Chapter 13: `RenderLoop`

`Vulkan` wrap

```
while(true) {
    vkAcquireNextImageKHR();

    vkResetCommandBuffer();     // req:- VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT
    vkBeginCommandBuffer();
        vkCmdBeginRenderPass();
            vkCmdSetViewport();
            vkCmdSetScissor();
            vkCmdBindPipeline();
                vkCmdDraw();
        vkCmdEndRenderPass();
    vkEndCommandBuffer();

    vkQueueSubmit();
    vkQueuePresentKHR();

    vkQueueWaitIdle();
    PROCESS_InputEvents();
    REY_NoobTimer::wait(10); // wait 10ms
}
```

`amVK` wrap

```
while(true) {
    W->dispatch_events_with_OSModalLoops();

    RP_FBs->RPBI_AcquireNextFrameBuffer();

    CB->BeginCommandBuffer(amVK_Sync::CommandBufferUsageFlags::Submit_Once);
    // ------------------------ CommandBufferRecording ----------------------------

        RP_FBs->CMDBeginRenderPass(CB->vk_CommandBuffer);
            RP_FBs->CMDSetViewport_n_Scissor(CB->vk_CommandBuffer);
            PLG->CMDBindPipeline(CB->vk_CommandBuffer);
                VB.CMDDraw(CB->vk_CommandBuffer);
        RP_FBs->CMDEndRenderPass(CB->vk_CommandBuffer);

    // ------------------------ CommandBufferRecording ----------------------------
    CB->EndCommandBuffer();

    // This was Done Before in CH4 : SwapChain
        amVK_SurfacePresenter  *PR = new amVK_SurfacePresenter();
                                PR->bind_Surface(S);
                                PR->bind_Device(D);
    // This was Done Before in CH4 : SwapChain

    PR->set_CommandBuffer(CB->vk_CommandBuffer);
    PR->submit_CMDBUF(D->Queues.GraphicsQ(0));
    PR->Present(D->Queues.GraphicsQ(0));

    vkQueueWaitIdle(D->Queues.GraphicsQ(0));
    REY_NoobTimer::wait(10); // wait 10ms
}
```

1. `vkAcquireNextImageKHR()`

- *https://vkdoc.net/man/vkAcquireNextImageKHR*
  - `.device` *Same as* `SwapChain`
  - `.swapchain`
  - `.timeout` `nanoseconds`
    - *specifies how long the function waits, in* `nanoseconds`*, if no image is available.*

      ```
      uint64_t ns_per_second = 1'000'000'000;
      ```
  - `.semaphore` *SubChapter 2*
  - `.fench` ChapterZZZ
  - `.pImageIndex`
    - *Well, this function doesn't return an* `VkImage` *but an index to it*

- 📜 *REY_DOCS*
  - Acquires the next image from SwapChain
  - *So, now you know which class this function has got to be inside* 😱

- ↩ *Return Codes*
  - 🔁 `VK_SUBOPTIMAL_KHR`
    - *if the window has been resized but the OS/platform's* `GPU-DriverImplementation` / `PresentationEngine` *is still able to scale the presented images to the new size to produce valid surface updates.*
    - *It is up to the application to decide whether it prefers to continue using the current swapchain in this state, or to re-create the swapchain to match resized window.*
  - 🔁 `VK_ERROR_OUT_OF_DATE_KHR`
    - *the images in the swapchain no longer matches the surface properties (e.g., the window was resized)*
    - *and the presentation engine can't present them,*
    - *so the application needs to create a new swapchain that matches the surface properties.*
  - *REFs:- 1. minerva*

- 📜 *REY_DOCs*
  - *For rendering, we need an image to render on to.*
  - *And before we start rendering onto an image, vulkan wants us to call* `AcquireNextImage()` *from* `SwapChain`

1.1. Can you use `1` SwapChainIMG? 🤦‍♀️

1. `1 IMG Modes`
   - `VK_PRESENT_MODE_IMMEDIATE_KHR`
     - *You will see* Tearing
     - `vkAcquireNextImageKHR()` *:- Returns* `VK_SUCCESS` *immediately (no synchronization).*
   - `VK_PRESENT_MODE_FIFO_RELAXED_KHR`
     - *If you submit frames slower than the display refresh rate, it might work with 1 image (but risky).*
2. `Multi IMG Modes`
   - `VK_PRESENT_MODE_FIFO_KHR` *a.k.a* VSync *:- Needs ≥2 images (front + back buffer).*
   - `VK_PRESENT_MODE_MAILBOX_KHR` *(*Triple buffering*) :- Needs ≥3 images.*
     - *Some games will also flag this as* VSync
3. *GPU Driver:-*

```
vkGetPhysicalDeviceSurfaceCapabilitiesKHR(...);
// caps.minImageCount is often 2+ (driver may ignore your request when creating SwapChain)
```

