跨平台视频渲染框架的构建思路和实验

目录

[思路](#_Toc1913121214_WPSOffice_Level1) [2](#_Toc1913121214_WPSOffice_Level1)

[图像更新方式](#_Toc1870230370_WPSOffice_Level1) [2](#_Toc1870230370_WPSOffice_Level1)

[二次处理：](#_Toc243687451_WPSOffice_Level1) [9](#_Toc243687451_WPSOffice_Level1)

[框架结构：](#_Toc403674128_WPSOffice_Level1) [10](#_Toc403674128_WPSOffice_Level1)

思路

Android、Windows端使用OpenGL实现跨平台，虽然OpenGL性能对比Vulkan和DX12这两个图形API，效率已经比不上了，但使用它来进行2d画面渲染、叠加、图像处理还是绰绰有余的，如果不同端使用该端专有API显得工作量较大，可以在OpenGL之上再封装一层仅为绘图和转码专用的函数，有需要时逐步替换内部内容即可。

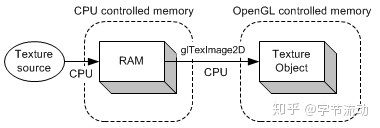
苹果系只能去摸索metal了。

综上，至少需要使用OpenGL和Metal两个图形API，这两个API使用方式还是有较大差异的，因此还是需要封装一个统一用于绘图的一系列函数。为了让多端都可以用，那只好用C/C++进行这个封装工作。但现在所有平台对OpenGL的支持依旧良好，因此先使用OpenGL实现进行跨平台渲染库，做稳定后再逐步分平台使用对应的图形API是性价比较高的做法。

图像更新方式

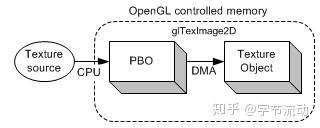
因为GLTexImage2D每次都需要把内存数据复制到显存中，消耗性能较大，所以使用PBO进行纹理更新是比较划算的。PBO只要设备支持OpenGL ES 3.0就可以了，也就是安卓4.3以上的机型（苹果系为iOS7以上，iPhone5代（5c除外）、iPad air、ipad mini2开始支持）均可使用，已经是不算太新的特性。建议使用双PBO进行纹理更新，使用shader进行YUV数据处理。

以前在上个公司做相机APP时，使用的是glTexImage2d来把yuv数据从内存靠到显存对应纹理上，流程如下：



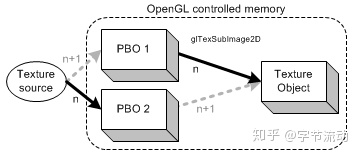
这样会导致一个问题就是这个拷贝操作是需要CPU进行操作的，如果数据量较大则一定会消耗较大的CPU时钟，高分辨率画面对低端CPU不太友好，可能会导致卡顿。

