浙江大学实验报告

课程名称：图象信息处理　　指导老师：宋明黎　　成绩：

实验名称：Assignment-5　Filtering

专业：求是科学班（计算机）

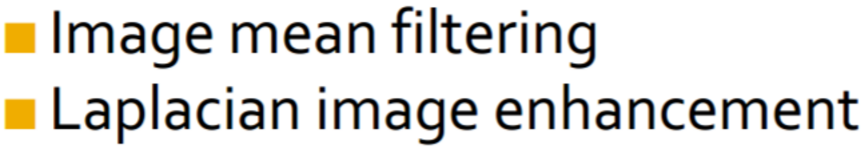
姓名：蒋仕彪

学号：3170102587

日期：2018/12/15

一、实验目的和要求

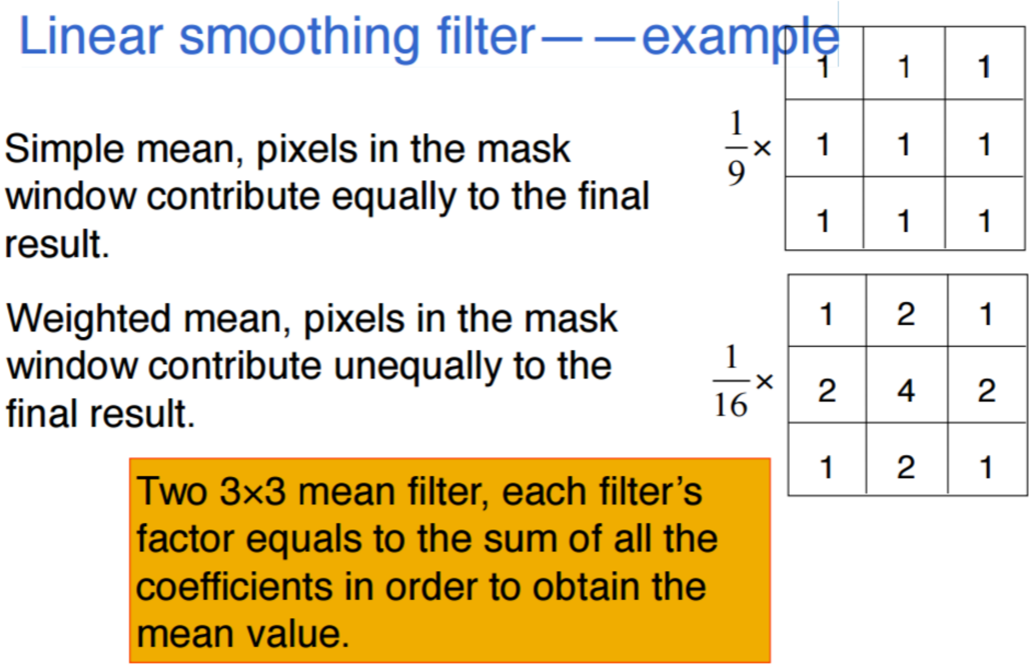
学习和掌握均值滤波和拉普拉斯算子。



- Image mean filtering
- Laplacian image enhancement

二、实验内容和原理

1. Mean Filter（均值滤波）



Linear smoothing filter——example

Simple mean, pixels in the mask window contribute equally to the final result.

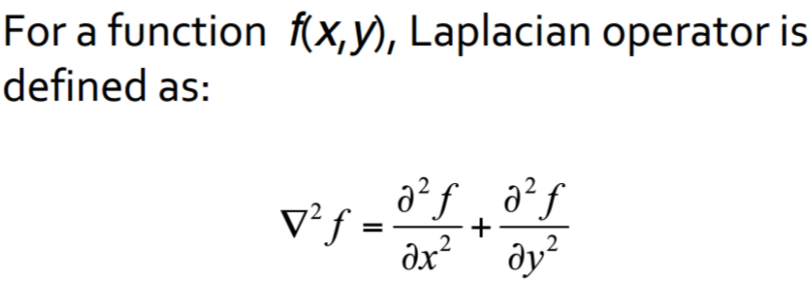
$$\frac{1}{9}\times\begin{array}{|c|c|c|}\hline 1 & 1 & 1\\\hline 1 & 1 & 1\\\hline 1 & 1 & 1\\\hline\end{array}$$

Weighted mean, pixels in the mask window contribute unequally to the final result.

$$\frac{1}{16}\times\begin{array}{|c|c|c|}\hline 1 & 2 & 1\\\hline 2 & 4 & 2\\\hline 1 & 2 & 1\\\hline\end{array}$$

Two 3×3 mean filter, each filter’s factor equals to the sum of all the coefficients in order to obtain the mean value.

