EX2 实验报告

15331159 李沁航

实验环境：Ubuntu 16.04 LTS 虚拟机
可执行文件：canny.out
代码文件: canny.h canny.cpp main.cpp

一、实验部分
学号尾数为 9，余数为 0，改写的是 code0

1. 将图像转换为灰度图
根据公式，转化的公式为 R*0.1226+G*0.7152+B*0.0722

```
CImg<unsigned char> canny::toGrayScale(int width, int height) {
    grayscaled = CImg<unsigned char>(width, height, 1, 1, 0);


    cimg_forXY(inputImg, x, y) {
        grayscaled(x,y) = inputImg(x, y, 0) * 0.299 + inputImg(x, y, 1) * 0.587 + inputImg(x, y, 2) * 0.114;
    }
    return grayscaled;
}
```

转化之后得到的灰度图如下图：

2. 生成 Gaussian Kernel，并用其对图片进行滤波处理。
首先根据核的大小计算核中每一个数值的大小，并且使得核中的数值的核为 1.0。

```
    //calculate each value in the kernel
    for(int i = 0; i < row; i++) {
      for(int j = 0; j < column; j++) {
        filter[i][j] = (exp(-((i-row/2)*(i-row/2) + (j-column/2)*(j-column/2))/constant))/(PI * constant);
        sum += filter[i][j];
      }
    }

    // normalize to make sure the sum equal to 1
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < column; j++) {
            filter[i][j] /= sum;
          }
        }

    return filter;
 }
```

然后使用生产的核对图像进行滤波处理。

```
//Gaussian
CImg<unsigned char> canny::gaussianFilter(CImg<unsigned char> img_in, vector<vector<double> > fiilter) {
    int size = (int)fiilter.size() / 2;
    CImg<unsigned char> filteredImg = CImg<unsigned char>(img_in._width - 2 * size, img_in._height - 2 * size, 1,1,0);

    for (int i = size; i < img_in._width - size; i++) {
        for (int j = size; j < img_in._height - size; j++) {
            double sum = 0;

            for (int x = 0; x < fiilter.size(); x++)
                for (int y = 0; y < fiilter.size(); y++) {
                    sum += fiilter[x][y] * (double)(img_in(i + x - size, j + y - size));
                }

            filteredImg(i - size, j - size) = sum;
        }

    }
    return filteredImg;
}
```

通过使用刚刚得到的核，计算被一个像素点的滤波之后的数值。

在使用 size=5，sigma=5 的情况下，滤波之后得到的图像为:

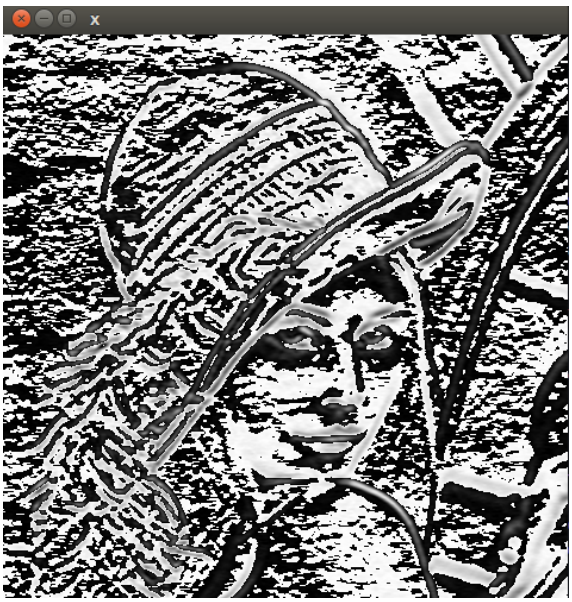
3. 利用 sobel 算子计算出图像在 x，y 方向的梯度大小，然后通过 x，y 两个方向求出的数值，计算得到在该点的方向以及量级。

