计算机视觉

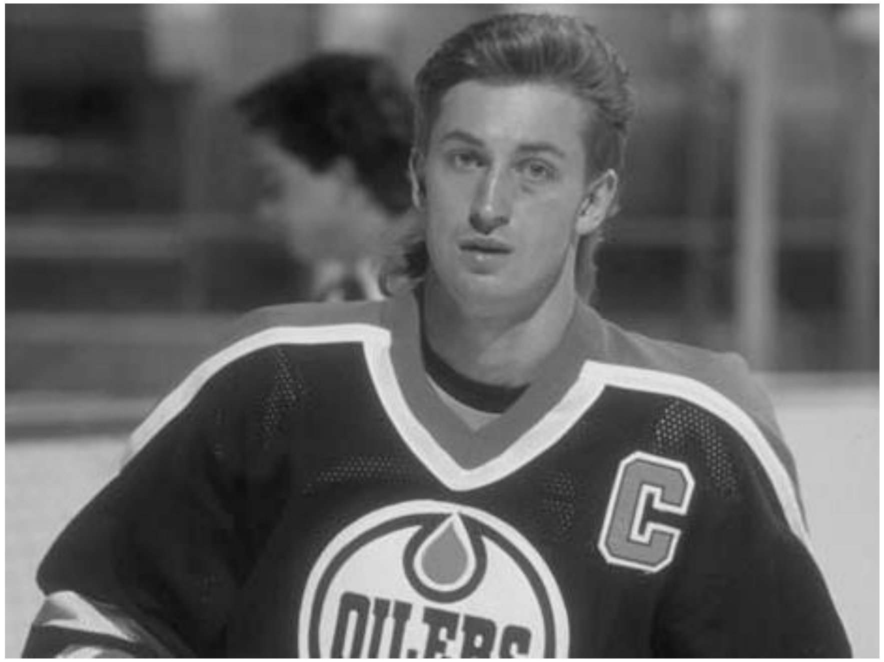
频率分析 上

中国传媒大学
COMMUNICATION UNIVERSITY OF CHINA

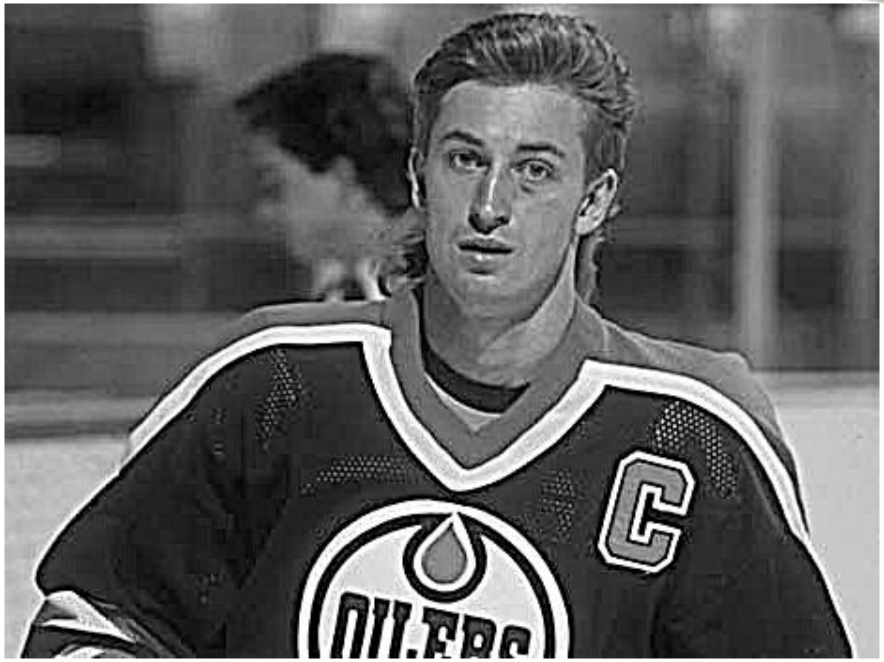
本节主题：

傅里叶变换

图像锐化

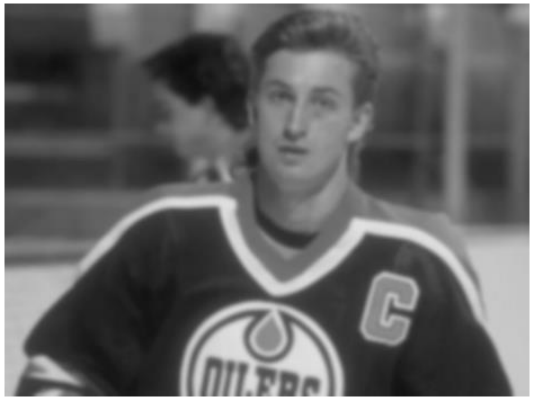


输入

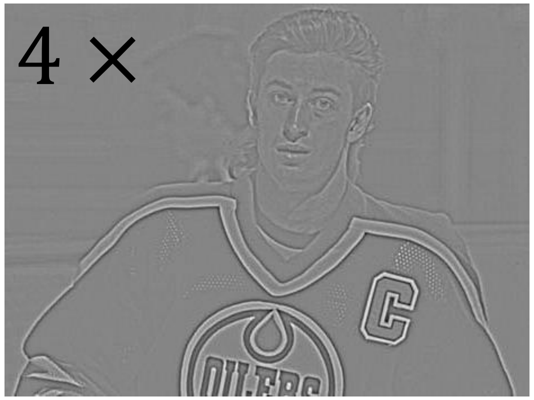


锐化的

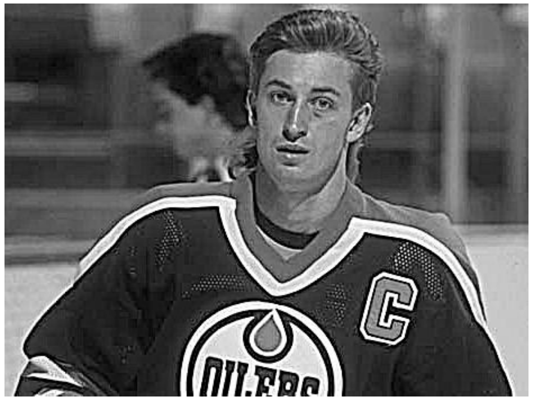
图像锐化



+



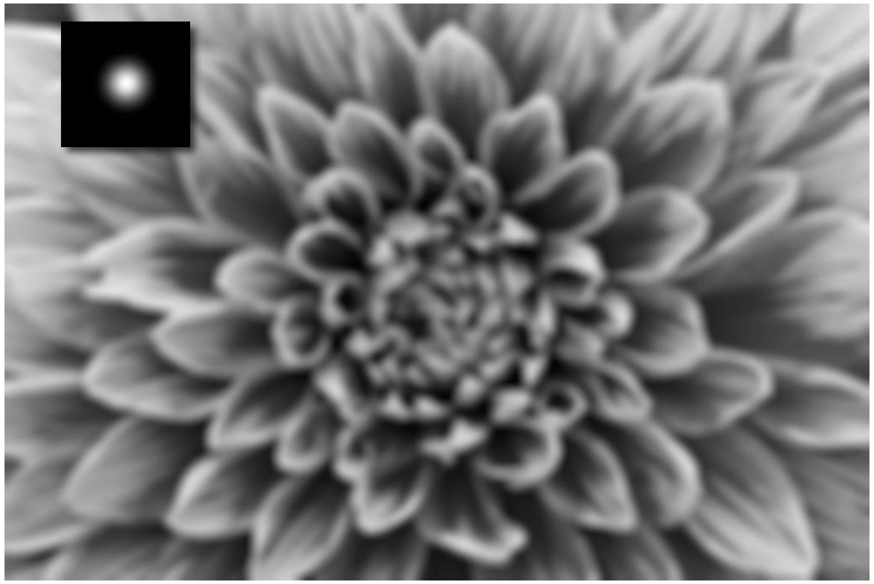
=

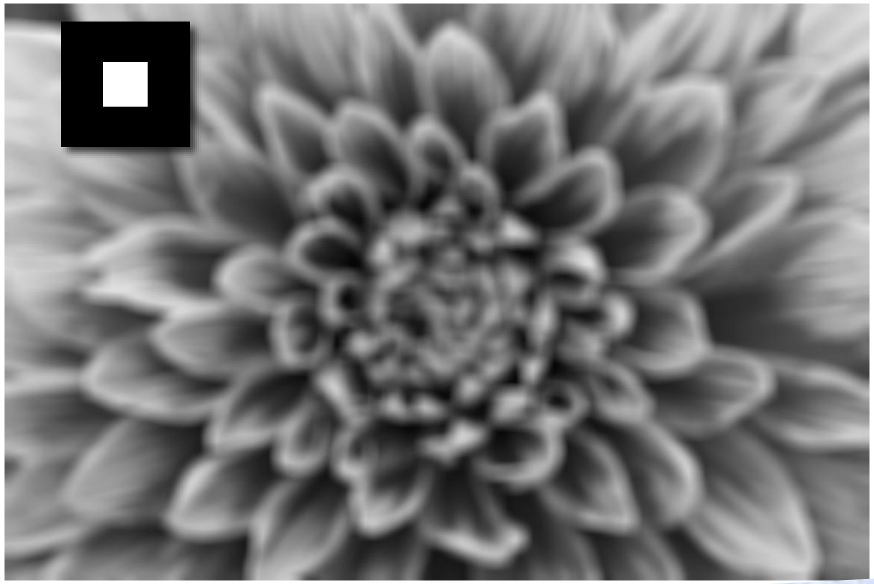


模糊的

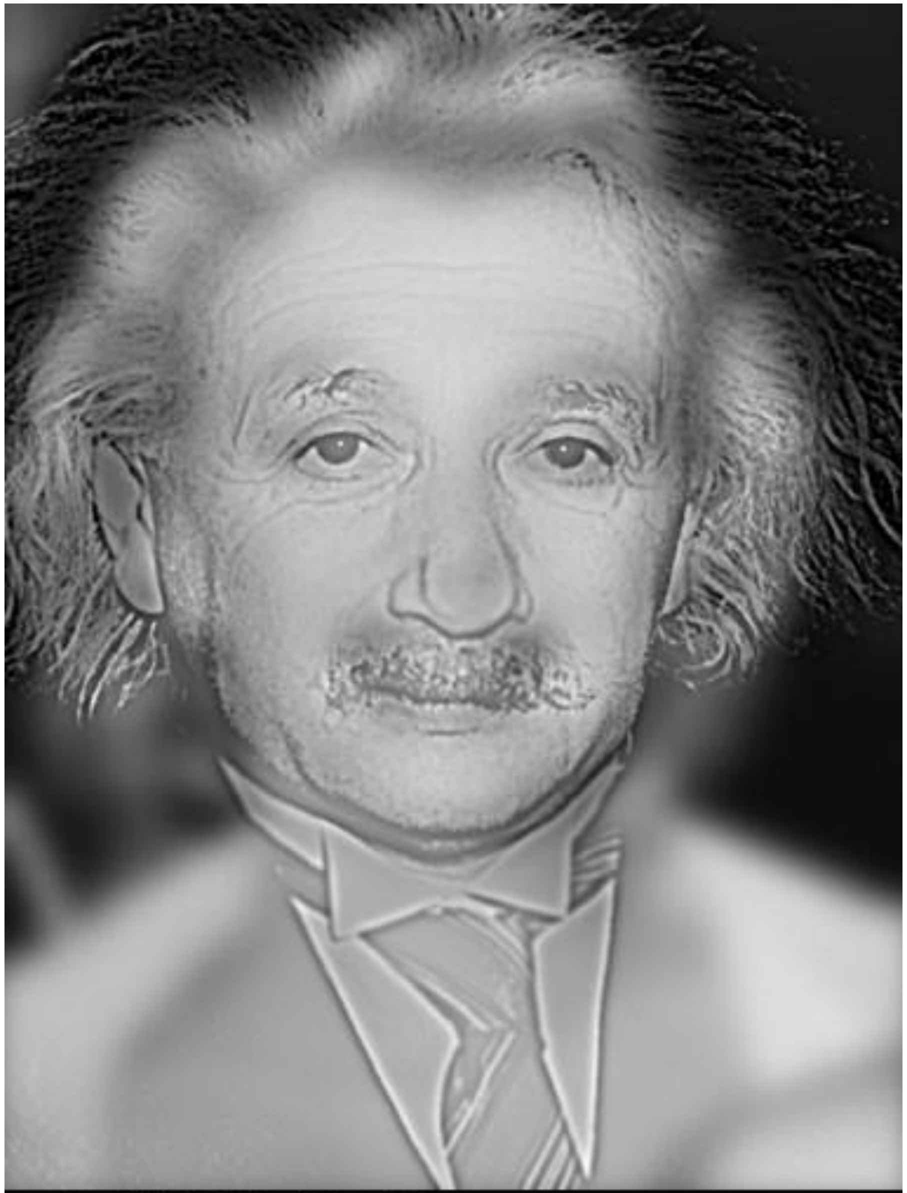
“锐利的东西”

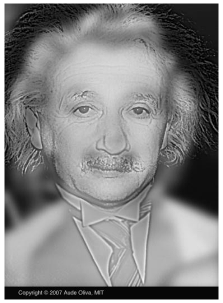
锐化的

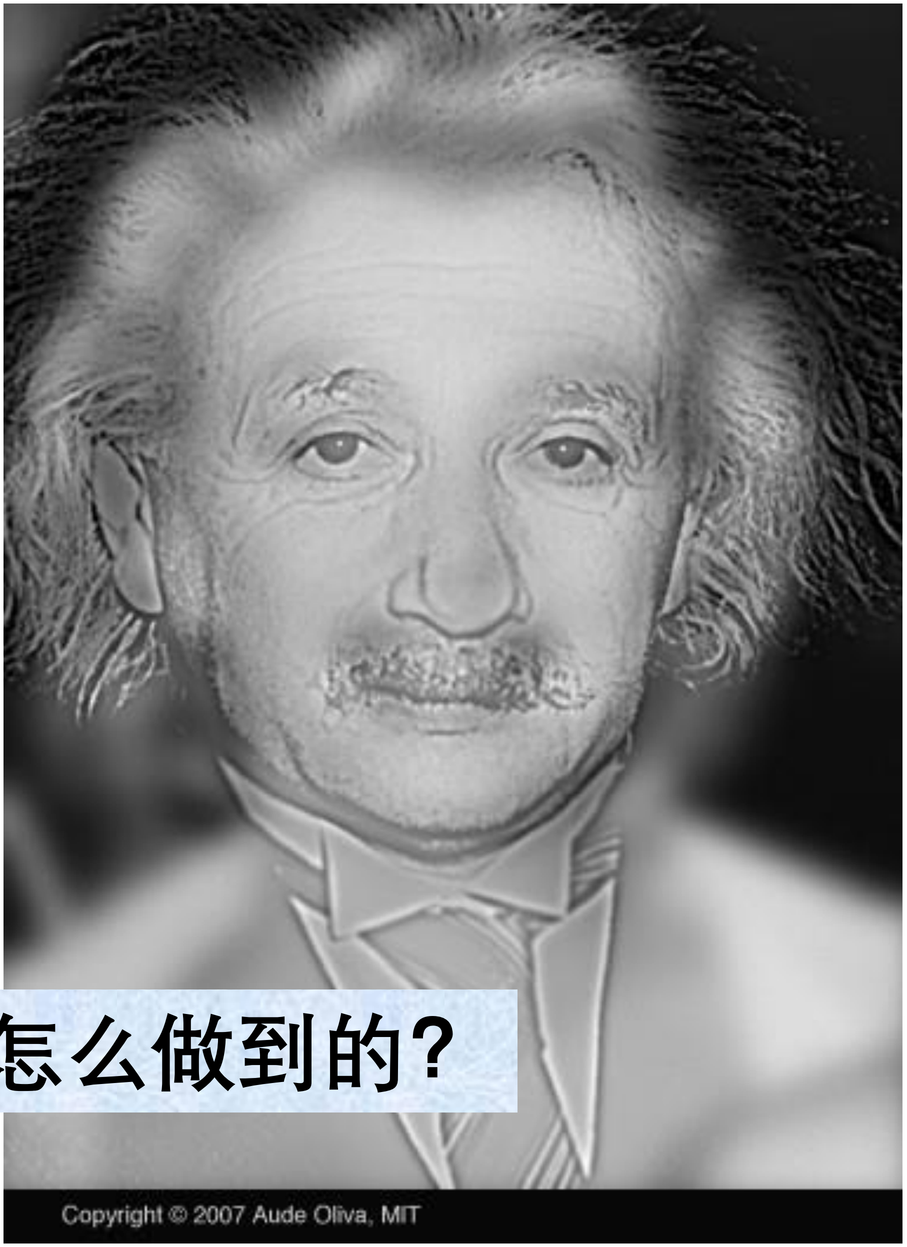




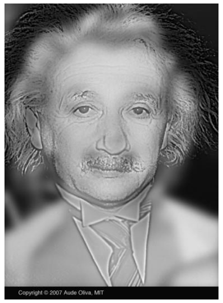
为什么高斯滤波后的图像比方框滤波后的图像更平滑?

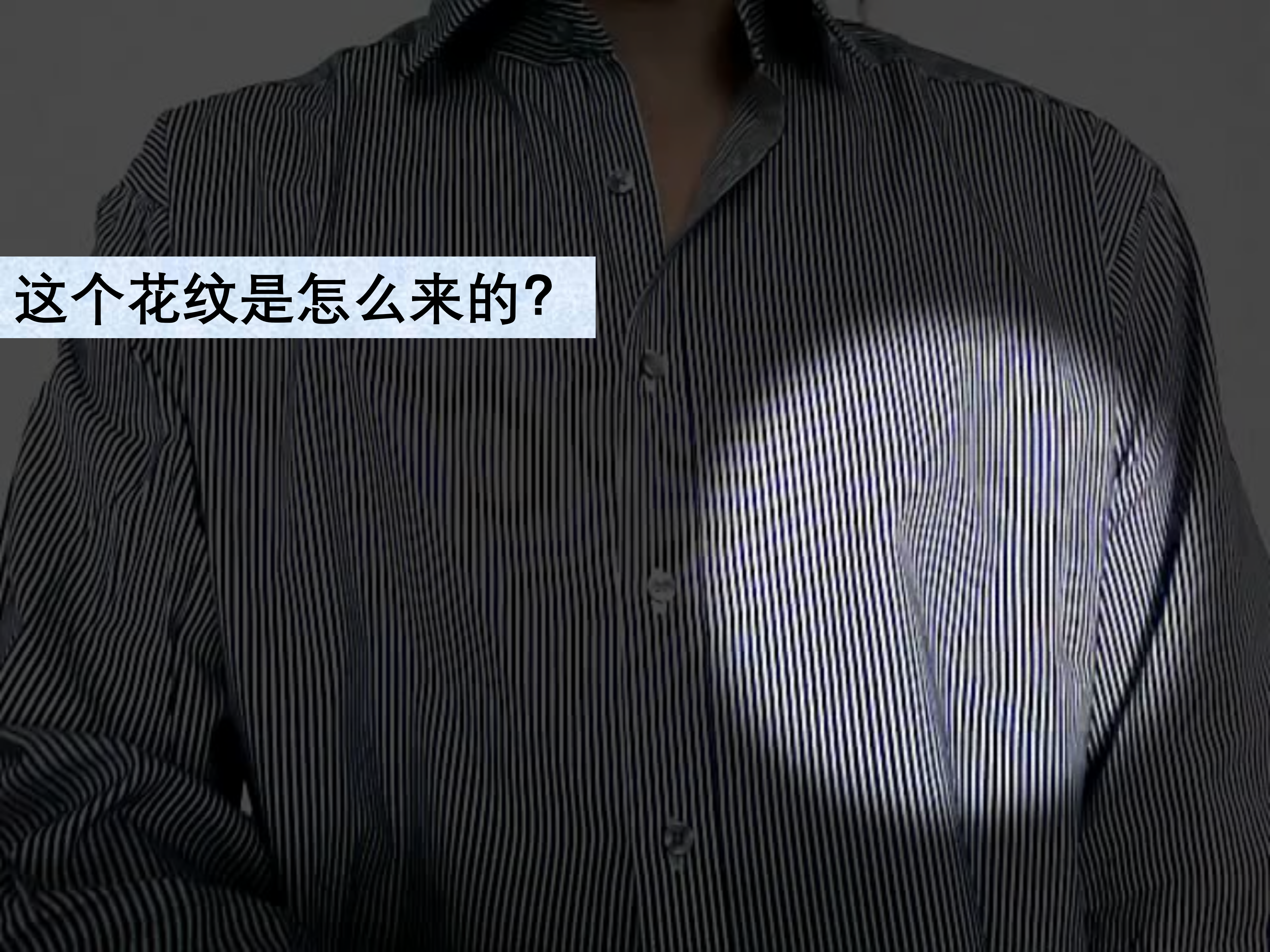



Copyright © 2007 Aude Oliva, MIT

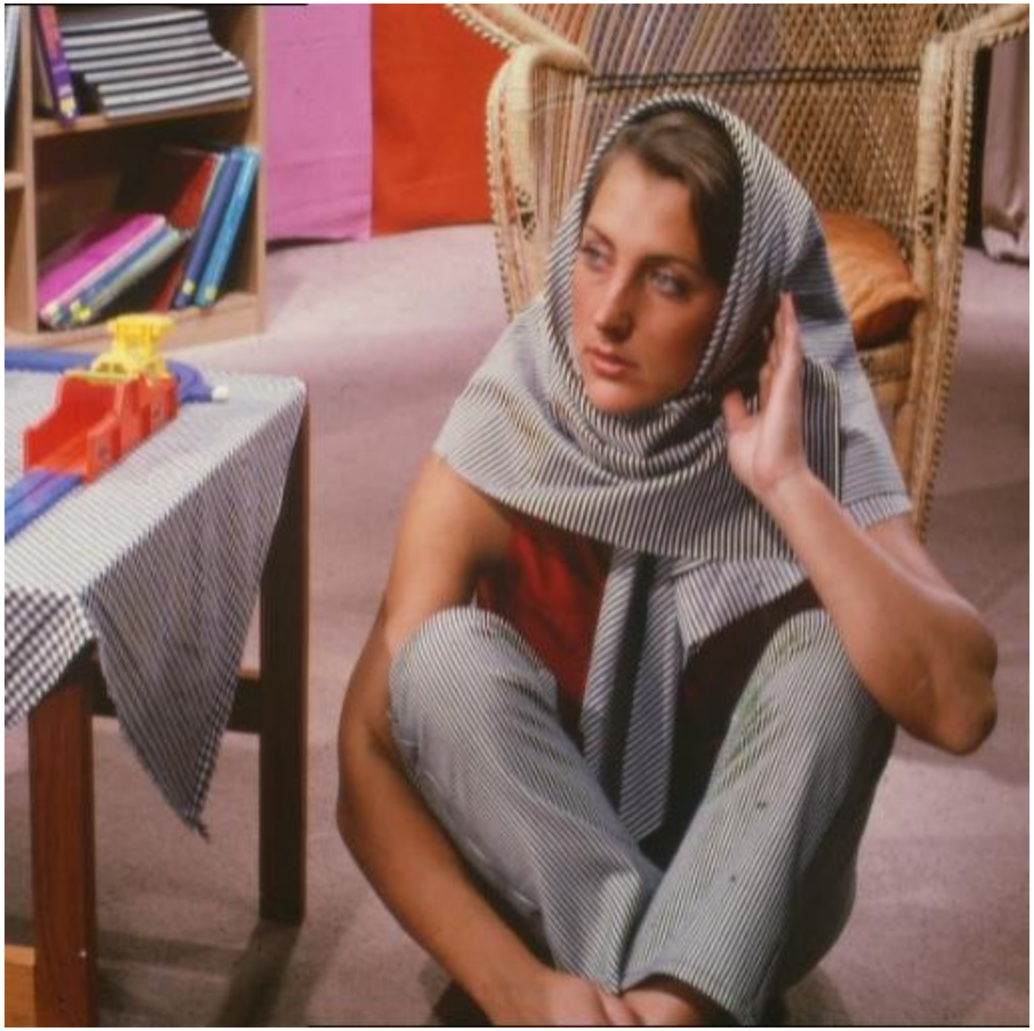

Copyright © 2007 Aude Oliva, MIT

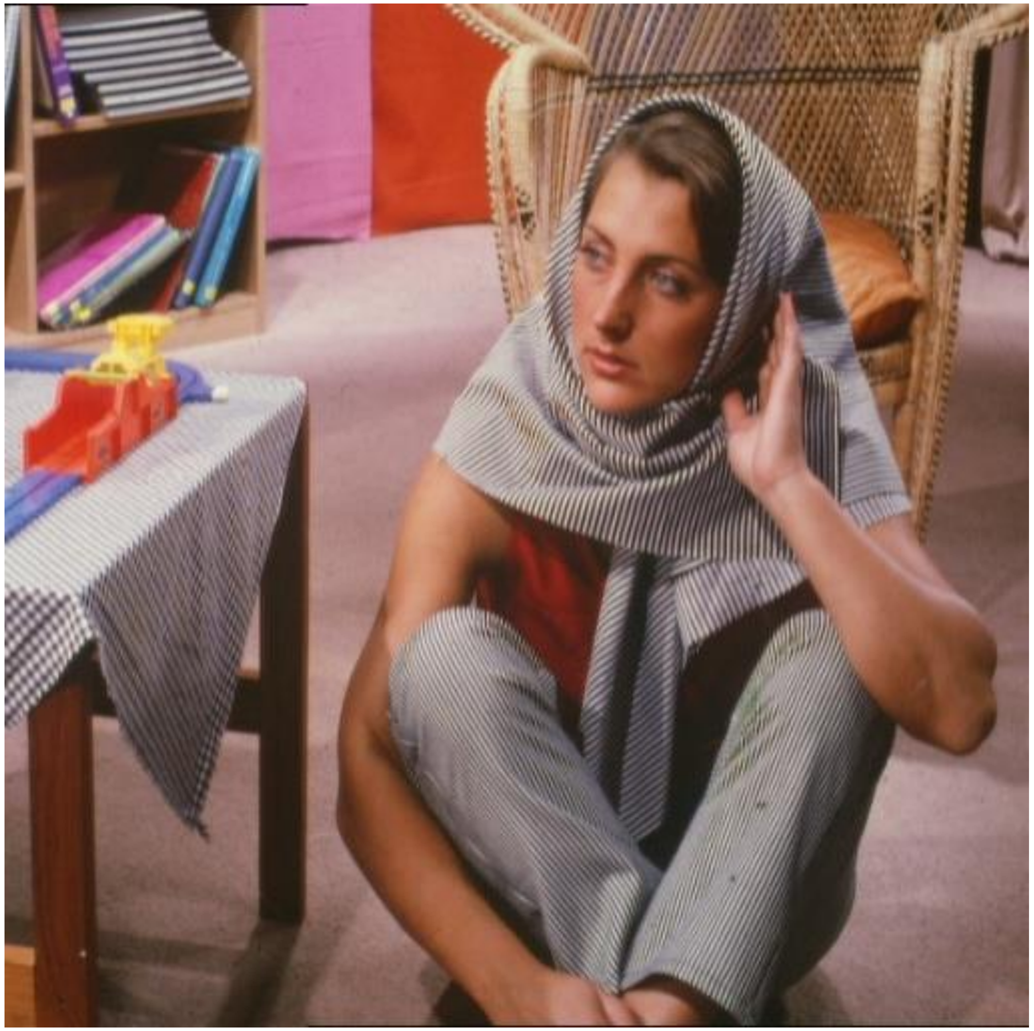
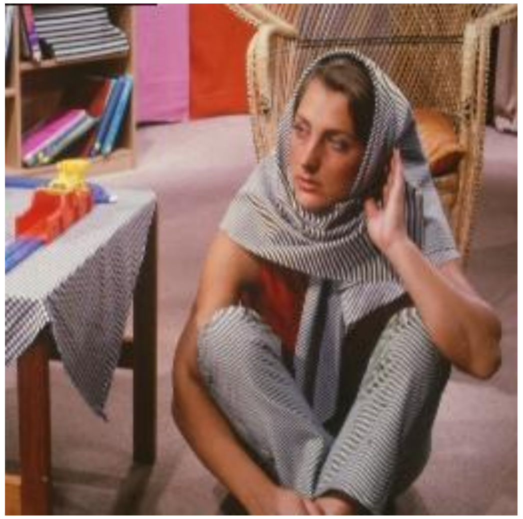
这是怎么做到的?

