【数字图像处理】七.MFC图像增强之图像普通平滑、高斯平滑、Laplacian、Sobel、Prewitt锐化详解

原创 置顶 Eastmount 最后发布于2015-06-08 18:39:07 阅读数 35866 ☆ 收藏 展开

Python+TensorFlow人工智能 ¥9.90

该专栏为人工智能入门专栏，采用Python3和TensorFlow实现人工智能相关算法。前期介绍安装流程、基础语法...

Eastmount 去订阅

本文主要讲述基于VC++6.0 MFC图像处理的应用知识，主要结合自己大三所学课程《数字图像处理》及课件进行讲解，主要通过MFC单文档视图实现显示BMP图像增强处理，包括图像普通平滑、高斯平滑、不同算子的图像锐化知识。希望该篇文章对你有所帮助，尤其是初学者和学习图像处理的学生。

【数字图像处理】一.MFC详解显示BMP格式图片

【数字图像处理】二.MFC单文档分割窗口显示图片

【数字图像处理】三.MFC实现图像灰度、采样和量化功能详解

【数字图像处理】四.MFC对话框绘制灰度直方图

【数字图像处理】五.MFC图像点运算之灰度线性变化、灰度非线性变化、阈值化和均衡化处理详解

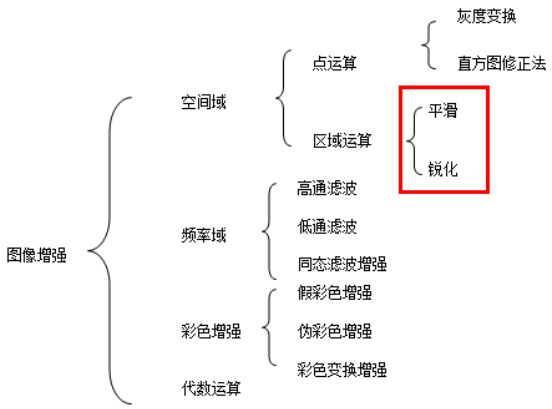
【数字图像处理】六.MFC空间几何变换之图像平移、镜像、旋转、缩放详解

免费资源下载地址：

http://download.csdn.net/detail/eastmount/8785591

一. 图像增强简介

图像增强是对图像进行处理，使其比原始图像更适合于特定的应用，它需要与实际应用相结合。对于图像的某些特征如边缘、轮廓、对比度等，图像增强是进行强调或锐化，以便于显示、观察或进一步分析与处理。图像增强的方法是因应用不同而不同的，研究内容包括：(参考课件和左飞的《数字图像处理》)



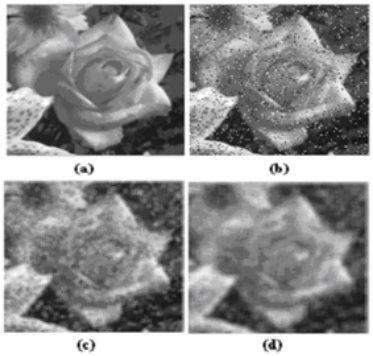
二. 图像平滑

图像平滑是一种区域增强的算法，平滑算法有邻域平均法、中指滤波、边界保持类滤波等。在图像产生、传输和复制过程中，常常会因为多方面原因而被噪声干扰或出现数据丢失，降低了图像的质量（某一像素，如果它与周围像素点相比有明显的不同，则该点被噪声所感染）。这就需要对图像进行一定的增强处理以减小这些缺陷带来的影响。

1.简单平滑-邻域平均法

图像简单平滑是指通过邻域简单平均对图像进行平滑处理的方法，用这种方法在一定程度上消除原始图像中的噪声、降低原始图像对比度的作用。它利用卷积运算对图像邻域的像素灰度进行平均，从而达到减小图像中噪声影响、降低图像对比度的目的。

但邻域平均值主要缺点是在降低噪声的同时使图像变得模糊，特别在边缘和细节处，而且邻域越大，在去噪能力增强的同时模糊程度越严重。



(a)原图像　　(b) 对(a)加椒盐噪声的图像

(c)3×3邻域平滑　　(d) 5×5邻域平滑

2.高斯平滑

为了克服简单局部平均法的弊端(图像模糊)，目前已提出许多保持边缘、细节的局部平滑算法。它们的出发点都集中在如何选择邻域的大小、形状和方向、参数加平均及邻域各店的权重系数等。

图像高斯平滑也是邻域平均的思想对图像进行平滑的一种方法，在图像高斯平滑中，对图像进行平均时，不同位置的像素被赋予了不同的权重。

在图像简单平滑中，算法利用卷积模板逐一处理图像中每个像素，这一过程可以形象地比作对原始图像的像素一一进行过滤整理，在图像处理中把邻域像素逐一处理的算法过程称为滤波器。平滑线性滤波器的工作原理是利用模板对邻域内像素灰度进行加权平均，也称为均值滤波器。

高斯平滑与简单平滑不同，它在对邻域内像素进行平均时，给予不同位置的像素不同的权值，下图的所示的3*3和5*5领域的高斯模板。

$$\frac{1}{16}\times\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

3x3 example

$$\frac{1}{273}\times\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 16 & 26 & 16 & 4\\ 7 & 26 & 41 & 26 & 7\\ 4 & 16 & 26 & 16 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix}$$

5x5 example

模板越靠近邻域中心位置，其权值越高。在图像细节进行模糊时，可以更多的保留图像总体的灰度分布特征。下图是常用的四个模板和matlab代码实现：

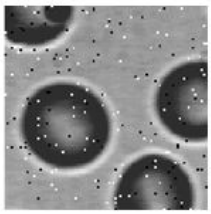
$$H_1=\frac{1}{10}\begin{bmatrix}1&1&1\\1&2&1\\1&1&1\end{bmatrix}\qquad H_2=\frac{1}{16}\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

$$H_3=\frac{1}{8}\begin{bmatrix}1&1&1\\1&0&1\\1&1&1\end{bmatrix}\qquad H_4=\frac{1}{2}\begin{bmatrix}0&\frac{1}{4}&0\\\frac{1}{4}&1&\frac{1}{4}\\0&\frac{1}{4}&0\end{bmatrix}$$

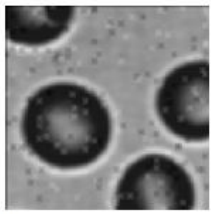
代码如下：

```
I1 = imread('blood1.tif');
I=imnoise(I1,'salt & pepper',0.04);                %对图像加椒盐噪声
imshow(I);
h1= [0.1 0.1 0.1; 0.1 0.2 0.1; 0.1 0.1 0.1];       %定义4种模板
h2=1/16.*[1 2 1;2 4 2;1 2 1];
h3=1/8.*[1 1 1;1 0 1;1 1 1];
h4=1/2.*[0 1/4 0;1/4 1 1/4;0 1/4 0];
I2=filter2(h1,I);                                  %用4种模板进行滤波处理
I3=filter2(h2,I);
I4=filter2(h3,I);
I5=filter2(h4,I);
figure,imshow(I2,[])                               %显示处理结果
figure,imshow(I3,[])
figure,imshow(I4,[])
figure,imshow(I5,[])
```

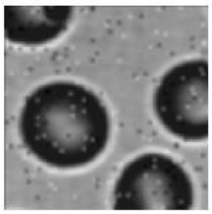
运行效果如下图所示：



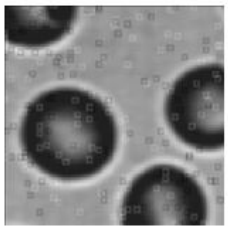
（a）有噪声的图像



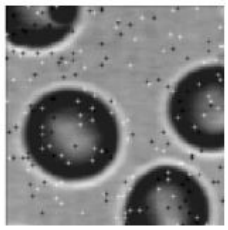
（b）模板1处理的结果图



（c）模板2处理的结果图



（d）模板3处理的结果图



（e）模板4处理的结果图

3.中值滤波

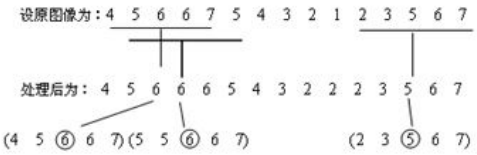
在使用邻域平均法去噪的同时也使得边界变得模糊。而中值滤波是非线性的图像处理方法，在去噪的同时可以兼顾到边界信息的保留。

