Ppsspp

通过NativeGLView 来实现GL绘制

设置了NativeRenderer绘制类

调用本地方法

**public** **native** **void** displayInit();

**public** **native** **void** displayRender();

关于本地方法

通过App-android.cpp实现java到本地的调用

通过NativeApp.cpp 实现本地方法

首先看

displayInit() 对应到本地方法

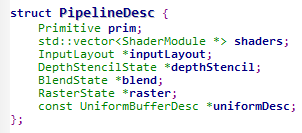


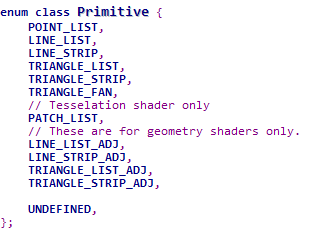
跳转到分支 NativeInitGraphics();



这里标注的方法和结构体初始化调用是对渲染管线的一系列初始化设置，看到

#include "thin3d/thin3d.h"





这里面主要是shader 及其枚举常量的初始化

colorPipeline = g\_draw->CreateGraphicsPipeline(colorDesc);

texColorPipeline = g\_draw->CreateGraphicsPipeline(texColorDesc);

这里设置了两条渲染管线，一条用来做普通绘制，另一条用来绘制文字。

下面注入SM中，后续的每帧绘制会用来渲染。

screenManager->setUIContext(uiContext);

screenManager->setDrawContext(g\_draw);

screenManager->setPostRenderCallback(&RenderOverlays, nullptr);

接下来先看displayRender函数



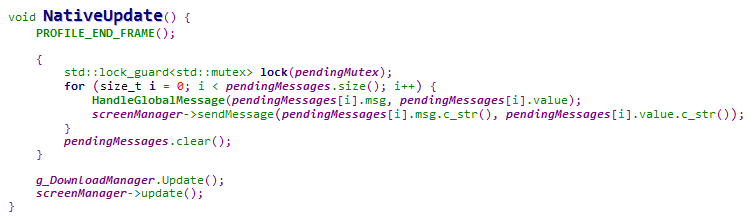
NativeUpdate();

NativeRender(graphicsContext);

time\_update();

关注三个调用

NativeUpdate();



这里有个重要的函数

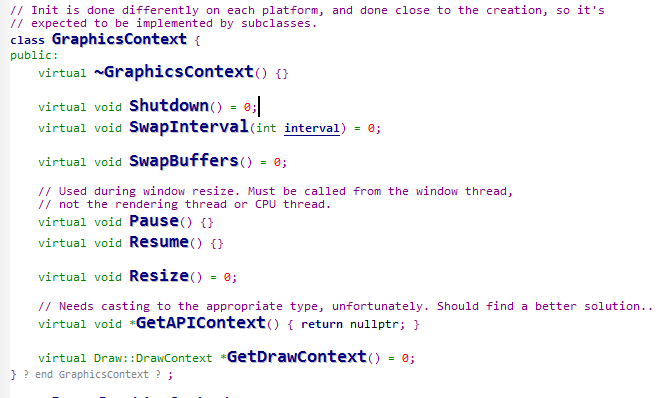
HandleGlobalMessage和类 screenManager 

猜测这里的HandleGlobalMessage只是用来为后续screenManager的消息机制作准备，比如清除背景，下面我们看screenManager，回到Java\_org\_ppsspp\_ppsspp\_NativeRenderer\_displayInit探索screenManager的机制。



先看到分支NativeInitGraphics 这里使用了前面生成的对象AndroidJavaEGLGraphicsContext：





因为父类提供了几个纯虚方法规范，所以回到AndroidJavaEGLGraphicsContext的实现上；

这里主要关注draw\_成员

Draw::T3DCreateGLContext();

看到T3DCreateGLContext();方法



这里返回的实际上是一个OpenGLContext对象，这里实现了opengl的功能封装，等待分析具体绘制命令我们再回来看这个类的具体实现；

总结下目前从displayInit 调用到NativeInitGraphics(graphicsContext);分支我们首次初始化了一个graphicsContext，里面包含了opengl层调用绘制glXXX方法的实现；

接下来看NativeInitGraphics(graphicsContext);