

An Introduction to Vulkan

Johannes Unterguggenberger
TU Wien, Huawei



Platinum Sponsors:



Vulkanised 2023

The 5th Vulkan Developer Conference
Munich, Germany / February 7–9



**Elias
Kristmann**
TU Wien



**Andreas
Wiesinger**
TU Wien



**Viktoria
Pogrzebacz**
TU Wien



**Annalena
Ulschmid**
TU Wien



**Lukas
Lipp**
TU Wien



**Johannes
Unterguggenberger**
TU Wien, Huawei

Platinum Sponsors:



Schedule

PART 1:

Setup
10 min
Starts at
09:00

Lecture
20 min
Starts at
09:10

Coding Session
90 min
Starts at
09:30



PART 2:

Lecture
15 min
Starts at
11:00

Coffee Break
25 min
Starts at
11:15

Coding Session
80 min
Starts at
11:40



Lunch Break 13:00 – 14:00

PART 3:

Lecture
15 min
Starts at
14:00

Coding Session
65 min
Starts at
14:15

Coffee Break
30 min
Starts at
15:20



PART 4:

Lecture
20 min
Starts at
15:50

Coding Session
70 min
Starts at
16:10

Closing
10 min
Starts at
17:20



Schedule

PART 1:

Setup
10 min
Starts at
09:00

Lecture
20 min
Starts at
09:10

Coding Session
90 min
Starts at
09:30

PART 2:

Lecture
15 min
Starts at
11:00

Coffee Break
25 min
Starts at
11:15

Coding Session
80 min
Starts at
11:40

Lunch Break 13:00 – 14:00

PART 3:

Lecture
15 min
Starts at
14:00

Coding Session
65 min
Starts at
14:15

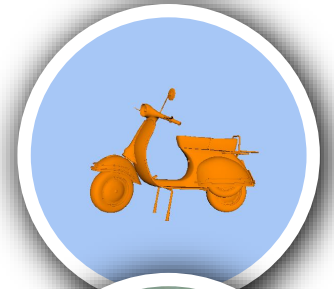
Coffee Break
30 min
Starts at
15:20

PART 4:

Lecture
20 min
Starts at
15:50

Coding Session
70 min
Starts at
16:10

Closing
10 min
Starts at
17:20



Schedule

PART 1:

Setup
10 min
Starts at
09:00

Lecture
20 min
Starts at
09:10

Coding Session
90 min
Starts at
09:30

PART 2:

Lecture
15 min
Starts at
11:00

Coffee Break
25 min
Starts at
11:15

Coding Session
80 min
Starts at
11:40

Lunch Break 13:00 – 14:00

PART 3:

Lecture
15 min
Starts at
14:00

Coding Session
65 min
Starts at
14:15

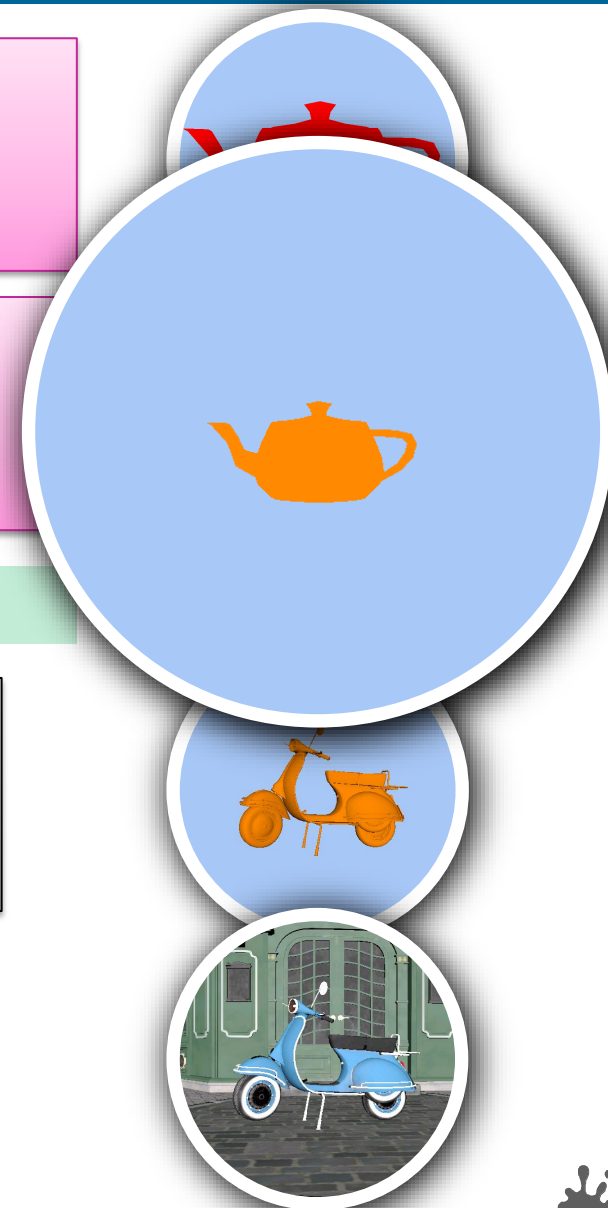
Coffee Break
30 min
Starts at
15:20

PART 4:

Lecture
20 min
Starts at
15:50

Coding Session
70 min
Starts at
16:10

Closing
10 min
Starts at
17:20



Schedule

PART 1:

Setup
10 min
Starts at
09:00

Lecture
20 min
Starts at
09:10

Coding Session
90 min
Starts at
09:30

PART 2:

Lecture
15 min
Starts at
11:00

Coffee Break
25 min
Starts at
11:15

Coding Session
80 min
Starts at
11:40

Lunch Break 13:00 – 14:00

PART 3:

Lecture
15 min
Starts at
14:00

Coding Session
65 min
Starts at
14:15

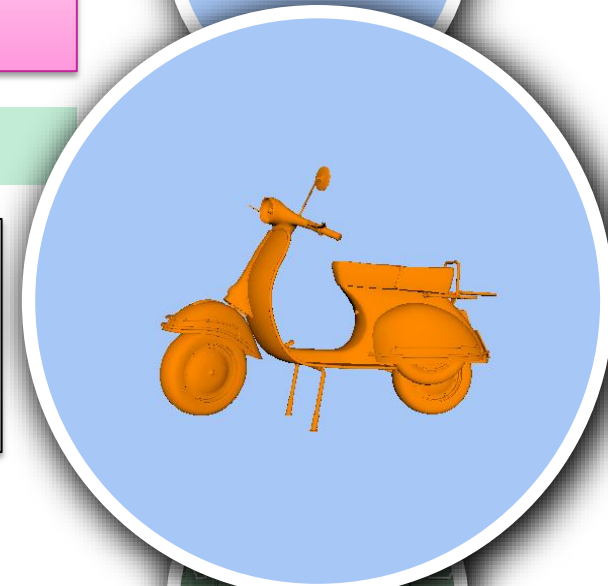
Coffee Break
30 min
Starts at
15:20

PART 4:

