**实验报告**

专业：计算机科学与技术

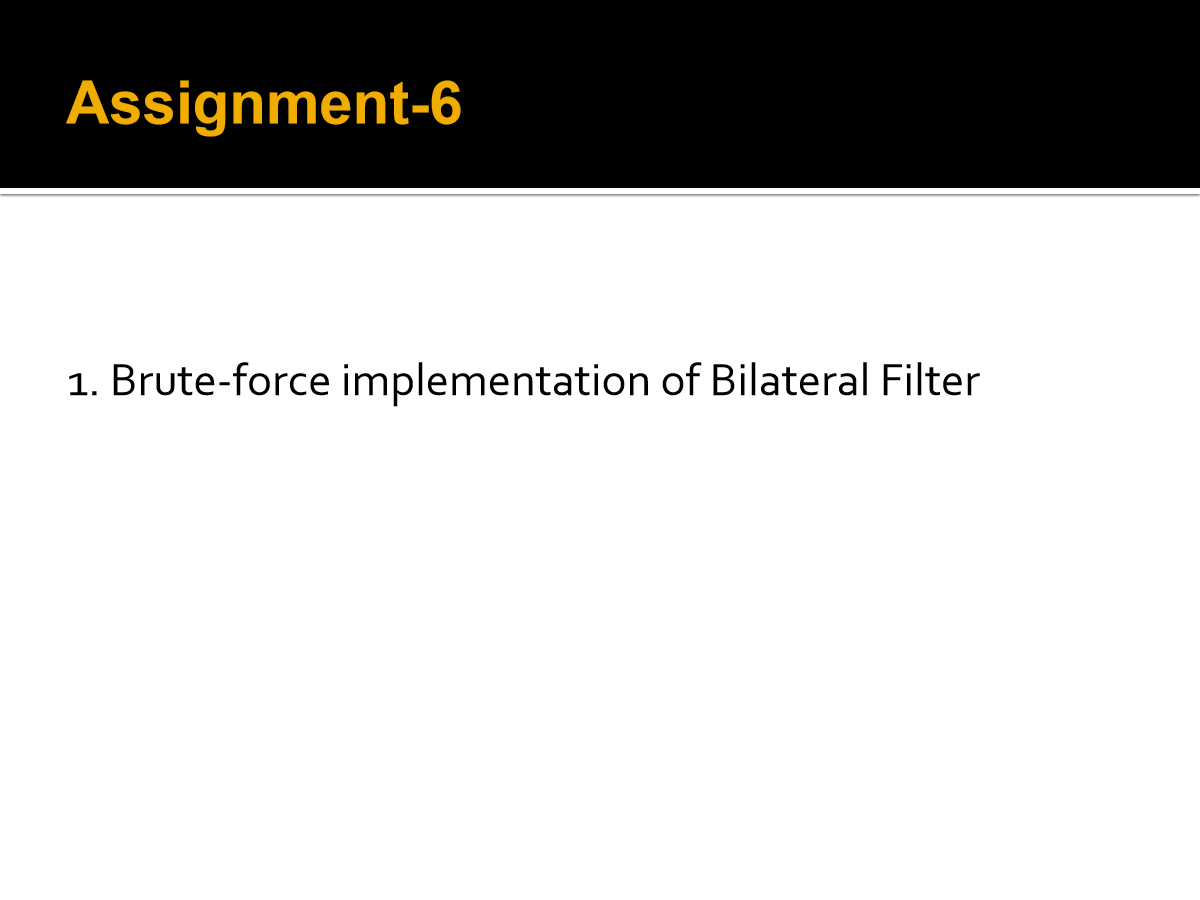
日期：2022/11/30

课程名称： 图像信息处理 指导老师： 宋明黎 成绩：

实验名称： 双边滤波的实现

1. **实验目的和要求**

1.用暴力求解的方式实现双边滤波



**二、实验内容和原理**

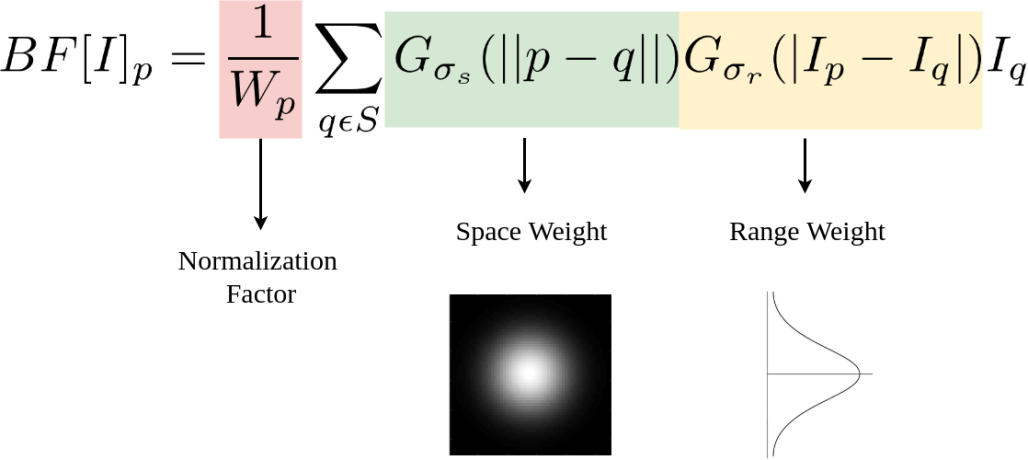
.bmp图像信息的贮存，往往有固定的格式，用二进制的方式打开bmp文件，可以设置相对映的数据结构接受文件头部信息、数据头部信息以及palette信息与bitmap data，从而进行相应的数据操作；本实验选用RGB模式图像文件（要求长宽像素均为8的倍数）进行暴力求解双边滤波操作，且在实验过程中，以对灰度值进行操作表示对图像像素值数据进行相应的操作。

[双边滤波](https://so.csdn.net/so/search?q=%E5%8F%8C%E8%BE%B9%E6%BB%A4%E6%B3%A2&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/liangfei868/article/details/_blank)（Bilateral Filtering）的基本思路是同时考虑像素点的空域信息和值域信息。即先根据像素值对要用来进行滤波的邻域做一个分割或分类，再给该点所属的类别相对较高的权重，然后进行邻域加权求和，得到最终结果。

双边滤波可以用于平滑图像并减少噪声，同时保留重要的边缘，是一种非线性双边滤波器。在平坦区域，临近像素的像素值的差值较小，对应值域权重接近于1，此时空域权重起主要作用，相当于直接对此区域进行高斯模糊。因此，平坦区域相当于进行高斯模糊。在边缘区域，临近像素的像素值的差值较大，对应值域权重接近于0，导致此处核函数下降，当前像素受到的影响就越小，从而保持了原始图像的边缘的细节信息。

总体而言，在像素强度变换不大的区域，双边滤波有类似于高斯滤波的效果，而在图像边缘等强度梯度较大的地方，可以保持梯度（方向不能保证，因而可能产生梯度逆转的现象），因而双边滤波有较好的滤波效果与保边的功能。

