# Android graphic简介

### Graphic的功能:

Android graphic系统是用于将各个应用以可见形式的图形展示给用户，用户可以通过点击相应图形来进行交互；所以图形系统应该包括两部分：图形展示和图形交互；

下面的内容主要包括图形展示部分，如何讲应用从简单的布局文件转化为可见图形；

应用的界面主要包含三类图形数据：

1.android封装好的一些图形控件：button，textview等，这一部分是需要使用opengl来绘制出来的；

2.图片:背景图片，这一部分只需要读取文件放到指定的缓冲区就好，不需要绘制

3.摄像头或者网络发送过来的图形数据，这一部分是可以直接显示的，或者经过多媒体转换之后可以显示的，也不需要重绘；

下面主要是和第一种图形控件相关的，从布局文件转化为可见图形可以划分下面几个阶段:

### 1.解析xml布局文件

解决的问题：怎么将xml格式的布局解析、主要属性解决

链接:http://blog.csdn.net/yanbober/article/details/45970721

在布局文件中已经包含了各个Layout和控件之间的关系，这一过程就是将xml转化为下图的结构：

在这里对每一个Activity都抽象出了上面的关系：一个ViewGroup来管理所有的view，view具有宽、高属性；更加具体的细节需要继续阅读；

首先要理解Android中类抽象的目标是什么？

Window是一块电子屏，PhoneWindow是一块手机电子屏，DecorView就是电子屏要显示的内容，Activity就是手机电子屏安装位置。

从layout到真实大小:

解决的问题：根据xml文件适配屏幕大小

画图：

何时开始画图：

画图过程：

画完之后：

## 2.Measure

上一个过程的到了一个View的视图，但是不同的显示设备大小不同，需要根据物理设备的大小进行调整；

先确定最上层的view的大小，然后确定对子view的约束；递归检查子view是否满足父view的约束；

先得到一个最顶层的DecorView的大小，然后进一步去测量内部view的大小；

### 3.Layout

已经得到了各个view的大小，下一步就是绘图；绘图有4大基本元素:

A Bitmap to hold the pixels, a Canvas to host the draw calls (writing into the bitmap), a drawing primitive (e.g. Rect,Path, text, Bitmap), and a paint (to describe the colors and styles for the drawing).

我们需要一个bitmap，通俗点讲就是内存；

这个过程最主要的是通过WM向SurfaceFlinger申请到了画图的内存

[SurfaceControl](http://localhost:8080/source/s?defs=SurfaceControl&project=multi-l) [surfaceControl](http://localhost:8080/source/s?defs=surfaceControl&project=multi-l) = [winAnimator](http://localhost:8080/source/s?defs=winAnimator&project=multi-l).[createSurfaceLocked](http://localhost:8080/source/s?defs=createSurfaceLocked&project=multi-l)();

SurfaceControl是surface的控制单元，可以通过surfaceControl来获取和设置surface。

### 4.Draw

终于开始进行实际的绘图了，过程分为3步：

1.获取一个进程可读的地址

2.调用OPENGL绘图

3.将绘制完毕的图形数据发送给surfaceflinger合成

合成

一个android上可以运行多个应用，每个应用都不知道其他应用绘图的位置，所以需要surfaceflinger进行统一管理；

分为如下几步:

1. 计算可见区域是否变化

运行多个应用时，在surfaceflinger侧会有对应的layer来管理属性数据；

如果可见区域没有任何变化，只是一些被遮挡的数据变化但是不会被用户看到，那么就没有重新合成的必要了；

计算可见区域就是从上到下计算可见区域的数据是否变化

2.如果可见区域变化了，那么需要重新合成图层；此时有两个选择：GLES或者HWC；在multiwindow上只有GLES可用，所以GLES完成合成的所有工作

3.合成完毕后会通知进行drm\_gralloc显示

### 画图过程函数调用:

获取surface：ViewRootImpl:draw-->ViewRootImpl:drawSoftware

Surface:lockCanvas->android\_view\_Surface.cpp:nativeLockCanvas-->Surface.cpp:lock-->Surface.cpp:dequeueBuffer-->mGraphicBufferProducer->dequeueBuffer

