基于JAVA的计算机图形学

〔日〕青野雅树 著
张文乐 译

科学出版社
www.sciencep.com

图字:01-2003-7987

内 容 简 介

本书是计算机图形学入门书，书中以具体实例介绍了计算机图形学的基本知识，主要内容涉及二维计算机图形学、三维计算机图形学及相关技术应用等，并收录了大量程序实例。书后附有专业术语注释，以帮助读者更好地学习、理解和掌握计算机图形学和Java语言编程的应用。

本书可供大学相关专业的师生，以及计算机图形、三维动画、多媒体研发等技术人员参考阅读。

图书在版编目(CIP)数据

基于Java的计算机图形学/(日)青野雅树著；张文乐译. —北京：科学出版社，2004
ISBN 7-03-012815-X

Ⅰ.基… Ⅱ.①青… ②张… Ⅲ.① 计算机图形学② Java语言-程序设计
Ⅳ.① TP391.41② TP312

中国版本图书馆CIP数据核字(2004)第005720号

责任编辑：王 炜 崔炳哲/责任制作：魏 谨
责任印制：白 羽/封面设计：李 祥

科学出版社 出版
北京东黄城根北街16号
邮政编码：100717
http://www.sciencep.com
源海印刷有限责任公司印刷

北京东方科龙图文有限公司 制作
http://www.okbook.com.cn
科学出版社发行 各地新华书店经销

*

2004年4月第 一 版 开本：16(787×1092)
2004年4月第一次印刷 印张：13 1/4
印数：1—5 000 字数：252 000

定 价：30.00元

(如有印装质量问题，我社负责调换(新欣))

Javaで学ぶ コンピュータグラフィックス
青野雅樹　オーム社 2003

著者简介

青野雅树

1957年　出生

1981年　东京大学理学部信息科学专业毕业

1994年　美国纽约州立大学伦塞勒(RENSSELAER)理工学院计算机科学专业毕业，获博士(Ph. D)学位

现　在　任职于日本IBM(株)

前　言

近年，随着计算机处理能力的飞速提高，超过 100GB 的大容量数据存储设备的普及，以及几百万像素数码相机的出现，多媒体数字化和网络共享化变得越来越必要。特别是基于二维和三维的计算机图形技术的动画技术（通常所说的 CG 动画）也日趋成熟。然而令人忧虑的是，随着各种数码生成工具软件的日益增多，基础技术慢慢地被淡忘了。为了开发具有独特创意的产品，掌握基础技术是非常重要的。

本书正是在这样的背景下，将计算机图形的基础技术利用具体的示例加以解说，使读者在一边阅读的同时便有“啊，原来如此”的感悟。本书适合于学生、工程师、CG 创作人员和开发人员，以及想在计算机或手机等液晶显示或 CRT 显示器的装置上进行图形开发的人员等。

本书首先避免将与计算机图形学有关的最新技术全部介绍给读者，同时也避免罗列出一堆繁杂的公式，在处理图像数据时也尽量避免使用特殊的处理方法。作为例子给出的程序语言，采用的是在任何平台上都能运行的 Java 语言。但本书不以解释 Java 语言和讲解程序的优化为目的，初学 Java 的人完全可以阅读本书。本书还介绍 Java 在因特网上以 Applet 或 Servlet 等形式的应用知识，并且尽量避免使用特殊的数学知识。

何谓 CG?

