GAMES 106

现代图形绘制流水线原理与实践

绘制流水线原理（第二课时）

主讲：高涛

课程安排

教学目标


Vulkan的基本架构，绘制流程
Vulkan绘制对象创建，内存管理，以及调试方法和工具
Vulkan的多线程同步，基于移动端，一些常见的优化和实践
作业和反馈

作业

扩展GLTF

作业

HW1 作业框架

载入GLTF → GLTF Render Pass

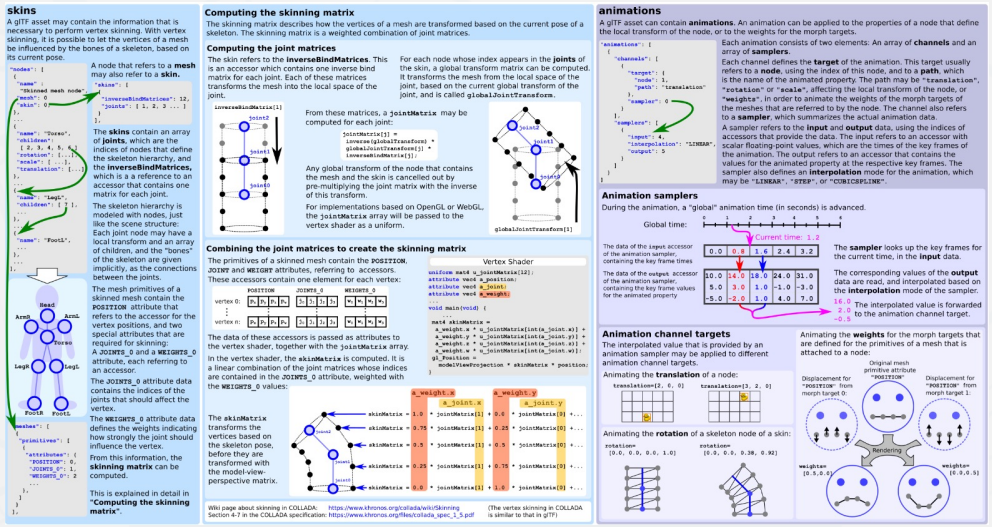
HW1 作业要求

载入GLTF
- 载入骨骼和动画数据
- 读取法线，自发光，PBR纹理贴图

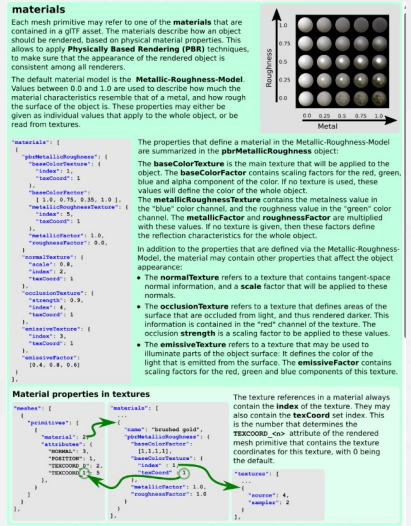
GLTF Render Pass
- Vertex Shader
  - 支持骨骼动画计算
- Fragment Shader
  - 实现PBR的直接光照和间接光照

