专业：自动化
姓名：万晨阳
学号：3210105327
日期：2022/11/29

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：图像的双边滤波操作

一、实验目的和要求

- 了解双边滤波的原理，公式中各个参数的含义。
- 通过双边滤波操作进行图像处理。

二、实验内容

双边滤波的公式：

$$\overline{I}(p)=\frac{1}{W_p}\sum_{q\in S}G_{\sigma_s}(\|p-q\|)G_{\sigma_r}(|I(p)-I(q)|)I(q)$$

其中W_p为：

$$W_p=\sum_{q\in S}G_{\sigma_s}(\|p-q\|)G_{\sigma_r}(|I(p)-I(q)|)$$

双边滤波中有两个衡量图像信息的核心变量，如上式中的 Gσs 为空间域核，Gσr 为图像像素域核。空间域核其实就是二维高斯函数，可以把它视作高斯滤波，像素域核就是衡量像素变化剧烈程度的量。将这两个变量相乘，就可以得到他们俩共同作用的结果：在图像的平坦区域，像素值变化很小，对应的像素范围域权重接近于 1，此时空间域权重起主要作用，相当于进行高斯模糊；在图像的边缘区域，像素值变化很大，像素范围域权重变大，从而保持了边缘的信息。

其中，空间域核计算方法如下：

$$G_{\sigma s}(||p-q||)=e^{-\frac{(i-m)^2+(j-n)^2}{2{\sigma_s}^2}}$$

像素域核的计算方法如下：

$$G_{\sigma r}(|I(p)-I(q)|)=e^{-\frac{[I(i,j)-I(m,n)]^2}{2{\sigma_r}^2}}$$

在上面两个式子中，σs 与 σr 都是已知的，或者说是自己输入的预设值，而其他的 i，j，

m，n 都是需要我们在遍历中确定的值。其中（i，j）代表是窗口中心值，（m，n）代表的是滑动窗口中的某个值。

求解双边滤波的过程如下：

- 读取原始图像。
- 设定窗口大小，遍历整个图像，将窗口覆盖内的所有像素值与窗口中心像素比对，求取空间域核与图像像素域核大小，并将两者相乘作为该点的特征值。
- 将每一点的特征值与该点像素值乘积相加，除以所有特征值的和，得到的结果即作为中心点的像素值。

在求解过程中，需要注意的是，如果是 RGB 三通道的图像，每个点的像素值就需要将 RGB 三通道分开求解。

三、实验步骤与分析

双边滤波的实质还是某种意义上的卷积。所以我们对上次作业中实现的卷积函数进行一些小的修改即可得到双边滤波操作的实现。其实现函数定义如下：

```
void BilateralFiltering(BMP *pImg, CACHE *pCache, double sigma_s, double sigma_r)
```

其中 pImg 为指向自定义的 BMP 结构体的指针，pCache 为指向保存自定义的储存图片部分信息的结构体。而 sigma_s 与 sigma_r 与双边滤波处理中所需要的手动设定的参数。

