GAMES 106

现代图形绘制流水线原理与实践

绘制流水线原理（第一课时）

主讲：高涛

课程目标

学情分析

谁适合这门课

- 刚开始接触RHI的初学者
- 有一定的RHI基础，比如学习过OpenGL和D3D11
- 已经初步学会使用C/C++

收获什么

- Vulkan的API的基本使用
- 渲染图形管线的理解
- 针对移动端的优化实践

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

目录

CONTENTS

Vulkan简介

开发环境

Windows/Linux

Vulkan SDK

Android

Android NDK

MacOS\iOS

MoltenVK (感谢valve)

下载链接

LunarXchange (lunarg.com)

真正的实现，依赖厂家的驱动实现。
PS:永远不要相信设备提供商声明的Vulkan特性支持。
可能个别特性，只存在Vulkan的标准文档中。这在移动端和一些核显笔记本上非常重要！！！

Vulkan简介

RHI对比

传统RHI

- RenderingContext：几乎所有的绘制命令都依赖隐式或者显式的Context调用
- 驱动帮你干了很多事情。（意味着你对GPU的掌控比较少）
- 驱动层一般会比较重

现代RHI

- D3D12/Vulkan/Metal
- 依赖程序自身的认知，需要开发者关心同步以及内存分配。
- 多线程友好。
- 开发者几乎拥有GPU计算的掌控权。

区别？

- 驱动几乎不帮你干活
- 要写出高性能的程序会比较困难。
- 熟练的程序员可以掌握雷电。
- 驱动层会非常的薄。

推荐传统->现代

- OpenGL：接口足够简单，能够快速上手
- D3D11：虽然还有状态机的概念，但是接口是基于OOP，对渲染状态有更好的概念
- Vulkan：进阶提升。更多的概念，更多的自由度，当然还有更难的实际应用。

程序的流水线

初始化 → 渲染主循环 → 退出程序

- 调用操作系统的API创建程序可以显示的窗口。可以由GLFW，SDL等一些第三方库来实现。
- 初始化Vulkan
  - 程序和Vulkan库取得联系。
  - Vulkan和显卡设备取得联系
- 创建Swapchian
  - Vulkan和显示窗口取得联系。

- 输入
  - 几何信息
  - Uniform
- 输出
  - 屏幕中显示的画面
- 逻辑
  - 着色器（shader）
- 更新CPU端的业务逻辑。
- 数据CPU拷贝到GPU。
- 生成Vulkan Command Buffer 把渲染任务给GPU并在屏幕中显示。

- 释放Vulkan对象，以及CPU的数据。
- 关闭程序窗口。

初始化Vulkan

初始化 → 渲染主循环 → 退出程序

初始化Vulkan

创建窗口

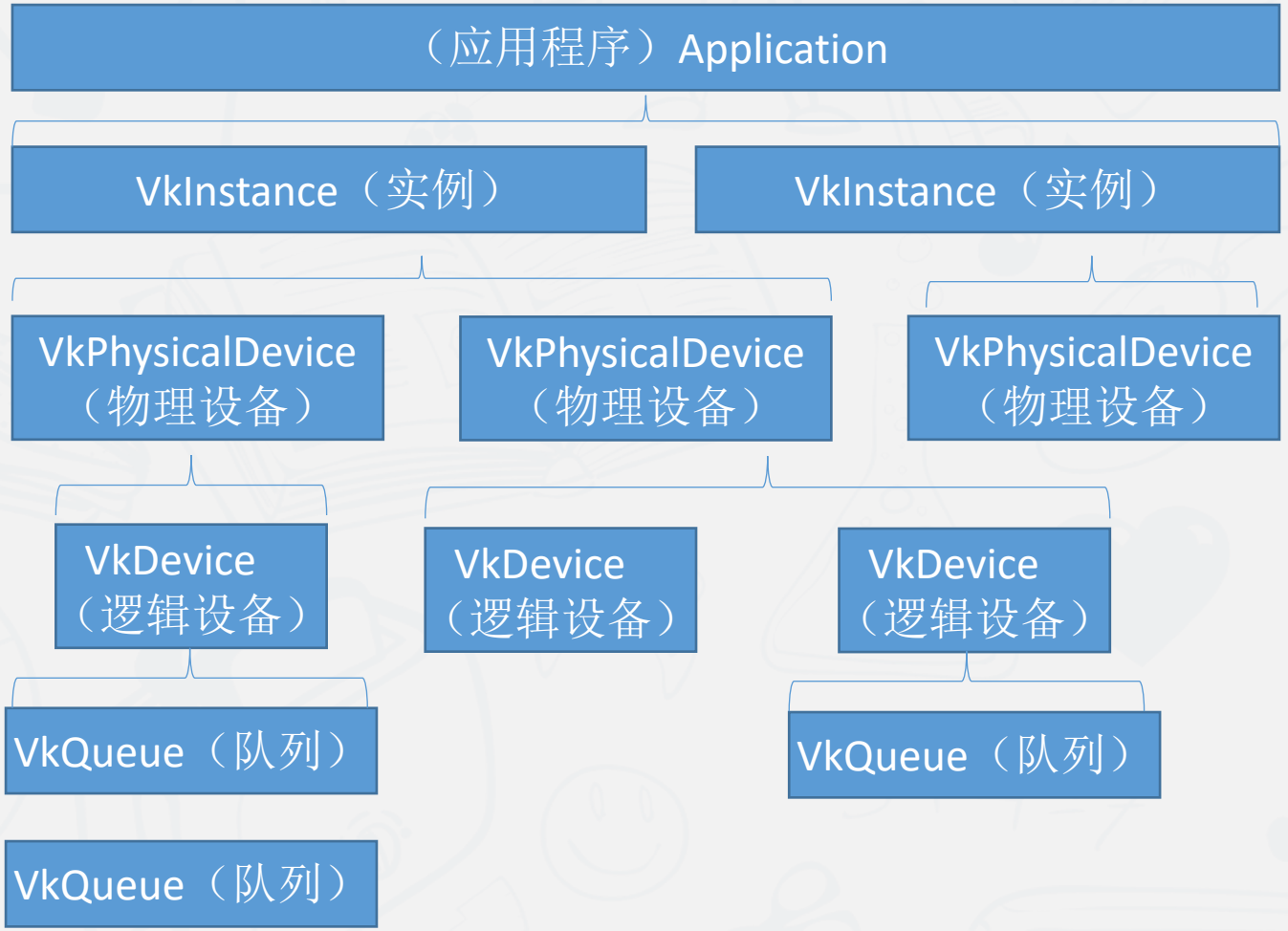
调用系统的API来创建窗口

- 通常需要关心窗口的类型是什么Win32，XCB还是WAYLAND。后续创建SwapChain会需要这些信息。
- SDL，GLFW等一些窗口第三方库会很方便的获得这些信息。甚至可以帮你完成很多Vulkan的初始化。

