

浙江大学实验报告

专业：计算机科学与技术
姓名：沈一芑
学号：322010827
日期：2023/12/20

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

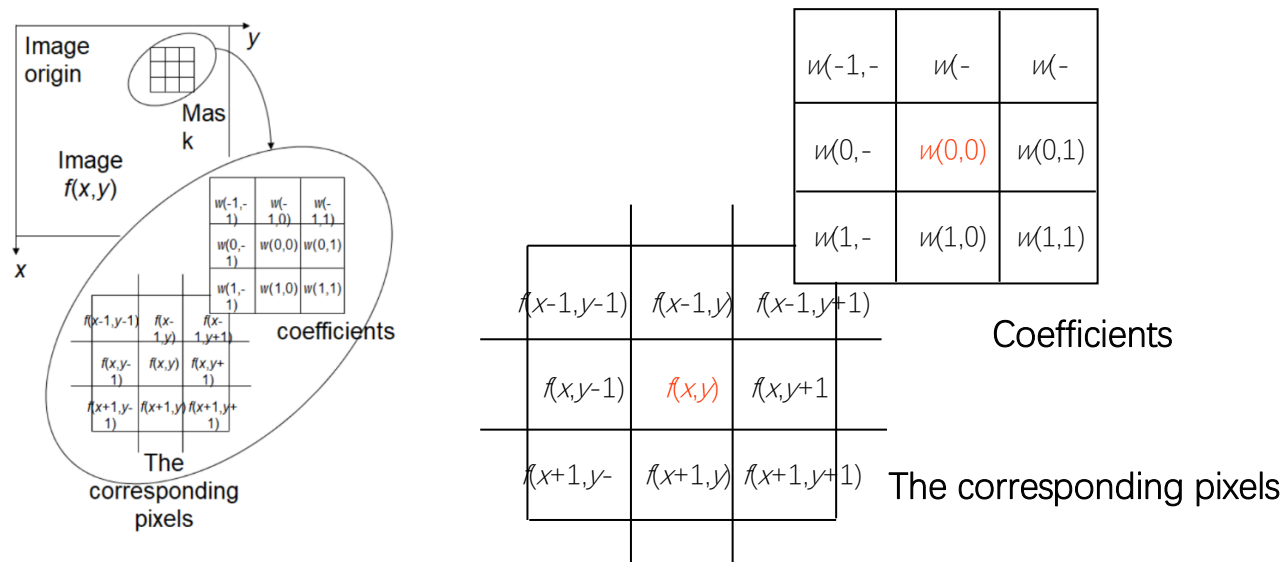
1. 实验目的和要求

- 暴力实现双边滤波

2. 实验内容和原理

2.1. 滤波

滤波器是一个大小为 $M \times N$ 的窗口。对于不同的滤波方式，令窗口中的元素与原图像的处于窗口内的像素进行相对应的运算，得到的结果作为新图像的一个像素（一般为滤波窗口中心像素点在原图像中重合位置处的像素）。当窗口滑过原图像并完成上述运算之后，就能够得到一幅新图像。



2.2. 高斯滤波

高斯滤波 (Gaussian filter) 是一种常见的滤波手段。此处的高斯滤波指高斯模糊 (Gaussian Blur)，是一种低通滤波，通过过滤调图像高频成分（细节部分），保留图像低频成分（平滑区域），以此对图像进行处理。结果表现为过滤掉噪声的同时也平滑化图像物体边缘，导致图像模糊。这种滤波方法对抑制高斯噪声（服从正态分布的噪声）非常有效。高斯滤波的公式如下：