Lecture
20 min
Starts at
15:50

Coding Session
70 min
Starts at
16:10

Closing
10 min
Starts at
17:20



Schedule

PART 1:

Setup
10 min
Starts at
09:00

Lecture
20 min
Starts at
09:10

Coding Session
90 min
Starts at
09:30



PART 2:

Lecture
15 min
Starts at
11:00

Coffee Break
25 min
Starts at
11:15

Coding Session
80 min
Starts at
11:40



Lunch Break 13:00 – 14:00

PART 3:

Lecture
15 min
Starts at
14:00

Coding Session
65 min
Starts at
14:15

Coffee Break
30 min
Starts at
15:20

PART 4:

Lecture
20 min
Starts at
15:50

Coding Session
70 min
Starts at
16:10

Closing
10 min
Starts at
17:20



Schedule

PART 1:

Setup
10 min
Starts at
09:00

Lecture
20 min
Starts at
09:10

Coding Session
90 min
Starts at
09:30



PART 2:

Lecture
15 min
Starts at
11:00

Coffee Break
25 min
Starts at
11:15

Coding Session
80 min
Starts at
11:40



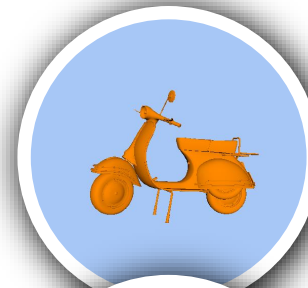
Lunch Break 13:00 – 14:00

PART 3:

Lecture
15 min
Starts at
14:00

Coding Session
65 min
Starts at
14:15

Coffee Break
30 min
Starts at
15:20



PART 4:

Lecture
20 min
Starts at
15:50

Coding Session
70 min
Starts at
16:10

Closing
10 min
Starts at
17:20



During each coding session...

Coding Session

90 min

Starts at
09:30

...you have two options:

1) Code yourself!
(recommended!)

+ assistance
from our team

2) Watch live coding



Google Meet

Code: **obz-chhh-npo**



Setup Instructions

STEP 1

Use our [VulkanLaunchpadStarter](#) template to create a **private(!)** project on **your** GitHub account

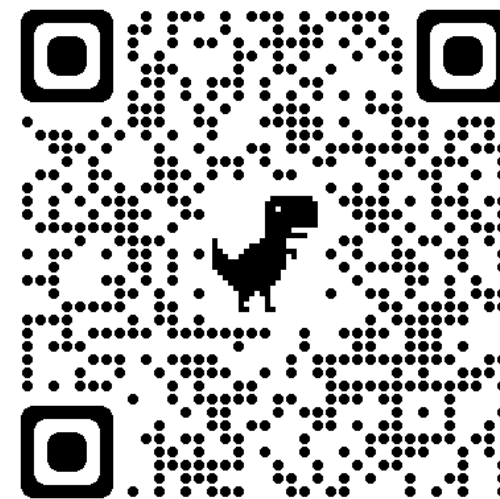


STEP 2

Join our "**Vulkanised**" Discord server:
<https://discord.gg/2Jfk6FjR>

STEP 3

Request access to [AnIntroductionToVulkan](#)
(via Discord, or personally)
It contains the **tasks descriptions** for the four parts.



PART 1

- Fundamental Vulkan Handles
- Window System Integration
- The Swap Chain



Platinum Sponsors:



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



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

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

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);
```



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



Vulkan Application

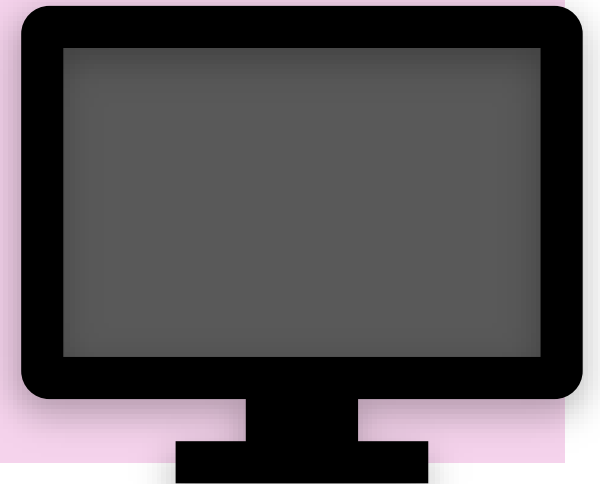
```
while (true) {  
    draw();  
}
```



Image 1

Operating System

```
present();
```



Window Surface

Vulkan Application

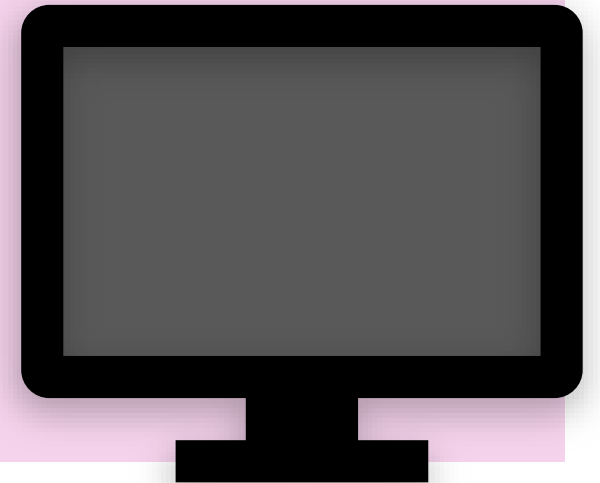
```
while (true) {  
    draw();  
}
```



Crytek Sponza, [CC BY 3.0](https://creativecommons.org/licenses/by/3.0/), © 2010 Frank Meinel, Crytek

Operating System

```
present();
```

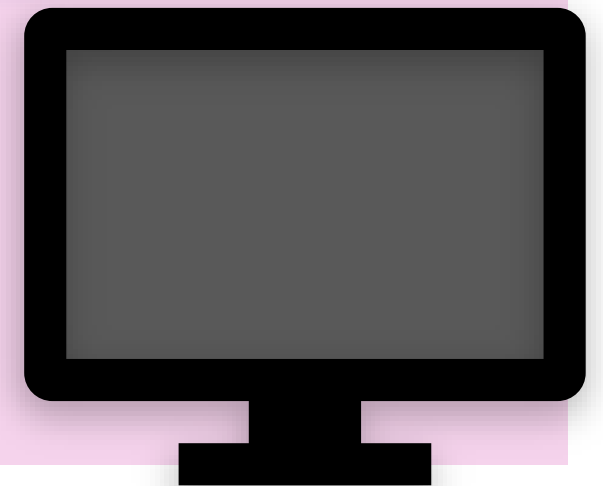


Vulkan Application

