An Introduction to Vulkan

Johannes Unterguggenberger
TU Wien, Huawei
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
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: | | PART 2: | | PART 3: | | PART 4: |
| --- | --- | --- | --- | --- | --- |
| **Setup** | **Lecture** | **Coding Session** | **Lecture** | **Coffee Break** | **Coding Session** | **Coffee Break** | **Lecture** | **Coding Session** | **Coffee Break** | **Closing** |
| 10 min | 20 min | 90 min | 15 min | 25 min | 80 min | 15 min | 65 min | 30 min | 70 min | 10 min |
| Starts at 09:00 | Starts at 09:10 | Starts at 09:30 | Starts at 11:00 | Starts at 11:15 | Starts at 11:40 | Starts at 14:00 | Starts at 14:15 | Starts at 15:20 | Starts at 15:50 | Starts at 16:10 | Starts at 17:20 |

**Lunch Break** 13:00 – 14:00
## Schedule

### PART 1:
- **Setup**
  - Duration: 10 min
  - Starts at 09:00
- **Lecture**
  - Duration: 20 min
  - Starts at 09:10
- **Coding Session**
  - Duration: 90 min
  - Starts at 09:30

### PART 2:
- **Lecture**
  - Duration: 15 min
  - Starts at 11:00
- **Coffee Break**
  - Duration: 25 min
  - Starts at 11:15
- **Coding Session**
  - Duration: 80 min
  - Starts at 11:40

### Lunch Break
- **Duration**: 13:00 – 14:00

### PART 3:
- **Lecture**
  - Duration: 15 min
  - Starts at 14:00
- **Coding Session**
  - Duration: 65 min
  - Starts at 14:15
- **Coffee Break**
  - Duration: 30 min
  - Starts at 15:20

### PART 4:
- **Lecture**
  - Duration: 20 min
  - Starts at 15:50
- **Coding Session**
  - Duration: 70 min
  - Starts at 16:10
- **Closing**
  - Duration: 10 min
  - Starts at 17:20
<table>
<thead>
<tr>
<th>PART 1</th>
<th>PART 2</th>
<th>PART 3</th>
<th>PART 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Setup</strong> 10 min</td>
<td><strong>Lecture</strong> 20 min</td>
<td><strong>Coding Session</strong> 90 min</td>
<td><strong>Closing</strong> 10 min</td>
</tr>
<tr>
<td>Starts at 09:00</td>
<td>Starts at 09:10</td>
<td>Starts at 09:30</td>
<td>Starts at 17:20</td>
</tr>
<tr>
<td><strong>Lecture</strong> 15 min</td>
<td><strong>Coffee Break</strong> 25 min</td>
<td><strong>Coding Session</strong> 80 min</td>
<td><strong>Coffee Break</strong> 30 min</td>
</tr>
<tr>
<td>Starts at 11:00</td>
<td>Starts at 11:15</td>
<td>Starts at 11:40</td>
<td>Starts at 15:20</td>
</tr>
<tr>
<td><strong>Lecture</strong> 15 min</td>
<td><strong>Coding Session</strong> 65 min</td>
<td><strong>Coffee Break</strong> 30 min</td>
<td><strong>Closing</strong> 10 min</td>
</tr>
<tr>
<td>Starts at 14:00</td>
<td>Starts at 14:15</td>
<td>Starts at 15:20</td>
<td>Starts at 17:20</td>
</tr>
<tr>
<td><strong>Coding Session</strong> 70 min</td>
<td><strong>Coffee Break</strong> 30 min</td>
<td><strong>Closing</strong> 10 min</td>
<td><strong>Closing</strong> 10 min</td>
</tr>
<tr>
<td>Starts at 16:10</td>
<td>Starts at 15:20</td>
<td>Starts at 17:20</td>
<td>Starts at 17:20</td>
</tr>
</tbody>
</table>

**Lunch Break** 13:00 – 14:00
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...

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

<table>
<thead>
<tr>
<th>Handle</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance</strong></td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td><strong>Window Surface</strong></td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td><strong>Physical Device</strong></td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td><strong>Queue</strong></td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td><strong>Logical Device</strong></td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td><strong>Swap Chain</strong></td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
Instance Creation

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

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

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

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

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

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**

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

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

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

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

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

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

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

class instance
{
    constant 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* 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

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

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

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

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

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

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:

<table>
<thead>
<tr>
<th><strong>Instance</strong></th>
<th>Vulkan on your system</th>
<th>** VkInstance **</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Window Surface</strong></td>
<td>A window of your OS</td>
<td>** VkSurfaceKHR **</td>
</tr>
<tr>
<td><strong>Physical Device</strong></td>
<td>A hardware device (GPU)</td>
<td>** VkPhysicalDevice **</td>
</tr>
<tr>
<td><strong>Queue</strong></td>
<td>Received commands to be executed on a physical device</td>
<td>** VkQueue **</td>
</tr>
<tr>
<td><strong>Logical Device</strong></td>
<td>Main interface to a physical device (active configuration)</td>
<td>** VkDevice **</td>
</tr>
<tr>
<td><strong>Swap Chain</strong></td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>** VkSwapchainKHR **</td>
</tr>
</tbody>
</table>
We need to create/get hold of a couple of handles:

<table>
<thead>
<tr>
<th>Instance</th>
<th>Vulkan on your system</th>
<th>VkInstance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Window Surface</strong></td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td>Physical Device</td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td>Queue</td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td>Logical Device</td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td>Swap Chain</td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
while (true) {
    draw();
}

Vulkan Application

Image 1

Operating System

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

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

present();
### Vulkan Application

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

### Operating System

```c
present();
```

**VkSurfaceKHR**

**Crytek Sponza, CC BY 3.0, © 2010 Frank Meinl, Crytek**
We need to create/get hold of a couple of handles:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Vulkan Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td>Window Surface</td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td>Physical Device</td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td>Queue</td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td>Logical Device</td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td>Swap Chain</td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
We need to create/get hold of a couple of handles:

<table>
<thead>
<tr>
<th>Handle</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td>Window Surface</td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td>Physical Device</td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td>Queue</td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td>Logical Device</td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td>Swap Chain</td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
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.
// 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.

```c
// 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, ...);
```
// 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.
Physical Device Selection

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

```c
// 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.

