February 2023 LunarG Vulkan Ecosystem & SDK Survey

End of Year Status Report

Executive Summary

Last April, LunarG provided a summary of the Ecosystem Survey Results. You can see that report here. This document is a follow up to that report, providing some updates from LunarG relative to feedback you provided in last years ecosystem survey. BLUE TEXT are the updates.

Highlights from the 2023 Ecosystem Survey

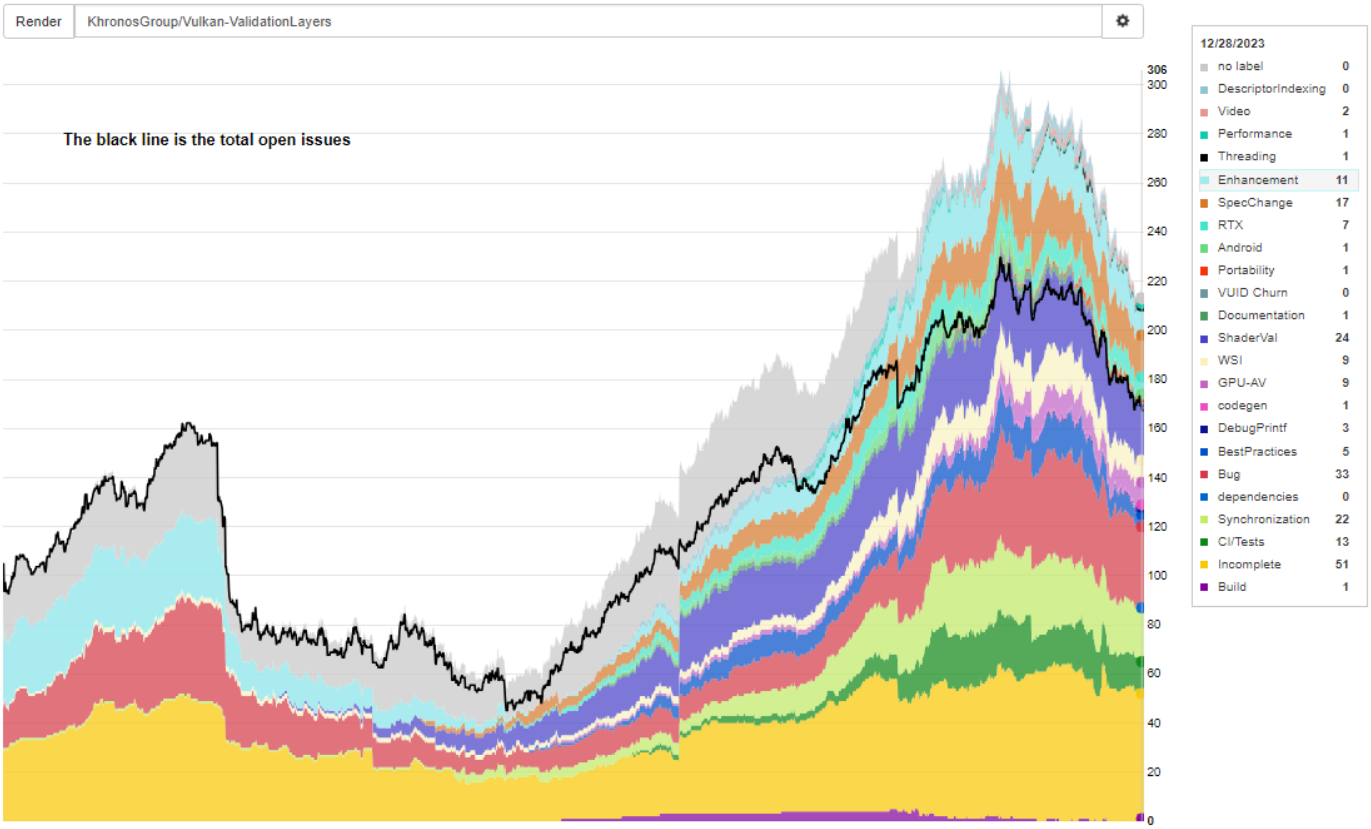
1. There were 275 respondents.
2. Target platforms for applications are (in order of usage) Windows desktop 10, Desktop Linux, Windows desktop 11, Android, macOS, Linux/ARM64, SteamDeck, and Windows/ARM64.
3. Some year over year insights:
    a. Compared to the previous year's survey, the number of folks who have released their Vulkan application for public use has increased from 28% to 36%.
    b. The top 8 use case categories have remained the same for multiple years.
    c. Windows, Linux, Android, and MoltenVK remain the top 4 target platforms (with the same ordering).
    d. glslangValidator (glsl->SPIR-V) or shaderc (glsls->SPIR-V) remains the most used front end from which to generate SPIR-V.
    e. Validation Layer coverage had a significant improvement from “medium coverage” to “high coverage”. This is good positive progress.
4. Validation Layer improvement themes were:
    a. Continue to increase validation layer coverage
    b. Error messages are very verbose and could be formatted better for easier reading
    c. Interpreting errors (finding root cause of my error) is difficult
    d. Improve the performance
5. There was some vocal open-ended feedback about macOS and MoltenVK and getting that platform up to par with the other platforms.
6. Multiple comments/concerns about the shader toolchain (needs more maintenance and enhancement). Should also note that 60% of the population uses glslangValidator vs. 20% of the population uses DXC.
7. In regards to developer tasks, the top items that were the easiest to perform with the existing toolset were
    a. Debugging visual artifacts
    b. Creating Vulkan conformant application code
    c. Debugging crashes
    d. Debugging shader issues
8. In regards to developer tasks, the most difficult tasks to perform with the existing tool sets were
    a. Identifying driver bugs
    b. Debugging layer issues
    c. Debugging Vulkan installation and configuration issues

