidia's Aftermath is needed.
17. Last time I used it I had to compile it from source which required fixing some Cmake setting, and then find the correct env variables to use and configure it. I would like to have it integrated into vkconfig.
    a. LunarG comment: The crash diagnostic layer IS integrated and selectable from vkconfig.
18. I haven't used it enough to form any feedback.
19. why the document doesnt mention VK_LAYER_LUNARG_crash_diagnostic at all. Only linux document have it. Thats why i have never seen this thing i read window docs : (

    a. LunarG comment: It is clearly documented in the SDK documentation and integrated into vkconfig. Perhaps you are on an old version of the SDK?
20. Just found this out thats great
21. Too confusing, no apparent entry path for the novice
    a. LunarG comment: A presentation was done at the 2025 Vulkanised. The slides can be found here: https://vulkan.org/user/pages/09.events/vulkanised-2025/T40-Jeremy-Gebben-LunarG.pdf The video can be found here: Introduction to the Crash Diagnostic Layer
    b. LunarG is working on a refreshed white paper on the crash diagnostic layer.
22. Don't recall. it's been awhile since I've debugged

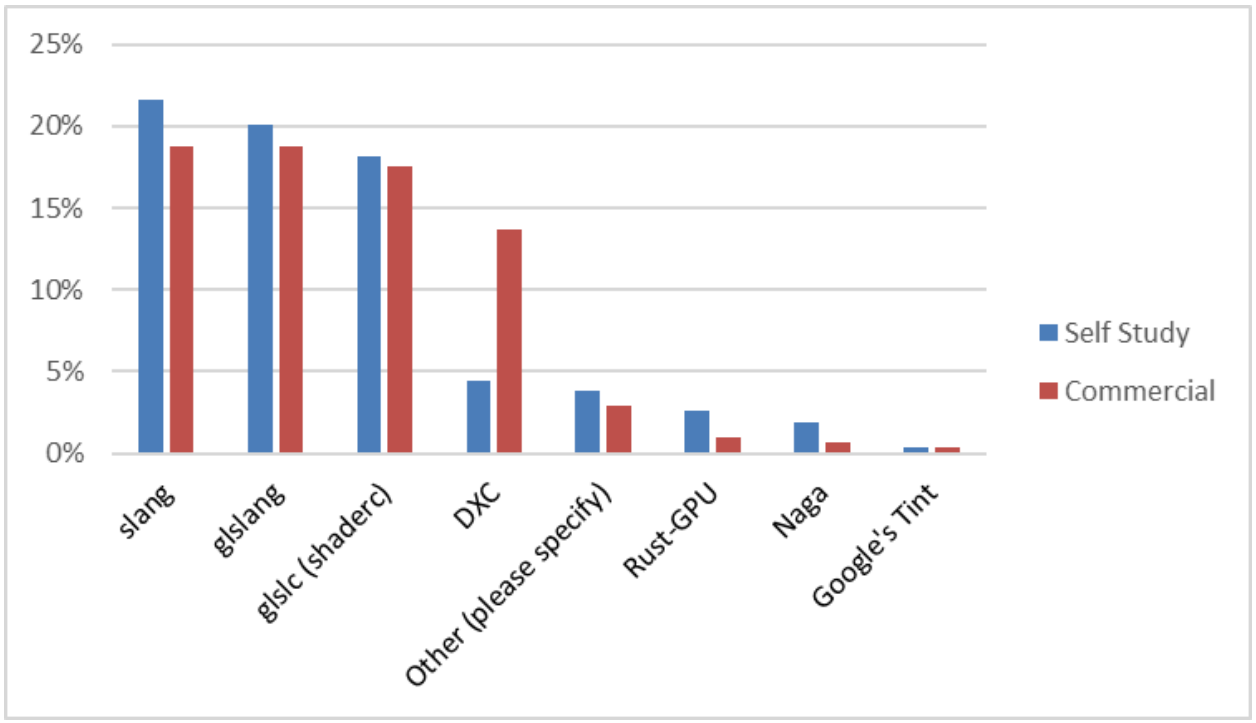
What is your preferred shading language? Check all that apply



Answered: 317

Every year, GLSL is the top preferred shading language.
Slang is growing in popularity. In 2025, it was 27% of the population. In 2026 it is 42% of the population.
HLSL 2020 is an erroneous selection option. I meant to have it be HLSL 2018 (the version that slang supports)

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



Answered: 314

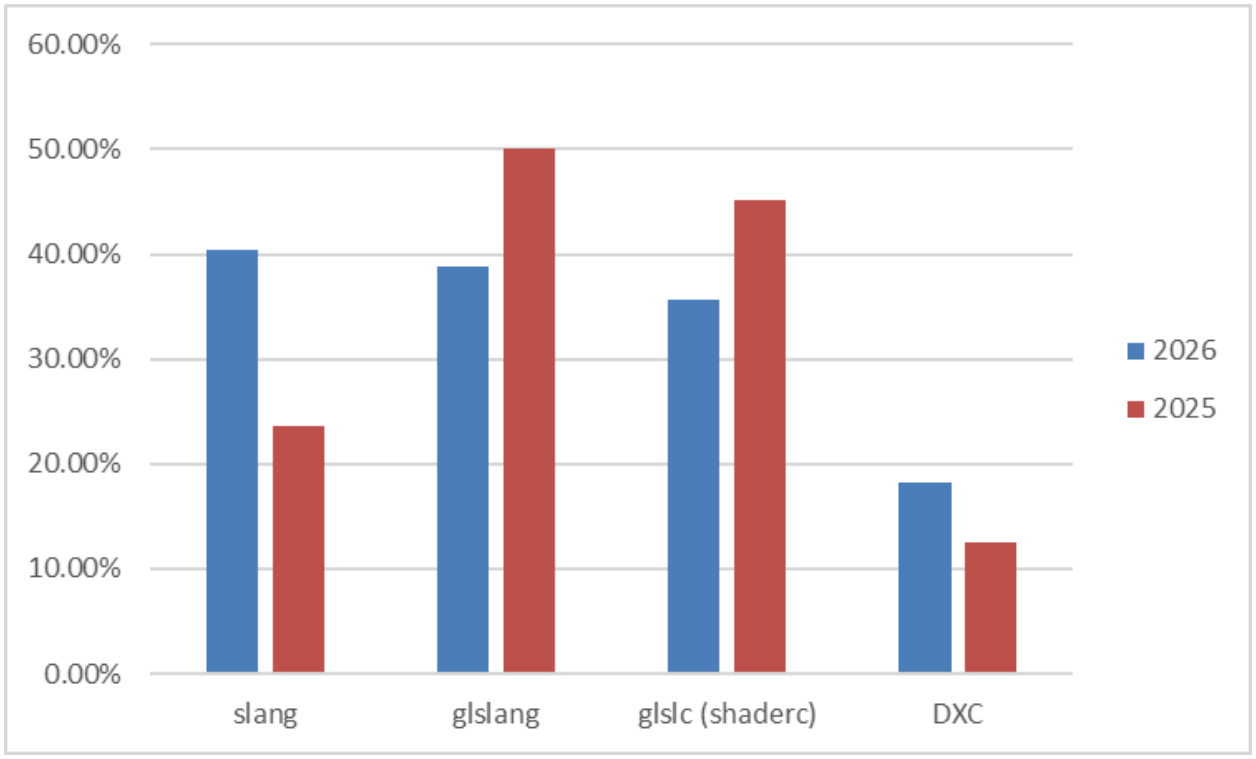
Glslang+glslc is still used by the largest percentage of users with slang strongly growing in popularity.

Other: (red text is commercial developers)

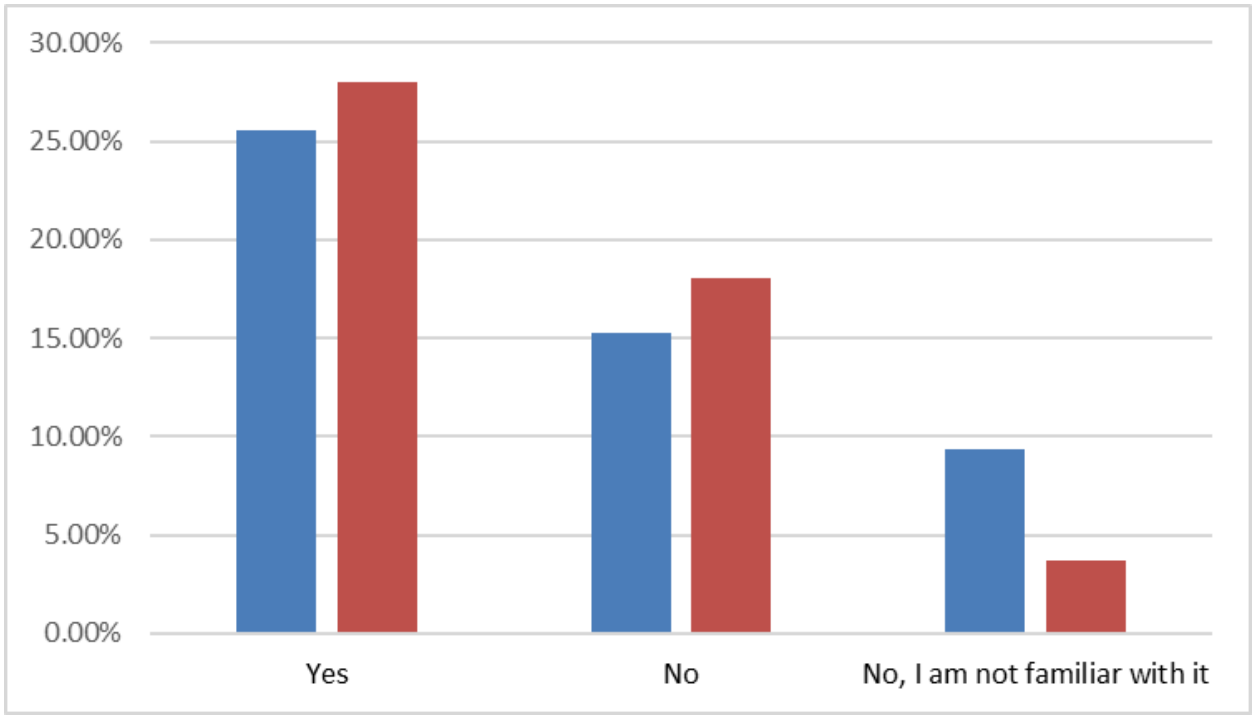
1. I code spvasm manually and then assemble them
2. Our application.
3. DragonJoker's ShaderWriter
4. own glslang/spirvtools/tin/spirvcross wrapper
5. Manual (own assembler)
6. SPIR-V generated for compute-only and non-graphics pipelines
7. Nabla Shader Compiler
8. don't know what SPIR-V is
9. custom compiler
10. zig
11. qfcc
12. Shady
13. clspv
14. local tool

15. NZSLC (Nazara Shading Language compiler)
16. home-made
17. Forgot
18. Zig
19. VCC
20. shadercross (SDL_shadercross)
21. No idea

Looking at 2025 vs. 2026:

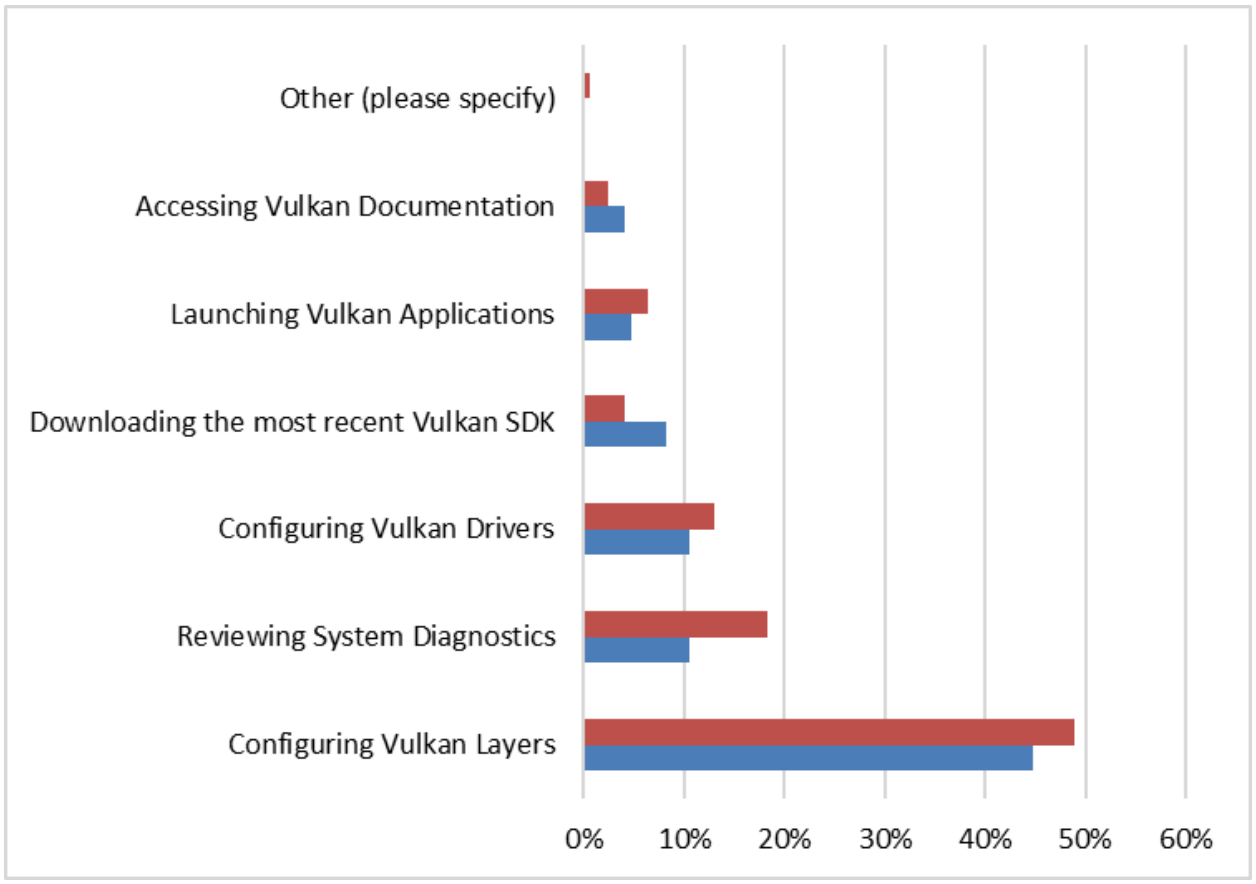


Do you use the Vulkan Configurator (vkconfig)?



Answered: 321

What is your usage of the Vulkan Configurator (check all that apply)?



