|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | | |
|  | | | | |
|  | | | | |
| 课程设计报告 | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | 设计题目： | 数字图像处理 | | |
|  | 学生姓名： | 孙淼 |  |  |
|  | 专 业： | 计算机科学与技术 |  |  |
|  | 班 级： | 18级2班 |  |  |
|  | 学 号： | 2018211958 |  |  |
|  | 指导教师： | 郭丹 |  |  |
|  | 完成日期： | 2021.5.29 |  |  |
|  |  |  |  |  |
| **合肥工业大学计算机与信息学院** | | | | |

目录

[一、需求和规格说明 3](#_Toc75889230)

[二、设计 4](#_Toc75889231)

[2.1）设计思想 4](#_Toc75889232)

[2.2）设计表示 8](#_Toc75889233)

[2.3）实现注释 11](#_Toc75889234)

[2.4）详细设计表示 12](#_Toc75889235)

[三、用户手册 19](#_Toc75889236)

[四、调试及测试 20](#_Toc75889237)

[4.1）主要问题 20](#_Toc75889238)

[4.2）程序运行的时空效率分析 25](#_Toc75889239)

[4.3）测试数据集 27](#_Toc75889240)

[4.4）运行实例 28](#_Toc75889241)

[4.5）改进设想 31](#_Toc75889242)

[4.6）对设计和编码的回顾讨论和分析、经验和体会 34](#_Toc75889243)

[五、选做部分 35](#_Toc75889244)

[5.1）平滑算子 35](#_Toc75889245)

[5.2）锐化算子 40](#_Toc75889246)

[5.3）图形界面 50](#_Toc75889247)

[六、附录 57](#_Toc75889248)

[6.1）源程序清单（文件排序按首字母排序） 57](#_Toc75889249)

[6.2）测试数据 88](#_Toc75889250)

[6.3）运行结果 90](#_Toc75889251)

# 一、需求和规格说明

一幅图像是包含位置集和颜色集的数据。考虑二维灰度图像，位置集就是一个矩阵的行列，矩阵的内容为颜色值，颜色为0~255间的整数，表示该位置的灰度等级，0为黑色，255为白色。

图像处理就是与该矩阵相关的计算，一种常见的计算就是通过一点和周围8个点的信息，共同决定该点的新值：如一点的新值为该点和周围8点颜色之和的平均，这一操作可用图1表示。

|  |  |  |
| --- | --- | --- |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

图1 平滑操作示意图

这样处理后图像会变得平滑，因此，称为平滑操作。

如果将上述操作变为图2：

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

图2 锐化操作后示意图

操作后图像的边缘变得更加突出，被称为锐化操作。

实现上述图像的平滑和锐化操作。

编程任务：

1）常见格式图像的读写(灰度图)

2）设计并实现上述平滑算子和锐化算子

3）（选作）查阅其他的平滑和锐化算子并实现

1. 观察处理后图像的变化，分析算子的作用。

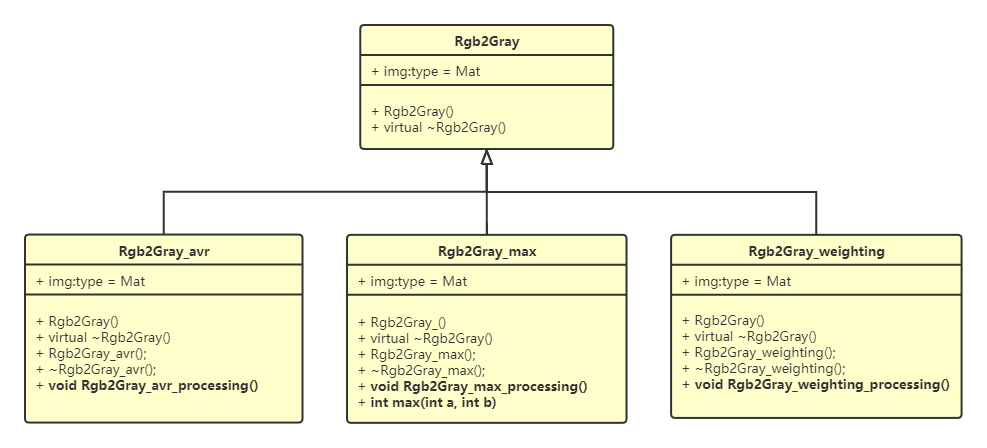
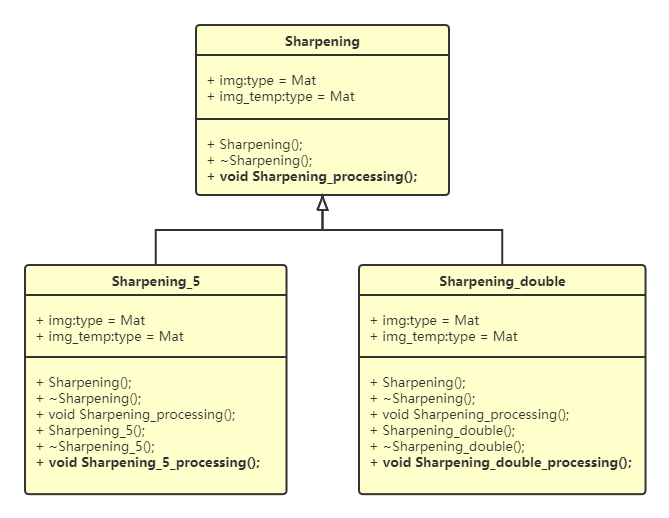
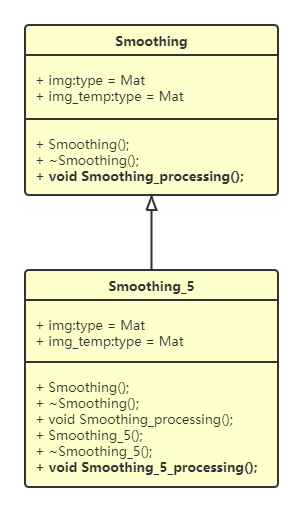
*考虑到四个编程任务的逻辑性，我们将任务3“（选做）查阅其他的平滑和锐化算子并实现”的相关工作作为第六部分-选做部分单独列出介绍。*

# 二、设计

## 2.1）设计思想

*程序结构（如类图），重要的数据结构。主要算法思想（文字描述，不要画框图）*

**2.1.1）对于必做部分，本程序主要设计了三个父类，其相关子类及关系如下面三个类图所示：**

* Rgb2Gray为父类的类图
* Sharpening为父类的类图
* Smoothing为父类的类图

**2.1.2）对于必做部分，本程序主要采用了如下数据结构：**

* cv::Mat Class Reference

<https://docs.opencv.org/4.0.0/d3/d63/classcv_1_1Mat.html#afbf34be80b7443123b2a032a67c3c4e3%29>

主要通过Mat来实现对jpg图像的读取，对图像上每个（rows，cols）位置上像素点的处理，通过

mat.at<Vec3b>(i, j)[k]

来完成对每个像素点的三个通道的值处理；

* 数组

主要通过数组来存储算子，如对于锐化操作，就可以通过下面语句

1. **int** oper[3][3] = { { -1,-1,-1 },{ -1,9,-1 },{ -1,-1,-1 } };

来完成锐化算子的存储，这样能够大大方便我们后续锐化操作的设计。

* + 1. **对于必做部分，本程序主要运用了如下的算法思想：**

对于RGB图转灰度图，由于是对每个像素点单独处理，不像平滑和锐化涉及到周围的像素点，所以我们只需要开辟一个Mat img即可对每个像素点单独处理。

对于均值灰度化，我们读取每个像素点的三个通道值，相加除3取得均值，再重新依次赋值给该像素点的三个通道；

1. **for** (**int** k = 0; k < 3; k++) {
2. img.at<Vec3b>(i, j)[k] = (img.at<Vec3b>(i, j)[0] + img.at<Vec3b>(i, j)[1] + img.at<Vec3b>(i, j)[2]) / 3;
3. }

对于最大值灰度化，我们首先写一个判断最大值函数int max(int a,int b）。

1. **int** max(**int** a, **int** b) {
2. **if** (a >= b)
3. **return** a;
4. **return** b;
5. }

然后通过这个函数，找出每个像素点的三个通道值中的最大值，然后重新依次赋值给该像素点的三个通道；

1. **for** (**int** k = 0; k < 3; k++) {
2. img.at<Vec3b>(i, j)[k] = max(max(img.at<Vec3b>(i, j)[0], img.at<Vec3b>(i, j)[1]), img.at<Vec3b>(i, j)[2]);
3. }

对于加权法灰度化，我们读取每个像素点的三个通道值，配合已知的权值计算得到对应的灰度值，然后重新依次赋值给该像素点的三个通道；

1. **for** (**int** k = 0; k < 3; k++) {
2. img.at<Vec3b>(i, j)[k] = img.at<Vec3b>(i, j)[0] \* 0.30 + img.at<Vec3b>(i, j)[1] \* 0.59 + img.at<Vec3b>(i, j)[2] \* 0.11;
3. }

对于锐化，首先通过数组oper限制出锐化算子，然后从开始读取每行每列的读取img的每个像素点，当该像素点所处的位置（当前像素点始终处于算子的中心位置）使得算子的一部分在图像之外时，就排除掉在外那部分算子，在内的那部分算子依旧正常运算，具体运算方法就是，将算子中相对于当前像素点的位置的值乘以图像中相对于当前像素点的像素点的值，得到temp值，需要注意的是，当temp大于255，则将其置为255，当temp小于0，就将其置为0。最终将这个temp值依次赋值给img\_temp的三个通道，得到锐化后的图像即为img\_temp。

1. **int** temp = 0;//用来计算新的中心像素点值
2. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
3. **for** (**int** l = 0; l < 3; l++) {
4. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
5. **continue**;
6. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
7. }
8. }
9. **if** (temp > 255) {
10. temp = 255;
11. }
12. //当计算结果小于0，就把它置为0
13. **if** (temp < 0) {
14. temp = 0;
15. }
16. **for** (**int** m = 0; m < 3; m++) {
17. img\_temp.at<Vec3b>(i, j)[m] = temp;
18. }

对于题目中设计的均值平滑，我们设计算子时需要注意不要设为一个3 \* 3且数组内的每个值都是1/9的二维数组，，因为如果将其内每个值设为1/9（小数），就需要考虑数据类型之间的转换，加大程序编写的复杂度，所以我们设计的算子是3 \* 3的二维数组，其中每个值都是1，这样的话，算子就是如下所示：

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

\* 1 / 9；也就是说，我们只需在最后除以算子的尺寸3 \* 3即可。

1. **int** temp = 0;//用来计算新的中心像素点值
2. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
3. **for** (**int** l = 0; l < 3; l++){
4. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时，排除图像中不存在的点
5. **continue**;
6. temp += (oper[k][l] + 1) \*img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
7. }
8. }
9. temp /= (3 \* 3);//之所以放后面是因为1 / 9 的影响
10. //将计算结果赋给dstimg中心像素点的三通道值
11. **for** (**int** m = 0; m < 3; m++) {
12. img\_temp.at<Vec3b>(i, j)[m] = temp;
13. }

对于其他几个子类的方法，如5 \* 5尺寸的平滑算子、锐化算子，我们只需要改变k，l，x值即可，其实也可以把函数修改成一个带参数的函数，这样就可以实现任何整数型的锐化算子，各个非Mat的参数值如下。

1. k = 算子的宽度
2. l = 算子的高度
3. x = (算子的尺寸 + 1) / 2

## 2.2）设计表示

*类名及其作用，类中数据成员名称及作用，类中成员函数原型及其功能，可以用表格形式表达。*

对于必做部分，我设计的类名及其作用，类中数据成员名称及作用，类中成员函数原型及其功能如下表格所示：

表1 Rgb2Gray主类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Rgb2Gray | 数据成员 | Mat | img | 图像的像素点矩阵 |
| 成员函数 |  | Rgb2Gray() | 无参构造函数 |
|  | ~Rgb2Gray() | 析构函数 |

其子类如下：

表1-1 Rgb2Gray\_avr子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Rgb2Gray\_avr | 数据成员 | Mat | img | 图像的像素点矩阵 |
| 成员函数 |  | Rgb2Gray() | 无参构造函数 |
|  | ~Rgb2Gray() | 析构函数 |
| void | Rgb2Gray\_avr\_processing() | 均值灰度化方法 |

表1-2 Rgb2Gray\_max子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Rgb2Gray\_max | 数据成员 | Mat | img | 图像的像素点矩阵 |
| 成员函数 |  | Rgb2Gray() | 无参构造函数 |
|  | ~Rgb2Gray() | 析构函数 |
| void | Rgb2Gray\_max\_processing() | 最大值灰度化方法 |

表1-3 Rgb2Gray\_weighting子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Rgb2Gray\_weighting | 数据成员 | Mat | Img | 图像的像素点矩阵 |
| 成员函数 |  | Rgb2Gray() | 无参构造函数 |
|  | ~Rgb2Gray() | 析构函数 |
| void | Rgb2Gray\_weighting\_processing() | 加权灰度化方法 |

表2 Sharpening主类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Sharpening | 数据成员 | Mat | img | 图像的像素点矩阵 |
| Mat | img\_temp | 用于存储新图像的像素点矩阵 |
| 成员函数 |  | Sharpening() | 无参构造函数 |
|  | ~Sharpening() | 析构函数 |
| void | Sharpening\_processing() | 题目设计的的3 \* 3锐化处理 |

其子类如下：

表2-1 Sharpening\_5子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Sharpening\_5 | 数据成员 | Mat | img | 图像的像素点矩阵 |
| Mat | img\_temp | 用于存储新图像的像素点矩阵 |
| 成员函数 |  | Sharpening\_5() | 无参构造函数 |
|  | ~Sharpening\_5() | 析构函数 |
| void | Sharpening\_5\_processing() | 自己设计的的5 \* 5锐化处理 |

表2-2 Sharpening\_double子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Sharpening\_double | 数据成员 | Mat | img | 图像的像素点矩阵 |
| Mat | img\_temp | 用于存储新图像的像素点矩阵 |
| 成员函数 |  | Sharpening\_double() | 无参构造函数 |
|  | ~ Sharpening\_double() | 析构函数 |
| void | Sharpening\_double\_processing() | 自己设计的的在3 \* 3基础上加倍算子的锐化处理 |

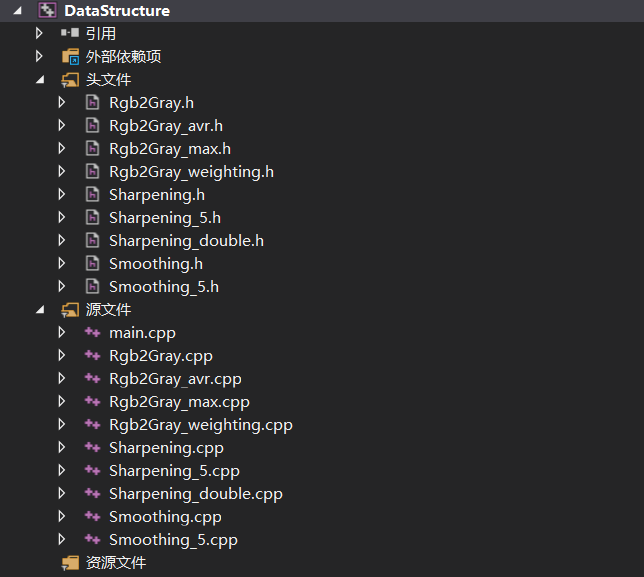
表3 Smoothing父类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Smoothing | 数据成员 | Mat | img | 图像的像素点矩阵 |
| Mat | img\_temp | 用于存储新图像的像素点矩阵 |
| 成员函数 |  | Smoothing() | 无参构造函数 |
|  | ~Smoothing() | 析构函数 |
| void | Smoothing\_processing() | 题目设计的3 \* 3均值平滑处理 |

其子类如下：

表3-1 Smoothing\_5子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Smoothing\_5 | 数据成员 | Mat | Img | 图像的像素点矩阵 |
| Mat | img\_temp | 用于存储新图像的像素点矩阵 |
| 成员函数 |  | Smoothing\_5() | 无参构造函数 |
|  | ~Smoothing\_5() | 析构函数 |
| void | Smoothing\_5\_processing() | 自己设计的5 \* 5均值平滑处理 |

必做部分最终的程序文件目录如下图所示：

## 2.3）实现注释

*实现注释：各项要求的实现程度、在完成基本要求的基础上还实现了什么功能？*

* 1. 常见格式图像的读写(灰度图)

