面向移动设备的矢量绘图平台设计与实现

张云贵

2013 年 6 月

中图分类号：TP391.41

UDC 分类号：744

面向移动设备的矢量绘图平台设计与实现

作 者 姓 名　　张云贵

学 院 名 称　　软件学院

指 导 教 师　　张春霞 副教授

答辩委员会主席　陈立平 研究员

申 请 学 位　　工程硕士

学 科 专 业　　软件工程

学位授予单位　　北京理工大学

论文答辩日期　　2013 年 6 月

Design and Implementation of Mobile Device-oriented Vector Drawing Platform

Candidate Name:	Zhang Yungui
School or Department:	School of Software
Faculty Mentor:	A.P. Zhang Chunxia
Chair, Thesis Committee:	Prof. Chen Liping
Degree Applied:	Master of Engineering
Major:	Software Engineering
Degree by:	Beijing Institute of Technology
The Date of Defence:	June 2013

研究成果声明

本人郑重声明：所提交的学位论文是我本人在指导教师的指导下进行的研究工作获得的研究成果。尽我所知，文中除特别标注和致谢的地方外，学位论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京理工大学或其他教育机构的学位或证书所使用过的材料。与我一同工作的合作者对此研究工作所做的任何贡献均已在学位论文中作了明确的说明并表示了谢意。

特此申明。

签　　名：　　日期：

关于学位论文使用权的说明

本人完全了解北京理工大学有关保管、使用学位论文的规定，其中包括：①学校有权保管、向有关部门送交学位论文的原件与复印件；②学校可以采用影印、缩印或其他复制手段复制并保存学位论文；③学校可允许学位论文被查阅或借阅；④学校可以学术交流为目的，复制赠送和交换学位论文；⑤学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

签　　名：　　日期：

导师签名：　　日期：

摘要

近年来，国内数字教育等行业对平板电脑（以 iOS 和 Android 为主）的需求变得迫切，面向移动设备的矢量绘图技术具有较大的发展空间和应用价值。本文针对移动绘图软件移植工作量大、开发困难、缺乏通用开发框架的情况，设计并实现了 iOS 和 Android 的交互式矢量绘图平台（TouchVG 开源框架）。

本文首先分析了移动设备的特性，提出了一种适合多种移动设备的矢量绘图平台的设计方法。在跨平台内核中使用 C++实现绘图功能；在设备相关模块中实现画布和视图适配器、触摸手势识别，并将手势信息委托内核处理。其优点是主要功能跨平台、内核易于扩展、设备相关模块易于实现。

其次，在 iOS 上基于 Quartz 2D 实现了画布适配器，根据实验提出了适合连续手绘的增量绘图的实现方法、快速动态绘图的多层绘图的实现方法，在图形量较大时加快了回显速度。

在 Android 上使用 SWIG 实现了界面层对本地 C++接口的扩展方式，解决了本地引用对象等内存问题。进而基于 android.graphics 包实现了画布适配器，允许跨平台内核回调。提出了两种回显较快的视图设计方式：使用增量绘图技术的普通视图方式；在 SurfaceView 中绘制动态图形的双层视图方式。

最后，给出了 iOS 和 Android 的矢量绘图平台（TouchVG）在数字教育等领域的应用效果。结果表明基于 TouchVG 平台的应用开发效率较高、跨平台性较好。

关键词：矢量绘图；跨平台；移动设备；绘图平台

Abstract

The demand for tablet computer, mostly based on iOS and Android, became more urgent in field of China's digital education, so vector drawing technology for mobile devices has a large space for development and important application value. However, few common development frameworks make it difficult to develop and port mobile drawing software. For this situation, an interactive vector drawing platform (TouchVG) for iOS and Android has been designed and implemented in this paper.

Firstly, this paper proposed a design method of drawing platform for a variety of mobile devices. In this design, most of drawing functions is implemented in the cross-platform kernel using C++. In contrast, other functions such as canvas adapter, view adapter and touch gesture recognition are implemented in the device-dependent module, which will dispatch gesture information to the kernel. This approach has the virtue that the main function is cross-platform, also it is easy to extend the kernel and implement device-dependent module.

Secondly, on iOS, this paper implemented the canvas adapter with Quartz 2D, proposed the incremental drawing technology and the multi-layer drawing technology according to the experiments. It brings faster drawing for hand painting even with large amount of shapes.

On Android, this paper implemented the canvas adapter with android.graphics package, which can be called back by the kernel through JNI generated by SWIG. This paper proposed two approaches to speed up interactive drawing: the incremental drawing technology in a common view, and double-layer technology that has a surface view to draw dynamic shapes.

Finally, this paper introduced application of this drawing platform on iOS and Android in digital education industry. As a result, the approaches for application development is proved to be high efficient and with good reusability in cross-platform.

Key Words: vector drawing; cross-platform; mobile device; drawing platform

目录

图目录

表目录

第1章 绪论

1.1 研究背景和意义

传统的移动设备不适合交互式绘图应用[1]。2010 年 iPad 发布后，涌现了各种平板电脑，目前，国内的平板电脑以 Android 和 iOS 操作系统为主。其高分辨率、较高配置、便携性和性价比使其适合于较复杂的交互式矢量图形应用。

在 iOS 和 Android 上，国内的二维交互式矢量绘图软件极少，缺乏通用绘图的开源框架。移动设备的硬件架构、交互方式及开发环境与 PC 机相差很大，开发难度高。2012 年国内数字化教育发展迅速，移动终端的应用成为重点[2]。面向移动设备的矢量绘图技术在移动学习和互动教学应用（例如手写批注、图文笔记、几何教学、汉字书写）中具有较大的发展空间和应用价值。

根据以上现状的分析，确定选题为研究面向移动设备（iOS 和 Android 操作系统）的交互式二维矢量绘图平台。因为不同应用领域的需求差异大、多数开发人员能够胜任软件界面定制和业务功能扩展的开发，所以将研究范围限定在二次开发平台（通用开发包）及底层实现上。以跨平台开源框架的形式避免在不同平台上重复开发、扩大适用范围。该平台将应用于互动课堂教学、阅读批注和图文笔记等多种软件中。

1.2 国内外研究现状

1.2.1 移动图形引擎的研究现状

在移动设备上的二维图形引擎主要分为下面三大研究方向。

（1）基于渲染流水线的 OpenGL ES[3]和 OpenVG[1,4,5]。前者适合于三维浏览和游戏应用，跨平台性和硬件加速是其主要优点，国内研究很多，有较多开源框架。但使用其进行二维绘图开发会加大系统复杂性、增加开发成本。而 OpenVG 适合于 SVG 和 Flash 等二维图形的高质量渲染场合，目前应用在 GIS 和导航等少数领域。

（2）基于画布模型的二维图形库[6-8]。相对于渲染流水线方式其开发难度较低、与操作系统的内置界面库融合程度高。iOS 和 Android 的核心图形库分别是 Quartz 2D 和 Skia（其 Java 封装框架为 Android Canvas）。Quartz 2D 内部可以使用 OpenGL ES 进行图形渲染和层合并。Android 从 3.0 起可以使用 OpenGL ES 对多数绘图 API 进行硬件加速。因此基于画布模型的图形库在移动设备上也具有较高的性能。

（3）基于 HTML5 Canvas 或移动 SVG 的实现方式[9]。通过 JavaScript 脚本实现交互绘图，其跨平台性、丰富渲染特性和开发难度较低是其主要优点。由于运行开销较大和网页浏览器兼容问题等原因，目前在移动设备上应用还不多。

1.2.2 矢量绘图相关的开源项目

在 iOS 和 Android 上，国外的二维交互式矢量绘图软件较多[10]，国内极少。开源软件或框架较少，表 1-1 列出主要的二维矢量绘图开源框架（OpenGL ES 通常用在游戏引擎和复杂的图形平台中，本文不作研究），这些框架大多在 2011-2012 年发布。

表 1-1 移动平台上的二维绘图开源框架

开源项目	编程语言	特点
PocketSVG	ObjC	从 SVG 文件生成 CAShapeLayers 或路径对象
SVGKit	ObjC	显示 SVG 图形，使用 CoreAnimation 实现图形显示，可交互操作图形
iPhoneTextReader	ObjC、C	使用 Quartz 2D 显示多种格式的电子书
Core-Plot	ObjC	使用 Quartz 2D 和 Core Animation 显示图表，可显示动态变化的图形和光滑曲线
ECGraph	ObjC	使用 Quartz 2D 显示静态图表
SmoothLineView	OjbC	使用 Quartz 2D 快速绘制光滑曲线
Vectoroid	Java	基于 ImageView 交互式绘制矢量图形，可读写 SVG 文件
AndroidPlot	Java	基于 View 或 Widget 显示静态和动态图表
svg-android	Java	基于 PictureDrawable 显示 SVG 文件的图形
TPSVG	Java	基于 Drawable 显示 SVG 文件的图形
AChartEngine	Java	显示多种图表，有扩展框架 ChartDroid
DroidGraph	Java	显示图表，饼图和线图
DroidCharts AFreeChart	Java	显示多种图表，JFreecharts 的 Android 移植版，在应用程序中将 Canvas 传给该框架实现绘图
jjoe64/GraphView	Java	显示条状图和线图，可滚动和放缩显示

iOS 绘图框架通常基于 Quartz 2D 图形库开发，基于 CoreAnimation 动画引擎实现高性能的动态绘图，基于图像视图或 CAShapeLayers 显示大量图形。Android 绘图框架则主要基于普通视图交互绘制。基于图像视图或可绘制对象的渲染方式用于图形较多的场合。这些开源绘图框架使用 C/C++实现的很少，难以同时适用于 iOS 和 Android。

1.2.3 iOS 绘图优化技术

杨硕飞在 2011 年使用 Quartz 2D 取得了较好的绘图效果[8]。除此之外国内对该图形库的学术研究较少，在互联网上的官方资料和翻译文章较多，应用较广泛。

双缓冲显示技术在 PC 上很常见，但照搬到 iOS 上容易出现问题：在 iPad 3 等显示屏设备上绘图很慢（创建位图没有考虑屏幕放大比例时会更慢）。其主要原因[11]是在缓冲位图上绘图无法使用 GPU 的加速特性，复制图像和插补运算很耗时。

离屏渲染技术是移动设备上常用的加速绘图技术，其原理是先渲染图形或图像到 CALayer（内部有矩形纹理）中，在界面显示时由 GPU 合成到屏幕帧缓存中。图形绘制阶段是在 CPU 上进行的，图形复杂时难以快速显示。为了更快显示复杂内容，可采用渐进式渲染技术[12]。在用户交互变慢或 CPU 空闲时在后台线程中进行填充等更丰富的渲染操作。对于拖动平移或捏合放缩动作，即时显示所有图形影响体验，可只改变 CALayer 的变换矩阵[13]，由 CoreAnimation 快速显示。

离屏渲染技术又称为预渲染技术，Kurt 总结了两种实现方法[11,14]：

（1）将绘制好的位图放到 UIImageView 中，让该视图以矩形纹理方式交给 GPU 合成显示，避免了在 CPU 上进行位图复制操作（位图的显示涉及内存复制操作）。

（2）将离屏缓冲位图作为视图的层（CALayer）的内容，但不能实现 drawRect 函数、不能调用 setNeedsDisplay 函数，并确保视图的内容模式不为重绘模式。

对于第一种方法，可以使用透明视图进行触摸响应和动态图形显示，在其下层采用图像视图，在触摸结束后在后台线程中合并图形到图像视图。由于避免了位图复制操作，该方法能达到 60FPS 的流畅程度[15]。

在使用离屏渲染技术可能遇到这两个问题[16]。（1）由于 Quartz 2D 使用 CPU 进行绘制，如果绘制时间太长就会出现 wait_fences 错误。（2）Quartz 2D 在屏幕显示时直接在矩形纹理上绘制，这可避免大内存复制，如果超出 GPU 的缓冲大小（超过屏幕大小或层太多），就会在普通内存块上绘图，然后提交到 GPU，这会导致性能问题。解决方法：（1）减少绘制内容。例如，不显示不可见的图形、在动态放缩时仅

显示图形外边框。（2）将复杂内容分成多个视图或子层渲染，不必全部重新渲染。

将 CALayer 标记为光栅化层能够减少复杂图形的渲染频度，但如果视图经常改变内容就会降低性能，需要禁用自动光栅化，不使用额外的缓冲位图。

对于大图片，避免通过剪裁显示和使用 drawRect 显示，可使用多个 CATiledLayer 分区拼接显示。其他优化方法有：（1）将多条线段合并到一个路径中批量绘制；（2）动画显示时不使用反走样、不显示文本内容；因为 Quartz 2D 需要使用单独缓冲渲染这两种内容，（3）使用整数像素坐标能避免自动反走样。

1.2.4 Android 绘图技术

Android 绘图的主要实现方式是基于 SurfaceView 的多线程绘制方式、在本地动态库中使用 Skia 的实现方式，而基于普通 View 的绘制方式性能相对较低，基于图像视图或可绘制对象的渲染方式用于图形较多的静态图形的绘制中。

高帧率绘图主要使用 OpenGL ES 或 SurfaceView 实现，后者容易使用。Chris 介绍了后者的经典实现方式[17]：在主线程处理触摸交互事件，在内容线程计算和管理图形，由 SurfaceHolder 控制的渲染线程只进行每帧的绘制。Chris 建议避免内存申请和释放，以避免垃圾回收的暂停影响。Chris 通过性能对比指出 Canvas 对于图形不是太多、更新频度不高的场合，其性能也不差。

从 Android 3.0 起在屏幕绘图时可使用 OpenGL ES 2.0 硬件加速绘图，Android 4.0 默认全面硬件加速。基于位图的 Canvas 仍然使用软件渲染方式。Ryan 指出该硬件加速特性的缺点是个别显示效果会改变、多占用进程的 2~8MB 内存，简单使用硬件加速绘制所有图形会导致在部分机型上性能变差[18]。视图的层使用硬件加速时，会占用较多内存，官方推荐仅在动画显示时使用硬件加速。

Robert 建议对程序背景图使用 RGB_565 编码（通常是 ARGB_8888），减少透明视图的层次，减少浮点运算量[20]。这样能大幅减少 CPU 的渲染时间。Android 官方针对性能和 JNI 设计提出了很多建议。例如，使用静态方法而不是虚方法。

1.2.5 图形引擎跨平台的方式

在使用基于画布模型的图形库方面，很多软件都基于某一种图形库（例如 Quartz 2D、GDI+）实现，如果要移植到不同的操作系统上就会很困难。对此侯炯总结了 WebKit 基于图形库适配器的跨平台策略[21]：定义统一的画布接口，在外部插件中使用某种图形库实现该接口。这样确保内核模块跨平台、对应用提供相同的接口。

图形中间件[22]是更进一步的跨平台方式。除了屏蔽图形库差异性外，还可根据应用对性能和内存资源的需求选择基于预缓存的渲染驱动方式和基于图形库适配的渲染驱动方式。还有一种方式是采用跨平台的渲染算法实现图形引擎[23]。

为了弥补某些图形库缺少特定类型曲线的问题，通常将曲线转换为三次贝塞尔曲线[24]。例如使用三次贝塞尔曲线表示任意角度的圆弧、四段 Bezier 曲线表示椭圆、将 B 样条曲线和三次参数样条曲线分解为连续的三次贝塞尔曲线。Mike 总结了基于三次贝塞尔曲线[25]的计算技术。例如曲线转换、包络框、点中测试、图形相交、分割和拼接、外扩、裁剪。

1.2.6 多点触摸和手势识别

Luke 总结了十种经典的核心手势，单指手势有单击、双击、拖动、快滑、长按，双指手势有缩放、旋转、长按并单击、长按并拖动[26]。最后两种手势在主流操作系统中没有内置支持①，单指手势都支持，在 Android 上需要自行识别旋转和缩放手势。研究表明食指、中指或五个手指在交互中使用得最多[27]。较多 Android 机型最多同时识别三个触点信号[28]。所以在绘图平台中应尽可能只使用单指和双指手势。

将触摸事件识别为手势的一个常见问题是手势二义性（手指动作可理解为多种相似的手势类型）。一般通过延迟发送技术解决[29]。基于触点的触摸状态、位移和时间的方法可解决几种手势的冲突[30]：（1）快滑和拖动通过滑行的距离判断；（2）单击、长按和双击手势通过两次点击的时间间隔和按下持续时间判断。在 iOS 上除了采用延迟开始和延时结束的方法外，还允许应用程序使用这几种方法：设置手势静态依赖关系、动态判断新触点能否加入到某个手势、允许对多个手势进行同时识别判断。

1.2.7 编程语言和开发方式

C++适合大多数移动平台[31]。如何在多种平台上实现统一的绘图接口是将要面临的一个难题。iOS 程序可以在一个文件中同时使用 ObjC 和 C++，通过 C++类重载的方式实现扩展和集成。在 Android 平台上主要使用 Java 及 Android SDK 开发应用程序，使用 C/C++及 Android NDK 开发 JNI 接口的本地动态库[31]。

Android 本地动态库的开发存在较多困难，主要有 JNI 编写、NDK 编译和调试、内存泄漏和溢出问题。SWIG 可以将 C++连接到 JNI 接口。Charles 在 2010 年发现使用 SWIG 生成的 Java 类不适合 Android 的 Dalvik 虚拟机[32]，改用 javah 工具从 Java

① 本研究发现：iOS 的长按手势允许有拖动状态，在拖动时相当于“按住并拖动”手势。官方并未宣传此用法。

类生成 JNI 导出函数，只能生成 C 语言的函数定义文件。目前，这些程序的 Java 代码很少调用 C++的类接口，一般是将 C++代码封装到 C 函数中。Android 代码从 C++类继承和扩展则几乎没有。Android 官方也建议少用虚函数、慎用 JNI 本地接口。因此需要研究 Java 与 C++衔接的有效方式，降低开发难度。

本文研究发现 SWIG 官方在 2012 年 4 月修复了该问题，Onur Cinar 在同年 12 月出版的专著中介绍了 SWIG 与 NDK 结合的实现方法[33]。因此可以使用 SWIG 新版本开发 Android 本地动态库。

1.2.8 Android 内存访问

在针对 Android 设计绘图接口时应尽可能使用简单数据类型，少用包装对象，以避免 JNI 接口的性能损失。由于 Dalvik 最多支持 512 个 JNI 本地引用对象，容易出现内存溢出的问题，在 JNI 内部实现上要注意释放 JNI 引用对象[34]。

Android 的每个进程默认最多使用 16MB 内存，容易出现内存溢出问题[35]。通常的解决方法是使用软引用缓存位图对象[36]。但 Android 2.3 以后软引用对象会被强行回收（弱引用对象回收更快），而且从 3.0 起位图数据改放到本地库内部，无法预知其释放时机，容易引起内存溢出问题。可使用 LruCache 或 DiskCache 进行缓存。其他方法有：（1）在显示小图片时降低采样率以便减少内存占用量。（2）对 Activity（应用程序组件）内的线程和异步消息响应者（Handler）特殊处理以便防止 Activity 不能释放。（3）对于背景图（BackgroundDrawable），在 Activity 的销毁函数中对 Activity 解除引用[36]。

1.3 本文的研究内容

移动设备的显示特性、交互方式及开发环境与 PC 机相差很大，如何在受限的硬件条件下开发高性能的绘图平台存在较多技术难题。为了避免在 iOS 和 Android 上重复开发、降低移植工作量，需要设计适合多种移动设备的跨平台绘图架构。

因此，本文对下列工作内容进行重点研究和阐述：

（1）设计跨平台的交互式绘图系统架构，适应 iOS 和 Android 等多种移动设备的显示特性和多点触摸的交互方式。同时减少重复开发、降低移植工作量。

具体设计方案是抽象出画布接口和视图接口，让跨平台内核与设备平台相关的图形库和界面库分离。设计跨平台的手势分发接口，在设备平台相关的界面适配器中识别手势，并映射到内核的手势分发接口，由内核进行命令转发或放缩计算。

（2）研究 iOS 上的矢量图形和图像的显示特性和优化方法，实现基于 Quartz 2D 图形库的高性能的通用交互式绘图平台。涉及多种离屏渲染技术的应用和优化。

（3）基于 SWIG 和 NDK 实现 Android 封装模块（Java）与跨平台内核（C++）的调用和回调扩展，研究并实现基于 Android Canvas 的矢量图形和图像的高性能显示方法，实现通用交互式绘图平台。

本绘图平台（TouchVG）的矢量图形和绘图功能与传统的 CAD 等绘图软件类似，在本文中不重点描述。

1.4 论文组织结构

本文共分为下列七章：

第一章，绪论。说明研究的意义、背景、现状，描述了研究的主要内容。

第二章，相关技术介绍。概括本文涉及的核心技术和基础理论。

第三章，移动绘图平台的架构设计。总结移动设备的特点，设计适合移动设备的扩展机制、平台架构、绘图内核模块及设备接口。

第四章，iOS 绘图平台的实现。试验 iOS 上基于 Quartz 2D 的多种显示技术，总结出性能较高的设计方案。

第五章，Android 绘图平台的实现。使用 SWIG 实现 C++内核与 Android 代码的互操作，实验和总结 Android 绘图技术，给出实现效果。

第六章，绘图平台的应用和评估。从应用方式和效果、平台对比等多个方面对绘图平台进行了评估和总结。

总结和展望。总结本文的主要工作、创新点和不足，及后续研究工作。

第2章　相关技术介绍

本章描述在移动设备上绘图所涉及的关键技术，后续章节将用到这些技术。

2.1　iOS 相关技术

2.1.1　设备参数

表 2-1 列出了 iOS 设备的参数[37]。可见其 CPU 和 GPU 的配置相当高。考虑到 Retina 高清屏在 UIKit 中每点对应两个像素，可知 iOS 设备具有较单一的分辨率、显示精度高：iPad 系列的 DPI 为 132，iPhone/ iPod Touch/ iPad Mini 系列的 DPI 为 163，iPad/iPad Mini 系列按点计算的分辨率都为 1024×768。

