Android 图形系统的分析与移植 *

叶炳发，孟小华
（暨南大学计算机系 广州 510632）

摘 要 本文对 Android 的图形系统底层实现技术进行研究，介绍了 Android 图形系统的组成。结合 Surface Manager 的工作原理分析了 Android 图形显示的实现，在进程通信原理的基础上分析了 Surface Manager 的实现过程，比较了在 Surface 和 OpenGL 中采用的双缓冲技术，描述了帧缓冲驱动移植的过程。

关键词 Android；图形系统；surface；帧缓冲

1 引言

Android 作为第一个完整、开放、免费的手机平台，自推出以来就是业界的热门话题，由于拥有良好的可移植性和强大的功能，在嵌入式设备方面的应用表现出良好的势头。在图形显示方面，Android 建立在 Linux 上，但并没有像一些桌面 Linux 使用 GTK（GIMP Toolkit）组建 XWindows（一个多平台的图形用户接口），也没有使用 Cairo 向量图形链接库实现图形显示，而是使用了专为 Android 而改良的一种 2D 向量图形处理函数库 Skia。在 3D 图形方面是基于嵌入式 3D 图形算法标准 OpenGL/ES 实现的，该库可以使用硬件加速。然而，虽然 Google 开放了 Android 的源代码，但相关技术文档很少，而且图形系统的实现原理比较复杂，所以本文集中对 Android 图形系统的底层原理进行研究。

* 广东省科技计划项目（No.2007B020715001）

2 Android 图形显示原理

2.1 图形系统组成

Android SDK 的图形包主要包括 android.graphics、android.view、android.widget 和 android.opengl，前 3 个是用于 2D 的图形开发，基于 SGL（Skia graphics library）。android.opengl 是用于 3D 的图形开发，基于 OpenGL/ES。

Skia 是一个开放源码的 2D 向量图形处理函数库，包含字型、坐标转换以及点阵图，有着高效能且简洁的表现，在 Android 平台中搭配 OpenGL/ES 与特定的硬件特征强化了显示效果。OpenGL/ES 是 OpenGL 的一个子集，是一个跨平台图形库，是专门为嵌入式系统而设计的。

Android 图形系统的组成如图 1 所示。上层应用调用 2D 和 3D 图形库对 Surface Manager 提供的 Surface 进行绘制，通过 Surface Manager 的合成器 SurfaceFlinger 对各个 Surface 进行合成，并由 EGL 接口实现在 Framebuffer 设备上的显示。

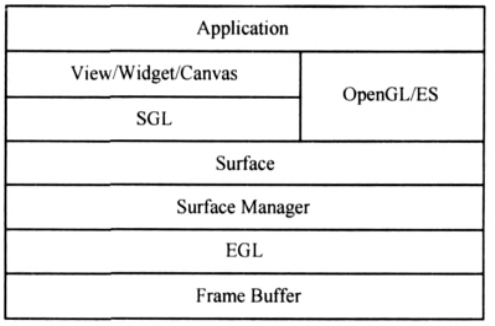

图 1 Android 图形系统的组成

2.2 Surface Manager 的工作原理

在 Android 的图形系统中，Surface Manager 是一个重要组成，Surface Manager 对上层提供 Surface 给应用进行绘制，管理对显示子系统的访问，对来自多个应用的 2D 和 3D 图像进行无缝的合成后传给底层的 EGL 进行处理，Surface Manager 的工作原理如图 2 所示。

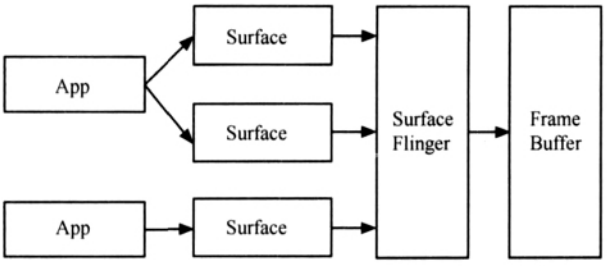

图 2 Surface Manager 的工作原理

Surface Manager 为 Application 准备一个或多个 Surface 后，把 Surface 传给 Application，让 Application 可以在上面作图形处理。应用程序先通过调用图形库提供基础的绘制图形原语和 JNI 函数，然后又通过 Native Method 的绘制图形原语调用 2D 和 3D 的图形库对 Surface 进行绘制。