Potential actions and future priorities

Feedback from this survey indicates that future action would be welcome in the following areas:
Updated with the results achieved over the last year

Validation layers

a. Continue focus on validation layer performance initiative
    i. To improve performance, an intensive refactor of the descriptor indexing validation code has been in progress for the last year. This was the largest performance bottleneck, affecting applications heavily using descriptor indices. The refactor is complete with positive results.
        1. 5% - 30% improvement in application performance
    ii. The refactor resulted in removing the CPU as the performance bottleneck and resulted in the GPU being the bottleneck. This is because the GPU assisted validation uses a wait idle. So we removed the wait idle, resulting in 70% more performance for one of the most intensive users of descriptor indexing (Doom Eternal).
b. Continue github issue responsiveness and filling out coverage
    i. The validation layer engineering team has been very busy this year. We have been driving the open github issues down nicely. The black line is the total number of open issues.



c. Would be nice if time permits: create tools to help developers debug their errors
d. GPU-AV may need some focused attention for more functionality and addressing bugs
    i. **LunarG Note: The refactor currently in progress to improve performance will touch much of the GPU-AV code. After that refactor is complete, additional functionality and bug fixing can occur.**
    ii. Yes, now that the performance refactor has been completed, resources can begin improving GPU-AV functionality, reliability, and performance.
e. Investigate methods to create more readable validation layer error messages.

i. We have begun improving the errors messages (both content and formatting), with a three prong theme:
    1. Provide more of the values that caused the error message
    2. Give better information where each value in the error message is coming from
    3. Better messages when two different items interact with each other
ii. To date, we have improved all of the error messages for core validation. Error messages for synchronization, GPU-AV, best practices still need to be improved.
iii. It will be an ongoing process.

f. A request for a better search engine on Vulkan's website from the error message/number. Some warnings don't have a number/ID and they are difficult to search for on the internet
    i. **LunarG note: Yes, in the specification there are some cases where the spec says it is invalid, but is missing a proper VU (but currently validated anyway). This makes it impossible to link back to the VUID in the specification when an error is reported. About two years ago there were about 80 known cases of UNASSIGNED VUIDs. LunarG is slowly removing them from the specification (replacing them with actual VUIDs) and today the count is around 25 UNASSIGNED VUIDs. Over time these UNASSIGNED VUIDs will be eliminated from the specification. There is a better communication channel with the working group now so future cases will be fixed right away.**
    ii. Update: Continued improvements were made to the specification over the last year and as a result, there are now only 9 UNASSIGNED VUIDs.

