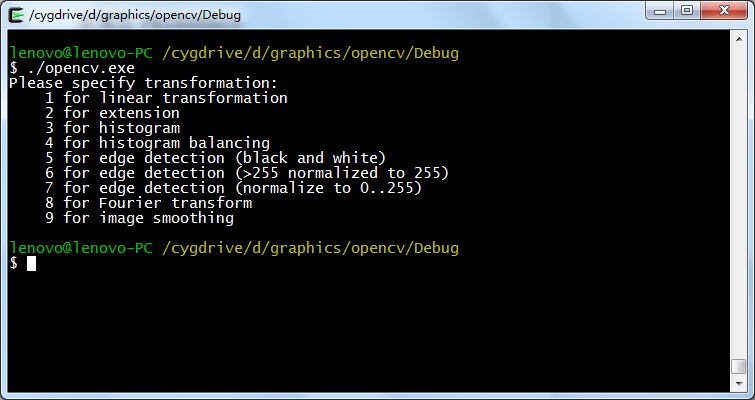
计算机图像学实验报告

李博杰 PB10000603

本实验使用OpenCV的纯C语言接口编程，在Visual Studio 2012下编译。

实验程序截图：（四个实验在一个程序中）

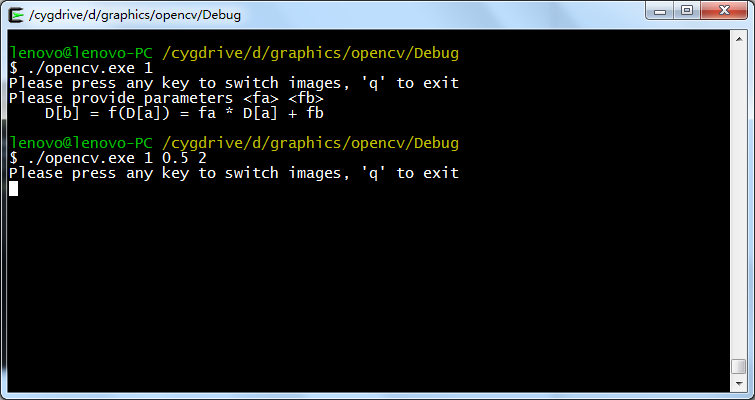


程序用法：首先解压，然后在命令行中启动opencv.exe，给定命令行参数1～9，表示程序所选择的功能。有的功能需要更多的命令行参数。

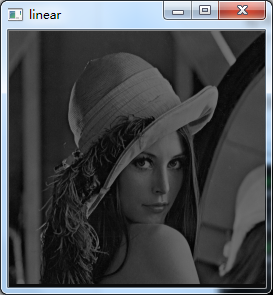
程序会从所在目录的images文件夹下查找图片文件，进行处理（因此images目录是要解压的）。按除q以外的任意键切换到下一张图片（循环显示），按q退出。

