Vulkan Ecosystem & SDK Survey

October 2021 LunarG Status Update

What's been accomplished since the November 2020 Survey

Executive Summary

Last November 2020, LunarG conducted an ecosystem survey and published the survey results in a public report. This status update document summarizes the ecosystem enhancements completed over the last year in response to feedback received in the 2020 survey.

Look for a new end-of-year 2021 Vulkan Ecosystem & SDK Survey from LunarG in November. We again will be capturing the requirements of Vulkan developers to help shape the priorities of the LunarG software engineering investments. Survey feedback will also help to inform priorities of the Khronos Vulkan community.

Thanks to all the Vulkan developers who have contributed their views to our annual Vulkan SDK and Ecosystem surveys. Should you have feedback related to this report, please send it to **info@lunarg.com**.

Open-Ended Feedback

In the November 2020 survey there were multiple places to provide open-ended feedback:

1. What tools would you recommend for the macOS SDK?
2. How could the validation layers be improved?
3. What features, tools, and/or improvements would you like to see added to the LunarG Vulkan SDK (Windows, Linux, and/or macOS)?
4. If your rating of the overall quality of the Vulkan ecosystem is fair or poor, what are your pain points?
5. What inhibits you from effectively and efficiently developing Vulkan applications?

All of the comments were grouped into the following categories and reviewed.

1. Tutorials/Samples/Developer Documentation (feedback provided to the Vulkan Samples team and included in this document)
2. Vulkan-HPP
3. macOS
4. Developer tools, ecosystem needs/pain points
5. Validation Layers
6. Shader Tool chain
7. SDK
8. Loader

9. API and API Complexity (feedback provided to the Vulkan WG)
10. Specification (feedback provided to the Vulkan WG)

In this status update, we've reported the progress made over the last year.

Tutorials, Samples, and Developer Documentation

This topic received much feedback from developers, who said they want good tutorials, good samples, and good documentation regarding best practices and common design patterns. All of the comments have been shared with the group of engineers contributing to the KhronosGroup/Vulkan-Samples repository and working on ecosystem documentation.

End-of-year update:

1. There are new samples in development that will be released to the repo over the coming 6 - 8 months. These new samples are based on the proposed and voted for samples on the public Trello project. In the meantime, please take a look at KhronosGroup/Vulkan-Samples to see if there are useful samples there and feel free to submit issues or contribute. You also can still contribute suggestions and vote for samples in the Trello project.
2. Khronos is also evaluating currently available internal and external tutorial resources for updates and improvements.
3. Discovery of high quality tutorials, samples, blogs, and other resources is an issue that Khronos is aware of and has been actively taking steps to improve. Khronos launched the updated www.vulkan.org site this year, which makes finding various resources much easier. Khronos is continuing to work on various improvements in the area that should improve the overall developer experience.

Vulkan-HPP (Feedback was shared with the Vulkan-HPP developers at NVIDIA. Any updates are from the NVIDIA developers maintaining/enhancing Vulkan-HPP)

1. Vulkan-HPP provided a better coding experience than the standard Vulkan C API. Better promotion of Vulkan-HPP and showcasing its proper use in official tutorials would be helpful.
    a. **End-of-year update:** Vulkan-HPP samples are being added to the KhronosGroup/Vulkan-Samples repository. The first sample has been implemented (hpp_hello_triangle).
2. Concern areas for Vulkan-HPP
    a. Newer versions of Vulkan-HPP in the Vulkan SDK frequently caused a breakage.
        i. **End-of-year update:** For the last year, there haven't been any Vulkan-HPP breakages. The Vulkan-HPP team would like to believe this is due to some regression tests that were added to help avoid breakages.
    b. The size of Vulkan-HPP
        i. **End-of-year update:** The generator now generates vulkan.hpp plus a couple of sub-headers. That is, technically, the size of vulkan.hpp is substantially reduced, but you still need all the sub-headers as well, so the compiler-work