```
while (true) {  
    draw();  
}
```

Operating System

present();



Crytek Sponza, [CC BY 3.0](https://creativecommons.org/licenses/by/3.0/), © 2010 Frank Meinel, Crytek



Vulkan Application

```
while (true) {  
    draw();  
}
```

Operating System

present();

VkSurfaceKHR



Crytek Sponza, [CC BY 3.0](https://creativecommons.org/licenses/by/3.0/), © 2010 Frank Meinel, Crytek



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



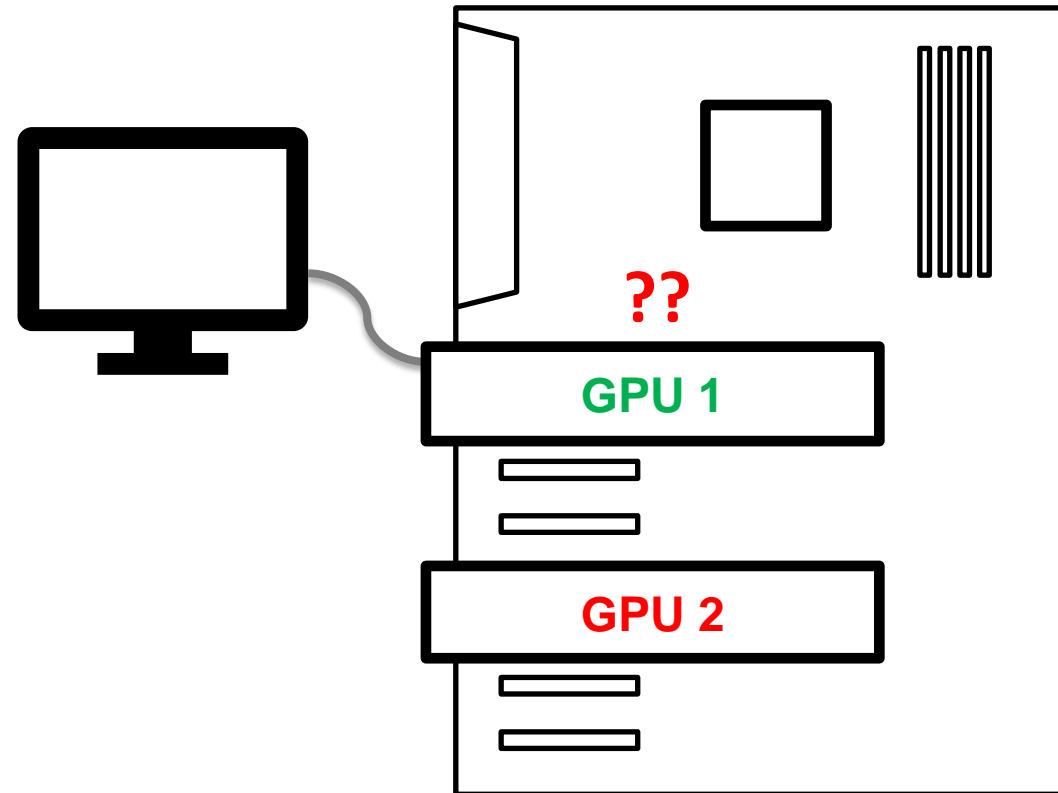
We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



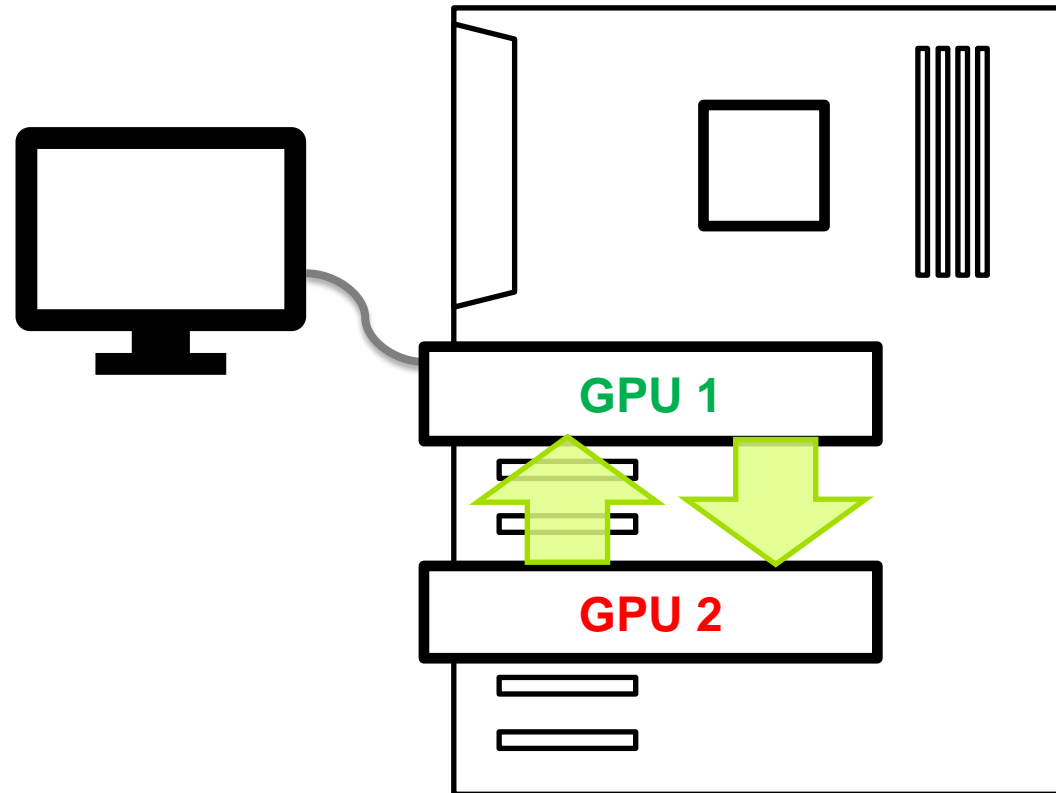
Physical Device Selection

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



Physical Device Selection

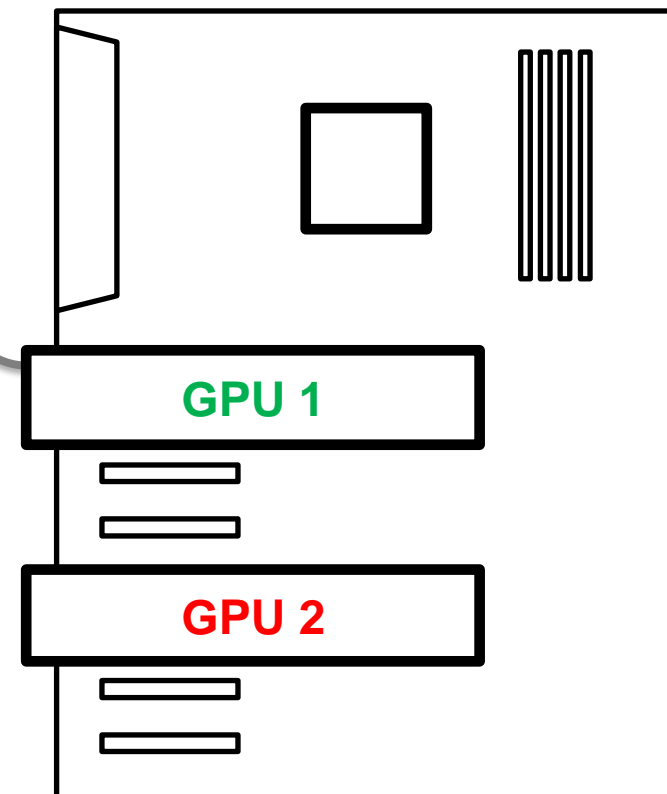
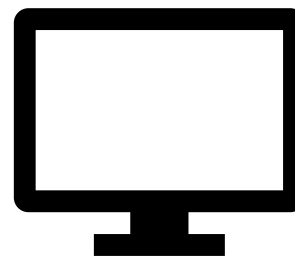
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.



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];
```