Tonemap
- 实现Tonemap Render Pass

GLTF文件中关于骨骼动画的描述



GLTF文件中关于材质以及纹理的描述



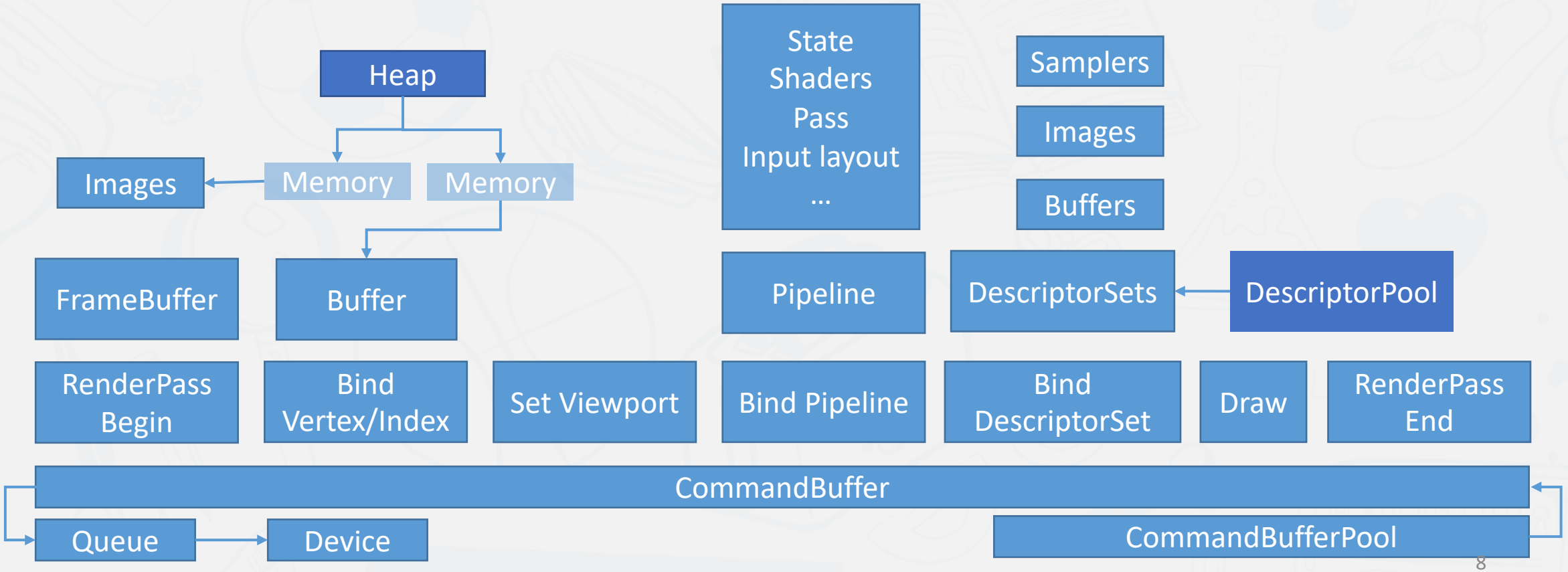
目录

CONTENTS

Vulkan创建对象

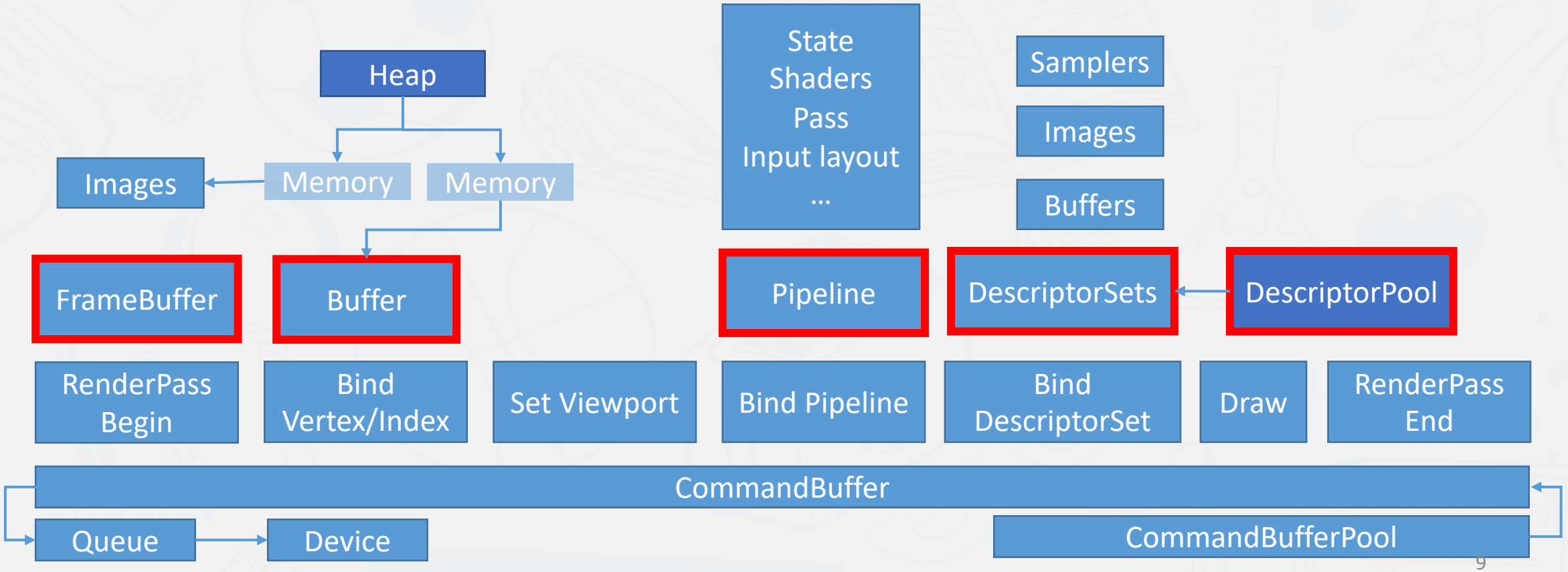
回顾Vulkan流程

Vulkan创建对象



重要的Vulkan对象

Vulkan创建对象



API风格

Vulkan创建对象

所有创建对象的方法都类似
vkCreateObject
一般有四个参数

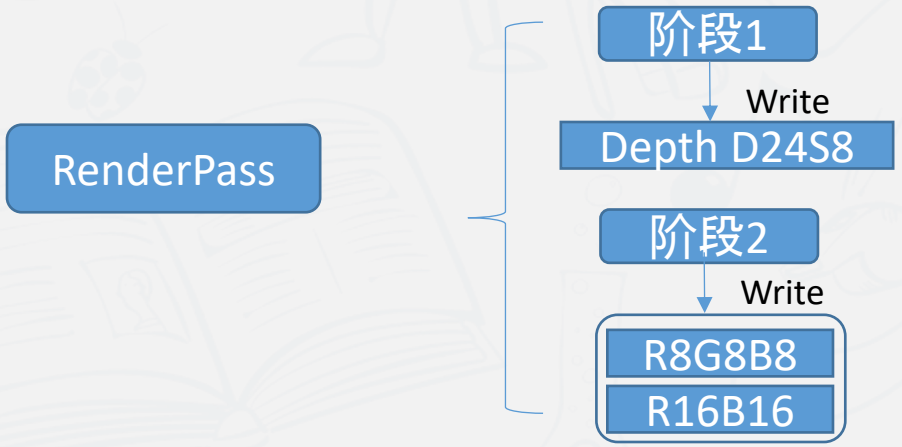
- LogicDevice
- VkObjectCreateInfo
- userAllocator
- Object对象地址