2. Spatial filtering（Laplacian operator）



For a function *f*(*x*,*y*), Laplacian operator is defined as:

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

Mask of Laplacian operator

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

0	1	0
1	-4	1
0	1	0

It is rotation invariant.

Extending the mask

The elements in the diagonal direction can also be taken into account :

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

Or

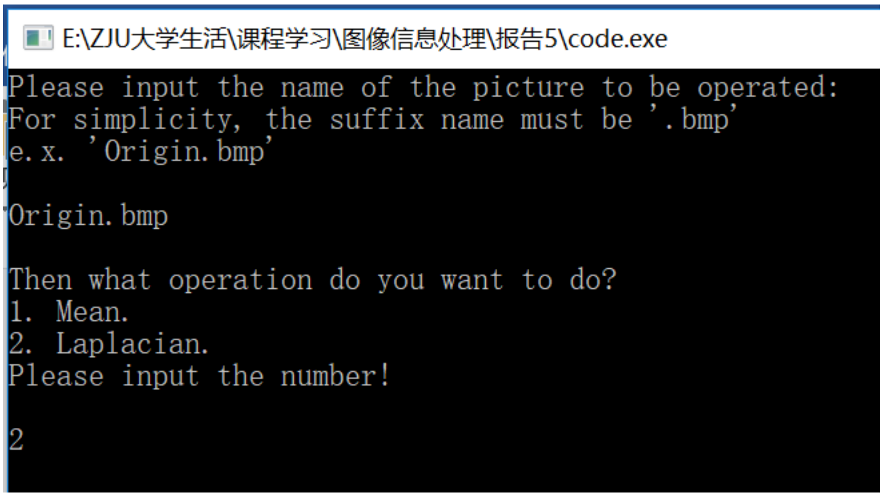
$$\nabla^2 f = \sum_{i=-1}^{1}\sum_{j=-1}^{1} f(x+i,y+j)-9f(x,y)$$

Application of Laplacian operator

Image enhancement by Laplaician:

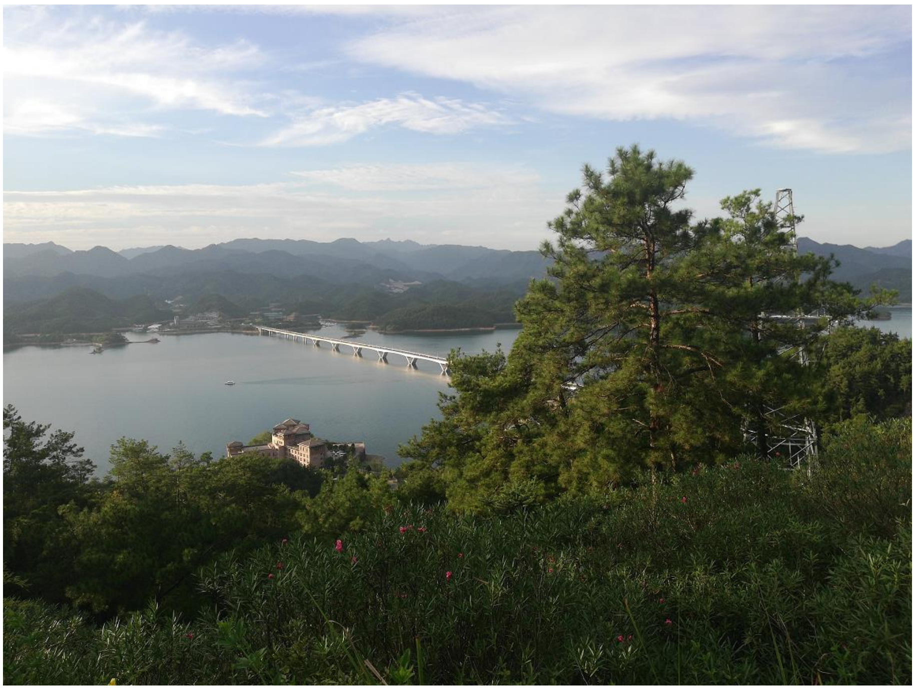
$$g(x,y)=\begin{cases} f(x,y)-\nabla^2 f(x,y) & \text{If the center element of the mask is negative}\\ f(x,y)+\nabla^2 f(x,y) & \text{If the center element of the mask is positive}\end{cases}$$

