# 基于 Qt/Embedded 的图形硬加速方法研究与实现

王　凯,宁　钰,周　威
(江南计算技术研究所第三处,江苏 无锡 214083)

**摘　要**:为了提高嵌入式 GUI 系统的性能,充分利用嵌入式设备的硬件设备,对嵌入式 GUI 系统提供图形硬件加速就变得十分必要。通过对 Qt/Embedded 库的软件体系结构和 Qt/Embedded 库图形引擎架构的分析,得出对 Qt/Embedded 库进行底层图形硬件加速的途径,利用底层帧缓冲系统的支持和接口,提出一种自下而上的嵌入式 Linux 系统的图形硬件加速架构。通过 Qt/Embedded 嵌入式图形支持,Qt/Embedded 库可以通过 Linux 的 VFS 文件系统访问底层帧缓冲系统提供的功能和接口,实现帧缓冲系统硬件加速功能在 Qt/Embedded 库中的运行。对于如何通过 GPU 的加速器实现具体的硬件加速功能,给出了加速功能的具体实现算法和逻辑流程。最后,在测试中比较了加速前后矩形填充操作所花费的时间,结果表明加速后同等条件下矩形填充花费时间明显减少,加速比稳定在 2 以上,达到了较好的加速效果。

**关键词**:图形硬件加速;Qt/Embedded;嵌入式系统;图形用户界面
**中图分类号**:TP311　　**文献标识码**:A　　**文章编号**:1673-629X(2018)06-0067-06
doi:10.3969/j.issn.1673-629X.2018.06.015

# Research and Implementation of Embedded GUI Display Architecture with Qt/Embedded

WANG Kai,NING Yu,ZHOU Wei
(Third Place of Jiangnan Institute of Computing Technology,Wuxi 214083,China)

**Abstract**:In order to improve the performance of embedded GUI system and make full use of the embedded system hardware,it is necessary to provide hardware accelerated graphics for the embedded GUI system. Through the analysis of software architecture and graphics engine architecture of Qt/Embedded library,the way of underlying graphics hardware acceleration of Qt/Embedded library is obtained. By using the support and interface of framebuffer system,we put forward a bottom-up graphics hardware acceleration architecture of embedded Linux system. Through Qt/Embedded embedded graphics support,the Qt/Embedded library can access the functions and interfaces of the underlying framebuffer system through the VFS Linux file system implementing that the framebuffer system hardware acceleration function running in the Qt/Embedded Library. For how to achieve the specific hardware acceleration function through the GPU accelerator,the concrete implementation of algorithm and logic process of acceleration function are given. Finally,the time of the rectangle filling operation is compared in the test before and after acceleration,which shows that rectangle filling time is significantly reduced under the same conditions,and the acceleration rate is more than 2,with the better effect of acceleration.

**Key words**:graphics hardware acceleration;Qt/Embedded;embedded system;GUI

## 0　引　言

嵌入式系统是一种“完全嵌入受控器件内部,为特定应用而设计的专用计算机系统”,在电子产品集成度越来越高的今天,广泛应用于工业生产和生活的各个方面。全球生产的 CPU(中央处理器),超过 80% 应用于各类嵌入式系统。随着嵌入式系统的日益发展,嵌入式处理器运算能力的不断增强,越来越多的嵌入式设备开始采用较为复杂的 GUI 系统,嵌入式设备的 GUI 系统发展得十分迅速[1]。传统的嵌入式 GUI 系统,如 Microwindows 等,由于项目规模较小,功能较薄弱,缺乏第三方支持等诸多原因,已经越来越不能满足用户的需要。

Qt/Embedded 是著名的 Qt 库开发商 Trolltech 公司开发的面向嵌入式系统的 Qt 版本。Qt/Embedded 与 X11 版本的 Qt 接口在最大程度上兼容,大部分基于 Qt 的 X-Window 程序移植到 Qt/Embedded 是非常

**收稿日期**:2017-04-24　　**修回日期**:2017-08-23　　**网络出版时间**:2018-02-08
**基金项目**:国家核高基重大专项(2015ZX01040-201)
**作者简介**:王　凯(1984-),男,工程师,研究方向为软件体系结构。