答：完全实现，对彩色图像转化为灰度图，我使用了三种方法：最大值法、平均值法、加权平均值法，并对视觉上的区别进行对比介绍。

* 1. 设计并实现上述平滑算子和锐化算子

答：完全实现，并且设计的平滑、锐化算子可以支持参数化设置多种尺寸，其中锐化算子可以支持自选算子作为参数输入。

3）（选作）查阅其他的平滑和锐化算子并实现；

答：将在第六部分进行这部分工作的介绍。

4）可视化展示处理前后图像的变化，并尝试进行分析。

答：可以看到，本课设的平滑算子使图像平滑是基于邻域平均法，其思想是通过一点和邻域内像素点求平均来去除突变的像素点，从而滤掉一定噪声，其优点是算法简单，计算速度快，其代价会造成图像在一定程度上的模糊。在第五部分我们会发现均值邻域的大小与平滑的效果直接相关，邻域越大平滑的效果越好，但邻域过大，平滑会使边缘信息损失的越大，从而使输出的图像变得模糊，因此需合理选择邻域的大小。

图像锐化是补偿图像的轮廓，增强图像的边缘及灰度跳变的部分，使图像变得清晰，这种处理方法提高了地物边缘与周围像元之间的反差，因此也被称为边缘增强。实质上是要增强原始图像的高频成分。它强调灰度的突变，可以增强图像的细节。

后面将要拓展的中值滤波是将像素（中值计算中包括的原像素值）邻域内灰度的中值代替该像素的值。中值滤波器的使用非常普遍，这是因为对于一定类型的随机噪声，它提供了优秀的去噪能力，比小尺寸的线性平滑滤波器的模糊程度明显要低。虽然说中值滤波器对脉冲信号很有效，但是仅采用简单的中值滤波将不可避免的丢失图像的细节，造成视觉效果的模糊。

*此外，为了更好地完成这部分“实现注释”的要求，对于灰度化操作，我重新设计了一个带参数的灰度化方法，实现了以字符串为参数的三种灰度化合一；对于锐化操作，我重新设计了一个带参数的锐化方法，进而实现了算子内容可调；对于平滑操作，我重新设计了一个带参数的平滑方法，进行实现了尺寸可调，详细设计将在下面“4）详细设计表示”中介绍。*

## 2.4）详细设计表示

*详细设计表示：主要算法的框架及实现此算法的成员函数接口。*

为起到举一反三的作用，此处介绍的平滑和锐化类是新编写的含参子类。

它们的“类名及其作用，类中数据成员名称及作用，类中成员函数原型及其功能，可以用表格形式表达”如下。

表1-4 Rgb2Gray\_param子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Rgb2Gray\_max | 数据成员 | Mat | img | 图像的像素点矩阵 |
| 成员函数 |  | Rgb2Gray() | 无参构造函数 |
|  | ~Rgb2Gray() | 析构函数 |
| void | Rgb2Gray\_param\_processing(String way) | 含参的灰度化方法，当参数way为“avr”时，为均值灰度化方法；当参数way为“max”时，为最大值灰度化方法；当参数way为“weighting”时，为加权灰度化方法 |

其头函数Rgb2Gray\_param.h为：

1. #pragma once
2. #include "Rgb2Gray.h"
3. **class** Rgb2Gray\_param :
4. **public** Rgb2Gray
5. {
6. **public**:
7. Rgb2Gray\_param();
8. ~Rgb2Gray\_param();
9. **void** Rgb2Gray\_param\_processing(String way);
10. };

其源文件Sharpening\_param.cpp：

1. #include "Sharpening\_param.h"
3. Sharpening\_param::Sharpening\_param()
4. {
5. }
7. Sharpening\_param::~Sharpening\_param()
8. {
9. }
11. **void** Sharpening\_param::Sharpening\_param\_processing(**int** oper[3][3]){
12. //由于n = 3 所以 x = 1
13. //int oper[3][3] = { { -1,-1,-1 },{ -1,9,-1 },{ -1,-1,-1 } };
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 3; l++) {
19. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
22. }
23. }
24. **if** (temp > 255) {
25. temp = 255;
26. }
27. //当计算结果小于0，就把它置为0
28. **if** (temp < 0) {
29. temp = 0;
30. }
31. **for** (**int** m = 0; m < 3; m++) {
32. img\_temp.at<Vec3b>(i, j)[m] = temp;
33. }
34. }
35. }
36. }

表2-3 Sharpening\_param子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Sharpening\_param | 数据成员 | Mat | img | 图像的像素点矩阵 |
| Mat | img\_temp | 用于存储新图像的像素点矩阵 |
| 成员函数 |  | Sharpening\_double() | 无参构造函数 |
|  | ~ Sharpening\_double() | 析构函数 |
| void | Sharpening\_param\_processing(int oper[3][3]) | 可以自定义任意3 \* 3锐化算子的锐化处理方法，其中参数为锐化算子数组 |

其头函数Sharpening\_param.h为：

1. #pragma once
2. #include "Sharpening.h"
3. **class** Sharpening\_param :
4. **public** Sharpening
5. {
6. **public**:
7. Sharpening\_param();
8. ~Sharpening\_param();
9. **void** Sharpening\_param\_processing(**int** oper[3][3]);
10. };

其源文件Sharpening\_param.cpp为：

1. #include "Sharpening\_param.h"
3. Sharpening\_param::Sharpening\_param()
4. {
5. }
7. Sharpening\_param::~Sharpening\_param()
8. {
9. }
11. **void** Sharpening\_param::Sharpening\_param\_processing(**int** oper[3][3]){
12. //由于n = 3 所以 x = 1
13. //int oper[3][3] = { { -1,-1,-1 },{ -1,9,-1 },{ -1,-1,-1 } };
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 3; l++) {
19. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
22. }
23. }
24. **if** (temp > 255) {
25. temp = 255;
26. }
27. //当计算结果小于0，就把它置为0
28. **if** (temp < 0) {
29. temp = 0;
30. }
31. **for** (**int** m = 0; m < 3; m++) {
32. img\_temp.at<Vec3b>(i, j)[m] = temp;
33. }
34. }
35. }
36. }
37. 表3-2 Smoothing\_param子类

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **类名** | **成员类别** | **类型** | **成员名** | **描述** |
| Smoothing\_param | 数据成员 | Mat | Img | 图像的像素点矩阵 |
| Mat | img\_temp | 用于存储新图像的像素点矩阵 |
| 成员函数 |  | Smoothing\_5() | 无参构造函数 |
|  | ~Smoothing\_5() | 析构函数 |
| void | Smoothing\_param\_processing() | 自己设计的可以自定义任意尺寸的平滑算子的平滑处理方法，其中参数为平滑算子的长度的开根号（或者说是对应的二维数组的高或宽） |

其头函数Smoothing\_param.h为：

1. #pragma once
2. #include "Smoothing.h"
3. **class** Smoothing\_param :
4. **public** Smoothing
5. {
6. **public**:
7. Smoothing\_param();
8. ~Smoothing\_param();
9. **void** Smoothing\_param\_processing(**int** n);
10. };

其源文件Smoothing\_param.cpp为：

1. #include "Smoothing\_param.h"
3. Smoothing\_param::Smoothing\_param()
4. {
5. }
7. Smoothing\_param::~Smoothing\_param()
8. {
9. }
11. **void** Smoothing\_param::Smoothing\_param\_processing(**int** n) {
12. **int** \* oper;
13. oper = **new** **int**[n \* n]() ;
14. **int** x = (n - 1) / 2;
15. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
16. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
17. **int** temp = 0;//用来计算新的中心像素点值
18. **for** (**int** k = 0; k < n; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
19. **for** (**int** l = 0; l < n; l++) {
20. **if** ((i - x + k < 0) || (j - x + l < 0) || (i - x + k >= img.rows) || (j - x + l >= img.cols))//当算子位于边缘区域时，排除图像中不存在的点
21. **continue**;
22. temp += (oper[k \* n + l] + 1) \*img.at<Vec3b>(i - x + k, j - x + l)[0];
23. }
24. }
25. temp /= (n \* n);//之所以放后面是因为1 / 9 的影响
26. //将计算结果赋给dstimg中心像素点的三通道值
27. **for** (**int** m = 0; m < 3; m++) {
28. img\_temp.at<Vec3b>(i, j)[m] = temp;
29. }
30. }
31. }
32. **delete** oper;
33. }

对于上述三种有含参方法的子类，在main函数中对它们的使用方法分别如下：

1. //参数灰度
2. String filename\_Rgb2Gray\_param = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Lenna\_Rgb2Gray\_param.jpg";
3. Rgb2Gray\_param rgb2Gray\_param;
4. rgb2Gray\_param.img = imread(filename, IMREAD\_COLOR);
5. rgb2Gray\_param.Rgb2Gray\_param\_processing("max");//此处参数可选：avr max weight valid\_input等
6. imwrite(filename\_Rgb2Gray\_param, rgb2Gray\_param.img);
8. //参数锐化
9. String filename\_Sharpening\_param = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Sharpening\_param.jpg";
10. Sharpening\_param sharpening\_param;
11. sharpening\_param.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
12. sharpening\_param.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
13. **int** oper[3][3] = { { -1,-1,-1 },{ -1,9,-1 },{ -1,-1,-1 } };
14. sharpening\_param.Sharpening\_param\_processing(oper);
15. imwrite(filename\_Sharpening\_param, sharpening\_param.img\_temp);
17. //参数平滑
18. String filename\_Smoothing\_param = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Smoothing\_param.jpg";
19. Smoothing\_param smoothing\_param;
20. smoothing\_param.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
21. smoothing\_param.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
22. smoothing\_param.Smoothing\_param\_processing(3);
23. imwrite(filename\_Smoothing\_param, smoothing\_param.img\_temp);

得到结果与不含参的方法一致，证明含参方法程序正确。

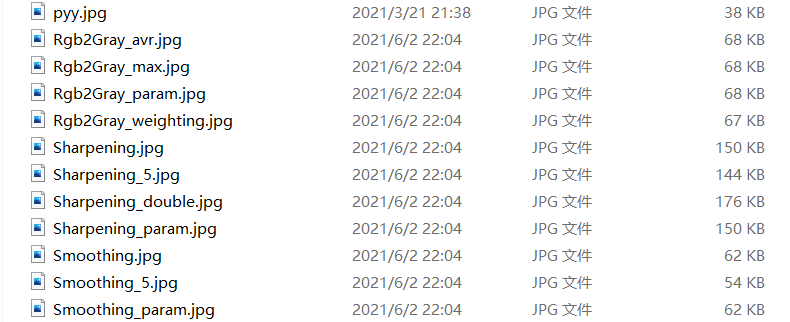
# 三、用户手册

*即使用说明（包括数据输入时的格式要求）。*

由于本程序原理是通过读取是通过路径读取制定路径“"D:\\大学作业\\3 下\\课程课程\\数据结构课程设计[\\pyy.jpg](file:///\\\\pyy.jpg)"”中图像pyy.jpg，得到其三种灰度图像，并选出其中最符合人眼分辨率的加权法得到的灰度图为进一步锐化、平滑处理的基本图像，所以唯一需要设置的就是一张图像的位置，我们可以：

1. 将图片放入主函数中对应目录"D:\\大学作业\\3 下\\课程课程\\数据结构课程设计[\\pyy.jpg](file:///\\\\pyy.jpg)"中，注意对图像文件名的修改，然后直接运行程序main.cpp即可，对于必做部分的程序而言，运行结果除原图之外将是：

* 一张均值灰度法处理得到的灰度图；
* 一张最大值灰度法处理得到的灰度图；
* 一张含参方法灰度法处理得到的灰度图；
* 一张加权灰度法处理得到的灰度图；
* 一张对加权灰度图按照题目要求3 \* 3锐化处理后得到的图；
* 一张对加权灰度图按照题目要求5 \* 5锐化处理后得到的图；
* 一张对加权灰度图按照题目要求加倍算子值3 \* 3锐化处理后得到的图；
* 一张对加权灰度图含参方法锐化处理后得到图；
* 一张对加权灰度图按照题目要求3 \* 3均值平滑处理后得到的图；
* 一张对加权灰度图5 \* 5均值平滑处理后得到的图；
* 一张对加权灰度图含参方法平滑处理后得到图，文件如下：

1. 修改程序main.cpp中主函数的文件路径为要进行图像处理的图片路径，然后直接运行程序main.cpp即可，运行结果同上。

# 四、调试及测试

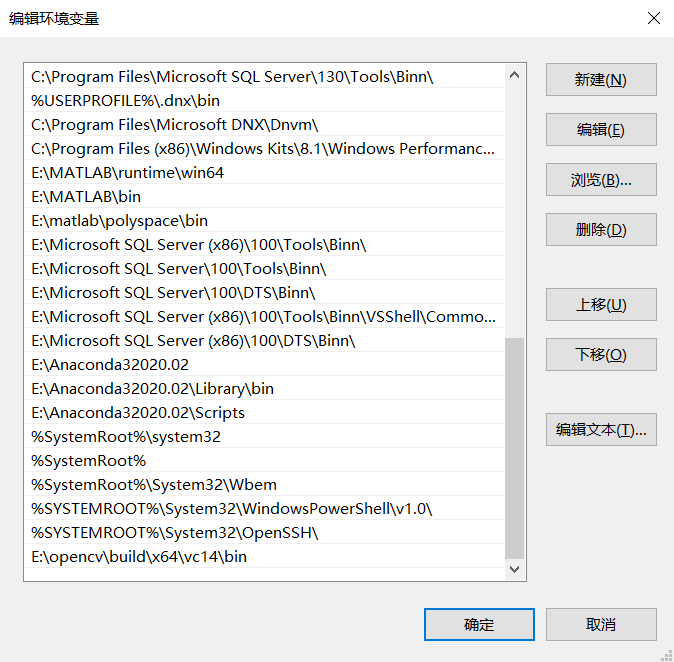
*调试过程中遇到的主要问题是如何解决的；对设计和编码的回顾讨论和分析；程序运行的时空效率分析；测试数据集；运行实例；改进设想；经验和体会等。*

## 4.1）主要问题

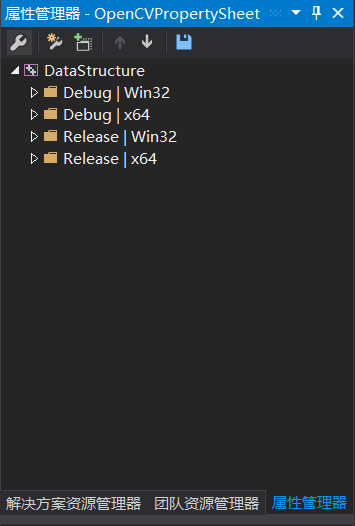
* 问题1——OpenCV的配置

对于本实验，首先要解决的第一个问题就是OpenCV的配置问题，由于本课设题目与图像处理关系密切，且课设要求上提及“考虑能否使用已有类库（包括直接使用或通过派生）以减少编程的工作量和提高程序的可靠性”，所以首先要进行的关键步骤就是进行OpenCV的下载和在VS 2015集成开发环境下的配置，步骤介绍如下：

首先在官网下载了最新的OpenCV4.5.1

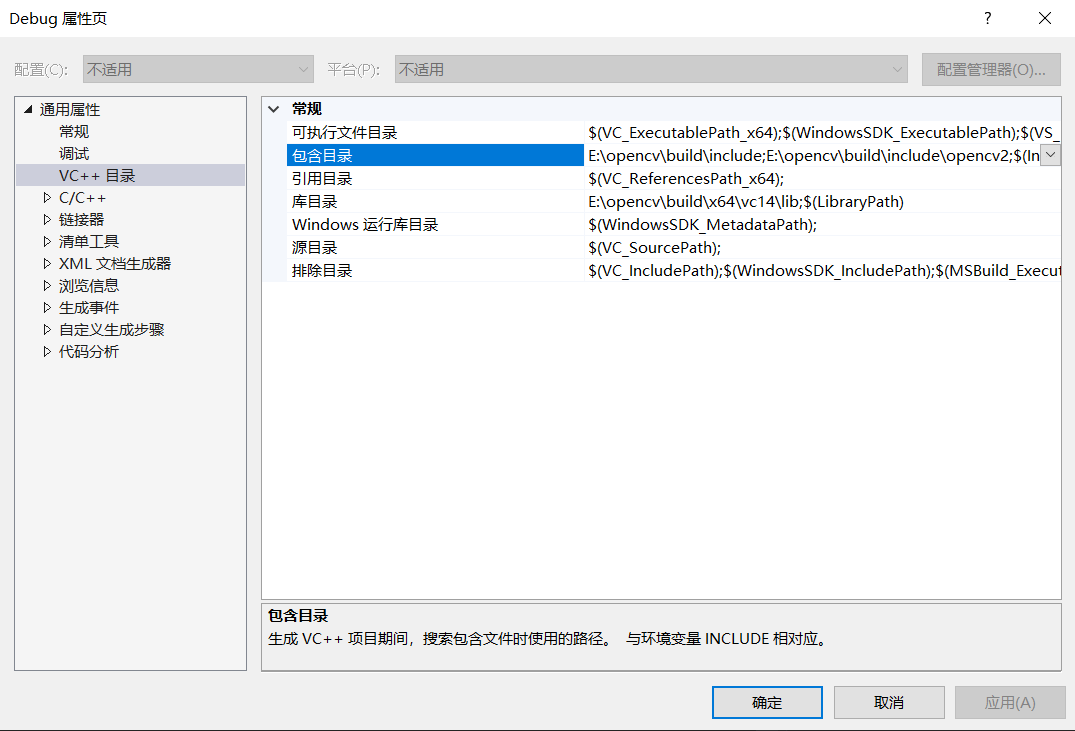
然后在系统的环境变量内新添加环境变量E:\opencv\build\x64\vc14\bin

