Introduction to Computer Graphics

186.832, 2021W, 3.0 ECTS

Vulkan Lecture Series, Episode 1:

Vulkan Essentials

Johannes Unterguggenberger

Institute of Visual Computing & Human-Centered Technology

TU Wien, Austria

Vulkan Essentials: Outline

Which Kind of API Is It?

Fundamental API Usage

Validation

Instance, Physical Device, Logical Device

Queues

Extensions

Which Kind of API Is It?

Vulkan Essentials: Which Kind of API Is It?

> Vulkan is a new generation graphics and compute API that provides high-efficiency, cross-platform access to modern GPUs used in a wide variety of devices from PCs and consoles to mobile phones and embedded platforms.
>
> *The Khronos Group. Vulkan Guide* [1)]

- Graphics **and** compute API
- Enables high efficiency by being low-level
- A very explicit (maybe even verbose) API
- Cross-platform and cross-device category

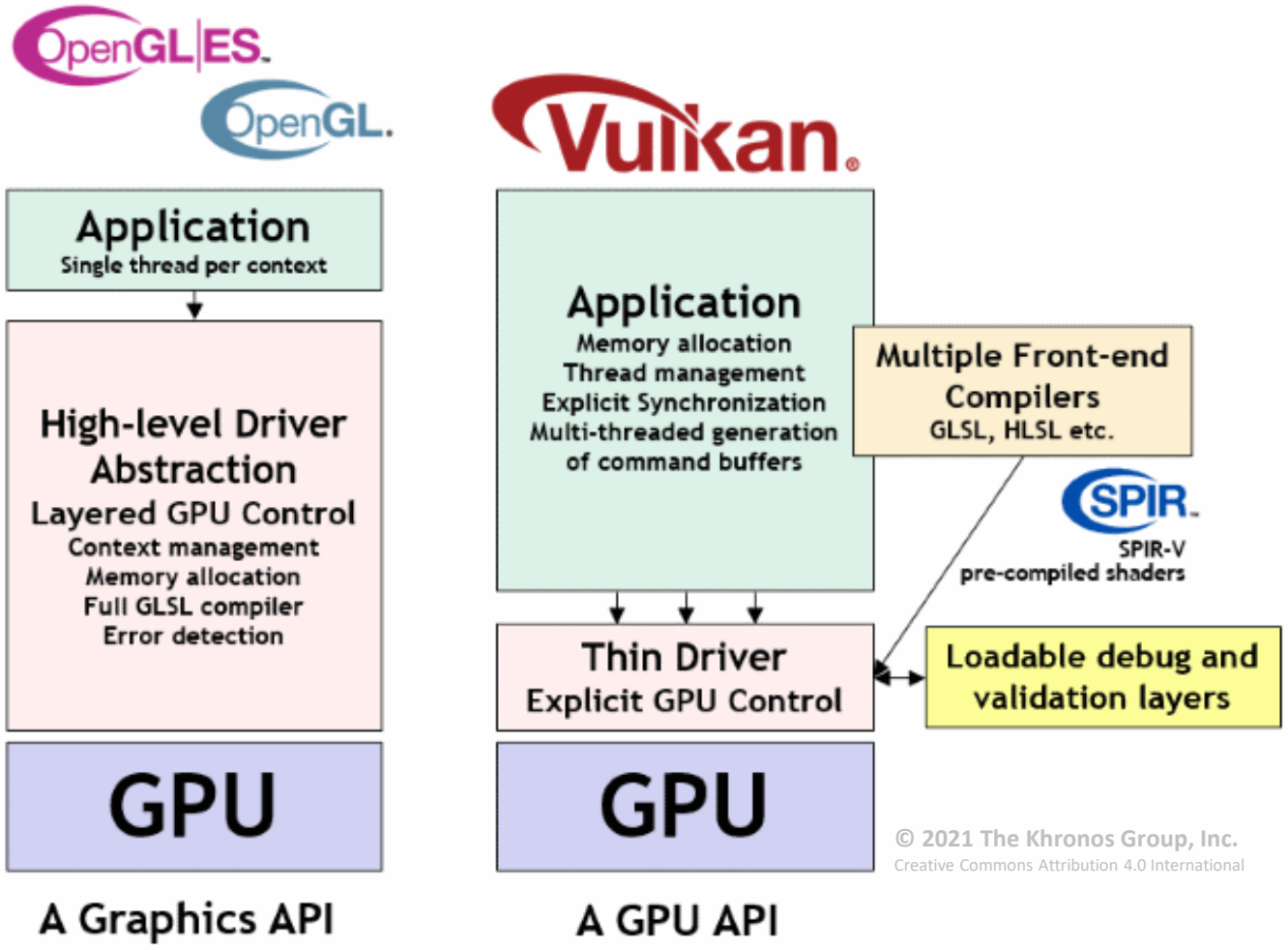
[1)] https://github.com/KhronosGroup/Vulkan-Guide

Vulkan Essentials: Which Kind of API Is It?

Is it like OpenGL?

nope!

- OpenGL
  - High-level API
  - Graphics API
  - Released Jun 30, 1992
- Vulkan
  - Low-level API
  - GPU API
  - Released Feb 16, 2016



Vulkan Essentials: Which Kind of API Is It?

- Derived from the Mantle API (by AMD and DICE)
- Created, maintained, and evolved by the Khronos® Group

Vulkan Essentials: Outline

Which Kind of API Is It?

Fundamental API Usage

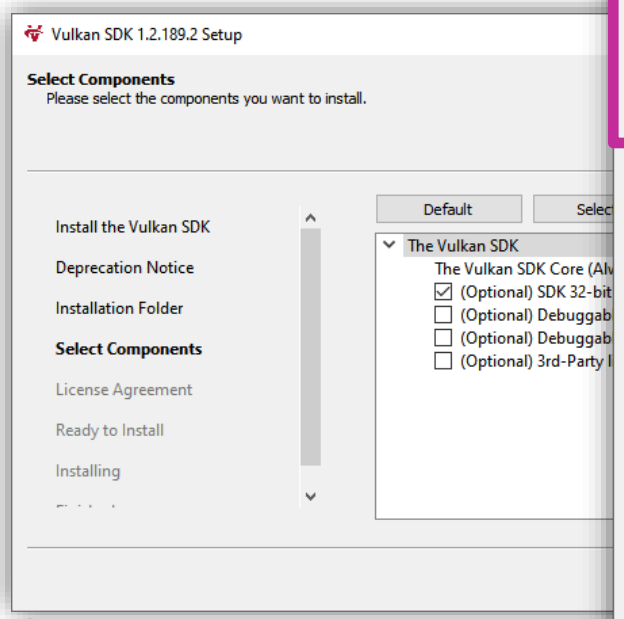
Validation

Vulkan Essentials: Fundamental API Usage, Validation

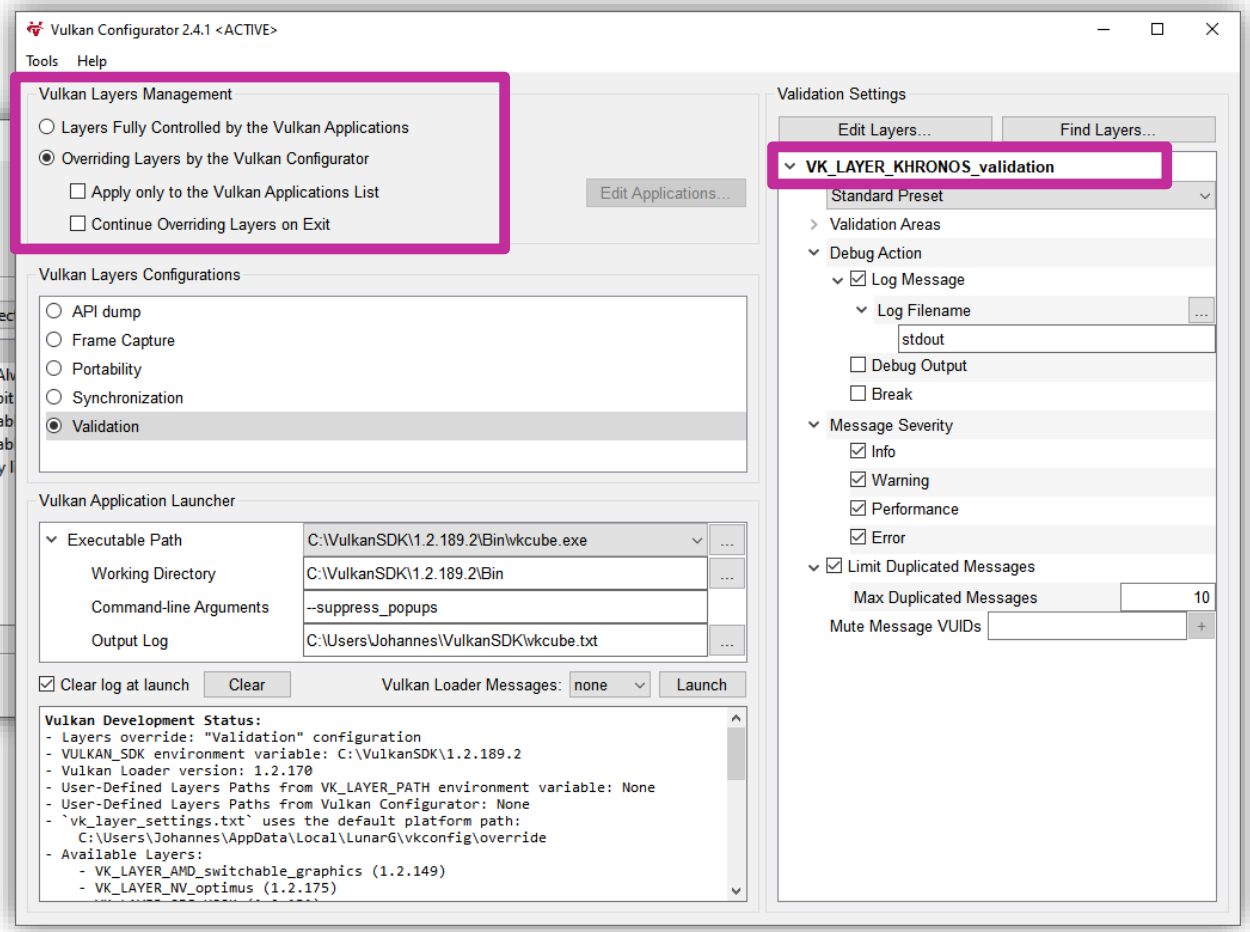
How to start:

- Get an SDK from LunarG's website
- Getting the latest driver for your GPU is always a good idea
- You might want to get a library for window management like GLFW.
- Take a look at Khronos' Vulkan Guide on GitHub
- Use Khronos' official Vulkan Samples repository!
- Keep the Vulkan specification (HTML or PDF) at hand, you'll need it a lot!
- All useful links can be found on vulkan.org

LunarG SDK



Installer



Vulkan Configurator (In Windows: `VULKAN_SDK\Tools\vkconfig.exe`)

SDK contains:

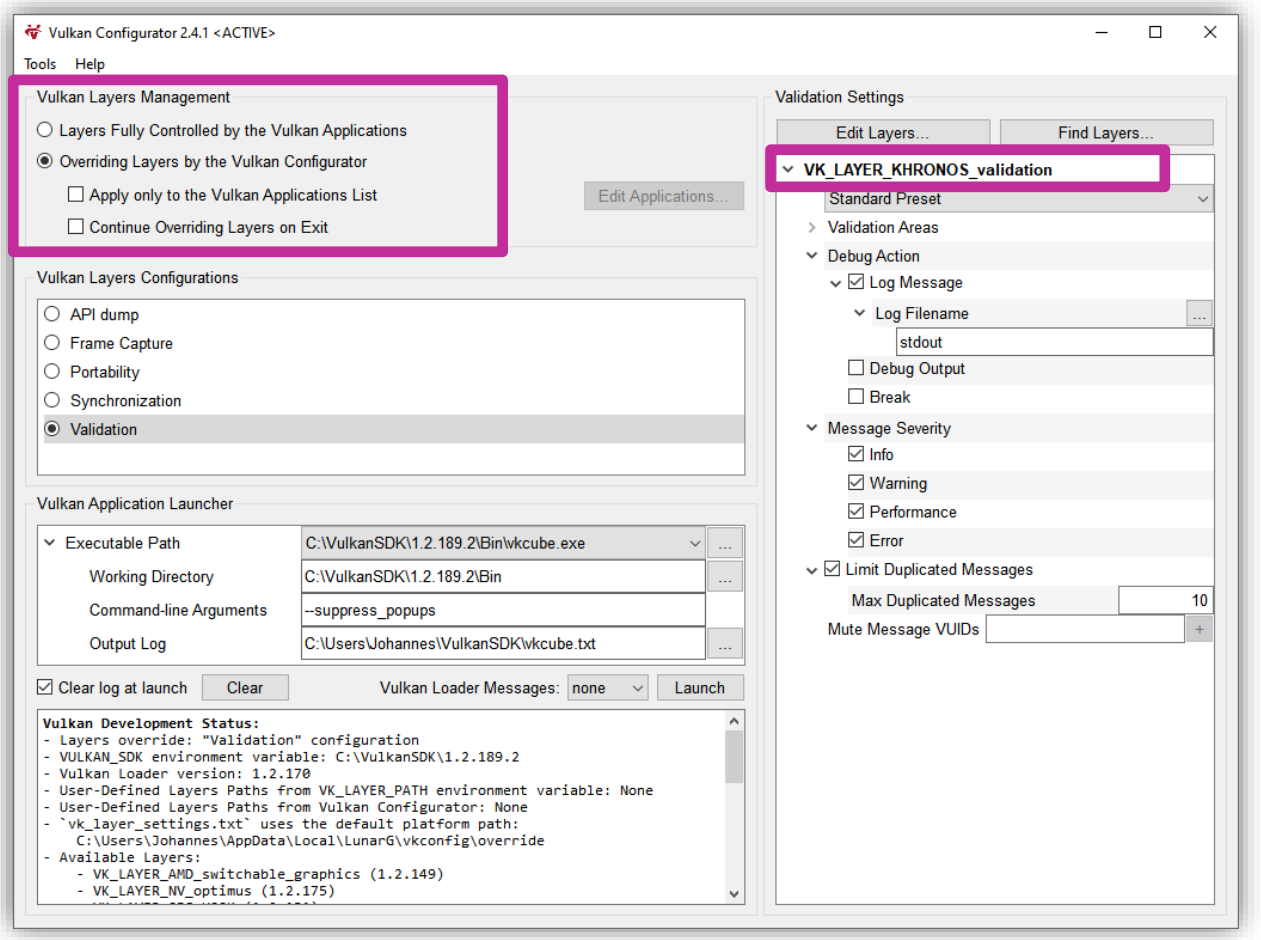
- Headers
- Libraries
- Tools

Programming Language Support:

- C API (native)
- C++ via Vulkan-Hpp (included in the SDK)
- Other language bindings available: see Khronosdotorg GitHub repository

Validation Layers

are your best friends!



Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Validation OFF

```
// Crea
VkImag
create_
create_
create_
create_
create_
create_
create_
create_
create_

VkImag
VkResu
CHECK_
```

I:\workwork\ECGFramework\repo\ECG_SolutionGenerator\SolutionTemplate\bin\Debug\ECG_VK_Solution.exe

```
INFO:  Vulkan operation returned status code: VK_SUCCESS (in Main.cpp at line #472)
```

Validation ON

```
// Crea
VkImag
create_
c
c
c
c
c
c
c
V
VkResu
CHECK_
```

Microsoft Visual Studio Debug Console

```
INFO:  Vulkan operation returned status code: VK_SUCCESS (in Main.cpp at line #472)

I:\workwork\ECGFramework\repo\ECG_SolutionGenerator\SolutionTemplate\bin\Debug\ECG_VK_Solution.exe (process 33400) exited with code -1.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

vkCreateImage(): if pCreateInfo->imageType is VK_IMAGE_TYPE_2D, pCreateInfo->extent.depth must be 1. The Vulkan spec states: If imageType is VK_IMAGE_TYPE_2D, extent.depth must be 1 (https://vulkan.lunarg.com/doc/view/1.2.189.2/windows/1.2-extensions/vkspec.html#VUID-VkImageCreateInfo-imageType-00957)

The Vulkan Specification

is your best friend, too!

```
// Cre
VkImag
create
create
create
create
create
create
create
create

VkImag
VkResu
CHECK_
```

about:blank | Vulkan® 1.2.189 - A Specificatio | Learn | Vulkan | Cross platform 3

https://vulkan.lunarg.com/doc/view/1.2.189.2/windows/1.2-extensions/vkspec.html#VUID-VkImageCreateInfo-imageType-00957

Table of Contents

- VUID-VkImageCreateInfo-imageType-00957
  If imageType is VK_IMAGE_TYPE_2D, extent.depth **must** be 1
- VUID-VkImageCreateInfo-mipLevels-00958
  mipLevels **must** be less than or equal to the number of levels in the complete mipmap chain based on extent.width, extent.height, and extent.depth
- VUID-VkImageCreateInfo-mipLevels-02255
  mipLevels **must** be less than or equal to imageCreateMaxMipLevels (as defined in Image Creation Limits)
- VUID-VkImageCreateInfo-arrayLayers-02256
  arrayLayers **must** be less than or equal to imageCreateMaxArrayLayers (as defined in Image Creation Limits)
- § VUID-VkImageCreateInfo-imageType-00961
  If imageType is VK_IMAGE_TYPE_3D, arrayLayers **must** be 1
- VUID-VkImageCreateInfo-samples-02257
  If samples is not VK_SAMPLE_COUNT_1_BIT, then imageType **must** be VK_IMAGE_TYPE_2D, flags **must** not contain VK_IMAGE_CREATE_CUBE_COMPATIBLE_BIT, mipLevels **must** be equal to 1, and imageCreateMaybeLinear (as defined in Image Creation Limits) **must** be VK_FALSE,
- VUID-VkImageCreateInfo-samples-02558
  If samples is not VK_SAMPLE_COUNT_1_BIT, usage **must** not contain VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT
- VUID-VkImageCreateInfo-usage-00963
  If usage includes VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT, then bits other than VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, and VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT **must** not be set
- VUID-VkImageCreateInfo-usage-00964
  If usage includes VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT, or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.width **must** be less than or equal to VkPhysicalDeviceLimits::maxFramebufferWidth
- VUID-VkImageCreateInfo-usage-00965
  If usage includes VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT, VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT, VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT, or VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT, extent.height **must** be less than or equal to VkPhysicalDeviceLimits::maxFramebufferHeight
- VUID-VkImageCreateInfo-usage-02559
  If usage includes VK_IMAGE_USAGE_FRAGMENT_DENSITY_MAP_BIT_EXT, extent.width **must** be less than or equal to $\left\lceil \frac{maxFramebufferWidth}{minFragmentDensityTexelSize_{width}} \right\rceil$

The Vulkan Specification

is very explicit.

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create
create
create
create
create
create
```

```
vkCreateImage: value of pCreateInfo->usage must not be 0.
The Vulkan spec states: usage must not be 0
(https://vulkan.lunarg.com/doc/view/1.2.189.2/windows/1.2
-extensions/vkspec.html#VUID-VkImageCreateInfo-usage-
requiredbitmask)
```

```
VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

The Vulkan Specification

has all the information.

```
// Cre
VkImag
create
create
create
create
create
create
create
create
create
```

```
VkImag
VkResu
CHECK_
```

https://www.khronos.org/registry/vulkan/specs/1.2-extensions/man/html/VkImageUsageFlagBits.html

VkImageUsageFlagBits(3) Manual Page

Name

VkImageUsageFlagBits - Bitmask specifying intended usage of an image

C Specification

Bits which **can** be set in VkImageViewUsageCreateInfo::usage, or VkImageStencilUsageCreateInfo::stencilUsage, or VkImageCreateInfo::usage, specifying intended usage of an image, are:

```
// Provided by VK_VERSION_1_0
typedef enum VkImageUsageFlagBits {
    VK_IMAGE_USAGE_TRANSFER_SRC_BIT = 0x00000001,
    VK_IMAGE_USAGE_TRANSFER_DST_BIT = 0x00000002,
    VK_IMAGE_USAGE_SAMPLED_BIT = 0x00000004,
    VK_IMAGE_USAGE_STORAGE_BIT = 0x00000008,
    VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT = 0x00000010,
    VK_IMAGE_USAGE_DEPTH_STENCIL_ATTACHMENT_BIT = 0x00000020,
    VK_IMAGE_USAGE_TRANSIENT_ATTACHMENT_BIT = 0x00000040,
    VK_IMAGE_USAGE_INPUT_ATTACHMENT_BIT = 0x00000080,
#ifdef VK_ENABLE_BETA_EXTENSIONS
  // Provided by VK_KHR_video_decode_queue
    VK_IMAGE_USAGE_VIDEO_DECODE_DST_BIT_KHR = 0x00000400,
```

Fundamental API Usage

First Example With Validation Errors Fixed

```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.extent.depth      =   1;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;
create_info.usage             = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

Fundamental API Usage

First Example With Validation Errors Fixed

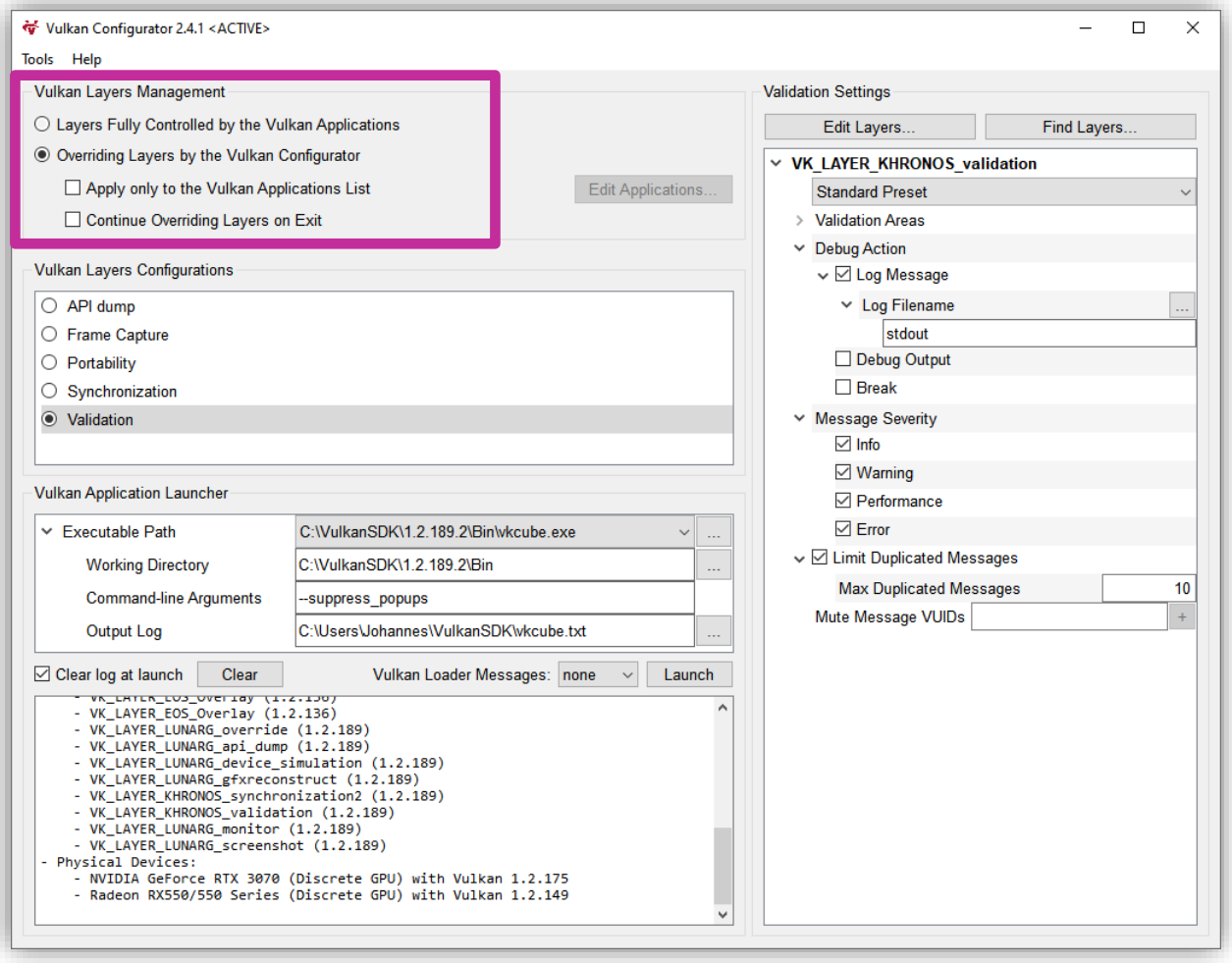
```
// Create a new image:
VkImageCreateInfo create_info = {};
create_info.sType             = VK_STRUCTURE_TYPE_IMAGE_CREATE_INFO;
create_info.imageType         = VK_IMAGE_TYPE_2D;
create_info.format            = VK_FORMAT_R8G8B8A8_UNORM;
create_info.extent.width      = 512;
create_info.extent.height     = 512;
create_info.extent.depth      =   1;
create_info.arrayLayers       =   1;
create_info.mipLevels         =   1;
create_info.samples           = VK_SAMPLE_COUNT_1_BIT;
create_info.usage             = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;

VkImage image;
VkResult result = vkCreateImage(device, &create_info, nullptr, &image);
CHECK_VULKAN_RESULT(result);
```

***Important:* Either zero-initialize, or set every member!**

***Caution:* Don't forget to set the structure type!** This member is present in almost all Vulkan API structs and a valid value starts with VK_STRUCTURE_TYPE_*

Validation Layers

are your best friends!



