Chapter 11: `Vertex` 🍭 & `VertexBuffer` 🗄

1. `Mesh/Vertices`

1. `amVK_Vertex`

```cpp
struct amVK_Vertex {
    float position[3];
    float color[4];
};
```

2. `Vertex Buffer`

---

3. `VkBufferCreateInfo`
   - https://vkdoc.net/man/VkBufferCreateInfo
     - `.sType` ▨ `VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO`
     - `.pNext` ▨ `nullptr`
     - `.flags` ▨ `VkBufferCreateFlagBits`
       - https://vkdoc.net/man/VkBufferCreateFlagBits | ivirtex-github
         - `SPARSE` ▨ *ChapterZZZ*
     - `.size` ▨ `sizeof(amVK_Vertex) * N`
     - `.usage` ▨ `VK_BUFFER_USAGE_VERTEX_BUFFER_BIT`
     - `.sharingMode` ▨ *ChapterZZZ*
       - `.queueFamilyIndexCount`
       - `.pQueueFamilyIndex`

4. `vkCreateBuffer()`
   - https://vkdoc.net/man/vkCreateBuffer
     - `.device` ▨
     - `.pCreateInfo` ▨
     - `.pAllocator`
     - `.pBuffer` ↩

5. `So far, The result` :- CH11.1.VertexBuffer.hh

2. `A lesson in Memory`

https://www.youtube.com/watch?v=uXgKXfVMeFw

(obviously i am not talking about `Vulkan` / `Implementation Programming`)

(i am talking about `Algorithms/CP/CodeForces/MIT6.046`)

1. `vkGetBufferMemoryRequirements()`
   - *https://vkdoc.net/man/vkGetBufferMemoryRequirements*
     - `.device` ▨ 🧑‍💻
     - `.buffer` ▨ 🧑‍💻
     - `.pMemoryRequirements` ↩ 📦

2. `VkMemoryRequirements`
   - *https://vkdoc.net/man/VkMemoryRequirements*
     - `.size` ➡ `VkMemoryAllocateInfo.allocationSize`
     - `.alignment`
     - `.memoryTypeBits`

---

3. `.memoryTypeIndex` | `VkPhysicalDeviceMemoryProperties`
   - *https://vkdoc.net/man/VkPhysicalDeviceMemoryProperties*
     - 🏷 `VkMemoryType memoryTypes[VK_MAX_MEMORY_TYPES];`
     - 🏷 `VkMemoryHeap memoryHeaps[VK_MAX_MEMORY_HEAPS];`

   - `VkMemoryType`
     - *https://vkdoc.net/man/VkMemoryType*
       - `.propertyFlags` 🏷 `VkMemoryPropertyFlags`
         - *https://vkdoc.net/man/VkMemoryPropertyFlags*
         - `VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT`
         - `VK_MEMORY_PROPERTY_HOST_VISIBLE_BIT`
         - `VK_MEMORY_PROPERTY_HOST_COHERENT_BIT`
         - `VK_MEMORY_PROPERTY_HOST_CACHED_BIT`
         - `VK_MEMORY_PROPERTY_LAZILY_ALLOCATED_BIT`
       - `.heapIndex` 🏷 `uint32_t`
   - `VkmemoryHeap`
     - *https://vkdoc.net/man/VkMemoryHeap*
       - `.size` 🏷 `VkDeviceSize`
       - `.flags` 🏷 `VkMemoryHeapFlags`
         - *https://vkdoc.net/man/VkMemoryHeapFlagBits* | *ivirtex-github*
         - `VK_MEMORY_HEAP_DEVICE_LOCAL_BIT`
         - `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT`
         - `VK_MEMORY_HEAP_TILE_MEMORY_BIT_QCOM`
         - `VK_MEMORY_HEAP_MULTI_INSTANCE_BIT_KHR`

   - `vkGetPhysicalDeviceMemoryProperties()`
     - *https://vkdoc.net/man/vkGetPhysicalDeviceMemoryProperties*
       - `.physicalDevice` ▨ 🧑‍💻
       - `.pFeatures` ↩ 📦

4. `VkPhysicalDeviceFeatures`
   - *https://vkdoc.net/man/VkPhysicalDeviceFeatures*
     - *Lots of* `VkBool32`

        - `Shaders`
        - `Texures`
        - `Sparse`
- `vkGetPhysicalDeviceFeatures()`
    - *https://vkdoc.net/man/vkGetPhysicalDeviceFeatures*
        - `.physicalDevice` ▨
        - `.pMemoryProperties`

