

# 图像滤波和傅里叶变换

## Lab 2

姓名：周炜

学号：32010103790

### 图像滤波

实现盒状均值滤波

实现高斯滤波

实现中值滤波

实现简单的双边滤波

### 利用傅里叶变换完成图像的频域变换

### 代码说明

### 运行环境

滤波

傅里叶变换

## 图像滤波

本次实验的第一部分为图像滤波。盒装均值滤波、高斯滤波、中值滤波和双边滤波都是常用的图像滤波方法,它们各有优缺点,具体如下:

### 盒装均值滤波(Box Filter) 优点:

- 实现简单,计算量小
- 能有效消除椒盐噪声和高斯噪声 缺点:
- 会导致边缘模糊

### 高斯滤波(Gaussian Filter) 优点:

- 能有效消除高斯噪声
- 边缘保持较好 缺点:
- 对椒盐噪声效果较差
- 计算量相对较大

### 中值滤波(Median Filter) 优点:

- 非常有效的去除椒盐噪声
- 能很好地保护边缘 缺点:
- 对高斯噪声效果一般
- 会使图像失真

### 双边滤波(Bilateral Filter) 优点:

- 能有效消除高斯噪声
- 能很好地保护边缘 缺点:
- 对椒盐噪声效果一般
- 计算量相对较大

- 参数选择较为困难

总的来说:

- 如果是去除高斯噪声,高斯滤波和双边滤波效果较好
- 如果是去除椒盐噪声,中值滤波效果最佳
- 如果要兼顾消噪和保边,双边滤波是较好选择
- 如果对计算效率要求较高,盒装均值滤波较为合适

## 实现盒状均值滤波

要求为实现一个名为 `BoxFilter` 的可执行程序, 用法如下:

```
BoxFilter <input-image> <output-image> <w> <h>
```

它使用矩形的卷积核进行均值滤波, 其中, `w` 和 `h` 是卷积核中心点到边界的距离, 卷积核矩阵的大小应当为  $(w*2+1)*(h*2+1)$

其中, 图像卷积的计算可以通过 [cv::filter2D](#) 完成

我采用了卷积核来进行实现, 定义空域卷积:

$$K(u, v) = \begin{cases} \frac{1}{|N|} & \text{if } (u, v) \in N \\ 0 & \text{else} \end{cases}$$

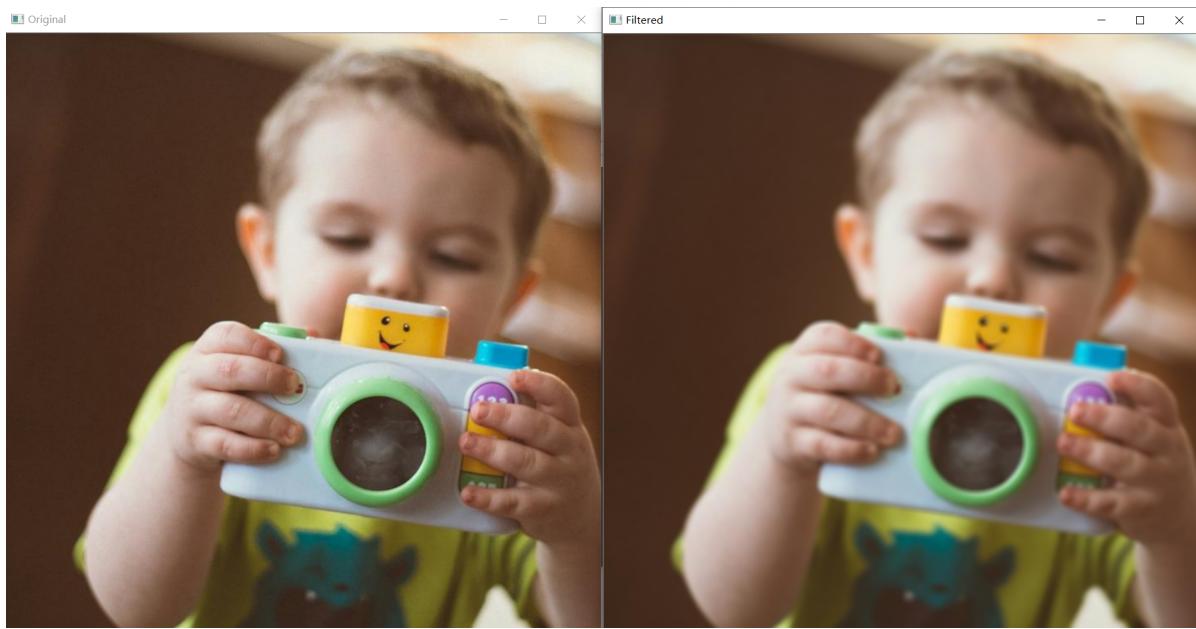
均值滤波可以写成:

$$\bar{I}(x, y) = \sum_{u,v} K(u, v) I(x + u, y + v)$$

代码实现为:

```
void BoxFilter(const Mat& input, Mat& output, int w, int h) {
    // 创建滤波器核
    Mat kern(2 * w + 1, 2 * h + 1, CV_64FC1);
    kern.setTo(1.0 / ((2 * w + 1) * (2 * h + 1)));
    // 进行滤波
    filter2D(input, output, input.depth(), kern);
}
```

实验结果如下( $w = h = 3$ )



可以发现过滤之后图片变模糊了

## 实现高斯滤波

通常我们认为图像像素之间的相关性随着距离增加应该不断减弱，但是(1)的均值并没有体现这一性质。在对图像进行均值滤波时，**如果图像中有一些很显著的亮点，滤波后它的周围会形成光斑**。这正是因为均值滤波无视了距离，对很远处的像素依旧采用同样的权重导致的。一些场合，我们为了美感会需要这种效果。另一些场合，这种结果是不利的，特别是在做图像处理时

高斯滤波是另一种均匀的线性滤波器，它具有如下形式：

$$\bar{I}(x, y) = \sum_{u,v} G(u, v) I(x + u, y + v)$$

其中  $G(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$  为高斯核函数

实现的程序用法如下：

```
GaussianFilter <input-image> <output-image> <sigma>
```

其中，`sigma` 代表了高斯核的参数，是一个浮点数。高斯核理论上大小是无穷的，你可以只保留中心  $5\sigma$  的大小，也就是卷积核矩阵大小为  $(2 * \lfloor 5\sigma \rfloor + 1) \times (2 * \lfloor 5\sigma \rfloor + 1)$

使用 `cv::filter2D` 进行卷积

代码实现如下：

首先需要构造一个高斯核：

```
Mat GetGaussianKernel(double sigma) {
    const double PI = acos(-1);
    int newsize = 2 * 5 * sigma + 1;
    Mat gaus = Mat(newsize, newsize, CV_64FC1);

    int center = newsize / 2;
    double sum = 0;
    for (int i = 0; i < newsize; i++) {
        for (int j = 0; j < newsize; j++) {
```

```

        gaus.at<double>(i, j) = (1 / (2 * PI * sigma * sigma)) * exp(-((i -
center) * (i - center) + (j - center) * (j - center)) / (2 * sigma * sigma));
        sum += gaus.at<double>(i, j);
    }
}

for (int i = 0; i < newsize; i++) {
    for (int j = 0; j < newsize; j++) {
        gaus.at<double>(i, j) /= sum;
    }
}
return gaus;
}

