Vulkan Loader Deep Dive
Mark Young and Lenny Komow
WELCOME!
Who We Are..

• **LunarG:**
  - Software engineering firm focusing on Graphics and Shader technologies
  - Valve provides financing to support Vulkan Ecosystem:
    - Vulkan desktop loader
    - Validation layers
    - Tools (Trace/Replay, VIA)

• **Mark Young:**
  - Current Vulkan desktop loader owner
  - OpenGL User-Mode drivers 9.5 years (ATI/AMD)
  - DirectX Kernel-Mode drivers 5 years (Qualcomm)

• **Lenny Komow**
  - Desktop Loader support (working towards co-owner)
  - 2D/3D ECAD rendering 1.5 years (Cadence Design)
Ask Yourself, “What Do I Want to Learn?”

“You’ve got to be very careful if you don’t know where you are going, because you might not get there.”

-Yogi Berra
What This Is Not...

- An introduction to Vulkan
- A Vulkan tutorial on writing:
  - Applications
  - Layers
  - Drivers
- An overview of using Validation Layers

Look for that and more on the Khronos Vulkan Resources page.
We **Will Be Covering**...

- A detailed look at the Vulkan Loader and its interfaces to:
  - Applications
  - Drivers
  - Layers

- Debug settings

- Possible optimizations
Vulkan App Developer View
Loader Position (High-Level)
Really Loader(s) Plural

- Intent is only one loader to rule them all, but reality is
  - Two different loaders:
    - Desktop Loader
      - Same source used for Linux/Windows
    - Android Loader
      - Nougat+ devices
  - Still, one loader interface design (in GitHub and LunarG Vulkan SDK)
Loader Interface Doc

• Available in [Vulkan-LoaderAndValidationLayers Github](https://github.com/lunarVR/Vulkan-LoaderAndValidationLayers/blob/master/loader/LoaderAndLayerInterface.md)

• Valid for all Vulkan loaders

---

**Architecture of the Vulkan Loader Interfaces**

**Table of Contents**

- Overview
  - Who Should Read This Document
  - The Loader
  - Layers
  - Installable Client Drivers
  - Instance Versus Device
  - Dispatch Tables and Call Chains

- Application Interface to the Loader
  - Interfacing with Vulkan Functions
  - Application Layer Usage
  - Application Usage of Extensions

- Loader and Layer Interface
  - Layer Discovery
## Vulkan Desktop Loader Versus OpenGL Loader

<table>
<thead>
<tr>
<th></th>
<th>OpenGL Loader</th>
<th>Vulkan Loader</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Controlled By</strong></td>
<td>Operating/Windowing System</td>
<td>Khronos</td>
</tr>
<tr>
<td><strong>Updated</strong></td>
<td>Infrequently</td>
<td>4-6 Weeks</td>
</tr>
<tr>
<td><strong>Available Source</strong></td>
<td>In some cases</td>
<td>Open Source (Github)</td>
</tr>
<tr>
<td><strong>Who Can Contribute</strong></td>
<td>Limited set of developers</td>
<td>Anyone*</td>
</tr>
</tbody>
</table>

* CLA required for contributions
# Github

![Github Logo](image)