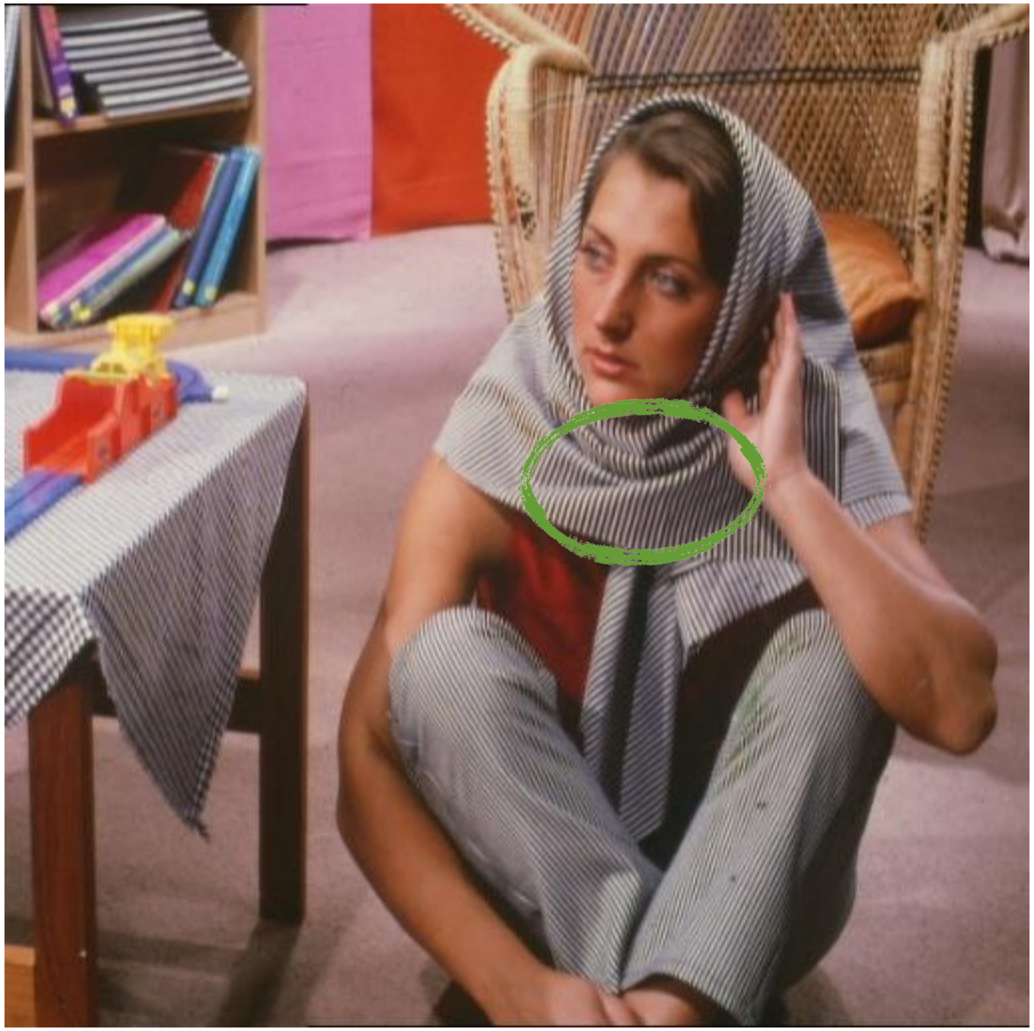
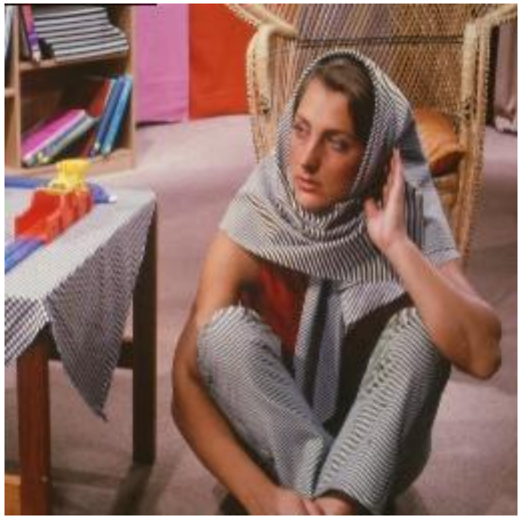

Copyright © 2007 Aude Oliva, MIT

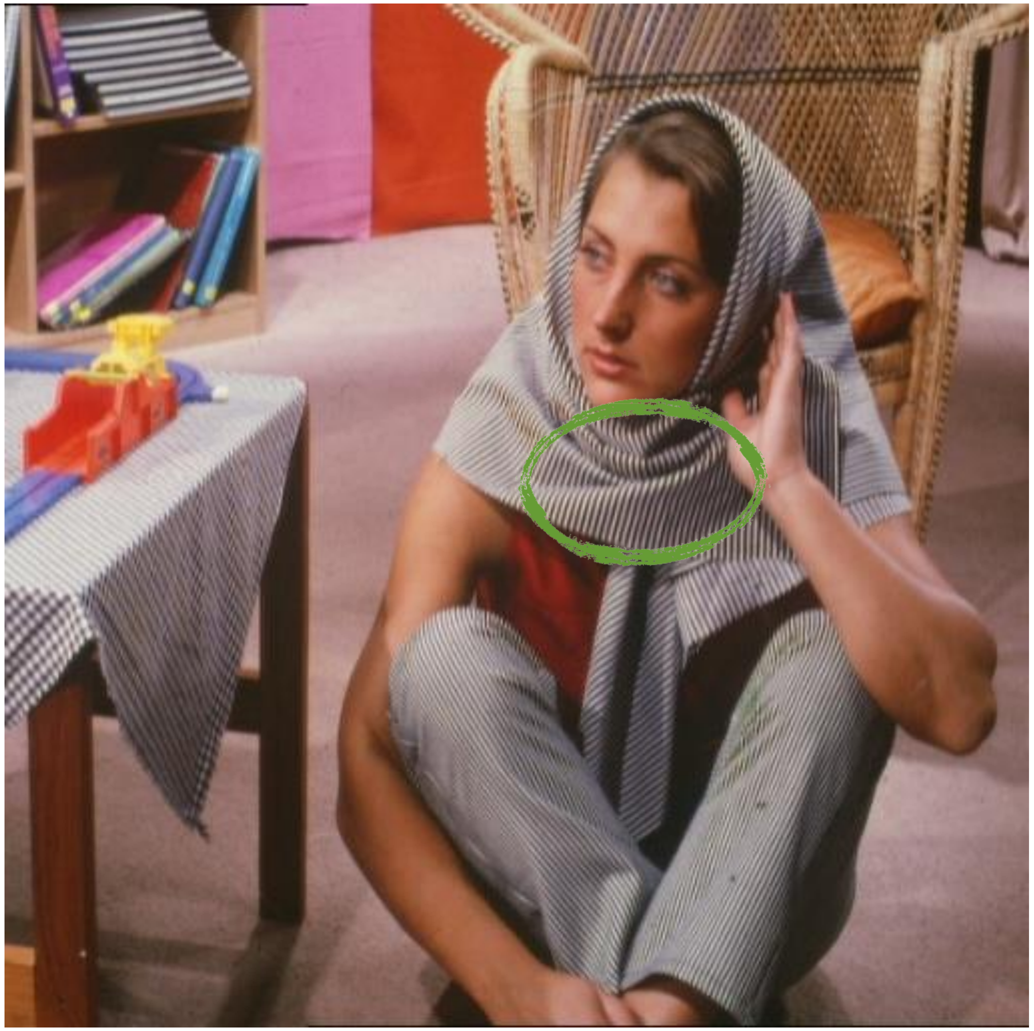
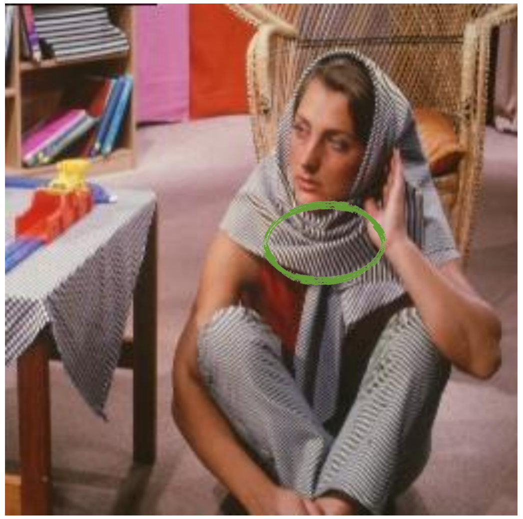


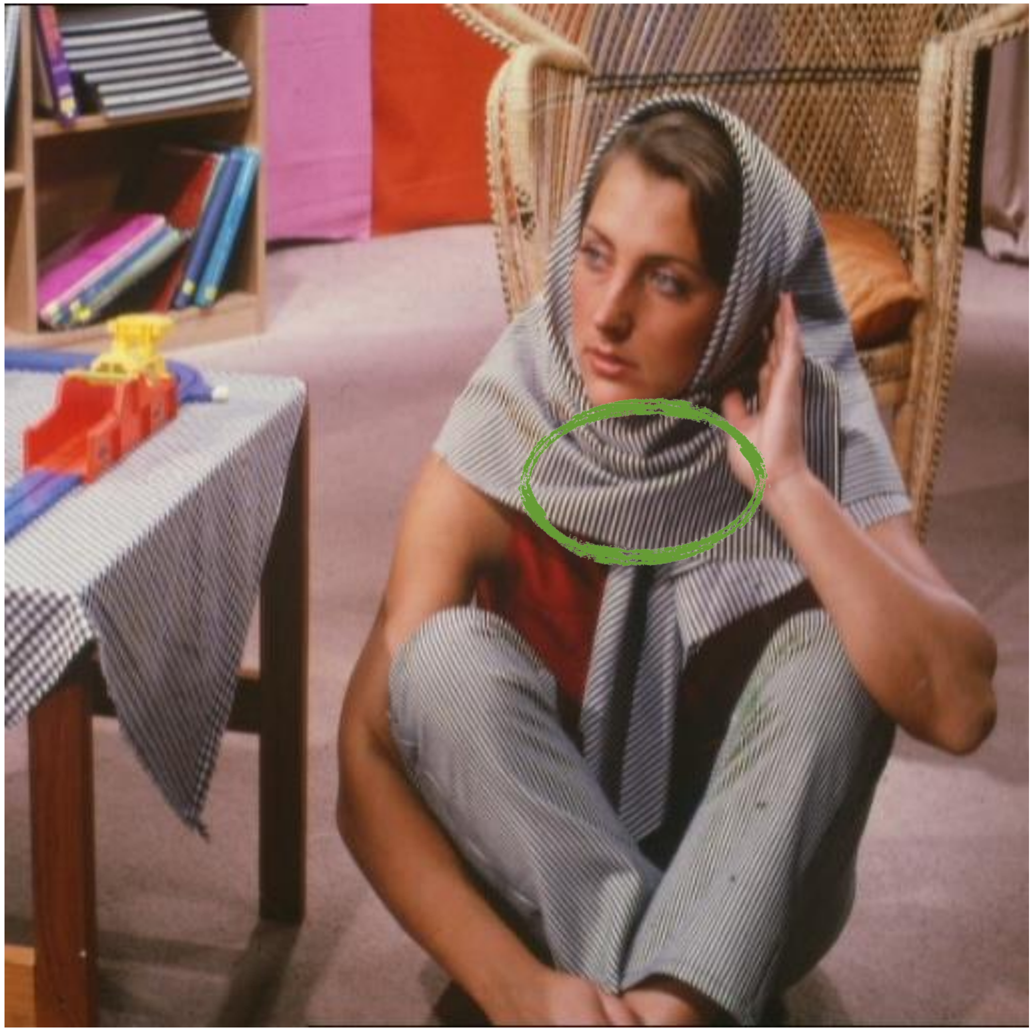
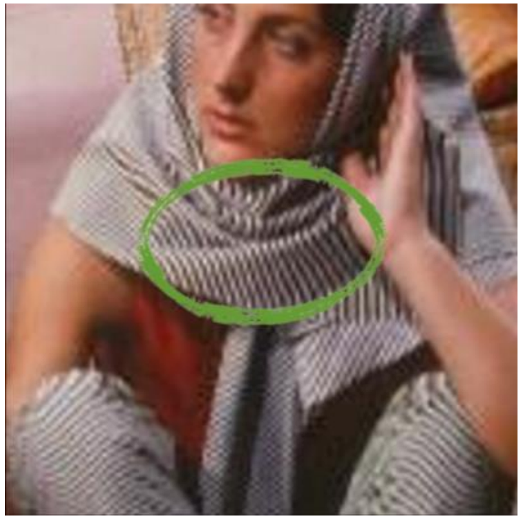
这个花纹是怎么来的?



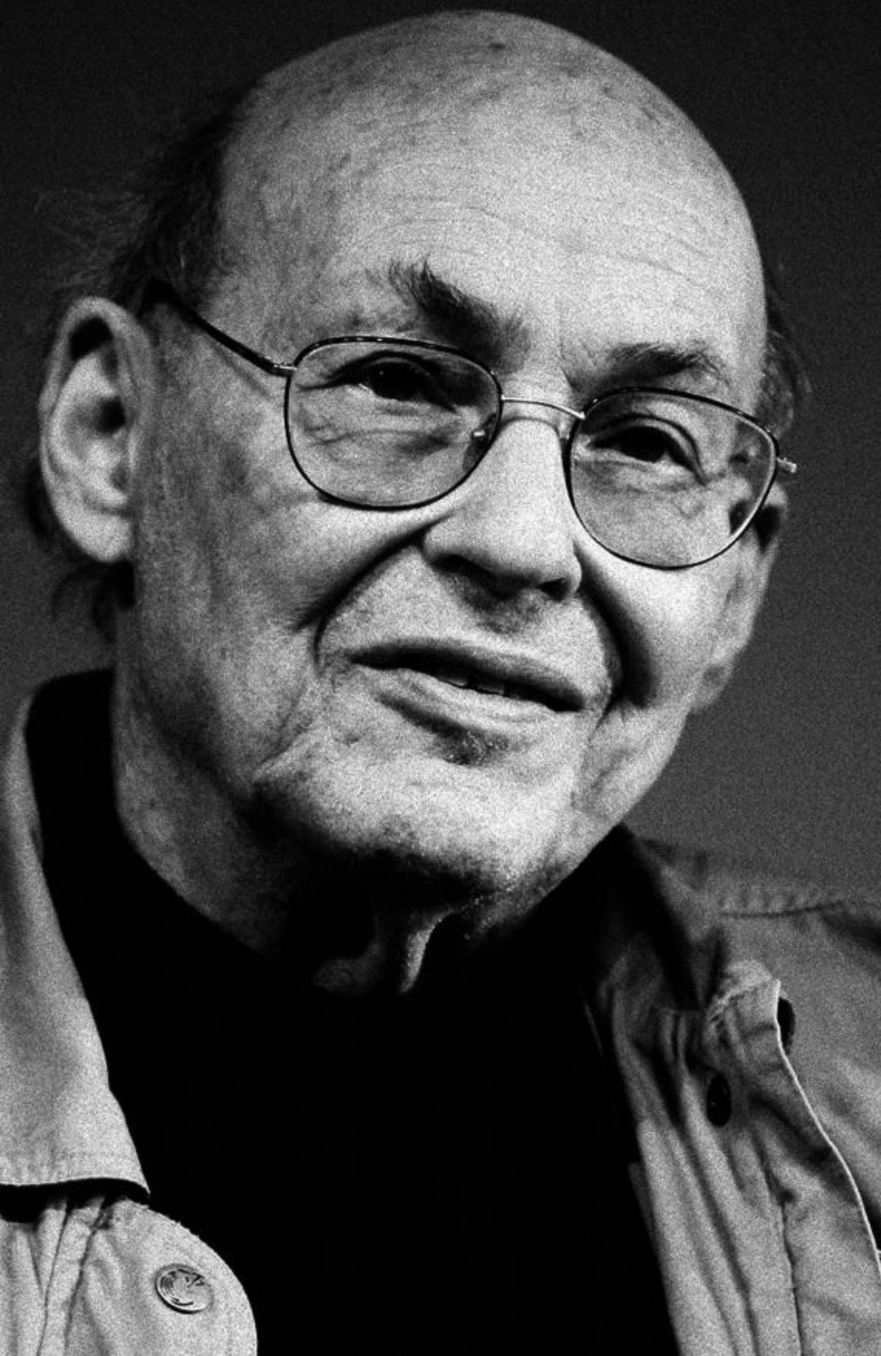



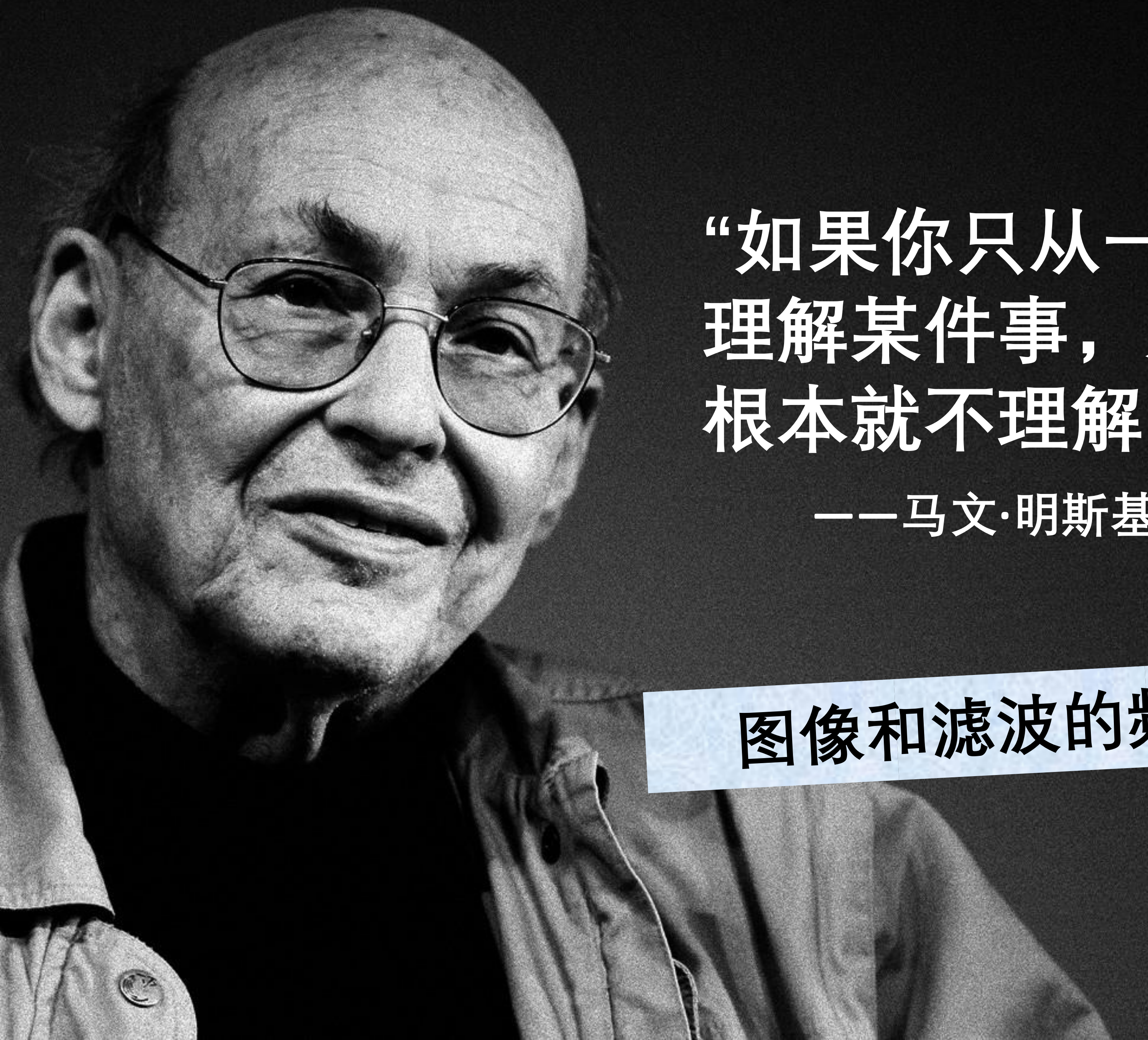



车轮是朝哪个方向转动的?



