

# 图像信息处理

## 实验六-双边滤波

黄昭阳 3130000696 - 15/20/12

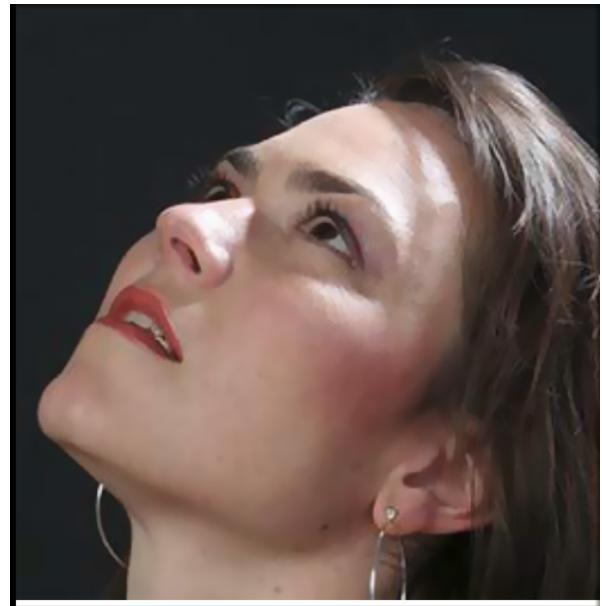


## 双边滤波

### 一、结果展示：

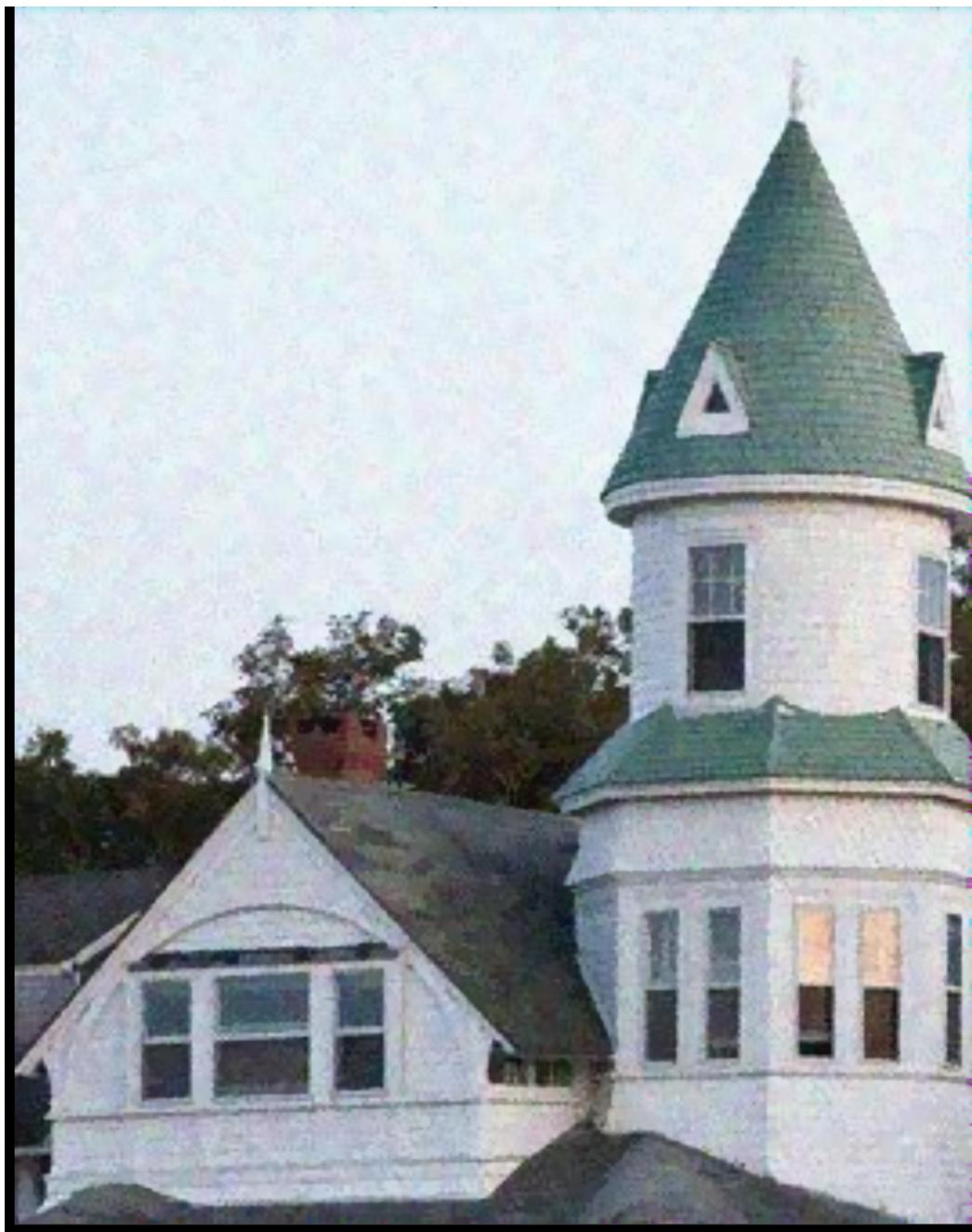
双边滤波采用了三种方法实现：

1. 直接根据灰度值进行双边滤波，对于彩色图也是用对RGB三通道分别处理的方法进行滤波，这样就忽略了RGB之间的连续性，不是太好，但是也有一些效果，像下面这张图，去噪效果还是十分明显的。下图取 $\text{sigmar} = \text{sigmas} = 10$

