**实验报告**

专业：计算机科学与技术

姓名：张三

学号：111111

日期：2022/11/21

课程名称： 图像信息处理 指导老师： 成绩：

实验名称： 均值滤波和拉普拉斯滤波

**一、实验目的和要求**

目的：  
对于bmp图像，运用均值滤波算法实现平滑，并且运用拉普拉斯滤波实现锐化。

要求：

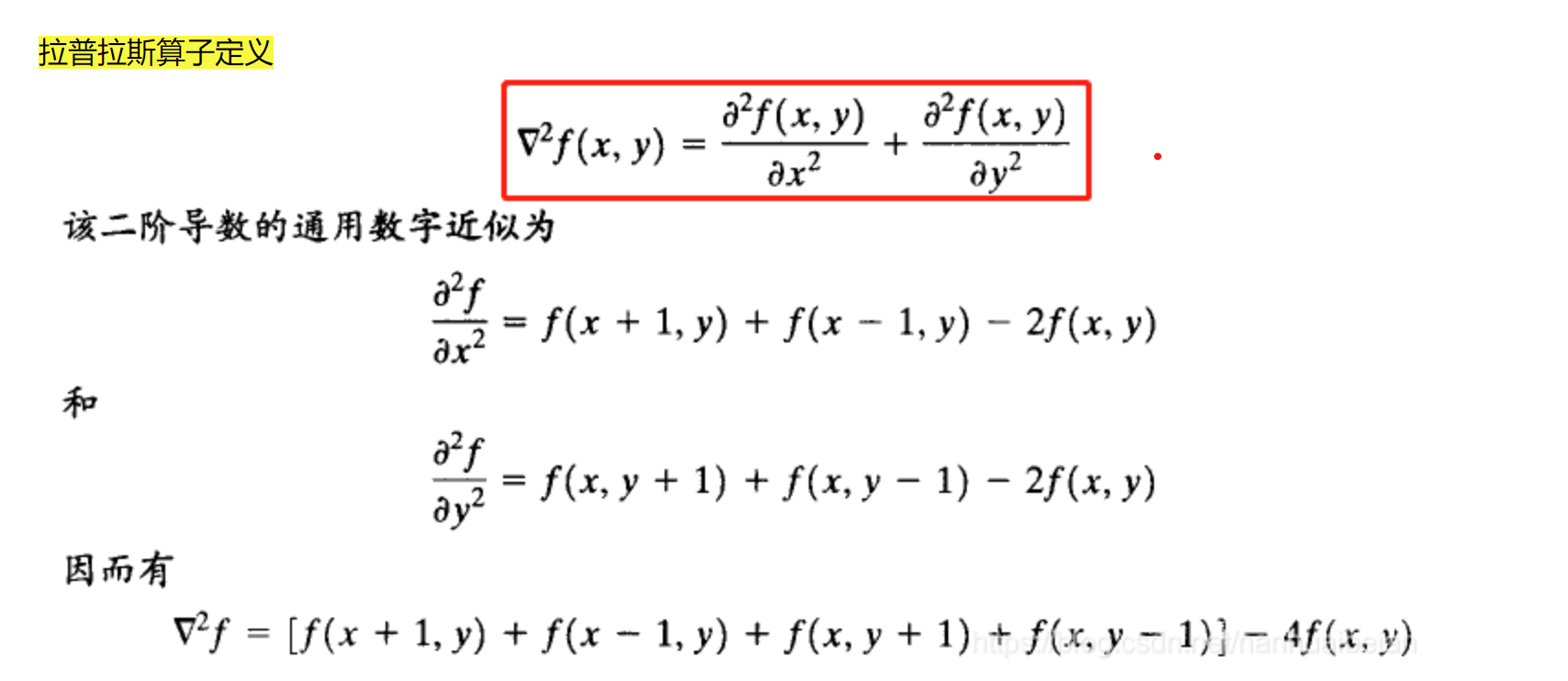
对于bmp图像，用C或者C++源代码实现，不允许调用相关的库函数。

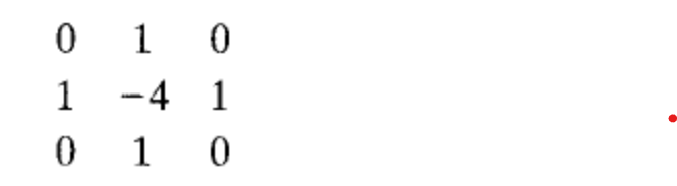
**二、实验内容和原理**

所谓的滤波，或者说是卷积哪一类的操作，实际上就是加权求和。

对于均值滤波，一般可以用于去除图片的噪点，均值滤波是对于一个3\*3或者5\*5的卷积核所有点的权重都一样，都设为一，相加后求平均得到的值作为中心点的新像素值。在本实验中使用的是5\*5的均值卷积核。

