第三部分 高级篇（第13~19章）

# 第13章 粒子系统

从本章开始进入进阶部分，带领大家更深入地了解Cocos2D的强大特性，比如的粒子系统。

本章将涉及粒子系统的基本知识、Cocos2D中的粒子系统类，并用教大家如何手动创建粒子系统；此外，介绍一款优秀的工具Particle Designer用于创建粒子系统。

掌握了粒子系统后，会让游戏效果锦上添花，更加吸引用户的眼球。

## Cocos2D中的粒子系统

粒子系统是计算机图形学中用来模拟一些特定效果的技术，特别是一些模糊的现象。这些现象用传统的渲染技术难以实现。使用粒子效果可以模拟的现象有火、爆炸、烟、水流、火花、落叶、云、雾、雪、尘、流星尾迹或发光轨迹这样的模糊而又抽象的视觉效果。

### 粒子系统相关的类

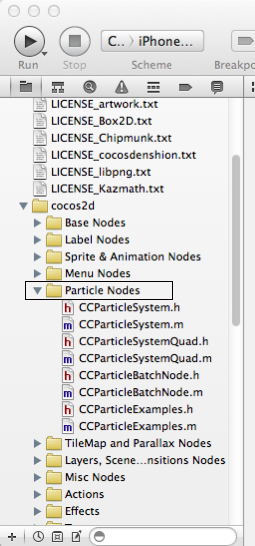
在Cocos2D中提供了几个与创建和编辑粒子系统相关的类。了解并掌握了这些类之后，就可以在游戏中轻松的创建和编译粒子系统。

本节重点介绍CCParticleSystem、CCParticleSystemQuad和CCParticleBatchNode类。

#### CCParticleSystem类

在Cocos2D中，是通过CCParticleSystem及其子类来创建粒子系统的。

在cocos2d-iphone的下载文件夹中找到cocos2d-ios.xcodeproj这个项目，在Xcode中打开后，在左侧点开cocos2d文件夹的下三角，然后点开Particle Nodes文件夹，会看到Cocos2D中所提供的和粒子系统相关的类，如图13-1所示。



1. Cocos2D中和粒子系统相关的类

打开CCParticleSystem.h，会发现CCParticleSystem这个类直接继承自CCNode，并遵循CCTextureProtocol协议。

也就是说，CCNode（节点类）这个父类的所有属性和方法同样适用于CCParticleSystem类。当然，它也有自己独享的属性和方法。关于粒子系统的属性将在13.3节做详细介绍，这里先简单介绍CCParticleSystem类的方法，如表13-1所示。

1. CCParticleSystem类方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| -(BOOL)addParticle | 将一个粒子添加到发射器中 |
| -(void)initParticle:(tCCParticle \*)particle | 初始化一个粒子 |
| -(id)initWithDictionary:(NSDictionary \*)dictionary | 从NSDictionary初始化一个粒子系统 |
| -(id)initWithFile:(NSString \*)plistFile | 从plist文件初始化一个粒子系统 |
| -(id)initWithTotalParticles:(NSUInteger) numberOfParticles | 使用固定数量的粒子来初始化一个粒子系统 |
| -(BOOL)isFull | 判断粒子系统是否已满 |
| +(id)particleWithFile:(NSString \*)plistFile | 类方法，使用plist文件来创建并初始化一个粒子系统。该plist文件可以手动创建，或是使用Particle Desinger来创建（13.3节将详细说明如何使用该工具） |
| -(void)resetSystem | 重置粒子系统，将当前所有的活跃粒子清除 |
| -(void)stopSystem | 停止发射粒子，已发射的粒子将继续运行，直到消失 |
| -(void)update:(ccTime)dt | 实时更新，应在游戏的每一帧调用 |

#### CCParticleSystemQuad类

CCParticleSystemQuad类继承自CCParticleSystem，属于具体的粒子类型。CCParticleSystemQuad具备以下特性：

* 创建的粒子系统可以缩放。
* 创建的粒子大小可以是任何浮点数。
* 支持子矩阵。

注意 如果在1代和2代iPhone上使用，CCParticleSystemPoint的速度要略高于CCParticleSystemQuad；而如果是在3代及以后的iPhone和iPad上使用，CCParticleSystemQuad的速度更快。CCParticleSystemQuad比CCParticleSystemPoint占用更多内存和GPU显存。

除了从CCParticleSystem类继承的方法外，还支持以下几个方法，如表13-2所示。

1. CCParticleSystem类方法

|  |  |
| --- | --- |
| 方法 | 描述 |
| -(void)initIndices | 初始化顶点的索引 |
| -(void)initTexCoordsWithRect:(CGRect)rect | 使用矩形来初始化一个纹理 |
| -(void)setDisplayFrame:(CCSpriteFrame\*)spriteFrame | 将一个新的CCSpriteFrame设置为粒子。注意，该方法属于实验性质，建议采用setTexture:withRect方法 |
| –(void)setTexture:(CCTexture2D \*)texture withRect:(CGRect) rect | 使用一个矩形来设置新的纹理，该矩形以点数值为单位 |

提示 从v2.0 Alpha版本起，CCParticleSystemPoint已经不再被支持。因此，如果在v2.0 Alpha以后的版本中使用CCParticleSystemPoint，系统会提示出错。

#### CCParticleBatchNode类

CCParticleBatchNode类是Cocos2D v1.1版本后新增的一个类，它的作用类似于精灵表单。CCParticleBatchNode和CCParticleSystem的关系就等同于CCSpriteBatchNode和CCSprite的关系。也就是说，一个CCParticleBatchNode中可以包含多个粒子系统，并且可以仅使用1次OpenGL命令调用就将所有的粒子系统绘制出来。

一个CCParticleBatchNode中只可引用一个纹理（纹理图片或者纹理图集）。只有包含在该纹理中的CCParticleSystem粒子系统才可以添加到CCParticleBatchNode中。