(which probably is meant with the size issue) is not reduced. Some elements were deprecated last fall, and when they are removed this fall, that will reduce the size slightly. However beyond this, there are not yet identified size optimizations that can be implemented.

macOS (Feedback was shared with Bill Hollings (MoltenVK author). Progress updates are from Bill Hollings)

1. There were multiple folks with concerns about the need to link directly against MoltenVK to get access to the MoltenVK API and extensions. Also, if you wanted to use the Vulkan Loader and Validation Layers you would lose access to the MoltenVK API and extensions.
    a. **End-of-year update:**
        i. MoltenVK's configuration functions, vkGet/SetMoltenVKConfigurationMVK() now have global scope and ignore the VkInstance that is provided. This makes it safe for them to be called with a VkInstance that was retrieved from another Vulkan Layer.
        ii. Other dedicated MoltenVK functions, such as vkGet/SetTexture(), remain problematic in this sense, in that they cannot be called with objects retrieved from other Vulkan Layers. There are plans in the future to move this functionality to new extensions within the external memory family of Vulkan extensions, specifically to handle Metal external objects.
        iii. Hence some incompatibilities still exist when using the validation layers with MoltenVK
2. The documentation between the macOS SDK and the KhronosGroup/MoltenVK README documentation in regards to the runtime is inconsistent and needs clarification.
    a. **End-of-year update:** This has been resolved as of this commit.
3. Installation and usage of the macOS SDK can be confusing and needs some clarification.
    a. **End-of-year update:** LunarG published a white paper (The State of Vulkan on Apple Devices) to clarify the development workflow for Vulkan in a macOS environment.
4. People want more Vulkan extension support in MoltenVK (such as ray tracing)
    a. **End-of-year update:** MoltenVK has advanced by adding numerous extensions on top of Vulkan 1.1 support, and is engaged in a significant drive towards conformance with all CTS tests suitable for a Vulkan 1.0 implementation under the Vulkan Portability subset (VK_KHR_Portability_subset). The list of extension implemented is:
        i. VK_KHR_create_renderpass2
        ii. VK_KHR_depth_stencil_resolve
        iii. VK_KHR_imageless_framebuffer
        iv. VK_KHR_portability_subset
        v. VK_KHR_sampler_mirror_clamp_to_edge (iOS)
        vi. VK_KHR_shader_subgroup_extended_types
        vii. VK_KHR_timeline_semaphore
        viii. VK_EXT_descriptor_indexing

ix. VK_EXT_post_depth_coverage (macOS)
x. VK_EXT_private_data
xi. VK_EXT_subgroup_size_control
xii. VK_EXT_texture_compression_astc_hdr
xiii. VK_IMG_format_pvrtc (macOS)
xiv. VK_AMD_shader_image_load_store (macOS)

5. There was a request for RenderDoc inclusion in the macOS SDK. Renderdoc is not supported yet in a macOS environment.
    a. **End-of-year update:** Due to resource constraints and demand, Renderdoc support is still not planned for the macOS environment.

Developer tools, ecosystem needs/pain points

1. A desire for additional bindings to the Vulkan API such as C# and Rust
    a. **End-of-year update:** The Vulkan WG is not creating any additional bindings beyond C currently. However you can find a Rust binding at this github
2. More helper and utility libraries are needed. Such as an EGL-like library for setting up a reasonable swapchain
    a. **End-of-year update:** Vk-bootstrap does swapchain creation. This project is not actively funded with daily focus but has some reasonable capabilities.
3. A continued need for a profiler that can be used across platforms and GPUs. Would like more tools to help with performance tuning.
    a. **End-of-year update:** To date, LunarG is unaware of an open source cross platform profiler. Of course the IHVs have their specific profilers which would be more powerful than an open source profiler since they have direct proprietary knowledge of their GPUs. There is a proprietary cross platform profiler that released in the last year called vktracer. More information about this profiler can be found at vktracer.com.
4. Some way (e.g. layer) to enable mocking vulkan calls (for testing boring things like allocators in contrived settings (different memory types, for example)); non-coherent memory type simulation layer, this would help when targeting some android systems.
    a. **End-of-year update:** No plans or progress. Maybe a future initiative...
5. An automatic check or a correction for memory alignment in buffer objects.
    a. The validation layers already do checks for correct memory alignment. If you find a case where an incorrect memory alignment is not detected, please submit an issue to the validation layers repository.

