



## Explore the Vulkan Loader and Validation Layers



# Explore the Vulkan Loader and Validation Layers



**Jon Ashburn**  
Principal Engineer  
Vulkan Loader Architect



**Mark Lobodzinski**  
Senior Engineer  
Validation Layers



**Courtney Goeltzenleuchter**  
Principal Engineer  
Loader & Validation Layers Architect



**Tobin Ehlis**  
Engineer  
Validation Layers Lead Engineer




**Karl Schultz**  
Principal Engineer  
Validation Layers and LunarG SDK



**Rolando Caloca**  
Sr. Rendering Engineer  
Unreal Engine 4 port to Vulkan



# Where to find info...

- **LunarG.com (<http://www.lunarg.com>)**
  - BoF slides: <https://lunarg.com/lunarg-birds-feather-session-siggraph-july-26-2016/>
- **Khronos Loader and Validation Layers github**
  - github: <https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers>
  - Loader specification and architecture: <https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers/blob/master/loader/LoaderAndLayerInterface.md>
- **Khronos Vulkan API**
  - Vulkan landing page: <https://www.khronos.org/vulkan/>
  - Vulkan forum: <https://forums.khronos.org/forumdisplay.php/114-Vulkan-High-Efficiency-GPU-Graphics-and-Compute>
- **LunarG Vulkan SDK** 
  - Download SDK, report SDK issues, read documentation: <https://vulkan.lunarg.com>

# Agenda

- **Vulkan Loader**
- **Vulkan Validation Layers**
- **Epic Games: Vulkan on Unreal Engine 4 - Validation Layers**
- **Q&A**

# Vulkan Loader

## **Vulkan Loader - Goals**

- **Validation**
- **Plug-n-play experience**
- **Extensible**

# Vulkan Loader - Goals

- **Layers**
  - Vulkan has been designed to support plug-in layers
  - The loader is the consistent method for enabling layers
  - Same layers for Windows, Linux and Android
- **Plug-n-play experience**
- **Extensible**

- Robust API validation in layer(s), not drivers
- No perf impact when not used. That is, there is no test in the code to see if validation should be done or not. This also applies to more indirect performance impact, such as there is no validation code loaded into the apps process space.
- Same validation for all platforms (Windows, Linux, Android)

# Vulkan Loader - Goals

- **Layers**
  - Vulkan has been designed to support plug-in layers
  - The loader is the consistent method for enabling layers
  - Same layers for Windows, Linux and Android
- **Plug-n-play experience**
  - **Support multiple Vulkan devices on desktop**
- **Extensible**

- Multiple drivers can live on a system and an application can select which to use without changing the system configuration. Avoids issues seen today with trying to use different drivers / graphics cards on a Linux system.



# Vulkan Loader - Goals

- **Layers**
  - Vulkan has been designed to support plug-in layers
  - The loader is the consistent method for enabling layers
  - Same layers for Windows, Linux and Android
- **Plug-n-play experience**
  - Support multiple Vulkan devices on desktop
- **Extensible**
  - **Layers can extend/enhance the API**
    - **Performance profiling**
    - **Image capture for image-based regression testing**
    - **API Dump**
  - **Layers or drivers can support extensions**

- Layers provide mechanism to extend the API and/or provide features without impacting the application or driver. E.g.
- performance profiling
- image capture for image-based regression testing
- Tracing / API Dump (output Vulkan calls & parameters for debugging)
- Layers or drivers can support extensions

See LoaderAndLayerInterface.md in Github for details.

# Vulkan Loader - Windows & Linux

- **Open source**
  - **Developed by LunarG, owned by Khronos**
- **JSON manifest files**

- developed by LunarG, owned by Khronos
- Github <https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers>

# Vulkan Loader - Windows & Linux

- **Open source**
  - Developed by LunarG, owned by Khronos
- **JSON manifest files**
  - **Encode library details in json files**
  - **Security, lower system impact**

- developed by LunarG, owned by Khronos
- Github <https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers>

# Vulkan Loader - Windows & Linux

- **Open source**
  - Developed by LunarG, owned by Khronos
- **JSON manifest files**
  - Encode library details in json files
  - Security, lower system impact

