LunarG 2024 Ecosystem Survey Report

Executive Summary

Methodology:

1. LunarG developed this Vulkan ecosystem survey with input from Khronos developer relations and Google. The purpose is to gauge the Vulkan community's use of and satisfaction with the current Vulkan ecosystem.
2. It was attempted to reach as many Vulkan developers as possible -- both SDK users and non-SDK users. The survey was advertised on Twitter, Reddit, LinkedIn, the Khronos Vulkan slack channel, Vulkan Discord, and sent directly to 13,000+ recipients of the LunarG Vulkan SDK mailing list. It was amplified by Khronos on their twitter account and newsletter mailings as well.
3. All comments from open-ended questions are included in this report (at the end of the report), regardless if they are repeated. This helps you see the frequency of certain types of feedback.

Highlights:

1. There were 258 respondents.
2. 47% of the respondents use Vulkan for commercial purposes. 52% of the respondents were self-studying Vulkan as part of a personal project or an Academic environment (non-commercial). Throughout the report, the data is shown for Non-commercial and Commercial developers, side by side.
3. 70% of the respondents are regular, advanced, or expert Vulkan developers. 30% are basic or beginner developers. Hence the feedback is coming from a more experienced population.
4. Both macOS and Android as a target of applications have more commercial developers targeting these platforms than non-commercial developers. This is a change from last year where more non-commercial developers were targeting those platforms.
5. Some Year over Year insights:
    a. Windows 11 has surpassed Windows 10 as a target platform for applications and as a development environment.
    b. Compared to the previous year's survey, the number of folks who have released their Vulkan application for public use has increased from 36% to 42%
    c. glslangValidator (glsl->SPIR-V) or shaderc (glsls->SPIR-V) remains the most used front end from which to generate SPIR-V. This is the same as last year.
        i. More commercial developers are using DXC-> SPIR-V or glslangValidator (HLSL->SPIR-V) than non-commercial developers.
6. Validation Layer themes:
    a. People rely on the validation layer and are satisfied and happy we keep making it better
        i. More people are trying to fix their validation errors in their application and trust there are no bugs in the validation layer
        ii. Most people report issues to github (instead of just ignoring it). This is because we are responsive to the github issues.

iii. Performance is an issue primarily for the non-hobbyist
iv. Best Practices can be a lot of spam. Core problem is that we are not investing in defining its scope, role, and improving it.
v. DebugPrintF is used a lot, but has not been maintained at the same level as GPU-AV recently.
vi. People really rely upon synchronization validation. The hard work from LunarG investments is greatly appreciated. But we still have a lot more to do to fill out the functionality and help with performance.
    1. Timeline semaphore is in high demand.
vii. Error messages can be very verbose, poorly formatted, or poorly worded.

b. Continue to increase validation layer coverage
    i. Specific complaints about the ray tracing extension missing a lot of validation.

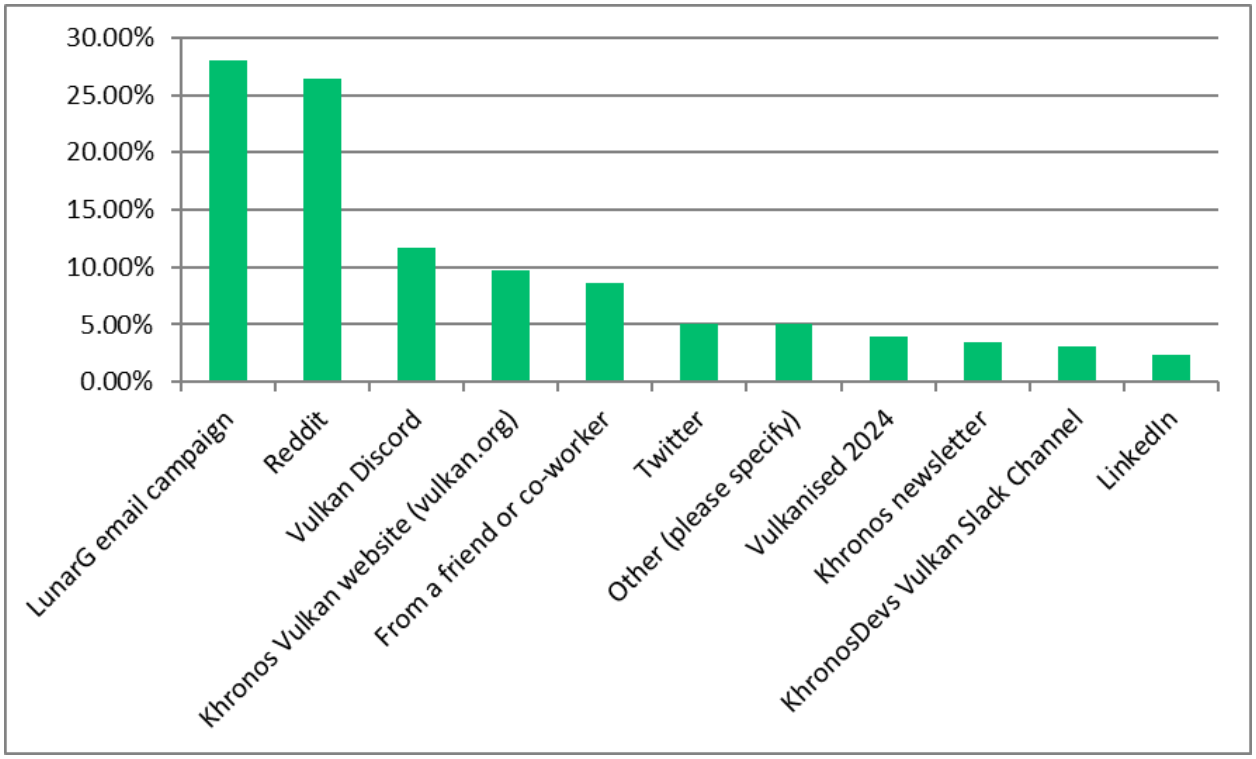
7. Specific complaints about mesh shaders not having required spirv-tools validation: https://github.com/KhronosGroup/SPIRV-Tools/issues/4919
    a. "More SPIRV validation. For example the EXT_mesh_shader validation in spirv-tools is not complete and hasn't progressed for months. Generally the spirv-val seems abandoned"
    b. LunarG comment: We have brought this to the attention of the responsible parties and development is resuming.
8. Continued concerns about the shader toolchain (better options, next generation languages needed, DXC has a complicated code base with bugs, need to improve HLSL support for Vulkan).
9. Continued concerns about the complexity of the Vulkan API (The Khronos Vulkan working group has access to the feedback data)

Potential actions being considered for the year to come

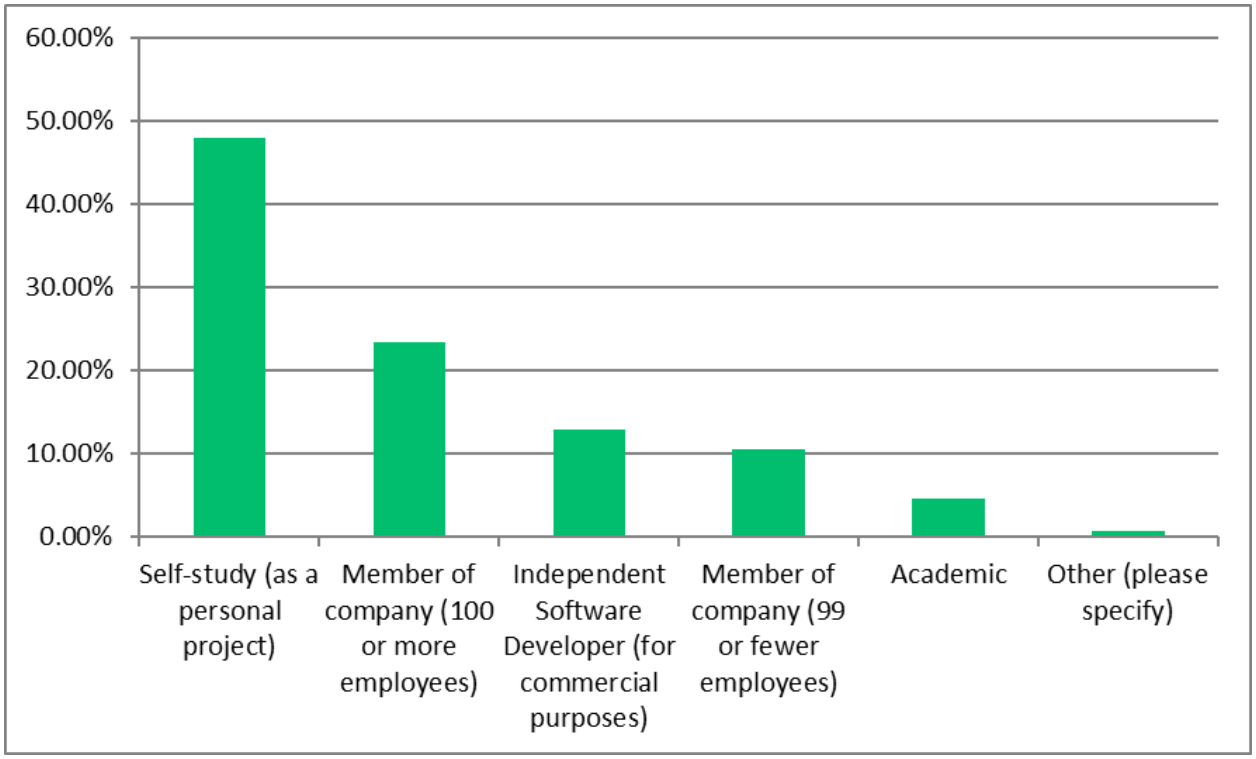
1. Validation layers.
    a. Continue GitHub issue responsiveness and filling out coverage
    b. Continue improving validation layer error messages.
    c. DebugPrintf needs maintenance and improvements
    d. Best Practices needs improvements
    e. Fill out more functionality for synchronization validation (timeline semaphore)
2. SDK additions for consideration
    a. Make Windows 11 support official (add it to CI across repositories and test with SDK)
    b. Add GLFW
    c. Add an SDK for ARM64 Windows and Linux
3. Tooling improvements for consideration
    a. Better support for ray tracing debugging (RenderDoc, validation layers, GFXReconstruct, …)
    b. Have validation layer support for a new API available with the API release (or at least much more quickly)

i. Vulkan Working Group comment: The workflow for an API today is specification and CTS first, validation layer development 2nd. It is typical that the validation layer development by the working group doesn't begin until after the API has gone public. The working group is in the process of modifying their workflows to require validation layer development to happen sooner for a new API.

Where did you hear about this survey?



What type of Vulkan developer are you?

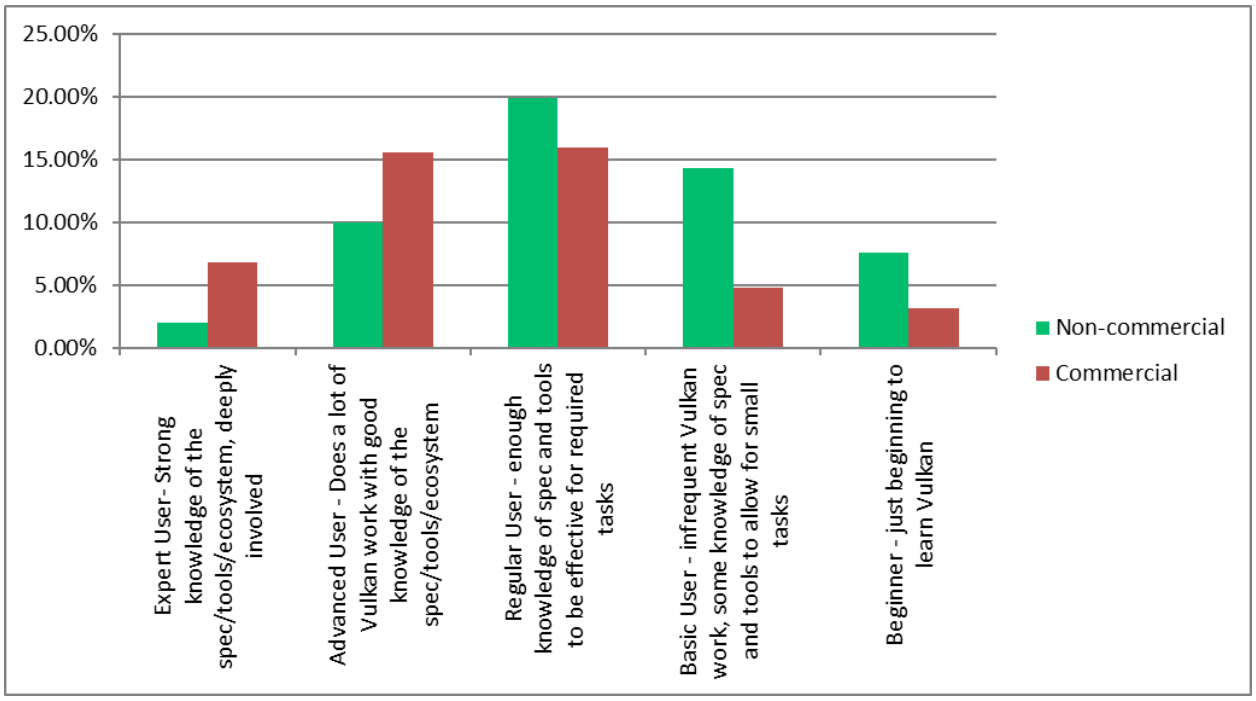


Other:

- For open source project (http://nice.graphics)
- Open Source Scene Graph developer

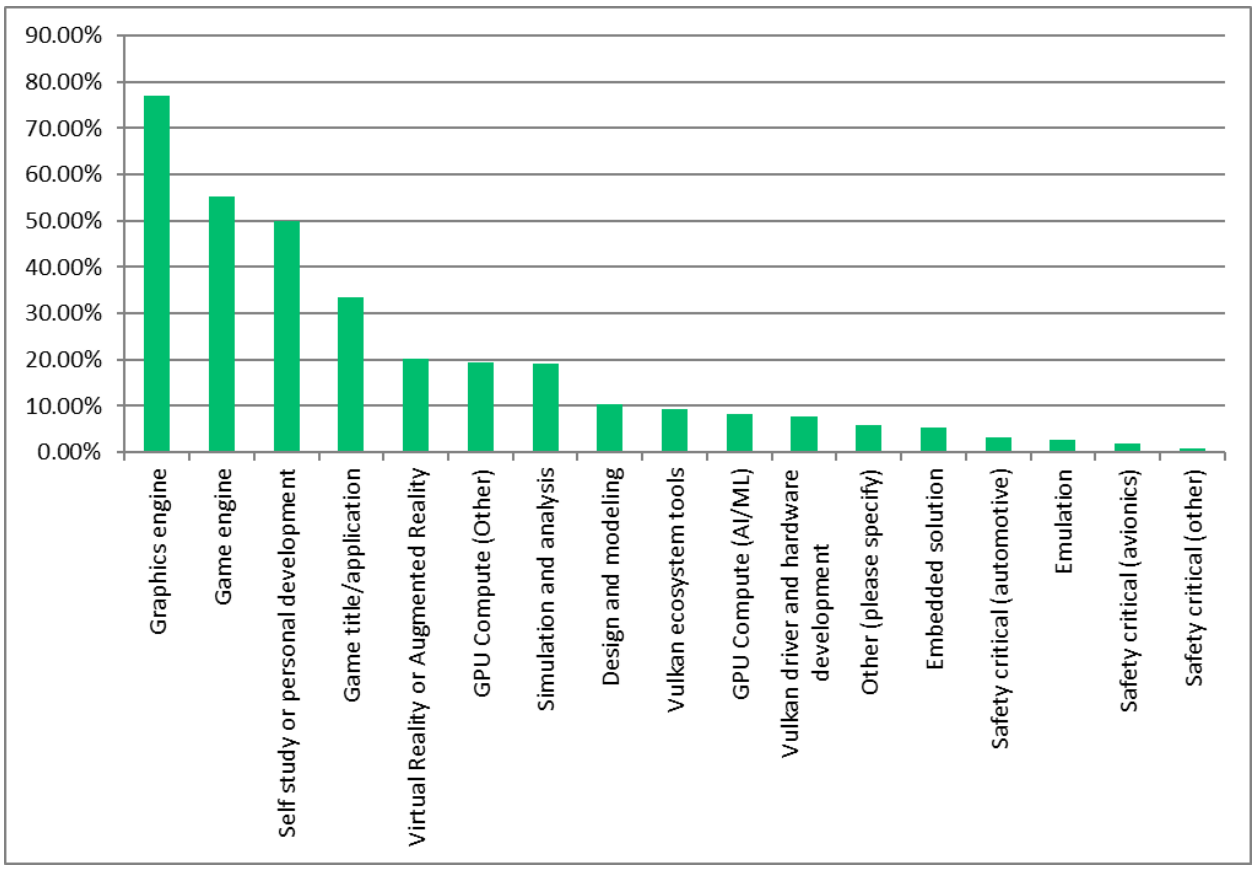
Developers for commercial purposes are 47%. Developers for personal projects or academic purposes is 52%.

How experienced of a Vulkan Developer are you?



70% or respondents are Regular, Advanced, or Expert developers.
30% are Basic or Beginner developers.
More of the commercial developers have higher levels of Vulkan experience.

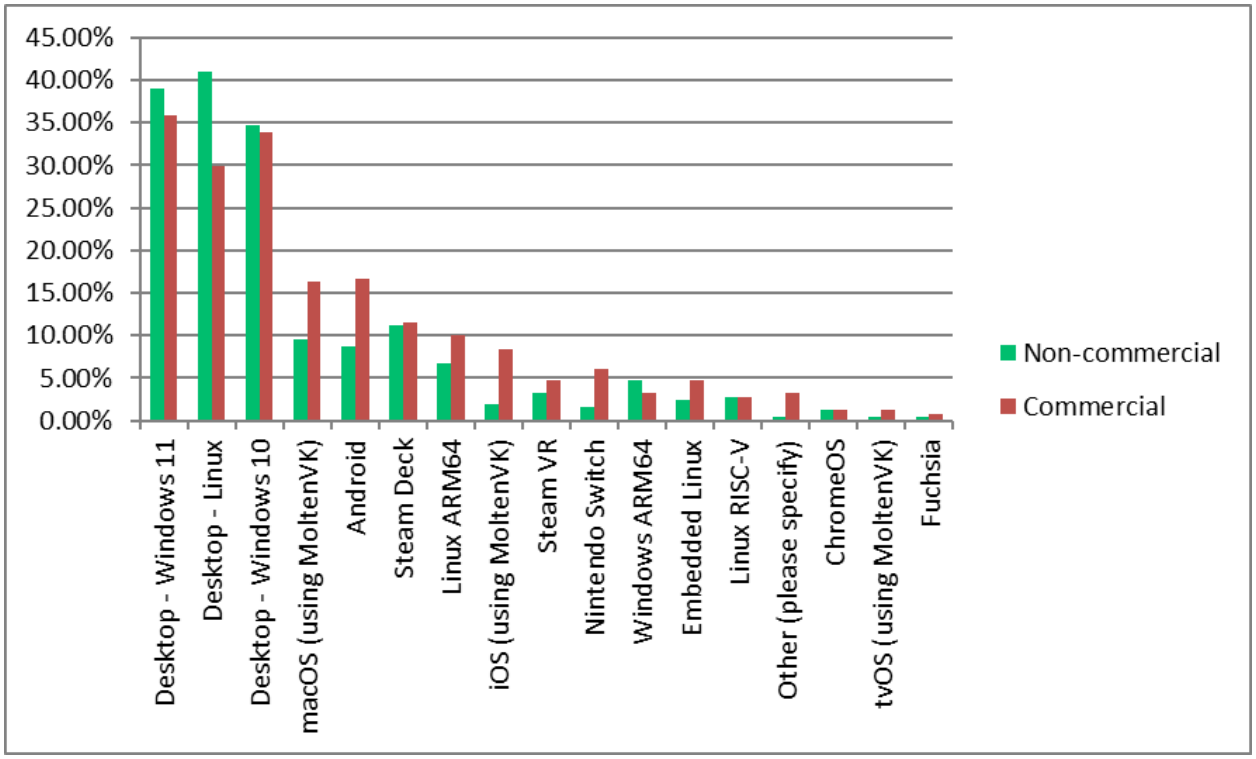
Your development is for what type of use case? (check all that apply)



Other:

1. Language bindings
2. CAD
3. Metaverse
4. Vulkan Layers
5. Ray Tracing based DCC tools
6. Media and Entertainment
7. Web browser, Dawn
8. Video coding acceleration!
9. VulkanSceneGraph
10. Ki
11. Media server
12. 3D map for navigator
13. Scientific visualization
14. Medical data visualization
15. VulkanSceneGraph

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

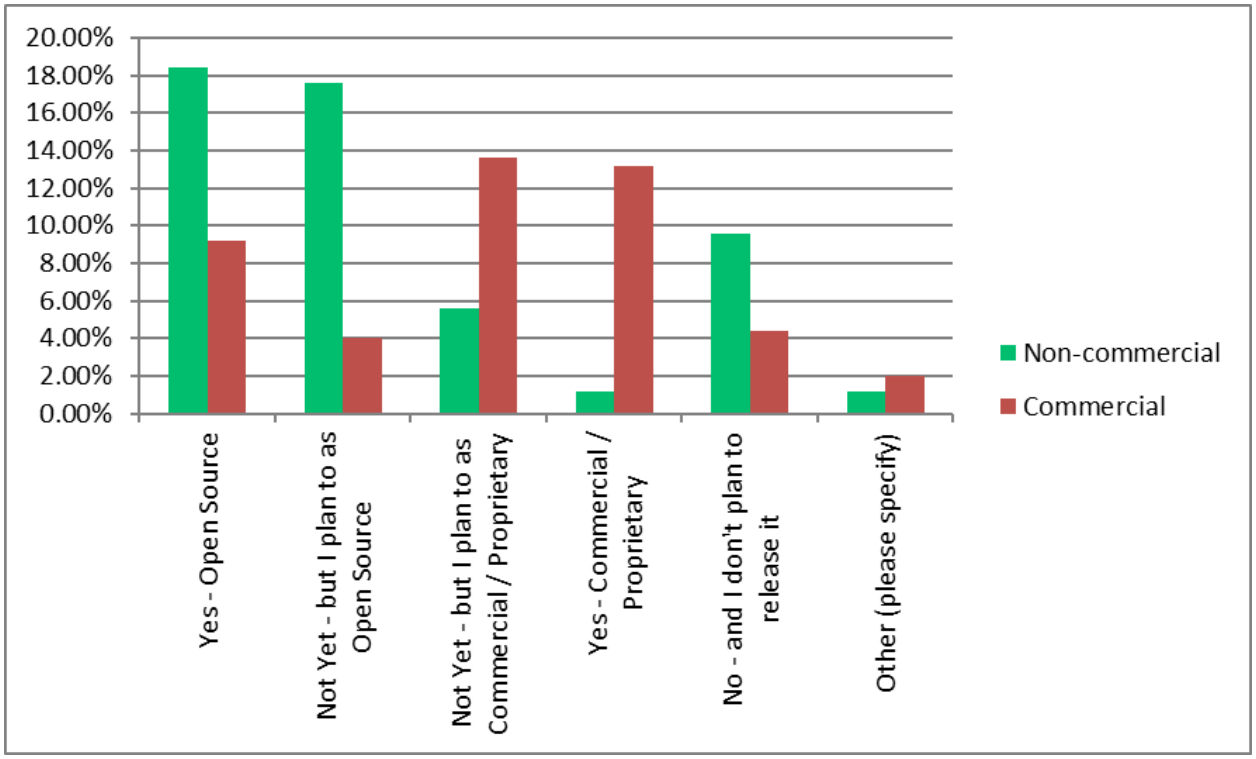


Other:

- headless linux
- Custom RISC-V
- non-desktop Linux
- RTOS
- QNX
- BSD Unix(GhostBSD)
- Open Harmony
- macOS Apple Silicon (using MoltenVK)
- I've designed the system to be able to support many platforms, but I haven't dropped or tested any but windows.

Observation: Much of the CI on the SDK repositories is not for Windows 11. However the SDK and repositories are working fine on Windows11. Probably a good idea to begin upgrading test suites to run on Windows 11 in the SDK repositories.

Have you released your Vulkan development project for public use?