5. `So far, The result`

```
class amVK_InstanceProps {
    static       void GetPhysicalDeviceFeatures(void);               // amVK_1D_GPUs_Features
    static       void GetPhysicalDeviceMemoryProperties(void);       // amVK_1D_GPUs_MEMProps

    static inline REY_Array<VkPhysicalDeviceFeatures>                   amVK_1D_GPUs_Features;
    static inline REY_Array<VkPhysicalDeviceMemoryProperties>           amVK_1D_GPUs_MEMProps;
}
    // The other one is copy of this one
void amVK_InstanceProps::GetPhysicalDeviceFeatures(void) {
    amVK_1D_GPUs_Features.reserve(amVK_1D_GPUs.n);
    amVK_LOOP_GPUs(k) {
        vkGetPhysicalDeviceFeatures(amVK_1D_GPUs[k], &amVK_1D_GPUs_Features[k]);
    }
    called_GetPhysicalDeviceFeatures = true;
}
```

---

6. `Visualization / [See it] / JSON Printing` :- 🔗 `GITHUB` amVK_InstancePropsExport_nlohmann.cpp#L1-L117
7. `REY_CategorizeMemoryHeaps()` 🔗 `GITHUB` amVK_GPUProps.cpp#L56-264
    - *Just Copy-Paste this one yk....*
    - *I Believe, the tags that I Created for this one, Vulkan should have given us those by default*
8. `Refactoring` is pretty smooth now, I did it again, in this commit 🔗 `GITHUB`
    - *https://github.com/REYNEP/amGHOST/tree/82311d2bd8586d07836be900448d8b7b9961c0ef*

---

9. `VkMemoryAllocateInfo`
    - *https://vkdoc.net/man/VkMemoryAllocateInfo*
        - *This documentation page is pretty big*
        - `.sType` ▨ `VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO`
        - `.pNext` `nullptr`
            - *interesting extensions*
        - `.allocationSize` ▨ `VkMemoryRequirements.size`
        - `.memoryTypeIndex` `uint32_t`
10. `vkAllocateMemory()`
    - *https://vkdoc.net/man/vkAllocateMemory*
        - `.device`

    - `.pAllocateInfo`
    - `.pAllocator`
    - `.pMemory`

11. `</> TheCode`

```
void amVK_VertexBuffer::AllocateMemory(void) {
    if(called_GetBufferMemoryRequirements == false) {
        this->GetBufferMemoryRequirements();
    }
    if (this->D->GPU_Props->called_REY_CategorizeMemoryHeaps == false) {
        this->D->GPU_Props->       REY_CategorizeMemoryHeaps();
    }
        AI.allocationSize  = vk_MemoryReq.size;
        AI.memoryTypeIndex = this->D->GPU_Props->MEMTypeID.CPU_GPU_Synced;

    VkResult return_code = vkAllocateMemory(this->D->vk_Device, &AI, nullptr, &this->vk_DeviceMemory);
    amVK_return_code_log( "vkAllocateMemory()" );
}
```

12. `vkMapMemory()`
    - https://vkdoc.net/man/vkMapMemory
13. `vkUnmapMemory()`
    - https://vkdoc.net/man/vkUnmapMemory
14. `vkBindBufferMemory()`
    - https://vkdoc.net/man/vkUnmapMemory

15. `</> TheCode`

```
void      amVK_VertexBuffer::MapMemory(void) {
    VkResult return_code = vkMapMemory(D->vk_Device, vk_DeviceMemory, 0, vk_MemoryReq.size, 0,
&vk_MappedMemoryData);
    amVK_return_code_log( "vkMapMemory()" );
}
void   amVK_VertexBuffer::CopyIntoMemory(void) {
    REY_memcpy(vk_MappedMemoryData, Vertices.data, CI.size);
}
void      amVK_VertexBuffer::UnMapMemory(void) {
    vkUnmapMemory(D->vk_Device, vk_DeviceMemory);
}
void      amVK_VertexBuffer::BindBufferMemory(void) {
    VkResult return_code = vkBindBufferMemory(D->vk_Device, vk_Buffer, vk_DeviceMemory, 0);
    amVK_return_code_log( "vkBindBufferMemory()" );
}
```

3. Enabling Validation Layers

```
class amVK_InstanceProps {
  public:
    static inline          REY_Array<VkLayerProperties>              amVK_1D_InstanceLayers;
    #define amVK_LOOP_ILayers(_var_) for (uint32_t _var_ = 0,  lim = amVK_1D_InstanceLayers.n;   _var_ < lim;  _var_++)

    static inline bool called_EnumerateInstanceLayerProperties = false;
    static void               EnumerateInstanceLayerProperties(void);                 // amVK_1D_InstanceLayers

    static bool                      isInstanceLayerAvailable(const char *layerName); // amVK_1D_InstanceLayers
}

class amVK_Instance {
    static inline REY_ArrayDYN<char*> amVK_1D_Instance_Layers_Enabled;
    static void                      addTo_1D_Instance_Layers_Enabled(const char* layerName);
    static void                        log_1D_Instance_Layers_Enabled(VkResult ret);  // CreateDevice() calls this
}

amVK_Instance::addTo_1D_Instance_Layers_Enabled("VK_LAYER_KHRONOS_validation");
```

GITHUB_WIP