而使用PBO则是这样的：



GPU驱动可以直接调用DMA异步地从内存对应的地址中拷贝数据块到显存里，不需要CPU参与，解决CPU性能消耗，减少功耗合发热。

双PBO刷新示意图如下：



实验代码与实测结果：

API使用: 安卓GLES30的JNI接口

机型：红米 Note 9 Pro

SoC：骁龙750

测试YUV420SP相片分辨率：1024x2048

shader代码：

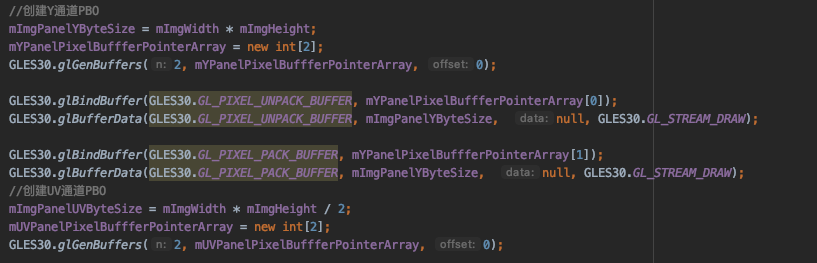


空纹理创建代码：

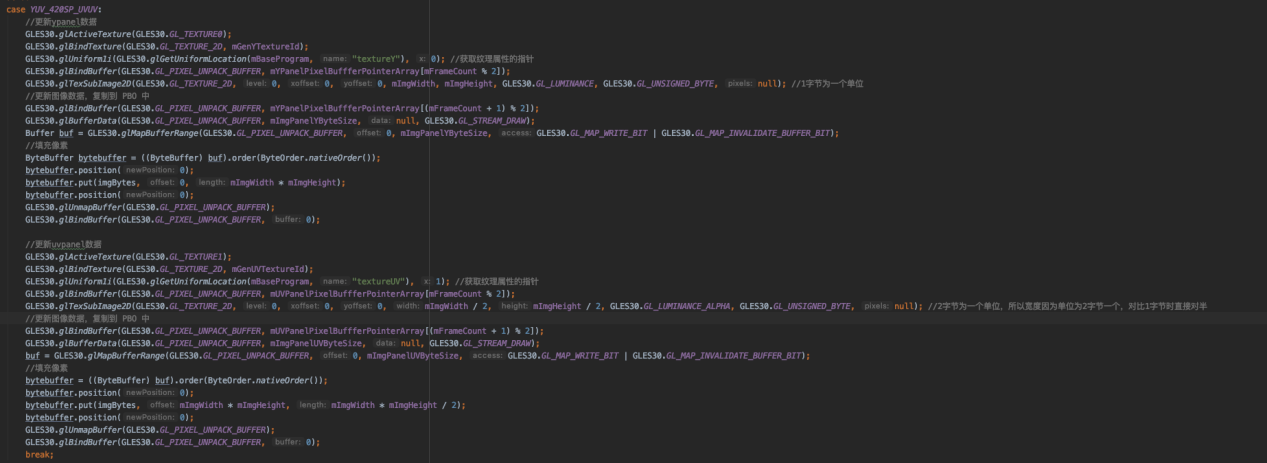
Y和UV通道对应字节格式的空纹理创建，Y通道使用1字节的LUMINANCE或者ALPHA即可，UV通道因为是两个字节为一个单位，用LUMINANCE\_ALPHA正好合适，每次纹理采样出来刚好有两个字节的结果。



