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-6 Bilateral Filters

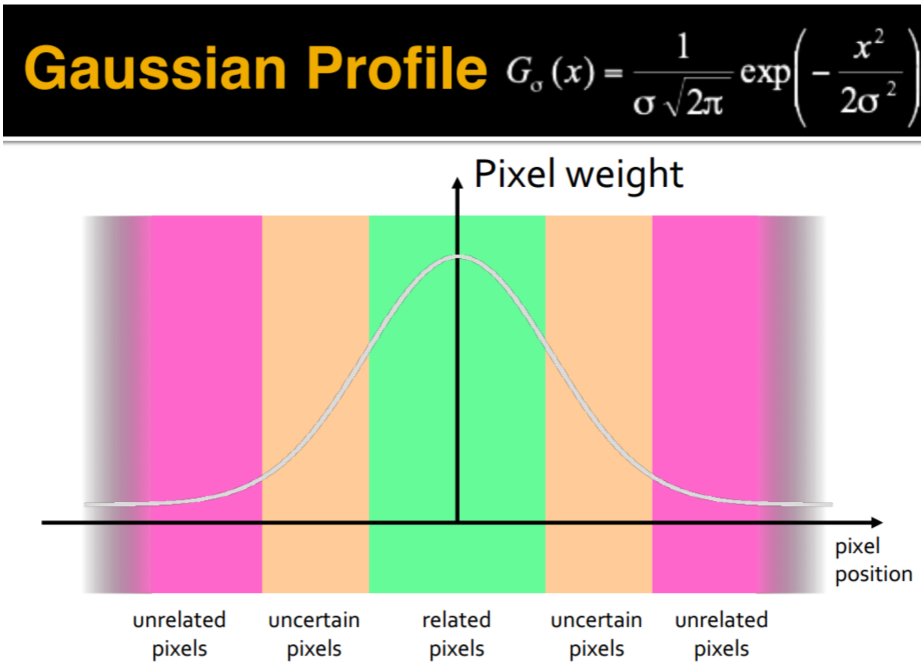
专业：求是科学班（计算机）
姓名：蒋仕彪
学号：3170102587
日期：2018/12/31