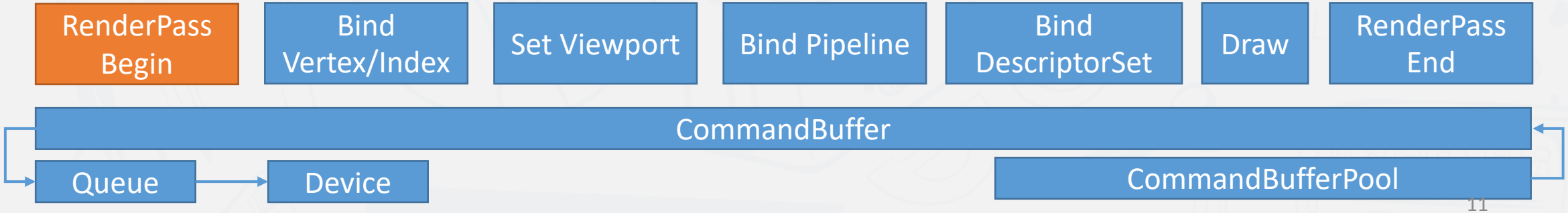
```
VkBufferCreateInfo vbci{};
vbci.sType = VK_STRUCTURE_TYPE_BUFFER_CREATE_INFO;
vbci.pNext = nullptr;
vbci.flags = 0;
vbci.size = Buffer_size_in_byte;
vbci.usage = VK_BUFFER_USAGE_UNIFORM_BUFFER_BIT;
vbci.queueFamilyIndexCount = 0;
vbci.pQueueFamilyIndices = nullptr;

VkBuffer buffer;
VkResult result = vkCreateBuffer(LogicalDevice, &vbci, Allocator, &buffer);

VkMemoryRequirements vmr;
vkGetBufferMemoryRequirements(LogicalDevice, buffer, &vmr);

VkMemoryAllocateInfo vmai;
vmai.sType = VK_STRUCTURE_TYPE_MEMORY_ALLOCATE_INFO;
vmai.pNext = nullptr;
vmai.allocationSize = vmr.size;
vmai.memoryTypeIndex = 0;

VkDeviceMemory device_memory;
result = vkAllocateMemory(LogicalDevice, &vmai, Allocator, &device_memory);
result = vkBindBufferMemory(LogicalDevice, buffer, device_memory, 0);
```

RenderPass

渲染通道
VkRenderPass

RenderPass 描述了一段绘制Pass中绑定的RenderTarget有哪些，有哪些绘制阶段(subpass), 以及对这些绑定的RenderTarges需要执行哪些操作。





RenderPass

渲染通道
VkRenderPass

VkSubpassDescription
- 指向Attachment
- VkAttachmentReference

VkAttachmentDescription
- 渲染通道输出数组
- 定义格式
- 输入输出的IO
- 渲染开始前后的布局（VkImageLayout），可以当作状态去理解

VkRenderPassCreateInfo

RenderPass Begin | Bind Vertex/Index | Set Viewport | Bind Pipeline | Bind DescriptorSet | Draw | RenderPass End

CommandBuffer

Queue → Device

CommandBufferPool

RenderPass

渲染通道
VkRenderPass

VkSubpassDescription
- 指向Attachment
- VkAttachmentReference

VkAttachmentDescription
- 渲染通道输出数组
- 定义格式
- 输入输出的IO
- 渲染开始前后的布局（VkImageLayout），可以当作状态去理解

VkRenderPassCreateInfo

VkSubpassDependency
- 有多个subpass的时候，定义相互依赖关系
- 单个subpass的时候通常可以设置位nullptr

RenderPass Begin | Bind Vertex/Index | Set Viewport | Bind Pipeline | Bind DescriptorSet | Draw | RenderPass End

CommandBuffer

Queue → Device

CommandBufferPool

FrameBuffer

帧缓冲区
VkFramebuffer

VkFramebufferCreateInfo

Width，Height，Layer
- 输出的维度，只有定义在这个区域内的像素才会被渲染

VkImageView
- 真正绘制输出的Image

RenderPass Begin | Bind Vertex/Index | Set Viewport | Bind Pipeline | Bind DescriptorSet | Draw | RenderPass End

CommandBuffer

Queue → Device

CommandBufferPool

Buffer

缓冲区
VkBuffer

VkBufferCreateInfo
- 定义buffer大小
- 使用的用途，Index，Vertex，Uniform可以多种用途混合

vkBindBufferMemory
绑定内存和buffer句柄

VkMemoryAllocateInfo
- 开辟GPU内存
- 可以指定到那一块区域的内存

缓冲区视图
VkBufferView

VkBufferViewCreateInfo
- 定义在着色器中buffer被使用的真正用途，uniform还是storage

RenderPass Begin | Bind Vertex/Index | Set Viewport | Bind Pipeline | Bind DescriptorSet | Draw | RenderPass End

