HZ BOOKS
华章教育

重点大学计算机教材

计算机图形学原理

张 康
德州大学达拉斯分校

Leen Ammeraal
Hogeschool Utrecht大学

王长波
华东师范大学

编著

Computer Graphics Principles

机械工业出版社
China Machine Press

重点大学计算机教材

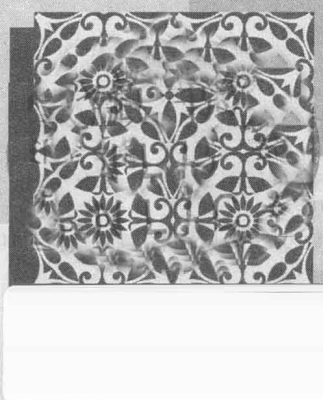
计算机图形学原理

张 康
德州大学达拉斯分校

Leen Ammeraal
Hogeschool Utrecht大学

王长波
华东师范大学

编著



Computer Graphics Principles

机械工业出版社
China Machine Press

本书系统地介绍了基本的计算机图形原理及算法，并给出其相关Java实现。本书从逻辑上分为三部分。第一部分为二维计算机图形，首先从基本概念入手，介绍怎样用逻辑坐标来画简单的图形和填色；为奠定基本数学基础，随后回顾了应用几何学用于图形学的基本概念，包括向量、多边形和图形变换方法；然后详细介绍了经典的图形生成算法和分形（Fractals）技术（包括Mandelbrot集和Julia集）以及色彩的基础知识。第二部分为三维计算机图形，深入地描述了三维透视法和线消隐、面消隐技术。第三部分介绍了Java 3D的编写原理和使用指南。

本书可以作为高校本科生或非计算机专业的研究生图形学课程教材，也可作为从事计算机图形学、游戏开发、动漫制作、手机应用软件、网络多媒体软件等开发工作的研究生、科研人员和企业开发人员的参考用书。

图书在版编目（CIP）数据

计算机图形学原理 / 张康等编著. —北京：机械工业出版社，2012.8
（重点大学计算机教材）

ISBN 978-7-111-39040-4

Ⅰ.计… Ⅱ.张… Ⅲ.计算机图形学-高等学校-教材 Ⅳ.TP391.41

中国版本图书馆CIP数据核字（2012）第146059号

机械工业出版社（北京市西城区百万庄大街22号 邮政编码 100037）
责任编辑：朱秀英
三河市杨庄长鸣印刷装订厂印刷
2012年9月第1版第1次印刷
185mm×260mm · 14.75印张
标准书号：ISBN 978-7-111-39040-4
定价：29.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991，88361066
购书热线：（010）68326294，88379649，68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

前　言

随着计算机软硬件技术的不断进步，计算机图形学技术也不断向多样化和人性化方向发展。以动漫、数字游戏为代表的数字娱乐产业的迅猛发展极大地推动了计算机图形学和相关学科的大力发展。各种新媒体技术的蓬勃发展也大大促进了计算机图形学在各领域的应用和推广。计算机图形学已成为计算机科学与技术、数字媒体、软件工程、数字设计等专业的核心专业课之一。

近年来，移动通信尤其是手机应用的飞速发展对于快速开发基于移动平台的应用提出了迫切的需求，网络媒体的发展也催生了网络数字媒体应用的蓬勃发展。Java编程语言的不断普及使得移动平台和网络媒体上的应用软件以指数级速度增长，并具有跨平台性、简单、便捷等优点。因此撰写基于Java语言的计算机图形学教材成为当务之急。本书正是在这样的形势下，基于作者多年的计算机图形学教学经验面向国内高校的教学需求编写的，以期成为与时俱进、理论与实践并重的教材。

本书系统地介绍了基本的计算机图形学原理及算法，并给出其相关Java实现。第一部分为二维计算机图形，首先从基本概念入手，介绍怎样用逻辑坐标来画简单的图形和填色；为奠定基本数学基础，随后回顾了应用几何学用于图形学的基本概念，包括向量、多边形和图形变换方法；然后详细介绍了经典的图形生成算法和分形（Fractals）技术（包括Mandelbrot集和Julia集）以及色彩的基础知识。第二部分为三维计算机图形，深入地描述了三维透视法和线消隐、面消隐技术。第三部分介绍了Java 3D的编写原理和使用指南。