“如果你只从一个方面理解某件事，那么你根本就不理解它。”

——马文·明斯基 (1927-2016)



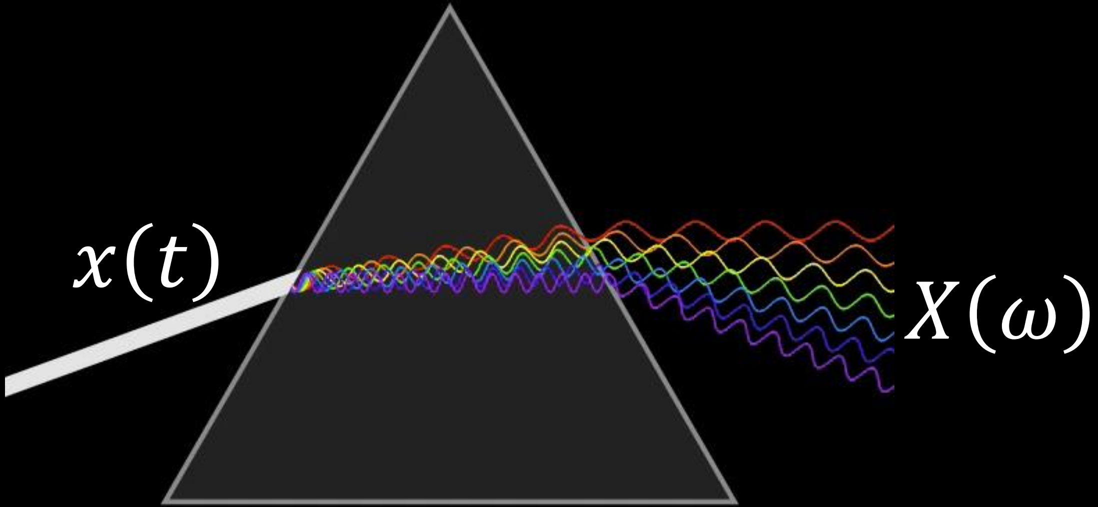
“如果你只从一个方面理解某件事，那么你根本就不理解它。”

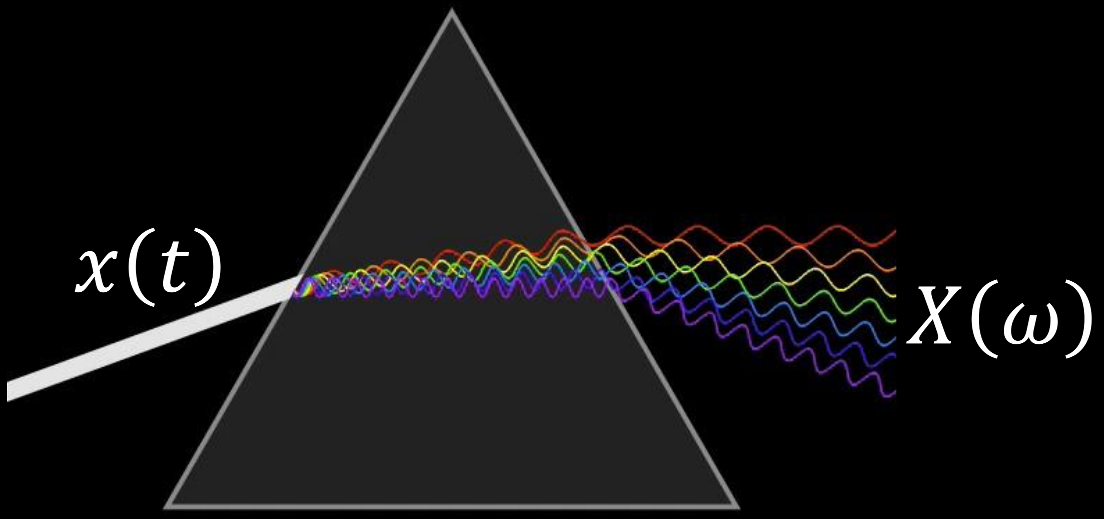
——马文·明斯基 (1927-2016)

图像和滤波的频域理解

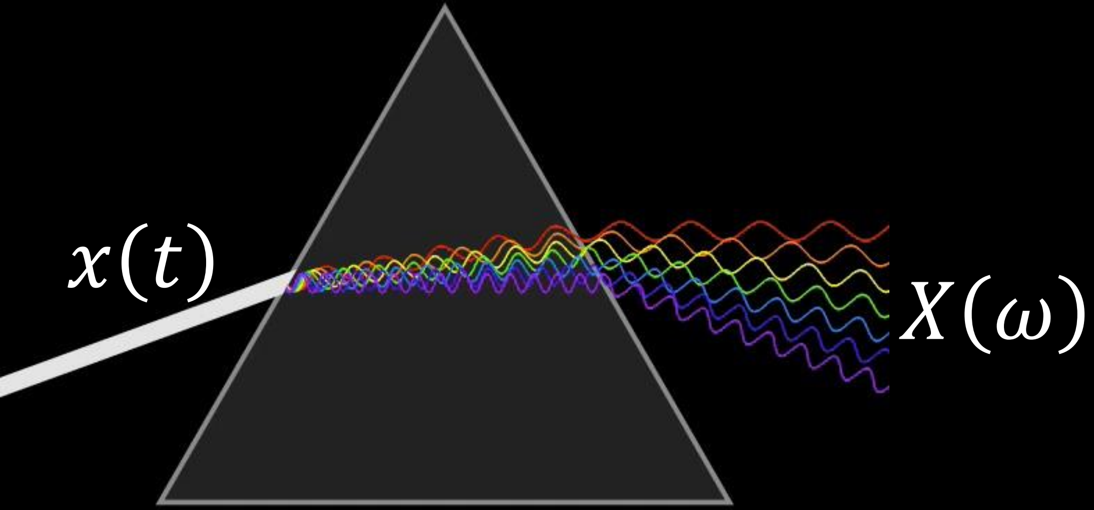
从**频率**的角度思考

$$x(t)$$

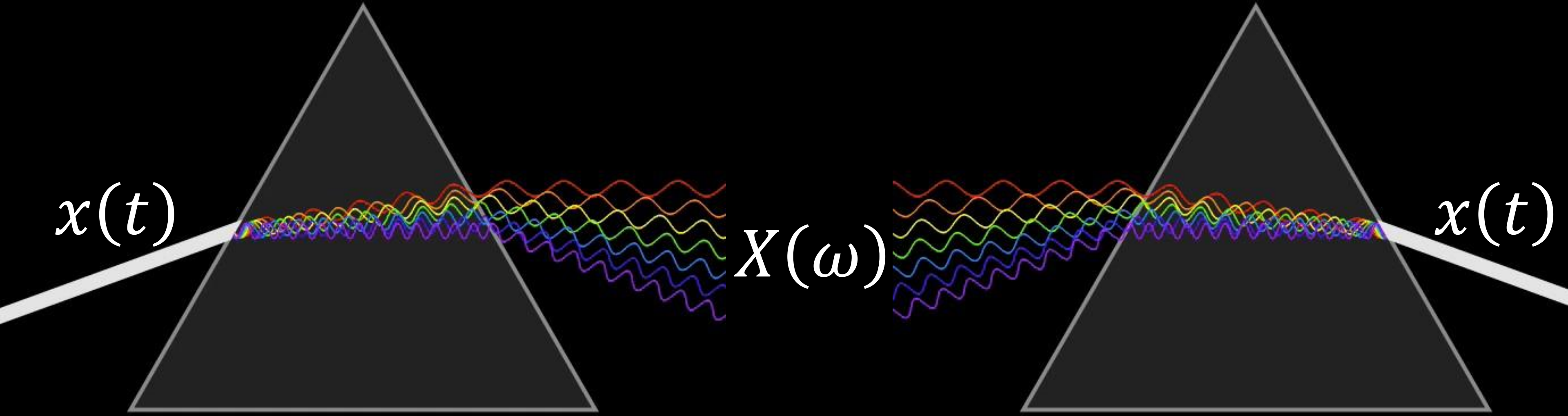

$x(t)$
$X(\omega)$



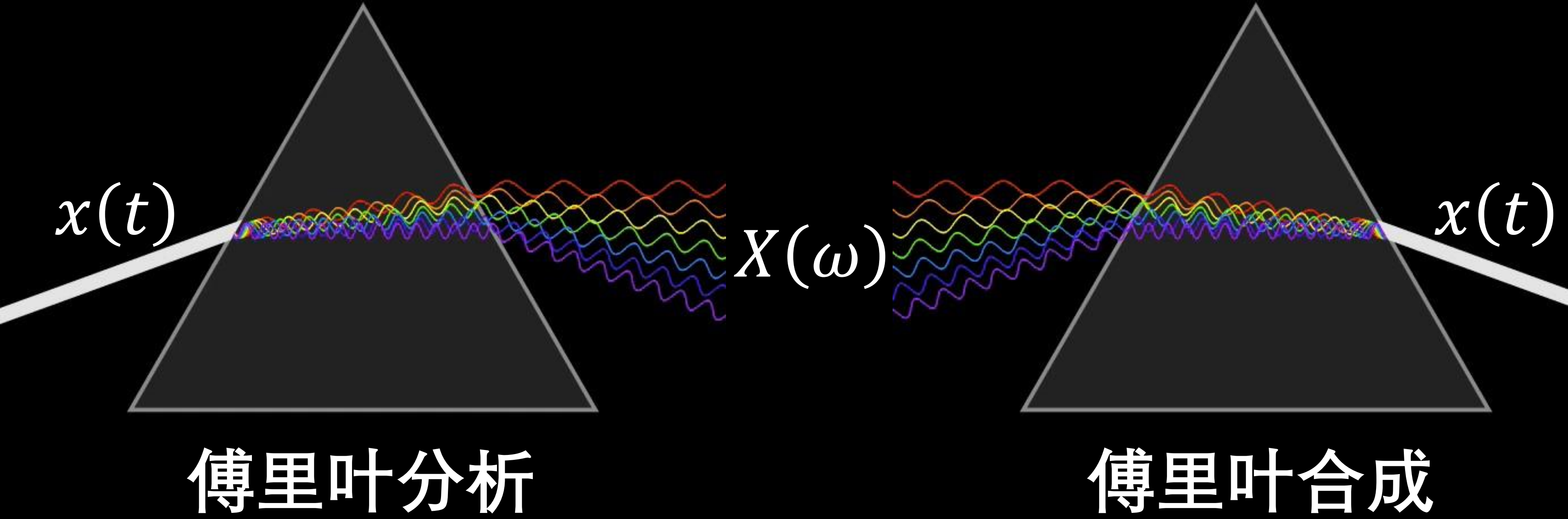
傅里叶分析



傅里叶分析



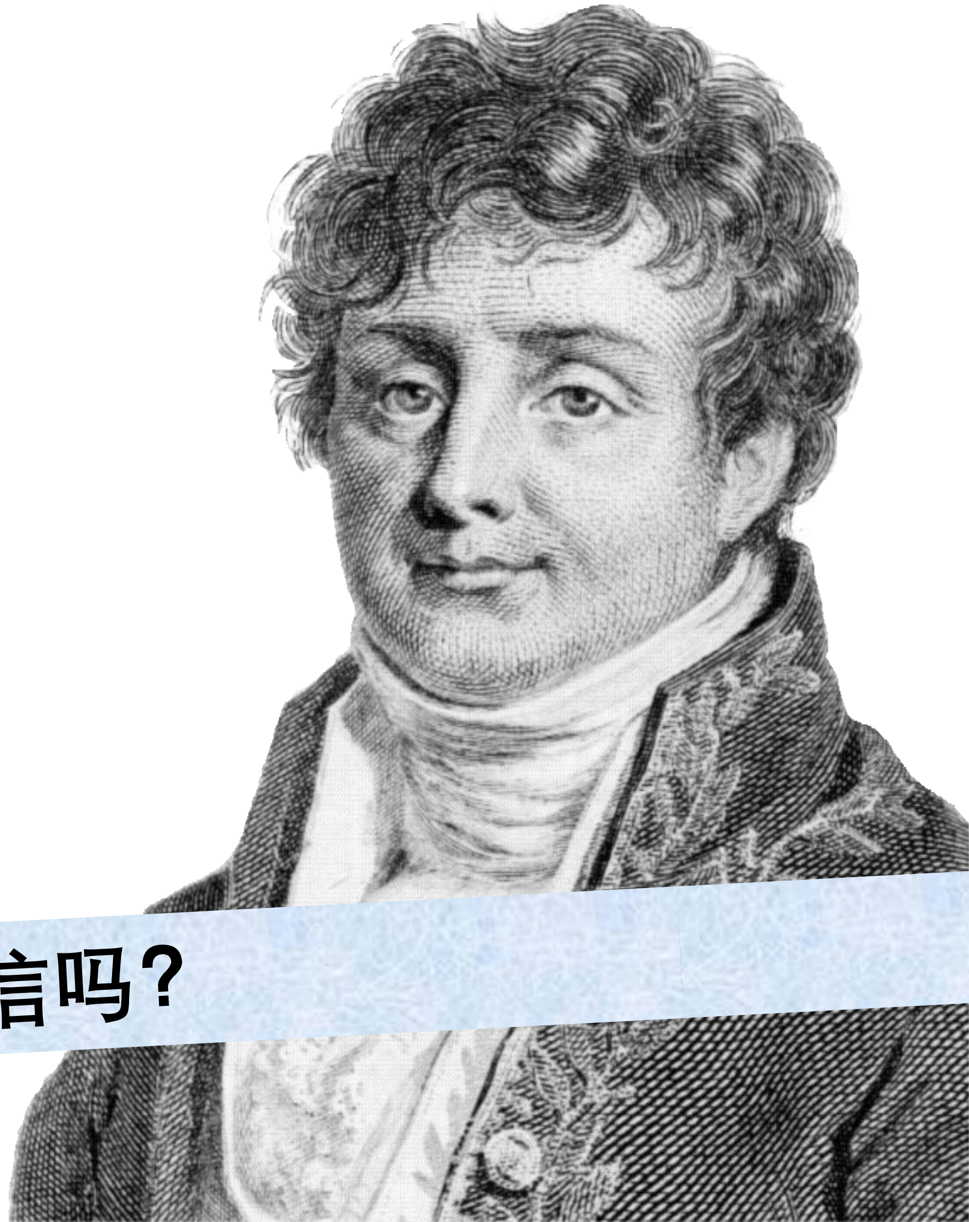
傅里叶分析


$x(t)$
$X(\omega)$
$x(t)$
傅里叶分析
傅里叶合成

让·巴普蒂斯·约瑟夫·傅里叶（1768-1830）

任何周期函数都可以
写成不同频率的正弦
波和余弦波的加权和。

任何周期函数都可以写成不同频率的正弦波和余弦波的加权和。



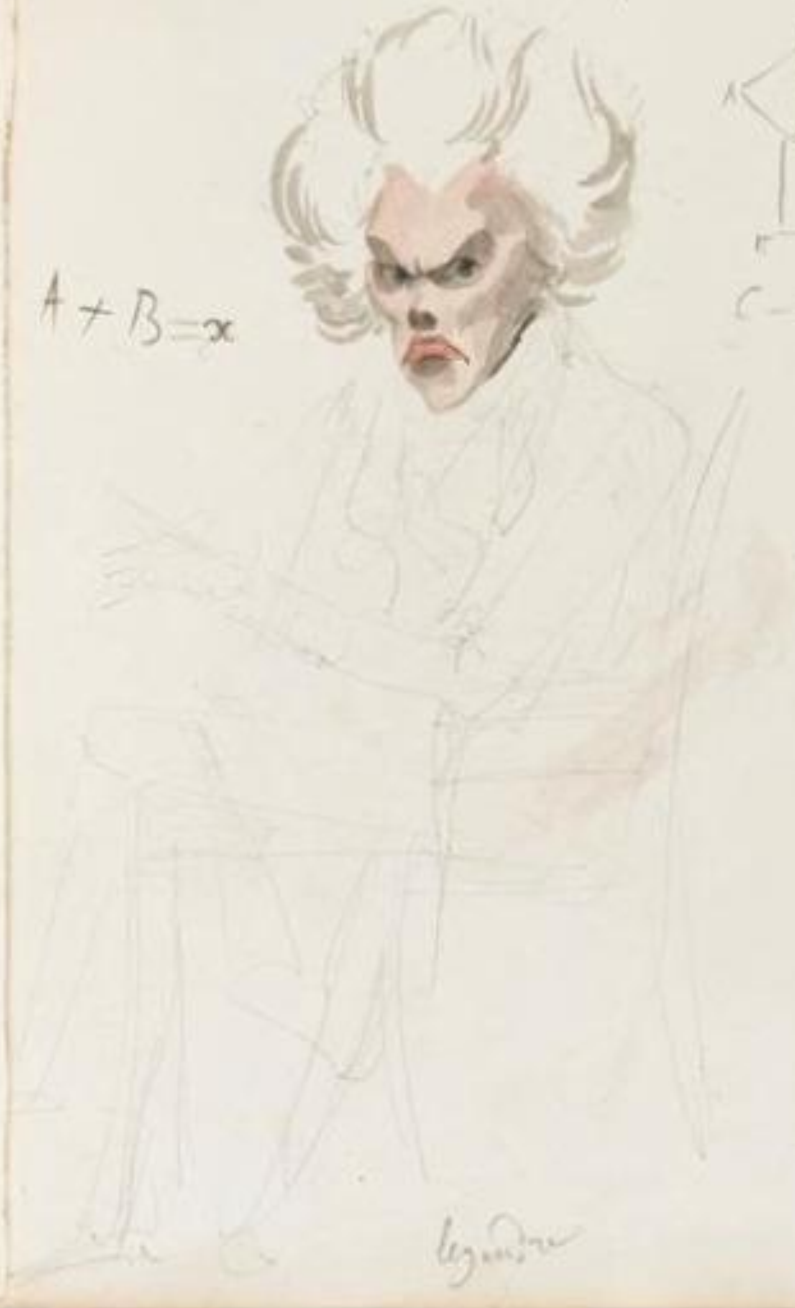
你相信吗？

皮埃尔-西蒙
拉普拉斯

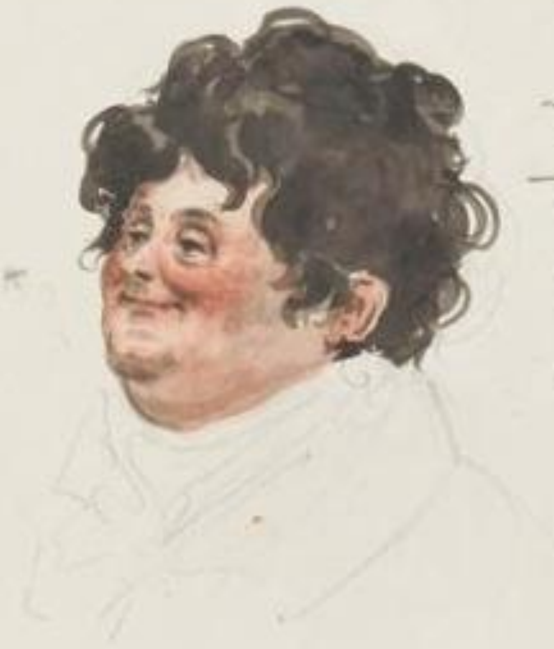
约瑟夫-路易斯
拉格朗日

阿德里安-马里
勒让德

$A + B = x$



$C - D = R + R$



[illegible]

$< [illegible] >$

$= x$

Legendre

Fourier

“...作者得出这些方程的方式并不是没有困难的，而且他对这些方程进行整合的分析仍然在一般性甚至严谨性方面留下了一些不足之处。”

——科学院奖委员会

约翰·彼得·古斯塔夫
勒热纳·狄利克雷

约翰·彼得·古斯塔夫
勒热纳·狄利克雷

一个信号存在傅里叶变换的充分不必要条件：

约翰·彼得·古斯塔夫
勒热纳·狄利克雷

一个信号存在傅里叶变换的充分不必要条件：

1. 在一周期内，连续或只有有限个第一类间断点

约翰·彼得·古斯塔夫
勒热纳·狄利克雷

一个信号存在傅里叶变换的充分不必要条件：

1. 在一周期内，连续或只有有限个第一类间断点

2. 在一周期内，极大值和极小值的数目应是有限个

一个信号存在傅里叶变换的充分不必要条件：

1. 在一周期内，连续或只有有限个第一类间断点

2. 在一周期内，极大值和极小值的数目应是有限个

3. 在一周期内，信号是绝对可积的

约翰·彼得·古斯塔夫
勒热纳·狄利克雷

约翰·彼得·古斯塔夫
勒热纳·狄利克雷

一个信号存在傅里叶变换的充分不必要条件：

1. 在一周期内，连续或只有有限个第一类间断点

2. 在一周期内，极大值和极小值的数目应是有限个

3. 在一周期内，信号是绝对可积的

科学素养文库·科学元典丛书

热的解析理论

Analytical Theory of Heat

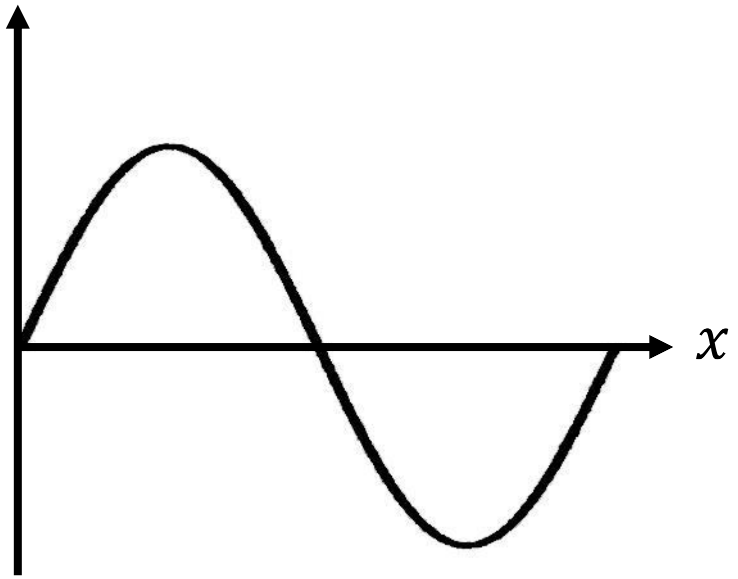
［法］傅立叶 著

科学元典是科学史和人类文明史上划时代的丰碑，是人类文化的优秀遗产，是历经时间考验的不朽之作。它们不仅是伟大的科学创造的结晶，而且是科学精神、科学思想和科学方法的载体，具有永恒的意义和价值。

北京大学出版社

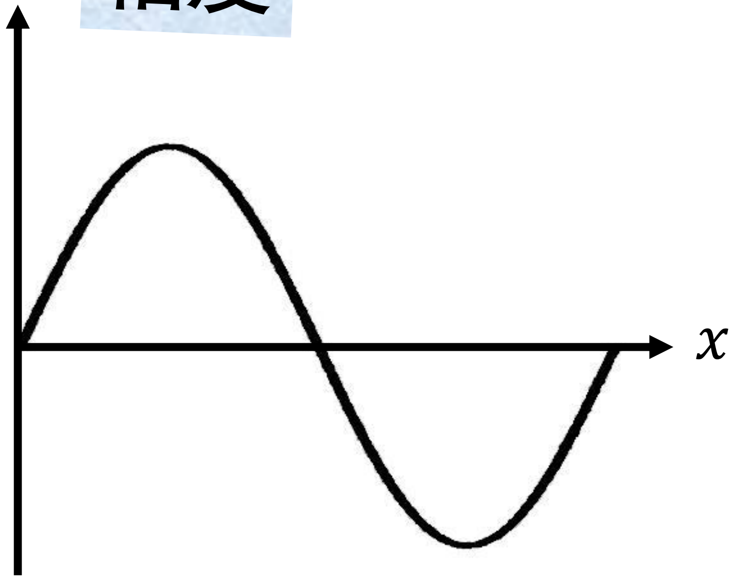
$$A \sin(\omega x + \phi)$$

$$A \sin(\omega x + \phi)$$



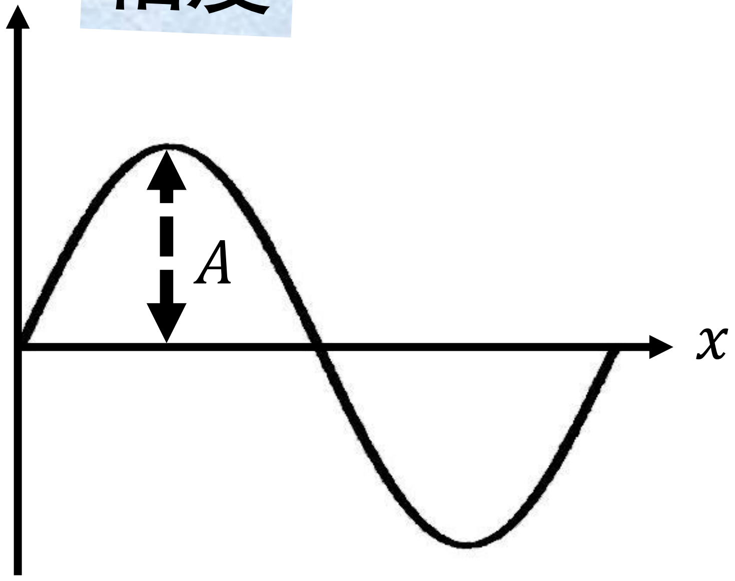
$$A\sin(\omega x + \phi)$$

幅度



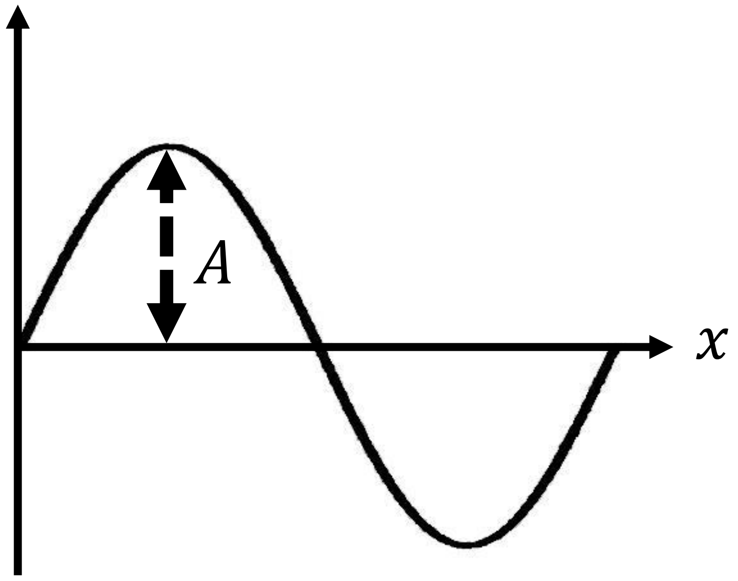
$$A\sin(\omega x + \phi)$$

幅度



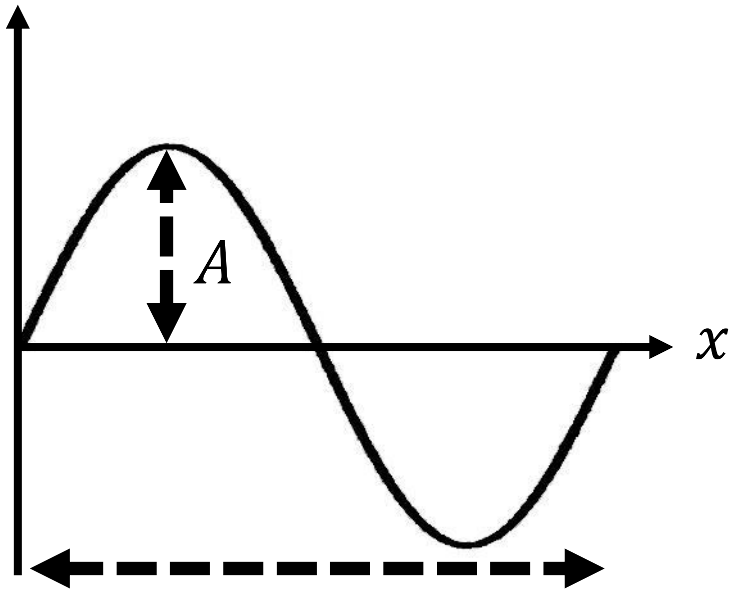
频率

$$A \sin(\omega x + \phi)$$



频率

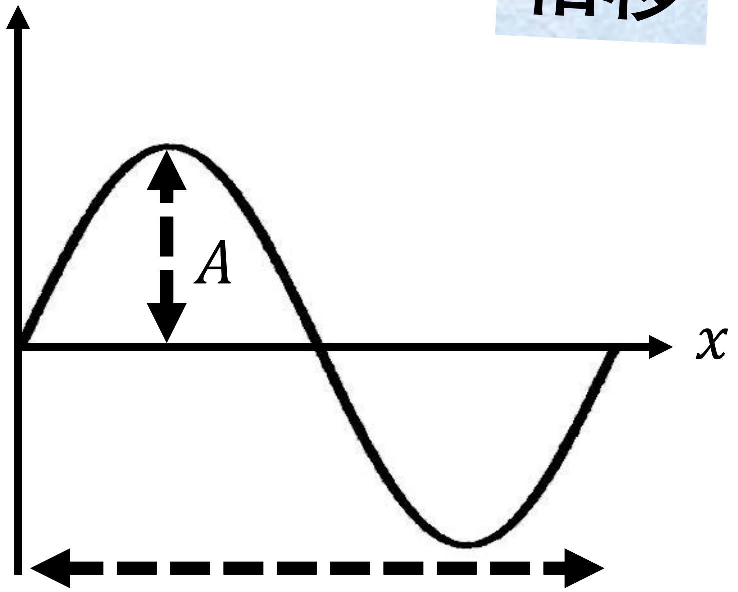
$$A\sin(\omega x+\phi)$$



$$T=\frac{2\pi}{\omega}$$

$$A\sin(\omega x + \bigcirc\!\!\!\!\phi\,)$$

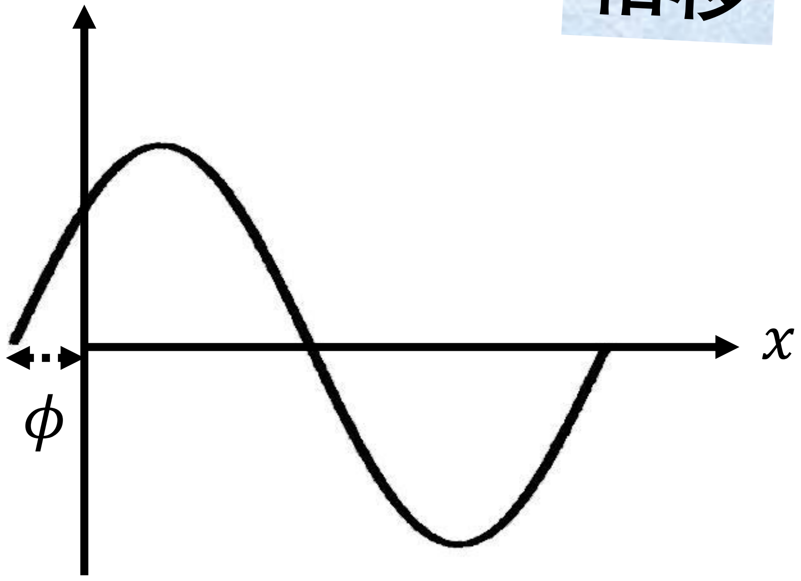
相移



$$T = \frac{2\pi}{\omega}$$

$$A\sin(\omega x + \phi)$$

相移



$$A \sin(\omega x + \phi)$$

$$\begin{aligned} &A\sin(\omega x + \phi) \\ &\qquad = A[\sin(\phi)\cos(\omega x) + \cos(\phi)\sin(\omega x)] \end{aligned}$$

