热点透视

Java 技术在二维图象处理方面的应用(下)

徐 鹏 王克宏
(清华大学计算机系,北京 100084)

3 文字和字体

您可以使用 Java 2D API 中的转换和绘制机制对文字和字符串进行处理。在 Java 2D API 中还增加了一些与文字相关的类,用以对文字的布局和字体提供更加完善的控制。

3.1 管理文字

Java 2D API 提供了一个强化的 Font 类,它针对字体提供了较 java.awt.Font 更加全面的控件。Font 类提供了详细的字体信息规范,同时您可以对一种字体及其中的每个元素的信息进行访问。

* 字符、字元和字体

一个字符串通常是由一系列字符按照一定序列组成的。当程序绘制字符串时,所选择的字体将决定字符串呈现的样子。然而,一种字体用来显示字符串所使用的形状并不总是与个别字符相对应。而一种字体用来代表一个字符串中字符的形状称为字元。一种字体可能代表一个字符或者一定的字符组合。一种字体可以被认为是多个字元的集合。一种单一的字体可能具备许多种外观形式,例如加黑、斜体、多哥式等等。一种字体的所有外观形式都具有一种相似的印刷设计,因此它们可以被认为是同一家族中的成员。字体类体现了系统所支持的所有字体中的一个实例,例如最常见的字体包括 Helvetica Bold 和 Courier Bold Italic。

3.1.1 指定和获取字体信息

用于描述一种字体的信息很多,其中包括字体名称、所使用的造字技术以及其类型参数等。每个 Font 对象包含了针对字体名称、大小、转换方式和其他字体特性的属性。您可以直接通过 Font 类所提供的很多便于使用的方法来访问这些数据。

程序员还可以通过 Font 类中的方法 getGlyphMetrics( )和 getGlyphOutline( )对量度和轮廓信息进行访问。您可以通过使用 Font 对象的大小和转换方式来对 getGlyphOutline( )返回的 Shape 对象进行放缩,但是这并不会对与任何 Graphics2D 对象有关的转换操作造成影响。

3.1.2 访问文字路径

您可以使用 Font.getGlyphOutline( )方法对一种字体中任何字元的形状进行访问,如图 4 所示。StyledString 类还提供了一个 getStringOutline( )方法,用以简化将整个的文字转换成轮廓图的过程。这个方法返回一个 Shape 对象,它对任何应用到 Font 对象上的与字符串相关的转换都产生影响。

3.1.3 转换文字

通过使用方法 FontderiveFont( ),您可以生成一个具有于现存 Font 对象不同属性的新的 Font 对象。例如,为了生成一个具有指定大小的字体,您可以生成一个具有特定大小的 Font 实例,并使用 Font.deriveFont 来应用一个 AffineTransform 生成一个新的 Font 对象。类似地,您可以将一个 AffineTransform 应用到 Font 对象上对文字进行倾斜处理,如图 10 所示。

图 10 经过转换的文字

在第 1 个图象中,字符串"Java"围绕中心点旋转了多次。在第 2 幅图中,对字体进行了旋转处理,从而生成了一种新的字体。

下面的程序就是使用方法 deriveFont( )来实现这种效果的:

```
// 产生变形的字体
AffineTransform fontAT = new AffineTransform();
fontAT.setToScale(72.0, 72.0);
// 描述您所使用的字体,并进行实例化
Font theFont = new Font("Helvetica", Font.PLAIN,
1);
Font theDerivedFont = theFont.deriveFont(fontAT);
// 定义显示变换
AffineTransform at = new AffineTransform();
at.setToTranslation(400.0, 400.0);
g2d.transform(at);
at.setToRotation(Math.PI / 2.0);
// 生成一个 StyledString 对象,指定文字和转换字体
StyledString ss = new StyledString("Java",
theDerivedFont);
// 每 90 度绘制一个字符串
for (int i = 0; i < 4; i++) {
  g2d.drawString(ss, 0.0f, 0.0f);
  g2d.transform(at);
}
// 生成一种斜字体
fontAT.append(new AffineTransform(1.0, 0.0,
-1.2, 1.0, 0.0, 0.0));
theDerivedFont = theFont.deriveFont(fontAT);
// 生成一个 StyledString 对象,指定文字和倾斜字体
ss = new StyledString("Java", theDerivedFont);
// 转换到一个新位置
at.setToTranslation(400.0, 0.0);
g2d.transform(at);
at.setToRotation(Math.PI / 2.0);
// 绘制具有倾斜度的字体
for (int i = 0; i < 4; i++) {
  g2d.drawString(ss, 0.0f, 0.0f);
  g2d.transform(at);
}
```

3.2 布　局

在显示文字之前,有必要对每个字符应当如何放置来进行准确地判断。大多数客户端程序将这个布局的过程留给了服务程序来完成,由后者来提供一系列基于在字体中包含信息的算法。Java 2D API 提供了文字布局功能,对一般的情况进行处理,其中包括具有固定字体的文字字符串、固定的语言和双向性文字。一些高级的客户程序可能希望由自身来计算文字布局,这样它们就可以对使用那些字元以及将这些字元安放在哪里等内容进行更好的控制。通过使用诸如字元大小、字间距表等信息,高级客户程序 可以使用它们自己的算法来计算文字布局,而回避系统的布局机制。

类 GlyphSet 提供了一种显示客户布局结构最终结果的方法。一个 GlyphSet 对象可以被认为是一种用于提取一个字符串并准确计算此字符串应当如何显示的算法的输出结果。系统具有一种内制的算法,同时 Java 2D API 允许高级客户程序可以定义它们自己的算法。通常,当您构造一个 StyledString 对象时,您将会传递一个希望显示的文字,而系统会对此文字进行处理,并基于企布局算法为您建构一个 GlyphSet 对象。一个 GlyphSet 对象基本上是一个字元以及字元位置的数组。字元被用于代替字符来提供对诸如字间距调整等布局特性的控制。例如,当显示字符串"final"时,您可能希望将最前面的子串"fi"用连字进行处理(即两个字母之间不留间隙),在这种情况下,GlyphSet 对象将具有比原始字符串中字符数量更少的字元。图 11 和图 12 展示了如何在缺省的布局结构和客户化的布局结构下使用 GlyphSet 对象。

