**Base code**

创建一个类来封装程序，将Vulkan对象作为私有类成员，并且添加函数来初始化（initVulkan）。当一切准备就绪，我们进入主循环函数来渲染（mainLoop）。当窗口关闭，主循环函数返回，我们需要在cleanup函数中释放资源。

**Resource management**

就像每一块由malloc分配的内存空间需要由free来释放一样，每一个Vulkan对象也需要显示的清除。在完成本示例之后，你就可以实现C++版的自动资源管理，在构造函数中获得Vulkan对象，在析构函数中销毁。

Vulkan对象的创建可以直接由函数vkCreateXXX实现，或者是通过另一个拥有vkAllocateXXX函数的对象来创建。销毁时使用vkDestroyXXX或者vkFreeXXX。这些函数的参数根据不同的对象而产生变化，但是他们都有一个共同的参数：pAllocator。这是一个可选参数，用于指定自定义内存分配的回调函数。在示例中将忽略这个参数，始终设置为nullptr。

**Instance**

**Creating an instance**

首先需要创建一个实例来初始化Vulkan library。实例是应用程序和Vulkan library之间的桥梁，创建的时候需要告知驱动一些关于应用的具体信息。

VkInstance instance；

创建实例之前，我们需要将应用相关的信息填充到一个结构体中。这些数据其实技术上是可选的，但是它能够给驱动提供一些有用的信息，以便优化我们的应用。

在Vulkan中，有很多的结构体都需要你显示地指定类型（sType成员）。

在Vulkan中，有很多的信息都是通过结构体来传递而非函数参数，我们需要填充一个或多个结构体来提供足够多的信息创建实例。接下来的这个实例不是可选的，并且告诉Vulkan驱动我们将要使用哪种全局扩展以及确认层（validation layers）。

**Checking for extension support**

**Validation layers**

Vulkan API 的设计理念是最小化驱动开销，这个理念的一个显著特征就是API默认的错误检测会非常有限。即使是枚举值错误或者传入空指针这种简单错误，也不会被显示地处理，从而导致崩溃或者没有定义的行为。因为Vulkan需要你显示地指出你所在做的任何事情，这很容易产生小的错误。例如使用一个新的GPU特性，但是忘了在逻辑设备创建时期获取。

但是，这并不意味着我们不能添加check到API中。Vulkan介绍了一种优雅的系统称之为validation layers。Validation layers是可选的组件，与函数调用挂钩来实现更多的操作。一些常见的操作包括：

* 将参数值与规范进行比对来检测错误
* 跟踪对象的创建和销毁来查找资源泄露
* 跟踪线程来检查线程安全
* 输出每一次调用以及参数
* 跟踪Vulkan调用，用于分析和复盘

这些validation layers可以自由的设置来囊括你所感兴趣的调试功能。你可以简单地为debug构建模式开启validation layers，为release构建模式关闭。

Vulkan有两种不同的validation layers类型：实例型和指定设备型。实例型只会检测与全局Vulkan对象相关的调用，而指定设备型只会检测与指定GPU相关的调用。后者已经被弃用。

**Message callback**

**Physical devices and queue families**

在完成Vulkan library的初始化，即创建实例之后，我们需要从系统选择一个显卡。实际上，我们可以选择任意数量的显卡并且同时使用他们。