**目 录**

[一、课程背景 1](#_Toc17615)

[二、课程选题 1](#_Toc4071)

[三、设计思想 1](#_Toc17909)

[1. 拉普拉斯算子 1](#_Toc379)

[2. 索贝尔算子 2](#_Toc2761)

[3. 罗伯特（Robert）算子 2](#_Toc31858)

[4. prewwit（普瑞维特）算子 3](#_Toc31529)

[5. 镜像，平移，颜色取反 4](#_Toc20125)

[四、数据结构与算法设计 4](#_Toc17769)

[1. 拉普拉斯算子 4](#_Toc11703)

[2. 模板运算函数 5](#_Toc28203)

[3. 索贝尔（sobel）算子 6](#_Toc25192)

[4. 罗伯特（Robert）算子 6](#_Toc16339)

[5. Prewwit算子 6](#_Toc10333)

[6. 平移，取反，镜像 6](#_Toc7889)

[平移： 6](#_Toc21486)

[取反： 7](#_Toc12573)

[镜像： 7](#_Toc4654)

[五、 运行结果与分析 7](#_Toc31374)

[1) 程序运行结果 7](#_Toc32355)

[1. 拉普拉斯算子 7](#_Toc23800)

[2. 索贝尔（Sobel）算子 8](#_Toc24617)

[3. 罗伯特（Robert）算子 9](#_Toc8761)

[4. Prewwit算子 9](#_Toc16340)

[5. 平移，取反，镜像 9](#_Toc10131)

[2） 分析 10](#_Toc10962)

[六、困难及解决方法 12](#_Toc15631)

[七、总结心得 14](#_Toc4117)

[八、程序操作说明及主要代码 16](#_Toc8380)

[1、 程序操作的简要说明书： 16](#_Toc7373)

[① 打开一幅图片 16](#_Toc31996)

[② 从菜单选取操作 16](#_Toc1020)

[③ 如果要放弃所做的操作 17](#_Toc19017)

[④ 如果要保存所做的操作 18](#_Toc12203)

[2、 主要代码 19](#_Toc14358)

[//拉普拉斯算子菜单的响应函数 19](#_Toc4300)

[//功能：空域上的模板运算 20](#_Toc7606)

[//功能：该函数用来镜像DIB图像。可以指定镜像的方式是水平还是垂直。 22](#_Toc11012)

[//功能：水平移动DIB图像。 23](#_Toc15210)

[//取反菜单的响应函数 25](#_Toc9204)

[九、 参考材料 26](#_Toc5391)

**一、课程背景**

数字图像处理技术的发展涉及信息科学、计算机科学、数学、物理学以及生物学等学科，因此数理及相关的边缘学科对图像处理科学的发展有越来越大的影响。近年来，数字图像处理技术日趋成熟，它广泛应用于空间探测、遥感、生物医学、人工智能以及工业检测等许多领域，并促使这些学科产生了新的发展。

图像是人类获取和交换信息的主要来源，因此，图像处理的应用领域必然涉及到人类生活和工作的方方面面。随着科学技术的发展，数字图像处理技术的应用领域也将随之不断扩大。

正因为图像处理在社会生活中有着越来越重要的作用，这也就体现了我们学习这门课程的必要性。而通过课程的学习可以让我们掌握一些图像处理的初步技术，从而为今后在图像处理的学习方面打下了基础。

**二、课程选题**

由于老师并没有做出规定去做哪一道题目，这给了我们很大程度上的自主性。我根据自身的情况和能力，选择了经典的几种边缘检测算子的特点的比较这个题目，和图象的平移和镜像，又因为在计算机图形学这门课中有讲到图片相加的具体程序，在这里图简单做一下颜色取反。通过对这些算法的编写，更加深入地了解了图像处理的过程，并且也在一定程度上提高了自己的编程能力。

边缘检测技术是所有基于边界分割的图像分析方法的第一步，首先检测出图像局部特性的不连续性，再将它们连成边界，这些边界把图像分成不同的区域，检测出边缘的图像就可以进行特征提取和形状分析。为了得到较好的边缘效果，现在已经有了很多的边缘检测算法以及一些边缘检测算子的改进算法。但各算子有自己的优缺点和适用领域。基于以上理由，在此对学到的一些一些经典边缘检测算子进行实际编程验证，以便实际应用中更好地发挥其长处。

**三、设计思想**

1. **拉普拉斯算子**

拉普拉斯算子是二阶微分算子，其差分的表示形式如下：



拉普拉斯边缘检测算子的模板如下图所示，模板的基本特征是中心位置的系数为正，其余位置的系数为负，且模板的系数之和为零。它的使用方法是用图中的两个点阵之一作为卷积核，与原图像进行卷积运算即可。

|  |  |  |
| --- | --- | --- |
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

1. 索贝尔算子

该算子是由两个卷积核与对原图像进行卷积运算而得到的。用差分代替一阶偏导，算子的计算方法如下：

 Sobel算子垂直方向和水平方向的模板如下图所示，前者可以检测出图像中的水平方向的边缘，后者则可以检测图像中垂直方向的边缘。实际应用中，图像中的每一个像素点都用这两个卷积核进行卷积运算，取其最大值作为输出。运算结果是一幅体现边缘幅度的图像。

|  |  |  |
| --- | --- | --- |
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -3 | 0 | 1 |

