

国内首本基于iOS的增强现实类著作，系统讲解增强现实的实用工具、核心技术和基本原理，以及如何将增强现实技术与iOS开发技术相融合

图文并茂、实例丰富，循序渐进地讲解了游戏、社交和面部识别等各种类型的iOS应用的开发方法，可操作性强

Pro iOS 5 Augmented Reality

iOS增强现实 应用开发实战

(美) Kyle Roche 著
徐学磊 译



机械工业出版社
China Machine Press

本书全面讲解了如何创建能够释放iOS所有潜能的增强现实应用，展示如何使用你的iOS设备的传感器和摄像头来整合面部识别和社交媒体功能与周围环境实现互动。

本书涵盖增强现实应用所需硬件和软件背景知识，以及创建iOS增强现实应用的基本原理、实用工具：从MapKit的使用，到加速计和指南针，再到面部识别和Facebook数据的整合，你将会在参与案例分析的过程中掌握如何创建实用、有趣的增强现实应用。

通过阅读本书，你将会学习到：

- 如何使用MapKit以及如何在你的应用中整合MapKit；
- 如何在一个增强现实应用中播放和记录声音；
- 如何使用iPhone或者iPad的拍照和录像功能；
- 如何使用加速计、陀螺仪和指南针编程；
- 如何使用cocos2D在摄像头视图上覆盖一个HUD层；
- 如何在你的应用中整合进面部识别功能；
- 如何创建功能丰富的增强现实游戏和Facebook应用。

掌握本书所讲内容后，你将能够创建多媒体增强现实应用，或者把所有最好的增强现实技术和工具融入到应用中。

Apress®

客服热线：(010) 88378991, 88361066
 购书热线：(010) 68326294, 88379649, 68995259
 投稿热线：(010) 88379604

读者信箱：hzsj@hzbook.com
 华章网站：www.hzbook.com
 网上购书：www.china-pub.com



上架指导：计算机/程序设计/移动开发

ISBN 978-7-111-42020-0



9 787111 420200 >

定价：59.00元

Pro iOS 5 Augmented Reality

iOS增强现实 应用开发实战

(美) Kyle Roche 著

徐学磊 译



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

iOS 增强现实应用开发实战 / (美) 罗什 (Roche, K.) 著; 徐学磊译. —北京: 机械工业出版社, 2013.5
(华章程序员书库)

书名原文: Pro iOS 5 Augmented Reality

ISBN 978-7-111-42020-0

I. i… II. ①罗… ②徐… III. 移动电话机-应用程序-程序设计 IV. TN929.53

中国版本图书馆 CIP 数据核字 (2013) 第 066946 号

版权所有·侵权必究

封底无防伪标均为盗版

本书法律顾问 北京市展达律师事务所

本书版权登记号: 图字: 01-2013-0215

国内首本基于 iOS 的增强现实类著作, 系统讲解增强现实的实用工具、核心技术和基本原理, 以及如何将增强现实技术与 iOS 开发技术相融合。图文并茂、实例丰富, 以通俗易懂的语言循序渐进地讲解了游戏、社交和面部识别等各种类型的 iOS 应用的开发方法, 可操作性强。更为重要的是, 它还讲解了如何将增强现实技术应用到已有的应用中!

本书共分 13 章, 具体内容如下: 第 1 章主要介绍创建增强现实应用的准备工作以及关键章节的简述; 第 2 章介绍检查各种硬件组件可用性的方法; 第 3 章介绍 iOS 的地图功能和将其集成到具体应用的高级技术; 第 4 章介绍加速计和陀螺仪、磁力计等 iOS 传感器; 第 5 章介绍如何在一个增强现实应用中播放和记录声音; 第 6 章, 介绍如何使用 iPhone 或者 iPad 的拍照和录像功能进行视频采集; 第 7 章讲解如何使用 cocos2D 在摄像头视图上覆盖一个 HUD 层; 第 8 章细述一个 cocos2D 增强现实应用的创建; 第 9 章介绍 String、Qualcomm 和 ARKit 增强现实工具包的应用; 第 10 章讲述利用 String、OpenGL ES 建立一个基于标记的增强现实应用; 第 11 章介绍如何建立一个社交型的增强现实应用; 第 12~13 章介绍面部识别技术及其在增强现实应用中的用法。

Pro iOS 5 Augmented Reality (ISBN: 9781430239123).

Original English language edition published by Apress L.P., 2560 Ninth Street, Suite 219, Berkeley, CA 94710 USA. Copyright © 2011 by Kyle Roche. Simplified Chinese-language edition copyright © 2013 by China Machine Press. All rights reserved.

This edition is licensed for distribution and sale in the People's Republic of China only, excluding Hong Kong, Taiwan and Macao and may not be distributed and sold elsewhere.

本书原版由 Apress 出版社出版。

本书简体字中文版由 Apress 出版社授权机械工业出版社独家出版。未经出版者预先书面许可, 不得以任何方式复制或抄袭本书的任何部分。

此版本仅限在中华人民共和国境内 (不包括中国香港、台湾、澳门地区) 销售发行, 未经授权的本书出口将被视为违反版权法的行为。

机械工业出版社 (北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑: 高婧雅

北京市荣盛彩色印刷有限公司印刷

2013 年 5 月第 1 版第 1 次印刷

186mm×240mm · 16.75 印张

标准书号: ISBN 978-7-111-42020-0

定 价: 59.00 元

凡购本书, 如有缺页、倒页、脱页, 由本社发行部调换

客服热线: (010) 88378991 88361066

购书热线: (010) 68326294 88379649 68995259

投稿热线: (010) 88379604

读者信箱: hzjsj@hzbook.com

译者序

增强现实是一个非常有趣的新兴领域，并且潜力无限。其实它在科幻电影和军事上已经存在很久了，直到最近几年，随着移动设备的兴起，终于开始融入到我们的生活中。

增强现实是一项很神奇的技术，使用这种技术可以产生很多非常吸引人的交互体验，例如帮助人们尝试各种新式发型、远程试穿衣服、实时显示产品信息、演示玩具的玩法，等等。而且这些还只是增强现实用途的一小部分，随着时间的推移，增强现实一定会在不远的将来渗透到我们生活的方方面面，甚至是实现现实世界与虚拟世界的融合。

本书为国内首本增强现实类书籍，涵盖实用、前沿的开发工具和技能知识，语言通俗易懂，容易掌握，是学习增强现实的理想选择之一。更为重要的是，它讲解了如何将增强现实技术应用到已有的应用中。本书从 iOS 设备的硬件配置讲起，逐步介绍了增强现实技术的各个技术细节，包括位置服务、传感器的使用、声音支持、拍照和录像、面部识别、cocos2D 的应用，以及多个第三方增强现实工具包。虽然本书是一本增强现实的入门书籍，需要你有一定的 Xcode 和 Objective-C 经验，但是我相信这难不倒你。建议喜欢不断尝试新鲜事物的技术人员以及有一定编程经验的创意人士阅读此书。本书内容翔实、图文并茂，而且包含丰富的例子程序，一定可以让你轻松地进入增强现实世界。学完本书后你就可以用最新的增强现实技术来实现你自己的创意。

本书的翻译得到了很多人的帮助。特别感谢机械工业出版社华章公司，谢谢他们的信任和支持。本书翻译过程中查阅、参考了大量的相关资料，力争译文通俗通畅，专业词汇准确，再现原著风貌。

最后也是最重要的，我想感谢选择阅读本书的读者。我希望本书能帮助你走进增强现实的神奇世界，祝你成功！

由于译者水平有限，错误在所难免，恳请广大读者给予批评指正。

徐学磊

2013 年 4 月

前言

这是一本写起来生动又有趣的书！增强现实是一个令人着迷的新领域，这个领域很活跃，有很多可以开发的内容，进而把技术融入到我们日常生活的地方。每周，大量的公司和工具包如雨后春笋般地出现，它们都试图占领这个新兴的市场。

本书的目的是为你开发这类应用程序提供一个思路。我从讨论程序的基本原理开始，例如指南针和加速计，再如更高级的图像处理背后的原理。

本书是面向有经验的 iOS 开发者的。你需要有一定的 Xcode 和 Objective-C 经验。我使用第三方的框架和一些 iOS 5 的新 API 向你展示如何建立增强现实应用程序，例如位置、社交和游戏。

你可以从 Apress.com 网站上本书的页面中下载书中的源代码，或者从 www.apress.com/source-code/ 下载。

致谢

我是在创业时的过渡期写的这本书。我们正从一家公司转移并启动了两个新项目。而且大部分时间 iOS 5 还处在测试中。写作这本书期间正好是我非常困难的时期。如果没有 Corbin Collins 和 Steve Anglin 领导的 Apress 团队的支持，完成这本书根本是不可能的。

从技术上来说，我想要感谢我的同事 Sergey Loshchilov。Sergey 是俄国下诺夫哥罗德国立科技大学的一个硕士研究生。他在 OpenCV 和 iOS 5 最新 API 方面对我帮助很大。Sergey 发布了一篇关于常见增强现实算法的比较文章。我会在 kylerocher.com 网站上发布指向这个文章的链接。

对于个人来说，在推进本书出版过程中的无数个漫长的夜晚和周末，我的妻子和四个孩子一直非常支持我。我感谢他们的耐心和贡献。在讲述面部识别那一章时，我有机会让孩子们一起参加，这是他们和我的共同乐趣。

推荐阅读



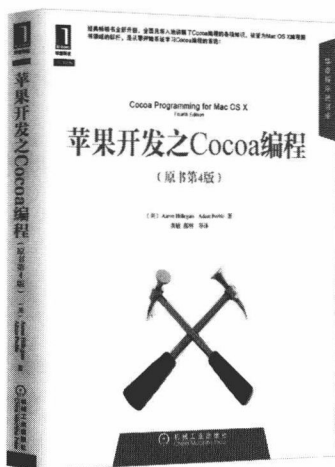
内容全面，系统讲解开发企业级iPhone应用所需掌握的各项核心技术，以及各种工具和框架的用法，包含大量技巧和最佳实践

实战性强，不仅为各个知识点精心设计了能辅助读者理解的小案例，而且还能指导读者实践的大案例，具备极强的可操作性



Amazon五星级畅销书，作者权威，在全球iOS/Mac开发者社区享有盛誉！

完美地展现了测试驱动开发方法与iOS开发的结合，能使iOS开发者在产品需求、软件设计、测试有效性与开发效率之间达成很好的平衡

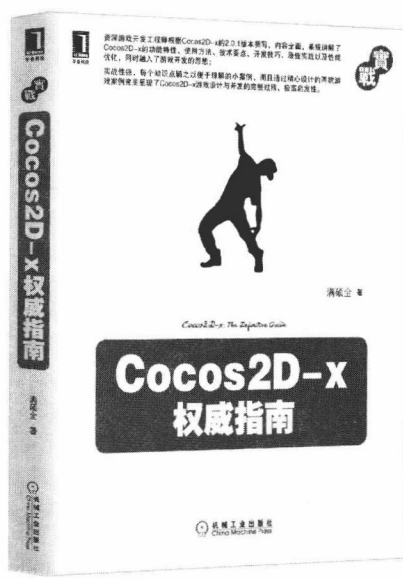


经典畅销书全新升级，系统且深入地讲解了Cocoa编程的各项知识，被誉为Mac OS X编程图书领域的标杆，被公认为从零开始学习Cocoa的首选！



资深iOS开发工程师撰写，Amazon五星级畅销书，国际Mac和iPhone开发者社区CocoaHeads联合创始人Mark Dalrymple等数位专家联袂推荐！

推荐阅读



Cocos2D权威指南

资深专家根据Cocos2D最新版本撰写，内容全面，系统讲解了Cocos2D的使用方法、技术要点、工作原理、开发技巧、最佳实践以及性能优化。

实战性强，通过精心设计的典型案例完美呈现了Cocos2D游戏设计与开发的完整过程，极富启发性。

Cocos2D-x权威指南

资深游戏开发工程师根据Cocos2D-x的2.0.1版本撰写，内容全面，系统讲解了Cocos2D-x的功能特性、使用方法、技术要点、开发技巧、最佳实践以及性能优化，同时融入了游戏开发的思想。

实战性强，每个知识点辅之以便于理解的小案例，而且通过精心设计的两款游戏案例完美呈现了Cocos2D-x游戏设计与开发的完整过程，极富启发性。



目 录

译者序

前言

第 1 章 引言	1
1.1 增强现实的应用实例	1
1.1.1 日常应用的趋势	2
1.1.2 游戏和基于位置的应用	3
1.2 准备工作	3
1.2.1 注册 GitHub 账户	3
1.2.2 从机器访问 GitHub	4
1.2.3 安装 Xcode 4.2 和创建开发者账户	5
1.2.4 连接 Xcode 工程到 GitHub	6
1.2.5 创建 Xcode 工程	8
1.2.6 连接工程到远程仓库	9
1.3 下一步做什么	9
1.3.1 位置服务	10
1.3.2 传感器编程	10
1.3.3 声音和视频采集	10
1.3.4 游戏框架	10
1.3.5 第三方框架	10
1.4 总结	10
第 2 章 硬件比较	11
2.1 除旧存新	11
2.2 硬件组件	12
2.2.1 摄像头支持	12
2.2.2 位置检测能力	15
2.2.3 数字指南针	16
2.2.4 声音支持	17
2.2.5 检查录像功能	18
2.2.6 加速计和陀螺仪	19
2.3 强制硬件需求	21
2.4 总结	22
第 3 章 使用位置服务	23
3.1 基础功能	23
3.1.1 标准位置服务	26
3.1.2 显著变更位置服务	29
3.1.3 地理区域监控服务	30
3.1.4 高度	34
3.2 在地图上查看	35
3.2.1 置中地图和设置显示区域	36
3.2.2 修改地图类型	38
3.2.3 在地图上添加注释	40
3.3 解析地理编码	42
3.4 总结	44
第 4 章 iOS 传感器	45
4.1 方向传感器	45
4.1.1 使用加速计	45
4.1.2 低通滤波	49

4.1.3 使用陀螺仪	50	7.3 初识增强现实应用视图	91
4.1.4 磁力计	55	7.3.1 调整默认视图	91
4.2 总结	60	7.3.2 添加摄像头视图	92
第 5 章 声音和用户反馈	61	7.3.3 缩放摄像头视图	93
5.1 音频数据格式	61	7.4 cocos2D 的概念	94
5.1.1 哪一种格式适合我们呢	62	7.4.1 场景	94
5.1.2 文件保存格式	62	7.4.2 控制器	95
5.1.3 比特率和质量	62	7.4.3 图层	95
5.1.4 采样率	62	7.5 添加效果	95
5.1.5 为在 iOS 中使用而转换音频格式	63	7.5.1 处理触摸事件	96
5.2 在 iOS 应用中播放声音	65	7.5.2 视觉效果	97
5.2.1 系统声音服务	66	7.5.3 添加声音效果	98
5.2.2 AVAudioPlayer 类	66	7.6 添加 HUD 层	99
5.2.3 测试多重音频播放	66	7.7 总结	103
5.2.4 播放位置声音	68	第 8 章 构建 cocos2D 增强现实游戏	104
5.2.5 通过震动进行用户反馈	68	8.1 概述	104
5.3 录音	69	8.2 创建工程	105
5.4 总结	71	8.3 创建游戏菜单	108
第 6 章 摄像头和视频采集	72	8.3.1 原图	110
6.1 快速浏览	72	8.3.2 辅助代码目录	113
6.2 拍照	73	8.3.3 完成菜单屏	113
6.2.1 使用故事板	74	8.4 添加菜单选项	120
6.2.2 使用摄像头	76	8.5 完成动作层	129
6.2.3 以不同的格式保存图像	78	8.6 南瓜来了	130
6.2.4 通过电子邮件发送图像	79	8.7 结束游戏	134
6.3 视频捕获	81	8.8 总结	136
6.3.1 建立一个视频预览基础	81	第 9 章 第三方增强现实工具包	138
6.3.2 为帧捕获建立基础	82	9.1 概述	138
6.4 总结	86	9.2 Powered by String 框架	138
第 7 章 把 cocos2D 用于增强现实	88	9.2.1 String 的基本工作流程	139
7.1 概况	88	9.2.2 额外功能	140
7.2 安装	88	9.2.3 整合 Unity	141
7.2.1 安装工程模板	89	9.2.4 高级着色和 OpenGL 功能	141
7.2.2 创建工程	90	9.3 Qualcomm 软件开发工具包	142

9.4 建立我们自己的 QCAR 演示	144	12.1.1 OpenCV	204
9.4.1 创建 Xcode 工程	146	12.1.2 iOS 5 的 CIDetector 类	204
9.4.2 EAGLView	147	12.1.3 face.com	205
9.4.3 重定向 UIView	158	12.2 使用 OpenCV 的方式	205
9.5 ARKit	160	12.2.1 为测试捕获图像	205
9.6 总结	160	12.2.2 哈尔级联分类器	209
第 10 章 使用 OpenGL ES 创建基于 标记的增强现实应用	161	12.2.3 OpenCV 综述	214
10.1 建立标记	161	12.3 使用 CIDetector 类的方式	215
10.1.1 我们的标记	161	12.4 使用 face.com API 的方式	217
10.1.2 OpenGL ES	162	12.4.1 faces.detect API 的调用	217
10.2 创建工程	162	12.4.2 添加 face.com 支持到例子中	218
10.2.1 添加 String 框架	162	12.4.3 face.com API Key	218
10.2.2 EAGLView	163	12.4.4 添加 face.com Callout	219
10.2.3 创建增强现实视图控制器	168	12.5 测试性能	222
10.3 总结	172	12.6 总结	229
第 11 章 构建社交型的增强现实应用	173	第 13 章 建立一个面部识别增强现实 应用	231
11.1 快速设置	173	13.1 应用的目的	231
11.1.1 创建 Facebook 应用	173	13.2 快速设置	232
11.1.2 克隆 Facebook iOS SDK	174	13.2.1 face.com	232
11.2 词汇表	175	13.2.2 cocos2D	233
11.2.1 方位角	175	13.2.3 建立 Twilio 账户	233
11.2.2 矫正方向	175	13.2.4 下载 ASI-HTTP-Request 库	233
11.3 构建应用	176	13.2.5 JSON 框架	233
11.3.1 致谢	176	13.3 工程结构	234
11.3.2 所需框架	176	13.4 建立主场景	235
11.3.3 添加 Facebook iOS SDK	176	13.5 face.com API	245
11.3.4 开始编码	176	13.5.1 使用 ASI-HTTP-Request 库	246
11.3.5 监听传感器更新	181	13.5.2 创建 POST 请求方法	247
11.3.6 存储坐标	183	13.5.3 创建 NSTimer	249
11.4 添加社交上下文	193	13.5.4 解析输出	251
11.5 总结	203	13.5.5 构造 HUD 层	254
第 12 章 面部识别技术	204	13.6 添加一个 Twilio 调出	259
12.1 面部识别的可选项	204	13.7 总结	260

第①章

引言

欢迎来到专业 iOS 5 增强现实的世界。增强现实 (Augmented reality, AR) 已经在科幻电影中应用几十年了, 在军事上被用于头部显示器 (head-up display, HUD) 中, 从现在直至未来它都将存在。从引入 iPhone 和 Android 操作系统以来, 随着移动应用的增加, 像 Layaar (www.layar.com)、Metaio 的 Junaio (www.junaio.com) 以及 Wikitude (www.wikitude.com) 已经把增强现实融入到日常生活中。本书将引导你为 iOS 创建你自己的增强现实应用。

《时代》杂志把增强现实指定为 2010 年的十大技术趋势。不过《时代》杂志仅仅触及了增强现实应用的表面。它们挑选了一些供应商的应用平台, 如 Layaar, 还讨论了一些更日常的应用, 例如美国邮政管理局 (United States Postal Service, USPS) 使用的增强现实应用。

1.1 增强现实的应用实例

2010 年, 美国邮政管理局引入了一个增强现实的应用到它的网站。如果你以前从邮局邮寄过东西, 你就知道, 不用排队而选择一个符合你需求的箱子是几乎不可能的。要么匆忙地选一个浪费很多空间的大箱子, 要么费劲地找一个合适的箱子把所有的东西塞进去, 而这样的话往往会导致排队现象。美国邮政管理局做了一个不需要你离开家或者办公室就能让这更容易实现的尝试。基本方法是, 在你去邮局之前, 可先到美国邮政管理局的网站 (www.prioritymail.com), 使用虚拟箱子模拟器和你的网络摄像头来测试不同大小的箱子。其工作流程如下:

打印出一个特定的图标 (美国邮政管理局的鹰图标) 以使模拟器知道虚拟箱子的全息图放在哪里, 如图 1-1 所示。

1) 确保你开启了网络摄像头。

2) 打开虚拟箱子模拟器。把打印的图片放入网络摄像头的视野内, 然后模拟器把货运纸箱的各种选项的全息图放在图片的周围, 如图 1-2 所示。

当你创建图标或者为识别做标记的时候有一些原则需要遵守。按标记的惯例, 你需要一个带有某个特定性并且一般不常见的高对比度的对象。事实上, 随机图片经常更有效。此外, 你需



图 1-1 美国邮政管理局根据这个打印的图标, 用一个货运纸箱来增强摄像头视图

要使用有一定的旋转并且水平和垂直方向上都不对称的图片。这有助于 AR 程序识别方向并做相应的调整。美国邮政管理局的图标是这些原则的一个好例子。

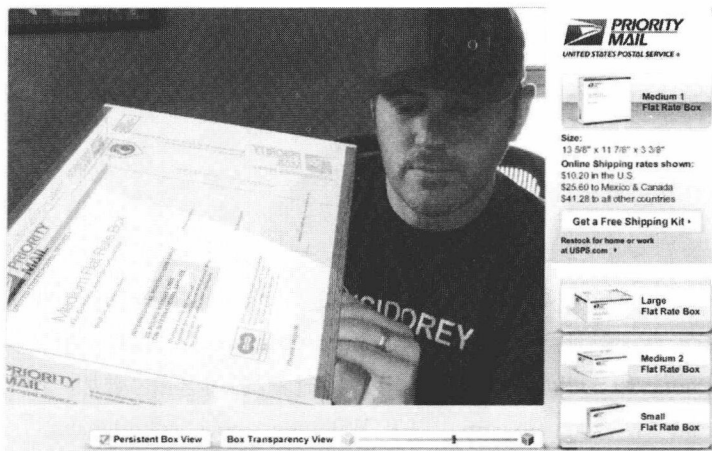


图 1-2 全息图正覆盖在打印的图标上

注意：

如图 1-2 所示，模拟器允许你调整透明度，以不同的角度移动和旋转你要运输的物品，以及完全体验你需要哪一个货运纸箱来运输你的物品。美国邮政管理局使用标记和某种识别算法在实时的摄像头视图中发现待运物品，跟踪它的方向，并用当前你选择的箱子增强图片。

1.1.1 日常应用的趋势

在广告、房地产、汽车行业，尤其是在消费支出中存在着数以百计的增强现实应用。虽然一项统计显示有超过半数的美国人已经尝试了网上购物，但根据维基百科，增强现实应用的收益只占总消费支出的 8%。显然，对于为什么没有增加更多的市场份额有各种各样的推测。其中包括在线隐私和安全等基本问题，但更重要的原因是缺乏对一个未知产品的物理交互。例如衣服，你就是需要看到和触摸到你正在买的东西。

2010 年年底，我们开始看到多个增强现实的体验渗透进零售市场。成长在 70 年代末的我，记得 Jane Jetson 通过按下一个新按钮来尝试新的发型，还记得 Luke 天行者通过一个全息三维的显示器来收听关于死亡之星的交涉方法的简讯。现在，这种类型的体验对于消费者来说已经是可行的了。从尝试新的服装及佩饰，到找出杂货店里的苹果是从哪里生长的，看一下最近的一些例子：

- 乐高的数字盒子：一个乐高的店内厅可以让孩子们在售货厅的摄像头前举起一个他或者她看中的套装，然后显示器中的盒子顶部会显示出完全组装好的套装。孩子们能够移动它、转动它，并获得一个感受，来决定这是不是它们圣诞节列表里面的那个套装。

- **Zugara** : Zugara 使用其魔镜, 可以让在线购物者不用键盘和鼠标的辅助, 站在网络摄像头前面尝试不同款式的衣服。除了从在线目录中试衣服, Zugara 还在摄像头视野内添加了控制功能, 这样用户就能使用手势去和目录选项进行交互或者在它们的社交网络里面分享它们的新套装。
- **FoodTracer**: 这项工程是 Giuseppe Costana 利用增强现实中的图像识别技术来为杂货店的购物者提供他们正要购买的东西的更多相关信息。购买者只需要在杂货店的货架前简单地用移动设备的摄像头对准货物, 信息就显示出来。

交互体验有明显的优势和吸引力。然而, 增强现实还有一些附加的价值。大部分这些应用的后台放置在云端。图像识别算法和摄像头显示主要在设备上运行, 而广告数据、上下文信息、位置目录和其他与增强现实视图相关的动态内容可以从云端加载, 并且在云端处理中心更新是无缝的, 因而应用能够永远保持最新。

1.1.2 游戏和基于位置的应用

并不是只有在商业中增强现实成为一种趋势。社交网络、基于位置的服务以及游戏同样正在逐步应用增强现实。想象一下, 在一个游戏情节里面, 你用摄像头与真实世界进行交互。我最近在一个会议上看到一个演示, 在这个演示里, 僵尸的 3D 模型被呈现在一个 iPhone 的增强现实视图中, 用户只需通过点击屏幕就可以射击它们。它催生了一个如 iPhone 枪配件一样的二级市场, 可参见 www.augmentedplanet.com。将这个步枪大小的配件添加到你的 iPhone 的视野中, 然后你就可以在一个增强现实的模式中有射击 3D 僵尸的真实体验。

在本书中, 我们将会讲解创建增强现实游戏的基础, 包括实现这个工程的各种方法, 以及能够加快产品上市时间的一些软件开发工具包。

1.2 准备工作

你需要做一些步骤以确保你的机器已经完全准备好用来进行 iOS 编程。在本书中, 我们只使用 Xcode 4.2, 并且我们会保存所有的工程到 GitHub。Xcode 集成了本地 Git 源代码管理功能, 所以使用它会使事情变得更容易, 还可以节省安装时间。

1.2.1 注册 GitHub 账户

如果你已经有一个 GitHub 账户, 你可以跳过本节。如果没有, 你将需要一个账户来下载每章的资源。打开浏览器, 转到 www.github.com 网站, 点击网页中央的 Signup 按钮, 如图 1-3 所示。

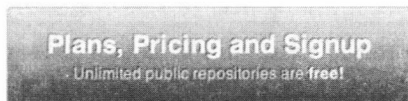


图 1-3 GitHub 的注册按钮

本书中，我们要访问已经为每章创建的 Git 仓库；并且如果你加入了共享，任何的改动我们都会告知读者。考虑到这一点，我们确实只需要“Free for open source”（开源免费）的账户类型。点击“Create a free account”（创建一个免费账户）按钮，然后填上你的信息。

1.2.2 从机器访问 GitHub

如果你以前用过 GitHub，你可以跳过本节，本节是针对想使用 GitHub 但没有创建 SSH 密钥的用户。

从机器访问 GitHub 远程仓库有多种方式。我们将会使用 SSH 访问，这意味着需要生成一个令牌并把它发送给 GitHub。从你的 Mac 中打开终端（Applications → Utilities → Terminal）。在终端窗口中按代码清单 1-1 显示的命令运行。接下来我将会解释各个步骤。

代码清单 1-1 在 Mac 上创建 SSH 密钥

```
Kyle-Roches-MacBook-Pro-2:~ kyleroches$ cd ~/.ssh
Kyle-Roches-MacBook-Pro-2:~.ssh kyleroches$ ls
known_hosts
Kyle-Roches-MacBook-Pro-2:~.ssh kyleroches$ ssh-keygen -t rsa -C "kyle@isidorey.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/kyleroches/.ssh/id_rsa):
Enter passphrase (empty for no passphrase): [enter a passphrase here]
Enter same passphrase again: [enter your passphrase again]
Your identification has been saved in /Users/kyleroches/.ssh/id_rsa.
Your public key has been saved in /Users/kyleroches/.ssh/id_rsa.pub.
The key fingerprint is:
26:9d:3a:82:fe:r9:gf:ba:39:30:6b:98:16:fe:3b:2c kyle@isidorey.com
The key's randomart image is:
+--[ RSA 2048 ]-----+
|
| . . . . 4
| . . . .N
| . o ..+r
| .. ...-|=
| ..+.-E.o
| +.==0000
+-----+
Kyle-Roches-MacBook-Pro-2:~.ssh kyleroches$ ls
id_rsa      id_rsa.pub  known_hosts
```

如果已经存在密钥，那么目录列表的命令可能会有不同的结果。在这种情况下，你可能要备份你的密钥目录，这样做仅仅是为了安全。首先，我们使用 `ssh-keygen` 实用程序创建一个公共/私人 rsa 密钥对。该实用程序会询问我们一个口令。这是可选的，但是口令确实可以提高安全性。在我们大多数人心目中，仅凭密码来保证安全并不是那么可靠。

生成一个没有口令的密钥对相当于把密码存储在你机器的一个明文文件中。现在任何获得权限的人都可以使用你的密钥。如果你懒于或烦于每次都要输入密码，请不要烦恼。Keychain（因为我们都是使用的 Mac）允许你在第一次使用这个密钥对的时候把它存储起来。

然后，你在目录列表中可以看到我们有一个密钥对，它保存在一个新创建的文件 `id_rsa.pub` 里面。在纯文本编辑器里面打开这个文件并复制它的所有内容。重要的是，你要复制所有内容（甚至是标题）。

返回 GitHub 网站，需要在你的浏览器中打开并转到你的账户页面。从左上部导航菜单打开 Account Settings（账户设置）页面。然后打开左手边的 SSH Public Keys（SSH 公共密钥对）的子标签。你将看到类似于图 1-4 的页面。

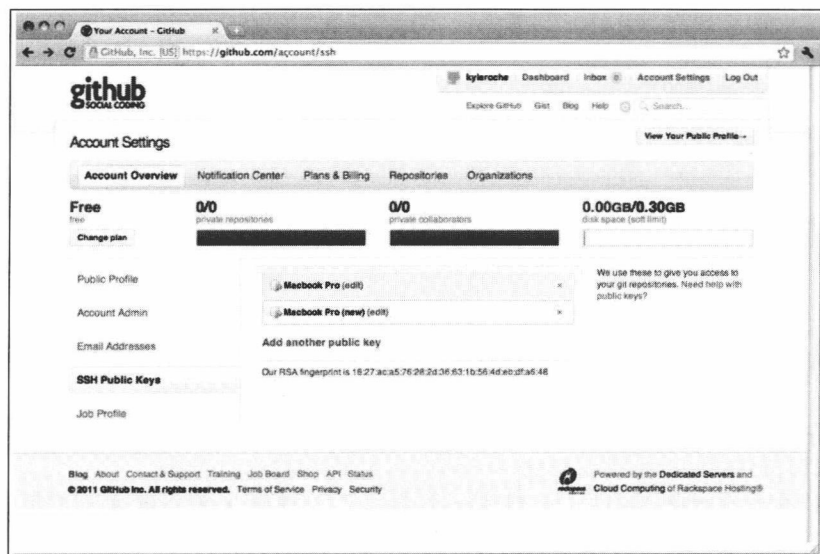


图 1-4 打开 GitHub 上的 SSH 公共密钥对话框

在页面中央找到 Add another public key（增加另一个密钥）链接。这将打开一个对话框，在对话框里面你可以粘贴我们刚刚创建的 `id_rsa.pub` 文件的内容。就是这样！你现在已经在 GitHub 上创建了仓库，机器可以通过 SSH 访问你的仓库。

因为我们在本书中将会使用 SSH 访问，在继续之前，先快速建立起默认首选项。

我们需要设置本地的 Git 客户端，使用在注册 GitHub 时收到的证书。首先，在终端窗口中运行代码清单 1-2 中的命令来为 Git 建立一些全局标记。全局标记结合 SSH 密钥会向远程仓库证明你的 Git 客户端。

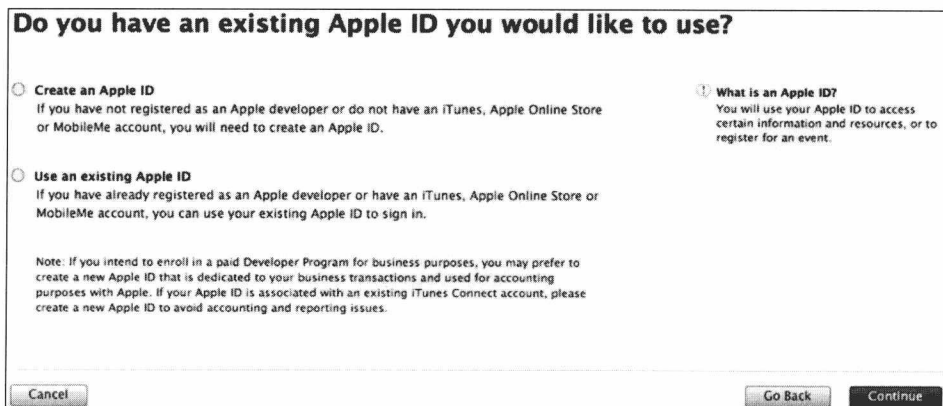
代码清单 1-2 在 Mac 上创建你的 SSH 密钥

```
Kyle-Roches-MacBook-Pro-2: kyleroches$ git config --global user.name "Kyle Roche"
Kyle-Roches-MacBook-Pro-2: kyleroches$ git config --global user.email "kyle@isidorey.com"
```

1.2.3 安装 Xcode 4.2 和创建开发者账户

如果你已经安装了 Xcode 4.2，你可以跳过本节。

为了向 App Store（应用商店）发布应用，你需要 Xcode 和一个苹果开发者账户。我们可以在同一时间完成这两个步骤。打开你的浏览器转到 <http://developer.apple.com/programs/register/> 并点击顶部的 Get Started 按钮。这里有多条路径可以进入。如果你想要使用一个已有的苹果 ID，你可以填入并继续，参见图 1-5。相反的，你也可以为你的 iOS 开发创建一个新的 ID。这似乎不太合理，但是只使用一个账户有很多缺陷。



Do you have an existing Apple ID you would like to use?

☒ **Create an Apple ID**
If you have not registered as an Apple developer or do not have an iTunes, Apple Online Store or MobileMe account, you will need to create an Apple ID.

☐ **Use an existing Apple ID**
If you have already registered as an Apple developer or have an iTunes, Apple Online Store or MobileMe account, you can use your existing Apple ID to sign in.

Note: If you intend to enroll in a paid Developer Program for business purposes, you may prefer to create a new Apple ID that is dedicated to your business transactions and used for accounting purposes with Apple. If your Apple ID is associated with an existing iTunes Connect account, please create a new Apple ID to avoid accounting and reporting issues.

What is an Apple ID?
You will use your Apple ID to access certain information and resources, or to register for an event.

Cancel Go Back Continue

图 1-5 使用已经存在的苹果 ID 或者创建一个新的

注意：

选择扩展已有的苹果 ID 还是创建一个新的 ID，这取决于将来发布应用时你的意图。对于一个账户能够链接的发布类型，苹果有所限制。有两种方式来发布应用：通过 App Store 或者通过苹果的企业分发程序。一个苹果 ID 不能够同时绑定两种发布方式。如果两种情况你都想要包括的话，请确保你决定了哪个 ID 负责哪种发布方式。

如果你只是想要使用账户来做开发和调试，那么可以使用一个已有的账户。这可能是最简单的方式了。在你注册之后，登录 iOS Dev Center，找到 Downloads 链接。在写这本书的时候只有两个选择：Download Xcode 4.2 和一系列的关于 iAd Producer 1.1 的链接。下载 Xcode 4.2 到你的电脑，下载是相当大的。这是 Xcode 的缺点。每一次版本更新都需要重新下载整个 IDE，而且从 iOS 更新越来越频繁。

我们现在已经有了 IDE 并设置了源码管理策略。我们将两者结合，并确保我们已经为开始做好了准备。

1.2.4 连接 Xcode 工程到 GitHub

在浏览器中打开 GitHub 网站，点击左上角导航栏的 Dashboard 按钮，然后找到 New Repository（新的仓库）按钮，在 Project Name（工程名）里面填上 iOS_AR_Ch1_Introduction。你可以随意选择自己的名字，或者你是一个有经验的 GitHub 用户，也可以以我的仓库为基础创建一个分支，网址是 <https://github.com/kyleroché>。图 1-6 显示了我选择的选项。

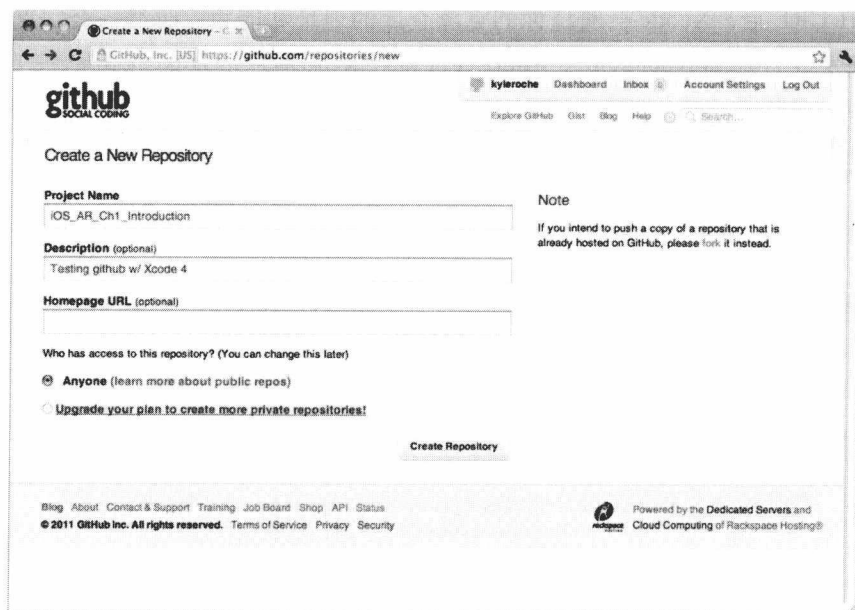


图 1-6 在 GitHub 创建一个新的仓库

下一步，记下仓库的 SSH URL，你将在确认页面的顶部看到它。它将以类似这样的形式出现：git@github.com:kylerocher/iOS_AR_Ch1_Introduction.git，在下一步中你会需要它。

在你的本地电脑里打开 Xcode，在启动时的 Welcome to Xcode（欢迎来到 Xcode）对话框的左栏中有一个 Connect to a Repository（连接到一个仓库）的选项。点击该选项，然后填入 GitHub 仓库的 SSH URL。图 1-7 显示了我的配置。

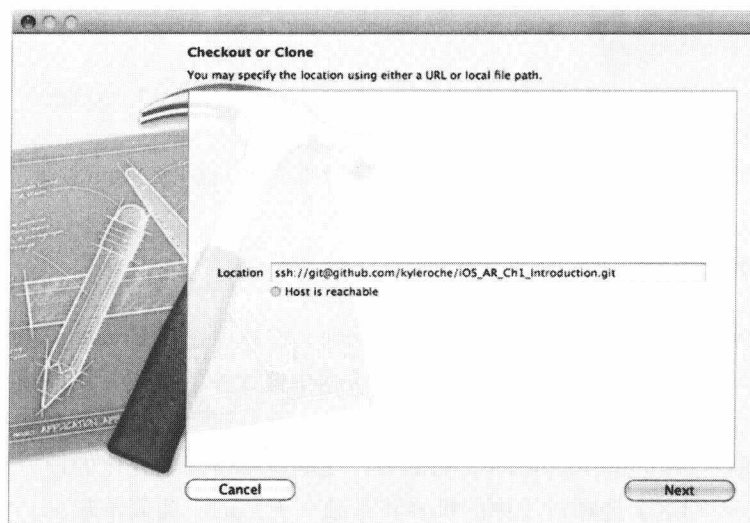


图 1-7 为本地访问克隆你的 GitHub 仓库

填入 SSH URL 后, Xcode 会验证你的克隆仓库的地址和能力。等待一会, 直到指示绿色并且消息状态变成 Host is reachable (主机已经准备好), 然后点击 Next 按钮。

你会看到一个提示, 请你为新工程命名。为简单起见, 我使用与 GitHub 仓库相同的名字, iOS_AR_Ch1_Introduction。确保 Type 已设置为 Git, 然后点击 Clone 按钮。

下一步, 为你的本地仓库选择一个存储位置, 然后点击 Next 按钮。

注意:

在编写本书的时候, Xcode 4.2 在使用 Git 的时候仍然有一些小错误。首个错误应该会在本例子的最后一步中显现。如果你的版本仍然有问题, 你将会得到一个类似图 1-8 的错误提示。如果是这种情况, 点击 Try Again 按钮, 选择相同的位置, 选择 Replace 选项后一切应该会恢复正常。

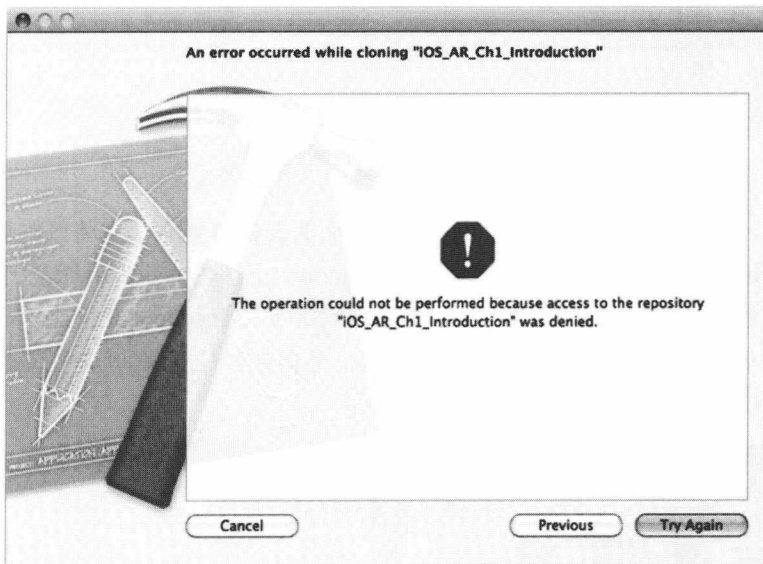


图 1-8 在早期的 Xcode 4.2 版本中的缺陷之一: 抛出无效错误。点击 Try Again 按钮, 该错误就会消除

1.2.5 创建 Xcode 工程

从 Xcode 的 Welcome to Xcode (欢迎来到 Xcode) 窗口中选择 Create a New Xcode Project (创建一个新的 Xcode) 选项。我们将不会在这个工程中做太多的编码, 所以模板类型不是太重要。为简单起见, 我选择了一个基于窗口的应用模板。接着你要填写产品名称, 该名称用来作为完全限定包标识符 (Bundle Identifier) 的后缀。这就是事情开始变化的地方。除非你参与了基于团队的开发, 否则这个选项对于你的机器将是独一无二的。为简单起见, 我将再次使用与 GitHub 仓库相同的名字。我的设置如图 1-9 所示。

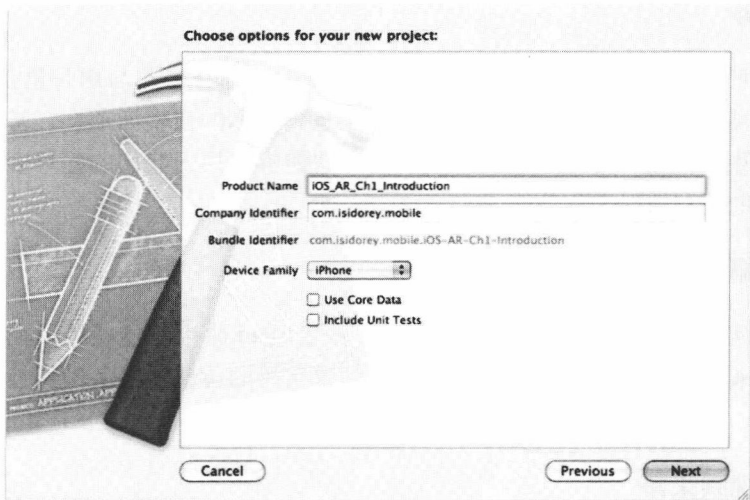


图 1-9 新工程选项设置

你的公司标识也将与众不同。在我们讨论是通过 App Store（应用商店）还是通过企业分发选项来分发程序之前，公司标识可以是一个反向的 DNS 格式。当你填完了所有的选项，点击 Next 按钮。

你最后会被提示为本工程寻找一个本地存储位置。确保你选择了与克隆 GitHub 仓库时相同的目录。同时，还要确保 Create local git repository for this project（为此工程创建本地 git 仓库）处于未选择状态，如图 1-10 所示。

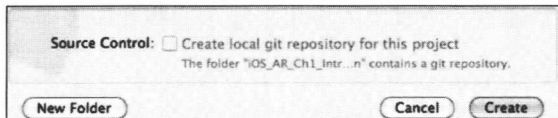


图 1-10 不要创建本地 Git 仓库

非常好！我们刚刚创建了我们的第一个工程。仅仅再需要几个步骤我们就可以向 GitHub 上传代码了。接下来的几个步骤可以按不同的顺序来完成。我按这个过程进行是因为我发现这样更容易。随着 Xcode 4.2 的成熟，我相信会看到一些基于 GUI 的功能改进。

1.2.6 连接工程到远程仓库

有相当多的关于结合 Xcode 和 GitHub 的在线教程。想要开始连接你的工程，并且想要学习最新的功能和更新，请访问这个网址：<http://help.github.com/mac-set-up-git/>。

1.3 下一步做什么

本节将对本书的一些关键章节进行简单介绍。

1.3.1 位置服务

大部分增强现实的应用案例，至少在早期的案例中，都有一些与用户位置相关联的元素。无论是在增强现实的视图中呈现的附近的餐厅，还是寻找你的夜生活朋友，基于位置的增强现实引发了这个趋势。在第 3 章中，我们将会使用一些能够应用于我们的程序中的真实例子，以此来学习 iOS 的地图和地理定位功能。

1.3.2 传感器编程

创建增强现实应用需要结合相当多的 iPhone 或者 iPad 原生的传感器。在第 4 章中，我将通过几个小工程来演示在我们的增强现实应用中频繁重用的传感器的关键特性，以此来向你介绍传感器开发。

1.3.3 声音和视频采集

在第 5 章中，我会讲解声音和用户反馈。声音并不是增强现实应用最显著的特性，但是它却实实在在地让用户体验变得更好。之后，在第 6 章，我们会深入介绍摄像头和视频编程。因为增强现实应用主要是基于摄像头视图，这是本书末尾我们构建大型增强现实应用之前需要理解的基础章节。

1.3.4 游戏框架

我选择使用 cocos2D 来演示增强现实的游戏潜力。在第 7 章，你会接触到 cocos2D 的一些基础的知识，然后在第 8 章通过一个真实的应用来展示这些知识。

1.3.5 第三方框架

在第 9 章，我会讨论一些能够让基于标识（marker-based）的增强现实应用的开发更容易的第三方框架。接着，我们举一个真实的例子，然后介绍更加复杂的框架，例如 OpenCV（Open Computer Vision，开源计算机之眼），是一个用于面部和复杂对象识别的开放源代码的类库。基于设备自身的面部识别有很多限制，大部分情况下，这些限制源于设备性能。我们将会讨论更多的创造性方法，例如使用公共可用的 API 来弥补面部识别的局限。

1.4 总结

我希望你从本书中学到很多知识和技能。增强现实是移动应用的一个新概念并且有无限的潜力。开发者社区仅仅开始揭开了其潜力的一个表面。我希望本书为你进入增强现实世界的旅程提供一个良好开端。

我们首先回顾一些我们的应用中的布局选项和框架以组合自己的增强现实应用。在第 2 章中，我们会讨论本书中即将用到的硬件和各种产品型号的主要特性。

第②章

硬件比较

每一个移动开发者都关心硬件的兼容性。但是，苹果 iOS 产品线开发的主要优势就是苹果硬件之间是标准化的。当然，也有多个不同分辨率的设备，但是却只有一个供应商：苹果！在其他移动操作系统中，你必须关注 OEM 供应商和它们硬件构造的无限变化。让我们仔细了解一些最近的 iOS 设备。

2.1 除旧存新

我们将在 iPhone 和 iPad 中测试这个例子工程。但是无论何时，代码都是平台间完全通用的，所以我们只在 iPhone 下编写测试代码即可。图 2-1 显示的是 iPhone 4 的物理尺寸。

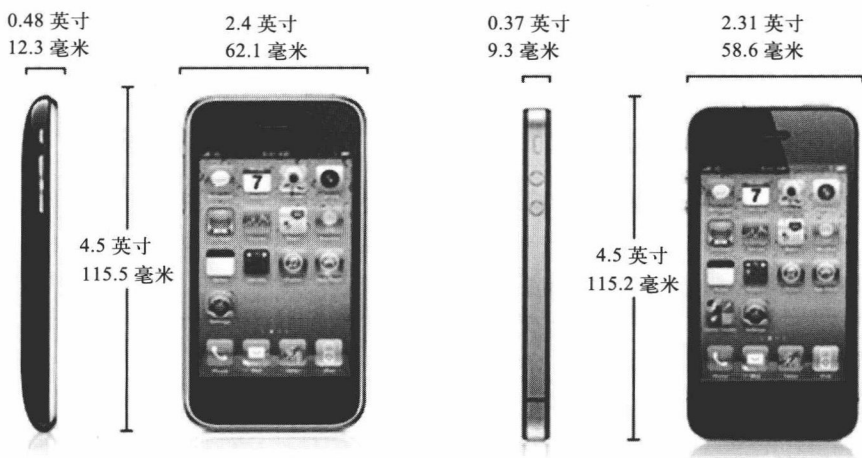


图 2-1 iPhone 4 的尺寸

iPad 在市场上的出现时间晚于 iPhone，它们的存在目的也并不相同。在本书中，我们将只使用 iPad 2 来演示例子。原因有多个，最重要的一点是 iPad（第一代）没有前置摄像头，而我们需要在本书的后面演示一些面部识别的程序。图 2-2 展示了 iPad 2 的规格参数。

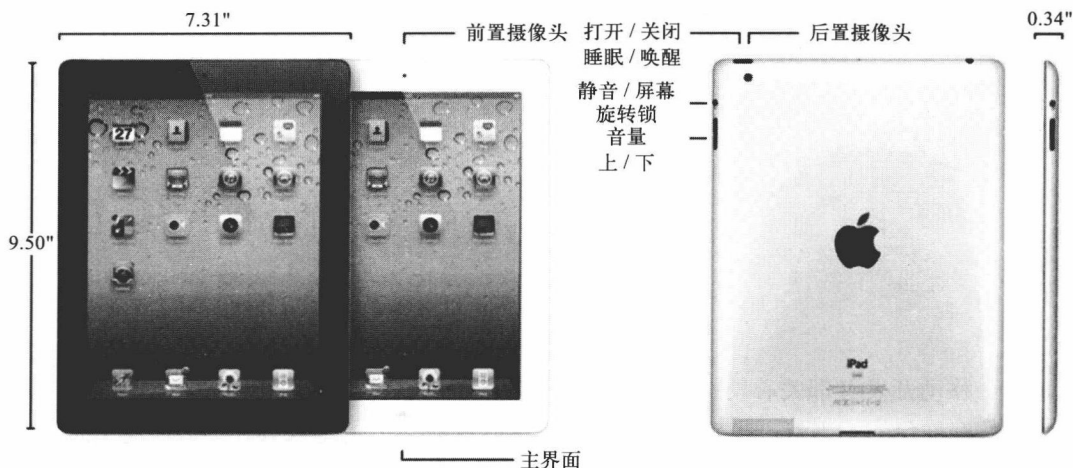


图 2-2 iPad 2 的规格参数

2.2 硬件组件

首先，我们考虑一下一个增强现实的应用程序会需要些什么，而不是仅仅列出所有这些设备的硬件规格。最明显的，需要一个摄像头，大部分增强现实案例程序都需要确定位置、方向，所以 GPS 和指南针将非常有用。声音输入/输出也是非常重要的，所以需要麦克风和扬声器。我们将会在例子中融入一些游戏元素，所以可能需要一些硬件加速计和一个图形工具包。让我们把这些翻译成苹果的术语，并且逐个说明这些各种各样的硬件组件。

注意：

当我们查看 iOS 设备的各种不同的硬件组件的时候，我将会发布一些小的代码片段。在本章中，它们将会显得断章取义，但是如果你想继续深入，所有的代码都在 GitHub 的仓库中 <https://github.com/kyleroché>。

2.2.1 摄像头支持

从 iPhone 3GS 投放市场开始就有摄像头了。虽然与其他以摄像头而出名的硬件设备比较起来还有很多可以改进的空间，但是这足以满足本书的需求。

利用手机的摄像能力，有两种方法可以用来创建增强现实的应用程序。第一种，你可以实时地利用视频捕获来检查元素、识别物体等。第二种，你可以使用视频采集作为你的应用程序的背景，同时完全地忽略内容。由于需要大量地处理实时的视频采集，所以这种方式我们在增强现实的浏览器中看到很多。在本书中，这两种方法我们都会讲到。

表 2-1 详解了本书要用到的设备的拍照和录像能力。

表 2-1 iPhone 3GS、iPhone 4 和 iPad 2 的摄像头详情

	iPhone 3GS	iPhone 4	iPad 2
后置摄像头录像能力	VGA，每秒 30 帧带音频	高清（720 像素），每秒 30 帧带音频	高清（720 像素），每秒 30 帧带音频
前置摄像头录像能力	—	VGA，每秒 30 帧带音频	VGA，每秒 30 帧带音频
后拍照能力	300 万像素	500 万像素	高清拍照，5 倍数数码变焦
闪光灯	—	LED 闪光灯	—

监测摄像头

通过 UIImagePickerController 类，我们可以编程检查设备上的摄像头是否可用。isSourceTypeAvailable 方法可以判断我们想要使用的摄像头类型在当前设备上是否可用。代码清单 2-1 显示了一个使用 isSourceTypeAvailable 方法的例子。

代码清单 2-1 监测有没有摄像头，然后再监测有没有前置摄像头

```

BOOL cameraAvailable = [UIImagePickerController
isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera];
    BOOL frontCameraAvailable = [UIImagePickerController
isSourceTypeAvailable:UIImagePickerControllerCameraDeviceFront];

    if (cameraAvailable) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Camera"
            message:@"Camera Available"
            delegate:self
            cancelButtonTitle:@"OK"
            otherButtonTitles:nil, nil];

        [alert show];
        [alert release];
    } else {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Camera"
            message:@"Camera NOT Available"
            delegate:self
            cancelButtonTitle:@"OK"
            otherButtonTitles:nil, nil];

        [alert show];
        [alert release];
    }

    if (frontCameraAvailable) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Camera"
            message:@"Front Camera
Available"
            delegate:self
            cancelButtonTitle:@"OK"
            otherButtonTitles:nil, nil];

        [alert show];
        [alert release];
    } else {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Camera"
            message:@"Front Camera NOT
Available"
            delegate:self
            cancelButtonTitle:@"OK"
            otherButtonTitles:nil, nil];

        [alert show];
        [alert release];
    }
}

```

这段代码会通过一个 UIAlertView 的快速弹出窗口来显示结果。实际上,在这个例子中你将看到两次弹出窗口的提示。在代码清单 2-1 中的刚开始几行,我们是在检查 UIImagePickerControllerSourceTypeCamera 是否存在,以此来判断摄像头是否可用。接下来使用 UIImagePickerControllerCameraDeviceFront 参数来检查是否存在前置摄像头。isSourceTypeAvailable 方法返回一个布尔值。我们使用这个布尔值在 if/else 语句中做判断,然后在 UIAlertView 中为每次检查显示恰当的结果。

现在,当我们使用 iPhone 3GS、iPhone 4 和 iPad 2 的时候,为什么还是必须要检查摄像头是否存在?它们不是都有摄像头吗?是的,它们都有摄像头。但是在 Xcode 中,我们也要在模拟器中编写代码。与其他移动操作系统的 IDE 不同的是,Xcode 的模拟器并不支持摄像头。

图 2-3 这两个屏幕截图是来自运行代码清单 2-1 的 iPhone 4 设备。因为在代码中的两个 if/else 代码块都运行了,所以我们得到了两个 UIAlertView 提示对话框。从对话框中你可以看到两个检查的返回值都是 True,这意味着摄像头和前置摄像头都是可用的。

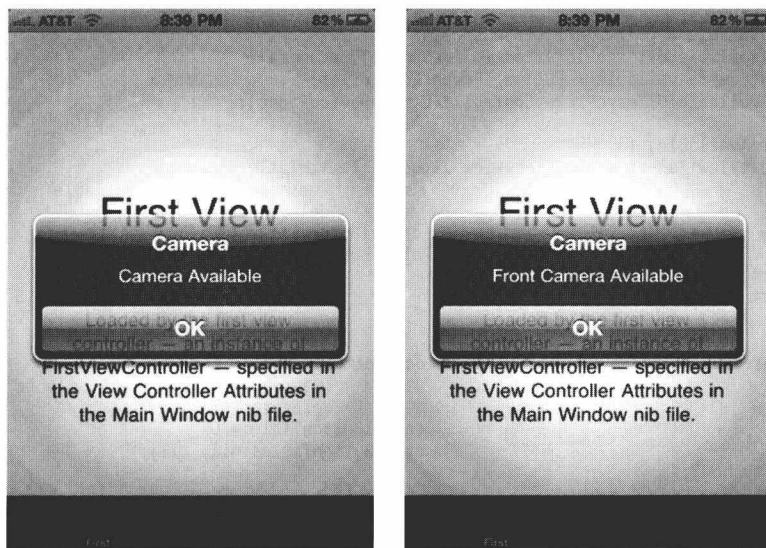


图 2-3 在 iPhone 4 上检测摄像头(左)和前置摄像头(右)的屏幕截图

为了证明这些函数确实检查了摄像头的可用性,我在 Xcode 的 iPhone 4.3 的模拟器中运行了相同的代码。最后对话框中的结果却是相反的,如图 2-4 所示。

我们也可以使用一些其他选项来检查摄像头。从技术上讲,第一次检查是为了寻找任何可用的相机。我们可以使用 UIImagePickerControllerCameraDeviceRear 作为源类型增强参数,并检查后置摄像头。或者,我们可以检查设备上一些关于闪光灯设置的选项。这有点不同,要么我们必须具体地检查将要验证的模式,如 UIImagePickerControllerCameraFlashModeOn,要么我们可以使用 isFlashAvailableForCameraDevice 方法。

在第 8 章,我们将会更多地讨论怎样使用摄像头来捕获图片和视频。



图 2-4 在 iPhone 4.3 的模拟器上检测摄像头（左）和前置摄像头（右）

2.2.2 位置检测能力

iOS 为与设备上的位置服务和硬件做交互，提供了 Core Location Framework。遗憾的是，Core Location Framework 没有提供任何对 GPS 存在性的监测。所以需要在应用中强制硬件需求（参见本章 2.3 节）。

现在，为了简单起见，你仍然能够检查位置服务是否已经开启。虽然位置服务在所有的 iOS 版本中都是可用的，但仍然有一些情况对于你的应用来说位置服务是不可用的。例如：

- 用户在应用设置里面关闭了位置服务。
- 用户拒绝了某个特定应用的位置服务请求。
- 设备处在飞行模式，并且无法启动必要的硬件。

由于这些原因，你应该在试图使用位置服务之前总是调用 `CLLocationManager` 的 `locationServicesEnabled` 类方法。如果用户有目的地关闭了这些服务，这时你试图使用位置服务的时候出现的自动提示应该会影响用户体验。

有两种确定用户位置的方法可用：

- 标准位置服务（Standard Location Service）是一个可设置的、通用的、支持所有 iOS 版本的解决方案。

- 显著变更位置服务（Significant-Change Location Service）使用蜂窝无线技术为设备提供一个低功率的位置服务。这个服务只支持 iOS 4.0 及其以后版本，能够唤醒暂停或者没有运行的应用。

在第 4 章，我们将会详细讨论怎样开启和使用位置服务。

2.2.3 数字指南针

我们已经讨论了增强现实的应用应该可以从导向能力中受益。事实上，这也是对任何一个基于位置的增强现实应用的要求，否则你无法确定用户正面向哪一个方向。监测数字指南针是非常简单的。在你能够使用 Core Location Framework（核心位置框架）之前，需要进行以下两步准备工作。

首先，你必须为应用连接核心位置（Core Location）类库。在 Xcode 的 Project Navigator（项目导航器）中单击工程名字。切换视图到应用的目标的 Build Phases（构建阶段）选项卡。有一个叫做 Link Binary With Libraries（用库链接至二进制）的区域。展开这个区域，单击“+”按钮来添加一个新的类库。在图 2-5 中，你能够看到正在添加到工程中的 Core Location Framework。

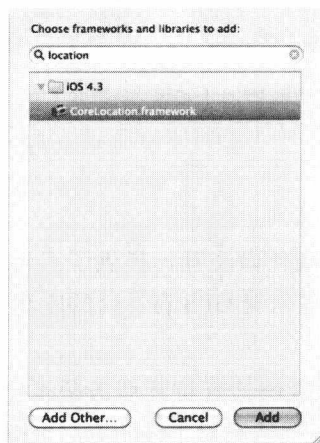


图 2-5 添加 Core Location Framework 到二进制应用中

其次，确保将代码清单 2-2 中的 import 语句添加到你的头文件中。

代码清单 2-2 导入 Core Location Framework

```
#import <CoreLocation/CoreLocation.h>
```

在你添加了 Core Location Framework 后，可以添加代码清单 2-3 到代码清单 2-1 的代码片段中来监测指南针的存在。所有的这些代码都属于 .m 文件。注意，我们正在使用 headingAvailable 类方法，而不是属性。最近属性方法被取而代之，类方法是监测指向是否可用的首选方式。

在发布到 GitHub 的那个例子中，我把所有的这些代码放到 viewDidLoad 方法中，以确保在视图加载完成后再运行这些代码。

代码清单 2-3 检查指南针

```

BOOL magnetometerAvailable = [CLLocationManager headingAvailable];
if (magnetometerAvailable) {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Magnetometer"
                                                         message:@"Magnetometer
Available"
                                                         delegate:self
                                                         cancelButtonTitle:@"OK"
                                                         otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
} else {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Magnetometer"
                                                         message:@"Magnetometer NOT
Available"
                                                         delegate:self
                                                         cancelButtonTitle:@"OK"
                                                         otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
}

```

就像摄像头，指南针是另一个模拟器不支持的硬件组件。图 2-6 表示的是在 iPhone 4.3 模拟器上运行代码清单 2-3 的代码得出的结果。

如果你在物理 iPhone 或者 iPad 设备上运行这个相同的代码块，你会得到相反的结果。

记住，你仍然能够使用硬件需求来阻止应用运行在一个特定的不支持指南针的设备上。参见 2.3 节以获得关于这个话题的更多详情和指导。在第 7 章我们会讲解数字指南针的更多高级用法。

2.2.4 声音支持

检查音频能力与检查其他组件的方法相同。检查声音需要使用 AVFoundation Framework。按本章早期我们连接 Core Location Framework 的方法连接此框架到你的代码中。接下来，添加合适的 import 语句到你的头文件中，如代码清单 2-4 所示。

代码清单 2-4 添加 import 语句

```
#import <AVFoundation/AVFoundation.h>
```

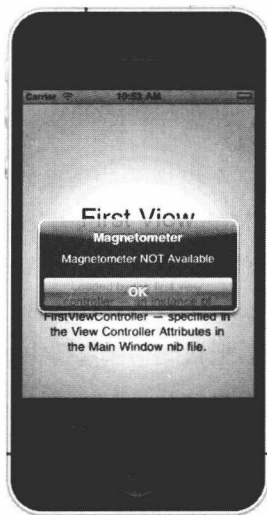


图 2-6 在 iPhone 4.3 模拟器上检查指南针

最终，切换到 .m 文件并反注释 viewDidLoad 方法。或者，如果你想把这个检测声音的代

码放在与检测位置服务和指南针相同的文件下，则可以简单地把代码清单 2-5 中的代码追加进去。

代码清单 2-5 导入 AVFoundation Framework

```
AVAudioSession *audioSession = [AVAudioSession sharedInstance];
BOOL audioAvailable = audioSession.inputIsAvailable;

if (audioAvailable) {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Audio"
                                                         message:@"Audio Available"
                                                         delegate:self
                                                         cancelButtonTitle:@"OK"
                                                         otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
} else {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Audio"
                                                         message:@"Audio NOT Available"
                                                         delegate:self
                                                         cancelButtonTitle:@"OK"
                                                         otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
}
```

2.2.5 检查录像功能

我们已经证实大部分创建一个增强现实的应用所需要的组件都是在设备上呈现的。但是有一个可用的摄像头并不意味着一定能够录像。然而，毕竟录像增强现实的应用之所以如此有魅力的原因。

检查一个摄像头的录像功能要比仅仅检查摄像头的存在性要麻烦得多。要做到这一点，我们需要将 Mobile Core Services Framework（移动核心服务框架）添加到工程中。这与我们检查指南针的时候添加 Core Location Framework 到工程中使用的方式相同，只是名字变成了 MobileCoreServices.framework。在你的头文件中的其他 import 语句后面添加代码清单 2-6 中的代码。然后把代码清单 2-7 中的代码放在 @end 前面。

代码清单 2-6 导入 Mobile Core Services Framework

```
#import <MobileCoreServices/UTCoreTypes.h>
```

代码清单 2-7 声明 isVideoCameraAvailable 方法

```
- (BOOL) isVideoCameraAvailable;

In Listing 2-7, we are declaring a method signature so we can use a helper function to
check for video. Switch to the .m file of your class and add the following helper
function from Listing 2-8.
```

在代码清单 2-7 中，我们声明了一个方法签名，然后就可以用这个辅助函数来检查录像能力。切换到类的 .m 文件，并添加代码清单 2-8 中的辅助函数到 .m 文件。

代码清单 2-8 添加辅助函数到 .m 文件

```

- (BOOL) isVideoCameraAvailable
{
    UIImagePickerController *picker = [[UIImagePickerController alloc] init];
    NSArray *sourceTypes = [UIImagePickerController
        availableMediaTypesForSourceType:picker.sourceType];
    [picker release];

    if (![sourceTypes containsObject:(NSString *)kUTTypeMovie]){
        return NO;
    }

    return YES;
}

```

在这个代码块中，我们简单地检查了所有可用的媒体资源类型，然后检测返回的数组，看里面是否包含一个值为 kUTTypeMovie 的 NSString 类型的对象。如果找到这个值，那么表明这个设备的摄像头支持录像。现在，已经声明过这个方法，因此可以在我们的类中使用这个方法，具体我们可以效仿检查其他组件时所用的相同的方式。在检查摄像头的代码后面添加代码清单 2-9 中所示的代码。

代码清单 2-9 调用新创建的辅助函数来检查摄像头对录像的支持

```

if ([self isVideoCameraAvailable]) {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Video"
        message:@"Video Available"
        delegate:self
        cancelButtonTitle:@"OK"
        otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
} else {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Video"
        message:@"Video NOT Available"
        delegate:self
        cancelButtonTitle:@"OK"
        otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
}

```

如你预料，在模拟器里面你同样得到摄像头不支持录像的结果（因为我们已经确定模拟器中没有摄像头）。

2.2.6 加速计和陀螺仪

有很多像第 1 章中我提到的虚拟试衣间和邮局一样的利用固定位置的摄像头来实现增强显示功能的使用案例。然而，在大部分的移动案例中，用户的移动设备会改变方向，或者来回移动。为了响应用户的动作，我们需要处理从加速计和陀螺仪中返回的数据。在我们开始确认它

们在设备上的存在性之前，让我们花一分钟的时间来了解一下这两个组件的不同点。

加速计通过测量 3 个轴来度量一个相对于地球表面静止的平台的方向。如果设备只朝一个特定的方向加速，那么这个加速就无法与地球万有引力导致的加速相区分。所以，这个测量的唯一问题是：它没有为监测某个特定的方向提供足够的信息。

从 iPhone 4 开始引入了陀螺仪，由于它有测量围绕一个轴的旋转率的能力，所以它弥补了加速计的不足。使用相同的例子，陀螺仪能够测量围绕一个轴旋转的连续状态，并能够在旋转停止或者变化时作出报告。典型的陀螺仪工作在 6 个轴下。iPhone 4 装载了一个三轴的陀螺仪。

长话短说，陀螺仪负责测量和跟踪方向，加速计负责监测震动。在之后的章节我们将会看看在何时何地使用这些组件。但是，我们还是首先看一下如何在应用中检测它们的存在性。

检查陀螺仪的存在性需要另外一个框架。在这种情况下，我们需要添加 Core Motion Framework（核心动作框架）到应用中。像本章我们添加其他框架一样连接这个框架到我们的二进制应用中。下一步，打开你想检查陀螺仪存在性的类的头文件。添加代码清单 2-10 中的代码到头文件中最后一个 import 语句的下面。

代码清单 2-10 导入 Core Motion Framework

```
#import <CoreMotion/CoreMotion.h>
```

下一步，在头文件中的 @end 的上部添加代码清单 2-11 中的代码，以声明新的辅助函数。

代码清单 2-11 声明辅助函数

```
- (BOOL) isGyroscopeAvailable;
```

下一步，我们将会建立辅助函数，然后就能像我们所进行的其他硬件检查那样：在 if/else 语句中调用这个函数。复制代码清单 2-12 中的代码到这个类的 .m 文件中。

代码清单 2-12 用以检查陀螺仪是否可用的辅助方法

```
- (BOOL) isGyroscopeAvailable
{
#ifdef __IPHONE_4_0
    CMMotionManager *motionManager = [[CMMotionManager alloc] init];
    BOOL gyroscopeAvailable = motionManager.gyroAvailable;
    [motionManager release];
    return gyroscopeAvailable;
#else
    return NO;
#endif
}
```

我们使用 CMMotionManager 类的 gyroAvailable 属性来检查陀螺仪的存在性。注意，我们把这个检查包含在了对 __IPHONE_4_0 宏的检查语句中。因为 iPhone 4 之前的版本中没有陀螺仪，所以我们也不需要检查。

最终，我们需要在 `viewDidLoad` 方法靠近其他组件的确认代码的地方调用 `isGyroscopeAvailable` 方法。在摄像机的检测代码下面追加代码清单 2-13 中的代码。

代码清单 2-13 调用陀螺仪辅助方法

```
if ([self isGyroscopeAvailable]) {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Gyroscope"
                                                         message:@"Gyroscope Available"
                                                         delegate:self
                                                         cancelButtonTitle:@"OK"
                                                         otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
} else {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Gyroscope"
                                                         message:@"Gyroscope NOT
Available"
                                                         delegate:self
                                                         cancelButtonTitle:@"OK"
                                                         otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
}
```

现在你可能已经猜到，在模拟器上陀螺仪也是不可用的。所以，如果你想对它进行测试，需要把应用部署到相应设备上。

2.3 强制硬件需求

在应用中，在你试图使用某个硬件之前先进行检测永远是重要的，但是你也能够在不符合硬件需求的设备上阻止应用的运行。通过设置应用的 `Info.plist` 文件可以为你做到这一点。`Info.plist` 文件是所有 iOS 模板的标准组成部分。Xcode 4.2 的模板在工程的 Supporting Files（支持文件）目录下生成了这个文件。它会被命名为 `ProjectName-Info.plist`，其中 `ProjectName` 是 Xcode 工程的名字。

为了向应用中添加硬件需求，首先需要添加另一个键到 `Info.plist` 文件。如果这个文件已经在 Xcode 中打开，你可以选择右键菜单中的 `Add Row`（增加行）。在 Xcode 4.2 中，新创建的键的类型是 `description`，你需要设置为 `Required device capabilities`（所需的设备能力）类型。这个键是一个与各种 iOS 设备的硬件组件相对应的值的数组。例如，你能够添加 `telephony` 键来设置设备需要具备电话功能，或者添加一个 `front-facing-camera` 键来设置设备需要具备前置摄像头的设备。

使用强制硬件需求可能会遇到诸多不便，当向 App Store 提交应用时也会有一些影响。你需要确保你所需的组件可用，但又不想限制你的设备选择。在 GitHub 的相同工程中（参见本章开始的说明），我添加了 `wifi` 和 `still-camera` 的硬件需求，所以你能够有一个运行良好的例子。

表 2-2 列出了对 `UIRequiredDeviceCapabilities` 键可用的其他各种键。

表 2-2 支持 UIRequiredDeviceCapabilities 的键的列表

键	描 述
telephony	检查电话功能的存在性。你也能够使用这个功能来打开利用 tel 策略的 URL
wifi	检查设备的 Wi-Fi 联网功能
sms	检查消息应用的存在性。你也能够使用这个特性来打开利用 tel 策略的 URL
still-camera	检查设备摄像头的存在性
auto-focus-camera	检查设备摄像头的自动对焦能力，通常用于支持微距摄影的应用，或者需要更清晰的图像处理的应用
front-facing-camera	检查前置摄像头的存在性
camera-flash	用于检查拍照或者录像的摄像头闪光灯的存在性
video-camera	检查设备上的摄像头是否具备录像能力
accelerometer	检查设备加速计的存在性。使用 Core Motion Framework 来接收加速事件的应用可以用这个键。如果你的应用只监测方向的变化，你不需要这个键
gyroscope	检查陀螺仪的存在性，应用能够使用 Core Motion Framework 从陀螺仪硬件中获取信息
location-services	使用 Core Location Framework 来检查获取设备当前位置的能力
gps	为跟踪位置而测试 GPS 硬件的存在性。如果你使用这个键，你也要使用 location-services 键。比起 location-services 键（通过 Wi-Fi 或者蜂窝无线），使用 gps 可以提供更高精度的定位
magnetometer	检查电子指南针硬件。使用这个可以通过 Core Location Framework 来接收相关的定向事件
gamekit	在 iOS 4.1 或更高版本的应用中强制/禁止 Game Center 的使用
microphone	检查内置麦克风或者一个提供麦克风的配件
opengles-1/opengles-2	强制/禁止 OpenGL ES 1.1 和 OpenGL ES 2.0 接口的使用
armv6/armv7	检查应用是否是用 armv6/armv7 编译的
peer-peer	检查是否支持蓝牙对等连接（iOS 3.1 或更高版本）

2.4 总结

在本章中，我们浏览了一个增强现实应用通常会用到的硬件组件。我们也讲解了检测一个设备中的硬件的必要步骤，甚至在不符合条件的时候限制应用的运行。在接下来的几章，我们会深入讲解这些组件中的每一个是怎样使用以及怎样从硬件获取信息的。这些都将为我们编写自己的全功能的增强现实应用做好了准备。

在第 3 章中，我们将会以 iOS 中提供的 view 和 layout 开始，并且将这些功能用于我们的示例程序中。

第③章

使用位置服务

虽然本书本质上是关于增强现实的，不过关于地图和位置服务的程序编写为一个成功的增强现实应用提供了很多所需的重要元素。在本章中，我们将着眼于 iOS 的地图功能，以及将这些服务集成到你的增强现实应用中的高级技术。

如果你实在是太激动而跳过了第 2 章，我奉劝你返回并快速浏览一下 2.2.2 节。这一节讲解了怎样确定位置服务是可用的，以及怎样防止应用试图访问用户禁止的服务。如果你准备好了，那让我们开始深入研究吧。

3.1 基础功能

我们以一个例子开始。首先，打开 Xcode，并创建一个新工程。选择 Tab Bar Application 作为模板，并将你的工程命名为 iOS_AR_Ch3_LocationServices。确保将 Device Family（设备族）设置为 iPhone。所有本章讲解的内容都可以在 iPad 应用中重用。所以，为简单起见，在本章中我们将只使用 iPhone。

这一步是可选的。我为这个 Xcode 工程创建了一个本地 Git 仓库。在我的终端中，我导航到这个新工程所在的目录，并运行了代码清单 3-1 中显示的命令。本章中已完成工程可以在 github.com/kyleroches 上获得。

代码清单 3-1 连接本地仓库到 GitHub

```
Kyle-Roches-MacBook-Pro-2:iOS_AR_Ch3_LocationServices kyleroches$ git remote add origin
git@github.com:kyleroches/iOS_AR_Ch3_LocationServices.git
Kyle-Roches-MacBook-Pro-2:iOS_AR_Ch3_LocationServices kyleroches$ git push -u origin master
Counting objects: 19, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (17/17), done.
Writing objects: 100% (19/19), 10.78 KiB, done.
Total 19 (delta 5), reused 0 (delta 0)
To git@github.com:kyleroches/iOS_AR_Ch3_LocationServices.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

我们选择了 Tab Bar Application（选项卡栏应用），因此可以继续遵循本章中的避免大范围

重复编写的原则。我们以在 Xcode 中打开 FirstViewController.h 文件来开始。我们将会声明一些组建这个演示所需要的插座变量 (outlet)。在 Xcode 中打开 FirstViewController.h 文件，在 @interface 代码块中添加代码清单 3-2 中的代码。

代码清单 3-2 在 interface 中声明 UITextView

