三种基于GDI+的图像灰度化实现方法

李贞培[1],李 平[2],郭新宇[2],陈树敏[2]
(1.西北工业大学 自动化学院,陕西 西安 710072;
2.辽宁石油化工大学 信息与控制工程学院,辽宁 抚顺 113001)

摘 要:随着数字图像处理对于快速性、实时性的要求越来越高,寻求快速有效的图像灰度化处理方法成为急待解决的问题。介绍了三种基于GDI+的图像灰度化实现方法:直接读写像素法、色彩变换法和直接读写内存图像数据法。并对这三种方法进行了性能、实现复杂度等方面的比较。可以得出:直接读写像素法处理过程缓慢,不能满足实际图像处理需要;色彩变换法处理速度快,处理速度是直接读写像素法处理速度的60倍以上;直接读写内存图像数据法处理速度最快,处理速度是色彩变换法处理速度的2.3倍左右,可以满足大部分情况下的实际图像处理需要。

关键词:GDI+;灰度化;实现方法;性能
中图分类号:TP391.41 **文献标识码**:A **文章编号**:1673-629X(2009)07-0073-03

Three Programming Methods of Gray Processing Based on GDI+

LI Zhen-pei[1], LI Ping[2], GUO Xin-yu[2], CHEN Shu-min[2]
(1. School of Automation, Northwestern Polytechnical University, Xi'an 710072, China;
2. School of Information and Control Engineering, Liaoning Shihua University, Fushun 113001, China)

Abstract: With the ever-higher demand of speed for digital image processing, it's urgent to look for fast and effective method for image gray processing. Introduces three programming method for gray processing based on GDI+: the method by directly reading/writing pixel data, the method by transformational color matrix, and the method by directly reading/writing image data in memory. The performance and programming complexity of those method is compared. The major conclusion is: the speed of processing by method one is very slow, can't meet the demand of practical image processing; the speed of processing by method two is more than 60 times faster than method one's; method three is the fatest one, its speed is about 2.3 times faster than method two's, the gray processing by method three can meet the demand of practical image processing in most cases.

Key words: GDI+; gray processing; programming method; performance

0 引 言

灰度图是指只含亮度信息,不含色彩信息的图像,广泛应用于图像模式识别、图像分割、图像增强等数字图像处理的各个领域。

灰度化处理是把含有亮度和色彩的彩色图像变换成灰度图像的过程。灰度化处理的结果是许多后续图像处理的基础。随着现代信息技术的快速发展,对数字图像处理的快速性、实时性要求越来越高,所以,寻求一种快速有效的灰度化处理方法具有极其重要的意义[1~3]。

收稿日期:2008-09-29;**修回日期**:2008-12-08
基金项目:中石化集团公司科研资助项目(305051)
作者简介:李贞培(1979-),男(苗族),贵州三都人,博士研究生,研究方向为工业过程先进控制与优化、油气储运自动化;李 平,教授,博士生导师,研究方向为工业过程先进控制与优化。

1 GDI+

GDI+[4]是GDI(Windows早期版本提供的图形设备接口)的后续版本,是Microsoft Windows XP操作系统及后续版本的图形引擎。GDI+是一个应用编程接口,通过一组C++类来提供接口的功能。出于兼容性考虑,Windows XP仍然支持以前版本的GDI,但是在开发新应用程序的时候,开发人员为了满足图形图像处理需要应该使用GDI+,因为GDI+对以前的Windows版本中GDI进行了优化,并添加了许多新的功能[5,6]。

作为图形设备接口的GDI+使得应用程序开发人员在输出屏幕和打印机信息的时候无需考虑具体显示设备的细节,开发人员只需调用GDI+库的类的一些方法即可完成图形操作,真正的绘图工作由这些方法交给特定的设备驱动程序来完成,GDI+使得图形硬件和应用程序相互隔离,从而使开发人员编写设备无

关的应用程序变得容易[5,7]。

使用 GDI + 时,不必像使用 GDI 那样关心句柄和设备环境这样的概念,只需创建 Graphics(图形)对象,然后调用 Graphics 的成员函数,就可以完成大部分的绘图或文本输出操作,Graphics 类是 GDI + 的核心。以下代码通过 Graphics 对象将一幅位图绘制到 PictureBox 控件的表面:

```
Bitmap bm = (Bitmap) Image. FromFile("d: \ \ Water lilies. bmp");
Graphics g = pictureBox1. CreateGraphics();
g. DrawImage(bm, 0, 0);
```

