Device Memory

1. Memory Data

Memory in our device can be classified as follow:

* **Video memory:** memory in graphics card.
* **System memory:** main memory.

And we have several generally usage:

* **Static data:** we only initialize it once.
* **CPU to GPU data:** we will modify it and need to update it often.
* **GPU to CPU data:** we need a coherent memory space. And the space is cached at system memory.

1. Device Memory

As mentioned at chapter 1, there are two memory spaces in our device. One is **system memory**, another is **video memory**. GPU can access video memory **directly** and access system memory **via driver.** So we can call memory allocated from video memory as **device local memory** and call allocated memory from system memory as **host memory**. But we need to notice one thing – GPU access host memory slower than device memory. Therefore, we should use host memory when we transfer data to device memory or small the data that maybe modify frequently.

There are memory types and heaps in our device. And each heap will specify its usage about memory. Memory type is used to tell us the usage about memory.

* 1. Memory Property

Memory properties are used to specify memory abilities. There are several memory types as follow:

|  |  |
| --- | --- |
| Memory property | Description |
| VK\_MEMORY\_PROPERTY\_DEVICE\_LOCAL\_BIT (0x00000001) | The memory can be accessed efficiently. It’s usually in video memory heap. |
| VK\_MEMORY\_PROPERTY\_HOST\_VISIBLE\_BIT (0x00000002) | The memory can be mapped. It means the system memories can be seen by GPU.  Across PCIe® bus, reads cached on GPU. So we need call flush after memcpy. |
| VK\_MEMORY\_PROPERTY\_HOST\_COHERENT\_BIT (0x00000004) | GPU access through PCIe®. So we don’t need to call flush after memcpy. CPU maybe can’t access. |
| VK\_MEMORY\_PROPERTY\_HOST\_CACHED\_BIT (0x00000008) | GPU access through PCIe®. Cached property includes coherent. CPU can access. |
| VK\_MEMORY\_PROPERTY\_LAZILY\_ALLOCATED\_BIT (0x00000010) | ???? |
| VK\_MEMORY\_PROPERTY\_PROTECTED\_BIT (0x00000020) | Mutually exclusive with host memory. But ???? |

One memory can follow one group of properties. Usual groups are listed as follow:

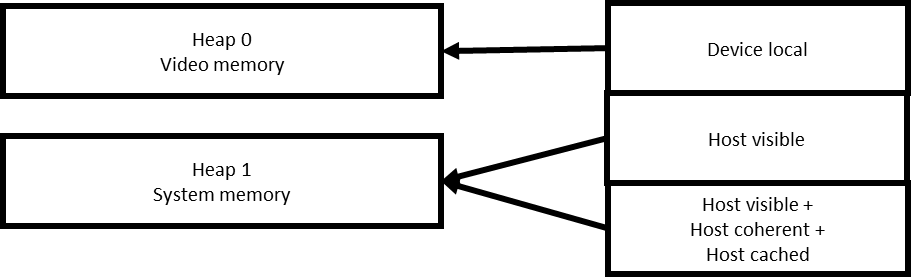
* DEVICE\_LOCAL\_BIT
* HOST\_VISIBLE\_BIT
* HOST\_VISIBLE\_BIT | HOST\_COHERENT\_BIT
* HOST\_VISIBLE\_BIT | HOST\_COHERENT\_BIT | HOST\_CACHED\_BIT
* DEVICE\_LOCAL\_BIT | HOST\_VISIBLE\_BIT(AMD only. 256MB on windows.)
  1. Memory Heap

Heap is the pool that is used to store memory. And the number of heaps is different between different devices. Each heap keeps particular type memory. For example, there are two heaps in my PC that it use 16GB RAM and graphics card Nvidia 1060 6GB. All memory types related with host memory is allocated from heap which size is approximately with DRAM. And all memory types related with device local memory is allocated from heap which size is approximately with graphics cards.