实验一

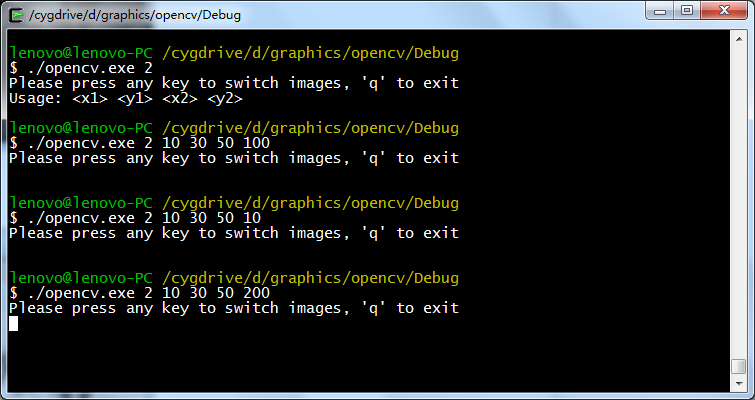
1. 灰度线性变换



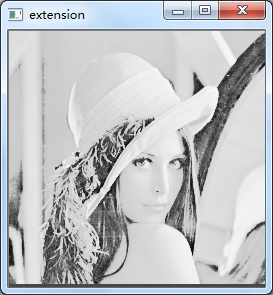
变换效果：



2. 灰度拉伸

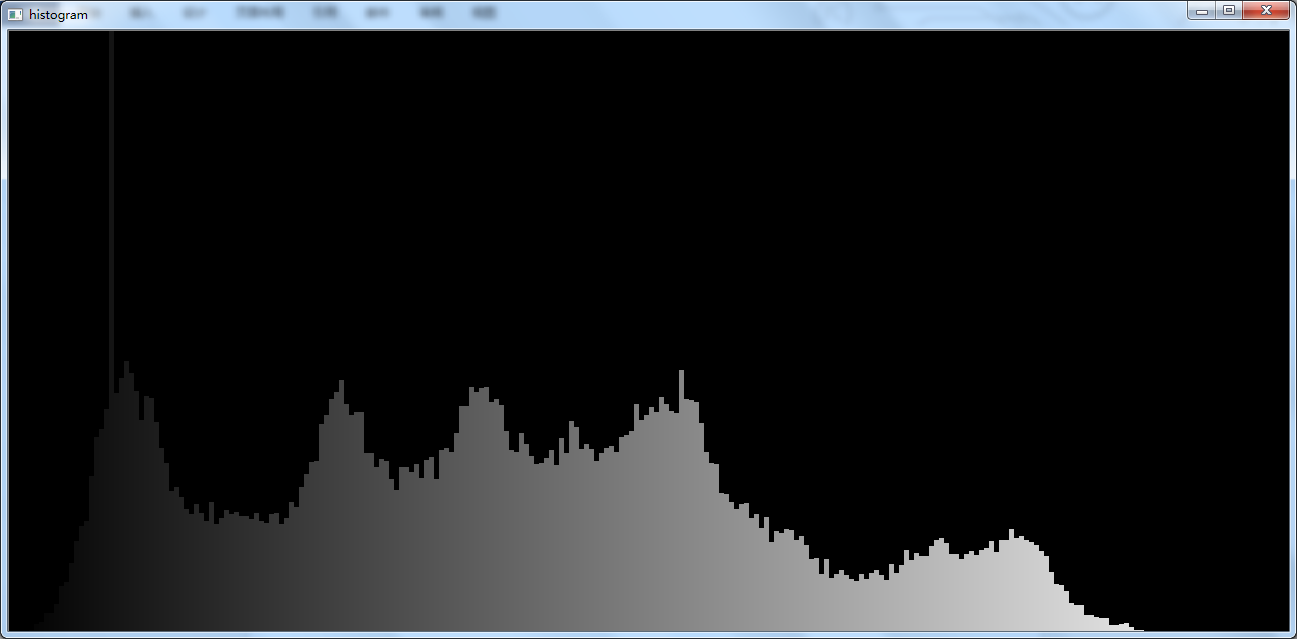


效果：（lena图，取点为10，30，50，200）



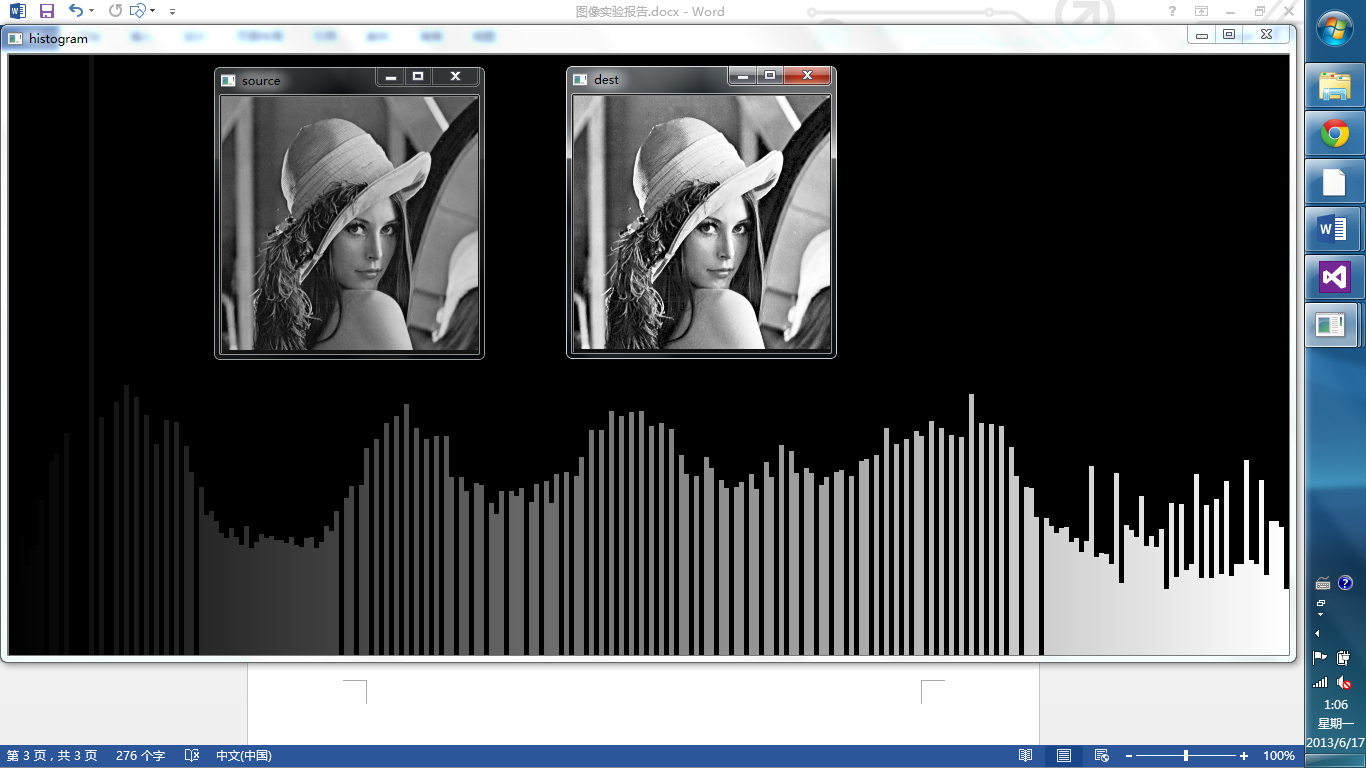
3. 灰度直方图

下图是lena图的灰度直方图，直方图中显示的灰度即为图像中的实际灰度，高度表示出现的次数。令出现次数最多的灰度撑满整个高度，其他灰度按照这个高度进行归一化。