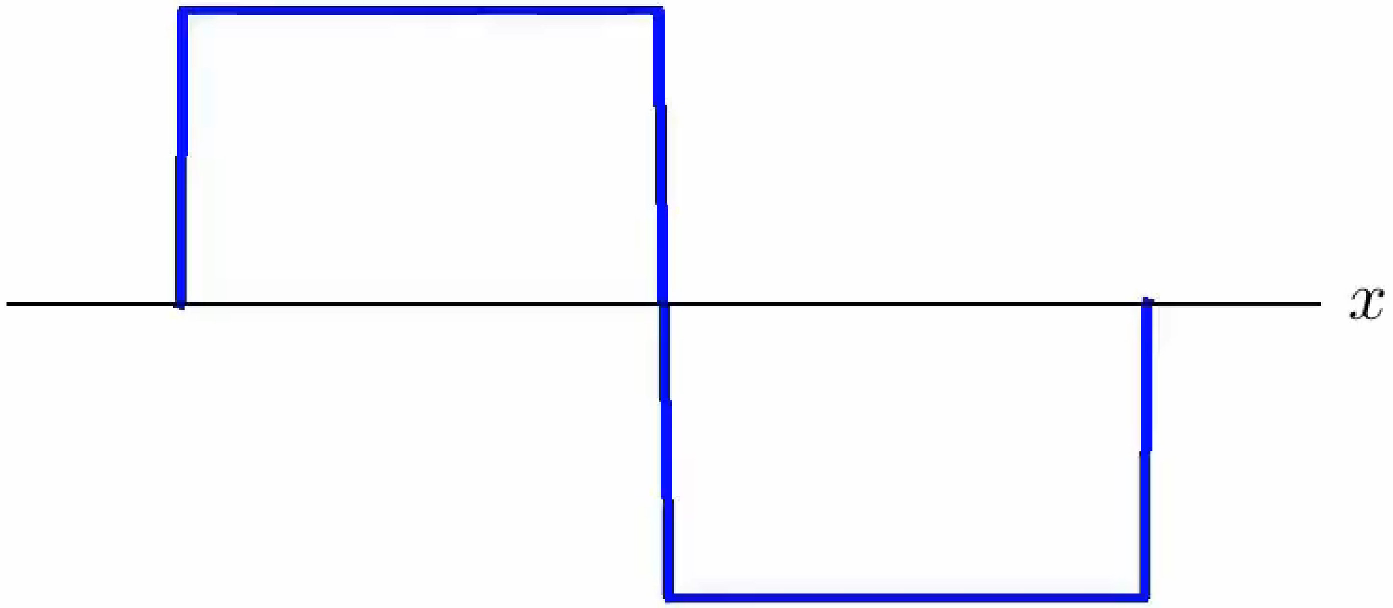
$$
\begin{aligned}
&A\sin(\omega x+\phi)\\
&\qquad = A[\sin(\phi)\cos(\omega x)+\cos(\phi)\sin(\omega x)]\\
&\qquad = A\sin(\phi)\cos(\omega x)+A\cos(\phi)\sin(\omega x)
\end{aligned}
$$

$$\begin{aligned} &A\sin(\omega x+\phi) \\ &\qquad = A[\sin(\phi)\cos(\omega x)+\cos(\phi)\sin(\omega x)] \\ &\qquad = \underbrace{A\sin(\phi)}_{\text{常数}}\cos(\omega x)+A\cos(\phi)\sin(\omega x) \end{aligned}$$

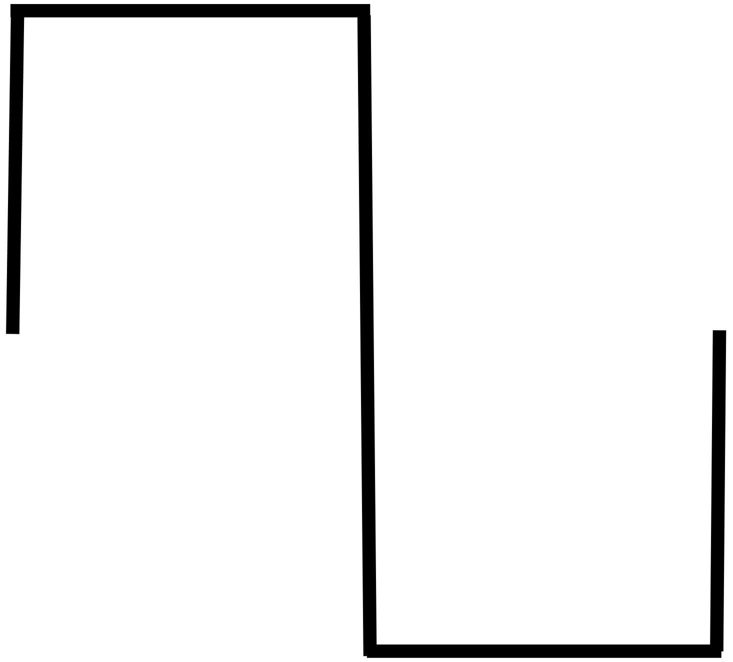
$$
\begin{aligned}
&A\sin(\omega x+\phi)\\
&\qquad = A[\sin(\phi)\cos(\omega x)+\cos(\phi)\sin(\omega x)]\\
&\qquad = \underbrace{A\sin(\phi)}_{\text{常数}}\cos(\omega x)+\underbrace{A\cos(\phi)}_{\text{常数}}\sin(\omega x)
\end{aligned}
$$

$$
\begin{aligned}
&A\sin(\omega x+\phi)\\
&\qquad = A[\sin(\phi)\cos(\omega x)+\cos(\phi)\sin(\omega x)]\\
&\qquad = A\sin(\phi)\cos(\omega x)+A\cos(\phi)\sin(\omega x)\\
&\qquad = \mathrm{\alpha}\cos(\omega x)+\mathrm{\beta}\sin(\omega x)
\end{aligned}
$$

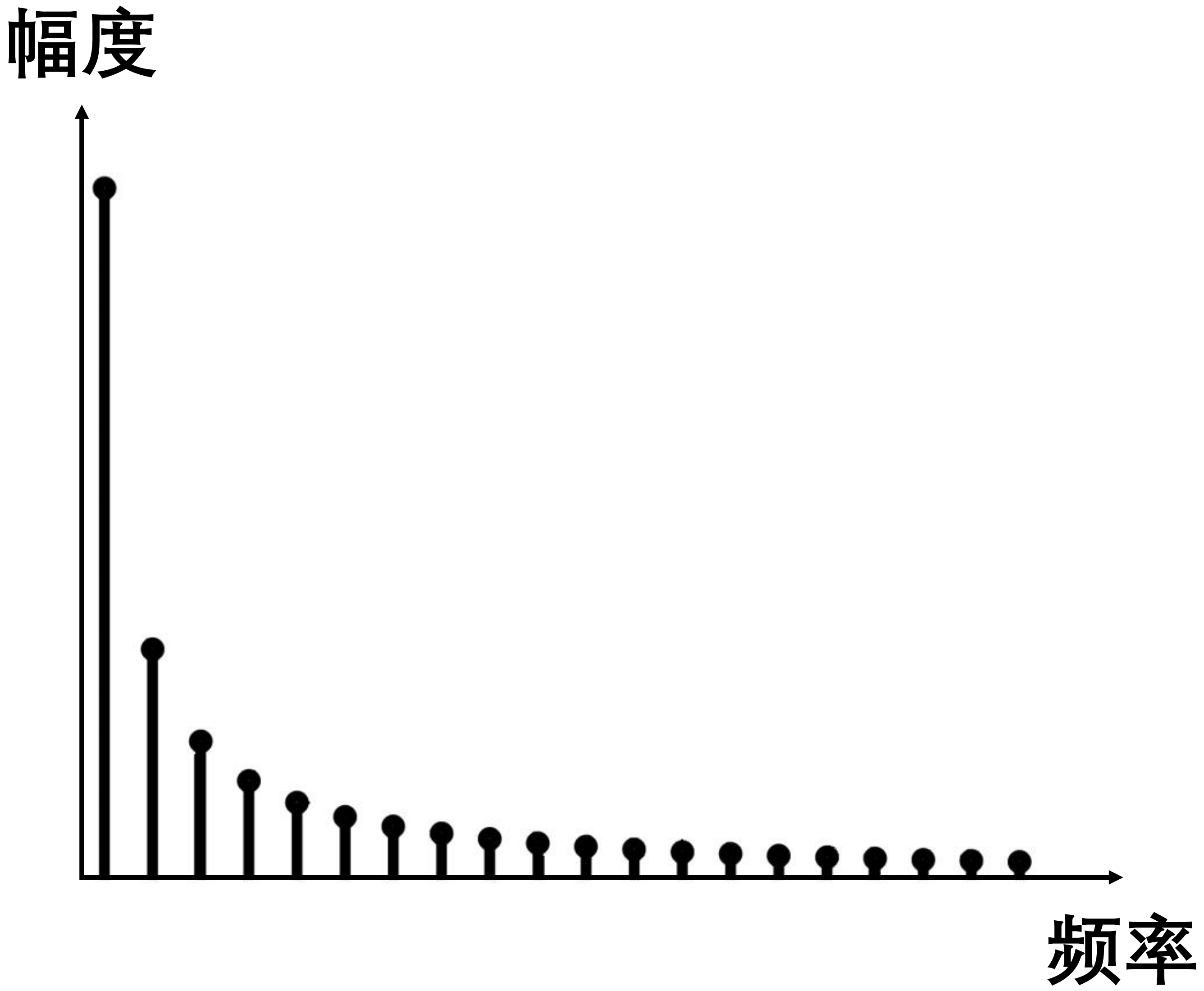
$$A\sin(\omega x + \phi) = \alpha\cos(\omega x) + \beta\sin(\omega x)$$



方波



$$= \frac{4}{\pi} \sum_{k=1}^{\infty} \frac{\sin(2\pi(2k-1)x)}{2k-1}$$


幅度
频率

傅里叶变换

$$f(x) \longrightarrow \boxed{\text{傅里叶变换}}$$

$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

对于每一个ω，$F(\omega)$保留了对应的正弦函数的幅度，A，和相位，ϕ

$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

对于每一个ω，$F(\omega)$保留了对应的正弦函数的幅度，A，和相位，ϕ

$F(\omega)$如何同时保留幅度和相位?

$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

对于每一个ω，$F(\omega)$保留了对应的正弦函数的幅度，A，和相位，ϕ

$F(\omega)$如何同时保留幅度和相位?

复数

$F(\omega)$如何同时保留幅度和相位?

复数

$$\alpha + i\beta$$

其中

$$i = \sqrt{-1}$$

$F(\omega)$如何同时保留幅度和相位?

复数

$$\alpha + i\beta = A(\cos\theta + i\sin\theta)$$

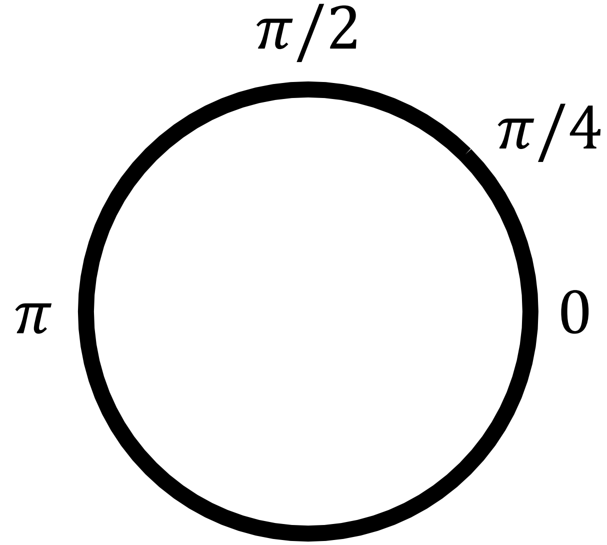
其中

$$i = \sqrt{-1}$$

$$\alpha + i\beta = A(\cos\theta + i\sin\theta)$$

其中

$$i = \sqrt{-1}$$

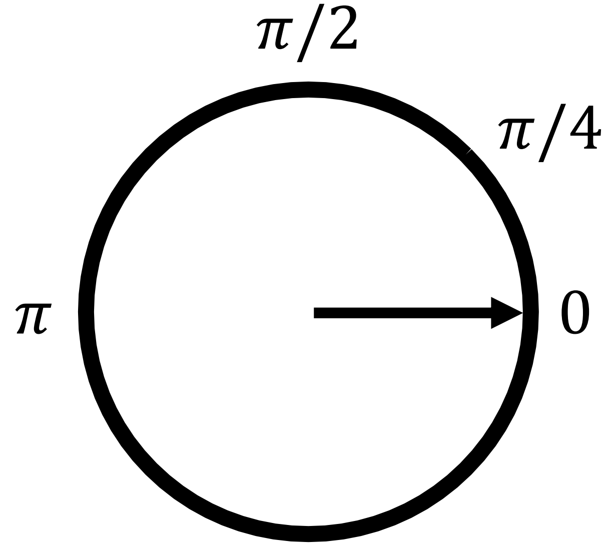


复平面

$$\alpha + i\beta = A(\cos\theta + i\sin\theta)$$

其中

$$i = \sqrt{-1}$$

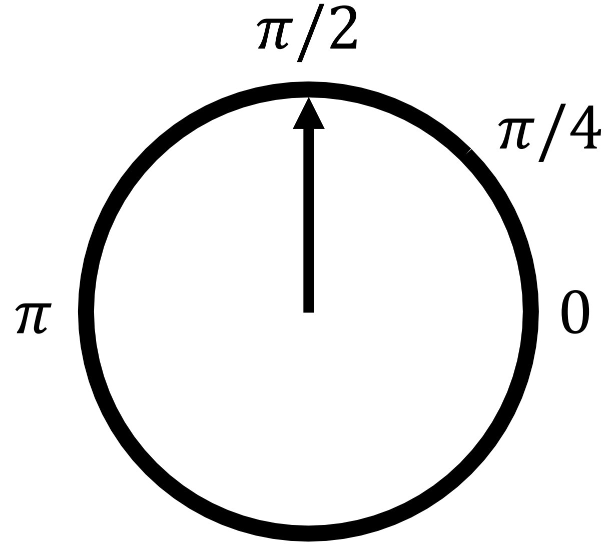


复平面

$$\alpha + i\beta = A(\cos\theta + i\sin\theta)$$

其中

$$i = \sqrt{-1}$$

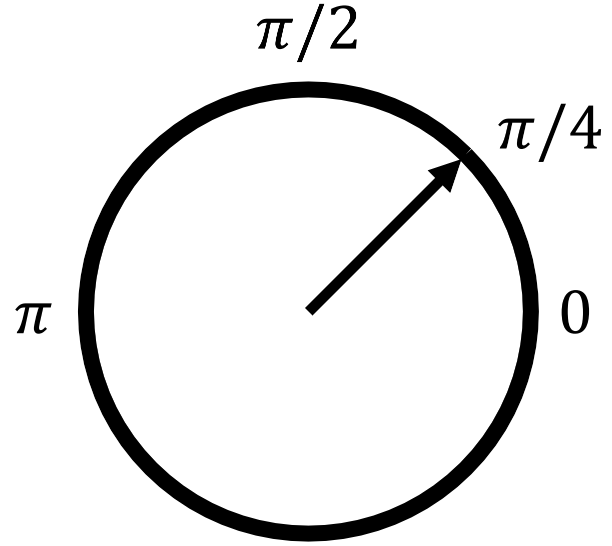


复平面

$$\alpha + i\beta = A(\cos\theta + i\sin\theta)$$

其中

$$i = \sqrt{-1}$$

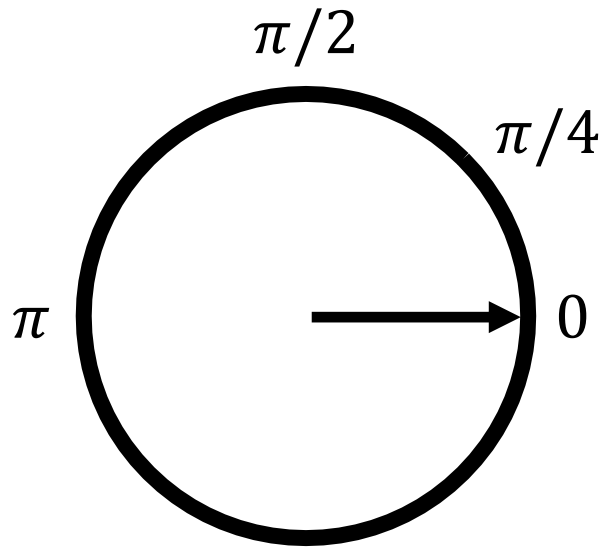


复平面

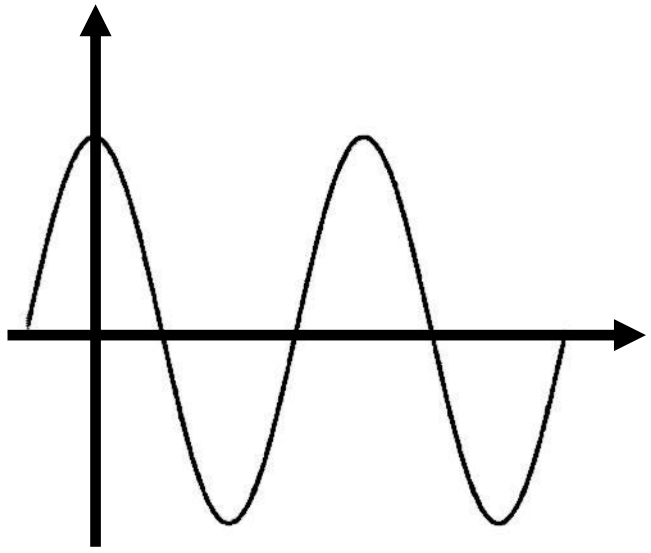
$$\alpha + i\beta = A(\cos\theta + i\sin\theta)$$

其中

$$i = \sqrt{-1}$$



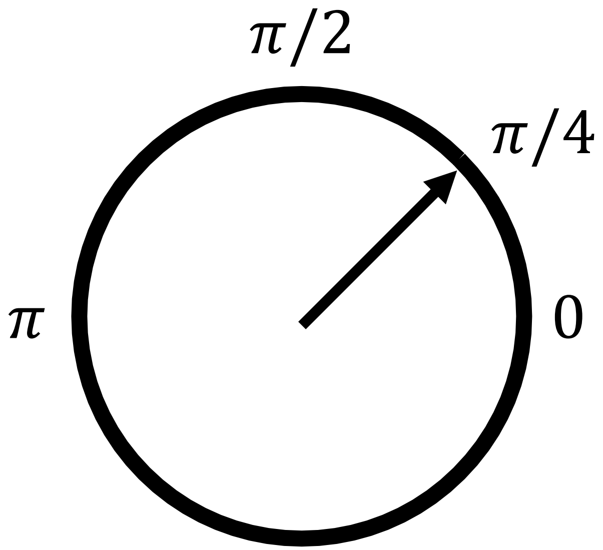
复平面



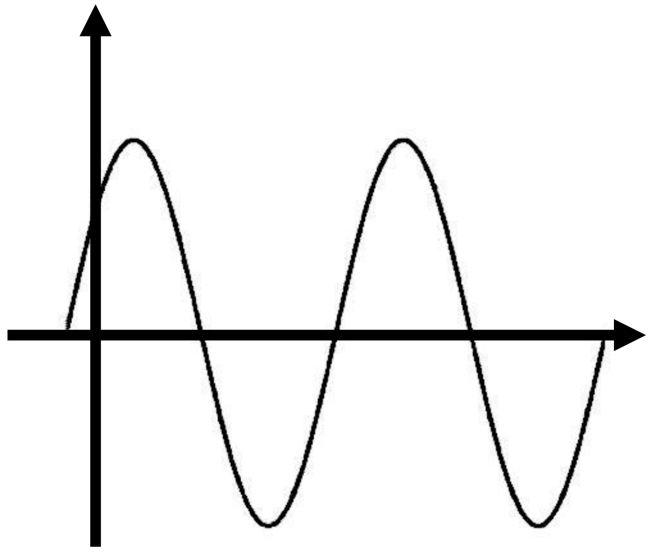
$$\alpha + i\beta = A(\cos\theta + i\sin\theta)$$

其中

$$i = \sqrt{-1}$$



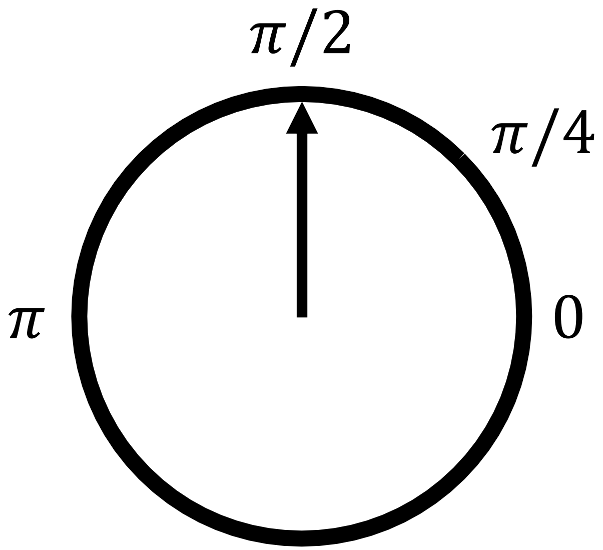
复平面



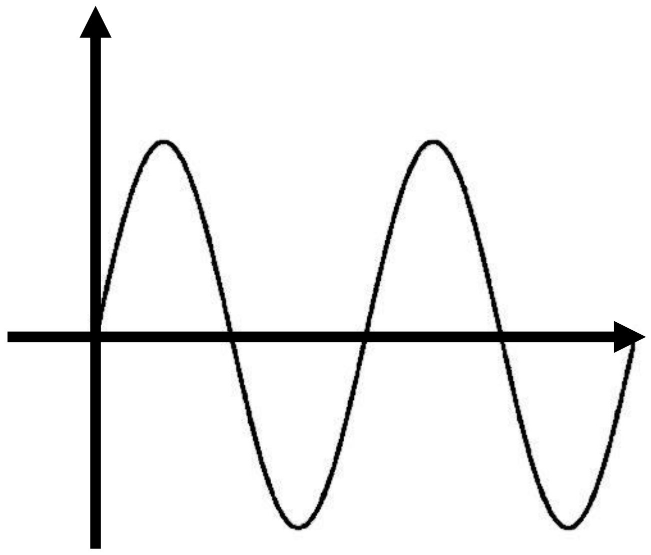
$$\alpha + i\beta = A(\cos\theta + i\sin\theta)$$

其中

$$i = \sqrt{-1}$$



复平面



$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

$$F(\omega) = \mathrm{Real}(\omega) + i \cdot \mathrm{Imaginary}(\omega)$$

$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

$$F(\omega) = \text{Real}(\omega) + i \cdot \text{Imaginary}(\omega)$$

$$M(\omega) = \|F(\omega)\| = \sqrt{\text{Real}(\omega)^2 + \text{Imaginary}(\omega)^2}$$

$$f(x) \longrightarrow \boxed{\text{傅里叶变换}} \longrightarrow F(\omega)$$

$$F(\omega) = \text{Real}(\omega) + i \cdot \text{Imaginary}(\omega)$$

$$M(\omega) = \|F(\omega)\| = \sqrt{\text{Real}(\omega)^2 + \text{Imaginary}(\omega)^2}$$