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

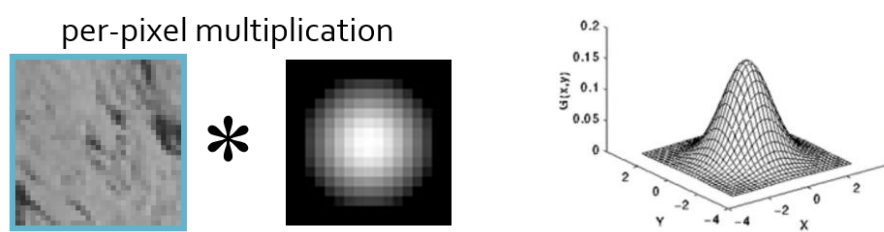
其中 G_{σ} 是标准差为 σ 的二维高斯核，定义如下：

$$G_{\sigma} = \frac{1}{2\pi\sigma} e^{-(x^2 + y^2)/2\sigma^2}$$

高斯函数具有如下特性：

- 旋转对称性，即滤波器在各个方向上的平滑程度相同。高斯滤波器在对像素点做卷积运算时不会偏向任一方向。
- 高斯函数的傅立叶变换频谱是单瓣的。这意味着平滑图像在不被不必要的高频信号（噪声）所污染的同时保留了大部分基本信号（图像基本特征）。
- 高斯滤波器宽度（平滑程度的决定因素）是标准差 σ 表征的；在一定范围内， σ 越大，高斯滤波器的频带就越宽，平滑程度就越好。而当 σ 趋于无穷大时，滤波器内每个点权重均相同，此时平滑效果类似于均值滤波。而 σ 越小，滤波器中心点的权重就越大，周围点权重越小，平滑滤波作用越小；当 σ 趋于零时，输出也趋于原图。

下图展示了高斯滤波器对图像每一个像素点的操作，以及滤波器的权重体现：



2.3. 双边滤波

一般图像具有两个主要特征：1. 空间域 S ，即图像中可能的位置集合。这与分辨率有关，即图像的行数和列数。2. 强度域 R ，即可能的像素值集合。不过用于表示像素值的位数可能会有所不同。高斯滤波因为只考虑到了图像空间域的特征而具有局限性。在图像边缘处，显然周围各点的贡献权重不能仅仅按照欧氏空间距离来分配。此时便引入了双边滤波，旨在通过同时关注空间域和强度域以达成消除噪声的同时保留图像物体边缘的效果。

对于空间域，我们可以采用高斯滤波器。而对于强度域，我们希望保留图像边缘；由于在图像边缘部分像素值变化较剧烈，在图像非边缘区域像素值变换较平坦，我们可以用构造一个衡量图像像素变换剧烈程度的变量。故而模仿高斯滤波器，双边滤波构造了 G_{σ_R} 核，同时考虑 G_{σ_S} 和 G_{σ_R} 。故双边滤波的公式如下：

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

其中 W_p 可表示为：

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

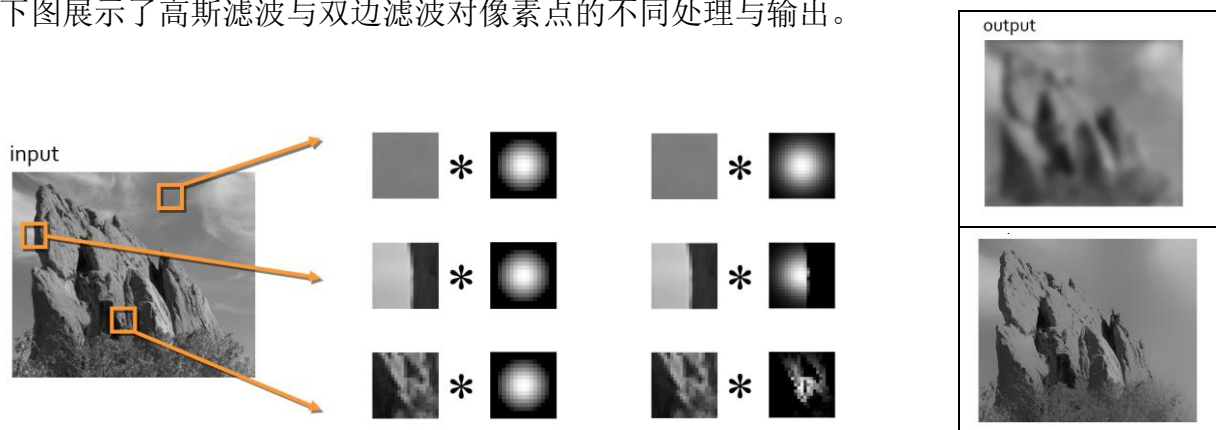
其中， G_{σ_S} 和 G_{σ_R} 分别表示为：

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

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

通过观察可以发现，当像素变化较为平坦时， G_{σ_R} 结果为 1，不改变权值。而当像素变化较为剧烈时， G_{σ_R} 趋于 0，乘积运算后 G_{σ_S} 的影响也相应趋于 0。

下图展示了高斯滤波与双边滤波对像素点的不同处理与输出。



3. 实验步骤与分析

3.1. 准备工作

- 定义结构体并声明变量如下；每个变量对应一张 BMP 图片：

```
• typedef struct{
•     BMFH bmfh;
•     BMIH bmih;
•     RGBQUAD rgbquad[256];
•     BYTE *data;
•     int LineBytes, WidthBMP, HeightBMP, SizeImageBMP;
•     //LineBytes is to compliment the width given that bytes mod 4 == 0
•     //size of image part, instead of whole bmp
• }IMAGE;
• IMAGE input, gray, gaussian_filtering, bilateral_filtering;
```