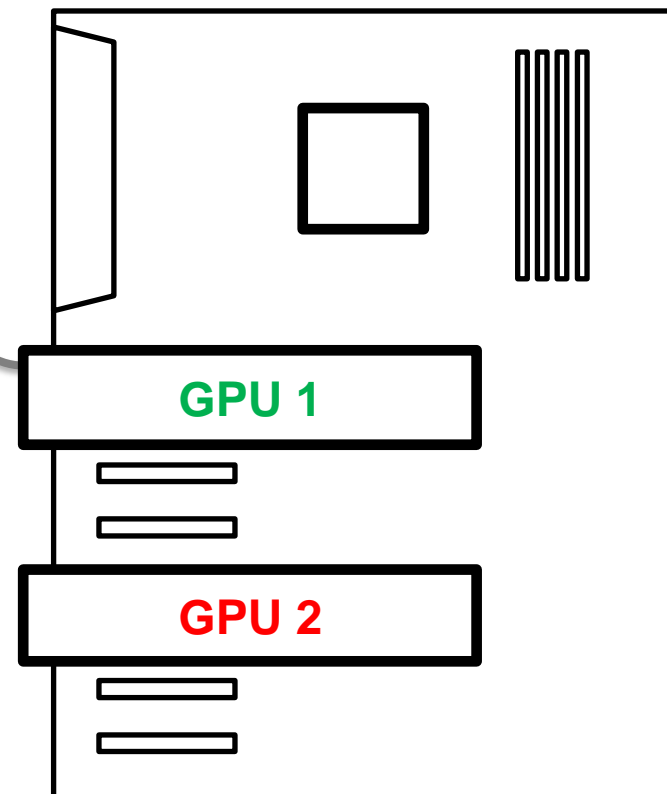
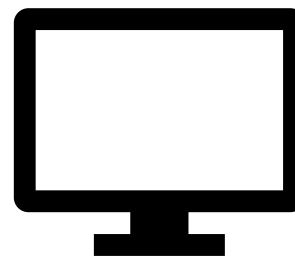
```
vkEnumeratePhysicalDevices(instance, &count, physical_devices);
```

```
// Select a physical device:
```

```
VkPhysicalDevice physical_device = physical_devices[0];
```