电视广告或节目中，电影或 IT 杂志中经常出现“CG”。这里的 CG 就是 Computer Graphics 的简称。特别在美国，像《玩具总动员》或《虫虫危机》这样的影片，完全用 CG 技术制作已经不稀奇了。

但是“计算机图形学（CG）”到底是什么呢？对此大概连那些利用最先进的 CG 技术进行开发的人也不一定能说够明白，比如将 CG 技术如何用于电影中，CG 怎样承担了一个庞大项目的一部分。开发 CG 技术、制作 CG 作品的人们对 CG 技术定义的不同，也说明了 CG 的有趣和博大。

本书的目的

本书是以 CG 初学者为对象的书籍，它并不是为了介绍最新的 CG 技术，而是尽量以具体的示例说明 CG 中重要的基础知识；另外，使本书成为在尽可能多的平台上都适用的一本接近网络时代的书。因此使用了 Web 浏览器也支持的 Applet 形式的 Java 语言。但是毕竟是为了说明 CG 的原理，所以书中不涉及程序的优化。本书并不是 Java 的说明书，所以，并没有说明 Java(Swing 等)的 GUI(Graphical User Interface)的制作方法。因此具有一定 Java 知识的读者阅读本书时可以跳过第 1 章。尽管简单，还是有必要对专用名词作解释，所以在本书的附录中有 Java 用语和 CG 用语集，为读者提供参考。另外，在本书各章末附上了该节中用于说明技术的 Java 源程序(基于 JDK1.1 的 Applet 程序)。

如果读者通过阅读本书，能够了解一些 CG 世界的基本知识，我将不胜欣喜。

鸣　谢

在本书的编写过程中，使用了 Ivan Sutherland 博士提供的画板照片，以及茅沼呼人先生的用于纹理绘制的镜像图形法的世界地图的数据。在此表示感谢。

青野雅树

目　录

第1章

Java的图形功能

本章主要介绍 Java 所提供的图形功能。在进入主题前，首先了解一下编程语言 Java。Sun Microsystems 公司在 1995 年发布了 Java，并于第二年发布了 **Java 开发工具包**(JDK1.0)。然后在 1997 年发布了 JDK1.1 版，1998 年升级到 JDK1.2(即 **Java2**)。在 Java2 中，有关图形方面的功能出现了质的飞跃。其中最值得一提的是 **Swing** 简单的用户界面工具包，它增加了大量的作图和图像处理的功能，使用户图形接口(GUI：Graphical User Interface)的开发非常接近平台无关性(WORA：Write Once Runs Anywhere)。另外，三维图形的 **Java3D**[1] 开发工具包也已经发布了。现在，许多系统平台已经发布了 JDK1.3(在图形功能方面，JDK1.3 与 JDK1.2 没有多大区别)。从 Java2 开始，图形功能得到很大的增强，但是因为考虑到一些系统平台的 Web 浏览器(例如网景公司的 Netscape Navigator 和微软的 Internet Explorer)并不支持 Java2，以及为了使用 Java2 功能必须在 HTML 文件里编写一些基于 Web 浏览器的特殊语句(规定)，本书基本上使用 JDK1.1 的功能编写程序[2]。

从发布 Java 之日起，Java 就是针对**因特网**进行设计的，因此随着因特网的迅速普及，Java 也在全世界推广开来。其中一个理由就是被称为 Applet(小应用程序)的 Java 程序，它一旦被 **Java 编译器**编译后，就成为不依赖于任何机器的中间语言的类文件，可以在 Web 浏览器上运行。这是由于 Web 浏览器中加入了 **Java 虚拟机**(Java VM：Java Virtual Machine)。最近，**个人数字助理**(PDA：Personal Digital Assistant)和移动电话也加入了 Java[3]。

使用 Java 进行编程时，首先要得到前面所述的 Java 开发工具包。几乎所有系统平台的 Java 开发工具包都可以免费得到。得到的方法可以参考本书的附录 1。

1.1 Java图形基础

除了单纯的计算程序在本地计算机上运行之外，Java 语言通常使用图 1.1 所示的窗口区域来表示程序执行的结果[4]。

1) 有关 Java 3D 的资料可以参考 http://java.sun.com/products/java-media/3D/index.html。

2) Java2 SDK (Software Development Kit)作为 Web 浏览器的插件也可以使用 Java2 的功能。

3) 但是，现阶段由于移动通信设备的 CPU 的性能和内存的限制，不加载 Java 虚拟机而通过使用 **J2ME**(Java2 Micro Edition)或 **KVM** 执行 Java 程序。