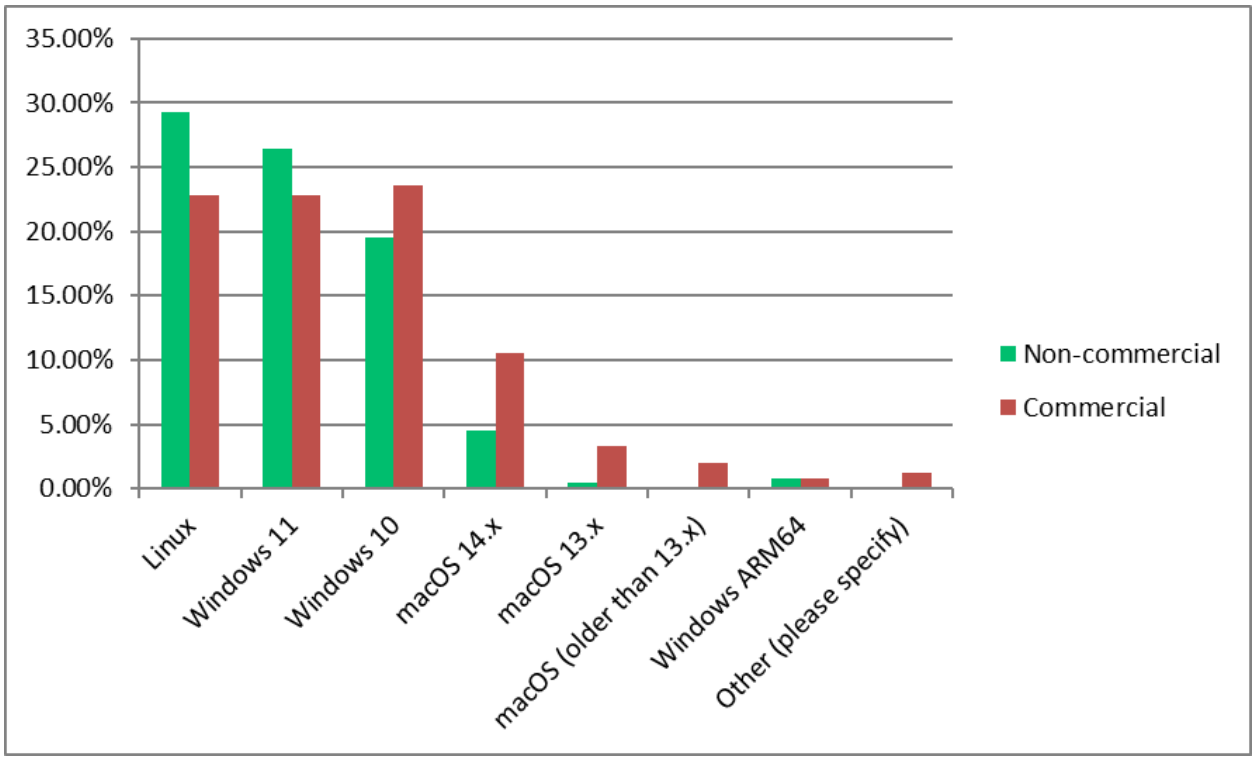


Other:

- Released small tools as open source but not the main project
- No, but I would if given permission
- One project is on GitHub, the other unreleased without any plans at the moment
- No, unknown if it will be released. If it is, it will probably be open source.
- No and I did not think about that
- No plans thus far as no project has reached any stage where it would be useful - everything is mostly a prototype for studying.
- Not yet, and we shall see

Percentage of release content: 42% (36% last year, 28% year before that)
Percentage of content planned to be released: 41%
Percentage on content not to be released: 14%

If you are developing Vulkan applications for the desktop, what is your development environment? (check all that apply)

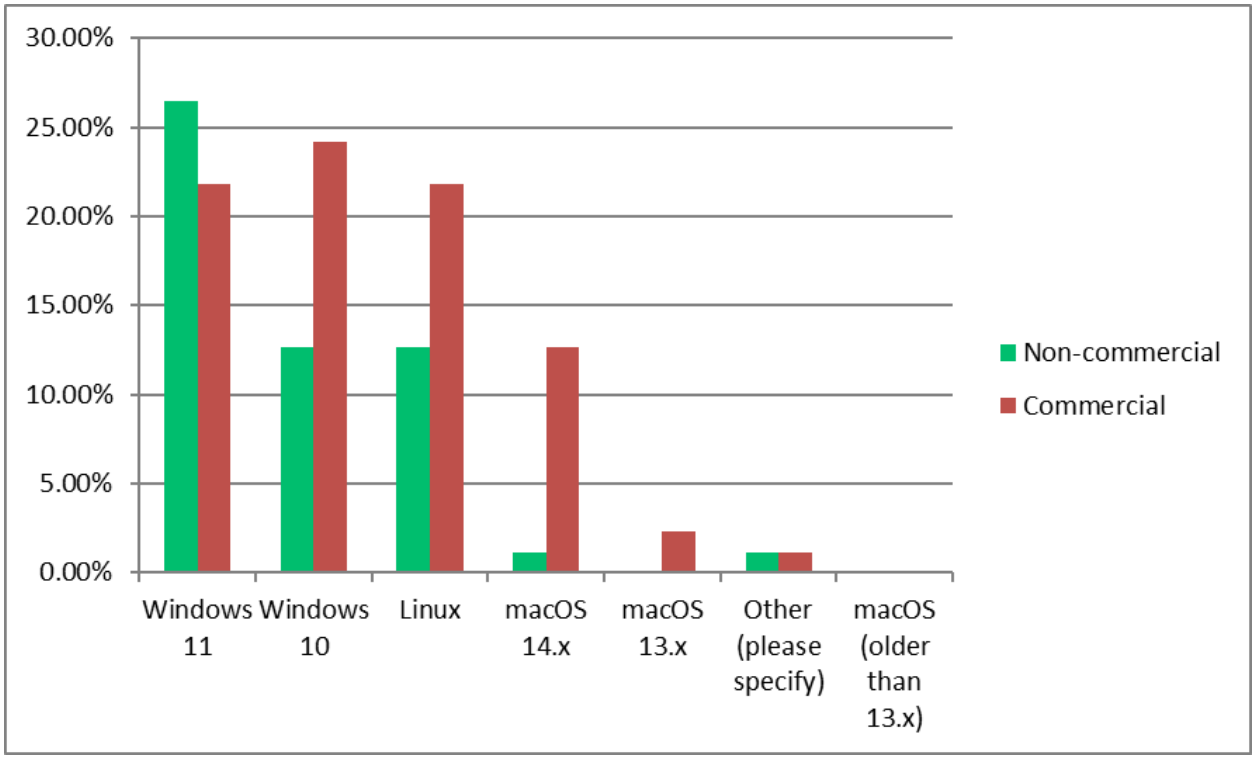


Other:

- Wine
- Linux ARM64
- not this part

Still a lot of Windows 10 development environments, especially for commercial developers.

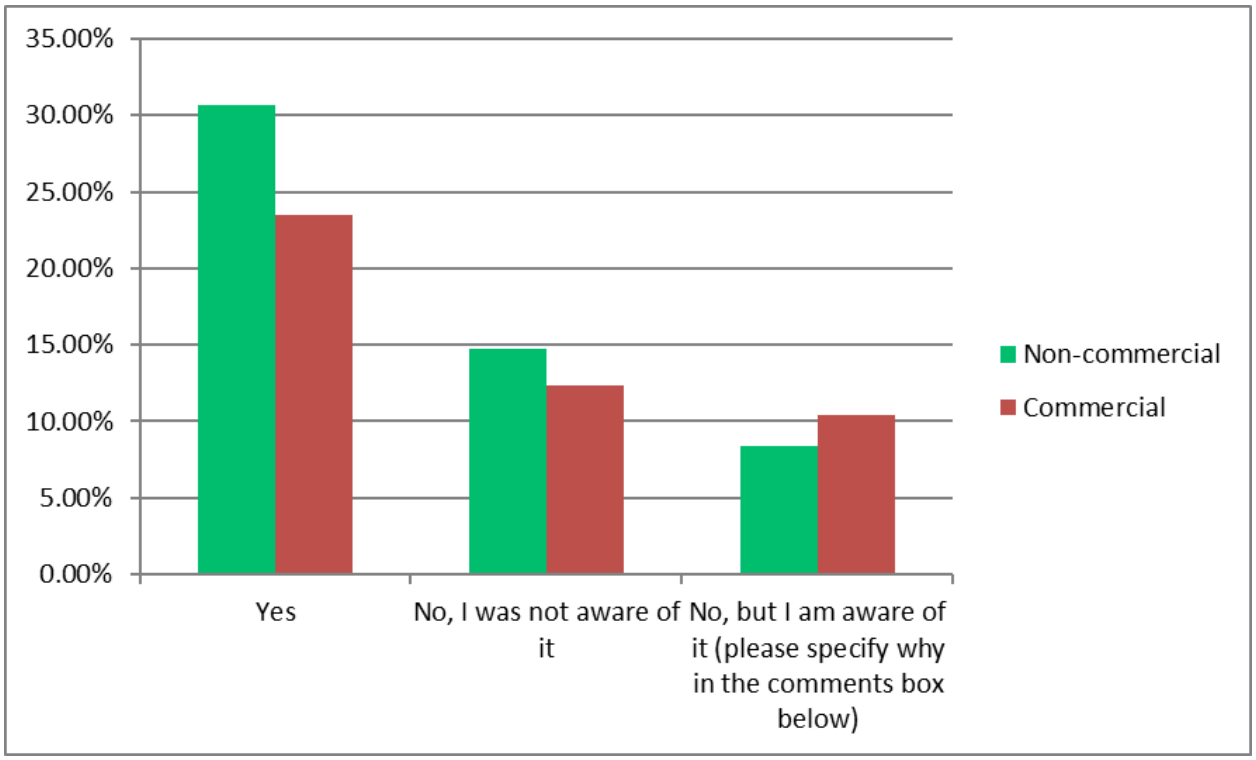
If you are developing Vulkan applications for Android, what is your OS development environment? (check all that apply)



Other:

- not applicable
- not this part

Do you use the new docs.vulkan.org site?

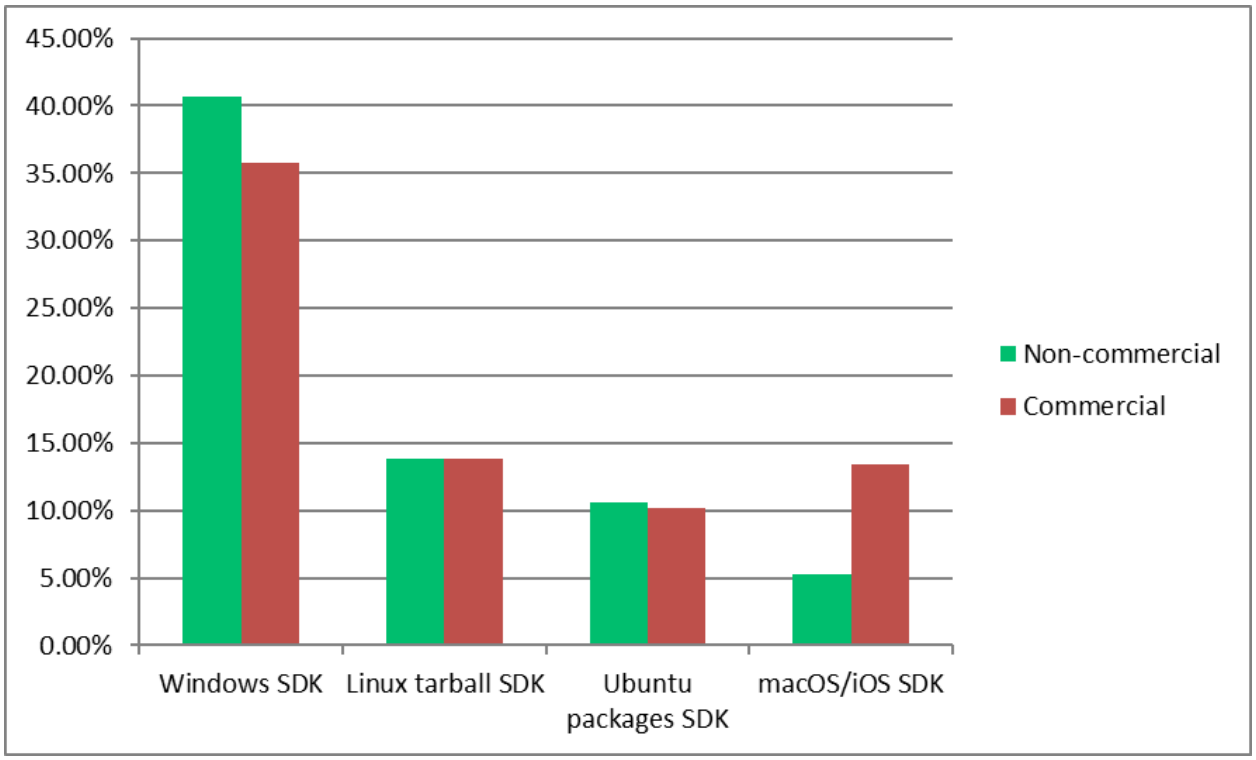


Reasons why you don't us it (please specify)

1. I mostly use the PDF spec
2. Google searches of API functions brought me to registry.khronos.org. I will give the new site a try.
3. It’s not easy to navigate. There should be more examples depending on what you want to achieve with accompanied code and most importantly an image of the output. More visual examples in general
4. I still use with the single-file vkspec.html, that one works perfectly well for all of my usecases.
5. When I need to look up references for a certain struct or function description, typing its name in the search engine usually gives registry.khronos.org as the first result.
6. I'm used to https://registry.khronos.org/vulkan/specs/ and it comes as the first result when googling
7. I tend to Google extension names, and the first result is always to the manpages on registry.khronos.org with internal links that point to the full spec.
8. I don't use it directly, because generally https://registry.khronos.org/vulkan/specs presents it in a more compact and direct way for my uses
9. Keep forgetting and not in my "goto" bookmarks. I will fix that.
10. I forget :)
11. I think I used it when it had better navigation

12. It doesn't come up first on google, and it doesn't really seem different than the other vulkan spec info.
13. I am using pdf doc. This way, I do not need internet connection. The docs.vulkan.org does not seem to bring any benefits over pdf yet. At least I am not aware of any.
14. I use the SPIR-V and Vulkan specs in PDF form because of the excellent performance over the HTML versions
15. too much bloat when coming from the old spec docs
16. Just goodle old spec when needed some-thing specific
17. I simply Google
18. I'm still using the https://registry.khronos.org/vulkan/specs/1.3-extensions/html/ site, haven't tried the new site yet.
19. Was not referenced in vulkan-tutorials.com, vkguide.dev, or in the Vulkan discord server as a resource for beginners..
20. Many existing links to old site
21. i only needed to look up something quick so far and google still shows the old documentation at the top
22. Currently the Vk dev work is on hold
23. I still use the PDF out of habit. I may use it in the future.
24. Can't access new site from Russia, but can freely access vulkan.org. Please fix. :(
25. I'm confortable with the spec
26. It is harder to find specific structs that are in the spec
27. I have the habit to use the specification now, but I will probably use it more and more with time.
28. The PDF specification works quickly regardless of internet connectivity

Do you use the following Vulkan SDKs? (check all that apply)

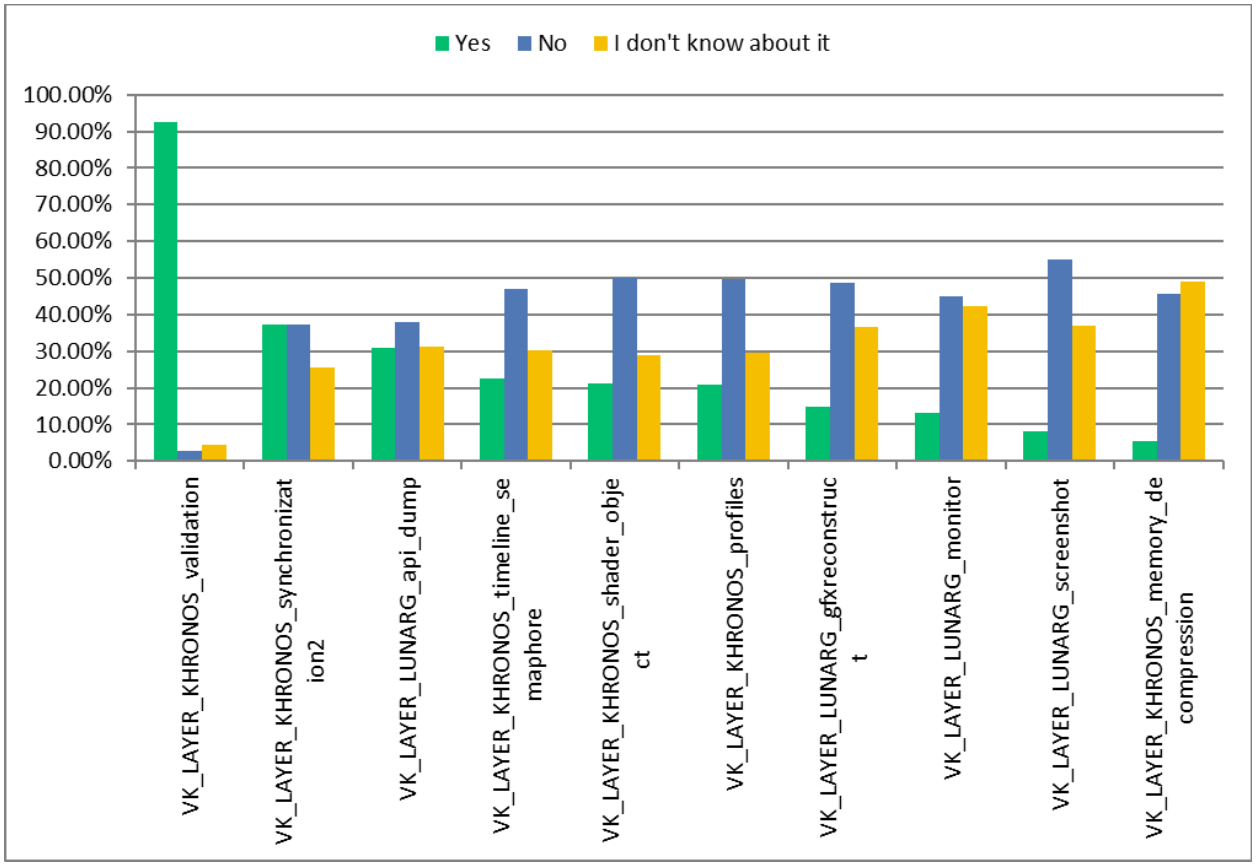


1. I don't use the Vulkan SDK or I use the following (please specify):
2. Vulkan core
3. vulkan-devel Arch Linux packages
4. I use our own in-house SDK
5. Some mishmash of packages provided by Arch Linux and self-compiled tools as necessary to get support for the latest extensions
6. I use the Rust crate vulkano, currently switching to ash. I did not specifically download an SDK.
7. Debian packages
8. arch distro packages
9. vulkan-devel on Arch Linux
10. arch linux packages
11. I install the offical packages in arch repos
12. arch packages
13. what?!
14. The Vulkan tools are built from source.
15. Build components ourselves.
16. I used whatever is available for Android and/or build from source
17. Debian packages SDK
18. debian package
19. want using it in our project

20. I'd like to use the macOS sdk but MVK never has a release that doesn't crash so i always need to manually update mvk, fix it's source and build from there.
21. Also, there's no static loader in the macos sdk so i dont know how to make it match with Windows' vulkan-1.lib setup.
22. Arch Linux via pacman
23. Sometimes I just use Fedora packages, e.g., vulkan-headers, vulkan-loader-devel, vulkan-tools, vulkan-validation-layers, glslvalidator
24. Packages by the distribution
25. I get drivers and libvulkan from distro packages. SDK not as useful for non-C/C++ users (e.g. Rust).
26. whatever is provided by the arch linux and aur if i need a bleeding-edge sdk
27. Don't use
28. I build what I want from source
29. dynamic loading
30. I build what I need from source

Which of the following Vulkan layers do you use?

Answered: 253
Skipped: 5



Which of the following Vulkan Tools do you use?

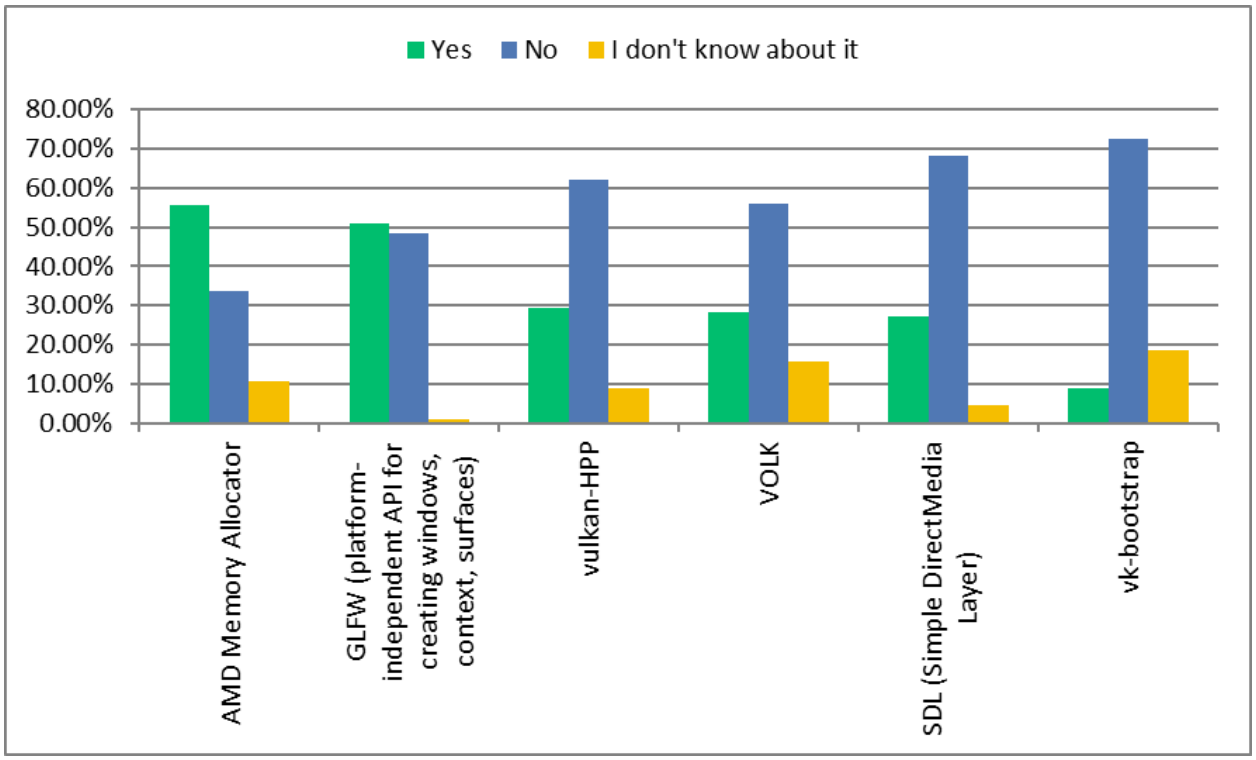
Answered: 250
Skipped: 8

Other:

1. using environment variables directly instead of vkconfig
2. RenderDoc! Nsight for profiling, Aftermath for GPU crash debugging. For Android development I use Mali Streamline for profiling, and the VkResult tells me when the GPU crashes
3. RenderDoc/nsight
4. RenderDoc
5. RenderDoc
6. Android Graphics Inspector (AGI) and Android's "vkjson" and RenderDoc
7. spirv-val, extensively
8. RenderDoc works fantastically well on windows and android
9. Renderdoc
10. NVidia NSight, RenderDoc
11. renderdoc, nvidia nsight graphics, pvrtexturetools

Which of the following libraries do you use for your Vulkan development?