使用CCParticleBatchNode的限制是：

* 当前版本只支持CCParticleSystemQuad；
* CCParticleBatchNode中的所有粒子效果必须使用相同的参数、OpenGL混合模式和纹理等。

如果在项目中需要使用多个粒子系统，那么不妨使用CCParticleBatchNode类。

### Cocos2D内置的粒子系统

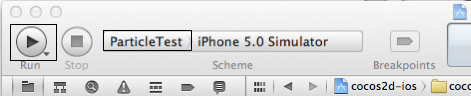
Cocos2D附带了一个粒子效果的示例，打开CCParticleExamples.h，会看到里面提供了11种粒子系统，可以在项目中直接使用这些默认的粒子系统类，或是对其进行修改。

这11种粒子系统如表13-3所示。

1. 粒子系统

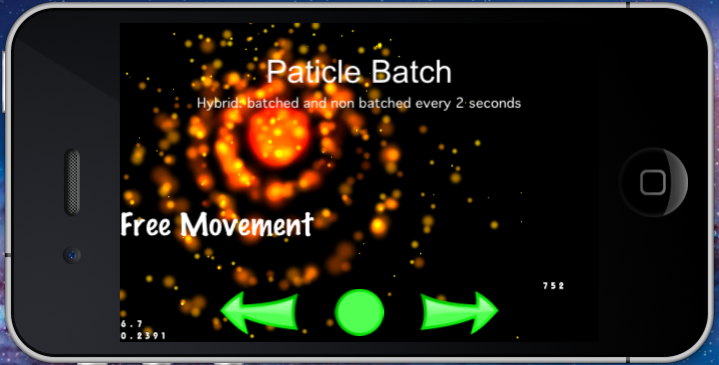
|  |  |
| --- | --- |
| 系统名称 | 描述 |
| CCParticleExplosion | 爆炸粒子系统 |
| CCParticleFire | 火焰粒子系统 |
| CCParticleFireworks | 烟花粒子系统 |
| CCParticleFlower | 花朵粒子系统 |
| CCParticleGalaxy | 银河系粒子系统 |
| CCParticleMeteor | 流星粒子系统 |
| CCParticleRain | 下雨粒子系统 |
| CCParticleSmoke | 烟雾粒子系统 |
| CCParticleSnow | 下雪粒子系统 |
| CCParticleSpiral | 螺旋粒子系统 |
| CCParticleSun | 太阳粒子系统 |

如果要查看每种粒子系统的实际视觉效果，可以选择ParticleTest,然后选择设备或模拟器进行测试，如图13-2所示。



1. 选择ParticleTest进行测试

编译运行项目，可以在模拟器或设备中看到Cocos2D内置的粒子系统效果，如图13-3所示。



1. ParticleTest在模拟器中的测试效果

在这个示例中，除了以上提到的11种粒子系统，还有一些其他的粒子效果，读者可以自己尝试。

## 手动创建粒子系统示例

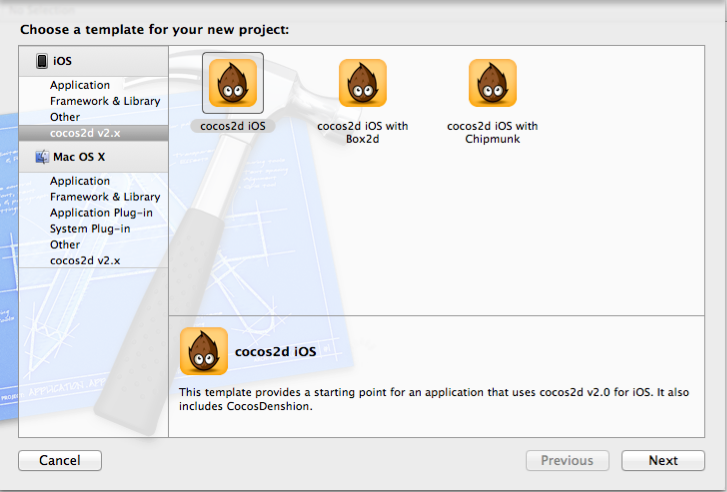
除了使用Cocos2D自带的粒子系统，有时开发者还会想要使用定制的粒子系统，以产生更加丰富多彩的炫丽效果。

本节将使用一个示例项目ParticleEffect来说明如何在项目中手动创建粒子系统。

1. 创建ParticleEffect项目

为了让大家更好的了解如何在项目中创建并使用粒子系统，需要创建一个示例项目ParticleEffect，并一一示范如何修改粒子系统的各种属性。

打开Xcode，单击File→New→New Project，在左侧选择iOS→cocos2d，选择标准的Cocos2D模板，如图13-4所示。



1. 选择Cocos2D标准模板创建一个新项目

单击Next，在Product name处输入“ParticleEffect”，单击Next，选择保存的位置后，单击确认。

1. 添加Cocos2D内置的粒子系统

在学习手动创建定制的粒子系统前，先尝试在项目中添加并使用Cocos2D中内置的粒子系统。

切换到HelloWorldLayer.m文件，清除init方法中的原有内容，并将代码替换成以下的内容，如代码清单13-1所示。

1. init方法的内容替换

// on "init" you need to initialize your instance

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

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

//创建一个基本的下雪粒子系统

CCParticleSnow \*rain = [CCParticleSnow node];

//将该粒子系统添加到当前层的子节点

[self addChild:snow z:1];

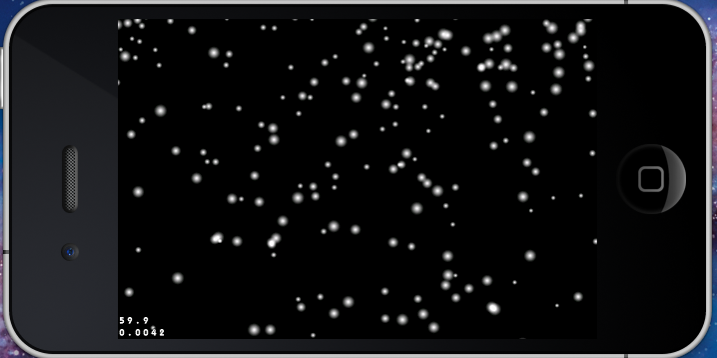
}

return self;