1.2. `vkQueueWaitIdle()`

- *https://vkdoc.net/man/vkQueueWaitIdle*
- 📜 *REY_DOCs*
  - *So that we can Acquire the next image without causing errors.*

1.3. Synchronization

2. Command Recording I

- *Recall Back :* 🛠 `VkCommandPoolCreateInfo`
  - https://vkdoc.net/man/VkCommandPoolCreateInfo
    - `.flags` ▭ `VkCommandPoolCreateFlagBits`
      - https://vkdoc.net/man/VkCommandPoolCreateFlagBits | ivirtex-github
        - ▭ `TRANSIENT`
        - ▭ `RESET_COMMAND_BUFFER` :- *Lets you call* `vkBeginCommandBuffer()` *on same* `CMDBUF` *more than once*
        - ▭ `PROTECTED`
      - ▨ `0` :- *Can't call* `vkBeginCommandBuffer()` *more than once on the same* `CMDBUF`

1. `VkCommandBufferBeginInfo`

- https://vkdoc.net/man/VkCommandBufferBeginInfo
  - `.sType` ▨ `VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO`
  - `.pNext` ▨ `nullptr`
    - ⌀ `VkDeviceGroupCommandBufferBeginInfo`
  - `.flags` 🔡 `VkCommandBufferUsageFlagBits`
    - https://vkdoc.net/man/VkCommandBufferUsageFlagBits | ivirtex-github
      - 🔡 `SIMULTANEOUS_USE`
      - 🔡 `RENDER_PASS_CONTINUE` *[secondary command buffer]*
      - 🔡 `ONE_TIME_SUBMIT` :- *usually coupled with* ▭ `RESET_COMMAND_BUFFER`
    - ▨ `0` :- *Now you can submit this command buffer multiple times*
  - `.pInheritanceInfo` ⌀ *[secondary command buffer]*

- 📜 REY_DOCs
  - *Rendering commands have to be Recorded in a CommandBuffer.*
  - *Only then the GPU can work on it* 💁.
  - *That's the idea, since decades ago, so yeah, xD.*

- 📜 REY_DOCs : There are a few ways that you can record `CMDBUF`
  a. *Recording a* `VkCommandBuffer` *only Once*
  b. *Recording a* `VkCommandBuffer` *more than Once*
     - *more than Once --> requires,* `CMDBUF` *to be reset by* `vkResetCommandBuffer()` *before recording again*
       - *Note:-* `vkBeginCommandBuffer()` *also does do an implicit reset*
       - *which I don't believe should exist* 💁
       - *I always believe* ***"Explicit is better than implicit"***
     - *REQ:-*
       - `VkCommandPoolCreateInfo.flags` ▨ `VK_COMMAND_POOL_CREATE_RESET_COMMAND_BUFFER_BIT`
  c. *Submitting a* `VkCommandBuffer` *only Once*
     - *REQ:-*
       - `VkCommandBufferBeginInfo.flags` ▨ `VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT`
     - *People usually uses* `SUBMIT_ONCE` *with a* `RESET_CMDBUF`
  d. *Implementing Synchronization features is so fked up. After Present Image, call* `vkQueueWaitIdle()`

2. `VkRenderPassBeginInfo`

- https://vkdoc.net/man/VkRenderPassBeginInfo
  - `.sType` ▨ `VK_STRUCTURE_TYPE_RENDER_PASS_BEGIN_INFO`
  - `.pNext` ⌀ `nullptr`
  - `.renderPass` ▨ 💁
  - `.framebuffer` ▨ 💁
  - `.renderArea` ➡ https://vkdoc.net/man/VkRect2D
  - `.pClearValues` ➡ https://vkdoc.net/man/VkClearValue

- 📜 REY_DOCs
  - *After hitting Start on the* `RecordCommandBuffer()`
  - *Generally the first thing we gotta do is call* `BeginRenderPass()`
  - *So that we can do & transition between the* `DepthPass`, `NormalPass`, `ShadowPass`, `AlbedoPass`, *other necessary subpasses*