```

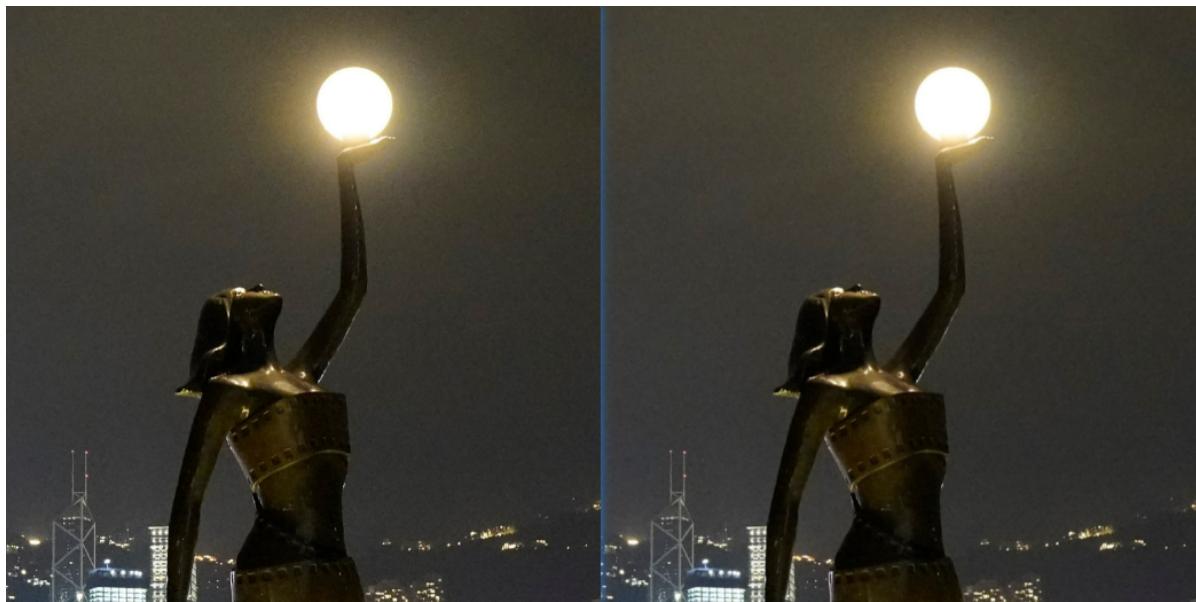
然后用高斯核来进行滤波

```

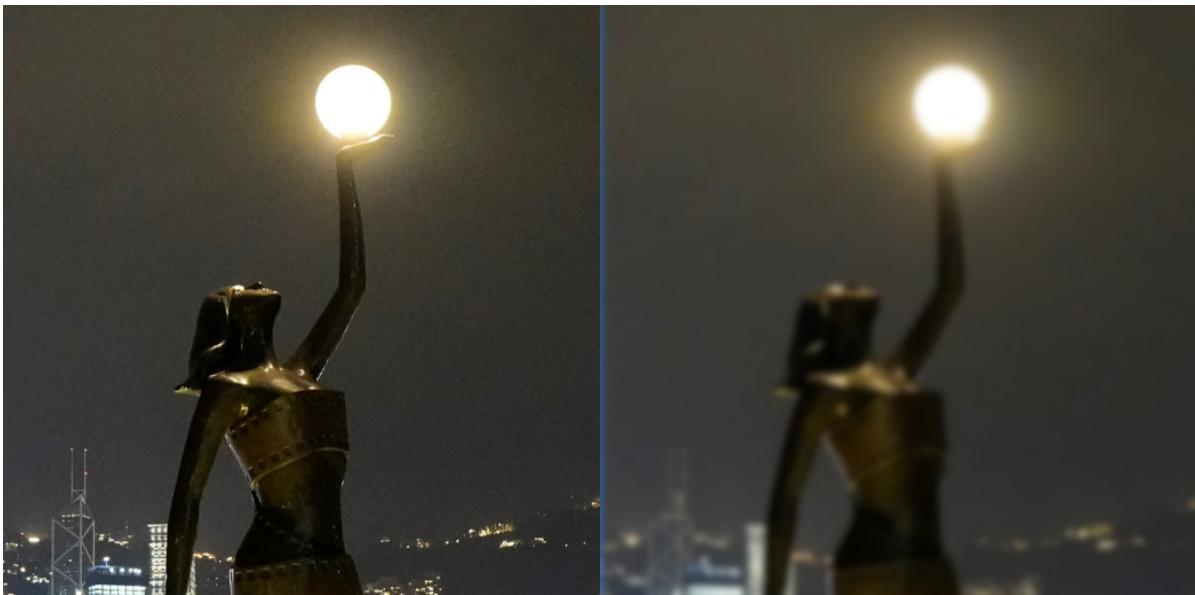
void GaussianFilter(const Mat& input, Mat& output, double sigma) {
    Mat gaus = GetGaussianKernel(sigma);
    filter2D(input, output, input.depth(), gaus);
}

```

当 $\sigma = 1.0$ 的时候可以发现没有什么作用 (左边为原图, 右边为滤波后的图)