全书使用众多通俗易懂且可即刻运行的Java程序来实现所学计算机图形学原理及算法；理论结合实际，着重培养学生的编程实现能力；图文并茂，交互演示，每章都有图例演示，提供相关程序运行结果截图和项目应用实例。本书同时附有一套经典的图形生成算法的演示软件（带源码），以进一步帮助初学者理解各算法的工作原理，也可供高级读者对其进行进一步完善。

本书可以作为高校本科生或非计算机专业的研究生计算机图形学课程的教材，也可作为从事计算机图形学、游戏开发、动漫制作、手机应用软件、网络多媒体软件等开发工作的研究生、科研人员和企业开发人员的参考用书。

本书在编写的过程中得到了机械工业出版社华章公司的大力支持和鼓励，在此表示衷心感谢！本书的编写也得到了崔锦、贾圆圆、叶鹏、肖昭等同学的帮助，在此一并表示感谢。

书中如有不妥或错误之处，恳请广大读者批评指正。

编者

2012年6月

教学建议

教学内容	教学要点及教学要求	课时安排（36）	
		本科计算机、软件专业课	非计算机、软件专业或研修课
第1章 图形学基本概念	了解计算机图形学的基础知识，如绘画点的离散特征和坐标系统 掌握最简单的图形生成技术	3	3
第2章 二维图形的数学基础	掌握矢量的点积、叉积等运算 掌握多边形的定向、面积、内部点的测试及其三角划分 掌握点与空间的关系判断	4	2 （根据数学基础略讲）
第3章 经典的图形算法	掌握直线和圆的Bresenham算法和直线的双步算法 掌握线段裁剪和多边形裁剪算法 掌握贝塞尔和B样条曲线的画法	6	4
第4章 分形	了解分形的主要思想和其应用范围 掌握科赫曲线的特征和生成方法 掌握串文法定义和生成各种曲线的方法 掌握Mandelbrot集和Julia集以及它们之间的关系	4	4
第5章 色彩、纹理和光照明	掌握图形学的颜色原理和常用的色彩模型 了解并能实现透明度方法 了解并能实现纹理映射 掌握Phone光照模型的原理	4	4
第6章 三维图形变换与透视	掌握矩阵运算基础 掌握二维及三维几何变换方法 掌握视图变换及透视变换的方法 掌握图形变换的程序实现方法	4	2 （根据数学基础略讲）
第7章 数据结构与实现方法	掌握三维图形数据结构的Java实现方法 掌握三维图形的描述语言并能熟练运用 掌握自动生成三维图形描述的技巧	4	4
第8章 线消隐与面消隐	了解线消隐和面消隐的基本原理 掌握线消隐算法及其实现 掌握画家算法和深度缓冲算法及其实现	4	2 （省略线消隐）
第9章 Java 3D介绍	了解Java 3D的基本框架 了解相应于本书其他章节概念的Java 3D实现原理和使用规则 会使用Java 3D实现简单图形	3	3

说明：

1）本教材主要是为计算机专业的本科“计算机图形学”课程编写的，也可用于非计算机专业的学生，如建筑、影视、数字媒体和动画类专业的学生。对于非计算机专业且无数学基础的学生，第2章的主要内容和第6章的数学内容均可跳过，而注重教材里实现各种图形学算法的Java程序的使用，并向艺术方面拓展分形、色彩、纹理、光照明等内容。

2）建议课堂授课时数36～48（包括习题课、课堂讨论等必要的课堂教学环节，实验课时12～18课时），部分高级内容可以略讲，不同学校可以根据各自的教学要求和计划学时数酌情对教材内容进行取舍。

3）本教材也可以作为有一定基础和研究兴趣的高年级本科生及硕士研究生的教学用书，还可作为在实际应用中有需求的科研和企业人员的参考书。书中所附的实现所有算法和图形学常用的功能的Java程序将是各类学生和应用人员必不可少的资源。