然后打开VS 2015，在c++下新建一个空项目；

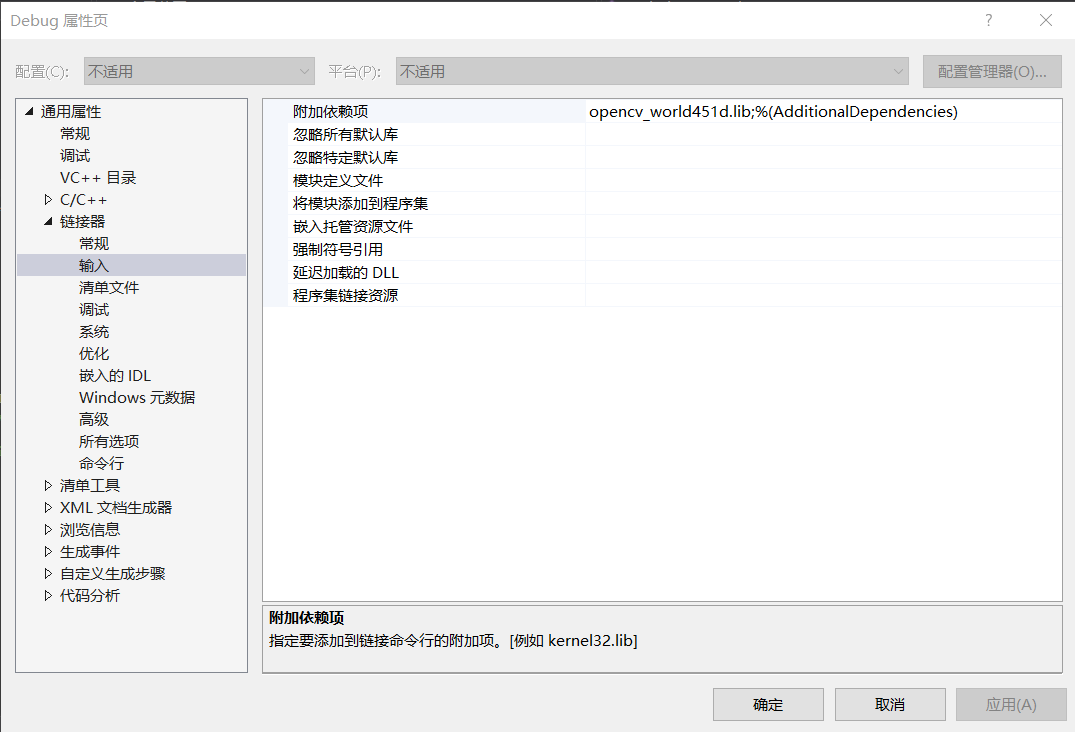
点击视图——其他窗口——属性管理器，右击Debug | x64选择属性;

在包含目录中添加：

（1）D:\OPENCV\opencv\build\include

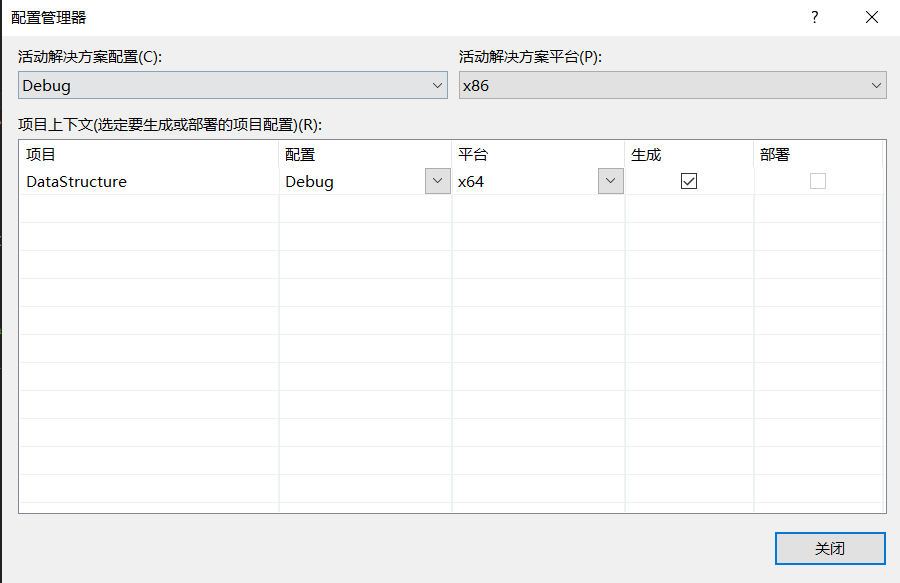
（2）D:\OPENCV\opencv\build\include\opencv2

在库目录中添加：D:\OPENCV\opencv\build\x64\vc14\lib（VS2015对应vc14）

点击链接器——输入，在附加依赖项中添加：opencv\_world450d.lib（注意d是debug，不带d是release）

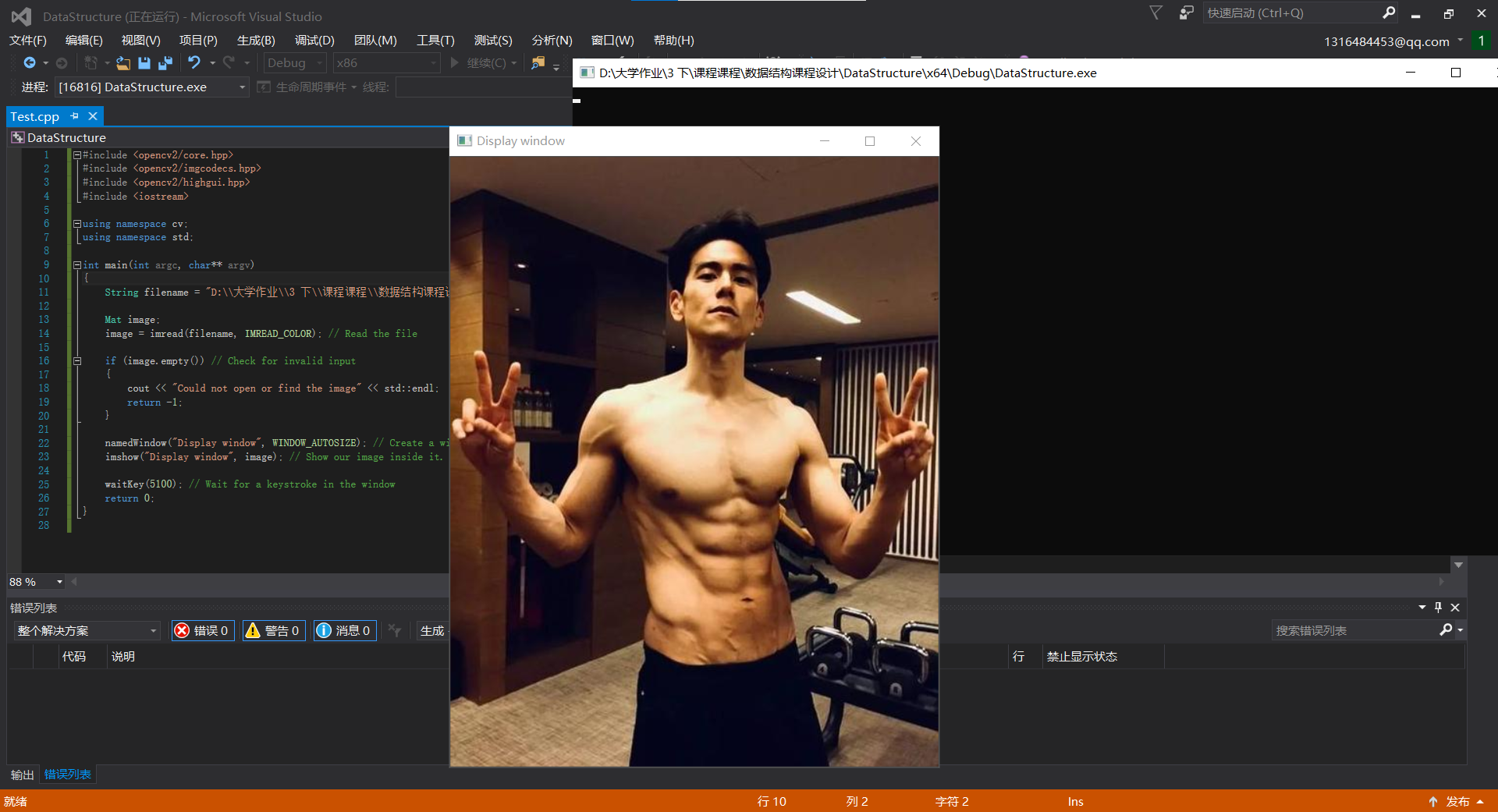
最后点击确定，全部配置完成。

完成上述配置过程之后运行还是报错“无法打开包括文件: “opencv2/core/core.hpp”: No such file or directory”

将项目管理器配置成Debug x64位模式如下。

运行测试程序如下：

1. #include <opencv2/core.hpp>
2. #include <opencv2/imgcodecs.hpp>
3. #include <opencv2/highgui.hpp>
4. #include <iostream>
6. **using** **namespace** cv;
7. **using** **namespace** std;
9. **int** main(**int** argc, **char**\*\* argv)
10. {
11. String filename = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\pyy.jpg";
13. Mat image;
14. image = imread(filename, IMREAD\_COLOR); // Read the file
16. **if** (image.empty()) // Check for invalid input
17. {
18. cout << "Could not open or find the image" << std::endl;
19. **return** -1;
20. }
21. namedWindow("Display window", WINDOW\_AUTOSIZE); // Create a window for display.
22. imshow("Display window", image); // Show our image inside it.
23. waitKey(5100); // Wait for a keystroke in the window
24. **return** 0;
25. }

结果如下，问题解决，至此，说明OpenCV配置完成。

* 问题2——锐化算子中对算子未覆盖部分的排除

通过如下代码即可解决。

1. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点

* 问题3——平滑算子不宜在算子中直接填入1/9

对于题目中设计的均值平滑，我们设计算子时需要注意不要设为一个3 \* 3且数组内的每个值都是1/9的二维数组，，因为如果将其内每个值设为1/9（小数），就需要考虑数据类型之间的转换，加大程序编写的复杂度，所以我们设计的算子是3 \* 3的二维数组，其中每个值都是1，这样的话，算子就是如下所示：

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

\* 1 / 9；也就是说，我们只需在最后除以算子的尺寸3 \* 3即可。

所以我们只需要

1. **int** oper[3][3];//平滑算子 n = 3
2. memset(oper, 0, **sizeof**(oper));//全部初始化为0

配合下面这段代码

1. temp += (oper[k][l] + 1) \*img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
3. temp /= (3 \* 3);//之所以放后面是因为1 / 9 的影响

即可解决这一问题。

* 问题4——含参平滑算子方法无法直接传递动态数组为参数

在Java中，这是完全可以的，比如我们运行如下程序：

1. import java.util.Arrays;
3. **public** **class** Test {
4. **public** **static** **void** main(String[] args) {
5. **int** num = 5;
6. **int**[] array = **new** **int**[num];
7. System.out.println(Arrays.toString(array));
8. }
9. }

输出结果为：[0, 0, 0, 0, 0]

但是在C++中则不然，在C++中是不支持变量作为数组长度参数的，如 int num=10；int array[num]；

这样写会提示编译错误"表达式必须含有常量值"。C++ Primer这本书中也确实这么写的，用VS编程编译也会报错，但是用codeblocks就不会，而且结果正确，还是不太明白原因。

但是我想实现类似的效果怎么办？

可以通过指针来动态申请空间实现动数组长度的变量赋值，写法如下：

1. **int** num= 10;
2. **int** \*array;
3. array = **new** **int**[num];

这样array就可以当做数组来用了，这个数组的长度可以在程序运行时由计算得来。如果是普通的数组如int is[10] 编译时必须能确定数组长度，不然会报编译错误，这样灵活性受限比较大。我想这个就是new的存在原因之一吧，在栈中分配的内存，大小都是编译时就确定好的，如果想在运行时来动态计算使用内存的大小的话，就要用new这样的动态分配函数，来达到更高的灵活性。

注意：C++ 用new分配空间以后，不用的时候要记得delete释放内存，不然会有内存泄露问题。

所以在我们的Smoothing\_param.cpp中相对应的代码就是：

1. **int** \* oper;
2. oper = **new** **int**[n \* n]() ;

和

1. **delete** oper;

注意：我们用的是

1. oper = **new** **int**[n \* n]() ;

而不是

1. oper = **new** **int**[n \* n] ;
2. memset(oper, 0, **sizeof**(oper));//全部初始化为0

因为后者无法将动态数组初始化全为0。

## 4.2）程序运行的时空效率分析

对于程序的时空效率复杂度，我们分析如下：

对于灰度化，我们以均值灰度化为例子，

1. **void** Rgb2Gray\_avr::Rgb2Gray\_avr\_processing()
2. {
3. **for** (**int** i = 0; i < img.rows; i++) {
4. **for** (**int** j = 0; j < img.cols; j++) {
5. **for** (**int** k = 0; k < 3; k++) {
6. img.at<Vec3b>(i, j)[k] = (img.at<Vec3b>(i, j)[0] + img.at<Vec3b>(i, j)[1] + img.at<Vec3b>(i, j)[2]) / 3;
7. }
8. }
9. }
10. }

我们假设图像长宽比为m：n，很容易看出，最内部的三通道分别赋值的代码部分为常数，所以均值灰度化的时间复杂度是 O（m \* n）；由于灰度化是读取一个像素点就处理一个像素点，无需新开辟空间，所以均值灰度化的空间复杂度是O（1）。

对于锐化操作，我们以题设的锐化算子对应的程序为例子，

1. **void** Sharpening::Sharpening\_processing(){
2. //由于n = 3 所以 x = 1
3. **int** oper[3][3] = { { -1,-1,-1 },{ -1,9,-1 },{ -1,-1,-1 } };
4. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
5. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
6. **int** temp = 0;//用来计算新的中心像素点值
7. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
8. **for** (**int** l = 0; l < 3; l++) {
9. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
10. **continue**;
11. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
12. }
13. }
14. **if** (temp > 255) {
15. temp = 255;
16. }
17. //当计算结果小于0，就把它置为0
18. **if** (temp < 0) {
19. temp = 0;
20. }
21. **for** (**int** m = 0; m < 3; m++) {
22. img\_temp.at<Vec3b>(i, j)[m] = temp;
23. }
24. }
25. }
26. }

我们假设图像长宽比为m：n，很容易看出，最内部的算子排除边缘的代码部分和三通道分别赋值的代码部分为常数，所以均值灰度化的时间复杂度是 O（m \* n）；由于锐化是读取像素点后，不能直接对当前像素点进行修改，否则前面的操作影响后面的结果，也即需要新开辟空间，大小就等于原图像尺寸，所以题设的锐化的空间复杂度是O（m \* n）。

对于平滑操作，我们以题设的锐化算子对应的程序为例子，

1. **void** Smoothing::Smoothing\_processing(){
2. **int** oper[3][3];//平滑算子 n = 3
3. memset(oper, 0, **sizeof**(oper));//全部初始化为0
4. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
5. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
6. **int** temp = 0;//用来计算新的中心像素点值
7. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
8. **for** (**int** l = 0; l < 3; l++){
9. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时，排除图像中不存在的点
10. **continue**;
11. temp += (oper[k][l] + 1) \*img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
12. }
13. }
14. temp /= (3 \* 3);//之所以放后面是因为1 / 9 的影响
15. //将计算结果赋给dstimg中心像素点的三通道值
16. **for** (**int** m = 0; m < 3; m++) {
17. img\_temp.at<Vec3b>(i, j)[m] = temp;
18. }
19. }
20. }
21. }

我们假设图像长宽比为m：n，很容易看出，最内部的算子排除边缘的代码部分和三通道分别赋值的代码部分为常数，所以均值灰度化的时间复杂度是 O（m \* n）；由于锐化是读取像素点后，不能直接对当前像素点进行修改，否则前面的操作影响后面的结果，也即需要新开辟空间，大小就等于原图像尺寸，所以题设的锐化的空间复杂度是O（m \* n）。

