

A night sky with a full moon and a silhouette of a tree. The background is a deep blue starry sky with a large, bright full moon in the upper right. In the lower left, the dark silhouette of a tree stands against the sky. The overall scene is serene and celestial.

# Simplify Vulkan Development with new Ecosystem Enhancements

Karen Ghavam  
LunarG, Inc

# 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/>

# 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)

The screenshot shows the Vulkan SDK website homepage. At the top left is the Vulkan logo. In the top right corner, there are two buttons: a green '+ Signup' button and a blue 'Signin' button with a user icon. On the left side, there is a navigation menu with the following items: 'SDK' (highlighted with a red box and a red arrow pointing to it), 'Issues', 'Docs', 'Licenses', and 'Khronos'. Below the navigation menu, it says 'Sponsored by' followed by the VALVE logo, and 'Developed by' followed by the LUNAR)G logo. The main content area features five news items, each with a 'Learn more' button: 1. 'New Vulkan 1.2.182.0 SDKs release new extensions, devsim enhancements, and macOS updates' with a 'Learn more' button. 2. 'LunarG publishes updated white paper: SDK Version Compatibility for Vulkan' with a 'Learn more' button. 3. 'Have you tried the new Vulkan Configurator (vkconfig)? New demo video and white paper available' with a 'Learn more' button. 4. 'New LunarG white paper: The State of Vulkan on Apple Devices' with a 'Learn more' button. 5. 'The Vulkan Capabilities Viewer is now available on iOS!' with a 'Learn more' button. At the bottom of the page, there is a large Vulkan logo, a welcome message: 'Welcome to the community for the Vulkan SDK. You can download the latest Vulkan SDK and get SDK questions answered at this site.', and a footer with 'info@lunarg.com' on the left and '© 2021 LunarG, Inc. Privacy Policy' on the right. The LUNAR)G logo is also present in the bottom right corner.

## DOWNLOAD DEVELOPER TOOLS FOR



SDK version query and download API

Sponsored by



Developed by



### Windows

[Download](#) Latest SDK  
[Download](#) Latest Runtime/Zip

Version Released	File SHA 256
<b>1.2.182.0</b> 05-Jul-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>VulkanSDK-1.2.182.0-Installer.exe</b> (256MB)  <small>7688d48011f38e7a012a5639e13c3a8385d1652ab7765c25d2966d5d9e4e24af8</small> </li> <li> <a href="#">Shader Libs (Debug) - Shader Debug Libs</a>  <b>VulkanSDK-1.2.182.0-DebugLibs.zip</b> (360MB)  <small>d210675c49508197b01c400096e18ef9180435d3b1bbee33597ecd5a79ac2aae7</small> </li> <li> <a href="#">SDK Config - Config.json</a>  <b>config.json</b> (0MB)  <small>0a9553bdc90dedadcd04c155a28990c9f6fc23fdd77675745412e3e42dad2</small> </li> <li> <a href="#">Runtime - Runtime Installer</a>  <b>VulkanRT-1.2.182.0-Installer.exe</b> (1MB)  <small>82396e969104950e5d28e49ab79933e0e2c2980734d1dfec15482a52eecc9</small> </li> <li> <a href="#">Runtime zip - Zip file of the runtime components.</a>  <b>VulkanSDK-1.2.182.0-Components.zip</b> (9MB)  <small>0bc391388543ca08f80444754da04c4958a9fd8b43ef83791e8f9cc70b2020b9</small> </li> </ul>
<b>1.2.176.1</b> 05-May-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>VulkanSDK-1.2.176.1-Installer.exe</b> (254MB)  <small>55d836d7f60394969d4be53be720330a4bf93704a02165aaa7bb33712cabd41</small> </li> <li> <a href="#">Shader Libs (Debug) - Shader Debug Libs</a>  <b>VulkanSDK-1.2.176.1-DebugLibs.zip</b> (356MB)  <small>1ed8de68844f28b6ba03e479f984330beeceaf9ba92c2787de587c95b3</small> </li> </ul>