Evaluate potential SDK additions

g. Windows 11 support (add to CI environments)
h. GLFW
i. iOS as a target (create Vulkan Loader for iOS) and fuller validation layer support
    i. We have the Vulkan Loader and Validation Layers building and functioning on iOS. The January 2024 SDK is the first SDK to provide iOS as a target in the macOS SDK.

Vulkan Configurator

j. As we design the next major revision on the Vulkan Configurator consider
    i. Ways to assist developers with layer issues
    ii. Ways to assist in debugging Vulkan installation and configuration issues
    iii. Improve the UI for more simplicity

Raise awareness in the Khronos WG for the need for more contributors and developers to quicken issue resolution and enhancements

k. MoltenVK. LunarG will begin doing MoltenVK work in 2024, working to get it to 1.1/1.2/1.3 conformance status.
l. DXC and glslang. LunarG increased resource investment on glslang.
m. RenderDoc

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

Note: underlined items already exist, *italicized items are WIP.* Blue text indicates progress since the survey.

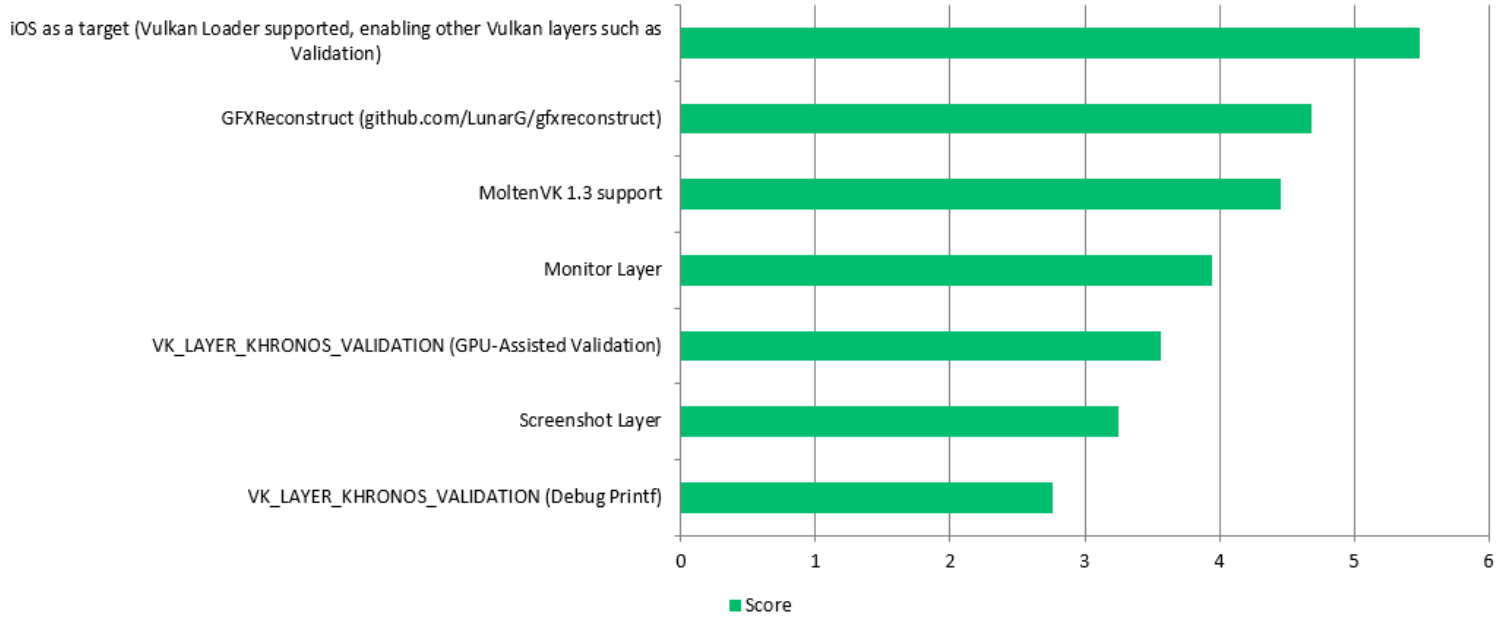
1. Samples focused on performance and soundness best practices.