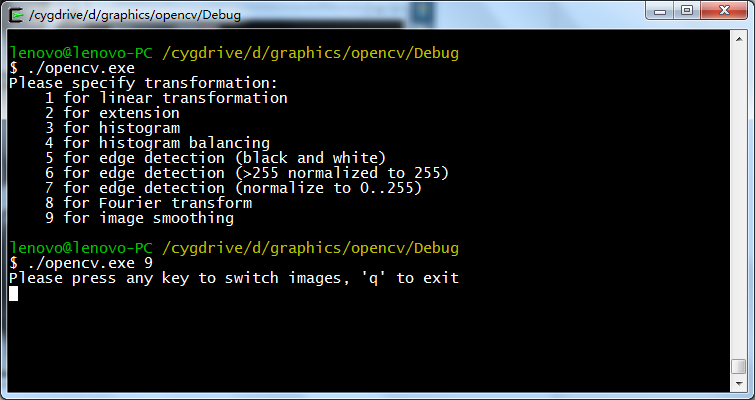
4. 直方图均衡化

lena图：左侧为原图，右侧为均衡化后的图，背景是均衡化后的直方图。



实验二

命令：



处理边界上的像素，是将边界外的像素用最靠近的边界上像素的灰度值代替。

代码中用下面的get宏来实现：

#define fix(i,max) ((i)==(max) ? (max)-1 : ((i)==-1 ? 0 : (i)))

#define get(i,j) \

(cvGet2D(src, fix(i,src->height), fix(j,src->width)).val[0])