```
UITextView *locationTextView;
```

现在，在头文件的 @end 之前，添加代码清单 3-3 中的代码。

代码清单 3-3 为类添加属性

```
@property (nonatomic, retain) IBOutlet UITextView *locationTextView;
```

我们将使用 UITextView 来打印将要检索的位置服务中获取的信息。在使用这个插座变量之前，我们必须在 .m 文件中使用 synthesize 语句合成它，并使用 release 语句释放 UILabel。在 Xcode 中切换到 FirstViewController.m 文件并添加代码清单 3-4 中的粗体行到其中。

代码清单 3-4 合成并释放控件 UILabel

```
#import "FirstViewController.h"

@implementation FirstViewController
@synthesize locationTextView; // synthesize *locationTextView

/*
// Implement viewDidLoad to do additional setup after loading the view, typically from a
nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
}
*/

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientatio
n
{
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc. that aren't in use.
}

- (void)viewDidUnload
{
```

```

[super viewDidLoad];

// Release any retained subviews of the main view.
// e.g. self.myOutlet = nil;
}

- (void)dealloc
{
    [locationTextView release]; // release the UITextView
    locationTextView = nil; // good practice to set to nil after release
    [super dealloc];
}
@end

```

你可能注意到我释放了这个变量，然后还把它设置为 nil。在大部分情况下，这是一个便于内存管理的好习惯。本书的重点不在于此，但是当它们出现的时候，我们会适当地指出关于增强现实和内存管理的细节。增强现实的应用会在类中使用很多的委托方法，同时内存管理是正确地构建增强现实应用的重要方面。

现在，我们已经定义了一个能够用来更新文字的 UITextView，现在可以在 XIB 文件中创建组件。在 Xcode 的 Project Navigator 中单击 FirstView.xib 打开设计视图。你应该能够看到类似于图 3-1 显示的内容。

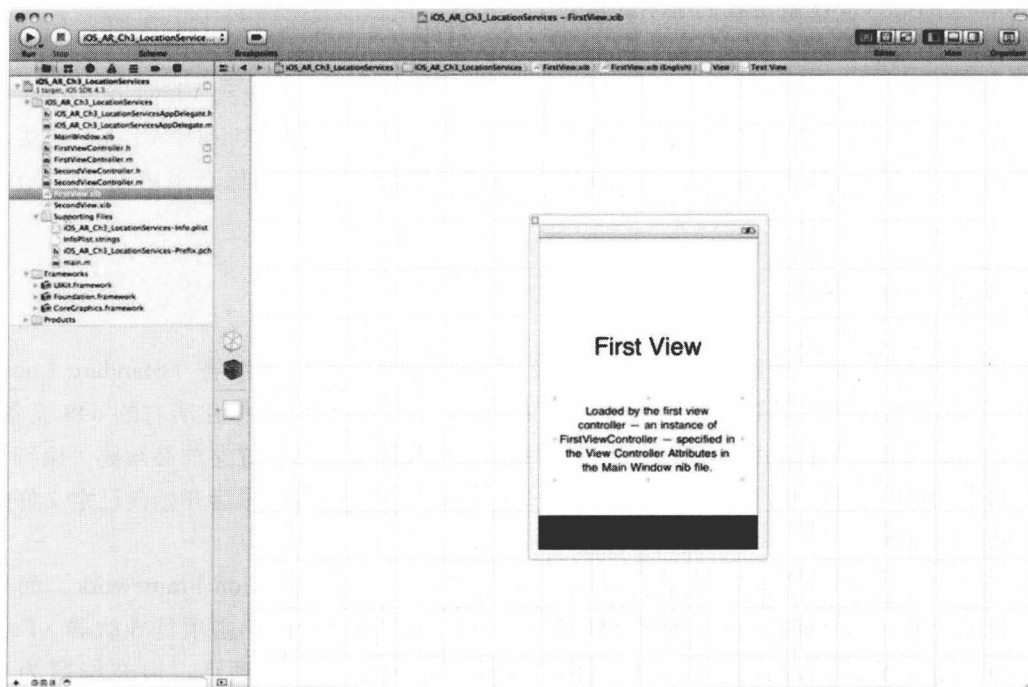


图 3-1 在设计视图中打开 FirstView.xib

你会注意到在 layout 上已经有一些组件。这就是为什么我们选择添加一个 UITextView 的原因。我们将简单地使用已经存在的组件中的一个。在设计窗口的左侧，有一个透明正方体

图标。当按下 Ctrl 键时，单击并拖曳那个图标到 iPhone 屏幕的 UITextView 的插座变量上，如图 3-2 所示。

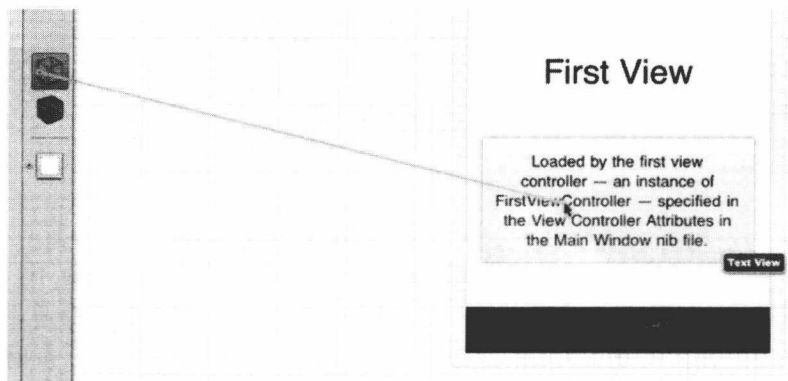


图 3-2 按下 Ctrl 键的同时拖曳 File's Owner 图标到 UITextView

当你在 UITextView 上释放鼠标的时候，会出现一个快捷菜单。你将看到两个插座变量可供选择。它们是：

- locationTextView(就是我们刚刚创建的)
- View

从列表中选择 locationTextView。我们现在已经把 UI 组件连线到 ViewController 类的 IBOutlet。返回到 Xcode 中的 FirstViewController.m 文件并反注释 viewDidLoad 方法。我们将要添加一些代码到这个方法中来开启位置服务并用最新的位置信息来更新 UITextView 元素。

3.1.1 标准位置服务

有两个用来监视位置改变的服务。让我们首先看看标准位置服务 (Standard Location Service)。这是比较常见的获取用户当前位置的方式，因为它可以工作在所有的 iOS 设备上。可以设置标准位置服务以指定位置数据所需的精度水平和在报告新位置之前必须要“旅行”的距离。当开启这个服务的时候，它首先确定要开启哪一个无线网络，然后开始向已定义好的委托服务发送位置报告。

在我们开启标准位置服务之前，需要在工程中添加 Core Location Framework。如果你没有跳过第 2 章，你可能会发现这段讲解有些多余。在 Xcode 中，单击项目导航器 (Project Navigator) 中工程的名字，然后进入 Build Phases (构建阶段) 选项卡。展开标题为 Link Binary With Libraries (用库链接至二进制) 的区域，然后添加 Core Location Framework (核心位置框架) 到工程中，如图 3-3 所示。

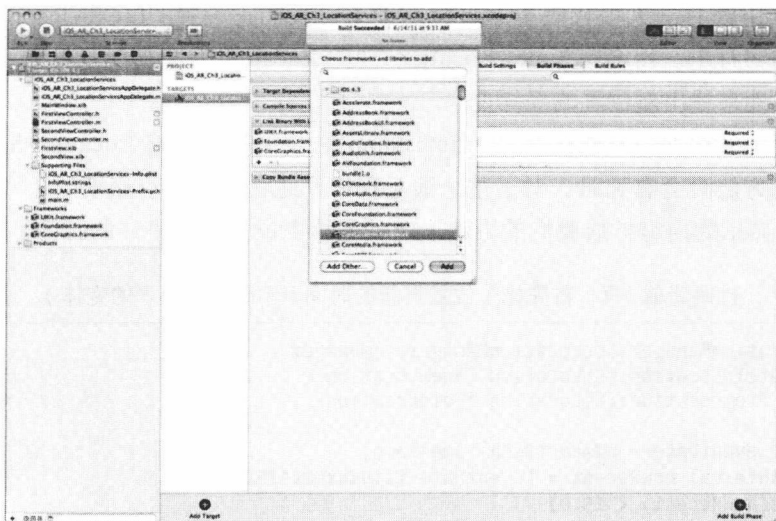


图 3-3 添加 Core Location Framework 到工程中

返回 FirstViewController.h 文件。导入 Core Location Framework 到头文件，同时声明这个类遵守 CLLocationManagerDelegate 协议。现在你的头文件应该看起来像代码清单 3-5。

代码清单 3-5 FirstViewController 的新头文件

```
#import <UIKit/UIKit.h>
#import <CoreLocation/CoreLocation.h>

@interface FirstViewController : UIViewController <CLLocationManagerDelegate> {
    UITextView *locationTextView;
}

@property (nonatomic, retain) IBOutlet UITextView *locationTextView;
@end
```

CLLocationManagerDelegate 协议发送信息到它的委托的 locationManager:didUpdateToLocation:fromLocation: 方法。如果当调用这个方法的时候出现了错误，委托源会转而调用 locationManager:didFailWithError: 方法。切换回 FirstViewController.m 文件。将代码清单 3-6 中的方法添加到你的类中。

代码清单 3-6 CLLocationManagerDelegate 类的委托方法

```
- (void)locationManager:(CLLocationManager *)manager didUpdateToLocation:(CLLocation
*)newLocation
    fromLocation:(CLLocation *)oldLocation
{
    locationTextView.text = [NSString stringWithFormat:@"latitude %+.6f, longitude
%+.6f\n",
        newLocation.coordinate.latitude,
        newLocation.coordinate.longitude];
}
```

在这个方法中没有一大堆代码要运行。基本上，我们正设置 UITextView 的文本为包含新的经纬度坐标的字符串。用名字为 newLocation 的 CLLocation 对象传递这些坐标给我们。我们使用它的坐标属性来找出新位置的经纬度。

苹果建议检查这个新位置的时间戳（timestamp），因为 Standard Location Service 有时候在递交信息到委托方法时会有延时。对于这个演示程序，此检查是没有必要的；不过，你可以按代码清单 3-7 中显示的内容扩展我们的方法，以首先检查位置更新事件的时间，仅供参考。

代码清单 3-7 首先检查位置更新的时间（代码清单 3-5 的变体）

```

- (void)locationManager:(CLLocationManager *)manager
  didUpdateToLocation:(CLLocation *)newLocation
    fromLocation:(CLLocation *)oldLocation
{
    NSDate* eventDate = newLocation.timestamp;
    NSTimeInterval howRecent = [eventDate timeIntervalSinceNow];
    if (abs(howRecent) < 15.0)
    {
        locationManager.text = [NSString stringWithFormat:@"latitude %+.6f, longitude
%+.6f\n",
                                newLocation.coordinate.latitude,
                                newLocation.coordinate.longitude];
    } else {
        locationManager.text = @"Update was old";
        // you'd probably just do nothing here and ignore the event
    }
}

```

我们的监听器已经准备好了接收位置更新。我们继续并开启服务。切换到头文件（FirstViewController.h），然后将代码清单 3-8 中的代码添加到 @end 语句的前面。

代码清单 3-8 声明开启位置服务的方法

```

- (void)startStandardUpdates;

```

现在切换回 .m 文件，然后添加代码清单 3-9 中的代码。

代码清单 3-9 startStandardUpdates 方法

```

- (void)startStandardUpdates
{
    CLLocationManager *locationManager = [[CLLocationManager alloc] init];
    locationManager.delegate = self;

    locationManager.desiredAccuracy = kCLLocationAccuracyKilometer;
    locationManager.distanceFilter = 500;

    [locationManager startUpdatingLocation];
}

```

这个方法有几个重要的方面。首先，在声明了 CLLocationManager 对象后，我们又设置 delegate 为 self。如果没有在前面的代码清单 3-5 中声明这个类遵守 CLLocationManagerDelegate

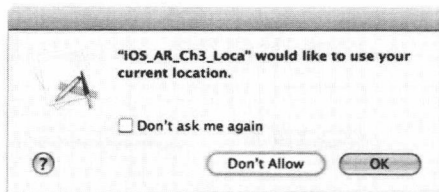
协议，这里就会抛出一个警告。接下来，我们设置了一些配置选项，以使标准位置服务知道我们想要在什么时候接收更新。我们设置需要的精度为千米并告诉位置管理器在监测到位置改变超过 500 千米时更新其 delegate。

最后，在你的 `viewDidLoad`：方法中，在 `[super viewDidLoad];` 语句的前面添加代码清单 3-10 中标记为粗体的代码。

代码清单 3-10 新的 `viewDidLoad` 方法

```
- (void)viewDidLoad
{
    [self startStandardUpdates];
    [super viewDidLoad];
}
```

我们现在已经准备好来测试演示程序。在 Xcode 的左上部有一个叫做 Scheme（策略）的下拉菜单。确保 iPhone 4.3 simulator（模拟器）处于选中状态，然后点击最左边的 Run 按钮。这时模拟器会启动并将自动打开我们的演示应用。



你应该会马上被一个如图 3-4 所示的模态对话框请求检查你的位置的许可权。

如果你想要这个对话框不再出现，选中 `Don't ask me again`（不要再次询问我）复选框。无论你使用哪种方式，一定要确保你点击的是 `OK` 按钮，否则演示应用就无法正常工作。图 3-5 展示的是运行中的应用。你能够看到 `UITextView` 现在填入了我们创建的包含我们经纬度的字符串。

3.1.2 显著变更位置服务

到目前为止，标准位置服务是获取设备位置的最常用的方式。然而，从 iOS 4 之后，如果你愿意降低精度节省功率可以使用显著变更位置服务（Significant-Change Location Service）。对于大部分应用来说，显著变更位置服务是足够精确的。它使用设备的蜂窝无线代替 GPS 无线来确定位置，这允许设备更积极地去管理电源使用。为了递交新坐标，显著变更位置服务也有唤醒暂停的应用的能力。



一旦你确定了哪一个位置服务最适合应用的需求，剩下的编码差异是非常基础的知识。把代码清单 3-11 中的方法加入到 `FirstViewController.m` 来开启显著变更位置服务。

代码清单 3-11 开启显著变更位置服务

```
- (void)startSignificantChangeUpdates
{
    CLLocationManager *locationManager = [[CLLocationManager alloc] init];
    locationManager.delegate = self;
    [locationManager startMonitoringSignificantLocationChanges];
}
```

因为我们指定这个类遵守 `CLLocationManagerDelegate` 协议并且我们设置了 `CLLocationManager` 的 `delegate` 属性为 `self`，运行这个方法而非我们添加的代码清单 3-9 中的方法，尽管相同模拟器的屏幕显示结果是相同的（不管怎么样，我们是这么认为的）。添加代码清单 3-12 中的代码到你的头文件并按代码清单 3-13 显示的那样修改 `viewDidLoad` 方法。

代码清单 3-12 在 `FirstViewController.h` 中声明新方法

```
- (void)startSignificantChangeUpdates;
```

代码清单 3-13 调用 Significant-Change Location Service 的新 `viewDidLoad` 方法

```
- (void)viewDidLoad
{
    // [self startStandardUpdates];
    [self startSignificantChangeUpdates];
    [super viewDidLoad];
}
```

如果你打开模拟器并运行了我们的这个新版本，你将会发现没有一点不同。换句话说，什么都没有发生。`UITextView` 从来没有改变。如果你部署这个演示到一个物理设备上，那么结果会随着你位置的改变而改变。现在不要担心这个。在 3.1.3 节，我将会介绍一个在模拟器中测试你的位置的方法。

3.1.3 地理区域监控服务

在一些情况下，监测一个精确的位置并不能完全解决问题。有时候，我们仅仅想知道的是我们是否或者何时接近某一个坐标。iOS 提供了一系列的关于区域监控服务的功能。地理区域监控服务（Geographic Region Monitoring Service）与其他位置服务的工作方式在很多方面相同。它提供了一些需要处理的关键委托方法来与区域内的变化做适当的交互。我们将使用 `didEnterRegion:` 和 `didExitRegion:` 这两个委托方法来演示功能。

这个例子中的代码只能在物理设备上正常工作。然而，为了在模拟器上做演示，我要在 Xcode 4.2 beta 的 iOS 5 模拟器上显示它。Xcode 4.2 引入了一个在调试器中模拟位置改变的方法，这可以很好地说明我们的例子。下面把代码清单 3-14 中的方法添加到 `FirstViewController.m` 文件中。

代码清单 3-14 添加新方法到 FirstViewController.m 文件

```

- (void)startRegionMonitoring
{
    NSLog(@"Starting region monitoring");
    CLLocationManager *locationManager = [[CLLocationManager alloc] init];

    locationManager.delegate = self;

    CLLocationCoordinate2D coord = CLLocationCoordinate2DMake(37.787359, -122.408227);
    CLRegion *region = [[CLRegion alloc] initWithCircularRegionWithCenter:coord
    radius:1000.0 identifier:@"San Francisco"];

    [locationManager startMonitoringForRegion:region
    desiredAccuracy:kCLLocationAccuracyKilometer];
}

```

这段代码与其他的跟踪位置的方法非常相似，除了最后开启区域监视服务的一行。我们用一个 San Francisco 的 GPS 坐标创建了一个 CLLocationCoordinate2D 对象。我选择 San Francisco 是因为它是 Xcode 4.2 beta 中创建的预加载 GPS 坐标中的一个，如图 3-6 所示。

不要忘记在头文件中声明你的新方法。把代码清单 3-15 中的代码添加到 FirstViewController.h 中 @end 的正上方。

代码清单 3-15 在 FirstViewController.h 中声明新方法

```

- (void)startRegionMonitoring;

```

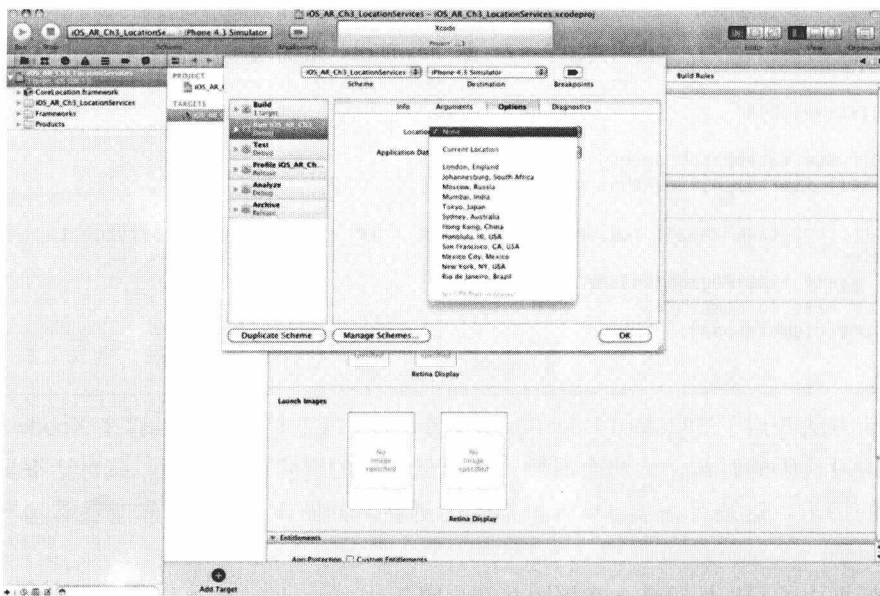


图 3-6 在 Xcode 4.2 beta 中选择预加载 GPS 坐标

现在区域监控服务已经开启了，我们接下来需要处理当用户进入或者离开我们监控的区域

时所需调用的委托方法。添加代码清单 3-16 中的方法到 FirstViewController.m。

代码清单 3-16 在 FirstViewController.m 中声明新方法

```
- (void)locationManager:(CLLocationManager *)manager didEnterRegion:(CLRegion *)region {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Region Alert"
        message:@"You entered the region"
        delegate:self
        cancelButtonTitle:@"OK"
        otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
}

- (void)locationManager:(CLLocationManager *)manager didExitRegion:(CLRegion *)region {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Region Alert"
        message:@"You exited the region"
        delegate:self
        cancelButtonTitle:@"OK"
        otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
}
```

测试区域监控服务的最后一步是去调用我们刚刚创建的 startRegionMonitoring 方法。在第 2 章中，我们讨论了在使用它们之前怎样检查服务的存在性和硬件的支持与否。在当前情况下，我们应该做同样的事情。CLLocationManager 类提供了一个同时检查区域检测服务是否开启以及该服务可用性的方法。按照代码清单 3-17 中显示的代码更新你的 viewDidLoad 方法。

代码清单 3-17 新 viewDidLoad 方法

```
- (void)viewDidLoad
{
    [self startStandardUpdates];
    //[self startSignificantChangeUpdates];

    if ([CLLocationManager regionMonitoringAvailable]) { // check is service is
        available
        [self startRegionMonitoring];
    } // else do something
    [super viewDidLoad];
}
```

我们的新方法开启了 Standard Location Service 和区域监控服务。当你在 Xcode 4.2 的新调试器中运行这段代码的时候，你将会看到一个稍微不同的许可权对话框，该对话框显示使用你的设备位置的请求。这是另一项 iOS 5 模拟器的更新。如图 3-7 所示，你能够看到一个新许可权对话框的例子。

点击对话框中的 OK 按钮来加载视图并开启服务。

Xcode 4.2 的另一个新功能是在应用运行时改变模拟器的位置。在 Xcode 的控制台窗口中，你将看到一个新的关于位置的下拉菜单，如图 3-8 所示。

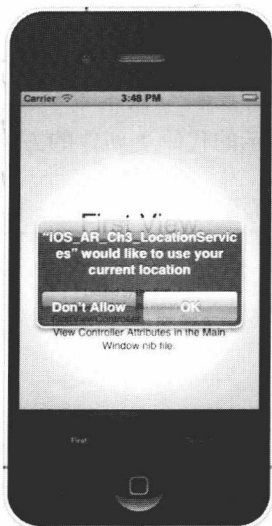


图 3-7 在新 iOS 授权对话框中允许权限

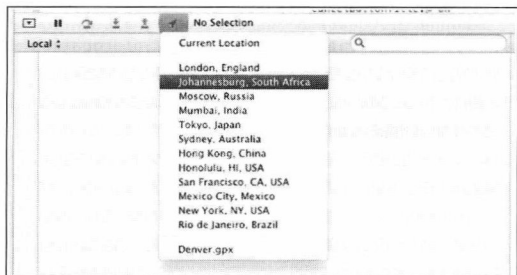


图 3-8 使用新版 Xcode 4.2 beta 的调试器的位置模拟器

当初始化区域监控服务时，我们以一个在 San Francisco 附近的坐标开始。所以，为了测试代码，我们需要进入和离开那个区域。如果你已经安装了 Xcode 4.2，通过调试器中的下拉菜单设置 San Francisco,CA,USA 作为你的位置。模拟器会在后台改变它的坐标；然后标准位置服务会获取这些改变，并把它们报告给委托方法。此外，因为我们进入了定义好的区域，所以 `didEnterRegion:` 方法也会被触发。你将会看到与图 3-9 中显示的内容相似的情景。

我们也可以使用调试器来测试 `didExitRegion:` 方法。把你的位置改成除了 San Francisco,CA,USA 以外的位置，然后你将会触发其他的委托方法，如图 3-10 所示。

图 3-9 `didEnterRegion:` 方法被触发图 3-10 `didExitRegion:` 方法被触发

到现在为止，我们已经学习了监控设备位置的各种位置服务，并建立了一些设备进入或者离开某一个地理区域时的提醒选项。在增强现实的应用中，目标点的可视化坐标同样需要我们知道设备到某一个目标坐标的距离。为了做到这一点，让我们对在本章早期定义的 `didUpdateToLocation:` 方法做一个快速修改。按代码清单 3-18 中显示的代码更新你的方法。

代码清单 3-18 新版 `didUpdateToLocation` 方法

```
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
      fromLocation:(CLLocation *)oldLocation
{
    NSDate* eventDate = newLocation.timestamp;
    NSTimeInterval howRecent = [eventDate timeIntervalSinceNow];
    if (abs(howRecent) < 15.0)
    {
        //locationTextView.text = [NSString stringWithFormat:@"latitude %+.6f, longitude %+.6f\n",
        //
        //                                newLocation.coordinate.latitude,
        //                                newLocation.coordinate.longitude];

        CLLocationDistance dist = [newLocation distanceFromLocation:oldLocation] / 1000;
        locationTextView.text = [NSString stringWithFormat:@"distance %5.1f traveled"];
    } else {
        locationTextView.text = @"Update was old";
        // you'd probably just do nothing here and ignore the event
    }
}
```

在 Xcode 4 的模拟器中，你看到的结果将会与图 3-11 相似。如果你正在使用 Xcode 4.2 beta 和 iOS 5 模拟器，你可以测试一些其他有趣的距离。测试这个功能的其他方式就是传递一个新的 `CLLocation` 坐标来代替 `oldLocation`。

3.1.4 高度

在增强现实过程中，检查高度有助于确保呈现给用户一个合适的摄像头角度的视图。获取设备的高度与获取位置的开始过程相同。苹果建议先检查一下高度读取是否可用，然后再试着从 GPS 或者无线蜂窝获取高度。现在，使用代码清单 3-19 中的代码来确定设备是否提供高度读取能力。



图 3-11 检查两次位置更新之间的距离

代码清单 3-19 检查高度读取是否可用

```
If (signbit(newLocation.verticalAccuracy)) {
    // get the altitude
}
```

高度读取有几个陷阱。第一，旧 iPhone（原版和 iPhone 3GS）非常的不精确。一些参考资

料中指出，与 iPhone 4 相比，这些设备的精度能差十倍。第二，与我们在本章早期的位置服务的例子一样，核心位置只会在设备移动了某个距离（非垂直）的时候才会更新。所以，我们必须要么重置 GPS，要么让用户自动在它们已经改变了高度的时候告诉我们。为了检查 GPS 读取，添加代码清单 3-20 中的代码。

代码清单 3-20 检查 GPS 读取能力

```
locationTextView.text = [NSString stringWithFormat:@"%6.2f m. ", newLocation.altitude];
```

通常，即使是在最新的设备上，高度读取的精度也是不够的。有一些其他的方法和公共可用网络服务能够用来确定某一个 GPS 坐标的高度。在后面章节的例子中，我们将重新审视这个主题。在我们继续在一个地图上可视化位置信息之前，我们先回顾一下在本章用过的类。表格 3-1 列出了最常用的方法和属性。

表 3-1 回顾 Core Location 的方法和属性

方法名 / 属性名		描 述
CLLocationManager		
delegate	属性	定义了一个与 CLLocationManagerDelegate 相对应的对象
desiredAccuracy	属性	设置所需精度为一个 CLLocationAccuracy 对象
distanceFilter	属性	定义了需要多长的横向移动才可触发一个位置更新事件
location	属性	最新的位置
CLLocationManagerDelegate		
locationManager:didUpdateToLocation:	方法	当一个更新事件发生时进行报告的委托方法
fromLocation:locationManager:didFailWithError:	方法	当一个更新事件失败时进行报告的委托方法
CLLocation		
altitude	属性	以米为单位指定位置的高度
coordinate	属性	以 CLLocationCoordinate2D 变量的形式返回一个位置
timestamp	属性	指示位置测量时的时间，类型为 NSDate

3.2 在地图上查看

我们已经探讨了获取用户位置的不同方法。我们看一下怎么把这些呈现到一个地图上。在增强现实的应用中，通常并不需要使用地图。但是，它为测试我们增强现实应用中使用的位置和坐标提供了良好的调试和可视化能力。我们将会在本章中讲解地图的应用，如此你就可以将其加入你的工具箱中了。

我们首先在演示工程中建立 SecondViewController 类。这是 tab view template 的第二个标签。到目前为止，如果你一直跟随本章的讲解，那么这个类应该没有用过。打开 SecondView-

Controller.h 文件并按照代码清单 3-21 显示的方法声明 MKMapView 的 IBOutlet。

代码清单 3-21 声明 MKMapView 的 IBOutlet

```
#import <UIKit/UIKit.h>
#import <MapKit/MapKit.h>

@interface SecondViewController : UIViewController <MKMapViewDelegate>{
    IBOutlet MKMapView *mapView;
}

@property (nonatomic, retain) IBOutlet MKMapView *mapView;
@end
```

代码清单 3-21 中粗体显示的是我们要添加到默认模板类的代码。首先，我们导入 Mapkit.h 头文件。同时，我们指定类遵守 MKMapViewDelegate 协议。这将允许这个类处理 MKMapView 的消息和更新。

下一步，打开 SecondViewController.m 并使用 synthesize 语句合成你的属性。同时，当你完成的时候不要忘记释放它。添加代码清单 3-22 中的代码到 SecondViewController.m 中。

代码清单 3-22 合成并释放对象

```
@synthesize mapView;

- (void)dealloc
{
    [mapView release];
    mapView = nil;
    [super dealloc];
}
```

在 Xcode 中打开 SecondView.xib 文件，移除里面默认的 UITextView 和 UILabel。从 Object Library（对象库）中拖动一个 Map View 到 view（视图）上并暂停它的自动缩放填充可用空间的属性。下一步，按住 Ctrl 键，然后在编辑器的左侧的正方体（文件所有者的图标）上单击并拖出一个线到新添加的 MKMapView 上。你应该能够看到一个与图 3-12 中相似的快捷菜单，选中菜单中的 mapView 属性。

3.2.1 置中地图和设置显示区域

返回到 SecondView.m 文件，我们首先反注释 viewDidLoad 方法，然后向这个文件添加代码清单 3-23 中的代码。

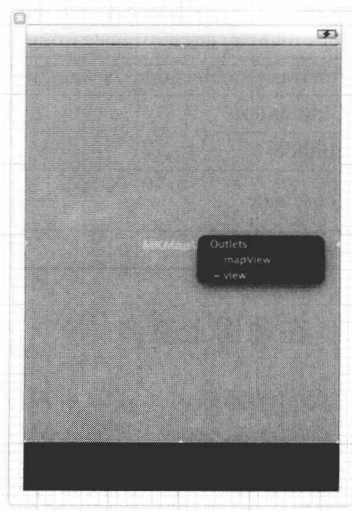


图 3-12 设置 MKMapView

代码清单 3-23 为建立地图开启一个新线程

```
- (void)viewDidLoad
{
    mapView.delegate=self;
    [self.view addSubview:mapView];

    [NSThread detachNewThreadSelector:@selector(displayMap) toTarget:self
    withObject:nil];
    [super viewDidLoad];
}
```

首先，我们设置 MKMapView 的 delegate 属性为 self。回顾一下，我们已经设置这个类遵守 MKMapViewDelegate 协议。最后，我们分出了一个新线程来加载地图，而不是在 viewDidLoad: 方法中加载。对于这个例子来说，这有点没必要，但这对于介绍线程有益，后面我们增强现实的演示应用中将会用到线程。我们的线程调用了方法，displayMap，这是一个我们还没有创建的方法。在 viewDidLoad: 方法的下面添加代码清单 3-24 中的代码来创建这个方法。

代码清单 3-24 创建新的 displayMap 方法

```
- (void)displayMap {
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

    CLLocationCoordinate2D coords;
    coords.latitude = 37.33188;
    coords.longitude = -122.029497;
    MKCoordinateSpan span = MKCoordinateSpanMake(0.002389, 0.005681);
    MKCoordinateRegion region = MKCoordinateRegionMake(coords, span);
    [mapView setRegion:region animated:YES];

    [pool drain];
}
```

在这个方法中有几个事情需要讨论。首先，我们运行应用看看会发生什么。图 3-13 展示的是在 iPhone 模拟器中运行这个应用的情况。现在忽略前面以粗体显示的代码，因为我们之后会回来讲解。我们先手动创建一个 CLLocationCoordinate2D 变量，并设置经纬度到 Apple HQ 中。我们创建了一个 MKCoordinateSpan 变量来显示屏中经纬度的变化量。最后，我们使用新的坐标和跨度来建立 MKCoordinateRegion 对象。

在这个例子中，我们使用 MKMapView 的 setRegion 方法来设置地图的默认值。有一些其他的能够添加到这里的附加的自定义选项。你可以设置 mapType 属性，我们接下来会介绍它，同时你也可以使用 zoomEnabled 和 scrollEnabled 方法来指定用户是否可以缩放和滚动地图。

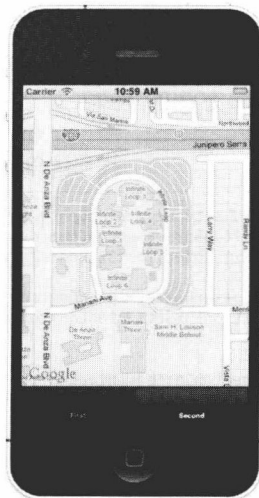


图 3-13 模拟器在地图上显示 Apple HQ
(泄漏或者不泄漏看起来并没有什么不同)

做个测试，注释掉粗体的代码然后重新运行这个应用。你的 Xcode 的控制台窗口很快就会充满了关于没有线程池的警告信息，并且提示你的应用正在发生内存泄漏。因为我们让方法运行在一个新的线程中，所以需要在新线程中建立一个新的 `NSAutoReleasePool` 来处理内存管理。

`NSAutoreleasePool` 这个类用于支持 reference-counted 内存系统。iOS 模板（我们正在用的是 tab view template（选项卡视图模板））在 Supporting Files 目录下的 main.m 文件中创建了一个自动释放池。这些自动释放池包含接收到的自动释放消息的对象。当你像我们在新 viewDidLoad 的最后一行一样排干（drain）一个池的时候，`NSAutoReleasePool` 会向这些对象中的每一个发送释放消息。因为现在的执行代码没有处在主线程中了，所以如果不创建这个内存池，那么将没有一个自动释放对象会收到它们的释放消息。因此，我们会有内存泄漏的潜在风险，这会导致控制台窗口出现警告信息。

3.2.2 修改地图类型

在 3.2.1 节我提到 `mapType` 属性可以用于改变我们正在显示的地图的类型。返回 `SecondViewController.h` 文件，并按代码清单 3-25 中显示的方法声明一个 `IBOutlet`。

代码清单 3-25 声明 `UISegmentedControl`

```
@interface SecondViewController : UIViewController <MKMapViewDelegate>{
    IBOutlet MKMapView *mapView;
    UISegmentedControl *buttonBarSegmentedControl;
}

@property (nonatomic, retain) IBOutlet MKMapView *mapView;
@property (nonatomic, retain) IBOutlet UISegmentedControl *buttonBarSegmentedControl;
```

我们通过代码添加这个控件，而不是在 XIB 文件中手动添加。在增强现实的应用中，我们会以相似的方式来添加控件到用户视图上。添加代码清单 3-26 中的方法到我们之前创建的 `displayMap` 方法的下面。

代码清单 3-26 创建 `UISegmentedControl` 来修改地图类型

```
- (void)setupSegmentedControl {
    buttonBarSegmentedControl = [[UISegmentedControl alloc] initWithItems:[NSArray
arrayWithObjects:@"Standard", @"Satellite", @"Hybrid", nil]];

    [buttonBarSegmentedControl setFrame:CGRectMake(30, 50, 280-30, 30)];
    buttonBarSegmentedControl.selectedSegmentIndex = 0.0; // start by showing the
normal picker

    [buttonBarSegmentedControl addTarget:self action:@selector(toggleToolBarChange:)
forControlEvents:UIControlEventValueChanged];

    buttonBarSegmentedControl.segmentedControlStyle = UIScrollViewIndicatorStyleWhite;
    buttonBarSegmentedControl.backgroundColor = [UIColor clearColor];
    buttonBarSegmentedControl.tintColor = [UIColor blackColor];
    [buttonBarSegmentedControl setAlpha:0.8];

    [self.view addSubview:buttonBarSegmentedControl];
}
```

这个方法也需要在头文件中声明。在运行这个应用之前，应确保你做了这一步。

这个方法首先用一个新数组中的元素初始化 `UISegmentedControl`。分段控件与 iOS 中的按钮工作机制相似。我们需要能够添加一个目标并恰当地对产生的事件进行响应。代码清单 3-27 中的粗体代码将 `target` 属性设置为 `self`，并设置 `action` 属性为一个叫做 `toggleToolBarChange` 的新方法。接下来我们将会创建这个方法。最后我们在 `setupSegmentedControl` 方法中为控件设置了一些基础的参数。我们设置了样式、背景颜色和浅色。显然这些都是可选的，但是我想演示一些选项，设置这些选项能够使控件的样式与本章中我们使用的 `tab bar`（选项卡栏）模板的默认样式相一致。如果你没有设置这些选项，控件将会默认显示为绿色。

我前面提到将 `action` 属性设置成一个名为 `toggleToolBarChange` 的新方法。这个方法的代码如代码清单 3-27 所示。添加该方法到类中。

代码清单 3-27 处理 `UISegmentedControl` 的 `action`

```
- (void)toggleToolBarChange:(id)sender
{
    UISegmentedControl *segControl = sender;
    switch (segControl.selectedSegmentIndex)
    {
        case 0: // Map
        {
            [mapView setMapType:MKMapTypeStandard];
            break;
        }
        case 1: // Satellite
        {
            [mapView setMapType:MKMapTypeSatellite];
            break;
        }
        case 2: // Hybrid
        {
            [mapView setMapType:MKMapTypeHybrid];
            break;
        }
    }
}
```

在我们测试这个新控件之前，最后一个步骤是在 `viewDidLoad` 方法中调用 `setupSegmentedControl` 方法。就在你分出一个新线程来建立地图之后，添加代码清单 3-28 中的代码。

代码清单 3-28 调用 `setupSegmentedControl`

```
[self setupSegmentedControl];
```

现在已经准备好了测试我们的新功能。在 iPhone 模拟器中运行组合起来的代码。切换到第二个选项卡，然后测试分段控件的按钮。图 3-14 显示的是在模拟器上运行选中的卫星类型的地图。

那么，我们添加了一个地图到视图，并学习了如何基于静态位置坐标来置中它，还动态添加了一个 UISegmentedControl 控件到视图中（以使我们可以在运行中改变地图的类型）。

接下来，我们看看如何在地图上添加注释。注释提供了一个标记指定位置的方式。在谷歌地图或者 Bing 中，它们的工作方式大体相同，你可以通过添加弹出窗口来向用户显示更多的相关信息。它们响应的动作以及图标能够被自定义。

3.2.3 在地图上添加注释

为了建立一个注释，我们将使用一个保存注释的定制化属性的新类。在 Project Navigator（项目导航器）中某处右击，在弹出的菜单中选择 New File 命令。在 Cocoa Touch 分类之下，选择 Objective-C 类模板，让这个类作为 NSObject 类的一个子类，并把这个新类命名为 MapAnnotation。地图上的注释有几个主要的规格参数：标题、副标题、坐标。我们将会为这些规格参数向新类添加属性，并提供一些方法以使我们可以在 map 类中对这些属性进行设置。首先添加代码清单 3-29 中的声明到 MapAnnotation.h 文件。

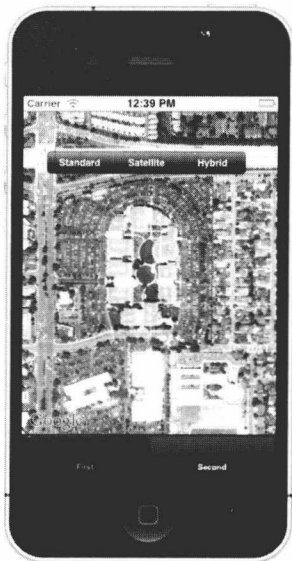


图 3-14 选中的卫星地图类型

代码清单 3-29 MapAnnotation.h 文件的变更

```
#import <Foundation/Foundation.h>
#import <MapKit/MapKit.h>

@interface MapAnnotation : NSObject<MKAnnotation> {
    CLLocationCoordinate2D coordinate;
    NSString *subtitledtext;
    NSString *titletext;
}
@property (nonatomic, readonly) CLLocationCoordinate2D coordinate;
@property (readwrite, retain) NSString *titletext;
@property (readwrite, retain) NSString *subtitledtext;
-(id)initWithCoordinate:(CLLocationCoordinate2D) coordinate;
- (NSString *)subtitle;
- (NSString *)title;
-(void)setTitle:(NSString*)strTitle;
-(void)setSubTitle:(NSString*)strSubTitle;

@end
```

再次，我们以导入 MapKit.h 为开始。我们为初始坐标位置（放置注释的位置）和注释的标题和副标题声明属性。我们也添加了一个新方法到类中。我们有一个坐标初始化方法，也有一系列的关于标题和副标题的 getter 和 setter 方法。

切换到 MapAnnotation.m，然后添加代码清单 3-30 中对应的属性和方法的实现。

代码清单 3-30 修改 MapAnnotation.m

```

@implementation MapAnnotation
@synthesize coordinate, titletext, subtitletext;

- (NSString *)subtitle{
    return subtitletext;
}

- (NSString *)title{
    return titletext;
}

-(void)setTitle:(NSString*)strTitle {
    self.titletext = strTitle;
}

-(void)setSubTitle:(NSString*)strSubTitle {
    self.subtitletext = strSubTitle;
}

-(id)initWithCoordinate:(CLLocationCoordinate2D) c{
    coordinate=c;
    return self;
}
@end

```

这个类没有什么新奇的。它是充当注释属性的占位符的基本重构。在 SecondViewController.h 中为 MapAnnotation.h 头文件添加一个 import 语句。接着，打开 SecondViewController.m，然后添加代码清单 3-31 中的代码到 displayMap 方法中，并调用 setRegion 方法的前面的语句。请关注 MapAnnotation 类的 3 个方法以及它们是如何被调用的。

代码清单 3-31 在 displayMap 方法中创建注释

```

MapAnnotation *addAnnotation = [[[MapAnnotation alloc] initWithCoordinate:coords]
retain];
[addAnnotation setTitle:@"My Annotation Title"];
[addAnnotation setSubTitle:@"this is my subtitle property"];
[mapView addAnnotation:addAnnotation];

```

首先，当创建 MapAnnotation 对象的时候，我们调用了 initWithCoordinate 方法来设置注释的默认位置。接下来两行调用 setter 方法来设置标题和副标题属性。最后，我们添加注释到 MKMapView 中。

继续在模拟器中运行。切换到第二个选项卡，然后你会在 Apple HQ 中看到地图视图和一个红色注释（之前置中地图中的相同坐标处）。如果你单击注释，你会看到与图 3-15 相似的结果。

你将会看到标题和副标题都已经被填入，并且注释正显示在正确的坐标上。我们有足够的可用选项来自定义注释。例如，如何动画注释和修改注释的颜色呢？为了自定义注释的外观，我们可以使用另一个叫做 viewForAnnotation 的委托方法。当创建

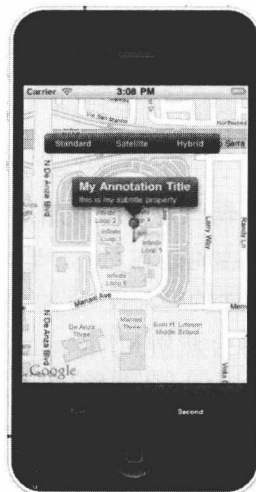


图 3-15 默认注释显示填入的标题和副标题

一个注释时，这个方法被调用，同时为你提供使用可重用对象的引用来定制特定视图的机会。添加代码清单 3-32 中的方法到类中以处理委托方法。

代码清单 3-32 处理 viewForAnnotation 委托方法

```
- (MKAnnotationView *) mapView:(MKMapView *)mapView viewForAnnotation:(id
<MKAnnotation>) annotation{
    MKPinAnnotationView *annView=[[MKPinAnnotationView alloc]
initWithAnnotation:annotation reuseIdentifier:@"MyPin"];
    annView.animatesDrop=TRUE;
    annView.canShowCallout = YES;
    [annView setSelected:YES];
    annView.pinColor = MKPinAnnotationColorPurple;
    annView.calloutOffset = CGPointMake(-50, 5);
    return annView;
}
```

我们对注释做了一些修改。如果你现在在模拟器中运行它，你将会看到类似图 3-16 所示的情景。

当模拟器切换到第二个选项卡的时候，你注意到的第一个改变是注释的显示方式是动画的。它以一个令人愉快的、友好的动画方式进入位置。我们使用 `animatesDrop` 属性实现这一点。现在它还是紫的，我们可以通过 `pinColor` 属性设置注释的颜色。最后，你会注意到对话框是偏离中心的。我们使用 `calloutOffset` 属性来设置一个偏移。当你使用的自定义注释图片无法像默认的图片一样协调位置的时候，这会很有用。

在 3.3 节，我们将介绍如何获取一个指定坐标的更多信息。

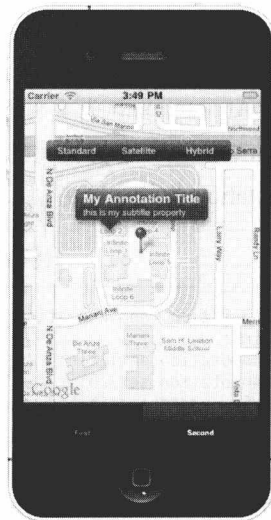


图 3-16 模拟器现在显示的是定制的注释

3.3 解析地理编码

最后，我们来解析地理编码。解析地理编码是利用经纬度值来获取地址或位置信息的过程，而地理编码是利用一个地址来取得经度和纬度值的过程。iOS 为开发者提供了一个类来轻松地解析地理编码。我们将扩展当前的例子并记录一些关于正在使用的坐标的真实信息。

我们将会使用 `MKReverseGeocoder` 类。这个类的属性和方法的使用说明如表 3-2 所示。但是，在使用它们中的任何一个方法或属性之前，需要设定 `SecondViewController` 类遵守 `MKReverseGeocoderDelegate` 协议。这是很重要的，否则解析地理编码就无法完成。

表 3-2 MKReverseGeocoder 类的方法和属性

方法 / 属性	类 型	介 绍
delegate	属性	为 geocoder 的消息指定委托对象（错误信息和位置信息）
coordinate	属性	用于获取位置信息的坐标
querying	属性	一个布尔值，用于指示 geocoder 当前是否正在处理位置数据
start:/cancel:	方法	开启和退出请求；如果请求自然完成，那么它会触发两个委托方法之一

为了简单起见，同时为了不用重构整个示例工程，我们解析截止到现在一直在使用的坐标的地理编码。找到 toggleToolBarChange 方法并按图 3-33 所示的内容重构这个方法。使用这个方法的原因是，我们想要通过用户的一个动作来开启解析地理编码的过程，以此来演示功能。当你单击分段控件上的按钮时，这个方法会被调用，所以它会为我们的目的服务。

代码清单 3-33 更新后的 toggleToolBarChange 方法调用解析地理编码

```

...
case 1: // Satellite
{
    CLLocationCoordinate2D coords;
    coords.latitude = 37.33188;
    coords.longitude = -122.029497;

    MKReverseGeocoder *geoCoder = [[MKReverseGeocoder alloc]
initWithCoordinate:coords];
    [geoCoder setDelegate:self];
    [geoCoder start];
    [mapView setMapType:MKMapTypeSatellite];
    break;
}
...

```

首先，我们重复利用了 CLLocationCoordinate2D 的代码。在任何一个真实的应用中，要实现这点要么被派生，要么设置在一个更高的作用域内。在又一次创建了我们的坐标之后，我们用那个相同的坐标初始化了 MKReverseGeocoder。这是使用解析地理编码的类所需要的。然后我们设置 delegate 为 self。最后，我们开启了地理编码过程。

正如你所期望的，因为我们设置类遵守 MKReverseGeocoderDelegate 协议，所以还有几个委托实现需要的方法。参考代码清单 3-34，并添加委托方法到 SecondViewController.m 文件。

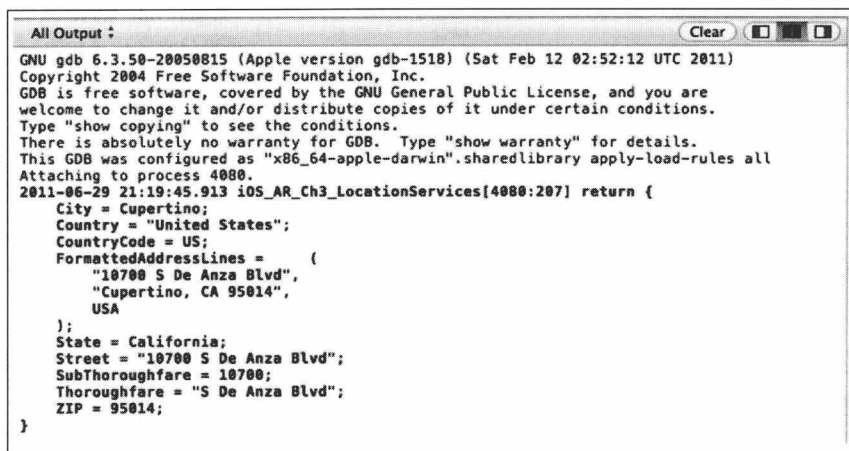
代码清单 3-34 MKReverseGeocoderDelegate 的方法

```

- (void)reverseGeocoder:(MKReverseGeocoder *)geocoder didFindPlacemark:(MKPlacemark
*)placemark {
    NSLog(@"return %@", placemark.addressDictionary);
}
- (void)reverseGeocoder:(MKReverseGeocoder *)geocoder didFailWithError:(NSError *)error
{
    NSLog(@"fail %@", error);
}

```

这里代码的关键部分是粗体的。如果在指定的坐标上发现任何相关信息，MKPlacemark 对象会返回到委托方法。图 3-17 显示的是 Xcode 的控制台窗口的输出。



```
All Output :
GNU gdb 6.3.50-20050815 (Apple version gdb-1518) (Sat Feb 12 02:52:12 UTC 2011)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "x86_64-apple-darwin".sharedlibrary apply-load-rules all
Attaching to process 4080.
2011-06-29 21:19:45.913 iOS_AR_Ch3_LocationServices[4080:207] return {
    City = Cupertino;
    Country = "United States";
    CountryCode = US;
    FormattedAddressLines = (
        "10700 S De Anza Blvd",
        "Cupertino, CA 95014",
        USA
    );
    State = California;
    Street = "10700 S De Anza Blvd";
    SubThoroughfare = 10700;
    Thoroughfare = "S De Anza Blvd";
    ZIP = 95014;
}
```

图 3-17 Xcode 的控制台显示解析地理编码的输出

3.4 总结

在本章中，我们全面讲解了位置服务。我们首先查看跟踪一个位置的各种可用选项，例如 Standard Location Service、Significant-Change Location Service、地理区域监控服务，以及确定高度的方法。我们了解了如何在一个地图上可视化位置服务以及如何从一个位置的地理编码来解析相关信息。

本章也有一些较难的课程。例如，基本线程，我们将会在以后的例子中继续解释这个问题；还有动态添加控件，这个在增强现实章节中将会在 HUD（头戴显示器）样式中较多地使用。

为了全面地创建增强现实应用，我们建立了这些工具。在第 4 章，我们将会介绍利用加速计和陀螺仪来工作的 iOS 传感器编程。

第④章

iOS 传感器

增强现实的编程需要使用多种硬件组件才能建立一个令人印象深刻的应用。这些组件中的大部分通常涉及传感器。传感器帮助我们监测应用运行时设备的方向和位置的变化，例如加速计和陀螺仪。因为我们将编写增强现实的应用，所以会将应用覆盖在设备的摄像头的真实视图中。追踪位置可以让我们能够按照现实的情景来保持应用的朝向，这就是为什么传感器如此重要的原因。

4.1 方向传感器

加速计和陀螺仪的使用目的是相似的，但是两者之间有一个细微的不同，并且大部分应用要依靠这两者的结合来实现最大化效果。我们看看这两个组件的相同点和不同点。

我们以加速计的介绍开始。iPhone 4 中的 3 轴加速计用于测量相对于地球表面的方向。如果设备碰巧做自由落体运行或者在一个方向上有一个恒定的加速度，加速计会返回 0（或者说应该返回 0），因为无法判断加速度是由地球的万有引力施加引起的还是其他力引起的。

陀螺仪有测量围绕指定轴的旋转率的能力。加速计并不测量旋转，这就是飞行计算机不能单单只依靠加速计的原因，它还需要一个陀螺仪来防止飞机滚动。

思考这些组件之间的差异的一个简单方法是陀螺仪用于测量方向，加速计用于测量摇动。使用这个作为基本原则，陀螺仪能够提供一个角速率（或旋转速率），而加速计只能能够测量线性加速度（跨所有 3 个轴的）。

我们首先逐个查看这些组件及其使用方法，然后结合这两个组件来获得更好的测量结果。本章将再次使用 Tab Bar application 模板以使示例应用的数量保持到最小。在 Xcode 中使用 Tab Bar application 模板创建一个新工程，设置目标设备为 iPhone。新工程命名为 iOS_AR_Ch4_Sensors，或者从这里下载本章的完整代码：github.com/kyleroche/Professional_iOS_AugmentedReality，或者 Apress 网站的 Source Code/Download 区域（www.apress.com）。

4.1.1 使用加速计

我们以加速计的介绍开始。在 Xcode 中，打开 FirstViewController.h 文件并按代码清单 4-1

中的代码更新它。

代码清单 4-1 更新 FirstViewController.h 文件

```
#import <UIKit/UIKit.h>

@interface FirstViewController : UIViewController {
    UIAccelerometer *accelerometer;

    UILabel *xLabel;
    UILabel *yLabel;
    UILabel *zLabel;

    UIProgressView *xProgressView;
    UIProgressView *yProgressView;
    UIProgressView *zProgressView;
}

@property (nonatomic, retain) UIAccelerometer *accelerometer;

@property (nonatomic, retain) IBOutlet UILabel *xLabel;
@property (nonatomic, retain) IBOutlet UILabel *yLabel;
@property (nonatomic, retain) IBOutlet UILabel *zLabel;

@property (nonatomic, retain) IBOutlet UIProgressView *xProgressView;
@property (nonatomic, retain) IBOutlet UIProgressView *yProgressView;
@property (nonatomic, retain) IBOutlet UIProgressView *zProgressView;

@end
```

接着，我们更新 FirstViewController.m 文件，然后逐句地解释这些新代码。在 Xcode 中，切换到 FirstViewController.m 文件并添加代码清单 4-2 中的代码。

代码清单 4-2 更新 FirstViewController.m

```
// after implementation
@synthesize xLabel, yLabel, zLabel, xProgressView, yProgressView, zProgressView,
accelerometer;
// replace existing dealloc from auto-gen code
- (void)dealloc
{
    [xLabel release];
    xLabel = nil;
    [yLabel release];
    yLabel = nil;
    [zLabel release];
    zLabel = nil;
    [xProgressView release];
    xProgressView = nil;
    [yProgressView release];
    yProgressView = nil;
    [zProgressView release];
    zProgressView = nil;
    [accelerometer release];
    accelerometer = nil;
    [super dealloc];
}
```

从顶部开始，我们声明了几个用于显示加速计的实时读数的 UI 元素。我们将会使用 3 个标签控件和 3 个进度条（progress bar）控件，每一个对应于加速计的一个轴。图 4-1 显示的是 X、Y、Z 坐标轴。

在 Xcode 中打开 FirstView.xib。添加标签和进度条元素到界面画布中。你应该能够看到与图 4-2 显示的相似的情景。

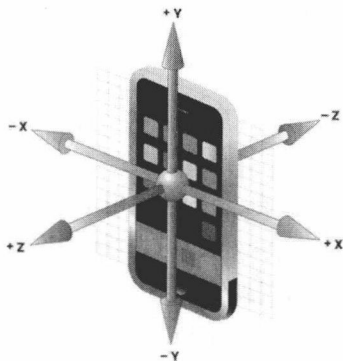


图 4-1 加速计使用 X、Y 和 Z 坐标轴

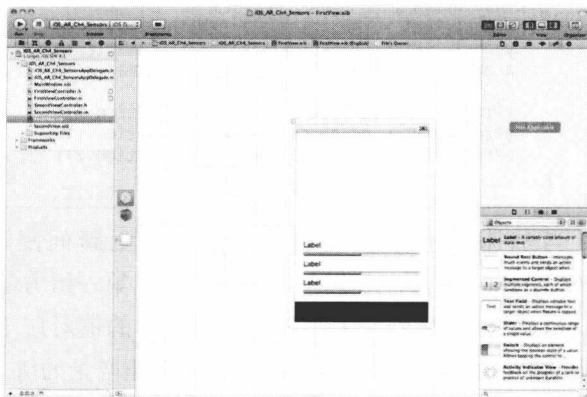


图 4-2 为加速计读数建立 XIB 文件

为简单起见，让我们以横向顺序把这 3 组值分配给插座变量。X 在顶部，然后 Z 在底部。因为一些人对 Xcode 4.2 比较陌生，我们将会再回顾一下这个知识点。为了将插座变量分配给头文件中的属性，按住 Ctrl 键的同时，拖动 File's Owner 图标到画布上的插座变量。这时会弹出一个对话框，显示用于分配的适当属性，并选择合适的插座变量。图 4-3 显示了一个例子。

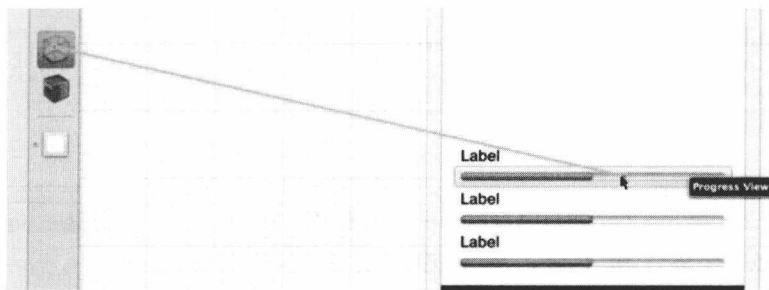


图 4-3 连接 IBOutlet 到 FirstViewController.h 的属性

为每一个 UILabel 插座变量和 UIProgressView 组件重复以上过程。

通过 UIAccelerometer 类来访问加速计。这个类的使用方式与第 3 章中针对位置服务使用的委托方式相同。这个类使用一个共享的 UIAccelerometer 类的单例来访问，而不是直接访问。因为这一点，你将会在这些例子如何处理内存方面注意到些许改变（具体来说，是关于如何处理我们订阅的硬件信息方面）。

1. UIAccelerometerDelegate 类

在开始监控加速计以用于更新之前，必须指定这个类遵守 UIAccelerometerDelegate 协议，并对 FirstViewController.h 文件进行相应的更新。接下来切换回 FirstViewController.m 文件并添加代码清单 4-3 中的方法。

代码清单 4-3 UIAccelerometerDelegate 类的委托方法

```

- (void)accelerometer:(UIAccelerometer *)accelerometer didAccelerate:(UIAcceleration
*)acceleration {
    xlabel.text = [NSString stringWithFormat:@"%f", @"X: ", acceleration.x];
    ylabel.text = [NSString stringWithFormat:@"%f", @"Y: ", acceleration.y];
    xlabel.text = [NSString stringWithFormat:@"%f", @"Z: ", acceleration.z];

    xProgressView.progress = ABS(acceleration.x);
    yProgressView.progress = ABS(acceleration.y);
    zProgressView.progress = ABS(acceleration.z);
}

```

当加速计每次监测到新的、在可更新间隔范围内的值的时候，它会调用委托类的 `didAccelerate` 方法。我们浏览一下代码。第一组方法格式化一个 `NSString` 值并相应地赋给 3 个 `UILabel` 的 `text` 属性。置于每个字符串中的值对应于每个特定轴的加速度值。第二个代码块基于相同的值设置了 `UIProgressView` 的插座变量的值。注意，还把测量值转换成了绝对值。这是因为进度条不能显示负值。如果有一个负的测量值，你会在 `UILabel` 中看到它。

在能够运行工程之前，必须设置加速计的更新间隔并为其分配一个委托类。如前所述，`UIAccelerometer` 类是一个单例类，需要创建一个 `sharedAccelerometer` 对象的引用以开始接收测量值。取消注释 `viewDidLoad` 方法并进行适当调整，如代码清单 4-4 所示。

代码清单 4-4 新版 `viewDidLoad` 方法

```

- (void)viewDidLoad
{
    accelerometer = [UIAccelerometer sharedAccelerometer];
    accelerometer.updateInterval = .5;
    accelerometer.delegate = self;

    [super viewDidLoad];
}

```

这个方法的第一行将 `UIAccelerometer` 对象分配给 `shareAccelerometer` 对象。接下来，需要设置以秒为单位的 `updateInterval`。这里设置的值为 0.5（每秒两次）。随意调整这个值并测试响应时间。最后，设置委托（`delegate`）为 `self`，然后就可以接收测量值。

确保你有一个已连接的物理设备并且已在 Xcode 的 Scheme 下拉菜单中选中。在选定的设备上运行工程。你应该可以看到类似图 4-4 的情景。

在测试这个应用的时候，试一试一些不同的情况。试着让设备向下来观察一个轴的快速变化。试着将设备在水平方向上转向一侧，这会增加 *X* 轴的值到满值（要么正的，要么负的，取决于转向哪一边）。接下来，让设备直立，*Y* 轴现在应该有一个满值，同时其他轴的读数趋向于 0。最后试着让屏幕向上，平放设备，

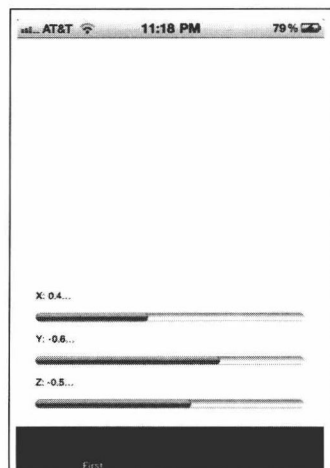


图 4-4 获得 `UIAccelerometer` 类的 *X*、*Y* 和 *Z* 轴测量值

你应该看到 Z 轴接近 1，其他轴接近 0。我们要学会获知设备的方向。在本书后面，我们将需要把这几种类型的读数整合到我们的应用中。

2. 摇动监测

也可以使用加速计来测量用户的动作。App Store 上一些流行的 iOS 应用使用摇动监测作为刷新动作，例如 Facebook。其他原生应用，如 messaging，使用摇动监测作为用户取消一个动作的方法。

返回 FirstViewController.m 的 didAccelerate 委托方法，并添加代码清单 4-5 中的代码到这个方法的底部。

代码清单 4-5 添加到 didAccelerate 委托方法

```
double const kThreshold = 2.0; // 2Gs is typical to measure shaking
if ( ABS(acceleration.x) > kThreshold
    || ABS(acceleration.y) > kThreshold
    || ABS(acceleration.z) > kThreshold) {
    // if shake is detected across any axis
    NSLog(@"Shake Detected!"); // Log it!
}
```

首先，定义了一个常量作为摇动监测的阈值。如果任何一条轴的测量震动值大于 2Gs[⊖]，这个方法就会把消息输出到控制台。在设备上运行更新后的工程，当应用打开后摇动设备进行测试。在 Xcode 控制台中，你应该会看到与图 4-5 相似的情景。

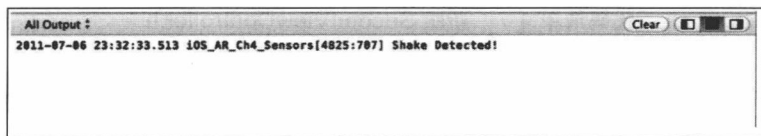


图 4-5 监测一个用户的摇动

第 5 章将会讨论声音和用户反馈，并向你展示如何促使设备震动。本章专注于收集传感器数据。

4.1.2 低通滤波

我们将会在示例中使用一个叫做低通滤波的技术。基本上，当我们接收到加速计数据的时候，我们移除了一切（除了重力读数）。这会帮助我们保持放入增强现实的视图中的 3D 对象的朝向。代码清单 4-6 显示了建立一个低通滤波器的示例代码。

代码清单 4-6 建立一个低通滤波器

```
accel[0] = acceleration.x * kFilteringFactor + accel[0] * (1.0 - kFilteringFactor);
accel[1] = acceleration.y * kFilteringFactor + accel[1] * (1.0 - kFilteringFactor);
accel[2] = acceleration.z * kFilteringFactor + accel[2] * (1.0 - kFilteringFactor);
```

⊖ Gs 是加速度单位，1Gs 就是一倍的地球表面重力加速度。——译者注

4.1.3 使用陀螺仪

陀螺仪实际上更容易读取。没有必要为陀螺仪建立一个委托类或者方法来处理更新。在启动陀螺仪之后，可以通过使用 `CMMotionManager` 类来简单地请求测量值。这个类充当了设备硬件的一个入口。

我们使用演示工程中的第二个选项卡来测量陀螺仪。打开 Xcode，选中 `SecondView.xib` 文件，按图 4-6 显示的画面设立界面画布。我们将基于它的 3 个姿态属性来进行陀螺仪的可视化，它们是 `roll`、`pitch` 和 `yaw`。

在开始连接这些插座变量之前，必须添加一个新的框架到工程中。与陀螺仪的交互需要用到 `Core Motion Framework`。通过选择 Xcode 的项目导航器（Project Navigator）中的工程名字，并在应用目标的 Build Phases 选项卡的 Link Binary With Libraries 部分中添加这个框架。打开 `SecondViewController.h` 文件，并添加将用来向刚刚创建的插座变量发送信息的属性。使用代码清单 4-7 中的代码作为参考。

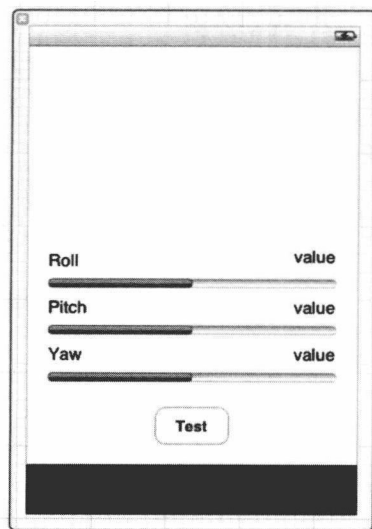


图 4-6 为监控陀螺仪建立界面

代码清单 4-7 新版 `SecondViewController.h`

```
#import <UIKit/UIKit.h>
#import <CoreMotion/CoreMotion.h>

@interface SecondViewController : UIViewController {
    CMMotionManager *motionManager;

    UILabel *rollLabel;
    UILabel *pitchLabel;
    UILabel *yawLabel;

    UIProgressView *rollProgressView;
    UIProgressView *pitchProgressView;
    UIProgressView *yawProgressView;
}

@property (nonatomic, retain) CMMotionManager *motionManager;

@property (nonatomic, retain) IBOutlet UILabel *rollLabel;
@property (nonatomic, retain) IBOutlet UILabel *pitchLabel;
@property (nonatomic, retain) IBOutlet UILabel *yawLabel;

@property (nonatomic, retain) IBOutlet UIProgressView *rollProgressView;
@property (nonatomic, retain) IBOutlet UIProgressView *pitchProgressView;
@property (nonatomic, retain) IBOutlet UIProgressView *yawProgressView;

- (IBAction)readGyroscope;

@end
```

在我们继续讲解之前，我们先快速浏览一下这段代码。在顶部，你会看到用于添加到应用

中的 Core Motion Framework 的 import 语句。接下来，在 interface 块，你会注意到，我们没有指定这个类为任何类型的委托，没有按加速计的方式处理。陀螺仪类充当了硬件传感器的一个界面入口。它并不会像加速计一样向一个委托方法发送消息。

UILabel 和 UIProgressBar 属性的出现并不罕见。它们与加速计例子中的用法相同。但是，在这个例子中还有一个新的属性。CMMotionManager 是一个用于与陀螺仪交互的类。

最后，为按钮声明了一个 IBAction 方法。我们将会手动调用这个方法来查询陀螺仪的当前测量值，而不是像在加速计例子中所做的一样——依靠一个委托方法。

在 Xcode 中，切换到 SecondViewController.m 文件，并使用 synthesize 语句合成新属性；不要忘记更新 dealloc 方法来使用 release 语句释放属性并设置它们为 nil，如代码清单 4-8 所示。

代码清单 4-8 合成并释放新属性

```
// after @implementation
@synthesize motionManager;
@synthesize rollLabel, rollProgressView, pitchLabel, pitchProgressView, yawLabel,
yawProgressView;

// new dealloc
- (void)dealloc
{
    [motionManager release];
    motionManager = nil;
    [rollLabel release];
    rollLabel = nil;
    [rollProgressView release];
    rollProgressView = nil;
    [pitchLabel release];
    pitchLabel = nil;
    [pitchProgressView release];

    pitchProgressView = nil;
    [yawLabel release];
    yawLabel = nil;
    [yawProgressView release];
    yawProgressView = nil;
    [super dealloc];
}
```

取消注释 viewDidLoad 方法并按代码清单 4-9 中显示的内容更新它。

代码清单 4-9 取消注释并更新 viewDidLoad 方法

```
- (void)viewDidLoad
{
    self.motionManager = [[[CMMotionManager alloc] init] autorelease];
    motionManager.deviceMotionUpdateInterval = 1.0/60.0;
    if (motionManager.isDeviceMotionAvailable) {
        [motionManager startDeviceMotionUpdates];
    }
    [super viewDidLoad];
}
```

首先，为 CMMotionManager 类的新实例设置 motionManager 属性。接下来，设置一个更新间隔。最后，如果动作管理器（motion manager）是可用的，则开启更新。如果你还记得第 2

章中验证传感器可用性的过程，这会看起来非常熟悉。如果你跳过了第 2 章，只要知道在访问一个传感器之前确认其可用性即可。

1. 姿态

在使用值来更新 UI 之前，仅将它们输出到控制台并确认我们能够成功地访问硬件。通过复制代码清单 4-10 中的代码到 SecondViewController.m 中，添加在头文件中声明的 IBAction 方法。

代码清单 4-10 添加 IBAction 方法 readGyroscope

```
- (void)readGyroscope {
    CMDeviceMotion *currentDeviceMotion = motionManager.deviceMotion;
    CMAAttitude *currentAttitude = currentDeviceMotion.attitude;

    NSLog(@"Attitude: %@", currentAttitude);
}
```

这个方法创建一个 CMDeviceMotion 对象，并从陀螺仪中读取 CMAAttitude 值。切换到 SecondView.xib 文件并设置 UIButton 的 IBAction 为 readGyroscope 方法。CMAAttitude 读数有一个关键参数集合（参见表 4-1），我们将在本书后面的例子应用中使用这些参数。

表 4-1 CMAAttitude 属性

属性名字	介 绍
pitch	以弧度计算的设备的倾斜。倾斜表示围绕横轴从设备的一侧旋转到另一侧
quaternion	返回一个描绘设备姿态的四元数
roll	以弧度计算的设备的横滚。横滚是围绕纵轴从设备的顶部旋转到底部
yaw	以弧度计算的设备的偏航。偏航表示围绕垂直穿过设备的轴所进行的旋转。这个轴以重心为原点，以设备底部为正向，垂直于设备体

在物理设备上运行当前应用。导航到第二个选项卡，并单击 Test 按钮。以不同的方向旋转设备并继续测试值。在 Xcode 控制台中，你应该会看到类似于代码清单 4-11 的输出记录。

代码清单 4-11 来自设备姿态记录的 NSLog 输出

```
2011-07-08 20:38:31.955 iOS_AR_Ch4_Sensors[6144:707] Attitude: Pitch: -41.931169, Roll:
-57.704060, Yaw: -177.801492 @ 0.000000
2011-07-08 20:38:33.572 iOS_AR_Ch4_Sensors[6144:707] Attitude: Pitch: -17.880685, Roll:
-62.436956, Yaw: 167.555800 @ 0.000000
2011-07-08 20:38:49.238 iOS_AR_Ch4_Sensors[6144:707] Attitude: Pitch: 28.940916, Roll:
11.359928, Yaw: -45.087161 @ 0.000000
```

姿态（attitude）是设备的 Pitch、Roll 和 Yaw 的结合。如果我们单独访问这些属性，你会分别得到以弧度计算的各自的值。通常，为了基本的增强现实目的，例如为了知道设备的视角和旋转度，必须将它们转换为读数。添加代码清单 4-12 中的代码到 readGyroscope 方法的末尾。

代码清单 4-12 添加到 readGyroscope 方法的末尾

```
rollLabel.text = [NSString stringWithFormat:@"ROLL: %f", currentAttitude.roll];
rollProgressView.progress = ABS(currentAttitude.roll);
pitchLabel.text = [NSString stringWithFormat:@"PITCH: %f", currentAttitude.pitch];
pitchProgressView.progress = ABS(currentAttitude.pitch);
yawLabel.text = [NSString stringWithFormat:@"YAW: %f", currentAttitude.yaw];
yawProgressView.progress = ABS(currentAttitude.yaw);
```

刚刚添加的代码块应该看起来很熟悉。为了快速调试，我们刚刚在 UI 元素中设置了可见值。运行这个应用，你应该可以看到类似图 4-7 的情景。

注意，在 Xcode 控制台中显示的值与我们在屏幕上看到的并不是完全匹配的。格式是相同的，但是我们现在看到的是在每条轴上旋转的弧度测量值。

在讲解本书后面的示例之前，我们将会讨论一个叫做 cocos2D 的框架。我们将为了动画和 2D 功能而使用这个框架。cocos2D 提供了一些非常简单的将弧度转换为角度的函数。

动作管理器自动将加速计和陀螺仪的数据混合在一起。但是，这种方法丢失了一个读数：旋转率。我们组建一个简单的例子来演示如何收集这些度量并与 UI 中的一个元素相协调。这将是增强现实编程的常见任务。大部分情况下，当手机旋转的时候，你希望覆盖在视图上的元素保持在正确的方向上。

2. 旋转率

在 Xcode 中，打开 SecondView.xib。在视图已有的元素上面添加一个 UIImageView 对象。添加一个图片到方向比较水平的元素上。本章的 GitHub 仓库里面有一个关于战斗机的图片，也许你会用到。在头文件中声明这个 UIImageView，并在实现文件中合成 / 释放这个对象。添加代码清单 4-13 中的代码到 readGyroscope 方法的下面。

代码清单 4-13 添加到 readGyroscope 方法的底部

```
float rotation;
float rate = motionManager.gyroData.rotationRate.z;
if (fabs(rate) > .2) {

    float direction = rate > 0 ? 1 : -1;
    rotation += direction * M_PI/90.0;
    imageView.transform = CGAffineTransformMakeRotation(rotation);
    NSLog(@"Rotation: %f", rotation);
}
```

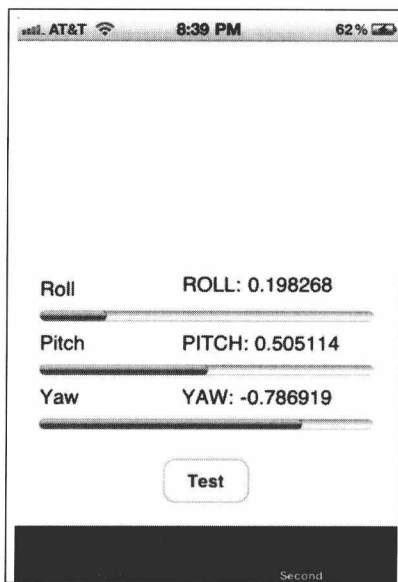


图 4-7 从 CMMotionManager 读取陀螺仪数据

这个方法将使用动作管理器的 gyroData 属性来获取 Z 轴的旋转率。现在有一个新的函数，

它可用于相对于 Z 轴的旋转率旋转图片。在本书后面深入讲解的例子中，我们将会看到多种关于这种类型的任务的方法。当然，我们实际上还没有启动对陀螺仪的监控，所以在能够看到任何的有用的结果之前仍需要启动它。按图 4-14 中显示的内容更新 viewDidLoad 方法。

代码清单 4-14 更新 viewDidLoad 方法

```

- (void)viewDidLoad
{
    self.motionManager = [[[CMMotionManager alloc] init] autorelease];
    motionManager.deviceMotionUpdateInterval = 1.0/60.0;

    if (motionManager.isDeviceMotionAvailable) {
        [motionManager startDeviceMotionUpdates];
        [motionManager startGyroUpdates];
    }
    [super viewDidLoad];
}

```

重要的需要注意的是，这段代码没有停止对陀螺仪读数的监听。恰当地启动和停止对传感器读数的监控是一个需要实现的非常重要的例程。

继续并运行这个工程。你将会发现它有点难以测试。当单击 Test 按钮的时候，我们想要验证的是：如果我们单独地显著地增加 Z 轴的旋转率的时候，图片会在 Z 轴上旋转相同的角度。这个地方的难点在于，我们必须同时做单击按钮和旋转设备这两件事，这会导致旋转率只有很小的变化，如图 4-8 所示。

可以使用 NSTimer 类来设立一个间隔并回调另一个方法，而不是通过单击按钮来手动地获取读数。

把代码清单 4-13 中的代码块重构为一个新方法，如代码清单 4-15 所示。

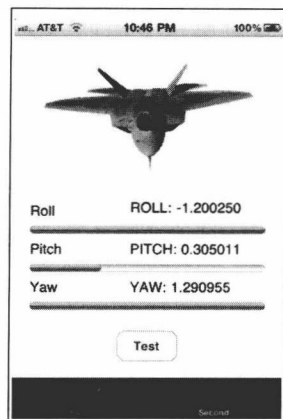


图 4-8 测试手动陀螺仪读数（注意到一个很小的图片滚动效果）

代码清单 4-15 重构代码显示新方法：doGyroUpdate