Validation Layers

are your best friends!

Vulkan Configurator 2.4.1 <ACTIVE>

Tools Help

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

Edit Applications...

Vulkan Layers Configurations

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

Vulkan Application Launcher

Executable Path	C:\VulkanSDK\1.2.189.2\Bin\wkcube.exe
Working Directory	C:\VulkanSDK\1.2.189.2\Bin
Command-line Arguments	--suppress_popups
Output Log	C:\Users\Johannes\VulkanSDK\wkcube.txt

Clear log at launch | Clear | Vulkan Loader Messages: none | Launch

```
    - VK_LAYER_EOS_Overlay (1.2.136)
    - VK_LAYER_LUNARG_override (1.2.189)
    - VK_LAYER_LUNARG_api_dump (1.2.189)
    - VK_LAYER_LUNARG_device_simulation (1.2.189)
    - VK_LAYER_LUNARG_gfxreconstruct (1.2.189)
    - VK_LAYER_KHRONOS_synchronization2 (1.2.189)
    - VK_LAYER_KHRONOS_validation (1.2.189)
    - VK_LAYER_LUNARG_monitor (1.2.189)
    - VK_LAYER_LUNARG_screenshot (1.2.189)
- Physical Devices:
    - NVIDIA GeForce RTX 3070 (Discrete GPU) with Vulkan 1.2.175
    - Radeon RX550/550 Series (Discrete GPU) with Vulkan 1.2.149
```

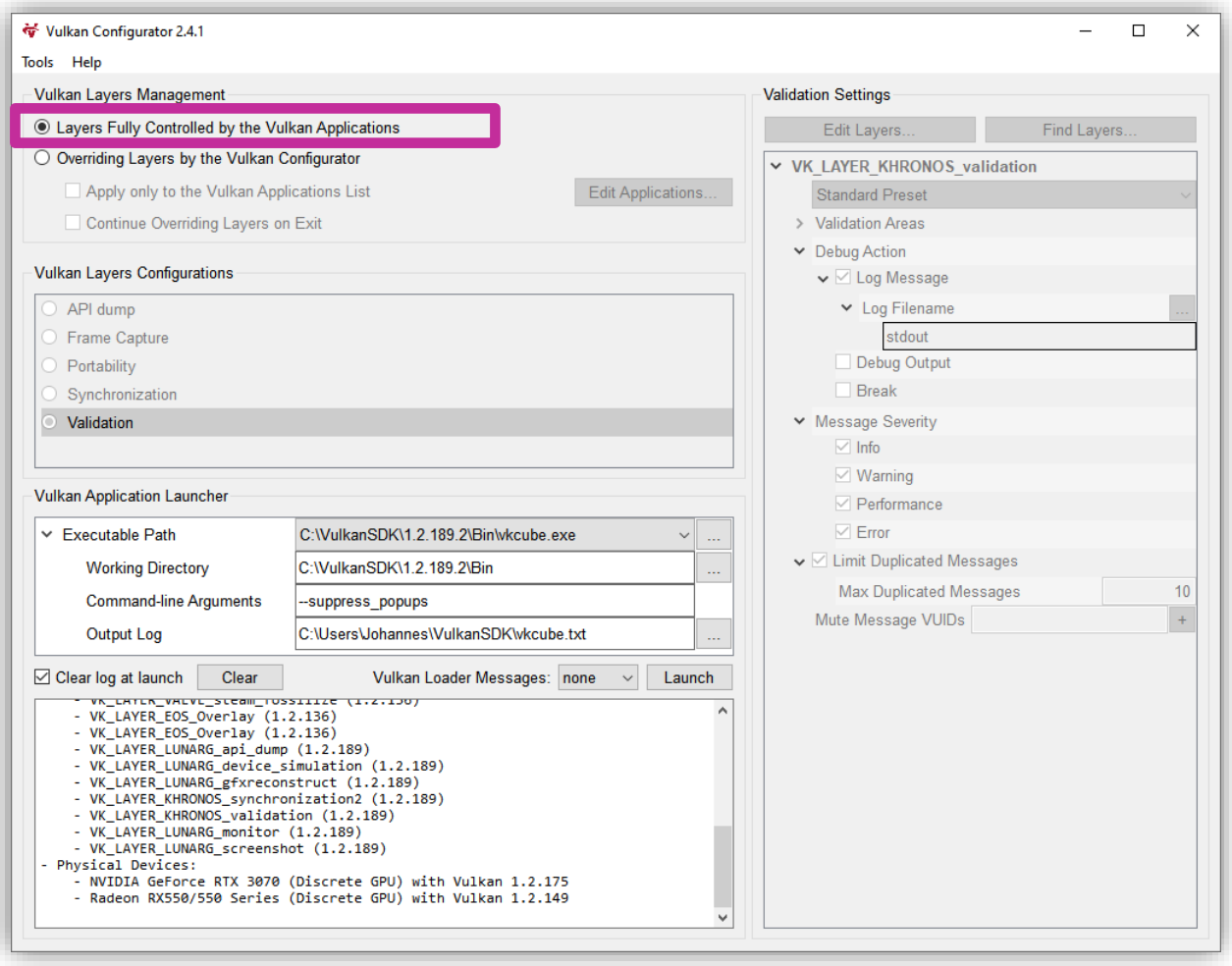
Validation Settings

Edit Layers... | Find Layers...

- VK_LAYER_KHRONOS_validation
  - Standard Preset
  - Validation Areas
  - Debug Action
    - Log Message
      - Log Filename: stdout
    - Debug Output
    - Break
  - Message Severity
    - Info
    - Warning
    - Performance
    - Error
  - Limit Duplicated Messages
    - Max Duplicated Messages: 10
  - Mute Message VUIDs

Validation Layers

can also be fully controlled by applications.



Validation Layers

can also be fully controlled by applications.

Use `vkCreateDebugUtilsMessengerEXT` to install a callback, which gets invoked whenever the validation layers have something to report.

C Specification

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    const VkDebugUtilsMessengerCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugUtilsMessengerEXT*                   pMessenger);
```


Best way to approach this: Follow the official Khronos example Debug Utilities for the best practice-approach!

Validation Layers

can also be fully controlled by applications.

Use `vkCreateDebugUtilsMessengerEXT` to install a callback, which gets invoked whenever the validation layers have something to report.

C Specification

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    const VkDebugUtilsMessengerCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugUtilsMessengerEXT*                   pMessenger);
```

Also in this case, a “CreateInfo” configuration struct is used.

Best way to approach this: Follow the official Khronos example Debug Utilities for the best practice-approach!

Validation Layers

can also be fully controlled by applications.

C Specification

The definition of `VkDebugUtilsMessengerCreateInfoEXT` is:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsMessengerCreateInfoEXT {
    VkStructureType                         sType;
    const void*                             pNext;
    VkDebugUtilsMessengerCreateFlagsEXT     flags;
    VkDebugUtilsMessageSeverityFlagsEXT     messageSeverity;
    VkDebugUtilsMessageTypeFlagsEXT         messageType;
    PFN_vkDebugUtilsMessengerCallbackEXT    pfnUserCallback;
    void*                                   pUserData;
} VkDebugUtilsMessengerCreateInfoEXT;
```

There's the callback function

Use `vkCreateDebugUtilsMessengerEXT` to install a callback, which gets invoked whenever the validation layers have something to report.

Also in this case, a "CreateInfo" configuration struct is used.

C Specification

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    const VkDebugUtilsMessengerCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugUtilsMessengerEXT*                   pMessenger);
```


Best way to approach this: Follow the official Khronos example Debug Utilities for the best practice-approach!

Validation Layers

can also be fully controlled by applications.

C Specification

The definition of VkDebugUtilsMessengerCreateInfoEXT is:

```
// Provided by VK_EXT_debug_utils
typedef struct VkDebugUtilsMessengerCreateInfoEXT {
    VkStructureType                         sType;
    const void*                             pNext;
    VkDebugUtilsMessengerCreateFlagsEXT     flags;
    VkDebugUtilsMessageSeverityFlagsEXT     messageSeverity;
    VkDebugUtilsMessageTypeFlagsEXT         messageType;
    PFN_vkDebugUtilsMessengerCallbackEXT    pfnUserCallback;
    void*                                   pUserData;
} VkDebugUtilsMessengerCreateInfoEXT;
```

There's the callback function

Use `vkCreateDebugUtilsMessengerEXT` to install a callback, which gets invoked whenever the validation layers have something to report.

It is an extension, that's why it is suffixed with "EXT".

C Specification

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    const VkDebugUtilsMessengerCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugUtilsMessengerEXT*                   pMessenger);
```

Also in this case, a "CreateInfo" configuration struct is used.

Best way to approach this: Follow the official Khronos example Debug Utilities for the best practice-approach!

Vulkan Essentials: Outline

Which Kind of API Is It?

Fundamental API Usage

Validation

Instance, Physical Device, Logical Device

Vk Essentials: Instance, Physical Device, Logical Device

Main handles to access the Vulkan API:

- Vulkan instance: `VkInstance`
- Physical device: `VkPhysicalDevice`
  It represents one Vulkan-capable hardware device, e.g., a GPU.
- Logical device: `VkDevice`
  - Main interface between your application and the physical device
  - Configuration in which your application runs on the physical device:
    - Which extensions are enabled for your application
    - Queue configuration which your application utilizes
    - Multiple logical devices (~applications) per physical device

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

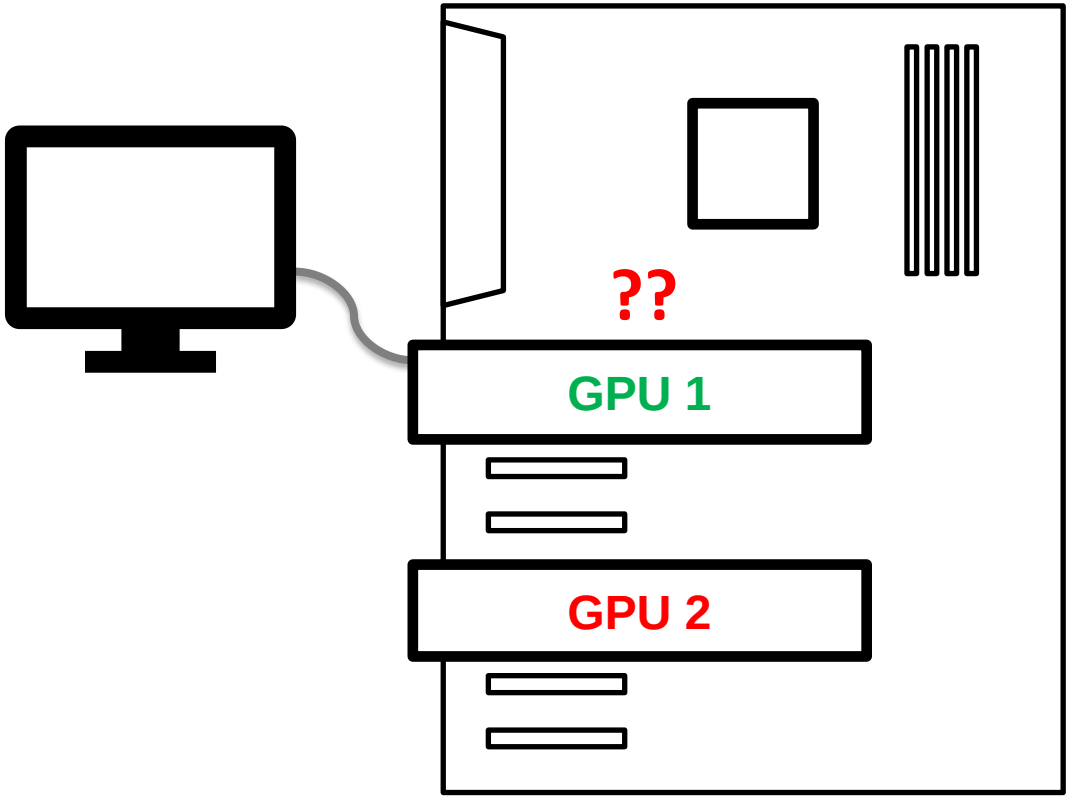
```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