表 2-1 iOS 设备列表

设备	内存	屏幕、分辨率、PPI	CPU	PowerVR GPU
iPad 1	256MB	9.7" 1024×768 132	A4 单核 1GHz	SGX535
iPad 2	512MB	9.7" 1024×768 132	A5 双核 1GHz	SGX543 双核
iPad 3	1GB	9.7" 2048×1536 264	A5X 双核 1GHz	SGX543 四核
iPad 4	1GB	9.7" 2048×1536 264	A6X 双核 1.4GHz	SGX554 四核
iPad Mini	512MB	7.9" 1024×768 163	A5 双核 1GHz	SGX543 双核
iPhone 3GS	256MB	3.5" 480×320 163	Cortex-A8 600MHz	SGX535
iPhone 4	512MB	3.5" 960×640 326	A4 单核 1GHz	SGX535
iPod Touch 4	256MB	3.5" 960×540 326	A4 单核 1GHz	SGX535
iPhone 4S	512MB	3.5" 960×640 326	A5 双核 1GHz	SGX543 双核
iPhone 5	1GB	4" 1136×640 326	A6 双核 1.3GHz	SGX543 三核
iPod Touch 5	512MB	4" 1136×640 326	A5 双核 1GHz	SGX543 双核

2.1.2　iOS 绘图相关框架

iOS 上绘图相关框架有 UIKit、Quartz 2D、OpenGL ES，其关系如图 2-1 所示。UIKit 界面框架包含 UIView、UIImageView、UIScrollView 等视图类，基于

CoreAnimation 框架实现图形渲染。CoreAnimation 动画框架使用 OpenGL ES 框架进行动画显示和硬件加速，包含 CALayer 层类。每个视图都有一个 CALayer 对象（实质是矩形纹理），用于缓存绘图内容和供 GPU 加速显示。

Quartz 2D 二维图形库用于绘制界面大多数内容，是 Core Graphics 框架的一部分。后者还包含 Quartz Compositor 层合成模块，用于对各个层基于 GPU 进行纹理合成。OpenGL ES 是 2D/3D 图形框架，可以直接在矩形纹理上使用 GPU 硬件加速渲染。Quartz 2D 在用于屏幕显示时自动使用 OpenGL ES 在 CALayer 上显示（由 CoreAnimation 在初始化 CGContext 时启用）。此时没有使用 GPU 的加速能力[38]。在位图等其他目标上绘图时基于 CPU 进行渲染，没有硬件加速能力。

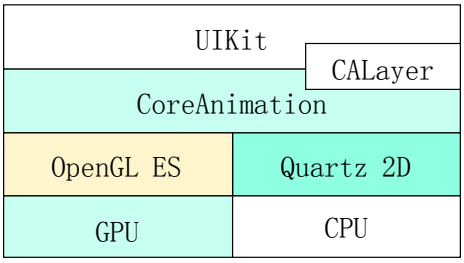


图 2-1 iOS 图形框架的关系

2.1.3 Quartz 2D 图形库和显示原理

Quartz 2D 以设备上下文（CGContext）的画布模型提供绘图接口，可以在屏幕视图、层、位图、PDF 等目标上绘图，只有在屏幕视图和层上绘图时能基于 GPU 加速绘图，在位图和 PDF 上绘图时不自动使用 OpenGL ES 渲染。对于离屏渲染，苹果官方不建议在位图上绘制图形，建议在层上绘制图形提高性能。

Quartz 2D 使用绘制模型进行描绘，所有基于路径的矢量图形、图像和文字内容都叠加描绘到一个虚拟的画布上。

UIKit 的坐标系默认是 ULO 类型，原点在视图左上角，单位为点。Quartz 2D 的默认坐标系是 LLO 类型，原点在绘图目标的左下角，单位是像素。窗口显示时由 UIKit 自动设置 Quartz 2D 的坐标系为 ULO 类型，其他绘图目标需要单独设置坐标系（设置当前变换矩阵），否则就是默认的 LLO 类型坐标系。使用 UIKit 的函数创建位图或 PDF 上下文时，会自动设置坐标变换矩阵。

在视图显示时，绘图内容会缓存在后备缓冲（CALayer 对象）中。在视图的 drawRect 函数中使用 Quartz 2D 等图形引擎绘图，或者直接指定内容到 CALayer。可以指定图像（CGImageRef）到视图的层，或者指定矢量图形路径对象到

CAShapeLayer，或者指定图像（UIImage）到图像视图。

视图的内容模式默认为自动撑满模式，在旋转屏幕或改变显示属性时自动将 CALayer 中已缓存内容重新显示到屏幕上，不会调用 drawRect 函数重新输出图形，这在图形较复杂时能减少 CPU 的消耗量。如果视图的内容模式为重绘模式，或者调用 setNeedDisplay 函数，视图的 drawRect 函数将会再次被调用。通过调用 setNeedDisplay 函数可以让视图显示新的图形内容，实现交互式绘图。交互式绘图的另一种实现方式是使用离屏渲染技术，在单独线程中预先将图形渲染到图像或层对象中，将新的图像或层对象应用到视图，实现快速显示动态图形的效果。

2.1.4 离屏渲染技术

离屏渲染技术又称为预渲染技术，是移动设备上常用的加速绘图技术，其原理是在线程中绘制图形到 CALayer 中，在界面显示时由 GPU 合成到屏幕帧缓存中。为了更快的显示复杂内容，可采用渐进式渲染技术，在用户交互变慢或 CPU 空闲时在后台线程中进行填充等更丰富的渲染操作。为了避免了位图复制操作，可以使用透明视图进行触摸响应和动态图形显示，在其下层采用图像视图，在触摸结束后合并到图像视图。

2.2 Android 相关技术

2.2.1 SWIG 的编程语言转换

SWIG 是支持 C/C++与主流编程语言集成的工具。在 SWIG 接口文件中指定要转换的 C++头文件和转换配置信息，在编译阶段运行 SWIG 程序后，自动从 C++头文件生成其他编程语言可调用的代码文件（对于 Android 就是 JNI 类）和 C++封装实现文件，允许 Java、C#、Python 等代码使用相同编程语言调用 C++的函数，在不同平台复用已有的 C++代码。

SWIG 通过类型重定向（TypeMap）将 C++中的数据结构和变量类型映射到目标语言中的类型，在目标语言中能使用熟悉的类型调用 C++中的接口功能。

SWIG 通过重定向机制（Director）为有虚函数的 C++类分别生成 C++子类和目标语言的类（假定为 B），在 C++子类的重载函数中调用目标语言的基类 B，基类 B 的派生类的相应重载函数就能被执行。该机制是基于虚函数的回调机制。

Android 的 Dalvik 虚拟机对 JNI 本地动态库有特殊限制：（1）必须实现

JNI_OnLoad()函数；（2）不能使用弱引用对象；（3）不能超过 256 个本地引用对象。在编写 SWIG 接口文件时需要注意这些限制，必要时修正 SWIG 所生成的封装文件。

2.2.2 Android 开发方式

在 Android 平台上主要使用 Java 及 Android SDK 开发应用程序，使用 C/C++及 Android NDK 开发本地动态库，本地动态库和应用程序通过 JNI 衔接。可以使用 SWIG 自动生成 JNI 类，以重定向回调机制在 Android 程序中扩展功能。

Android 程序的开发工具通常是跨平台的 Eclipse 集成开发环境和 ADT（Android Development Tools）插件，后者提供了调试（DDMS）和日志（LogCat）功能。Android 本地库的调试配置工作较复杂，可使用日志辅助调试，2013 年发布的 ADT Bundle 套件（含 ADT 21.1）针对本地库提供了简易的集成调试功能，能避免繁琐的配置工作。

2.2.3 Android 绘图相关框架

在 Android 上，常见的绘图框架及其关系如图 2-2 所示。

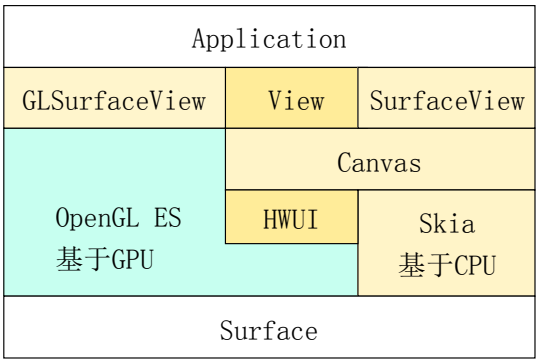


图 2-2 Android 图形组件的关系

绘图视图主要有三种类型：（1）SurfaceView 类，有独立的 Surface（会占用较多显示内存），可在线程中获取画布进行绘图，实现快速动态绘图。该类基于 CPU 绘图，没有硬件加速能力。（2）基于 OpenGL ES 的 GLSurfaceView 类，是特殊的 SurfaceView 类，可直接在 Surface 上硬件加速渲染图形，高帧率显示图形。（3）其他视图类，如普通视图（View）、图像视图（ImageView）。这些视图共同在一个 Surface 上绘图，由根视图遍历调用所有子视图的显示函数实现 Surface 渲染。从 Android 3.0 起 HWUI 组件允许这些视图使用 GPU 的硬件加速能力，先在视图的显示列表中自动缓存绘图指令，然后使用 OpenGL ES 2.0 对绘图指令进行渲染。

Canvas 画布类提供了二维图形、图像和文本的绘制接口，在没有硬件加速的条

件下使用 Skia 图形库实现绘图接口，在具备硬件加速的条件下使用 OpenGL ES 2.0 实现绘图接口。仅在视图显示（由 HWUI 组件启用加速）和层显示时才能使用 GPU 硬件加速能力，在位图上绘图是采用软件渲染方式。

Surface 对应于显示内存区域（即 SurfaceFlinger 进程中的 Layer 对象，通常有两个缓冲区，前台缓冲区用于合成显示，后台缓冲区用于图形渲染），OpenGL ES 或 Skia 在其上渲染图形。Surface 缓存了显示内容，由 GPU 对其进行合成和动画渲染。

普通视图默认没有层，多个视图在同一个 surface 上渲染。Android 3.0 以后可以设置视图的层类型，其渲染方式见表 2-2，绘制速度由快到慢依次是硬件层、默认的显示列表方式、软件层。

表 2-2 Android 视图的层类型的渲染方式

层类型	视图启用加速	视图不硬件加速
无（默认）	在 surface 上加速显示	在 surface 上软件渲染
硬件层	在 GPU 纹理上加速渲染	在位图上软件渲染
软件层	在位图上渲染，不加速	在位图上软件渲染

2.3 本章小结

本章分析了 iOS 绘图相关框架及其关系，总结了要实现高性能交互式绘图的方法：充分利用 CALayer 的图形缓存和图形处理器的硬件加速能力，结合离屏渲染技术提高交互式绘图的显示帧率。

对于 Android，本章给出了 SWIG 的核心功能和注意点，结论是 SWIG 用于本地动态库的开发中是可行的。分析了 Android 绘图相关框架的关系和特点，总结了普通视图和 SurfaceView 在绘图速度和内存占用上的差异，为合理选择视图类型和离屏渲染技术提供了设计依据。

在后续章节将根据本章的分析结论设计绘图方案，在 iOS 和 Android 上实现高性能的交互式矢量绘图平台。

第3章　移动绘图平台的架构设计

本章根据移动设备的特点，设计了跨平台绘图内核及设备接口。为了在 iOS 和 Android 上实现矢量绘图平台，本章设计了适配器的实现方式。后面两章将基于本章内容具体阐述在 iOS 和 Android 上适配器的实现方式。

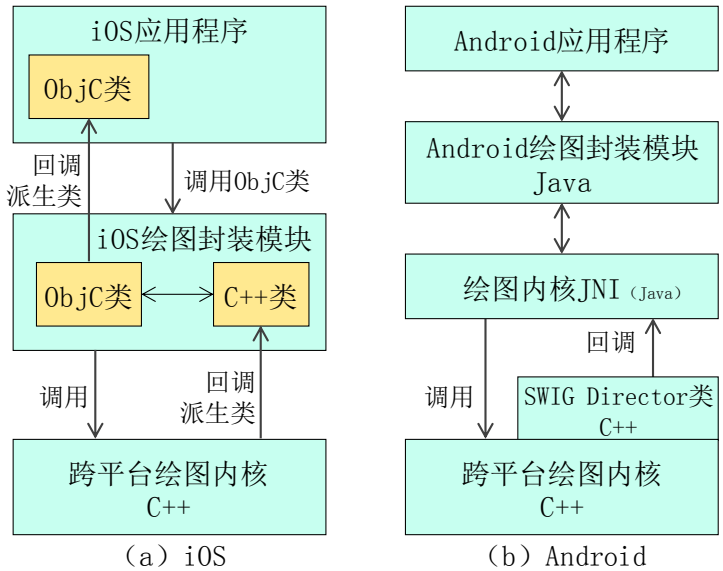
3.1　绘图应用的跨平台设计

相对于 Web 应用，本地应用通常开发成本高、难以实现跨平台。iOS 和 Android 的主要编程语言分别是 ObjC（即 Objective-C）和 Java，相互移植很困难。本文提出了基于虚函数多态行为的跨平台扩展机制。该机制将主要功能在跨平台内核中使用 C++实现，而在不同的移动平台上实现适配器模块，使其易于扩展。

3.1.1　基于多态的回调扩展机制

对于 iOS，界面封装层采用 ObjC 和 C++实现（实现文件的后缀名为.mm）。ObjC 类可以调用 C++类和全局函数，但不能从 C++类继承。国内外的相似软件通常使用 ObjC 实现所有绘图功能，不能跨平台。

本文针对 iOS 设计了如图 3-1（a）所示的扩展机制：与设备平台相关的类从绘图内核的 C++接口派生，以虚函数多态方式实现定制。其优点是跨平台内核可以调用与设备平台相关的 C++类，C++类再调用 ObjC 类。创新点是可以在跨平台内核中使用 C++实现易于扩充的功能，使用 ObjC 对内核进行扩展。



（a）iOS　　（b）Android

图 3-1 跨平台内核与设备平台的双向调用关系

对于 Android，界面封装层采用 Java 实现，在内核层与界面封装层之间提供 JNI 封装层，实现 C++到 Java 的衔接转化。通常的实现方案是使用 Java 实现所有绘图功能，很少从 C++类继承和扩展。本文设计了如图 3-1（b）所示的调用和回调机制：界面封装层通过 JNI 调用内核，内核通过 SWIG 所生成的 Director 类回调到 JNI 的 Java 类，界面封装层再从 Java 类派生，以虚函数多态方式实现定制。创新点是可在跨平台内核中使用 C++实现易于扩充的绘图功能，通过 SWIG 在 Android 上实现定制。

基于多态的回调扩展机制的创新在于实现了“一次编写，多处应用”，解决了目前在移动平台之间代码移植困难的问题。在跨平台内核中不断扩充功能不影响设备平台适配器模块，从而增强了扩展性，减少了移植工作。

3.1.2 跨平台编码的策略

跨平台内核采用 C++实现。C++适合 iOS、Android、Windows Phone、Windows Desktop 等操作系统。本文试验总结了下列跨平台编码策略：

（1）使用标准 C/C++函数库（libc、libstdc++）进行数学计算、字符串操作、动态内存管理，不使用 C++异常类型（exception）以避免跨编程语言的异常未捕获问题。

（2）避免使用各个平台的保留关键字与已有名称作为变量和函数名。例如，在 iOS 上不使用 id 作为变量名。为了在 iOS 上避免宏定义、枚举定义及变量的同名冲突，不使用命名空间来区分名称，而是加上命名前缀，对枚举值的名称以 k 开头。

（3）不使用复杂的多线程管理函数和锁类型。将多线程管理放在设备相关的模块中，少用全局变量和共享数据。使用轻量级的原子计数加减函数[②]进行访问保护。

（4）在 iOS 和 Android 上，使用泛型编程和标准模板库（STL）的容器类，不使用专用的容器等数据结构类型。

（5）因为 iOS 和 Android 不支持宽字符（wchar_t）类型，字符串默认采用 UTF-8 编码，所以在跨平台内核中采用 UTF-8 编码、char*类型。

3.2 适应多种设备的绘图模型

3.2.1 移动平台的异同分析

目前，主流的移动操作系统是 iOS 和 Android。本文通过对这两个平台的特性分析得出下列与绘图相关的结论：

[②] 采用条件编译方式实现原子计数加减函数，在 Android 上使用 __sync_add_and_fetch 和 __sync_sub_and_fetch 函数，在 iOS 上使用 OSAtomicIncrement32 和 OSAtomicDecrement32 函数。

（1）使用多点触摸手势的交互方式。单指和双指手势具有普遍适用性，设备平台已提供这些手势的识别器，不需要重新开发手势识别器。

（2）界面主要采用二维图形引擎显示。图形引擎内部可以使用 OpenGL ES 加速绘制，使用层（矩形纹理）进行显示内容的缓存和动画合成。官方推荐采用离屏渲染技术加速绘图，在层上缓存渲染内容。

（3）交互式动态绘图的刷新方式主要有两种：标记视图的无效区域，依靠消息机制延后刷新内容；在后台线程中预渲染，完成后直接提交到视图的层中。

（4）与绘图相关的设备平台差异，主要是使用了不同的编程语言、本地图形库、界面框架和消息处理方式。视图和层采用树状层次结构显示内容和传递消息事件。

3.2.2 跨平台绘图模型

根据 iOS 和 Android 的异同分析，本文提出了一种如图 3-2 所示的内核跨平台的交互式绘图模型。

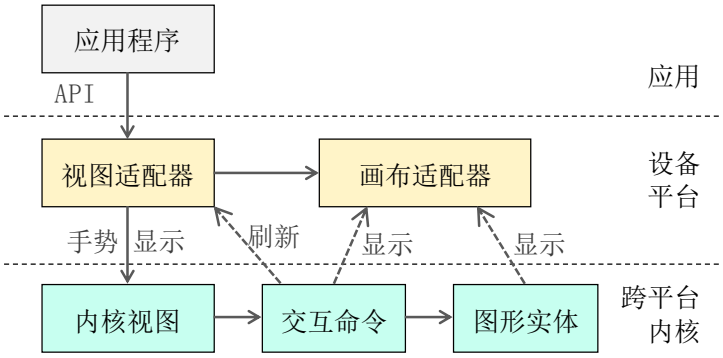


图 3-2 跨平台绘图模型

本绘图模型包含六大元素，具体描述如下。

（1）应用程序。集成绘图平台，定制界面布局和效果，应用绘图数据。

（2）视图适配器。使用界面框架实现占位视图，响应绘制和触摸消息，将显示请求和识别出的手势委托内核视图处理。允许内核以回调方式刷新内容。

（3）画布适配器。将画布接口的显示原语映射到特定图形库的绘制函数。

（4）内核视图。管理图形，将显示和手势请求转发给交互命令和图形实体。

（5）交互命令。将手势请求转换为绘图步骤，实现交互逻辑。对显示请求，以动态图形回显交互效果。扩充交互命令可实现更多绘图功能。

（6）图形实体。在内核中管理和显示图形数据，扩充类型实现更多绘图功能。

本绘图模型的关键设计点：抽象出画布接口和视图接口，采用适配器模式解决平

台差异问题，基于委托模式实现图形显示和触摸交互功能的跨平台。

3.3 总体设计

3.3.1 分层架构

为了降低模块耦合性、在不同的设备平台上提高可复用性，本论文的矢量绘图平台（TouchVG）按图 3-3 所示的分层架构设计。总体上分为两大层：跨平台绘图内核层和设备平台相关的界面封装层。在跨平台绘图内核层中实现设备平台无关的功能。在界面封装层中提供设备平台特有的通用功能，供顶层的各种应用程序使用。

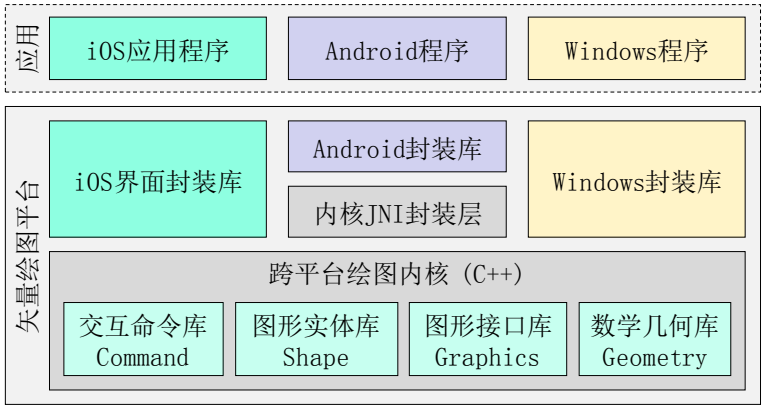


图 3-3 绘图平台的体系结构

对于 iOS，界面封装层采用 ObjC 和 C++实现。对于 Android，界面封装层采用 Java 实现。在绘图内核层与界面封装层之间提供 JNI 封装层，允许 Android 的 Java 代码与内核的 C++代码相互调用。实现相互调用的关键技术是使用 SWIG 进行编程语言的转换和集成。在 Windows 桌面应用中，可以使用 C#和 WPF 开发界面封装层及程序（也可以使用 MFC 和 GDI+实现）。SWIG 也应用在 C++与 C#之间的衔接转换上。