三、成果展示

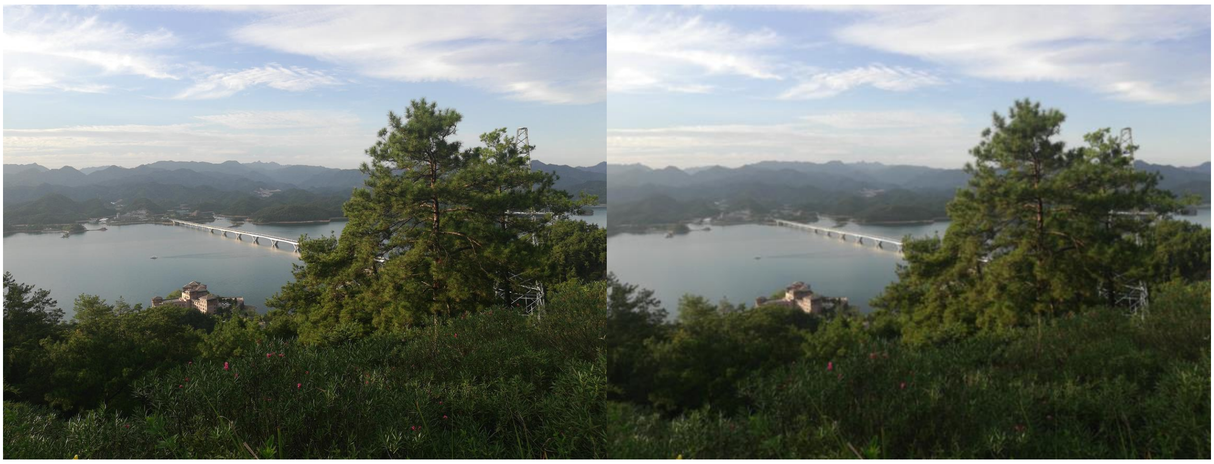


```
Please input the name of the picture to be operated:
For simplicity, the suffix name must be '.bmp'
e.x. 'Origin.bmp'

Origin.bmp

Then what operation do you want to do?
1. Mean.
2. Laplacian.
Please input the number!

2
```

