知易 Cocos2D-iPhone 游戏开发教程 07

知易 iPhone 游戏开发 http://blog.sina.com.cn/carol

目录

为什么需要粒子系统.....1
粒子系统的组成.....3
使用例子系统.....8
关于例子系统示例.....13

为什么需要粒子系统

如果只是做卡通效果的游戏，我们可以通过动画实现所有游戏视觉效果，ZYG006 中的爆炸效果就是这样的。但是如果需要更加真实的效果模拟，实现更加随机的、栩栩如生的烟雾、闪电、雨雪、火花掉落效果，我们就必须使用粒子系统。粒子系统让你的游戏显得更加真实而富有生命感（Ben Britten Smith-《iPhone Advanced Projects》）。

通过对这些自然现象的分析我们发现：这些现象的出现并不涉及一个可以触摸的、很明确的实体，而且现象的本身是动态的（随时间迅速变化的），而这种变化的效果是由大量微小微粒组合而成的，大量的粒子效果叠加成了我们看到的整体效果。这样一来，明确的、可以用数学公式表述的图形、图像效果都很难模拟出这样的效果,因为其中包含着大量的、不确定性的、随机性、混乱的模糊内涵。

物理学上关于大量粒子无规则运动的学科是热力统计学，一个令大家耳熟能详的概念是“熵”。关于这个物理量的典型描述是热力学第二定律，也称为“熵增原理”。当然还有一个

更加令人费解但确实又是无可奈何的必然概念是“系综”。

无论是我们主观的分析判断也好，还是热力统计学的基本定律也好，共同的一点认识就是：这些自然现象的存在涉及到大量无规则运动的微粒。因此，为了模拟这样的系统所产生的现象，我们要建立这样一个粒子系统：

1）包含大量所谓粒子对象（Particles）。

2）宏观特性： 每一个例子都要遵守的主要规律。

3）微观特性：每个例子都在宏观属性上的随机、变异特性。

4）过程动态特性：每个粒子系统模拟的都是一个不断变化的动态效果而不是静态不变的，因此这就有一个持续更新状态的要求。

按照预先设计好的规则，不断产生、喷射出大量粒子，每个粒子按照自己的运动参数不断变化、运动，大量粒子效果叠加成为我们所需要的宏观现象。这样我们就可以模拟出许多栩栩如生的自然效果。

大约在 1982 年-1983 年，一个叫做威廉姆·瑞午斯（William Reeves）的游戏开发者最先使用了这样的系统。他的算法实现了宏观上对粒子群整体的控制，同时保证了微观上各个粒子的无规则乱序运动。这就比较完美的给游戏系统带来的全新的体验。

当然，在一些简单的卡通类游戏和纸牌游戏中添加几丝粒子效果，也可以为玩家带来欣喜、兴奋的视觉体验。让你的游戏显得更加精致而令人珍惜。

粒子系统的组成

一个例子系统通常由以下几个关键部分组成：

- 粒子（Particles）

  每一个粒子就是一个图形对象，可以使一色点或者一个图片。下面就是 CocosNoded 关于粒子的定义：（文件 CCParticleSystem.h）

```
typedef struct sParticle
{
    CGPoint   pos;            // A 位置
    CGPoint startPos;         // A 开始位置
    CGPoint   dir;            // A 运动方向
    float  radialAccel;       // 径向加速度。逃离发射原点的速度
    float  tangentialAccel;   // 切线加速度。围绕发射原点旋转速度
    ccColor4F color;          // A 颜色
    ccColor4F deltaColor;     // B 颜色变化
    float  size;              // A 大小
    float  deltaSize;         // B 大小变化
    float  angle;             // A 角度
```

```
        float  deltaAngle;      // B 角度变化量
        float  life;            // A 存在时间
    } Particle;
```

每一个粒子都有自己独立的属性，按照之前关于粒子系统的定义包括两大类:

A 类，表述粒子的主属性，也就是宏观系统需要的宏观特性。

B 类，代表每个例子的无规则变化属性，也就是微观特性。

这里需要说明的是，发射器对象在生成每一个的时候，已经会在宏观特性的基础之上设置每一个粒子的个性，B 类变量保存了这种变化在生命周期中的单步变换量。

- 发射器（emitter）

发射器对象就是一个粒子系统的整体，一片云，一团雾，一次闪电，一股烟都是由一个独立的粒子系统来模拟的，Cocos2d-iPhone 中关于粒子系统的定义如下：(文件 CCParticleSystem.h)：

```
@interface CCParticleSystem : CCNode <CCTextureProtocol>
{
    int id;

    // is the particle system active ?
    BOOL active;
    // duration in seconds of the system. -1 is infinity
    float duration;
    // time elapsed since the start of the system (in seconds)
    float elapsed;

    /// Gravity of the particles
    CGPoint gravity;        //A 所有粒子的纵、横运动速度。

    // position is from "superclass" CocosNode
    // Emitter centerOfGravity position
    CGPoint centerOfGravity; //粒子重力原点。
    // Position variance
    CGPoint posVar;         //B 粒子喷发原点的宽高变化范围。

    // The angle (direction) of the particles measured in
degrees
    float angle;            //A 粒子喷发角度
    // Angle variance measured in degrees;
    float angleVar;         //B 粒子喷发角度的变化范围

    // The speed the particles will have.
```

```
float speed;            //A 粒子运动速度
// The speed variance
float speedVar;         //B 粒子运动速度的变化范围

// Tangential acceleration
float tangentialAccel;  //A 粒子切向加速度
// Tangential acceleration variance
float tangentialAccelVar;  //B 粒子切向加速度的变化范围

// Radial acceleration
float radialAccel;      //A 粒子径向加速度
// Radial acceleration variance
float radialAccelVar;   //B 粒子径向加速度变化范围

// start ize of the particles
float startSize;        //A 粒子起始大小
// start Size variance
float startSizeVar;     //B 粒子起始大小尺寸变化范围
// End size of the particle
float endSize;          //A 粒子完毕尺寸
// end size of variance
float endSizeVar;       //B 粒子完毕尺寸变化范围

// How many seconds will the particle live
float life;             //A 粒子生命周期
// Life variance
float lifeVar;          //B 粒子生命周期变化范围

// Start color of the particles
ccColor4F startColor;   //A 粒子开始颜色
// Start color variance
ccColor4F startColorVar;   //B 粒子开始颜色变化范围
// End color of the particles
ccColor4F endColor;     //A 粒子结束颜色
// End color variance
ccColor4F endColorVar;  //B 粒子结束颜色变化范围

// start angle of the particles
float startSpin;        //A 粒子开始自旋角度
// start angle variance
float startSpinVar;     //B 粒子开始自旋角度变化范围
// End angle of the particle
float endSpin;          //A 粒子结束自旋角度
// end angle ariance
float endSpinVar;       //B 粒子结束自旋角度变化范围

// Array of particles
Particle *particles;
```

```
        // Maximum particles
        int totalParticles;
        // Count of active particles
        int particleCount;

        // additive color or blend
        BOOL blendAdditive;
        // color modulate
        BOOL colorModulate;

        // How many particles can be emitted per second
        float emissionRate;     //粒子喷发速率
        float emitCounter;

        // Texture of the particles
        CCTexture2D *texture_;
        // blend function
        ccBlendFunc   blendFunc_;

        // movment type: free or grouped
        tPositionType positionType_;

        // Whether or not the node will be auto-removed when there
are not particles
        BOOL   autoRemoveOnFinish_;

        //  particle idx
        int particleIdx;
    }
```

通过以上定义我们可以看到，A 类属性定义了整个粒子系统的宏观特性，B 类属性定义了粒子微观特性，B 类属性是由粒子系统在生成每一个微观粒子时通过随机函数 CCRANDOM_MINUS1_1 随机赋予的，一旦被激活，就可以模拟整个系统的随机、混乱的效果。

粒子和发射器对象描述了一个粒子系统的复杂性的全部属性，我们还可以根据自己游戏设计内容，通过增加新的 A、B 类属性来增加上述复杂性：闪烁、随机扰动等等。

- 动态效果

粒子和发射器是我们程序的数据对象，下面的两个过程才实现了整个粒子系统的动态效果。

  - 新粒子诞生

下面这个函数是用于生成每一个粒子的：

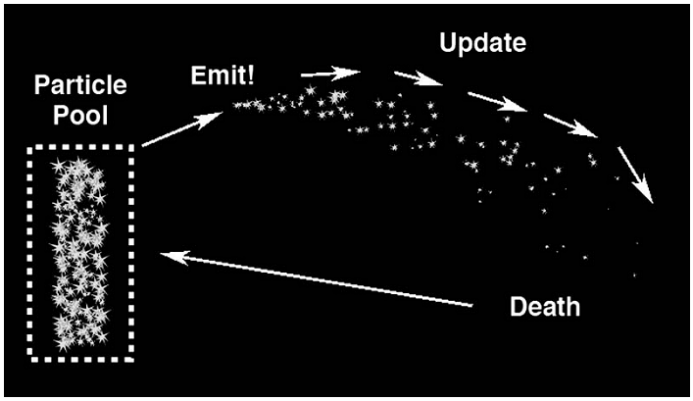
-(void) initParticle: (Particle*) particle

