学习Vulkan API 与其它API一样，一个重要部分是了解其中定义了哪些类型的对象，它们代表什么以及它们之间如何相互关联。为了帮助解决这个问题，我们创建了一张图，显示了所有Vulkan对象以及某些关系，尤其是创建另一个对象的顺序。

每个Vulkan对象都是带有前缀Vk的特定类型的值。为了清楚起见，这些前缀在图中被省略了，就像vk 函数名的前缀一样 。例如，图中的Sampler表示存在一个名为VkSampler的Vulkan对象类型 。这些类型不应视为指针或序数。您不应该以任何方式解释它们的值。只是将它们视为不透明的句柄，将它们从一个函数传递给另一个函数，当然不要忘了在不再需要它们时销毁它们。绿色背景的对象没有自己的类型。相反，它们由 其父对象内部uint32\_t类型的数字索引表示 ，例如QueryPool中的Queries (查询)。

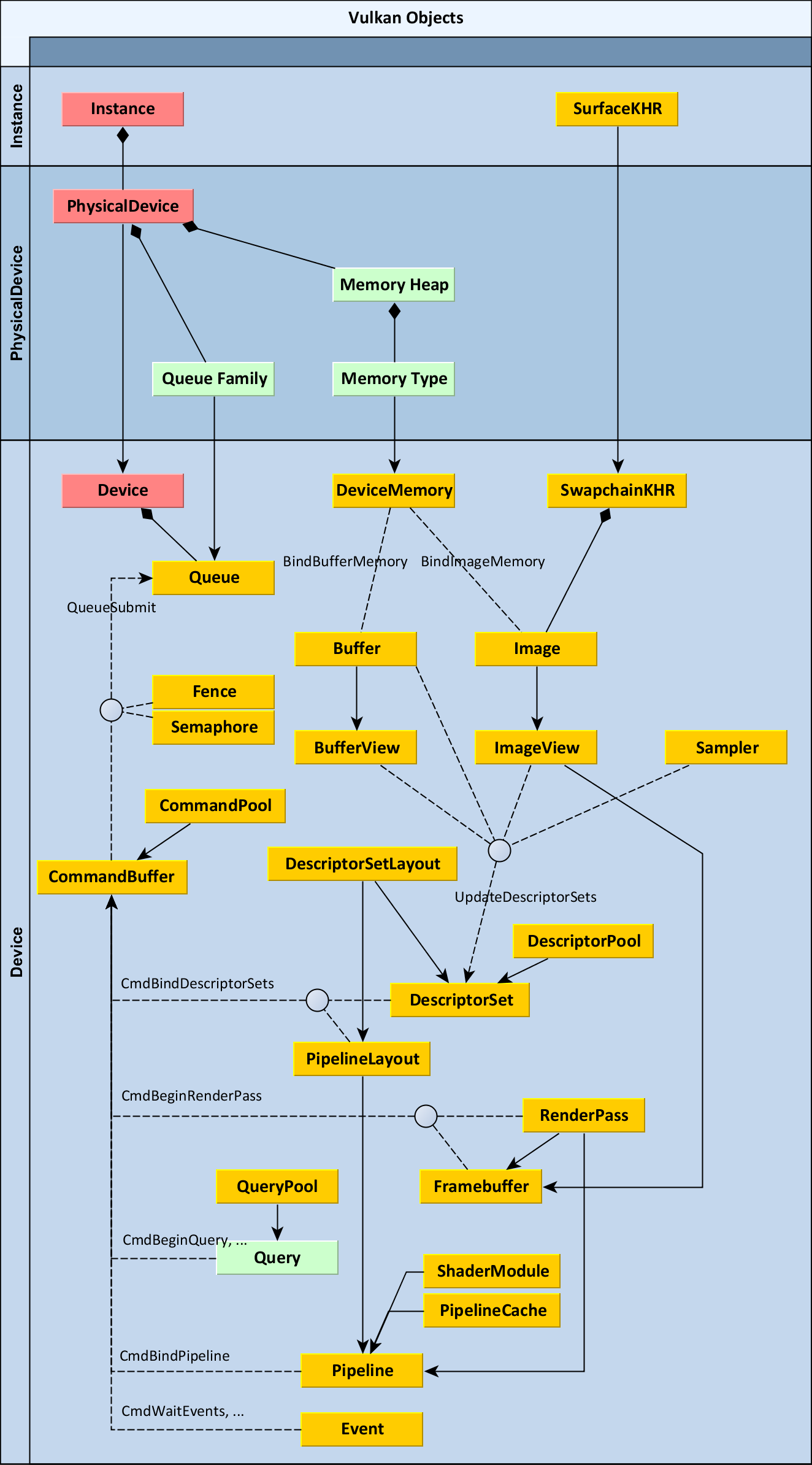
带有箭头的实线表示创建的顺序。例如，您必须指定一个现有的DescriptorPool来创建DescriptorSet。

