专业：计算机科学与技术
姓名：龙永奇
学号：3220105907
日期：2023/11/30

浙江大学实验报告

课程名称：图像信息处理 指导老师：宋明黎 成绩：

实验名称：图像均值滤波和图像拉普拉斯变换增强

一、实验目的和要求

1. 实现图像的均值滤波
2. 通过拉普拉斯变换实现图像增强（锐化）

二、实验内容和原理

- **图像滤波**

图像滤波，又称图像平滑处理。根据滤波增强目的可分为平滑滤波和锐化滤波：

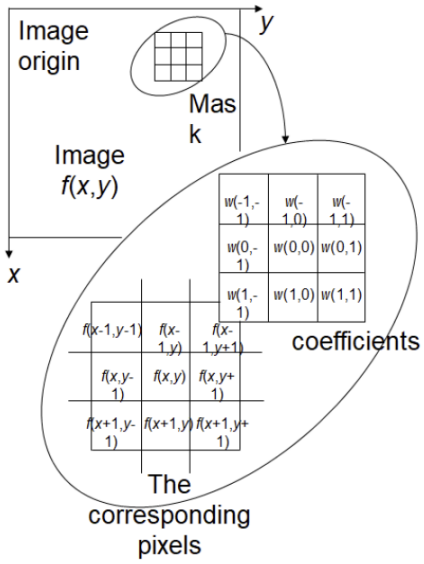
1. 平滑滤波：减弱或消除图像中的高频分量，但不影响低频分量

   由于高频分量的像素在图像中对应的附近区域灰度值变化率较快，因此平滑滤波可通过减少局部灰度的变化率，从而减弱或消除图像中的高频分量，但不影响低频分量，平滑滤波的一个重要作用就是消除噪声。

2. 锐化滤波：减弱或消除图像中的低频分量，但不影响高频分量

   由于低频分量对应的是图像中灰度值变化较慢的区域，因此和图像的整体特性有关，锐化滤波的目的就是增强图像的反差，从而增强图像中的模糊细节和景物边缘。

对于图像中的每个像素，可根据窗口中各像素值之间的关系计算滤波后的像素值，关系如下：



其中 The corresponding 表示滤波窗口的源图像像素值，coefficients 为滤波系数，卷积表示为：

$$h(x) = f(x) * h(x) = \frac{1}{M} = \sum_{t=0}^{M-1} f(t)h(x-t)$$

如果图像中中的噪点过多，可通过滤波进行平滑操作，但会使图像模糊，本次实验中我们使用线性滤波。

- **线性滤波**

线性滤波通过计算遮罩中的像素平均值，从而去除比遮罩窗口还小的不需要的区域

$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

在均值滤波中，每个遮罩的系数等于所有遮罩窗口内系数之和取倒数，数学表达式为：

$$g(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)f(x+s,y+t)}{\sum_{s=-a}^{a}\sum_{t=-b}^{b} w(s,t)}$$

其中遮罩窗口大小需要为奇数，从而确定一个中心点，定义为

$$(2a+1)\times(2b+1)$$

可以看见遮罩窗口大小对最终的变换结果影响较大，若遮罩过小，则效果微弱，若太大则太模糊，因此需要根据背景对象找到合适的遮罩大小。

- **通过 Laplace 变换对图像进行增强（锐化）**

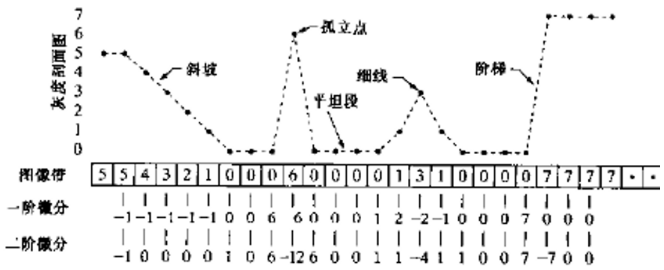
锐化与上述滤波的效果相反，由于模糊是通过积分（卷积）实现，因此可通过微分实现锐化，通过微分算子这一锐化工具，基于相邻像素值之间的变化率，其效果和作用区域的变化率成正比，从而增强图像中的边缘和噪点，弱化灰度变化率低的区域。

微分算子差分表示：

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

二阶微分算子：

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



对于函数 f (x, y)，定义向量：

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

幅度为：

$$\nabla f = (G_x^2 + G_y^2)^{\frac{1}{2}} = \left[(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2 \right]^{\frac{1}{2}}$$