```

- (void)doGyroUpdate {
    float rotation;
    float rate = motionManager.gyroData.rotationRate.z;
    if (fabs(rate) > .2) {
        float direction = rate > 0 ? 1 : -1;
        rotation += direction * M_PI/90.0;
        imageView.transform = CGAffineTransformMakeRotation(rotation);
        NSLog(@"Rotation: %f", rotation);
    }
}

```

在开启对陀螺仪事件的监听后，创建一个 NSTimer 来调用新方法。首先，在头文件中声明 NSTimer。我命名我的 NSTimer 为 timer。在 viewDidLoad 方法中创建 timer。代码应该看起来与代码清单 4-16 相似。同样，在这个例子中，我们没有在创建 timer 以后真正地清理代码，因为我们的目的是读取数据。这个应用没有 UI 流。

代码清单 4-16 建立 NSTimer

```
timer = [NSTimer scheduledTimerWithTimeInterval:1/30.0
        target:self
        selector:@selector(doGyroUpdate)
        userInfo:nil
        repeats:YES];
```

如果你再次运行这个应用，你将会在控制台中看到很多输出，以及对于设备旋转的一个更平滑的响应。陀螺仪类正以亚秒更新，同时我们使用这些读数来旋转图片。

4.1.4 磁力计

磁力计是测量设备周围磁场强度的传感器。假设在设备周围没有强磁场，这个读数会与围绕地球的磁场相关联，这允许我们使用该读数来确定方向。设备的指向是设备相对于北极的地磁方向。像其他大部分关于这个主题的参考资料一样，苹果在其开发者文档中指出：地磁方向与遵照地理北极的真实方向可能由于设备位置的不同而有很大的不同。

我们测试设备地磁方向的方式与我们测试陀螺仪或者加速计读数的方法非常相似。我们扩展工程以同时显示方向。

在 Xcode 中，添加一个新的 UIViewController 类到工程中，命名为 headingViewController，并确保在创建时选中了 With XIB for user interface 复选框。我使用谷歌图片搜索发现了一个没有版权的指南针图片。它包含在 GitHub 的工程中，也可以在 Apress 网站（www.apress.com）的 Source Code/Download 区域获得。如果你用自己的图片代替了这个，要确保北方（North）在图像的顶部居中。我们将基于设备的方向旋转这个图片，所以想要以图片的北相对于垂直轴 0° 的角度来开始，以使事情尽可能简单。在你获得图片后，将它添加到 Xcode 工程中。

1. 磁力计可用性

如第 3 章所讨论的，CLLocationManager 需要用户开启位置更新才能正常工作。位置管理器能够同时返回设备的真实方向和磁力方向。但是设备的真实方向只有在设备开启了位置更新后才可用。然而，磁力方向的更新总是可用的，无论用户对于位置更新的设置是怎样的。这是因为指出自身的方向并不会损害用户的隐私，所以不需要获取许可。

总之，你不需要像在其他传感器中做的那样，在使用之前检查其可用性。

2. 校准

因为我们有获得两个读数的巨大可能性，所以我们在 iOS 中有一些允许校准设备以确定地磁方向与真实方向之间的差异的函数。CLLocationManager 定义了一个叫做 locationManagerShouldDisplayHeadingCalibration 的委托方法。这个方法覆盖一个指向图片来指导用户按图 4-8 所示的模式进行设备旋转，直到位置管理器能够区分地球磁场与任何本地磁场。位置管理器使用真实方向作为参考。

3. 检查设备朝向

在界面构建器（interface builder）中打开 headingViewController.xib。添加一个 UIImageView 和两个 UILabel outlet。你可以在图 4-9 中看到如何建立例子的界面。

在助理模式编辑器中打开 headingViewController.h，并在拖曳界面构建器中的 UIImageView 和 UILabel 组到 interface 文件的时候按下 Ctrl 键，如图 4-10 所示。

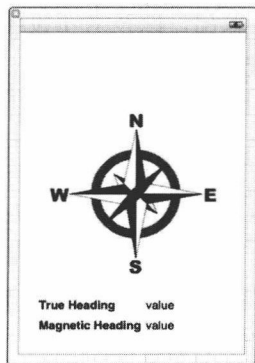


图 4-9 磁力计界面的结构（headingViewController.xib）

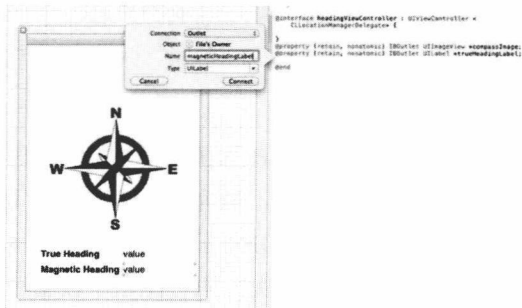


图 4-10 这里是为 IBOutlet 建立属性

确保导入了 <CoreLocation/CoreLocation.h> 头文件并指定 headingViewController 遵守 CLLocationManagerDelegate 协议。

完成以上操作后，在 Xcode 中切换到 headingViewController.m 文件。按代码清单 4-17 中的代码更新 viewDidLoad 方法。

代码清单 4-17 新版 viewDidLoad 方法

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.

    CLLocationManager *locationManager = [[CLLocationManager alloc] init];
    locationManager.delegate = self;
    locationManager.headingFilter = 5;
    if ([CLLocationManager locationServicesEnabled] && [CLLocationManager
headingAvailable]) {
        [locationManager startUpdatingHeading];
        [locationManager startUpdatingLocation];
    } else {
        NSLog(@"Error starting location updates");
    }
}
```

我们先简单浏览这段代码。首先，建立了 CLLocationManager 实例。这应该看起来与我们在第 3 章中的操作相似。设置 delegate 为 self，但是之后引入了一个新的属性。headingFilter 属性定义了一个变化量的阈值，超过这个阈值就需要向委托方法更新。设置这个值为 5（度），因此只有在方向变化量大于 5° 的时候位置管理器才会更新委托方法。

我已经提到你不需要检查磁力计的可用性。那么，为什么我在 viewDidLoad 方法中包含

了一个检查？这是因为方向信息只在 iPhone 3GS 及其以后版本中可用。这个检查可以确保用户拥有最新的硬件。同时，特别需要注意的是，我使用的是类方法 `locationServicesEnabled` 和 `headingAvailable`。很多在线示例应用使用实例方法来做这个检查。从 iOS 4 开始，iOS 放弃了使用实例方法。请确保你的应用使用的是类方法。

最后，一旦证实读数是可用的，就开启位置和方向更新。

接下来，必须在界面中添加委托方法来处理更新和响应变化。添加代码清单 4-18 中的方法到 `viewDidLoad` 方法的正下方。

代码清单 4-18 `didUpdateHeading` 委托方法

```
-(void)locationManager:(CLLocationManager *)manager didUpdateHeading:(CLHeading *)newHeading {
    if (newHeading.headingAccuracy > 0) {
        float magneticHeading = newHeading.magneticHeading;
        float trueHeading = newHeading.trueHeading;

        magneticHeadingLabel.text = [NSString stringWithFormat:@"%f", magneticHeading];
        trueHeadingLabel.text = [NSString stringWithFormat:@"%f", trueHeading];

        float heading = -1.0f * M_PI * newHeading.magneticHeading / 180.0f;
        compassImage.transform = CGAffineTransformMakeRotation(heading);
    }
}
```

你可以看到在进行其他任何事情之前，首先检查了 `headingAccuracy` 的值。如果位置更新没有开启，`headingAccuracy` 的值会设置为 -1。检查值是否大于 0，可以确保我们能够从磁力计和位置更新中获得正确的读数。我们简单地设置两个 UILabel 为读数的值，如此，你就可以看到它们之间的细微差别。我们使用 `CGAffineTransformMakeRotation` 方法来旋转图片。我们乘以负的常量，然后乘以 π ，最后除以 180。我们使用 180 是因为无论方向怎么变，朝向总是与设备的顶部相关联。以后我们会详细讨论这一点。

在一个物理设备上运行工程。你将会看到与图 4-11 所示的相似的结果。

4. 处理方向的变化

当运行这个示例应用的时候，我们测试一下设备方向在横向模式下会发生什么。你会看到北对应的是设备的顶部，而不是可视屏幕的顶部。如果你以横向模式放置你的设备而没有对你的指向进行相应的改变，设备会通过指向左侧或者右侧来报告方向（取决于你的调整）。我们可以通过向类快速添加代码修复这个问题。

在界面构建器中打开 `headingViewController.xib` 文件。添加另一个 UILabel IBOutlet 到界面，命名为 `orientationLabel`，并将它添加到 `headingViewController.h` 文件，就像我们处理其他的插座变量一样。在 `interface` 文件中声明代码清单 4-19 中的方法。

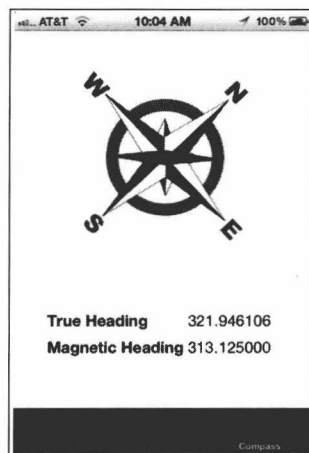


图 4-11 获取真实方向和磁力方向的值

代码清单 4-19 辅助方法

```
- (float)heading:(float)heading fromOrientation:(UIDeviceOrientation) orientation;
- (NSString *)stringFromOrientation:(UIDeviceOrientation) orientation;
```

切换到 headingViewController.m 文件，并添加代码清单 4-20 中的两个方法。

代码清单 4-20 获取基于方向的读数的方法

```
- (float)heading:(float)heading fromOrientation:(UIDeviceOrientation)orientation {
    float correctedHeading = heading;

    switch (orientation) {
        case UIDeviceOrientationPortrait:
            break;
        case UIDeviceOrientationPortraitUpsideDown:
            correctedHeading = heading + 180.0f;
            break;
        case UIDeviceOrientationLandscapeLeft:
            correctedHeading = heading + 90.0f;
            break;
        case UIDeviceOrientationLandscapeRight:
            correctedHeading = heading - 90.0f;
            break;
        default:
            break;
    }
    while ( heading > 360.0f ) {
        correctedHeading = heading - 360;
    }
    return correctedHeading;
}

- (NSString *)stringFromOrientation:(UIDeviceOrientation) orientation {
    NSString *orientationString;
    switch (orientation) {
        case UIDeviceOrientationPortrait:
            orientationString = @"Portrait";
            break;
        case UIDeviceOrientationPortraitUpsideDown:
            orientationString = @"Portrait Upside Down";
            break;
        case UIDeviceOrientationLandscapeLeft:
            orientationString = @"Landscape Left";
            break;
        case UIDeviceOrientationLandscapeRight:
            orientationString = @"Landscape Right";
            break;
        case UIDeviceOrientationFaceUp:
            orientationString = @"Face Up";
            break;
        case UIDeviceOrientationFaceDown:
            orientationString = @"Face Down";
            break;
        case UIDeviceOrientationUnknown:
            orientationString = @"Unknown";
            break;
        default:
            orientationString = @"Not Known";
            break;
    }
    return orientationString;
}
```

第一个方法基于设备的方向调整指向。这是一个相当基本的校正。结合陀螺仪和方向你可

以获得更复杂与精确的校正。由于这个例子的目的，我们假设设备是完全 90° 旋转的采用横向模式的，所以我们是用 90° 作为因数来校正。

第二个方法把设备的方向转化成人类可读的字符串。我在一些在线文章中找到了这个函数。我不知道这个功劳归功于谁，但是它非常有用。

我们必须对 `didUpdateHeading` 委托方法做一些调整，之后我们就可以测试修改的示例应用。按代码清单 4-21 中的代码更新 `didUpdateHeading` 方法。

代码清单 4-21 新版 `didUpdateHeading` 方法

```
- (void)locationManager:(CLLocationManager *)manager didUpdateHeading:(CLHeading *)newHeading {
    if (newHeading.headingAccuracy > 0) {
        //float magneticHeading = newHeading.magneticHeading;
        //float trueHeading = newHeading.trueHeading;
        UIDevice *device = [UIDevice currentDevice];
        orientationLabel.text = [self stringFromOrientation:device.orientation];

        float magneticHeading = [self heading:newHeading.magneticHeading
        fromOrientation:device.orientation];
        float trueHeading = [self heading:newHeading.trueHeading
        fromOrientation:device.orientation];

        magneticHeadingLabel.text = [NSString stringWithFormat:@"%f", magneticHeading];
        trueHeadingLabel.text = [NSString stringWithFormat:@"%f", trueHeading];

        float heading = -1.0f * M_PI * newHeading.magneticHeading / 180.0f;
        compassImage.transform = CGAffineTransformMakeRotation(heading);
    }
}
```

我们浏览一下粗体的代码块。首先，我们建立了一个代表用户的实际手机的 `UIDevice` 实例。之后，我们使用辅助方法来建立了一个人类可读的字符串，并设置了指示方向的值或标签。接下来，我们使用辅助方法计算出了基于设备方向的正确的指向值。其他的代码保持相同。

在物理设备上运行这个工程。图 4-12 和图 4-13 显示了结果。仔细观察指示方向的标签。你会看到在横向模式下指向已纠正了。

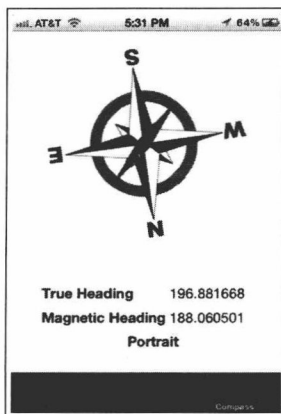


图 4-12 显示纵向模式下的指向

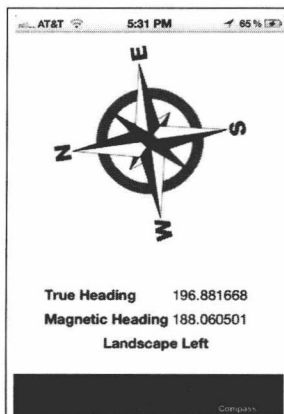


图 4-13 显示横向模式下的指向

4.2 总结

在本章中，我们做了很多关于访问传感器的工作。在学习本章的过程中，你可能感觉这有点底层或者脱离主题，但是当我们开始创建自己的增强现实应用时会非常方便。

我们讨论了加速计和陀螺仪，以及两者的结合。之后我们讲解了磁力计，并组建了一个为我们指示设备方向的示例应用。我们正慢慢地利用任何创建自己的增强现实应用所需要的元素来完善我们的工具箱。

接下来，在第 5 章中，我们会学习声音和用户反馈。

第⑤章

声音和用户反馈

无声电影仍然具有怀旧价值，并且是娱乐历史的一个关键部分。然而，无声视频游戏或者无声交互应用似乎不属于这个类别。用户体验需要有吸引力。辅助功能选项能够帮助确保所有的用户交互体验的级别相同。本章会讨论如何在应用中添加事件影响和积极反馈（例如震动）。

5.1 音频数据格式

我们从基本知识开始介绍。在 iOS 中有很多关于声音的可用选项。我们会首先讨论各种格式和它们的细微差别，之后会讨论怎样从一种格式转换成另一种。表 5-1 列出了各种 iOS 支持的格式。

表 5-1 iOS 支持的音频数据格式

数据格式	介 绍
AAC	这个代表 Advanced Audio Coding，其目的是用更好的压缩和低比特率来取代 MP3
HE-AAC	这是一个 AAC 超集，其中 HE 代表 High Efficiency。它非常适合低比特率音频（例如，流媒体音频）
AMR	这个代表 Adaptive Multi-Rate。这个格式可为语音进行优化，并有很低的比特率
ALAC	又称为 Apple Lossless，它对音频数据的编码压缩是无损的。压缩率大约是 40% ~ 60%
iLBC	这个适合语音和 VOIP 音频
IMA4	这个格式可提供 4:1 的压缩率和 16 位的音频文件
线性 PCM	PCM 代表 Pulse Code Modulation。这个格式转换模拟声音为数字格式。这个格式是无压缩的，这意味着它播放速度最快，但也占用更多的存储空间
MP3	它是这个列表中最常见的格式。MP3 是一个数字编码格式，它通过丢弃一些数据来压缩数据。这种做法叫做有损压缩（lossy compression），也是工业标准格式

5.1.1 哪一种格式适合我们呢

遗憾的是，这个问题有一个“它取决于”样式的答案。你的使用情况实际决定哪一种声音格式最适合应用。例如，如果存储空间不是问题，那么你可以使用线性 PCM 格式。它将是播放音频的最快方式，并且播放音频的时候不会影响 CPU 性能（因为它不需要同时负责解压缩文件）。但是，如果存储空间有限，你可能会为了高压缩率带来的好处而想要使用 AAC 格式来播放背景音乐或者大文件，并且可能会为了声音效果和由于 16 位文件的优势而实现的小文件而使用 IMA4 格式。

5.1.2 文件保存格式

数据格式是一件事，在哪里以及怎样保存文件是另一件事。iOS 支持各种不同的文件格式，包括 MPEG-1（.map3）、MPEG-2（.aac）、AIFF、WAVE 以及 CAF。CAF 是 iOS 上的首选格式，因为它能够包含表 5-1 中列出的所有数据格式。

5.1.3 比特率和质量

音频播放时的情景决定了特定项目的质量需求。例如，语音应用与广播交响乐团演出有非常不同的声音需求。

质量能够通过音频文件的一个叫做比特率的变量来调节。比特率是一个文件每秒使用的字节数量。表 5-1 中提到的格式中只有一部分支持比特率的调节。表 5-2 展示的是一些常用的比特率。

表 5-2 在应用中常用的比特率配置

比特率	介 绍
32 kbit/s	AM 质量音频
64 kbit/s	Podcast 质量音频
96 kbit/s	FM 质量音频
128 kbit/s	大部分普通 MP3 格式的比特率
192 kbit/s	数字质量音频
500 kbit/s	无损音频

5.1.4 采样率

音频的采样率定义了每单位时间的样本数量。通常，它以每赫兹样本量来测量，或者更普遍地描述为赫兹速率。取样频率的倒数（如果你还不知道的话）就是两个样本之间的取样间隔或者时间。

表 5-3 介绍了一些常用的采样率。

表 5-3 常用采样率和它们的使用案例

采样率	常见使用案例
8000 Hz	电话、无线电话机、对讲机、无线麦克风
11 025 Hz	低质量 PCM、MP3
32 000 Hz	MiniDV 摄像机
44 100 Hz	音频 CD 的质量（也是 iOS 中最常见的采样率）
48 000 Hz	专业录像机的音频采样率
192 000 Hz	DVD 音频

5.1.5 在 iOS 中使用而转换音频格式

你可以从互联网的各种地方下载音频。像 audiomicro.com 和 soundsnap.com 这样的网站会提供各种类别的无版权的音频。在 freesound.org 网站上也有一些共用许可的可用音频。大部分情况下，这些音频的格式并不符合你用来增强应用的要求。所以，你应该使用工具来帮助转换这些媒体格式。

我们通过 3 个典型的行动来测试媒体文件。首先，获得想要使用的文件的信息，然后转换这些文件为其他格式，最后，测试并播放这些文件。

本章的实例代码可以在 GitHub 上获得，地址：https://github.com/kylerochke/professional_iOS_AugmentedReality。如果在本地电脑上没有示例音频文件，你可以在 Xcode 工程中使用 GitHub 上的示例。

1. 获取媒体文件的信息

在 Mac OS X 中有一个叫做 `afinfo` 的工具。这个工具可在终端中使用。代码清单 5-1 显示了我将本章的 GitHub 仓库中的音频文件传递给这个命令时的输出。

代码清单 5-1 `afinfo waterfall.caf`

```
Kyle-Roches-MacBook-Pro-2:iOS_AR_Ch5_SoundUserFeedback kylederoche$ afinfo waterfall.caf
File:          waterfall.caf
File type ID:   caff
Data format:    1 ch, 44100 Hz, 'lpcm' (0x0000000C) 16-bit little-endian signed
integer
               no channel layout.
estimated duration: 1.950204 sec
audio bytes: 172008
audio packets: 86004
audio 86004 valid frames + 0 priming + 0 remainder = 86004
bit rate: 705600 bits per second
packet size upper bound: 2
audio data file offset: 4096
optimized
source bit depth: I16
sound check:
  approximate duration in seconds      1.95
----
```

这个输出的关键部分是用粗体突出显示的部分。其中一部分粗体在 `Data format` 标题的下

面。这与我们以前提到的采样率相同。44100Hz 是最常用的 CD 质量音频，也是在 iOS 应用中推荐使用的一个采样率。接下来突出的是比特率。这不是任何一个我们以前提到的比特率，但是它处在标准音频质量的范围内。最后，在最后一行，afinfo 返回了以秒为单位的大致时间长度，如果你在游戏或者以面向时间轴的方式使用这个声音，那么这非常重要。

再测试一下 afinfo。我在 iTunes 资源库中运行这个命令来查看我导入的旧音乐与从 iTunes 下载的新内容的采样的不同。代码清单 5-2 给出了一个下载音乐输出的例子。

代码清单 5-2 从 iTunes 文件返回的 afinfo 结果

```
Kyle-Roches-MacBook-Pro-2:Ukulele Songs kyleroches$ afinfo 10\ You're True.m4a
File:          10 You're True.m4a
File type ID:  m4af
Data format:   2 ch, 44100 Hz, 'aac ' (0x00000000) 0 bits/channel, 0 bytes/packet,
1024 frames/packet, 0 bytes/frame
Channel layout: Stereo (L R)
estimated duration: 203.333333 sec
audio bytes: 6491497
audio packets: 8759
audio 8967000 valid frames + 2112 priming + 104 remainder = 8969216
bit rate: 255340 bits per second

packet size upper bound: 1197
audio data file offset: 594724
optimized
format list:
[ 0] format:   2 ch, 44100 Hz, 'aac ' (0x00000000) 0 bits/channel, 0 bytes/packet,
1024 frames/packet, 0 bytes/frame
Channel layout: Stereo (L R)
source bit depth: I16
sound check:
  sc ave perceived power coeff      2973 3040
  sc max perceived power coeff     16060 12146
  sc peak amplitude msec           26006 25263
  sc max perceived power msec      25263 24520
  sc peak amplitude                32766 32766
----
```

看看这个输出，其与前面的输出相比，最大的不同是在 sound check（声音检查）代码段中增加的信息。明显的，高质量的声音文件会有更多的信息。

我们现在已经有了一个检查比特率、采样率和媒体格式的工具。接下来，我们看看怎么转换声音文件为恰当的格式。

2. 转换文件类型

我们继续使用开始测试时使用的从 GitHub 工程中获得的的声音文件。下面来介绍一个叫做 Audio File Converter（音频文件转换）的新工具，也可以叫 afconvert。afconvert 会用指定的文件和数据格式将一个源音频文件转换成一个新音频文件。代码清单 5-3 显示的是 afconvert 的使用方法。

代码清单 5-3 afconvert 使用方法

```
Usage:
afconvert [option...] input_file [output file]
  Options may appear before or after the direct arguments. If output file
  is not specified, a name is generated programmatically and the file
  is written into the same directory as input file.
afconvert input_file [-o output_file [option...]]...
  Output file options apply to the previous output_file. Other options
  may appear anywhere.
Help options:
  { -hf | --help-formats }
    print a list of supported file/data formats
  { -h | --help }
    print help
```

你能够使用 `afconvert` 来转换文件格式、媒体格式或者用一个不同的比特率来保存文件。我们使用比特率和音频文件来测试 `afconvert` 的使用。在终端窗口中运行代码清单 5-4 中显示的命令。

代码清单 5-4 使用 `afconvert` 来改变比特率

```
afconvert -d aac -f 'caff' -b 32768 waterfall.caf waterfallNew.caf
```

如果检查一下目录，你会注意到一个新文件被创建了，并命名为 `waterfallNew.caf`。我们使用 `-d` 选项来设置数据格式，使用 `-f` 选项来设置文件格式，使用 `-b` 选项来设置比特率。我们首先传递已存在的文件给 `afconvert`，之后传递目的文件。如果你用 `afinfo` 工具来查看这个新文件，你会发现比特率的不同。

3. 测试你的新声音

我要提及的最后一个工具是 `afplay`。这是另一个命令行工具，它提供测试音频文件的快速方法。`afplay` 简单地传递想播放的文件的名称，并用默认输出硬件播放音频文件。

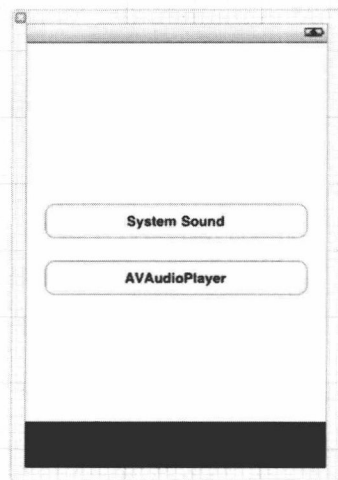
5.2 在 iOS 应用中播放声音

我们把其中的一些音频文件加入到一个 iOS 应用中。同样，本章的代码可以从 GitHub 上获得，但是如果你想从零开始学习，那么请创建一个新的 Tab Bar 应用，同时设置目标设备为 iPhone。

打开 `FirstView.xib` 文件并按图 5-1 显示的样式建立界面。

我们将会演示在 iOS 应用的后台中播放音频的两种不同方法。界面很简单。清除默认的控制并创建两个新的 `UIButton` 元素。使用如图 5-1 一样的标题。

我们将要讨论的这两种方法实现起来是相对容易的。图 5-1 两个用于启动将要创建的动作



的 UIButton

5.2.1 系统声音服务

在设备上播放声音效果的时候使用系统声音服务（System Sound Services）是一个不错的选择。它用于播放短声音效果，例如按钮单击、错误蜂鸣或者游戏动作。使用系统声音服务播放的声音不受音频会话配置的约束。这导致你不能保持系统声音服务音频的行为与应用中的其他音频的行为内联。这就是避免为任何音频使用系统声音服务的最重要原因，除了有意使用的情况外。系统声音服务需要你添加 AudioToolbox 框架到工程中。

为了确保满足作为一个快速声音播放器的目的，系统声音服务有一些限制。你的声音文件必须满足以下条件：

- 时长不超过 30 秒；
- 用线性 PCM 或者 IMA4（IMA/ADPCM）的格式；
- 以 .caf、.aif 或者 .wav 打包的文件。

另外，当你使用 AudioServicesPlaySystemSound 函数的时候：

- 声音会以当前系统音量播放，没有可编程的音量控制可用；
- 声音会立即播放；
- 循环和立体定位是不可用的；
- 同时播放是不可用的，在一个时间内你只能播放一个声音。

5.2.2 AVAudioPlayer 类

AVAudioPlayer 使用更加灵活，但也有限制。最大的功能限制是在排队和播放文件时的明显延时。因此，它最适合长时间播放的音乐、背景音乐及类似的使用情况。AVAudioPlayer 需要把 AVFoundation 框架加入到工程。使用 AVAudioPlayer 可以：

- 在任何时刻播放声音；
- 从文件或者内存播放声音；
- 循环声音；
- 同时播放多个声音（虽然没有精确同步）；
- 为播放中的每一个声音设置相关的回放级别；
- 寻找到一个声音文件的特定点，可以用来实现快进和快退的功能；
- 获取音频的功率数据，你可以在音频电平表中使用。

5.2.3 测试多重音频播放

好的，我们开始吧！你已经创建了拥有两个 UIButton 的界面。在简短介绍了播放声音的不同方式后，我使用的标题会更有意义。在开始讲解之前，我们需要添加本章中提到的两个框架到工程中。打开应用目标的构建阶段（Build Phases）选项卡，添加 AVFoundation 和

AudioToolbox 两个框架。

添加代码清单 5-5 中的代码到 FirstViewController.h 文件。

代码清单 5-5 新版 FirstViewController.h

```
#import <UIKit/UIKit.h>
#import <AudioToolbox/AudioToolbox.h>
#import <AVFoundation/AVFoundation.h>

@interface FirstViewController : UIViewController <AVAudioPlayerDelegate> {
    SystemSoundID _systemSound;
    AVAudioPlayer *_audioPlayer;
}

- (IBAction)systemSoundAction;
- (IBAction)avAudioPlayerAction;

@end
```

首先导入了 AudioToolbox 和 AVFoundation 类库到头文件中。然后指定类遵守 AVAudioPlayerDelegate 协议。在本书中，我们还没有真正地建立过一个多媒体应用，所以我不会深入讲解委托类 FirstViewController 和它的方法。但是，如果你需要在音频播放时处理事件，这将是需要进一步研究的事情。

在界面中，我们为两种使用情况创建了两个实例变量。AVAudioPlayer 需要一个指针，因为我们将会在这个类中创建实例。最后，我们声明了两个新方法来处理我们添加到用户界面中的 UIButton 的单击事件。

接下来，返回 Xcode 中的 FirstView.xib 文件，并把两个 IBAction 方法与我们的两个按钮联系起来。

切换到 FirstViewController.m 文件。在这个例子中，使用的仅仅是实例变量，所以不需要做任何合成操作。添加代码清单 5-6 中的代码到实现文件。

代码清单 5-6 处理系统声音服务方法

```
- (void)systemSoundAction {
    NSString *soundFilePath = [[NSBundle mainBundle] pathForResource:@"waterfall"
ofType:@"caf"];

    NSURL *soundFileURL = [NSURL fileURLWithPath:soundFilePath];
    AudioServicesCreateSystemSoundID((CFURLRef)soundFileURL, &_systemSound);
    AudioServicesPlaySystemSound(_systemSound);
}

- (void) avAudioPlayerAction {
    // we'll get to this next
}
```

我引用的是 GitHub 示例工程中的 CAFF 文件。如果你想要使用自己的文件替代示例中的文件，请确保名字匹配。特别需要注意的是，pathForResource 参数传入的是文件名的关键部分，而不是整个文件名。在 iOS 模拟器中运行工程并单击 System Sound Action（系统声音动作）按钮。你会注意到一个快速（即时）响应，然后文件将开始播放。

关闭模拟器。在能够完成第二个方法之前，我们还有一点工作要做。因为 AVAudioPlayer 更适合播放背景音乐，所以我想找一个比瀑布更合适的音乐，因为我不是一个瀑布爱好者（不是说有什么不对的）。瀑布音乐来自 cocos2D 示例（第 7 章我们会讲到）。另一个获得音乐的地方是 audiomicro.com，我搜索了它们的免费背景音乐以寻找一个更合适的。

打开终端，定位到下载了 MP3 的目录。如果在例子中使用的是 MP3，导航到 GitHub 的本地克隆目录。运行代码清单 5-7 中的命令。

代码清单 5-7 转换 MP3 为 CAFF 格式

```
afconvert -d aac -f 'caff' "007.mp3" backgroundMusic.caf
```

使用 afconvert 来转换 MP3 文件为 CAFF 格式，便于在工程中使用。直接播放 MP3 格式文件也是可以的，但是那就不能帮助我们演示何时使用这些工具了。

在 Xcode 工程中返回 FirstViewController.m 文件。用代码清单 5-8 中的代码完成 avAudioPlayerAction 方法。

代码清单 5-8 完成 avAudioPlayerAction 方法

```
- (void)avAudioPlayerAction {
    NSError *setCategoryError = nil;
    [[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryAmbient
    error:&setCategoryError];

    NSString *backgroundMusicPath = [[NSBundle mainBundle]
    pathForResource:@"backgroundMusic" ofType:@"caf"];
    NSURL *backgroundMusicURL = [NSURL fileURLWithPath:backgroundMusicPath];
    NSError *error;

    _audioPlayer = [[AVAudioPlayer alloc] initWithContentsOfURL:backgroundMusicURL
    error:&error];
    [_audioPlayer play];
}
```

有很多种方式可以完成这个方法。我选择最简单的方法，只播放文件。可将文件放在队列中稍后播放，或者直接从委托方法中处理文件。

在 iPhone 模拟器中运行这个工程。现在，播放这两个文件。在 AVAudioPlayer 方法中你会观察到一个明显的延时。

5.2.4 播放位置声音

在增强现实的应用中，用户通过来回移动屏幕来获得各种视图，此时就会需要播放位置音频。也可以使用 cocos2D 来实现位置音频。这个将会留在第 7 章讲解而非现在，因为我们还没有介绍 cocos2D。

5.2.5 通过震动进行用户反馈

有一些听不见的可用选项可以用来向用户提供反馈。最常用的是非常简单的震动效果。在

FirstViewController 类中创建一个叫做 `vibrate` 的新方法。添加代码清单 5-9 中的代码。

代码清单 5-9 创建 `vibrate` 方法

```
- (void)vibrate {
    AudioServicesPlaySystemSound (kSystemSoundID_Vibrate);
}
```

就是这样。在示例工程中，我创建了另一个按钮并与这个方法相关联，因此你可以在工作状态下看到它。如果在运行的时候有任何问题，请确保在设置面板里开启了 iPhone 的震动效果。

5.3 录音

到目前为止，通过本章的讲解，我们已经了解了 iOS 支持的声音类型和它们的一些复杂性，以及一些在 iOS 中如何播放声音的方法。接下来，我们花一些时间来了解如何记录和保存从设备获得的声音。

我们将使用与上一个播放音频的例子中相同的 AVFoundation 框架，不过现在是为了简化音频录制。我们已经将框架添加到工程中，所以你不需要为这个步骤担忧。如果你以一个新工程开始，那你就需要按以前讲解的那样添加框架。

初始化录音机

使应用准备好录音的第一步是建立一个 AVAudioRecorder 对象。你可以手动创建这个对象并对它进行设置，但是苹果开发者文档和大部分在线教程都建议以其他初始化选项中的一个开始。

在 Xcode 中打开 SecondViewController.h 文件，并按代码清单 5-10 中显示的内容更新它。

代码清单 5-10 新版 SecondViewController.h 头文件

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>

@interface SecondViewController : UIViewController <AVAudioRecorderDelegate,
AVAudioPlayerDelegate>{
    AVAudioRecorder *_soundRecorder;
}

- (IBAction)setupRecorder;
- (IBAction)stopRecorder;
- (IBAction)playAudioRecording;

@end
```

浏览一下这段代码，我们声明了一个 AVAudioRecorder 对象，还有 3 个方法。在最终应用中，你可能会想要合并这些音频录制动作或者利用委托方法，但是，我们要在这里把它们分开，以供参考。你可能已经注意到其他的两个项在本章出现过多次。我们导入了 AVFoundation 头文件，并指定这个类遵守 AVAudioRecorderDelegate 和 AVAudioPlayerDelegate 协议。

切换到 SecondView.xib 文件，并连接 IBAction 和一些新的 UIButton。我按图 5-2 所示的

布局我的屏幕。在界面上有 3 个 UIButton，每个都将会连接到自己的 IBAction。

我绑定 setupRecorder 到 Record Audio（记录音频）按钮。接下来，我绑定 stopRecorder 到 Stop Audio（停止音频）按钮。最后，我绑定 playAudioRecording 方法到 Play Recording（播放音频）按钮。

接下来，我们必须在 SecondViewController.m 文件中实现这些方法。

在 Xcode 中打开 SecondViewController.m 文件。我们必须实现声明的方法。我们以第一个方法 setupRecorder 开始。把代码清单 5-11 中的方法添加到实现文件中。

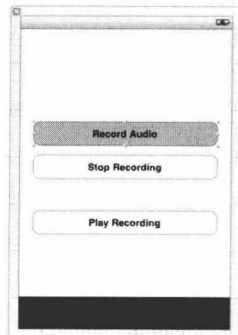


图 5-2 这里我们可以看
到测试录音的屏幕布局

代码清单 5-11 setupRecorder 方法

```
- (void)setupRecorder {
    NSString *filePath = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/recording.caf"];
    NSDictionary *recordSettings = [[NSDictionary alloc] initWithObjectsAndKeys:
        [NSNumber numberWithInt: 44100.0],
        AVSampleRateKey,
        [NSNumber numberWithInt: kAudioFormatAppleLossless],
        AVFormatIDKey,
        [NSNumber numberWithInt: 1], AVNumberOfChannelsKey,
        [NSNumber numberWithInt: AVAudioQualityMax],
        AVEncoderAudioQualityKey, nil];

    _soundRecorder = [[AVAudioRecorder alloc] initWithURL:[NSURL
        fileURLWithPath:filePath]
        settings:
        recordSettings error: nil];
    _soundRecorder.delegate = self;
    [_soundRecorder record];
}
```

这个方法以声明一个代表待保存文件的路径的 NSString 开始。接下来，建立了一个用于保存 AVAudioRecorder 属性的 NSDictionary 对象。最后，设置 AVAudioRecorder 的委托属性为 self，并使用 record 方法开启录音。

从代码清单 5-12 中复制下一个方法到 setupRecorder 的后面。

代码清单 5-12 stopRecorder 方法

```
- (void)stopRecorder {
    [_soundRecorder stop];
}
```

这个方法看起来并不难。我们只是停止了录音。

在完成最后一个方法之前，添加代码清单 5-13 中的方法到 stopRecorder 方法的下面。

代码清单 5-13 AVAudioRecorder 的委托方法

```
- (void)audioRecorderDidFinishRecording:(AVAudioRecorder *)recorder
successfully:(BOOL)flag {
    NSLog(@"did finish recording");
}
```

```

- (void)audioRecorderBeginInterruption:(AVAudioRecorder *)recorder {
    NSLog(@"recording was interrupted");
}

- (void)audioRecorderEndInterruption:(AVAudioRecorder *)recorder {
    NSLog(@"interruption ended... back to it");
}

```

我们以前建立的 `AVAudioRecorderDelegte` 定义了一些与 `AVAudioRecorder` 交互的方法。我们只是对这些方法进行记录，以使你知道这些事件是何时触发的。在这组动作中唯一会触发的是 `audioRecorderDidFinishRecording` 方法。其他的两个只是为了测试的趣味性。如果要在一个 iPhone 上测试，可以在调试的过程中给自己打电话；当铃响的时候，会触发 `audioRecorderBeginInterruption` 方法。当挂断电话的时候，会触发 `audioRecorderEndInterruption` 方法。这些事件是委托接口的一部分。

我们录制音频会话，然后将它存储在本地文件系统中。最后一步是从本地硬盘中回放这个音频。添加代码清单 5-14 中的方法到我们刚刚定义的 3 个委托方法的后面。

代码清单 5-14 回放音频

```

- (void)playAudioRecording {
    NSString * filePath = [NSHomeDirectory() stringByAppendingPathComponent:
@"Documents/recording.caf"];
    AVAudioPlayer *newPlayer = [[AVAudioPlayer alloc] initWithContentsOfURL: [NSURL
fileURLWithPath:filePath] error: nil];
    newPlayer.delegate = self;
    [newPlayer play];
}

```

这段代码应该看起来很熟悉。在本章开篇，我们学习了与之相同的音频播放过程。我们现在已经准备好测试应用。要正确地做到这一点，你应该正使用一个物理设备进行测试而非一个模拟器。虽然在模拟器上大部分功能都工作良好，但是最好还是在一个真实的设备上测试录音。

运行这个应用并切换到第二个选项卡。从顶部开始，单击 `Record Audio` 按钮。要么对着麦克风讲话，要么确保有一个足够的背景噪声来获得一个真实的样品录制。完成后，单击 `Stop-Recording` 按钮。在控制台中，你会看到记录信息 `did finish recording`。最后单击 `Play Recording` 按钮，通过扬声器你会听到声音的回放。请确保你的电话没有强制静音。

5.4 总结

本章学习了 iOS 音频格式的不同内部构件。我们讨论了比特率和文件格式，以及在 iOS 中支持哪种文件格式，哪种是首选格式。我们也讲解了一些可以用来测试声音文件和转换格式的可用工具。

在第 6 章，我们将从音频切换到视频，并开始奠定增强现实编程的基础。在这一点上，第 6 章会覆盖将要创建的 application 的各种基本组件，真正接触类似增强现实的应用。

第⑥章

摄像头和视频采集

增强现实应用程序通常有一个共同点：它们都建立在一个实时的视频资源之上。

在本章中，我们以摄像头使用的基本概念开始介绍，之后快速进入视频采集的高级例子进行讲解，最后逐帧分析视频。

在后面的章节中，我们使用这些基本概念来创建覆盖在视频资源之上的应用。

6.1 快速浏览

在第2章，我们讨论了当你想要在代码中访问一个传感器或者硬件组件之前，检查其存在性和可用性的重要性。

你可以使用 UIImagePickerController 类编写程序代码来监测你设备上的摄像头是否可用。有一个叫做 isSourceTypeAvailable 的方法，我们可以用来确定想要使用的摄像头类型是否可用。代码清单 6-1 显示了我们在第2章使用的用来监测一个摄像头是否存在以及当它可用的时候使用前置摄像头的例子的方法。

代码清单 6-1 检查摄像头是否存在，之后检查前置摄像头

```
BOOL cameraAvailable = [UIImagePickerController
isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera];
    BOOL frontCameraAvailable = [UIImagePickerController
isSourceTypeAvailable:UIImagePickerControllerCameraDeviceFront];

    if (cameraAvailable) {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Camera"
                                                                message:@"Camera Available"
                                                                delegate:self
                                                                cancelButtonTitle:@"OK"
                                                                otherButtonTitles:nil, nil];

        [alert show];
        [alert release];
    } else {
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Camera"
                                                                message:@"Camera NOT Available"
                                                                delegate:self
                                                                cancelButtonTitle:@"OK"
                                                                otherButtonTitles:nil, nil];

        [alert show];
        [alert release];
    }
}
```

```

if (frontCameraAvailable) {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Camera"
                                                         message:@"Front Camera
Available"
                                                         delegate:self
                                                         cancelButtonTitle:@"OK"
                                                         otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
} else {
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Camera"
                                                         message:@"Front Camera NOT
Available"
                                                         delegate:self
                                                         cancelButtonTitle:@"OK"
                                                         otherButtonTitles:nil, nil];

    [alert show];
    [alert release];
}

```

这个代码块会在一个快速 UIAlertView 弹出窗口中显示结果。实际上，在这个例子中有两个弹出窗口。在代码清单 6-1 中的前面几行，你可以看到我们正在检查 UIImagePickerControllerSourceTypeCamera 的存在性，以此来判断摄像头是否可用。接下来我们使用 UIImagePickerControllerCameraDeviceFront 参数检查了前置摄像头的存在性。isSourceTypeAvailable 方法返回一个布尔值。我们在 if/else 语句中使用这个布尔值，以为每一个检查显示恰当的 UIAlertView。

在第 2 章中，我们曾提到过 Xcode 模拟器不支持摄像头或视频采集。因此，你必须在一个物理设备上运行本章中的所有例子。

在讲解录像之前，我们先从基本的拍照开始介绍。

6.2 拍照

在 iOS 中通过编程来拍一张照片并不是一件非常困难的任务。我们用来确定摄像头是否可用的 UIImagePickerController 类提供了一个访问摄像头、拍一张照片，甚至是预览结果的简单方法。

为本章创建一个新的 Xcode 工程。我已经创建了一个叫做 Ch6 的工程，源码可以在 Apress 网站的 Source Code/Download 区域获得，或者可以从 https://github.com/kyleroch/Professional_iOS_Augmentedreality 获取。确保你使用的是 tab bar application 模板，并设置你的 device family 为 Universal。图 6-1 显示的是我的设置。请注意，在本章中我们正在使用的是 Automatic Reference Counting（自动引用计数）选项（iOS 5 新加的），通过选中 Use Automatic Reference Counting 复选框来开启。

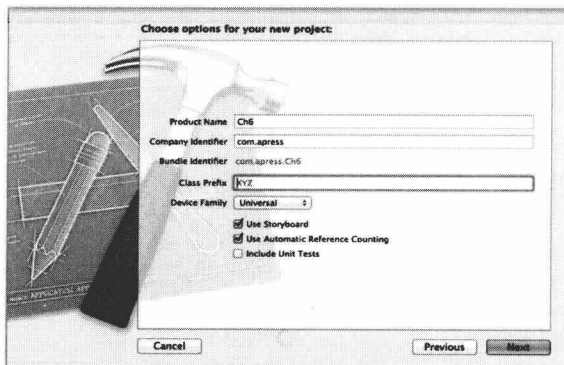


图 6-1 本章 Xcode 工程的设置

6.2.1 使用故事板

在本书中，我们还没有讨论过故事板（storyboard）。我们花点时间做一个快速介绍。故事板为你的所有应用视图提供了一个统一的 XIB 文件。你仍然可以通过编程从一个单独的 interface builder 类加载 XIB 文件，但使用故事板你会有更多的灵活性和节省时间的机会。例如，图 6-2 显示的是基于 tab bar 界面的默认故事板。

不用故事板的话，将会有 3 个不同的界面文件，并全都需要不同类型的连接来整合在一起。使用故事板，不用切换 context（上下文）就可以得到应用程序流程的完整外观。

我们可以直接使用提供的默认布局。然而，为了能够更好地理解故事板，我们删除这些，从零开始。从 Xcode 工程里面删除 FirstViewController.h、FirstViewController.m、SecondView Controller.h 和 SecondViewController.m 文件，同时，删除与每一个视图相联系的 .png 文件。

打开两个故事板文件。在设计窗口中单击 FirstViewController 和 SecondViewController。然后按两次 Delete。第一次按 Delete，你从视图中移除了类，得到一个蓝色高亮边框的空视图；第二次按 Delete，彻底地删除了视图。在两个视图控制器中重复这个过程。

在 interface builder（界面构建者）视图中，你会得到一个空的 tab bar 控制器。你的工程应该只剩下一个 AppDelegate，两个故事板文件和支持文件目录。现在有一个干净的工程，让我们手动地把它创建回来。

我们将以拍一张照片并把它保存在资源库中为开始来进行创建。因此，我们为此例子添加一个视图控制器，并在 tab bar 控制器中的一个 tab 中加载它。右击工程文件夹并从弹出菜单中选择 New File 命令。选择 UIViewController 子类作为这个文件的模板。单击 Next 按钮。在选项窗口确保没有选项被选中，并命名这个类为 PhotoViewController。在单击 Next 按钮之前，确保你的选项窗口与图 6-3 显示的相同。

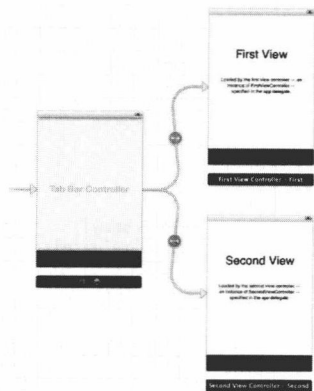


图 6-2 为 tab bar 界面模板使用故事板

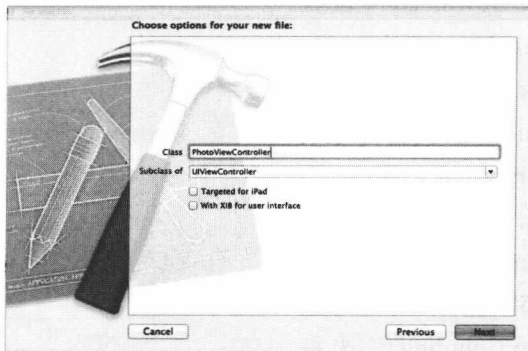


图 6-3 创建 PhotoViewController 类

从 Object 资源库中拖曳一个视图控制器 (View Controller) 到界面。单击新的视图控制器, 并切换到 identity inspector (标识检查器) 并修改 Custom Class (自定义类) 为 PhotoViewController。

在继续讲解之前, 确保你已指定 PhotoViewController 类遵守 UINavigationController Delegate 和 UIImagePickerControllerDelegate 协议。

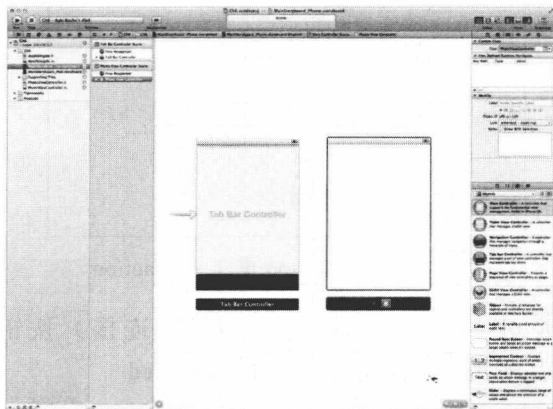


图 6-4 添加 PhotoViewController

在界面构建器中右击 Tab Bar Controller (选项卡栏控制器) 视图。在 Storyboard Sequences 类别下, 有一个叫做 Relationship-viewControllers 的项。Ctrl+ 点击 + 拖曳项目右边的“加图标”到界面中的 Photo View Controller (照片视图控制器) 视图。现在你会看到与图 6-5 类似的情景。

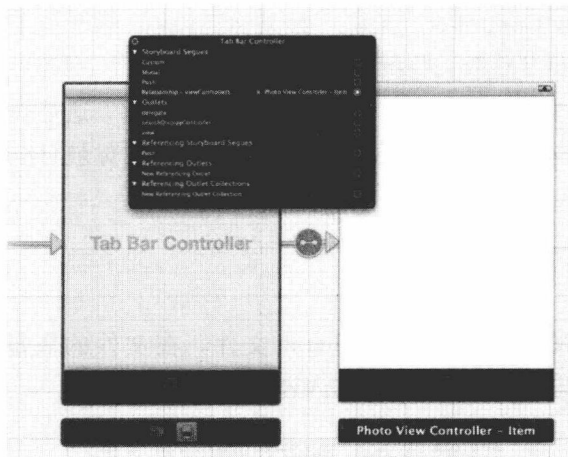


图 6-5 连接 Relationship Storyboard Seque 到 PhotoViewController

为 iPad 的故事板重复这个过程, 如果你正使用 iPad 进行测试, 或者如果你实在是想要应用运行在所有两个设备上。

你需要在相关视图中改变刚刚开启的与 tab bar 项相关的名字、图标或者标记。你无法再从父 tab bar 控制器中编辑这些项目。

接下来，拖动一个 UIButton 到界面构建器中的 PhotoViewController 的中间。确保开启了 assistant editor（辅助编辑器），并且呈现出了 PhotoViewController.h，按住 Ctrl 键，同时拖曳 UIButton 到头文件以创建一个与 outlet 相联系的动作（动作）。使用图 6-6 作为参考。

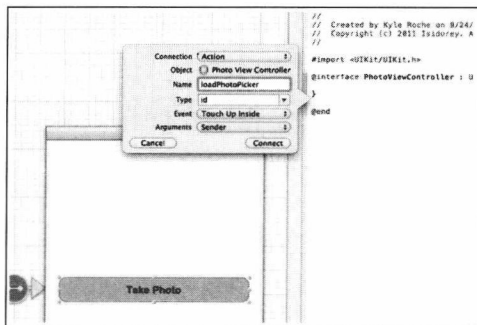


图 6-6 为 UIButton 创建一个 action

命名 action 为 loadPhotoPicker，然后单击 Connect（连接）按钮。如果你正使用一个 iPad，或者想要你的应用是通用的，你需要（再次）在 iPad storyboard 里面重复这一步。但是，连接 action 到已经创建的事件有点不同。打开 iPad 故事板并添加一个相似的 UIButton 到界面。我已经把我的两个 UIButton 按钮的标题全部设置为 Take Photo。为了连接 UIButton 到一个现有的 action，你必须 Ctrl+ 单击 + 拖动 UIButton 到故事板视图的下面的视图控制器（View Controller）图标。如果你还没有看明白这个过程，参考图 6-7 获取一些指导。

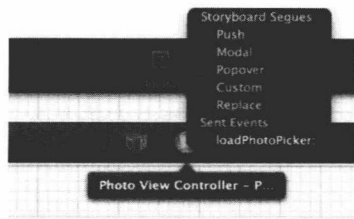


图 6-7 当连接一个 outlet 到一个现有 action 的时候使用这个菜单

对于简单地连接一个按钮来说，这个指导有点多，不是吗？这是因为我想要详细地讲解这个过程，那么，当我们在本章后面建立更复杂的例子的时候，你就可以把它作为参考。我们继续讲解下面的内容。

6.2.2 使用摄像头

在 Xcode 中，切换到 PhotoViewController.m 文件。在文件的底部，我们上一步添加的 action 方法是空的。按代码清单 6-2 中显示的内容扩展 action 方法。

代码清单 6-2 loadPhotoPicker 方法

```
-(IBAction)loadPhotoPicker:(id)sender {
    UIImagePickerController *imagePicker = [[UIImagePickerController alloc] init];
    imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;
    // uncomment for front camera
    // imagePicker.cameraDevice = UIImagePickerControllerCameraDeviceFront;
    imagePicker.delegate = self;
    imagePicker.allowsEditing = NO;
    [self presentViewController:imagePicker animated:YES];
}
```

这个方法首先创建了一个 UIImagePickerController 类的实例。这个类实质上打开了在 iOS 中我们都会用到的摄像头接口。我们设置 sourceType 属性为后置摄像头。如果你更想设置 SourceType 的属性为前置摄像头（同时，你应该首先验证该摄像头的存在性），你可以反注释这行下面已经注释的代码。

接下来，我们设置 delegate 为 self，并禁止编辑图像采集器。之后，我们在 PhotoView Controller 的一个模态对话框里呈现 UIImagePickerController。

如果你现在运行它，图像采集器（摄像头）也会有显示，但什么都不会保存。添加代码清单 6-3 中的方法到实现中。

代码清单 6-3 保存照片图像的方法

```
- (void) imagePickerController:(UIImagePickerController *)picker
didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    UIImage *image = [info objectForKey:@"UIImagePickerControllerOriginalImage"];

    UIImageWriteToSavedPhotosAlbum(image, self,
    @selector(image:didFinishSavingWithError:contextInfo:), nil);
}

- (void)image:(UIImage *)image didFinishSavingWithError:(NSError *)error
contextInfo:(void *)contextInfo
{
    UIAlertView *alert;
    if (error) {
        alert = [[UIAlertView alloc] initWithTitle:@"Error"
        message:@"Unable to save image to Photo
        Album."
        delegate:self cancelButtonTitle:@"Ok"
        otherButtonTitles:nil];
    } else {
        alert = [[UIAlertView alloc] initWithTitle:@"Success"
        message:@"Image saved to Photo Album."
        delegate:self cancelButtonTitle:@"Ok"
        otherButtonTitles:nil];
    }
    [alert show];
    [self dismissModalViewControllerAnimated:YES];
}
```

第一个方法是 UIImagePickerController 类的委托方法。当控制器中的一个图片被选中的时候，该方法会被触发。我们为这个图片分配了一个对象，并把它发送给代码清单 6-3 中的第二个方法。

第二个方法，保存文件之后设置为 selector 的方法。用于检查错误并在一个 UIAlertView 控件中显示成功或者失败信息。你可能已经注意到，我没有释放 UIAlertView 实例。在本章早期，当创建这个工程的时候，我们开启了 Automatic Reference Counting（自动引用计数）。Automatic Reference Counting（iOS 5 新加的功能）禁止对释放（release）方法的显式调用。

我们已经准备好对工程进行测试。在 iPhone 或者 iPad 设备上启动该工程。因为我们要使用摄像头，所以这个工程不能运行在模拟器上。

单击界面中间的按钮，照一张相片，并单击 Use 按钮来选择那个照片。你会看到一个确认文件正确保存的 UIAlertView 对话框。为了测试，打开使用的 iPad 或者 iPhone 的 Photo 应用。你会发现照片已经保存在那里。

6.2.3 以不同的格式保存图像

到目前为止，我们已经保存了一个图像到照片库中。把这个图像用于其他目的，如对象或者面部识别怎么样？在本书后面的章节我们会在面部识别当中使用视频缓冲中的图像。UIKit 包含了一些帮助我们将图像导出到 iOS 设备的文件上的 C 函数。用于导出的最常用的两个格式是 JPEG 和 PNG，因此，我们先看重介绍这两个。

在 Xcode 中打开 PhotoViewController.m 文件。找到 didFinishPickingMediaWithInfo 方法，并添加代码清单 6-4 中的代码到这个方法的末尾。

代码清单 6-4 转换并保存 UIImage 对象

```
// start saving files

NSString *pngPath = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/ConvertedPNG.png"];
NSString *jpgPath = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/ConvertedJPEG.jpg"];

[UIImageJPEGRepresentation(image, 1.0) writeToFile:jpgPath atomically:YES];
[UIImagePNGRepresentation(image) writeToFile:pngPath atomically:YES];

// optional (check for files)
NSError *error;
NSFileManager *fileMgr = [NSFileManager defaultManager];
NSString *documentsDirectory = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents"];
NSLog(@"Documents: %@", [fileMgr contentsOfDirectoryAtPath:documentsDirectory
error:&error]);
```

首先，我们设置了两个字符串，用来表示包含最终文件名的完整目录。我们接着使用 C 函数（UIImageJPEGRepresentation 和 UIImagePNGRepresentation）来保存文件到相应的目录。我们传递图像压缩质量作为参数设置的一部分。图像的压缩率是以 0.0 ~ 1.0 的比例来计量的。因此，我们设置的是用最好的质量和尽可能小的压缩率来保存图像。

使用任一物理设备运行这个工程。确保你的控制台窗口已经打开，以使你能够看到输出。图片保存之后，你会看到与图 6-8 相似的情景。

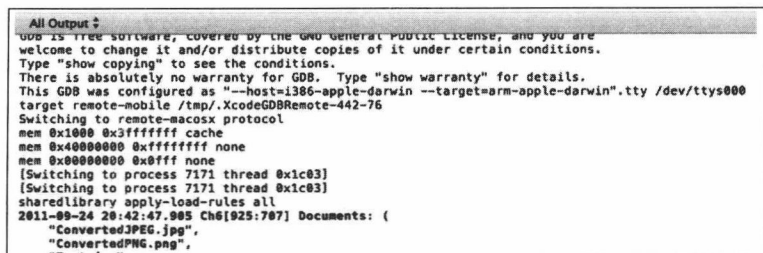


图 6-8 图像被保存、转换并存储

6.2.4 通过电子邮件发送图像

也许保存文件还不够，你更喜欢把图片附加到一个电子邮件消息中。如果你想通过编程来发送电子邮件消息，你首先需要导入 MessageUI 框架。

单击 Xcode 导航器中的工程名字。切换到 Build Phases（构建阶段）选项卡并导入 MessageUI 框架。下一步，在 Xcode 中打开 PhotoViewController.h 文件并按代码清单 6-5 中显示的内容更新 interface。

代码清单 6-5 更新 PhotoViewController 的 interface

```
#import <UIKit/UIKit.h>
#import <MessageUI/MessageUI.h>
#import <MessageUI/MFMailComposeViewController.h>

@interface PhotoViewController : UIViewController <UINavigationControllerDelegate,
UIImagePickerControllerDelegate, MFMailComposeViewControllerDelegate> {
}
- (IBAction)loadPhotoPicker:(id)sender;

@end
```

我们导入了几个头文件并指定这个类遵守 MFMailComposeViewControllerDelegate 协议，因此我们能够处理用户发送消息时的事件回调。添加代码清单 6-6 中的方法到实现。

代码清单 6-6 更新 PhotoViewController 类的接口

```
- (void)sendMessage:(UIImage *)image
{
    NSLog(@"Sending Email");

    MFMailComposeViewController *picker = [[MFMailComposeViewController alloc] init];
    picker.mailComposeDelegate = self;

    [picker setSubject:@"iOS Augmented Reality - Chapter 6"];

    [picker setToRecipients:[NSArray arrayWithObjects:@"kyle.m.roche@gmail.com", nil]];

    NSString *emailBody = @"Hi Kyle, this stuff actually works.";
    [picker setMessageBody:emailBody isHTML:NO];

    NSData *data = UIImagePNGRepresentation(image);

    [picker addAttachmentData:data mimeType:@"image/png" fileName:@"Ch6ScreenShot"];

    [self presentViewController:picker animated:YES];
}

- (void)mailComposeController:(MFMailComposeViewController*)controller
didFinishWithResult:(MFMailComposeResult)result error:(NSError*)error
{
    [self dismissModalViewControllerAnimated:YES];
}
```

我们逐行讲解这两个方法，先以 sendMessage 方法开始讲解。首先，分配一个新的 MFMailComposeViewController 对象并设置委托属性为 self，以使我们能够处理事件回调。接

下来，我们设置了邮件的主题、接收地址和消息主体。最后，我们把照片附加到邮件中，并在一个模态对话框中向用户呈现草稿邮件。

接下来，我们定义了另一个叫做 `didFinishWithResult` 的方法。这个方法是 `MFMComposeViewControllerDelegate` 协议的委托方法。当用户真正发送了邮件之后就会触发该方法。当这个事件触发的时候，我们移除了该模态对话框（因为不再需要）。

我们几乎做好了测试这个代码的准备。首先，必须声明 `sendMessage` 方法，以使我们能够在实现中调用它。添加代码清单 6-7 中的代码到 `PhotoViewController.m` 文件的 `import` 语句的后面。

代码清单 6-7 声明 sendMessage 为 Private 方法

```
@interface PhotoViewController (Private)
- (void)sendMessage:(UIImage *)image;
@end
```

在 `didFinishSavingWithError` 方法的末尾添加对 `sendMessage` 方法的调用。如果你现在运行这个代码，你会得到一个有趣的结果。似乎一切都在按预期工作，但是在本该这个邮件对话框显示的时刻你会得到一个与代码清单 6-8 相似的错误。

代码清单 6-8 错误消息

```
2011-09-24 21:35:32.153 Ch6[1128:707] Sending Email
[Switching to process 9731 thread 0x2603]
wait fences: failed to receive reply: 10004003
```

这不是通常的错误描述，对吧？我们会讲解它的含义。通常 `wait_fences` 信息表明一个正在运行的动画与一个新的请求动画产生了冲突。在例子中，当我们试图创建一个 `withAnimation` 属性为 YES 的新的模态对话框时，我们也正在 `withAnimation` 属性为 YES 的状态下移除一个模态对话框。无论你运行这个工程多少次，这一行都永远无法成功运行通过。

所以，在测试这个工程之前，我们必须对以前的代码做一些调整。按代码清单 6-9 中显示的内容更新 `didFinishSavingWithError` 消息。

代码清单 6-9 更新 didFinishSavingWithError

```
-(void)image:(UIImage *)image didFinishSavingWithError:(NSError *)error
contextInfo:(void *)contextInfo
{
    /*UIAlertView *alert;
    if (error) {
        alert = [[UIAlertView alloc] initWithTitle:@"Error"
        message:@"Unable to save image to Photo
Album."
        delegate:self cancelButtonTitle:@"Ok"
        otherButtonTitles:nil];
    } else {
        alert = [[UIAlertView alloc] initWithTitle:@"Success"
        message:@"Image saved to Photo Album."
        delegate:self cancelButtonTitle:@"Ok"
        otherButtonTitles:nil];
```

```

    }
    [alert show];*/
    [self dismissModalViewControllerAnimated:NO];

    [self sendEmailMessage:image];
}

```

现在，你可以看到粗体显示的代码处在一个注释块中。我们做的第二处修改（也以粗体标注）是设置了动画标记为 NO，因此将不会有冲突。对于经验老道的读者，我需要指出的是，这个问题还有很多解决方式。然而，大量的动画会留下踪迹或者模糊问题以及其他的复杂问题，这些问题本来通过类似以上的简单修改就可以很容易地避免。

图 6-9 显示的是工程的成功执行，同时包含正确格式的邮件的对话框信息。

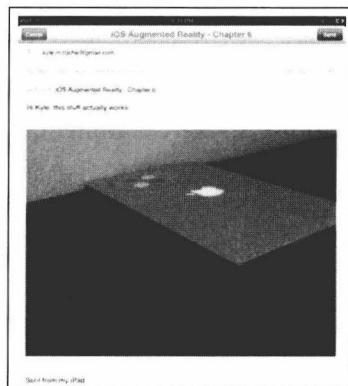


图 6-9 正接收一个正确格式的邮件

6.3 视频捕获

创建一个在视频资源之上的应用有两种主要方法。首先有一种不需要分析视频帧的方法。这种方法常在一些基于位置的增强现实应用中看到，在第 11 章中，我们会学习更多关于这种方法的知识。这些方法使用方向和位置类，以及陀螺仪来确定增强现实目标的方向。但是视频只是一个噱头。看到视频上基于位置的对象会给你它们是一个整体的感觉。这个方法（节省了时间和计算机资源）是一种打开视频预览作为应用基础的简单方式。

第二，有一种需要对视频资源进行分析的方法。这种方法可以见于基于标记的增强现实应用中（参见第 10 章），以及面部识别的增强现实应用（参见第 13 章）中。这两种应用都需要分析视频的每一帧。

6.3.1 建立一个视频预览基础

在第 7 章，我们将会讨论如何覆盖 cocos2D 层到预览视频之上来建立增强现实游戏。因此本节是对这些核心概念的简单介绍。

在 Xcode 中打开 PhotoViewController.m 文件，并找到 loadPhotoPicker 方法。这个方法打开了 UIImagePickerController 对象，并允许用户拍照。按代码清单 6-10 中显示的内容更新这个方法。

代码清单 6-10 更新 loadPhotoPicker 方法

```

- (IBAction)loadPhotoPicker:(id)sender {
    UIImagePickerController *imagePicker = [[UIImagePickerController alloc] init];
    imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;
    // uncomment for front camera
    //imagePicker.cameraDevice = UIImagePickerControllerCameraDeviceFront;
    imagePicker.cameraDevice = UIImagePickerControllerCameraCaptureModeVideo;
    imagePicker.showsCameraControls = NO;
    imagePicker.toolbarHidden = YES;
}

```

```
imagePicker.navigationBarHidden = YES;  
imagePicker.wantsFullScreenLayout = YES;  
  
imagePicker.delegate = self;  
imagePicker.allowsEditing = NO;  
[self presentViewController:imagePicker animated:YES];  
}
```

我们为 UIImagePickerController 对象设置了几个新的选项。首先，我们设置设备类型为视频摄像头。之后，我们隐藏了摄像头的控制器、工具栏和导航栏。最后，我们命令视图控制器让 UIImagePickerController 以全屏模式显示。

经过这些修改后，如果你运行这段代码，你会得到与图 6-10 类似的结果。



图 6-10 没有控制器的全屏视频

这个干净的视频界面为我们提供了一个用于建立增强现实应用的完美背景。

如果我们想要分析这些视频的帧，那么不得不建立一个 NSTimer，并每隔一段时间捕获、保存预览画面。另外，我们还可以设置一些用户动作以保存图片，或者设置一些程序性循环。所有这些选项没有一个是完美的，并且从本章中的上一个例子中你可能已经注意到手动保存图片要耗费 1 ~ 2 秒的处理器时间。我们探讨一个捕获用于分析的视频帧的更好方法。

6.3.2 为帧捕获建立基础

帧捕获会话的工作有点不同。为了演示这一点，我们将会使用一个叫做 AVCaptureSession 的新类。我们进入例子应用的一个单独选项卡，之后我们就能方便地参考差异。

首先，我们添加将会用来与 AV Foundation 类库一起工作的框架。添加以下框架到工程中：

- CoreVideo
- CoreMedia
- AVFoundation
- ImageIO

下一步，添加一个新的 UIViewController 的子类文件到工程中。确保你没有选中 Targeted-for iPad 和 With XIB for user interface。命名这个类为 VideoViewController。

参见前一个例子建立另一个选项卡（tab bar），并将其连接到 iPad 和 iPhone 的故事板。打开故事板文件，按照我们以前的同样步骤将它们连接到选项卡。完成后，你会在 iPhone 的故事板里看到与图 6-11 相似的结果。

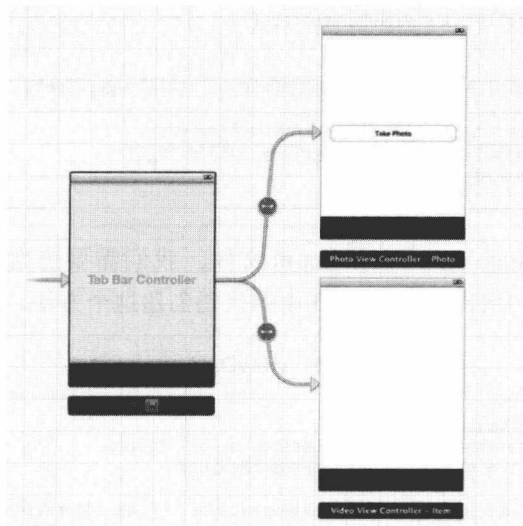


图 6-11 添加新的 tab bar 项目到 iPhone 的故事板

为 iPad 的故事板执行相同的步骤。在两个界面上，我们会添加几个插座变量。首先，添加一个 UIView 到 VideoViewController NIB 文件。命名这个 UIView 为 videoPreview。确保你连接了这个插座变量到 VideoViewController 的头文件。当 VideoViewController.h 文件在 assistant-editor（辅助编辑器）中打开的时候，导入 AVFoundation.h 和 ImageIO/CGImageProperties.h 头文件。你的新 interface 文件应该看起来与代码清单 6-11 相似。

代码清单 6-11 更新 VideoViewController.h

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import <ImageIO/CGImageProperties.h>

@interface VideoViewController : UIViewController {
}
@property (strong, nonatomic) IBOutlet UIView *videoPreview;
@end
```

接下来，我们将添加一个 UIImageView outlet 和一个 UIButton 到 VideoViewController 布局里。命名 UIImageView 为 videoImage，并连接一个称为 captureScreen 的动作到 UIButton。

为了从预览视频里面保存一个静态图片，我们将会使用一个 AVCaptureStillImageOutput 对象。为这个对象创建一个叫做 stillImageOutput 的属性。

VideoViewController.h 应该如代码清单 6-12 所示。

代码清单 6-12 更新 VideoViewController.h

```

#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import <ImageIO/CGImageProperties.h>

@interface VideoViewController : UIViewController {
}
@property(n nonatomic, retain) AVCaptureStillImageOutput *stillImageOutput;

@property (strong, nonatomic) IBOutlet UIView *videoPreview;
@property (strong, nonatomic) IBOutlet UIImageView *videoImage;
- (IBAction)captureScreen:(id)sender;
@end

```

在 Xcode 中切换到 VideoViewController.m 文件。我们需要创建一个 viewDidLoad 方法来开启摄像头的预览会话。用代码清单 6-13 中的代码创建这个方法。

代码清单 6-13 viewDidLoad 方法

```

- (void)viewDidLoad:(BOOL)animated {
    AVCaptureSession *session = [[AVCaptureSession alloc] init];
    session.sessionPreset = AVCaptureSessionPresetMedium;

    AVCaptureVideoPreviewLayer *captureVideoPreviewLayer = [[AVCaptureVideoPreviewLayer
alloc] initWithSession:session];
    captureVideoPreviewLayer.frame = self.videoPreview.bounds;
    [self.videoPreview.layer addSublayer:captureVideoPreviewLayer];

    AVCaptureDevice *device = [AVCaptureDevice
defaultDeviceWithMediaType:AVMediaTypeVideo];

    NSError *error = nil;
    AVCaptureDeviceInput *input = [AVCaptureDeviceInput deviceInputWithDevice:device
error:&error];
    if (!input) {
        NSLog(@"ERROR: trying to open camera: %@", error);
    }
    [session addInput:input];
    // Placeholder
    [session startRunning];
}

```

在继续测试这个应用之前，我们先浏览一下这段代码：首先，创建了一个 AVCaptureSession 的实例；然后，设置应用的视频质量为中等；接下来，创建了一个 AVCaptureVideoPreviewlayer 实例。这个实例将被用于存储视频缓冲器的实时预览。

接下来，我们设置帧填满 videoPreview UIView 的大小。之后建立了上一个例子中讲到的 AVCaptureDevice，并检查确认它是可用的。

最后的几行是非常重要的。如果你没有用 startRunning 方法开启 AVCaptureSession，那么视图中什么都不会显示。

现在，你可以运行一下应用。你会看到 UIView 被摄像头的预览填充。这并不令人兴奋，不是吗？这在照片例子中已经实现过。我们扩展这个例子，以便在不影响实时预览的情况下，UIButton 能够调用一个从视频缓冲器捕获静态图片的例程。

在 Xcode 中, 返回 VideoViewController.m 文件。在 viewDidLoad 方法的末尾, 我们留下了一个注释标记 //Placeholder。找到该标记区域并用代码清单 6-14 中的代码替换掉。

代码清单 6-14 替换 Placeholder 注释行

```
stillImageOutput = [[AVCaptureStillImageOutput alloc] init];
NSDictionary *outputSettings = [[NSDictionary alloc] initWithObjectsAndKeys:
    AVVideoCodecJPEG, AVVideoCodecKey, nil];
[stillImageOutput setOutputSettings:outputSettings];

[session addOutput:stillImageOutput];
```

这段代码有一个特殊的目的。它从摄像头捕获静态图像并将它保存到以前创建的 AVCaptureStillImageOutput 对象。

找到 captureScreen 方法。用代码清单 6-15 中的代码更新这个方法。

代码清单 6-15 captureScreen 方法

```
- (IBAction)captureScreen:(id)sender {
    AVCaptureConnection *videoConnection = nil;
    for (AVCaptureConnection *connection in stillImageOutput.connections)
    {
        for (AVCaptureInputPort *port in [connection inputPorts])
        {
            if ([[port mediaType] isEqual:AVMediaTypeVideo] )
            {
                videoConnection = connection;
                break;
            }
        }
        if (videoConnection) { break; }
    }

    NSLog(@"about to request a capture from: %@", stillImageOutput);
    [stillImageOutput captureStillImageAsynchronouslyFromConnection:videoConnection
    completionHandler:^(CMSampleBufferRef imageSampleBuffer, NSError *error)
    {
        CFDictionaryRef exifAttachments = CMGetAttachment( imageSampleBuffer,
        kCGImagePropertyExifDictionary, NULL);
        if (exifAttachments)
        {
            // Do something with the attachments.

            NSLog(@"attachements: %@", exifAttachments);
        }
        else
            NSLog(@"no attachments");

        NSData *imageData = [AVCaptureStillImageOutput
        jpegStillImageNSDataRepresentation:imageSampleBuffer];
        UIImage *image = [[UIImage alloc] initWithData:imageData];

        self.videoImage.image = image;
    }];
}
```

在这个方法中有两个主要的代码块。第一个代码块检查 AVCaptureStillImageOutput 对象中的 AVCapture Connection。我们这样做是为了确认这是一个视频输出, 并设置一个

AVCaptureConnection 实例的本地引用。

接下来,我们异步地从摄像头捕获图像。我们建立了一个图像缓冲来保存图像。之后,我们使用 ImageIO 框架来捕获图像的 EXIF 附件信息。这对建立过滤器或者保存照片质量的值以作其他用途可能是非常有用的。

最后,我们设置用 UIImageView outlet 显示捕获的视频帧。如果运行这个工程,你会看到与图 6-12 相似的情景。

我们的例子运行完美。当你捕获一个屏幕图时,你会听到扬声器发出的快门声。这是因为我们捕获静态图片的方式。在第 13 章的例子应用中,我们将会轻微地修改这个方法以移除快门声响效果。

最后,我们快速浏览一下静态图片的 EXIF 附件信息。代码清单 6-16 显示的是控制台的输出。

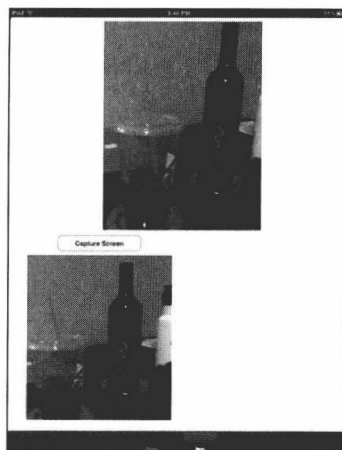


图 6-12 我们捕捉的葡萄酒和冰咖啡的视频迹象

代码清单 6-16 EXIF 附件

```
2011-09-25 15:46:40.751 Ch6[183:707] attachments: {
    ApertureValue = "2.526068811667588";
    BrightnessValue = "-1.04630972052318";
    ExposureMode = 0;
    ExposureProgram = 2;
    ExposureTime = "0.041666666666666666";
    FNumber = "2.4";
    Flash = 32;
    FocalLenIn35mmFilm = 32;
    FocalLength = "2.03";
    ISOSpeedRatings = (
        800
    );
    MeteringMode = 5;
    PixelXDimension = 640;
    PixelYDimension = 480;
    SceneType = 1;
    SensingMethod = 2;
    Sharpness = 0;
    ShutterSpeedValue = "4.584985584026477";
    WhiteBalance = 0;
}
```