### Linux

[Download](#) Latest SDK Tarball

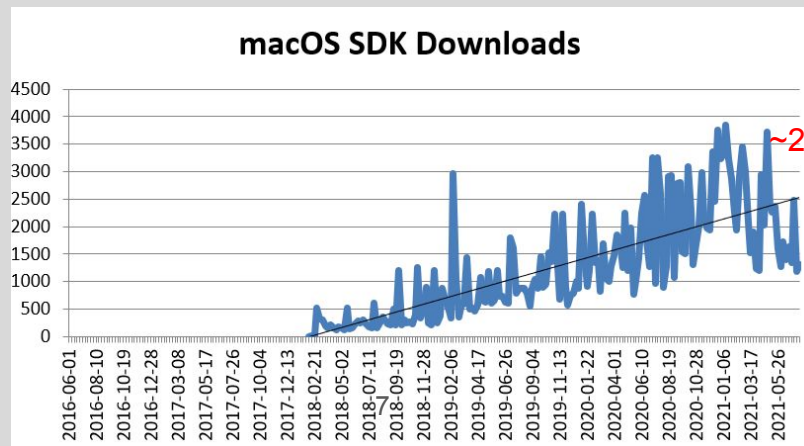
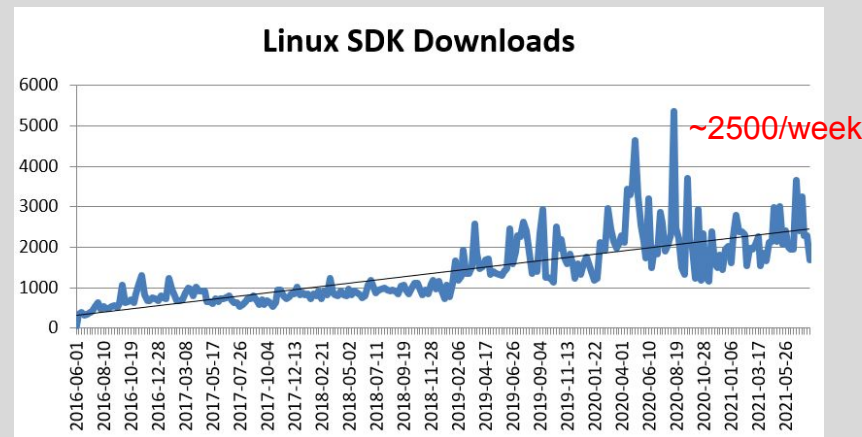
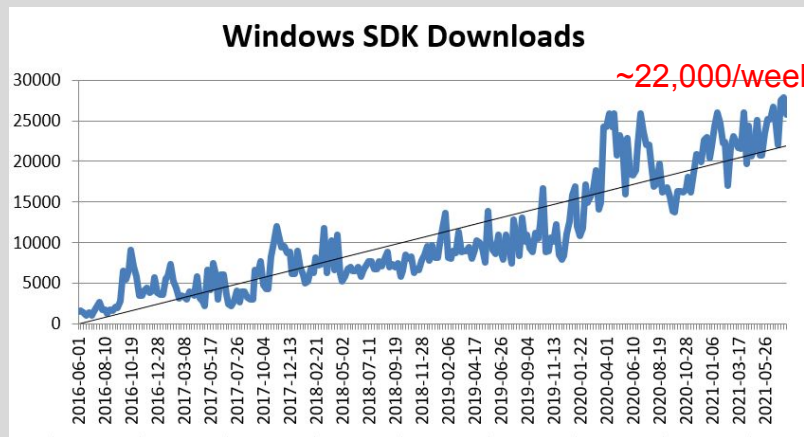
Version Released	File SHA 256
<b>1.2.182.0</b> 05-Jul-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>vulkansdk-linux-x86_64-1.2.182.0.tar.gz</b> (207MB)  <small>359f25ebf0ec9e73f9100a9e00219623a01a1e322e087ee07b20073c16203aae</small> </li> <li> <a href="#">SDK Config - Config.json</a>  <b>config.json</b> (0MB)  <small>f6599a8b71d0786e11ad9e33bcc07aabc2259e8a9f1d45484b02ed3aba20d943a</small> </li> </ul>
<b>1.2.176.1</b> 05-May-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>vulkansdk-linux-x86_64-1.2.176.1.tar.gz</b> (207MB)  <small>9f6300f1584a9e90f173bd7722b6e7ea8f1913b0873bd5ce938b28482c245b5</small> </li> <li> <a href="#">SDK Config - Config.json</a>  <b>config.json</b> (0MB)  <small>3817a6825981433ef7f699959d73e9ebfa9baffcc92fb7849672bdad0d7740b</small> </li> </ul>
<b>1.2.170.0</b> 26-Feb-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>vulkansdk-linux-x86_64-1.2.170.0.tar.gz</b> (206MB)  <small>c1e4b6e824479273383c892a9468ab7f6b85804e890d157279fb9448f01f0d</small> </li> <li> <a href="#">SDK Config - Config.json</a>  <b>SDK-1.2.170.0-CONFIG.json</b> (0MB)  <small>86c308b2b15292a9b315aaa0ab74b5e89380c1792bd0378e82259fb21a3fdb38</small> </li> </ul>
<b>1.2.162.1</b> 12-Jan-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a> </li> </ul>