选一个含有奇数点的窗口W，将这个窗口在图像上扫描，把窗口中所含的像素点按灰度级的升或降序排列，取位于中间的灰度值来代替该点的灰度值。

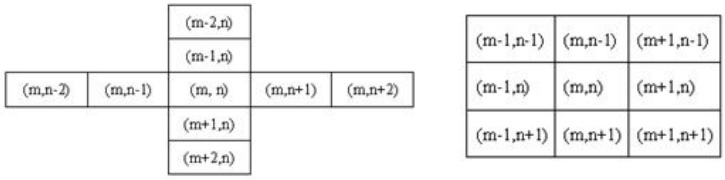
$$g(m,n)=Median\{f(m-k,n-l),(k,l)\in W\}$$

例如选择滤波的窗口如下图，是一个一维的窗口，待处理像素的灰度取这个模板中灰度的中值，滤波过程如下：

图9.9 一维窗口



常用的窗口还有方形、十字形、圆形和环形。不同形状的窗口产生不同的滤波效果，方形和圆形窗口适合外轮廓线较长的物体图像，而十字形窗口对有尖顶角状的图像效果好。



中值滤波对于消除孤立点和线段的干扰十分有用，尤其是对于二进噪声，但对消除高斯噪声的影响效果不佳。对于一些细节较多的复杂图像，可以多次使用不同的中值滤波。matlab实现参考：http://blog.csdn.net/timidsmile/article/details/6904381

4.边界保持类滤波

K近邻均值滤波器(KNNF)是指在m*m的窗口中，属于同一集合类的像素，它们的灰度值将高度相关。被处理的像素(对应于窗口中心的像素)可以用窗口内与中心像素灰度最接近的k个近邻像素的平均灰度来替代。步骤如下：