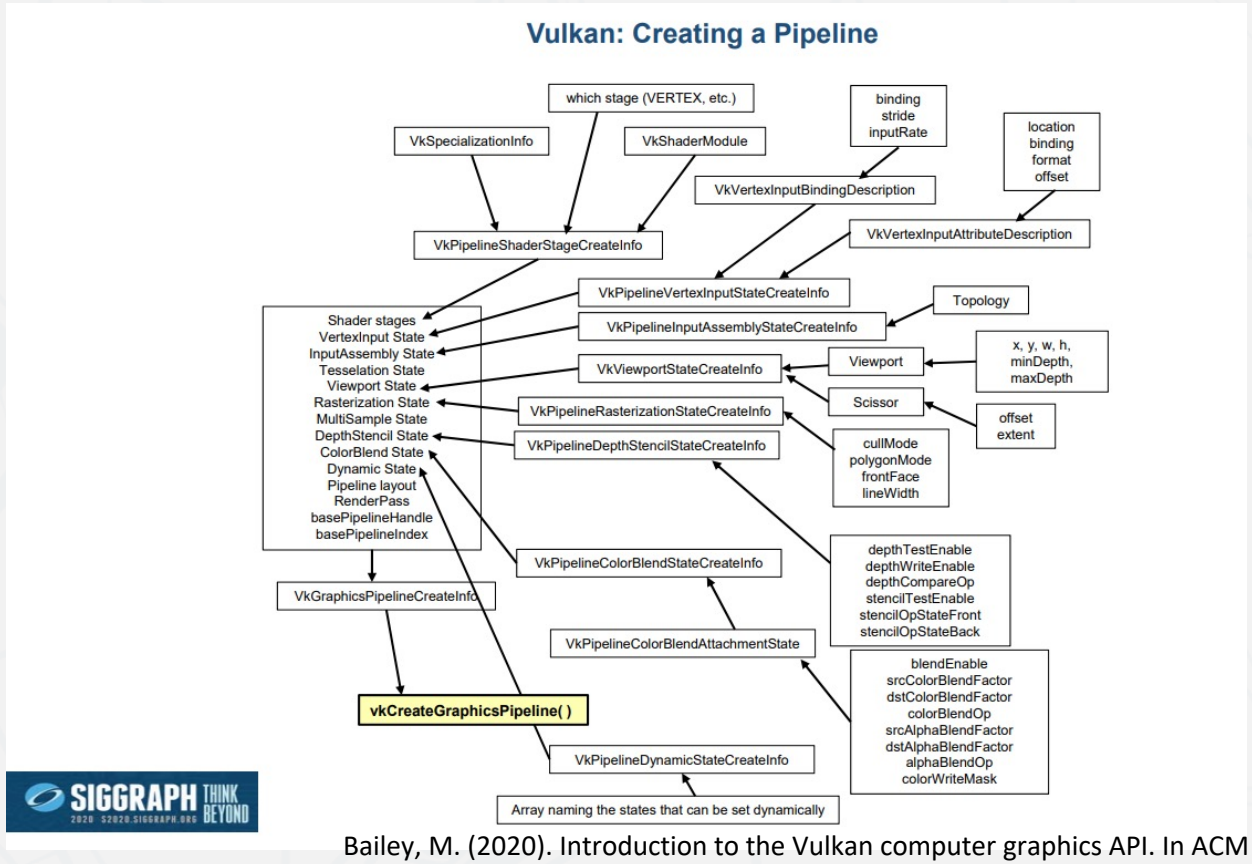
CommandBuffer

Queue → Device

CommandBufferPool

Pipeline

图形管线
VkPipeline



Bailey, M. (2020). Introduction to the Vulkan computer graphics API. In ACM SIGGRAPH 2020 Courses (pp. 1-128).

RenderPass Begin | Bind Vertex/Index | Set Viewport | Bind Pipeline | Bind DescriptorSet | Draw | RenderPass End

CommandBuffer

Queue → Device

CommandBufferPool

Image

图片
VkImage

VkImageCreateInfo
- 定义图片格式
- 图片大小，mipmap层级
- 图片的用途
  - 可以被采样
  - 可以用作颜色输出
  - 可以用作深度输出
- 定义图片布局，可以按照状态理解

vkBindImageMemory
绑定内存和image句柄

VkMemoryAllocateInfo
- 开辟GPU内存
- 可以指定到那一块区域的内存

图片视图
VkImageView

VkImageViewCreateInfo
- 可以被被定义和Image兼容的格式
- RGBA通道的映射关系
- 可以被采样的范围

RenderPass Begin | Bind Vertex/Index | Set Viewport | Bind Pipeline | Bind DescriptorSet | Draw | RenderPass End

CommandBuffer

Queue → Device

CommandBufferPool

PipelineLayout

管线布局
VkPipelineLayout

VkPipelineLayoutCreateInfo
- 定义多个描述符集合

VkDescriptorSetLayoutBinding
- 定义每一个资源对象的序号和类型

描述符集
DescriptorSetLayout

VkDescriptorSetLayoutCreateInfo
- 定义每一个槽位的uniform 输入，包括对象和类型

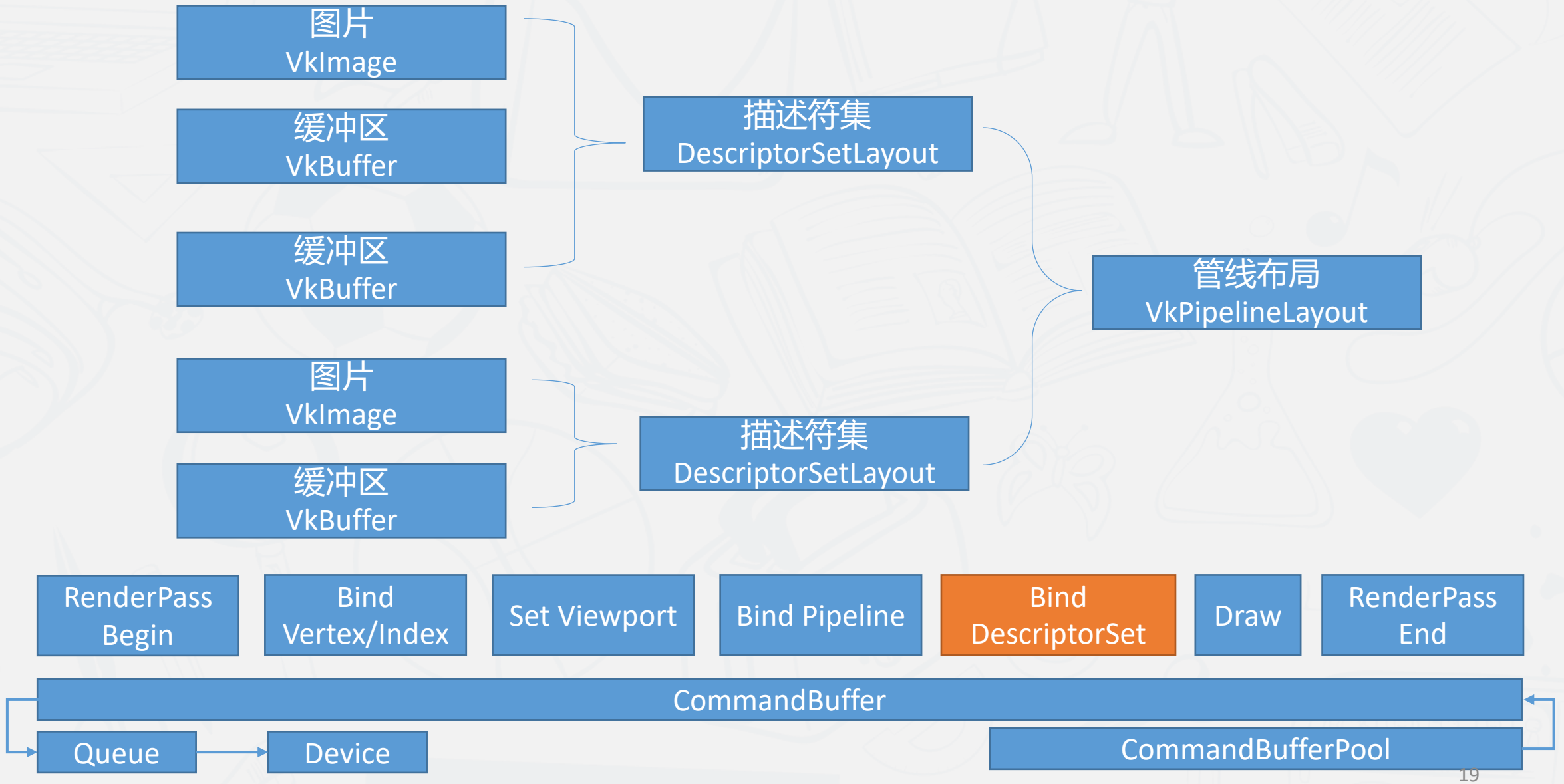
RenderPass Begin | Bind Vertex/Index | Set Viewport | Bind Pipeline | Bind DescriptorSet | Draw | RenderPass End