Answered: 245
Skipped: 13



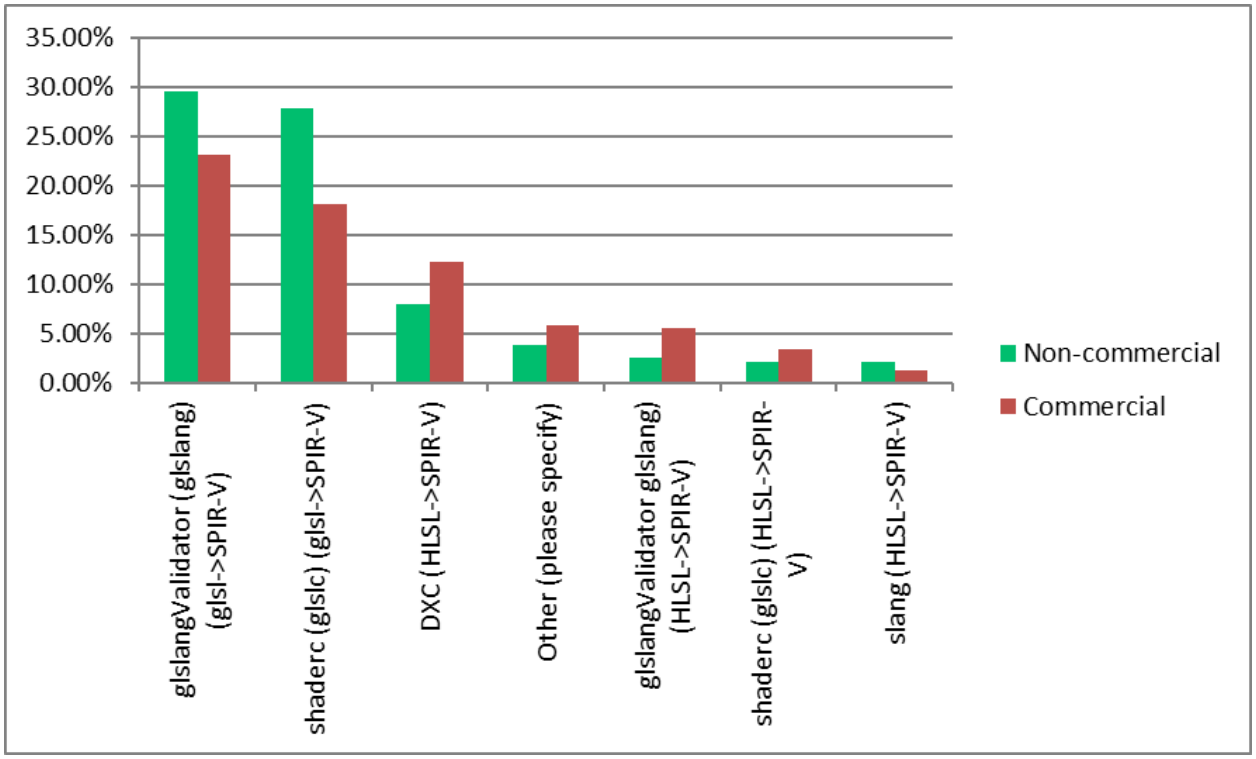
Other:

1. vulkanscenegraph
2. VulkanSceneGraph
3. VulkanSceneGraph
4. VulkanSceneGraph
5. VulkanSceneGraph
6. VulkanSceneGraph
7. VulkanSceneGraph
8. VulkanSceneGraph
9. VulkanSceneGraph (https://github.com/vsg-dev/VulkanSceneGraph)
10. VulkanSceneGraph
11. glm
12. glm
13. GLM
14. glm
15. GLM
16. glm
17. vulkano
18. vulkano

19. Vulkano
20. Imgui
21. Dear ImGUI
22. spirv-reflect
23. spirv-reflect
24. libKTX
25. KTX
26. libktx
27. Tracy
28. Slang (shading language),
29. ash
30. Rust: ash,
31. Ash
32. Ash (in Rust)
33. ash_window
34. lightwieght vk
35. Winit
36. winit
37. winit (rust)
38. github.com/Dav1dde/glad (webservice for glad2)
39. vuk
40. I don't use a specific library.
41. VSG
42. My own engine
43. Silk.NET
44. I experimented with vulkan-HPP and found it to have multiple major problems. The biggest was that it seriously impacted build times and was painful to debug with.
45. unreleased spirv parser/reflection tool.
46. GLFW/SDL in C++ land.
47. spirvcross
48. tinygltf
49. Own libraries for learning purpose
50. stb
51. WINAPI
52. glad
53. XCB, win32

What is your front end for creating SPIR-V? (check all that apply)

Answered: 243
Skipped: 15



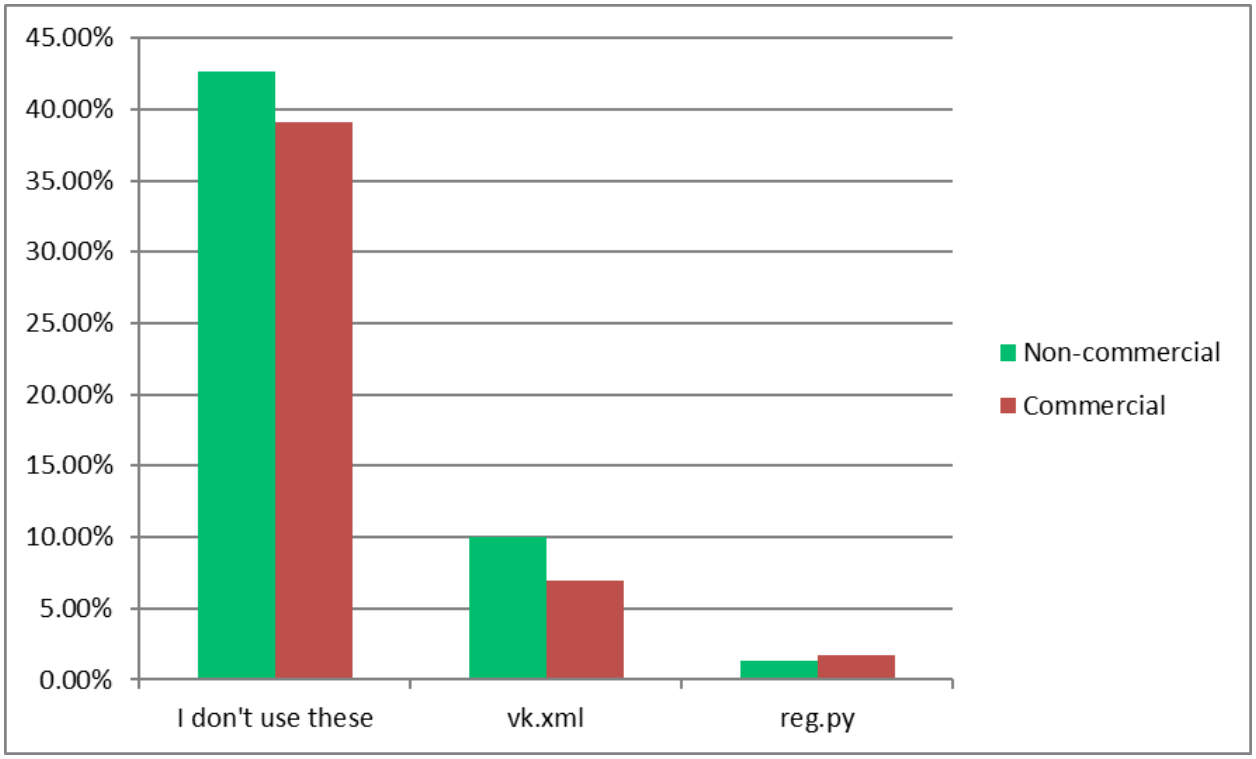
Other:

1. Custom C++ frontend -> LLVMIR -> MLIR -> SPIR-V
2. I don’t know about that
3. vulkano_shaders:shader!
4. naga
5. Naga (wgsl->SPIR-V)
6. NZSL
7. Tint (WGSL-> SPIR-V)
8. Tint (WGSL->SPIR-V)
9. WGSL
10. ANGLE (glsl->SPIR-V)
11. I'm the maintainer of shady and Vcc
12. clspv
13. self-built
14. also use spirv-reflect to extract cpu-facing structure definitions for UBOs, PCBs, vertiex formats, and specialization constants.
15. rust-gpu (partially)
16. rust-gpu
17. Naga
18. ShaderWriter

19. SparkSL
20. Custom WIP shading language
21. My own compiler
22. glslc is what I use.
23. VulkanSceneGraph uses the glsang library to compile GLSL to SPIR-V at runtime when required.
24. Very interested in potentially adopting vcc (https://xol.io/blah/introducing-vcc/) as it matures...
25. I also use spirv-cross-reflect to extract shader bindings, etc

From Vulkan-Headers, which do you use for parsing the API? (check all that apply)

Answered: 236
Skipped: 22

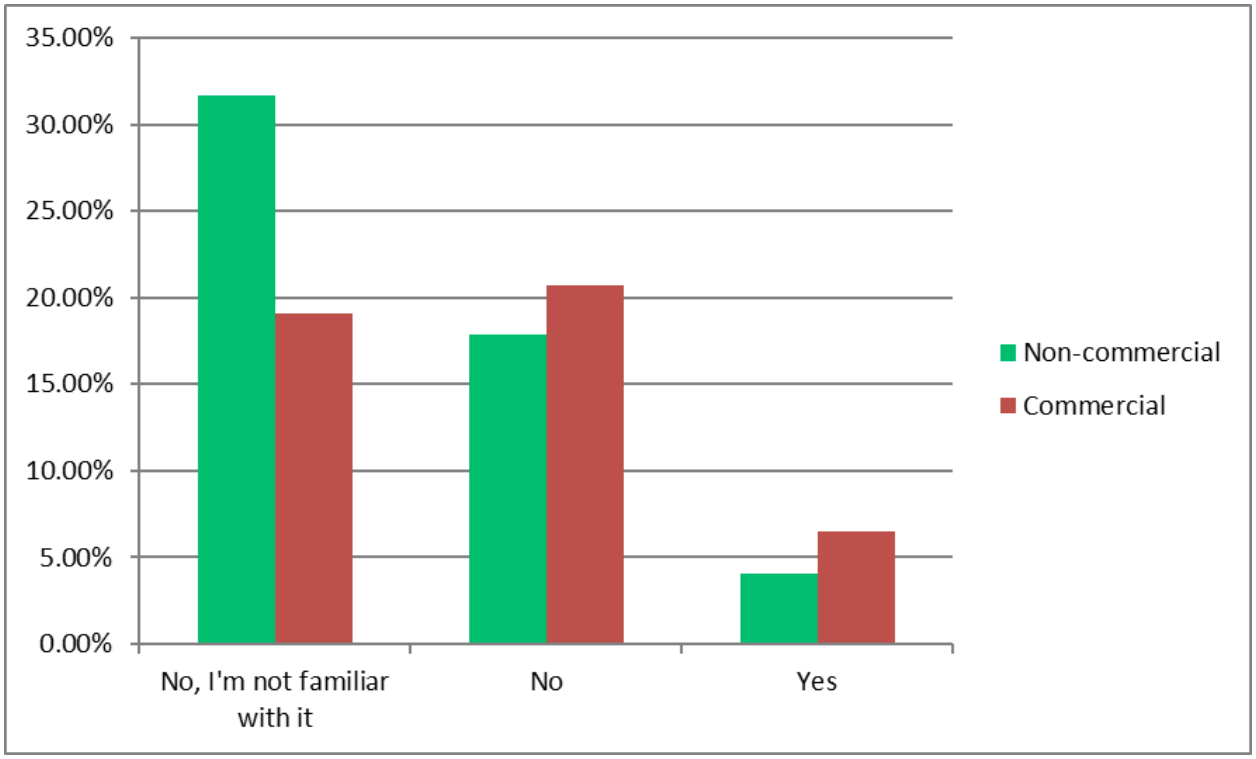


Please specify what you do use (if any)

1. My own
2. I search the core.h and .hpp files.
3. so vulkan.hpp which is generated for us
4. vulkan.hpp
5. I use spirv.core.grammar.json
6. use volk, doesn't think how exactly ot works
7. spirv-headers specs for auto-generating a parser/reflection library
8. Ash
9. I use Vulkan_core with other platform VkSurface specific
10. i use the ones that come with the repo, so i dont generate them myself
11. CMake 3.21+ Linking for headers from SDK and VK_NO_PROTOTYPES
12. I don't know what this does.

Do you use the Vulkan Profiles Toolset?

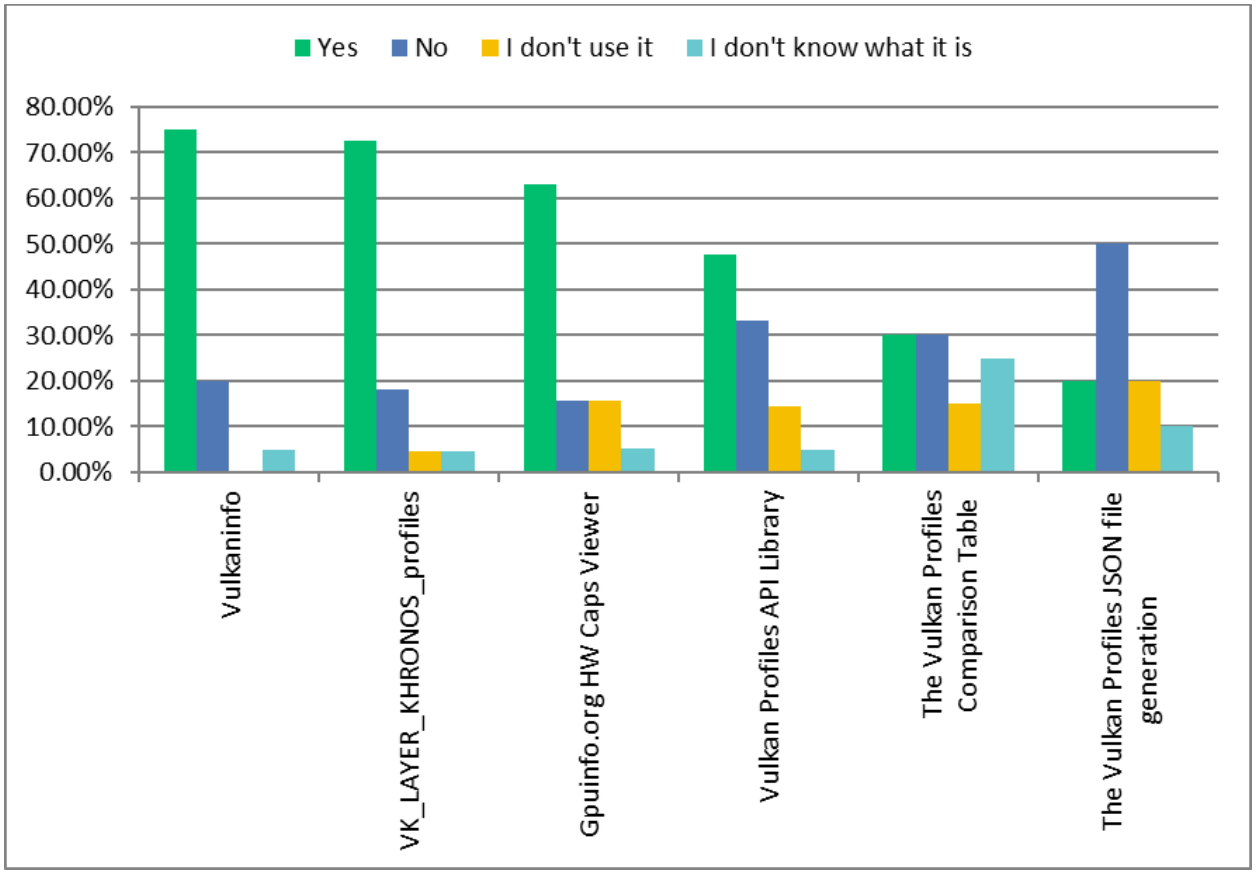
Answered: 252
Skipped: 6



Many people are not yet using the profiles toolset. Slightly more commercial developers are familiar with the tool and using it.

Which of the Vulkan Profiles toolset tools do you use?

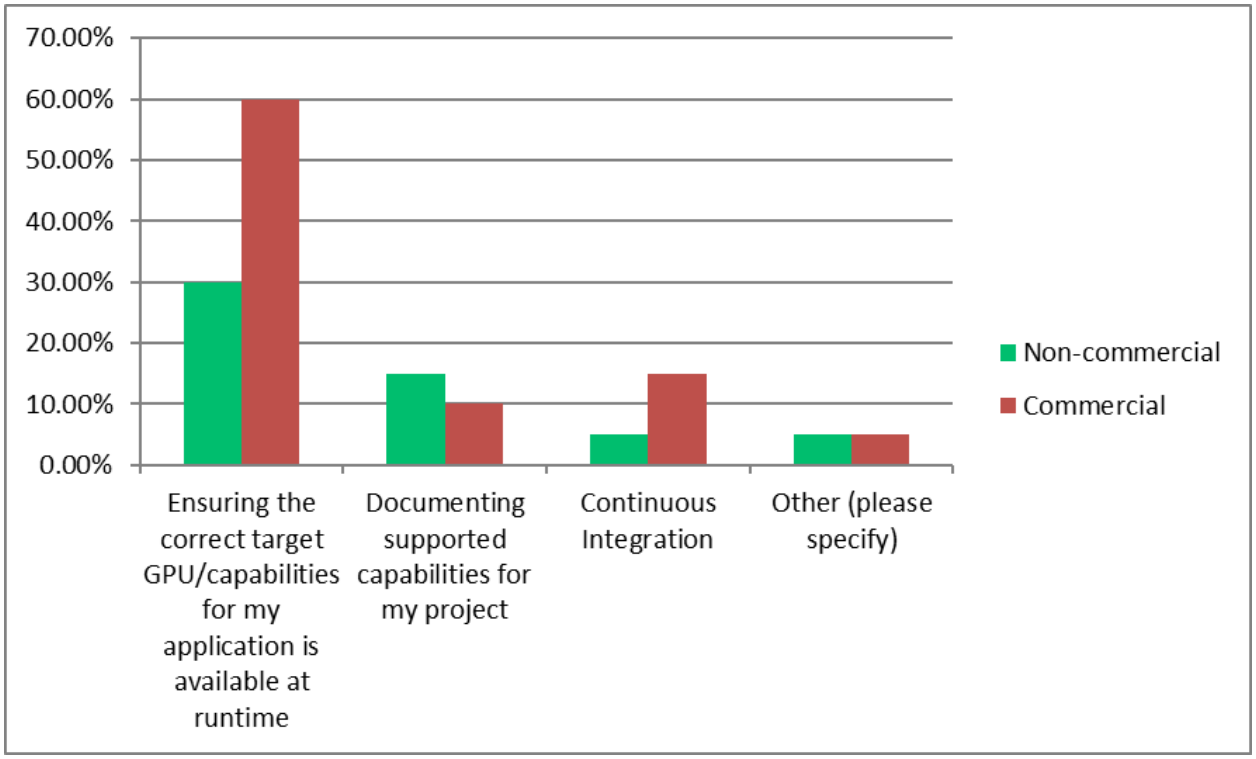
Answered: 23
Skipped: 235



As suspected, for those using the Profiles toolset, the Profiles Layer is used more heavily than the Profiles API Library.

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

Answered: 20
Skipped: 238

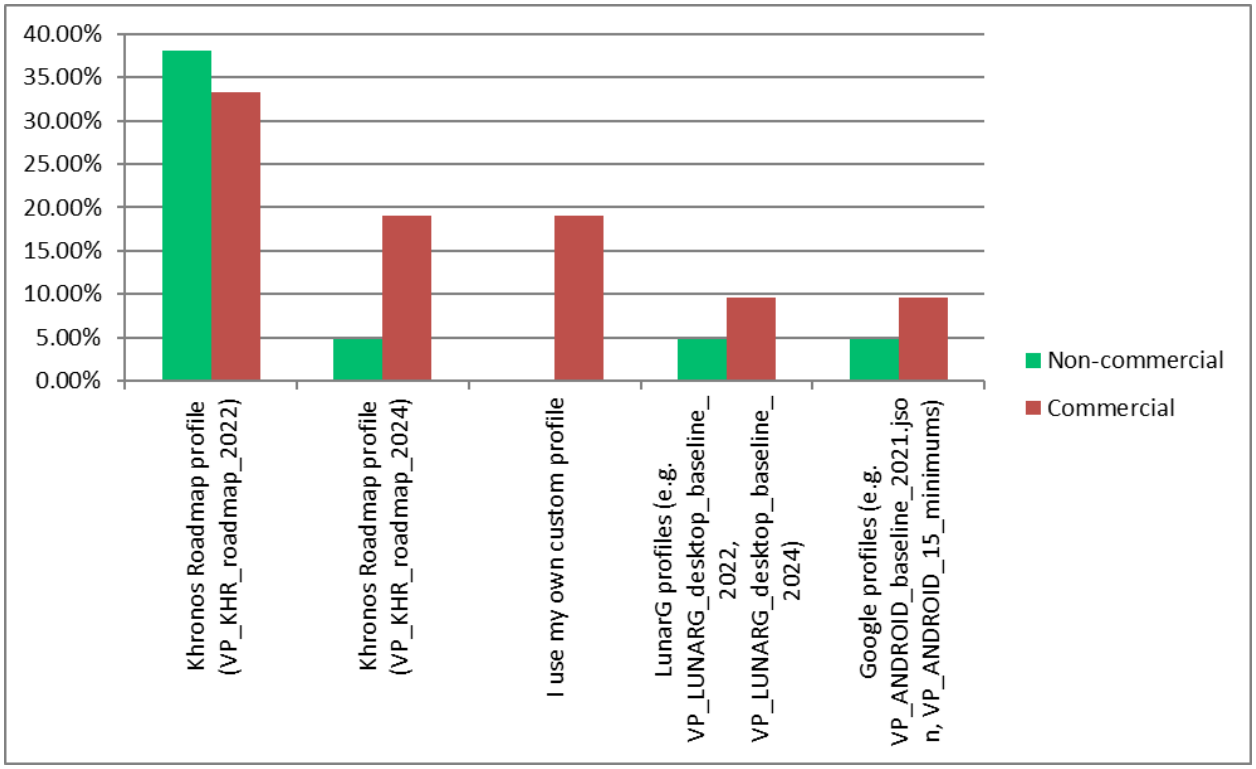


Other:

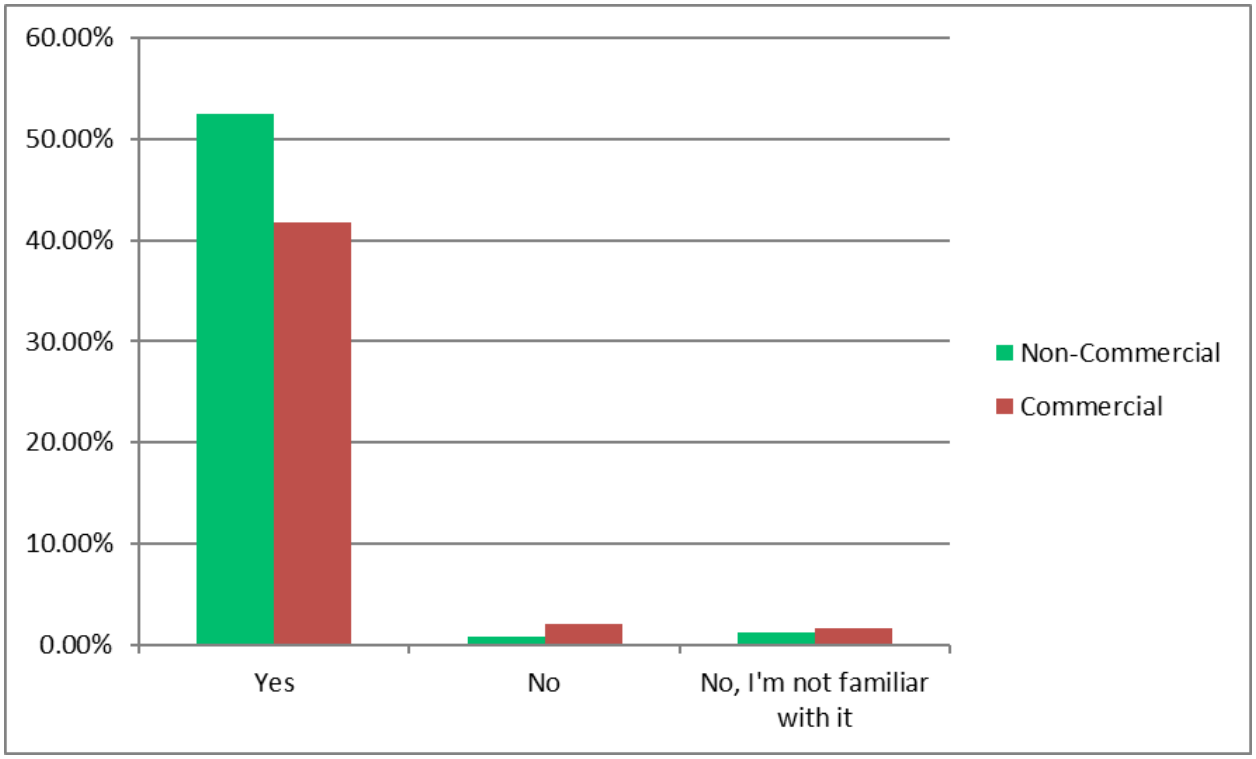
1. I'm using the Android Roadmap 2021 profile while developing on desktop to make sure I don't accidentally use a capability that my target market doesn't have
2. Test generation (I actually have somebody else use it for me:-)

Which of the Vulkan Profiles provided in the Vulkan SDK do you use? (check all that apply)

Answered: 21
Skipped: 237

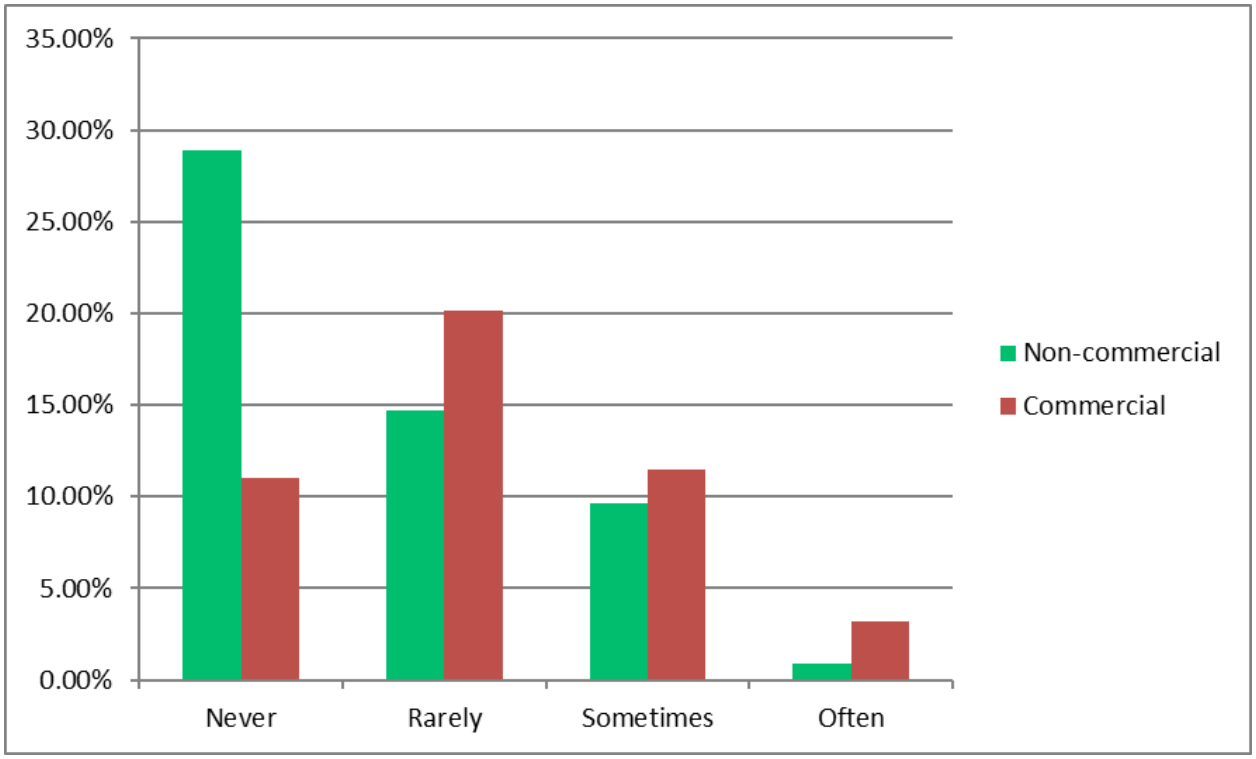


Do you use the Khronos Vulkan Validation Layer (VK_LAYER_KHRONOS_validation)?