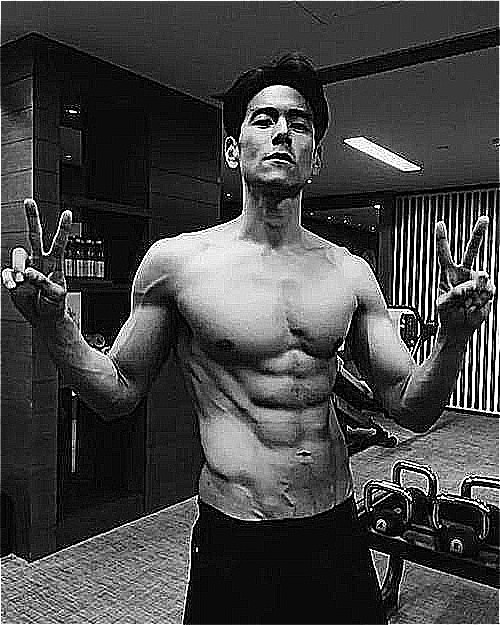
## 4.3）测试数据集

测试集选择的一张是彭于晏的照片pyy.jpg。

另外一张是图像处理女神“Lenna”的照片Lenna.jpg。4

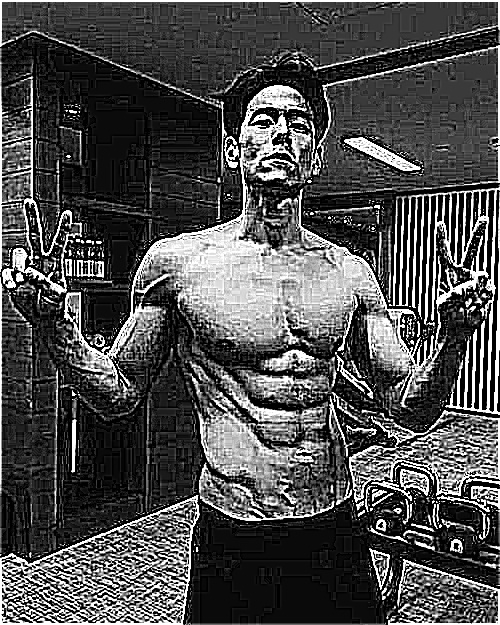
## 4.4）运行实例

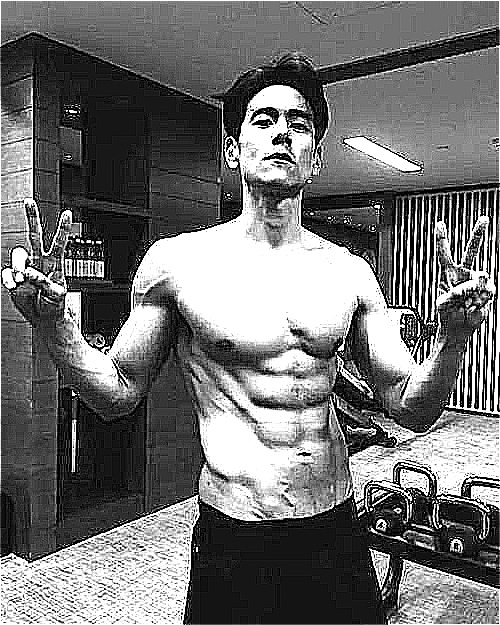
对测试数据集分别进行测试，彭于晏得到结果如下：

* 均值灰度化：
* 最大值灰度
* 加权法灰度
* 题设锐化（中心为9）
* 题设平滑（算子尺寸为3 \* 3）

## 4.5）改进设想

对于锐化，我进行了两种尝试，一是尺度上的尝试，我将锐化算子改换成5 \* 5的，算子中心周围的24个点的值均为 -1，对于中心值的选取，我做了不同的尝试，发现在21-25之间，锐化后图像存在有效信息，其中22效果最好，对边缘起到了检测效果，具体如下列图对比所示。

* 5 \* 5算子中心值为21
* 5 \* 5算子中心值为22
* 5 \* 5算子中心值为23
* 5 \* 5算子中心值为24
* 5 \* 5算子中心值为25

二是算子值上的尝试，我将锐化算子内容值翻倍，具体结果如下图所示。

对于平滑，我进行了尺度上的尝试，我将锐化算子改换成5 \* 5的，对于均值平滑而言，也就是说其平滑算子为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1/25 | 1/25 | 1/25 | 1/25 | 1/25 |
| 1/25 | 1/25 | 1/25 | 1/25 | 1/25 |
| 1/25 | 1/25 | 1/25 | 1/25 | 1/25 |
| 1/25 | 1/25 | 1/25 | 1/25 | 1/25 |
| 1/25 | 1/25 | 1/25 | 1/25 | 1/25 |

结果如下图所示，可以看到说好听是去除了更多的噪声，说难听点就是更模糊了。

## 4.6）对设计和编码的回顾讨论和分析、经验和体会

关于锐化和平滑，在上面都已介绍，下面谈谈对灰度化的不同处理方法，我们知道，灰度图像是指在RGB模型中，当R=G=B时，彩色表示一种灰度颜色，其中R（或G、B）的值叫做灰度值[1]，灰度值的取值范围为0~255，其中灰度值为0时表示黑色，为255时表示白色，中间的值代表不同程度的灰色。

根据网上查阅资料的情况，灰度还有最大值法和加权平均值法：顾名思义，最大值法就是将彩色图像中的三分量亮度的最大值作为灰度图的灰度值，也即f(i,j) =max（R(i,j)+G(i,j)+B(i,j)）；

而加权平均值法是根据重要性及其他指标，将三个分量以不同的权值进行加权平均，由于人眼对绿色的敏感度最高，对蓝色的敏感度最低，因此，按下式对RGB三分量进行加权平均能得到较合理的灰度图像。f(i,j) =0.30R(i,j)+0.59G(i,j)+0.11B(i,j)；

可以看到，算数平均值得到的灰度图颜色最深，最大值处理得到的过于亮（值靠近255）了，而且面部的阴影失真，而根据人眼的对颜色敏感度设计的加权平均值得到的结果最逼近原图的感觉，所以我们后面平滑和锐化处理的优化部分基础的灰度图使用加权灰度图；

做完上面所有内容之后，我脑海里唯一的想法就是图像处理真是太有趣了，看着处理的结果随着算子的变化而变化，这个“**猜想——实验——验证——猜想**”的循环过程真的很有趣，未来如果能有机会接触到科研，我会选择在CV方向做一点事情。

# 五、选做部分

*查阅其他的平滑和锐化算子并实现；*

## 5.1）平滑算子

在这里，为了看出平滑算子的去噪效果，我们选择含噪音的图片来处理，如下：

* 中值平滑算子

中值滤波法是一种非线性平滑技术，它将每一像素点的像素值设置为该点某邻域窗口内的所有像素点像素值的中值。统计排序滤波器，中值对椒盐噪声（有最大小值特性）有很好的抑制作用，效果是图像中瑕疵更光滑。（没用卷积）

操作方法：窗口内的像素从小到大排序取中值赋给中间像素。在此，我们选择用3 \* 3的中值算子处理，核心方法代码如下：

1. **void** Smoothing\_mid::Smoothing\_mid\_processing(){
2. **int** oper[9];//用来存放模板对应的9个像素点的值
3. memset(oper, 0, **sizeof**(oper));//初始化为0
4. **for** (**int** i = 0; i < img.rows; i++) {
5. **for** (**int** j = 0; j < img.cols; j++) {
6. **int** count = 0;//标记共有多少个合法点
7. **for** (**int** k = 0; k < 3; k++) {
8. **for** (**int** l = 0; l < 3; l++) {
9. //当算子位于边缘区域时，排除图像中不存在的点
10. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))
11. **continue**;
12. oper[count] = img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
13. count++;
14. }
15. }
16. //将temp数组中的像素值从小到大排序
17. sort(oper, oper + count);
18. //将temp数组的中值赋给中心像素点的三通道
19. **for** (**int** m = 0; m < 3; m++) {
20. img\_temp.at<Vec3b>(i, j)[m] = oper[(count - 1) / 2];//求得中位数
21. }
22. }
23. }
24. }

处理结果与原灰度图对比如下（左图为加权灰度图，右图为中值平滑处理结果）：

**可以看到去噪效果很不错。（中值平滑对椒盐噪声（有最大小值特性）有很好的抑制作用）**

* 高斯平滑算子

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。高斯平滑算子有以下两种，一是3 \* 3的：

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

\* 1 / 16

二是5 \* 5的：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |

\* 1 / 273

我们都实现一下，3 \* 3高斯平滑核心方法代码如下：

1. **void** Smoothing\_Gauss\_3::Smoothing\_Gauss\_3\_processing(){
2. //由于n = 3 所以 x = 1
3. **int** oper[3][3] = { { 1,2,1 },{ 2,4,2 },{ 1,2,1 } };
4. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
5. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
6. **int** temp = 0;//用来计算新的中心像素点值
7. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
8. **for** (**int** l = 0; l < 3; l++) {
9. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
10. **continue**;
11. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
12. }
13. }
14. temp = temp / 16;
15. **if** (temp > 255) {
16. temp = 255;
17. }
18. //当计算结果小于0，就把它置为0
19. **if** (temp < 0) {
20. temp = 0;
21. }
22. **for** (**int** m = 0; m < 3; m++) {
23. img\_temp.at<Vec3b>(i, j)[m] = temp;
24. }
25. }
26. }
27. }

5 \* 5高斯平滑核心方法代码如下：

1. **void** Smoothing\_Gauss\_5::Smoothing\_Gauss\_5\_processing(){
2. //由于n = 3 所以 x = 1
3. **int** oper[5][5] = { { 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 } };
4. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
5. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
6. **int** temp = 0;//用来计算新的中心像素点值
7. **for** (**int** k = 0; k < 5; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
8. **for** (**int** l = 0; l < 5; l++) {
9. **if** ((i - 2 + k < 0) || (j - 2 + l < 0) || (i - 2 + k >= img.rows) || (j - 2 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
10. **continue**;
11. temp += (oper[k][l]) \* img.at<Vec3b>(i - 2 + k, j - 2 + l)[0];
12. }
13. }
14. temp /= 273;
15. **if** (temp > 255) {
16. temp = 255;
17. }
18. //当计算结果小于0，就把它置为0
19. **if** (temp < 0) {
20. temp = 0;
21. }
22. **for** (**int** m = 0; m < 3; m++) {
23. img\_temp.at<Vec3b>(i, j)[m] = temp;
24. }
25. }
26. }
27. }

由于高斯平滑，对于椒盐噪声，高斯平滑的效果不理想，如下是对椒盐噪声的图分别3 \* 3高斯平滑和5 \* 5高斯平滑的结果，可以看出对椒盐噪音的处理，高斯平滑不如中值平滑：

换一张再试试看：

处理结果如下（左边为3 \* 3的高斯平滑结果，右边为5 \* 5的高斯平滑结果）：

## 5.2）锐化算子

考虑到锐化算子对边缘的检测、增强作用，我们对于锐化算子选择的测试数据如下所示：

* Prewitt算子

Prewitt算子是一种一阶[微分算子](https://baike.baidu.com/item/%E5%BE%AE%E5%88%86%E7%AE%97%E5%AD%90/9706908" \t "_blank)的[边缘检测](https://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E6%A3%80%E6%B5%8B/3221648)，利用像素点上下、左右邻点的[灰度](https://baike.baidu.com/item/%E7%81%B0%E5%BA%A6/4615393" \t "_blank)差，在边缘处达到极值检测边缘，去掉部分伪边缘，对噪声具有平滑作用 。其原理是在图像空间利用两个方向模板与图像进行[邻域](https://baike.baidu.com/item/%E9%82%BB%E5%9F%9F/6323269" \t "_blank)[卷积](https://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF/9411006)来完成的，这两个方向模板一个检测水平边缘，一个检测垂直边缘。

检测水平边缘如下：

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

其水平方向用于计算均值，平滑噪声；

竖直方向模板如下：

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

其水平方向用于计算均值，平滑噪声。

对[数字图像](https://baike.baidu.com/item/%E6%95%B0%E5%AD%97%E5%9B%BE%E5%83%8F/5199238)f(x，y)，Prewitt算子的定义如下：

G(i)=|[f(i-1,j-1)+f(i-1,j)+f(i-1，j+1)]-[f(i+1,j-1)+f(i+1，j)+f(i+1，j+1)]|

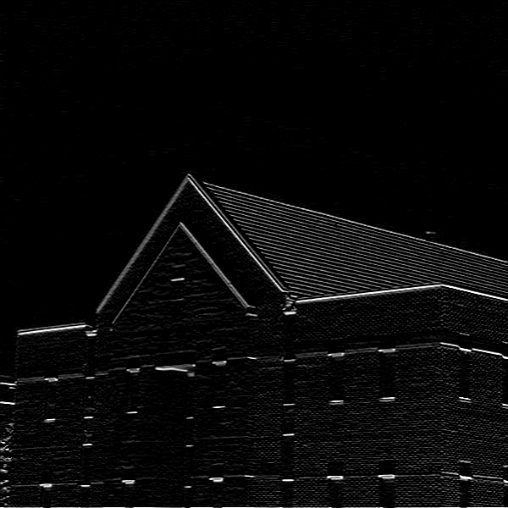
G(j)=|[f(i-1,j+1)+f(i,j+1)+f(i+1，j+1)]-[f(i-1,j-1)+f(i,j-1)+f(i+1，j-1)]|

则 P(i,j) = max[G(i),G(j)]或 P(i,j) = G(i)+G(j)

经典Prewitt算子认为：凡[灰度](https://baike.baidu.com/item/%E7%81%B0%E5%BA%A6/4615393" \t "_blank)新值大于或等于[阈值](https://baike.baidu.com/item/%E9%98%88%E5%80%BC)的像素点都是边缘点。即选择适当的阈值T，若P(i,j)≥T，则(i,j)为边缘点，P(i,j)为[边缘图像](https://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E5%9B%BE%E5%83%8F/22031182" \t "_blank)。这种判定是欠合理的，会造成边缘点的误判，因为许多噪声点的[灰度值](https://baike.baidu.com/item/%E7%81%B0%E5%BA%A6%E5%80%BC/10259111)也很大，而且对于幅值较小的边缘点，其边缘反而丢失了。

我们选择的方法是P(i,j) = max[G(i),G(j)]，并且没有选择阈值，只取水平和竖直方向上的最大值作为最后的输出值，核心方法源代码如下：

1. **void** Prewitt::Prewitt\_processing(){
2. **int** oper\_y[3][3] = { { -1,0,1 },{ -1,0,1 },{ -1,0,1 } };
3. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
4. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
5. **int** temp = 0;//用来计算新的中心像素点值
6. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
7. **for** (**int** l = 0; l < 3; l++) {
8. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
9. **continue**;
10. temp += (oper\_y[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
11. }
12. }
13. **if** (temp > 255) {
14. temp = 255;
15. }
16. //当计算结果小于0，就把它置为0
17. **if** (temp < 0) {
18. temp = 0;
19. }
20. **for** (**int** m = 0; m < 3; m++) {
21. img\_y.at<Vec3b>(i, j)[m] = temp;
22. }
23. }
24. }
26. **int** oper\_x[3][3] = { { 1,1,1 },{0,0,0 },{ -1,-1,-1 } };
27. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
28. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
29. **int** temp = 0;//用来计算新的中心像素点值
30. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
31. **for** (**int** l = 0; l < 3; l++) {
32. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
33. **continue**;
34. temp += (oper\_x[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
35. }
36. }
37. **if** (temp > 255) {
38. temp = 255;
39. }
40. //当计算结果小于0，就把它置为0
41. **if** (temp < 0) {
42. temp = 0;
43. }
44. **for** (**int** m = 0; m < 3; m++) {
45. img\_x.at<Vec3b>(i, j)[m] = temp;
46. }
47. }
48. }
50. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
51. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
52. **for** (**int** m = 0; m < 3; m++) {
53. img\_temp.at<Vec3b>(i, j)[m] = max(img\_x.at<Vec3b>(i, j)[0], img\_y.at<Vec3b>(i, j)[0]);
54. }
55. }
56. }
57. }

水平方向上结果如下:

竖直方向上的结果如下：

最终处理结果如下：

* Sobel算子

索贝尔算子是计算机视觉领域的一种重要处理方法。主要用于获得数字图像的一阶梯度，常见的应用和物理意义是[边缘检测](https://baike.baidu.com/item/%E8%BE%B9%E7%BC%98%E6%A3%80%E6%B5%8B/3221648)。索贝尔算子是把图像中每个像素的上下左右四领域的灰度值加权差，在边缘处达到极值从而检测边缘。

索贝尔算子主要用作边缘检测。在技术上，它是一离散性差分算子，用来运算图像亮度函数的梯度之近似值。在图像的任何一点使用此算子，将会产生对应的梯度矢量或是其法矢量。

索贝尔算子不但产生较好的检测效果，而且对噪声具有平滑抑制作用，但是得到的边缘较粗，且可能出现伪边缘。

**与[Prewitt算子](https://baike.baidu.com/item/Prewitt%E7%AE%97%E5%AD%90" \t "_blank)相比，**[**Sobel算子**](https://baike.baidu.com/item/Sobel%E7%AE%97%E5%AD%90/11000092)**对于像素的位置的影响做了加权，可以降低边缘模糊程度，因此效果更好。**由于Sobel算子是滤波算子的形式，用于提取边缘，可以利用快速卷积函数， 简单有效，因此应用广泛。美中不足的是，Sobel算子并没有将图像的主体与背景严格地区分开来，换言之就是Sobel算子没有基于图像灰度进行处理，由于Sobel算子没有严格地模拟人的视觉生理特征，所以提取的图像轮廓有时并不能令人满意。 在观测一幅图像的时候，我们往往首先注意的是图像与背景不同的部分，正是这个部分将主体突出显示，基于该理论，我们给出了下面阈值化轮廓提取算法，该算法已在数学上证明当像素点满足正态分布时所求解是最优的。