4) 本书不使用专用的开发工具开发 Java 的 GUI 功能，因为这样的工具是需要购买或是对计算机或操作系统有特殊要求的。

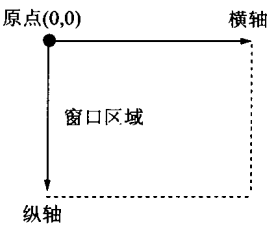

图 1.1 Java 的坐标系

Java 的窗口坐标左上为原点，横轴的右侧为正，纵轴的下方为正。定义窗口时只需要用整数给定其宽度和高度。由于原点的坐标通常为(0,0)，所以原点不需要指定。为什么 Java 用这样的坐标系或窗口设定方法？根据大多数的图像数据的格式，数据从左上开始向右面存放，一行结束后移到下一行，从下一行开始向右面存放数据，不断循环直到图像的最后一行。如果从这个角度考虑，以横轴的右面为正，纵轴的下面为正来处理图像数据，比较容易[1]。另外，如果窗口的大小用整数表示，窗口内的数据就是**像素**(pixel)的集合，即横轴的像素＝横轴屏幕的分辨率，纵轴的像素＝纵轴屏幕的分辨率。这里，像素就是屏幕上的一个点。例如，800×600 的屏幕大小在显示器上定义时，也就意味着像素为 800×600。

1. Applet描画

下面我们谈谈 Java 的描画方法。基于包含 JDK1.1 基本功能的标准 Java 虚拟机，有关 Java 的窗口描画全部使用包含在**抽象窗口工具包**(AWT：Abstract Windowing Toolkit)中的**图形类**的方法。有关包、类、方法等 Java 术语请参考附录 2 的术语集。本书中，Java 程序中的 import java.awt.* 等语句表示使用(import)AWT 包。

下面是一个简单的描画示例。

程序 1.1 lineDrawApplet.java

```
// lineDrawApplet.java  ← // 表示注释
//简单的画线例子
import java.awt.*; //使用 java.awt 包
import java.applet.Applet; //使用 java.applet.Applet 类
public class lineDrawApplet extends Applet {//lineDrawApplet 类继承了 Applet 类
    public void paint(Graphics g){//定义描画用的 paint()方法
        Dimension d = getSize();//取得 Applet 大小
        g.drawLine(0,0,d.width,d.height);//从(0,0)到(d.width,d.height)描画
    }
}
```

程序 1.1 描画了一条直线，但实际执行的是 **paint()**方法中的 **drawline()**方法。在编辑器中将 lineDrawApplet.java 编写后，在**命令行提示符**[2]中输入 Java 的编译器(假设是 **javac**)命令。

1) 但是，对于 Windows 或 OS/2 的 BMP 图形，图像的数据是从左下角开始的。

2) 命令行提示符是 UNIX 操作系统及 WindowsNT/2000/XP 的术语，相当于 Windows95/98/Me 等的 MS-DOS 提示符。

```
>javac lineDrawApplet.java
```

这里“>”是提示符。提示符根据操作系统的不同有“%”,“#”等。编译通过后,生成 lineDrawApplet.class 文件。lineDrawApplet.class 文件名的后缀表示该文件是被称为**Java 字节码**的不依赖机器的中间语言的**类文件**。类文件是不可以直接执行的。Applet 的源文件经过编译后生成的类文件是用 **HTML(Hyper Text Markup Language)文件**的**<APPLET>标识**来执行的[1)]。窗口的大小通过对**<APPLET>标识**中的 WIDTH=窗口宽度,HEIGHT=窗口高度的设定来实现。**程序 1.2** 就是使用最少标识的标识 lineDrawApplet.html 文件[2)]。