### MacOS

[Download](#) Latest SDK

Version Released	File SHA 256
<b>1.2.182.0</b> 05-Jul-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>config.json</b> (0MB)  <small>f6599a8b71d0786e11ad9e33bcc07aabc2259e8a9f1d45484b02ed3aba20d943a</small> </li> <li> <a href="#">SDK - SDK Installer</a>  <b>vulkansdk-macos-1.2.182.0.dmg</b> (119MB)  <small>9e89291283e41d82673bf48007dac784e99a504d145ee4d70d2d32e3287ee40</small> </li> </ul>
<b>1.2.176.1</b> 05-May-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>config.json</b> (0MB)  <small>3817a6825981433ef7f699959d73e9ebfa9baffcc92fb7849672bdad0d7740b</small> </li> <li> <a href="#">SDK - SDK Installer</a>  <b>vulkansdk-macos-1.2.176.1.dmg</b> (120MB)  <small>9e5c20e4e403379fa8a9699590306021220e5370c2271719203eea4c292</small> </li> </ul>
<b>1.2.170.0</b> 26-Feb-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>vulkansdk-macos-1.2.170.0.dmg</b> (469MB)  <small>5c7264e96c57918917a2b62dc052f6bc0a671915819b4c9420cd4317808372d</small> </li> <li> <a href="#">SDK Config - Config.json</a>  <b>SDK-1.2.170.0-CONFIG.json</b> (0MB)  <small>86c308b2b15292a9b315aaa0ab74b5e89380c1792bd0378e82259fb21a3fdb38</small> </li> </ul>
<b>1.2.162.1</b> 12-Jan-2021	<ul style="list-style-type: none"> <li> <a href="#">SDK - SDK Installer</a>  <b>vulkansdk-macos-1.2.162.1.dmg</b> (471MB)  <small>2781e0334987598c2828e8a3398aef7b7c84a29204c60d9503396e40c7a03fd5c</small> </li> </ul>

# Vulkan SDK Downloads are Healthy and Continue to Grow



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

**Vulkan**

+ Signup Signin

- SDK
- Issues
- Docs
- Licenses**
- Khronos

Sponsored by **VALVE**

Developed by **LUNAR)G**

**Vulkan**

info@lunarg.com

© 2021 LunarG, Inc. Privacy Policy

New Vulkan 1.2.182.0 SDKs release new extensions, devsim enhancements, and macOS updates [Learn more](#)

LunarG publishes updated white paper: SDK Version Compatibility for Vulkan [Learn more](#)

Have you tried the new Vulkan Configurator (vkconfig)? New demo video and white paper available [Learn more](#)

New LunarG white paper: The State of Vulkan on Apple Devices [Learn more](#)

The Vulkan Capabilities Viewer is now available on iOS! [Learn more](#)

Welcome to the community for the Vulkan SDK. You can download the latest Vulkan SDK and get SDK questions answered at this site.

🔑 Releases	<b>Releases</b>
📦 Packages	1.2.182.0
📄 Licenses	1.2.176.1
📖 README	1.2.176.0
	1.2.170.0
	1.2.162.1
	1.2.162.0
	1.2.154.0
	1.2.148.0
	1.2.141.0
	1.2.135.0
	1.2.131.2
	1.2.131.1
	1.2.131.0
	1.1.130.0

## Licenses for Vulkan SDK Version 1.2.182.0

Summary includes the text of each license included in the SDK. Detail includes a list of all source files and their associated license and copyright information.



📄 License Summary (txt)

📄 License Detail (csv)



📄 License Summary (txt)

📄 License Detail (csv)



📄 License Summary (txt)

📄 License Detail (csv)



📄 License Summary (txt)

📄 License Detail (csv)

- 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:

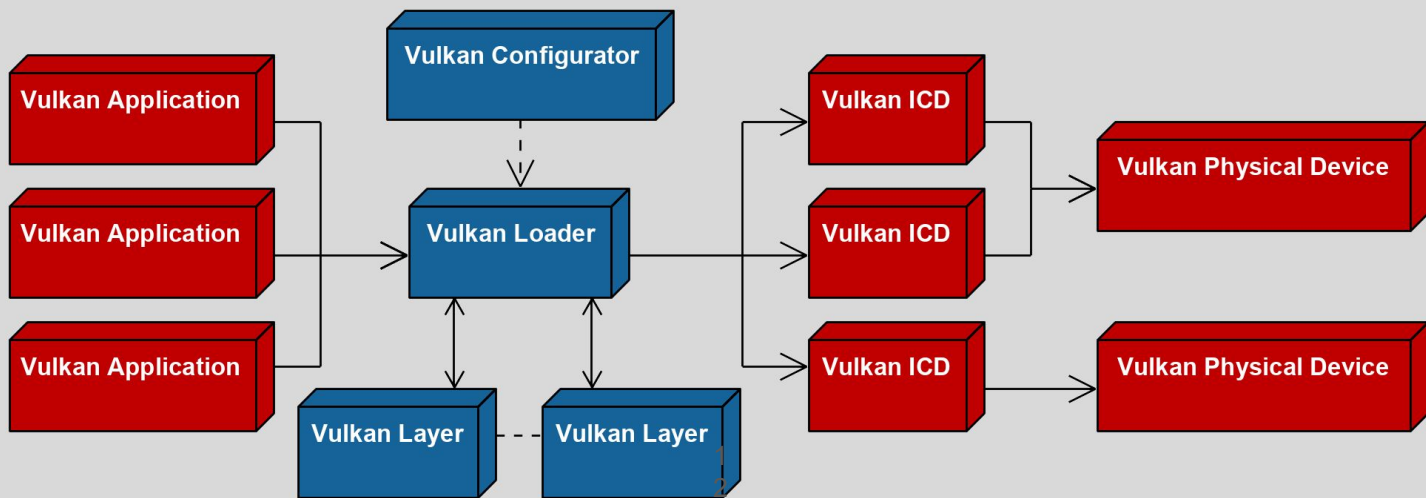
1. **VK\_LAYER\_KHRONOS\_validation** - validate application correct usage of the Vulkan API
  - a. **GPU Assisted Validation** - runtime validation executed on the GPU (rather than the CPU)
  - b. **Best Practice** - catch correct Vulkan API usage that still could cause application issues
  - c. **Synchronization 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.