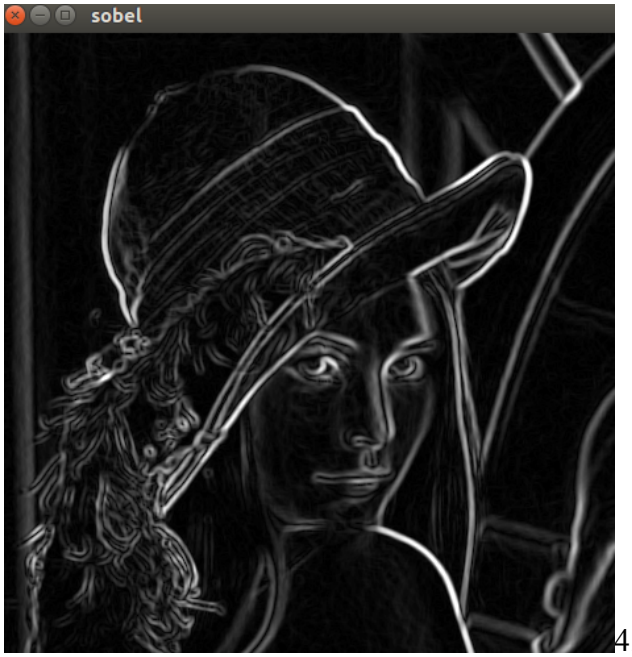
算子如下：

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

```
for (int x = 0; x < xFilter.size(); x++) {
    for (int y = 0; y < xFilter.size(); y++) {
        sumX += xFilter[x][y] * (double)(gaussianFiltered(i + x - size, j + y - size)); //Sobel_X Filter Value
        sumY += yFilter[x][y] * (double)(gaussianFiltered(i + x - size, j + y - size)); //Sobel_Y Filter Value
    }
  }
```

完成之后的结果如图所示：



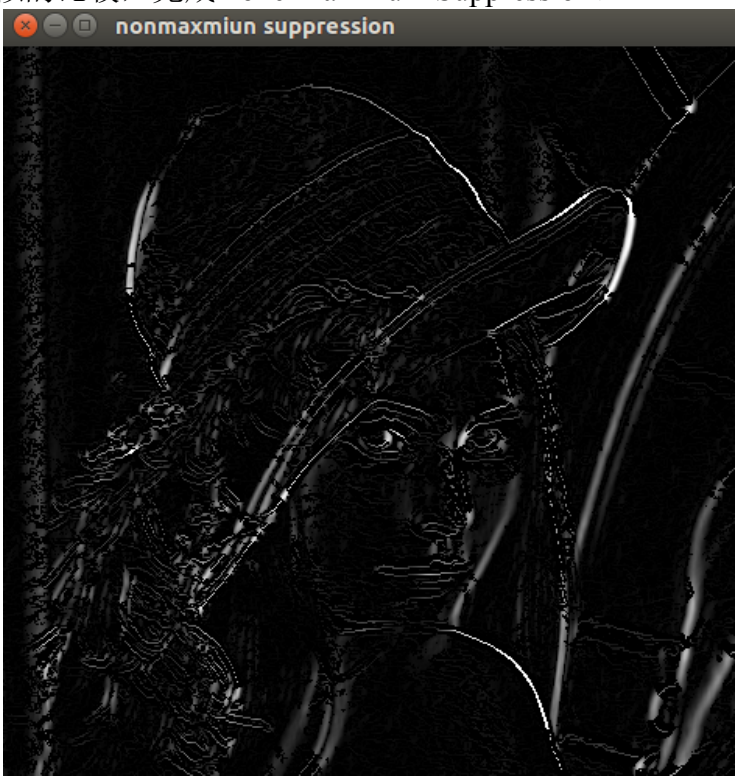


4. 进行 NoneMaximum Suppression，减小 edge 的宽度。

在各个像素点的，沿着其梯度方向，比较前后两个点的大小，若该点的值大于前后两个点的值，则保留，否则将其置 0.其中分别为上下、左右、左上右下、右上左下几个方向。

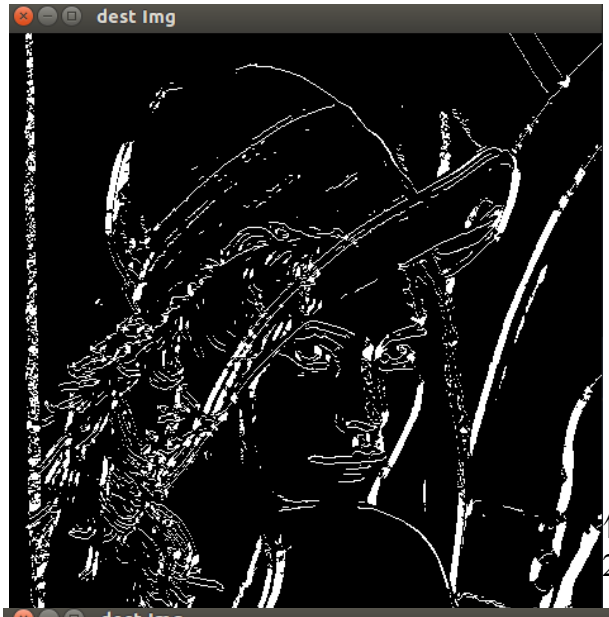
```
if (((-22.5 < Tangent) && (Tangent <= 22.5)) || ((157.5 < Tangent) && (Tangent <= -157.5))) {

    if ((sobelFiltered(i, j) < sobelFiltered(i, j + 1)) || (sobelFiltered(i, j) < sobelFiltered(i, j - 1)))
        nonMaxSupped(i - 1, j - 1) = 0;
}
```

