专业：计算机科学与技术
姓名：龙永奇
学号：3220105907
日期：2023/12/27

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：暴力实现双边滤波

一、实验目的和要求

暴力实现双边滤波

二、实验内容和原理

➢ 高斯滤波

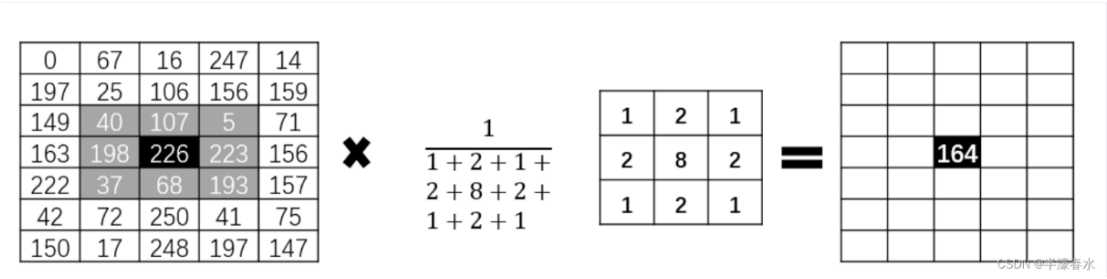
高斯滤波是一种线性平滑滤波方法，其效果是滤除掉图像中随机出现的高斯噪声，其本质是一种低通滤波，通过在滤除噪声的过程中对图像的边缘信息进行平滑从而使得图像变得模糊，就是对整幅图像进行加权平均，每一点像素值由其本身和邻域内的其他像素值加权平均得到。以 q 为中心的窗口中，某一点的 p 高斯滤波中的权重计算算法公式为：

$$GB[I]_p = \sum_{q \in S} G_\sigma(||p-q||) I_q$$

σ 为窗口大小，其通常被设置为图像大小的比例

- σ 越大平滑效果越明显。当 σ 趋于∞时，等价为均值滤波，每个权重相同
- σ 越小中心点的权重越大，对图像的滤波效果越弱。当 σ 趋于 0 时输出原图

其具体操作为：用一个卷积或者掩膜扫描图像中的每个像素，用模板确定邻域内像素的加权平均灰度替代中心像素点的值。以 3*3 卷积核为例：

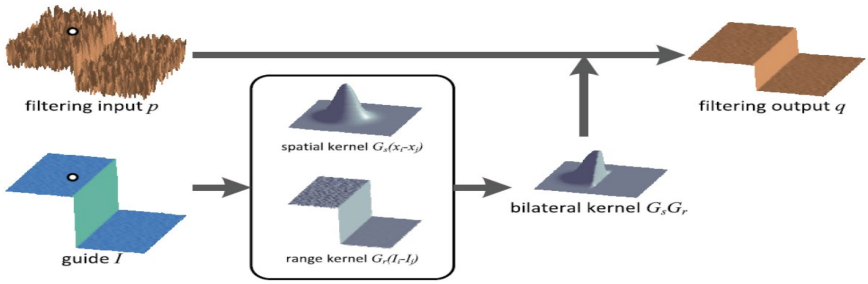


效果如下：


但是高斯滤波只考虑了距离，因此在平坦区域距离越近的部分像素分布较为接近，但是当像素值变化率较大的区域边缘，高斯滤波会在一定程度上造成边缘信息的丢失。

- **双边滤波**

在上一部分的结尾我们发现高斯滤波会在一定情况下造成边缘信息的丢失，使得边缘模糊，因此我们需要使用例如双边滤波的边缘保护滤波法。其核心思想在于每个样本被周围的加权平均取代，此权重受到与中心像素的距离以及像素值和中心像素值相似程度的双重影响。