程序 1.2　lineDrawApplet.html

```
<!
    画线的 Applet(lineDrawApplet.html)
->
<HTML>
<TITLE>画线的 Applet </TITLE>
<BODY BGCOLOR = "888888">
<CENTER>
<APPLET CODE = lineDrawApplet.class WIDTH = 300 HEIGHT = 200>
JavaApplet 不能正常运行。</APPLET>
<BR>
画线的 Applet
</CENTER>
</BODY>
</HTML>
```

这个 Applet 的执行从命令行输入。

```
>appletviewer lineDrawApplet.html
```

这里 appletviewer(**小应用程序浏览器**)是 JDK 自带的程序,在操作系统中也称为 Applet runner(Applet 执行器)。正常执行结果如图 1.2 所示,从窗口的左上角向右下角画一条线。图 1.3 是以典型的浏览器 Internet Explorer 为例的执行结果。

2. Applet的运行原理

这里简单说明 Applet 的运行原理。

1) Web 浏览器不支持标准的 Java 虚拟机,如果使用 Java2 插件,请参考 http://java.sun.com/products/plugin/1.3/docs/ja/index.docs.html。

2) “JavaApplet 不能正常运行”只在 Web 浏览器不能处理<APPLET>标识时显示。

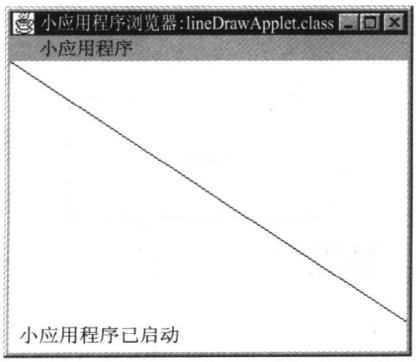

图 1.2 lineDrawApplet 在 appletviewer 中的执行结果

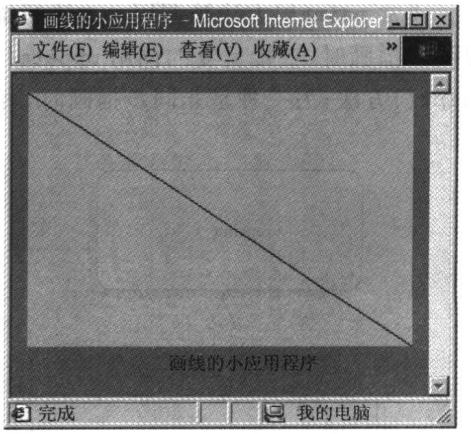

图 1.3 lineDrawApplet 在 Internet Explorer 中的执行结果

Applet 的运行:执行 Applet 时,Java 虚拟机首先将 Applet 加载到 Web 浏览器或 AppletViewer 中;此时,如果 Applet 中有 **init()**方法,就执行 init()方法。然后显示 Applet;如果 Applet 中有 **start()**方法,就执行 start()方法。同样,如果 Applet 中定义了 **paint()**方法,就执行 paint()方法。描画的窗口的大小由 HTML 文件的<APPLET>标识中的 WIDTH 和 HEIGHT 的值来决定。如果用户从当前浏览的 Web 页面跳转到其他页面或者关闭 Web 浏览器,则 **stop()**方法被执行,Applet 的执行被暂停,用 **destory()**方法永久停止 Applet 的执行。

以上就是简单 Applet 的运行。实际上根据 Web 浏览器的不同 Applet 的安装方法也有所不同。例如 Netscape Navigator[1],Applet 一旦被加载就一直运行到 Web 浏览器关闭,即使包含 Applet 的 HTML 文件被更新,Applet 也不会被初始化。而对于 Internet Explorer,即使曾经被加载过的 Applet,随着包含 Applet 的 HTML 文件被更新 Applet 也重新被初始化。

3. AWT包的描画原理

通常当被描画的窗口更新时(例如,该窗口被点击后产生在最顶层表示的**事件**时),**paint()**方法被调用。**paint()**方法是 **AWT** 包的 Component 类中的方法,并被同一个 Component 类的 **update()**方法调用,而 **update()**方法被同一个 Component 类的 **repaint()**方法调用。**repaint()**方法通常非同步执行 **update()**方法。**repaint()**方法被窗口的更新事件发生时执行。这三个方法的关系如图 1.4 所示。