TouchVG 在各个操作系统上有相应的绘图平台，这些平台有相同的内核。例如，图 3-4 所示的 iOS 和 Android 绘图平台。这些绘图平台都是本地应用形式，能实现更大程度的融合和性能优化。在绘图内核中实现主要的绘图功能，通过在不同平台上编译同一代码的方式适应多种平台、避免重复开发、提高可复用性。

为 iOS 提供的绘图平台以静态库的形式供各种绘图应用程序使用，如图 3-4（a）所示。应用程序无需使用绘图内核的接口，只需使用界面封装层所提供的简易接口，从而降低了应用开发的工作量。界面封装层和应用程序都基于 iOS 平台的 SDK 开发，可以实现统一的界面风格，降低实现难度。

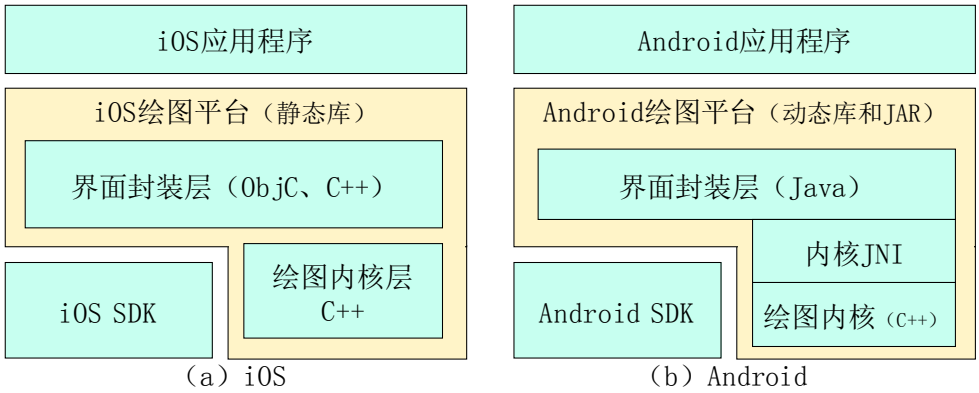


图 3-4 iOS 和 Android 绘图平台

为 Android 应用程序提供的绘图平台包含一个 JAR 包（内核 JNI 和界面封装层的 Java 类）和一个绘图内核的本地动态库，如图 3-4（b）所示。应用程序通过使用界面封装层所提供的简易接口降低了集成难度。界面封装层和应用程序都基于 Android 平台的 SDK 开发，可以最大程度地利用平台 SDK 的特性，实现紧密融合。

3.3.2 MVC 架构

TouchVG 平台采用如图 3-5 所示的 MVC 架构模式设计动态行为。

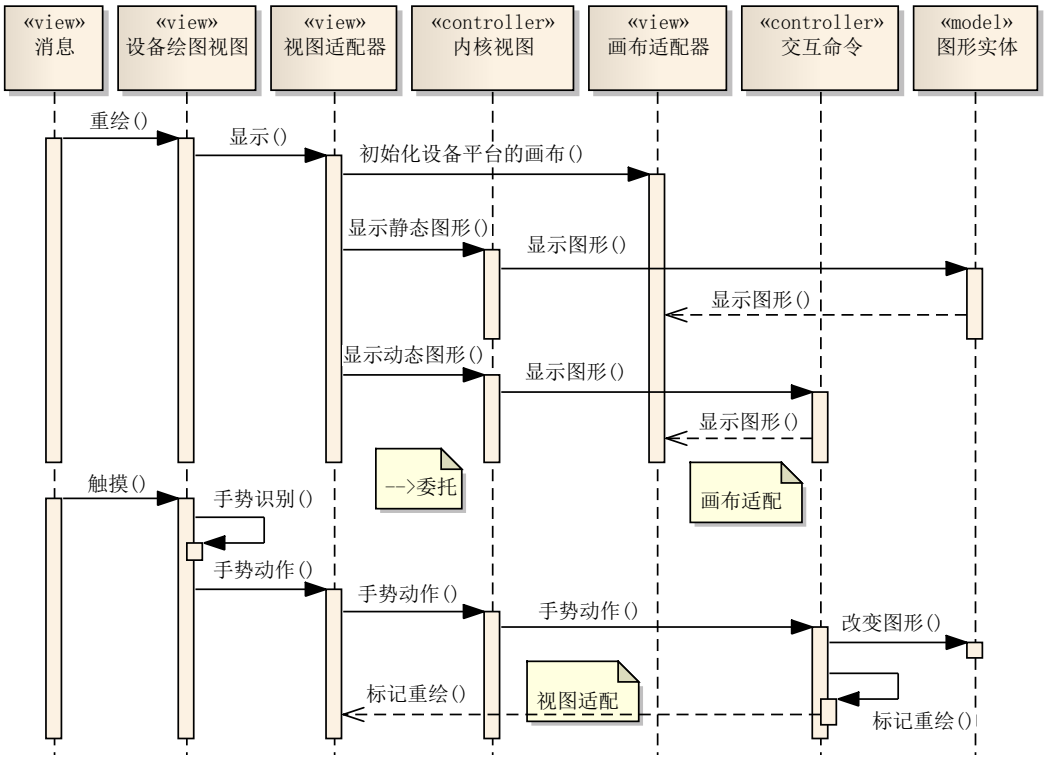


图 3-5 绘图 MVC 架构

（1）在绘图视图中响应重绘消息，使用当前的绘图上下文初始化画布适配器，

将此画布适配器以抽象画布接口对象传入内核视图，请求绘图。

（2）内核视图将静态绘图和动态绘图的请求分别转发给图形实体和当前绘图命令。图形实体和绘图命令调用画布接口对象绘图。在绘图时画布适配器将被回调。

（3）在绘图视图中响应触摸消息，识别出手势后委托内核视图处理，由内核视图转发给当前绘图命令进行交互式绘图。

（4）绘图命令根据触摸位置改变临时图形，调用抽象视图接口的重绘函数触发重绘消息，在下次重绘时显示该临时图形，从而实现动态绘图。在一次触摸结束后改变图形实体，调用视图接口的重新生成函数，这样就会显示新的图形实体内容。

3.3.3 系统组成

为各种 iOS 绘图程序提供的 TouchVG 绘图平台以静态库的形式出现。为 Android 程序提供的 TouchVG 绘图平台包含 JAR 包和绘图内核的本地动态库。TouchVG 绘图平台包含的模块如图 3-6 所示。其中，refine 表示有某个类实现了指定的接口。

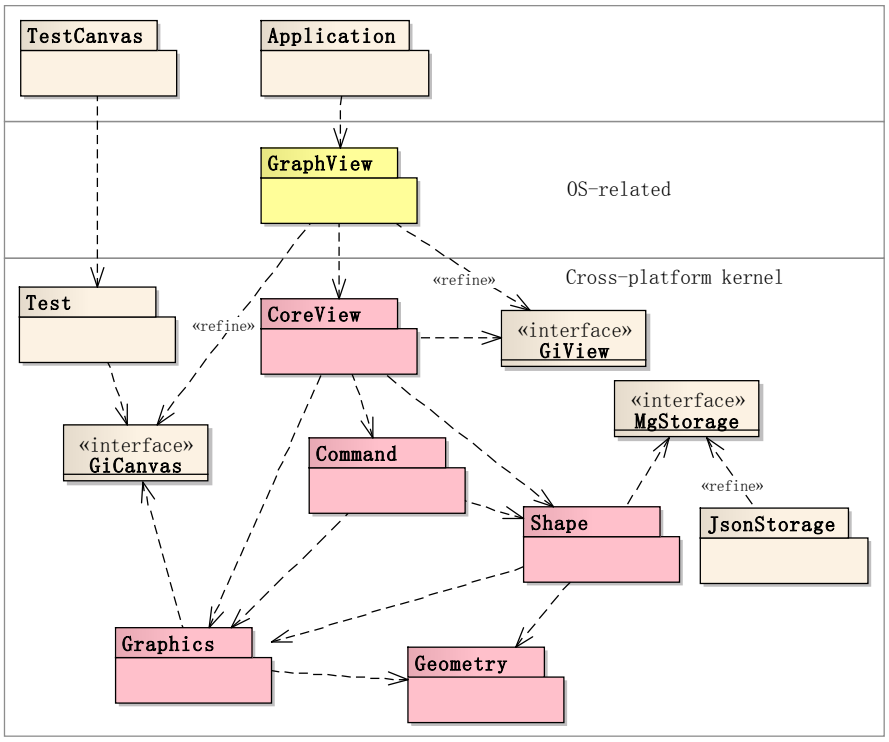


图 3-6 绘图平台的模块关系

TouchVG 有下列六个核心模块，除了 GraphView 模块外都是跨平台的模块。

（1）GraphView：设备平台相关的适配器模块，包含视图适配器和画布适配器。

（2）CoreView：内核视图模块，为上层适配器提供访问接口，管理图形数据，

将显示和手势请求转发给图形实体或交互命令。

（3）Command：交互命令模块，实现选择命令及多个绘图命令。这些命令用于绘制和修改各种图形，为上层界面提供动作消息接口以便接受触摸手势或鼠标消息数据。交互命令模块在内部将动作消息分发给相应的命令对象，实现交互式绘图。

（4）Shape：图形实体模块，主要功能是常见图形的存储、渲染和交互计算。

（5）Graphics：图形接口模块，实现显示坐标系变换、放缩平移计算、多种曲线形状的显示输出。在图形接口库中不应用任何图形库，只是提供了适合多种平台的画布接口。由界面封装层的画布适配器类使用某一种图形库来实现该画布接口。

（6）Geometry：数学几何模块，实现点、矢量、变换矩阵、矩形框、基于 Bezier 的曲线、路径、剪裁、最短距离等几何计算功能及方程组求解等数学算法。

其他模块或类：

（1）JsonStorage：用 JSON 实现的一种序列化类，实现了 MgStorage 接口。

（2）Test：跨平台的单元测试模块，供设备平台的 TestCanvas 等程序使用。

3.3.4 代码目录结构

TouchVG 平台的代码目录结构如图 3-7 所示，各个文件夹节点的含义见表 3-1。

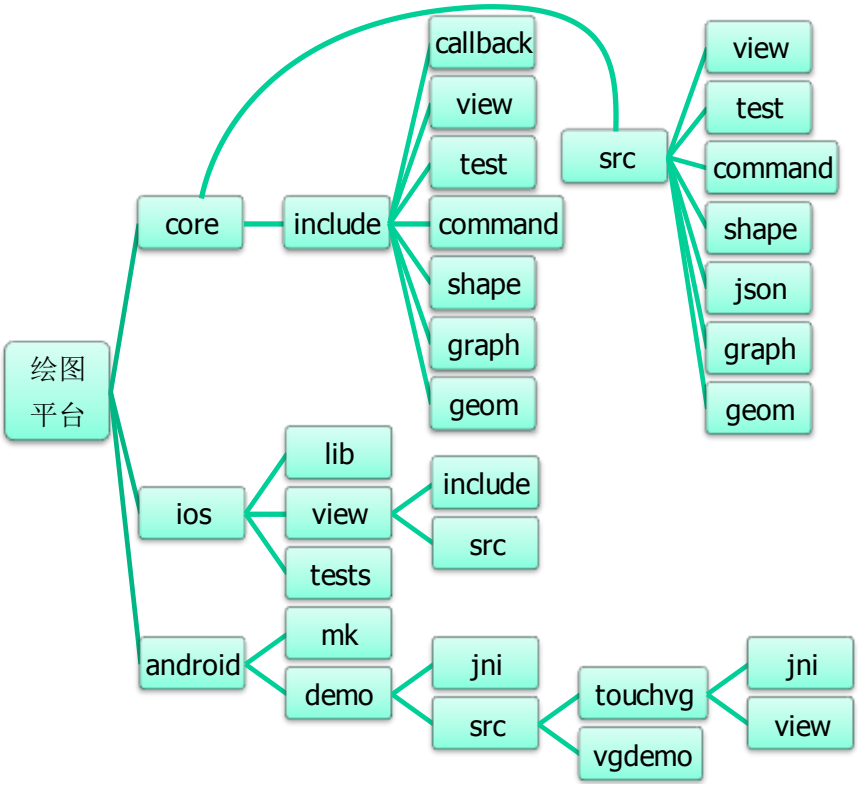


图 3-7 代码目录结构

表 3-1 代码文件夹列表

主目录	文件夹名	含义
core	callback	画布接口和视图接口
	view	供视图适配器调用的内核视图分发接口
	test	画布接口的单元测试类
	command	交互命令模块
	shape	图形实体模块
	json	JSON 序列化器模块
	graph	图形接口模块
	geom	数学几何模块
ios	view	视图适配器和画布适配器
	tests	iOS 绘图测试程序
	lib	iOS 绘图平台的静态库工程
android	mk	编译脚本
	demo/jni	本地动态库的配置文件和自动生成的源文件
	touchvg/jni	SWIG 生成的 JNI 类文件
	touchvg/view	视图适配器和画布适配器
	vgdemo	Android 绘图测试类、测试程序

3.4 跨平台绘图内核的实现方式

本节描述与设备平台显示相关的实现策略和跨平台绘图内核相关的实现方式，为在 iOS 和 Android 上实现了画布适配器和视图适配器设计接口。

3.4.1 矢量图形显示的实现方式

内核跨平台的矢量图形显示方式如表 3-2 所示。在设备平台上实现可自定义绘图的视图类（例如 MyDeviceView），响应视图的重绘消息，将绘图上下文信息传入画布适配器（例如 CanvasAdapter）进行初始化，然后将此画布适配器传入内核视图

（GiCoreView）请求绘图，内核视图再分发给相应的对象进行显示。

在内核中调用抽象画布接口（GiCanvas）的绘图函数时，画布适配器的对应实现函数将自动使用特定的图形库完成绘制。因为设备相关的视图类与显示哪些内容无关，所以在跨平台内核中扩充图形结构后不影响视图类，实现了跨平台绘图。

表 3-2 矢量图形显示实现方式的伪代码

```
void MyDeviceView::onDraw(Context ctx) { // 响应重绘消息
    CanvasAdapter canvas;
    if (canvas.beginPaint(ctx)) {        // 使用绘图上下文初始化画布适配器
        _coreView.draw(canvas);          // 向内核传入 canvas 委托绘图
        canvas.endPaint();
    }
}
class CanvasAdapter : GiCanvas {         // 实现画布接口
    virtual void drawLine(float x1, float y1, float x2, float y2) {
        _ctx.drawLine(x1, y1, x2, y2);   // 使用某种图形库实现
    }
    ......
}
class GiCoreView {                       // 内核的视图分发器
    void draw(GiCanvas canvas) {         // 由上层视图传来画布对象
        _shapes.draw(canvas);            // 委托图形实体显示
        // 在某个图形中调用 canvas.drawLine(...)，将回调到 CanvasAdapter
    }
    ......
}
```

3.4.2 交互式绘图的实现方式

交互式绘图在上述静态图形显示的基础上，根据多点触摸动作的位置信息，动态改变和显示图形。如表 3-3 所示，其执行步骤如下：

（1）在绘图视图类中识别出多点触摸手势动作，委托内核分发手势动作。在绘图视图类（例如 MyDeviceView）中利用系统内置的手势识别器从多点触摸信息中识别出某种触摸手势，然后转换手势动作参数并委托内核处理（第 8~10 行）。

（2）内核视图（GiCoreView）将手势动作分发给当前的绘图命令（第 21~27 行），由绘图命令根据触摸位置来改变临时图形的形状。

（3）绘图命令改变临时图形后，调用视图接口（如表 3-4 所示的 GiView）的 redraw 函数，视图适配器自动设置重绘区域和触发重绘消息（第 28、13~14 行）。

（4）视图类在重绘响应函数中委托内核显示动态图形（第 1~4、18~19 行），内核的绘图命令将调用抽象画布接口（GiCanvas）绘制动态图形（例如拖曳效果）。

表 3-3 交互式绘图实现方式的伪代码

```
01 void MyDeviceView::onDraw(Context ctx) { // 响应重绘消息
02    CanvasAdapter canvas;
03    if (canvas.beginPaint(ctx)) {          // 使用绘图上下文初始化画布适配器
04        _coreView.dyndraw(canvas);         // 向内核传入 canvas 委托动态绘图
05        canvas.endPaint();
06    }
07 }
08 void MyDeviceView::onTouch(TouchEvent e) {  // 响应触摸或手势消息
09     _coreView.onGesture(this, GiGestureType.kGiGesturePan,
10         GiGestureState.kGiGestureBegan, e.x, e.y); // 向内核传递手势动作
11 }
12 class ViewAdapter : GiView {              // 实现视图回调接口
13     virtual void redraw() {               // 重载 GiView 的刷新函数
14         _view.setNeedRedraw();            // 设置整个区域无效，待重绘
15     }
16 }
17 class GiCoreView {                         // 内核视图，分发器
18     void dyndraw(GiCanvas canvas) {        // 由上层视图传来画布对象
19         getActiveCommand().draw(canvas);   // 显示当前命令的动态图形
20     }
21     void onGesture(GiView view, int gestureType,
22                    int gestureState, float x, float y) {
23         _motion.view = view;
24         _motion.point = Point2d(x, y);
25         if (gestureType == GiGestureType.kGiGesturePan
26               && gestureState == GiGestureState.kGiGestureBegan) {
27             getActiveCommand().touchBegan(_motion); //传递动作到当前命令
28             // 在命令中调用 view.redraw() 触发重绘消息
29         }
30     }
31     ......
32 }
```

表 3-4 视图回调接口 GiView

函数名称	接口函数定义	对应的 GiCoreView 函数
重新构建显示	void regenAll()	void drawAll(GiCanvas& canvas)
追加显示新图形	void regenAppend()	bool drawAppend(GiCanvas& canvas)
更新显示	void redraw()	void dynDraw(GiCanvas& canvas)

根据 iOS 和 Android 的手势识别特点（例如 Android 需要自行实现放缩和旋转手势的识别算法），设计了如表 3-5 所示的绘图命令的手势原语。其中，click、doubleClick、longPress 是单指手势，touchBegan、touchMoved、touchEnded 是单指拖

动手势（使用最多，故分解为三个），twoFingersMove 对应于双指捏合、旋转和拖动手势（三种手势合并为一个手势原语是为了避免在设备平台识别具体手势类型）。

表 3-5 绘图手势原语

手势原语	接口函数定义
点击	bool click(const MgMotion* sender)
双击	bool doubleClick(const MgMotion* sender)
长按	bool longPress(const MgMotion* sender)
开始拖动	bool touchBegan(const MgMotion* sender)
正在拖动	bool touchMoved(const MgMotion* sender)
拖动结束	bool touchEnded(const MgMotion* sender)
鼠标掠过	bool mouseHover(const MgMotion* sender)
双指触摸	bool twoFingersMove(const MgMotion* sender, int state, const Point2d& pt1, const Point2d& pt2)

MgMotion 包含当前触点、起始触点、触摸动作状态和视图 GiView 对象等数据，通过 MgMotion 的 view 成员变量将设备平台的视图适配器对象传递到内核。

3.4.3 画布接口

TouchVG 平台参考 HTML5 Canvas 标准[③]设计了跨平台的画布接口，定义了表 3-6 中所列出显示原语，适合多数图形库。

（1）基于路径的矢量图形显示。

画布接口支持子路径，包含 beginPath、moveTo、lineTo、bezierTo、quadTo、closePath 及 drawPath 等 7 种显示原语。设计要点：a、避免使用数组、点等复合数据类型，没有折线和多边形等需要可变数量坐标的绘图函数，主要使用浮点数和整数等简单类型，以免在 JNI 中生成复杂的参数类型。b、坐标使用单精度浮点数而不是整数，与 iPhone 4、iPad 3 等高像素密度（PPI）的显示屏相适应。

因为矩形、线段、圆、椭圆是较简单且经常使用的图形，大多数图形库都提供了这些图形的绘制函数，所以在画布接口中提供了 drawRect、drawLine、drawEllipse 绘

[③] Canvas 2D Context：http://dev.w3.org/html5/2dcontext/ 。

图函数。其他矢量图形可以用这些路径显示原语表示。例如使用三次贝塞尔曲线表示任意角度的圆弧、四段贝塞尔曲线表示椭圆、将 B 样条曲线和三次参数样条曲线分解为连续的三次贝塞尔曲线。