（a） （b）

Sobel算子模板

3. 罗伯特（Robert）算子

Robert梯度算子所采用的是对角方向相邻两像素值之差，所以用差分代替一阶偏导，算子形式可表示如下：



上述算子对应的两个模板如下图所示。实际应用中，图像中的每个像素点都用这两个模板进行卷积运算，为避免出现负值，在边缘检测时常提取其绝对值。

|  |  |
| --- | --- |
| 1 | 0 |
| 0 | -1 |

|  |  |
| --- | --- |
| 0 | 1 |
| -1 | 0 |

（a） （b）

图 Robert算子模板

1. prewwit（普瑞维特）算子

用差分代替一阶偏导可得算子形式如下：

 Prewitt边缘检测算子的两个模板如下图所示，它的使用方法同Sobel算子一样，图像中的每个点都用这两个核进行卷积，取得最大值作为输出。Prewitt算子也产生一幅边缘图像。

|  |  |  |
| --- | --- | --- |
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

（a） （b）

Prewitt算子模板

1. 镜像，平移，颜色取反

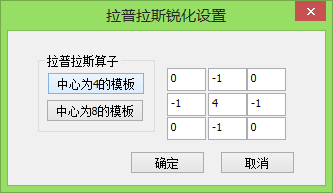
都是对原图像（矩阵）进行遍历

**四、数据结构与算法设计**

1. **拉普拉斯算子**

很明显，算子用矩阵表示，在c++里面是二维数组。

通过对话框确定使用哪个运算模板



通过调用模板运算函数实现

具体的伪代码如下：

* 获得doc类的指针对象，因为指向位图数据的句柄这个成员变量在doc类里面；
* 得到指向图像数据开始（不包含文件头）的指针，并且锁住图像所占用的那块内存；
* 判断是否是8位的bmp图片，如果不是，报错返回，如果是则继续
* 弹出选择具体模板的对话框，
* If(选择了模板，并且按下了确定按钮)

{

调用doc类里面dib类对象的模板运算函数；

If(函数模板运算成功)

{

图像设置脏标记；

迫使程序重绘

}

Else

报错，返回

}

**因为下面几个算子的情况大同小异，下面不再详述！**

1. **模板运算函数**

模板卷积在空域实现的主要步骤如下。

(1) 将模板在图中漫游，并将模板中心与图中某个像素位置重合

(2) 将模板上的各个系数与模板下各对应像 素的灰度值相乘

(3) 将所有乘积相加（为保持灰度范围，常将结果再除以模板的系数个数）

(4) 将上述运算结果（模板的输出响应）赋 给图中对应模板中心位置的像素

具体的伪码如下：

* 得到打开的图像的信息，计算每行的字节数；
* 通过已经打开的图像的信息，新申请一块内存，并将打开的图像数据复制进来，
* 得到指向新申请的内存的指针，锁定指针所指向的内存；

for (i = iTempMY; i < lHeight - 2\*iTempMY; i++) // 行(除去边缘几行)

{

for (j = iTempMX; j < lWidth - 2\*iTempMX; j++)// 列(除去边缘几列)

{

// 指向新DIB第i行，第j个象素的指针

//pTemp = (unsigned char\*)lpNewDIBBits + lLineBytes \* i + j;这样的话图像会倒置

pTemp = (unsigned char\*)lpNewDIBBits + byteofLine \* (lHeight - 1 - i) + j;// 保存当前像素点的灰度值

result = 0;

for (int k = 0; k < iTempH; k++)

{

for (int l = 0; l < iTempW; l++)

{

// 指向DIB第i - iTempMY + k行，第j - iTempMX + l个象素的指针

lpSrc = (unsigned char\*)lpDIBBits + byteofLine \* (lHeight - 1

- (i - iTempMY + k)) + (j - iTempMX + l);

// 保存象素值

result += (\*lpSrc) \* fpArray[k \* iTempW + l];

}

}

result \*= fCoef; // 乘上系数

result = (FLOAT)fabs(result); // 取绝对值

if (result > 255)//因为是256色图，超过直接赋值为255

{

\*pTemp = 255;

}

else

{

\*pTemp = (unsigned char)(result + 0.5);//四舍五入取整

}

}

}

将新开辟的内存里面的数据复制给以前的打开的图像所占的内存；

为新开辟的内存解锁；

释放掉新开辟的内存；

return TRUE;

}

**3. 索贝尔（sobel）算子**

对原图像进行两次模板运算，然后取两次模板运算的值中最大值作为新的图像的灰度值

1. **罗伯特（Robert）算子**

在此没有调用模板运算的函数，直接进行对角元素相减。

1. **Prewwit算子**

除了模板和sobel算子不一样，其余的全都一样

1. **平移，取反，镜像**

**平移：**

* 新申请一块内存，初始化为已经打开的图像；
* 平移是用两个for循环，遍历临时图像的每个点；
* 如果临时图像的点平移回去没有出界，则保留，否则，直接赋为255（出界的点直接剪切）；
* 将新申请的内存的内容复制到打开的图像所在的内存
* 重绘

**取反：**

* 新申请一块内存，初始化为已经打开的图像；
* 平移是用两个for循环，遍历临时图像的每个点，每个点的值=255-点的值；
* 将新申请的内存的内容复制到打开的图像所在的内存；
* 重绘

