VS OpenGL

http://imgtec.eetrend.com/d6-imgtec/news/2016-01/7121.html

http://imgtec.eetrend.com/d6-imgtec/news/2016-01/7067.html

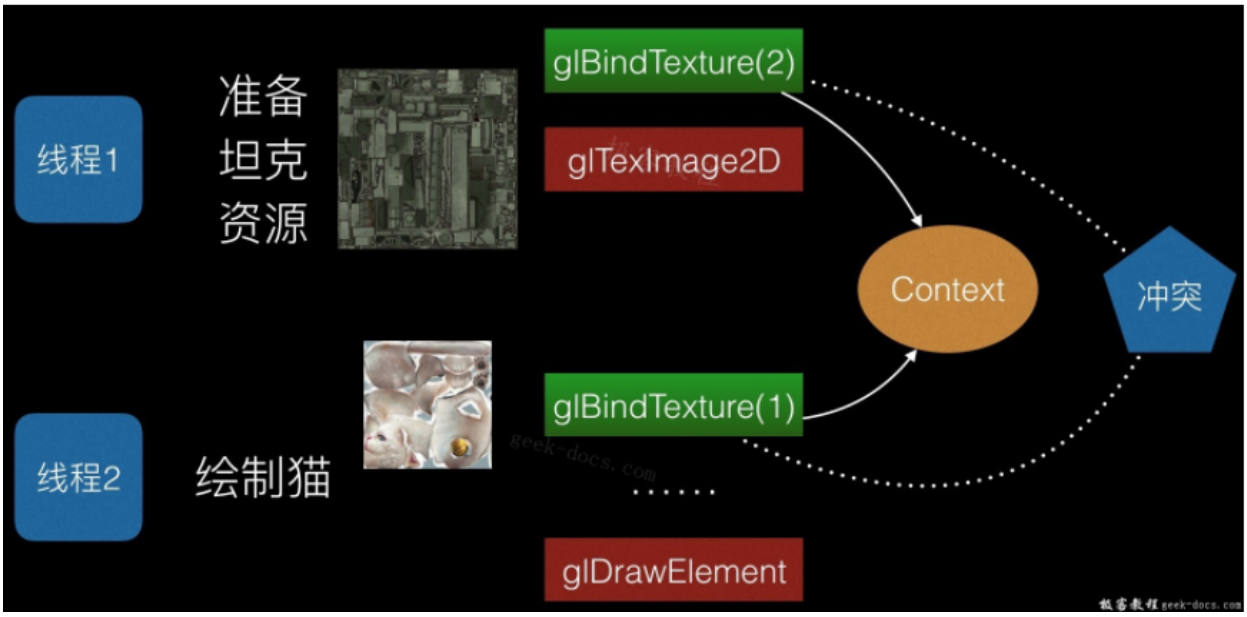
https://zhuanlan.zhihu.com/p/528801461

Vulkan的API在设计层面上就很明显有以下优势:

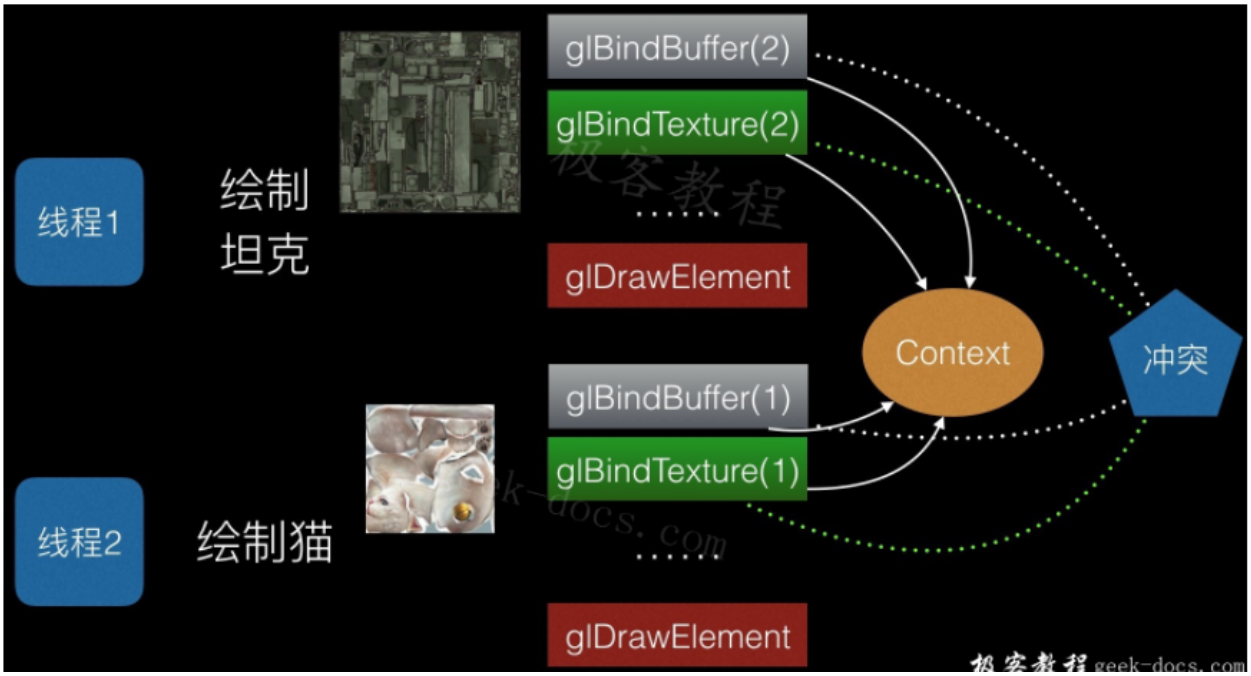
- 多线程更友好,Vulkan 相对而言不再需要依赖于绑定在某个线程上的Context,而是全新的基于Queue的方式向GPU进行递交任务,并且提供了多种Synchronization的组件让多线程编程更加亲民。
- 更强调复用,大多数Vulkan API的组件都可以高效的被复用。
- 更贴近硬件的API,让程序有更多的权限和责任自主的处理调度和优化