实验教学建议：

1）实验一：Java基本编程练习，包括绘制简单图形和用户界面。

2）实验二：基本图元算法的实现，包括Bresenham直线算法、双步画线算法、线段和多边形裁剪等算法。

3）实验三：曲线绘制算法，包括贝塞尔曲线和B样条曲线。

4）实验四：分形曲线中串文法的灵活应用，生成更多新颖、有趣的曲线。

5）实验五：拓展Mandelbrot集和Julia集，产生变化多样的交互图形，或将两集相结合。

6）实验六：颜色编码和纹理的应用，以及光照的编程实现。

7）实验七：自动生成复杂3D图形的数据描述（如图7-7中的螺旋形阶梯）。

8）实验八：将所学经典算法做成演示系统，可参考或重用本书提供的开源演示软件代码。

9）实验九：分3～4个阶段实现俄罗斯方块游戏，最后作为课题项目扩充成3D俄罗斯方块游戏。

目　录

第 1 章　图形学基本概念

本书是一本讲授计算机图形学基础知识的教材。我们将主要讨论图形学的数学基础以及如何用Java编制和实现各种图形算法。本章将讲述图形显示中的离散原理及其在低分辨率显示下的视觉效果，然后从Java画线入手介绍Java的使用方法。本章的重点是引入设备坐标和逻辑坐标的概念、逻辑坐标的必要性，以及设备坐标和逻辑坐标之间的互相转换。

1.1　离散点现象

在数学定义中，线段是连续的而且没有粗细之分。但在计算机图形学中，线段的定义则有所不同。通常，显示画面上最小显示单元称为**像素**(pixels，由picture elements简化而来)。表达其位置的x和y的整数坐标直接对应于显示器的像素，所以叫**设备坐标**(device coordinates)。线段由离散的像素组成，这是因为计算机显示屏上只有有限个像素，所以不存在连续线段，只能用有限个像素来近似表示。显示屏的分辨率越高，这样的近似就越精确。不过，用离散点来近似表示线段可能会产生一些特殊情况，尤其是在低分辨率的显示屏上。如图1-1所示，假定我们需要将一个6×6像素的正方形*ABCD*的右下角一半的三角形*ABC*填充为黑色，而左上角一半的三角形*ACD*填充为灰色。显然，我们会产生一个疑问，对角线*AC*应该填为黑色还是灰色？如果将对角线算成黑色部分，则黑一半的三角形就会比灰一半的三角形大一圈。反之，将对角线算成灰色部分，则灰一半的三角形又会比黑一半的三角形大一圈。在这种情况下，不可能填充出两个相同大小的三角形。

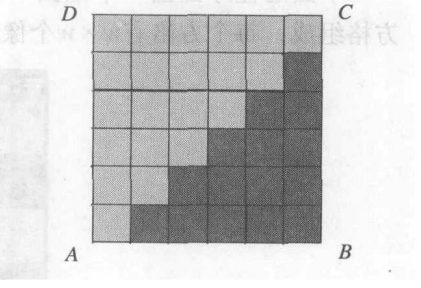


图1-1　离散像素填充造成的不精确性

类似的问题在下面的实例中所用到的Java提供的drawRect画矩形和fillRect 填矩形方法中再次出现。

```
g.drawRect(x, y, w, h);
```

上面的函数可以画一个以 (x, y) 和 $(x + w, y + h)$为两个顶角的矩形。drawRect前的g是通常用于paint方法里的graphics context。这里，drawRect方法中的w和h用来表达所画矩形的宽度和高度，而不是右下角的坐标。这同时也说明，该矩形宽为w + 1个像素、高为h + 1个像素。但下面的调用：

```
g.fillRect(x, y, w, h);
```

填出的新矩形的左上角为 (x, y) ，右下角为 $(x + w-1, y + h -1)$ 。该矩形宽为w个像素、高为h个像素，共有$w \times h$个像素，比g.drawRect(x, y, w, h)画出的空心矩形在长和宽上各少一个像素，所以要稍小一点。