**镜像：**

垂直镜像的话，交换第一行的像素和倒数第一行的像素，交换第二行的像素和倒数第二行的像素·······；水平镜像同样的道理。

1. **运行结果与分析**
2. **程序运行结果**

**原图片**

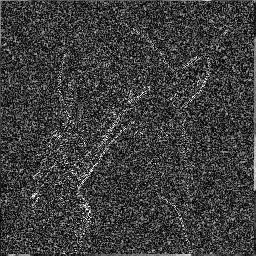
****

原始图片 有噪声的原始图片

1. **拉普拉斯算子**



**图一、中心为4的拉普拉斯 图二、 中心为8**

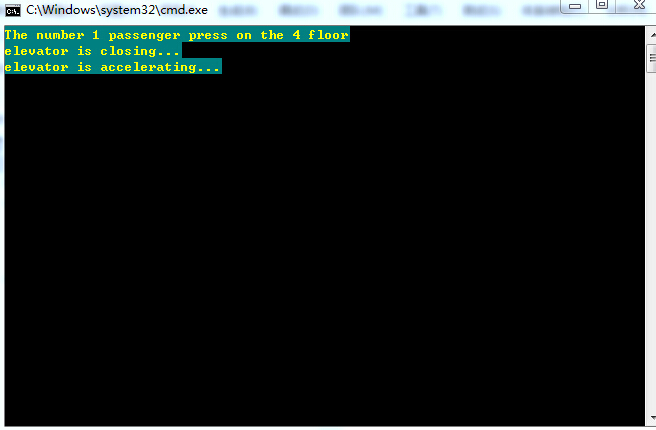


**图三、中心为4的 图四、中心为8的**

1. **索贝尔（Sobel）算子**



**图五 图六**



**3. 罗伯特（Robert）算子**



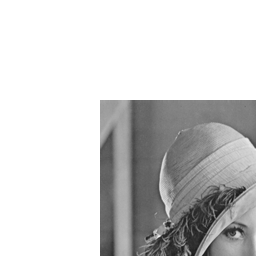
**图七 图八**

1. **Prewwit算子**



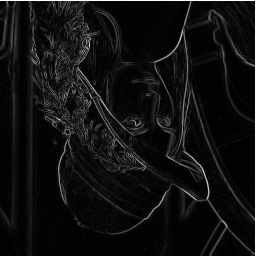
**图九 图十**

**5. 平移，取反，镜像**



**图十一 平移 图十二 取反**

****



**图十三 垂直镜像 图十四 水平镜像**

1. **分析**

* 图三四，和图六、八、十相比，很明显的得出结论，拉普拉斯算子对噪声敏感，在应用过程中应注意先进行平滑预处理。
* 图三、四、六、八、十纵向对比，可见Robert算子对噪声的抵抗能力最好，sobel和prewwit算子次之
* 比较图五六和图九十可见sobel算子和Prewwit算子几乎一样。

|  |  |  |
| --- | --- | --- |
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -3 | 0 | 1 |

（a） （b）

图 Sobel算子模板

|  |  |  |
| --- | --- | --- |
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

（a） （b）

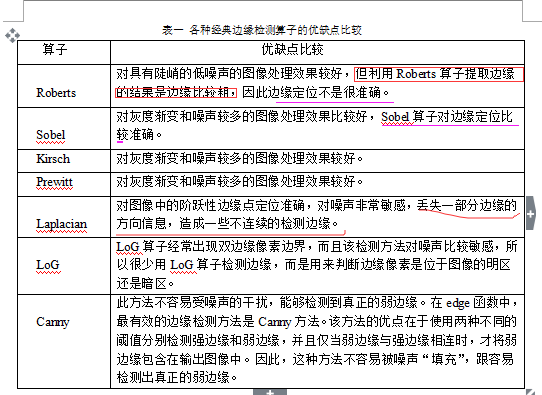
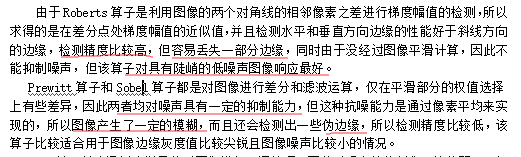
图 Prewitt算子模板

这一点从他们的模板中也可以看出一二

* 这几个算子中只有拉普拉斯算子是二阶算子，其余的均是一阶算子。但是也数拉普拉斯算子丢失边的信息最为严重。纵向对比图九七五二一，图一、二，也就是拉普拉斯算子的那两幅图，丢失lena身后的柱子信息（图最左面）最为严重，图七（Robert算子处理的那幅图）次之，剩余的保留信息最好。观察帽子顶的信息，图一和图七最差，图二次之，图五和图九最好。
* 很容易看出，sobel算子和prewwit算子检测出的边最为厚，尤其是sobel算子，Robert算子检测出的边最细。

目前，边缘检测技术在很多领域中都得到了广泛的应用。如对医学图像的边缘特征提取、车牌识别、人脸检测技术等。本文比较了几种常用的经典边缘检测算子。可以看出它们都不是具有绝对优势的方法，都各自存在优缺点。所以对于图像的边缘检测，要根据不同的系统，针对不同的环境条件和要求，选择合适的算子。

以上是我从自己处理的图像中所看出来的，并没有严谨的理论分析，实际测试的样本数量也不是很多，仅仅作为验证，结果也基本符合网上的别人的总结，但还是有出入的。通过对这几种算子的比较，加深了对这几种算子的印象，在以后的学习和工作中能比较准确的判断使用哪种算子，为将来的学习工作打下了基础。



（别人的总结）

注：红线的地方是结论基本一致的地方，画框是出入很大的地方，没划线的是我这里是没有结论的。

**六、困难及解决方法**

1. 自己的结果产生错误

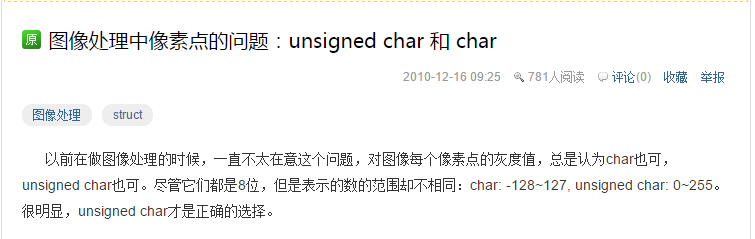
自己历尽千辛万苦好不容易出来一幅图片，没有产生中断而崩掉，激动之后，自己做的结果到底对不对啊？？