## KhranosGroup / Vulkan-LoaderAndValidationLayers

This repository contains Vulkan loader and validation layers. It has 7,415 commits, 227 branches, 39 releases, and 100 contributors. The license is Apache-2.0.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Description</th>
<th>Commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>build-android</td>
<td>Vulkan: update to header 1.0.42</td>
<td>2 days ago</td>
</tr>
<tr>
<td>cmake</td>
<td>Cmake: Add files needed to build with Mir</td>
<td>3 months ago</td>
</tr>
<tr>
<td>common</td>
<td>repo: Clang-format LLVM source files using Google</td>
<td>a month ago</td>
</tr>
<tr>
<td>demos</td>
<td>demos: Use device local heap for depth</td>
<td>9 days ago</td>
</tr>
<tr>
<td>external_revisions</td>
<td>externals: Update GitHub master commit refs</td>
<td>3 days ago</td>
</tr>
<tr>
<td>include/vulkan</td>
<td>Vulkan: update to header 1.0.42</td>
<td>2 days ago</td>
</tr>
<tr>
<td>layers</td>
<td>layers: Remove redundant validation</td>
<td>2 days ago</td>
</tr>
<tr>
<td>libs</td>
<td>build: Cleanup warnings when compiling with clang</td>
<td>2 months ago</td>
</tr>
<tr>
<td>loader</td>
<td>docs: Add new layer registries to loader doc</td>
<td>22 hours ago</td>
</tr>
<tr>
<td>scripts</td>
<td>scripts: Update loader automation</td>
<td>2 days ago</td>
</tr>
<tr>
<td>tests</td>
<td>tests: Add tests for allocation</td>
<td>19 hours ago</td>
</tr>
<tr>
<td>windowsRuntimeInstaller</td>
<td>windows: Build runtime from ReWithDebinfo</td>
<td>21 days ago</td>
</tr>
<tr>
<td>appveyor.yml</td>
<td>build: Change appveyor to build VS 2013 and 2015</td>
<td>9 days ago</td>
</tr>
<tr>
<td>clang-format</td>
<td>repo: Change source style from LLVM to Google</td>
<td>a month ago</td>
</tr>
</tbody>
</table>

Latest commit 346423820 29 days ago
How Does the Loader Get on a User’s System?

- **Android:**
  - Installed as part of Android OS with Nougat (or newer)

- **Desktop:**
  - Driver Packages
  - Application Installs
  - LunarG SDKs
  - Manually built/installed (GitHub)

You may have multiple copies, but that’s ok. Really!
Example of Multiple Installs