- developed by LunarG, owned by Khronos
- Github <https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers>
- Dynamic trampoline code
  - For unknown device extensions. No JIT
  - Dispatch destination determined at CreateDevice
-

# Vulkan Loader - Windows & Linux

- **Search paths for driver and layer json manifest files**
  - **Linux:** /etc/vulkan/\*, /usr/share/vulkan/\*, \$HOME/.local/share/vulkan/\*
  - **Windows:** HKEY\_LOCAL\_MACHINE\SOFTWARE\Khronos\Vulkan\\*
  - **Environment override:** VK\_LAYER\_PATH and VK\_ICD\_FILENAMES
- **Implicit layers**
- **Other useful environment variables**

- Linux: /etc/vulkan/\*, /usr/share/vulkan/\*, \$HOME/.local/share/vulkan/\*
- Windows: registry HKEY\_LOCAL\_MACHINE\SOFTWARE\Khronos\Vulkan\\*
- environment variables can be used to override these search paths (VK\_LAYER\_PATH and VK\_ICD\_FILENAMES)

# Vulkan Loader - Windows & Linux

- **Search paths for driver and layer json manifest files**
  - Linux: /etc/vulkan/\*, /usr/share/vulkan/\*, \$HOME/.local/share/vulkan/\*
  - Windows: registry HKEY\_LOCAL\_MACHINE\SOFTWARE\Khronos\Vulkan\\*
  - Environment override VK\_LAYER\_PATH and VK\_ICD\_FILENAMES
- **Implicit layers**
  - **Automatically enabled by the loader**
  - **Disable if needed**
  - **Limited environments**
- **Other useful environment variables**

- platform installed layers such as the Steam overlay
- enabled automatically by the loader rather than by the app
- for security each implicit layer must have an environment variable to disable
- environment variable for enabling in limited environments within a platform

# Vulkan Loader - Windows & Linux

- **Search paths for driver and layer json manifest files**
  - Linux: /etc/vulkan/\*, /usr/share/vulkan/\*, \$HOME/.local/share/vulkan/\*
  - Windows: registry HKEY\_LOCAL\_MACHINE\SOFTWARE\Khronos\Vulkan\\*
  - Environment override VK\_LAYER\_PATH and VK\_ICD\_FILENAMES
- **Implicit layers**
  - Steam overlay
  - Automatically enabled by the loader
  - Disable if needed
  - Limited environments
- **Other useful environment variables**
  - **VK\_INSTANCE\_LAYERS=** layers to be enabled at CreateInstance
  - **VK\_LOADER\_DEBUG=** (all, error, info, warn, debug)

- VK\_INSTANCE\_LAYERS= a list of layers to be enabled at CreateInstance
- VK\_LOADER\_DEBUG= (all, error, info, warn, debug) log loader messages to (debug) console

# Vulkan Loader - Android

- **Same loader-layer interface, but own code**
  - See *LoaderAndLayerInterface.md* for details
- **Nougat / Android-24**

- Android loader uses same interfaces, but own code base
  - *LoaderAndLayerInterface.md*
  - Does not use layer json files, layers must implement introspection functions (*vkEnumerateInstanceLayerProperties*, *vkEnumerateInstanceExtensionProperties*, etc.)



# Vulkan Loader - Android

- **Same loader-layer interface, but own code**
  - See *LoaderAndLayerInterface.md* for details
- **Nougat / Android-24**
  - Includes Vulkan headers and `vulkan.so` library to link against
  - Android loader available on all Nougat devices
  - Some support on Marshmallow (e.g. NVIDIA Shield Tablet and Shield Console, Samsung S7)
  - Use `vkEnumeratePhysicalDevices` to determine if Vulkan is supported
  - Layer source & binaries included with Android NDK v12 and newer

Details: <https://developer.android.com/ndk/guides/graphics/index.html>

- TODO: Link to Android Vulkan page?