(1).作一个m*m的作用模板

(2).在其中选择K个与待处理像素的灰度差为最小的像素

(3).用这K个像素的灰度均值替换掉原来的值

◆ 模板为3×3，$k=3$的K近邻均值滤波器。

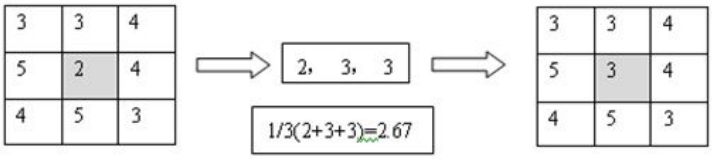


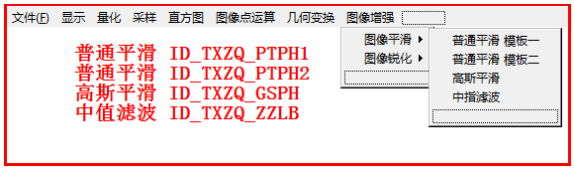
图4.12 K近邻均值滤波器

在K近旁均值滤波器(KNNMF)中，不选K个邻近像素的平均灰度来替代，而选K个邻近像素的中值灰度来替代，上图中2,3,3中选择3即可。

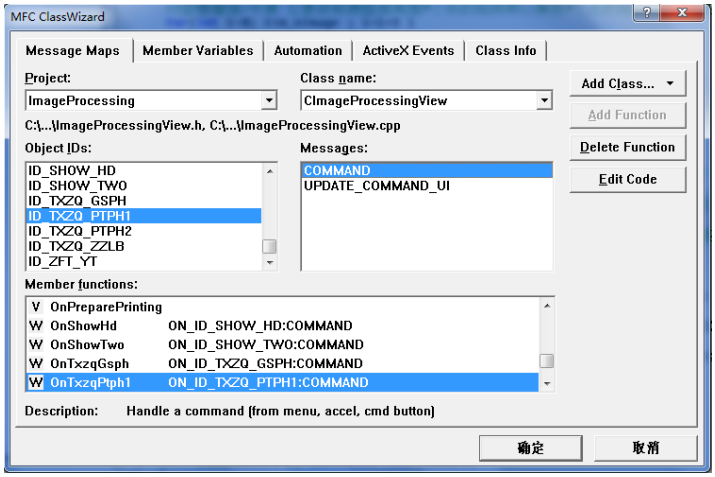
下面介绍具体MFC VC++6.0代码实现过程。

三. 图像平滑代码实现

第一步：在资源视图的Menu中添加子菜单“图像增强”，然后添加“图像平滑”四个选项如下图所示：



第二步：打开类向导，在ImageProcessingView类中添加相应的四个实现函数：



第三步：就是具体的平滑实现函数。

1.普通平滑 模板一

该算法采用的模板如下：

$$H_1 = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

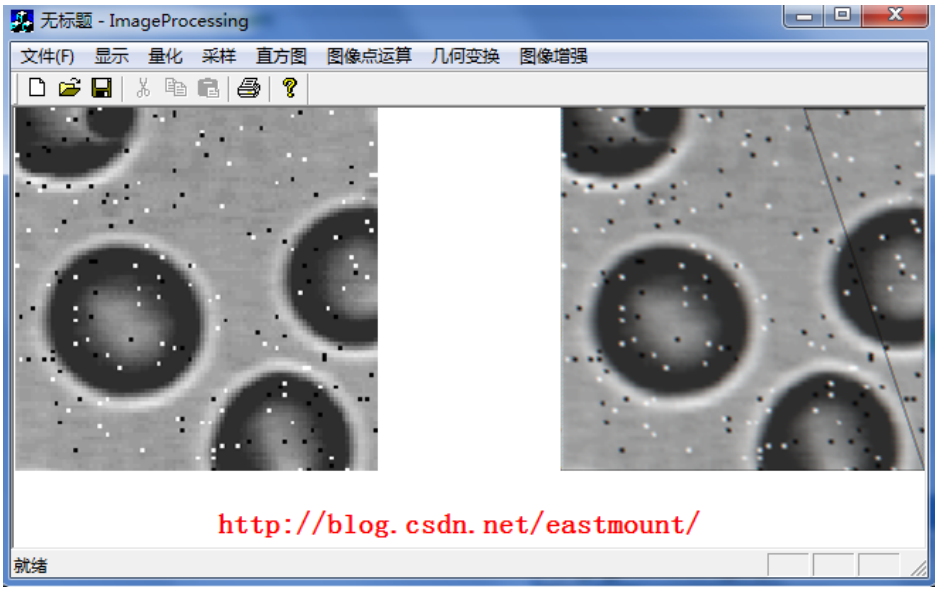
代码如下：

```
/************************************************
  第九章--图像增强
  图像平滑  普通平滑  模板

float H1[3][3]={{1.0/10,1.0/10,1.0/10}, //模板一:系数1/10
                {1.0/10,2.0/10,1.0/10},
                {1.0/10,1.0/10,1.0/10}};

float H2[3][3]={{1.0/16,2.0/16,1.0/16}, //模板二:系数1/16
                {2.0/16,4.0/16,2.0/16},
                {1.0/16,2.0/16,1.0/16}};

float H3[3][3]={{1.0/8,1.0/8,1.0/8},    //模板三:系数1/8,此种情况为把点转为空心矩形
                {1.0/8,0.0/8,1.0/8},
                {1.0/8,1.0/8,1.0/8}};

float H4[3][3]={{0.0,1.0/8,0.0},        //模板四:系数乘数据后的矩阵
                {1.0/8,1.0/2,1.0/8},
                {0.0,1.0/8,0.0}};

/************************************************/
```

```
void CImageProcessingView::OnTxzqPtph1()  {
        if(numPicture==0) {
                AfxMessageBox("载入图片后才能图像增强(平滑)!",MB_OK,0);
                return;
        }
        AfxMessageBox("图像增强(平滑)!选取的模板为:普通平滑  模板一",MB_OK,0);

        /*****************************************************************/
        /* 图想平滑的算法:
        /*   1.定义常用的四个模板,它们的维数均为3,矩阵的个数均为9个数据
        /*   2.它的思想是把一个点分散到这周围的9个点上,这样使图像更模糊
        /*   3.通过卷积计算围绕该点的矩阵像素和,计算其平均值(除9)赋值给点
        /*   4.模块不同,处理后的图像也各不相同
        /*****************************************************************/

        /*第一步: 先定义数据模板*/
        int HWS=3;                                  //模板维数:此四个模板均为3维的
        float H1[3][3]={{1.0/10,1.0/10,1.0/10},   //模板一:系数1/10
                                        {1.0/10,2.0/10,1.0/10},
                                        {1.0/10,1.0/10,1.0/10}};

        //打开临时的图片
        FILE *fpo = fopen(BmpName,"rb");
        FILE *fpw = fopen(BmpNameLin,"wb+");
        fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
    fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);
        fwrite(&bfh,sizeof(BITMAPFILEHEADER),1,fpw);
        fwrite(&bih,sizeof(BITMAPINFOHEADER),1,fpw);
        fread(m_pImage,m_nImage,1,fpo);

        //new和delete有效的进行动态内存的分配和释放
    unsigned char *ImageSize;
    ImageSize = new unsigned char[m_nImage];
        float red,green,blue;
        int X,Y;                //一维坐标转换为二维坐标
        int TR,TG,TB;           //记录红绿蓝坐标位置

        //图像增强:平滑  它要获取源图像周围9个点的矩阵乘以模板9个点的矩阵,故一维图像转二维
        for(int i=0; i<m_nImage ; i=i+3 )
        {
                //原图: 一维矩阵转换为二维矩阵
                X=(i/3)%m_nWidth;    //图像在X列
                Y=(i/3)/m_nWidth;    //图像在Y行

                //赋值为黑色,相当于清零
                red=green=blue=0;

                //对图像进行像素求和并取平均值 HWS维数
                for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                         //第j行
                {
                        for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                     //第k列
                        {
                                if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )      //防止越界
                                {
                                        //模板一 进行模板平均,把该点像素分散到四周
                                        TR=j*m_nWidth*3+k*3;
                                        red+=H1[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TR]);
                                        TG=j*m_nWidth*3+k*3+1;
                                        green+=H1[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TG]);
                                        TB=j*m_nWidth*3+k*3+2;
                                        blue+=H1[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TB]);
```

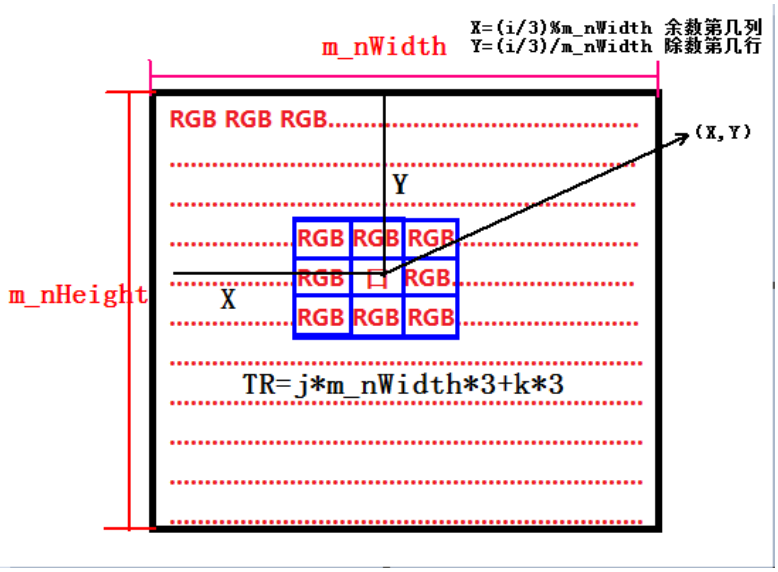
```
                        }
                                                    }
                }
                //对新图像赋值
                ImageSize[i]=(unsigned char)(red);
                ImageSize[i+1]=(unsigned char)(green);
                ImageSize[i+2]=(unsigned char)(blue);
        }

        fwrite(ImageSize,m_nImage,1,fpw);
        fclose(fpo);
        fclose(fpw);
        numPicture = 2;
        level=400;
        Invalidate();
}
```

运行效果如图所示，图像平滑模糊了，但效果不是很好。



其中实现的具体原理如下：

首先将图像像素矩阵转换为(X,Y)的二维矩阵进行操作，同时获取(X,Y)坐标为中心的3*3矩阵，再通过它与3*3模板进行像素平均操作，就是两个3*3矩阵互乘。需要注意的是矩阵一个格子是RGB三字节(24位BMP)，同时获取该中心点位置时，通过两层循环for(k=n-1;k<=n+1;k++)实现获取矩阵中九个点的像素。最后对该点(X,Y)的RGB进行赋值操作即可。