$\sigma = 10.0$ 的时候可以发现明显的光斑, 比如左下角的一些灯光已经变成了光斑



课程网站的示例滤波后结果如下：



原始图像

高斯滤波

圆形邻域进行的均值滤波

## 实现中值滤波

高斯滤波适用于图像带有高斯噪声情况下的去噪。在图像被非高斯噪声污染的情况下，高斯滤波不一定能得到理想的去噪效果。

中值滤波是一种序统计滤波器（Order-Statistic Filter），序统计滤波器是依据邻域的值在统计上的次序关系来进行过滤的。这里，中值滤波器用邻域内像素亮度的中值来取代原本的像素值，即：

$$\bar{I}(x, y) = \text{Median} I(x + u, y + v) | (u, v) \in N$$

由定义可见中值滤波器是非线性的

该节实现了名为 `MedianFilter` 的程序，用法如下：

```
MedianFilter <input-image> <output-image> <w> <h>
```

参数 `w` 和 `h` 的含义与均值滤波中对应参数的含义相同

具体来说我的代码实现如下：

```
void MedianFilter(const Mat& input, Mat& output, int w, int h) {
    output = input.clone();
    for (int i = 0; i < input.rows; i++) {
        for (int j = 0; j < input.cols; j++) {
            // 对每个颜色通道分别处理
            for (int c = 0; c < 3; c++) {
                std::vector<int> neighborhood;
                for (int dy = -h; dy <= h; dy++) {
                    for (int dx = -w; dx <= w; dx++) {
                        if (i + dy >= 0 && i + dy < input.rows && j + dx >= 0 && j + dx < input.cols) {
                            neighborhood.push_back(input.at<uchar>(i + dy, j + dx));
                        }
                    }
                }
                sort(neighborhood.begin(), neighborhood.end());
                output.at<uchar>(i, j, c) = neighborhood[(neighborhood.size() - 1) / 2];
            }
        }
    }
}
```

```

        int ny = i + dy;
        int nx = j + dx;
        if (nx >= 0 && nx < input.cols && ny >= 0 && ny <
input.rows) {
            Vec3b color = input.at<Vec3b>(ny, nx);
            neighborhood.push_back(color[c]);
        }
    }
    std::nth_element(neighborhood.begin(), neighborhood.begin() +
neighborhood.size() / 2, neighborhood.end());
    int median = neighborhood[neighborhood.size() / 2];
    output.at<Vec3b>(i, j)[c] = static_cast<uchar>(median);
}
}
}

```

实验效果如下（原图含有椒盐噪声）



原图

1 × 1

3 × 3

9 × 9

可以发现，中值滤波可以很好地过滤椒盐噪声，更大的邻域使得图像边界变得圆滑

并且验证发现前面两种滤波都无法处理椒盐噪声（均值选取  $h = w = 1$ , 高斯的  $\sigma = 3$ ）

其余的两种方法都可以看到有明显的斑点



原图

$1 \times 1$  中值

1 × 1 均值

Sigma=3高斯

课程网站的示例滤波后结果如下：



带椒盐噪声的原图



中值滤波后

## 实现简单的双边滤波

前面的三种滤波器都会破坏图像的边界，在卷积核很大的时候，均值滤波和高斯滤波都会让边界变得模糊，在邻域很大时，中值滤波会减小边界的曲率。由于物体边界是物体的一个重要特征，很多任务里我们不希望图像边界被破坏。

计算方式如下：

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

其中， $S$ 为邻域， $G_{\sigma_s}$ 为空间域的高斯函数， $G_{\sigma_r}$ 为灰度域的高斯函数， $W_p$ 为归一化系数：

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

双边滤波提供了一种降噪同时保持边界的方法。它的思路很简单：如果邻域内像素的亮度差异很大，它在加权平均时的贡献也应当小。我们可以在高斯滤波加权平均的基础上引入一个新的项，反应亮度差带来的加权

为什么为什么要有 $W_p$ ？两个高斯核还需要 $\frac{1}{2\pi\sigma^2}$ 的归一项么？

用于值得归一化；不需要 $\frac{1}{2\pi\sigma^2}$ 的归一项，因为 $W_p$ 已经起到了过滤的作用

本次实验实现名为 `BilateralFilter` 的程序，用法如下：

```
BilateralFilter <input-image> <output-image> <sigma-s> <sigma-r>
```