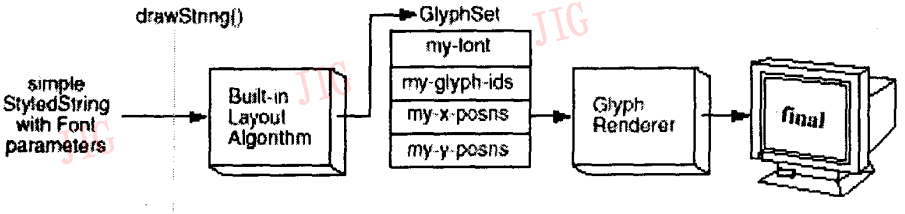

图 11　使用内制结构算法

在图11中，客户端建立了一个StyledString对象，并将其传递给方法drawString( )。内制的结构算法决定了应当使用哪些字元，并将每个字元安放在哪里。这一信息通过使用GlyphSet实例来实现存储。这些GlyphSet对象被传递给一个进行实际绘制操作的字元显示例程。

在图12中，客户端集合了进行文字布局所需的信息。一个客户化的结构算法被用于决定使用使用哪些和如何使用字元。在这个实例中，子串"fi"通过连字处理。之后这种结构信息被存储于GlyphSet对象中，之后GlyphSet对象被传递给方法drawString( )，同时直接将其传递到最终的字元显示程序中。

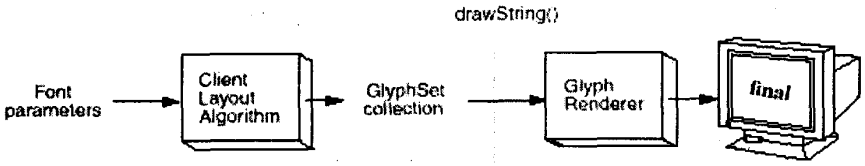

图12 使用客户化的结构算法

4 颜色管理

彩色成像系统是任何图形系统的一个基本的组成部分，它还常常是成像模型中最复杂的一部分。利用Java 2D API设计的程序可以提供高质量彩色输出效果，同时使高档客户程序可以更加充分地使用颜色。

4.1 指定颜色

为了显示一个具有特定颜色的矩形，您需要一种利用Java语言来描述这种颜色的方式。目前存在着许多不同的方式来描述一种颜色，例如一种颜色可以被描述为红绿兰(RGB)三原色的组合，也可以是兰绿色、紫红色、黄色和黑色(CMYK)的组合。这些用以指定颜色的不同技术被称为颜色空间。

计算机屏幕所显示的颜色是通过将红、绿、兰三种颜色按照一定比例混合而成的。因此使用RGB颜色空间是在计算机屏幕上进行成像的一种标准空间。与此类似，四颜色打印过程使用兰绿色、紫红色、黄色和黑色墨水在一张纸上产生颜色，而打印出的颜色可以通过一个CMYK颜色空间中的百分比来指定。由于计算机显示器和彩色打印的普及，RGB和CMYK颜色空间都常被用于描述颜色信息。然而，这两种颜色空间都具有一个重要的不足之处，这就是它们具有设备依赖性。一个打印机使用的兰绿色墨水可能并不能够完全与另外一台打印机的兰绿色墨水匹配。与此相似，一种RGB空间中的颜色在一台显示器上呈现蓝色，而在另外一台显示器中则可能会略带紫色。Java 2D API将RGB和CMYK作为颜色空间类型。一台具有特殊磷光质的显示器模型定义了自己的RBG颜色空间，同样一台特定的打印机模型也定义了自己的RGB颜色空间或CMYK颜色空间。而最常用的一种独立于设备的颜色空间就是XYZ颜色空间，当您使用CIEXYZ来指定一种颜色时，这种颜色就不在依附于设备了。但由于在某些情况下使用其他颜色空间中的颜色更加有效，因此并不总是能够在CIEXYZ颜色空间中描绘颜色。为了在使用一种依赖于设备的颜色空间(特别是RBG)空间来呈现一种颜色时保质结果的一致性，有必要揭示出RGB空间是如何与一种独立于设备的空间进行关联的。

一种的两个颜色空间之间进行映射的方式就是附加有关描述一种依赖于设备的空间如何于一种独立于设备的空间进行关联的信息。这种信息称为一个仿形图(Profile)。图13展示了如何将一种原色和一幅经过扫描的图象传递给Java 2D API，以及它们是如何在不同的输出设备上显示的。正象您在图中看到的那样，输入颜色和图象均附着了仿形图。

一旦API具备了一种指定的颜色，它必需在针对一种输出设备对此颜色进行重新处理。由于这些输出设备具有自身的成像特点，因此这一重新处理的过程非常重要，只有这样才能够确保产生正确的效果。另外一个仿形图与每个输出设备有关，它用以描述颜色需要如何进行转换以便产生正确的输出结果。为了获得一致且正确的颜色，必须在一种标准的颜色空间的基础上对输入颜色和输出颜色进行仿形。例如，一种输入颜色可以从其最初的颜色空间映射到一个标准的独立于设备的空间中，之后再从这

个空间中映射到输出设备的颜色空间中。在许多方面,颜色的转换是模仿在(x, y)坐标空间中的图形转换而进行的。在这两种情况下,转换操作都是首先在一个标准空间中指定坐标,之后将这些坐标映射到一个指定设备的空间中进行输出。