Answered: 170

Other: (red text is from commercial developers)

1. overriding device selection while testing

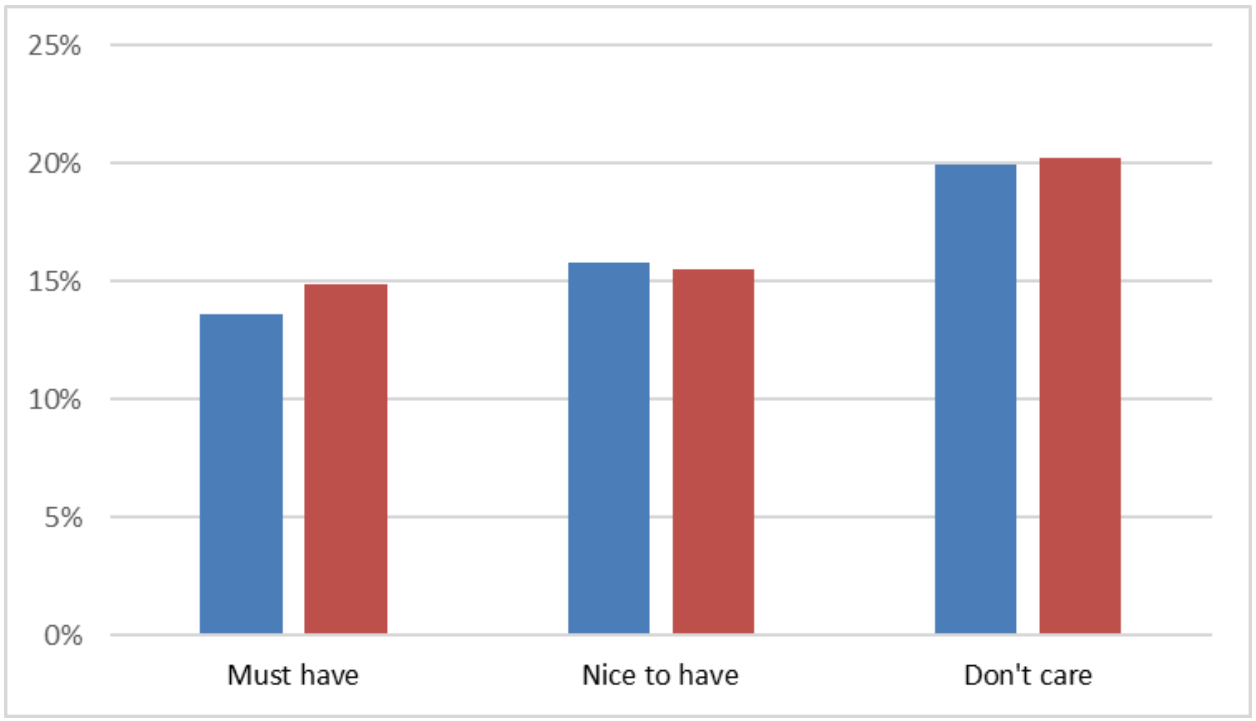
What suggestions do you have for improving the Vulkan Configurator (vkconfig)?

Comments (red text is from commercial developers):

1. Big ol dialog box o tabs haha I love it
2. Add information to what each feature does more detailed.
    a. LunarG comment: The vulkan configurator currently has documentation on each feature that is shown when you hover over its button.
3. "- user-defined layer settings: support multiple, import/export (json)

    a. LunarG comment: Adding user-defined layer presets is something we are considering and tracking with this issue: https://github.com/LunarG/VulkanTools/issues/2331
    b. However, it's already possible to create multiple user-defined layers configurations. This can be done using the context menu of the "layers configurations" list.
4. - application settings: support multiple, import/export (json)"
5. Can you please be clear that in order for Vulka layers to work the VKConfig needs to be running all the time
    a. LunarG comment: There is a setting on the Preferences tab "Keep Vulkan Configurator running in the system tray when closing the main window".
6. I see a lot of people with no idea that sync validation is a separate layers they have to enable separately. would it be plausible to auto-enable it in the standard validation preset? I know y'all's have done a lot to make it faster
    a. LunarG comment: There is a specific validation layer preset for synchronization validation. The default and standard preset DO NOT enable synchronization due to potential performance impacts.
7. I didn't even know you can use it to download new SDK versions. An optional auto-update feature would be neat!
    a. LunarG comment: There is an open issue for this feature: https://github.com/LunarG/VulkanTools/issues/2177 However it isn't currently a high priority due to the effort vs. the value
8. Make the configurator GUI more intuitive to use by relying on conventional idioms of user interface. Currently, I can't tell when things get set, what things are set, etc. since there aren't any "Save", "Okay", "Cancel" buttons, etc.
9. Descriptions for all the layer settings would be good.
    a. LunarG comment: For each of the layer settings, you can right click and a pop-up will be enabled that provides detailed description of the layer setting.
10. I'm pretty happy with it aside from the occasional bug.
11. Hire a UI/UX designer to make the GUI more straight forward.
12. Qt looks bad on my system, I prefer GTK.
13. Dark theme
14. Make some improvements on UI

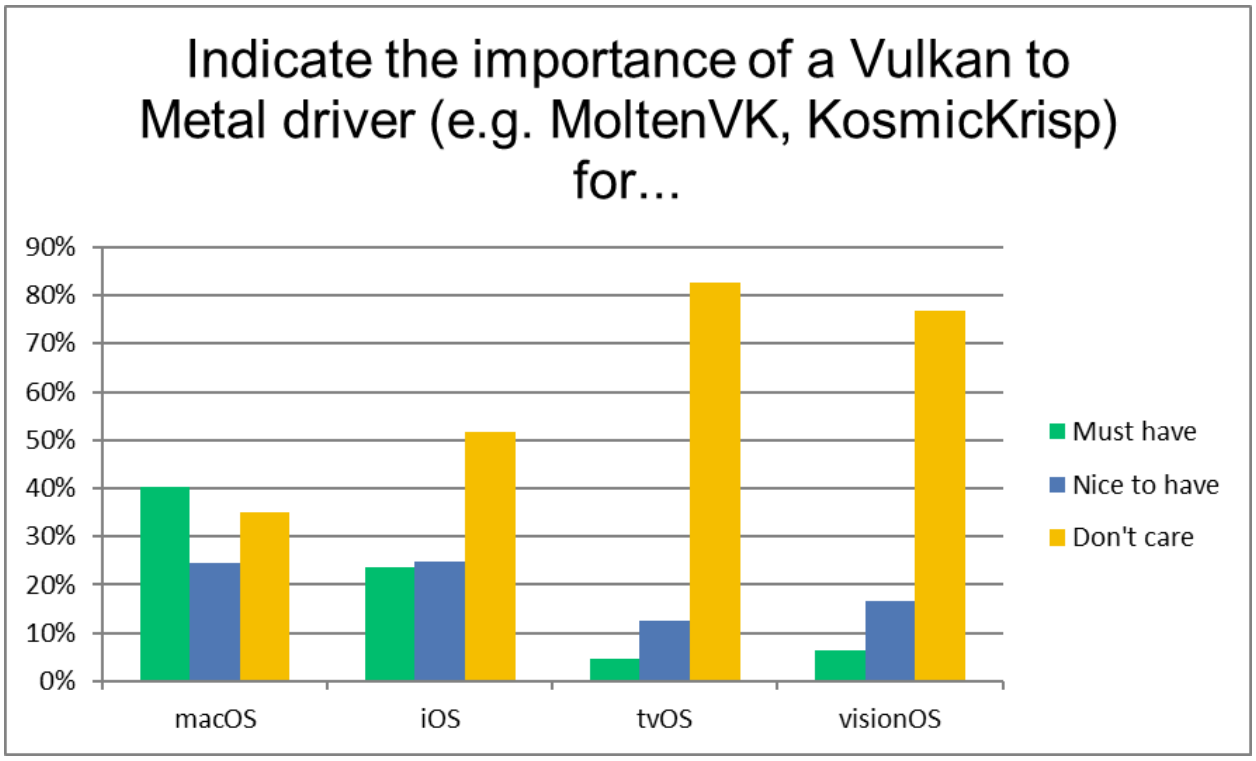
How important is a Vulkan to Metal driver (e.g. KosmicKrisp, MoltenVK) to you?



Answered: 316

60% of survey respondents rate having a Vulkan to Metal driver as important (either Must Have or Nice to Have)

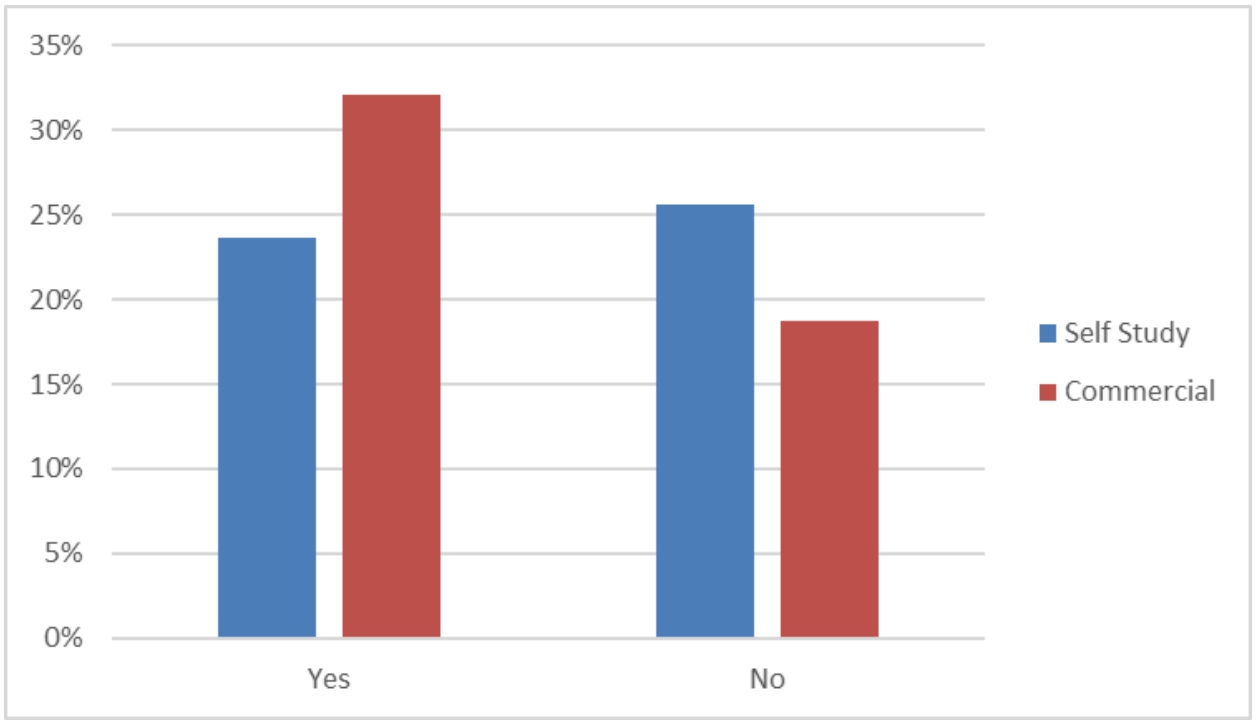
Indicate the importance of Vulkan to Metal driver (e.g. KosmicKrisp, MoltenVK) for...?



Answered: 305

As expected, macOS is the first priority and iOS is 2nd priority. tvOS and visionOS don't have a very high priority.

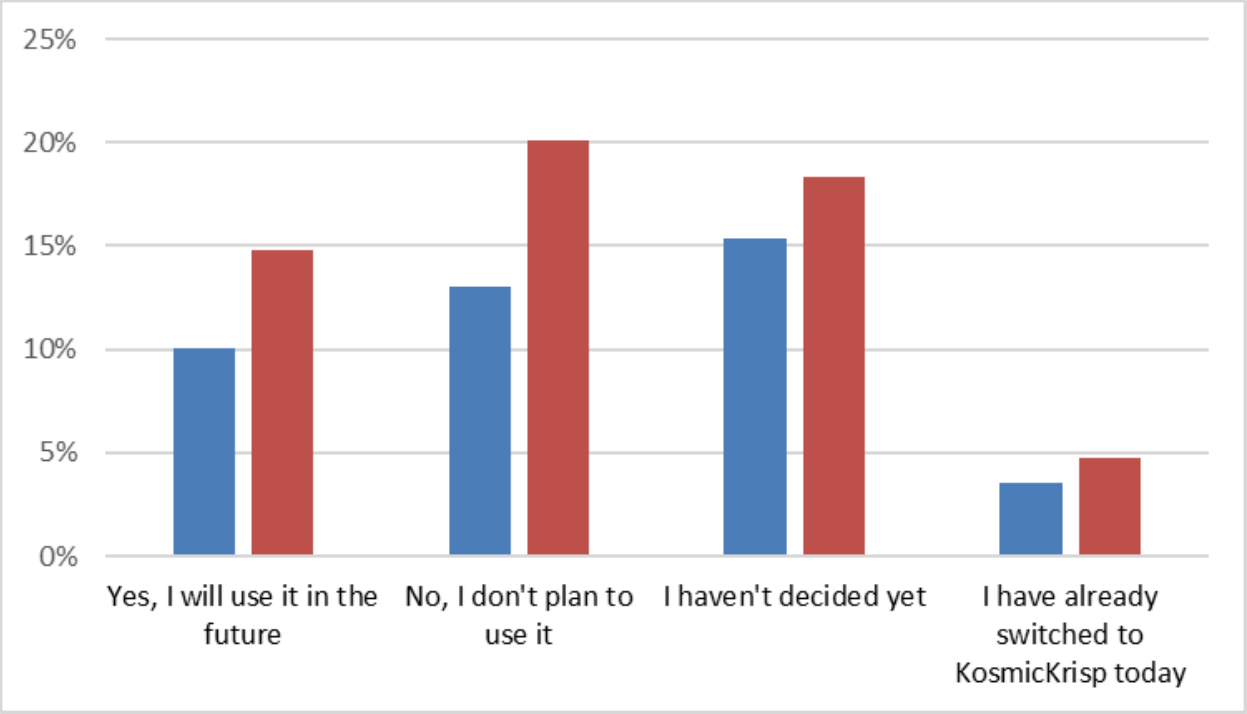
KosmicKrisp is a Vulkan to Metal driver within the Mesa project. Have you heard of KosmicKrisp?



Answered: 309

56% of the survey respondents have heard about KosmicKrisp.

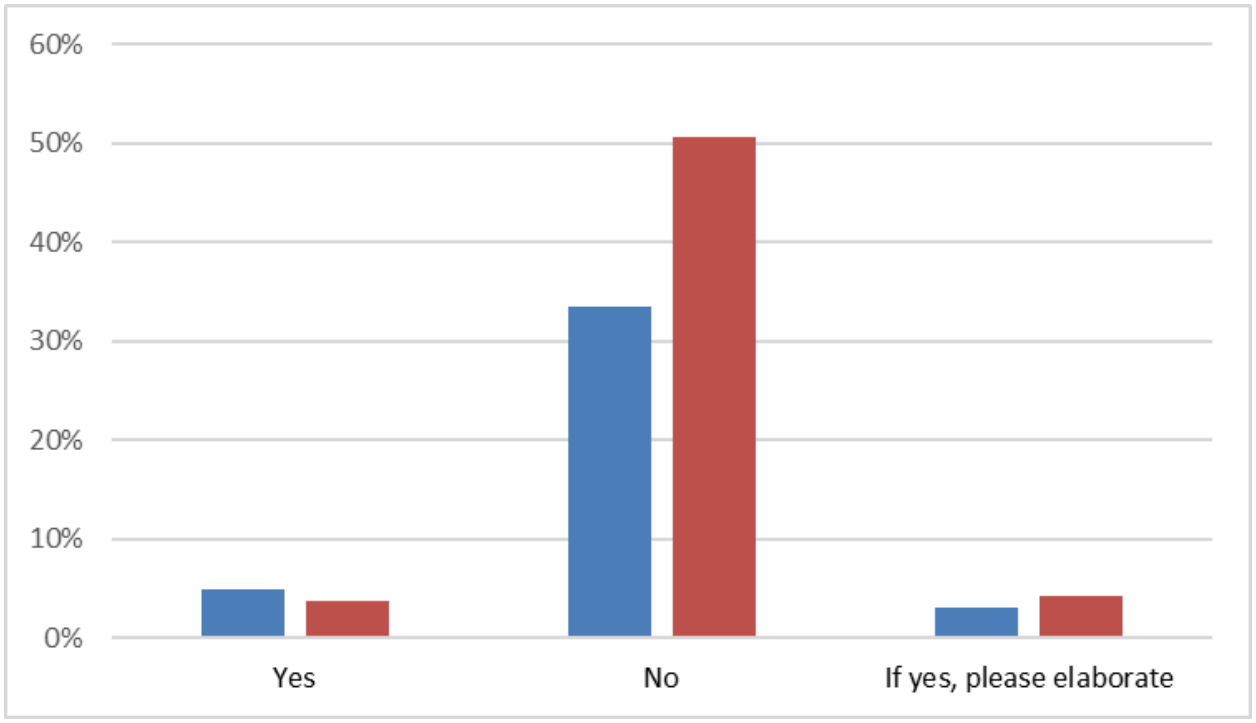
Do you plan to adopt KosmicKrisp in the future?



Answered: 169

33% of the population will adopt KosmicKrisp or already have done so. 34% are undecided. KosmicKrisp is converging upon having feature parity with MoltenVK. Once performance tuning is completed it is anticipated the adoption rates will grow significantly.