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

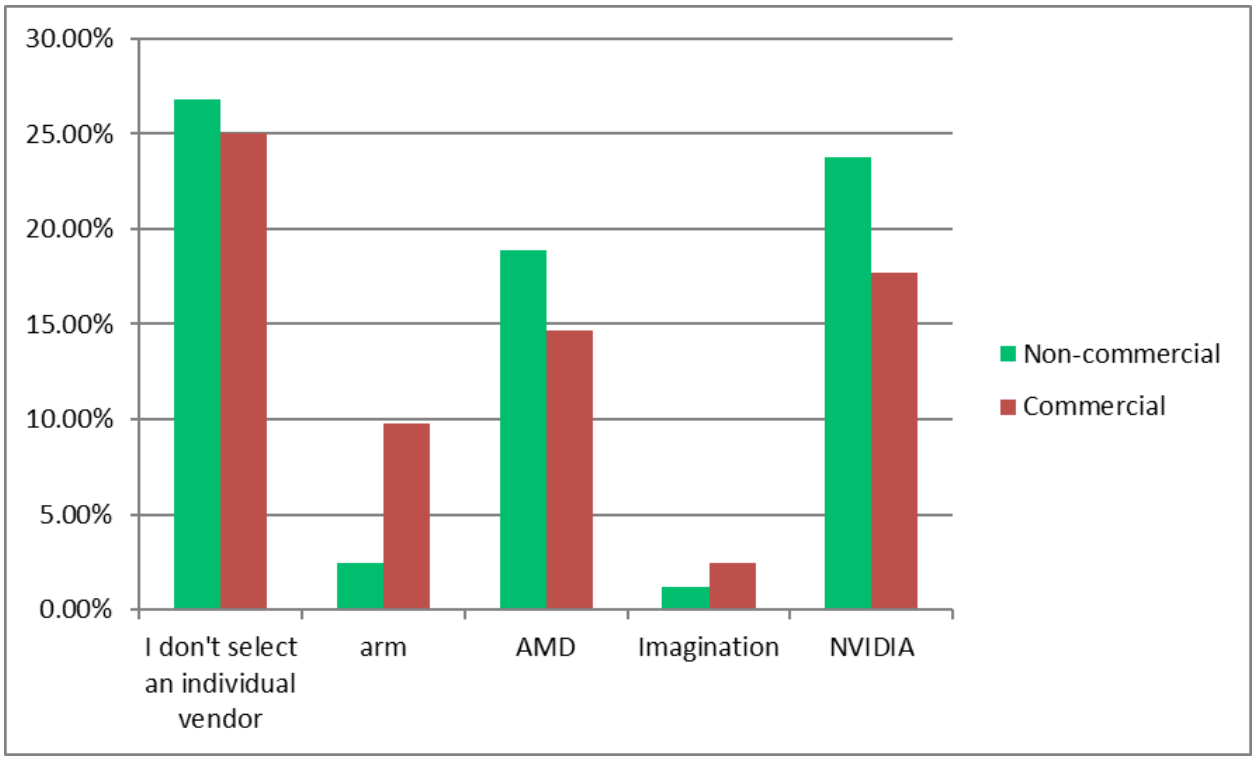
Answered: 224
Skipped: 34



1. Game-like app. FPS drops from ~120 to 15.
2. There is sometimes something that triggers an error in the validation layers when ImGui is integrated with a basic renderer such as the ine from vulkan-tutorial
3. Enshrouded
4. Debugging a Vulkan layer that runs on top of a video game
5. When the extension is attached to Android devices
6. Even on a high-end system (RTX 3080 + Ryzen 9 5950X + 64GB RAM) rendering multiple instances of meshes or different meshes slows down very quickly. But debug mode is required for any basic debugging, and validation layers are also basically required for any development of Vulkan applications. So it gets difficult to properly develop in those situations.
7. It's mainly annoying that the developers of SteamVR have not fixed the validation errors in their application for years, such that their validation errors get spammed in the log whenever I enable the validation layers in my VR application. (Oh, there was a VK_LAYER_DUPLICATE_MESSAGE_LIMIT? I never knew about that!)

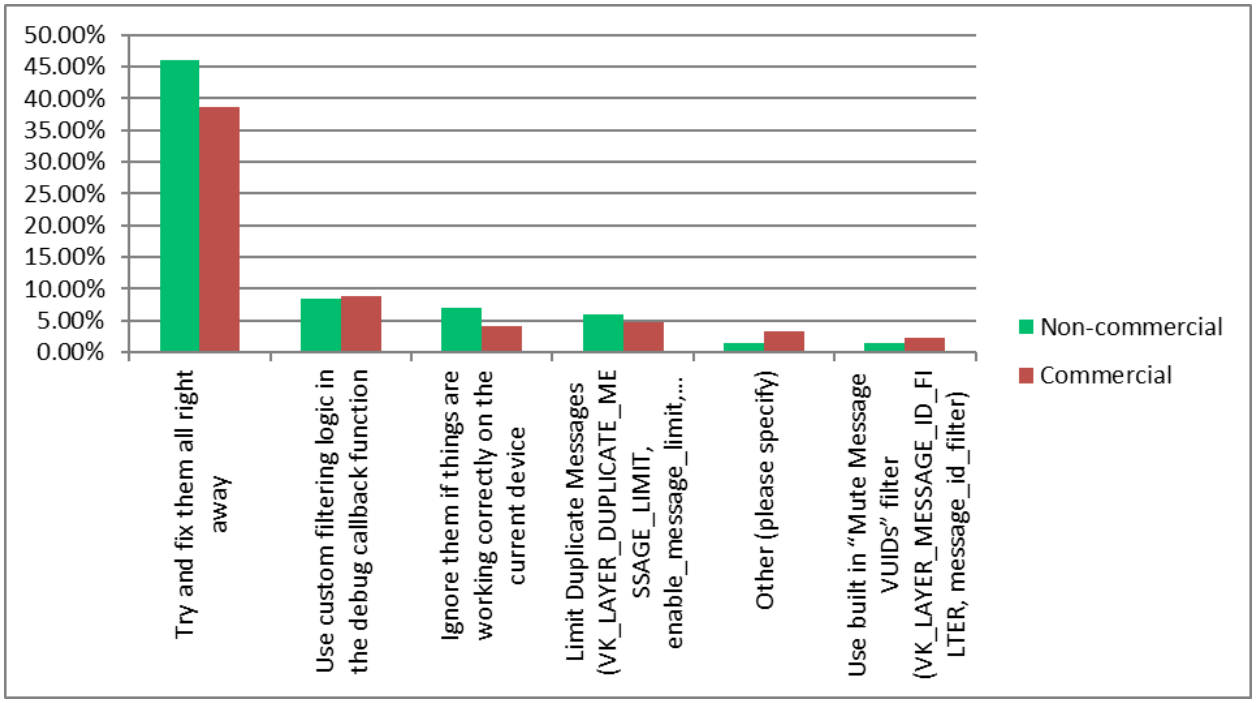
If you use the Best Practices layers (VK_VALIDATION_FEATURE_ENABLE_BEST_PRACTICES_EXT), which vendors do you select? (check all that apply)

Answered: 169
Skipped: 89



When you get many Validation Layer messages, what do you do? (check all that apply)

Answered: 221
Skipped: 37

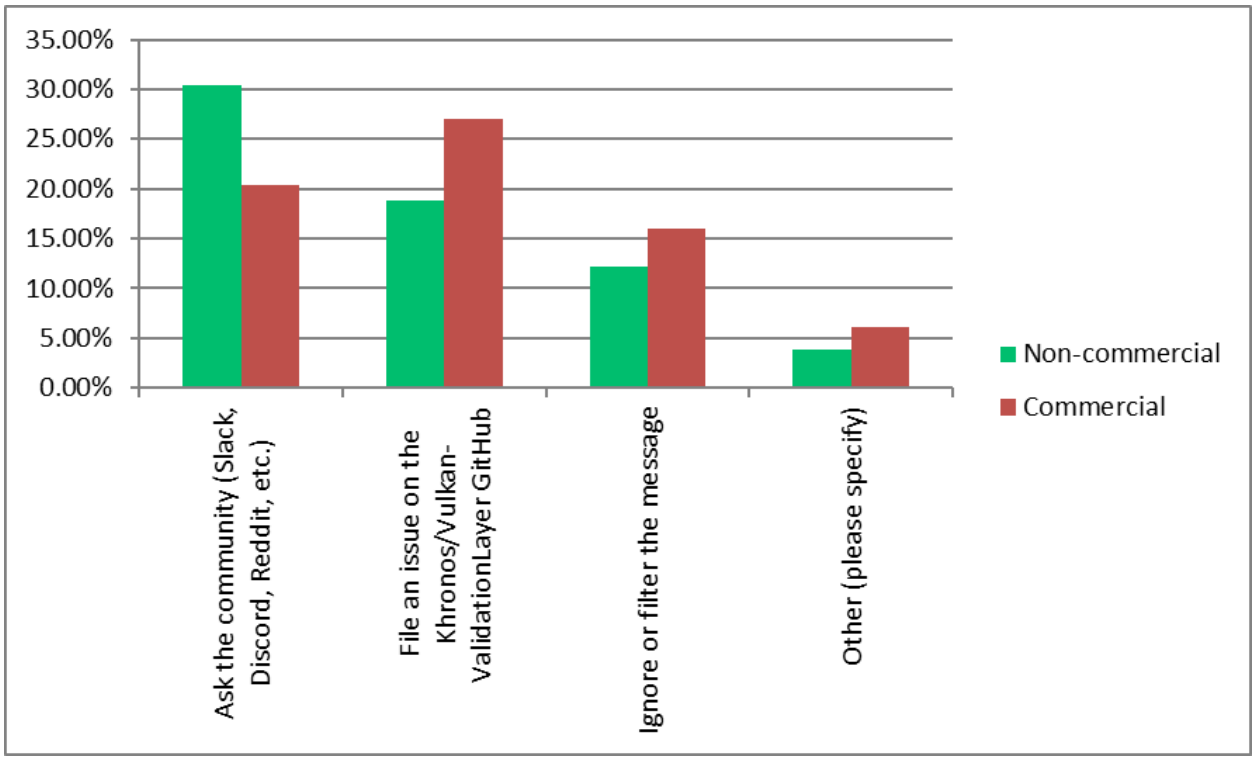


Other:

1. I'll try to fix one or two, run my program again, fix one or two of any remaining messages, run my program again...
2. I rely on the validation layers to catch errors in automated unit tests. If they fail, code can't be submitted.
3. Some messages we decide to ignore because they are either 1) known bugs in our code that will take longer to fix 2) bugs in the VVLs 3) things that are technically disallowed but work in practice (looking at the lack of STORE_OP_NONE)
4. Log everything to a file and search it
5. Panic and run to the nearest corner of the room and cry against the wall because it probably has something to do with synchronization.
6. some messages are due to driver bugs or non-compliance (e.g. linux soft driver, WSL2 driver) and need custom filtering
7. Fix them after completing a feature, if not important at the moment
8. Fix ERRORS but ignore WARNINGS
9. Fix them once a feature is completed on the local device.
10. reducde log level

When you think you have found a bug in the Validation Layers, what do you do? (check all that apply)

Answered: 187
Skipped: 71

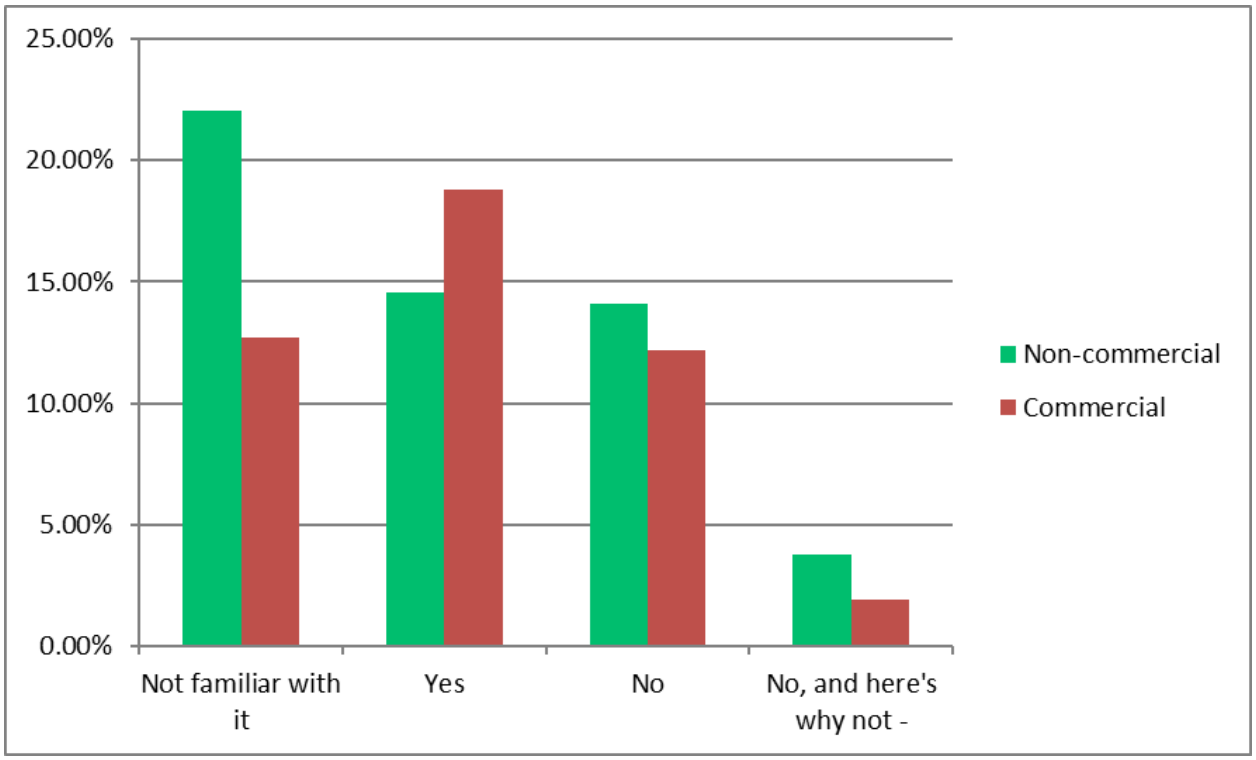


Other:

1. Curse and swear!
2. I haven’t encountered one yet
3. I've never found a bug in the validation layers
4. The few times this occured there were already newer versions with fixes.
5. I also sometimes try to fix the bug.
6. Contact my LunarG engineer colleagues
7. Unable to understand a bug if there was one. I am a beginner.
8. Mark them in our code-base to research/understand better why it would be a bug. In the future I will reach out more via discord to get earlier feedback.
9. Remember that I have not and continue looking for my bug
10. Pray to the all mighty graphics gods.
11. Have not found any
12. Create pull request for fix it.https://github.com/KhronosGroup/Vulkan-ValidationLayers/commits?author=Andreyogld3d
13. Not that confident lol
14. Haven't needed to yet :-)
15. Hasn't occured.

Do you use Debug Printf (debugPrintfEXT, GLSL_EXT_debug_printf, VK_VALIDATION_FEATURE_ENABLE_DEBUG_PRINTF_EXT)?

Answered: 219
Skipped:39



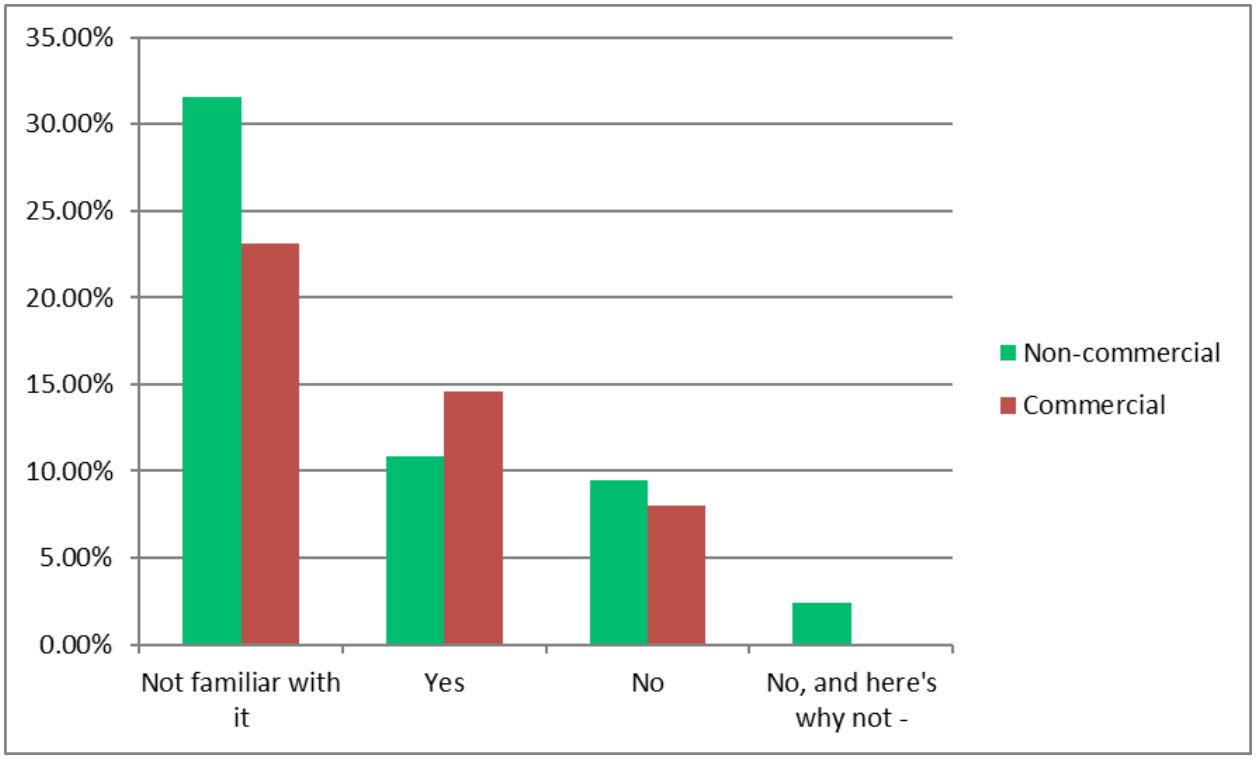
Here's why not:

1. Printing messages from millions of threads isn't very useful. Proper debug tools are needed (especially for compute shaders with raytracing/ray queries extensions)
2. It didn't work reliably the last time I tried. That was some significant time ago though.
3. I keep forgetting it exists tbh
4. Renderdoc
5. I use slot 0 usually, it messes how I set up my descriptor sets and usually I'd prefer an *assert* over a debug_printf.
6. My applications start with a command window, then windows and then Vulkan. Debug errors are displaced in the command window and written to a txt file.
7. I tried but it prints different values than what's actually being used in the shader i was debugging.
8. it has a lot of noise, and mostly doesn't work better than a debug image.
    a. Comment from LunarG: It is acknowledged that since the first release of the debugPrintf functionality, there hasn't been additional improvements made.

9. My current development pipelined based on kernel_slicer solves debugging for 99%. The rest 1% coul be solved just with traditional prontf kung-fu debugging (copy data to CPU and print them).
10. It's not very practically useful due to the massive amount of output it generates and the bad performance
11. https://github.com/KhronosGroup/Vulkan-ValidationLayers/issues/7178 would love to use it but GPU-AV support is limited when you're using some extensions.
12. Tried it once, it just crashed. But that was some time ago.

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

Answered: 218
Skipped: 40

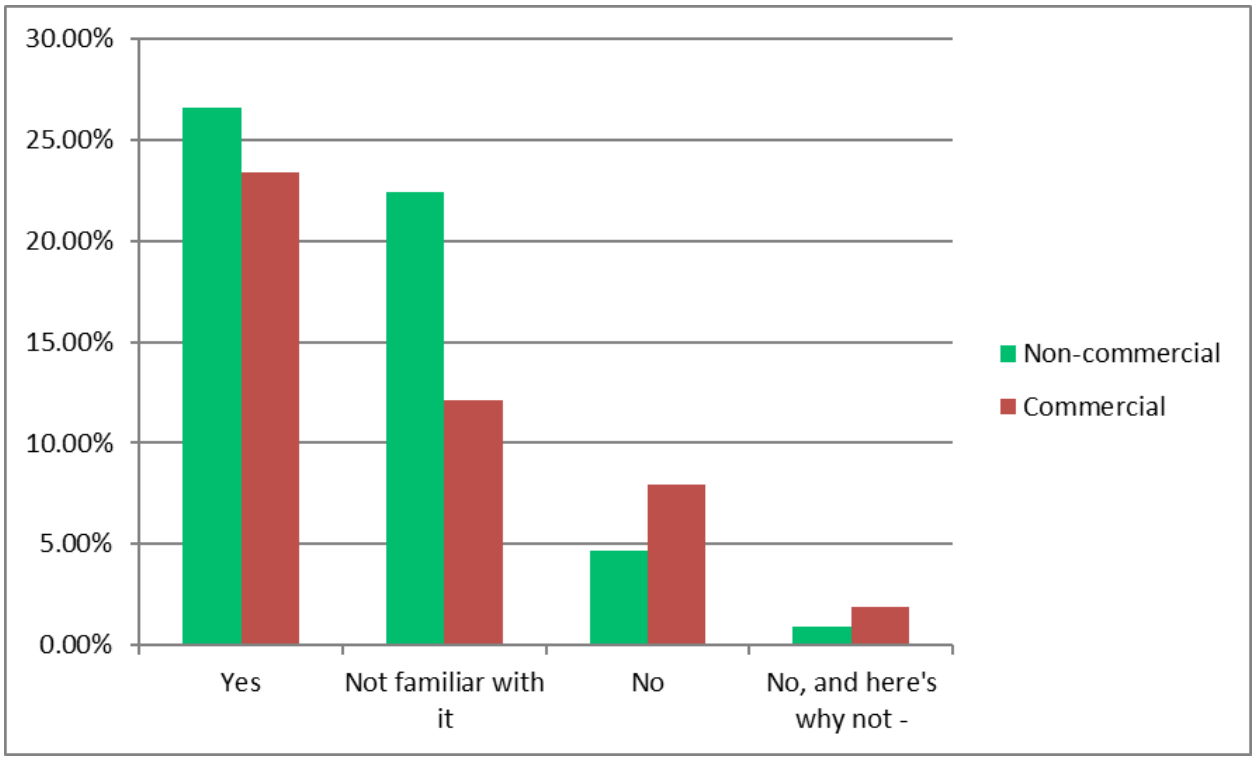


No, and here's why not:

1. On nvidia drivers, GPU assisted validation currently doesn't work well with VK_EXT_descriptor_buffer.
2. Conflicts with debug printf
3. Beginner so projects are not complicated enough to require using it.
4. Not really sure what kind of errors this catches in addition to regular validation
5. https://github.com/KhronosGroup/Vulkan-ValidationLayers/issues/6025
6. Does not work with descriptor buffer, last time I checked.