Instance Creation

```
VkApplicationInfo application_info = {};
application_info.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
application_info.apiVersion = VK_API_VERSION_1_2;

const char* instance_extensions[2] = { "VK_KHR_surface", "VK_KHR_win32_surface" };
const char* enabled_layers[1] = { "VK_LAYER_KHRONOS_validation" };

VkInstanceCreateInfo create_info = {};
create_info.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
create_info.pApplicationInfo = &application_info;
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames = instance_extensions;
create_info.enabledLayerCount = 1;
create_info.ppEnabledLayerNames = enabled_layers;

VkInstance instance;
VkResult result = vkCreateInstance(&create_info, nullptr, &instance);
CHECK_VULKAN_RESULT(result);
```

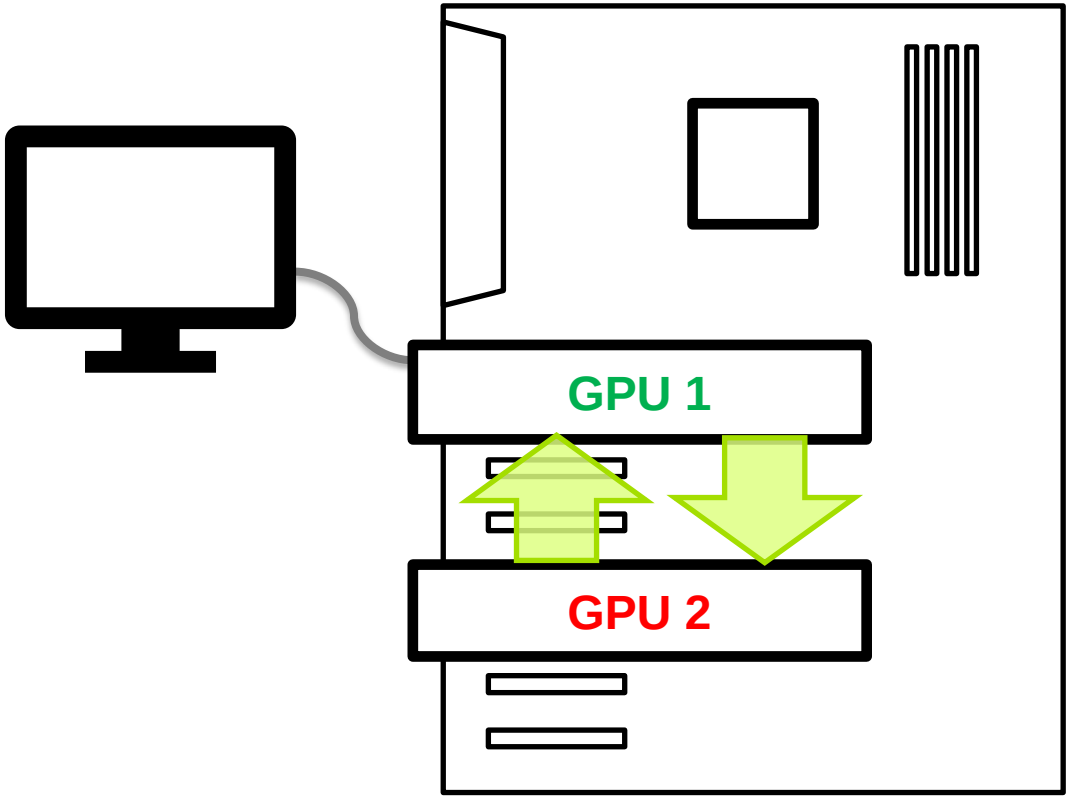
Physical Device Selection

If you have two GPUs, which GPU does the rendering?



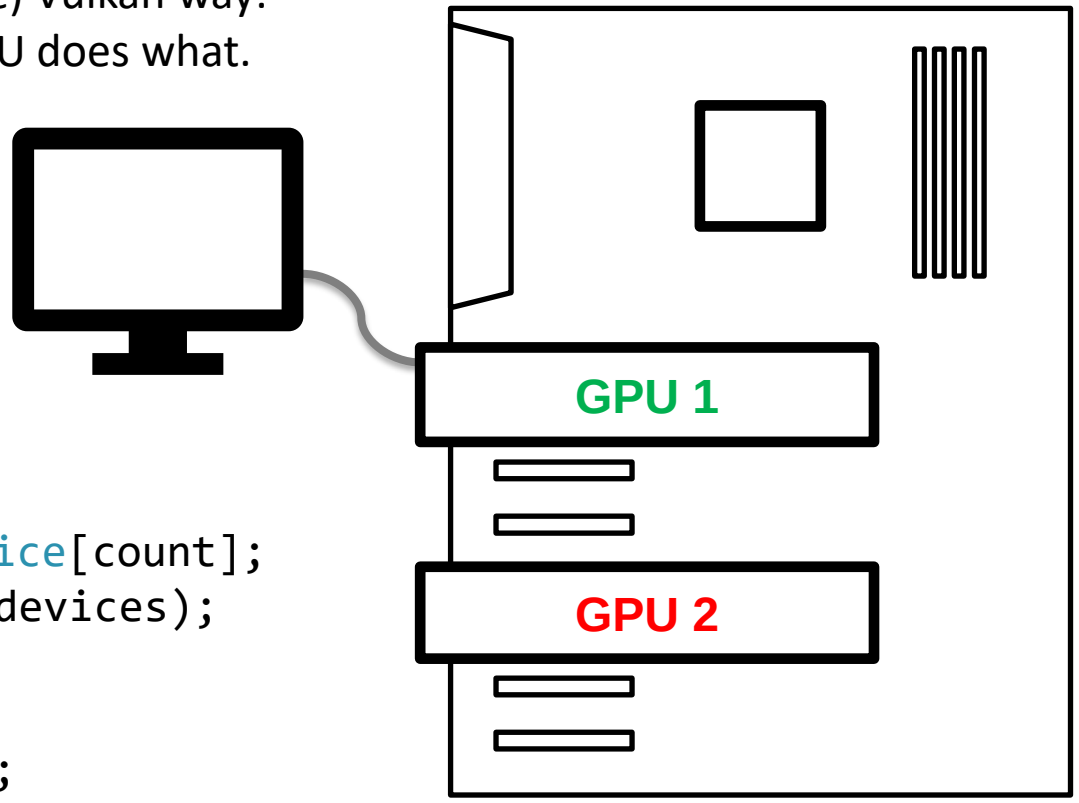
Physical Device Selection

The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.



Physical Device Selection

The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.

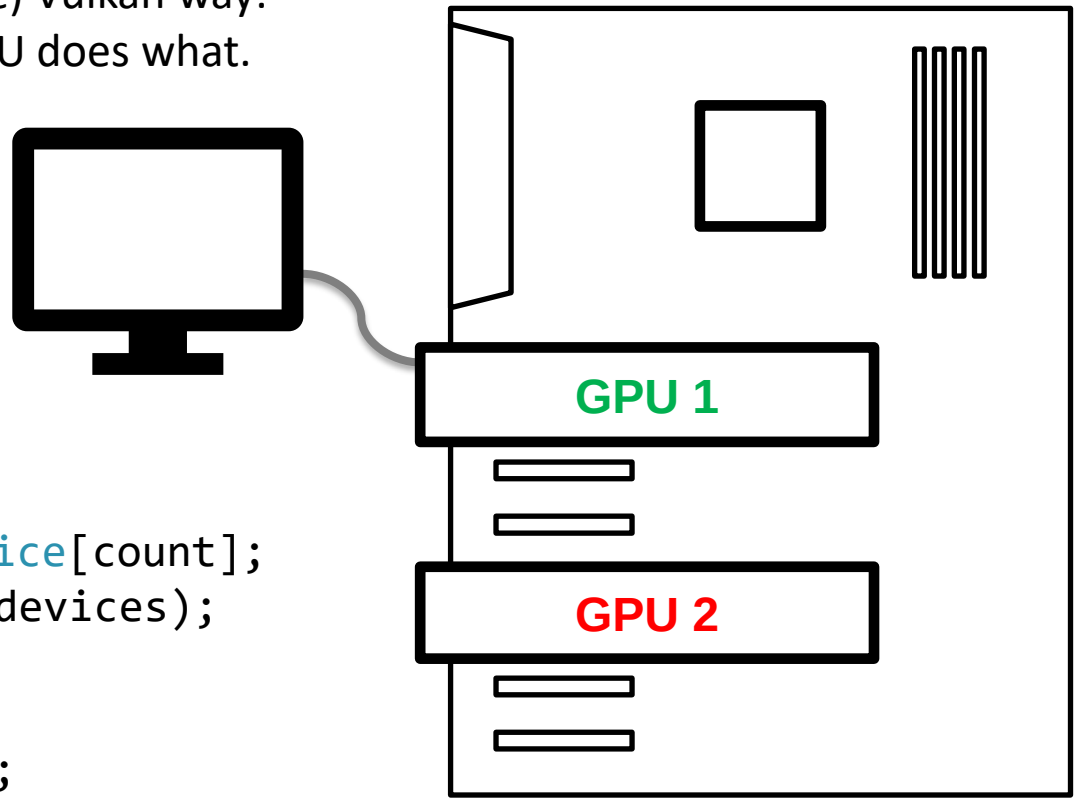



```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Physical Device Selection

The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.

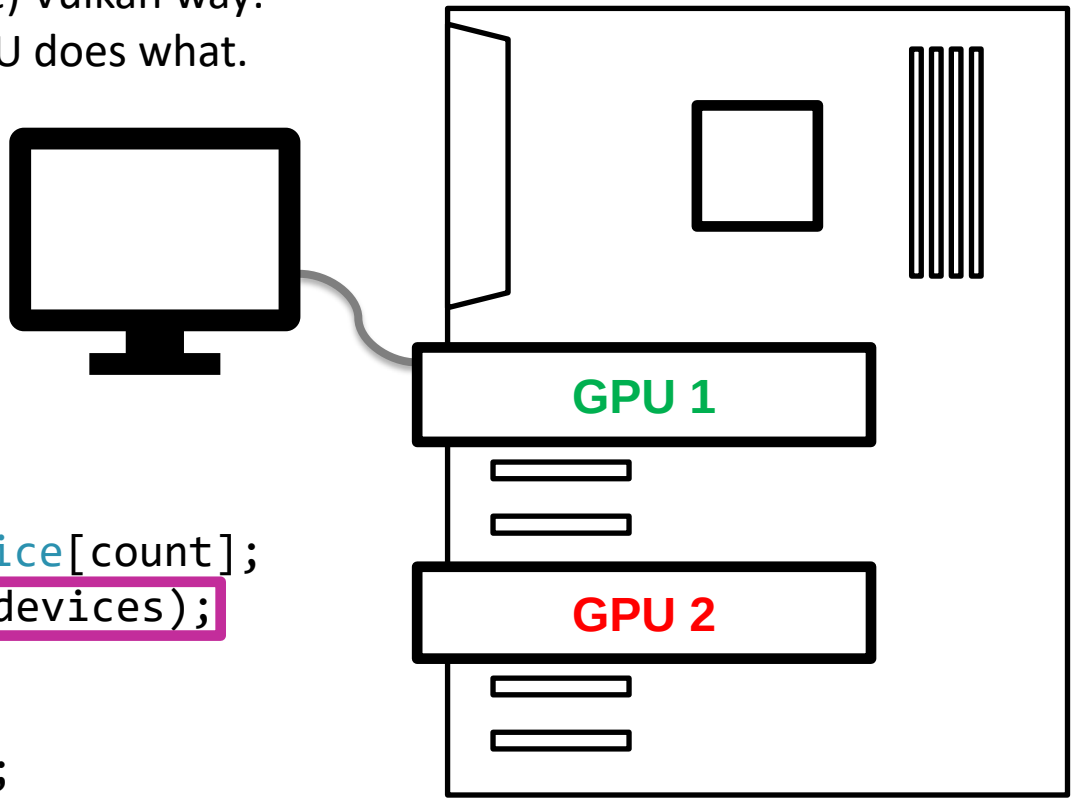
```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```