}

在以上的代码中， 首先使用Cocos2D默认提供的粒子系统CCParticleRain来创建并初始化一个下雨效果的粒子系统，并将其添加为当前层的子节点。

编译运行项目，可以在设备或模拟器上看到下雪的粒子效果，如图13-5所示。



1. 模拟器中显示的下雪粒子效果

当然，也可以轻松地用类似方法在项目中使用其他10种默认的粒子系统，整个操作非常简单。

1. 手动创建粒子系统类

在熟悉了添加Cocos2D内置的粒子系统后，接下来将要学习如何手动创建一个粒子系统类。

在Xcode中右键单击ParticleEffect的项目名称，选择New File，选择iOS→Cocoa Touch，然后选择右侧的Objective-C class，在Class的后面输入MyParticle，并选择Subclass of NSObject，单击Next，并单击Create以确认创建。

1. 切换到MyParticle.h，并使用以下代码替代其中的内容，如代码清单13-2所示。
2. 创建MyParticle类

#import "cocos2d.h"

#import "CCParticleSystemQuad.h"

@interface MyParticle : CCParticleSystemQuad{

}

@end

在以上代码中创建了一个继承自CCParticleSystemQuad类的MyParticle类。

1. 切换到MyParticle.m，使用以下代码替代其中的内容，创建一个定制的粒子系统。如代码清单13-3所示。
2. MyParticle.m的内容替换

#import "MyParticle.h"

@implementation MyParticle

-(id)init{

return [selfinitWithTotalParticles:150];

}

-(id)initWithTotalParticles:(NSUInteger)numberOfParticles{

if((self =[superinitWithTotalParticles:numberOfParticles])){

//duration

duration =kCCParticleDurationInfinity;

//gravity mode

self.emitterMode =kCCParticleModeGravity;

if(self.emitterMode == kCCParticleModeGravity){

//gravity mode:gravity

self.gravity = ccp(0,-90);

//gravity mode:radial

self.radialAccel = 0;

self.radialAccelVar = 0;

self.tangentialAccel = 120;

self.tangentialAccelVar =10;

//gravity mode: speed of particles

self.speed = 180;

self.speedVar = 50;

} elseif(self.emitterMode == kCCParticleModeRadius){

self.startRadius = 100;

self.startRadiusVar = 0;

self.endRadius = 10;

self.endRadiusVar = 0;

self.rotatePerSecond = -180;

self.rotatePerSecondVar = 0;

}

//emitter position

CGSize winSize = [CCDirectorsharedDirector].winSize;

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

self.posVar = CGPointZero;

self.positionType = kCCPositionTypeFree;

//angle

angle = 90;

angleVar =20;

//life of particles

life = 1.5f;

lifeVar =1;

//emits per frame

emissionRate = totalParticles/life;

//color of particles

startColor.r = 0.5f;

startColor.g = 0.5f;

startColor.b = 0.5f;

startColor.a = 1.0f;

startColorVar.r = 0.5f;

startColorVar.g = 0.5f;

startColorVar.b = 0.5f;

startColorVar.a = 0.1f;

endColor.r = 0.1f;

endColor.g = 0.1f;

endColor.b = 0.1f;

endColor.a = 0.2f;

endColorVar.r = 0.1f;

endColorVar.g = 0.1f;

endColorVar.b = 0.1f;

endColorVar.a = 0.2f;

// size, in pixels

startSize = 8.0f;

startSizeVar = 2.0f;

endSize = kCCParticleStartSizeEqualToEndSize;

endSizeVar =0;

self.texture = [[CCTextureCachesharedTextureCache] addImage: @"fire.png"];

//blendFunc

self.blendFunc = (ccBlendFunc){GL\_SRC\_ALPHA,GL\_DST\_ALPHA};

// additive

self.blendAdditive = NO;

}

returnself;

}

@end

通过以上代码，手动创建了一个定制的粒子系统。

关于该粒子系统的具体属性设置，在13.2.4节中将进行详细的解释。

1. 切换到HelloWorldLayer.m，在文件的顶部添加代码，如下所示：

#import "MyParticle.h"

这里只是简单导入了刚刚创建的粒子系统类，以便后续使用。

1. 在init方法中的最后添加一行代码，目的是让当前层支持触摸交互。如下所示：

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

if( (self=[superinit])) {

// //创建一个基本的下雨粒子系统

// CCParticleRain \*rain = [CCParticleRain node];

//

// //将该粒子系统添加到当前层的子节点

// [self addChild:rain z:1];

self.isTouchEnabled = YES;

}

return self;

}

以上代码的作用是让当前层支持触摸交互。

1. 在init方法的下面添加以下方法，如代码清单13-4所示。
2. 在init方法的下面添加方法

-(void)registerWithTouchDispatcher{

CCDirector \*director = [CCDirector sharedDirector];

[[director touchDispatcher]addTargetedDelegate:self priority:0 swallowsTouches:YES];

}

-(BOOL)ccTouchBegan:(UITouch \*)touch withEvent:(UIEvent \*)event{

returnYES;

}

-(void)ccTouchMoved:(UITouch \*)touch withEvent:(UIEvent \*)event{

}

-(void)ccTouchEnded:(UITouch \*)touch withEvent:(UIEvent \*)event{

CGPoint touchLocation = [touch locationInView:[touch view]];

touchLocation = [[CCDirector sharedDirector]convertToGL:touchLocation];

touchLocation =[self convertToNodeSpace:touchLocation];

MyParticle \*myParticle =[MyParticle node];