和同学做的结果对照一下，很容易发现不一样：



**（这两幅图片都是用prewitt算子处理的，sobel也有这样的问题）**

后来发现是有符号的char和无符号的char的问题。



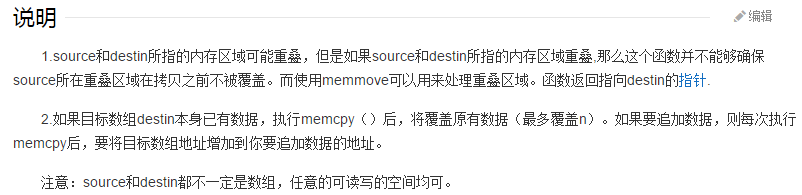
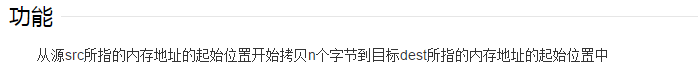
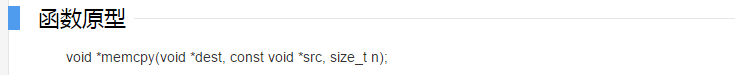
（<http://blog.csdn.net/huangkangying/article/details/6079157>）

百度之后，问题解决了。

1. MFC编程过程中的问题

很明显，在做课程设计的时候要用到内存的开辟，复制，释放等等。

在做这个的过程中，学到了一个新的函数memcpy



（百度百科）

但是在用的过程中很容易出现崩掉的情况，碰到这种情况之后只能自己检查数组在哪个地方越界。

还有一些“灵异事件”，程序崩了，怎么看怎么对，调试也没调出来，新建一个工程，重新拷贝一份，又好了······

（3）选题

  由于是自己选题，所以想找个简单点的，所以在空域里面做，选了上面的几个题目，课程里面涉及到的边缘检测算子里面还有两个，一个是马尔算子，用到了平滑，另一个是Canny算子，要用到高斯滤波器，所以这两个算子没做。其实我也想在频域里面做一下，感受一下高通滤波的效果，但是让傅里叶变换就给治住了，所以就放弃了。

（4）对这几个算子的理解

很明显从上面可以看出，我对这几个算子的理解还是很肤浅的，直接把它们的离散化的形式拿过来用，直接把卷积模板套用一下。我还是希望有机会，能推导一个算子的公式，加深一下理解。但是，通过自己在编码过程中的思考以及在代码中调试，也同样加深了自己对于该方法的理解，并且在程序中得到了很好地实现，收获还是很多的。

1. 查资料

很明显，去网上可以搜到很多的关于这方面的资料，但是有的是论文，看起来很难，有的呢是比较垃圾的。关键就靠我们自己，擦亮眼睛，去找难度适合的能满足我们需求的资料。

**七、总结心得**

通过一学期的课程学习我们虽说还没有完全掌握数字图像处理技术，但也收获了不少，对于数字图像方面的知识有了深入的了解，更加理解了数字图像的本质，即是一些数字矩阵，但灰度图像和彩色图像的矩阵形式是不同的。对于一些耳熟能详的数字图像相关术语有了明确的认识，比如常见的：像素（衡量图像的大小）、分辨率（衡量图像的清晰程度）、位图（放大后会失真）、矢量图（经过放大不会失真）等大家都能叫上口却知识模糊的名词。也了解图像处理技术中一些常用处理技术的实质，比如锐化处理是使模糊的图像变清晰，增强图像的边缘等细节。而平滑处理是的目的是消除噪声，模糊图像，在提取大目标之前去除小的细节或弥合目标间的缝隙。对常提的RGB图像和灰度图像有了明确的理解，这对大家以后应用PHOTOSHOP等图像处理软件对图像进行处理打下了坚实的基础，就像武侠小说里面的“内功”。更重要的是了解到了数字图像处理的思想。通过学习也是对C++编程应用的很好的实践与复习。