6.4 总结

在本章中,我们学习了 iOS 设备上的视频摄像头及其功能的关键概念。我们以基本的拍照开始介绍。我们建立了一个 UIImagePickerController 对象来捕获用户选中的图片。接下来,我们学习了如何将这些照片保存和转换到本地文件系统。最后,我们以附件的形式通过 E-mail 发送这个文件。与此同时,我们介绍了 iOS 5 的新特性故事板。

接下来,我们介绍了视频捕获。这将是本书剩下的大部分章节的主要组成部分。我们学

习了怎样移除摄像头控制和怎样使用全屏显示的视频预览。另外，我们建立了一个视频捕获会话，以便在标记或者面孔识别的应用中能够分析帧。

在第7章，我们将介绍 cocos2D，它是创建游戏的重要框架。我会告诉你如何使用 cocos2D 以及本章的概念，以便为你的第一个增强现实游戏建立基础。

第⑦章

把 cocos2D 用于增强现实

我们已经讨论了增强现实应用的各种使用案例。除了基于位置信息的应用之外，游戏应用占据了增强现实市场空间的大部分比重。

在本章中，我将介绍用于 iPhone 的 cocos2D。iPhone 版 cocos2D 是一个用于建立 2D 游戏和图形交互应用的框架。它基于 cocos2D 框架，你可以在 www.cocos2d.org 网站上学习更多相关知识。

7.1 概况

iPhone 版 cocos2D 的基础是一个开源的 cocos2D 框架。原工程是用 Python 写就的。显然，原版并不适合 iOS 程序员。iPhone 版 cocos2D 是这个框架的 Objective-C 版分支。

cocos2D 有一些关键特性，例如它的易用性、速度和灵活性，还有它是开源的。cocos2D 恰好是一个拥有庞大社区支持的开源项目。最后，cocos2D 最重要的特性（如果你想发布一个应用）是它的使用是获得 App Store 认可的。

7.2 安装

首先，我们需要建立恰当的环境。在本章中，我们将会逐步讲解安装、核心概念，同时引入一个简单的增强现实例子应用。在第 8 章，我们会用 cocos2D 建立一个完整的增强现实游戏。

访问 www.cocos2d-iphone.org/download 并下载用于 iPhone 的最新稳定版 cocos2D。当文件下载后，在 Finder 中双击它并解压文档。

在新目录下，有一个叫做 `cocos2D-ios.xcodeproj` 的文件。这是用于测试 cocos2D 下载和先决条件的 Xcode 工程。在 Xcode 中打开工程，工程中应该有接近 70 个目标。它们中的大部分用于测试 iPhone 版 cocos2D 的一个特定功能。修改 Scheme（模式）来指定一个测试目标以确保你安装了所有需要的。参见图 7-1，它显示了如何在 Xcode 中调整你的策略。

为了更好地测试，多换几个目标，在模拟器上多运行几个工程。它们大部分是用于演示某一个特定功能的一连串单击操作的例子。

7.2.1 安装工程模板

cocos2D 为 Xcode 提供工程模板。刚才的下载文件包含了安装模板需要的所有文件。这些模板提供了工程需要的必要组件，并可以节省建立时间。有 3 个主要的工程模板：

- cocos2D stand-alone 模板
- cocos2D+box2D 模板
- cocos2D+chipmunk 模板

在运行这些脚本之前，确保你已经把解压的目录放在了一个你中意的位置。打开终端程序并导航到所在目录。为了安装模板，运行代码清单 7-1 中显示的命令。

代码清单 7-1 安装模板（失败）

```
cocos2d-iphone-1.0.1$ ./install-templates.sh -u
cocos2d-iphone template installer
Installing Xcode 4 cocos2d iOS template
-----
templates already installed. To force a re-install use the '-f' parameter
```

如果你已经用过 cocos2D，你可能会得到一个与我接收到的相似的消息（代码清单 7-1 显示的）。如果是这样，则按错误消息中指出的添加 -f 选项。代码清单 7-2 显示了结果。如果你之前从来没有使用过 cocos2D，你的输出应该与代码清单 7-2 相似，并且不需要添加 -f 选项。

代码清单 7-2 用 -f 选项安装

```
cocos2d-iphone-1.0.1$ ./install-templates.sh -f -u
cocos2d-iphone template installer
Installing Xcode 4 cocos2d iOS template
-----
removing old libraries: /Users/kylerocher/Library/Developer/Xcode/Templates/cocos2d/
...creating destination directory:
/Users/kylerocher/Library/Developer/Xcode/Templates/cocos2d/
...copying cocos2d files
...copying cocoslive files
...copying TouchJSON files
...copying CocosDenshion files
...copying CocosDenshionExtras files
...copying FontLabel files
...copying template files
[much more of this type of stuff... then...]
done!
```

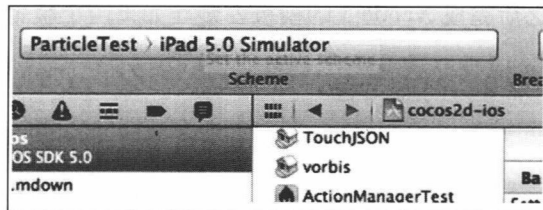


图 7-1 从 Scheme 下拉菜单里面选择 ParticleTest（或者另一个目标）

为简单起见，我简化了代码清单 7-2 的运行输出。安装器将循环安装 Xcode 4 的各种模板，之后完成 Xcode 3 的模板安装（如果两个版本你都在使用的话）。

7.2.2 创建工程

打开 Xcode，如果正确安装了模板，你应该在启动屏幕上看到与图 7-2 类似的情景。

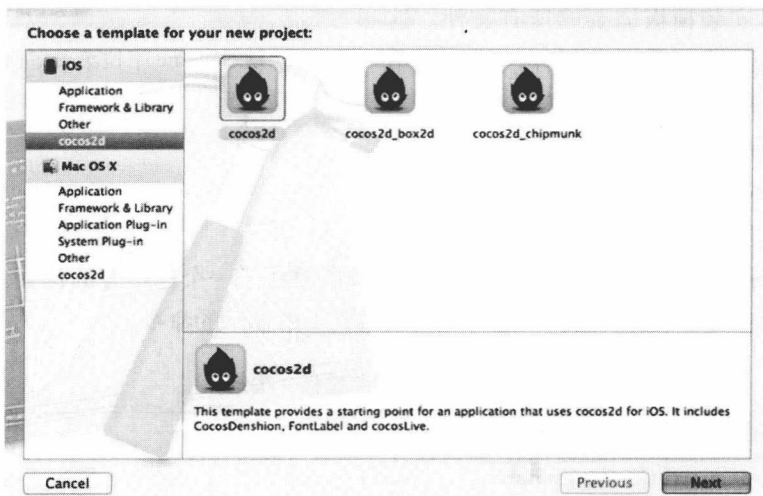


图 7-2 成功安装了新的 cocos2D 模板

选择 cocos2D 模板，并创建一个新工程。命名这个工程为 Ch7。这个工程的代码可以在 www.github.com/kylerochke 上获得，或者可以在 Apress 网站（www.apress.com）的 SourceCode/Download 区域获得。

以默认格式运行工程。使用 iPad 或者 iPhone 模拟器。你会得到一个与图 7-3 相似的结果。这是所有 cocos2D 模板的默认工程。



图 7-3 运行 cocos2D 的默认工程 Hello World

我们转换这个 Hello World 为一个增强现实版的 Hello World 应用，之后讨论了一些 cocos2D 中的可用功能。

7.3 初识增强现实应用视图

如你所知，增强现实应用是建立在一个实时的摄像头视图背景之上的。在例子工程中，我们使用一个黑色屏幕作为背景。这是我们首先需要替换的东西。

在 7.1 节中，我简短地提到了 iPhone 版 cocos2D 是基于 cocos2D 的，cocos2D 是 OpenGL 用来渲染 2D 世界的一个库。OpenGL 是这两个 cocos2D 源码类库的基础。因此，我们必须用摄像头视图替换叫做 drawable 的层。drawable 层是 OpenGL 对 EAGLDrawable 协议的实现，这个协议是用来渲染界面和向屏幕显示 EAGLContext 对象的。在第 10 章中，当创建基于标记的增强现实应用时，我们会学习更多关于 EAGLContext 的知识。从现在开始，必须意识到我们不能使用 cocos2D 模板默认的 16 位缓存格式来显示摄像头视图。我们必须切换到 32 位的颜色格式。

7.3.1 调整默认视图

在 Xcode 中打开 AppDelegate.h 文件。按代码清单 7-3 中显示的代码添加一个 UIView 到界面中。

代码清单 7-3 为摄像头覆盖图添加一个 UIView

```
UIView *cameraView;
```

切换到 AppDelegate.m 文件，让我们开始工作。首先，我们必须将缓存转换为 32 位缓存。这是 cocos2D 模板里面的一个注释掉的值，这是开发者的常见修改。找到看起来与代码清单 7-4 相似的行。

代码清单 7-4 修改 EAGLView 的 pixelFormat

```
//
// Create the EAGLView manually
// 1. Create a RGB565 format. Alternative: RGBA8
// 2. depth format of 0 bit. Use 16 or 24 bit for 3d effects, like CCPageTurnTransition
//
EAGLView *glView = [EAGLView viewWithFrame:[window bounds]
                    pixelFormat:kEAGLColorFormatRGB565      //
                    kEAGLColorFormatRGBA8                    //
                    depthFormat:0];
```

你可以看到注释对这个进行了一些解释。因为我们不能为摄像头视图使用 RGB565、16 位格式，所以用注释掉的那个值替换 pixelFormat 的当前值。现在，代码块看起来应该与代码清单 7-5 相似。我们将会设置 depthFormat 的值为 0，因为当前还没有在 3D 下工作。

代码清单 7-5 修改 pixelFormat 的值

```
EAGLView *glView = [EAGLView viewWithFrame:[window bounds]
                    pixelFormat: kEAGLColorFormatRGBA8 depthFormat:0];
```

接下来，我们将会添加新声明的 UIView 到屏幕以替换默认的黑色背景。找到代码清单 7-6 显示的代码段。

代码清单 7-6 主视图接收一个子视图

```
// make the View Controller a child of the main window
[window addSubview: viewController.view];
```

在这行的正下方，添加代码清单 7-7 中的代码。接下来，我将逐行为你解释。

代码清单 7-7 为摄像头准备覆盖视图

```
[CCDirector sharedDirector].openGLView.backgroundColor = [UIColor clearColor];
[CCDirector sharedDirector].openGLView.opaque = NO;

glClearColor(0.0, 0.0, 0.0, 0.0);

cameraView = [[UIView alloc] initWithFrame:[UIScreen mainScreen] bounds];
cameraView.opaque = NO;
cameraView.backgroundColor=[UIColor clearColor];
[window addSubview:cameraView];
```

首先，我们设置 OpenGL 视图的背景颜色为 clearColor。这将允许设置透明度，以便我们能够添加对象和文本到屏幕内摄像头视图的上部。接下来，我们确保这个层不是不透明的（为了同样的目的）。我们使用 glClearColor 方法来确保“clear”版本实现真正的透明。你可以稍微调节一下这个地方，例如用一个模糊半透明的层来代替本章中正在创建的不可见的层。第二段代码看起来类似。我们创建了新的 UIView，设置它扩展为屏幕大小，并设置为透明状态。最后，我们添加新 UIView 到窗口。

如果你再次运行工程，你将都看不到可视后的区别。我们还没有添加任何产生影响的东西。

7.3.2 添加摄像头视图

就在代码清单 7-7 代码的下方，我们将会添加代码以创建 UIImagePickerController，并以摄像头模式显示。其中一些代码会让我们联想到第 6 章。在代码清单 7-7 的代码正下方添加代码清单 7-8 中的代码。

代码清单 7-8 创建 UIImagePickerController，并把它添加到屏幕

```
UIImagePickerController *imagePicker;

@try {
    imagePicker = [[[UIImagePickerController alloc] init] autorelease];
    imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;
    imagePicker.showsCameraControls = NO;
    imagePicker.toolbarHidden = YES;
    imagePicker.navigationBarHidden = YES;
    imagePicker.wantsFullScreenLayout = YES;
}
@catch (NSException * e) {
    [imagePicker release];
    imagePicker = nil;
}
```

```

}
@catch (NSException * e) {
    [imagePicker release];
    imagePicker = nil;
}
@finally {
    if(imagePicker) {
        [cameraView addSubview:[imagePicker view]];
        [cameraView release];
    }
}

[window bringSubviewToFront:viewController.view];

```

在运行这个工程之前，我们讲解一下这段代码。首先，我们创建了 UIImagePickerController，并确保它的 source type（资源类型）设置为 Camera（摄像头），以便看到实时的视频内容。我们隐藏了控制、工具栏和导航栏，以便得到一个全屏的摄像头视图。

还有最后一件事情要做。切换到 HelloWorldLayer.m 文件，按代码清单 7-9 显示的内容修改标签的文本和大小。

代码清单 7-9 修改标签的文本

```

// create and initialize a Label
CCLabelTTF *label = [CCLabelTTF labelWithString:@"Hello Augmented World"
    fontName:@"Marker Felt" fontSize:48];

```

与第 6 章的例子相似，因为模拟器不支持摄像头，所以我们必须在一个真实的设备上运行例子工程。在设备上启动工程。你会注意到图像有点偏移，如图 7-4 所示。



图 7-4 有点偏移的摄像头视图

摄像头没有覆盖整个屏幕！在边缘有一个黑带，就好像模板应用没有填充整个屏幕。稍等，我们设置 wantsFullScreenLayout 为 YES 后发生了什么？

7.3.3 缩放摄像头视图

那么，为什么会有这个奇怪的黑条占据 UI 的一部分呢？这是因为屏幕和摄像头的纵横比

是不同的（在横向模式下）。基本上，iPhone 屏幕有一个 3:4 的纵横比，而 iPhone 摄像头有一个 4:3 的纵横比。因此，我们需要缩放摄像头图像以匹配。这解释了图 7-4 中的黑条。

为了修复这个问题，我们需要缩放 UIImagePickerController 来匹配设备的纵横比。在 Xcode 中返回到 AppDelegate.m 文件。找到从代码清单 7-8 添加的代码段，并把代码清单 7-10 中的粗体行加入到这段代码中。

代码清单 7-10 缩放 UIImagePickerController

```
@try {  
    UIImagePickerController *imagePicker = [[[UIImagePickerController alloc] init] autorelease];  
    imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;  
    imagePicker.showsCameraControls = NO;  
    imagePicker.toolbarHidden = YES;  
    imagePicker.navigationBarHidden = YES;  
    imagePicker.wantsFullScreenLayout = YES;  
    imagePicker.cameraViewTransform =  
    CGAffineTransformScale(imagePicker.cameraViewTransform, 1.0, 1.3);  
}
```

再次运行工程。你会看到我们成功地移除了黑条，现在摄像头视图占据了整个屏幕，如图 7-5 所示。



图 7-5 没有黑条

7.4 cocos2D 的概念

我们添加一些闪光点到应用中。现在，它并不响应任何触摸或者展示任何有趣的事情。首先，我们熟悉一下一个 cocos2D 应用是如何建立的。

7.4.1 场景

场景（scene）是 cocos2D 视图的基础。你的应用可以有多个场景，但是无论何时，它们中

只有一个可以处于激活状态。在例子工程中，我们在 HelloWorldLayer 场景的下面添加了摄像头视图。

在 cocos2D 中，场景是由 CCScene 对象创建的。它们是应用的整个工作流的独立部分。在典型的游戏中，有各种场景，有用于目录菜单的，有用于不同关卡的，有用于关卡之间胜利或者失败后的过场动画内容的。这些 CCScene 对象中的每一个都是由一个或多个层（layer）组成的。在例子应用中，我们有带有 label（标签）的干净的层，以及带有 UIImagePickerController 的干净的层。

你也可以把场景用于场景之间的过渡。这种类型的场景是用 CCTransitionScene 对象创建的。

7.4.2 控制器

毫不奇怪，控制器（director）会用于管理当前哪一个场景处于激活状态。控制器是一个 CCDirector 的 singleton 对象的实例。它维护内存中哪一个场景是激活的详细信息，同时管理一个允许推入一个新场景、暂停当前场景，以及返回原先场景的栈。真正修改 CCScene 的是 CCDirector 的 singleton。这个 singleton 同时初始化 OpenGL ES 环境。代码清单 7-11 中的代码来自 AppDelegate.m，它用单例类（或者共享的类）初始化了控制器对象。

代码清单 7-11 创建 Director Singleton 的实例

```
CCDirector *director = [CCDirector sharedDirector];
```

7.4.3 图层

一个 CCLayer 拥有整个可绘制区域的大小。它可以是半透明的（为下一层提供一个洞）或者对下层是全透明的（如我们在前面的应用中做的那样）。层定义了一个场景的外观和行为。

CCLayer 同时定义了事件处理者。当一个传播事件的时候，它从最前面的层开始，并继续传递事件，直到某一个层捕获并接收了该事件。因此，在栈中，越是后面的层越有机会截获并处理事件。

cocos2D 定义了一系列在游戏中可用的预定义层。很少需要用户创建自己的 CCLayer 类或者扩展。

7.5 添加效果

cocos2D 有很多内建的、可以在游戏中使用的效果和过渡。对于增强现实应用，在呈现任何可视化效果之前，我们将会使用触摸事件来处理来自用户的交互。因此，我们首先需要学习如何处理触摸事件。

7.5.1 处理触摸事件

还有另一个 singleton 对象, CCTouchDispatcher, 它用于发送屏幕上的触摸事件消息。为了在应用中能够处理触摸事件, 我们必须开启触摸功能, 注册 CCTouchDispatcher, 然后处理触摸事件本身。

在概述中, 我简短地提到事件要在每一个单独的层中处理。它们从最前面的层开始逐步传递到最后面的层, 直到一个层接受并处理了这个事件。

截止到现在, 我们创建的示例与一般的 2D 游戏有点不同。我们有一个总是处于栈底部的基础层 (摄像头视图)。我们使用一个简单的层 (带有标签视图的层) 向屏幕显示内容。典型的 2D 游戏是建立在静态图形层或者平铺地图之上的。之所以可以成为一个增强现实的游戏是因为以实时的摄像头视图作为基础。在第 8 章将要创建的例子应用中, 我们会添加更多的高级增强现实概念。尽管如此, 对于触摸事件是如何传递的理解会最大化地帮助我们处理 HelloWorldLayer 层的事件。

在 Xcode 中打开 HelloWorldLayer.m 文件并找到 init 方法。在 if 循环的底部添加代码清单 7-12 的语句。

代码清单 7-12 开启触摸

```
self.isTouchEnabled = YES;
```

下一步, 我们必须把一个层注册到 CCTouchDispatcher 以接收这些类型的事件。在 init 方法的后面添加代码清单 7-13 中的代码。

代码清单 7-13 注册到 CCTouchDispatcher

```
- (void)registerWithTouchDispatcher {
    [[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:0
    swallowsTouches:YES];
}
```

设置 delegate 属性为 self, 并吞没了这个触摸以保证在我们确定这个层能否处理并接收当前事件之前没有任何其他层会做出响应。使用 ccTouchBegan 方法来接收事件。就在 registerWithTouchDispatcher 方法的后面添加代码清单 7-14 中的代码。

代码清单 7-14 接收并处理事件

```
- (BOOL)ccTouchBegan:(UITouch *)touch withEvent:(UIEvent *)event {
    return YES;
}

- (void)ccTouchEnded:(UITouch *)touch withEvent:(UIEvent *)event {
    CGPoint location = [self convertTouchToNodeSpace:touch];
    NSLog(@"Touch %@: ", NSStringFromCGPoint(location));
}
```

第一个方法向调度器 (dispatcher) 返回一个 YES, 让收发器知道它会接收该事件, 而不

需要再去寻找响应者了。第二个事件通过在屏幕上记录触摸位置来对触摸做出响应。我们使用 `NSStringFromCGPoint` 方法来正确转换 C 结构体为一个字符串。

在设备上再次运行应用。在一些随机位置触摸屏幕。在调试器窗口，你会看到与图 7-6 相似的输出。

```

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    CGPoint touchPoint = [self convertTouchToNodeSpace:touches.firstObject];
    NSLog(@"Ch7: Touch at (%f, %f)", touchPoint.x, touchPoint.y);
}

Ch7[1720:707] cocos2d: GL supports NPOT textures: YES
Ch7[1720:707] cocos2d: GL supports discard_framebuffer: YES
Ch7[1720:707] cocos2d: compiled with NPOT support: NO
Ch7[1720:707] cocos2d: compiled with VBO support in TextureAtlas : YES
Ch7[1720:707] cocos2d: compiled with Affine Matrix transformation in CCNode : YES
Ch7[1720:707] cocos2d: compiled with Profiling Support: NO
Ch7[1720:707] cocos2d: surface size: 480x320
Ch7[1720:707] cocos2d: Frame interval: 1
Ch7[1720:707] Touch {239.5, 207.5}:
Ch7[1720:707] Touch {291, 92}:
Ch7[1720:707] Touch {85.5, 222.5}:
Ch7[1720:707] Touch {380, 152}:

```

图 7-6 记录触摸事件

我们之后会使用这个 `CGPoint` 来做一些更令人兴奋的事情。我们简短地讨论一下效果，然后添加一个效果到我们的场景中。

7.5.2 视觉效果

cocos2D 提供了各种我们可以用在应用中的图像效果。例如，下面的清单包含了在 cocos2D 的分发中可用的各种粒子发生器：

- `CCParticleFire`
- `CCParticleFireworks`
- `CCParticleSun`
- `CCParticleGalaxy`
- `CCParticleFlower`
- `CCParticleMeteor`
- `CCParticleSpiral`
- `CCParticleSmoke`
- `CCParticleExplosion`
- `CCParticleSnow`
- `CCParticleRain`

有一些方法可用于设计你自己的粒子系统。有一个叫做 Particle Designer（可从 <http://particledesigner.71squared.com/> 获得）的工具可以让你创建自己的粒子效果，并提供一个可访问的社区共享效果库。在写作本书时，这个工具是低于 10 美元的。

图像效果并不是本书的重点，因此如果你想获取更多信息的话，参考 Apress (www.apress.com) 网站上任何关于 cocos2D 的主题。

回到工程，在 Xcode 中打开 HelloWorldLayer.m 文件。因为已经处理了触摸事件（通过在控制台窗口记录），这有助于我们用一些粒子效果扩展应用的功能性。这里，我们的目的是让用户的触摸在屏幕上产生爆炸或者类似的效果。

找到 CCTouchEnded 方法。在你设置本地变量语句的后面添加代码清单 7-15 中的代码。

代码清单 7-15 添加 CCParticleSystem

```
CCParticleSystem* emitter = [CCParticleExplosion node];
emitter.position = ccp(location.x, location.y);
emitter.life = 3.0f;
emitter.duration = 2.7f;
emitter.lifeVar = 0.1f;
emitter.totalParticles = 200;
[self addChild:emitter z:20];
```

快速浏览一下本节开头列出的 cocos2D 的可用粒子效果。在这个代码块的第一行，你会注意到我们使用了 CCParticleExplosion 类。当你在本节中进行测试的时候，可以自由地修改粒子效果为其他任何一个本节列出的效果。

在声明了 CCParticleSystem 后，我们设置位置为触摸事件的 x, y 坐标，定义了粒子系统的生命周期，并设置了总粒子数。同时，这些都是需要调节和学习的重要的值。

最后，我们添加 CCParticleSystem 到层。如果你运行这个工程，并在屏幕上点击，你会看到与图 7-7 相似的情景。

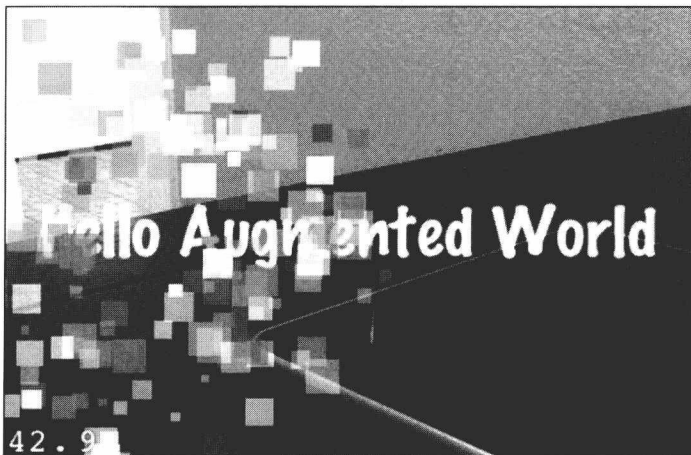


图 7-7 点击来产生一个触摸爆炸效果

7.5.3 添加声音效果

没有声音的爆炸就不叫爆炸，你同意吧？我们制造一些响声。回顾第 5 章，我们讨论了如

何添加声音到 iOS 应用中。我介绍了一些你可以下载免费声音效果的网站。第 5 章展示了如何下载一个 MP3，并使用 afconvert 工具将它转换成 CAF 文件。示例的源码包含在 GitHub 中，也可以从 Apress 网站（www.apress.com）的 Source Code/Download 区域下载。我命名要创建的文件为 explosion.caf。如果你想使用自己的声音效果，请确保名字与以下代码块匹配。

在 Xcode 中打开 HelloWorldLayer.m 文件，并按代码清单 7-16 中显示的方法导入 SimpleAudioEngine 头文件。

代码清单 7-16 导入 SimpleAudioEngine 头文件

```
#import "SimpleAudioEngine.h"
```

下一步，定位 init 方法。在设置 isTouchEnabled 为 YES 的语句后面添加代码清单 7-17 中的代码。

代码清单 7-17 预加载声音效果

```
[[SimpleAudioEngine sharedEngine] preloadEffect:@"explosion.caf"];
```

预加载声音效果是为了当我们需要时可以快速访问该效果。最后，找到 ccTouchEnded 事件处理方法。在添加粒子效果的语句后面添加代码清单 7-18 中的代码。

代码清单 7-18 播放声音效果

```
[[SimpleAudioEngine sharedEngine] playEffect:@"explosion.caf"];
```

在你的设备上运行工程。在触摸事件里，你应该有了一个很好的爆炸声音效果。这将有助于游戏为用户带来更多的乐趣。

7.6 添加 HUD 层

在 cocos2D 游戏中，游戏世界通常比设备当前呈现的屏幕视图更大。因此，如果你直接在屏幕上呈现元素的话，元素的位置可能会与场景的当前焦点对不齐。HUD（heads-up display）层通常用于解决这个问题。HUD 层是位于场景中活动层上部的一个半透明层。在游戏中，我们看到过很多用于显示得分，生命值等的 HUD 层。

我们添加一个 HUD 层到当前例子以计算用户引发的爆炸数量。添加一个新的 Objective-C 类到工程。这个类应该是一个 NSObject 的子类。命名这个类为 HUDLayer.mm，.mm 扩展名并非印刷错误。使用能够混合 Objective-C 和 C++ 语法的 .mm 文件是一个良好的习惯。这比告诉编译器以不同的方式处理单一的文件更容易，并且它让后来阅读代码的开发者知道你使用了 C++。

打开 HUDLayer.h 文件，并用代码清单 7-19 中显示的代码替换它。

代码清单 7-19 新版 HUDLayer.h

```
#import "cocos2d.h"
@interface HUDLayer : CCLayer {
    CCLabelTTF *_counterLabel;
}
- (void)incrementCounter;
@end
```

我们把 HUD 类修改为了 CCLayer 的子类，并创建了一个私有（private）的 CCLabelTTF 对象在屏幕上显示标签的值。我们还创建了一个叫做 incrementCounter 的实例方法。触摸事件会调用这个方法让 HUD 层知道出现了一个新的触摸，并增加标签显示的数值。

在 Xcode 中切换到 HudLayer.mm 文件。首先，导入 HelloWorldLayer 头文件。接下来，我们将会创建一个本地变量用来存储当前数量的值，并为 HUDLayer 创建 init 方法。复制代码清单 7-20 中的代码到 HUDLayer.mm 文件。

代码清单 7-20 HUDLayer.mm 的 init 方法

```
int counter = 0;
- (id)init {
    if ((self = [super init])) {
        _counterLabel = [CCLabelTTF labelWithString:[NSString
stringWithFormat:@"Explosions: %d", counter] fontName:@"Marker Felt" fontSize:24];

        CGSize size = [[CCDirector sharedDirector] winSize];

        _counterLabel.position = ccp( size.width * 0.85 , size.height * 0.9 );

        [self addChild: _counterLabel z:10];
    }
    return self;
}
```

init 方法会使用 HelloWorldLayer 类的一些代码。我们简单地创建了一个新的标签并把它添加到屏幕上。在这里，我们以 Z 轴的值为 10，添加它到屏幕的右上方，因此任何爆炸都会得以体现（数字变化）。

通过添加代码清单 7-21 中的代码来实现 incrementCounter 方法，把这个方法放在 init 方法的后面。

代码清单 7-21 incrementCounter 方法

```
- (void)incrementCounter {
    counter++;
    _counterLabel.string = [NSString stringWithFormat:@"Explosions: %d", counter];
}
```

incrementCounter 方法应该很容易理解。我们简单地增加了存储在 HUD 层而不是游戏层的 counter（计数器），并更新了添加到视图中的标签控件。

以上就是对 HUD 层的介绍。现在，我们必须从 HelloWorldLayer 层使用它。首先，在 Xcode 中切换到 HelloWorldLayer.h，并导入 HUDLayer.h 文件。接下来，我们声明了一个

HUDLayer 类型的叫做 *_hud 的私有变量。最后，我们创建了一个叫做 initWithHUD 的新方法。你的新版 HelloWorldLayer.h 文件应该看起来与代码清单 7-22 相似。

代码清单 7-22 新版 HelloWorldLayer.h

```
#import "cocos2d.h"
#import "CCTouchDispatcher.h"
#import "HUDLayer.h"

// HelloWorldLayer
@interface HelloWorldLayer : CCLayer {
    HUDLayer *_hud;
}

// returns a CCScene that contains the HelloWorldLayer as the only child
+(CCScene *) scene;
- (id)initWithHUD:(HUDLayer *)hud;
@end
```

切换到 HelloWorldLayer.m 文件，并按代码清单 7-23 修改静态 scene 方法。

代码清单 7-23 新版 scene 方法

```
+(CCScene *) scene
{
    CCScene *scene = [CCScene node];

    HUDLayer *hud = [HUDLayer node];
    [scene addChild:hud z:1];

    // 'layer' is an autorelease object.
    //HelloWorldLayer *layer = [HelloWorldLayer node];
    HelloWorldLayer *layer = [[[HelloWorldLayer alloc] initWithHUD:hud] autorelease];

    // add layer as a child to scene
    [scene addChild: layer];

    // return the scene
    return scene;
}
```

我们讨论一下粗体行。首先，我们声明了一个新的 HUDLayer 变量并将它添加到场景。我们注释掉旧的 init 方法，并用一个叫做 initWithHUD 的方法来替换它。（我们还没有实现这个方法，但是不要着急，接下来就是。）

代码清单 7-24 用这个代码块替换 init 方法

```
-(id) initWithHUD:(HUDLayer *)hud
{
    // always call "super" init
    // Apple recommends to re-assign "self" with the "super" return value
    if( (self=[super init])) {
        _hud = hud;
        // create and initialize a label
        CCLabelTTF *label = [CCLabelTTF labelWithString:@"Hello Augmented World"
        fontName:@"Marker Felt" fontSize:48];

        // ask director the the window size
        CGSize size = [[CCDirector sharedDirector] winSize];
```

```
// position the label on the center of the screen
label.position = ccp( size.width /2 , size.height/2 );

// add the label as a child to this Layer
[self addChild: label];

self.isTouchEnabled = YES;
[[SimpleAudioEngine sharedEngine] preloadEffect:@"explosion.caf"];
}
return self;
}
```

除了方法签名，我们只添加了一行到 initWithHUD 方法。我们设置私有变量 _hud 为我们在静态 scene 方法中创建的新 HUDLayer 对象。

在测试这个应用之前还有一件事要做。找到 ccTouchEnded 方法，并添加代码清单 7-25 中的代码到 NSLog 语句的上方。

代码清单 7-25 调用 HUDLayer 的 incrementCounter 方法

```
[_hud incrementCounter];
```

我们的新 HUD 层应该被加载到场景中，并会在屏幕右上部处理当前 label 数值的增加。运行设备上的工程。声音和粒子效果会保持不变。但是你应该在顶部的标签看到你触摸屏幕的次数。触摸几次，你应该最终看到与图 7-8 相似的情景。



图 7-8 在屏幕的右上部你会看到 HUD 层

你可以添加更复杂的、能够自己处理用户输入的 HUD 层，如按钮或者选项。这些我们不会在本书中讲解。想要获取更多的关于 cocos2D 的信息或者高级游戏主题，参考 Apress 网站上关于 cocos2D 的主题。

7.7 总结

本章介绍了 cocos2D。我们建立了例子模板，并把只有一个标签控件的 Hello World 工程转换成了一个简单的增强现实游戏。我们缩放摄像头视图来匹配设备在横向模式下的纵横比。之后，我们为用户触摸事件建立了事件处理器。

我们创建了一个简单的游戏来处理用户触摸事件，为事件可视化创建、开启了一个粒子系统，甚至还有一个爆炸声音效果。我们还创建了一个独立于游戏层的 HUD 层来计数，并向用户显示该计数。

在第 8 章，我们将会让增强现实的游戏更上一层楼，并使用陀螺仪来创建一个 360 度的第一人称增强现实射击游戏。

第⑧章

构建 cocos2D 增强现实游戏

在本章中，我们将会以第 7 章中学到的知识为基础来扩展并构建一个完整的增强现实游戏。有很多关于增强现实游戏的剧本或故事线。虽然这个情节有点可笑，但它演示了增强现实游戏的一些重要功能。我们将学习如何构建一个有菜单和启动选项的 cocos2D 游戏。我们也会处理分数和游戏结束场景。整个游戏将会运行在一个摄像头视图之上。

8.1 概述

每个游戏都开始于一个故事。我们的故事是：一些外星南瓜已经接管了我们的星球。宇宙警察都配备了一个特殊的装置，可以让我们看到南瓜，并能通过在摄像头视图上单击来摧毁它们。当然，此装置是 iPad 或 iPhone。

当游戏启动时，我们将会使用一个叫做 CIDetector 的 iOS 5 的新类。第 12 章会详细讨论这个类。现在，你只需要知道这个类可以在摄像头视图中识别人物的面孔，并允许我们覆盖南瓜图片到人物头部。当在摄像头视图发现面孔的时候，你会看到类似图 8-1 的情景。



图 8-1 覆盖在人脸上的南瓜（示例）

我们已经为编码做好了准备。在第7章，我们安装了 cocos2D 并学习了一个简单的例子。在本章中，我们将会使用与 cocos2D 的安装相同的模板。更精确地说，我们将会使用包含 cocos2D 和 chipmunk 物理类库的模板。在游戏中，我们不会使用 chipmunk 做太多的事情。我们使用它来让用户觉得菜单屏幕更有趣一点。

这个游戏的计分会有些困难。因为我们随机地放置南瓜到人物的脸部，之后打掉它们，我们必须考虑一个南瓜应该在多长时间内重现一次。为了避免产生一个赢不了的游戏，我们将会使用一个静态的南瓜集合。南瓜们很可能会不止一次地发现相同的受害者（除非你在更大范围内测试这个游戏）。在用户清除了整个外星南瓜集团的时候，我们会让用户赢得游戏。我们还会放置一个计时器，以使外星南瓜有一个公平的继续存在的机会。

8.2 创建工程

如果你跳过了第7章，返回并确保你完成了安装那一节的步骤。cocos2D 框架是本章的重点内容。

在 Xcode 中创建一个新工程。确保你选择了 cocos2d_chipmunk 模板，如图 8-2 所示。

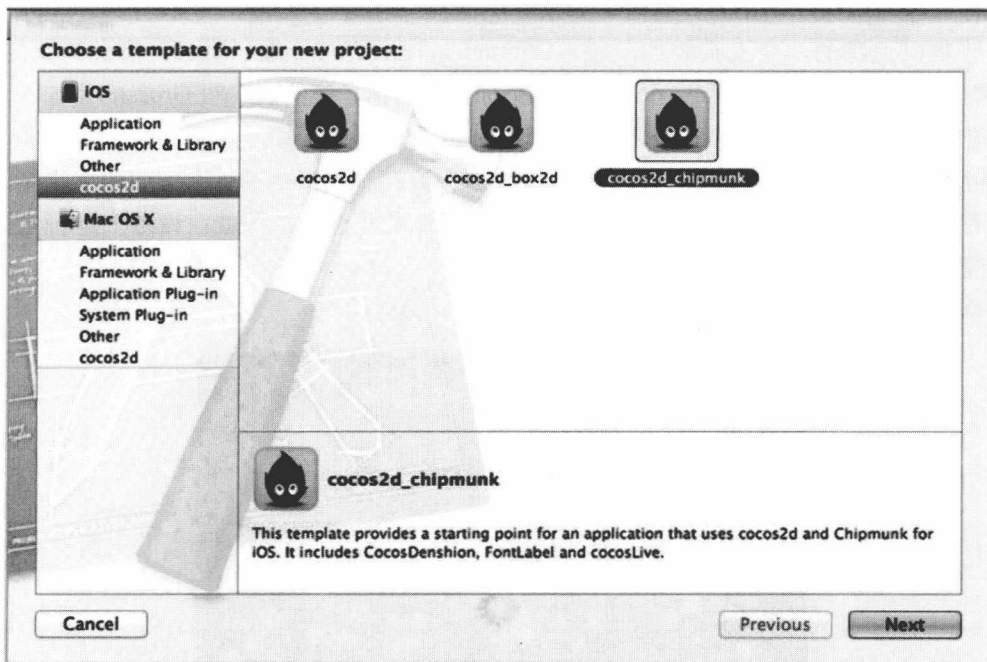


图 8-2 选择 cocos2d_chipmunk 模板

命名你的工程。我命名我的工程为 Ch8，因此你可以在 GitHub 仓库或者 Apress 网站（www.apress.com）的 Source Code/Download 区域找到。确保将工程设置为一个通用项目（Universal Project）。设置最小化所支持的设备方向（Supported Device Orientations）为只

支持右横向 (Landscape Right), 如图 8-3 所示。我们这样做是为了最小化南瓜的放置逻辑处理。

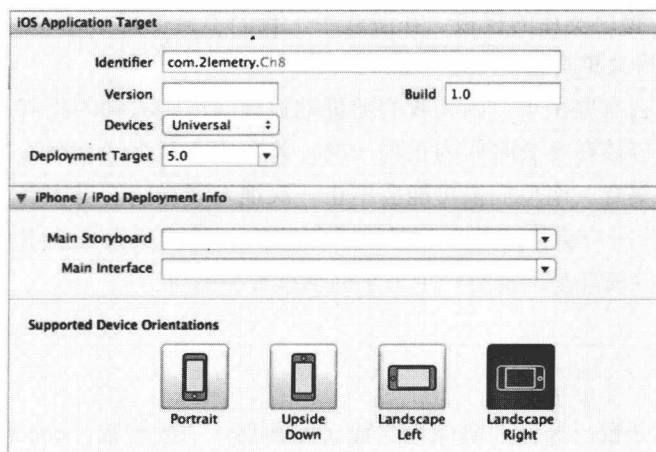


图 8-3 设置 Supported Device Orientations 为 Landscape Right

运行工程并确保一切正常。有时, 如果你一次对工程进行了很多改变, 那么 chipmunk 会引起一些难以追踪的、令人头痛的问题。Chipmunk 有一个与标准 cocos2D 模板不同的 Hello World 样式。你会在屏幕上看到一个男人。如果你愿意的话, 他的名字叫 Grossini。

现在, 这是一个非常酷的 Hello World 工程。如果你单击屏幕, 会出现一个新的 Grossini, 并使附近的其他 Grossini 不占据它的存在空间。这个应用也会对加速计和方向做出响应, 因此如果你有兴趣看一看 chipmunk 的这个简单例子会为你做些什么, 那就在设备上运行它, 而不仅仅是模拟器中运行。

在单击了屏幕多次之后, 我得到了类似图 8-4 的结果。

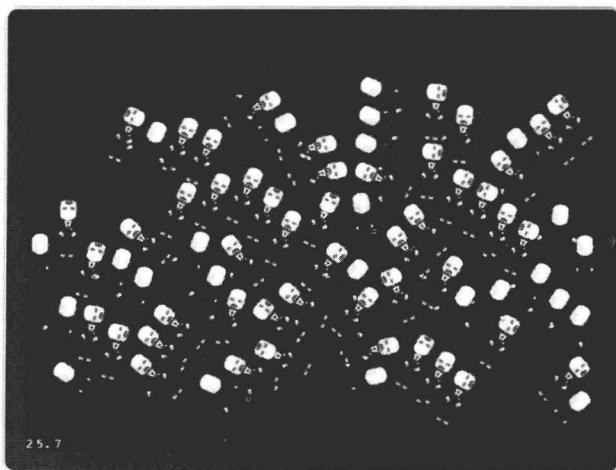


图 8-4 单击屏幕几次, 查看 Grossini 的增加

关闭应用并返回 Xcode。我们删除不用的文件，并从零开始建立这个工程。从工程中删除 HelloWorldLayer.h、HelloWorldLayer.m 和 grossini_dance_atlas.png 文件。

打开 AppDelegate.m 文件。删除与我们刚刚删除的类相关的导入语句。之后，注释掉代码清单 8-1 显示的行，该代码可以在 applicationDidFinishLaunching 方法的末尾找到。

代码清单 8-1 注释掉 HelloWorld 场景

```
[[[CCDirector sharedDirector] runWithScene: [HelloWorldLayer scene]]];
```

确保应用建立之后，你还不需要运行它，但此时应该可以无错编译。

摄像头视图

与第 7 章讲述的操作相似，我们首先设置摄像头视图为基础层。添加下面的框架到工程。

- CoreImage
- CoreMedia
- CoreVideo

在 Xcode 中打开 AppDelegate.h 文件。按代码清单 8-2 显示的代码更新头文件。

代码清单 8-2 新版 AppDelegate.h 文件

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>

@class RootViewController;

@interface AppDelegate : NSObject <UIApplicationDelegate,
AVCaptureVideoDataOutputSampleBufferDelegate> {
    UIWindow *window;
    RootViewController *viewController;

    AVCaptureSession *_session;
    UIView *_cameraView;
    UIImageView *_imageView;
}

@property (nonatomic, retain) UIWindow *window;

@end
```

与第 7 章的操作非常相似，我们建立了 UIView 和一个 UIImageView 来为用户处理和显示摄像头视图。切换到 AppDelegate.m 文件。添加代码清单 8-3 中的方法到实现。

代码清单 8-3 AppDelegate.m 的新方法

```
-(AVCaptureDevice *)frontFacingCameraIfAvailable
{
    NSArray *videoDevices = [AVCaptureDevice devicesWithMediaType:AVMediaTypeVideo];
    AVCaptureDevice *captureDevice = nil;
    for (AVCaptureDevice *device in videoDevices)
    {
        if (device.position == AVCaptureDevicePositionFront)
        {

```

```

        captureDevice = device;
        break;
    }
}

// couldn't find one on the front, so just get the default video device.
if (!captureDevice)
{
    captureDevice = [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];

    }

    return captureDevice;
}

- (void)setupCaptureSession {
    NSError *error = nil;

    _session = [[AVCaptureSession alloc] init];
    _session.preset = AVCaptureSessionPresetMedium;

    //AVCaptureDevice *device = [AVCaptureDevice
defaultDeviceWithMediaType:AVMediaTypeVideo];
    AVCaptureDevice *device = [self frontFacingCameraIfAvailable]; // for debug
    AVCaptureDeviceInput *input = [AVCaptureDeviceInput deviceInputWithDevice:device
error:&error];

    if (!input) {
        NSLog(@"Some kind of error... handle it here");
    }

    [_session addInput:input];

    AVCaptureVideoDataOutput *output = [[AVCaptureVideoDataOutput alloc] init];
    [_session addOutput:output];

    dispatch_queue_t queue = dispatch_queue_create("pumpkins", NULL);
    [output setSampleBufferDelegate:self queue:queue];
    dispatch_release(queue);

    output.videoSettings =
    [NSDictionary dictionaryWithObject:
    [NSNumber numberWithInt:kCVPixelFormatType_32BGRA]
    forKey:(id)kCVPixelBufferPixelFormatTypeKey];

    [_session startRunning];
}

```

第一个方法 `frontFacingCameraIfAvailable` 在这里的唯一目的是调试。如果你现在没有在人群中编码，这样做会非常方便，因此你可以使用前置摄像头来测试。当你不再调试的时候，应该去掉粗体行。意思是，要么注释掉它，要么强制使用前置摄像头（如果可用）。一次只能使用一种方式。因为我们没有人群中编码并设置使用前置摄像头。

在实际添加摄像头到视图之前，我们先用 `chipmunk` 物理引擎来建立一个菜单屏，以使等待游戏开始的用户感到愉悦。

8.3 创建游戏菜单

按图 8-5 显示的设置，用 `cocos2D` 的 `CCNode` 模板创建一个新文件。

确保文件是 CCLayer 的子类，名字是 MenuLayer.m。在 Xcode 中打开 MenuLayer.h。按代码清单 8-4 显示的代码更新头文件。

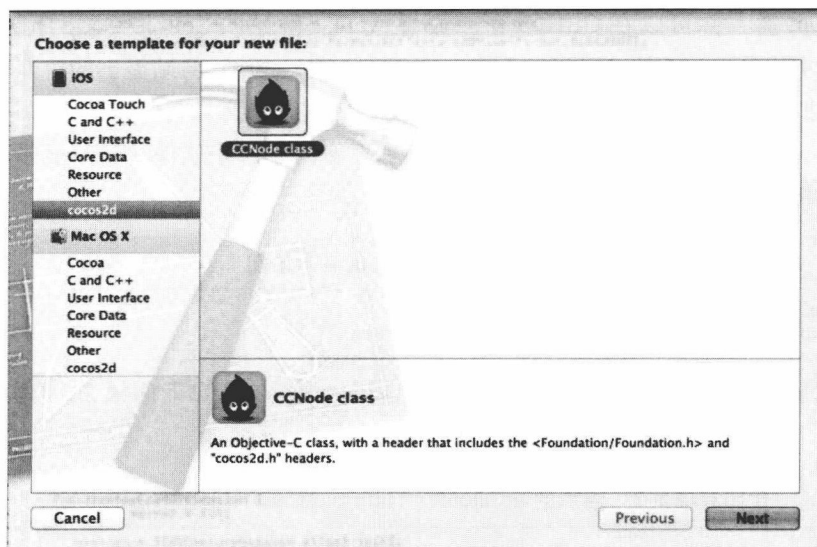


图 8-5 使用 CCNode 模板

代码清单 8-4 更新 MenuLayer.h 文件

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"

@interface MenuLayer : CCLayer {
}
+ (id)scene;
@end
```

切换到 MenuLayer.m 文件，添加代码清单 8-5 中的方法到实现。

代码清单 8-5 MenuLayer.m 的 scene 方法和 init 方法

```
+ (id)scene {
    CCScene *scene = [CCScene node];
    MenuLayer *layer = [MenuLayer node];
    [scene addChild:layer];
    return scene;
}

- (id)init {
    if ((self = [super init])) {
        CGSize winSize = [[CCDirector sharedDirector].winSize];

        CCLabelBMFont *label = [CCLabelBMFont labelWithString:@"Hello, Chipmunk!"
        fntFile:@"Arial.fnt"];
        label.position = ccp(winSize.width/2, winSize.height/2);
        [self addChild:label];
    }
    return self;
}
```

在这里，我们还没有做任何与游戏相关的事情。我们建立了一个 Hello World 类型的例子来确保一切工作正常。你会注意到，在这个例子中，我们使用了一个自定义字体文件和 CCLabelBMFont 类。你可以在 GitHub 仓库或者 Apress 网站的 Source Code/Download 区域找到源码。这些字体是 Ray Wenderlich (www.raywenderlich.com) 网站提供的。在试图运行工程之前，你需要把字体添加到工程。如果你想原样运行工程，那么你可以现在添加这些字体。

我们返回 AppDelegate.m 文件。我们需要添加更多的东西来确保将菜单显示给了用户。将代码清单 8-6 中的导入语句添加到工程。

代码清单 8-6 额外的导入语句

```
#import "chipmunk.h"
#import "MenuLayer.h"
```

在 AppDelegate.m 中的 applicationDidFinishLaunching 方法的下面添加代码清单 8-7 中的代码。

代码清单 8-7 添加到 applicationDidFinishLaunching 方法的底部

```
cpInitChipmunk();
[[CCDirector sharedDirector] runWithScene: [MenuLayer scene]];
```

在 iPad 模拟器中运行工程，如果一切正常，你会看到类似图 8-6 的结果。

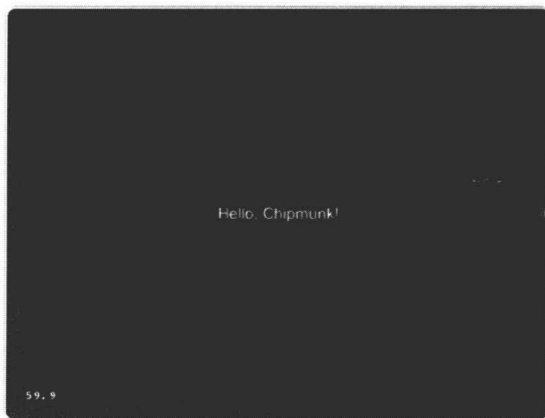


图 8-6 运行工程，看到“Hello, Chipmunk！”

我们让它变得更有趣一点。菜单层旨在当用户在决定是否开始一个新游戏时，我们可以给用户一些事情做。因为这是一个关于南瓜的游戏，因此我们会在屏幕上添加一个可以通过触摸交互而让用户来回抛掷的南瓜。

8.3.1 原图

对于 cocos2D 游戏，最好是使用精灵表。而且，如果你能够合并你的图像到一些精灵表中

以优化存储空间的话就更好了。因为本书是关于增强现实而非 iOS 游戏的，所以我已经为你创建并优化了精灵表，这些表可以在 GitHub 仓库的 Art 目录或者 Apress 网站 (www.apress.com) 的 Source Code/Download 区域找到。在继续学习之前，复制 Art 目录到你的 Xcode 工程。你应该拥有下面的 4 个新文件：

- particleTexture.png
- PumpkinExplosion.plist
- Pumpkins.plist
- Pumpkins.png

前两个文件是我为本工程创建的粒子发生器文件。当我们消灭南瓜的时候，它会使南瓜呈现橙色溶解的效果。第三、第四个文件是我们将会用到的包含不同南瓜的 PNG 文件。图 8-7 显示的是完整的精灵表。



图 8-7 工程要用的南瓜精灵表

精灵表中的南瓜被命名为 pumpkin#.png，其中 # 符号是介于 5 ~ 16 的数字。南瓜 1 到南瓜 4 没有面孔，所以我们将它们直接丢弃。

我们将会添加这些南瓜中的一个到菜单屏，以让用户抛掷。使用任意模板创建一个新文件，我们将重写所有代码，命名这个新类为 CPSprite。

用代码清单 8-8 中的代码重写头文件。

代码清单 8-8 CPSprite.h

```
#import "cocos2d.h"
#import "chipmunk.h"

typedef enum {
    kCollisionTypeGround = 0x1,
    kCollisionTypeCat,
    kCollisionTypeBed
} CollisionType;

@interface CPSprite : CCSprite {
    cpBody *body;
    cpShape *shape;
    cpSpace *space;
    BOOL canBeDestroyed;
}
```

```

@property (assign) cpBody *body;

- (void)update;
- (void)createBoxAtLocation:(CGPoint)location;
- (id)initWithSpace:(cpSpace *)theSpace location:(CGPoint)location
  spriteFrameName:(NSString *)spriteFrameName;
- (void)destroy;

@end

```

我们其实不需要太关心这个类具体做什么。我们实质上是建立了一个 CCSprite 类的封装，以使我们能够更容易地与屏幕上的对象交互。在例子中，这会帮助我们建立一个能够在屏幕上来回抛掷的南瓜。

切换到 CPSprite.m 文件，并用代码清单 8-9 中的代码替换实现代码。

代码清单 8-9 CPSprite.m

```

#import "CPSprite.h"

@implementation CPSprite
@synthesize body;

- (void)update {
    self.position = body->p;
    self.rotation = CC_RADIANS_TO_DEGREES(-1 * body->a);
}

- (void)createBoxAtLocation:(CGPoint)location {
    float mass = 1.0;
    body = cpBodyNew(mass, cpMomentForBox(mass, self.contentSize.width,
self.contentSize.height));
    body->p = location;
    body->data = self;
    cpSpaceAddBody(space, body);

    shape = cpBoxShapeNew(body, self.contentSize.width, self.contentSize.height);
    shape->e = 0.3;
    shape->u = 1.0;
    shape->data = self;
    shape->group = 1;
    cpSpaceAddShape(space, shape);
}

- (id)initWithSpace:(cpSpace *)theSpace location:(CGPoint)location
  spriteFrameName:(NSString *)spriteFrameName {
    if ((self = [super initWithSpriteFrameName:spriteFrameName])) {
        space = theSpace;
        canBeDestroyed = YES;
        [self createBoxAtLocation:location];
    }
    return self;
}

- (void)destroy {
    if (!canBeDestroyed) return;

    cpSpaceRemoveBody(space, body);
    cpSpaceRemoveShape(space, shape);
}

```

```

    [self removeFromParentAndCleanup:YES];
}

@end

```

实现文件建立了菜单中将会用到的属性。大部分情况下，你没有必要太担心这个文件中发生了什么。这个文件以及包含在 Helper Code 目录下的源代码中的两个文件，是用来让实现文件更容易阅读的功能方法。

8.3.2 辅助代码目录

在我们继续讲解之前，还有 4 个以上的文件需要从源目录复制。它们可以在 GitHub 仓库的 Helper Code 目录或者 Apress 网站（www.apress.com）的 Source Code/Download 区域获得。

这些文件是：

- cpMouse.c
- cpMouse.h
- drawSpace.h
- drawSpace.c

复制这些文件到你的 Xcode 工程。如果你想要建立工程，你会得到一个编译错误。在 Xcode 导航器中单击 cpMouse.c 文件。在右面的面板中打开 Identity and Type（标识和类型）视图并设置 File Type（文件类型）为 Objective-C source（源码），如图 8-8 所示。

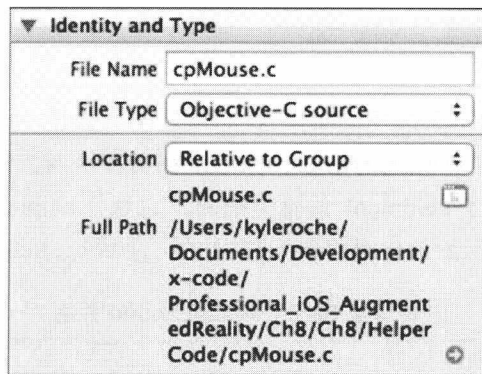


图 8-8 设置 cpMouse.c 的文件类型

为 drawSpace.c 文件重复这个过程。这时你的工程应该没有错误了。

8.3.3 完成菜单屏

返回 MenuLayer.h 文件。现在，在我们的工程中有一个辅助类了，事情会更顺利。按代码清单 8-10 中显示的代码更新头文件。

代码清单 8-10 更新 MenuLayer.h 文件

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"
#import "chipmunk.h"

@interface MenuLayer : CCLayer {
    cpSpace *space;
    cpBody *ground;
}

+ (id)scene;

@end
```

在导入 chipmunk 库以后，我们声明了两个新的私有变量。首先，我们为 chipmunk 建立了一个 cpSpace 类型的 iVar。这定义了 chipmunk 将会追踪的物理动作的空间。之后我们添加了一个 cpBody 到类中。它将作为一种类型的地板呈现在屏幕的底部。如果我们不这样做，南瓜会在任何方向上飞出屏幕。拥有一个地面也会帮助 chipmunk 更容易地获悉重力和摩擦力。

1. 用 chipmunk 工作

切换到 MenuLayer.m 文件。从我们的辅助代码类中导入 drawSpace.h 头文件。添加代码清单 8-11 中的方法到实现。

代码清单 8-11 createSpace 方法

```
- (void)createSpace {
    space = cpSpaceNew();
    space->gravity = ccp(0, -750);
    cpSpaceResizeStaticHash(space, 400, 200);
    cpSpaceResizeActiveHash(space, 200, 200);
}
```

这个方法定义了 chipmunk 的作用空间。我们设置了 X 轴的引力为 0（没有一侧到另一侧的引力），并向 Y 轴施加了一个相当的重量。你可以在实现中调节这个值，直到获得你想要的感觉。接下来两行代码优化了 chipmunk 的碰撞检测。它划分 chipmunk 为网格并告诉框架忽略那些不在同一个格内的对象。添加代码清单 8-12 中的方法到 createSpace 方法的下面。

代码清单 8-12 createGround 方法

```
- (void)createGround {

    CGSize winSize = [CCDirector sharedDirector].winSize;
    CGPoint lowerLeft = ccp(0, 0);
    CGPoint lowerRight = ccp(winSize.width, 0);

    ground = cpBodyNewStatic();

    float radius = 10.0;
    cpShape *groundShape = cpSegmentShapeNew(ground, lowerLeft, lowerRight, radius);

    groundShape->e = 0.5; // elasticity
    groundShape->u = 1.0; // friction

    cpSpaceAddShape(space, groundShape);
}
```

如前所述，我们正在创建一个地面效果以便南瓜不会完全脱离屏幕。我们还设立了摩擦力和弹力。如果你想要南瓜更有趣味性，你可以设置很低的弹力并设置摩擦力为 0，以获得类似冰上的效果。或者，你可以增减摩擦力和弹力来得到蹦床的效果。

2. 在 chipmunk 中创建对象

把代码清单 8-13 中的方法添加到实现。

代码清单 8-13 createBoxAtLocation 和 draw (Override) 方法

```
- (void)createBoxAtLocation:(CGPoint)location {
    float boxSize = 60.0;
    float mass = 1.0;
    cpBody *body = cpBodyNew(mass, cpMomentForBox(mass, boxSize, boxSize));
    body->p = location;
    cpSpaceAddBody(space, body);

    cpShape *shape = cpBoxShapeNew(body, boxSize, boxSize);
    shape->e = 1.0;
    shape->u = 1.0;
    cpSpaceAddShape(space, shape);
}

- (void)draw {
    drawSpaceOptions options = {
        0, // drawHash
        0, // drawBBs,
        1, // drawShapes
        4.0, // collisionPointSize
        4.0, // bodyPointSize,
        2.0 // lineThickness
    };

    drawSpace(space, &options);
}
```

第一个方法将会只用这一次。我们将会创建一些盒子并把它们放到屏幕上以确保 chipmunk 保持正确的行为。第二个方法用于开启 chipmunk 的调试绘图模式。没有这个方法，很难说你实际呈现到屏幕上的会是什么样子。如果你在游戏开发过程中使用，这个方法会非常方便。按代码清单 8-14 显示的代码更新 init 方法并添加一个叫做 update 的新方法。

代码清单 8-14 新的 init 和 update 方法

```
- (id)init {
    if ((self = [super init])) {
        CGSize winSize = [[CCDirector sharedDirector].winSize;

        [self createSpace];
        [self createGround];
        [self createBoxAtLocation:ccp(100,100)];
        [self createBoxAtLocation:ccp(200,200)];

        [self scheduleUpdate];
    }
    return self;
}
```

```
- (void)update:(ccTime)dt {  
    cpSpaceStep(space, dt);  
}
```

好的，事情变得更有意思了。有了这些新的对象和方法，我们将建立 chipmunk 空间的代码、地面体，并创建两个不同坐标的箱子。update 方法会逐步更新在 chipmunk 空间中自上一次迭代的变化。chipmunk 会为你处理所有其空间内的对象的更新。继续并使用模拟器运行工程你就明白我什么意思了。你应该会看到与图 8-9 类似的情景。



图 8-9 运行工程查看屏幕上的箱子

同样，没有什么太令人兴奋的，是吧？我们添加一个 mouse joint，以使我们能够抓住和抛出盒子。joint 是 chipmunk 中的一种约束。在这个例子中，我们将使用两种类型的 joint。我们将使用 mouse joint 来让用户抓住盒子，并且我们会使用 cpDampedSpring joint 来让南瓜在我们扔出它的时候弹回来。

切换到 MenuLayer.h 文件并从 Helper Code 目录导入 cpMouse.h 头文件。之后声明一个叫做 mouse 的 cpMouse 类的私有 iVar。

把下面代码清单 8-15 中的代码加入到 init 方法中，放在调用 scheduleUpdates 方法的前面。

代码清单 8-15 添加 init 方法

```
mouse = cpMouseNew(space);  
self.isTouchEnabled = YES;
```

第一行设置新的 mouse iVar 来处理 chipmunk 空间中的 mouse 事件。第二行为我们的类启动触摸事件。

启动触摸事件是一件事；处理它们是另一件更复杂的事。在 cocos2D 中有两种基本的处理触摸事件的方法。对于本例来说，我们只需要单点触摸事件（相对于多点触摸）。奇怪的是，这比多点触摸的处理需要更多的代码。添加代码清单 8-16 中的代码到实现。

代码清单 8-16 处理触摸事件

```

- (void)registerWithTouchDispatcher {
    [[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:0
    swallowsTouches:YES];
}

- (BOOL)ccTouchBegan:(UITouch *)touch withEvent:(UIEvent *)event {
    CGPoint touchLocation = [self convertTouchToNodeSpace:touch];
    cpMouseGrab(mouse, touchLocation, false);
    return YES;
}

- (void)ccTouchMoved:(UITouch *)touch withEvent:(UIEvent *)event {
    CGPoint touchLocation = [self convertTouchToNodeSpace:touch];
    cpMouseMove(mouse, touchLocation);
}

- (void)ccTouchEnded:(UITouch *)touch withEvent:(UIEvent *)event {
    cpMouseRelease(mouse);
}

- (void)ccTouchCancelled:(UITouch *)touch withEvent:(UIEvent *)event {
    cpMouseRelease(mouse);
}

- (void)dealloc {
    cpMouseFree(mouse);
    cpSpaceFree(space);
    [super dealloc];
}

```

这一系列的事件让 chipmunk 获悉用户想要抓住的与触摸坐标碰撞的对象。chipmunk 会更新对象的位置直到用户释放触摸事件。这时，重力、弹力，以及任何其他 chipmunk 环境变量就会发生作用，并且对象会返回到一个自然路径。在模拟器上再次运行工程，现在你能够在屏幕上来回扔这些对象。要注意，虽然你实际上能够把它们扔出屏幕，但你无法让它们返回！

因此，屏幕上的盒子并不完全是我们目的的一部分，对吧？为了让这些盒子变成南瓜，我们需要连接一个精灵图片到这个盒子，并确保当 chipmunk 更新位置的时候图像会跟随盒子。并且，由于盒子不完全如它代表的一样有用，因为用户能够把南瓜完全地抛出屏幕，所以我们将为南瓜在屏幕中添加一个 groove joint。把 groove joint 想象成一个旗杆或者固定轨道。与这种 joint 绑定的对象能够沿着路径移动，但是无法转向而远离杆或轨道。

在 Xcode 中返回到 MenuLayer.h 文件。按代码清单 8-17 中显示的代码更新头文件。

代码清单 8-17 更新 MenuLayer.h 文件

```

#import <Foundation/Foundation.h>
#import "cocos2d.h"
#import "chipmunk.h"
#import "cpMouse.h"
#import "CPSprite.h"

@interface MenuLayer : CCLayer {
    cpSpace *space;
    cpBody *ground;
    cpMouse *mouse;

    CCSpriteBatchNode *pumpkinBatchNode;
}

```

```

    CPSprite *menuPumpkin1;
}
+ (id)scene;
@end

```

首先，我们导入了之前创建的 CPSprite 类。这会帮助我们方便地添加一个南瓜到屏幕。接下来，我们声明了一个私有 CCSpriteBatchNode 变量，还有一个 CCSprite 变量。

注意：

当用 cocos2D 开发游戏的时候，最好使用 CCSpriteBatchNodes 和精灵表来节省内存。我们将需要使用尽可能多的内存，因为我们将会有一个很高的面部识别处理需求。

3. 在 chipmunk 中创建一个 spring joint

切换到 MenuLayer.m 文件，按代码清单 8-18 中显示的代码更新其 init 方法。

代码清单 8-18 更新 init 方法

```

- (id)init {
    if ((self = [super init])) {
        CGSize winSize = [CCDirector sharedDirector].winSize;

        [self createSpace];
        [self createGround];
        //[self createBoxAtLocation:ccp(100,100)];
        //[self createBoxAtLocation:ccp(200,200)];

        [self scheduleUpdate];

        mouse = cpMouseNew(space);
        self.isTouchEnabled = YES;

        [[CCSpriteFrameCache sharedSpriteFrameCache]
        addSpriteFramesWithFile:@"pumpkins.plist"];
        pumpkinBatchNode = [CCSpriteBatchNode batchNodeWithFile:@"pumpkins.png"];
        [self addChild:pumpkinBatchNode];

        menuPumpkin1 = [[[CPSprite alloc] initWithSpace:space location:ccp(347, 328)
        spriteFrameName:@"pumpkin15.png"] autorelease];
        [pumpkinBatchNode addChild:menuPumpkin1];

        cpConstraint *pumpkin1 = cpDampedSpringNew(menuPumpkin1.body, ground, ccp(-
        100,100), ccp(250,700), 100, 5, 1);
        cpSpaceAddConstraint(space, pumpkin1);
    }
    return self;
}

```

首先，我们用共享的精灵表创建了一个 CCSpriteFrameCache。之后，我们使用了创建的 PLIST 文件，因此 CCSpriteFrameCache 知道精灵表的内容。在动作层，我们将再次使用它。我们设置南瓜 PNG 文件的内容为 batch 节点。

接下来的两个语句更有趣。我们设置 CPSprite 类型的 menuPumpkin1 为一个带有一个位置和一个默认南瓜图片的新 CCSprite 实例。

最后，如前所述，我们使用 `cpDampedSpringNew` 宏来建立一个新的 `cpDampedSpring` 约束。如果你想的话，可以修改这里我设置的值。它们将会增加或者减少弹簧的强度，还有偏移值。要记得，我们开启了调试绘图模式，因此我们可以在运行中查看弹簧。

注意：

如果你无法在 iPad 上测试这个工程，则代码清单 8-18 中的 `x`, `y` 坐标需根据 iPhone 的屏幕大小调节一下。把它们值减去一半应该会成功。

依照代码清单 8-19 中的粗体代码修改 `update` 方法。

代码清单 8-19 更新 `update` 方法

```
- (void)update:(ccTime)dt {
    cpSpaceStep(space, dt);
    for (CPSprite *sprite in pumpkinBatchNode.children) {
        [sprite update];
    }
}
```

这是一个非常小的代码量，但是它有一个巨大的作用。试着分别在有和没有刚刚添加的代码的情况下运行工程。在添加了新代码的情况下，当加载游戏菜单的时候，我们会看到与图 8-10 相似的情景。



图 8-10 运行更新的 `update` 方法中的代码来显示正确的游戏菜单

注意一下加入的弹簧和围住南瓜的盒子。当被扯下来或者被在屏幕上来回抛的时候，精灵会与容器保持一致。现在，如果你删除刚刚添加的那个更新的代码块，你会得到类似图 8-11 的结果。

如你所看到的，我们的容器仍然按预期进行工作。但是精灵已经从屏幕消失。从这个练习中你应该学到的第一件事情是调试绘图模式很重要。如果我们没有开启调试绘图模式，那么试

图找出南瓜为什么会从屏幕上消失将是一个非常艰巨的任务。你应该学到的第二件事是当任何变化发生的时候你必须无一例外地更新场景。把细节留给 chipmunk 和 cocos2D 来处理，但是要确保你考虑了 update。

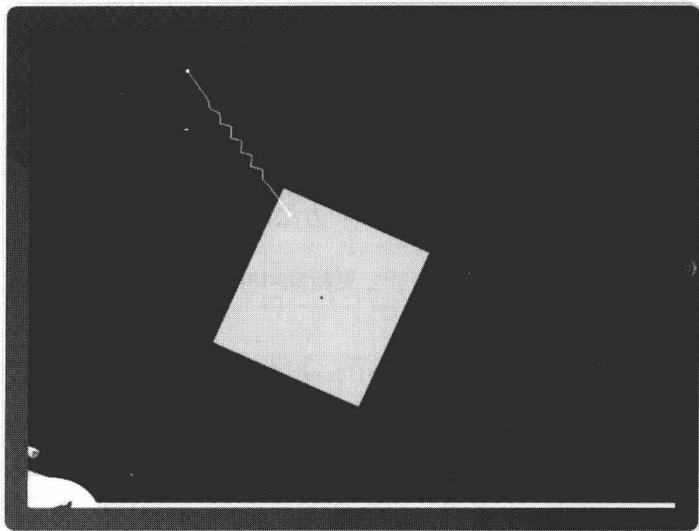


图 8-11 若没有 sprite 更新代码，你将会在屏幕上看到这个

8.4 添加菜单选项

这是一个简单的游戏，但是我们仍然需要一个游戏开始选项。在将它添加到 MenuLayer 之前，先创建一个基础类以便游戏开始时我们能知道它是否工作。

使用 CCNode 模板再次创建一个新文件。这次命名类为 ActionLayer。确保它是 CCLayer 的子类。在 Xcode 中打开 ActionLayer.h 文件并按代码清单 8-20 显示的代码更新它。

代码清单 8-20 更新 ActionLayer.h 文件

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"

@interface ActionLayer : CCLayer {
}

+ (CCScene *) scene;

@end
```

就像 MenuLayer 类，我们声明了一个静态的 CCScene 方法。对于 cocos2D 的 scene 类来说这是普遍的。这就是 CCDirector 用来显示一个新场景的方法。

切换到 ActionLayer.m 文件并按代码清单 8-21 显示的代码更新实现。

代码清单 8-21 更新 ActionLayer.m 文件

```

#import "ActionLayer.h"

@implementation ActionLayer

+(CCScene *) scene
{
    // 'scene' is an autorelease object.
    CCScene *scene = [CCScene node];

    // 'layer' is an autorelease object.
    ActionLayer *layer = [ActionLayer node];

    // add layer as a child to scene
    [scene addChild: layer];
    // return the scene
    return scene;
}

- (id)init {
    if ((self = [super init])) {
        CGSize winSize = [CCDirector sharedDirector].winSize;

        CCLabelBMFont *label = [CCLabelBMFont labelWithString:@"Game on pumpkins!"]
        fntFile:@"Arial.fnt"];
        label.position = ccp(winSize.width/2, winSize.height/2);
        [self addChild:label];
    }
    return self;
}

@end

```

后面我们将会替换这段代码的大部分。现在，它会用一个标签来显示关于南瓜的一小段消息，以便让我们知道用于开始游戏场景的所有事情都已就绪。

返回 MenuLayer.m 文件。导入新的 ActionLayer.h 类。找到用来向屏幕添加弹簧约束的代码。就在那段代码的下面，添加代码清单 8-22 中的代码。

代码清单 8-22 添加到 init 方法的底部，弹簧约束的后面

```

CCLabelBMFont *newGame;
if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad) {
    newGame = [CCLabelBMFont labelWithString:@"New Game" fntFile:@"Arial-
hd.fnt"];
} else {
    newGame = [CCLabelBMFont labelWithString:@"New Game" fntFile:@"Arial.fnt"];
}

CCMenuItemLabel *newGameItem = [CCMenuItemLabel itemWithLabel:newGame
target:self selector:@selector(newGameTapped:)];
newGameItem.position = ccp(winSize.width * 0.8, winSize.height * 0.3);

CCMenu *menu = [CCMenu menuWithItems:newGameItem, nil];
menu.position = CGPointZero;

[self addChild:menu];

```

这个代码块为用户创建了一个 cocos2D 菜单。我们正加载一个依赖于设备类型的字体，之后呈现给用户一个单独的用于开始一个新游戏的选项。如果 NewGame（新游戏）标签被单击

了，我们将会触发选择器（selector）newGameTapped 方法。这是一个我们还没有创建的方法。现在让我们创建它。添加代码清单 8-23 中的方法到实现。

代码清单 8-23 newGameTapped 方法

```
- (void)newGameTapped:(id)sender {  
    [[CCDirector sharedDirector] replaceScene:[CCTransitionRadialCCW  
transitionWithDuration:1.0 scene:[ActionLayer node]]];  
}
```

当用户单击新游戏选项的时候，我们会使用一个 cocos2D 过渡来以一个时髦的方式呈现新游戏。使用 iPad 模拟器运行工程。在启动的时候，你会看到一个新菜单，如图 8-12 所示。



图 8-12 运行工程以看到 New Game 菜单

如果你单击 New Game 按钮，你会看到一个快速的文字放大效果，之后会出现一个屏幕过渡到 ActionLayer，如图 8-13 所示。



图 8-13 单击 New Game 来过渡屏幕并“召唤”出南瓜

那么，添加游戏逻辑的几乎所有事情都准备好了。返回 AppDelegate 类并开启之前添加的摄像头支持。从现在开始往后，你将必须使用一个物理设备而不是模拟器来测试工程。

开启摄像头支持

下面将开启摄像头支持并以摄像头的视图代替我们目前正在显示的黑色背景。在 Xcode 中打开 AppDelegate.h 文件，并按代码清单 8-24 中显示的代码更新 AppDelegate.h 文件。

代码清单 8-24 更新 AppDelegate.h 文件

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import "ActionLayer.h"

@class RootViewController;

@interface AppDelegate : NSObject <UIApplicationDelegate,
AVCaptureVideoDataOutputSampleBufferDelegate> {
    UIWindow *window;
    RootViewController *viewController;

    AVCaptureSession *_session;
    UIView *_cameraView;
    UIImageView *imageView;
    BOOL _settingImage;
}
@property (nonatomic, retain) UIWindow *window;
@property (nonatomic) BOOL isPlaying;
@property (nonatomic, retain) ActionLayer *actionLayer;

@end
```

在导入了 ActionLayer.h 头文件之后，我们声明了两个新属性。首先，我们用一个布尔型的属性来追踪用户当前是否正在玩游戏。之后，我们为游戏保持了一个对 ActionLayer 的引用。这是用来确保面部识别请求只作用于活动的游戏层。如果你要在 App Store 上发布这个游戏，最好是重构它为另一个独立的类来管理状态。因此，使用我们正在采取的方法是恰当的。

最后，我们添加一个布尔值来跟踪我们是否正在使用摄像头缓冲来设置图像视图。

切换到 AppDelegate.m 文件。在 applicationDidFinishLaunching 方法中找到 EAGLView 的声明。按代码清单 8-25 显示的代码更新它。

代码清单 8-25 修改 OpenGL ES 视图的 pixelFormat 属性

```
EAGLView *glView = [EAGLView viewWithFrame:[window bounds]
                    pixelFormat:kEAGLColorFormatRGBA8
                    depthFormat:0];
```

这应该与第 7 章相似。找到调用 window 实例的 makeKeyAndVisible 函数的代码行。添加代码清单 8-26 中的代码到那个函数调用的上方。

代码清单 8-26 为摄像头设置背景为透明视图

```

[CCDirector sharedDirector].openGLView.backgroundColor = [UIColor clearColor];
[CCDirector sharedDirector].openGLView.opaque = NO;

glClearColor(0.0, 0.0, 0.0, 0.0);
_cameraView = [[UIView alloc] initWithFrame:[UIScreen mainScreen] bounds];
_cameraView.opaque = NO;
_cameraView.backgroundColor = [UIColor clearColor];

[window addSubview:_cameraView];

_imageView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen] bounds];
[_cameraView addSubview:imageView];
[window bringSubviewToFront:viewController.view];

```

同样，我们这段代码并不陌生。我们在第 7 章学习了这个方法。需要注意的地方是，iOS 5 引入了一个建立增强现实应用的新方法。事实上你可以设置 OpenGL ES 视图的背景为一个显示摄像头视图的纹理。但是，cocos2D 是建立在 OpenGL 之上的，仍然有一些版本冲突没有完全解决。至少在这本书编写时这是最好的方法。

在继续图像处理之前，有一些事情我们还没有做。确保你用 `synthesize` 语句合成了 `isPlaying` 属性。之后，添加代码清单 8-27 中的代码到 `applicationDidFinishLaunching` 方法中 `chipmunk` 类库初始化代码的后面。

代码清单 8-27 初始设置

