第 5 章

精　　灵

在前面的章节用到了精灵对象但没有深入地介绍，本章我们深入地介绍精灵的使用。精灵是游戏中非常重要的概念，围绕着精灵还有很多概念，如精灵帧缓存、动作和动画等内容。

5.1　Sprite 精灵类

Sprite 类图如图 5-1 所示，从图中可见 Sprite 是 Node 子类，Sprite 包含很多类型，例如广告牌精灵类 BillBoard 也都属于精灵。

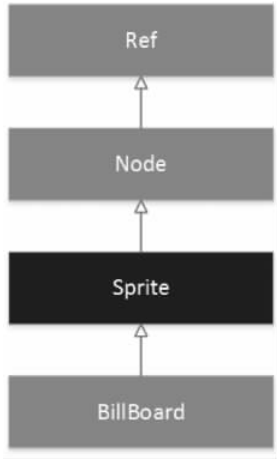

图 5-1　Sprite 类图

Sprite 类直接继承了 Node 类，具有 Node 的基本特征。此外，Cocos2d-x 现在还提供了 3D 精灵类 Sprite3D，关于 Cocos2d-x 中的 3D 特征将在第 14 章详细介绍。

5.1.1　创建 Sprite 精灵对象

创建精灵对象有多种方式，其中常用的函数如下：

(1) static Sprite * create()：创建一个精灵对象，纹理(texture)表示物体表面细节的一幅或几幅二维图形，也称纹理贴图，当把纹理按照特定的方式映射到物体表面上的时候能使精灵看上去更加真实。该属性需要在创建后设置。

(2) static Sprite * create(const std::string &filename)：指定图片创建精灵。

(3) static Sprite * create(const std::string &filename, const Rect &rect)：指定图片和裁剪的矩形区域来创建精灵。

(4) static Sprite * createWithTexture(Texture2D * texture)：指定纹理创建精灵。

(5) static Sprite * createWithTexture(Texture2D * texture, const Rect &rect, bool rotated=false)：指定纹理和裁剪的矩形区域来创建精灵，第三个参数指定是否旋转纹理，默认不旋转。

(6) static Sprite * createWithSpriteFrame(SpriteFrame * pSpriteFrame): 通过一个精灵帧对象创建另一个精灵对象。

(7) static Sprite * createWithSpriteFrameName(const std::string &spriteFrameName): 通过指定帧缓存中精灵帧名创建精灵对象。

上述 create 函数在前面的章节中介绍过,而且 create 函数比较简单,就不再介绍了。

5.1.2 实例: 使用纹理对象创建 Sprite 对象

使用纹理 Texture2D 对象创建 Sprite 对象是使用 createWithTexture 函数实现的。本节通过一个实例介绍纹理对象创建 Sprite 对象使用,如图 5-2 所示。其中,地面上的草是放在背景(见图 5-3)中的,场景中的两棵树是从图 5-4 所示的"树"纹理图片中截取出来的。图 5-5 所示是树的纹理坐标,注意它的坐标原点在左上角。

