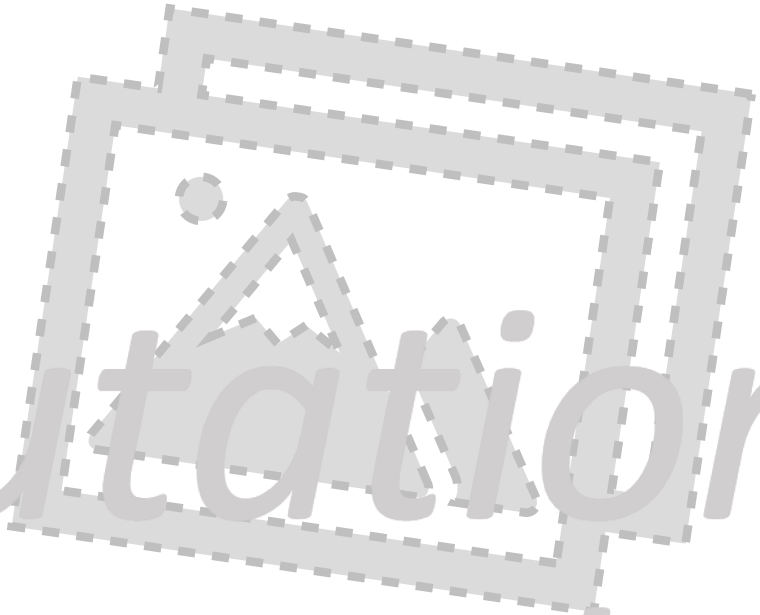
LAB02
图像滤波和傅里叶变换

计算摄影学 2025春夏

2025/2/25



Computation Photography

BEFORE

- 本次实验内容需要在学在浙大上提交作业，本次作业计入成绩。
- 提交要求：
  - 将**源代码、可执行程序和实验文档**打包
  - 请在 README或可执行程序的文件名中注明可执行程序的**运行环境。**
  - 压缩包名称为 Lab2-学号-姓名.zip/7z。
  - 在**3月10日23:59:59**前提交至学在浙大。

实验任务

- **空域滤波**：
  - 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
- **傅里叶变换：**
  - 完成图像的频域变换

均值滤波

均值滤波以像素邻域内的平均值代替原先的像素，即：

$$\bar{I}(x,y)=\frac{1}{|N|}\sum_{(u,v)\in N} I(x+u,y+v) \tag{1}$$

引入卷积核 K，

$$K(u,v)=\begin{cases}\frac{1}{|N|} & \text{if}(u,v)\in N,\\ 0 & \text{else.}\end{cases} \tag{2}$$

则均值滤波可写为

$$\bar{I}(x,y)=\sum_{u,v} K(u,v)I(x+u,y+v) \tag{3}$$

卷积核K与图像域坐标(x, y)无关，因此我们说均值滤波是**线性的**。

均值滤波(Cont.)

- 在 OpenCV 中实现均值滤波，我们可以依照(1)朴素实现；
- 本实验要求计算(2)的卷积核并依照(3)进行卷积。

- OpenCV 中图像的卷积计算： cv::filter2D

  图像通常是有边界的， filter2D 函数可以处理边界。通过指定 cv::BorderType 可以指定处理边界的方法。默认方法是 BORDER_REFLECT_101 ，即以边界像素为轴进行对称拓展。

均值滤波(Cont.)

- 在 OpenCV 中实现均值滤波，我们可以依照(1)朴素实现；
- 本实验要求计算 (2) 的卷积核并依照(3)进行卷积。

- OpenCV 中图像的卷积计算： cv::filter2D
- 对于矩形卷积核，可以使用 cv:: boxFilter 实现。**本次实验要求手工实现一个名为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)。

高斯滤波

均值滤波均等地把空域内的像素和目标像素联系起来，而高斯滤波图像则体现了像素之间的相关性随着距离增加不断减弱。

高斯滤波的卷积核是

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

其中

$$G(u,v) = \frac{1}{2\pi\sigma^2} \exp - \frac{u^2+v^2}{2\sigma^2}$$

高斯滤波(Cont.)

本次实验要求手工实现一个名为GaussianFilter的可执行程序，
用法如下：

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

高斯核理论上大小是无穷的，在实现时可以只保留中心 5σ 的区域，即卷积核矩阵大小为 (2*[5σ]+1)×(2*[5σ]+1)。

中值滤波

高斯滤波适用于带有高斯噪声的图像，在某些非高斯噪声污染情况下不一定能得到理想效果，如椒盐噪声。

椒盐噪声 (Salt and Pepper Noise) 是一种常见的噪声类型，表现为随机出现的黑点(pepper,值为0)和白点(salt,值为255)；噪声像素与原始图像内容完全无关,随机分布。椒盐噪声是替换性噪声，受影响像素的分布更加离散。

中值滤波是处理椒盐噪声的一种好方法。

中值滤波(Cont.)