There are three heap flag in Vulkan:

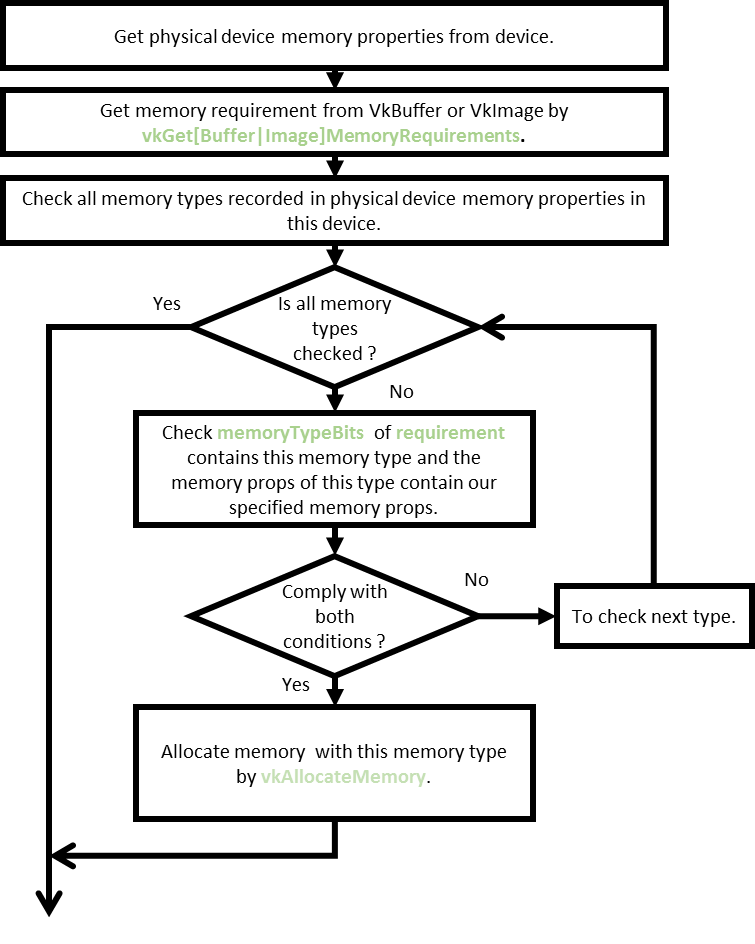
* **0:** DRAM heap.
* **VK\_MEMORY\_HEAP\_DEVICE\_LOCAL\_BIT:** Video memory heap.
* **VK\_MEMORY\_HEAP\_MULTI\_INSTANCE\_BIT:** Video memory heap. Allocate this memory at all graphics devices (SLI ????).

1. Physical Device Memory Properties



We can get memory information from physical device. API **vkGetPhysicalDeviceMemoryProperties** is used to get all supported **memory types** and **heaps** in this graphics device. Structure **VkPhysicalDeviceMemoryProperties** is used to get device memory type. In the structure, all supported **memory types** and **heap** will be recorded. There are two important details about each memory type. One is **memory properties**. The member **propertyFlags** is used to tell us the abilities in memory of this type. Another is **heap index**. The member **heapIndex** represents all memory with this type will be allocated and kept at target heap. The details about heap are **heap flags** and **its size**. The introduction is description at section 2.2.