1) 基于 Netscape Navigator 6.0 以前版本。

通常 **paint()**方法被调用一次进行描画。但是要反复描画时，有两种方法：一种是想要同步描画时，直接反复的调用 **update()**方法或(没有用 AWT 等做成的标题或按钮等) **paint()**方法；另一种是非同步描画时，直接反复调用 **repaint()**方法[1]。

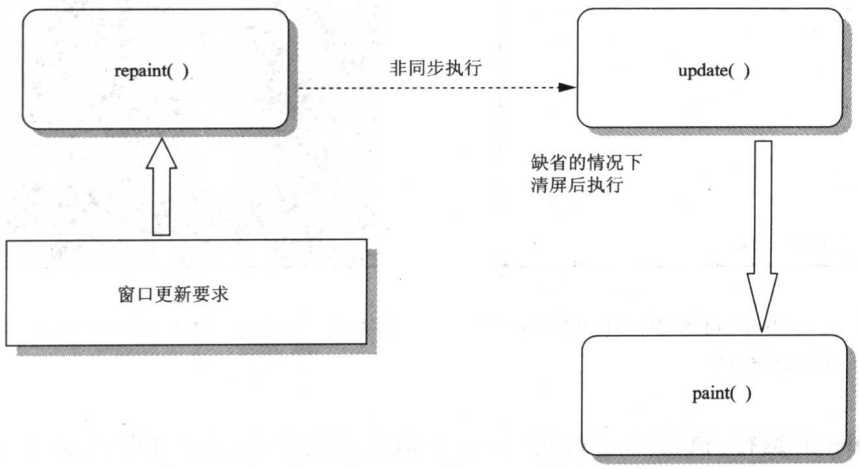

图 1.4　repaint()，update()，paint()的关系

1.2 使用offscreen buffer的双重缓存

上一节中简单说明了 Java(JDK1.1)所提供的图形功能，基本上用 Component 类中的 **paint()**方法进行描画。对于静止图像的窗口不需要更新，**paint()**方法只要一次调用。本节以 Java 的基本图形功能为前提，介绍了用 offscreen buffer[2] 来实现二维动画的原理。

1. 不使用offscreen buffer

为了体验为什么要使用 offscreen buffer，用**程序 1.3**，一个不使用 offscreen buffer 的二维动画，几个随机方向弹射的小球的程序来说明。为了产生**随机方向**使用了**随机数类**方法 **Math.random()**[3]。该方法随机产生 0.0 到 1.0 之间的实数。

1) 在 **paint()**方法中调用 **repaint()**方法也能反复描画。

2) 相对用户可见的画面，系统内部保持的画面通常称为 offscreen buffer。从 Java2 开始 offscreen buffer 称为缺省设置，用户不需要去特别注意，但从 JDK1.1 的话，就要如本节所示自己花点功夫。

3) 随机数类方法 **Math.random()**对游戏程序非常有用。用于生成随机数。

为了实现动画请注意 **paint()**方法中的 **repaint()**方法的调用。这里只是作为不使用 offscreen buffer 的例子，程序的其他细节可以忽略。

如果将程序 1.5 所示的 HTML 文件在 AppletViewer 或合适的 Web 浏览器中执行，就会发生剧烈的抖动。抖动发生的原因有两个：一个是 Java 的 AWT 包中缺省安装的 **update()**方法，缺省的动作 **update()**方法首先将窗口清除[1]后调用 **paint()**方法。这个窗口清除动作是抖动的主要原因之一；另一个是填充多个图形时(这里是向随机方向弹射的圆形小球)，每个图形的一边进行**剪切处理**(参见 2.1 节第 6 项)一边在 Applet 的窗口执行，产生了**描画时间差**。