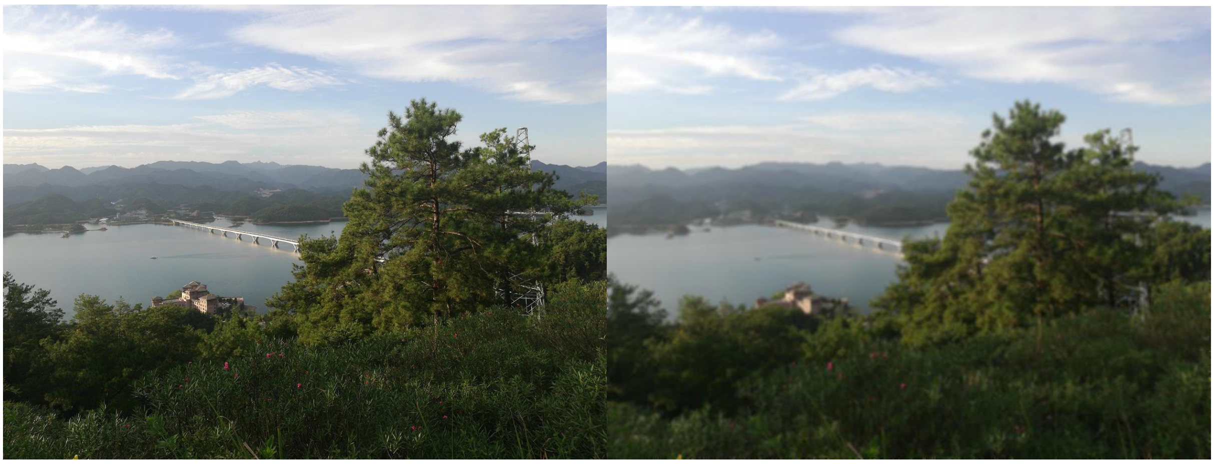
交互界面



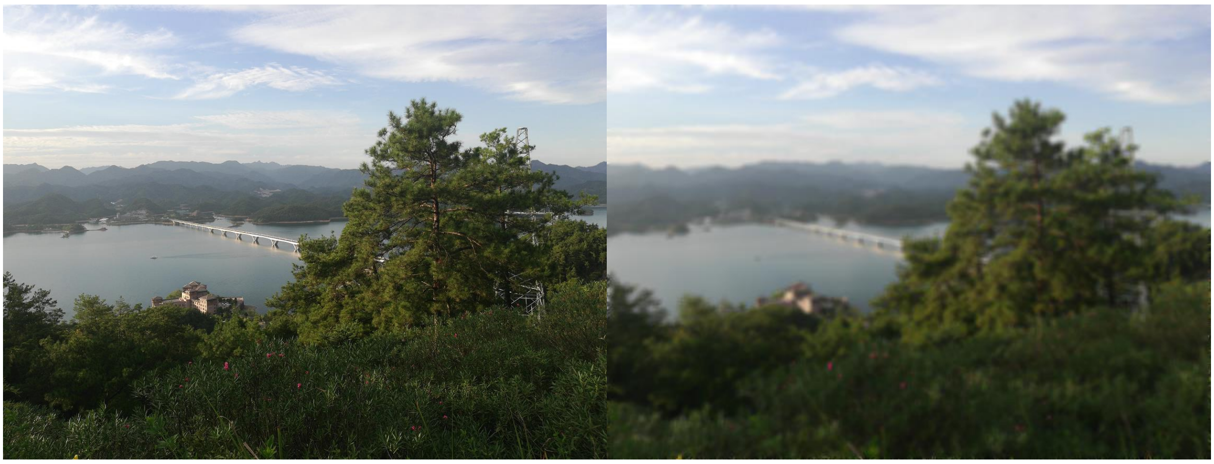
原图像



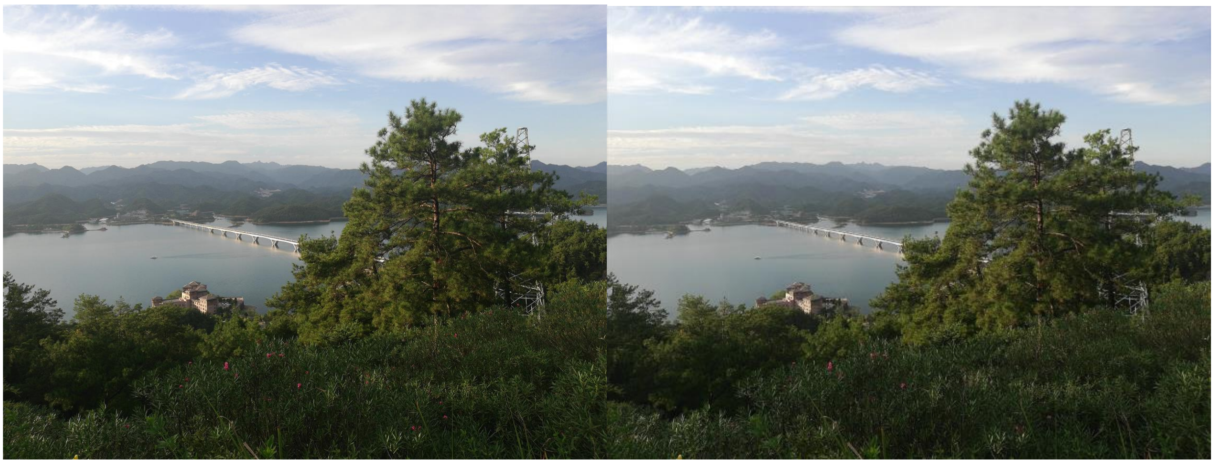
均值滤波（边长为3）（左边为原图）