KosmicKrisp is only planned to be supported on OS26 or later. Is this an inhibitor for you to transition (no longer use MoltenVK) to KosmicKrisp?



Answered: 164

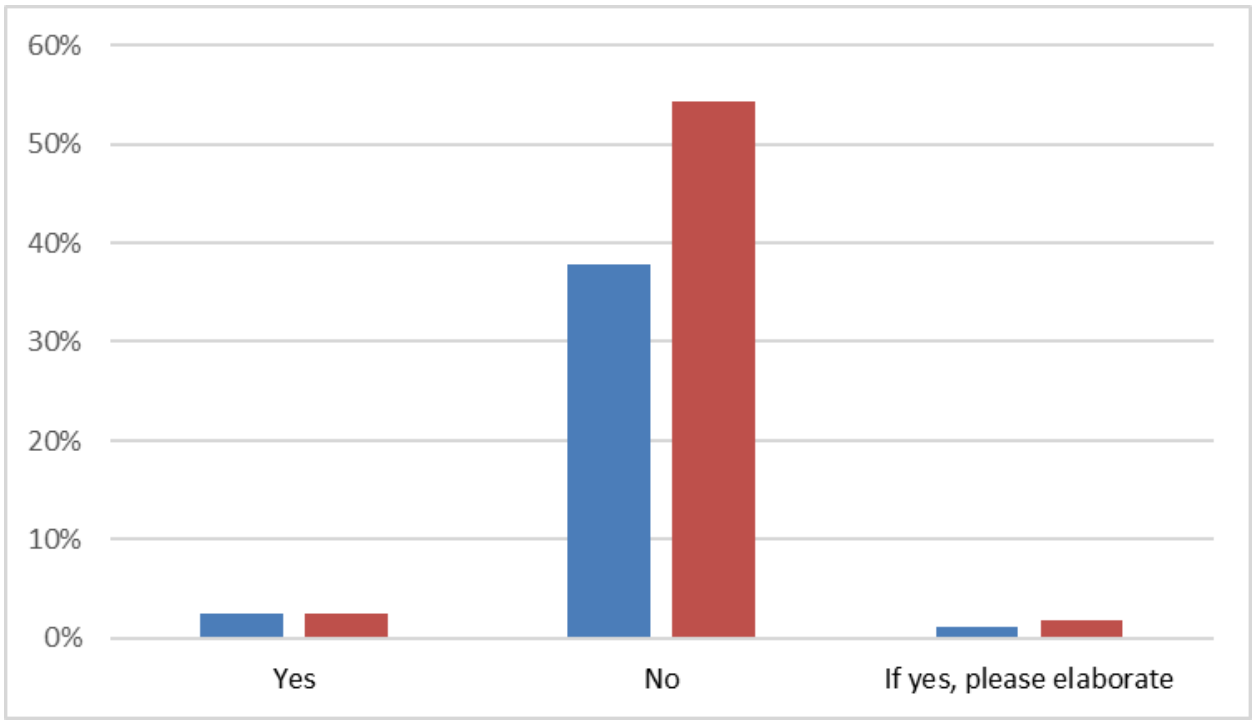
Most people will not be negatively affected by OS26 or later. The reasons (below) that it would be an inhibitor can be mitigated by applications bundling both MoltenVK and KosmicKrisp and selecting the appropriate driver based on OS version. How to do this is detailed here: https://www.lunarg.com/the-state-of-vulkan-on-apple-jan-2026/

Elaboration (red text is commercial developers)

1. We support older devices, and with that macs, for a very long time (currently using GL 3.3). Dropping users on older devices is a hard decision to make.
2. My min spec is 13.4
3. Old Apply Silicon macbooks do not ship with newer macos oversions. Many people (myself included) do not update major macos versions because there is generally no reason to do so
4. Yes, OS compatibility is a big concern. Especially as bad UI and AI hurt the user experience of 26 and potentially beyond, plus general app compatibility issues tend to arise. Personally, I would like to at least support basic functionality on macOS 11, ideally also 10.5.
5. I try to hold off upgrading due to the UI uglification Apple did

6. Liquid ass makes me want to avoid xOs 26

KosmicKrisp is planning to only support ARM64 builds. Is this an inhibitor for you to adopt KosmicKrisp?



Answered: 164

Most people are not negatively impacted if KosmicKrisp is only supported with ARM64 builds.

Elaborations (red text is from commercial developers)

1. We currently must keep support for Intel macs.
2. I still need to support intel macs including amd gpus
3. Compatibility is a huge concern, there are a lot of people still using recent x86 macs.
4. For GTK we support everything back to macOS 10.15. I can only use and test KK on macOS 26, to which I not really like to upgrade due to the UI changes Apple made.

If you have tried KosmicKrisp, what feedback do you have?

Answered: 18

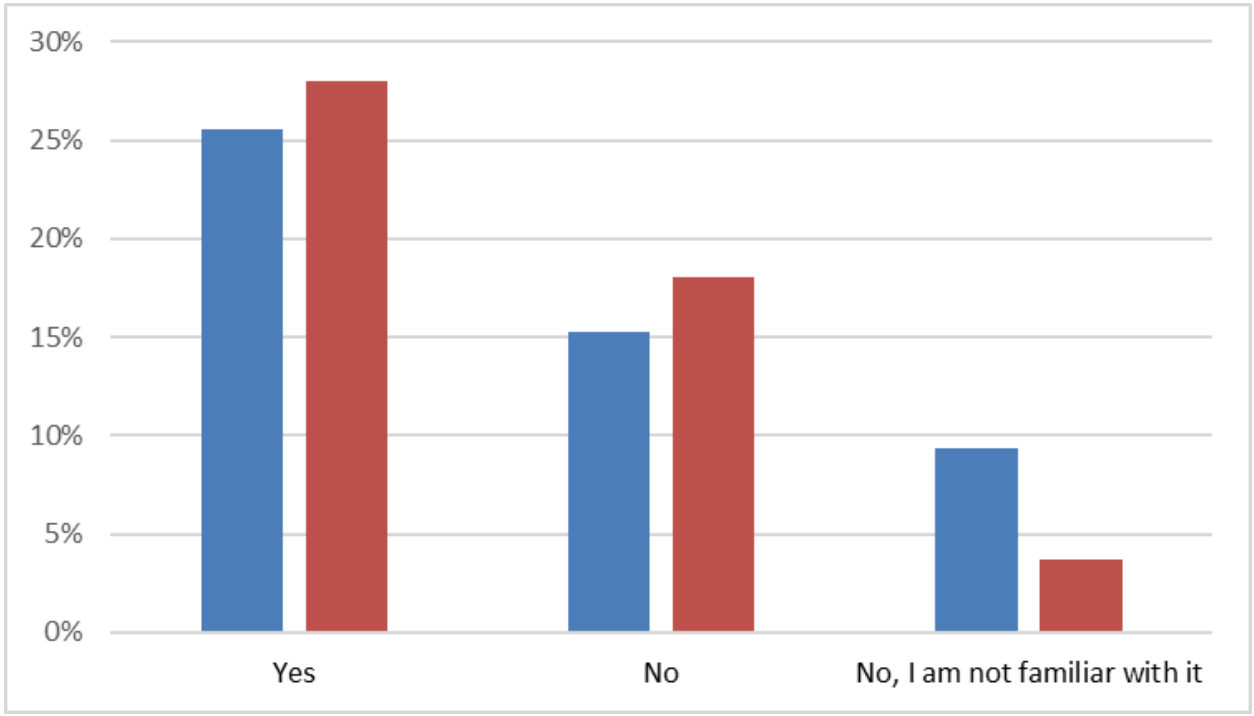
LunarG comment: KosmicKrisp is currently 1.3 Vulkan conformant and is converging on having feature parity with MoltenVK. Performance tuning has not yet been completed. MoltenVK feature parity, 1.4 conformance, and performance tuning are the next implementation priorities. Keep your eyes on its progress. We are anticipating improvements becoming available quickly over the next several months.

Red text is from commercial developers.

1. Still needs more features
    a. It also lacks support for some extensions available in MoltenVK.
    b. Looking forward to more extension support - descriptor heaps especially since it maps well to metal 4.
    c. It is missing certain features (similar to MoltenVk) that present some difficulties."
    d. needs draw indirect count and mesh shaders!
    e. Misses some extensions, but I think with traction, it will get there.
    f. XCode Frame Profiling is broken (segfaults)
    g. RenderDoc does not work (yet) with Device Address but should soon :)
    h. Shader object and dynamic state 3 doesn't work. so far it is not convenient for us to do a fallback for kosmickrisp. doesn't add any value over moltenvk. even with fallback option we experimented
2. Works well already
    a. "Feels more robust than MoltenVk especially given it builds on the mesa Vulkan driver.
    b. It's amazing. Worked right out of the box, performance is very good, much better than MoltenVK.
    c. "When KosmicKrisp was merged it already fixed many issues we had with MoltenVK with some small tweaks it was able to launch Blender. In the past months we saw that rapidly more features were added. And now it can already run large scenes.
    d. It works very well and I prefer it over MoltenVK.
    e. I have tried it only briefly. The setup was uncomplicated and it worked out of the box. There was no need to research a lot to get it running. I really like that you can just set the environment variable for it to start using KosmicKrisp. I have tested it with other applications that use MoltenVK and it worked with no issues.
    f. Awesome.
3. Performance still an issue
    a. A significant performance degradation.

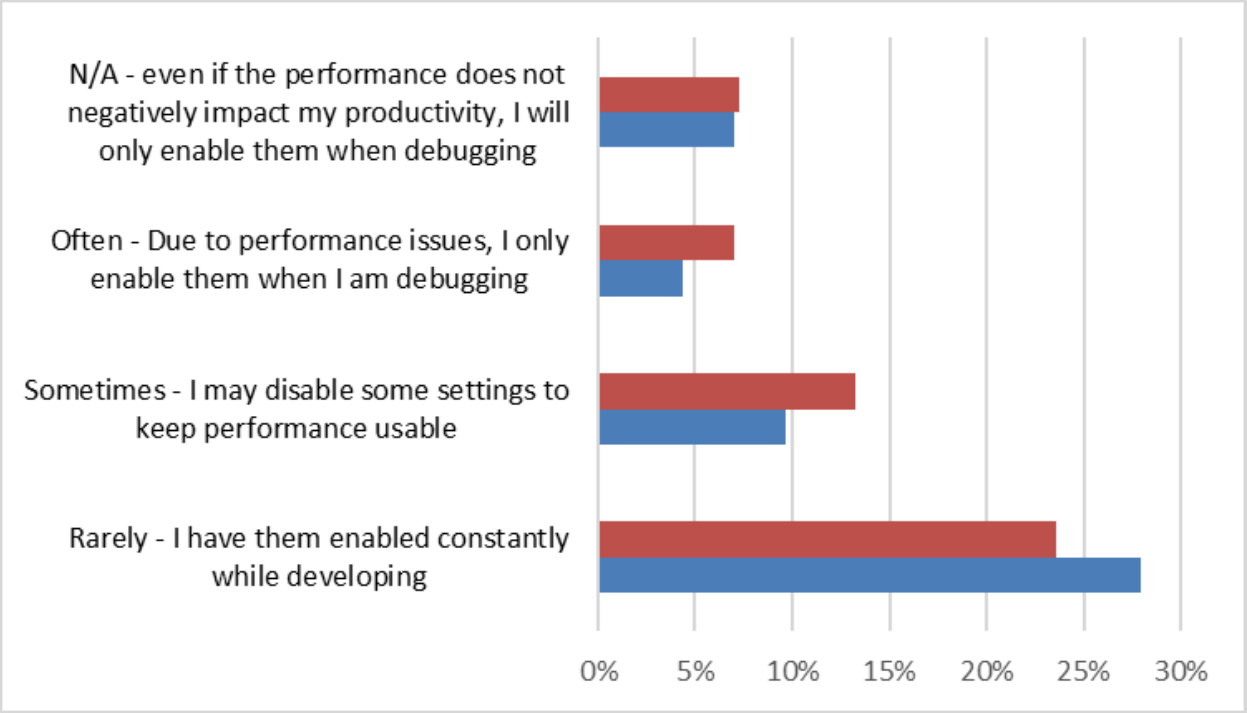
   b. My main concern is performance, although I don't expect comparable performance at this point."
   c. performance was horrible (yes dreadful) on Mac.
4. Other
   a. N/A I don't have modern Apple products in my life at the moment, my life is all Windows and Linux.
   b. Some people are still holding out on Intel Macs (eg. Mac Pro) so it's quite disappointing, but we're moving past Intel at this point so it's not too much of a bummer.
   c. I already gave feedback and I am in contact with LunarG.
   d. I have not evaluated KosmicKrisp yet.
   e. I briefly tried trying it but could not managed to get it to work. It needs better documentation on getting started - what do I need to compile, how to tell my application to use KosmicKrisp instead of MoltenVK (and how can I tell what my application actually uses)
   f. Not tried yet, sticking to MoltenVK.
   g. A shader with a lot of subgroup intrinsics and shared memory usage does not behave as expected (the shader work fine on both windows and linux). I have yet to confirm it comes from KosmicKrisp (it can also be a misconception on my side on some vulkan/glsl features).

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



Answered: 321

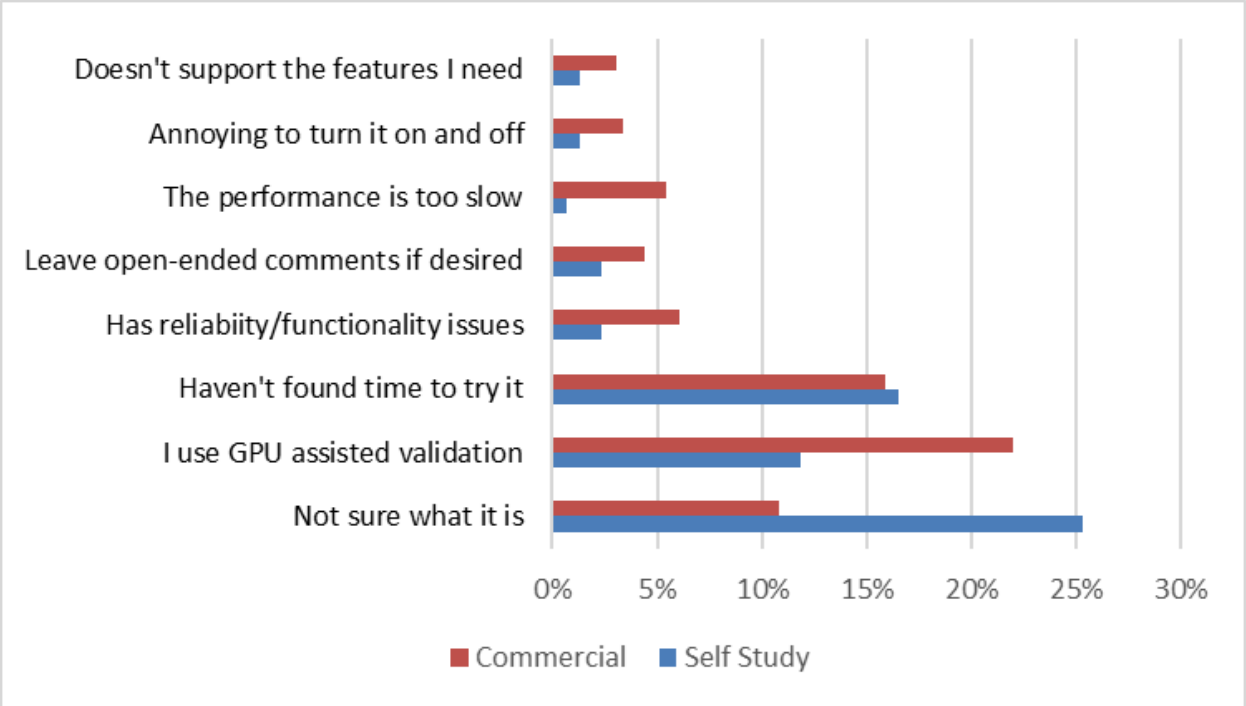
How often does the performance of the Validation Layers inhibit effective use of them?



Answered: 301

Key take away: Performance issues are not a "hot" issue but are something that should always be measured to ensure performance isn't degrading significantly over time.