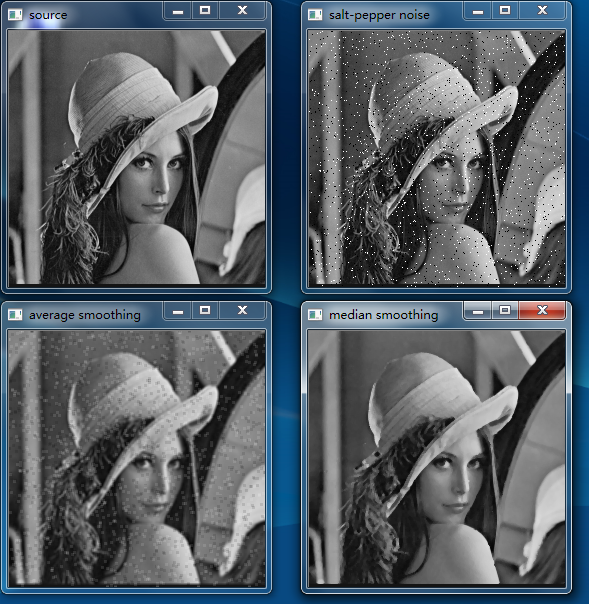
显示的图：

左上为原图

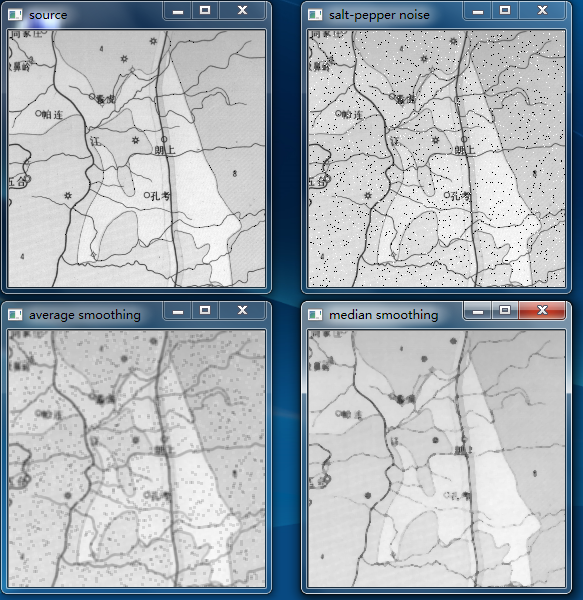
右上为噪声图

左下为均值滤波

右下为中值滤波







对于椒盐噪声，中值滤波的效果比均值滤波好很多。

均值滤波和中值滤波都会使得图像边缘变模糊（从上面的地图可见，字已经看不清了），均值滤波能更好地保持图像的细节（边界）。

实验三

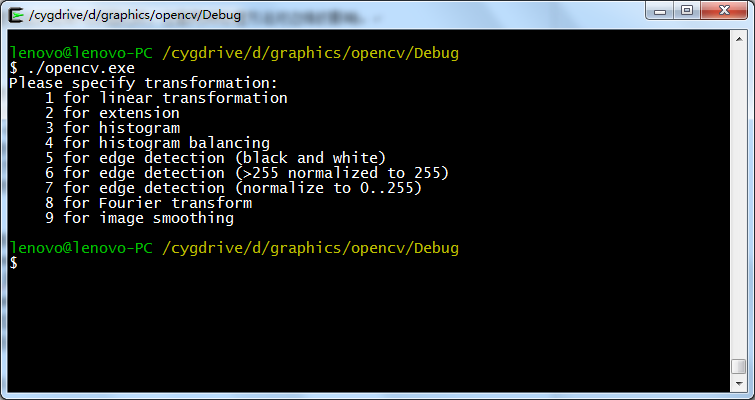
