
Simplify Vulkan Development with new Ecosystem Enhancements
Karen Ghavam
LunarG, Inc
1
LUNARG

Vulkan Ecosystem Enhancements

Speaker: Karen Ghavam, LunarG Inc.

CEO and Engineering Manager with 30 years of experience working with 3D graphics solutions in engineering management and program management roles.

Slides are available at:
https://www.lunarg.com/news-insights/white-papers/vulkan-ecosystem-enhancements-siggraph-2021/

LUNARG

Agenda

- The Vulkan SDK and the License Registry
- LunarG Vulkan Ecosystem Improvement Initiatives
  - Vulkan Configurator
  - Vulkan Synchronization
  - Vulkan Layer Performance Improvements
  - GPU-Assisted Validation
  - Debug Printf
  - Validation Layer Error Reporting Improvements
  - GFXReconstruct
  - Device Simulation Layer
  - Shader Tool Chain enhancements

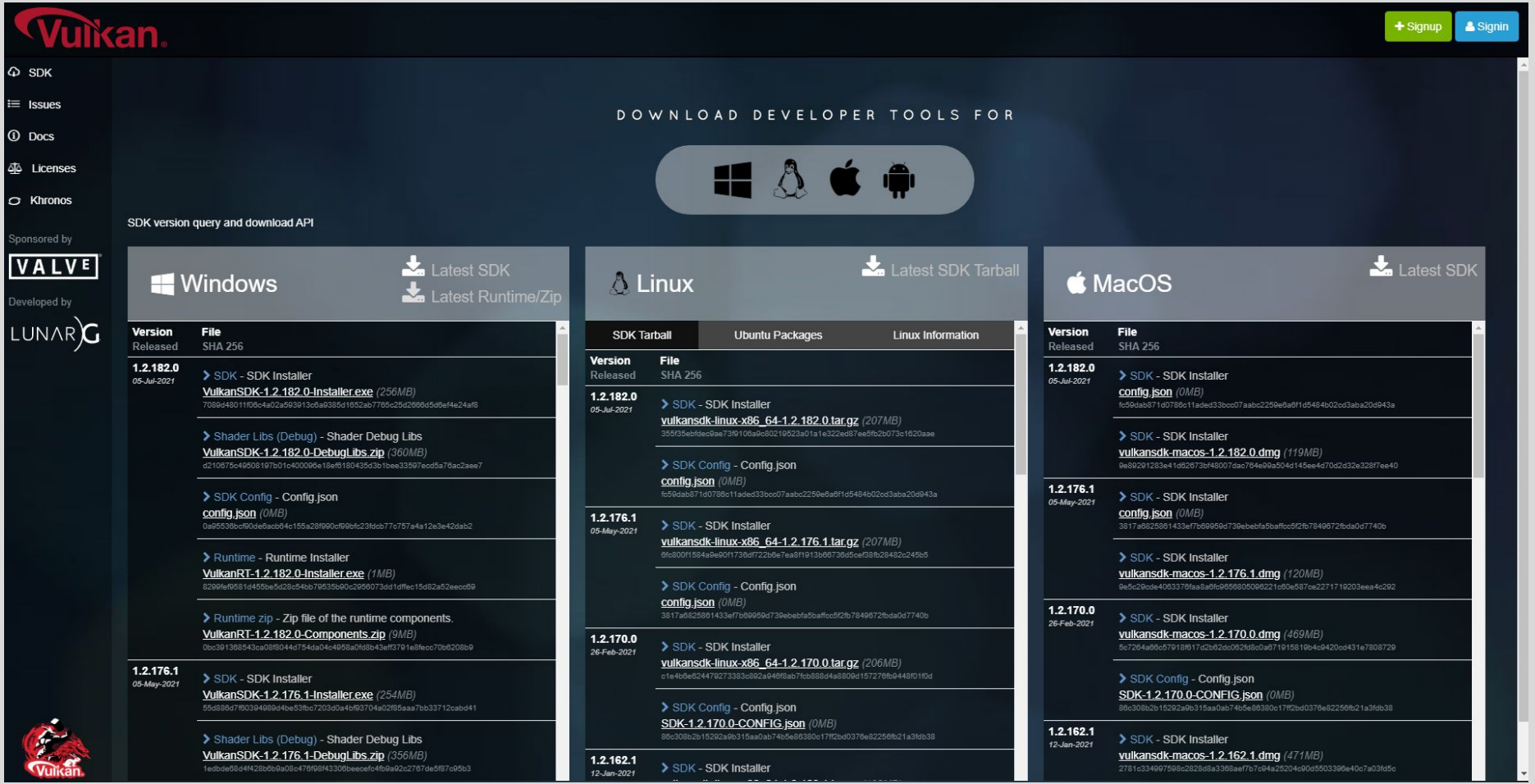
During the presentation, enter any questions you have in the chat window. We will address them at the end.

What is Vulkan?

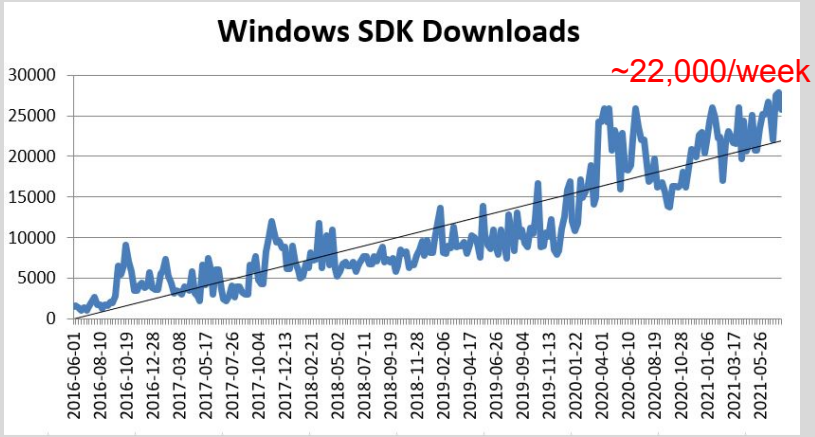
Vulkan is a next generation graphics and compute API that provides high efficiency, cross-platform access to modern GPUs used in PCs, consoles, mobile devices, and embedded platforms.

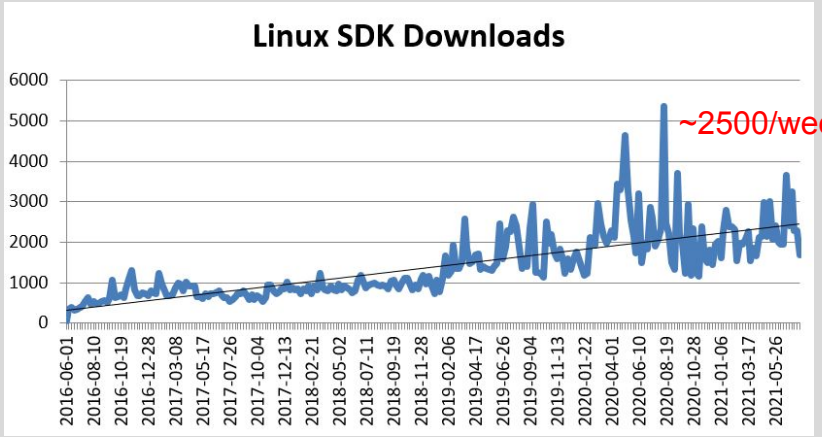
The Vulkan SDK (vulkan.lunarg.com)