多线程

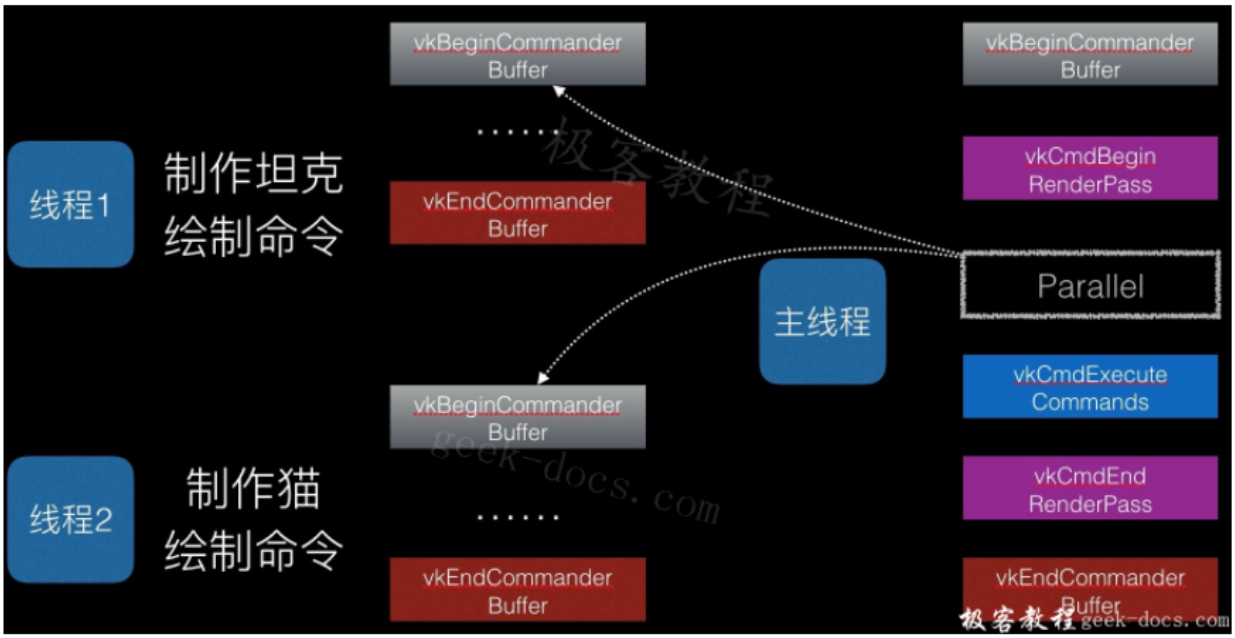
使用OpenGL时，如果把数据交互放到另一个独立线程中完成，将会引起冲突，这个原因是上传资源和进行绘制时都需要改变上下文：



由于绘制时要改变上下文，OpenGL的并行绘制无疑也不可能了：



Vulkan可以并行创建Command Buffer，Command Buffer 提交后就都是GPU驱动怎么执行的事了，执行的过程没必要也没可能用多线程加速。



复用

OpenGL 每帧绘制时，都需要在**驱动层**重新建一个Command Buffer 传递下去，而 Vulkan 是在**应用层**建好 Command Buffer，每帧绘制时Sub上去。

底层驱动

虽然初看上去Vulkan比OpenGL复杂了许多，需要多写不少代码，但真正到软件开发时，由于Vulkan、OpenGL大部分情况都是用来写引擎，中间件的，维护代码的时间会远大于开发代码的时间，多写那几行代码根本不算啥。 Vulkan更容易封装，各子模块之间互不影响，软件架构设计会轻松不少，开发维护起来更为方便。

而基于OpenGL开发的各子模块之间总有各种各样因为状态机的缘故引发的Bug，比如：

- A模块用了VBO，B模块没有用，集成在一起时由于B模块没有把 GL_ARRAY_BUFFER 重新绑定为0,出现段错误。
- A模块内部使用了FBO，使用FBO时重新设置了裁剪区域和视口大小，使用完成后没有恢复，导致集成之后，后续的模块绘制出现问题。 如果基于Vulkan开发，上述的状态不一致的问题将会少很多。

当然，基于Vulkan开发的引擎一般会用多线程加速，这个也会有不少坑，但为了更好的性能，也是值得的。

OpenGL	Vulkan
单一全局状态	基于对象，没有全局状态
状态与单一环境相联系	所有状态概念都可以本地化到命令缓冲区
操作只能顺序执行	可以进行多线程编程
GPU的内存和同步通常是被隐藏的	清晰的显存管理和同步化控制
广泛的错误检查	Vulkan驱动程序在运行时不进行错误检查；有一个针对开发人员的验证层