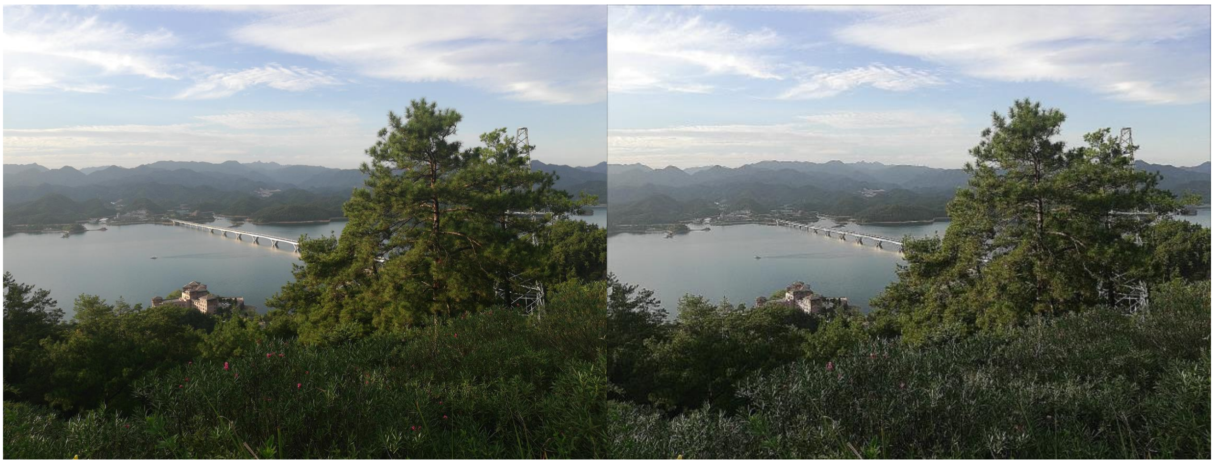
均值滤波（边长为7）（左边为原图）



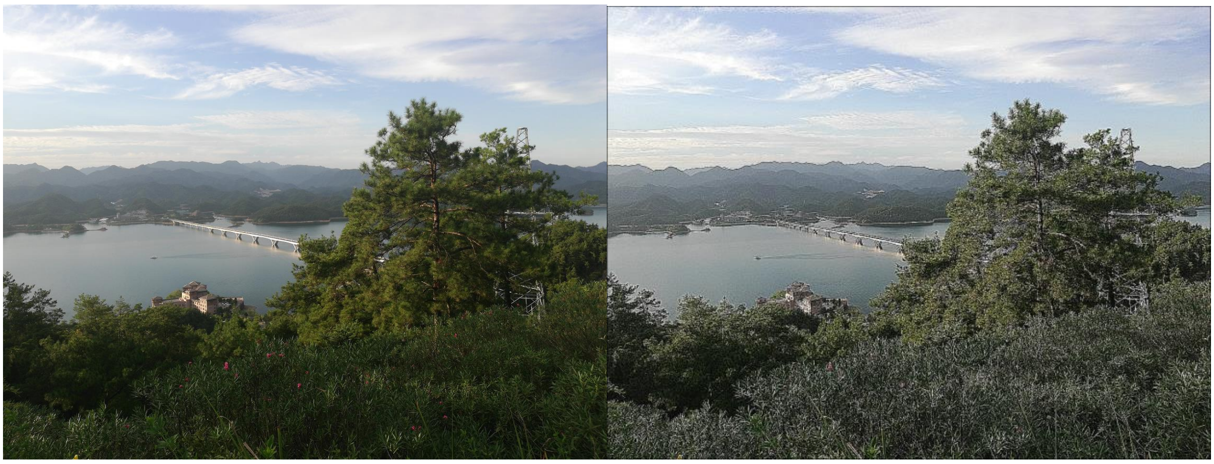
均值滤波（边长为11）（左边为原图）



锐化（系数为0.1）（左边为原图）



锐化（系数为0.4）（左边为原图）



锐化（系数为1）（左边为原图）

