浙江科技学院学报，第19卷第3期，2007年9月
Journal of Zhejiang University of Science and Technology
Vol. 19 No. 3, Sep. 2007

基于Delphi扫描线技术的快速图像处理

王　华[a]，蒋惠忠[b]
（浙江科技学院 a. 自动化与电气工程学院；b. 中德学院，杭州 310023）

摘　要：传统的图像处理一般采用像素点赋值的方法，处理速度极慢，对于大型图像几乎无能为力，而Delphi扫描线方法是对图像的每一行进行扫描，获得各像素的内存地址。这种内存操作比常规的像素点赋值效率高很多，从而可以大大提高大型位图图像的处理速度。

关键词：内存操作；图像处理；扫描线
中图分类号：TP311　　**文献标识码**：A　　**文章编号**：1671-8798(2007)03-0189-04

Fast Image Processing Based on Scanline of Delphi

WANG Hua[a], JIANG Hui-zhong[b]
(a. School of Automation and Electronic Enginerring; b. Chinese-German School,
Zhejiang University of Science and Technology, Hangzhou 310023, China)

Abstract: For the traditional method of image processing is based on pixel assignment, the speed of processing is quite slow and hardly acceptable in processing the big bitmap. Scanline method of Delphi introduced in the paper can scan every line of the bitmap image and obtain each pixel's address. The memory operation method is faster than the traditional method based on pixels, so the speed of image processing has been greatly improved.

Key words: memory operation; image processing; scanline

Borland公司推出的Windows编程工具Delphi以其高度优化的编译器[1]、强大的面向特性而越来越受到广大编程爱好者青睐，在许多大型项目如大型数据库系统设计、纹织CAD等企业级解决方案上有独特的优势。以往在图像处理领域往往是VC++的天下，其实Delphi在图像处理领域的强大功能丝毫不逊色于VC++，在处理的速度上也基本与VC++相当，图像操作软件已成为Delphi一个重要的应用方向。Delphi开发图像处理软件一般有两种基本方法：第一种方法是利用Delphi已经封装好了的可视化控件库VCL中已有的控件、Delphi中定义的类、对象、方法等，如TImage控件的一系列方法，TBitmap类的Scanline方法等，这种方法开发难度较低，开发者易于掌握；第二种方法是利用原生API或者经过Delphi稍微封装的函数开发[2]，这种方法的特点是代码运行效率更高、节约资源、速

收稿日期：2007-03-02
基金项目：浙江省科技计划资助项目(2004C31114)
作者简介：王　华(1978—　)，男，浙江淳安人，讲师，硕士，主要从事数字图像测控系统、网络监控系统和多媒体技术的研究。

度快，但同时开发难度稍大些。

1 扫描线技术(Scanline)进行快速处理的原理

Delphi 提供了 Canvas. Pixels 属性，可以在画布(Canvas)上直接访问某一点的像素值。但用这个属性来执行访问速度很慢。比如，如果使用 Pixels 属性来把一个位图旋转 90°，这种方法对于小的位图处理效果还是不错的，但是对于大一些的位图就无法接受了。它唯一的好处是，Canvas. Pixels 属性对于所有的像素大小(1～24 位)处理的方法都是一样的[3]。在 Delphi 中，可以替代 Canvas. Pixels 属性的就是直接引用 Windows API 来访问像素点数据(例如 GetDIBits)，但用 Windows API 的 DIB (Device Independent Bitmap)来访问像素数据更为复杂。假如使用 DIB 数据，则必须经常把它转换到 TBitmap，并以 TImage 来显示。Delphi 中 TBitmap 的 Scanline 和 PixelFormat 属性就提供了一个更好的 Canvas. Pixels 属性的代替。在 Delphi 的帮助中可以知道对 Scanline 的解释如下：

property ScanLine[Row: Integer]: Pointer;