方便的。Qt/Embedded 延续了在 X-Window 上的强大功能[2-3]。Qt/Embedded 类库完全采用 C++封装,丰富的控件资源和较好的可移植性是 Qt/Embedded 最为优秀的特点。Qt 是完全面向对象的,易扩展,并且允许真正的组件编程。使用 X-Window 下的开发工具 Qt Creator 可以直接开发基于 Qt/Embedded 的 UI(用户操作接口)界面。采用 Qt/Embedded 开发嵌入式 Linux 下的应用软件的第三方公司也日益增加。

文中分析了 Qt/Embedded 的体系结构和底层支持的方式,阐述了 Qt/Embedded 图形事件服务的源码和架构原理,通过结合帧缓冲驱动体系与 Qt/Embedded 图形引擎,对在嵌入式系统下如何实现图形硬件加速进行了详细的研究和实现。

## 1 Qt/Embedded 底层支持分析

### 1.1 Qt/Embedded 的体系结构

Qt/Embedded 库是 Qt 图形界面库的嵌入式版本,为了适应嵌入式操作系统环境,做出了许多改动和调整。它放弃了原本使用的 X11 也就是 X Window 体系去构建 GUI 环境,因此也就不需要使用较大的 Xlib 库,所以内存占用十分小,能够适应嵌入式开发的需求。在放弃了 X Window 体系之后,Qt/Embedded 库底层使用帧缓冲体系作为底层图形接口,并使用输入事件作为具体的输入设备的抽象[4],比如 keyboard 和 mouse 等。在上层,Qt/Embedded 库继续使用原本的 Qt 架构,从而保证用户使用的便利性和一致性,并能够与 X Window 系统下的程序兼容,使得用户可以方便地进行程序移植。

Qt/Embedded 图形引擎的基础是图形帧缓冲体系(framebuffer),它是一种采用 mmap 系统调用的驱动接口,正是依靠帧缓冲驱动,屏幕才能显示内容,帧缓冲驱动为上层 QWS Server 提供图形支持,同时也为下层操作系统提供对图形事件访问服务的接口[5]。

### 1.2 Qt/Embedded 的图形引擎架构

Qt/Embedded 的 GUI 系统采用了 C/S 结构,每个独立的 Qt/Embedded 程序都允许作为系统中唯一的一个 GUI Server 存在。在应用程序首次以系统 GUI Server 的方式加载时,将创建 QWS Server 实体。QWS Server 创建底层显示设备抽象基类 QScreen,其中声明了对于显示设备的基本描述和操作方式,如打开、关闭、获得显示能力、创建 GFX 操作对象等等[6]。同时,QWS Server 调用 QWSServer::openDisplay()函数创建窗体,在窗体创建过程中调用 QWSServer::Data::Init()进行初始化工作,之后创建 Qt/Embedded 的图形引擎中帧缓冲驱动类 QLinuxFbScreen 的实体对象,在 QLinuxFbScreen 对象中获取设置 QScreen 关键指针 qt_screen 的信息并调用 connect()打开帧缓冲体系中的抽象显示设备(dev/fb0)。在 QWSServer 中所有对显示设备的调用都由 qt_screen 发起[7]。

至此,完成了 Qt/Embedded 中图形引擎的创建,并通过 Qt/Embedded 的图形引擎中的帧缓冲驱动类与系统帧缓冲驱动直接相关联,由系统帧缓冲驱动提供进一步的支持。

## 2 帧缓冲驱动体系结构

### 2.1 帧缓冲驱动显示原理

帧缓冲(framebuffer)是 Linux 系统的显示设备驱动接口,Linux 系统通过它屏蔽不同图形硬件的底层差异性,对显示缓冲区进行抽象,实现上层应用在图形模式下直接对显示缓冲区进行 I/O 操作[8]。用户不需要关心物理显示缓冲区的具体位置及数据存放方式,这些都由帧缓冲设备的驱动本身来完成。