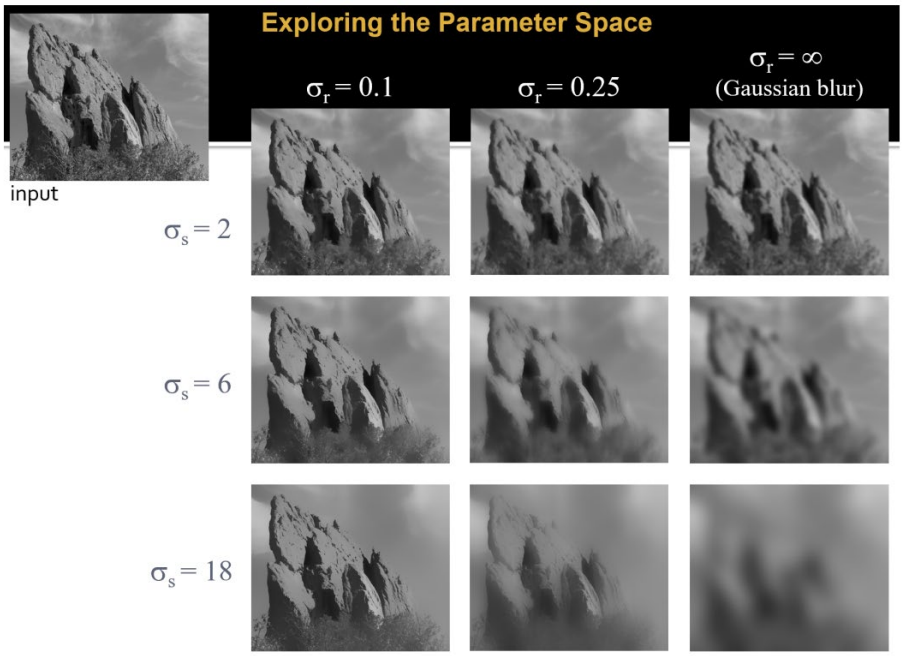
其算法公式为：

$$BF[I]_p = \frac{1}{W_p}\sum_{q\in S} G_{\sigma_s}(||p-q||)G_{\sigma_r}(|I_p-I_q|)I_q$$

- $1/W_p$ 为归一化系数
- $G_{\sigma_s}(||p-q||)$ 为空间权重，同高斯滤波
- $G_{\sigma_r}(|I_p-I_q|)I_q$ 为灰度权重

其中 σ_s 的取值方法同高斯滤波，对于 σ_r，其取值越大意味着边缘越模糊，当趋

于∞时和空间域模板做积后等价于均值滤波；其取值越小，边缘清晰，当趋于 0 时与空间域模板做积后即为原图像。具体取值对图像的效果影响如下（取自 ppt）：



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.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;
// 给八位灰度图加上调色盘，本次实验分别对 24 位和 8 位进行图像几何变换
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);
// 平移
TrAnslate(&Org, &Ans, BMPwidth/2, BMPheight/2);
free(Ans.pix_val);
// 双边滤波
Bilateral(&Org, &Ans, 9);
free(Ans.pix_val);
```

- **双边滤波实现**

```
//偏移
int Position(int x, int y, int byte_offset, int pix_val, int Height, int
Width){
    if(x < 0) x = 0;
    else if(x >= Height) x = Height - 1;
    if(y < 0) y = 0;
    else if(y >= Width) y = Width - 1;
    return x * byte_offset + y * pix_val;
}
//Guass 函数
double G(double sqr, double sig){
    return exp(- sqr / (2 * pow(sig, 2)));
}
// 双边滤波
void Bilateral(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_1, now_position_2, i, j, p, q, edge = scale / 2; // 窗口大
小
    int row_offset = (Org->IH.biBitCount / 8 * Org->IH.biWidth + 3) / 4 * 4;
    double sig_r = 180; // 可调
    double sig_s = 0.2 * ((Org->IH.biHeight + Org->IH.biWidth) / 2); // 可调
    double tmp_R, tmp_G, tmp_B, sum_R, sum_G, sum_B, WR, WG, WB, GrayR, GrayG,
GrayB, prstR, prstG, prstB, Dis, sum, Gray, W, tmp, prst;
    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
                sum_R = sum_G = sum_B = WR = WG = WB = 0.0;
                now_position_1 = Position(i, j, row_offset, 3, Org->IH.biHeight,
Org->IH.biWidth);
```

