

二、实验内容和原理

1、Image mean filtering

均值滤波是典型的线性滤波算法，它是指在图像上对目标像素给一个模板，该模板包括了其周围的临近像素（以目标像素为中心的周围8个像素和目标像素本身，构成一个滤波模板），再用模板中的全体像素的平均值来代替原来像素值。类似于下面的这个矩阵，求取某个像素周围的8个像素加上自己的平均值。

$$\frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (11)$$

均值滤波可以视为一种卷积运算，这种运算比较简单，可以用如下公式来表示。

$$g(x, y) = f(x, y) * \tau(x, y) = \frac{1}{9} \cdot \sum_{i=x-1}^{i=x+1} \sum_{j=y-1}^{j=y+1} f(i, j) \quad (12)$$

均值滤波本身存在着固有的缺陷，即它不能很好地保护图像细节，在图像去噪的同时也破坏了图像的细节部分，从而使图像变得模糊，不能很好地去除噪声点。

2、Laplacian image enhancement

在很多实际运用中，我们更注意图像的边缘情况，希望强调边缘，达到增强的效果。因此引入了拉普拉斯算子作为卷积核来帮助增强边缘。

在二维中，拉普拉斯算子可以表述成如下形式。

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (13)$$

图像是离散化的数值，我们可以用差分代替微分。

对于x轴方向上的值，其差分有如下表达，我们近似认为其差分和微分数值相同。

$$\begin{aligned} \frac{\partial^2 f}{\partial x^2} &= [f(x+1, y) - f(x, y)] - [f(x, y) - f(x-1, y)] \\ &= f(x+1, y) + f(x-1, y) - 2f(x, y) \end{aligned} \quad (14)$$

同理，沿着y轴我们也可以得到如下表达式。

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \quad (15)$$

综上我们可以把拉普拉斯算子写成如下的形式，方便计算。

$$\nabla^2 f = [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 4f(x, y) \quad (16)$$

用矩阵形象地表示就如下所示，但不是矩阵运算，是一种卷积运算。

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (17)$$

值得注意的是，拉普拉斯算子是一种各向同性滤波器。若我们希望图像旋转后响应不变，要求滤波模板自身是对称的。如果不对称，结果就是当原图旋转90°时，原图某一点能检测出细节（突变）的，现在却检测不出来。即产生了各向异性。我们更偏向各向同性滤波模板，对图像的旋转不敏感。

我们再来讲一下拉普拉斯算子的根本原理。容易知道，拉普拉斯算子有两种形式：一种是中心为负值的，一种是中心为正值的，这两种元素在最后的计算模式也不一样。g代表增强后的图像像素值，f代表原来的背景图像像素值。

$$\begin{aligned} \nabla^2 f_{negative} &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\ g(x, y) &= f(x, y) - \nabla^2 f_n(x, y) \end{aligned} \quad (18)$$

$$\begin{aligned} \nabla^2 f_{positive} &= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \\ g(x, y) &= f(x, y) + \nabla^2 f_p(x, y) \end{aligned}$$

如果我们有如下一个九宫格，那么我们运用拉普拉斯负中心掩膜就会得到中心的梯度值（是一个正值）极高，在最后的运算中，这个像素点就会降低，也就是变黑。所以说，拉普拉斯负中心掩膜就是使梯度变化剧烈的地方，黑的像素值更黑。

$$\begin{bmatrix} 255 & 255 & 255 \\ 255 & 10 & 255 \\ 255 & 255 & 255 \end{bmatrix} \quad (19)$$

同理，如果我们有一个如下的九宫格，运用拉普拉斯正中心掩膜，就会在中心得到一个极大的梯度值，在最后的运算中加上梯度值，这个像素点的灰度值会增加。也就是拉普拉斯正中心掩膜就是使梯度变化剧烈的地方，白的像素值更白。

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad (20)$$

如果单独选用一种算子，那么效果只有一半。正中心掩膜增强了黑白交替的白的边，负中心掩膜增强了黑白交替的黑的边。如果单单采用某一种算法，都只会增强一边，因此我尝试是否能用两次卷积运算，既增强黑白交替的白色边，又增强黑白交替的黑的边，结果不错，在下面有具体阐述对比。