在 Android 平台下，每个 Surface 都有一个 Front Buffer 和一个 Back Buffer，每个窗口都以一个 Surface 对象作为基础。每个 Surface 又对应一个 Layer，SurfaceFlinger 将各个 Layer 的 Front Buffer 合成后绘制到 Frame Buffer 上。

关于 SurfaceFlinger，在 Surface Manager 中用于管理逻辑上众多的 Surface，其功能特点如下：

- SurfaceFlinger 在一个系统范围内合成 Surface 的功能，并把合成后的显示内容传给帧缓冲设备；
- SurfaceFlinger 能一起合成来自多个程序的 2D 或 3D 显示的 Surface；
- Surface 通过 Android 的 IPC 机制 Binder 以缓冲的形式进行递交。

2.3 Surface Manager 的实现

从 Surface Manager 工作原理分析，可以理解 Application 与 Surface Manager 是以 C/S 的模式进行交互的，Application 处理 Surface 的部分是客户端，而 Surface Manager 提供服务，它们之间通过 Android 的 IPC 机制 Binder 来协助完成，如图 3 所示。

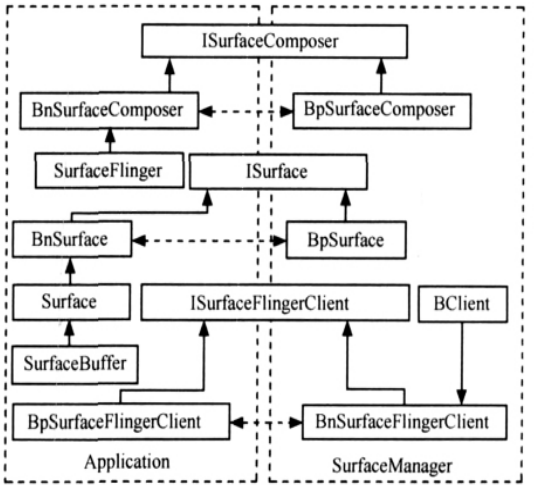

图 3 处理 Surface 的进程通信

在 Binder 中，主要包括两个方面：本地（native），如 BnSurfaceFlingerClient，这是一个需要被继承和实现的类；代理（proxy），如 BpSurfaceFlingerClient，这是一个在接口框架中被实现，但是在接口中没有体现的类。在客户端中，BpSurfaceFlingerClient 被调用，通过与 BnSurfaceFlingerClient 通信，而 BpSurfaceFlingerClient 和 BnSurfaceFlingerClient 派生自 ISurfaceFlingerClient，BClient 派生自 BnSurfaceFlingerClient。

在客户端中通过类 SurfaceComposerClient，调用 BnSurfaceComposer 和 BnSurface 来响应服务端 BpSurfaceComposer 和 BpSurface，并通过调用 BpSurfaceFlingerClient 来调用服务端的 BnSurfaceFlingerClient，由此完成了客户端和服务端的交互。在交互中的 3 个接口分别介绍如下。

ISurfaceFlingerClient：派生出 BpSurfaceFlingerClient 和 BnSurfaceFlingerClient，由 BClient 实现，通过调用 createSurface 函数创建一个 Surface 供 Application 应用。

ISurface：派生出 BpSurface 和 BnSurface，主要完成对 Surface 的处理，在 Surface（派生自 BnSurface）的函数 lock、unlockAndPost 等，实现了 Surface 在双缓冲的处理，SurfaceBuffer（派生自 Surface）实现了 Layer 上的处理。