# Vulkan Loader - Android

- **Layers**
  - **No json, layers must implement introspection functions (e.g. *vkEnumerateInstanceLayerProperties*)**
  - **Same layer source as Windows/Linux**
  - **No implicit layers.**
  - **Latest layer source on Github**
- **Applications own Layers**

# Vulkan Loader - Android

- **Layers**

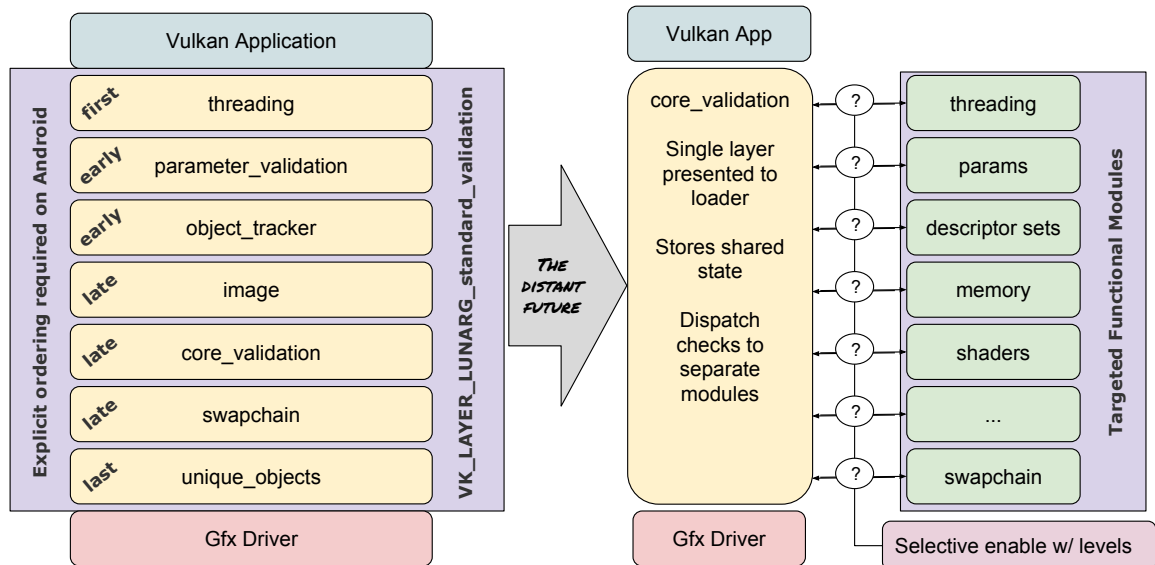
- No json, layers must implement introspection functions (e.g. *vkEnumerateInstanceLayerProperties*)
- Same layer source as Windows/Linux
- No implicit layers.
- Latest layer source on Github

- **Applications own Layers**

- **Application must include layers in apk**
- **No "VK\_LAYER\_LUNARG\_standard\_validation"**
- **Debuggable application can enumerate and enable layers located in /data/local/vulkan/debug.**
- **Check logcat for loader messages**

# Vulkan Validation Layers

# Validation Layers - Overview & future



The Validation layers are used during app development to flag errors in the application's use of the API. Prior to releasing an application, the app should be "validation clean" meaning that it runs without triggering any errors. In release mode the validation layers will not be enabled so they have no performance impact for a release app.

- **Threading** -- Checks that objects are not violating threading restrictions
- **Parameter Validation** -- Stateless API parameter validation - valid usage, validity, device limits, etc.
- **Object Tracker** -- Validates objects for correctness, proper creation, and lifetime
- **Image** -- Validates items related to image generation and usage
- **Core Validation** -- Draw\_state, shader\_checker, mem\_tracker, device\_limits. Main layer for validation
- **Swapchain** -- Validates swapchain-related API calls
- **Unique Objects** -- Remaps object handles to enforce that they are unique

Layer ordering requires threading at the top of the chain and unique\_objects at the bottom. This must be explicitly enabled when requesting layers on Android. On desktop the alternative, single meta-layer "VK\_LAYER\_LUNARG\_standard\_validation" can be used.

In the future we'd like to move to a single layer that handles all of the top-level intercepts. This layer then dispatches work for checks to specific functional

domain modules. These modules will be able to be enabled and disabled as a whole using “level” flags. This will allow for more fine-grained control so that early on in app development, all levels can be turned on, then, as sections of code solidify those areas of validation can be selectively disabled to improve performance.