由于拉普拉斯算子没有选择性，实验中经常会出现很多噪声点，反而降低了图片质量，在实验中我有尝试不同的方法实现拉普拉斯增强，并且在讨论中提出了自己的一些想法。

三、实验步骤与分析

1、读入灰度图

```

1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<time.h>
4
5 typedef struct FILEHEADER{
6     unsigned int    bfSize;           // Size of Figure
7     unsigned short  bfReserved1;     // Reserved1, must be 00
8     unsigned short  bfReserved2;     // Reserved2, must be 00
9     unsigned int    bfOffBits;       // Offsets from header to info
10 }BMPFILEHEADER;
11
12 typedef struct INFOHEADER{
13     unsigned int    biSize;          // Size of this structure
14     unsigned int    biWidth;         // Width
15     unsigned int    biHeight;        // Height
16     unsigned short  biPlanes;       // always is 1
17     unsigned short  biBitCount;     // Kind of BMP, bit/pixel = 1 4 8
16 24 32
18     unsigned int    biCompression;
19     unsigned int    biSizeImage;
20     unsigned int    biXPelsPerMeter;
21     unsigned int    biYPelsPerMeter;
22     unsigned int    biClrUsed;
23     unsigned int    biClrImportant;
24 }BMPINFOHEADER;
25
26 typedef struct YUVSPACE{
27     unsigned char   Y;
28     unsigned char   U;
29     unsigned char   V;
30 }YUV;
31
32 int main(){
33     FILE *infp,*trfp;
34
35     unsigned short bftype;
36     // head includes head information of BMP file.
37     BMPFILEHEADER head;

```

```

38 // info includes figure information of BMP file.
39 BMPINFOHEADER info;
40
41 // read bmp
42 if((infp=fopen("RGB2YUV.bmp","rb")) == NULL){
43     printf("OPEN FAILED!\n");
44     return 0;
45 }
46
47 // bftype must be 0x4d42.
48 fread(&bftype,1,sizeof(unsigned short),infp);
49 if(bftype != 0x4d42){
50     printf("This figure is not BMP!\n");
51     Sleep(1000);
52     return 0;
53 }
54
55 // read HEADER
56 fread(&head,1,sizeof(BMPFILEHEADER),infp);
57 fread(&info,1,sizeof(BMPINFOHEADER),infp);
58
59 // We takes 24 bitcount bmp for experiment item.
60 if(info.biBitCount != 24){
61     printf("This BMP is not 24 biBitCount!\n");
62     Sleep(1000);
63     return 0;
64 }
65 /////////////////////////////////
66 // size, width, height.
67 int size = head.bfSize / 3;
68 int width = info.biWidth;
69 int height = info.biHeight;
70
71 // define array.
72 YUV yuv[height][width];
73
74 // read information.
75 fread(yuv,size,sizeof(YUV),infp);
76 fclose(infp);

```

2、保存图片

```

1 ///////////////////////////////
2 // Output BMP.
3 if((mfp = fopen("MeanFiltering.bmp","wb")) == NULL){

```

```

4     printf("Create image FAILED!\n");
5     Sleep(1000);
6     return 0;
7 }
8
9 // Write header information and data into file.
10 fwrite(&bftype,1,sizeof(unsigned short),mfp);
11 fwrite(&head,1,sizeof(BMPFILEHEADER),mfp);
12 fwrite(&info,1,sizeof(BMPINFOHEADER),mfp);
13
14 for(i=0;i<height;i++){
15     for(j=0;j<width;j++){
16         fwrite(&mf[i][j].Y,1,sizeof(unsigned char),mfp);
17         fwrite(&mf[i][j].Y,1,sizeof(unsigned char),mfp);
18         fwrite(&mf[i][j].Y,1,sizeof(unsigned char),mfp);
19     }
20 }
21
22 fclose(mfp);
23 printf("Output Image successfully!\n");
24
25 //////////////////////////////////////////////////////////////////
26 return 0;

```

3、均值滤波

```

1 //////////////////////////////////////////////////////////////////
2 // define array.
3 YUV mf[height][width];
4
5 int i,j;
6 int temp;
7
8 // Image mean filtering operation.
9 for(i=1;i<height-1;i++){
10     for(j=1;j<width-1;j++){
11         temp = 0;
12         temp += yuv[i-1][j-1].Y;
13         temp += yuv[i-1][j].Y;
14         temp += yuv[i-1][j+1].Y;
15         temp += yuv[i][j-1].Y;
16         temp += yuv[i][j].Y;
17         temp += yuv[i][j+1].Y;
18         temp += yuv[i+1][j-1].Y;
19         temp += yuv[i+1][j].Y;