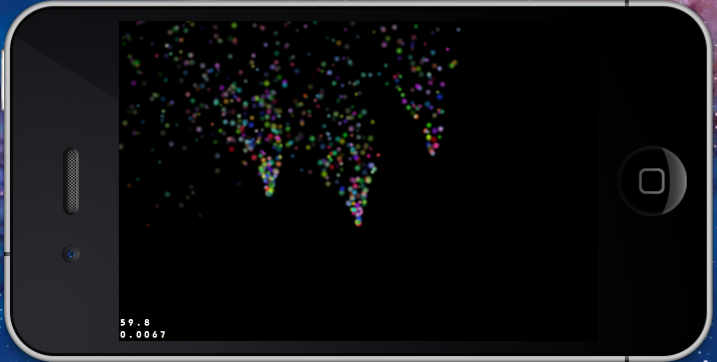
[self addChild:myParticle z:1];

myParticle.position = touchLocation;

}

在ccTouchEnded:withEvent:方法中，首先获取了用户在屏幕当前视图中的触摸点，并将其转换成Cocos2D空间坐标。接着创建并初始化一个myParticle粒子系统，并将其添加为当前层的子节点。最后设置粒子系统的出现位置为用户的触摸点。

编译运行项目，在设备中用手指触摸屏幕或在模拟器中用鼠标单击屏幕的时候，会看到屏幕上出现绚丽的粒子效果，如图13-6所示。



1. 手动创建的炫丽粒子效果
2. 设置粒子系统的属性

粒子系统包含提供粒子的发射器，以及粒子来源的位置信息。其中的每个粒子都用一个单独的图片来标示，并有多个属性可供设置，以确定粒子系统的整体视觉效果。

接下来将对如何设置粒子系统的各个属性进行详细的解释。

#### 属性变动范围

在以上所创建的MyParticle粒子系统类中，有很多属性带有Var后缀。同构Var，可以让对应的属性具有一定的模糊范围。例如，当属性life=5，而lifeVar=1时，意味着每个粒子的平均生命周期是5秒。而lifeVar=1则意味着粒子的生命周期可以是5-1~5+1之间，即4~6秒之间。

如果不需要设置该变动范围，则可以将Var变量设置为0。

#### 粒子数量

在设置粒子系统效果的时候，可以使用totalParticles属性控制粒子的数量。一般情况下在initWithTotalParticles方法中设置，但可以在后续进行修改。粒子数量对于粒子系统的视觉效果有非常明显的影响。

在Xcode中切换到MyParticle.m，查看init方法的内容如下所示:

-(id)init{

return [selfinitWithTotalParticles:150];

}

如果粒子数量过少，可能会发现粒子的效果没有那么明显。但如果粒子数量过多，就会发现粒子会叠加在一起，最后看到一片白光。此外，过多的粒子数量会大大影响游戏帧数。一般情况下粒子数量不应超过2000个。如果可能，应以尽可能少的粒子数量来实现所需要的效果。

有兴趣的读者可以尝试修改init方法中的数字，看看实际的粒子效果会有怎样的变化。

#### 粒子发射器的持续时间

在initWithTotalParticles:方法中找到以下代码：

//粒子发射器的持续时间

duration =kCCParticleDurationInfinity;

这里的duration用于设置粒子的发射时间。kCCParticleDurationInfinity代表粒子效果会永远持续下去。

如果将duration设置为2，则粒子发射器会持续创建2秒的新粒子，然后停止，如下所示：

duration = 2.0f;

如果希望在粒子系统停止发射粒子，且最后发射的粒子已消失时从父节点中自动删除粒子效果节点，可以使用autoRemoveOnFinish属性，如下所示：

self.autoRemoveOnFinish = YES;

该属性对粒子效果不会持续的系统无效。

#### 发射器模式

Cocos2D提供两种发射器模式：

* gravity（重力模式）
* radius（半径模式）

可以使用emitterMode属性设置粒子发射器的模式。即便其他参数相同，两种不同模式下的粒子效果也不同。

##### 重力模式

使用重力模式可以让粒子从一个中心点发散或是朝这个点汇聚，可以用来创建非常动态化的效果。设置重力模式的代码如下所示：

self.emitterMode =kCCParticleModeGravity;

当在项目中设置重力模式后，可以设置以下属性：

//gravity mode:gravity

self.gravity = ccp(0,-90);

//gravity mode:radial

self.radialAccel = 0;

self.radialAccelVar = 0;

self.tangentialAccel = 130;

self.tangentialAccelVar =10;

//gravity mode: speed of particles

self.speed = 180;

self.speedVar = 50;

在以上的代码中，使用gravity属性设置粒子在x轴和y轴的加速度。

* radialAccel：粒子的径向加速度，使用负值代表粒子在朝外扩散时减速。
* tangentialAccel：粒子的切向加速度，使用负值代表粒子沿着顺时针旋转。
* Speed：粒子的运动速度。

尝试修改以上属性值，看看重力模式下的粒子效果。

##### 半径模式

使用半径模式可以让粒子以圆形旋转。此外，还可以用来创建让粒子朝内和朝外旋转的螺旋效果。使用以下代码设置发射器模式，如下所示：

self.emitterMode = kCCParticleModeRadius;

和重力模式类似，半径模式也有自己的特定属性，仅当把发射器模式修改为该模式后才可以使用，如下所示：

self.startRadius = 100;

self.startRadiusVar = 0;

self.endRadius = 10;

self.endRadiusVar = 0;

self.rotatePerSecond = -180;

self.rotatePerSecondVar = 0;

* startRadius属性：决定所发射粒子和粒子效果节点位置的距离。
* endRadius属性：决定粒子要旋转到的位置和节点位置间的距离。
* rotatePerSecond属性：影响粒子移动的方向和速度，以及当startRadius和endRadius不同时的旋转时间。

提示 如果想要产生一个完美的圆形效果，可以将endRadius和startRadius设置相同的值。

#### 粒子发射器的位置

使用position、posVar和positionType属性可以设置粒子的位置和位置类型，如下所示：

//粒子发射器的位置

CGSize winSize = [CCDirectorsharedDirector].winSize;