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

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

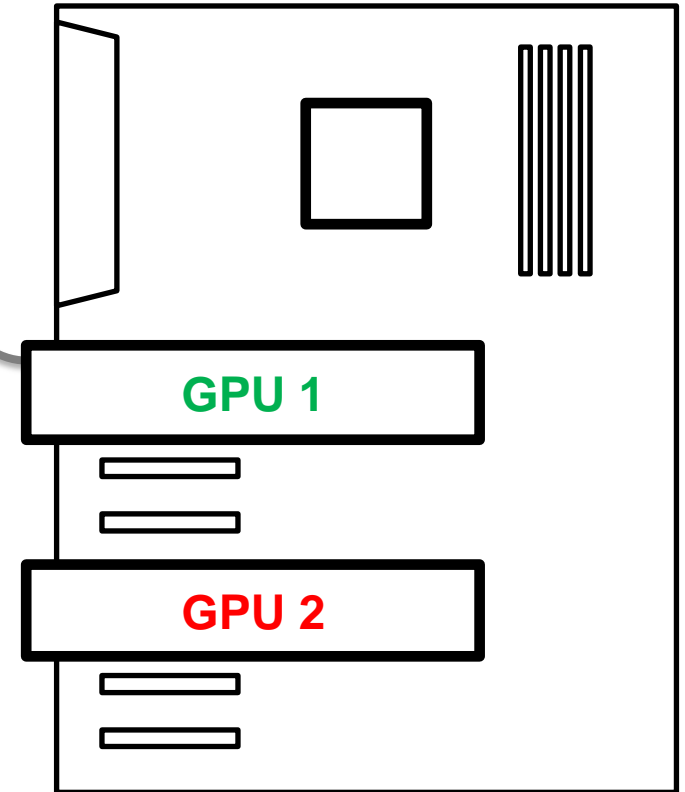


Physical Device

Physical Device Selection

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

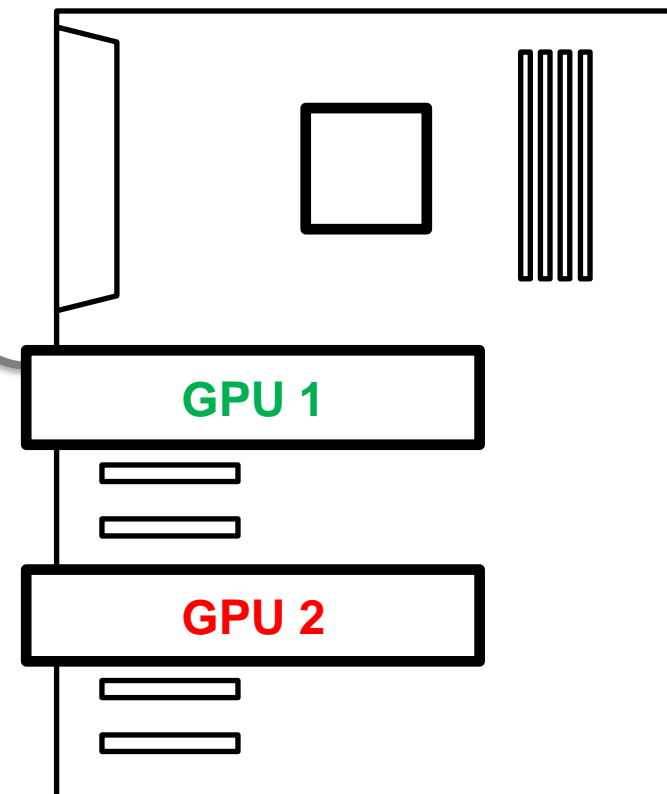
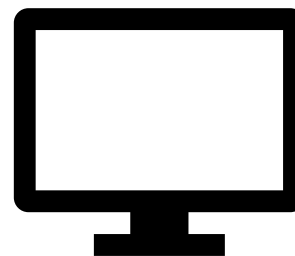
```
// Query the number of physical devices:  
uint32_t count;  
vkEnumeratePhysicalDevices(instance, &count, nullptr);  
assert(count > 0);  
  
// Get all physical device handles:  
VkPhysicalDevice* physical_devices = new VkPhysicalDevice[count];  
vkEnumeratePhysicalDevices(instance, &count, physical_devices);  
  
// 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

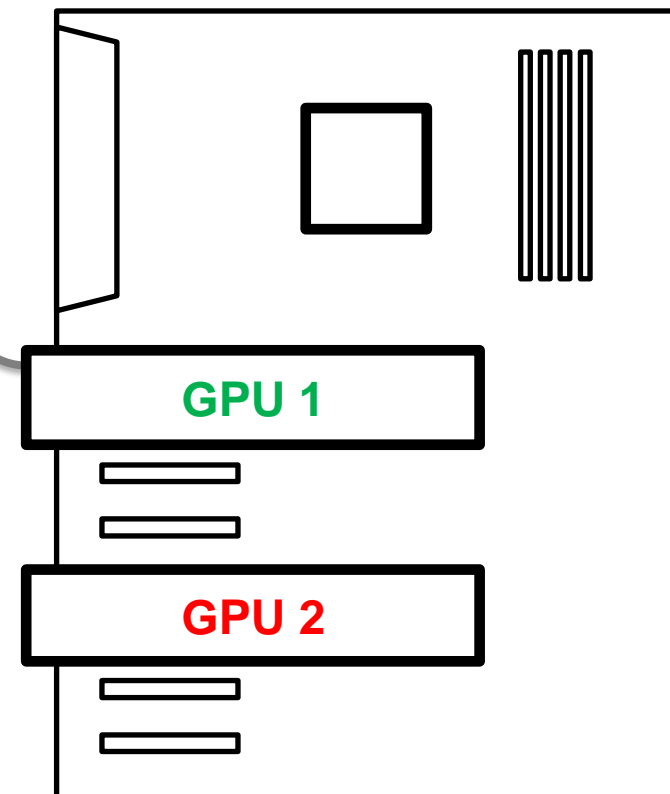
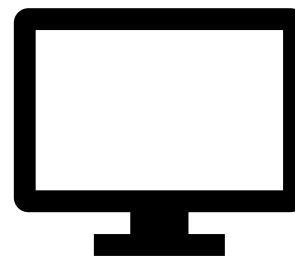
The (expressive and verbose) Vulkan way:
Explicitly express which GPU does what.

```
// 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, ...);
```



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



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

QUEUE



CMD 1



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

QUEUE

CMD 1

CMD 2



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

QUEUE

CMD 1

CMD 2

CMD 3



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);
```



- 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”



QUEUE 1

CMD 1

CMD 2

QUEUE 2

CMD 3

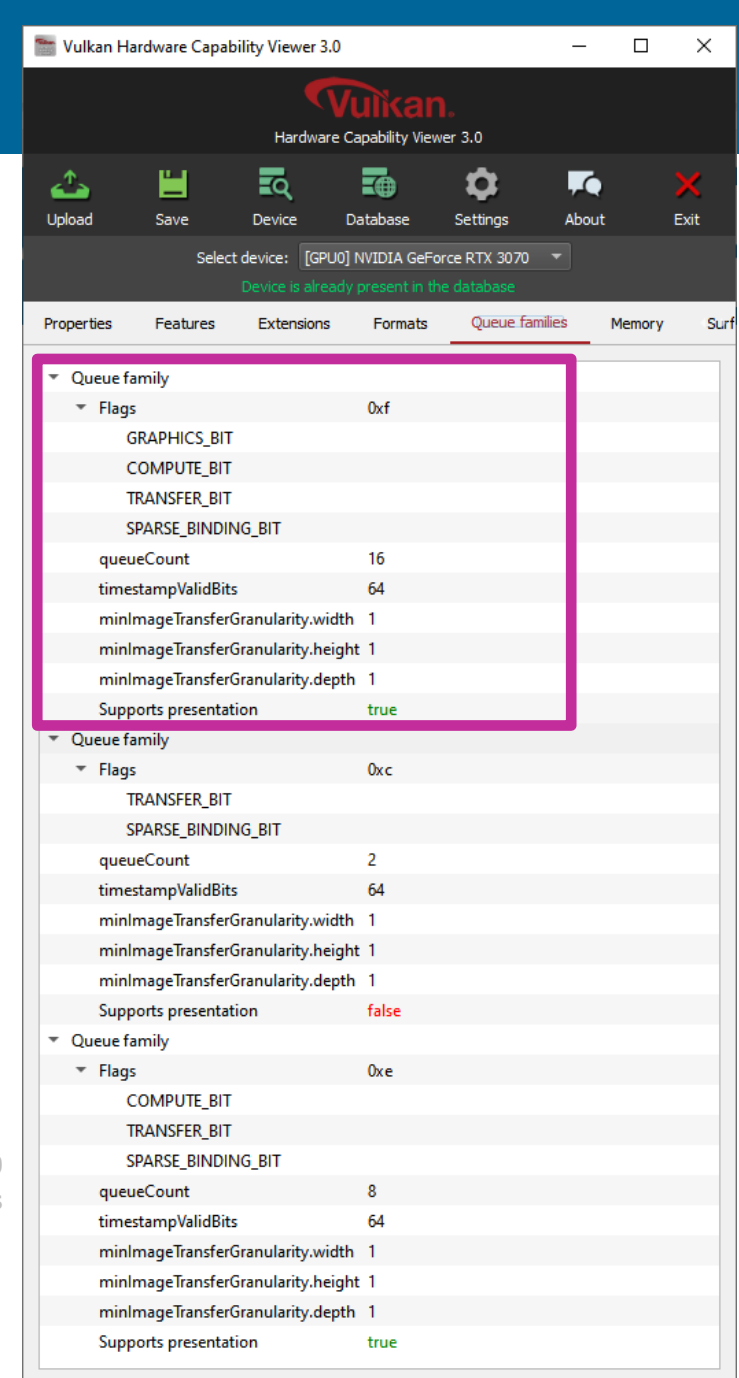
CMD 4



Queues

- 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”