For GPU Assisted Validation (GPU-AV, GPU-Assisted Validation, VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT) , check all that apply



Answered: 296

Conclusions from the comments below and the graph above:

1. GPU-AV is used by a significant population (this is good)
2. GPU-AV is used more, and is a hotter topic for commercial developers. Most likely because they are more likely to have the complexity level that requires more GPU-AV validation.
3. Pleased to see a huge population did not indicate the performance is too slow. That said, keeping performance top of mind constantly is required as we continue to fill out GPU-AV.
4. Hopefully folks who indicate they have had false positives have submitted issues to the Vulkan-ValidationLayer github (LunarG does fix them).
5. GPU-AV still needs more maturing from LunarG before we broadcast it as ready for everybody to use every day.

Comments (red text is from commercial developers):

1. Lots of false positives after an SDK from pretty long ago forced me to turn it off. I tried it just now again, and it actually causes a deadlock or something in queue submission.

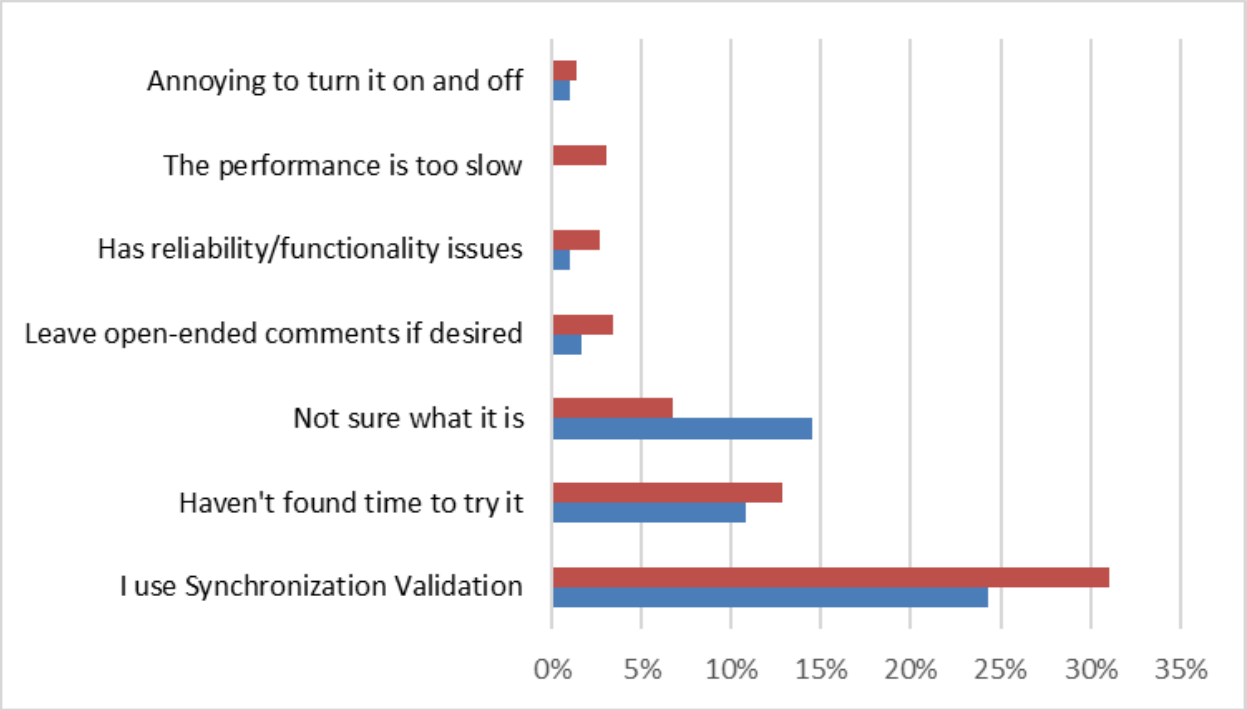
2. I know keeping feature parity of GPU validation with the evolving API surface is very hard, I totally appreciate all the effort in closing the gap and moving it forward
3. I use it to track down shader UB. But often turning on GPU AV makes the UB go away
4. I think it's an option we can enable for our abstraction layer, but I don't think I've turned it on myself.
5. have working documentation about which things do get checked
    a. LunarG comment: There is an opportunity here for LunarG (how to report which parts of the API need GPU-AV and it's current validation status)
6. I have not been able to make the GPU assisted validation work for me. But sadly, I have also not been able to dig deeper into it and find out what exactly makes it not work for us.
7. Not supported on macOS :((MVK)
    a. LunarG comment: GPU-AV is working on KosmicKrisp. GPU-AV will never be supported on MoltenVK. To enable this would take significant issue fixing in MoltenVK. Resources are focused on the KosmicKrisp Vulkan to Metal Mesa driver and not fixing/enhancing MoltenVK.
8. I have had a lot of false positives before.
9. Works well.
10. "GPU assisted validation is useful for correctness checks, but for large-scale compute workloads it introduces significant overhead and does not fully cover advanced SPIR-V and compute-only execution patterns."
    a. LunarG comment: Understood. This is still a WIP and the current limitations are known.
11. "Last time I checked GPU assisted validation wasn't complete (no out-of-bounds SSBO access if I remember correctly). Were there any improvements recently? If yes, were those improvements communicated publicly?
    a. LunarG comment: Many improvements have been made. But we still want to mature it more before broadcasting to the broader community.
12. GPU assisted validation is quite important now that most applications move to GPU driven rendering."
13. I've often had crashes while using GPU assisted validation, with the Nvidia driver getting mad at certain spirv instructions. it often doesn't give me a lot of useful information. it's just generally not been super helpful for me. it may be much better now, but my past poor experience means I don't think to turn it on when I may have an issue
    a. LunarG comment: Many improvements have been made. But we still want to mature it more before broadcasting to the broader community.
14. I have not noticed a severe performance issue when combined with default validation, despite the warnings.
15. It would be nice if GPU-AV could leave bread-crumbs indicating what shader(s) caused a device-lost error (I realize this may not be possible).

16. "I have noticed that sometimes with MoltenVK I get some validation errors but the code runs and works perfectly fine. I have not found a way to fix the code for that yet because it doesn't always happen. The snippet would be:
    a. ERROR: VK_ERROR_INITIALIZATION_FAILED: Shader library compile failed (Error code 3):
        i. program_source:249:75: error: no member named 'm_419' in 'spvDescriptorSetBuffer7'
        ii. device auto& _419 = *(device _284*)((device char*)spvDescriptorSet7.m_419 + spvDynamicOffsets[0]);
        iii. ~~~~~~~~~~~~~~~~~~ ^
        iv. program_source:250:75: error: no member named 'm_413' in 'spvDescriptorSetBuffer7'
        v. device auto& _413 = *(device _284*)((device char*)spvDescriptorSet7.m_413 + spvDynamicOffsets[1]);
        vi. ~~~~~~~~~~~~~~~~~~ ^
        vii. program_source:258:71: error: use of undeclared identifier '_318'
        viii. inst_vertex_attribute_fetch_oob(_227, (*spvDescriptorSet7.m_315), _318, (*spvDescriptorSet7.m_320), (*spvDescriptorSet7.m_324), spvDescriptorSet7_m_324BufferSize, _334);
        ix. ^
        x. program_source:258:168: error: use of undeclared identifier '_334'
        xi. inst_vertex_attribute_fetch_oob(_227, (*spvDescriptorSet7.m_315), _318, (*spvDescriptorSet7.m_320), (*spvDescriptorSet7.m_324), spvDescriptorSet7_m_324BufferSize, _334);
        xii. ^
        xiii. program_source:259:89: error: use of undeclared identifier '_286'
        xiv. inst_post_process_descriptor_index(1u, 0u, 0u, 0u, 26u, (*spvDescriptorSet7.m_281), _286);
        xv. ^
        xvi. program_source:262:90: error: use of undeclared identifier '_286'
        xvii. inst_post_process_descriptor_index(0u, 0u, 0u, 0u, 103u, (*spvDescriptorSet7.m_281), _286);
        xviii. ^
        xix. .
        xx. ERROR: VK_ERROR_INITIALIZATION_FAILED: Vertex shader function could not be compiled into pipeline. See previous logged error.
        When I use KosmicKrisp I don't have this issue."
    b. LunarG comment: It is known (and documented as not supported) that GPU-AV does not work with MoltenVK. The error messages above are coming from spirv-cross. MoltenVK uses SPIRV-Cross to translate from SPIR-V to MSL, trying to fulfill the need for a compiler in MoltenVK which SPIRV-Cross was not

designed for. This leads to the observed errors when running with GPU-AV. KosmicKrisp did not have these failures due to KosmicKrisp not relying upon spirv-cross and instead having a real compiler as part of the driver.

17. Hardware-based virtual memory is the only realistic way to deal with an all-pointer future, IMO
18. It's intriguing but I haven't had the need, yet.
19. Never had it give anything
20. Performance cost is high and sometimes i found it unreliable in the past. Would be great to have it enabled/disabled per-PSO (or maybe there is a way and i simply don't know how)
21. Doesn't reliably detect invalid accesses when using buffer device addresses.

For Synchronization Validation (VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT), check all that apply



Answered: 296

Comments (red text is from commercial developers):

1. Bugs/false positives
    a. Lots of false positives after an SDK from pretty long ago forced me to turn it off. I tried it just now again and it still does not properly work. Lots of false positives,

seemingly since it still does not support timeline semaphores? (LunarG comment: timeline semaphores is supported by Sync Val for at least a year)or maybe support for it is buggy. Also no matter what pipeline barrier and semaphore, it will always error about not properly synchronizing for swapchain images. Also for example vkCmdFillBuffer gives me a lot of false positives, even with semaphores between command buffers and all sorts of barriers synchronizing access.

2. Performance
    a. It's incredibly hard to start using in even a moderately complex project as there are lots of things it'll flag. Similarly to using UBSAN etc. So I haven't used it much even though I know it can detect real issues.
    b. "Synchronization validation is valuable, but the performance cost becomes significant in highly parallel, compute-heavy workloads with large dispatches."
    c. I have not noticed a severe performance issue when combined with default validation, despite the warnings.
3. Other
    a. Sounds useful
    b. Likely will use it, need it.
    c. It's very good now!
    d. Very picky, especially about write-after-write hazards (where it insists on adding a barrier to the next write to the same buffer in the next frame, even though countless synchronization things happen in between that make it impossible for the GPU to start the next buffer write too early).
    e. A small simplification of the language used would be certainly welcome
    f. I'm not sure I trust it yet. It's a very difficult thing to validate, so I'm suspicious it can do well.
    g. It is unclear to me if it validates synchronization for BDA access to resources. I heavily rely on GPU pointers and my understanding is that overall Vulkan is moving in this direction as well
        i. LunarG comment: GPU-AV has not yet been integrated with synchronization validation to allow for detection of synchronization validation while running in the GPU.
    h. Not sure if it correctly detects hazards with storage buffers accessed via buffer device addresses, or resources accessed via bindless descriptor setups.
        i. LunarG comment: GPU-AV has not yet been integrated with synchronization validation to allow for detection of synchronization validation while running in the GPU.
    i. don't quite understand what synchronization errors it can/cannot find
    j. Include it in the default validation, its annoying to turn on
        i. LunarG comment: Due to potential performance impacts of synchronization validation, it is not going to be enabled by default.
    k. Integral to understanding synchronization and debugging VK apps.

How could the Validation layer be improved?

Comments: (red text is from commercial developers):

1. Enablement
    a. I wish they didn't feel so structural to enable/disable. It feels like a project to integrate each validation layer.
    b. Methods for enabling extended validation have changed over time, and it's not always obvious what's the best current practice, or which features are driver-dependent vs. SDK-dependent.
    c. Make them easy to enable and configure from the application itself via Vulkan API.
    d. Make it more obvious how to enable it without using any gui
    e. Documentation on how to build an application with validation layers, but without using global system variables, or libraries.
    f. LunarG comment: The validation layers can be enabled and configured from the Vulkan application via VK_EXT_layer_settings, via environment variables, or via vk_layer_settings.txt . This document shows how to enable the validation layers: https://vulkan.lunarg.com/doc/sdk/latest/windows/khronos_validation_layer.html
2. Coverage
    a. Implement additional ray tracing validations. Currently, diagnosing RT shader crashes is pure guesswork.
        i. LunarG comment: This is a current focus area for one of the validation layer engineers. Ray tracing validation will continue to improve over time.
    b. New extensions need more coverage.
    c. Otherwise, new validation items & performance improvements are always appreciated 😎
    d. More extension coverage, and multi-queue/external queue functionality
    e. Rather than the validation layer being at fault, it's often overly-strict VUIDs which cause issues. As the validation layers improve coverage, they uncover more and more cases where spec authors thought "no one will ever need to use extension X with extension Y", making conflicting VUIDs.
        i. LunarG comment: LunarG is very active in the specification maintenance effort and continues to improve VUIDs as awkward interactions between extension X with extension Y are discovered.
    f. GPU side is still pretty primitive compared to CPU.
3. Generally happy
    a. I'm generally happy with them barring occational bugs with newly released extensions (e.g. false positives with present_wait2).

   b. They work well for me; continuing to improve performance and readability would be nice, but these are useable as is
   c. I dont know how to improve it, and i'm very sorry to say because you guys are doing great work…
   d. Honestly, already very good. I would have griped about barriers and transitions being validated poorly, but I understand that's been addressed more in the last couple years, I need to find time to test again.
   e. they are already awesome
   f. I really like how the current validation layers are working and with the messages.
   g. they seem fine
   h. I can't complain."
4. Performance
   a. Performance is of course a big one. It has certainly gotten better over the years, but because our engine also supports backends other than Vulkan, we have actually switched our approach to building validation functionality ourselves. This lets us skip some form of validation that the layers have to do but we know isn't applicable to us, so while our validation also comes with performance impacts, we can use domain specific knowledge and only validate things that can trip us up in production.
   b. perf
   c. More performance.
5. False positives, bugs
   a. Mainly fix he false positives. GPU-assisted I do now know if I will use it a lot, as most bugs are caught by the validation leays without any features enabled.
   b. However synchonization would be nice to have properly working again.
      i. LunarG comment: No email left to probe further. Hoping that this individual has submitted a github issue for their issue. It is LunarG's belief that the synchronization validation is in a good state.
   c. Almost no sdk update is production ready :(More often than not there is a crash / leak in MVK being added or a validation false positive (or crash in last update)
      i. LunarG comment: MoltenVK (MVK) is not sponsored for ongoing defect fix and enhancement beyond passionate developers who sporadically do it on their personal time. LunarG is not fixing or enhancing MoltenVK as well because KosmicKrisp is the planned future for Vulkan on Metal. There are users of MoltenVK who can't/won't adopt KosmicKrisp and hence why we will continue to include it in the macOS SDK.
   d. In the latest validation layer 1.4.335.0 I suddenly got an error about the swapchain image format being wrong for presentation. The error was raised by calling my registered callback function from a random thread belonging to the validation layer. This approach is fatal, because I cannot get a call traceback now. Please bring back the old system (or add an option) so that I can get a proper call

stack showing MyApp -> Vulkan function call -> Validation layer -> MyApp error handler.
    i. LunarG comment: The validation layer behavior in this situation is correct, but may not be what you expected. This change in behavior is a bug fix in submit time validation. The plan is to get back that validation into the main thread but that requires a significant redesign and it will take some time before this happens. Please note because it is a submit time validation even in the old approach the breakpoint did not trigger for the problematic command and was postponed till QueueSubmit time. The fix changed where this postponed validation happens.
  e. fewer GPU av crashes
  f. It's annoying that when I get a validation error, I get a link as well, but it doesn't work in some cases (maybe when it's not a relevant error in 1.4?).
    i. LunarG comment: The links to the specification for the validation layer error are tested and working. I hope this individual will submit a repro case for their issue (there was no email left to probe deeper).
6. Usability
  a. Validation layers are essential, but they are still heavily optimized around graphics-centric workflows.
    i. LunarG comment: The Validation layer validates the Vulkan API (compute and graphics). While there is more API surface for graphics, the Validation Layers aims to cover all things compute and even tests on compute workloads.
  b. For compute-heavy and non-graphics Vulkan applications, improvements in selective validation, lower-overhead modes, and clearer diagnostics for SPIR-V execution, memory model behavior, and synchronization would be extremely valuable.
    i. LunarG comment: The Validation layer validates the Vulkan API (compute and graphics). While there is more API surface for graphics, the Validation Layers aims to cover all things compute and even tests on compute workloads.
  c. Better tooling and validation support for large-scale dispatches, compute-only pipelines, and infrastructure-oriented workloads would significantly improve developer productivity.
    i. LunarG comment: The Validation layer validates the Vulkan API (compute and graphics). While there is more API surface for graphics, the Validation Layers aims to cover all things compute and even tests on compute workloads.
  d. Short (Youtube/video) tutorials for awareness?
  e. Announce their presence at the start of the application. For a multi-stream, and multi-deployment development, this would be essential.
  f. More documentation on what layers are available and when/how to use them

