REYNEP's Vulkan "Adventure Guide"

Where, you adventure on your own ☺, I only 'guide', showing you the roadmap

Chapter 0: Prerequisites

`Suggested Reading` (before embarking on this journey)

1. https://paminerva.github.io/docs/LearnVulkan/01.A-Hello-Window
   - *Read the* `1 - Introduction` *part from here only* 😊 *[untill* `1.2. Why Vulkan?` *]*
   - 😊 *00-Introduction-and-prerequisites.pdf*
   - 😜 *01.A-Hello-Window.pdf*
2. Alternatively:- you can give this page a try too:-
   - *https://vkdoc.net/chapters/fundamentals*
   - *that is, if you are okay with "official formal-documentation"*

`The 5 Questions`

1. What is `Vulkan`? ....🤔.... Why `Vulkan`?
   - `Suggested Reading 1:- p.a.minerva`
2. Why should *'you'* learn/use `Vulkan`?
   i. *Faster*
   ii. *More Control*
   iii. *Lower Level API*
3. Why is this Important?
   - *Well if you are planning on becoming a game dev, then yeah, this kinda is important!*
   - *otherwise, if you are just here for CreatingShaders:-* `OpenGL` *is fine enough*
     a. `Shader Enthusiast:-` *https://www.shadertoy.com/*
        a. *https://www.youtube.com/playlist?list=PL9Zb80ovNLWGRFZVL4LcckTWnEGN73dFS*
        b. *https://www.youtube.com/playlist?list=PLGmrMu-IwbguU_nY2egTFmlg691DN7uE5*
        c. *https://www.youtube.com/playlist?list=PLCAFZV4XJzP-jGbTke6Bd3PNDpP1AbIKo*
        d. *https://www.youtube.com/playlist?list=PLGmrMu-IwbgtMxMiV3x4IrHPlPmg7FD-P*
        e. *https://www.youtube.com/watch?v=5J-0sy2pu_8&t=357s&pp=ygUVc2hhZGVyVG95IHJheW1hcmNoaW5n*
        f. *https://www.youtube.com/watch?v=khblXafu7iA&pp=ygUJc2hhZGVyVG95*
     b. `Making an App/UI :-` *doing everything with* `OpenGL` *-> would be just fine*
        a. *TheCherno OpenGL Playlist [YT]*
        b. *TheCherno Game Engine Playlist [YT]*

```
4. When will "You" need `vulkan`?
    kinda never  ->  unless you have grown tired of OpenGL
    kinda   yes  ->  when you wanna understand "How the heck does the GPU Work?"

    but yes, Big AAA games would need `vulkan` for even that last 5-10% performance

5. How does `vulkan` work?
    - Rest of this entire guide is dedicated to answer this question 😊
```

1. grab `vulkan-sdk` , `cmake` , `amGHOST`

1. if you don't have `vscode` & `C++ Compiler`
    - *see 4.guide.CH0.vscode.md*

2. https://vulkan.lunarg.com/sdk/home
    - *make sure* `VULKAN_SDK` & `VK_SDK_PATH` *environment variables are set*
    - *restart vscode after installing*
3. https://cmake.org/download/
    - *[optional] https://enccs.github.io/intro-cmake/hello-cmake/*
    - *[optional] OR: Watch 6/7 videos from this playlist:- https://www.youtube.com/playlist?list=PLK6MXr8gasrGmIiSuVQXpfFuE1uPT615s*
    - *restart vscode after installing*
    - **REY_DOCs**

```
cmake_minimum_required(VERSION 3.25 FATAL_ERROR)

project("idk_PROJECT" VERSION 0.1)

    set(CMAKE_CXX_STANDARD 23)
    set(CMAKE_CXX_STANDARD_REQUIRED ON)

# --------------------
    set(SRC
        "main.cpp"
    )

    set(INC
        ${CMAKE_CURRENT_SOURCE_DIR}
    )
# --------------------

# --------------------
# set_source_files_properties(main.cpp PROPERTIES COMPILE_FLAGS "/P /C")
# Output Preprocessed File
           add_executable (idk ${SRC})
target_include_directories (idk PUBLIC ${INC})

# ------amGHOST-------
         add_subdirectory  (amGHOST)
    target_link_libraries  (idk PUBLIC amGHOST)

# ------install-------
    install(TARGETS idk
        DESTINATION ${CMAKE_CURRENT_SOURCE_DIR})
```

4. `amGHOST`
    - *amateur's Generic Handy Operating System Toolkit*
        - *[secretly inspired by* `blender's GHOST` *xP 😜]*
    - `git clone -b win32-intro https://github.com/REYNEP/amGHOST`
    - *Open it with VSCode*
    - `F1` --> `CMake: Configure`
    - `F1` --> `CMake: Build`
    - `F1` --> `CMake: Install` --> `.insall` *dir*
    - *check's* ***amGHOST's Usage Example*** *inside* `amGHOST/README.md`

- `Option 1` :- *use* `cmake` *for your project too.... using* `add_subdirectory(amGHOST)`
- `Option 2` :- *use* `libamGHOST.lib` *after installing &* `#include amGHOST/<header>`

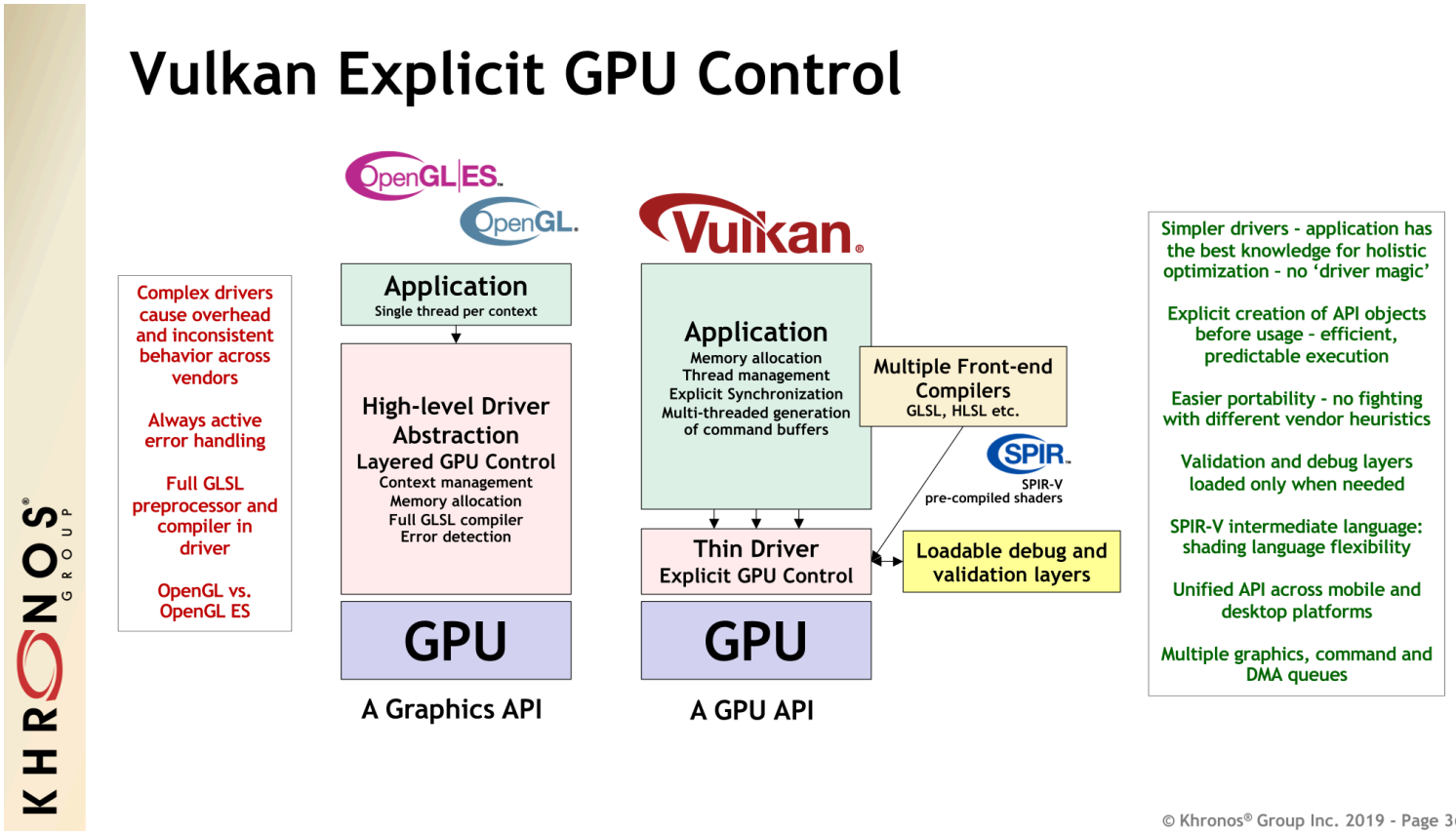
- *just copy paste* ***amGHOST's Usage Example*** *into a* `main.cpp` *for your program*

```
#include "amGHOST/amGHOST_System.hh"

int main(int argumentCount, char* argumentVector[])
{
    amGHOST_System::create_system();    // initializes amG_HEART

    amGHOST_Window* W = amG_HEART->new_window_interface();
    W->create(L"Whatever", 0, 0, 500, 600);

    REY::cin.get();     // wait for terminal input
    W->destroy();
}
```

- *[shorter than* `readme ex. 1` *]*
- *now you shall have a OS-Window* 😊

The Real "Adventure" begins here!

[well, not really. I believe the real adventure is it SHADERs and Algorithms!]