参数 `<sigma-s>` 和 `<sigma-r>` 分别对应 $\sigma_s$ 和 $\sigma_r$ ：

具体实现代码如下：

```
double PI = 3.1415926535;
double Gaussian(double sigma, double x) {
    return exp(-pow(x, 2) / (2 * pow(sigma, 2))) / (sigma * sqrt(2 * PI));
}
double GetDistance(int x1, int y1, int x2, int y2) {
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}
void BilateralFilter(const Mat& input, Mat& output, double sigmaS, double sigmaR)
{
    CV_Assert(input.depth() == CV_8U); // 确保图像深度为8位
    output = Mat(input.size(), input.type());
    int w = 5; // 半径窗口大小
    int d = 2 * w + 1; // 窗口直径
```

```

// 对每个像素应用双边滤波
for (int i = 0; i < input.rows; i++) {
    for (int j = 0; j < input.cols; j++) {
        double weightSum = 0.0;
        Vec3d Isum(0, 0, 0);

        for (int ii = -w; ii <= w; ii++) {
            for (int jj = -w; jj <= w; jj++) {
                int x = j + jj;
                int y = i + ii;

                if (x >= 0 && x < input.cols && y >= 0 && y < input.rows) {
                    Vec3d diff = input.at<Vec3b>(y, x) - input.at<Vec3b>(i,
j);
                    double ds = exp(-0.5 * (ii * ii + jj * jj) / (sigmas *
sigmas));
                    double dr = exp(-0.5 * (diff.dot(diff)) / (sigmaR *
sigmaR));
                    double weight = ds * dr;

                    Isum += weight * input.at<Vec3b>(y, x);
                    weightSum += weight;
                }
            }
        }
        Vec3b newVal = (1.0 / weightSum) * Isum;
        output.at<Vec3b>(i, j) = newVal;
    }
}
}

```

下列是双边滤波的效果，括号后面的数值是每个函数的可变参数的值，可以发现双边滤波的轮廓是最为清晰的



可以发现随着 $\sigma_s$ 与 $\sigma_r$ 数值的增长，图片的清晰度有所降低，但是轮廓依然非常清晰（看起来有点像是美颜）



# 利用傅里叶变换完成图像的频域变换

复现对一个单通道、中心放置一个矩形的图像进行傅里叶变换的实验

对于大小为  $W \times H$  的图像，其二维傅立叶变换定义为：

$$\hat{I}(\omega_1, \omega_2) = \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} I(x, y) e^{-j2\pi(\frac{x\omega_1}{W} + \frac{y\omega_2}{H})}$$

上式定义在复数域中，因此傅里叶变换的输入和输出都是复数域上的。但我们用实数域表示图像，因此在操作时要额外注意类型：变换到频域时，采用复数表达，变换回图像时，只保留实数部分。

在 OpenCV 中，图像的傅里叶变换可以用 `cv::dft` 函数完成

为了简化实验，我们只使用单通道图像。我们首先创建一幅 512x512 的单通道浮点数图像 `I`，在中心放置一个 20x60 的矩形

```
Mat I(512, 512, CV_32FC1);
I(Rect(256-10, 256-30, 20, 60)) = 1.0;
```

我们将其进行傅里叶变换，得到 512x512 的复 (CV\_32FC2) 矩阵 `J`，然后利用 `fftshift` 函数将零频率放置到中心。

```
Mat J(I.size(), CV_32FC2);
dft(I, J, DFT_COMPLEX_OUTPUT);
fftshift(J, J);
```

`fftshift` 函数利用给出的代码将零频率移到中央

我们计算 `J` 的每个元素的模长：

```
Mat Mag;
vector<Mat> K;
split(J, K); // 将实数和虚数部分分解到 K[0] 和 K[1]
pow(K[0], 2, K[0]); // 计算平方
pow(K[1], 2, K[1]);
Mag = K[0]+K[1]; // 两个分量的平方和
```

