

## Common Vulkan Installation Issues

Charles Giessen, LunarG



Presentation:  
<http://bit.ly/3RUe4eg>



# Summary

- Understanding the Vulkan-Loader
- Guide to the loader log
- Issues loading the Vulkan-Loader
- Debugging layer issues
- Dealing with Implicit Layers
- Debugging driver issues

# DISCLAIMER

- This talk is exclusively about the Desktop Vulkan-Loader!
- While some of the architectural ideas are shared, android has its own loader

# Understanding the Vulkan-Loader

# What is the Vulkan-Loader?

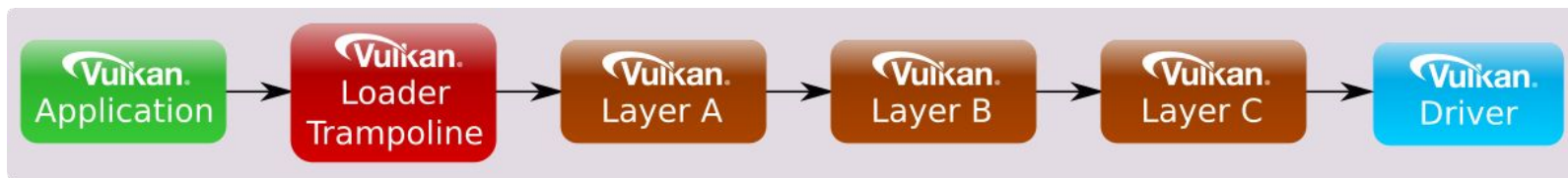
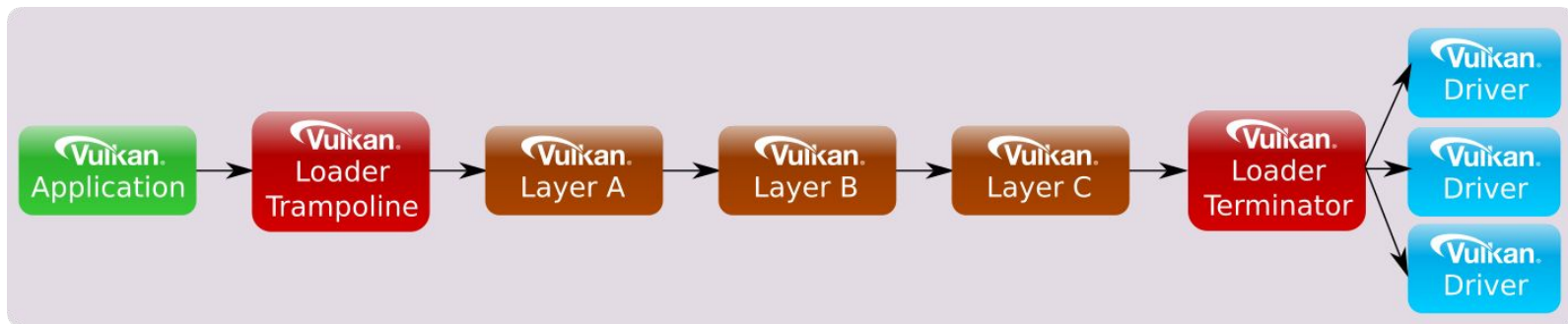
- Library responsible for finding and loading Drivers & Layers
- The “gatekeeper” to the Vulkan ecosystem
  - When the loader breaks, everything breaks
- Manages VkInstance’s and loading function pointers

# Why have the Vulkan-Loader at all?

- Simplifies the task of finding Vulkan drivers on a system
- Allows multiple Vulkan drivers to exist on user systems without interference
- Supports “Vulkan Layers” - third party plugin code
- Coordinates API call chains between applications, layers, & drivers

# The Vulkan API call chain

- API calls work their way through the loader, layers, and drivers in order
- Call chain goes in reverse order after going all the way 'down' the chain



# But what is a Layer?

- “Plugin interface to the Vulkan API”
- Able to intercept some or even all Vulkan API calls made by applications
- Allows many third parties to seamlessly work together
- Installed system wide, enabled based on the type of layer
- Available types are Explicit and Implicit