对每一个方向进行类似的比较，完成 none Maximum Suppression。

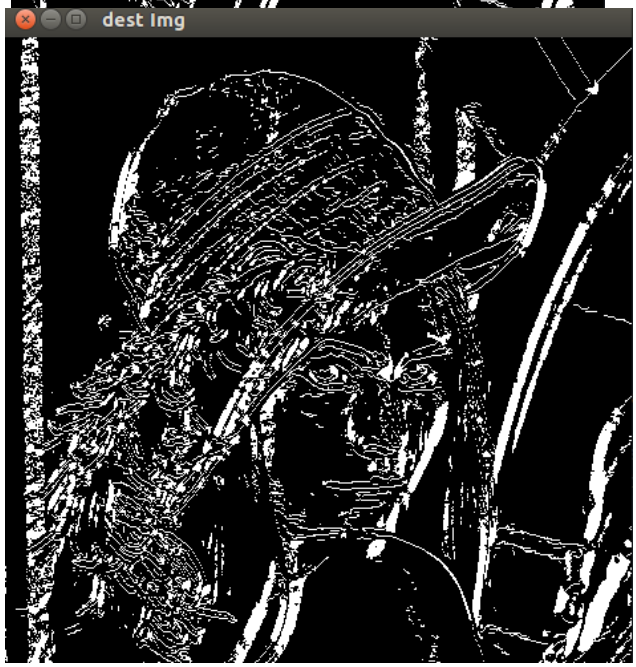


结果如图，可以发现边界确实变细了。

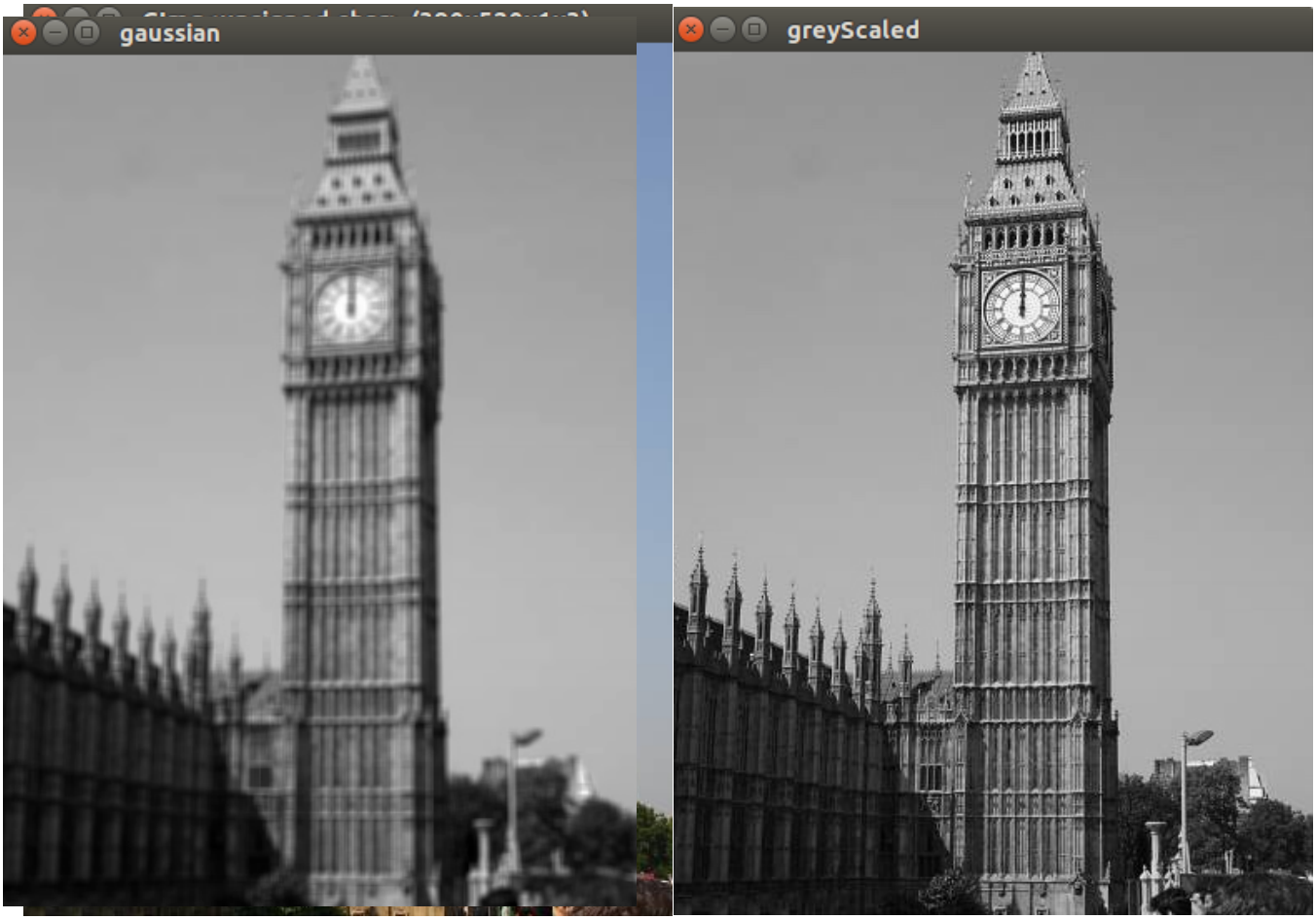
5. 使用阀值进行处理。
使用上下阀值分别为 160， 40 得到的结果如下图：



