# 第14章 CocosBuilder和Shader

Cocos2D官方支持的可视化图形界面编辑工具是CocosBuilder，使用该工具能够非常方便地编辑游戏中的菜单、设置等游戏场景，这是Cocos2D唯一的官方编辑器。CocosBuilder采用插件式方式组织，并且采用MIT开源协议，读者可以下载其源代码，贡献自己的插件，或者修改现有插件以适用游戏的需要。此外，本章还会介绍Cocos2D 2.0独有的新特性着色器（Shader）。

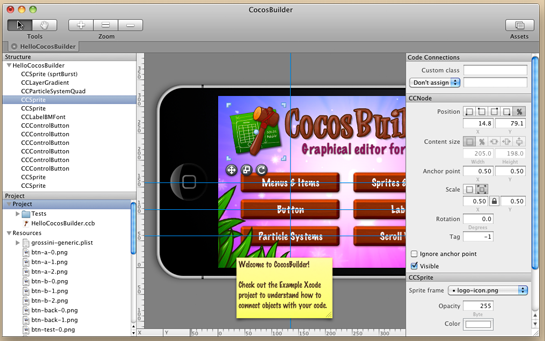
## CocosBuilder初体验

以前，开发者们在制作游戏场景的时候，主要采用手动计算场景元素坐标，然后不停地“test and try”。因为游戏设计师的想法是多变的，而且随着游戏功能的添加和需求的变化，经常需要调整或者修改游戏场景元素。这给游戏开发工作增加了难度，而且“test and try”工作特别枯燥乏味，所以很多公司也开始制作自己的游戏场景编辑器。不过并不是每一家公司都有能力开发自己的编辑器，后面陆续出现了一些开源的场景编辑器比如Cocoshop。但是这个工具的功能不够强大，满足不了复杂游戏场景编辑的需要。

这时候CocosBuilder诞生了，刚开始出现就受到社区的关注且是采用MIT开源协议。到1.1版本的时候，作者就不开源了，只在GitHub上保留1.0版本。后来Zynga与Riq和该作者合作，CocosBuilder开始飞速发展。

### 下载安装CocosBuilder

CocosBuilder是专门为Cocos2D制作的一款免费的可视化图形编辑工具，它可以帮助布局管理游戏中的精灵、层和场景。使用它可以非常快速地制作出游戏中的菜单等游戏场景，但是，它的功能并不止于此，例如使用它制作粒子系统，也可以把它当作关卡编辑器来使用。图14-1所示是CocosBuilder 2.0的编辑状态。



1. CocosBuilder 2.0编辑状态的示意图

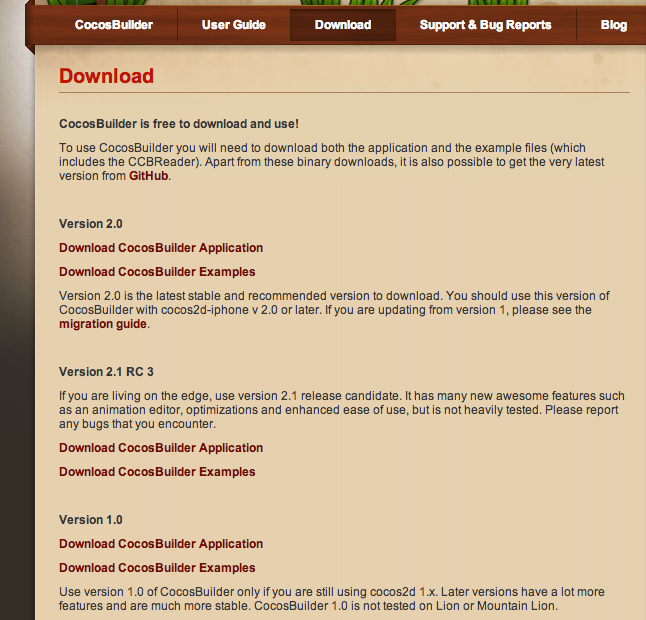
在游戏中引用CocosBuilder制作场景非常简单，只需要把所有CCBReader添加到工程中，再添加导出的ccbi文件，最后在代码中只需要简单的一行代码就可以显示这些场景了。

CocosBuilder和Cocos2D配合相当完美，只需要在编辑器中的custom classes和assign variable里输入定制的类名和变量名，然后编写定制类和变量，最后通过CCBReader读取ccbi文件的时候，就会自动把变量关联到界面元素上去。这个过程听起来可能有点抽象，后面会有详细的示例程序加以说明。

目前，最新版本CocosBuilder 3.0也正在开发当中，其中一个核心功能就是Action和Animation编辑器，类似于Flash的key frame。还有就是支持js来开发游戏原型。随着Cocos2D js-binding的逐渐成熟，相信新的功能特性可以大大提高开发效率。这个版本目前也可以在GitHub中获取，clone完仓库之后，切换到develop分支中就可以获取源码了。感兴趣的读者可以自行下载源码去编辑运行体验。

下载安装的步骤如下。

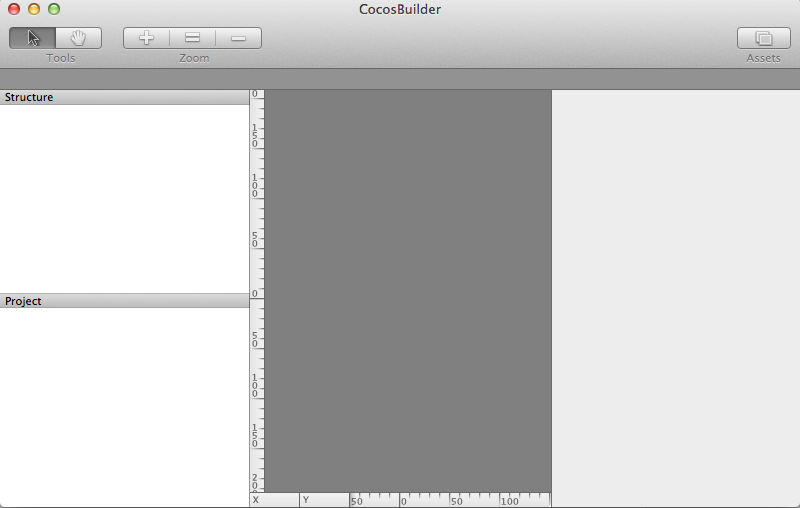
打开浏览器，在地址栏中输入地址<http://cocosbuilder.com>，在页面的菜单部分找到Download，单击打开，将会得到如图14-2所示输出。



1. 下载安装界面

注意 该下载页面同时提供了1.0、2.0和2.1RC3 版本。但是，推荐大家下载2.0正式版本，因为它兼容Cocos2D 2.0及更高版本。1.0虽然说是目前最稳定的版本，但是只有Snow Leopard上面测试过，Lion和Mountain Lion尚未测试。最重要的是1.0版本只支持Cocos2D 1.x系列版本。在学习完本章之后，读者可以自行下载1.0版本测试。

下载完以后，运行CocosBuilder程序，会得到图14-3所示界面。

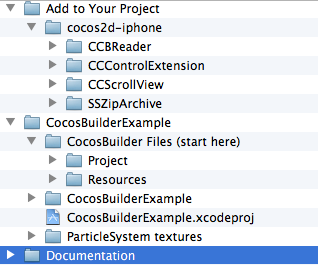


1. 运行CocosBuilder程序

工具的启动界面看起来并无特别之处：左边上半部分是每一个场景的界面元素结构，下方是项目文件及资源列表，中间灰色部分是场景的编辑工作区，右边类似于Xcode的Inspector界面，用来设置场景中元素的各种属性。

### 查看Example目录结构

在介绍该工具的使用之前，先打开下载的Example，目录结构如图14-4所示。



1. Example目录结构

这些样例需要重点研究和学习，CocosBuilder目前的文档资料极其匮乏，所以本章也将借助这些样例介绍CocosBuilder的使用。

目录包含了使用CocosBuilder时需要引用的文件（Add to your project下面的所有文件），一个综合示例（CocosBuilderExample目录），以及5个简单的文档。

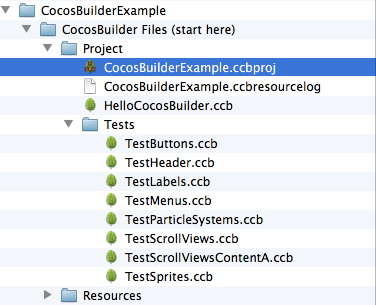
### 如何打开样例文件

双击CocosBuilderExample.xcodeproj文件，编译并运行，将会得到如图14-5所示输出。



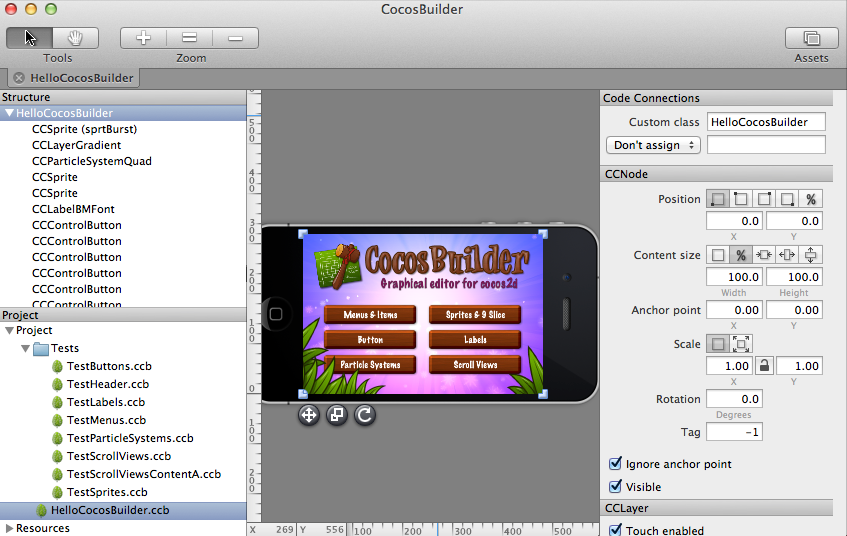
1. CocosBuilderExample.xcodeproj编译运行

双击CocosBuilderExample.ccbproj文件，默认用CocosBuilder打开。如果没有自动打开也不要紧，选择CocosBuilder的菜单File→Open打开工程，如图14-6所示。