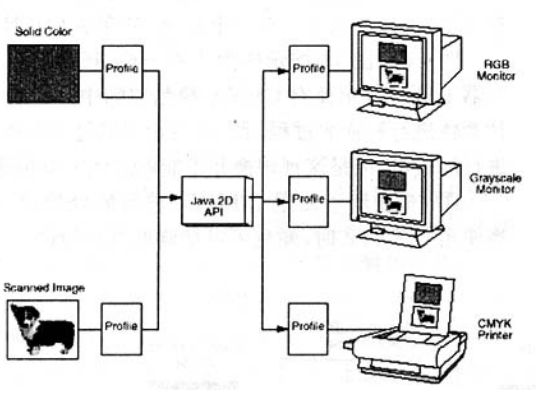

图 13　使用仿形图在不同颜色空间中进行映象

在 Java 2D API 中关键的颜色管理类包括 Color、ColorModel 和 ColorSpace。其中 Color 类依据在特定颜色空间中的颜色组建来描绘一种颜色,ColorModel 类提供了将一个 Image 或者 BufferdImage 对象中一个象素的各个组件转换成一个特定颜色空间中颜色组件所必需的信息。而 ColorSpace 类提供的方法可以转换一个特定颜色空间中的颜色组件。

4.2　颜　色

Color 类提供了对一个特定颜色空间中一种颜色的描述信息。一个 Color 对象实例包含了颜色组件的取值和一个 ColoSpace 对象。由于 ColorSpace 对象可以在生成一个新的 Color 实例是指定,因此 Color 类可以处理任何颜色空间中的颜色。Color 类具有很多方法,对一种称为 sRGB 的标准 RGB 颜色空间提供了支持。SRGB 是 Java 2D API 中缺省的颜色空间。Color 类所定义的许多构造程序均忽略了 ColorSpace 参数。这些构造程序假定当前颜色的 RGB 取值在 sRGB 中定义了,同时使用一个缺省的 ColorSpace 实例在代表当前的颜色空间。Java 使用 sRGB 主要是为了给程序设计人员提供方便。许多应用程序都与 RGB 图象和显示器有关,同时定义一种标准的 RGB 颜色空间可以使编写程序更加容易。ColorSpace 类定义的方法 toRGB( )和 fromRGB( ),这样开发人员就可以很方便地在这种标准空间中获取颜色了。但这些方法对于高精度的颜色校验和转换操作来说并不适用。

为了在 sRGB 以外的颜色空间中生成一种颜色,您可以适用 Color 构造程序,它采用了一个 ColorSpace 对象和一个代表颜色组件的浮点数组,ColorSpace 对象等同于一个颜色空间。

4.3　ColorModel

ColorModel 类包含了用于解释一个图象中象素的数据,图象与一个特定颜色空间之间的映射组件,它还可以从经过打包的象素数据中提取出象素组件、通过一个查找表转换象素数据。为了判断一个图象中特定象素的颜色取值,您需要对每个象素中颜色信息的编码方式有所了解。于一个图象相关的 ColorModel 类考虑到象素值在其构成的颜色组件之间的转换,将相应的方法和数据进行了封装。

除了在 JDK 1.1 中定义的 DirectColorModel 和 IndexColorModel 以外,Java 2D API 还提供了两种颜色模型,它们分别是:

* ComponentColorModel:可以处理任意一种 ColorSpace 和一个颜色组件数组。这个模型可以被用于代表大多数 GraphicsDevices 上的大多数颜色模型;

* PackedColorModel:这是一个针对代表象素值模型的基础类,这些模型具有直接嵌入在一个整型象素比特位中的颜色组件;

一个 PackedColorModel 存储了描述如何从管道中提取颜色和 alpha 组件的打包信息。在 JDK 1.1 中的 DirectColorModel 就是一个 PackedColorModel 对象。

4.4　ColorSpace

一个 ColorSpace 对象代表了一个颜色空间,例如 RGB 或 CMYK 空间。一个 ColorSpace 对象可以作为一个颜色空间标识符,它定义了一个 Color 对象的特定颜色空间,或者通过一个 ColorModel 对象定义一个 Image、BufferedImage 或者 GraphicsConfiguration 对象的颜色空间。ColorSpace 提供了用于在 sRGB 和 CIEXYZ 颜色空间之间进行颜色转换的方法。所有 ColorSpace 对象能够从当前的颜色空间中将一种颜色映射到 sRGB 空间中,并将一种 sRGB 颜色转换到当前的颜色空间里。由于每个 Color 对象包含了一个 ColorSpace 对象,因此每个 Color 对象也能够在 sRGB 颜色空间之间进行转换。与此相

类似，由于每个 GraphicsConfiguration 对象也于一个 ColorSpace 对象相关，因此任何 sRGB 颜色可以在任何输出设备上显示出来。

这个过程所使用的方法包括：

* toRGB：用于将当前颜色空间（例如 CMYK 空间）中的一个 Color 对象转换成 sRGB 空间中的 Color 对象；

* fromRGB：用于将 sRGB 空间中的 Color 对象转换成当前颜色中的 Color 对象；

虽然通过 sRGB 来进行映射可以起到一定的作用，但是这并不是一种最佳的方案。对于一个图象来说，sRGB 不能够呈现出象 CIEXYZ 颜色空间中那么丰富多彩的颜色。如果在一个区域中所指定的颜色超过了 sRGB 的颜色范围，那么使用 sRGB 来作为一种中介空间将会导致信息的丢失。为了解决这个问题，ColorSpace 类可以将颜色映射到另外一个颜色空间，这就是作为“转换空间”的 CIEXYZ。方法 toCIEXYZ( )和 fromCIEXYZ 的功能是从当前的颜色空间中将颜色取值映射到转换空间中。这些方法对任意两个颜色空间中在高精度条件下进行的转换提供了支持，每次可以对一个 Color 对象进行转换。

图 14 和图 15 展示出为了在一台 RGB 彩色显示器上进行显示而对 CMYK 颜色空间中指定的一种颜色进行转换的过程。图 14 展示了通过 sRGB 来进行的映射，但是这种转换并不能够达到十分精确。