由于梯度的计算时间较长，可使用绝对值代替梯度计算，做一种近似：

$$\nabla f \approx |G_x| + |G_y|$$

可以得出其 Laplace 算子：

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

根据之前的定义，我们在 x、y 方向有以下关系：

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

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

从而得到离散形式的 Laplace 函数

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

可以额外考虑对角线方向的元素：

$$\begin{aligned} \nabla^2 f = & [f(x-1, y-1) + f(x, y-1) + f(x+1, y-1) \\ & + f(x-1, y) + f(x+1, y) \\ & + f(x-1, y+1) + f(x, y+1) + f(x+1, y+1)] \\ & - 8f(x, y) \end{aligned}$$

简化为：

$$\nabla^2 f = \sum_{i=-1}^{1}\sum_{j=-1}^{1} f(x+i,y+j) - 9f(x,y)$$

Laplace 算子的遮罩窗口如下：

0	1	0	1	1	1
1	−4	1	1	−8	1
0	1	0	1	1	1
0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1

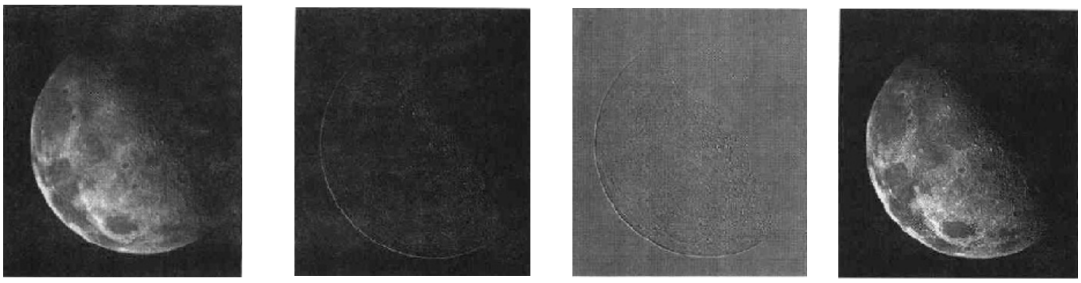
将拉普拉斯结果和原始图像融合在一起时，必须考虑它们之间的符号差异。如果使用的定义中心系数为负数，则需要减去 Laplace 图像从而获得锐化效果， 其基本方法为：

$$g(x,y) = f(x,y) + c\left[\nabla^2 f(x,y)\right]$$

根据课程中的介绍，f 和 g 分别为原图和锐化图：

$$g(x,y) = \begin{cases} f(x,y) - \nabla^2 f(x,y), If\ the\ center\ of\ the\ mask\ is\ negative \\ f(x,y) + \nabla^2 f(x,y), If\ the\ center\ of\ the\ mask\ is\ positive \end{cases}$$

将两者融合就能保持锐化，从而恢复掩盖掉的细节信息，以下是变换效果：



从左至右依次为原图、Laplacian 变换后、Rearranged Laplacian、合并结果。

三、实验步骤与分析

- **图像的读取**

本次实验使用变量 Org 储存原图像，Ans 储存变换后的图像，分别将两图像传入变换函数中，在函数内进行图像的信息复制以及输出，读取函数如下：

```
FILE *fp;
FILESTRUCT Org, Ans; // 原图像 Org，变换后的图像 Ans
int BMPSize, BMPheight, BMPwidth;
fp = fopen("zjz_Color.bmp", "rb"); // 读取本目录下的 bmp 文件
if (!fp) printf("憨憨丢掉了:(\n");
else printf("憨憨在这里:)\n");
fread(&(Org.FH), sizeof(FILE_HEAD), 1, fp);
fread(&(Org.IH), sizeof(INF_HEAD), 1, fp);
if (!Org.IH.biSizeImage) Org.IH.biSizeImage = Org.FH.bfSize - Org.FH.bfOffBits;
BMPheight = Org.IH.biHeight;
BMPwidth = Org.IH.biWidth;
BMPSize = Org.IH.biSizeImage;
if(Org.IH.biBitCount == 8){
    for(int i = 0; i < 256; i++){
        Org.Color[i].rgbBlue = Org.Color[i].rgbGreen = Org.Color[i].rgbRed = i;
        Ans.Color[i].rgbBlue = Ans.Color[i].rgbGreen = Ans.Color[i].rgbRed = i;
    }
}
fseek(fp, Org.FH.bfOffBits, SEEK_SET);
Org.pix_val = (unsigned char *)malloc(sizeof(unsigned char) * BMPSize);
fread(Org.pix_val, BMPSize * sizeof(unsigned char), 1, fp);
fclose(fp);
```