Do you use Synchronization Validation (VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT)?

Answered: 220
Skipped: 38

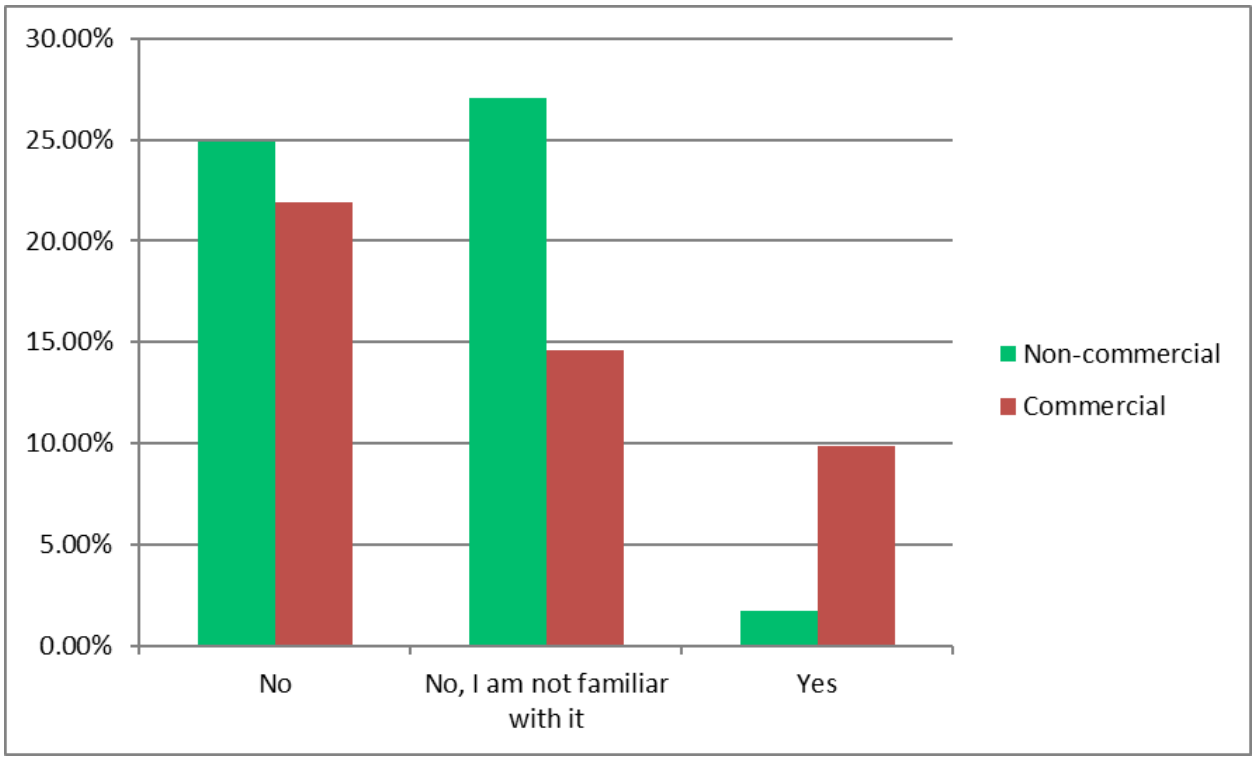


No, and here's why not:

1. Used to but problems with it, likely related to lack of timeline semaphore support, has forced us to disable it for now.
2. It's incomplete and still in development right? Last time i tried it gave false positives anyway and didn't even do cross cmdbuffer or cross queue validation so yeah..
3. Beginner so projects are not complicated enough to require using it.
4. seems to work incorrectly for complex setup (multiple render threads)

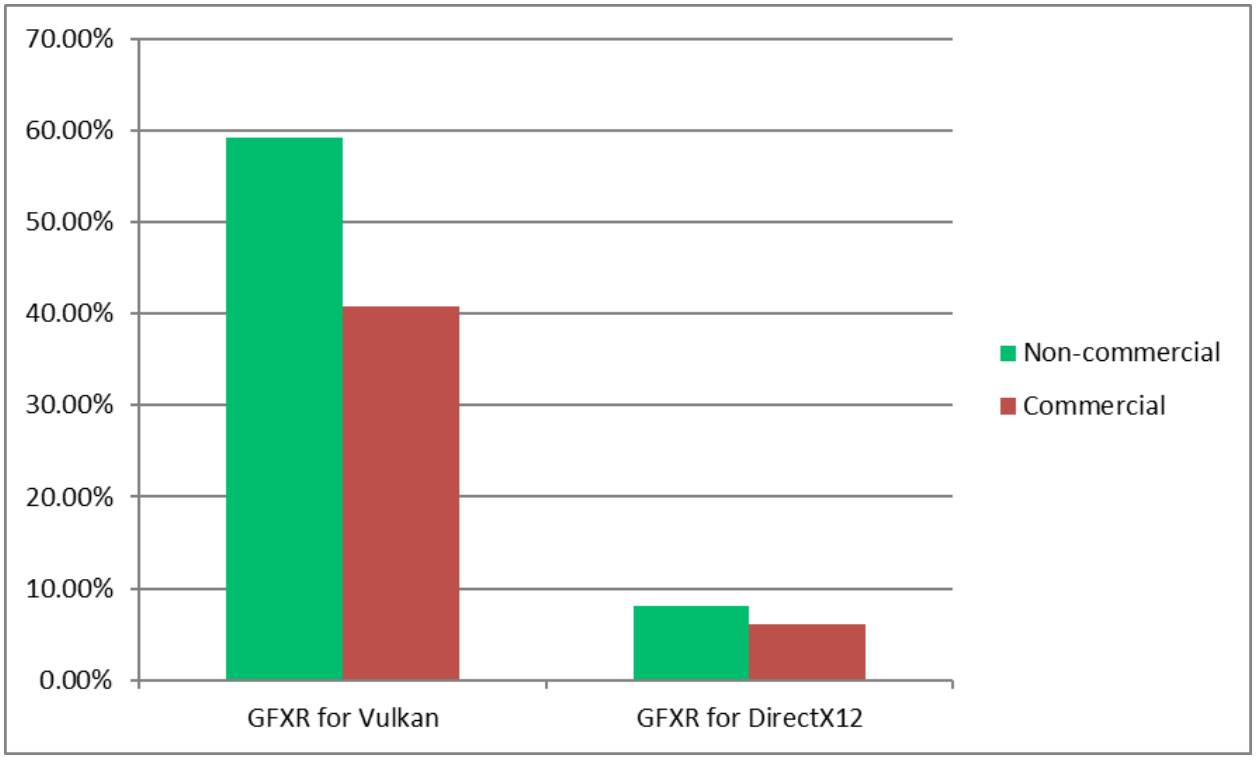
Do you use GFXReconstruct?

Answered	240
Skipped	18

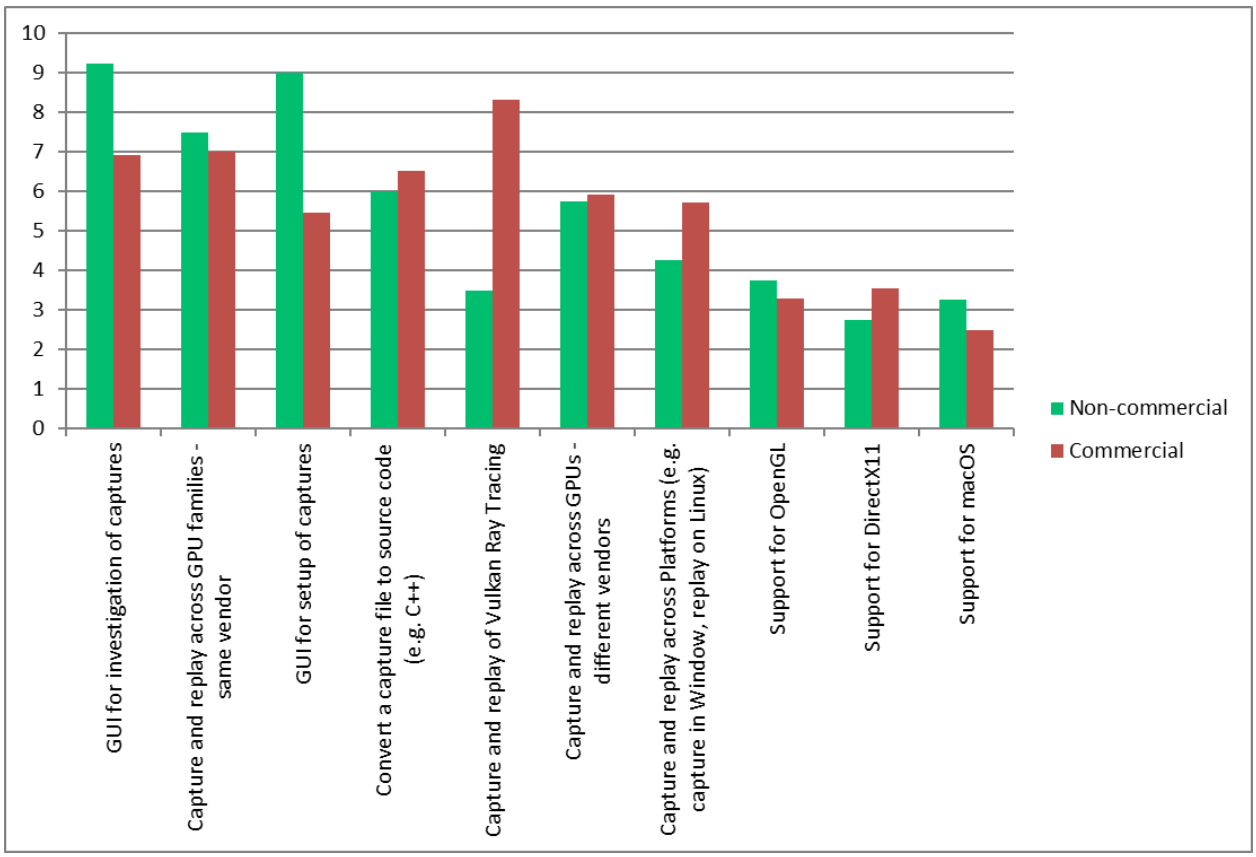


Which version of GFXReconstruct do you use? (check all that apply)

Answered	29
Skipped	229

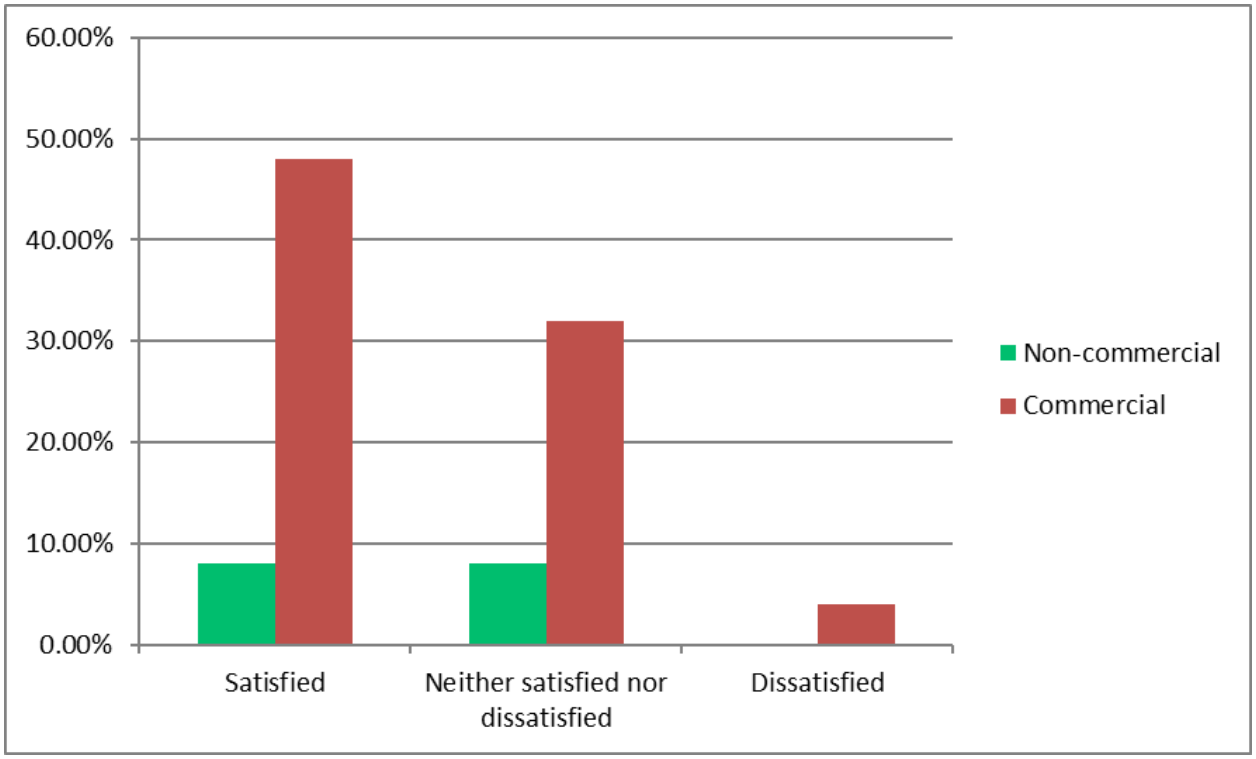


If you use GFXReconstruct, rank the following possible new GFXReconstruct features in terms of usefulness for your projects: (1 = Highest, 10 = Lowest)



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

Answered 27
Skipped 231

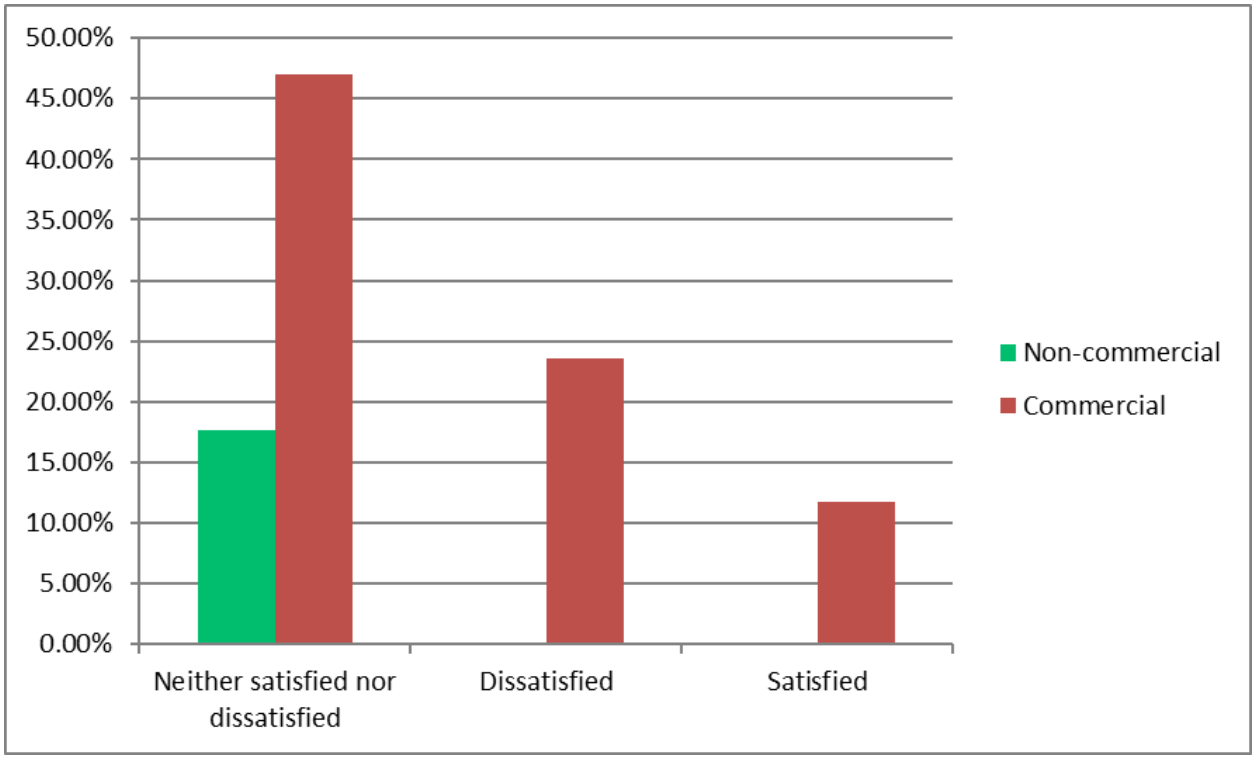


Comments:

1. It was good enough to do a capture to file a bug report to MESA. Had to build from source. No experience beyond that.

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

Answered 19
Skipped 239



Comments:

1. but haven't been using it much yet - so far no issues. wonder if swapchain replay bugs are still around (those were quite annoying in vkreplay)
2. I understand there are issues
3. Haven't tried it.
4. I have only tried using it with Meta Quest 3 and had issues running replay - gave up due to the time sink.

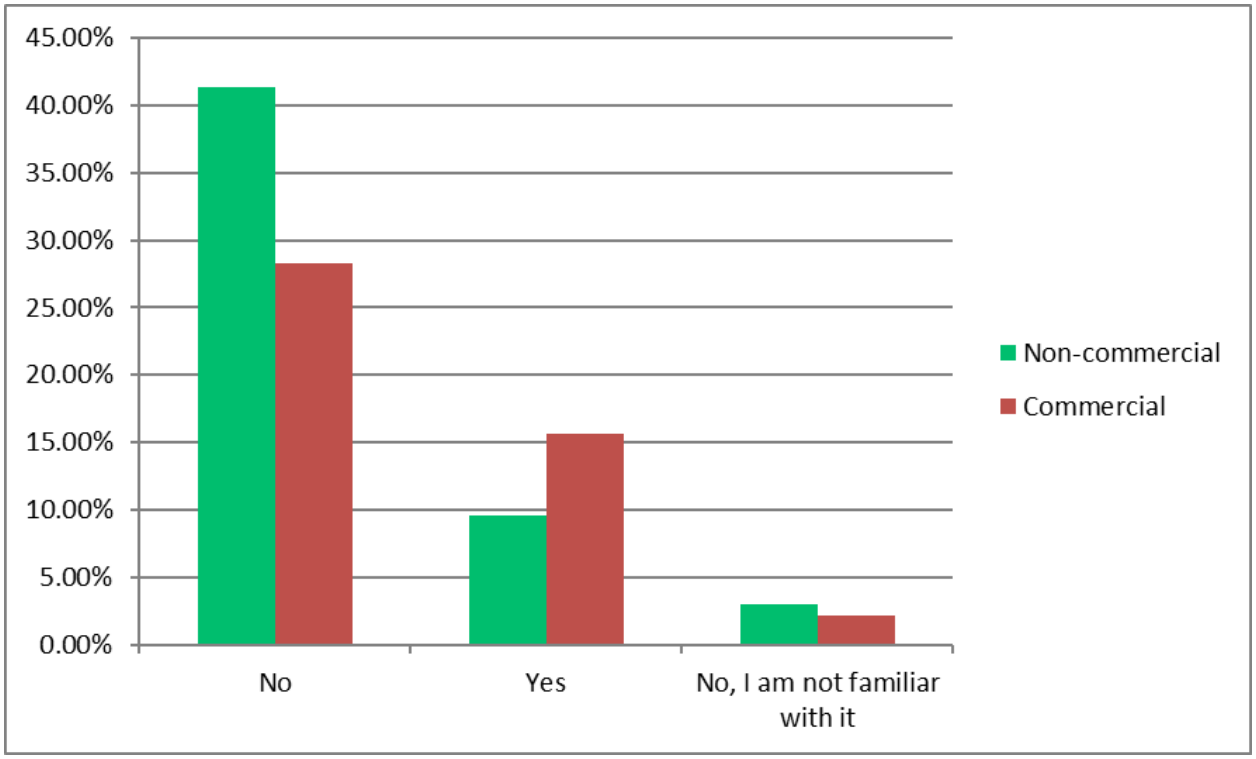
What improvements or enhancements would you like to have added to GFXReconstruct?

Answered 7
Skipped 251

1. workgraphs support
2. I just want to mention that the Capture and replay of VK raytracing is almost an impossible task because of the lack of guidance we had in terms of binding stuff to the pipeline from day one. So now every project that is using VK RT is using raw memory addresses to pull data into their shaders. So in order to solve this problem you'd have to instrument the shader to see where the user is pulling the memory address and then try to fudge it in replay. So perhaps there needs to be an api way of supplying this data into RT pipelines.
3. Driver compatibility and ASIC compatibility needs to improve, ex: if we captured on one configuration replaying on other config irrespective of ASIC and drivers.
4. Life without python
5. Make a debuger with it similar to RenderDoc. Good to have more choices
6. Ray tracing often crashes
7. Make a debugger with it like renderdoc

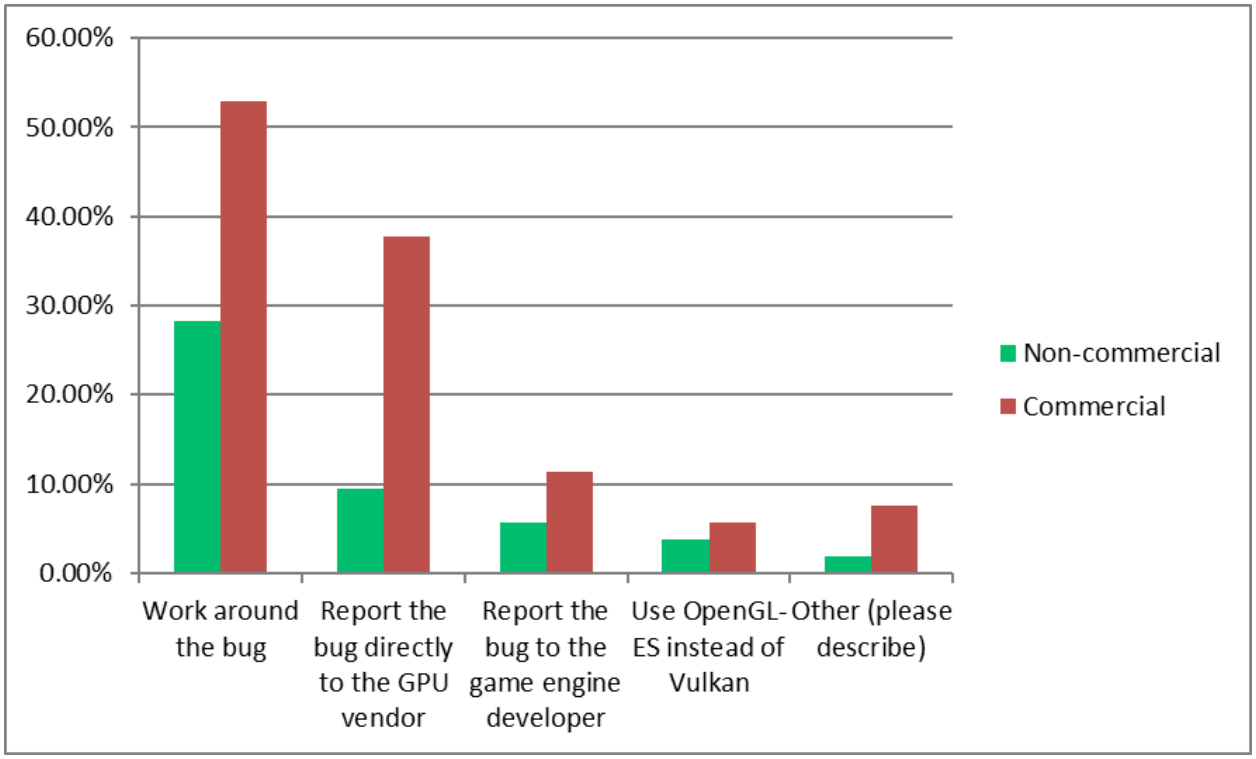
Is Android a target of your application development?