在显示过程中,用户进程的显示数据将会存放在帧缓冲映射的内存中,然后显示设备会从帧缓冲映射的内存中将数据取出,然后显示到屏幕上。通过系统调用 mmap 将显示设备物理内存抽象为帧缓冲区,使得用户可以直接访问。对用户来说,它即是物理显存的一个映像,帧缓冲系统将其映射到用户虚拟进程地址空间[9]。对于用户而言,帧缓冲区可以直接读、写、映射。这样用户程序就可以对这块内存区域填充任何已经定义的像素及颜色,而屏幕也就可以在帧缓冲系统的控制下显示画面[10]。

### 2.2 Linux 帧缓冲驱动程序的体系结构

对于 Linux 系统,帧缓冲设备是标准的字符设备,采用“文件层—驱动层”的接口方式,主设备号为 29,从设备号为 0~31[11]。对应的设备文件为/dev/fb*。在 linux 中,fb 设备驱动的源码主要在 Fb.h(linux\include\linux)和 Fbmem.c(linux\drivers\video)两个文件中,它们是 fb 设备驱动的中间层,为上层提供系统调用,为底层驱动提供接口。

由于 Linux 将设备也认为是一个文件,Framebuffer 驱动核心 Fbmem.c 文件中定义了帧缓冲设备的文件层接口,该接口即为 file_operations 结构体,它是提供给用户空间应用程序使用的。在 file_operations 中,统一实现了字符设备文件层次上的文件操作接口函数,用户空间的应用程序通过它的接口函数对帧缓冲设备进行访问[12]。在驱动核心层上方的是 Framebuffer 特定帧缓冲设备层,在这一层中实现具体的 Framebuffer 驱动接口,例如特定帧缓冲设备 fb_info 结构体的注册函数,fb_ops 结构体成员函数以及使用完毕后的结构体注销函数[13]。fb_ops 结构体是特定帧缓冲设备层的核心模块,它的成员函数实现了对应的

帧缓冲驱动功能,最终直接控制显示设备的硬件寄存器[14-15]。

## 3　Qt/Embedded GUI 环境下图形硬件加速实现方法

### 3.1　基于 Qt/Embedded 的嵌入式图形硬件加速架构

通过对 Qt/Embedded 底层支持和帧缓冲驱动体系结构的分析,提出一种基于 Qt/Embedded 环境的嵌入式系统 GUI 图形加速支持体系结构,如图 1 所示。

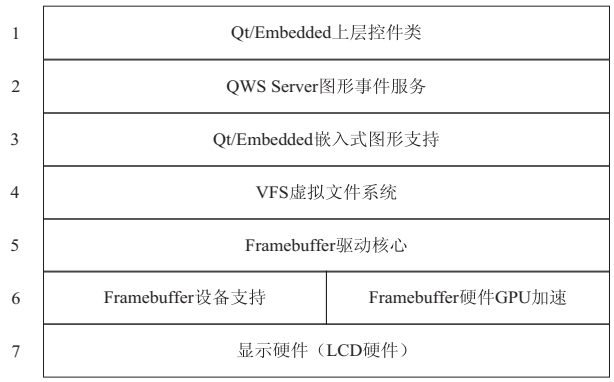


图 1　基于 Qt/Embedded GUI 环境的嵌入式系统图形加速支持体系结构

图 1 描述了实现嵌入式系统图形加速方法的体系结构,从上至下可以分为三部分:

(1)Qt/Embedded GUI 支持及实现部分:该部分包含图中最上方 1,2,3 层,在该部分中,通过 Qt/Embedded 嵌入式图形支持中的 QLinuxFbScreen 对象,实现了 Qt/Embedded 库和帧缓冲体系结构内接口的对接,Qt/Embedded 库使用 Linux 操作系统提供的 open,write,read,close 等文件系统接口函数,基于帧缓冲驱动体系对帧缓冲进行操作。

(2)Framebuffer 驱动核心部分:该部分为图中第 4,5 层,实现了 VFS 虚拟文件系统与帧缓冲驱动体系间的对接,在这一层实现 file_operations 结构体成员函数的内部功能,Qt/Embedded 应用程序通过访问 file_operations 结构体统一实现的文件操作函数对帧缓冲设备进行读写等操作,上层使用的文件系统接口在这里得到具体实现。