表 3-6 画布回调接口 GiCanvas

显示原语	接口函数定义
设置画笔	void setPen(int argb, float width, int style, float phase)
设置画刷	void setBrush(int argb, int style)
清除区域	void clearRect(float x, float y, float w, float h)
显示矩形	void drawRect(float x, float y, float w, float h, bool stroke, bool fill)
显示椭圆	void drawEllipse(float x, float y, float w, float h, bool stroke, bool fill)
显示线段	void drawLine(float x1, float y1, float x2, float y2)
开始新的路径	void beginPath()
添加子路径	void moveTo(float x, float y)
添加线段	void lineTo(float x, float y)
添加贝塞尔曲线	void bezierTo(float x1, float y1, float x2, float y2, float x, float y)
添加抛物线	void quadTo(float cpx, float cpy, float x, float y)
闭合当前路径	void closePath()
绘制路径	void drawPath(bool stroke, bool fill)
保存剪裁区域	void saveClip()
恢复剪裁区域	void restoreClip()
矩形作为路径	bool clipRect(float x, float y, float w, float h)
区域作为路径	bool clipPath()
显示控制点	void drawHandle(float x, float y, int type)
显示图像	void drawBitmap(const char* name, float xc, float yc, float w, float h, float angle)
单行文字	float drawTextAt(const char* text, float x, float y, float h, int align)

（2）设置画笔和画刷。

包含 setPen 和 setBrush 设置函数。设计要点：a、颜色值使用整数（int argb），在 Android 上可直接应用该颜色值，在 iOS 等平台上按字节顺序提取各个颜色分量并转换为平台特有的颜色类型。b、在跨平台内核的 GiGraphics 类中保存线宽等绘图参数，自动判断和调用画布接口的 setPen 和 setBrush 函数，避免在各个设备平台的画布适配器中重复实现该功能。

（3）图形剪裁。

使用图形库的剪裁功能，包含 saveClip、restoreClip、clipRect、clipPath 函数。

（4）显示图像。

为了简化实现、隔离平台的差异性，图像在设备平台中管理，在画布接口中只使用名称来标识图像对象。drawHandle(x, y, type)显示原始大小的控制点图标等点状图像，drawBitmap(name, xc, yc, w, h, angle)在一个矩形框内显示图像。对于 drawBitmap，使用实际宽高表达图像放缩信息、角度表达图像旋转信息、图像的中心位置表达位移信息，这三者共同表达图像的矩阵变换信息，以任意角度和大小矢量化显示图像。

3.4.4 交互式绘图命令

在跨平台内核中使用命令模式和模板方法模式设计交互命令体系结构，如图 3-8 所示，每个交互式绘图命令都对应于一个命令类，支持表 3-5 所示的手势原语接口。每个命令类对应有一个唯一标识的命令名称，所有命令对象都在 MgCmdManager 命令管理器类中缓存。

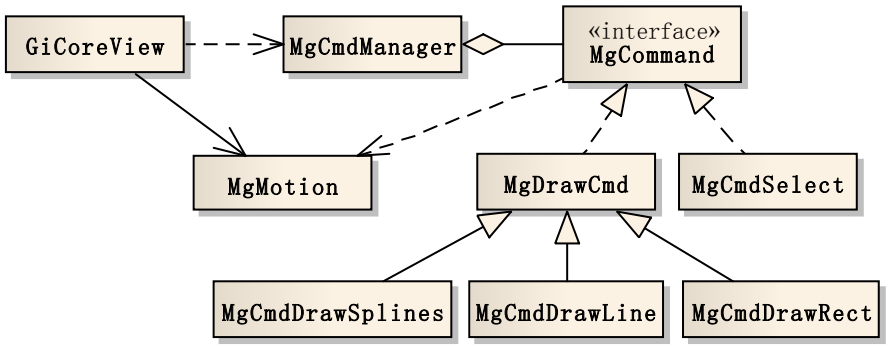


图 3-8 绘图命令体系结构示意图

内核中提供常用的基本绘图命令，外界模块可以实现更多的交互命令类，将命令名称和类工厂函数登记到命令管理器，在实际使用该命令时才创建命令对象。

在设备平台相关的模块中可以不直接访问交互命令，通过传递命令名称启动相应的交互命令（例如 `view.commandName="circle"`），由 GiCoreView 类将显示请求和手势动作转发给当前命令。

3.4.5 矢量图形的仿射变换

TouchVG 平台采用二维变换矩阵实现矢量图形的仿射变换（例如无级放缩显示、平移显示、旋转变形），使用三种坐标系：模型坐标系、世界坐标系和显示坐标系。其中，模型坐标系和显示坐标系都参照世界坐标系使用一个变换矩阵表示，分别记为 $\boldsymbol{M}$ 和 $\boldsymbol{D}$。模型坐标系到显示坐标系的变换矩阵为 $\boldsymbol{M}\times\boldsymbol{D}^{-1}$。显示坐标系的变换矩阵计算涉及下列变量：

（1）视图中心点的世界坐标（*xc*，*yc*），单位为毫米。

（2）视图显示比例（*scale*）。为 1.0 时表示按世界坐标系等比显示。

（3）视图的宽度（*width*）和高度（*height*），单位为点，与设备相关。

（4）视图显示时每英寸的点数（*dpi*），与设备相关。

按如下式（3-1）计算显示坐标系相对于世界坐标系的变换矩阵：

$$\boldsymbol{D}=\begin{bmatrix} \dfrac{25.4}{dpi\times scale} & 0 & 0 \\ 0 & \dfrac{-25.4}{dpi\times scale} & 0 \\ xc-\dfrac{width\times 25.4}{2\times dpi\times scale} & yc+\dfrac{height\times 25.4}{2\times dpi\times scale} & 1 \end{bmatrix} \tag{3-1}$$

3.5 本章小结

本章提出了一种适合多种移动平台的基于虚函数多态行为的跨平台扩展机制，将主要功能在跨平台内核中使用 C++实现，在不同的移动平台上开发适配器模块，易于扩展和实现。在 Android 上使用 SWIG 实现 C++与 Java 的调用和扩展。

通过对移动平台的差异分析，总结了 iOS 和 Android 在屏幕显示和触摸交互方式等方面的特点。提出了一种跨平台绘图模型，设计了跨平台交互式绘图内核，关键点是抽象出画布接口和视图接口，采用适配器模式解决平台差异问题，基于委托模式实现图形渲染和触摸操作功能，具备跨平台特性。

TouchVG 平台采用分层和 MVC 架构，主体功能在跨平台内核中实现。为移动平台上的适配器实现工作描述了关键实现方式、画布接口和视图接口的设计意图。

第4章 iOS 绘图平台的实现

本章阐述了 iOS 绘图平台的实现方法，主要是在跨平台内核的基础上实现 iOS 上的画布适配器和视图适配器，对图形显示优化技术进行实验研究。

4.1 基于 Quartz 2D 实现画布适配器

4.1.1 画布原语与 Quartz 2D 的映射

在 iOS 上（Xcode 开发环境）基于 Quartz 2D 图形库实现了画布适配器类 GiQuartzCanvas。该类实现了画布接口 GiCanvas，将其画布原语映射到 Quartz 2D 的绘图函数，如表 4-1 和表 4-2 所示。

这些画布原语多数直接使用 Quartz 2D 的函数（CG 开头的函数）实现，图像和文字使用了 UIKit 框架的 UIImage、UIFont 类以及 Foundation 框架的 NSString 文字显示函数。因为 UIKit 封装了图像和文字的常用功能函数，所以使用该框架能简化实现。

表 4-1 画布原语映射到 Quartz 2D

画布原语	测试号	Quartz 2D 函数映射
drawRect	b	CGContextFillRect、CGContextStrokeRect
drawEllipse	c	CGContextFillEllipseInRect、CGContextStrokeEllipseInRect
beginPath	多个	CGContextBeginPath
moveTo	多个	CGContextMoveToPoint
lineTo	e	CGContextAddLineToPoint
bezierTo	f	CGContextAddCurveToPoint
quadTo	g	CGContextAddQuadCurveToPoint
closePath	e	CGContextClosePath
drawPath	多个	CGContextDrawPath
drawHandle	i	CGContextDrawImage、CGContextConcatCTM、[UIImage CGImage]
drawBitmap	i	CGContextDrawImage、CGContextConcatCTM、[UIImage CGImage]

注：其中的测试号为图 4-4 中的测试子图号。

表 4-2 其他画布原语与 Quartz 2D 的映射

画布原语	测试号	Quartz 2D 函数映射
setPen	多个	CGContextSetRGBStrokeColor、CGContextSetLineWidth CGContextSetLineDash、CGContextSetLineCap
setBrush	多个	CGContextSetRGBFillColor
clearRect	a	CGContextClearRect
saveClip	h	CGContextSaveGState
restoreClip	h	CGContextRestoreGState
clipRect	h	CGContextClipToRect
clipPath	h	CGContextClip
drawLine	d	CGContextBeginPath、CGContextMoveToPoint CGContextAddLineToPoint、CGContextStrokePath
drawTextAt	j	[NSString drawAtPoint:]、[UIFont systemFontOfSize:]

在实现这些函数时，本文对下列内容进行了特殊处理。

（1）设置虚线模式（CGContextSetLineDash）时需要与线宽（如果大于 1）成正比例，以保持线型与形状的比例。针对手绘应用，本文将线型与线端类型按图 4-1 搭配使用：a、为了让线宽较大的短线更像一个圆点，对于实线的线型使用圆端的线端类型；b、为了让点划线等虚线类型的线条的空白间隙整齐，对于所有的虚线类型搭配使用平端的线端类型。

图 4-1 线型与线端搭配的效果

（2）使用 CGContextDrawPath 函数进行绘制并描绘闭合路径时，不能分两次调

用分别描边和填充，因为该函数执行完后会清空当前路径（CGContextClip 也一样），需要以参数 kCGPathFillStroke 或 kCGPathEOFill 同时填充和描边。

（3）使用 CGContextDrawPath 进行填充时，参数应设置为 kCGPathEOFill 或 kCGPathEOFillStroke（以奇偶规则填充），不能设置为 kCGPathFill 或 kCGPathFillStroke，否则难以在教材页面上显示如图 4-2 所示的中间透明的环形。原因是一个路径的子路径是分别填充的，为了避免环形中心被填充，需要采用 kCGPathEOFill 或 kCGPathEOFillStroke 填充路径。

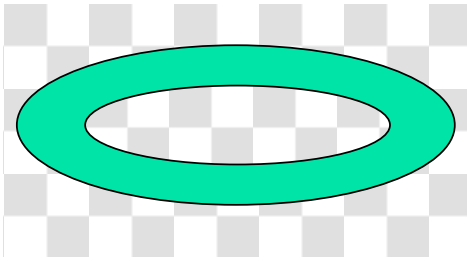


图 4-2 以奇偶规则填充环形

4.1.2 画布适配器的跨平台单元测试

本文采用测试驱动开发方式逐步实现了画布适配器，设计结构如图 4-3 所示。单元测试类 TestCanvas 是跨平台内核中的 C++类。

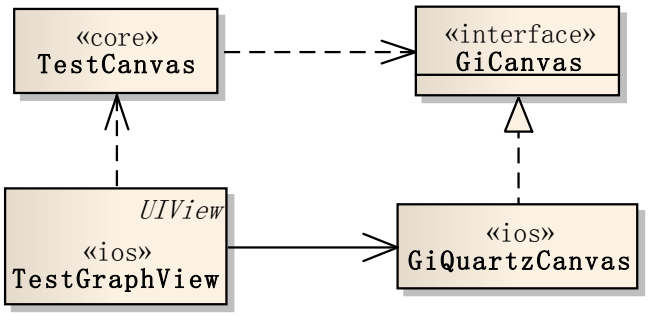
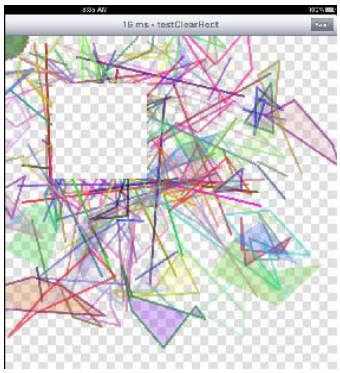


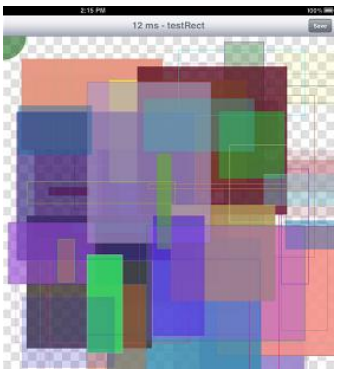
图 4-3 画布适配器的测试结构

本设计结构的创新点：（1）经实验证明在跨平台内核中可以使用 C++在 iOS 上绘图。（2）TestCanvas 类可以替换为图形实体类或绘图命令类，替换后不影响设备平台相关的画布适配器和视图类。因此，本设计结构易于扩展和移植。

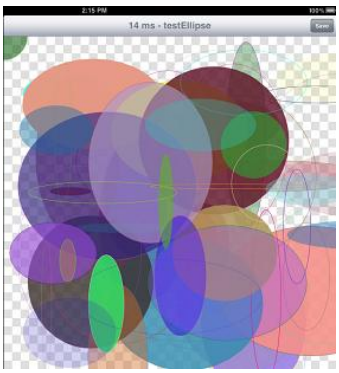
在单元测试类 TestCanvas 中使用随机函数绘制各种图形，在 iPad 上的测试图形效果如图 4-4 所示。其中，井字格的背景用于表示绘图视图为透明视图。因为通常图文批注等绘图应用需要在宿主页面上显示图形，所以将绘图视图设置为透明视图。各个子图分别为相应的画布原语的单元测试结果，对应关系见 4.1.1 节。



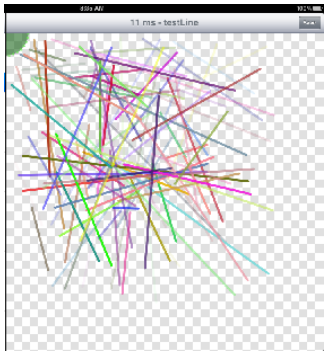
（a）testClearRect



（b）testRect

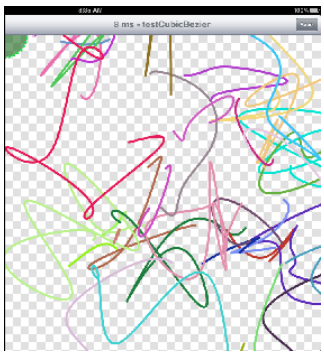


（c）testEllipse

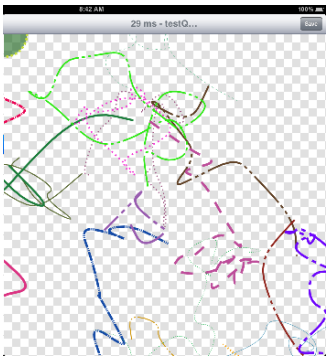


（d）testLine

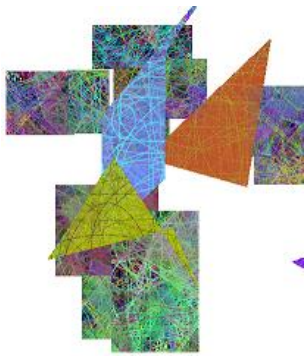
（e）testPolygon



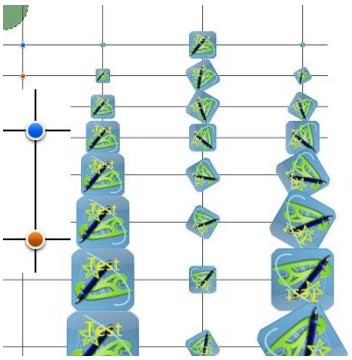
（f）testCubicBezier



（g）testQuadBezier



（h）testClipPath



（i）testHandle

（j）testTextAt

（k）GiGraphView

图 4-4 iPad 上的画布适配器的测试效果

4.1.3 图像的矢量化显示

使用 UIKit 框架的 UIImage 类显示图像，图像的显示接口函数定义为：

void drawBitmap(const char* name, float xc, float yc, float w, float h, float a)

其中，使用名称 name 标识图像对象，（xc，yc）为图像的中心显示位置，w 和 h 为显示目标宽高，a 为旋转的角度（世界坐标系中的逆时针方向）。

按照图 4-5 所示的矩阵变换过程使用 Quartz 2D 显示图像，算法过程如下：

（1）Y 上下颠倒，原点移到（xc，yc），即计算矩阵：

af = CGAffineTransformMake(1, 0, 0, -1, xc, yc) (4-1)

（2）以原点为中心旋转 a 角度，即计算矩阵：

af = CGAffineTransformRotate(af, a) (4-2)

（3）使用 CGContextConcatCTM 应用变换矩阵 af；

（4）显示图像充满到矩形 CGRectMake(-w/2, -h/2, w, h)；

（5）还原矩阵，即再应用（3）中矩阵的逆反矩阵。

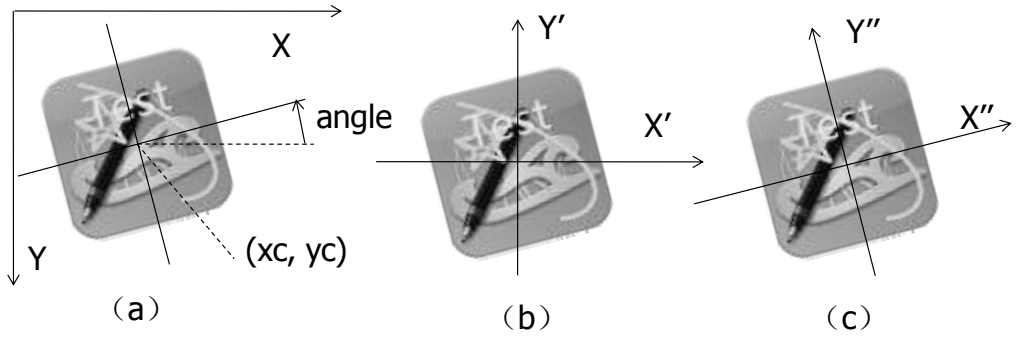


图 4-5 图像显示的矩阵变换过程

4.1.4 图像资源的管理

由应用程序添加 UIImage 对象，在 GiViewController 类（见 4.3.1 的说明）中缓存 UIImage 对象及标识名称。应用程序在切换到后台或接收到内存紧张通知时，调用绘图平台的释放缓存函数，这些图像对象就释放掉。

在需要显示时由 GiViewController 根据标识名称调用 getImageShapePath 函数从标识名称得到实际图片文件的地址，重新加载图像。应用程序可重载 GiViewController 的 getImageShapePath 函数指定不同的存放地址。

4.1.5 控制点的图像显示

iOS 绘图软件通常使用图像显示控制点，本文按下面方式实现图 4-6 的效果。

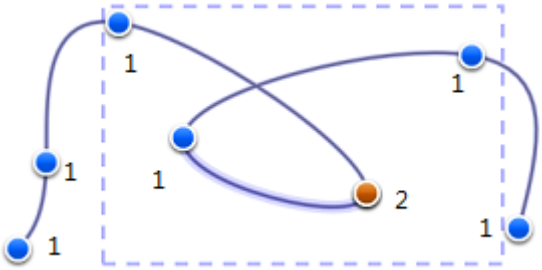


图 4-6 控制点图像显示效果

（1）在跨平台内核中调用 drawHandle 函数显示控制点，该函数的定义为：

void drawHandle(float x, float y, int type)

其中，（x，y）为控制点坐标，type 为控制点的图像类型。例如，“1”表示普通点的蓝色圆点图像，“2”表示热点的红色圆点图像。

（2）在 iOS 画布适配器的 drawHandle 实现函数中，根据 type 自动从程序资源中加载和缓存圆点图像（UIImage 对象）。

（3）因为绘图上下文的单位为点，所以将图像宽高转换到点单位，记为 w 和 h。

（4）按照图 4-7 所示的矩阵变换过程使用 Quartz 2D 显示图像，应用变换矩阵：

af = CGAffineTransformMake(1, 0, 0, -1, x – w/2, y + h/2)　　(4-3)

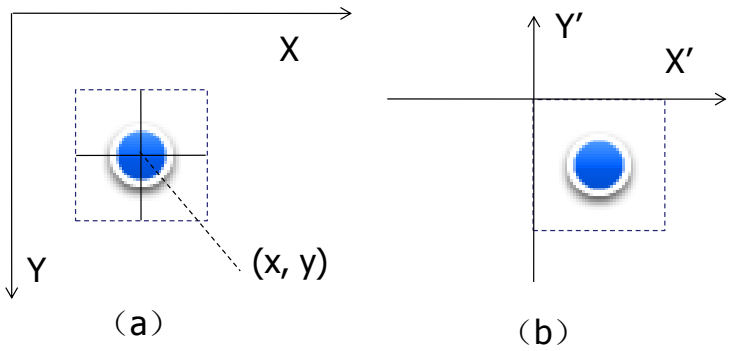


图 4-7 控制点显示的矩阵变换过程

（5）显示图像充满到矩形 CGRectMake(0, 0, w, h)。

（6）还原矩阵，即再应用（4）中矩阵的逆反矩阵。

4.2 显示优化技术研究

4.2.1 基于位图的双缓冲绘图

本节通过实验证明了基于位图的双缓冲绘图技术不适合 iOS 设备。双缓冲绘图技术是传统的绘图优化技术，主要涉及两种位图的用法：（1）缓存位图（Cached

Bitmap），是图形显示内容的快照，下次视图重绘时直接显示该位图，以免重新显示图形费时。（2）缓冲位图（Buffered Bitmap），用于避免直接在目标上下文中绘图引起的逐步显示闪烁问题，先绘制图形到缓冲位图，然后一次性复制到目标上下文。