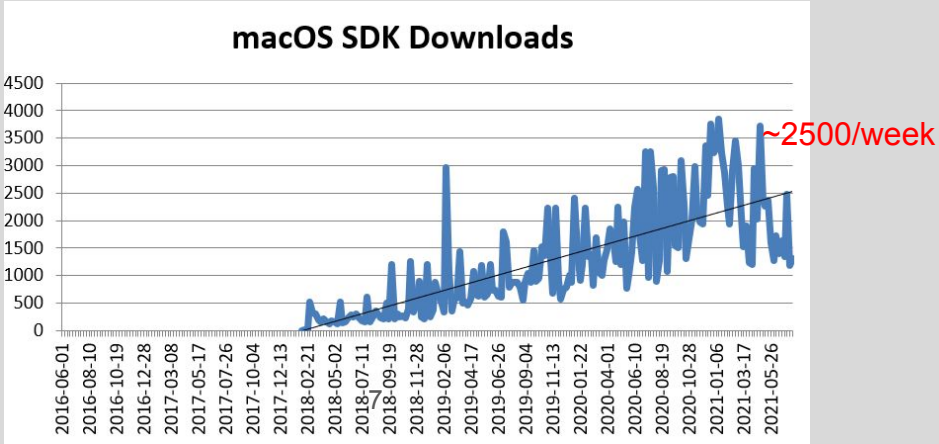
LUNARG


Vulkan
+ Signup
Signin
SDK
Issues
Docs
Licenses
Khronos
Sponsored by
VALVE
Developed by
LUNARG
DOWNLOAD DEVELOPER TOOLS FOR
SDK version query and download API
Windows
Latest SDK
Latest Runtime/Zip
Version
Released
File
SHA 256
1.2.182.0
05-Jul-2021
SDK - SDK Installer
VulkanSDK-1.2.182.0-Installer.exe (256MB)
7089d48011f06c4a02a593913c6a9385d1652ab7765c25d2666d5d6ef4e24af8
Shader Libs (Debug) - Shader Debug Libs
VulkanSDK-1.2.182.0-DebugLibs.zip (360MB)
d210675c49508197b01c400096e18ef6180435d3b1bee33597eod5a76ac2aee7
SDK Config - Config.json
config.json (0MB)
0a95530bcf90de6acb64c155a28f990cf99bfc23fdcb77e757a4a12e3e42dab2
Runtime - Runtime Installer
VulkanRT-1.2.182.0-Installer.exe (1MB)
8299fef9581d455be5d28c54bb79535b90c2956073dd1dffec15d82a52eecc69
Runtime zip - Zip file of the runtime components.
VulkanRT-1.2.182.0-Components.zip (9MB)
0bc391368543ca08f8044d754da04c4958a0fd8b43eff3791e8fecc70b6208b9
1.2.176.1
05-May-2021
SDK - SDK Installer
VulkanSDK-1.2.176.1-Installer.exe (254MB)
55d886d7f60394989d4be53fbc7203d0a4bf93704a02f85aaa7bb33712cabd41
Shader Libs (Debug) - Shader Debug Libs
VulkanSDK-1.2.176.1-DebugLibs.zip (356MB)
1edbde86d4f428b6b9a08c479f09f43306beecefc4fb9a92c2787de5f87c95b3
Linux
Latest SDK Tarball
SDK Tarball
Ubuntu Packages
Linux Information
Version
Released
File
SHA 256
1.2.182.0
05-Jul-2021
SDK - SDK Installer
vulkansdk-linux-x86_64-1.2.182.0.tar.gz (207MB)
355f35ebfdec9ae73f9106a9e80219523a01a1e322ed87ee5fb2b073c1620aae
SDK Config - Config.json
config.json (0MB)
fc59dab871d0786c11aded33bcc07aabc2259e6a6f1d5484b02cd3aba20d943a
1.2.176.1
05-May-2021
SDK - SDK Installer
vulkansdk-linux-x86_64-1.2.176.1.tar.gz (207MB)
6fc800f1584a9e90f1736df722b6e7ea8f1913b66736d5cef38fb28482c245b5
SDK Config - Config.json
config.json (0MB)
3817a682586143ef7b69959d739ebebfa5baffcc5f2fb7849672fbda0d7740b
1.2.170.0
26-Feb-2021
SDK - SDK Installer
vulkansdk-linux-x86_64-1.2.170.0.tar.gz (206MB)
c1e4b6e6244792723383c892a946f8ab7fcb888d4a8909d157276fb9448f01f0d
SDK Config - Config.json
SDK-1.2.170.0-CONFIG.json (0MB)
86c308b2b15292a9b315aa0ab74b5e86380c17ff2bd0376e82256fb21a3fdb38
1.2.162.1
12-Jan-2021
SDK - SDK Installer
MacOS
Latest SDK
Version
Released
File
SHA 256
1.2.182.0
05-Jul-2021
SDK - SDK Installer
config.json (0MB)
fc59dab871d0786c11aded33bcc07aabc2259e6a6f1d5484b02cd3aba20d943a
SDK - SDK Installer
vulkansdk-macos-1.2.182.0.dmg (119MB)
9e89291283e41d62673bf4807dac764e99a504d145ee4d70d2d32e328f7ee40
1.2.176.1
05-May-2021
SDK - SDK Installer
config.json (0MB)
3817a682586143ef7b69959d739ebebfa5baffcc5f2fb7849672fbda0d7740b
SDK - SDK Installer
vulkansdk-macos-1.2.176.1.dmg (120MB)
9e5c29cde4063376faa8afc9656805096221c60e587ce2271719203eea4c292
1.2.170.0
26-Feb-2021
SDK - SDK Installer
vulkansdk-macos-1.2.170.0.dmg (469MB)
5c7264a66c57918f617d2b62dc062fd8c0a671915819b4c9420cd431e7808729
SDK Config - Config.json
SDK-1.2.170.0-CONFIG.json (0MB)
86c308b2b15292a9b315aa0ab74b5e86380c17ff2bd0376e82256fb21a3fdb38
1.2.162.1
12-Jan-2021
SDK - SDK Installer
vulkansdk-macos-1.2.162.1.dmg (471MB)
2781c334997598c2828d8a3368aef7b7c94a25204c90d5503396e40c7a03fd5c

Vulkan SDK Downloads are Healthy and Continue to Grow

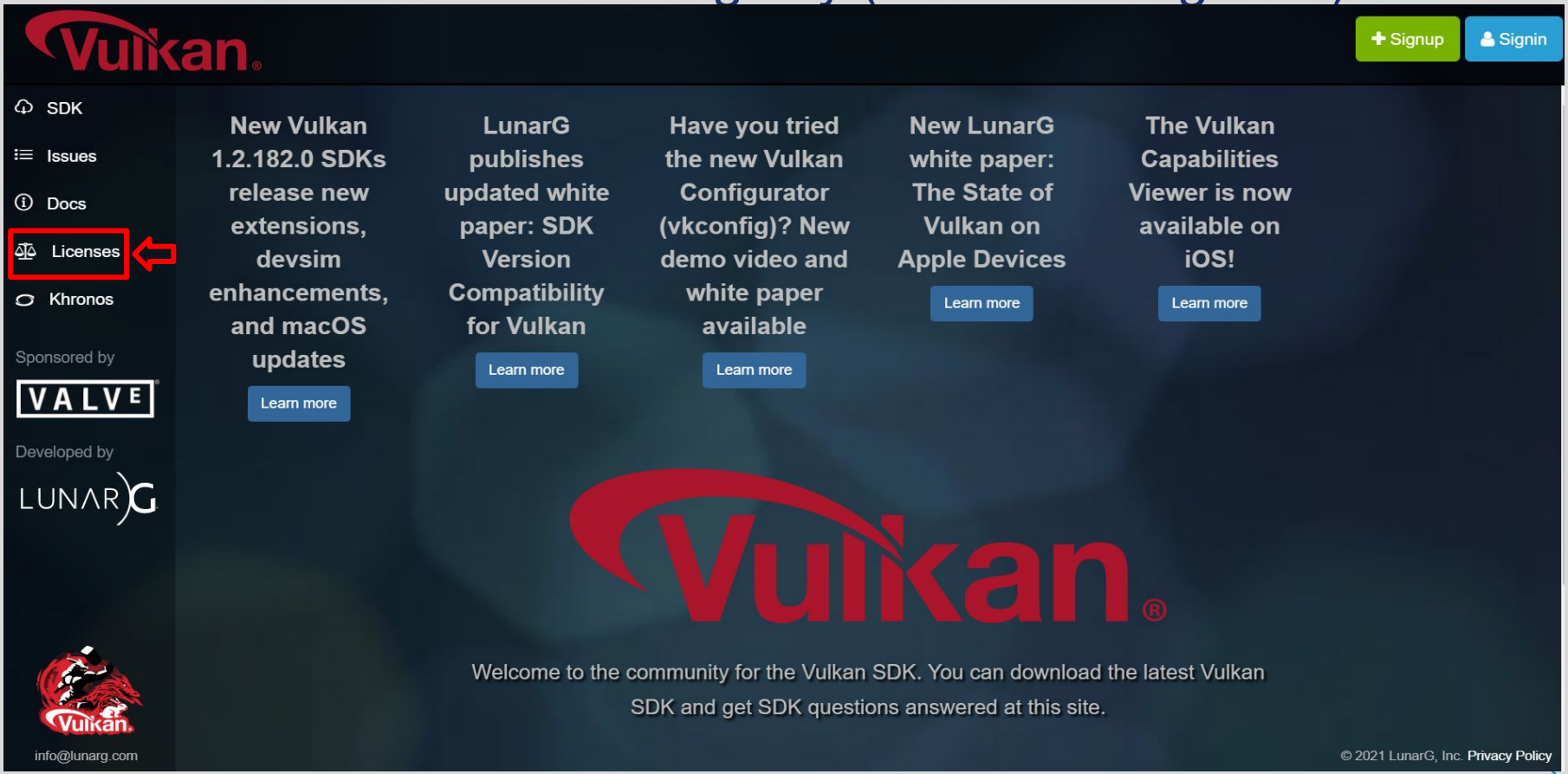






7

The Vulkan SDK License Registry (vulkan.lunarg.com)