2. How to use timestamp queries properly.
3. A sample on VK_EXT_descriptor_buffer
4. Some wrong or bad usage samples of Vulkan are also useful. Learning from failures maybe better than learning from success. Comparing the wrong or bad usages with the correct or good usages will help to master the correct the usages.
5. *Examples about how to detect the performance bottle neck and how to fix them and get the improved results.*
    a. Working on support for the Tracy profiler. We will have samples showing how to use it in the future.
6. Frame pacing / present timing
7. Ray Tracing, Pipeline cache, Mesh Shading
8. Vulkan SC
9. I have never used Vulkan samples as learning resources, but maybe:
    a. How to use VK_EXT_descriptor_buffer properly
    b. *How to expand ""render pass"" over multiple command buffers with VK_EXT_dynamic_rendering*
10. Multi Copy Engine Synchronization with Main Rendering
11. Vulkan Samples using VMA, VulkanHpp and best practices additional to the ""Raw""-Vulkan samples.
12. More samples about timeline synchronization. etc."
13. macos example using headers from the sdk + loader + VK_EXT_metal_objects + VK_MVK_moltenvk extensions. (the latter one is written as extension but actually isn't ... apparently?)
14. Compute shaders
15. Video decoding and encoding, "bindless" drawing
16. More specific DirectX->Vulkan conversion samples.
17. video encode/decode
18. *mesh shader*
19. Handling Renderpasses and Subpasses
20. Bindless Descriptorsets
21. Performance oriented samples for desktop apps
22. OpenGL-Vulkan interop
23. Signed Distance Field Based Global Illumination
24. Swapchain recreation for embedded systems
25. Not sure
26. Deferred renderer, Gpu based rendering and async compute
27. More rtx features/samples
28. VK_EXT_descriptor_buffer
29. *Subgroup operations (I don't think there was a sample for this)*
    a. *In development*
30. Sparse images and sampler feedback
31. YUV (with and without the sampler conversion ext)
32. In general more tutorials on Vulkan compute
    a. See Compute Shader chapter in the Khronos Vulkan Tutorial (docs.vulkan.org)
33. I think examples that showcase popular extensions would be beneficial, especially for beginners
34. *Use of video encoding and decoding*
35. VK_EXT_DESCRIPTOR_BUFFER
36. *Better video samples that don't require Nvidia proprietary dlls*
37. An abstraction that leverages SM6.6 style bindless

38. Vulkan SC, *Vulkan Video*
39. VK_EXT_mesh_shader *with HLSL!*
    a. *Working on HLSL flavors of shaders*
40. using VK_GOOGLE_hlsl_functionality1
41. using VK_GOOGLE_user_type
42. *Sparse resources*
43. More synchronisation examples and use cases for performance enhancements (other than synchronisation for necessity) - outlining where synch could degrade instead of enhance.
    a. See the Multi-threading_render_passes performance sample
44. Best practices
45. *OpenXR + Vulkan + OpenCL integration*
46. A larger example application using vulkan_raii.hpp that combines all of the current samples and shows how to e.g. render multiple textured models with varying shaders/descriptor sets in a basic render loop (with swapchain, image acquisition, etc.), preferably in as compact code as possible and with minimal references to external files (like the confusing vk::su namespace). Such an example would be very useful as a reference for beginners who haven't yet fully figured out how all of the different parts of the API fit together, and would also serve as a great sanity check for more experienced developers who want to make sure they are using various features in the way the API designers intended. It would be extra useful if the sample included solutions to some common problems such as how to handle window resizing.
47. Don't know.
48. Proper SI usage.
49. Android OpenXR interop
50. Texture and Mesh Streaming
51. Ray tracing, gpu compute/NN acceleration
52. Samples (or even better: one big sample) that show(s) how to design an efficient and (easily) expandable Vulkan renderer.
53. Something about raytracing for both real-time and offline rendering.
54. Instanced rendering
55. Multipass rendering
56. Usage of HDR
57. Usage of Gbuffer for deferred rendering
58. Setup of piepeline for advanced lighting of many objects
59. Timeline semaphores galore
60. *Swapchain modes and synchronization*
61. Windows resize handling best practices
62. *Compiling code using shader_c with #include support or an example using dxc for HLSL raytracing on windows.*
63. Mesh shaders
64. Lossless GPU (de)compression (E.g. VK_NV_memory_decompression)
65. Anything and everything. all of the time 😂 maybe some optimisations examples would be nice
66. Terrain rendering, more detailed shadow mapping, frustum culling, how to pass array of sampler2D to shader
67. Updating examples with primary usage of newest extensions.
68. Mesh shaders, using VMA efficiently
69. read write texture
70. Synchronization