Validation Layers

There were many, many comments and LunarG read every one to make sure it was already logged in our github issue tracker or on our TO DO list. One of my favorite comments was: “Honestly, keep on trucking - the validation layer is awesome." We take pride and work hard to continuously improve

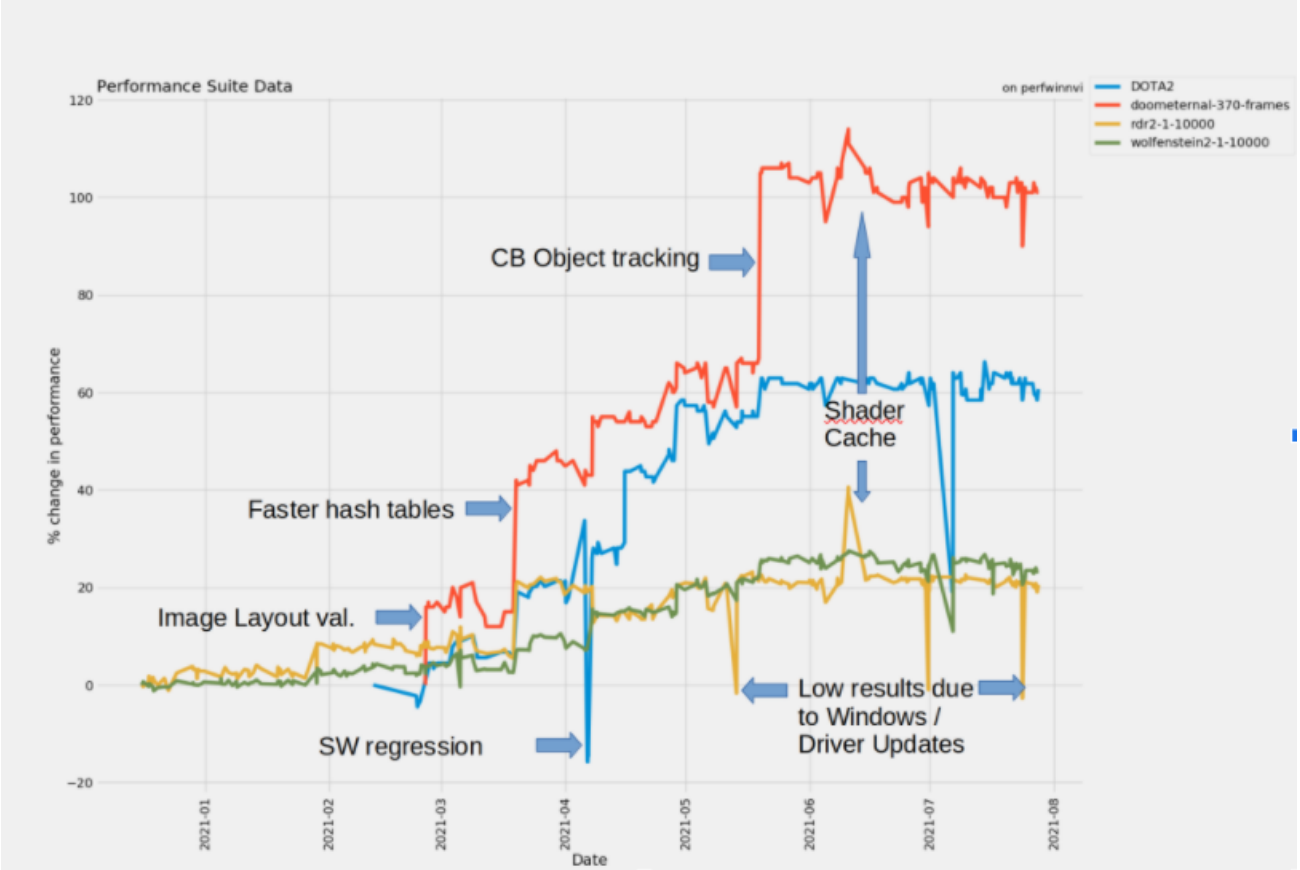
this important component of the Vulkan ecosystem. And we do acknowledge there is still room to improve.