Physical Device Selection

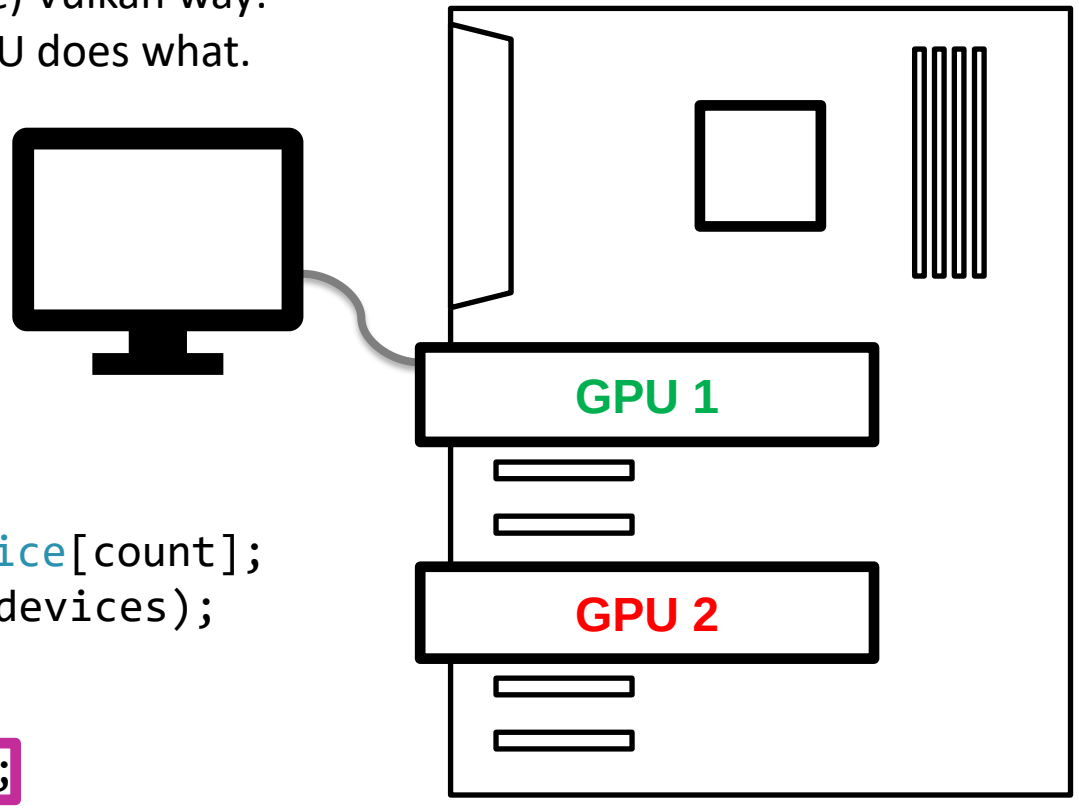
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.

```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```



Physical Device Selection

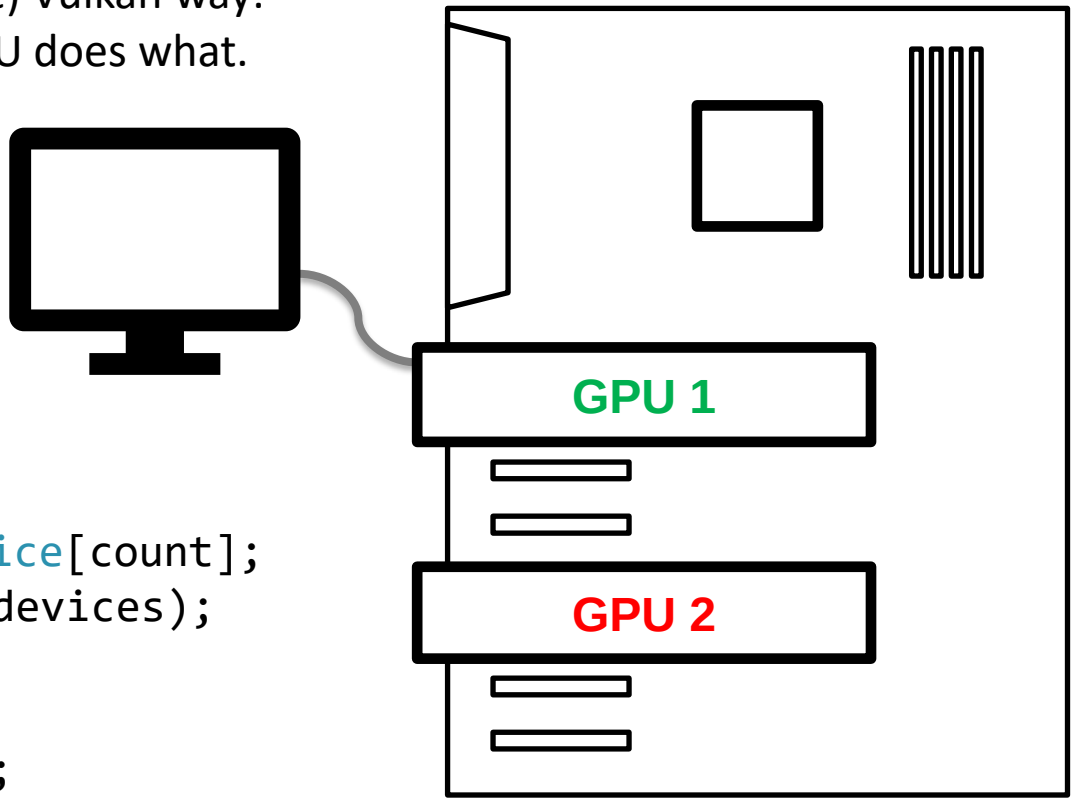
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.




```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Physical Device Selection

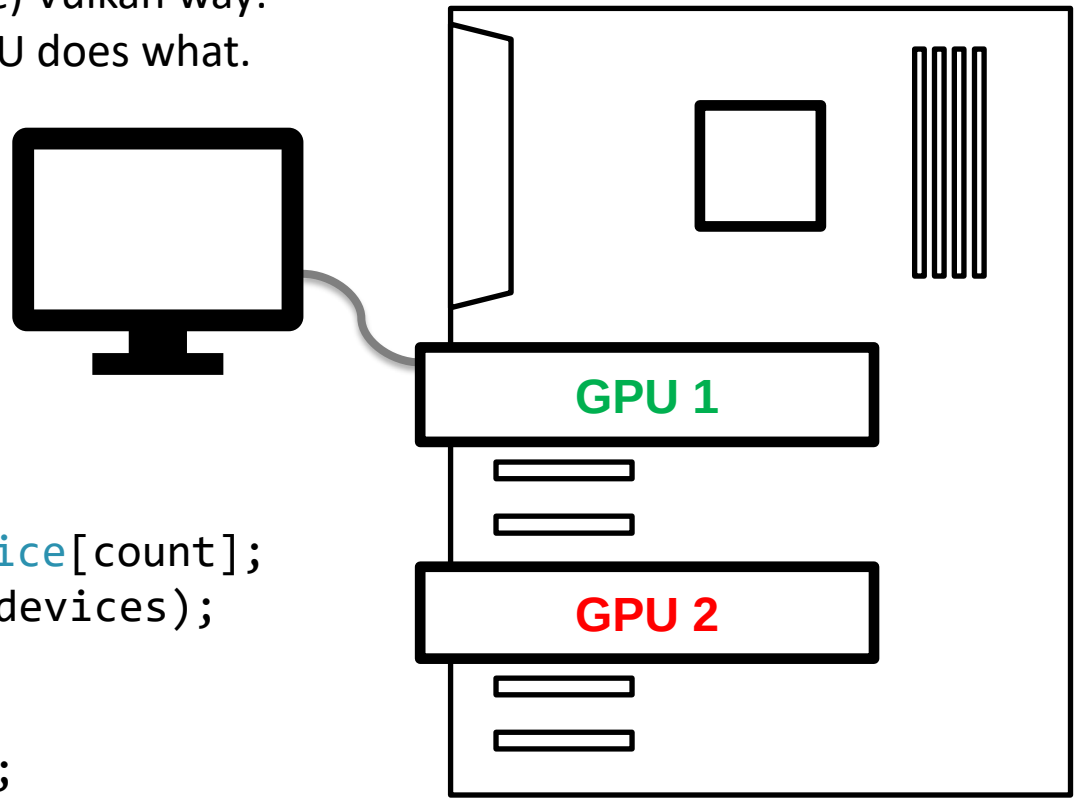
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.




```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Physical Device Selection

The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.




```
// Query the number of physical devices:
uint32_t count;
vkEnumeratePhysicalDevices(instance, &count, nullptr);
assert(count > 0);

// Get all physical device handles:
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];
vkEnumeratePhysicalDevices(instance, &count, physical_devices);

// Select a physical device:
VkPhysicalDevice physical_device = physical_devices[0];

vkEnumerateDeviceExtensionProperties(physical_device, ...);
vkGetPhysicalDeviceProperties(physical_device, ...);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Vulkan Essentials: Outline

Which Kind of API Is It?

Fundamental API Usage

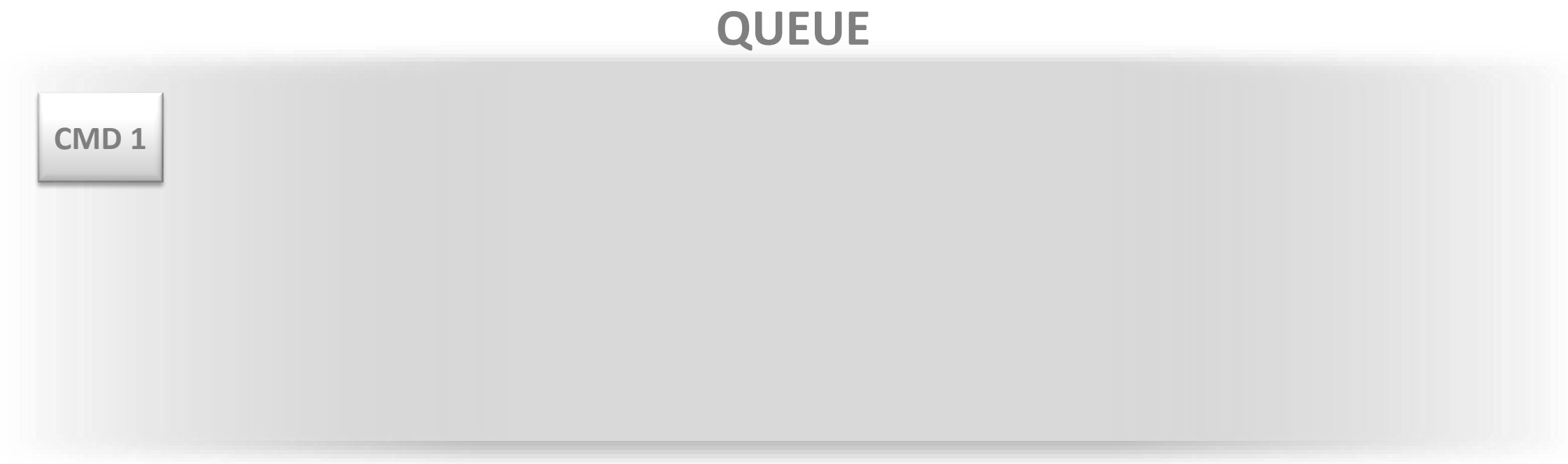
Validation

Instance, Physical Device, Logical Device

Queues

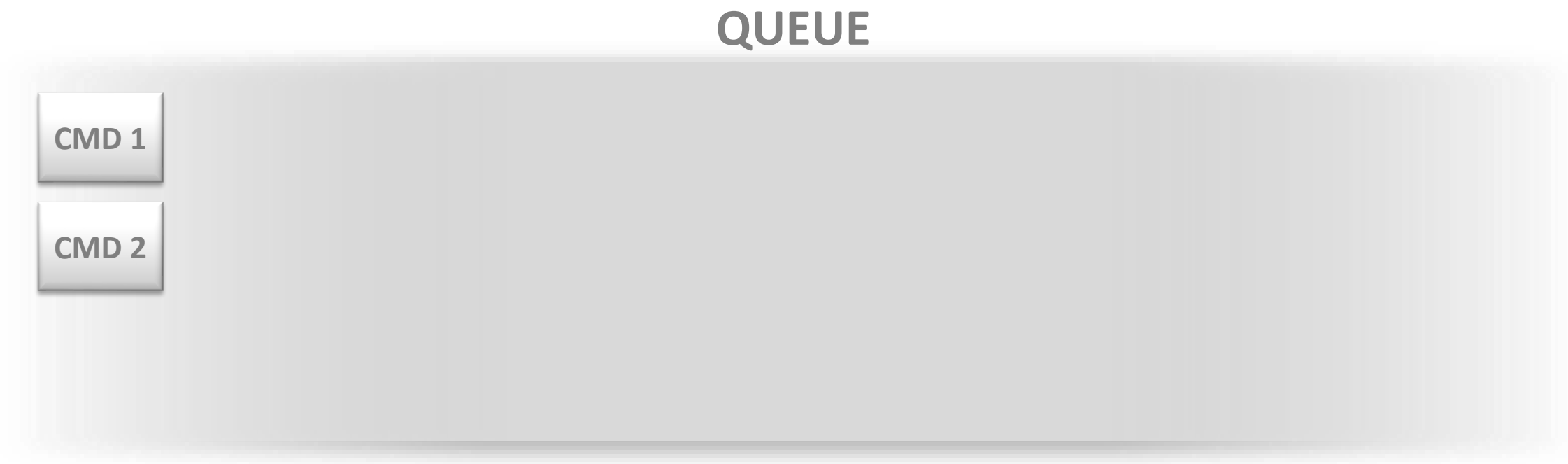
Vulkan Essentials: Queues

- A queue receives commands which are to be processed by the physical device.
- Commands (more precisely: command buffers) are queued for processing.
- Commands start being processed in submission order; can complete out of order