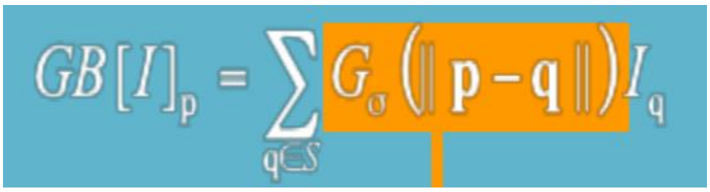
一、实验目的和要求

学习和掌握双边滤波（Bilateral Filters）。

二、实验内容和原理

1. Gaussian Blur（高斯模糊）

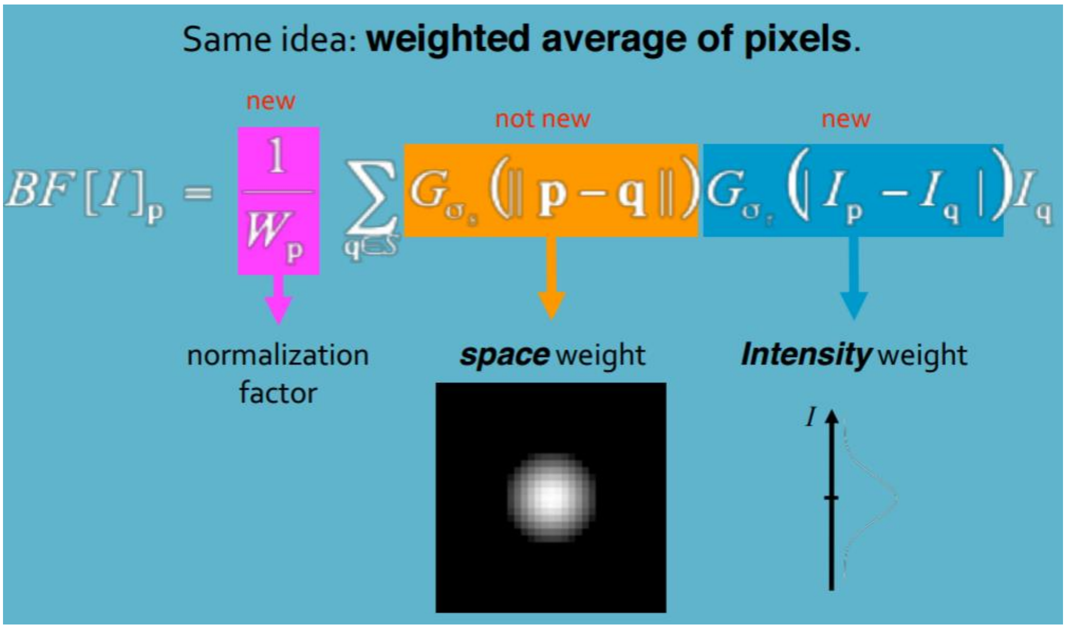


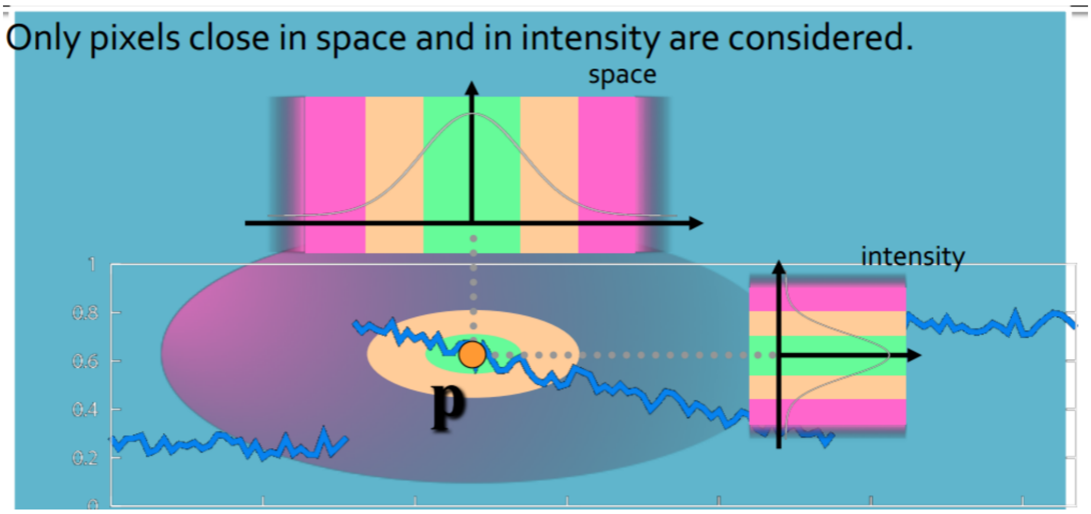


高斯模糊用来处理图像中的噪点，使图像变得更加“光滑”——但是也自然地会变“糊”。

具体做法是，将每一个像素赋值成它周围像素点的加权平均。离它近的点显然比重应该搞一些，然后很自然地想到了正态分布函数。所以它周围每一个点的权值对于它的比重就是一个 μ=0，σ 可以调整的正态分布。注意到，实际处理图像的时候为了加速，计算每个像素新的亮度值时，只需取出它周围的一个“窗格”（例如 20*20）来进行计算，以为离得远了正态分布函数迅速衰减。

2. 双边滤波（Bilateral Filters）





高斯模糊会有一个缺点：在去掉噪声的同时，把一些边界也弄得模糊了，这不符合我们的需求；而双边滤波可以很好地解决这个问题。

双边滤波其实就是在高斯模糊的基础上，还加上了一个亮度大小的权。感性地想一想，和中心亮度值相近的点应该被赋予更多的权重，以避免边界被外侧的点影响。完全类似高斯模糊的方法，也用 μ=p_i，σ可以调整的正态分布来赋予权重（p_i 是这个像素的亮度值）。