LUNARG

- The License Registry allows you to examine the licenses for all of the SDK components:
  - License Summary - lists all licenses used in an SDK
  - License Detail - shows licenses by SDK component

Developer tools in the Vulkan SDK

Vulkan Configurator - GUI application to configure layers used by Vulkan applications at runtime with built-in configurations for the SDK included layers:

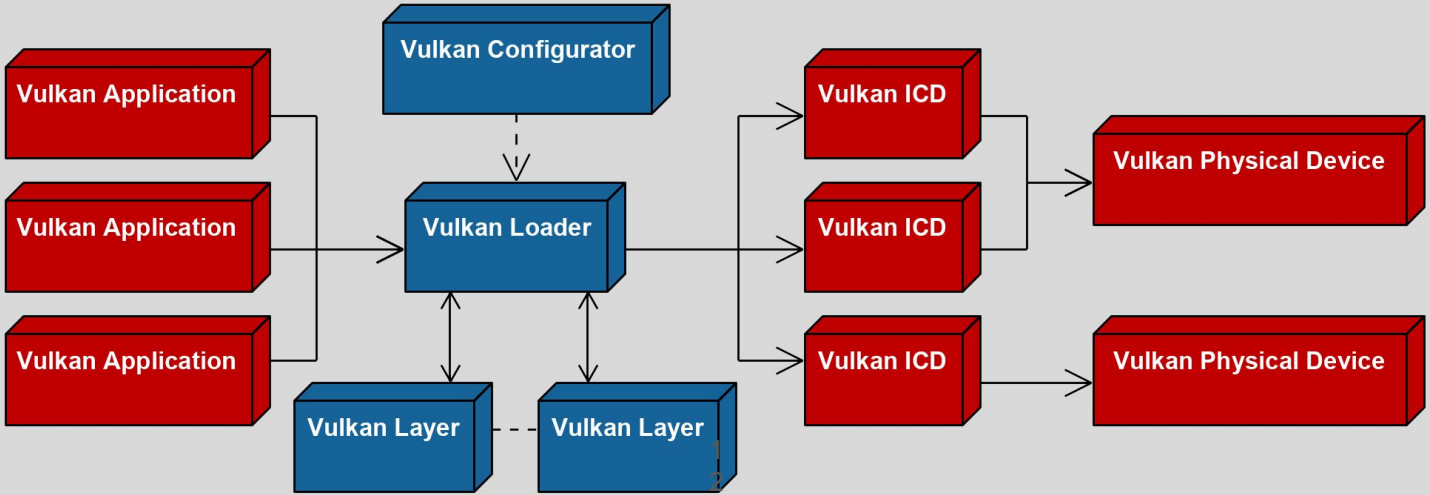
1. **VK_LAYER_KHRONOS_validation** - validate application correct usage of the Vulkan API
   a. **GPU Assisted Validation** - runtime validation executed on the GPU (rather than the CPU)
   b. **Best Practice** - catch correct Vulkan API usage that still could cause application issues
   c. **Synchronization Validation** - identify resource access conflicts due to incorrect synchronization operations between actions
   d. **Debug Printf** - debug shader code using printf inside a shader
2. **VK_LAYER_KHRONOS_synchronization2 -** Emulates the VK_KHR_synchronization2 API
3. **VK_LAYER_LUNARG_api_dump** - ascii output of Vulkan API calls
4. **VK_LAYER_LUNARG_device_simulation** - By simulating query results for various GPUs, enables development of portable Vulkan applications targeting a hardware ecosystem
5. **GFXReconstruct:** Capture (with VK_LAYER_LUNARG_gfxreconstruct) and Replay

Developer tools in the Vulkan SDK

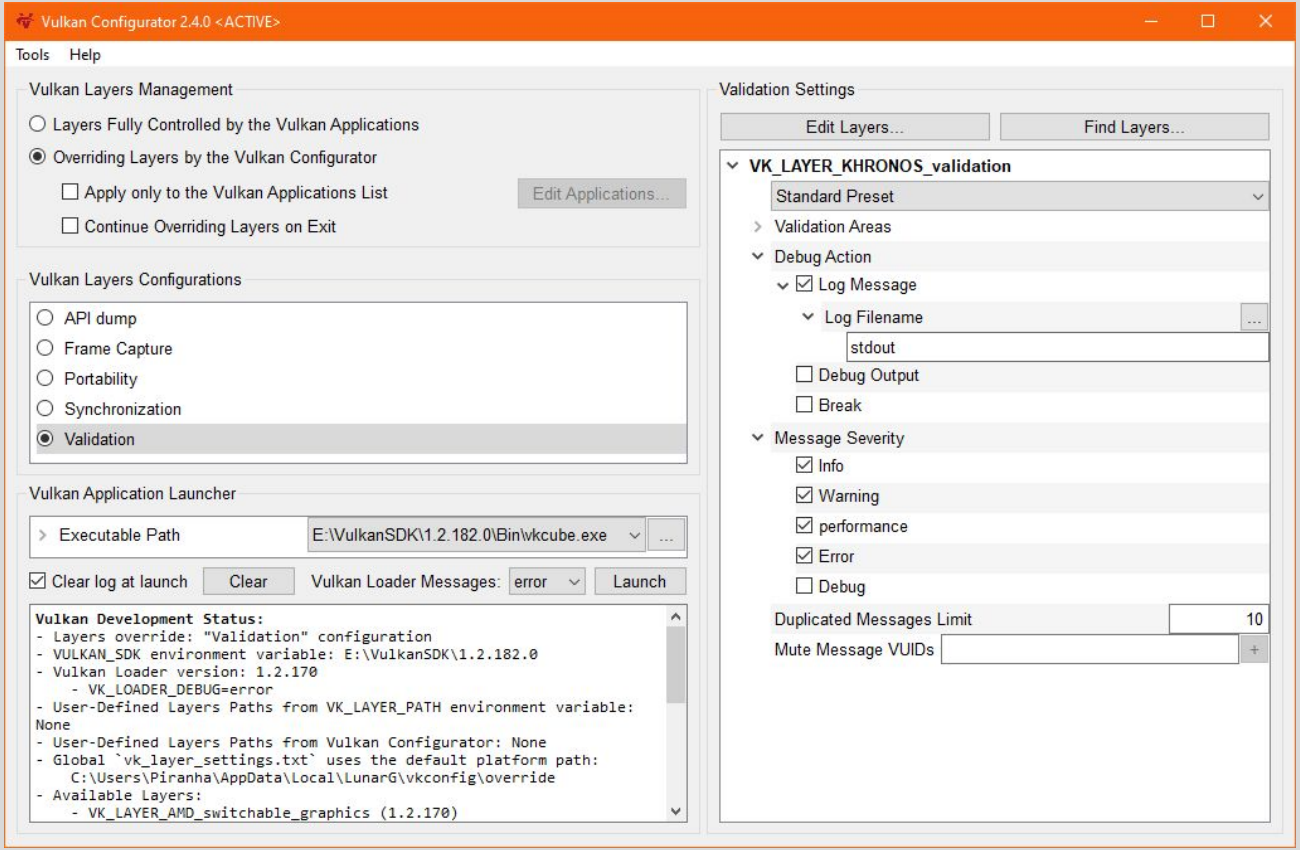
- **Vulkaninfo** - Show GPU device properties and extensions, installed layers, supported image formats, properties...
- **vkvia** (Vulkan Installation Analyzer)
- **Shader Tool chain** - offline executables and API libraries for:
  a. SPIRV-Tools (validator, optimizer, assembler, disassembler)
  b. glslang SPIR-V generator
  c. DXC (DirectX Shader Compiler)
  d. Shaderc SPIRV-Tools wrapper for better integration with build tools
  e. SPIRV-CROSS, a practical tool and library for performing reflection on SPIR-V and disassembling SPIR-V back to high level languages

