Java 技术在图象处理中的演变

伍祥生
(湖南师范大学计算机系，长沙 410006)

摘 要 本文根据 Java 技术对图象处理所经历的三个阶段，即：Java AWT、Java 2D API 和 Java Advanced Imaging (JAI) API，较详细地论述了三个阶段各自的图象处理基础和图象处理技术；并就 JAI 的特征、图象处理功能作了较全面的阐述。

关键词 Java 技术 图象处理 演变

0 引 言

随着计算机“网络化”、“可视化”技术的发展，人们对计算机处理图象质量的要求不断提高，对独立于平台的图象的需求日益增多。为满足用户和程序员的需要，从 Java 正式推出至今四年多的时间内，对图象处理的技术已经历了三个阶段：Java AWT、Java 2D API 和 Java Advanced Imaging (JAI) API。最初推出的 Java AWT 只能处理一般的 HTML 页面，几乎不能对图象进行处理；Java 2D API 在 Java AWT 的基础上增添了一些图象处理类，可以对图象进行几何图形转变、裁剪变换、比例变换、图象旋转、对比度增强和阈值转换法等方面的处理；Java Advanced Imaging(JAI)API 进一步扩充了 Java 平台(包括 Java 2D API)，它所提供的图象处理类允许复杂、高性能的图象处理并入 Java 小应用程序(applet)和应用程序。JAI 实现图象平铺(image tiling)、兴趣范围和迟延执行等功能，它的图象处理能力超过了 JFC。JAI 的设计宗旨是：满足所有的成象应用程序的需要。现就 Java 技术对图象处理的三个阶段论述如下：

1 Java AWT 图象处理

1.1 Java AWT 图象处理的基础

Java AWT 进行图象处理，主要应用了三个接口和一个类：

(1) 接口 ImageProducer

在 Java AWT 中，凡是实现 ImageProducer 接口的类都可用来生成图象。在 java.awt.image 中提供了两个类来实现这个接口[1]，即类 FilteredImageSource 和类 MemoryImageSource。后者使用一个数组来得到图象中每个象素点的值；前者通过一个已经载入的图象和图象过滤器来生成新的图象。在 Java AWT 中，图象的生成是通过 ImageProducer 对象来完成。一旦 ImageProducer 对象生成后，就可以调用方法 CreateImage()来生成图象。每个图象包含一个 ImageProducer，当图象的大小发生改变时，该接口可以重新构造图象。

(2) 接口 ImageConsumer

ImageConsumer 适合于那些通过 ImageProducer 接口图象数据的对象。当一个消费者加入一个图象生产者时，图象生产者调用该接口所定义的方法提供图象的所有数据。ImageConsumer 定义了 9 个常量标志以判断图象的各种情况，如：COMPLETESCANLINES、IMAGEABORTED、IMAGEERRORT 和 RANDOMPIXELORDER 等等。它还定义了 6 种方法，即：imageComplete (int)、setDimensions (int, int) 、setProperties (hashtable) 和 setcolorModel() 等等。当 ImageProducer 提供源图象包含的全部象素时，ImageComplete()方法被调用。

(3) 接口 ImageObserver

ImageObserver 是一个接口。在调用某些异步执行的方法(如：类 Image 的方法 getWidth()和 getHeight()等)时，通过给它的一些参数，可监视图象生成的情况。ImageObserver 定义了 8 个常量标志以判断图象的生成情况，如：ABORT 、ERROR、ALLBITS 等等。该接口定义的方法只有 imageUpdate()。