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。

中值滤波器用邻域内像素亮度的**中值**取代原本的像素值，从而抑制椒盐噪声这种极值噪声。可以和 Pooling 对比理解。中值滤波可以这样表述：

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

中值滤波是非线性的。**本次实验要求手工实现一个名为 `MedianFilter` 的可执行程序**，用法如下：

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

中值滤波(Cont.)

你可以通过 cv::copyMakeBorder 手动处理边界。为了方便，本实验只要求实现处理黑白图像。如果该程序能处理RGB图像，会得到本次实验额外的加分。

双边滤波

均值滤波、高斯滤波和中值滤波都会影响图像中存在的边界。双边滤波通过一种简单的思路，在抑制噪声的同时保持边界：

邻域内亮度差异较大的像素在加权平均时的贡献较小。

通过对高斯滤波的加权中引入一个新项，就可以同时反应像素间的相关性和距离、亮度差异的关系：

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

双边滤波(Cont.)

通过对高斯滤波的加权中引入一个新项，就可以同时反应像素间的相关性和距离、亮度差异的关系：

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

显然 $G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I(p) - I(q)|)$ 是权重，因而有归一化因子

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

本次实验要求手工实现名为 BilateralFilter 的程序，用法如下：

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

双边滤波(Cont.)

原图

双边滤波

高斯滤波

对于双边滤波，较大的领域会降低处理速度，你可以自行试验出合适的大小。

傅里叶变换

RECALL

一维傅里叶变换正变换： $F(u) = \int_{-\infty}^{\infty} f(x)e^{(-j2\pi ux)}\,\mathrm{d}x$

逆变换： $f(x) = \int_{-\infty}^{\infty} F(u)e^{(j2\pi ux)}\,\mathrm{d}u$

二维傅里叶变换正变换： $F(u,v) = \int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}\,\mathrm{d}x\mathrm{d}y$

逆变换： $f(x,y) = \int_{-\infty}^{\infty} F(u,v)e^{j2\pi(ux+vy)}\,\mathrm{d}u\mathrm{d}v$

f(·) 是实数域的原始信号，F(·) 是频率域的变换结果 。

对于大小为W*H的图像I，离散傅里叶变换(DFT)定义为：

$$\hat{\mathrm{I}}(\omega_1,\omega_2) = \sum_{\mathrm{x}=0}^{\mathrm{W}-1}\sum_{\mathrm{y}=0}^{\mathrm{H}-1} \mathrm{I}(\mathrm{x},\mathrm{y})\mathrm{e}^{-\mathrm{j}2\pi\left(\frac{\mathrm{x}\omega_1}{\mathrm{W}}+\frac{\mathrm{y}\omega_2}{\mathrm{H}}\right)}$$

在 OpenCV 中，图像的傅里叶变换可以使用 cv::dft 函数。

```
Mat dft_result(image.size(), CV_32FC2);
dft(image, dft_result, DFT_COMPLEX_OUTPUT);
```

傅里叶变换(Cont.)

变换后的结果中，左上角对应了零频率。在可视化时我们希望将零频率置于中心。我们提供函数 fftshift 将零频率移到中央 。

傅里叶变换(Cont.)

```
void fftshift(const Mat &src, Mat &dst) {
	dst.create(src.size(), src.type());
	int rows = src.rows, cols = src.cols;
	Rect roiTopBand, roiBottomBand, roiLeftBand, roiRightBand;
	if (rows % 2 == 0) {
		roiTopBand = Rect(0, 0, cols, rows / 2);
		roiBottomBand = Rect(0, rows / 2, cols, rows / 2);
	}
	else {
		roiTopBand = Rect(0, 0, cols, rows / 2 + 1);
		roiBottomBand = Rect(0, rows / 2 + 1, cols, rows / 2);
	}
	if (cols % 2 == 0) {
		roiLeftBand = Rect(0, 0, cols / 2, rows);
		roiRightBand = Rect(cols / 2, 0, cols / 2, rows);
	}
```