The Vulkan Configurator (vkconfig): An Introduction

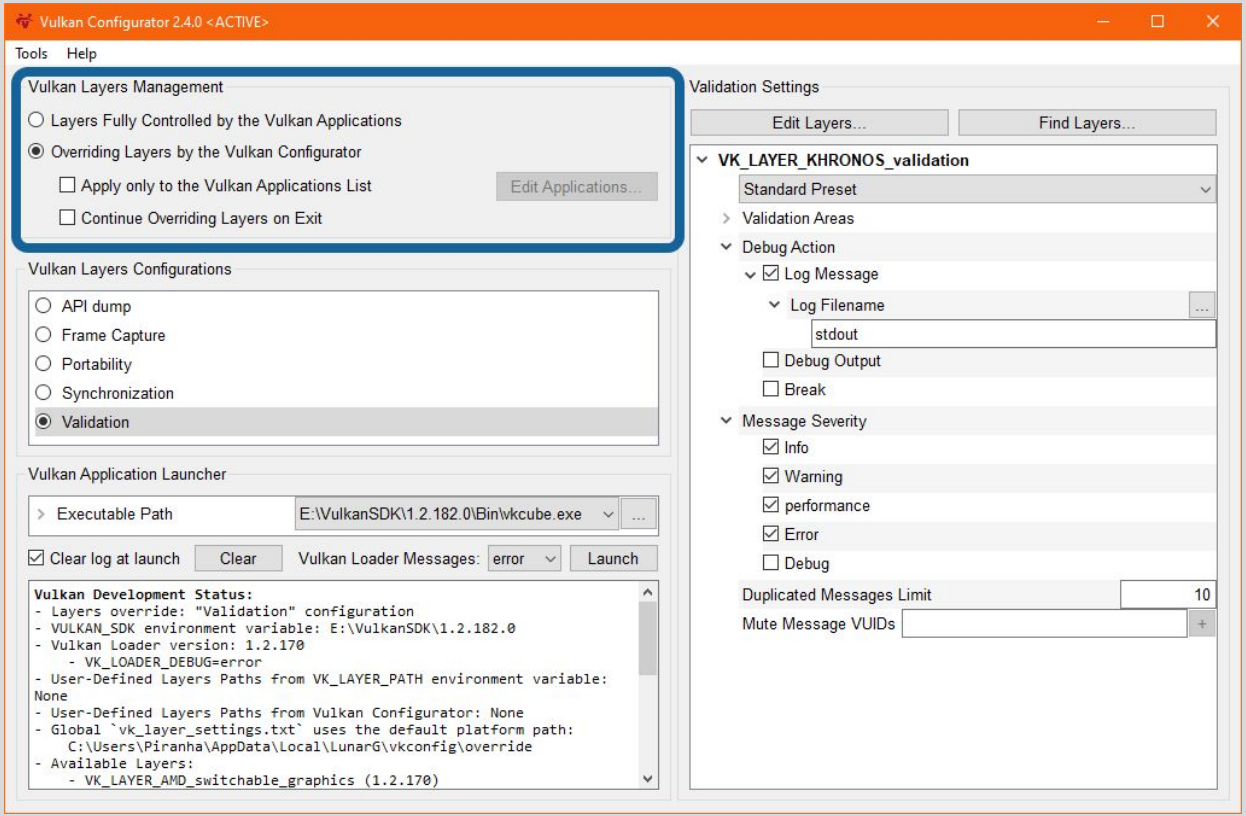
- A GUI to intuitively configure the layers used by Vulkan applications at runtime.
- Design Goals:
  - Make it TRIVIAL and FAST to setup Vulkan Layers for Vulkan application development.
    - Eg: We can validate a Vulkan application with a single click.
    - Can easily change layers configuration without application rebuild.
  - In application documentation available for advanced usages and deeper understanding.
- An alternative to using environment variables and manually writing vk_layer_settings.txt file.