# Layers Types

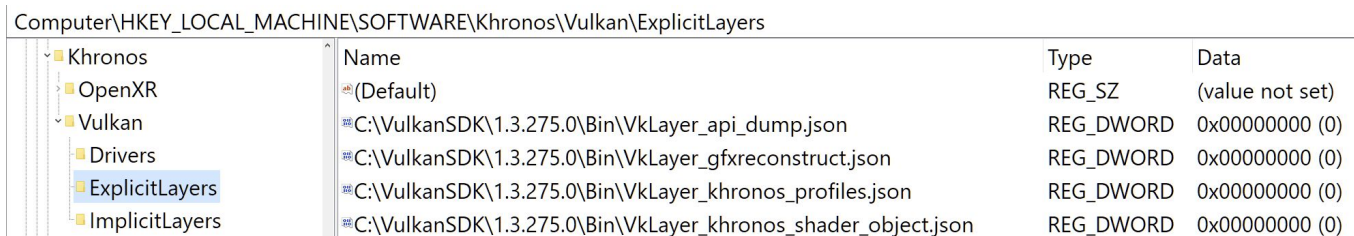
- **Explicit**
  - Active when enabled by application, environment variables, or Vulkan Configurator
  - For Validation, API capture, Extension Implementation, etc
- **Implicit**
  - Always active
  - Overlays, Post processing, Screen capture, Physical Device reordering
  - Implicit Layers can filter of extensions - helps capture layers behave correctly

# Manifest Files

- A JSON file describing a driver or layer
- Loader looks for manifests to find layers and drivers
- Contains a “library\_path” that points to the binary to load
  - Relative path eg. “../libVkLayer\_khronos\_validation.so”
  - Absolute path eg. “C:/VulkanSDK/1.3.275/Bin/VkLayer\_khronos\_validation.dll”
  - Naked path eg. “libVkLayer\_khronos\_validation.dylib”
    - Uses the dynamic linkers default search mechanism to find the library
- Name is taken from the “name” field, not the filename

# Manifest Discovery - Windows Registry

- HKEY\_LOCAL\_MACHINE\SOFTWARE\Khronos\Vulkan\<manifest\_type>
- HKEY\_CURRENT\_USER\SOFTWARE\Khronos\Vulkan\<manifest\_type>
- HKEY\_LOCAL\_MACHINE\WOW6432Node\SOFTWARE\Khronos\Vulkan\<manifest\_type>
- HKEY\_CURRENT\_USER\WOW6432Node\SOFTWARE\Khronos\Vulkan\<manifest\_type>
- Plus special paths found through Windows API driver discovery



Name	Type	Data
(Default)	REG_SZ	(value not set)
C:\VulkanSDK\1.3.275.0\Bin\VkLayer_api_dump.json	REG_DWORD	0x00000000 (0)
C:\VulkanSDK\1.3.275.0\Bin\VkLayer_gfxreconstruct.json	REG_DWORD	0x00000000 (0)
C:\VulkanSDK\1.3.275.0\Bin\VkLayer_khronos_profiles.json	REG_DWORD	0x00000000 (0)
C:\VulkanSDK\1.3.275.0\Bin\VkLayer_khronos_shader_object.json	REG_DWORD	0x00000000 (0)

# Manifest Discovery - Linux

- Scan folders based on contents of environment variables
  - XDG\_CONFIG\_HOME
  - XDG\_CONFIG\_DIRS
  - SYSCONFDIR
  - EXTRASYSCONFDIR
  - XDG\_DATA\_HOME
  - XDG\_DATA\_DIRS
- Example search paths:
  - /home/me/.config/vulkan/<manifest\_type>
  - /etc/xdg/vulkan/<manifest\_type>
  - /usr/local/etc/vulkan/<manifest\_type>
  - /etc/vulkan/<manifest\_type>

# Manifest Discovery - MacOS

- Follows Linux pattern of searching folders
- Uses fallbacks instead of XDG environment variables
  - Searches */etc & /usr/local/share/:/usr/share/*
- Additionally searches inside of App Bundles
  - `<bundle>/Contents/Resources/<manifest_type>`
  - This is the preferred location for shipping Vulkan-Loader & Layers with an application
  - The only way for layers to be found on iOS

# Manifest Discovery - Environment Variables

- VK\_LAYER\_PATH
- VK\_ADD\_LAYER\_PATH
- VK\_DRIVER\_FILES
  - Alias: VK\_ICD\_FILENAMES
- VK\_ADD\_DRIVER\_FILES
- Not used when application is running with elevated privileges
  - “Admin mode” on windows