图 15 展示了使用 CIEXYZ 进行的处理过程。当使用 CIEXYZ 时，颜色可以精确地进行传送。

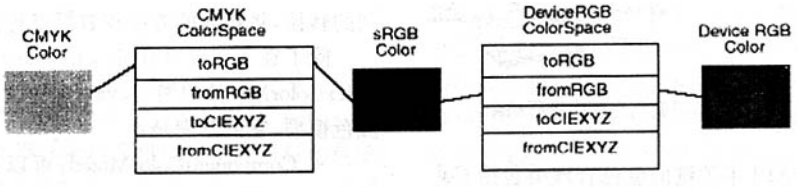

图 14 通过 sRGB 进行映射

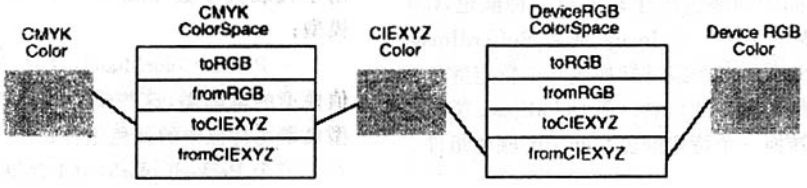

图 15 通过 CIEXYZ 进行映射

5 成 像

成像处理过程主要是围绕对光栅图象的操作展开的，通常是提高图象的可视化效果或者提高图象轮廓的光滑程度。在一些流行的图象编辑软件中所提供的很多图象处理效果都可以通过使用 Java 2D API 中的图象处理类或者对这些类进行扩展后实现。

5.1 图象处理与强化

Java 2D API 提供了一系列定义对 BufferedImage 和 Tile 对象进行操作的类。其中 BufferedImage 对象是对 java.awt.Image 的一个扩展，程序员可以通过使用这个对象来访问一个图象中的独立象素。而 Tile 定义了在一个图象中的数据和数据布局。这些图象处理类具有一种普遍的体系结构。每个图象处理操作被嵌入一个类中，而每个类定义了一个进行实际图象处理的过滤方法。这个过滤方法可以对一个图象源和目标 BufferedImage 对象，或者一个图象源和一个目标 Tile 对象进行操作。图 16 展示了 Java 2D API 进行图象处理操作中的基本模型。

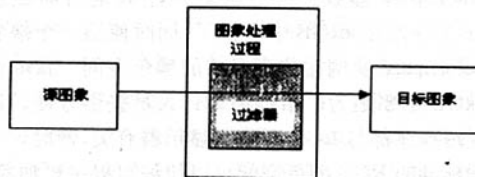

图 16 图象处理模型

在这里所支持的操作包括:

* 仿射转换;
* 增幅缩放;
* 查找表修改;
* 频带的线性组合;
* 颜色转换;
* 图象旋转;

实现这些操作的类包括 AffineTransformOp 及其子类 BilinearAffineTransformOp 和 NearestNeighborAfineTransformOp、BandCombineOp、ColorConvertOp、ConvolveOp、LookupOp、RescaleOp 和 ThresholdOp。这些类可以被用于对图象进行几何转换、模糊处理、图象锐化、增强对比度、设置域值和颜色校验等操作。图 17 展示了边界检测和强化操作,在医疗成像和映象应用程序中得到了广泛使用。边界检测被用于增强一个图象中邻近结构之间的对比度,这样浏览者就可以清晰地分辨出图象中的细节。

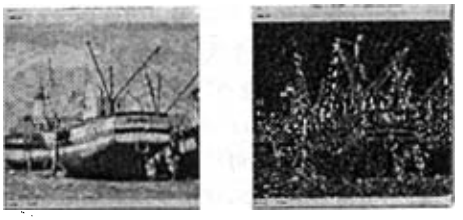

图 17　边界检测与强化处理

在图 18 中展示了对查找表进行重新设置域值并进行了放缩操作后对图象显示效果的影响。对查找表进行放缩操作后可以增强一个黑白图象的动态变化范围,进一步表现出不同区域中的细节。域值的设置可以将一定强度范围内的所有颜色转换成一种特定的颜色。

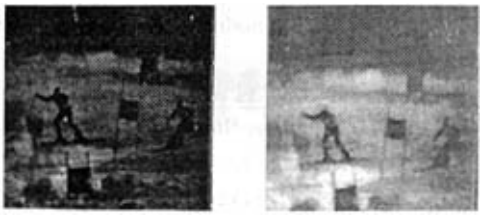

图 18　查找表处理

我们通过表 1 对 Java 2D API 提供的图象处理类进行总结,其中每个类都可以针对一个 BufferedImage 或 Tile 对象进行操作。

LookupOp	使用一个查找表来对象素数据进行重新映射,从一个深度映射到另外一种颜色深度
RescaleOp	通过一个比例关系并增加一个偏移量对每个象素进行修改,这样就形成了对数据的重新放缩处理
ThresholdOp	将一个特定范围内的象素深度设置为一个常量

表 1　图象处理类

类名	功能描述
AffineTransformOp	针对一个图象或者 tile 对象上仿射信息的基础类
BilinearAffineTransformOp	使用双线性插补操作对图象或 tile 对象进行仿射转换
NearestNeighborAffineTransformOp	通过使用邻近插补操作对图象和 tile 对象进行仿射变换
BandCombineOp	对一个图象和 tile 对象进行任意的线性组合
ColorConvertOp	进行颜色转换操作
ConvolveOp	进行空间过滤

下面这段代码展示了使用图象处理类 ConvolveOp 的方法。旋转是一种处理方法,它构成了最具有空间性的过滤处理算法,同时它还是对一个图象中具有相近象素取值的每个象素值进行加权或平均的处理方法。这样每个输出的象素就可以通过在 Kernel 中从数学角度来指定取值。在这个实例中,源图象中的每个象素的取值为其周围 8 个点的平均值。

```
float weight = 1.0f/9.0f;
float[]elements = new float[9]; // 生成 2D 数组
// 用 9 个取值相同的元素填充数组
for (i = 0; i < 9; i++) {
```

```
    elements[i] = weight;
}
// 使用数组元素作为参数来生成一个 Kernel 对象
private Kernel myKernel = new Kernel(3, 3, 1, 1, elements);
public ConvolveOp simpleBlur = new ConvolveOp(myKernel);
// sourceImage 和 destImage 为 BufferedImage 实例
simpleBlur.filter(sourceImage, destImage) // 对图象进行模糊处理
```

