基于嵌入式 GPU 虚拟仪表图形软件的实现①

郭 云, 康 涛

(特种显示国家工程实验室 现代显示国家重点实验室, 上海 201114)

摘 要: 提出并实现了一种基于嵌入式 GPU(OES: OpenGL® ES)的跨平台图形应用软件的系统框架. 它包括外部事件的驱动, 图形应用软件, 嵌入式系统入口, 嵌入式系统硬件等四个模块. 外部事件的驱动主要是响应外部数据或事件的变化, 从而控制图形显示内容的实时更新, 以及功能画面的实时切换. 图形应用软件模块包括了三个组成部分(1)接口界面 (2)中间通讯层(3)处理单元. 图形应用软件的接口界面主要是实现客户化的目标要求, 采用 C++类的面向对象的设计方法. 中间通讯层, 是为了实现图形应用的任务而安排的结构化的类. 处理单元是各种最基本内容的单元实现, 它建立在我们的各种实用库之上. 嵌入式系统入口, 它封装了图形软件的核心函数功能, 实现和上层的处理单元间的数据调度. 嵌入式系统硬件模块主要是各主流平台(CPU, GPU)相关的数据信息, 支持上层的图形应用. 本文在虚拟仪表盘面上实践了上述应用软件系统, 满足了实时响应, 高效处理, 高质量图形显示的要求. 为实现嵌入式平台的图形显示应用打下了重要的基础. 同时, 本文的工作提出并解决了若干嵌入式图形显示技术的优化问题, 为嵌入式图形显示开发提供了有力的帮助.

关键词: 嵌入式 GPU(OpenGL ES); 图形显示; 虚拟仪表; 嵌入式图形优化技术

Virtual Instrument Software System of Cross Platform Based on GPU

GUO Yun, KANG Tao

(National Engineering Laboratory for Special Display, State Key Laboratory of Modern Display, Shanghai 201114, China)

Abstract: We have developed a virtual instrument software system of cross platform based on graphics processing unit (GPU) and OpenGL® ES. This software system can do real-time rendering graphics context according to the external event trigger. It includes event driver, graphics display, embedded system entry and embedded system hardware modules which have the flexibility and extendibility. We take some key technologies to optimize the flow of embedded graphics application.

Key words: embedded GPU (OpenGL® ES); graphics display; virtual instrument; optimization technology for embedded graphics

1 引言

嵌入式系统的图形应用正在迅猛的发展到各种技术和工业应用领域, 包括各种移动通讯设备, 手机, 航空仪表, 车载显示, 游戏娱乐等. 在仪表显示中主要以虚拟仪表的应用最为新颖. 以事实工业标准 ARM®, 和图形工业标准 OpenGL® ES 为组合的嵌入式开发系统, 已经得到广泛的业界支持, 并且已经成为各个软硬件应用厂家的开发标准. 在不同的操作系统中都支持 ARM+OpenGL ES 的体系结构, 如 Windows CE, Linux, 和实时操作系统 VxWorks 等. 基于 SoC 设计的 GPU(Graphics Process Unit)使我们开发高速和实时的应用系统成为现实. 各种 GPU 的开发, 设计公司, 都会支持 OpenGL ES 的图形标准, 开发各自的驱动程序来满足 OpenGL ES 的技术要求. OpenGL ES 的标准是在 Khronos Group Inc®公司进行维护和更新. 目前主要有 OpenGL ES 1.0 , OpenGL

① 收稿时间:2012-02-29;收到修改稿时间:2012-04-09

ES1.1 和 OpenGL ES 2.0 版本. OpenGL ES 1.0, OpenGL ES1.1 支持固定图形管线的几何和图像渲染操作，而 OpenGL ES 2.0 是支持 GLSL(OpenGL shading language)的可编程的几何和像素的渲染管线. OpenGL ES 的开发包组成是: OpenGL ES®的标准 C 函数库, OpenGL ES®的扩展函数(包括 Nvidia®, S3®等厂家的技术扩展), 可以参看 OpenGL® ES Common/ Common-Lite Profile Specification[10]和 OpenGL® ES Extension Pack Specification[11]. OpenGL® ES2.0 不具有向前的完全兼容性(Khronos Group Inc®), 它是主推可变顶点和像素渲染的可以自定义几何和像素处理的图形引擎.