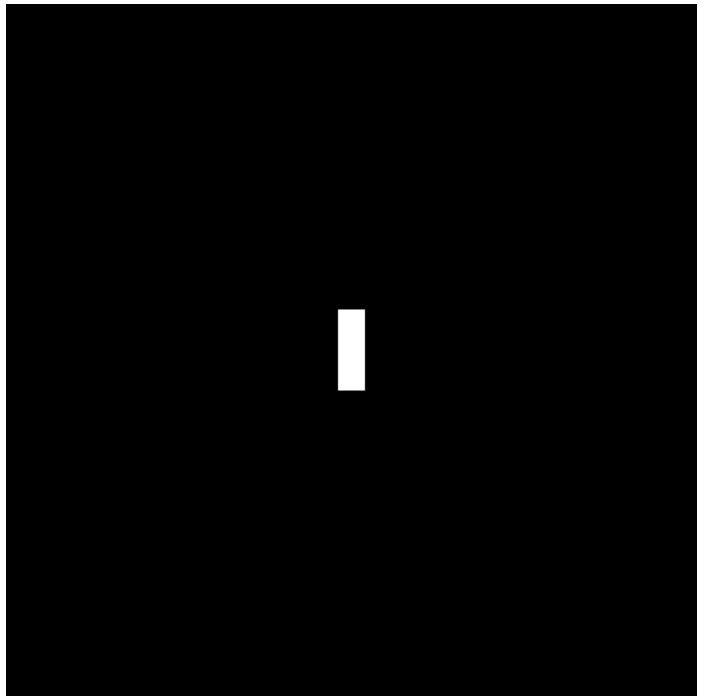
傅里叶变换(Cont.)

```
        else  {
               roiLeftBand =  Rect(0,  0, cols /  2  +  1, rows);
               roiRightBand =  Rect(cols /  2  +  1,  0, cols /  2, rows);
        }
        Mat srcTopBand =  src(roiTopBand);
        Mat dstTopBand =  dst(roiTopBand);
        Mat srcBottomBand =  src(roiBottomBand);
        Mat dstBottomBand =  dst(roiBottomBand);
        Mat srcLeftBand =  src(roiLeftBand);
        Mat dstLeftBand =  dst(roiLeftBand);
        Mat srcRightBand =  src(roiRightBand);
        Mat dstRightBand =  dst(roiRightBand);
        flip(srcTopBand, dstTopBand,  0);
        flip(srcBottomBand, dstBottomBand,  0);
        flip(dst, dst,  0); flip(srcLeftBand, dstLeftBand,  1);
        flip(srcRightBand, dstRightBand,  1); flip(dst, dst,  1);
}
```

傅里叶变换(Cont.)

实验使用单通道图像。通过下面的代码创建中心有 20x60 矩形的 512x512 单通道浮点数图像 I：

```
Mat I(512, 512, CV_32FC1);
I = 0;
I(Rect(256-10, 256-30, 20, 60)) = 1.0;
```



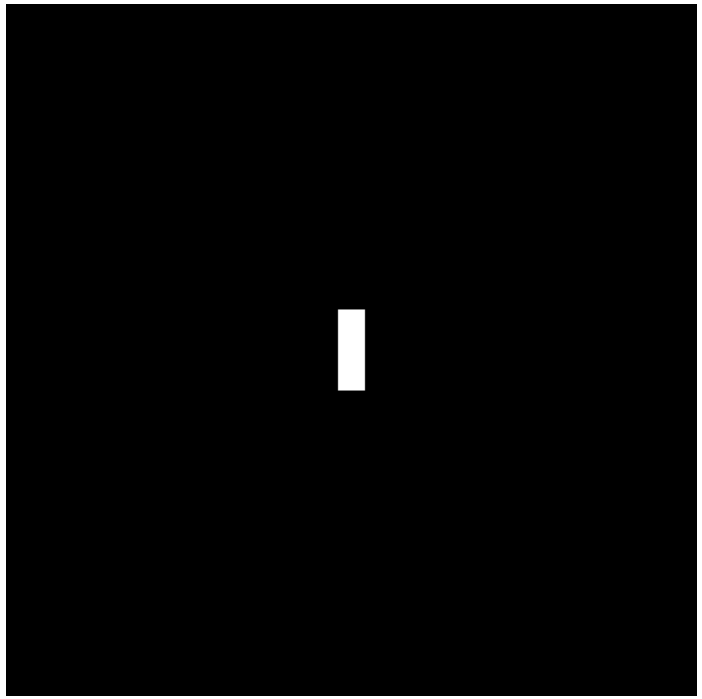
并进行傅里叶变换，把零频率放到中心：

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

傅里叶变换(Cont.)