```

```

20         temp += yuv[i+1][j+1].Y;
21         mf[i][j].Y = temp / 9;
22     }
23 }
24 // Fill the "holes" with original pixel value.
25 for(i=0;i<height;i++){
26     mf[i][0].Y = yuv[i][0].Y;
27     mf[i][width-1].Y = yuv[i][width-1].Y;
28 }
29
30 for(i=0;i<width;i++){
31     mf[0][i].Y = yuv[0][i].Y;
32     mf[height-1][i].Y = yuv[height-1][i].Y;
33 }

```

4、Laplacian Enhancement

```

1 //////////////////////////////////////////////////////////////////
2 // define array.
3 /*Laplacian negative center*/
4 unsigned char lrn[height][width];
5
6 /*Laplacian negative center rearrange*/
7 unsigned char lrnr[height][width];
8
9 /*Laplacian positive center*/
10 unsigned char lrp[height][width];
11
12 /*Laplacian positive center rearrange*/
13 unsigned char lrpr[height][width];
14
15 /*Laplacian change figure*/
16 unsigned char lpe[height][width];
17
18 int i,j;
19 int temp;
20
21 // Mirror operation.
22 for(i=1;i<height-1;i++){
23     for(j=1;j<width-1;j++){
24         temp = 0;
25         temp += (int)yuv[i-1][j].Y;
26         temp += (int)yuv[i][j-1].Y;
27         temp -= 4*(int)yuv[i][j].Y;
28         temp += (int)yuv[i][j+1].Y;

```

```

29     temp += (int)yuv[i+1][j].Y;
30     //temp += (int)yuv[i-1][j-1].Y;
31     //temp += (int)yuv[i-1][j+1].Y;
32     //temp += (int)yuv[i+1][j-1].Y;
33     //temp += (int)yuv[i+1][j+1].Y;
34     lrn[i][j] = (unsigned char)(cut(temp));
35     lrnr[i][j] = (unsigned char)(cut(temp*1.2));
36     lrp[i][j] = (unsigned char)(cut(-temp));
37     lrpr[i][j] = (unsigned char)(cut(-temp-3));
38 }
39 }
40 //////////////////////////////////////////////////////////////////

```

四、实验环境及运行方法

本实验运行在MACOS系统下VM Ware软件下Windows 10虚拟机的Dev C++中。源代码可编译成功。

五、实验结果展示

1、原始图片

本次实验采用的对象是Lena图的灰度图。



2、均值滤波

均值滤波的目的是使图像平滑，对像素及其周围的像素值进行平均处理。从效果上来看，图片的灰度的确更加平滑，但是相应的也更加模糊了。很多细节由于平滑的处理，被缺省了，这也是均值滤波的缺点。



3、拉普拉斯算子增强

在这一部分，我采取了三种不同的方法进行滤波，前两种就是分别利用拉普拉斯正中心掩膜、负中心掩膜对图像进行处理，最后一种我将两种方法叠加在一起，得到的效果各有不同。最后进行了图像不同成分的比较，具体实验了拉普拉斯算子的特点。