带有菱形的实线表示合成，这意味着您不必创建该对象，但是它已经存在于其父对象中并且可以从其获取。例如，您可以从Instance对象枚举PhysicalDevice对象。

虚线表示其他关系，例如向CommandBuffer提交各种命令。

该图分为三个部分。每个部分都有一个主要对象，以红色显示。每部分中的所有其他对象都是直接或间接从该主要对象创建的。例如， vkCreateSampler –创建采样器的函数–将 VkDevice 其作为第一个参数。为了清楚起见，未在此图上绘制与主要对象的关系。

1. vulkan对象关系图
2. 参考链接：https://gpuopen.com/learn/understanding-vulkan-objects/



这是所有对象的简要说明：

1. **[Instance](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkInstance.html)([实例](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkInstance.html))**是您创建的第一个对象。它表示从应用程序到Vulkan运行时的连接，因此在应用程序中仅应存在一次。它还存储使用Vulkan所需的所有特定于应用程序的状态。因此，您必须指定创建实例时要启用的所有层（例如验证层）和所有扩展。
2. **[PhysicalDevice](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkPhysicalDevice.html)(物理设备)**表示特定的Vulkan兼容设备，例如图形卡。您可以从“实例”列举这些，然后你就可以查询它们的 vendorID ， deviceID 和支持的功能，以及其他属性和限制。

枚举系统上的物理设备-vkEnumeratePhysicalDevices:

|  |  |  |  |
| --- | --- | --- | --- |
| 实例对象 | | | |
| 物理设备0 | 物理设备1 | 物理设备2 | 物理设备3 |

物理设备可以枚举所有可用的**队列族**类型。图形队列是主要的队列，但是您可能还有其他仅支持计算或传输的队列。

物理设备还可以枚举其中的内存堆和内存类型。一个**内存堆代表特定的RAM池。**。它可能会提取主板上的系统RAM或专用图形卡上的视频RAM中的特定内存空间，或者实现要公开的任何其他特定于主机或设备的内存。分配内存时必须指定**内存类型**。它具有对内存Blob的特定要求，例如对主机可见，一致（在CPU和GPU之间）和已缓存。这些可能会任意组合，具体取决于设备驱动程序。

1. **[Device](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkDevice.html)([设备](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkDevice.html))**可以被视为逻辑设备或打开的设备。它是主要对象，代表已准备好创建所有其他对象的已初始化Vulkan设备。这与DirectX®中的Device对象类似。在设备创建期间，您需要指定要启用的功能，其中一些功能是基本功能，例如各向异性纹理过滤。您还必须说明将要使用的所有队列，它们的编号和它们的队列族。
2. **[Queue](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkQueue.html)([队列](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkQueue.html))**是一个对象，代表要在设备上执行的命令的队列。通过 填充CommandBuffer并使用函数vkQueueSubmit将其提交到队列，从而请求GPU完成所有实际工作 。如果您有多个队列，例如主图形队列和计算队列，则可以向每个队列提交不同的CommandBuffer。这样，您就可以启用异步计算，如果执行正确的话，可以大大提高速度。
3. **[CommandPool](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkCommandPool.html)**(**命令池**)是一个简单的对象，用于分配CommandBuffer。它已连接到特定的队列族。
4. **[CommandBuffer](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkCommandBuffer.html)**(**命令缓冲区**)是从特定的CommandPool分配的。它代表逻辑设备要执行的各种命令的缓冲区。您可以在命令缓冲区上调用各种函数，所有函数均以vkCmd 开头 。它们用于指定将CommandBuffer提交到队列并最终由设备使用时应执行的任务的顺序，类型和参数。
5. **[Sampler](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkSampler.html)**(**[采样器](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkSampler.html)**)未绑定到任何特定图像。它只是一组状态参数，例如过滤模式（最近或线性）或寻址模式（重复，钳位到边缘，钳位到边界等）。