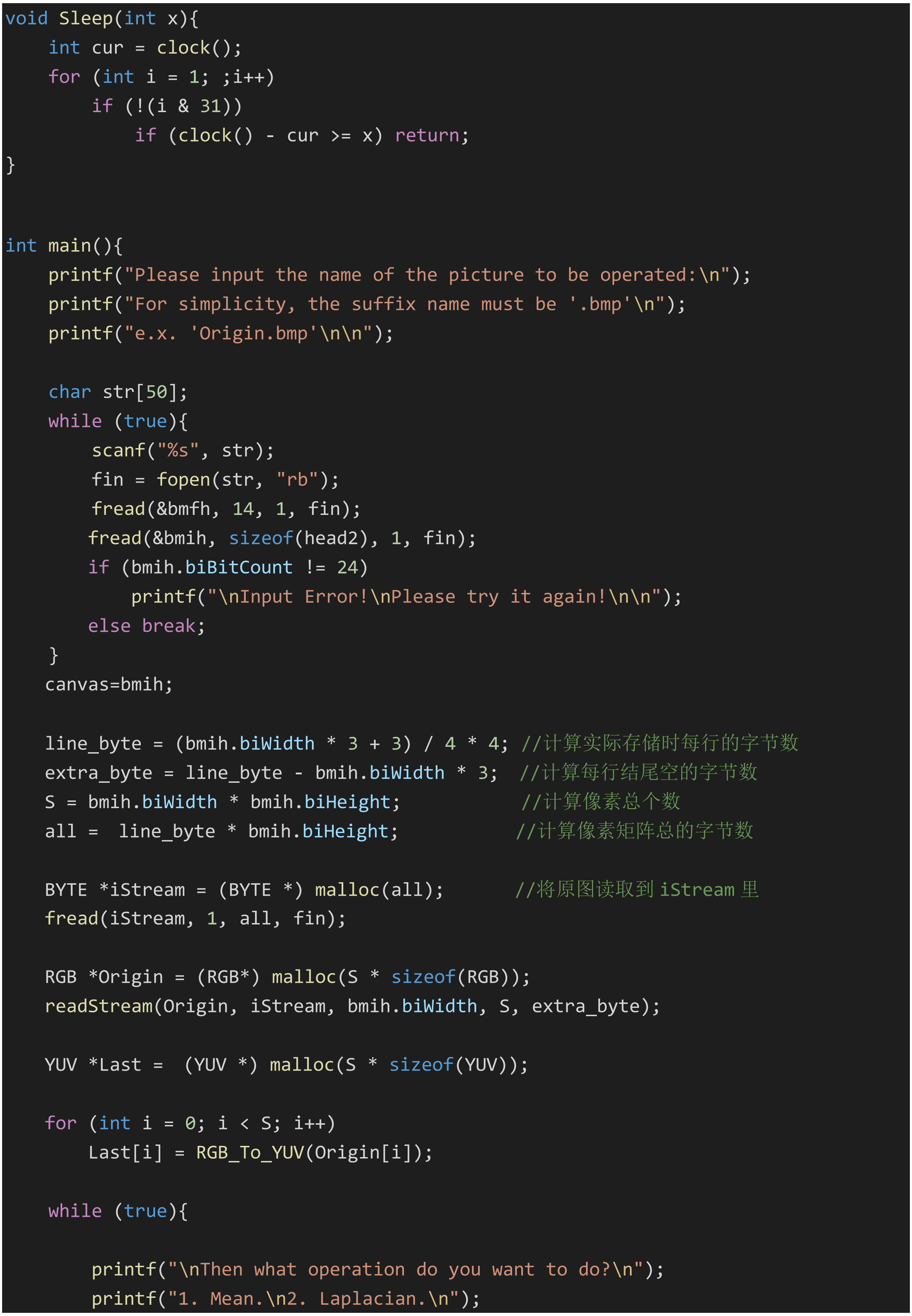
四、源代码与分析}

Mean 操作和 Laplacian 算子真是有趣呀，它们的效果完全是互逆的。Mean 是把当前格子取周围一圈的平均值，以达到“模糊”的效果；而 Laplacian 算子通过求二阶导，进一步加大和周围的差距，达到“锐化”的效果。