(1) 正中心掩膜



- 拉普拉斯正中心掩膜计算得到的结果

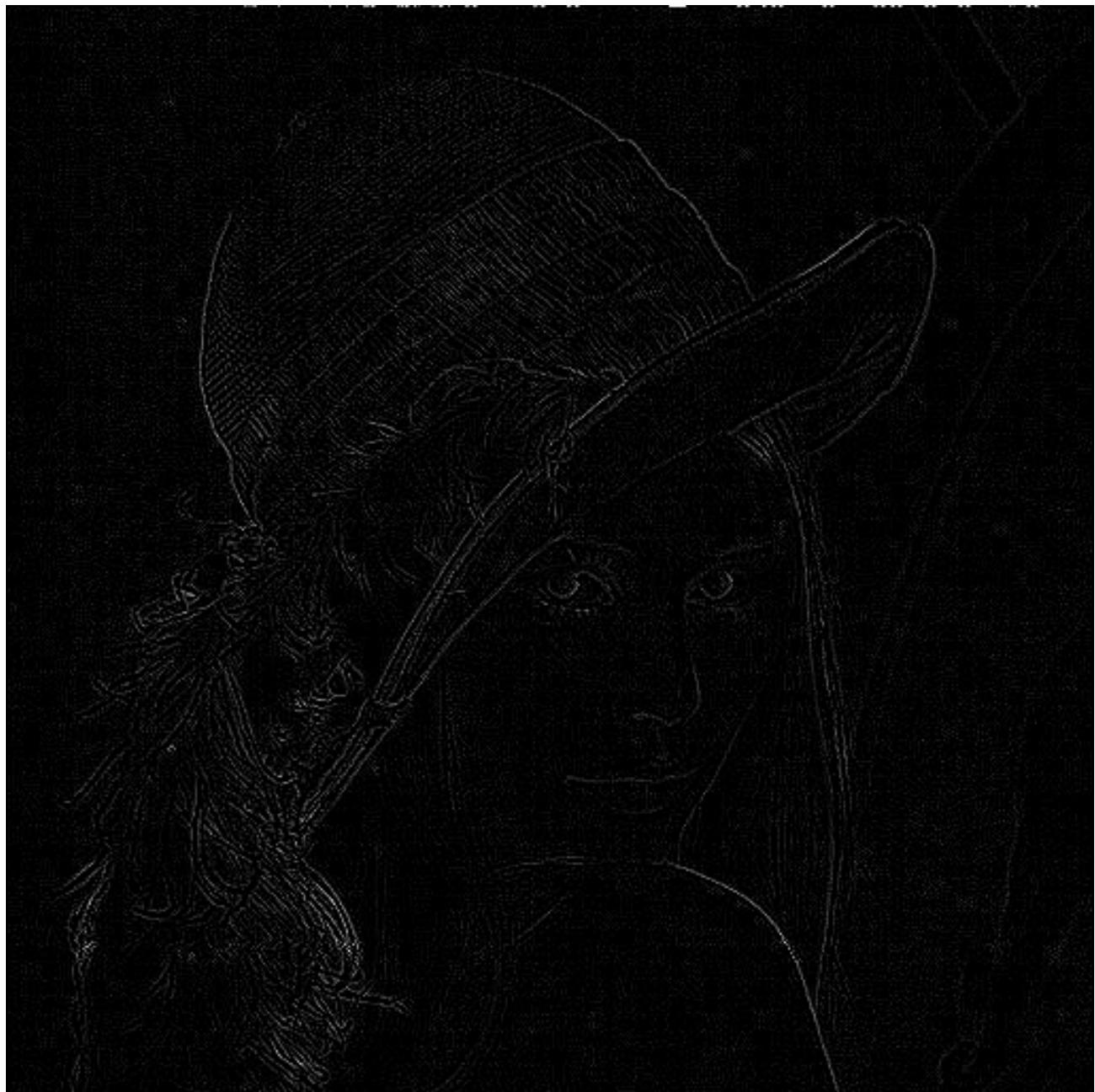


- 拉普拉斯正中心掩膜调整



- 拉普拉斯正中心掩膜增强结果
- 图像中的高亮度边界的灰度值得到了提升，产生了亮的更亮的表现。