我们看到，每一个粒子在生成的瞬间被赋予不同的属性，其中 B 类属性是通过系统的随机函数 CCRANDOM_MINUS1_1()来随机赋予每一个粒子唯一属性变化的。

由于系统是通过 CGPoint posVar;来描述粒子生成位置的变化范围的，因此 Cocos2d-iPhone 的粒子诞生范围只能是一个点、纵横直线、矩形面而不能是其他形状。因此，对于其他发生效果可以通过重新实现该函数来模拟。

- 批量粒子变化

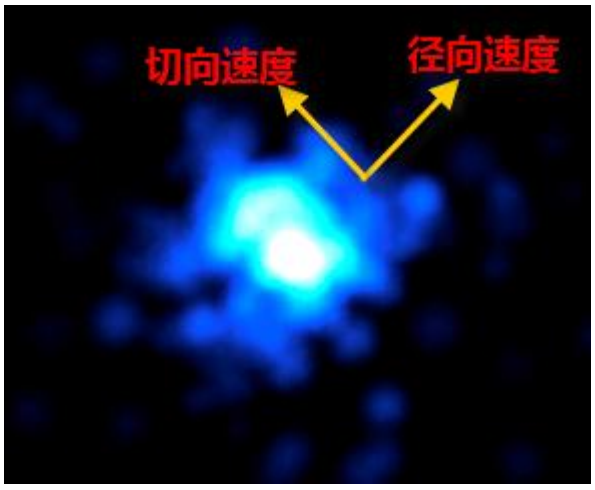
粒子一旦诞生都要经历完整的生命周期：



粒子系统的下述函数负责实现整个粒子系统的演化：

-(void) step: (ccTime) dt

下图是系统提供的星云效果图：



我们可以看到：

1）由于粒子都是从原点喷发出来的，因此原点附近（图中白色部分）积累了大量粒子。

2）每个粒子除了基本的速度和方向之外，还有径向、切向的速度。

使用例子系统

通过前面的分析，我们已经掌握了一个粒子系统的基本组成：粒子、发射器、动态过程。这几个组成部分的具体实现算法 Cocos2d-iPhone 已经替我们实现了。在粒子系统在算法上中间颜色的计算、内存分配效率是主要的难点，这里我们不做深入的说明，我们需要说明的是如何利用上述来实现我们在具体游戏开发中需要的效果。

使用例子系统的基本原则是：通过设置不同的参数属性来实现一些完全不一样的效果。Cocos2d - iPhone 系统中，可以通过以下 3 个方面来设置实现不同的效果：

1）粒子的喷发位置

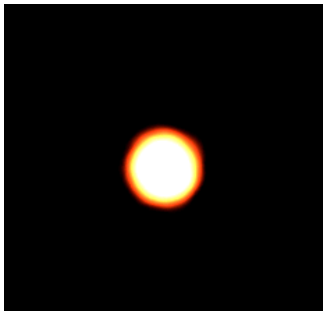
控制粒子的喷发点，实现预期的效果。主要包括以下 3 种方式：

- 点控制

模拟太阳，为了确保基本上保持圆形，因此粒子喷发点为一个严格的点：

```
// emitter position
CGSize winSize = [[CCDirector sharedDirector] winSize];
self.position = ccp(winSize.width/2, winSize.height/2);
posVar = CGPointZero;
```

输出效果如下：



- 线控制

模拟雨、雪效果，粒子喷发点是条水平的直线：

```
// emitter position
self.position = (CGPoint) {
    [[CCDirector sharedDirector] winSize].width / 2,
    [[CCDirector sharedDirector] winSize].height + 10
};
posVar = ccp( [[CCDirector sharedDirector] winSize].width / 2, 0 );
```