CommandBuffer

Queue → Device

CommandBufferPool

PipelineLayout



Vulkan内存管理

硬件

Vulkan内存管理

VB，IB，SRV，RT，DS，UAV

GPU内存
内存池
Device

Default Heap
Device Local

PCIE 传输数据

CPU内存
内存池
Host

Update Heap
Host Visible|
Host Coherent

Readback Heap
Host Visible |
Host Coherent |
Host Cached

Vulkan内存描述

- 根据不同的硬件架构，会有不同的内存池子
- 每一个内存池根据硬件会有不同的属性
  - DEVICE_LOCAL
    - 能被GPU快速访问。
  - HOST_VISIBLE
    - 能够被CPU访问。
  - HOST_COHERENT
    - 这块内存保证了CPU和GPU的内存一致，不需要做同步操作。
  - HOST_CACHED
    - CPU会缓存的内容，经常被CPU读取的内存。
  - LAZILY_ALLOCATED
    - Vulkan会在后续根据需要分配。


```
{
    "memoryTypeCount": 5,
    "memoryTypes" : [
        {"propertyFlags":"0", "heapIndex":"1" },
        {"propertyFlags":"DEVICE_LOCAL", "heapIndex":"0" },
        {"propertyFlags":"HOST_VISIBLE | HOST_COHERENT", "heapIndex":"1" },
        {"propertyFlags":"HOST_COHERENT | LAZILY_ALLOCATED", "heapIndex":"1" },
        {"propertyFlags":"DEVICE_LOCAL | HOST_VISIBLE | HOST_COHERENT", "heapIndex":"2" },
    ],
    "memoryHeapCount": 3,
    "memoryHeaps": [
        {"size":8421113856, "flags":"DEVICE_LOCAL" },
        {"size":8552943616, "flags":0 },
        {"size":224395264, "flags":"DEVICE_LOCAL" },
    ]
}
```


来自一张3070的桌面显卡的内存池

Vulkan内存API

Vulkan内存管理

申请GPU内存

- 需要确定GPU内存大小
- 需要根据使用情况确定从哪一个内存区域申请
- 申请的次数是有限的，Vulkan标准要求不小于4096次！！！

```
typedef struct VkMemoryAllocateInfo
{
    VkStructureType sType;
    const void* pNext;
    VkDeviceSize allocationSize;
    uint32_t memoryTypeIndex;
} VkMemoryAllocateInfo;
```

```
{
    "memoryTypeCount": 5,
    "memoryTypes" : [
        {"propertyFlags":"0", "heapIndex":"1" },
        {"propertyFlags":"DEVICE_LOCAL", "heapIndex":"0" },
        {"propertyFlags":"HOST_VISIBLE | HOST_COHERENT", "heapIndex":"1" },
        {"propertyFlags":"HOST_COHERENT | LAZILY_ALLOCATED", "heapIndex":"1" },
        {"propertyFlags":"DEVICE_LOCAL | HOST_VISIBLE | HOST_COHERENT", "heapIndex":"2" },
    ],
    "memoryHeapCount": 3,
    "memoryHeaps": [
        {"size":8421113856, "flags":"DEVICE_LOCAL" },
        {"size":8552943616, "flags":0 },
        {"size":224395264, "flags":"DEVICE_LOCAL" },
    ]
}
```

这一切都太复杂了！！！

Vulkan Memory Allocator

Vulkan内存管理

感谢AMD的vma库，方便用更容易理解的方式来管理内存，
并且帮助你做内存分配

- VMA_MEMORY_USAGE_GPU_ONLY
  - 被GPU读写，只会从CPU拷贝数据一次到GPU
- VMA_MEMORY_USAGE_CPU_ONLY
  - 用于频繁地从CPU拷贝数据到GPU
- VMA_MEMORY_USAGE_CPU_TO_GPU
  - 用于偶尔从CPU拷贝数据到GPU
- VMA_MEMORY_USAGE_GPU_TO_CPU
  - 用于偶尔从GPU拷贝数据到CPU
- VMA_MEMORY_USAGE_CPU_COPY
  - 用于频繁从GPU拷贝数据到CPU
- VMA_MEMORY_USAGE_GPU_LAZILY_ALLOCATED
  - 根据需要在后续分配

AMD Vulkan Memory Allocator

Vulkan内存管理

感谢AMD的vma库，方便用更容易理解的方式来管理内存，
并且帮助你做内存分配

- VMA_MEMORY_USAGE_GPU_ONLY
  - 被GPU读写，只会从CPU拷贝数据一次到GPU
- VMA_MEMORY_USAGE_CPU_ONLY
  - 用于频繁地从CPU拷贝数据到GPU
- VMA_MEMORY_USAGE_CPU_TO_GPU
  - 用于偶尔从CPU拷贝数据到GPU
- VMA_MEMORY_USAGE_GPU_TO_CPU
  - 用于偶尔从GPU拷贝数据到CPU
- VMA_MEMORY_USAGE_CPU_COPY
  - 用于频繁从GPU拷贝数据到CPU
- VMA_MEMORY_USAGE_GPU_LAZILY_ALLOCATED
  - 根据需要在后续分配