# Guide to the loader log

# The Vulkan-Loader log:

- The main tool for debugging issues
- Writes to stderr
- What the log contains:
  - Where it searches for drivers and layers
  - What drivers and layers are found
  - Loading of said drivers and layers
  - Errors, warnings, and all relevant information



# Enabling the log: VK\_LOADER\_DEBUG

- From the command line
  - Windows Command Prompt: set VK\_LOADER\_DEBUG=all
  - Windows Powershell: \$env:VK\_LOADER\_DEBUG="all"
  - Linux/MacOS: export VK\_LOADER\_DEBUG=all
- Inside of an application
  - Add VkDebugUtilsMessengerCreateInfoEXT to pNext chain of VkInstanceCreateInfo
    - Only logs what is happening during vkCreateInstance
- From an IDE
  - Too many IDE's to list here - refer to your IDE of choice's user manual

# Granular logging control

- Can substitute “all” for more granular output control
  - error Report any errors encountered by the loader
  - warn Report any warnings encountered by the loader
  - info Report info-level messages generated by the loader
  - debug Report debug-level messages generated by the loader
  - layer Report all layer-specific messages generated by the loader
  - driver Report all driver-specific messages generated by the loader
- Use a comma delimited list to log multiple options together
  - set VK\_LOADER\_DEBUG=error,warn,info

# Logging loading of a Layer

```
LAYER:           Checking for Layer Manifest files in Registry at  
HKEY_LOCAL_MACHINE\SOFTWARE\Khronos\Vulkan\ImplicitLayers
```

```
INFO | LAYER:     Located json file "C:\Program Files\RenderDoc\renderdoc.json" from  
registry "HKEY_LOCAL_MACHINE\SOFTWARE\Khronos\Vulkan\ImplicitLayers"
```

```
INFO:           Layer "VK_LAYER_RENDERDOC_Capture" using deprecated  
'vkGetInstanceProcAddr' tag which was deprecated starting with JSON file version  
1.1.0. The new vkNegotiateLoaderLayerInterfaceVersion function is preferred, though  
for compatibility reasons it may be desirable to continue using the deprecated tag.
```

```
DEBUG | LAYER:    Loading layer library C:\Program Files\RenderDoc\.\renderdoc.dll
```

```
INFO | LAYER:     Insert instance layer "VK_LAYER_RENDERDOC_Capture" (C:\Program  
Files\RenderDoc\.\renderdoc.dll)
```

# Example vkCreateInstance Layer callstack

```
LAYER:          vkCreateInstance layer callstack setup to:
LAYER:          <Application>
LAYER:          ||
LAYER:          <Loader>
LAYER:          ||
LAYER:          VK_LAYER_OBS_HOOK
LAYER:          Type: Implicit
LAYER:          Disable Env Var:  DISABLE_VULKAN_OBS_CAPTURE
LAYER:          Manifest: C:\ProgramData\obs-studio-hook\obs-vulkan64.json
LAYER:          Library:
C:\ProgramData\obs-studio-hook\.\graphics-hook64.dll
LAYER:          ||
LAYER:          VK_LAYER_KHRONOS_validation
LAYER:          Type: Explicit
LAYER:          Manifest:
C:\VulkanSDK\1.3.275.0\Bin\VkLayer_khronos_validation.json
LAYER:          Library:
C:\VulkanSDK\1.3.275.0\Bin\.\VkLayer_khronos_validation.dll
LAYER:          ||
LAYER:          <Drivers>
```

# Example unloading at shutdown

```
DEBUG | LAYER:    Unloading layer library  
C:\WINDOWS\System32\DriverStore\FileRepository\u0384125.inf_amd64_7509a20b0a  
165522\B384149\.\amdvk64.dll
```

```
DEBUG | LAYER:    Unloading layer library  
C:\ProgramData\obs-studio-hook\.\graphics-hook64.dll
```

```
DEBUG | LAYER:    Unloading layer library  
C:\VulkanSDK\1.3.275.0\Bin\.\VkLayer_khronos_validation.dll
```

# Issues loading the Vulkan-Loader

# Vulkan-Loader always comes first

- If the Vulkan-Loader is missing, no Vulkan application can launch
- Different Operating Systems call it different things
  - Windows: vulkan-1.dll
  - Linux: libvulkan.so
  - MacOS: libvulkan.dylib
- Applications can “link” or “load” the Vulkan-Loader
  - Link - request that the dynamic linker find Vulkan-Loader at startup
  - Load - Application will manually call LoadLibrary/dlopen to open the Vulkan-Loader

# Why would Vulkan-Loader be missing?

- Vulkan capable drivers haven't been installed
- Linux: Required package is missing
- Windows: Vulkan drivers install vulkan-1.dll for you
- MacOS/iOS: Up to developer to make vulkan available
- Unlikely: On hardware that doesn't support Vulkan
  - Vulkan is almost 8 years old now
  - All desktop hardware released in 2015 supported Vulkan!