其中 Row 为位图图像的行号，返回的是一个指向位图像素数据的指针，通过 Scanline 可以获取某一行的所有像素点的信息，方便而且高速。经测试，Scanline 的速度为 Canvas. Pixels[x][y]方法的 20 倍以上[3]。Delphi 的 Scanline 包含了像素点的数据，但在访问像素数据以前，必须知道在内存中 PixelFormat 属性对 Scanline 属性的设计，因为不同的 PixelFormat 属性代表不同的位图格式，因而就有不同的 Scanline 方法。PixelFormat 已在 Graphics. pas 中定义了，其中包括 pfCustom, pfDevice, pf1bit, pf4bit, pf8bit, pf15bit, pf16bit, pf24bit 和 pf32bit。操作 pf8bit 位图相对容易，因为它的每个像素正好占一个字节，能够在 pByteArray 中直接访问。不过此时 Scanline 的值表示的是调色板上颜色的索引。调色板上包含了确切的 R、G、B 色。对于 pf24bit 的位图，首先定义一个类型：

```
Type
pRGBTripleArray=^TRGBTripleArray;
TRGBTripleArray= ARRAY[0..32767] OF TRGBTriple;
```

由于 24bit 的位图一个像素点是 3 个字节，不含调色板信息，所有的字节信息就是颜色信息，而由于 pf32bit 的位图一个像素点代表 4 个字节，Borland 公司为此定义了一个类型(在 Windows. pas 下面)：

```
pRGBQuad=^TRGBQuad;
TRGBQuad=PACKED RECORD
    rgbBlue: BYTE;
    rgbGreen: BYTE;
    rgbRed: BYTE;
    rgbReserved: BYTE
```

在颜色上，TRGBQuad 和 TRGBTriple 是一样的。两者都用 24 位来记录颜色信息，8 位红色，8 位绿色，8 位蓝色。TRGBQuad 有更大一些的空间区域——“alpha”通道(在苹果机上较多采用)。alpha 通道在某些应用程序上用来传送灰度掩码信息，但现在还没有 alpha 通道相关的定义。要使用 pf32bit 的 Scanline 就相当于要像使用单值动态数组一样来使用 TBitmap。在测试中，用 pf32bit 的 Scanline 要比 pf24bit 快 5%。这就是 32 位属性的特点。所以用 pf32bit Scanline 有一些优势，但是对于 pf32bit 来说，却没有那么多足够的空间来保存 Scanline 的值，一般在程序中将位图的格式设置为 pf24bit 就足够了。

2 图像的滤镜处理

图像的滤镜处理可以用卷积运算来解决，卷积时使用一个奇数维的矩阵来表示[3]。该矩阵体现在程序中就是模板概念，区域中的每个像素分别与模板中相应的元素相乘，相乘之和除以某个系数即为区域中心像素新值。例如一个 3×3 的图像区域 $\boldsymbol{B}$ 和模板 $\boldsymbol{Q}$ 卷积后，图像区域 $\boldsymbol{B}$ 的中心像素 B_5 像素值表示为：

$$B_5=\sum_{i=1}^{9}B_i\cdot Q_i$$

其中：

$$\boldsymbol{B}=\begin{bmatrix}B_1 & B_2 & B_3\\ B_4 & B_5 & B_6\\ B_7 & B_8 & B_9\end{bmatrix}$$

$$\boldsymbol{Q}=\begin{bmatrix}Q_1 & Q_2 & Q_3\\ Q_4 & Q_5 & Q_6\\ Q_7 & Q_8 & Q_9\end{bmatrix}$$

不同的模板可以得到不同的效果，滤镜所用模板采用 3×3 矩阵，也可以采用 5×5 的矩阵，维数越大，处理时间越久。在采用模板操作时必须解决两个问题：一个是边界点问题，一般可以忽略第一列和最后一列像素的操作，或者直接进行边界像素的拷