```

cpInitChipmunk(); // should exist
[[CCDirector sharedDirector] runWithScene: [MenuLayer scene]]; // should exist
isplaying = NO;
[self setupCaptureSession];

```

`AppDelegate` 类已经指定为一个遵守 `AVCaptureVideoDataOutputSampleBufferDelegate` 协议的类。我们还没有实现任何的委托方法。现在，我们添加委托方法到 `AppDelegate` 的实现中。添加代码清单 8-28 中的代码。

代码清单 8-28 图像处理方法

```

- (UIImage *) imageFromSampleBuffer:(CMSampleBufferRef) sampleBuffer
{
    // Get a CMSampleBuffer's Core Video image buffer for the media data
    CVImageBufferRef imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
    // Lock the base address of the pixel buffer
    CVPixelBufferLockBaseAddress(imageBuffer, 0);

    // Get the number of bytes per row for the pixel buffer
    void *baseAddress = CVPixelBufferGetBaseAddress(imageBuffer);

    // Get the number of bytes per row for the pixel buffer
    size_t bytesPerRow = CVPixelBufferGetBytesPerRow(imageBuffer);
    // Get the pixel buffer width and height
    size_t width = CVPixelBufferGetWidth(imageBuffer);
    size_t height = CVPixelBufferGetHeight(imageBuffer);

    // Create a device-dependent RGB color space
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();

```

```

// Create a bitmap graphics context with the sample buffer data
CGContextRef context = CGBitmapContextCreate(baseAddress, width, height, 8,
                                             bytesPerRow, colorSpace,
                                             kCGBitmapByteOrder32Little | kCGImageAlphaPremultipliedFirst);
// Create a Quartz image from the pixel data in the bitmap graphics context
CGImageRef quartzImage = CGBitmapContextCreateImage(context);
// Unlock the pixel buffer
CVPixelBufferUnlockBaseAddress(imageBuffer, 0);

// Free up the context and color space
CGContextRelease(context);
CGColorSpaceRelease(colorSpace);

// Create an image object from the Quartz image
UIImage *image = [UIImage imageWithCGImage:quartzImage];

// Release the Quartz image
CGImageRelease(quartzImage);

return (image);
}

- (void)captureOutput:(AVCaptureOutput *)captureOutput
didOutputSampleBuffer:(CMSampleBufferRef)sampleBuffer
fromConnection:(AVCaptureConnection *)connection
{
    UIImage *image = [self imageFromSampleBuffer:sampleBuffer];

    if(_settingImage == NO){
        _settingImage = YES;
        [NSThread detachNewThreadSelector:@selector(setImageToView:) toTarget:self
        withObject:image];
    }
}

- (void)setImageToView:(UIImage*)image {
    //UIImage * capturedImage = [self rotateImage:image
    orientation:UIImageOrientationRight ];
    UIImage * capturedImage = [self rotateImage:image
    orientation:UIImageOrientationLeftMirrored ];
    _imageView.image = capturedImage;
    _settingImage = NO;

    if (isPlaying) {
        //NSLog(@"Playing. Send to Pumpkin layer");
        if (!actionLayer.isProcessingRequest) {
            UIImage * image = [_imageView.image retain];
            if(image!=nil){
                UIInterfaceOrientation orient = [UIApplication
                sharedApplication].statusBarOrientation;
                UIImage * rotatedImage = image;
                switch (orient) {
                    case UIInterfaceOrientationPortrait:
                        NSLog(@"Device orientation portrait");
                        rotatedImage = [self rotateImage:image orientation:
                        UIImageOrientationRight];
                        break;
                    case UIInterfaceOrientationPortraitUpsideDown:
                        rotatedImage = [self rotateImage:image orientation:
                        UIImageOrientationLeft];
                        NSLog(@"Device orientation portrait upside down");
                        break;
                    case UIInterfaceOrientationLandscapeLeft:
                        rotatedImage = [self rotateImage:image orientation:
                        UIImageOrientationRight];
                        NSLog(@"Device orientation landscape left");
                        break;
                    case UIInterfaceOrientationLandscapeRight:

```

```

        rotatedImage = [self rotateImage:image orientation:
UIImageOrientationLeft];
        NSLog(@"Device orientation landscape right");
        break;
    };
    [actionLayer facialRecognitionRequest:rotatedImage];
}
}
}

- (UIImage *) rotateImage:(UIImage*)image orientation:(UIImageOrientation) orient {
    CGImageRef imgRef = image.CGImage;
    CGAffineTransform transform = CGAffineTransformIdentity;
    //UIImageOrientation orient = image.imageOrientation;
    CGFloat scaleRatio = 1;
    CGFloat width = image.size.width;
    CGFloat height = image.size.height;
    CGSize imageSize = image.size;
    CGRect bounds = CGRectMake(0, 0, width, height);
    CGFloat boundHeight;

    switch(orient) {
        case UIImageOrientationUp:
            transform = CGAffineTransformIdentity;
            break;
        case UIImageOrientationUpMirrored:
            transform = CGAffineTransformMakeTranslation(imageSize.width, 0.0);
            transform = CGAffineTransformScale(transform, -1.0, 1.0);
            break;
        case UIImageOrientationDown:
            transform = CGAffineTransformMakeTranslation(imageSize.width,
imageSize.height);
            transform = CGAffineTransformRotate(transform, M_PI);
            break;
        case UIImageOrientationDownMirrored:
            transform = CGAffineTransformMakeTranslation(0.0, imageSize.height);
            transform = CGAffineTransformScale(transform, 1.0, -1.0);
            break;
        case UIImageOrientationLeftMirrored:
            boundHeight = bounds.size.height;
            bounds.size.height = bounds.size.width;
            bounds.size.width = boundHeight;
            transform = CGAffineTransformMakeTranslation(imageSize.height,
imageSize.height);
            transform = CGAffineTransformScale(transform, -1.0, 1.0);
            transform = CGAffineTransformRotate(transform, 3.0 * M_PI / 2.0);
            break;
        case UIImageOrientationLeft:
            boundHeight = bounds.size.height;
            bounds.size.height = bounds.size.width;
            bounds.size.width = boundHeight;
            transform = CGAffineTransformMakeTranslation(0.0, imageSize.width);
            transform = CGAffineTransformRotate(transform, 3.0 * M_PI / 2.0);
            break;
        case UIImageOrientationRightMirrored:
            boundHeight = bounds.size.height;
            bounds.size.height = bounds.size.width;
            bounds.size.width = boundHeight;
            transform = CGAffineTransformMakeScale(-1.0, 1.0);
            transform = CGAffineTransformRotate(transform, M_PI / 2.0);
            break;
        case UIImageOrientationRight:
            boundHeight = bounds.size.height;
            bounds.size.height = bounds.size.width;
            bounds.size.width = boundHeight;
            transform = CGAffineTransformMakeTranslation(imageSize.height, 0.0);
            transform = CGAffineTransformRotate(transform, M_PI / 2.0);
    }
}

```



```

        break;
    default:
        [NSException raise:NSInternalInconsistencyException format:@"Invalid image
orientation"];
    }
    UIGraphicsBeginImageContext(bounds.size);
    CGContextRef context = UIGraphicsGetCurrentContext();
    if (orient == UIImageOrientationRight || orient == UIImageOrientationLeft) {
        CGContextScaleCTM(context, -scaleRatio, scaleRatio);
        CGContextTranslateCTM(context, -height, 0);
    } else {
        CGContextScaleCTM(context, scaleRatio, -scaleRatio);
        CGContextTranslateCTM(context, 0, -height);
    }
    CGContextConcatCTM(context, transform);
    CGContextDrawImage(UIGraphicsGetCurrentContext(), CGRectMake(0, 0, width, height),
imgRef);
    UIImage *imageCopy = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();

    return imageCopy;
}

```

这是我们添加的最大的单个代码块。它的大部分代码是我们在第7章看到过的。但是，在怎么设置图像到屏幕视图方面有一些差异。当我们设置图像到屏幕的时候，在发送一个新图像请求给动作层之前，我们也会检查动作层是否繁忙。我们还没有在动作层实现这个功能。这就是接下来要做的。

在 AppDelegate.m 文件中按代码清单 8-29 显示的代码更新这个私有方法的声明。

代码清单 8-29 更新私有方法的声明

```

@interface AppDelegate (Private)
- (void)setupCaptureSession;
- (UIImage *) rotateImage:(UIImage*)image orientation:(UIImageOrientation) orient;
@end

```

在 AppDelegate 类中将会有两个编译错误，是由于在动作层缺少属性和方法引起的。切换到 ActionLayer.h 文件并添加代码清单 8-30 中的代码到头文件。

代码清单 8-30 添加到 ActionLayer.h 文件

```

@property (nonatomic) BOOL isProcessingRequest;
- (void)facialRecognitionRequest:(UIImage *)image;

```

我们的 AppDelegate 类缺少了两个元素。切换到 ActionLayer.m 文件。导入 AppDelegate.h 并合成名为 isProcessingRequest 的布尔属性。按代码清单 8-31 中显示的代码更新 init 方法。

代码清单 8-31 更新 init 方法

```

- (id)init {
    if ((self = [super init])) {
        isProcessingRequest = NO;
        [AppDelegate instance].actionLayer = self;
        [AppDelegate instance].isPlaying = YES;
    }
    return self;
}

```

我们移除了创建南瓜消息的代码，并添加了一些默认变量。你会注意到 AppDelegate 类的这个新实例方法显示了一个警告。我们还没有实现这个静态方法，因此现在可以忽略这个警告。如果你试图编译工程，你也会由于找不到接下来将会实现的静态实例方法而得到一个错误，按代码清单 8-32 显示的内容实现 facialRecognitionRequest 方法。

代码清单 8-32 方法 facialRecognitionRequest

```
- (void)facialRecognitionRequest:(UIImage *)image {  
    NSLog(@"Image is: %f by %f", image.size.width, image.size.height);  
}
```

现在我们只需要记录事件的一些详细信息以确保我们正接收一个合适的图像。在 Xcode 中打开 AppDelegate.h 文件并按代码清单 8-33 显示的代码声明静态方法。

代码清单 8-33 AppDelegate.h 的静态实例方法

```
+ (AppDelegate *)instance;
```

这个方法对于本工程的目的来说并不是很重要。它更多的是我的一个习惯。它提供了一个获得代表 AppDelegate 的单例的简单方法。

代码清单 8-34 AppDelegate.m 的静态实例方法

```
+ (AppDelegate *)instance {  
    return (AppDelegate *)[[UIApplication sharedApplication] delegate];  
}
```

当你在编辑 AppDelegate.m 文件时，确保你合成了我们之前创建的 actionLayer 层。在一个物理设备上运行工程（最好是 iPad 2）。你会看到一个欢迎菜单，如图 8-14 所示。



图 8-14 再次运行工程以查看一个带有摄像头背景的游戏菜单

如果你单击 New Game（新游戏）菜单，你可以通过调试控制台来观察更新。你会看到出

现了很多消息，如图 8-15 所示。

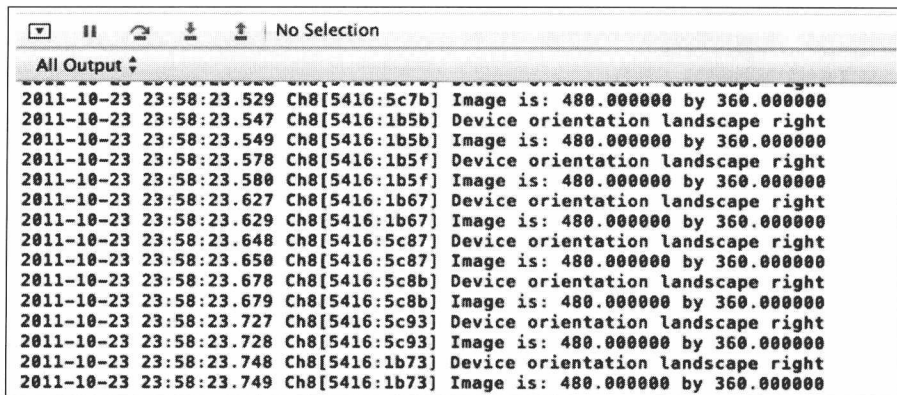


图 8-15 观察记录消息的屏幕缓存

你可以看到我们对动作层的调用是非常频繁的。事实上，这是一个需要改进的地方。放慢对动作层的调用过程可以帮助我们优化游戏的性能。

8.5 完成动作层

我们想想还需要做什么来让这个游戏有一些结构。很明显，我们缺少南瓜。除此之外，我们应该实现记录有多少外星南瓜被消灭了的计数器。在精灵表中有 12 种不同的南瓜图像。让我们把这个游戏设置成为一个倒计时游戏，也就是说在游戏结束以前用户必须消灭完 12 种南瓜。

首先要解决面部识别问题，我们将使用一个叫做 CIDetector 的 iOS 5 新类。第 12 章将详细讨论它。在 ActionLayer.h 文件中添加代码清单 8-35 中的属性。

代码清单 8-35 CIDetector 属性

```
@property (nonatomic, retain) CIDetector *detector;
```

打开 ActionLayer.m 文件并合成属性。在 init 方法的 if 代码块中添加代码清单 8-36 中的代码。

代码清单 8-36 建立 CIDetector

```
NSDictionary *detectorOptions = [NSDictionary
dictionaryWithObjectsAndKeys:CIDetectorAccuracyLow, CIDetectorAccuracy, nil];
self.detector = [CIDetector detectorOfType:CIDetectorTypeFace context:nil
options:detectorOptions];
```

如我之前提到的，我们将会在第 12 章详细讨论这个类。现在，你只需要知道 CIDetector 类可以追踪从图像中识别出来的面孔的 x, y 坐标。在例子中，我们将使用它来追踪摄像头视图中的面孔。按代码清单 8-37 中显示的代码更新 facialRecognitionRequest 方法。

代码清单 8-37 更新 facialRecognitionRequest 方法

```

- (void)facialRecognitionRequest:(UIImage *)image {
    if (!isProcessingRequest) {
        isProcessingRequest = YES;
        NSArray *arr = [detector featuresInImage:[CIImage imageWithCGImage:[image
CGImage]]];
        if ([arr count] > 0) {
            NSLog(@"Faces found.");
        } else {
            NSLog(@"No faces found");
        }
    }
    isProcessingRequest = NO;
}

```

我说过我们会在后面讨论，但是处理一个图像的面部识别所需要的这一小段代码实在令人印象深刻。它真的只需要粗体的那一行。如果在摄像头视图中发现一个面孔，我们会把它记录到控制台窗口中；如果没有发现，也会记录下来。

继续在一个物理设备上再次运行应用。你会在控制台窗口看到与图 8-16 类似的输出。

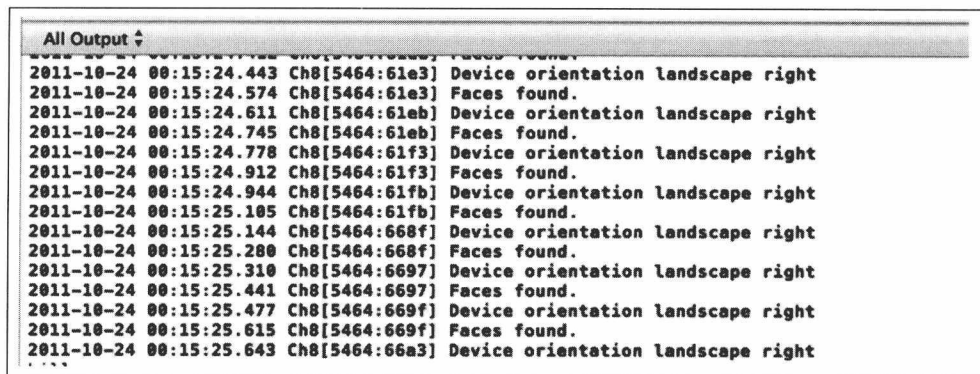


图 8-16 查看记录消息的屏幕缓冲，并注意“Faces found”消息

看起来进展顺利。我们添加一些用来在目标面孔上显示南瓜的逻辑。

8.6 南瓜来了

在任何给定的时刻，我们将只允许一个外星南瓜出现。基本上，我们找到的第一个面孔会是一个受南瓜控制的不幸受害者。

为了实现这一点，我们将会添加一个南瓜到屏幕并让它不可见。当我们检测到一个面孔的时候，我们会移动南瓜到面孔所在的坐标并让南瓜恢复不透明以便南瓜在摄像头视图上显示出来。

打开 ActionLayer.h 文件。声明代码清单 8-38 显示的私有变量。

代码清单 8-38 新的私有变量

```
CCSpriteBatchNode *pumpkinBatchNode;
CCSprite *pumpkin;
int pumpkin_count;
```

这些变量是用来建立我们最初的南瓜图像的。就像以前那样，我们将会使用 CCSpriteBatchNode 来保存我们的图像。Pumpkin_count（整型）将用于从 12 开始倒计时，以便我们知道还剩下多少外星南瓜没有被消灭。切换到 ActionLayer.m 文件。按代码清单 8-39 显示的代码更新 init 方法。

代码清单 8-39 新版 init 方法

```
- (id)init {
    if (self = [super init]) {
        //CGSize winSize = [CCDirector sharedDirector].winSize;

        NSDictionary *detectorOptions = [NSDictionary
dictionaryWithObjectsAndKeys:CIDetectorAccuracyLow, CIDetectorAccuracy, nil];
        self.detector = [CIDetector detectorOfType:CIDetectorTypeFace context:nil
options:detectorOptions];

        pumpkinBatchNode = [CCSpriteBatchNode batchNodeWithFile:@"pumpkins.png"];
        [self addChild:pumpkinBatchNode];

        pumpkin = [CCSprite spriteWithSpriteFrameName:@"pumpkin5.png"];
        pumpkin.position = ccp(0,0);
        pumpkin.opacity = 0;
        [self addChild:pumpkin];

        // Start the game
        isProcessingRequest = NO;
        pumpkin_count = 12;
        [AppDelegate instance].actionLayer = self;
        [AppDelegate instance].isPlaying = YES;

        self.isTouchEnabled = YES;
    }
    return self;
}
```

这里我们用之前使用过的相同的精灵表建立了 CCSpriteBatchNode。之后我们使用精灵表中的 pumpkin5.png 框架初始化南瓜变量为一个新的 CCSprite。我们接下来设置它为透明并把它添加到场景。

确保你的工程仍然可以无错编译。没有什么需要查看的，因此没有必要在设备上运行它。按代码清单 8-40 显示的代码更新 facialRecognitionRequest 方法。

代码清单 8-40 新版 facialRecognitionRequest 方法

```
- (void)facialRecognitionRequest:(UIImage *)image {
    //NSLog(@"Image is: %f by %f", image.size.width, image.size.height);

    if (!isProcessingRequest) {
        isProcessingRequest = YES;
        //NSLog(@"Detecting Faces");
        NSArray *arr = [detector featuresInImage:[CIImage imageWithCGImage:[image
CGImage]]];
```

```

if ([arr count] > 0) {
    NSLog(@"Faces found.");
    for (int i = 0; i < 1; i++) { //< [arr count]; i++) {
        CIFaceFeature *feature = [arr objectAtIndex:i];
        double xPosition = (feature.leftEyePosition.x +
        feature.rightEyePosition.x+feature.mouthPosition.x)/(3*image.size.width) ;
        double yPosition = (feature.leftEyePosition.y +
        feature.rightEyePosition.y+feature.mouthPosition.y)/(3*image.size.height);

        double dist = sqrt(pow((feature.leftEyePosition.x -
        feature.rightEyePosition.x),2)+pow((feature.leftEyePosition.y -
        feature.rightEyePosition.y),2))/image.size.width;

        yPosition += dist;
        CGSize size = [[CCDirector sharedDirector] winSize];
        pumpkin.opacity = 255;
        pumpkin.scale = 5*(size.width*dist)/256.0;

        [pumpkin setDisplayFrame:[CCSpriteFrameCache sharedSpriteFrameCache]
        spriteFrameByName:[NSString stringWithFormat:@"pumpkin%d.png", pumpkin_count + 4]]];
        CCMoveTo *moveAction = [CCMoveTo actionWithDuration:0
        position:ccp((size.width * (xPosition)), (size.height * ((yPosition))))];
        [pumpkin runAction:moveAction];
    }
    } else {
        pumpkin.opacity = 0;
    }
}
isProcessingRequest = NO;
}

```

这是整个游戏中最重要的代码块，因此我们逐步讲解它。首先，检查是否有一个当前请求正在进行中。如果没有，我们就运行 `CIDetector` 方法以在摄像头图像中寻找面孔。如果发现至少一个面孔，我们就继续前进。

这时我实际上硬编码了一个不会超过一次迭代的循环。如果你想要扩展这个游戏，这将是另一个可以改进的地方。在这个循环中实现了一个随机发生器，选择一个随机的南瓜附加到摄像头视图中的一个随机的目标上。追踪哪一个南瓜会落在哪个用户的头上会产生相当数量的处理器消耗，但这会让游戏体验更好。

接下来，我们让 `CIDetector` 发现的眼睛的 `x` 和 `y` 坐标增至 3 倍，以便更好地把南瓜放置到屏幕真实图像的上部，而不是我们处理中的按比例缩小的图像。

我们使用古老、优秀的勾股定理来计算距离并把它除以屏幕的宽度。在你远离屏幕的时候，这会帮助我们缩放图像以适合你的面孔的大小。当运行这个应用的时候你可以自己测试一下。南瓜将会在屏幕上跟随你并自适应缩放大小。

最后，我们设置不透明度为 255（以使它可见）并移动精灵到发现面孔的位置。我们使用 `cocos2D` 的两个宏来简化这个步骤。我们首先设置精灵框架为 `pumpkin_count` 指定的图像（稍后讲解），之后使用 `CCMoveTo` 来移动精灵到面孔的 `x`, `y` 坐标。

在一个物理设备上运行这个应用。通过单击菜单选项中的 `New Game` 来开始游戏。你会看

到与图 8-17 相似的情景。



图 8-17 老兄，我是一个南瓜！

我不确定我是否喜欢成为一个南瓜。之前，我们为 ActionLayer 类开启了触摸，但是一直没有实现响应方法。在 ActionLayer.h 文件中声明一个新的叫做 emitter 的 CCParticleSystemQuad 类型的私有变量。打开 ActionLayer.m 文件并添加代码清单 8-41 中的方法。

代码清单 8-41 触摸方法

```
- (void)registerWithTouchDispatcher {
    [[CCTouchDispatcher sharedDispatcher] addTargetedDelegate:self priority:0
    swallowsTouches:YES];
}

- (BOOL)ccTouchBegan:(UITouch *)touch withEvent:(UIEvent *)event {
    return YES;
}

- (void)ccTouchEnded:(UITouch *)touch withEvent:(UIEvent *)event {
    CGPoint location = [self convertTouchToNodeSpace:touch];
    if (CGRectContainsPoint(pumpkin.boundingBox, location)) {
        pumpkin_count--;

        emitter = [CCParticleSystemQuad particleWithFile:@"PumpkinExplosion.plist"];
        emitter.position = ccp(location.x, location.y);
        [self addChild:emitter z:1];

        if (pumpkin_count == 0) {
            NSLog(@"You won");
        }
    }
    //NSLog(@"Touch %@: ", NSStringFromCGPoint(location));
}
```

在本章之前，我们从例子工程中导入了 Art 目录。我们还用 Particle Designer (www.71squared.com) 创建了一个南瓜被蒸发的粒子效果。

当一个南瓜蒸发的时候，我们想要减少所剩南瓜的数量并更改南瓜的图像（因此在游戏过

程中，每种南瓜都会看到一次）。

我们添加的代码清单 8-41 中的代码首先检查触摸的位置。我们使用 `convertTouchToNodeSpace` 宏来得到触摸事件的屏幕坐标。在一些 cocos2D 游戏中，“游戏世界”会超过屏幕的边界（例如类似塞尔达传说的 tile 地图游戏），因此我们需要转换游戏世界的坐标为设备屏幕的坐标。接下来，我们检查触摸位置是否在南瓜边界框之内。如果是的话，我们就建立用来添加蒸发效果的粒子发生器，同时减小了 `pumpkin_count`，并且刷新了场景。如果你消灭了足够的南瓜，控制台窗口将会显示 “You Won” 的消息。蒸发南瓜的效果如图 8-18 所示。

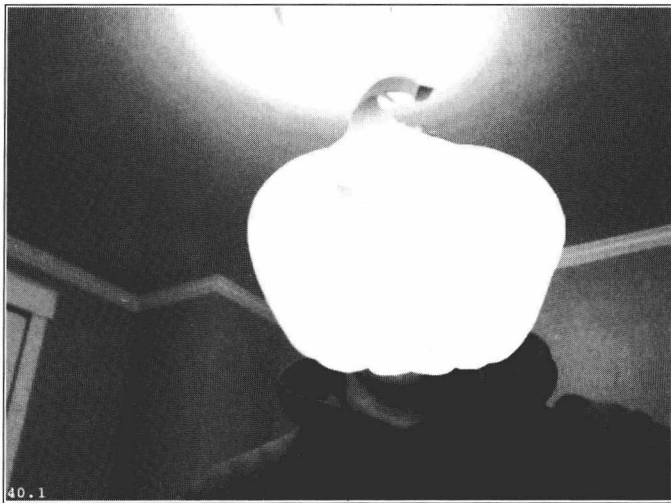


图 8-18 现在，我是一个蒸发的南瓜

8.7 结束游戏

永远不会结束的游戏并不好玩。当蒸发了一个南瓜后，我们减少了南瓜计数变量的值。我们添加一个用来处理游戏结束状态和重新开始（如果用户有兴趣的话）的场景到游戏中。

创建一个叫做 `EndLayer` 的新类。确保它使用 `CCNode` 模板，并且是一个 `CCLayer` 的子类。在 Xcode 中打开 `EndLayer.h` 文件并按代码清单 8-42 中显示的代码更新头文件。

代码清单 8-42 更新 `EndLayer.h` 文件

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"

@interface EndLayer : CCLayer {
}

+ (CCScene *) scene;

@end
```


看起来很熟悉，不是吗？我们又一次添加静态 CCScene 方法到新层中。切换到 EndLayer.m 文件。以导入 ActionLayer.h 文件开始。之后添加代码清单 8-43 中的方法到实现。

代码清单 8-43 EndLayer.m 文件的新方法

```
+(CCScene *) scene
{
    // 'scene' is an autorelease object.
    CCScene *scene = [CCScene node];

    // 'layer' is an autorelease object.
    EndLayer *layer = [EndLayer node];

    // add layer as a child to scene
    [scene addChild: layer];

    // return the scene
    return scene;
}

- (id)init {
    if ((self = [super init])) {
        CGSize winSize = [CCDirector sharedDirector].winSize;

        CCLabelBMFont *titleLabel;
        if (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad) {
            titleLabel = [CCLabelBMFont labelWithString:@"New Game" fntFile:@"Arial-
hd.fnt"];
        } else {
            titleLabel = [CCLabelBMFont labelWithString:@"New Game"
fntFile:@"Arial.fnt"];
        }

        CCMenuItemLabel *newGameItem = [CCMenuItemLabel itemWithLabel:titleLabel
target:self selector:@selector(playTapped:)];
        newGameItem.position = ccp(winSize.width * 0.8, winSize.height * 0.3);

        CCMenu *menu = [CCMenu menuWithItems:newGameItem, nil];
        menu.position = CGPointZero;

        [self addChild:menu];
    }
    return self;
}

- (void)playTapped:(id)sender {
    [[CCDirector sharedDirector] replaceScene:[CCTransitionRadialCCW
transitionWithDuration:1.0 scene:[ActionLayer scene]]];
}
```

scene 和 init 方法应该没有什么令人吃惊的地方。在 init 方法中，我们建立了另一个允许用户结束游戏的游戏菜单。在显示 EndLayer 场景之前，我们需要清除一些 ActionLayer 层的变量。返回 ActionLayer.m 文件。首先导入 EndLayer.h 头文件。

找到代码清单 8-44 中显示的代码。用代码清单 8-45 中的代码替换它。

代码清单 8-44 找到这个代码

```
if (pumpkin_count == 0) {  
    NSLog(@"You won");  
}
```

代码清单 8-45 用这个替换

```
if (pumpkin_count == 0) {  
    NSLog(@"You won");  
    isProcessingRequest = YES;  
    [AppDelegate instance].isPlaying = NO;  
    [[CCDirector sharedDirector] replaceScene:[EndLayer scene]];  
}
```

当 ActionLayer 类初始化了以后，我们设置 isProcessingRequest 为 YES。我们还设置 AppDelegate 类的 isPlaying 布尔属性为 NO。在我们能够结束游戏并允许用户重新开始之前，我们必须重置游戏环境。做了这些之后，我们通知 CCDirector 运行 EndLayer 场景。

在你蒸发了 12 个南瓜之后，会呈现一个重新开始菜单，如图 8-19 所示。



图 8-19 所有南瓜蒸发了，一个新游戏菜单出现了

如果你选中 New Game，游戏会以默认状态重新开始（12 个南瓜需要被消灭）。

注意：

在你把这个游戏带到一个派对之前，你可能想要把前置摄像头转成默认视频设备。此外，请确保你把旋转效果修改回来了。CIDetector 对于非垂直图像的处理有一些问题。同时，你可以从 MenuLayer 里面移除调试绘图方法来让菜单的编码装置更少。

8.8 总结

这是非常有趣的一章。我们以默认的 cocos2D 的 chipmunk 模板开始，建立了一个完整的

游戏。我们首先要让菜单更有趣。我们取了一个南瓜并把它固定到一个弹簧约束上面，以便我们可以随便地使用，而它不会飞出屏幕。

我们建立了游戏菜单来启动动作层并开始面部识别请求。我们学习了精灵表和批量节点，以及一些类似 CCMoveTo 和设置精灵透明度的处理覆盖的简单方法。

我们介绍了将会在第 12 章学习的 CIDetector 类来识别面孔，以便可以移动南瓜到正确的位置上。我们以一些蒸发外星南瓜的粒子效果结束，并建立了一个游戏结束菜单。

如我们在上两章看到的，比起使用 OpenGL ES 从零开始建立这些功能，使用 cocos2D 框架会让游戏开发更容易。

在第 9 章，我们会介绍一些其他的增强现实工程可用的第三方框架和软件开发套件 (Software Development Kit, SDK)。这些框架中的大部分都专注于基于位置或者基于标记的增强现实。第 13 章将会看到更多的游戏特性。

第⑨章

第三方增强现实工具包

在本书中，我们将会讲解一些可以对你的增强现实开发有帮助的技术。关键是要把应用快速推向市场而不是重新发明。虽然增强现实是一个新的领域，但这一点仍然是正确的。对于开发者来说，可以使用一些工具包来开发基于位置的，基于标记的甚至是基于 3D 绘图的应用。

在本章中，我们会讨论其中的一些工具以及它们的长处和短处。与此同时，我们也会建立一些例子应用。

9.1 概述

在本章中我们将会讨论表 9-1 列出的工具包。

表 9-1 本章涉及的框架

名 字	下载位置	费 用	主要优点
String SDK	http://poweredbystring.com/licensing	Demo 版 免费，完 全版 499 美元	几乎没有开发时间
Qualcomm	https://ar.qualcomm.at/qdevnet	免费	为缩放跟踪局部标记
ARKit	https://github.com/zac/iphonemarkit	开源	大 约 有 100 个 不 同 的 GitHub 分支

这些工具包中的每一个都有自己独一无二的优势，同时在一些情况下与其他工具包相比会有一些不足。我们将逐个讨论这些工具包并学习一些例子。

9.2 Powered by String 框架

我们以 Powered by String SDK 开始。在你的浏览器中打开 <http://poweredbystring.com/developers/register>。你会来到注册页面；填写所要求的信息。

在你注册完并登录后，来到 <http://poweredbystring.com/licensing> 页面并下载这个工具包的 Demo 版。你需要提供更多信息，之后来到与图 9-1 类似的页面。

下载这个 SDK 的最新版本。在你的本地机器上解压文档并找到 OGL Tutorial 目录。打开 Xcode 工程并在一个物理设备上运行它。在你的计算机上打开 Marker1.png 文件。在应用启动

后, 调节摄像头使标记成为焦点。图 9-2 显示了结果。

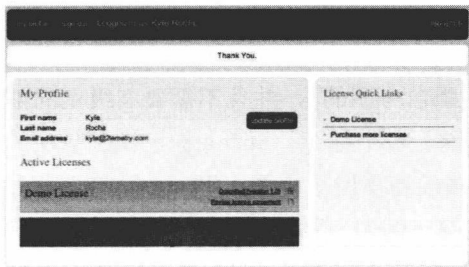


图 9-1 打开 Powered by String 下载页面获取 demo 版

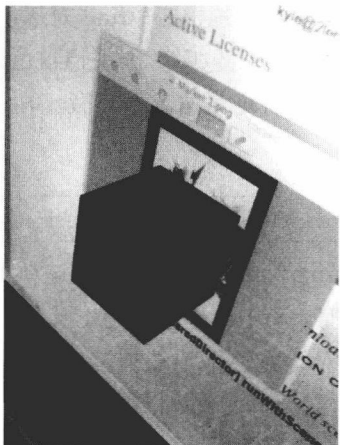


图 9-2 在 String SDK 示例应用中查看标记

我们将会在第 10 章深入讨论这个应用, 因此现在你没有必要对这个内容关注太多。然而, 如果这是你的第一个基于标记的增强现实应用, 你可能觉得非常有趣。仅仅用几行代码, String 就可以从我们的摄像头视图加载和捕获标记。

9.2.1 String 的基本工作流程

同样, 在下一章我们将会学习关于 String 的一个完全教程, 但是这里讲解的是用 String 的 SDK 实现一个 OpenGL 的增强现实应用所需要的基本步骤。

首先, 你需要从下载的源码中添加 Libraries 和 Headers 文件夹 (在那里你发现示例应用)。你还需要 AVFoundation、CoreGraphics、CoreMedia 以及 CoreVideo 框架。

String SDK 的重要特性之一是你可以将它添加到任何视图控制器中 (甚至是另一个类)。你只需简单地创建一个 stringOGL 类的实例并实现 StringOGLDelegate 协议。这个协议只需要你实现一个 render 方法。

在你初始化 String 之前, 你应该建立了帧缓存 (framebuffer)、渲染缓存 (renderbuffer)、工程矩阵 (projection matrix)、视口 (viewport) 以及其他像往常在 OpenGL 应用程序中一样的初始状态。这些中的大部分都是由 Xcode 提供的 OpenGL ES 游戏模板帮你处理的。你需要添加的初始化 String 的代码与代码清单 9-1 中的代码相似。

代码清单 9-1 String 示例初始化代码

```
stringOGL = [[StringOGL alloc] initWithDelegate: self context: myEAGLContext
frameBuffer: myFrameBuffer leftHanded: NO];
[stringOGL setProjectionMatrix: myProjectionMatrix viewport: myViewport
orientation: [self interfaceOrientation] reorientIPhoneSplash: NO];
```

String 建议你不要让应用旋转。在我们的一些例子应用中, 我们实际上接受同样的建议,

即使并不是每一个工具包都要求或者建议这种方式。

String 需要标记。String 对于标记的限制需求相对来说是最少的。标记应该以 PNG 文件格式包含在你的主包中。令人惊讶的是，图像越小工作得越好。如你所料，工具包的加载时间比标记的解析有更多的工作要做。在包含了 PNG 文件后，你可以加载它们，参见代码清单 9-2。

代码清单 9-2 加载 String 标记

```
myMarkerID = [stringOGL loadImageMarker:@"MyMarker" ofType:@"png"];
```

String 对于高对比度的标记执行得非常好（接近黑白分明的），并且不需要高级别的细节信息。同时，String 付款版的工具包可以在同一时间追踪无数个标记。String 标记识别算法的缺点是缺少对不完整标记的追踪能力。如果它被完全阻挡或者移出了视图，String 就完全失去了标记。同样，String 的对象比是根据标记的缩放比设定的。例如，String 认为所有的标记都有一个一单位长度的对角线。因此，如果你需要标记的对角线的长度为 5，你需要在投影图像之前把你追踪到的位置乘以 5。这些设置可以被施加到普通的 OpenGL 变换中。

如我提到的，这个协议只需要实现一个委托方法。你需要在类中实现 render 方法。代码清单 9-3 是该方法的推荐格式。

代码清单 9-3 render 方法的推荐格式

```
- (void)render
{
    // Read data for markers that were detected this frame
    const int maxMarkerCount = 10;
    struct MarkerInfoMatrixBased markerInfo[10];
    |
    int markerCount = [stringOGL getMarkerInfoMatrixBased: markerInfo
    maxMarkerCount: maxMarkerCount];
    // Iterate through detected markers
    for (int i = 0; i < markerCount; i++)
    {
        // Draw appropriate content for this image marker
    }
}
```

让这个框架运行并不需要很多的代码。你简单地追踪了标记，之后遍历了标记信息。如果你使用一个授权来跟踪多个标记，你可以对每一个标记有不同的动作。你可以分辨哪一个标记正在被 markerInfo 类跟踪。

每一个标记都有一个特定的颜色、imageID 和 uniqueInstanceID。如果你有相同的多个标记的话，uniqueInstanceID 是非常有用的。使用这个属性，能分别跟踪多个标记。

9.2.2 额外功能

String 也提供了一些增强现实应用中可能会需要的有用功能的简单封装。在一些情况下，比如本书后面的面部识别例子中，我们需要直接访问帧缓存。String 实际上是拥有屏幕缓存委托的，因此我们需要使用 getCurrentVideoBuffer 方法从 String 中请求每一个帧。代码清单 9-4

显示的就是这个方法。

代码清单 9-4 获取当前帧缓存

```
- (void)getCurrentVideoBuffer: (unsigned *)buffer viewToVideoTextureTransform:
(float *)viewToVideoTextureTransform;
```

这个方法获取了当前视频的纹理和一个用于从视图空间向 OpenGL 的 texture 空间进行坐标变换的矩阵。对于我们其他的例子来说，我们实际上并不需要所有这些信息。如果你真的决定要跟踪标记并分析帧缓存，你需要在每个帧中调用这个方法，因为视频流是双缓存的。

第二个有用的功能是 String 提供的屏幕缓存快照功能。这在通过电子邮件发送应用的图片或者将它们发送到 Facebook 等的情况下是非常有用的。代码清单 9-5 显示了这个例子方法。

代码清单 9-5 从帧缓存获取一个快照

```
- (void)takeSnapshotAndPause;
```

这个方法几乎是一个异步回调方法。你需要等待 handleSnapshot 方法（你需要实现的）完成后，才可以调用 resume 方法。

9.2.3 整合 Unity

在本书中，我们将不会讲解 Unity 3D，在 Apress 网站上有很多关于这个问题的主题。Unity 是一个可以与 String 原生整合的强大 3D 游戏引擎。如果你对 Unity 感兴趣，你可以用它的 starter project（初学者工程），不用一行代码创建一个增强现实应用。

9.2.4 高级着色和 OpenGL 功能

如果你想要看一个可以证明使用 String 能够创建什么的示例，那么可以查看他们网站的展示或者从 App Store 下载示例应用。

String 与高级着色和灯光效果协作良好。它可以快速地对处理对象的加载并且对于标记的显示有一个近乎即时的响应。

图 9-3 展示了一个用 String 建立的演示应用的屏幕截图。它从一个标记处加载了一个 3D 的灯并且允许用户使用 OpenGL ES 的着色和灯光效果来打开和关闭这个灯。

这个演示工程扩展了那个立方体示例（注意图 9-3 中的第二个选项卡）并且为 Lantern 选项卡呈现了一个不同的对象。源码可以在 GitHub 上本书的补充材料里或者 Apress 网站（www.apress.com）上的 Source Code/Download 区域获得。

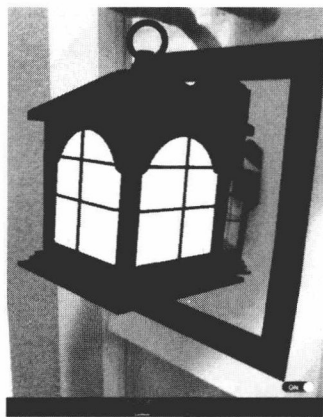


图 9-3 String 的演示应用加载了一个能够打开和关闭的 3D 的灯

9.3 Qualcomm 软件开发工具包

Qualcomm 在 2011 年夏末（正好在 iOS 5 推出以前）发布了一个增强现实软件开发工具包。Qualcomm 与 Powered by String 是非常不同的，这就是我选择讲解这两个框架的原因之一。

访问 Qualcomm 的网站 <https://ar.qualcomm.at>，并注册一个免费的账户。注册后，你可以从 Android 选项卡切换到 iOS 选项卡（见图 9-4），并为 Mac 下载 QCAR SDK。

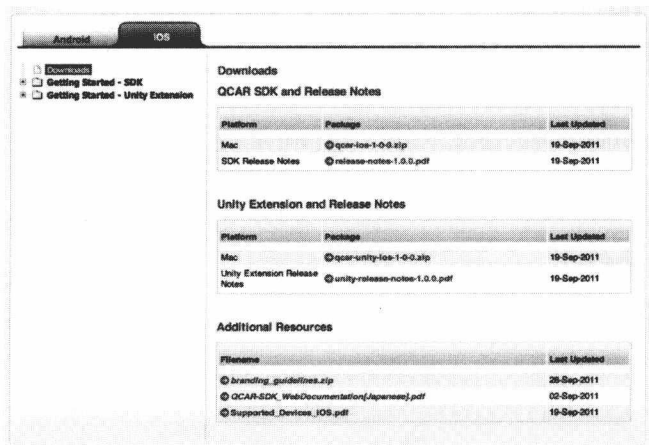


图 9-4 下载 QCAR SDK

与 String 的静态资源库和文件集合不同的是，QCAR 实际上提供了一个安装程序，以使资源库与代码分开。这使你升级这个工具包变得更容易。

注意：

在安装程序启动之前，你可能必须要升级 Java。

解压文档并启动安装程序。你会看到与图 9-5 相似的情景。

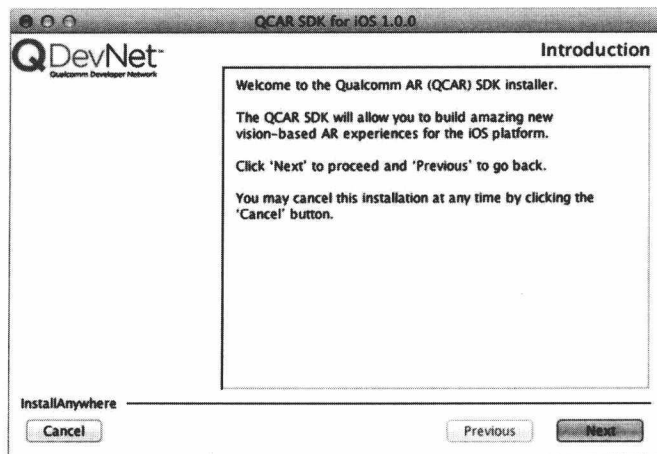


图 9-5 启动安装程序

单击 Next 按钮，你会得到一个询问工具包安装位置的对话框。我将我的安装位置移动到一个用来安装相似工具包的目录，但是你也可以保持默认位置。

接受许可协议并单击 Next 按钮。在你准备好前进后单击 Install 按钮。当资源库安装完毕后，单击 Done 按钮来关闭安装程序。

这个库的目录结构值得关注一下。在这个工具包中有相当数量的令人印象深刻的演示。查看表 9-2 以获悉这个安装目录包含什么，以及从哪里找到它。

表 9-2 QCAR SDK

路 径	包含的内容
build/	Qualcomm 增强现实工具包
build/include	注释过的头文件
build/lib	静态链接库
samples/	示例应用
samples/Dominoes	演示虚拟按钮、声音和触屏交互的示例
samples/ImageTargets	跟踪两个图像目标
samples/FrameMarkers	跟踪多个标记
samples/MultiTargets	跟踪多重目标
samples/VirtualButtons	显示交互用的虚拟按钮
assets/	工具包的额外有用的东西
readme.txt	说明文档

让我们以多米诺骨牌应用开始。这个应用的背后有相当多的代码及数学计算，但是它演示了这个框架的所有功能。在 Xcode 中启动工程并在一个物理设备上运行。

确保你在屏幕上呈现或者显示了在工程的 Media 子目录中的 stones.jpg 文件。这是应用的标记。把这个标记置于摄像头视图中。它会要求你用手指从屏幕上慢慢划过。当你这样做的时候，它会为你建立一组多米诺骨牌。

这就是惊人的部分。你可以使用菜单来开启一个虚拟按钮，使用这个按钮可以在混合现实的空间里轻弹多米诺骨牌（当投射增强场景的时候，你的手指正在摄像头视图里面）。

图 9-6 显示的是运行中的工程。

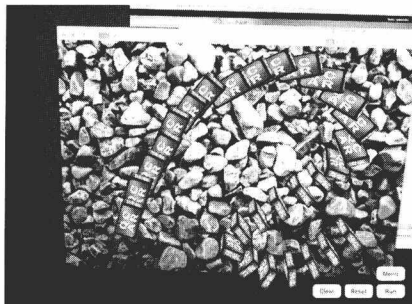


图 9-6 运行多米诺骨牌

在左边 1 号多米诺骨牌的下面有一个绿色的圆点，这是虚拟按钮。如果用户用手指点击了那个圆点，工具包就会记录并处理这个事件。当前事件是让多米诺骨牌按顺序倒下。

这是一个相当惊人的“Hello World”级应用，不是吗？我们将从零开始建立一些东西，以便你习惯使用这个工具包。

9.4 建立我们自己的 QCAR 演示

在从零开始创建之前，我们先为应用创建一个标记。点击 Qualcomm 网站上的 My-Trackables 链接，然后点击 New Project（新工程）链接来创建一个新的工程。我命名我的工程为 Apress。

你会来到一个类似于图 9-7 所示的页面。

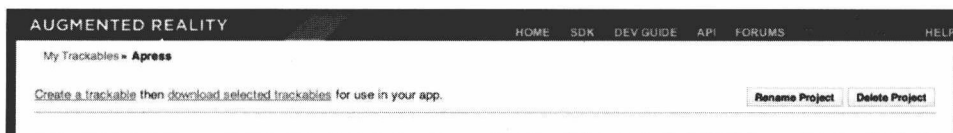


图 9-7 使用 New Project 链接来创建一个新工程

单击 Create a trackable 链接。命名 trackable 为 ApressTrackable 并设置类型为 single image 类型，宽度为 100。

值得注意的是，这个宽度并不与标记的宽度对应，也没有必要将标记调整为这个宽度。在标记周围的增强现实空间中，你将要加载到场景中的 3D 对象会有一个关于标记的相对大小。设置宽度为 100，以此为其他对象设置初始对照。你的 trackable 应该看起来与图 9-8 相似。

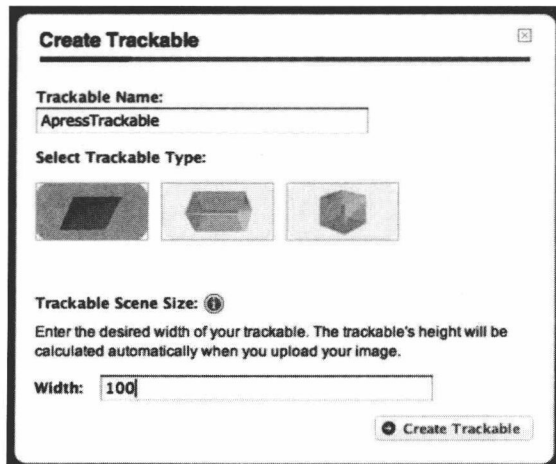


图 9-8 我们创建了一个新的 trackable

现在，相比 String，Qualcomm 更愿意使用一个非常不同的标记。Qualcomm 更喜欢一个高分辨率、高细节的标记，以便增加它能跟踪的点的数量。标识物必须有大量的小细节。类似于

例子中的河石是最好的选择。

为了比较图像，我随使用“Free high resolution wallpaper”（免费高分辨率墙纸）关键字在谷歌图片里面搜索了一下，找到了我认为相当高细节的图像。

通过 Qualcomm 的 trackable 测试，我测试了这些图像中的一部分，结果基本与图 9-9 相似。

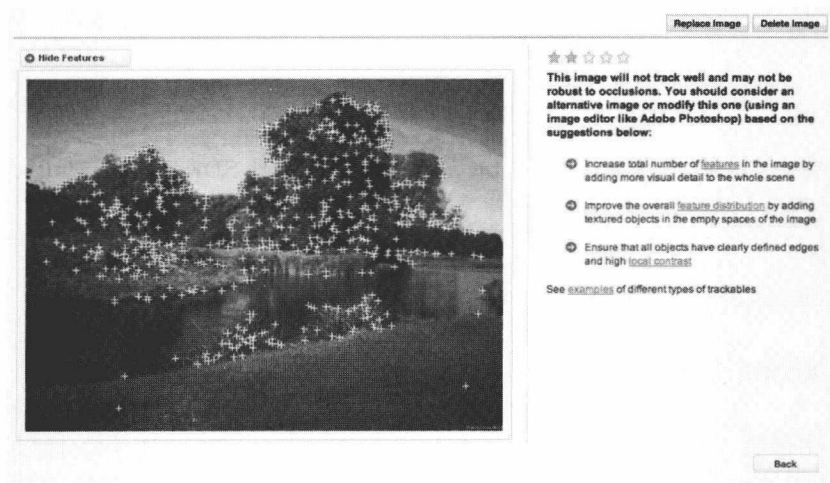


图 9-9 新的 trackable 失败了

如果你想用自己的图像，那就多测试一下。确保你找到了一个细节占据其空间的大部分图像。我重新加载了名为 stones.jpg 的图像。它可以在多米诺骨牌例子的 Media 子目录中找到。这个图的测试结果会更好，如图 9-10 所示。



图 9-10 下一个 trackable 成功了

通过单击 Back 按钮来保存这个 trackable，然后网页会再次返回到 My Trackables 页面。现在在你的 trackable 列表中会有一个新的图像，点击这个图像，你会来到一个页面，在这个页面

里你可以选择一个 trackable 并下载以在我们的应用中使用。屏幕看起来会与图 9-11 类似。

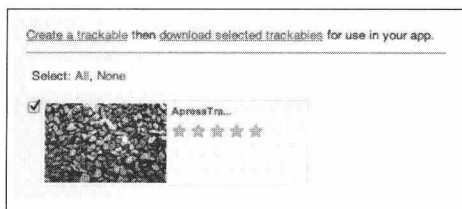


图 9-11 下载 trackable

使用选择窗口顶部的链接下载 trackable。确保你选择的是 SDK 而不是 Unity 版本。文档会包含下面两个文件：

- config.xml
- qcar-resources.dat

9.4.1 创建 Xcode 工程

在 Xcode 中创建一个新工程。使用 Single View application（单视图应用）模板。命名工程为 Ch9。确保你使用了 Automatic Reference Counting（自动引用计数）。我的设置如图 9-12 所示。

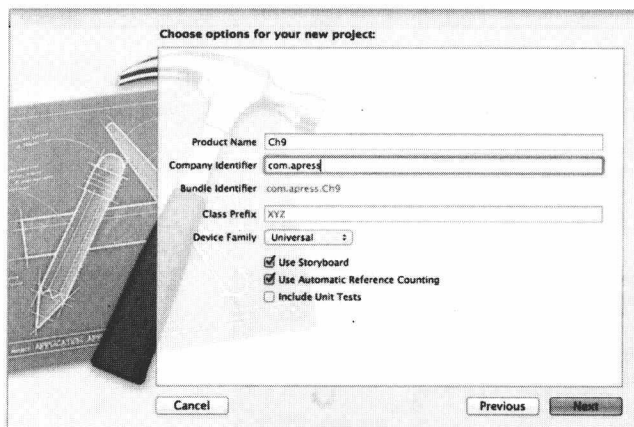


图 9-12 设置工程参数

打开 Finder，导航到 QCAR 工具包的安装位置。有一个叫做 build 的子目录。在 Finder 中打开那个目录。把这个目录下的内容（lib 和 include 文件夹）复制到工程目录。

在 Xcode 工程设置页面打开 Build Settings（构建设置）选项卡。更新 Header Paths（首目录）以包含 \$(SRCROOT)/Ch9/include。现在，用 Finder 打开工程所在目录，然后拖放 lib 目录到 Xcode 工程中。确保你没有选中 lib 目录并将它复制到资源（因为它们已经在工程目录中了）。

从 GitHub 上关于本书的源码仓库或者 Apress 网站（在 www.apress.com 的 Source Code/Download 区域）中复制 GLProgram.h 和 GLProgram.m 文件到工程。这些文件最初是由 Jeff

LaMarche 编写的。它们已经兼容 iOS 5 的自动引用计数功能。

你还要从源码仓库里面复制 Cube.h 文件。这个文件是由 Qualcomm 开发，用于测试 OpenGL ES 绘图。

最后，复制 SimpleLightShader.vsh 和 SimpleLightShader.fsh 文件到工程。打开 Build Phases（构建阶段）选项卡并确保这两个文件已复制以作为捆绑资源，如图 9-13 所示。

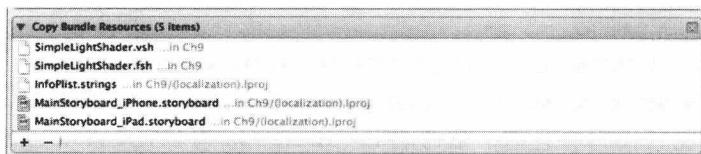


图 9-13 复制捆绑资源

我们需要添加一些框架到工程。在 Xcode 中添加下面的框架：

- OpenGLES.framework
- Security.framework
- AVFoundation.framework
- CoreVideo.framework
- CoreMedia.framework
- SystemConfiguration.framework
- QuartzCore.framework

好的，现在我们已经准备好添加标记文件。复制这两个从 Qualcomm 网站下载的文件到工程。

与 String 一样，我通常建议在增强现实的应用中限制设备的动态重定向，尤其是那些使用第三方软件开发工具包的应用。打开工程的 Summary（摘要）选项卡并确保应用只开启了 Landscape Right（右横向）方向。

9.4.2 EAGLView

首先，创建 UIView 类的一个新子类，命名为 EAGLView。然后，在 Xcode 中打开 EAGLView.h 文件，并按代码清单 9-6 显示的代码导入头文件。

代码清单 9-6 导入语句

```
#import <OpenGLES/EAGL.h>
#import <OpenGLES/ES1/gl.h>
#import <OpenGLES/ES1/glex.h>
#import <OpenGLES/ES2/gl.h>
#import <OpenGLES/ES2/glex.h>
#import <QCAR/Tool.h>
#import <QCAR/UIViewProtocol.h>
#import "GLProgram.h"
```

在 import（导入）语句的下面建立一个枚举来追踪应用的当前运行状态。复制代码清单 9-7 中的代码。

代码清单 9-7 状态枚举

```
typedef enum _status {
    APPSTATUS_UNINITED,
    APPSTATUS_INIT_APP,
    APPSTATUS_INIT_QCAR,
    APPSTATUS_INIT_APP_AR,
    APPSTATUS_INIT_TRACKER,
    APPSTATUS_INITED,
    APPSTATUS_CAMERA_STOPPED,
    APPSTATUS_CAMERA_RUNNING,
    APPSTATUS_ERROR
} status;
```

在我们更新实现的代码块之前，重命名 EAGLView.m 为 EAGLView.mm。因为 Qualcomm 工具包是基于 C++ 的，所以我们将需要以这个扩展名来访问 C++ 类。

按代码清单 9-8 显示的代码更新 interface 块。

代码清单 9-8 EAGLView 的 interface

```
@interface EAGLView : UIView <UIGLViewProtocol> {
    EAGLContext *context;
    GLint framebufferWidth;
    GLint framebufferHeight;

    GLuint defaultFramebuffer;
    GLuint colorRenderbuffer;
    GLuint depthRenderbuffer;

    QCAR::Matrix44F projectionMatrix;
    CGRect screenRect;
    int QCARFlags;
    status appStatus;
    GLProgram *shader;
    GLint shaderPositionAttribute, shaderNormalAttribute, shaderModelViewMatrixUniform,
    shaderProjectionMatrixUniform, shaderColorUniform;
}
@end
```

这里，我们首先为 OpenGL 的上下文建立一个实例变量。之后，我们为帧缓存的宽和高以及默认的缓存处理建立了一些变量。接下来，我们建立了用于复合 3D 坐标的投影矩阵，就是这一行需要把扩展名改成 .mm。如果你想测试一下的话，你可以把它改回来。你会看到一个生成错误。

最后，我们保留了一些关于屏幕大小、QCAR 标记，以及应用状态的变量（来自我们的枚举）。在头文件中声明代码清单 9-9 中的方法。

代码清单 9-9 新的方法声明

```
- (void)renderFrameQCAR;
- (void)onCreate;
- (void)onDestroy;
- (void)onResume;
- (void)onPause;
```

第一个方法将会被 QCAR 工具包调用来显示帧。当应用状态发生改变的时候，视图控制器会调用剩下的几个方法。

在 Xcode 中切换到 EAGLView.mm 文件，添加代码清单 9-10 中的导入语句。

代码清单 9-10 新的导入语句

```
#import <QuartzCore/QuartzCore.h>
#import <QCAR/QCAR.h>
#import <QCAR/CameraDevice.h>
#import <QCAR/Tracker.h>
#import <QCAR/VideoBackgroundConfig.h>
#import <QCAR/Renderer.h>
#import <QCAR/Tool.h>
#import <QCAR/Trackable.h>

#import "Cube.h"
```

我们将会需要工具包所提供的几乎所有的头文件。因此，意味着这个地方会有很多的导入语句。你可能已经注意到，在易用性和时间节约方面，Qualcomm 和 String 有所不同。不过，如果你需要更多的 QCAR 工具包的高级跟踪能力的话，那这些努力就是值得的。

添加下面代码清单 9-11 中的私有方法声明代码块到类中。

代码清单 9-11 私有方法

```
@interface EAGLView (PrivateMethods)
- (void)setFramebuffer;
- (BOOL)presentFramebuffer;
- (void)createFramebuffer;
- (void)deleteFramebuffer;
- (void)updateApplicationStatus:(status)newStatus;
- (void)bumpAppStatus;
- (void)initApplication;
- (void)initQCAR;
- (void)initApplicationAR;
- (void)loadTracker;
- (void)startCamera;
- (void)stopCamera;
- (void)configureVideoBackground;
- (void)initRendering;
@end
```

添加代码清单 9-12 中的静态方法到实现。这个方法在 OpenGL 向层绘图的时候会用到。

代码清单 9-12 layerClass 静态方法

```
+ (Class)layerClass
{
    return [CAEAGLLayer class];
}
```

我们将会从故事板的视图控制器 NIB 文件中加载这个类。因此我们将必须实现 initWithCoder 方法。复制代码清单 9-13 中的方法。

代码清单 9-13 initWithCoder

```

- (id)initWithCoder:(NSCoder*)coder
{
    self = [super initWithCoder:coder];

    if (self) {
        NSLog(@"Initialising EAGLView");
        CAEAGLLayer *eaglLayer = (CAEAGLLayer *)self.layer;

        eaglLayer.opaque = TRUE;
        eaglLayer.drawableProperties = [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:FALSE],
            kEAGLDrawablePropertyRetainedBacking,
            kEAGLColorFormatRGBA8,
            kEAGLDrawablePropertyColorFormat,
            nil];

        context = [[EAGLContext alloc] initWithAPI:kEAGLRenderingAPIOpenGLES2];
        QCARFlags = QCAR::GL_20;

        NSLog(@"QCAR OpenGL flag: %d", QCARFlags);

        if (!context) {
            NSLog(@"Failed to create ES context");
        }
    }

    return self;
}

```

这段代码与我们在 OpenGL ES 模板或者其他 OpenGL 应用中通常看到的代码的关键不同是 QCAR 标记。我们在上面的代码中建立了 QCAR 标记，以使工具包知道会发生什么。

复制代码清单 9-14 中的方法到实现。

代码清单 9-14 createFramebuffer 方法

```

- (void)createFramebuffer
{
    if (context && !defaultFramebuffer) {
        [EAGLContext setCurrentContext:context];

        // Create default framebuffer object
        glGenFramebuffers(1, &defaultFramebuffer);
        glBindFramebuffer(GL_FRAMEBUFFER, defaultFramebuffer);

        // Create colour render buffer and allocate backing store
        glGenRenderbuffers(1, &colorRenderbuffer);
        glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);

        // Allocate the renderbuffer's storage (shared with the drawable object)
        [context renderbufferStorage:GL_RENDERBUFFER fromDrawable:(CAEAGLLayer*)
self.layer];
        glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_WIDTH,
&framebufferWidth);

        glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_HEIGHT,
&framebufferHeight);

        // Create the depth render buffer and allocate storage
        glGenRenderbuffers(1, &depthRenderbuffer);
        glBindRenderbuffer(GL_RENDERBUFFER, depthRenderbuffer);
        glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT16, framebufferWidth,
framebufferHeight);
    }
}

```

```

        // Attach colour and depth render buffers to the frame buffer
        glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_RENDERBUFFER,
        colorRenderbuffer);
        glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER,
        depthRenderbuffer);

        // Leave the colour render buffer bound so future rendering operations will act
        on it
        glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);

        if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE) {
            NSLog(@"Failed to make complete framebuffer object %x",
            glCheckFramebufferStatus(GL_FRAMEBUFFER));
        }
    }
}

```

这个方法以及后面的一些方法都来自 OpenGL 模板。我们通过代码清单 9-14 中复制来的方法建立了帧缓存，这使在层上用 OpenGL 绘图成为可能。

我们还将需要帧缓存的拆除方法。代码清单 9-15 中的方法在帧缓存不再需要内存的时候将其释放。

代码清单 9-15 deleteFramebuffer 方法

```

- (void)deleteFramebuffer
{
    if (context) {
        [EAGLContext setCurrentContext:context];
        if (defaultFramebuffer) {
            glDeleteFramebuffers(1, &defaultFramebuffer);
            defaultFramebuffer = 0;
        }

        if (colorRenderbuffer) {
            glDeleteRenderbuffers(1, &colorRenderbuffer);
            colorRenderbuffer = 0;
        }

        if (depthRenderbuffer) {
            glDeleteRenderbuffers(1, &depthRenderbuffer);
            depthRenderbuffer = 0;
        }
    }
}

```

复制代码清单 9-16 中的方法。这个方法建立了我们之前声明的 defaultFramebuffer 实例变量。

代码清单 9-16 setFrameBuffer

```

- (void)setFramebuffer
{
    if (context) {
        [EAGLContext setCurrentContext:context];

        if (!defaultFramebuffer) {
            // Perform on the main thread to ensure safe memory allocation for
            // the shared buffer. Block until the operation is complete to
            // prevent simultaneous access to the OpenGL context
            [self performSelectorOnMainThread:@selector(createFramebuffer)
            withObject:self waitUntilDone:YES];
        }
    }
}

```

```

    }
    glBindFramebuffer(GL_FRAMEBUFFER, defaultFramebuffer);
}
}

```

我们就快把无聊的事情做完了。复制代码清单 9-17 中的方法到实现。

代码清单 9-17 presentFramebuffer 和 layoutSubviews 方法

```

- (BOOL)presentFramebuffer
{
    BOOL success = FALSE;

    if (context) {
        [EAGLContext setCurrentContext:context];

        glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);

        success = [context presentRenderbuffer:GL_RENDERBUFFER];
    }

    return success;
}

- (void)layoutSubviews
{
    [self deleteFramebuffer];
}

```

刚刚复制的第一个方法为我们的类保存 colorRenderbuffer 环境。当 EAGLView 的子视图发生变化或者视图发生重绘的时候，第二个方法会被调用。当这些发生的时候，我们希望释放当前图像占用的内存。

当建立了 interface 文件后，我们讨论过这个类最有可能被一个外部视图控制器调用。接下来的一组事件是关于视图控制器的。复制代码清单 9-18 中的方法。

代码清单 9-18 ViewController 的外部方法

```

- (void)onCreate
{
    NSLog(@"EAGLView onCreate()");
    appStatus = APPSTATUS_UNINITED;

    [self updateApplicationStatus:APPSTATUS_INIT_APP];
}

- (void)onDestroy
{
    NSLog(@"EAGLView onDestroy()");

    // Deinitialise QCAR SDK
    QCAR::deinit();
}

- (void)onResume
{
    NSLog(@"EAGLView onResume()");
    // QCAR-specific resume operation
    QCAR::onResume();
}

```

```

        if (APPSTATUS_CAMERA_STOPPED == appStatus) {
            [self updateApplicationStatus:APPSTATUS_CAMERA_RUNNING];
        }
    }

- (void)onPause
{
    NSLog(@"EAGLView onPause()");
    // QCAR-specific pause operation
    QCAR::onPause();

    if (APPSTATUS_CAMERA_RUNNING == appStatus) {
        [self updateApplicationStatus:APPSTATUS_CAMERA_STOPPED];
    }
}

```

在应用的生命周期中，这些方法中的每一个都会在不同的时刻被调用。它们都不是很复杂。每一个都会改变状态为枚举中的一个特定值，要么暂停，要么开始，要么恢复 QCAR 工具包的操作。

现在，当应用改变状态的时候，我们必须对这些改变做相应的处理。复制代码清单 9-19 中的方法来处理应用的状态改变。

代码清单 9-19 处理状态改变

```

- (void)updateApplicationStatus:(status)newStatus
{
    if (newStatus != appStatus && APPSTATUS_ERROR != appStatus) {
        appStatus = newStatus;

        switch (appStatus) {
            case APPSTATUS_INIT_APP:
                [self initApplication];
                [self updateApplicationStatus:APPSTATUS_INIT_QCAR];
                break;

            case APPSTATUS_INIT_QCAR:
                [self performSelectorInBackground:@selector(initQCAR) withObject:nil];
                break;

            case APPSTATUS_INIT_APP_AR:
                [self initApplicationAR];
                [self updateApplicationStatus:APPSTATUS_INIT_TRACKER];
                break;

            case APPSTATUS_INIT_TRACKER:
                [self performSelectorInBackground:@selector(loadTracker)
                withObject:nil];
                break;

            case APPSTATUS_INITED:
                QCAR::setHint(QCAR::HINT_IMAGE_TARGET_MULTI_FRAME_ENABLED, 1);
                QCAR::setHint(QCAR::HINT_IMAGE_TARGET_MILLISECONDS_PER_MULTI_FRAME, 25);
                [self updateApplicationStatus:APPSTATUS_CAMERA_RUNNING];
                break;

            case APPSTATUS_CAMERA_RUNNING:
                [self startCamera];
                break;

            case APPSTATUS_CAMERA_STOPPED:
                [self stopCamera];
                break;

            default:
                NSLog(@"updateApplicationStatus: invalid app status");
                break;
        }
    }

    if (APPSTATUS_ERROR == appStatus) {
        UIAlertView* alert = [[UIAlertView alloc] initWithTitle:@"Error"

```

```

message:@"Application initialisation failed." delegate:self cancelButtonTitle:@"OK"
otherButtonTitles:nil];

    [alert show];
}
}

```

请特别注意一下粗体行。第一行调用了一个后台线程来初始化 QCAR 工具包。第二行在后台加载了一个标记图像。我们还没有实现过这种方法。我们接下来实现它。

复制代码清单 9-20 中的两个方法。

代码清单 9-20 加载标记（或者说 Trackable）

```

- (void)bumpAppStatus
{
    [self updateApplicationStatus:(status)(appStatus + 1)];
}

- (void)loadTracker
{
    int nPercentComplete = 0;

    // Background thread must have its own autorelease pool
    // Load the tracker data
    do {
        nPercentComplete = QCAR::Tracker::getInstance().load();
    } while (0 <= nPercentComplete && 100 > nPercentComplete);

    if (0 > nPercentComplete) {
        appStatus = APPSTATUS_ERROR;
    }

    // Continue execution on the main thread
    [self performSelectorOnMainThread:@selector(bumpAppStatus) withObject:nil
    waitUntilDone:NO];
}

```

这些方法用于加载标记。当标记加载后，改变应用到枚举中的下一个状态（如粗体行显示的）。我们引用但是还没有实现的其他选择器（selector）是 QCAR 的初始化方法。我们将为整个类一次添加这 3 个初始化方法。复制代码清单 9-21 中的方法到实现。

代码清单 9-21 各种 init 方法

```

- (void)initApplication
{
    screenRect = [[UIScreen mainScreen] bounds];

    NSLog(@"Screen rect %@",screenRect);

    QCAR::onSurfaceCreated();
    QCAR::onSurfaceChanged(screenRect.size.height, screenRect.size.width);
}

- (void)initQCAR
{
    QCAR::setInitParameters(QCARFlags);

    int nPercentComplete = 0;

    do {

```

```

        nPercentComplete = QCAR::init();
    } while (0 <= nPercentComplete && 100 > nPercentComplete);
    NSLog(@"QCAR::init percent: %d", nPercentComplete);

    if (0 > nPercentComplete) {
        appStatus = APPSTATUS_ERROR;
    }

    [self performSelectorOnMainThread:@selector(bumpAppStatus) withObject:nil
     waitUntilDone:NO];
}

- (void)initApplicationAR
{
    [self initRendering];
}

```

第一个方法获取屏幕的尺寸，创建了 QCAR 的表面并调节它的大小来匹配屏幕的尺寸。当这些完成的时候它还发出了通知（onSurfaceCreated）。

接下来，我们初始化了 Qualcomm 工具包。这个过程是在后台线程进行的。当完成后，我们再次移动到枚举的下一个状态值。

我们刚刚实现的最后一个方法初始化了显示。这是对于应用的增强现实部分的主要初始化。

我们需要两个方法，可以在暂停和恢复的例程中进行调用来开启和关闭摄像头。代码清单 9-22 显示的就是这两个方法。

代码清单 9-22 开始和停止摄像头视图

```

- (void)startCamera
{
    NSLog(@"Start Camera!");
    // Initialise the camera
    if (QCAR::CameraDevice::getInstance().init()) {
        // Configure video background
        [self configureVideoBackground];

        // Select the default mode
        if
        (QCAR::CameraDevice::getInstance().selectVideoMode(QCAR::CameraDevice::MODE_DEFAULT)) {
            // Start camera capturing
            if (QCAR::CameraDevice::getInstance().start()) {
                // Start the tracker
                QCAR::Tracker::getInstance().start();

                // Cache the projection matrix
                const QCAR::CameraCalibration& cameraCalibration =
                QCAR::Tracker::getInstance().getCameraCalibration();
                projectionMatrix = QCAR::Tool::getProjectionGL(cameraCalibration, 2.0f,
                2000.0f);
                [self onResume];
            }
        }
    }
}

- (void)stopCamera
{
    QCAR::Tracker::getInstance().stop();
    QCAR::CameraDevice::getInstance().stop();
    QCAR::CameraDevice::getInstance().deinit();
}

```

这些都是 QCAR 类库的 C++ 函数。其中的结构体与我们在 Objective-C 中处理事务或者在视频中使用 OpenGL 纹理非常相似，但方法是工具包特有的。

我们几乎为测试应用做好了准备。在没有进行任何测试的情况下，我们更新了这么多代码。

我们设置视频作为我们的背景层。复制代码清单 9-23 中的方法。

代码清单 9-23 设置视频为背景图像

```

- (void)configureVideoBackground
{
    // Get the default video mode
    QCAR::CameraDevice& cameraDevice = QCAR::CameraDevice::getInstance();
    QCAR::VideoMode videoMode =
    cameraDevice.getVideoMode(QCAR::CameraDevice::MODE_DEFAULT);

    // Configure the video background
    QCAR::VideoBackgroundConfig config;
    config.mEnabled = true;
    config.mSynchronous = true;
    config.mPosition.data[0] = 0.0f;
    config.mPosition.data[1] = 0.0f;

    // Compare aspect ratios of video and screen. If they are different
    // we use the full screen size while maintaining the video's aspect
    // ratio, which naturally entails some cropping of the video.
    // Note: screenRect is portrait but videoMode is always landscape,
    // which is why "width" and "height" appear to be reversed.
    float arVideo = (float)videoMode.mWidth / (float)videoMode.mHeight;
    float arScreen = screenRect.size.height / screenRect.size.width;

    if (arVideo > arScreen)
    {
        // Video mode is wider than the screen. We'll crop the left and right edges of
        the video
        config.mSize.data[0] = (int)screenRect.size.width * arVideo;
        config.mSize.data[1] = (int)screenRect.size.width;
    }
    else
    {
        // Video mode is taller than the screen. We'll crop the top and bottom edges of
        the video.
        // Also used when aspect ratios match (no cropping).
        config.mSize.data[0] = (int)screenRect.size.height;
        config.mSize.data[1] = (int)screenRect.size.height / arVideo;
    }

    // Set the config
    QCAR::Renderer::getInstance().setVideoBackgroundConfig(config);
}

```

这个方法是从一个例子应用中直接复制过来的。Qualcomm 正设置视频与屏幕匹配，接着把摄像头视图作为我们的背景。

我们为 OpenGL 的光设置加载着色器。复制代码清单 9-24 中的方法。

代码清单 9-24 加载 Test Shaders

```

- (void)loadShaders {
    // Loading shaders for light only

```

```

    shader = [[GLProgram alloc] initWithVertexShaderFilename:@"SimpleLightShader"
fragmentShaderFilename:@"SimpleLightShader"];
    [shader addAttribute:@"a_position"];
    [shader addAttribute:@"a_texCoord"];
    [shader addAttribute:@"a_normal"];

    if (![shader link])
    {
        // Compilation failed
        NSLog(@"light shader link failed");
        NSString *progLog = [shader programLog];
        NSLog(@"Program Log: %@", progLog);
        NSString *fragLog = [shader fragmentShaderLog];
        NSLog(@"Frag Log: %@", fragLog);
        NSString *vertLog = [shader vertexShaderLog];
        NSLog(@"Vert Log: %@", vertLog);
        shader = nil;
    } else {
        NSLog(@"Light Shader compiled successfully!");
    }

    shaderPositionAttribute = [shader attributeIndex:@"a_position"];
    shaderNormalAttribute = [shader attributeIndex:@"a_normal"];
    shaderColorUniform = [shader uniformIndex:@"a_color"];
    shaderModelViewMatrixUniform = [shader uniformIndex:@"modelViewMatrix"];
    shaderProjectionMatrixUniform = [shader uniformIndex:@"projectionMatrix"];
}

```

在能够用 `EAGLView` 类来代替我们应用的 `UIView` 类之前，我们还有两个任务要做。我们需要初始化 `OpenGL` 渲染并呈现帧。你可以在代码清单 9-25 中找到这两个方法。

代码清单 9-25 初始化 `OpenGL` 渲染并呈现帧

```

- (void)initRendering
{
    // Define the clear colour
    glClearColor(0.0f, 0.0f, 0.0f, QCAR::requiresAlpha() ? 0.0f : 1.0f);

    [self loadShaders];
}

- (void)renderFrameQCAR
{
    [self setFramebuffer];

    // Clear colour and depth buffers
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    QCAR::State state = QCAR::Renderer::getInstance().begin();

    glEnable(GL_DEPTH_TEST);
    glEnable(GL_CULL_FACE);

    for (int i = 0; i < state.getNumActiveTrackables(); ++i) {
        // Get the trackable
        const QCAR::Trackable* trackable = state.getActiveTrackable(i);

        QCAR::Matrix44F modelViewMatrix = QCAR::Tool::convertPose2GLMatrix(trackable->
        getPose());

        [shader use];

        float Sash_Kd [] = {0.589414, 0.042139, 0.042139};
        // Set the sampler texture unit to 0
        glUniformMatrix4fv(shaderProjectionMatrixUniform, 1, 0,

```

```

&projectionMatrix.data[0]);
    glUniformMatrix4fv(shaderModelViewMatrixUniform, 1, 0,
&modelViewMatrix.data[0]);
    glUniform3fv(shaderColorUniform, 1, Sash_Kd);

    glVertexAttribPointer(shaderPositionAttribute, 3, GL_FLOAT, GL_FALSE, 0, (const
GLvoid*)&cubeVertices[0]);
    glVertexAttribPointer(shaderNormalAttribute, 3, GL_FLOAT, GL_FALSE, 0, (const
GLvoid*)&cubeNormals[0]);

    glEnableVertexAttribArray(shaderPositionAttribute);
    glEnableVertexAttribArray(shaderNormalAttribute);

    glDrawElements(GL_TRIANGLES, NUM_CUBE_INDEX, GL_UNSIGNED_SHORT, (const
GLvoid*)&cubeIndices[0]);
}

glDisable(GL_DEPTH_TEST);
glDisable(GL_CULL_FACE);

QCAR::Renderer::getInstance().end();
[self presentFramebuffer];
}