变量 simpleBlur 包含了一个新的 ConvolveOp 实例，它实现对 BufferedImage 或 Tile 对象的模糊操作。假设 sourceImage 和 destImage 为两个 BufferedImage 实例。当您调用 ConvolveOp 类的核心方法 filter( )时，它将一个象素点周围的 8 个点取值求平均，然后将这个平均值作为此象素点的最后取值。在这个实例中的变换内核可以通过下面的矩阵来表示：

$$\boldsymbol{K}=\begin{bmatrix}0.111\,1 & 0.111\,1 & 0.111\,1\\ 0.111\,1 & 0.111\,1 & 0.111\,1\\ 0.111\,1 & 0.111\,1 & 0.111\,1\end{bmatrix}$$

当对一个图象进行卷积处理时，最终生成图象中的每个象素点的取值可以通过对周围象素点取值进行加权来计算。下面的公式展示了在进行卷积运算时如何将加权值与源图象中的象素点对应起来。在内核中的每个值都与图象中的一个空间位置相对应。

$$\boldsymbol{K}=\begin{bmatrix}i\text{-}1,j-1 & i,j-1 & i+1,j-1\\ i-1,j & i,j & i+1,j\\ i-1,j+1 & i,j+1 & i+1,j+1\end{bmatrix}$$

一个最终象素点的取值是内核加权值与相应源象素取值乘积的总和。对于很多简单操作来说，内核是一个均匀对称矩阵，其权值总和为 1。在这个实例中的卷积内核相对比较简单。它对源图象中每个象素进行均匀加权。通过选择一个在较高或者较低级别上对源图象进行加权处理的内核，一个程序可以对最终图象的深度进行增加或减少。在 ConvolveOp 构造程序中设置了一个 Kernel 对象，它的功能是对运行的过滤处理的类型进行判断。通过设置其他取值，您可以完成其他类型的卷积处理，其中包括模糊处理、锐化以及平滑操作。

5.2 屏外缓冲区

在图象处理过程中准备一个图形元素缓冲区是非常有必要的，特别是在图象比较复杂或者重复使用某一图形时更是这样。例如，如果您希望多次显示一个比较复杂的图形，那么您可以将其一次绘制到一个屏外缓冲区中，之后将缓冲区中的内容拷贝到窗口里的不同位置上。通过一次绘制图形并对其进行拷贝，您可以实现更加快速地显示图形。java.awt 包实现了屏外缓冲区，这样您就可以象绘制一个窗口一样绘制出一个 Image 对象。我们可以通过调用 drawImage 来将屏外缓冲区中的内容绘制到屏幕上。在显示动画图象时经常使用屏外缓冲区。例如您可以使用一个屏外缓冲区来实现一次绘制出一个对象之后将其复制到窗口的任何位置上。与此类似，您可以使用一个屏外缓冲区来根据用户的鼠标操作来移动图形。在一般状态下，系统在每个鼠标位置上重新绘制出图象，然而在 Java 2D API 中，您可以一次将图形绘制到一个屏外缓冲区中，之后根据用户的鼠标操作事件将缓冲区中的内容复制到鼠标位置上。

图 19 展示了一个程序如何将图象绘制到一个屏外缓冲区中，之后多次将图象复制到一个窗口中。生成一个作为屏外缓冲区图象的最简单方式就是使用方法 Component.createImage( )。这个方法生成一个位于屏幕以外、可绘制的图象，同时这个图象具有程序员指定的大小，这个生成的图象不透明，同时具有 Component 的前景和背景颜色。您不能够调整这个图象的透明度。您还可以直接构造一个 BufferedImage 对象来作为一个屏外缓冲区使用。当您需要对屏外图象的类型或透明度进行控制时这种

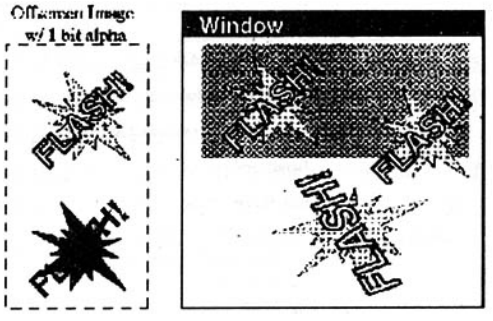

图 19 使用一个屏外缓冲区

方式是比较有效的。BufferedImage 支持许多预定义的图象类型，其中包括：

* TYPE—3BYTE—BGR；

* TYPE—4BYTE—ABGR;
* TYPE—4BYTE—ABGR—PRE;
* TYPE—BINARY;
* TYPE—CUSTOM;
* TYPE—INT—ARGB;
* TYPE—INT—ARGB—PRE;
* TYPE—INT—RGB;

GraphicsConfiguration 提供了很多便于使用的方法,它们可以自动按照正确的格式生成缓冲图象,您还可以查询于图形设备相关的图形配置,这些信息是您在构造一个兼容型 BufferedImage 对象所必需的。这种机制可以被应用到包括打印机在内的很多输出设备中。例如,您可以构造一个与一台特定打印机兼容的 BufferedImage 对象,这样图象的颜色空间、深度和象素布局就可以在打印机上充分地表现出来了。

程序员可以通过使用类 BufferedImage 直接对一幅图象中的象素点进行处理。您可以通过这个类指定图象的 ColorModel、图象数据等参数对一个图象进行控制。通过为图象数据提供存储空间或访问现有的存储数据,您可以直接对一幅图象中的数据进行处理。与其他图象类似,一个 BufferedImage 对象可以包含一个 alpha 通道。正象图 20 所示,我们用 1 比特的深度的 alpha 通道就可以将经过绘制的区域和未经绘制的区域区分开来。Java 2D API 使图象于图形有效地结合了起来(在这个实例中的图形为一个矩形)。在其他环境下,您可能希望具有一种更深层的 alpha 通道,这样就可以对图象的透明度进行控制。您可以通过选择相应的深度、处理 alpha 通道中的数据并修改被用于绘制 BufferedImage 的 Graphics2D 对象中的合成操作来控制一个 BufferedImage 对象中的 alpha 字符集。

一个图象的 ColorModel 对象指定了在图象的 Tile 中数据的颜色空间,以及数据与颜色和 alpha 组件的映射方式。Tile 类定义了图象中的数据和数据布局。通过利用由 Tile 和 ColorSpace 对象所封装的信息,您可以直接对一个图象中的象素进行访问和处理。