Chapter 1: `VkInstance`

0. `amVK` wrap

```
    // TwT
amVK_Instance::AppInfo              // VkApplicationInfo        [public]
amVK_Instance::CI                   // VkInstanceCreateInfo     [public]
    // You can modify these as you wish 😊
amVK_Instance::CreateInstance();    // initializes amVK_HEART
```

1. `VkApplicationInfo`

- https://vkdoc.net/man/VkApplicationInfo
  - *do remember to check the* `Valid Usage` *section* 😉
- yes, what are you waiting for, go go, shooo....
  i. `#include <vulkan/vulkan.h>`
  ii. *take an instance of that* `Struct` *-> Fill it up [😉][have the vkdoc.net as assist]*
- **REY_DOCs**
  - `.pApplicationName`
  - `.applicationVersion`
  - `.apiVersion` -> `lowest Vulkan API version` *Your APP "can run" on. [*clarification needed:- lowest or highest]*
  - `.pEngineName` *-> Also, we can set the* `name`
  - `.engineVersion` *-> and* `version` *of the* `engine` *(if any) used to create Your APP.*
    - *This can help* `vulkan driver implementations` *to perform "ad-hoc" optimizations.*
      - *e.g. like if a Triple-A [AAA] game used, for say,* `Unreal Engine Version 4.1.smth` *idk* 🤷
  - *REFs:- 1. minerva*

2. `VkInstanceCreateInfo`

- https://vkdoc.net/man/VkInstanceCreateInfo
  - *yeah, do remember to check the* `Valid Usage` *section* 😉
  - `.ppEnabledLayerNames` *-> "ChapterZZZ"*
  - `.ppEnabledExtensionNames` *-> Chapter4.2*
    - *Don't hesitate about* `EnabledLayer` & `EnabledExtensions` *right now*
      - *come back and add them when you need to* 😉
      - *This is what I would mean, when i would point smth to a later chapter*
- **REY_DOCs**
  - *Nothing that I need to add, in this section*
  - *Tho if this section gets big, I will create a separate* `.md` *file for that thingy*

3. `VkInstance m_instance = nullptr;`

- https://vkdoc.net/man/VkInstance
  - *again.... yeah, do remember to check the* `Valid Usage` *section* 😉

4. `vkCreateInstance(CI, &m_instance)`

- https://vkdoc.net/man/vkCreateInstance
  - `Valid Usage` *section.... (yeah, everytime)*

5. Error Handling / Checking / Logging

- check out my `amVK_log.hh`
  - *uses REY_LoggerNUtils inside amGHOST*
    - *has a simple* `stackTracer()` *that i basically stripped from blender3D codebase* 😆

6. The Result

- Check out:- 4.guide.chapter1.hh

7. The Unused ones

1. `vkEnumerateInstanceExtensionProperties()` -> *Chapter4.2*
   - *https://vkdoc.net/man/vkEnumerateInstanceExtensionProperties*
2. `Add_InstanceEXT_ToEnable(const char* extName)` -> *Chapter4.2*
   - *this is a amVK/REY Custom Function*

Overview

We need to create/get hold of a couple of handles:		
Instance	1 `VkInstance` per program/app	`VkInstance`
Window Surface	*Surface(OS-Window)* [for actually linking Vulkan-Renders to Screen/Surface]	`VkSurfaceKHR`
Physical Device	*An Actual HARDWARE-GPU-device*	`VkPhysicalDevice`
Queue	*Queue(Commands)* to be executed on the GPU	`VkQueue`
Logical Device	*The "Logical" GPU Context/Interface* (Software Layer)	`VkDevice`
Swap Chain	*Sends Rendered-Image to the Surface(OS-Window)* *Keeps a backup image-buffer to Render onto*	`VkSwapchainKHR`

Take a look into this awesome slide from slide-26 onwards, to understand what each of steps *"feel like"/mean/"how to imagine them"*.

*slide = Vulkanised 2023 Tutorial Part 1

Chapter 2: `VkDevice`

0. `amVK` wrap

```
// TwT
amVK_GlobalProps::EnumeratePhysicalDevices();
amVK_GlobalProps::GetPhysicalDeviceQueueFamilyProperties();

amVK_Device* D = new amVK_Device(amVK_GlobalProps::GetARandom_GPU());
    D->select_QFAM_Graphics();
    D->CI                           // VkDeviceCreateInfo       [public]
    D->QCI.Default                  // VkDeviceQueueCreateInfo  [public]
    D->QCI.Array                    // REY_ArrayDYN<VkQCI>      [public]
        // You can take your own VkDeviceQueueCreateInfo & push_back into this array
    D->CreateDevice();
```

1. `vkEnumeratePhysicalDevices(m_instance, &m_deviceCount, nullptr)`

- https://vkdoc.net/man/vkEnumeratePhysicalDevices
- **REY_DOCs**

```
uint32_t deviceCount = 0;
    // [implicit valid usage]:- must be 0 [if 3rd-param = nullptr]
    vkEnumeratePhysicalDevices(m_instance, &deviceCount, nullptr);
        // it's kinda like the function is 'output-ing into' deviceCount

std::vector<VkPhysicalDevice> HardwareGPU_List(gpuCount);
    // best to save this as a class member variable
    vkEnumeratePhysicalDevices(m_instance, &deviceCount, HardwareGPU_List.data());
        // note: it does return     VkResult return_code
```

  - *Visualization / [See it] / JSON Printing:-* *4.guide.chapter2.1.json.hh*
  - *So far, The result:-* *4.guide.chapter2.1.midway.hh*

2. `vkCreateDevice()`

- https://vkdoc.net/man/vkCreateDevice
  - `param pAllocator` -> *"ChapterZZZ"*
- **REY_DOCs**
  - *we are not gonna call the* `vkCreateDevice()` *yeeeet....*
    - *but, yes, we've already made the class container around it* 😅
      - *4.guide.chapter2.2.midway.hh*
    - *we'll call this functiion in* ***Chapter2.9***
  - *but we did need to know first about* `vkCreateDevice()`
    - *because, the idea is, our sole task is to* ***fill it up step by step***

3. `VkDeviceCreateInfo`

- https://vkdoc.net/man/VkDeviceCreateInfo
  - `.LayerInfo` -> *Deprecated*
  - `.ExtensionInfo` -> *"ChapterZZZ"*
  - `.pQueueCreateInfos` -> *next part*
    - *So far, The result:-* *4.guide.chapter2.3.midway.hh*
- **REY_DOCs**
  - `.pQueueCreateInfos` -> *yes, you 'can' mass multiple* 😉
  - *Sometimes there will be* `.zzzCreateInfoCount` *&* `.pZZZCreateInfos`
    - *So you could like pass in an array/vector*
    - *You will see this in lots of other places*

-- | -- | -- | ----------

4. `VkDeviceQueueCreateInfo` - ***'The Real Deal'***

- https://vkdoc.net/man/VkDeviceQueueCreateInfo
  - `.queueFamilyIndex` -> *next 3 subchapters*
    - *So far, The result:-* *4.guide.chapter2.4.midway.hh*
- **REY_DOCs:- Support for multiple QCI**
  - `.pQueuePriorities` -> *yes, this can be multiple "Priorities"* 😆 *[idk yet why tho]*

```
/* ============ REY_LoggerNUtils::REY_Utils.hh ============ */
REY_ArrayDYN<VkDeviceQueueCreateInfo> Array = REY_ArrayDYN<VkDeviceQueueCreateInfo>(2);
```

```
    // allocate enough space for 2 elements
REY_ARRAY_PUSH_BACK(Array) = this->Default_QCI;
REY_ARRAY_PUSH_BACK(Array) =          Your_QCI;

/* ============ std::vector ============ */
std::vector<VkDeviceQueueCreateInfo> Array = std::vector<VkDeviceQueueCreateInfo>(2);
Array.push_back(this->Default_QCI);
Array.push_back(         Your_QCI)
```

- *So far, The result:- 4.guide.chapter2.7.TheEnd.hh*

5. `vkGetPhysicalDeviceQueueFamilyProperties()`

- https://vkdoc.net/man/vkGetPhysicalDeviceQueueFamilyProperties
- **REY_DOCs**
  - *a GPU can have "multiple QueueFamilies"*
    - *a* `QueueFamily` *might support* `VK_QUEUE_GRAPHICS_BIT`
    - *another* `QueueFamily` *might support* `VK_QUEUE_COMPUTE_BIT`
    - *another* `QueueFamily` *might support* `VK_QUEUE_TRANSFER_BIT`
    - *another* `QueueFamily` *might support* `VK_QUEUE_VIDEO_ENCODE_BIT_KHR`
    - *another* `QueueFamily` *might support a-mixture of multiple*
    - *talking about this in -> the next part [chapter2.6.]*

```
static inline REY_Array<REY_Array<VkQueueFamilyProperties>>   s_HardwareGPU_QFamProps_List2D;
#define amVK_2D_GPUs_QFAMs                      amVK_Instance::s_HardwareGPU_QFamProps_List2D
    // "REY_LoggerNUtils/REY_Utils.hh" 😁

static inline void GetPhysicalDeviceQueueFamilyProperties(void) {
    amVK_2D_GPUs_QFAMs.reserve(amVK_GPU_List.n);            // malloc using "new" keyword
    for ( uint32_t k = 0;  k < amVK_GPU_List.n; k++ )       // for each GPU
    {
        REY_Array<VkQueueFamilyProperties> *k_QFamProps = &amVK_2D_GPUs_QFAMs.data[k];

        uint32_t queueFamilyCount = 0;
            vkGetPhysicalDeviceQueueFamilyProperties(amVK_GPU_List[k], &queueFamilyCount,
nullptr);

        k_QFamProps->n = queueFamilyCount;
        k_QFamProps->data = new VkQueueFamilyProperties[queueFamilyCount];
            vkGetPhysicalDeviceQueueFamilyProperties(amVK_GPU_List[k], &k_QFamProps->n,
k_QFamProps->data);
    }
}
```

- **Visualization / [See it] / JSON Printing:- 4.guide.chapter2.5.json.hh**
  - *Check the 3070 JSON by REY*
- **So far, The result:- 4.guide.chapter2.5.amVK.hh**
  - *Compare to -> 4.guide.chapter2.1.midway.hh*
    - `2DArray_QFAM_Props` *part & below were added only compared to* `Chapter2.1.`

6. `VkQueueFamilyProperties`

- https://vkdoc.net/man/VkQueueFamilyProperties
- **REY_DOCs**
  - `.queueFlags` *-> we are gonna choose a* `QCI.queueFamilyIndex` *based on these flags*
    - *primarily, for the least, we wanna choose a* `QueueFamily` *that supports* `VK_QUEUE_GRAPHICS_BIT`
    - *all kinds of amazing things can be done using*
      - `VK_QUEUE_COMPUTE_BIT`
      - `VK_QUEUE_TRANSFER_BIT`
      - `VK_QUEUE_VIDEO_ENCODE_BIT_KHR`
  - `.queueCount` *-> yes there is a limit to 'how many* `Queues` *we are allowed to work with'* 😂

7. `VkDeviceQCI.queueFamilyIndex`

- `QCI => QueueCreateInfo`
  - *[VkDeviceQueueCreateInfo]*
- **REY_DOCs**
  - `Task:-` *is to choose a* `QueueFamily` *that supports* `VK_QUEUE_GRAPHICS_BIT` 😉
    - *(if you've followed on so far -> this should be easy* 😉*)*
  - *Resolving all of this into* `amVK_Device.hh`

```
void amVK_Device::Select_QFAM_GRAPHICS(void) {
    if (!amVK_Instance::called_GetPhysicalDeviceQueueFamilyProperties) {
        amVK_Instance::EnumeratePhysicalDevices();
    }

    if (!amVK_Instance::called_GetPhysicalDeviceQueueFamilyProperties) {
        amVK_Instance::GetPhysicalDeviceQueueFamilyProperties();
    }

    amVK_Instance::amVK_PhysicalDevice_Index index =
amVK_HEART->GetARandom_PhysicalDevice_amVK_Index();
    this->QCI.Default.queueFamilyIndex =
amVK_Instance::ChooseAQueueFamily(VK_QUEUE_GRAPHICS_BIT, index);
        // If you wanna see the implementation for this function
}
```

  - *So far, The result:-* *4.guide.chapter2.9.Props.hh*
  - *So far, The result:-* *4.guide.chapter2.9.amVK.cpp*

-- | -- | -- | ----------

8. back to `vkCreateDevice()` [finally calling it ☺]