(2) 负中心掩膜



- 拉普拉斯负中心掩膜计算得到的结果



- 拉普拉斯负中心掩膜调整



- 通过拉普拉斯负中心掩膜处理后，边界处黑色部分出现了颜色更黑的现象，增强了边界的轮廓感。但缺点就是图像中出现了很多噪声点。

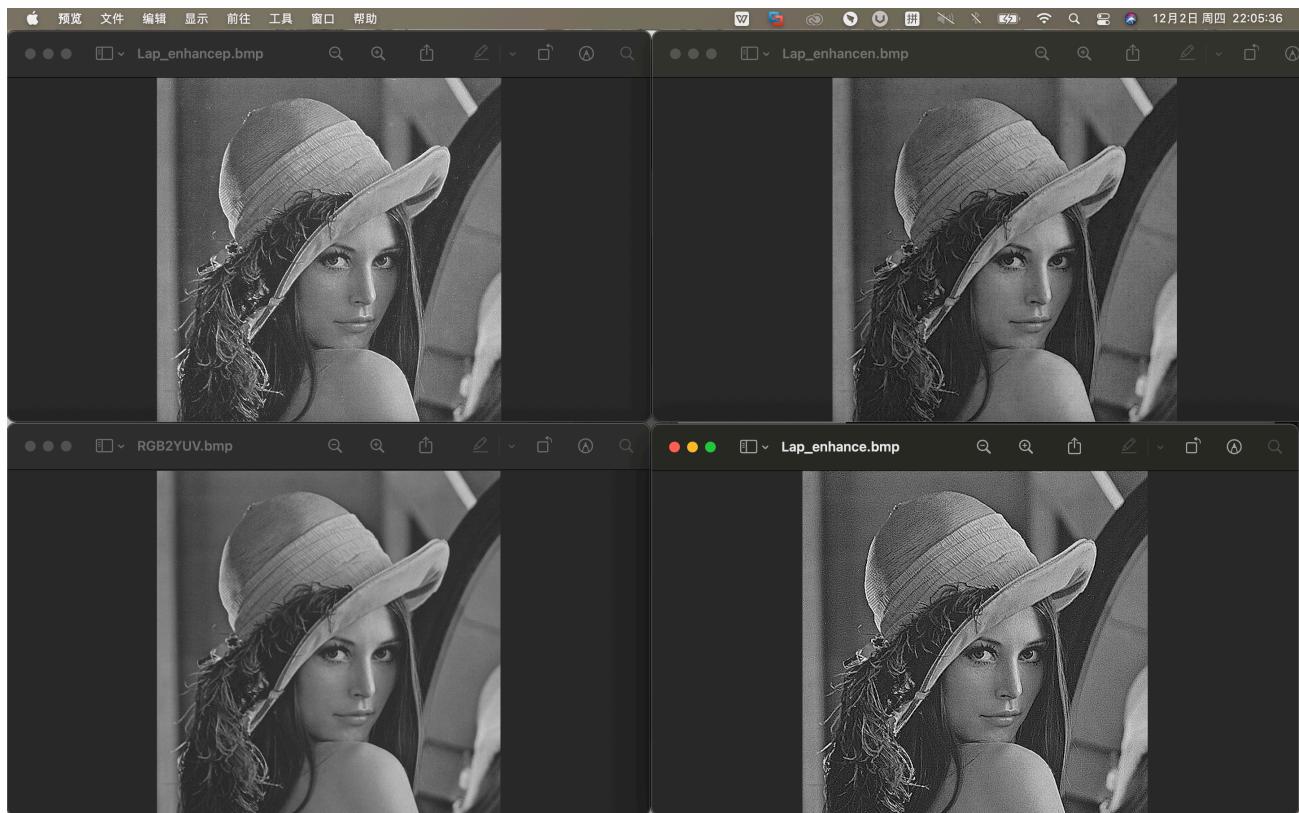