1. 打开工程

打开文件以后，CocosBuilder的窗口界面就会有一些非常明显的变化，如图14-7所示。



1. CocosBuilder的窗口界面变化

之前介绍过CocosBuilder的界面元素布局，现在应该有了更直观的了解了。大家可以尝试着修改这些界面元素的位置和属性，修改完以后马上就可以在CocosBuilder主界面看到效果，这即是所谓的“所见即所得”编辑器的好处。从此再也不用为了布局某个按钮而不断地猜测坐标了。

选中“Graphical editor for cocos2d”标签，同时在右边的Inspector里把内容修改为“Hello World CocosBuilder！”。最后，从菜单中选择File→Publish。此时，会把修改打包到ccb.zip中去。回到Xcode，编译并运行，效果如图14-8所示。



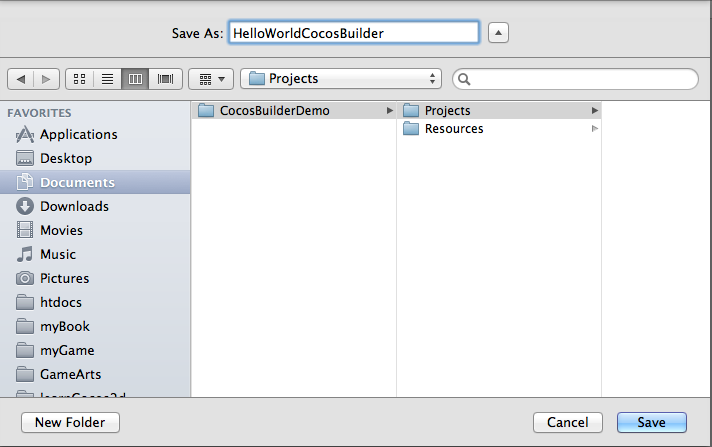
1. 修改后编译运行输出

## 第一个CocosBuilder项目

为了帮助大家更好地理解和学习CocosBuilder，这里将采用Example自带的资源文件制作一个示例项目。通过此示例项目制作过程的学习，读者再回过头去看此Example工程，相信会有更大的收获。

### 创建HelloWorldCocosBuilder场景

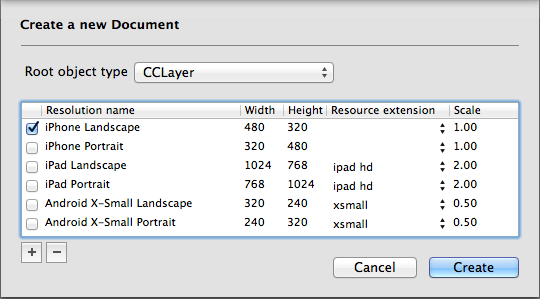
首先，打开CocosBuilder，选择File→New Project新建一个工程，取名为“HelloWorldCocosBuilder”，把该项目保存到与其资源文件夹平行的Projects文件夹中，如图14-9所示。



1. 新建工程

#### 新建文档对象

这时候，选择File→New File新建一个文档对象，此时会出现图14-10所示对话框。



1. 新建文档对象

#### 选择模板并保存

这里预先定义了多种不同的模板供选择，有iPhone、iPad，甚至还有Android（因为CocosBuilder输出的格式可以被Cocos2D-X使用，而Cocos2D-X支持开发跨iOS和Android平台的手机游戏）。这里选择iPhone Landscape，同时“Root Object type”选择为CCLayer，因为这是第一个要加载的场景。这个列表中还有CCNode、CCSprite和CCParticleSystemQuad供选择。选择确认以后，单击Create。这时候又会有一个对话框出现，提示选择保存文件的路径。在保存对话框中键入HelloWorldCocosBuilder，单击Save，这时候CocosBuilder如图14-11所示。



1. 选择模板并保存

如图所示，左上角出现新对象CCLayer，下面的Project则显示刚刚新建的文档，中间是一个水平放置的iPhone模拟器（因为这里选择的是iPhone Landscape）。

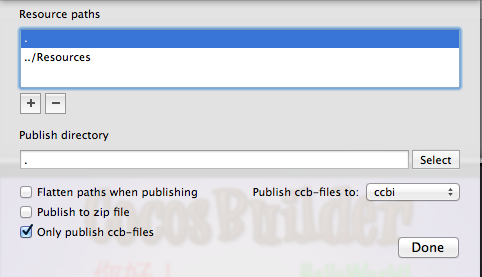
右边的Inspector界面包含3部分，分别是Code Connections、CCNode和CCLayer。

Code Connections部分是每种类型的共有属性，用来指定代码与文档对象之间连接方式。而下面的几个面板是根据新建对象的不同而变化。这些数据与对象的继承层次有关，这里建立的是CCLayer，它继承自CCNode。因此只有两个面板，上面的面板显示与CCNode对象相关的属性设置，而下面的面板则显示与CCLayer相关的属性设置。此时在Code Connections的Custom class中输入HelloWorldCocosBuider，就可以在xcode工程里自定义一个HelloWorldCocosBuider类来与此文档场景建立关联了。

此时，还需要把资源文件也添加进来。

#### 选择资源路径和发布目录

选择File→Project settings，然后选择资源路径和发布目录，如图14-12所示。



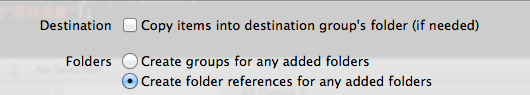
1. 选择资源路径和发布目录

注意 这里一定要选择图片资源路径，否则，在创建场景文档的时候会没有图片资源可供选择，那也就成了“巧妇拿为无米之炊了”。图中的Publish directory为一点，则表明生成的ccbi文件的发布路径与工程文件在同一个目录下面，这与工程设置里默认的资源路径是一样的。因此，这里也就有个小技巧。在新建CocosBuilder工程的时候，可以直接选择把工程文件路径保存到与图片资源同一个目录下面，这样就无需再另外选择图片资源路径了。

几个复选框的作用如下：

##### Flatten paths when publishing

当资源文件下面有子目录的时候，比如mySubDirectory/myImage.png这样的路径，如果勾选中这个选项。那么在发布的ccbi文件中，就会转化成myImage.png，也即取消文件路径，只保留文件名。这时候，往Xcode项目中新增资源文件的时候，可以直接添加黄色文件夹，这个是默认的方式。如果不勾选此复选框，则在Xcode中添加资源的时候需要添加蓝色文件夹。想要添加蓝色文件夹，只需要在添加文件夹的时候，按照下图所示方式，选择“Create folder references for any added folders”添加即可，如图14-13所示。



1. 选择“Create folder references for any added folders”

##### Publish to zip files

此选项可用来打包生成的ccbi文件为一个ZIP压缩包。它的好处是可以减少游戏资源的大小。

##### Only publish ccb-files

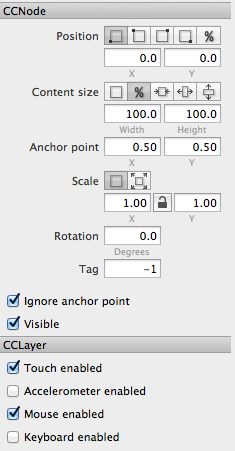
此选项的作用即是只发布生成ccbi文件，不发布图片资源文件。如果不选中此选项，那么在发布工程的时候，会把所有的Resource目录下的图片和位图资源全部发布出来。

在本示例项目中采用的方式是手动在Xcode添加资源图片引用（黄色文件夹），所以只需要勾选“Only publish ccb-files”选项即可。因为资源文件夹中不存在子目录，所以也无需勾选“Flatten paths when publishing”选项。

项目基本设置好，就可以使用CocosBuilder制作游戏场景了。

#### 制作游戏场景

单击HelloWorldCocosBuilder节点，设置右边的属性如图14-14所示。



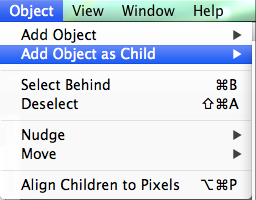
1. 属性设置

这里一定要设置Content Size，选择中间的“%”，然后Width和Height都输入100，即100%。因为之前在新建文档对象的时候选择的是iPhone Landscape，所以100%默认的大小也就是iPhone Landscape状态下的大小（480\*320）。此时单击Content Size的其他选项，再结合每一个选项上面的tooltip，就可以知道这些选项的意思了。

查看tooltip的方式非常简单，把鼠标停留在每个选项上面，过一会就会出现提示信息了。通过tooltip可以非常快速地了解和学习所有属性面板的作用。大家在实际操作过程中，一定要多动手，多体会这些属性的作用与用法。在属性面板里修改的内容，在中间的面板中会马上得到体现。这种所见即所得地方式会极大地提高制作游戏场景的效率，也能加快对此工具的学习。

提示 基本上每新建一个Root Object文档都需要设置根节点的ContentSize，如果读者在制作场景的过程中遇到一些奇怪的坐标显示问题，也可以检查是否有正确设置ContentSize。

