Vulkan

Emanuele Franchi

February 2, 2022

Abstract

Thesis about Vulkan

Dedication

Bla Bla Bla

Acknowledgments

I want to thank...

Contents

List of Figures

Listings

Chapter 1

Vulkan

1.1 What is Vulkan?

Figure 1.1: Vulkan logo

Vulkan is a modern graphics API. It is maintained by the Khronos Group. Vulkan is meant to abstract how modern GPUs work. Using Vulkan, the programmer can write more performant code. The better performance comes at the cost of having a more verbose and low level API compared to other existing APIs such as OpenGL or Direct3D 11 and prior. Vulkan is not the only modern graphics API, other such APIs are Direct3D 12 and Metal. Nonetheless, Vulkan has the advantage of being fully cross platform.

1.2 What problems does Vulkan solve?

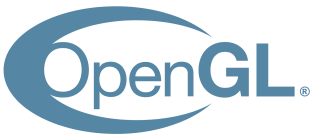

Figure 1.2: OpenGL logo

Common graphics APIs like OpenGL or Direct3D were developed during the 1990s. At that time, graphics card hardware was very limited not only in terms of computational power but also from a functionality standpoint. As time progressed, graphics card architectures continued to evolve, offering new functionalities. All these new functionalities had to be integrated with the old existing APIs. The more new functionalities were integrated, the more the GPU's driver complexity grew. Such complicated GPU drivers are inefficient and are also the cause of many inconsistencies between implementations of the same graphics API but on different GPUs.

1.3 How does Vulkan solve these problems?

Vulkan doesn't suffer from the problems we saw above because it has been designed from scratch and with modern GPU's architecture in mind. It reduces the driver overhead by being more verbose and low level. It is also designed to be multithreaded, allowing the programmer to submit GPU commands from different threads. This is very beneficial to performance, since modern CPUs usually have more than one core.

Chapter 2

Initializing Vulkan

2.1 VkInstanceCreateInfo

To access any of the functionalities offered by Vulkan we first have to create a Vulkan instance. To do this we call vkCreateInstance. When calling this function we need to pass a pointer to a VkInstanceCreateInfo struct. This struct collects all the information needed to configure our Vulkan instance.

```
1 VkInstanceCreateInfo createInfo = {};
2 createInfo.sType = VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO;
3 createInfo.pApplicationInfo = &appInfo;
4 createInfo.enabledLayerCount = layerCount;
5 createInfo.ppEnabledLayerNames = layers;
6 createInfo.enabledExtensionCount = extensionCount;
7 createInfo.ppEnabledExtensionNames = extentions;
```

Listing 2.1: VkInstanceCreateInfo initialization

2.1.1 VkApplicationInfo

We can see that the VkInstanceCreateInfo struct is not the only thing we need. We have to specify a pointer to a VkApplicationInfo struct. Such struct describes our Vulkan application.

```
1 VkApplicationInfo appInfo = {};
2 appInfo.sType = VK_STRUCTURE_TYPE_APPLICATION_INFO;
3 appInfo.pApplicationName = "Vulkan example";
4 appInfo.apiVersion = VK_API_VERSION_1_2;
```

Listing 2.2: VkApplicationInfo initialization

2.1.2 Layers

Vulkan is designed to be very lean and with as little overhead as possible. This

2.1.3 Extensions

Chapter 3

Conclusion

Appendix A

Appendix

Bibliography

[1] Introduction - Vulkan Tutorial. "`https://vulkan-tutorial.com/`".

[2] Pawel Lapinski. *Vulkan Cookbook: work through recipes to unlock the full potential of the next generation graphics API-Vulkan.* Packt, Birmingham, 2017.

[3] Graham Sellers. *Vulkan programming guide: the official guide to learning Vulkan.* OpenGL series. Addison-Wesley, Boston, 2017.