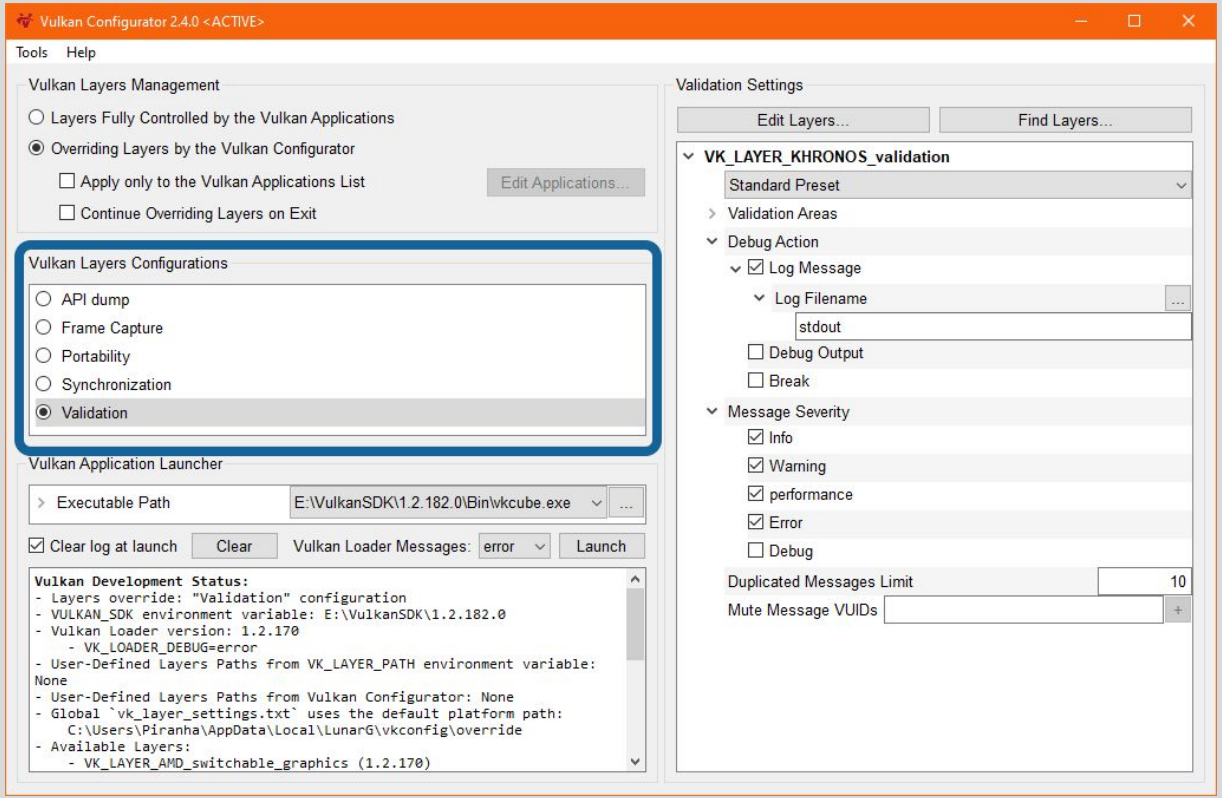
Vulkan Configurator: UI Layout



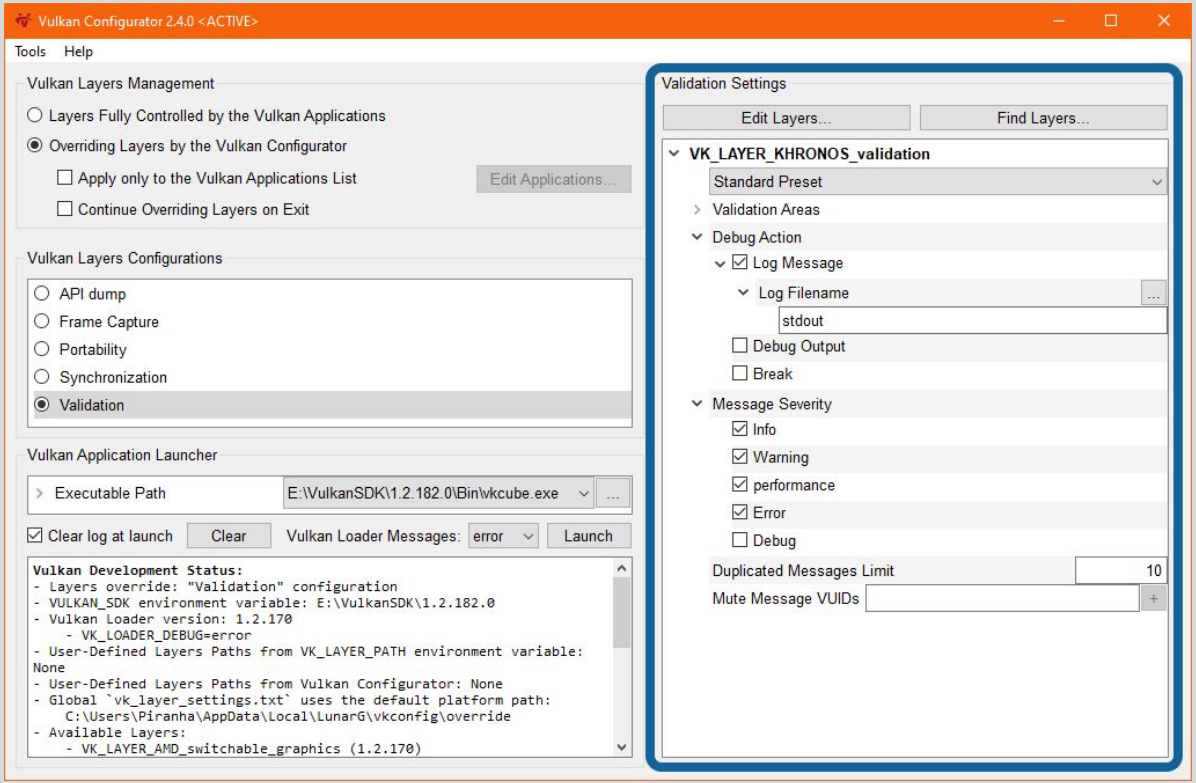
Vulkan Configurator: Layers Management modes



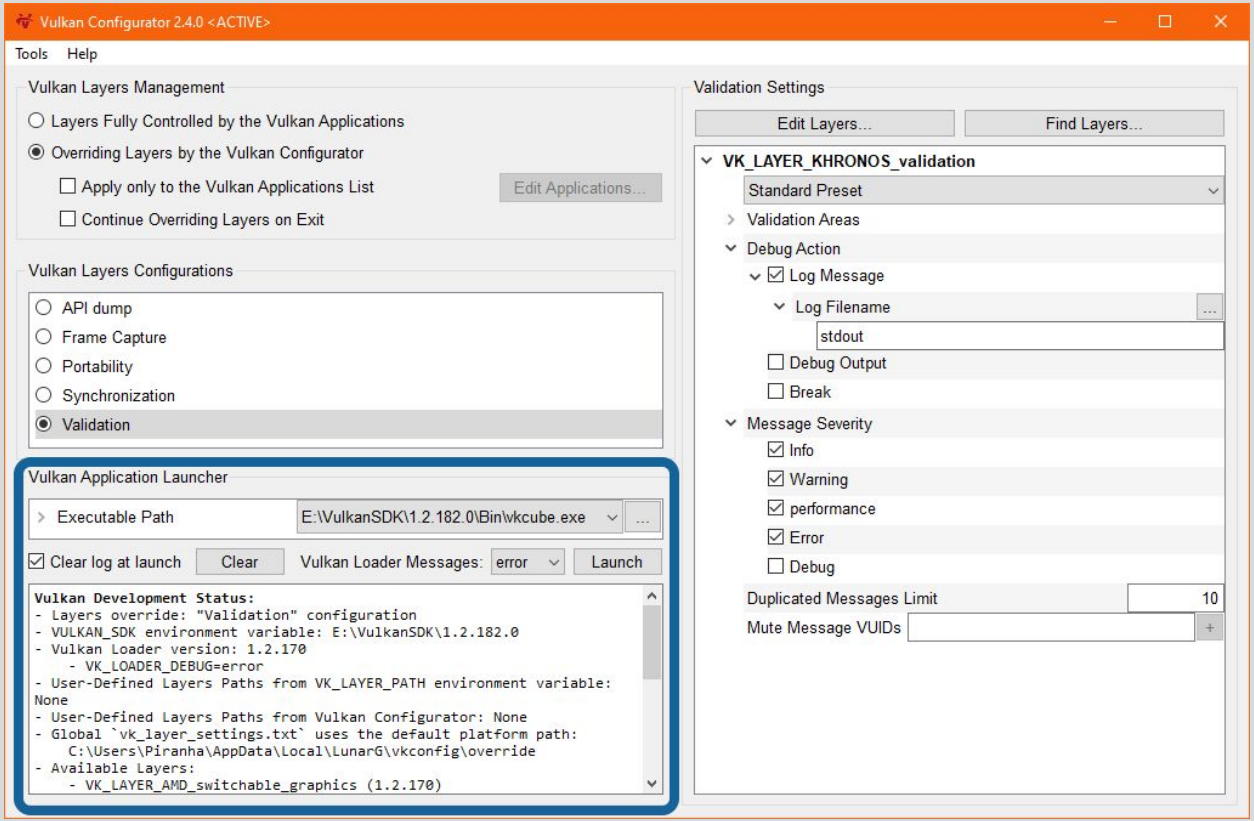
Vulkan Configurator: Layer Configurations



Vulkan Configurator: Layer Settings

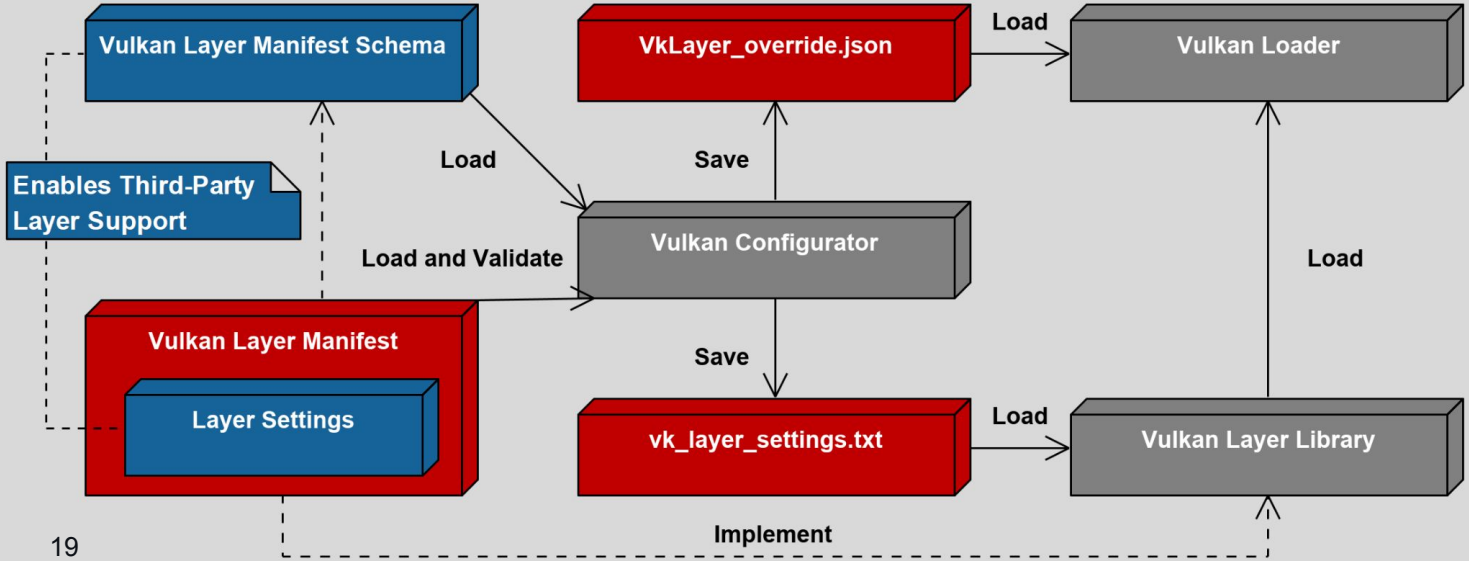


Vulkan Configurator: Application Launcher



Vulkan Configurator: Layer Settings

- The layer setting interface is stored in the layer manifest JSON file.
- A JSON Schema specifies the layer manifest format *for any layer* to expose publicly its settings.
- Settings are read by Vulkan Configurator and used to configure the Vulkan developer system by creating:
  - The `VkLayer_override.json` file that lists the Vulkan Layers loaded by the Vulkan Loader.
  - The `vk_layer_settings.txt` file that lists the settings loaded by each Vulkan Layers.