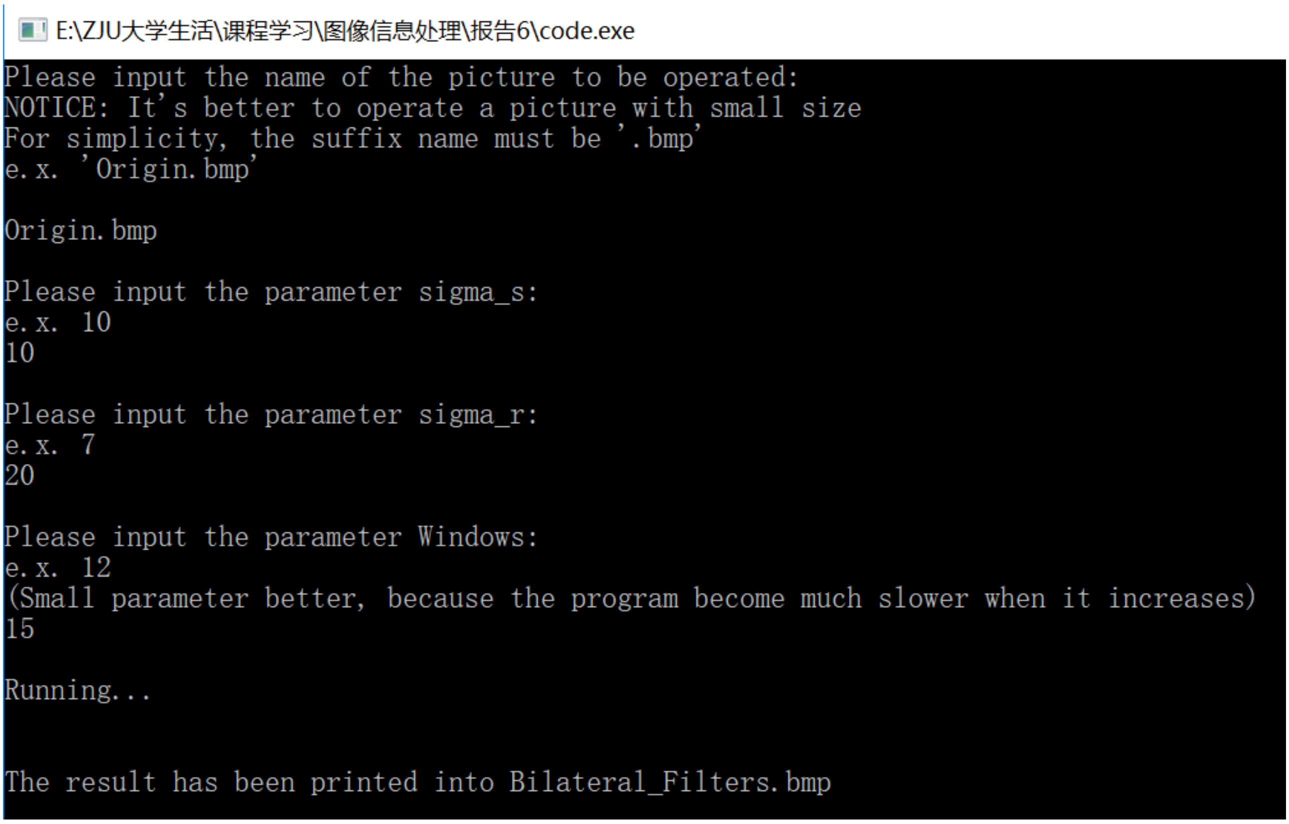
将正态分布展开，实际的权重公式为：

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{\|f(i,j)-f(k,l)\|^2}{2\sigma_r^2}\right)$$

其中有 σ_1 和 σ_2 两个参数可以调节。

我在实现的时候，每次枚举的窗格 K 也是可调的，所以一共要输入三个参数。事实上，窗格只要不要过小（如>10），得到的结果都比较接近。

三、成果展示



交互界面

左 1：原图　　左 2：调用（σ_1, σ_2, K）=（10,9,15）　左 3：调用（σ_1, σ_2, K）=（10,20,15）

左 1：原图　　左 2：调用（σ_1, σ_2, K）=（20,7,12）

左 1：原图　　左 2：调用（σ_1, σ_2, K）=（1.5,1000,8）　　左 3：调用（σ_1, σ_2, K）=（3.5,1000,8）

左 1：原图　　左 2：调用（σ_1, σ_2, K）=（10,7,12）　　左 3：调用（σ_1, σ_2, K）=（10,18,12）

四、源代码与分析

双边滤波的处理简单，但结果却意外的好。其本质其实不难理解，在高斯模糊的的基础上，加强周围相似点的权重以起到防止边界被“污染”的作用。

原理简单，但是调参还是挺复杂的。对于不同的图片，要尝试各种不同的参数才能得到一个比较完美的结果。通过多次测试发现，我的参数 K 没有太大的影响（当然 K 必须满足大于一个下界）。也就是说，高斯分布的衰减很快， -8~8 的区域和-20~20 没有太大区别，远处的格子很难影响当前的答案。

```
#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

int sqr(int x){return x*x;}

void Bilateral_Filters(YUV *p, int w, int h, double sigmas = 10, double sigmar = 7, int
block = 12){
    sigmas = sigmas * sigmas * 2;
    sigmar = sigmar * sigmar * 2;
    short *q = (short *)malloc(S * sizeof(short));
    for (int i = 0; i < h; i++)
        for (int j = 0; j < w; j++){
            double sum = 0, tot = 0;
            for (int dx = -block; dx <= block; dx++)
                for (int dy = -block; dy <= block; dy++){
                    int x = i + dx;
                    int y = j + dy;
                    if (x < 0 || x >= h || y < 0 || y >= w) continue;
                    int dist = dx * dx + dy * dy;
                    double weight = exp(- dist / sigmas - sqr(p[i * w + j].Y - p[x *
w + y].Y) / sigmar);
                    tot += weight; sum += weight * p[x * w + y].Y;
                }
            q[i * w + j] = round(sum / tot);
        }

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


int main(){
    printf("Please input the name of the picture to be operated:\n");
    printf("NOTICE: It's better to operate a picture with small size\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]);



    printf("\nPlease input the parameter sigma_s:\ne.x. 10\n");
    double sigma_s;
    scanf("%lf", &sigma_s);

    printf("\nPlease input the parameter sigma_r:\ne.x. 7\n");
    double sigma_r;
    scanf("%lf", &sigma_r);

    printf("\nPlease input the parameter Windows:\ne.x. 12\n");
    printf("(Small parameter better, because the program become much slower when it
increases)\n");
    int windows;
    scanf("%d", &windows);

    puts("\nRunning...\n");
    Bilateral_Filters(Last, bmih.biWidth, bmih.biHeight, sigma_s, sigma_r, windows);
    printf("\nThe result has been printed into ""Bilateral_Filters.bmp""\n\n");
    system("pause");
    return 0;
}
```