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

self.posVar = CGPointZero;

self.positionType = kCCPositionTypeFree;

当移动粒子节点的时候，粒子系统的效果也会移动。粒子效果的posVar属性用来决定粒子创建的位置变化。默认情况下，position和posVar都位于节点的中心，如下所示：

self.position = CGPointZero;

self.posVar = CGPointZero;

使用positionType设置粒子效果相对节点的运动，或者粒子效果是否会受到节点位置的影响。例如，想要在玩家角色的周围创建某个粒子效果时，指定positionType让粒子效果跟随节点的运动，如下所示：

self.positionType = kCCPositionTypeGrouped;

而如果想要让粒子随着玩家的运动产生一个轨迹，可以进行以下设置：

self.positionType = kCCPositionTypeFree;

对于类似雾气、火焰、引擎尾烟和类似的粒子效果，一般设置为kCCPositionTypeFree。

#### 粒子的大小

设置startSize和endSize属性可以用像素指定粒子的大小。其中startSize是粒子发射时的大小，endSize是粒子要消失时的大小。相关的代码下所示：

// 粒子大小，以像素为单位

startSize = 8.0f;

startSizeVar = 2.0f;

endSize = kCCParticleStartSizeEqualToEndSize;

endSizeVar =0;

以上代码使用kCCparticleStartSizeEqualToEndSize，致使粒子大小在其声明周期中不会发生变化。

#### 粒子的发射方向

设置angle属性可以指定粒子的初始发射方向。如果angle设置为0，达标粒子会向上发射，但仅在重力模式下才是这样。在半径模式下，angle属性代表粒子被发射的startRadius，数值越高，则发射点会沿半径逆时针移动，如下所示：

//粒子发射角度

angle = 90;

angleVar =20;

#### 粒子的生命周期

粒子的生命周期决定了粒子从开始到消失所经历的时间。使用life属性可以设置粒子的生命周期。粒子的生命周期越长，指定时间内就会有更多的粒子出现在屏幕中。如果已经达到粒子数量的上限，那么在已有粒子消失之前不会产生新的粒子。相关的代码如下所示：

//粒子的生命周期

life = 1.5f;

lifeVar =1;

emissionRate属性会影响每秒所创建的粒子数量。结合totalParticles属性一起使用，可以对粒子系统的显示效果产生很大的影响，如下所示：

//每帧所发射的粒子数量

emissionRate = totalParticles/life;

#### 粒子的颜色

每个粒子都会从起始颜色过渡到最终颜色，从而创建了颜色的变动。因为粒子的默认色彩是黑色，所以必须在粒子系统中设置startColor属性。颜色的类型是ccColor4F结构体，里面包括四个浮点数：r、g、b和a，分别代表色彩的红色、绿色、蓝色和alpha透明通道。每个值的范围都是从0到1，1代表该色彩通道的全色彩值。如果我们需要得到白色的粒子颜色，则需要将r、g、b和a都设置为1。

在示例项目中所创建的粒子系统类中，相关的代码如下所示：

//粒子的颜色

startColor.r = 0.5f;

startColor.g = 0.5f;

startColor.b = 0.5f;

startColor.a = 1.0f;

startColorVar.r = 0.5f;

startColorVar.g = 0.5f;

startColorVar.b = 0.5f;

startColorVar.a = 0.1f;

endColor.r = 0.1f;

endColor.g = 0.1f;

endColor.b = 0.1f;

endColor.a = 0.2f;

endColorVar.r = 0.1f;

endColorVar.g = 0.1f;

endColorVar.b = 0.1f;

endColorVar.a = 0.2f;

#### 粒子的混合模式

粒子的混合模式(blending)是指粒子在显示到屏幕前，对粒子像素值的计算。blendFunc属性使用ccBlendFunc结构体来提供源和目标混合模式，如下所示：

//粒子混合模式

self.blendFunc = (ccBlendFunc){GL\_SRC\_ALPHA,GL\_DST\_ALPHA};

可以尝试使用不同的混合模式，包括GL\_ZERO、GL\_ONE、GL\_SRC\_COLOR、GL\_ONE\_MINUS\_SRC\_COLOR、GL\_SRC\_ALPHA、GL\_ONE\_MINUS\_SRC\_ALPHA、GL\_DST\_ALPHA、GL\_ONE\_MINUS\_DST\_ALPHA等。

混合模式涉及OpenGL ES中的混合计算，这里不做详细介绍，感兴趣的读者可以参考：[www.khronos.org/opengles/documentation/opengles1\_0/html/glBlendFunc.html](http://www.khronos.org/opengles/documentation/opengles1_0/html/glBlendFunc.html)。

通常使用如下的混合模式组合创建附加混合，从而让许多粒子重叠在一起：

self.blendFunc = (ccBlendFunc){GL\_SRC\_ALPHA,GL\_ONE};

也可以简单地将blendAdditive属性设为YES，从而达到相同的效果：

self.blendAdditive = YES;

比较常用的混合模式是使用GL\_SRC\_ALPHA和GL\_ONE\_MINUS\_SRC\_ALPHA的混合模式组合创建透明的粒子，如下所示：

self.blendFunc =(ccBlendFunc){GL\_SRC\_ALPHA,GL\_ONE\_MINUS\_SRC\_ALPHA};

#### 粒子的纹理

如果不为粒子设置纹理图片，所有的粒子都将是扁平的色块。通常需要使用CCTexture2D为粒子设置纹理，如下所示：

//设置粒子的纹理

self.texture = [[CCTextureCachesharedTextureCache] addImage: @"fire.png"];

通过以上设置，就可以在项目职工轻松创建完全定制化的粒子系统。

## plist文件

在13.1节中曾提到过，CCParticleSystem类可以使用plist文件来创建粒子系统。由于plist文件的内容是XML文档，可以轻松地在Xcode中编辑或修改粒子系统的内容。