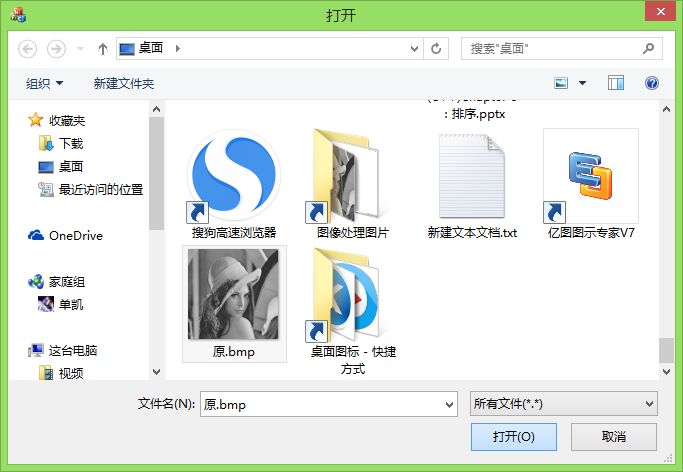
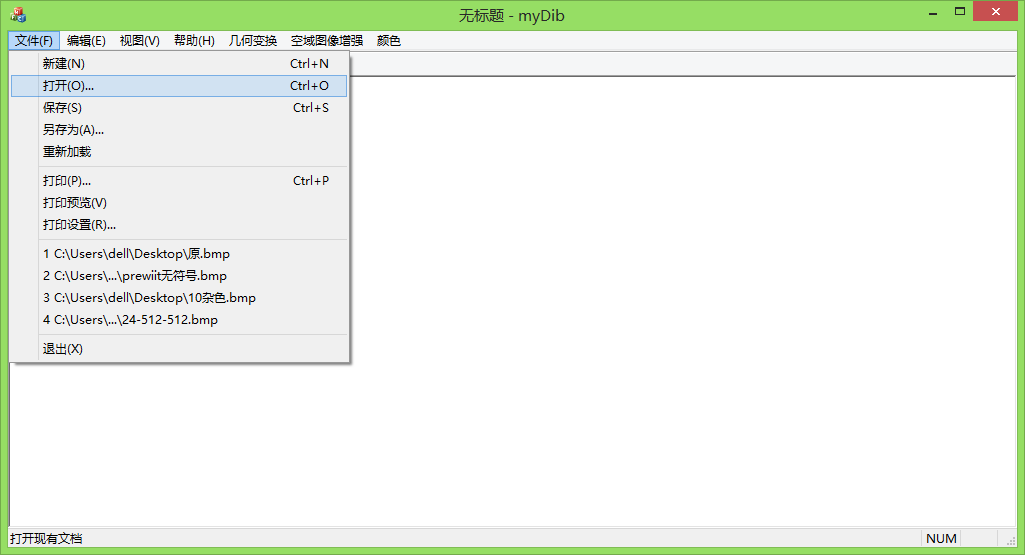
当然通过几周的的课程学习还是远远不够的，也有许多同学收获甚微，我总结了下大家后期的学习态度与前期的学习热情相差很大的原因。刚开始大家是有很高的热情学习这门课的，可是随着课程的逐渐深入学习，大家渐渐发现课程讲授内容与自己起初想学的实用图像处理技术是有很大的差别的，大家更着眼于如何利用一些软件、技术去处理图像而得到满意的效果，或者进行一些图像的创意设计，可是课程的内容更偏重于如何通过编程实现实现如何对图像进行一些类似于锐化、边缘提取、模糊、去除噪声等基础功能的实现，这其中涉及很多算法、函数，需要扎实的数学基础和编程基础，并且需要利用大量时间在课下编写代码，并用VISUAL C++软件实现并进行调试，然而大部分人的C++实践能力以及编程能力还有待提高，尤其是对于矩阵进行操作的编程尤为是个考验，并且后半学期课程任务较重，时间不是很充裕，这对于需要大量实践的数字图像处理课程就是个很大的问题。

在老师授课方面建议可以在课上多进行具体操作，这样可以提起大家学习的兴趣，也可以让大家在课下积极准备，然后在上课由学员进行演示，还可以加入一些数字图像处理的经典范例，加深同学们的学习热情。

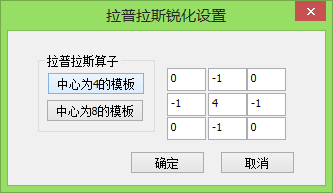
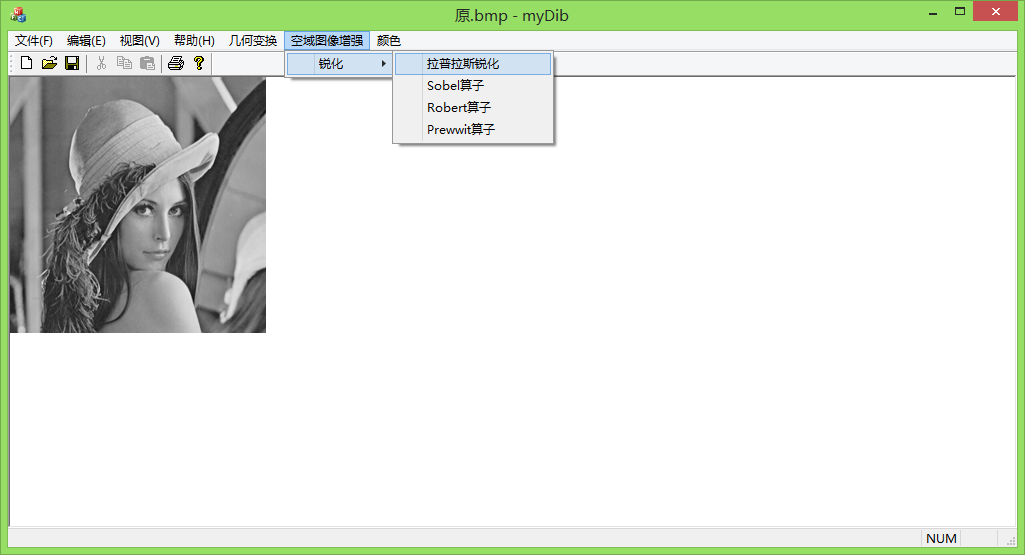
**八、程序操作说明及主要代码**