拉普拉斯滤波实际上是在求取离散的图像点之间的二阶导。



用卷积核表示出来就是：  


用输入图像的的每个像素点减去（如果卷积核的中心是正4则改为加上）拉普拉斯算子得到的结果就能得到加强后的图像。

**三、实验步骤与分析**

#include <stdio.h>

#include <stdlib.h>

#include <windows.h>

#include <math.h>

typedef struct BMP{

    FILE\* file;

    BITMAPFILEHEADER bf;

    BITMAPINFOHEADER bi;

} BMP;

int write\_gray\_data(BMP bmp,double\* graydata);//write the gray data into graybmp, because we use the palette, we only need to write the y component

int write\_rgbdata(BMP,unsigned char\*);

int write\_data(BMP bmp,unsigned char\* black\_white);//write the prepared bmp data into file

int initial\_bmp(BMP);//write the file header into the file

BMP create\_bmp();//read the first bmp in D://test.bmp

int fill\_rgb\_yuv\_array(BMP,unsigned char\*,double\*);//read bmp data from the 24color file and transfer it into gray number array

int mean(double \*yuv,BMP);//calculate the pixels after mean operation ,which will be stored in where former u component is stored

int lap(double \*yuv,BMP);//calculate the pixels after laplace opeartion ,which will be stored in where former u component id stored

int main(){

    BMP rawbmp = create\_bmp();

    unsigned char \*rgbdata= malloc (sizeof(unsigned char)\*3\*rawbmp.bi.biWidth\*rawbmp.bi.biHeight);// to store the gray number for gray bmp

    double \*yuv= malloc (sizeof(double)\*3\*rawbmp.bi.biWidth\*rawbmp.bi.biHeight);

    fill\_rgb\_yuv\_array(rawbmp,rgbdata,yuv);//fill in the array

    RGBQUAD rgbquad[256];//create the palette

    for(int i=0; i<256; i++){

        rgbquad[i].rgbBlue = i;

        rgbquad[i].rgbGreen = i;

        rgbquad[i].rgbRed = i;

        rgbquad[i].rgbReserved = 0 ;

    }

    FILE \* grayfile =fopen("D://gray.bmp","wb");//create the gray bmpso that  we can compare the origin picture with the processed picture

    BMP graybmp = {grayfile,rawbmp.bf,rawbmp.bi};

    graybmp.bf.bfOffBits += 256 \*sizeof(RGBQUAD);

    graybmp.bf.bfSize = graybmp.bf.bfOffBits + (graybmp.bi.biWidth+3/4\*4) \*graybmp.bi.biHeight;

    graybmp.bi.biBitCount = 8;

    graybmp.bi.biSizeImage =  (graybmp.bi.biWidth+3/4\*4) \*graybmp.bi.biHeight;

    initial\_bmp(graybmp);

    fwrite(&rgbquad,sizeof(rgbquad),1,grayfile);

    write\_gray\_data(graybmp,yuv);

    FILE \* meanfile = fopen("D://mean.bmp","wb");//create the mean bmp and do necessary initialization

    BMP meanbmp = {meanfile,graybmp.bf,graybmp.bi};

    initial\_bmp(meanbmp);

    fwrite(&rgbquad,sizeof(rgbquad),1,meanfile);

    mean(yuv,meanbmp);//act the convolution and store the processed data in u component

    write\_gray\_data(meanbmp,yuv+1);

    FILE \* lapfile = fopen("D://lap.bmp","wb");//create the modified bmp and do necessary initialization

    BMP lapbmp = {lapfile,graybmp.bf,graybmp.bi};

    initial\_bmp(lapbmp);

    fwrite(&rgbquad,sizeof(rgbquad),1,lapfile);

    lap(yuv,lapbmp);//act the convolution and store the processed data in u component

    write\_gray\_data(lapbmp,yuv+1);