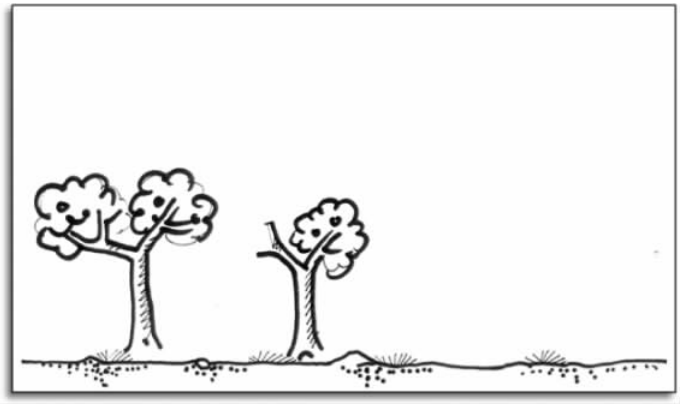

图 5-2 创建 Sprite 对象实例

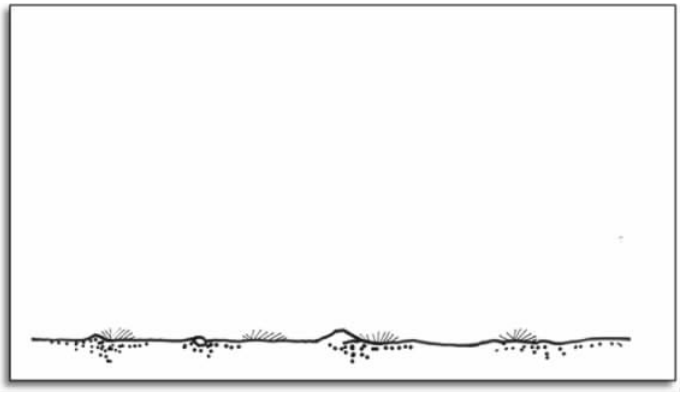

图 5-3 场景背景图片

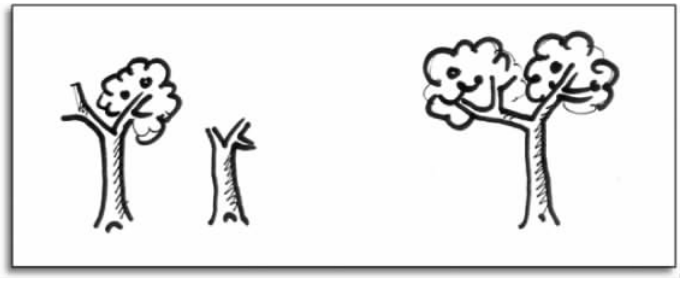

图 5-4 “树”纹理图片

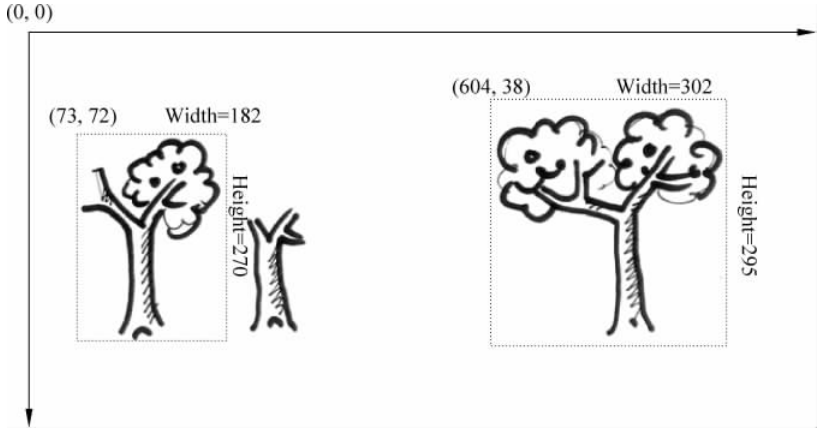

图 5-5 “树”纹理坐标

HelloWorldScene.cpp 实现的 init 函数代码如下：

```
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    auto background = Sprite::create("background.png");                          ①
    background->setAnchorPoint(Vec2::ZERO);                                      ②
    this->addChild(background,0);

    auto tree1 = Sprite::create("tree1.png",Rect(604, 38, 302, 295));            ③
    tree1->setPosition(Vec2(200,230));
    this->addChild(tree1,0);

    Texture2D* cache = Director::getInstance()->getTextureCache()->addImage("tree1.
```

```
png");                                                                    ④
    auto tree2 = Sprite::create();                                        ⑤
    tree2->setTexture(cache);                                             ⑥
    tree2->setTextureRect(Rect(73, 72,182,270));                          ⑦
    tree2->setPosition(Vec2(500,200));
    this->addChild(tree2,0);

    return true;
}
```

其中,代码第①行 Sprite::create("background.png")通过 background.png 图片创建精灵,background.png 图片如图 5-3 所示;代码第②行是设置背景的锚点。

代码第③行 Sprite::create("tree1.png",Rect(604, 38, 302, 295))通过 tree1.png 图片和矩形裁剪区域创建精灵,矩形裁剪区域为(604, 38, 302, 295),如图 5-5 所示。

Rect 类可以创建矩形裁剪区。Rect 构造函数如下:

```
Rect (float x, float y, float width, float height)
```

其中,x,y 是 UI 坐标,坐标原点在左上角;width 是裁剪矩形的宽度;height 是裁剪矩形的高度。

代码第④行通过纹理缓存 TextureCache 创建纹理 Texture2D 对象,通过 Director 的 getTextureCache()函数可以获得 TextureCache 实例,TextureCache 的 addImage("tree1.png")函数可以创建纹理 Texture2D 对象,其中的 tree1.png 是纹理图片名。

代码第⑤行创建一个空的 Sprite 对象,所以还要通过后面的很多函数设置它的属性。其中,代码第⑥行 tree2-> setTexture(cache)是设置纹理,代码第⑦行 tree2-> setTextureRect(Rect(73, 72,182,270))是设置纹理的裁剪区域。