幅度

$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

$$F(\omega) = \text{Real}(\omega) + i \cdot \text{Imaginary}(\omega)$$

$$M(\omega) = \|F(\omega)\| = \sqrt{\text{Real}(\omega)^2 + \text{Imaginary}(\omega)^2}$$

$$\phi(\omega) = \tan^{-1}\left(\frac{\text{Imaginary}(\omega)}{\text{Real}(\omega)}\right)$$

$$\begin{tikzcd} f(x) \arrow[r] & \boxed{\text{傅里叶变换}} \arrow[r] & F(\omega) \end{tikzcd}$$

$$F(\omega) = \text{Real}(\omega) + i \cdot \text{Imaginary}(\omega)$$

$$M(\omega) = \|F(\omega)\| = \sqrt{\text{Real}(\omega)^2 + \text{Imaginary}(\omega)^2}$$

$$\phi(\omega) = \tan^{-1}\left(\frac{\text{Imaginary}(\omega)}{\text{Real}(\omega)}\right)$$

相位

欧拉公式

$$Ae^{ik} = A(\cos k + i \sin k)$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

其中

$$e^{-i2\pi\omega x} = \cos(-2\pi\omega x) + i\sin(-2\pi\omega x)$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

其中

$$e^{-i2\pi\omega x} = \cos(-2\pi\omega x) + i\sin(-2\pi\omega x)$$

测量函数中每个频率分量的大小

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

其中

$$e^{-i2\pi\omega x} = \cos(-2\pi\omega x) + i\sin(-2\pi\omega x)$$

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

其中

$$e^{-i2\pi\omega x} = \cos(-2\pi\omega x) + i\sin(-2\pi\omega x)$$

$$F(\omega) = \underline{\int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx} + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

其中

$$e^{-i2\pi\omega x} = \cos(-2\pi\omega x) + i \sin(-2\pi\omega x)$$

$$F(\omega) = \underline{\int_{-\infty}^{\infty} f(x) \cos(-2\pi\omega x)\, dx} + i \underline{\int_{-\infty}^{\infty} f(x) \sin(-2\pi\omega x)\, dx}$$

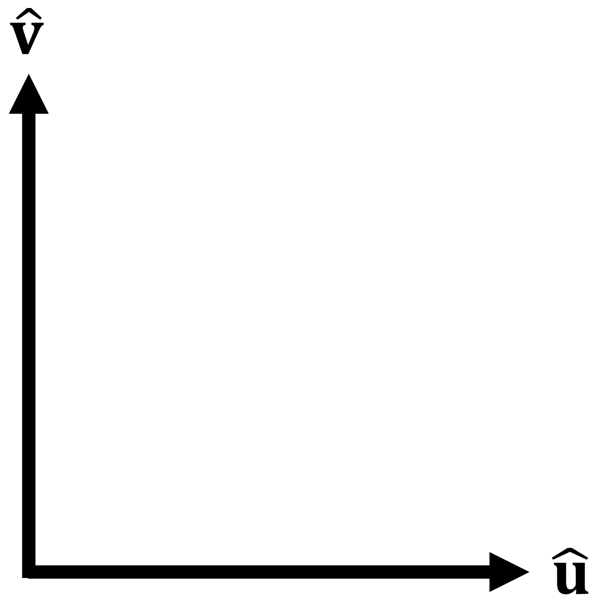
回顾： 线性代数

定义：

基 (basis) 是一组线性无关的向量，通过线性组合可以表示给定向量空间中的每一个向量

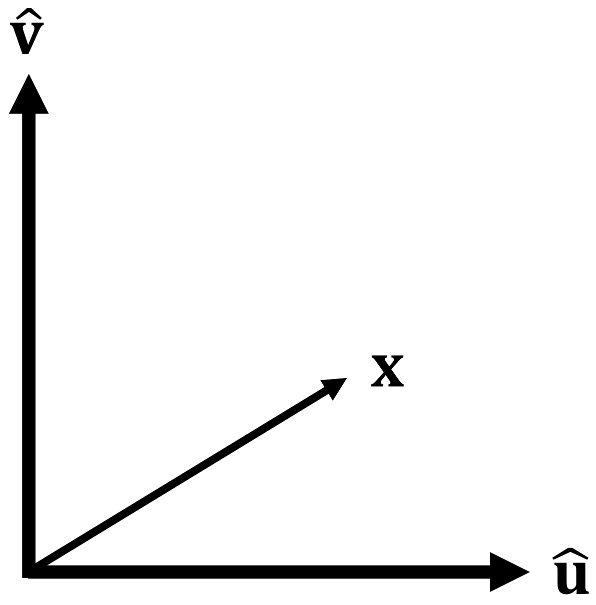
定义：

基 (basis) 是一组线性无关的向量，通过线性组合可以表示给定向量空间中的每一个向量



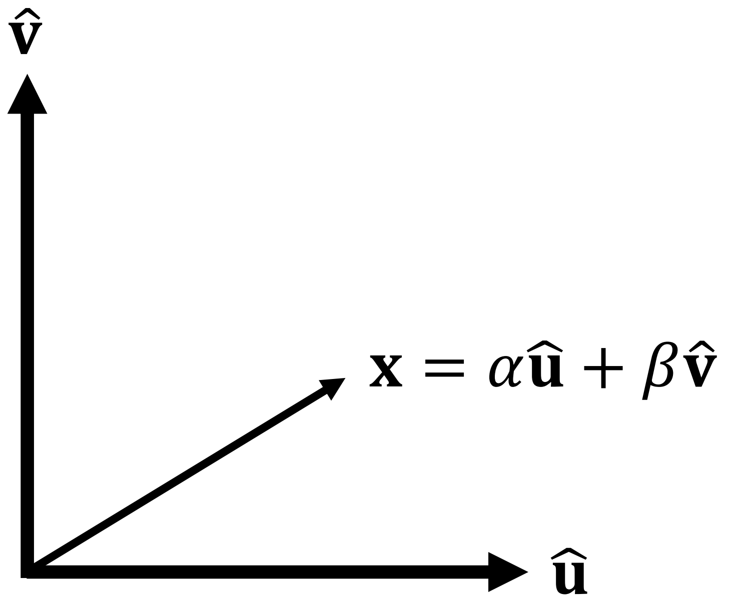
定义：

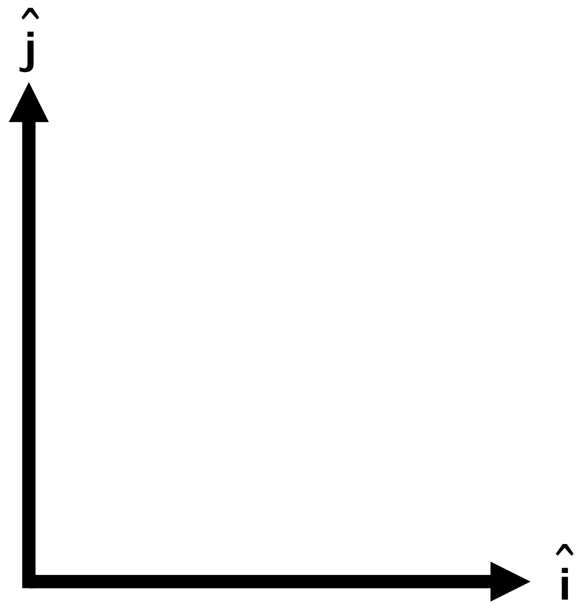
　　基 (basis) 是一组线性无关的向量，通过线性组合可以表示给定向量空间中的每一个向量

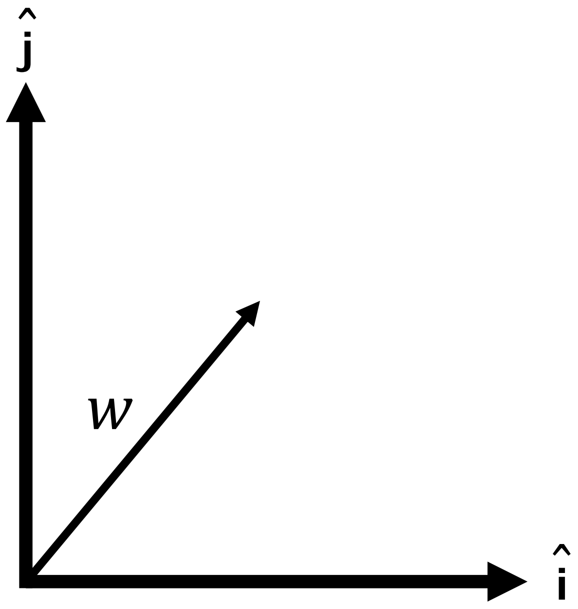


定义：

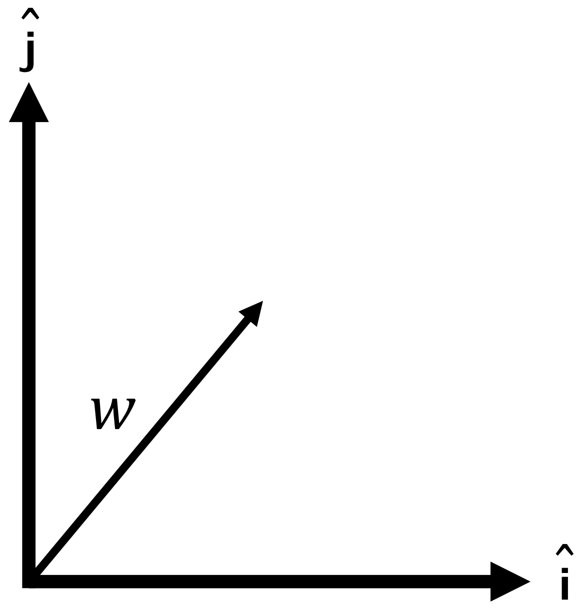
基 (basis) 是一组线性无关的向量，通过线性组合可以表示给定向量空间中的每一个向量


$\hat{\mathbf{v}}$
$\mathbf{x} = \alpha\hat{\mathbf{u}} + \beta\hat{\mathbf{v}}$
$\hat{\mathbf{u}}$

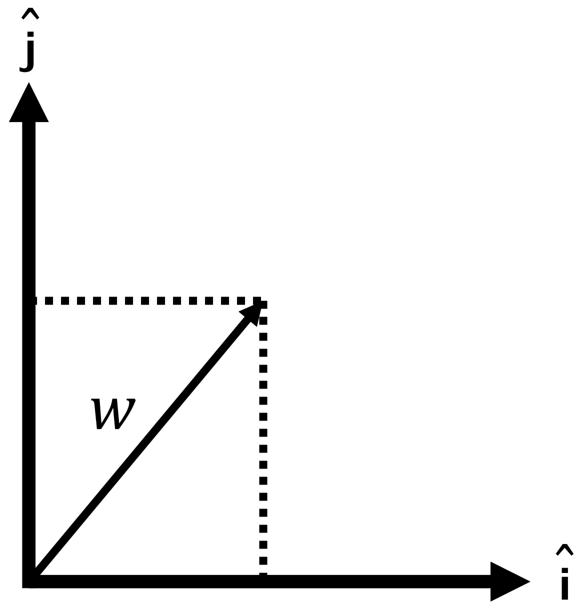

ĵ
î



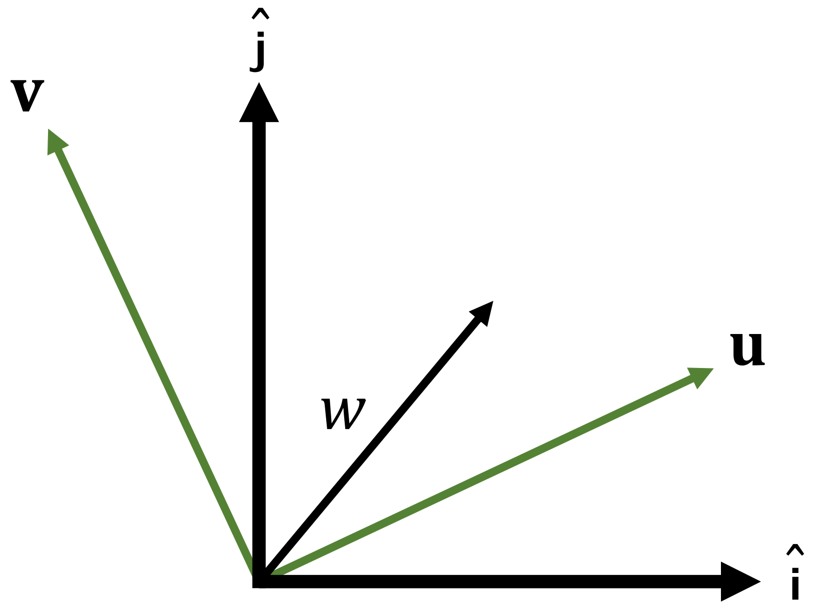
$\hat{j}$
w
$\hat{i}$



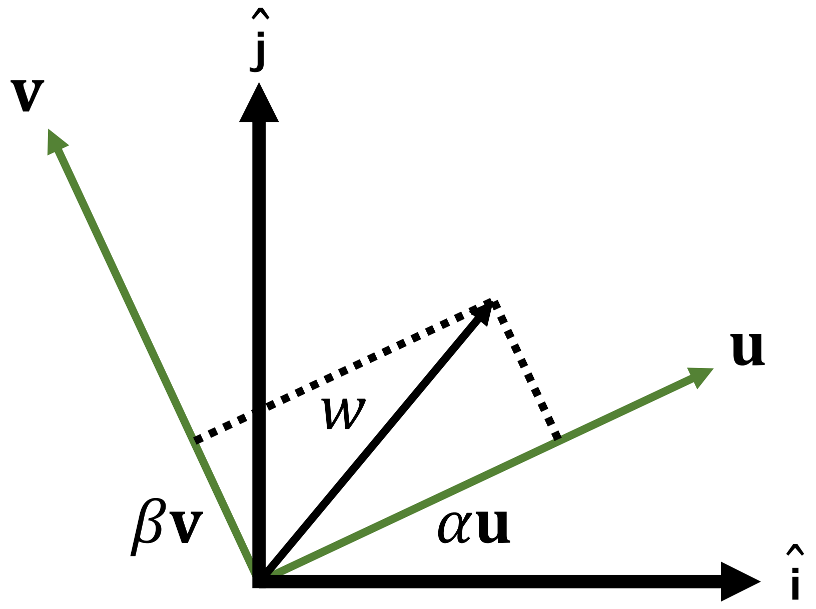
向量 (2D) 可以表示为两个向量的和



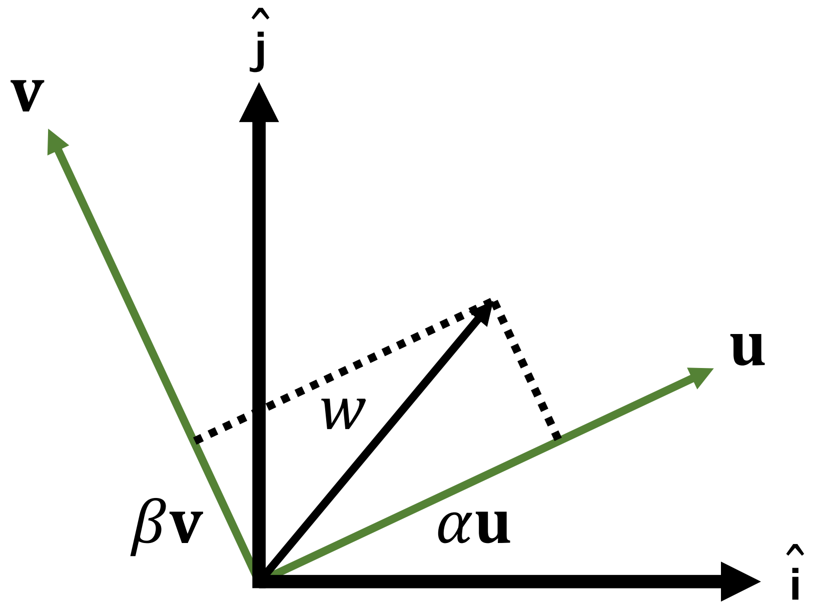
向量 (2D) 可以表示为两个向量的和



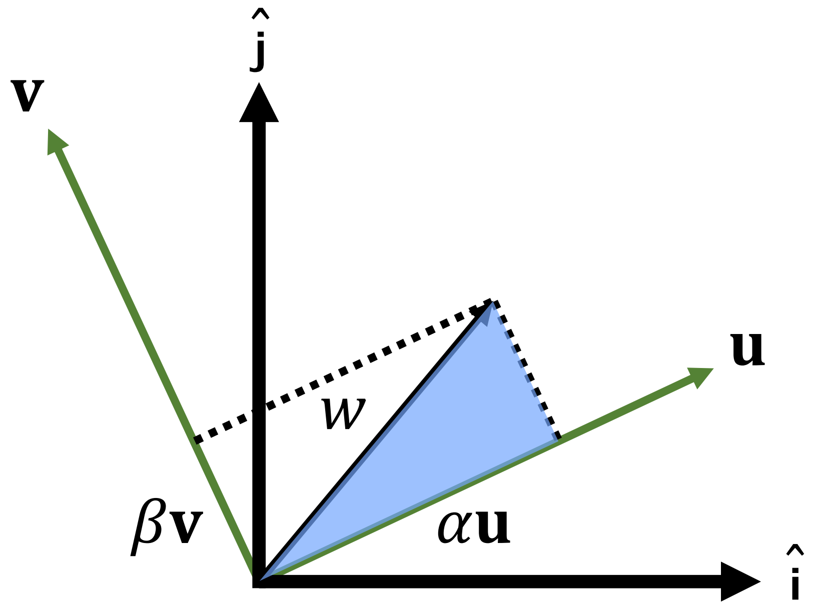
向量 (2D) 可以表示为两个向量的和



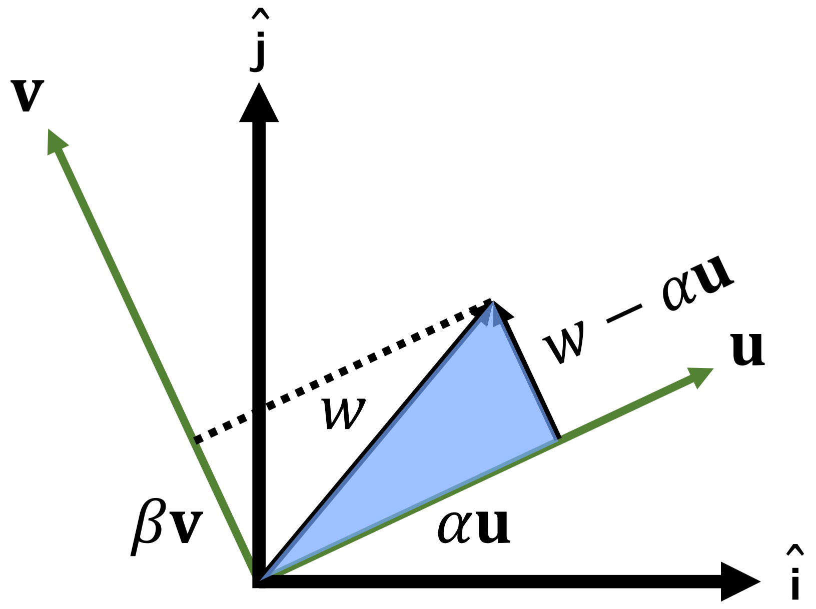
向量 (2D) 可以表示为两个向量的和



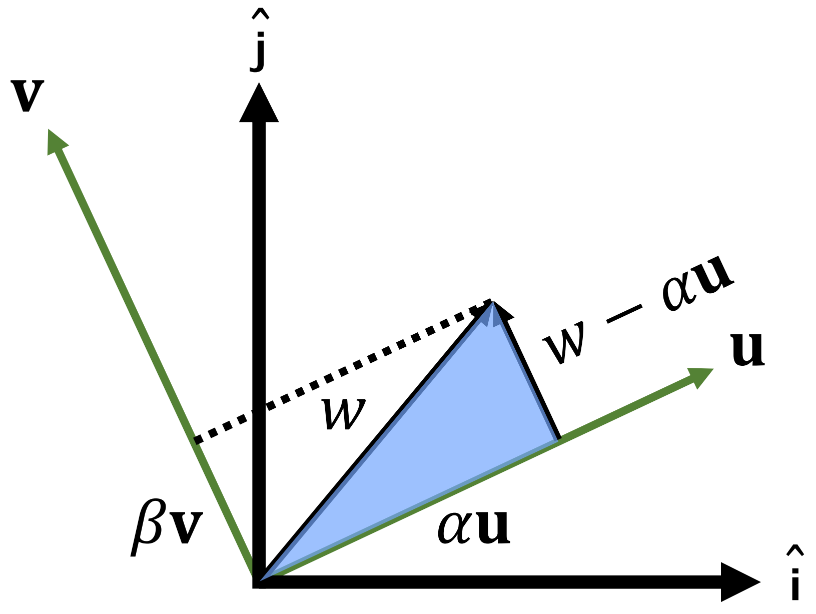
假设基是正交的，那么α和β是什么？



假设基是正交的，那么α和β是什么？

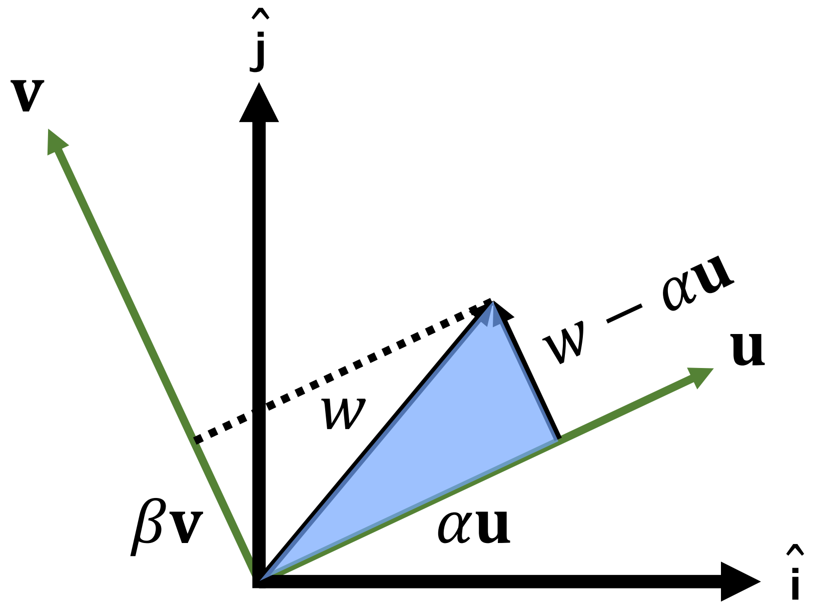


假设基是正交的，那么α和β是什么？



假设基是正交的，那么α和β是什么？

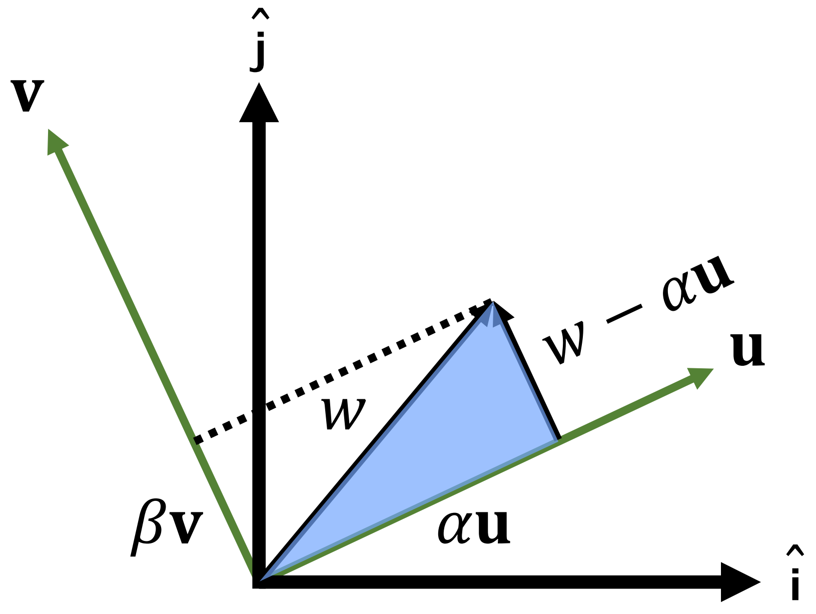
$$\alpha\mathbf{u} \cdot (w - \alpha\mathbf{u}) = 0$$



