TP03
Vulkan
Logical device, Queues, Buffers and Memory allocation

Robin Faury

01/15/20

Abstract

In this practical work, we will see how to create logical device and its queue and create buffers to the GPU.

1 Logical device and Queues

In the last practical you chose the best physical device for GPGPU. We now need a logical device to interface with it. For your information it's possible to have multiple logical devices from the same physical device if you need several specific features. The handle of a logical device is VkDevice. As usual you need to create the VkDeviceCreateInfo with the sType VK_STRUCTURE_TYPE_DEVICE_CREATE_INFO. The first member to fill (pQueueCreateInfos) is a pointer on a VkDeviceQueueCreateInfo struct to specify how queues will be created. Create this structure like this:

```
VkDeviceQueueCreateInfo deviceQueueCreateInfo = {};
deviceQueueCreateInfo.sType = VK_STRUCTURE_TYPE_DEVICE_QUEUE_CREATE_INFO;
deviceQueueCreateInfo.queueFamilyIndex = /*The index of the queue family*/;
deviceQueueCreateInfo.queueCount = 1;
```

It is possible to have several queues for a queue family, but we should avoid it. Drivers allow you to create multiple queues if you need to call command to the GPU from different CPU threads. Each queue has a pointer on a priority (0.f to 1.f).

```
float priority = 1.0f;
deviceQueueCreateInfo.pQueuePriorities = &priority;
```

The second member of the VkDeviceQueueCreateInfo is the number of queueCreateInfo. We'll set this parameter to 1. All other parameter should be set to 0 or null for the moment.

```
VkPhysicalDeviceFeatures physicalDeviceFeatures = {};
VkDeviceCreateInfo deviceCreateInfo = {};
//...
deviceCreateInfo.pEnabledFeatures = physicalDeviceFeatures;
deviceCreateInfo.enabledExtensionCount = 0;
deviceCreateInfo.enabledLayerCount = 0;
```

We can now use vkCreateDevice to instantiate the logical device with the correct queue (set the VkAllocationCallback to nullptr). Don't forget to destroy it with vkDestroyDevice. Feel free to throw an error if the creation failed. The last step is to get the VkQueue handle from the VkDevice. For that we just need to ask it with:

```
vkGetDeviceQueue(logicalDevice, /*The index of the queue family*/, 0, &queue);
```

We create only one queue that why we ask to the logical device the first queue created of the queue family.

2 Buffer

A GPU buffer need more information than a CPU buffer. A pointer on the first element and the buffer size isn't enough. We need to add a VkDeviceMemory to operate on data in device memory and a VkMemoryPropertyFlags to store specified properties for a memory type.
Create a class Buffer with those members:

- VkBuffer buffer
- VkDeviceMemory bufferMemory
- pointer or reference on the VkDevice
- size_t (the size of the buffer)

2.1 Buffers

In this part we will create the handle VkBuffer. Create this function

```
VkBuffer createBuffer(const VkDevice& device, uint32_t bufferSize)
```

In this function we need to fill the VkBufferCreateInfo with its sType VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO. The other member must be filled like this:

- size : bufferSize
- usage : VK_BUFFER_USAGE_STORAGE_BUFFER_BIT
- sharingMode : VK_SHARING_MODE_EXCLUSIVE

The exclusive sharing mode means the buffer is visible by a single queue family at a time.
Call the

```
vkCreateBuffer(logicalDevice, &bufferCreateInfo, nullptr, &buffer)
```

and throw a runtime error if the buffer creation failed. And don't forget to destroy the buffer using vkDestroyBuffer.

2.2 Buffer memory

The buffer handle can be seen as a pointer to his properties, but its memory isn't allocated yet. Firstable, we need to find the memory type that match to the buffer requirements. For that we use this Vulkan object:

```
VkMemoryRequirements memoryRequirements;
vkGetBufferMemoryRequirements(logicalDevice, buffer, &memoryRequirements);
```

In this object you can find the size in bytes needed, the alignment and the memory type. Actually, there are several memory inside a graphic card and we must choose the best one for our application. To find the best match we use this object to list all memory properties:

```
VkPhysicalDeviceMemoryProperties memoryProperties;
vkGetPhysicalDeviceMemoryProperties(physicalDevice, &memoryProperties);
```

The memoryProperties object contain all the VkMemoryType. For each memoryType check if the propertyFlags contain VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT.

```
VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT & memoryProperties.memoryTypes[i].propertyFlags
== VK_MEMORY_PROPERTY_DEVICE_LOCAL_BIT
```

We must check if the memoryTypeBits match the current memoryType. Add in the condition this check:

```
memoryRequirements.memoryTypeBits & (1u << i)
```

We can select the first index who respects those two conditions. This index will be used to Allocate our buffer.

2.3 Buffer memory allocation

We have all information we need now to allocate the memory for our buffer. Create the VkMemoryAllocateInfo and fill the struct.

```
VkMemoryAllocateInfo memoryAllocateInfo = {};
memoryAllocateInfo.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
memoryAllocateInfo.allocationSize = memoryRequirements.size;
memoryAllocateInfo.memoryTypeIndex = /* index of the memory type*/;
```

As usual, allocate the memory, check the return type and throw an error if it's not VK_SUCCESS.

```
vkAllocateMemory(logicalDevice, &memoryAllocateInfo, nullptr, &bufferMemory)
```

We just need now to link the buffer to its memory:

```
vkBindBufferMemory(logicalDevice, buffer, bufferMemory, 0);
```

The last parameter is set to 0. This is the offset of the first member of the buffer inside the memory. That means you can have several buffer with offset using the same memory.
Don't forget to free the memory.

```
vkFreeMemory(logicalDevice, bufferMemory, nullptr);
```

3 going further

Since the beginning, we set the pAllocator to nullptr. Be aware than Vulkan allow you to allocate, most of the time, only 4096 objects. If you want more buffers you need to allocate big buffer, one by memory type, and play with offset to address more data. You can take a look to the Nvidia sample :

https://github.com/nvpro-samples/shared_sources/tree/master/nvvkpp#Allocators-in-nvvkpp