5.2 精灵的性能优化

游戏是一种很耗费资源的应用,特别是移动设备中的游戏,性能优化是非常重要的。性能优化有很多方面,这一节只是介绍精灵相关的性能优化;关于其他方面的优化,会在后面的第 22 章介绍。

精灵的性能优化可以使用精灵表和缓存。下面从这两个方面介绍精灵的性能优化。

5.2.1 使用纹理图集

纹理图集(texture atlas)也称为精灵表(sprite sheet),它把许多小的精灵图片组合到一张大图里面。使用纹理图集(或精灵表)主要有如下优点:

(1) 减少文件读取次数,读取一张图片比读取一堆小文件要快。

(2) 减少 OpenGL ES 绘制调用并且加速渲染。

(3) 减少内存消耗。OpenGL ES 1.1 仅仅能够使用 2 的 n 次幂大小的图片(即宽度或者高度是 2,4,8,16,32,64,…)。如果采用小图片 OpenGL ES1.1 会分配给每张图片 2 的 n

次幂大小的内存空间，即使这张图片达不到这样的宽度和高度也会分配大于此图片的 2 的 n 次幂大小的空间。那么运用这种图片集的方式将会减少内存碎片。虽然在 Cocos2d-x 2.0 后使用了 OpenGL ES 2.0，它不会再分配 2 的几次幂的内存块了，但是减少读取次数和绘制的优势依然存在。

(4) Cocos2d-x 全面支持 Zwoptex 精灵表制作工具(http://www.zwopple.com/zwoptex/)和 TexturePacker 精灵表制作工具(http://www.codeandweb.com/texturepacker)，所以创建和使用纹理图集是很容易的。

通常可以使用纹理图集制作工具 Zwoptex 和 TexturePacker 设计和生成纹理图集文件(见图 5-6)，以及纹理图集坐标文件(plist)组成。

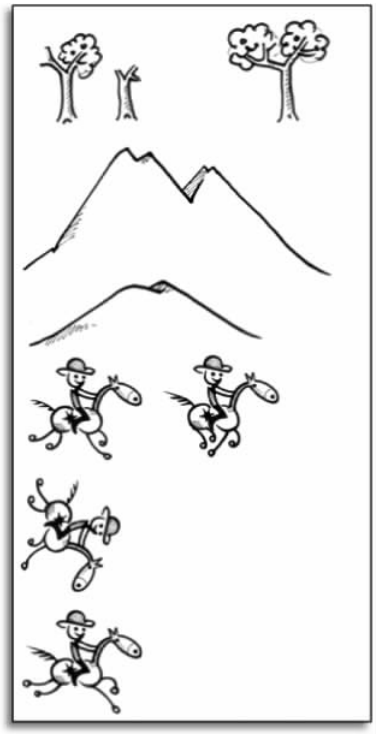

图 5-6 精灵表文件 SpirteSheet.png

plist 是属性列表文件，它是一种 XML 文件。SpriteSheet.plist 文件代码如下：

```
<?xml version = "1.0" encoding = "UTF - 8"?>
<! DOCTYPE plist PUBLIC " - //Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList - 1.0.dtd">
< plist version = "1.0">
    < dict >
        < key > frames </key >
        < dict >                                                    ①
            < key > hero1.png </key >                               ②
            < dict >
```

```
                <key>frame</key>
                <string>{{2,1706},{391,327}}</string>                    ③
                <key>offset</key>
                <string>{6,0}</string>
                <key>rotated</key>
                <false/>
                <key>sourceColorRect</key>
                <string>{{17,0},{391,327}}</string>
                <key>sourceSize</key>
                <string>{413,327}</string>                               ④
            </dict>
            ...
            <key>mountain1.png</key>
            <dict>
                <key>frame</key>
                <string>{{2,391},{934,388}}</string>
                <key>offset</key>
                <string>{0, -8}</string>
                <key>rotated</key>
                <false/>
                <key>sourceColorRect</key>
                <string>{{0,16},{934,388}}</string>
                <key>sourceSize</key>
                <string>{934,404}</string>
            </dict>
            ...
        </dict>
        <key>metadata</key>
        <dict>
            <key>format</key>
            <integer>2</integer>
            <key>realTextureFileName</key>
            <string>SpirteSheet.png</string>
            <key>size</key>
            <string>{1024,2048}</string>
            <key> smartupdate </key><string>$TexturePacker:SmartUpdate:5f186491d3aea289c5-
0ba9b77716547f:abc353d00773c0ca19d20b55fb028270:755b0266068b8a3b8dd250a2d186c02b$</string>
            <key>textureFileName</key>
            <string>SpirteSheet.png</string>
        </dict>
    </dict>
</plist>
```