双边滤波的函数形式如下：



因为双边滤波需要对每个像素点进行一定高斯核空间内求和再归一，因而对于较大的图像信息而言，时间的开销十分巨大；时间的开销随着高斯核大小的变化也随之发生改变。

在本实验中，我选用图像总对角线像素大小 2% 作为像素值差值权重，自定义距离差值的权重，并根据两倍自定义距离差值定义高斯核的大小（保证其为奇数），用这些参数，配合一元正态分布函数进行双边滤波的的计算，预期得到消除噪声，并保留边缘的结果。

**三、实验步骤与分析**

1. 建立存储图像信息的数据结构（头文件）与bmp附带信息基本一一对应。

typedef unsigned char bit;

typedef unsigned short word;

typedef unsigned long dword;

struct tagBITMAPFILEHEADER

{

word bftype; //文件类型

dword bfsize; //文件大小（字节为单位）

word bfreserved1; //保留，必须设为0

word bfreserved2; //保留，必须设为0

dword bfoffbits; //到实际图像实际开始的偏移量

};

struct tagBITMAPINFOHEADER

{

dword bisize; //结构所需字节数

long biwidth; //图像宽度（像素为单位）

long biheight; //图像长度（像素为单位）--正值对映图像倒立

word biplanes; //位面数，常为1

word bibitcounts; //比特/像素比

dword bicompression; //压缩类型

dword bisizeimage; //图像大小

long bixpelspermeter; //水平分辨率

long biypelspermeter; //垂直分辨率

dword biclrused; //颜色索引数

dword biclrimportant; //重要影响颜色索引数

};