}

int lap(double\*yuv,BMP lapbmp){

    int off[3] = {-1,0,1};// the offset in coordinate

    int mask[3][3] = {{0,-1,0},{-1,4,-1},{0,-1,0}};//the covolution kernel

    double \*off\_yuv = yuv +1;//where we store the result

    for(int i=0;i<lapbmp.bi.biWidth;i++){//store the most top nd bottom edge ,where wo wont do any modification

        off\_yuv[3\*i] = yuv[3\*i];

        off\_yuv[3\*i + (lapbmp.bi.biHeight-1)\*lapbmp.bi.biWidth\*3] = yuv[3\*i + (lapbmp.bi.biHeight-1)\*lapbmp.bi.biWidth\*3];

    }

    for(int i=0;i<lapbmp.bi.biHeight;i++){//store the most left and right edge ,where we wont do any modification

        off\_yuv[i\*lapbmp.bi.biWidth\*3] = yuv[i\*lapbmp.bi.biWidth\*3];

        off\_yuv[(i+1)\*lapbmp.bi.biWidth\*3-3] = yuv[ (i+1)\* lapbmp.bi.biWidth\*3-3];

    }

    for(int i=1;i<lapbmp.bi.biWidth-1;i++){//traverse the matrix

        for(int j=1;j<lapbmp.bi.biHeight-1;j++){

            double sum = 0;

            for(int n1=0;n1<3;n1++){//traverse the nearby pixels

                for(int n2=0;n2<3;n2++){

                    sum += yuv[3\*lapbmp.bi.biWidth\*(j+off[n1])+ 3\*(i+off[n2])]\*mask[n1][n2];//covolution

                }

            }

            off\_yuv[3\*i+j\*lapbmp.bi.biWidth\*3] = sum;

            if (off\_yuv[3\*i+j\*lapbmp.bi.biWidth\*3] < 0) off\_yuv[3\*i+j\*lapbmp.bi.biWidth\*3] = 0;//adjust if it is less than 0

            off\_yuv[3\*i+j\*lapbmp.bi.biWidth\*3] += yuv[3\*lapbmp.bi.biWidth\*j + 3\*i];

            off\_yuv[3\*i+j\*lapbmp.bi.biWidth\*3] = off\_yuv[3\*i+j\*lapbmp.bi.biWidth\*3]>255 ? 255 : off\_yuv[3\*i+j\*lapbmp.bi.biWidth\*3];//adjust if it is bigger than 255

        }

    }

}

int fill\_rgb\_yuv\_array(BMP rawbmp,unsigned char\* rgbdata, double \* yuv){

    int width = rawbmp.bi.biWidth\*3;

    int adjusted\_width = (width+3)/4\*4;

    unsigned char dustbin;

    int count=0;

    for(int i=0; i< rawbmp.bi.biWidth \* rawbmp.bi.biHeight; i++){

        fread(&rgbdata[3\*i],1,1,rawbmp.file);// read the  rgb number

        fread(&rgbdata[3\*i+1],1,1,rawbmp.file);

        fread(&rgbdata[3\*i+2],1,1,rawbmp.file);

        unsigned char y = 0.299\*rgbdata[3\*i+2] + 0.587\*rgbdata[3\*i+1] + 0.114\*rgbdata[3\*i];

        yuv[3\*i] = 0.299\*rgbdata[3\*i+2] + 0.587\*rgbdata[3\*i+1] + 0.114\*rgbdata[3\*i] ;//claculate the yuv number

        yuv[3\*i+1] = -0.147\*rgbdata[3\*i+2] + -0.289\*rgbdata[3\*i+1] + 0.435 \*rgbdata[3\*i];

        yuv[3\*i+2] = 0.615\*rgbdata[3\*i+2] + -0.515\*rgbdata[3\*i+1] + -0.1\*rgbdata[3\*i];

        count+=3;

        if (count == width){// read the 0s that just take up places

            while(count++ != adjusted\_width){

                fread(&dustbin,1,1,rawbmp.file);

            }

            count = 0;

        }

    }

}

BMP create\_bmp(){

    BMP newbmp;

    FILE \* bmp;

    BITMAPFILEHEADER bf;