Sobel算子垂直方向和水平方向的模板如下图所示，**实际应用中，每个像素点取两个模板卷积的最大值作为该像素点的输出值**，运算结果是一副边缘图像。前者可以检测出图像中的水平方向的边缘；

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

后者则可以检测图像中垂直方向的边缘。

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

所以其实代码和prewitt一致，只需将算子内容修改即可，故不表。

水平方向上结果如下：

竖直方向上结果如下：

最终合并结果为：

* Roberts算子

Roberts算子，又称罗伯茨算子，是一种最简单的算子，是一种利用局部差分算子寻找边缘的算子。他采用对角线方向相邻两象素之差近似梯度幅值检测边缘。检测垂直边缘的效果好于斜向边缘，定位精度高，对噪声敏感,无法抑制噪声的影响。

1963年，Roberts提出了这种寻找边缘的算子。Roberts边缘算子是一个2x2的模板，采用的是对角方向相邻的两个像素之差。从图像处理的实际效果来看，边缘定位较准，对噪声敏感。

Roberts算子的模板分为水平方向和垂直方向，如下式所示，从其模板可以看出，Roberts算子能较好的增强正负45度的图像边缘。

水平方向的算子：

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

竖直方向的算子：

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

处理的像素点中心点位于算子的左上角。

故核心方法的源代码为：

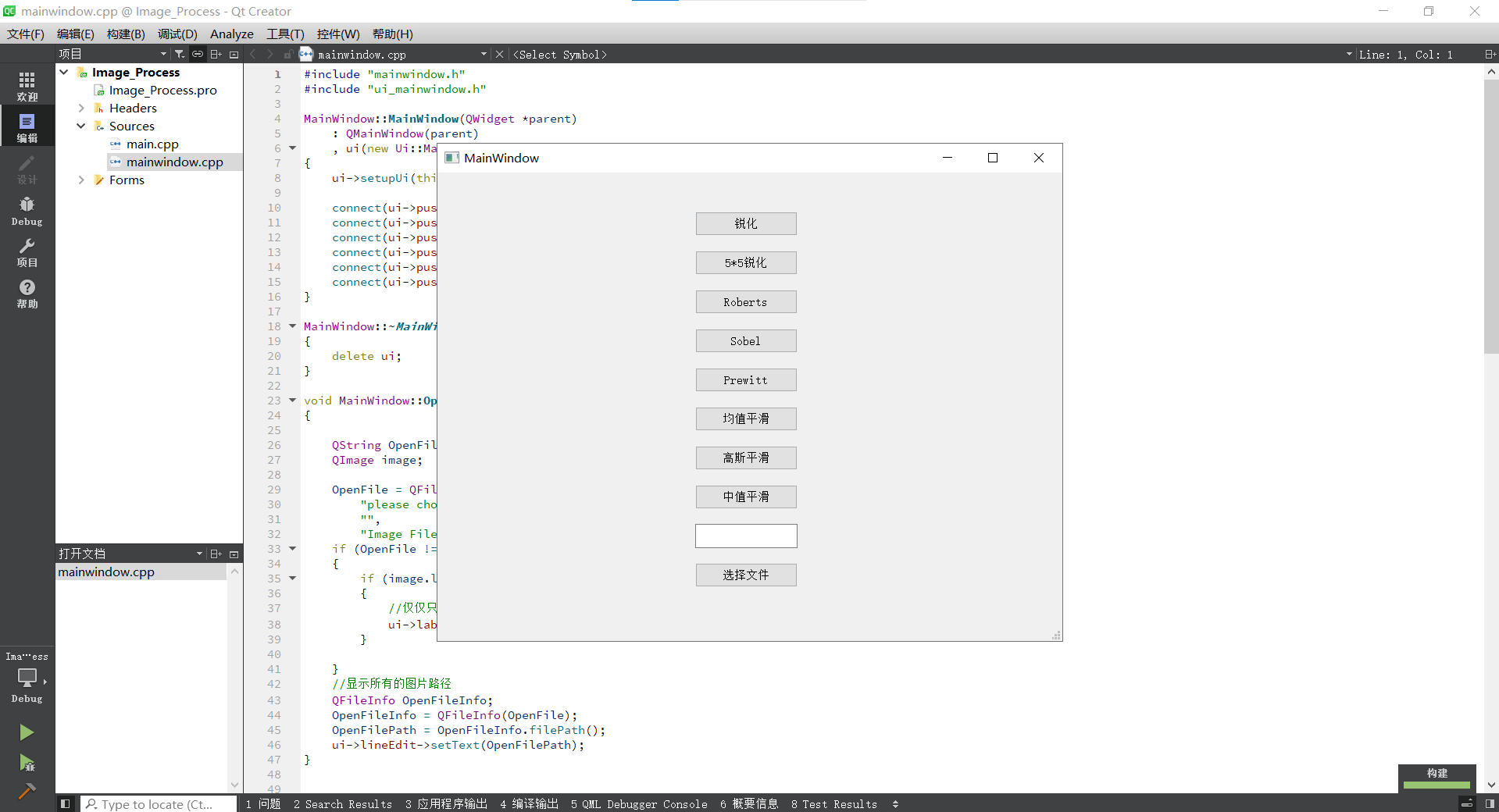
1. **void** Roberts::Roberts\_processing(){
2. **int** oper\_y[2][2] = { { 0,-1 },{ 1,0 } };
3. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
4. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
5. **int** temp = 0;//用来计算新的中心像素点值
6. **for** (**int** k = 0; k < 2; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
7. **for** (**int** l = 0; l < 2; l++) {
8. **if** ((i + k < 0) || (j + l < 0) || (i + k >= img.rows) || (j + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
9. **continue**;
10. temp += (oper\_y[k][l]) \* img.at<Vec3b>(i + k, j + l)[0];
11. }
12. }
13. **if** (temp > 255) {
14. temp = 255;
15. }
16. //当计算结果小于0，就把它置为0
17. **if** (temp < 0) {
18. temp = 0;
19. }
20. **for** (**int** m = 0; m < 3; m++) {
21. img\_y.at<Vec3b>(i, j)[m] = temp;
22. }
23. }
24. }
26. **int** oper\_x[2][2] = { { -1,0 },{ 0,1 } };
27. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
28. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
29. **int** temp = 0;//用来计算新的中心像素点值
30. **for** (**int** k = 0; k < 2; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
31. **for** (**int** l = 0; l < 2; l++) {
32. **if** ((i + k < 0) || (j + l < 0) || (i + k >= img.rows) || (j + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
33. **continue**;
34. temp += (oper\_x[k][l]) \* img.at<Vec3b>(i + k, j + l)[0];
35. }
36. }
37. **if** (temp > 255) {
38. temp = 255;
39. }
40. //当计算结果小于0，就把它置为0
41. **if** (temp < 0) {
42. temp = 0;
43. }
44. **for** (**int** m = 0; m < 3; m++) {
45. img\_x.at<Vec3b>(i, j)[m] = temp;
46. }
47. }
48. }
50. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
51. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
52. **for** (**int** m = 0; m < 3; m++) {
53. img\_temp.at<Vec3b>(i, j)[m] = max(img\_x.at<Vec3b>(i, j)[0], img\_y.at<Vec3b>(i, j)[0]);
54. }
55. }
56. }
57. }

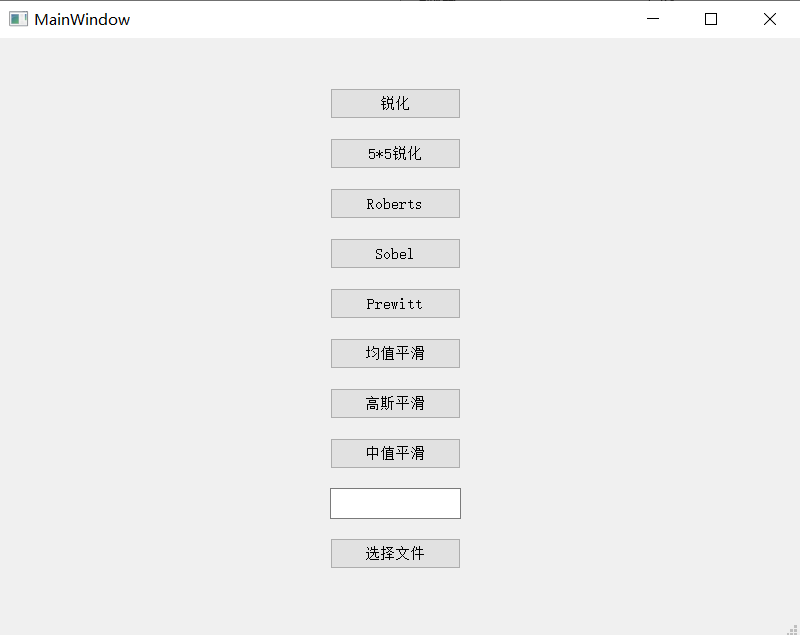
2\*2大小的模板在概念上很简单，但是他们对于用关于中心点对称的模板来计算边缘方向不是很有用。处理结果如下，水平方向上结果如下：

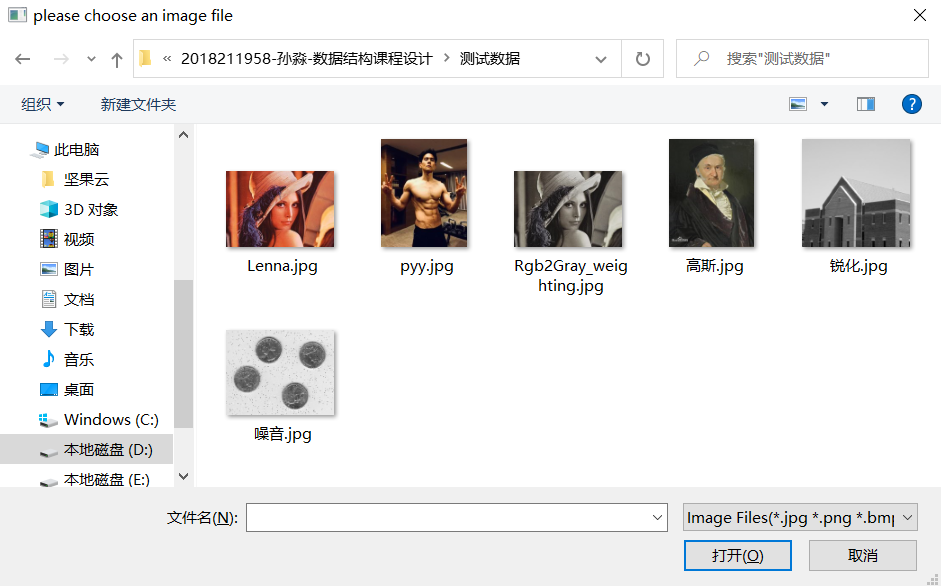
竖直方向上结果如下：

最终结果如下：

## 5.3）图形界面

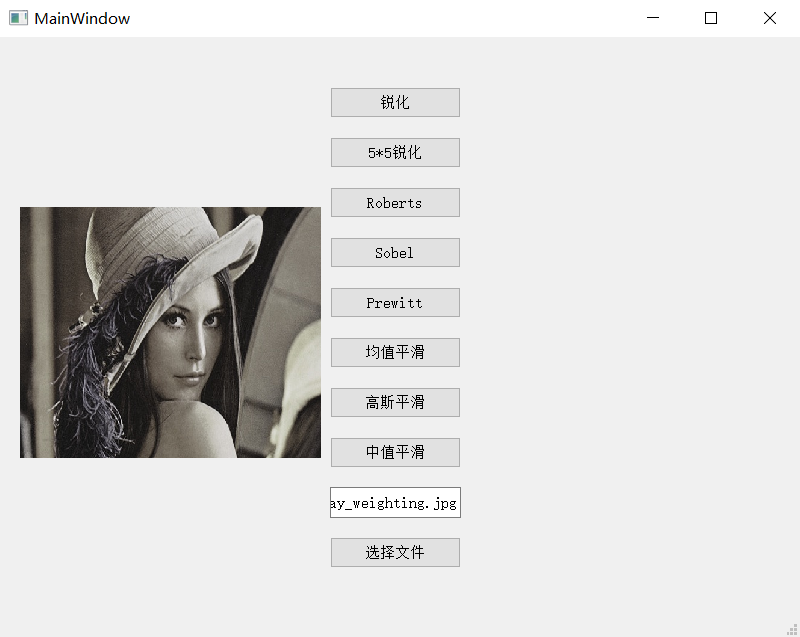
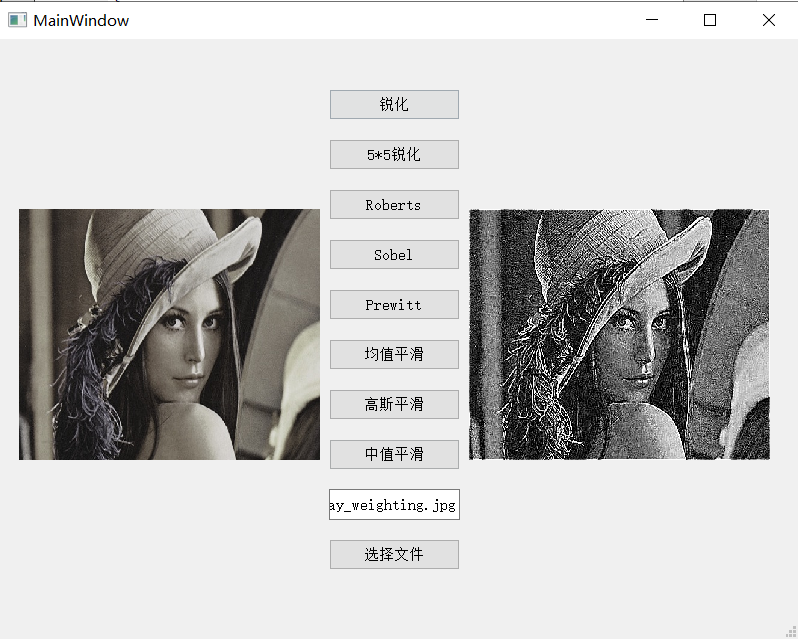
对于图形，我选择通过Qt来实现，运行结果如下。