```
                prstB = Org->pix_val[now_position_1];
                prstG = Org->pix_val[now_position_1 + 1];
                prstR = Org->pix_val[now_position_1 + 2];
                for(p = i - edge; p <= i + edge; p++){
                    for(q = j - edge; q <= j + edge; q++){
                        now_position_2 = Position(p, q, row_offset, 3,
Org->IH.biHeight, Org->IH.biWidth);
                        tmp_B = Org->pix_val[now_position_2];
                        tmp_G = Org->pix_val[now_position_2 + 1];
                        tmp_R = Org->pix_val[now_position_2 + 2];
                        // 求灰度
                        GrayR = G(pow(tmp_R - prstR, 2), sig_r);
                        GrayG = G(pow(tmp_G - prstG, 2), sig_r);
                        GrayB = G(pow(tmp_B - prstB, 2), sig_r);
                        // 求和中心的距离
                        Dis = G(pow(i - p, 2) + pow(j - q, 2), sig_s);
                        // 求归一化系数
                        WR += GrayR * Dis;
                        WG += GrayG * Dis;
                        WB += GrayB * Dis;
                        // 求和
                        sum_R += GrayR * Dis * tmp_R;
                        sum_G += GrayG * Dis * tmp_G;
                        sum_B += GrayB * Dis * tmp_B;
                    }
                }
                Ans->pix_val[now_position_1] = sum_B / WB;
                Ans->pix_val[now_position_1 + 1] = sum_G / WG;
                Ans->pix_val[now_position_1 + 2] = sum_R / WR;
            }else{ // 8 位灰度图
                sum = W = 0;
                now_position_1 = Position(i, j, row_offset, 1, Org->IH.biHeight,
Org->IH.biWidth);
                prst = Org->pix_val[now_position_1];
                for(p = i - edge; p <= i + edge; p++){
                    for(q = j - edge; q <= j + edge; q++){
                        now_position_2 = Position(p, q, row_offset, 1,
Org->IH.biHeight, Org->IH.biWidth);
                        tmp = Org->pix_val[now_position_2];
                        // 求灰度
                        Gray = G(pow(tmp - prst, 2), sig_r);
                        // 求和中心距离
                        Dis = G(pow(i - p, 2) + pow(j - q, 2), sig_s);
                        // 求归一化系数
```