2.普通平滑 模板二

该算法采用的模板如下：

$$H_3 = \frac{1}{8}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

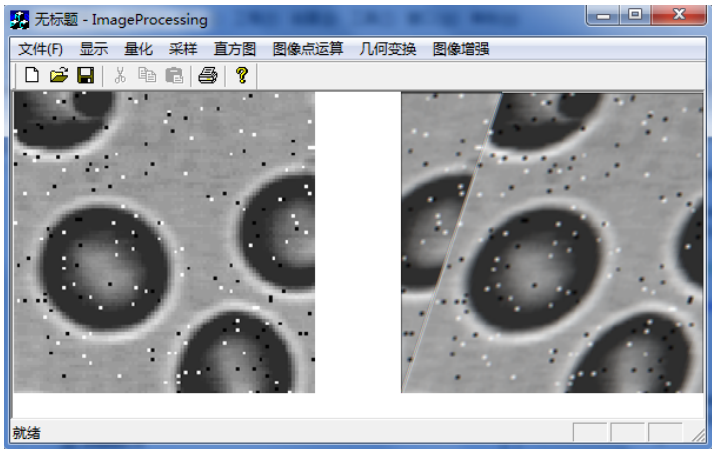
代码如下：

```
void CImageProcessingView::OnTxzqPtph2()
{
        if(numPicture==0) {
                AfxMessageBox("载入图片后才能图像增强(平滑)!",MB_OK,0);
                return;
        }
        AfxMessageBox("图像增强(平滑)!选取的模板为:普通平滑  模板二",MB_OK,0);

        /*第一步：先定义数据模板*/
        int HWS=3;
        float H2[3][3]={{1.0/8,1.0/8,1.0/8},     //模板三:系数1/8 此种情况为把点转为空心矩形
                                        {1.0/8,0.0/8,1.0/8},
                                        {1.0/8,1.0/8,1.0/8}};

        //打开临时的图片
        FILE *fpo = fopen(BmpName,"rb");
        FILE *fpw = fopen(BmpNameLin,"wb+");
        fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
    fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);

        //重点:图像的每行像素都必须是4的倍数：1*1的图像为 r g b 00H
        int num;            //记录每行多余的图像素数个数
        int sfSize;         //补齐后的图像大小
    if(m_nWidth*3%4!=0) {
        num=(4-m_nWidth*3%4);
        sfSize=(m_nWidth*3+num)*m_nHeight; //每行多number个
    }
    else {
```

```
    num=0;
                    sfSize=m_nWidth*m_nHeight*3;
}

/*更改文件头信息 定义临时文件头结构变量*/
BITMAPFILEHEADER bfhsf;
BITMAPINFOHEADER bihsf;
bfhsf=bfh;
bihsf=bih;
bfhsf.bfSize=sfSize+54;
fwrite(&bfhsf,sizeof(BITMAPFILEHEADER),1,fpw);
fwrite(&bihsf,sizeof(BITMAPINFOHEADER),1,fpw);
fread(m_pImage,m_nImage,1,fpo);

    //new和delete有效的进行动态内存的分配和释放
unsigned char *ImageSize;
ImageSize = new unsigned char[sfSize];
    float red,green,blue;
    int X,Y;                //一维坐标转换为二维坐标
    int TR,TG,TB;           //记录红绿蓝坐标位置
int countWidth=0;      //记录每行的像素个数,满行时变回0
    int place=0;            //建立临时坐标 记录起始坐标(0,0)平移过来的位置

    //图像增强 平滑
    for(int i=0; i<m_nImage; )
    {
            //原图一维矩阵转换为二维矩阵
            X=(i/3)%m_nWidth;    //图像在X列
            Y=(i/3)/m_nWidth;    //图像在Y行

            //赋值为黑色,相当于清零
            red=green=blue=0;

            //对图像进行像素求和并取平均值 HWS维数
            for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                        //第j行
            {
                    for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                    //第k列
                    {
                            if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )      //防止越界
                            {
                                    //模板二 进行模板平均,把该点像素分散到四周
                                    TR=j*m_nWidth*3+k*3;
                                    red+=H2[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TR]);
                                    TG=j*m_nWidth*3+k*3+1;
                                    green+=H2[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TG]);
                                    TB=j*m_nWidth*3+k*3+2;
                                    blue+=H2[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TB]);
                            }
                    }
            }
            //对新图像赋值
            //通过变量place赋值变换后的图像 i始终指向原图3的倍数 为了补0而添加place变量
            ImageSize[place]=(unsigned char)(red);
            i++; place++;
            ImageSize[place]=(unsigned char)(green);
            i++; place++;
            ImageSize[place]=(unsigned char)(blue);
            i++; place++;
            countWidth=countWidth+3;

            if(countWidth==m_nWidth*3)
```

```
        {                               if(num==0)
            {
                countWidth=0;
                place=Y*m_nWidth*3;
            }
            else //num为补0
            {
                for(int n=0;n<num;n++)
                {
                    ImageSize[place]=0;
                                        place++;
                }
                countWidth=0;
                place=Y*(m_nWidth*3+num); //重点 添加Num
            }
                }
        }

        fwrite(ImageSize,sfSize,1,fpw);
        fclose(fpo);
        fclose(fpw);
        numPicture=2;
        level=400;
        Invalidate();
}
```

你可能注意到了，在图像处理过程中，如果每行的字节数不是4的倍数，可能会出现斜线之类的处理BUG，所以需要手动补0筹齐4的倍数，代码中补0后运行效果如下图所示，我也一直没找到原因，可能是思想和深度还没有达到，以后有机会在解决吧！同时后面的算法都不准备再进行补0处理，主要讲述算法的思想！



3.高斯平滑

采用的模板如下：

$$H_2 = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

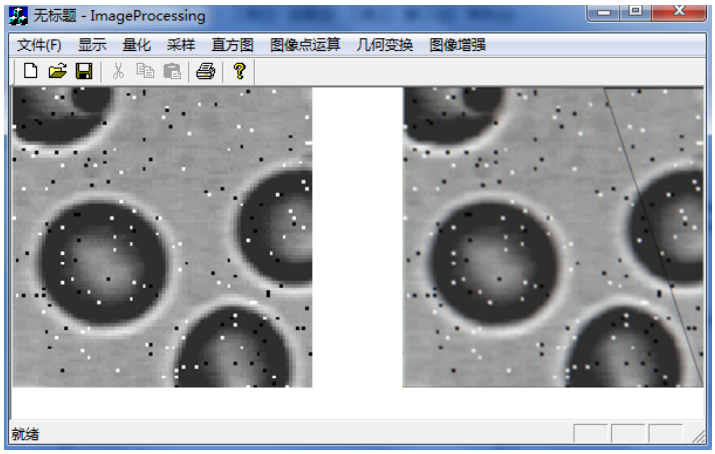
代码如下图所示：

```
//高斯平滑
void CImageProcessingView::OnTxzqGsph()
```

```
{
          if(numPicture==0) {
                AfxMessageBox("载入图片后才能图像增强(平滑)!",MB_OK,0);
                return;
        }
        AfxMessageBox("图像增强(平滑)!选取的模板为:高斯平滑",MB_OK,0);

        /*第一步: 先定义数据模板*/
        int HWS=3;                                  //模板维数为3维
        float H[3][3]={{1.0/16,2.0/16,1.0/16},    //高斯模板 系数1/16
                                {2.0/16,4.0/16,2.0/16},
                                {1.0/16,2.0/16,1.0/16}};

        //打开临时的图片
        FILE *fpo = fopen(BmpName,"rb");
        FILE *fpw = fopen(BmpNameLin,"wb+");
        fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
    fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);
        fwrite(&bfh,sizeof(BITMAPFILEHEADER),1,fpw);
        fwrite(&bih,sizeof(BITMAPINFOHEADER),1,fpw);
        fread(m_pImage,m_nImage,1,fpo);

        //new和delete有效的进行动态内存的分配和释放
    unsigned char *ImageSize;
    ImageSize = new unsigned char[m_nImage];
        float red,green,blue;
        int X,Y;                //一维坐标转换为二维坐标
        int TR,TG,TB;           //记录红绿蓝坐标位置

        //图像增强:平滑
        for(int i=0; i<m_nImage ; i=i+3 )
        {
                //原图: 一维矩阵转换为二维矩阵
                X=(i/3)%m_nWidth;    //图像在X列
                Y=(i/3)/m_nWidth;    //图像在Y行

                //赋值为黑色,相当于清零
                red=green=blue=0;

                //对图像进行像素求和并取平均值 HWS维数
                for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                     //第j行
                {
                        for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                  //第k列
                        {
                                if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )      //防止越界
                                {
                                        //模板二 进行模板平均,把该点像素分散到四周
                                        TR=j*m_nWidth*3+k*3;
                                        red+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TR]);
                                        TG=j*m_nWidth*3+k*3+1;
                                        green+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TG]);
                                        TB=j*m_nWidth*3+k*3+2;
                                        blue+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TB]);
                                }
                        }
                }
                //对新图像赋值
                ImageSize[i]=(unsigned char)(red);
                ImageSize[i+1]=(unsigned char)(green);
                ImageSize[i+2]=(unsigned char)(blue);
        }
```

```
                        fwrite(ImageSize,m_nImage,1,fpw);
        fclose(fpo);
        fclose(fpw);
        numPicture = 2;
        level=400;
        Invalidate();
}
```

运行效果如下图所示：



4.中值滤波

中值滤波我的理解是：它不但可以去除孤点噪声，而且可以保持图像的边缘特性，不会产生显著的模糊；它的方法是把局部区域的像素按灰度等级进行排序，再取该邻域中灰度的中值作为当前像素的灰度值。其步骤如下：