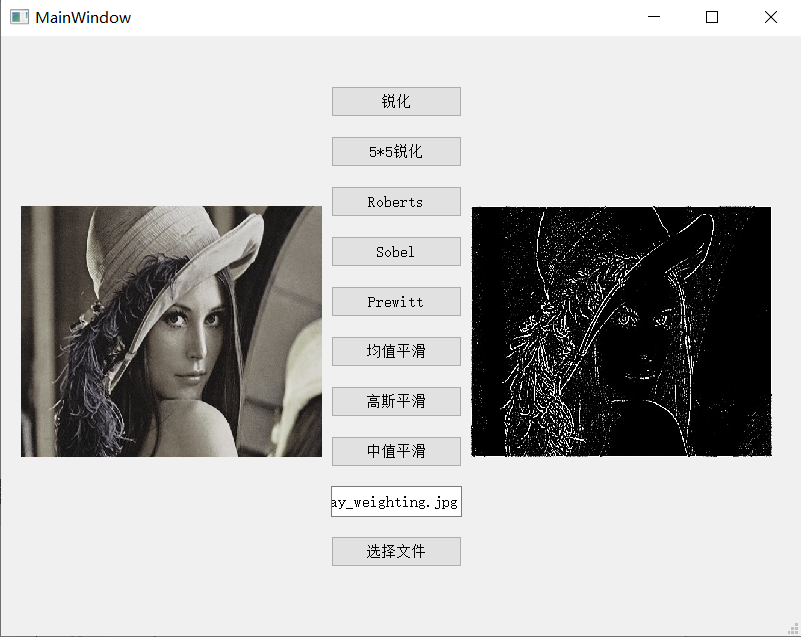
其中界面如下：

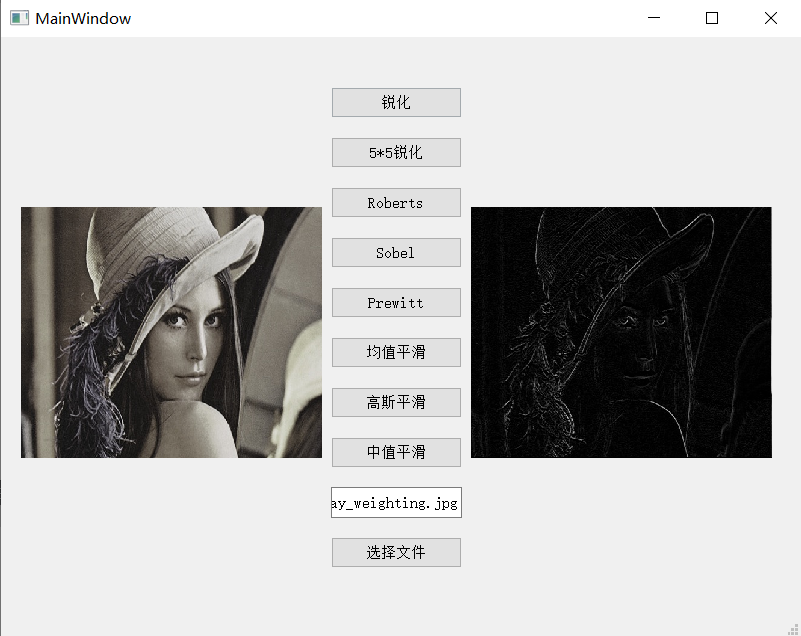
然后点击选择文件可以在文件夹中选择要处理的文件：

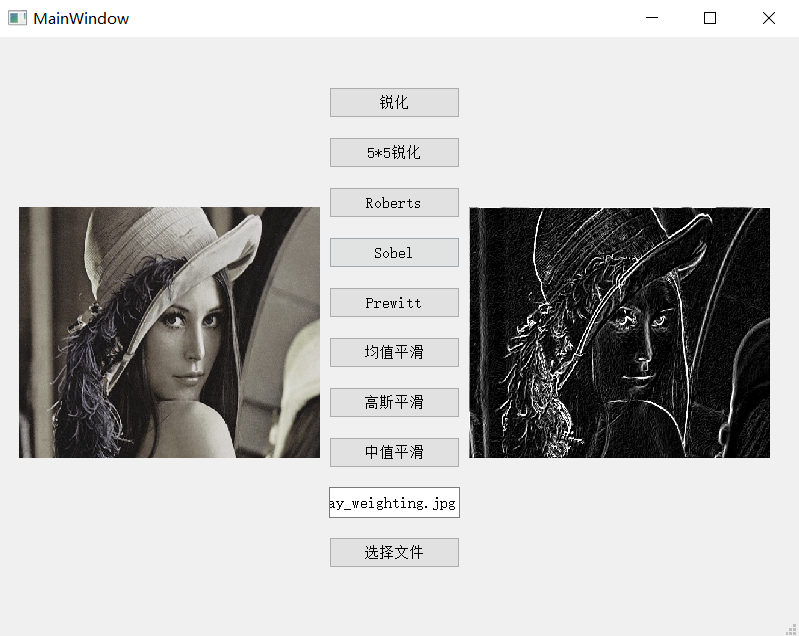
我们选择Rgb2Gray\_weighting.jpg。

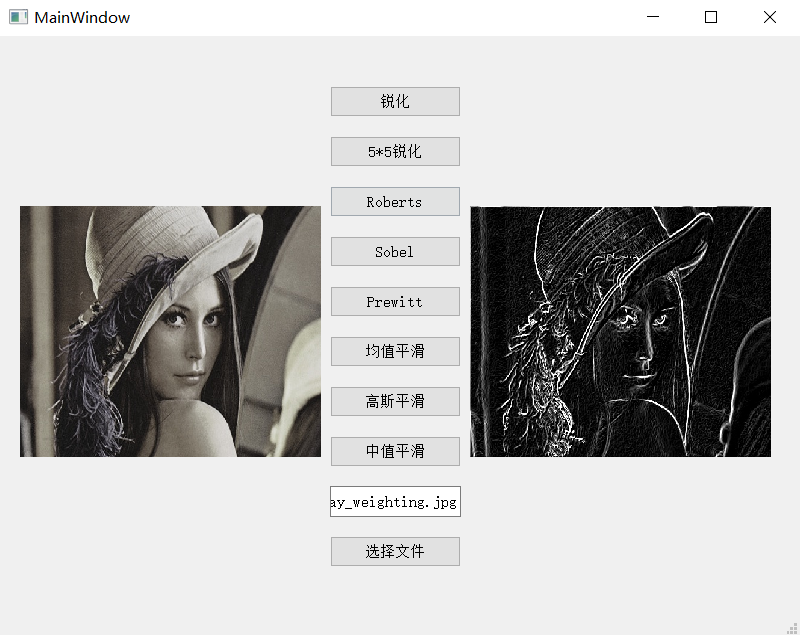
则界面如下。

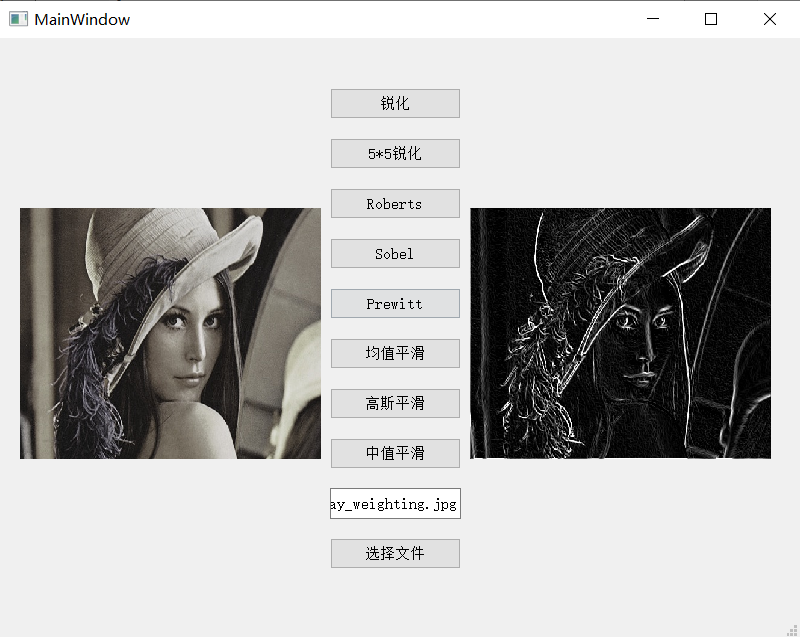
此时，点击对应的按钮，就可以得到按钮上对应操作的图像处理结果，如锐化。

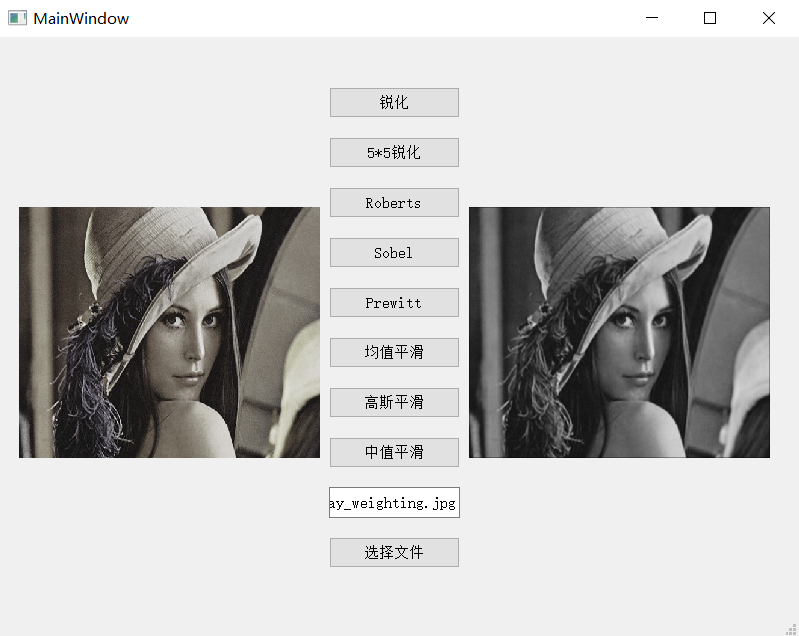
点击5\*5锐化。

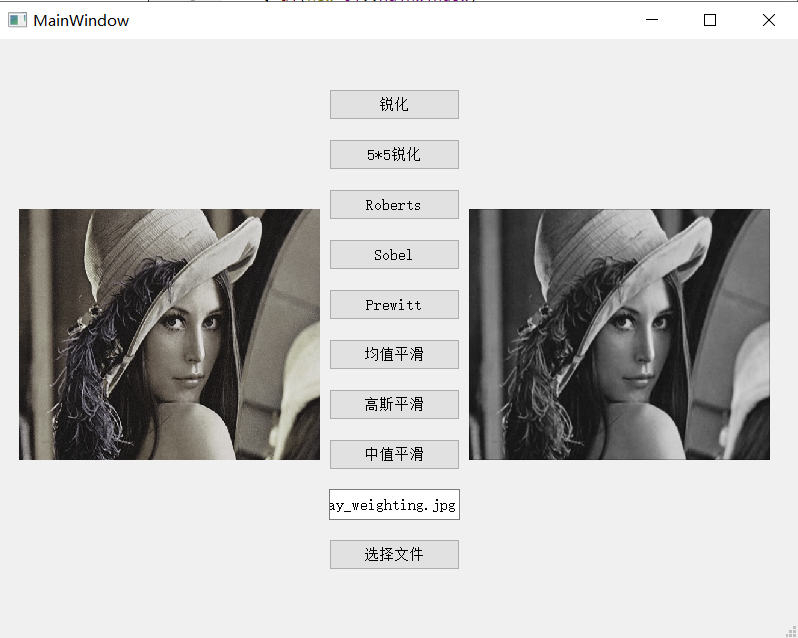
点击Roberts。

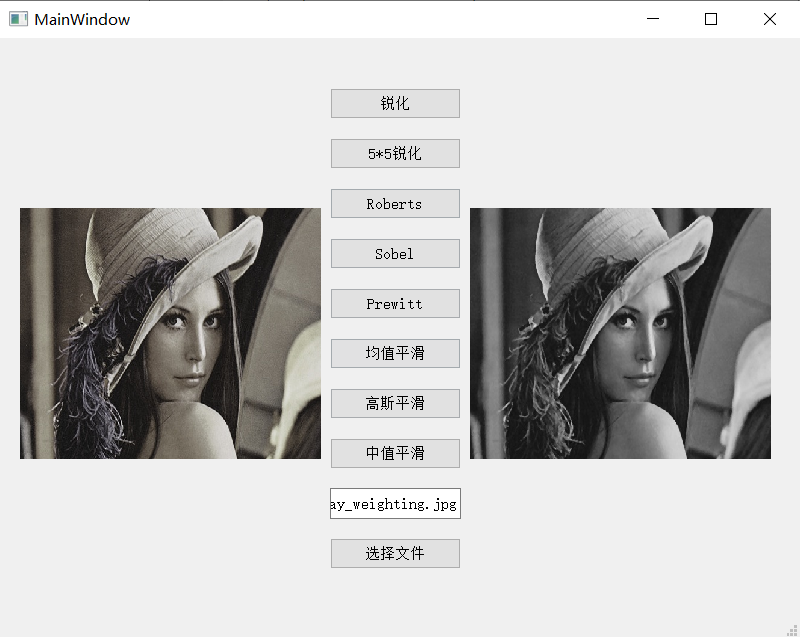
点击Roberts。

点击Sobel。

点击Prewitt。

点击均值平滑。

点击高斯平滑。

点击中值平滑。

# 六、附录

## 6.1）源程序清单（文件排序按首字母排序）

*打印文本和磁盘文件，磁盘文件是必须的。源程序要加注释，除原有注释外再用钢笔加一些必要的注释和断言。*

**Prewitt.h**

2. #pragma once
3. #include <iostream>
4. #include <highgui.hpp>
5. #include <core.hpp>
6. #include <cstring>
8. **using** **namespace** std;
9. **using** **namespace** cv;
10. **class** Prewitt
11. {
12. **public**:
13. Mat img;
14. Mat img\_temp;
15. Mat img\_x;
16. Mat img\_y;
17. Prewitt();
18. ~Prewitt();
19. **void** Prewitt\_processing();
20. **int** max(**int** a, **int** b) {
21. **if** (a >= b)
22. **return** a;
23. **return** b;
24. }
25. };

**Prewitt.cpp**

2. #include "Prewitt.h"


6. Prewitt::Prewitt()
7. {
8. }

11. Prewitt::~Prewitt()
12. {
13. }
15. **void** Prewitt::Prewitt\_processing(){
16. **int** oper\_y[3][3] = { { -1,0,1 },{ -1,0,1 },{ -1,0,1 } };
17. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
18. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
19. **int** temp = 0;//用来计算新的中心像素点值
20. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
21. **for** (**int** l = 0; l < 3; l++) {
22. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
23. **continue**;
24. temp += (oper\_y[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
25. }
26. }
27. **if** (temp > 255) {
28. temp = 255;
29. }
30. //当计算结果小于0，就把它置为0
31. **if** (temp < 0) {
32. temp = 0;
33. }
34. **for** (**int** m = 0; m < 3; m++) {
35. img\_y.at<Vec3b>(i, j)[m] = temp;
36. }
37. }
38. }
40. **int** oper\_x[3][3] = { { 1,1,1 },{0,0,0 },{ -1,-1,-1 } };
41. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
42. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
43. **int** temp = 0;//用来计算新的中心像素点值
44. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
45. **for** (**int** l = 0; l < 3; l++) {
46. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
47. **continue**;
48. temp += (oper\_x[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
49. }
50. }
51. **if** (temp > 255) {
52. temp = 255;
53. }
54. //当计算结果小于0，就把它置为0
55. **if** (temp < 0) {
56. temp = 0;
57. }
58. **for** (**int** m = 0; m < 3; m++) {
59. img\_x.at<Vec3b>(i, j)[m] = temp;
60. }
61. }
62. }
64. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
65. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
66. **for** (**int** m = 0; m < 3; m++) {
67. img\_temp.at<Vec3b>(i, j)[m] = max(img\_x.at<Vec3b>(i, j)[0], img\_y.at<Vec3b>(i, j)[0]);
68. }
69. }
70. }
71. }

**Rgb2Gray.h**

1. #pragma once
2. #include <iostream>
3. #include <highgui.hpp>
4. #include <core.hpp>
5. #include <cstring>
7. **using** **namespace** std;
8. **using** **namespace** cv;
9. **class** Rgb2Gray
10. {
11. **public**:
12. //我设计的主类的目的 主要是为子类提供一个Mat img
13. Mat img;
14. Rgb2Gray();
15. **virtual** ~Rgb2Gray();
16. };

**Rgb2Gray.cpp**

1. #include "Rgb2Gray.h"
3. Rgb2Gray::Rgb2Gray()
4. {
5. }

8. Rgb2Gray::~Rgb2Gray()
9. {
10. }

**Rgb2Gray\_avr.h**

1. #pragma once
2. #include "Rgb2Gray.h"
3. **class** Rgb2Gray\_avr :
4. **public** Rgb2Gray
5. {
6. **public**:
7. Rgb2Gray\_avr();
8. ~Rgb2Gray\_avr();
9. **void** Rgb2Gray\_avr\_processing();
10. };

**Rgb2Gray\_avr.cpp**

1. #include "Rgb2Gray\_avr.h"
3. Rgb2Gray\_avr::Rgb2Gray\_avr()
4. {
5. }
7. Rgb2Gray\_avr::~Rgb2Gray\_avr()
8. {
9. }
11. **void** Rgb2Gray\_avr::Rgb2Gray\_avr\_processing()
12. {
13. **for** (**int** i = 0; i < img.rows; i++) {
14. **for** (**int** j = 0; j < img.cols; j++) {
15. **for** (**int** k = 0; k < 3; k++) {
16. img.at<Vec3b>(i, j)[k] = (img.at<Vec3b>(i, j)[0] + img.at<Vec3b>(i, j)[1] + img.at<Vec3b>(i, j)[2]) / 3;
17. }
18. }
19. }
20. }

**Rgb2Gray\_max.h**

1. #pragma once
2. #include "Rgb2Gray.h"
3. **class** Rgb2Gray\_max :
4. **public** Rgb2Gray
5. {
6. **public**:
7. Rgb2Gray\_max();
8. ~Rgb2Gray\_max();
9. **void** Rgb2Gray\_max\_processing();
10. **int** max(**int** a, **int** b) {
11. **if** (a >= b)
12. **return** a;
13. **return** b;
14. }
15. };

**Rgb2Gray\_max.cpp**

1. #include "Rgb2Gray\_max.h"
3. Rgb2Gray\_max::Rgb2Gray\_max()
4. {
5. }
7. Rgb2Gray\_max::~Rgb2Gray\_max()
8. {
9. }
11. **void** Rgb2Gray\_max::Rgb2Gray\_max\_processing(){
12. **for** (**int** i = 0; i < img.rows; i++) {
13. **for** (**int** j = 0; j < img.cols; j++) {
14. **for** (**int** k = 0; k < 3; k++) {
15. img.at<Vec3b>(i, j)[k] = max(max(img.at<Vec3b>(i, j)[0], img.at<Vec3b>(i, j)[1]), img.at<Vec3b>(i, j)[2]);
16. }
17. }
18. }
19. }