Buffer and Image(缓冲区和图像)是占用设备内存的两种资源。

1. **Buffer** (**[缓冲区](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkBuffer.html))**比较简单。它是任何具有其长度（以字节表示）的二进制数据的容器。
2. **Image**(**[图像](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkImage.html))**，另一方面，表示一组像素。这是其他图形API中称为纹理的对象。需要更多参数来指定图像的创建。它可以是一维，二维或三维，具有各种像素格式（如 R8G8B8A8\_UNORM 或 R32\_SFLOAT ），并且还可以包含许多离散的图像，因为它可以具有多个阵列层或MIP级别（或两者）。图像是一种单独的对象类型，因为它不一定只包含可以直接访问的线性像素集。图像可以具有由图形驱动程序管理的不同的特定于实现的内部格式（平铺和布局）。

创建一定长度的缓冲区或具有特定尺寸的图像不会自动为其分配内存。这是一个三步过程，必须由您手动执行。您也可以选择使用我们的[Vulkan内存分配器](https://github.com/GPUOpen-LibrariesAndSDKs/VulkanMemoryAllocator)库，该库将为您分配资源。

1. 分配DeviceMemory，
2. 创建缓冲区或图像，
3. 使用函数vkBindBufferMemory 或vkBindImageMemory 将 它们绑定在一起 。

因此，您还必须创建一个[DeviceMemory](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkDeviceMemory.html)对象。它代表从特定内存类型（由PhysicalDevice支持）分配的内存块，具有特定的字节长度。您不应该为每个缓冲区或图像分配单独的DeviceMemory。相反，您应该分配更大的内存块，并将其中的一部分分配给缓冲区和图像。分配是一项昂贵的操作，并且最大分配数量也受到限制，所有这些都可以从PhysicalDevice中查询。

为每个图像分配和绑定DeviceMemory的义务的一个例外是交换链的创建。这是用于在操作系统上绘制到屏幕上或窗口内的最终图像的概念。因此，创建它的方式取决于平台。如果您已经拥有了使用系统API初始化的窗口，则首先需要创建一个**SurfaceKHR**对象。它需要Instance对象以及一些与系统相关的参数。例如，在Windows上，这些是：实例句柄（ HINSTANCE ）和窗口句柄（ HWND ）。您可以将SurfaceKHR对象想象为窗口的Vulkan表示。

从**SurfaceKHR**对象可以创建**SwapchainKHR**。该对象需要一个设备。它表示可以在表面上显示的一组图像，例如，使用双缓冲或三缓冲。您可以从交换链中查询包含的图像。这些映像已经由系统分配了其后备内存。

缓冲区和图像并非总是直接在渲染中使用。在它们之上还有另一层，称为视图。您可以像在数据库中查看视图那样思考它们-可用于以所需方式查看一组基础数据的参数集。[BufferView](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkBufferView.html)是基于特定缓冲区创建的对象。您可以在创建期间传递偏移量和范围，以将视图限制为仅缓冲区数据的子集。同样，[ImageView](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkImageView.html)是一组引用特定图像的参数集。在那里，您可以将像素解释为具有其他（兼容）格式，对所有组件进行调整，并将视图限制为特定范围的MIP级别或阵列层。