Here is a list of some comments with specific responses that may be useful for developers:

1. Coverage
    a. More validation for descriptor indexing was requested.
        i. **End-of-year update:** This is logged in the enhancement list for GPU Assisted Validation (but has not been done yet) (https://github.com/KhronosGroup/Vulkan-ValidationLayers/projects/3)
    b. Continue filling out validation layer VUID checks.
        i. **End-of-year update:** Filling out validation layer VUID checks is continued at LunarG throughout the year. The github issues that are submitted as missing VUID checks are categorized as “incomplete” and given top priority since they are missing VUID checks found by actual application developers. Once all of them are complete, the work can begin on filling out additional VUID checks not reported by users.
2. Error messages
    a. Create a stack trace of the error’s origin (API call) when validation is done at queue submit time.
        i. **End-of-year update:** This is currently tracked as an enhancement to the message reporting in the validation layers error message improvements project (but has not yet been implemented). (https://github.com/KhronosGroup/Vulkan-ValidationLayers/projects/2)
    b. There was a request to have debugPrintf messages have their own type flags, instead of being bundled with the “info” flag, so they can be enabled without having to enable the other “info” messages.
        i. **End-of-year update:** This has been logged as a potential enhancement request in the project for GPU Assisted Validation. It hasn’t been implemented yet. (https://github.com/KhronosGroup/Vulkan-ValidationLayers/projects/3)
    c. Request: extensions for common debuggers such as Visual Studio which supports validation layer errors being displayed through the debug output window
        i. **End-of-year update:** This is currently supported through the VK_DBG_LAYER_ACTION_DEBUG_OUTPUT layer setting. This setting is defined in the vk_layer_settings.txt file found in the Vulkan SDK in the Config directory. You can also enable this feature using vkconfig (select the “Debug Output” under Debug Action for any of the Validation Vulkan Layer Configurations).
    d. Linking to the chunked specification would be nice.
        i. **End-of-year update:** Still no plans to add this to the SDK. The error messages from the validation layers point to the specification built for the SDK and need to be able to point to an anchor inside the specification. The

chunked specification provides challenges to get the URL to the correct "chunk."

3. Improve the performance of the validation layers.
    a. **End of Year Update**:
        i. For a year now LunarG has had a dedicated resource focused purely on improving the performance of the validation layers. This is a challenging problem that requires extensive profiling, triaging, code refactoring, fixing and then repeating that loop as each performance bottleneck is removed.
        ii. LunarG also created a performance regression test suite that is run nightly to catch performance regressions as validation layer development continues. This is to avoid performance degradation over time where possible.
        iii. Identified key performance bottlenecks are
            1. Many active threads in Vulkan applications; but a single lock per Validation Object in the Validation Layer
            2. Image Layout Transition validation
            3. Large "bindless" DescriptorSets
        iv. Some performance optimizations made so far (see the chart on the next page):
            1. Image Layout Transition validation
            2. "Command Buffer State" (CB) validation; track which are in use by each command buffer
            3. Faster hash table implementation (for Vulkan Objects)
            4. Shader validation results caching (controlled by a Validation Layer setting)



v. Future areas of work

1. Finer grained locking in the Validation Layer (in progress)
    a. This is a major effort that first requires significant code refactoring to get data structures organized to enable less frequent locking and locking for smaller time intervals.
    b. Once the refactoring is completed, we can then begin working on fine tuning the locking
    c. This is work in progress and very likely that by the time you are reading this it may be merged.
2. Further Image Layout Transition optimizations
3. Then look for the next bottleneck...