(4) 类 Image

Image 是一个抽象类。它是所有描述图形图象

类的超类。它所要求的图象是以一个特定的平台方式获得。它定义了一些常量来判断图象描述算法等情况。它定义的方法可用来确定图象的高度、宽度、象素或图象的闪烁等。

1.2 Java AWT 图象处理

Java AWT 所能处理的图象格式为 GIF 和 JPEG 两种。它对图象的处理是由简单图象生产者(ImageProducer)和图象消费者(Imageconsumer)的过滤器模型完成。ImageProducer 是生产图象数据的线程,ImageConsumer 是消费图象数据的线程。AWT 提供图象过滤器(Image filter)来进行图象处理,过滤器可作为源图象文件的一个 ImageConsumer,过滤一幅图象的目的是获得一个 AWT 图象对象,即一幅图象的数据经图象过滤器处理之后,可得到一幅新的图象。AWT 提供的图象过滤器包括 ImageFilter 及其子类 CropImageFilter 和 RGBImageFilter。其中 ImageFilter 实现"空"处理(即不对图象数据进行处理),它是作为所有图象过滤器的父类而存在;CropImageFilter 用于提取图象中指定矩形区域的图象;RGBImageFilter 用于对图象的色彩进行处理。

在图象生产者/图象消费者模型中,容易出现下列问题:

(1) 图象数据丢失的问题

当 ImageProducer 的速度大于 ImageConsumer 时,即产生的一个图象数据放入时,前面的图象数据还没被取走,则会丢失图象数据;

(2) 图象数据重迭的问题

生产的速度小于消费的速度,当 ImageConsumer 取走一个图象数据后,再取下一个图象数据时,生产者没有产生新的图象数据,故会出现图象数据重迭。

为了消除上述问题,Java AWT 的图象生产者和消费者模型被设计为一种完全属于"进栈"(Push)的模型。ImageConsumer 从不询问图象数据,它只是等待 ImageProducer 给它"Push"数据;由方法 ImageComplete()确定图象数据的提供情况,从而确定图象的显示。

Java AWT 图象处理不适应高性能图象处理代码的发展,没有一个持久图象数据对象,图象过滤器的模型受到限制,图象数据格式只有两种,这些都不能适应用户的需求。

2 Java 2D API 图象处理

Java 2D API 是用作高级 2D 图象类的一个集合。就数字成像而言,Java 2D API 在一定程度上保持了 AWT 生产者/消费者的模型,但也增添了许多内容,例如:支持存储持久的图象数据对象、一组扩充 2D 图象的过滤器、变化范围大的图象数据形式和颜色模型等等。Java 2D API 为图象合成和 α 通道图象提供了广泛的支持,它有一组类可提供精确颜色空间定义和转换,还有丰富的面向显示的成像操作符。

2.1 Java 2D API 图象处理基础

Java 2D 这个直接方式成像 API 可分为 6 类:接口、图象数据类、图象操作类、采样模型类、颜色模型类和例外。下面介绍在直接方式成像模型中(immediate mode imaging model)起关键作用的类和接口。

(1) BufferedImage 类

BufferedImage 是直接方式成像模型(Immediate Mode Model)中的关键类。它的作用是处理存储器中的图象,还可以保存和处理来自一个文件或 URL 检索的图象数据。BufferedImage 提供了许多方法用于象素数据的存储、解释和染色等等。BufferedImage 类包括图象数据的 ColorModel 类和 Raster 类,如图 1 所示。

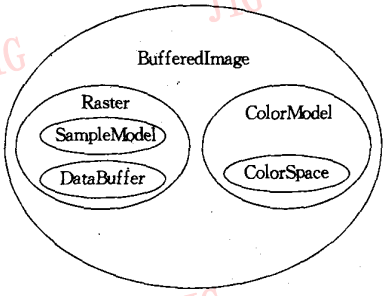


图 1 BufferedImage 类示意图

(2) Raster 类

Raster 类提供图象数据的管理,它封装了一个数据缓冲区,可用来存储采样值(sample value)和一个 SampleModel 类。它描绘图象的矩形坐标,维护了存储器中的图象数据,并为缓冲区中的单一图象数据创建多重子图象提供了机制。Raster 类也为访问图象中的特殊象素提供了一些方法。一个 Raster

对象包含两个其他的对象:DataBuffer 和 SampleModel,前者用来处理存储器中的象素数据;后者用来解释缓冲区的象素数据。

(3) 图象包提供了一些附加的类,用来定义 BufferedImage 和 Raster 对象的过滤操作。图象处理的操作封装在一个类中,即为图象操作类,它实现 BufferedImageOp 接口、RasterOp 接口或两者的接口。操作类定义了过滤器的一些方法,这些方法完成真正的图象变换。