```c
// 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:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Vk Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td>Window Surface</td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td>Physical Device</td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td>Queue</td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td>Logical Device</td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td>Swap Chain</td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
We need to create/get hold of a couple of handles:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Vulkan Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td>Window Surface</td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td>Physical Device</td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td>Queue</td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td>Logical Device</td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td>Swap Chain</td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
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
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
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
**Logical Device Creation**

```c
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);
```
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);
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”
Queues

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

<table>
<thead>
<tr>
<th>We need to create/get hold of a couple of handles:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance</strong></td>
<td>Vulkan on your system</td>
</tr>
<tr>
<td><strong>Window Surface</strong></td>
<td>A window of your OS</td>
</tr>
<tr>
<td><strong>Physical Device</strong></td>
<td>A hardware device (GPU)</td>
</tr>
<tr>
<td><strong>Queue</strong></td>
<td>Received commands to be executed on a physical device</td>
</tr>
<tr>
<td><strong>Logical Device</strong></td>
<td>Main interface to a physical device (active configuration)</td>
</tr>
<tr>
<td><strong>Swap Chain</strong></td>
<td>Sends images to a monitor, Provides images to render into</td>
</tr>
</tbody>
</table>
### Overview

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

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>VK Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance</strong></td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td><strong>Window Surface</strong></td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td><strong>Physical Device</strong></td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td><strong>Queue</strong></td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td><strong>Logical Device</strong></td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td><strong>Swap Chain</strong></td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
Logical Device Creation

```c
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

```c
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

```c
float priority = 1.0f;
VkDeviceQueueCreateInfo queue_create_info = {};
queue_create_info.sType = VK_STRUCTURE_TYPEDEVICE_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_TYPEDEVICE_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

```c
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

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

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

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

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

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

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

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

<table>
<thead>
<tr>
<th>Handle Type</th>
<th>Description</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Instance</strong></td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td><strong>Window Surface</strong></td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td><strong>Physical Device</strong></td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td><strong>Queue</strong></td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td><strong>Logical Device</strong></td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td><strong>Swap Chain</strong></td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
The Swap Chain

Application/Render Loop

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

Swap Chain

- available images:
  - Image 1
  - Image 2

- presented image:
The Swap Chain

Application/Render Loop

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

Swap Chain

available images:

- Image 1
- Image 2

presented image:
while (true) {
    acquireNextImage();
    draw();
    present();
}

available images:
Image 1
Image 2

presented image:
The Swap Chain

Application/Render Loop

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

Swap Chain

available images:

- Image 2

presented image:

Crytek Sponza, CC BY 3.0, © 2010 Frank Meiml, 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 Meini, 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 Meinl, Crytek
The Swap Chain

Application/Render Loop

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

Swap Chain

available images:

Image 2

presented image:

Current Frontbuffer

Crytek Sponza, CC BY 3.0. © 2010 Frank Meinl, Crytek
The Swap Chain

**Application/Render Loop**

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

**Swap Chain**

- **Available images:**
  - Image 1
  - Image 2

- **Presented image:**
  - Current Frontbuffer

---

Crytek Sponza, CC BY 3.0 © 2010 Frank Meiml, Crytek
The Swap Chain

Application/Render Loop

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

Swap Chain

available images:

presented image:
The Swap Chain

Application/Render Loop

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

Swap Chain

available images:

- Image 1
  - Current Frontbuffer

- Image 2
  - Current Backbuffer

presented image:
The Swap Chain

Application/Render Loop

```java
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:

Image 1

Image 2

Crytek Sponza, CC BY 3.0. © 2010 Frank Meili, Crytek
The Swap Chain

Application/Render Loop

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

Swap Chain

available images:

- Image 1

presented image:

- Image 2

Crytek Sponza, CC BY 3.0, © 2010 Frank Meinl, Crytek
We need to create/get hold of a couple of handles:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Vulkan Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td>Window Surface</td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td>Physical Device</td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td>Queue</td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td>Logical Device</td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td>Swap Chain</td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
We need to create/get hold of a couple of handles:

<table>
<thead>
<tr>
<th>Handle Type</th>
<th>Description</th>
<th>Vk Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instance</td>
<td>Vulkan on your system</td>
<td>VkInstance</td>
</tr>
<tr>
<td>Window Surface</td>
<td>A window of your OS</td>
<td>VkSurfaceKHR</td>
</tr>
<tr>
<td>Physical Device</td>
<td>A hardware device (GPU)</td>
<td>VkPhysicalDevice</td>
</tr>
<tr>
<td>Queue</td>
<td>Received commands to be executed on a physical device</td>
<td>VkQueue</td>
</tr>
<tr>
<td>Logical Device</td>
<td>Main interface to a physical device (active configuration)</td>
<td>VkDevice</td>
</tr>
<tr>
<td>Swap Chain</td>
<td>Sends images to a monitor, Provides images to render into</td>
<td>VkSwapchainKHR</td>
</tr>
</tbody>
</table>
// 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);

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!

```c
// 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);
```
// 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);

The Vulkan Specification
is very explicit.
PART 1

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

GOOD LUCK!
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