- **REY DOCs**

```
amVK_Device* D = new amVK_Device(amVK_HEART->GetARandom_PhysicalDevice());
    // VkDeviceCreateInfo CI => Class Member
    // VkDeviceQueueCreateInfo QCI => Class Member
D->Select_QFAM_GRAPHICS();
D->CreateDevice();
```

- *Think of this as a PSeudoCode / or / check out my code if you wanna*
- `CreateInfo` *=> By default has initial values inside* `amVK_Device`

9. Organizing stuff into classes....

1. `amVK_GlobalProps.hh`
    i. `class amVK_GlobalProps`
        - `amVK_Instance::GetPhysicalDeviceQueueFamilyProperties()`
        - `amVK_Instance::EnumeratePhysicalDevices()`
        - *& Everything related to those two + The Data + The Properties*
    - *https://github.com/REYNEP/amGHOST/tree/3e44b982902a3f3fa4ac584aefb19da3d4cdfcc6*
    - *So far, The result:-*
        - *4.guide.chapter2.9.Props.hh*
        - *4.guide.chapter2.9.amVK.cpp*

10. `vkGetPhysicalDeviceProperties()`

- https://vkdoc.net/man/vkGetPhysicalDeviceProperties
- `VkPhysicalDeviceProperties` :- https://vkdoc.net/man/VkPhysicalDeviceProperties
    - `.deviceType` :- *https://vkdoc.net/man/VkPhysicalDeviceType*
    - `.limits` :- *save it for later* 😉
    - *you don't need to read the whole documentation of this page*
- for now we won't need, we will need in **"ChapterZZZ"**

Chapter 3: Common Patterns: *if someone missed to catch it yet* 😉

```
Object  Vk      VkInstance
Types   Vk      VkInstanceCreateInfo
Funcs   vk      vkCreateInstance()
Enums   VK_     VK_STRUCTURE_TYPE_INSTANCE_CREATE_INFO

Extensions
    KHR:- Khronos authored,
    EXT:- multi-company authored

Creating "VkZZZ" object
    1. take `VkZZZCreateInfo` --> fill it up
    2. call `vkCreateZZZ()`
    3. also `vkDestroyZZZ()` before closing your app

    4. Some objects get "allocated" rather than "created"
        `VkZZZAllocateInfo` --> `vkAllocateZZZ` --> `vkFreeZZZ`

    5. Sometimes there will be `.zzzCreateInfoCount` & `.pZZZCreateInfos`
                        e.g. `.queueCreateInfoCount` & `.pQueueCreateInfos``
            -> So you could like pass in an array/vector
            -> You will see this in lots of other places

Getting List/Properties
    1. vkEnumerateZZZ() --> \see `[Chapter2.1.] vkEnumeratePhysicalDevices()` example
```

-- | -- | -- | ----------

7. `sType` & `pNext`
    - *Many Vulkan structures include these two common fields*
8. `sType` :-
    - *It may seem somewhat redundant, but this information can be useful for the* `vulkan-loader` *and actual* `gpu-driver-implementations` *to know what type of structure was passed in through* `pNext` .
9. `pNext` :-
    - *allows to create a linked list between structures.*
    - *It is mostly used when dealing with extensions that expose new structures to provide additional information to the* `vulkan-loader` , `debugging-validation-layers` , *and* `gpu-driver-implementations` .
        - *i.e. they can use the* ***`pNext->stype`*** *field to know what's ahead in the linked list*

-- | -- | -- | ----------

```
10. Do remember to check the `Valid Usage` section within each manual-page
```

11. **CreateInfo StartingPoint**

```
VkRenderPassCreateInfo CI = {
    .sType = VK_STRUCTURE_TYPE_SWAPCHAIN_CREATE_INFO_KHR,
```

```
    .pNext = nullptr,
    .flags = 0
};
```

12. **Keywords in my Vulkan Guide**

```
1. ChapterZZZ => Unknown WIP/TBD Chapter
2. Chapter2.4 =>
        If LATER-CHAPTER => Dont hesitate right now, Do this when you each that LATER-Chapter
        If  PREV-CHAPTER => You can go back and check 😜
        🔗 `SurfCAP.currentTransform`
        🔗 Chapter2.4

3. https://vkdoc.net/man/vkGetPhysicalDeviceSurfaceFormatsKHR
    - SHORT CRUCIAL/MUST-KNOW/WARNING info about "params/members"
4. REY_DOCs
    - Actual Notes
    - Mostly, vkdoc.net documentation is good enough. But if I wanna add smth extra, it goes
here
5. So far, The result
6. Visualization / [See it] / JSON Printing
7. Implement Exactly like **_Chapter2.1_** 😉
    - `vkEnumeratePhysicalDevices()`

8. 2DriverIMPL:- To The People Who are gonna Implement the Driver
    - Other Keyword:- "DriverGurantee"

9. Gotta add more emojis for common stuffs
```

Two Questions I keep on pondering 🤔

```
- a) Would this make sense to someone else?
- b) Would this make sense to a 5 year old?
```

```
**_Chapter1.2_**
**"ChapterZZZ"**

##### REY_DOCs
##### So far, The result:- [4.guide.chapter4.2.TheEnd.hh](./examples/4.guide.chapter4.2.TheEnd.hh)

https://vkdoc.net/man/vkGetPhysicalDeviceSurfaceFormatsKHR

Visualization / [See it] / JSON Printing:-
[4.guide.chapter2.1.json.hh](./examples/4.guide.chapter2.1.json.hh)
So far, The result:- [4.guide.chapter2.1.midway.hh](./examples/4.guide.chapter2.1.midway.hh)

Implement Exactly like **_Chapter2.1_** 😊
    - `vkEnumeratePhysicalDevices()`

2DriverIMPL:- To The People Who are gonna Implement the Driver
_DriverGurantee_
-->
```

```
<div class="REY_NOSHOW_PDF">

------------------------------------------------------------------------
</div>
<div class="REY_PAGEBREAK" style="page-break-after: always;"></div>
<div class="REY_NOSHOW_PDF">

------------------------------------------------------------------------
</div>
```

Chapter 4: `VkSwapchainKHR`

0. `VkSwapchainCreateInfoKHR` i

- https://vkdoc.net/man/VkSwapchainCreateInfoKHR
  - `.flags` -> *"ChapterZZZ"*
  - `.surface` -> *next part [Chapter4.2]*
  - *image options -> next part [Chapter4.4]*
    - `.minImageCount` ->
    - `.imageFormat` -> 😳
    - `.imageColorSpace` -> 😳
    - `.imageExtent` -> 😊
    - `.imageArrayLayers`
    - `.imageUsage`
    - `.imageSharingMode` -> `EXCLUSIVE/CONCURRENT` *[Toggle]*
  - `VK_SHARING_MODE_CONCURRENT` -> *"ChapterZZZ"*
    - `.queueFamilyIndexCount` -> *if using, must be* `greated than 1`
    - `.pQueueFamilyIndices`
  - *more image options -> next part*
    - `.preTransform` :- `VkSurfaceTransformFlagBitsKHR`
    - `.compositeAlpha` :- `VkCompositeAlphaFlagBitsKHR`
    - `.presentMode` :- `VkPresentModeKHR`
    - `.clipped` :- `VkBool32`
  - `.oldSwapchain` -> *"ChapterZZZ"*

1. `amVK` wrap 1

```
// TwT

    REY_LOG("");
amVK_GlobalProps::EnumerateInstanceExtensions();
amVK_Instance::Add_InstanceEXT_ToEnable("VK_KHR_surface");
amVK_Instance::Add_InstanceEXT_ToEnable(amGHOST_System::get_vulkan_os_surface_ext_name());
    // amGHOST_VkSurfaceKHR::create_surface() needs that extension enabled
amVK_Instance::CreateInstance();

    REY_LOG("");
VkSurfaceKHR  VK_S = amGHOST_VkSurfaceKHR::create_surface(W, amVK_Instance::vk_Instance);

// another amVK_Wrap, at the end of this file
```

2. `VkSurfaceKHR` 🏄‍♀️

- https://vkdoc.net/man/VkSurfaceKHR
- https://vkdoc.net/extensions/VK_KHR_surface
    - *Yaaaay, we have reached our first extension to enable*
        - *we need to enable it back in* `vkCreateInstance()` *from* **Chapter1.2**

1. `vkEnumerateInstanceExtensionProperties()`
    - *https://vkdoc.net/man/vkEnumerateInstanceExtensionProperties*
    - *Implement Exactly like* **Chapter2.1** 😉
        - `vkEnumeratePhysicalDevices()`
2. `IS_InstanceEXT_Available(const char* extName)`

```
bool amVK_GlobalProps::IS_InstanceEXT_Available(const char *extName) {
    for (uint32_t k = 0, lim = amVK_EXT_PROPs.n; k < lim; k++) {
        if (strcmp(amVK_EXT_PROPs[k].extensionName, extName) == 0) {    // <cstring>
            return true;
        }
    }
    return false;
}
```

3. `Add_InstanceEXT_ToEnable(const char* extName)`

```
static inline REY_ArrayDYN<char*> s_Enabled_EXTs = REY_ArrayDYN<char*>(nullptr, 0, 0);
    // It will be automatically allocated, resize, as we keep adding 😊
#include <string.h>
void amVK_Instance::Add_InstanceEXT_ToEnable(const char* extName)
{
    if (!amVK_GlobalProps::called_EnumerateInstanceExtensions) {
         amVK_GlobalProps::EnumerateInstanceExtensions();
    }

    if (amVK_GlobalProps::IS_InstanceEXT_Available(extName)) {
        char  *dont_lose = new char[strlen(extName)];
        strcpy(dont_lose, extName);

        s_Enabled_EXTs.push_back(dont_lose);

        amVK_Instance::CI.enabledExtensionCount = s_Enabled_EXTs.neXt;
        amVK_Instance::CI.ppEnabledExtensionNames = s_Enabled_EXTs.data;
    }
    else {
        REY_LOG_notfound("Vulkan Extension:- " << extName);
    }
}
```

4. OS Specfic SurfaceEXT & Creating it

```
amVK_Instance::Add_InstanceEXT_ToEnable(amGHOST_System::get_vulkan_os_surface_ext_name());
    // or
amVK_Instance::Add_InstanceEXT_ToEnable("VK_KHR_win32_surface");
    // or some other surface name
```

i. `VkWin32SurfaceCreateInfoKHR` & `vkCreateWin32SurfaceKHR()`
   - *https://vkdoc.net/man/VkWin32SurfaceCreateInfoKHR*

```
pure-virtual VkSurfaceKHR amGHOST_VkSurfaceKHR_WIN32::create(VkInstance I)
{
    amGHOST_SystemWIN32 *heart_win32 = (amGHOST_SystemWIN32 *) amGHOST_System::heart;
    VkWin32SurfaceCreateInfoKHR CI = {
        .sType = VK_STRUCTURE_TYPE_WIN32_SURFACE_CREATE_INFO_KHR,
        .pNext = NULL,
        .flags = 0,
        .hinstance = heart_win32->_hInstance,
        .hwnd = this->W->m_hwnd
            // W = amGHOST_WindowWIN32
    };

    VkSurfaceKHR S = nullptr;
    VkResult return_code = vkCreateWin32SurfaceKHR(I, &CI, nullptr, &S);
    amVK_return_code_log( "vkCreateWin32SurfaceKHR()" );
    return S;
}
```

ii. `VkXlibSurfaceCreateInfoKHR` & `vkCreateXlibSurfaceKHR()` 🛠 *[wip]*

iii. **REY_DOCs**
   - *you can also check* `amGHOST_VkSurfaceKHR::create_surface()` 😊

iv. **So far, The result:-** **4.guide.chapter4.2.TheEnd.hh**
   - *in the end people will just use 1 line*

```
VkSurfaceKHR VK_S = amGHOST_VkSurfaceKHR::create_surface(amG_WindowOBJ,
amVK_Instance::s_vk);
```