2. 使用offscreen buffer

首先为了消除第一个原因"窗口清除"将 paint()方法重新改写。只要简单的重载 **update()**方法就可以取消窗口清除。

```
public void update(Graphics g){
    paint(g);
}
```

但如果将这个方法加入程序 1.3 中，第二个原因时间差没有解决，小球的活动轨迹残留在 Applet 描画的窗口上，没有得到预想的结果。这里 offscreen buffer 的概念就出现了。**把现在正在描画的窗口比作屏幕，offscreen buffer 的意义就是描画不直接表示在前台屏幕上(称为前台窗口)，而是表示在后台屏幕上**。图形描画的命令发出后先在此后台屏幕上描画，等全部的描画结束后才将 offscreen buffer 显示到前台屏幕上，这样便消除了描画的时间差。幸而 **JDK1.1 版的 Java 可以用 Component 类的 createImage()方法来实现 offscreen buffer**。将程序 1.3 改造一下，改变一下 **update()**方法并且引入 offscreen buffer，就成为了没有抖动的随机方向弹射的小球的**程序 1.4**。

3. offscreen buffer使用与否的区别

程序 1.5 是能将程序 1.3 和 1.4 加以比较的 HTML 文件。请用合适的 Web 浏览器比较一下。这样就可以实现使用 offscreen buffer 的没有抖动的动画。图 1.5 示出了 offscreen buffer 的原理。简单的说明一下，如图所示，首先在 offscreen buffer 中描画，等描画结束后将 offscreen buffer 的图像移动到前台屏幕上；然后再在 offscreen buffer 中描画循环往复就实现了没有抖动的动画。

---

1) 窗口的清除是由 Graphics 类中的 **clearRect()**方法实现的。

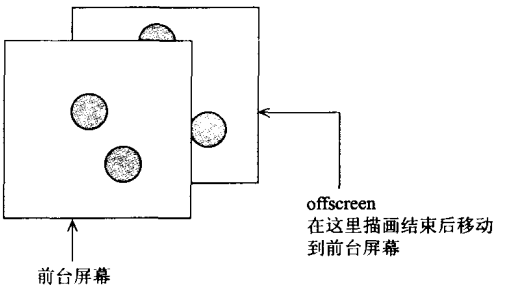

图 1.5　offscreen buffer 和前台屏幕

◇本节的程序集

程序 1.3　flickerApplet.java

```
// flickerApplet.java
// 简单的带闪烁的动画例子
import java.awt.*;//使用 java.awt 包
import java.applet.Applet;//使用 java.applet.Applet 类
public class flickerApplet extends Applet {
    private int w; // Applet 的宽度
    private int h; // Applet 的高度
    private int radius = 20; // 球的半径
    private float[] dx; // 球横向单位时间的移动量
    private float[] dy; // 球纵向单位时间的移动量
    // -1.0 <= dx, dy <= 1.0
    private int[] lastx, lasty; // 球的最新位置
    private int[] x,y; // 球的新位置
    private static int numBalls = 20;//球的总数
    final static float SPEED = 5.0f;//球的速度
    final static double EPSILON = 1.0E-5;//非常小的数

    public void init(){//Applet 开始时初始化方法
        Dimension d = getSize();//取得 Applet 大小
        w = d.width;// Applet 的宽度
        h = d.height;// Applet 的高度
        radius = Math.min(w,h)/10;//计算球的半径
        lastx = new int[numBalls];//分配球的最新横轴位置数据的内存空间
        lasty = new int[numBalls];//分配球的最新纵轴位置数据的内存空间
        dx = new float[numBalls];//分配球的横向单位时间的移动量的内存空间
        dy = new float[numBalls];//分配球的纵向单位时间的移动量的内存空间
        x = new int[numBalls];//分配球的横轴现在位置数据的内存空间
        y = new int[numBalls];//分配球的纵轴现在位置数据的内存空间
        for (int i=0 ; i < numBalls ; i++ ){ //球总数的循环
```