1. particle.plist文件内容

使用TextEdit打开本章资源文件夹中的particle.plist，会看到以下内容：

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">

<plist version="1.0">

<dict>

<key>angle</key>

<real>180</real>

<key>angleVariance</key>

<real>0</real>

<key>blendFuncDestination</key>

<integer>1</integer>

<key>blendFuncSource</key>

<integer>1</integer>

<key>duration</key>

<real>-1</real>

<key>emitterType</key>

<real>0</real>

<key>finishColorAlpha</key>

<real>1</real>

<key>finishColorBlue</key>

<real>1</real>

<key>finishColorGreen</key>

<real>1</real>

<key>finishColorRed</key>

<real>1</real>

<key>finishColorVarianceAlpha</key>

<real>0</real>

<key>finishColorVarianceBlue</key>

<real>0</real>

<key>finishColorVarianceGreen</key>

<real>0</real>

<key>finishColorVarianceRed</key>

<real>0</real>

<key>finishParticleSize</key>

<integer>16</integer>

<key>finishParticleSizeVariance</key>

<real>0</real>

<key>gravityx</key>

<real>0</real>

<key>gravityy</key>

<real>0</real>

<key>maxParticles</key>

<real>10</real>

<key>maxRadius</key>

<real>100</real>

<key>maxRadiusVariance</key>

<real>0</real>

<key>minRadius</key>

<real>0</real>

<key>particleLifespan</key>

<real>2</real>

<key>particleLifespanVariance</key>

<real>0</real>

<key>radialAcceleration</key>

<real>0</real>

<key>radialAccelVariance</key>

<real>0</real>

<key>rotatePerSecond</key>

<real>0</real>

<key>rotatePerSecondVariance</key>

<real>0</real>

<key>rotationEnd</key>

<real>0</real>

<key>rotationEndVariance</key>

<real>0</real>

<key>rotationStart</key>

<real>0</real>

<key>rotationStartVariance</key>

<real>0</real>

<key>sourcePositionVariancex</key>

<real>0</real>

<key>sourcePositionVariancey</key>

<integer>160</integer>

<key>sourcePositionx</key>

<integer>480</integer>

<key>sourcePositiony</key>

<integer>160</integer>

<key>speed</key>

<integer>500</integer>

<key>speedVariance</key>

<integer>0</integer>

<key>startColorAlpha</key>

<real>1</real>

<key>startColorBlue</key>

<real>1</real>

<key>startColorGreen</key>

<real>1</real>

<key>startColorRed</key>

<real>1</real>

<key>startColorVarianceAlpha</key>

<real>0</real>

<key>startColorVarianceBlue</key>

<real>0</real>

<key>startColorVarianceGreen</key>

<real>0</real>

<key>startColorVarianceRed</key>

<real>0</real>

<key>startParticleSize</key>

<integer>16</integer>

<key>startParticleSizeVariance</key>

<real>0</real>

<key>tangentialAcceleration</key>

<real>0</real>

<key>tangentialAccelVariance</key>

<real>0</real>

<key>textureFileName</key>

<string>star2.png</string>

<key>textureImageData</key>

<string></string>

</dict>

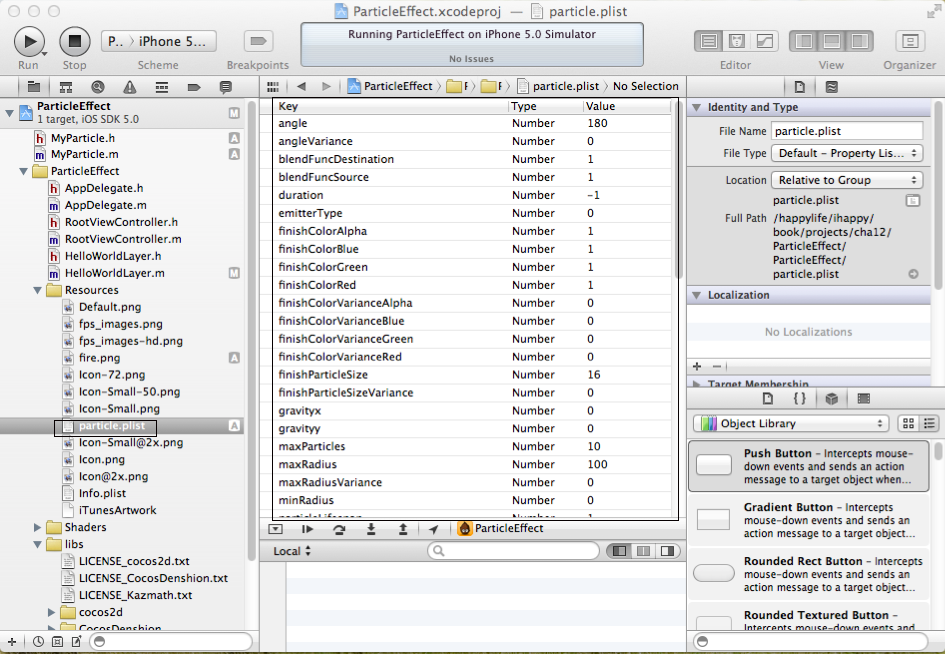
</plist>

其中，key标签的内容是粒子系统的各属性名称，而real和string标签中的内容是具体的属性值。最后以H4开头的一长串内容是粒子的纹理图片数据。

1. 使用plist文件创建粒子系统

接下来了解如何使用plist文件创建粒子系统。

把particle.plist拖到Xcode的Resources中，确保选中“copy items into destination’s group(if needed)”，然后单击打开该文件，会看到所有相关属性的键对值，如图13-7所示。



1. particle.plist中的属性键对值

开发者可以直接更改属性列表中的任何属性，从而得到自己所需要的属性。

在Xcode中切换到HelloWorldLayer.m，将ccTouchEnded:withEvent:方法中的内容使用以下代码替代，如代码清单13-5所示。

1. ccTouchEnded:withEvent:方法中的内容替换

-(void)ccTouchEnded:(UITouch \*)touch withEvent:(UIEvent \*)event{

CGPoint touchLocation = [touch locationInView:[touch view]];

touchLocation = [[CCDirectorsharedDirector]convertToGL:touchLocation];

touchLocation =[self convertToNodeSpace:touchLocation];