Also, by consolidating to single layer it will simplify enabling layers for app developers going forward in that the list of layers will never change. Developers who are interested in the fine-grained control of levels can use that capability, while other developers can just run with all levels enabled. The checks may also be divided into 2-3 “meta-levels” that capture broad swaths of validation capability.

# Validation Layers - Usage

- **Validation Layer Output**
  - Debug Callbacks
  
  - Leverage layer settings file
  
  - Debug callbacks return true/false to continue with subject API call

Quickly hit on three areas, callbacks, output control, and message spam

# Validation Layers - Usage

- **Validation Layer Output**
  - Debug Callbacks
    - **Temp**
    - **Default**
    - **Application**
  - Leverage layer settings file
  - Debug callbacks return true/false to continue with subject API call

Temp callbacks -- for enabling debug output during Instance creation before instance callbacks are available. Example: cube demo.

default callbacks -- If no user callbacks are created, default callbacks will output to stdout/outputdebugstring on Windows, stdout on Linux

app callback -- complete control of what happens -- skip\_call

Debug callbacks can continue with execution of the Vulkan API function by returning false or bail out by returning true



# Validation Layers - Usage

- **Validation Layer Output**
  - Debug Callbacks
    - Temp
    - Default
    - Application
  - Leverage layer settings file
    - **Set level info, warn, error, perf\_warn**
    - **Output to VK\_DBG\_LAYER\_ACTION\_LOG\_MSG or VK\_DBG\_LAYER\_ACTION\_DEBUG\_OUTPUT**
  - Debug callbacks return true/false to continue with subject API call

Settings file must go in dir with executable

Pick levels (info, warn, error, etc), use combinations

Control Output -- LOG\_MSG to go to a file or stdout (if no file), DEBUG\_OUTPUT to use OutputDebugString in Windows

# Validation Layers - Usage

- **Validation Layer Output**
  - Debug Callbacks
    - Temp
    - Default
    - Application
  - Leverage layer settings file
    - Set level info, warn, error, perf\_warn
    - Output to VK\_DBG\_LAYER\_ACTION\_LOG\_MSG or VK\_DBG\_LAYER\_ACTION\_DEBUG\_OUTPUT
  - **Debug callbacks return true/false to continue with subject API call**

Filter by type, location (line number), layer, string match.

Debug callbacks can continue with execution of the Vulkan API function by returning false or bail out by returning true

# Validation Layers - Usage

- **Best Practices**

- Use `standard_validation` when possible
- Address `ERRORS` immediately - errors result in undefined behavior and often to crashes
- `WARNING` does *not* imply incorrect behavior
- The `api_dump` layer is your friend
- Debug layer loading issues with `VK_LOADER_DEBUG=all`

LAYER DEPENDENCIES: Examples -- `unique_objects`, parameter validation, object tracker, etc.

- Order does matter. `unique_objects` must be closest to the driver
- `threading`, `parameter_validation` and `object_tracker` need to go early in the chain to prevent invalid references, etc.
- Easiest way (on desktop) is to use the `standard_validation` meta-layer. On mobile, use the list referenced in `vk_validation_layer_details.md` or `layers.html`
- For performance, some layers can be disabled temporarily -- `threading`, `uniq`

`ERRORS` can cascade to cause further validation failures

`WARNING` is a signal to make sure you know what you are doing, NOT a failure.

API-dump: Like many of us used to do with APITrace, we can follow execution and dependencies through time. EG., Layout Transitions -- Many reports of incorrect validation - most turn out to be application-side issues. API-dump is excellent for this.

`VK_LOADER_DEBUG=all` useful for validating that layers are getting loaded correctly

# Validation Layers - Status

- **Activity**
  - **For SDKs 1.0.1 through 1.0.21 (8 SDKs over about 6 months):**
    - **1450+ commits**
    - **222+ Github and 180+ LunarXchange issues closed**
- **Coverage**

Validation layers are a very active area of development in the Vulkan ecosystem.

There have been a lot of Github issues filed

Good news is that there have been a lot of commits to fix the issues.

The fixes are all in GitHub.

Fixes also made available over 8 SDKs since Vulkan went public about 6 months ago.

KEY message is that you should always get the latest validation layers because they are improving and evolving rapidly.