着色器可以通过描述符访问这些资源（缓冲区，图像和采样器）。描述符并不独立存在，而是始终按描述符集分组。但是在创建描述符集之前，必须通过创建[DescriptorSetLayout](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkDescriptorSetLayout.html)来指定其布局，该布局的行为类似于描述符集的模板。例如，被渲染通道用于绘制3D几何图形的着色器可能期望：

|  |  |
| --- | --- |
| 绑定槽 | 资源资源 |
| 0 | 一个统一的缓冲区（在DirectX中称为常量缓冲区）可用于顶点着色器阶段。 |
| 1 | 片段着色器阶段可以使用的另一个统一缓冲区。 |
| 2 | 采样图像。 |
| 3 | 采样器，也可用于片段着色器阶段。 |

您还需要创建一个[DescriptorPool](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkDescriptorPool.html)。这是一个用于分配描述符集的简单对象。创建描述符池时，必须指定最大的描述符集和要从中分配的不同类型的描述符。

最后，分配一个[DescriptorSet](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkDescriptorSet.html)。您需要传递DescriptorPool对象和DescriptorSetLayout对象。DescriptorSet表示保存实际描述符的内存，可以对其进行配置，以使描述符指向特定的Buffer，BufferView，Image或Sampler。您可以使用函数vkUpdateDescriptorSets来实现  。

可以将多个DescriptorSet绑定为CommandBuffer中的活动集，以供渲染命令使用。为此，请使用函数 vkCmdBindDescriptorSets 。此函数还需要另一个对象– [PipelineLayout](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkPipelineLayout.html)，因为可能有多个DescriptorSet绑定，并且Vulkan想提前知道它应该预期它们的数量和类型。PipelineLayout根据将哪种类型的描述符集绑定到CommandBuffer来表示渲染管线的配置。您从DescriptorSetLayouts数组创建它。

在其他图形API中，您可以采用立即模式方法，并且仅呈现列表中接下来的内容。这在Vulkan中是不可能的。相反，您需要预先计划框架的渲染，并将其组织为通道和子通道。子通道不是单独的对象，因此我们在这里不再讨论，但它们是Vulkan渲染系统的重要组成部分。幸运的是，在准备工作负载时，您不需要了解所有详细信息。例如，您可以指定要提交时渲染的三角形的数量。在Vulkan中定义[RenderPass](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkRenderPass.html)时，至关重要的部分是将在该过程中使用的附件的数量和格式。

附件是Vulkan的名称，即您可能称为渲染目标的名称-用作渲染输出的图像。您无需在此处指向特定的图像-您只需描述其格式即可。例如，一个简单的渲染通道可以使用格式为R8G8B8A8\_UNORM的颜色附件和格式为D16\_UNORM的深度模板附件  。您还可以指定在通道开始时应保留，丢弃还是清除附件的内容。

1. **[Framebuffer](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkFramebuffer.html)**(**[帧缓冲区](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkFramebuffer.html)**)（不要与SwapchainKHR混淆）表示指向可以用作附件（渲染目标）的实际图像的链接。您可以通过指定RenderPass和一组ImageViews创建一个Framebuffer对象。当然，它们的数量和格式必须符合RenderPass的规范。帧缓冲区是图像之上的另一层，基本上将这些ImageView分组在一起，以在渲染特定RenderPass时作为附件绑定。每当您开始渲染RenderPass时，就调用函数 vkCmdBeginRenderPass ，并将帧缓冲区也传递给它。
2. **[Pipeline](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkPipeline.html)**(**管线**)是大对象，因为它组合了前面列出的大多数对象。它代表整个管线的配置，并且具有很多参数。其中之一是PipelineLayout –它定义描述符和推送常量的布局。管线有两种类型-ComputePipeline和GraphicsPipeline。ComputePipeline是简单的，因为它所支持的只是纯计算程序（有时称为计算着色器）。GraphicsPipeline更为复杂，因为它包含所有参数，例如顶点，片段，几何形状，计算和细分（如果适用），以及顶点属性，图元拓扑，背面剔除和混合模式等，仅举几例。所有这些参数曾经是许多较旧的图形API（DirectX 9，OpenGL）中的单独设置，后来随着API的发展（DirectX 10和11）被分组为较少数量的状态对象，并且现在必须像当今的Vulkan这样的现代API一起烘焙为单个大的，不变的对象。对于渲染期间所需的每个不同参数集，必须创建一个新的管线。然后可以通过调用函数 vkCmdBindPipeline将其设置为CommandBuffer中的当前活动管道 。