选中此根节点，从菜单中选择Object→Add Object as Child，如图14-15所示。



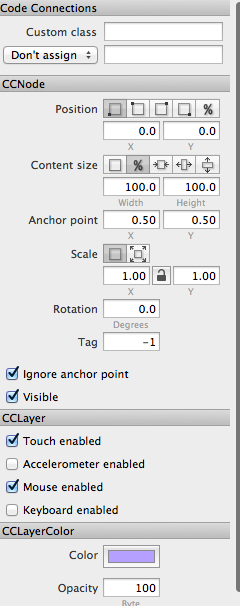
1. 新建Root Object文档

首先，由于根结点只是一个容器，背景完全是黑色的，显示极不好看。所以首先添加一个CCLayerColor作为子结点。它们的父子关系可以通过显示列表中的右递进来体现，如图14-16所示。



1. 显示列表中的右递进

此时，选中CCLayerColor，把右边的属性设置为如图14-17所示。



1. 右边的属性设置

提示 这里也有个小技巧，就是设置此图层全屏显示的时候，把ContentSize选择为%，然后把Width和Height都设置为100就可以了。

最后把颜色修改成淡紫色，同时把透明度（Opacity）设置成100。

一个紫色背景的基本游戏场景就制作好了。虽然很简单，但这是一个好的开端。保存工程，同时选择File→Publish，把发布的文件名取名为HelloWorldCocosBuilder，然后选择该项目的保存目录。此时，会在Projects目录下面生成一个HelloWorldCocosBuilder.ccbi文件。

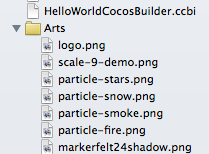
接下来将在Xcode项目中引用此文档并显示出来。

### 加载HelloWorldCocosBuilder场景

打开Xcode，新建一个Cocos2D 2.0的工程，把HelloWorldLayer.h和HelloWroldLayer.m两个文件都删除。同时新建一个HelloWorldCocosBuilder类，继承自CCLayer。最后修改IntroLayer.m文件，把头文件包含里的“#import "HelloWorldLayer.h"”删除掉，同时把makeTransition此函数内部的代码全部删除。

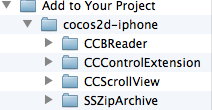
此时，编译并运行，确保工程没有错误再继续。

接下来先把发布的HelloWorldCocosBuilder.ccbi文件添加到Resource分组下。然后把相关的图片资源也全部加载进来，如图14-18所示（图片资源比较多，只截取部分图）。



1. 相关的图片资源

此时 需要添加ccbi解析类CCBReader。类文件在示例目录下面，如图14-19所示。



1. ccbi解析类CCBReader位置

这几个类文件夹对于使用CocosBuilder都非常有帮助。这里先不管其他类，只添加CCBReader到工程中。

注意 选择CCBReader.h和CCBReader.m文件，而不要选择整个文件夹。因为这样引用头文件的时候会非常方便，而不用带路径信息。

此时，打开IntroLayer.m文件，新添“#import "CCBReader.h"”头文件，然后把makeTransition函数用以下代码替换：

CCScene\* scene = [CCBReader sceneWithNodeGraphFromFile:@"HelloWorldCocosBuilder.ccbi"];

[[CCDirector sharedDirector] replaceScene:[CCTransitionFade transitionWithDuration:1.0

scene:scene

withColor:ccWHITE]];

编译并运行，背景为紫色，效果如图14-20所示，。

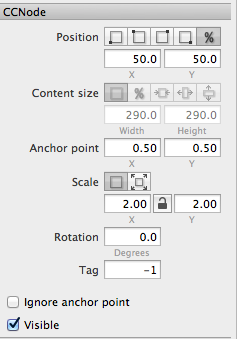


1. 加载场景

### 建立文档与代码的连接

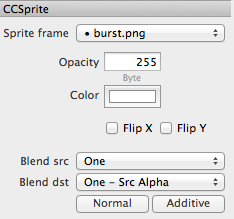
上面只是通过简单的一句代码调用，就把CocosBuilder生成的场景加载并显示出来。接下来要在CocosBuilder场景中创建一个精灵图片，然后在代码里定义一个精灵变量，并与之建立关联。同时在代码里对此精灵对象进行操作。

首先，回到CocosBuilder，为HelloWorldCocosBuilder新添加一个子节点，类型为CCSprite。注意，刚添加的精灵会出现在模拟器的左下角，选中精灵左下角的一个十字架按钮就可以任意拖动进行布局。如果想让它居中显示，直接把位置设置成相对容器的中点，即两个50%，如图14-21所示。



1. 精灵的布局

此时，为CCSprite指定一个纹理图片，这里选择的是一个burst.png，如图14-22所示。



1. 指定纹理图片

上图是精灵特有的属性选择，除了Sprite frame之外，其他几个选项通常保留默认的。此时的效果图如图14-23所示。



1. 添加纹理图片效果

这显然不是最终效果。只需要更改新建的精灵对象和CCLayerColor对象的顺序即可。更改之后如图14-24所示。



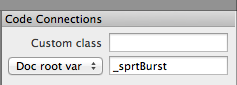
1. 更改新建的精灵对象和CCLayerColor对象的顺序

顺序的作用就是z order的作用。越靠近HelloWorldCocosBuilder节点，z值越小，反之则越大。因此，交换位置以后，效果如图14-25所示。



1. 位置交换以后的效果

我们需要在代码里得到此精灵对象，并且要控制它不停地旋转。因此，把CCSprite的Code connection设置成如图14-26所示。



1. Code connection设置

注意 有两种类型的建立方法，一种是Doc root var，另一种是owner var。Doc root var是根节点关联的文档类的属性对象，即HelloWorldCocosBuilder类里定义的成员变量。

此时，保存并发布。这时候会重新生成HelloWorldCocosBuilder.ccbi文件。接下来，只需要在代码里实现旋转精灵的功能就可以了。

打开Xcode，找到HelloWorldCocosBuilder.h文件，新添一个CCSprite\* \_sprtBurst成员变量。然后在HelloWorldCocosBuilder.m里，定义一个新的方法didLoadFromCCB。在这个方法里可以初始化和操作场景中的元素。didLoadFromCCB方法的实现如下所示：

[\_sprtBurst runAction:[CCRepeatForever actionWithAction:[CCRotateBy actionWithDuration:4.0 angle:360]]];

此时编译并运行，效果如图14-27所示。



1. 旋转的精灵对象

此时，会看到刚刚定义的精灵对象在永无止境地旋转着。

### 完善HelloWorldCocosBuilder场景

本节添加更多的精灵，以及一些标签和按键，进一步完善HelloWorldCocosBuilder场景。

#### 添加一个logo精灵

打开CocosBuilder，添加一个logo精灵。同时选择Sprite frame为logo.png，效果如图14-28所示。



1. 选择Sprite frame为logo.png

#### 添加两个标签

添加两个标签，一个是CCLabelTTF，一个是CCLabelBMFont。标签的属性设置没什么讲究的，关键是选对字体、字体大小等。具体新建的效果图如图14-29所示。



1. 添加标签

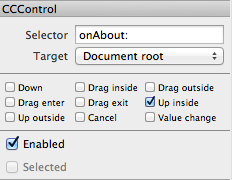
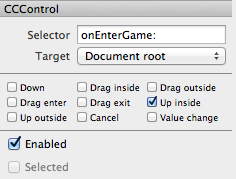
#### 添加两个按钮

接下来再添加两个CCControlButton类型的按钮，如图14-30所示。



1. 添加两个按钮

既然是按钮，那么一定要定义按钮响应函数。可以在CCControlButton的Inspector界面看到，分别定义如图14-31所示。



1. CCControlButton的Inspector界面

#### 锦上添花

最后再给场景添加一点草作为点缀，最终效果图如图14-32所示。



1. 场景添加一点草点缀

接下来，要在代码中加载此场景。注意，由于这里使用新的控制CCControlButton类，所以需要把之前的CCControlButton类库添加进来。同时在HelloWorldCocosBuilder类文件中声明两个函数：onEnterGame和onAbout。

#### 加载场景

打开HelloWordCocosBuilder.m，添加下面两个函数：

-(void) onEnterGame:(id)sender{

CCLOG(@"onEnterGame!");

}

-(void) onAbout:(id)sender{

CCLOG(@"onAbout");

}

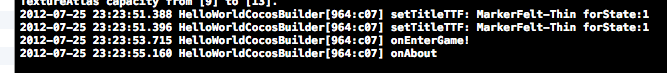
此时，编译并运行工程，效果如图14-33所示。



1. 声明两个函数后编译运行

提示 图14-33与图14-32是一样的。注意这两张图的背景有细微差别。图14-32是静态的编辑器中的状态，背景是场景编辑器。图14-33是代码运行之后动态的效果,图背景是代码。

按下“进入游戏”和“关于”按钮时，会触发HelloWorldCocosBuilder类中的回调函数onEnterGame和onAbout。这时候，控制台输出如图14-34所示。



1. 触发HelloWorldCocosBuilder类回调函数

到目前为止，一个简单的CocoBuilder场景就制作完成了。接下来，带领大家走进丰富多彩的CocosBuilder世界。如多个ccbi文件调用、文档对象包含ccbi文件，以及一些场景跳转的功能实现。

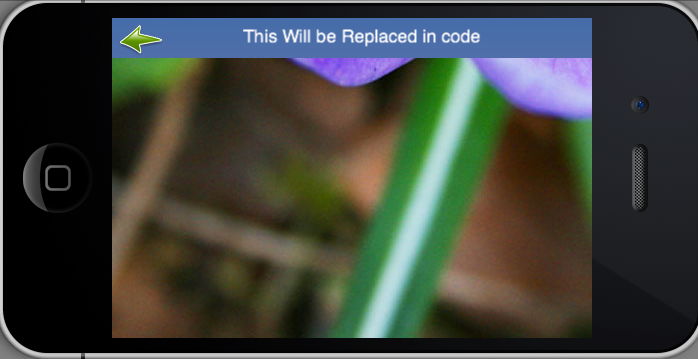
## HelloWorldCocosBuilder项目功能完善

HelloWorldCocosBuilder.ccb场景实现了两个按钮，接下来，要为这两个按钮提供两个场景，一个是GameMenu场景，最终效果图如图14-35所示。