# Validation Layers - Status

- **Activity**
  - For SDKs 1.0.1 through 1.0.21 (8 SDKs over ~ 6 months):
    - 1450+ commits
    - 222+ Github and 180+ LunarXchange issues resolved
- **Coverage**
  - **All areas of the spec have coverage - ongoing work in thinner areas**
  - **Valid usage coverage is in the 60-70% range at a minimum**
  - **Areas needing additional attention:**
    - **Compute**
    - **Sparse resources**
    - **Compressed format validation**

It is pretty tricky to evaluate coverage.

There is some coverage over the entire spec.

But some areas need deeper and more detailed coverage.

There are a few areas that especially need additional work

# How You Can Help: submit issues

- **For GitHub issues for both the loader and layers**
  - Provide test case for issue
  - Pull requests preferred!
  - See CONTRIBUTING file in the GitHub repo for details
  - <https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers>
- **For validation layer issues**
  - Submit issues for false positives as well as for missing validation checks
- **For SDK-specific issues**
  - Report issues with SDK or issues related to loader and layers
  - Ask questions
  - Submit to LunarXchange @ <https://vulkan.lunarg.com>

Best place to report problems is the LoaderAndValidationLayers GitHub

Test cases are great!

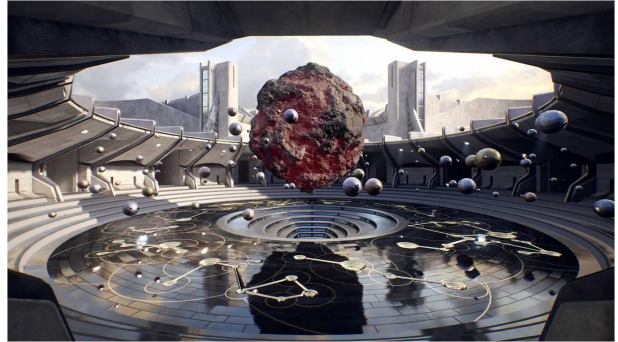
If you want to dig into the Validation Layer code, submit a patch via a pull request

The LunarXchange web site is also a good place to report problems.

# **Vulkan on UE4: Validation Layers**

# Vulkan on UE4: Validation Layers

- **What to expect**
- **Be sure to check out tomorrow's talk!**
  - Will explain how UE4 works with Vulkan and the Protostar demo





## Vulkan on UE4: Key Learnings

- **Use the Validation Layers!**
  - To find bugs:
    - Try to draw using deleted sampler
    - Present using uninitialized image/backbuffer
  - To diagnose cross-platform issues:
    - Missing resource transitions/barriers on images
    - Memory leaks
    - Bad bits/properties
  - To check for performance issues:
    - Writing to disabled attachments
- **Vital for getting UE4 up & running!**

Invalid VkSampler Object 0x<handle>

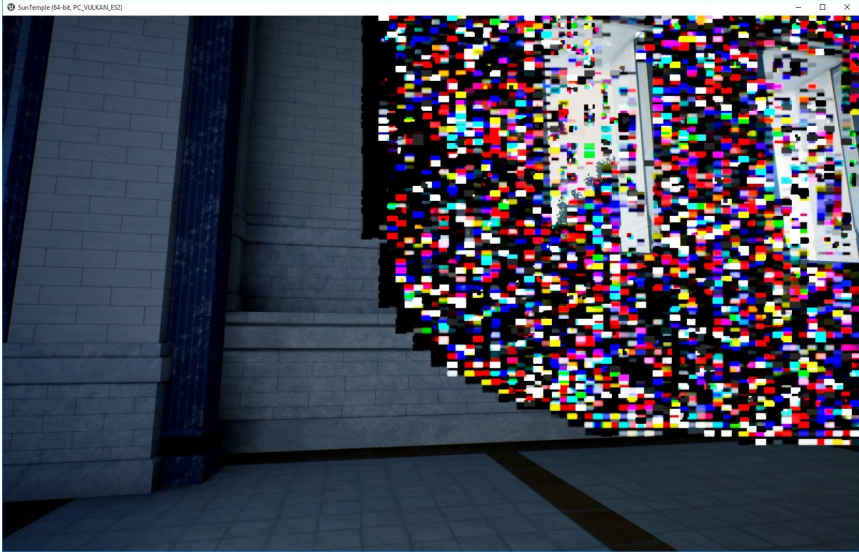
# Vulkan on UE4: Validations

- **Draw using deleted Sampler**
  - VK ERROR: [OBJTRACK] Code 4 : Invalid VkSampler Object 0x443