假设基是正交的，那么α和β是什么？

$$\alpha\mathbf{u} \cdot (w - \alpha\mathbf{u}) = 0$$

$$\alpha\mathbf{u} \cdot w - \alpha\mathbf{u} \cdot \alpha\mathbf{u} = 0$$

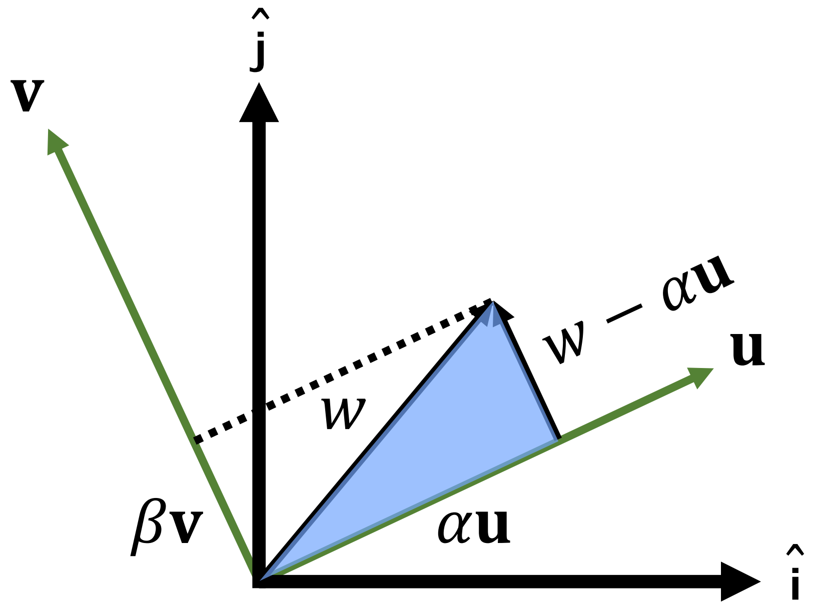


假设基是正交的，那么α和β是什么？

$$\alpha\mathbf{u} \cdot (w - \alpha\mathbf{u}) = 0$$

$$\alpha\mathbf{u} \cdot w - \alpha\mathbf{u} \cdot \alpha\mathbf{u} = 0$$

$$\alpha\mathbf{u} \cdot \mathbf{u} = \mathbf{u} \cdot w$$



假设基是正交的，那么α和β是什么？

$$\alpha\mathbf{u} \cdot (w - \alpha\mathbf{u}) = 0$$

$$\alpha\mathbf{u} \cdot w - \alpha\mathbf{u} \cdot \alpha\mathbf{u} = 0$$

$$\alpha\mathbf{u} \cdot \mathbf{u} = \mathbf{u} \cdot w$$

向量投影 $\alpha = \dfrac{\mathbf{u} \cdot w}{\mathbf{u} \cdot \mathbf{u}}$ $\beta = \dfrac{\mathbf{v} \cdot w}{\mathbf{v} \cdot \mathbf{v}}$

图像看作
向量空间
中的点

图像看作向量空间中的点

给定$N \times N$图像，它可以看作是向量

图像看作向量空间中的点

给定$N \times N$图像，它可以看作是向量

$$\begin{bmatrix} x_{00} & x_{10} & \cdots & x_{(N-1)(N-1)} \end{bmatrix}^{\mathrm{T}}$$

图像看作向量空间中的点

给定$N \times N$图像，它可以看作是向量

$$\begin{bmatrix} x_{00} & x_{10} & \cdots & x_{(N-1)(N-1)} \end{bmatrix}^{\mathrm{T}}$$

标准基是将单个像素设置为1的向量集

图像看作向量空间中的点

给定$N \times N$图像，它可以看作是向量

$$\begin{bmatrix} x_{00} & x_{10} & \cdots & x_{(N-1)(N-1)} \end{bmatrix}^{\mathrm{T}}$$

标准基是将单个像素设置为1的向量集

$$\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}}$$

标准
图像基

3	8
10	50

标准
图像基

$$\underset{\mathbf{u}}{\begin{bmatrix}3\\8\\10\\50\end{bmatrix}}$$

标准
图像基

$$\underset{\mathbf{u}}{\begin{bmatrix} 3 \\ 8 \\ 10 \\ 50 \end{bmatrix}} = 3 \underset{\mathbf{e}_1}{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}}$$

标准
图像基

$$\underset{\mathbf{u}}{\begin{array}{|c|}\hline 3 \\ \hline 8 \\ \hline 10 \\ \hline 50 \\ \hline\end{array}} = 3\underset{\mathbf{e}_1}{\begin{array}{|c|}\hline 1 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline\end{array}} + 8\underset{\mathbf{e}_2}{\begin{array}{|c|}\hline 0 \\ \hline 1 \\ \hline 0 \\ \hline 0 \\ \hline\end{array}}$$

标准
图像基

$$\underset{\mathbf{u}}{\begin{bmatrix}3\\8\\10\\50\end{bmatrix}} = 3\underset{\mathbf{e}_1}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}} + 8\underset{\mathbf{e}_2}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}} + 10\underset{\mathbf{e}_3}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}$$

标准图像基

$$\underset{\mathbf{u}}{\begin{bmatrix}3\\8\\10\\50\end{bmatrix}} = 3\underset{\mathbf{e}_1}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}} + 8\underset{\mathbf{e}_2}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}} + 10\underset{\mathbf{e}_3}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}} + 50\underset{\mathbf{e}_4}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}$$

标准
图像基

$$\underset{\mathbf{u}}{\begin{bmatrix}3\\8\\10\\50\end{bmatrix}} = 3\underset{\mathbf{e}_1}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}} + 8\underset{\mathbf{e}_2}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}} + 10\underset{\mathbf{e}_3}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}} + 50\underset{\mathbf{e}_4}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}$$

$$\mathbf{u} = \sum \alpha_i \mathbf{e}_i$$

标准图像基

$$\underset{\mathbf{u}}{\begin{bmatrix}3\\8\\10\\50\end{bmatrix}} = 3\underset{\mathbf{e}_1}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}} + 8\underset{\mathbf{e}_2}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}} + 10\underset{\mathbf{e}_3}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}} + 50\underset{\mathbf{e}_4}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}$$

$$\mathbf{u} = \sum \alpha_i \mathbf{e}_i = \sum (\mathbf{u} \cdot \mathbf{e}_i)\mathbf{e}_i$$

标准图像基

$$\underset{\mathbf{u}}{\begin{bmatrix}3\\8\\10\\50\end{bmatrix}} = 3\underset{\mathbf{e}_1}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}} + 8\underset{\mathbf{e}_2}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}} + 10\underset{\mathbf{e}_3}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}} + 50\underset{\mathbf{e}_4}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}$$

$$\mathbf{u} = \sum \alpha_i \mathbf{e}_i = \sum \boxed{(\mathbf{u}\cdot\mathbf{e}_i)}\mathbf{e}_i$$

标准
图像基

$$\underbrace{\begin{bmatrix}3\\8\\10\\50\end{bmatrix}}_{\mathbf{u}} = 3\underbrace{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}}_{\mathbf{e}_1} + 8\underbrace{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}}_{\mathbf{e}_2} + 10\underbrace{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}_{\mathbf{e}_3} + 50\underbrace{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}_{\mathbf{e}_4}$$

$$\mathbf{u} = \sum \alpha_i \mathbf{e}_i = \sum (\mathbf{u}\cdot\mathbf{e}_i)\mathbf{e}_i$$

向量投影

回顾：线性代数

已结束

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

其中

$$e^{-i2\pi\omega x} = \cos(-2\pi\omega x) + i\sin(-2\pi\omega x)$$

$$F(\omega) = \underline{\int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx} + i\underline{\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx}$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

其中

$$e^{-i2\pi\omega x} = \cos(-2\pi\omega x) + i\sin(-2\pi\omega x)$$

$$F(\omega) = \underline{\int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx} + i\,\underline{\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx}$$

信号在正弦基集上的投影

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

信号在正弦基集上的投影

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

正弦和余弦是正交的

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

正弦和余弦是正交的

$$\langle \cos(\omega_i x), \sin(\omega_j x) \rangle = \int_{-\pi}^{\pi} \cos(\omega_i x)\sin(\omega_j x)\,dx = 0$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

不同频率的正弦波是正交的

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

不同频率的正弦波是正交的

$$\langle \sin(\omega_i x), \sin(\omega_j x)\rangle = \int_{-\pi}^{\pi} \sin(\omega_i x)\sin(\omega_j x)\,dx = \pi\delta_{ij}$$

$$\langle \cos(\omega_i x), \cos(\omega_j x)\rangle = \int_{-\pi}^{\pi} \cos(\omega_i x)\cos(\omega_j x)\,dx = \pi\delta_{ij}$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

不同频率的正弦波是正交的

$$\langle \sin(\omega_i x), \sin(\omega_j x)\rangle = \int_{-\pi}^{\pi} \sin(\omega_i x)\sin(\omega_j x)\,dx = \pi\delta_{ij}$$

$$\langle \cos(\omega_i x), \cos(\omega_j x)\rangle = \int_{-\pi}^{\pi} \cos(\omega_i x)\cos(\omega_j x)\,dx = \pi\delta_{ij}$$

1D傅里叶变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x)\cos(-2\pi\omega x)\,dx + i\int_{-\infty}^{\infty} f(x)\sin(-2\pi\omega x)\,dx$$

不同频率的正弦波是正交的

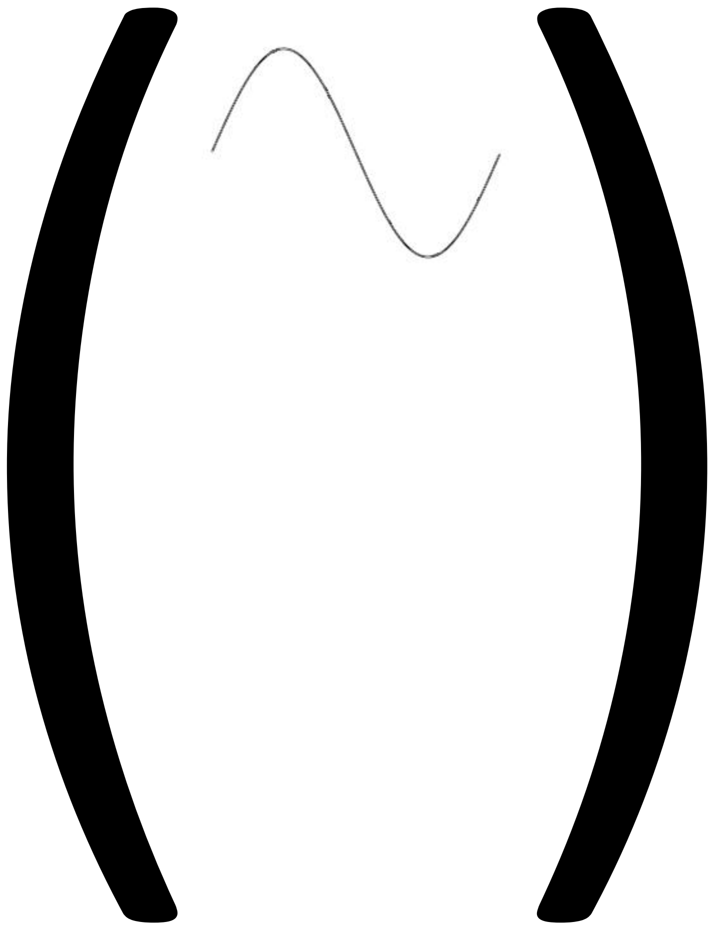
$$\langle \sin(\omega_i x), \sin(\omega_j x)\rangle = \int_{-\pi}^{\pi} \sin(\omega_i x)\sin(\omega_j x)\,dx = \pi\delta_{ij}$$

$$\delta_{ij} = \begin{cases} 1, \text{if } i = j \\ 0, \text{if } i \neq j \end{cases}$$

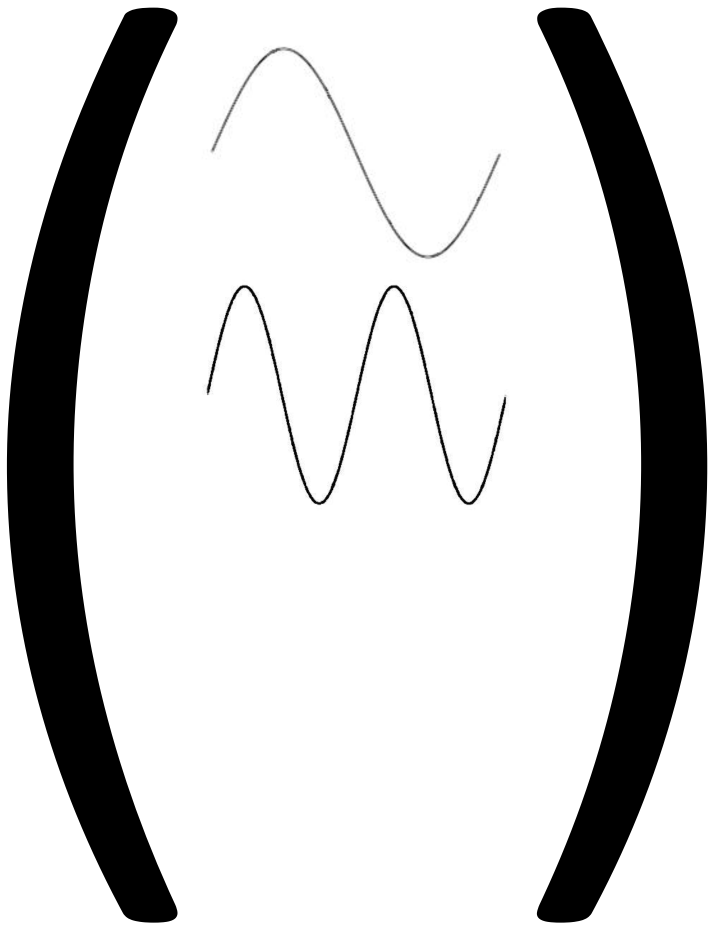
$$\langle \cos(\omega_i x), \cos(\omega_j x)\rangle = \int_{-\pi}^{\pi} \cos(\omega_i x)\cos(\omega_j x)\,dx = \pi\delta_{ij}$$

