专业：计算机科学与技术
姓名：秦嘉俊
学号：3210106182
日期：2022 年 12 月 2 日

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

1. 暴力实现双边滤波

二、 实验内容和原理

2.1 高斯滤波

高斯滤波是最常用的图像去噪方法之一，它能很好地滤除掉图像中随机出现的高斯噪声，但是在之前的博客中提到过，高斯滤波是一种低通滤波（有兴趣的点击这里，查看之前的博客），它在滤除图像中噪声信号的同时，也会对图像中的边缘信息进行平滑，表现出来的结果就是图像变得模糊。

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

这里的 σ 是我们选取窗口的大小。如何设置 σ? 根据经验，通常的策略是是设置为图像大小的一个比例，如 2σ 越大，图像越平滑，趋于无穷大时，每个权重都一样，类似均值滤波; σ 越小，中心点权重越大，周围点权重越小，对图像的滤波作用越小，趋于零时，输出等同于原图。

可以起到平滑效果，但会使图像模糊，因为只考虑了距离因素。这种只关注距离的思想在某些情况下是可行的，例如在平坦的区域，距离越近的区域其像素分布也越相近，自然地，这些点的像素值对滤波中心点的像素值更有参考价值。但是在像素值出现跃变的边缘区域，这种方法会适得其反，损失掉有用的边缘信息。

(a) 原图　　(b) 高斯滤波处理后的图像

图 1: 高斯滤波

2.2 双边滤波

正如前文所说，高斯滤波不具有保边性，它会使边缘模糊。此时就出现了一类算法——边缘保护滤波方法，双边滤波就是最常用的边缘保护滤波方法。

双边滤波的核心思想在于，每个样本都被周围的某种加权平均替代（同高斯滤波），但这个权重既要反映距离中心像素的远近，又要反映像素值和中心像素值的相似度。因此我们可以得到双边滤波的公式：

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

其中

- $\frac{1}{W_p}$ 是归一化因子
- $G_{\sigma_s}(||p-q||)$ 表示空间 (spatial) 的权重，和高斯滤波中相同 (σ_s 表示核的空间范围)
- $G_{\sigma_r}(|I_p - I_q|)$ 表示灰度 (range) 的权重 (注意这里只是一范式而非二范式，因为灰度只是标量)(σ_r 表示灰度的范围)

如何设置这些参数？对于 σ_s 我们的思路和高斯滤波相同, σ_s 越大，图像越平滑，趋于无穷大时，每个权重都一样，类似均值滤波; σ_s 越小，中心点权重越大，周围点权重越小，对图像的滤波作用越小，趋于零时，输出等同于原图。

对于 σ_r, σ_r 越大，边缘越模糊，极限情况为 σ_r 无穷大，值域系数近似相等（忽略常数时，将近为 $e^0 = 1$），与高斯模板（空间域模板）相乘后可认为等效于高斯滤波; σ_r 越小，边缘越清晰，极限情况为 σ_r 无限接近 0，值域系数除了中心位置，其他近似为 0（接近 $e^\infty = 0$），与高斯模板（空间域模板）相乘进行滤波的结果等效于源图像。

三、 实验步骤与分析

其中，图像信息头、图像文件头等结构体的定义，以及图像的输出基本同前，这里不再重复。本次实验我们的程序可以输入一张 24 位彩色 BMP 图像或者一张 8 位 BMP 灰度图像。