那么用g.drawRect(x, y, w, h)画出最小可能的正方形应该是2×2个像素，语句如下：

```
g.drawRect(x, y, 1, 1);
```

如想在屏幕上只画一个像素点，用drawRect是无法做到的。如果在对drawRect的调用中，把第三和第四个参数设为零，则该语句将画不出任何东西。有意思的是，Java语言没有专门画一个像素的方法。要想只画一个像素，我们可以调用：

```
g.fillRect(x, y, 1, 1);
```

它比g.drawRect(*x*, *y*, 1, 1)所画的2×2个像素的正方形正好小一圈，所以只有一个像素。另一种画单个像素的方法是调用：

```
g.drawLine(x, y, x, y);
```

即画一个线段，其两个端点为同一点。而如下的调用：

```
g.drawLine(xA, y, xB, y);
```

画的是一条由|*xB*−*xA*| + 1个像素组成的水平线。

下面我们将用上面的语句组织完整的Java程序。[⊖]如果读者对Java编程不熟悉，可以先借助介绍Java编程的书籍学习如何编写Java程序。

下面的程序画出一个如图1-2所示的国际象棋棋盘。这个棋盘由8×8个深灰和浅灰相间的方格组成，每个方格有*w*×*w*个像素（程序中将*w*定为50）。

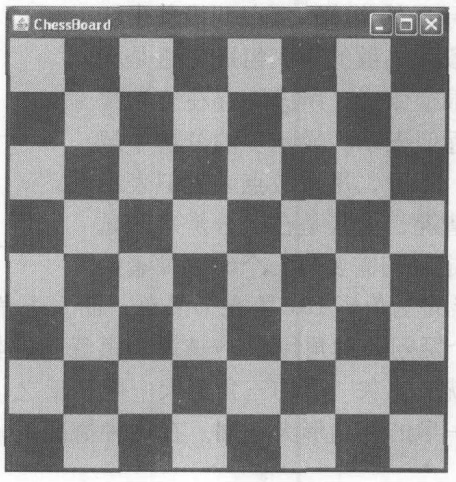


图1-2 国际象棋棋盘

```
// ChessBoard.java: Draw a 8x8 chess board, filled with light and dark grays.

import java.awt.*;
import java.awt.event.*;

public class ChessBoard extends Frame
{  public static void main(String[] args){new ChessBoard();}

   ChessBoard()
   {  super("ChessBoard");
      addWindowListener(new WindowAdapter()
```

⊖ 正如前言所述，本书中的所有程序都可以从华章网站上下载，网址为http://www.hzbook.com。读者可从http://java.sun.com/上下载并安装Java Development Kit。

```
        {public void windowClosing(WindowEvent e){System.exit(0);}});
      setSize(408, 434); //exact size to fit chess board
      add("Center", new CvChessBoard());
      show();
   }
}

class CvChessBoard extends Canvas
{  public void paint(Graphics g)
   {  Dimension d = getSize();
      int w = 50; //number of pixels in a square = w*w
      for (int i=0; i<8; i++)
        for (int j=0; j<8; j++)
        {  g.setColor((i + 8 - j) % 2 == 0 ?
              Color.lightGray : Color.darkGray);
           g.fillRect(i * w, j * w, w, w);
        }
   }
}
```

如果我们只画方格的边框而不填色，相应地画成深灰和浅灰，只需将上面程序中的fillRect调用替换为下面的调用即可：

```
g.drawRect(x + i * w, y + j * w, w - 1, w - 1);
```

请注意，根据前面所看到的drawRect和fillRect的区别，该调用的最后两个参数应该是$w-1$而不是w。下面的画线问题更好地说明了离散点现象。

1.2 用Java2D画线

在计算机屏幕上画线段的最简便的方法就是根据所输入的线段的两个端点来画。Java里的Graphics 中的 drawLine 方法是专门用来画线的，但它要求这两个端点为整数。我们假定这两个端点为P和Q，线段PQ就可用下面的方法来画：

```
g.drawLine(xP, yP, xQ, yQ);
```