初始化Vulkan

Vulkan层次

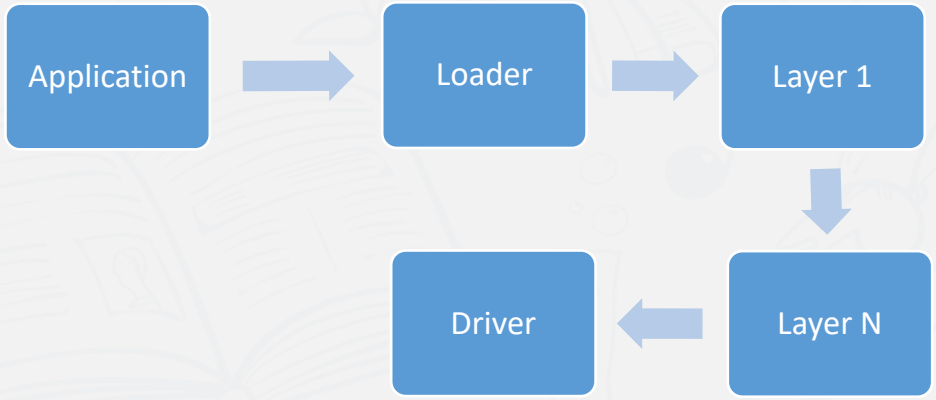
- 非常复杂的Vulkan使用场景。
- 通常情况
  - 一个实例。
  - 一个物理设备。
  - 一个逻辑设备。
  - 一个或者多个队列。
- Why so complicated？
  - 考虑一下如果你拥有四块4090的时候。



初始化Vulkan

VKInstance

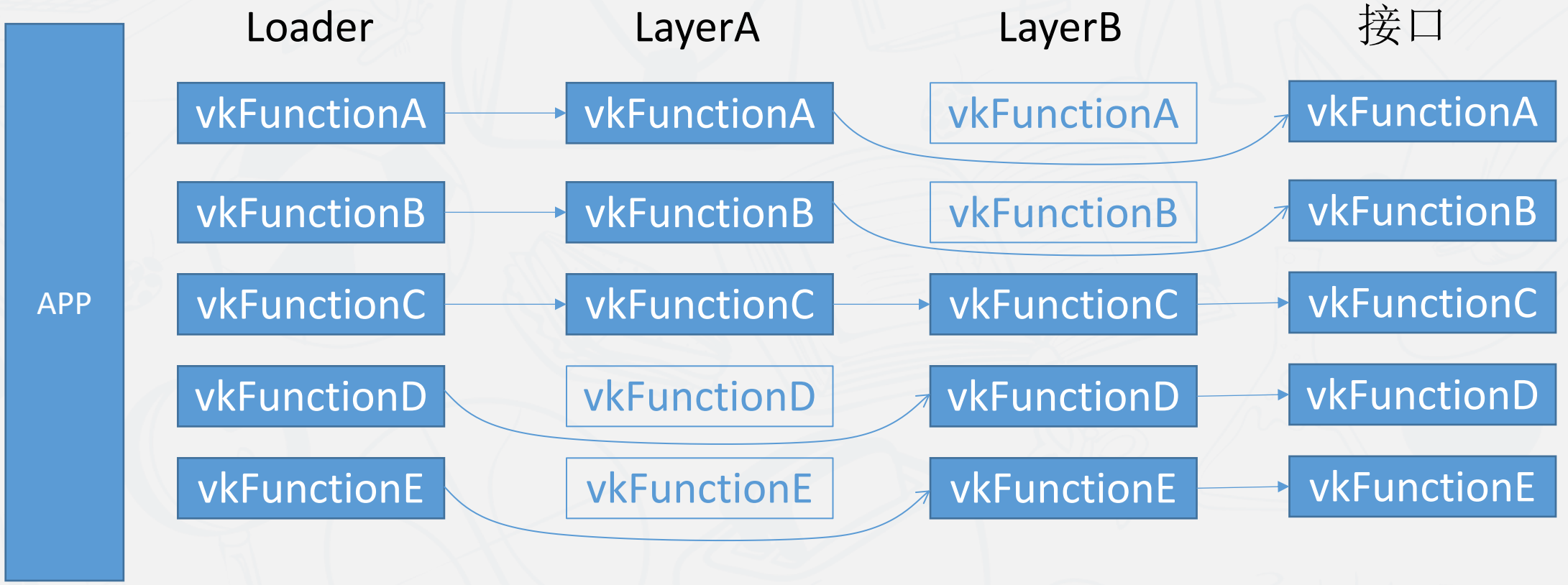
- 创建Vulkan Instance初始化Vulkan library。
  - Instance是程序和Vulkan库之间沟通的桥梁。
- 需要指定你需要开启的扩展
  - 创建swapchain的类型。
  - 给Vulkan对象设置名字的调试功能。
  - 检查Vulkan的错误调用的回调函数。
- Vulkan Layer
  - Vulkan将很多的功能拆分到了不同的layer中。
  - Validation layer: VK_LAYER_KHRONOS_validation
  - RenderDoc: VK_LAYER_RENDERDOC_Capture



相关API和结构体
vkCreateInstance，
VkInstanceCreateInfo
VkApplicationInfo
作业代码
VulkanExampleBase::createInstance()