(3) 两种综合



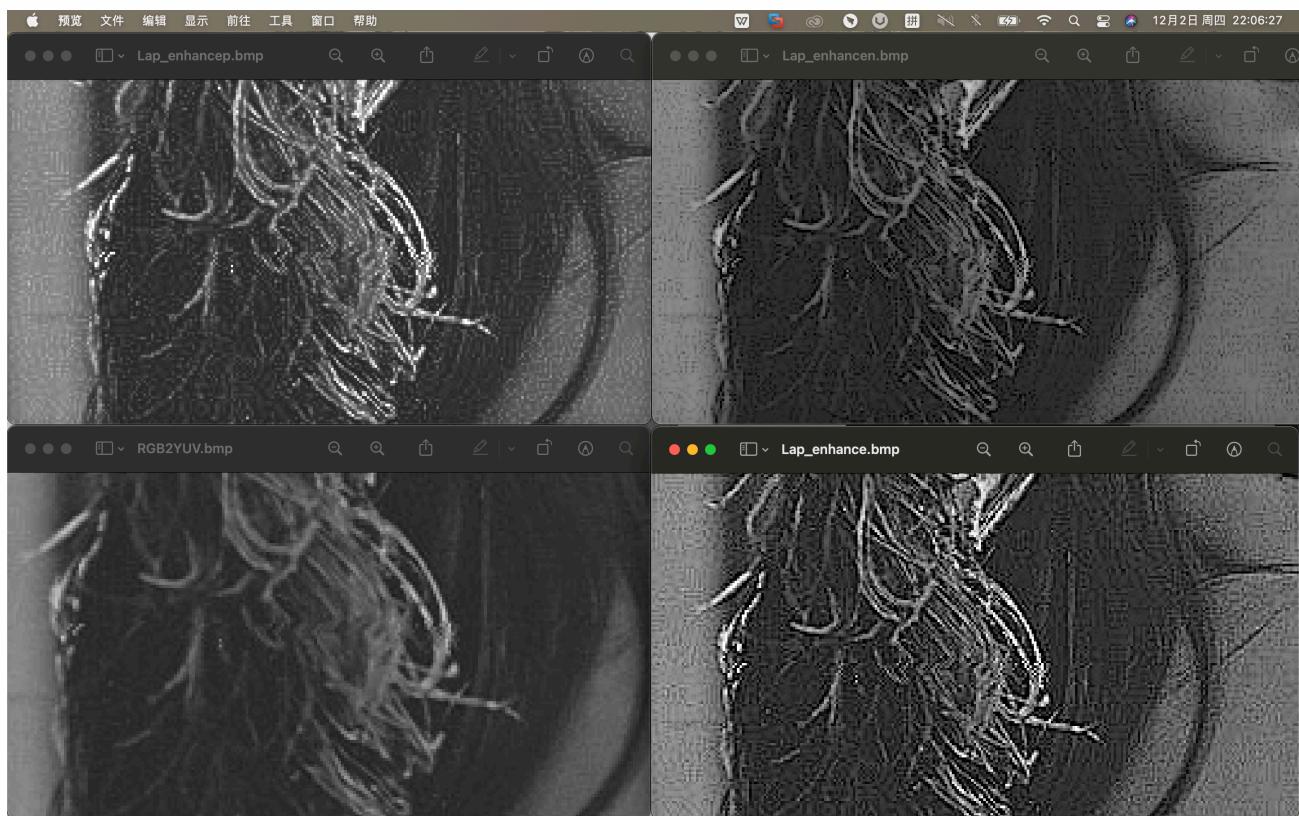
- 同时运用拉普拉斯正中心掩膜和拉普拉斯负中心掩膜可以得到上面的图片。可以观察到，这张图片的对比度以及边缘都要比上述两种算子单独作用的效果要来的明显。