该语句和下面的语句画出的线段完全一样：

```
g.drawLine(xQ, yQ, xP, yP);
```

也就是说，线段两个端点的顺序对画出的线段不产生任何影响。

图1-3所示的canvas宽为11个像素，高为5个像素。该图还显示了从点(0, 0) 到点(10, 4)的11个像素近似成为一条直线。

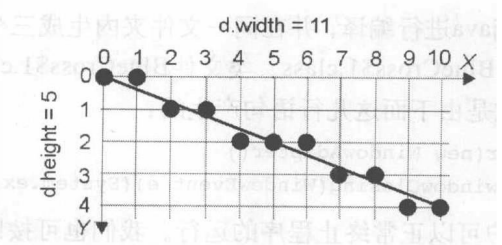


图1-3 一个11×5 canvas中的像素坐标(此处，$maxX=10$，$maxY=4$)

图1-3中在近似直线上的黑点代表那些设成前景颜色的像素，而其他像素为背景颜色。前景颜色假定为黑色，背景颜色假定为白色。下面的语句将会把这11个点画成黑色：

```
g.drawLine(0, 0, 10, 4);
```

下面是一段完整的程序，用于在Java画板（canvas）上画出最大可能的蓝色叉形：

```
// BlueCross.java: Draw the largest possible blue cross.
import java.awt.*;
import java.awt.event.*;

public class BlueCross extends Frame
{  public static void main(String[] args){new BlueCross();}

   BlueCross()
   {  super("BlueCross");
      addWindowListener(new WindowAdapter()
         {public void windowClosing(WindowEvent e){System.exit(0);}});
      setSize(360, 150);
      add("Center", new CvBlueCross());
      show();
   }
}

class CvBlueCross extends Canvas
{  public void paint(Graphics g)
   {  Dimension d = getSize();
      int maxX = d.width - 1, maxY = d.height - 1;
      g.drawString("d.width  = " + d.width,  130, 90);
      g.drawString("d.height = " + d.height, 130, 110);
      g.setColor(Color.blue);
      g.drawLine(0, 0, maxX, maxY);
      g.drawLine(0, maxY, maxX, 0);
   }
}
```

该程序含两个类：

- BlueCross：用于frame的类，同时也可以用来关闭此应用程序。
- CvBlueCross：用于canvas并显示图形输出的类。

使用下面的命令：

```
javac BlueCross.java
```

将程序BlueCross.java进行编译，并在同一文件夹内生成三个新的类文件：BlueCross.class、CvBlueCross.class 和 BlueCross$1.class。类文件BlueCross$1.class因为没有名字所以叫无名类(anonymous class)。它是由下面这几行语句产生的：

```
addWindowListener(new WindowAdapter()
   {public void windowClosing(WindowEvent e){System.exit(0);}});
```

这些语句使得用户可以正常终止程序的运行。我们也可按照Java的习惯把这几行语句的结构写得更明确些：

```
addWindowListener
(  new WindowAdapter()
   {  public void windowClosing(WindowEvent e)
      {  System.exit(0);
      }
   }
);
```

addWindowListener方法的参数必须是实现接口的WindowListener类的目标。也就是说这个类必须定义7个方法，其中之一就是windowClosing。WindowAdapter的基类把这7个方法定义成空函数。在上面的程序段里，addWindowListener的参数代表WindowAdapter的一个无名子类的目标。我们在这个子类里改写 windowClosing方法。

从BlueCross构件可看出frame的大小为360 × 150。canvas的尺寸比它要稍小一些（假定用户不用鼠标在屏幕上修改尺寸）。编译后如键入如下命令：

```
java BlueCross
```

