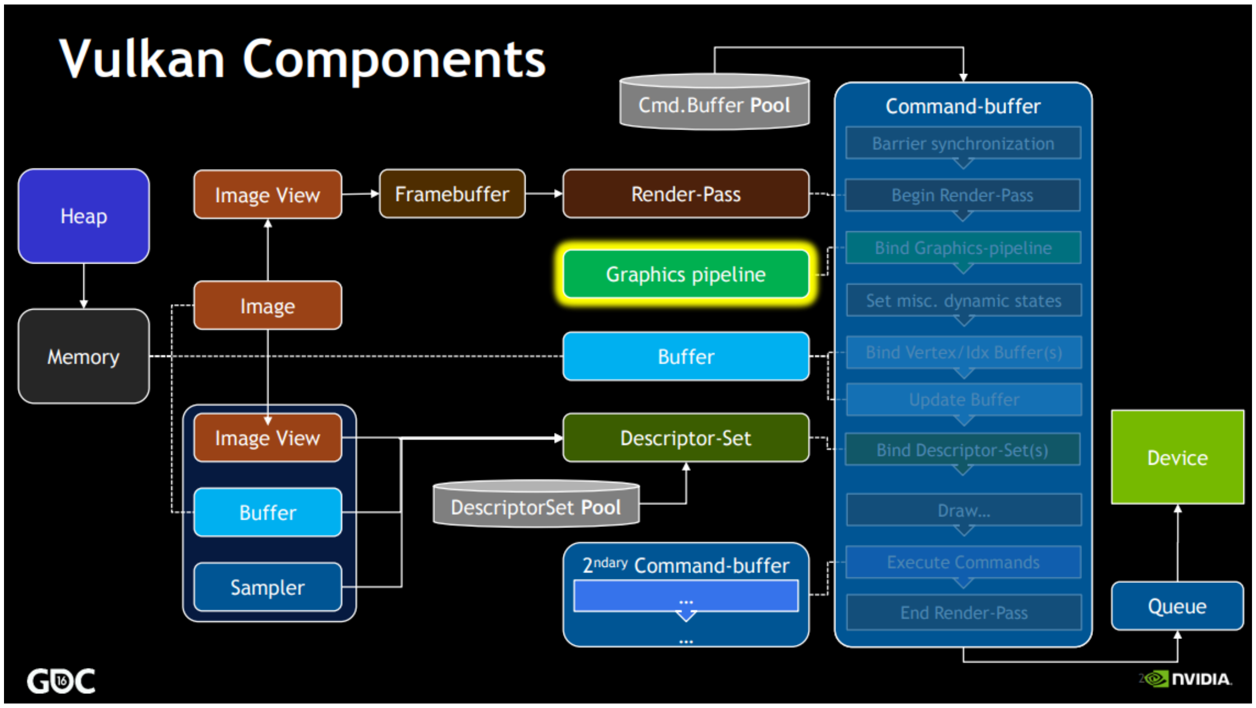
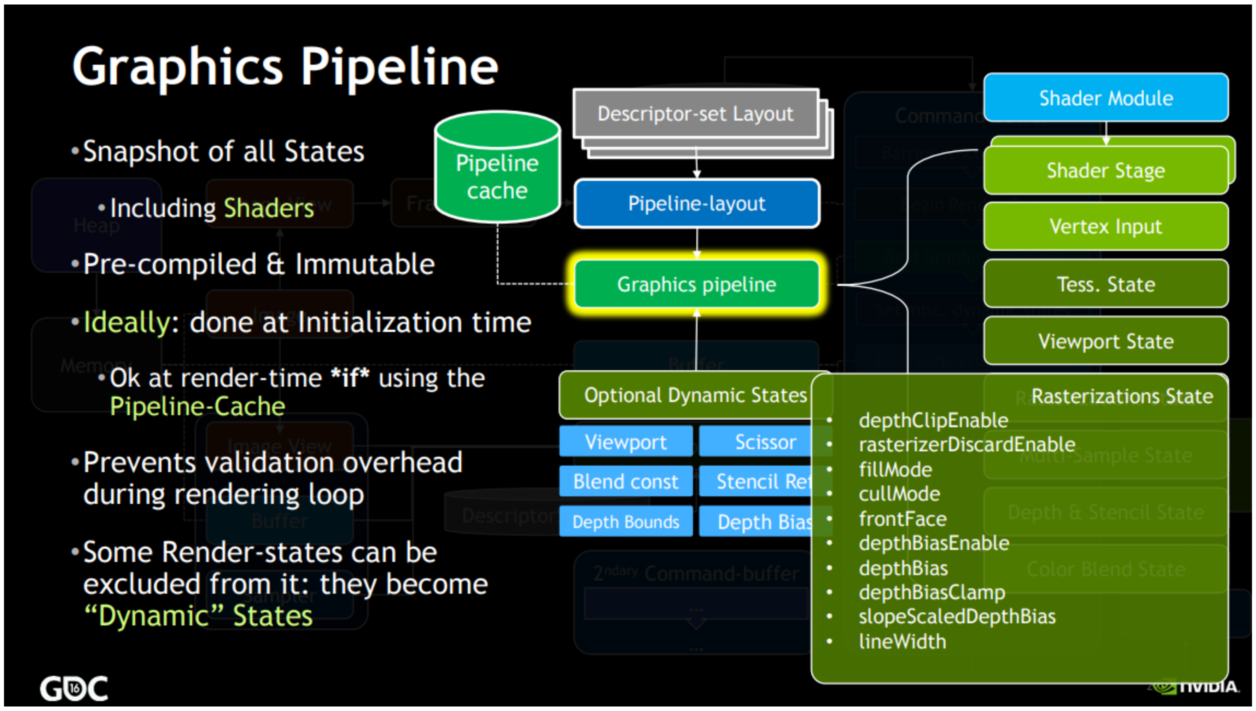
Graphics Pipeline