实现了5种边界检测算法，并有三种处理阈值的方法：

1. 二值化

2. 将大于255的delta值normalize成255

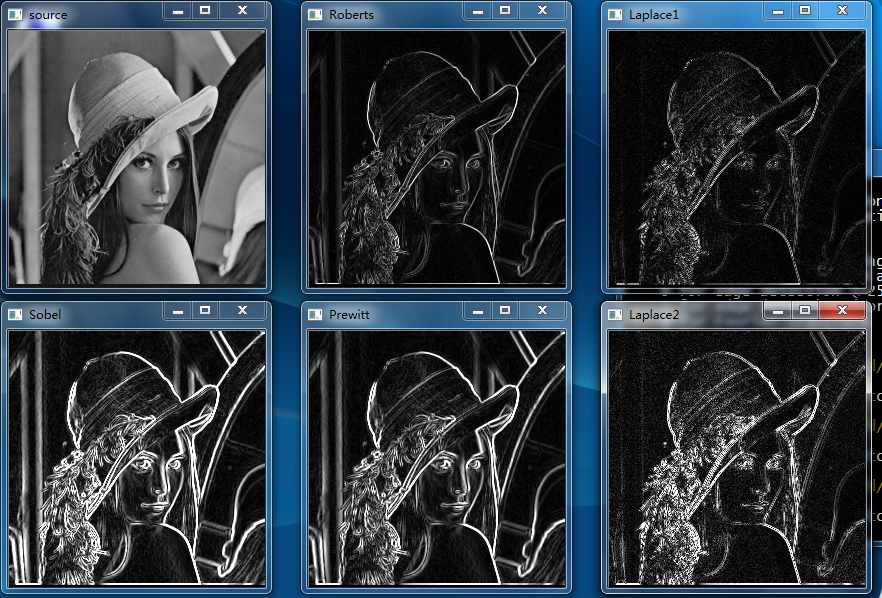
3. 将边界delta值的区间范围normalize到0..255

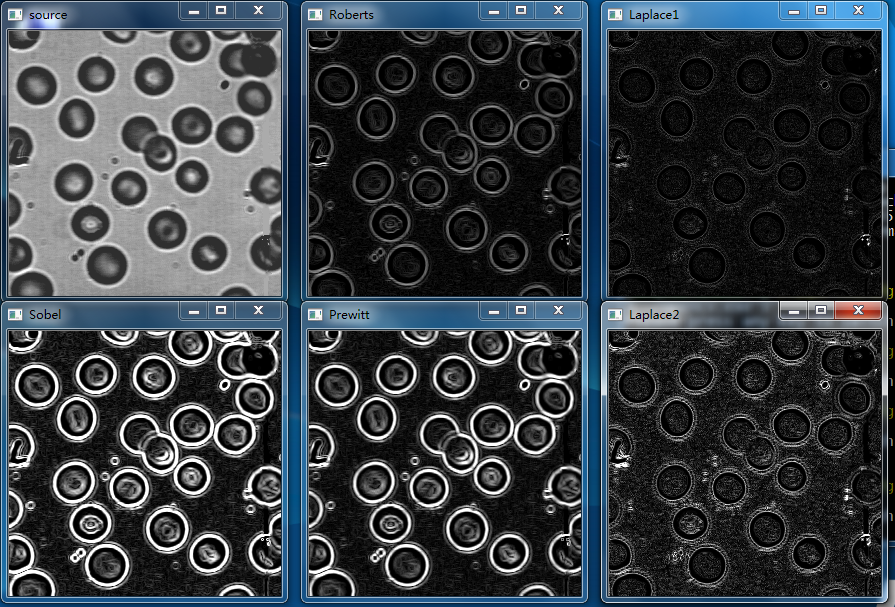
处理边界的方法与实验二相同，是将边界外的像素用最靠近的边界上像素的灰度值代替。