1. GameMenu场景最终效果图

另一个场景是About场景，这里实现了一个ScrollView，用来滚动显示一张大图片，最终效果图如图14-36所示。



1. About场景最终效果图

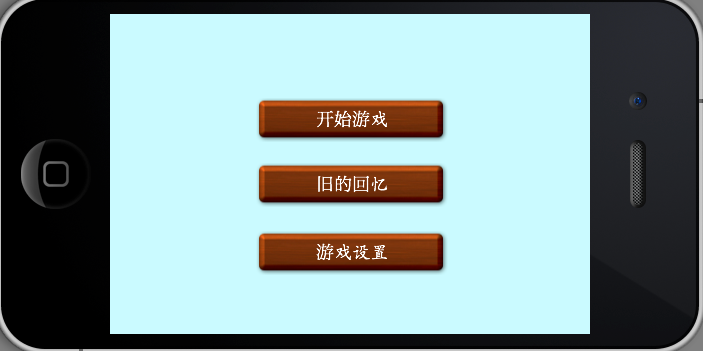
这两个场景上方都有一个返回按钮，以及一段“This will be Replaced in code”标语。这其实也是一个CCB文件。通过这两个场景的制作过程，读者可以学习到以下知识：

* 如何制作多个CCB文件并相互调用。
* CCScrollView的使用。
* 如何关联owner var变量，以及如何在场景中使用CCBFile对象。

### 制作GameMenu场景

和制作HelloWorldCocosBuilder场景一样，首先新建一个Root Object类型为CCLayer的文档对象，在Custom Class中取名为GameMenu，并且设置contenSize为100%\*100%。

接下来新建一个CCLayerColor并把contentSize也设置为100%\*100%，并且颜色设置为淡蓝色。然后 纵向放置3个CCControlButton，效果如图14-37所示。



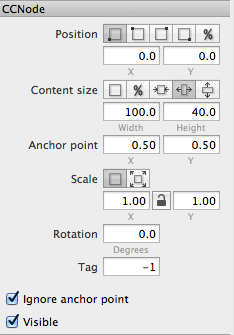
1. 3个CCControlButton纵向排列

为这3个按钮关联3个回调函数，分别为onStartGame:、onRecordGame:和onGameSettings:，同时这3个回调函数的Target都设置为DocumentRoot，就可以在定制类GameMenu中定义这3个回调函数了。

单击HelloWorldCocosBuilder场景中的“进入游戏”按钮时，初始化GameMenu场景，并跳转该页面。此时，需要有某种途径返回之前的场景。所以，这里介绍一种重复使用的方案——CCBFile对象。

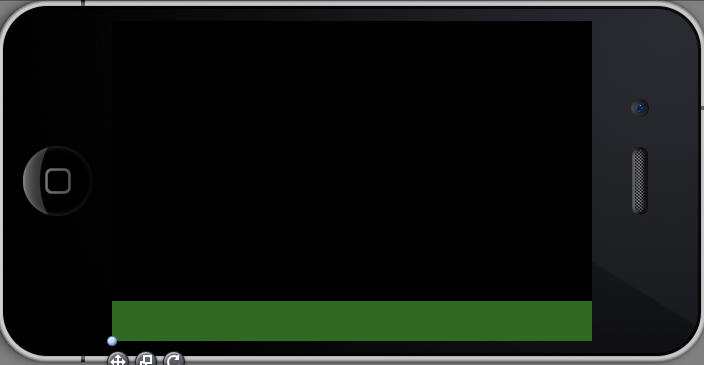
1. 创建一个Root Object类型为CCLayer的文档。

单击File→New File创建一个Root Object类型为CCLayer的文档。在Custom class中输入TitleBar，同时把contentSize设置为100%的宽度，40像素的高度。具体属性设置如图14-38所示。



1. 属性设置
2. 新建一个CCLayerColor对象。

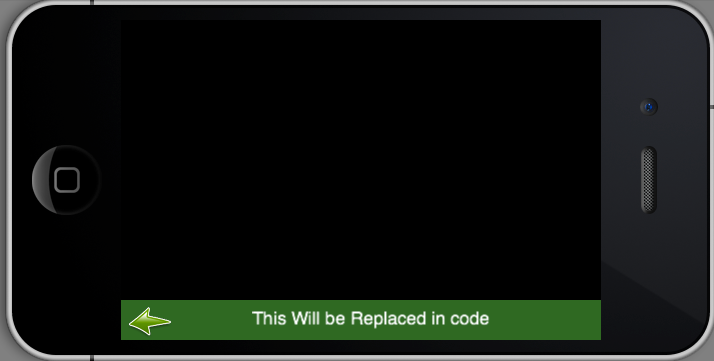
ContentSize设置为100%\*100%，这时候的屏幕输出如图14-39所示。



1. 新建CCLayerColor对象

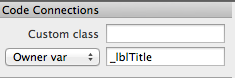
大家一定很奇怪，为什么CCLayerColor对象没有填充整个屏幕，而是显示40像素高度。这与之前设置的根节点的ContentSize有关。所有子节点的ContentSize的百分比设置都是相对于父节点的。

1. 新建一个CCLabelTTF和一个CCMenu，如图14-40所示。



1. 新建一个CCLabelTTF和一个CCMenu

为了让不同的场景在引用此文档时，可以改变CCLabelTTF的值，这里为CCLabelTTF设置lblTitle变量，同时基类型为Owner var，如图14-41所示。



1. CCLabelTTF设置lblTitle变量

同时，为CCMenu对象添加一个CCMenuItemImage对象，选择一个返回箭头作为显示图片。为其指定回调函数为onBack:，Target依旧是Document root。

1. 添加一个CCBFile孩子节点。

双击“Projects”面板中的GameMenu.ccb文件，选择菜单中的Object，为其添加一个CCBFile孩子节点，效果如图14-42所示。



1. 添加一个CCBFile孩子节点
2. 打开Xcode，找到HelloWorldCocosBuilder.h，添加一个新的成员变量：

CCLabelTTF \*\_lblTitle;

1. 打开HelloWorldCocosBuilder.m，添加一个新的函数，如下所示：

-(void) goCCBScene:(NSString\*)ccbName{

CCScene \*scene = [CCBReader sceneWithNodeGraphFromFile:ccbName owner:self];

[\_lblTitle setString:ccbName];

[[CCDirector sharedDirector] replaceScene:scene];

}

1. 把onEnterGame函数用以下代码替换之：

-(void) onEnterGame:(id)sender{

CCLOG(@"onEnterGame!");

[self goCCBScene:@"GameMenu.ccbi"];

}

1. 新建GameMenu和TitleBar两个类。

这两个类都继承自CCLayer，同时在GameMenu中实现之前定义的3个回调方法。而TitleBar类中也要实现onBack方法。这4个回调方法的实现如代码清单14-1所示。

1. 4个回调方法的实现

-(void) onRecordGame:(id)sender{

CCLOG(@"旧的回忆");

}

-(void) onStartGame:(id)sender{

CCLOG(@"开始游戏");

}

-(void) onGameSettings:(id)sender{

CCLOG(@"游戏设置");

}

-(void) onBack:(id)sender{

CCScene \*scene = [CCBReader sceneWithNodeGraphFromFile:@"HelloWorldCocosBuilder.ccbi"];

[[CCDirector sharedDirector] replaceScene:scene];

}

别忘了打开CocosBuilder软件，Publish生成GameMenu.ccbi和TitleBar.ccbi文件。

1. 最后，打开Xcode工程，把新导出的GameMenu.ccbi和TitleBar.ccbi两个文件添加进来。

编译并运行。单击“进入游戏”按钮的时候，就会自动进入GameMenu场景了，如图14-43所示。



1. 自动进入GameMenu场景

这时候，单击顶部的返回按钮，也能够回到HelloWorldCocosBuilder场景。

### 制作About场景

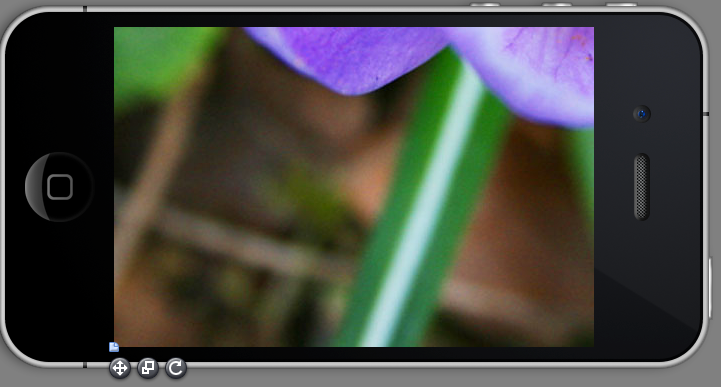
一般游戏的About场景都是关于游戏、公司的介绍或者游戏制作团队的感谢等信息。如果内容比较多，可以采用ScrollView方式，让玩家有选择性地查看About信息。因此，在制作About场景的时候选择使用CCScrollView控件来演示。

1. 新建文件。

选择New→New File，此时，不选择CCLayer作为Root Object type，而选择CCNode。同时为此文档取名为AboutScrollViewContent，因为现在制作的是CCScrollView将要显示的内容。

1. 设置CCNode。

设置CCNode的ContentSize为1024\*1024。为什么呢？因为接下来会添加一个精灵对象，精灵对象的ContentSize也是1024\*1024。如此巨大的精灵，一个屏幕显然是显示不了的，所以使用CCScrollView来显示。最终AboutScrollViewContent的效果图如图14-44所示。