---

3. `Naming Patterns` 📛

- example naming patterns for storing all these data.... cz *it's gonna get overwhelming pretty soon, pretty fast*

1. `Arrays`

```
class amVK_GlobalProps {
    public:
        // Array of `HardWare amVK_1D_GPUs` connected to motherboard
    static inline REY_Array<VkPhysicalDevice>                          amVK_1D_GPUs;
    static inline REY_Array<REY_Array<VkQueueFamilyProperties>>        amVK_2D_GPUs_QFAMs;
    static inline REY_Array<VkExtensionProperties>                     amVK_1D_InstanceEXTs;
    static inline REY_ArrayDYN<char*>
amVK_1D_InstanceEXTs_Enabled;
    static inline REY_ArrayDYN<SurfaceInfo>                            amVK_1D_SurfaceInfos;
    static inline REY_Array<REY_Array<VkExtensionProperties>>          amVK_2D_GPUs_EXTs;
        // REY_Array  doesn't allocate any memory by default

    #define amVK_LOOP_GPUs(_var_)           \
```

```
        for (uint32_t _var_ = 0, lim = amVK_1D_GPUs.n;              _var_ < lim;  _var_++)
    #define amVK_LOOP_QFAMs(_k_, _var_)     \
        for (uint32_t _var_ = 0, lim = amVK_2D_GPUs_QFAMs[_k_].n;  _var_ < lim;  _var_++)
};
```

2. `ChildrenStructs`

```
class amVK_GlobalProps {
    public:
    /**
     * VULKAN-EXT:- `VK_KHR_surface`
        *       IMPL:- `amVK_1D_SurfaceInfos`
        */
    class SurfaceInfo {
        public:
        VkSurfaceKHR S = nullptr;
        SurfaceInfo(void) {}
        SurfaceInfo(VkSurfaceKHR pS) {this-> S = pS;}

                REY_Array<REY_Array<VkSurfaceFormatKHR>>          amVK_2D_GPUs_ImageFMTs;

        bool called_GetPhysicalDeviceSurfaceFormatsKHR = false;
        void        GetPhysicalDeviceSurfaceFormatsKHR(void);  // amVK_2D_GPUs_ImageFMTs
    };
};
```

3. `VkFuncCalls`

```
class amVK_GlobalProps {
    public:
    static inline bool called_EnumeratePhysicalDevices = false;
    static inline bool called_GetPhysicalDeviceQueueFamilyProperties = false;
    static inline bool called_EnumerateInstanceExtensions = false;

    public:
    static void EnumeratePhysicalDevices(void);                          // amVK_1D_GPUs
    static       void GetPhysicalDeviceQueueFamilyProperties(void);      // amVK_2D_GPUs_QFAMs
    static void EnumerateInstanceExtensions(void);                       // amVK_1D_InstanceEXTs
};
```

- **REY_DOCs**
  - *Lots of other nice stuffs are happening inside* `amVK_GlobalProps.hh`
- **So far, The result:-**
  - *4.guide.chapter4.3.Props.hh*
  - *4.guide.chapter4.3.Props.cpp*
  - *4.guide.chapter4.3.PropsOLD.hh*

4. `SwapChain Image Options` 🖼

1. `vkGetPhysicalDeviceSurfaceFormatsKHR()`
   - *https://vkdoc.net/man/vkGetPhysicalDeviceSurfaceFormatsKHR*
     - `param surface`
   - **REY_DOCs**
     - *Implement Exactly like Chapter2.5* 😊
       - `vkGetPhysicalDeviceQueueFamilyProperties()`
       - *Only difference is,* `Formats` *might be a bit different as per* `VkSurfaceKHR`
2. `VkSurfaceFormatKHR`
   - *https://vkdoc.net/man/VkSurfaceFormatKHR*
   - **REY_DOCs**
     - *Combo of* `ImageFormat` & `ColorSpace`
       - *so, the gpu kinda expects you to respect these combos, instead of mumbo-jumbo-ing & mixing random stufs alltogether....*
       - *altho, even if you do so, gpu is probably gonna show you the result of WRONG COLORSPACE/IMAGEFORMATs on the screen*

---

3. `Life is Hard without Images/Visualization`
   - *So we are gonna Export to JSON/YAML*
   - *4.guide.chapter4.4.3.Enum2String.hh*
   - *4.guide.chapter4.4.3.data.jsonc*
   - *4.guide.chapter4.4.3.Export.cpp*
     - *dw, don't use this code, it will be refactored & organized in Chapter4.4.6*

---

4. `VkSurfaceCapabilitiesKHR`
   - *https://vkdoc.net/man/VkSurfaceCapabilitiesKHR*
   - **REY_DOCs**
     - `.minImageCount`
       - *2DriverIMPL:- **must** be at least 1*
     - `.currentExtent`
       - *as the OS Window size changes,* `SurfCaps` *also change*
       - *call* `vkGetPhysicalDeviceSurfaceCapabilitiesKHR()` *to get updated* `WindowSize` / `SurfCaps`
     - `.maxImageArrayLayers`
       - *2DriverIMPL:- **must** be at least 1*
     - `.supportedTransforms`
       - *2DriverIMPL:- at least 1 bit **must** be set.*
     - `.supportedUsageFlags`
       - *2DriverIMPL:-* `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT` ***must** be included in the set. Implementations **may** support additional usages.*
     - `.supportedCompositeAlpha`
       - `ALPHA-Blending/Transparency/GlassEffect` *:- you'd have to enable blending/transparency @ OS-Level first, iguess* 🤔
       - *Transparency -> "ChapterZZZ"*
5. `vkGetPhysicalDeviceSurfaceCapabilitiesKHR()`
   - *https://vkdoc.net/man/vkGetPhysicalDeviceSurfaceCapabilitiesKHR*
   - **REY_DOCs**
     - *we add on top of Chapter4.4.1* 😊
       - `vkGetPhysicalDeviceSurfaceFormatsKHR()`
     - *4.guide.chapter4.4.5.midway.cpp*

---

6. `Life is Hard without Images/Visualization 2`
    - *Soooooo many things to keep track of, So here we go again*
    - *4.guide.chapter4.4.6.Export.cpp*
    - *4.guide.chapter4.4.6.data.jsonc*

---

7. `VkSharingMode`
    - *https://vkdoc.net/man/VkSharingMode*
    - *it's like a Toggle/Button ->* `EXCLUSIVE/CONCURRENT`
8. **So far, The result:-**

```
amVK_SwapChain *SC = new amVK_SwapChain(VK_Surface);
    SC->CI.imageFormat = VK_FORMAT_B8G8R8A8_UNORM;
    SC->CI.imageColorSpace = VK_COLOR_SPACE_SRGB_NONLINEAR_KHR;
    SC->CI.minImageCount    =
amVK_GlobalProps::amVK_1D_SurfaceInfos[0].amVK_1D_GPUs_SurfCAP[0].minImageCount;
    SC->CI.imageExtent      =
amVK_GlobalProps::amVK_1D_SurfaceInfos[0].amVK_1D_GPUs_SurfCAP[0].currentExtent;
    SC->CI.imageArrayLayers =
amVK_GlobalProps::amVK_1D_SurfaceInfos[0].amVK_1D_GPUs_SurfCAP[0].maxImageArrayLayers;
        // You can just use "1" too, which is guranteed by DRIVER_IMPLEMENTATION [2DriverIMPL]
    SC->CI.imageSharingMode = VK_SHARING_MODE_EXCLUSIVE;
        // `EXCLUSIVE/CONCURRENT` [Toggle]
    SC->CI.imageUsage       = VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT;
        // 2DriverIMPL:- VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT is guranteed to be supported by
SurfCAP
```

---

9. `Abbreviations`
    - `PD` -> *PhysicalDevice*
    - `GPUs` -> *PhysicalDevices*
    - `CI` -> *CreateInfo*
    - `QCI` -> *QueueCreateInfo*
    - `QFAM` -> *QueueFamily*
    - `SurfCAP` -> *https://vkdoc.net/man/VkSurfaceCapabilitiesKHR*
    - `SurfFMT` -> *https://vkdoc.net/man/VkSurfaceFormatKHR*
    - `SC` -> *SwapChain*

---

10. `VkSwapchainCreateInfoKHR`

- https://vkdoc.net/man/VkSwapchainCreateInfoKHR
    - `.flags` -> *"ChapterZZZ"*
    - `.surface` -> *Chapter4.2* `VkSurfaceKHR` 🐍
    - *image options ->* *Chapter4.4*
        - `.minImageCount` -> 😉 `SurfCAP.minImageCount`

- `.imageFormat` -> 😉 `SurfFMT[x].format`
- `.imageColorSpace` -> 😱 `SurfFMT[x].colorSpace`
  - *Choosing a Combo -> "ChapterZZZ"*
  - *Compositing & ColorSpaces -> "ChapterZZZ"*
- `.imageExtent` -> 😉 `SurfCAP.minImageCount`
- `.imageArrayLayers` -> `1`
  - *DriverGurantee*
- `.imageUsage` -> `VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT`
  - *DriverGurantee*
- `.imageSharingMode` -> `EXCLUSIVE/CONCURRENT` *[Toggle]*
  - `VK_SHARING_MODE_CONCURRENT` -> *"ChapterZZZ"*
    - *we aren't gonna use concurrent for now*
    - `.queueFamilyIndexCount` -> `0`
    - `.pQueueFamilyIndices` -> `nullptr`