2. Command Recording II

3. `vkBeginCommandBuffer()`

- https://vkdoc.net/man/vkBeginCommandBuffer
  - `.commandBuffer`
  - `.pBeginInfo`
- `</> TheCode`

```
amVK_CommandPool {
  public:
    REY_Array<VkCommandBuffer>      vk_CommandBuffers;
    REY_Array<VkCommandBuffer> AllocateCommandBuffers(void);

  public:
    VkCommandBufferBeginInfo BI = {
        .sType = VK_STRUCTURE_TYPE_COMMAND_BUFFER_BEGIN_INFO,
        .pNext = 0,
        .flags = VK_COMMAND_BUFFER_USAGE_ONE_TIME_SUBMIT_BIT,
            // People usually uses `SUBMIT_ONCE` with a `RESET_CMDBUF` & records CMDBUF every frame
        .pInheritanceInfo = nullptr
    };
    void BeginCommandBuffer(uint32_t CMDBUF_Index) {
        VkResult return_code = vkBeginCommandBuffer(vk_CommandBuffers[CMDBUF_Index], &BI);
        amVK_return_code_log( "vkBeginCommandBuffer()" );
    }
}
```

4. `vkCmdBeginRenderPass()`

- https://vkdoc.net/man/vkCmdBeginRenderPass
  - `.commandBuffer`
  - `.pRenderPassBegin`
  - `.contents` `VK_SUBPASS_CONTENTS_INLINE`
    - https://vkdoc.net/man/VkSubpassContents | ivirtex-github
      - `INLINE`
      - `SECONDARY_COMMAND_BUFFERS` *[secondary command buffer]*
      - `INLINE_AND_SECONDARY_COMMAND_BUFFERS_KHR` *[VK_KHR_maintenance7]*
      - `INLINE_AND_SECONDARY_COMMAND_BUFFERS_EXT` *[VK_EXT_nested_command_buffer]*

---

5. `vkCmdSetViewport()`

- https://vkdoc.net/man/vkCmdSetViewport
  - `.commandBuffer`
  - `.firstViewport` `0`
  - `.viewportCount` `1`
  - `.pViewports` `VkViewport`
    - https://vkdoc.net/man/VkViewport

- REY_DOCs
  - *This is smth i consider part of the RenderPass*

6. `vkCmdSetScissor()`

- https://vkdoc.net/man/vkCmdSetScissor
  - `.pScissors` `VkRect2D`
    - https://vkdoc.net/man/VkRect2D

- REY_DOCs
  - *This is smth i consider part of the RenderPass*

7. `vkBindPipeline()`
   - https://vkdoc.net/man/vkBindPipeline
     - `.commandBuffer`

   - REY_DOCs
     - *Once we bind a pipeline, then we can call* `Draw()`
     - *After binding pipeline, is where you specify* `VertexBuffers`
       - *More on VertexBuffer* Chapter16
8. `vkCmdDraw()`
   - https://vkdoc.net/man/vkCmdDraw

---

9. `vkCmdEndRenderPass()`
   - https://vkdoc.net/man/vkCmdEndRenderPass
     - `.commandBuffer`
10. `vkEndCommandBuffer()`
    - https://vkdoc.net/man/vkEndCommandBuffer
      - `.commandBuffer`

3. `Submit Command Buffer` & Present

1. `VkSubmitInfo`
   - https://vkdoc.net/man/VkSubmitInfo
     - `.sType` `VK_STRUCTURE_TYPE_SUBMIT_INFO`
     - `.pNext` `NULL`
     - `.pWaitSemaphores` *Chapter9.1*
       - `amVK_SwapChain::AcquireNextImage_SemaPhore`
     - `.pWaitDstStageMask` `VK_PIPELINE_STAGE_COLOR_ATTACHMENT_OUTPUT_BIT`
     - `.pCommandBuffers`
     - `.pSignalSemaphores`
       - `amVK_SurfacePresenter::RenderingFinished_SemaPhore`

---

2. `vkQueueSubmit()`
   - https://vkdoc.net/man/vkQueueSubmit
     - `.queue` `GraphicsQueue`
     - `.submitCount` `1`
     - `.pSubmits`
     - `.fench` `VK_NULL_HANDLE`
3. `vkGetDeviceQueue()`
   - https://vkdoc.net/man/vkGetDeviceQueue
     - `.device`
     - `.queueFamilyindex` *Chapter2.7*
       - `amVK_Device::amVK_1D_QCIs::select_QFAM_Graphics()`
     - `.queueIndex` *Chapter2.4*
       - `VkDeviceQueueCreateInfo.queueCount`
     - `.pQueue`