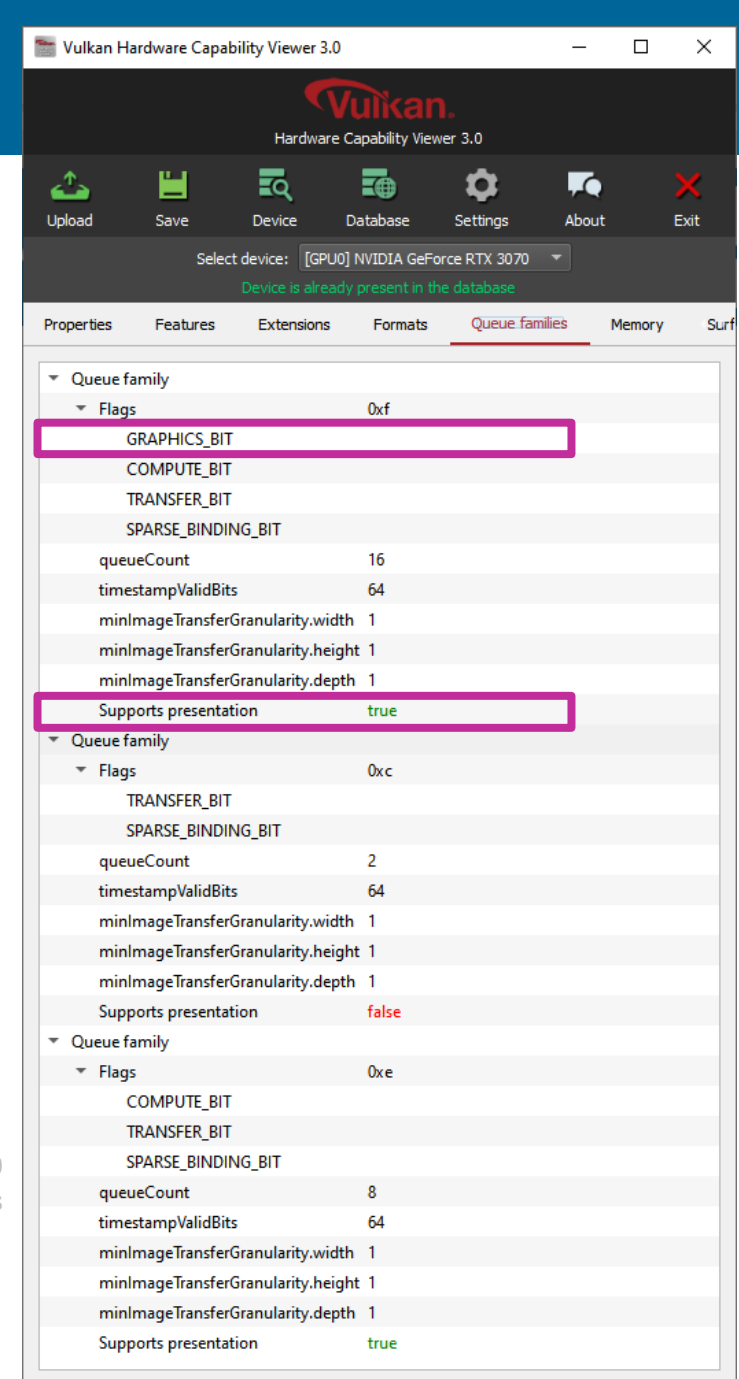
Vulkan Hardware Capability Viewer 3.0
© 2016-2020 by Sascha Willems



Queues

- 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



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



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);
```



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



The Swap Chain

Application/Render Loop

```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```

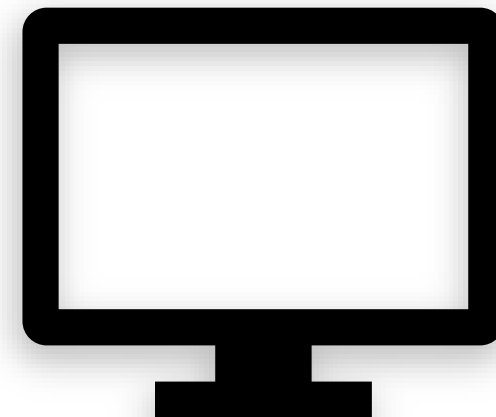
Swap Chain

available images:

Image 1

Image 2

presented image:



The Swap Chain

Application/Render Loop

```
while (true) {
```

```
    acquireNextImage();
```

```
    draw();
```

```
    present();
```

```
}
```

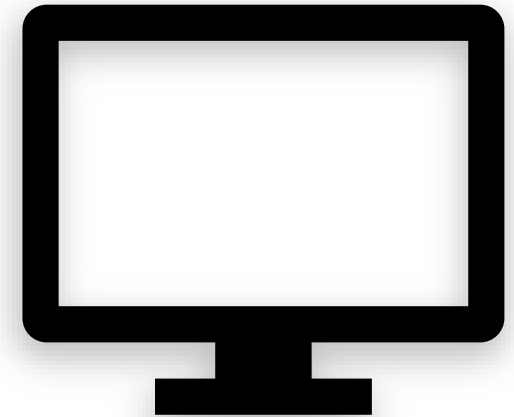
Image 1
Current **Backbuffer**

Swap Chain

available images:

Image 2

presented image:



The Swap Chain

Application/Render Loop

```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```

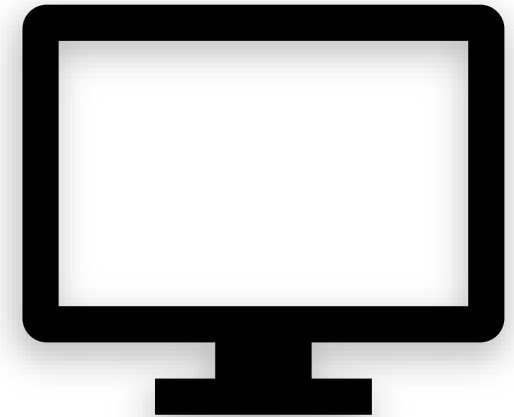
Image 1
Current **Backbuffer**

Swap Chain

available images:

Image 2

presented image:



The Swap Chain

Application/Render Loop

