# Cognitive Dimensions Data

## Abstraction Level

### Direct3D 12

Minimum: Memory alignment and pointer arithmetics. Updating GPU resources with memcpy. Command Allocator fencing.
Maximum: ID3D12Device and ID3D12CommandQueue.

### Vulkan

Minimum: Memory alignment and pointer arithmetics. Updating GPU resources with memcpy. Ensuring physical device requirements and enabling GPU features manually.
Maximum: CommandPool and RenderPasses

## Learning Style

### Direct3D 12

Top-down. Developer needs knowledge of GPU architecture and graphics programming beforehand. Incremental learning possible after the first learning curve

### Vulkan

Top-down. Developer needs knowledge of GPU architecture and graphics programming beforehand. Incremental learning possible after the first learning curve.

## Working Framework

### Direct3D 12

Depends on task, but is frequently large. Developers often need to keep command lists, shaders, resources and parameter bindings, like descriptor tables, in mind at once.

### Vulkan

Depends on task, but is frequently large. Developers often need to keep command buffers, shaders, resources and parameter bindings, like descriptor sets, in mind at once.

## Work-step Unit

### Direct3D 12

Most tasks take several steps. In the extreme case of making a new texture available to a shader, the shader must be updated, then the texture should be read from file. Afterwards the texture should be uploaded to a new default buffer on the GPU through an upload buffer. Finally a parameter binding must be made by an update of the rootsignature, such as creating a root descriptor or root descriptor table.

### Vulkan

Most tasks take several steps. In the extreme case of making a new texture available to a shader, the shader must be updated, then the texture should be read from file. Afterwards the texture should be uploaded to a new default buffer on the GPU through an upload buffer. Finally a new parameter binding must be made by creating a new descriptor set, which we must make sure fits with the established pipeline layout.

## Progressive Evaluation

### Direct3D 12

Substantial tasks, like the addition of a new texture, often need to be made in one go before any reliable feedback can be received. Does have a debug layer, which allows for errors and warnings about the system to be given at runtime.

### Vulkan

Substantial tasks, like the addition of a new texture, often need to be made in one go before any reliable feedback can be received. The Lunar SDK comes with a validation layer feature, which allows for errors and warnings about the system to be given at runtime.

## Premature Commitment

### Direct3D 12

Most objects depend on something else. When updating parameter bindings, we must be sure that the resources are available and that the shaders can take the parameters as input.

### Vulkan

Most objects depend on something else. When updating parameter bindings, we must be sure that the resources are available and that the shaders can take the parameters as input. Features like anisotropic filtering need to be enabled on the device, before they can be used in code.

## Penetrability

### Direct3D 12

Not very penetrable. Arbitrary naming like DXGISwapchain3 hurts penetrability.   Adding a single line to code does not change the program in an informative way, and it will often make it crash. Parallelization of command list construction is rather clear, as we just need to distribute the initialization of individual objects.

### Vulkan

Not very penetrable. Adding a single line to code does not change the program in an informative way, and it will often make it crash. Parallelization can be a bit difficult, as command pools may only be accessed from one thread, and RenderPass objects need to be considered.

## API Elaboration

### Direct3D 12

It is very natural for the developer to make their own wrappers around the API. For instance, we made wrappers encapsulating 3D models along with their matrices, in addition to methods for creating different types of command list contents.

### Vulkan

It is very natural for the developer to make their own wrappers around the API. We have however not made many wrappers in the application ourselves.

## API Viscosity

### Direct3D 12

The API is very viscous, as most changes require a large part of the code to be update, such as in the case where a new resource must be added to a shader.  Yet, some parts are easy to change, such as enabling anisotropic filtering simply by adding it to the sampler on the pipeline state object.

### Vulkan

The API is very viscous, as most changes require a large part of the code to be update, such as in the case where a new resource must be added to a shader.

## Consistency

### Direct3D 12

There are consistency issues. Why may there be an arbitrary number of samplers on the root signature, but a fixed number of descriptors? Samplers sometimes refer to texture samplers and sometimes to sampling for anti-aliasing.  We need two pointer objects to the same GPU resource, one for updating it and one for parameter binding. We do like how most objects are instantiated in the same manner through the construction of a DESC object, which is then submitted to the appropriate constructor.

### Vulkan

Good consistency. Most object are instantiated in the same manner through a CreateInfo object. Yet, even with the API being very new, it already has deprecated parts like the VkDeviceCrateInfo.EnabledLayerCount field.

## Role Expressiveness

### Direct3D 12

Roles are not always clearly expressed. Some class names include version numbering like IDXGIFactory4. How does this differ from IDXGIFactory3? In addition, naming like command list and command queue does not indicate that command lists are submitted to a queue, from naming it would seem that they serve a similar function.

### Vulkan

Roles are not always clearly expressed. By naming alone, how do we know that the data from a command buffer lies in a command pool, or that a command buffer can be placed in a command queue?  There is also the question of whether a method resides on the PhyscialDevice or Device class.

## Domain Correspondence

### Direct3D 12

Fits well with the low level domain of graphics devices. Concepts like swapchain, queues and commands are  found directly in hardware. Does some abstraction, such as combining graphic and present queues into one queue type.

### Vulkan

Fits well with the low level domain of graphics devices. Concepts like swapchain, queues and commands are,found directly in hardware.