---

4. `VkPresentInfoKHR`
   - https://vkdoc.net/man/VkPresentInfoKHR
     - `.sType` `VK_STRUCTURE_TYPE_PRESENT_INFO_KHR`
     - `.pNext` `NULL`
       - *Maybe some interesting extensions, idk*
     - `.pWaitSemaphores` *Chapter9.6*
       - `amVK_SwapChain::RenderingFinished_SemaPhore`
     - `.pSwapchains`
     - `.pImageIndices`
     - `.pResults`

5. `vkQueuePresentKHR()`
   - https://vkdoc.net/man/vkQueuePresentKHR
     - `.queue`
     - `.pPresentInfo`

6. *Coloring Window*
7. `So far, The result` `GITHUB`
   - https://github.com/REYNEP/amGHOST/tree/805df077be97835083dd7716c3597b2f0e9347cb/amVK

4. The RenderLoop

0. `amVK` wrap 🩹

```
while(true) {
    W->dispatch_events_with_OSModalLoops();         // No way to disable ModalLoop so far in win32

    RP_FBs->RPBI_AcquireNextFrameBuffer();

    CB->BeginCommandBuffer(amVK_Sync::CommandBufferUsageFlags::Submit_Once);
    // ------------------------ CommandBufferRecording ---------------------------

        RP_FBs->CMDBeginRenderPass(CB->vk_CommandBuffer);
            RP_FBs->CMDSetViewport_n_Scissor(CB->vk_CommandBuffer);
            PLG->CMDBindPipeline(CB->vk_CommandBuffer);
                VB.CMDDraw(CB->vk_CommandBuffer);
        RP_FBs->CMDEndRenderPass(CB->vk_CommandBuffer);

    // ------------------------ CommandBufferRecording ---------------------------
    CB->EndCommandBuffer();

    // This was Done Before in CH4 : SwapChain
        amVK_SurfacePresenter  *PR = new amVK_SurfacePresenter();
                                PR->bind_Surface(S);
                                PR->bind_Device(D);
    // This was Done Before in CH4 : SwapChain

    PR->set_CommandBuffer(CB->vk_CommandBuffer);
    PR->submit_CMDBUF(D->Queues.GraphicsQ(0));
    PR->Present(D->Queues.GraphicsQ(0));

    vkQueueWaitIdle(D->Queues.GraphicsQ(0));
    REY_NoobTimer::wait(10); // wait 10ms
}
```

- 📜 *REY_DOCs*
  - *We must respond to OS InputEvents (Mouse/Keyboard)* ▨ Chapter14
    - *Otherwise, the OS might flag the app as Not Responsive*
    - *gotta make sure to call the Operating System APIs* `OS::ProcessEvents()`
    - *& that is exactly why* `amGHOST` *exists* 👩‍🔧
  - *Then do all the rendering shit that I talked about, in this* ▨ Chapter13
  - win32
    - *if you don't ask* win32 *for the Events of the WINDOW, then windows is gonna go insane, it's gonna flag your app as* "Not Reposnding"
  - ModalLoop ▨ Chapter14

---

5. Synchronization & SemaPhores

`VkSemaphore` ▩ ChapterZZZ

- ***https://vkdoc.net/man/VkSemaphore***
  - *I wouldn't suggest reading it right now tho* 😅
  - *But, basically,*
    - `SemaPhore` *will be used to synchronize the rendering and presentation of images*

1. `VkSemaphoreCreateInfo`

- *https://vkdoc.net/man/VkSemaphoreCreateInfo*
  - `.sType` ▨ `VK_STRUCTURE_TYPE_SEMAPHORE_CREATE_INFO`
  - `.pNext` ⌀ `NULL`
  - `.flags` ☐ `0`

2. `vkCreateSemaphore`

- https://vkdoc.net/man/vkCreateSemaphore
  - `.device`
  - `.pCreateInfo` 
  - `.pAllocator`  ChapterZZZ
  - `.pSemaphore` 

 eXtras / neXt chapters

- *6. Handling OS  InputEvents  Chapter14*
- *7. Resizing & SwapChain ReCreation  Chapter15*
- *8. MultiThreading  Chapter16*