```

第一个方法加载了着色器并定义了一个可以在层中引用的清除颜色。在 OpenGL 渲染进行了初始化以后，我们可以绘制 QCAR 帧了。

在这个方法中，我们遍历了能够找到的所有 trackable 的数组，就像在 String 工具包中所做的那样，并且在 trackable 正上面的 3D 空间中显示了一个立方体。你可以用任何的 OpenGL 渲染例程来代替这个代码块。

9.4.3 重定向 UIView

修改 ViewController.m 文件名为 ViewController.mm。之前，我们对 EAGLView 进行了同样的操作。我们将需要访问 Qualcomm 工具包的 C++ 函数，因此这个名字符合编译器的需要。

打开 ViewController.mm 文件。导入 EAGLView.h 头文件。按代码清单 9-26 显示的代码更新 viewDidLoad 方法。

代码清单 9-26 新版 viewDidLoad

```

- (void)viewDidLoad
{
    [super viewDidLoad];

    EAGLView * eaglview = (EAGLView*) self.view;
    [eaglview onCreate];
    // Do any additional setup after loading the view, typically from a nib.
}

```

我们创建了一个 EAGLView 类的新实例并通知类该实例的创建。按代码清单 9-27 显示的代码更新 viewWillAppear 和 viewWillDisappear 方法。

代码清单 9-27 新版 viewWillAppear 和 viewWillDisappear 方法

```

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    EAGLView * eaglview = (EAGLView*) self.view;
    [eaglview onResume];
}

- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated];
    EAGLView * eaglview = (EAGLView*) self.view;
    [eaglview onPause];
}

```

是的，我们成功了！好的，我们还需要做一些事情来确保我们的类要在实际中应用。打开故事板中的任意一个（或两个都打开），并修改 UIView 的 class 属性为 EAGLView，如图 9-14 所示。

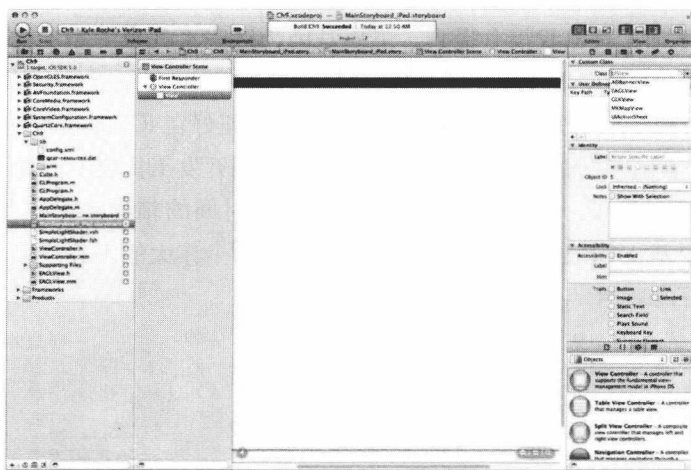


图 9-14 修改 UIView 的 class 属性为 EAGLView

现在你可以运行工程了。不要忘记打印标记或者把它置于屏幕视图中。如果直接从本书的源代码仓库加载的配置文件，你可以在 Qualcomm 的下载目录 /Samples/Dominos/media 下找到这个标记。

这个应用应该以一个单独的全屏摄像头视图启动。让摄像头对准标记，你会看到与图 9-15 类似的情景。

老实说，当我运行它的时候我很紧张。这是一个很大的、事先未进行过任何测试的代码复制。祝贺你！如果你想让这个应用更有趣



图 9-15 演示应用成功了

一点，你可以用其他例子应用中的代码替换本例中的 OpenGL 渲染方法。如果你想学习关于 OpenGL 的更多知识，可以查看 Apress (www.apress.com) 上的相关主题。

9.5 ARKit

ARKit 是 Zac White 发起的一个老牌的 GitHub 项目。我提到这个项目是因为我（以及其他 100 个人）为了一些有用的功能已经对这个库做了分支。在第 11 章我会把它的一些功能用于基于位置的增强现实。如果你想要从零开始创建一些东西，可以浏览一下 www.github.com 上的各种分支。另外，social code 等于 fun。

9.6 总结

还有好多软件开发工具包我没有提到。主要是因为这些工具包的帮助文档做得很好。例如 Metaio，在网上已有很多示例工程和帮助文档。String 是在 2011 年夏天发布的，但是在他们的网站上仍然没有帮助文档（在本书编写的时候）。Qualcomm 正好在 iOS 5 发布之前发布了他们的工具包。在他们的开发者门户网站上有一些帮助文档，同时在开发工具包中也有一些惊人的例子。

我希望你在本章学到了一些新的可以节省应用开发时间的辅助类库。在第 10 章，我会使用 String 建立另一个基于标记的增强现实应用。你会在两个伟大的工具包以及它们的不同实现方式之间有一个良好的对照。

第 10 章

使用 OpenGL ES 创建基于标记的增强现实应用

在第 9 章，我们学习了 iOS 开发者可用于建立增强现实应用的各种工具包。本章将以这些工具包中的一个开始，即 iOS 版 String SDK，并建立一个基于标记的增强现实应用。

增强现实标记是用于指导应用怎么根据物理世界来定向和缩放的物理标识。例如在第 1 章中，我展示了一个美国邮政管理局怎么使用标记来定向和缩放一个货运纸箱的例子。你可以在你的应用中做相同的事情。

在应用中使用标记的好处是可以为现实世界已经存在的东西增加价值。例如，假设你有一个客户，他拥有一家低成本运转的纸制期刊。使用现有的广告或者图标之一，你就能把一个 3D 广告带到现实中。除了增加价值，它还为客户提供了进一步货币化其广告客户端的机会。

广告是基于标记的增强现实的一个重要使用案例。我们将会用一个现实的例子来扩展这个使用案例。

10.1 建立标记

标记有一些特性，能确保应用能够正确使用标记。首先，一个增强现实的标记只能有一个正确的方向。你应该能够清楚地区分标记是倒置还是侧放。正方形标记的效果并不好，因为应用将无法确定它的正确方向。其次，一个增强现实标记应该由一个高对比度的、独一无二的图像构成。使用黑和白，或者其他在弱光环境里可以相互区分的颜色。在大部分情况下，如果你正打印标记的话，你要避免强光和反射面。

10.1.1 我们的标记

对于本例，我把标记放到了 GitHub 仓库中。本章的所有代码都可以从 https://github.com/Kyleroch/Professional_iOS_AugmentedReality 得到。同时也可以从 Arpress 网站（www.arpress.com）的 Source Code/Download 区域获得这些源代码。

我们将会使用本书的封面作为我们的增强现实标记。假设印刷的过程中图片一直没有变化，这会让测试稍微容易一点，因为你不用再去打印任何东西了。

为了投影 3D 模型，我们将会使用 OpenGL ES（用于嵌入式系统的 OpenGL）。Xcode 中有一个现成的 OpenGL Gaming 模板可用。但是，为了教学目的，我们从零开始建立这个应用。在开始之前，我们先了解下 OpenGL 的基本功能。

10.1.2 OpenGL ES

OpenGL ES 是一个用于高级嵌入式图形的底层、轻量应用程序编程接口，是 OpenGL 的一个定义清晰的子集。它提供了一个介于应用和软 / 硬件图形引擎之间的底层应用程序编程接口。

使用 OpenGL ES 来开发有很多优势：

- 是行业标准且无版税：任何人都能够下载 OpenGL ES 说明书并实现和发布一个基于 OpenGL ES 的产品。
- 占用空间小、低功耗：嵌入空间根据处理能力的大小而变化。OpenGL ES 通过尽可能少的数据存储以实现尽可能少的空间占用以适应这些不同。
- 可扩展、可升级：OpenGL 的扩展机制在 OpenGL ES 中同样起作用，因此你可以添加新的可用硬件的配置文件。
- 易用性和良好的帮助文档资源：OpenGL 是 OpenGL ES 的基础。在网上以及出版物中有很多可用的培训资料。

无论这本书还是这一章都不能让你成为 OpenGL ES 专家。但是，我们应该记住创建这个应用所用到的功能，并且如果你想学习更多关于 iOS 的 3D 编程知识，我会引用一些你能够访问的资料。

10.2 创建工程

打开 Xcode 并创建一个新工程。使用 Single View Application（单一视图应用）模板来获得工程的一些默认设置。确保你选择了 Universal 作为你的 Device Family（设备族）。你可以开启 Automatic Reference Counting（自动引用计数）。

10.2.1 添加 String 框架

解压从 String 网站下载的文件到本地机器的一个目录中。从 Libraries 文件夹复制 libString OGL*.a 文件到你的 Xcode 工程。从 Headers 文件夹复制 StringOGL.h 和 Tracker Output.h 文件到你的 Xcode 工程。这就是 String 框架中我们需要的所有文件。我们继续添加工程依赖关系，之后将返回到 String 框架怎么使用的主题上来。

工程依赖关系

我们需要添加一些框架到我们的 Xcode 工程来确保 String 框架拥有了它所需要的资源。在 Xcode 导航器中单击你的工程的名字。切换到 Build Phases（构建阶段）选项卡并添加 QuartzCore、

OpenGL ES、AVFoundation、CoreGraphics、CoreMedia 以及 CoreVideo 框架到 Link Binary with Libraries（用库链接至二进制）区域。

从 Build Phases（构建阶段）选项卡切换到 Build Settings（构建设置）选项卡。找到叫做 Other Linker Flags（其他链接者标识）的配置项，设置其值为 -lstdc++。参见图 10-1，该图显示的是迄今为止我们所完成的操作。

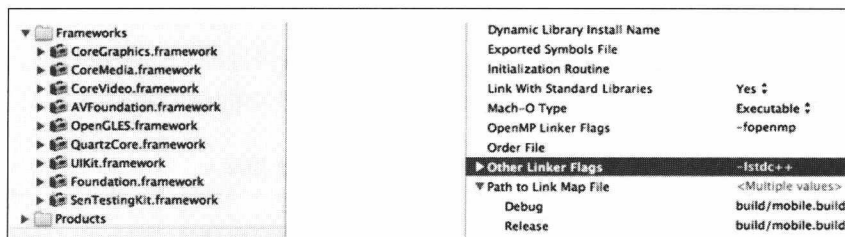


图 10-1 我们已经正确建立了 Frameworks 和 Linker Flags

10.2.2 EAGLView

在 Xcode 的上一个版本中，OpenGL 模板包含了 EAGLView 模板。Xcode 4.2 以及 iOS 5 引入了一个新的 OpenGL Gaming 模板并移除了上一个 OpenGL application 模板。这个新模板不再提供上一个版本中的 EAGLView 文件。因此我们必须自己创建。iOS 4.3 的模板并不支持 Automatic Reference Counting（自动引用计数）。我们也会添加这个到工程中。

在你的 Xcode 工程中创建一个新 Objective-C 类文件。命名这个类为 EAGLView。现在暂时让它作为一个 NSObject 的子类，稍后我们再更改。

在 Xcode 中打开 EAGLView.h 文件并移除现存的导入语句。添加代码清单 10-1 中的导入语句到其中。

代码清单 10-1 导入语句

```
#import <UIKit/UIKit.h>
#import <OpenGL/ES1/gl.h>
#import <OpenGL/ES1/glex.h>
#import <OpenGL/ES2/gl.h>
#import <OpenGL/ES2/glex.h>
@class EAGLContext;
```

接下来，修改接口声明以使 EAGLView 继承自 UIView 而不是 NSObject。我们将会声明一些与我们将会用来呈现 3D 对象的帧缓存有关的属性。按代码清单 10-2 中显示的代码更新 EAGLView.h 类。

代码清单 10-2 为 EAGLView 更新 @interface

```
@interface EAGLView : UIView {
    GLint framebufferWidth;
    GLint framebufferHeight;
```

```

        GLuint defaultFramebuffer, colorRenderbuffer;
        GLuint depthRenderbuffer;
    }

    @property (nonatomic, retain) EAGLContext *context;
    @property (nonatomic, readonly) GLuint defaultFramebuffer;
    @property (nonatomic, readonly) GLint framebufferWidth;
    @property (nonatomic, readonly) GLint framebufferHeight;

    - (void)setFramebuffer;

```

这个代码块的大部分代码与以前的 iOS 4.3 模板是相当接近的。为简单起见，我们对它做了一些精简。切换到 EAGLView.m 文件并添加代码清单 10-3 中的导入语句。

代码清单 10-3 导入 QuartzCore 类库

```
#import <QuartzCore/QuartzCore.h>
```

接下来我们要为类声明一些私有方法。就在导入语句的下面，实现代码块的上面添加代码清单 10-4 中的代码。

代码清单 10-4 私有方法

```

@interface EAGLView (PrivateMethods)
- (void)createFramebuffer;
- (void)deleteFramebuffer;
@end

```

我们稍后会实现这些方法。首先，我们要建立类的基础。我们需要使用 `synthesize` 语句合成我们添加到类接口中的属性。在 EAGLView.m 文件的实现代码块里面，添加代码清单 10-5 中的代码。

代码清单 10-5 合成属性

```

@synthesize context;
@synthesize framebufferWidth;
@synthesize framebufferHeight;
@synthesize defaultFramebuffer;

```

在 iOS 应用中支持 OpenGL 内容绘制的是 CAEGLayer 类。因为我们正使用 OpenGL 来渲染 3D 对象，因此我们需要对 CAEGLayer 类进行引用。在 `synthesize` 语句的后面添加代码清单 10-6 中建立的方法。

代码清单 10-6 OpenGL 应用需要的 layerClass 方法

```

+ (Class)layerClass {
    return [CAEGLayer class];
}

```

在建立与层相关的视图之前，我们必须修改我们想要使用的渲染属性。`drawableProperties` 属性可以让你设置渲染表面和内容的颜色格式。我们会在 `init` 方法中设置这个属性的值。复制

代码清单 10-7 中的方法到实现。

代码清单 10-7 initWithCoder 方法

```

- (id)initWithCoder:(NSCoder*)coder
{
    self = [super initWithCoder:coder];
    if (self) {
        CAEAGLLayer *eagllayer = (CAEAGLLayer *)self.layer;

        eagllayer.opaque = TRUE;
        eagllayer.drawableProperties = [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithBool:FALSE],
            kEAGLDrawablePropertyRetainedBacking,
            kEAGLColorFormatRGBA8,
            kEAGLDrawablePropertyColorFormat,
            nil];
    }

    return self;
}

```

粗体显示的代码就是我们为这个渲染设置 `drawableProperties` 属性的地方。想要获得关于 OpenGL 的颜色和格式选项的更多信息，请参考 *Pro OpenGL ES for iOS (Apress)*，具体网址为 www.apress.com/mobile/ios/9781430238409。

因为渲染表面是用 Core Animation 来呈现给用户的，因此所有施加的用来影响 3D 内容的效果和动画都会呈现给用户。苹果公司的文档建议遵守下面的最佳做法：

- 设置层的不透明属性为 `TRUE`。
- 设置层的边框与屏幕尺寸匹配。
- 确保层没有被变换。
- 避免在 `CAEAGLLayer` 对象上面绘制其他的层。其他的，非 OpenGL 内容，可能会对表现产生负面影响。
- 当在一个纵向显示器上绘制一个横向的内容的时候，你需要亲自旋转内容自身，而不是使用 CAEGL 进行变换。

现在我们获悉了一些基本原则和最佳做法，我们将创建 OpenGL 层。我们在 `interface` 文件中声明了一些方法。添加代码清单 10-8 中的代码到实现。

代码清单 10-8 setContext 方法

```

- (void)setContext:(EAGLContext *)newContext
{
    if (context != newContext) {
        [self deleteFramebuffer];

        context = newContext;

        [EAGLContext setCurrentContext:nil];
    }
}

```

这个方法会在父 `UIViewController` 类的 `viewDidLoad` 方法中被调用。我们将会使用这

个方法以在我们设置帧缓存之前建立 OpenGL 内容。复制代码清单 10-9 中的代码来建立帧缓存。

代码清单 10-9 createFramebuffer 方法

```
- (void)createFramebuffer
{
    if (context && !defaultFramebuffer) {
        [EAGLContext setCurrentContext:context];

        // 1
        if ([self respondsToSelector:@selector(setContentScaleFactor:)])
        {
            float screenScale = [UIScreen mainScreen].scale;

            self.contentScaleFactor = screenScale;
        }

        // 2
        glGenFramebuffers(1, &defaultFramebuffer);
        glBindFramebuffer(GL_FRAMEBUFFER, defaultFramebuffer);

        // 3
        glGenRenderbuffers(1, &colorRenderbuffer);
        glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);
        [context renderbufferStorage:GL_RENDERBUFFER fromDrawable:(CAEAGLLayer
        *)self.layer];
        glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_WIDTH,
        &framebufferWidth);
        glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_HEIGHT,
        &framebufferHeight);

        glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_RENDERBUFFER,
        colorRenderbuffer);

        // 4
        glGenRenderbuffers(1, &depthRenderbuffer);
        glBindRenderbuffer(GL_RENDERBUFFER, depthRenderbuffer);
        glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT16,
        framebufferWidth, framebufferHeight);

        glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER,
        depthRenderbuffer);

        if (glCheckFramebufferStatus(GL_FRAMEBUFFER) != GL_FRAMEBUFFER_COMPLETE)
            NSLog(@"Failed to make complete framebuffer object %x",
            glCheckFramebufferStatus(GL_FRAMEBUFFER));
    }
}
```

这个方法有点复杂。寻找粗体标注的注释行，我们将逐段讲解它们。标注为 1 的注释行用于处理 OpenGL 层的缩放。我们设置它大体上与屏幕的比例相匹配。第 2 段创建了默认的 Framebuffer 对象。第 3 段创建了颜色缓存以及为内容分配的缓存仓库（buffer storage）。最后一段，第 4 段，创建了深度缓存并附加它。

对于基于标记的增强现实应用来说，这些代码中的大部分都是相同的。通常你可以简单地在渲染层中用你自己的标记图像和矩阵替换掉以前的。

我们已经创建了处理帧缓存的方法。我们还需要创建一些方法来清理这些对象。添加代码清单 10-10 中的代码到实现。

代码清单 10-10 deleteFramebuffer (和 dealloc) 方法

```
- (void)deleteFramebuffer
{
    if (context) {
        [EAGLContext setCurrentContext:context];

        if (defaultFramebuffer) {
            glDeleteFramebuffers(1, &defaultFramebuffer);
            defaultFramebuffer = 0;
        }

        if (colorRenderbuffer) {
            glDeleteRenderbuffers(1, &colorRenderbuffer);
            colorRenderbuffer = 0;
        }
    }
}

- (void)dealloc
{
    [self deleteFramebuffer];
}
```

这些封装检查 context 来确定哪种类型的缓存应该删除，并在之后移除那个特定的缓存。我们还在 dealloc 中添加一个对这个方法的调用。最后，我们还需要一个能够绑定帧缓存到视图的方法。复制代码清单 10-11 中的代码。

代码清单 10-11 setFramebuffer 方法

```
- (void)setFramebuffer
{
    if (context) {
        [EAGLContext setCurrentContext:context];

        if (!defaultFramebuffer)
            [self createFramebuffer];

        glBindFramebuffer(GL_FRAMEBUFFER, defaultFramebuffer);

        glViewport(0, 0, framebufferWidth, framebufferHeight);
    }
}
```

你只需要注意这个方法最后两行。前一行绑定帧缓存到层。后一行设置层的位置到屏幕的左上部 (0,0)，并扩展了定义的帧缓存的宽和高。

这就是 EAGLView 类中所有我们需要修改的。我们要确保视图使用了新类。

在每一个故事板中，对于 MainStoryboard_iPhone.storyboard 和 MainStoryboard_iPad.storyboard，我们需要设置视图的 custom class 属性为 EAGLView。图 10-2 是对于这个过程的详细指导。

在左边的导航器中，我选中了故事板。之后展开了视图控制器并选中了它的默认视图。从右边的 Identity Inspector (标识检查器) 选项卡中，我们为视图选中了 EAGLView 类型。确保

你为两个故事板重复了这个过程。

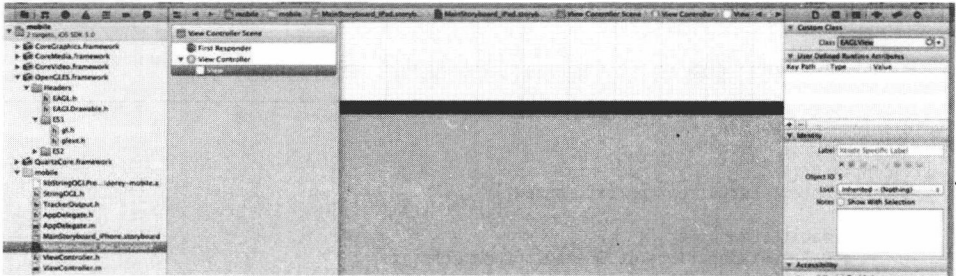


图 10-2 设置视图为我们的 EAGLView 类型

现在，已经设置好了视图，我们还要做一些必要的修改。

10.2.3 创建增强现实视图控制器

在 Xcode 中打开 ViewController.h 文件。首先，我们需要导入所需的头文件来建立视图。代码清单 10-12 显示的是这个类需要的导入语句。

代码清单 10-12 ViewController 的导入语句

```
#import "StringOGL.h"
#import <OpenGL/EAGL.h>
#import <OpenGL/ES1/gl.h>
#import <OpenGL/ES1/glex.h>
#import <OpenGL/ES2/gl.h>
#import <OpenGL/ES2/glex.h>
```

下一步，我们必须指定 ViewController 遵守 StringOGLDelegate 协议。在做这些的时候，我们还要同时定义实例变量。复制代码清单 10-13 中的代码到 interface 块。

代码清单 10-13 实例变量

```
@interface ViewController : UIViewController <StringOGLDelegate> {
    EAGLContext *context;
    StringOGL *stringOGL;
    float projectionMatrix[16];
    BOOL animating;
}
```

下一步，定义代码清单 10-14 中的方法。

代码清单 10-14 实例方法

```
- (void)startAnimation;
- (void)stopAnimation;
```

本章后面会解释这些方法能够做什么。切换到 ViewController.m 文件。我们需要添加的第一个方法可以为 3D 模型创建一个标准的投影矩阵。此方法会创建一个尽可能基本的投影矩阵。

它以 `glFrustum` 规范为蓝本。（`glFrustum` 描述了一个会产生立体投影的立体矩阵。）这是通过用 `glFrustum` 矩阵乘以当前矩阵并接着用结果矩阵替换当前矩阵来完成的。添加代码清单 10-15 中的代码到 `didReceiveMemoryWarning` 方法的前面。

代码清单 10-15 创建投影矩阵

```
- (void)createProjectionMatrix: (float *)matrix verticalFOV: (float)verticalFOV
aspectRatio: (float)aspectRatio nearClip: (float)nearClip farClip: (float)farClip
{
    memset(matrix, 0, sizeof(*matrix) * 16);

    float tan = tanf(verticalFOV * M_PI / 360.f);

    matrix[0] = 1.f / (tan * aspectRatio);
    matrix[5] = 1.f / tan;
    matrix[10] = (farClip + nearClip) / (nearClip - farClip);
    matrix[11] = -1.f;
    matrix[14] = (2.f * farClip * nearClip) / (nearClip - farClip);
}
```

我再次建议你参考 Apress 上关于 iOS 中的 OpenGL 编程的出版物《Pro OpenGL ES for iOS》来了解这个方法。但是，当你学习这个方法的时候，要记住深度缓存精度是受 `nearClip` 和 `farClip` 的值影响的。当 `farClip` 和 `nearClip` 增加的时候，在视图中相邻的表面之间的深度缓存将更难区分。`nearClip` 永远不能设置为 0，因为当 `nearClip` 接近 0 的时候乘数将接近无限大。

我们在头文件中定义了两个方法，接下来将要实现它们。但是，在继续之前，我们需要导入 `EAGLView` 头文件到 `ViewController.h` 文件中。之后，添加代码清单 10-16 中的代码到 `ViewController`。

代码清单 10-16 动画开始和停止方法

```
- (void)startAnimation
{
    if (!animating) {
        [stringOGL resume];
        animating = TRUE;
    }
}

- (void)stopAnimation
{
    if (animating) {
        [stringOGL pause];
        animating = FALSE;
    }
}
```

对于一个类来说，实现 `StringOGLDelegate` 协议只需要一个方法——`render` 方法。这个方法可以处理当 `String` 工具包识别出一个可能的标记时将会进行的动作。添加代码清单 10-17 中的方法到 `ViewController`。

代码清单 10-17 显示 3D 对象

```

- (void)render
{
    [(EAGLView *)self.view setFramebuffer];

    static const GLfloat squareVertices[] = {
        -0.33f, -0.33f,
        0.33f, -0.33f,
        -0.33f, 0.33f,
        0.33f, 0.33f,
    };

    static const GLubyte squareColors[] = {
        255, 255, 0, 255,
        0, 255, 255, 255,
        0, 0, 0, 0,
        255, 0, 255, 255,
    };

    const int maxMarkerCount = 10;

    struct MarkerInfoMatrixBased markerInfo[10];

    int markerCount = [stringOpenGL getMarkerInfoMatrixBased: markerInfo
maxMarkerCount: maxMarkerCount];
    for (int i = 0; i < markerCount; i++)
    {
        glMatrixMode(GL_PROJECTION);
        glLoadMatrixf(projectionMatrix);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glMultMatrixf(markerInfo[i].transform);

        glVertexPointer(2, GL_FLOAT, 0, squareVertices);
        glEnableClientState(GL_VERTEX_ARRAY);
        glColorPointer(4, GL_UNSIGNED_BYTE, 0, squareColors);
        glEnableClientState(GL_COLOR_ARRAY);

        glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
    }
}

```

该方法有几个目的。首先我们设置了渲染位置的顶点以及颜色。接下来，我们设置了 String 工具包识别的标记的个数的上限，此处设置为 10。我们创建了一个结构体，以在识别到标记时与 MarkerInfo 联系，之后我们遍历了 getMarkerInfoMatrixBased 方法的结果。这个方法会使用所有加载到工程中的图像，并试图在摄像头视图中识别它们。

循环中的是标准的 OpenGL 代码。这个例子会显示一个标准的 OpenGL 立方体，与我们在 iOS 4.3 模板中看到的类似。在更新 viewDidLoad 方法之前，修改类使其只在纵向模式下显示。按代码清单 10-18 中显示的代码更新 shouldAutorotateToInterfaceOrientation 方法。

代码清单 10-18 只允许纵向

```
return (interfaceOrientation == UIInterfaceOrientationPortrait);
```

我们快成功了。但是，在测试应用之前我们需要更新 viewDidLoad 方法。按代码清单 10-19 显示的代码更新 viewDidLoad 方法。

代码清单 10-19 更新 viewDidLoad 方法

```

- (void)viewDidLoad
{
    [super viewDidLoad];
    animating = NO;

    EAGLContext *aContext = [[EAGLContext alloc]
initWithAPI:kEAGLRenderingAPIOpenGLES1];

    if (!aContext)
        NSLog(@"Failed to create ES context");
    else if (![EAGLContext setCurrentContext:aContext])
        NSLog(@"Failed to set ES context current");

    context = aContext;

    EAGLView *eaglView = (EAGLView *)self.view;

    [(EAGLView *)self.view setContext:context];
    [(EAGLView *)self.view setFramebuffer];

    int viewport[4] = {0, 0, eaglView.framebufferWidth, eaglView.framebufferHeight};
    viewport[1] = (eaglView.framebufferHeight - viewport[3]) / 2;

    glViewport(viewport[0], viewport[1], viewport[2], viewport[3]);

    float aspectRatio = viewport[2] / (float)viewport[3];

    [self createProjectionMatrix: projectionMatrix verticalFOV: 47.22f aspectRatio:
aspectRatio nearClip: 0.1f farClip: 100.f];

    // Initialize String
    stringOGL = [[StringOGL alloc] initWithDelegate: self context: aContext
frameBuffer:[eaglView defaultFramebuffer] leftHanded: NO];

    [stringOGL setProjectionMatrix:projectionMatrix viewport:viewport orientation:[self
interfaceOrientation] reorientIPhoneSplash:YES];

    // Load image markers
    [stringOGL loadImageMarker: @"bookcover" ofType: @"png"];
}

```

这个方法的前半部分使用本章中创建的方法建立了 OpenGL 的 context。这段代码的粗体行专门用于 String 工具包。首先，我们在 delegate 为 self 的情况下初始化了 String，并创建了一个适当的帧缓存。接下来，我们建立了投影矩阵。最后，我们加载了标记。如果你有多个标记，可以在这里把它们都添加进来。

我们最后需要修改的是 viewWillAppear 和 viewWillDisappear 方法。按代码清单 10-20 显示的代码更新它们。

代码清单 10-20 开始和停止动画

```

- (void)viewWillAppear:(BOOL)animated
{
    [self startAnimation];

    [super viewWillAppear:animated];
}

```

```
- (void)viewWillDisappear:(BOOL)animated
{
    [self stopAnimation];
    [super viewWillDisappear:animated];
}
```

在调用 `resume` 方法之前，`String` 实际上不会开始运行。如果你还记得，我们在 `startAnimation` 封装中引用过这个方法。这就是我们将其放到 `viewWillAppear` 方法中的原因。

在一个物理设备上运行这个工程。因为我们把这个工程建立为一个通用程序，因此它在 iPad 或者 iPhone 上都能运行良好。确保你把一本书或者一个图像置于摄像头视图中。你会看到与图 10-3 类似的情景。

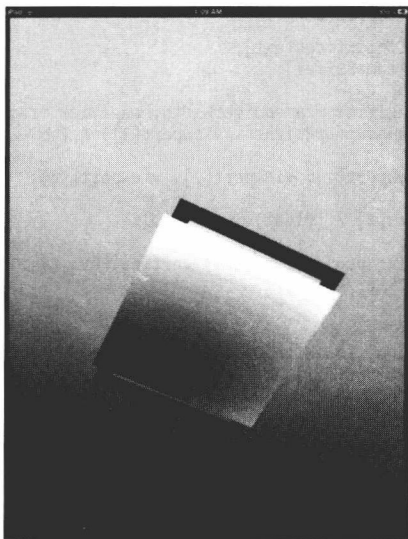


图 10-3 在 action 中查看增强现实标记应用

10.3 总结

在第 9 章，我们讨论了一些市场上存在的供开发者建立增强现实应用的工具包。在大部分情况下，对象识别算法是不用自己实现的，只要利用好各种工具库即可。这些工具包的价值是帮你处理基础工作并让你集中于应用的业务层或者功能。

本章以一个单一视图应用开始，并以 `String` 的增强现实工具包为基础建立了一个基于标记的增强现实应用。我们仅触及了 `OpenGL` 和 3D 模型的表面，但是这为更复杂的应用做好了准备。

第 11 章将会转型到社交媒体并建立一个基于地理坐标来显示社交数据的增强现实应用。

第⑪章

构建社交型的增强现实应用

在第 10 章，我们讨论了使用 String 工具包的基于标记的增强现实应用。由于在手持设备上识别标记需要较高的处理能力，因此基于位置的增强现实应用比基于标记的增强现实应用出现得更早。

本章将学习一个更传统的增强现实应用，建立一个与曾掀起 AppStore 和 Android 市场革命的那个开创性的应用接近的应用。

我们将建立一个开启了 GPS 的增强现实应用来寻找附近的 Facebook 位置。

11.1 快速设置

首先，确保为开发一个与 Facebook 的 Open Graph 相关的应用做好了准备。这个过程有几个步骤，大致如下：

- 创建一个 Facebook 应用
- 克隆 Facebook iOS 工具包的 GitHub 仓库
- 为应用实现独立登录

11.1.1 创建 Facebook 应用

创建 Facebook 应用，并给出 API 以及用户权限和数据的上下文。如果没有应用，通过 Open Graph API 检索就会没有范围。Facebook 的应用允许用户决定给予你的应用什么类型的权限。

请访问 <https://developers.facebook.com/apps>。如果你以前用 Facebook API 做过开发，你应该已经为创建一个新的 Facebook 应用做好了所有的准备。在开发者控制面板的右上部有一个叫做 Create New App（创建新应用）的按钮。单击这个按钮并按照操作指南建立你的应用。我已经命名我的应用为 kyleroché。在建立的时候，你应该会在应用控制面板中看到类似图 11-1 的情景。

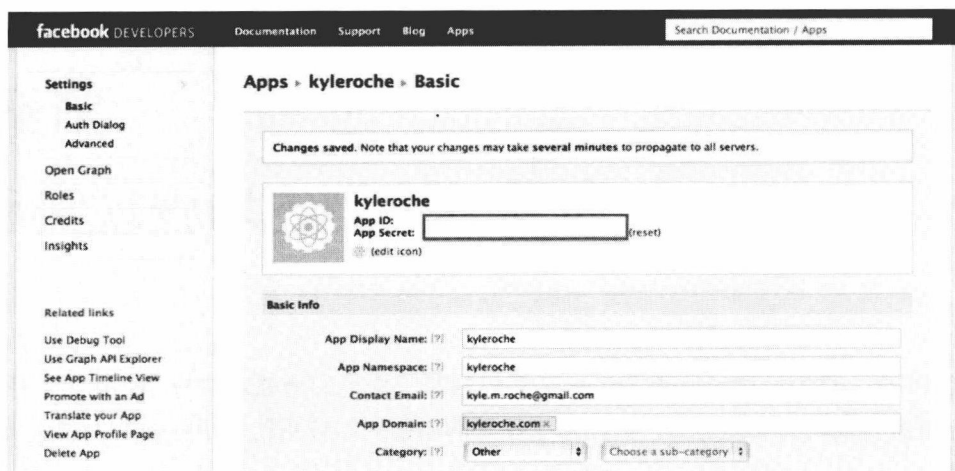


图 11-1 建立 Facebook 应用

请记住你的 App ID 和 App Secret（密码）。当你建立 Xcode 工程的时候将会需要 App ID。

11.1.2 克隆 Facebook iOS SDK

访问 <https://github.com/facebook/facebook-ios-sdk> 下载 Facebook iOS SDK 的最新版本。克隆该仓库到你的本地电脑。如果你有兴趣的话，你可以运行其中的一些例子。在 sample 子目录中打开 Xcode 工程。在 Xcode 中打开 DemoAppViewController.m 文件，并找到代码清单 11-1 显示的那个代码段。

代码清单 11-1 设置 Facebook 的 App ID

```
// 在运行这个应用之前，必须要设置你的 Facebook 应用 Id
// 请访问 http://www.facebook.com/developers/createapp.php
// 同样，你的应用必须要绑定到 fb[app_id]:// URL 策略（用你的真实 Facebook 应用 id 代替 [app_id]）
static NSString* kAppId = nil;
```

确保你修改了 nil 为你的 Facebook App ID。在对 SDK 提供的演示应用做测试之前你还需要完成一个步骤。打开你的应用的 .plist 文件并找到 URL Schemes 项。这个数组的第 0 项需要一个 fb 的值 [你的 App id] 才能正常运行。当确认来自一个本地 iOS 应用后，Facebook 使用 URL Scheme 来证实本地应用的真实性。在 iOS 中，URL Scheme 是用来在上下文中启动其他本地应用的。前缀（现在是你的 App ID）是唯一用来识别该应用的。例如，如果我的 App ID 是 123456，我会设置我的 URL Scheme 为 fb123456，在验证身份后 Facebook 会核实本地应用是否绑定到了 URL Scheme: fb123456://[action]。

此时，应用还不能工作，除非你在所在设备上安装了 Facebook 应用。

11.2 词汇表

到目前为止，我们已经建立了基于标记的增强现实和游戏场景中的增强现实的应用例子。我们还没有讨论过与基于位置的增强现实之间的一些不同。本节将会讨论一些在这个例子中使用的新术语。

11.2.1 方位角

Azimuth 是出自阿拉伯语的一个单词，其字面意思是方位角。Azimuth 是球面坐标系统的一个角度度量，例如用于空间定位计算中。方位角是在一个定义明确的参考平面上的投影矢量与参照矢量之间的角度。例如，假设在一个基于位置的应用里面有一个目标点，如果想要知道我们需要面向哪一个方向才能对准这个目标点，可以计算一下那一点相对于北极的方位角。我们绘制（象征性的）一个从我们当前位置到北极的垂直面，并从目标点穿过海平面的垂直面（或者在我们的位置之下的一个等价平面），介于这两个平面之间的角度就是方位角。

俗话说，“一张图片胜过千言万语”。参见图 11-2 以获得更详细的说明。

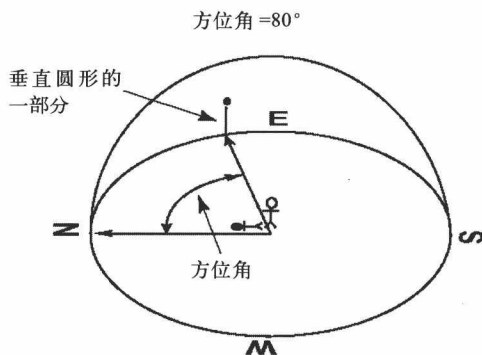


图 11-2 计算方位角（目标点和北极的方位角是 80° ）

在这个例子中，我们将会建立并讨论用来计算从视点到另一个 GPS 坐标的视点之间的方位角的公式。但是，如果你就是等不了这么久，那么给出一个例子：如果我们站在纬度为 ϕ_1 ，经度为 0 的位置，并且我们想要计算从我们的视点到纬度为 ϕ_2 ，经度为 L （正东）的点 2 的视点之间的方位角，那么图 11-3 中显示的就是计算公式。

$$\Lambda = (1 - e^2) \frac{\tan \phi_2}{\tan \phi_1} + e^2 \sqrt{\frac{1 + (1 - e^2)(\tan \phi_2)^2}{1 + (1 - e^2)(\tan \phi_1)^2}}$$

图 11-3 计算方位角的公式

11.2.2 矫正方向

我们实际上已经在本书中讨论过这个概念。我不确定是否有这方面的一个标准术语，但是

请回顾一下第 4 章关于磁力计的讨论，其中我们讨论过方向改变。假设有一个以纵向模式指向北极的 iPhone。如果你把这个 iPhone 转向一侧，变成是垂直横向模式（打个比方），此时你仍然面向北，但是 iPhone 的指向可能会误以为你的方向已变为西。

当在增强现实的应用中方向发生变化的时候（尤其是基于位置信息建立的应用中），你必须注意对你的指向做相应的调节。

11.3 构建应用

我们讲解。打开 Xcode 并使用 Empty Application（空白应用）模板创建一个名为 Ch11 的工程。这个模板会使用一个单独的窗口建立一个简单的应用。

11.3.1 致谢

在 github.com、stackoverflow.com、Apple 的开发者论坛以及其他的各种在线社区中有很多关于基于位置的增强现实的成果。有与 ARKit (<https://github.com/zac/iphonemarkit>) 类似的各种工具包，这个工具包从贡献者开始分出了几乎 100 个有不同的变化分支。因此，我想要感谢 Zac White 和 Niels Hansen，具体来说感谢他们为我们提供了一个创造的起点。

11.3.2 所需框架

我们将建立一个基于位置的应用。如你所料，我们需要添加 CoreLocation 框架到工程。在你继续创建之前，请确保你已经包含了该框架。

11.3.3 添加 Facebook iOS SDK

在之前下载 SDK 的时候，有一个示例应用程序。因为我们现在正以一个干净的应用开始，我们需要把 SDK 添加回这个工程。

在 Finder 中打开归档的 SDK 目录。从 SDK 中拖拽 src 目录到 Xcode 工程。确保你选中了复制资源到工程目录的选项。本章后面会再次讲解这一点。

11.3.4 开始编码

好的，设置齐备，让我们编码吧。在 Xcode 工程中创建一个叫做 ARController 的组。我们将会分离增强现实相关的代码，以便之后在其他工程中可以重用它。

在 ARController 组中创建一个叫做 ARController.m 的新类。确保这个类是 NSObject 的子类而不是 UIViewController 的子类。在 Xcode 中打开 ARController.h 文件。按代码清单 11-2 显示的代码更新 ARController.h 文件。

代码清单 11-2 更新 ARController.h

```

#import <Foundation/Foundation.h>
#import <CoreLocation/CoreLocation.h>
#import <UIKit/UIKit.h>

@interface ARController : NSObject <UIAccelerometerDelegate, CLLocationManagerDelegate>
{
}

@property (nonatomic, retain) UIViewController *rootViewController;
@property (nonatomic, retain) UIImagePickerController *pickerController;
@property (nonatomic, retain) UIView *hudView;
@property (nonatomic, retain) CLLocationManager *locationManager;
@property (nonatomic, retain) UIAccelerometer *accelerometer;

- (id)initWithViewController:(UIViewController *)viewController;
- (void)presentModalARControllerAnimated:(BOOL)animated;

@end

```

我们正建立一些属性，以备后用。我将会在实现文件中详细解释每一个属性。切换到 ARController.m 文件并添加代码清单 11-3 中的代码。

代码清单 11-3 ARController.m

```

#import "ARController.h"

@implementation ARController
@synthesize rootViewController = _rootViewController;
@synthesize pickerController = _pickerController;

@synthesize hudView = _hudView;
@synthesize locationManager = _locationManager;
@synthesize accelerometer = _accelerometer;

- (id)initWithViewController:(UIViewController *)viewController {
    self.rootViewController = viewController;
    CGRect screenBounds = [[UIScreen mainScreen] bounds];
    self.hudView = [[UIView alloc] initWithFrame:screenBounds];
    self.rootViewController.view = self.hudView;

    self.pickerController = [[[UIImagePickerController alloc] init] autorelease];
    self.pickerController.sourceType = UIImagePickerControllerSourceTypeCamera;
    self.pickerController.cameraViewTransform = CGAffineTransformScale(
self.pickerController.cameraViewTransform, 1.13f, 1.13f);

    self.pickerController.showsCameraControls = NO;
    self.pickerController.navigationBarHidden = YES;
    self.pickerController.cameraOverlayView = _hudView;

    self.locationManager = [[CLLocationManager alloc] init];
    self.locationManager.headingFilter = kCLLocationHeadingFilterNone;
    self.locationManager.desiredAccuracy = kCLLocationAccuracyBest;
    self.locationManager.delegate = self;
    [self.locationManager startUpdatingHeading];
    [self.locationManager startUpdatingLocation];

    _accelerometer = [UIAccelerometer sharedAccelerometer];
    _accelerometer.updateInterval = 0.25;
    _accelerometer.delegate = self;

    [[NSNotificationCenter defaultCenter] addObserver:self

```

```

selector:@selector(deviceOrientationDidChange:)
name:UIDeviceOrientationDidChangeNotification object:nil];
    [[UIDevice currentDevice] beginGeneratingDeviceOrientationNotifications];

    return self;
}

- (void)presentModalARControllerAnimated:(BOOL)animated {
    [self.rootViewController presentModalViewController:[self pickerController]
    animated:animated];
    _hudView.frame = _pickerController.view.bounds;
}

- (void)deviceOrientationDidChange:(NSNotification *)notification {
}

@end

```

我们先逐步讲解下这一段。首先，我们需要使用 `synthesize` 语句合成实现文件中定义的属性。迄今为止，这个类中有 3 个方法。第一个方法 `initWithViewController` 是用来向视图添加一个 `UIImagePickerController` 的（参见第 7 章相应内容），之后在其上覆盖一个新类型的 HUD 层。

我们接着建立了位置管理器和加速计并开启了 `update` 服务。这与第 4 章测试 iOS 传感器编程的代码非常相似。单凭这个类并不是那么有用。我们要建立一些会产生影响的新类。

使用 `UIViewController` 模板在 Xcode 中创建一个叫做 `RootViewController` 的新文件。确保你没有选中 `With XIB for user interface` 选项。在 Xcode 中打开 `RootViewController.h` 文件并按代码清单 11-4 做相应的修改。

代码清单 11-4 更新 `RootViewController.h` 文件

```

#import <UIKit/UIKit.h>
@class ARController;

@interface RootViewController : UIViewController {
}

@property (nonatomic, retain) ARController *arController;
@end

```

我们引用了 `ARController` 类，并建立了一个相同类型的新属性。在 Xcode 中切换到 `RootViewController.m` 文件。按代码清单 11-5 显示的代码更新这个类。

代码清单 11-5 更新 `RootViewController.m` 文件

```

#import "RootViewController.h"
#import "ARController.h"

@implementation RootViewController
@synthesize arController = _arController;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

```

```

- (void)didReceiveMemoryWarning
{
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

#pragma mark - View lifecycle

// Implement loadView to create a view hierarchy programmatically, without using a nib.
- (void)loadView
{
    _arController = [[ARController alloc] initWithViewController:self];
}

- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
    [self.arController presentModalARControllerAnimated:NO];
}

- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
}

/*
// Implement viewDidLoad to do additional setup after loading the view, typically from a
nib.
- (void)viewDidLoad
{
    [super viewDidLoad];
}
*/

- (void)viewDidUnload
{
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientatio
n
{
    // Return YES for supported orientations
    return YES;
}

@end

```

这里我们做了一些小修改。在 `viewDidAppear` 方法中，我们显示了模态增强现实控制器，这允许我们在其他的应用中重用已建立的这些代码，并且不需要重建基础的 `UIViewController`。在 `loadView` 方法中，我们使用 `UIImagePickerController` 和 HUD 层创建了新 `arController` 实例。

在建立起用户界面下带有一个 `UIImagePickerController`（摄像头视图）的基础工程之前，我们还有一个步骤要做。在 Xcode 中打开 `AppDelegate.h` 文件。按代码清单 11-6 显示的代码更新头文件。

代码清单 11-6 更新 AppDelegate.h 文件

```
#import <UIKit/UIKit.h>

@class RootViewController;
@interface AppDelegate : UIResponder <UIApplicationDelegate> {

}

@property (strong, nonatomic) UIWindow *window;
@property (nonatomic, retain) RootViewController *rootViewController;
@end
```

在 Xcode 中打开 AppDelegate.m 文件。我们将会重写 didFinishLaunchingWithOptions 方法以便能够添加 RootViewController 到视图堆中。按代码清单 11-7 中的相应标记更新 AppDelegate.m 文件。

代码清单 11-7 更新 AppDelegate.m 文件

```
#import "AppDelegate.h"
#import "RootViewController.h"

@implementation AppDelegate

@synthesize window = _window;
@synthesize rootViewController = _rootViewController;

- (void)dealloc
{
    [_window release];
    [super dealloc];
}

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[[UIWindow alloc] initWithFrame:[UIScreen mainScreen] bounds]]
autorelease];
    // Override point for customization after application launch.
    _rootViewController = [[RootViewController alloc] init];
    [_window addSubview:_rootViewController.view];

    self.window.backgroundColor = [UIColor clearColor];
    [self.window makeKeyAndVisible];
    return YES;
}
```

同样，一切都很熟悉。我们导入了 RootViewController 类，合成了变量，并在之后添加它作为 UIWindow 的一个子视图。

在一个支持 Core Location (iPhone 或者 iPad) 的物理设备上运行这个应用。因为我们将使用摄像头，因此不能用模拟器来调试。位置管理器会询问你是否允许这个应用获取当前位置，如图 11-4 所示。

在你接受这个请求后，你会看到一个全屏的摄像头视图，如图 11-5 所示。



图 11-4 通过位置管理权限来允许位置分享



图 11-5 允许位置分享后来到了摄像头视图

11.3.5 监听传感器更新

我们已经开启了方向和位置的更新，但是我们还没有对其做任何设置。我们也没有响应设备方向的变化。第 7 章讨论了弧度和角度的转化，以及在屏幕上计算角度时这是多么有用。因此，我们要建立一些便利方法。打开 Ch11_Prefix.pch 文件并添加代码清单 11-8 中的代码。

代码清单 11-8 角度和弧度的互相转换

```
#define degreesToRadians(x) (M_PI * (x) / 180.0)
#define radiansToDegrees(x) ((x) * 180.0/M_PI)
```

完成这个后，我们还需要定义一些变量。打开 ARController.h 文件并添加代码清单 11-9 中的变量。

代码清单 11-9 更新 ARController.h 文件

```
@property (nonatomic) UIDeviceOrientation deviceOrientation;
@property (nonatomic) double range;
```

切换到 ARController.m 文件并按代码清单 11-10 显示的合成变量。

代码清单 11-10 合成属性

```
@synthesize deviceOrientation = _deviceOrientation;
@synthesize range = _range;
```

我们现在有一个存储当前设备方向和视图范围的地方了。变量 `range` 将用于保存屏幕对着的视图的角度。图 11-6 显示的是一个计算对角的例子。在点 A 处有一个大树对着的 25 度角。我们将会计算一个可接受的，对着整个屏幕宽度的视图，假设是一个接近 15 度的视图窗口。

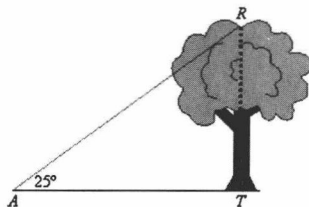


图 11-6 用一个对角计算可接受的视图

在 `ARController.m` 文件中按代码清单 11-11 中显示的代码更新 `deviceOrientationDidChange` 方法。

代码清单 11-11 新版 `deviceOrientationDidChange` 方法

```
- (void)deviceOrientationDidChange:(NSNotification *)notification {
    UIDeviceOrientation orientation = [[UIDevice currentDevice] orientation];
    UIApplication *app = [UIApplication sharedApplication];

    if ( orientation != UIDeviceOrientationUnknown &&
        orientation != UIDeviceOrientationFaceUp &&
        orientation != UIDeviceOrientationFaceDown) {

        CGAffineTransform transform =
CGAffineTransformMakeRotation(degreesToRadians(0));
        CGRect bounds = [[UIScreen mainScreen] bounds];
        [app setStatusBarHidden:YES];
        [app setStatusBarOrientation:UIInterfaceOrientationPortrait animated:
NO];

        if (orientation == UIDeviceOrientationLandscapeLeft) {
            transform
CGAffineTransformMakeRotation(degreesToRadians(90));
            bounds.size.width = [[UIScreen mainScreen] bounds].size.height;
            bounds.size.height = [[UIScreen mainScreen] bounds].size.width;
            [app
setStatusBarOrientation:UIInterfaceOrientationLandscapeRight animated: NO];
        } else if (orientation == UIDeviceOrientationLandscapeRight) {
            transform
CGAffineTransformMakeRotation(degreesToRadians(-90));
            bounds.size.width = [[UIScreen mainScreen] bounds].size.height;
            bounds.size.height = [[UIScreen mainScreen] bounds].size.width;
            [app setStatusBarOrientation:UIInterfaceOrientationLandscapeLeft
animated: NO];
        } else if (orientation == UIDeviceOrientationPortraitUpsideDown) {
            transform =
CGAffineTransformMakeRotation(degreesToRadians(180));
            [app
setStatusBarOrientation:UIInterfaceOrientationPortraitUpsideDown animated: NO];
        }
        _hudView.transform = transform;
        _hudView.bounds = bounds;
        _range = _hudView.bounds.size.width / 12;
    }
}
```



```

    }
    _deviceOrientation = orientation;
}

```

我们从头开始逐步讲解这段代码。首先，我们声明了几个在全局应用中可以引用的变量。我们建立了一个对于设备方向的引用，还有一个对 `shareApplication` 单例的引用。

我们接着使用在 `prefix` 文件中定义的转换方法来转换角度为弧度，并变换屏幕以使旋转不会正面朝上或者朝下（我们忽略了未知的方向）。

最后，我们对 HUD 层施加了变换，并且设置了屏幕的范围。请记住，虽然用于范围的计算式可能看似静态，但它实际上根据设备方向的不同而有一个不同的分子（`numerator`）。

返回到 `ARController.h` 文件并再建立一些变量来处理我们的位置和方向的更新。添加代码清单 11-12 中显示的属性。

代码清单 11-12 用于方向和位置的属性

```

@property (nonatomic, retain) CLLocation *deviceLocation;
@property (nonatomic, retain) CLLocation *deviceHeading;

```

合成代码清单 11-13 中的属性。

代码清单 11-13 合成属性

```

@synthesize deviceHeading = _deviceHeading;
@synthesize deviceLocation = _deviceLocation;

```

我们将会在后面的例子中使用这些变量。

11.3.6 存储坐标

我们几乎为位置、屏幕视图范围，以及设备方向的保存做好了所有的准备。但是我们还没有定义一个用于保存目标点坐标的地方。这一节会有一些新的内容，但是基于 Alasdair Allan 的研究成果（其研究成果又是基于 Zac White 的 ARKit 的工作）。

创建一个继承自 `NSObject` 的名为 `ARCoordinate` 的新类。打开头文件并添加代码清单 11-14 中的代码。

代码清单 11-14 `ARCoordinate.h`

```

#import <Foundation/Foundation.h>

@interface ARCoordinate : NSObject {
}

@property (nonatomic, retain) NSString *name;
@property (nonatomic, retain) NSString *place;
@property (nonatomic) double distance;
@property (nonatomic) double inclination;
@property (nonatomic) double azimuth;

```

```
- (id)initWithRadialDistance:(double)distance inclination:(double)inclination
azimuth:(double)azimuth;
@end
```

我们最终会被社交网络中的一些 Facebook places 吸引。我们将会坐标之上显示人名和 Facebook place 名。其他的属性用来存储距离、倾角以及我们正在跟踪的位置的方位角。

我们定义的方法会用指定的属性创建一个新的坐标实例。

切换到 ARCoordinate.m 文件，让我们完成这个类的实现。添加代码清单 11-15 中的代码到 ARCoordinate.m 文件。

代码清单 11-15 ARCoordinate.m

```
#import "ARCoordinate.h"

@implementation ARCoordinate

@synthesize name = _name;
@synthesize place = _place;
@synthesize distance = _distance;
@synthesize inclination = _inclination;
@synthesize azimuth = _azimuth;

- (id)initWithRadialDistance:(double)distance inclination:(double)inclination
azimuth:(double)azimuth {
    if (self = [super init]) {
        _distance = distance;

        _inclination = inclination;
        _azimuth = azimuth;
    }
    return self;
}

- (void)dealloc {
    [_name release];
    name = nil;
    [_place release];
    _place = nil;
}

@end
```

上述代码也似曾相识。我们简单地建立了类来检索和保存坐标的属性。我们需要另一个继承自 ARCoordinate 的类来添加位置信息。

创建一个叫做 ARGeoCoordinate 的新类。在 Xcode 中打开这个新类，并添加代码清单 11-16 显示的代码。

代码清单 11-16 ARGeoCoordinate.h

```
#import <Foundation/Foundation.h>
#import <CoreLocation/CoreLocation.h>
#import "ARCoordinate.h"

@interface ARGeoCoordinate : ARCoordinate {

}

@property (nonatomic, retain) CLLocation *geoLocation;
```

```

- (id)initWithCoordinate:(CLLocation *)location name:(NSString *)name place:(NSString
*)place;
- (id)initWithCoordinateAndOrigin:(CLLocation *)location name:(NSString *)name
place:(NSString *)place origin:(CLLocation *)origin;
- (float)angleFromCoordinate:(CLLocationCoordinate2D)first
second:(CLLocationCoordinate2D)second;
- (void)calibrateUsingOrigin:(CLLocation *)origin;
@end

```

我们导入 CoreLocation 框架，以及我们刚刚创建的 ARCoordinate 类。之后修改父类属性 NSObject 为 ARCoordinate。

最后，我们声明了一个用于保存目标点的 CLLocation 的属性，并声明了一些我们将会稍后讨论的方法。切换到 ARGeoCoordinate.m 文件并添加代码清单 11-17 中的代码。

代码清单 11-17 ARGeoCoordinate.m

```

#import "ARGeoCoordinate.h"

@implementation ARGeoCoordinate
@synthesize geoLocation = _geoLocation;

- (id)initWithCoordinate:(CLLocation *)location name:(NSString *)name place:(NSString
*)place {
    if (self = [super init]) {
        self.geoLocation = location;
        // Properties of base class
        self.name = name;
        self.place = place;
    }
    return self;
}

- (id)initWithCoordinateAndOrigin:(CLLocation *)location name:(NSString *)name
place:(NSString *)place origin:(CLLocation *)origin {
    if (self = [super init]) {
        self.geoLocation = location;
        // Properties of base class
        self.name = name;
        self.place = place;
        [self calibrateUsingOrigin:origin];
    }
    return self;
}

- (float)angleFromCoordinate:(CLLocationCoordinate2D)first
second:(CLLocationCoordinate2D)second {
    float longDiff = second.longitude - first.longitude;
    float latDiff = second.latitude - first.latitude;
    float aprxAzimuth = (M_PI * 5f) - atan(latDiff / longDiff);

    if (longDiff > 0) {
        return aprxAzimuth;
    } else if (longDiff < 0) {
        return aprxAzimuth + M_PI;
    } else if (latDiff < 0) {
        return M_PI;
    }

    return 0.0f;
}

- (void)calibrateUsingOrigin:(CLLocation *)origin {

```

```

        double baseDistance = [origin distanceFromLocation:_geolocation];
        self.distance = sqrt(pow([origin altitude] - [_geolocation altitude], 2) +
pow(baseDistance, 2));
        float angle = sin(ABS([origin altitude] - [_geolocation altitude]) / self.distance);

        if ([origin altitude] > [_geolocation altitude]) {
            angle = -angle;
        }

        self.inclination = angle;
        self.azimuth = [self angleFromCoordinate:[origin coordinate] second:[_geolocation
coordinate]];
    }

@end

```

我们浏览一下刚刚添加的代码。首先，我们合成了刚刚声明的属性。接着，我们创建了一个叫做 `initWithCoordinate` 的方法。这个方法带来了坐标的名字和位置并设置了基类 `ARCoordinate` 的属性，以便我们能够在后面的可视化层使用它们。

我们接着创建了一个叫做 `initWithCoordinateAndOrigin` 的方法。这与 `initWithCoordinate` 方法的目的相同，但还允许你定义一个原点。主要的不同点是这个方法会调用一个我们接下来将会讨论的新的内部方法。

方法 `calibrateUsingOrigin` 获取介于两个点之间的距离（设备和目标点），并计算原距离、角度（考虑属性的变化）以及方位角。这些属性被设置在基类中，以便在之后的例子中我们能够引用它们。

最后，我们创建了一个叫做 `angleFromCoordinate` 的方法。这个方法计算介于两点之间的角度的距离。

返回到 `ARController.h` 文件并在 `import` 语句的后面插入代码清单 11-18 中的代码。

代码清单 11-18 ARController.h 文件

```

@class ARCoordinate;
@class ARGeoCoordinate;

```

接下来，按代码清单 11-19 显示的为 `ARCoordinate` 声明的一个属性。

代码清单 11-19 ARCoordinate 属性

```

@property (nonatomic, retain) ARCoordinate *coordinate;

```

切换到 `ARController.m` 文件并合成新属性。截止到现在，本章一直在为私有变量使用相同的下划线样式。按代码清单 11-20 显示的代码调整 `initWithViewController` 方法。这个代码在方法的底部。

代码清单 11-20 更新 `initWithViewController` 方法

```

.....
_coordinate = [[ARCoordinate alloc] initWithRadialDistance:1.0 inclination:0 azimuth:0];
return self;
}

```

添加代码清单 11-21 中的代码到 ARController 实现文件顶部、import 语句的下面。

代码清单 11-21 ARController 的导入语句和私有方法

```
#import "ARCoordinate.h"
#import "ARGeoCoordinate.h"

@interface ARController (Private)
- (void)updateCurrentLocation:(CLLocation *)newLocation;
- (void)updateLocations;
- (void)updateCurrentCoordinate;
@end
```

在代码清单 11-21 中，我们声明了一个新的私有方法用于更新位置数组。同时我们导入了之前创建的 ARGeoCoordinate 和 ARCoordinate 类。添加代码清单 11-22 中的方法到实现文件。

代码清单 11-22 位置和方向方法

```
- (void)updateCurrentLocation:(CLLocation *)newLocation {
    self.deviceLocation = newLocation;

    for (ARGeoCoordinate *geoLocation in _coordinates) {
        if ([geoLocation isKindOfClass:[ARGeoCoordinate class]]) {
            [geoLocation calibrateUsingOrigin:self.deviceLocation];
        }
    }
}

- (void)updateCurrentCoordinate {
    double adjustment = 0;
    if (_deviceOrientation == UIDeviceOrientationLandscapeLeft)
        adjustment = degreesToRadians(270);
    else if (_deviceOrientation == UIDeviceOrientationLandscapeRight)
        adjustment = degreesToRadians(90);
    else if (_deviceOrientation == UIDeviceOrientationPortraitUpsideDown)
        adjustment = degreesToRadians(180);

    _coordinate.azimuth =
        degreesToRadians(_deviceHeading.magneticHeading) - adjustment;

    [self updateLocations];
}

- (void)updateLocations {
    // we'll get to this one later
}
```

稍后我们将会解释这些方法以及它们会做什么。我们使用了一些还没有声明的变量。现在，我们对这个做一些处理。切换到 ARController.h 文件。按代码清单 11-23 显示的代码更新头文件。

代码清单 11-23 更新 ARController.h

```
#import <Foundation/Foundation.h>
#import <CoreLocation/CoreLocation.h>
```

```

#import <UIKit/UIKit.h>

@class ARCoordinate;
@class ARGeoCoordinate;

@interface ARController : NSObject <UIAccelerometerDelegate, CLLocationManagerDelegate>
{
    NSMutableArray *_coordinates;
}

@property (nonatomic, retain) UIViewController *rootViewController;
@property (nonatomic, retain) UIImagePickerController *pickerController;
@property (nonatomic, retain) UIView *hudView;

@property (nonatomic, retain) CLLocationManager *locationManager;
@property (nonatomic, retain) UIAccelerometer *accelerometer;

@property (nonatomic) UIDeviceOrientation deviceOrientation;
@property (nonatomic) double range;

@property (nonatomic, retain) CLLocation *deviceLocation;
@property (nonatomic, retain) CLHeading *deviceHeading;

@property (nonatomic, retain) ARCoordinate *coordinate;
@property (nonatomic) double viewAngle;

- (id)initWithViewController:(UIViewController *)viewController;
- (void)presentModalARControllerAnimated:(BOOL)animated;

- (void)addCoordinate:(ARCoordinate *)coordinate animated:(BOOL)animated;
- (void)removeCoordinate:(ARCoordinate *)coordinate animated:(BOOL)animated;

@end

```

粗体显示的代码是新的。切换到 ARController.m，合成属性并创建新方法。添加代码清单 11-24 中的代码。

代码清单 11-24 更新 ARController.m 文件

```

// other synthesize statements
@synthesize viewAngle = _viewAngle;

// other methods
- (void)addCoordinate:(ARCoordinate *)coordinate animated:(BOOL)animated {
    [_coordinates addObject:coordinate];
}

- (void)removeCoordinate:(ARCoordinate *)coordinate animated:(BOOL)animated {
    [_coordinates removeObject:coordinate];
}

```

现在来说，这些只是方法存根。稍后我们会实现它们。如果你看一下我们目前所有的代码，你会发现 updateLocations 方法仍然没有任何作为。在实现这个方法之前，我们需要实现一个基于 UIView 的类来在 ARView 之上显示我们的每个坐标。

创建一个继承自 UIView 的 Objective-C 新类。命名新类为 ARAnnotation。打开 ARAnnotation.h 并按代码清单 11-25 显示的代码更新它。

代码清单 11-25 ARAnnotation.h

```
#import <UIKit/UIKit.h>

@class ARCoordinate;
@interface ARAnnotation : UIView {

}

- (id)initWithCoordinate:(ARCoordinate *)coordinate;

@end
```

切换到 ARAnnotation.m 文件。移除这个文件内的所有代码，并用代码清单 11-26 显示的代码代替它。

代码清单 11-26 ARAnnotation.m

```
#import "ARAnnotation.h"
#import "ARCoordinate.h"

#define ANNOTATION_WIDTH 150
#define ANNOTATION_HEIGHT 100

@implementation ARAnnotation

- (id)initWithCoordinate:(ARCoordinate *)coordinate
{
    CGRect annotationFrame = CGRectMake(0, 0, ANNOTATION_WIDTH, ANNOTATION_HEIGHT);

    if (self = [super initWithFrame:annotationFrame]) {
        UILabel *nameLabel = [[UILabel alloc] initWithFrame:CGRectMake(0, 0,
            ANNOTATION_WIDTH, 20.0)];
        nameLabel.backgroundColor = [UIColor whiteColor];
        nameLabel.textAlignment = UITextAlignmentCenter;
        nameLabel.text = coordinate.name;
        [nameLabel sizeToFit];
        [nameLabel setFrame:CGRectMake(0, 0, nameLabel.bounds.size.width + 8.0,
            nameLabel.bounds.size.height + 8.0)];
        [self addSubview:nameLabel];

        UILabel *placeLabel = [[UILabel alloc] initWithFrame:CGRectMake(25, 0,
            ANNOTATION_WIDTH, 20.0)];
        placeLabel.backgroundColor = [UIColor whiteColor];
        placeLabel.textAlignment = UITextAlignmentCenter;
        placeLabel.text = coordinate.place;
        [placeLabel sizeToFit];
        [placeLabel setFrame:CGRectMake(25, 0, placeLabel.bounds.size.width + 8.0,
            placeLabel.bounds.size.height + 8.0)];
        [self addSubview:placeLabel];
    }
    return self;
}

@end
```

相对于我们刚刚复制的代码量，这里并没有太多变化。我们简单地创建了两个标签（一个用于名字，一个用于位置）并且把它们放在一个 150×100 像素的矩形中。创建后的标签会放置在 HUD 层的每个坐标上面。在 Xcode 中打开 ARCoordinate.h 文件，并添加代码清单 11-27 中的代码。

代码清单 11-27 更新 ARCoordinate.h 文件

```
// Declare the class
@class ARAnnotation;
// create this property
@property (nonatomic, retain) ARAnnotation *annotation;
```

在实现文件中，用私有实例变量合成属性，并确保在 dealloc 方法中释放了它。

切换到 ARController.m 文件。导入 ARAnnotation.h 头文件。按代码清单 11-28 显示的代码更新 addCoordinate 方法。

代码清单 11-28 更新 addCoordinate 方法

```
- (void)addCoordinate:(ARCoordinate *)coordinate animated:(BOOL)animated {
    ARAnnotation *annotation = [[ARAnnotation alloc] initWithCoordinate:coordinate];
    coordinate.annotation = annotation;
    [annotation release];

    [_coordinates addObject:coordinate];
}
```

这个更新创建了一个新的 ARAnnotation 实例以用于坐标。在完成 updateLocations 方法之前，我们还有一些方法要创建。复制代码清单 11-29 中的声明到 ARController.m 文件顶部的私有类接口区域。

代码清单 11-29 新版私有方法

```
- (BOOL)viewportContainsCoordinate:(ARCoordinate *)coordinate;
- (double)deltaAzimuthForCoordinate:(ARCoordinate *)coordinate;
- (CGPoint)pointForCoordinate:(ARCoordinate *)coordinate;
- (BOOL)isNorthForCoordinate:(ARCoordinate *)coordinate;
```

我们来一次实现这些方法并讲解一下它们完成了什么。首先，复制代码清单 11-30 中的方法。

代码清单 11-30 viewportContainsCoordinate 方法

```
- (BOOL)viewportContainsCoordinate:(ARCoordinate *)coordinate {
    double deltaAzimuth = [self deltaAzimuthForCoordinate:coordinate];
    BOOL result = NO;
    if (deltaAzimuth <= degreesToRadians(_range)) {
        result = YES;
    }

    return result;
}
```

这个方法会检查我们之前定义的范围（在屏幕上能够看到什么）并验证传入的坐标是否在指定的范围内。这个方法会帮助我们忽略可视屏幕以外的坐标。接下来，复制代码清单 11-31 中的代码。

代码清单 11-31 deltaAzimuthForCoordinate 方法

```

- (double)deltaAzimuthForCoordinate:(ARCoordinate *)coordinate {
    double currentAzimuth = _coordinate.azimuth;
    double pointAzimuth   = coordinate.azimuth;

    double deltaAzimuth = ABS( pointAzimuth - currentAzimuth);

    if (currentAzimuth < degreesToRadians(_range) &&
        pointAzimuth > degreesToRadians(360-_range)) {
        deltaAzimuth = (currentAzimuth + ((M_PI * 2.0) - pointAzimuth));
    } else if (pointAzimuth < degreesToRadians(_range) &&
        currentAzimuth > degreesToRadians(360-_range)) {
        deltaAzimuth = (pointAzimuth + ((M_PI * 2.0) - currentAzimuth));
    }
    return deltaAzimuth;
}

```

这个方法会计算位置的方位角与传入坐标的方位角之间的差值。这会帮助我们在一个 360 度的视图里面找到点。我们还调节了参照北极的正的和负的角度。添加代码清单 11-32 中的方法到实现文件。

代码清单 11-32 pointForCoordinate 和 isNorthForCoordinate 方法

```

- (BOOL)isNorthForCoordinate:(ARCoordinate *)coordinate {
    BOOL isBetweenNorth = NO;
    double currentAzimuth = _coordinate.azimuth;
    double pointAzimuth   = coordinate.azimuth;

    if ( currentAzimuth < degreesToRadians(_range) &&
        pointAzimuth > degreesToRadians(360-_range) ) {
        isBetweenNorth = YES;
    } else if ( pointAzimuth < degreesToRadians(_range) &&
        currentAzimuth > degreesToRadians(360-_range)) {
        isBetweenNorth = YES;
    }
    return isBetweenNorth;
}

- (CGPoint)pointForCoordinate:(ARCoordinate *)coordinate {
    CGPoint point;
    CGRect viewBounds = _hudView.bounds;

    double currentAzimuth = _coordinate.azimuth;
    double pointAzimuth   = coordinate.azimuth;
    double pointInclination = coordinate.inclination;

    double deltaAzimuth = [self deltaAzimuthForCoordinate:coordinate];
    BOOL isBetweenNorth = [self isNorthForCoordinate:coordinate];

    if ((pointAzimuth > currentAzimuth && !isBetweenNorth) ||
        (currentAzimuth > degreesToRadians(360-_range) &&
         pointAzimuth < degreesToRadians(_range))) {
        // Right side of Azimuth
        point.x = (viewBounds.size.width / 2) + ((deltaAzimuth /
degreesToRadians(1)) * 12);
    } else {
        // Left side of Azimuth
        point.x = (viewBounds.size.width / 2) - ((deltaAzimuth /
degreesToRadians(1)) * 12);
    }
    point.y = (viewBounds.size.height / 2)
+ (radiansToDegrees(M_PI_2 + _viewAngle) * 2.0)
+ ((pointInclination / degreesToRadians(1)) * 12);
}

```

```

        return point;
    }

```

第一个方法，isNorthForCoordinate，遵循了检查两个坐标的方位角的增量差值时使用的模式。我们现在在检查坐标方向和北极方向之间的角度时也遵从此法。

在 pointForCoordinate 方法中，我们转换 ARCoordinate 类为一个在视图中可用的 CGPoint。我们需要知道坐标差是否在方位角值的范围内以及这个坐标点是否在我们和北极方向之间。我们稍后将返回这个点（坐标），以便它能够在视图中更新。

我们完成 ARController.m 实现文件中的 updateLocations 方法。按代码清单 11-33 显示的代码更新这个方法。

代码清单 11-33 更新 updateLocations 方法

```

- (void)updateLocations {
    if ( !_coordinates || [_coordinates count] == 0 ) return;

    int totalDisplayed = 0;

    for (ARCoordinate *item in _coordinates) {

        UIView *viewToDraw = item.annotation;

        if ([self viewportContainsCoordinate:item]) {

            CGPoint point = [self pointForCoordinate:item];
            float width = viewToDraw.bounds.size.width;
            float height = viewToDraw.bounds.size.height;

            viewToDraw.frame = CGRectMake(point.x - width / 2.0, point.y -
(height / 2.0), width, height);

            if ( !([viewToDraw superview]) ) {
                [_hudView addSubview:viewToDraw];
                [_hudView sendSubviewToBack:viewToDraw];
            }
            totalDisplayed++;

        } else {
            [viewToDraw removeFromSuperview];
        }
    }
}

```

新版的 updateLocations 方法会遍历 _coordinates 数组，并添加来自 pointForCoordinate 方法的 CGPoint 对象。这些点会放置在屏幕上，如果它们经过 viewportContainsCoordinate 方法的验证而在视图的范围内的话。

我们快成功了。在测试应用之前，我们需要添加处理位置和方向更新的方法。复制代码清单 11-34 中的方法到实现文件。

代码清单 11-34 处理位置和方向

```

- (void)locationManager:(CLLocationManager *)manager didUpdateHeading:(CLHeading
*)newHeading {
    if (newHeading.headingAccuracy > 0) {
        deviceHeading = newHeading;
        [self updateCurrentCoordinate];
    }
}

```

```

    }
}
- (BOOL)locationManagerShouldDisplayHeadingCalibration:(CLLocationManager *)manager {
    return YES;
}
- (void)locationManager:(CLLocationManager *)manager didUpdateToLocation:(CLLocation
*)newLocation fromLocation:(CLLocation *)oldLocation {
    if (newLocation != oldLocation) {
        [self updateCurrentLocation:newLocation];
    }
}
- (void)locationManager:(CLLocationManager *)manager didFailWithError:(NSError *)error {
    [self.locationManager stopUpdatingLocation];
}

```

这些方法应该是相当简单的。当接收到一个方向更新时，我们更新当前坐标。当我们接收到一个位置变化时，我们更新位置。如果我们接收到一个错误，则停止位置管理器的更新。

在 ARController.m 文件中找到 initWithViewController 方法。添加代码清单 11-35 中的代码来为我们的属性设置一些初始值。

代码清单 11-35 设置初始值

```

_coordinates = [[NSMutableArray alloc] init];
_range = _hudView.bounds.size.width / 12;
_deviceLocation = [[CLLocation alloc] initWithLatitude:39.75 longitude:-104.86];

```

切换到 RootViewController.m 文件，导入 ARGeoCoordinate 头文件。在 loadView 方法中添加代码清单 11-36 中的代码。

代码清单 11-36 初始化我的位置

```

_arController = [[ARController alloc] initWithViewController:self];

ARGeoCoordinate *tempCoordinate;
CLLocation *tempLocation;

tempLocation = [[CLLocation alloc] initWithLatitude:39.550051 longitude:-105.782067];
tempCoordinate = [[ARGeoCoordinate alloc] initWithCoordinate:tempLocation name:@"Kyle
Roche" place:@"Denver"];
[self.arController addCoordinate:tempCoordinate animated:NO];

```

我们终于为测试做好了准备。与我们的其他示例应用相同，你不能使用模拟器做测试。在物理设备上，你会看到与图 11-7 相似的情景。

11.4 添加社交上下文

应用快创建出来了。我们已经建立了处理位置和方向的更新以及在屏幕上显示一些标签的功能。目前，我们已



图 11-7 迄今为止我们所建立的

经硬编码了一个单一的位置（对我来说）来调试功能。让我们加入 Facebook 上下文，并引入（pull）一些我们的 Facebook 朋友的位置。

如果你没有跳过本章中的那一节，你应该还记得 Facebook iOS SDK 在它的本地应用中使用 URL Schemes 来管理 OAuth 回调。我们已经复制 Facebook SDK 到工程中了，在使用它们之前，我们必须在 .plist 文件中设置 URL Schemes。

在 Xcode 中的 Resources 组中找到 Ch11-Info.plist 文件。

- 1) 打开文件并添加一个新行。
- 2) 设置 Key 为 URL 类型。
- 3) 展开新组。添加一个新的子行。
- 4) 命名 Key 为 URL Schemes。
- 5) 展开新的 Key。设置项目的值为 fb[appID]，其中 appID 是你的 Facebook APP ID。

我们已经在同 SDK 一起下载的 Facebook 演示应用中测试了这个过程。

打开 RootViewController.h 文件。按代码清单 11-37 显示的代码更新接口文件。

代码清单 11-37 添加对于 RootViewController 的 Facebook 链接

```
#import <UIKit/UIKit.h>
#import "FBConnect.h"

@class ARController;

@interface RootViewController : UIViewController <FBRequestDelegate, FBDialogDelegate,
FBSessionDelegate> {
    NSArray *_permissions;
}

@property (nonatomic, retain) Facebook *facebook;
@property (nonatomic, retain) ARController *arController;

@end
```

那个 permissions 数组实际上是没有必要的。我们可以自定义登录方法，但是我一直遵循演示应用和帮助文档中的代码方式，以便你能够更容易地把这个用在你将来的应用中。

检查你的工程以确保它能够正确编译。如果你没有正确地包含 Facebook SDK 的话，在 import 语句的后面，你会看到一个编译错误。

切换到 RootViewController.m 文件。添加代码清单 11-38 中的代码到你最后一个 synthesize 语句的后面。显然，要替换为你自己的 App ID。

代码清单 11-38 为 FB App ID 创建变量

```
@synthesize facebook = _facebook;
static NSString* kAppId = @"your app id";
```