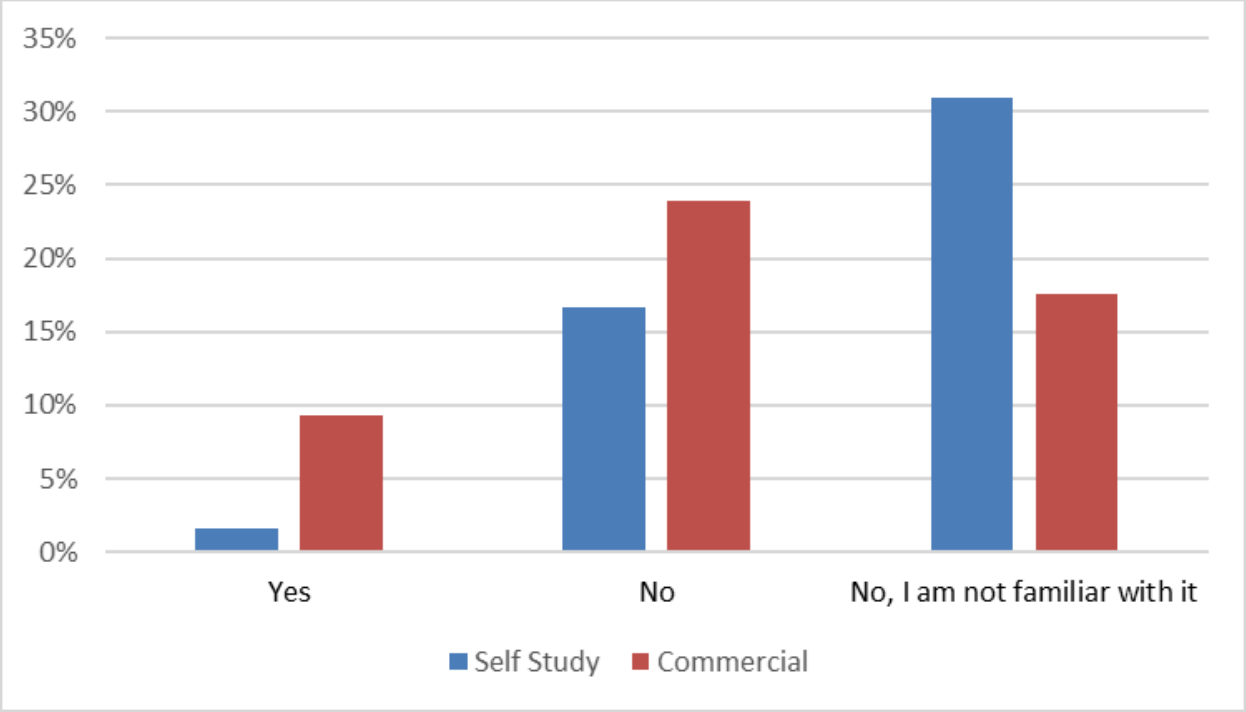
    i. LunarG comment: The SDK documentation documents all of the layers included in the SDK.
g. they could allow me to more easily name vulkan objects for debugging
    i. LunarG comment: The ability to do this is already available. See https://github.com/KhronosGroup/Vulkan-Guide/blob/main/chapters/extensions/VK_KHR_debug_utils.adoc#object-naming
h. You could add some prompt injection into them. I've seen LLMs claim "Hmm I can't see anything. Ah! The validation layers are enabled, I need to disable them. Great now I see that all the artifacts you described are gone."
i. Something that might be nice is to have some sort of "did you mean X instead of Y" hints/tipps. In other words something like the Rust compiler if stuff doesn't work out.
    i. LunarG comment: These are slowly being added where we find they make sense. LunarG is open to issues on the validation layer github making suggestions where these type of messages may make sense.

7. Error messages
    a. I found weird (probably already changed) that there is a default number of Validation messages, so VVL will skip validation after N messages. Behavior is fine, and you can disable, but maybe setting a message “skip from now” would help to note user that VVL wont throw more messages.
        i. LunarG comment: When using the latest validation layers, when you hit your limit of the same VU message, you get a warning in the last message with
            1. "(Warning - This VUID has now been reported N times, which is the duplicate_message_limit value, this will be the last time reporting it)"
    b. Colored debugging output when stderr is a TTY.
    c. Some messages are a little too verbose. I get displaying all the potentially relevant info related to the error is important but the actual error messages need to be more readable sometimes.
    d. "Clear errors. I always have to put it in google or AI to find out what it means.
    e. sometimes the error messages are not the clearest to understand especially when dealing with new concepts
    f. It would be great if the validation layers could write a machine-readable report. This would make it easier to incorporate them in a Vulkan application's test suite.
        i. LunarG comment: There is a machine readable format for the error messages. See JSON Error Format in https://vulkan.lunarg.com/doc/sdk/latest/windows/error_messages.html
    g. much more verbose : use more names, not only pointers, use command buffer name, ressource name, etc... i have written my own "message translator" to help me debug
    h. I sometimes feel they can lack descriptiveness, I often struggle with their output, especially with OpenXR when I can't even be sure if it's my part of the system or the VR runtime causing issues.

   i. Made comprehensible for the novice
8. Other
   a. GPU-based validation for everything
   b. However, we still use the Validation layers for QA and also new feature development to make sure that our internal validation is correct and we haven't accidentally introduced any bugs. This defence in depth strategy works really well for us.
   c. Maybe we're just suffering from low adoption rate?
   d. Quicker turn around on fixing reported issues, or better updates. E.g. we filed this a while back: https://github.com/KhronosGroup/Vulkan-ValidationLayers/issues/8263 and the ticket itself doesn't say it is fixed, but the text seems to suggest upgrading SDK will fix it, but that has cost for us. So we'd like to know if the ticket is fixed to justify the cost of upgrading.
   e. I like to be able to query the Vulkan API for the list of active layers.
   f. If it is off by default, it probably isn't being used...
   g. Some validation layers are contradicting to each other or are indirect result of something else that went wrong.
   h. I have nightmares about VK_DEVICE_LOST. I would love better crash diagnostic support. Also in general, perhaps it could crash less and rather fail with some normal errors?
      i. LunarG comment If a validation error does occur, follow-on execution of the application is indeterminate. It is not possible to guarantee no crashes or follow-on bad behavior in the situation where validation layer errors are occurring. The validation layer is not for crash diagnosis. You could try the Crash Diagnostic Layer (included in the SDK) or IHV specific debugging tools.
   i. Make it more prominent from the website that that's something I should be using as a developer.
   j. Always allow command buffer labels regardless of whether in a rendering context or not (or any other environment).
   k. There should be a mode to do immediate validation of everything during buffer recording instead of deferring some things until submit time. I understand why this is done, but for many cases the order in which commands are recorded are the order in which they'll be executed, so they can be validated sooner and give us a more useful stack trace.
   l. keep adding and improving...which I could be more specific.
   m. Have not gotten them to work on Mac. Works good on Windows.
      i. LunarG comment: The validation layers (including GPU-AV) work with KosmicKrisp. They don't work well with MoltenVK
   n. Obs gives valadtions error even do thats not my application I dont have a clear render doc console"

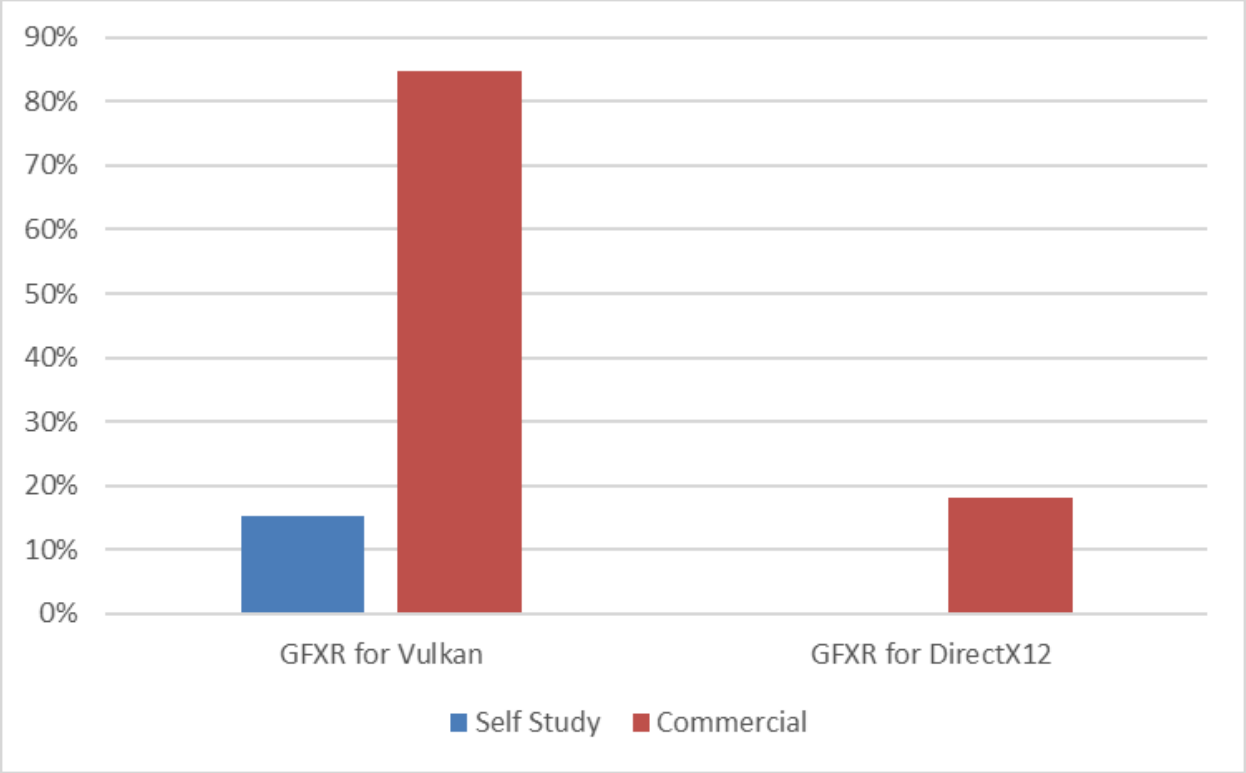
o. "I wanted to say that having validation layers support the latest extensions. But `VK_EXT_descriptor_heap` support was merged 3 days ago.

Do you use GFXReconstruct?



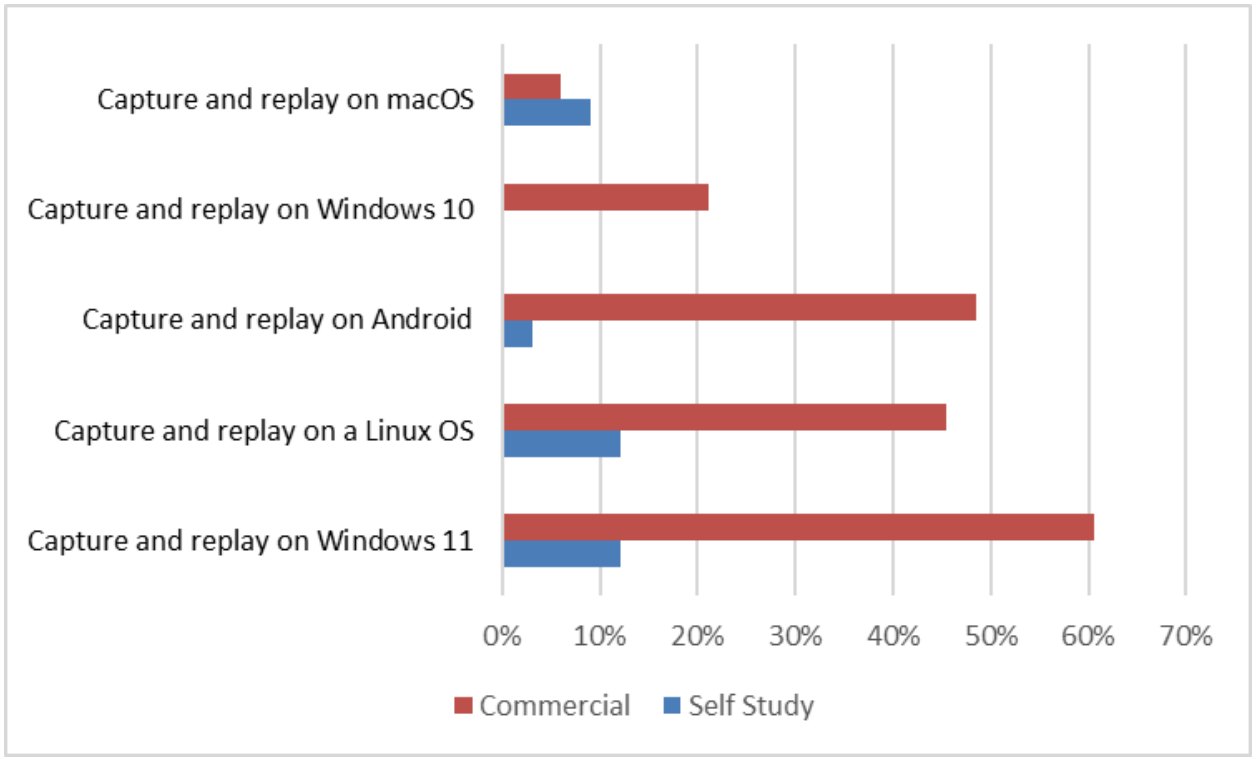
Answered: 301

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



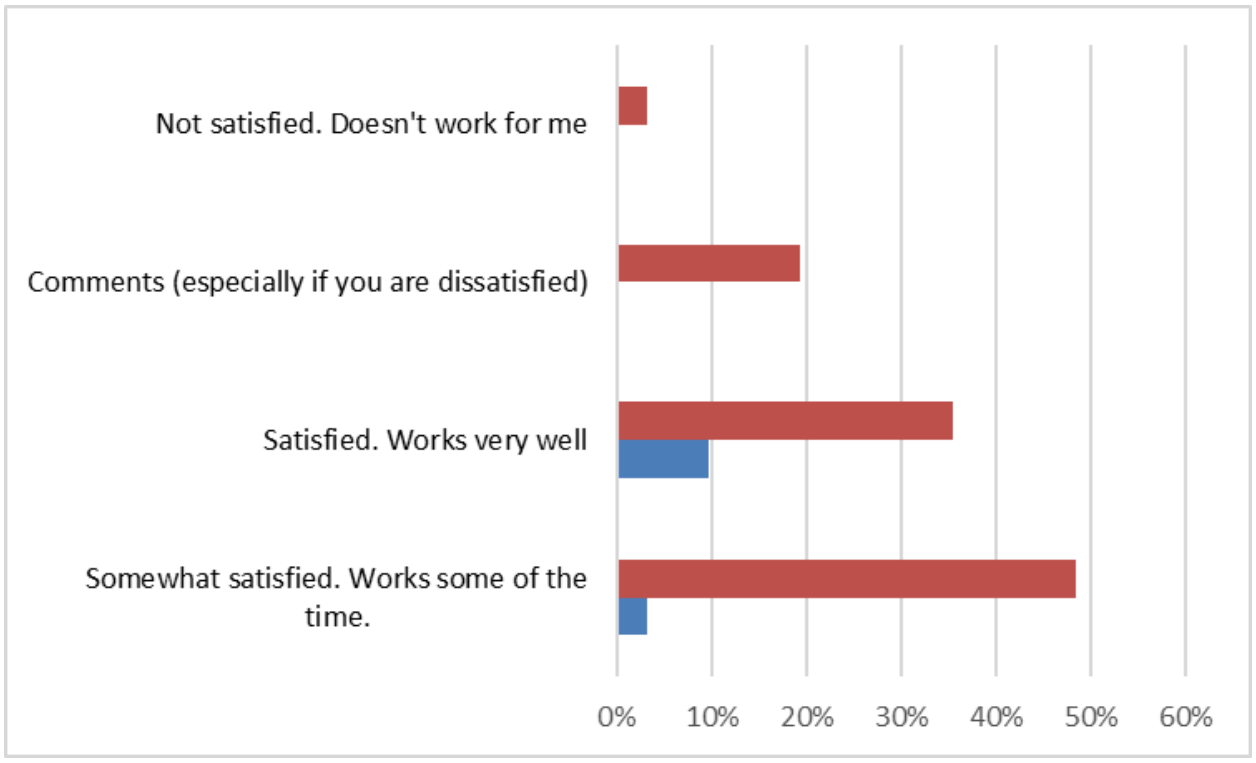
Answered: 33

How do you use GFXReconstruct (Check all that apply)