从以上代码可以看到,基于 GDI + 的图像处理不必再像以往那样拘泥于种类繁多的图像文件格式及其数据存储结构和相应的处理方法,也不必再像以往那样进行必不可少的复杂的设备环境操作,而且变得更加简明、灵活、高效。

在 GDI + 中还新增了许多新的对象,如 Bitmap 类,该类封装了位图操作的常用功能,使图像处理变得直观、简洁而功能更加强大;新增的矩阵对象使得开发人员不仅能够通过坐标变换实现图形图像的变形、旋转、平移等,而且还能够实现色彩的变换,这为图像色彩的处理提供了新的途径。

2 基于 GDI + 的图像灰度化实现方法

灰度化就是使彩色的 R,G,B 分量值相等的过程。灰度化算法[8] 主要有以下三种:

(1) 最大值法。这种方法使 R,G,B 分量值均等于三个值中最大的一个,即

$$R = G = B = \max(R,G,B)$$

最大值法会形成亮度值很高的灰度图像。

(2) 平均值法。这种方法使 R,G,B 分量值均等于三个值的平均值,即

$$R = G = B = (R + G + B)/3$$

平均值法会形成较柔和的灰度图像。

(3) 加权平均值法。这种方法根据重要性或其它指标给 R,G,B 各分量赋予不同的权值,并使 R,G,B 的值等于它们的加权平均值,即

$$R = G = B = (W_rR + W_gG + W_bB)/3$$

式中 W_r,W_g,W_b 分别是 R,G,B 的权值。当 W_r,W_g,W_b 取不同的值时,将得到不同灰度的灰度图像。实验表明,当 $W_r = 0.299, W_g = 0.587, W_b = 0.114$ 时,能获得最符合人眼视觉感受的灰度图像。因此,文中的灰度化算法采用加权平均值法。

2.1 直接读写像素法

GDI + 的 Image 类封装了对 BMP, GIF, JPEG, PNG, TIFF, WMF(Windows 元文件)和 EMF(增强 WMF)图像文件的调入、格式转换以及简单处理的功能。而 Bitmap 类是从 Image 类继承的一个图像类,它封装了 Windows 位图操作的常用功能。Bitmap 类的 GetPixel 方法可获得一个像素的颜色值,SetPixel 可设置一个像素的颜色值。

本实现方法的思路比较直观,循环遍历每个像素,通过 Bitmap 类的 GetPixel 方法获得像素的 RGB 值,经加权计算后通过 SetPixel 方法设置该像素的灰度值。笔者将实现过程封装成一个函数,以利于代码重用,实现函数如下,其中参数 bm 为待灰度化的 Bitmap 类对象指针:

```
void GrayScaleBySetPixel(Bitmap * bm)
{
  int width = bm - >GetWidth();
  int height = bm - >GetHeight();
  Color oldColor;
  BYTE r = 0,g = 0,b = 0;
  int pix = 0;
  int i = 0, j = 0;
  for(i = 0; i < height; i + + )
  {
    for (j = 0; j < width; j + + )
    {
      bm - >GetPixel(j,i,&oldColor);
      r = g = b = (BYTE)(.299 * oldColor. GetR() + .587 * oldColor. GetG() + .114 * oldColor. GetB());
      bm - >SetPixel(j,i,Color(r,g,b));
    }
  }
}
```

经过验证,一幅 800 * 600 的 24 色位图,采用此方法的灰度化过程耗时 6.625 秒,处理速度非常慢,不能满足实际图像处理需要。

2.2 色彩变换法

对于一个二维坐标,可以通过变换矩阵来对其进行坐标变换。通过变换矩阵来进行坐标位置变换的原理,同样也可以运用于色彩的变换。一个色彩信息可以表示为一个四维向量的形式(Red, Green, Blue, Alpha),同样,可以使用如图 1 所示的 4 阶色彩变换矩阵来修改色彩的每一个分量值。

$$\begin{vmatrix} red & 0 & 0 & 0 \\ 0 & green & 0 & 0 \\ 0 & 0 & blue & 0 \\ 0 & 0 & 0 & alpha \end{vmatrix}$$

图 1 色彩变换矩阵

使用 4 阶的色彩变换矩阵来修改色彩,只能够对

色彩的每一个分量值进行乘除运算(线性变换),如果还要对这些分量值进行加减运行(即进行仿射变换),只能通过 5 阶矩阵来完成,可通过在 4 阶色彩变换矩阵上增加一个哑元坐标(通常为 1)来构造 5 阶色彩变换矩阵,这样如果修改矩阵主对角线的值,可对 R,G,B 分量进行乘法操作;修改矩阵其它值,可对 R,G,B 分量进行加减操作[7]。对于灰度化,变换矩阵为图 2 所示矩阵。

$$\begin{vmatrix} 0.299 & 0.299 & 0.299 & 0 & 0 \\ 0.587 & 0.587 & 0.587 & 0 & 0 \\ 0.114 & 0.114 & 0.114 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix}$$