(1).将滤波模板(含若干个点的滑动窗口)在图像中漫游，并将模板中心与图像中的某个像素位置重合；

(2).读取模板中各对应像素的灰度值；

(3).将这些灰度值从小到大排序；

(4).取这一列数据的中间数据，将其赋值给对应模板中心位置的像素。

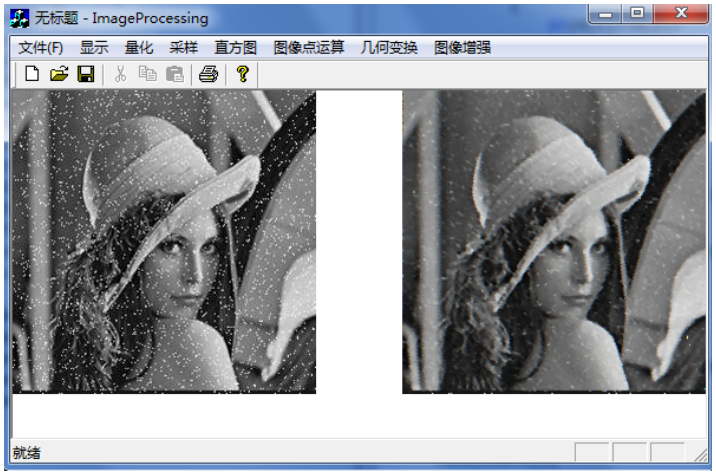
我采用的是3*3的模本，取矩阵中间位置像素替代原像素。代码如下：

```
//中值滤波
void CImageProcessingView::OnTxzqZzlb()
{
        if(numPicture==0) {
                AfxMessageBox("载入图片后才能图像增强(平滑)!",MB_OK,0);
                return;
        }
        AfxMessageBox("图像增强(平滑)!选取的模板为:中值滤波",MB_OK,0);

        //打开临时的图片
        FILE *fpo = fopen(BmpName,"rb");
        FILE *fpw = fopen(BmpNameLin,"wb+");
        fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
    fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);
        fwrite(&bfh,sizeof(BITMAPFILEHEADER),1,fpw);
        fwrite(&bih,sizeof(BITMAPINFOHEADER),1,fpw);
        fread(m_pImage,m_nImage,1,fpo);

        //new和delete有效的进行动态内存的分配和释放
    unsigned char *ImageSize;
    ImageSize = new unsigned char[m_nImage];
        int X,Y;                //一维坐标转换为二维坐标
        int TR,TG,TB;           //记录红绿蓝坐标位置
```

```
        //选取它为中心的周围9个点像素（注意一个点为RGB）
                                                        int H[9]={0,0,0,0,0,0,0,0,0};
        int HWS=3;              //维数为三维

        //图像增强:平滑 它要获取源图像周围9个点的矩阵乘以模板9个点的矩阵,故一维图像转二维
        for(int i=0; i<m_nImage ; i=i+3 )
        {
                //原图: 一维矩阵转换为二维矩阵
                X=(i/3)%m_nWidth;    //图像在X列
                Y=(i/3)/m_nWidth;    //图像在Y行

                //第一行 第一列 最后一行 最后一列 直接复制
                if(X==0 || Y==0 || X==m_nWidth*3 || Y==m_nHeight)
                {
                        if(i+2>m_nImage) break;
                        ImageSize[i] = m_pImage[i];
                        ImageSize[i+1] = m_pImage[i+1];
                        ImageSize[i+2] = m_pImage[i+2];
                        continue;
                }

                //对图像进行像素求和并取平均值 HWS维数
                int num=0;
                for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                     //第j行
                {
                        for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                  //第k列
                        {
                                if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )      //防止越界
                                {
                                        //获取当前位置Red像素 k一次增加RGB三个像素 R=G=B
                                        TR = j*m_nWidth*3+k*3;
                                        H[num] = m_pImage[TR];
                                        num++;
                                }
                        }
                }
                //排序获取中间值
                int temp=0;
                for(int x=0;x<9;x++)
                {
                        for(int y=x;y<9;y++)
                        {
                                if(H[x]>=H[y])
                                {
                                        temp=H[x];
                                        H[x]=H[y];
                                        H[y]=temp;
                                }
                        }
                }
                //CString str;
                //str.Format("矩阵:%d %d %d, %d %d %d, %d %d
%d",H[0],H[1],H[2],H[3],H[4],H[5],H[6],H[7],H[8]);
                //AfxMessageBox(str);
                //对新图像赋值 灰度图像RGB相同
                ImageSize[i]=H[4];
                ImageSize[i+1]=H[4];
                ImageSize[i+2]=H[4];
        }

        fwrite(ImageSize,m_nImage,1,fpw);
        fclose(fpo);
```

```
        fclose(fpw);
                        numPicture = 2;
        level=400;
        Invalidate();
}
```

运行效果如下图所示：



PS:这部分总算讲述完成，算法都是根据自己的理解用底层代码实现的，而不是向其它的通过调用GDI+库实现。可能存在因为理解不够或其它的错误，欢迎提出修改~

推荐资料：

图像平滑处理——OpenCV　　数字图像处理学习笔记——图像平滑锐化

中值滤波

四. 图像锐化

有时还需要加强图像中景物的边缘和轮廓，边缘和轮廓通常位于图像中灰度突出的地方，因而可以直观的想到用灰度的差分对边缘和轮廓进行提取，通常可以通过梯度算子进行提取。图像锐化的目的是提高图像的对比度，从而使图像更清晰，通过提高邻域内像素的灰度差来提高图像的对比度。

下面介绍图像锐化的几种算子及效果。

1.拉普拉斯算子(Laplacian)

拉普拉斯算子是图像邻域内像素灰度差分计算的基础，通过二阶微分推导出的一种图像邻域增强算法。它的基本思想是当邻域的中心像素灰度低于它所在邻域内的其他像素的平均灰度时，此中心像素的灰度应该被进一步降低；当高于时进一步提高中心像素的灰度，从而实现图像锐化处理。

在算法实现过程中，通过对邻域中心像素的四方向或八方向求梯度，并将梯度和相加来判断中心像素灰度与邻域内其他像素灰度的关系，并用梯度运算的结果对像素灰度进行调整。

一个连续的二元函数f(x,y)，其拉普拉斯运算定义为：

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

对于数字图像，拉普拉斯算子可以简化为：

$$g(i,j)=4f(i,j)-f(i+1,j)-f(i-1,j)-f(i,j+1)-f(i,j-1)$$

也可以表示为卷积的形式：

$$g(i,j)=\sum_{r=-k}^{k}\sum_{s=-l}^{l}f(i-r,j-s)H(r,s),\quad i,j=0,1,2,\cdots N-1$$

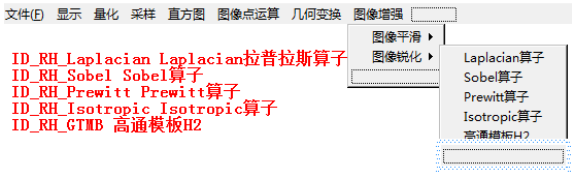
其中K=1，I=1时H(r,s)取下式，四方向模板：

$$H_1=\begin{bmatrix}0&-1&0\\-1&4&-1\\0&-1&0\end{bmatrix}$$

通过模板可以发现，当邻域内像素灰度相同时，模板的卷积运算结果为0；当中心像素灰度高于邻域内其他像素的平均灰度时，模板的卷积运算结果为正数；当中心像素的灰度低于邻域内其他像素的平均灰度时，模板的卷积为负数。对卷积运算的结果用适当的衰弱因子处理并加在原中心像素上，就可以实现图像的锐化处理。