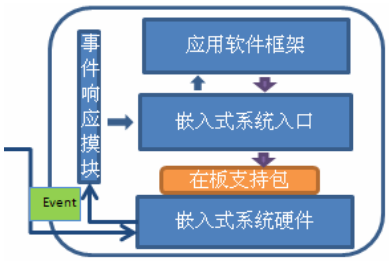


图 1　跨平台图形应用软件框架

图 2　虚拟仪表盘的 GPU 图形显示

虚拟仪表的软件实现架构(Virtual Instrument Software Architecture: VISA)是虚拟仪器仪表应用的一个工业标准，当前由 IVI Foundation[3]组织在维护, National Instruments, Agilent Technologies, Rohde & Schwarz, Tektronix 和 Kikusui 等国际公司都在贯彻和执行该软件标准. 基于嵌入式 GPU 架构上的虚拟仪表的图形软件，在总体设计和模块的执行中，需要考虑和技术上实现相应的架构功能.

文献[13]探讨了使用 OpenGL 中颜色渲染，以及用反走样来实现一些图形单元在汽车仪表盘中的显示. 我们考虑跨平台的，基于嵌入式的图形显示技术，并实现一个具柔韧性和扩展性的通用的应用框架，屏蔽掉不同系统版本的不兼容性问题，不同驱动厂家的 GPU 支持的差异性，和不同操作系统环境下的配置差异等问题，充分利用当前 GPU 性能和系统配置能力，实现高速，实时，高质量的图形显示系统.

2　嵌入式图形应用软件框架的实现

本文将详细叙述我们提出的基于嵌入式 GPU (OES)的跨平台的图形应用软件框架和它的实现方式. 主要的软件开发的指导思想是按软件工程的理念，考虑软件的生命周期，尽可能有效降低开发成本，简化开发和维护软件的生产流程、提高软件跨平台使用的通用性能，最大限度的利用嵌入式系统的计算和显示能力. 为了实现上述目的，基本采用了下面的技术方案.

下图 3 是软件系统的全体架构图. 它主要包括了，外部的事件响应模块， 接收虚拟仪表的外部数据和事件的变化，图形应用软件框架，嵌入式系统入口，嵌入式系统硬件. 图形应用软件框架运行在不同的嵌入式系统软件和硬件之上，具有柔韧性和可扩展性. 它的界面接口可以允许我们接收不同的客户化需求. 嵌入式系统模块可以保证运行于不同的嵌入式操作环境和不同的 GPU 芯片上，并且能随着不同厂家对 OpenGL® ES 的驱动的更新，来适应和增加新的应用功能.

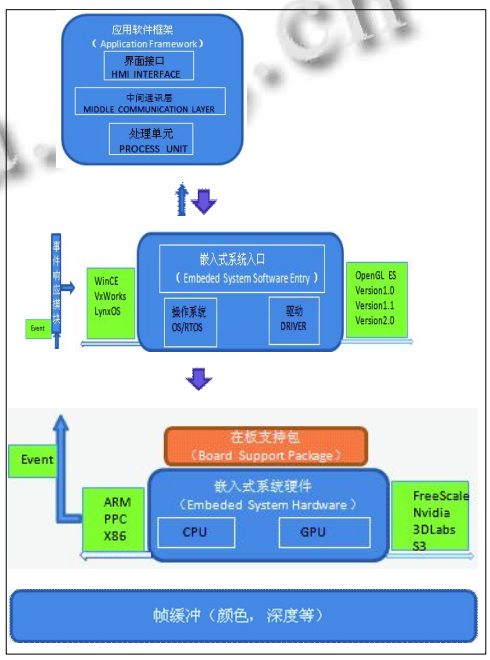


图 3　软件的系统框架(粗箭头是事件的数据流, 细箭头是程序执行流)

整个软件系统的开发，我们采用了柔韧的面向对象的结构化设计思想. 主要是以快速计算的 C 结构和类强化的 C++为组合， 使用在不同层次的程序执行中. 下面我们逐步介绍各个模块的内容和它的实现.