4. Synchronization Validation. Synchronization validation layers have been incredibly useful for application developers. People want more...
    a. **End of Year Update**:
        i. The first version of validation for synchronization was released in August of 2020 as alpha release quality. This validation was for what we referred to as “Phase I” functionality which covered:
            1. Identify resource access conflicts due to missing or incorrect synchronization operations between actions (draw, copy, dispatch, blit, etc.) reading or writing the same regions of memory.
            2. Functionality includes commands within a single buffer.
        ii. Since that first release, we conducted extensive quality reviews of the code by ensuring that 100% of Vulkan Conformance tests passed and also addressed any issues found by users in the community.
        iii. We also updated the synchronization validation to include events within a command buffer. This validation includes both resource hazard detection and detection of race conditions between the various event calls.
        iv. With the 100% pass of the Vulkan conformance test suite and the addition of the validation for events, we removed the “alpha” designation from the synchronization validation implementation in January of 2021.
        v. March 1, 2021, the VK_KHR_Synchronization2 extension was released. This extension provided extensive improvements to Vulkan queue submission, events, and pipeline barriers resulting in significant API usability enhancements for developers. With its release, LunarG implemented a new layer called VK_LAYER_KHRONOS_synchronization2 to emulate the extension functionality if your device driver doesn’t support the extension. In addition, validation support was added to the validation layers to validate usage of the VK_KHR_Synchronization2 extension.
5. Warnings/Best Practices
    a. There were multiple requests for “Best Practice” warnings such as:
        i. Warn on deprecated function usage
        ii. Some insight into common "mis-patterns" or unusual behavior
        iii. Some insights into potential performance warnings
    b. The BEST PRACTICES validation layer object covers many of these items to some degree. More information on the BEST PRACTICES can be found here: https://vulkan.lunarg.com/doc/sdk/latest/windows/best_practices.html
    c. **End-of-year update:** AMD recently added Best Practices to the Validation Layer. They have been included in the 1.2.189.2 SDK and are exposed by the Vulkan Configurator (vkconfig) for easy enablement.

Shader Tool Chain

1. Make sure online compilation in addition to offline compilation is supported well.
    a. **End-of-year update:**
        i. Prior to this ecosystem survey conducted in November of 2020, many of the shader tool chain API libraries had already been added to the SDK in April of 2020. This included KhronosGroup/glslang, Google/shaderc, KhronosGroup/SPIRV-Cross, and KhronosGroup/SPIRV-Tools.
        ii. In June of 2020, the DXC offline compiler was added.
        iii. In May of 2021, the API library for DXC was added to the SDK.

SDK

This list represents the most commonly requested features or feedback for the Vulkan SDK.

1. Add more features to the macOS SDK.
    a. GPU Assisted Validation (and Debug Printf)
        i. **End-of-year update:**
            1. GPU-AV and Debug Printf require Vulkan 1.1 + fragmentStoresAndAtomics, vertexPipelineStoresAndAtomics, shaderint64. MoltenVK is at Vulkan 1.1 and supports the Atomics.
            2. MoltenVK is currently focused on getting the Vulkan 1.0 CTS tests passing at 100% (very close). MoltenVK has also added Vulkan 1.1 support. Once CTS is passed, the validation layer test suite that tests debugPrintf and GPU-AV will be used to identify any additional gaps in MoltenVK functionality to add this support. Per preliminary analysis, it is believed that GPU-AV and Debug Printf should just work.
    b. iOS as a target of the SDK
        i. **End-of-year update:** This still remains on the TO DO list.
    c. Port to macOS
        i. Devsim
            1. **End-of-year update:** With the macOS SDK delivered January 2021 (version 1.2.162.1), devsim has been ported to macOS and included with the macOS SDK.
        ii. GFXReconstruct
            1. **End-of-year update:** This is still a low priority.

iii. Monitor
1. **End-of-year update:** This is still a low priority.
iv. Screenshot
1. **End-of-year update:** This is still a low priority.