---

5. `SwapChain Compositing Options` 🤷‍♂️

1. `.compositeAlpha`
   - *https://vkdoc.net/man/VkCompositeAlphaFlagBitsKHR*
   - **REY_DOCs**
     - `Options` :- *Don't use / Pre-multiplied / Post-multiplied / inherit from OS-native window system*
     - `Requirement` :-
       - *You would have to enable @ OS level first, to enable ALPHA/Transparency/GlassEffect for window-s/surfaces*
       - *then after that, if you query for* `vkGetPhysicalDeviceSurfaceCapabilitiesKHR()`
         - `SurfCAP.supportedCompositeAlpha` *will change*
       - *by default, it's prolly always gonna support*
         - `VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR`
         - *i.e. if you haven't done any mastery wizardry yet, to enable ALPHA/Transparency/GlassEffect*
2. `.preTransform`
   - *https://vkdoc.net/man/VkSurfaceTransformFlagBitsKHR*
   - **REY_DOCs**
     - 🔗 `SurfCAP.currentTransform`
     - *you should probably log it if* `currentTransform` *isn't*
       - `VK_SURFACE_TRANSFORM_IDENTITY_BIT_KHR`
3. `.clipped`
   - **REY_DOCs**
     - *Setting clipped to* `VK_TRUE` *allows the implementation to discard rendering outside of the surface area*
4. `.presentMode` 🏷 `VkPresentModeKHR`
   - *https://vkdoc.net/man/VkPresentModeKHR*
   - **REY_DOCs**
     - `Options` :- *IMMEDIATE / MAILBOX / FirstInFirstOut / FIFO_Relaxed*
5. `.oldSwapChain`
   - **REY_DOCs**
     - *if you are "re-creating" swapchain & you had an oldSwapchain*
     - *We do this when*
       a. *Window Size / WindowExtent / Surface was Changed*

6. **So far, The result:-**

```
amVK_SwapChain *SC = new amVK_SwapChain(VK_Surface);
    ... Image Stuffs
    SC->CI.compositeAlpha   = VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR;
    SC->CI.preTransform     =
amVK_GlobalProps::amVK_1D_SurfaceInfos[0].amVK_1D_GPUs_SurfCAP[0].currentTransform;
    SC->CI.clipped          = VK_TRUE;
    SC->CI.presentMode      = VK_PRESENT_MODE_FIFO_KHR;
    SC->CI.oldSwapchain     = nullptr;
```

6. `SwapChain Extension Enabling` ⚠ [`VK_KHR_swapchain`]

1. `vkEnumerateDeviceExtensionProperties()`
   - *https://vkdoc.net/man/vkEnumerateDeviceExtensionProperties*
     - *honestly this should be named* `vkEnumeratePhysicalDeviceExtensionProperties()`
     - *bcz,*
       - *it doesn't associate with* `VkDevice`
       - *but rather with* `VkPhysicalDevice`
   - **REY_DOCS**

```
class amVK_GlobalProps {
    ...
    static inline bool called_EnumerateDeviceExtensionProperties = false;
    static void               EnumerateDeviceExtensionProperties(void); // amVK_2D_GPUs_EXTs

    static inline REY_Array<REY_Array<VkExtensionProperties>>              amVK_2D_GPUs_EXTs;
    #define amVK_LOOP_GPU_EXTs(_k_, _var_)  for (uint32_t _var_ = 0,  lim =
amVK_2D_GPUs_EXTs[_k_].n;   _var_ < lim;  _var_++)

    static bool      IS_GPU_EXT_Available(PD_Index GPU_k, const char *extName); //
amVK_2D_GPUs_EXTs
    // kinda copy of IS_InstanceEXT_Available
    ...
};
```

2. `amVK_Device::Add_GPU_EXT_ToEnable(const char* extName)`

```
class amVK_Device {
    ...
    REY_ArrayDYN<char*>   amVK_1D_DeviceEXTs_Enabled;
    void Log_GPU_EXTs_Enabled(VkResult ret);
    void Add_GPU_EXT_ToEnable(const char* extName);
        // Copy of `amVK_GlobalProps::Add_InstanceEXT_ToEnable()` -> but not static anymore....
⛑
};
```

3. So far, The result:-
   - *4.guide.chapter4.6.newStuffs.hh*
   - *4.guide.chapter4.7.Props.hh*

- *4.guide.chapter4.7.Props.cpp*

7. `vkCreateSwapchainKHR()` 🌱

- https://vkdoc.net/man/vkCreateSwapchainKHR
- [TODO]:- Add the commit-tree Link
- It took me 5days to complete ***Chapter4*** 🏁
    - *(well, i worked on a houdini project 🤕 for 2 days.... so yeah 😅)*

8. `amVK` wrap 2

```
{
    amVK_GlobalProps::EnumerateDeviceExtensionProperties();

    amVK_Device* D = new amVK_Device(amVK_GlobalProps::GetARandom_GPU());
        D->select_QFAM_Graphics();
        D->Add_GPU_EXT_ToEnable("VK_KHR_swapchain");
        D->CreateDevice();
}
```

9. `amVK` wrap 3

```
    // TwT

        REY_LOG("")
    amVK_Surface   *S  = new amVK_Surface(VK_S);
    amVK_Presenter *PR = S->PR;
                           PR->bind_Device(D);
                           PR->create_SwapChain_interface();
                               // This amVK_SwapChain is Bound to this amVK_Surface
    amVK_SwapChain *SC =    PR->SC;
        SC->konf_ImageSharingMode(VK_SHARING_MODE_EXCLUSIVE);
        SC->konf_Images(
            amVK_IF::RGBA_8bpc_UNORM,    // VK_FORMAT_R8G8B8A8_UNORM
            amVK_CS::sRGB,               // VK_COLOR_SPACE_SRGB_NONLINEAR_KHR
            amVK_IU::Color_Display       // VK_IMAGE_USAGE_COLOR_ATTACHMENT_BIT
        );
        SC->konf_Compositing(
            amVK_PM::FIFO,               // VK_PRESENT_MODE_FIFO_KHR
            amVK_CC::YES,                // Clipping:- VK_TRUE
            amVK_TA::Opaque              // VK_COMPOSITE_ALPHA_OPAQUE_BIT_KHR
        );
        SC->sync_SurfCaps();             // refresh/fetch & set/sync ---> latest SurfCaps

        SC->CI.oldSwapchain     = nullptr;
        SC->CreateSwapChain();
```

🏴‍☠️ Part 2: The True Arcane Secrets of

RenderPass

(SubPass + Image Layer Transition) & FrameBuffers

Welcome to the inner sanctum where GPU gods decide how fast your pixels live or die.

- ChatGPT

Chapter 5: `RenderPass` 🏴‍☠️

> " *`subpasses` are the soul of RenderPass!* . But it's not just about `subpasses` only...." - ChatGPT

0. Why `RenderPass` ?

- *"This is one of the most convoluted parts of the Vulkan specification, especially for those who are just starting out."* - P.A. Minerva
- ex. 1:- `PostProcessing Effects`

```
RenderPass:
- color attachment
- depth attachment

subpasses:
- Subpass 0: render geometry
- Subpass 1: post-process effects
    // multiple rendering steps without switching FrameBuffers/AttachMents

// All defined in ONE render pass
```

- ex. 2:- `Deferred Shading`

```
attachments:
- position: offscreen image
- normal: offscreen image
- albedo: offscreen image
- depth: depth image
- finalColor: swapchain image
subpasses:
- Subpass 0: G-buffer generation (write position, normal, albedo)
- Subpass 1: Lighting pass (read G-buffers, write to finalColor)
```

  - *Without `subpasses`, you'd need to switch framebuffers (expensive!).*
  - *With `subpasses`, Vulkan can optimize this by keeping data in GPU memory (especially tile-based GPUs).*

- ex. 3:- `Post-Processing Chain`

```
attachments:
- scene: offscreen image
- postProcessOut: swapchain image
subpasses:
- Subpass 0: scene render → scene
- Subpass 1: post-process → postProcessOut
```

  - *Purpose:- After rendering the main scene, do effects like bloom, blur, or color correction.*
  - *Why a RenderPass?*
    - *Again, Vulkan sees the full plan and can optimize the transitions.*
    - *You can define layout transitions (e.g.* `COLOR_ATTACHMENT_OPTIMAL → SHADER_READ_ONLY_OPTIMAL` *)*

- ex. 4:- `Shadow Map Pass` / Render from light's POV, to a depth-only image

```
attachments:
- depth: depth image
subpasses:
- Subpass 0: write to depth only (no color)
```

  - *Why a RenderPass?*
    - *This pass is often done offscreen, then used as a texture later.*

- ex. 5:- `3D Scene -> Depth Testing`

```
attachments:
- color: swapchain image
- depth: depth image
subpasses:
- Subpass 0:
    - color attachment: color
    - depth attachment: depth
```

1. What is `RenderPass` ? 🚀

1. `RenderPass` **is designed around** `subpasses` **.**
   - *The core purpose of a* `RenderPass` *is to tell Vulkan:*
     - *"Hey, I'm going to do these* ***rendering stages*** *(* `subpasses` *), in this order, using these* ***attachments****."*
   - *So yeah,* `subpasses` *are the main reason for a RenderPass to exist.* ***subpasses are the soul of RenderPass!***
   - *But it's not just about* `subpasses` *only:-*
     a. 🐞 `Load/Store Ops` — *"What should I do with the image before & after rendering?"*
        - 📱 *loadOp — When RenderPass begins:*

```
     LOAD: Keep whatever was already in the attachment.
    CLEAR: Wipe it to a specific value (e.g., clear color to black).
DONT_CARE: Vulkan can throw away old contents (faster, if you don't care).
```

        - 📱 *storeOp — When RenderPass ends:*

```
    STORE: Save the result (e.g., to present to the screen or use later).
DONT_CARE: Vulkan can discard the result (like shadow maps or intermediate stuff
you don't need to read later).
```

        - *ex.*

```
colorAttachment.loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR;
colorAttachment.storeOp = VK_ATTACHMENT_STORE_OP_STORE;
    // Meaning: Clear the image before rendering, and store the result so we can
present it.
```

     b. 🎯 `Image Layout Transitions` — *"How should the GPU access this image during the pass?"*

```
                        [ VK_IMAGE_LAYOUT_UNDEFINED ]
                                    👇 clear
                        [ VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL ]
                                    👇 render
                        [ VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL ]
                                    👇 post-process
                        [ VK_IMAGE_LAYOUT_PRESENT_SRC_KHR ]
```

     c. 📜 `Attachments` — *"What images are we using?"*
        - `RenderPass Attachment` *is not an actual thing!*
        - `RenderPass Attachment Description/Descriptor` *is a thing!*
          - *However, the idea is.... We do "define" the* `Attachments` *right here, as we send the* `AttachmentDescriptions` -> `RenderPass`
        - `RenderPass Attachment` != `FrameBuffer Attachment`
        - `FrameBuffer Attachment`
          - ----> *actual* `VkImageView` *s of* `SwapChain Images`

2. `Image Layout Transitions`

i. *1. Different hardware units = different memory access patterns*

```
GPU Unit                        Access Pattern
------------------      ----------------------
Fragment Shader              Texture-like (random)
Render Output Unit             Tiled or linear (write-heavy)
Compute Shader                 Raw buffer-style
Display Engine                 Linear format

- for ex.
```

