渲染流程

渲染必须在渲染窗口中才能进行，所以必须要和宿主操作系统中建立一个可支持 OpenGL 渲染的窗口。这里我们目前采用的是第三方库的实现方案，采用的是 SDL（Simple DirectMedia Layer），这是一个跨平台的多媒体开发库，支持 Linux、Windows、Mac OS X 等，类似 Windows 环境中的 DirectX。它提供了图形、声音和用户输入输出的良好支持，目前这个项目仅仅使用了其图形和 IO 的功能。

渲染的总控是在 Engine::Root 类中，为了提高渲染的效率，防止主线程在无谓的等待帧同步而浪费时间，我们将渲染场景分成了两个线程，两个线程的主函数都在 Engine::Root 类中，一个线程是程序的主线程，另一个线程是渲染线程。下面分别详细介绍这两个主函数：

主线程的主函数 startRendering（）做了两件事情，一个事情是执行用户设定的回调函数（通过 Root::addFrameListener 来添加回调），另一个事情就是通知程序中的每一个 SceneManager 去准备好自己的渲染队列。

渲染线程的主函数 renderThreadFunc（）也做了两件事情，首先是将显卡资源相关的资源进行初始化，包括纹理对象的初始化，缓冲区对象的初始化；其次是将主线程中准备好的渲染队列进行一次渲染。

两个线程的同步是一个单一的生产者消费者问题的模型，主线程（生产者）处理完用户回调和准备好渲染队列后通知渲染线程（消费者）去取渲染队列进行渲染，然后等待消费者直到其取出该渲染队列。渲染线程在获得渲染队列后会唤醒主线程，让主线程进行下一次循环过程。该模型的实现使用了 Linux 中的条件变量（pthread_cond_t）来进行的，Engine 中对其进行了简单的封装。

这种同步和单线程相比高效的地方在于，主线程最费时的准备渲染队列操作和渲染线程渲染整个场景的操作可以完全的并发执行。为了达到此目的，我们在每个 SceneManager 中都准备了两个渲染队列，一个用于读，一个用于写，并且每切换一帧，两者的顺序就会发生改变（即用于读的变成用于写的，用于写的变成用于读的），非常类似于 OpenGL 中的双缓冲机制。

所以整个场景关于渲染的部分的流程可以用下面的时序图-1 表示：

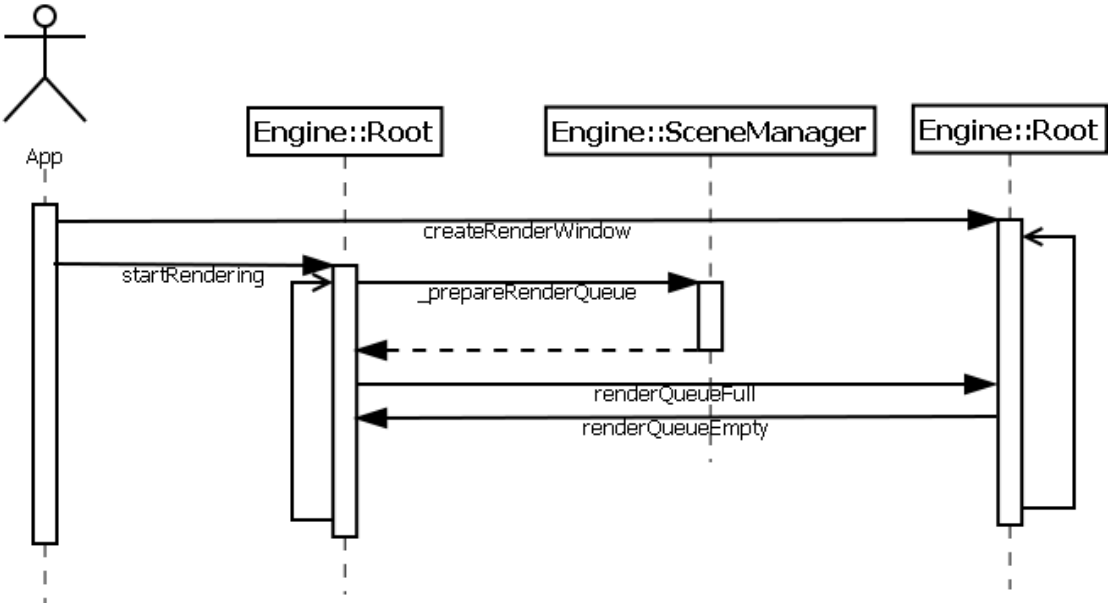

图 1 渲染流程图

所以，一个应用程序如果想使用Engine进行图形的渲染，必须要先创建一个Engine::Root对象，调用 Root::createRenderWindow 创建一个渲染窗口，然后调用 Root::startRendering 进行渲染。