进行对数变换

```
Mat logMag;
log(Mag+1, logMag);
```

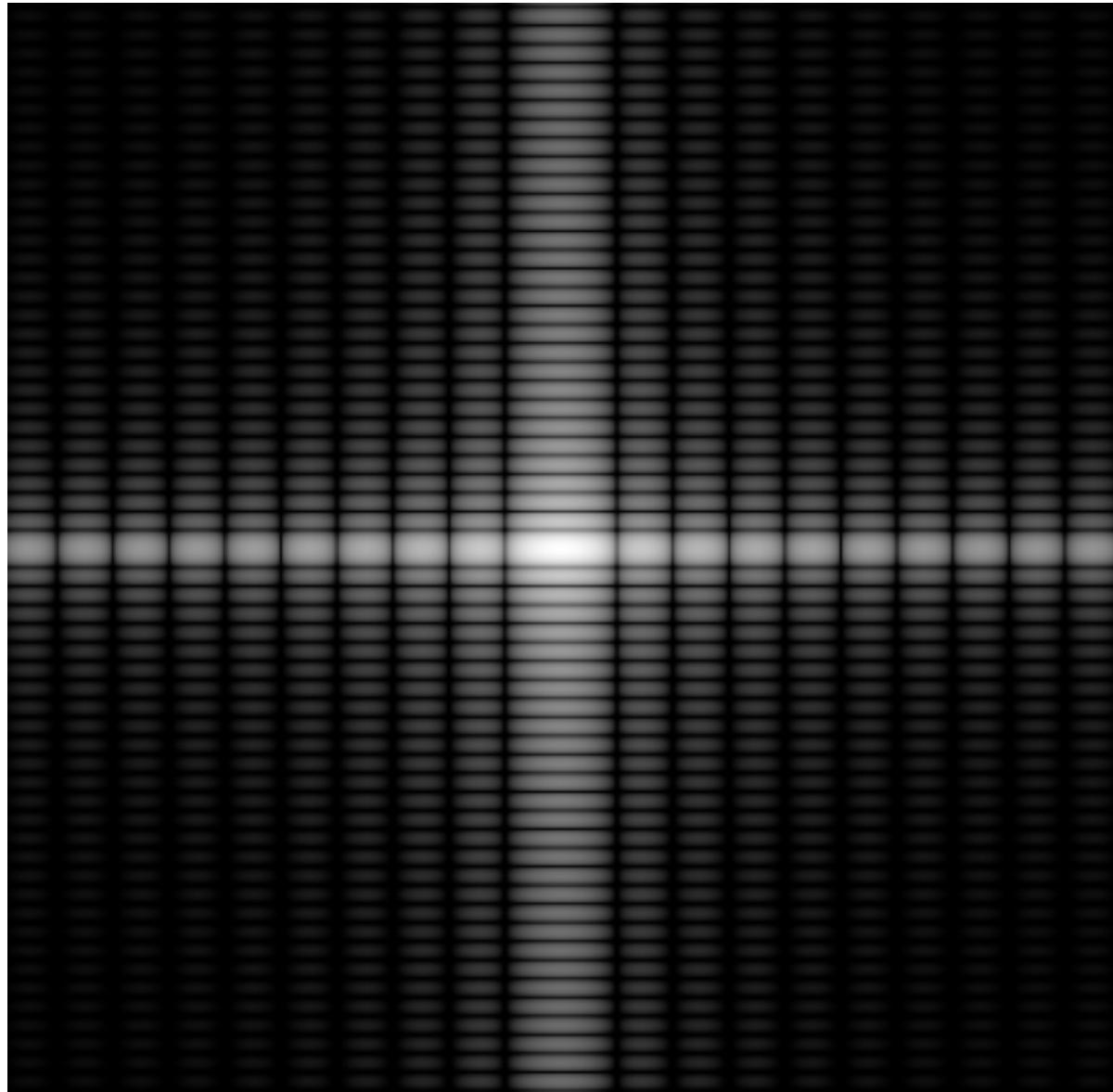
归一化

```
normalize(logMag, logMag, 1.0, 0.0, NORM_MINMAX);
```

最后可视化显示

```
imshow("Magnitude", logMag);
```

显示的结果如下：



## 代码说明

我写了两份 .cpp 代码，内容如下：

```
demo # exe存储的文件夹  
filter.cpp # 滤波的代码  
fourier.cpp # 傅里叶变换的代码
```

