基于 JAVA 平台的图像处理程序的设计与实现

付 豪，ZY1601119

摘 要：本文基于 Java 平台实现了图像处理程序的设计与编程。该程序具有打开一副图像，进行直方图均衡，将灰度线性变换，将灰度拉伸的功能。本文重点描述了图像读取、灰度化处理、直方图均衡、灰度线性变换及直方图拉伸的原理与源码设计。

1 程序设计与实现

1.1 整体架构设计

本文利用Java进行了图像处理程序的实现。目的在于实现打开一副图像，进行直方图均衡，将灰度线性变换，将灰度拉伸。程序目录结构如图1所示。其中程序源码各类功能如下：

GrayImage：输入RBG图像，进行图像信号的转换，进行灰度化处理；

GrayStretchImg：进行灰度修正中的分段线性变换，即灰度拉伸；

HistogramEqualizationImg：进行直方图均衡；

ImageProcessing：提供用户界面，进行输入输出；

LineTranImg：进行灰度修正中的灰度线性变换；

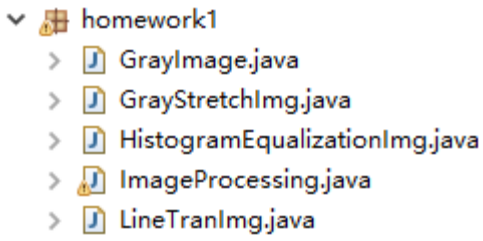

图 1 程序的目录结构

1.2 图片读取原理

本程序由文件系统读取图片的方法readFile()如图2（a）所示，利用的库如图2（b）所示。

(a)
```
vate static void readFile() throws IOException {
    String filePath;
    System.out.println("请输入文件路径inFile:");
    Scanner in = new Scanner(System.in);
    filePath = in.nextLine();
    File input = new File(filePath);
    image = ImageIO.read(input);
}
```

(b)
```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
```

图 2 图片读取的实现

（a）读取图片的源代码实现；（b）读取图片利用的库

如图2（a）所示，本程序图片读取的实现主要利用了Java的ImageIO的read方法，其首先会从当前已注册的ImageReader中自动选择一个去解码输入的File，之后会返回形成的BufferedImage类型的对象。该输入的File包装在ImageInputStream中。

1.2 灰度化处理原理

本程序的灰度化处理主要由GrayImage类中的changeToGray()方法实现，其源码如图3所示。

```
private void changeToGray() {
    grayImg = new BufferedImage(imgWidth, imgHeight, BufferedImage.TYPE_3BYTE_BGR);
    for (int i = 0; i < imgWidth; i++) {
        for (int j = 0; j < imgHeight; j++) {
            int rgb = image.getRGB(i, j);
            int blue = (rgb & 0x000000FF) >> 0;
            int green = (rgb & 0x0000FF00) >> 8;
            int red = (rgb & 0x00FF0000) >> 16;
            int gray = (int) (red * 0.3 + green * 0.59 + blue * 0.11);
            rgb = (255 << 24) | (gray << 16) | (gray << 8) | (gray);
            grayImg.setRGB(i, j, rgb);
        }
    }
}
```

图 3 灰度化处理的源码实现

如图3所示，本文灰度化处理的主要原理为YUV的颜色空间中，Y的分量的物理意义是点的亮度，由该值反映亮度等级，而根据RGB和YUV颜色空间的变化关系可建立亮度Y与R、G、B三个颜色分量的对应关系式，基于下式表达图像的灰度值：