我们设计的[事件响应模块]，在系统外部的某个条件发生变化的时候就会激发某个应用程序，通知我们的图形应用框架，执行响应的显示任务. 我们设置了两种通用的独立事件机制，队列和即时回调，来保障不同的接收事件和事件的处理方式.

队列事件是用来响应按顺序发生的事件的存储，处理和图形显示任务的执行. 它在时间上有个顺延的过程. 即时回调函数是对外部事件的立即响应和执行. 本文对虚拟仪表显示任务中可能出现的事件，做了类型定义，对不同类型的事件，在系统接收到相应的任务后，会做出不同的显示响应. 同一种类型的事件，允许在不同的事件机制间切换.

下面的图 4，说明了事件的处理流程. FIFO(First In First Out)和 OnWait 是队列事件的两种处理模式. FIFO 是按事件发生的先后，进入嵌入式系统，而 OnWait 是系统在队列里寻找特定的事件，该事件存在的话，就立即处理. 对一个事件，定义了它的分类，类型，ID，激发机制，以及所携带的数据等. 通过定义的事件类型和它所属的事件处理机制， 判断它是进入队列或是立即执行. 在通过系统层次的消息传唤机制进入到嵌入式系统入口后，由图形应用软件框架判断它的数值或逻辑意义，即时地更换虚拟仪表盘面的帧显示内容. 图 5 是一个模拟显示，根据检测到的外部空气速度变化，通过上述机制在模拟显示器上即时显示的内容.

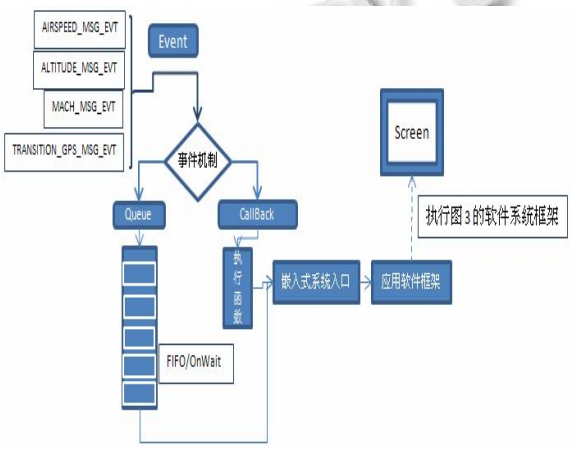


图 4 事件的数据流(实线箭头方向)

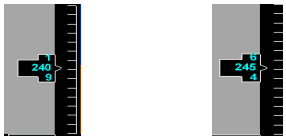


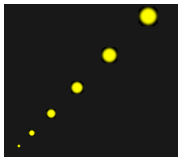
图 5 数据变化的实时图形显示

[图形应用软件框架]主要包括三个组成 (1)界面接口(HMI: Human-Machine Interface) (2)中间通讯层(Middle Communication Layer) (3)处理单元(Processing Unit). 采用面向对象的 C/C++设计方法，按照嵌入式开发的流程，考虑的设计思想是支持软件开发的各个阶段，包括需求分析(Requirement Analysis)，设计和原型(Design and Prototyping)，仿真和测试(Simulation and Test).