(3)Framebuffer 帧缓冲设备支持部分:该部分为图中第 6 层,定义了特定帧缓冲设备 fb_info 结构体,它是特定帧缓冲物理设备的软件实体和驱动层接口的核心数据结构,记录了关于帧缓冲设备属性、参数以及操作函数指针的完整信息。系统中每一个帧缓冲设备对应一个 fb_info 结构体。在对该部分的具体实现过程中,通过使用硬件 GPU 的图形加速功能,优化 fb_info 结构体中对应软件功能接口函数,从而完成 Qt/Embedded GUI 环境下的图形硬件加速。



### 3.2　嵌入式 Linux 下的图形硬件加速接口设计

在嵌入式 Linux 的 GUI 图形硬件加速架构下,framebuffer 框架最重要的实体就是 fb_info,其中涉及驱动的操作接口 fb_ops,这些操作接口是 framebuffer 框架实现交互的桥梁,也是实现图形硬件的途径和接口。

fb_ops 以结构体形式展现,内部接口函数数量繁多,如 fb_check_var( ) 检查可修改的屏幕参数并调整到硬件所支持的值,fb_setcolreg( ) 设置颜色寄存器来实现伪颜色表和颜色表的填充等等。fb_ops 结构体的成员函数最终与显示适配器和 LCD 控制器硬件打交道,需要根据显示适配器的硬件设置及 LCD 显示屏的硬件参数进行编写。

fb_ops 结构体图形硬件加速成员函数接口:fb_fillrect( ) 实现屏幕上矩形区域的颜色填充与绘制,fb_copyarea( ) 实现对屏幕区域显示数据的复制与重新绘制,fb_imageblit( ) 实现位图数据的读取与绘制。这 3 个接口函数是实现 GUI 图形硬件加速功能的关键函数,它们的函数声明代码如下:

```
struct fb_ops {
……
/*绘制矩形*/
void(*fb_fillrect)(struct fb_info *info, const struct fb_fillrect *rect);
/*从一个区域复制数据到另一个区域*/
void(*fb_copyarea)(struct fb_info *info, const struct fb_copyarea *region);
/*绘制一幅位图到显示设备*/
void(*fb_imageblit)(struct fb_info *info, const struct fb_image *image);
……
}
```

通过上述 fb_ops 图形硬件加速成员函数接口,在函数体中可以使用硬件图形绘制命令和算法,从而释放软件绘制图形占用的 CPU 时间,实现图形绘制的硬件加速。

Qt/Embedded GUI 应用程序通过 fbmem.c 文件的 file_operations 结构体统一实现文件操作接口对帧缓冲硬件设备的访问,特定帧缓冲硬件设备的软件抽象与 fb_info 结构体一一对应,fb_info 结构体的注册、注销及其中成员函数的实现工作由内核中对应的 xxxfb.c 文件来完成。指定的 fb_ops 结构体成员函数作为驱动层的核心模块实现对应的图形硬件加速操作,最终通过图形硬件控制及加速算法实现图形绘制工作硬件化。

### 3.3　嵌入式 Linux 下的图形硬件加速方法实现

通过前面建立起的嵌入式 Linux 下的图形硬件加

速架构和硬件加速系统的设计,在实现过程中可以充分利用图形硬件加速的优势:2D 图形硬件加速没有大量的循环操作占用 CPU 时间,在硬件加速过程中,只有两个基本的 CPU 操作—等待和赋值,极大地节约了 CPU 时间,从而大大地提高了图形操作的效率。其中具体的绘图工作通过 DMA(直接内存存取)由 GPU(图形处理单元)完成,在使用 GPU 的过程中,依赖 GPU 提供的各种功能寄存器对 GPU 的功能进行控制。等待操作和 DMA 操作的具体流程如图 2 所示。

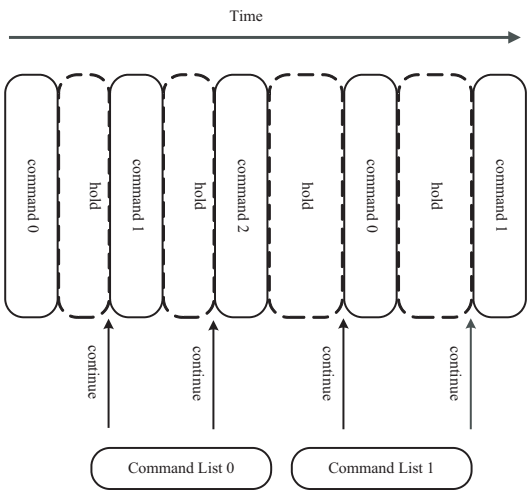


图 2 等待操作和 DMA 操作的流程

图 2 中一共有两组不同的绘图操作,每组操作内部都是等待 DMA 命令状态和绘图操作执行交替执行。在 DMA 等待过程中,释放 CPU 时间片,CPU 处于空闲状态。控制 DMA 等待和开始执行的寄存器为 DMA 控制命令寄存器 DMA_COMMAND。

