Resource & Memory

Resource:

- **Creating a buffer**
- **Allocating and binding a memory object for a buffer**
- **Setting a buffer memory barrier**
- **Creating a buffer view**
- **Creating an image**
- **Allocating and binding a memory object to an image**
- **Setting an image memory barrier**
- **Creating an image view**
- **Creating a 2D image and view**
- **Creating a layered 2D image with a CUBEMAP view**
- **Mapping, updating, and unmapping host-visible memory**
- **Copying data between buffers**
- **Copying data from a buffer to an image**
- **Copying data from an image to a buffer**
- **Using a staging buffer to update a buffer with a device-local**
- **memory bound**
- **Using a staging buffer to update an image with a device-local**
- **memory bound**
- **Destroying an image view**
- **Destroying an image**
- **Destroying a buffer view**
- **Freeing a memory object**
- **Destroying a buffer**

Vulkan 中的两种重要的资源类型: stored--buffers and images.

它们均没有自己的缓存空间,需要收到进行内存分配以及绑定.

```
typedef struct VkBufferCreateInfo {
    VkStructureType        sType;
    const void*            pNext;
    VkBufferCreateFlags    flags;
    VkDeviceSize           size;
    VkBufferUsageFlags     usage;
    VkSharingMode          sharingMode;
    uint32_t               queueFamilyIndexCount;
    const uint32_t*        pQueueFamilyIndices;
} VkBufferCreateInfo;
 //·VK_BUFFER_USAGE_TRANSFER_SRC_BIT表示缓冲区可以是复制操作的数据源
 // ·VK_BUFFER_USAGE_TRANSFER_DST_BIT表示可以将数据赋值到缓冲区
 // ·VK_BUFFER_USAGE_UNIFORM_TEXEL_BUFFER_BIT表示缓冲区可以在着色器中用作统一纹理缓冲区
 // ·VK_BUFFER_USAGE_STORAGE_TEXEL_BUFFER_BIT表示缓冲区可以在着色器中用作储存纹理缓冲区
 // ·VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT表示缓冲区可以在着色器中用作统一变量的值来源
 // ·VK_BUFFER_USAGE_STORAGE_BUFFER_BIT表示可以在着色器中将数储存在缓冲区中
 // ·VK_BUFFER_USAGE_INDEX_BUFFER_BIT表示缓冲区可以在绘制期间用作顶点索引数据源
 // ·VK_BUFFER_USAGE_VERTEX_BUFFER_BIT表示缓冲区可以是绘图期间指定的顶点属性的数据源
 // ·VK_BUFFER_USAGE_INDIRECT_BUFFER_BIT表示缓冲区可以包含间接绘制期间将使用的数据
```

在创建Buffer时候需要指明Buffer的创建类型

同样Image的创建与Buffer的创建基本一样,只不过Image图像可以指定表示具有一维、二维或三维的数据.

```
typedef struct VkImageCreateInfo {
    VkStructureType          sType;
    const void*              pNext;
    VkImageCreateFlags       flags;
    VkImageType              imageType;
    VkFormat                 format;
    VkExtent3D               extent;
    uint32_t                 mipLevels;
    uint32_t                 arrayLayers;
    VkSampleCountFlagBits    samples;
    VkImageTiling            tiling;
    VkImageUsageFlags        usage;
    VkSharingMode            sharingMode;
    uint32_t                 queueFamilyIndexCount;
    const uint32_t*          pQueueFamilyIndices;
    VkImageLayout            initialLayout;
} VkImageCreateInfo;
```

与缓冲区类似，图像不是使用绑定的内存存储创建的。我们需要隐式创建一个内存对象并将其绑定到图像。也可以使用现有内存来实现此目的。

```
bool Tutorial06::AllocateImageMemory( VkImage image, VkMemoryPropertyFlagBits property, VkDeviceMemory *memory ) {
    VkMemoryRequirements image_memory_requirements;
    vkGetImageMemoryRequirements( GetDevice(), image, &image_memory_requirements );

    VkPhysicalDeviceMemoryProperties memory_properties;
    vkGetPhysicalDeviceMemoryProperties( GetPhysicalDevice(), &memory_properties );
```

```
    for( uint32_t i = 0; i < memory_properties.memoryTypeCount; ++i ) {
      if( (image_memory_requirements.memoryTypeBits & (1 << i)) &&
        (memory_properties.memoryTypes[i].propertyFlags & property) ) {

        VkMemoryAllocateInfo memory_allocate_info = {
          VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO,     // VkStructureType                       sType
          nullptr,                                    // const void                           *pNext
          image_memory_requirements.size,             // VkDeviceSize                          allocationSize
          i                                           // uint32_t                              memoryTypeIndex
        };

        if( vkAllocateMemory( GetDevice(), &memory_allocate_info, nullptr, memory ) == VK_SUCCESS ) {
          return true;
        }
      }
    }
    return false;
```