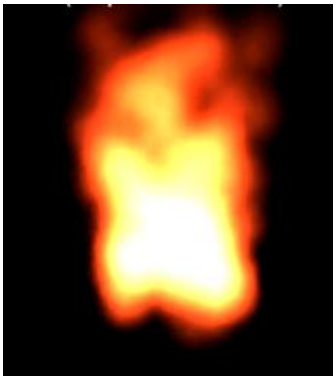
输出效果：

- 面控制

模拟一片火焰，粒子的喷发点是一个指定的矩形区域：

```
// emitter position
CGSize winSize = [[CCDirector sharedDirector] winSize];
self.position = ccp(winSize.width/2, 60);
posVar = ccp(40, 20);
```

输出效果：



- 路径

如果需要直接控制输出位置，需要进一步修改 cocos2d-iphone 的基础的实

现代码：

```
-(void) initParticle: (tCCParticle*) particle
{
    CGPoint v;

    // position
    // XXX: source should be deprecated.
    particle->pos.x = (int) (centerOfGravity.x + posVar.x * CCRANDOM_MINUS1_1());
    particle->pos.y = (int) (centerOfGravity.y + posVar.y * CCRANDOM_MINUS1_1());
```

也可以考虑让一个后者多个“太阳”类点效果按照指定的路径移动。

2）粒子的属性

发射器的所有参数都可以修改来实现游戏的效果，主要包括以下几类：

- 速度类
  - 径向速度
  - 切向速度
  - 速度
  - 速度的角度
- 颜色类
  - 开始颜色极其变化范围
  - 结束颜色极其变化范围
  - 结束颜色饱和度、颜色调和（blend）
- 尺寸类
  - 开始尺寸及其变化范围
  - 结束尺寸及其变化范围
- 自旋类
  - 开始自旋及其变化范围
  - 结束自旋及其变化范围

这些复杂参数的组合设置可以实现令人难以想象的效果，读者可以仔细分析类库自带的 ParticleTest 示例。花上一些时间试试哪些参数，你可以慢慢找到感觉的。

在这里，我们找一个最有代表性的实例：

使用 CCQuadParticleSystem 为发射器。

下面的代码是实现上图效果的关键：

```
// angle      360 度的喷射角范围，确保是一个圆形。
emitter.angle = 90;
emitter.angleVar = 360;

// speed of particles 设置一个较大的发射速度。
emitter.speed = 160;
emitter.speedVar = 20;

// radial
emitter.radialAccel = -120; 径向负加速度将缓慢拉回喷射出去的粒子
emitter.radialAccelVar = 0;

// tagential
emitter.tangentialAccel = 30; 切向加速度让整个图像旋转起来。
emitter.tangentialAccelVar = 0;

// emitter position
emitter.position = ccp(160,240);
emitter.posVar = CGPointZero;

// life of particles
emitter.life = 3;
emitter.lifeVar = 1;

// spin of particles

emitter.startSpin = 0;   设定粒子自旋速度的变化。
emitter.startSpinVar = 0;
emitter.endSpin = 0;
emitter.endSpinVar = 2000;
```

3）粒子图像

千万不要忽略因为粒子会很小而忽略粒子的图像。如果你没有设置 texture 属性，系统会给你一个默认正方形的图像。多数情况下，这是最差的效果。小的五角星，雪花状都

会让你的游戏增添一分灵气。

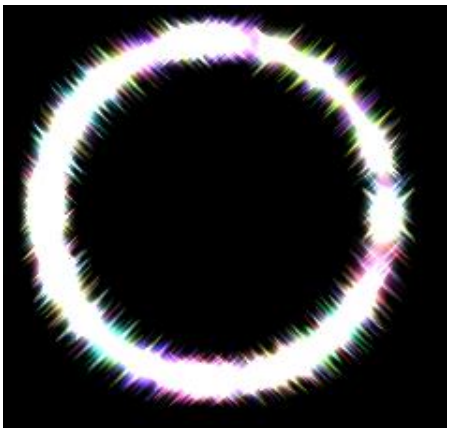
4）喷射速率

如果不是那个叫“DemoRing”的例子，我是很难想到通过直接设置 emissionRate 来实现圆环式喷发效果。通过设置速度变化范围我们可以实现不同宽度的圆环。

emissionRate 通常都是通过下面的公式计算的：

```
// emits per frame
emitter.emissionRate = emitter.totalParticles/emitter.life;
```