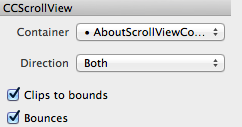
1. AboutScrollViewContent的效果图
2. 新建About文件。

选择New→New File，选择Root Object type为CCLayer，同时取名为About，保存。这时候，选择根节点CCLayer，把Custom class命名为About，因为接下来要在代码文件中也添加一个About类，可以在其中定义一些变量和方法。

按照惯例，会在所有根节点为容器的时候，调用一个CCLayerColor或者CCLayerGradient对象。这是因为，如果不添加一个这样的对象，背景会是黑色的。如果前面场景布局有空隙的时候，不至于影响场景的美观性。

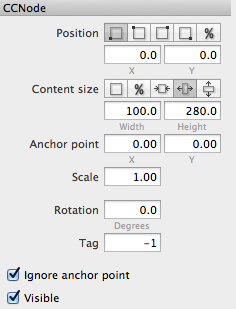
1. 添加一个CCScrollView子节点。

添加一个CCScrollView子节点，注意把它的Container属性设置为之前制作的AboutScrollViewContent.ccb，如图14-45所示。



1. Container属性设置

同时，添加一个CCBFile对象，此对象引用的是TitleBar.ccb文件。此时，要修改ScrollView的ContentSize，如图14-46所示。



1. 修改ScrollView的ContentSize

这里宽度是100％，即占满整个屏幕宽度。高度是280，上面留有40像素。这个预留的高度正好是之前定义的TitleBar高度。因此，About场景的最后效果如图14-47所示。



1. About场景的最后效果
2. 保存工程。

选择File→Publish。把生成的AboutScrollViewContent.ccbi和About.ccbi文件都添加到项目中，同时新建一个About类，此类无需任何实现。

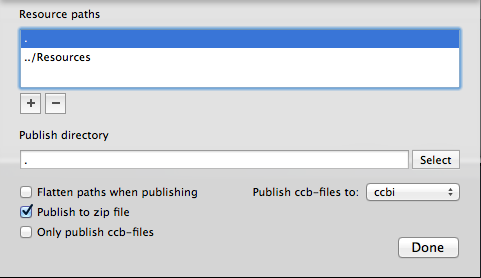
1. 编译并运行项目。

单击“关于”按钮，会进入到刚刚创建的场景之中。可以单击鼠标拖动scrollView中的内容。

通过对前面的学习，相信大家已经掌握使用CocosBuilder制作游戏场景了。如果每次添加新的场景，都要添加新的ccbi文件，似乎显得有点麻烦。同时，如果场景中引用一些新的图片，那么Xcode项目中也必须添加相应的图片。这样才能保证CocosBuilder制作的场景能够用代码正确地显示出来。接下来，介绍一种更方便的方法，把ccbi文件和图片、字体文件等全部打包成一个ZIP文件，实现游戏中的所有场景。

### 发布ZIP格式场景资源文件

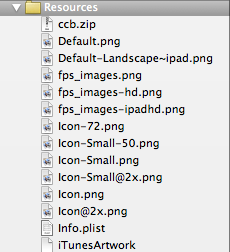
使用CocosBuilder发布ZIP格式的场景资源文件非常简单，选择File→Project settings。然后按照图14-48所示设置就行了。



1. CocosBuilder发布ZIP格式的场景资源文件

这里把“Only publish ccb-files”选项去掉，因为想让导出的ZIP文件中同时包含ccbi文件和ccbi所需的图片、字体等资源文件。设置好之后，选择Done，然后Publish。这时候，会在导出的目录下面发现一个ccb.zip文件（读者可以解压缩看看ZIP文件中的内容）。

打开Xcode，首先删除之前添加的一系列ccbi和资源文件，然后右键添加新的ZIP文件，此时的Resource分组如图14-49所示。



1. Resource分组

多整洁，只需要添加一个ZIP文件，而且如何CocosBuilder那端如果有添加新的资源文件或者新添ccbi文件，也只需要简单的Publish就ok了。接下来，看看如何在代码中使用此资源文件。其实非常简单，只需要以下2步：

1. 添加CCB\_ENABLE\_UNZIP预处理宏，记住Debug和Release模式下都要添加此宏。此宏可以在项目的Build Settings里面添加。
2. 在Cocos2D启动之前（即AppDelegate中的application: didFinishLaunching WithOptions函数最开始）添加下面两行代码：

[CCBFileUtils sharedFileUtils];

[CCBReader unzipResources:@"ccb.zip"];

此时，编译并运行，成功！

### 如何支持多个设备分辨率

现在，CocosBuilder 2.0支持一个ccb文件适配多个屏幕分辨率，iPhone/iPad以及各种Andriod分辨率。前面已经介绍过，新建一个ccb文件时，可以选择想要支持的分辨率。想要让此文件还支持其他分辨率，只需以下三个步骤：

1. 选择Root节点，把ContentSize设置为100%\*100%。这样会使场景覆盖整个屏幕区域，而不用理会实际屏幕的分辨率大小。
2. 当给根节点添加子结点的时候，有许多种方式可以设置子节点的坐标。如果想到支持多个分辨，最好使用百分比的方式。比如，如果想让一个节点放置在屏幕的中心点，那么可以设置相对于父节点的ContentSize的（50，50）处。
3. 对于那些可以设置ContentSize的节点，比如CCNode、CCScale9Sprite 节点，把它们的ContentSize设置为父节点的百分比。

具体如何实现一个ccb文件支持多种分辨率，可以查看Example，选择一个节点，选择View→Resolution就可以看到多个分辨率的选项了。

如果想要支持更多分辨率，选择下方的Edit Resolutions，然后编辑名字、屏幕大小，文件后缀和scale大小就可以了。

注意 设置多个分辨率以后，一定要尽可能相对定位的方式去布局结点坐标，可以相对于父容器的contenSize的百分比，也可以是相对于屏幕的anchorPoint。如果要支持多个分辨率，一定要为每个ccbi文件设置多个分辨率。最后，不要忘记测试，尽可能地充分测试才能保障最终程序布局的正确性。

## 着色器

Cocos2D 2.0与Cocos2D 1.x版本最大的区别就在于前者使用OpenGL ES 2.0，而后者使用OpenGL ES 1.1。而OpenGL ES 2.0与OpenGL ES1.1最大的区别就在于其渲染方式的不同。1.1使用固定渲染管线（fixed pipeline）,采用一些内置的函数实现诸如灯光、颜色和Camera的功能。而2.0则使用可编程渲染管线（programming pipeline），支持使用Shader直接操作GPU，从而实现其想要的光照、纹理映射等功能。那到底什么是Shader呢？简言之，就是一小段程序代码，语法酷似C语言，可以直接在GPU上运行来实现渲染效果。

### 着色器的种类及工作原理

OpenGL ES 2.0主要使用了以下二种着色器。

顶点着色器（Vertex Shader）：用于处理每个顶点，将顶点的空间位置投影在屏幕上，即计算顶点的二维坐标。同时，它也负责顶点的深度缓冲（Z-Buffer）的计算。顶点着色器可以掌控顶点的位置、颜色和纹理坐标等属性，但无法生成新的顶点。顶点着色器的输出传递到流水线的下一步。如果有之后定义了几何着色器，则几何着色器会处理顶点着色器的输出数据，否则，光栅化器继续流水线任务。

片断着色器（Fragment Shader）：用于处理来自光栅化器的数据。光栅化器已经将多边形填满并通过流水线传送至像素着色器，后者逐像素计算颜色。像素着色器常用来处理场景光照和与之相关的效果，如凸凹纹理映射和调色。名称片断着色器似乎更为准确，因为对于着色器的调用和屏幕上像素的显示并非一一对应。举个例子，对于1像素，片断着色器可能会被调用若干次来决定它最终的颜色，那些被遮挡的物体也会被计算，直到最后的深度缓冲才将各物体前后排序。

#### 着色器的功能

首先，它是现代图形学编程所推荐的方式，使用Shader可以直接操作GPU，所以，它会更灵活，功能也更加强大。同时，直接操作GPU也能极大地提高图形渲染性能，当然前提是Shader要写好。然后，有很多功能，在固定渲染管线方式下是达不到的。比如，当用户想要完成诸如对区域光的支持，逐个面片的计算光照而不是逐个顶点计算或者使用OpenGL ES 1.x渲染模型遇到某些限制时，就需要为这些程序编写Shader了。

使用Shader可以实现的简单功能如下：

* 具有真实感的材质：金属、石头、木头、图画等。
* 更加逼真的光照效果：区域光、软影等。
* 自然现象的渲染：火、烟、水、云等。
* 非真实感材质：画家手绘效果、素描画、展示技术的模拟。
* 纹理存储器的其他使用方法：纹理可以被用作存储纹理、光泽度、多项式系数等。
* 图象处理：卷积、平滑处理、复杂融合等。
* 动画效果：关键帧插值、粒子系统、过程化定义的动画。
* 用户编程实现的抗锯齿化算法。

有关Shader介绍和编程的更多内容，可以参考《OpenGL ES 2.0 Programming Guide》。

#### 着色器工作原理

首先，顶点着色器和片断着色器不能够单独使用，它们必须相互配合才能发挥功效。一个Shader程序应同时包含一段顶点着色器程序和一段片断着色器程序。它的工作方式通常如下所示：

1. 顶点着色器为屏幕上显示的每一个顶点定义属性。
2. 每一个顶点会分解成一系列的像素，这些像素可以通过片断着色器来继续处理。
3. 这样最终处理的像素值再显示到屏幕上面。

因为Cocos2D 2.0使用OpenGL ES 2.0，所以，即使最简单的CCNode节点，也包含一个shaderProgram实例变量，这个变量指向一段预先定义好的Shader程序，当节点需要绘制的时候就会调用此Shader程序。

##### 顶点着色器

Cocos2D定义了CCShaderCache类，它允许使用预先定义好的Shader程序。同时，它也可以缓存我们自已定义的一些Shader程序，这样就不用每次都重新加载了。

预先定义好的Shader程序的命名为ccShader\_xxx.h，它是一个字符串，包含了Shader程序的内容。读者可以找到ccShader\_PositionTexture\_vert.h文件，内容如下所示：

//1.

attribute vec4 a\_position;

attribute vec2 a\_texCoord;

//2.

uniform mat4 u\_MVPMatrix;

//3.