6 图形设备

GraphicsDevice 对象代表了包括显示器和打印机在内的所有图形设备,其中封装了一个设备的功能和属性信息。Java 2D API 使程序员可以通过 GraphicsEnvironment 类和这些 GraphicsDevice 对象以及相关的 GraphicsConfiguration 对象,对应用程序运行环境的属性进行查询。

6.1 图形环境

您可以通过 GraphicsEnvironment 对整个操作系统的信息进行访问,GraphicsEnvironment 类的功能包括:

* 提供了代表输出设备的 GraphicsDevice 对象列表;
* 对一个系统的文本性能进行封装;

您可以对图形设备列表进行检测,并查询相关的图形配置以便判断当前设备的性能。通常,您不需要对这类信息进行访问。您还可以通过 GraphicsEnvironment 类对字体方面的属性(例如字体的名称和类型等)进行搜索。您还可以使用它来列出所有使用的字体。这一功能在您实现一个字体面板或字体菜单时更显得非常有用。

* GraphicsDevice

Java 2D API 使用 GraphicsDevice 类来代表一种实际的输出设备。通常,您不需要访问这个类。GraphicsDevice 对象直接与实际的输出设备对应起来,例如打印机或窗口。每个 GraphicsDevice 具有一个与其相关的 GraphicsConfiguration 对象列表。在 X—windows 环境下,在相同的显示器上不同的窗口中可以具有不同的象素布局,而同一个显示器可以通过不同的象素配置从而实现不同的显示效果。在 Macintosh 和 Windows 环境中,一台显示器上每个窗口中的象素具有相同的配置。您可以通过 GraphicsEnvironment 和 GraphicsConfiguration 来获得对一个 GraphicsDevice 的引用。其中 GraphicsEnvironment 包含了一个对应与 Java 2D API 认同设备的 GraphicsDevice 对象列表;而 GraphicsConfiguration 定义了一个方法,这个方法的返回值是与配置相关的 GraphicsDevice 对象。

* GraphicsConfiguration

您可以通过调用方法 getDeviceconfiguration( ) 来获得与一个 Graphics2D 对象相关的 GraphicsConfiguration 对象。您可以对一个 GraphicsConfiguration 对象进行查询以检测一个 GrapihcsDevice 的性能,同时可以使用方法 createCompatibleImage( )来生成 BufferedImage 对象。您还可以恢复 ColorModel 对象和缺省配置,并进行普通的转换操作。ColorModel 具有一个描述设备彩色输出性能的 ColorSpace 对象,在显示过程中它被用于进行颜色匹

配。

展示出一个 GraphicsDevice 对象是如何与其他 2D 对象结合起来使用的。

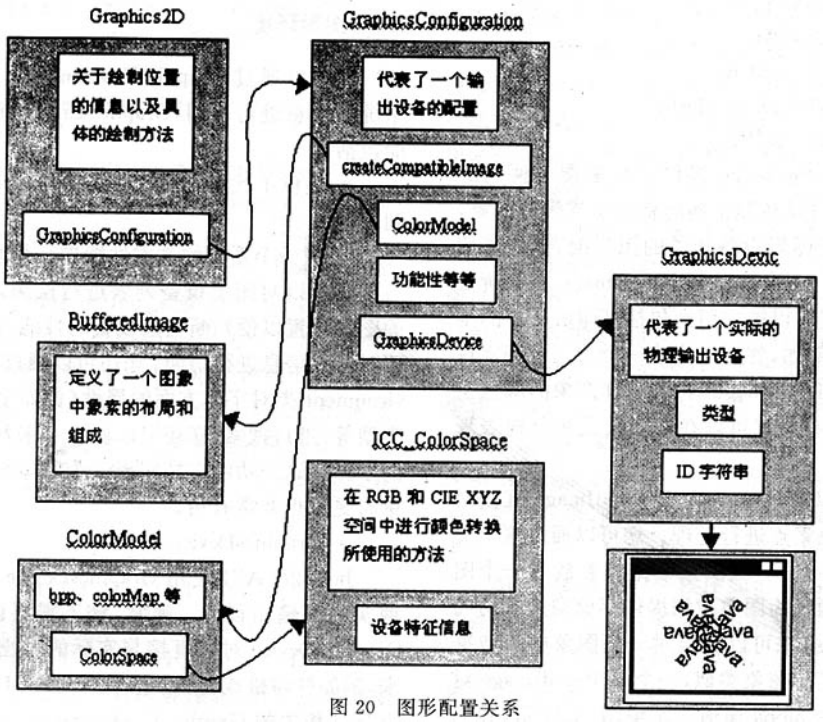

图 20 图形配置关系

7 Java 2D API 总结

下面对 Java 2D API 中的接口和类进行归纳。

7.1 接口

表 2 列出了在 Java 2D API 中定义的接口，并进行了简要的描述。

表 2 Java 2D API 中的接口

接口	描述
BufferedImageOp	描述针对 BufferedImage 对象而进行的单输入/单输出操作。通过 AffineTransformOp、ConvolveOp、RescaleOp 和 ThresholdOp 来实现。
Composite	定义了利用指定图形区域组成一次绘制任务的方法。通过 AlphaComposte 来实现。
CompositeContext	针对一次合成操作定义经过封装和优化的环境。
MultipleMaster	代表 Type1 字体。
OpenType	代表 Open Type 和 True Type 字体。
Paint	扩展：Transparency 定义了生成在 Graphics 操作中所使用的颜色的方式。通过 GradientPaint 和 TexturePaint 来实现。
PaintContext	针对一次绘制操作定义经过封装和优化的环境。
PathIterator	定义用于从一个路径中获取元素的方法
Shape	提供用于描述和检测几何路径对象的方法。通过 GeneralPath 实现。
Stroke	使一个 Graphics2D 对象可以获取封闭图形的边界。通过 BasicStroke 实现。
StyledCharacterIterator	定义用于获取文字和类型信息的的方法
TileImageConsumer	用于处理图象数据。通过 BufferedImageFilter 实现。
TileOp	定义在 Tile 对象上进行的单输入/单输出操作。通过 AffineTransformOp、ConvolveOp 和 LookupOp 来实现。
Transparency	定义三种透明度模式：不透明、半透明和透明。通过 ColorModel 实现。