初始化Vulkan

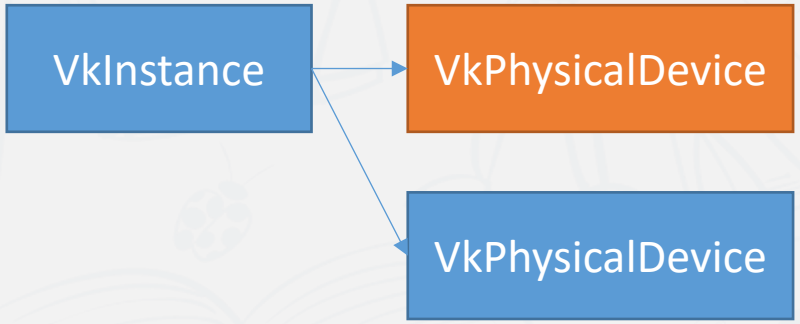
Layer



Layer给Vulkan提供了一个可以重写函数的方式。Layer可以重载对应的Vulkan函数，并且传给下一层Layer

初始化Vulkan

PhysicalDevice



- 获取Vulkan Instance和显卡的联系。
- Vulkan 提供一个能力检索当前计算机所有可用的显卡设备，以及他们的能力。
  - xxxxx
- 根据程序的需要判断需要最低的能力是否满足渲染的需要
  - 你的程序需要某一个特别的扩展，显卡无法提供支持，就可以选择初始化失败。
  - 在双显卡的笔记本中，选择独立显卡来驱动Vulkan

相关API和结构体
vkEnumeratePhysicalDevices
作业代码
VulkanExampleBase::initVulkan

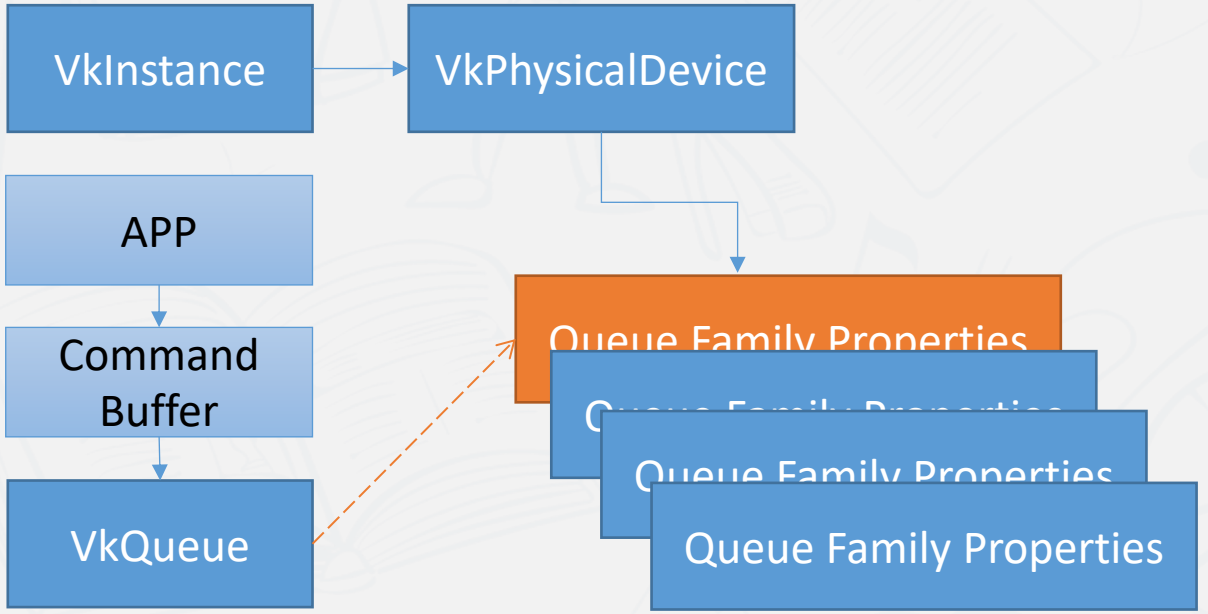
初始化Vulkan

QueueFamily

Vulkan所有的Command都需要上传到命令队列（VkQueue）中，包括绘制，数据上传指令等。不同类型的队列，来源不同的队列簇(Queue Family Properties)，每个队列簇只允许部分类型的Command。

- 支持图形操作
  - VK_QUEUE_GRAPHICS_BIT
- 支持计算着色器相关的操作
  - VK_QUEUE_COMPUTE_BIT
- 支持复制Buffer和Image的操作。
  - VK_QUEUE_TRANSFER_BIT

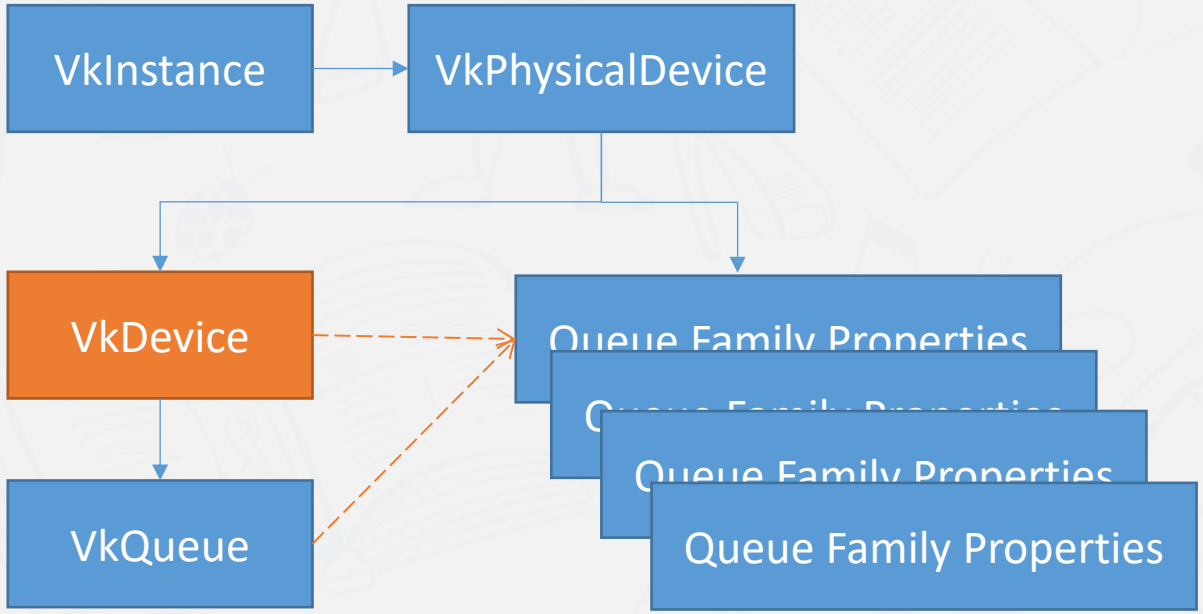
根据程序需要查询合适的队列簇，我们的示例程序需要的类型是graphics 和 transfer。