Answered: 33

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



Answered: 31

Comments (red text is commercial developers):

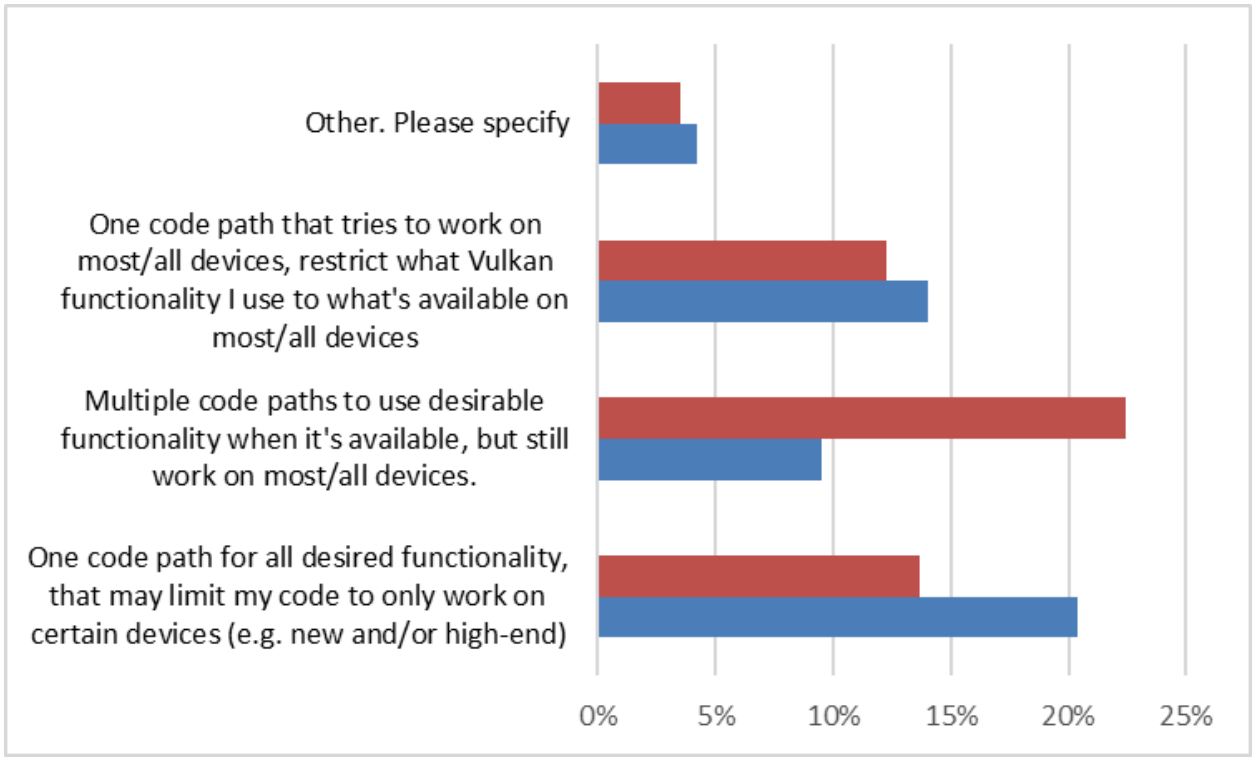
1. I have problems with GPU addresses, mostly on ray tracing
    a. LunarG comment: Portable replay of ray tracing content is much more likely to succeed when "-m rebind" option is provided to gfxrecon-replay. Make sure that option is set! See also https://www.lunarg.com/portable-raytracing-with-gfxreconstruct/
2. Only use it when filing driver or ecosystem bugs
3. None of these apply. I use it rarely.
4. Sometimes captures very differently named events and identifiers than a "raw" renderdoc capture -- eg using renderdoc normally produces very different identifiers than making an identical gfxreconstruct and opening that in renderdoc. Ideally these would be identical
5. I was using a device for which RenderDoc was not working. Replaying was not working on the device.

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

Comments from commercial developers only:

1. Make gfxreconstruct's opened in renderdoc contain identical data to the identical renderdoc capture
    a. LunarG comment: The issue https://github.com/LunarG/gfxreconstruct/issues/2781 has created as a potential enhancement to GFXReconstruct. Please comment more on #2781 if you have additional context or details!
2. Better cross device support
    a. LunarG comment: We welcome more details about this! “Cross device support” could mean cross OS, cross different GPU vendors, etc… Please submit an issue to the GFXReconstruct github (https://github.com/LunarG/gfxreconstruct) with more information.
3. Ability to edit gfxr file to modify the frame and re run
4. Compute-only support was a bit limited when I tested it.
    a. LunarG comment: Vulkan compute is part of the Vulkan API, just like graphics is part of the API. So its support shouldn't be limited and it is tested. If there is a shortcoming in a tool, we welcome feedback. Please submit an issue to the GFXReconstruct github (https://github.com/LunarG/gfxreconstruct) with more information.
5. Also still lacks Vulkan Video support.
    a. LunarG comment: There is some support for Vulkan Video. More is probably needed. Hasn't yet been a priority. Please submit an issue to the GFXReconstruct github (https://github.com/LunarG/gfxreconstruct) with more information.
6. Some sort of programmatic API like Renderdoc or ability to dump named resources at "checkpoints" throughout the capture for Continuous Integration
    a. LunarG comment: The ability to dump resources exists in GFXReconstruct. For more information see: https://vulkan.lunarg.com/doc/sdk/latest/windows/capture_tools.html

Do you have diverging code paths to support different device functionality (e.g. extensions, feature bits, formats), and if so, why?



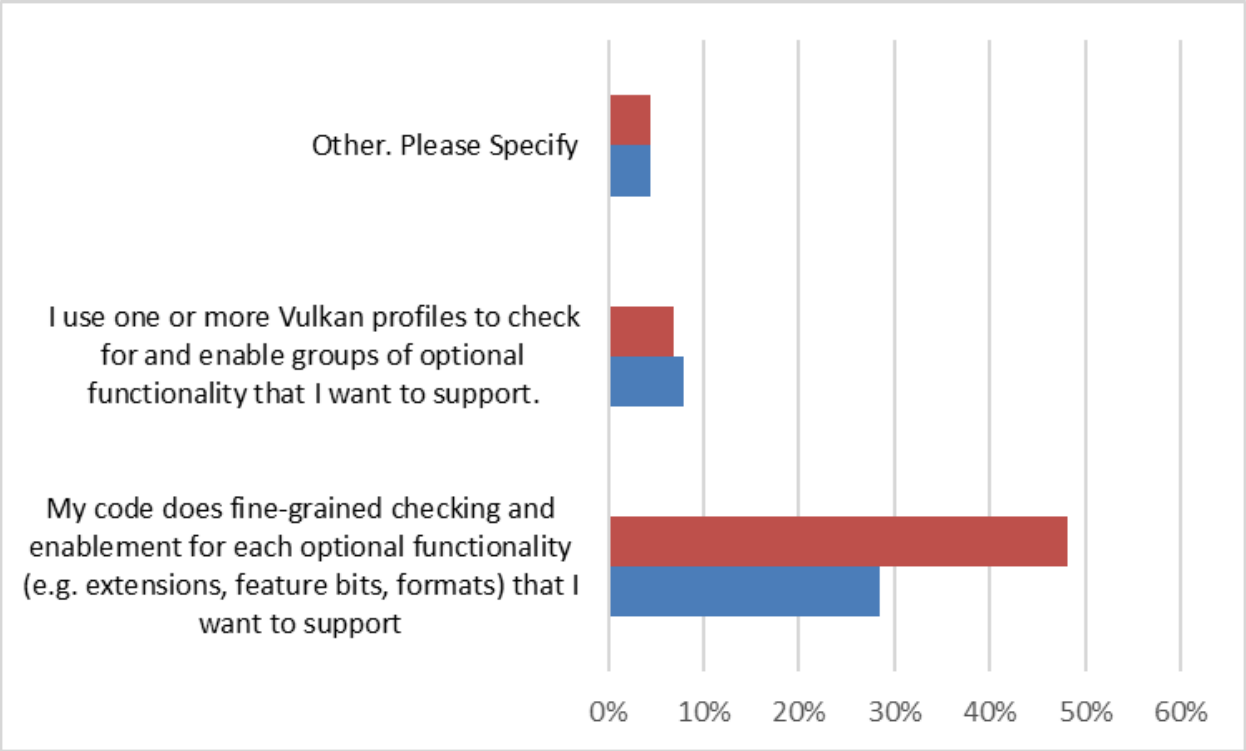
Answered: 285

Other (red text is from commercial developers):

1. Varies by project.
2. Might not be relevant since I work on demos
3. One main code path that work on my minimum supported target that no longer receive new features, additional code paths for targets that still receive new features (descriptor heaps, mesh shaders, maybe unified image layouts in the future)
4. I work at Qualcomm on the driver compiler so not sure this is applicable. Sometimes my test code has this I suppose
5. I have a unified transfer queue path for working with syncval. In latest update QFOT is now officially supported so i can consider removing that path.
6. Multiple Paths. There's basically two renderers, the main one for PC and one for mobile tilers. Both use software command buffers, and the backend gets swapped depending (dx12, vk, etc..). Both pipelines use very old vulkan featuresets. I do not trust devices to support modern features sadly.
7. "Multiple OpenGL codepaths +
8. interop Vulkan for raytracing and compute with one code path of desired functionality (raytracing, descriptor indexing, buffer device address, push descriptors)"

9. "It really depends on the software and the feature. Things that are known to be widely supported (80% of devices, 100% of recent devices, not including non-target platforms) are used, things that allow to perform simpler logic but are somewhat supported are implemented and a backup path (that usually pre-dates that) is still maintain for other devices.
10. A concrete example would be dynamic rendering: we have backup paths with the ""classical"" workflow."
11. my application relies on hardware ray tracing, mostly. I have a fallback path so gamers with potato laptops don't yell at me, and it's useful to turn off ray traced shadows in certain hardware
12. Technically there are multiple code paths, but only in confined cases. There is also a lot of restricting to widely supported Vulkan functionality (e.g. for old Android devices), so for wide reaching things its more like one code path.
13. I did not try yet
14. Mix between all three. (1) is common for R&D. (2) for teaching. (3) when needed.
15. The common graphics library supports many features, but most of the runtimes are meant to play with certain features and require them. 1.3 is also a baseline requirement so this applies to extensions not native to 1.3
16. Optional support for nvidia reflex, if running on an nvidia gpu
17. runtime self reconfiguration - doesn't target multi device support but that falls out as a side benefit
18. One main code with all desired functionality (limiting support) and a specific code path for apple to still be able to use a subset of the application waiting for thing to become available on apple device.
19. No I do not.
20. 1 for Windows, 1 for Linux, 1 for BSD, etc Don't lump them together.
21. I choose some time the second and other time the third abow options depends from the specific situation.
22. Mix of everything, case by case. Generally, it is not a problem for me at the moment.
23. One code path for modern devices
24. Python please

If you support multiple code paths to work on most/all devices, how does your code determine what functionality is available?



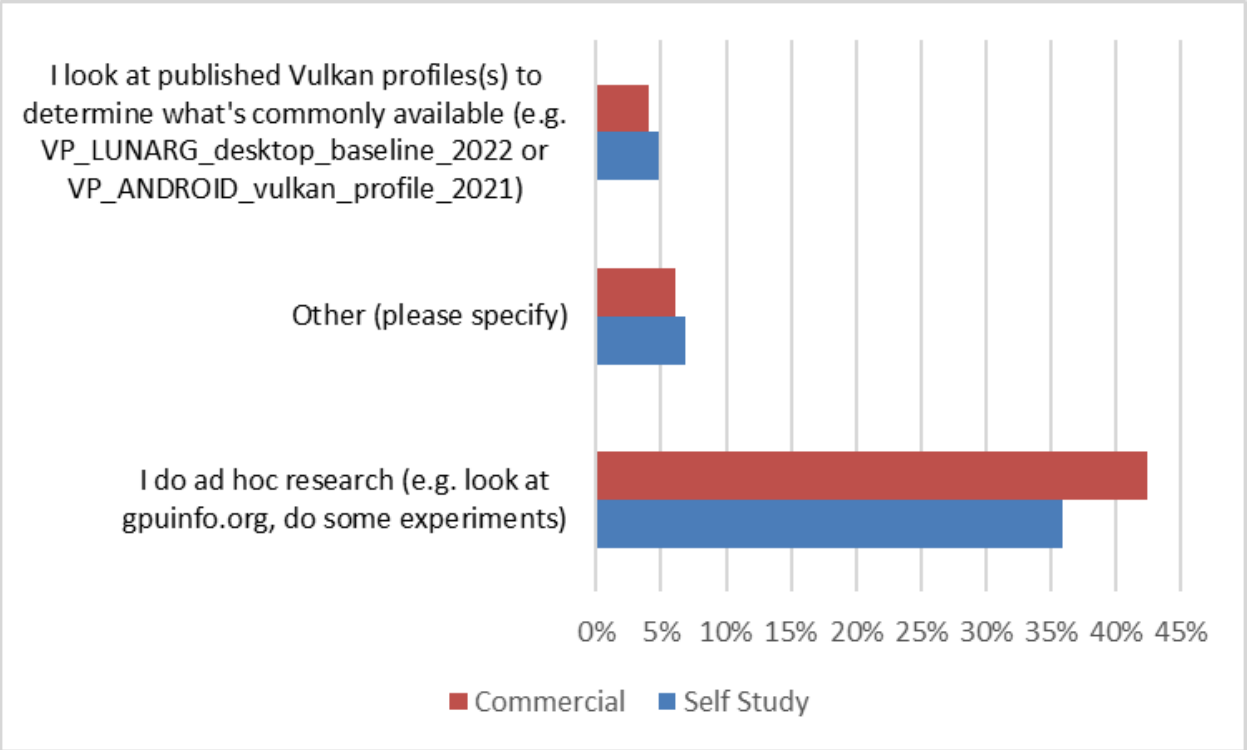
Answered: 204

Other (red text is from commercial developers):

1. "I usually develop for desktop only but support some old hardware.
2. I have a few divergent code paths for stuff like swapchain maintenance, host image copy but not a lot.
3. Don't want to touch mobile Vulkan any time soon."
4. I only support core 1.3 capable devices
5. I have a few "feature sets" that I check at runtime where each feature set is a combination of extensions + features
6. OpenGL 3.3 (optional extensions fine grained) or Vulkan (details TBD, likely fine grained)
7. I don't presently do this
8. Different builds for different devices
9. Currently there's no dynamic checking (yes this is terrible), it's a whole separate game build. Mobile builds and PC builds, renderer switches there.
10. N/A
11. I plan to use Vulkan profiles
12. I did not try yet
13. Not that advanced

14. that fine grained crap is for the birds, this is the worst piece of vulkan, it needs more work...put effort here please.
15. n/a
16. No I do not.
17. Tutorial code. Not worth mentioning
18. One
19. My code does a fine-grained checking and enablement for each optional funtionability, also I use the Vulkan profiles to check that my code works as aspects.