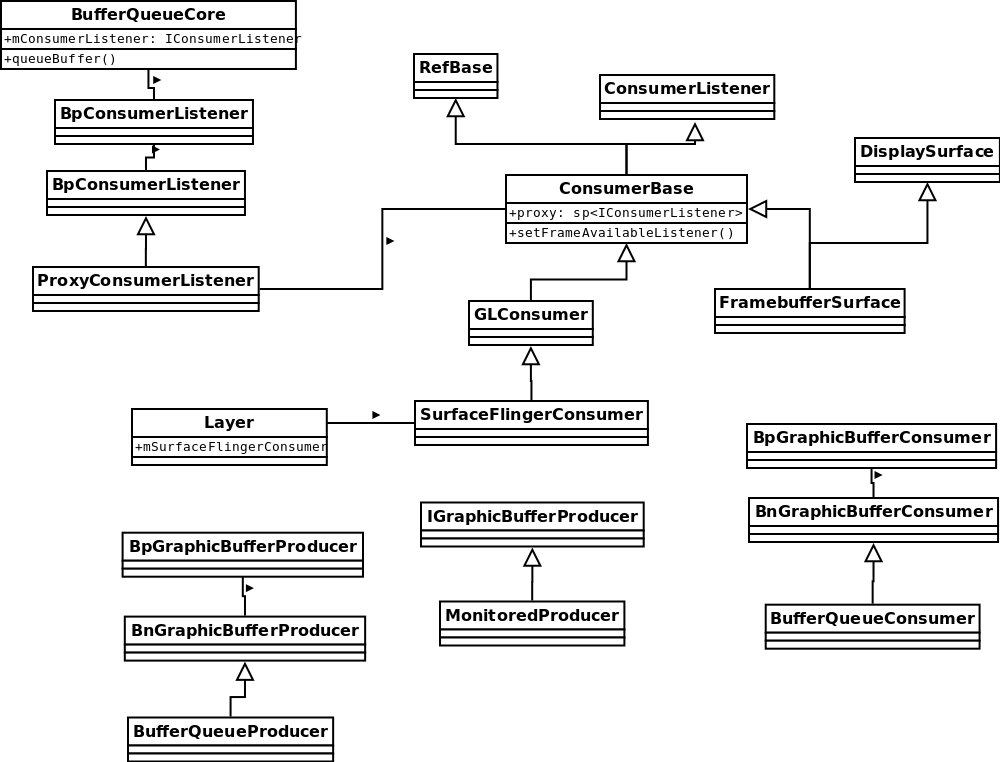
绘图：View:draw-->Canvas:drawXXX-->HardwareCanvas:-->GLES20Canvas:drawXXX-->android\_view\_GLES20Canvas.cpp::android\_view\_GLES20Canvas\_drawXXX-->存入DisplayListRenderer

释放surface,并通知surfaceflinger：Surface:unlockCanvasAndPost->android\_view\_Surface.cpp:nativeUnlockCanvasAndPost-->Surface::unlock-->Surface::queueBuffer-->mGraphicBufferProducer->queueBuffer

### 合成过程函数调用:

暂无

### BufferQueue简读:



分为3部分:

Producer:生产者，主要是产生数据，在这里指两部分：APP绘制完的图形和surfaceflinger合成完的图形，前者需要surfaceflinger进行合成，后者需要surfaceflinger进行显示

Consumer:消费者，处理数据，也分为两部分:Surfaceflinger处理APP绘制完的图形和显示器显示surfacefligner合成完的图形

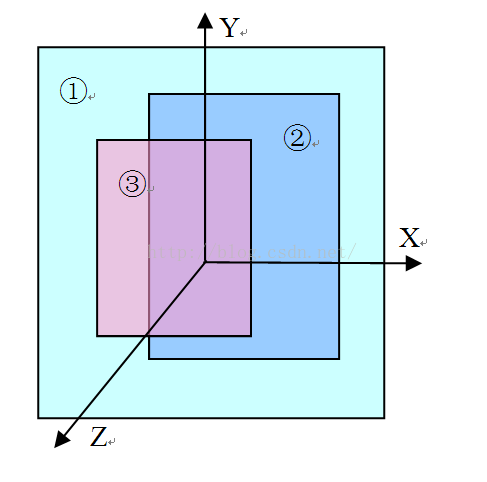
Core:这一部分主要是维持bufferqueue，有下面几个特性:

当现有的buffer满足不了producer的要求时，可能是请求者太多或者请求的size变化，Core部分会重新申请buffer

当buffer不在使用，根据引用关系逐步释放buffer

当生产者将buffer放回queue后，通知consumer进行处理

### 可见区域计算:



<http://blog.csdn.net/innost/article/details/47208337>

·  屏幕位于一个三维坐标系中，其中Z轴从屏幕内指向屏幕外。

·  编号为①②③的矩形块叫显示层（Layer）。每一层有自己的属性，例如颜色、透明度、所处屏幕的位置、宽、高等。除了属性之外，每一层还有自己对应的显示内容，也就是需要显示的图像。

在Android中，Surface系统工作时，会由SurfaceFlinger对这些按照Z轴排好序的显示层进行图像混合，混合后的图像就是在屏幕上看到的画面了。这种按Z轴排序的方式符合我们在日常生活中的体验，例如前面的物体会遮挡住后面的物体。

### OPENGL相关

OPENGL是一个跨平台的接口规范，又有各种实现的版本，最经典的是wayland和mesa；

这里以mesa为例：具体的客户端是android，具体的显卡是i915系列；

mesa是需要和具体显卡硬件设备打交道的，所以需要具体的硬件驱动，i915是libdrm\_intel;

Android虽然是一个操作系统，但是在这里扮演的是一个mesa的客户端，和android结合到一起的时候需要讲android的操作转化为本地类型处理；简单来说就是定义一系列函数接口，但是具体的实现由android自己注册；

中间还有一个比较重要的EGL:位于mesa和android之间，android->EGL->mesa->具体设备/android回调函数，主要功能:进行graphic上下文管理，surface 申请/销毁，同步机制等；最明显的是向android提供调用mesa的API

渲染引擎:

暂无