1. 声明本实验所需使用函数（具体解析见下）。

//函数声明

//写入文件函数

void write(tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information, bit\* rgbpic, long int number, const char name[]);

//YUV转换RGB函数

void change(long int picheight, long int picwidth, float\* newy, float\* yuvpic, bit\* rgbpic1);

//一元正态分布函数

double function(double var,double x);

1. 二进制打开.bmp文件，读取文件内数据信息，文件指针偏移至bitmap data数据区域，记录bitmap data区域对映像素索引矩阵。

// 定义图片信息接收变量

tagBITMAPFILEHEADER file;

tagBITMAPINFOHEADER information;

//定义图像长宽

long int picwidth;

long int picheight;

//定义循环指数

long int i;

//提取bmp文件头部信息

FILE\* fp;

fp = fopen("lena.bmp", "rb");

fread(&(file.bftype), 1, sizeof(word), fp);

fread(&(file.bfsize), 1, sizeof(dword), fp);

fread(&(file.bfreserved1), 1, sizeof(word), fp);

fread(&(file.bfreserved2), 1, sizeof(word), fp);

fread(&(file.bfoffbits), 1, sizeof(dword), fp);

fread((&information), sizeof(tagBITMAPINFOHEADER), 1, fp);

//获取bmp大小像素信息

picwidth = information.biwidth;

picheight = information.biheight;

//接收bmp图像数据信息

bit\* rgbpic = (bit\*)malloc(picwidth \* picheight \* 3 \* sizeof(bit));

bit\* rgbpic1 = (bit\*)malloc(picwidth \* picheight \* 3 \* sizeof(bit));

fseek(fp, file.bfoffbits, SEEK\_SET);

fread(rgbpic, sizeof(bit), picwidth \* picheight \* 3, fp);

float\* yuvpic = (float\*)malloc(picwidth \* picheight \* sizeof(float) \* 3);

float\* newy = (float\*)malloc(picwidth \* picheight \* sizeof(float) \* 3);

float\* recordy = (float\*)malloc(picwidth \* picheight \* sizeof(float) \* 3);

1. 将图像从 RGB 格式转化为 YUV格式，便于后续双边滤波操作。

//利用公式将BGR类型转换为YUV类型

for (i = 0;i < picwidth \* picheight;i++)

{

yuvpic[3 \* i] = 0.11 \* rgbpic[3 \* i] + 0.59 \* rgbpic[3 \* i + 1] + 0.3 \* rgbpic[3 \* i + 2];

yuvpic[3 \* i + 1] = 0.493 \* (rgbpic[3 \* i] - yuvpic[3 \* i]);

yuvpic[3 \* i + 2] = 0.877 \* (rgbpic[3 \* i + 2] - yuvpic[3 \* i]);

}

1. 对图像文件主体部分进行双边滤波操作。

int x, y;

//根据对角线长度的2%定义sigema\_r

double optimal;

optimal = sqrt(pow(picwidth, 2) + pow(picheight, 2));

//定义sigema\_s和sigema\_r，前者对映距离权重，后者对映像素值权重

double omg = 7;

//取高斯核大小为2\*sigema\_s，并保证其为奇数

int rage = 2 \* omg + 1;

double smd = 0.02 \* optimal;

//定义距离，像素值差值等各项权重

double distance;

double light;

//定义归一化总和

double sum;

//定义总像素值之和

double real;

double record;

//处理图像主体部分像素值

for (i = rage / 2 ;i < picheight - rage / 2;i++)

for (j = rage / 2 ;j < picwidth - rage / 2;j++)

{

//对每一高斯核，重置化距离、像素值权重为0

sum = 0;

real = 0;

//遍历高斯核中每一点

for (x = 0;x < rage;x++)

for (y = 0;y < rage;y++)

{

//计算每一点的距离权重与像素值差值权重

distance = sqrt(pow(rage / 2 - x, 2) + pow(rage / 2 - y, 2));

light = fabs(newy[i \* picheight + j] - newy[(i - rage / 2 + y) \*

picheight + j - rage / 2 + x]);

record = function(pow(omg, 2), distance) \* function(pow(smd, 2), light);

//更新归一化数值与像素值计算数值

real = real + record \* newy[(i - rage / 2 + y) \* picheight + j - rage / 2 + x];

sum = sum + record;