- *When an image is used as a **color attachment**, it might be stored tiled in memory for fast write performance.*
- *But when you use the same image as a texture, the shader expects it to be in a format optimized for random read access.*
- *If you tried to read from a tiled format as if it were a texture, you'd either:*
  - *Get `garbage`*
  - *Or pay a `huge perf penalty` as the driver does conversion.... (every single time you access a single pixel) (a single pixel would = an element in an 2D Array) (Texture might have Millions of Pixel)*

ii. *Physical Layout in VRAM (Tiles vs Linear)*

- *Most modern GPUs store **image data** in **tiles** internally.*
  - *(like Z-order, Morton order, or other optimized memory layouts).*
  - *This helps GPUs fetch memory in cache-friendly blocks for faster rendering.*
- *But when an image is to be presented to the **screen/monitor**, it must be **Flat (linear)** (as HDMI/display engines can't decode tiles).*
  - *Yes — by "linear", we mean a simple 2D array where pixels are stored in a straightforward, left-to-right, top-to-bottom format.*
- *So when you do this:-*

```
finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR;
```

  - *What you're really telling Vulkan is:*
    - *"Yo, I'm done rendering — please un-tile this, decompress it, and arrange it in scanlines for display."*
  - *If you don't tell Vulkan, it has to guess or stall — or worse, copy the whole thing behind your back.*

iii. *Transitions let the driver do reordering, compression, or memory reallocation*

```
    //  When you declare:-
finalLayout = VK_IMAGE_LAYOUT_SHADER_READ_ONLY_OPTIMAL;
    // you are not just giving a hint....
    // ---- you are saying:-
```

> - *"After rendering, I'm going to sample this as a texture — so prepare it."*

```
This allows the GPU driver to:
- flush caches
- Decompress the image (some GPUs compress attachments during render!)
- Move memory or restructure tiles
- Or even alias memory with another attachment in a single memory block
- In modern GPUs, there's hardware image compression, like:-
    - ARM's AFBC (Arm Frame Buffer Compression)
    - AMD's DCC (Delta Color Compression)
    - NVIDIA has their own secret sauce too
```

iv. 🌀 *Aliasing & Tile Memory Reuse [`ImageLayouts + Barriers`]*

- *One of the sickest optimizations is this one*
- *You can use the same memory for multiple attachments (e.g. shadow map, depth, HDR buffer), as long as you don't use them at the same time.*
- *But to do that safely, Vulkan needs to know:*
    - *"When does this memory stop being 'render target' and start being 'texture' or 'compute input'?"*

```
Layouts + barriers = safe aliasing.

Drivers can now:
- Use the same memory pool
- Skip clearing
- Not double allocate

You become a GPU memory ninja 🥷🚀
```

v. 📜 *Predictability = Performance*

```
Explicit layouts give Vulkan this power:
- It knows exactly when and how you are going to use an image.
- So it can avoid runtime guessing, which causes:
    - CPU stalls
    - Cache flushes
    - Sync fences
    - Or even full GPU pipeline bubbles 😵

- Compared to OpenGL or DirectX11, where the driver had to guess what you meant and do
hidden magic – Vulkan is like:
```

- *"If you don't tell me what layout you want, I'll trip and fall flat on my face 🤕"*

vi. *You can skip transitions altogether if you do it right*

- *This is the reward -> If your RenderPass is smart — using* `VK_ATTACHMENT_LOAD_OP_DONT_CARE` *and reusing image layouts cleverly — you can avoid layout transitions entirely.*
  - *This is massive for tile-based GPUs (like on mobile phones):*
  - *No layout transition = no VRAM flush*
  - *Everything happens on-chip, like magic*

vii. *Analogy: Baking Cookies*

```
Let's say you're:
- Baking cookies (rendering)
- Then you plate them for display (presenting)
- Later you want to show them off or decorate them (sample in shaders)
```

- *Here's the deal:*

```
Vulkan Image Layout                          Cookie Stage
-------------------------      ---------------------------------
UNDEFINED                                 Empty tray, nothing on it yet
COLOR_ATTACHMENT_OPTIMAL                 You're baking the cookies
SHADER_READ_ONLY_OPTIMAL                 You've finished baking and wanna decorate (like
sampling in a post-process shader)
PRESENT_SRC_KHR                             You're plating the cookies to serve  (sending to
the screen)
```

- *But... here's the twist:*
  - *You can't decorate cookies while they're still baking in the oven.*
  - *And you definitely can't serve someone cookies that are still stuck in a 200°C tray.*
- *So Vulkan says:*
  - *"Please transition between layouts, so I know what stage your cookie is in — and I'll move it to the right place, with oven mitts, spatulas, etc."*

viii. *Why does this matter?*

- *If you don't do the transitions:*

```
You may try to grab a cookie off a 200°C tray and get burned ( invalid reads)
The cookies may not be fully baked ( undefined writes)
Or worse: you show your customer an empty plate because Vulkan never moved them to the
PRESENT_SRC_KHR plate
```

ix. *What makes Vulkan powerful?*

```
You get to say:
1. "Bake in tray A"
2. "Decorate using buffer B"
3. "Present from plate C"

But you must tell Vulkan when to move cookies from one surface to another.
Layouts = telling Vulkan exactly thaaat!
```

x. 🩹 *Subpass Optimization (Tile-Based GPUs)*

```
On tile-based GPUs (like PowerVR or Mali):
- Entire framebuffers live on-chip, in tiles
- You can run all subpasses without touching VRAM!

But it only works if Vulkan knows:
- The image will stay in the same layout
- No unnecessary STORE or layout transitions

By carefully using:
    layout = VK_IMAGE_LAYOUT_ATTACHMENT_OPTIMAL;
    loadOp = DONT_CARE;
    storeOp = DONT_CARE;

You unlock 🥷 zero-cost rendering.
```

- *That's why subpasses and layouts are so closely linked — **no layout change → no memory movement.***

3. 🏁 TL;DR: `Image Layout Transitions` Aren't Just Bureaucracy

```
🎯 They are literal instructions to the driver:
    - "Where this image lives"
    - "How it's structured"
    - "What GPU unit will touch it next"
    - "Whether you need to prepare, flush, decompress, or alias it"

🎯 And by explicitly telling the GPU, you:
    - Avoid expensive guesses
    - Skip hidden memory ops
    - Unlock mobile-level optimizations
    - Prevent subtle bugs and undefined behavior
```

4. 📜 `RenderPass Attachments Desc.`
   - `RenderPass Attachment` *is not an actual thing!*
   - `RenderPass Attachment Description/Descriptor` *is a thing!*
     - *However, the idea is.... We do "define" the* `Attachments` *right here, as we send the* `AttachmentDescriptions` -> `RenderPass`
   - `RenderPass Attachment Description/Descriptors` *are not actual images — they're a template for what the RenderPass expects!*
     - *& The FrameBuffers must delivery* `RenderPass` *exactly with that*

   - `RenderPass Attachment` != `FrameBuffer Attachment`

RenderPass Attachments	*Framebuffers*
Define what is needed	*Provide which resources to use*
Abstract (format, usage, layout)	*Concrete (image views)*
Reusable across Framebuffers	*Swapchain-dependent (often 1:1)*

```
Think of it like a Socket & Plug
    - `RenderPass` 🧱 = The RenderPass defines the socket (shape, voltage).
    - `Framebuffer` 📦 = The Framebuffer provides the plug (actual wires) that fits the
socket.
```

5. 🛍️ `FrameBuffer Attachment`
   - *Actual* `VkImageView`

```
Image Views (VkImageView):
    Handles to specific images (e.g., swapchain images, depth textures).
Compatibility:
    Must match the RenderPass's attachment definitions (format, sample count, size).
Swapchain Link:
    Typically, one Framebuffer per swapchain image.
```

6. 🛒 `FrameBuffers` [🥓🍎🥚🍾]
   - *Binds concrete* `ImageViews` *(e.g.,* `SwapChain Images`, `Depth Textures`*) to the* `attachments` *defined in the* `RenderPass`.
   - *Must match the* `RenderPass's Attachment Descriptions` *(format, size, sample count).*
   - *Is* `SwapChain` *-dependent (e.g., each* `SwapChainImage` *typically has its own* `Framebuffer`*).*
   - *Analogy*

```
- `RenderPass` 🧱 = A recipe requiring "2 eggs and 1 cup of flour" (attachments).
- `Framebuffer` 🛒 = The actual eggs and flour (image views) you use to bake a cake
(render a frame).
```

7. 🎁 `Attachments`

```
RenderPass Attachments (Blueprint)   →       Framebuffer (Instance)
┌--------------------------------┐         ┌------------------------------┐
| Attachment 0: Color (SRGB)     |         | Image View 0: Swapchain IMG |
| Attachment 1: Depth (D32_SF)   |         | Image View 1: Depth Texture |
└--------------------------------┘         └------------------------------┘
```

- *Attachments are simply images (or buffers) where Vulkan stores or reads data during a RenderPass.*
- *Attachments are the actual framebuffer images (swapchain images, depth buffers, offscreen render targets, etc.)*

i. 🖌 *Color Attachments = where the pretty pixels (RGBA) are painted and stored. This is like your paint palette!* 🎨
ii. 🏔 *Depth Attachments = the landscapes that prevent objects from clipping or showing up out of order. Imagine topography maps for depth!* 🗺
iii. 🖍 *Stencil Attachments = the guides that show where we can paint, like drawing a "map" where only certain areas can be modified.* 🗺

- *What's inside?*
  - *A framebuffer that stores things like RGBA values (Red, Green, Blue, Alpha/Transparency).*
  - *For example,*
    - *Color Attachment 0 might hold the albedo or the final color of an object, while*
    - *Color Attachment 1 could store the lighting information or additional passes like ambient occlusion.*

```
Each attachment you declare includes:
- Format (VK_FORMAT_B8G8R8A8_SRGB, etc.)
- Sample count (for MSAA)
- Load/store ops
- Layouts (see above)
```

- *Then, each subpass tells Vulkan:*

  > - *"From all the attachments I've declared, I'm gonna use these ones in this subpass."*

- *in Code:*

```
attachments[0] = colorAttachment;   // swapchain image
attachments[1] = depthAttachment;   // depth image

subpass.colorAttachment = &attachments[0];
subpass.depthAttachment = &attachments[1];
```

```
So even if your RenderPass only has one subpass, the Vulkan driver still wants to know:
- How many attachments
- What to do with them (clear/store?)
- What layouts they go into and come out as
```