- **均值滤波**

这里我们在窗口超过图像边界时采取了临近像素代替的思想，如果坐标为负数或者超过了图像的长或宽，就限制在图像以内，防止越界，具体实现如下，其中 scale 为窗口大小：

```
    void Mean_Filtering(FILESTRUCT *Org, FILESTRUCT *Ans, int scale){
    memcpy(&(Ans->FH), &(Org->FH), sizeof(FILE_HEAD)); // 图像信息的复制
    memcpy(&(Ans->IH), &(Org->IH), sizeof(INF_HEAD));
    Ans->pix_val = (unsigned char *)malloc(sizeof(unsigned char) *
      Org->IH.biSizeImage);
    int now_position, i, j, p, q;
    int edge = scale / 2; // 窗口大小
    int row_offset = (Org->IH.biBitCount / 8 * Org->IH.biWidth + 3) / 4 * 4;
    for(i = 0; i < Org->IH.biHeight; i++){
        for(j = 0; j < Org->IH.biWidth; j++){
            if(Ans->IH.biBitCount == 24){ // 24 位彩色图
                int sum_R = 0, sum_G = 0, sum_B = 0;
                for(p = i - edge; p <= i + edge; p++){
                    for(q = j - edge; q <= j + edge; q++){
                        now_position = Position(p, q, row_offset, 3,
      Org->IH.biHeight, Org->IH.biWidth);
```

```
                    sum_B += Org->pix_val[now_position];
                    sum_G += Org->pix_val[now_position + 1];
                    sum_R += Org->pix_val[now_position + 2];
                }
            }
            now_position = i * row_offset + j * 3;
            Ans->pix_val[now_position] = sum_B / (scale * scale);
            Ans->pix_val[now_position + 1] = sum_G / (scale * scale);
            Ans->pix_val[now_position + 2] = sum_R / (scale * scale);
        }else{ // 8 位灰度图
            int sum = 0;
            for(p = i - edge; p <= i + edge; p++){
                for(q = j - edge; q <= j + edge; q++){
                    now_position = Position(p, q, row_offset, 1,
                        Org->IH.biHeight, Org->IH.biWidth);
                    sum += Org->pix_val[now_position];
                }
            }
            now_position = i * row_offset + j;
            Ans->pix_val[now_position] = sum / (scale * scale);
        }

    }
  }
  Output(Ans, "Mean_Filtering.bmp");
}
```

- Laplace 变换 1

变换矩阵为：

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

值得注意的是，两种 Laplace 变换都需要将像素值限制在 0 - 255 以内，否则有可能越界，代码如下：

```
void Lapacian_Transform_1(FILESTRUCT *Org, FILESTRUCT *Ans){
    // 0  1  0
    // 1 -4  1
    // 0  1  0
    memcpy(&(Ans->FH), &(Org->FH), sizeof(FILE_HEAD));
    memcpy(&(Ans->IH), &(Org->IH), sizeof(INF_HEAD));
```

```
    Ans->pix_val = (unsigned char *)malloc(sizeof(unsigned char) *
Org->IH.biSizeImage);
    int now_position, i, j, p;
    int ary_x[5] = {-1, 0, 0, 0, 1}; // 遍历五个位置，使用大小为 5 的数组，同实验 2
的方法
    int ary_y[5] = {0, -1, 0, -1, 0}; // y 方向
    int row_offset = (Org->IH.biBitCount / 8 * Org->IH.biWidth + 3) / 4 * 4;
    for(i = 0; i < Org->IH.biHeight; i++){
        for(j = 0; j < Org->IH.biWidth; j++){
            if(Org->IH.biBitCount == 24){ // 24 位彩色图
                int sum_B = 0, sum_R = 0, sum_G = 0;
                for(p = 0; p <= 4; p++){
                    now_position = Position(i + ary_x[p], j + ary_y[p],
row_offset, 3, Org->IH.biHeight, Org->IH.biWidth);
                    sum_B += Org->pix_val[now_position];
                    sum_G += Org->pix_val[now_position + 1];
                    sum_R += Org->pix_val[now_position + 2];
                }
                now_position = i * row_offset + j * 3;
                Ans->pix_val[now_position] = Restrict(6 *
Org->pix_val[now_position] - sum_B);
                Ans->pix_val[now_position + 1] = Restrict(6 *
Org->pix_val[now_position + 1] - sum_G);
                Ans->pix_val[now_position + 2] = Restrict(6 *
Org->pix_val[now_position + 2] - sum_R);
            }else{ // 8 位灰度图
                int sum = 0;
                for(p = 0; p <= 4; p++){
                    now_position = Position(i + ary_x[p], j + ary_y[p],
row_offset, 3, Org->IH.biHeight, Org->IH.biWidth);
                    sum += Org->pix_val[now_position];
                }
                now_position = i * row_offset + j;
                Ans->pix_val[now_position] = Restrict(6 *
Org->pix_val[now_position] - sum);
            }
        }
    }
    Output(Ans, "Lapacian_Transform_1.bmp");
}
```