相关API和结构体
VkDeviceQueueCreateInfo
vkGetPhysicalDeviceQueueFamilyProperties
作业代码
VulkanDevice::createLogicalDevice

初始化Vulkan LogicDevice

- 程序有了物理设备之后，需要一个逻辑设备和Vulkan做交互。
- 根据VkPhysicalDevice查询到的队列簇，创建我们需要类型的队列簇。
- 可以选择可选的特性，在结构体VkPhysicalDeviceFeatures中看到由哪些可选项。这些可选项开启首先需要物理设备本身支持该项功能，通过vkGetPhysicalDeviceFeatures接口查询。
  - 几何着色器功能
  - 细分着色器功能
  - 压缩纹理
- 成功创建好逻辑设备之后，就可以从逻辑设备中获取命令队列。

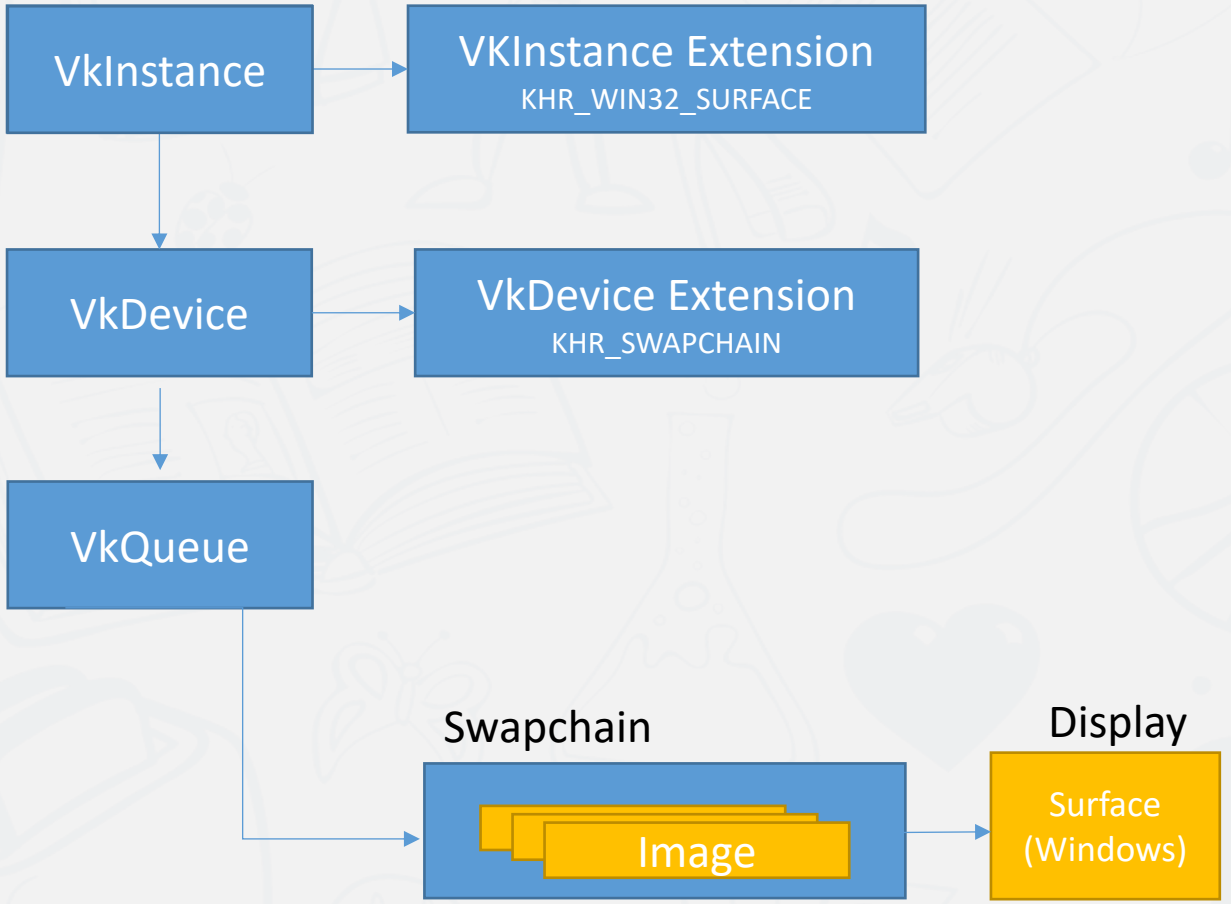


相关API和结构体
VkDeviceQueueCreateInfo
VkDeviceCreateInfo
vkCreateDevice
作业代码
VulkanDevice::createLogicalDevice

初始化Vulkan

Swapchain

- 首先要通过VkInstance 创建Surface，vulkan和显示窗口取得联系。
- 需要查询使用的队列簇是否同时支持surface和graphic
- 创建Swapchain和surface获取联系
- 创建swapchain的时候选择属性
  - 颜色空间: SRGB/HDR
  - 刷新模式: Vsync
  - 缓冲数量: Double Buffer/Triple Buffer
- 从Swapchain中获取缓冲的image，用于后面创建Frame Buffer用于渲染