上述代码是 plist 文件,其中代码第①～④行描述了一个精灵帧(小的精灵图片)位置。代码第②行是精灵帧的名字,一般情况下它的命名与原始的精灵图片名相同;代码第③行描述了精灵帧的位置和大小,{2,1706}是精灵帧的位置,{391,327}是精灵帧的大小。由于不需要自己编写 plist 文件,就不再介绍其他的属性了。

提示 关于工具 Zwoptex 和 TexturePacker 等纹理图集工具的使用,请参考本系列图书的工具卷(《Cocos2d-x 实战:工具卷》(第 2 版))。

使用精灵表文件最简单的方式是使用 Sprite 的 create(const std::string &filename, const Rect &rect)函数,其中创建矩形 Rect 对象可以参考坐标文件中代码第③行的{{2,1706},{391,327}}数据。使用 create 代码如下:

```
auto mountain1 = Sprite::create("SpirteSheet.png",Rect(2,391, 934, 388));
mountain1->setAnchorPoint(Vec2::ZERO);
mountain1->setPosition(Vec2(-200,80));
mountain1->addChild(mountain1,0);
```

在创建纹理 Texture2D 对象时,也可以使用精灵表文件。代码如下:

```
Texture2D* cache = Director::getInstance()->getTextureCache()->addImage("SpirteSheet.
png");
auto hero1 = Sprite::create();
hero1->setTexture(cache);
hero1->setTextureRect(Rect(2,1706,391,327));                                    ①
hero1->setPosition(Vec2(800,200));
this->addChild(hero1,0);
```

上述代码第①行中的 setTextureRect 函数,使用坐标文件中描述的数据。

5.2.2 使用精灵帧缓存

精灵帧缓存是缓存的一种。缓存有如下几种:

(1) 纹理缓存(TextureCache):使用纹理缓存可以创建纹理对象,在上一节已经用到了。

(2) 精灵帧缓存(SpriteFrameCache):能够从精灵表中创建精灵帧缓存,然后再从精灵帧缓存中获得精灵对象,反复使用精灵对象时,使用精灵帧缓存可以节省内存消耗。

(3) 动画缓存(AnimationCache):动画缓存主要用于精灵动画,精灵动画中的每一帧是从动画缓存中获取的。

这一节主要介绍精灵帧缓存(SpriteFrameCache),要使用精灵帧缓存涉及的类有 SpriteFrame 和 SpriteFrameCache。使用 SpriteFrameCache 创建精灵对象的主要代码如下:

```
SpriteFrameCache::getInstance()->addSpriteFramesWithFile("SpirteSheet.plist");    ①
auto mountain1 = Sprite::createWithSpriteFrameName("mountain1.png");              ②
```

上述代码第①行是通过 SpriteFrameCache 创建精灵帧缓存对象,它是采用单例设计模式进行设计的,getInstance()函数可以获得 SpriteFrameCache 单一实例,addSpriteFramesWithFile 函数是将精灵帧添加到缓存中,其中 SpriteSheet.plist 是坐标文件。可以多次调用 addSpriteFramesWithFile 函数添加更多的精灵帧到缓存中。

代码第②行 Sprite::createWithSpriteFrameName("mountain1.png")是通过 Sprite 的 createWithSpriteFrameName 函数创建精灵对象,其中的参数 mountain1.png 是 SpriteSheet.plist 在坐标文件中定义的精灵帧名(见 SpriteSheet.plist 文件代码中的第②行)。