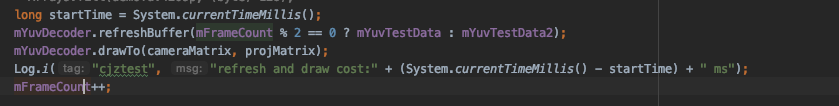
PBO创建代码：



数据刷新代码，注意这里有java字节数组转换为C Native数组的开销：



数据更新耗时计算方式：



执行结果：

YUV420SP数据顺利解码显示



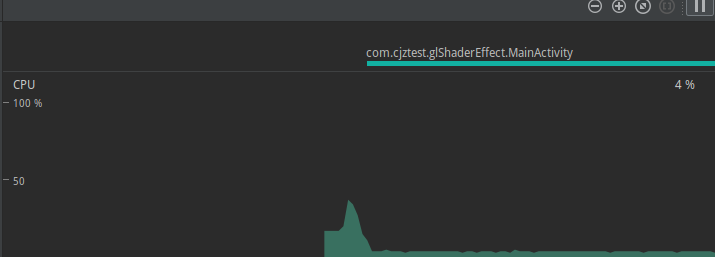
数据更新耗时：

在有Java转C数组消耗的情况下约2～5ms完成更新和draw call

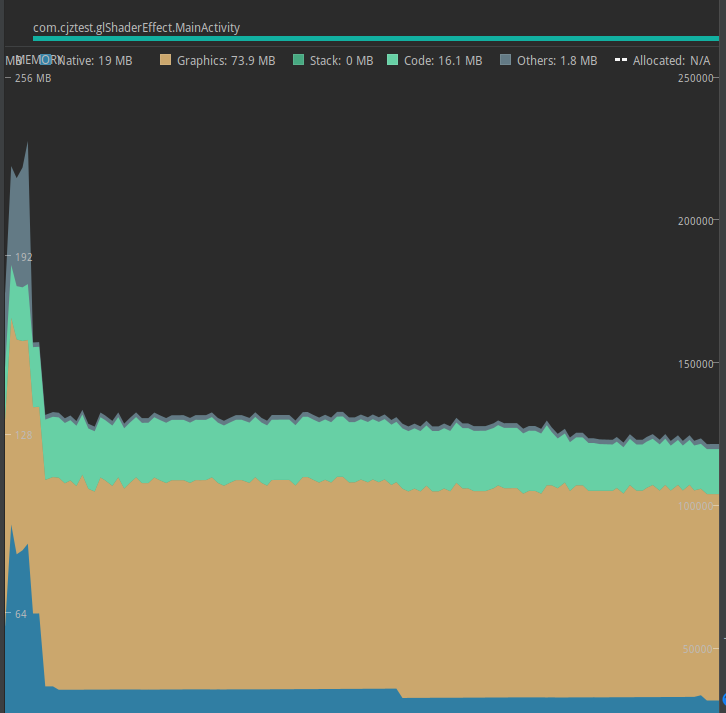


CPU占用：

很低，不到4%，而且是在有java转C数组的消耗的前提下：



内存占用：

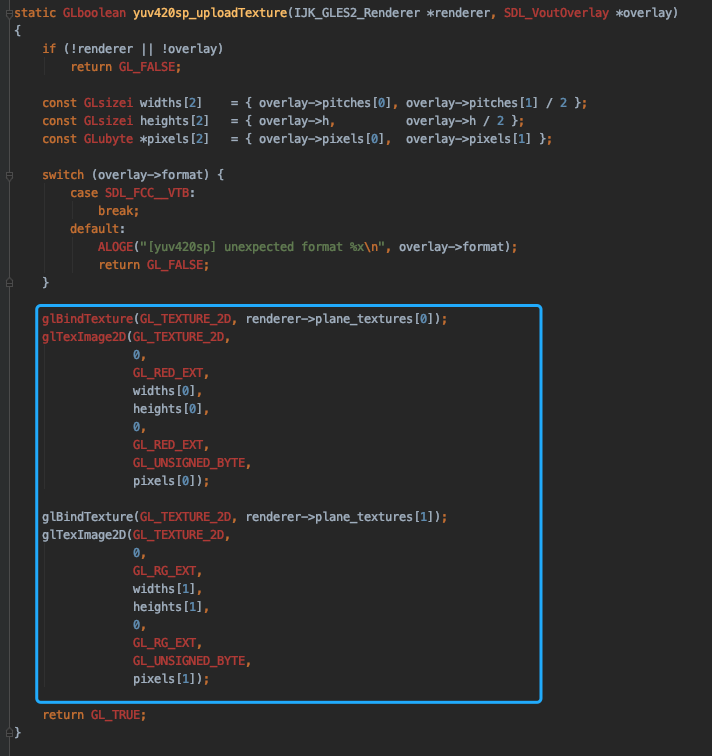


显存占用74MB左右

综上所述，使用OpenGL + 双PBO高效更新纹理 + shader转成RGB显示是非常合适于视频数据渲染的，这些API调用代码完全可以用C一模一样地实现，避免Java本身带来的消耗，估计能在本机实现1ms内的更新时间和4%以内的CPU占用率，而且OpenGL本身能实现Windows + 安卓端的跨平台，同样的代码可以在两个平台之间基本不经修改地运作，减少开发量和维护负担。另外利用FrameBuffer Object特性，可以使YUV转码后的结果作为一个Framebuffer对象，然后bindTexture到其他shader Program，实现滤镜、打水印、锐化画面等二次渲染效果，或是搭配PBO或glReadPixels实现处理结果输出到数组等操作。好处是比较多的。

