TURING 图灵程序设计丛书

Apress®

# Flash ActionScript 3.0 动画高级教程

[美] Keith Peters 著
苏金国 荆涛 等译

跟随Flash大师进入ActionScript开发辉煌殿堂
丰富的经典示例和专家技巧
涵盖Flash 10最新特性

人民邮电出版社
POSTS & TELECOM PRESS

# Flash ActionScript 3.0

TURING 图灵程序设计丛书

# Flash ActionScript 3.0 动画高级教程

[美] Keith Peters 著
苏金国 荆涛 等译

人民邮电出版社
北 京

**图书在版编目（CIP）数据**

Flash ActionScript 3.0 动画高级教程 /（美）彼得斯（Peters, K.）著；苏金国等译．—北京：人民邮电出版社，2010.1
（图灵程序设计丛书）
书名原文：AdvancED ActionScript 3.0 Animation
ISBN 978-7-115-21625-0

Ⅰ. ① F… Ⅱ. ①彼…②苏… Ⅲ. ①动画－设计－图形软件，Flash ActionScript 3.0－教材 Ⅳ. ① TP391.41

中国版本图书馆CIP数据核字（2009）第193958号

## 内 容 提 要

本书是介绍 Flash 10 ActionScript 动画高级技术的经典之作，是作者在这一领域中多年实践经验的结晶。书中不仅涵盖了 3D、最新绘图 API 以及 Pixel Bender 等 Flash 10 ActionScript 特性，深入介绍了碰撞检测、转向、寻路等 Flash 游戏开发技术，还通过实例具体讲解了等角投影和数值积分的基本理论和应用。

本书内容紧扣实际应用，适合各层次 Flash 开发人员阅读。

图灵程序设计丛书

**Flash ActionScript 3.0 动画高级教程**

◆ 著 [美] Keith Peters
译 苏金国 荆 涛 等
责任编辑 傅志红
执行编辑 罗 婧

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子函件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000 1/16
印张：24
字数：573千字 2010年1月第1版
印数：1－3 000册 2010年1月北京第1次印刷

著作权合同登记号 图字：01-2009-5723号

ISBN 978-7-115-21625-0

定价：65.00元

**读者服务热线：(010)51095186 印装质量热线：(010)67129223**
**反盗版热线：(010)67171154**

# 版权声明

献给亲爱的Miranda和Kristine，再次感谢她们的耐心与支持。

# 译 者 序

这本书是鼎鼎大名的Keith Peters继《Flash ActionScript 3.0动画教程》之后的又一力作，相比之下，它将带领我们从“基础”进阶到“高级”领域。

那么，我们从中能得到些什么？也许并没有高深的理论，但这里的技术绝对贴近实战，如果充分领会并巧妙应用这些技术，定能让你的作品标新立异，也会让别人对你刮目相看。书中各个主题相对独立，又彼此关联，你可以有目的地对感兴趣的章节重点研究，也可以跟随作者的思路通读全文。作者不仅解决了开发游戏中最关心的碰撞检测、转向和寻路问题，还通过生动的实例让我们了解了等角投影和数值积分的基本理论和应用途径，特别谈到了将摄像头和麦克风纳入输入手段的简捷方法，另外补充了Flash 10中的3D特性和绘图API，最后帮助我们了解了Pixel Bender和各种补间引擎的作用和用法。

阅读这本书，肯定会让你跃跃欲试，迫不及待地想要亲手体验这些高级技术带来的绝妙效果，这也正是本书要达到的目的。技术如果只是纸上谈兵、束之高阁，那它就是死的，只有真正用于实践并不断扩展演进，才能算是名副其实的实用技术。特别值得一提的是，英文书名中AdvancED一词中的“ED”很值得玩味，也许是为了强调“高级”，也许是要告诉你“做过才知道”，各种理解见仁见智，不过可以肯定的是，只有真正读过、做过，才会有最大的收获。

在此深深地感谢我们的家人和朋友。在翻译过程中，他们给予了我们莫大的关心、支持和帮助。

全书主要由苏金国、荆涛翻译，并得到高强、王小振、刘鑫、范松峰、王恒、牛亚峰、刘亮、刘跃邦等人的帮助，最后由苏金国、荆涛统稿。

由于时间仓促，且译者的水平有限，译文中难免会出现一些错误，请读者批评指正。

# 致　谢

本书里只有很少一部分（如果有的话）是我自己构想出来的。要感谢成百上千的程序员、开发人员、科学家、数学家和物理学家们，是他们的努力研究和探索，精心编程和转换，才使得千百万人能够受益于他们的工作成果。

## 排版约定

为了让本书尽可能简明易读，本书将采用以下排版约定。

- 重要的词或概念在第一次出现时一般会用黑体突出显示。
- 代码用Letter Gothic字体表示。
- 新的代码或有修改的代码通常用粗体Letter Gothic字体表示。
- 伪代码和变量输入采用斜体Letter Gothic字体。