一般而言，Mean 的效果更加不错，因为它是在更充足的信息里模糊图像。

Laplacian 算子虽然达到了锐化（有点像是清晰）的效果，但是图片的信息量不会变大，导致运行后会有很多白色的不和谐的色块。

```
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <stdlib.h>
#include <algorithm>
#include <cstring>
#include <time.h>

using namespace std;

typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned int DWORD;
typedef int LONG;

FILE *fin , *fout;

typedef struct tagBITMAPFILEHEADER{
    WORD type;
    DWORD bfSize;
    WORD bfReserved1;
    WORD bfReserved2;
    DWORD bfOffBits;
}head1;
//定义第一个头

typedef struct tagBITMAPINFOHEADER{
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD  biCompression;
    DWORD  biSizeImage;
    LONG  biXPelsPerMeter;
    LONG  biYPelsPerMeter;
    DWORD  biClrUsed;
    DWORD  biClrImportant;
}head2;
//定义第二个头

typedef struct _RGB{
    BYTE R;
    BYTE G;
    BYTE B;
}RGB;

typedef struct _YUV{
    short Y;
    short U;
    short V;
}YUV;
//YUV 格式可能会有负数，就直接用 short 存了

typedef struct _HSV{
    short H;
    short S;
    short V;
}HSV;

YUV RGB_To_YUV(RGB cur){
    YUV ret;
    ret.Y = round(0.299 * cur.R + 0.587 * cur.G + 0.114 * cur.B);
    ret.U = round(-0.147 * cur.R - 0.289 * cur.G + 0.435 * cur.B);
    ret.V = round(0.615 * cur.R - 0.515 * cur.G - 0.100 * cur.B);
    return ret;
}

BYTE In(short cur){
    if (cur > 255) cur = 255;
    if (cur < 0)   cur = 0;
    return (BYTE)cur;
}
//担心 YUV 转 RGB 时导致 RGB 范围出错，写一个框定范围的函数


RGB YUV_To_RGB(YUV cur){
    RGB ret;
    ret.R = In(round(cur.Y + 1.14 * cur.V));
    ret.G = In(round(cur.Y - 0.395 * cur.U - 0.581 * cur.V));
    ret.B = In(round(cur.Y + 2.033 * cur.U));
    return ret;
}

int line_byte, extra_byte, S, all;
head1 bmfh;
head2 bmih, canvas;
//原图

struct exRGB{
    short R;
    short G;
    short B;
};

void readStream(RGB *cur, BYTE *p, int W, int S, int extra_byte){
    for (int i = 0; i < S; i++){
        cur->R = *p++;
        cur->G = *p++;
        cur->B = *p++;
        if ((i + 1) % bmih.biWidth == 0)
            p = p + extra_byte;
        cur++;
    }
}
//从读入流里获取宽度为 W，总大小为 S 的像素矩阵

void printStream(short *Y, YUV *Z, BYTE *p, int W, int S, int extra_byte){
    for (int i = 0; i < S; i++){
        YUV T = Z[i]; T.Y = Y[i];
        RGB now = YUV_To_RGB(T);
        *p++ = now.R;
        *p++ = now.G;
        *p++ = now.B;
        if ((i + 1) % W == 0)
            for (int k = 0; k < extra_byte; k++)
                *p++ = 0;
    }
}
//将宽度为 W，总大小为 S 的像素矩阵放入输出流 p 里。

void printPicture(short *q, YUV *Last, head2 canvas, char *str){
    BYTE *oStream = (BYTE *) malloc(canvas. biSizeImage);
    printStream(q, Last, oStream, canvas.biWidth, S, extra_byte);
    fout = fopen(str, "wb");
    fwrite(&bmfh, 14, 1, fout);
    fwrite(&canvas, sizeof(head2), 1, fout);
    fwrite(oStream, 1, canvas. biSizeImage, fout);
}
//将像素矩阵 p 里的结果输出至 str 文件


void Mean(YUV *p, int w, int h, int L = 3){
    L >>= 1;
    short *q = (short *)malloc(S * sizeof(short));
    for (int i = 0; i < h; i++)
        for (int j = 0; j < w; j++)
            if (i < L || i + L >= h || j < L || j + L >= w)
                q[i * w + j] = p[i * w + j].Y;
            else {
                short Q = 0;
                for (int dx = -L; dx <= L; dx++)
                    for (int dy = -L; dy <= L; dy++)
                        Q += p[(i + dx) * w + j + dy].Y;
                q[i * w + j] = round(Q / (1.0 * (2 * L + 1) * (2 * L + 1)));