- 定义高斯滤波器如下：

```
• typedef struct{
•     int size=3;
•     double sigma=0.5;
•     double* coef;
•     Coord* pos;
• }Filter_Gaussian;
```

其中，size 为窗口宽度，pos 坐标结构体数组指针指向窗口中各点到中心点的相对位置，coef 指针指向高斯滤波的窗口权重（通过预处理减少重复计算）。

- 定义双边滤波器如下（嵌套高斯滤波器，借用 pos，使用 coef 与 sigma）：

```
• typedef struct{
•     int size = 3;
•     double sigma_s, sigma_r;
•     Filter_Gaussian F_Gaussian;
• }Filter_Bilateral;
```

其中，size 为窗口宽度，sigma_s 和 sigma_r 分别表征了公式中的 σ_s 和 σ_r 两个标准差。

3.2. 双边滤波

首先，初始化滤波器，对高斯滤波的权重和高斯/双边滤波器与中心像素点的相对位置做预处理计算。其次，遍历图像每一个像素点，用双边滤波器进行滤波操作。

对每一个像素点，灰度图只考虑 Y 值，彩色图则对 R, G, B 三个通道分别进行双边滤波。进行滤波时要注意归一化，归一化通过 WY/B/G/R 的累加实现。对于图像边缘的点，则只考虑滤波器与图像内部重合的部分。

```
1. void Do_bilateral_filtering(IMAGE *input, IMAGE *output){
2.
3.     if(input->bmi.bmiBitCount!=8) CopyColor(input, output);
4.     else CopyGray(input, output);
5.
6.     Filter_Bilateral BilateralFilter;
7.     BilateralFilter.size=7;
8.     BilateralFilter.sigma_s=3;
9.     //0.02*sqrt(input->HeightBMP*input->HeightBMP+input->WidthBMP*input->WidthBMP);
10.    BilateralFilter.sigma_r=10;
11.    BilateralFilter.F_Gaussian.coef = new
        double[BilateralFilter.size*BilateralFilter.size];
12.    BilateralFilter.F_Gaussian.pos = new
        Coord[BilateralFilter.size*BilateralFilter.size];
13.    int midx, midy;
14.    midx = midy = BilateralFilter.size/2;
15.    for(int i=0;i<BilateralFilter.size;i++){
16.        for(int j=0;j<BilateralFilter.size;j++){
17.            double expo=((i-midx)*(i-midx)+(j-midy)*(j-midy));
18.            expo=expo*(-1)/2/BilateralFilter.sigma_s/BilateralFilter.sigma_s;
19.            BilateralFilter.F_Gaussian.coef[i*BilateralFilter.size+j]=exp(expo);
20. // different from the simple gaussian filtering by
21. // ignoring the coef /2.0/M_PI/BilateralFilter.sigma_s/BilateralFilter.sigma_s;
22.        }
23.    for(int i=0;i<BilateralFilter.size;i++){
24.        for(int j=0;j<BilateralFilter.size;j++){
25.            BilateralFilter.F_Gaussian.pos[i*BilateralFilter.size+j].x = i-
                BilateralFilter.size/2;
26.            BilateralFilter.F_Gaussian.pos[i*BilateralFilter.size+j].y = j-
                BilateralFilter.size/2;
27.        }
28.    }
```

```

29.     for(int i=0;i<output->HeightBMP;i++){
30.         for(int j=0;j<output->WidthBMP;j++){
31.             int tempx, tempy;
32.             double ori_B, ori_G, ori_R, ori_Y;
33.
34.             if(input->bmih.biBitCount==8){
35.                 ori_Y=input->data[i*output->LineBytes+j];
36.             }else{
37.                 ori_B=input->data[i*output->LineBytes+j*3+0];
38.                 ori_G=input->data[i*output->LineBytes+j*3+1];
39.                 ori_R=input->data[i*output->LineBytes+j*3+2];
40.             }
41.
42.             double nB=0, nG=0, nR=0, nY=0;
43.             double gB=0, gG=0, gR=0, gY=0;
44.             double IB=0, IG=0, IR=0, IY=0;
45.             double WB=0, WG=0, WR=0, WY=0;
46.
47.             for(int k=0;k<BilateralFilter.size*BilateralFilter.size;k++){
48.                 tempx=i+BilateralFilter.F_Gaussian.pos[k].x;
49.                 tempy=j+BilateralFilter.F_Gaussian.pos[k].y;
50.                 if(tempx<0||tempy<0||tempx>=output->HeightBMP||tempy>=output->WidthBMP) continue;
51.                 if(input->bmih.biBitCount==8){
52.                     IY=input->data[tempx*output->LineBytes+tempy];
53.                     gY=calc_exp(IY, ori_Y, BilateralFilter.sigma_r);
54.                     WY+=gY*BilateralFilter.F_Gaussian.coef[k];
55.                     nY+=gY*BilateralFilter.F_Gaussian.coef[k]*IY;
56.                 }else{
57.                     //cout << GaussianFilter.coef[GaussianFilter.filter_num][k] << endl;
58.                     IB=input->data[tempx*output->LineBytes+tempy*3+0];
59.                     gB=calc_exp(IB, ori_B, BilateralFilter.sigma_r);
60.                     WB+=gB*BilateralFilter.F_Gaussian.coef[k];
61.                     nB+=gB*BilateralFilter.F_Gaussian.coef[k]*IB;
62.
63.                     IG=input->data[tempx*output->LineBytes+tempy*3+1];
64.                     gG=calc_exp(IG, ori_G, BilateralFilter.sigma_r);
65.                     WG+=gG*BilateralFilter.F_Gaussian.coef[k];
66.                     nG+=gG*BilateralFilter.F_Gaussian.coef[k]*IG;
67.
68.                     IR=input->data[tempx*output->LineBytes+tempy*3+2];
69.                     gR=calc_exp(IR, ori_R, BilateralFilter.sigma_r);
70.                     WR+=gR*BilateralFilter.F_Gaussian.coef[k];
71.                     nR+=gR*BilateralFilter.F_Gaussian.coef[k]*IR;