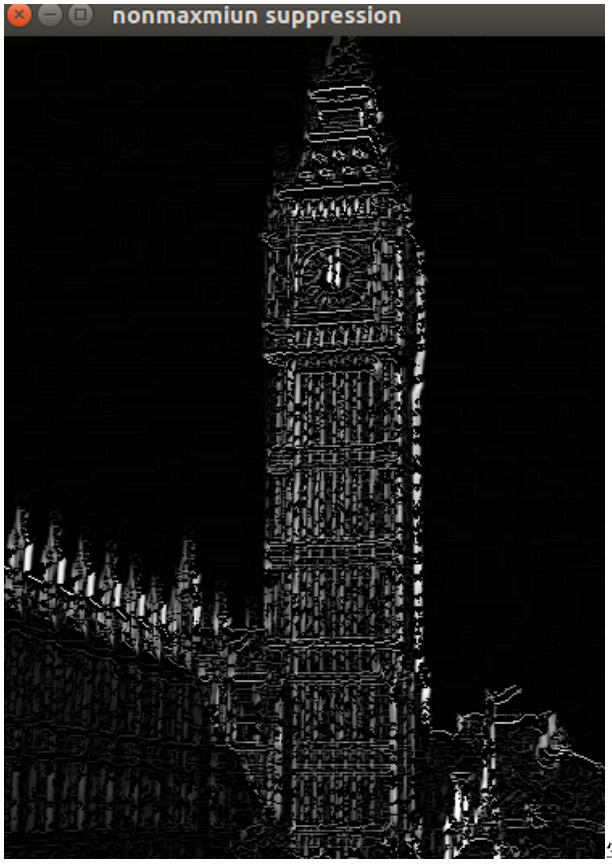
使用上下阀值分别为 50，20 得到的结果如下图。

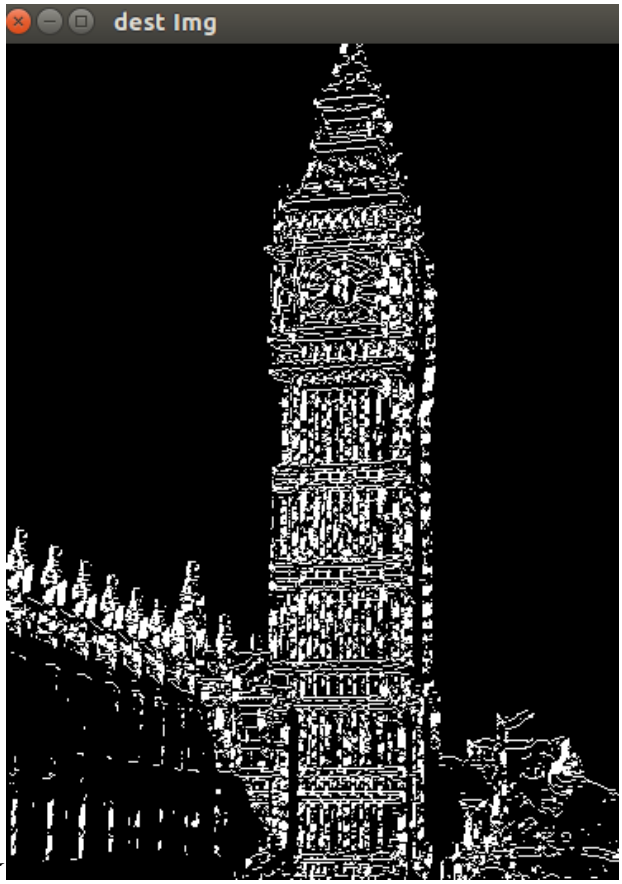


实验结果：




sobel





实验之后可以发现，设置的阀值过高会导致遗漏部分的信息，阀值过低可能会导致检测出虚假的边缘。初次之外，在滤波的时候，较大的滤波器会导致更模糊的现象，有利于尖刺较大较平滑的边缘，而较小的滤波器产生的模糊效果比较小，就有利于检测出较小，变化较明显的的边缘。