图 2　灰度化色彩变换矩阵

在 GDI+中,这个 5 阶色彩变换矩阵用 ColorMatrix 结构来表示,ColorMatrix 实际是一个数组,定义为 struct ColorMatrix{REAL m[5]}。在 GDI+中实现色彩变换的一般步骤是:

(1)初始化色彩变换矩阵 ColorMatrix 结构;

(2)实例化 ImageAttribtues(图像属性)类的一个对象,使用该类的方法 SetColorMatrix 设置色彩变换矩阵;

(3)由位图创建 Graphics 对象,即绘图表面就是位图本身,然后将 ImageAttribtues 对象作为参数传递给对 Graphics 类的 DrawImage 方法,实现位图色彩的变换。

将 ColorMatrix 结构设为如图 2 所示色彩变换矩阵,可实现彩色位图的灰度化,实现函数如下:

```
void GrayScaleByColorMatrix(Bitmap * bm)
{
  ColorMatrix colorMatrix=
  {0.299f,0.299f,0.299f,0,0,
    0.587f,0.587f,0.587f,0,0,
    0.114f,0.114f,0.114f,0,0,
    0,0,0,1,0,
    0,0,0,0,1
  };
    ImageAttributes attr;
    attr.SetColorMatrix(&colorMatrix);
    Graphics g(bm);
    Rect destRect(0, 0, bm->GetWidth(),bm->GetHeight
());
    g.DrawImage(bm, destRect, 0, 0,bm->GetWidth(), bm
->GetHeight(), UnitPixel, &attr);
}
```

经过验证,一幅 800 * 600 的 24 色位图,采用此方法的灰度化过程耗时 0.109375 秒,处理速度快,可以满足一定速度要求范围内的实际图像处理需要。

2.3　直接读写内存图像数据法

GDI+的 Bitmap 类提供 LockBits 方法,该方法将位图特定大小的矩形区域锁定在内存中,并返回一个 BitmapData 类对象,BitmapData 指定了位图锁定区域的属性,如大小、像素格式、像素数据在内存中的起始地址以及每个扫描行的长度(步幅)等。BitmapData 的成员 Scan0 指示了像素数据在内存中的起始地址,获得了像素数据在内存中的起始地址,就可以直接读写像素数据,因每相邻三个字节代表一个像素的 R,G,B 分量(不含每个扫描行的补 0 字节),所以以相邻三个字节为单位,经加权计算后依次设置每个字节的新值。需要注意的是,在 GDI+中,像素 R,G,B 分量的字节排列顺序是 $B、G、R$,而不是 $R、G、B$;同时注意读写像素字节的时候要跳过每个扫描行的补 0 字节。

该方法实现函数如下:

```
bool GrayScaleByLockData(Bitmap * bm)
{
  int width=bm->GetWidth();
  int height=bm->GetHeight();
  Rect rect(0,0,width,height);
  BitmapData data;
  Status status;
    status=bm->LockBits(&rect, ImageLockModeWrite, bm
->GetPixelFormat(),&data);
    if(status! =0)
    {
      bm->UnlockBits(&data);
      return false;
    }
    byte * p=(byte * )data.Scan0;
    int offset=data.Stride-width * 3;
    int i=0, j=0;
    BYTE r, g, b;
    for (i=0; i<height; i++)
    {
      for (j=0; j<width; j++)
      {
        b=p[0];
        g=p[1];
        r=p[2];
        p[0]=p[1]=p[2]=(BYTE)(.299 * r+.587 * g+.
114 * b);
        p+=3;
      }
      p+=offset;
    }
```

(下转第 79 页)

因此读入内存的数据可以被几个学习器所共用，这样用于读取数据部分的时间增加的并不会太大，并且由于此集成算法是类似投票算法的简单算法，所以运行时间增加也是可以接受的，而且不像大多识别拒识样本算法那样花费大量时间在判断样本领域上（因为本算法每个样本只判断三个领域）。

从表 3 也可以看出，总起来说三个个体学习器的集成算法中，个体学习器学习加上集成总的执行时间不超过个体学习器时间的二倍。从上面的实验结果及分析可以看出，集成算法无论在效果还是在执行时间上都是可行有效的。

5　结束语

基于集成学习的覆盖算法是通过对领域覆盖算法中拒识样本的处理，仅使用三个学习器就可以使算法性能有了很好的提升，它提供了一种增加领域覆盖算法泛化能力的方法。集成学习和覆盖算法结合是一种新的尝试，今后可进一步研究覆盖算法的改进及其与集成学习的结合。

参考文献：

[1] 张　铃，张　钹. M－P 神经元的几何意义及其应用[J]. 软件学报，1998，9(5)：334－338.