本文使用 Quartz 2D 进行双缓冲绘图技术实验，按照是否有缓冲位图、缓存位图重建和重绘、图形规模差异进行组合实验，在 iPad 3 上的实验结果见表 4-3，其中的六列的含义如下。

表 4-3 在 iPad 3 上的双缓冲绘图时间（毫秒）

	无缓冲 重建缓存	无缓冲 4 倍图形	无缓冲 重绘缓存	有缓冲 重建缓存	有缓冲 4 倍图形	有缓冲 重绘缓存
创建位图上下文	-	-	-	0.1	0.1	0.1
清除背景	-	-	-	21	20	20
显示缓存位图	-	-	46	-	-	46
重建缓存位图	33	32	-	1	1	-
应用缓冲位图	-	-	-	64	76	64
重新显示图形	71	260	-	71	261	-
drawRect 总计	108	297	47	168	411	138

（1）无缓冲、重建缓存：直接在当前图形上下文（视图上下文）上绘制图形，缓存位图需要重新生成，当视图第一次显示或图形改变后刷新显示时属于该条件。

（2）无缓冲、4 倍图形：在上面（1）显示条件的基础上，显示 4 倍的图形量。

（3）无缓冲、重绘缓存：直接在当前视图上下文上绘制，一次性显示缓存位图，不重新显示图形。

（4）有缓冲、重建缓存：使用缓冲位图绘图，缓存位图需要重新生成，当视图第一次显示或图形改变后刷新显示时属于该显示条件。

（5）有缓冲、4 倍图形：在上面（4）的显示条件基础上，显示 4 倍的图形量。

（6）有缓冲、重绘缓存：先在缓冲位图上下文中显示有图形内容的缓存位图，不重新显示图形，然后将缓冲位图显示到视图上下文。

表 4-3 中的各个评测参数解释如下：

（1）创建位图上下文：使用 CGBitmapContextCreate 函数创建缓冲位图上下文，

需用将坐标系由默认的 LLO 坐标系改为 ULO 坐标系、计算位图宽高需用考虑到屏幕放大比例（即将视图的点单位转换为像素单位）。

（2）清除背景：使用 CGContextClearRect 函数将显示区域填充为透明背景。

（3）显示缓存位图：使用 CGContextDrawImage 函数显示已创建的缓存位图。因为 Quartz 2D 内部坐标系是 LLO 类型，图像的 Y 轴正方向朝上，显示前需用将绘图上下文的当前转换矩阵上下临时颠倒为 LLO 坐标系。

（4）重建缓存位图：使用 CGBitmapContextCreateImage 函数对绘图上下文创建一个快照图像，图像包含了各种图形的显示内容。

（5）应用缓冲位图：首先使用 CGBitmapContextCreateImage 函数从缓冲位图上下文生成快照图像，然后显示到视图上下文中，显示前将当前转换矩阵上下颠倒。

（6）重新显示图形：使用画布接口显示所有图形。

（7）drawRect 总计：以上显示工作都是在视图的 drawRect 函数中进行的，此处统计了所有显示工作的时间。

本文对表 4-3 的显示时间进行对比分析，得出下列结论：

（1）在 iOS 设备上无需使用基于缓冲位图的显示技术，直接在当前图形上下文上绘图更快，原因是 iOS 内部使用了基于矩形纹理的缓冲显示技术。

（2）清除背景较耗时。使用 UIGraphicsBeginImageContextWithOptions 函数用时约 9 毫秒，相对较快，并能自动背景透明和设置当前变换矩阵，因此在需要位图上下文绘图时要用该函数，不使用更底层的 CGBitmapContextCreateImage 函数。

（3）缓存位图较大，显示较慢。应当在重新显示图形所需时间超过缓存位图的显示时间时才使用缓存位图。可动态记录该阀值和决定是否缓存图形内容。

本文经实验发现图形数量与显示时间成如图 4-8 所示的线性比例关系。绘图上下文内显示图形前的初始化时间与图形数量无关，在 iPod Touch 4 上约为 40 毫秒。

本文对在视图上下文和位图上下文上的图形显示速度进行评测，在 iPod Touch 4 上的实验结果如图 4-8 所示。可见显示时间与图形数量成线性比例关系，在视图上下文上的显示略慢。结合 iOS 显示原理的分析结论，本文做出下列推测：

（1）两者的矢量图形显示速度接近，视图上下文没有使用硬件加速能力。

（2）在视图上下文的内部显示流程中，先将矢量图形以 PostScript 指令缓存到显示列表，然后使用 OpenGL ES 渲染这些指令，增加了图形缓存时间，所以比位图上下文略慢。缓存 PostScript 指令的优点是在动画显示和放大显示时保持高质量。

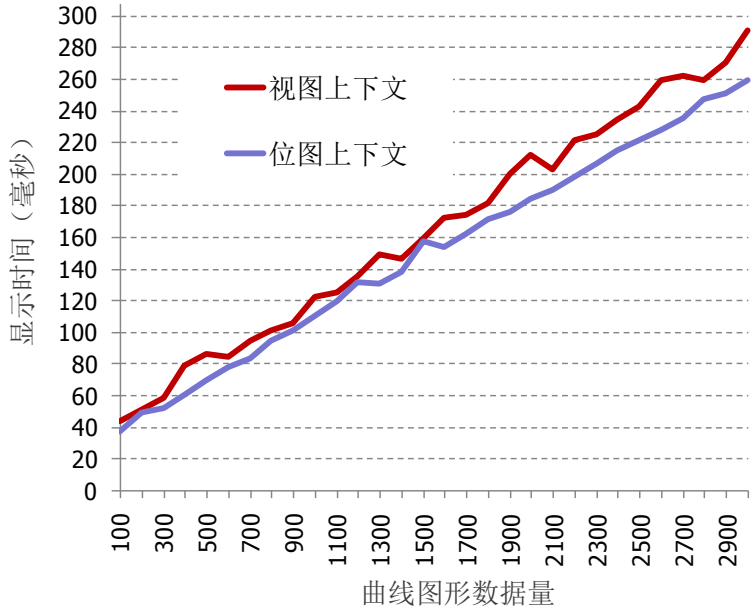


图 4-8 位图上下文和视图上下文的显示速度对比

4.2.2 快速手绘的增量绘图技术

在快速手绘原笔迹图形时，每次增加了一段笔迹图形（内部为多段贝塞尔曲线）都需要在视图显示新的内容。如果每次都全部重新显示所有图形就会越来越慢，影响快速手绘应用的回显体验。本文设计了一种增量绘图技术的实现方式，使得图形显示时间由递增趋势变为常量时间（通常能保持在 60 毫秒以下）。实现方式如下：

（1）在触摸过程中设置当前临时图形的几何参数，动态显示该图形：调用视图接口的 redraw 函数，在视图适配器的 redraw 实现函数中调用视图的 setNeedDisplay 函数，在下次系统调用视图的 drawRect 函数时调用内核的 dynDraw 函数显示图形。

（2）一次触摸完成后，在内核中将该临时图形提交到图形列表，记下新图形，然后调用视图接口的 regenAppend 函数。如果调用 regen 函数则会显示所有图形。如果触摸太快来不及显示，就记下更多的新图形，后续批量显示。

（3）在视图适配器的 regenAppend 实现函数中，先得到视图的当前快照图像（使用 [CALayer renderInContext:] 函数），然后通知视图重绘。

（4）在视图重绘消息响应函数 drawRect 中，先显示并销毁该快照图像，然后调用内核的 drawAppend 函数，由后者显示（2）中记录的新图形。

（5）继续触摸绘图，转到步骤（1），实现快速增量绘图。

本文所实现的增量绘图技术的关键点在于每次新增图形后只需要显示快照图像和新增图形，无需显示之前已有图形。以 iPad 3 为例，使用 renderInContext 生成快照

图像约 36 毫秒，显示快照图像约 46 毫秒，生成和显示是在不同的消息响应函数中进行的，整体显示很流畅，不受图形数量影响。

4.2.3 快速动态绘图的多层绘图技术

动态绘图的过程：在触摸过程中改变临时图形，调用视图接口的 redraw 函数，由视图适配器设置视图无效区域并触发重绘消息，在重绘消息响应函数中显示这些临时图形，最终实现了及时回显的交互式动态绘图。

在一个视图中同时显示静态图形和动态临时图形的问题是动态绘图相对于当前触摸位置有短暂延迟，会产生拖尾现象。虽然可以使用快照图像避免重新显示所有图形，但也需要几十毫秒进行图像显示。

本文设计了如图 4-9 所示的多层视图结构，将静态图形和动态图形分别在单独的视图中显示。静态图形视图是应用宿主视图中的子视图，设置为背景透明以便显示出阅读器等应用宿主视图中的内容。动态图形视图与静态图形视图大小相同，动态图形视图通常在所有视图的顶端。因为 iOS 中每个视图都有独立的层，能够独立绘制，所以分离视图后可以让动态图形视图不显示静态图形内容，改善回显体验。

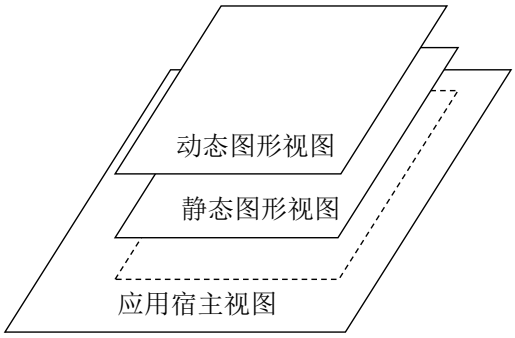


图 4-9 绘图视图的层次结构

从静态图形视图获取快照图像（例如用于预览）是较常用的功能。本文比较了下列获取快照的方法，结论是调用层的 renderInContext 函数是最快的方法。

方法 1：在位图上下文中绘制所有图形，与在视图显示图形的时间相近，慢。

方法 2：在位图上下文中调用层的 drawInContext 函数，比方法 1 慢 10 毫秒。

方法 3：在位图上下文中调用视图的层的 renderInContext 函数，最快，与图形数量无关。例如，在 iPad 3 上用时 36 毫秒，在 iPod Touch 4 上用时 11 毫秒。与表 4-3 对比还能得出结论：renderInContext 比显示缓存位图更快。

对静态图形视图调用层的 renderInContext 函数会得到该层及所有子层的快照图

像，包含了动态临时图形的显示内容，不满足实际需要。为了仅得到静态图形视图的显示内容，本文的解决方法是将这两类图形所在的视图设置为应用宿主视图的同级子视图。但在静态图形视图中响应触摸消息进行绘图时发现一个问题：第一次触摸能正常工作，后续触摸不能工作。本文试验了两种方法解决该问题：一是在动态图形视图中也响应触摸消息并转发给静态图形视图；二是将动态图形视图设置为禁止交互（userInteractionEnabled = NO）。第二种方法更简单。

4.2.4 动态绘图的参数优化

由于动态交互式绘图需要频繁更新显示内容（静态图形较少更新），提高动态绘图的性能就比较关键。本文除了上述显示优化技术外，还针对一些关键的绘图参数进行了优化分析，在 iPad 2 上显示 100 个椭圆的测试结果见图 4-10。

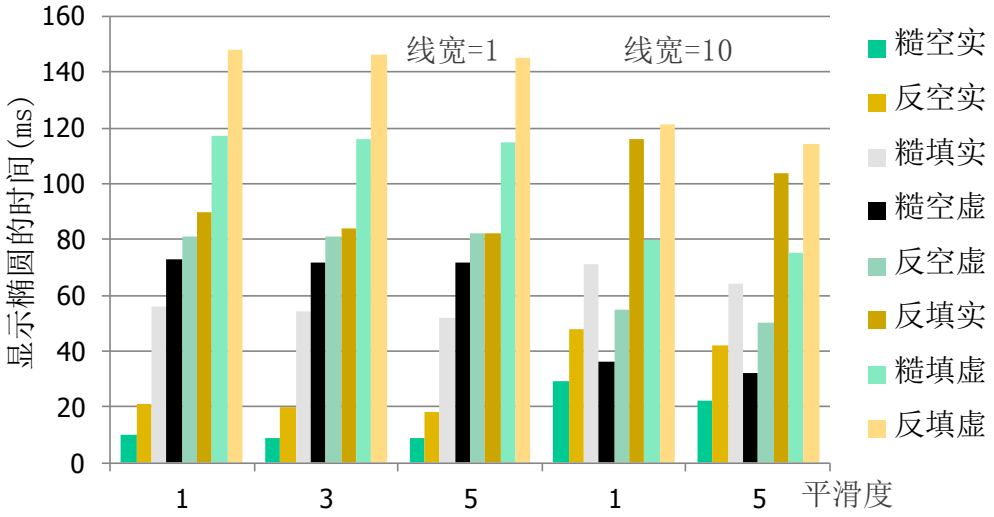


图 4-10 绘图参数对显示时间的影响

图 4-10 中的测试参数有：

（1）平滑度。Quartz 2D 内部的拟合折线与实际曲线的最大偏差距离，屏幕像素值，小于 1 时高精度渲染、费时。

（2）线宽。Quartz 2D 采用填充方式实现线宽效果，一个图形内部的重叠部分不会重复填充。

（3）反走样（糙，不反走样。反，反走样）。iOS 在单独的离屏位图上对图形渲染实现反走样效果。

（4）填充（空，不填充。填，填充）。在闭合形状的区域中填充单一颜色。

（5）线型（实，实线。虚，虚线）。指定线条阵列描绘形状路径。

从图 4-10 可以得到下列结论：

（1）平滑度小于 3 像素时较费时，超过 3 后显示时间变化较小、影响美观性。

（2）线型影响较大，虚线的显示时间是实线的两倍以上。

（3）反走样所需时间是不反走样的接近两倍。

（4）填充对显示性能影响较大，显示时间是不填充的两倍以上。

（5）线宽影响较小。

因此，平滑度可以设置为 3，快速显示时不使用虚线等线型、不反走样、不填充能显著加快显示速度。可以采用逐步渲染技术（例如，在子线程中分精度等级渲染、先批量描边后批量填充）提高响应速度。

4.3 iOS 绘图平台的结构

4.3.1 静态结构

根据 4.2 节显示优化技术的研究结果，iOS 绘图平台按图 4-11 设计静态类结构（省略了跨平台内核的结构），相应类的说明如下。

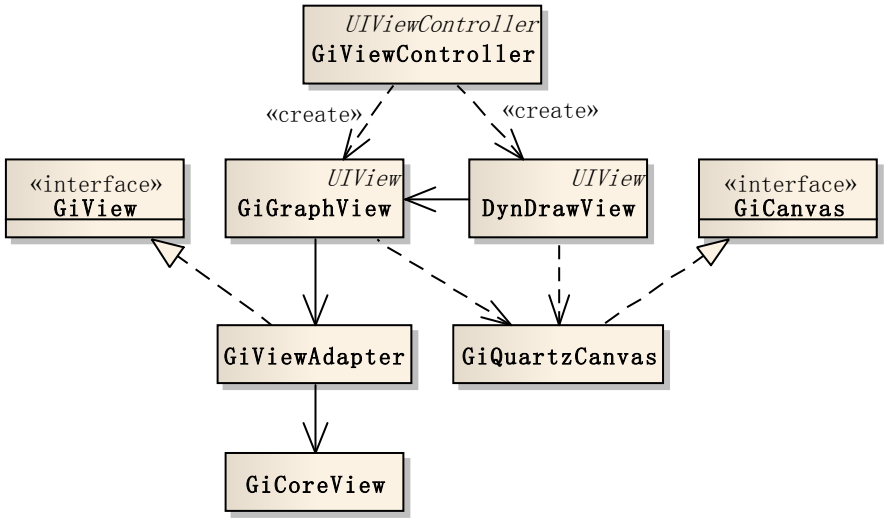


图 4-11 iOS 绘图适配模块的结构

（1）GiViewController。面向应用程序的绘图封装接口类，提供常用 API，从 UIViewController 派生。

（2）GiGraphView。显示静态图形的视图类，负责触摸手势识别，委托内核的 GiCoreView 实现图形显示和手势操作。

（3）DynDrawView。显示动态图形的视图类，不负责触摸手势识别，委托内核的 GiCoreView 显示动态图形。

（4）GiViewAdapter。视图适配器，允许内核回调 iOS 视图，通知刷新显示。

（5）GiQuartzCanvas。使用 Quartz 2D 实现的画布适配器。

（6）GiCoreView。跨平台内核的视图分发器，托管图形对象，分发显示请求和手势信息给图形列表和当前命令。

4.3.2 应用效果

在 iOS 绘图平台中应用多层绘图技术分离 GiGraphView 和 DynDrawView 视图，提高了动态交互式绘图的回显速度。在 GiGraphView 视图中应用增量绘图技术，连续绘制自由曲线等图形时不出现明显的拖尾现象。在旋转屏幕和动态放缩过程中应用绘图参数优化技术，提高显示反馈速度。采用这些技术后，绘图体验较流畅。

在跨设备平台的内核中使用绘图命令可以显示各种图形，在内核视图中使用仿射变换实现放缩显示，图 4-12 展示了实际绘图效果[④]。

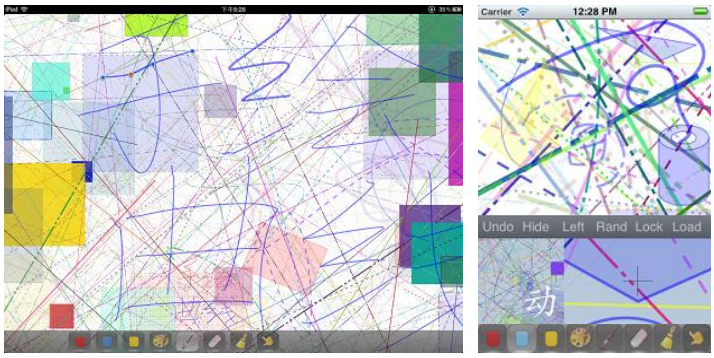


图 4-12 iOS 综合绘图效果

目前，iOS 绘图平台已在数字教育等领域应用，降低了应用开发的工作量。图 4-13（a）和（b）展示了在阅读器页面上进行批注式教学的效果。图 4-13（c）展示了第三方公司基于 TouchVG 开发的“脑力风暴”iPad 软件的图文笔记效果[⑤]。

（a）全屏批注　　（b）局部批注　　（c）图文笔记

图 4-13 绘图平台在数字教育等领域的应用效果

[④] 手绘曲线采用三次参数样条曲线模型，本文对曲线模型和数据点采样法不做研究和论述。

[⑤] 已在 AppStore 发布，地址为 https://itunes.apple.com/us/app/nao-li-feng-bao/id565836136?mt=8 。

4.4 本章小结

本章描述了基于 Quartz 2D 实现画布适配器的方式，实现了图形和图像的矢量化显示，针对手绘应用设计了虚线模式和线端类型的匹配规则。画布适配器的单元测试使用了跨平台内核自动绘制图形，证明了在跨平台内核中可以使用 C++在 iOS 上交互式绘图。

本章对双缓冲技术进行了实验，结论是基于缓冲位图的显示技术不适合 iOS 等移动设备，需要根据时间阀值动态决定是否使用缓存位图。总结了图形数量与显示时间的线性比例规律。

本章设计了适合连续手绘的增量绘图技术的实现方式和快速动态绘图的多层绘图技术的实现方式，通过利用缓存位图和层加快了交互式绘图的回显速度，总结了绘图属性的优化方法。

最后，描述了 iOS 绘图平台的结构和应用效果。

第5章 Android绘图平台的实现

本章阐述了 Android 绘图平台的实现方法，主要是在跨平台内核的基础上实现 Android 画布适配器和视图适配器，对图形显示优化技术进行了实验研究。

5.1 开发环境

5.1.1 SWIG的工作原理分析

如图 5-1 所示，借助于 SWIG 实现 Android 程序的 Java 代码通过 JNI 访问 C++的类。在编译阶段 SWIG 工具从 C++生成 JNI 的 Java 类文件和相应的 C++实现文件，该实现文件与原有的 C++实现文件一起通过 NDK 编译为本地动态库。