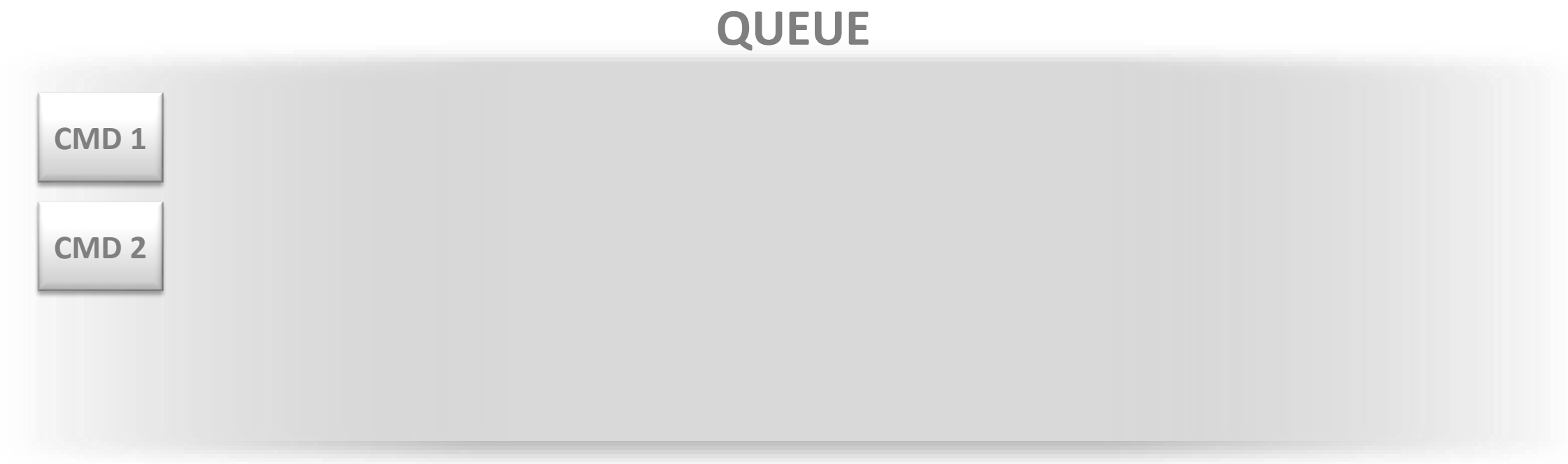
Vulkan Essentials: Queues

- A queue receives commands which are to be processed by the physical device.
- Commands (more precisely: command buffers) are queued for processing.
- Commands start being processed in submission order; can complete out of order



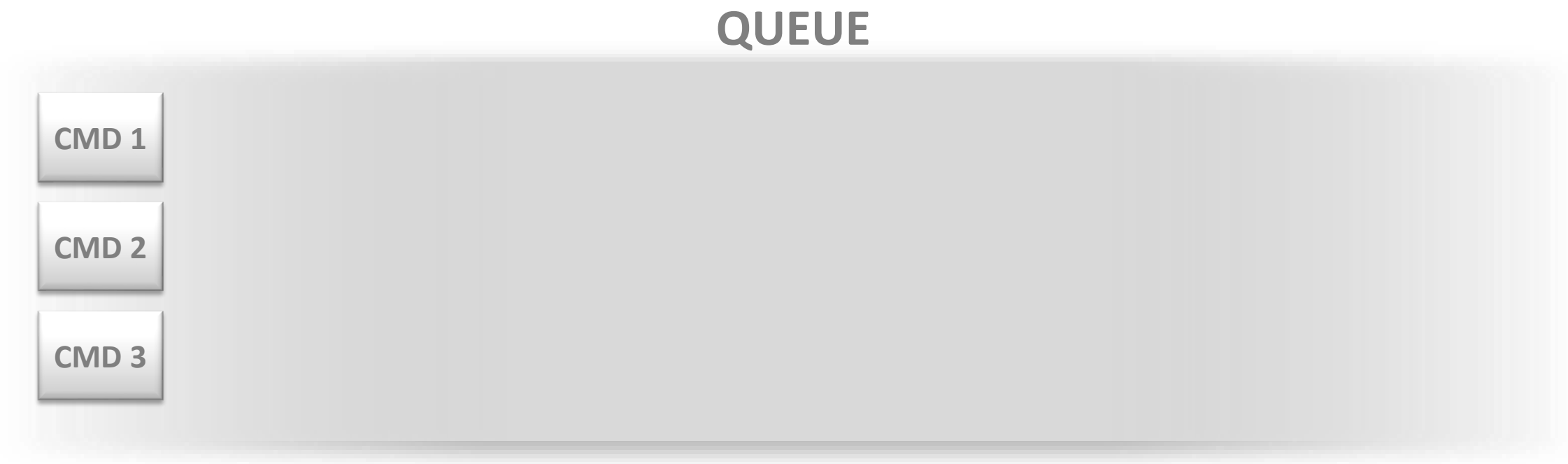
Vulkan Essentials: Queues

- A queue receives commands which are to be processed by the physical device.
- Commands (more precisely: command buffers) are queued for processing.
- Commands start being processed in submission order; can complete out of order



Vulkan Essentials: Queues

- A queue receives commands which are to be processed by the physical device.
- Commands (more precisely: command buffers) are queued for processing.
- Commands start being processed in submission order; can complete out of order



Vulkan Essentials: Queues

QUEUE 1

CMD 1

CMD 2

QUEUE 2

CMD 3

CMD 4

Logical Device Creation

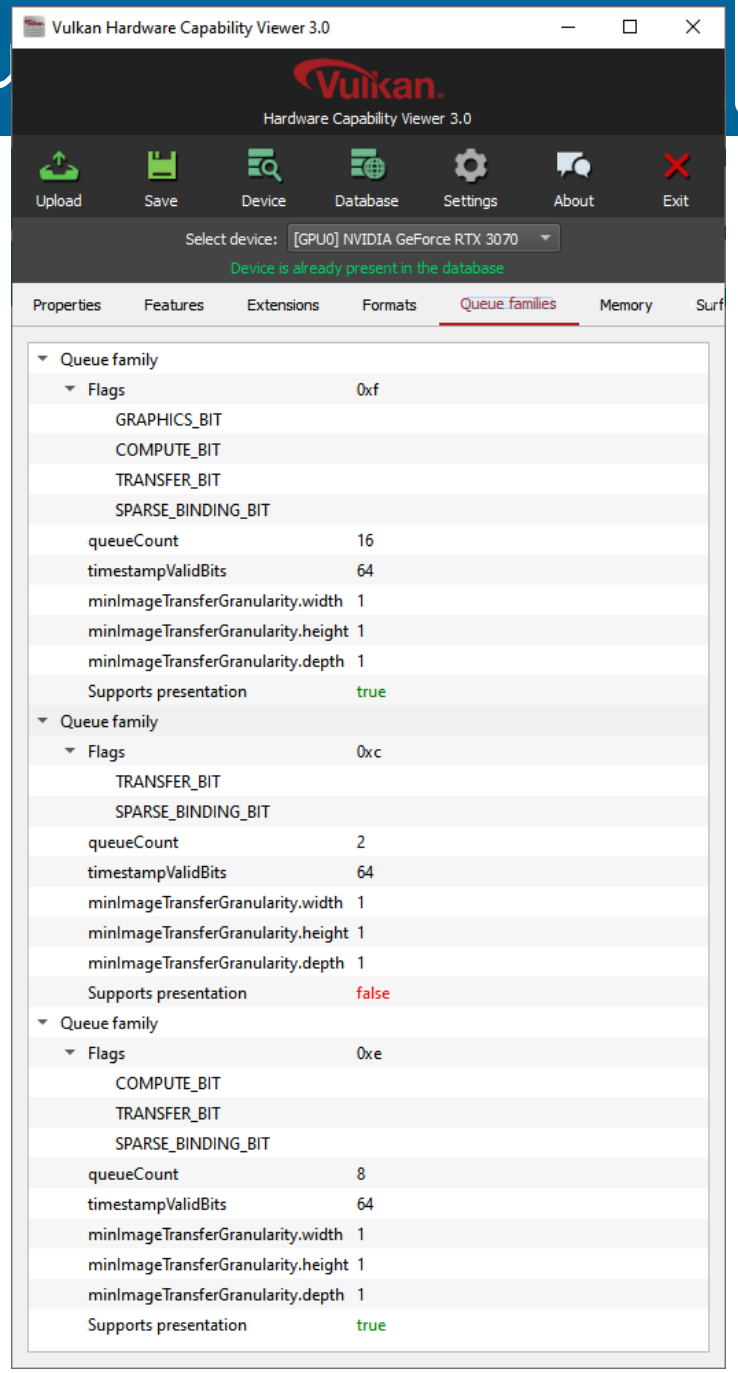
```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Logical Device Creation

```
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info  = {};
queue_create_info.sType                    = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
queue_create_info.queueFamilyIndex         = 0;
queue_create_info.queueCount               = 1;
queue_create_info.pQueuePriorities         = &priority;

const char* enabled_extensions[1]          = { "VK_KHR_swapchain" };

VkDeviceCreateInfo create_info             = {};
create_info.sType                          = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.queueCreateInfoCount           = 1;
create_info.pQueueCreateInfos              = &device_queue_create_info;
create_info.enabledExtensionCount          = 1;
create_info.ppEnabledExtensionNames        = enabled_extensions;