输出的显示如图1-4所示。

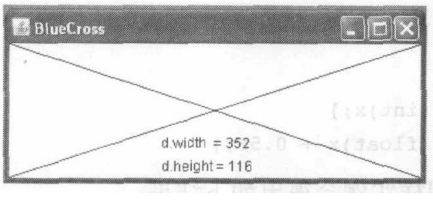


图1-4 canvas内最大可能的蓝色叉形及canvas的尺寸

我们在Microsoft Windows编程中称frame里用于输出图形的空白区域为client rectangle。在这里我们会叫它canvas，实际上是AWT class的子类（BlueCross.java程序中的CvBlueCross也是一样）。假如我们直接在frame里输出图形，就会遇到坐标系的问题：其原点就会在frame的左上角。换句话说，*x*坐标由左向右增加，而*y*坐标则由上往下增加。所以为方便起见，我们将总是使用canvas。

在以上实例中，虽然整个框架内最长的横线有360个像素，但canvas上只有352个，其余的8个像素用在了左右边框上了。纵向上类似，在整个框架内的150个像素中，有116个属于canvas，剩下的34个像素用在了上下边框和标题栏上。这些尺寸数据显然会随着用户改变窗口尺寸而改变，而且不同的程序尺寸也会不一样。所以程序中应该设有确定canvas尺寸的功能。Canvas的超类Component类中的getSize方法就是专为此目的而设计的。下面的几行语句给出如何在paint方法中获取canvas的尺寸并使用：

```
Dimension d = getSize();
int maxX = d.width - 1, maxY = d.height - 1;
```

getSize方法返回canvas上纵横两维的像素数目。因为我们是从零算起（包括零），所以纵横两维的最大像素数目*maxX*和*maxY*分别为getSize方法返回的像素数目减一。这就如同Java和C语言中数组的维数和索引之间的关系一样。例如，在语句中：

```
int[] a = new int[8];
```

数组的最大索引是7而不是8。

1.3 逻辑坐标

在实际应用中，如计算机辅助设计CAD，要画的物体可能尺寸很大，也可能很小，以至于用小数来表示。在这种情况下，我们不应停留在已有的离散、基于整数、面向设备层的坐标系统，而应定义一种连续的、基于浮点的、面向应用层的坐标系统。我们在前面已将前一种坐标系统定义为设备坐标系统，后一种则称为逻辑坐标系统。要写一个从设备坐标到逻辑坐标的转换程序倒不是一件难事。我们知道，从浮点数转化成整数有两种方法，即四舍五入法和截断法。设想我们要写以下转换程序：

iX(*x*), *iY*(*y*)：将某点的逻辑坐标*x*和*y*转换为设备坐标。

fx(*x*), *fy*(*y*)：将某点的设备坐标*x*和*y*转换为逻辑坐标。

先考虑*x*坐标，第一种方法是**四舍五入法**：

```
int iX(float x){return Math.round(x);}
float fx(int x){return (float)x;}
```

用此方法可以产生如下结果：

iX(6.7) = 7 和 *fx*(6) = 6.0

第二种方法是**截断法**：

```
int iX(float x){return (int)x;}
float fx(int x){return (float)x + 0.5F;}
```

用这种转换方法，上述同样实例会得出如下结果：

iX(6.7) = 6 和 *fx*(6) = 6.5

使用这两种方法，一个浮点数*x*和将它转换成设备坐标后再转换回逻辑坐标的浮点数*fx*(*iX*(*x*)) 之间的差一定不会大于0.5。在实际应用中，如果逻辑坐标多半为浮点数，四舍五入法损失精度的机会比用截断法要小，所以本书将使用四舍五入法。

上述用于*x*坐标的方法 *iX* 和 *fx*也可以用于*y*坐标，这里还要考虑到*y*坐标轴的方向。设备坐标系的原点由显示器的设置而定，通常设在canvas的左上角，这样*y*轴由上而下正向增加，在文字书写上很自然，但却和我们平常在数学中所学的*y*轴方向相反，所以不便用于计算机图形的应用。比如，我们一般认为自左向右并向上的线段斜率为正。庆幸的是，我们可以很容易地将*y*轴定义成我们所习惯的相反的方向：

$$y' = maxY - y$$

这样在canvas的左下角，设备的*y*坐标点为*maxY*，而逻辑*y*坐标点为0，这就是为什么在下列转换方法中表达式*maxY*−*y*在来回转换中都要用到：

```
int iX(float x){return Math.round(x);}
int iY(float y){return maxY - Math.round(y);}
float fx(int x){return (float)x;}
float fy(int y){return (float)(maxY - y);}
```