Render Target， Depth Stencil

AMD Vulkan Memory Allocator　　Vulkan内存管理

感谢AMD的vma库，方便用更容易理解的方式来管理内存，
并且帮助你做内存分配

- VMA_MEMORY_USAGE_GPU_ONLY
  - 被GPU读写，只会从CPU拷贝数据一次到GPU
- VMA_MEMORY_USAGE_CPU_ONLY
  - 用于频繁地从CPU拷贝数据到GPU
- VMA_MEMORY_USAGE_CPU_TO_GPU
  - 用于偶尔从CPU拷贝数据到GPU
- VMA_MEMORY_USAGE_GPU_TO_CPU
  - 用于偶尔从GPU拷贝数据到CPU
- VMA_MEMORY_USAGE_CPU_COPY
  - 用于频繁从GPU拷贝数据到CPU
- VMA_MEMORY_USAGE_GPU_LAZILY_ALLOCATED
  - 根据需要在后续分配

Render Target， Depth Stencil

Vertex Buffer， Uniform Buffer， Image

AMD Vulkan Memory Allocator　　Vulkan内存管理

感谢AMD的vma库，方便用更容易理解的方式来管理内存，
并且帮助你做内存分配

- VMA_MEMORY_USAGE_GPU_ONLY
  - 被GPU读写，只会从CPU拷贝数据一次到GPU
- VMA_MEMORY_USAGE_CPU_ONLY
  - 用于频繁地从CPU拷贝数据到GPU
- VMA_MEMORY_USAGE_CPU_TO_GPU
  - 用于偶尔从CPU拷贝数据到GPU
- VMA_MEMORY_USAGE_GPU_TO_CPU
  - 用于偶尔从GPU拷贝数据到CPU
- VMA_MEMORY_USAGE_CPU_COPY
  - 用于频繁从GPU拷贝数据到CPU
- VMA_MEMORY_USAGE_GPU_LAZILY_ALLOCATED
  - 根据需要在后续分配