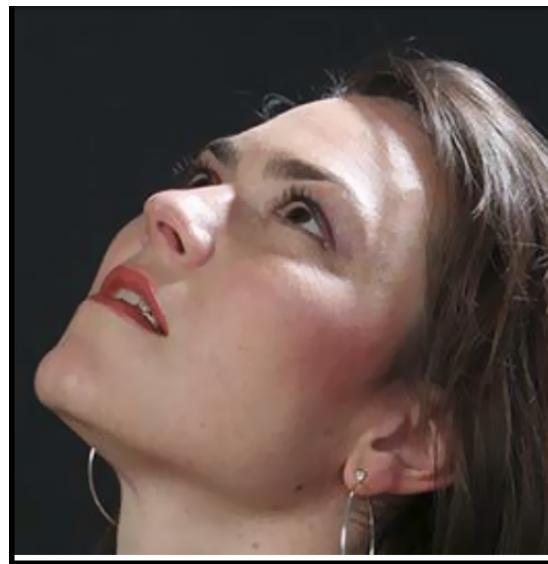


---

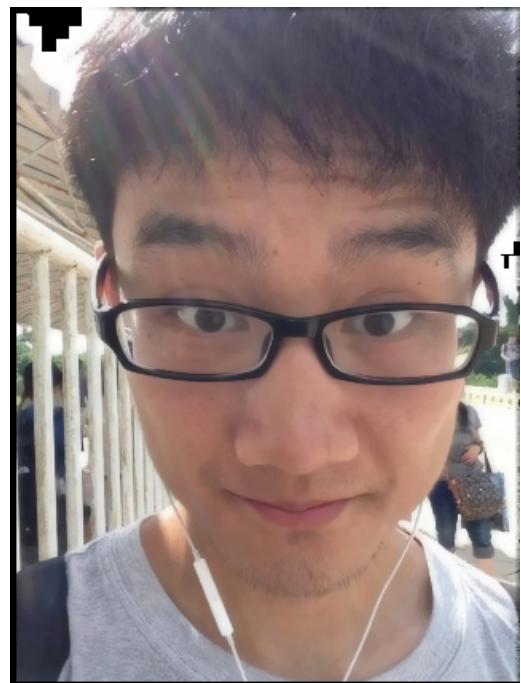
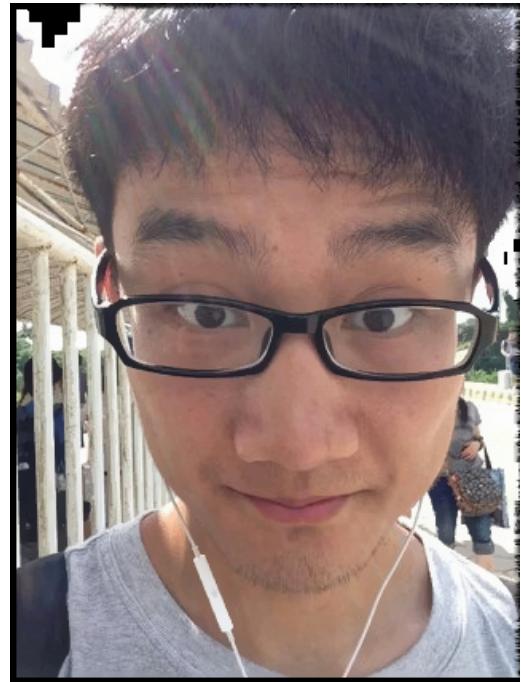
下面这张图，是对老师所给的图978x1242使用参数(sigmas = 20,sigmar = 10)的效果，保留了房屋的纹理细节，但是消去了大量噪声点。



- 
2. 对于彩图，使用梯度值（RGB）作为衡量标准，参数与1中的相同。



3. 对于一张图，使用这张图大小的2%来生成一个用于space的高斯模板，然后对于每个像素点，依据模板来生成sigmar，这里取了两种方法，一者是取域内的均方差，一者是取域内梯度值的平均值，均方差效果噪声点去除的比较少，平均值效果较好，左边是使用方差，下面是使用梯度值平均值，明显下方更柔和，噪点更少。