该函数的实现逻辑如下：我们采用 7×7 的感受野（未进行 padding），通过嵌套循环分别在 RGB 三个通道上计算某一周围像素点与中心像素点在空间域和色彩域上的距离。最后通过双边滤波的计算公式得到中心像素点更新后的 RGB 值。实现如下：

```
1. void BilateralFiltering(BMP *pImg, CACHE *pCache, double sigma_s, double sigm
   a_r)
2. {
3.
4.     DWORD rows = pCache->ImgHeight;
5.
6.     DWORD cols = pCache->ImgWidth;
7.
8.     DWORD M = pCache->LineBytes, N = 3;
9.
10.    BYTE *Temp = (BYTE *)malloc(pCache->ImgSize * sizeof(BYTE));
11.
12.    memcpy((char *)Temp, (BYTE *)pImg->pImgInfo, pCache->ImgSize * sizeof(BYT
   E));
```

```
13.
14.     double new_R = 0, new_G = 0, new_B = 0;
15.
16.     double k;
17.
18.     for (int i = 3; i < rows - 3; i++)
19.     {
20.         for (int j = 3; j < cols - 3; j++)
21.         {
22.             double W_p_B, W_p_G, W_p_R, W_P_B, W_P_G, W_P_R;
23.
24.             W_p_B = W_p_G = W_p_R = W_P_B = W_P_G = W_P_R = 0;
25.
26.             for (int m = i - 3; m <= i + 3; m++)
27.             {
28.                 for (int n = j - 3; n <= j + 3; n++)
29.                 {
30.
31.                     double G_s = exp(dist(i, j, m, n) / (-
    2 * (sigma_s) * (sigma_s)));
32.
33.                     double G_r_B = exp((Temp[i * M + j * N] - Temp[m * M + n
    * N]) * (Temp[i * M + j * N] - Temp[m * M + n * N]) / (-
    2 * (sigma_r) * (sigma_r)));
34.
35.                     double G_r_G = exp((Temp[i * M + j * N + 1] - Temp[m * M
    + n * N + 1]) * (Temp[i * M + j * N + 1] - Temp[m * M + n * N + 1]) / (-
    2 * (sigma_r) * (sigma_r)));
36.
37.                     double G_r_R = exp((Temp[i * M + j * N + 2] - Temp[m * M
    + n * N + 2]) * (Temp[i * M + j * N + 2] - Temp[m * M + n * N + 2]) / (-
    2 * (sigma_r) * (sigma_r)));
38.
39.                     W_p_B += G_s * G_r_B;
40.
41.                     W_p_G += G_s * G_r_G;
42.
43.                     W_p_R += G_s * G_r_R;
44.
45.                     W_P_B += G_s * G_r_B * Temp[m * M + n * N];
46.
47.                     W_P_G += G_s * G_r_G * Temp[m * M + n * N + 1];
48.
49.                     W_P_R += G_s * G_r_R * Temp[m * M + n * N + 2];
```

```
                }
            }

            new_B = W_P_B / W_p_B;

            new_G = W_P_G / W_p_G;

            new_R = W_P_R / W_p_R;

            pImg->pImgInfo[i * M + j * N] = (BYTE)new_B;

            pImg->pImgInfo[i * M + j * N + 1] = (BYTE)new_G;

            pImg->pImgInfo[i * M + j * N + 2] = (BYTE)new_R;

            new_R = new_G = new_B = 0;
        }
    }
}
```

其中 *dist* 是计算欧几里得距离平方的函数，实现如下：

```
double dist(double x1, double y1, double x2, double y2)
{
    return ((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
}
```