作业代码
class VulkanSwapChain

绘制主循环

初始化 → 渲染主循环 → 退出程序

绘制主循环

绘制管线

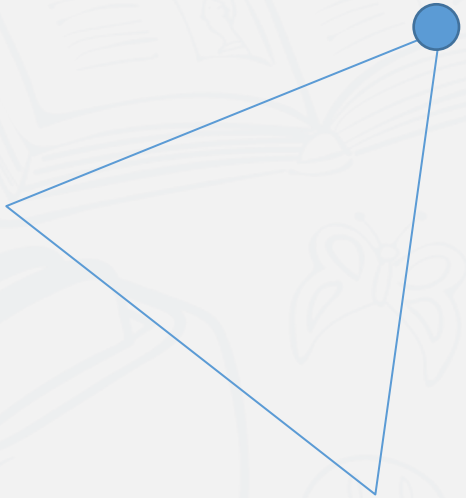
- 实时渲染的核心组件。
- 一般情况下，通过绘制管线，把几何，贴图，材质数据绘制到屏幕中。
- 这里显示的是最核心的绘制管线（不考虑细分，几何，mesh shader等一些进阶的功能。
- 不同的RHI，细节上会有一些不一样的概念，但整体的逻辑都是一致的。

绘制主循环

顶点输入

顶点输入

- 顶点数据缓存（vertex buffer）



- 顶点三维坐标
- 顶点法线
- 纹理
- 其他顶点属性

Vulkan对象

- 描述内存分布
  - VkPipelineVertexInputStateCreateInfo
- 设置顶点数据缓存
  - vkCmdBindVertexBuffers

绘制主循环

顶点输入

顶点输入

- 顶点数据缓存（vertex buffer）

Vulkan对象

- 描述内存分布
  - VkPipelineVertexInputStateCreateInfo
- 设置顶点数据缓存
  - vkCmdBindVertexBuffers

顶点数据缓存的排布对的vertex shader会有一定的性能影响
常用的优化算法库meshoptimizer

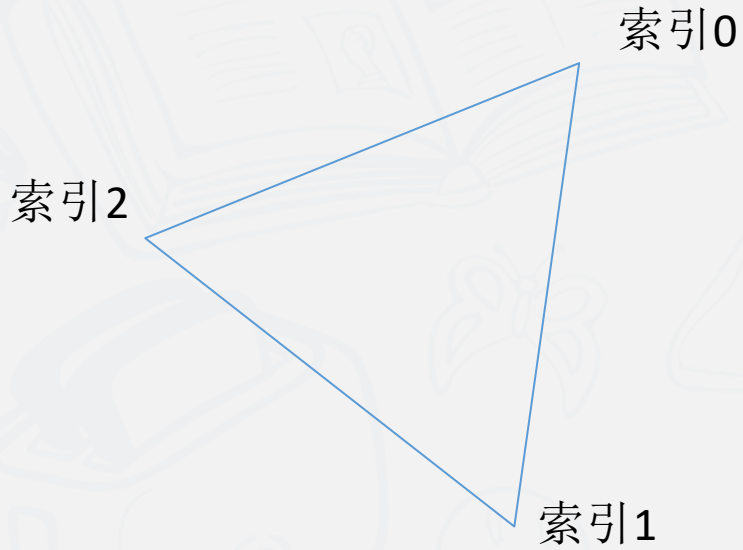
绘制主循环

顶点输入

顶点输入

- 顶点数据缓存（vertex buffer）
- 索引缓存（index buffer）

Vulkan对象
- 设置索引缓存
  - vkCmdBindIndexBuffer

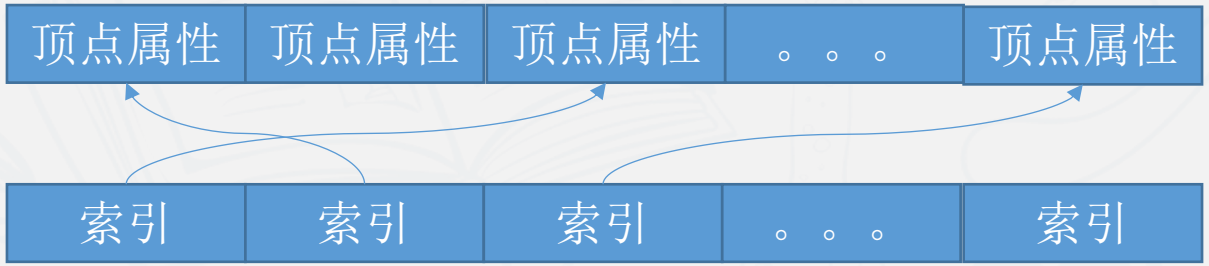


绘制主循环

顶点输入

顶点输入

- 顶点数据缓存（vertex buffer）
- 索引缓存（index buffer）



Vulkan对象

- 设置索引缓存
  - vkCmdBindIndexBuffer

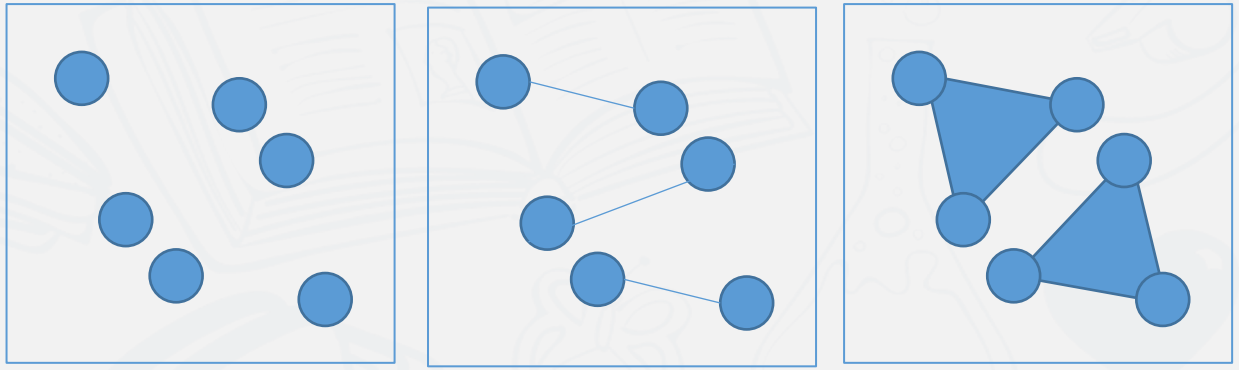
索引缓存对后续vertex shader也会有一定的性能影响，避免跳转区域过大，产生cache miss