7.2 类

表 3 列举并描述了在 Java 2D API 中定义的类。**（转下页）**

类	描述
AffineTransform	实现:Cloneable 代表了一种二维仿射变换,在不同的 二维坐标空间中进行映射。
AffineTrans- formOp	实现:BufferedImageOp、TileOp 进行仿射变换
AlphaComposite	实现:Composite 实现图形和图象的基本组成规则。 对源 tile 和目的 tile 对象进行操作。使用一个 alpha 通道。
Arc2D	实现:RectangularShape 代表了通过边界矩形、起始角、旋转范围 、关闭类型所定义的一个弧形
Arc2D. Float	实现:Arc2D 具有特定浮点精度的弧形。
Area	实现:sun. awt. Albert. area 代表一种支持布尔运算的几何区域。
BasicStroke	实现:Stroke 定义实线的显示特性
BilinearAffine-TransformOp	实现:AffineTransformOp 使用一种具有双线性插补处理 仿射变换的对图象进行转换。
BufferedImage	实现:Image 利用可访问缓冲区中的图象数据来 描述一个图象。它由 ColorModel 和一个 Tile 所构成。
BufferedImage-Filter	实现:TileImageConsumer、Cloneable 扩展:ImageFilter 使用一个单源/单目的的图象操作符对 ImageProducer/Consumer/Observer 范型中的 BufferedImage 或 Tile 进行 过滤。
BandCombineOp	实现:bufferedImageOp、 TileOp 允许对原始图象的色调进行 小的改变
Color	实现:Paint、Serializable 对缺省 Srgbyanse 空间和其他通过 一个 ColorSpace 对象定义的颜色空 间中的颜色进行封装。
ColorConvertOp	实现:BufferedImageOp、TileOp 对源图象进行象素之间的颜色数据转换 处理。
ColorModel	实现:Transparency 对用于将象素值转换为颜色组件的方法 进行封装。
ColorSpace	定义一个 Color 对象、Image、 BufferedImage 或 GraphicsDevice 的颜色空间。具有在 RGB 和 CIEXYZ 颜色空间之间进行转换的方法。
Component-ColorModel	扩展:ColorModel 处理 ColorSpace 和一个颜色组件数组以 与 ColorSpace 进行匹配。
ConvolveOp	实现:BufferedImageOp、TileOp 实现从源到目的的卷积。
CubicCurve2D	实现:Shape 代表着在(x, y)坐标空间中的三次 参数曲线。
CubicCurve-2D. Float	扩展:CubicCurve2D 通过浮点坐标指定的一个三次 参数曲线。
Dimension2D	对一个具有指定宽度和高度的面积 进行封装。
Ellipse2D	扩展:RectangularShape 代表由一个由边界矩形定义的矩形。
Ellipse2D. Float	扩展:Ellipse2D 代表在浮点精度下指定的一个椭圆。
FlatteningPathIterator	实现:PathIterator 提供了一种平展操作。
Font	实现:Serializable 产生字体对象。取代了 java. awt. Font, 实现了更加完善的活板功能。
FontDescriptor	包含了对一种字体的描述信息。
FontDesignMetrics	对字符的显示比例等信息进行描述。
FontFeature	代表了字距调整、半黑字体等显示信息。
FontFeatureSet	实现:FeatureSet
FontObject	代表了字体特性的一个集合, 例如权重和形态。
GeneralPath	实现:Shape 代表了通过直线、二次方程和 三次方程而构造的一个几何路径。
GeneralPath-Iterator	实现:PathIterator 被用于提取在一个 GeneralPath 中的 元素。
GlyphMetrics	提供在显示字符过程中使用的量度信息。
GlyphSet	一个电刻和电刻位置的数组。最基本 的信息被用于绘制文字。使用了精确的电 刻代码和位置信息。
GradientPaint	实现:Paint 提供了一种利用线性颜色梯度模型来填 充图形的方法。颜色的变化从起始点 P1 的颜色 C1 变化到结束点 P2 的颜色 C2。
Graphics2D	扩展:Graphics 这是使用 Java 语言进行二维图形、 图象显示的基本类。 这是对最初的 java. awt. Graphics 类的扩展。
GraphicsConf-iguration	描述了一个图形输出端的规格参数。

GraphicsDevice 描述了在一个图形环境中适用的图形设备。

GraphicsEnvironment 指定图形环境。

ICC_ClorSpace 扩展:ColorSpace
代表了独立于设备和从属于设备、基于 ICC 仿形格式规范的颜色空间。

ICC_Profile 代表了颜色仿形数据。

ICC_ProfileGray 扩展:ICC_Profile
代表了灰色的颜色空间。

ICC_ProfileRGB 扩展:ICC_Profile
代表了 RGB 类型的颜色空间。

ImageFilter 实现:ImageConsumer、Cloneable

IndexColorModel 扩展:ColorModel
代表了在 ColorModel 的颜色空间中一个固定颜色映象中作为指数的象素值;