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Publisher</th>
<th>Installed On</th>
<th>Size (MB)</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Visual C++ 2013 Redistributable (x64) - 12...</td>
<td>Microsoft Corporation</td>
<td>9/2/2016</td>
<td>20.5</td>
<td>12.0.30591.0</td>
</tr>
<tr>
<td>Microsoft Visual C++ 2013 Redistributable (x64) - 12...</td>
<td>Microsoft Corporation</td>
<td>9/2/2016</td>
<td>17.1</td>
<td>12.0.30299.1</td>
</tr>
<tr>
<td>Microsoft Visual C++ 2013 Redistributable (x64) - 12...</td>
<td>Microsoft Corporation</td>
<td>9/2/2016</td>
<td>17.1</td>
<td>12.0.30591.0</td>
</tr>
<tr>
<td>Microsoft Visual C++ 2013 Redistributable (x64) - 14...</td>
<td>Microsoft Corporation</td>
<td>9/2/2016</td>
<td>23.5</td>
<td>14.0.24215.1</td>
</tr>
<tr>
<td>Microsoft Visual Studio Community 2013 with Update 4</td>
<td>Microsoft Corporation</td>
<td>12/19/2016</td>
<td>797</td>
<td>12.0.31101</td>
</tr>
<tr>
<td>Microsoft Windows Internet Explorer</td>
<td>Microsoft Corporation</td>
<td>12/19/2016</td>
<td>287</td>
<td>14.0.2397.178</td>
</tr>
<tr>
<td>NVIDIA GeForce Experience</td>
<td>NVIDIA Corporation</td>
<td>2/22/2017</td>
<td>378</td>
<td>378.66</td>
</tr>
<tr>
<td>NVIDIA Graphics Driver 378.66</td>
<td>NVIDIA Corporation</td>
<td>2/22/2017</td>
<td>378</td>
<td>378.66</td>
</tr>
<tr>
<td>NVIDIA PhysX System Software 9.16.0318</td>
<td>NVIDIA Corporation</td>
<td>9/9/2016</td>
<td>409.5</td>
<td>5.10.0318</td>
</tr>
<tr>
<td>OpenVPN 2.4.0-401</td>
<td>OpenVPN Technologies, Inc.</td>
<td>11/26/2017</td>
<td>6.35</td>
<td>2.4.0.401</td>
</tr>
<tr>
<td>Python 2.7.12</td>
<td>Python Software Foundation</td>
<td>9/30/2016</td>
<td>117.8</td>
<td>2.7.12.105</td>
</tr>
<tr>
<td>Python 3.5.2 (32-bit)</td>
<td>Python Software Foundation</td>
<td>9/2/2016</td>
<td>84.2</td>
<td>3.5.2.150</td>
</tr>
<tr>
<td>Python 3.5.2 (64-bit)</td>
<td>Python Software Foundation</td>
<td>9/2/2016</td>
<td>448.9</td>
<td>3.5.2.121</td>
</tr>
<tr>
<td>Qt</td>
<td>The Qi Company Ltd</td>
<td>2/2/2017</td>
<td>6.20.9</td>
<td>2.0.0</td>
</tr>
<tr>
<td>Realtek Card Reader</td>
<td>Realtek Semiconductor Corp.</td>
<td>9/1/2016</td>
<td>114.6</td>
<td>6.0.1.7.628</td>
</tr>
<tr>
<td>Realtek High Definition Audio Driver</td>
<td>Realtek Semiconductor Corp.</td>
<td>10/4/2016</td>
<td>40.9</td>
<td>6.0.1.7.628</td>
</tr>
<tr>
<td>Blender</td>
<td>Baldr Karlsson</td>
<td>2/27/2017</td>
<td>46.4</td>
<td>0.33</td>
</tr>
<tr>
<td>Synfig</td>
<td>Synfig Ltd</td>
<td>1/28/2017</td>
<td>53.6</td>
<td>1.87</td>
</tr>
<tr>
<td>VAPID-Window 9.21.2</td>
<td>VAPID-Window 9.21.2</td>
<td>1/4/2017</td>
<td>9.21</td>
<td>2.4</td>
</tr>
<tr>
<td>VCL media player</td>
<td>VCL media player</td>
<td>9/8/2016</td>
<td>170.8</td>
<td>1.23.4</td>
</tr>
<tr>
<td>LunarG Inc.</td>
<td>LunarG Inc.</td>
<td>2/22/2017</td>
<td>1.66</td>
<td>1.0.39.1</td>
</tr>
<tr>
<td>LunarG Inc.</td>
<td>LunarG Inc.</td>
<td>3/3/2017</td>
<td>1.66</td>
<td>1.0.42.0</td>
</tr>
<tr>