8. `FrameBuffers` v/s `Attachments` :- The Last Fight, (If Above stuffs got you confused):-

i. Quick Comparison Table

Aspect	*Attachments (RenderPass)* ❌	*Framebuffers* 🖼️
Purpose	*Define what resources are needed (format, usage, layout transitions)* 📝	*Specify which actual images (image views) to use for those resources* 🖍️
Concrete/ Abstract	*Abstract (blueprint)* 📋	*Concrete (instance)* 🏗️
Lifetime	*Long-lived (reused across frames)* ♻️	*Short-lived (often recreated with swapchain)* ↻
Dependencies	*Independent of images/swapchain* ⛔ 🖼️	*Tied to swapchain images or specific textures* 🔗
Example	*"Need a color attachment (SRGB) and depth attachment (D32_SFLOAT)"* 🎨 ➕ 🎱	*"Use this swapchain image and that depth texture"* 🖼️ 🖼️ 🖼️

ii. Lifecycle Flowchart

```
RenderPass Creation 🏗️
│
├── Define Attachments 📄 (format, load/store ops, layouts)
│   └── "I need a color slot (B8G8R8A8_SRGB) and depth slot (D32_SFLOAT)" 🎨🎱
│
└── Define Subpasses 🔗 (how attachments are used in rendering steps)

↓

Framebuffer Creation 🛠️
│
├── Bind Image Views to Attachment Slots 🔌
│   └── "Slot 0 = Swapchain Image View 🖼️, Slot 1 = Depth Texture View 📦"
│
└── Validate Compatibility ☑️ (size, format, sample count)
```

iii. Use-Case Scenarios

Scenario	*Attachments (RenderPass)* 🧩	*Framebuffers* 🖼️
Swapchain Rendering	*Define color/depth formats and layouts.* 🎨 🔄 🎱	*Bind swapchain images + depth texture.* 🖼️ 🔗 📦
Deferred Rendering	*Define G-Buffer attachments (Albedo, Normal, Position).* 📦 📦 📦	*Bind actual G-Buffer image views.* 🖼️ 🖼️ 🖼️ 🖼️
Post-Processing	*Define input (e.g., HDR color) + output (e.g., SRGB).* ✨ ➡️ 🎨	*Bind input texture + swapchain image.* 🧩 🖼️

iv. *Key Interactions*

```
RenderPass Begin Command 🚦
│
├── Uses RenderPass Attachments 🧩 (format, load/store rules)
│
└── Uses Framebuffer 🖼️ (actual images to write to)

↓

GPU Renders 🎮
│
├── Reads/Writes to Framebuffer's Image Views 📊
│
└── Follows Attachment Rules (clearing, layout transitions) 🔄
```

v. *Emoji Analogy Time!* 🤯

- *Attachments = Recipe Ingredients List 📜 (e.g., "2 eggs 🥚🥚, 1 cup flour 🍚").*
- *Framebuffers = Actual Ingredients 🛒 (e.g., "This egg 🥚 from the fridge, that flour 🍚 from the pantry").*
- *Rendering = Baking the Cake 🎂 (combine them using the recipe steps!).*

9. Next Chapter will be on 🛒 `FrameBuffers` !!!! 🔷

Everything above is written with help from chatGPT

Everything below is not!

0. `amVK` Wrap ☺

```
// TwT
    SC->GetSwapChainImagesKHR();
    SC->CreateSwapChainImageViews();

amVK_RenderPass *RP = PR->create_RenderPass_interface();
```

2. `vkCreateRenderPass()`

- https://vkdoc.net/man/vkCreateRenderPass
- **REY_DOCs**
  - *Copy Paste* `amVK_SwapChain.hh` *Current Implementation & Change it as needed*
    - *Trust me, this is the most fun way of doing this, xP*

3. `VkRenderPassCreateInfo()`

- https://vkdoc.net/man/VkRenderPassCreateInfo
  - `.flags` -> *Only Option:- used for Qualcom Extension*
  - `.pAttachments` -> *this->SubChapter4*
  - `.pSubpasses` -> *this->SubChapter5*
  - `.pDependencies` -> *this->SubChapter6*

4. `ImageViews`

1. `vkGetSwapchainImagesKHR()`
   - *https://vkdoc.net/man/vkGetSwapchainImagesKHR*
   - *Implement Exactly like Chapter2.5* ☺
     - `vkGetPhysicalDeviceQueueFamilyProperties()`
   - **REY_DOCs**

```
class amVK_SwapChain  {
    ...
  public:
    amVK_Device *D = nullptr;
    VkSwapchainKHR SC = nullptr;
    REY_Array<VkImage>    amVK_1D_SC_IMGs;
    REY_Array<amVK_Image> amVK_1D_SC_IMGs_amVK_WRAP;
    bool called_GetSwapchainImagesKHR = false;

  public:
    ...
```

2. `vkCreateImageView()`
   - *https://vkdoc.net/man/vkCreateImageView*
   - **REY_DOCs**

```
void CreateSwapChainImageViews(void) {
    REY_Array_LOOP(amVK_1D_SC_IMGs_amVK_WRAP, i) {
        amVK_1D_SC_IMGs_amVK_WRAP[i].createImageView();
    }
}
```

     - `amVK_Image.hh` :- *4.guide.chapter5.3.2.Image.hh*

3. `VkImageViewCreateInfo`
    - *https://vkdoc.net/man/VkImageViewCreateInfo*
    - **REY_DOCs**

```
void amVK_SwapChain::CreateSwapChainImageViews(void) {
    REY_Array_LOOP(amVK_1D_SC_IMGs_amVK_WRAP, i) {
            // ViewCI.image
            // ViewCI.format
                // should be set inside amVK_SwapChain::GetSwapchainImagesKHR()
        amVK_1D_SC_IMGs_amVK_WRAP[i].ViewCI.viewType = VK_IMAGE_VIEW_TYPE_2D;

        amVK_1D_SC_IMGs_amVK_WRAP[i].ViewCI.components = { // Equivalent to:
            VK_COMPONENT_SWIZZLE_R,         // VK_COMPONENT_SWIZZLE_IDENTITY
            VK_COMPONENT_SWIZZLE_G,         // VK_COMPONENT_SWIZZLE_IDENTITY
            VK_COMPONENT_SWIZZLE_B,         // VK_COMPONENT_SWIZZLE_IDENTITY
            VK_COMPONENT_SWIZZLE_A          // VK_COMPONENT_SWIZZLE_IDENTITY
        };

        amVK_1D_SC_IMGs_amVK_WRAP[i].ViewCI.subresourceRange.aspectMask =
VK_IMAGE_ASPECT_COLOR_BIT;
        amVK_1D_SC_IMGs_amVK_WRAP[i].ViewCI.subresourceRange.baseMipLevel = 0;
        amVK_1D_SC_IMGs_amVK_WRAP[i].ViewCI.subresourceRange.levelCount = 1;
        amVK_1D_SC_IMGs_amVK_WRAP[i].ViewCI.subresourceRange.baseArrayLayer = 0;
        amVK_1D_SC_IMGs_amVK_WRAP[i].ViewCI.subresourceRange.layerCount = 1;

        amVK_1D_SC_IMGs_amVK_WRAP[i].createImageView();
    }
}
```

5. `VkAttachmentDescription`

- https://vkdoc.net/man/VkAttachmentDescription

6. `VkSubpassDescription`

- https://vkdoc.net/man/VkSubpassDescription

7. `VkSubpassDependency`

- https://vkdoc.net/man/VkSubpassDependency

8. All the last 3 together ---> Code

```
class amVK_RenderPass  {
  public:
    REY_ArrayDYN<VkAttachmentDescription> attachments;
    REY_ArrayDYN<VkSubpassDescription> subpasses;
    REY_ArrayDYN<VkSubpassDependency> dependencies;

    void set_attachments_subpasses_dependencies(void);
}
```

- `amVK_RenderPass.hh` [Full Implementation]:- 4.guide.chapter5.8.RenderPass.hh

```
amVK_RenderPass *RP = new amVK_RenderPass(D);
    RP->attachments.push_back({
        .format = SC->CI.imageFormat,                       // Use the color format selected by the
swapchain
        .samples = VK_SAMPLE_COUNT_1_BIT,                   // We don't use multi sampling in this
example
        .loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR,              // Clear this attachment at the start of
the render pass
        .storeOp = VK_ATTACHMENT_STORE_OP_STORE,
            // Keep its contents after the render pass is finished (for displaying it)
        .stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE,
            // Similar to loadOp, but for stenciling (we don't use stencil here)
        .stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE,
            // Similar to storeOp, but for stenciling (we don't use stencil here)
        .initialLayout = VK_IMAGE_LAYOUT_UNDEFINED,
            // Layout at render pass start. Initial doesn't matter, so we use undefined
        .finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR,
            // Layout to which the attachment is transitioned when the render pass is finished
            // As we want to present the color attachment, we transition to PRESENT_KHR
    });

    VkAttachmentReference colorReference = {
        .attachment = 0,
        .layout = VK_IMAGE_LAYOUT_COLOR_ATTACHMENT_OPTIMAL
    };
    RP->subpasses.push_back({
        .pipelineBindPoint = VK_PIPELINE_BIND_POINT_GRAPHICS,
```

```
        .inputAttachmentCount = 0,
            // Input attachments can be used to sample from contents of a previous subpass
        .pInputAttachments = nullptr,           // (Input attachments not used by this example)
        .colorAttachmentCount = 1,              // Subpass uses one color attachment
        .pColorAttachments = &colorReference,   // Reference to the color attachment in slot 0

        .pResolveAttachments = nullptr,
            // Resolve attachments are resolved at the end of a sub pass and can be used for e.g.
multi sampling
        .pDepthStencilAttachment = nullptr,     // (Depth attachments not used by this sample)
        .preserveAttachmentCount = 0,
            // Preserved attachments can be used to loop (and preserve) attachments through subpasses
        .pPreserveAttachments = nullptr         // (Preserve attachments not used by this example)
    });

    RP->dependencies.push_back({
        // Setup dependency and add implicit layout transition from final to initial layout for the
color attachment.
        // (The actual usage layout is preserved through the layout specified in the attachment
reference).
        .srcSubpass = VK_SUBPASS_EXTERNAL,
        .dstSubpass = 0,
        .srcStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT,
        .dstStageMask = VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT,
        .srcAccessMask = VK_ACCESS_NONE,
        .dstAccessMask = VK_ACCESS_COLOR_ATTACHMENT_WRITE_BIT | VK_ACCESS_COLOR_ATTACHMENT_READ_BIT,
    });

    RP->set_attachments_subpasses_dependencies();
    RP->createRenderPass();

    ------------------------------------ Made with help from P.A.Minerva Vulkan Guide
-------------------------------------
    https://paminerva.github.io/docs/LearnVulkan/01.A-Hello-Window#416---creating-a-render-pass
```

- `main.cpp` [Full Implementation]:- 4.guide.chapter5.8.main.cpp

9. By This time, VkSurfaceKHR deserves it's very own class `amVK_Surface`

- `amVK_Surface.hh` [Full Implementation]:- 4.guide.chapter5.9.Surface.hh

Chapter 6: `amVK_ColorSpace.hh`, `amVK_Surface`, `amVK_Presenter`, Renaming Things in amVK