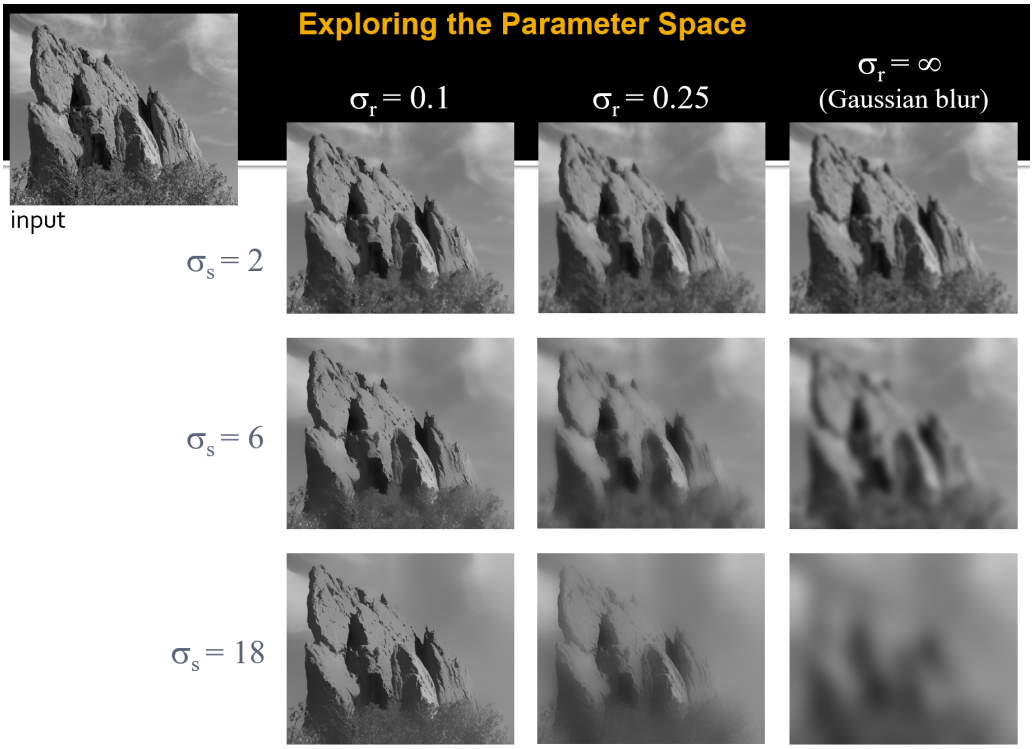


图 2: 双边滤波及其参数调整

3.1 BMP 文件的读入

```
int main()
{
    BMPFILE a, b;
    FILE *fp;
    /* 读入 */
    fp = fopen("LN.bmp", "rb"); // rb 打开一个二进制文件
    if (!fp) {
        printf("BMP Image Not Found!\n");
        exit(0);
    }
    printf("Successfully open the image\n");
    fread(&(a.bmfh), sizeof(BITMAPFILEHEADER), 1, fp);
    fread(&(a.bmih), sizeof(BITMAPINFOHEADER), 1, fp);
    ImageHeight = a.bmih.biHeight;
    ImageWidth = a.bmih.biWidth;

    if(! a.bmih.biSizeImage) // 注意 biSizeImage 可能为 0 !
        a.bmih.biSizeImage = a.bmfh.bfSize - a.bmfh.OffBits;
    ImageSize = a.bmih.biSizeImage; // 所有像素所占的字节数.
    row_byte = (a.bmih.biBitCount / 8 * ImageWidth + 3) / 4 * 4; // 注意字节必须是 4 的整数倍
    if (a.bmih.biBitCount == 8) // 如果是 8 位图片的话，我们需要加上调色板
        for (int i = 0; i < 256; i++)
            a.aColors[i].rgbBlue = a.aColors[i].rgbGreen = a.aColors[i].rgbRed = i,
            b.aColors[i].rgbBlue = b.aColors[i].rgbGreen = b.aColors[i].rgbRed = i;
    a.aBitmapBits = (BYTE *)malloc(sizeof(BYTE) * ImageSize);
    fread(a.aBitmapBits, ImageSize * sizeof(BYTE), 1, fp);
    fclose(fp);
    /* 开始操作 */
```

```
    memcpy(&(b.bmfh), &(a.bmfh), sizeof(BITMAPFILEHEADER));
    memcpy(&(b.bmih), &(a.bmih), sizeof(BITMAPINFOHEADER));
    b.aBitmapBits = (BYTE *)malloc(sizeof(BYTE) * ImageSize);
    Bilateral(&a, &b, 9);
    Print(&b, "Bilateral.bmp");
}
```

在 main 中我们的结构体 a 是读入的 BMP 文件，b 用来存储图像变换后的 BMP 文件。和以往略有不同的点在于，我们如果读入是灰度图，那么在图像读入时就为其加上调色板，以便后面操作。