接口 HMI，采用客户化的目标方式，目标是根据客户对虚拟仪表显示内容的不同要求，调用相应的模块函数，具有可定制性，可扩展性. 向上与客户数据源接口，同时也接受外部事件来的数据，通过嵌入式系统入口模块传递到 HMI，完成数据驱动的显示任务. 向下通过 OpenGL® ES 的扩展支持(本文的工作中，采用的应用接口是: PowerVR[12])来实现与嵌入式操作系统入口资源的交互. 在虚拟仪表(VISA)[2]的规格明细中定义了一个 Soft Front Panel 的概念，提出了仪表盘面的通用组件等，我们把虚拟仪表的一个显示画面里的各种功能组件，封装成不同的分块 Model，各个 Model 块由用户给出需求(颜色，位置，透明，字体，线渲染，三角面渲染，纹理图像资源需要否等)，因此它是基于客户的可定制的界面接口.

中间通讯层 MCL，根据客户化的虚拟仪表画面的定义数据，调用处理单元和 GPU 的资源，完成 HMI 来的或外部事件驱动的实时计算和显示任务. 整个核心函数，都做了平台，系统无关性的封装，采用宏定义和快速的 C 结构化定义等技术，向上连接客户化的类 C++结构数据和向下连接各功能单元函数库(C，宏等实现)，帮助实现中间的快速计算. 同时将所需求的显示任务通过四个程序阶段完成: 显示环境(View Volume)，虚拟仪表内的各形状结构配置的外表(Visual: 图形显示的纹理图像资源等)，虚拟仪表内图形单元的建模(Modeling)和各个图形单元的渲染 Rendering. 大部分的 CPU 和 GPU 间的计算和资源通讯在这个阶段完成，按照客户的设定目标，确保高效和高质量的效果.

处理单元是最基本的功能单元，在嵌入式 OpenGL® ES 中，具备标准的几何体素(点，线，三角形)的绘制，由这些体素组合成各种形状的物体. 所有这些物体形状数据都转化成顶点数组的形式，可以存放在 OpenGL® ES 的客户端或服务器端. 点的大小和线的宽度等属性，由于客户化目标的多样性，在嵌入式 GPU 驱动中不能满足所有的要求[8]. 本工作提供了自己的处理函数，可以利用 GPU 的快速性能，实现几何体素的大小属性等的变化，如图 6, 7 所示.


(a)扩展的点

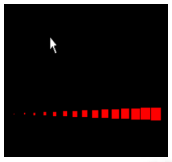
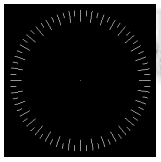

(b)OpenGL ES 自有的点

图 6　不同大小的点


(a) OpenGL ES 自有线

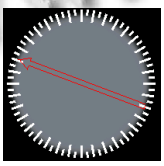

(b)扩展的线

图 7　可变大小的线宽

对于嵌入式环境下的字体处理，主要是有图像纹理和基于矢量字库的两种方法，在 PC 桌面电脑上，基于矢量字库的典型应用是 FTGL[14]，它使用基于表结构的数据存储方式，即时查询，处理所需字符的显示. 由于 PC 机的好的硬件性能和系统资源，可以处理庞大的 FreeType 的字体库. 而对于小资源的嵌入式系统，表结构和即时查询都不是最佳的实现方式，本文采用了基于图像纹理的技术，其中字体的数据都是经过预处理存放在 GPU 中，实际显示字符时，从 GPU 的内部存储中获得数据显示. 基于图像纹理的方式，需要我们预先设置好整个显示任务所需要的字体图像，也不限于语言种类，具备一个基本图形体素在 OpenGL ES 中所有的操作.

(a) 字体


(b) 表盘的数字

图 8　字体的显示(汉字显示例如图 2)

对图形体素的基本效果处理，包括颜色，纹理，透明叠加等也封装在处理单元中. 一些共同的组件，如常用的数学运算(Math)，常数(Const)定义，简单的宏函数(Utilize Macro Function)等可快速实现的结构化数据操作都作为处理单元的一个部分.