# Vulkan on UE4: Validations

- **Present with uninitialized image/backbuffer**
  - VK ERROR: [MEM] Code 12 : vkQueuePresentKHR(): Cannot read *invalid swapchain image 0x42a*, please fill the memory before using.

# Vulkan on UE4: Bad image layouts



# Vulkan on UE4: Missing barriers



# Vulkan on UE4: Missing barriers



# Vulkan on UE4: Validations

- **Memory leak**

- VK ERROR: [OBJTRACK] Code 3 : OBJ ERROR :  
VK\_DEBUG\_REPORT\_OBJECT\_TYPE\_SHADER\_MODULE\_EXT object  
0x43d has not been destroyed.

# Vulkan on UE4: Validations

- **Cross platform potential issues**
  - **VK ERROR: [DS] Code 27 : vkUpdateDescriptorSets() *failed write update validation for Descriptor Set* 0x17de with error: Write update to descriptor in set 00000000000017DE binding #1 failed with error message: Attempted write update to combined image sampler descriptor failed due to: *ImageView* (00000000000017AC) has layout VK\_IMAGE\_LAYOUT\_GENERAL and is using depth/stencil image of format VK\_FORMAT\_D24\_UNORM\_S8\_UINT but it has both STENCIL and DEPTH aspects set, which is illegal. **When using a depth/stencil image in a descriptor set, please only set either VK\_IMAGE\_ASPECT\_DEPTH\_BIT or VK\_IMAGE\_ASPECT\_STENCIL\_BIT depending on whether it will be used for depth reads or stencil reads respectively.****



# Vulkan on UE4: Validations

- Cross platform potential issues
  - **VK ERROR: [DS] Code 27 : vkUpdateDescriptorSets() *failed write update validation for Descriptor Set 0x17de [...]***

```
UE4Editor-VulkanRHI-Win64-Debug.dll!DebugReportFunction Line 65 C++
VkLayer_core_validation.dll!debug_report_log_msg Line 122 C++
VkLayer_core_validation.dll!log_msg Line 360 C++
VkLayer_core_validation.dll!cvdescriptorset::ValidateUpdateDescriptorSets Line 868 C++
VkLayer_core_validation.dll!core_validation::PreCallValidateUpdateDescriptorSets Line 6087 C++
VkLayer_core_validation.dll!core_validation::UpdateDescriptorSets Line 6103 C++
VkLayer_object_tracker.dll!object_tracker::UpdateDescriptorSets Line 3936 C++
VkLayer_device_limits.dll!device_limits::UpdateDescriptorSets Line 588 C++
VkLayer_parameter_validation.dll!parameter_validation::UpdateDescriptorSets Line 3508 C++
VkLayer_threading.dll!threading::UpdateDescriptorSets Line 1323 C++
UE4Editor-VulkanRHI-Win64-Debug.dll!FVulkanBoundShaderState::UpdateDescriptorSets Line 1216 C++
UE4Editor-VulkanRHI-Win64-Debug.dll!FVulkanPendingState::PrepareDraw Line 588 C++
UE4Editor-VulkanRHI-Win64-Debug.dll!FVulkanCommandListContext::RHIDrawIndexedPrimitive Line 556 C++
UE4Editor-RHI-Win64-Debug.dll!FRHICommandDrawIndexedPrimitive::Execute Line 153 C++
```

# Vulkan on UE4: Layer Performance

- **But:**
  - Be wary of runtime performance cost
  - Selectively enable validation layers:
    - Start with all the layers:
      - VK\_LAYER\_LUNARG\_standard\_validation
    - At minimum keep:
      - VK\_LAYER\_LUNARG\_parameter\_validation
    - Mem leaks:
      - VK\_LAYER\_LUNARG\_object\_tracker

# Vulkan on UE4: All Validation Layers On

PROFILING WITH AI LOGGING ON!  
 PROFILING WITH GC VERIFY ON!