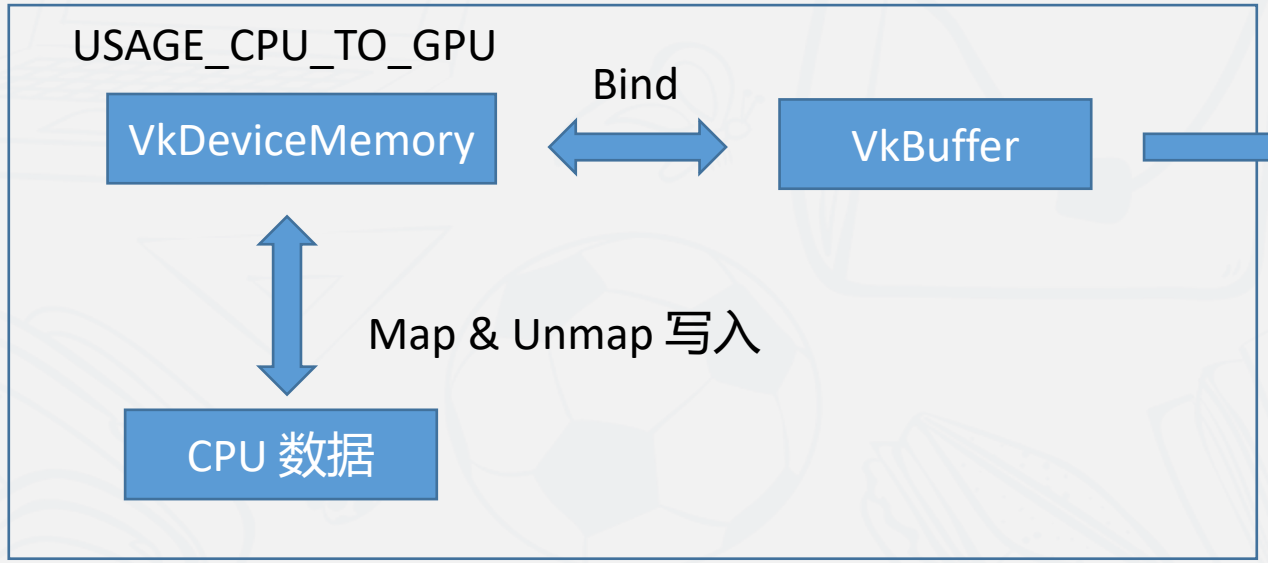
Render Target， Depth Stencil

Staging Buffer

Vertex Buffer， Uniform Buffer， Image

还可以更加高效地创建

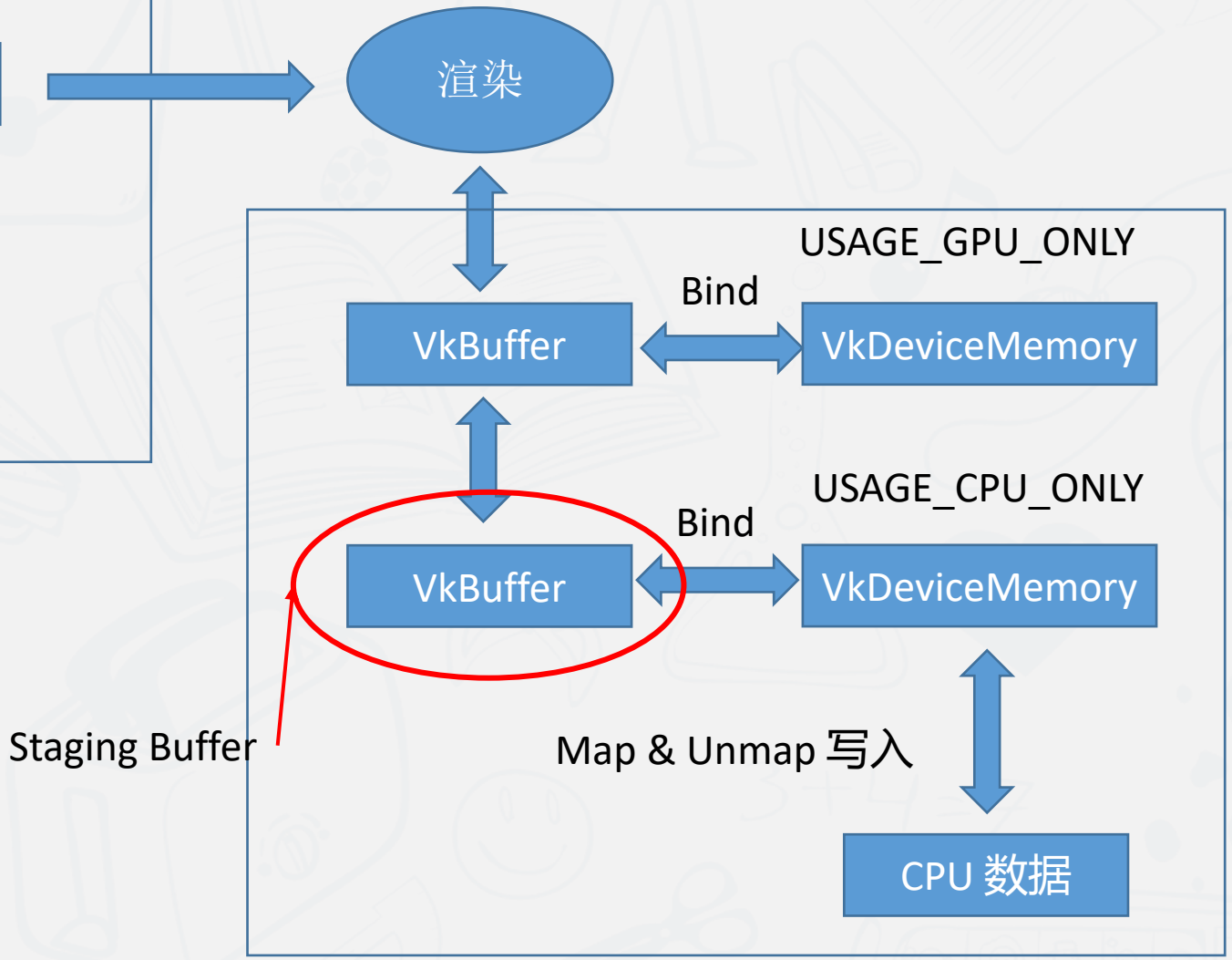
Staging Buffer



更新UniformBuffer的一般方法

优势

- 减少CPU和GPU之间的数据传输次数
- 提高性能，减少CPU和GPU等待时间
- 简化代码，减少异步导致的问题



通过staging buffer更新UniformBuffer

Vulkan绘制

Vulkan绘制

CommandBuffer

vkBeginCommandBuffer

vkCmdBeginRenderPass

vkCmdEndRenderPass

vkEndCommandBuffer

vkCmdSetViewport

vkCmdSetScissor