Vulkan Configurator: Near Future

- Publish a whitepaper explaining how to create a Vulkan Layer in symbiose with the Vulkan Configurator
  - How to write the Layer Manifest to expose the layer settings in Vulkan Configurator.
  - How to implement the layer settings in the layer.
  - Architecture of interactions between the Vulkan Loader and Vulkan Layers.
- Publish a helper C++ library to manage layer settings within layer code.

Vulkan Synchronization

- By design, Vulkan is an explicit API
  - The programmer must tell Vulkan when 2 commands depend on each other
  - This is done by defining barriers
  - Execution Dependencies
    - Most Vulkan commands are started in queue submission order but may execute in any order
    - Even commands using the same pipeline stages!
  - Memory Dependencies
    - GPUs have lots of caches and are accessed by pipeline stages

See the SIGGRAPH BoF, *Ensure Correct Vulkan Synchronization by Using Synchronization Validation*, for a tutorial covering Vulkan Synchronization and validating Vulkan Synchronization usage by your application

- Presentation link: https://www.lunarg.com/news-insights/white-papers/vulkan-synchronization-siggraph-2021/

Vulkan Synchronization

- VK_KHR_synchronization2 (introduced March 2021)
  - Extensive improvements to Vulkan queue submission, events, and pipeline barriers resulting in significant API usability enhancements for developers.
- Emulation layer for devices that don’t support the VK_KHR_synchronization2 API

Validation of Vulkan Synchronization

- Implementation Phases
  - Phase I:
    - 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.
    - Functionality includes commands within a single buffer
  - Phase II:
    - Same as phase I, but adding multiple command buffers, both at queue submit and vkCmdExecute time
- Phase I validation released August of 2020
- Validation for VK_KHR_Synchronization2 released March, 2021
- Phase II validation is under development

Validation of Vulkan Synchronization

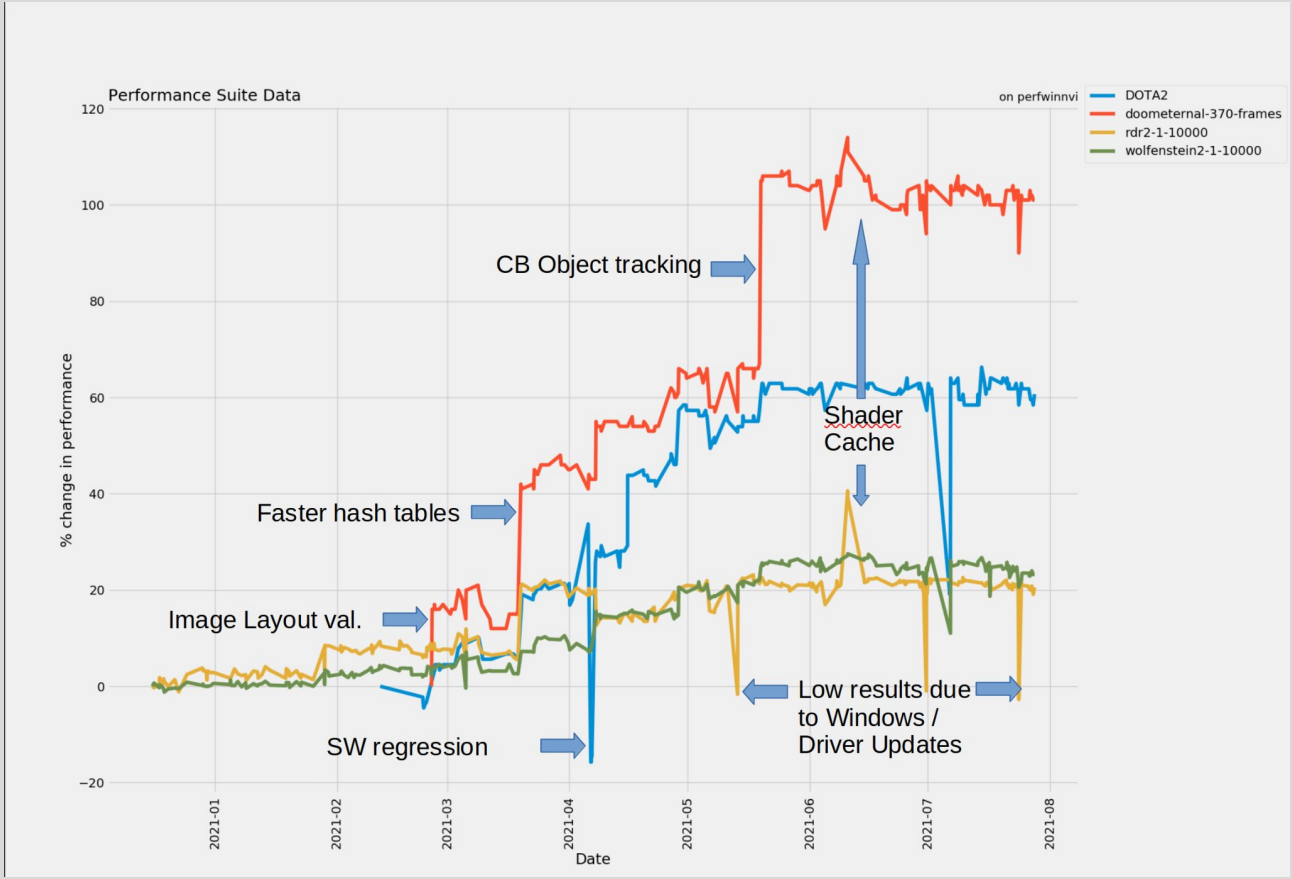
Synchronization Validation is not enabled by default

- To enable, perform the following:
  - vk_layer_settings.txt file:
    khronos_validation.enables=VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT
  - Environment variable: Windows: set
    VK_LAYER_ENABLES=VK_VALIDATION_FEATURE_ENABLE_SYNCHRONIZATION_VALIDATION_EXT
  - Or use the Vulkan Configurator to enable/disable synchronization validation

Validation Layer Performance Improvement Initiative

- Performance regression test suite
  - Catches performance regressions (avoid performance degradation over time)
- Problem areas
  - Many active threads in Vulkan applications; but a single lock per Validation Object in the Validation Layer
  - Image Layout Transition validation
  - Large “bindless” DescriptorSets
- Some performance optimizations so far
  - Image Layout Transition validation
  - “Command Buffer State” validation; track which are in use by each command buffer
  - Faster hash table implementation (for Vulkan Objects)
  - Shader validation results caching (controlled by a Validation Layer setting)
- Future areas of work
  - Finer grained locking in the Validation Layer (in progress)
  - Further Image Layout Transition optimizations
  - Then look for the next bottleneck...