ISurfaceComposer：派生出 BpSurfaceComposer 和 BnSurfaceComposer，SurfaceFlinger（派生自 BnSurfaceComposer）主要为合成器 SurfaceFlinger（Surface Manager 的组成部分）对 Surface 合成的相关处理提供实现方法。

3 Android 的双缓冲技术

在 Android 平台中，双缓冲技术分别在 Surface 的处理和底层 Framebuffer 的处理中使用到，在对 Framebuffer 处理的双缓冲技术根据 OpenGL 的标准实现，而对 Surface 处理的双缓冲技术则有所不同，下面将对两种双缓冲技术进行比较。

3.1 OpenGL 中的双缓冲技术

在 OpenGL 中利用双缓冲技术，分配两个帧缓冲区，在连续显示三维曲面时，一个帧缓冲区中的数据执行绘制曲面命令的同时，另一个帧缓冲区中的数据进行图形显示。当前可见视频缓冲称为前台视频缓冲，不可见的、正在绘图的视频缓冲称为后台视频缓冲。当后台视频帧缓冲中的数据要求显示时，OpenGL 就将它拷贝到前台视频帧缓冲，显示硬件不断地读可见视频缓冲中的内容，并把结果显示在屏幕上。应用双缓冲，每一帧三维曲面只在绘制完成后才显示出来，所以观察者可以看到每一帧完整三维曲面，而不是曲面的绘制过程。

3.2 Surface 中的双缓冲技术

每个 Surface 中都带有两个帧缓冲区，分别称为 Front Buffer 和 Back Buffer，与 OpenGL 中双缓冲技术有所不同，当绘制 Back Buffer 后需要显示时，并没有将 Back Buffer 中的数据拷贝到 Front Buffer 中，而是直接显示 Back Buffer 中的数据到屏幕中，从而最大限度地减少了数据的复制。

图 4 是使用 Canvas 绘制 Surface 的原理，具体过程如下：

- 创建一个 bitmap 与 Canvas 关联起来；
- 把准备显示的图形提交到 Canvas 上；
- lockCanvas，锁定 Canvas；
- drawCanvas，把 bitmap 中的数据写入到 Back Buffer 中；
- unlockCanvasAndPost，解锁 Canvas，Back Buffer 替换 Front Buffer，替换后 Back Buffer 作为前台缓冲，Front Buffer 作为后台缓冲。

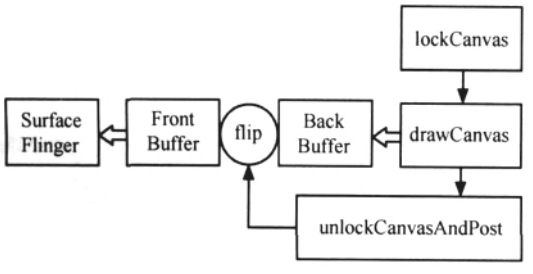

图 4 Canvas 绘制 Surface 原理

4 底层接口与驱动移植

4.1 Android 中的 EGL 接口

OpenGL/ES 为附加功能和可能的平台特性开发了扩展机制，但仍然需要一个可以让 OpenGL/ES 和本地视窗系统交互且平台无关的层。EGL 是 OpenGL/ES 和底层 Native 平台视窗系统之间的接口，是为 OpenGL/ES 提供平台独立性而设计。OpenGL/ES 本质上是一个图形渲染管线的状态机，而 EGL 则是用于监控这些状态以及维护 Frame Buffer 和其他渲染 Surface 的外部层。

在 Android 的底层源代码中，egl_native_window_t 是一个提供了对本地窗口的所有定义以及用于 EGL 操作本地窗口的所有方法的类。EGLNativeSurface 派生自 egl_native_window_t，EGLDisplaySurface 派生自 EGLNativeSurface。EGLDisplaySurface 通过函数 mapFrameBuffer 打开 Framebuffer 设备，并创建两个缓冲区，函数 swapBuffer 把后台视频缓冲区复制到前台视频缓冲区。DisplayHardware 类中初始化了 EGL，SurfaceFlinger 使用了 DisplayHardware 去和本地窗口打交道。

