# Instances, Devices and Validation

Setting up Vulkan to use a device, and enabling Validation Layers to validate code

# Vulkan Instance

- A Vulkan Instance is a reference to the Vulkan context.
- Defines the Vulkan version, and its capabilities.
- All Vulkan applications start by creating a Vulkan Instance
- Physical Devices accessible to the Instance are enumerated and one or more are chosen…
- Then the Instance creates a Logical Device to handle the rest of the work!
- Instances are rarely used beyond this.

# What is a device?

- **Vulkan recognises two kinds of device:**
  - Physical Device
  - Logical Device
- **Physical Device:** The GPU itself. Holds memory and queues to process pipeline, but can't be interacted with directly.
- **Logical Device:** An interface to the Physical Device. This will be used a lot. Through Logical Device we set up everything on the GPU.

# Physical Device

- **Physical Device is a reference to the GPU itself.**
- **Contains two important aspects:**
  - **Memory:** When we want to allocate memory to resources, it must be handled through the Physical Device.
  - **Queues:** Process commands submitted to GPU in FIFO order. Different queues can be used for different types of command.
- **Physical Devices are "retrieved" not created like most Vulkan concepts (you can't create a physical concept out of thin air!)**
- **Do this by enumerating over all devices and picking suitable one.**
- **We'll cover queues briefly now, then memory in a later lesson.**

# Queues

- **Physical Devices can have multiple types of queue.**
- **Types are referred to as “Queue Families”.**
- **A family can have more than one queue in it.**
- **Example queue families:**
  - **Graphics:** A family for processing graphics commands.
  - **Compute:** A family for processing compute shaders (generic commands).
  - **Transfer:** A family for processing data transfer operations.
- Queue families can be and often are combinations of these!
- When we enumerate Physical Devices, we need to check the device has the queue families we require for the application.

# Logical Device

- **Acts as an interface to the Physical Device.**
- **Will be referenced in a lot of Vulkan functions.**
- **Most Vulkan objects are created on the device, and we use the reference to the Logical Device to state which device to create those objects on.**
- **Creation is relatively simple:**
  - Define queue families and number of queues you wish to assign to the Logical Device from the Physical Device.
  - Define all the device features you wish to enable (e.g. Geometry Shader, anisotropy, wide lines, etc.)
  - Define extensions the device will use.
  - In the past you would define Validation Layers too. As of Vulkan 1.1, this is deprecated.

# Extensions

- By default, Vulkan has no understanding of what a “window” is.
- Makes sense since Vulkan is cross-platform and windows on different systems are defined differently.
- Vulkan uses extensions to add window functionality.
- These extensions are so commonly used that they come pre-packaged with Vulkan anyway!
- Can choose required extensions manually… But GLFW library has function to choose them for us!

# GLFW

- GLFW is the “Graphics Library Framework”.
- Originally designed for OpenGL but updated to work with Vulkan.
- Allows cross-platform window creation and automatic interfacing with OpenGL/Vulkan.
- Contains function that identifies the required Vulkan extensions for the host system, and returns a list of them!
- glfwGetRequiredInstanceExtensions(...)
- Can then use this list to set up Vulkan with the correct extensions.

# Validation Layers

- **By default, Vulkan does not validate code. Will not report errors, and will simply crash if it encounters a fatal error.**
- **This is to avoid unnecessary overhead of error checking in release code.**
- **Must enable a Validation "Layer" to check.**
- **Each "Layer" can check different functions.**
  - For example, VK_LAYER_LUNARG_swapchain validates Swapchain functionality. VK_LAYER_LUNARG_standard_validation is a common all-round layer.
- **Layers are similar to extensions and are not built-in to the core Vulkan code. Must be acquired from third parties (however Vulkan SDK we will acquire will already have some).**
- **Additionally, the reporting of validation errors is not a core Vulkan function and will require another extension to be applied.**
- **Note: Before Vulkan 1.1, validation layers could be specified separately for an Instance and a Logical Device. This is no longer the case, now the Instance validation layers cover both.**

# Summary

- Vulkan applications start by creating a Vulkan Instance that defines the Vulkan application.
- Enumerate Physical Devices to pick an adequate GPU.
- Create Logical Device to interface with chosen Physical Device.
- Multiple types of queue (queue families) on a Physical Device can be assigned to a Logical Device. Each one processes different types of command.
- Extensions can be applied, such as extensions to enable displaying to a window.
- We will use GLFW to create our window and choose the appropriate extensions.
- Validation Layers are optional and allow us to validate our code at run-time. They can be disabled for release versions to reduce overhead.

**See you next video!**