其中实现过程步骤如下：

添加子菜单和类向导添加实现函数

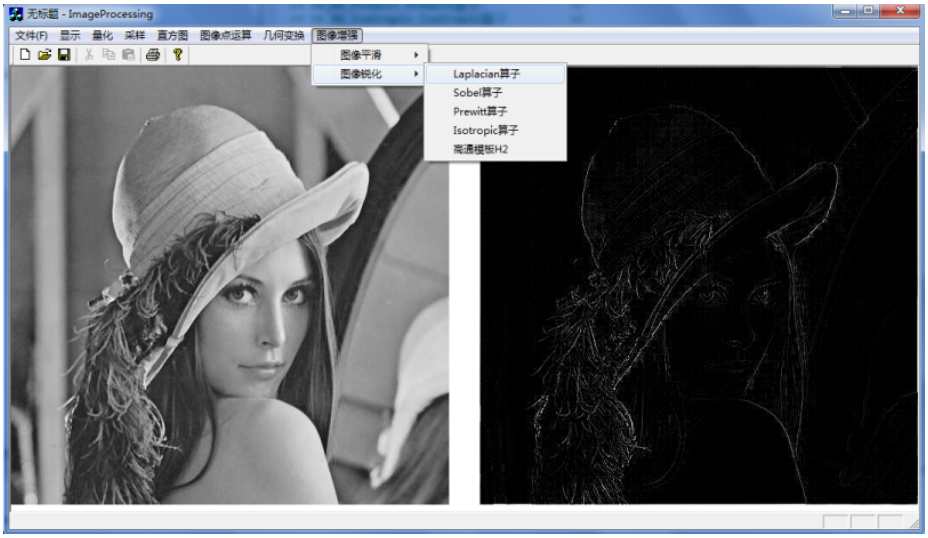


代码如下：

```
/*****************************************/
/* 图像锐化:我在menu中创建5个子的menu     */
/* 防止出现平滑错误,一次只调用一个下拉单 */
/* ID_RH_Laplacian Laplacian拉普拉斯算子 */
/* ID_RH_Sobel Sobel算子                  */
/* ID_RH_Prewitt Prewitt算子              */
/* ID_RH_Isotropic Isotropic算子          */
/* ID_RH_GTMB 高通模板H2                  */
/*****************************************/

void CImageProcessingView::OnRHLaplacian()
{
        if(numPicture==0)
        {
                AfxMessageBox("载入图片后才能图像增强(锐化)!",MB_OK,0);
                return;
        }
        AfxMessageBox("图像增强(锐化):采用拉普拉斯(Laplacian)算子!");

        //模板维数:此四个模板均为3维的
        int HWS=3;
        int H[3][3]={{0,-1,0},    //模板为拉普拉斯算子(中心为4的Laplacian)
                               {-1,4,-1},
                               {0,-1,0}};

        //读写文件
        FILE *fpo = fopen(BmpName,"rb");
        FILE *fpw = fopen(BmpNameLin,"wb+");
        fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
        fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);
        fwrite(&bfh,sizeof(BITMAPFILEHEADER),1,fpw);
        fwrite(&bih,sizeof(BITMAPINFOHEADER),1,fpw);
        fread(m_pImage,m_nImage,1,fpo);
```

```
        //new和delete有效的进行动态内存的分配和释放
                                                unsigned char *ImageSize;
        ImageSize=new unsigned char[m_nImage];
        int red,green,blue;
        int X,Y;        //一维坐标转换为二维坐标
        int TR,TG,TB;  //记录红绿蓝坐标位置

        //图像增强 锐化
        for(int i=0; i<m_nImage ; i=i+3 )
        {
                X=(i/3)%m_nWidth;    //X列
                Y=(i/3)/m_nWidth;    //Y行
                red=green=blue=0;

                //对图像进行像素求和并取平均值 HWS维数
                for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                    //第j行
                {
                        for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                  //第k列
                        {
                                if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )
                                {

                                        TR=j*m_nWidth*3+k*3;
                                        red+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TR]);
                                        TG=j*m_nWidth*3+k*3+1;
                                        green+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TG]);
                                        TB=j*m_nWidth*3+k*3+2;
                                        blue+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TB]);
                                }
                        }
                }
                //对新图像赋值
                if(red>=0 && red<256) ImageSize[i]=red;
                else if(red<0) ImageSize[i]=0;      //ImageSize[i]=-red;
                else ImageSize[i]=0;

                if(green>=0 && green<256) ImageSize[i+1]=green;
                else if(green<0) ImageSize[i+1]=0;  //ImageSize[i+1]=-green;
                else ImageSize[i+1]=0;

                if(blue>=0 && blue<256) ImageSize[i+2]=blue;
                else if(blue<0) ImageSize[i+2]=0;   //ImageSize[i+2]=-blue;
                else ImageSize[i+2]=0;
        }

        fwrite(ImageSize,m_nImage,1,fpw);
        fclose(fpo);
        fclose(fpw);
        numPicture = 2;
        level=400;
        Invalidate();
}
```

运行效果如下图所示：



2.高通滤波

常用的高通模板如下所示，其中H2有的书又称为拉普拉斯八方向的锐化模板。

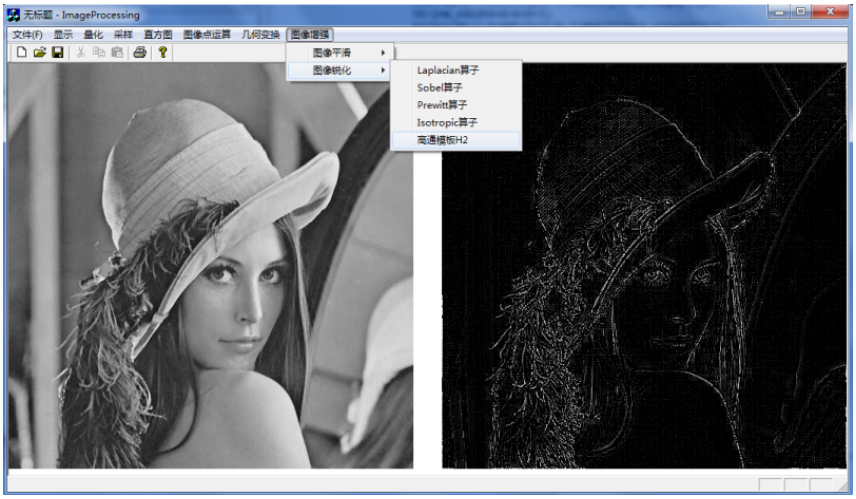
$$H_2=\begin{bmatrix}-1&-1&-1\\-1&8&-1\\-1&-1&-1\end{bmatrix}\quad H_3=\begin{bmatrix}1&-2&1\\-2&4&-2\\1&-2&1\end{bmatrix}\quad H_4=\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$$

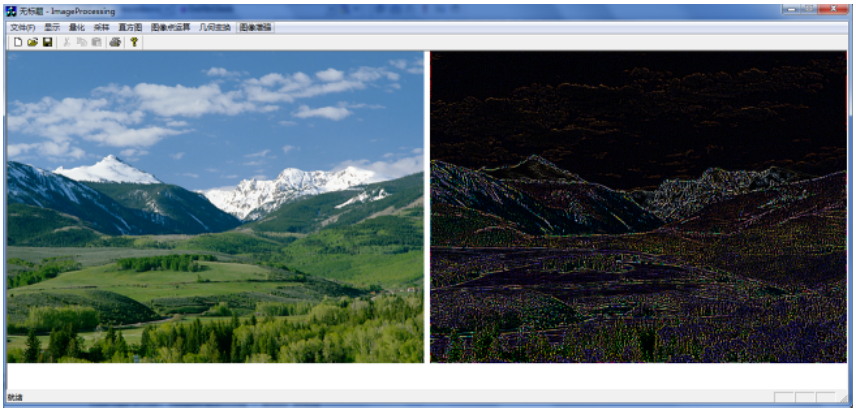
选取H2模板，代码如下所示：

```
//高通模板
void CImageProcessingView::OnRhGtmb()
{
	if(numPicture==0)
	{
		AfxMessageBox("载入图片后才能图像增强(锐化)!",MB_OK,0);
		return;
	}
	AfxMessageBox("图像增强(锐化):采用高通模板!");

	int HWS=3;
	int H[3][3]={{-1,-1,-1},
				{-1,8,-1},
				{-1,-1,-1}};

	FILE *fpo = fopen(BmpName,"rb");
	FILE *fpw = fopen(BmpNameLin,"wb+");
	fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
	fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);
	fwrite(&bfh,sizeof(BITMAPFILEHEADER),1,fpw);
	fwrite(&bih,sizeof(BITMAPINFOHEADER),1,fpw);
	fread(m_pImage,m_nImage,1,fpo);

	unsigned char *ImageSize;
	ImageSize=new unsigned char[m_nImage];
	int red,green,blue;
	int X,Y;
	int TR,TG,TB;

	//图像增强 锐化
```

```
        for(int i=0; i<m_nImage ; i=i+3 )        {
                X=(i/3)%m_nWidth;    //X列
                Y=(i/3)/m_nWidth;    //Y行
                red=green=blue=0;

                //对图像进行像素求和并取平均值 HWS维数
                for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                    //第j行
                {
                        for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                  //第k列
                        {
                                if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )
                                {

                                        TR=j*m_nWidth*3+k*3;
                                        red+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TR]);
                                        TG=j*m_nWidth*3+k*3+1;
                                        green+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TG]);
                                        TB=j*m_nWidth*3+k*3+2;
                                        blue+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TB]);
                                }
                        }
                }

                //对新图像赋值
                if(red>=0 && red<256) ImageSize[i]=red;
                else if(red<0) ImageSize[i]=0;      //ImageSize[i]=-red;
                else ImageSize[i]=0;

                if(green>=0 && green<256) ImageSize[i+1]=green;
                else if(green<0) ImageSize[i+1]=0;  //ImageSize[i+1]=-green;
                else ImageSize[i+1]=0;

                if(blue>=0 && blue<256) ImageSize[i+2]=blue;
                else if(blue<0) ImageSize[i+2]=0;   //ImageSize[i+2]=-blue;
                else ImageSize[i+2]=0;
        }

        fwrite(ImageSize,m_nImage,1,fpw);
        fclose(fpo);
        fclose(fpw);
        numPicture = 2;
        level=400;
        Invalidate();
}
```