# 目　录

# 第1章 高级碰撞检测

碰撞检测可谓一门数学、艺术或科学，或者就是一般意义上的猜测推断，用来确定某个对象是否与另一个对象发生碰撞。这听起来很简单，不过当你面对的是只存在于计算机内存中并由一组不同属性表示的对象时，复杂性便会接踵而至。

碰撞检测的基本方法在《Flash ActionScript 3.0 动画教程》[①]一书中做了详细说明。本章将讨论该书中未涉及的一种碰撞检测方法，并介绍一种处理大量对象互相碰撞的策略。

需要指出，主题是碰撞检测，因此不会深入研究在检测到碰撞之后要做的事情。如果你在开发一个游戏，可能希望发生碰撞的对象爆炸、改变颜色，或者只是简单地消失。处理碰撞结果还有一种相当复杂的方法，这个方法在《Flash ActionScript 3.0 动画教程》的11.3节中做了介绍。不过，最终还是要由你（根据所构建应用或游戏的规范）来确定检测到碰撞时该如何做出响应。

## 1.1 不规则形状对象的碰撞测试

《Flash ActionScript 3.0 动画教程》中介绍了几个碰撞检测的基本方法，包括内置的hitTestObject和hitTestPoint方法，以及基于距离的碰撞检测。以上各种方法分别适用于不同形状的参与碰撞检测的对象。hitTestObject方法非常适合检测两个矩形对象之间的碰撞，但是对于其他形状往往会生成假警报。hitTestPoint方法适用于以下两种情况：查看鼠标是否位于一个特定对象之上，或者一个很小的点状对象是否与任何其他形状的对象发生碰撞，但是这个方法对于检测两个较大的对象则束手无策。基于距离的碰撞检测非常适于处理圆形对象，不过对于其他形状的对象常常会产生碰撞漏检。

Flash中实现碰撞检测的理想方法（相当于这个领域的“圣杯”）是对两个不规则形状的对象进行相互测试，从而准确地知道它们是否接触。从Flash 8开始，已经有一种方法可以通过BitmapData类做到这一点（不过《Flash ActionScript 3.0 动画教程》中没有讨论这种方法）。实际上，这种方法有一个很贴切的名字：hitTest。

首先，需要对一些术语做一点说明。ActionScript包含一个BitmapData类，其中保存要显示的具体位图图像，另外还有一个Bitmap类，这是包含一个BitmapData的显示对象，并允许将这个

① 此书已由人民邮电出版社于2008年4月出版。——编者注

BitmapData添加到显示列表中。在后面的介绍中，如果我特别指出其中某一个类或者某个类的实例，会使用首字母大写的类名(BitmapData或Bitmap)。不过，通常情况下，我可能会用**位图**(bitmap)一词不太正式地指一个位图图像。不要把它与Bitmap类相混淆。

BitmapData.hitTest会比较两个BitmapData对象，指出它们的像素是否发生重叠。重申一次，这听起来很容易，但是如果再多加考虑就会发现其中存在一些复杂性。位图是像素的矩形网格，所以如果只是采用最简单的形式，这个方法不会比作用于显示对象的hitTestObject方法更复杂(当然同样也不会有太大用处)。只有当使用了绘制有一个形状的透明位图时这个方法才真正有用。

创建一个BitmapData对象时，要在构造函数中明确指定它是否支持透明性：

```
new BitmapData(width, height, transparent, color);
```

第3个参数是一个布尔值(true/false)，这个参数将设置透明性选项。如果将其设置为false，位图就是完全不透明的。最初这呈现为一个用指定背景颜色填充的矩形。可以使用不同的BitmapData方法来改变位图中的任何像素，不过它们总是完全不透明的，并且会覆盖该BitmapData之下的所有内容。各像素的颜色值是一个形式为0xRRGGBB的24位数字。这是一个6位的十六进制数，其中第一对数指定了红色通道的值，取值范围是00（0）～FF（255），第二对设置了绿色通道，第三对设置蓝色通道。例如，0xFFFFFF是白色，0xFF0000是红色，0xFF9900是橙色。要设置和获取单个像素的值，可以使用setPixel和getPixel方法，它们都使用24位颜色值。

不过，在BitmapData类中将透明性选项指定为true时，每个像素还会支持一个*α*通道，相应地会使用一个格式为0xAARGGBB的32位数。在这里，前两位十六进制数表示一个像素的透明度，00为完全透明，FF则是完全不透明。在一个透明的BitmapData中，需要使用setPixel32与getPixel32来设置和读取单个像素的颜色。这些方法都取32位数作为参数。需要指出，如果向其中某个方法传入一个24位数，*α*通道将计算为0，也就是完全透明。