图1-5显示了一段canvas程序，canvas的*maxY*为16。

图中的像素由黑点表示，画在方格的中间。在本书中，我们将用小写字母*x*和*y*来表示逻辑坐标，大写字母*X*和*Y*表示设备坐标（但在Java程序的习惯使用中，变量名都是以小写字母开头的，所以我们在程序中就不用*X*和*Y*来表示设备坐标了）。在图1-5中，每个点所标的设备坐标

(*X*, *Y*) 都带有圆括号。比如，该局部canvas的右上角有个设备坐标为(8, 2)的像素，其逻辑坐标为(8.0, 14.0)，通过如下程序转换得到：

```
iX(8.0)  = Math.round(8.0) = 8
iY(14.0) = 16 - Math.round(14.0) = 2
fx(8)    = (float)8 = 8.0
fy(2)    = (float)(16 - 2) = 14.0
```

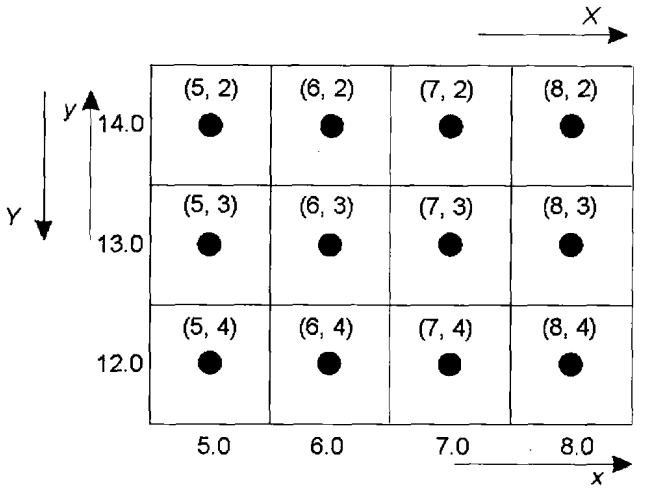


图1-5 逻辑与设备坐标，假定 $maxY = 16$

该点周围的方格表示所有满足下列条件的点 (x, y)：

$7.5 \leqslant x < 8.5$

$13.5 \leqslant y < 14.5$

我们的方法*iX*和*iY*把在以上范围内的所有点转换成了位于 (8, 2)的像素。

接下来我们用一个程序来讲述如何将浮点逻辑坐标转换成整数设备坐标。该程序先画一个正方形*ABCD*，*AB*为底边，*CD*为顶边。然后用

$q = 0.05$

$p = 1 - q = 0.95$

和下面的公式

```
xA1 = p * xA + q * xB;
yA1 = p * yA + q * yB;
xB1 = p * xB + q * xC;
yB1 = p * yB + q * yC;
xC1 = p * xC + q * xD;
yC1 = p * yC + q * yD;
xD1 = p * xD + q * xA;
yD1 = p * yD + q * yA;
```

计算接近*A*、*B*、*C*和*D*，并分别落在*AB*、*BC*、*CD*和*DA*上的新四点*A′*、*B′*、*C′*和*D′*。新画的正方形*A′ B′ C′ D′*比*ABCD*稍小一点，而且逆时针转了个小角度。我们用同样的原理画第三个正方形*A″ B″ C″ D″*、第四个正方形，等等，一直画到第80个正方形，其结果如图1-6所示。