在Vulkan中，着色器编译是一个多阶段过程。首先，Vulkan不支持任何高级着色语言，例如GLSL或HLSL。相反，Vulkan接受任何高级语言都可以发出的称为SPIR-V的中间格式。使用SPIR-V中充满数据的缓冲区来创建[ShaderModule](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkShaderModule.html)。该对象代表一段着色器代码，可能采用部分编译的形式，但它还不能执行GPU。仅在为要使用的每个着色器阶段（顶点，曲面细分控制，曲面细分评估，几何图形，片段或计算）创建管线时，才指定ShaderModule以及入口点函数的名称（例如“ main”）。

还有一个名为[PipelineCache](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkPipelineCache.html)的辅助对象，可用于加速管道创建。这是一个简单的对象，您可以在创建管线的过程中有选择地传递它，但这确实可以通过减少内存使用量和管线的编译时间来帮助提高性能。驱动程序可以在内部使用它来存储一些中间数据，从而可能更快地创建类似的管线。您还可以将PipelineCache对象的状态保存并加载到二进制数据的缓冲区中，以将其保存在磁盘上并在下次执行应用程序时使用。我们建议您使用它们！

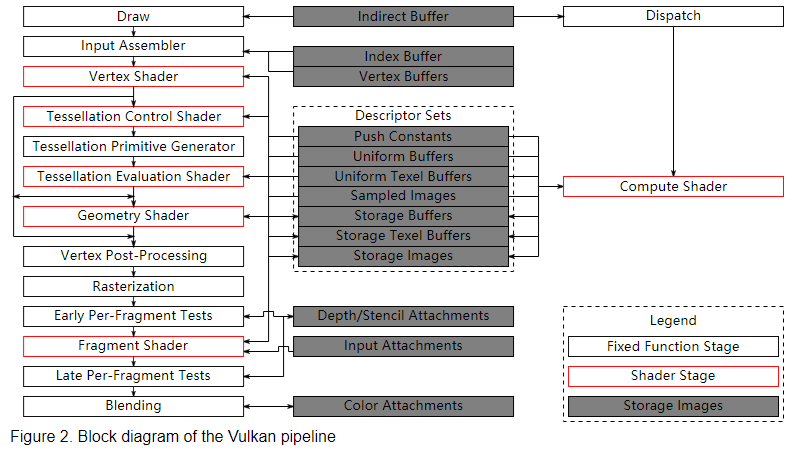
1. **Query(查询)**是Vulkan中的另一种对象。它可用于回读由GPU写入的某些数值。有多种查询，例如遮挡（告诉您是否渲染了某些像素，即它们通过了所有着色前和着色后测试并传递到帧中）或时间戳（来自某些GPU硬件计数器的时间戳值） 。Query没有自己的类型，因为它始终驻留在QueryPool中，并且仅由uint32\_t 索引表示 。可以通过指定要包含的查询的类型和数量来创建[QueryPool](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkQueryPool.html)。然后，他们可以用来发出命令到CommandBuffer一样 vkCmdBeginQuery ， vkCmdEndQuery 或 vkCmdWriteTimestamp 。

最后，还有一些用于同步的对象：[Fence](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkFence.html)，[Semaphore](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkSemaphore.html)和[Event](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkEvent.html)。

1. **Fence(栅栏)**发信号给主机任务的执行完成。可以在主机上等待，轮询和手动取消信号发送。它没有自己的命令功能，但在调用vkQueueSubmit时会被传递  。一旦提交的队列完成，就会发出相应的**Fence**信号。
2. **[Semaphore](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkSemaphore.html)([信号量](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkSemaphore.html))**创建不需要配置参数。它可用于控制跨多个队列的资源访问。它既可以作为命令缓冲区提交的一部分用信号发送或等待，也可以通过调用vkQueueSubmit ，也可以在一个队列（例如计算）上用信号发送 ，而在其他队列（例如图形）上等待。
3. **[Event](https://www.khronos.org/registry/vulkan/specs/1.0/man/html/VkEvent.html)(事件)**创建也不带参数。它可以等待上或用信号通知在GPU上作为一个单独的命令提交到CommandBuffer，使用函数 vkCmdSetEvent vkCmdResetEvent , and vkCmdWaitEvents 。也可以设置，重置和等待它（通过轮询 从一个或多个CPU线程发出的调用vkGetEventStatus ) 。 如果在GPU上的单个点发生同步，或者可以在渲染内使用subpass依赖项，vkCmdPipelineBarrier 也可以用于类似目的。