在具体的实现过程中,2D 图形硬件加速包含如下内容:矩形填充、位块传输 BitBLT、图像旋转和地址定位、图像剪切、光栅操作、矩形区域复制等。

现选取较为有代表性的矩形填充、位块传输 BitBLT、矩形区域复制三个操作进行具体实现。

### 3.3.1 矩形填充

矩形填充,指的是对一个矩形区域进行像素的填充。矩形填充功能的实现,依赖于 GPU 提供的 Fast Solid Color Fill(快速固色填充)功能,通过前文提到的 fb_fillrect( )接口实现。在 fb_fillrect( )接口的输入参数结构体 fb_info 中,获得了目标图像的基地址、目标图像颜色以及图像颜色模式和通道信息。在输入参数结构体 fb_fillrect 中,获得了目标图像的左上角坐标和右下角坐标,确定了图像的位置,在确定目标图像信息后,将这些信息输入到快速固色填充功能寄存器中,启动 GPU 进行图像的绘制。矩形填充的逻辑流程见图 3。

在矩形填充开始前,需等待上一个 DMA 操作结束,通过访问前文提到的 DMA 控制命令寄存器 DMA_COMMAND 来获取当前 DMA 状态,在 DMA 可用之后,从结构体 fb_info 中读出目标图像基地址,将它写入地址寄存器 DST_BASE_ADDR_REG,然后从 fb_info 中读取出目标图像颜色相关信息,从结构体 fb_fillrect 中读出图像最终显示位置坐标,将它们分别写入每个对应的功能寄存器,最后通过使能 Fast Solid Color Fill 命令寄存器 BITBLT_COMMAND_REG 启动 GPU 的快速固色填充功能,完成图像矩形填充的绘制工作。

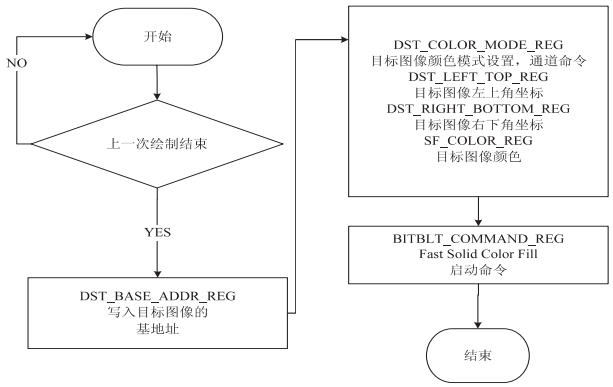


图 3 矩形填充的逻辑流程

### 3.3.2 位块传输 BitBLT

位块传输 BitBLT,指的是将源设备环境区域中的像素进行位块(bit_block)转换,以传送到目标设备环境。位块传输功能依赖于 GPU 中的 BitBLT 功能,在前文提到的接口 fb_imageblit( )中实现。在 fb_imageblit( )的参数结构体 fb_info 中,获取目标图像的基地址和左上角和右下角坐标,目标图像的前景色和背景色,目标图像的 stride 值等信息,在参数结构体 fb_image 中,获得了源图像的左上角和右下角坐标,源图像的数据指针、颜色模式、通道模式等信息。在获得所有相关信息后,使能位块传输的命令寄存器启动 GPU 的位块传输功能。位块传输 BitBLT 的逻辑流程见图 4。