Answered	236
Skipped	22



If you believe you have found a Vulkan driver bug on Android, what do you do? (check all that apply)

Answered 55
Skipped 203

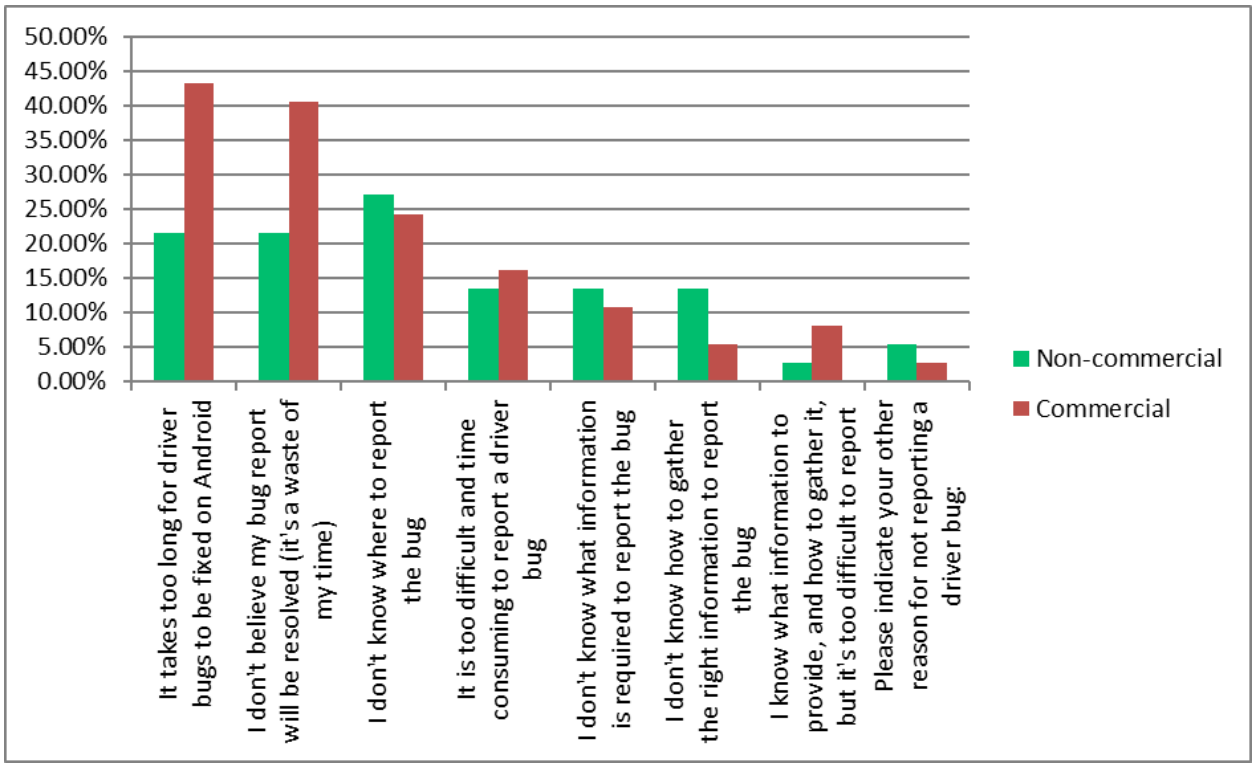


Other:

1. Have to workaround, as android never updates drivers
2. Report the bug to OEM/SoC vendor
3. Cry
4. look at internet
5. Ask to devs in Vulkan Android dev discord channel

If you do not report a driver bug that you found on Android, why not? (check all that apply)

Answered 39
Skipped 219

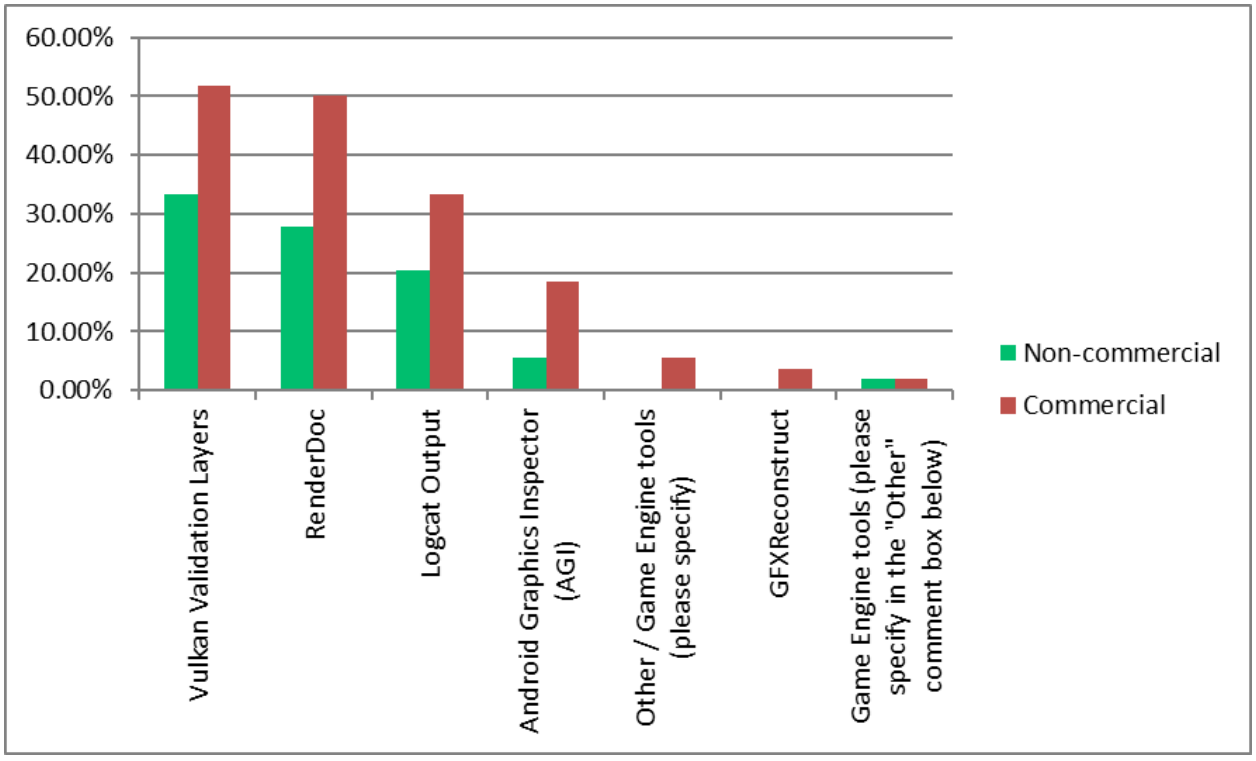


Other reasons:

1. I'm shipping on a lot of phones that no longer get driver updates. Reporting bugs will help future phones, and some current ones, but I still need to work around the bug in my code to ship on the older phones that people are still using
2. There are too many vendors ...
3. I do report, but mostly to get advice on how to work around it - I do not expect mobile vendors to fix the issue.

When you need to debug a rendering problem on Android, what tool(s) do you use? (check all that apply)

Answered 56
Skipped 202

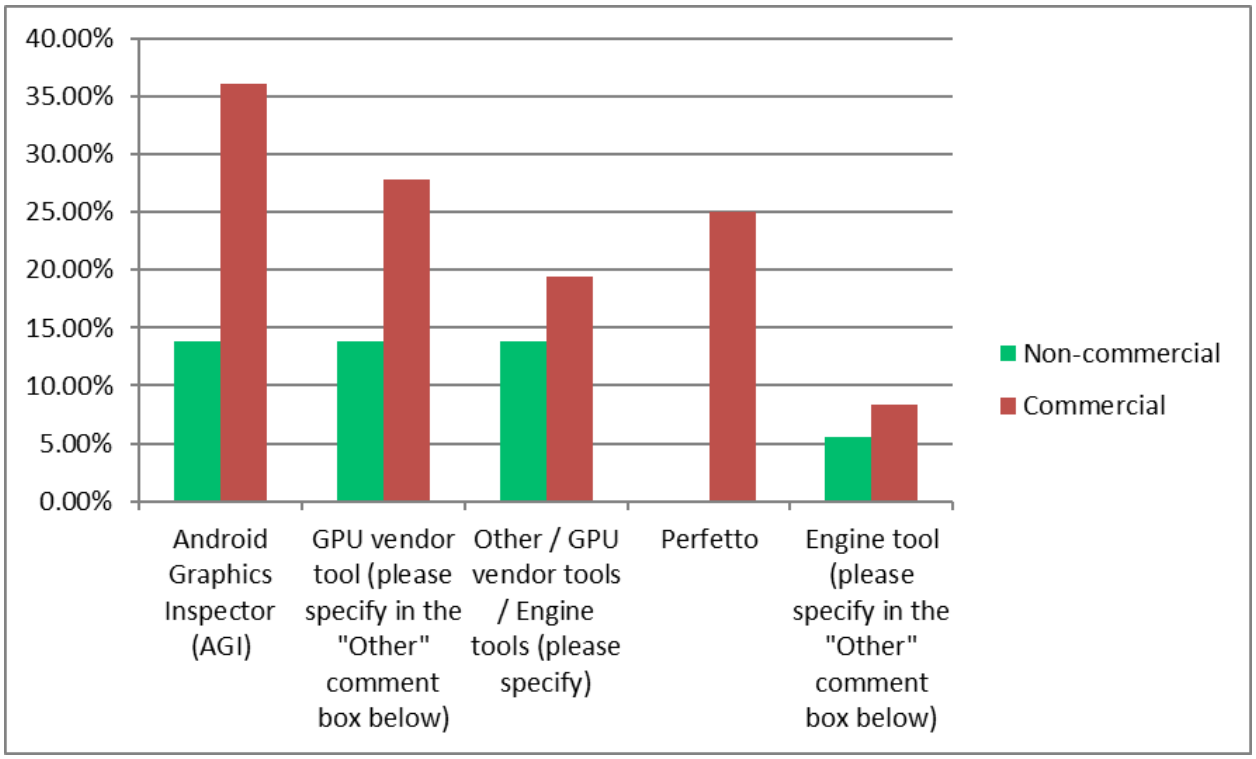


Other:

1. we have a custom engine with internal instrumentation and error reporting.
2. Shader printf
3. Shader printf

What graphics performance/profiling tools do you use on Android? (check all that apply)

Answered 38
Skipped 220

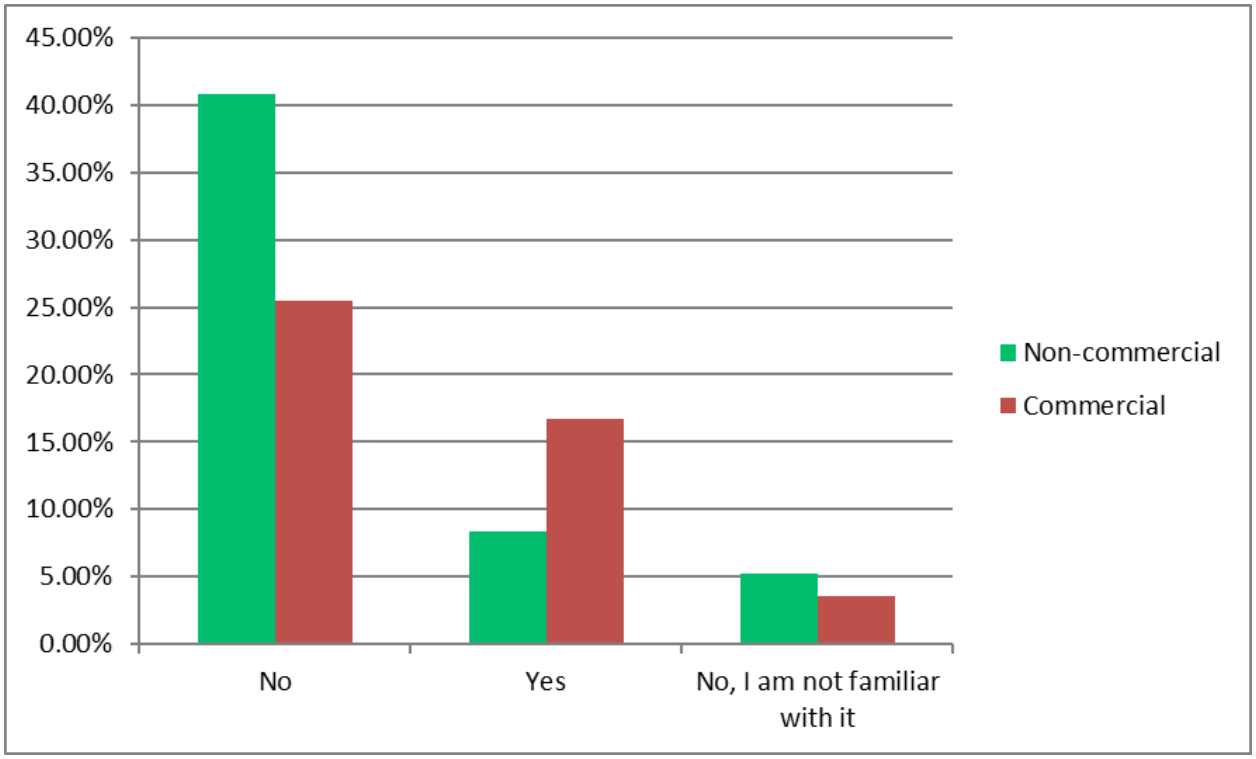


Other:

1. vkCmdWriteTimestamp :(
2. Mali Streamline/Arm Perfomance Studio
3. performance queries, none of the above actually work
4. Mali Streamline, Snapdragon Profiler, Tracy in my own engine, Unreal Insights if using Unreal Engine
5. Snapdragon Profiler mostly.
6. ARM profiler
7. Qcom profiler
8. RenderDoc
9. PowerVR Tune
10. Tracy
11. Snapdragon Profiler
12. Snapdragon Profiler, NSight
13. simpleperf for CPU side stuff. For GPU times, the timing on RenderDoc as a rough estimate.

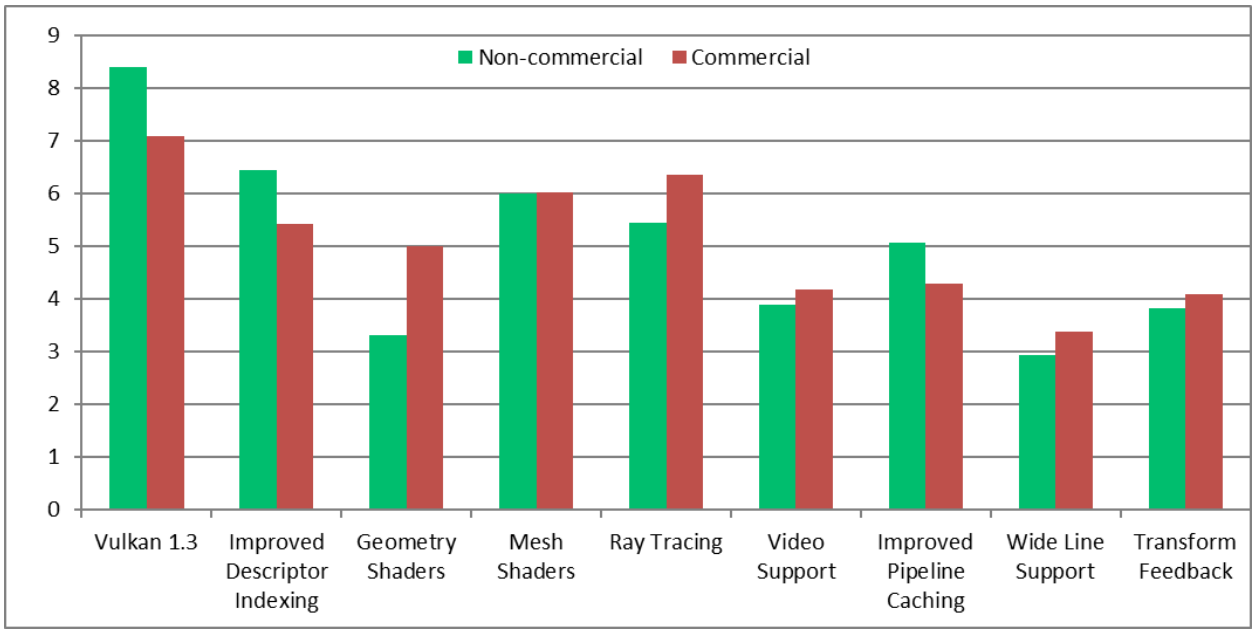
Do you use MoltenVK?

Answered	234
Skipped	24



Rank order the importance of the following Vulkan features being added to MoltenVK (1 = Highest, 9 = Lowest)

Answered 58
Skipped 200



List any Vulkan extensions that you would like to see added to MoltenVK

Answered 11
Skipped 247

1. none
2. Most of them i think
3. debugPrintfEXT
4. VK_KHR_video_queue!
5. VK_KHR_ray_tracing_pipeline
6. VK_EXT_shader_object
7. VK_KHR_ray_tracing_pipeline
8. VK_KHR_shader_maximal_reconvergence (good luck)
9. VK_KHR_indirect_draw_count
10. do not know for the moment
11. gl_DrawID is the only blocker I had getting my app running on MoltenVK.
12. VIDEO

Open-Ended Feedback

There were several questions in the survey asking for open-ended feedback:

1. Do you use the new docs.vulkan.org site?
2. If you use the docs.vulkan.org site, what suggestions do you have to improve the site?
3. If you use the Vulkan SDK, what suggestions do you have for the Vulkan SDK?
4. What Vulkan Samples would be most useful for Khronos to add in the future?
5. What prevents you from being effective and productive while doing your Vulkan development?
6. When using the Vulkan Profiles toolset, what are the inhibitors for you to use them easily or effectively?
7. How could the validation layers be improved?
8. Provide any additional feedback you would like to provide regarding MoltenVK
9. Is there anything else you would like to share?

All of the open-ended feedback in the survey resulting from these questions is compiled in this section. Common feedback is grouped into appropriate sub-groups. No feedback is deleted, even if it is repeated multiple times. Seeing the frequency of a feedback theme is also useful.

Open-ended Feedback : Validation Layer

1. Performance related
    a. performance
    b. Faster
    c. Performance of some of the features could be better, but nothing really major issues in current use.
    d. less performance overhead; slows down test suite execution by a lot
    e. perf
    f. Improve performance, reduce overhead
    g. Better performance
2. Error messages
    a. Error messages from the validation layers are often cryptic.
    b. Sub-categories for 'Best Practices' to avoid too many suggestions
    c. some best-practice messages are just spam (later one was a complain on general_layout + storage_image, as if there is an alternative)
        i. Comment from LunarG: The Best Practices have not had improvements made for some time. In addition, there still isn't clarity of what the best practice layer should or should not be reporting.
    d. Formatted for readability by humans, actual normal humans, not LunarG devs
        i. Comment from LunarG: What this really means is when writing logic for a VU it is very easy to see all the parts, but when you hit in your app, it is hard to see what is going on with the many connected parts.... This is a difficult problem to solve. We should keep it in mind while writing the VUs.
    e. Easier to read error messages
    f. Include resource names from VK_EXT_debug_utils in messages, if available.
    g. with useful messages about the origin of the errors. Most of the time the messages are completely cryptic, and this is a huge problem when you are learning Vulkan, only when you have more experience will you be able to locate the source of the errors.
        i. Comment from LunarG: The validation layers are not able to print a Vulkan Guide level explanation for every error. It is part of the learning curve unfortunately. You can use tools such as apidump or GFXReconstruct to see the sequence of calls your application made to help debug them.
    h. More readable error messages or links to an explanatory page for error messages that don't track back to the specs.
    i. Possibly some suggestions on what the expected values are. The validation layer logs are particularly verbose so it is difficult to parse what parameter is incorrect and what the value should / should not be.
        i. Comment from LunarG: The validation layer can't know what you intended to do and therefore what would be the expected values.
    j. Easier readable messages
    k. Hard to track down what is generating validation errors

l. The single most annoying thing I encounter is URLs that I get back from tooling that reference the primary vulkan specification, which takes a long time to load and can bog down browser performance. It would be nice if there were an extension that could automatically intercept such URLs and redirect people to the corresponding single-page version of whatever was being linked.
m. Links to full spec with long load times
n. Messages about crashes of the instance / device are usually not helpful / need more information
o. Combine error messages with API calls to see what call caused what msg.
p. Better output: e.g. html page, tree structure?
q. Format the messages to make them more readable
r. Other than that. a improve that formatting of validation messages would be most welcome: UNASSIGNED-CreateInstance-status-message(INFO / SPEC): msgNum: -2016116905 - Validation Information: [UNASSIGNED-CreateInstance-status-message] Object 0: handle = 0x55ce13436270, type = VK_OBJECT_TYPE_INSTANCE; | MessageID = 0x87d47f57 | vkCreateInstance(): Khronos Validation Layer Active: Settings File: Found at ~/.local/share/vulkan/settings.d/vk_layer_settings.txt specified by VkConfig application override. Current Enables: VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION, VALIDATION_CHECK_ENABLE_SYNCHRONIZATION_VALIDATION_QUEUE_SUBMIT. Current Disables: VK_VALIDATION_FEATURE_DISABLE_SHADER_VALIDATION_CACHING_EXT. Objects: 1 [0] 0x55ce13436270, type: 1, name: NULL
s. For muted message VUIDs there should be a setting that raises them at least once. We like to be reminded of those validation messages if we think they are a bug (or they are a confirmed bug) but not be spammed by them (although we want to be spammed by the non-filtered ones).
    i. LunarG comment: The Vulkan Configurator has an option to limit duplicate messages. You can set that to 1

3. Coverage
    a. Bindless resource accesses in shaders are not checked, that would help us a lot if possible. E.g. an index into a texture array comes from a push constant, but the index is outside the arrays range.
    b. bindless resource checks in the validation layers would be on my wishlist.
    c. Support for descriptor buffers would be nice
    d. improve coverage of new exensions (descriptorbuffers, timelinesem sync)
    e. More support for GPU-AV
    f. Feature complete to match the validation rules in the spec.
    g. Focus on core validation rules starting in v1.0, v1.1, v1.2 etc. before adding rules for new extensions or new features"
    h. IIRC usages of descriptor buffers is not yet validated.

   i. More SPIRV validation. For example the EXT_mesh_shader validation in spirv-tools is not complete and hasn't progressed for months. Generally the spirv-val seems abandoned
   j. More coverage. For example the spirv validation of EXT_mesh_shader is not fully implemented and generally spirv validation doesn't see many updates
      i. Comment from LunarG: These statements are correct. https://github.com/KhronosGroup/SPIRV-Tools/issues/4919 has been open for a long time. We will see if we can get an appropriate person from the working group to finish the work.
   k. Also, the ray tracing extension has a lot of missing validation.
   l. Raytracing validation
   m. fix false positives in the lastest version.
4. Synchronization Validation
   a. GPU assisted synchronization validation - would be nice to catch VU violations triggered within shaders.
   b. validation layers are slow. hard to figure out sync hazards. QueueSubmit synchronization diagnostics should be improved (e.g. print debug labels of involved resources)
   c. debugging layout transition validation errors
   d. Sync validation messages, they are too complex and hard to understand without ability to break in specific places during commands recording.
   e. Timeline semaphore support for synchronization validation is needed
   f. More support of Synchronization Validation with bindless is needed
   g. Validation Sync error can possibly contain some suggestions, like "This situation is likely the result of.. "
5. DebugPrintf
   a. if GPU-assisted validation and Debug Printf could be used simultaneously, it would help me sometimes.
   b. I would like to inject debugPrintf without a need to do any changes to application binary
   c. Debug printf could have a dump pass to write all shaders to spirV or GLSL(if provided) and provide a way to override them in second pass and inject all needed extensions to add those printfs.
   d. Separate debug printf layer from rest of validation layers if possible.
   e. Want to programmatically enable debug printf only.
      i. Comment from LunarG: You can programmatically enable debug printf. Take a look at this whitepaper to learn how to do it.
6. Timely release of Validation Layer support for newly released Vulkan API
   a. But it would be rather nice if a extension was released in tandem with validation layers support.
   b. Providing pre-built binaries from GitHub and having layer support for new extensions outside of SDK releases

c. I've had a lot of cases where validation layers didn't support a certain extension. And trying to figure out what is causing a segfault without validation layers support is. Less than pleasant.
    i. Comment from LunarG:. The current workflow utilized by the Vulkan Working Group results in the validation layer development not starting until the extension is released (in many situations). This causes the lag between extension release and validation layer support for the extension. The working group is beginning some changes to the workflow to enable validation layer development in parallel with CTS development. It will take some time to get there, but over time things will improve.