IntegerComponentTile 扩展:Tile
定义了一个 Tile 对象,其中的象素是由一个或多个 32 位通道所组成的,它们存储在一个整数数组中。

Kernel 定义了在过滤处理中使用的一个 Kernel 对象。

Line2D 实现:Shape
代表了在(x, y)坐标空间中一段直线。

QuadCurve2D. Float 扩展:QuadCurve2D
代表了在浮点精度的(x, y)坐标空间中二次参量的曲线。

RectangularShape 实现:Shape
提供了对具有矩形边界的图形进行操作的一般规则

Rectangle2D 扩展:RectangularShape
代表了在(x, y)坐标空间中的一个

Rectangle2D. Double 扩展:Rectangle2D
代表了在双精度坐标空间中的一个矩形。

Rectangle2D. Float 扩展:Rectangle2D
代表了在浮点精度坐标空间

ShortBandedTile 扩展:Tile
定义了一个具有由多个 16 比特组件构成象素的 Tile 对象。

ShortComponentTile 扩展:Tile

StyledString 对绘制文字所需的数据进行封装。

TextHitInfo 代表了在一个 TextLayout 中字符的命中测试信息。

TextLayout 实现:Cloneable

ThresholdOp 实现:BufferedImageOp、

Line2D. Float 扩展:Line2D
代表了在浮点坐标空间中指定的一段直线。

LookupOp 实现:BufferedImageOp、TileOp
实现了从源到目的的查找操作。

LookupTable 定义了一个查找表对象。

NearestNeighborAffineTransformOp 扩展:AffineTransformOp
使用一个具有邻近内插法的仿射转换对一个图象进行转换。

PackedColorModel 扩展:ColorModel
代表了具有颜色组件的象素值

Point2D 代表(x, y)坐标空间中的一个点。

Point2D. Double 扩展:Point2D
代表在双精度条件下(x, y)坐标空间中的一个点。

Point2D. Float 扩展:Point2D
代表在浮点精度条件下(x, y)坐标空间中的一个点。

QuadCurve2D 实现:Shape
代表了在(x, y)坐标空间中二次参量的曲线。

中的一个矩形。

RoundRectangle2D 扩展:RectangularShape
代表了在(x, y)坐标空间中一个具有圆抹角的矩形,尺寸为(w, x, h),并具有特定的边角弧度的宽度和高度。

RoundRectangle2D. Float 扩展:RoundRectangle2D
代表了在浮点坐标空间中指定的一个具有圆抹角的矩形。

RescaleOp 实现:BufferedImageOp、TileOp
通过一个放缩因子对图象中每个象素乘积处理,然后加上偏移量,这样实现对图象的放缩处理。

定义了一个具有由多个 16 比特组件构成象素的 Tile 对象,这些组件存储在一个 short 整数数组中。

ShortLookupTable 扩展:LookupTable
定义了一个查找表对象,其中包含了用于多个 tile 通道的 short 数据。

对绘制文字所需的布局信息进行封装。

TexturePaint 实现:Paint 提供了一种利用特定的结构填充一个图形的方法。这个结构通过一个 BufferedImage 来指定。TileOp 对源进行域值检测。

7.3 例外

列出了在 Java 2D API 中定义的例外。

表 4　Java 2D API 中的例外

例外类	描述
NoninvertibleTransformException	如果一项操作要求进行逆向转换,而这种转换本身并不可逆,那么抛弃此例外。
NoSuchProfileDataException	如果试图访问在一个 ICC_Profile 对象中的数据,那么抛弃此例外。
TileFormatException	如果在 Tile 中存在不合法的布局信息,那么抛弃此例外。

康柏发布双端口英特尔兼容 NIC 简化向组合快速以太网的升级工作

美国康柏电脑公司于近日发布了一款基于英特尔芯片的双端口 PCI 网络接口控制器(NIC),旨在简化从快速以太网向组合快速以太网升级的工作。用于工作组到中档服务器的康柏 NC3122 快速以太网 NIC,允许用户无需更换 NIC 或操作系统驱动程序即可大幅度提高网络性能,从而确保关键环境正常运行。此外,该控制器还基于英特尔的快速以太网 ASIC,其驱动与康柏基于英特尔芯片的现有 NIC 完全兼容,使用户不必在网络中再支持多种驱动程序。

康柏 NC3122 快速以太网 NIC 专门为寻求高性能、高冗余和更高服务器可用性的客户而设计,它包括如下特性:

- PCI 热插拔——康柏 PCI 服务器技术,允许用户不必关闭系统即可更换或增加 NIC。
- 网络容错——当一个 NIC、交换机或集线器链路出现故障时,通过自动向后备 NIC 或 NIC 端口切换或进行"故障恢复",可使 NIC 始终保持一条有效链路。
- 自动负载平衡——在一台服务器中组合达 4 个快速以太网端口来提高可用带宽。当一个网络连接出现故障时,其余端口可接替故障端口继续工作。
- Cisco Fast Etherchannel——Fast Etherchannel 兼容交换机支持端口组合,数据吞吐量可达 800Mbps。

HP、MarketNet 和微软公司携手解决 2000 年问题

针对华尔街各机构的交易所迎接 2000 年(即 Y2K)需要,HP 公司、MarketNet 公司及微软公司在由安全工业协会举行的技术管理及展览会上联合宣布将为这些交易所在 2000 年到来之际保持平稳运行提供技术和服务支持。为测试解决金融机构 Y2K 问题的关键部件——HP NetServer 系统,微软 Windows NT 服务器操作系统和 MarkeNet 的咨询和集成服务,三家公司联合建立了一个测试中心。

Y2K 问题是全球各贸易系统共同面临的问题,大多数和贸易系统都需要调整以能识别以 4 位数字表示的年份,而不是仅仅以后 2 位数字表示的年份。否则,计算机就会将 2005 年解释为 1905 年,从而引起计算混乱。目前,资本市场上技术领先的供应商正努力帮助加速将交易所转向 Windows NT。根据 Towe Group 的统计,Windows NT 服务器在证券业中每年正以 100% 的速度增长,这要归功于其较低的总体拥有成本和较大的应用灵活性。不管你是在对各种不同数据服务进行集成,还是向新的技术进行转移,MarketNet 公司都能为交易所提供快速、可靠且价格合理的专业性服务,增值产品和 IT 方案。

HP 公司金融服务业务部资金市场经理 Lizzic Sipiere 先生说:"评估工作包括综合评价和技术转移计划,确保将对各项贸易活动的干扰降到最低程度。通过提供能完全与未来贸易活动保持一致的计算环境,我们能使这些贸易公司很容易解决 Y2K 问题。"微软公司的全球评券业经理 Matt Conners 先生也表示说:"我们已认识到 2000 年问题是证券业所面临的最重大的技术问题,我们承诺要帮助我们的共同用户解决这个难题,我们非常高兴能与 HP 和 MarketNet 公司合作帮助贸易公司解决他们的 Y2K 问题。"(汪)