<td>Microsoft Windows Driver Kit - Windows 10 14393.0</td>
<td>Microsoft Corporation</td>
<td>12/19/2016</td>
<td>21.96</td>
<td>10.0.14393.0</td>
</tr>
<tr>
<td>Microsoft Windows Driver Kit - Windows 10 14393.0</td>
<td>Microsoft Corporation</td>
<td>9/2/2016</td>
<td>754.4</td>
<td>12.0.31014.0</td>
</tr>
<tr>
<td>Microsoft Visual Studio 2015 Update 4</td>
<td>Microsoft Corporation</td>
<td>12/19/2016</td>
<td>287.4</td>
<td>14.0.2397.178</td>
</tr>
<tr>
<td>Microsoft Visual Studio Community 2015</td>
<td>Microsoft Corporation</td>
<td>12/19/2016</td>
<td>287.4</td>
<td>14.0.2397.178</td>
</tr>
<tr>
<td>Microsoft Visual Studio Community 2015</td>
<td>Microsoft Corporation</td>
<td>12/19/2016</td>
<td>287.4</td>
<td>14.0.2397.178</td>
</tr>
<tr>
<td>Windows Phone 8.1 Emulator - EMUI</td>
<td>Microsoft Corporation</td>
<td>9/2/2016</td>
<td>754.4</td>
<td>12.0.31014.0</td>
</tr>
<tr>
<td>Wix Toolkit v10.3.3.007</td>
<td>Microsoft Corporation</td>
<td>12/19/2016</td>
<td>170.8</td>
<td>3.10.3.3007</td>
</tr>
</tbody>
</table>
```

Currently installed programs: Total size: 13.6 GB
77 programs installed
Which File to Use?

- Possibly multiple files on your machine

- Original file contains full version (major, minor, patch)
  - libvulkan.so.1.0.39.1
  - libvulkan.so.1.0.42.0
  - vulkan-1-1-0-39-1.dll
  - vulkan-1-1-0-42-0.dll

- Apps should target the base name shared object/DLL
  - libvulkan.so.1 symbolic link to latest vulkan-1.dll copy of latest

- The 1 is the ABI version
  - Indicates backwards compatibility
Your Ticket to the Fast Lane!

The loader is designed to be:
- Optimized
- Efficient
- Ignorant

It sets you up and steps out of your way as much as possible.

*Where there is great power there is great responsibility*
- Winston Churchill
With Some Caution

- Loader will crash if you use it improperly

- Examples:
  - Attempting to use extensions that aren’t enabled
  - Use Device procs queried from a different device
    - vkGetDeviceProcAddr for vkQueueSubmit on Physical Device A (Nvidia)
    - Call it with Physical Device B (Intel)
  - Passing in a wrong handle to a call
    - Using a VkInstance in place of a VkDevice
Expensive Loader Calls

- Most expensive loader calls are:
  - vkCreateInstance
  - vkCreateDevice
  - vkEnumerateXXX

- Loader has to generate state on these (but calls are supposed to be stateless)

- Some overhead can include loader re-gen/re-alloc data every call

- Why?
  - External GPUs
  - Layer info changing
  - System state change
    - Switching to battery
Loader Debugging

Desktop Loader Debug [Environment Variable]:

`VK_LOADER_DEBUG` warn, error, info, perf, debug, <or> all

Enable Validation Layers
Heading Deeper

• Dispatchable Objects
• Instance versus Device Objects/Commands
• Trampolines and Terminators
• Manifest Files (Desktop)
• What’s Exposed by Default?
• Extension Handling
• Loader/ICD Interaction
• Loader/Layer Interface
• Loader Security
Dispatchable Objects

• Most commands take an opaque dispatchable object as first parameter