在 loadView 方法的尾部添加代码清单 11-39 中的代码。

代码清单 11-39 开启 Facebook oAuth 对话框

```

_permissions = [[NSArray arrayWithObjects:@"read_stream", @"publish_stream",
@"offline_access", @"friends_checkins", nil], retain];
_facebook = [[Facebook alloc] initWithAppId:kAppId andDelegate:self];
[_facebook authorize:_permissions];

```

如你所见，Facebook 的身份验证有 3 个基本的步骤。第 1 步，建立一个应用将会需要的权限列表。第 2 步，用 App ID 初始化 facebook 类，用 self 初始化了 delegate 属性。第 3 步，使用 authorize 实例方法在一个网页浏览器中启动 oAuth 对话框。别忘了，Facebook 会使用我们之前建立的 URL Scheme 来回调你的应用。

Facebook 平台是围绕 Graph API 建立的。它提供了一个简单的、一致的用于检索和更新与用户社交图表相关的信息的机制。然而，对我们来说，这样做的缺点是我们需要查询一个截面的数据。也就是说，我们需要一个朋友（们）及其最后已知载入记录的位置的子集。这对于标准 Graph API 来说太复杂了。因此，我们会通过一个更高级的叫做 FQL（Facebook Query Language）的媒介来查询 Graph API。FQL 会通过类似 SQL 的接口来显示 Graph API 内的数据。

为了演示 Graph API 调用和 FQL 调用，我们会从标准 Graph API 中调出我们的朋友列表，之后使用 FQL 调用来调出每一个朋友最后载入记录的位置。让我们开始吧。

朋友图表

打开 RootViewController.m 文件。因为这是在我们的例子中设置 GPS 坐标的地方，它有助于扩展这个类来检索 Facebook 位置。

Facebook 的 iOS SDK 使用一个我们已经添加到 RootViewController.h 文件的委托协议来处理 API 调用的结果。在我们调用 Graph API 之前，我们需要创建委托方法。添加代码清单 11-40 中的代码到 RootViewController.m 文件。

代码清单 11-40 Facebook 需要的委托方法

```

/**
 * Called when the Facebook API request has returned a response. This callback
 * gives you access to the raw response. It's called before
 * (void)request:(FBRequest *)request didLoad:(id)result,
 * which is passed the parsed response object.
 */
- (void)request:(FBRequest *)request didReceiveResponse:(NSURLResponse *)response {
    NSLog(@"received response");
}

/**
 * Called when a request returns and its response has been parsed into
 * an object. The resulting object may be a dictionary, an array, a string,
 * or a number, depending on the format of the API response. If you need access
 * to the raw response, use:
 *
 * (void)request:(FBRequest *)request
 *     didReceiveResponse:(NSURLResponse *)response
 */
- (void)request:(FBRequest *)request didLoad:(id)result {
    NSLog(@"RESULT: %@", result);
}

```

```

};

/**
 * Called when an error prevents the Facebook API request from completing
 * successfully.
 */
- (void)request:(FBRequest *)request didFailWithError:(NSError *)error {
    NSLog(@"ERROR: %@", [error localizedDescription]);
};

```

这些方法直接复制自 SDK 附带的例子应用。它们处理对 Facebook 的一次调用会发生的 3 个事件：接收响应，载入响应，响应出现错误。现在，让我们来记录结果。

身份验证的回调委托也需要一些方法。添加代码清单 11-41 中的方法到 RootViewController.m 文件。

代码清单 11-41 Facebook 身份验证的委托方法

```

- (void)fbDidLogin {
    NSLog(@"LOGGED IN");
}

- (void)fbDidNotLogin:(BOOL)cancelled {
    NSLog(@"did not login");
}

- (void)fbDidLogout {
    NSLog(@"LOGOUT");
}

```

在 Xcode 中打开 AppDelegate.m 文件。添加代码清单 11-42 中显示的方法。

代码清单 11-42 处理 oAuth 对话框发送的 URLScheme

```

- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    return [[_rootViewController facebook] handleOpenURL:url];
}

```

那么，我们已经为查询、URL Schemes 以及认证请求的响应做好了所有的准备。让我们开启认证对话框。在你第一次使用这个应用的时候，你会看到与图 11-8 类似的情景。在接下来的认证尝试中，你会看到与图 11-9 类似的情景。在这个过程中注意观察控制台窗口。当身份验证回调时，你会看到注释。

如你所见，应用还没有被授权我们所请求的权限（我们传递到 authorize 方法中的 NSArray）。你只需要在应用第一次登录时进行设置。



图 11-8 最初的身份认证对话框



图 11-9 随后的登录尝试带来了一个不同的对话框

有能够完全避免这个弹出消息的方法（在接下来的登录中）。如果你保存了从 oAuth 回调返回的访问令牌，你可以将它传递给 `authorize` 方法来获得一个刷新的令牌。但是，不这样做也可以完成我们的目的。

在 `RootViewController.h` 文件中，添加两个叫做 `_friendsRequest` 和 `_checkinRequest` 的私有 `FBRequest` 变量。在回调中，Facebook 会发送一个针对创建这个调用的 `FBRequest` 对象的引用。这是我们区分正在为其解析响应的 Graph API 调用的唯一方法。

在 Xcode 中打开 `RootViewController.m` 文件。让我们做一个对 Graph API 的调用并在控制台窗口中查看结果。按代码清单 11-43 显示的代码更新 `fbDidLogin` 方法。

代码清单 11-43 更新 `fbDidLogin` 方法

```
- (void)fbDidLogin {
    NSLog(@"LOGGED IN");
    _friendsRequest = [_facebook requestWithGraphPath:@"me/friends" andDelegate:self];
}
```

这个调用使用预先确定的路径对 Graph API 做了一个请求。我们使用的是 `me` 关键字，其代表当前用户。`friends` 路径会返回一个代表朋友列表的 JSON。你会注意到这个调用正引用我们创建的 `private_friendsRequest` `FBRequest` 对象。

如果你再次运行应用，你应该会在控制台窗口看到与代码清单 11-44 类似的情景。

代码清单 11-44 从 `me/friends` 返回的结果（名字和 ID 已更改）

```
2011-10-08 10:14:55.281 Ch11[1302:707] RESULT: {
    data = (
        {
            id = 12345;
            name = "Bob Smith";
```

```

    },
    {
        id = 12345;
        name = "Bob Smith";
    },
    {
        id = 12345;
        name = "Bob Smith";
    },

```

我们使用返回的前 10 个结果，并在增强现实视图里面加载它们的最后记录。按代码清单 11-45 显示的代码更新 Facebook 查询委托方法。

代码清单 11-45 更新 didLoad 方法

```

- (void)request:(FBRequest *)request didLoad:(id)result {
    if (request == _friendsRequest) {
        for (int i = 0; i < 1; i++) { // change this to whatever number you like
            [_facebook requestWithGraphPath:[NSString stringWithFormat:@"%s/checkins",
                [[[result objectForKey:@"data"] objectAtIndex:i] objectForKey:@"id"]] andDelegate:self];
        }
    } else {
        NSLog(@"RESULT: %@", [[[result objectForKey:@"data"] objectAtIndex:1]]);
    }
}

```

我们花点时间来了解下这个代码块。我们要检查引用的 FBRequest 对象是我们的 _friendsRequest 对象。如果是的话，我们就建立一个 for 循环（如果将来你想要扩展的话，修改 i<#）来创建对于 Graph API 的第二个调用以调出用户的记录。我们会记录下所有结果。

代码清单 11-46 显示的是一个响应例子（同样，名字改变了）。

代码清单 11-46 JSON 响应例子

```

2011-10-08 10:49:01.573 Ch11[1523:707] RESULT: {
    application = {
        id = 350685531728;
        name = "Facebook for Android";
    };
    comments = {
        data = {
            {
                "created_time" = "2011-10-03T21:43:16+0000";
                from = {
                    id = 345435;
                    name = "Bob Smith";
                };
                id = "42_4";

                message = "Can't wait to hear all about the new assignment.";
            },
            {
                "created_time" = "2011-10-04T01:33:16+0000";
                from = {
                    id = 345345;
                    name = "Dustin Smith";
                };
                id = "602267025184_841815";
                message = "Maybe do this one faster?";
            }
        };
    };
    "created_time" = "2011-10-03T19:52:47+0000";
}

```



```

from = {
    id = 34534435;
    name = "Jason Smith";
};
id = 602267025184;
likes = {
    data = {
        {
            id = 09870928;
            name = "Bob Barbin";
        }
    };
};
message = "Starting the next project. Should be fun!";
place = {
    id = 110506962309835;
    location = {
        city = "Palo Alto";
        country = "United States";
        latitude = "37.41761460129";
        longitude = "-122.15007660582";
        state = CA;
        street = "1601 S California Ave";
        zip = 94304;
    };
    name = "Facebook HQ";
};
}

```

幸运的是，我们有一个与 Facebook 相关的记录。这至少让它显得相关一点。如果你分析一下数据结构，你会发现有一个叫做 place 的元素包含了发送一个新的 GPS 坐标到 ARView 所需要的大部分数据。将登记报文而排名字附加到注释上，以便我们能够使用一些真实的数据。

按代码清单 11-47 显示的代码调整 didLoad 委托方法。

代码清单 11-47 更新 didLoad 方法

```

- (void)request:(FBRequest *)request didLoad:(id)result {
    if (request == _friendsRequest) {
        for (int i = 0; i < 10; i++) {
            [_facebook requestWithGraphPath:[NSString stringWithFormat:@"%s/checkins",
                [[[result objectForKey:@"data"] objectAtIndex:i] objectForKey:@"id"]]] andDelegate:self];
        }
    } else {
        if ([[[result objectForKey:@"data"] count] > 0) {
            NSString *placeName = [[[[result objectForKey:@"data"] objectAtIndex:0]
                objectForKey:@"place"] objectForKey:@"name"];
            NSString *placeLat = [[[[result objectForKey:@"data"] objectAtIndex:0]
                objectForKey:@"place"] objectForKey:@"location"] objectForKey:@"latitude"];
            NSString *placeLong = [[[[result objectForKey:@"data"] objectAtIndex:0]
                objectForKey:@"place"] objectForKey:@"location"] objectForKey:@"longitude"];
            NSString *checkinMessage = [[[[result objectForKey:@"data"] objectAtIndex:0]
                objectForKey:@"message"];

            ARGeoCoordinate *tempCoordinate;
            CLLocation *tempLocation;

            tempLocation = [[CLLocation alloc] initWithLatitude:[placeLat floatValue]
                longitude:[placeLong floatValue]];
            tempCoordinate = [[ARGeoCoordinate alloc] initWithCoordinate:tempLocation
                name:checkinMessage place:placeName];
            [self.arController addCoordinate:tempCoordinate animated:NO];
            [tempLocation release];
        }
    }
}

```

新代码建立了一些新的临时变量以存储来自 JSON 应答的值。临时变量被传递到 addCoordinate 方法以使它们显示在我们的增强现实视图里。JSON 解析是相当简单的，转换成一个 NSDictionary 就可以了，就像这里我们做的一样。你只要沿着路径结构使用 objectForKey 方法就可以了。

再次运行应用。如果检查出你的大部分朋友都位于地理位置相近的位置，你可能需要调节视图中一次可以显示的位置的数量。图 11-10 展示的是将控制台中的结果显示在增强现实视图中的情景。

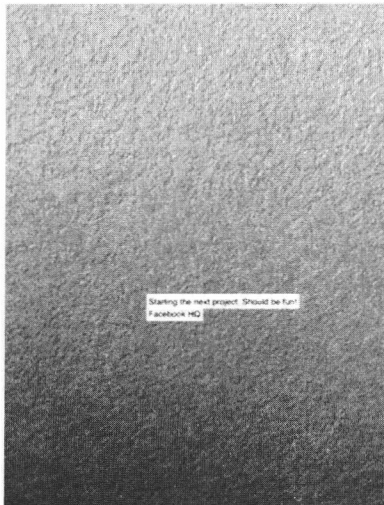


图 11-10 Facebook HQ 显示为一个 Facebook 位置

或许我们可以把这个“乔装打扮”一下并添加一些交互。我们添加一个开启 Facebook 分享对话框的 UIButton 怎么样？

打开 ARAnnotation.m 文件。按代码清单 11-48 显示的代码调整类文件的顶部。

代码清单 11-48 更新 ARAnnotation

```
#import "ARAnnotation.h"
#import "ARCoordinate.h"
#import "AppDelegate.h"
#import "RootViewController.h"
#import "FBConnect.h"

#define ANNOTATION_WIDTH 150
#define ANNOTATION_HEIGHT 200

@implementation ARAnnotation
static NSString* kAppId = @"your app id";
```

我们需要导入 RootViewController，还有 AppDelegate。这是因为 RootViewController 拥有对 Facebook 类的引用并且将作为分享对话框的委托方法。

添加代码清单 11-49 中的方法到类中。

代码清单 11-49 方法 shareButtonClicked

```
- (IBAction)shareButtonClicked:(id)sender {
    AppDelegate *_appDelegate = (AppDelegate *)[[UIApplication sharedApplication]
delegate];
    RootViewController *_rootViewController = _appDelegate.rootViewController;
```

```
NSMutableDictionary* params = [NSMutableDictionary dictionaryWithObjectsAndKeys:
    kAppId, @"app_id",
    @"http://amzn.com/1430239123", @"link",
    @"http://bit.ly/qILSZh", @"picture",
    @"Facebook Places AR App", @"name",
    @"Sharing information on a Facebook Place",
    @"caption",
    @"Look, All my Facebook places are in Augmented
    Reality view.", @"description",
    @"Buy the book", @"message",
    nil];

[_rootViewController.facebook dialog:@"feed"
    andParams:params
    andDelegate:_rootViewController];
}
```

这个方法将会由添加到弹出对话框中的 UIButton outlet 调用。它寻找一个应用运行时的对于 RootViewController 实例的引用，并在之后用我们调用 Facebook API 所需要的 POST 参数创建一个 NSMutableDictionary。我们接着使用 RootViewController 的 facebook 属性调用 dialog 方法，在我们分享的时候，应用会启动一个提示我们添加评语的对话框。

按代码清单 11-50 显示的代码更新 initWithCoordinate 方法。

代码清单 11-50 更新 initWithCoordinate 方法

```
- (id)initWithCoordinate:(ARCoordinate *)coordinate
{
    CGRect annotationFrame = CGRectMake(0, 0, ANNOTATION_WIDTH, ANNOTATION_HEIGHT);

    if (self = [super initWithFrame:annotationFrame]) {
        UIButton *shareButton = [UIButton buttonWithTypeCustom];
        [shareButton setFrame:CGRectMake(0, 0, 32, 32)];
        [shareButton setBackgroundImage:[UIImage imageNamed:@"Facebook-Places.png"]
        forState:UIControlStateNormal];
        [shareButton addTarget:self action:@selector(shareButtonClicked:)
        forControlEvents:UIControlEventTouchUpInside];
        [self addSubview:shareButton];

        UILabel *nameLabel = [[UILabel alloc] initWithFrame:CGRectMake(40, 0,
        ANNOTATION_WIDTH, 20.0)];

        nameLabel.backgroundColor = [UIColor clearColor];
        nameLabel.textAlignment = NSTextAlignmentCenter;
        nameLabel.text = coordinate.name;
        [nameLabel sizeToFit];
        [nameLabel setFrame:CGRectMake(40, 0, nameLabel.bounds.size.width + 8.0,
        nameLabel.bounds.size.height + 8.0)];
        [self addSubview:nameLabel];

        UILabel *placeLabel = [[UILabel alloc] initWithFrame:CGRectMake(40, 0,
        ANNOTATION_WIDTH, 20.0)];
        placeLabel.backgroundColor = [UIColor clearColor];
        placeLabel.textAlignment = NSTextAlignmentCenter;

        placeLabel.text = coordinate.place;
        [placeLabel sizeToFit];
        [placeLabel setFrame:CGRectMake(40, 25, placeLabel.bounds.size.width + 8.0,
        placeLabel.bounds.size.height + 8.0)];
        [self addSubview:placeLabel];

        [self setBackgroundColor:[UIColor clearColor]];
    }
    return self;
}
```

我们通过把现存的 outlet 向右移动 40 个像素来为 UIButton 腾出空间。我使用了一个来自谷歌图像搜索的图片。任何相同名字的图片都可以用在这个地方。我们还为要清除的标签设置了背景色，并且连线了 UIButton 的 UIControlEventTouchUpInside 到 shareButtonClicked 方法。

请注意，我们设置了分享对话框的 delegate 属性为 RootViewController，因此我们还需要对这个类做个小改动。

打开 RootViewController.m 文件并添加代码清单 11-51 中的方法。

代码清单 11-51 来自 FBDialogDelegate 协议的 dialogDidComplete 方法

```
- (void)dialogDidComplete:(FBDialog *)dialog {  
    NSLog(@"publish successful");  
}
```

这个方法是 FBDialogDelegate 协议的委托方法。当对话框完成或者关闭的时候它会被调用。运行应用，你会看到一个稍微有所变化的界面。我在我的增强现实视图里找到相同的记录以便做一些对比。参见图 11-11 的演示。

你可以看到，我们的字有一个清晰的背景，而且 label 左边的按钮也有了合适的放置空间。如果你单击一下这个按钮，一个分享对话框就会启动，参见图 11-12。

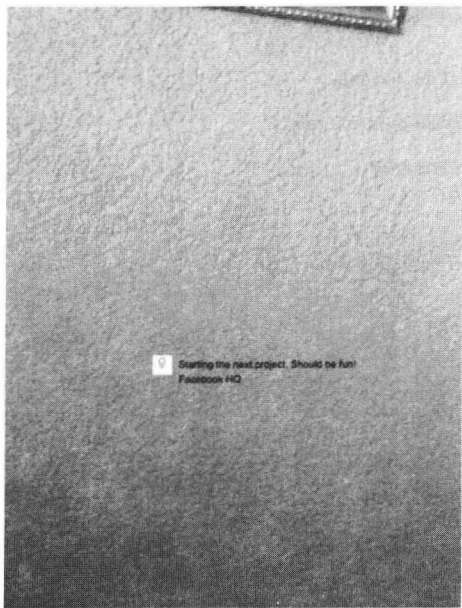


图 11-11 更新 Facebook HQ 为一个 Facebook 位置

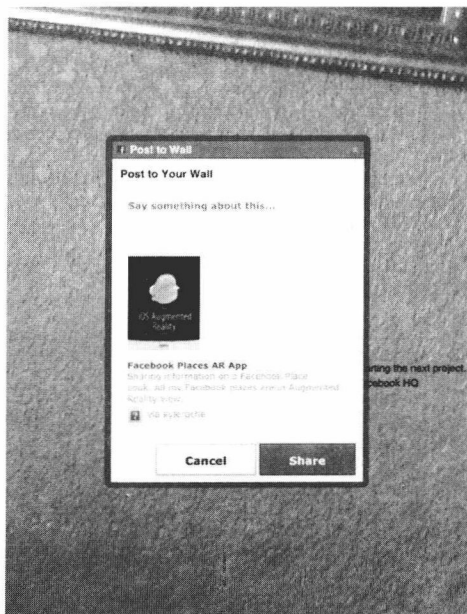


图 11-12 启动分享对话框

如图 11-12 所示，你可以添加一个评语或者直接分享。在添加内容到你的涂鸦墙后，你会在 Facebook 的时光轴中看到它，参见图 11-13。



图 11-13 将分享对话框中的内容发布到了时光轴中

11.5 总结

祝贺你！这是一个非常复杂的例子。我们以一个现有的叫做 ARKit 的开源工程开始，扩展成了一个具有完整功能的社交型增强现实应用。感谢 Zac、Neil 以及 Alasdair 的工作，我们应用的建立才有了这么好的起点。

我们添加了处理自定义应用视图和注释的功能，通过编程修改了它们的外观，并用一个处理交互的 UIButton 扩展了它们。

在使用 Facebook iOS SDK 连接我们的例子到 Facebook 后，我们就可以使用 Graph API 来调出我们的朋友列表并添加它们最后被记录的坐标。

我们使用一个模板为增强现实视图中的每一条记录添加一个按钮，以使你知道怎么将分享放到用户的涂鸦墙。

这是充满趣味的一章。我希望将来在 GitHub 上扩展这个例子。我同时期待来自你的关于这个例子的 GitHub 分支。添加社交内容到增强现实拥有无限的可能。

在第 12 章，我们会讨论面部识别的方法和技术。我们将会在一个增强现实的例子中学习 3 种识别面孔的方法。之后，在第 13 章，我们会选择一个方式并把它做成一个完整的例子应用。

第 12 章

面部识别技术

围绕面孔识别的增强现实应用的编程是最令人印象深刻的。从能够辨别一个友善的人或者一个已知的危险的安全应用，到能够在人群中扫描你的 Facebook 朋友的社交应用，再到允许你试戴太阳镜或者不同发型的零售应用，面部识别增强现实实在是太神奇了。

传统上，面部识别编程的挑战在于实时处理图像的机器所需要的资源的数量。这自然使得在移动设备上实现面部识别变得非常困难。这就是为什么我们看到的大部分的面部识别运算都是通过一个浏览器或者安装程序来运行的原因。

这种情况在 iOS 和最新版本的移动设备上发生了变化。iPad 2 拥有足够的处理能力来处理面部识别应用的更加密集的 CPU 需求。

本章将会讨论一些在移动的 iOS 环境中的面部识别的新方法。

12.1 面部识别的可选项

有很多添加面部识别能力到你的应用中的方法。我们将会学习其中的 3 种。第一个方法出现最早，关于它的可用资料也最多。这个方法叫做 Open Computer Vision 或者 OpenCV。

12.1.1 OpenCV

OpenCV 最早是由 Intel 发起并在 2006 年发布到开源社区的。它获得了 Willow Garden 的资助，并且在编写本书的时候仍然在积极开发中。

OpenCV 的原始代码是基于 C 语言的，因此兼容大部分平台。支持从 C# 到 Java 的所有平台，当然也支持 iOS。

OpenCV 是本章要讲解的最复杂、最高级的主题。由于它的“年龄”和基于 C 语言的特点，它的开发者友好性是最低的。但是，其相比其他方法的优势是在网上有大量的可用例子和教程。在编写本书的时候我还无法在网上找到关于其他两种方法的例子。

12.1.2 iOS 5 的 CIDetector 类

iOS 5 引入了一个 CoreImage 框架下的新的类集合，用于面部侦测。我们将会使用

CIDetector 和 CIFeature 类在实时的摄像头视图里面寻找面孔。

CIDetector 类是作为 iOS 5 的一部分发布的。这个类使用一个预定义的侦测器集合来在图像中寻找指定的特征。iOS 5 有一个定义明确的单独用于面部识别的侦测器。据推测，这个类是为了未来其他常用对象贴图的扩展而构建的。

当建立一个侦测器的时候并没有太多的可用选项，这个我们稍后讨论。基本上，你可以设置侦测器类型（写作本书时只有 CIDetectorTypeFace 一种类型可用）以及侦测精度（高或者低）。

然而，值得一提的是，使用 CIDetector 的速度明显要比 OpenCV 快。

12.1.3 face.com

我们可以把 face.com 看做是第三方的 API 方法。得益于增长如此迅速的 Internet 连接，我们已经能够测试以前根本不可能实现的想法。像 face.com 这样的平台会提供一个免费的用于处理图像中面孔侦测的 REST API。

我会在本章中介绍这个 API 以及它的一些特性，并且第 13 章将更进一步地使用这种方法以建立一个具备完整功能的应用。

12.2 使用 OpenCV 的方式

我们以一个 OpenCV 的例子开始。在创建 Xcode 工程之前，需要下载并安装这个库。在浏览器中打开 <http://www.eosgarden.com/en/opensource/opencv-ios/download/>。这是一个编译的 iOS 版 OpenCV。如果你有兴趣重新编译的话，其源码可以在 <http://github.com/macmade/OpenCV-iOS> 找到。

在本例中，我们将使用 eosgarden 用于 iPad 的预编译版本。下载标记为 iPad 的文件。在 Finder 中导航到下载目录并打开文档。这个例子将会建立在这个版本之上。

让我们首先检查下载的文件结构。如果你展开下载的工程的子目录，你会看到与图 12-1 类似的情景。

需要注意的是，主要文件在 OpenCV 目录下。如果你要下载并建立自己的版本，将只会在 src 和 patch 目录下有文件。目录 lib 和 bin 是为 iOS 模拟器和 iOS 物理设备预编译的版本。

12.2.1 为测试捕获图像

在界面构建器中打开 MainViewController.xib。现在并不需要

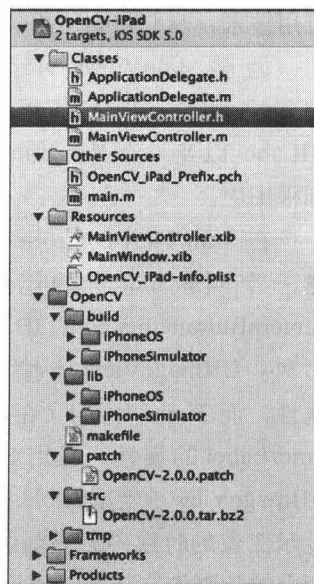


图 12-1 下载并展开 OpenCV-iPad 例子应用的目录

对这个文件进行太多操作。如果你现在启动应用，它会简单地显示标题和链接标签，如图 12-2 所示。

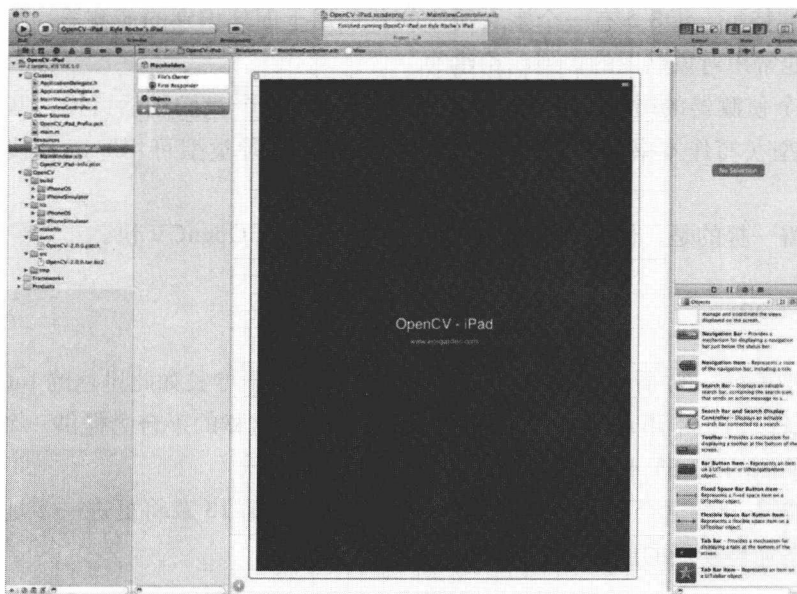


图 12-2 启动应用以显示 MainViewController 的默认状态

第 13 章将会讨论用于面部识别的实时摄像头资源的分析。为了介绍一些用于面部识别的可用技术，我们将会使用静态摄像头图像以使测试在性能和易用性方面更容易一点。

调整 MainViewController，如图 12-3 所示。我们将会添加一个工具栏、一个 UIImageView、一个 UILabel 以及一个 Bar Button 项（带有摄像头标识符）到视图中。

在 MainViewController.h 文件中为 UIToolbar 创建一个叫做 toolbar 的属性。同时，连接一个叫做 cameraButtonClicked 的 IBAction 方法到 Bar Button 项。另外，UIImageView 连接到一个叫做 cameraView 的属性，而 Timer Label（在右下部）则连接到一个叫做 timerLabel 的属性。我用 600×800 的尺寸创建了一个 UIImageView 来匹配 iPad 摄像头的纵横比，以便我们显示图像的时候不必重新缩放。第 13 章将会讨论在运行中缩放摄像头视图以便用正确的纵横比来显示原始的摄像头资源。

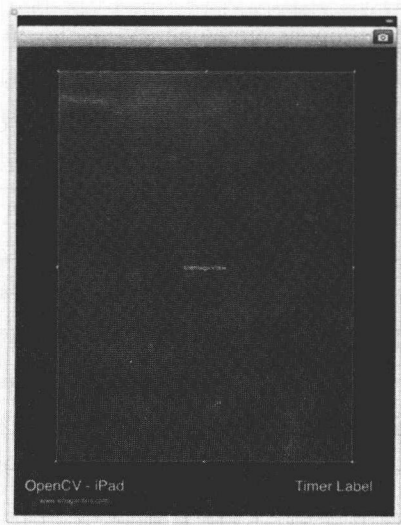


图 12-3 调整 MainViewController 的布局

当 MainViewController.h 文件在 Xcode 中仍处于打开状态的时候，声明一个叫做

_imagePicker 的 UIImagePickerController。另外，指定这个类遵守 UINavigationControllerDelegate、UIImagePickerControllerDelegate、UIPopoverControllerDelegate 以及 UIActionSheetDelegate 协议。

最后，创建 UIPopoverController 类的一个叫做 _imagePopover 的 ivar。我们将会用它控制应用中打开照片库或者保存照片的弹出窗口。

你的新版 MainViewController.h 文件应该与代码清单 12-1 类似。

代码清单 12-1 新版 MainViewController.h 文件

```
#import <UIKit/UIKit.h>
@interface MainViewController : UIViewController <UINavigationControllerDelegate,
UIImagePickerControllerDelegate, UIActionSheetDelegate, UIPopoverControllerDelegate> {
    UIImagePickerController *_imagePicker;
    UIPopoverController *_imagePopover;
}

@property (retain, nonatomic) IBOutlet UIImageView *cameraView;
@property (retain, nonatomic) IBOutlet UILabel *timerLabel;
@property (retain, nonatomic) IBOutlet UIToolbar *toolbar;
- (IBAction)cameraButtonClicked:(id)sender;
@end
```

切换到 MainViewController.m 文件。如果你已经使用界面构建器连接了 IBOutlets，那么就没有必要在实现文件中添加任何的 synthesize 或者 release 语句了。但无论如何，你应该在实现文件的 dealloc 方法中释放 _imagePicker。

注意：

这个类包含了来自 <https://github.com/macmade/facedetect> 的两个方法。在 GitHub 和 Apress (www.apress.com) 的源码的头文件中有相应的许可证文本。

找到 viewDidLoad 方法并按代码清单 12-2 显示的代码更新它。

代码清单 12-2 MainViewController.h 文件中的新版 viewDidLoad 方法

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.
    _imagePicker = [[UIImagePickerController alloc] init];
    _imagePicker.delegate = self;
}
```

当试图在一个模态视图中显示它的时候，需要确保已经为 UIImagePickerController 分配了内存，否则最终会因为要显示一个不存在的 ViewController 而得到一个错误。接下来简单地设置 delegate 为 self，以便能够在类中处理 didFinishPickingImage 委托方法。

找到界面构建器所声明的 cameraButtonClicked 方法。添加代码清单 12-3 中显示的代码。

代码清单 12-3 更新 cameraButtonClicked 方法

```
- (IBAction)cameraButtonClicked:(id)sender {
    UIActionSheet *sheet = [[UIActionSheet alloc] initWithTitle:@"Choose Source"
```

```

    UIAlertController * sheet = [[UIAlertSheet alloc] initWithTitle:@"Choose Source"
    delegate:self cancelButtonTitle:@"Cancel" destructiveButtonTitle:nil
    otherButtonTitles:@"Camera", @"Library", @"Photo Album", nil];
    [_sheet showInView:self.view];
    [_sheet release];
}

```

这个方法可以简单地将控制权转发到 UIAlertController。我们将会 UIAlertController 的委托方法中处理用户的选择。接下来让我们实现该方法。添加代码清单 12-4 中显示的方法到实现中。

代码清单 12-4 UIAlertController 委托方法

```

- (void)actionSheet:(UIAlertSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex {
    if (buttonIndex == 0) {
        _imagePicker.sourceType = UIImagePickerControllerSourceTypeCamera;
        [self presentViewController:_imagePicker animated:YES];
        return;
    } else if (buttonIndex == 1) {
        _imagePicker.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    } else {
        _imagePicker.sourceType = UIImagePickerControllerSourceTypeSavedPhotosAlbum;
    }
    // only for iPad
    _imagePopover = [[UIPopoverController alloc]
initWithContentViewController:_imagePicker];
    _imagePopover.delegate = self;

    [_imagePopover presentPopoverFromRect:toolbar.frame
                                inView:self.view
                                permittedArrowDirections:UIPopoverArrowDirectionAny
                                animated:YES];
    [_imagePicker release];
}

```

这个方法为 UIImagePickerController 呈现了 3 种不同的面部识别的来源 (source) 选择。我们将简单地显示默认的摄像头，但是如果你想要在模拟器上测试这个应用，你需要修改来源类型为照片库或者影集应用。

请记住，这是一个 iPad 应用。iPad 需要 UIImagePickerController 显示在 UIPopoverController 中，而不是显示在 ModalViewController 中。这不同于 iPhone 或者 iPod touch，因此，如果你不想在 iPad 上做测试，请确保是在一个模态对话框中开启所有的 UIImagePickerController，而不是在一个 UIPopoverController 中。

在用户选择了图像之后，UIImagePickerController 类会调用其 didFinishPickingImage 委托方法（如第 6 章所述）。我们将会用代码清单 12-5 中的代码处理这个方法。

代码清单 12-5 委托方法 UIImagePickerController

```

- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo {
    [self dismissModalViewControllerAnimated:YES];
    cameraView.image = image;
}

```

你的应用应该既能够在模拟器上测试，又能够在物理设备上测试。运行这个应用。点击工具栏中的 camera 按钮，然后你会看到一个 UIAlertController，如图 12-4 所示。

如果你正在模拟器上运行，一定要选择 Library 或者 Photo Album。你会看到一个类似于图 12-5 的弹出窗口。如果你选择了 Camera 视图，你会看到一个全屏的摄像头视图。

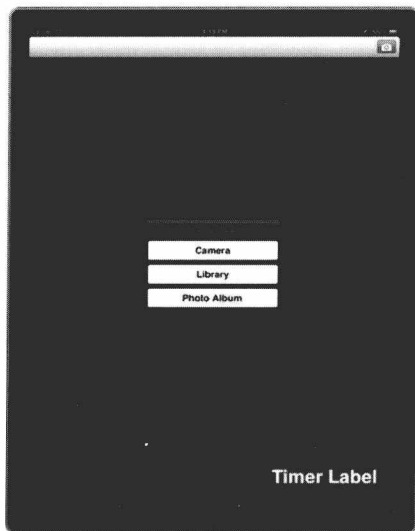


图 12-4 从 UIAlertController 中选择 Camera 作为来源

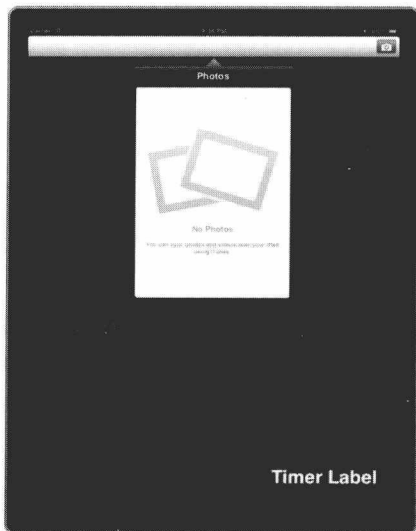


图 12-5 UIPopoverController 在 iPad 上显示了 UIImagePickerController

12.2.2 哈尔级联分类器

在使用 OpenCV 之前，需要添加几个哈尔级联分类器（Haar cascades）到工程中。哈尔级联分类器是一个有组织的用于对象识别的级联分类器或者数字图像功能的集合。这个名字起源于用在第一代面孔识别系统中的哈尔小波（Haar wavelets）。

在哈尔小波出现之前，图像识别过程会分析每个像素的 RGB 值的强度，这会耗费大量的时间和计算机资源。哈尔级联分类器是用来分析一个侦测窗口中的相邻矩形区域并计算这些区域的像素强度的，以便它们可以在区与区之间相互对比，而不是在像素与像素之间。这种方法记录在 Viola 和 Jones 于 2001 年发表的“Rapid Object Detection using Boosted Cascade of Simple Features”中（你可以在这个网址找到完整的论文：http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_CVPR2001.pdf），该论文收录在 2001 年计算机视觉与模式识别大会论文集中。

如果你想进一步研究这个问题的话，这个运算背后的数学原理是有据可查的。我们将使用 OpenCV 的源版本附带的哈尔级联分类器。你可以在 <http://opencv.willowgarage.com/wiki/> 这个网页找到该分发。如果你不想下载另一个版本的 OpenCV（更精确地说，这个是原版），你可以直接从 GitHub 上关于本书的例子代码中或者从 Apress 网站（www.apress.com）的 Source Code/Download 区域复制相关的文件。

找到你将会使用的级联分类器的目录，然后将它们拖曳到你的 Xcode 工程中。确保你选中

了复制所需资源的选项。

注意：

如果你是编译的自己的 iOS 5 版 OpenCV 分发，你可以直接跳到下一节。把编译的库添加到工程中，然后你就完全设置好了。

切换到 MainViewController.m 文件。在 import 语句的后面添加代码清单 12-6 中的代码。

代码清单 12-6 私有方法

```
@interface MainViewController (Private)
- (IplImage *)createIplImage:(UIImage *)image;
- (void)openCVFaceDetect;
@end
```

我们简单地声明了一些将会用来分析图像的私有方法。现在，让我们实现这些方法，然后再逐步地讲解它们。

首先添加代码清单 12-7 中的方法到实现中。如前所述，接下来的这两个实例方法来自 eosgarden 示例工程。由于版本原因，它们无法在 iOS 5 上工作，但是我们会马上修复这个问题。

代码清单 12-7 createIplImage 方法

```
- ( IplImage * )createIplImage: ( UIImage * )image
{
    CGImageRef    imageRef;
    CGColorSpaceRef colorSpaceRef;
    CGContextRef  context;
    IplImage      * iplImage;
    IplImage      * returnImage;

    imageRef      = image.CGImage;
    colorSpaceRef = CGColorSpaceCreateDeviceRGB();
    iplImage      = cvCreateImage( cvSize( image.size.width, image.size.height ),
    IPL_DEPTH_8U, 4 );
    context       = CGContextCreate
    (
        iplImage->imageData,
        iplImage->width,
        iplImage->height,
        iplImage->depth,
        iplImage->widthStep,
        colorSpaceRef,
        kCGImageAlphaPremultipliedLast | kCGBitmapByteOrderDefault
    );

    CGContextDrawImage( context, CGRectMake( 0, 0, image.size.width, image.size.height
    ), imageRef );
    CGContextRelease( context );
    CGColorSpaceRelease( colorSpaceRef );

    returnImage = cvCreateImage( cvGetSize( iplImage ), IPL_DEPTH_8U, 3 );

    cvCvtColor( iplImage, returnImage, CV_RGBA2BGR );
    cvReleaseImage( &iplImage );

    return returnImage;
}
```

当我们从摄像头或者资源库取回选中的图像的时候，我们把它存储在一个 UIImageView 中。OpenCV 无法使用 UIImage 格式，因此需要把这个图像转换成它能处理的形式。OpenCV 使用一种叫做 IplImage 的格式，它是 Intel Image Processing Library（英特尔图像处理库）的一部分。在这个演示中没有太多我们需要知道的。辅助函数，诸如来自 eosgarden 的这个，在互联网有各种格式的版本。

我们的转换方法将要在 OpenCV 处理模块中被调用，它会接收一个 UIImage，并用与这个 UIImage 相同的图画绘制一个新的 IplImage。这对于我们继续使用 x、y 坐标在任何检测到的面孔周围覆盖一个半透明的正方形来说，尺寸和颜色都足够接近。

接下来把代码清单 12-8 中的方法复制到实现中。

代码清单 12-8 openCVFaceDetect 方法

```
- ( void )openCVFaceDetect
{
    NSInteger          i;
    NSUInteger          scale;
    NSAutoreleasePool * pool;
    IplImage            * image;
    IplImage            * smallImage;
    NSString            * xmlPath;
    CvHaarClassifierCascade * cascade;
    CvMemStorage         * storage;
    CvSeq               * faces;
    UIAlertView         * alert;
    CGImageRef          imageRef;
    CGColorSpaceRef     colorSpaceRef;
    CGContextRef         context;
    CvRect              rect;
    CGRect              faceRect;

    pool = [ [ NSAutoreleasePool alloc ] init ];
    scale = 2;

    cvSetErrMode( CV_ErrModeParent );

    xmlPath = [ [ NSBundle mainBundle ] pathForResource:
@"haarcascade_frontalface_default" ofType: @"xml" ];
    image = [ self createIplImage: cameraView.image ];
    smallImage = cvCreateImage( cvSize( image->width / scale, image->height / scale ),
IPL_DEPTH_8U, 3 );

    cvPyrDown( image, smallImage, CV_GAUSSIAN_5x5 );

    cascade = ( CvHaarClassifierCascade * )cvLoad( [ xmlPath cStringUsingEncoding:
NSASCIIStringEncoding ], NULL, NULL, NULL );
    storage = cvCreateMemStorage( 0 );
    faces = cvHaarDetectObjects( smallImage, cascade, storage, ( float )1.2, 2,
CV_HAAR_DO_CANNY_PRUNING, cvSize( 20, 20 ) );

    cvReleaseImage( &smallImage );

    imageRef = cameraView.image.CGImage;
    colorSpaceRef = CGColorSpaceCreateDeviceRGB();
    context = CGContextCreate
(
    NULL,
    cameraView.image.size.width,
    cameraView.image.size.height,
    8,
```

```

        cameraView.image.size.width * 4,
        colorSpaceRef,
        kCGImageAlphaPremultipliedLast | kCGBitmapByteOrderDefault
    );

    CGContextDrawImage
    (
        context,

        CGRectMake( 0, 0, cameraView.image.size.width, cameraView.image.size.height ),
        imageRef
    );

    CGContextSetLineWidth( context, 1 );
    CGContextSetRGBStrokeColor( context, ( CGFloat )0, ( CGFloat )0, ( CGFloat )0, (
    CGFloat )0.5 );
    CGContextSetRGBFillColor( context, ( CGFloat )1, ( CGFloat )1, ( CGFloat )1, (
    CGFloat )0.5 );

    if( faces->total == 0 )
    {
        alert = [ [ UIAlertView alloc ] initWithTitle: @"No faces" message: @"No faces
        were detected in the picture. Please try with another one." delegate: NULL
        cancelButtonTitle: @"OK" otherButtonTitles: nil ];

        [ alert show ];
        [ alert release ];
    }
    else
    {
        for( i = 0; i < faces->total; i++ )
        {
            rect = *( CvRect * )cvGetSeqElem( faces, i );
            faceRect = CGContextConvertRectToDeviceSpace( context, CGRectMake( rect.x *
            scale, rect.y * scale, rect.width * scale, rect.height * scale ) );

            CGContextFillRect( context, faceRect );
            CGContextStrokeRect( context, faceRect );
        }

        cameraView.image = [ UIImage imageWithCGImage: CGBitmapContextCreateImage(
        context ) ];
    }

    CGContextRelease( context );
    CGColorSpaceRelease( colorSpaceRef );
    cvReleaseMemStorage( &storage );
    cvReleaseHaarClassifierCascade( &cascade );
    cvReleaseImage( &smallImage );

    [pool release];
}

```

我们来浏览一下这段代码。首先，我们建立了一个 `NSAutoReleasePool` 用来在图像处理的过程中管理内存的释放和分配。从文件目录加载了哈尔级联分类器，并为侦测器设置了一些类似大小、宽度以及缩放比例的基本信息。

在使用新辅助方法将 `UIImage` 转换为 `IplImage` 后，开始使用 `cvHaarDetectObjects` 方法在图片中侦测面孔。这时会出现几种情况：第一种，如果没有找到任何的面孔，会触发一个 `UIAlertView` 来提示用户选择一个新的图像；第二种，如果确实找到了面孔，会遍历它们，并在图像上每个面孔的周围覆盖一个半透明的正方形。

如果你在阅读本书之前深入学习过 cocos2D，这里的几个有趣的方法你可能会熟悉。DeviceSpace 和 WorldSpace 通常是游戏编程中的概念。有时候，你会以像素为单位来处理一幅图像，这可能不同于其他系统中的 CGPoints 或者百分比。这些基于 C 语言的转换函数使特殊系统之间的切换变得很容易。

两个处理方法已经就位。在从资源库或者摄像头获得 UIImage 后，需要调用这两个方法。找到 didFinishPickingImage 委托方法，并添加代码清单 12-9 中的代码行到这个例程的末尾。

代码清单 12-9 调用 OpenCV 例程

```
[self openCVFaceDetect];
```

现在已为测试做好了准备。在模拟器或者一个物理设备上启动应用。选择资源库中的图像。选择图像以后，发生了意外：控制台显示了类似代码清单 12-10 中的输出。

代码清单 12-10 不好的输出

```
Detected an attempt to call a symbol in system libraries that is not present on the
iPhone:
strtod$UNIX2003 called from function _ZL10icv_strtodP13CvFileStoragePcPS1_ in image
OpenCV-iPad.
```

幸运的是，我们知道是什么原因造成了这种情况。在大部分可用的 OpenCV 版本中，没有一个方法是为设备和模拟器的架构创建的。在例子中使用了一个需要更新的静态库。让我们更新后再试一遍。

OpenCV 有很多伟大的开源构建脚本。我最常使用的版本是由 Yoshimasa Niwa 贡献的 (<https://github.com/niw>)。我们将照此建议来修复例子工程。从以下网址下载 OpenCV 2.2.0 的源码：<http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.2/OpenCV-2.2.0.tar.bz2/download>。

注意：

可能会有 OpenCV 的新版本。这个例程应该仍然能运行良好。如果你无法找到 2.2.0，试着使用一个新的版本。

在 Mac 上打开命令行工具。导航到你下载的文件所在的目录。运行代码清单 12-11 中的命令。

代码清单 12-11 提取 OpenCV

```
Kyle-Roches-MacBook-Pro-2:Downloads kyleroches$ tar xjvf OpenCV-2.2.0.tar.bz2
x OpenCV-2.2.0/.DS_Store
x OpenCV-2.2.0/3rdparty/CMakeLists.txt
x OpenCV-2.2.0/3rdparty/ilmimf/LICENSE
x OpenCV-2.2.0/3rdparty/ilmimf/README
.... on and on and on....
```

既然我们提取了类库，就需要打一个小补丁以确保其能够用于 iOS。从 https://github.com/niw/iphone_opencv_test 下载补丁文件。为简单起见，下载整个 GitHub 仓库并移动 OpenCV 版

本到 GitHub 所在的目录。运行代码清单 12-12 显示的命令。

代码清单 12-12 为 OpenCV 打补丁

```
Kyle-Roches-MacBook-Pro-2:OpenCV-2.2.0 kyleroche$ patch -p1 < ../OpenCV-2.2.0.patch
patching file CMakeLists.txt
patching file modules/CMakeLists.txt
```

注意：

我们将在下一节用到这个 cmake（补丁文件）。如果你还没有安装，你可以从 Macports 或者 Homebrew 获得它。如果你确实必须要安装 cmake，你可能会需要在启动后喝一杯咖啡。这个安装会花很长时间。

现在我们有了一个打补丁后的 OpenCV 版本，我们可以为当前架构重建它。运行代码清单 12-13 显示的命令。

代码清单 12-13 为模拟器建立 OpenCV 静态库

```
Kyle-Roches-MacBook-Pro-2:build_simulator kyleroche $ export IPHONEOS_VERSION_MIN="4.0"
Kyle-Roches-MacBook-Pro-2:build_simulator kyleroche$ ../opencv_cmake.sh Simulator
../OpenCV-2.2.0
Kyle-Roches-MacBook-Pro-2:build_simulator kyleroche$ make -j 4
Kyle-Roches-MacBook-Pro-2:build_simulator kyleroche$ make install
```

可以重复这个过程来建立用于设备的 OpenCV 版本。在 opencv_cmake.sh 命令中使用 Device 代替 Simulator 作为参数（代码清单 12-13 中粗体显示的）。

好的，你应该让一个新 build 目录来代替当前工程中的那个无法工作的。打开 Xcode 并从工程中删除 OpenCV 目录。添加新的 build 目录，记得包含 Simulator 和 Device 目标（如果你计划在一个设备上做测试的话）。

注意：

如果在一个模拟器上做测试，需要把 Accelerate 框架添加到工程中。这并没有包含到例子工程中。

我们试着再次运行这个工程。使用摄像头或者从资源库中选择一个文件。你会看到与图 12-6 类似的情景。

我们在照片库中的图片中找到了 Elvis。

12.2.3 OpenCV 综述

OpenCV 的用户友好。虽然有好的帮助文档，也没有为 iOS 5 提供一些较好的示例，并且没有为当前架构编译。OpenCV 的能力是巨大的，但是它

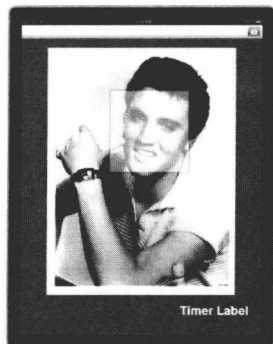


图 12-6 OpenCV 现在运行在 iPad 模拟器上

实现某些功能要作好多工作。作为练习，使用 UILabel 来测量 OpenCV 分析图像所需要的时间。你可以把结果与接下来的两个方法做下对比。

12.3 使用 CIDetector 类的方式

CIDetector 类是 iOS 5 的新类。在 iOS 5 发布的时候只有一个侦测器可用，并且是用于面部识别的。让我们看看使用原生类与使用 OpenCV 的不同。

为了使这个比较尽可能公平，我们将使用相同的应用并添加一些使用 CIDetector 代替 OpenCV 以分析面孔的方法。

再次打开 OpenCV-iPad 工程，然后添加 CoreImage 框架到工程。我们将使用这个框架来访问 CIDetector 类。打开 MainViewController.h，并按代码清单 12-14 显示的代码更新它。

代码清单 12-14 更新 MainViewController.h

```
#import <UIKit/UIKit.h>
#import <CoreImage/CoreImage.h>

#define DETECT_IMAGE_MAX_SIZE 1024

@interface MainViewController : UIViewController <UINavigationControllerDelegate,
UIImagePickerControllerDelegate, UIActionSheetDelegate, UIPopoverControllerDelegate> {
    UIImagePickerController *_imagePicker;
    UIPopoverController *_imagePopover;
}

@property (nonatomic, retain) CIDetector *detector;
@property (retain, nonatomic) IBOutlet UIImageView *cameraView;
@property (retain, nonatomic) IBOutlet UILabel *timerLabel;
@property (retain, nonatomic) IBOutlet UIToolbar *toolbar;
- (IBAction)cameraButtonClicked:(id)sender;
@end
```

我们将需要一个 CIDetector 类型的属性来分析图像。切换到 MainViewController.m 文件并合成新的侦测器属性。接下来，添加代码清单 12-15 中的声明到 import 语句下面的 Private 代码块内。

代码清单 12-15 CIDetectorFaceDetect 的私有方法声明

```
-(void) CIDetectorFaceDetect {
    NSLog(@"CI Face Detect started");
    NSArray *arr = [self.detector featuresInImage:[CIImage
imageWithCGImage:[cameraView.image CGImage]]];
    NSLog(@"Set up Array");
    if([arr count]>0){
        for(int i=0;i<[arr count];i++){
            NSLog(@"%d Face found!", i + 1);
            CIFaceFeature * feature = [arr objectAtIndex:i];
            if(feature.hasLeftEyePosition){
                NSLog(@"Left eye position: (%f,
%f)", feature.leftEyePosition.x, feature.leftEyePosition.y);
            }
            if(feature.hasRightEyePosition){
                NSLog(@"Right eye position: (%f,
%f)", feature.rightEyePosition.x, feature.rightEyePosition.y);
            }
        }
    }
}
```

```

        if(feature.hasMouthPosition){
            NSLog(@"Mouth position: (%f,
%f)", feature.mouthPosition.x, feature.mouthPosition.y);
        }
    }
} else {
    NSLog(@"Nothing detected");
}
}

```

请确保你在实现文件中合成并释放了相应的属性。最后，我们需要按代码清单 12-16 显示的代码更新 `didFinishPickingImage` 委托方法。

代码清单 12-16 更新 `didFinishPickingImage` 方法

```

- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo {
    [self dismissModalViewControllerAnimated:YES];
    cameraView.image = image;

    //[self openCVFaceDetect];
    [self CIDetectorFaceDetect];
}

```

`CIDetector` 类的方式非常不同于 `OpenCV`，尤其如果你是一个 Objective-C 编码员，并且不喜欢工程中包含大量的 C 代码的情况下。这种方法简单易学。我们可以调用 `CIDetector` 类的 `featuresInImage` 实例方法来分析 `UIImage`。没有必要像我们在 `OpenCV` 中做的那样将图像转换为 `IplImage` 格式。`CIDetector` 可以使用原生的 `UIImage` 格式来分析图像。

在模拟器或者物理设备上运行工程。观察控制台，然后你会看到与图 12-7 类似的结果。

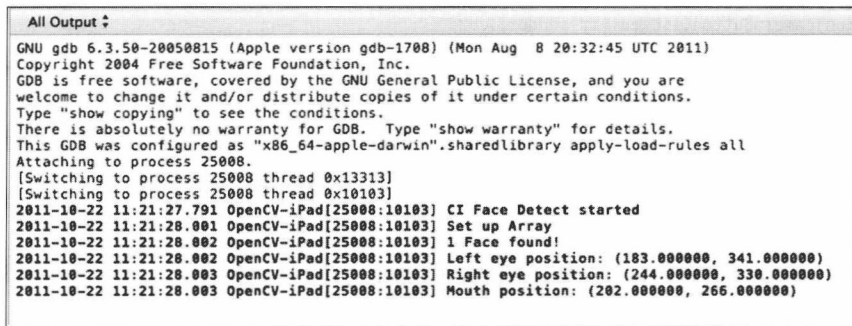


图 12-7 `CIDetector` 正工作在 iPad 模拟器上

如果你愿意的话，你可以扩展这个例子来在图像周围绘制一个盒子。如果你想与我用来测试的 Elvis 的照片做对比的话，你可以从 x, y 坐标上来区分。你可以看到左眼在 (183,341) 位置，右眼在 (244,330) 位置。需要注意的是，这个图像是镜像的。因此，当在模拟器上观察的时候，在我们右侧的右眼是在模拟器中看到的我们左侧的眼睛的右边稍低处。这很准确地给予了我们 Elvis 的脸的角度。

`CIDetector` 返回了一个 `CIFaceFeature` 对象，该对象带有一些记录到控制台的属性。我们希

望这个类会随着时间的推移而扩展，并增加一些新的功能，例如表情识别等。

CIDetector 综述

用不到 15 行的实际代码（扣除做日志记录的代码），我们就能够分析一张照片中的面孔，也不再需要手动添加第三方类库或者建立任何的静态库。它简单地用 iOS 5 来工作。

另外，如果你使用 CIDetector 类来分析一个实时的摄像头视图上，那么你可以看到它能够跟上摄像头缓存的速度。作为一个练习，如果你是用 OpenCV 进行面部识别的话，那么可使用 timerLabel 来比较迄今为止我们讨论的两种方法的性能。

12.4 使用 face.com API 的方式

在以前的例子中，我们受益于原生处理的力量和速度。然而，我们要么牺牲了安装的容易性，要么缺少了关于识别的面孔对象的信息。

如果有一个既能够从照片中识别面孔，又能够识别表情等其他有用信息的方法岂不是更好。

face.com 使用一个 REST API 实现了这一点。对于我们移动开发者来说，其缺点是它需要一直与 Internet 连接。但是，最近，这个缺点可能不是那么突出了。

12.4.1 faces.detect API 的调用

这个 API 调用（来自 face.com）会返回在一个或者多个照片中侦测到的标签，包括几何信息，如眼睛、鼻子、嘴；还有各种属性，如性别、正戴着眼镜，以及正在微笑。

照片还可以直接在 API 请求中上传。上传照片的请求必须以 MIME multipart（多部分）消息的形式用 POST 数据的方式来发送。每一个参数包括 raw 图像数据都应该被指定为一个独立的表单数据块。

以下是使用 face.com API 需要注意的一些事情：

- 所有的坐标都以百分比值的形式提供以支持任何的图片缩放。照片的高度和宽度（对应标签是 height/width）以像素为单位。Yaw（倾斜）、roll（滚动）和 pitch（偏航）在 -90° 到 90° 的范围内。
- 一个照片的宽度或者高度的最大值是 900 像素。你可以通过 POST 或者发送链接的形式发送更大尺寸的照片，但是在内部它们会被重新调整大小以改善性能。
- 我们返回的每一个标签都有一个属性集。其中最重要的属性是面部属性。它包含该标签确定是一个面孔的置信水平。如开发者工具演示中显示的一样，我建议使用置信值超过 50% 的标签。低置信值的标签可能是一个面孔，但更可能是一个检测错误。

12.4.2 添加 face.com 支持到例子中

在 Xcode 中再次打开 OpenCV-iPad。从 GitHub 仓库或者 Apress 网站 (www.apress.com) 的 Source Code/Download 区域复制下面的文件。你也可以在这个网页自己下载这些文件：<http://allseeing-i.com/ASIHTTPRequest>。我们将在第 13 章更多地讨论这个库。现在，只要复制这些文件到你的工程中即可。

- ASIHTTPRequestConfig.h
- ASIHTTPRequestDelegate.h
- ASIProgressDelegate.h
- ASICacheDelegate.h
- ASIHTTPRequest.h
- ASIHTTPRequest.m
- ASIDataCompressor.h
- ASIDataCompressor.m
- ASIDataDecompressor.h
- ASIDataDecompressor.m
- ASIFormDataRequest.h
- ASIInputStream.h
- ASIInputStream.m
- ASIFormDataRequest.m
- ASINetworkQueue.h
- ASINetworkQueue.m
- ASIDownloadCache.h
- ASIDownloadCache.m
- ASIAuthenticationDialog.h
- ASIAuthenticationDialog.m
- Reachability.h(在 External/Reachability 文件夹)
- Reachability.m(在 External/Reachability 文件夹)

另外，你需要添加 CFNetwork、SystemConfiguration、MobileCoreServices 以及 libz.dylib 框架到工程中。在你继续之前请确保你的工程没有编译错误。

12.4.3 face.com API Key

在浏览器中打开 <http://www.face.com> 链接。在右上角，应该会有一个叫做 Developers 的链接。打开这个链接，然后点击页面中央的那个标识为 Get Started 的大图片。face.com 网站的注册过程是完全无害的。如果你非常放心的话，可以提供其他的信息。无论哪种方式，通

过完成 CAPTCHA 来证明你是一个人类, 然后点击 Signup。按照确认邮件中的说明来验证你的账户。

登录后转到 API Keys 选项卡, 创建一个新的应用, 为这个应用设定一个任意的名字。我用 iOS AR Demo 作为我的应用的名字。你会看到一个与图 12-8 相似的页面。

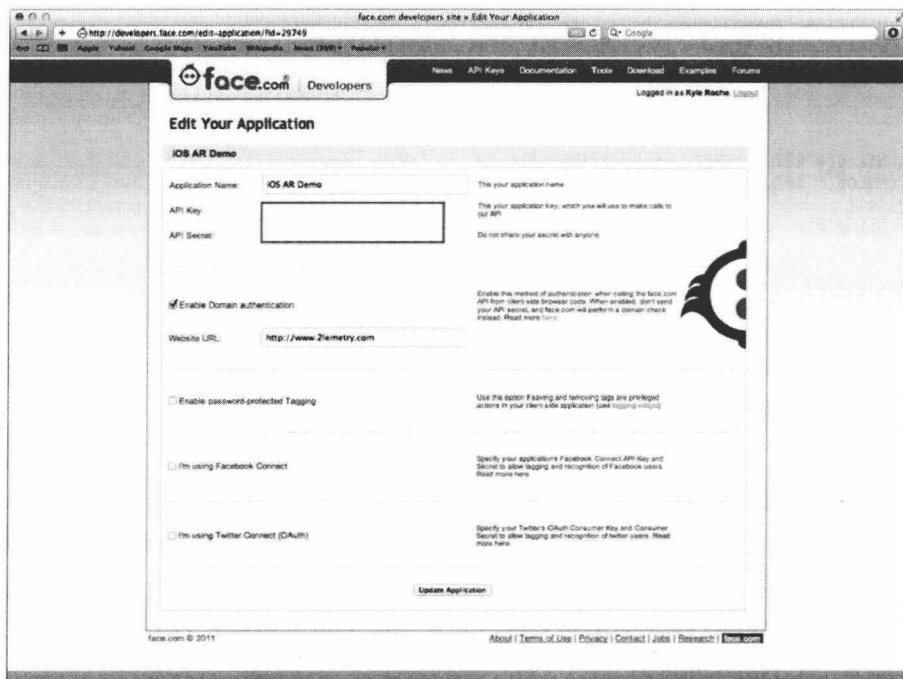


图 12-8 建立你的 face.com 应用

12.4.4 添加 face.com Callout

在 Xcode 中打开 MainViewController.h 文件。导入 ASIFormDataRequest.h 到你的头文件中。切换到 MainViewController.m 文件。在 import 语句的下面、private 块中声明一个叫做 FaceDotComFaceDetect 的新方法。

添加代码清单 12-17 中的方法。

代码清单 12-17 FaceDotComFaceDetect 方法

```
(void)FaceDotComFaceDetect {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSData * imageData = UIImageJPEGRepresentation(cameraView.image, 90);

    NSURL * url = [NSURL URLWithString:@"http://api.face.com/faces/detect.json"];

    ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];
    [request addPostValue:@"" forKey:@"api_key"];
    [request addPostValue:@"" forKey:@"api_secret"];
```

```

[request addPostValue:@"all" forKey:@"attributes"];
[request addData:imageData withFileName:@"image.jpg" andContentType:@"image/jpeg"
forKey:@"filename"];

[request startSynchronous];

NSError *error = [request error];
if (!error) {
    NSString *response = [request responseString];
    NSDictionary *feed = [NSJSONSerialization JSONObjectWithData:[request
responseData]
                                options:kNilOptions
                                error:&error];

    NSLog(@"RETURN: %@", [feed allKeys]);
    NSLog(@"%@", response);
} else {
    NSLog(@"An error occured %d",[error code]);
}

[pool drain];
}

```

在我们讲解这段代码之前，要确保你用你自己的 face.com 资格证书替换了代码中第一部分以粗体显示的代码。稍后我们会详细讨论第二段粗体显示的代码。

这个方法与我们其他的两个用于面部识别的方法类似。首先，我们建立了一个 `NSAutoreleasePool` 来管理内存。接着，我们从 `cameraView UIImageView` 中获取 `UIImage`。但是这次我们为 face.com API 建立了一个 HTTP POST 请求。我们添加 raw JPEG 图画数据到 POST 主体中作为一个 null 参数。

下一步，我们分析了应答以确保其未抛出异常。如果没有错误，我们继续串行化 JSON 应答并将它转换为一个 `NSDictionary`。

接着，我们简单地显示应答信息到控制台。最后一次更新 `didFinishPickingImage` 方法来注释掉前一种方法，并调用我们的新方法。按代码清单 12-18 显示的代码更新这个方法。

代码清单 12-18 更新 `didFinishPickingImage` 方法

```

- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo {
    [self dismissModalViewControllerAnimated:YES];
    cameraView.image = image;

    //[self openCVFaceDetect];
    //[self CIDetectorFaceDetect];
    [self FaceDotComFaceDetect];
}

```

在我们运行这个应用之前，先讨论一下代码清单 12-17 中第二段粗体显示的代码。iOS 5 为框架对象引入了对 JSON 的原生支持。你可以使用新的 `NSJSONSerialization` 类来转换 JSON 为 Foundation 对象或者转换 Foundation 对象为 JSON。

一个能够转换成 JSON 的对象必须具有以下属性：

- 顶级对象必须是一个 `NSArray` 或者 `NSDictionary`；
- 所有的对象都必须是 `NSString`、`NSNumber`、`NSArray`、`NSDictionary` 或者 `NSNull` 的

实例;

- 所有的 dictionary 的 keys 都是 NSString 的实例;
- Numbers 不是 NaN 或者无穷大。

好了, 现在运行工程。我将再次选择 Elvis 图像来使事情保持一致。代码清单 12-19 显示的是新请求的控制台输出。

代码清单 12-19 来自 face.com 请求的控制台输出

```
2011-10-22 14:41:08.392 OpenCV-iPad[25731:10103] RETURN: (
    status,
    photos,
    usage
)
2011-10-22 14:41:08.393 OpenCV-iPad[25731:10103]
{
    "photos":[
        {
            "url":"http://face.com/images/ph/f1a498f5c46aab132f4a1d980297698f.jpg",
            "pid":"F@daa96ea9dcce39470a09ea5648c87356_cd31b28498224cf9ccb39f9194569af8",
            "width":380,
            "height":481,
            "tags":[
                {
                    "tid":"TEMP_F@daa96ea9dcce39470a09ea5648c87356_cd31b28498224cf9ccb39f9194569af8_53.68_35.55_0_0",
                    "recognizable":true,
                    "threshold":null,
                    "uids":[
                    ],
                    "gid":null,
                    "label":"",
                    "confirmed":false,
                    "manual":false,
                    "tagger_id":null,
                    "width":32.11,
                    "height":25.36,
                    "center":{
                        "x":53.68,
                        "y":35.55
                    },
                    "eye_left":{
                        "x":48.95,
                        "y":27.54
                    },
                    "eye_right":{
                        "x":63.83,
                        "y":32.24
                    },
                    "mouth_left":{
                        "x":45.09,
                        "y":39.27
                    },
                    "mouth_center":{
                        "x":49.54,
                        "y":41.74
                    },
                    "mouth_right":{
                        "x":56.75,
                        "y":42.99
                    },
                    "nose":{
                        "x":50.21,
```

```

        "y":37.12
    },
    "ear_left":null,
    "ear_right":null,
    "chin":null,
    "yaw":-14.78,
    "roll":21.81,
    "pitch":-7.49,
    "attributes":{
        "glasses":{
            "value":"false",
            "confidence":88
        },
        "smiling":{
            "value":"true",
            "confidence":83
        },
        "face":{
            "value":"true",
            "confidence":92
        },
        "gender":{
            "value":"male",
            "confidence":33
        },
        "mood":{
            "value":"happy",
            "confidence":76
        },
        "lips":{
            "value":"parted",
            "confidence":92
        }
    }
}
    ]
}
},
"status":"success",
"usage":{
    "used":1,
    "remaining":4999,
    "limit":5000,
    "reset_time_text":"Sat, 22 Oct 2011 21:41:07 +0000",
    "reset_time":1319319667
}
}
}

```

除了识别出图像中的人物有 33% 的可能性是男性，剩下的结果与我们以前看到的相同。在数据结构的差异中有一些事情需要注意。face.com 使用 x, y 的百分比来代替像素。同时，y 坐标是从顶部开始的而不是下部。

花点时间来分析下 face.com 返回的详细信息。face.com 检测到了用户的表情，以及一些如是否有眼睛和嘴是否张开的可识别的特性。有大量的应用能够受益于这个级别的细节。我们希望有一天在 CIDetector 类中也能够看到这些。

12.5 测试性能

这些都是相互之间稍微有所不同的方法，但是对比一下它们的速度以供参考。在你的工程

中创建一个继承自 NSObject 的叫做 CodeTimestamps.m 的新类。按代码清单 12-20 显示的代码更新头文件。

直接从 GitHub 或者 Apress (www.apress.com) 的工程中复制这些文件可能更容易些。在本书的代码仓库中有合适的头文件和版权声明可用。

代码清单 12-20 CodeTimestamps.h

```
#import <Foundation/Foundation.h>

// Comment out this line to disable timestamp logging.
#define USE_TIMESTAMPS 1

// How often timing data is output to the logs.
#define kLogTimeInterval 10.0

// Macro to give us an efficient one-time function call.
// The token trickiness is from here:
// http://bit.ly/fQ6G1h
#define TokenPasteInternal(x,y) x ## y
#define TokenPaste(x,y) TokenPasteInternal(x,y)
#define UniqueTokenMacro TokenPaste(unique, __LINE__)
#define OneTimeCall(x) \
{ static BOOL UniqueTokenMacro = NO; \
if (!UniqueTokenMacro) {x; UniqueTokenMacro = YES; }}

// Speed performance-tuning functions & macros.
void LogTimeStampInMethod(const char *fnName, int lineNum);
void LogTimestampChunkInMethod(const char *fnName, int lineNum, BOOL isStart, BOOL isEnd);
void printAllLogs();
#ifdef USE_TIMESTAMPS

#define LogTimestamp LogTimeStampInMethod(__FUNCTION__, __LINE__)
#define LogTimestampStartChunk LogTimestampChunkInMethod(__FUNCTION__, __LINE__, YES, NO)
#define LogTimestampMidChunk LogTimestampChunkInMethod(__FUNCTION__, __LINE__, NO, NO)
#define LogTimestampEndChunk LogTimestampChunkInMethod(__FUNCTION__, __LINE__, NO, YES)

#else

#define LogTimestamp
#define LogTimestampStartChunk
#define LogTimestampMidChunk
#define LogTimestampEndChunk

#endif

#ifndef PrintName
#define PrintName NSLog(@"%s", __FUNCTION__)
#endif
```

在 Xcode 中打开 CodeTimestamps.m 文件，并添加代码清单 12-21 中的代码。

代码清单 12-21 CodeTimestamps.m

```
#import "CodeTimestamps.h"

#import <mach/mach.h>
#import <mach/mach_time.h>

#define kNumSlowestChunks 5
#define kNumMidPoints 5
```

```

static NSMutableArray *chunkData = nil;

@class ChunkTimeInterval;

@interface LogHelper : NSObject {
@private
    NSTimer *logTimer;
    NSMutableArray *pendingLines;
    NSMutableArray *slowestChunks;
}

+ (LogHelper *)sharedInstance;

- (void)startLoggingTimer;
- (void)printOutTimingData:(NSTimer *)timer;
- (void)addLogString:(NSString *)newString;

- (void)maybeAddTimeIntervalAsSlowest:(ChunkTimeInterval *)timeInterval;
- (void)logSlowestChunks;

- (void)consolidateTimeIntervals:(NSMutableArray *)timeIntervals;

@end

@interface ChunkStamp : NSObject {
@public
    const char *fnName;
    int lineNum;

    uint64_t timestamp;
    NSThread *thread;
    BOOL isStart;
    BOOL isEnd;
}

- (NSComparisonResult)compare:(id)other;

@end

void printAllLogs() {
    [[LogHelper sharedInstance] printOutTimingData:nil];
}

uint64_t NanosecondsFromTimeInterval(uint64_t timeInterval) {
    static struct mach_timebase_info timebase_info;
    OneTimeCall(mach_timebase_info(&timebase_info));
    timeInterval *= timebase_info.numer;
    timeInterval /= timebase_info.denom;
    return timeInterval;
}

// This function needs to be _fast_ to minimize interfering with
// timing data. So we don't actually NSLog during it, using LogHelper.
void logTimeStampInMethod(const char *fnName, int lineNum) {
    OneTimeCall([[LogHelper sharedInstance] startLoggingTimer]);
    static uint64_t lastTimestamp = 0;
    uint64_t thisTimestamp = mach_absolute_time();
    NSString *logStr = nil;
    if (lastTimestamp == 0) {
        logStr = [NSString stringWithFormat:@"%s:%4d", fnName, lineNum];
    } else {
        uint64_t elapsed = NanosecondsFromTimeInterval(thisTimestamp - lastTimestamp);
        logStr = [NSString stringWithFormat:@"%s:%4d - %9llu nsec since last
timestamp",
                                fnName, lineNum, elapsed];
    }
    [[LogHelper sharedInstance] addLogString:logStr];
    lastTimestamp = thisTimestamp;
}

```

```

void InitChunkData() {
    if (chunkData) return;
    chunkData = [NSMutableArray new];
}

void LogTimestampChunkInMethod(const char *fnName, int lineNum, BOOL isStart, BOOL
isEnd) {
    OneTimeCall(InitChunkData());
    OneTimeCall([[LogHelper sharedInstance] startLoggingTimer]);
    ChunkStamp *stamp = [[ChunkStamp new] autorelease];
    stamp->fnName = fnName;
    stamp->lineNum = lineNum;
    stamp->timestamp = mach_absolute_time();
    stamp->thread = [NSThread currentThread];
    stamp->isStart = isStart;
    stamp->isEnd = isEnd;

    @synchronized(chunkData) {
        [chunkData addObject:stamp];
    }
}

@interface ChunkTimeInterval : NSObject {
@public
    NSString *intervalName; // strong
    uint64_t nanoSecsElapsed;
}
- (id)initFromStamp:(ChunkStamp *)stamp1 toStamp:(ChunkStamp *)stamp2;
@end

@implementation ChunkTimeInterval
- (id)initFromStamp:(ChunkStamp *)stamp1 toStamp:(ChunkStamp *)stamp2 {
    if (![super init]) return nil;
    intervalName = [NSString stringWithFormat:@"%s:%d - %s:%d",
        stamp1->fnName, stamp1->lineNum, stamp2->fnName, stamp2->lineNum]
    retain];
    nanoSecsElapsed = NanosecondsFromTimeInterval(stamp2->timestamp - stamp1-
>timestamp);
    return self;
}
- (void)dealloc {
    [intervalName release];
    [super dealloc];
}
- (NSString *)description {
    return [NSString stringWithFormat:@"< %@ %p> %@ %llu", [self class], self,
intervalName, nanoSecsElapsed];
}
@end

@implementation LogHelper

+ (LogHelper *)sharedInstance {
    static LogHelper *instance = nil;
    if (instance == nil) instance = [LogHelper new];
    return instance;
}

- (id)init {
    if (![super init]) return nil;
    pendingLines = [NSMutableArray new];
    slowestChunks = [NSMutableArray new];
    return self;
}

```

```

- (void)startLoggingTimer {
    if (logTimer) return;
    logTimer = [NSTimer scheduledTimerWithTimeInterval:kLogTimeInterval
                                                target:self
                                                selector:@selector(printOutTimingData:)
                                                userInfo:nil
                                                repeats:YES];
}

- (void)printOutTimingData:(NSTimer *)timer {
    BOOL didLogAnything = NO;

    // Handle pending lines.
    if ([pendingLines count]) {
        NSLog(@"==== Start non-chunk timestamp data (from \"LogTimestamp\") ====");
        for (NSString *logString in pendingLines) {
            NSLog(@"%@", logString);
        }
        [pendingLines removeAllObjects];
        didLogAnything = YES;
    }

    // Handle chunk data.
    if ([chunkData count]) {
        NSLog(@"==== Start chunk timestamp data (from \"LogTimestamp{Start,Mid,End}Chunk\") ====");
        @synchronized(chunkData) {
            [chunkData sortUsingSelector:@selector(compare:)];
            NSThread *thread = nil;
            NSMutableArray *timeIntervals = [NSMutableArray array];
            uint64_t totalNanoSecsThisChunk;
            uint64_t totalNanoSecsThisThread;
            int numRunsThisThread;
            BOOL thisThreadHadChunks = NO;
            BOOL midChunk = NO;
            ChunkStamp *lastStamp = nil;
            NSString *chunkName = nil;
            for (ChunkStamp *chunkStamp in chunkData) {
                if (chunkStamp->thread != thread) {
                    if (thisThreadHadChunks) {
                        NSLog(@"++ Chunk = %@, avg time = %.4fs", chunkName,
                            (float)totalNanoSecsThisThread / numRunsThisThread / 1e9);
                    }

                    thread = chunkStamp->thread;
                    NSLog(@"--- Data for thread %p ---", thread);
                    [timeIntervals removeAllObjects];
                    midChunk = NO;
                    thisThreadHadChunks = NO;
                    totalNanoSecsThisChunk = 0;
                    totalNanoSecsThisThread = 0;
                    numRunsThisThread = 0;
                }
                if (chunkStamp->isStart) {
                    if (midChunk) {
                        NSLog(@"ERROR: LogTimestampStartChunk hit twice without a LogTimestampEndChunk between them.");
                    }
                    midChunk = YES;
                    thisThreadHadChunks = YES;
                    chunkName = [NSString stringWithFormat:@"%s:%d", chunkStamp->fnName,
                        chunkStamp->lineNum];
                } else if (midChunk) {
                    ChunkTimeInterval *timeInterval = [[[ChunkTimeInterval alloc]
                        initWithStamp:lastStamp toStamp:chunkStamp] autorelease];

```

```

        [timeIntervals addObject:timeInterval];
        totalNanoSecsThisChunk += timeInterval->nanoSecsElapsed;
        if (chunkStamp->isEnd) {
            totalNanoSecsThisThread += totalNanoSecsThisChunk;
            numRunsThisThread++;
            chunkName = [NSString stringWithFormat:@"%@" - %s:%d", chunkName,
chunkStamp->fnName, chunkStamp->lineNum];
            NSLog(@"+ Chunk = %@, time = %.4fs", chunkName,
(float)totalNanoSecsThisChunk/1e9);