2. SDK is a big “blob." Breaking it into smaller packages and inclusion of all the build types (debug, static, dynamic, PDB, …) would be more useful.
   a. **End-of-year update:**
      i. As of the 1.2.189.1 SDK delivered in September 2020, the Windows SDK has been repackaged into a modular format with a core package and optional packages. The initial release had the following optional packages:
         1. Shader debug libraries
         2. 32-bit binaries
         3. Third party libraries
      ii. The repackaged SDK will now enable more optional packages in the future.
3. Create separate packages for debug builds and PDB files for Windows SDK components
   a. **End-of-year update:**
      i. The repackaged SDK will now enable more optional packages in the future.
4. Some simple getting started libraries such as vk-bootstrap would be useful.
   a. **End-of-year update:** No resources have yet been focused on this.
5. Add packages for more Linux distributions such as BSD or Arch Linux.
   a. **End-of-year update:** Currently LunarG created packages for Ubuntu 18.04 and 20.04 and the packages are updated with each SDK release. Due to resources, it isn’t possible for LunarG to pick up package creation and maintenance for additional Linux distributions.
6. SDK Additions
   a. VOLK
      i. **End-of-year update:** On the LunarG TO DO list. Other SDK priorities (like repackaging of the windows SDK) took priority with the limited resources.
   b. Vulkan Hardware Capability Viewer
      i. **End-of-year update:** The Hardware Capability Viewer was ported to iOS in February of 2021 and is released on the Apple Store. The macOS, Windows, and Linux versions are not yet included with the SDK. It is on the TO DO list for the future.
   c. Vulkan Memory Allocator
      i. **End-of-year update:** On the LunarG TO DO list. Other SDK priorities (like repackaging of the windows SDK) took priority with the limited resources.
   d. Include a script to build all the repository versions included in the SDK.
      i. **End-of-year update:**
         1. The release notes for each SDK has a list of branches or commits for all the included repositories. In addition, with each SDK, there is a CONFIG.json that is a programmatic way to determine all the branches and commits. You can read about the API to download this