```
VkResult vkGetEventStatus(VkDevice device, VkEvent event);
```

• First field in each dispatchable object is a dispatch table pointer
  – Used by loader trampoline code
  – ICDs must reserve first element of created objects for a pointer the loader will fill in
Instance Versus Device Objects/Commands

**Instance:** High-level construct (similar, but not the same, as a GL Context)
Works with all ICDs and Physical Devices
Includes: \texttt{VkInstance} and \texttt{VkPhysicalDevice}

**Device:** Logical accessor to a particular Physical Device (through a particular ICD)
Includes: \texttt{VkDevice}, \texttt{VkQueue}, \texttt{VkCmdBuffer} and other objects derived from these
Trampolines and Terminators
Trampolines and Terminators

Trampoline
Loader entrypoint for a command
Triggers the proper call-chain start
Start could be a layer, the terminator, or an ICD

Terminator
Loader code that distributes calls to multiple ICDs
All Instance and certain Device commands
Call Chains

Sequence of function calls from the application to its final destination

May include stepping into:
- Loader (one or more times)
- Any enabled layers
- One or more ICDs

Different for Instance versus Device commands
- Instance chain generated during \texttt{vkCreateInstance}
- Device chain generated during \texttt{vkCreateDevice}
Example Call Chains

Instance Call Chain

Device Call Chain using loader exports *

Device Call Chain using vkGetDeviceProcAddr *

* Some special cases still require a specific device call chain to include a trampoline/terminator
Manifest Files

Based on Manifest lists used by ships

Listed all Crew, Passengers, Cargo
Manifest File Usage

• Only used by Desktop Loader
  – Primary purpose to reduce security risk

• JSON file format, read on layer/ICD info query
  – vkEnumerateInstanceLayerProperties
  – vkEnumerateInstanceExtensionProperties

• Lists all important properties of interest to the app/loader

• Loader interaction with actual library delayed
  – ICD interaction not initiated until Manifest file appears valid
  – Layer interaction not initiated until enabled during vkCreateInstance
What’s Exposed By Default?

Loader exports following symbols by default:

• Core Vulkan (currently 1.0)
• WSI for specific platform
  – All Platforms:
    • VK_KHR_surface
    • VK_KHR_swapchain
    • VK_KHR_display
    • VK_KHR_display_swapchain
  – Windows Additional:
    • VK_KHR_win32_surface
  – Linux Additional:
    • VK_KHR_xlib_surface
    • VK_KHR_xcb_surface
    • VK_KHR_wayland_surface
    • VK_KHR_mir_surface
  – Android Additional:
    • VK_KHR_android_surface
Extension Handling

Behavior Dependent on Loader Knowledge

Known extension, “No”-n problems

  Ok, sorry about the bad pun!

Unknown Device Extensions:
  • Loader handles automatically

Unknown Instance Extensions:
  • Loader must know in order to handle properly
Why Unknown Instance Extensions cause problems

Device Call Chain using vkGetDeviceProcAddr

Instance Call Chain

Can handle unknown parameter list easily with tail-call optimization

Can’t handle unknown parameter list because must process for individual ICDs
Instance Extension Name Filtering

Loader filters out Unknown Instance Extension names
- Won’t appear in `vkEnumerateInstanceExtensionProperties`
- Throw an error in `vkCreateInstance`

Disable Environment Variable:
`VK_LOADER_DISABLE_INST_EXT_FILTER`

**Beware:** Can cause crashes if you attempt to use a command from a ‘forced-on’ instance extension!
Loader Generated Code

New Vulkan header/xml integrated into tree about once a week

All extension code automatically generated
   • Python scripts (under scripts folder)
   • Run every build

Loader script of interest → scripts/loader_extension_generator.py
Still Awake?

I love sleep. My life has the tendency to fall apart when I’m awake, you know?

Ernest Hemingway
Installable Client Drivers (ICDs)

• Provided by hardware company (or Open Source community)

• Vulkan Drivers have associated JSON Manifest file
  – Install should set this up properly

• Versioned interface with loader
  – Detailed in LoaderAndLayerInterface.md in Github
  – Also detailed in SDK as LoaderAndLayerInterface.html
Desktop ICD Manifest Search Locations

Windows Registry: [Simply points to location of JSON file]
   HKEY_LOCAL_MACHINE\SOFTWARE\Khronos\Vulkan\Drivers

   On 64-bit Windows, 32-bit ICD JSON found in:
      HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Khronos\Vulkan\Drivers

Linux: [Actual location]
   /usr/local/etc/vulkan/icd.d
   /usr/local/share/vulkan/icd.d
   /etc/vulkan/icd.d
   /usr/share/vulkan/icd.d
   $HOME/.local/share/vulkan/icd.d
Example ICD Manifest File

For Intel’s Open Source Mesa Vulkan Driver