---

一下为使用梯度值均值滤波，一次迭代，二次迭代，三次迭代得到的图。







## 二、代码分析

```
//对应生成方法1的双边滤波模板
void bmpFile::generateBilateralFilter(Matrix* inputFilter,Matrix*
resultFilter, float sigmar,int r,int c,int times,int k)
//根据输入模板的内容和大小, 增加与像素相关的因素, 生成一个新模板
//inputFilter 为传入的像素模板
//resultFilter 为计算后传出的像素模板, sigmar为像素强度的sigma
//r,c为当前中心像素的坐标, times为颜色通道数, k为第几个颜色通道
{
    int startRow = r-inputFilter->Row/2;
    int startCol = c-inputFilter->Col/2;
//根据中心像素坐标和模板大小计算出左上角坐标
    int row,col;
    float intensity_dis;
    for(int i=0;i<inputFilter->Row; i++)
    {
        row = i+startRow;
        if(row>=bmpInfoHeader.biHeight) break;
        if(row<0) continue;
        for(int j=0;j<inputFilter->Col; j++)
        {
            col = startCol+j;
            if(col>=bmpInfoHeader.biWidth) break;
            if(col<0) continue;
//((row,col)为当前需要计算的像素
            intensity_dis = fabs(bmpFileData[row][col*times+k]-
bmpFileData[r][c*times+k]);
//根据颜色通道数和当前的颜色通道, 计算两个像素之间的强度距离
            resultFilter->data[i][j] =
Matrix::GaussianFunction(sigmar,intensity_dis)*inputFilter->data[i][j];
//结果矩阵生成该距离对应的高斯函数值, 乘到模板上
        }
    }
}
```

```
//对应生成方法2的双边滤波模板
void bmpFile::generateColorfulBilateralFilter(Matrix* inputFilter,Matrix* resultFilter, float sigmar, int r, int c)
//综合考虑RGB三个通道，因此不再传入times、k与颜色通道相关的参数，其他参数类似
{
    int startRow = r - inputFilter->Row/2;
    int startCol = c - inputFilter->Col/2;
    int row, col;
    float intensity_dis;
    int times = 3;
    for(int i=0;i<inputFilter->Row;i++)
    {
        row = i + startRow;
        if(row>=bmpInfoHeader.biHeight) break;
        if(row<0) continue;
        for(int j=0;j<inputFilter->Col;j++)
        {
            col = startCol + j;
            if(col>=bmpInfoHeader.biWidth) break;
            if(col<0) continue;
            intensity_dis = 0;
//与灰度值双边滤波器不同，这里使用三维颜色向量之差的模来计算颜色距离
            for(int k=0;k<times;k++)
                intensity_dis += sqr(bmpFileData[row][col*times+k]-
bmpFileData[r][c*times+k]);
            intensity_dis = sqrt(intensity_dis);
            resultFilter->data[i][j] =
Matrix::GaussianFunction(sigmar,intensity_dis)*inputFilter->data[i][j];
//增加颜色向量的因素到滤波器之中
        }
    }
}
```

```

//对应方法1的灰度值双边滤波
void bmpFile::BilateralFilter(float sigmas, float sigmar)
{
    Matrix* filter = Matrix::GaussianFilter(sigmas);
    Matrix* GaussianFilter = Matrix::GaussianFilter(sigmas);
    Matrix m[3] = {
        Matrix(bmpInfoHeader.biHeight, bmpInfoHeader.biWidth),
        Matrix(bmpInfoHeader.biHeight, bmpInfoHeader.biWidth),
        Matrix(bmpInfoHeader.biHeight, bmpInfoHeader.biWidth)
    };
    //m用来存储计算结果
    float sum[3];
    int times;
    if(bmpInfoHeader.biBitCount == 24) times = 3;
    else times = 1;
    //根据bibitcount判断颜色通道数目
    for(int k=0;k<times;k++)
    {
        //对于每个颜色通道都独立的做一遍双边滤波
        for(int i=0;i<bmpInfoHeader.biHeight;i++)
            for(int j=0;j<bmpInfoHeader.biWidth;j++)
        //以每个像素点为中心做一次计算
        {
            sum[k] = 0;
            generateBilateralFilter(GaussianFilter,filter,sigmar,i,j,times,k);
            //输入一个高斯滤波器模板，生成一个颜色强度高斯方差为sigmar的双边滤波器
            for(int ii = 0;ii<filter->Row;ii++)
                for(int jj=0;jj<filter->Col;jj++)
                    sum[k]+=filter->data[ii][jj];
            //计算滤波器值之和
            m[k][i][j] = Convolution(filter,i,j,times,k);
            //m[k][i][j]存储第k个颜色通道，像素中心为(i,j)的计算结果，即双边滤波结果
            if(sum[k]!=0) m[k][i][j] = m[k][i][j]/sum[k];
            //做归一化
        }
        for(int i=0;i<bmpInfoHeader.biHeight;i++)
            for(int j=0;j<bmpInfoHeader.biWidth;j++)
                bmpFileData[i][j*times+k] = (unsigned
char)clip(bmpFileData[i][j*times+k]-m[k][i][j]);
        //抽取变化前后的两者颜色之差，用作halo，这里可以忽视
    }
    exportToFile("!.bmp");
    for(int k=0;k<times;k++)
        for(int i=0;i<bmpInfoHeader.biHeight;i++)
            for(int j=0;j<bmpInfoHeader.biWidth;j++)
                bmpFileData[i][j*times+k] = (unsigned char)m[k][i][j];
    //将计算结果放到bmp的数据中
}