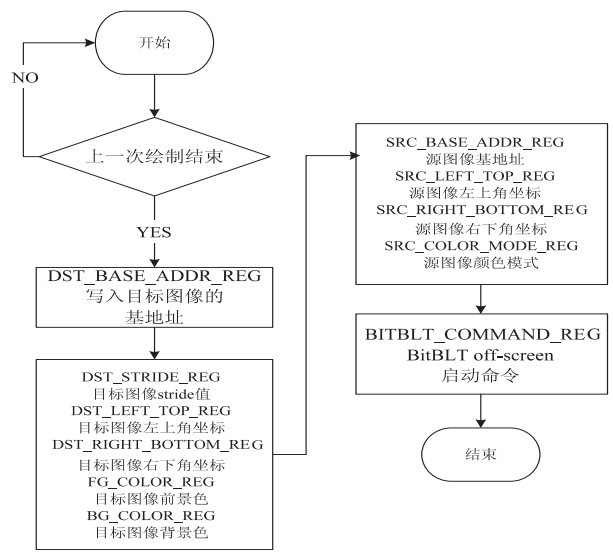


图 4 位块传输 BitBLT 的逻辑流程

在实现位块传输的过程中,需要注意的是 GPU 硬

件中的 BitBLT 操作分为 On-Screen Rendering(在屏渲染)和 Off-Screen Rendering(离屏渲染)。on-screen BitBLT 操作在屏幕上拷贝一个矩形像素块到屏幕上的另一个位置,而 off-screen BitBLT 操作是将像素数据从 off-screen 内存放到屏幕内存帧缓冲区,当 SRC_COLOR_MODE_REG 值和 DST_COLOR_MODE_REG 不相同时,该操作会自动进行颜色模式转换。在这里实现的是 off-screen 位块传输 BitBLT 操作。

3.3.3　矩形区域复制

矩形区域复制,指的是选择一个矩形区域的图像并把它复制到另一个矩形区域显示。实现该功能由 GPU 提供的 Clipping 功能以及 GPU 的 BitBLT 功能中的 On-Screen Rendering 共同完成。GPU 的 Clipping 功能提供一个 clipping window,使用这个 clipping window 确定裁剪的图像大小及位置,clipping window 的大小必须小于屏幕大小。在前文的 fb_copyarea( )接口中,通过 fb_copyarea 结构体获得需要裁剪的区域位置和大小,并将位置和大小参数传到 Clipping 功能寄存器,然后通过 on-screen BitBLT 操作将裁剪的图像绘制到屏幕。绘制的目标图像的信息从参数结构体 fb_info 中获取。矩形区域复制的逻辑流程基本与图 4 类似,在矩形区域复制的过程中,无需进行图 4 中的源图像寄存器相关操作。

在矩形区域复制的过程中,源图像的基本信息大部分都由 GPU 硬件 Clipping 操作获得,在实现过程中有 3 点需要注意:源基地址和目标基地址一致;源 stride 值和目标 stride 值一致;源颜色模式和目标颜色模式一致。

反映在寄存器对应关系上即为:

SRC_BASE_ADDR_REG = DST_BASE_ADDR_REG

SRC_STRIDE_REG = DST_STRIDE_REG

SRC_COLOR_MODE_REG = DST_COLOR_MODE_REG

这也是矩形区域赋值的前提要求。

## 4　图形硬件加速测试

### 4.1　测试平台与环境

文中使用的测试平台是讯为科技的 iTop-4412 开发板,iTop-4412 开发板精英版搭载三星 Exynos 4412 处理器,配备 1 GB DDR3 内存,4 GB 固态硬盘 EMMC 存储。

三星 Exynos 4412 处理器搭载 4 核心 Cortex-A9,每个核心的主频大小为 1.6 GHz,内部图形核心为 Mail400MP。

测试平台的软件环境为嵌入式 Linux+Qt 操作系统,Linux 内核版本为 3.0.15,Qt/Embedded 库版本为 4.7.1,使用 u-boot-1.3.4 引导启动软件环境,交叉编译器为 arm-2009q3。

### 4.2　测试方法

测试以最典型的矩形填充功能为例,编写基于 Qt/Embedded 库的 Qt GUI 测试例程。在该例程中,使用 Qt/Embedded 库自带的矩形绘制功能 drawRcet 函数进行矩形填充,通过内部计时器计算出 Qt/Embedded 库的绘图时间,同时在该例程中通过 ioctl 接口调用前面实现的底层硬件加速功能进行矩形填充,按同样方式计算出绘图时间,并将该时间与 Qt/Embedded 库的绘制时间进行对比,从而得出测试结果。