贝；还有一个是越界问题，必须保证中心像素点的各分量在 0～255 范围。为此定义一个存取模板矩阵值的数组 y，长度为 9，对于不同的滤镜效果，只需改变模板的矩阵值，而无需改动该处理过程[4]。同时创建两个位图对象 bmp1，bmp2，处理的思路是将 Image 控件上的位图分别加载到两个位图对象 bmp1，bmp2 中，其中从 bmp1 获取要处理的像素的信息，从 bmp2 获取要处理的像素的 8 个相邻像素信息。这样做的好处是从 bmp2 获取的是原始的未修改的信息，最后将 bmp1 显示于 Image 控件上。核心代码部分如下：

```
Var
  bmp1, bmp2: TBitmap;//定义位图对象
  p1, p2, p3, p4: pByteArray;//定义字节指针
  i, j, z: integer;
  y: array[0..8] of integer;//定义模板数组
begin
  z := 1; //加权后要除的系数
  bmp1 := TBitmap.Create;//创建位图对象 1
  bmp2 := TBitmap.Create;//创建位图对象 2
  //加载位图到对象 1 中
  bmp1.Assign(image1.Picture.Bitmap);
  //格式转为 24bit
  bmp1.PixelFormat:=pf24bit;
  //加载位图到对象 2 中
  /p2.Assign(image1.Picture.Bitmap);
  //格式转为 24bit
  bmp2.PixelFormat := pf24bit;
  for j:=1 to bmp1.height-2 do
    begin
    ////扫描位图对象 1 第 j 行
    p1 := bmp1.ScanLine[j];
    //扫描位图对象 2 的第 j-1 行
    p2 := bmp2.ScanLine[j-1];
    //扫描位图对象 2 的第 j 行
    p3 := bmp2.ScanLine[j];
    //扫描位图对象 2 的第 j+1 行
    p4 := bmp2.ScanLine[j+1];
    for i := 1 to bmp1.Width-2 do
      begin
      //计算第 i 个像素的红色分量
      p1[3*i+2]:=min(255, max(0, ((y[0]
* p2[3 * (i-1)+2]+y[1] * p2[3 * i+2]+y
[2] * p2[3(i+1)+2]+y[3] * p3[3 * (i-1)+
2]+y[4] * p3[3 * i+2]+y[5] * p3[3 * (i+
1)+2]+y[6]* p4[3 * (i-1)+2]+y[7] * p4
[3 * i+2]+y[8] * p4[3 * (i+1)+2]))));
      //计算第 i 个像素的绿色分量
      p1[3* i+1] := min(255, max(0, ((y[0] *
p2[3 * (i-1)+1]+y[1] * p2[3 * i+1]+y[2]*
p2[3 * (i+1)+1]+y[3] * p3[3 * (i-1)+1]+y
[4] * p3[3 * i+1]+y[5] * p3[3 * (i+1)+1]+
y[6]* p4[3* (i-1)+1]+y[7] * p4[3 * i+1]+
y[8] * p4[3 * (i+1)+1])) div z));
      //计算第 i 个分量的蓝色分量
      p1[3*i]:=min(255, max(0, ((y[0] * p2
[3 * (i-1)]+y[1]*p2[3*i]+y[2]*p2[3*(i
+1)]+y[3]*p3(i-1)]+y[4] * p3[3* i]+y
[5] * p3[3* (i+1)]+y[6] * p4[3 * (i-1)]
+y[7]* p4[3 * i]+y[8] * p4[3 * (i+1)])) ));
      end;
    end;
  //将用模板处理后的位图重新显示
  Image1.Picture.Bitmap.Assign(Bmp1)
  Bmp1.free;//释放位图对象资源
  Bmp2.free;
end;
```

对于一幅 24 位的真彩色位图，代码中巧妙地利用 Scanline 获得了相邻三行像素的内存的起始地址，并由此取得特定像素点的 R、G、B 三个分量，其速度比用逐个的像素点操作快将近 20 倍，程序中的 max 和 min 函数有效地防止了中心像素点的越界可能。为了达到不同的滤镜效果，可以改变权系数数组 y，也就是改变模板矩阵 Q。