绘制主循环

顶点输入

顶点输入

- 顶点数据缓存（vertex buffer）
- 索引缓存（index buffer）
- 图元拓扑类型（Primitive Topology）



Point List　Line List　triangle List

Vulkan对象

- 图元拓扑类型的枚举
  - VkPrimitiveTopology
- 设置图元拓扑类型
  - VkPipelineInputAssemblyStateCreateInfo::topology

绘制主循环

顶点输入

顶点输入

- 顶点数据缓存（vertex buffer）
- 索引缓存（index buffer）
- 图元拓扑类型（Primitive Topology）
- Draw，Draw Index，Draw Indirect， Draw Index Indirect

Vulkan对象
- vkCmdDraw
- vkCmdDrawIndexed
- vkCmdDrawIndirect
- vkCmdDrawIndexedIndirect

绘制主循环

光栅化流程

顶点着色器

- 顶点shader 经过编译器生成SPIR-V
  - 编译器
    - DirectX Shader Compiler
    - glslc/ glslang(glslangValidator)
  - 源代码可以是HLSL或者GLSL

Vulkan对象
- vkCreateShaderModule
- VkGraphicsPipelineCreateInfo::pStages
- VkPipelineShaderStageCreateInfo

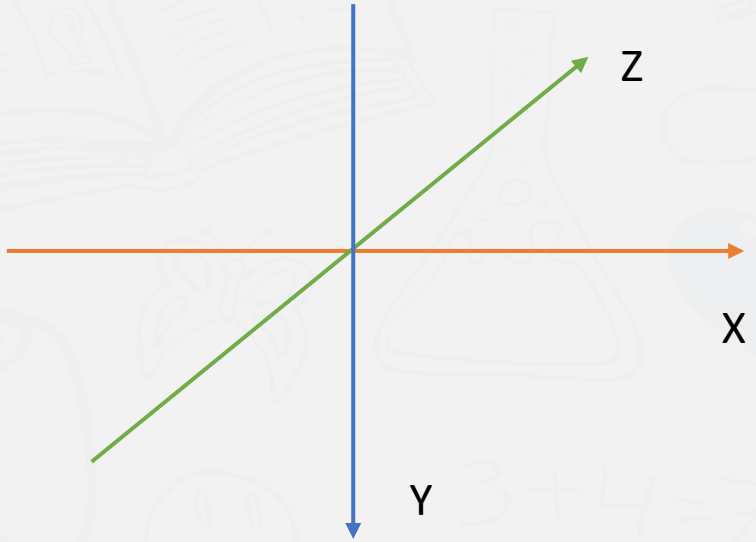
```
#version 450

layout (location = 0) in vec3 inPos;
layout (location = 1) in vec3 inColor;

...
layout (binding = 0) uniform UBO
{
    mat4 projectionMatrix;
    mat4 modelMatrix;
    mat4 viewMatrix;
} ubo;

layout (location = 0) out vec3 outColor;

...
out gl_PerVertex
{
    vec4 gl_Position;
};


void main()
{
    outColor = inColor;
    gl_Position = ubo.projectionMatrix * ubo.viewMatrix * ubo.modelMatrix * vec4(v0: inPos.xyz, v1: 1.0);
}
```

绘制主循环

光栅化流程

顶点着色器

- 顶点shader 经过编译器生成SPIRV
- 主要用于计算顶点位置，经过矩阵变换到屏幕空间的位置。
- 顶点的其他属性的计算。



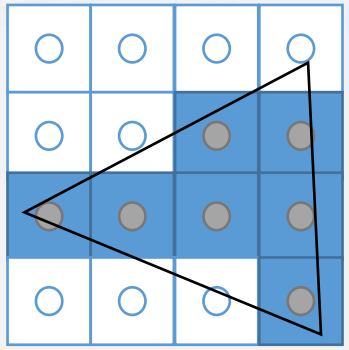
Vulkan NDC坐标系

绘制主循环

光栅化流程

Rasterization State

- 填充方式（Polygon Mode）
- 剔除方式（Cull Mode）
- 正方向（Front Face）
- 线宽度
- Depth Bias（用于解决z fighting）