Vulkan RHIL [STATGROUP\_VulkanRHIL]

Cycle counters (lat)

	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
Draw call time	32	3.14 ms	3.64 ms	0.95 ms	1.11 ms
Draw call prep time	32	2.19 ms	2.55 ms	0.15 ms	0.21 ms
Topology Bind	32	1.11 ms	1.29 ms	0.32 ms	0.47 ms
Update DescriptorSets	32	0.92 ms	1.07 ms	0.07 ms	0.09 ms
Upload DS	32	0.81 ms	0.94 ms	0.31 ms	0.39 ms
Bind Vertex Streams	75	0.25 ms	0.32 ms	0.29 ms	0.32 ms
SRV Update Time	2	0.05 ms	0.07 ms	0.05 ms	0.07 ms
Set uniform buffer	78	0.05 ms	0.06 ms	0.05 ms	0.06 ms
Deletion Queue	1	0.04 ms	0.06 ms	0.04 ms	0.05 ms
Create uniform buffer time	26	0.03 ms	0.04 ms	0.02 ms	0.03 ms
Apply DS Uniform Buffers	164	0.02 ms	0.04 ms	0.02 ms	0.04 ms
Set Shader Param	275	0.02 ms	0.03 ms	0.02 ms	0.03 ms
Get or create pipeline	32	0.01 ms	0.02 ms	0.01 ms	0.02 ms
Apply DS Shader Resources	164	0.01 ms	0.02 ms	0.01 ms	0.02 ms
Uniform Buffer Creation Time	26	0.01 ms	0.02 ms	0.01 ms	0.02 ms
DrawPrim UP Prep Time	50	0.01 ms	0.02 ms	0.01 ms	0.02 ms
Clear Dirty DS State	32	0.00 ms	0.01 ms	0.00 ms	0.01 ms

Counters

	Average	Max
Num Render Passes	380.00	
Num WriteDescriptors Cmd	267.00	267.00
Num Desc Sets Updated	164.00	164.00
Num Bound Shader States		26.00



# Vulkan on UE4: No Validation Layers

PROFILING WITH AI LOGGING ON!  
 PROFILING WITH GC VERIFY ON!

Vulkan RHIL [STATGROUP\_VulkanRHIL]

Cycle counters (lat)

	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
Draw call time	77	0.22 ms	0.88 ms	0.09 ms	0.28 ms
Draw call prep time	77	0.17 ms	0.22 ms	0.04 ms	0.05 ms
Update DescriptorSets	77	0.05 ms	0.11 ms	0.04 ms	0.05 ms
Pipeline Bind	77	0.05 ms	0.06 ms	0.04 ms	0.04 ms
Create uniform buffer time	26	0.05 ms	0.05 ms	0.02 ms	0.03 ms
Set uniform buffer	78	0.05 ms	0.04 ms	0.03 ms	0.04 ms
SRV Update Time	2	0.03 ms	0.05 ms	0.03 ms	0.05 ms
VU update DS	77	0.02 ms	0.03 ms	0.02 ms	0.03 ms
Uniform Buffer Creation Time	26	0.01 ms	0.03 ms	0.01 ms	0.03 ms
Bind Vertex Streams	70	0.01 ms	0.02 ms	0.01 ms	0.02 ms
Get or create pipeline	77	0.00 ms	0.01 ms	0.00 ms	0.01 ms
Apply DS Uniform Buffers	155	0.01 ms	0.02 ms	0.01 ms	0.02 ms
Set Shader Param	257	0.01 ms	0.02 ms	0.01 ms	0.02 ms
Deletion Queue	1	0.01 ms	0.02 ms	0.01 ms	0.02 ms
Apply DS Shader Resources	155	0.01 ms	0.02 ms	0.01 ms	0.02 ms
Draw Prim UP Prep Time	45	0.00 ms	0.01 ms	0.00 ms	0.01 ms
Clear Dirty DS State	77	0.00 ms	0.00 ms	0.00 ms	0.00 ms