// MyParticle \*myParticle =[MyParticle node];

//

// [self addChild:myParticle z:1];

//

// myParticle.position = touchLocation;

CCParticleSystemQuad \*particle = [CCParticleSystemQuad particleWithFile:@"particle.plist"];

[self addChild:particle z:1];

particle.position = touchLocation;

}

和以前的方法相比，首先注释了使用手动创建的粒子系统类MyParticle的相关代码。然后使用particle.plist文件创建并初始化了一个CCParticleSystemQuad类，将其添加为当前层的子节点，并设置其出现的位置为触摸点。

编译运行项目，当手指触碰（设备）或鼠标单击（模拟器）时，屏幕中出现繁星点点，如图13-8所示。



1. 使用plist文件创建的繁星点点的粒子效果

## 粒子工具Particle Designer

现在，大家已经熟知了如何手动创建粒子系统，并逐一调整其属性值，但遗憾的是以上的操作无法立即看到粒子系统的视觉效果。即便是一个经验丰富的粒子效果创建人员，也无法凭空想象出最终的效果。如果能有一款工具帮助大家视觉化创建粒子效果，无疑是最佳之选。Particle Designer就是这样一款好工具。

本节将介绍Particle Designer的下载安装，以及如何使用该工具创建项目所需的粒子效果。

1. 下载安装Particle Designer

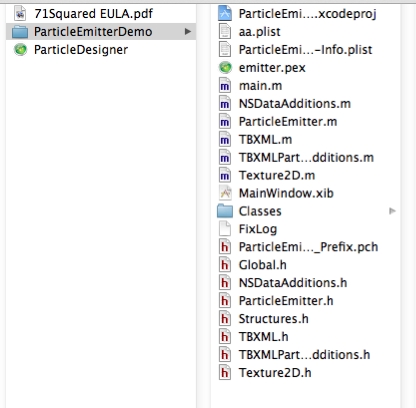
Particle Designer是一款非常优秀的粒子效果创建工具，虽然免费版也可以使用，但很多功能被限制。如果在游戏开发中要经常用到粒子系统，建议购买这款工具。

单击链接（http://particledesigner.71squared.com/）即可进入Particle Designer的官方网站，购买界面如图13-9所示。



1. Particle Designer的购买界面

当前的最新版本是1.3.7，从下载页面下载之后解压缩会看到其中的内容，如图13-10所示。

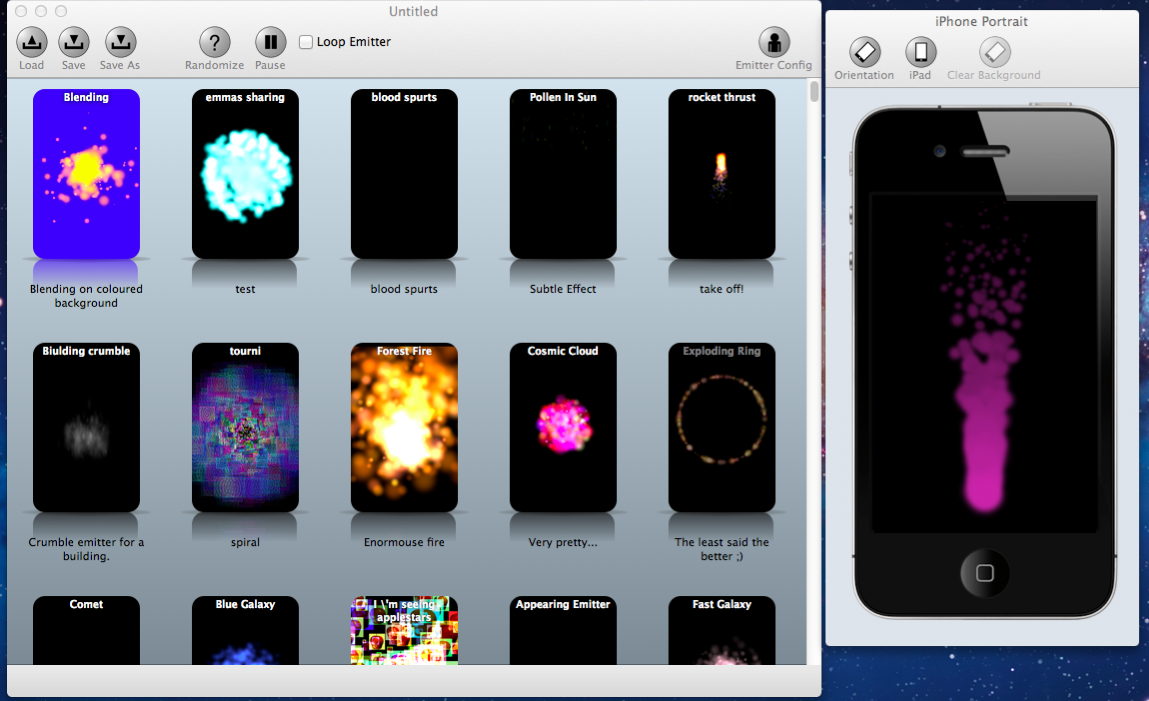


1. Particle Designer安装包的内容

其中，PDF文件是一些基本介绍，ParticleEmitterDemo是一个示例项目，而带有粒子效果样式的图标则是Particle Designer的主程序。

1. Particle Designer工具栏

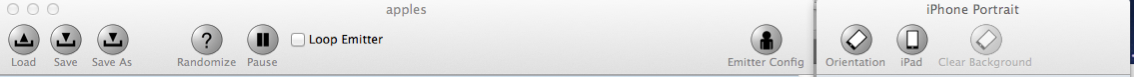
打开Particle Designer（注册版）后，会显示工具中已有的粒子效果，如图13-11所示。