Vulkan对象

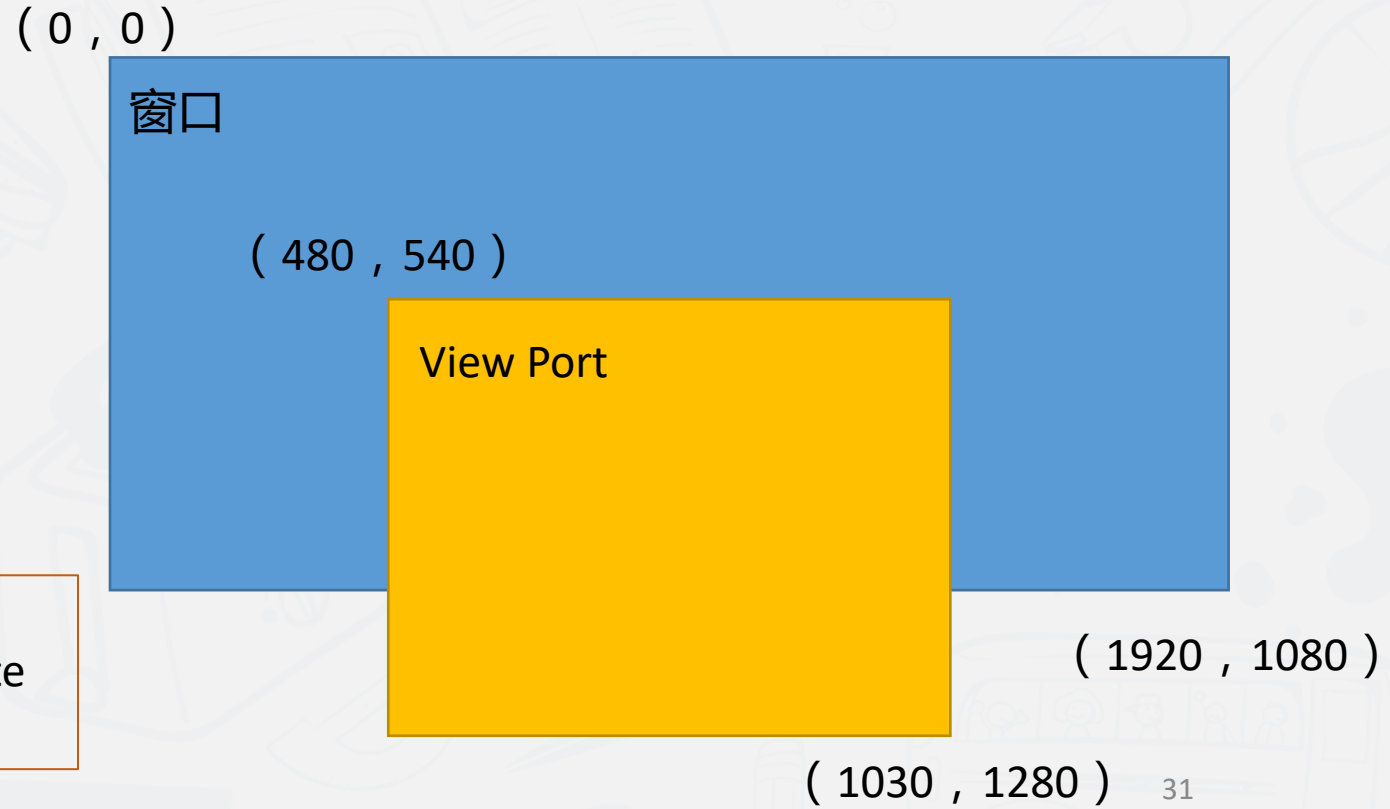
- VkGraphicsPipelineCreateInfo::pRasterizationState
- VkPipelineRasterizationStateCreateInfo

绘制主循环

光栅化流程

ViewPort

- 设置窗口真正绘制的区域。
- 一般情况下viewport区域和窗口是一致的。



Vulkan对象

- VkGraphicsPipelineCreateInfo::pViewportState
- VkPipelineViewportStateCreateInfo

绘制主循环

光栅化流程

Viewport

- 设置窗口真正绘制的区域。
- 一般情况下viewport区域和窗口是一致的。
- Cascaded Shadow Maps通常通过控制View Port实现。通过设置一个非常大的View Port，达到增加部分区域分辨率的目的

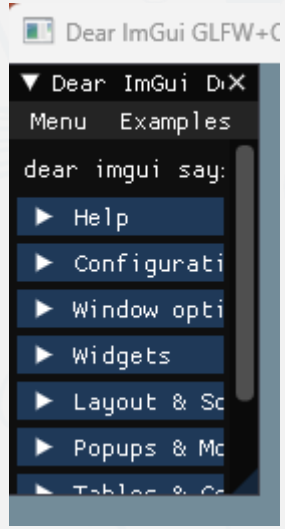
绘制主循环

光栅化流程

Scissor

- 在Viewport的基础下裁剪出会被绘制的部分。
- 在UI绘制中会经常广泛使用。

关闭Scissor



开启Scissor

Vulkan对象

- VkPipelineViewportStateCreateInfo::pScissors

绘制主循环

光栅化流程

片段着色器

- 绘制管线中最核心的部分，绝大部分的计算都会在这里实现。
- 来自Vertex Shader的顶点属性，经过插值之后作为输入。
- 纹理，一般用于颜色，有时候会用于数据的编码。
- 缓冲区Buffer，uniform buffer（std140），storage buffer（std430），push_constant
- 移动端平台需要注意浮点数的精度。减少需要的带宽。

Vulkan对象

- vkCreateShaderModule
- VkGraphicsPipelineCreateInfo::pStages
- VkPipelineShaderStageCreateInfo

绘制主循环

光栅化流程

片段输出

- 把fragment shader的输出绘制到对应的render target中。
- Target blend， fragment shader和render target中已有的颜色做混合。

Depth State / Stencil State

- 深度检测和模板测试。
- Fragment shader中不做discard或者修改深度的操作，一般会提前到fragment shader之前。

Vulkan对象

- 定义输出的格式以及IO
  - VkRenderPass
- 定义输出的对象
  - VkFramebuffer

- 混合
  - VkGraphicsPipelineCreateInfo::pColorBlendState
  - VkPipelineColorBlendStateCreateInfo

- 深度和模板检查
  - VkGraphicsPipelineCreateInfo::pDepthStencilState
  - VkPipelineDepthStencilStateCreateInfo

绘制主循环

Vulkan流程

Heap

Memory

Memory

Images

Buffer

FrameBuffer

State
Shaders
Pass
Input layout
...