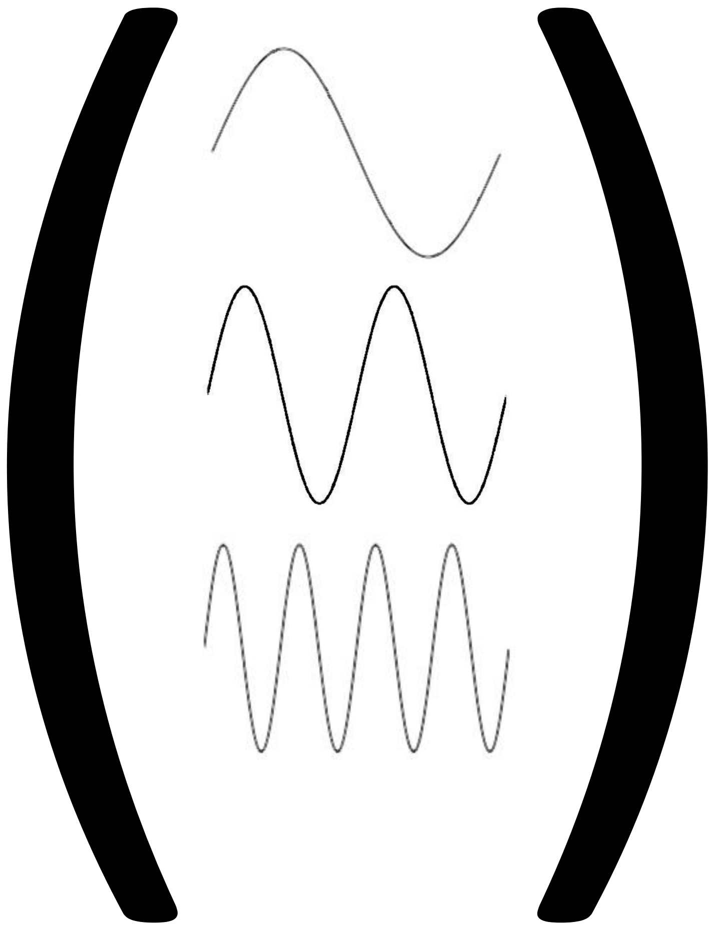
傅里叶变换直观

傅里叶变换直观



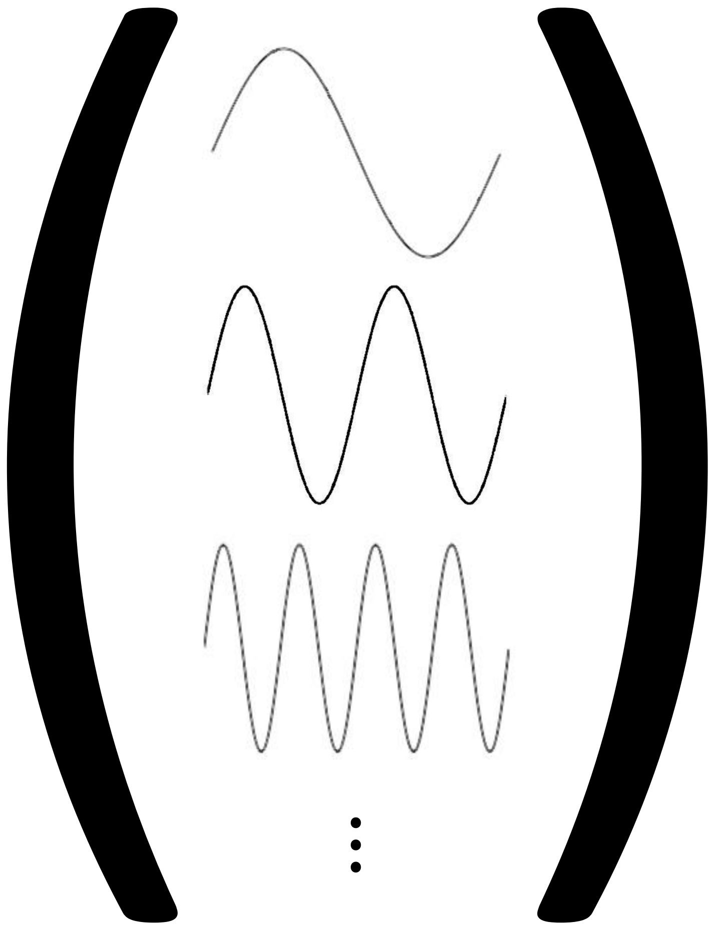
傅里叶变换直观



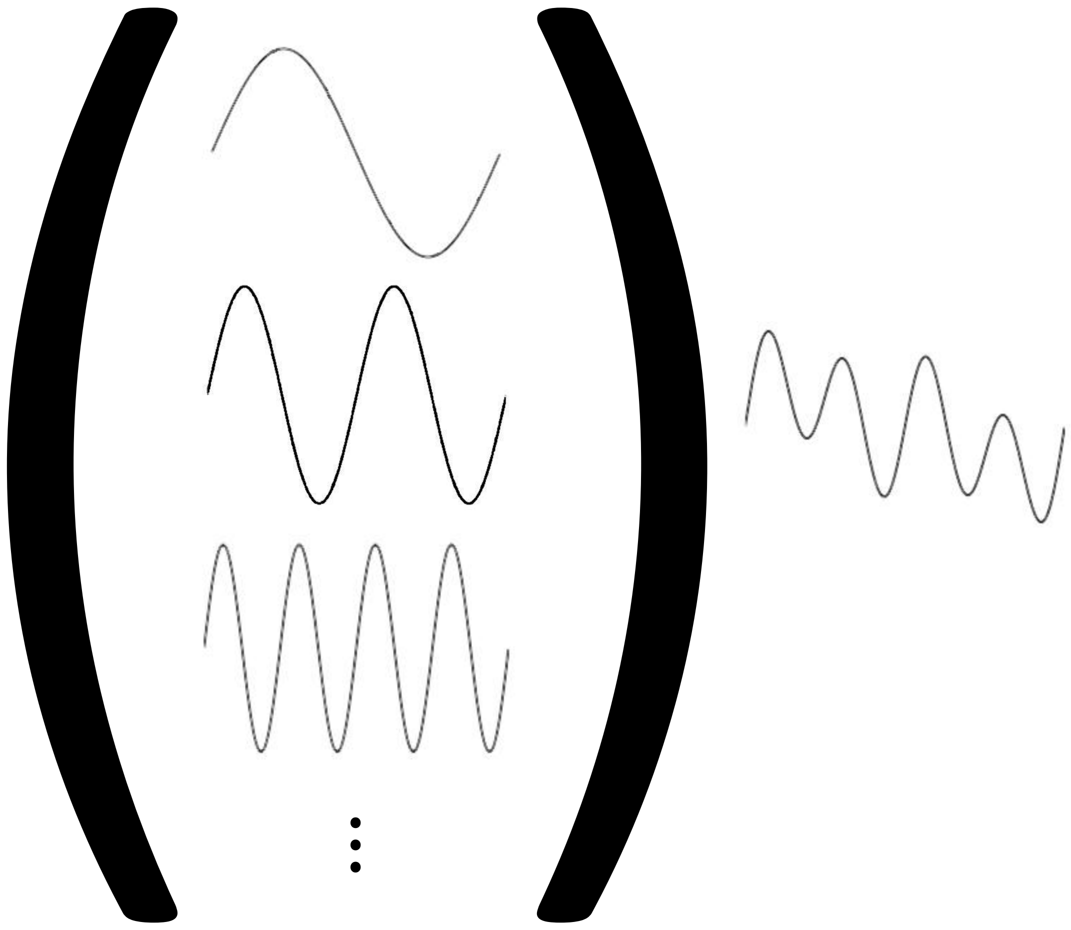


傅里叶变换直观

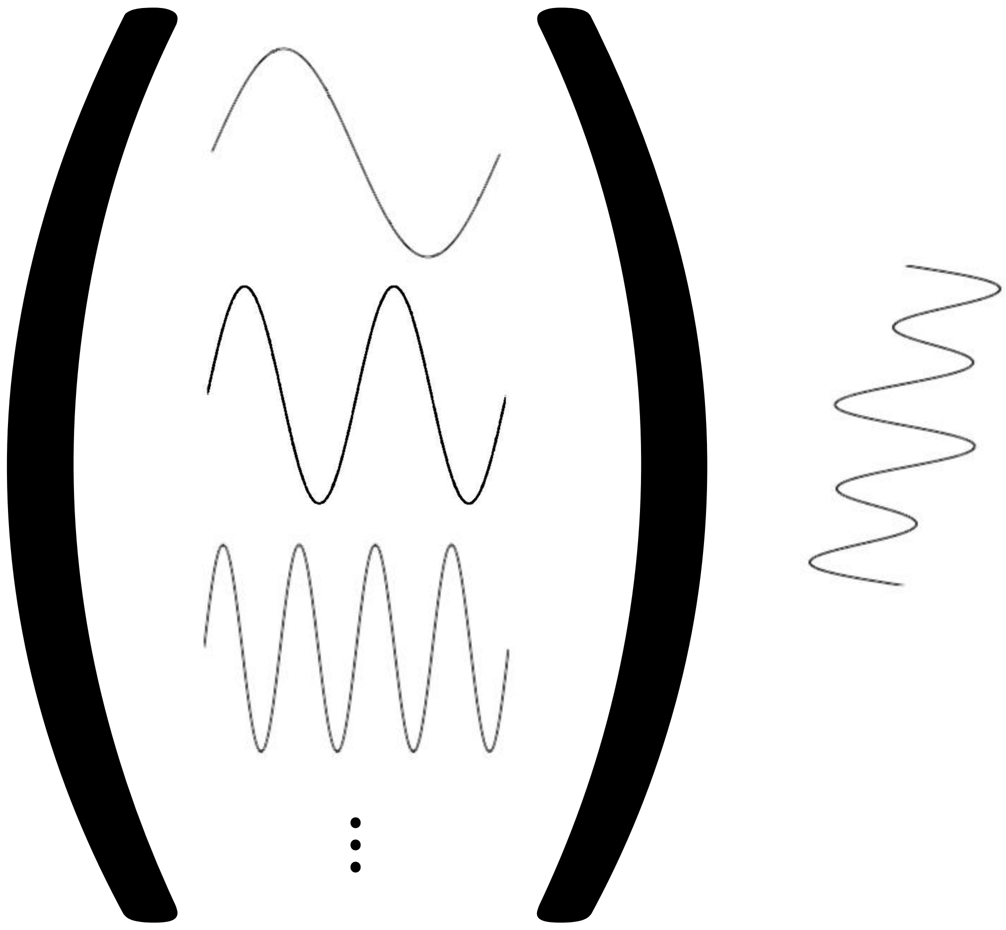
傅里叶变换直观



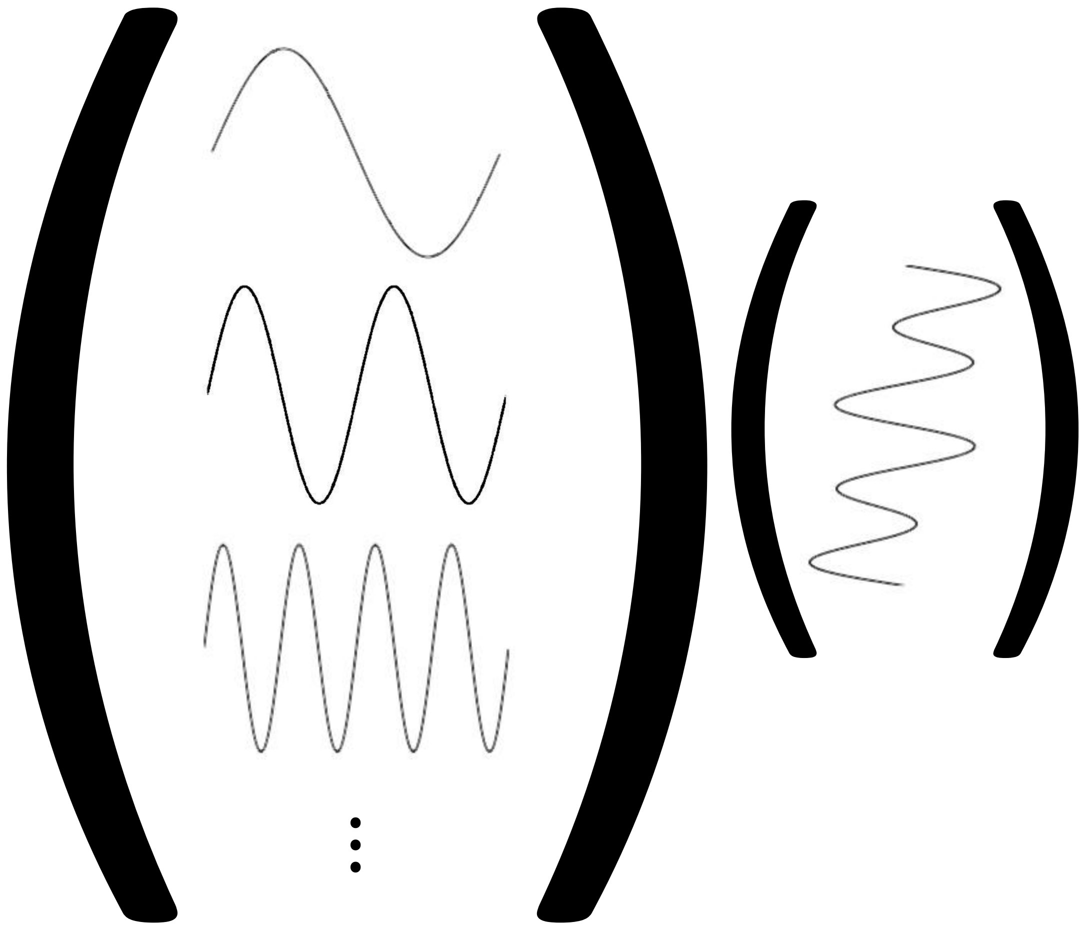
傅里叶变换直观



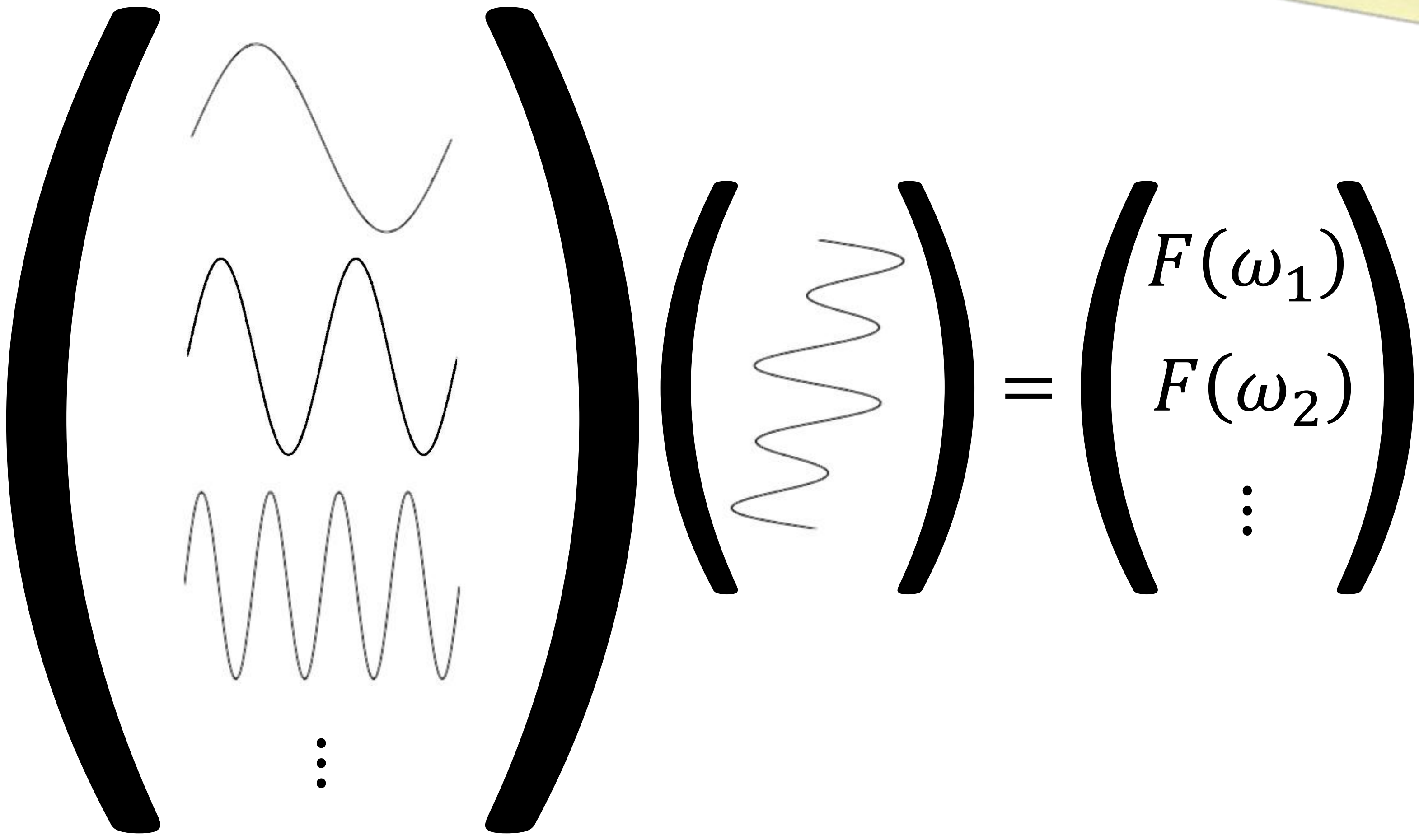
傅里叶变换直观



傅里叶变换直观

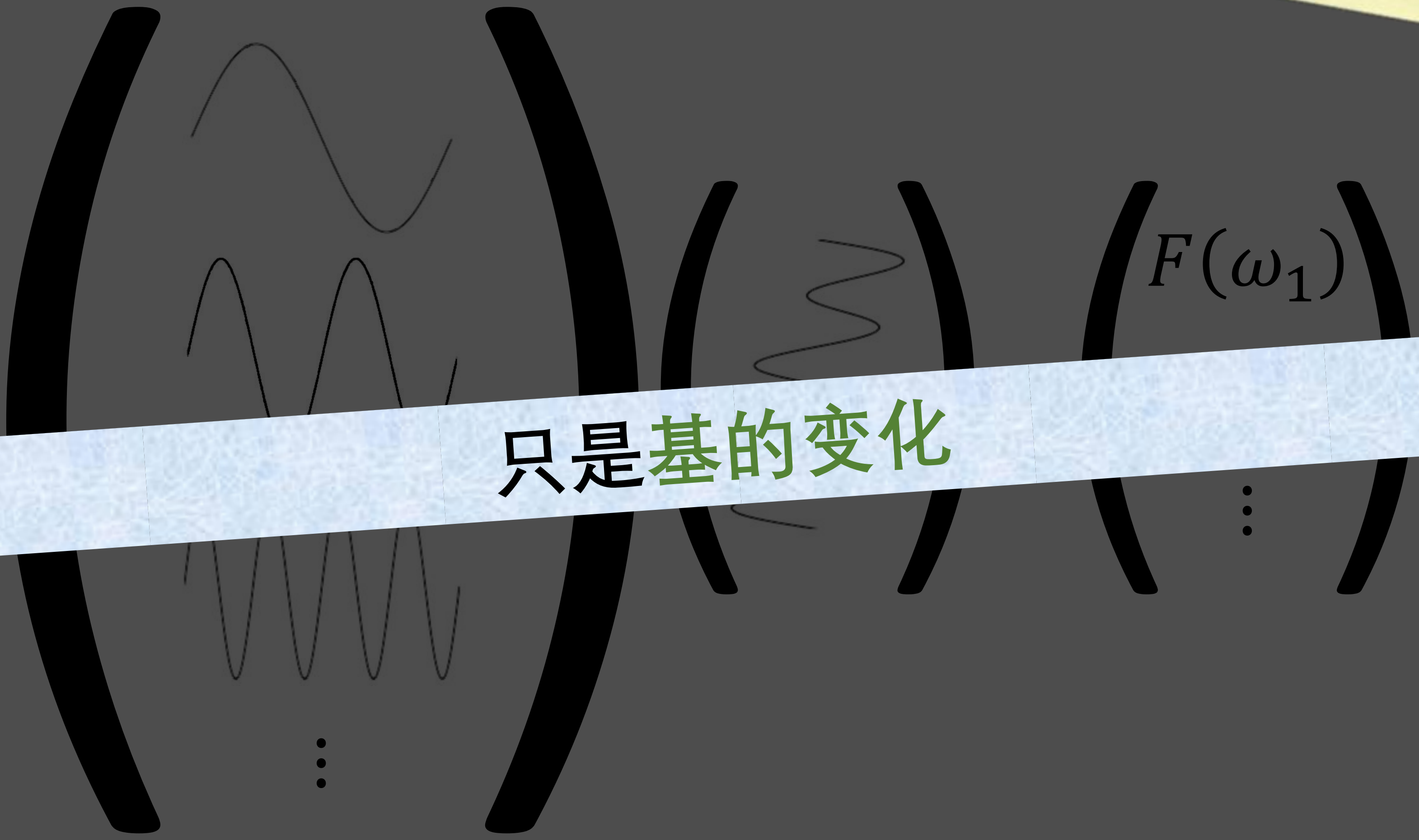


傅里叶变换直观



$$= \begin{pmatrix} F(\omega_1) \\ F(\omega_2) \\ \vdots \end{pmatrix}$$

傅里叶变换直观



只是基的变化

$$\begin{tikzcd} F(\omega) \arrow[r] & \boxed{\text{傅里叶逆变换}} \arrow[r] & f(x) \end{tikzcd}$$

$$F(\omega) \longrightarrow \boxed{\text{傅里叶逆变换}} \longrightarrow f(x)$$

非近似

1D傅里叶逆变换

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{i2\pi\omega x} d\omega$$

其中

$$e^{i2\pi\omega x} = \cos(2\pi\omega x) + i\sin(2\pi\omega x)$$

1D傅里叶逆变换

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{i2\pi\omega x} d\omega$$

其中

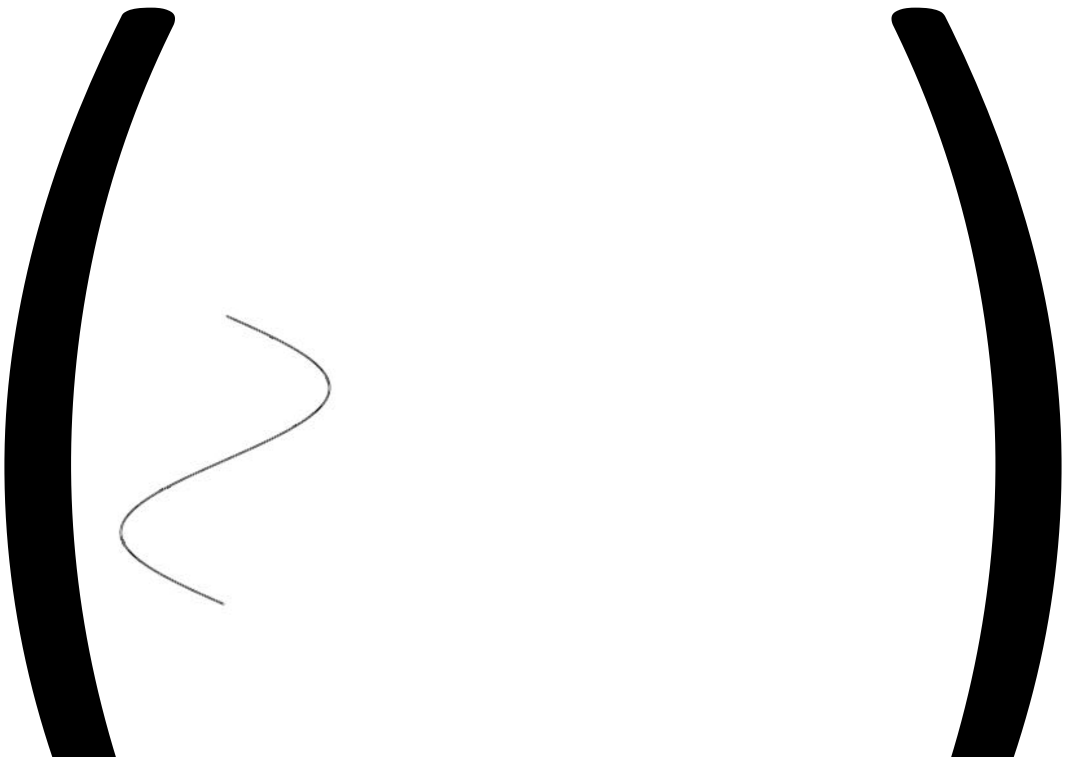
$$e^{i2\pi\omega x} = \cos(2\pi\omega x) + i\sin(2\pi\omega x)$$

结合每个频率的贡献
来表示"空间"域中的信号

傅里叶逆
变换直观

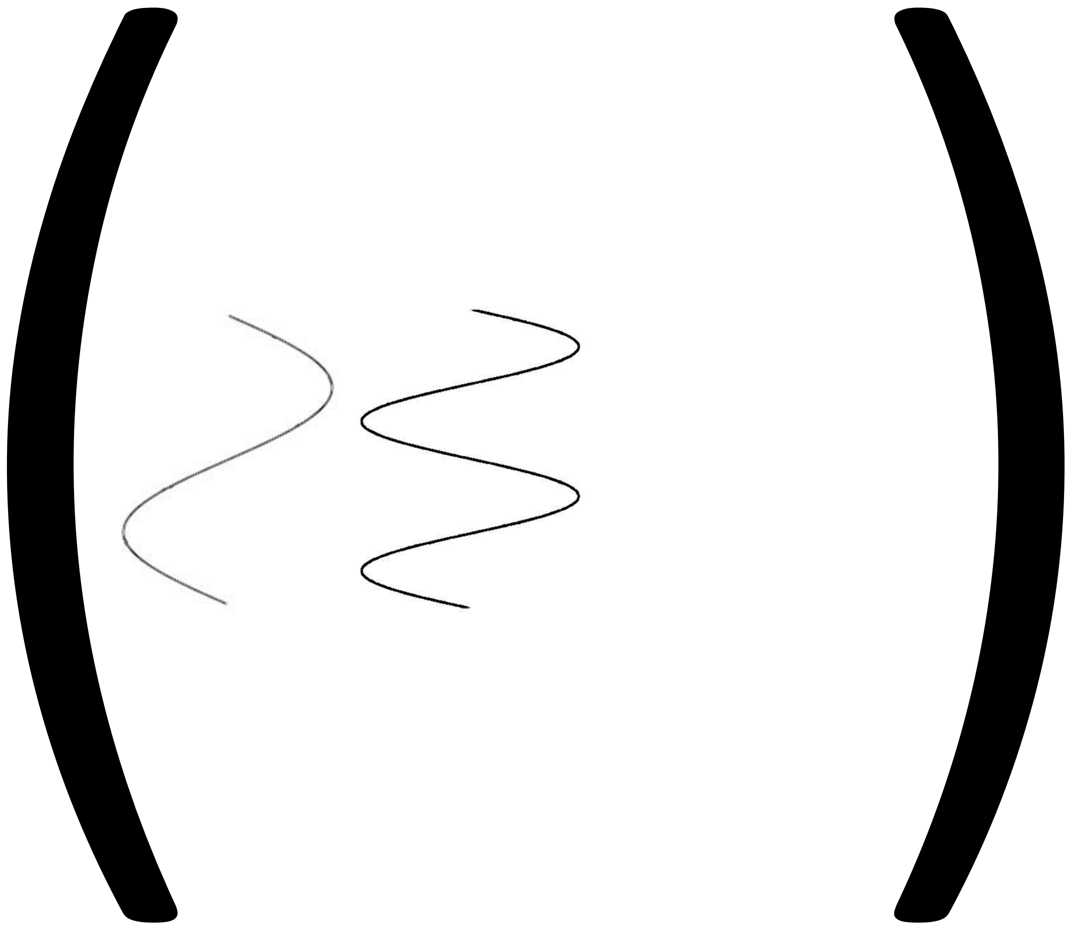
傅里叶逆变换直观

$$\left(\qquad\qquad\right)$$

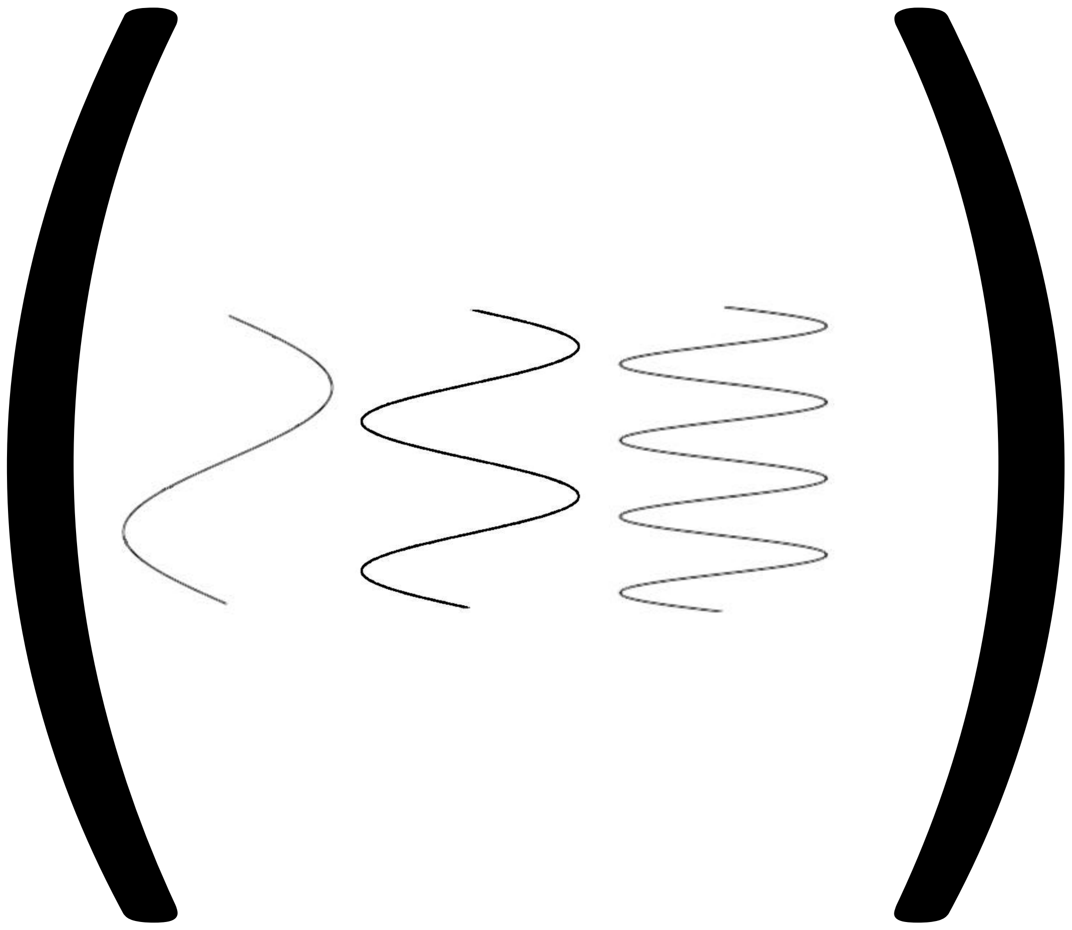


傅里叶逆变换直观

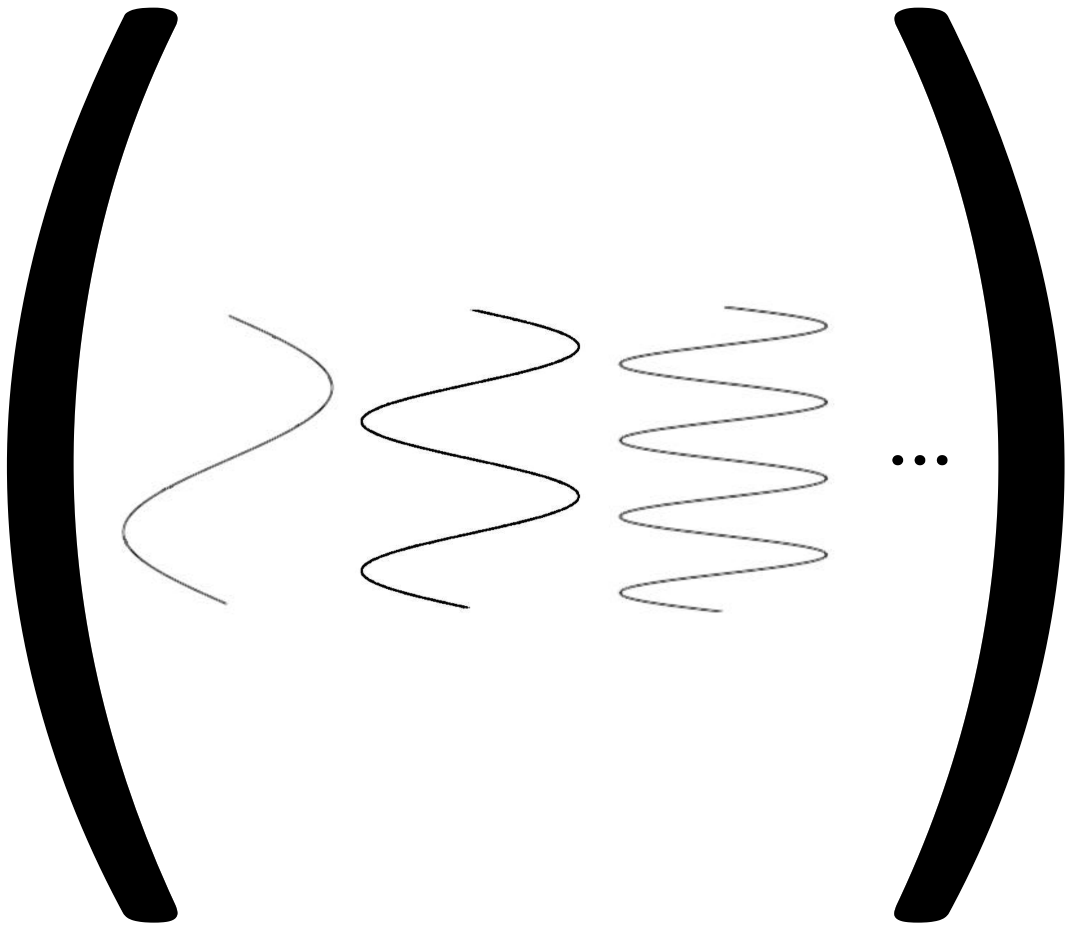
傅里叶逆变换直观



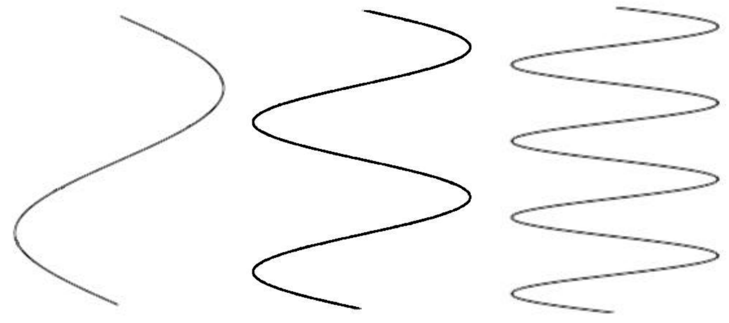
傅里叶逆变换直观



傅里叶逆变换直观

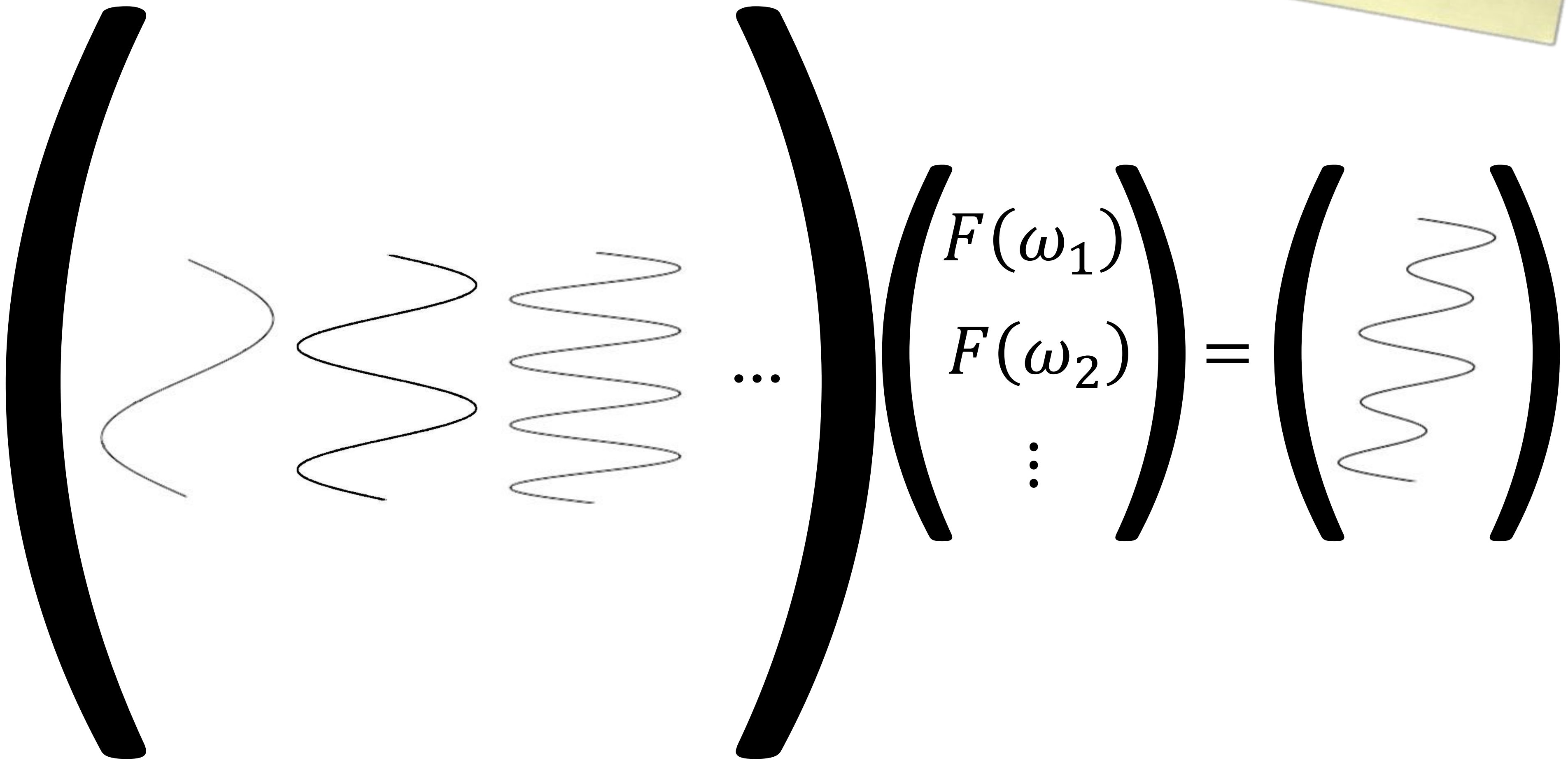


傅里叶逆变换直观



$$\left(\quad \cdots \right)\begin{pmatrix} F(\omega_1) \\ F(\omega_2) \\ \vdots \end{pmatrix}$$

傅里叶逆变换直观



正变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

正变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi\omega x}dx$$

vs.

$$f(x) = \int_{-\infty}^{\infty} F(\omega)e^{i2\pi\omega x}d\omega$$

逆变换

正变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{\boxed{-i2\pi\omega x}} dx$$

vs.