直接设置一个超过当前粒子系统总数的大数值可以实现环形的喷发效果，径向负值加速度可以在生命周期以内连续缩小放大，切向加速度可以让圆环旋转起来。如下图所示：



对应的代码如下，使用 CCParticleFlower 为发射器：

```
    // Radial
    emitter.radialAccel = -80;
    emitter.radialAccelVar = 0;

    // tagential
    emitter.tangentialAccel = 30;
    emitter.tangentialAccelVar = 0;


    emitter.texture = [[CCTextureCache sharedTextureCache] addImage:
@"stars-grayscale.png"];

    emitter.lifeVar = 0;
    emitter.life = 10;
    emitter.speed = 120;
    emitter.speedVar = 00;

    emitter.emissionRate = 10000;
```

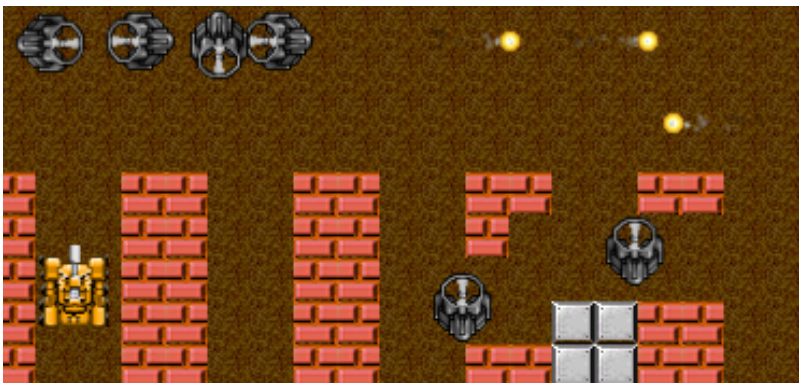
除了以上 4 个方面，通过深入发射器的实现代码或许还会有新的发现和用法。但对于

通常的游戏效果模拟来说已经基本足够了。

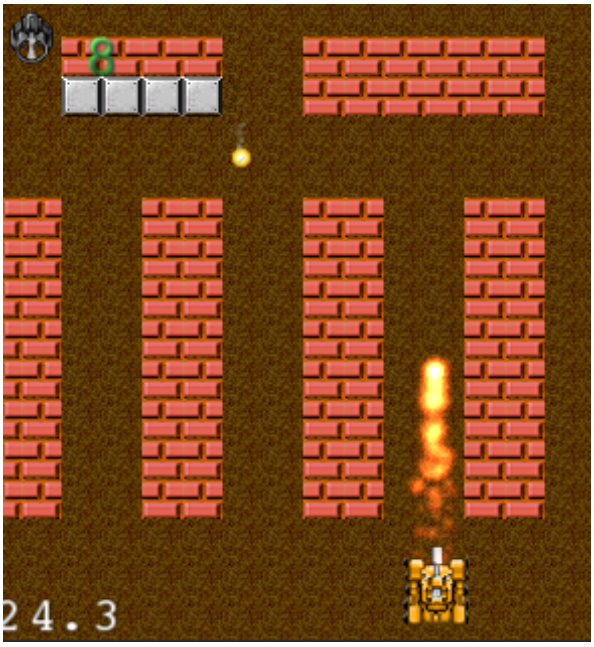
关于例子系统示例

有了前面对粒子系统的全面了解，本章示例是很容易理解的。我们需要给 ZYG006 例子中的火炮系统增加一些生动的火焰和烟雾效果。

我们计划给敌方坦克的火炮发射增添一点烟雾效果：



同时，让我方坦克的火炮系统增添火焰喷射效果：



为此，我们需要修改 BulletSprite 类，为了方便我们让每一个子弹都具备两种粒子效果，

因此我们将粒子效果增加到子弹的类定义中：

```
@interface BulletSprite : CCSprite {
    CCLayer *gLayer;
    CCParticleSystem  *emitter;
    CCParticleSystem  *emitter2;
}
```

在子弹初始化函数中分别使用烟雾类(CCParticleSmoke)和太阳类(CCParticleSun ）来实现烟雾和火焰效果。

为了实现按照坦克的类型实现粒子效果，我们为子弹类新增消息：

```
-(void) FireFrom:(CGPoint)ptFrom To:(CGPoint)ptTo bulletType:(int)bType;
```

这样我们就可以在开炮的实现中判断当前坦克类型，再将粒子系统附属在子弹身上，这样子弹移动的时候就自然实现了烟火效果。