Validation Layer Performance Improvement Initiative



GPU-Assisted Validation

- Much of the validation performed by the Vulkan Validation Layer is done on the CPU
  - However much of an application's activity is on the GPU
  - GPU-Assisted validation first introduced January 2019 but has had many enhancements over the last few years
- GPU-Assisted Validation is comprised of:
  - Instrumenting shader code at runtime to perform runtime checking of shaders
  - Reporting any error conditions to the Validation Layer (and then to the user)
- Types of GPU-Assisted Validation available today:
  - Bindless Descriptor Access
    - Validating indexing into an array of descriptors
  - Descriptor Indexing
    - Validating scenarios that arise when the VK_EXT_descriptor_indexing extension is enabled
  - Buffer Device Address Validation
    - Validation of buffer accesses with the VK_EXT_buffer_device_address enabled
  - Buffer Accesses Out of Bounds
    - Validate buffer accesses beyond the declared size of the buffer

See White Paper: https://www.lunarg.com/news-insights/white-papers/vulkan-gpu-assisted-validation/

GPU-Assisted Validation - Activation

- GPU-Assisted Validation is off by default and must be enabled
  - Performance loss due to the shader instrumentation heavily dependent upon the shader
  - Can’t be co-enabled with DebugPrintf
- vk_layer_settings.txt:
  - khronos_validation.enables = VK_VALIDATION_FEATURE_ENABLE_GPU_ASSISTED_EXT
- Configure using the Vulkan Configurator

Debug Printf

- Debug Printf allows developers to debug their shaders
  - Insertion of debug printf statements to retrieve values of variables, determine execution paths, etc.
  - Output is sent to the validation layer debug callback
- Debug Printf can be used programmatically:
  - SPIR-V shaders using:
    - OpExtension "SPV_KHR_non_semantic_info"
    - %N0 = OpExtInstImport NonSemantic.DebugPrintf
  - GLSL shaders using the GL_EXT_debug_printf extension
  - HLSL shaders (use glslangValidator or DXC to generate SPIR-V for the shader)
- Debug Printf can be used from RenderDoc
  - Release 1.14 or later
  - Debug Printf message will be received and logged in the Event Browser window

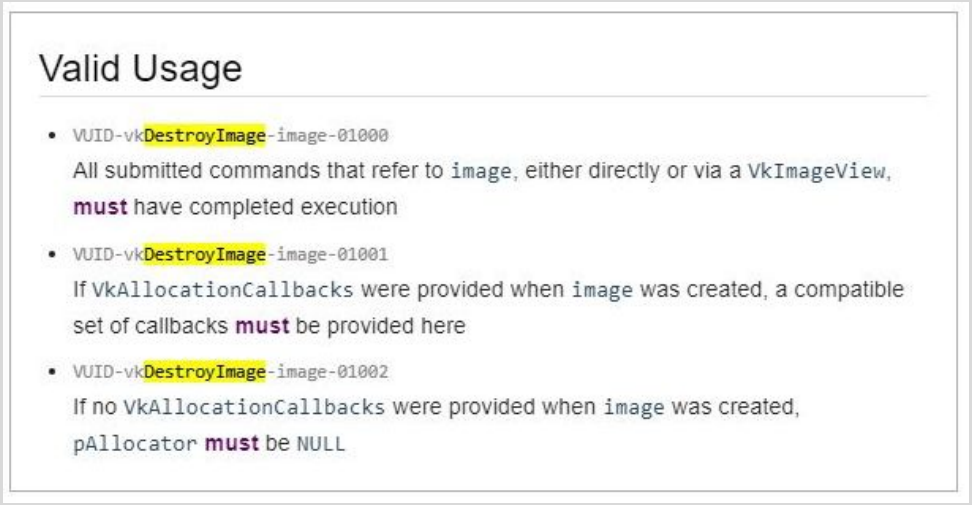
> Dan Ginsberg (Valve) - "Vulkan Debug Printf + @RenderDoc is a great way to debug compute shaders. You can use it to easily narrow in on a particular problematic invocation."

See White Paper: https://www.lunarg.com/news-insights/white-papers/using-debug-printf/

Validation Layer Error Reporting Improvements

VUID annotated version of the Vulkan Specification

- Plain-text VUIDs are part of the VUID text included
- Persistent links to the SDK version of the specification
- Specific VUID included in the validation error message is clearly visible and gives the user immediate context and clarity as to the source of the violation:



See White Paper: www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements_08_20.pdf

LUNARG

Validation Layer Error Reporting Improvements

Unique Message Identifiers

- Validation message output includes a hash of the VUID string, allowing simpler handling of specific messages in message callbacks or debugging utilities.
- Numeric value is printed as part of the normal message output and also included in the callback data as the message IDNumber:

```
Validation Error: [ VUID-VkImageResolve-dstImage-00276 ] Object 0: handle = 0x2aa03237048, type =
VK_OBJECT_TYPE_COMMAND_BUFFER; Object 1: handle = 0x3ba5830000000006, type =
VK_OBJECT_TYPE_IMAGE; | MessageID = 0x3c65a6c9 | vkCmdResolveImage(): dstImage (VkImage
0x3ba5830000000006[]) is 1D but pRegions[0] dstOffset.y (0) is not 0 or extent.height (2) is not 1. The
Vulkan spec states: If the calling command's dstImage...
```

See White Paper: www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements_08_20.pdf

Validation Layer Error Reporting Improvements
Message Filtering

- Allows suppression of specific validation messages
- Example: Preventing output of two messages with message ID Numbers of 3012204 and 0x02044177
  - vk_layer_settings.txt file:
    - khronos_validation.message_id_filter=3012204, 0x02044177
  - Environment variable:
    - Windows: set VK_LAYER_MESSAGE_ID_FILTER=3012204;0x02044177
    - Linux: export VK_LAYER_MESSAGE_ID_FILTER=3012204:0x02044177
  - Configure using the Vulkan Configurator

See White Paper: www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements_08_20.pdf

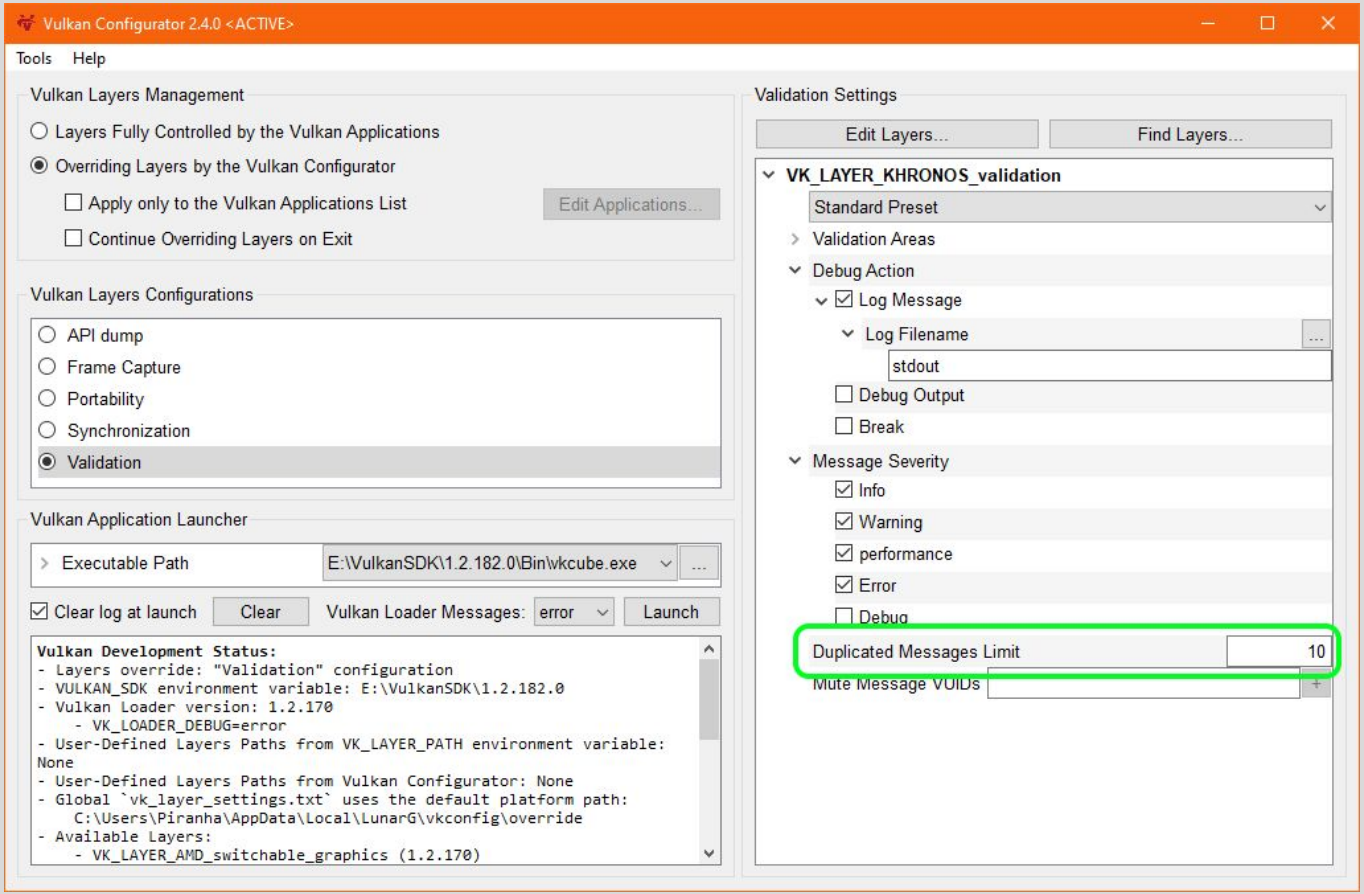
Validation Layer Error Reporting Improvements:
Message Duplication Limit