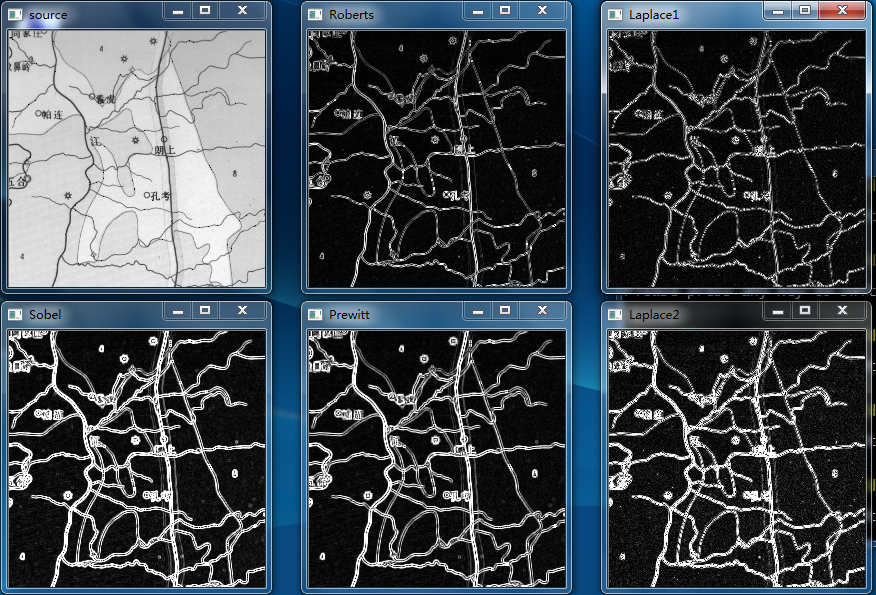


下面每组6张图中，左上角为原图，其他均为不同边缘检测算法的结果图。

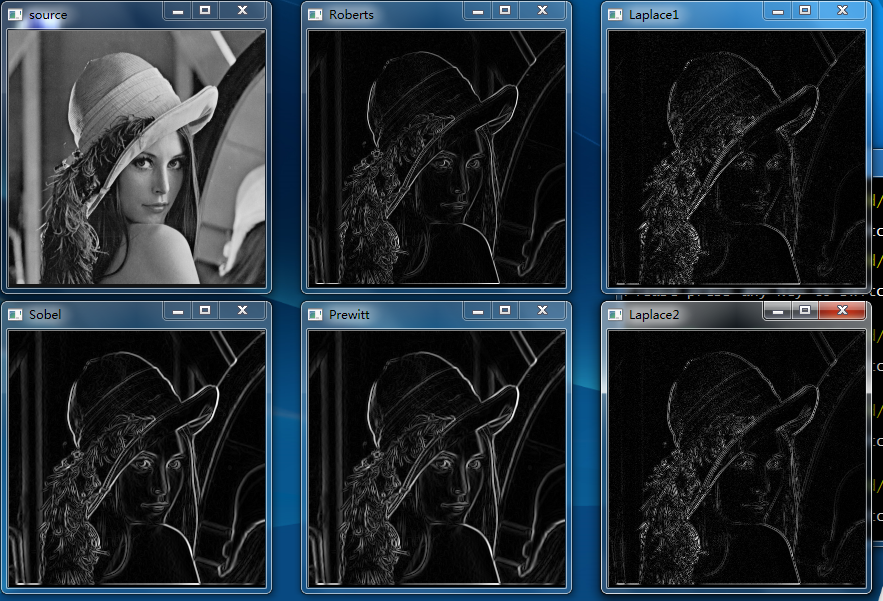
将大于255的值normalize成255：

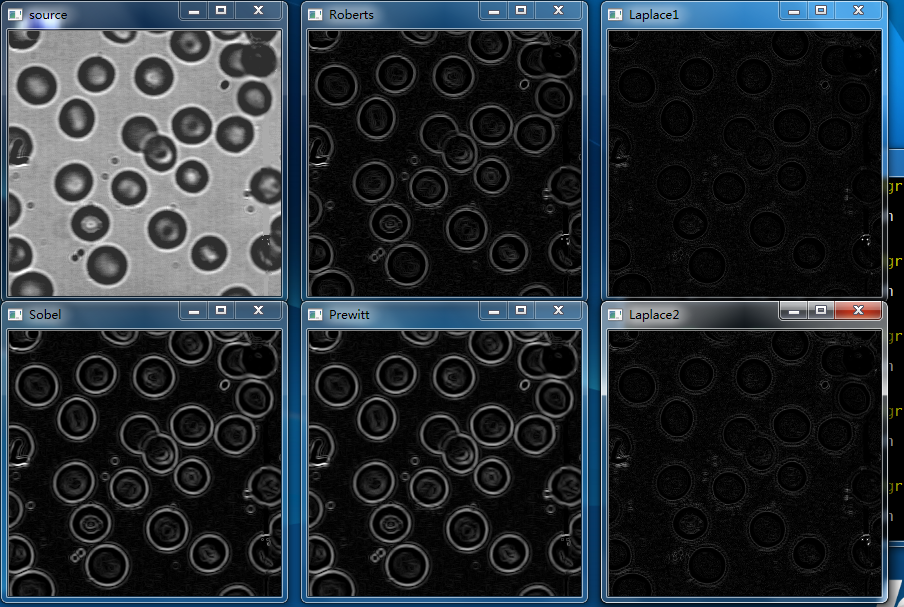


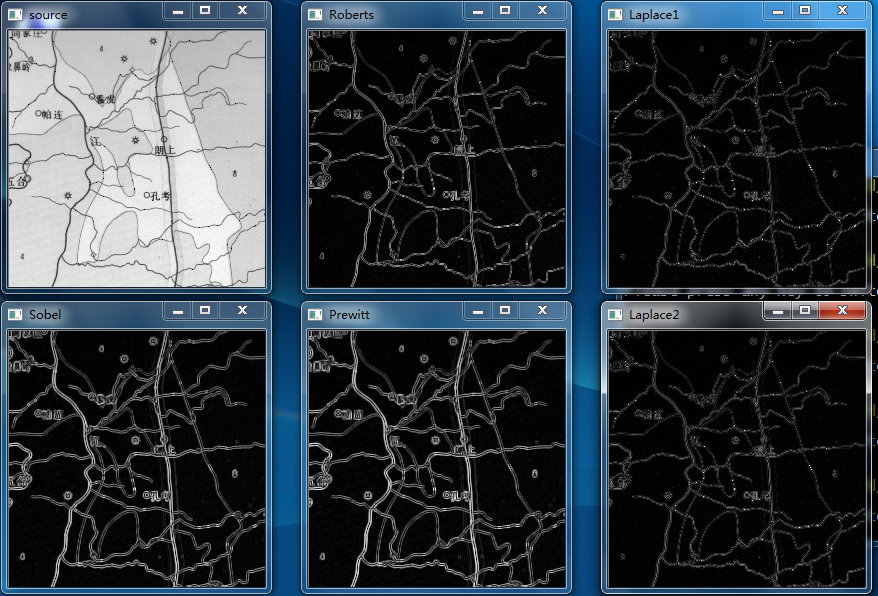




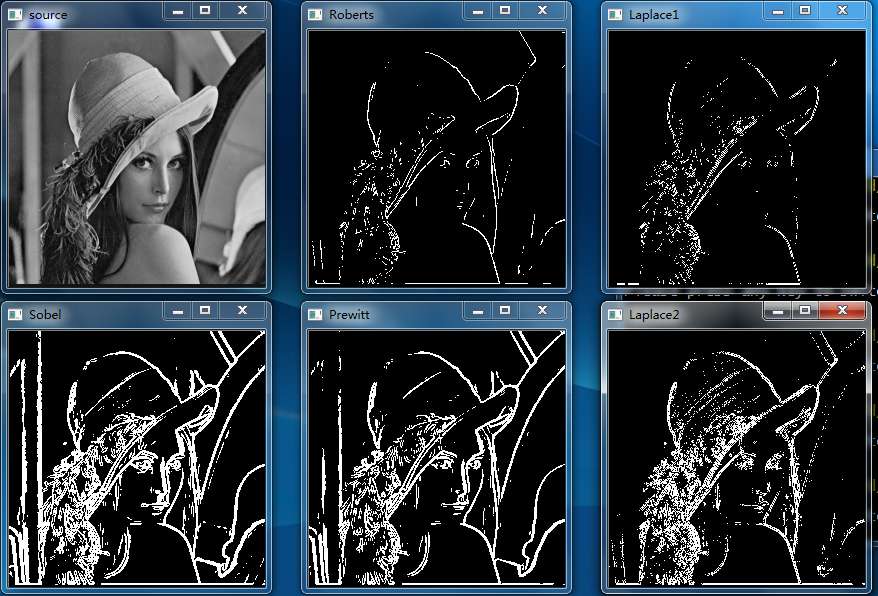