```

```
72.         }
73.     }
74.
75.     if(input->bmih.biBitCount==8)
76.         nY=nY/WY;
77.     else{
78.         nB=nB/WB;
79.         nG=nG/WG;
80.         nR=nR/WR;
81.     }
82.
83.     //if(Y<0)   Y*=-1;   if(B<0) B*=-1;   if(G<0) G*=-1;   if(R<0) R*=-1;
84.     if(nY<0)    nY=0;    if(nB<0)    nB=0;
85.     if(nG<0)    nG=0;    if(nR<0)    nR=0;
86.     if(nY>255)  nY=255;  if(nB>255)  nB=255;
87.     if(nG>255)  nG=255;  if(nR>255)  nR=255;
88.
89.     if(input->bmih.biBitCount==8)
90.         output->data[i*input->LineBytes+j]=nY;
91.     else{
92.         output->data[i*input->LineBytes+j*3+0]=nB;
93.         output->data[i*input->LineBytes+j*3+1]=nG;
94.         output->data[i*input->LineBytes+j*3+2]=nR;
95.     }
96. }
97. }
98.
99. }
```


4. 实验结果展示

本次实验有如下可以更改的参数： size , σ_s , σ_r , 且可以对双边滤波算法进行多次迭代。

4. 1. 基础效果展示

| Origin | Bilateral Filter |
|---|--|
|  |  |
| Size = 7, $\sigma_s = 16$, $\sigma_r = 20$; 脸上粗糙部分有效磨平, 且保留绝大部分边缘细节 | |
|  |  |
| Size = 7, $\sigma_s = 10$, $\sigma_r = 50$; 噪点被模糊, 房屋建筑边缘保持良好 | |



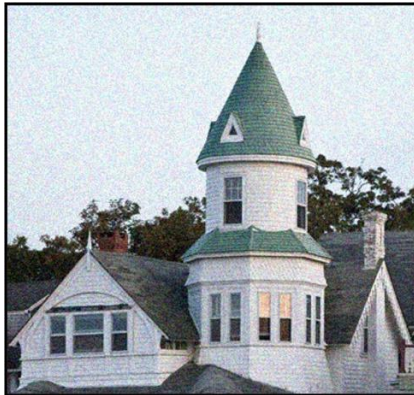





Size = 7, $\sigma_s = 20$, $\sigma_r = 60$; 噪点被平滑模糊, 房屋建筑边缘较为清晰



Size = 7, $\sigma_s = 20$, $\sigma_r = 40$; 对本就清楚的图片几乎没有改变。

4. 2. 探究 σ_r 的作用







将 size 定为 15×15 ，固定 σ_s 为推荐值（图像对角线长度 $\times 2\%$ ）。

| | | |
|---|--|---|
| Origin | $\sigma_r = 5$ | $\sigma_r = 10$ |
|  |  |  |
| $\sigma_r = 20$ | $\sigma_r = 60$ | $\sigma_r = 300$ |
|  |  |  |