四、实验环境及运行方法

- 实验环境：

  使用 C 语言编写；编译器：gcc.exe 11.2.0

- 运行方法：

  img 文件夹中包含 test.bmp（24 位位图），是生成后续实验结果图像的原图像。

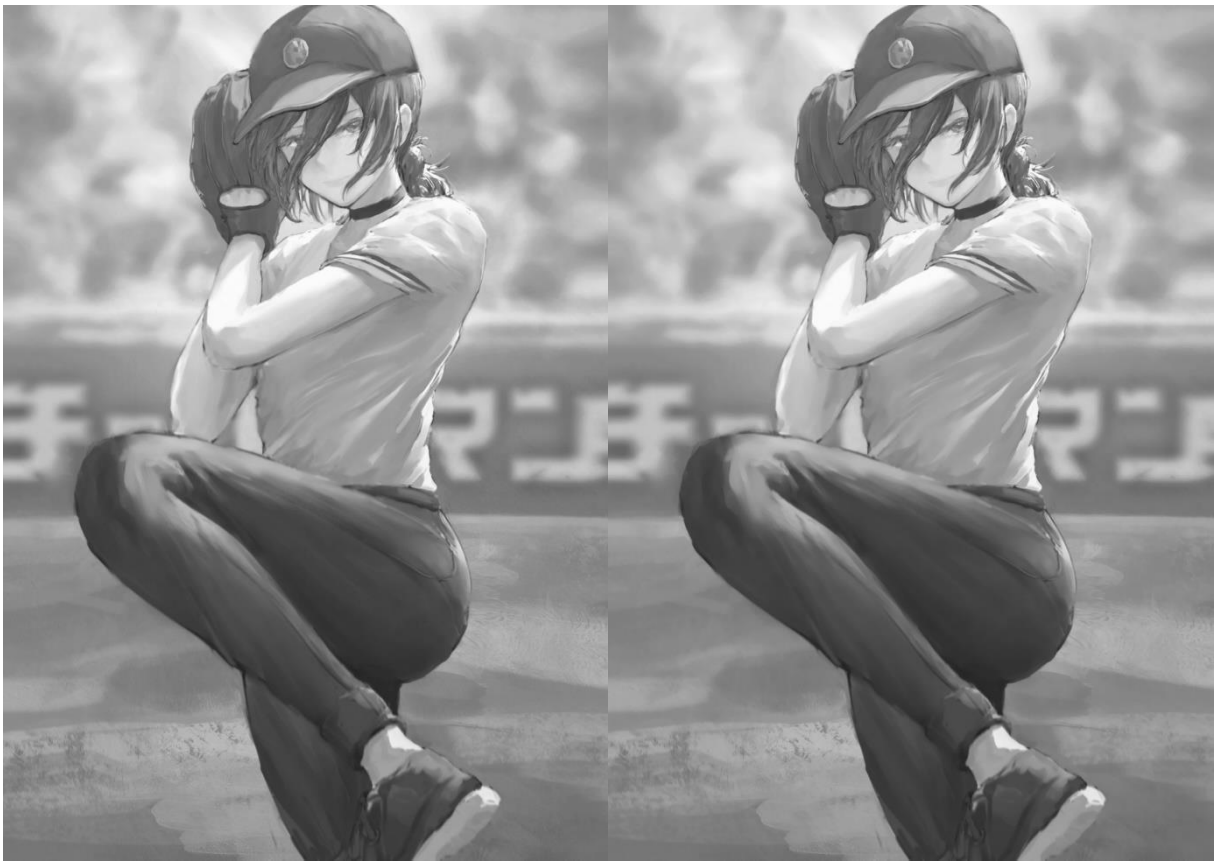
  运行 bilateral_filtering.exe 之后，请依照提示输入 σ_s 和 σ_r 的参数值。结果图像 bilateral_filtering.bmp 在 img 文件夹中生成。