4.2 Android 帧缓冲驱动的移植

Android 是基于 Linux 的，但在 Linux 的帧缓冲驱动中并没有直接支持双缓冲，修改驱动包括两个方面：一是划分两个缓冲区；二是添加缓冲区的切换功能。本文是在 PXA270 上进行研究的，对应的驱动文件是/drivers/video/pxafb.c。

在 Android 中，double buffer 设计成上下两个 buffer 的模式，在函数 pxafb_setmode ()“var->yres_virtual= var->yres”修改为“var->yres_virtual= var->yres * 2”，在检查参数的函数 pxafb_check_var()中“var->yres_virtual=max(var->yres_virtual, var->yres)”修改为“var->yres_virtual =max(var->yres_virtual, var->yres * 2)”。相对应的缓冲长度也要修改为默认的两边，即在函数 pxafb_decode_mode_info()中添加“smemlen *= 2;”。

当一个缓冲区已写好，在切换时就需要调用到 pan

函数，该函数将一个新的 yoffset 传给 LCD 控制器。在结构体 fb_ops pxafb_ops 中添加 pan 函数，即插入“.fb_pan_display = pxafb_pan_display,” 到 fb_ops pxafb_ops 中。在初始化帧缓冲的函数 pxafb_init_fbinfo()中，将“fbi->fb.fix.ypanstep= 0;”修改为“fbi->fb.fix.ypanstep= 1;”，这里是说明驱动需要用到 pan 函数。以下是需要添加或修改的函数。

```
static int pxafb_pan_display (struct fb_var_screeninfo *var, struct fb_info *info){
    struct pxafb_info *fbi = (struct pxafb_info *)info;  //下一个显示的 fb 的 yoffset
     fbi->fb.var.yoffset = var->yoffset;  //将 fb 的信息和控制状态插入 fb 的工作队列
    pxafb_schedule_work(fbi,C_CHANGE_DMA_BASE);
     return 0;
}
static void set_ctrlr_state(struct pxafb_info *fbi, u_int state){
    …
    case C_CHANGE_DMA_BASE: //将 fb 传给 LCD 的 DMA
     fbi->dma_buff->dma_desc[DMA_UPPER].fsadr = fbi->screen_dma +(fbi->fb.var.xres*fbi->fb.var.yoffset*fbi->fb.var.bits_per_pixel/8);
     break;
     …
}
```

5 结束语

对 Android 的应用开发来说，图形开发是其中一个主要工作，了解 Android 图形系统的工作原理可以对应用程序性能上的提供有所帮助。在 Android 移植到其他嵌入式设备中，Android 图形系统的底层驱动移植是其中一个关键部分，通过对底层图形接口以及对帧缓冲驱动移植的研究，将更有效地实现 Android 在其他嵌入式设备上的移植。

参考文献

1 宋宝华.Linux 设备驱动开发详解.北京:人民邮电出版社,2008

2 姚昱旻,刘卫国.Android 的架构与应用开发研究.计算机系统应用,2008(11)

3 Patrick Brady. Anatomy & Physiology of an Android. http://sites.google.com/site/io/anatomy--physiology-of-an-android

4 David Blythe, Affie Munshi. OpenGL ES 1.0.02 Specification. http://www.khronos.org/registry/gles/specs/1.0/opengles_spec_1_0.pdf

Analysis and Transplantation of Android Graphic System

Ye Bingfa, Meng Xiaohua

(Department of Computer Science, Jinan University, Guangzhou 510632,China)

Abstract Researches on the underlying implement technology of Android graphic system, and introduces the composition of Android graphic system. Analyses the implement of Android graphics display with the work principle of surface manager, and the analysis of surface manager implementing is based on the process communication. Compares double buffer technologies using in surface and openGL, and describes the implementation of frame buffer driver.

Key words Android, graphic system, surface, framebuffer

(收稿日期:2009-12-12)