运行效果如下图所示，该效果相对较好：





3.Sobel算子

$$S=\left(d_x^{\,2}+d_y^{\,2}\right)^{\frac{1}{2}}$$

$$d_x=[f(i-1,j-1)+2f(i-1,j)+f(i-1,j+1)]-[f(i+1,j-1)+2f(i+1,j)+f(i+1,j+1)]$$

$$d_y=[f(i-1,j+1)+2f(i,j+1)+f(i+1,j+1)]-[f(i-1,j-1)+2f(i,j-1)+f(i+1,j-1)]$$

用模板来表示:

$$d_x=\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}\qquad d_y=\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}$$

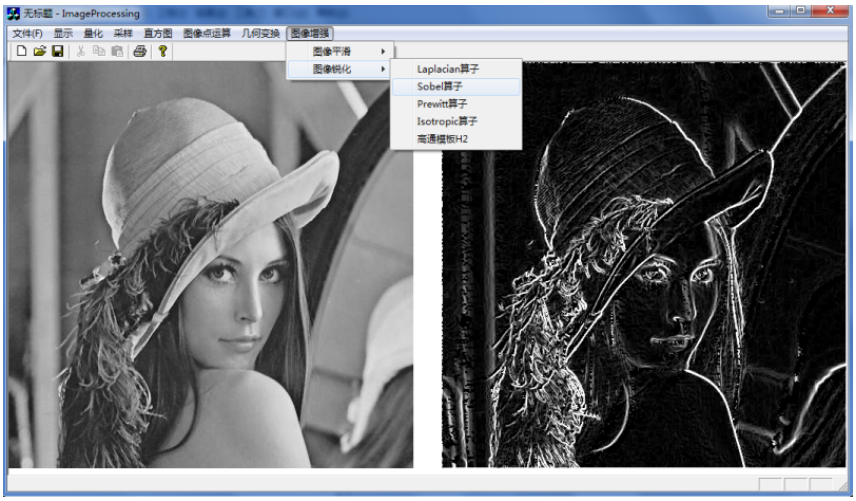
代码如下所示，需要注意X和Y两个方向的模板处理：

```
//Sobel算子采用PPT上的d(x)d(y)模板
void CImageProcessingView::OnRHSobel()
{
	if(numPicture==0)
	{
		AfxMessageBox("载入图片后才能图像增强(锐化)!",MB_OK,0);
		return;
	}
	AfxMessageBox("图像增强(锐化):采用Sobel算子!");

	int HWS=3;
```

```
//模板为Sobel算子
                        int HX[3][3]={{1,0,-1},{2,0,-2},{1,0,-1}};
int HY[3][3]={{-1,-2,-1},{0,0,0},{1,2,1}};

FILE *fpo = fopen(BmpName,"rb");
FILE *fpw = fopen(BmpNameLin,"wb+");
fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);
fwrite(&bfh,sizeof(BITMAPFILEHEADER),1,fpw);
fwrite(&bih,sizeof(BITMAPINFOHEADER),1,fpw);
fread(m_pImage,m_nImage,1,fpo);

unsigned char *ImageSize;
ImageSize=new unsigned char[m_nImage];
int redX,greenX,blueX;
int redY,greenY,blueY;
int X,Y;
int TR,TG,TB;

//图像增强 锐化
for(int i=0; i<m_nImage ; i=i+3 )
{
        X=(i/3)%m_nWidth;    //X列
        Y=(i/3)/m_nWidth;    //Y行
        redX=greenX=blueX=0;
        redY=greenY=blueY=0;

        //对图像进行像素求和并取平均值 HWS维数
        for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                        //第j行
        {
                for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                    //第k列
                {
                        if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )
                        {

                                TR=j*m_nWidth*3+k*3;
                                redX+=HX[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TR]);
                                redY+=HY[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TR]);
                                TG=j*m_nWidth*3+k*3+1;
                                greenX+=HX[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TG]);
                                greenY+=HY[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TG]);
                                TB=j*m_nWidth*3+k*3+2;
                                blueX+=HX[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TB]);
                                blueY+=HY[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TB]);
                        }
                }
        }
        //s=(d(x)*d(x)+d(y)*d(y))开根号
        int R,G,B;
        R=(int)(sqrt(redX*redX*1.0+redY*redY*1.0));
        G=(int)(sqrt(greenX*greenX*1.0+greenY*greenY*1.0));
        B=(int)(sqrt(blueX*blueX*1.0+blueY*blueY*1.0));

        if(redX<0 && redY<0) ImageSize[i]=0;
        else if(R>255) ImageSize[i]=255;
        else ImageSize[i]=R;

        if(greenX<0 && greenY<0) ImageSize[i+1]=0;
        else if(G>255) ImageSize[i+1]=255;
        else ImageSize[i+1]=G;

        if(blueX<0 && blueY<0) ImageSize[i+2]=0;
```

```
                else if(B>255) ImageSize[i+2]=255;
                                                        else ImageSize[i+2]=B;
        }

        fwrite(ImageSize,m_nImage,1,fpw);
        fclose(fpo);
        fclose(fpw);
        numPicture = 2;
        level=400;
        Invalidate();
}
```

运行效果如下图所示:



如果采用Sobel边缘细化，建议二值化(0和255阈值化)处理后再锐化，彩色图建议先灰度处理再进行其他处理。

4.Isotropic算子

$$S_I = \left(d_x^{\,2} + d_y^{\,2}\right)^{\frac{1}{2}}$$

用模板表示 d_x, d_y **：**

$$d_x = \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix} \qquad d_y = \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$$