读入结束后，我们对这张图像进行双边滤波操作。

3.2 双边滤波

```
int Get_Position(int x, int y, int bytes, int pixels)
{
    if (x < 0) x = 0; /* padding */
    if (y < 0) y = 0; /* padding */
    if (x >= ImageHeight) x = ImageHeight - 1; /* padding */
    if (y >= ImageWidth) y = ImageWidth - 1; /* padding */
    return x * bytes + y * pixels;
}
int Update(int x)
{
    if (x < 0) return 0;
    if (x > 255) return 255;
    return x;
}

/* 返回 e^{-x/2sigma^2} */
double Gauss(double x_square, double sigma)
{
    return exp(- x_square / (2 * sigma * sigma));
}
/* 双边滤波操作 */
void Bilateral(BMPFILE *a, BMPFILE *ans, int scale)
{
    int i, j, k1, k2;
    int length = scale / 2; /* 对于中心点为 (i,j) 的窗口，
    窗口范围是 i:[i-scale/2,i+scale/2] j:[j-scale/2,j+scale/2 */
    double tmpR, tmpG, tmpB, x, y, G_s, G_rR, G_rG, G_rB, nowR, nowG, nowB;
    double sigma_s = 0.02 * ((ImageWidth + ImageHeight) / 2);
    double sigma_r = 15; /* 可调参 */
    for (i = 0; i < ImageHeight; i++)
        for (j = 0; j < ImageWidth; j++)
            if (a->bmih.biBitCount == 24) { /* 判断是否为彩色图 */
            double sumR = 0.0, sumG = 0.0, sumB = 0.0;
            double W_R = 0, W_G = 0, W_B = 0;
            int pos = Get_Position(i, j, row_byte, 3);
            nowB = a->aBitmapBits[pos];
            nowG = a->aBitmapBits[pos+1];
            nowR = a->aBitmapBits[pos+2];
            for (k1 = i - length; k1 <= i + length; k1++)
                for (k2 = j - length; k2 <= j + length; k2++) {
```

```
                int now = Get_Position(k1, k2, row_byte, 3);
                tmpB = a->aBitmapBits[now];
                tmpG = a->aBitmapBits[now+1];
                tmpR = a->aBitmapBits[now+2];
                G_s = Gauss((i - k1) * (i - k1) + (j - k2) * (j - k2), sigma_s); /* 空间，二维距离 */
                G_rR = Gauss((tmpR - nowR) * (tmpR - nowR), sigma_r); /* 灰度，一维标量 */
                G_rG = Gauss((tmpG - nowG) * (tmpG - nowG), sigma_r);
                G_rB = Gauss((tmpB - nowB) * (tmpB - nowB), sigma_r);
                W_R += G_rR * G_s; W_G += G_rG * G_s; W_B += G_rB * G_s; /* 计算归一化因子 */
                sumR += G_rR * G_s * tmpR; sumG += G_rG * G_s * tmpG; sumB += G_rB * G_s * tmpB;
            }
        ans->aBitmapBits[pos] = sumB / W_B;
        ans->aBitmapBits[pos+1] = sumG / W_G;
        ans->aBitmapBits[pos+2] = sumR / W_R;
    }
    else if (a->bmih.biBitCount == 8){ /* 判断是否为灰度图*/
        double sumB = 0.0;
        double W_B = 0;
        int pos = Get_Position(i, j, row_byte, 1);
        nowB = a->aBitmapBits[pos];
        for (k1 = i - length; k1 <= i + length; k1++)
            for (k2 = j - length; k2 <= j + length; k2++) {
                int now = Get_Position(k1, k2, row_byte, 1);
                tmpB = a->aBitmapBits[now];
                G_s = Gauss((i - k1) * (i - k1) + (j - k2) * (j - k2), sigma_s); /* 空间，二维距离 */
                G_rB = Gauss((tmpB - nowB) * (tmpB - nowB), sigma_r); /* 灰度，一维标量 */
                W_B += G_rB * G_s;           /* 计算归一化因子 */
                sumB += G_rB * G_s * tmpB;
            }
        ans->aBitmapBits[pos] = sumB / W_B;
    }
}
```

这里我们首先预设好参数 σ_s, σ_r. 需要注意的, σ_r 的设置没有太多可以参考的经验，因此我们选择不断调节参数以期取得较好的效果。随后判断是否为彩色图，如果是的话则遍历窗口内的像素（窗口大小默认 9×9）统计每个像素点的距离、灰度高斯权重并相加，得到归一化因子。同时我们将这个权重乘上灰度，得到这个像素点对中心像素的贡献。最后将算出的加权平均值赋值给中心像素即可。灰度图的处理类似，从 RGB 通道变为灰度单通道即可。