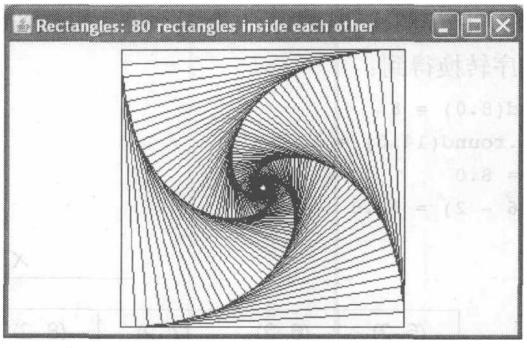


图1-6 相互嵌套并旋转一个小角度的多个正方形

如果我们改变窗口大小，新的同样嵌套在一起的正方形又会出现在canvas的中间，而且长宽比例保持不变。如果没有浮点逻辑坐标系统和向上增加的y轴，下面的程序就不那么容易写了：

```
// Rectangles.java: Draws 50 rectangles inside each other.

import java.awt.*;
import java.awt.event.*;

public class Rectangles extends Frame
{  public static void main(String[] args){new Rectangles();}

   Rectangles()
   {  super("Rectangles: 80 rectangles inside each other");
      addWindowListener(new WindowAdapter()
         {public void windowClosing(WindowEvent e){System.exit(0);}});
      setSize(600, 400);
      add("Center", new CvRectangles());
      show();
   }
}

class CvRectangles extends Canvas
{  int maxX, maxY, minMaxXY, xCenter, yCenter;

   void initgr()
   {  Dimension d = getSize();
      maxX = d.width - 1; maxY = d.height - 1;
      minMaxXY = Math.min(maxX, maxY);
      xCenter = maxX/2; yCenter = maxY/2;
   }
   int iX(float x){return Math.round(x);}
   int iY(float y){return maxY - Math.round(y);}

   public void paint(Graphics g)
   {  initgr();
      float side = 0.95F * minMaxXY, sideHalf = 0.5F * side,
```

```
            xA, yA, xB, xD, yB, xC, yC, yD,
            xA1, yA1, xB1, yB1, xC1, yC1, xD1, yD1, p, q;
        q = 0.05F;
        p = 1 - q;
        xA = xCenter - sideHalf; //Bottom left
        yA = yA = yCenter - 0.5F * side;
        xB = xCenter + sideHalf; //Bottom right
        yB = yA;
        xC = xCenter + sideHalf; //Top right
        yC = yA + side;
        xD = xCenter - sideHalf; //Top left
        yD = yA + side;
        for (int i=0; i<80; i++)
        {  g.drawLine(iX(xA), iY(yA), iX(xB), iY(yB));
           g.drawLine(iX(xB), iY(yB), iX(xC), iY(yC));
           g.drawLine(iX(xC), iY(yC), iX(xD), iY(yD));
           g.drawLine(iX(xD), iY(yD), iX(xA), iY(yA));
           xA1 = p * xA + q * xB;
           yA1 = p * yA + q * yB;
           xB1 = p * xB + q * xC;
           yB1 = p * yB + q * yC;
           xC1 = p * xC + q * xD;
           yC1 = p * yC + q * yD;
           xD1 = p * xD + q * xA;
           yD1 = p * yD + q * yA;
           xA = xA1; xB = xB1; xC = xC1; xD = xD1;
           yA = yA1; yB = yB1; yC = yC1; yD = yD1;
        }
    }
}
```

在canvas类中的CvRectangles类里， initgr方法结合paint前的其他语句也可以用于其他程序。

这里有一点原则值得注意，每个正方形上的端点计算，都是基于上次计算得到的浮点坐标，而不是基于由浮点转换而来的整数坐标。可以形象地表达如下（该原则可适用于许多计算机图形学应用）：

```
float → float → float → float → float → float → float
  ↓       ↓       ↓       ↓       ↓       ↓       ↓
 int     int     int     int     int     int     int
```

而不应该是下面的顺序：

```
float   float   float   float   float   float   float
  ↓ ↘     ↓ ↘     ↓ ↘     ↓ ↘     ↓ ↘     ↓ ↘     ↓
 int     int     int     int     int     int     int
```

如果按照后一种顺序进行转换，已损失精度的整型设备坐标不仅用来确定图形输出，而且还参与了后续步骤的计算。如此一来，损失的精度就不断地被积累起来。总结上述讨论，我们将逻辑和设备坐标系统进行比较，主要包括以下几方面：1) 本书中所用的习惯（不适用于书中的Java程序），2) 编程语言的数据类型，3) 坐标值特征，4) y轴的正向方向。结果如表1-1所示。