```
{
  "file_format_version": "1.0.0",
  "ICD": {
    "library_path": "/usr/lib64/libvulkan_intel.so",
    "api_version": "1.0.3"
  }
}
```

- Manifest JSON file format
- Location of library (dll/so)
- Version of Vulkan API written against
Disabling Windows ICDs in Registry

Simply write 1 to the ICD 32-bit Key Value

Loader will now ignore it
Desktop ICD Debug Environment Variable

Force a particular ICD or set of ICDs:

```
export VK_ICD_FILENAMES=/home/.../intel_icd/i915.json:/home/.../nvidia.json
```
Loader/ICD Interface

ICDs should expose at least three commands:

- `vk_icdNegotiateLoaderICDInterface`
- `vk_icdGetInstanceProcAddr`
- `vk_icdGetPhysicalDeviceProcAddr`

Names can actually be different (overridden in Manifest file)

Loader looks for Manifest files during certain calls:

- `vkEnumerateInstanceExtensionProperties`
- `vkCreateInstance`

Loader then “negotiates” the interface version with the ICD
Loader/ICD Interface Version Negotiation

Loader calls ICD using

```c
VkResult vk_icdNegotiateLoaderICDInterfaceVersion(
    uint32_t *pVersion);
```

*pVersion initially contains loader's desired interface version

ICD then determines if it can run that version:

- If it can, it returns VK_SUCCESS
- If it can't, it determines if it can run at a lower version
  - If it can, it updates *pVersion with the new interface version and return VK_SUCCESS
  - If not, it returns VK_ERROR_INCOMPATIBLE_DRIVER

Now the loader queries all the appropriate commands using `vk_icdGetInstanceProcAddr`. 
More Details on Loader/ICD Interface

Documented in LoaderAndLayerInterface.md file
Under “Loader and ICD Interface Negotiation” section

Interface defined in vk_icd.h
In Github and in the installed SDK as include/vulkan/vk_icd.h
Layers

Layers generated by many different sources

Details we’ll cover:

- Instance and Device Layers
- Implicit versus Explicit Layers
- Layers manifest files
- Loader/Layer Interface
Instance and Device Layers

Both vkCreateInstance and vkCreateDevice have places to enable layers
vkCreateInstance in VkInstanceCreateInfo structure

typedef struct VkInstanceCreateInfo {
    VkStructureType sType;
    const void* pNext;
    ...
    uint32_t enabledLayerCount;
    const char* const* ppEnabledLayerNames;
    ...
} VkInstanceCreateInfo;

vkCreateDevice in VkDeviceCreateInfo structure

typedef struct VkDeviceCreateInfo {
    VkStructureType sType;
    const void* pNext;
    ...
    uint32_t enabledLayerCount;
    const char* const* ppEnabledLayerNames;
    ...
} VkDeviceCreateInfo;
Device Layers Deprecated!

Not used anymore

typedef struct VkDeviceCreateInfo {
    VkStructureType                    sType;
    const void*                        pNext;
    VkDeviceCreateFlags                flags;
    uint32_t                           queueCreateInfoCount;
    const VkDeviceQueueCreateInfo*     pQueueCreateInfos;
    uint32_t                           enabledLayerCount;
    const char* const*                 ppEnabledLayerNames;
    uint32_t                           enabledExtensionCount;
    const char* const*                 ppEnabledExtensionNames;
    const VkPhysicalDeviceFeatures*    pEnabledFeatures;
} VkDeviceCreateInfo;

All Layers enabled in Instance

Device Layers are ignored

Makes it simpler for you
Desktop Layer Loading

Implicit
- Enabled by default
- Disable with Environment Variable (Defined in JSON)
- Example:
  - Nvidia’s Optimus Layer
  - Valve’s Vulkan Steam Overlay Layer

Explicit
- Must be enabled by app or environment
- Example:
  - Validation Layers

Only “Explicit Layers” on Android
Example Explicit Layer Manifest File

VkTrace Layer Manifest File:

```json
{
    "file_format_version": "1.0.0",
    "layer": {
        "name": "VK_LAYER_LUNARG_vktrace",
        "type": "GLOBAL",
        "library_path": ".\VkLayer_vktrace_layer.dll",
        "api_version": "1.0.42",
        "implementation_version": "1",
        "description": "Vktrace tracing library",
        "functions": {
            "vkGetInstanceProcAddr": "__HOOKED_vkGetInstanceProcAddr",
            "vkGetDeviceProcAddr": "__HOOKED_vkGetDeviceProcAddr"
        }
    }
}
```

<table>
<thead>
<tr>
<th>Manifest JSON file format</th>
<th>Layer Name</th>
<th>Layer Type (only GLOBAL or INSTANCE)</th>
<th>Location of library (dll/so)</th>
<th>Version of Vulkan API written against</th>
<th>Layer implementation version</th>
<th>Overriding function names</th>
</tr>
</thead>
</table>

All Layer Data

---

**Layer Manifest File Format:**

- **file_format_version**: Version of the file format.
- **layer**: Details about the layer:
  - **name**: Name of the layer.
  - **type**: Type of the layer (GLOBAL or INSTANCE).
  - **library_path**: Path to the library file.
  - **api_version**: Version of the Vulkan API.
  - **implementation_version**: Implementation version of the layer.
  - **description**: Description of the layer.
  - **functions**: Overriding function names.

**Example:**

The example shows a VkTrace layer manifest file with the following details:

- **name**: VK_LAYER_LUNARG_vktrace
- **type**: GLOBAL
- **library_path**: .\VkLayer_vktrace_layer.dll
- **api_version**: 1.0.42
- **implementation_version**: 1
- **description**: Vktrace tracing library
- **functions**: Overriding function names for `vkGetInstanceProcAddr` and `vkGetDeviceProcAddr`
Example Implicit Layer Manifest File

Nvidia’s Optimus Layer Manifest file:

```json
{
    "file_format_version": "1.0.0",
    "layer": {
        "name": "VK_LAYER_NV_optimus",
        "type": "INSTANCE",
        "library_path": ".\nvogl64.dll",
        "api_version": "1.0.42",
        "implementation_version": "1",
        "description": "NVIDIA Optimus layer",
        "functions": {
            "vkGetInstanceProcAddr": "vk_optimusGetInstanceProcAddr",
            "vkGetDeviceProcAddr": "vk_optimusGetDeviceProcAddr"
        },
        "disable_environment": {
            "DISABLE_LAYER_NV_OPTIMUS_1": ""
        }
    }
}
```

Environment variable used to disable this implicit layer
Layer Manifest File Search Locations

- **Windows Registry:** [Simply points to location of JSON file]
  HKEY_LOCAL_MACHINE\SOFTWARE\Khronos\Vulkan\ { ImplicitLayers \ ExplicitLayers }
  HKEY_CURRENT_USER\SOFTWARE\Khronos\Vulkan\ { ImplicitLayers \ ExplicitLayers }

  - On 64-bit Windows, 32-bit Layer JSON locations found in:
    HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Khronos\Vulkan\ { ImplicitLayers \ ExplicitLayers }
    HKEY_CURRENT_USER\SOFTWARE\WOW6432Node\Khronos\Vulkan\ { ImplicitLayers \ ExplicitLayers }

- **Linux:** [Actual location]
  /usr/local/etc/vulkan/ {explicit_layer.d / implicit_layer.d }
  /usr/local/share/vulkan/ {explicit_layer.d / implicit_layer.d }
  /etc/vulkan/ {explicit_layer.d / implicit_layer.d }
  /usr/share/vulkan/ {explicit_layer.d / implicit_layer.d }
  $HOME/.local/share/vulkan/ {explicit_layer.d / implicit_layer.d }
Desktop Layer Debug Environment Variables

Force on a Layer from outside the application:

| VK_INSTANCE_LAYERS | Delimited list of layer names to enable. |

For example:

```
set VK_INSTANCE_LAYERS=VK_LAYER_LUNARG_vktrace;VK_LAYER_LUNARG_standard_validation
```

Force/Override the Layer path:

| VK_LAYER_PATH      | Delimited list of paths to search for layer JSON files. |

For example:

```
set VK_LAYER_PATH=C:\VulkanSDK\1.0.42.1.alpha.102\Bin
```
Overall Desktop Layer Order

Layers are added in this order

NOTE: Duplicates removed (first occurrence added)
Loader/Layer Interface

• Just like ICDs, Layers negotiate an interface version with the loader

• Loader/Layer interface an improvement on Loader/ICD interface
  – Reduces back and forth
  – Allows future growth

• Spec details in LoaderAndLayerInterface.md file
  – Under “Layer Version Negotiation” section

• Header contents found in include/vulkan/vk_layer.h
  – Found in Github and SDK
Layer defines and exports new `vkNegotiateLoaderLayerInterfaceVersion` function

- NOTE: Name can be overridden in Manifest file

```c
VkResult vkNegotiateLoaderLayerInterfaceVersion(
    VkNegotiateLayerInterface *pVersionStruct);
```

- Uses new negotiation structure

```c
typedef struct VkNegotiateLayerInterface {
    VkNegotiateLayerStructType sType;
    void *pNext;
    uint32_t loaderLayerInterfaceVersion;
    PFN_vkGetInstanceProcAddr pfnGetInstanceProcAddr;
    PFN_vkGetDeviceProcAddr pfnGetDeviceProcAddr;
} VkNegotiateLayerInterface;
```
Loader/Layer Negotiation Process

Loader creates and initializes VkNegotiateLayerInterface struct