    BITMAPINFOHEADER bi;

    if( !(bmp = fopen("D:\\test.bmp","rb"))){

        printf("Not find file in D:\\test.bmp");

        return newbmp;

    }

    fread(&bf,sizeof(BITMAPFILEHEADER),1,bmp);

    fread(&bi,sizeof(BITMAPINFOHEADER),1,bmp);

    newbmp.bf = bf;

    newbmp.bi = bi;

    newbmp.file = bmp;

    return newbmp;

}

int initial\_bmp(BMP bmp){//write file headers

    fwrite(&bmp.bf,sizeof(bmp.bf),1,bmp.file);

    fwrite(&bmp.bi,sizeof(bmp.bi),1,bmp.file);

}

int write\_gray\_data(BMP bmp,double\* graydata){

    int width = bmp.bi.biWidth\*3;

    int adjusted\_width = (width+3)/4\*4;

    int count=0;

    unsigned char y;

    for(int i=0;i<bmp.bi.biHeight\*bmp.bi.biWidth;i++){

        y = (unsigned char)graydata[3\*i];

        fwrite(&y,sizeof(unsigned char),1,bmp.file);

        count+=1;

        if (count == width){// write the 0s that just take up places

            while(count++ != adjusted\_width){

                fwrite(graydata,1,1,bmp.file);

            }

            count = 0;

        }

    }

}

int mean(double \* yuv,BMP meanbmp){

    int off[5] = {-2,-1,0,1,2};//the offset of coordinates

    double \*off\_yuv = yuv +1;//where we store our result

    for(int i=0;i<meanbmp.bi.biWidth;i++){//store the top two and bottom two edges

        off\_yuv[3\*i] = yuv[3\*i];

        off\_yuv[3\*i+meanbmp.bi.biWidth\*3] = yuv[3\*i+meanbmp.bi.biWidth\*3];

        off\_yuv[3\*i + (meanbmp.bi.biHeight-2)\*meanbmp.bi.biWidth\*3] = yuv[3\*i + (meanbmp.bi.biHeight-2)\*meanbmp.bi.biWidth\*3];

        off\_yuv[3\*i + (meanbmp.bi.biHeight-1)\*meanbmp.bi.biWidth\*3] = yuv[3\*i + (meanbmp.bi.biHeight-1)\*meanbmp.bi.biWidth\*3];

    }

    for(int i=0;i<meanbmp.bi.biHeight;i++){//store the left two and right two edges

        off\_yuv[i\*meanbmp.bi.biWidth\*3] = yuv[i\*meanbmp.bi.biWidth\*3];

        off\_yuv[i\*meanbmp.bi.biWidth\*3+3] = yuv[i\*meanbmp.bi.biWidth\*3+3];

        off\_yuv[(i+1)\*meanbmp.bi.biWidth\*3-6] = yuv[ (i+1)\* meanbmp.bi.biWidth\*3-6];

        off\_yuv[(i+1)\*meanbmp.bi.biWidth\*3-3] = yuv[ (i+1)\* meanbmp.bi.biWidth\*3-3];

    }

    for(int i=2;i<meanbmp.bi.biWidth-2;i++){//traverse the matrix

        for(int j=2;j<meanbmp.bi.biHeight-2;j++){

            double sum = 0;

            for(int n1=0;n1<5;n1++){//traverse the nearby pixels

                for(int n2=0;n2<5;n2++){

                    sum += yuv[3\*meanbmp.bi.biWidth\*(j+off[n1])+ 3\*(i+off[n2])];

                }

            }

            sum /= 25;// get the average

            off\_yuv[3\*i+j\*meanbmp.bi.biWidth\*3] = sum;//store

        }

    }

}

**四、实验环境及运行方法**

直接用gcc 编译运行就可以，需要注意的是D:/test.bmp必须存在，作为输入文件，输出的文件gray.bmp mean.bmp lap.bmp也会直接存储在D:/目录下

**五、实验结果展示**

原图：



灰度图：



均值处理：



拉普普拉斯滤波处理：



**六、心得体会**

本次实验和之前那次形态学操作有许多相似的地方，所以均值操作还比较简单，但是拉普拉斯滤波要注意一些向上溢出和向下溢出的问题，需要把他调整到255和0，最后的结果就比较理想。