vkCmdBindDescriptorSets

vkCmdBindPipeline

vkCmdBindVertexBuffers

vkCmdBindIndexBuffer

vkCmdDrawIndexed

调试工具及方法

调试工具及方法

RenderDoc

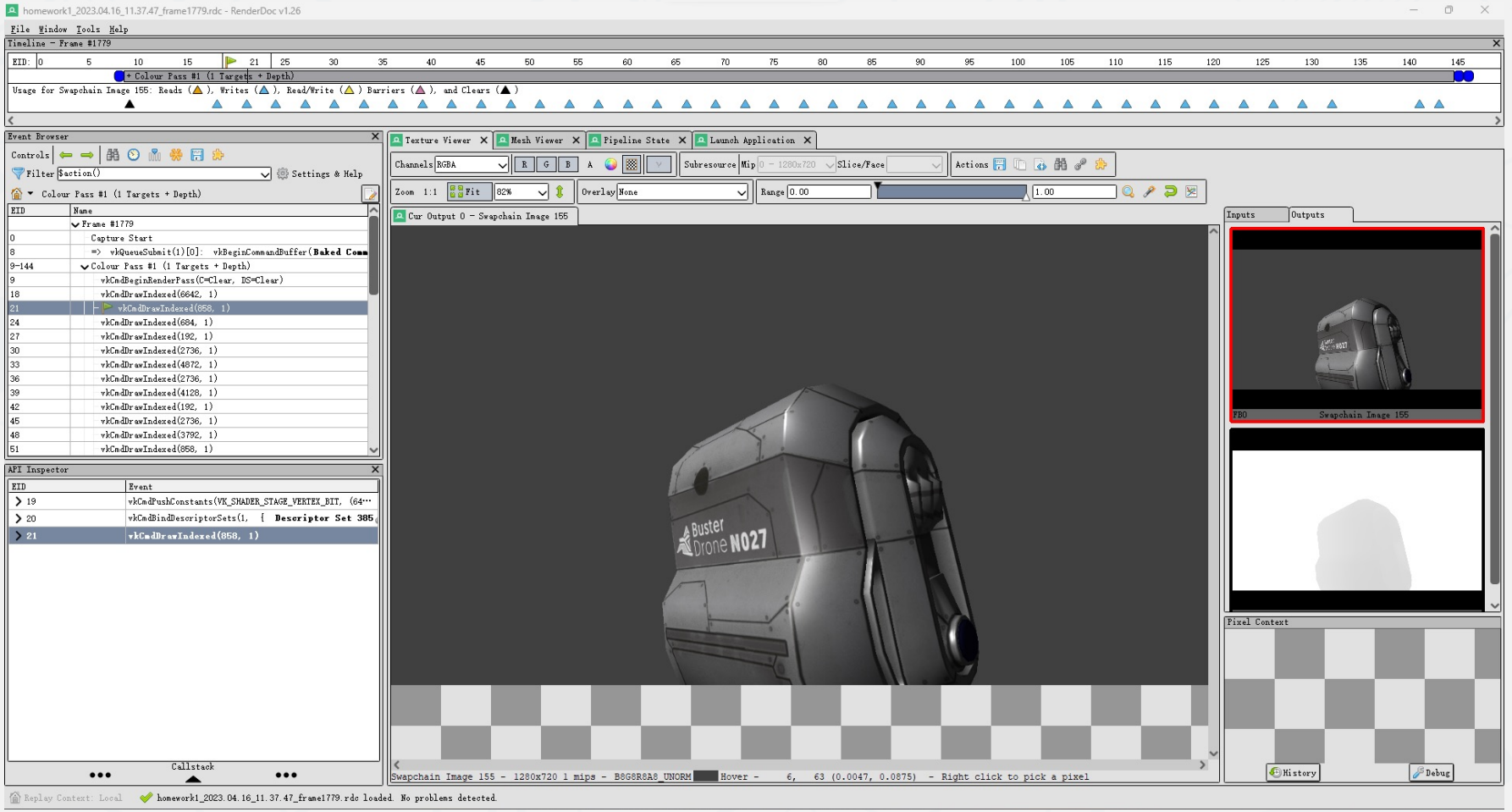
下载链接RenderDoc

优点

- 目前最方便的图形调试工具
- Shader 可以但不调试
- 可视化能力强

缺点

- 性能分析能力不足
- 会对Vulkan调用做hook，无法很好地调试由异步导致的问题。
- 会占用比较大的内存



调试工具及方法

RenderDoc

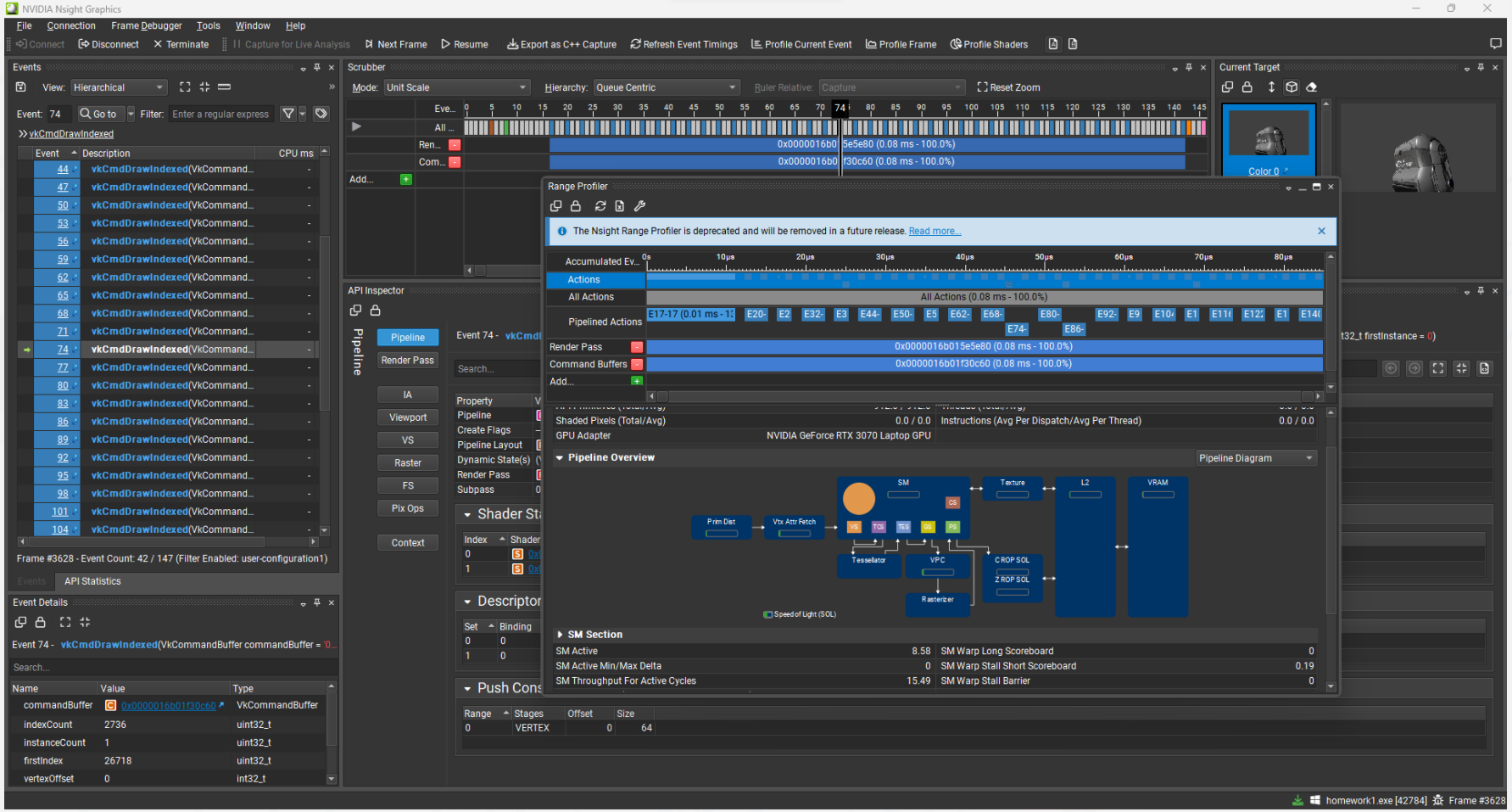
下载链接
[NVIDIA Nsight Graphics]

优点

- Profile能力非常强
- 可以看到所有的资源的视图
- 占用内存小

缺点

- 必须在Nvidia平台
- 无法调试初始化阶段出现的问题，也就是第一帧出现的问题



谢谢