Counters

	Average	Max
Num Render Passes	382.00	
Num Write Descriptors Cmd	259.30	255.00
Num Desc Sets Updated	154.87	155.00
Num Bound Shader States		26.00



# Vulkan on UE4: Layer Performance

- **In this sample (x64 optimized)**
  - ~80 draw calls
  - Draw call time: 3.14ms -> 0.26ms
  - vkUpdateDescriptorSets: 0.81ms ->0.02ms
  - vkCmdBindPipeline and BindDescriptorSets: 1.11ms->0.05ms

## Vulkan on UE4: Key Learnings

- **Step into Validation Layers source code!**
  - Immensely useful to find out \*why\* it's failing
  - Even after reading the spec, it might not be clear \*how\* it's supposed to work, so the source will guide you on the how
- **Continuously improving**
  - Every SDK has had more information added and catches more error/perf cases
- **But:**
  - All software has bugs :)
  - Make sure a new warning is not a bug in the validation layer itself!

# Vulkan on UE4

- **But wait, there's more!**
  - With great power comes more complexity!
  - Layer usage is non-negotiable :)
  - Having a second machine is useful as sometimes it *\*is\** a driver issue...

# Vulkan on UE4: Sad Trombone



Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (61% complete)


If you'd like to know more, you can search online later for this error: PAGE\_FAULT\_IN\_NONPAGED\_AREA (dxgmm2.sys)



## Vulkan on UE4

- **Finally...**
  - Write your own layers!
  - Report bugs!
  - Contribute!
- **Thanks!**

# Where to find info...

- **LunarG.com (<http://www.lunarg.com>)**
  - BoF slides: <https://lunarg.com/lunarg-birds-feather-session-siggraph-july-26-2016/>
- **Khronos Loader and Validation Layers github**
  - github: <https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers>
  - Loader specification and architecture: <https://github.com/KhronosGroup/Vulkan-LoaderAndValidationLayers/blob/master/loader/LoaderAndLayerInterface.md>
- **Khronos Vulkan API**
  - Vulkan landing page: <https://www.khronos.org/vulkan/>
  - Vulkan forum: <https://forums.khronos.org/forumdisplay.php/114-Vulkan-High-Efficiency-GPU-Graphics-and-Compute>
- **LunarG Vulkan SDK** 
  - Download SDK, report SDK issues, read documentation: <https://vulkan.lunarg.com>

# BACKUP SLIDES

# QUESTIONS

- 1) If I was worried about perf with layers enabled, what is the minimum validation you would recommend?
- 2) What could cause my application to work when layers are enabled, but fail when layers are disabled? (unique objects, threading locking, bugs in layers may hide incorrect behavior...)
- 3) Where can we find info on suggested behaviours to consider when creating a new layer? (i.e. no wrapping, continue down the call-chain, loader and layer interface guide, look at existing layers, ...)
- 4) Can I write a layer and get it added to Android? (common layer open source ecosystem, contribute and ends up on all platforms including Android if you did a good job)
- 5) What if I wanted to insert my own layer in the middle of the standard validation layers?
- 6) What happened to device layers?
- 7) Why isn't the loader owned by the OS like it is on OpenGL? (MS doesn't do this....)
- 8) Can I write my own loader? (yes, but why?)
- 9) Where is the specification for the layers? (no formal spec, look at validation layer implementation, see loader-layer interface document in the SDK)
- 10) Should there be formal spec definition of validation cases? (spec describes correct behavior. many many cases for incorrect behavior and almost impossible to specify)
- 11) What are the most important validation tasks going fwd? (coverage of valid usage cases, performance, code clean up)
- 12) Why does performance take such a hit with my multi-threaded app when I turn on validation? Plans to improve? (validation is complex, many un-optimized locks, work is ongoing to improve)
- 13) Can a Vulkan application run on Android Marshmallow?

# Vulkan Loader - Extensions

## **Instance Extensions**

Instance extensions are intended to cover / affect all Vulkan devices, layers, etc. Current instance extensions all use different aggregation mechanisms that are implemented inside the loader. We expect future instance extensions to likewise require specific loader support and recommend developers to avoid them if possible.

## **Device Extensions**

Device extensions are implemented in drivers and/or layers and shouldn't require specific loader support.