将整个delta区间normalize到0..255：

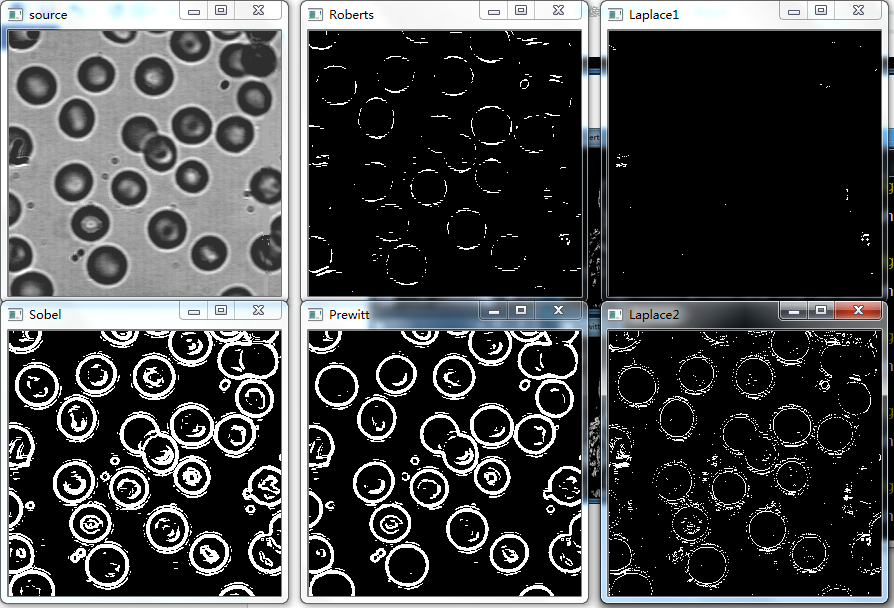


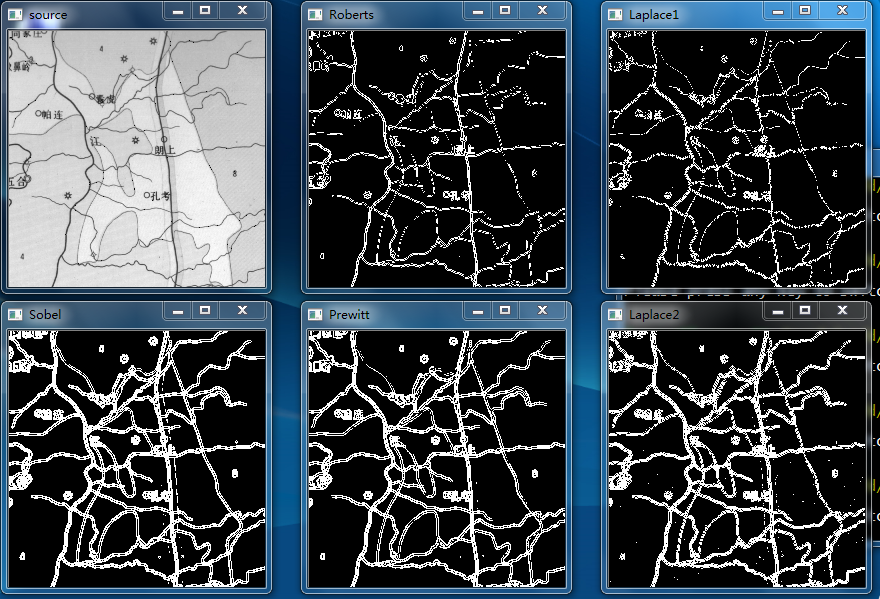




二值化方法：（阈值为100）







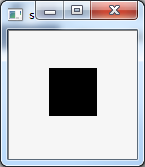
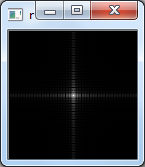
实验四