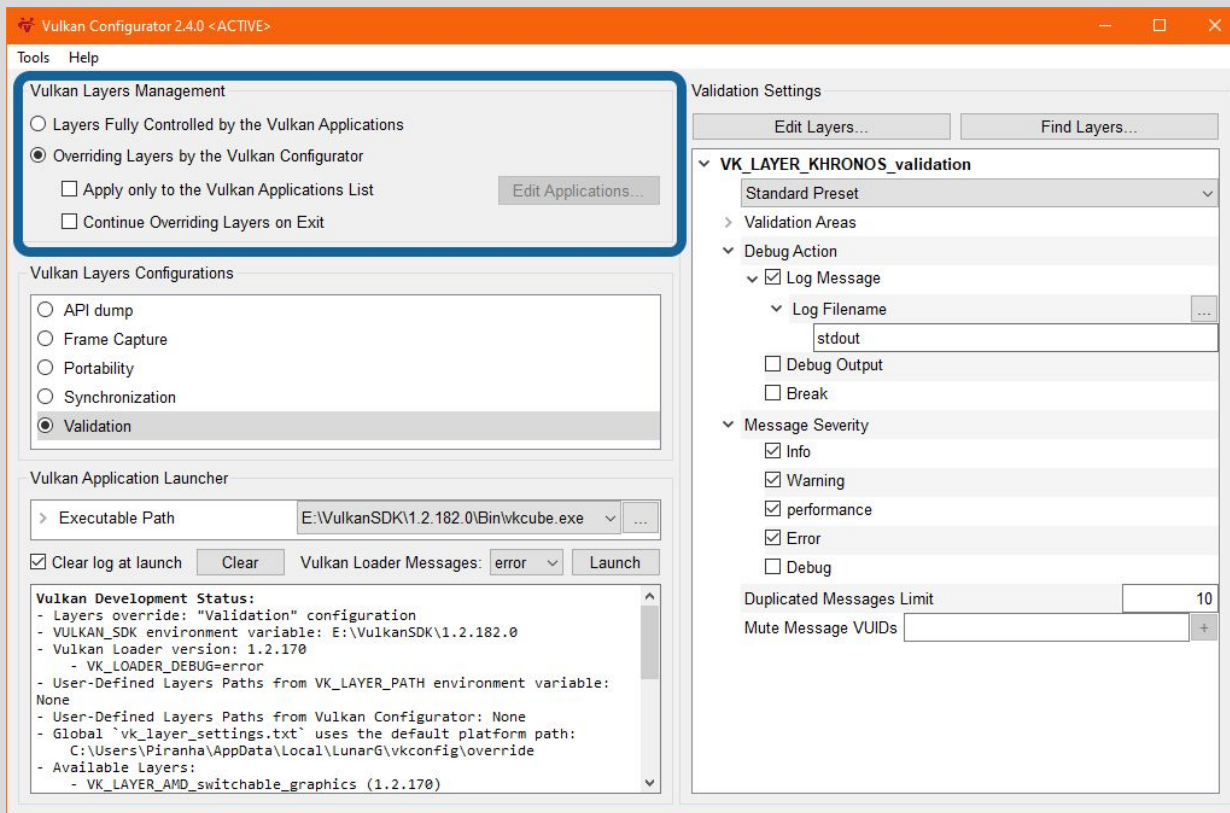
# Vulkan Configurator: UI Layout

The screenshot displays the Vulkan Configurator 2.4.0 interface, which is organized into several main sections:

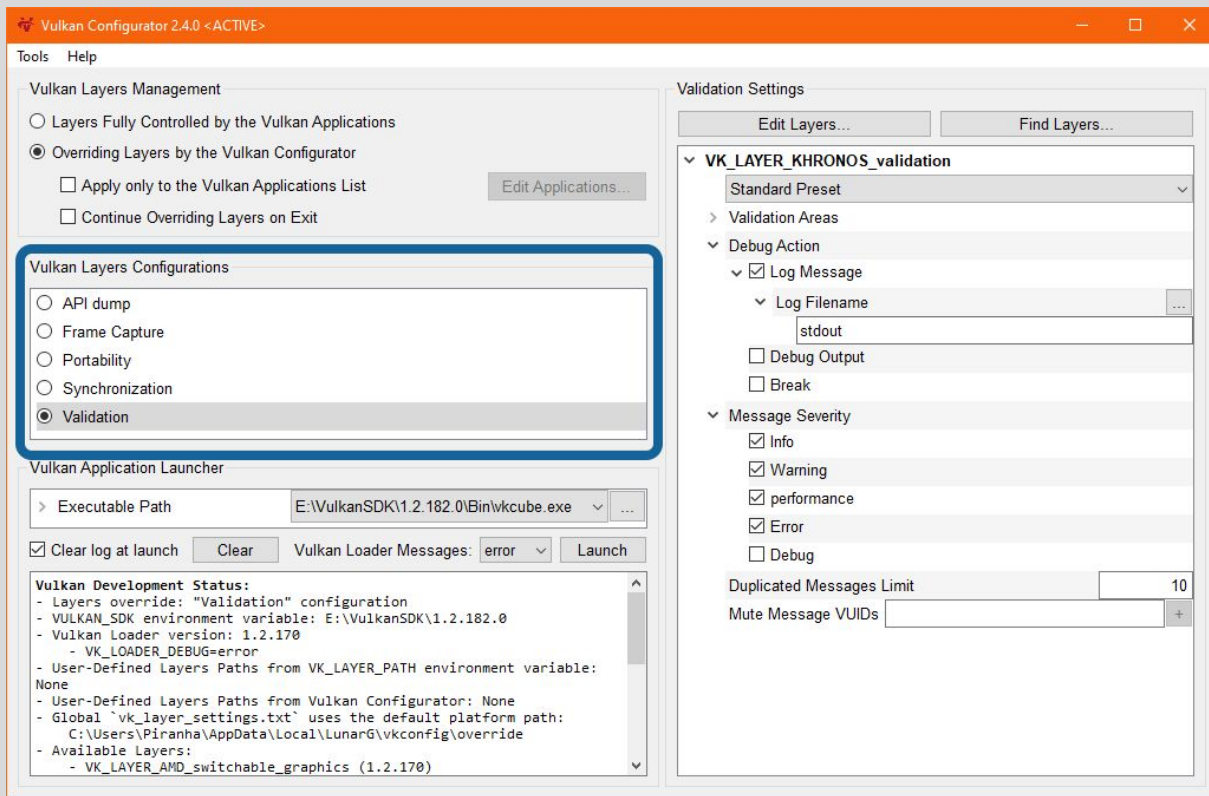
- Vulkan Layers Management:** Contains radio buttons for "Layers Fully Controlled by the Vulkan Applications" and "Overriding Layers by the Vulkan Configurator" (selected). Below are checkboxes for "Apply only to the Vulkan Applications List" and "Continue Overriding Layers on Exit", along with an "Edit Applications..." button.
- Vulkan Layers Configurations:** Features radio buttons for "API dump", "Frame Capture", "Portability", "Synchronization", and "Validation" (selected).
- Vulkan Application Launcher:** Includes a text field for the "Executable Path" (set to "E:\VulkanSDK\1.2.182.0\Bin\vkcube.exe"), a "Clear log at launch" checkbox, a "Clear" button, a "Vulkan Loader Messages" dropdown (set to "error"), and a "Launch" button.
- Validation Settings:** Contains an "Edit Layers..." button, a "Find Layers..." button, and a tree view for "VK\_LAYER\_KHRONOS\_validation". This tree view includes:
  - "Standard Preset" dropdown (set to "Standard Preset")
  - "Validation Areas" (expanded)
  - "Debug Action" (expanded) with a checked "Log Message" option and a "Log Filename" dropdown (set to "stdout").
  - "Message Severity" (expanded) with checked options for "Info", "Warning", and "performance", and an unchecked "Error" option.
  - "Duplicated Messages Limit" set to "10".
  - "Mute Message VUIDs" field with a "+" button.
- Vulkan Development Status:** A scrollable text area showing system and application details:

```
Vulkan Development Status:  
- Layers override: "Validation" configuration  
- VULKAN_SDK environment variable: E:\VulkanSDK\1.2.182.0  
- Vulkan Loader version: 1.2.170  
  - VK_LOADER_DEBUG=error  
- User-Defined Layers Paths from VK_LAYER_PATH environment variable:  
None  
- User-Defined Layers Paths from Vulkan Configurator: None  
- Global `vk_layer_settings.txt` uses the default platform path:  
  C:\Users\Piranha\AppData\Local\LunarG\vkconfig\override  
- Available Layers:  
  - VK_LAYER_AMD_switchable_graphics (1.2.170)
```