- Laplace 变换 2

  变换矩阵为：

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

代码实现和第一种类似：

```
void Lapacian_Transform_2(FILESTRUCT *Org, FILESTRUCT *Ans){
    // 1  1  1
    // 1 -8  1
    // 1  1  1
    memcpy(&(Ans->FH), &(Org->FH), sizeof(FILE_HEAD));
    memcpy(&(Ans->IH), &(Org->IH), sizeof(INF_HEAD));
    Ans->pix_val = (unsigned char *)malloc(sizeof(unsigned char) *
Org->IH.biSizeImage);
    int now_position, i, j, p, q;
    int row_offset = (Org->IH.biBitCount / 8 * Org->IH.biWidth + 3) / 4 * 4;
    for(i = 0; i < Org->IH.biHeight; i++){
        for(j = 0; j < Org->IH.biWidth; j++){
            if(Org->IH.biBitCount == 24){ // 24 位彩色图
                int sum_B = 0, sum_R = 0, sum_G = 0;
                for(p = i - 1; p <= i + 1; p++){
                    for(q = j - 1; q <= j + 1; q++){
                        now_position = Position(p, q, row_offset, 3,
Org->IH.biHeight, Org->IH.biWidth);
                        sum_B += Org->pix_val[now_position];
                        sum_G += Org->pix_val[now_position + 1];
                        sum_R += Org->pix_val[now_position + 2];
                    }
                }
                now_position = i * row_offset + j * 3;
                Ans->pix_val[now_position] = Restrict(10 *
Org->pix_val[now_position] - sum_B);
                Ans->pix_val[now_position + 1] = Restrict(10 *
Org->pix_val[now_position + 1] - sum_G);
                Ans->pix_val[now_position + 2] = Restrict(10 *
Org->pix_val[now_position + 2] - sum_R);
            }else{ // 8 位灰度图
                int sum = 0;
                for(p = i - 1; p <= i + 1; p++){
                    for(q = j - 1; q <= j + 1; q++){
                        now_position = Position(p, q, row_offset, 1,
Org->IH.biHeight, Org->IH.biWidth);
                        sum += Org->pix_val[now_position];
                    }
```

```
                }
                now_position = i * row_offset + j;
                Ans->pix_val[now_position] = Restrict(10 *
Org->pix_val[now_position] - sum);
            }
        }
    }
    Output(Ans, "Laplacian_Transform_2.bmp");
}
```

四、实验环境及运行方法

1. 实验环境

系统 Windows11 编译器 gcc 10.3.0x86_mingw32

2. 运行方法

在文件夹中，Lab5.c 为源文件，运行代码，如果出现“憨憨在这里:)”说明读取图片文件成功，否则会出现“憨憨丢掉了: (”.

随后根据读取的图象是彩色或者灰度程序会分别输出：

- 均值滤波图像：Mean_Filtering.bmp
- 第一种 Laplace 变换图像：Laplacian_Transform_1.bmp
- 第二种 Laplace 变换图像：Laplacian_Transform_2.bmp

五、实验结果展示

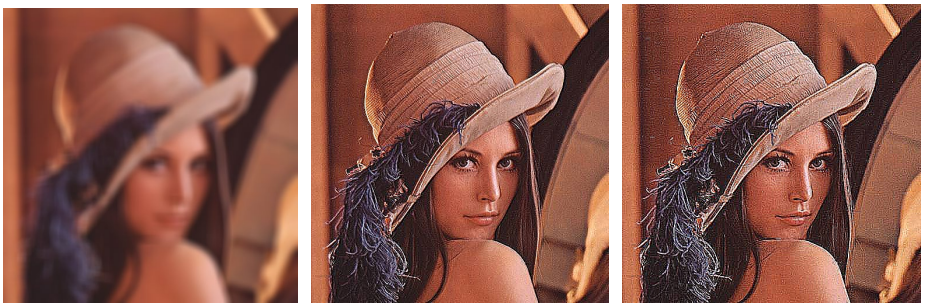
1. 原图像

2. 均值滤波 Scale = 17

3. Laplace 1

4. Laplace 2

➢ 测试用例：莱纳



六、心得体会

本次实验涉及到图像的均值滤波和锐化，其中均值滤波即通过简单的窗口取平均值实现，

锐化则是通过两种 Laplace 变换实现。在未经缩放的情况下，直接应用拉普拉斯变换会生成像素点密集的输出结果，因为本程序使用 unsigned char 来表示像素值，在这种数据类型中，负值就是加 255 后的正值。对于灰度图，直接输出结果可能导致为白色 255，正值为黑色 0。因此需要缩放操作，进行对像素值进行限制，获得适当的拉普拉斯变换后的输出。

总体来说，本次实验的难度不大，但是内容相当有趣，简单的代码实现了之前 P 图中常用的操作。