7. Other
    a. No suggestions.
    b. do not for the moment
    c. I don't see any. It's THE main reason that drives me to use Vulkan. They are so nice and I really feel like once all validation errors are gone, only logic bugs remain. Which is a great thing.
    d. I'm learning how to use it just now, so it's a bit early to say, for me.
    e. Create a more streamlined process of initializing the validation layers and customizing them.
    f. Validation layers are good and people should be using them more often, the problem is that if you post about your problem online, it is usually due to another library integration error from their side or the problem occurs from an obscure issue that not all users would encounter so waiting for an answer could take ages if it's correct in the first place
    g. Shader location information is very unreliable for me in most cases, so much that I don't trust it anymore but rather use the validation output to pinpoint the offending location manually.
        i. Comment from LunarG: The shader interface code (and error messages) have been refactored in the last year. It can be hard to understand if one is not familiar with the terms used in the spec. We are open to any issues/suggestions on how to improve the wording. Feel free to submit any suggestions on the github: https://github.com/KhronosGroup/Vulkan-ValidationLayers
    h. Checking for more issues is always good. I've made a couple GitHub issues with specific suggestions already
    i. Provide more sample code
    j. By separating the validation logic from the business logic of our application
    k. Also printing in hanging shaders (to uncached memory) would be great.
    l. Enable every existing validation by default, use opt-out flags instead of opt-in
        i. Comment from LunarG: We intentionally do not enable all the options by default. For example, debugPrintf is not compatible with GPU-AV. As well, by enabling them all, you will get absolutely terrible performance. It is better to run core validation first, clean up the issues, and then start enabling other options.

m. The issues are known just a matter of fixing them.
n. No regressions when new versions are released.
o. Overriding VK_USE_64_BIT_PTR_DEFINE, i.e., the underlying VK_DEFINE_NON_DISPATCHABLE_HANDLE isn't really supported on all platforms. For example, setting VK_USE_64_BIT_PTR_DEFINE to 1 with MSVC just crashes (probably because of size mismatch). I don't think this is fixable from the layers code. BUT perhaps the SDK could provide a header with compile-time checks (static_assert) that could ensure no such things happens.
    i. Comment from LunarG: This is a known issue. Issue https://github.com/KhronosGroup/Vulkan-ValidationLayers/issues/5961
p. Some compile time validation would be amazing but I don’t see how that would be possible.
q. Make it simple on Android. Windows you just install it, on Android is bunch of setup
r. A mechanism for passing in Tracy style SourceLocationData so that tools can get information about the calling code. In December 2023 I added extensible instrumentation support to the VulkanSceneGraph project, and as part of this to support Tracy frame profile I created a vsg::SourceLocation struct that is compatible with the Tracy SourceLocationData, perhaps this could be leveraged for Vulkan layers.

Open-ended Feedback : Vulkan SDK

1. Packaging and installation
    a. Use an MSI installer and simplify side-by-side installations of multiple SDKs
        i. LunarG comment: The windows SDK can only put one path to the layers in the windows registry. However you can point vkconfig to an alternate folder for your layer files. Making it obvious how to do this with vkconfig could be improved.
    b. Option to replace the existing SDK, I usually don't need more than one and uninstalling the old one is an extra step.
        i. LunarG comment: The Qt Installer technology is used for the SDK packaging and it will not allow installation in a folder that already has contents. However an existing SDK could be detected, deleted, and then restart the installation. This has been logged as an enhancement request and may be considered in the future.
    c. Switch back to tar.gz files on mac so they can just be expanded into a folder.
        i. LunarG comment:
            1. The macOS SDK is not a zip file intentionally. We need to track download rates for optional packages to determine actual usage to influence future decisions.
            2. There are improvements we could make to the installer (not be a dmg file) that would make command line interface usage of the installer (as documented in the Getting Started Guide) to be used in an automation environment. This has been added to our enhancement tracker for future consideration.
    d. macOS integration
    e. Configuration under macOS should be simplified.
        i. LunarG Comment: Without an email address to ask more, we are unsure why it is believed to be complicated or what is meant by "macOS integration". Perhaps this comment is more related to requirements of developing in an Apple environment and not the SDK itself?
    f. I would appreciate an rpm package for fedora systems
        i. LunarG comment: Many Linux distributions are getting packages created by the distribution packagers. It is out of scope (resources don't exist) for LunarG to create Linux packages for many Linux distributions. It was decided to create the packages for only Ubuntu because
            1. 2 LTS releases of Ubuntu are tested/supported by the Linux SDK
            2. There is a lag of when Linux distros will have packages with latest validation layers and tools, so we delivered the latest in the Ubuntu packages but can't afford to do it across many Linux distributions.
    g. I wish the SDK was a single library under a single namespace that handled everything.

h. I've found several tools offering "install.zip" files which makes me believe there is actually a formal way to install those into the sdk? Namely shaderc and spirv tools. Couldn't figure it out and just added them to my PATH variable.
    i. LunarG comment: shaderc and spirv-tools are already part of the SDK.
i. Please provide up to date vcpkg packages
    i. LunarG comment: It appears that packages are already being updated by the community (https://vcpkg.io/en/packages, search for Vulkan) in a timely manner.
j. Can we have a simple standalone version of vkconfig please ?
    i. LunarG comment: The concept of a standalone vkconfig doesn't really make sense. vkconfig in the SDK is a way to access all the layers and tools in the SDK. So you need the SDK.

2. New features for the SDK
    a. Bring Vulkan layer factory back.
        i. LunarG comment: The SDK target user is a person developing Vulkan applications. Layer development was not considered part of that audience. In addition, due to resource bandwidth, the layer factory was not getting any cycles for maintenance or improvement. Hence it was removed from the SDK and also deprecated from the VulkanTools repository and placed into a read-only repository. There currently is not a plan to bring it back to the SDK.
    b. Ship a static loader on macos and remove bloat like glslang etc
        i. LunarG comment: A static version of the loader is already included in the macOS SDK.
    c. Validation layers are used by developers. Would it be possible to include debug information besides the releases so we can see more information when debugging/breaking
        i. LunarG Comment: We have discussed this many times internally and each time we discuss it we decided again to not include the validation layer source in the SDK for the following reasons:
            1. Including the Validation Layer source could mislead some developers to believe they can build the validation layers from that included source. To enable a build of the validation layers from that source also adds additional complexities:
                a. Include additional repositories to satisfy build dependencies (spirv-tools, robin-hood hashing, glslang)
                b. Would want to prune out git-specific artifacts from the repository (e.g. .git/, .github/, git settings like local user and remote URL), but doing so makes the repository look unfamiliar to developers (so use the github to access the source)
            2. It is easy to go get the validation layer source from github

        a. All SDK releases are fully branched and tagged with “Vulkan-SDK-X.Y.ZZZ.w” and so it is easy to clone the correct version
        b. Accessing the source from the github also enables users to build the validation layers if desired in a supported manner
        c. The associated PDB libraries included with the SDK are only for the validation layers and not for spirv-tools or glslang. A user would not be able to step into the spir-v validation code or the glslang code used by the validation layers.
    d. Can we include some debugging kit included in SDK to find weather any issue pointing to vulkan or other components.
        i. LunarG comment: It isn't clear what is being requested. If the request is to debug the Vulkan installation on a system, the SDK already has tools to do this. vkconfig provides debug output from the Loader as well as a tab for running vkvia. These tools can help determine missing Vulkan components from your system.
    e. For myself, I really like the way it is provided right now. However, since the vast majority of developed Vulkan projects will require a windowing system such as SDL or GLFW, for a generic user of the SDK maybe you could offer versions of it bundled together with one of the most popular graphic libraries/APIs such as the ones mentioned above.
        i. LunarG Comment: SDL is already included with the SDK. GLFW is coming…
    f. Offline documentation
        i. Comment from LunarG: This is in our TODO list, just hasn't bubbled up in priority yet…
    g. I would love a pure C (non cpp, oop, ans whatnot) abstraction over vulkan. Vk-ez was a great start but sadly got abandoned.
    h. implement something like VulkanScengraph to make it easy to use by other developers, it takes a humongous amount of time to learn
        i. LunarG comment: To enable the existing resources to deliver a usable SDK, the target audience of the SDK was defined to be tools used by those developing Vulkan applications. It was a conscious decision to not include samples, tutorials, and abstraction layers to help improve the learning experience for Vulkan. There are many resources in the ecosystem that already provide this such as vulkan.tutorial.com and the KhronosGroup samples.
    i. I am missing Linux ARM, maybe also Windows ARM.
3. <u>shader/compiler</u>
    a. Make a windows application to compile GLSL to SPIR-V. Command line operation is pretty basic.
        i. LunarG comment: This is best suited for IDE developers and not the Vulkan SDK.

4. Build environments
    a. Vulkan Integration and Build Environment (with and Without SDK)
    b. Get versions with fixed CMake pushed downstream as quickly as possible
    c. Make sure it's CI friendly
    d. better integration with cmake (to generate cmake config files during SDK installation)
    e. Include cmake configs for all the libraries provided by the VulkanSDK to help 3rd party tools find the include, libs, versions etc.
        i. LunarG comment: The above 5 comments represent a non-trivial project but would provide value to SDK users. LunarG would have to make sure all the upstream repositories have cmake config files and also test builds with all of the cmake config files to make sure we aren't shipping something that is broken. As such it isn't clear if we will have the resources to get to it soon. It has been logged as an enhancement to the SDK that would provide good improvements for folks building their projects using SDK components.
5. Other
    a. If possible, change the way you can check for a present compatible device queue to not need a surface.
    b. VK_EXT_debug_utils enhancements
        i. LunarG comment: Without more specifics, it is hard to know what enhancements are needed. However we speculate that the fact that some of the layers (like the Profiles layer) don't implement this extension may be an improvement that is needed.
    c. A better way to handle initialization of the instance and device. There are lots of great helper libraries to take inspiration from."
    d. Deprecation tags for things that shouldn't be used in specific versions of Vulkan. i.e. if you make a define that you're targeting 1.3, the out of date functions should get tagged as deprecated with comments that point to the newer functions to use.
    e. Documentation is sometimes out of date.
    f. Nothing really, works well.
    g. It is definitely much better than what we have with OpenGL, still feels a bit short in overall tooling experience when comparing against the experience provided by Metal, DirectX, NVN and LibGNM(X) SDKs.
    h. Its great.
    i. none
    j. I don't think I have any. Keep up the good work!
    k. No suggestions.
    l. Keep up the great work!
    m. do not for the moment
    n. Nothing at the moment
    o. none
    p. Nothing

q. I am pleasantly surprised how Vulkan SDK experience has improved versus what we ever got for OpenGL, although there is still some room to improve versus the proprietary APIs developer experience.
r. There is still a bit of ""build your own boat before going to fishing"" kind of experience."

Open-ended Feedback : Profiles Toolset

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

Answered: 5
Skipped: 253

1. I'm just getting started with it after hearing so much about Profiles at Vulkanized, so I haven't run into any major pain points yet
2. They work well for our case.
3. Still not as easy to use as in other APIs, where there is no need to parse profile configuration data.
4. using predefined profiles is fine, would like to have an easy way to use profile plus some extensions I can specify
    a. LunarG comment:
    b. To use a "profile + extensions" with the Vulkan Profiles API library
        i. This is possible already at least at VkDevice and VkInstance creation. You can see the "Basic usage of the Vulkan Profiles library" in the Vulkan Profiles tools white paper for this. https://www.lunarg.com/wp-content/uploads/2024/02/The-Vulkan-Profiles-Tools-LunarG-Christophe-Riccio-02-01-2024.pdf
        ii. However, it's true this is not possible for checking whether a profile + extension is supported. I guess for that case, a dedicated user defined profile can be created including these extensions then we can check whether each profile is supported and create VkDevice and VkInstance using both profiles or better require the predefined profile in your user defined profile that include the extra extensions and only check the support of your user defined profile.
    c. To use a "profile + extensions with the Vulkan Profiles layer,
        i. We can still create a dedicated user defined profile including additional extensions and requiring the predefined profile. Then we can load the user defined profile in the Vulkan Profiles layer.
        ii. `"VP_EXAMPLE_myprofile": {`
        iii. `"version": 1,`
        iv. `"api-version": "1.1.142",`
        v. `"label": "My profile",`
        vi. `"description": "Description of my profile",`
        vii. `"profiles": [`
        viii. `"VP_ANDROID_baseline_2021" // A predefined profile`
        ix. `],`
        x. `"capabilities": [`
        xi. `"my_additional_extensions_block"`

xii.]

xiii. },

5. interface difficult
    a. LunarG comment: That's probably the Profiles layer settings and that a lot of the features are using command line scripts. There are possible improvements here that can be made.

Open-ended Feedback : RenderDoc

1. Renderdoc's missing support for the lastest extensions and features is becoming a bigger problem, e.g. descriptor buffers, mesh shaders, ray tracing. gfxreconstruct can be used instead in some situations, but it's not a full debugger (at least not yet?). Having debugging tools is very important and I hope in the future we will get one that supports all features in a reasonable timeframe.
2. Sponsoring / adding ray tracing support to RenderDoc would be a great help
3. I know its not part of the vulkan sdk but there are not much debugging tools for the ray tracing extensions yet (e.g. in renderdoc)
4. bugs in RenderDoc affects my effectiveness and productivity when developing to Vulkan
5. Lack of ray tracing support in RenderDoc affects my effectiveness and productivity when developing to Vulkan
6. Missing ray tracing debugger tools (there is no remderdoc support. Nsight is available though).

Open-ended Feedback : High level Shader Language / Compilers

1. GLSL Language not moving forward; need more modern shader language (LunarG comment: There isn't enough industry interest in advancing the language features of GLSL for there to be sponsors to move GLSL forward. There are other promising solutions such as slang (https://github.com/shader-slang/slang)
    a. Next gen shading language would be on my wishlist
    b. need better shading language options/productivity with out compromises
    c. I'm worried about the shader ecosystem. GLSL seems to be in maintenance-only mode, dxc has a very complicated codebase and the Vulkan/SPIR-V support still seems to be pretty experimental in certain areas. It's hard to judge whether there are other viable shader compilers/languages.
    d. A new language designed for larger codebases would make things easier and it could still be compiled to SPIRV. I briefly looked at Slang it might be what I'm looking for.
    e. glslc is shipped without a class necessary for include support. It would help to have that on board.
    f. GLSL language is constraining, e.g. passing a buffer as a function arg requires macro workarounds, etc.
    g. Mostly GLSL being awkward to use (no generics, weird BDA syntax), but I have not tried slang yet.
    h. GLSL is basically as up-to-date as I need it to be for spir-v features, but lacks basic programming productivity features (operator overloading and generic programming). we are forced to use it regardless because HLSL doesn't support our use case. We've considered doing RustGPU, even though our code-base is C++ because of how awful the other options are, but it's not clear what is done/usable, and some of their code transformations result in poor performance, you can use SPIR-V from it, and unlike GLSL, using it isn't in indefinite limbo.
    i. GLSL is old and no longer feasible for effective shader development
    j. All these shader languages are either bad or immature. Slang just got pointers, but they crash the compiler. I still need to build and try Vcc.
    k. Lack of good shading languages, every one has horrible compromises. HLSL has better programming features, but sticks with out-dated HLSL SV model from the 90s, and doesn't support absolutely critical features like physical addressing (buffer_reference(2)) properly. I also can never trust it to be up-to-date with SPIR-V features.
    l. We need to improve HLSL support for Vulkan
    m. C++ style support in shader programming
    n. The need for good shader languages. GLSL is old and does not get updated (aside from when new extensions are needed for a new Vulkan feature/extension). GLSL is a burden to work with, but I must wait with HLSL since it still needs to properly support GLSL_EXT_buffer_reference. It was apparently admitted that GLSL would not be developed anymore, which means

that the entire API rests upon Microsoft's HLSL development to also hack in the new Vulkan features, which is a ""community contribution"".

o. It would be very nice to have an actual C++-like (minimum 17, ideally C++23/26)) language to work with it (The Vulkan Clang Compiler looks promising, but it will take very long until it is properly usable and efficient, and if there is no official support from Khronos/LunarG, I do not see it becoming a safe bet unfortunately).

2. VCC (LunarG comment: VCC is not yet endorsed by many key developers and wider spread adoption in the industry would be needed before it is included in the Vulkan SDK).
    a. Maybe my #1 blocker nowadays is just headaches with getting large, complex shader code to perform well. glslc and glslangValidator are designed for compiling relatively small shaders. This article does a great job of describing the problem: https://therealmjp.github.io/posts/shader-permutations-part1/ https://therealmjp.github.io/posts/shader-permutations-part2/ I opted for dynamic branching / a GPU bytecode interpreter to solve the shader permutation problem, but it was very difficult to write the code in a way that glslc would compile it and the code would still be performant when lowered to machine code and executed on a (NVIDIA/AMD) GPU. I would be very interested in seeing more modern language features such as real function calls and pointers in GPU shader code, like what VCC allows for: https://xol.io/blah/introducing-vcc/"
    b. Though it is at an early/experimental stage of development, I think the Vulkan SDK should take a serious look at VCC as an future alternative to glslangValidator and glslc for translating human-readable code into SPIR-V. One of my biggest headaches as a developer has been fighting performance issues in larger, more complex shaders due to how GPU code gets compiled. VCC might help relieve me of these headaches. For example, with real function calls, I might be able to avoid having large complex ray-tracing / dynamic branching code get inlined/copy-pasted everywhere in the SPIR-V to the point that the shader's performance tanks. https://xol.io/blah/introducing-vcc/ "
    c. Vcc rulez
    d. Put Vcc in the SDK
3. DXC
    a. The myriad of DXC bugs
    b. Bugs in DXC
    c. HLSL to SPIR-V translation is quite problematic at the moment. Lot's of DXC bugs that never get fixed.
4. Other
    a. I don't know where to find information about how glslang gets compiled. Does it get SPIR-V-Tools (i.e., supports HLSL) ? But maybe I'm just blind, so sorry if it's documented somewhere."
        i. LunarG comment: glslang supports HLSL (shader model 5 and below)
    b. I don't think the glslang c interface gets exported in the SDK. I may be using a version too old (I didn't check each version after 1.3.243). I mainly work in C++, so this isn't a big deal for me, but this may affect C developers.

    i. LunarG Comment: glslang has a C interface and it is documented in the README

c. Shader authoring is a somewhat painful experience in glsl. Few (if any) IDE extensions have real language support beyond syntax highlighting, nor do they understand the various extensions Vulkan adds to the language.
d. consuming GLSL shaders from OpenGL
e. OptiX won out over Vulkan due to the shading language. Vulkan is not seen as a serious alternative to OptiX (/CUDA) within my company, despite OpenGL being viewed favorably.

Open-ended Feedback : Developer Tools

1. Vulkan headers
    a. huge VulkanHPP header (long compilation time)
    b. When developing in C++, I found the lack of modern constructs provided by vulkan.hpp a bit painful. But the design of these C++ bindings is based on the fact it should reflect the explicitness of the underlying C API, so I wouldn't complain about this.
    c. Nothing directly linked to the API specs or tooling in themselves. However, I realised that my compilation times go wild very quickly when using vulkan.h or vulkan.hpp (I don't have a big config'). I ended up writing my own vulkan.h, which instead of including platform-specific headers (windows.h for example...), just forward-declares the needed types.
2. Abstraction Layer, boilerplate utility
    a. making API to abstract vulkan stuff
    b. The cost of getting to a blank frame rendering. Once you have a frame running with the underlying framebuffers everything becomes really easy. However, getting to this point takes so long and if I want to decouple my projects I have to write this boilerplate myself. There should be platform specific defaults/helpers at some point to welcome more junior engineers into VK. The more adoption VK gets for learning purposes, more consideration it will receive to be used in the game industry. DX comes with all sorts of helpers for example. Including the shader binding table.
    c. Better higher level abstractions and easier integration with managed languages, now that OpenGL is frozen in 4.6. It is no accident that people are looking into WebGPU instead of Vulkan for native coding, when they aren't game engine developers.
    d. Vulkan still needs a huge amount of boilerplate code to set up.
    e. The amount of boilerplate code required oftentimes
3. Lack of good shader debugging
4. Debugging compute GLSL shaders with ray queries is mostly guesswork at the moment. In order to properly debug our shaders, we've ported them to the CPU (with some preprocessor magic to have the same shader source for GLSL and C++ code). While this works, it isn't productive.
5. runtime tool to show buffer and image content during app debugging
6. Poor Wayland support
7. maybe, to optionally install cpu vulkan device with the SDK (?)
    a. LunarG comment: It is intentional that the SDK does not install Vulkan drivers. This is the responsibility of the IHV.
8. enabling/disabling particular features and extensions in a layer (for app testing purposes)
    a. LunarG Comment:
        i. The VK_EXT_layer_settings API allows you to enable/disable features in a layer. See this whitepaper for more information:

https://www.lunarg.com/wp-content/uploads/2024/01/Configuring-Vulkan-Layers-LunarG-Christophe-Riccio-01-16-2024.pdf

ii. Vulkan Profiles will allow you to change the reported Vulkan features and extensions via the VK_KHRONOS_Profiles_layer. See this whitepaper for more information: https://www.lunarg.com/wp-content/uploads/2024/02/The-Vulkan-Profiles-Tools-LunarG-Christophe-Riccio-02-01-2024.pdf

9. changing order of enumerated physical devices or hiding some of them in a layer (for app testing purposes)
    a. LunarG comment: Implicit layers delivered by IHVs can do this. As well, vkconfig (Vulkan Configurator) allows you to select a specific physical device.
10. The synchronization could also use some work. It can get very tedious to try and remember what to use. It would be extremely useful if we had a visualization of multiple frames with a dependency graph.
11. ray tracing debugging would be on my wishlist
12. I want to inform that debugging on linux environment is bit complex, so if we include customized tool for debugging on linux will be plus
13. I'm mildly interested in seeing further development of MoltenVK. If the Apple Vision Pro takes off, it might be nice to target it with my Vulkan VR software, though I don't even have a Mac OSX machine at the moment.
14. I have used a number of the LunarG tools in the past, but for our current project we are using our own in house tools. I did not work on these, so can't comment on how they were implemented.
15. I love Vulkan and I won't give up on it. I will maintain it by myself if I must. It deserves to have so much more and in order to do that we must make it friendly for newcomers. No one should be going through 800 lines to draw a single triangle on the screen. There are definitely sensible defaults but for each platform. So we should ask the platforms to supply them.
16. Well, share kernel_slicer again https://github.com/Ray-Tracing-Systems/kernel_slicer i have a paper about it https://www.semanticscholar.org/paper/kernel_slicer%3A-high-level-approach-on-top-of-GPU-Frolov-Sanzharov/9b53f4dc9c6e7589bed5c1b6fc91ea79867e2f14?utm_source=direct_link please e-mail me of you can't access paper text
17. please create a C API for vk-bootstrap, just like VMA has a C API. i need it for writing vulkan with zig and it makes starting with vkguide difficult.

Open-ended Feedback : GPU crashes/hangs

1. Debugging GPU crashes/timeouts is always very tedious. The only reliable way that works so far (also for older hardware) is to have a reliable repro, and bisect where the issue happens by disabling various parts of the code.
2. Debugging DeviceLost errors.
3. Difficult-to-debug GPU crashes
4. DEVICE LOST error, Nvidia's aftermath does provide information, but many times the info is of no use.
5. Unpredictable GPU crashes on AMD's integrated cards

Open-ended Feedback : Android

1. Lack of good Android tools impacts my ability to be effective and productive while developing to Vulkan
2. As a mobile developer the issue is that drivers are rarely updated. Whether a political or technical solution is necessary it would be really nice to be able to use shader objects and all the other improvements since 1.1 on mobile without losing 95% of the devices in the wild.
3. Bugs in drivers
4. I'm shipping on a lot of phones that no longer get driver updates. Reporting bugs will help future phones, and some current ones, but I still need to work around the bug in my code to ship on the older phones that people are still using
5. There are too many vendors ...
6. I do report, but mostly to get advice on how to work around it - I do not expect mobile vendors to fix the issue.
7. Android driver quality is a huge issue. When will Google do something about it? Advert revenue is sweet but the luck of updates is slowly killing Android."
8. Make this better: https://github.com/googlesamples/android-vulkan-tutorials It is really bad currenty.
9. I need a Dxc compiler library for Android. I couldn't configure project using CMake. HLSL language this is the future of Vulkan.
10. My main issue is currently the monolithic vulkan SDK to learn basic android samples.
11. The lack of driver updates on Android (or even other OSes) which means we have to target versions of Vulkan that aren't the latest.

Open-ended Feedback : MoltenVK

Provide any additional feedback you would like to provide regarding MoltenVK.