            [self consolidateTimeIntervals:timeIntervals];
            for (int i = 0; i < [timeIntervals count] && i < kNumMidPoints;
++i) {
                ChunkTimeInterval *timeInterval = [timeIntervals
objectAtIndex:i];
                int percentTime = (int)round(100.0 * (float)timeInterval-
>nanoSecsElapsed / totalNanoSecsThisChunk);
                NSLog(@"    %2d%% in %@", percentTime, timeInterval-
>intervalName);
            }

            ChunkTimeInterval *totalInterval = [[ChunkTimeInterval new]
autorelease];
            totalInterval->intervalName = [chunkName retain];
            totalInterval->nanoSecsElapsed = totalNanoSecsThisChunk;
            [self maybeAddTimeIntervalAsSlowest:totalInterval];

            [timeIntervals removeAllObjects];
            totalNanoSecsThisChunk = 0;
            midChunk = NO;
        }
        lastStamp = chunkStamp;
    }
    if (thisThreadHadChunks) {
        NSLog(@"++ Chunk = %@, avg time = %d nsec", chunkName,
totalNanoSecsThisThread / numRunsThisThread);
    }
    [chunkData removeAllObjects];
    didLogAnything = YES;
}
if (didLogAnything) {
    [self logSlowestChunks];
    NSLog(@"==== End timestamp data ====");
}
}

- (void)addLogString:(NSString *)newString {
    [pendingLines addObject:newString];
}

- (void)maybeAddTimeIntervalAsSlowest:(ChunkTimeInterval *)timeInterval {
    if ([slowestChunks count] < kNumSlowestChunks ||
        ((ChunkTimeInterval *)[slowestChunks lastObject])->nanoSecsElapsed <
timeInterval->nanoSecsElapsed) {
        [slowestChunks addObject:timeInterval];
        NSSortDescriptor *sortByTime = [[[NSSortDescriptor alloc]
initWithKey:@"nanoSecsElapsed" ascending:NO] autorelease];
        [slowestChunks sortUsingDescriptors:[NSArray arrayWithObject:sortByTime]];
        if ([slowestChunks count] > kNumSlowestChunks) [slowestChunks removeLastObject];
    }
}
}

```

```

- (void)logSlowestChunks {
    if ([slowestChunks count] == 0) return;
    NSLog(@"==== Slowest chunks so far ====");
    for (ChunkTimeInterval *timeInterval in slowestChunks) {
        NSLog(@"# Chunk = %@, time = %.4fs", timeInterval->intervalName,
            (float)timeInterval->nanoSecsElapsed/1e9);
    }
}

- (void)consolidateTimeIntervals:(NSMutableArray *)timeIntervals {
    NSSortDescriptor *sortByName = [[[NSSortDescriptor alloc]
    initWithKey:@"intervalName" ascending:YES autorelease];
    [timeIntervals sortUsingDescriptors:[NSArray arrayWithObject:sortByName]];

    NSMutableArray *consolidatedIntervals = [NSMutableArray array];
    NSString *lastName = nil;
    ChunkTimeInterval *thisInterval = nil;
    for (ChunkTimeInterval *timeInterval in timeIntervals) {
        if ([lastName isEqualToString:timeInterval->intervalName]) {
            thisInterval->nanoSecsElapsed += timeInterval->nanoSecsElapsed;
        } else {
            thisInterval = [[ChunkTimeInterval new] autorelease];
            thisInterval->intervalName = [timeInterval->intervalName retain];
            thisInterval->nanoSecsElapsed = timeInterval->nanoSecsElapsed;
            [consolidatedIntervals addObject:thisInterval];
        }
        lastName = timeInterval->intervalName;
    }
    [timeIntervals removeAllObjects];
    [timeIntervals addObjectsFromArray:consolidatedIntervals];

    NSSortDescriptor *sortByTime = [[[NSSortDescriptor alloc]
    initWithKey:@"nanoSecsElapsed" ascending:NO] autorelease];
    [timeIntervals sortUsingDescriptors:[NSArray arrayWithObject:sortByTime]];
}

@end

@implementation ChunkStamp

(NSComparisonResult)compare:(id)other {
    ChunkStamp *otherStamp = (ChunkStamp *)other;
    if (thread != otherStamp->thread) {
        return (thread < otherStamp->thread ? NSOrderedAscending : NSOrderedDescending);
    }
    if (strcmp(fnName, otherStamp->fnName) != 0) {
        return (strcmp(fnName, otherStamp->fnName) < 0 ? NSOrderedAscending :
        NSOrderedDescending);
    }
    if (timestamp == otherStamp->timestamp) return NSOrderedSame;

    return (timestamp < otherStamp->timestamp ? NSOrderedAscending :
    NSOrderedDescending);
}

@end

```

这段代码是来自 Pulse 的开源 moriarty 库的一部分。你可以在他们的网站上阅读到更多相关内容 (www.pulse.me)。基本上，你需要在你想要测量的方法的开始和结束位置插入对 LogTimestamp 宏的函数调用。该段代码可以将测量精确到纳秒，因此你可以精确地记录工程中

每一个方法所耗费的时间。

返回到 MainViewController.m 文件并导入我们刚刚创建的新类。在每一个面部识别方法的顶部和底部添加代码清单 12-22 中的代码。

代码清单 12-22 LogTimestamp 宏

```
LogTimestamp;
```

按代码清单 12-23 显示的代码更新 didFinishPickingImage 方法。

代码清单 12-23 更新 didFinishPickingImage 方法

```
- (void)imagePickerController:(UIImagePickerController *)picker
didFinishPickingImage:(UIImage *)image editingInfo:(NSDictionary *)editingInfo {
    [self dismissModalViewControllerAnimated:YES];
    cameraView.image = image;

    [self openCVFaceDetect];
    //[self CIDetectorFaceDetect];
    //[self FaceDotComFaceDetect];
}
```

运行工程。你应该看到你的控制台输出会与代码清单 12-24 相似。显示输出会耗费些时间，请耐心等待。

代码清单 12-24 控制台输出

```
2011-10-22 15:12:27.753 OpenCV-iPad[26058:10103] ==== Start non-chunk timestamp data
(from "LogTimestamp") ====
2011-10-22 15:12:27.754 OpenCV-iPad[26058:10103] * -[MainViewController
openCVFaceDetect]: 76
2011-10-22 15:12:27.754 OpenCV-iPad[26058:10103] * -[MainViewController
openCVFaceDetect]: 163 - 664215125 nsec since last timestamp
2011-10-22 15:12:27.755 OpenCV-iPad[26058:10103] ==== End timestamp data ====
```

看上去好像使用 OpenCV 处理图像耗费了 61 231 366 纳秒。为其他方法重复这个过程。我的结果显示出 CIDetector 耗费了 OpenCV 所用时间的 29%。另外，如果你对应用启动时间做了一些测量的话，你会注意到 OpenCV 启动所耗费的时间要比其他方法耗费的时间大得多。

CIDetector 只耗费了 face.com 所用时间的 18%。当然，由于网络调出和返回的大量应答以及使用本地 JSON 串行化类所做的解析也需要时间，所以这是可以理解的。

OpenCV 与 face.com 的对比非常有趣。OpenCV 启动时间再次耗费的整秒数大约比整个 face.com 执行所用的时间快 35%。在不到一秒的时间内，对用户来说几乎是无法区分的。

12.6 总结

本章中我们学习了很多。在类似 OpenCV 的面部识别的已有方法中有一些着重点知识值得掌握。这些类库已经存在很久了，并且失去了一些我们习以为常的在原生 Objective-C 函数和

宏中的易用性。但是，它们是强大的类库可以为类似我们演示的面部识别的新方法的创新铺平了道路。

我们讨论了 iOS 5 的新的 CIDetector 类，它具有不需要加载 Haar cascades 或者使用例子训练类库即可用原生 UIImage 对象做面孔识别的能力。

最后，我们讲解了使用一个有能力分析 raw JPEG 数据的免费的 REST API 的非本地方法。通过这个方法的使用，我们稍带介绍了 iOS 5 中的新功能：原生 JSON 解析。

所有这些方法都有自己的优势和劣势。我们使用 iPad 2 快速浏览了一下图像处理的速度性能并且发现新的 iOS 5 的 CIDetector 类明显比其他两种方法做得好。特别要注意的是，face.com 需要一个完整的网络调出和应答，并且 face.com 几乎跟上了本地方法的速度。同时，如果你正使用一个跨平台的应用，或者你有一个基于网络 API 的额外功能，face.com 方法比起完全的本地方法可能会更好地适合你的需求。

在第 13 章，我们将会深入学习 face.com 方法。我们将会每隔几秒分析屏幕缓存并向 face.com 发送一个请求。我们将使用 cocos2D 在屏幕上呈现（overlay）一些与返回的应答相关的信息。

第 13 章

建立一个面部识别增强现实应用

在第 12 章，我们介绍了一些能够为你的应用带来面孔和对象识别的方法。一些方法实现难度更大一些。本章将会选择这些方法中的一种来建立一个使用面部识别和一些其他讨论过的技术的具有完整功能的增强现实应用。

13.1 应用的目的

为这个应用想出一个特别的目的是有点困难的。我没有试图解决一个很实际的问题，而是试着尽可能多地演示出至今为止本书中学到的关键概念。

说了这么多，这个应用的前提是服务客户。诚然，这有点夸大，但不管怎么样这会很有趣。对于这个工程，设想一个场景：放置在一个餐馆的桌子上的或者一个酒吧的常见位置的一个 iPad。我们将使用这个 iPad 的摄像头来分析人们的表情，并且在我们发现他或者她生气的表情的时候发出客户服务的提醒。

顺便说一下，我们会基于迄今为止在本书中学到的经验来建立这个应用。不幸的是，迄今为止我们所学习到的方法并不会按照我们的预期工作。我想提前提醒你认识到这一点。当本章中的应用看起来“坏”了一半的时候，那是故意的。那并不是你的错误！后续我们会修复它，并学习一点关于 OpenGL 及其有关超线程的知识。

使用的技术

起初，我们只是在屏幕上放置一些关于用户的信息。这为在 cocos2D 中建立一个更加复杂的 HUD 层做了铺垫。我们将扩展这个应用以向客户服务代表发送一个 SMS（Short Message Service，短消息服务）来报告事件。

我们将会使用的技术的核心部件是 face.com 的 REST API。如我们在第 12 章讨论的，face.com 通过一个 REST 请求来接收一个 JPEG 图像并返回识别的面孔的 x, y 坐标，以及与人物的表情、是否戴眼镜和嘴的姿态的相关信息。它是一个非常有用且迅捷的 API。

13.2 快速设置

我们先把这个例子应用所需要的技术准备好。你必须再忍耐 3 个主要的注册过程才能开始有趣的部分。

13.2.1 face.com

在你的浏览器中打开链接：<http://www.face.com>。在右上角，应该会有一个叫做 Developers 的链接。打开这个链接，然后点击页面中央标识为 Get Started 的大图片。face.com 网站的注册过程是完全无害的。如果你非常放心的话，可以提供一些其他的信息。无论哪种方式，通过完成 CAPTCHA 来证明你是一个人类，然后点击 Signup 按钮。按照确认邮件中的说明来验证账户。

在你登录后，转到 API Keys 选项卡。通过点击标识为“Click here to set up a new application”（单击这里建立一个新应用）的链接来创建一个新的应用。你的应用具体是什么名字无关紧要。我命名我的应用为 iOS AR Demo。你会看到一个与图 13-1 类似的页面。

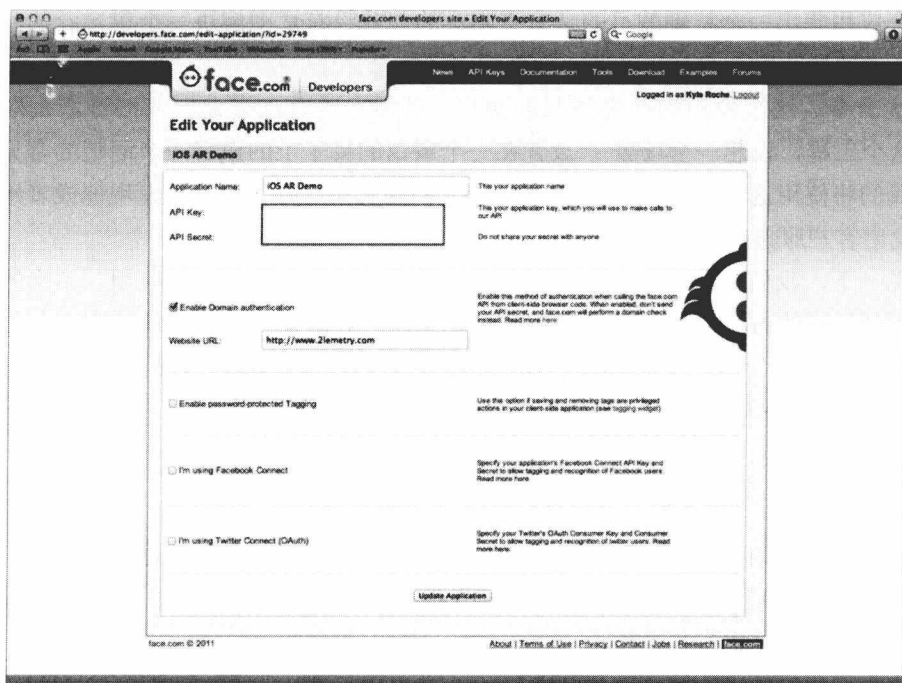


图 13-1 完成 face.com 的应用页面的设置

这里有两个在你开始建立应用的时候会用到的设定：API Key 和 API Secret。face.com 现在来说这样就够了。

13.2.2 cocos2D

如果你还没有安装 cocos2D, 请访问 www.cocos2d-iphone.org/download 下载用于 iPhone 的最新稳定版 cocos2D。我们在第 7 章讲解了怎么在 Xcode 4.2 中安装 cocos2D, 因此如果你是那种喜欢跳过章节的人, 你现在应该返回那一章。

13.2.3 建立 Twilio 账户

Twilio 是一个提供短信服务、语音服务, 甚至是 VoIP 通信的云通信平台。请访问 www.twilio.com/try-twilio 来建立你的免费试用账户。

在建立好你的账户并登录后, 你会在 Dashboard 页面的顶部看到与图 13-2 相似的情景。

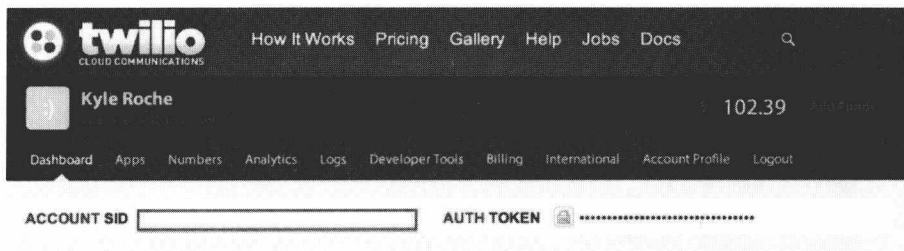


图 13-2 登录并来到 Twilio Dashboard 页面

注意:

与 face.com 的证书类似, 你需要记下 Account (账户) SID 和 Auth Token, 以便能够在我们的演示应用中访问 Twilio 的 API。若没有 Auth Token 则你的 Account SID 是没用的。因此, 永远不要泄露你的 Auth Token !

13.2.4 下载 ASI-HTTP-Request 库

有一些在 iOS 中处理 HTTP 请求的库。事实上, 大部分的功能都可以从零开始编写。但是, 我们需要通过 HTTP 以一个 multipart 表中的一部分的形式来上传 raw 图像数据。有一个第三方库非常适合这个特定的目的。

ASI-HTTP-Request 是一个 GitHub 上的开源库, 可在 GitHub 的 [pokeb/asi-http-request](https://github.com/pokeb/asi-http-request) 下载。我已经为这个例子做了它的一个分支版本, 位于 [kylerocher/asi-http-request](https://github.com/kylerocher/asi-http-request), 以防止你使用新版本的时候碰到问题。保存这个分支版本到你本地电脑的某个位置。

13.2.5 JSON 框架

我们还需要一些帮助来解析 JSON (JavaScript Object Notation) 消息, 因为 JSON 是 face.com API 的首选输出 (并且在 Twilio 中也工作良好)。与 ASI-HTTP-Request 类似, 在 GitHub

上的 stig/json-framework 位置有一个开源工程, 我已经对它做了一个分支 kyleroché/json-framework, 以供参考。保存它到你本地电脑的某个位置。

13.3 工程结构

我们开始创建工程。打开 Xcode 并按图 13-3 显示的使用 cocos2D 模板以创建一个新的工程。

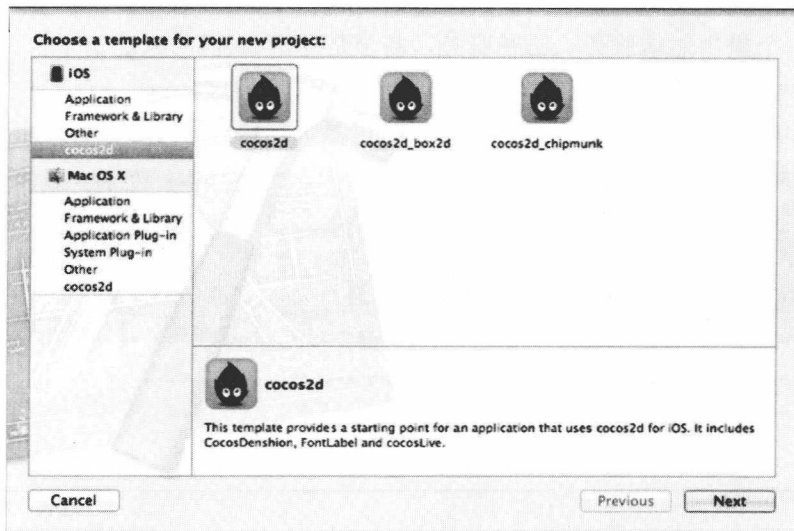


图 13-3 创建新的 cocos2D 工程

命名该工程为 Ch13。如果你需要参考本书的源码的话, 可以在 https://github.com/kyleroché/Professional_iOS_AugmentedReality 或者 www.apress.com 上获得。

首先, 确保为运行这个应用准备好了所需要的所有引用。添加图 13-4 显示的框架和库。

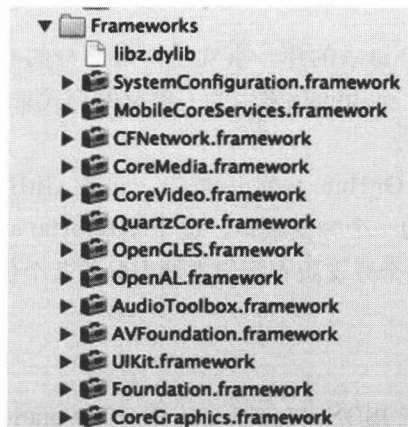


图 13-4 添加所需的框架

在继续之前，你需要确保你的工程仍然能够无错编译。Cocos2D 通常会有一些警告消息，你可以忽略它们（与我们编写的代码不相关的）。

在 Xcode 工程中创建一个叫做 HUD 的新组。我们将会在这个组中保存用于 HUD 覆盖层的资源。在这个组中创建一个新文件。使用 cocos2D 集合的 CCNode 类模板。确保这个新文件继承自 CCLayer。命名这个文件为 HUDLayer.m。

在 GitHub 仓库中有一个叫做 crosshair.png 的文件（在 Ch13 目录下）。复制这个文件到你的工程中。通常，你应该把这个放在工程的 Resources 文件夹下。确保在复制文件到工程中的时候选中了 Copy items into destination group's folder (if needed)（复制选项到目标组的文件夹（若需要））选项。图 13-5 显示的就是 crosshair.png 文件。我们将会把它放在我们从摄像头视图中识别出来的面孔的 x, y 坐标上。

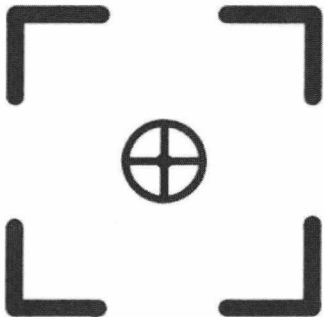


图 13-5 复制 crosshair.png 到你的工程

下一步，我们需要创建一个用于控制面部识别例程的逻辑的类。我们命名这个类为 FaceDetectionLayer。使用 CCNode 模板来创建另一个类，并确保它也继承自 CCLayer。这个类应该被命名为 FaceDetectionLayer.m。

现在，因为我有“洁癖”，我将对默认结构做一个小修改并移除一些用不到的文件。移除 HelloWorldLayer.h 和 HelloWorldLayer.m 文件。你可以选择永久删除它们。

在 Xcode 中打开 FaceDetectionLayer.h 文件，我们开始创建之旅吧！

13.4 建立主场景

为清晰起见，移除 HelloWorldLayer 文件是很好，但是这让工程缺少一个主场景。此时，工程是无法编译的。在 AppDelegate 中有一个对于 HelloWorldLayer 场景的引用。

在 Xcode 中打开 FaceDetectionLayer.h 文件。导入 HUDLayer.h 头文件以便在工程中可以引用。创建一个叫做 _hud 的私有变量来引用 HUD 层类。同时，声明一个会返回 scene 对象的静态方法并且像我们在第 7 章中的例子中做的那样创建一个叫做 initWithHUD 的新方法。

你的新 FaceDetectionLayer.h 文件看起来应该与代码清单 13-1 相似。

代码清单 13-1 更新 FaceDetectionLayer.h 文件

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"
#import "HUDLayer.h"

@interface FaceDetectionLayer : CCLayer {
    HUDLayer *_hud;
}

+ (CCScene *)scene;
- (id)initWithHUD:(HUDLayer *)hud;

@end
```

切换到 FaceDetectionLayer.m 文件。添加代码清单 13-2 中的方法。

代码清单 13-2 静态场景方法

```
+ (CCScene *)scene {
    CCScene *scene = [CCScene node];

    HUDLayer *hud = [HUDLayer node];
    [scene addChild:hud z:1];

    FaceDetectionLayer *layer = [[[FaceDetectionLayer alloc] initWithHUD:hud]
    autorelease];
    [scene addChild:layer];

    return scene;
}
```

这个方法应该看起来很熟悉（除非你跳过了第 7 章）。我们将会为 cocos2D 创建一个新的场景。接着，我们用一个大于主场景的 z 索引值创建了另一个 HUDLayer 的实例。这允许我们方便地在摄像头视图（将会用来处理面孔侦测）上放置一些有用的类似 crosshair PNG 文件的图像或者关于目标的表情的文字信息（高兴、伤心、平和）。

最后，我们调用了 initWithHUD 方法。说到这里，我们也需要添加这个方法的代码。添加代码清单 13-3 中的方法到静态场景方法的后面。

代码清单 13-3 initWithHUD 方法

```
- (id)initWithHUD:(HUDLayer *)hud {
    if (self = [super init]) {
        _hud = hud;
    }
    return self;
}
```

这个方法简单地替换了 cocos2D 模板提供的默认的 init 方法。我们使用 HUD 实例创建了自己的层以代替 FaceDetectionLayer 的基础层。

我们还有最后一个需要添加到 FaceDetectionLayer.m 文件中的方法——dealloc 方法。添加代码清单 13-4 中的方法到实现文件的底部。

代码清单 13-4 dealloc 方法

```
- (void)dealloc {
    [super dealloc];
}
```

现在，如果你试图编译工程，你仍然会收到一个错误提示。它是一个与我们在 AppDelegate 类中对 HelloWorldLayer 的引用相关的 mach-o-link 错误。在 Xcode 中打开 AppDelegate.m 文件，将有关导入 HelloWorldLayer 的语句修改为导入 FaceDetectionLayer.h。

找到代码清单 13-5 中的代码，并用代码清单 13-6 中的代码替换它。

代码清单 13-5 找到这个

```
// Run the intro Scene
[[CCDirector sharedDirector] runWithScene: [HelloWorldLayer scene]];
```

代码清单 13-6 用这个替换

```
// Run the intro Scene
[[CCDirector sharedDirector] runWithScene: [FaceDetectionLayer scene]];
```

到这里你的工程应该能够无错编译了。这个工程仍然没有任何的功能，但是我们至少为继续前进做好了准备。

应该指出的是：刚刚添加的代码实际上其中一些并不是最终的应用需要的。这可能就是项目中一个典型的应用开始的地方。我们有一个 CCNode 类，这个类含有一个用于返回主题的静态方法。但是，在这个例子中我们需要做的事情有点不同。当我们在后面最终移除了刚刚添加的代码中的一些的时候，这会变得更明显。

开启摄像头

我敢确定你正等着看摄像头视图，因此我们开始吧。在 Xcode 中打开 AppDelegate.h 文件。导入 AVFoundation 和 FaceDetectionLayer 头文件。下一步，指定 AppDelegate 遵守 AVCaptureVideoDataOutputSampleBufferDelegate 协议。

我们还需要声明一个叫做 _session 的私有 AVCaptureSession 变量，并创建一个叫做 setupCaptureSession 的 void 方法，这个方法是用来保存关于捕获会话的参数的。

你修改后的 AppDelegate.h 文件应该看起来与代码清单 13-7 类似。

代码清单 13-7 更新后的 AppDelegate.h 文件

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import "FaceDetectionLayer.h"

@class RootViewController;

@interface AppDelegate : NSObject <UIApplicationDelegate,
    AVCaptureVideoDataOutputSampleBufferDelegate> {
    UIWindow *window;
    RootViewController *viewController;
}
```

```

    AVCaptureSession *_session;
}

@property (nonatomic, retain) UIWindow *window;
- (void)setupCaptureSession;

@end

```

切换到 AppDelegate.m 文件。现在你可以移除对于 FaceDetectionLayer 的索引了，如我们在头文件中对它操作的一样。找到代码清单 13-8 中显示的代码段。用代码清单 13-9 中显示的代码替换它。

代码清单 13-8 找到这个

```

EAGLView *glView = [EAGLView viewWithFrame:[window bounds]
                    pixelFormat:kEAGLColorFormatRGB565 // kEAGLColorFormatRGBA8
                    depthFormat:0                      // GL_DEPTH_COMPONENT16_OES
];

```

代码清单 13-9 用这个替换

```

EAGLView *glView = [EAGLView viewWithFrame:[window bounds]
                    pixelFormat: kEAGLColorFormatRGBA8 // kEAGLColorFormatRGBA8
                    depthFormat:0                      // GL_DEPTH_COMPONENT16_OES
];

```

我们在第 7 章也简单讨论过这个。如果我们想要把应用的层覆盖在一个摄像头视图上而不是一个静态的背景类的话，我们就需要修改 pixelFormat 属性。

为了调试更容易，我将在这个地方添加一个小的实用方法。如果你是在人群中的某个地方做的测试或者有大量的被测者对准你的 iPad 摄像头的话，你可以跳过这一段。然而，“孤狼型”的编码者可能在这个例子中受益于对前置摄像头的使用。添加代码清单 13-10 中的代码到 dealloc 方法的上部。

代码清单 13-10 检查前置摄像头可用性

```

-(AVCaptureDevice *)frontFacingCameraIfAvailable
{
    NSArray *videoDevices = [AVCaptureDevice devicesWithMediaType:AVMediaTypeVideo];
    AVCaptureDevice *captureDevice = nil;
    for (AVCaptureDevice *device in videoDevices)
    {
        if (device.position == AVCaptureDevicePositionFront)
        {
            captureDevice = device;
            break;
        }
    }

    if (!captureDevice)
    {
        captureDevice = [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];
    }

    return captureDevice;
}

```


这个方法会简单地检查前置摄像头，如果它存在就返回一个对它的引用。如果它不存在的话，就返回对后置摄像头的引用。当设置采集设备的时候我们会用到这个方法。

在头文件中，我们声明了一个叫做 `setupCaptureSession` 的方法。现在我们创建它。添加代码清单 13-11 中的方法到我们刚刚添加的 `frontFacingCameraIfAvailable` 方法的下面。

代码清单 13-11 设置 `CaptureSession`

```
(void)setupCaptureSession
{
    NSError *error = nil;

    // Create the session.
    _session = [[AVCaptureSession alloc] init];

    // Configure the session to produce lower resolution video frames, if your
    // processing algorithm can cope. We'll specify medium quality for the
    // chosen device.
    _session.sessionPreset = AVCaptureSessionPresetMedium;

    // Find a suitable AVCaptureDevice.
    AVCaptureDevice *device = [self frontFacingCameraIfAvailable]; /*[AVCaptureDevice
defaultDeviceWithMediaType:AVMediaTypeVideo];*/

    // Create a device input with the device and add it to the session.
    AVCaptureDeviceInput *input = [AVCaptureDeviceInput deviceInputWithDevice:device
                                                                    error:&error];
    if (!input) {
        // Handling the error appropriately.
    }
    [_session addInput:input];

    // Create a VideoDataOutput and add it to the session.
    AVCaptureVideoDataOutput *output = [[AVCaptureVideoDataOutput alloc] init];
    [_session addOutput:output];

    // Configure your output.
    dispatch_queue_t queue = dispatch_queue_create("chapter13", NULL);
    [output setSampleBufferDelegate:self queue:queue];
    dispatch_release(queue);

    // Specify the pixel format.
    output.videoSettings =
    [NSDictionary dictionaryWithObject:
    [NSNumber numberWithInt:kCVPixelFormatType_32BGRA]
    forKey:(id)kCVPixelBufferPixelFormatTypeKey];

    // If you wish to cap the frame rate to a known value, such as 15 fps, set
    // minFrameDuration.

    [output.minFrameDuration = CMTimeMake(1, 15);

    // Start the session running to start the flow of data.
    [_session startRunning];
}
```

这段代码的一部分直接来自苹果开发者文档。我们从顶部开始讲解。在为任何未预期的错误建立一个占位器之后，我们建立了一个新的 `AVCaptureSession` 类型的实例。如我们在第 6 章讨论的，这个类是用于从摄像头捕获视频帧的。它在显示时要比 `UIImagePicker` 更灵活一点。

接下来，我们使用之前创建的实用方法检测是否有一个可用的前置摄像头。如果没有前置

摄像头或者你是在人群中做的测试的话，我在代码注释中留下了供你使用的默认选项。为使用摄像头设备，我们建立了 `AVCaptureDeviceInput` 对象，并向 `AVCaptureSession` 类添加了这个输入。

在这个例子中我们还会使用视频输出。因此，我们接着分配了一个新的 `AVCaptureVideoDataOutput` 对象并添加它到 `AVCaptureSession`。

这一步对我们来说是新的。我们需要建立一个 `dispatch_queue_t` 的 ivar 来保存我们队列的视频帧。在这个例子中我们不会讨论 GCD (Grand Central Dispatch)，但是使用 GCD 来代替我们刚建立的 `dispatch_queue_t` 类型的 ivar 将会非常有趣。

最后，我们设置了 `AVCaptureVideoDataOutput` 实例的 `pixelFormat` 属性，接着我们终于可以开启采集会话了。

找到 `applicationDidFinishLaunching` 方法的最后一行并添加代码清单 13-12 中的代码。

代码清单 13-12 调用新方法

```
[self setupCaptureSession];
```

如果你现在运行应用，将只会看到一个黑屏。这是因为我们还没有设置背景为透明。找到代码清单 13-13 显示的代码行，其位于 `applicationDidFinishLaunching` 方法的中间。

代码清单 13-13 找到这一段

```
// make the View Controller a child of the main window
[window addSubview: viewController.view];
```

在那一行的后面，我们将会设置视图为透明以便能够看到底下的摄像头视图。在这个之前，我们还需要在 `AppDelegate.h` 文件里声明一个叫做 `_cameraView` 的 `UIView` 类型的私有变量。实际上，在这个地方我们还添加了一个叫做 `_imageView` 的 `UIImageView` 类型的实例。接着，切换回 `AppDelegate.m` 文件并在代码清单 13-13 中的代码所在的位置的下面添加代码清单 13-14 中的代码。

代码清单 13-14 设置视图为透明

```
// Set view to be transparent.
[[CCDirector sharedDirector].openGLView.backgroundColor = [UIColor clearColor];
[[CCDirector sharedDirector].openGLView.opaque = NO;
glClearColor(0.0, 0.0, 0.0, 0.0);

// Prepare the overlay view and add it to the window.
_cameraView = [[UIView alloc] initWithFrame:[UIScreen mainScreen] bounds];
_cameraView.opaque = NO;
_cameraView.backgroundColor = [UIColor clearColor];

[window addSubview:_cameraView];

_imageView = [[UIImageView alloc] initWithFrame:[UIScreen mainScreen] bounds];
[_cameraView addSubview:_imageView];

[window bringSubviewToFront:viewController.view];
```

首先，用来自单例类型的 CCDirector 类的 sharedDirector 设置了 backgroundColor(背景色) 和 opaque(不透明) 属性，以使两个设置对屏幕而言都是透明的。

继续并照原样运行工程。当像在这里使用 UIImagePickerController 一样处理一个 cocos2D 工程的时候，我们使用了同样的步骤，并且可在摄像头视图之上的层中开始构建。但是，你可能在运行应用的时候注意到了，运行后依然是黑色全屏。这是因为（如果你还记得第 6 章的相关内容的话）还有两个与 AVCaptureSession 的使用相关的步骤：我们排队等候来自缓存的图像，但是我们还需要对这些图像做些什么。

为了访问缓存中的图像，我们可以使用 AVCaptureVideoDataOutputSampleBufferDelegate 协议的一个叫做 didOutputSampleBuffer 的委托方法。这个方法允许我们在捕获到图像的时候从缓存调出图像。

在介绍委托方法之前，请再次切换到 AppDelegate.h 文件并声明一个叫做 _settingImage 的布尔标记，以便能够跟踪何时捕获到了图像。添加代码清单 13-15 中的实用方法到实现文件中。

代码清单 13-15 setImageView 方法

```
-(void)setImageView:(UIImage*)image {
    _imageView.image = image;
    _settingImage = NO;
}
```

这个方法的代码并不多。它简单地设置发送到这个方法中的图像给 UIImageView。我们把这个方法与接收缓存图像的委托方法联系起来。添加代码清单 13-16 中的方法到 setImageView 方法的下面。

代码清单 13-16 从缓存中调出图像的方法

```
-(UIImage *) imageFromSampleBuffer:(CMSampleBufferRef) sampleBuffer
{
    // Get a CMSampleBuffer's Core Video image buffer for the media data
    CVImageBufferRef imageBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
    // Lock the base address of the pixel buffer
    CVPixelBufferLockBaseAddress(imageBuffer, 0);

    // Get the number of bytes per row for the pixel buffer
    void *baseAddress = CVPixelBufferGetBaseAddress(imageBuffer);

    // Get the number of bytes per row for the pixel buffer
    size_t bytesPerRow = CVPixelBufferGetBytesPerRow(imageBuffer);
    // Get the pixel buffer width and height
    size_t width = CVPixelBufferGetWidth(imageBuffer);
    size_t height = CVPixelBufferGetHeight(imageBuffer);

    // Create a device-dependent RGB color space
    CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();

    // Create a bitmap graphics context with the sample buffer data
    CGContextRef context = CGBitmapContextCreate(baseAddress, width, height, 8,
                                                bytesPerRow, colorSpace,
                                                kCGBitmapByteOrder32Little | kCGImageAlphaPremultipliedFirst);
    // Create a Quartz image from the pixel data in the bitmap graphics context
    CGImageRef quartzImage = CGBitmapContextCreateImage(context);
    // Unlock the pixel buffer
```

```
CVPixelBufferUnlockBaseAddress(imageBuffer,0);

// Free up the context and color space
CGContextRelease(context);
CGColorSpaceRelease(colorSpace);

// Create an image object from the Quartz image
UIImage *image = [UIImage imageWithCGImage:quartzImage];

// Release the Quartz image
CGImageRelease(quartzImage);

return (image);
}

- (void)captureOutput:(AVCaptureOutput *)captureOutput
didOutputSampleBuffer:(CMSampleBufferRef)sampleBuffer
fromConnection:(AVCaptureConnection *)connection
{
    UIImage *image = [self imageFromSampleBuffer:sampleBuffer];

    if(_settingImage == NO){
        settingImage = YES;
        [NSThread detachNewThreadSelector:@selector(setImageToView:) toTarget:self
        withObject:image];
    }
}
```

第一个方法也是我们直接从苹果开发者文档中复制的。这个方法返回一个来自 CMSampleBufferRef 缓存的 UIImage 对象。我们在 didOutputSampleBuffer 委托方法的第一行调用它。下一步，在 didOutputSampleBuffer 方法里，我们检查是否已经处在为 UIImageView 设置一个图像的过程中。如果不是，我们就为之前创建的 setImageToView 实用方法分出一个新线程。

再次运行工程。视频显示了，但并不完全是以我们期望的方式。图 13-6 显示的是我的屏幕。

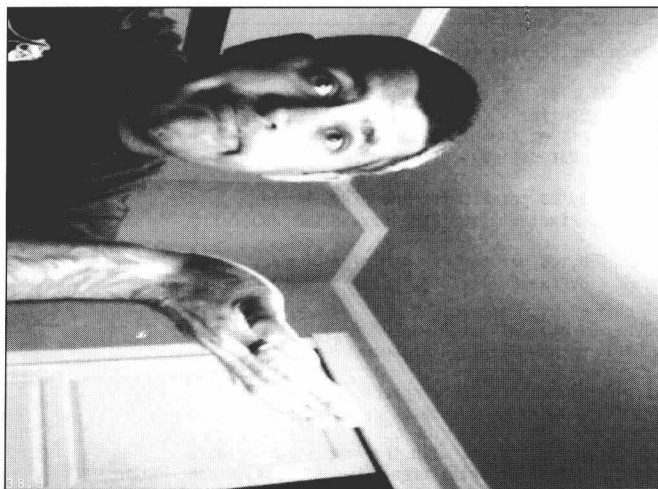


图 13-6 我们侧身了

你可以使用来自 cocos2D 模板的数字叠加，以供参考。对于这个例子（虽然当我们完成的时候它会旋转过来），我是以 Landscape Right（右横向）模式来测试屏幕截图的。

因此我们需要重新将方向调整 90°，并防止图像拉伸。添加来自代码清单 13-17 中的实用方法到 AppDelegate.m 文件。确保你把这个方法放到了 setImageToView 方法的上部，否则当你试图编译工程的时候会收到一个警告消息。

代码清单 13-17 rotatelImage 实用方法

```
-(UIImage *) rotateImage:(UIImage*)image orientation:(UIImageOrientation) orient {
    CGImageRef imgRef = image.CGImage;
    CGAffineTransform transform = CGAffineTransformIdentity;
    //UIImageOrientation orient = image.imageOrientation;
    CGFloat scaleRatio = 1;
    CGFloat width = image.size.width;
    CGFloat height = image.size.height;
    CGSize imageSize = image.size;
    CGRect bounds = CGRectMake(0, 0, width, height);
    CGFloat boundHeight;

    switch(orient) {
        case UIImageOrientationUp:
            transform = CGAffineTransformIdentity;
            break;
        case UIImageOrientationUpMirrored:
            transform = CGAffineTransformMakeTranslation(imageSize.width, 0.0);
            transform = CGAffineTransformScale(transform, -1.0, 1.0);
            break;
        case UIImageOrientationDown:
            transform = CGAffineTransformMakeTranslation(imageSize.width,
imageSize.height);
            transform = CGAffineTransformRotate(transform, M_PI);
            break;
        case UIImageOrientationDownMirrored:
            transform = CGAffineTransformMakeTranslation(0.0, imageSize.height);
            transform = CGAffineTransformScale(transform, 1.0, -1.0);
            break;
        case UIImageOrientationLeftMirrored:
            boundHeight = bounds.size.height;
            bounds.size.height = bounds.size.width;
            bounds.size.width = boundHeight;
            transform = CGAffineTransformMakeTranslation(imageSize.height,
imageSize.height);
            transform = CGAffineTransformScale(transform, -1.0, 1.0);
            transform = CGAffineTransformRotate(transform, 3.0 * M_PI / 2.0);
            break;
        case UIImageOrientationLeft:
            boundHeight = bounds.size.height;
            bounds.size.height = bounds.size.width;
            bounds.size.width = boundHeight;
            transform = CGAffineTransformMakeTranslation(0.0, imageSize.width);
            transform = CGAffineTransformRotate(transform, 3.0 * M_PI / 2.0);
            break;
        case UIImageOrientationRightMirrored:
            boundHeight = bounds.size.height;
            bounds.size.height = bounds.size.width;
            bounds.size.width = boundHeight;
            transform = CGAffineTransformMakeScale(-1.0, 1.0);
            transform = CGAffineTransformRotate(transform, M_PI / 2.0);
            break;
        case UIImageOrientationRight:
            boundHeight = bounds.size.height;
            bounds.size.height = bounds.size.width;
            bounds.size.width = boundHeight;
```

```

        transform = CGAffineTransformMakeTranslation(imageSize.height, 0.0);
        transform = CGAffineTransformRotate(transform, M_PI / 2.0);
        break;
    default:
        [NSException raise:NSInternalInconsistencyException format:@"Invalid image
orientation"];
    }
    UIGraphicsBeginImageContext(bounds.size);
    CGContextRef context = UIGraphicsGetCurrentContext();
    if (orient == UIImageOrientationRight || orient == UIImageOrientationLeft) {
        CGContextScaleCTM(context, -scaleRatio, scaleRatio);
        CGContextTranslateCTM(context, -height, 0);
    } else {
        CGContextScaleCTM(context, scaleRatio, -scaleRatio);
        CGContextTranslateCTM(context, 0, -height);
    }
    CGContextConcatCTM(context, transform);
    CGContextDrawImage(UIGraphicsGetCurrentContext(), CGRectMake(0, 0, width, height),
imgRef);
    UIImage *imageCopy = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return imageCopy;
}

```

我们早在第 7 章就看到过这个方法的片段。基本上，我们检查了设备的方向，接着适当重定向并缩放了图像。按代码清单 13-18 显示的代码更新 setImageToView 方法。

代码清单 13-18 更新 setImageToView 方法

```

-(void)setImageToView:(UIImage*)image {
    UIImage * capturedImage = [self rotateImage:image
orientation:UIImageOrientationLeftMirrored ];
    _imageView.image = capturedImage;
    _settingImage = NO;
}

```

这个方法先对图像做了处理，而不是直接设置 UIImage。我们首先使用刚创建的实用方法对图像做了适当的旋转。之后我们将返回的（处理过的）UIImage 返回给调用者。

再次运行工程。你会看到一个更好的结果。图 13-7 是我的 iPad 屏幕截图。



图 13-7 我们不侧身了

那么，“世界”再次回到正确方向了。接下来，在捕获图像并适当旋转之后，现在我们要

怎么处理这个图像呢？请记住，此处的目的是发送摄像头缓存中的图像到 face.com 并返回从照片中识别出来的任何面孔的坐标。

我们必须为这个功能设置一个方法。解决方法有很多，每一种都有自己的优缺点。首先，我们浏览一下 face.com API 和所需的参数，以便决定怎么发送图像。

13.5 face.com API

face.com API 是一个非常简单的 REST API，它使用一个 multipart 表，通过将想要分析的图像的 raw 数据植入请求的主体中来上传。

我们将只使用一个 face.com API 的调用。faces.detect 调用会返回在一个或者多个照片中侦测到的面孔的标签，包括标签的几何信息，如眼睛、鼻子、嘴巴，还有各种属性信息，例如性别、是否戴眼镜、是否在微笑。

以下是这个调用的使用说明（来自在线文档）：

- 所有的坐标都以百分比值的形式提供，以支持任何的图片缩放。照片的高度和宽度（对应标签是 height/width）以像素为单位。Yaw、roll 和 pitch 在 $-90^{\circ} \sim 90^{\circ}$ 的范围内。
- 一个照片的宽度或者高度的最大值是 900 像素。你可以通过 POST 或者发送链接的形式发送更大尺寸的照片，但是在应用内部它们会被重新调整大小以改善性能。
- 我们返回的每一个标签都有一个属性集。其中最重要的属性是 face 属性。它包含该标签确定是一个面孔的置信水平。我建议使用置信值超过 50% 的标签。
- 面孔侦测是为用户添加数据集以识别索引的一个前提步骤（我们不会这样做）。

这个 API 也有速度限制，在写作本书的时候，这个限制是同一个应用中每分钟最多 5000 次请求。因此，我觉得这个例子可以运行良好。

这个调用的 REST URL 是 `http://api.face.com/faces/detect.[format]`。对于我们来说，我们将会设置 format 等于 JSON。表 13-1 是 POST 请求的输入参数。

表 13-1 faces.detect 调用的参数

必 须	名 字	描 述
是	api_key	你的 face.com 的 API Key
是	api_secret	你的 face.com 的 API Secret
是	urls	以逗号分隔的 JPG 照片的列表
不是	[no name]	照片的 raw 图像数据（urls 替换所需参数）
不是	detector	设置面孔侦测器模式：Normal 或者 Aggressive。Aggressive 模式可能会找到更多的面孔，但是更慢
不是	attributes	取值 all 或者 none，或者一个以逗号分隔的属性的列表
不是	format	使用“json”或者“xml”格式（我们将使用 json）
不是	callback	用于封装一个 json 格式的应答（JSONP 支持）的 JavaScript 方法
不是	callback_url	异步回调 URL

如果你浏览了参数列表, 会发现这个调用实际上是非常简单的。我们的选择是要么上传某个地方的图像并发送一个引用 URL 以便 face.com 能够获取并分析它, 要么我们直接在 POST 请求中发送 raw 图像数据。不用实际测试, 我认为把图像上传到其他某个地方, 再让 face.com 下载并分析的效率要远低于直接在 POST 请求中上传图像。因此, 我们将按后一种方法进行。

现在有了方法导向, 我们再看看计时。我们已经有了能够转换成 JPEG 并附加到 POST 请求中的 UIImage 对象, 但是这很容易导致请求耗费运行自己以及一大堆不必要的事务的 CPU 时间。我建议使用 NSTimer 定期抓取屏幕图像; 接着在后台线程中将图像发送到 face.com 以供分析。当请求返回的时候, 我们会在 HUD 层上呈现关于找到的属性的信息, 并且我们将退到后台直到下一次请求被发送。

13.5.1 使用 ASI-HTTP-Request 库

在继续构建之前, 我们需要使用在前面章节下载的库。找到我们保存的已下载的文档。复制下面的文件到你的工程中。由于我喜欢工程结构精密并且干净, 因此我创建了另一个叫做 ASI 的组来保存这些文件。

- ASIHTTPRequestConfig.h
- ASIHTTPRequestDelegate.h
- ASIProgressDelegate.h
- ASICacheDelegate.h
- ASIHTTPRequest.h
- ASIHTTPRequest.m
- ASIDataCompressor.h
- ASIDataCompressor.m
- ASIDataDecompressor.h
- ASIDataDecompressor.m
- ASIFormDataRequest.h
- ASIInputStream.h
- ASIInputStream.m
- ASIFormDataRequest.m
- ASINetworkQueue.h
- ASINetworkQueue.m
- ASIDownloadCache.h
- ASIDownloadCache.m
- ASIAuthenticationDialog.h
- ASIAuthenticationDialog.m
- Reachability.h(在 External/Reachability 文件夹)
- Reachability.m(在 External/Reachability 文件夹)

该库还需要一些引用，但是由于我是超前思维，因此我们本章早期就添加了它们。

我们暂时后退一步，想想 HTTP POST 请求的最佳位置。请记住，与其他的 iOS 应用类似，cocos2D 应用也是建立在视图的层次结构之上的。参考图 13-8，该图是一个逻辑模型，并不是一个具体的类的代表。

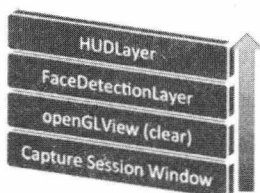


图 13-8 场景的逻辑层次显示了堆叠的视图

向 face.com 发送的 POST 请求会在一个单独的线程里面执行。因此，我们希望返回代码在一个能够与 HUD 层相互作用的层里被解析。HUD 层就是我们实际创建覆盖在界面上的图像和文本的地方，因此我们将会处理这些行动的方法。

看起来处理 POST 请求的最佳选择是 FaceDetectionLayer。如果在那里处理 POST 请求，我们就没有必要在 AppDelegate 中遍历两个层。

我们创建发送 POST 请求的方法并用 NSLog 记录其应答。接着，我们会创建 NSTimer 并以解析 JSON 应答结束。

13.5.2 创建 POST 请求方法

在 Xcode 中打开 FaceDetectionLayer.m 文件。导入 ASIFormDataRequest.h 和 AppDelegate.h 头文件。声明一个叫做 _sendingRequest 的私有布尔变量来跟踪何时处在一个 POST 请求过程中，然后创建一个叫做 facialRecognitionRequest 的带有一个 UIImage 参数的 void 方法。你更新后的 FaceDetectionLayer.h 文件应该与代码清单 13-19 类似。

代码清单 13-19 更新后的 FaceDetectionLayer.h

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"
#import "HUDLayer.h"
#import "ASIFormDataRequest.h"
#import "AppDelegate.h"

@interface FaceDetectionLayer : CCLayer {
    HUDLayer * hud;
    BOOL _sendingRequest;
}

+ (CCScene *)scene;
- (id)initWithHUD:(HUDLayer *)hud;
- (void)facialRecognitionRequest:(UIImage *)image;
@end
```

切换到 FaceDetectionLayer.m 文件。按代码清单 13-20 显示的代码创建方法。

代码清单 13-20 facialRecognitionRequest 方法

```
- (void)facialRecognitionRequest:(UIImage *)image {
    NSLog(@"Image width = %f height = %f",image.size.width, image.size.height);

    if (!_sendingRequest) {
        _sendingRequest = YES;

        NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

        NSData * imageData = UIImageJPEGRepresentation(image, 90);

        NSURL * url = [NSURL URLWithString:@"http://api.face.com/faces/detect.json"];

        ASIFormDataRequest *request = [ASIFormDataRequest requestWithURL:url];
        [request addPostValue:@"your face.com api_key" forKey:@"api_key"];
        [request addPostValue:@"your face.com api_secret" forKey:@"api_secret"];
        [request addData:imageData withFileName:@"image.jpg"
        andContentType:@"image/jpeg" forKey:@"filename"];

        [request startSynchronous];

        NSError *error = [request error];
        if (!error) {
            NSString *response = [request responseString];

            NSLog(@"Response: %@",response);
        } else {

            NSLog(@"Error: %d",[error code]);
        }

        _sendingRequest = NO;

        [pool drain];
    }
}
```

从这个方法的顶部开始，我们首先通过查看 `_sendingRequest` 布尔值来检查是否有另一个调用正在进行中。如果没有调用正在进行中，我们设置这个布尔值为 YES 以防止任何其他的调用（在单独的线程中的）与这次执行相冲突。

接下来，我们分配了一个 `NSAutoreleasePool`。`NSAutoreleasePool` 类（来自苹果开发者文档）是用来支持 cocoa 的引用计数型内存管理系统的。自动释放池会在被排干的时候向它存储的对象发送 `release` 消息。

在一个引用计数的环境里（与使用垃圾收集相反），`NSAutoreleasePool` 对象会容纳已经接收到 `autorelease` 消息的对象，并且在排干时向其中的每一个对象发送一个 `release` 消息。因此，向一个对象发送一个 `autorelease` 消息而不是 `release` 消息会延长这个对象的生命周期，至少到池自身被排干的时候（如果对象接着被保留了的话可能会更久）。一个对象可以被推入相同的池多次，每次被推入池都会收到一个 `release` 消息。

在 `NSAutoreleasePool` 就位后，我们创建了一个 `NSData` 对象来保存 JPEG 图像的 raw 图像数据。我们建立了用于 face.com 的 `NSURL` REST 目标、HTTP POST 值（其中之一是 raw 图像数据），并且开启了一个对于 face.com 的同步调用。

稍后我们将添加 JSON 解析逻辑。现在，我们只是记录下输出。

为了从 FaceDetectionLayer 发送 POST 请求，我们将需要在 AppDelegate 中引用这个类。如果你现在进行此操作，将会建立一个介于这两个头文件之间的循环依赖。这很容易解决，因为我们的方案还不是那么复杂。把 FaceDetectionLayer.h 文件中对于 AppDelegate.h 文件的 import 语句移动到 FaceDetectionLayer.m 文件中，然后我们就不会碰到这个问题了。如果你遇到一个以 expected specifier-qualifier-list before... 开头的编译错误，那就表示在这个工程的其他某个地方仍然存在循环引用。

我们开始创建 NSTimer，以及从 AppDelegate 向 FaceDetectionLayer 发送 UIImage 对象的过程，以便我们为 POST 请求做好准备。

13.5.3 创建 NSTimer

在 Xcode 中打开 AppDelegate.h 文件。创建一个叫做 _layer 的 FaceDetectionLayer 类型的私有变量，一个叫做 _timer 的 NSTimer 类型的私有变量。代码清单 13-21 显示的是我们对头文件的修改。

代码清单 13-21 对 AppDelegate.h 文件的修改

```
@interface AppDelegate : NSObject <UIApplicationDelegate,
AVCaptureVideoDataOutputSampleBufferDelegate> {
    ...

    FaceDetectionLayer *_layer;
    NSTimer *_timer;
}

...
```

切换到 AppDelegate.m 文件。找到代码清单 13-22 显示的那行。

代码清单 13-22 在 AppDelegate.m 文件中找到这一行

```
[[CCDirector sharedDirector] runWithScene: [FaceDetectionLayer scene]];
```

这一行使用 CCDirector 单例启动了一个默认的 FaceDetectionLayer 场景。在 FaceDetectionLayer 中有一个用来创建场景的静态方法。该方法使用 HUDLayer 来启动场景。为了防止管理太多的实例或者创建另一组单例类，我们将会从静态场景方法中移动一些功能到 AppDelegate 中。

使用代码清单 13-23 中的代码替换代码清单 13-22 中的行。

代码清单 13-23 以不同的方式启动 FaceDetectionLayer

```
CCScene *scene = [CCScene node];
HUDLayer *hud = [HUDLayer node];
[scene addChild:hud z:1];
_layer = [[[FaceDetectionLayer alloc] initWithHUD:hud] autorelease];
[scene addChild:_layer];

[[CCDirector sharedDirector] runWithScene: scene];
```

这段代码来自 FaceDetectionLayer 类的静态场景方法。如果你现在运行应用，它看起来应该差不多相同。你会看到摄像头视图以正确的方向占据了整个屏幕。

添加代码清单 13-24 中的方法到 AppDelegate.m 文件。

代码清单 13-24 添加背景方法

```
-(void)backgroundRequest:(UIImage *) image{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    UIInterfaceOrientation orient = [UIApplication
    sharedApplication].statusBarOrientation;
    UIImage * rotatedImage = image;
    switch (orient) {
        case UIInterfaceOrientationPortrait:
            NSLog(@"Device orientation portrait");
            break;
        case UIInterfaceOrientationPortraitUpsideDown:
            NSLog(@"Device orientation portrait upside down");
            break;
        case UIInterfaceOrientationLandscapeLeft:

            rotatedImage = [self rotateImage:image orientation:
            UIImageOrientationRight];
            NSLog(@"Device orientation landscape left");
            break;
        case UIInterfaceOrientationLandscapeRight:
            rotatedImage = [self rotateImage:image orientation: UIImageOrientationLeft];
            NSLog(@"Device orientation landscape right");
            break;
    };

    [_layer facialRecognitionRequest:rotatedImage];

    [pool drain];
}
```

这个方法将会是我们之前在 FaceDetectionLayer 类中创建的 facialRecognitionRequest 方法的调用者。我们再次使用 NSAutoReleasePool 来管理内存。接下来，我们检查了设备的方向。同样，我使用 Landscape Right 进行所有的测试，这种方法应该足以其他方向对图像进行适当的重定向。基本上，我们必须这么做以使我们发送到 face.com 的图像正好朝上。如果我们发送的图像是倾斜或者倒置的，face.com 就无法从图像中识别出面孔了。

当完成图像旋转后，我们发送更新后的图像对象到 FaceDetectionLayer 的 facialRecognitionRequest 方法。好，我们创建定时器回调，接着将创建 NSTimer。回调是每次定时器到期时触发的事件。这会是一个简单的方法，我们只需要每隔几秒在一个新线程中调用新的 backgroundRequest 方法即可。复制代码清单 13-25 中的方法到 backgroundRequest 方法的上面。

代码清单 13-25 定时器回调方法

```
-(void) timerCallback {
    [NSThread detachNewThreadSelector:@selector(backgroundRequest:) toTarget:self
    withObject:_imageView.image];
}
```

我们简单地分出一个新线程来运行 `backgroundRequest` 方法，并指定 `delegate` 为 `self`。

那么，这个过程的最后一步是开启计时器。在 `applicationDidFinishLaunching` 的最后一行我们开启了 `AVCaptureSession`。就在这一行的前面添加代码清单 13-26 中的代码。

代码清单 13-26 开启一个 5 秒的计时器

```
_timer = [NSTimer scheduledTimerWithTimeInterval:5.0 target:self
selector:@selector(timerCallback) userInfo:nil repeats:YES];
```

粗体代码显示的是在运行的过程中你应该在哪个地方重新设置计时器。如果你运行在一个非常慢的网络连接下，你可能需要增加这个设置的值。

如果现在运行这个工程，并且让它运行几秒，你应该会在调试控制台中看到与图 13-9 类似的结果。



图 13-9 输出显示了 face.com 请求的循环执行

13.5.4 解析输出

下一步我们需要解析来自 face.com 的输出。为了做到这个，我们将会使用本章早期从 GitHub 下载的 JSON-Framework（JSON 框架）库。

找到你解压下载文件的目录，有一个叫做 `Classes` 的子目录，打开那个目录。像在处理 ASI-HTTP-Request 库时一样，在 Xcode 中拥有良好的组织文件是我的偏好。在 Xcode 工程中我创建了一个叫做 `JSON` 的组，但这是没有必要的。从 `Classes` 目录复制所有的文件到你的 Xcode 工程中。确保你选中了 `Copy items into destination group's folder (if needed)` 选项。

在 Xcode 中打开 `FaceDetectionLayer.h` 文件并导入 `SBJson.h` 头文件。切换到 `FaceDetectionLayer.m` 文件并找到代码清单 13-27 中显示的代码行。

代码清单 13-27 找到这一段

```
if (!error) {
    NSString *response = [request responseString];
```

在代码清单 13-27 中的代码的后面，我们将会添加用于解析 JSON 应答的逻辑。在开始之前，我们再看看来自 face.com 的应答。代码清单 13-28 显示的是来自我的执行的示例应答。

代码清单 13-28 来自 face.com 的示例应答

```
{
  "photos":[
    {
      "url":"http://face.com/images/ph/90e76f377f93e949df78b552903a48b0.jpg",
      "pid":"F@6f518683514fac2eebe8d749b6121cc3_cd31b28498224cf9ccb39f9194569af8",
      "width":480,
      "height":360,
      "tags":[
        {
          "tid":"TEMP_F@6f518683514fac2eebe8d749b6121cc3_cd31b28498224cf9ccb39f9194569af8_62.50_41.11_0_0",
          "recognizable":true,
          "threshold":null,
          "uids":[
            {
              "gid":null,
              "label":"","
              "confirmed":false,
              "manual":false,
              "tagger_id":null,
              "width":28.75,
              "height":38.33,
              "center":{
                "x":62.5,
                "y":41.11
              },
              "eye_left":{
                "x":58.63,
                "y":30.98
              },
              "eye_right":{
                "x":71.52,
                "y":33.14
              },
              "mouth_left":{
                "x":58.34,
                "y":49.64
              },
              "mouth_center":{
                "x":64.25,
                "y":51.62
              },
              "mouth_right":{
                "x":69.34,
                "y":50.91
              },
              "nose":{
                "x":66.49,
                "y":44.14
              },
              "ear_left":null,
              "ear_right":null,
              "chin":null,
              "yaw":26.25,
              "roll":7.16,
              "pitch":-1.99,
              "attributes":{
                "glasses":{
                  "value":"false",
                  "confidence":16
                },
                "smiling":{
                  "value":"true",
```

```

        "confidence":96
    },
    "face":{
        "value":"true",
        "confidence":87
    },
    "gender":{
        "value":"male",
        "confidence":46
    },
    "mood":{
        "value":"happy",
        "confidence":51
    },
    "lips":{
        "value":"parted",
        "confidence":98
    }
    }
    }
    }
    },
    "status":"success",
    "usage":{
        "used":31,
        "remaining":4969,
        "limit":5000,
        "reset_time text":"Thu, 29 Sep 2011 02:40:10 +0000",
        "reset_time":1317264010
    }
}

```

粗体显示的代码段是本例中会用到的。让我们首先试着捕获状态值。

由于有了 SBJson 库，所以解析 JSON 是非常简单的。我们首先创建一个新的 SBJsonParser 对象，然后用我们从 face.com 获取来的 JSON 字符串创建一个 NSDictionary 对象。添加代码清单 13-29 中的代码到代码清单 13-27 中的代码的后面。

代码清单 13-29 识别 JSON 数组中的 Key

```

SBJsonParser *jsonParser = [SBJsonParser new];
NSDictionary *feed = [jsonParser objectWithString:response error:nil];
NSLog(@"RETURN: %@", [feed allKeys]);

```

由于我们能够转换 JSON 字符串为一个 NSDictionary，验证你期望在 NSDictionary 的 root 中找到的 key 与解析器解析出的字典的 root 中的 key 是否相匹配，这通常是一个好习惯。我们是通过 NSDictionary 的 allKeys 属性来做到这一点的。如果你在添加了上述代码后再次运行这个工程，你会在控制台中看到与图 13-10 相似的情景。

```

2011-09-28 19:40:10.240 Ch13[2353:5dc7] RETURN: (
    status,
    photos,
    usage
)

```

图 13-10 控制台显示的是 allKeys 属性显示的值

你可以看到 status 元素（我们正在寻找的）在字典的最顶层。因为这是一个单一元素，我们可以简单地获取这个 key 的值并记录到控制台。添加代码清单 13-30 中的代码到记录 allKeys 输出的代码的后面。

代码清单 13-30 记录状态信息

```
if ([[feed valueForKey:@"status"] isEqualToString:@"success"]) {
    NSLog(@"face.com request = success");
}
```

如果你再次运行工程，你会在控制台看到成功的消息。我们将继续添加代码到这个 if 代码块，并捕获识别的面孔的 x, y 坐标、宽度以及高度属性。

添加代码清单 13-31 中的代码到 success 代码块中。

代码清单 13-31 在图像中找到面孔的位置

```
double xPosition = [[[[[photo objectForKey:@"tags"] objectAtIndex:0]
valueForKey:@"center"] valueForKey:@"x"] doubleValue];
double yPosition = [[[[[photo objectForKey:@"tags"] objectAtIndex:0]
valueForKey:@"center"] valueForKey:@"y"] doubleValue];
NSString *mood = [[[[[photo objectForKey:@"tags"] objectAtIndex:0]
objectForKey:@"attributes"] objectForKey:@"mood"] valueForKey:@"value"];
NSDictionary *photo = [[feed objectForKey:@"photos"] objectAtIndex:0];
```

那么，这些代码行会解析剩下的 JSON 元素来找到我们在代码清单 13-28 中标为粗体的值。接下来我们将使用它们来建立 HUD 层。

13.5.5 构造 HUD 层

如果你是开发者的话，就应用本身而言是非常酷的。但是，从用户的角度来看，我们真的无法分辨发生了什么。我们将通过为本章中创建的 HUD 层添加一些反馈来改变这种状况。请记住，HUD 层是透明的，但它位于其他场景堆的顶部。因此我们可以在顶层上覆盖一切。

回想一下，我提到事情不会按预期的进行。这里就是我提到的那个部分的开始。如果你忽略它也可以，但是它是一个小的弯路；因此，建议你遵照代码，如果你愿意的话。

打开 HUDLayer.h 文件并声明代码清单 13-32 中显示的方法。

代码清单 13-32 加载 Crosshair 和 Mood 标签

```
- (void)loadCrosshair:(NSString *)mood x:(double)x y:(double)y;
```

切换到 HUDLayer.m 文件并实现代码清单 13-33 中的 loadCrosshair 方法。

代码清单 13-33 实现 loadCrosshair 方法

```
- (void)loadCrosshair:(NSString *)mood x:(double)x y:(double)y {
    CGSize size = [[CCDirector sharedDirector] winSize];

    CCLabelTTF *label = [CCLabelTTF labelWithString:@"test" fontName:@"Marker Felt"
fontSize:48];
    // position the label on the center of the screen
    label.position = ccp( size.width / 2 , size.height / 2 );
    // add the label as a child to this Layer
    [self addChild: label];

    CCSprite *crosshair = [CCSprite spriteWithFile:@"crosshair.png"
rect:CGRectMake(0,0,390,390)];
    crosshair.position = ccp((size.width * (x/100)), (size.height * (1 - (y/100))));
    [self addChild:crosshair];
}
```

在这个方法中，我们使用一些基本的函数来添加 PNG 文件到屏幕，并建立一个用于显示用户表情的标签。label 对象的 position 属性使用 cocos2D 的 ccp 宏来设置。这个宏为我们创建一个锚点，用来在屏幕上放置图像。我们设置锚点的位置为屏幕的中央（宽度的一半，高度的一半）。

下一步，我们从 PNG 文件加载了一个精灵（sprite）并放置到屏幕。face.com 通过其应答的一部分给予我们识别的面孔的位置（中心）。它是以屏幕的百分比来标识的，以便在你发送缩放图像的时候（如我们在本章做的）不必重新计算像素。由于百分数都是整数，我们把百分数除以 100，并在 face.com 识别面孔的相同位置为这个精灵创建一个锚点。

切换到 FaceDetectionLayer.m 文件并找到代码清单 13-31 中的代码（解析 face.com 请求的表情和 x, y 坐标）。添加代码清单 13-34 中的代码到这些代码行的后面。

代码清单 13-34 调用 loadCrosshair 方法

```
[_hud loadCrosshair:mood x:xPosition y:yPosition];
```

我们将会使用 HUD 层来调用新方法。如果你回想一下我们堆放视图的图表，工程应该可以良好工作。运行工程。结果应该与图 13-11 相似。

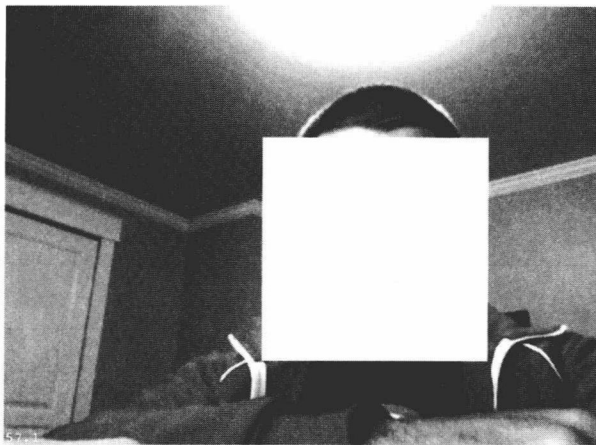


图 13-11 稍等，这是不对的

好消息是图像好像已经被放置到了正确的点上。坏消息是你无法看到图像。这似乎是屏幕上的一个不透明的块。

这取决于 OpenGL 怎么处理线程。我们分出一个线程来处理 HTTP 请求，然而 OpenGL 想要我们通过主线程与屏幕交互。OpenGL 处理线程有多种方法，并且有大量的关于这个主题的文章。这是在一个 cocos2D 应用中使用更有组织的线程的必要性的基本例子。如果你想学习更多关于 OpenGL 和线程的知识，可以参考 Apress 网站上 Mike Smithwick 的《Pro OpenGL for iOS》这本书。

让我们解决这个问题。事实上，这个应用不需要 HUD 层。由于我们的基本逻辑在

AppDelegate 中, FaceDetectionLayer 实际上是我们的 HUD 层。我们把绘图功能在堆中下降一个等级, 并使用主线程来引发屏幕上的标签的变化。

让我们添加一个属性以便从类的外部访问它, 而不是在运行时创建标签和精灵。同时, 我们需要一个属性来引用 RootViewController。

按代码清单 13-35 显示的代码调整 FaceDetectionLayer.h 文件。我注释掉了不再使用的代码行。

代码清单 13-35 修改 FaceDetectionLayer.h 文件

```
#import <Foundation/Foundation.h>
#import "cocos2d.h"
#import "HUDLayer.h"
#import "ASIFormDataRequest.h"
#import "SBJson.h"

@interface FaceDetectionLayer : CCLayer {
    //HUDLayer *_hud;
    BOOL _sendingRequest;

    CCSprite *_crosshair;
}

@property (retain) UIViewController * root;
@property (retain) CCLabelTTF * label;

//+ (CCScene *)scene;
//- (id)initWithHUD:(HUDLayer *)hud;
- (void)facialRecognitionRequest:(UIImage *)image;
@end
```

切换回 FaceDetectionLayer.m 文件并按代码清单 13-36 显示的代码合成新属性。

代码清单 13-36 合成新属性

```
@synthesize root;
@synthesize label = _label;
```

把 scene 和 initWithHUD 方法注释掉。我们不再使用这两个方法了。使用代码清单 13-37 中的代码创建一个新的 init 方法。

代码清单 13-37 新的 init 方法

```
-(id) init {
    if(self = [super init]){
        CGSize size = [[CCDirector sharedDirector] winSize];

        crosshair = [CCSprite spriteWithFile:@"crosshair.png" ];
        crosshair.position = ccp((size.width * (50.0/100)), (size.height * (1 -
(50.0/100))));
        crosshair.opacity = 0;
        [self addChild:crosshair];

        _label = [CCLabelTTF labelWithString:@"" fontName:@"Marker Felt" fontSize:48];
        label.position = ccp( size.width / 2 , size.height/2 );
        [self addChild: _label];
    }
    return self;
}
```

我们简单地移动 HUDLayer 中的一些代码到 FaceDetectionLayer 的 init 方法中。在这个代码中的唯一真正的区别是我们只是没有设置标签的字符串。我们稍后将在 RootViewController 中做这个。

用代码清单 13-39 中的代码替换代码清单 13-38 中的代码行。

代码清单 13-38 找到这个

```
[_hud loadCrosshair:mood x:xPosition y:yPosition];
```

代码清单 13-39 用这个替换

```
CGSize size = [[CCDirector sharedDirector] winSize];
crosshair.opacity = 255;

crosshair.position = ccp((size.width * (xPosition/100)), (size.height * (1 -
(yPosition/100))));
[root performSelectorOnMainThread:@selector(updateMood:) withObject:mood
waitUntilDone:YES];
```

我们正使用 ccp 宏移动精灵，并把标签的更新发送回主线程，而不是向 HUDLayer 类发送请求。updateMood 函数还不存在。我们接下来会创建它。

打开 RootViewController.h 文件。导入 FaceDetectionLayer.h 头文件并添加代码清单 13-40 显示的属性和方法。

代码清单 13-40 更新 RootViewController

```
#import <UIKit/UIKit.h>
#import "FaceDetectionLayer.h"

@interface RootViewController : UIViewController {

}
@property (retain) FaceDetectionLayer* fdLayer;
-(void)updateMood:(NSString*)mood;
@end
```

切换到 RootViewController.m 文件并合成我们刚刚创建的属性。接着创建代码清单 13-41 中的方法。

代码清单 13-41 updateMood 方法

```
-(void)updateMood:(NSString *) mood {
    [[[self fdLayer] label] setString:mood];
}
```

在 Xcode 中打开 AppDelegate.m 文件。用代码清单 13-42 中的代码更新 applicationDidFinishLaunching 方法。

代码清单 13-42 更新 applicationDidFinishLaunching 方法

```
...
// Removes the startup flicker
[self removeStartupFlicker];
// Run the intro Scene
```

```
CCScene *scene = [CCScene node];
_layer = [[[FaceDetectionLayer alloc] init] autorelease];
[scene addChild:_layer];
viewController.fdLayer = _layer;
_layer.root = viewController;
[[CCDirector sharedDirector] runWithScene: scene];

_timer = [NSTimer scheduledTimerWithTimeInterval:5.0 target:self
selector:@selector(timerCallback) userInfo:nil repeats:YES];
[self setupCaptureSession];
}
```

我们正使用新的 init 方法来实例化 FaceDetectionLayer，而不是运行 initWithHUD 方法。在这个代码片段中的另一个不同是我们设置 viewController（RootViewController）的 fdLayer 属性为 FaceDetectionLayer。

如果你按现状运行应用，在请求被解析后，你会在屏幕上看到正确的覆盖物和表情标签。在这一部分我获得了我的孩子的帮助。结果显示在图 13-12 ~ 图 13-14。

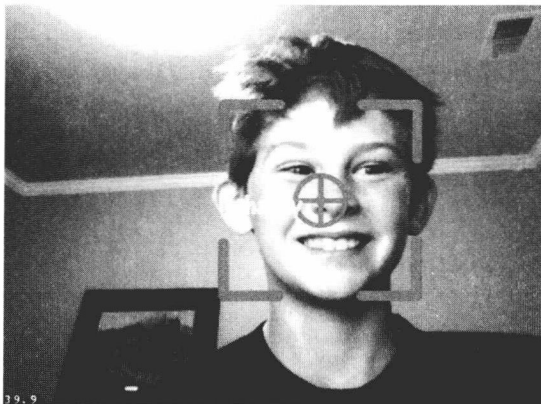


图 13-12 Aodhan 是高兴的



图 13-13 Avery 生气了



图 13-14 Avery 感到惊讶

我们要添加到应用中的最后一个功能是 SMS 功能，以便在检测到生气或者伤心的面孔的时候提示客户服务。

如果你想扩展这个例子以做练习的话，请记住，在摄像头视图中可能有不止一个人。你需要做的唯一的调整是创建一个遍历从 face.com 的 JSON 应答返回的所有照片的循环。我们硬编码 objectAtIndex:0 来调出识别的第一个面孔。因此，如果你想分析人群的话，这个地方将需要做下调整。

13.6 添加一个 Twilio 调出

Twilio 使用 REST 来初始化一个呼出电话或者 SMS。我们使用 SMS 的原因之一是它只需要 REST 调出。我们没有必要在一个网络服务器上寄放任何代码。添加代码清单 13-43 中的方法到 FaceDetectionLayer.m 文件。

代码清单 13-43 发送 SMS

```
- (void)sendSMS {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    NSLog(@"Sending request");
    NSString *accountSid = @"Your Account SID Here";
    NSString *authToken = @"Your AuthToken Here";
    NSString *urlString = [NSString
stringWithFormat:@"https://%@:%%@api.twilio.com/2010-04-01/Accounts/%@/SMS/Messages",
accountSid, authToken, accountSid];

    NSURL *url = [NSURL URLWithString:urlString];

    ASIDataRequest *request = [ASIDataRequest requestWithURL:url];
    [request addPostValue:@"Your TO Number Here" forKey:@"To"];
    [request addPostValue:@"Angry Face Detected" forKey:@"Body"];
    [request addPostValue:@"Valid Twilio Number" forKey:@"From"];

    [request startSynchronous];

    NSError *error = [request error];
    if (!error) {
        NSString *response = [request responseString];
        NSLog(@"%@", response);
    } else {
        NSLog(@"An error occured %d; %@", [error code], [request responseString]);
    }

    [pool drain];
}
```

这段代码非常简单。如你刚刚看到的，Twilio 有一个非常简单的 API 来向你的应用添加核心价值。你可以在 facialRecognitionRequest 方法中引用这个新功能。只要把它封装在一个用于匹配生气表情的快速检查中，你就都设置好了。

图 13-15 显示的是 SMS 提示。



图 13-15 SMS 发送成功了

13.7 总结

本章充满乐趣。与在第7章做的一样，我们首先创建了一个 HUD 层。在经过一些实验后，我们意识到那是没有必要的。通过一步一步的过程，我们学习了关于 OpenGL 的线程（cocos2D 的基础）。

我们扩展例子来创建一个 AVCaptureSession，并发送 UIImage 到 face.com REST API。我们使用两个不同的开源库来向我们的示例添加功能。ASI-HTTP-Request 用一小段代码帮助我们实现 REST 调出封装，SBJson 库帮助我们解析来自 face.com 的应答。

我们使用 cocos2D 来在目标面孔上放置一个精灵和一个带着来自 face.com 的表情的标签。最后，在检测到一个生气面孔的时候，我们使用 Twilio 的 REST API 通过 SMS 来提示客户服务。

在你使用面部识别例子创建应用之前，你应该检查所在地的隐私保护法。在某些地区，捕获或者存储脸谱可能会侵犯隐私权。

我希望在本书的例子中你学到了很多。如果你有任何问题或者反馈，请随时与我联系，可通过我的 twitter 地址 @kylemroche 或者 GitHub。