# What if the Vulkan-Loader is not found?

- It is not your responsibility to provide a Vulkan-Loader
  - If it is missing, that likely means no Vulkan drivers are installed
- It is recommended to NOT include a Vulkan-Loader in your application
  - Doesn't help when drivers do not exist
- It is recommended to load the Vulkan-Loader
  - Allows the application to startup and provide useful error reporting
  - Use the Volk library to handle it for you
    - <https://github.com/zeux/volk>

# Debugging Layer issues

# Diagnosing Layers not working

- Multiple ways to enable layers
  - Environment variables
    - VK\_INSTANCE\_LAYERS
    - VK\_LOADER\_LAYERS\_ENABLE
  - Vulkan Configurator
  - In-application API (VkInstanceCreateInfo::ppEnabledLayerNames)
- Multiple reasons
  - Layer manifest couldn't be found
  - Layer library couldn't be loaded
  - Error during vkNegotiateLoaderLayerInterfaceVersion

# Layer manifest can't be found

- Loader was unable to locate the 'manifest JSON' file
  - The layer was not in the paths the loader searches
- For environment variables:
  - Verify `VK_LAYER_PATH` or `VK_ADD_LAYER_PATH` is correct
  - Make sure app isn't running with elevated privileges
- Windows: Make sure manifest path is in appropriate registry
- Linux/MacOS: Make sure manifest path is in correct folder

# Layer library couldn't be loaded

- Layer binary failed to be loaded
- Loader will log the error message from the Dynamic Linker
- Check that the library file is actually present
  - Loader doesn't check for library file until right before loading it
  - May need to set `LD_LIBRARY_PATH` (linux) or `DYLD_LIBRARY_PATH` (macOS)
- Check the architecture (32bit vs 64bit)
- Check the linker dependencies of the library
- Check code signing

# Other library loading issues

- Layer was disabled by environment variable
  - `VK_LOADER_LAYERS_DISABLE`
- Failure during `vkCreateInstance`
  - May be trying to run a layer when it doesn't want to, eg Renderdoc & incompatible extensions
  - Layer may also not be written correctly

# Dealing with Implicit Layers

# Implicit Layers

- “Always enabled”
  - Except if it requires that a env-var is set with the value specified by “enable\_environment”
- Intended for “always on” use cases,
  - VkPhysicalDevice selection, Overlays, Screen capture
- Solve a real problem - allowing interception of API calls
- But is common source of friction
  - Known to change behavior, create validation errors, or crash applications



# Disabling a specific Implicit Layer

- Set an environment variable with the same name as “disable\_environment”
  - Value of the environment variable doesn't matter, so setting it to “1” will work
- This is the best way to disable badly acting implicit layers
  - Granular, supported in all versions of the Vulkan-Loader

- Example from manifest file

```
"disable_environment": {  
    "DISABLE_LAYER_OVERLAY_1": ""  
}
```

# But how to find the `disable_environment`?

- Helpfully printed in the log of any loaded implicit layer

```
LAYER:          VK_LAYER_MISBEHAVING_layer
LAYER:          Type: Implicit
LAYER:          Disable Env Var:  DISABLE_THIS_LAYER
LAYER:          Manifest: /home/me/layer/path
LAYER:          Library: /home/me/layer/library.so
```