1. **程序操作的简要说明书：**

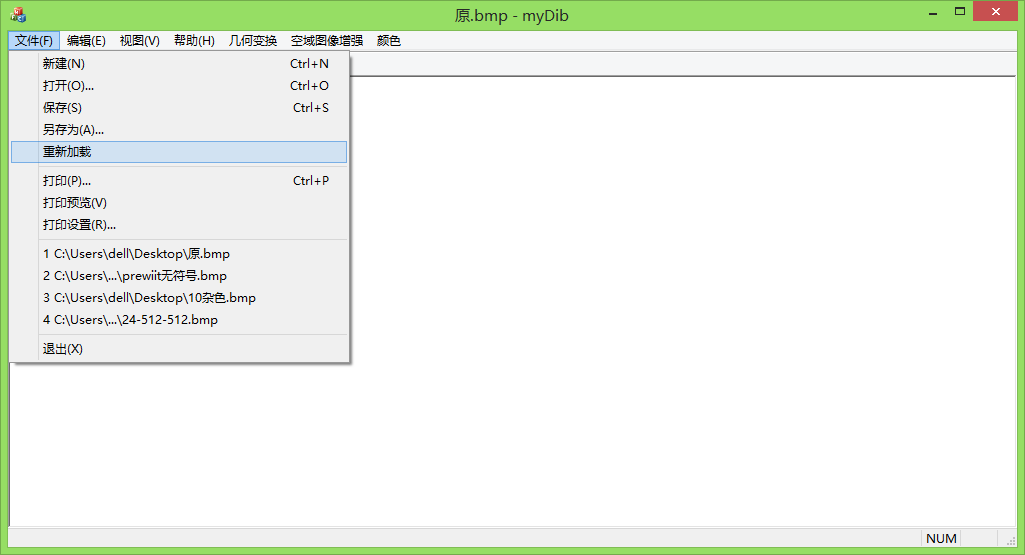
**① 打开一幅图片**



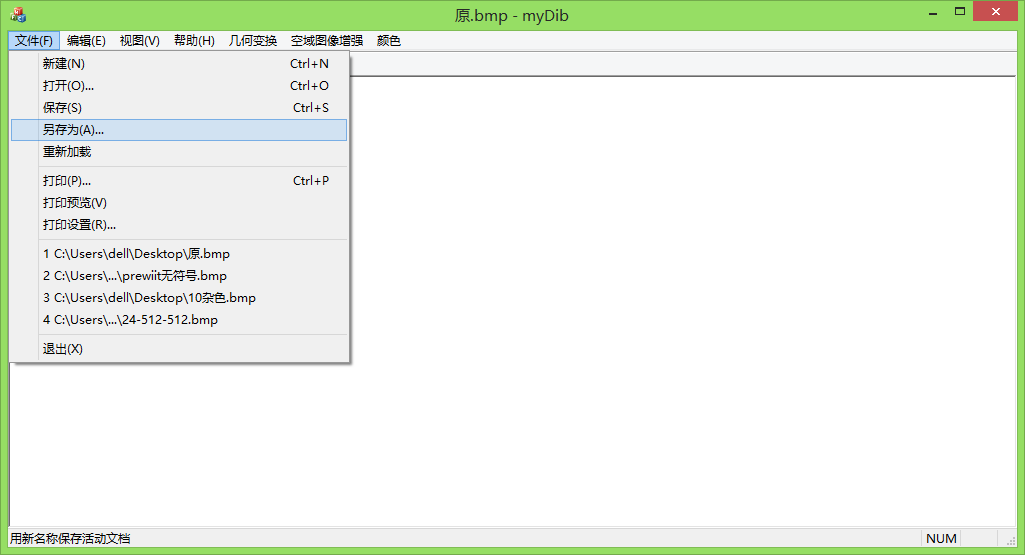
**② 从菜单选取操作**



**③ 如果要放弃所做的操作**



**④ 如果要保存所做的操作**



1. **主要代码**

//拉普拉斯算子菜单的响应函数

void CmyDibView::OnLplsSharp()

{

CmyDibDoc\* pDoc = GetDocument();

LPSTR lpDIB; // 指向DIB的指针

LPSTR lpDIBBits; // 指向DIB象素指针

int iTempH=3; // 模板高度

int iTempW=3; // 模板宽度

float fTempC=1; // 模板系数

int iTempMX=1; // 模板中心元素X坐标

int iTempMY=1; // 模板中心元素Y坐标

float aValue[9]; // 模板元素数组

lpDIB = (LPSTR) ::GlobalLock((HGLOBAL)pDoc->GetHDIB());

// 找到DIB图像象素起始位置

lpDIBBits = pDoc->GetDibImage()->FindDIBData(lpDIB);

// 判断是否是8-bpp位图

if (pDoc->GetDibImage()->DIBNumColors(lpDIB) != 256)

{

MessageBox(\_T("只支持256色位图的锐化！"), \_T("系统提示"),

MB\_ICONINFORMATION | MB\_OK);

::GlobalUnlock((HGLOBAL)pDoc->GetHDIB());

return;

}

CLPLSDlg dlg;

if (IDOK == dlg.DoModal())

{

aValue[0] = dlg.m\_n1; aValue[1] = dlg.m\_n2; aValue[2] = dlg.m\_n3;

aValue[3] = dlg.m\_n4; aValue[4] = dlg.m\_n5; aValue[5] = dlg.m\_n6;

aValue[6] = dlg.m\_n7; aValue[7] = dlg.m\_n8; aValue[8] = dlg.m\_n9;

// 调用Template()函数用拉普拉斯模板锐化DIB

if (pDoc->GetDibImage()->Template(lpDIBBits, pDoc->GetDibImage()->DIBWidth(lpDIB),

pDoc->GetDibImage()->DIBHeight(lpDIB), iTempH, iTempW, iTempMX,

iTempMY, aValue, fTempC))

{

pDoc->SetModifiedFlag(TRUE); // 设置脏标记

pDoc->UpdateAllViews(NULL); // 更新视图

}

else

{

MessageBox(\_T("分配内存失败！"), \_T("系统提示"), MB\_ICONINFORMATION | MB\_OK);

}

::GlobalUnlock((HGLOBAL)pDoc->GetHDIB());

}

}

**其他几个算子差不多，就不再在此复制粘贴代码！！！！！！！！**

//功能：空域上的模板运算

//参数:

// lpDIBBits 指向源DIB图像指针 lWidth 源图像宽度（象素数）

// lHeight 源图像高度（象素数） iTempH 模板的高度

// iTempW 模板的宽度 iTempMX 模板的中心元素X坐标 ( < iTempW - 1)