此外，原理上高斯函数应该有系数 $\frac{1}{\sqrt{2\pi}\sigma}$, 但这里因为每个像素点的距离、灰度高斯乘积对应的系数一致，而且最后要做归一化操作，因此我们可以省去系数。

具体过程可以参考代码及注释。

四、 实验环境及运行方法

4.1 实验环境

Windows 10 系统

gcc 10.3.0 (tdm64-1) x86_64-w64-mingw32

4.2 运行方法

源文件为 lab6.c, 在源文件里有我们的样例输入, 一张 24 位彩色 BMP 图像文件 (LN.bmp)(或者一张 8 位灰度图像 LN_gray.bmp), 使用 VSCode 打开这个文件夹，并选中 lab6.c 点击 Run Code 即可开始运行。当终端出现“Successfully open the image”时说明我们成功打开了图像，否则会输出“BMP Image Not Found!”.

成功运行后会得到 1 个新图像，Bilateral.bmp，是双边滤波的结果。（代码中我们采用的是默认参数, 即 σ_s 是长宽均值的 0.02 倍, σ_r 默认 15, 窗口大小默认 9×9）

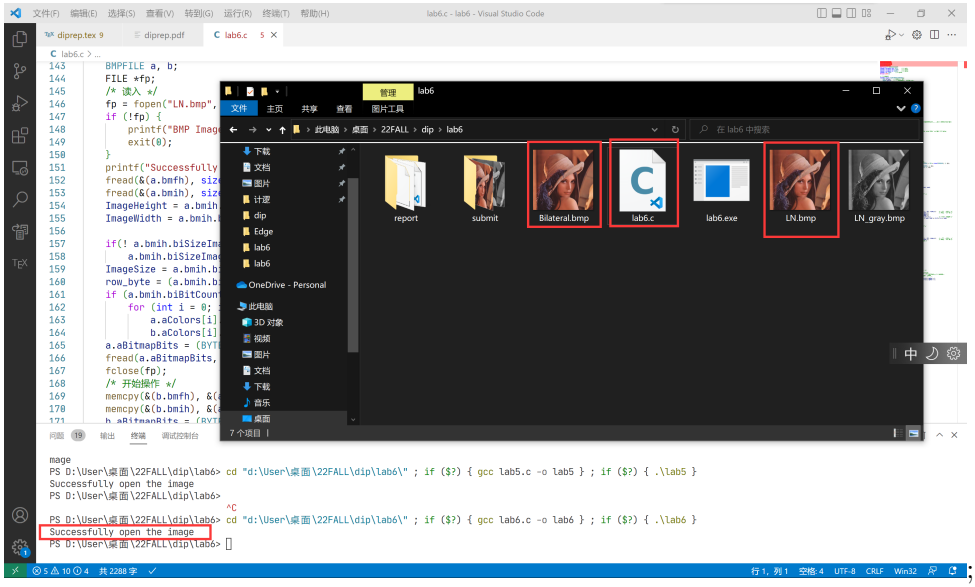


图 3: 运行

五、 实验结果展示

我们依然用经典的莱纳图作为测试用例

(a) 输入图像 (LN.bmp)

(b) $\sigma_r = 5$ 的双边滤波 (Bilateral5.bmp)

(c) $\sigma_r = 15$ 的双边滤波 (Bilateral15.bmp)

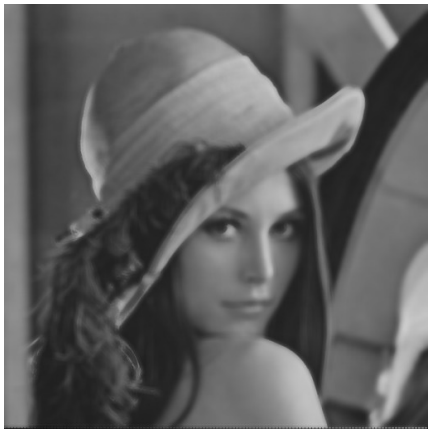
(d) $\sigma_r = 100$ 的双边滤波 (Bilateral100.bmp)

对于灰度莱纳图也有类似的结果

(e) 输入图像 (LN_gray.bmp)