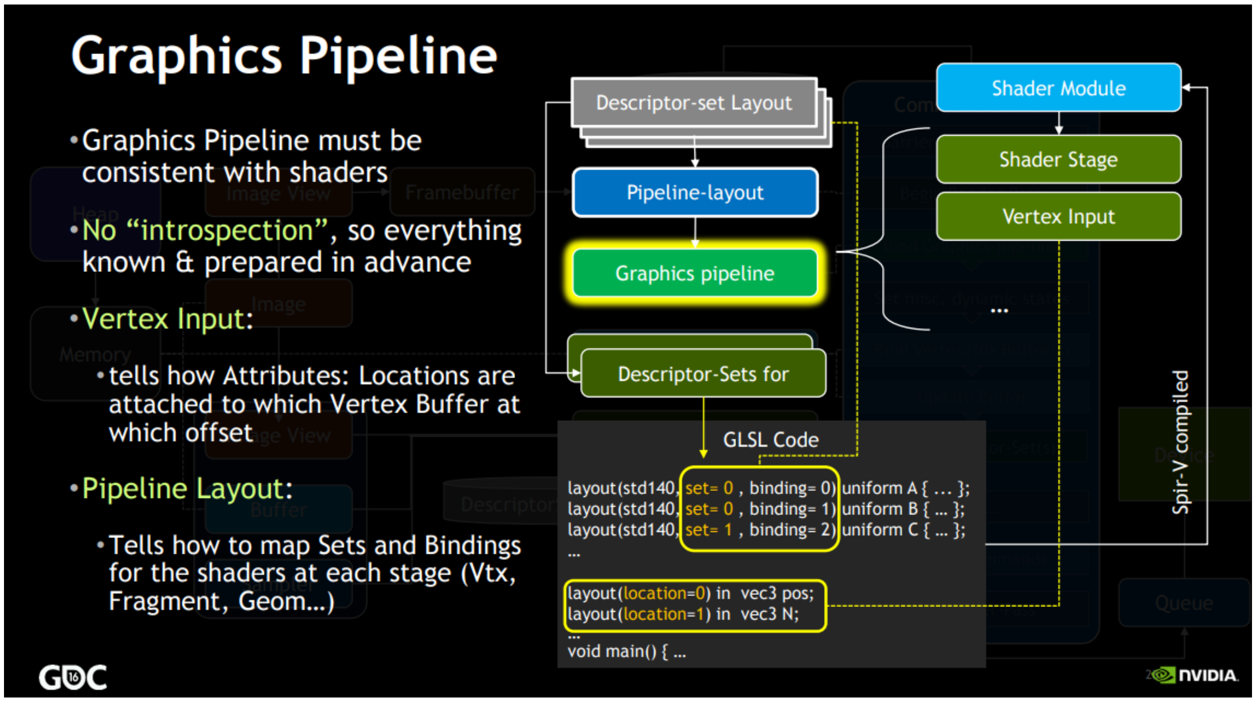


Vulkan Components



Graphics Pipeline

- Snapshot of all States
  - Including Shaders
- Pre-compiled & Immutable
- Ideally: done at Initialization time
  - Ok at render-time *if* using the Pipeline-Cache
- Prevents validation overhead during rendering loop
- Some Render-states can be excluded from it: they become "Dynamic" States



Graphics Pipeline

- Graphics Pipeline must be consistent with shaders
- No “introspection”, so everything known & prepared in advance
- Vertex Input:
  - tells how Attributes: Locations are attached to which Vertex Buffer at which offset
- Pipeline Layout:
  - Tells how to map Sets and Bindings for the shaders at each stage (Vtx, Fragment, Geom...)

```
layout(std140, set= 0 , binding= 0) uniform A { ... };
layout(std140, set= 0 , binding= 1) uniform B { ... };
layout(std140, set= 1 , binding= 2) uniform C { ... };
...
layout(location=0) in  vec3 pos;
layout(location=1) in  vec3 N;
...
void main() { ...
```

以
https://github.com/GameTechDev/IntroductionToVulkan/tree/master/Project/Tutorials/04
例子中函数CreatePipeline来介绍.

Vulkan中的管线分为两种：Compute Pipeline 和 Graphics Pipeline。Compute Pipeline 用于异构并行计算，Graphics Pipeline 用于绘制渲染。Graphics Pipeline 可以由上图得知大概分为以下三个步骤:

- 提供shader
- 绑定资源
- 管理状态

VkPipeline 的创建需初始化VkGraphicsPipelineCreateInfo结构体：

```
typedef struct VkGraphicsPipelineCreateInfo {
    VkStructureType                                  sType;