VkDevice device;
VkResult result = vkCreateDevice(physical_device, &create_info, nullptr, &device);
CHECK_VULKAN_RESULT(result);
```

Vulkan Essentials: Queu

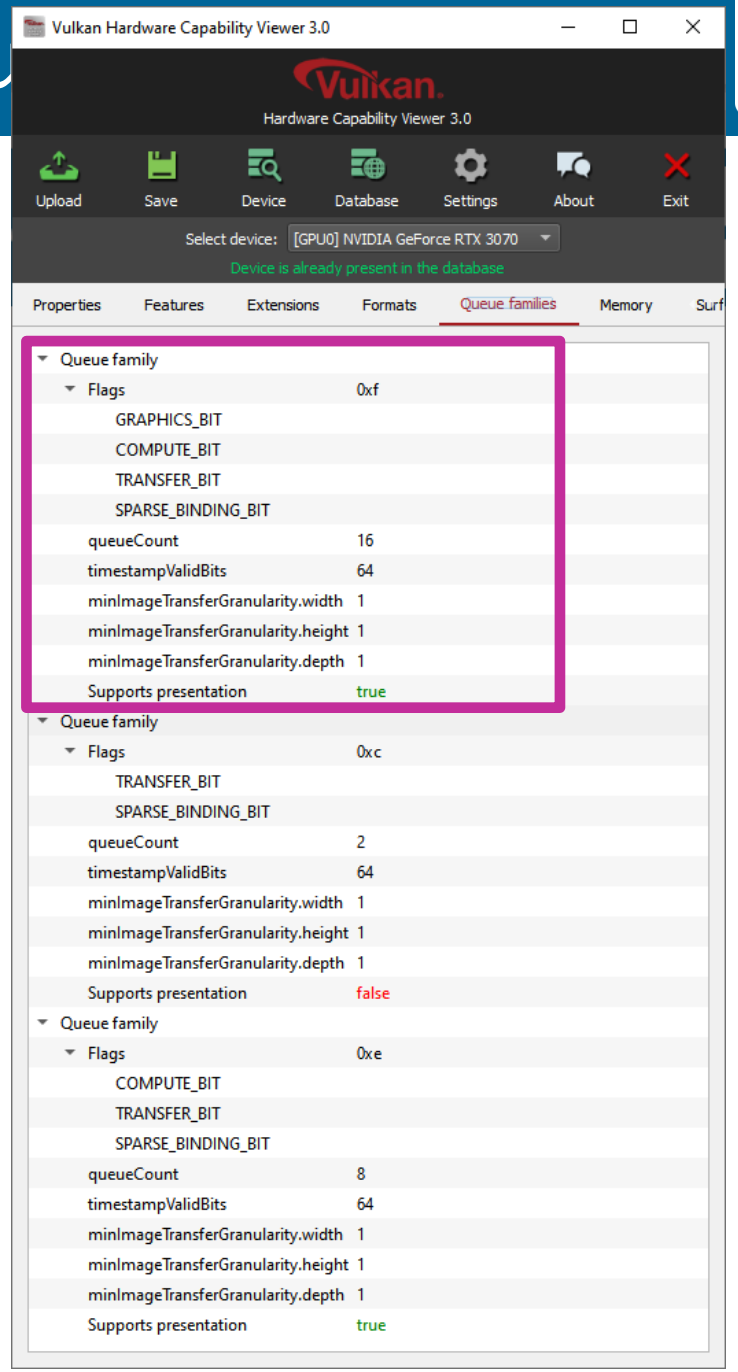
- A **queue** always belongs to a **queue family**.
- Queue families
  - A physical device **can** support different queue families … or only one.
  - Different queue families have different properties.
  - Multiple queues of the same queue family **can** be created and used.
- Why use multiple queues?
  - Increase concurrency
  - (Potentially) increase performance with specialized queues:
    - e.g., a “transfer queue”
    - e.g., an “async compute queue”



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems

Vulkan Essentials: Queu

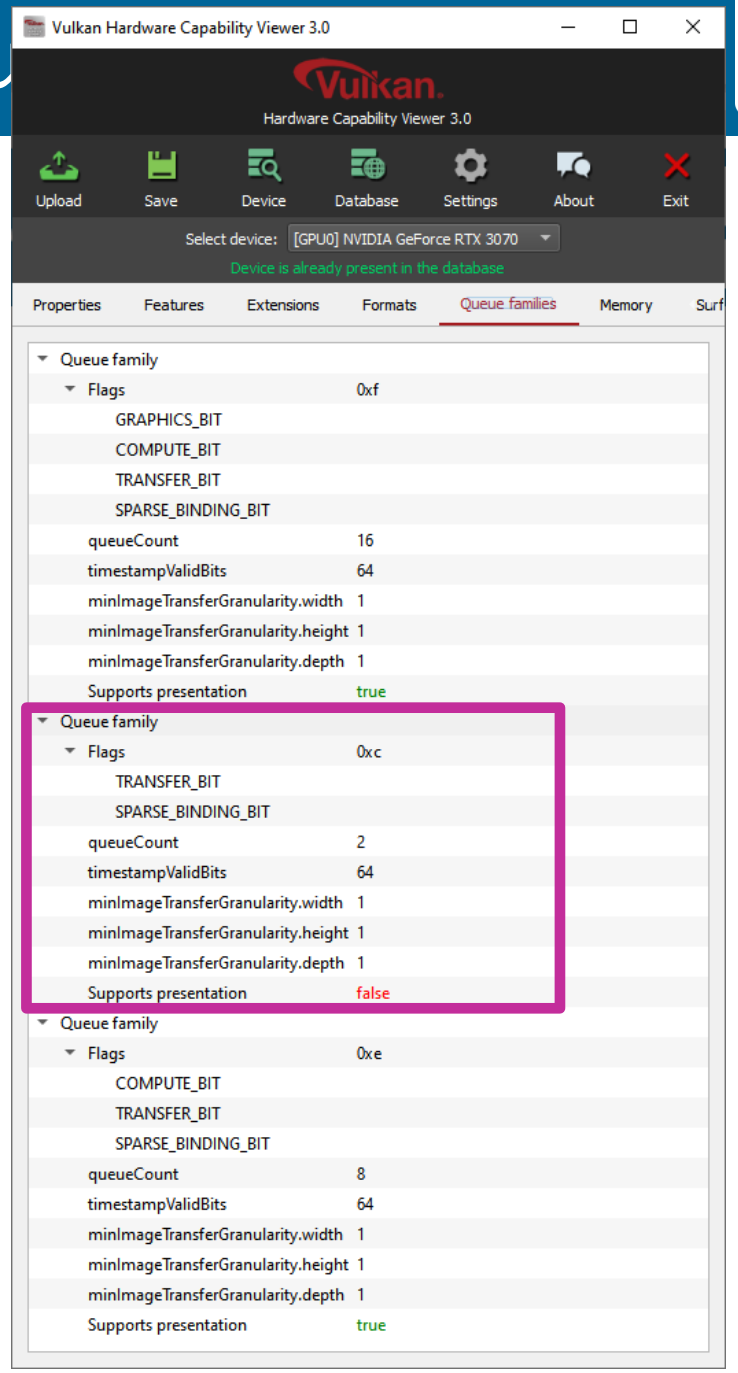
- A **queue** always belongs to a **queue family**.
- Queue families
  - A physical device **can** support different queue families … or only one.
  - Different queue families have different properties.
  - Multiple queues of the same queue family **can** be created and used.
- Why use multiple queues?
  - Increase concurrency
  - (Potentially) increase performance with specialized queues:
    - e.g., a “transfer queue”
    - e.g., an “async compute queue”



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems

Vulkan Essentials: Queu

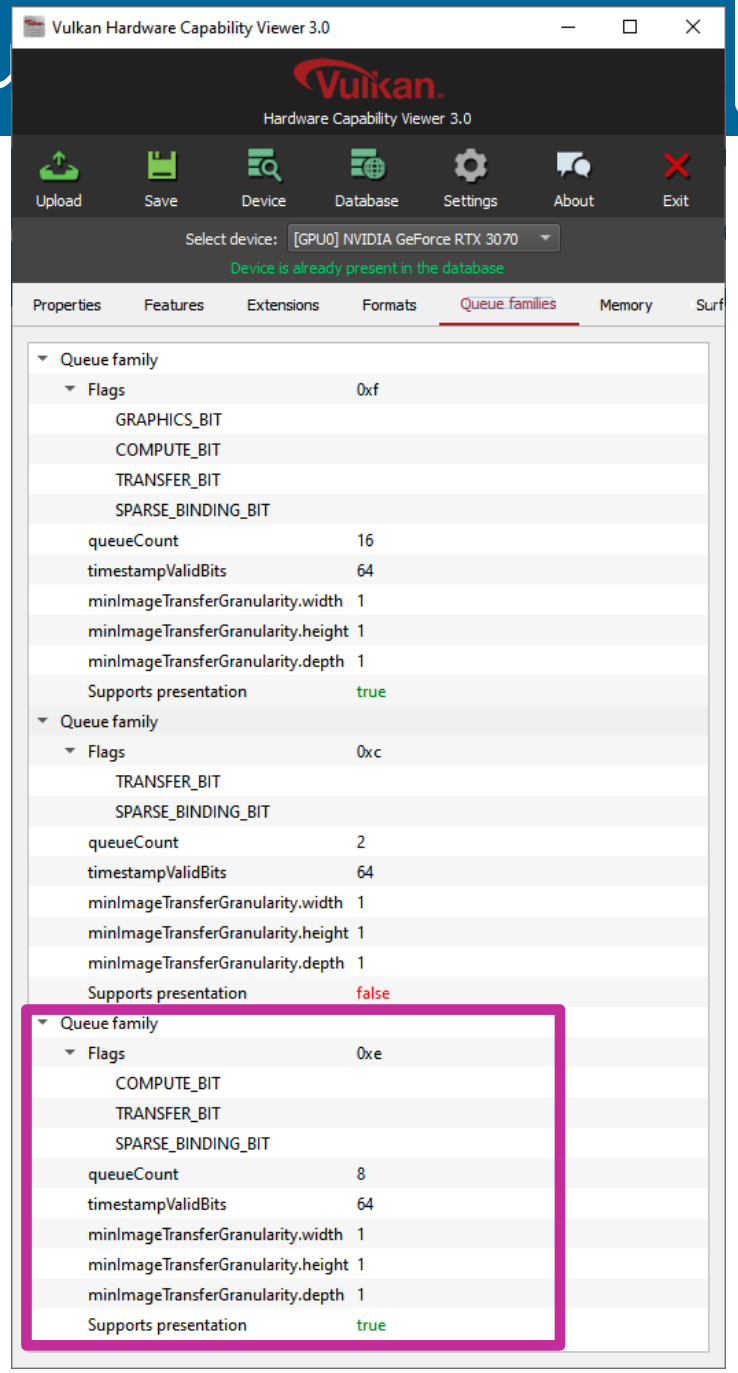
- A **queue** always belongs to a **queue family**.
- Queue families
  - A physical device **can** support different queue families … or only one.
  - Different queue families have different properties.
  - Multiple queues of the same queue family **can** be created and used.
- Why use multiple queues?
  - Increase concurrency
  - (Potentially) increase performance with specialized queues:
    - e.g., a "transfer queue"
    - e.g., an "async compute queue"



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems

Vulkan Essentials: Queu

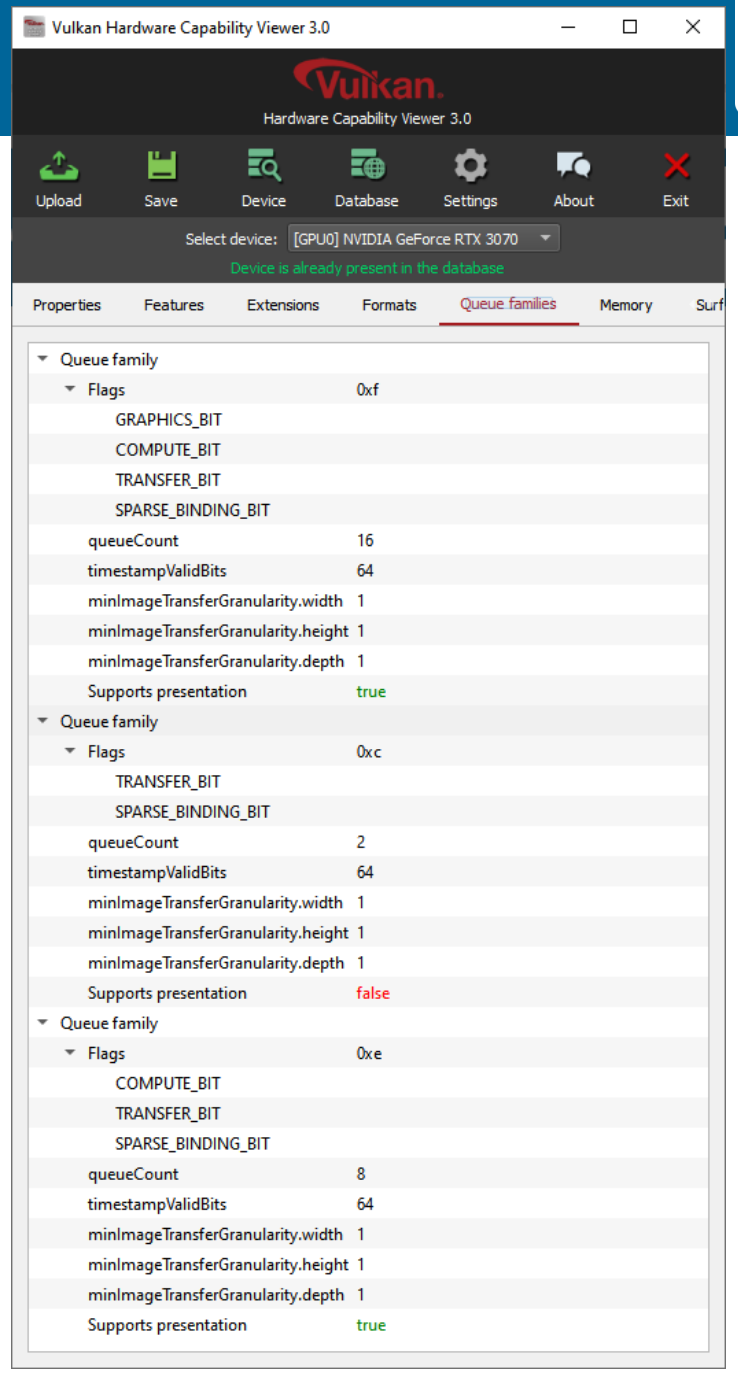
- A **queue** always belongs to a **queue family**.
- Queue families
  - A physical device **can** support different queue families … or only one.
  - Different queue families have different properties.
  - Multiple queues of the same queue family **can** be created and used.
- Why use multiple queues?
  - Increase concurrency
  - (Potentially) increase performance with specialized queues:
    - e.g., a “transfer queue”
    - e.g., an “async compute queue”



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems

Tools

- Vulkan Hardware Capability Viewer:
  - https://www.saschawillems.de/creations/vulkan-hardware-capability-viewer
  - https://github.com/SaschaWillems/VulkanCapsViewer
- Data can be uploaded into the GPUinfo database
  - https://vulkan.gpuinfo.org
  - Browse the results!
  - Check if a GPU supports Vulkan, queue families, extensions, image formats, etc.
- Thank you, Sascha Willems! :)



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems

Vulkan Essentials: Outline

Which Kind of API Is It?

Fundamental API Usage

Validation

Instance, Physical Device, Logical Device

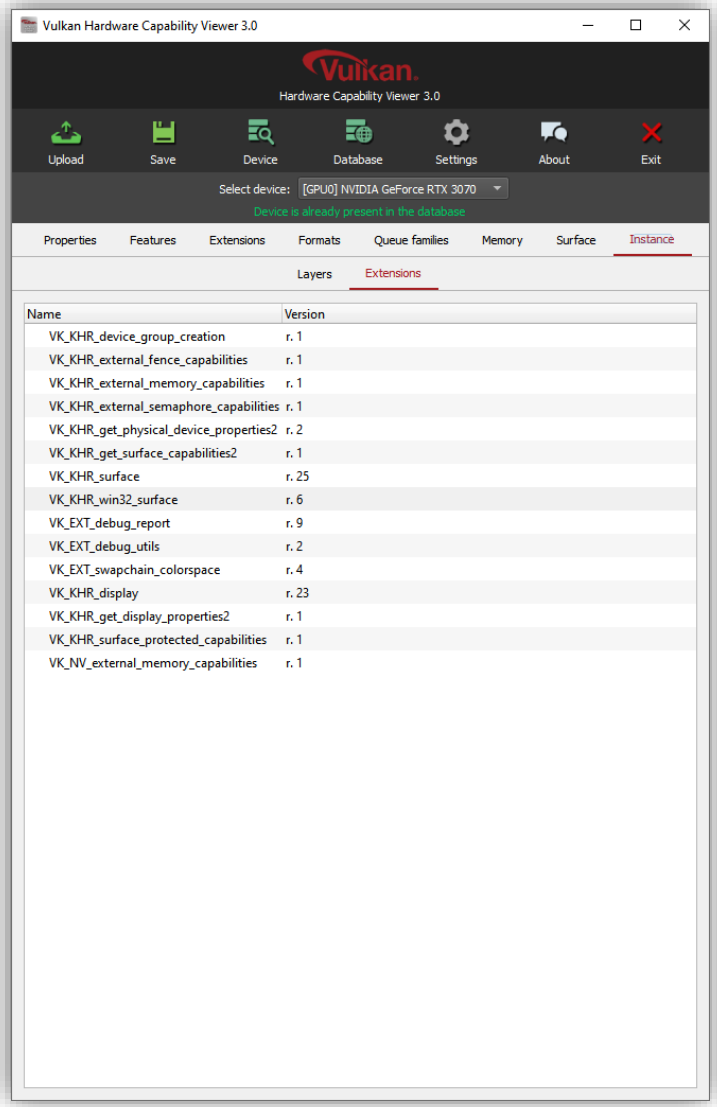
Queues

Extensions

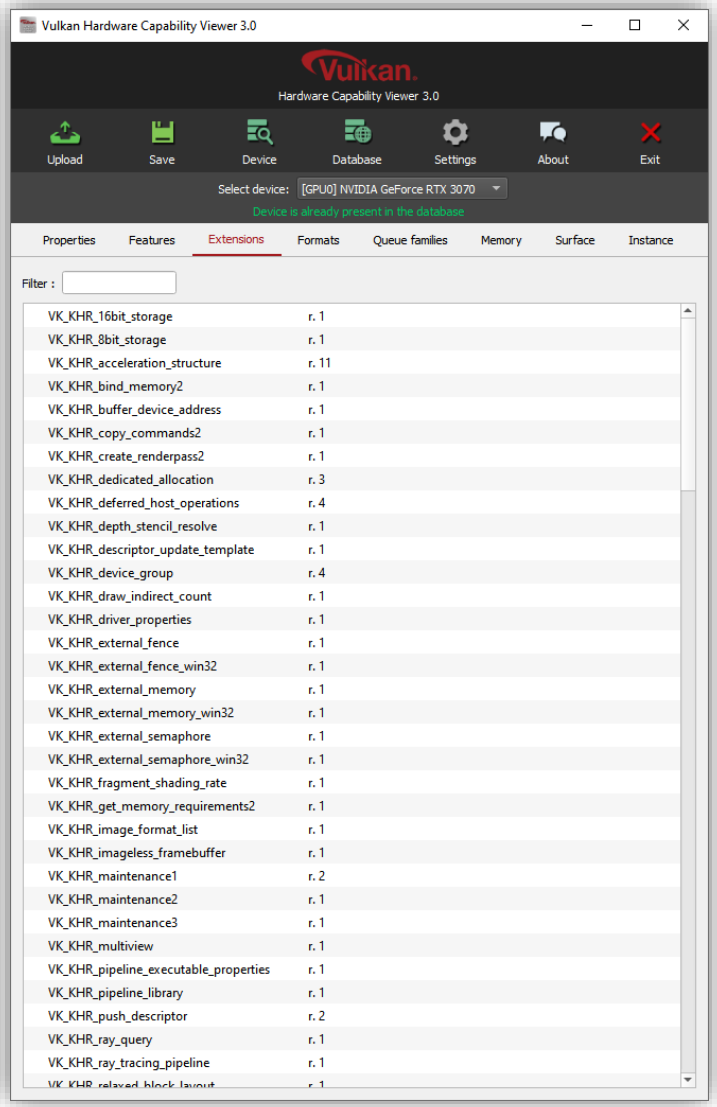
Vulkan Essentials: Extensions

Two extension types:

- Instance Extensions (left) **w.r.t the Vulkan installation**
  - Debug features
  - OS-specific features
  - Cross-device/instance/ process memory
- Device Extensions (right) **w.r.t. a specific device**
  - Capabilities of a physical device or driver



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems



Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems

Instance Extensions

Use `vkCreateDebugUtilsMessengerEXT` to install a callback, which gets invoked whenever the validation layers have something to report.

It is an extension, that's why it is suffixed with "EXT".

C Specification

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    const VkDebugUtilsMessengerCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugUtilsMessengerEXT*                   pMessenger);
```