#ifdef GL\_ES

varying mediump vec2 v\_texCoord;

#else

varying vec2 v\_texCoord;

#endif

//4.

void main()

{

//5.

gl\_Position = u\_MVPMatrix \* a\_position;

//6.

v\_texCoord = a\_texCoord;

}

注意 为了可阅读性，这里去掉了字符串一些\n\和“”部分

在深入细节之前，我们站在更高的角度看看顶点着色器程序。每一个顶点着色器程序都包含输入和输出。

* **输入：**接收每一个顶点的坐标，如果是Sprite的话，那么就是Sprite的4个角的顶点坐标。它同时也接受每一个顶点上面所映射的纹理坐标，外加一个变换矩阵。这个变换矩阵可以使Sprite改变坐标位置、旋转和缩放。Cocos2D在运行Shader程序之前，会把这些输入变量传进来。
* **输出：**它会产出顶点在屏幕上显示的最终坐标，同时还有每个顶点上面映射的最终纹理坐标。然后片断着色器会使用这些输出变量。后面介绍片断着色器的时候会加以说明。

接下来，按照上面程序所标序号逐段解释：

1. 定义了输入顶点的数据结构。Attribute关键字告诉编译器，接下来定义的是一个输入变量。这里的vec2和vec4是一个浮点类型的向量，分别含2、4个分量。它们分别定义了顶点坐标格式和纹理坐标格式。
2. 当从源代码中传入一些外部变量到Shader程序中时，需要把它们声明为uniform。Mat4是一个4\*4的矩阵数据类型。
3. 顶点着色器处理完之后，发送一些数据给片断着色器做进一步的处理。当需要把一个变量从顶点着色器传到片断着色器的时候，应该把此变量声明为varying类型。注意，varying类型的变量是以插值替换的。打个比方，如果把顶点A的varying变量设置为0，顶点B的varying变量设置为1。那么，当在片断着色器中获取A与B中间的像素时，OpenGL会自动把值设置为0.5。每一个片断着色器都有它自己的插值计算方法，正如所看到的mediump，当然还有highp和lowp。更高的精度意味着更多的空间和计算，当然效果也会更加好。
4. 每一个顶点着色器都有一个main函数。
5. 顶点着色器需要给内置的gl\_Position变量赋值，这个值可以用矩阵加以变换。上面的程序中，把输入的顶点坐标乘以ModelViewProjection矩阵，这个矩阵是由Cocos2D自动传入进来的，它定义了精灵的位置、缩放和旋转。
6. 这里把输入的纹理坐标直接传递给片断着色器进行处理。

##### 片断着色器

接下来，看看与之对应的片断着色器的代码，打开ccShader\_PositionTexture\_frag.h文件，代码如下所示：

//1

#ifdef GL\_ES **precision** **mediump** **float**; #endif

//2 **varying** **vec2** v\_texCoord;

//3 **uniform** **sampler2D** u\_texture;

**void** main() {

//4 gl\_FragColor = **texture2D**(u\_texture, v\_texCoord);

}

记住，当程序运行至片断着色器的时候，OpenGL会为精灵的每一个像素点调用一次片断着色器程序。其实，片断着色器的目标就是去计算每一个像素点的颜色值。默认的Shader程序非常简单，直接把纹理图片上的像素点映射过去。

下面是片断着色器的分段解释：

1. 设置片断着色器的的浮点型数据精度，这里设置为medium（中等）精度。
2. 在顶点着色器中传过来的任何变量，在片断着色器的开头都要进行定义，且类型是varying。
3. 片断着色器也可以使用uniform类型的变量，它是一个常量值，也是从程序代码里传过来的。Cocos2D使用一个uniform类型的变量sampler2D把纹理传递给片断着色器。
4. gl\_FragColor是一个内置变量，被赋值最终屏幕上需要显示的像素的颜色值。这里把顶点着色器传过来的纹理坐标与程序传进来的像素值进行运算。使用的是texture2D函数，这的作用就是计算相应顶点使用纹理对应位置的颜色值。

目前为止，大家已经了解Shader程序的基本结构和写法了，接下来，理解这一切是如何联系起来的。

##### 如何调用Shader程序

这些Shader程序都在CCGrid.m中调用，所以，打开CCGrid.m找到init函数，找到以下这句代码：

self.shaderProgram = [[CCShaderCache sharedShaderCache] programForKey:kCCShader\_PositionTexture];

这里加载了kCCShader\_PositionTexture类型的Shader程序，这个宏定义在CCGLProgram.h中。如果想继续探究CCShaderCache的实现，可直接查看CCShaderCache类的实现。这个留给读者自己去完成。

初始化完之后，调用bilt函数载入Shader程序，代码如代码清单14-2所示。

1. 调用bilt函数调用Shader程序

-(void)blit

{

NSInteger n = gridSize\_.x \* gridSize\_.y;

ccGLEnableVertexAttribs( kCCVertexAttribFlag\_Position | kCCVertexAttribFlag\_TexCoords );

[shaderProgram\_ use];

[shaderProgram\_ setUniformForModelViewProjectionMatrix];

//

// Attributes

//

// position

glVertexAttribPointer(kCCVertexAttrib\_Position, 3, GL\_FLOAT, GL\_FALSE, 0, vertices)

// texCoods

glVertexAttribPointer(kCCVertexAttrib\_TexCoords, 2, GL\_FLOAT, GL\_FALSE, 0, texCoordinates);

glDrawElements(GL\_TRIANGLES, (GLsizei) n\*6, GL\_UNSIGNED\_SHORT, indices);

CC\_INCREMENT\_GL\_DRAWS(1);

}

首先，激活需要传递给顶点着色器的两个变量属性kCCVertexAttribFlag\_Position 和kCCVertexAttribFlag\_TexCoords。然后调用Shader程序。调用完之后，设置模型视图变换矩阵。接下来，定义顶点属性数组和纹理坐标属性数组，最后是渲染。

关于Cocos2D如何使用Shader更详尽的信息，可以参考相关书籍和Cocos2D源代码。接下来，带领大家编写自己的Shader程序。

### 编写自己的Shader程序

首先，打开Xcode，新建一个Cocos2D V 2.x工程，命名为Cocos2DShaderDemo。因编译并运行之后，如图14-50所示。



1. 新建一个Cocos2D V2.x工程

自带的模板提供了GameCenter的Achievements和Leaderboard支持，这里显然不需要。因此要对新建的模板进行一些修改。

首先，找到HelloWorldLayer.h，去掉@interface的layer实现的两个接口：

* GKAchievementViewControllerDelegate
* GKLeaderboardViewControllerDelegate

打开HelloWorldLayer.m文件，删除achievementViewControllerDidFinish和leaderboardViewControllerDidFinish这两个方法。用以下代码替代init方法，如代码清单14-3所示。

1. 替代init方法

-(id) init

{

if( (self=[super init]) ) {

CGSize size = [[CCDirector sharedDirector] winSize];

[CCMenuItemFont setFontSize:28];

CCMenuItem \*itemVertexShader = [CCMenuItemFont itemWithString:@"VertexShaderTest" block:^(id sender)

{

}];

CCMenu \*menu = [CCMenu menuWithItems:itemVertexShader, nil];

[menu alignItemsVerticallyWithPadding:20];

[menu setPosition:ccp( size.width/2, size.height/2)];

[self addChild:menu];

}

return self;

}

这时候编译并运行，效果如图14-51所示。



1. 准备工作的输出

新建另外一个VertexShaderTest层，在此层中测试Vertex Shader，步骤如下。

1. 新建一个BackLayer，继承自CCLayer，打开BackLayer.m，把init方法用代码清单14-4替换。
2. init方法替换

-(id) init{

if(self = [super init]){

// Default font size will be 28 points.