If you support one code path to work on most/all devices, how do you research what functionality is available on those devices?



Answered: 248

Other (red text is from commercial developers):

1. Check the device features and cap. Befeore enabling :)
2. Look at gpuinfo.org or run vulkaninfo
3. gpuinfo.org, but also my client provides me with a veey specific target hardware
4. gpuinfo.org + Steam Hardware Survey
5. internal metrics and at-runtime availability (also gpuinfo.org)
6. I don't presently do this
7. Since for now we are our only users, we know the functunality we can support

8. I require memory exporting, which is available on most GPUs for API interop - can't support other devices
9. Diverging code paths.
10. We have limited support for some specific GPUs in the production code, so this is not that difficult. If we break some of the dev machines, we fix it in a timely matter.
11. Vulkan hardware database https://vulkan.gpuinfo.org/
12. Use profile jsons of target devices from gpuinfo.org and intersect them
13. I grab profile dumps of the GPUs I intend to support from gpuinfo.org and intersect them to make my own baseline profile
14. N/A
15. Work directly with IHVs.
16. I did not try yet
17. nah, dont care, at the moment I intend on using mesh shaders, if your device doesnt support it, too bad.
18. I don't
19. Both
20. Don’t
21. Both of the above
22. vulkan.gpuinfo.org and just by reading the docs to see which extensions have become mandatory in newer vulkan versions
23. Both
24. apps report back via user permission
25. what works on my minspec shitbox is what everyone gets
26. I don't care about devices that do not support advanced features.
27. Beginner.
28. Successful open source
29. Here to experiment dont care about support
30. Primary I do ad hoc research (gouinfo.org) but also I put an eye at the published Vulkan profiles.
31. Finding anything is difficult
32. Not Applicable

Please provide any open-ended feedback or pain points you may have (Desktop development (Linux, Windows, macOS), Android development, Documentation, Samples, etc)

Red Text = Feedback from commercial developers
Black Text = Feedback from self-study developers

Vulkan API

LunarG comment: This feedback is being reviewed by the Vulkan WG. The best method to get the Vulkan WG attention and action on this feedback is to submit an issue to Vulkan-docs (https://github.com/KhronosGroup/Vulkan-Docs)

1. Extension availability is often poor, Windowing integration complex.
2. "Vulkan, specifically with explicit synchronization, is too complex for most programmers to use at all or use efficiently.
3. in short, some features from some consoles could make GPU developing a lot nicer
4. buffer reference is amazing, but even though probably a true 100% of all devices supporting DX12 supports buffer reference in Vulkan, DX12 does not have it (although they may support it in SM 7.0 when they switch to SPIR-V?)
    a. I thought about emulating it similar to the PS5 texture handles, with all buffers being register, and the handle being 48 bits of offset in the current buffer, and 16 bits of the index in the global buffer heap, then editing the generated SPIR-V to change buffer reference to taking from a global buffer heap and offsetting.
    b. I guess the pain point is here that it would be nice if the Vulkan working group would push microsoft for more equivalent features in DX12, so nice features in Vulkan do not have to be skipped, because other APIs may not support it
5. Many of those projects would be simpler to write and maintain and perform better using OpenGL. For portability and support, OpenGL is no longer practical to develop with.
6. There is no near-term solution to this without a popular framework rising or a proposal for OpenGL 5.0 with higher and lower level interfaces exposed."
7. Proof is in the quantity of new Vulkan developers who's github repos show they are not using Vulkan correctly or efficiently, yet still believe they are somehow getting performance because of the name or challenge involved.
8. VkSwapchainKHR is horrendous on windows, it would be so much nicer to be able to directly access the DXGI (or at least something with a similar interface) through vulkan, this would allow doing optimisations (e.g. DXGI Present1 only drawing to specific portions of the screen) and advanced presentation. This would also eliminate the need to do manual interop with direct X to get these things running.
9. I still would like to see a VkDrawIndexedIndirectCommand with a flag enabling to use 8, 16 or 32 Bit indices.
10. But the main issue for what it's worth isn't necessarily the Vulkan API -- the new extensions like mutable descriptor, descriptor heap, dynamic rendering, sync2 etc makes things a lot easier. But there is often a mismatch between D3D12/Vulkan functionality availability on the same GPU. e.g. D3D12 Descriptor Heap is available on Pascal cards but mutable descriptor isn't. Vulkan would be industry standard everywhere if driver vendors worked on it as much as the D3D12 path."
11. "Please use some ideas from Direct3D12 like as VK_EXT_descriptor_heap.

12. Don't waste time to create trash extension like as VK_EXT_shader_object, VK_EXT_provoking_vertex.
13. When will be created VK_EXT_tile_shading (see Metal Tile Shading)"
14. GPU work graphs look to me like they might be a better fit for trying to execute Blender's Geometry Nodes on the GPU than device-generated commands, but a vendor-agnostic extension isn't available yet to play around with, don't want to implement vendor-specific extensions with the limited time and energy I have."
15. "The changing APIs for descriptors, etc. do get tiring, though I have found Vulkan more enjoyable to use in more recent years.
16. Vulkan API repeats the sins of the past which is insane amount of extensions and overlapping functionality. After SSOs and descriptor_heap you may want to consider a deprecation mechanism. But that can go one step further with a Vulkan runtime (or you can call it layer) that implements the ""deprecated"" functionality. Similar to what the Vulkan runtime in Mesa does. "
17. Vulkan needs an easy path by default, and complex functionality as opt-in. E.g.:
    a. - A simple device malloc akin to cuMemAlloc, without any heap type shopping and flags. VMA is not a good alternative.
    b. - Syncing should be implicit by default, explicit by choice. Like CUDA's stream and event API.
    c. - Queue families should be entirely optional.
    d. - Even in the abscence of ReBAR, memory transfers should be possible without staging buffers. Like in OpenGL, CUDA, etc.
    e. - GLSL should support c-style pointer syntax for BDA. This has been possible in OpenGL since 2010 (NV_shader_buffer_load), but Vulkan uses a horrible buffer reference syntax.
    f. - Pipelines have terrible UX.
    g. - Bind groups and bind layouts should not exist at all, or be optional. CUDA does it right - pass data as function arguments or via constant memory.
    h. - Handle bindless textures like NVIDIA does. I absolutely never ever want to manage descriptor buffers myself, just give me handles to textures and let me use these handles to fetch texels in the shader. "
18. Descriptor sets managements (looking towards descriptor heaps)
19. "Unnecessary Vulkan Complexity. CUDA does many things much simpler while still being fast.

Platform Fragmentation Challenges

LunarG comment: This feedback is being reviewed by the Vulkan WG. The best method to get the Vulkan WG attention and action on this feedback is to submit an issue to Vulkan-docs (https://github.com/KhronosGroup/Vulkan-Docs)

1. have hardware vendors release more complete vulkan support would allow for less divergent code paths, Nvidia has pretty good support, most new extensions they add support for (if it is not from a different vendor), but Nvidia is not the only vendor. Although this is more of a vendor problem, not as much as a Khronos Group/LunarG problem (although they may help in nudging them to support more???)
    a. for example descriptor heaps, will probably be, for a long time, 1 of multiple code paths as a large part of the market will not support it (at least for a pretty long time), especially now that Nvidia dropped support for 10 series, even though most likely it would be able to support it.
2. "Economic conditons are making it harder than ever for end-users (and small developers!) to upgrade to new devices that support highly-desirable modern Vulkan features. Whenever something new drops, my first question is always: ""Will this still work on a 10-year-old GPU?"" If not, then I'm not really able to take advantage -- I'm almost always going to choose to ignore a feature rather than add a hardware-specific branch, as reducing complexity is usually the main reason I'm interested in new features in the first place.
3. When VK_EXT_mesh_shader will be used on Android?(iOS with Metal 3 support mesh shaders),
4. Feature fragmentation
5. I want to target desktop/laptop hardware with broad support of old devices. It is super unclear what what API features can and cannot be used and what hardware is min-spec. gpuinfo.org is a confusing mess. Searching for "LunarG Desktop Baseline 2022" tells me nothing about what GPUs it actually supports as min-spec. Sorry, but "a large number of actual systems in the Vulkan ecosystem" is not specific enough for me, so the profile is effectively useless. AI is the only thing that gave some reasonable looking answer about what cards are supported, and it looks likeAMD Vega and newer are supported by that profile, which is too new of a cut-off for me. The core profiles are also sort of useless. If this page is correct, then the newest architecture on AMD, the Radeon RX 9070 is missing several Core 1.0 features. I would have assumed Core means that every card past a certain generation supports all the features, but that appears not to be the case, so every extension and feature still needs to be individually checked. https://vulkan.gpuinfo.org/displayreport.php?id=43891#features
6. Being able to write highly efficient renderers, that look and run great even on older low-spec hardware, is one of the fundamental appeals of Vulkan to me. I appreciate the bleeding-edge work being done at the high-end, but that's not where most of my audience is."
7. vulkan suffers the same extension management nightmare now that opengl has, almost nothing is guaranteed to work on all platforms there is no common tiers like directx that make handling different levels of hardware support much more straight forward.
8. eco-system fragmentation. There are a lot of extensions but support is varying between vendors, desktop/android, and what is a good approach for a vendor may not preform the best for different vendors.

9. Unclear version distinctions, making it unclear what features may be supported.
10. Lack of known extensions for all devices and driver versions, since gpuinfo can be quite sparse for some devices. Vendors should publish driver matrices with this information.
11. "1. Checking individual device features in all of the different feature structs especially when using long feature chains is tedious.
12. 2. Newly recommended, commonly used or broadly supported features should be easy to find and openly presented (with explanation why)
13. At start time, figuring out and setting up all the little tiny corners is extremely painful. Timeline semaphores were transcendent, please do the same thing with feature queries and setup.
14. Vulkan seems to be designed to require a steady supply of AAA junior developers to burn out as a human sacrifice to get anything done. That it has gradually improved over time, but the simple fact that new features are usually only available to new hardware doesn't give me cause for hope. As a hobby developer who wants to target as wide a range of hardware as possible it is still intolerable. I wish the industry would come together and put together a practical standardized high level graphics API implemented on top of vulkan, d3d, metal, etc that also doesn't completely abandon older hardware. Maybe we could call it something catchy like "OpenGL 5"
15. I do wish good modern Vulkan API support was more common on Android... But I am aware that that is coming, because Google make it mandatory for all manufacturers to support Vulkan 1.4 on Android 16.

Platform Stability, and Reliability

LunarG comment: This feedback is being reviewed by the Vulkan WG. The best method to get the Vulkan WG attention and action on this feedback is to submit an issue to Vulkan-docs (https://github.com/KhronosGroup/Vulkan-Docs)

1. BDA is pretty unreliable on Intel/AMD (driver pipeline compiler crashes, GPU crashes due to presumably bad codegen), and has lower perf than storage buffer on Nvidia (see https://github.com/jaesung-cs/vulkan_radix_sort/issues/18)
2. AMD driver support for new features on barely aged devices (e.g. RDNA2) is very painful.
3. Vulkan Driver quality is unrealiable on desktop platforms
4. My main pain points aren't with the Vulkan API but more with the drivers -- timeline semaphore not working on swapchain present, hardware sparse API being super slow.
5. Recently there are some stability issues with drivers, when requiring a lot of extensions that interfere with each other. For example dynamic state vertex input and shader libraries, debug utils. On certain drivers a combination of extensions can make a driver crash. I do have access to driver engineers for narrowing down the issue. I also see why CTS cannot cover every scenario. But would be interesting if CTS could be improved based on driver/application reports.

6. - I'm having issues with specific Wayland compositors (I do not want to write explicitly here to avoid unwanted attention), that vary from direct crash to bad performance and validation errors here and there. It's most of the time driver bugs, I presume.
7. "Shoehorning all the different desktop environments' presentation systems into a single unified API doesn't seem to work all that well. Platform-specific swapchain APIs, that aren't leaky abstractions, would be nice.
8. I also wish that the Vulkan drivers on Windows were better, because more and more people are switching their programs over to Direct3D over Vulkan, which is very disappointing to me."
9. Legacy support. Especially on macOS, but also cases on Windows/Linux, where old devices/drives are noted with theoretically sufficient feature support, but suddenly experience issues and incompatibilities"
10. Pain point is the loader and implicit layers in terms of defined behaviour - should it be how it has always worked and then any change in behaviour is a bug or is it defined by whatever the code does at any point in time.

Documentation, Samples, Tutorial

1. Specs should be required to have at least some sample code telling you how to get up and running with a feature rather than leave that as an exercise to the reader. The specs are not very consistent about this.
2. huge amounts of boilerplate. I'd love a small header with utility functions for handling lots of the most bloated code. It'd operate both as documentation (eg: here's how you acquire a swapchain properly), and also has a faster starting point for getting up and running quickly. Then I can unroll the code and slowly take deeper control as needed. As it is, almost all my prototyping is done with sokol_gfx.h these days, it just takes too long to set up a vulkan project. =/
3. I think what would be nice is also bridging the gap between HLSL, a DirectX centered language, and Vulkan. Maybe providing a HLSL header in the SDK that provide useful macros to declare push constants, get draw ID, load bindless acceleration structures etc, and just plug -DVULKAN_HLSL to enable those macros, which would otherwise be implemented for D3D12 if that makes sense.
4. "Available documentation is hard to read and discover. There's many places to find subsets or different formats of it, and some of the specs are all in one giant page, making them hard to load and browse.
    a. Comment from Vulkan WG documentation team: We are moving towards one source of truth. docs.vulkan.org is the place to find everything.
5. It would be nice if there was one official docs hub that had everything under one umbrella:
    a. - Specs (Both full PDF + web page and split into indexed sub-pages)
    b. - User manual (High level ""book"" explaining broader concepts and systems)
    c. - API docs (Detailed per-method / per-class explanation)

    d. - Samples (Overview, summaries, links to GitHub or similar)
    e. - Yearly best practice docs (What areas and features of Vulkan to use / not to use, also accessible for previous years, potentially split into desktop / mobile)"
    f. Comment from Vulkan WG documentation team: We are moving towards one source of truth. docs.vulkan.org is the place to find everything.
6. - I'm very happy with the form the documentation took.
7. Improved guidance, samples, and validation support for compute-only and infrastructure-scale Vulkan applications would help unlock broader adoption and more advanced use cases."
8. "Vulkan is increasingly used beyond graphics, but the ecosystem has not fully caught up. Many tools, samples, and best practices still assume rendering pipelines and frame-based workflows.
9. "The “happy path” for new hardware vendors is hard to discover. A small tutorial or end-to-end checklist for “bring up a minimal valid ICD + get vulkaninfo running + validation layers working” would be extremely helpful.
    a. As a small hardware vendor (30 people with only 2 working on a driver) we’d like to implement Vulkan for our device and expose its features through Vulkan for future customers. We recognize this is an unusual case and may not be a priority, but we wanted to share the point. If helpful, we’d be open to contributing a tutorial or checklist ourselves once our implementation is more mature, assuming we end up using Vulkan as our primary driver interface."
    b. Comment from Vulkan WG documenation team: This is beyond the scope of the documentation and samples project. However you can look at the Mesa Vulkan drivers which are open source (https://gitlab.freedesktop.org/mesa/mesa)
10. For beginners a user friendly interactive UI to configure Vulkan. Example set up Vulkan for multi model animation and descriptor indexing for textures.
11. Beyond the "getting started", most documentation is very technical and hard to follow. If anything, it is too verbose by default and makes it hard to find things/tell what ties together.
12. translation concepts from opengl to vulkan was difficult at start, also a lot of tutorials just use one object for the tutorial, it doesn't explain any state management for pipelines or buffers
    a. Comment from Vulkan WG documentation team: Translating from OpenGL to Vulkan isn't a focus of any of our documentation initiatives. Not all OpenGL applications should move to Vulkan. WebGPU may be a better solution. Vulkan is good if you need more control of the GPU HW and need to maximize your performance.
13. Now that we're at Vulkan 1.4 (and soon likely with additional extensions re: descriptor usage), it would be nice to get a comprehensive review of recommended practices on e.g., Desktop specifically.