在仪表显示中有些通用的组件，如指南针，罗盘，天地球等，可以把它定义为虚拟仪表的公共组件，然后把它作为一个最小的处理单元，可以在不同的仪表盘面中使用.

嵌入式系统入口：包括操作系统(OS/RTOS)和驱动(OpenGL ES)，是系统提供的用于实时渲染的基本 C 库(图形核心函数和外部事件来的驱动)，是直接和底层硬件打交道的软件实现. 它主要是传递外部事件到达系统的通知，针对不同显示屏幕的驱动，以及执行 GPU 功能的核心函数的实现. 不同硬件厂家的 GPU 的驱动函数，由于具有多态型，因此需要抽象出该层的函数结构，便于实现跨平台的，版本自适应的图形计算要求.

嵌入式系统硬件：主要是包括 CPU，GPU，I/O 接口，各 CPU 定义的数据类型，指令集是不一样的,他们控制了系统的计算能力，我们的应用目标是考虑常用的嵌入式 CPU，而本文的工作是基于 ARM 平台. 识别 ARM 以外的 CPU，配置相应的操作系统接口是应用软件框架要改善的将来的工作.

3　基于GPU的嵌入式系统的图形显示优化

在 GPU(OpenGL® ES)的环境下，要保证高速，有效的显示各种不同的客户化目标(虚拟仪表的各种不同形状组合，涉及多边形，圆弧，点划线，园，及它们的布尔运算结果等，以及各种不同的显示效果，如色彩，纹理，透明，Alpha 联合，字体等)，需要我们充分的理解 OpenGL® ES 图形管线下的各功能实现的详细过程.

如下图 9 所示是 OpenGL® ES 的图形管线. 管线上的各种功能是通过图形核心函数库去实现的. 在我们调用某个核心函数功能的时候，图形管线就会去执行该功能所在位置的后续操作，各种图形显示效果的实现都要求我们利用管线的各个阶段资源，设计自己的代码. 也正是这种流水线式的操作形式，给通用功能模块的设计和实现带来了一定的挑战. 相对于桌面 PC 电脑来说，小资源的嵌入式环境下开发复杂的显示内容，不得不进行各种面向显示任务的优化工作.

OpenGL® ES 在 iPhone 环境中的开发[9]提到了一

些优化的问题，也比较详细的介绍了针对 OpenGL® ES 的一些优化方法和实现例子. 对于虚拟仪表的图形显示内容，考虑和提出了下面一些方法.

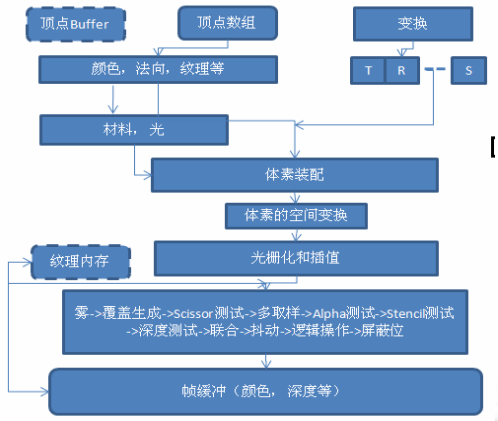


图 9 OpenGL® ES 的图形管线(虚线是 GPU 的存储)

(1) 批量的数据输入，建立形状组合的模型数据，按顶点缓存的形式，保留在 GPU 里面. 例如，对于虚拟仪表盘面上，若干个矩形， 我们定义了一个 QuadGroup，存放在顶点缓存中，完成群组数据的渲染.

(2) 我们使用了一种纹理 atlasp[7]的技术，非常适合在 tile-based 的硬件渲染环境中使用. 它具有线性和二次过滤的反走样能力，在实现文字显示，几何形状外表，和透明效果的时候，可以帮助减少图像调用和 CPU/GPU 的往复处理时间，实现高速运算和图形效果显示.