3　图像的 90°快速旋转处理

传统方法是将位图数据源像素信息赋予目的位置像素，最简单的方法就是采用 Canvas.Pixel[x, y]赋值方式实现，这种方法的缺点是对于大型位图其实现速度非常慢[4]。而扫描线方法可以获得旋转前位图和旋转后位图对应点的像素内存地址，通过把旋转前某点的像素值赋给旋转后对应点以达到整幅图像的旋转，这种基于内存的操作速度极快，核心代码如下：

```
var
  aStream: TMemorystream;//定义内存流
```

```
header: TBITMAPINFO;//定义位图信息头
dc: hDC;//定义设备 DC
//定义一个 THelpRGB 类型的指针
P: ^THelpRGB;
x, y, b, h: Integer;
RowOut: pRGBArray;//字节数组指针
Begin
  //创建一个内存流对象
  aStream := TMemoryStream.Create;
  //设置流的大小
   aStream.SetSize(Bitmap.Height * Bitmap.
Width * 4);
  dc := GetDC(0);//获得设备 DC
  P := aStream.Memory;//获得流的位置指针
   GetDIBits(dc, Bitmap.Handle, 0, itmap.
Height, P, header, dib_RGB_Colors);
  ReleaseDC(0, dc);//释放设备 DC
  b := bitmap.Height;//旋转前位图的高度
  h := bitmap.Width;//旋转前位图的宽度
  bitmap.Width := b;//旋转后位图的宽度
  bitmap.height := h;//旋转后位图的高度
  for y := 0 to (h-1) do
    begin
    //获得第 y 行像素信息,这个是最重要的一
步,节省时间
    rowOut := Bitmap.ScanLine[y];
    //获得内存指针
    P := aStream.Memory;
    //将内存指针 p 位置增加 y
    inc(p, y);
    for x := 0 to (b-1) do
      begin
      //读取像素信息并且赋予相应位置
      rowout[x] := p^.rgb;
      //将内存指针 p 位置增加 h
      inc(p, h);
      end;
    end;
  aStream.Free;//释放流资源
end;
```

4 实验结果对比

将一幅 1 024×768 的 24 bit 位图分别用 Scanline 方法和 Canvas.Pixel[x,y]赋值方法分别进行旋转和高斯滤波,所用时间对比如表 1 所示。

表 1 Scanline 的方法和 Canvas.Pixel[x,y]赋值方法处理时间比较 ms

实验名称	Scanline 方法所用时间	Canvas.Pixel[x,y]赋值法所用时间
高斯滤波	130	2 705
90°旋转	150	3 240

由表 1 也可以看出,采用 Canvas.Pixel[x,y]方法的时间是 Scanline 方法的近 20 倍,其他的诸如位图的亮度、色度、饱和度等处理时间比较,Canvas.Pixel[x,y]方法的时间也接近 Scanline 方法时间的 20 倍。

5 结 语

Delphi 的扫描线方法通过对位图图像的逐行扫描得到像素点的内存地址信息,所有的图像处理策略都在内存中实现,速度很快,这种优化方法对点阵图像的处理效果非常适合,对于大型位图图像,速度也很理想[5],用 Scanline 方法进行图像处理速度是像素点赋值速度的 20 倍以上。因此,Scanline 方法越来越受到 Delphi 图像处理程序员的推崇。

参考文献:

[1] MARCO Cantu,王辉. Delphi 从入门到精通[M]. 李澎东,等译. 北京:电子工业出版社,2002.

[2] 王华,梁志刚,王众,等. Delphi 编程实例与技巧[M]. 北京:机械工业出版社,2002.

[3] 阮秋琦. 数字图像处理学[M]. 北京:电子工业出版社,2002.

[4] 刘俊,王华. Delphi 数字图像处理高级编程[M]. 北京:科学出版社,2003.

[5] 王华. 基于内存映射文件的特大机械图纸快速处理技术[J]. 浙江科技学院学报,2005,17(3):180-182.