下面通过一个实例介绍精灵帧缓存使用,如图 5-7 所示,在游戏场景中有背景、山和英雄[①]三个精灵。

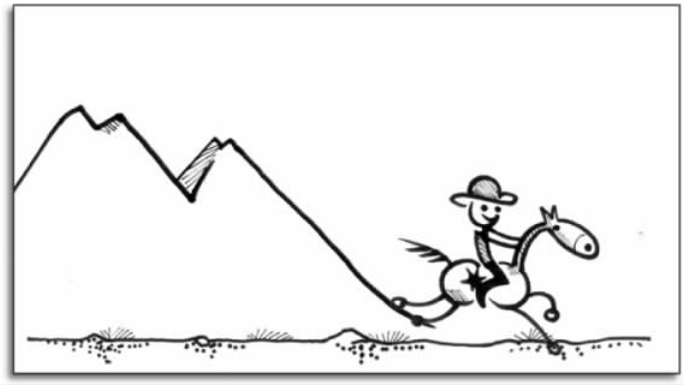

图 5-7 使用精灵帧缓存实例

在 HelloWorldScene.cpp 实现的 init 函数代码如下:

```
bool HelloWorld::init()
{
    if ( !Layer::init() )
    {
        return false;
    }

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    auto background = Sprite::create("background.png");                          ①
    background->setAnchorPoint(Vec2::ZERO);
    this->addChild(background,0);

    SpriteFrameCache *frameCache = SpriteFrameCache::getInstance();              ②
    frameCache->addSpriteFramesWithFile("SpirteSheet.plist");                    ③

    auto mountain1 = Sprite::createWithSpriteFrameName("mountain1.png");         ④
    mountain1->setAnchorPoint(Vec2::ZERO);
    mountain1->setPosition(Vec2(-200,80));
    this->addChild(mountain1,0);

    SpriteFrame *heroSpriteFrame = frameCache->getSpriteFrameByName("hero1.png"); ⑤
    Sprite *hero1 = Sprite::createWithSpriteFrame(heroSpriteFrame);              ⑥
    hero1->setPosition(Vec2(800,200));
    this->addChild(hero1,0);

    return true;
}
```

① 我们把玩家控制的精灵称为“英雄”,把计算机控制的反方精灵称为“敌人”。

上述代码第①行是创建一个背景精灵对象，这个背景精灵对象，并不是通过精灵缓存创建的，而是通过精灵文件直接创建的，事实上也完全可以将这个背景图片放到精灵表中。

代码第②行是获得精灵缓存对象。代码第③行是通过 addSpriteFramesWithFile 函数为精灵缓存添加精灵帧。在前面的介绍中，使用一条语句 SpriteFrameCache::getInstance()->addSpriteFramesWithFile("SpriteSheet.plist")替代代码第②和第③行两条语句。在这里分成两条语句是因为后面还要使用 frameCache 变量。

代码第④行是使用 Sprite 的 createWithSpriteFrameName 函数创建精灵对象，其中的参数是精灵帧的名字。

代码第⑤～⑥行是使用精灵缓存创建精灵对象的另外一种函数，其中代码第⑤行是使用精灵缓存对象 frameCache 的 getSpriteFrameByName 函数创建 SpriteFrame 对象，SpriteFrame 对象就是“精灵帧”对象，事实上在精灵缓存中存放的都是这种类型的对象。代码第⑥行是通过精灵帧对象创建。代码第⑤和⑥行使用精灵缓存方式主要应用于精灵动画时，相关的知识将在精灵动画部分介绍。

精灵缓存不再使用后要移除相关精灵帧，否则如果再有相同名称的精灵帧时，就会出现一些奇怪的现象。移除精灵帧的缓存函数如下：

(1) void removeSpriteFrameByName(const std::string & name)：指定具体的精灵帧名移除。

(2) void removeSpriteFrames()：指定移除精灵缓存。

(3) void removeSpriteFramesFromFile(const std::string &plist)：指定具体的坐标文件移除精灵帧。

(4) void removeUnusedSpriteFrames()：移除没有使用的精灵帧。

如果为了防止该场景中的精灵缓存对下一个场景产生影响，可以在当前场景所在层的 onExit 函数中调用这些函数。相关代码如下：

```
void HelloWorld::onExit()
{
    Layer::onExit();
    SpriteFrameCache::getInstance()->removeSpriteFrames();
}
```

onExit()函数是层退出时回调的函数，与 init 函数类似都属于层的生命周期中的函数。要在 h 文件中定义，在 cpp 文件中声明。HelloWorld.h 文件的相关代码如下：

```
#ifndef __HELLOWORLD_SCENE_H__
#define __HELLOWORLD_SCENE_H__

#include "cocos2d.h"

class HelloWorld : public cocos2d::Layer
{
  public:
    ...
```

```
    virtual bool init();
    //退出 Layer 回调函数
    virtual void onExit();
    …
    CREATE_FUNC(HelloWorld);
};

#endif // __HELLOWORLD_SCENE_H__
```

当然,精灵缓存清除工作也可以放到下一个场景创建时,也就是下一个场景所在层的init 函数中实现。相关代码如下:

```
bool HelloWorld::init()
{
    …
    SpriteFrameCache::getInstance()->removeSpriteFrames();
    SpriteFrameCache::getInstance()->addSpriteFramesWithFile("SpirteSheet.plist");
    …
}
```

本章小结

通过对本章的学习,了解 Cocos2d-x 中精灵的相关知识和如何创建精灵对象。此外,本章还介绍了精灵的性能优化,优化方式包括使用精灵表和使用精灵帧缓存。