```c
sType = LAYER_NEGOTIATE_INTERFACE_STRUCT
pNext = NULL;    // For future growth
loaderLayerInterfaceVersion = Loader’s desired interface version
```

Layer determines if it can support that version:
- If it can, it fills in function pointers and returns VK_SUCCESS
- If it can’t, it determines if it can run at a lower version
  - If it can, it:
    - Updates loaderLayerInterfaceVersion with the new interface version
    - Fills in function pointers appropriate to that version (may be none)
    - Return VK_SUCCESS
  - If not, it:
    - Returns VK_ERROR_INITIALIZATION_FAILED

If layer doesn’t fill in function pointers, loader has to call via OS GetProcAddress call
Security!

Loader is just a user-mode DLL (i.e. No escalation)

Desktop loader uses Manifest files to access ICD/layer info prior to loading

Memory Validation:
• Valgrind verification
• Conformance testing validation
Scanning

Runtime and SDK -> VirusTotal

- Checks 50+ virus scanners
  - Avast
  - AVG
  - BitDefender
  - Kapersky
  - Malwarebytes
  - McAfee
  - Microsoft
  - Symantec

- Sometimes 1 or 2 false positives
## Desktop Loader Environment Variable Review

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_LOADER_DEBUG</td>
<td>Enable debug output of the loader to stderr.</td>
<td>warn, error, info, perf, debug, all</td>
</tr>
<tr>
<td>VK_LOADER_DISABLE_INST_EXT_FILTER</td>
<td>Disable unknown instance extension name filtering</td>
<td>0 = Do nothing, Non-0 = Disable filtering</td>
</tr>
<tr>
<td>VK_ICD_FILENAMES</td>
<td>Delimited list of ICD JSON Manifest files indicating the list of driver files to use instead of the default.</td>
<td>Full path to JSON file: /home/.../icds/myicd1.json; /home/.../icds/myicd2.json</td>
</tr>
<tr>
<td>VK_INSTANCE_LAYERS</td>
<td>Add additional instance layers to load during the vkCreateInstance call. Duplicates are ignored.</td>
<td>Should be name of layer, not name of JSON file: VK_LAYER_GOOGLE_tracking</td>
</tr>
<tr>
<td>VK_LAYER_PATH</td>
<td>Delimited list of directories the loader should use when looking for layer JSON Manifest files</td>
<td>Full path to JSON directory: /home/.../mylayers1; /home/.../mylayers2</td>
</tr>
</tbody>
</table>
Please Contribute!

- Code
- Bugs
- Improvement suggestions
Thanks!

Questions
Useful Links

Khronos LoaderAndValidationLayers Github

https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers

Loader, ICD, and Layer Interface Specification

https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers/blob/master/loader/LoaderAndLayerInterface.md

Validation Layers Webinar

https://www.khronos.org/news/events/webinar-vulkan-validation-layers-deep-dive

LunarG Vulkan SDK (includes latest loader and layers)

https://vulkan.lunarg.com/

Khronos

https://www.khronos.org
Extra Slides
Vulkan Surfaces (VkSurfaceKHR)

• Started out being created/owned by loader
  – Would end at loader terminator and return

• Can be created by an ICD (if it chooses) by:
  – Supporting version 3 of ICD/loader interface
  – Exposing appropriate WSI functions for current platform

• Loader still intercepts all calls using VkSurfaceKHR (instance and device commands)
  – Has to convert from loader VkSurfaceKHR to ICD VkSurfaceKHR
Layer Discovery

- Layers discovered by loader during:
  - `vkEnumerateInstanceExtensionProperties`
  - `vkEnumerateInstanceLayerProperties`
  - `vkCreateInstance`

- Desktop loader uses Layer Manifest JSON files similar to ICDs:
  - Desktop Layers not actually loaded during:
    - `vkEnumerateInstanceExtensionProperties`
    - `vkEnumerateInstanceLayerProperties`
  - Only enabled layers loaded in:
    - `vkCreateInstance`