filter.cpp 为要求实现的4种滤波方式的代码，我把4种滤波方法都封装为了函数，函数的接口与实验文档一致，具体的每个函数已经在前面给出，下面是主体的代码

```
# filter.cpp  
#include <opencv2/opencv.hpp>  
#include <math.h>  
#include <iostream>  
  
using namespace cv;  
  
void BoxFilter(const Mat& input, Mat& output, int w, int h) {  
    // ....  
}  
void GaussianFilter(const Mat& input, Mat& output, double sigma) {
```

```

    // ....
}

void MedianFilter(const Mat& input, Mat& output, int w, int h) {
    // ....
}

void BilateralFilter(const Mat& input, Mat& output, double sigmas, double sigmaR)
{
    // ....
}

int main(int argc, char* argv[]) {
    std::string inname, outname;
    int w, h;
    double sigma, sigma_s, sigma_r;

    char mode;
    std::cout << "choose a mode \n\b\b b:BoxFilter;\n\b\b g:GaussianFilter;\n\b\b m:MedianFilter;\n\b\b x: BilateralFilter\n please type a char here: ";
    std::cin >> mode;

    char choice;
    std::cout << "Do you want to use default settings? (y/n): ";
    std::cin >> choice;

    // 使用默认参数或者读入参数
    if (choice == 'y' || choice == 'Y') {
        inname = "test4.jpg";
        outname = "output.jpg";
        w = 9;
        h = 9;
        sigma = 9.0;
        sigma_s = 36.0;
        sigma_r = 36.0;
    }
    else if (mode == 'b' || mode == 'B' || mode == 'm' || mode == 'M') {
        std::cout << "Enter input image name(path): ";
        std::cin >> inname;
        std::cout << "Enter output image name(path): ";
        std::cin >> outname;
        std::cout << "Enter width(double): ";
        std::cin >> w;
        std::cout << "Enter height(double): ";
        std::cin >> h;
    }
    else if (mode == 'g' || mode == 'G') {
        std::cout << "Enter input image name(path): ";
        std::cin >> inname;
        std::cout << "Enter output image name(path): ";
        std::cin >> outname;
        std::cout << "Enter sigma(double): ";
        std::cin >> sigma;
    }
    else if (mode == 'x' || mode == 'X') {
        std::cout << "Enter input image name(path): ";
        std::cin >> inname;
        std::cout << "Enter output image name(path): ";
    }
}

```

```

        std::cin >> outname;
        std::cout << "Enter sigma s(double): ";
        std::cin >> sigma_s;
        std::cout << "Enter sigma R(double): ";
        std::cin >> sigma_r;
    }
else {
    std::cout << "Invalid mode selected." << std::endl;
    return 1;
}

// 读取图像
Mat image = imread(inname);
if (image.empty()) {
    std::cerr << "Error: Could not read the image file." << std::endl;
    return 1;
}
Mat outimage;

// 进行滤波
if (mode == 'b' || mode == 'B') {
    BoxFilter(image, outimage, w, h);
}
else if (mode == 'g' || mode == 'G') {
    GaussianFilter(image, outimage, sigma);
}
else if (mode == 'm' || mode == 'M') {
    MedianFilter(image, outimage, w, h);
}
else if (mode == 'x' || mode == 'X') {
    BilateralFilter(image, outimage, sigma_s, sigma_r);
}

// 显示原始图像和处理后的图像
imshow("Original", image);
imshow("Filtered", outimage);
// 保存处理后的图像
imwrite(outname, outimage);
// 关闭所有窗口
waitKey(0);
destroyAllWindows();
return 0;
}

```

`fourier.cpp` 即傅里叶变换那一部分的代码，没有额外需要强调的点

## 运行环境

### 滤波

对于滤波，运行 demo 文件下的 `filter.exe` 即可（可能需要 windows 环境），刚开始需要选择滤波的种类，然后是参数，如果使用默认参数，默认参数的内容如下：

```
inname = "test4.jpg";
outname = "output.jpg";
w = 9;
h = 9;
sigma = 9.0;
sigma_s = 36.0;
sigma_r = 36.0;
```

运行的情况比如如下



```
C:\Users\Administrator\Desktop\计算摄影学\lab2-3210103790_周炜\demo\fliter.exe
choose a mode
b:BoxFilter;
g:GaussianFilter;
m:MedianFilter;
x: BilateralFilter
please type a char here: x
Do you want to use default settings? (y/n): y
-
```

等待一会之后就可以跳出滤波前和滤波后的图片了

输出的图片为被命名为 output.jpg

## 傅里叶变换

运行 demo 文件下的 fourier.exe 即可(可能需要windows环境), 就会出现如上面描述的图片