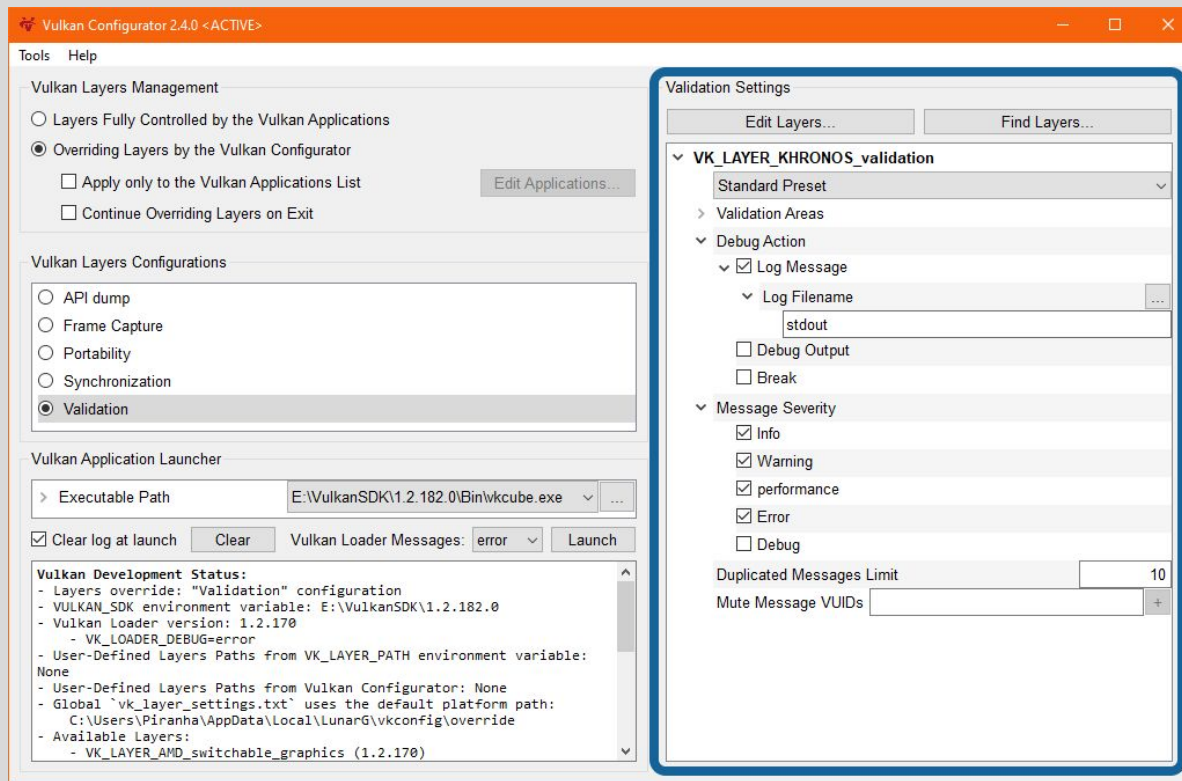
# Vulkan Configurator: Layers Management modes



# Vulkan Configurator: Layer Configurations



# Vulkan Configurator: Layer Settings



# Vulkan Configurator: Application Launcher

**Vulkan Layers Management**

- Layers Fully Controlled by the Vulkan Applications
- Overriding Layers by the Vulkan Configurator
  - Apply only to the Vulkan Applications List
  - Continue Overriding Layers on Exit

**Vulkan Layers Configurations**

- API dump
- Frame Capture
- Portability
- Synchronization
- Validation

**Vulkan Application Launcher**

Executable Path: E:\VulkanSDK\1.2.182.0\Bin\vkcube.exe

Clear log at launch  Vulkan Loader Messages: error

**Vulkan Development Status:**

- Layers override: "Validation" configuration
- VULKAN\_SDK environment variable: E:\VulkanSDK\1.2.182.0
- Vulkan Loader version: 1.2.170
  - VK\_LOADER\_DEBUG=error
- User-Defined Layers Paths from VK\_LAYER\_PATH environment variable: None
- User-Defined Layers Paths from Vulkan Configurator: None
- Global `vk\_layer\_settings.txt` uses the default platform path: C:\Users\Piranha\AppData\Local\LunarG\vkconfig\override
- Available Layers:
  - VK\_LAYER\_AMD\_switchable\_graphics (1.2.170)