至于老旧机型，需要根据GL版本或系统版本判断是否能支持PBO，如果不行的话则需要更换为glTexImage2D。

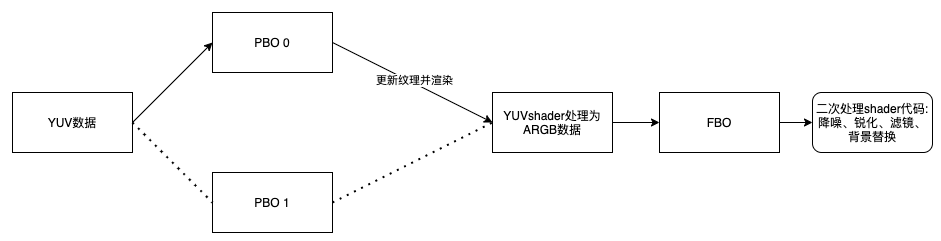
对比ijkplayer中使用的glTexImage2d



使用双PBO理论上能节约CPU时钟，效率更高。

二次处理：

流程如下：

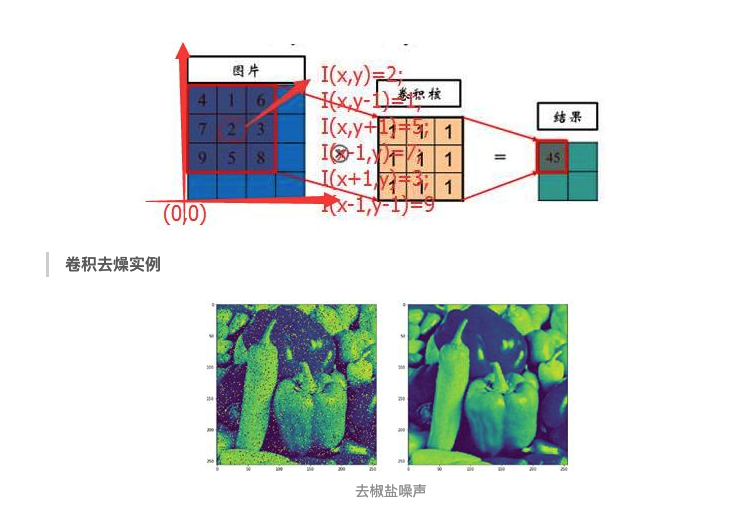


实测效果：

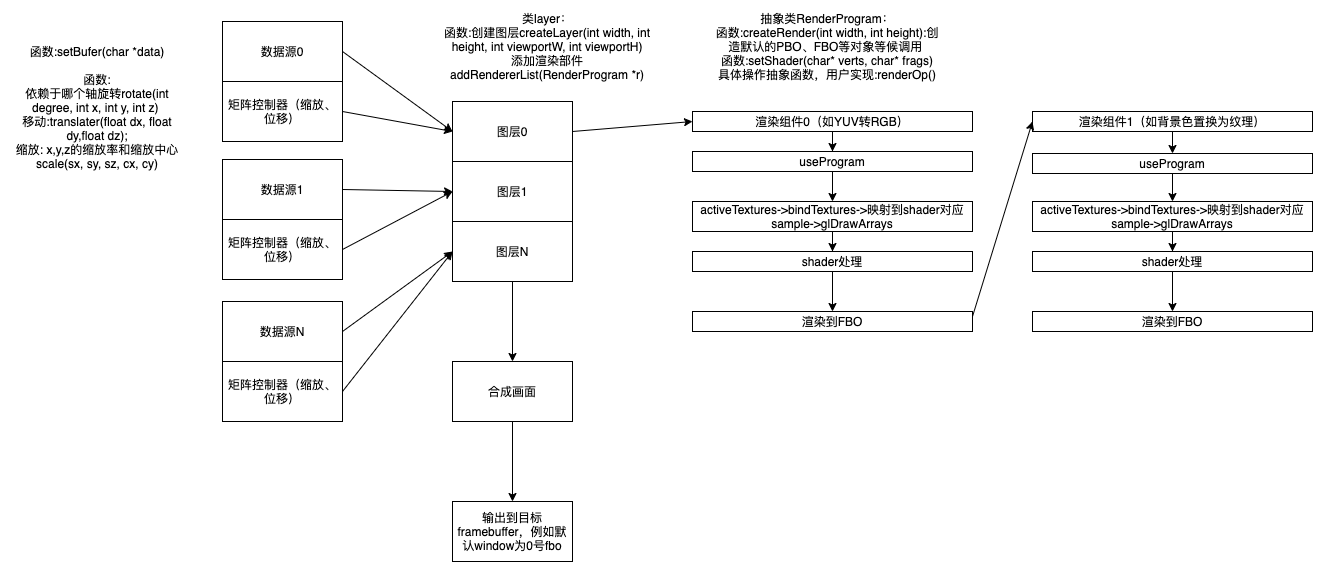
例子：YUV转换后通过shader代码，使用卷积核实时提取画面物体边缘。



二次渲染通过FBO实现之后，可以实现诸如视频降噪、背景替换等视频实时二次处理效果：



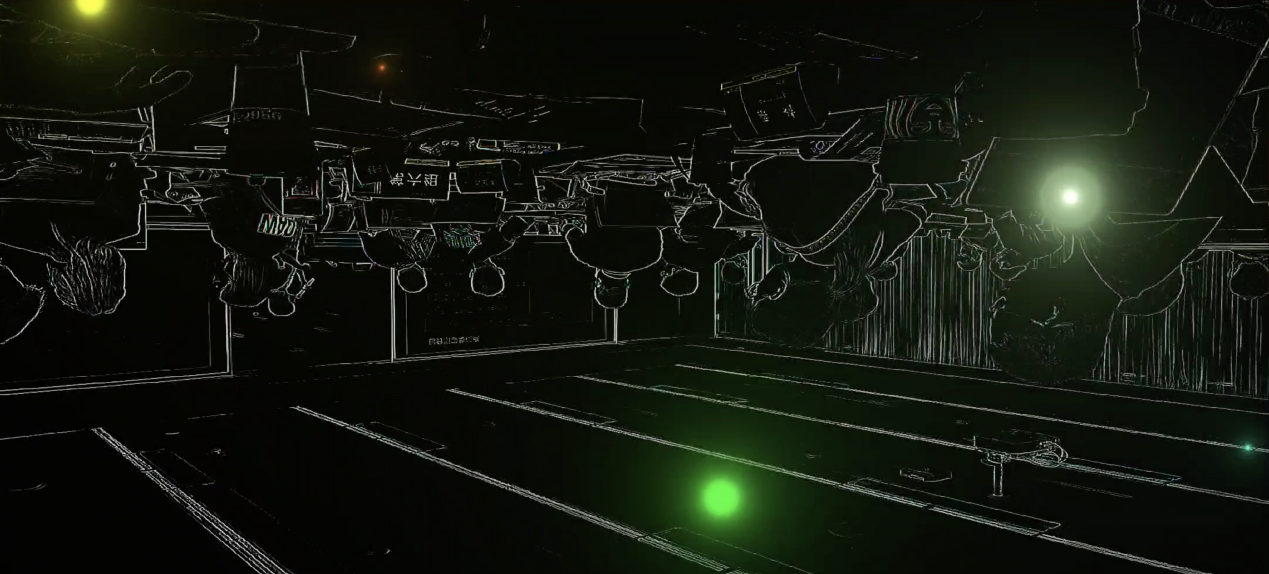
框架结构：



按照此结构把OpenGL中渲染所需要的所有步骤抽象成一系列函数，并提供一系列RenderProgram实现模板，如多种YUV数据的转换program，降噪program等，方便调用者直接套用。而且图层结构非常有利于实现画中画、多人视频、视频表面叠加气氛视频（例如放烟花）等叠加效果。每个图层可以在队列中添加多个RenderProgram使画面得到多次处理，如下面一个例子就是使用了YUV转换数据为ARGB的Program后，再使用边沿提取的Program进行二次处理后的画面，当然也可以替换或叠加成滤镜Program等。

现阶段OpenGL可以实现Mac、iOS、Android、Windows的跨端，但总有一日需要替换为对应系统的实现，例如Metal，可以在基本流程都固定下来不需作修改时，再分平台替换函数中的代码即可。

按照此结构实现的demo截图，一个图层YUV转换，转换完后在第二轮处理中实现仅显示物品边沿，并在第二个图层显示光点，最后合成画面如下：



这个方案可以灵活实现多种视频处理风格，并且实现过程不依赖于CPU，效率较高，而且可以比较方便地实现多种效果。