71. Sample showing how to bind different resources at different frequencies per-frame using different descriptor-set layouts as described in this post: https://developer.nvidia.com/vulkan-shader-resource-binding
72. If there alreafy exists one I couldn't find it.
73. Specifically going from 1 textured square to 2 I found surprisingly difficult and confusing.
74. Gaming
75. FreeSync and minimal latency app - how to render with minimal frame latency, possibly starting shortly before frame deadline; using the FreeSync for low frame latency
76. How to develop minimalist Vulkan application. (The API is incredibly complicated)
77. sample about using different queues, and when using more queues actually hurt performance
78. sample about performance difference between, normal descriptor use, using a mindless model, and using a bindless model using VK_EXT_descriptor_buffer
79. sample about 8/16 bit int usage in shader, not only floats"
80. Idk
81. VK_EXT_descriptor_buffer
82. Suprise me!
83. Complete app able to display data driven scenes that illustrates performant techniques that would scale well in real-world use. Right now it's difficult to surmise from to samples which alternative features are worth optimizing for, and which wouldn't provide a discernible benefit.
84. basic usage examples of:
85. task shaders, mesh shaders
86. An overview of a good practices how to structure a framegraph-based renderer
87. Perfomance and optimisation
88. *Video decoding/encoding*
89. I haven’t seen all that are currently published, so I don’t know what could be missing and added yet.
90. More glslang doc and exemples
91. *More stuff with Vulkan hpp please!*
92. How to manage VkImage Layout in runtime.
93. More samples about multithreading
94. How to instal it use in ON1 plugins for photoshop
95. Recommendations for what features to use and not use with revisions (like 1.0 to 1.1 etc).

If you use the macOS SDK, rank the importance of the following layers/tools that could be added to the SDK. (1 is most important)



As stated earlier, iOS as a target in the macOS SDK is now available as of the January 2024 SDK. This feature enables you to run the validation layers on your iOS device. Other layers are also supported such as the apidump layer or the VK_EXT_shader_object emulation layer.

Beginning in 2024, LunarG has been contracted to work on MoltenVK to help push it to Vulkan 1.3 conformance status.

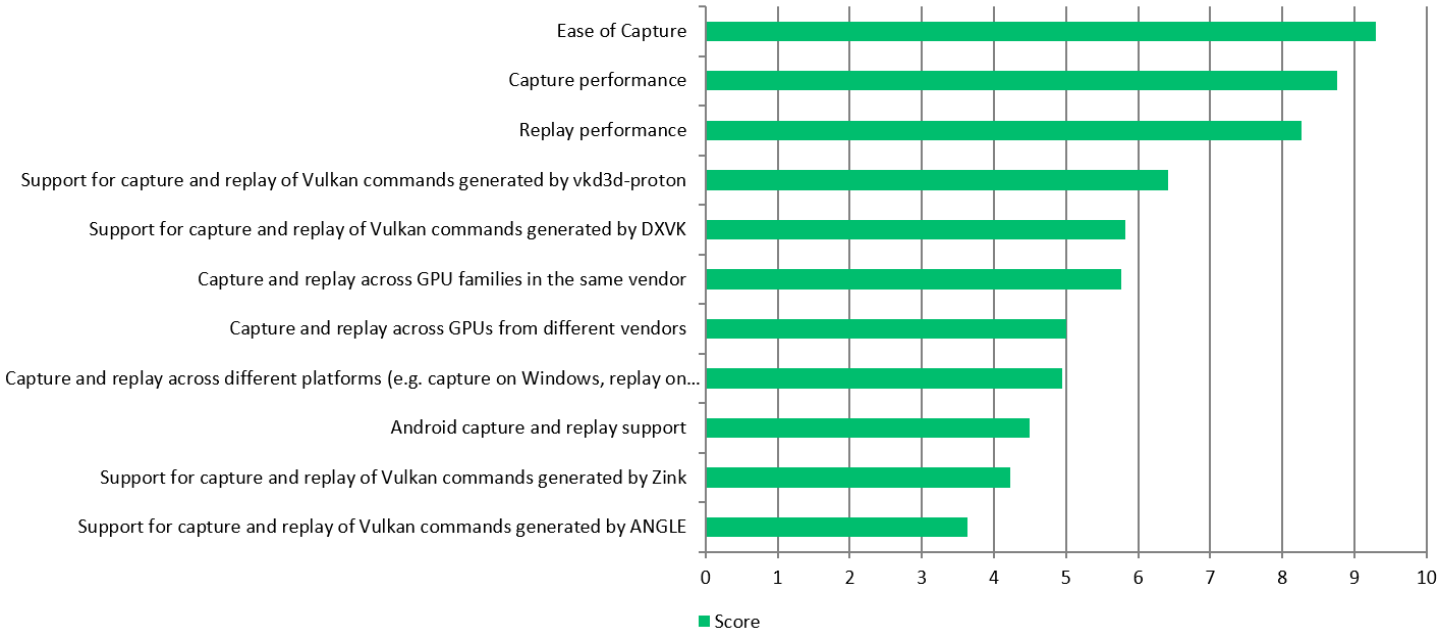
LunarG is currently triaging the failures in the GPU-Assisted validation and the debugPrintf capability in the validation layers on MoltenVK. It is anticipated that MoltenVK may need some functionality upgrades for these portions of validation to work correctly. It has been identified that Metal (in its current definition) maynot support this functionality and as such the implementation to emulate this functionality in MoltenVK is a significant sized project.