[CCMenuItemFont setFontSize:28];

CCMenuItem \*backItem = [CCMenuItemFont itemWithString:@"MainMenu" block:^(id sender)

{

CCScene \*scene = [CCScene node];

HelloWorldLayer \*layer = [HelloWorldLayer node];

[scene addChild:layer];

[[CCDirector sharedDirector] replaceScene:scene];

}];

backItem.position = ccp(80,280);

CCMenu \*menu = [CCMenu menuWithItems:backItem, nil];

menu.position = CGPointZero;

// Add the menu to the layer

[self addChild:menu];

}

return self;

}

1. 打开VertexShaderTest.m，在文件顶部包含头文件：

#import "BackLayer.h"

1. 再往init方法中添加下列代码：

BackLayer \*layer = [BackLayer node];

[self addChild:layer];

1. 打开HelloWorldLayer.m文件，把itemVertexShader的初化用以下代码替换：

CCMenuItem \*itemVertexShader = [CCMenuItemFont itemWithString:@"VertexShaderTest" block:^(id sender)

{

CCScene \*scene = [CCScene node];

VertexShaderTest \*layer = [VertexShaderTest node];

[scene addChild:layer];

[[CCDirector sharedDirector] replaceScene:scene];

}];

1. 编译并运行。

单击VertexShaderTest就可以跳转至VertexShaderTest场景中了，此时，VertexShaderTest场景中只有一个MainMenu菜单项，单击该菜单项时，可以回到HelloWorldLayer场景。

至此，所有的准备工作都做好了，真正的挑战来了！

### 如何为Sprite定制顶点着色器

本节主要介绍如何为Sprite定制顶点着色器，并且使用顶点着色器对Sprite进行一些简单的几何变换，比如缩放、旋转和平移等操作。

为Sprite定制顶点着色器的步骤如下。

1. 打开VertexShaderTest.m文件，声明成员变量，如下所示：

@implementation VertexShaderTest{

CCSprite \*sprite;

}

提示 为什么放在.m文件声明呢?因为这更符合私有变量的定义，这也是Apple推荐的做法。

1. 把在init方法的BackLayer初始化之前加入代码清单14-5所示代码。
2. 在init方法的BackLayer的初始化之前加入代码

CGSize winSize = [[CCDirector sharedDirector] winSize];

// 1

sprite = [CCSprite spriteWithFile:@"Icon.png"];

sprite.position = ccp(winSize.width/2,winSize.height/2);

[self addChild:sprite];

// 2

NSString \*vertexShaderFile = [[CCFileUtils sharedFileUtils] fullPathFromRelativePath:@"CustomVertexShader.vsh"];

const GLchar \* vertexSource = (GLchar\*) [[NSString stringWithContentsOfFile:vertexShaderFile encoding:NSUTF8StringEncoding error:nil] UTF8String];

sprite.shaderProgram = [[CCGLProgram alloc] initWithVertexShaderByteArray:vertexSource fragmentShaderByteArray:ccPositionTexture\_frag];

//3

[sprite.shaderProgram addAttribute:kCCAttributeNamePosition index:kCCVertexAttrib\_Position];

[sprite.shaderProgram addAttribute:kCCAttributeNameTexCoord index:kCCVertexAttrib\_TexCoords];

//4

[sprite.shaderProgram link];

[sprite.shaderProgram updateUniforms];

//5.set uniform

int scaleLocation = glGetUniformLocation(sprite.shaderProgram->program\_, "u\_scale");

glUniform1f(scaleLocation, 2.0f);

int moveLocation = glGetUniformLocation(sprite.shaderProgram->program\_, "u\_move");

glUniform2f(moveLocation, 100, 0);

int rotateLocation = glGetUniformLocation(sprite.shaderProgram->program\_, "u\_rotate");

glUniform1f(rotateLocation, 30);

//6.

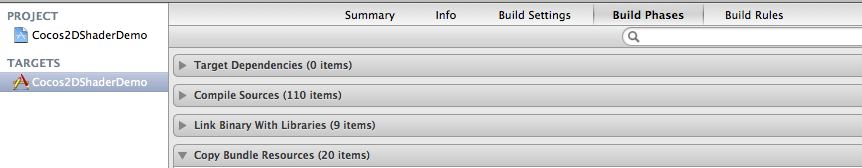
[sprite.shaderProgram use];

1. 在详细分解这段代码之前，再新建一个CustomVertexShader.vsh文件。

选择File→New File，然后选择c and c++，再选择Header File，命名为CustomVertexShader.vsh，这时候会提示到底使用什么后缀，当然选择.vsh。

1. 要让这段Vertex Shader跑起来，还需要把刚刚添加的Shader程序添加到Resource bundle中。

单击工程，选择Target，选择Build phase，再往Copy Bundle Resource中添加CustomVertexShader.vsh。Copy Bundle Resource的位置如图14-52所示。



1. Copy Bundle Resource的位置

编译并运行，效果如图14-53所示。



1. 定制顶点着色器效果

虽然效果不是很惊奇，但这却是一个了不起的开始，因为可以直接操作GPU啦。

现在，看看之前的代码到底做了什么事，按照代码中的序号，一段段来看：

1. 这段代码非常简单，使用Icon.png图片初始化一个精灵，同时把它添加到屏幕的中间位置。
2. 首先使用CCFileUtils获取CustomVertexShader的实际完整路径，并转化为UTF-8类型的字符串。接下来给Sprite的shaderProgram属性赋值。
3. 这个shaderProgram也就是Sprite渲染时所关联的Shader程序，它同时包含一段Vertex Shader和一段Fragment Shader。这里使用自定义的Vertex Shader，而Fragment Shader使用系统内置的ccPositionTexture\_frag。这个默认的Fragment Shader前面章节已经介绍过了。
4. 设置shaderProgram的输入变量，即之前提到的attribute vec4 a\_position和attribute vec2 a\_texCoord 这两个变量。它们分别对应着纹理图片的顶点数据和纹理坐标数据。
5. 链接Shader程序，把片断着色器和顶点着色器加载到Shader程序里。同时更新uniform变量。之后，我们就可以自定义自己的uniform类型的数据传递给Shader程序了。
6. 给Shader程序传递3个uniform变量，分别定义Sprite的缩放、平移和旋转。这里使用glGetUniformLocation获得Shader程序中变量的位置，然后使用glUniform{n}f赋值变量。注意，这里的n可以是1、2、3、4，分别用来给Shader程序赋值float类型、vec2、vec3和vec4类型的参数。注意，给uniform变量赋值的操作必须放到updateUniform之后进行。
7. 使用精灵的spriteShader程序。这样当精灵的draw方法被调用时，就会调用CustomVertexShader这段Shader程序了。
8. CustomVertexShader.vsh的具体实现如代码清单14-6所示。
9. CustomVertexShader.vsh的具体实现

attribute vec4 a\_position;

attribute vec2 a\_texCoord;

uniform mat4 u\_MVPMatrix;

uniform float u\_scale;

uniform vec2 u\_move;

uniform float u\_rotate; //角度

#ifdef GL\_ES

varying mediump vec2 v\_texCoord;

#else

varying vec2 v\_texCoord;

#endif

const float PI = 3.141592653;

void main()

{

//不能直接操作a\_position，而需要定义一个中转变量vec4 a来操作顶点坐标

vec4 a = a\_position;

//2d中点的缩放

a.x = a.x \* u\_scale;

a.y = a.y \* u\_scale;

//2d中点的平移

a.x = a.x + + u\_move.x;

a.y = a.y + u\_move.y;

//2d中的点中(x,y)按逆时针旋转θ弧度,可使用以下公式:

//x′ = x \* cos θ - y \* sin θ; y′ = y \* cos θ + x \* sin θ

a.x = a.x \* cos(u\_rotate \* PI / 180.0 ) - a.y \* sin(u\_rotate \* PI / 180.0);

a.y = a.y \* cos(u\_rotate \* PI/ 180.0) + a.x \* sin(u\_rotate \* PI / 180.0);

gl\_Position = u\_MVPMatrix \* a;

v\_texCoord = a\_texCoord;

}

这些变量类型的定义，之前的章节已经讲得很清楚了。这里新添3个uniform类型的变量，这几个变量都在程序中有赋值了。接下来主要看看如何使用这3个变量来操作精灵的顶点数据，使之进行缩放、平移和旋转操作。

1. 缩放：

vec4 a = a\_position;

//2d中点的缩放

a.x = a.x \* u\_scale;

a.y = a.y \* u\_scale;

这段代码很简单，就是把顶点的xy坐标都乘以u\_scale就可以了。线性代数也讲过向量的缩放，就是拿向量乘以一个标量。（这里注意不能直接操作a\_position变量，而需要用一个中间变量来转换。）

1. 平移：

//2d中点的平移

a.x = a.x + + u\_move.x;

a.y = a.y + u\_move.y;

1. 旋转：

//2d中的点中(x,y)按逆时针旋转θ弧度,可使用以下公式:

//x′ = x \* cos θ - y \* sin θ; y′ = y \* cos θ + x \* sin θ

a.x = a.x \* cos(u\_rotate \* PI / 180.0 ) - a.y \* sin(u\_rotate \* PI / 180.0);

a.y = a.y \* cos(u\_rotate \* PI/ 180.0) + a.x \* sin(u\_rotate \* PI / 180.0);

可以把这些操作跟线性代数的向量几何变换联系在一起。最后，记得给gl\_Position赋值，这是Shader的内置变量，即最终的顶点数据。

前面分别给u\_scale、u\_move和u\_rotate赋值2.0、（100，0）和30。它们组合后的最终几何意义就是放大2倍，右移100像素，同时逆时针旋转30度。

Vertex Shader是不是没有想像中那么难呢？不过在编程的时候要注意两点：

* 所有的操作都是float类型，因此数字常量都需要加小数位，同时不能带有在Shader中不能带有f字样。
* 不可以直接操作a\_position变量，需要用中间变量进行中转。

好了，读者可以充分发挥自己的数学天才，玩玩Vertex Shader，体会一下吧。接下来，介绍Fragment Shader。

### 如何定制片段着色器

首先，新建一个CCLayer，命名为FragmentShaderTest。具体步骤如下。

1. 打开HelloWorldLayer.m，添加一个新的菜单按钮，如下所示：

CCMenuItem \*itemFragmentShader = [CCMenuItemFont itemWithString:@"FragmentShaderTest" block:^(id sender)

{

CCScene \*scene = [CCScene node];

FragmentShaderTest \*layer = [FragmentShaderTest node];

[scene addChild:layer]; [[CCDirector sharedDirector] replaceScene:scene];

}];

1. 修改Menu的初始化语句：

CMenu \*menu = [CCMenu menuWithItems:itemVertexShader,itemFragmentShader, nil];

1. 打开FragmentShaderTest.m文件，用代码清单14-7所示代码替换。
2. FragmentShaderTest.m文件的代码替换

@implementation FragmentShaderTest{

CCSprite \*sprite;

}

-(id) init{

if (self = [super init]) {

CGSize winSize = [[CCDirector sharedDirector] winSize];

sprite = [CCSprite spriteWithFile:@"Icon.png"];

sprite.position = ccp(winSize.width/2,winSize.height/2);