    const void*                                      pNext;
    VkPipelineCreateFlags                            flags;
    uint32_t                                         stageCount;
    const VkPipelineShaderStageCreateInfo*           pStages;
    const VkPipelineVertexInputStateCreateInfo*      pVertexInputState;
    const VkPipelineInputAssemblyStateCreateInfo*    pInputAssemblyState;
    const VkPipelineTessellationStateCreateInfo*     pTessellationState;
    const VkPipelineViewportStateCreateInfo*         pViewportState;
    const VkPipelineRasterizationStateCreateInfo*    pRasterizationState;
    const VkPipelineMultisampleStateCreateInfo*      pMultisampleState;
    const VkPipelineDepthStencilStateCreateInfo*     pDepthStencilState;
    const VkPipelineColorBlendStateCreateInfo*       pColorBlendState;
    const VkPipelineDynamicStateCreateInfo*          pDynamicState;
    VkPipelineLayout                                 layout;
    VkRenderPass                                     renderPass;
    uint32_t                                         subpass;
    VkPipeline                                       basePipelineHandle;
    int32_t                                          basePipelineIndex;
} VkGraphicsPipelineCreateInfo;
```

由创建的函数参数可知,不同的步骤用不同的结构体表示,最后将通过VkGraphicsPipelineCreateInfo来进行汇总创建,具体参数请自行参考示例代码

参考链接:https://zhuanlan.zhihu.com/p/49112352