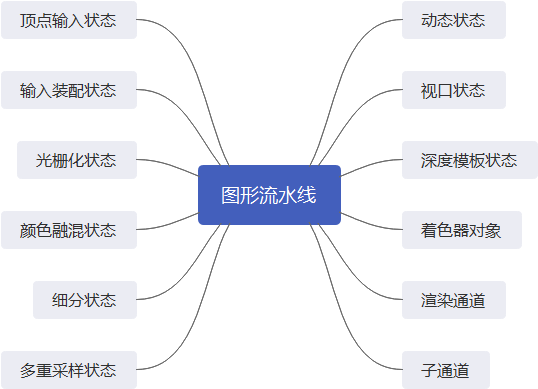
之后，您只需要学习如何调用Vulkan函数（可能并行！），即可使GPU在每一帧中执行实际工作。然后，您便可以充分利用现代GPU提供的强大，灵活的计算能力。

Vulkan图形和计算管线结构:



<https://www.khronos.org/registry/vulkan/specs/1.2/html/chap11.html#pipelines>

图形管线以及对应的接口控制器



物理设备支持的队列类型：

|  |  |  |  |
| --- | --- | --- | --- |
| 队列类型 | | | |
| 图形队列 | 计算队列 | 传输队列 | 稀疏队列 |

物理设备可能包含一个或多个队列族，每个队列族所支持的队列类型各不相同，每个队列族中可能包含一个或多个队列:

|  |  |  |
| --- | --- | --- |
| 物理设备 | | |
| 队列族属性1 | 队列族属性2 | 队列族属性3 |
| 队列族1中的所有队列 | 队列族2中的所有队列 | 队列族3中的所有队列 |

缓冲区资源创建工作流:



命令池和命令缓冲区:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 命令池 | | | | | |
| 命令缓冲区1 | 命令缓冲区2 | 命令缓冲区3 | 命令缓冲区4 | 命令缓冲区5 | 命令缓冲区N |

记录命令缓冲区过程:

|  |  |  |
| --- | --- | --- |
| 命令缓冲区 | 开始录制(vkBeginCommandBuffer) |  |
| vkCmdBeginRenderPass | Vulkan API调用 |
| vkCmdBindPipe |
| vkCmdBindDescriptionSets |
| vkCmdBindVertexBuffers |
| vkCmdSetViewport |
| vkCmdSetScissor |
| vkCmdDraw |
| vkCmdEndRenderPass |
| 结束录制(vkEndCommandBuffer) |  |

Vulkan资源类型:缓冲区和图像。

|  |  |
| --- | --- |
| 堆内存 | |
| 图像 | 缓冲区 |
| 图像视图 | 缓冲区视图 |

交换链实现的工作流：



管线布局对象的内容:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 流水线布局 | | | | | | | | | | | | | |
| 描述符集布局0 | | | 描述符集布局1 | | | 描述符集布局2 | | | | | | | |
| 缓存  0 | 缓存  1 | 缓存  2 | 纹理  0 | 缓存  0 | 采样  0 | 缓存  0 | 缓存  1 | 纹理  0 | 纹理  1 | 纹理  2 | 纹理  3 | 采样  0 | 采样  1 |

同步图元类型：

|  |  |
| --- | --- |
| 同步图元类型 | 描述 |
| 栅栏(Fence) | 提供宿主机和设备之间的同步机制 |
| 信号量(Semaphore) | 提供队列之间，以及队列内的同步机制。 |
| 事件(Event) | 队列提交时的同步 |
| 屏障(Barrier) | 指令缓存中各个指令之间的同步 |