(4) 比较

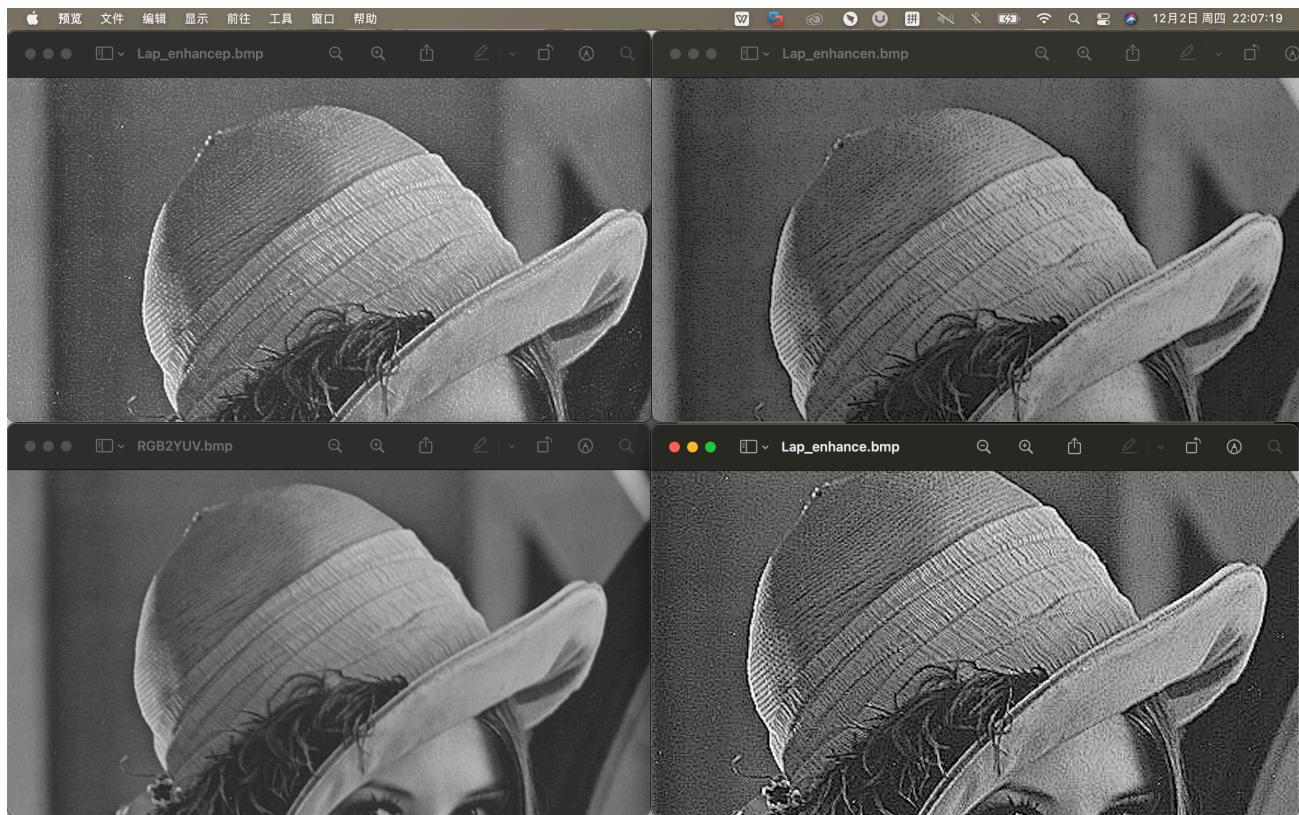
左上：正中心掩膜；右上：负中心掩膜；左下：原图；右下：两种算子叠加。



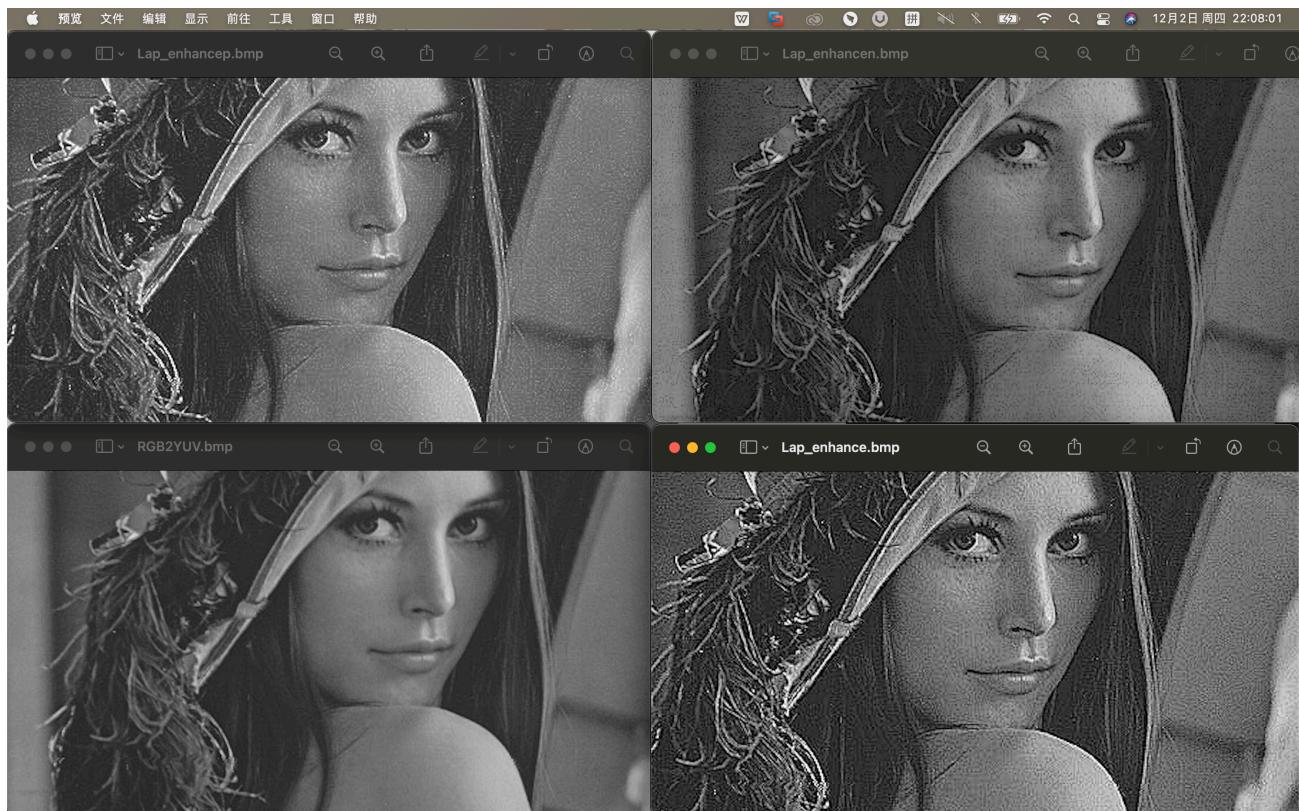
- 从整体观察，在拉普拉斯算子运算后，图片的锐度得到了大的提升。



- 观察帽饰。拉普拉斯正中心掩膜增强了图片中的高光边界，负中心掩膜加深了边界处的暗边。将两种算子叠加在一起，可以兼顾二者优点。相较原图，帽饰的细节轮廓得到了大的改善。



- 这一部分是帽子的褶皱。通过同时增加帽子褶皱边两侧的亮度和暗度，使得褶皱更加明显。对比原图，立体感增强了不少。



- 再观察肩膀的曲线。原始图片肩膀的曲线不清楚，在拉普拉斯算子增强后，肩膀的曲线变得更加

清晰。观察帽沿的曲线，在拉普拉斯增强后，帽沿的曲线得到了强化，更佳清楚明朗。双眼的清晰度上升，特别是瞳孔，黑白对比增强，人物更加有神。

六、心得体会

1、讨论

(1) 在实验中，拉普拉斯算子的确起到了增强边缘的效果。预估的效果，黑色边缘变黑，白色边缘变白也在实验结果中得到了验证。特别是两种算子叠加的方法，同时使得黑色部分更黑，白色的部分更白，兼顾两种算子的优点。操作简单，效果明显。能有效增强图像的锐度。

(2) 由于拉普拉斯算子没有选择性，对图像上所有的像素点都会进行操作。图像上某一些微弱的变化，都会由于算子而被放大，结果就是产生了很多的噪声点，反而降低了图像的质量。可以观察到图像的颗粒感十分明显。其实我还做了加入对角线元素的拉普拉速算子的增强，处理后的图像颗粒感更加明显，噪声点很多，效果不佳。这也是影响拉普拉斯算子的效果关键点。