  运行 minus.exe 可以得到原图片与滤波操作后图片相减的结果。

五、实验结果展示

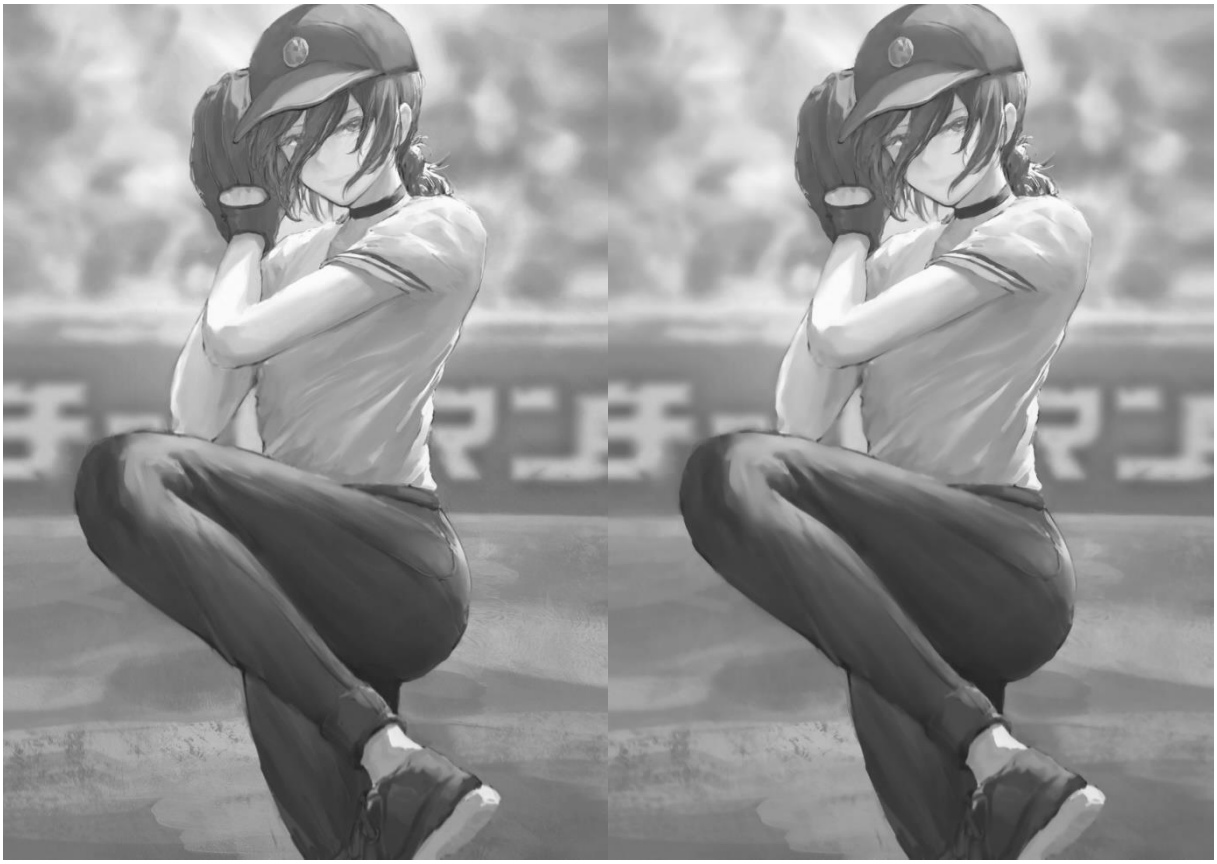
以下展示的图像中左边为原图像，右边为采用双边滤波处理后的图片。

1. σs = 10 , σr = 3

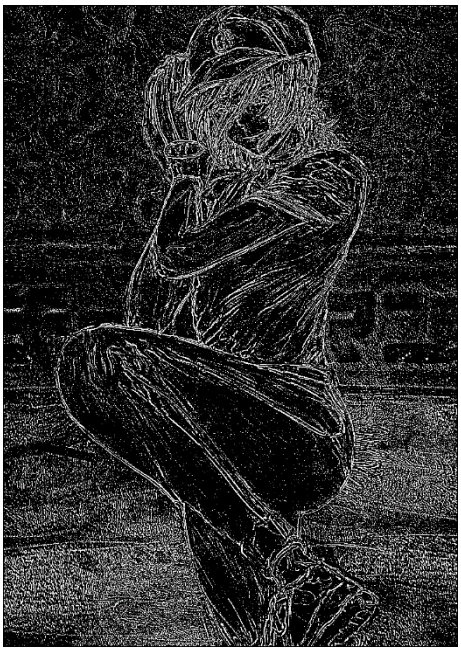


相减结果：

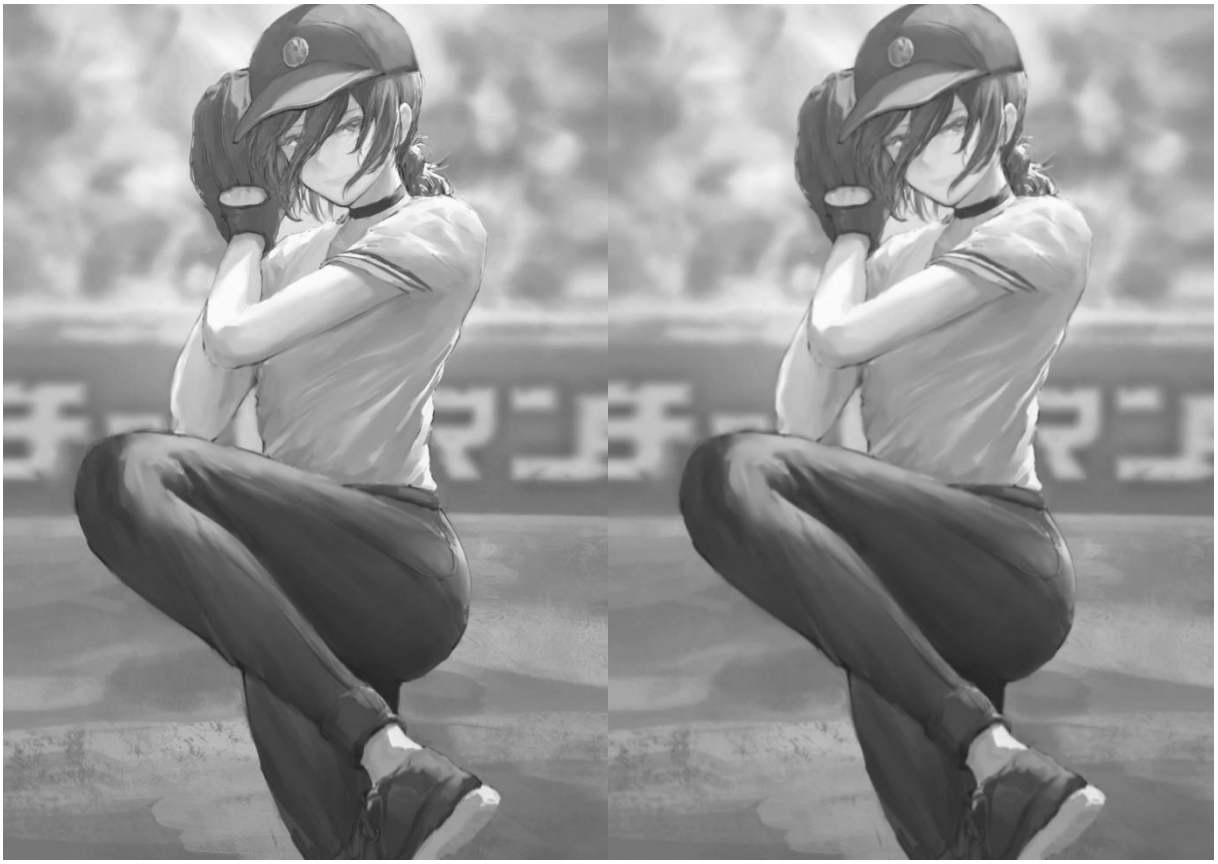