```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present(); Current Backbuffer  
}
```

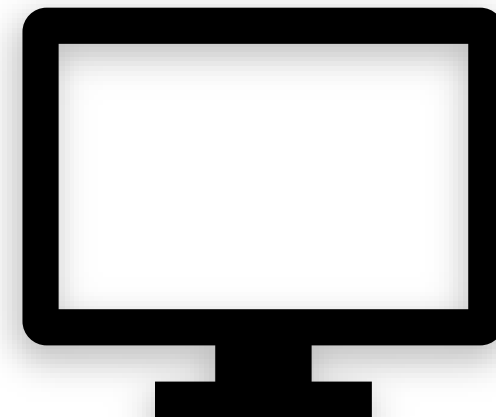


Swap Chain

available images:



presented image:



The Swap Chain

Application/Render Loop

```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```

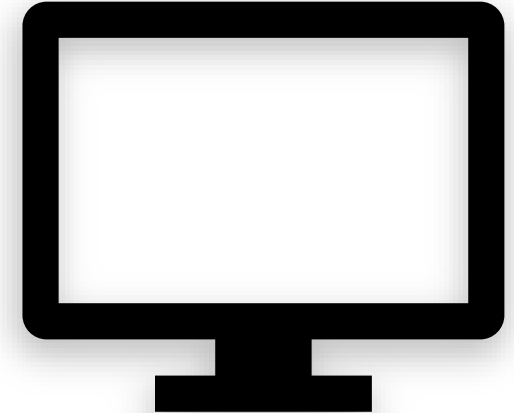


Swap Chain

available images:



presented image:



Crytek Sponza, [CC BY 3.0](#), © 2010 Frank Meinl, Crytek



The Swap Chain

Application/Render Loop

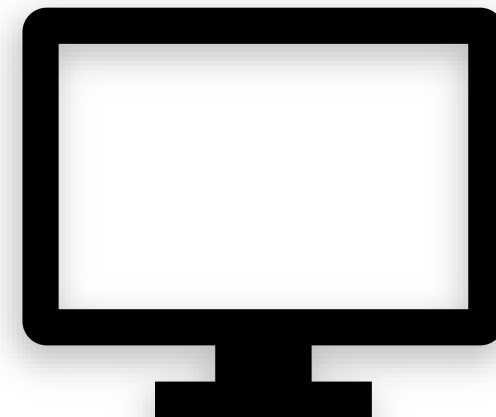
```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```

Swap Chain

available images:



presented image:



Crytek Sponza, CC BY 3.0, © 2010 Frank Meinel, Crytek



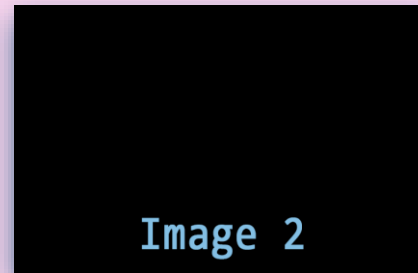
The Swap Chain

Application/Render Loop

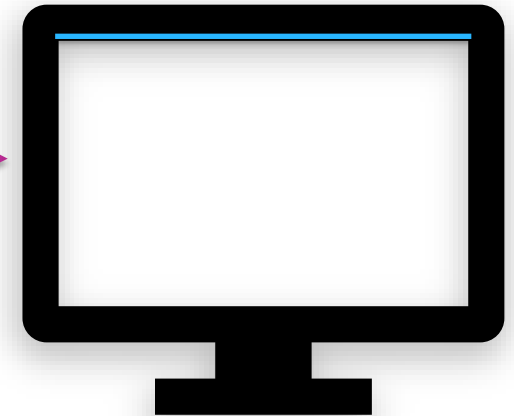
```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```

Swap Chain

available images:



presented image:



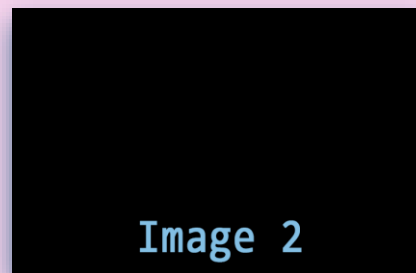
The Swap Chain

Application/Render Loop

```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```

Swap Chain

available images:



presented image:

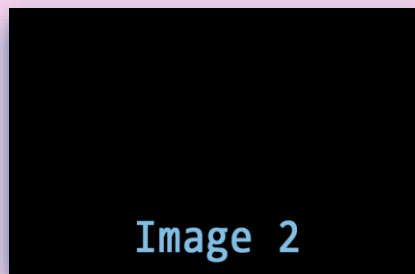


Application/Render Loop

```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```

Swap Chain

available images:



presented image:



The Swap Chain

Application/Render Loop

```
while (true) {
```

```
    acquireNextImage();
```

```
    draw();
```

```
    present();
```

```
}
```



Swap Chain

available images:

presented image:



Crytek Sponza, [CC BY 3.0](#), © 2010 Frank Meinl, Crytek



The Swap Chain

Application/Render Loop

```
while (true) {  
    acquireNextImage();
```

```
    draw();
```



```
    present(); Current Backbuffer
```

```
}
```

Swap Chain

available images:



presented image:



The Swap Chain

Application/Render Loop

```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```



Swap Chain

available images:



presented image:



Crytek Sponza, [CC BY 3.0](#), © 2010 Frank Meinl, Crytek



The Swap Chain

Application/Render Loop

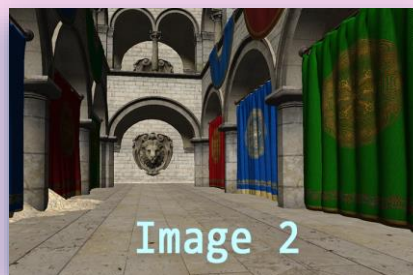
```
while (true) {  
    acquireNextImage();  
  
    draw();  
  
    present();  
}
```



Swap Chain

available images:

presented image:



Crytek Sponza, CC BY 3.0, © 2010 Frank Meinel, Crytek



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



We need to create/get hold of a couple of handles:

Instance	Vulkan on your system	VkInstance
Window Surface	A window of your OS	VkSurfaceKHR
Physical Device	A hardware device (GPU)	VkPhysicalDevice
Queue	Received commands to be executed on a physical device	VkQueue
Logical Device	Main interface to a physical device (active configuration)	VkDevice
Swap Chain	Sends images to a monitor, Provides images to render into	VkSwapchainKHR



Validation OFF