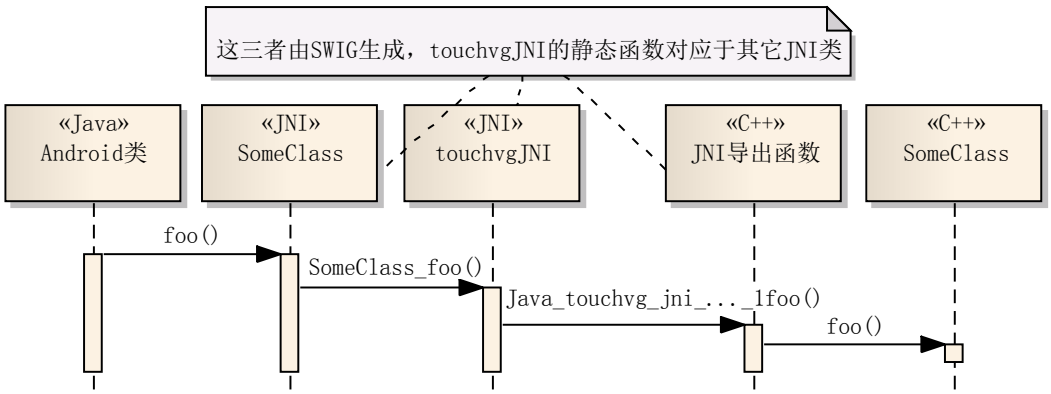


图 5-1 Android 程序调用 C++类的原理

如图 5-2 所示，利用 SWIG 的 Director 特性，指定某个具有虚函数的 C++类可重定位，然后再生成 C++导出函数文件和 JNI 的 Java 类文件。在应用层中从对应的 JNI 类继承并实现其函数，在执行该 C++类的虚函数时对应的 Android 函数就被执行。

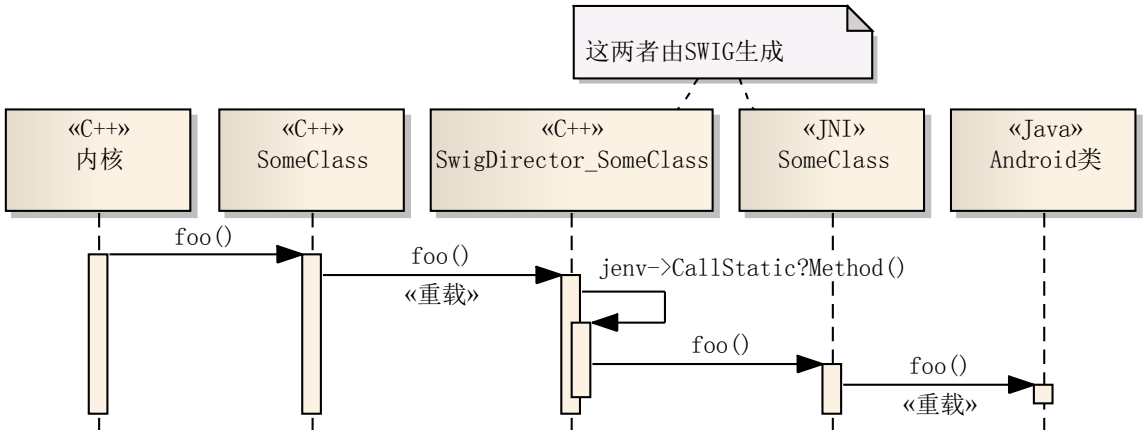


图 5-2 Android 类从 C++类的虚函数重载的原理

图 5-2 中，SwigDirector_SomeClass 从 C++的 SomeClass 派生，Android 类从 JNI 的 SomeClass 类派生，在 SwigDirector_SomeClass 中通过调用 JNIEnv 类的 CallStaticVoidMethod 等函数实现在 C++中调用 Java 的类函数，这样 Android 类中相应的重载函数便得到调用，实现使用 Android SDK 的 Java 类来扩展 C++类。

5.1.2 SWIG 的运行性能分析

TouchVG 平台使用 SWIG 实现 Android 程序的 Java 类与内核的 C++类之间的双向调用，即 Java 类通过 JNI 调用 C++类、C++类利用 Director 特性回调 Java 类。

本文对这两种调用方式进行评测，结果见图 5-3。

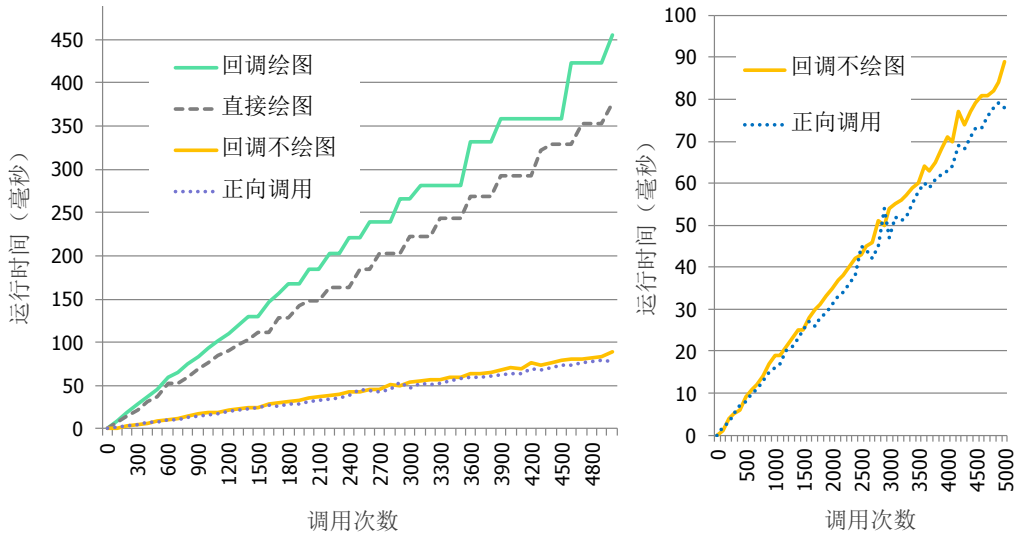


图 5-3 SWIG 在 Android 中的性能评测结果

图 5-3 包含下列四个评测项目：

（1）回调绘图：Android 程序通过 JNI 调用跨平台内核的一个测试函数，在该测试函数中多次回调画布适配器的绘制直线段的函数，在该绘制函数中使用 android.graphics 包绘图。其性能影响因素有 JNI 调用、回调和绘图。

（2）回调不绘图：与上一项目的差别是将画布适配器的绘制函数改为空实现，不受图形库的影响。其性能影响因素有 JNI 调用和回调。

（3）直接绘图：Android 程序直接调用绘制直线段的函数，与 JNI 无关。

（4）正向调用：Android 程序通过 JNI 多次调用跨平台内核的一个测试函数。其性能影响因素是 JNI 调用，与 JNI 回调及绘图无关。

评测结果表明，基于虚函数重定位技术的回调方式的性能与普通的 JNI 调用方式的差别较小，SWIG 所增加的封装函数并不会使绘图性能明显下降。

5.1.3 开发方式

Android 绘图平台的实现方式如图 5-4 所示，编译得到的绘图平台 JAR 包和内核本地动态库可供应用程序使用。借助于 SWIG 的 Director 机制，使用 Android SDK 实现了画布适配器和视图适配器，实现对内核功能的扩展。

将跨平台内核使用 NDK 编译到本地动态链接库中，接口形式为 JNI 和封装类库。采用 SWIG 将 C++类转换为 JNI 的 Java 类，SWIG 所生成的 C++导出函数文件与跨平台内核的代码文件一起编译为动态库。编译过程中使用 Python 脚本自动修正 SWIG 所生成的代码中的缺陷，并自动将包含中文字符的文件由 UTF8 临时转换为 GBK 编码以便正常编译转换。

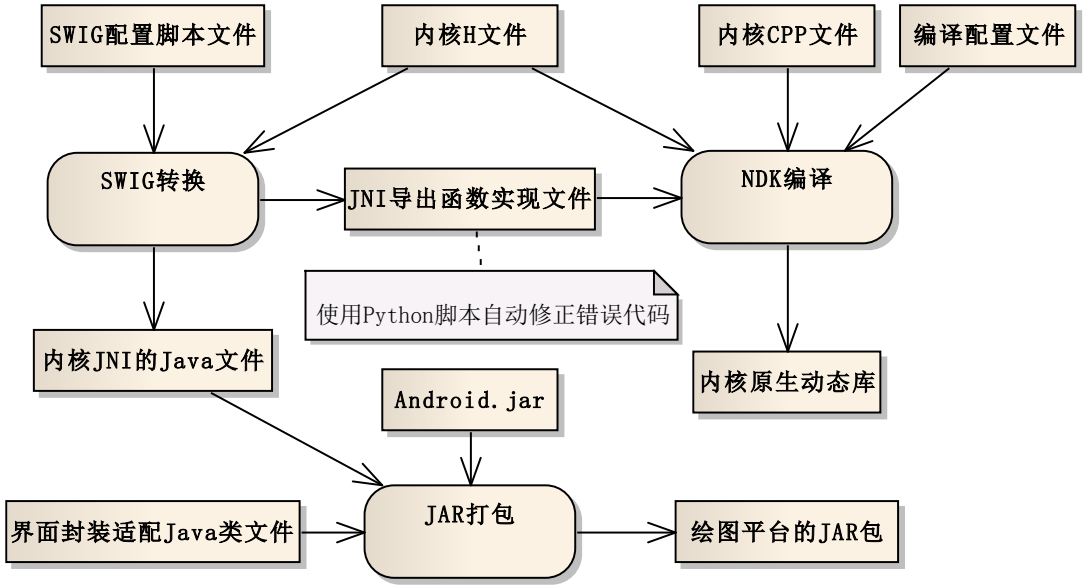


图 5-4 Android 绘图平台的实现方式

对于 Android 本地动态链接库的调试定位难题，利用 NDK 提供的日志输出 C 函数和库文件，通过输出日志文字的方式来解决。用该方法诊断出了 SWIG 引起的 JNI 内存问题所在位置，最终采用 Python 脚本自动修正 SWIG 所生成的文件缺陷。

5.1.4 开发工具

使用了下列工具分别在 Mac OS X 10.7 和 Windows 7 上开发 Android 绘图平台：

（1）Android 开发包（the ADT Bundle）r21.1，可以在 Eclipse 中调试本地动态库。

（2）Android NDK r8e，用于开发本地动态库。

（3）SWIG 2.0.10，用于从 C++头文件生成 JNI 的类文件和 C++封装文件。

（4）Python 2.7，用于运行 Python 脚本自动修正 SWIG 所生成的文件缺陷。

（5）MSYS（Minimalist GNU for Windows）1.0，用于在 Windows 上模拟 UNIX 环境，执行 Shell 编译脚本。

5.1.5 SWIG 编译配置

在 touchvg.swig 文件[⑥]中配置 SWIG 编译选项，主要配置内容如下：

（1）在文件前面定义下面两个宏：

```
SWIG_JAVA_NO_DETACH_CURRENT_THREAD
SWIG_JAVA_ATTACH_CURRENT_THREAD_AS_DAEMON
```

定义前者以便在每次调用本地代码后不与当前线程断开（使用 SWIG 的 Director 机制后，在本地函数调用结束时一些 JNI 对象还需要继续有效，不能与当前线程断开）。定义后者将 JNI 环境附加在守护线程上（默认是附加在界面主线程上，在 Activity 退出时可能会崩溃）。

（2）输出 JNI_OnLoad 函数。Dalvik 虚拟机要求必须实现 JNI_OnLoad 函数，本平台仅简单返回 JNI_VERSION_1_6，由 SWIG 生成的代码自动注册本地函数。

（3）指定 GiCanvas 和 GiView 需要生成重定向类，并导出相应的头文件。

（4）输出要在 Android 代码中使用的内核接口。例如，GiCoreView 类。

（5）添加 TmpJOBJ 辅助类，在析构函数中自动释放 JNI 本地引用对象。SWIG 生成的 Director 类中某些形参的本地引用对象没有释放，会因超出 256 个 JNI 引用对象的限制而溢出崩溃。例如，在 GiCanvas 的 drawBitmap 函数中，name 字符串对象所对应的本地引用对象“jstring jname”在调用了 NewStringUTF 函数后没有调用 DeleteLocalRef 函数。

本文针对该问题提出的解决方法：将 SWIG 所生成的封装文件中的“jstring jname = 0”替换为“jstring jname = 0; TmpJOBJ jtmp(jenv, &jname)”，通过 TmpJOBJ 的析构作用自动调用 DeleteLocalRef 函数释放引用。使用 Python 脚本[⑦]自动进行替换 SWIG 生成的封装文件中的这类问题。

编写了 Shell 脚本（mk/swig.sh），用于运行 SWIG 工具生成 JNI 导出函数的封装文件（touchvg_java_wrap.cpp）和 JNI 类文件。JNI 类的包名为 touchvg.jni，其文件输出到工程的 src/touchvg/jni 目录下，将与视图适配器的代码（src/touchvg/view 目录）共同生成为一个 JAR 文件。

⑥ 详细的 SWIG 编译选项见文件：https://raw.github.com/rhcad/vglite/master/android/demo/jni/touchvg.swig 。
⑦ Python 脚本见文件：https://raw.github.com/rhcad/vglite/master/android/demo/jni/replacejstr.py 。

5.1.6 NDK 编译配置

Android 绘图平台的代码目录结构见第 19 页的图 3-7。在工程的 jni/Android.mk 文件[⑧]中配置本地动态库的 NDK 编译选项，主要有：

（1）基于绝对地址$(LOCAL_PATH)/../../../core/include 在 LOCAL_C_INCLUDES 中指定内核的头文件路径，基于相对地址 ../../../core/src 在 LOCAL_SRC_FILES 中指定跨平台内核的实现文件（*.cpp，使用绝对的路径无法编译）。

（2）因为 SWIG 的 Director 代码使用了 RTTI 运行时类型信息，所以在 LOCAL_CFLAGS 中指定-frtti 选项。

（3）为了使用 STL，在 jni/Application.mk 中指定“APP_STL := stlport_static”。

本文编写了 Shell 脚本（ndk.sh），在其中进入 android\demo\jni 目录自动运行 ndk-build 编译出本地动态链接库 libtouchvg.so，在编译过程中自动应用 Android.mk 中的配置信息。Onur Cinar[33]介绍了在 Android.mk 中包含脚本的方法，可自动运行脚本。

本文在多个平台编译时发现 Shell 脚本文件应使用 Unix 行结束符（LF），不能是 DOS 结束符或 Mac 结束符，尽可能避免使用中文字符。

5.2 基于 Android Canvas 实现画布适配器

在第 11 页的 2.2.3 节介绍了 Android 二维绘图主要涉及的框架。本文主要基于两种视图类设计绘图视图类：android.view.View 和 android.view.SurfaceView，在绘图视图类中使用 Android Canvas 画布类（使用 android.graphics 包）渲染。

5.2.1 画布原语与 Android Canvas 的映射

本文基于 android.graphics 包设计画布适配器类 touchvg.view.CanvasAdapter，该类实现 touchvg.jni.GiCanvas 中的画布原语函数，后者是通过 SWIG 从跨平台内核的 GiCanvas 接口自动生成的。在内核中调用画布接口 GiCanvas 的函数时，画布适配器将被回调执行，从而允许使用 Android Canvas 渲染。

画布适配器主要使用了 android.graphics 包中这些类：Canvas 画布类访问绘图函数接口，Paint 类指定颜色等绘图属性，Path 类构建路径，Bitmap 指定位图数据。在绘图视图的 onDraw 函数中将 Canvas 画布对象传入画布适配器，后续绘图将在该画布对象上进行。在离屏位图上渲染时，从位图构建画布对象，接着传入画布适配器。

⑧ NDK 编译配置文件见：https://raw.github.com/rhcad/vglite/master/android/demo/jni/Android.mk 。

因为 Paint 对象只能指定一个颜色，无法区分画笔颜色和画刷颜色，所以画布适配器针对画笔、画刷和文字显示分别使用一个 Paint 对象：mPen、mBrush、mTextPen。以显示一个红边蓝底的椭圆为例，先设置 mPen 的颜色为红色、mBrush 的颜色为蓝色，然后分别使用 mPen 和 mBrush 作为参数绘制椭圆。为了让文字颜色和图形颜色同步，在 setPen 函数中同时设置画笔 mPen 和文字属性 mTextPen 的颜色。

这三种 Paint 对象的参数设置见表 5-1。

表 5-1 Paint 对象的参数设置

画笔	画刷	文字
mPen.setAntiAlias(true)		mTextPen.setAntiAlias(true)
mPen.setDither(true)		mTextPen.setDither(true)
mPen.setStyle(STROKE)	mBrush.setStyle(FILL)	
mPen.setPathEffect(null)	mBrush.setColor(0)	
mPen.setStrokeCap(Cap.ROUND)		
mPen.setStrokeJoin(Join.ROUND)		

表 5-1 中，画刷默认填充颜色为透明色，即不填充。画笔的默认线型为实线，线端为圆端，这样在绘制短线时更像一个圆点。为了让点线等虚线类型的空白间隙整齐，在 setPen 函数中对所有虚线类型设置平端的线端类型。

与 iOS 绘图平台的实现类似，Android 画布适配器按表 5-2 所示的映射方法实现了画布原语函数。由跨平台内核中的 TestCanvas 类生成和显示矢量图形和图像。

在实现这些函数时，本文对下列内容进行了特殊处理或总结。

（1）在 View 中调用画布适配器的 clearRect 函数，无法使指定区域透明，如图 5-5（i）所示。只能在原有图形基础上填充颜色，指定透明色将填充为黑色。在 SurfaceView 中调用画布适配器的 clearRect 函数，可以擦除指定区域内的图形，变为透明区域，如图 5-5（k）所示。

（2）当程序和视图使用了硬件加速特性后，调用 clipPath 会崩溃，其原因是在硬件加速时不支持 clipPath 函数。解决方法是在 UnsupportedOperationException 异常出现后将对应的视图的层类型设置为软件实现方式（LAYER_TYPE_SOFTWARE）。

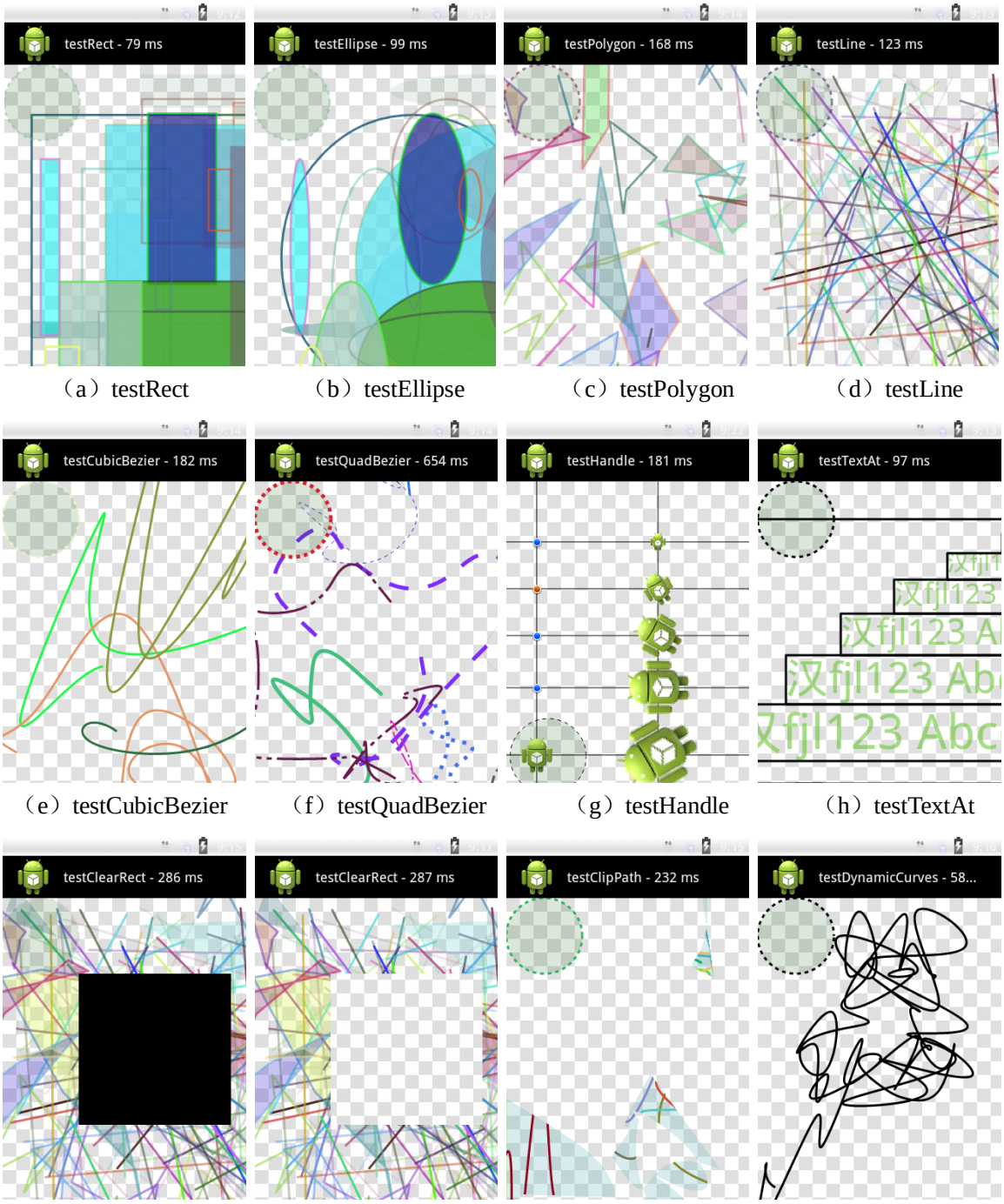
（3）在 SurfaceView 视图中使用渲染线程连续绘图能够达到 48~56FPS 的更新速

度，实验效果如图 5-5（n）所示。测试用例为绘制不断延长的三次贝塞尔曲线，测试条件为 MOTO MZ606 平板电脑（Android 4.0.3，1280×800）。