1. Particle Designer的主界面

其中左侧是系统提供的粒子系统，单击某粒子系统，会在右侧的模拟器中看到放大的粒子效果。

顶部工具栏如图13-12所示。



1. 粒子系统顶部工具栏

各按键的作用如表13-2所示。

1. 按键及作用

|  |  |
| --- | --- |
| 按键名称 | 作用 |
| Load | 载入一个已有的粒子系统，Save是保存当前的粒子系统，而Save As则是另存为，这些按钮的作用一目了然 |
| Randomize | 随机设置粒子系统的所有属性值，以产生一种随机的粒子效果 |
| Pause | 暂停对当前粒子效果的预览 |
| Emitter Config | 调整粒子系统的属性，稍后详细说明 |
| Orientation | 切换模拟器的朝向 |
| iPad | 让模拟器在iPhone和iPad间切换 |

重点了解Emitter Config。

1. 如何设置Emitter Config属性

选择一种粒子效果Spinning Peas，单击Emitter Config按钮，会看到一个设置界面，如图13-13所示。



1. 单击Emitter Config按钮后的设置界面

在设置界面左侧做的任何属性调整都会立即在右侧的模拟器上看到对应的视觉效果，非常方便。

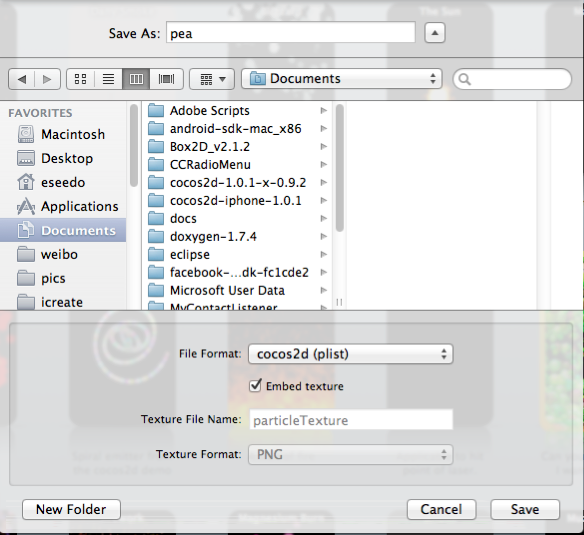
其中各部分的作用如表13-5所示。

1. 属性及作用

|  |  |
| --- | --- |
| 名称 | 作用 |
| Particle Configuration | 设置粒子的数量、生命周期、起始和结束大小、旋转角度、发射角度等 |
| Background Color | 在这里设置的其实并非是粒子系统本身的属性，而是用了更清晰的了解粒子系统相对背景的视觉效果 |
| Emitter Type | 粒子发射器类型，即重力模式和半径模式 |
| Gravity Configuration(Radial Configuration) | 根据Emitter Type的不同，下面会显示Gravity Configuration或Radial Configuration。其中Gravity Configuration中设置的是重力模式下的粒子系统属性，而Radial Configuration设置的是半径模式下的粒子系统属性。其中唯一需要说明的是Deg.Per Second其实就是rotatePerSecond属性 |
| Emitter Location | 粒子发射器的位置 |
| Particle Color | 粒子的颜色 |
| Blend Function | 粒子的混合模式 |

1. 如何在项目中使用粒子系统

通过Emitter Config对粒子系统的各种属性进行调整后，单击Save As，在File Format中选择cocos2d(plist)，选择Embed texture，如图13-14所示。



1. 保存设置完成的粒子效果文件

将创建的pea.plist文件拖到Xcode的Resources中，确保选中“Copy item into destination’s group(if needed)”。切换到HelloWorldLayer，找到init方法，使用代码清单13-6内容替代其中的代码。

1. init方法的内容替换

-(id) init

{

// always call "super" init

// Apple recommends to re-assign "self" with the "super" return value

if( (self=[superinit])) {

// //创建一个基本的下雨粒子系统

// CCParticleRain \*rain = [CCParticleRain node];

//

// //将该粒子系统添加到当前层的子节点

// [self addChild:rain z:1];

CCParticleSystemQuad \*pea = [CCParticleSystemQuad particleWithFile:@"pea.plist"];

[self addChild:pea z:1];

// self.isTouchEnabled = YES;

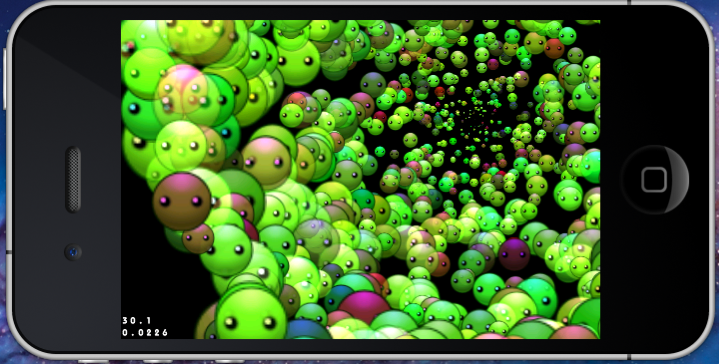
}

returnself;

}

以上代码只是使用新的plist方法创建并初始化一个粒子系统，并将其添加为当前层的子节点。

编译运行项目，可以看到屏幕中出现刚才使用Particle Designer创建的粒子效果，如图13-15所示。



1. 使用Particle Designer创建的粒子效果

## 本章小结

本章首先了解了粒子系统的基本知识，并对Cocos2D中的粒子系统类做了详细介绍；接着通过一个示例项目ParticleEffect带领大家手动创建一个粒子系统，并对其中的属性一一做了解释；然后学习如何使用plist文件创建和修改粒子系统；最后详细介绍一款非常优秀的粒子系统创建工具Particle Designer的安装及使用。

完成本章的学习，就可以在自己的项目中轻松创建并使用各种炫目吸引人的粒子效果。

第14章将介绍如何在项目中整合苹果官方的Game Center，从而让大家一起分享游戏的快乐！