```
// Create  
VkImage  
create
```

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

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>)

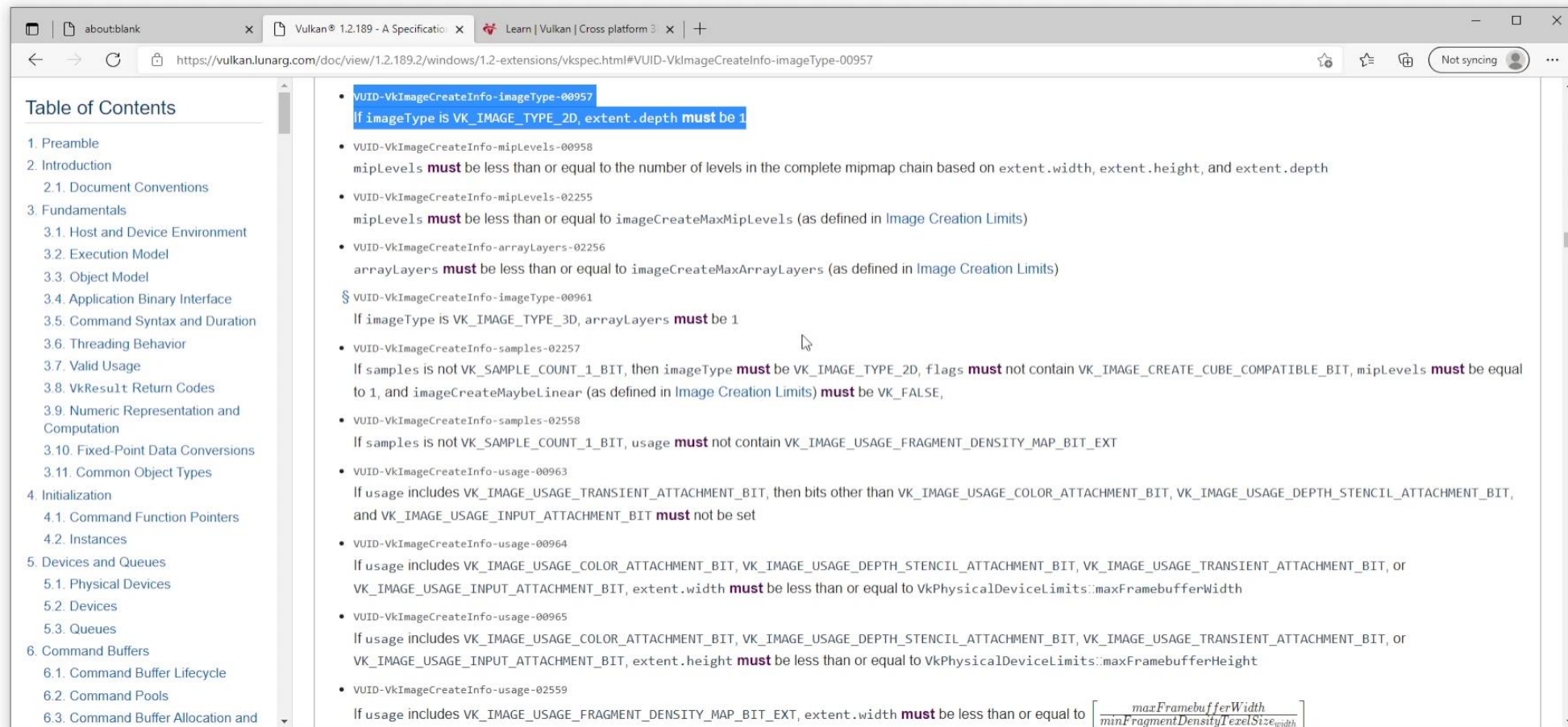
```
VkResult  
CHECK_
```



The Vulkan Specification is your best friend!

```
// Create VkImage  
create  
create  
create  
create  
create  
create  
create
```

```
VkImage  
VkResult  
CHECK_
```



The screenshot shows a web browser window displaying the Vulkan Specification. The address bar shows the URL: <https://vulkan.lunarg.com/doc/view/1.2.189.2/windows/1.2-extensions/vkspec.html#VUID-VkImageCreateInfo-imageType-00957>. The page has a "Table of Contents" on the left and a list of VUIDs on the right. The VUIDs are listed with their corresponding requirements.

Table of Contents

1. Preamble
2. Introduction
 - 2.1. Document Conventions
3. Fundamentals
 - 3.1. Host and Device Environment
 - 3.2. Execution Model
 - 3.3. Object Model
 - 3.4. Application Binary Interface
 - 3.5. Command Syntax and Duration
 - 3.6. Threading Behavior
 - 3.7. Valid Usage
 - 3.8. VkResult Return Codes
 - 3.9. Numeric Representation and Computation
 - 3.10. Fixed-Point Data Conversions
 - 3.11. Common Object Types
4. Initialization
 - 4.1. Command Function Pointers
 - 4.2. Instances
5. Devices and Queues
 - 5.1. Physical Devices
 - 5.2. Devices
 - 5.3. Queues
6. Command Buffers
 - 6.1. Command Buffer Lifecycle
 - 6.2. Command Pools
 - 6.3. Command Buffer Allocation and

VUIDs:

- **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 $\left\lfloor \frac{\text{maxFramebufferWidth}}{\text{minFragmentDensityTexelSize}_{\text{width}}} \right\rfloor$



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_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;

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



PART 1

- Fundamental Vulkan Handles
- Window System Integration
- The Swap Chain

GOOD LUCK!



Platinum Sponsors:



Schedule

PART 1:

Setup
10 min
Starts at
09:00

Lecture
20 min
Starts at
09:10

Coding Session
90 min
Starts at
09:30



PART 2:

Lecture
15 min
Starts at
11:00

Coffee Break
25 min
Starts at
11:15

Coding Session
80 min
Starts at
11:40



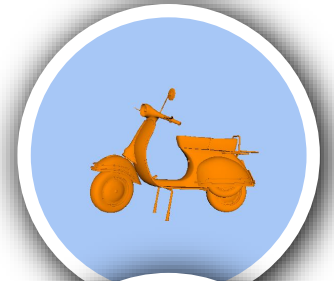
Lunch Break 13:00 – 14:00

PART 3:

Lecture
15 min
Starts at
14:00

Coding Session
65 min
Starts at
14:15

Coffee Break
30 min
Starts at
15:20



PART 4:

Lecture
20 min
Starts at
15:50

Coding Session
70 min
Starts at
16:10

Closing
10 min
Starts at
17:20