表 5-2 画布原语与 android.graphics 的映射

画布原语	测试号	Android 函数对应关系
clearRect	i k	mCanvas.drawColor(mBkColor, Mode.CLEAR)，需要设置剪裁区域
drawRect	a	mCanvas.drawRect，使用 mPen 和 mBrush
drawEllipse	b	mCanvas.drawOval，宽高不超过 1 时使用 drawPoint
beginPath	多个	创建路径对象 mPath
moveTo	多个	mPath.moveTo
lineTo	c	mPath.lineTo
bezierTo	e	mPath.cubicTo
quadTo	f	mPath.quadTo
closePath	c	mPath.close
drawPath	多个	mCanvas.drawPath(mPath, mBrush)、.drawPath(mPath, mPen)
drawHandle	g	mCanvas.drawBitmap，在指定点显示
drawBitmap	g	mCanvas.drawBitmap，指定 Matrix 矩阵变换对象
drawTextAt	h	mTextPen.setTextSize、mCanvas.drawText，用到 FontMetrics
setPen	多个	mPen.setColor、mPen.setStrokeWidth、mTextPen.setColor mPen.setPathEffect、mPen.setStrokeCap
setBrush	多个	mBrush.setColor
saveClip	m	mCanvas.save(CLIP_SAVE_FLAG)
restoreClip	m	mCanvas.restore()
clipRect	m	mCanvas.clipRect
clipPath	m	mCanvas.clipPath，硬件加速时需要将视图的层改为软件实现类型
drawLine	d	mCanvas.drawLine

注：其中的测试号为图 5-5 中的测试子图号。



（a）testRect （b）testEllipse （c）testPolygon （d）testLine

（e）testCubicBezier （f）testQuadBezier （g）testHandle （h）testTextAt

（i）clearRect in View （k）clearRect in SurfaveView （m）testClipPath （n）dynamicCurves

图 5-5 Android 画布适配效果

5.2.2 图像的显示和管理

在绘图视图中管理图像对象，画布适配器从视图获取图像。显示接口函数为：

void drawBitmap(String name, float xc, float yc, float w, float h, float angle)

其中，使用名称 name 标识图像对象，（xc，yc）为图像的中心显示位置，w 和 h 为

显示目标宽高，angle 为旋转的角度（世界坐标系中的逆时针方向）。

图像绘制的基点在图像的左上角，画布的坐标系为 ULO 类型，绘制过程为：

（1）根据 name 从绘图视图获取 Bitmap 对象；

（2）计算变换矩阵：将显示基点由图像的左上角平移到中心，反向旋转 angle 角度（弧度转换为度），将宽高分别放缩到 w 和 h，最后平移到（xc，yc）。

（3）使用此矩阵显示图像对象。

本文实验发现在显示大图片时，加载图片所需时间远大于显示图像的时间，因此减少图片加载次数能加快显示速度。本文采用下面两种方法进行图像管理：

（1）在绘图视图类中使用 LruCache 缓存图片。定义 LruCache<String, Bitmap> 类型的成员变量，以图像标识串（drawBitmap 中的 name）为键值管理图像对象。

（2）加载图片前先检查图片的宽高，如果太大就以降低采样率方式加载图片。

5.3 绘图视图的设计和实验

为了提高视图的显示质量和性能，本文针对 View、SurfaceView 进行了实验。

5.3.1 实现方式

绘图视图使用画布适配器 CanvasAdapter 绘图，由跨平台内核中的 GiCoreView 和 TestCanvas 类自动显示测试图形。绘图视图类的关系见图 5-6，实现方式说明如下。

（1）GraphView。从 View 派生，在 onDraw 中绘图，调用 invalidate()重绘。在 onDraw 中调用内核视图的 drawAll 函数显示所有图形。在触摸响应函数中调用内核视图的 onGesture 函数传递手势动作，由后者在某个交互命令中调用视图的 redraw 等函数，这将回调到 GraphView 的视图适配器（ViewAdapter），后者调用视图的 invalidate()标记需要重绘。

（2）面板表面视图。从 SurfaceView 派生。调用 setZOrderOnTop(true)设置为面板窗口，显示于宿主窗口之上。调用 getHolder().setFormat(TRANSPARENT)设置其 Surface 背景透明，以显示宿主窗口的内容。在 Surface 就绪和刷新显示时启动渲染线程，在绘图线程中获取画布绘图，由内核视图的 drawAll 函数显示所有图形。

（3）媒体表面视图。从 SurfaceView 派生。默认就是媒体窗口，显示于宿主窗口之下，自动在宿主窗口上设置透明区域以便让 SurfaceView 上的内容可见。在 Surface 就绪和刷新显示时启动渲染线程，在绘图线程中获取画布绘图。

（4）GraphViewCached。从 View 派生，使用一个位图缓存图形内容，在 onDraw

函数中显示该位图。

内核调用 regenAll 函数时销毁该位图，下次 onDraw 函数执行时重新生成位图。应用增量绘图技术，添加新图形后调用 regenAppend 函数，直接在该位图上绘制新图形，下次 onDraw 函数执行时显示有新内容的位图。

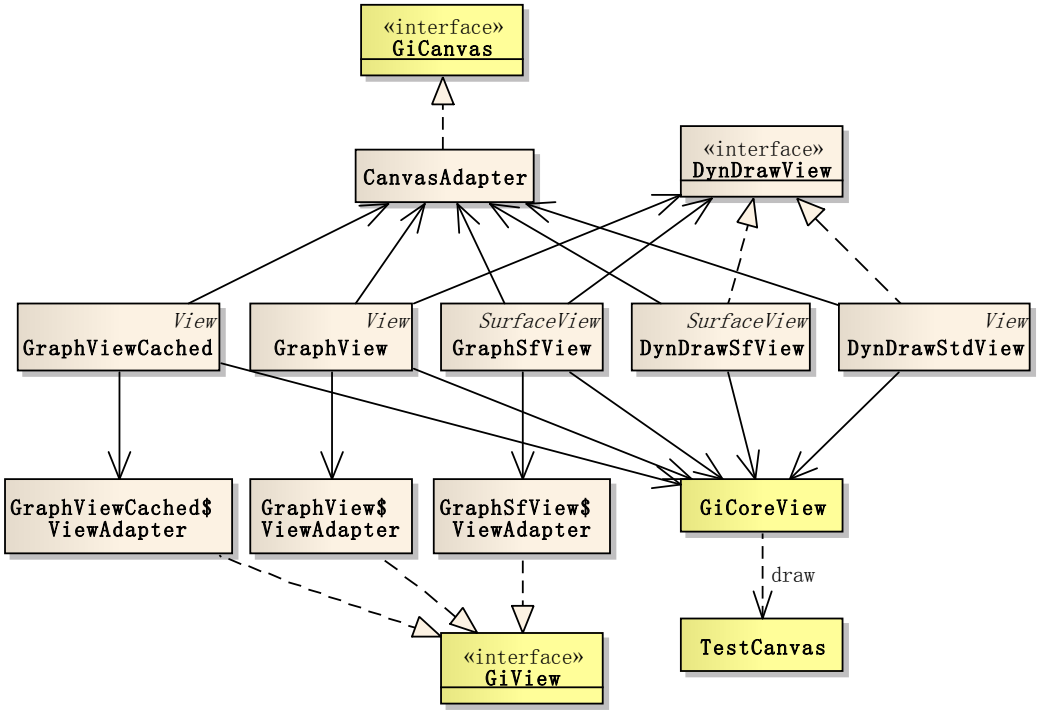


图 5-6 Android 渲染视图的类关系

（5）静态 View + 动态 View。在布局视图中创建两个基于 View 的视图类（GraphView 和 DynDrawStdView），分别显示不变的图形和经常改变的内容，前者渲染的内容通常较多。

（6）面板表面视图 + View。在布局视图中创建 GraphSfView 和 DynDrawStdView 视图。在 GraphSfView 中显示静态图形，GraphSfView 位于窗口顶端。在 DynDrawStdView 中显示动态图形。

（7）静态 View + 面板表面视图。在布局视图中创建 GraphView 和 DynDrawSfView 视图。在 GraphView 中显示静态图形，在 DynDrawSfView 中显示动态图形。DynDrawSfView 是面板表面视图，在子线程中获取画布绘图，每次刷新显示时启动渲染线程。

（8）面板表面视图 + 面板表面视图。在两个位于窗口顶端的视图类中分别显示静态图形和动态图形。

（9）媒体表面视图 + View。在 GraphSfView 中显示静态图形，GraphSfView 位于根视图层次的底端，在 DynDrawStdView 中显示动态图形。

（10）媒体表面视图 + 面板表面视图。在两个基于 SurfaceView 的视图类中分别渲染静态图形和动态图形，静态图形在窗口底端渲染，动态图形在顶端渲染。

以上的视图类按表 5-3 调用内核视图的显示函数，由 GiCoreView 显示图形。

表 5-3 Android 视图类与内核显示函数的对应关系

视图类	对应的 GiCoreView 显示函数	视图类	GiCoreView 函数
GraphView	drawAll	DynDrawStdView	dynDraw
GraphSfView	drawAll	DynDrawSfView	dynDraw
GraphViewCached	drawAppend、dynDraw、drawAll		

5.3.2 实验结果

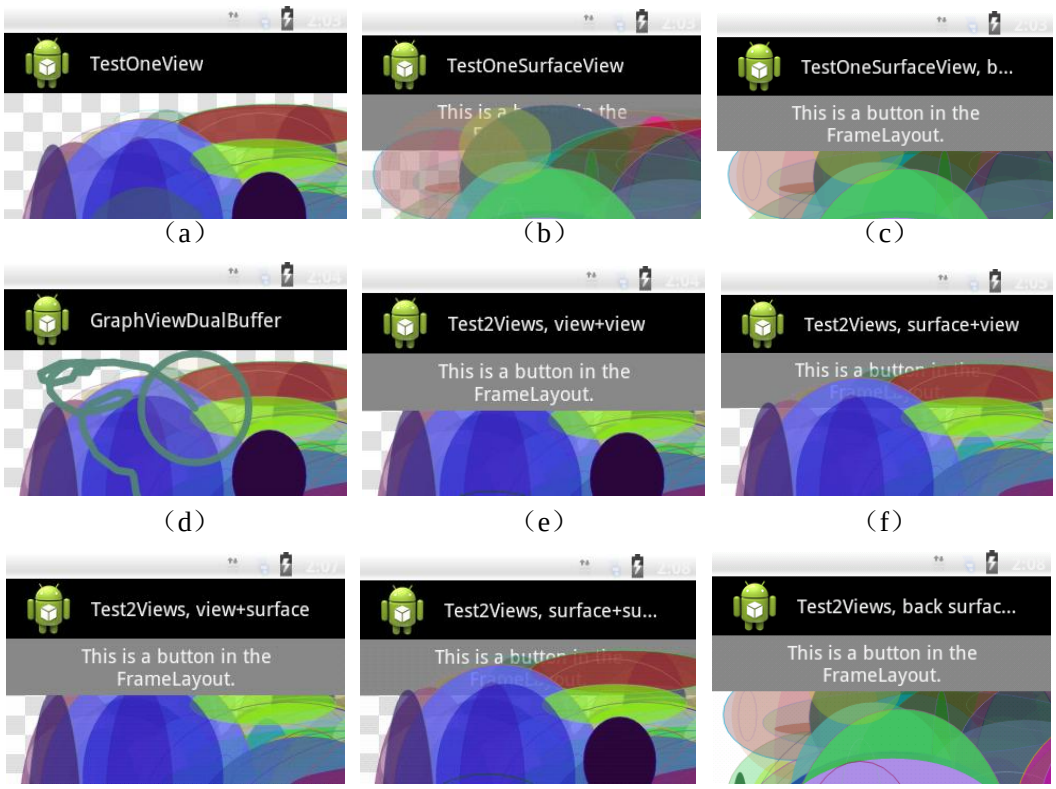
本文针对 View、SurfaceView 进行了上述十组实验，实验结果见图 5-7 和表 5-4。实验条件为：Android 3.0、模拟器（320×480），其中使用较小分辨率是便于在本文中插入屏幕截图。在 MOTO MZ606 平板电脑（Android 4.0.3，1280×800）上实验后也得出相同的结论。

从这十组实验得到下列结论：

（1）普通的绘图方式基于 View 实现定制视图，在 onDraw 函数中使用 Canvas 进行绘图。该方式使用简单，适合绘制简单图形。缺点是绘图速度较慢，刷新一个视图会使同级的其他视图被动刷新，容易引起显示性能下降问题。

（2）交互式绘图显示速度快的方式有：使用增量绘图技术的普通视图（GraphViewCached）；在 SurfaceView 中绘制动态图形的双层绘图视图。使用增量绘图技术的优点是可以只需要一个绘图视图，双层绘图视图的优点是可以在子线程中绘图，能提高刷新帧率。可以将两者的优点结合起来，在 GraphViewCached 中显示静态图形，在 SurfaceView 中绘制动态图形。

（3）如果要在 SurfaceView 中异步绘制静态图形，合适的使用条件有：a、与其他内容视图没有重叠区域；b、在窗口顶端透明显示，不要在此区域显示按钮等临时界面控件；c、在窗口底端显示，窗口里没有不透明的大面积界面元素。



（a） （b） （c）

（d） （e） （f）

（g） （h） （i）

图 5-7 Android 渲染视图效果

表 5-4 Android 渲染视图的组合实验情况

图号	视图搭配类型	显示速度和问题
a	GraphView	慢
b	GraphSfView（面板表面视图）	慢，图形遮挡按钮
c	GraphSfView（媒体表面视图）	慢
d	GraphViewCached	动态和静态绘图都很快
e	静态 View + 动态 View	慢，另一视图被动刷新
f	面板表面视图 + View	快，静态图形遮挡按钮和动态图形
g	静态 View + 面板表面视图	快
h	面板表面视图 + 面板表面视图	快，动态绘图拖尾明显，遮挡按钮
i	媒体表面视图 + View	快，不透明视图会遮挡静态图形
-	媒体表面视图 + 面板表面视图	快，不透明视图会遮挡静态图形

注：其中的图号为图 5-7 中的子图号。

5.4 Android 绘图平台的结构

5.4.1 静态结构

根据绘图视图的实验结果，Android 绘图平台按图 5-8 设计静态类结构（省略了跨平台内核的内部结构和 SWIG 的 Director 类），相应类的说明见表 5-5。

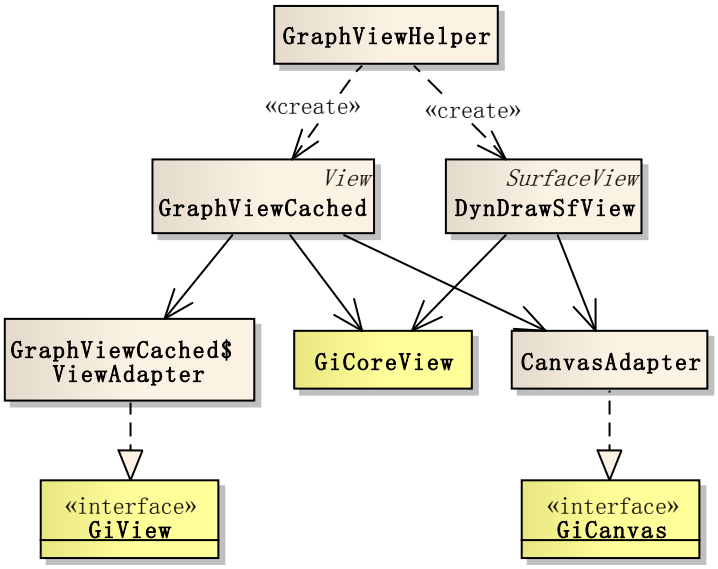


图 5-8 Android 绘图适配模块的结构

表 5-5 Android 绘图适配模块的类

类	含义和职责
GraphViewHelper	面向应用程序的绘图封装接口类，提供常用 API
GraphViewCached	显示静态图形的视图类，使用了基于缓存位图的增量绘图技术，负责触摸手势识别，委托内核的 GiCoreView 实现图形显示和手势操作
DynDrawSfView	显示动态图形的 SurfaceView 视图类，委托 GiCoreView 显示动态图形
ViewAdapter	视图适配器，允许内核回调 Android 视图，通知刷新显示
CanvasAdapter	使用 Android Canvas 实现的画布适配器
GiCoreView	跨平台内核的视图分发器，托管图形对象，分发显示请求和手势信息给图形列表和当前命令

应用程序使用绘图视图有两种方式：（1）仅使用 GraphViewCached 视图，适合图形量不太多的场合。（2）通过 GraphViewHelper 创建一个布局视图，自动创建 GraphViewCached 和 DynDrawSfView 视图，适合动态绘图帧率要求较高的场合。

5.4.2 应用效果

在 Android 绘图平台（属于 TouchVG 框架）中应用多层绘图技术分离静态图形视图和动态图形视图，提高了动态交互式绘图的回显速度。在静态图形视图中应用增量绘图技术，在连续绘制曲线图形时没有明显的拖尾现象。因此，绘图体验较流畅。

在跨设备平台的内核中使用绘图命令可以显示各种图形，在内核视图中使用仿射变换可以实现放缩显示。图 5-9 展示了在不同 Android 版本的模拟器和平板电脑上的实际绘图效果。

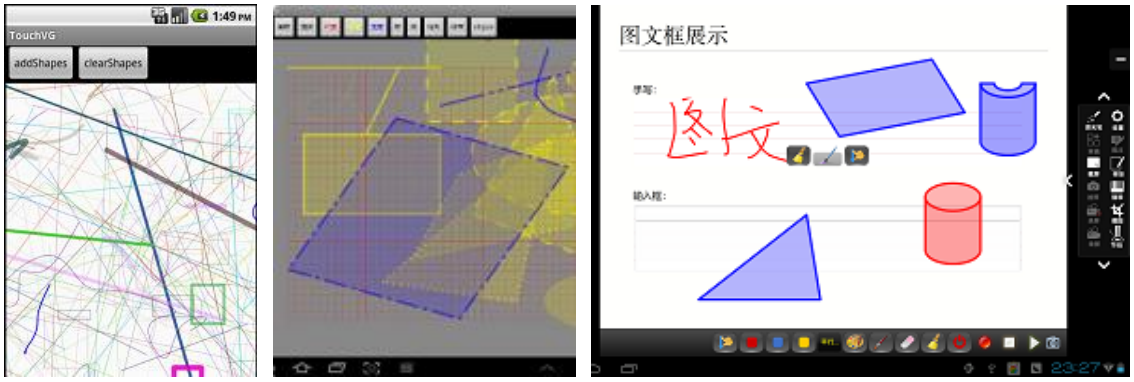


图 5-9 Android 综合绘图效果

5.5 本章小结

本章详细描述了 SWIG 在 Android 中的应用方法和扩展机制，针对出现的本地引用问题提出了修正方法。实验表明，SWIG 所增加的封装函数并不会使绘图性能明显下降。描述了基于 Android Canvas 实现画布适配器的方式，实现了图形和图像的矢量化显示。画布适配器的单元测试使用了跨平台内核自动绘制图形，证明在内核中可以使用 C++在 Android 上交互式绘图。

对普通视图、面板表面视图和媒体窗口进行了组合实验，总结出交互式绘图显示速度快、不出现遮挡问题的两种方式：使用增量绘图技术的单一视图方式；在 SurfaceView 中绘制动态图形的双层视图方式。

最后给出了 Android 绘图平台的设计结构和应用效果。

第6章　绘图平台的应用和评估

本章总结了 TouchVG 平台的应用方式，从应用效果、特性、与同类平台对比等多个方面对 TouchVG 平台进行了评估。

6.1　绘图平台的应用方式

TouchVG 平台在数字教育移动设备软件中主要以阅读批注形式应用，即在已有的阅读器页面视图上附加批注视图进行交互式绘图。其应用形式如图 6-1 所示。

（a）全屏批注　　（b）局部批注

图 6-1 绘图平台的应用形式

批注界面形式主要有两种：（1）在全屏页面上绘图，如图 6-1（a）所示。用户可自由切换阅读模式和批注绘图模式。（2）在页面局部区域的交互组件上绘图，如图 6-1（b）所示。当在组件区域内触摸时可进行批注绘图操作，当在组件区域外触摸时可进行阅读和翻页等操作。这两种界面形式的主要区别是显示区域不同、绘图按钮数量不同以及进入页面后能否自动切换到绘图模式。

交互式批注绘图界面的主要实现类如图 6-2 所示，其含义和职责见表 6-1。

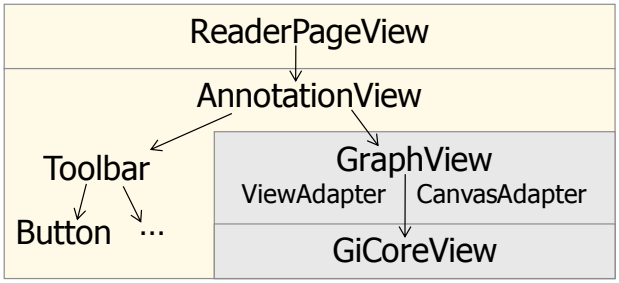


图 6-2 批注绘图的应用方式

在 iOS 和 Android 上相应实现类的类名略有不同，其命名遵循相应的习惯。例

如，GraphView 的类名分别为 GiGraphView 和 touchvg.view. GraphViewCached。

表 6-1 阅读批注绘图的主要类

类	含义和职责
ReaderPageView	阅读器的页面视图，显示页面内容（底图）
AnnotationView	应用层的批注封装视图，负责在页面视图中界面布局
Toolbar、Button	应用层的批注按钮，调用 GraphView 的函数传递动作
GraphView	通用绘图平台的交互式绘图视图
GiCoreView	跨平台内核的视图分发器，托管图形对象，分发显示请求和手势信息给图形列表和当前命令