```

```

//对应方法2的全局颜色高斯方差恒等，但像素距离使用向量之差的模的滤波方式
//具体行为十分类似，这里只介绍不同之处
void bmpFile::ColorfulBilateralFilter(float sigmas, float sigmar)
{
    Matrix* filter = Matrix::GaussianFilter(sigmas);
    Matrix* GaussianFilter = Matrix::GaussianFilter(sigmas);
    Matrix m[3] = {
        Matrix(bmpInfoHeader.biHeight, bmpInfoHeader.biWidth),
        Matrix(bmpInfoHeader.biHeight, bmpInfoHeader.biWidth),
        Matrix(bmpInfoHeader.biHeight, bmpInfoHeader.biWidth)
    };

    float sum;
    int times;
    if(bmpInfoHeader.biBitCount == 24) times = 3;
    else return;
    //必定是彩色图，如果不是则直接退出
    for(int i=0;i<bmpInfoHeader.biHeight;i++)
        for(int j=0;j<bmpInfoHeader.biWidth;j++)
    {
        sum = 0;
        generateColorfulBilateralFilter(GaussianFilter,filter,sigmar,i,j);
        //上一个函数中这里使用的是generateBilateralFilter，计算的到的滤波器不同
        //上方已经详细介绍了两个生成滤波器的不同之处，使用该滤波器同时对三通道进行滤波
        for(int ii = 0;ii<filter->Row;ii++)
            for(int jj=0;jj<filter->Col;jj++)
                sum+=filter->data[ii][jj];
        for(int k=0;k<times;k++)
        {
            m[k][i][j] = Convolution(filter,i,j,times,k);
            if(sum!=0) m[k][i][j] = m[k][i][j]/sum;
        }
        //对三个颜色通道的值都是用该滤波器修改
    }
    for(int k=0;k<times;k++)
        for(int i=0;i<bmpInfoHeader.biHeight;i++)
            for(int j=0;j<bmpInfoHeader.biWidth;j++)
                bmpFileData[i][j*times+k] = clip((bmpFileData[i][j*times
+k]-m[k][i][j])*10);
    exportToFile("!!!.bmp");
    for(int k=0;k<times;k++)
        for(int i=0;i<bmpInfoHeader.biHeight;i++)
            for(int j=0;j<bmpInfoHeader.biWidth;j++)
                bmpFileData[i][j*times+k] = (unsigned char)m[k][i][j];
}

```

```

//计算以r, c为像素中心,以模板大小为界限, 获得其内平均像素向量, 然后取这个向量与其他向量
//的差的模的平均值, 作为像素强度高斯函数的方差
float bmpFile::getVariance(Matrix* inputFilter, int r, int c, int times)
{
    int startRow = r - inputFilter->Row/2;
    int startCol = c - inputFilter->Col/2;
    int row, col;
    float sum[3] = {0, 0, 0}; //计算和向量
    float number = 0; //记录向量个数
    for(int i=0; i<inputFilter->Row; i++)
    {
        row = i + startRow;
        if(row >= bmpInfoHeader.biHeight) break;
        if(row < 0) continue;
        for(int j=0; j < inputFilter->Col; j++)
        {
            col = startCol + j;
            if(col >= bmpInfoHeader.biWidth) break;
            if(col < 0) continue;
            number++;
            for(int k=0; k < times; k++)
                sum[k] += bmpFileData[row][col * times + k];
        }
    }
    float average[3];
    for(int k=0; k < times; k++)
        average[k] = sum[k] / number; //记录平均向量
    float res = 0, tmp; //res为返回值
    for(int i=0; i < inputFilter->Row; i++)
    {
        row = i + startRow;
        if(row >= bmpInfoHeader.biHeight) break;
        if(row < 0) continue;
        for(int j=0; j < inputFilter->Col; j++)
        {
            col = startCol + j;
            if(col >= bmpInfoHeader.biWidth) break;
            if(col < 0) continue;
            tmp = 0; //用tmp记录三个维度的平方和
            for(int k=0; k < times; k++)
                tmp += sqr(bmpFileData[row][col * times + k] - average[k]);
            res += sqrt(tmp); //res记录模长之和
        }
    }
    return res / number; //返回平均值
    //之前此处计算向量的均方差, 效果不太好于是舍弃了
}

```