```
                    W += Gray * Dis;
                    // 求和
                    sum += Gray * Dis * tmp;
                }
            }
            Ans->pix_val[now_position_1] = sum / W;
        }
    }
}
Output(Ans, "Bilateral.bmp");
}
```

我们默认的窗口大小是 9 * 9。首先设定两个 σ 值，值具体的设定可以参考第二部分借用 ppt 的图，这里默认是 180 和 0.2。在遍历图像的过程中先判断是不是彩色图，统计距离以及灰度高斯权重在求和，乘以灰度得到具体像素值赋给 Ans 即可。

四、实验环境及运行方法

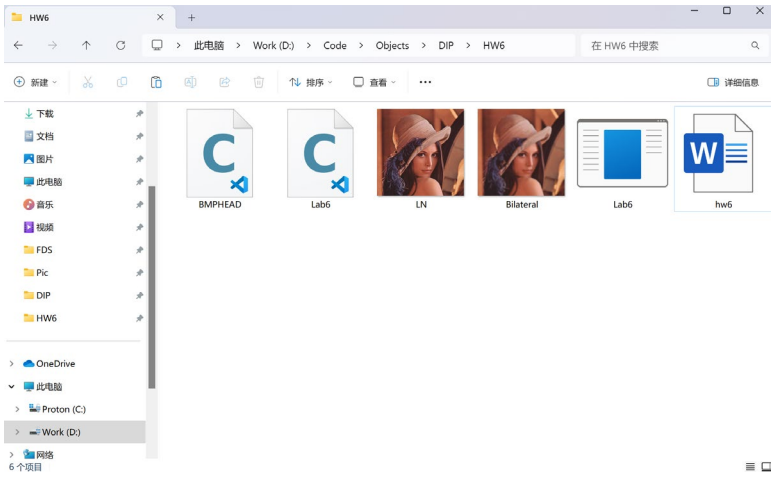
1. 实验环境

系统 Windows11 编译器 gcc 10.3.0x86_mingw32

2. 运行方法

在文件夹中，Lab6.c 为源文件，运行代码，如果出现“读取成功:)”说明读取图片文件成功，否则会出现“读取失败:(”.

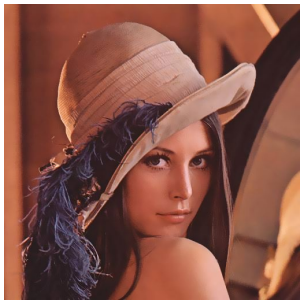
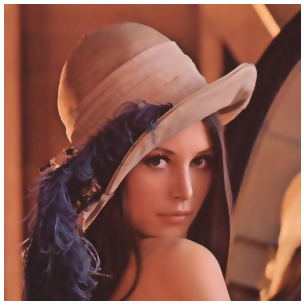
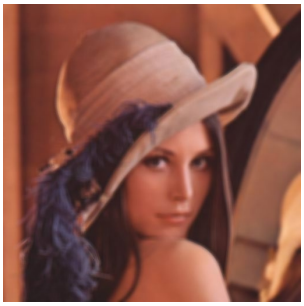
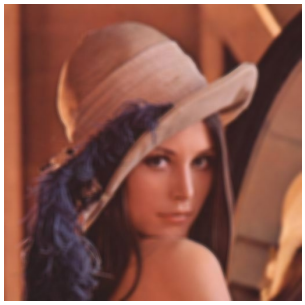
随后根据读取的图象是彩色或者灰度程序会输出双边滤波后的灰色或彩色图像（由输入决定）：Bilateral.bmp



五、实验结果展示

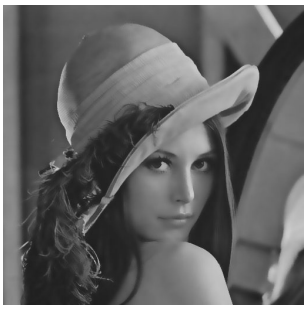
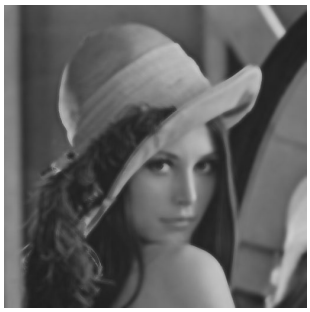
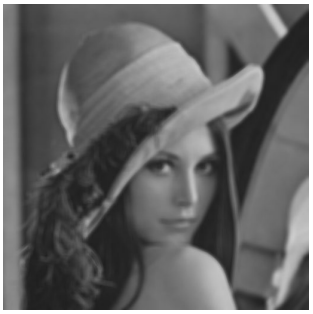
- 测试样例一：原图像我们使用莱娜，效果更明显一点

彩色原图

$\sigma_r = 10$　　$\sigma_r = 30$　　$\sigma_r = 90$　　$\sigma_r = 180$

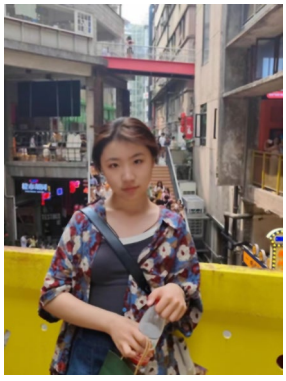
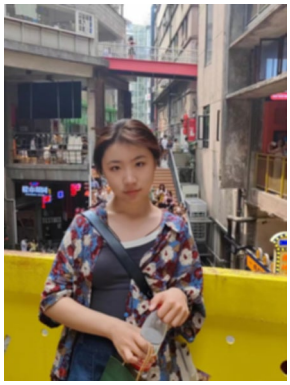
灰度原图

$\sigma_r = 10$　　$\sigma_r = 30$　　$\sigma_r = 90$　　$\sigma_r = 180$

- 测试样例二：人像

原图

$\sigma_r = 10$　　$\sigma_r = 100$　　$\sigma_r = 10000$

- 测试样例三：景象

原图



$\sigma_r = 10$　　$\sigma_r = 100$　　$\sigma_r = 10000$

六、心得体会

本次实验实现了图像暴力双边滤波，相比于均值滤波，双边滤波后的图像边缘保持的较完整，并且在参数合适的情况下不会使得图像边缘模糊，对于人像可以起到美颜的作用，使得皮肤光滑，通过莱娜这张图可以非常明显的看到脸上的噪点得到了淡化：


在调整参数的过程中，如果σ_r非常小，和原图没什么区别，只有适当的值才会取得较好的滤波效果，但是当值非常大的时候，图像会愈加模糊，效果会越来越差。而且当图片像素越来越大或者窗口大小较大时，暴力双边滤波会非常耗时，我在实验的过程中尝试了千万像素级的图像，并把窗口调到了 21，没有在有效时间内得到结果，因此没有在实验报告中给出

总结来看本次实验不是很难，而且非常好玩有趣。