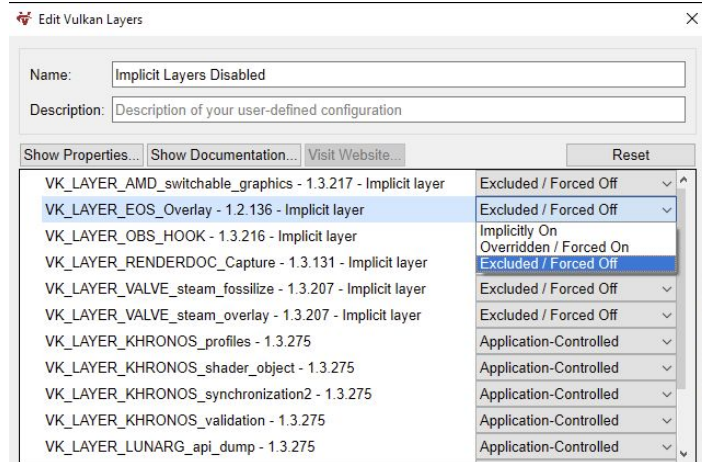
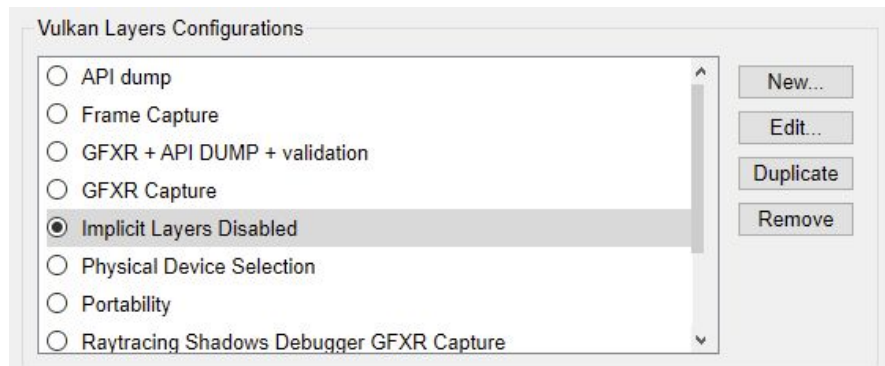
- Just set the environment variable to any value (eg, “1”) before calling `vkCreateInstance`

# Alternate way to disable Layers

- Another environment variable!
  - `VK_LOADER_LAYERS_DISABLE=<Layer_name>,<other_layer>`
- Supports wildcards with `*`
  - `VK_LOADER_LAYERS_DISABLE=*validation*`
- Can disable all layers with `~all~`
- Can disable all explicit or implicit layers with `~explicit~` and `~implicit~`
- Only supported with loader version 1.3.234 and above

# Using Vulkan Configurator to disable Layers

- Create a new configuration
- Set each Implicit Layer to “Excluded / Forced Off”
- Select this configuration in the main panel



# Debugging Driver Issues

# Driver issues

- Driver searching & loading has similar issues to layers
  - Invalid paths, incompatible libraries, incorrectly written driver implementations
- Environment Variables
  - `VK_DRIVER_FILES`, `VK_ADD_DRIVER_FILES`, `VK_ICD_FILENAMES` ignored when running with elevated privileges

# VK\_ERROR\_INCOMPATIBLE\_DRIVER

- Indicates that the loader could not find any compatible Vulkan drivers
- Most likely means that drivers were improperly installed
- Could be from implicit layers
- Solving this depends on the platform & drivers in question

# Physical Device reordering

- Layer's who try to put the 'right' VkPhysicalDevice first in the list
- Many of these layers
  - VK\_LAYER\_AMD\_switchable\_graphics
  - VK\_LAYER\_MESA\_device\_select
  - VK\_LAYER\_NV\_optimus
  - VK\_LAYER\_MCOF\_device\_filter
- May be causing Vulkan applications to use VkPhysicalDevices that aren't intended
- Solution: Figure out how to configure the layer or disable the layer entirely



# Application issues

- Sometimes it's the application that is at fault
- Forgetting to enable extensions & features
- Invalid Vulkan API calls - use the Validation layer to find & fix
- Forgetting to enable `VK_KHR_portability_enumeration` on macOS & iOS

# Recap

# Recap

- Use the log!
- Check every assumption
  - Make sure every path, file, & name is absolutely correct
- Try disabling implicit layers - may be the source of issues
- Loader is complex - but not impossible to understand

# Contact me!

- While I hope these slides are thorough, they aren't complete
- Email: [charles@lunarg.com](mailto:charles@lunarg.com)
- Vulkan Discord: CharlesG - LunarG



Help Us Improve the  
Vulkan SDK and Ecosystem

Share Your Feedback

**Take the LunarG annual developer's survey**

<https://www.surveymonkey.com/r/KTBZDCM>

- Survey results are tabulated
- Shared with the Vulkan Working Group
- Actions are assigned
- Results are reported

**Survey closes February 26, 2024**



Today's  
Presentation:

<http://bit.ly/3RUe4eg>



Get A FREE Tumbler  
at the LunarG Sponsor Table!



Thank you!

**QUESTIONS?**