1. Memory Allocation

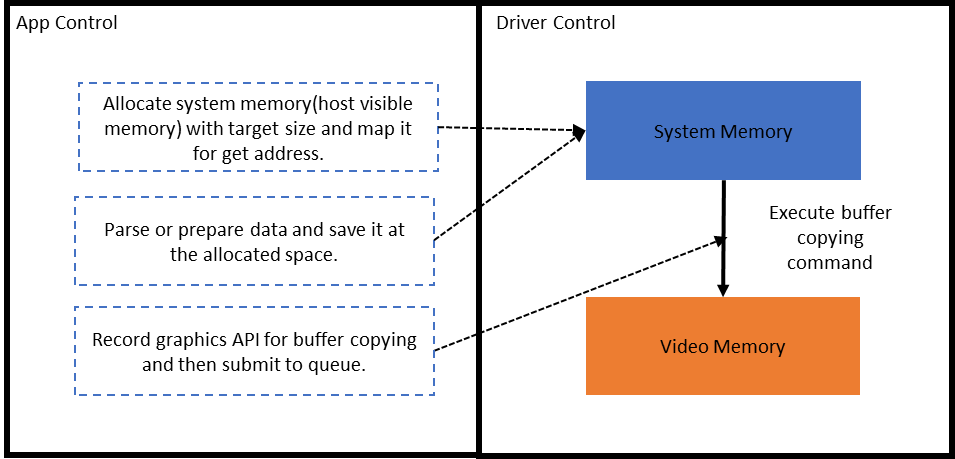


The figure lists all steps for allocating device memory. At first, we need to get physical device memory properties of this device. And then we need to get requirement of **VkImage** or **VkBuffer** by **vkGetBufferMemoryRequirements** and **vkGetImageMemoryRequirements**. The structure of requirement is **VkMemoryRequirements**. The member variable **memoryTypeBits** is used to tell us **that we can allocate this memory for image or buffer with which types**. If the type i is complied with our requirement, **the [i]th bit in memoryTypeBits will be set 1**. Another member variable **size** tell us the memory size should be allocated. The final member alignment tells us **the size of one memory page in this requirement**. That is, the allocated size is great than or equal to data size and the **allocated size can be divisible by alignment**.

After getting memory requirement, we will need to find out the suitable memory type which can be allocated for this requirement and its memory properties contain our requirements. And then, we will write **VkMemoryAllocateInfo** and write the suitable memory type and requirement size into the info structure. Finally, we will call **vkAllocateMemory** with the allocation info.

1. Data transfer

If we want to save data at video memory, we need to allocate system memory that can be access by GPU. So, we need to allocate a system memory for a buffer firstly. We call the host memory buffer as **staging buffer**. And then we need to **memcpy** data to the allocated system memory. Finally, we need to execute buffer copying by graphics API.



The figure reveals that all allocated memory controlled by driver. We only parse data and save the one at system memory firstly. Please note if we want to write data to host memory, we should map it before writing. After finishing data writing, we need to un-map the staging buffer and then recording command(vkCmdCopyBuffer) about buffer copying. Finally, we send the recorded command to queue for executing.

1. Compare with OpenGL

In OpenGL 3.X, buffer is classified to three type. They are **static**, **dynamic**, and **stream**. The static buffer is equal to using buffer binded memory which property is DEVICE\_LOCAL. The dynamic buffer is equal to using buffer bound memory which property is HOST\_VISIBLE. The stream buffer is equal to using buffer bound which properties are HOST\_VISIBLE, HOST\_COHERENT and HOST\_CACHED.

In OpenGL 4.X, we will create storage. It’s equal to device memory in Vulkan. And we will specify properties of storage too.

The table is used to show the equivalence about buffer among old OpenGL, OpenGL4 and Vulkan.4.

|  |  |  |
| --- | --- | --- |
| OpenGL3/gles3.0 | OpenGL4.0(Need to verify) | Vulkan |
| GL\_STATIC\_DRAW | Only map for read/write. | DEVICE\_LOCAL |
| GL\_DTNAMIC\_DRAW | GL\_DYNAMIC\_STORAGE\_BIT | HOST\_VISIBLE |
| GL\_STREAM\_DRAW | GL\_DYNAMIC\_STORAGE\_BIT | GL\_MAP\_PERSISTENT\_BIT |  GL\_MAP\_COHERENT\_BIT | HOST\_VISIBLE | HOST\_COHERENT | HOST\_CACHED |

1. Reference

[1] Khronos PPT <https://www.khronos.org/assets/uploads/developers/library/2018-vulkan-devday/03-Memory.pdf>

[2] Nvidia Vulkan

<https://developer.nvidia.com/vulkan-memory-management>

[3] AMD Vulkan

<https://gpuopen.com/vulkan-device-memory/>

[4] OpenGL Insight

<https://www.seas.upenn.edu/~pcozzi/OpenGLInsights/OpenGLInsights-AsynchronousBufferTransfers.pdf>