1. `amVK_ColorSpace.hh`

```
/**
 * ex. 1   amVK_IF::RGBA_8bpc_UNORM
 */
namespace amVK_ImageFormat {
    // 8bpc = 8-bits per channel
    inline constexpr VkFormat RGBA_8bpc_UNORM    = VK_FORMAT_R8G8B8A8_UNORM;    // 37
    inline constexpr VkFormat RGBA_8bpc_SNORM    = VK_FORMAT_R8G8B8A8_SNORM;    // 38
    inline constexpr VkFormat RGBA_8bpc_USCALED  = VK_FORMAT_R8G8B8A8_USCALED;  // 39
    inline constexpr VkFormat RGBA_8bpc_SSCALED  = VK_FORMAT_R8G8B8A8_SSCALED;  // 40
    inline constexpr VkFormat RGBA_8bpc_UINT     = VK_FORMAT_R8G8B8A8_UINT;     // 41
    inline constexpr VkFormat RGBA_8bpc_SINT     = VK_FORMAT_R8G8B8A8_SINT;     // 42
    inline constexpr VkFormat RGBA_8bpc_SRGB     = VK_FORMAT_R8G8B8A8_SRGB;     // 43

    // Common Depth/Stencil Formats
    inline constexpr VkFormat D32_SFLOAT         = VK_FORMAT_D32_SFLOAT;
    inline constexpr VkFormat D24_UNORM_S8_UINT  = VK_FORMAT_D24_UNORM_S8_UINT;
}
#define amVK_IF amVK_ImageFormat
#define amVK_PF amVK_ImageFormat
#define amVK_PixelFormat amVK_ImageFormat
```

- Entire Code:- amVK_ColorSpace.hh

2. `amVK_Surface`

```
/**
 * VULKAN-EXT:- `VK_KHR_surface`
 *       IMPL:- `amVK_1D_SurfaceInfos`
 */
class amVK_Surface {
  public:
    VkSurfaceKHR S = nullptr;       // Set in CONSTRUCTOR
    amVK_Presenter *PR = nullptr;   // Set in CONSTRUCTOR

    amVK_Surface(void) {}
    amVK_Surface(VkSurfaceKHR pS);

                REY_Array<REY_Array<VkSurfaceFormatKHR>>            amVK_2D_GPUs_ImageFMTs;
                REY_Array<VkSurfaceCapabilitiesKHR>                 amVK_1D_GPUs_SurfCAP;

    bool called_GetPhysicalDeviceSurfaceFormatsKHR = false;
    bool called_GetPhysicalDeviceSurfaceCapabilitiesKHR = false;
```

```
    void        GetPhysicalDeviceSurfaceInfo(void);
    void        GetPhysicalDeviceSurfaceCapabilitiesKHR(void);
};
```

- Entire Code:- 4.guide.chapter6.3.Surface.hh

3. `amVK_Presenter`

```
class amVK_Presenter {
  public:
    amVK_Surface  *S  = nullptr;
    amVK_SwapChain *SC = nullptr;
    amVK_RenderPass *RP = nullptr;
        //   SC.VkDevice = RP.VkDevice
    amVK_Device        *D = nullptr;
    VkPhysicalDevice   GPU = nullptr;
        // amVK_Device.m_PD = this->GPU;
    amVK_GPU_Index GPU_Index = 0;

  public:
    void bind_Device(amVK_Device *D);
    amVK_Presenter  (amVK_Surface* pS) {this->S = pS;}

  public:
    amVK_SwapChain*  create_SwapChain(void);
    amVK_RenderPass* create_RenderPass(void);
    // Defined currently inside amVK_SwapChain.cpp

    void                      refresh_SurfCaps(void) {
this->S->GetPhysicalDeviceSurfaceCapabilitiesKHR(); }
    VkSurfaceCapabilitiesKHR* fetched_SurfCaps(void) {
        return &( this->S->amVK_1D_GPUs_SurfCAP[this->GPU_Index] );
    }
};
```

- Entire Code:- 4.guide.chapter6.3.Surface.hh

4. `amVK` Naming Conventions ☺

1. Calling Vulkan Library Functions:-

```
bool called_GetPhysicalDeviceSurfaceFormatsKHR = false;
bool called_GetPhysicalDeviceSurfaceCapabilitiesKHR = false;
void        GetPhysicalDeviceSurfaceInfo(void);
void        GetPhysicalDeviceSurfaceCapabilitiesKHR(void);
```

2. `vkCreateZZZ()` wrappers

```
amVK_SwapChain {
    void CreateSwapChain(void) {
        VkResult return_code = vkCreateSwapchainKHR(this->D->m_device, &CI, nullptr, &this->SC);
        amVK_return_code_log( "vkCreateSwapchainKHR()" );     // above variable "return_code"
can nott be named smth else
    }
}
```

3. `amVK_Object` /Instance-Creation

```
amVK_SwapChain* amVK_Presenter::create_SwapChain(void);
```

4. `amVK_Object::Functions()`

```
amVK_SwapChain*   create_SwapChain(void);           // Creates amVK_Object
amVK_RenderPass*  create_RenderPass(void);          // Creates amVK_Object
void                      refresh_SurfCaps(void);   // SurfCapabilities changes if Window is
Resized
VkSurfaceCapabilitiesKHR* fetched_SurfCaps(void);   // Returns the REFRESHED/FETCHED element

void            amVK_SwapChain::sync_SurfCaps(void);/** Refreshes & Syncs `SurfaceCapabilites`
*/
void            amVK_SwapChain::konf_Images(
    VkFormat IF,
    VkColorSpaceKHR CS,
    VkImageUsageFlagBits IU,
    bool autoFallBack = true
)
void            amVK_SwapChain::konf_Compositing(
    VkPresentModeKHR PM,
    amVK_CompositeClipping CC,
    VkCompositeAlphaFlagBitsKHR CA
);
void            amVK_SwapChain::konf_ImageSharingMode(VkSharingMode ISM);
VkFormat        amVK_SwapChain::active_PixelFormat(void)                    {return
CI.imageFormat;}
VkColorSpaceKHR amVK_SwapChain::active_ColorSpace (void)                    {return
CI.imageColorSpace;}
```

5. `VkObject` Variables

```
class amVK_Image {
  public:
    amVK_Device *D = nullptr;
    VkImage     vk_Image = nullptr;
    VkImageView vk_ImageView = nullptr;
};

class amVK_FrameBuffer {
  public:
    amVK_Presenter *PR = nullptr;       // Basically, Parent Pointer
    VkFramebuffer vk_FrameBuffer = nullptr;
};

class amVK_RenderPass {
  public:
    amVK_Presenter *PR = nullptr;       // Basically, Parent Pointer
    VkRenderPass vk_RenderPass = nullptr;
};

class amVK_Surface {
  public:
    amVK_Presenter *PR = nullptr;   // Created in CONSTRUCTOR
    VkSurfaceKHR vk_SurfaceKHR = nullptr;   //     Set in CONSTRUCTOR
}
```

4. amVK_RenderPass_Descriptors.hh

```
namespace amVK_RP_AttachmentDescription
{
        // Change .format before using
    inline VkAttachmentDescription ColorPresentation = {
        .format = VK_FORMAT_UNDEFINED ,                     // you should Use the color format
selected by the swapchain
        .samples = VK_SAMPLE_COUNT_1_BIT,                   // We don't use multi sampling in this
example
        .loadOp = VK_ATTACHMENT_LOAD_OP_CLEAR,              // Clear this attachment at the start of
the render pass
        .storeOp = VK_ATTACHMENT_STORE_OP_STORE,            // Keep its contents after the render
pass is finished (for displaying it)
        .stencilLoadOp = VK_ATTACHMENT_LOAD_OP_DONT_CARE,   // Similar to loadOp, but for stenciling
(we don't use stencil here)
        .stencilStoreOp = VK_ATTACHMENT_STORE_OP_DONT_CARE, // Similar to storeOp, but for stenciling
 (we don't use stencil here)
        .initialLayout = VK_IMAGE_LAYOUT_UNDEFINED,         // Layout at render pass start. Initial
doesn't matter, so we use undefined
        .finalLayout = VK_IMAGE_LAYOUT_PRESENT_SRC_KHR,     // Layout to which the attachment is
transitioned when the render pass is finished
```

```
                                                    // As we want to present the color
attachment, we transition to PRESENT_KHR
    };
};

#define amVK_RPADes amVK_RP_AttachmentDescription
#define amVK_RPARef amVK_RP_AttachmentReference
#define amVK_RPSDes amVK_RP_SubpassDescription
#define amVK_RPSDep amVK_RP_SubpassDependency

...
```

- You should kinda check the `amVK_RenderPass_Descriptors.hh` file yourself

```
amVK_RenderPass *RP = PR->create_RenderPass_interface();
    amVK_RPADes::ColorPresentation.format = SC->CI.imageFormat;

    RP->AttachmentInfos .push_back(amVK_RPADes::ColorPresentation);
    RP->SubpassInfos    .push_back(amVK_RPSDes::ColorPresentation);
    RP->Dependencies    .push_back(amVK_RPSDep::ColorPresentation);

    RP->sync_Attachments_Subpasses_Dependencies();
    RP->CreateRenderPass();
```

Chapter 7: 🛒 `FrameBuffer` [🫳🍎🥚🦴]

1. `vkCreateFramebuffer()`

- https://vkdoc.net/man/vkCreateFramebuffer
- **REY_DOCs**
  - *Copy Paste* `amVK_RenderPass.hh` *Current Implementation & Change it as needed*
    - *Trust me, this is the most fun way of doing this, xP*

2. `VkFramebufferCreateInfo()`

- https://vkdoc.net/man/VkFramebufferCreateInfo
  - `.flags` -> *Only Option:-* `VK_FRAMEBUFFER_CREATE_IMAGELESS_BIT`
  - `.renderPass` -> 😳
  - `.pAttachments` -> *[VkImageView]* this->*SubChapter3*
  - `.width` ->
  - `.height` ->
  - `.layers` ->
- **REY_DOCs**
  - *Start With basic copy paste of* `amVK_RenderPass.hh` *:- 4.guide.chapter7.2.FrameBuffer.hh*

3. `VkImageView .pAttachments`

- https://vkdoc.net/man/VkImageView
  - *For Now, We are gonna choose 1 VkImageView per FrameBuffer*
- **REY_DOCs**

```
#include "amVK_FrameBuffer.hh"
void amVK_Presenter::create_FrameBuffers(void) {
    this->FBs.reserve(this->SC->amVK_1D_SC_IMGs.n);

    REY_Array_LOOP(this->FBs, k) {
        amVK_FrameBuffer* FB = new amVK_FrameBuffer(this);

            FB->CI.attachmentCount = 1;
            FB->CI.pAttachments = &(this->SC->amVK_1D_SC_IMGs_amVK_WRAP[k].vk_ImageView);

            FB->CI.width = 0;
            FB->CI.height = 0;

            FB->CreateFrameBuffer();

        this->FBs[k] = FB;
    }
}
```

Chapter 8: `CommandBuffer`

1. `vkCreateCommandPool()`
   - https://vkdoc.net/man/vkCreateCommandPool
   - **REY_DOCs**
     - *Copy Paste* `amVK_FrameBuffer.hh` *Current Implementation & Change it as needed*
       - *Trust me, this is the most fun way of doing this, xP*

2. `VkCommandPoolCreateInfo`
   - https://vkdoc.net/man/VkCommandPoolCreateInfo

3. `vkAllocateCommandBuffers()`
   - https://vkdoc.net/man/vkAllocateCommandBuffers

4. `VkCommandBufferAllocateInfo`
   - https://vkdoc.net/man/VkCommandBufferAllocateInfo

1. The Rendering Loop

- https://paminerva.github.io/docs/LearnVulkan/01.A-Hello-Window#42---the-rendering-loop