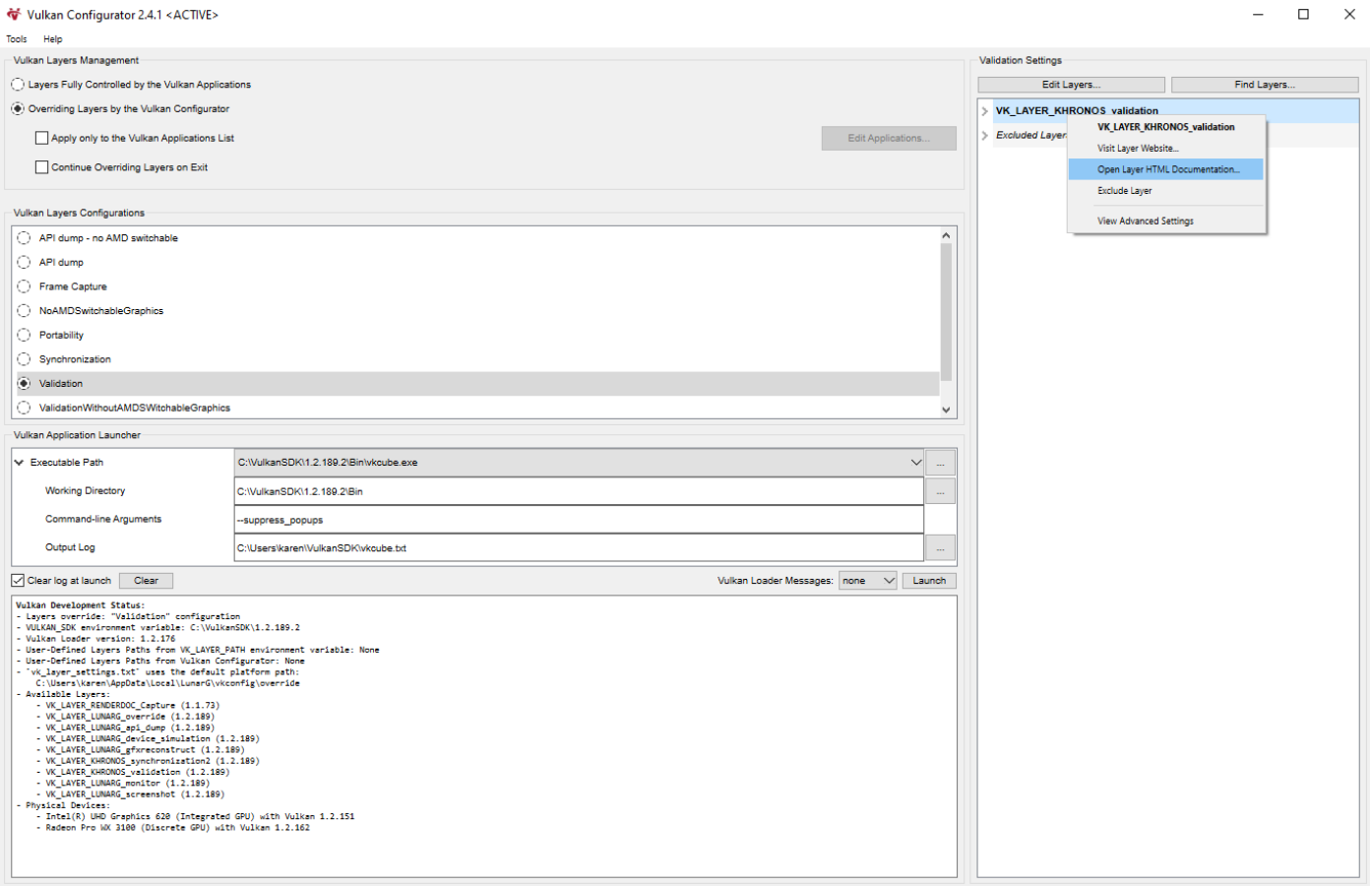
information on the SDK download site (see the SDK version query and download API documentation).

2. An individual could then go to each repository and build based on these branches/tags, and commits. We have not published a script that will clone all of the repositories and do the builds for the user. For individuals that are motivated to build all the SDK components themselves, they could create such a script relatively easily. Hence this has not been a high priority.

e. Include vk-bootstrap in the SDK as a simple getting started example.
   i. **End-of-year update:** The SDK moved away from including code samples. To enable a focus for limited resources, it was decided that SDK was not to teach the Vulkan API, but rather to provide tools to application developers to develop to the API. There are multiple tutorial resources to be found that the users can study. And the KhronosGroup/Vulkan-Samples repository is working on creating high quality samples.

f. For synchronization validation, create a tool to visualize the synchronization, both as defined by the specification and what is being actualized by the application.
   i. **End of Year Update**: No progress. It is on a TO DO list, but we are first focused on finishing the validation of synchronization in the validation layers.