**Rgb2Gray\_param.h**

1. #pragma once
2. #include "Rgb2Gray.h"
3. **class** Rgb2Gray\_param :
4. **public** Rgb2Gray
5. {
6. **public**:
7. Rgb2Gray\_param();
8. ~Rgb2Gray\_param();
9. **void** Rgb2Gray\_param\_processing(String way);
10. };

**Rgb2Gray\_param.cpp**

1. #include "Rgb2Gray\_param.h"
3. Rgb2Gray\_param::Rgb2Gray\_param()
4. {
5. }
7. Rgb2Gray\_param::~Rgb2Gray\_param()
8. {
9. }
11. **void** Rgb2Gray\_param::Rgb2Gray\_param\_processing(String way){
12. **if** (way.compare("avr") == 0) {
13. **for** (**int** i = 0; i < img.rows; i++) {
14. **for** (**int** j = 0; j < img.cols; j++) {
15. **for** (**int** k = 0; k < 3; k++) {
16. img.at<Vec3b>(i, j)[k] = (img.at<Vec3b>(i, j)[0] + img.at<Vec3b>(i, j)[1] + img.at<Vec3b>(i, j)[2]) / 3;
17. }
18. }
19. }
20. }
21. **else** **if** (way.compare("max") == 0) {
22. **for** (**int** i = 0; i < img.rows; i++) {
23. **for** (**int** j = 0; j < img.cols; j++) {
24. **for** (**int** k = 0; k < 3; k++) {
25. img.at<Vec3b>(i, j)[k] = max(max(img.at<Vec3b>(i, j)[0], img.at<Vec3b>(i, j)[1]), img.at<Vec3b>(i, j)[2]);
26. }
27. }
28. }
29. }
30. **else** **if** (way.compare("weighting") == 0) {
31. **for** (**int** i = 0; i < img.rows; i++) {
32. **for** (**int** j = 0; j < img.cols; j++) {
33. **for** (**int** k = 0; k < 3; k++) {
34. img.at<Vec3b>(i, j)[k] = img.at<Vec3b>(i, j)[0] \* 0.30 + img.at<Vec3b>(i, j)[1] \* 0.59 + img.at<Vec3b>(i, j)[2] \* 0.11;
35. }
36. }
37. }
38. }
39. **else** {
40. cout << "wrong input param";
41. }
42. }

**Rgb2Gray\_weighting.h**

1. #pragma once
2. #include "Rgb2Gray.h"
3. **class** Rgb2Gray\_weighting :
4. **public** Rgb2Gray
5. {
6. **public**:
7. Rgb2Gray\_weighting();
8. ~Rgb2Gray\_weighting();
9. **void** Rgb2Gray\_weighting\_processing();
10. };

**Rgb2Gray\_weighting.cpp**

1. #include "Rgb2Gray\_weighting.h"
3. Rgb2Gray\_weighting::Rgb2Gray\_weighting()
4. {
5. }
7. Rgb2Gray\_weighting::~Rgb2Gray\_weighting()
8. {
9. }
11. **void** Rgb2Gray\_weighting::Rgb2Gray\_weighting\_processing(){
12. **for** (**int** i = 0; i < img.rows; i++) {
13. **for** (**int** j = 0; j < img.cols; j++) {
14. **for** (**int** k = 0; k < 3; k++) {
15. img.at<Vec3b>(i, j)[k] = img.at<Vec3b>(i, j)[0] \* 0.30 + img.at<Vec3b>(i, j)[1] \* 0.59 + img.at<Vec3b>(i, j)[2] \* 0.11;
16. }
17. }
18. }
19. }

**Roberts.h**

1. #pragma once
2. #include <iostream>
3. #include <highgui.hpp>
4. #include <core.hpp>
5. #include <cstring>
7. **using** **namespace** std;
8. **using** **namespace** cv;
9. **class** Roberts
10. {
11. **public**:
12. Mat img;
13. Mat img\_temp;
14. Mat img\_x;
15. Mat img\_y;
16. Roberts();
17. ~Roberts();
18. **void** Roberts\_processing();
19. **int** max(**int** a, **int** b) {
20. **if** (a >= b)
21. **return** a;
22. **return** b;
23. }
24. };

**Roberts.cpp**

1. #include "Roberts.h"
3. Roberts::Roberts()
4. {
5. }
7. Roberts::~Roberts() {
9. }
11. **void** Roberts::Roberts\_processing(){
12. **int** oper\_y[2][2] = { { 0,-1 },{ 1,0 } };
13. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
14. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
15. **int** temp = 0;//用来计算新的中心像素点值
16. **for** (**int** k = 0; k < 2; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
17. **for** (**int** l = 0; l < 2; l++) {
18. **if** ((i + k < 0) || (j + l < 0) || (i + k >= img.rows) || (j + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
19. **continue**;
20. temp += (oper\_y[k][l]) \* img.at<Vec3b>(i + k, j + l)[0];
21. }
22. }
23. **if** (temp > 255) {
24. temp = 255;
25. }
26. //当计算结果小于0，就把它置为0
27. **if** (temp < 0) {
28. temp = 0;
29. }
30. **for** (**int** m = 0; m < 3; m++) {
31. img\_y.at<Vec3b>(i, j)[m] = temp;
32. }
33. }
34. }
36. **int** oper\_x[2][2] = { { -1,0 },{ 0,1 } };
37. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
38. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
39. **int** temp = 0;//用来计算新的中心像素点值
40. **for** (**int** k = 0; k < 2; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
41. **for** (**int** l = 0; l < 2; l++) {
42. **if** ((i + k < 0) || (j + l < 0) || (i + k >= img.rows) || (j + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
43. **continue**;
44. temp += (oper\_x[k][l]) \* img.at<Vec3b>(i + k, j + l)[0];
45. }
46. }
47. **if** (temp > 255) {
48. temp = 255;
49. }
50. //当计算结果小于0，就把它置为0
51. **if** (temp < 0) {
52. temp = 0;
53. }
54. **for** (**int** m = 0; m < 3; m++) {
55. img\_x.at<Vec3b>(i, j)[m] = temp;
56. }
57. }
58. }
60. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
61. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
62. **for** (**int** m = 0; m < 3; m++) {
63. img\_temp.at<Vec3b>(i, j)[m] = max(img\_x.at<Vec3b>(i, j)[0], img\_y.at<Vec3b>(i, j)[0]);
64. }
65. }
66. }
67. }

**Sharpening.h**

1. #pragma once
2. #include <iostream>
3. #include <highgui.hpp>
4. #include <core.hpp>
5. #include <cstring>
7. **using** **namespace** std;
8. **using** **namespace** cv;
9. **class** Sharpening
10. {
11. **public**:
12. Mat img;
13. Mat img\_temp;
14. Sharpening();
15. ~Sharpening();
16. **void** Sharpening\_processing();
17. };

**Sharpening.cpp**

1. #include "Sharpening.h"
3. Sharpening::Sharpening()
4. {
5. }
7. Sharpening::~Sharpening()
8. {
9. }
11. **void** Sharpening::Sharpening\_processing(){
12. //由于n = 3 所以 x = 1
13. **int** oper[3][3] = { { -1,-1,-1 },{ -1,9,-1 },{ -1,-1,-1 } };
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 3; l++) {
19. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
22. }
23. }
24. **if** (temp > 255) {
25. temp = 255;
26. }
27. //当计算结果小于0，就把它置为0
28. **if** (temp < 0) {
29. temp = 0;
30. }
31. **for** (**int** m = 0; m < 3; m++) {
32. img\_temp.at<Vec3b>(i, j)[m] = temp;
33. }
34. }
35. }
36. }

**Sharpening\_5.h**

1. #pragma once
2. #include "Sharpening.h"
3. **class** Sharpening\_5 :
4. **public** Sharpening
5. {
6. **public**:
7. Sharpening\_5();
8. ~Sharpening\_5();
9. **void** Sharpening\_5\_processing();
10. };

**Sharpening\_5.cpp**

1. #include "Sharpening\_5.h"
3. Sharpening\_5::Sharpening\_5()
4. {
5. }
7. Sharpening\_5::~Sharpening\_5()
8. {
9. }
11. **void** Sharpening\_5::Sharpening\_5\_processing(){
12. //由于n = 3 所以 x = 1
13. **int** oper[5][5] = { { -1,-1,-1,-1,-1 },{ -1,-1,-1,-1,-1 },{ -1,-1,21,-1,-1 },{ -1,-1,-1,-1,-1 },{ -1,-1,-1,-1,-1 } };
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 5; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 5; l++) {
19. **if** ((i - 2 + k < 0) || (j - 2 + l < 0) || (i - 2 + k >= img.rows) || (j - 2 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l]) \* img.at<Vec3b>(i - 2 + k, j - 2 + l)[0];
22. }
23. }
24. **if** (temp > 255) {
25. temp = 255;
26. }
27. //当计算结果小于0，就把它置为0
28. **if** (temp < 0) {
29. temp = 0;
30. }
31. **for** (**int** m = 0; m < 3; m++) {
32. img\_temp.at<Vec3b>(i, j)[m] = temp;
33. }
34. }
35. }
36. }

**Sharpening\_double.h**

1. #pragma once
2. #include "Sharpening.h"
3. **class** Sharpening\_double :
4. **public** Sharpening
5. {
6. **public**:
7. Sharpening\_double();
8. ~Sharpening\_double();
9. **void** Sharpening\_double\_processing();
10. };

**Sharpening\_double.cpp**

1. #include "Sharpening\_double.h"
3. Sharpening\_double::Sharpening\_double()
4. {
5. }
7. Sharpening\_double::~Sharpening\_double()
8. {
9. }
11. **void** Sharpening\_double::Sharpening\_double\_processing(){
12. //由于n = 3 所以 x = 1
13. **int** oper[3][3] = { { -2,-2,-2 },{ -2,18,-2 },{ -2,-2,-2 } };
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 3; l++) {
19. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
22. }
23. }
24. **if** (temp > 255) {
25. temp = 255;
26. }
27. //当计算结果小于0，就把它置为0
28. **if** (temp < 0) {
29. temp = 0;
30. }
31. **for** (**int** m = 0; m < 3; m++) {
32. img\_temp.at<Vec3b>(i, j)[m] = temp;
33. }
34. }
35. }
36. }

**Sharpening\_param.h**

1. #pragma once
2. #include "Sharpening.h"
3. **class** Sharpening\_param :
4. **public** Sharpening
5. {
6. **public**:
7. Sharpening\_param();
8. ~Sharpening\_param();
9. **void** Sharpening\_param\_processing(**int** oper[3][3]);
10. };

**Sharpening\_param.cpp**

1. #include "Sharpening\_param.h"
3. Sharpening\_param::Sharpening\_param()
4. {
5. }
7. Sharpening\_param::~Sharpening\_param()
8. {
9. }
11. **void** Sharpening\_param::Sharpening\_param\_processing(**int** oper[3][3]){
12. //由于n = 3 所以 x = 1
13. //int oper[3][3] = { { -1,-1,-1 },{ -1,9,-1 },{ -1,-1,-1 } };
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 3; l++) {
19. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
22. }
23. }
24. **if** (temp > 255) {
25. temp = 255;
26. }
27. //当计算结果小于0，就把它置为0
28. **if** (temp < 0) {
29. temp = 0;
30. }
31. **for** (**int** m = 0; m < 3; m++) {
32. img\_temp.at<Vec3b>(i, j)[m] = temp;
33. }
34. }
35. }
36. }

**Smoothing.h**

1. #pragma once
2. #include <iostream>
3. #include <highgui.hpp>
4. #include <core.hpp>
5. #include <cstring>
6. #include<algorithm>//
8. **using** **namespace** std;
9. **using** **namespace** cv;
10. **class** Smoothing
11. {
12. **public**:
13. Mat img;
14. Mat img\_temp;
15. Smoothing();
16. ~Smoothing();
17. **void** Smoothing\_processing();
18. };

**Smoothing.cpp**

1. #include "Smoothing.h"
3. Smoothing::Smoothing()
4. {
5. }
7. Smoothing::~Smoothing()
8. {
9. }
11. **void** Smoothing::Smoothing\_processing(){
12. **int** oper[3][3];//平滑算子 n = 3
13. memset(oper, 0, **sizeof**(oper));//全部初始化为0
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 3; l++){
19. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l] + 1) \*img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
22. }
23. }
24. temp /= (3 \* 3);//之所以放后面是因为1 / 9 的影响
25. //将计算结果赋给dstimg中心像素点的三通道值
26. **for** (**int** m = 0; m < 3; m++) {
27. img\_temp.at<Vec3b>(i, j)[m] = temp;
28. }
29. }
30. }
31. }

**Smoothing\_5.h**

1. #pragma once
2. #include "Smoothing.h"
3. **class** Smoothing\_5 :
4. **public** Smoothing
5. {
6. **public**:
7. Smoothing\_5();
8. ~Smoothing\_5();
9. **void** Smoothing\_5\_processing();
10. };

**Smoothing\_5.cpp**

1. #include "Smoothing\_5.h"
3. Smoothing\_5::Smoothing\_5()
4. {
5. }
7. Smoothing\_5::~Smoothing\_5()
8. {
9. }
11. **void** Smoothing\_5::Smoothing\_5\_processing(){
12. **int** oper[5][5];//平滑算子 n = 3
13. memset(oper, 0, **sizeof**(oper));//全部初始化为0
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 5; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 5; l++) {
19. **if** ((i - 2 + k < 0) || (j - 2 + l < 0) || (i - 2 + k >= img.rows) || (j - 2 + l >= img.cols))//当算子位于边缘区域时，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l] + 1) \*img.at<Vec3b>(i - 2 + k, j - 2 + l)[0];
22. }
23. }
24. temp /= (5 \* 5);//之所以放后面是因为1 / 9 的影响
25. //将计算结果赋给dstimg中心像素点的三通道值
26. **for** (**int** m = 0; m < 3; m++) {
27. img\_temp.at<Vec3b>(i, j)[m] = temp;
28. }
29. }
30. }
31. }

**Smoothing\_Gauss\_3.h**

1. #pragma once
2. #include "Smoothing.h"
3. **class** Smoothing\_Gauss\_3 :
4. **public** Smoothing
5. {
6. **public**:
7. Smoothing\_Gauss\_3();
8. ~Smoothing\_Gauss\_3();
9. **void** Smoothing\_Gauss\_3\_processing();
10. };

**Smoothing\_Gauss\_3.cpp**

1. #include "Smoothing\_Gauss\_3.h"
3. Smoothing\_Gauss\_3::Smoothing\_Gauss\_3()
4. {
5. }
7. Smoothing\_Gauss\_3::~Smoothing\_Gauss\_3()
8. {
9. }
11. **void** Smoothing\_Gauss\_3::Smoothing\_Gauss\_3\_processing(){
12. //由于n = 3 所以 x = 1
13. **int** oper[3][3] = { { 1,2,1 },{ 2,4,2 },{ 1,2,1 } };
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 3; l++) {
19. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
22. }
23. }
24. temp = temp / 16;
25. **if** (temp > 255) {
26. temp = 255;
27. }
28. //当计算结果小于0，就把它置为0
29. **if** (temp < 0) {
30. temp = 0;
31. }
32. **for** (**int** m = 0; m < 3; m++) {
33. img\_temp.at<Vec3b>(i, j)[m] = temp;
34. }
35. }
36. }
37. }

**Smoothing\_Gauss\_5.h**

1. #pragma once
2. #include "Smoothing.h"
3. **class** Smoothing\_Gauss\_5 :
4. **public** Smoothing
5. {
6. **public**:
7. Smoothing\_Gauss\_5();
8. ~Smoothing\_Gauss\_5();
9. **void** Smoothing\_Gauss\_5\_processing();
10. };

**Smoothing\_Gauss\_5.cpp**

1. #include "Smoothing\_Gauss\_5.h"
3. Smoothing\_Gauss\_5::Smoothing\_Gauss\_5()
4. {
5. }
7. Smoothing\_Gauss\_5::~Smoothing\_Gauss\_5()
8. {
9. }
11. **void** Smoothing\_Gauss\_5::Smoothing\_Gauss\_5\_processing(){
12. //由于n = 3 所以 x = 1
13. **int** oper[5][5] = { { 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 } };
14. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
15. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
16. **int** temp = 0;//用来计算新的中心像素点值
17. **for** (**int** k = 0; k < 5; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
18. **for** (**int** l = 0; l < 5; l++) {
19. **if** ((i - 2 + k < 0) || (j - 2 + l < 0) || (i - 2 + k >= img.rows) || (j - 2 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
20. **continue**;
21. temp += (oper[k][l]) \* img.at<Vec3b>(i - 2 + k, j - 2 + l)[0];
22. }
23. }
24. temp /= 273;
25. **if** (temp > 255) {
26. temp = 255;
27. }
28. //当计算结果小于0，就把它置为0
29. **if** (temp < 0) {
30. temp = 0;
31. }
32. **for** (**int** m = 0; m < 3; m++) {
33. img\_temp.at<Vec3b>(i, j)[m] = temp;
34. }
35. }
36. }
37. }