The screenshot layer has been ported to macOS and is now included in the January 2024 SDK.

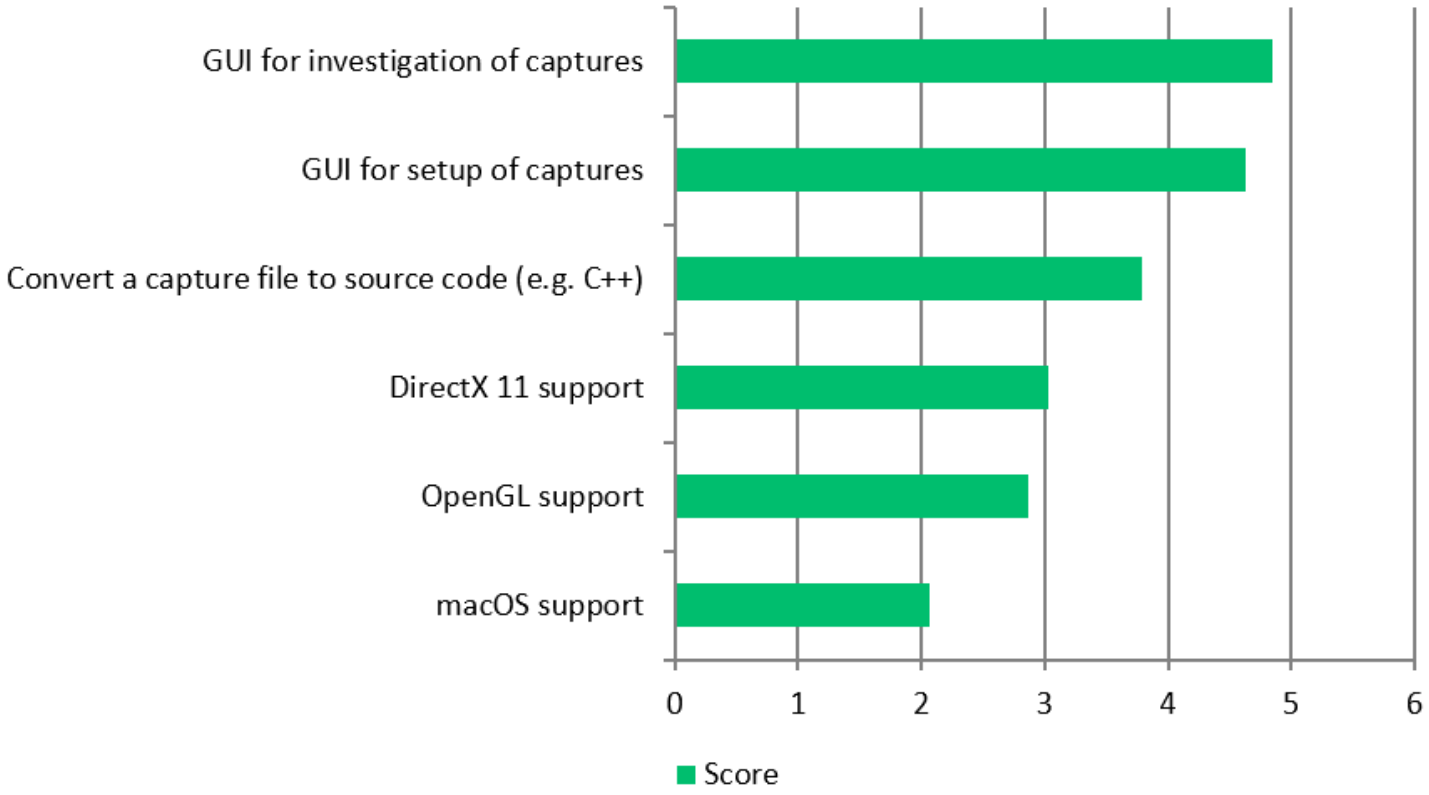
Some good progress has been made to enable MoltenVK to be a better citizen of the Vulkan ecosystem. MoltenVK is a layered Vulkan implementation on top of Apple's Metal API. There are behaviors of Metal that can be adjusted for performance and appearance tweaking that are not available via Vulkan. Mac/iOS developers do not wish to have these features withheld because they chose Vulkan as their primary API. Vulkan application developers need a method to get access to some Apple/Metal only features in a Vulkan conformant method. There are many functions that are currently available when linking to MoltenVK statically or dynamically that do not work when using the Vulkan Loader and Layers. As of the January 2024 SDK, these functions are being marked as deprecated and will be removed from future MoltenVK releases. Access