### 4.3　测试结果与分析

测试 1,绘制宽度 * 高度为 40 * 30 像素的矩形,重复次数为 10 000,20 000,30 000,40 000,50 000,结果见图 5。

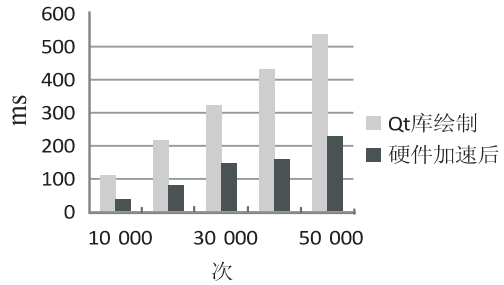


图 5　不同绘制次数下的加速效果对比

从图中可以看出,随着绘制次数的增加,Qt 绘制矩形的时间基本呈线性增长,在使用硬件加速功能后,绘制同样次数矩形的时间大大缩短;同样,随着绘制次数的增加,加速后的绘制时间也单调递增,但增加的幅度略有波动。对图中数据进行计算后可知,加速比(加速前时间/加速后时间)基本在 2.3 ~ 2.7 之间,实现了图形绘制硬件加速的效果。

测试 2,绘制 50 000 次不同像素的矩形,像素宽度 * 高度分别为 40 * 30,60 * 45,80 * 60,100 * 75,120 * 90,结果见图 6。

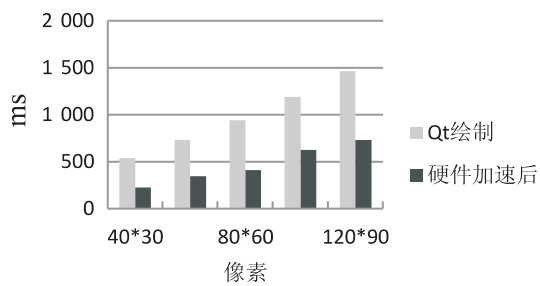


图 6　绘制不同像素矩形的加速效果对比

从图中可知,随着绘制矩形像素数目的增加,无论是 Qt 绘制,还是硬件加速后绘制,绘制所需的时间都基本是线性增加的,使用 Qt 绘制时间的增加基本与像素数目的增加成正比,增量较为平滑,使用硬件加速后

的绘制时间的增量略有波动。

综合图 5 和图 6 的数据可知,硬件加速后,在矩形填充这个基本图形操作上,硬件加速获得了较好的效果,加速比基本维持在 2 以上,实现了预期的目的。

## 5 结束语

通过对 Qt/Embedded 体系结构和 Qt/Embedded 图形引擎架构的分析,利用嵌入式 Linux 下的帧缓冲体系结构,提出一种 Qt/Embedded GUI 环境下图形硬件加速实现方法和图形加速实现架构。实现了帧缓冲体系对 Qt/Embedded GUI 体系的底层硬件支持,依赖于 Linux 帧缓冲体系的加速接口,使用硬件 GPU 内部加速功能对 Qt/Embedded GUI 环境下的基本图形绘制进行了硬件加速,取得了较好的加速效果。

**参考文献:**


[1] 蒋 飞. 基于嵌入式 Linux 系统的数字电视 GUI 图形加速设计[D]. 北京:北京邮电大学,2010.

[2] 李升亮,徐剑锋,李峻林. 嵌入式系统中的多窗口 GUI 系统的研究[J]. 计算机与数字工程,2008,36(10):126-128.

[3] 周 鸿. 基于嵌入式系统的智能缝制设备研究[D]. 西安:西安电子科技大学,2010.

[4] 李军民,祝红军. 基于 ARMlinux 平台的 QT/E 键盘实现[J]. 微计算机信息,2008,24(26):27-29.

[5] 周 开,倪 伟. 基于 Qt/E 的嵌入式 Linux GUI 研究与实现[J]. 淮阴工学院学报,2015,24(3):10-13.