// iTempMY 模板的中心元素Y坐标 ( < iTempH - 1) fpArray 指向模板数组的指针

// FLOAT fCoef 模板系数

//返回值:成功返回TRUE，否则返回FALSE。

bool CMyDib::Template(LPSTR lpDIBBits, long lWidth, long lHeight,

int iTempH, int iTempW, int iTempMX, int iTempMY,

float \* fpArray, float fCoef)

{

LPSTR lpNewDIBBits; // 指向复制图像的指针

HLOCAL hNewDIBBits; //指向一块内存区域的指针

unsigned char\* lpSrc; // 指向源图像的指针

unsigned char\* pTemp; // 指向要重新赋值区域的像素点指针

float result; // 计算结果

long byteofLine; // 图像每行的字节数

byteofLine = LineBytes(lWidth \* 8);

// 暂时分配内存，以保存新图像

hNewDIBBits = LocalAlloc(LHND, byteofLine \* lHeight);

if (hNewDIBBits == NULL)

{

return FALSE;

}

lpNewDIBBits = (char \*)LocalLock(hNewDIBBits);

// 初始化图像为原始图像

memcpy(lpNewDIBBits, lpDIBBits, byteofLine \* lHeight);

for (int i = iTempMY; i < lHeight - 2\*iTempMY; i++) // 行(除去边缘几行)

{

for (int j = iTempMX; j < lWidth - 2\*iTempMX; j++)// 列(除去边缘几列)

{

// 指向新DIB第i行，第j个象素的指针

//lpDst = (unsigned char\*)lpNewDIBBits + lLineBytes \* i + j;

//这样的话图像会倒置

pTemp = (unsigned char\*)lpNewDIBBits + byteofLine \* (lHeight - 1 - i) + j;

result = 0;

for (int k = 0; k < iTempH; k++)

{

for (int l = 0; l < iTempW; l++)

{

// 指向DIB第i - iTempMY + k行，第j - iTempMX + l个象素的指针

lpSrc = (unsigned char\*)lpDIBBits + byteofLine \* (lHeight - 1

- (i - iTempMY + k)) + (j - iTempMX + l);

// 保存象素值

result += (\*lpSrc) \* fpArray[k \* iTempW + l];

}

}

result \*= fCoef; // 乘上系数

result = (FLOAT)fabs(result); // 取绝对值

if (result > 255)

{

\*pTemp = 255;

}

else

{

\*pTemp = (unsigned char)(result + 0.5);

}

}

}

// 复制变换后的图像

memcpy(lpDIBBits, lpNewDIBBits, byteofLine \* lHeight);

LocalUnlock(hNewDIBBits);

LocalFree(hNewDIBBits);

return TRUE;

}

//功能：该函数用来镜像DIB图像。可以指定镜像的方式是水平还是垂直。

//参数:

// LPSTR lpDIBBits 指向源DIB图像指针

// LONG lWidth 源图像宽度（象素数）

// LONG lHeight 源图像高度（象素数）

// BOOL bDirection 镜像的方向，TRUE表示水平镜像，FALSE表示垂直镜像

//返回值:镜像成功返回TRUE，否则返回FALSE。

bool CMyDib::MirrorDIB(LPSTR lpDIBBits, LONG lWidth, LONG lHeight,

BOOL bDirection)

{

LPSTR lpSrc; // 指向源图像的指针

LPSTR lpDst; // 指向要复制区域的指针

LPSTR lpBits; // 指向复制图像的指针

HLOCAL hBits;

LONG i; // 循环变量

LONG j;

LONG lLineBytes; // 图像每行的字节数

lLineBytes = LineBytes(lWidth \* 8);// 计算图像每行的字节数

hBits = LocalAlloc(LHND, lLineBytes); // 暂时分配内存，以保存一行图像

if (hBits == NULL)

{

return FALSE;

}

lpBits = (char \*)LocalLock(hBits);

// 判断镜像方式

if (bDirection) // 水平镜像

{

// 针对图像每行进行操作

for (i = 0; i < lHeight; i++)

{

// 针对每行图像左半部分进行操作

for (j = 0; j < lWidth / 2; j++)

{

// 指向倒数第i行，第j个象素的指针

lpSrc = (char \*)lpDIBBits + lLineBytes \* i + j;

// 指向倒数第i行，倒数第j个象素的指针

lpDst = (char \*)lpDIBBits + lLineBytes \* (i + 1) - j;

// 备份一个象素

\*lpBits = \*lpDst;

// 将倒数第i行，第j个象素复制到倒数第i行，倒数第j个象素

\*lpDst = \*lpSrc;

// 将倒数第i行，倒数第j个象素复制到倒数第i行，第j个象素

\*lpSrc = \*lpBits;

}

}

}

else // 垂直镜像

{

// 针对上半图像进行操作

for (i = 0; i < lHeight / 2; i++)

{

// 指向倒数第i行象素起点的指针

lpSrc = (char \*)lpDIBBits + lLineBytes \* i;

// 指向第i行象素起点的指针

lpDst = (char \*)lpDIBBits + lLineBytes \* (lHeight - i - 1);

// 备份一行，宽度为lWidth

memcpy(lpBits, lpDst, lLineBytes);

// 将倒数第i行象素复制到第i行

memcpy(lpDst, lpSrc, lLineBytes);

// 将第i行象素复制到倒数第i行

memcpy(lpSrc, lpBits, lLineBytes);

}

}

LocalUnlock(hBits);

LocalFree(hBits);

return TRUE;

}

//功能：水平移动DIB图像。

函数不会改变图像的大小，移出的部分图像

// 将截去，空白部分用白色填充。

//参数：lpDIBBits 指向源DIB图像指针

// lWidth 源图像宽度（象素数）

// lHeight - 源图像高度（象素数）

// lXOffset - X轴平移量（象素数）

// lYOffset - Y轴平移量（象素数）

//返回值：平移成功返回TRUE，否则返回FALSE。

bool CMyDib::TranslationDIB1(LPSTR lpDIBBits, long lWidth, long lHeight,

long lXOffset, long lYOffset)