}

//更新中心像素点像素值

recordy[i \* picheight + j] = real / sum;

}

6.对图像文件边界部分进行变形（高斯核大小发生相应的改变）的双边滤波操作，以最下部分的边界为例；其余最上，最右，最左部分的边界处理思路与之相同。

//计算最下部分的边界像素点像素值

for (i = 0;i < rage / 2;i++)

for (j = 0;j < picwidth - rage / 2;j++)

{

//对每一高斯核，重置化距离、像素值权重为0

sum = 0;

real = 0;

//遍历高斯核中每一点

for (x = 0;x < rage / 2;x++)

for (y = 0;y < rage / 2;y++)

{

// 计算每一点的距离权重与像素值差值权重

distance = sqrt(pow(x, 2) + pow(y, 2));

light = fabs(newy[i \* picheight + j] - newy[(i + y) \* picheight + j + x]);

record = function(pow(omg, 2), distance) \* function(pow(smd, 2), light);

//更新归一化数值与像素值计算数值

real = real + record \* newy[(i + y) \* picheight + j + x];

sum = sum + record;

}

//更新中心像素点像素值

recordy[i \* picheight + j] = real / sum;

}

7.YUV 转化为 RGB 函数

输入值：1.原图像的总列像素个数

2.原图像的总行像素个数

3.新图像的灰度值

4.原图像的灰度值

5.更新后 RGB 值的贮存区域

输出值：无

//YUV转换为RGB

void change(long int picheight, long int picwidth, float\* newy, float\* yuvpic, bit\* rgbpic1)

{

//将YUV格式转化为RGB格式

//若YUV相应转换值大于255，则保留255

//若YUV相应转换值小于0，则保留0

int i;

for (i = 0; i < picwidth \* picheight; i++)

{

//YUV转化B

if (newy[i] + 2.0284 \* yuvpic[3 \* i + 1] < 0)

rgbpic1[3 \* i] = 0;

else if (newy[i] + 2.0284 \* yuvpic[3 \* i + 1] > 255)

rgbpic1[3 \* i] = 255;

else

rgbpic1[3 \* i] = newy[i] + 2.0284 \* yuvpic[3 \* i + 1];

//YUV转化G

if (newy[i] - 0.3781 \* yuvpic[3 \* i + 1] - 0.5798 \* yuvpic[3 \* i + 2] < 0)

rgbpic1[3 \* i + 1] = 0;

else if (newy[i] - 0.3781 \* yuvpic[3 \* i + 1] - 0.5798 \* yuvpic[3 \* i + 2] > 255)

rgbpic1[3 \* i + 1] = 255;

else

rgbpic1[3 \* i + 1] = newy[i] - 0.3781 \* yuvpic[3 \* i + 1] - 0.5798 \* yuvpic[3 \* i + 2];

//YUV转化R

if (newy[i] + 1.1403 \* yuvpic[3 \* i + 2] < 0)

rgbpic1[3 \* i + 2] = 0;

else if (newy[i] + 1.1403 \* yuvpic[3 \* i + 2] > 255)

rgbpic1[3 \* i + 2] = 255;

else

rgbpic1[3 \* i + 2] = newy[i] + 1.1403 \* yuvpic[3 \* i + 2];

}

}

8.写入函数

输入值：1.bmp文件信息头文件

2.bmp文件数据头文件

3.bmp文件各像素RGB数值记录表

4.bmp文件总像素大小（行像素总数乘以列像素总数）

5.bmp文件名称

//写入文件函数

void write(tagBITMAPFILEHEADER file, tagBITMAPINFOHEADER information, bit\* rgbpic, long int number, const char name[])

{

FILE\* fp;

//打开/新建文件

fp = fopen(name, "wb");

//写入bmp图像各类数据

fwrite(&(file.bftype), 1, sizeof(file.bftype), fp);

fwrite(&(file.bfsize), 1, sizeof(file.bfsize), fp);

fwrite(&(file.bfreserved1), 1, sizeof(file.bfreserved1), fp);

fwrite(&(file.bfreserved2), 1, sizeof(file.bfreserved2), fp);

fwrite(&(file.bfoffbits), 1, sizeof(file.bfoffbits), fp);

fwrite(&information, 1, sizeof(information), fp);

fwrite(rgbpic, sizeof(bit), number \* 3, fp);

//关闭bmp图像文件

fclose(fp);