2. σs = 10 , σr = 10



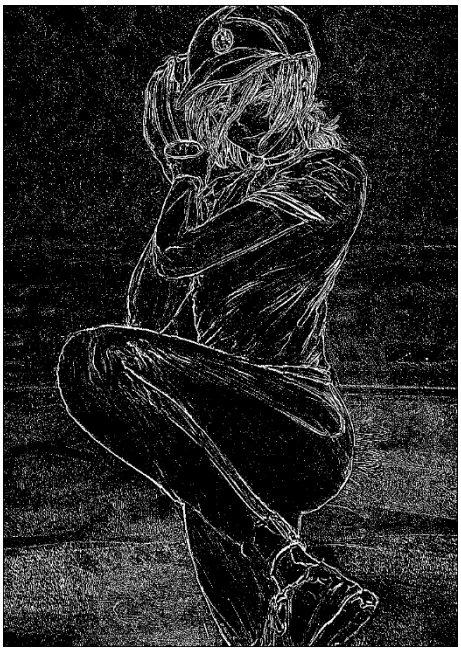
相减结果：



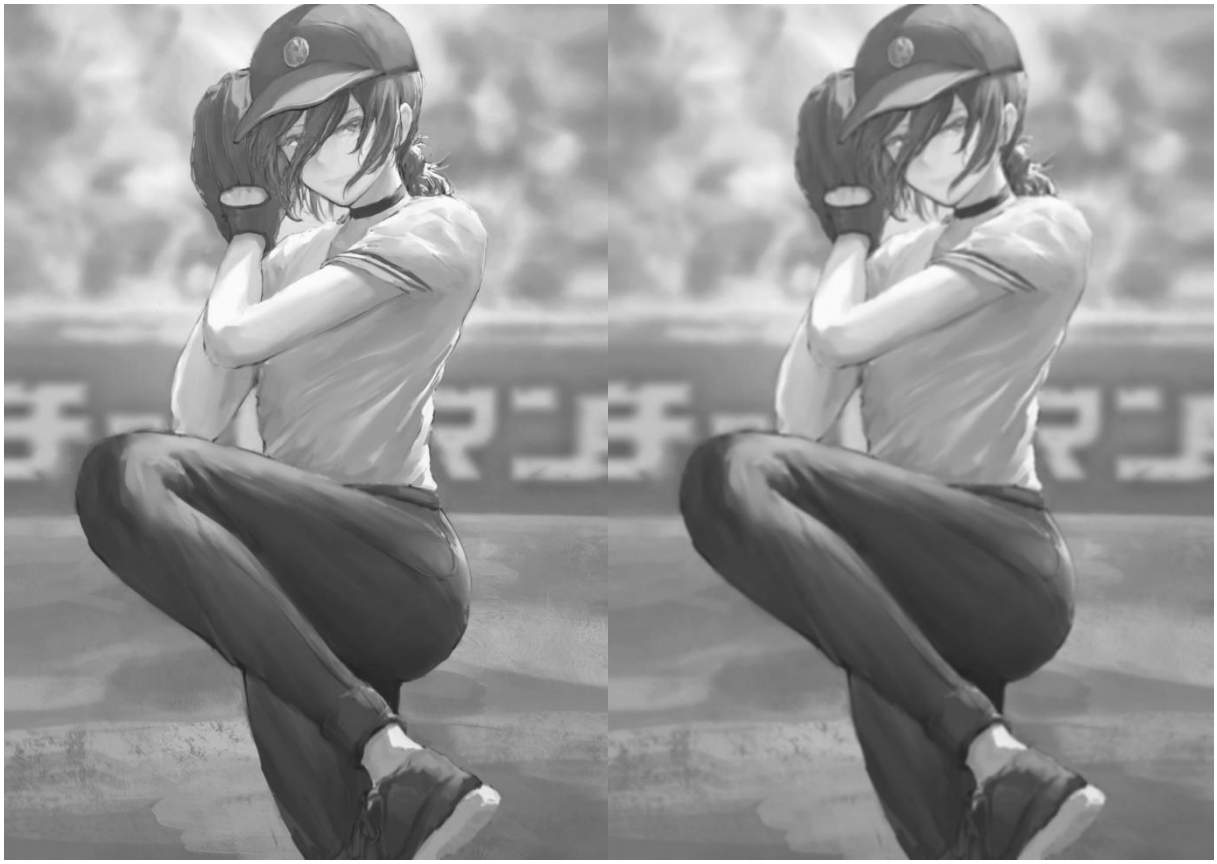
3. σs = 1 , σr = 100



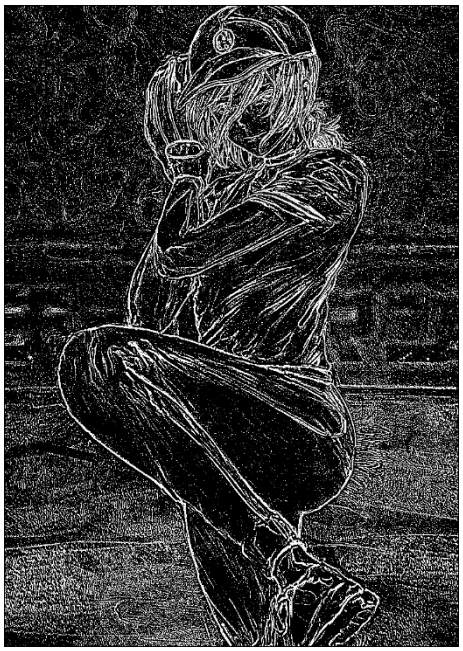
相减结果：



4. σs = 1 , σr = 10000



相减结果：



六、心得体会

双边滤波是非常常用的一种滤波，在前面我们已经了解了三种滤波：均值滤波、高斯滤波与中值滤波。这三种滤波都能够在一定程度上消除噪声，但是其作用范围有限，只能针对特定种类的噪声。例如高斯滤波针对高斯噪声效果较好，而中值滤波针对椒盐噪声的效果较好。而且，这三种滤波对图像的边缘信息都会有不同程度的损坏，究其原因是没有考虑到图像边缘的信息。因而，我们引入双边滤波，其在利用高斯滤波去噪的同时，能够较好地保存图片的边缘信息。归根结底，其优势在于公式中既考虑了像素域的距离又考虑了空间域的距离，这是前面几种滤波所不具备的。当然其劣势就在于计算速度更慢。