2.2 Java 2D API 图象处理

Java 2D API 对图象的处理,需要经过仿射、比例变换、查找表变换、颜色变换、旋转等操作。Java 2D API 提供了这些操作类,并定义了过滤方法,通过这些方法,能完成对图象处理的操作。Java 2D API 图象包提供了两个接口 BufferedImageOp 和 RasterOp,用它们来定义 BufferedImage 和 Raster 对象过滤操作。实现这两个接口有下列几个类:AffineTransformOp、BandCombineOp、ColorConvertOp、ConvolveOp、LookupOp 和 RescaleOp 等等。

图象数据在计算机中是以位图的形式存在。位图是一个矩形点阵,它上面的每个点对应着一个象素。而描述每个象素需要 8 位数据,即一个字节。在图象数据中,每个字节是与画面上横着排列的 8 个象素相对应的。Java 2D API 提供了一个类 Kernel,该类是一个矩阵,用来描述一个特殊的象素和它的周围象素怎样影响过滤操作中规定象素的灰度值。另一个操作类是 ConvolveOp,它可使 Kernel 对源图象求卷积,卷积是构成大多数空间过滤算法基础的进程[2]。

3 Java Advanced Imaging 图象处理

随着计算机可视化程度的不断提高,为满足用户对图象处理更多的特殊要求,由 SUN 微系统公司、Autometric 公司、Eastman Kodak 公司和 Siemens 联合研究公司共同开发了 Java Advanced Imaging (简称 JAI)[3]。JAI 的设计方案旨在满足所有成象市场对图象各方面的需要。与其它成象方案相比较,JAI 为应用程序的开发者提供了极大的方便。它的特征如下所述。

3.1 JAI 的特征

归纳起来,JAI 的特征有以下几个方面:

(1) 跨平台成像

鉴于大多数成像的 API 是为一个特定的操作系统而设计的,JAI 允许 Java 运行时库模型提供平台独立性。只要具备 Java 虚拟机的计算机,就可以运行 JAI 的应用程序。JAI 是一个真正的跨平台的成像 API,这就意味着,只要编写一次 JAI 的应用程序,它将可以到处运行。

(2) 分布式成像

JAI 通过 Java 平台的网络体系结构和远程实现技术的方式,也能很好地适合于客户机/服务器成像,远程实现是基于 JavaRMI(远程方法调用),JavaRMI 允许客户机上的 Java 代码通过调用方法访问远方计算机的对象。

(3) 面向对象的 API

如同 Java 一样,JAI 完全是面向对象的。在 JAI 中,图象和图象处理操作都定义为对象。

(4) 灵活性和可扩充性

任何成像 API 必须支持一些基本的成像技术,例如:图象获取和显示、基本变换、增强、几何变换、分析和压缩等等。JAI 提供了一组运算符支持基本的成像技术,还提供了一个可扩充的框架用于特殊图象处理的需要,它允许特殊的图象处理方案加入核心 API。JAI 也提供了一组标准的图象压缩和解压缩的方法。

(5) 设备独立性能

在 JAI 中,根据需要可以将图象处理指定在设备独立的坐标中。JAI 的"renderable"方式把所有的图象源作为是染色独立的(rendering-independent)。

(6) 鲁棒性

JAI 支持复杂的图象格式,包括上至三维和任意频带数的图象,它直接支持许多成像算法。JAI 实现一组核心图象处理能力,包括图象平铺、兴趣范围和迟延执行;该 API 也实现一组核心图象处理操作符,包括:许多公共点、面积和频率域操作。

(7) 高性能

JAI 可进行各种各样的实现,包括最优化的实现,即能获得硬件加速和平台介质能力的最优化。

(8) 互用性

JAI 与 Java 介质 API 并入一起,使得丰富的介质应用程序能展现在 Java 平台上。JAI 与其它的 Java API 工作得很好。JAI 是一个 Java 介质 API,它是典型地作为一个扩充到 Java 平台。JAI 提供的成像功能超出了 JAVA FOUNDATION CLASSES (JFC)所提供的成像功能,不过在大多数场合,这些