**Validation Settings**

Standard Preset: Standard Preset

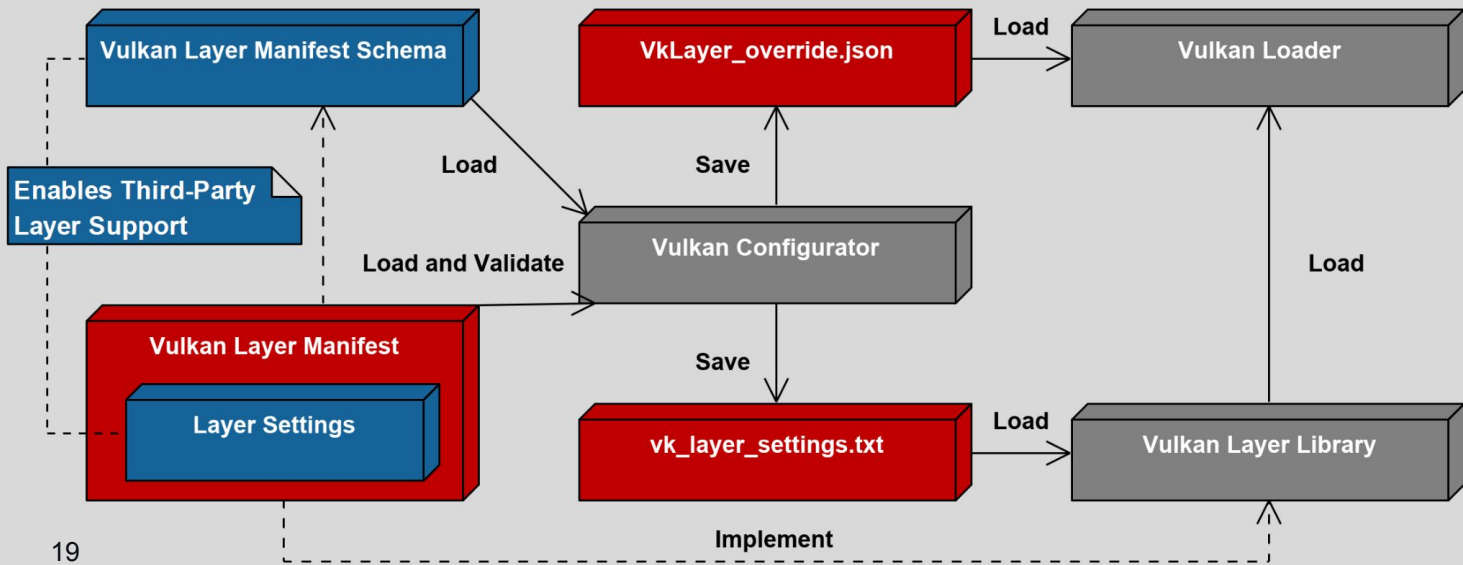
- Validation Areas
- Debug Action
  - Log Message
    - Log Filename: stdout
  - Debug Output
  - Break
- Message Severity
  - Info
  - Warning
  - performance
  - Error
  - Debug