Y=0.3R+0.59G+0.11B

其中：Y为亮度，R为红色分量，G为绿色分量，B为蓝色分量。

值得注意的是，在读取与写入RGB数值时，蓝色分量存放在三字节中的第一个字节，绿色分量存放在第二个字节，红色分量存放在第三个字节。

1.3 直方图均衡原理

本程序的直方图均衡主要由HistogramEqualizationImg类中的histogramEqualization()方法实现，其源码如图4所示。

```
private void histogramEqualization() throws InterruptedException {
    int[] histogram = new int[256];
    int[] pixels = new int[imgWidth * imgHeight];
    hisEqualImg = new BufferedImage(imgWidth, imgHeight, BufferedImage.TYPE_3BYTE_BGR);
    PixelGrabber pg = new PixelGrabber(image, 0, 0, imgWidth, imgHeight, pixels, 0, imgWidth);
    pg.grabPixels();
    for (int i = 0; i < imgHeight - 1; i++) {
        for (int j = 0; j < imgWidth - 1; j++) {
            int gray = pixels[i * imgWidth + j] & 0xFF;
            histogram[gray]++;
        }
    }
    double divPiNum = (double) 255 / (imgWidth * imgHeight);
    double[] Sk = new double[256];
    Sk[0] = (double) histogram[0] * divPiNum;
    for (int i = 1; i < 256; i++) {
        Sk[i] = Sk[i - 1] + histogram[i] * divPiNum;
    }
    for (int i = 0; i < imgHeight; i++) {
        for (int j = 0; j < imgWidth; j++) {
            int gray = pixels[i * imgWidth + j] & 0xFF;
            int newGray = ((int) Sk[gray]);
            int rgb = (255 << 24) | (newGray << 16) | (newGray << 8) | (newGray);
            hisEqualImg.setRGB(j, i, rgb);
        }
    }
}
```

图4 直方图均衡的源码实现

在建立直方图histogram时，利用创建PixelGrabber对象，将image图像中的像素抓取到pixels数组中。即像素(j,i)的RGB数据存储在pixels[(i-0)*imgWidth+(j-0)]中。

数组Sk[i]存放原灰度级i应变为的新灰度级k，其计算公式如下式：

$$Sk[i] = \sum_{j=0}^{i} \frac{n_j}{N} \times 255$$

其中：n_j为原图像第j级灰度出现的次数，N为原图像像素总数，$Sk[i]$为原灰度级i的像素应变为的新灰度级k。

1.4 灰度线性变换原理

本程序的灰度线性变换主要由LineTranImg类中的lineTrans()方法实现，其源码如图5所示。

```
private void lineTrans() throws InterruptedException {
    int[] pixels = new int[imgWidth * imgHeight];
    int[] histogram = new int[256];
    lineTranImg = new BufferedImage(imgWidth, imgHeight, BufferedImage.TYPE_3BYTE_BGR);
    PixelGrabber pg = new PixelGrabber(image, 0, 0, imgWidth, imgHeight, pixels, 0, imgWidth);
    pg.grabPixels();
    for (int i = 0; i < imgHeight - 1; i++) {
        for (int j = 0; j < imgWidth - 1; j++) {
            int gray = pixels[i * imgWidth + j] & 0xFF;
            histogram[gray]++;
        }
    }
    int a = 255;
    int b = 0;
    for (int i = 0; i < 256; i++) {
        if (histogram[i] > 0) {
            if (i < a) {
                a = i;
            }
            if (i > b) {
                b = i;
            }
        }
    }
    double k = (double) (d - c) / (b - a);
    for (int i = 0; i < imgHeight; i++) {
        for (int j = 0; j < imgWidth; j++) {
            int gray = pixels[i * imgWidth + j] & 0xFF;
            int rgb;
            gray = (int) (k * (gray - a) + c);
            rgb = (255 << 24) | (gray << 16) | (gray << 8) | (gray);
            lineTranImg.setRGB(j, i, rgb);
        }
    }
}
```