[6] 严吉国. 基于嵌入式 Linux 的 200MHz 数字存储示波器的设计与实现[D]. 南京:东南大学,2009.

[7] CHEN F, FAN X. Embedded system's performance analysis with RTC and QT[C]//Proceedings of the 7th international conference on advanced parallel processing technologies. Berlin:Springer-Verlag,2007:569-579.

[8] 郭小梅. Linux 下的帧缓冲设备驱动研究与应用[J]. 工业控制计算机,2012,25(6):3-4.

[9] YANG L, SANDER P V, LAWRENCE J. Geometry-aware framebuffer level of detail[J]. Computer Graphics Forum, 2008,27(4):1183-1188.

[10] MAO C, JOHNSON K M. Fast-switching liquid-crystal-on-silicon microdisplay with framebuffer pixels and surface-mode optically compensated birefringence[J]. Optical Engineering, 2006,45(12):1269-1278.

[11] 赵 洁,龚 威. 嵌入式 Linux 帧缓冲设备驱动程序[J]. 计算机系统应用,2010,19(12):208-211.

[12] CHANG C Y, HUANG C H, CHU Y S. Efficient memory access methods for framebuffer-less video processing applications[C]//IEEE international symposium on circuits and systems. [s. l.]:IEEE,2013:3026-3029.

[13] 宋方伟,刘 勇,聂诗良,等. SM502 移动多媒体协处理器在嵌入式系统中的应用[J]. 兵工自动化,2010,29(2):91-92.

[14] 苏哲欣,刘鸿飞,薛 晓. 基于嵌入式 Linux 的 LCD 驱动分析与实现[J]. 工业控制计算机,2009,22(2):29-30.

[15] 黄相平,余水宝,夏 灿. 基于 S3C6410 平台的嵌入式 Linux 系统 LCD 驱动模块[J]. 微型机与应用,2013,32(13):9-12.


(上接第 66 页)


漂移方法[J]. 计算机工程,2008,34(16):210-211.

[3] 战守义,井 新. 加入时间因素的个性化信息过滤技术[J]. 北京理工大学学报,2005,25(9):782-785.

[4] 蒋 萍. 基于用户兴趣挖掘的个性化模型研究与设计[D]. 苏州:苏州大学,2005.

[5] 史朝辉,王晓丹,杨建勋. 一种 SVM 增量训练淘汰算法[J]. 计算机工程与应用,2005,41(23):187-189.

[6] 李 娜. 基于垂直搜索引擎的农业信息推荐关键技术研究[D]. 沈阳:沈阳农业大学,2016.

[7] 韩春晓. 中文期刊个性化搜索引擎的设计与实现[D]. 哈尔滨:哈尔滨工业大学,2014.

[8] 张梅芳. 基于改进 PageRank 算法和用户兴趣的个性化搜索研究[D]. 天津:河北工业大学,2014.

[9] 王 哲. 一种基于位置服务的个性化美食搜索算法研究与实现[D]. 长沙:湖南大学,2013.

[10] 黄华东. 基于用户模型的个性化搜索研究[D]. 上海:华东理工大学,2013.

[11] 邓晓嘉. 一种基于 RSS 用户兴趣的个性化搜索系统[D]. 北京:北京工业大学,2010.

[12] 石志伟,刘 涛,吴功宜. 一种快速高效的文本分类方法[J]. 计算机工程与应用,2005,41(29):180-183.

[13] QIU Feng, CHO J. Automatic identification of user interest for personalized search[C]//Proceedings of the 15th international conference on world wide web. Edinburgh, Scotland, UK: ACM,2006:23-26.

[14] KOUTRIKA G, IOANNIDISY. Personalized queries under a generalized preference model[C]//Proceedings of the 21st international conference on data engineering. Tokoyo, Japan: IEEE,2005.

[15] CLAYPOOL M, LE P, WASEDA M, et al. Implicit interest indicators[C]//Proceedings of the 6th international conference on intelligent user interfaces. Santa Fe, New Mexico, USA: ACM,2001:33-40.

[16] SHEN Xuehua, TAN Bin, ZHAI Chengxiang. Implicit user modeling for personalized search[C]//Proceedings of the 14th ACM international conference on information and knowledge management. Bremen, Germany: ACM,2015:824-831.