Duplicated Messages Limit: 10

Mute Message VUIDs: +

# 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

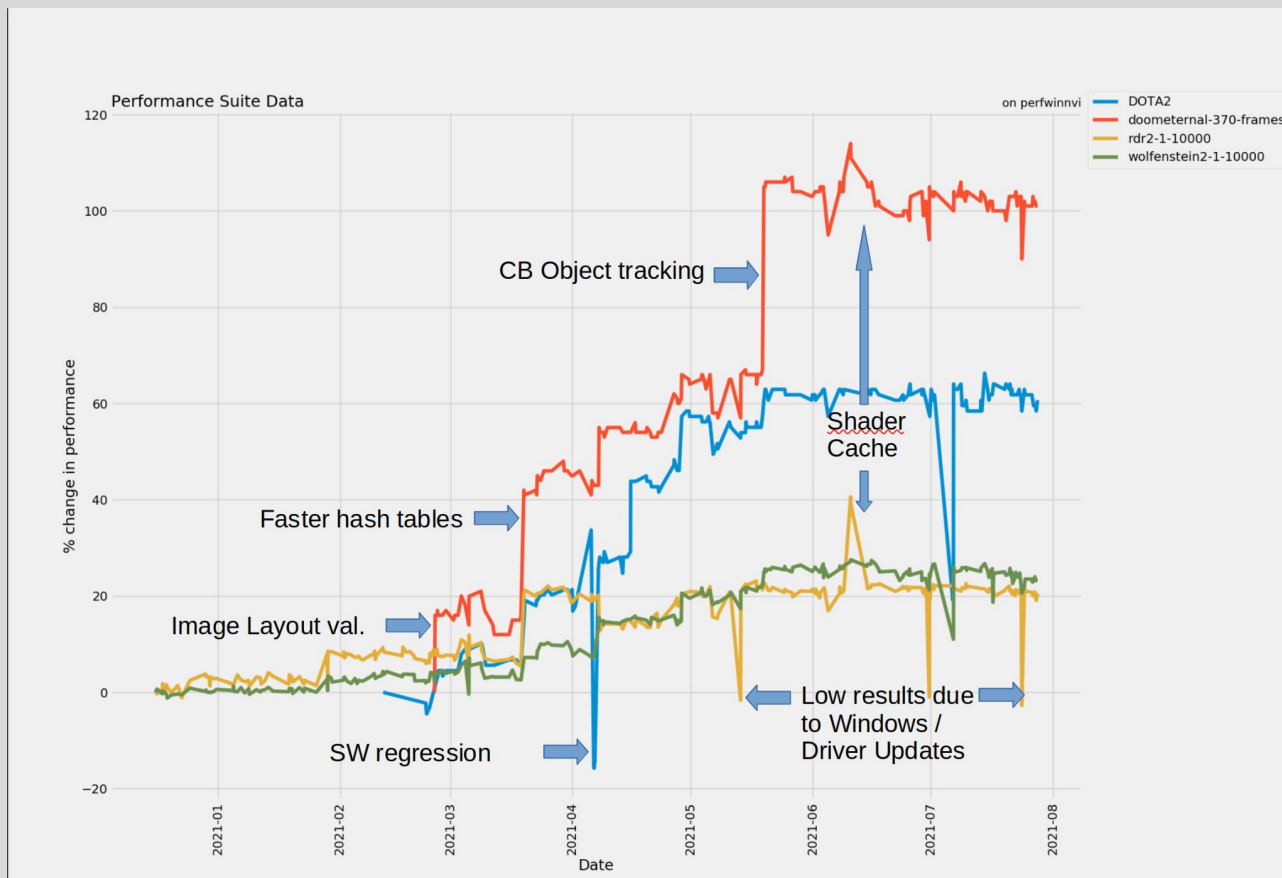
## Synchronization Validation is not enabled by default

- 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:

### Valid Usage

- VUID-vkDestroyImage-image-01000  
All submitted commands that refer to `image`, either directly or via a `VkImageView`, **must** have completed execution
- VUID-vkDestroyImage-image-01001  
If `VkAllocationCallbacks` were provided when `image` was created, a compatible set of callbacks **must** be provided here
- VUID-vkDestroyImage-image-01002  
If no `VkAllocationCallbacks` were provided when `image` was created, `pAllocator` **must** be NULL

See White Paper: [www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements\\_08\\_20.pdf](http://www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements_08_20.pdf)

# 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](http://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](http://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](http://www.lunarg.com/wp-content/uploads/2020/08/Final-ValidationLayerErrorReportingImprovements_08_20.pdf)

# Validation Layer - Message Duplication Limit

## Using the Vulkan Configurator:

The screenshot shows the Vulkan Configurator 2.4.0 interface. The 'Validation Settings' panel is open, showing the configuration for the 'VK\_LAYER\_KHRONOS\_validation' layer. The 'Duplicated Messages Limit' is highlighted with a green box and set to 10. Other settings include 'Log Message' (checked), 'Log Filename' (stdout), 'Debug Output' (unchecked), 'Break' (unchecked), 'Message Severity' (Info, Warning, performance, Error checked; Debug unchecked), and 'Mute Message VOIDS' (empty field).

**Vulkan Layers Management**

- Layers Fully Controlled by the Vulkan Applications
- Overriding Layers by the Vulkan Configurator
  - Apply only to the Vulkan Applications List
  - Continue Overriding Layers on Exit

**Vulkan Layers Configurations**

- API dump
- Frame Capture
- Portability
- Synchronization
- Validation

**Vulkan Application Launcher**

Executable Path: E:\VulkanSDK\1.2.182.0\Bin\vkcube.exe

Clear log at launch

Vulkan Loader Messages: error

**Vulkan Development Status:**

- Layers override: "Validation" configuration
- VULKAN\_SDK environment variable: E:\VulkanSDK\1.2.182.0
- Vulkan Loader version: 1.2.170
  - VK\_LOADER\_DEBUG=error
- User-Defined Layers Paths from VK\_LAYER\_PATH environment variable: None
- User-Defined Layers Paths from Vulkan Configurator: None
- Global "vk\_layer\_settings.txt" uses the default platform path: C:\Users\Piranha\AppData\Local\LunarG\vkconfig\override
- Available Layers:
  - VK\_LAYER\_AMD\_switchable\_graphics (1.2.170)

# 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](http://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](http://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) (<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](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](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](http://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>