在 iOS 和 Android 上分别提供了本文所描述的通用绘图平台，阅读批注应用程序在使用 TouchVG 平台时主要的开发工作有：

（1）在批注封装视图（AnnotationView）中实现视图的生命期管理和视图布局，根据不同的使用场景设置底部工具栏所要包含的按钮。在按钮的响应函数中调用 GraphView 的函数。例如，“选择”按钮调用 view.commandName=“select”；“绘图”按钮调用 view.commandName=“line”；“清除”按钮调用 view.clear()函数。

因为批注封装视图所调用的绘图平台函数都不复杂、不涉及矢量绘图的底层细节，所以应用系统的开发工作量较小。

（2）在系统中应用绘图数据。绘图数据主要有矢量图形的 JSON 格式文本内容和背景透明的绘图快照图片。前者用于数据交换和存储管理，后者用于预览和页面快速显示。因为不涉及图形存储等实现细节，所以应用开发的工作量也较小。

6.2 绘图平台的评估

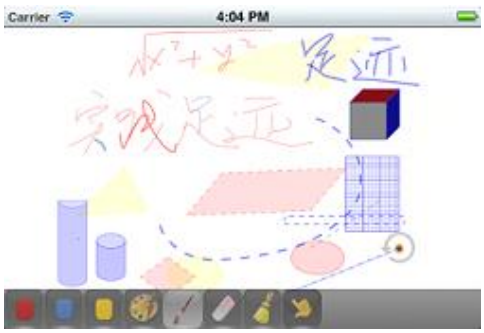
6.2.1 跨平台应用效果

TouchVG 平台的应用效果如图 6-3 所示[9]。该图展示了 TouchVG 平台在 iPhone 模拟器、iPad 真机、Android 平板电脑以及 Windows 上的绘图效果，体现了 TouchVG 平台在各种移动设备上的跨平台一致性、矢量图形的可交换性和分辨率自适应性。

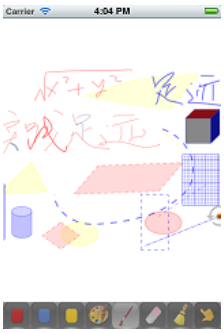
图 6-3 中，（a）和（b）是 TouchVG 平台在 iPhone 模拟器上的测试程序效果。

[9] 手绘曲线采用三次参数样条曲线模型，本文对曲线模型和数据点采样法不做研究和论述。

仅显示矢量图形，没有背景图。（c）和（d）是 TouchVG 平台在 iPad 和 Android 平板电脑上的批注效果。实现方式是在阅读器页面视图上附加批注外壳界面，外壳界面包含绘图视图和底部工具栏。（e）是 TouchVG 平台在 Windows 上的显示效果。使用 GDI+实现了画布适配器，使用 MFC 实现了视图适配器。



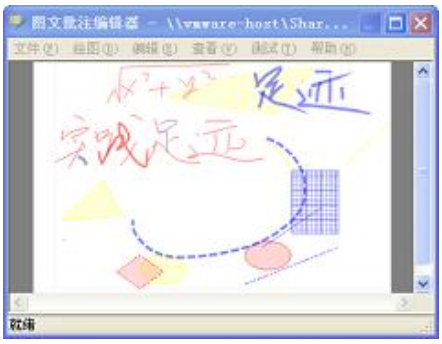
（a）iPhone 横屏



（b）iPhone 竖屏

（c）iPad 页面绘图

（d）Android 页面绘图



（e）Windows 绘图

图 6-3 跨平台应用效果

测试方法：首先，将在 iPad 上绘制的所有图形保存为 JSON 格式的矢量图形文件。然后，通过提取该矢量图形文件，在其他设备上加载该文件。iPhone 或 iPod Touch 的屏幕分辨率与平板电脑的差异较大，需要放缩显示才能显示出所有图形。

TouchVG 平台从设计上保证了跨平台效果：在跨平台内核中实现交互绘制、图形显示、图形保存以及加载等主要功能；在 iOS 和 Android 上仅实现画布和视图适配器、批注外壳界面（包括工具栏、按钮、视图布局）等少量功能。

6.2.2 绘图平台的特性

本节从代码量、实现难度和可扩展性等多个方面对 TouchVG 平台进行了评估。

（1）代码量和比例。

使用 SourceMonitor[10]工具对 TouchVG 进行统计的结果：总共 2.5 万行代码，跨平

[10] SourceMonitor：代码质量度量工具，支持对 C/C++、Java、C#等编程语言的代码统计分析。

台内核 18 740 行代码，iOS 适配器模块 3 713 行代码，Android 适配器模块 2 197 行代码。github.com 网站对 TouchVG 平台的代码统计结果：C/C++占 77%，Java 占 14%，Objective-C 占 8% ，其他 1%。主要代码是跨平台的 C++代码。

（2）实现难度和可扩展性。

TouchVG 平台的主要功能在跨平台内核中使用 C++实现，其实现难度和可扩展性主要体现在跨平台内核上。跨平台内核主要需要对绘图命令和图形实体进行扩展，增加更多绘图功能。绘图命令的实现结构见图 6-4，其含义见表 6-2。

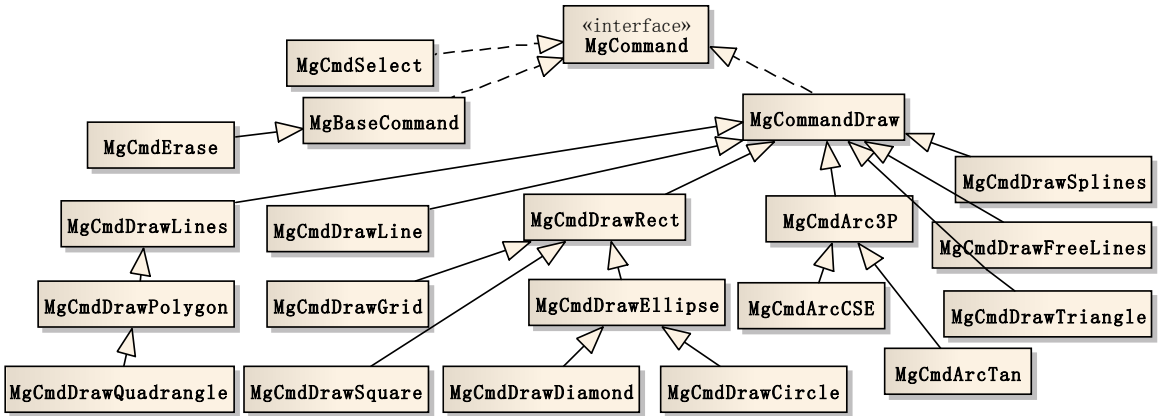


图 6-4 交互命令类的结构

表 6-2 绘图命令类的说明

类	含义	类	含义
MgCommand	命令接口	MgBaseCommand	命令基类
MgCmdSelect	选择命令	MgCommandDraw	绘制命令基类
MgCmdErase	删除命令	MgCmdDrawCircle	圆形命令
MgCmdDrawLines	折线命令	MgCmdDrawSplines	自由曲线命令
MgCmdDrawPolygon	多边形命令	MgCmdDrawFreeLines	自由折线命令
MgCmdDrawQuadrangle	四边形命令	MgCmdDrawTriangle	三角形命令
MgCmdDrawLine	直线命令	MgCmdArc3P	三点圆弧命令
MgCmdDrawRect	矩形命令	MgCmdArcCSE	圆心圆弧
MgCmdDrawGrid	网格命令	MgCmdArcTan	切线圆弧
MgCmdDrawSquare	正方形命令	MgCmdDrawEllipse	椭圆命令

使用 SourceMonitor 对交互绘图命令的实现结果进行了统计，从表 6-3 的结果可见，绘图命令类的代码行数较少、复杂度较低，证明跨平台的交互命令容易实现。

表 6-3 交互命令类的实现统计

文件	语句行数	平均复杂度	文件	语句行数	平均复杂度
mgcmdselect.cpp	461	2.82	mgcmd.h	69	1.00
mgdrawlines.cpp	136	2.74	mgdrawlines.h	64	1.00
mgcmddraw.cpp	126	2.09	mgdrawtriang.h	60	1.00
mgdrawsplines.cpp	120	4.15	mgcmddraw.h	46	1.14
mgdrawtriang.cpp	111	2.92	mgdrawline.cpp	33	1.25
mgcmderase.cpp	102	2.40	mgcmderase.h	30	1.00
mgdrawrect.cpp	85	1.42	mgdrawline.h	28	1.00
mgdrawrect.h	76	1.00	mgdrawsplines.h	25	1.00
mgcmdselect.h	72	1.00	mgselect.h	18	1.00

在可扩展性方面，增加一个绘图命令功能的主要开发工作是基于命令基类实现一个交互命令类，从表 6-3 的定量分析结果可见命令类的代码量较少，容易扩展。

6.2.3 同类平台的特性对比

目前，国内还缺乏可应用在 iOS 和 Android 上的二维绘图平台（开发包），而国外的同类平台则相对较多。表 6-4 列出了一些典型平台的对比分析结果。

表 6-4 同类平台的特性对比

绘图平台	设备环境	开发成本	功能特点	图形种类
Core-Plot	iOS	不高	动态图表	图表
Vectoroid	Android	较高	较全面、SVG	图表、图形
TouchDraw	iOS/Android	未开源	很全面	流程图、图库
TouchVG	iOS/Android/ Windows	集成容易	绘图基本图形、智能捕捉、JSON 读写	基本曲线图形、图像、单行文字

对比可知，TouchVG 平台在多平台支持、应用开发难度等方面有一定的优点，在功能支持方面有一定的不足。因此，TouchVG 平台适合于对性能和图形量要求适中、功能不是很多的常用矢量绘图应用，而对于 CAD、GIS 等专业领域则存在较多不足。

6.3 本章小结

本章描述了 TouchVG 平台在不同的移动平台上的绘图效果和开发方式。结果表明，TouchVG 平台具有较好的跨平台一致性、矢量图形的可交换性和分辨率自适应性。因为不涉及矢量绘图的底层细节，所以基于 TouchVG 平台的应用开发工作量较小。与相似平台相比，TouchVG 平台在多种设备环境的支持、应用开发工作量等方面具有较多优点，而在功能广泛性方面则有一定的不足。

总结和展望

本文在 iOS 和 Android 上验证了基于虚函数多态行为的跨平台扩展机制的可行性和实用性。提出了一种适合多种移动设备的矢量绘图平台设计方法：在跨平台内核中使用 C++实现图形管理和交互绘图功能，在不同的移动平台上开发画布适配器和视图适配器。该方法既能在跨平台内核中扩充更多的功能，又能复用高质量图形库和界面库，降低了移植难度和应用开发工作量。

在 iOS 平台上，基于 Quartz 2D 实现了画布适配器，提出了适合连续手绘的增量绘图的实现方法和快速动态绘图的多层绘图实现方法，通过有效利用缓存位图和层加快了交互式绘图的回显速度。实验证明传统的双缓冲绘图技术不适合 iOS。本文总结了各种绘图属性对性能的影响，总结了图形数量与显示时间的线性比例规律，为应用逐步渲染技术提供了实验依据。

在 Android 上，选用 SWIG 实现了应用层与跨平台绘图模块的集成和基于多态的扩展方式。实验表明 SWIG 不会使绘图性能明显下降。基于 Android Canvas 实现了画布适配器，实现了图形和图像的矢量化显示。对普通视图和 SurfaceView 进行了组合实验，总结出了解决交互式绘图显示较慢、出现遮挡问题的两种方式：使用增量绘图技术的单一视图方式；在 SurfaceView 中绘制动态图形的双层视图方式。

目前，iOS 和 Android 的通用矢量绘图平台（开源框架 TouchVG）已经在数字教育领域投入使用，取得了一定的经济效益。第三方公司基于 TouchVG 平台开发的“脑力风暴”图文笔记软件已经在 AppStore 上架。北京邮电大学和地质大学等高校的个别研究生已经开始应用 TouchVG 平台进行课题实验。

受开发周期和经验的限制，TouchVG 平台的功能还不够丰富，在不同领域的应用中暴露了一些扩展性和理解问题。进一步的研究方向有下面几点：

（1）简化使用接口，对多种应用场景设计开发范例，提高易用性。

（2）对 iOS 层的高性能绘图进行了实验，实现高帧率动画显示。

（3）深入分析相关开源框架的优点，集成手绘光滑曲线形状等模块。

由于项目的开发时间较短、笔者经验不足，在本论文中难免存在一些问题，请各位老师和同学批评指正。

参考文献

[1] 何高奇. 面向移动设备的图形绘制技术研究[D]. 杭州:浙江大学计算机学院, 2007.

[2] 王东. 2012：数字化教育出版转型年[EB/OL]. [2012-04-19]. http://www.dajianet.com/digital/2012/0419/185839.shtml.

[3] 官酩杰. 基于 OpenGL ES 的移动平台图形渲染研究与实现[D]. 北京:北京交通大学, 2010.

[4] 沈江. 基于 OpenVG 的图形绘制关键技术研究[D]. 杭州:杭州电子科技大学, 2010.

[5] 何必仕, 万健, 徐小良. 基于 OpenVG 矢量图渲染加速研究[J]. 计算机应用与软件, 2010, 27(1):111-113.

[6] 周方晓, 李昌华, 丁有军. 用 GDI+和面向对象设计方法构建交互式图形平台[J]. 微电子学与计算机, 2010, 27(10): 165-169.

[7] 鲁力, 李欣. 基于 AGG 算法库的通用图形接口设计[J]. 微计算机信息, 2009, 25(2):266-267.

[8] 杨硕飞. 动态几何画板的研究及其在 iPhone 平台的实现[D]. 成都:电子科技大学, 2011.

[9] 李慧云, 何震苇, 李丽等. HTML5 技术与应用模式研究[J]. 电信科学, 2012, 28(5):24-29.

[10] CADalyst. Application on the mobile device [EB/OL]. [2012-06-18]. http://www.singaporebim.com/Articles_Oversea/147.html.

[11] Kurt R. Drawing image with CoreGraphics on Retina iPad is slow [EB/OL]. [2012-05-03]. http://stackoverflow.com/questions/10410106/drawing-image-with-coregraphics-on-retina-ipad-is-slow/10424635.

[12] Ariya H. Understanding Hardware Acceleration on Mobile Browsers [EB/OL]. [2011-07-15]. http://www.sencha.com/blog/understanding-hardware-acceleration-on-mobile-browsers.

[13] Tommy. Core Graphics Performance on iOS [EB/OL]. [2012-11-07]. http://stackoverflow.com/questions/13277031/core-graphics-performance-on-ios.

[14] Kurt R. Back-buffering performance on iOS with Quartz [EB/OL]. [2012-07-01]. http://www.jwz.org/blog/2012/07/back-buffering-performance-on-ios-with-quartz.

[15] Freerunnering. Drawrect with CGBitmapContext is too slow [EB/OL]. [2012-07-03]. http://stackoverflow.com/questions/11261450/drawrect-with-cgbitmapcontext-is-too-slow.

[16] Aaron H. CGContextDrawLayerAtPoint is slow on iPad 3 [EB/OL]. [2012-06-19]. http://stackoverflow.com/questions/11100984/cgcontextdrawlayeratpoint-is-slow-on-ipad-3.

[17] Chris P. Writing Real time Games for Android [EB/OL]. [2009-06-22]. http://www.linuxgraphics.cn/android/write_real_time_for_android.html.

[18] Ryan W. The truth about hardware acceleration on Android [EB/OL]. [2011-12-06]. http://www.extremetech.com/computing/107995-the-truth-about-hardware-acceleration-on-android.

[19] Bhanu C. Learning about Android Graphics Subsystem [EB/OL]. [2012-04-11]. http://developer.mips.com/2012/04/11/learning-about-android-graphics-subsystem.

[20] Robert G. Getting Great Game Performance [EB/OL]. [2009-05-25]. http://www.rbgrn.net/content/290-light-racer-2

0-days-32-33-getting-great-game-performance.

[21] 侯炯. Webkit 分析报告 II[EB/OL]. [2009-02-10]. http://www.doc88.com/p-78940304236.html.

[22] 张江水, 华一新, 唐衡丽等. 嵌入式 GIS 跨平台技术的研究与实现[J]. 测绘科学技术学报, 2012, 29(3):214-217.

[23] 刘楠, 李欣. 跨平台高质量二维图形库设计与实现[J]. 计算机工程与设计, 2010, 31(7):1599-1622.

[24] Fain G. Curves and surfaces for CAGD: A practical guide, 5th Edition [M]. San Francisco: Morgan Kaufmann Publishers Inc., 2002.

[25] Mike K. A primer on Bezier curves [EB/OL]. [2011-10-05]. http://processingjs.nihongoresources.com/bezierinfo.

[26] Luke W. Touch Gesture Reference Guide [EB/OL]. [2010-04-15]. http://www.lukew.com/touch.

[27] Epps J, Lichman S, Wu M. A study of hand shape use in tabletop gesture interaction[A]. Proceedings of CHI EA '06 [C]. New York: ACM, 2006: 748-753.

[28] 机锋网. Android 旗舰机多点触控测试[EB/OL]. [2011-03-12]. http://3g.163.com/coop/ucweb/mobile/11/0312/00/6UTG4KEE00112K95_0.html.

[29] 孟祥亮. 显示表面上多触点手势研究[D]. 北京:清华大学计算机科学与技术系, 2010.

[30] 季红艳. 基于多点触摸技术的人机交互研究[D]. 上海:华东师范大学软件学院, 2011.

[31] Charles Wilde. Developing an Android Mobile Application [EB/OL]. [2009-04-13]. http://www.aton.com/developing-an-android-mobile-application.

[32] Charles Wilde. Android Native Libraries for Java Applications [EB/OL]. [2009-09-23]. http://www.aton.com/android-native-libraries-for-java-applications.

[33] Onur Cinar. Pro Android C++ with the NDK [M]. Berkeley: Apress, 2012.

[34] 解维东. 在 JNI 编程中避免内存泄漏[EB/OL]. [2011-04-25]. http://www.ibm.com/developerworks/cn/java/j-lo-jnileak/index.html.

[35] Matt Clark. Developing Apps within Android's 16MB Memory Limit [EB/OL]. [2010-07-16]. http://blog.gorges.us/2010/07/developing-apps-within-androids-16mb-memory-limit.

[36] 金德. Android 内存溢出的解决方法总结[EB/OL]. [2012-12-28]. http://blog.csdn.net/jindegegesun/article/details/8447434.

[37] Wikipedia. List of iOS devices[EB/OL]. [2012-10-26]. http://en.wikipedia.org/wiki/List_of_iOS_devices.

[38] Reda L. Designing for iOS: Graphics & Performance [EB/OL]. [2012-11-26]. http://robots.thoughtbot.com/post/36591648724/designing-for-ios-graphics-performance.

攻硕期间发表的论文与研究成果

成果 1：开源项目 TouchVG（http://www.oschina.net/p/touchvg），矢量图形框架，2012 年 5 月开始开发，目前已在 iOS、Android 和 Windows 平台应用。

成果 2：开源项目 vglite（https://github.com/rhcad/vglite），绘图优化实验项目。

成果 3：方正慧云互动课堂教学应用系统的 iOS 和 Android 客户端。

成果 4：方正飞翔创意版的图文批注模块，Windows 上的矢量绘图软件。

成果 5：开源项目 X3PY（https://github.com/rhcad/x3py），通用跨平台 C++插件框架，2011 年底开始开发，支持 python、C#、Java。

成果 6：开源项目 X3C（https://github.com/rhcad/x3c），通用跨平台 C++插件框架，2011 年开发，目前已应用在多家公司的软件中。

致谢

本论文得以完成，首先感谢导师张春霞副教授。张春霞老师学风严谨、认真负责，从论文开题到写作的一年内经常主动监督和指导我的工作，提出了很高的要求，多次当面指导，耐心细致的提出了各种修改意见。

感谢我的企业导师郭宗明研究员、博导。郭老师在治学态度、研究方法上给我很多指导甚至是批评，在论文选题、写作思路上给了很多指导意见，在繁忙的工作中仔细评阅了各种文档。

从论文开题起，所在的方正电子公司的曹学军部长一直鼓励我，给我充足的研究时间和完善的硬件条件。史航同学在论文写作思路上给了很多启发，对论文进行了仔细评阅。IBM 公司的黄冶和浙江大学的徐周翔博后对英文摘要进行了审阅和指正。同事汤寿麟、李文博在绘图平台的测试和应用上给了很多建议，王纬和王元在开源项目上进行了帮助。对他们所有支持和帮助，在此一并表示感谢。

最后感谢各位专家评委对本论文的指正和建议。