A comment from Bill Hollings: We at MoltenVK appreciate hearing valuable feedback like this, and we always welcome direct and specific feedback on the MoltenVK Issues List at https://github.com/KhronosGroup/MoltenVK/issues

1. It is essential for us to support Mac, so steady development is very welcome!
2. Double precision for modeling/simulation… though I know this is a Metal limitation.
3. Keep up the good work.
4. MoltenVK is brilliant - it would great if it could keep up with "native Vulkan" without too much lag.
5. CMake build support for MoltenVK so that's easier to integrate in our own CMake based product build.
6. Linking against the Vulkan libraries should be simplified, this is sometimes ridiculously complicated. Installation should also be simplified.
7. MoltenVK originated as a developer library to reach out to more developers. This is still the way how we at Blender uses it. It is a tool to validate that our shaders and test cases work on other platforms without booting into another machine. We don't expect to use MoltenVK in a release due to maintainability aspects. Context: In Blender studios/professionals can extend Blender using Python, we try to solve any missing feature/texture format usage there is to add a smooth transition from our OpenGL Backend to Vulkan. Adding workarounds for missing features would include most of the workarounds we have added for Metal to be included for Vulkan as well. This would increase maintenance, which leads to less development of new features. This adds pressure to the small group experienced in these topics.
8. the SPIRVCross code-base is pretty scary given how load-bearing it is. it's full of one-off hacks tuned for applications and contributing to it is not that easy (I've gotten some patches in to begin refactoring SPIRType)
9. What can i say... I'm happy that it exists but unhappy about the poor quality of releases.
10. It's an astoundingly good piece of software that does a very difficult job extremely well. The ability to ship on Android & iOS with a unified graphics code base is a massive benefit. Thank you!
11. do not for the moment
12. Tile shaders for Vulkan. Ie. compute shader access to tile memory.
13. just not good enough to actually continue MacOS vulkan development at this time (too many restrictions compared to Windows). I see that also limited mac GPU capabilities are a problem. Could emulation layers be solution? (would work for development, not production, but ok for now)
14. DebugPrintfEXT() support in shaders on macOS (i.e. via MoltenVK).
    a. LunarG comment: Yes, we know about this gap. See this issue: https://github.com/KhronosGroup/MoltenVK/issues/1214

Open-ended Feedback : Vulkan Ecosystem Documentation

If you use the docs.vulkan.org site, what suggestions do you have to improve the site?

1. Provide download of man pages for quick terminal use.
2. add images, visualizing concepts
3. Needs more spacing or whitespace
4. Google search brings up links to old docs.
5. Better search functionality
6. dark theme please, my eyes hurt
    a. Vulkan Working Group comment (Developer Relations): This will be coming shortly
7. I think the biggest challenge right now is SEO. When I search for Vulkan things on DuckDuckGo, the spec pages come up and not anything from docs.vulkan.org
8. When I look up a Vulkan function or struct, it's usually because I want to see API docs for it. Things like an explanation of its parameters, or seeing other items in an enum, or seeing related functions. The Vulkan spec works really well for that, but I haven't had as much success using docs.vulkan.org. Searching on the site brings up samples, links to a Vulkan tutorial, the Vulkan Guide, as well as the spec. However, it links to places in the spec that mention the thing I'm looking up, not the spec section matching the exact thing I want. My workflow would be easier if the link to the item in the spec was always at the top, at least if there's a spec entry for what I'm searching for Or honestly, you could just put sections of the Vulkan spec as function and struct docs in vulkan.h. That's be super simple to use and very discoverable
    a. Vulkan Working Group comment (Developer Relations): This will be fixed in the next release. Searches now default to the document you have open unless you explicitly select to search all docs on the site
9. Fix the duplicate search results
10. idk, I don't use this that much
11. Works well for me, no particular suggestions.
12. better use more images
13. More information about performance and use cases for features and regular commands.
14. Since we target Vulkan 1.0 + extensions / Vulkan 1.1 it would be nice to have these versions of the spec somewhere on docs.vulkan.org instead of having to dig in the Vulkan registry :)
15. By default, it should make it easy to find major categories. Instead, it looks like it's a GUI to look at the Vulkan spec (no help at all). However, if you go looking, you'll find links to helpful material. This helpful material should be front-and-center!
16. The site is somewhat slower than using the standalone PDF as it takes slightly more time to load each section. On a long coding session it adds up to quite a bit of wasted time.
17. not applicable
18. Vulkan Basics is written with mobile/OpenGL ES target audience in mind.

19. Improving extension samples. For example Synchronization2 doesn't explain anything. Perhaps also a bit of backstory of the extension + link to its proposal.
    a. Vulkan Working Group comment (Developer Relations): Thanks for the suggestion. We are working on tutorials to fill in the backstory on extension samples and your feedback is useful for our prioritization.
20. There are some dead links like inside https://docs.vulkan.org/samples/latest/samples/extensions/README.html (I should report this one :-))
    a. Vulkan Working Group comment (Developer Relations): We have addressed many of these. If you continue to see more, please open an issue on https://github.com/KhronosGroup/Vulkan-Site
21. VUIDs are endless lists, making difficult to read other important information. Could it be collapsible, for instance? Yes, VUIDs are very useful and they have very important place in docs, and very useful at some moments when you need them. But you are not always interested in them while they take tooooo much space.
22. No suggestions.
23. Sometimes it's difficult to determine what a specific feature is for, links to the proposal help here, but a brief sentence of explanation would be nice in the spec itself.
24. Faster loading is always appreciated. It's fine most of the time though.
25. Would be nice if it didn't promote writing ancient C++ or at least took basic core guidelines into account.
26. More example code
27. More discussion of the motivations behind the decisions made in the Vulkan API, more focus on the practices one should adopt to write "modern Vulkan" targeting 1.3 and higher.
28. Specific examples like Pipelines Shaders etc.
29. It needs more tutorials
    a. Vulkan Working Group comment (Developer Relations): We will continue to include tutorials on all new samples and are continuing to add additional tutorials to the Khronos Vulkan Tutorial on docs.vulkan.org
30. dark mode
    a. Vulkan Working Group comment (Developer Relations): Coming soon
31. Practical end-to-end example of a barebones 3d engine.
32. A beginner tutorial for ""modern"" Vulkan 1.3+ (or whatever moltenvk supports).
33. Many links are still going back to the old spec
34. if i google ""vkCreateSampler"", top hit is this: https://registry.khronos.org/vulkan/specs/1.3-extensions/man/html/vkCreateSampler.html . but then, there is slightly more info here: https://registry.khronos.org/vulkan/specs/1.3-extensions/html/vkspec.html#vkCreateSampler. and then slightly more info here: https://docs.vulkan.org/spec/latest/chapters/samplers.html. could all of this info be in one place instead? (ideally with subsections like docs.vulkan.org and not everything in one big place because its slow to load/search)
35. add chapter number and increase menu unfold level.

36. I don't have, it look good
37. Try mimics dx12 doc
38. Links on site to possible snippets, maybe even github backed.
39. Links in the Lexicon/Glossary to other terms defined in the Glossary when they are used.
40. Dark mode please.
    a. Vulkan Working Group comment (Developer Relations): Coming soon
41. Some exemple snippets with only what's needed for it to works
42. So far, not much to improve there! Or, perhaps, one thing could be slightly improved... When showing some code snippet to modify or add to existing (previously written) code, the docs could be a bit more explicit on where the new code should be plugged in... as the written code grows with every new section of the tutorial, it might get a bit tricky to understand where the new code should go. However, the ambiguity happens only here and there, most of the time, if one has properly understood the code written so far, it's pretty obvious where the new code should be inserted; in a few cases, however, it is a bit more difficult, especially when implementing completely new or optional functionalities (such as validation layers).
43. No suggestions, seems to be well-put together as far as I can tell :)
44. Honestly, compared to msdn your documentation is amazing. I know many Vulkan applications use hlsl and compile it down to spirv (mine included). Hlsl documentation is very outside your wheel house, but is the weakest part of the dev cycle in my opinion, maybe you can coordinate something there.

Other documentation/tutorial related open-ended feedback from the survey (consolidated here):

1. What prevents you from being effective and productive while doing your Vulkan development?
    a. The documentation fragmentation issue. Need to read n+1 extension specs to get a coherent understanding of how to do X.
    b. Hlsl documentation primarily
    c. No coherent and central documentation for vulkan design principles/architecture. Central concepts are hard to grasp as they're fragmented throughout the official documentation.
    d. An official (digital) and maintained book(See Professional CMake by the developer of CMake) would be very handy. I'd love to pay for that."
    e. Lack of tutorials / non-specification docs
    f. Also companies who publishes things as PDFs, which are never updated and contain out-of-date and just plain wrong information... cough... cough... LunarG
    g. You do not have proper textbooks for setting up a game engine or any viable project. You simply make a spinning pyramid and that's it. The majority of beginners probably stop there.
    h. Lack of Doxygen-like comments for headers which can IDE shows what this symbol does and parameters without looking documentations.
2. Is there anything else you would like to share?
    a. Its not so much lack of documentation rather than what documentation is produce is so shit that it should not be subjected upon anybody looking for answers, and

whoever is writing it should be prevented from producing anything that another human will gaze upon.

b. Vulkan needs more improvements on organizing documentation and https://docs.vulkan.org is a great step on that.
c. Now we desperately need educational resources to make everyone turned off by Vulkan 1.0 renderpasses and whatnot to realize things have changed.
d. More tutorials for beginners through to advanced
e. Please provide learning textbooks to follow. Ideally, a series of official videos on Youtube would be extremely helpful. If not, it's just going to be the 7 people who know the secret of Vulkan and no one else.

Open-ended Feedback : Vulkan Samples

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

1. best practices for pixel art rendering, including batching, texture atlases, maybe sparse textures
2. Post-processing / multisampling with dynamic rendering
3. samples using new extensions would be interesting
4. Samples that cover new extensions would be very useful.
    a. Comment from the Vulkan WG (developer relations): New extensions get samples on a regular basis for selected extensions.
5. I don't engage with the samples enough to know which ones are and aren't available, so I'm not qualified to answer this question. In general, I think that samples that show how multiple Vulkan features together, especially multiple newer features that may only be in extensions, are the most useful
6. More bindless
7. A Vulkan layer demonstrating best practises for layer development
8. Vulkan SC compatible versions of some of the basic samples might be good.
9. more samples on the gpu address extension and more explanation for the current samples as some samples doesn't have much explanation just code while it's useful it could be much better with some explanation.
    a. Comment from the Vulkan WG (developer relations): We now include tutorials with every new sample and plan to backfill previous samples over time.
10. Rust sample ?
11. 2. VK_EXT_host_image_copy
12. 3. more samples for VK_KHR_ray_tracing_pipeline
13. 4. VK_EXT_opacity_micromap
14. 5. more samples for compute shaders
15. More examples would be great: - how to use volk - how to use vkbootstrap - samples showing basic things like getting a window up, using extensions, how to fix validation errors
16. and memory like VK_STRUCTURE_TYPE_IMPORT_MEMORY_WIN32_HANDLE_INFO_KHR, so as close as we can to Direct Storage of DirectX
17. Better font rendering samples could be useful
18. 1. How to add text overlays using FreeType and TrueType fonts. 2. How to create 3D letters from the above.
19. Frame pacing / present timing. bindless textures and resources
20. Stutter free rendering using fixed framerate timing. (DisplayLink on macOS, google display timing on Linux, ???? on Windows)
21. More pure C examples.
22. More complex raytracing samples. It is lost on the beginners how someone would get into their bound (or unbound) memory using the instance, geometry and primitive ids to

extract data. So many complications here. Also a strong helper class for the shader binding table management is a must!

23. "Best practices" examples for each minor Vulkan version.
24. Optimize shaders in GPU
25. Multi threading resource loading.
26. Interop with OptiX. Larger ray tracing (i.e. multiple objects, multiple materials, multiple ray types) - The current examples take some short cuts that don't easily translate to larger projects
27. Other languages besides C and C++.
28. complex builds with cmake: dev/prod, shaders, including OpenXR and other Khronos libs
29. gi
30. How to draw 2d shapes, polygons, beziers, ..
31. RTX, more and complex render and subpass samples efficient CPU handshake samples and lastly sample for Next Windows RTXIO or differnt naming DirectStorage data transfer examples
32. I am aware of the Kompute project. I think this is a very good initiative, and potentially viable alternative to proprietary APIs like CUDA. As far as I know, I don't think "pure" compute samples are put into the spotlight for Vulkan Samples. By "pure", I mean "not strictly related to graphics". I believe we have N-body compute and post-processing effects, but nothing that would get the interest of people doing parallel computing.
33. The vulkan kube was awesome sample for learning the basics
34. Multiple shadow maps
35. Skinned Mesh Animation, IK
36. no more triangle example, use real models via gltf and interface that allows using the sdk easily, e.g. what if somebody wants to develop an IG
37. VK_KHR_cooperative_matrix
38. Some Vulkan Interop things. Or samples that inspire implementation ideas.
39. Some API examples to get the best available performance possible with Vulkan when they are available to use and some binding between shaders (glsl & hlsl) and Vulkan
    a. Comment from the Vulkan WG (developer relations): We are working on adding HLSL support to the samples with the goal of having both GLSL and HLSL shaders for most samples. We should have the first HLSL shaders in place soon.
40. I have not checked the Samples yet.
41. Not sure I've seen many samples for Vulkan Video Encode/Decode yet?
42. Video Encoding and Decoding;
43. more video encode,
44. 1. video decode/encode (VK_KHR_video_queue, etc.)
45. Comment from the Vulkan WG (developer relations):
    a. Requests that already have implemented samples or are WIP:
        i. ray tracing pipeline
        ii. Ray tracing
        iii. ray tracing
        iv. Ray tracing, both simple and advanced

v. OpenGL Interop
vi. Vulkan Raytracing samples and saschawillems,etc..
vii. Buffer/Texture/Semaphore sharing with all other graphics APIs
viii. Proper way to handle sync with swapchain objects, especially in complex cases, like this: https://github.com/KhronosGroup/Vulkan-Docs/issues/2007
ix. Please create simple self-isolated sample just for android. In other words do this, but for ANDROID: https://vulkan-tutorial.com/
    1. Google comment: A simple hello-vulkan sample is available here https://github.com/android/ndk-samples/tree/main/hello-vulkan and can be downloaded directly from Android Studio by clicking on the 'Import an Android Code Sample' button

Additional feedback about samples from open-ended feedback (consolidated here):

1. If you use the Vulkan SDK, what suggestions do you have for the Vulkan SDK?
    a. A lot more working samples, for every extension
    b. A bit more examples to show how to start interacting with Vulkan and some basic rendering shaders
2. What prevents you from being effective and productive while doing your Vulkan development?
    a. Samples
    b. Lack of examples on tutorials on more advanced subjects. Even beginner tutorials don't cover the very basic parts
    c. There are fewer samples for newly released extensions.
    d. Vulkan-Samples are useful (thanks, Sascha) for individual features, but they are difficult to build a comprehensive understanding about a feature.
    e. Lack of good tutorials on new extensions.
    f. "Lack of standalone simple vulkan samples. Google provide very bad ones, poorly writen. Don't do this: https://github.com/googlesamples/android-vulkan-tutorials But do this for android: https://vulkan-tutorial.com/
        i. Comment from the Vulkan WG (developer relations): The Vulkan samples work across many platforms including Android. This includes a basic "hello triangle" sample that works well on Android.
3. Is there anything else you would like to share?
    a. It really is the lack or working examples for all extensions. Often you will find an example but only after long searches with an engine. It should be minatory to have working sample, in Github. We often feel extensions are going in, to the sole benefit of one developer

Open-ended Feedback : Vulkan API

LunarG comment: This feedback has been shared with the Vulkan working group. LunarG is unable to make any editorial comments.

1. API Complexity
    a. I love the explicitness but sometimes it's too much.
    b. Too many ways to do simple things.
    c. having trouble visualizing things, and being told to use OpenGL to learn graphics programming which feels way too magical
    d. Having to constantly check the spec, but that is just the nature of a very explicit API like Vulkan.
    e. The needless, stupid complexity of the API in its entirety. Scrap it all and start over.
    f. The complexity of the spec and all the cases to handle depending on hardware.
    g. Right now Vulkan gives us an infinite number of ways to do everything, but no way to query the GPU for which of those ways is optimal on a given piece of hardware. I desperately wish for some extension that's about a half step higher level from vulkan, which would let me have the control vulkan offers where I care, and let me delegate the things I'm indifferent about to the GPU (such as texture loading and unsupported format conversion).
    h. Better understanding of synchronization (fences, semaphores, setting up sensible stages at which to synchronize)
    i. Renderpasses
    j. Missing Features
    k. So so so many API with each so so so many different parameters, when starting to learn Vulkan & 3D graphics you are never sure if you use the API decently and if you use correct parameters even when Vulkan validation says nothing problematic.
    l. Akward parts of spec. From recent memory: only Vulkan missing zero-length SSBO, for no good reason.
    m. Too much to learn
    n. verbosity
    o. Lack of deep GPU understanding
    p. Vulkan is too difficult 🥺
    q. if any issue occurs finding component
    r. Having to look up stuff in the specs or online constantly…
    s. Constantly having to look up the arguments to certain constructors of classes in the vulkan-hpp library.
    t. way to difficult to use, learning curve is huge. I know is hard to develop SDK but not everybody has the time to be a master on computer graphics
2. API enhancements
    a. need VK_EXT_shader_object for RT pipelines. MaterialX/MDL/generated shader trend causes many shaders -> long pipeline compilation times

b. need cross-vendor VK_NV_ray_tracing_invocation_reorder extension. The extension does _a lot_ more than just reordering; perhaps it should be split into two. F.i. it allows to execute multiple hit shaders on only one traceRay call. This can be used to reduce divergence and payload sizes."
c. Ray tracing is the best thing that happened to modern graphics, I would like further development in this direction.
d. Although it affects me less, something similar like ray tracing shader tables or even better: callable shaders form 64 bit integers, similar to buffer reference, could eliminate the need for sorting for render calls completely (if shader tables, then it should be able to play nicely with draw indirect, so the calls could still be culled on the GPU). "
e. Compatibility with D3D12 ResourceDescriptorHeap (Not having truly bindless support).
f. Also the lack of a buffer reference equal for images and samplers. NVidia already has it, and uses it for DLSS (VK_NVX_image_view_handle). If there was something similar for images (which should be possible considering NVidia already has such an extension + OpenGL's bindless textures) it would make rendering a lot more efficient in combination with buffer reference.
g. ResourceDescriptorHeap like system please!

3. other
    a. I think there is too much fragmentation due to a proliferation of extensions. Vulkan should be faster about making things required core features and removing old functionality in new versions to avoid API cruft. Lots of new extensions (dynamic rendering for example) are big improvements but it's hard be able to write code that depend on them existing which partially defeats their purpose in many situations. Layer based solutions like for sychronization2 help since they move the burden from the developer into the SDK
    b. I appreciate the power of the Vulkan API.
    c. This VUIDs are technically very useful, but sometimes, they make doc in pdf reading very unfriendly. I found myself using very old Vulkan doc pdf instead because I can find more easily what I want.
    d. OpenCL and Vulkan using Spir-V but not accepting each other's code doesn't make sense to me
    e. Vulkan is way, way worse than DirectX.
    f. The last major thing we need for parity is device side enqueue, however we see AMD has already submitted a proposal for this https://github.com/KhronosGroup/Vulkan-Docs/blob/main/proposals/VK_AMDX_shader_enqueue.adoc with VK_AMDX_Shader_enqueue. We are very excited to see a cross platform solution to the problem come to fruition, Vulkan doesn't even have parity with Metal in this regard.
    g. need physical pointers to shared memory, or at least a method of loading type-punned data into and out of shared memory in fixed size blocks, which is one of two things left causing issues with parity between other code bases. We lose to cuda and metal in some analogous applications *purely* because of this

issue, we'll have a large thing that needs to be read in, and used for all threads in a local workgroup, which we would normally use all threads to cooperate to load in, but can't because we aren't able to instruct threads to type-pun shared memory/workgroup memory in SPIR-V as some other type (treating a incoming struct as an array of u32s).

Open-ended Feedback : Other

1. A terrible and toxic community ranging from insanely pedantic spec Nazis to excessively opinionated ideologue trolls who contribute nothing but their terrible and uninformed opinions.
2. segmentation violation from inside vulkan functions
3. Discord
4. The issue I have is more compatibility with directx than vulkan itself, there's a couple features I want to use (like scalar layout and BDA) but because how difficult it'd be to emulate for my directx backend I don't use them.
5. Issues with specifying which installed version will be used. (There are multiple versions of vulkan installed on my machine, one of which had a bug which caused the program to segfault. I ended up stuck on this for a few months, unfortunately.)
6. I would like to have an AI model that would create deqp tests, to test my new driver functionality in complicated cases.
7. Vendor support; Not all vendors support the latest extensions/features, minimizing the extensions/features you use increases the user base you can support.
    a. Can actually be addressed as a feature that I'd like to see in https://vulkan.gpuinfo.org. That is, being able to filter by GPU age & Vendor & Profile & OS. LunarG comment: Sascha is investigating what may be possible here.
    b. Suppose you're a developer that wants to support OS X & Y, Vendor X & Y, with medium-end desktop graphics cards made in the last 5 years with the VP_LunarG_desktop_baseline_2023 profile, what subset of extensions/features/limits/formats can I work with?
8. not having a second development workstation
9. Nothing after i have created kernel_slicer to speed-up Vulkan dev. https://github.com/Ray-Tracing-Systems/kernel_slicer
10. Nothing
11. My dogs
12. Nothing really
13. do noy know for the moment
14. Nothing much tbh.
15. As a solo dev iteration time can be slow.
16. Not really, the survey failed to work at some parts, Firefox 122.0 (64-bit)
17. Limited experience
    a. My inexperience as a beginner.
    b. Being a beginner in knowledge about C++.
    c. I still have OpenGL-brain from years of OpenGL and much less experience with Vulkan.
    d. first time using Vulkan
    e. Mainly my very limited knowledge of it, as only recently I started to learn the Vulkan API and its functionalities.
    f. My laziness

    g. lack of time as i can only to it in my free time currently
18. Driver quality
    a. nvidia's linux drivers
19. I use Vulkan for my hobby projects and in no way professionally, but I tend to read to spec. for fun and discover interesting things. Like certain extensions/features. When implementing these features I sometimes come across broken behavior. Like broken validation layers/driver bugs. It surprises me from time to time no-one has reported them yet, but it makes me wonder how much people really get out of what Vulkan has to offer. As a simple example, do people make use of a few shaders that utilize triangles primitives in a single graphics queue and abstract everything else away? Or do they, make use of other primitives and compute shaders, with multiple different queues using extensions/features x, y, z, and rewrite the descriptor set on the GPU timeline? The former is most likely for a various different reasons, (this is all speculation):
    a. - What is taught/online resources;
    b. -- the more/quality teaching resources something has the more likely people will follow that way of programming Vulkan. (Vulkan-samples is a great resource that counters this point)
    c. -- What ain't taught is ill supported, and ill supported things ain't taught.
    d. - Time/exploratory learning; Not everyone takes the time or has the time or the interest to read about the new methods.
    e. - Vendor support; Not all vendors support the latest extensions/features, minimizing the extensions/features you use increases the user base you can support.
20. I'm working on Vulkan bindings for PHP https://github.com/iggyvolz/vulkan-php
21. Push for native Vulkan integration and support on iOS devices rather than MoltenVK.
22. I'm not sure of the hardware capabilities, but it'd be great if VK_EXT_mutable_descriptor_type was available on earlier drivers. I don't see a reason why drivers capable of SM 6.6 with directx couldn't support this, yet on NVIDIA drivers it's basically RTX+ cards only.
23. is there some way can connect webgpu and vulkan? or release web version vulkan.
24. Just a few words to stress the relevance of the questions put in this survey. I think that mentioning relatively recent, not yet well-known tooling is particularly clever and should continue in the next surveys (note: this is the first time I'm answering one).
25. For people developing using bindings in languages that are not C nor C++, it may be interesting to ask what bindings they use. I know there are several for Rust for example, but all are unofficial (I think), so their may be interesting information to gather from such devs too, though I am not sure how much they represent in the community.

Open-ended Feedback : Thank You

1. The progress of the validation layers are amazing!
2. Keep up the good work

3. Thank you for your hard work. I love vulkan SDK, it makes vulkan development so much easier.
4. Vulkan, although non-trivial, is a great API. Thank you for making a great SDK that is portable across platforms for indie video game development.
5. Thank you for your involvement, driving force for Vulkan. Without you Vulkan would not be the ecosystem it currently is!
6. Thanks for the great work and for the ecosystem, its already very good."
7. When visiting Vulkanised it became more apparent to me what your role is in the whole ecosystem. I will definitely reach out to you more often and ask feedback of the challenges we have."
8. Anyway BIG THANKS!!! for improving Vulkan ecosystem."
9. Love the work <3
10. Thank you for all your work on this outstanding SDK!
11. Thanks very much for your service.
12. Great work! I switched from DirectX dev to Vulkan and love it!
13. I believe that tools created by your team are great
14. Thank you for the amazing tools <3
15. I think you are doing a great job with the SDK!
16. With Vulkan 1.3 (or 1.2 and bunch of exts), graphics programming can be fun again!
17. I appreciate the material on the Khronos GitHub repository and I got positive feedback directly by the authors of the tutorial. Great job!