图5 灰度线性变换的源码实现

如图5所示，灰度线性变换公式如下：

$$g(x,y) = \frac{d-c}{b-a}(f(x,y)-a)+c$$

其中：输入图像f(x,y)的灰度范围为[a,b]，变换后的输出图像g(x,y)的灰度范围变为[c,d]。

1.5 灰度拉伸原理

本程序的灰度拉伸主要由GrayStretchImg类中的grayStretch()方法实现，其源码如图6所示。

```
private void grayStretch() throws InterruptedException {
    int[] pixels = new int[imgWidth * imgHeight];
    grayStrImg = new BufferedImage(imgWidth, imgHeight, BufferedImage.TYPE_3BYTE_BGR);
    PixelGrabber pg = new PixelGrabber(image, 0, 0, imgWidth, imgHeight, pixels, 0, imgWidth);
    pg.grabPixels();
    for (int i = 0; i < imgHeight; i++) {
        for (int j = 0; j < imgWidth; j++) {
            int gray = pixels[i * imgWidth + j] & 0xFF;
            int rgb;
            double k1, k2, k3;
            k1 = (double) c / a;
            k2 = (double) (d - c) / (b - a);
            k3 = (double) (255 - d) / (255 - b);
            if (gray < a) {
                gray = (int) (k1 * gray);
            } else if (gray >= a && gray < b) {
                gray = (int) (k2 * (gray - a) + c);
            } else if (gray >= b) {
                gray = (int) (k3 * (gray - b) + d);
            }
            rgb = (255 << 24) | (gray << 16) | (gray << 8) | (gray);
            grayStrImg.setRGB(j, i, rgb);
        }
    }
}
```

图6 灰度拉伸的源码实现

如图6所示，灰度拉伸公式如下：

$$g(x,y) = \begin{cases} \frac{c}{a} f(x,y), 0 \le f(x,y) < a \\ \frac{d-c}{b-a}(f(x,y)-a)+c, a \le f(x,y) < b \\ \frac{255-d}{255-b}(f(x,y)-b)+d, b \le f(x,y) \le 255 \end{cases}$$