(f) $\sigma_r = 5$ 的双边滤波 (Bilateralgray5.bmp)

(g) $\sigma_r = 15$ 的双边滤波 (Bilateralgray15.bmp)



(h) $\sigma_r = 100$ 的双边滤波 (Bilateralgray100.bmp)

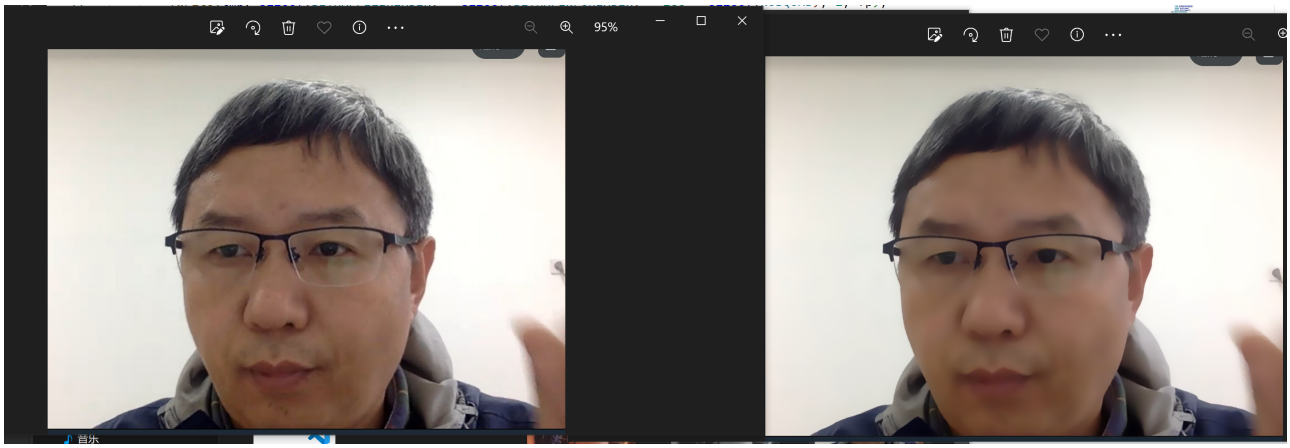
此外，我们还测试了 PPT 上的用例。值得一提的是，之前的参数设置在这张图表现并不好，于是我反复地调节参数，最后选择了 25×25 的窗口大小，$\sigma_s = 0.02 * ((ImageWidth + ImageHeight)/2)$, $\sigma_r = 20$ 的搭配，效果如图。

(i) 输入图像 (3.bmp)

(j) 双边滤波后图像 (Bilateral.bmp)

六、 心得体会

可以看到，相比之前的均值滤波（高斯滤波）双边滤波在保边性这方面有显著的效果，一般来说不会让图像变得过于模糊。同时对于人像，放大可以清晰看到经过双边滤波后明显皮肤更加光滑，如胡子、痣等也得到淡化，相当于是给图片做了“磨皮效果”。这里我们斗胆使用网课期间宋老师的图片作为尝试。（这里使用的窗口大小为 25×25, $\sigma_s = 0.01 * ((ImageWidth + ImageHeight)/2)$, $\sigma_r = 20$）[1]



可以看到磨皮效果非常明显，老师看上去更青春更有活力了！但磨皮效果好的同时，也必然会使图像变得模糊，这其中需要做一个 tradeoff. 个人猜测或许可以利用其他手段（如人工智能中的一些方法）在磨皮后对人脸做图像恢复，这样既能有好的磨皮效果，也不会使图像变得模糊。

此外参数的调节也是一个学问。我们可以对比实验结果的图像，发现 σ_r 越大，磨皮降噪效果越好，但也正如原理部分所说，当 σ_r 较小 (< 1) 时，输出的图像和原图几乎没有什么变化，而当 σ_r 较大时图像会更加模糊，效果近似于均值滤波。

同时本次实验中是暴力实现双边滤波的方法，并没有进行优化，这导致一旦窗口较大运行开销将非常高昂，因此我们也没有尝试过大的窗口。未来可以使用上课讲的 paper 中的优化，提高速度的同时也能更多参数的尝试。

总的来说，本次实验不算难，而且十分有趣，期待下次实验！

[1] 左为原图，右为双边滤波后的图像。限于种种原因原图和处理后的图像未在提交的压缩包中放出