{

LPSTR lpSrc; // 指向源图像的指针

LPSTR lpDst; // 指向要复制区域的指针

LPSTR lpNewDIBBits; // 指向复制图像的指针

HLOCAL hNewDIBBits;

LONG i; // 象素在新DIB中的坐标

LONG j;

LONG i0; // 象素在源DIB中的坐标

LONG j0;

LONG lLineBytes; // 图像每行的字节数

lLineBytes = LineBytes(lWidth \* 8);// 计算图像每行的字节数

// 暂时分配内存，以保存新图像

hNewDIBBits = LocalAlloc(LHND, lLineBytes \* lHeight);

if (hNewDIBBits == NULL)

{

return FALSE;

}

lpNewDIBBits = (char \*)LocalLock(hNewDIBBits);

for (i = 0; i < lHeight; i++) // 每行

{

for (j = 0; j < lWidth; j++) // 每列

{

// 指向新DIB第i行，第j个象素的指针

// 注意由于DIB中图像第一行其实保存在最后一行的位置，因此lpDst

// 值不是(char \*)lpNewDIBBits + lLineBytes \* i + j，而是

// (char \*)lpNewDIBBits + lLineBytes \* (lHeight - 1 - i) + j

lpDst = (char \*)lpNewDIBBits + lLineBytes \* (lHeight - 1 - i) + j;

// 计算该象素在源DIB中的坐标

i0 = i - lXOffset;

j0 = j - lYOffset;

/\*i0 = i;

j0 = j;\*/

// 判断是否在源图范围内

if ((j0 >= 0) && (j0 < lWidth) && (i0 >= 0) && (i0 < lHeight))

{

// 指向源DIB第i0行，第j0个象素的指针

// 同样要注意DIB上下倒置的问题

lpSrc = (char \*)lpDIBBits + lLineBytes \* (lHeight - 1 - i0) + j0;

\*lpDst = \*lpSrc; // 复制象素

}

else

{

//对于源图中没有的象素，直接赋值为255

\*((unsigned char\*)lpDst) = 255;

}

}

}

// 复制平移后的图像

memcpy(lpDIBBits, lpNewDIBBits, lLineBytes \* lHeight);

LocalUnlock(hNewDIBBits);

LocalFree(hNewDIBBits);

return TRUE;

}

//取反菜单的响应函数

void CmyDibView::OnAddcolor()

{

CmyDibDoc\* pDoc = GetDocument();

LPSTR lpDIB; // 指向DIB的指针

LPSTR lpDIBBits; // 指向DIB象素指针

lpDIB = (LPSTR) ::GlobalLock((HGLOBAL)pDoc->GetHDIB());

// 找到DIB图像象素起始位置

lpDIBBits = pDoc->GetDibImage()->FindDIBData(lpDIB);

int imgWidth = pDoc->GetDibImage()->DIBWidth(lpDIB);

int ImgHeight = pDoc->GetDibImage()->DIBHeight(lpDIB);

int byteofLine = LineBytes(imgWidth \* 8);

LPSTR lpNewDIBBits; // 指向复制图像的指针

HLOCAL hNewDIBBits; //指向一块内存区域的指针

// 暂时分配内存，以保存新图像

hNewDIBBits = LocalAlloc(LHND, byteofLine \* ImgHeight);

if (hNewDIBBits == NULL)

{

return ;

}

lpNewDIBBits = (char \*)LocalLock(hNewDIBBits);

// 初始化图像为原始图像

memcpy(lpNewDIBBits, lpDIBBits, byteofLine \* ImgHeight);

for (int y = 0; y<ImgHeight; y++)

{

for (int x = 0; x<imgWidth; x++)

{

lpNewDIBBits[y\*imgWidth + x] = 255 - lpNewDIBBits[y\*imgWidth + x];

}

}

memcpy(lpDIBBits, lpNewDIBBits, byteofLine \* ImgHeight);

LocalUnlock(hNewDIBBits);

LocalFree(hNewDIBBits);

Invalidate();

}

1. **参考材料**

[**http://blog.csdn.net/lanbing510/article/details/8176231**](http://blog.csdn.net/lanbing510/article/details/8176231)

[**http://www.articleswriting.net/article/3474918632/**](http://www.articleswriting.net/article/3474918632/)

[**http://blog.csdn.net/liygcheng/article/details/12854269**](http://blog.csdn.net/liygcheng/article/details/12854269)

[**http://wenku.baidu.com/link?url=39qMm0LoZNMMIEmEp-si6skGGxNV5RP\_SIYA9c9815jAyei9XJ6ztAN0YF1NCNDHhDNOjvMuCBgit1Pym1ATQG\_DzQf8JKZD6dida8hstVa**](http://wenku.baidu.com/link?url=39qMm0LoZNMMIEmEp-si6skGGxNV5RP_SIYA9c9815jAyei9XJ6ztAN0YF1NCNDHhDNOjvMuCBgit1Pym1ATQG_DzQf8JKZD6dida8hstVa)

[**http://blog.csdn.net/huangkangying/article/details/6079157**](http://blog.csdn.net/huangkangying/article/details/6079157)

[**http://blog.csdn.net/lovemysea/article/details/5275612**](http://blog.csdn.net/lovemysea/article/details/5275612)

[**http://blog.sina.com.cn/s/blog\_77894dad0101p7pp.html**](http://blog.sina.com.cn/s/blog_77894dad0101p7pp.html)

**课本、PPT等。**