            }
    printPicture(q, p, canvas, (char *)"Mean.bmp");
}

void Laplacian(YUV *p, int w, int h, double Xi){
    short *q = (short *)malloc(S * sizeof(short));
    for (int i = 0; i < h; i++)
        for (int j = 0; j < w; j++)
            if (i == 0 || i == h - 1 || j ==0 || j == w - 1)
                q[i * w + j] = p[i * w + j].Y;
            else {
                q[i * w + j] = -9 * p[i * w + j].Y;
                for (int dx = -1; dx <= 1; dx++)
                    for (int dy = -1; dy <= 1; dy++)
                        q[i * w + j] += p[(i + dx) * w + j + dy].Y;
            }

    for (int i = 0; i < w * h; i++)
        q[i] = p[i].Y - round(q[i] * Xi);

    printPicture(q, p, canvas, (char *)"Laplacian.bmp");
}
```



```
void Sleep(int x){
    int cur = clock();
    for (int i = 1; ;i++)
        if (!(i & 31))
            if (clock() - cur >= x) return;
}


int main(){
    printf("Please input the name of the picture to be operated:\n");
    printf("For simplicity, the suffix name must be '.bmp'\n");
    printf("e.x. 'Origin.bmp'\n\n");

    char str[50];
    while (true){
        scanf("%s", str);
        fin = fopen(str, "rb");
        fread(&bmfh, 14, 1, fin);
        fread(&bmih, sizeof(head2), 1, fin);
        if (bmih.biBitCount != 24)
            printf("\nInput Error!\nPlease try it again!\n\n");
        else break;
    }
    canvas=bmih;

    line_byte = (bmih.biWidth * 3 + 3) / 4 * 4; //计算实际存储时每行的字节数
    extra_byte = line_byte - bmih.biWidth * 3;  //计算每行结尾空的字节数
    S = bmih.biWidth * bmih.biHeight;             //计算像素总个数
    all =  line_byte * bmih.biHeight;          //计算像素矩阵总的字节数

    BYTE *iStream = (BYTE *) malloc(all);      //将原图读取到 iStream 里
    fread(iStream, 1, all, fin);

    RGB *Origin = (RGB*) malloc(S * sizeof(RGB));
    readStream(Origin, iStream, bmih.biWidth, S, extra_byte);

    YUV *Last =  (YUV *) malloc(S * sizeof(YUV));

    for (int i = 0; i < S; i++)
        Last[i] = RGB_To_YUV(Origin[i]);

    while (true){

        printf("\nThen what operation do you want to do?\n");
        printf("1. Mean.\n2. Laplacian.\n");
        printf("Please input the number!\n\n");

        int id;
        scanf("%d", &id);puts("");
        switch (id){
            case 1:{
                printf("Please input the size length (odd number) of the block for averaging (1 ~ 11).\n");
                printf("ex.  3\n");
                int L;
                scanf("%d", &L);
                Mean(Last, bmih.biWidth, bmih.biHeight, L);
                printf("\nOriginal picture have printed in ""Mean.bmp""!\n");
                break;
            }
            case 2:{
                printf("Please input the Sharpening coefficient (0 ~ 1).\n");
                printf("e.x.  0.3\n");
                double theta;
                scanf("%lf", &theta);
                Laplacian(Last, bmih.biWidth, bmih.biHeight, theta);
                printf("\nOriginal picture have printed in ""Laplacian.bmp""!\n");
                break;
            }
            default:
                printf("Wrong input!\n");
        }
        Sleep(1000);
        printf("\nDo you want to try again?\n  1. Yes\n  2. Quit\n\n");
        int q; scanf("%d", &q);
        if (q == 2) break;
        system("cls");
    }

    return 0;
}
```