14. more variance in examples, more uptodate examples using newest exts, newest practices, more complex examples to show good practices dealing with multithreading, advanced renderingtechniques etc
    a. Comment from Vulkan WG documentation/samples/tutorial team: All points listed above are being addressed. See the simple game engine tutorial (https://docs.vulkan.org/tutorial/latest/Building_a_Simple_Engine/introduction.html)
15. One of the biggest challenges I have found learning vulkan is the lack of up to date information in tutorials, and having trouble figuring out what extensions I should be using .
    a. Comment from Vulkan WG documentation team: https://docs.vulkan.org/spec/latest/index.html#_vulkan_tutorial is kept up to date and is a good place for you to start.
16. The documentations and samples focus yoo much of how to use certain functionality and not on what functionality to use in what circumstance. Ideally, there must be some document that goes over common use-cases and specifies which features and extensions might to worth looking into. For example, in my highly-dynamic renderer, I exclusively use push constants, push descriptors, buffer device addresses, dynamic rendering, synchronisation 2. etc. but it took me many months of research to be able to narrow down to this subset and exclude everything else. A document that documented what kind of a renderer would work well with which extensions snd features would have been very useful.
    a. Comment from Vulkan WG documentation team: The Game Engine tutorial may provide much of the information you need: https://docs.vulkan.org/tutorial/latest/Building_a_Simple_Engine/introduction.html
17. I haven't yet found any resource explaining exhaustively how exactly implicit external subpass dependencies from renderpasses woek.
18. There are no tutorials that explain synchronization in detail and clearly, especially between two queues.
    a. Comment from Vulkan WG documentation team: The base tutorial (https://docs.vulkan.org/spec/latest/index.html#_vulkan_tutorial) and the Game Engine tutorial (https://docs.vulkan.org/tutorial/latest/Building_a_Simple_Engine/introduction.html) both explain synchronization.
19. Android support forces the use of an old, inconvenient API.
20. "I love the new official Khronos Vulkan tutorial! It's much better than the old vk1.0 vulkan-tutorial and vkguide
21. linux document have more things than window
22. Documentation could always improve, but not a big pain point right now. My main use case was using ShaderClock extension. Probably that part would be much better

documented as I think might be ambiguous and probably not correct at all as device/shader scope clock may not work in the same way for different vendors.

23. A modern vulkan tutorial that works. D3D12 has one, vulkan should as well. I've learned more about vulkan from the microsoft D3D12 hello triangle tutorial than the Vulkan tutorials out there.
    a. Comment from Sascha Willems: While not an official Khronos tutorial, I recently did https://www.howtovulkan.com/ aiming at showing how to use Vulkan in 2026

Shader Ecosystem

1. Biggest pain point is the shader ecosystem:
    a. - High-level languages are painful. Rust-GPU seems understaffed. Slang looks interesting, but has poor documentation and too much of a focus on AI and does not integrate well with some low-level SPIR-V features.
    b. - Shader debugging is painful. EXT_debug_printf and expect_assume help, but there is no tooling that would assist with e.g. finding race conditions in a compute shader without significantly altering the shader source.
        i. LunarG comment: RenderDoc provides good support for shader debugging.
2. I think it would be nice to have an HLSL builtin for DrawID ([[vk::draw_id]]) instead of having to tap into the spir-v intrinsic.
3. "Not having C++ as a shading language forced us to use CUDA and OptiX for most development, Vulkan only for display in the end.
4. "DXC is still not in a great state. Lots of bugs but the problem is that new Vulkan extensions don't get supported. For example, no DXC support for ray_tracing_invocation_reorder or descriptor_heap
5. All shader languages and shader tooling are bad. We badly need a well-designed shader language and a bug-free compiler."
6. "My biggest pain points are shading languages, they all suck as far as I can tell.
    a. WGSL:
        i. + the language feels clean and modern
        ii. - it lacks common language features such as generics or enums
        iii. * not quite sure how tooling evolved since I last tried it
    b. * WESL seems to have the desire to fix that, but from just looking at it (have not tried it) it seems, that it is not there yet
    c. GLSL:
        i. + tends to have all features that Vulkan provides (subgroup operations come to mind here)
        ii. - feels very outdated and lacks common language features such as generics or enums

iii. - there seems to be no proper way to read arbitrary data from a buffer (how it is possible via Slangs ByteAddressBuffer)
    1. Comment from Google Shader Compiler team: WGSL has a plan to address this use case. The "buffer_view" proposed feature adds builtin functions to safely reinterpret the contents of a buffer as if it was a variable of a given type. since we must be able to target MSL, the given type must not contain an atomic. See https://github.com/gpuweb/gpuweb/blob/main/proposals/buffer-view.md
iv. - last time I tried it, tooling sucked

d. Slang:
    i. * currently what I use (and wish there was something better)
    ii. + actually has modern language features
    iii. - feels bloated and unrefined (e.g. there are multiple ways on how declare an array or a function and there did not seem to be any guidance on what the preferred way is)
    iv. - documentation is very poor, I very frequently resort to open HLSL documentation in hopes to find the same function there and pray, that it works the same in Slang, also still haven't found where the documentation for SV_* inputs / outputs is
    v. - tooling is poor (to clarify, my reference here is Rust and C# tooling, which is excellent)
e. Rust-GPU:
    i. - although I am using Rust for my privates projects, I am not convinced that it is the right language to be used on the GPU
    ii. - does not look ready at all (requires nightly Rust compiler)"

7. "GLSL could use some small quality of life improvements (like parameter passing). SLANG is nice, but requires rewrite of whole codebase."
    a. Comment from slang team: It may be possible to use Slang incrementally with Slang's GLSL ingestion path.
8. I'm just sad about the state of shading languages.

Developer Tools

1. Vulkan Profiles
    a. effort put into vulkan profiles is relatively useless unless we can bring those profiles to app stores or other places
    b. It is difficult to see how well a Vulkan Profile is covered, currently it is only on/off. This makes it difficult to debug Vulkan Profiles.
    c. The Vulkan profiles library is broken, which becomes useless with dynamic function loading.

        i. LunarG comment: specific issue: https://github.com/KhronosGroup/Vulkan-Profiles/issues/734
2. gpuinfo.org
    a. gpuinfo.org while probably unofficial and unrelated is very slow and often unreachable. Any kind of similar site or offline app would be great. To be able to query feature support / limits.
        i. Comment from Sascha Willems: An initiative is underway to improve the performance of gpuinfo.org. You can read more about it from Sascha at this Reddit post: https://www.reddit.com/r/vulkan/comments/1rtdo2v/improving_performance_of_the_vulkan_hardware/
    b. "- gpuinfo.org would be nice to allow giving my own GPU database from a json or something, like which GPUs are there, and what is the percentage of their use. That way I can use the Steam Hardware Survey to see what Vulkan features are actually supported by the market right now.
        i. Comment from Sascha Willems: This is not in plan. It is not desirable to add mark share data to the database in an accurate manner.
3. Memory Allocator on C#
4. Windows development is great
5. Selecting a physical device in a multi GPU setup is somewhat cumbersome, if it's not just ""an integrated and a dedicated GPU"" but two dedicated cards.
6. Multi GPU - interop scenarios are usually a pain due to not being able to choose a GPU in old APIs."
7. I wish the system headers could be factored out of the Vulkan.h header, so the Vulkan headers could be used in global pch without polluting my namespaces with system symbols.

Debugging

8. RenderDoc
    a. RenderDoc/nSight occasionally crash when encountering incorrect Vulkan code without providing any info, for example, on incorrect reads from pointers on the device
    b. Renderdoc does not work with Wayland. I miss something like caniuse to know which extensions are available where.
    c. RenderDoc has limited support for Wayland clients.
    d. Lack of Wayland support in RenderDoc, no cross-platform dma-buf like functionality.

9. Debug and profiling tools needs more features, RenderDoc is nice but also working with DX12 I wish we could have something on the level of PIX (which is not vendor locked like nsight).
10. "For desktop development : Debugging is still the #1 pain point. Everything that can't be easily accessed in RenderDoc requires vendor specific and/or finicky software.
    a. Benchmarking is even worst. Having a real PIX equivalent would go a long way."
11. Tools for Android could be improved, but I think it is largely a problem with the hardware vendors. For example there are devices that are not supported by Android Graphics Inspector.
    a. LunarG comment: Samsung, and LunarG are working on a new generation of android tooling. Sokatoa 1.0.0 was recently released. For more information:
        i. Samsung Launches Sokatoa to Enhance GPU Performance Analysis on Android
        ii. How Samsung Built Its New Android GPU Profiler on LunarG's GFXReconstruct
        iii. https://github.com/sarc-acl/sokatoa/releases/tag/v1.0.0
12. "In vulkanized 2026 there was a lot of discussion about debuggers and profilers, yet no one seems to mention that none of them work on linux. And I just tried them again. On ARM the UI simply freezes and the only button that works is the quit button. On intel you have to manually install kernel modules, but if you're not running exactly the correct kernel version you're out of luck, which is not even the most recent one so supporting the most recent hardware. And for Nvidia and RenderDoc the UI works, but you can't capture the app. For example Nvidia simply says:
    a. Launched process: vkmark (pid: 257183)
    b. Attempting to automatically connect…
    c. Searching for attachable processes on localhost:49152-49215…
    d. Launch process exited. Searching for attachable child processes…
    e. Searching for attachable processes on localhost:49152-49215…
    f. Failed to connect. The target process may have exited.
    g. And no window has ever spawned. Part of me is disappointed that you can run the latest AAA games without any flaw, yet running a debugger is out of reach. The other part of me is disappointed because I hate the dishonesty, I would be fine if they would say ""We don't care about Linux for any other purpose than compute and if you use the wrong API (anything besides DX12) and wrong OS, we don't care about your problems, because if you had real problems you would switch to a real OS and real API."" but they don't, and this dishonesty is what makes me angry."
13. 3. Easier Runtime Profiling inside Vulkan
14. GPU crashes are annoying and difficult to debug
15. VK_DEVICE_LOST followed by black screen is my most common VK error and not a good experience. I would love if the driver didn't just crash, but rather failed with an error at least generally describing what's wrong.

Vulkan on Metal

LunarG comment: MoltenVK (MVK) is not sponsored for ongoing defect fix and enhancement beyond passionate developers who sporadically do it on their personal time. LunarG is not fixing or enhancing MoltenVK as well because KosmicKrisp is the planned future for Vulkan on Metal. There are users of MoltenVK who can't/won't adopt KosmicKrisp and hence why we will continue to include it in the macOS SDK.

1. I personally do not care for KosmicKrisp/MoltenVK as I think it is generally always better to implement the vendor recommended API (Metal, NVN) if it provides better performance.
2. "- macOS is simply a non-objective for us, because we depend on tech stacks that are simply not available on this platform (namely, CUDA).
    a. - However, I believe having a common stack on all main platforms is a critical task and follow closely what are done in this direction outside my work time.
3. Desktop development for macOS
4. The adoption of new extensions in MoltenVK is very slow. Almost three years after hardware support was added, I still cannot use ray tracing extensions on macOS, and this is making me consider using Metal directly on macOS and iOS.
5. Infrequent issues with swapchain synchronization on MacOS (may be resolved with newer KosmicKrisp)
6. macOS development with MoltenVK has been a bit of a pain with bugs causing errors with buffer lifetime and hard-to-find device lost errors. Looking forward to trying out Kosmic Krisp in more detail, to see if that works better for our purposes.

Miscellaneous

1. - I watched a talk about how they ported Ghost of Tsushima to PC (https://www.youtube.com/watch?v=tostM4-rK6o), and they showed a bit of what shaders they had on PS5, which they needed to change for PC. They were able to just put a texture (Texture<float4>) inside the struct, which they passed from the CPU. That would be one of the best way, although if you only support hardware support descriptor buffer and descriptor heaps, you could emulate this by registering every texture in a global heap/buffer and pass indices as handles. Then always have the global texture heap/buffer bound.
2. We have no any reasons to support legacy GLSL/OpenGL.
3. I love hearing about features that could potentially make code more performant and or convenient, eg. bindless textures and VkShader
4. Плохо обновляемые образцы, условно версия 1.3, мало примеров и использование расширений - (translated to: Poorly updated samples, conditionally version 1.3, few examples and use of extensions -)
5. "SPIR-V for Vulkan is really limiting compared to OpenCL, the addressing model is extremely fragile leading to fragmented capabilities acrosss Storage Classes.

Furthermore lack of forward progress guarantees leaves us to rely on and abuse black boxes like WorkGraphs and RT Pipelines for Dynamic Parallelism which have inconsistent availability or quality across vendors.

6. LLVM-pipe should be beyond WARP levels of Vulkan coverage, would be really nice to use that for CI on free CPU VPS runners.
7. Generally speaking CI for Vulkan is a bit of a wild-west."
8. "1) i never have be able to use the pipeline compilation library extension for accelerating shader compilation time, expecily for looooooong rtx raygen shader, that take several minutes to compile whith deferred compilation enabled, and mush less for the next times with a cache, but i never be able to use this library.
9. 2) i my opinion, many things in term of good performance choice, good with to do, is, as an advanced developper, hard to imagine. thanks to claude code for learning te me many things, that i should have learn in a better way
10. I no longer use Vulkan. Switched to D3D12.
11. "I use the Qt crossplatform framework as the foundation of my program. Qt's Vulkan implementation is not well documented, so that was a pain when I started out.I have now written my own version that renders Vulkan inside a QWindow, using pure Vulkan, not the wrapped Qt functions.So now I am fine."
12. OpenGL is different from Vulkan (low-level). There is no bird's eye-view map to see where is what. Different graphics experts write their code completely differently from each other i.e. Professor A writes a different set of code from Professor B
13. It is not possible to render whole scene with one vkDrawIndexed() call if my indices are scattered over 32GB of GPU memory. I would like to have support of 64-bit indices in vkDrawIndexed() call.
14. No pathway for learners
15. "The usefulness of Vulkan as a HAL is routinely undermined in online discussions by people who insist that NDA platforms, usually game consoles, cannot support Vulkan. This is used to support arguments for writing DirectX12 and Metal paths.
16. I have no visibility or direct interest in NDA platforms, but I have a strong interest in open standards and this has been a consistent talking point against Vulkan for many years."
17. I'm really looking forward to descriptor_heaps and rust-gpu becoming more powerful.
18. personally, I find the extension mechanism of .pNext stuff bizarre, maybe its because im c++ dev, and it doesn't bother me too much as I hide it behind abstractions.
19. I think that vk.xml can be improved a lot by first changing it to another format and second, by stripping unnecessary data. By unnecessary data I mean stuff like VK_*_MAX_ENUM being something mandatory to generate when parsing. Also all of the VK_SOMETYPENAME_ACTUALLYUSEFULENUMNAME is unnecessary in my opinion. An enum should just contain the name, the type(u32, s32, etc.), the enumerates without any namespacing(GENERAL instead of VK_LAYOUT_GENERAL) and thats it. It will greatly improve bindings generation for other languages(even for c++).

General Positive Feedback & Gratitude

1. "I don't have many complaints.
2. The tools and documentation are great.
3. Thank you for everything you do, I appreciate you.
4. So far so good!
5. My main feedback is that I really love Vulkan and its ecosystem. It's an absolute pleasure to use, especially when compared to Metal. I love the spec, I love the IHV support and that everyone clearly loves Vulkan as well and is deeply invested in it. It's an absolute pleasure to use, so thank you very much for it!
6. Thanks everyone for all their hard work and thanks to the sponsors for making it freely available to everyone.
7. Overall, I'm very happy with Vulkan development.
8. Just learnig
9. 3) love you guys, vulkan is so great ! thanks !"
10. idk it's a pretty smooth experience overall. y'all's have done great work
11. Keep up the good work.
12. Validation layers are amazing. Thank you Spencer.
13. Also just want to say that its been clearly showing that you guys care about improving the experience and I appreciate all the effort!"
14. but everything else is pretty awesome. I really appreciate the work that LunarG does.