Best way to approach this: Follow the official Khronos example Debug Utilities for the best practice-approach!

Instance Extensions

```
// During instance creation:

const char* instance_extensions[2] = {
    "VK_KHR_surface",
    "VK_KHR_win32_surface"
};

VkInstanceCreateInfo create_info = {};
// Set further create_info properties.
create_info.enabledExtensionCount = 2;
create_info.ppEnabledExtensionNames
    = instance_extensions;

// Use create_info with
// vkCreateInstance ...
```

Use `vkCreateDebugUtilsMessengerEXT` to install a callback, which gets invoked whenever the validation layers have something to report.

It is an extension, that's why it is suffixed with "EXT".

C Specification

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    const VkDebugUtilsMessengerCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugUtilsMessengerEXT*                   pMessenger);
```


Best way to approach this: Follow the official Khronos example Debug Utilities for the best practice-approach!

Instance Extensions

```
// During instance creation:

const char* instance_extensions[3] = {
    "VK_KHR_surface",
    "VK_KHR_win32_surface"
    ,"VK_EXT_debug_utils"
};

VkInstanceCreateInfo create_info = {};
// Set further create_info properties.
create_info.enabledExtensionCount = 3;
create_info.ppEnabledExtensionNames
    = instance_extensions;

// Use create_info with
// vkCreateInstance ...
```

Use `vkCreateDebugUtilsMessengerEXT` to install a callback, which gets invoked whenever the validation layers have something to report.

It is an extension, that's why it is suffixed with "EXT".

C Specification

A debug messenger triggers a debug callback with a debug message when an event of interest occurs. To create a debug messenger which will trigger a debug callback, call:

```
// Provided by VK_EXT_debug_utils
VkResult vkCreateDebugUtilsMessengerEXT(
    VkInstance                                  instance,
    const VkDebugUtilsMessengerCreateInfoEXT*   pCreateInfo,
    const VkAllocationCallbacks*                pAllocator,
    VkDebugUtilsMessengerEXT*                   pMessenger);
```


Best way to approach this: Follow the official Khronos example Debug Utilities for the best practice-approach!

Device Extensions

For example, how to enable real-time ray tracing?

- Its support depends on the GPU
  - Therefore, it must be a device extension.
- Which extension to use?
  - `"VK_NV_ray_tracing"` ?
  - `"VK_KHR_ray_tracing"` ?
  - `"VK_KHR_ray_tracing_pipeline"` ?
- Turns out that `"VK_KHR_ray_tracing_pipeline"` is the way to go, but has additional dependencies.
  - Check the specification!

`"VK_KHR_ray_tracing_pipeline"`
requires the following extensions in addition:

```
"VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing",
"VK_KHR_buffer_device_address",
"VK_KHR_deferred_host_operations"
```

Vulkan Essentials: Extensions

```
const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;
rt_pipe_features.pNext = &as_features;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;
rt_pipe_features.pNext = &as_features;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceVulkan12Features vulkan12_features = {};
vulkan12_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES;
vulkan12_features.descriptorIndexing = VK_TRUE;
vulkan12_features.bufferDeviceAddress = VK_TRUE;

VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;
rt_pipe_features.pNext = &as_features;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceVulkan12Features vulkan12_features = {};
vulkan12_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES;
vulkan12_features.descriptorIndexing = VK_TRUE;
vulkan12_features.bufferDeviceAddress = VK_TRUE;

VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;
rt_pipe_features.pNext = &as_features;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceVulkan12Features vulkan12_features = {};
vulkan12_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES;
vulkan12_features.descriptorIndexing = VK_TRUE;
vulkan12_features.bufferDeviceAddress = VK_TRUE;

VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;
rt_pipe_features.pNext = &as_features;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceVulkan12Features vulkan12_features = {};
vulkan12_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES;
vulkan12_features.descriptorIndexing = VK_TRUE;
vulkan12_features.bufferDeviceAddress = VK_TRUE;

VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;
as_features.pNext = &vulkan12_features;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;
rt_pipe_features.pNext = &as_features;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceVulkan12Features vulkan12_features = {};
vulkan12_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES;
vulkan12_features.descriptorIndexing = VK_TRUE;
vulkan12_features.bufferDeviceAddress = VK_TRUE;

VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;
as_features.pNext = &vulkan12_features;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;
rt_pipe_features.pNext = &as_features;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

```
VkPhysicalDeviceVulkan12Features vulkan12_features = {};
vulkan12_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_VULKAN_1_2_FEATURES;
vulkan12_features.descriptorIndexing = VK_TRUE;
vulkan12_features.bufferDeviceAddress = VK_TRUE;

VkPhysicalDeviceAccelerationStructureFeaturesKHR as_features = {};
as_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_ACCELERATION_STRUCTURE_FEATURES_KHR;
as_features.accelerationStructure = VK_TRUE;
as_features.pNext = &vulkan12_features;

VkPhysicalDeviceRayTracingPipelineFeaturesKHR rt_pipe_features = {};
rt_pipe_features.sType = VK_STRUCTURE_TYPE_PHYSICAL_DEVICE_RAY_TRACING_PIPELINE_FEATURES_KHR;
rt_pipe_features.rayTracingPipeline = VK_TRUE;
rt_pipe_features.pNext = &as_features;

const char* enabled_extensions[5]   = { "VK_KHR_ray_tracing_pipeline", "VK_KHR_acceleration_structure",
"VK_EXT_descriptor_indexing", "VK_KHR_buffer_device_address", "VK_KHR_deferred_host_operations" };
VkDeviceCreateInfo create_info      = {};
create_info.sType                   = VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO;
create_info.enabledExtensionCount   = 5;
create_info.ppEnabledExtensionNames = enabled_extensions;
create_info.pNext                   = &rt_pipe_features;

// Use create_info with vkCreateDevice ...
```

Vulkan Essentials: Extensions

- Two different types: instance-level, and (physical) device-level extensions
- Query if an extension is supported
  - For instance extensions: `vkEnumerateInstanceExtensionProperties`
  - For device extensions: `vkEnumerateDeviceExtensionProperties`
- Sometimes, a `pNext` chain can be necessary for configuration.
- Different types of extensions:
  - Khronos extensions: `VK_KHR_*`
  - Vendor-specific extensions: `VK_AMD_*`, `VK_INTEL_*`, `VK_NV_*`, …
  - Multivendor extensions: `VK_EXT_*`
- Use predefined macros instead of extension names directly:
  `VK_KHR_RAY_TRACING_PIPELINE_EXTENSION_NAME` instead of `"VK_KHR_ray_tracing_pipeline"`

Vulkan Essentials: Outline

Which Kind of API Is It?

Fundamental API Usage

Validation

Instance, Physical Device, Logical Device

Queues

Extensions

Introduction to Computer Graphics

186.832, 2021W, 3.0 ECTS

Thank you for your attention!

Johannes Unterguggenberger

Institute of Visual Computing & Human-Centered Technology

TU Wien, Austria