Setting Up Vulkan

Richard Cunard
What I’m covering

- Broad overview of setting up a Vulkan program
- The functions and data structures involved
- What needs to be set up and where
- There are going to be a LOT of details I’ll gloss over.
First Steps

- Load the Vulkan Library
- Acquire pointers to the key Vulkan functions (I’m not covering either of those)
- Create a Vulkan Instance
Creating an Instance

- Done with the function vkCreateInstance() (Look, the names aren’t meant to be creative)
- You’ll find this in Init01Instance in the sample code
- Acts as a source for pointers to additional functions to interact with the hardware
vkCreateInstance

Uses three Parameters:

- Metainfo: Vulkan version, application specifics, etc.
- A pointer to memory functions (optional)
- Address of where we want to store the instance handle
Accessing the Hardware

Using the instance, you can access a set of function pointers using `vkGetInstanceProcAddr()`, and use those to query the hardware.

These functions can be used to query hardware and use the resulting info to set up your logical device.

There are several different functions for getting info from your devices:

- `vkEnumeratePhysicalDevices` gives a list of available devices.
- `vkGetPhysicalDeviceFeatures` returns the capabilities of a device.
- `vkGetPhysicalDeviceProperties` returns the… Device Properties.
- There’s like fifteen of these functions, so you’ll need to look them up yourself.
Device Level Functions

- From here we can finally use device level functions.
- We acquire these functions with our old friend `vkGetInstanceProcAddr()`, by passing a device handle.
- This gives us the hardware specific dispatch commands.
Queue Family

- To be able to process data, you need to set up groups of queues called families.
- Families are groups of queues that can process the same type of data.

```cpp
code_snippet
```
Submitting a queue

- Handles, handles, handles…
- Actually isn’t too hard
- Queue is given by vkGetDeviceQueue
Cleanup

- When finishing a program, you need to destroy what you built
- Generally needs to be done in reverse order from how you built it

```cpp
if ( Vulkan.Device != VK_NULL_HANDLE ) {
    vkDeviceWaitIdle( Vulkan.Device );
    vkDestroyDevice( Vulkan.Device, nullptr );
}

if ( Vulkan.Instance != VK_NULL_HANDLE ) {
    vkDestroyInstance( Vulkan.Instance, nullptr );
}

if ( VulkanLibrary ) {
    #if defined(VK_USE_PLATFORM_WIN32_KHR)
    FreeLibrary( VulkanLibrary );
    #elif defined(VK_USE_PLATFORM_XCB_KHR) || defined(VK_USE_PLATFORM_XLIB_KHR)
    dlclose( VulkanLibrary );
    #endif
}
What comes next?

- Actual graphics stuff (which I’m not gonna cover)
- Swap chains and display systems
- Putting together your command buffers

https://renderdoc.org/vulkan-in-30-minutes.html

https://github.com/gametechdev/IntroductionToVulkan