为了了解二者的具体差别，下面分别创建一个例子。可以在一个Flex Builder 3或4版本的ActionScript工程中使用下面的类作为主类，或者在Flash CS3或CS4中作为文档类。这个BitmapCompare类可以从本书网站www.friendsofed.com下载[①]。

```
package
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.geom.Rectangle;

    public class BitmapCompare extends Sprite
    {
        public function BitmapCompare()
        {
            stage.align = StageAlign.TOP_LEFT;
            stage.scaleMode = StageScaleMode.NO_SCALE;
```

① 本书代码也可以从图灵网站www.turingbook.com本书网页免费注册下载。——编者注

```
            // draw a bunch of random lines
            graphics.lineStyle(0);
            for(var i:int = 0; i < 100; i++)
            {
                graphics.lineTo(Math.random() * 300, Math.random() * 400);
            }

            // create an opaque bitmap
            var bmpd1:BitmapData = new BitmapData(300, 200, false, 0xffffff);
            bmpd1.fillRect(new Rectangle(100, 50, 100, 100), 0xff0000);
            var bmp1:Bitmap = new Bitmap(bmpd1);
            addChild(bmp1);

            // create a transparent bitmap
            var bmpd2:BitmapData = new BitmapData(300, 200, true, 0x00ffffff);
            bmpd2.fillRect(new Rectangle(100, 50, 100, 100), 0xffff0000);
            var bmp2:Bitmap = new Bitmap(bmpd2);
            bmp2.y = 200;
            addChild(bmp2);
        }
    }
}
```

这段代码首先在场景上画一组随机的线条，以便区分场景和位图。然后创建两个位图，并在各位图的中央画上红色的方块。上面的位图是不透明的，会完全覆盖所有线条。下面的位图是透明的，所以只有红色方块覆盖了场景中的线条。其结果如图1-1所示。

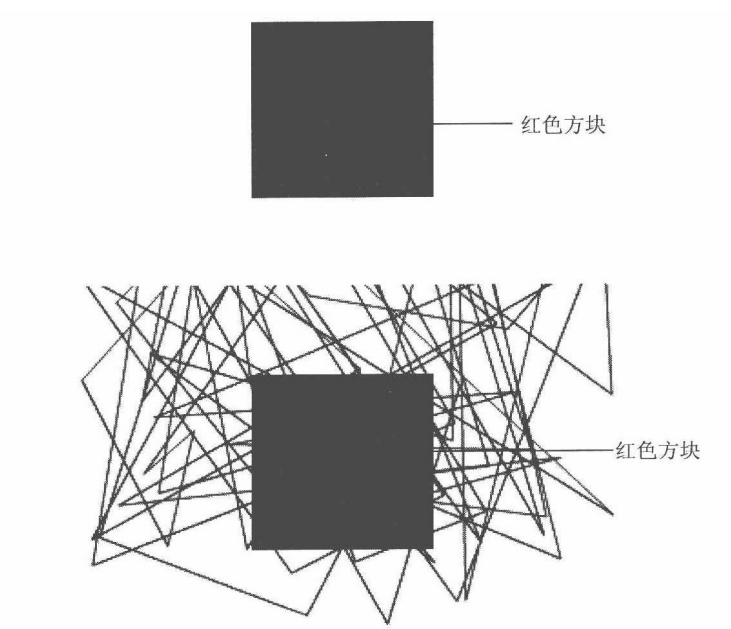


图1-1 上面是不透明位图，下面是透明位图

另外，对于透明位图还可以使用部分透明度。对以上代码示例中的第二个fillRect语句做如下修改：

```
bmpd2.fillRect(new Rectangle(100, 50, 100, 100), 0x80FF0000);
```