}

**四、实验环境及运行方法**

本实验使用C语言进行编写，vs正常编译环境运行，其中需要以下语句忽略vs对fopen函数的安全性中断。

#pragma warning(disable:4996) //忽略vs对fopen的安全性错误

对于实验结果，点击运行即可。

输入图像为 24位色RGB 格式图像，文件名为“try.bmp”，仅输出一张新生成的bmp图片，文件名“look4.bmp”。此文件为经过实际运行对比，输出结果较为满意的双边滤波处理图像文件，若想运行对比不同的双边滤波处理条件，可以更改sigema\_s,sigema\_r的对映数值，本实验中，高斯核的大小会根据sigema\_s的变化发生相应的变化，若采取过大的sigema\_s值，可能将导致程序运行时间大大提升，因而需要谨慎选取，或将高斯核大小与sigema\_s的数值分割指定。

对于sigema\_r参数的选取，本实验借鉴了对映的理论经验值，即图像文件总对角线像素值的 2% 作为理想选择，读者可以根据需要自行修改大小。

其中最终预期预期文件夹中存放着一系列的双边滤波处理结果，对应于经验最优

sigema\_r值下，随着sigema\_s的不断变大，平滑效果逐渐明显的对比图，可供参考与复刻。

**五、实验结果展示**

下图为本实验原始输入图片（图片来源（截取）于PPT），可以关注模特的左脸部分，在原图中，脸上存在着较多的自然黑斑。



通过双边滤波操作，输出系列结果如下：

通过双边滤波操作，输出系列结果如下：





上面列出的四张图片，从左至右从上至下分别对映sigema\_r取值图像总对角线像素值2%，sigema\_s 分别取值 3，5，7，9，且对映高斯核大小随sigema\_s发生相应变化下的双边滤波输出结果。

从前三张输出结果可以看出，随着双边滤波的使用，模特脸上的斑点逐渐被模糊掉，图像变得平滑，脸上的颜色也显得连贯而自然，可见平滑的滤波效果十分明显，对比第三张输出图片和第四章输出图片，可以发现模特脸上的腮红部分显得更加自然而平滑，一些带有棱角的红色区域也在脸上大大减小，可见平滑滤波的细节成功性。

在明显的边角方面，可以发现平滑滤波并没有消除模糊这些边，人脸的轮廓及头发部分的清晰度依然保持较高的状态，由此也可以得出结果，双边滤波的保边效果得到较好的保证。

最后，确实在图像的一些边缘部分出现了异常颜色，发脚部分的交界处有些许明显，这也印证了双边滤波保留边际时的无方向性，有一定的可能造成梯度逆转的现象发生。

**六、心得体会**

逐渐步入轨道的dip project，第一次感受到现代电脑也要运行良久的暴力运算程序。

双边滤波的作业，宋老师没有第一时间布置，而是选择在讲话双边滤波以及一系列后人改进的措施之后，再让我们用最原始的方法去解决一个看似不优的问题。在PPT的系列对比图中，我们可以看到双边滤波的优越性，对于P图爱好者来说，又可以保边，又可以平滑图像，这样的效果，怎能不爱，但是，原始双边滤波的最大掣肘，就是运行的时间代价过大，对于每个像素点，都要进行高斯核大小面积的运算，再使用归一化求得新像素值，且对于边界来说，高斯核的取用还需要进行一些变化，因而对于运算的需求，即使在现代电脑上跑，也要以min为单位的运行时间。

另外，双边滤波并没有给出选取最优权重参数的方法，一切参数的选取都依赖前人的经验以及程序运行后的测试结果；结合实际的程序开销，寻找最优的双边滤波结果，真是耗时又消耗算力的事。

最后，也同样是碰到了一个小问题，我程序的实验极度依赖于长宽像素的个数均为8的倍数，不然图像输出就会产生异常，这可能是结构设置与使用的问题，我暂时还没有快捷的优解（可以考虑扩展或收缩到8的倍数，但这会造成图像像素的损失，或无效像素的添加，且改变了图像的原始大小，我并不打算如此操作）。另外，因为本来使用的lena图像使用双边滤波的效果不是很明显，我则去PPT中寻找图像，以显著的效果对比说明程序的正确性。

最后也是无意中发现，WPS的压缩过程中，会自动帮我进行类似双边滤波的操作，因而压缩会导致噪声被直接去除。