代码实现如下：

```
//Isotropic算子采用PPT上的d(x)模板 d(y)
void CImageProcessingView::OnRHIsotropic()
{
        if(numPicture==0)
        {
                AfxMessageBox("载入图片后才能图像增强(锐化)!",MB_OK,0);
                return;
        }

        AfxMessageBox("图像增强(锐化):采用Isotropic算子!");

        int HWS=3;
        //模板为Isotropic算子
        float HX[3][3]={{1,0,-1},
                                        {sqrt(2.0),0,-sqrt(2.0)},
                                        {1,0,-1} };
        float HY[3][3]={{-1,-sqrt(2.0),-1},
                                        {0,0,0},
                                        {1,sqrt(2.0),1} };

        FILE *fpo = fopen(BmpName,"rb");
        FILE *fpw = fopen(BmpNameLin,"wb+");
        fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
        fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);
        fwrite(&bfh,sizeof(BITMAPFILEHEADER),1,fpw);
        fwrite(&bih,sizeof(BITMAPINFOHEADER),1,fpw);
        fread(m_pImage,m_nImage,1,fpo);

        unsigned char *ImageSize;
        ImageSize=new unsigned char[m_nImage];
        float redX,greenX,blueX;
        float redY,greenY,blueY;
        int X,Y;
        int TR,TG,TB;

        //图像增强
        for(int i=0; i<m_nImage ; i=i+3 )
        {
                X=(i/3)%m_nWidth;    //X列
                Y=(i/3)/m_nWidth;    //Y行
                redX=greenX=blueX=0;
                redY=greenY=blueY=0;

                //对图像进行像素求和并取平均值 HWS维数
                for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                    //第j行
                {
                        for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                    //第k列
```

```
                {                                                   if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )
                        {

                                TR=j*m_nWidth*3+k*3;
                                redX+=HX[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TR]);
                                redY+=HY[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TR]);
                                TG=j*m_nWidth*3+k*3+1;
                                greenX+=HX[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TG]);
                                greenY+=HY[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TG]);
                                TB=j*m_nWidth*3+k*3+2;
                                blueX+=HX[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TB]);
                                blueY+=HY[(j-Y+HWS/2)][(k-X+HWS/2)]*(float)(m_pImage[TB]);
                        }
                }
            }
            //对新图像赋值 s=(d(x)*d(x)+d(y)*d(y))开根号
            int R,G,B;
            R=(int)(sqrt(redX*redX*1.0+redY*redY*1.0));
            G=(int)(sqrt(greenX*greenX*1.0+greenY*greenY*1.0));
            B=(int)(sqrt(blueX*blueX*1.0+blueY*blueY*1.0));

            if(redX<0 && redY<0) ImageSize[i]=0;
            else if(R>255) ImageSize[i]=255;
            else ImageSize[i]=R;

            if(greenX<0 && greenY<0) ImageSize[i+1]=0;
            else if(G>255) ImageSize[i+1]=255;
            else ImageSize[i+1]=G;

            if(blueX<0 && blueY<0) ImageSize[i+2]=0;
            else if(B>255) ImageSize[i+2]=255;
            else ImageSize[i+2]=B;
        }
        fwrite(ImageSize,m_nImage,1,fpw);
        fclose(fpo);
        fclose(fpw);
        numPicture = 2;
        level=400;
        Invalidate();
}
```

运行效果如下图所示，效果与上面的Sobel类似：

5.Prewitt算子

Prewitt算子 $S_p = \left(d_x^{\,2} + d_y^{\,2}\right)^{\frac{1}{2}}$

用模板表示d_x, d_y

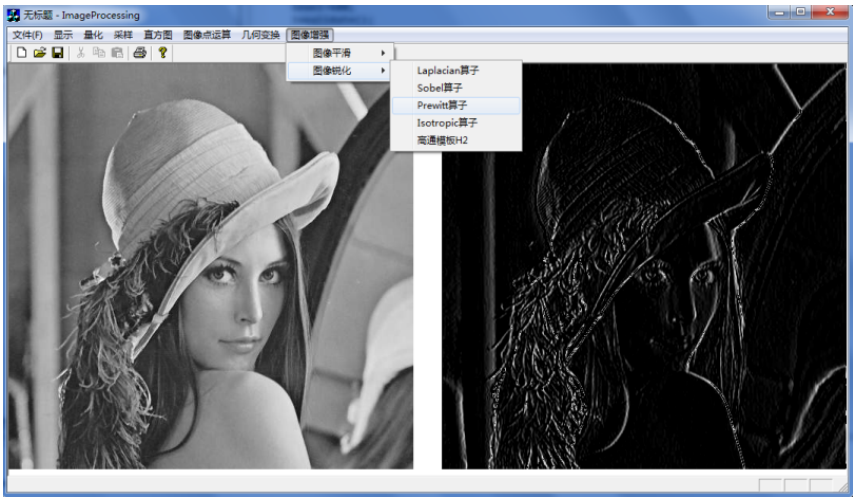
$$d_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \qquad d_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

代码如下图所示:

```
//Prewitt算子采用PPT上的d(x)模板,不是d(y)
void CImageProcessingView::OnRHPrewitt()
{

	if(numPicture==0)
	{
		AfxMessageBox("载入图片后才能图像增强(锐化)!",MB_OK,0);
		return;
	}
	AfxMessageBox("图像增强(锐化):采用Prewitt算子!");

	int HWS=3;
	int H[3][3]={{1,0,-1},    //模板为Prewitt算子
			{1,0,-1},
			{1,0,-1}};

	FILE *fpo = fopen(BmpName,"rb");
	FILE *fpw = fopen(BmpNameLin,"wb+");
	fread(&bfh,sizeof(BITMAPFILEHEADER),1,fpo);
	fread(&bih,sizeof(BITMAPINFOHEADER),1,fpo);
	fwrite(&bfh,sizeof(BITMAPFILEHEADER),1,fpw);
	fwrite(&bih,sizeof(BITMAPINFOHEADER),1,fpw);
	fread(m_pImage,m_nImage,1,fpo);

	unsigned char *ImageSize;
	ImageSize=new unsigned char[m_nImage];
	int red,green,blue;
	int X,Y;
	int TR,TG,TB;

	//图像增强:平滑
	for(int i=0; i<m_nImage ; i=i+3 )
	{
		X=(i/3)%m_nWidth;    //X列
		Y=(i/3)/m_nWidth;    //Y行
		red=green=blue=0;

		//对图像进行像素求和并取平均值 HWS维数
		for(int j=Y-HWS/2 ; j<Y+HWS/2+1 ; j++ )                    //第j行
		{
			for(int k=X-HWS/2 ; k<X+HWS/2+1 ; k++ )                  //第k列
			{
				if( j>=0 && k>=0 && k<m_nWidth && j<m_nHeight )
				{

					TR=j*m_nWidth*3+k*3;
					red+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TR]);
```

```
                                    TG=j*m_nWidth*3+k*3+1;
                                    green+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TG]);
                                    TB=j*m_nWidth*3+k*3+2;
                                    blue+=H[(j-Y+HWS/2)][(k-X+HWS/2)]*(m_pImage[TB]);
                                }
                                                    }
                }
                //对新图像赋值
                if(red>=0 && red<256) ImageSize[i]=red;
                else if(red<0) ImageSize[i]=0;      //ImageSize[i]=-red;
                else ImageSize[i]=0;

                if(green>=0 && green<256) ImageSize[i+1]=green;
                else if(green<0) ImageSize[i+1]=0;  //ImageSize[i+1]=-green;
                else ImageSize[i+1]=0;

                if(blue>=0 && blue<256) ImageSize[i+2]=blue;
                else if(blue<0) ImageSize[i+2]=0;   //ImageSize[i+2]=-blue;
                else ImageSize[i+2]=0;
        }
        fwrite(ImageSize,m_nImage,1,fpw);
        fclose(fpo);
        fclose(fpw);
        numPicture = 2;
        level=400;
        Invalidate();
}
```

运行效果如下图所示，只选取了X分量：



最后还是希望文章对你有所帮助，如果文章有不足或错误之处，请海涵。自己给自己点个赞，挺不容易的，但还会继续写完~同时后面的图像处理准备研究些感兴趣的东西，而不是这样的长篇大论了，例如怎样实现验证码提取、如何实现图像恢复、DICOM图像等知识吧！

(By:Eastmount 2015-06-08 下午6点 http://blog.csdn.net/eastmount/)

点赞 18 收藏 分享 ...

Eastmount 博客专家
发布了450 篇原创文章 · 获赞 6318 · 访问量 501万+
他的留言板 关注