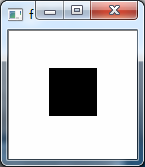
本实验没有使用OpenCV自带的Fourier变换库，是自己实现的二维DFT。由于没有使用优化算法，每行DFT需要O(n^2)的时间复杂度，对所有行变换，再对所有列变换，总的时间复杂度是O(n^3)，n为图像的边长。由于其中有很多三角运算，程序比较慢。

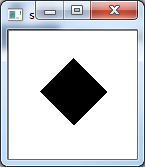
下面每组图依次是

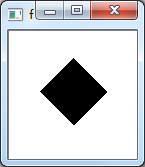
1. 原图
2. 变换后的Magnitude：低频移到中心点，log-scale
3. 变换后的Phase：低频移到中心点，normal-scale
4. 只按Magnitude反变换：log-scale
5. 按Magnitude和Phase反变换：normal-scale

如下可见，Fourier变换后的Magnitude在两个方向上重复，就像干涉条纹；变换后的Phase也是周期性的。仅按Magnitude反变换会丢失信息，而使用Magnitude和Phase两方面信息能无损地恢复原图。

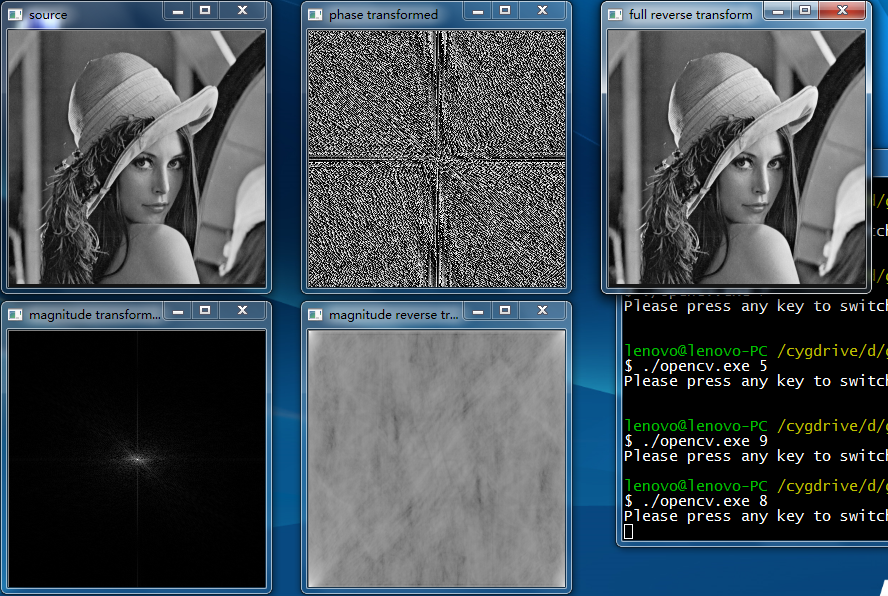
  

下面是其他一些图的变换结果（图的顺序与上面不同）



编写实验四的程序花了不少时间。

其间主要遇到两个问题：

1. 反三角函数的值域问题

C标准数学库中的atan函数值域是 (-pi/2, pi/2)，使用这个函数直接得到的幅角会丢失信息（模长和幅角相同的点有两个，在第一或第三象限，或第二或第四象限，实际只能得到第一、第四象限的点），导致恢复出来的图像形状不正确。

我的解决办法是：

x1[i] >= 0 ? atan(y1[i] / x1[i]) : 3.1415926 + atan(y1[i] / x1[i])

这样值域就是 (-pi/2, 3pi/2) 了，四个象限都可以区分开，（实部，虚部）和（模长，幅角）点对可以一一对应，此时恢复出来的图像是很精确的。

后来知道C标准数学库中有一个atan2函数，接受两个参数，因此上式可改为：

atan(y1[i], x1[i])

2. 整数损失精度问题

开始存储Fourier变换后的结果用的是整型（int），结果发现变换后的图片很模糊，原因是int损失了太多精度。换用double后就正常了。