计算复矩阵 J 的每个元素的模长，并进行可视化：

```
Mat Mag;
vector<Mat> K;
split(J, K); // 分解实数和虚数部分
pow(K[0], 2, K[0]);
pow(K[1], 2, K[1]);
Mag = K[0]+K[1];
Mat logMag;
log(Mag+1, logMag);
normalize(logMag, logMag, 1.0, 0.0,
NORM_MINMAX);
// ...
imshow("Magnitude", logMag);
```

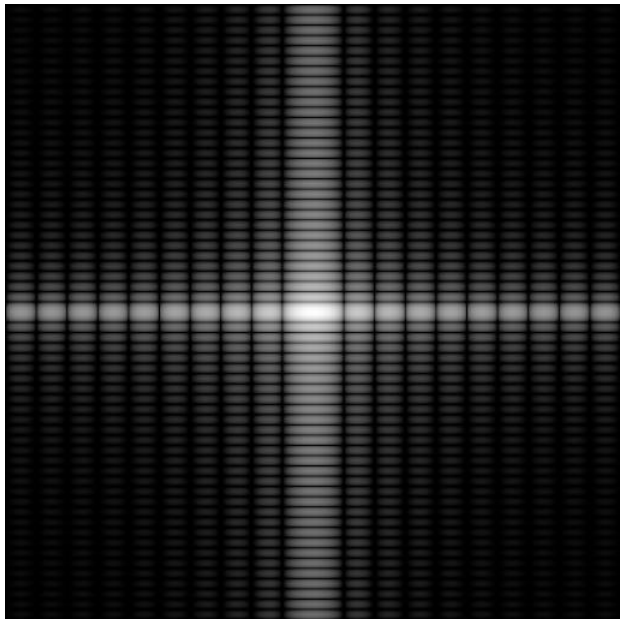
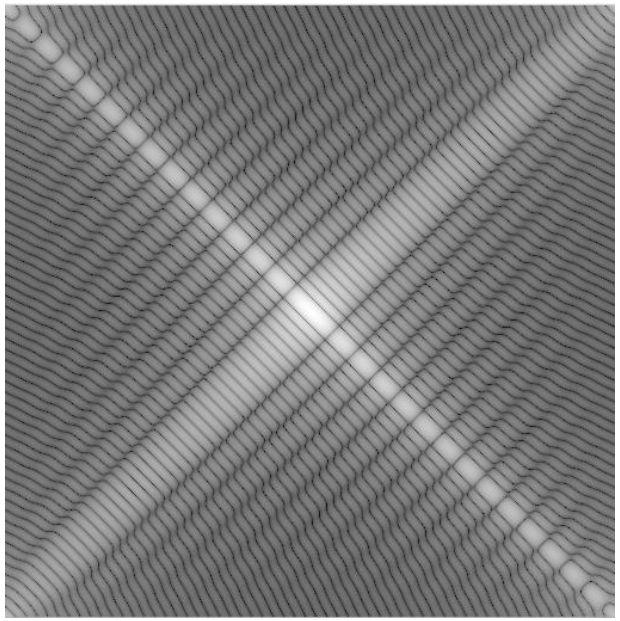
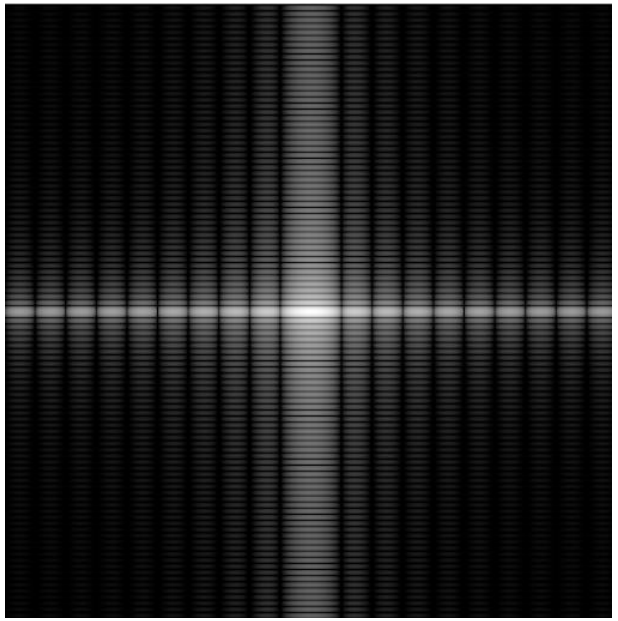


傅里叶变换(Cont.)

1. 对DFT得到的结果进行 DFT 变换，并展示。
2. 如果将图像 I 中心的矩形分别进行**旋转、平移、修改矩形的长和宽**三种操作，结果会怎么变化？给出答案并修改代码得到可视化结果。
3. 对 2 中DFT得到的结果进行DFT变换，展示并进行分析。

如果有兴趣，你可以在此基础上完成高通和低通滤波。

预期结果

- 本次实验**需要提交源代码、可执行程序和实验文档。**
- **确保你的可执行程序名正确，以确保能顺利通过自动化测试。**

TIPS

- 怎么手动处理图像边界？
  - HINT：copyMakeBorder
- 双边滤波器在卷积核较大时运行较慢，怎么选择合适的大小？
- 怎么绘制旋转的矩形？
  - HINT：RotatedRect

Questions are welcome

计算摄影学 2025春夏

2025/2/25

Computation Photography