**Smoothing\_mid.h**

1. #pragma once
2. #include "Smoothing.h"
3. **class** Smoothing\_mid :
4. **public** Smoothing
5. {
6. **public**:
7. Smoothing\_mid();
8. ~Smoothing\_mid();
9. **void** Smoothing\_mid\_processing();
10. };

**Smoothing\_mid.cpp**

1. #include "Smoothing\_mid.h"
3. Smoothing\_mid::Smoothing\_mid()
4. {
5. }
7. Smoothing\_mid::~Smoothing\_mid()
8. {
9. }
11. **void** Smoothing\_mid::Smoothing\_mid\_processing(){
12. **int** oper[9];//用来存放模板对应的9个像素点的值
13. memset(oper, 0, **sizeof**(oper));//初始化为0
14. **for** (**int** i = 0; i < img.rows; i++) {
15. **for** (**int** j = 0; j < img.cols; j++) {
16. **int** count = 0;//标记共有多少个合法点
17. **for** (**int** k = 0; k < 3; k++) {
18. **for** (**int** l = 0; l < 3; l++) {
19. //当算子位于边缘区域时，排除图像中不存在的点
20. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))
21. **continue**;
22. oper[count] = img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
23. count++;
24. }
25. }
26. //将temp数组中的像素值从小到大排序
27. sort(oper, oper + count);
28. //将temp数组的中值赋给中心像素点的三通道
29. **for** (**int** m = 0; m < 3; m++) {
30. img\_temp.at<Vec3b>(i, j)[m] = oper[(count - 1) / 2];//求得中位数
31. }
32. }
33. }
34. }

**Smoothing\_param.h**

1. #pragma once
2. #include "Smoothing.h"
3. **class** Smoothing\_param :
4. **public** Smoothing
5. {
6. **public**:
7. Smoothing\_param();
8. ~Smoothing\_param();
9. **void** Smoothing\_param\_processing(**int** n);
10. };

**Smoothing\_param.cpp**

1. #include "Smoothing\_param.h"
3. Smoothing\_param::Smoothing\_param()
4. {
5. }
7. Smoothing\_param::~Smoothing\_param()
8. {
9. }
11. **void** Smoothing\_param::Smoothing\_param\_processing(**int** n) {
12. **int** \* oper;
13. oper = **new** **int**[n \* n]() ;
14. **int** x = (n - 1) / 2;
15. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
16. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
17. **int** temp = 0;//用来计算新的中心像素点值
18. **for** (**int** k = 0; k < n; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
19. **for** (**int** l = 0; l < n; l++) {
20. **if** ((i - x + k < 0) || (j - x + l < 0) || (i - x + k >= img.rows) || (j - x + l >= img.cols))//当算子位于边缘区域时，排除图像中不存在的点
21. **continue**;
22. temp += (oper[k \* n + l] + 1) \*img.at<Vec3b>(i - x + k, j - x + l)[0];
23. }
24. }
25. temp /= (n \* n);//之所以放后面是因为1 / 9 的影响
26. //将计算结果赋给dstimg中心像素点的三通道值
27. **for** (**int** m = 0; m < 3; m++) {
28. img\_temp.at<Vec3b>(i, j)[m] = temp;
29. }
30. }
31. }
32. **delete** oper;
33. }

**Sobel.h**

1. #pragma once
2. #include <iostream>
3. #include <highgui.hpp>
4. #include <core.hpp>
5. #include <cstring>
7. **using** **namespace** std;
8. **using** **namespace** cv;
9. **class** Sobel
10. {
11. **public**:
12. Mat img;
13. Mat img\_temp;
14. Mat img\_x;
15. Mat img\_y;
16. Sobel();
17. ~Sobel();
18. **void** Sobel\_processing();
19. **int** max(**int** a, **int** b) {
20. **if** (a >= b)
21. **return** a;
22. **return** b;
23. }
24. };

**Sobel.cpp**

1. #include "Sobel.h"
3. Sobel::Sobel()
4. {
5. }
7. Sobel::~Sobel()
8. {
9. }
11. **void** Sobel::Sobel\_processing(){
12. **int** oper\_x[3][3] = { { -1,-2,-1 },{ 0,0,0 },{ 1,2,1 } };
13. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
14. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
15. **int** temp = 0;//用来计算新的中心像素点值
16. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
17. **for** (**int** l = 0; l < 3; l++) {
18. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
19. **continue**;
20. temp += (oper\_x[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
21. }
22. }
23. **if** (temp > 255) {
24. temp = 255;
25. }
26. //当计算结果小于0，就把它置为0
27. **if** (temp < 0) {
28. temp = 0;
29. }
30. **for** (**int** m = 0; m < 3; m++) {
31. img\_x.at<Vec3b>(i, j)[m] = temp;
32. }
33. }
34. }
36. **int** oper\_y[3][3] = { { -1,0,1 },{ -2,0,2 },{ -1,0,1 } };
37. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
38. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
39. **int** temp = 0;//用来计算新的中心像素点值
40. **for** (**int** k = 0; k < 3; k++) {//程序第一次运行时，算子开始扫，并且排除最边缘点
41. **for** (**int** l = 0; l < 3; l++) {
42. **if** ((i - 1 + k < 0) || (j - 1 + l < 0) || (i - 1 + k >= img.rows) || (j - 1 + l >= img.cols))//当算子位于边缘区域时,不变，排除图像中不存在的点
43. **continue**;
44. temp += (oper\_y[k][l]) \* img.at<Vec3b>(i - 1 + k, j - 1 + l)[0];
45. }
46. }
47. **if** (temp > 255) {
48. temp = 255;
49. }
50. //当计算结果小于0，就把它置为0
51. **if** (temp < 0) {
52. temp = 0;
53. }
54. **for** (**int** m = 0; m < 3; m++) {
55. img\_y.at<Vec3b>(i, j)[m] = temp;
56. }
57. }
58. }
60. **for** (**int** i = 0; i < img.rows; i++) {//第 i = 1 行开始
61. **for** (**int** j = 0; j < img.cols; j++) {//i = 1 扫描所有 j 列
62. **for** (**int** m = 0; m < 3; m++) {
63. img\_temp.at<Vec3b>(i, j)[m] = max(img\_x.at<Vec3b>(i, j)[0], img\_y.at<Vec3b>(i, j)[0]);
64. }
65. }
66. }
67. }

**main.cpp**

1. #include <iostream>
2. #include <highgui.hpp>
3. #include <core.hpp>
4. #include <cstring>
5. #include "Rgb2Gray.h"
6. #include "Rgb2Gray\_avr.h"
7. #include "Rgb2Gray\_max.h"
8. #include "Rgb2Gray\_weighting.h"
9. #include "Smoothing.h"
10. #include "Sharpening.h"
11. #include "Smoothing\_5.h"
12. #include "Sharpening\_5.h"
13. #include "Sharpening\_double.h"
15. #include "Rgb2Gray\_param.h"
16. #include "Sharpening\_param.h"
17. #include "Smoothing\_param.h"
19. #include "Smoothing\_mid.h"
20. #include "Smoothing\_Gauss\_3.h"
21. #include "Smoothing\_Gauss\_5.h"
23. #include "Prewitt.h"
24. #include "Sobel.h"
25. #include "Roberts.h"
27. **using** **namespace** std;
28. **using** **namespace** cv;
29. **int** main(**int** argc, **char**\*\* argv)
30. {
31. String filename = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\锐化.jpg";
33. //平均值灰度图
34. String filename\_Rgb2Gray\_avr = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Rgb2Gray\_avr.jpg";
35. Rgb2Gray\_avr rgb2Gray\_avr;//创建类的对象
36. rgb2Gray\_avr.img = imread(filename, IMREAD\_COLOR);//将类的img属性读取到值
37. rgb2Gray\_avr.Rgb2Gray\_avr\_processing();//对img处理——平均值灰度值
38. imwrite(filename\_Rgb2Gray\_avr, rgb2Gray\_avr.img);//将img读出 到"D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Lenna\_Rgb2Gray\_avr.jpg"中存着

41. //最大值得到灰度
42. String filename\_Rgb2Gray\_max = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Rgb2Gray\_max.jpg";
43. Rgb2Gray\_max rgb2Gray\_max;
44. rgb2Gray\_max.img = imread(filename, IMREAD\_COLOR);
45. rgb2Gray\_max.Rgb2Gray\_max\_processing();
46. imwrite(filename\_Rgb2Gray\_max, rgb2Gray\_max.img);
48. //加权法得到灰度
49. String filename\_Rgb2Gray\_weighting = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Rgb2Gray\_weighting.jpg";
50. Rgb2Gray\_weighting rgb2Gray\_weighting;
51. rgb2Gray\_weighting.img = imread(filename, IMREAD\_COLOR);
52. rgb2Gray\_weighting.Rgb2Gray\_weighting\_processing();
53. imwrite(filename\_Rgb2Gray\_weighting, rgb2Gray\_weighting.img);
55. /\*
56. 考虑到加权灰度法得到的结果比较好 我们后续都使用加权法得到的灰度图进行后续实验
57. \*/
59. /\*
60. 平滑
61. 1/9 1/9 1/9
62. 1/9 1/9 1/9
63. 1/9 1/9 1/9
64. \*/
65. String filename\_Smoothing = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Smoothing.jpg";
66. Smoothing smoothing;
67. smoothing.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
68. smoothing.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
69. smoothing.Smoothing\_processing();
70. imwrite(filename\_Smoothing, smoothing.img\_temp);
72. /\*
73. 锐化
74. -1 -1 -1
75. -1  9 -1
76. -1 -1 -1
77. \*/
78. String filename\_Sharpening = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Sharpening.jpg";
79. Sharpening sharpening;
80. sharpening.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
81. sharpening.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
82. sharpening.Sharpening\_processing();
83. imwrite(filename\_Sharpening, sharpening.img\_temp);
85. /\*
86. 平滑 5
87. 1/25 1/25 1/25 1/25 1/25
88. 1/25 1/25 1/25 1/25 1/25
89. 1/25 1/25 1/25 1/25 1/25
90. 1/25 1/25 1/25 1/25 1/25
91. 1/25 1/25 1/25 1/25 1/25
92. \*/
93. String filename\_Smoothing\_5 = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Smoothing\_5.jpg";
94. Smoothing\_5 smoothing\_5;
95. smoothing\_5.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
96. smoothing\_5.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
97. smoothing\_5.Smoothing\_5\_processing();
98. imwrite(filename\_Smoothing\_5, smoothing\_5.img\_temp);
100. /\*
101. 锐化 5 尝试在 23 时候效果最好
102. -1 -1 -1 -1 -1
103. -1 -1 -1 -1 -1
104. -1 -1 23 -1 -1
105. -1 -1 -1 -1 -1
106. -1 -1 -1 -1 -1
107. \*/
108. String filename\_Sharpening\_5 = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Sharpening\_5.jpg";
109. Sharpening\_5 sharpening\_5;
110. sharpening\_5.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
111. sharpening\_5.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
112. sharpening\_5.Sharpening\_5\_processing();
113. imwrite(filename\_Sharpening\_5, sharpening\_5.img\_temp);
115. /\*
116. 锐化 double
117. -2 -2 -2
118. -2 18 -2
119. -2 -2 -2
120. \*/
121. String filename\_Sharpening\_double = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Sharpening\_double.jpg";
122. Sharpening\_double sharpening\_double;
123. sharpening\_double.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
124. sharpening\_double.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
125. sharpening\_double.Sharpening\_double\_processing();
126. imwrite(filename\_Sharpening\_double, sharpening\_double.img\_temp);
128. //参数灰度
129. String filename\_Rgb2Gray\_param = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Rgb2Gray\_param.jpg";
130. Rgb2Gray\_param rgb2Gray\_param;
131. rgb2Gray\_param.img = imread(filename, IMREAD\_COLOR);
132. rgb2Gray\_param.Rgb2Gray\_param\_processing("max");//此处参数可选：avr max weight valid\_input等
133. imwrite(filename\_Rgb2Gray\_param, rgb2Gray\_param.img);
135. //参数锐化
136. String filename\_Sharpening\_param = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Sharpening\_param.jpg";
137. Sharpening\_param sharpening\_param;
138. sharpening\_param.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
139. sharpening\_param.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
140. **int** oper[3][3] = { { -1,-1,-1 },{ -1,9,-1 },{ -1,-1,-1 } };
141. sharpening\_param.Sharpening\_param\_processing(oper);
142. imwrite(filename\_Sharpening\_param, sharpening\_param.img\_temp);
144. //参数平滑
145. String filename\_Smoothing\_param = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Smoothing\_param.jpg";
146. Smoothing\_param smoothing\_param;
147. smoothing\_param.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
148. smoothing\_param.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
149. smoothing\_param.Smoothing\_param\_processing(3);
150. imwrite(filename\_Smoothing\_param, smoothing\_param.img\_temp);
152. //以下为选做部分代码
153. //中值平滑
154. String filename\_Smoothing\_mid = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Smoothing\_mid.jpg";
155. Smoothing\_mid smoothing\_mid;
156. smoothing\_mid.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
157. smoothing\_mid.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
158. smoothing\_mid.Smoothing\_mid\_processing();
159. imwrite(filename\_Smoothing\_mid, smoothing\_mid.img\_temp);
161. //Gauss\_3平滑
162. String filename\_Smoothing\_Gauss\_3 = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Smoothing\_Gauss\_3.jpg";
163. Smoothing\_Gauss\_3 smoothing\_Gauss\_3;
164. smoothing\_Gauss\_3.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
165. smoothing\_Gauss\_3.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
166. smoothing\_Gauss\_3.Smoothing\_Gauss\_3\_processing();
167. imwrite(filename\_Smoothing\_Gauss\_3, smoothing\_Gauss\_3.img\_temp);
169. //Gauss\_5平滑
170. String filename\_Smoothing\_Gauss\_5 = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Smoothing\_Gauss\_5.jpg";
171. Smoothing\_Gauss\_5 smoothing\_Gauss\_5;
172. smoothing\_Gauss\_5.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
173. smoothing\_Gauss\_5.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
174. smoothing\_Gauss\_5.Smoothing\_Gauss\_5\_processing();
175. imwrite(filename\_Smoothing\_Gauss\_5, smoothing\_Gauss\_5.img\_temp);
177. //prewitt
178. String filename\_Prewitt = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Prewitt.jpg";
179. String filename\_Prewitt\_x = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Prewitt\_x.jpg";
180. String filename\_Prewitt\_y = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Prewitt\_y.jpg";
181. Prewitt prewitt;
182. prewitt.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
183. prewitt.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
184. prewitt.img\_x = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
185. prewitt.img\_y = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
186. prewitt.Prewitt\_processing();
187. imwrite(filename\_Prewitt, prewitt.img\_temp);
188. imwrite(filename\_Prewitt\_x, prewitt.img\_x);
189. imwrite(filename\_Prewitt\_y, prewitt.img\_y);

192. //sobel
193. String filename\_Sobel = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Sobel.jpg";
194. String filename\_Sobel\_x = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Sobel\_x.jpg";
195. String filename\_Sobel\_y = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Sobel\_y.jpg";
196. Sobel sobel;
197. sobel.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
198. sobel.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
199. sobel.img\_x = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
200. sobel.img\_y = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
201. sobel.Sobel\_processing();
202. imwrite(filename\_Sobel, sobel.img\_temp);
203. imwrite(filename\_Sobel\_x, sobel.img\_x);
204. imwrite(filename\_Sobel\_y, sobel.img\_y);
206. //roberts
207. String filename\_Roberts = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Roberts.jpg";
208. String filename\_Roberts\_x = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Roberts\_x.jpg";
209. String filename\_Roberts\_y = "D:\\大学作业\\3 下\\课程课程\\数据结构课程设计\\Roberts\_y.jpg";
210. Roberts roberts;
211. roberts.img = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
212. roberts.img\_temp = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
213. roberts.img\_x = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
214. roberts.img\_y = imread(filename\_Rgb2Gray\_weighting, IMREAD\_COLOR);
215. roberts.Roberts\_processing();
216. imwrite(filename\_Roberts, roberts.img\_temp);
217. imwrite(filename\_Roberts\_x, roberts.img\_x);
218. imwrite(filename\_Roberts\_y, roberts.img\_y);
219. waitKey(10000); // Wait for a keystroke in the window
220. **return** 0;
221. }

## 6.2）测试数据

*即列出测试数据集*

**Lenna.jpg**

**pyy.jpg**

**高斯.jpg**

**锐化.jpg**

**噪音.jpg**

## 6.3）运行结果

*上面测试数据输入后程序运行的结果*

**所有运行结果均在上述报告中列出，图形界面的演示视频也有。**