类具有兼容性。

3.2 JAI 图象处理基础

在 JAI 图象处理过程中,主要起作用的是下面的接口和类。

(1) RenderableImage 接口

RenderableImage 是一个用于染色独立(rendering-independent)图象的公共接口,其格式为:

public abstract interface RenderableImage

一个 RenderableImage 由 CreateRendering() 方法可返回一个 RenderedImage,RenderedImage 的构成由 RenderContext 来确定。

(2) 类 RenderableImageOp

类 RenderableImageOp 的实例化操作,需要运用相关联的一个 ContextualRenderedImageFactory 的帮助。ContextualRenderedImageFactory 提供了一个接口功能程序,在两个 RenderableImageOp 实例之间有不相同的接口功能程序。因此,对 RenderableImage 的不同操作,可以通过一个单独的类的多重实例的使用来完成,例如:使用类 RenderableImageOp 通过 ContextualRenderedImageFactory 来实现。ContextualRenderedImageFactory 的名字通常缩短为"CRIF"。所有使用一个染色独立(rendering-independent)链中的操作必须实现"CRIF",实现这个接口的类必须提供一个构造函数。

(3) RenderedOp

RenderedOp 为 JAI 中染色成像链的一个节点,它存储一个操作名(作为一个字符串)、一个表示源程序和各种参数的参数块(ParameterBlock)和一个包含解释提示的查找表。一组节点通过它们的参数块里面的源程序向量可连接在一起形成一个 DAG(受控非循环图),该图的拓扑结构(例如:连接性)可由改变参数块来改动;操作名、参数和解释提示也可以改变。

(4) 类 TiledImage

类 TiledImage 是接口 WritableRenderedImage 的具体实现。它是 JAI 中可写图象主要的类。TiledImage 提供一个直接实现 WritableRenderedImage 的接口,它使用该接口的功能来描述图象的多重平铺(Tiles)。一个 WritableRenderedImage 的平铺(Tiles)必须共享一个 SampleModel,因为 SampleModel 可以确定它们的宽、高和象素格式。平铺(Tiles)形成一个正规的网格,该网格可占用平面的任何矩形范围,超过图象状态边界的平铺(Tiles)象素有不确定的值。由 set() 方法提供单一 PlamarImage 源程序可以定义一个 TiledImage 的内容。修改 TiledImage 内容的方式有多种,例如:set() 方法提供一种方式(例如:使用一个软边缘掩码)有选择性地覆盖一部分 TiledImage;调入方法 CreateGraphics()也可以修改 TiledImage 的内容。

3.3 JAI 图象处理

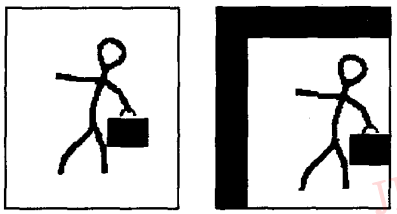
JAI 对图象处理有多种功能,其中几何变换包括平移、比例、旋转和仿射等。在此仅从 JAI 的几何变换阐述如下:

(1) 平移变换

图象平移是指一幅图象向上、下、左或右方向的空间移动。JAI 中的平移操作需要 3 个参数,即:X 方向的位移、Y 方向的位移和一个插值(interpolation)。图 2 是使用最邻近(nearest-neighbor)插值法的一个平移操作例子。

(2) 比例变换

JAI 中的比例变换是对一个图象进行放大或缩小的操作。这种比例变换操作需要 5 个参数,即:X 比例因子、Y 比例因子、X 平移、Y 平移和插值。图 3 是利用最邻近插值法所进行的无平移的比例操作。