注意，这里使用了一个32位AARRGGBB颜色值来填充，$\alpha$值减半为0x80，即十进制数128。这样红色方块就会变得半透明，如图1-2所示。

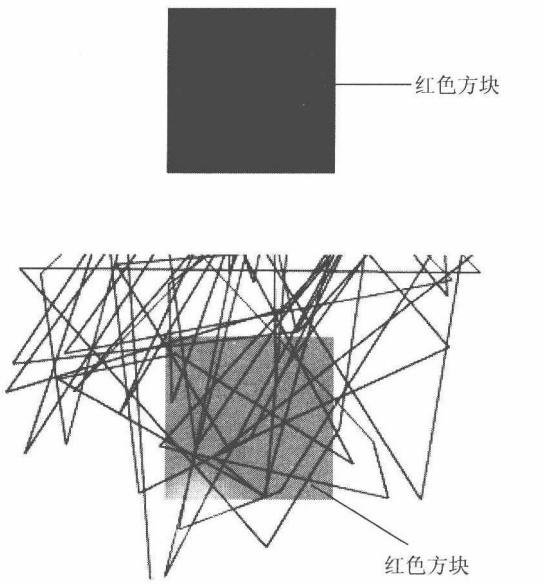


图1-2　一个半透明的方块

### 1.1.1　位图用于碰撞检测

现在来看如何用位图实现碰撞检测。首先需要一个适当的不规则形状来完成测试，五角星就很合适，可以建立一个五角星类以便以后重用。以下就是Star类，这个类也可以从本书网站下载得到。

```
package
{
    import flash.display.Sprite;

    public class Star extends Sprite
    {
        public function Star(radius:Number, color:uint = 0xFFFF00):void
        {
            graphics.lineStyle(0);
            graphics.moveTo(radius, 0);
            graphics.beginFill(color);
            // draw 10 lines
            for(var i:int = 1; i < 11; i++)
```

```
            {
                var radius2:Number = radius;
                if(i % 2 > 0)
                {
                 // alternate radius to make spikes every other line
                    radius2 = radius / 2;
                }
                var angle:Number = Math.PI * 2 / 10 * i;
                graphics.lineTo(Math.cos(angle) * radius2, Math.sin(angle) * radius2);
            }
        }
    }
}
```

这里只是画出角度不断递增而且半径交替变化的一系列线条，从而巧妙地构成了一个五角星。下面这个类可以完成碰撞测试。类似于本书中的大多数代码，这个类可以作为一个文档类在Flash CS3或CS4中使用，也可以作为主应用类在Flex Builder 3或4中使用，同样也可以从本书网站下载得到。

```
package
{
    import flash.display.Bitmap;
    import flash.display.BitmapData;
    import flash.display.Sprite;
    import flash.display.StageAlign;
    import flash.display.StageScaleMode;
    import flash.events.MouseEvent;
    import flash.filters.GlowFilter;
    import flash.geom.Matrix;
    import flash.geom.Point;

    public class BitmapCollision1 extends Sprite
    {
        private var bmpd1:BitmapData;
        private var bmp1:Bitmap;
        private var bmpd2:BitmapData;
        private var bmp2:Bitmap;

        public function BitmapCollision1()
        {
            stage.align = StageAlign.TOP_LEFT;
            stage.scaleMode = StageScaleMode.NO_SCALE;

            // make a star
            var star:Star = new Star(50);

            // make a fixed bitmap, draw the star into it
            bmpd1 = new BitmapData(100, 100, true, 0);
            bmpd1.draw(star, new Matrix(1, 0, 0, 1, 50, 50));
            bmp1 = new Bitmap(bmpd1);
            bmp1.x = 200;
            bmp1.y = 200;
            addChild(bmp1);
```

```
            // make a moveable bitmap, draw the star into it, too
            bmpd2 = new BitmapData(100, 100, true, 0);
            bmpd2.draw(star, new Matrix(1, 0, 0, 1, 50, 50));
            bmp2 = new Bitmap(bmpd2);
            addChild(bmp2);

            stage.addEventListener(MouseEvent.MOUSE_MOVE, onMouseMoving);
        }

        private function onMouseMoving(event:MouseEvent):void
        {
            // move bmp2 to the mouse position (centered).
            bmp2.x = mouseX - 50;
            bmp2.y = mouseY - 50;

            // the hit test itself.
            if(bmpd1.hitTest(new Point(bmp1.x, bmp1.y), 255, bmpd2,
                             new Point(bmp2.x, bmp2.y), 255))
            {
                bmp1.filters = [new GlowFilter()];
                bmp2.filters = [new GlowFilter()];
            }
            else
            {
                bmp1.filters = [];
                bmp2.filters = [];
            }
        }
    }
}
```

这里使用Star类创建了一个五角星，并分别绘制在两个位图中。我们使用了一个矩阵从而在绘制时将五角星在各个轴上分别偏移50个像素，因为五角星的对准点位于其中心，而位图的对准点位于左上角。通过偏移就可以看到整个五角星。

其中一个位图（bmp1）位于场景中的固定位置，另一个位图（bmp2）设置为跟随鼠标移动。这里的关键代码如下所示：

```
if(bmpd1.hitTest(new Point(bmp1.x, bmp1.y), 255, bmpd2,
                 new Point(bmp2.x, bmp2.y), 255))
```

正是这段代码确定了两个位图是否接触。BitmapData.hitTest方法的签名如下：

```
hitTest(firstPoint:Point,
        firstAlphaThreshold:uint,
        secondObject:Object,
        secondPoint:Point,
        secondAlphaThreshold:uint);
```

注意到，参数可分为两组：分别以first和second开头[①]。要分别为其提供一个点值（第一个

① 分别对应第一个对象和第二个对象。——译者注