(3) 我尝试通过减掉一些梯度值不大的像素点，即对获得的梯度值再做一次高通滤波。但是效果不佳：噪声点并非只是梯度值小的像素点，更突出的它是散布在整个画布中的，并不能很好的被去除。反而一些正常边缘位置的梯度值被削弱，导致边缘清晰程度下降。

(4) 对于出现的噪声点，我考虑能否用之前学过的图像腐蚀等操作来去除。但效果也不佳：如果图像经历了二值化，那么不同的梯度信息就会丢失，造成图片质量下降。为保证图片质量，我采用直接运用像素点与周围像素点的梯度值的大小设定阈值做腐蚀操作。但是梯度值本身分布的非常接近，如果选择较大的阈值，那么几乎没有梯度值能被保留，如果选择较小的阈值，噪声点并不能很好的被去除。而且很多有效梯度值和噪声点梯度值是相同的，很难进行滤波。

2、注意点

(1) 拉普拉斯算子是一个奇数*奇数的卷积核，因此理论上图像最外面的一圈的像素是不应该被遍历到的。在实际操作中要注意，否则容易出现Segment Fault 段错误。

(2) 对于外面一圈边可以采用填充的办法，将原始图像的像素点值填充入未计算的像素点。

3、心得

本次实验我们对图像进行了均值滤波和拉普拉斯可视增强。特别是后者，将数学梯度引入到图像处理中，呈现了一种较为简单高效的边缘计算方法。它的效果令人惊喜，但是缺点也十分明显，虽然我经过一些尝试试图消除噪点的影响，但是效果均不显著。