(3) 减少帧渲染时候的几何变换操作. 由于图形的几何形状使用自己的物体坐标系建立，在摆放到实际的虚拟仪表盘面上的时候， 我们不得不去使用 glTranslate，glRotate，glScale 来完成相应的变换定位，这增加了系统的计算费用. 因此自定义的变换矩阵来完成各个图形单元的定位.

(4) 避免重复使用相同的状态查询，设置的操作命令.

```
glGetFloatv (GL_LINE_WIDTH , &linewidth);
glEnableClientState(GL_VERTEX_ARRAY);
glDrawElements(GL_LINES,vNum1,GL_UNSIGNED_BYTE,vInx1);
glDisableClientState(GL_VERTEX_ARRAY);
. . . . . .
glEnableClientState(GL_VERTEX_ARRAY);
glDrawElements(GL_LINES,vNum2,GL_UNSIGNED_BYTE,vInx2);
glDisableClientState(GL_VERTEX_ARRAY);
```

改写成:

```
glEnableClientState(GL_VERTEX_ARRAY);
glDrawElements(GL_LINES,vNum1,GL_UNSIGNED_BYTE,vInx1);
. . . . . .
glDrawElements(GL_LINES,vNum2,GL_UNSIGNED_BYTE,vInx2);
glDisableClientState(GL_VERTEX_ARRAY);
```

4 仪表图形应用软件的实施

如图 1 是一个框架示意图. 我们模拟飞机的显示仪表盘面上的内容，实现了虚拟盘面的图形显示，如图 2. 该系统使用了 IMX52 的 GPU 处理器， 同时采用了 ARM Cortex-A8 的 CPU，对于虚拟仪表中主显示画面的一些图形要素，将这些图形单元按功能分割为各个模型块，在嵌入式系统入口处，我们安排了仪表主显示画面相关的各种事件，在取得事件信号后，就立即在帧显示内容中做出反应. 图形应用软件框架的界面接口，主要是采用了一个 C++的可面向对象的类来实现，里面封装了客户化目标所需要的对象(包括数据驱动，如图 10)，该对象调用了中间通讯层的各个需要的实例(rendering，visual 等)， 以及完成实例任务所需要的执行函数. 该对象通过第三方的软件(本文采用 PowerVR[12])提供的系统接口(PVRShell，如图 11 所示)向下与系统资源实现交互. 中间通讯层，如结构类 ShapeContext, ViewVolume 等调度和维护 CPU 和 GPU 的内部资源，完成计算任务. 如前所述的处理单元包括了各种内容，是基本的完成任务单位. 它涉及到函数调用中各参数的可用范围配置，常数定义，OpenGL 版本相关信息，常用数学运算，2D 体素模型数据，体素渲染，图像文件，纹理调用，透明函数等. 它是一个可以扩展的独立模块，它的主要表现形式是一些块结构的宏函数，结构化的指针函数等. 例如下面函数定义了一个园的处理模块:

```
GLvoid Circle2DModel( Modeling *gmd, const GLint x0, const GLint y0, const GLint radius)
```

在客户化的目标有需要的时候，我们可以追加相应的函数，建立一个可以扩展的单元层次. 嵌入式系统入口的核心图形函数，执行了图 12 所示的抽象封装，它可以抽象出核心函数的功能，便于不同版本的支持.