to these Metal capabilities will be achieved by the VK_EXT_metal_objects, and the VK_EXT_layer_settings extensions.

Rank the following GFXReconstruct improvement areas (1 is most important)



Rank the following possible new GFXReconstruct features in terms of usefulness for your projects



1. MacOS support is now available as a PR in the GFXReconstruct repository.
2. There is a PR for DirectX 11 support, however it is lacking trimming functionality and as such we haven't yet decided to accept it.
3. The ability to convert a capture file to a C++ program is under development. An alpha version of the tool exists in the repository. Development will continue to improve its functionality.
4. The Vulkan Configurator provides an interface for setting up GFXReconstruct options before launching capture.

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

1. A possibility to inspect and modify the content of the capture. Either a visual editor, some text export/import like RenderDoc has with XML format, or just C++ code generation.
    a. "Convert" tool outputs JSON describing all Vulkan commands and metadata and also saves all large BLOBs as files.
    b. C++ code generation has been implemented (alpha version) and will continue to be improved
2. Convert a capture file to source code (e.g. C++)
    a. C++ code generation has been implemented (alpha version) and will continue to be improved

3. After replay, giving some metric data, such as, timing, GPU counter and so on, about each API or some user defined actions.
4. Ability to modify existing capture (or with GUI tool or with export/import option)
    a. C++ can be generated from a capture file and the resulting C++ can be modified before building the application.
5. please use C not C++ for source code capture conversion
6. deferred capture
7. GUI for capture investigation (especially if even limited editing capabilities would be supported) would be wonderful and help in multi-frame issues investigation. And even limited ability to edit the capture would be great to quickly test theories why something doesn't work, since in RenderDoc it's possible to edit only shaders.
8. Better ways to diagnose missing extensions on replay.
    a. PR 1221, Print differences in supported extensions and features, added more reporting of missing extensions and features on replay.
9. Better integration/documentation with device-sim layers to control feature support during captures
    a. The Profiles layer allows subsetting features and extensions during capture.
10. Dump captured resources like textures
    a. The “Convert” tool outputs JSON describing all Vulkan commands and metadata and also saves in files all large BLOBs passed from the application to the Vulkan implementation; FillMemory metadata command represents e.g. CPU writes to mapped memory.
11. multithreaded replay
12. Performance is poor for some games that cause a lot of page-guard exceptions inserted by gfx-reconstruct while VisualStudio debugger is attached. I'm not aware of any workaround, it makes debugging capture issues harder.