- Limit the number of times a single validation message is reported
- Example: Cap the number of message repeats to 10:
  - Vk_layer_settings.txt file:
    - khronos_validation.duplicate_message_limit=10
  - Environment variable:
    - Windows: set VK_LAYER_DUPLICATE_MESSAGE_LIMIT=10
    - Linux: export VK_LAYER_DUPLICATE_MESSAGE_LIMIT=10
  - Configure using the Vulkan Configurator (next slide):

See White Paper: www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements_08_20.pdf

Validation Layer - Message Duplication Limit

Using the Vulkan Configurator:



Validation Layer Error Reporting Improvements

Plain-text Object Types and Names

- Validation messages previously outputted the object type with the handle as an integer enumeration value.
- Messaging now outputs the enumeration name with the object

Validation Error: [VUID-VkImageResolve-srcImage-00268] Object 0: **handle = 0x2aa03237048, type = VK_OBJECT_TYPE_COMMAND_BUFFER**; Object 1: **handle = 0x983e60000000003, type = VK_OBJECT_TYPE_IMAGE**; Object 2: **handle = 0xa540ac0000000009, type = VK_OBJECT_TYPE_IMAGE**; | MessageID = 0x7711e6f5 | vkCmdResolveImage(): pRegions[0] baseArrayLayer must be 0 and layerCount must be 1 for all subresources if the src or dst image is 3D...

See White Paper: www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements_08_20.pdf

Validation Layer Error Reporting Improvements

Returning Relevant Vulkan Objects in Debug Callback

- Validation Layers have been scrubbed to ensure
  - Any object output in the text of a message is also passed to the user through the callback
  - This allows programmatic access to these objects for debugging purposes
  - Each of the Vulkan objects returned also includes the object name/label

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsMessengerCallbackDataEXT {
    VkStructureType                              sType;
    const void*                                  pNext;
    VkDebugUtilsMessengerCallbackDataFlagsEXT    flags;
    const char*                                  pMessageIdName;
    int32_t                                      messageIdNumber;
    const char*                                  pMessage;
    uint32_t                                     queueLabelCount;
    const VkDebugUtilsLabelEXT*                  pQueueLabels;
    uint32_t                                     cmdBufLabelCount;
    const VkDebugUtilsLabelEXT*                  pCmdBufLabels;
    uint32_t                                     objectCount;
    const VkDebugUtilsObjectNameInfoEXT*         pObjects;
} VkDebugUtilsMessengerCallbackDataEXT;
```

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsObjectNameInfoEXT {
    VkStructureType    sType;
    const void*        pNext;
    VkObjectType       objectType;
    uint64_t           objectHandle;
    const char*        pObjectName;
} VkDebugUtilsObjectNameInfoEXT;
```

See White Paper: www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements_08_20.pdf

GFXReconstruct

Capture and Replay your Vulkan API calls

- As of June 2020, replaces vktrace/vkreplay
- Repository: https://github.com/LunarG/gfxreconstruct
- Key Features
  - Excellent Android capability & reliability
  - Automatic code generation to accommodate evolving API
  - Reliable trimming
  - Extract/replace shaders
  - Increased portability
    - X86 vs. x64 differences
    - Cross OS portability (i.e. trace on windows, replay on linux. Same GPU/driver)
    - WIP: Cross vendor GPU support (trace on one GPU, replay on another)
  - LZ4 compression for trace data
- Designed to minimize performance impact
- Dan Ginsburg (Valve) Success Story
  - Easily used tool to debug DEVICE_LOST on NVIDIA with Half Life: Alyx
  - Created capture, sent to NVIDIA

Device Simulation Layer - Introduction

- Devsim helps ensure Vulkan applications are portable for the minimum set of capabilities of the target market
- Devsim simulates the following capabilities: limits, properties, and features of a device
  - Modifies device responses to application queries
  - The underlying device or driver function is never changed, it merely appears to have the capabilities of a different device
  - The Validation Layer then informs the developer about application behaviors that do not adhere to the proper limits

Device Simulation Layer - Add Vulkan 1.1 and 1.2 support

- Added support for Vulkan 1.1 and 1.2 core capabilities
- Added minimum configuration files:
  - Minimum properties, limits, and features based on the Vulkan 1.0, 1.1, and 1.2 specifications
- Added desktop portability configuration files:
  - Minimum properties, limits, and features based on the actual Vulkan 1.0 and 1.2 desktop hardware ecosystem

Device Simulation Layer - Add VK_KHR_portability_subset

- Added support for the `VK_KHR_portability_subset` extension
  - Devsim add the `VK_KHR_portability_subset` extension to the device extensions list,
  - Pre-populates the `VkPhysicalDevicePortabilitySubsetPropertiesKHR` and `VkPhysicalDevicePortabilitySubsetFeaturesKHR` structures provided by said extension with default values.
  - For more information about the Khronos Portability initiative read the Khronos Portability Blog (https://www.khronos.org/blog/new-release-of-vulkan-sdk)

Device Simulation Layer - Array Combination Modes

- Added new layer settings to control the simulation of system capabilities
  - Array combination modes for better modification of queries that return arrays
  - Support for surface query modification:
    - Surface capabilities
    - Surface formats
    - Present modes

For more information, see the SDK Devsim documentation at
https://vulkan.lunarg.com/doc/sdk/1.2.182.0/windows/device_simulation_layer.html

Coming soon: An in-depth white paper detailing devsim benefits and usage

Device Simulation Layer - Configuration Files Changes

- Devsim config files use the JSON format to store data about a simulated device
  - In the past, these values were strictly defined by JSON schemas hosted at https://schema.khronos.org/vulkan/.
    - Requires multiple files to describe different data for the same device
- Coming soon:
  - Deprecation of the schemas at https://schema.khronos.org/vulkan
  - Read all supported JSON elements from the configuration files
    - Allows the entire device capabilities to be defined in a single file
  - Support provided for the format used by JSON files found at gpuinfo.org

Shader Toolchain additions to the Vulkan SDK

- There are two workflows used for generating SPIR-V
  - glsl->SPIR-V
    - glslangValidator
  - hlsl->SPIR-V
    - glslangValidator (supports up to shader model 5)
    - DXC (DirectX shader compiler)
- Originally the SDK only included the offline executables for generating SPIR-V (glslangValidator, DXC)
- All of the API libraries for glslang and DXC are now also included in the SDK
  - SPIRV-Tools (validator, optimizer, assembler, disassembler)
  - glslang SPIR-V generator
  - Spirv-cross cross compiler and reflection
  - DXC (DirectX Shader Compiler),
  - Shaderc SPIRV-Tools wrapper for better integration with build tools

Simplify Vulkan Development with new Ecosystem Enhancements - SIGGRAPH 2021

Slides are available at:
https://www.lunarg.com/news-insights/white-papers/vulkan-ecosystem-enhancements-siggraph-2021/

After the presentation

Questions or presentation feedback?

Contact Karen Ghavam: @kghavam on the Vulkan KhronosDevs slack channel

- https://app.slack.com/client/TDMDFS87M/CDTJ9BELF

- Or sign up for the KhronosDevs slack channel here: https://www.khronos.org/news/permalink/khronos-developer-slack-5bfc62eb261764.20435008