其中：输入图像f(x,y)的感兴趣的灰度区间为[a,b]，变换后的输出图像g(x,y)中感兴趣的灰度区间拉伸后灰度范围变为[c,d]。

2 实验结果

2.1 灰度化处理结果展示

灰度化处理前后图像及明度直方图如图 7 所示。

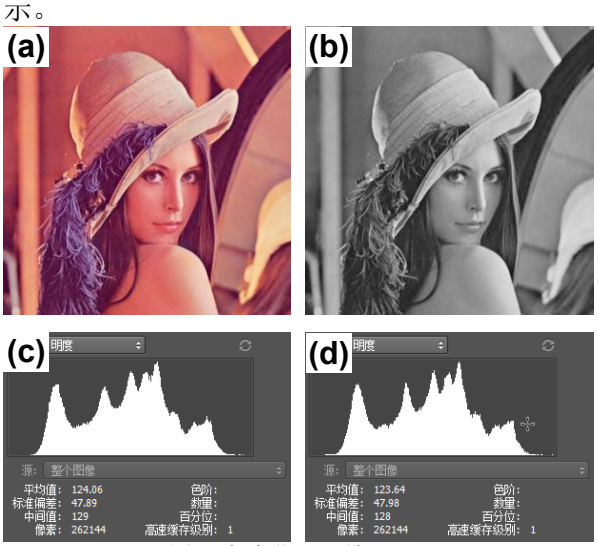

图 7 灰度化处理结果展示

（a）原图像；（b）处理后图像；（c）原图像明度直方图；（d）处理后图像明度直方图

2.2 直方图均衡结果展示

直方图均衡前后图像及直方图如图 8 所示。

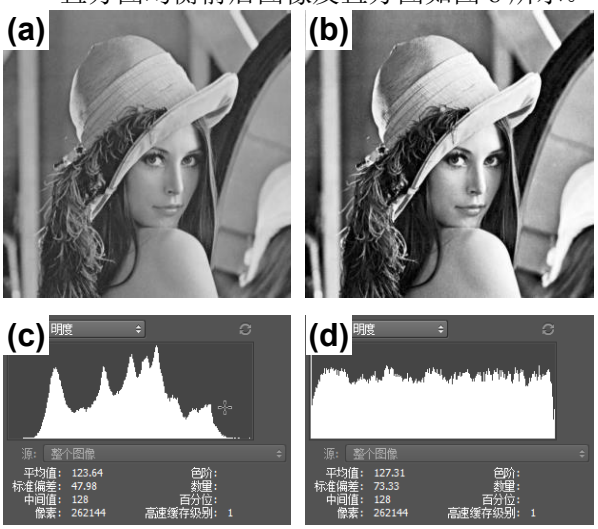

图 8 直方图均衡结果展示

（a）原图像；（b）处理后图像；（c）原图像直方图；（d）处理后图像直方图

2.3 灰度线性变换结果展示

设 c=0，d=128，灰度线性变换前后图像及直方图如图 9 所示。

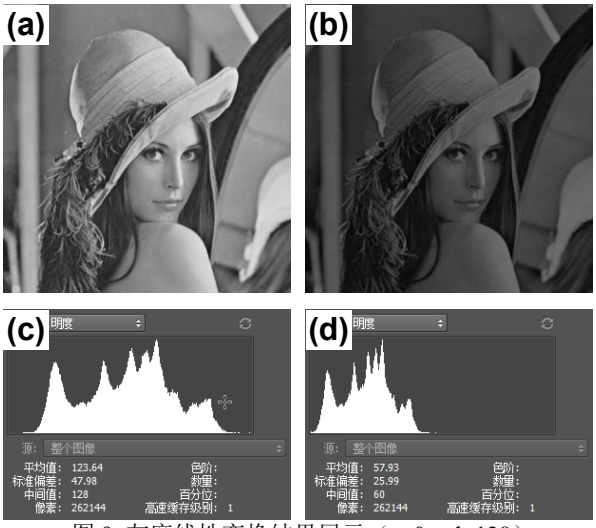

图 9 灰度线性变换结果展示（c=0，d=128）
（a）原图像；（b）处理后图像；（c）原图像直方图；
（d）处理后图像直方图

2.4 灰度拉伸结果展示

设 a=100，b=200，c=50，d=250，灰度拉伸前后图像及直方图如图 10 所示。

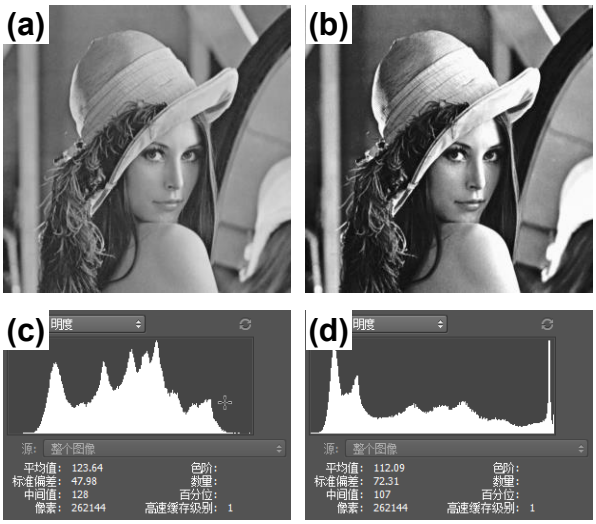

图 10 灰度拉伸结果展示（a=100，b=200，c=50，
d=250）
（a）原图像；（b）处理后图像；（c）原图像直方图；
（d）处理后图像直方图

3 结 论

（1）本文基于 Java 平台实现了图像处理程序的设计与编程，该程序具有打开一副图像，进行直方图均衡，将灰度线性变换，将灰度拉伸的功能。

（2）本文重点描述了图像读取、灰度化处理、直方图均衡、灰度线性变换及直方图拉伸的原理与源码设计。