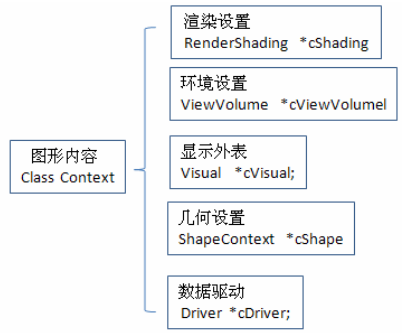


图 10 图形应用软件框架实现

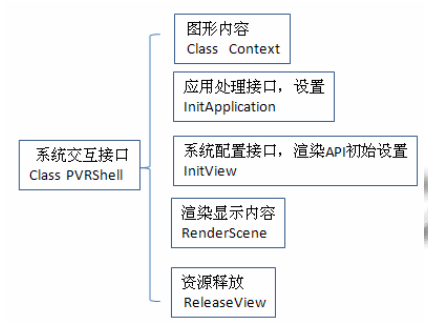


图 11 系统配置资源接口

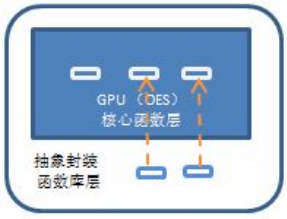


图 12 核心函数的抽象封装

5 结论

在 Windows CE 环境，基于 FreeScale M51X 图形加速芯片，及操作系统接口 PowerVR 的架构下，我们开发了基于 OpenGL ES 功能环境的图形应用软件框架，按 VISA 的标准,定义了软件的事件接口，在具体的显示事例中，我们开发了若干嵌入式环境下的优化技术，满足不同环境下的显示需求.

本文的工作与虚拟仪表软件设计的目标相比较，还是存在很多差距，从结构设计，功能区分，模块划分，视觉表现等等方面还有很多的具体工作需要完善.

目前开发的环境是基于 Windows CE6，在嵌入式系统入口处，应用软件框架只是提供了与 Windows CE 6 相适应的 PowerVR 的接口. 因此开发一个跨操作系统平台的嵌入式系统接口，是本文应用软件框架可以跨平台使用的一个前提. 另外，在一个稍显复杂的仪表显示画面上，如何有效的分布和定位各个图形显示单元，通过调用 OpenGL ES 函数的方式来逐步的定位，相当费时. 目前的国外商业化软件，采取了由设计人员布局画面来生成绘图代码，是一个可以参考的改进模式. 还有，在图形体素，仪表显示组件的种类和数量上，需要丰富和强化，方便用户使用和提高视觉的效果.

6 致谢

我们感谢公司同仁在该系统开发中给与的良好讨论和建议. 感谢汪力立工程师在系统调试和嵌入式平台的搭建中给予的帮助,让我们可以专注于系统的开发，同时也感谢张维强项目经理给予关于飞行仪表的介绍和相关技术的解释.

参考文献

1 Presagis Inc. Embedded Graphics Products, 2007.

2 VXI plug & play Systems Alliance. VPP-2: System. Frameworks Specification, Revision 5.4 February 24, 2010.

3 Virtual Instrument Software Architecture. VISA Specification. http://www.ivifoundation.org

4 VXI plug & play Systems Alliance. VPP-4.3: The VISA Library. Revision 5.0, June 9, 2010.

5 VISA Specification. http://www.ivifoundation.org/.

6 Purnomo B, Cohen JD, Kumar S. Seamless Texture Atlases, ACM SIGGRAPH /Eurographics Symposium on Geometry Processing. 2004.

7 IPhone OpenGL ES Line Width. http://lists.apple.com/archives/mac-opengl/2008/Jun/mas00074.html.

8 The architecture for the digital world. http://www.arm.com/zh/products/processors/index.php.

9 OpenGL ES Programming Guide for iOS, 2011.02.24, Apple Inc.

10 OpenGL® ES Common/Common-Lite Profile Specification Version 1.1.12 (Full Specification) April 24, 2008.

11 OpenGL® ES 1.1 Extension Pack Specification Version 1.03 (Annotated) Editor: Aaftab Munshi.

12 PowerVR Insider Developer Forum. http://www.imgtec.com/PowerVR/insider.

13 聂文琪.嵌入式系统中基于OpenGL的虚拟仪表设计.仪器仪表学报,2005,8.

14 FTGL User Guide, http://homepages.paradise.net.nz/ henryj.