(a)平移前的图象 (b)x、y 位移量为正值的图象平移

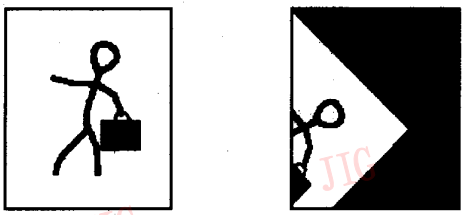
图 2 平移操作示意图

(a) 变换前的原图象 (b)X,Y 为 1.2 的比例变换 (c)$X=0.8$,$Y=1.0$ 的比例变换

图 3 比例操作示意图

(3) 旋转变换

JAI 中进行旋转操作需要 4 个参数,即:旋转的 X 原点、Y 原点、径向旋转角和插值。图 4 给出一个旋转操作的例子。

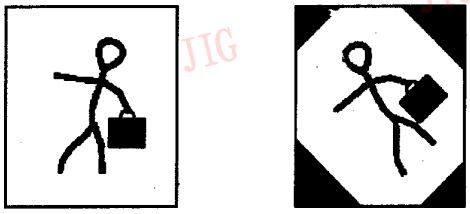


(a) 旋转前的原图象　　(b) 绕 $X=0, Y=0$ 旋转 45°

图 4　旋转操作示意图

(4) 仿射变换

JAI 中的仿射变换需要两个参数,即:转换矩阵和插值。图 5 是一个仿射操作的实例,即绕中心逆时针旋转 45°的仿射操作。



(a) 仿射变换前的图象　　(b) 绕中心逆时针旋转 45° 的仿射变换

图 5　仿射操作示意图

其程序段如下:

```
// 下载图象。
String filename="images/Trees.gif";
PlanarImage im=(PlanarImage)JAI.create
                ("fileload", filename);
// 创建仿射转换矩阵。
AffineTransform tr=new AffineTransform
                (0.707107,-0.707106,
                0.707106,0.707106,
                0.0,0.0);
// 指定插值类型。
Interpolation interp=new InterpolationNearest();
// 创建仿射操作。
PlanarImage im2=(PlanarImage)JAI.create
                ("affine", im, tr,interp);
```

4　结束语

JAI 图象处理的功能还有很多,下面列举其中的部分介绍。

(1) JAI 中实现透视变换

在图象处理中,透视失真可应用一个透视变换修正。JAI 实现透视变换是使用 perspectiveTransform 类。该类包括许多方法,通过这些方法可以实现透视变换[4]。

(2) JAI 对图象处理还可以进行转置、裁剪、弯曲(Warping)等变换。

(3) JAI 对图象处理包括许多操作符,如:兴趣控制范围(Region of interest control)、相关操作符、逻辑操作符、算术操作符、抖动和箝位(Clamping)象素值等等。简述几个如下:

① JAI 支持两种不同类型的 ROI(region of interest)掩码,一个是布尔掩码,另一个是阈值。ROI 掩码可以控制被处理的源图象象素和被恢复的目标图象象素。它的实现是使用 ROI 类和 ROIshap 类。

② JAI 支持 monodic、dyadic 和 Unary 逻辑操作。monodic 逻辑操作包括:一幅源图象和一个常量生成一个目标图象之间的象素到象素的 AND、OR 和 XOR 操作;dyadic 逻辑操作是指:两幅源图象生成一幅目标图象之间的象素到象素的 AND、OR 和 XOR 操作;Unary 逻辑操作是一个 NOT 操作。

③ JAI 提供两个操作用于抖动一幅图象,即顺序抖动(ordered dither)和误差扩散(error-diffusion)抖动,抖动的选择取决于期望速度(desired speed)和图象质量。

④ JAI API 图象增强技术包括:增加边界、裁剪图象、振幅重新调节、查找表修改、中值过滤、频率范围处理和象素点处理等等。

参 考 文 献

1　王克宏主编. Java 语言编程技术. 北京:清华大学出版社.

2　伍祥生,王克宏,Java 2D API 技术及其实现方法. 中国图象图形学报,1998,3(8).

3　//:java.sun.com/products/java-media/jai/;

4　Sun Microsystems. Programming in Java Advanced Imaging, Release 1.0. July 1999.