观察发现随着 σ_r 的增大，图像的模糊效果越来越重。当 σ_r 较小时，强度域集中于中心像素点，处理效果趋近于原图。当 σ_r 趋于较大值时，强度域较为分散，图像接近高斯模糊效果。对本张图片，当 $\sigma_r = 20$ 时，双边滤波效果最好。此时既消除了明显的噪点，又保留了物体边缘，同时在建筑墙壁上也没有出现过度模糊不清的效果。

4.3. 探究 σ_s 的作用







将 size 定为 15×15 ，固定 σ_r 为 20（经验而言此时效果较好）。

| Origin | $\sigma_s = 0.5$ | $\sigma_s = 1.5$ |
|---|--|---|
|  |  |  |
| $\sigma_s = 3$ | $\sigma_s = 5$ | $\sigma_s = 40$ |
|  |  |  |

观察发现随着 σ_s 的增大，图像的模糊效果同样越来越重。不过当 σ_s 达到一定大小时，之后的模糊效果便不明显。而当 σ_s 较小时，空间域集中于中心像素点，处理效果也趋近于原图。当 σ_s 趋于较大值时，强度域较为分散，图像接近高斯模糊效果。对本张图片，当 $\sigma_s = 3$ 时双边滤波效果最好，脸上粉刺被磨皮，且各处细节保持较好； $\sigma_s = 5$ 时，已经有部分脸部轮廓边界出现颜色逸散，且唇部细节模糊严重； σ_s 再增大基本已无影响。

4.4. 探究 Size 的作用







固定 σ_R 为 20，固定 σ_S 为推荐值（图像对角线长度 $\times 2\%$ ，此处 ≈ 16 ）。

| Origin | size = 3 | size = 5 |
|---|---|---|
|  |  |  |
| size = 7 | size = 15 | size = 30 |
|  |  |  |

观察发现随着 size 的增大，图像的双边处理效果越来越好，运行时间也随之急剧增长。当 size = 15 时，双边滤波效果已经很好，较为干净地模糊去除了大部分噪点，同时对建筑轮廓有清晰地保留。且当 size 在 15 之上继续增长时，一张图像的处理时间过长，且 size = 30 与 size = 60 在处理效果上难分优劣。故 size = 15 是一个较为合适的参数。

4.5. 探究双边滤波的迭代

固定 $\text{size} = 15$, $\sigma_R = 20$, σ_S 为推荐值 (图像对角线长度 $\times 2\%$, 此处 ≈ 16)。

| Origin | BF = 1 | BF = 2 |
|---|---|---|
|  |  |  |
| BF = 3 | BF = 4 | BF = 5 |
|  |  |  |

通过观察可以发现, 双边滤波在经过迭代后会逐渐模糊图像内部细节, 最终侵蚀到物体边缘。在多次迭代后图像会失去原有轮廓。

5. 心得体会

通过这次实验我更加深刻的理解了卷积与滤波的关系，并实现了双边滤波。

本次实验中，高斯滤波对高频、低频信息的过滤让我对图像信息的处理有了新的认识，而双边滤波在原理上将图像的空间域和强度域共同纳入考量，用强度域牵制空间域从而做到只在部分区域模糊、去除噪声的做法更是十分有趣。

在完成双边滤波模型后，我使用 violet 图像进行测试，更改参数发现并无模糊效果，以为实现有错。后经推理验证发现，双边滤波难以在较小 size 下对高清图片产生效果。

针对 size, σ_R , σ_S 等参数，经过图像生成比对，我也有更直观地了解。总体而言，随着参数的增大，图像模糊的效果都会越来越好。起初这体现为在保留边缘的同时对噪点的去除效果增强；然而当参数大到一定程度后其便在运行时间、模糊区域、模糊程度上出现各自的劣势。故而选择合适的参数对于双边滤波十分重要。且参数并非固定，对于不同的图片和不同的预期处理效果，参数的选择也有区别；比如对于磨皮与对不同颗粒大小噪点的去除，有不同的最佳试验参数。

此外，当对双边滤波进行迭代时，图像的平坦区域模糊程度会加重，最终侵蚀到图像轮廓，导致双边滤波保留边缘的效果消失。验证了课件中提及的效果。

本次实验还是有较多收获，也在多次的比对运行中得到了很多乐趣。希望能在以后的实验中用到本次所学的滤波来处理图像，也希望以后能有机会实现双边滤波的加速算法以及其他能保留图形边缘的滤波算法。

2023. 12. 20