[self addChild:sprite];

NSString \*fragmentShaderFile = [[CCFileUtils sharedFileUtils] fullPathFromRelativePath:@"CustomFragmentShader.fsh"];

const GLchar \* fragmentSource = (GLchar\*) [[NSString stringWithContentsOfFile:fragmentShaderFile encoding:NSUTF8StringEncoding error:nil] UTF8String];

sprite.shaderProgram = [[CCGLProgram alloc] initWithVertexShaderByteArray:ccPositionTexture\_vert fragmentShaderByteArray:fragmentSource];

[sprite.shaderProgram addAttribute:kCCAttributeNamePosition index:kCCVertexAttrib\_Position];

[sprite.shaderProgram addAttribute:kCCAttributeNameTexCoord index:kCCVertexAttrib\_TexCoords];

[sprite.shaderProgram link];

[sprite.shaderProgram updateUniforms];

//设置uniform一定要放在updateUniforms之后,放在use之前或者之后都可以

int colorLocation = glGetUniformLocation(sprite.shaderProgram->program\_, "u\_left\_color");

glUniform4f(colorLocation, 1.0f, 0.0f,0.0f,1.0f);

colorLocation = glGetUniformLocation(sprite.shaderProgram->program\_, "u\_right\_color");

glUniform4f(colorLocation, 0.0f, 1.0f,0.0f,1.0f);

[sprite.shaderProgram use];

BackLayer \*layer = [BackLayer node];

[self addChild:layer];

}

return self;

}

这大段代码的作用和之前VertexShaderTest相似。唯一不同的是，这次使用自定义的Fragment Shader，而使用的默认顶点着色器ccPositionTexture\_vert。同样，在程序中定义两个外部变量u\_left\_color和u\_right\_color。这两个变量都是一个颜色值，在Shader中引用的类型是vec4，所以使用glUniform4f赋值。因为本小节想实现的功能，是把精灵左边设置成u\_left\_color，而精灵右边设置成u\_right\_color。在没有使用Shader的时候，实现起来比较麻烦，现在有了Fragment Shader，这一切会变得非常简单。

1. 新建CustomFragmentShader。

选择File→New File，选择C and C++→Header file。把文件名改成CustomFragmentShader.fsh，保存。把内容清空，再加入代码清单14-8所示代码。

1. 新建CustomFragmentShader的代码替换

#ifdef GL\_ES

precision mediump float;

#endif

varying vec2 v\_texCoord;

uniform sampler2D u\_texture;

uniform vec4 u\_left\_color;

uniform vec4 u\_right\_color;

void main()

{

//注意运算全部要是浮点型

vec2 v\_tex = v\_texCoord;

vec4 color = texture2D(u\_texture,v\_tex);

vec4 new\_color = color;

//左边是u\_left\_color,右边是u\_right\_color

if(v\_tex.x <= 0.5){

new\_color \*= u\_left\_color;

}

if(v\_tex.x >= 0.5){

new\_color \*= u\_right\_color;

}

//gl\_FragColor = vec4(1.0,0,0,1.0);

gl\_FragColor = new\_color;

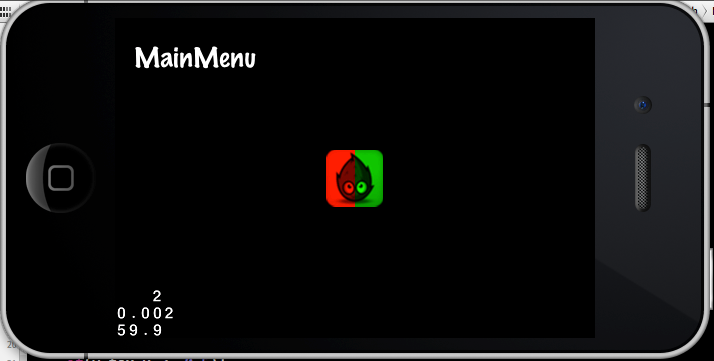
代码非常容易，但是关于v\_texCoord变量要作一些说明。

首先，此变量是通过顶点着色器传过来的，所以需要声明为varying类型。

其次，它是一个二维向量，xy的取值范围都是0-1。如果x为0，表示图片的最左边，1表示最右边。而y值如果为0，表示最上边，为1则表示最下边。（0.5，0.5）表示图片的中点。因此，上面的Shader代码通过判断v\_texCoord的x值和0.5的关系，就可以实现左右两边用不同的颜色渲染。

1. 在编译运行看效果之前，别忘了添加Bundle Resource，把刚刚的Shader文件添加进去。

添加好之后，编译运行效果如图14-54所示。



1. 定制片段着色器效果

### 旋风效果示例

本节主要介绍旋风效果（SwirlEffect）的实现，具体步骤如下。

1. 新建一个CCLayer，命名为SwirlEffectShader，在.m文件中声明下列成员变量：

@implementation SwirlEffectShader{

CCSprite \*sprite;

float \_angle;

int \_angleLocation;

}

1. 把init方法替换成代码清单14-9所示代码。
2. init方法替换

-(id) init{

if (self = [super init]) {

CGSize winSize = [[CCDirector sharedDirector] winSize];

// 1

sprite = [CCSprite spriteWithFile:@"Default.png"];

sprite.rotation = 90.0;

sprite.position = ccp(winSize.width/2,winSize.height/2);

[self addChild:sprite];

// 2

NSString \*fragmentShaderFile = [[CCFileUtils sharedFileUtils] fullPathFromRelativePath:@"SwirlEffectShader.fsh"];

const GLchar \* fragmentSource = (GLchar\*) [[NSString stringWithContentsOfFile:fragmentShaderFile encoding:NSUTF8StringEncoding

error:nil] UTF8String];

sprite.shaderProgram = [[CCGLProgram alloc] initWithVertexShaderByteArray:ccPositionTexture\_vert

fragmentShaderByteArray:fragmentSource];

[sprite.shaderProgram addAttribute:kCCAttributeNamePosition index:kCCVertexAttrib\_Position];

[sprite.shaderProgram addAttribute:kCCAttributeNameTexCoord index:kCCVertexAttrib\_TexCoords];

[sprite.shaderProgram link];

[sprite.shaderProgram updateUniforms];

//设置uniform一定要放在updateUniforms之后,放在use之前或者之后都可以

int colorLocation = glGetUniformLocation(sprite.shaderProgram->program\_, "rt\_w");

glUniform1f(colorLocation, 480.0);

colorLocation = glGetUniformLocation(sprite.shaderProgram->program\_, "rt\_h");

glUniform1f(colorLocation, 320.0);

\_angleLocation = glGetUniformLocation(sprite.shaderProgram->program\_, "angle");

\_angle = 0.01f;

[self scheduleUpdate];

BackLayer \*layer = [BackLayer node];

[self addChild:layer];

}

return self;

}

上面这段代码跟前面的Fragment Shader并无太大差异，只是定义uniform类型的变量有所差别。不过，这里并没有调用shaderProgram的use方法。而是放到update方法中。update方法的代码如下：

-(void) update:(ccTime)dt{

\_angle += 0.01f;

if (\_angle >= 2.0) {

\_angle = 0.01f;

}

[sprite.shaderProgram use];

glUniform1f(\_angleLocation, \_angle);

}

这里主要是把uniform变量\_angle不断地更新它的值，然后传到Shader程序中。注意，glUniform1f的调用必须在use方法之后。

1. 新建一个SwirlEffectsShader.fsh并把它加到Resource Bundle中。

SwirlEffectShader的代码实现如代码清单14-10所示。

1. SwirlEffectShader的代码实现

#ifdef GL\_ES

precision mediump float;

#endif

varying vec2 v\_texCoord;

// Scene buffer

uniform sampler2D u\_texture;

// width of the current render target

uniform float rt\_w;

// height of the current render target

uniform float rt\_h;

uniform float angle;

// Swirl effect parameters

const float radius = 160.0;

const vec2 center = vec2(240.0, 160.0);

vec4 PostFX(sampler2D tex, vec2 uv, float time)

{

vec2 texSize = vec2(rt\_w, rt\_h);

vec2 tc = uv \* texSize;

tc -= center;

float dist = length(tc);

if (dist < radius)

{

float percent = (radius - dist) / radius;

float theta = percent \* percent \* angle \* 8.0;

float s = sin(theta);

float c = cos(theta);

tc = vec2(dot(tc, vec2(c, -s)), dot(tc, vec2(s, c)));

}

tc += center;

vec3 color = texture2D(u\_texture, tc / texSize).rgb;

return vec4(color, 1.0);

}

void main (void)

{

vec2 uv = v\_texCoord;

gl\_FragColor = PostFX(u\_texture, uv, time);

}

关于这段Shader具体的算法可以查阅相关资料，这里不具体阐述。

编译并运行，将看到一张图片慢慢地旋转变成图14-55所示。



1. 旋风效果

这个效果是不是很酷呢？因为有了Shader，如果数学和图形学学得足够好，几乎可以实现任何想要达到的图形效果，完全取决于自己的想象力了。

提示 本书随书源代码还提供了局部像素放大和图片遮罩的Shader效果，读者可自行研究。

## 本章小结

本章主要讲解了Cocos2D官方场景编辑器CocosBuilder的使用，通过本章的学习，读者应该要掌握使用CocosBuilder进行场景编辑的基本方法，熟悉元素布局，以及如何支持多个设备分辨率和打包所有资源文件为ZIP。

此外，本章介绍了以及Cocos2D 2.0的新特性Shader。详细介绍Shader的工作原理和各种组成部分，以及如何编写自己的Vertex Shader和Fragment Shader，希望帮助读者踏入Shader编程的大门。最后，本书向大家展示了一个非常有意思的Shader效果，期待可以在游戏中看到大家运用此特效。