[2] 张　铃，张　钹. 前向神经网络设计问题的回顾和探索[J]. 计算机工程和科学，1998，20(4)：1－10.

[3] 张　铃，张　钹. 多层前向网络的交叉覆盖设计算法[J]. 软件学报，1999，10(7)：337－342.

[4] 吴　涛，张　铃. 机器学习中的核覆盖算法[J]. 计算机学报，2005，28(8)：1295－1301.

[5] 张燕平，张　铃. 机器学习中的多侧面递进算法 MIDA[J]. 电子学报，2005，33(2)：327－331.

[6] 张　铃，吴　涛. 覆盖算法的概率模型[J]. 软件学报，2007，18(11)：2691－2699.

[7] 王倩倩，段　震，张燕平. 基于交叉覆盖算法的文本分类[J]. 计算机技术与发展，2007，17(6)：113－115.

[8] 汪小寒，陈　洁. 基于核覆盖算法的煤价预测[J]. 计算机技术与发展，2006，16(12)：81－85.

[9] 张燕平，张　铃. 构造性核覆盖算法在图像识别中的应用[J]. 中国图像图形学报，2004，14(9)：1304－1308.

[10] 唐　伟，周志华. 基于 Bagging 的选择性聚类集成[J]. 软件学报，2005，16(4)：496－502.

[11] 俞　扬，周志华. 集成学习中完全随机学习策略研究[J]. 计算机工程，2006，32(17)：100－102.

[12] 梁英毅. 集成学习综述[EB/OL]. 2006. http://soft.cs.tsinghua.edu.cn/keltin/docs/ensemble.pdf.

[13] Garcia－Pedrajas N. Nonlinear Boosting Projections for Ensemble Construction[J]. Journal of Machine Learning Research，2007(8)：1－33.

[14] 周志华. 通过集成学习进行知识获取[J]. 重庆邮电大学学报，2008，20(3)：361－362.

（上接第 75 页）

```
    bm->UnlockBits(&data);
    return true;
}
```

经过验证，一幅 800 * 600 的 24 色位图，采用此方法的灰度化过程耗时 0.046875 秒，处理速度非常快，可以满足大部分情况下的实际图像处理需要。

3　结束语

介绍了三种基于 GDI＋的图像灰度化实现方法，进行了性能、实现复杂度等方面的比较。结论是直接读写内存图像数据法处理速度最快，但编码稍微繁琐；利用 GDI＋的色彩变换方法处理速度快，但耗时比读写内存图像数据法多大约 1 倍，编码简明；直接读写像素法处理速度非常慢，不能满足实际数字图像处理需要。

介绍的基于 GDI＋的图像灰度化实现方法和比较结论，可为开发人员的实际开发应用提供借鉴和参考。代码在 Visual Studio C＋＋.net 2005，Windows Xp 环境下调试编译通过。

参考文献：

[1] Pratt W K. Digital Image Processing[M]. [s.l.]: John Wiley & Sons，1991.

[2] Gonzalez R C，Woods R E. Digital Image Processing[M]. [s.l.]: Prentice Hall Press，2007.

[3] 彭召意. 16 位位图的灰度化处理方法[J]. 中国包装工业，2002(6)：150－151.

[4] Microsoft Corporation. Microsoft Developer Network(MSDN)[EB/OL]. 2008. http://www.Microsoft.com/msdn.

[5] 闫宇晗，常　鑫. 在 C# 中用 GDI＋实现图形动态显示[J]. 计算机技术与发展，2006，16(12)：117－118.

[6] 黄烟波，赵旭华，刘中宇. 基于.NET 的流程图绘制程序的设计与实现[J]. 计算机技术与发展，2007，17(5)：231－233.

[7] 周鸣杨，赵景亮. 精通 GDI＋编程[M]. 北京：清华大学出版社，2004.

[8] 周长发. 精通 Visual C＋＋图像处理编程[M]. 北京：电子工业出版社，2006.