Samplers

Images

Buffers

Pipeline

DescriptorSets

DescriptorPool

RenderPass Begin

Bind Vertex/Index

Set Viewport

Bind Pipeline

Bind DescriptorSet

Draw

RenderPass End

CommandBuffer

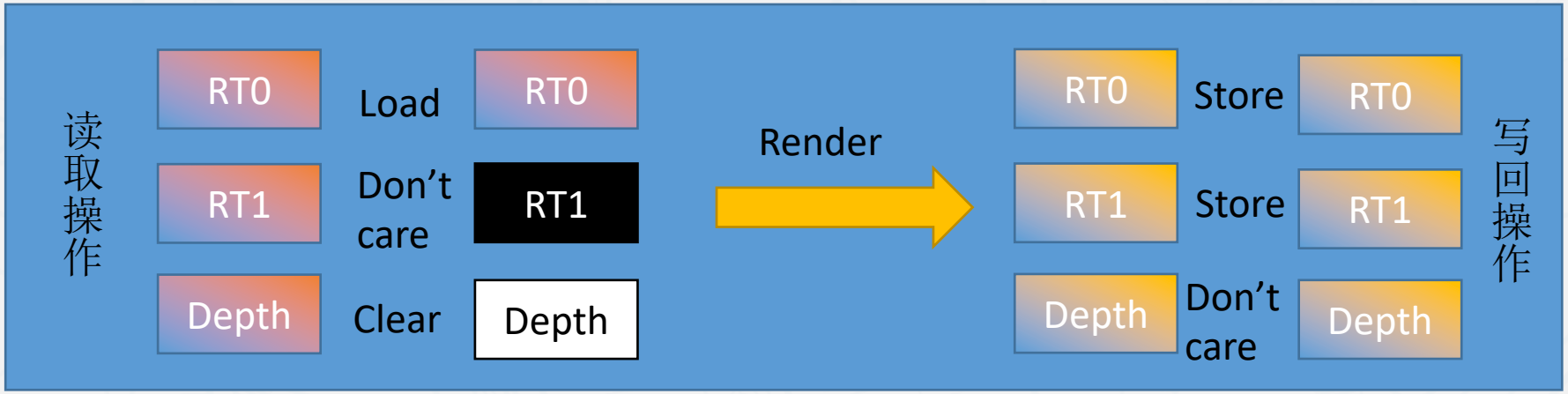
Queue

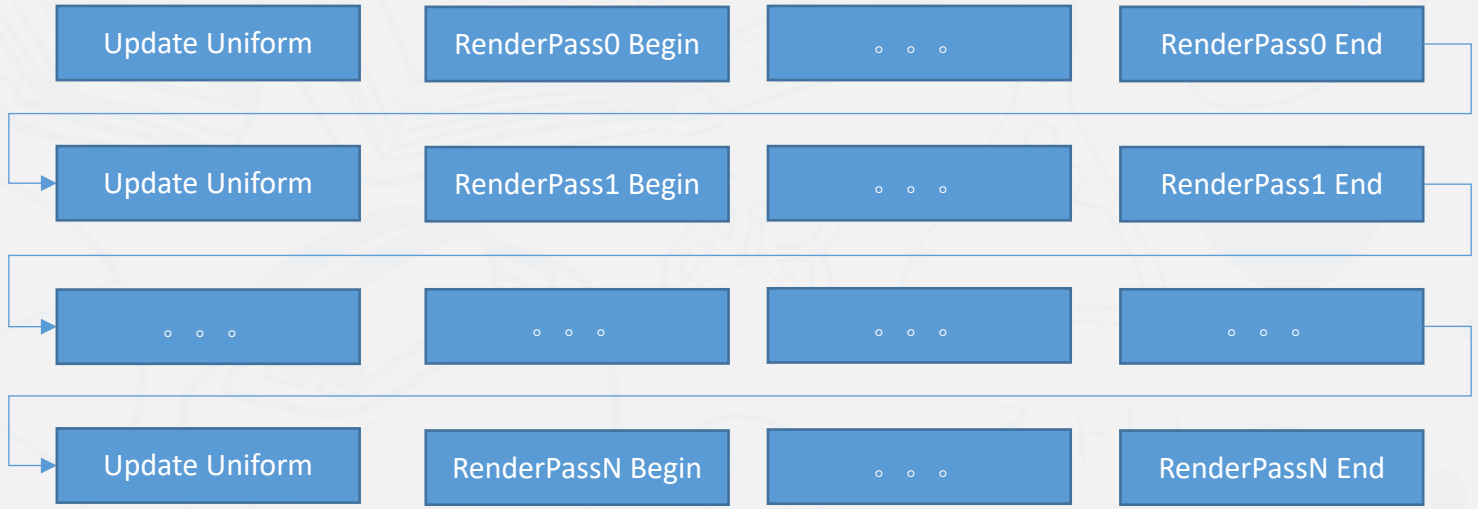
Device

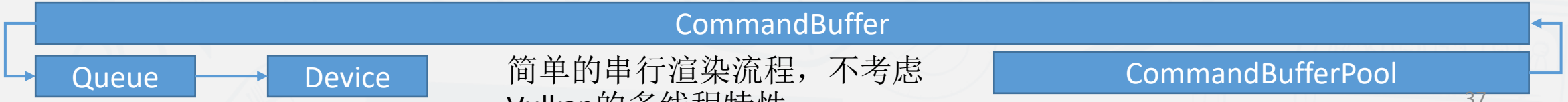
CommandBufferPool

绘制主循环

- RenderPass 定义一次图形绘制的最小单元。
- 定义了绘制管线的输出FrameBuffer（Render Target）。
- Vulkan定义了FrameBuffer的IO操作。
  - 在开始的时候是否需要保持FrameBuffer中原有的数据。
  - 在绘制结束之后是否要把FrameBuffer的数据写回。
  - 这个特性在移动端一些显卡带宽小的设备中非常重要。







简单的串行渲染流程，不考虑Vulkan的多线程特性。

参考资料

参考资料

Vulkan 参考资料

- 《Vulkan应用开发指南》
- Brief guide to Vulkan layers (renderdoc.org)
- Vulkan in 30 minutes (renderdoc.org)
- vulkan11-reference-guide.pdf (khronos.org)
- Vulkan® 1.3.246 - A Specification (khronos.org)
- Vulkan 教程|极客教程 (geek-docs.com)

Vulkan示例代码

- GitHub - SaschaWillems/Vulkan: Examples and demos for the new Vulkan API
- GitHub - KhronosGroup/Vulkan-Samples: One stop solution for all Vulkan samples
- GitHub - google/filament: Filament is a real-time physically based rendering engine for Android, iOS, Windows, Linux, macOS, and WebGL2

Vulkan调试工具

- RenderDoc
- Nvidia Nsight Tool
- AMD Radeon GPU Profiler
- Qualcomm Snapdragon Profiler
- Arm Mobile Studio

谢谢