7. For the Vulkan Configurator, add these additional enhancements:
   a. Ability for vkconfig to detect and configure/order non-SDK layers
      i. **End-of-year update:** As of July 2021, this is now delivered in the SDK. In August of 2021, we completed a layer settings helper library (vk_layer_settings.h), and published a white paper that comprehensively shows layer developers how to develop their layers to benefit from the ecosystem tools used by the LunarG layers and the Khronos Validation Layer.
   b. Link to Vulkan specification for reported VUID violations
      i. **End-of-year update:** The validation layers include a link into the Vulkan Specification for the violated VUID. These URLs will display in the Vulkan Configurator log window.
   c. Include in-application documentation for layers
      i. **End-of-year update:** As of July 2021, the Vulkan Configurator will display “in application” documentation for layer settings. Vkconfig does this real time by reading the layer settings manifest file. The following screen capture shows how once you right click on the layer, you can “Open Layer HTML Documentation” to view all of the layer settings available.



d. Add diagnostic capabilities to inspect the health and status of my Vulkan installation
    i. **End-of-year update:** Not yet satisfying: Currently, Vulkan Configurator can run “Vulkan Installation Analyser” and provide a “Vulkan Development Status” at launch and then selecting layers but we could significantly improve this aspect of the tool.
e. Search capability for the output log area
    i. **End of Year Update**: Not yet implemented
f. Add a command line interface to Vulkan Configurator features
    i. **End-of-year update:** Using the command vkconfig --help, the tool lists all the commands supported. The support allows the C.I. systems to rely on Vulkan Configurator to run tests with layers.
g. Make Vulkan Configurator a library for 3rd party Vulkan developer tools
    i. **End-of-year update:** Not done

Loader

1. Applications need a way to query all the layers (can do this today), but to also know which layers are implicit and which layers are explicit (can't differentiate between them today). Applications are having issues with bad implicit layers causing crashes in their applications and need a way to query the layers for logging purposes, and possibly disable them.
    a. **End of Year Update:**
        i. From the November 2020 ecosystem survey initial report: LunarG intends to take this to the Working Group to get an API defined to enable this querying (and potential disabling of layers by an application).
            1. Define a vkEnumarateInstanceLayerProperties2 (or similar) that will list the layers and also indicate which layers are implicit.
            2. Define a programmatic way for users to disable layers at create Instance time, most likely as a pNext struct in vkCreateInstance.
            3. **Change of plan**: Although the original intention was to define an extension along the lines of what was described above, it is now realized that the approach above would be deficient in that you would not necessarily know which layers are enabled until after create instance time. It is believed that a better interface to the loader can be defined but is going to take some more time.
        ii. From the November 2020 ecosystem survey initial report: Another feature that will be released soon is "safe mode for layer loading" (will be implemented in the loader).
            1. The current plan is to implement a safe mode that only disables layers that were not explicitly enabled, either through vkCreateInstance or environment variables. This means that explicit layers would behave the same as outside safe mode. Implicit layers would still run when they are enabled with an environment variable (either VK_INSTANCE_LAYERS or the layer-specific envar) but would not ever run purely because they are installed on the system. The goal is to eliminate the layers that no one knows are installed on a system, allowing tools like steam and renderdoc to still work in safe mode (because both enabled their tools through envars), but would disable layers that aren't being used.
            2. **Change in plan**:
                a. After more review it was decided not to implement the safe mode loader due to potential bad side effects such as losing loading of desirable implicit layers on a per application basis, and applications running differently on different systems based on safe mode settings set by the user.
                b. Instead, additional layer logging was added (see above) and an initiative is in place at LunarG to clearly articulate Loader

policies in regards to layers and ICDs (implementations) and enforce the policies where possible.

iii. Although the above items originally communicated a year ago as in plan were not delivered due to a change in plan, one enhancement was recently made to the loader:

1. Applications can't directly query the layers, but indirectly can determine all explicit and implicit layers by setting VK_LOADER_DEBUG=layer. Additionally, in this output, if a layer is implicit, it lists the disable environment variable needed to disable the layer. Vulkan-Loader commit which adds this information