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{i2\pi\omega x} d\omega$$

逆变换

正变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{\boxed{-i2\pi\omega x}} dx$$

vs.

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{\boxed{i2\pi\omega x}} d\omega$$

逆变换

正变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

vs. $\mathbf{u} = \sum (\mathbf{u}\cdot\mathbf{e}_i)\mathbf{e}_i$

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{i2\pi\omega x} d\omega$$

逆变换

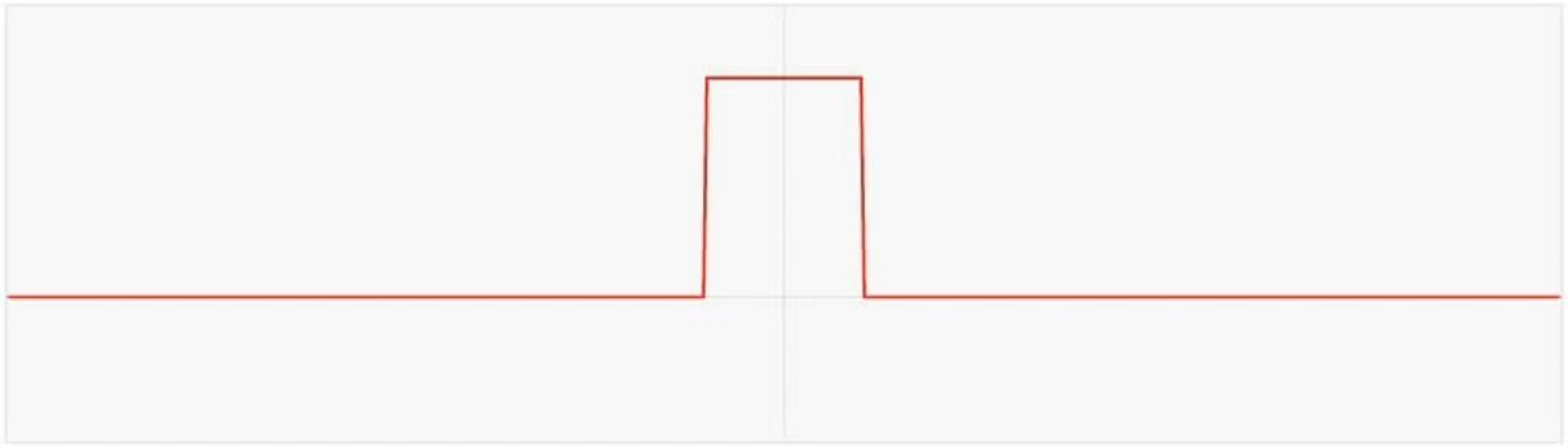
正变换

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

vs. $\mathbf{u} = \sum (\mathbf{u} \cdot \bar{\mathbf{e}}_i)\mathbf{e}_i$

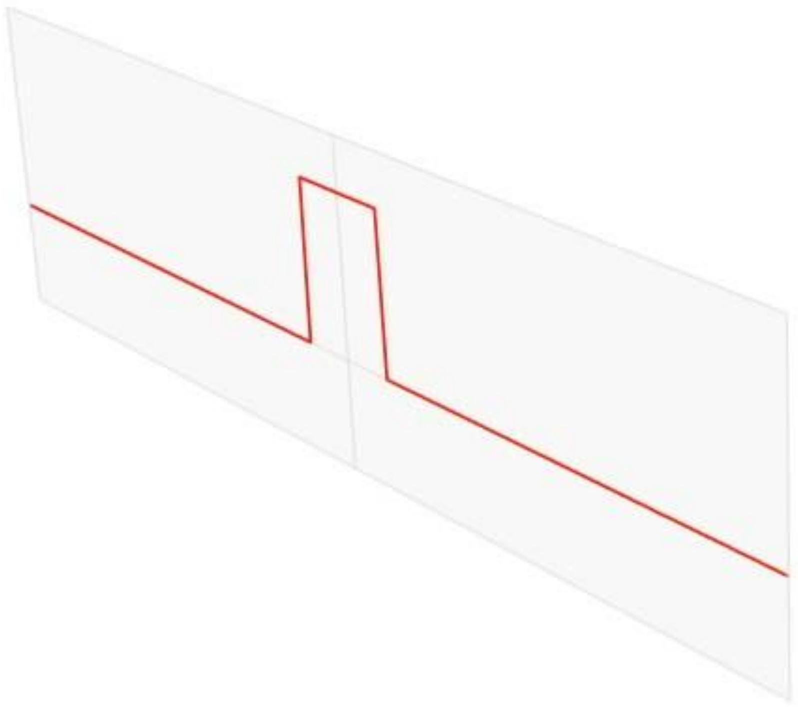
$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{i2\pi\omega x} d\omega$$

逆变换

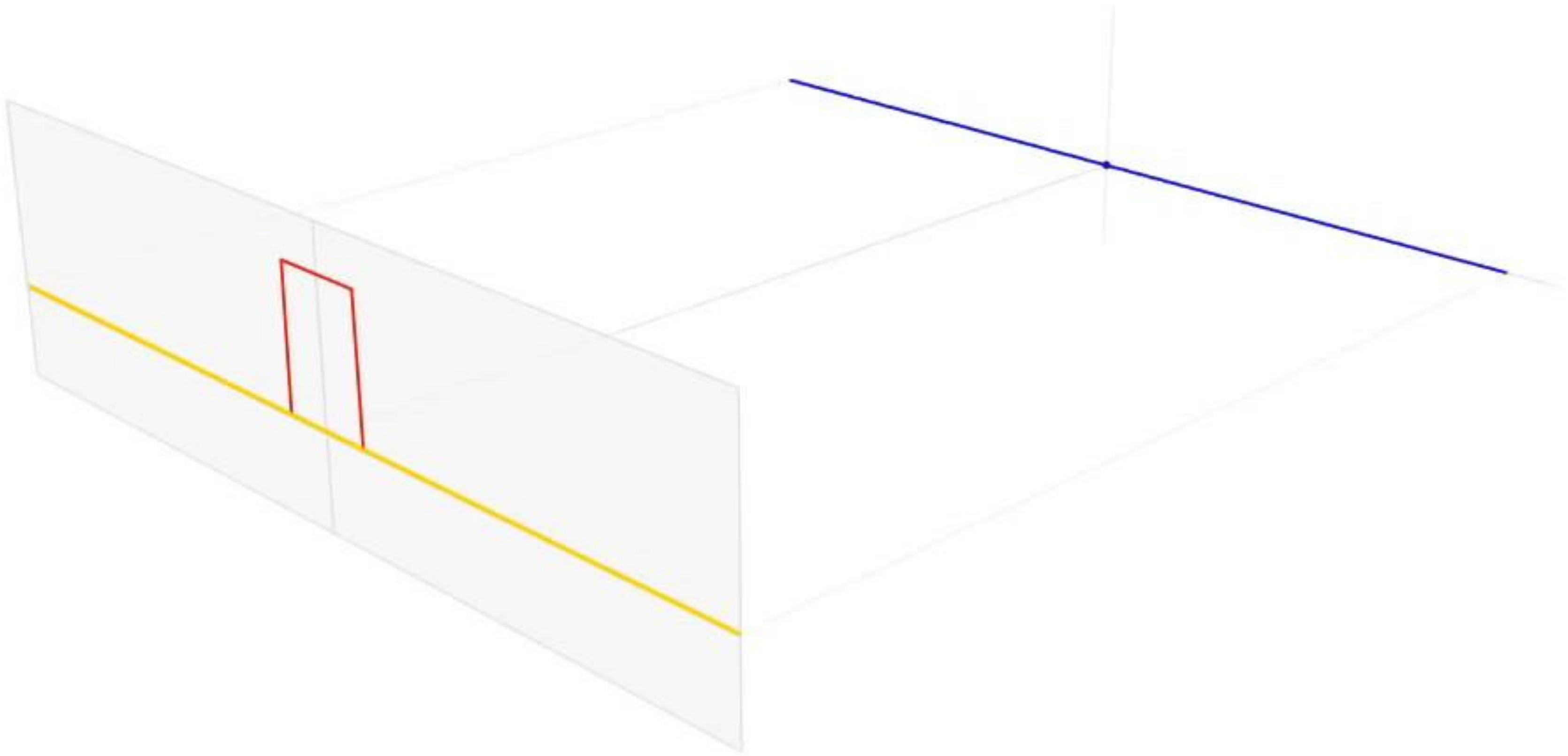


$f(x)$

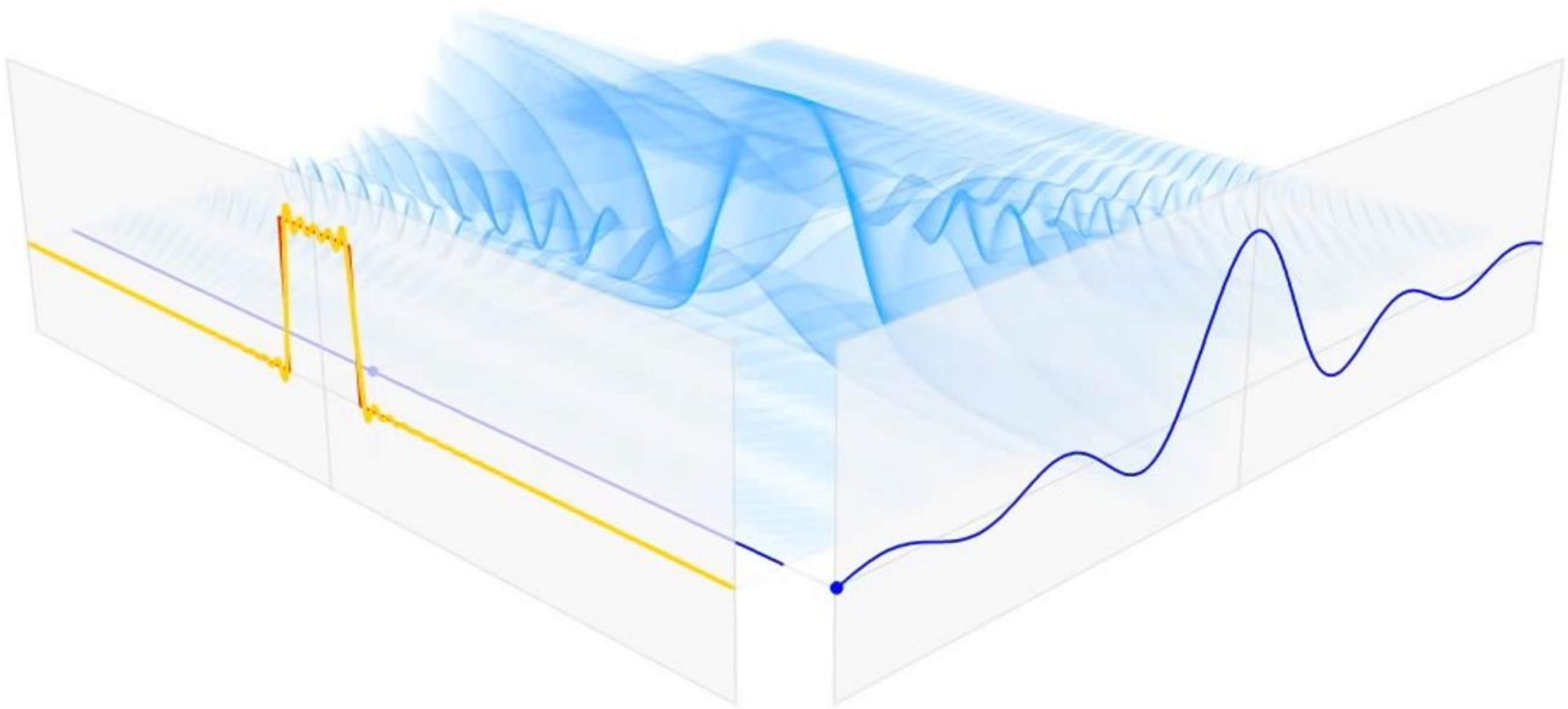
$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$



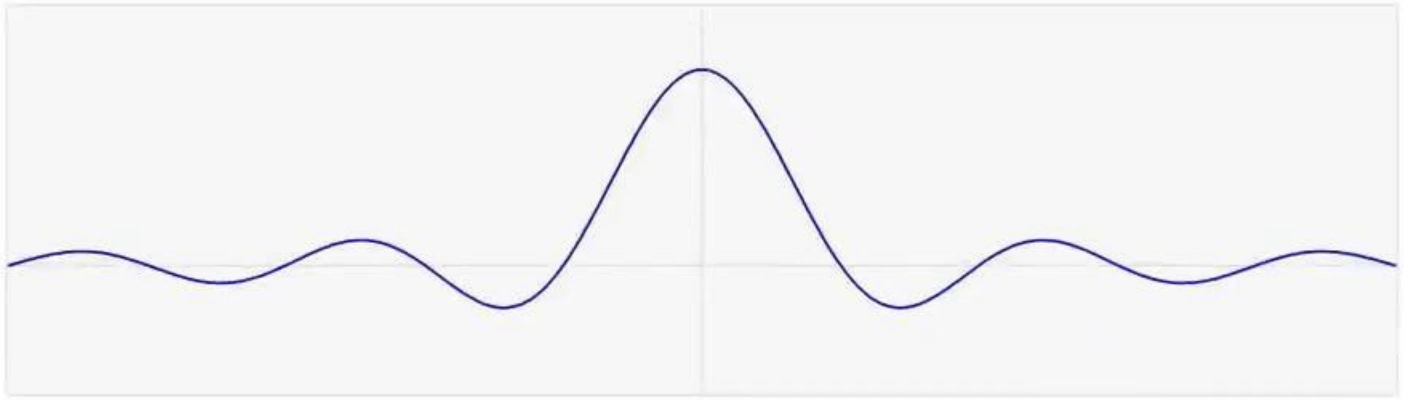
$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$



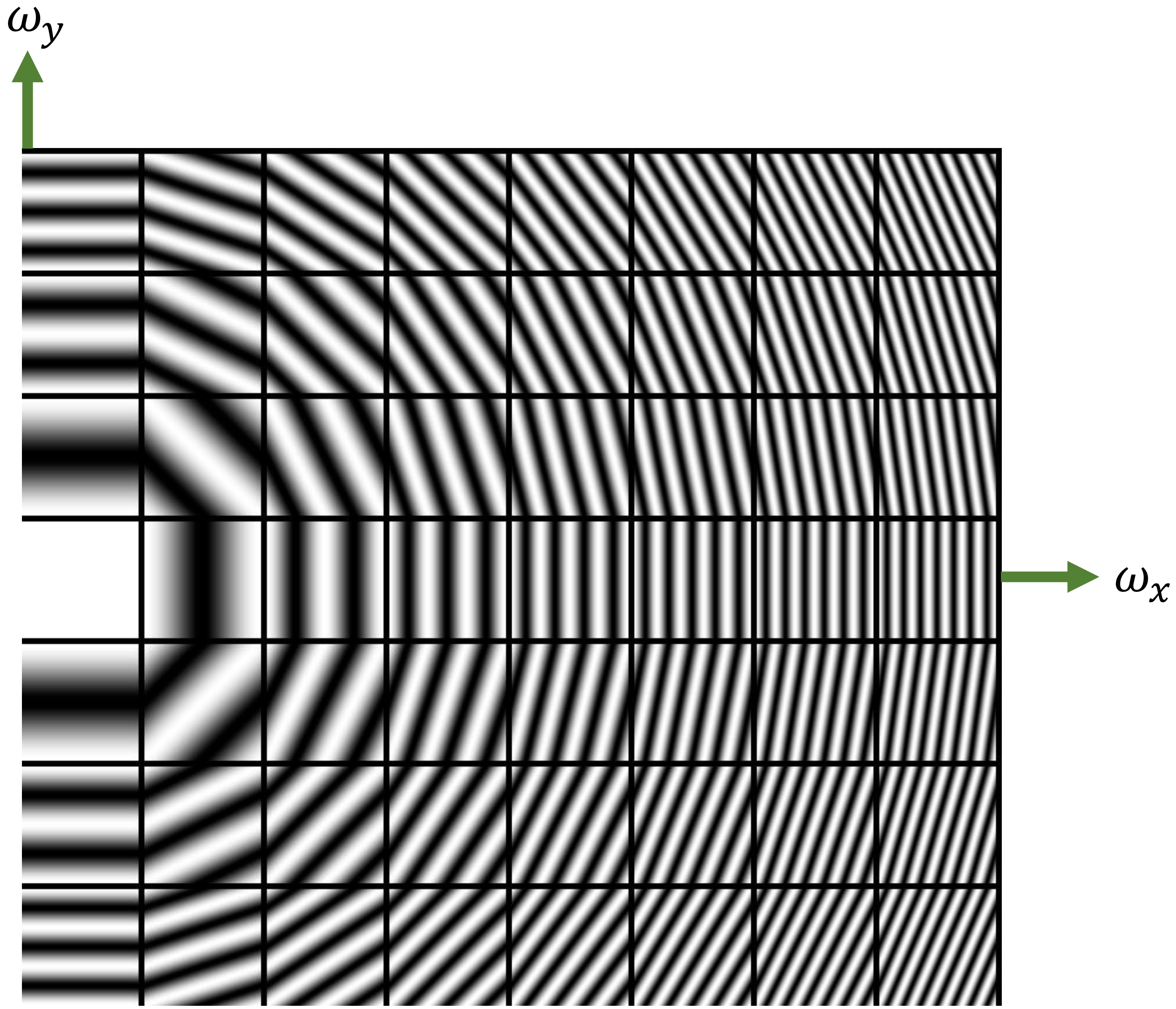
$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

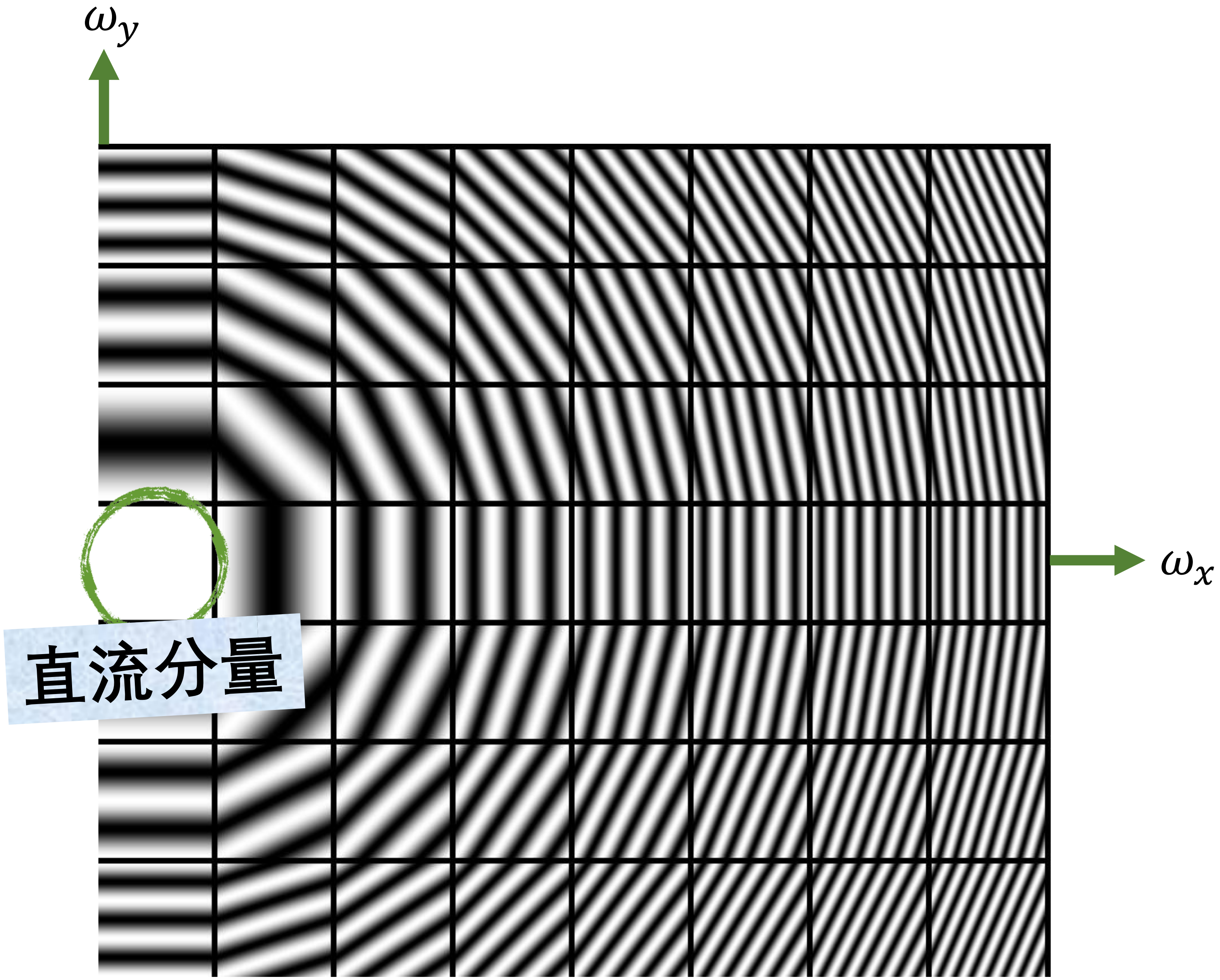


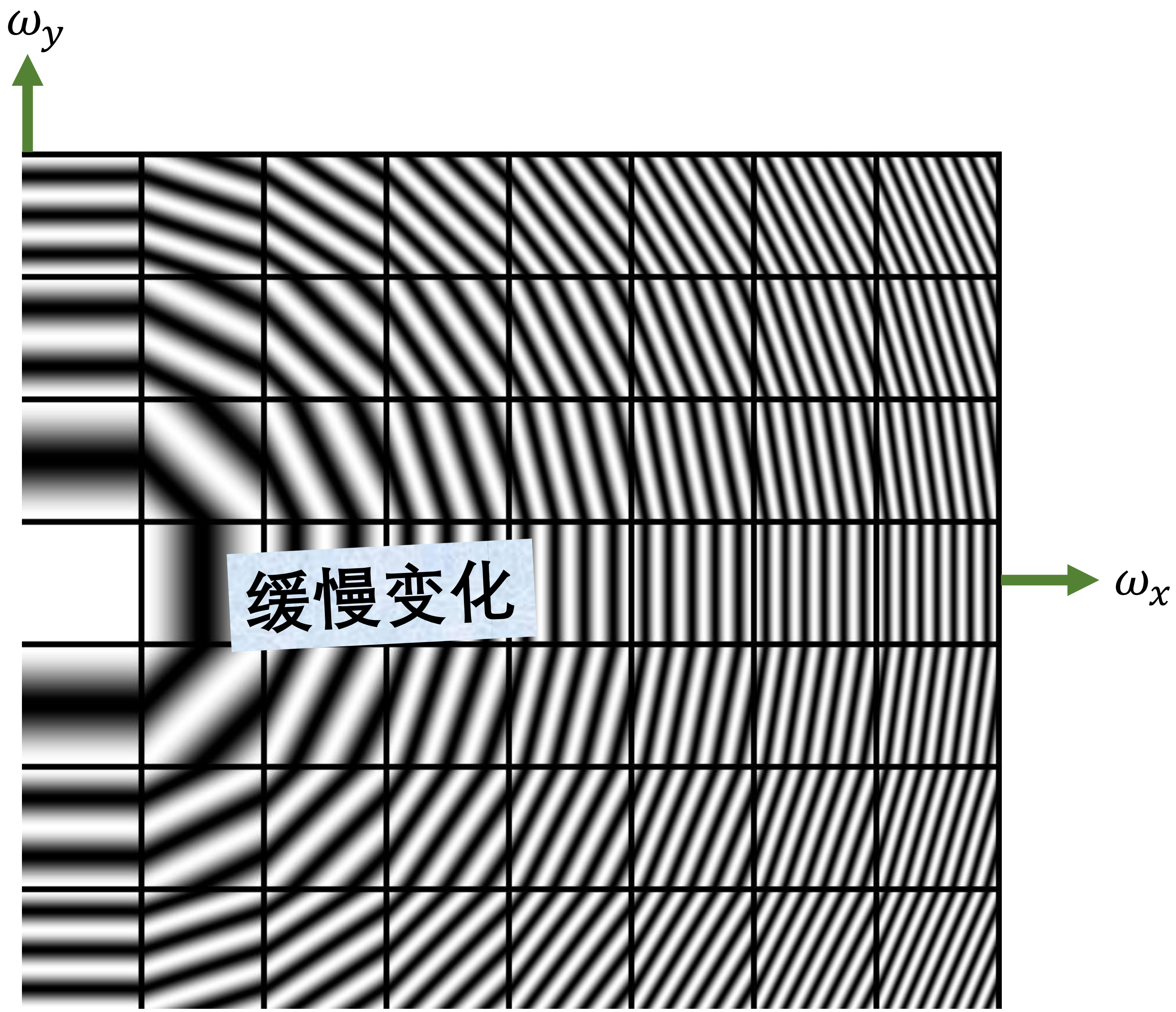
$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi\omega x} dx$$

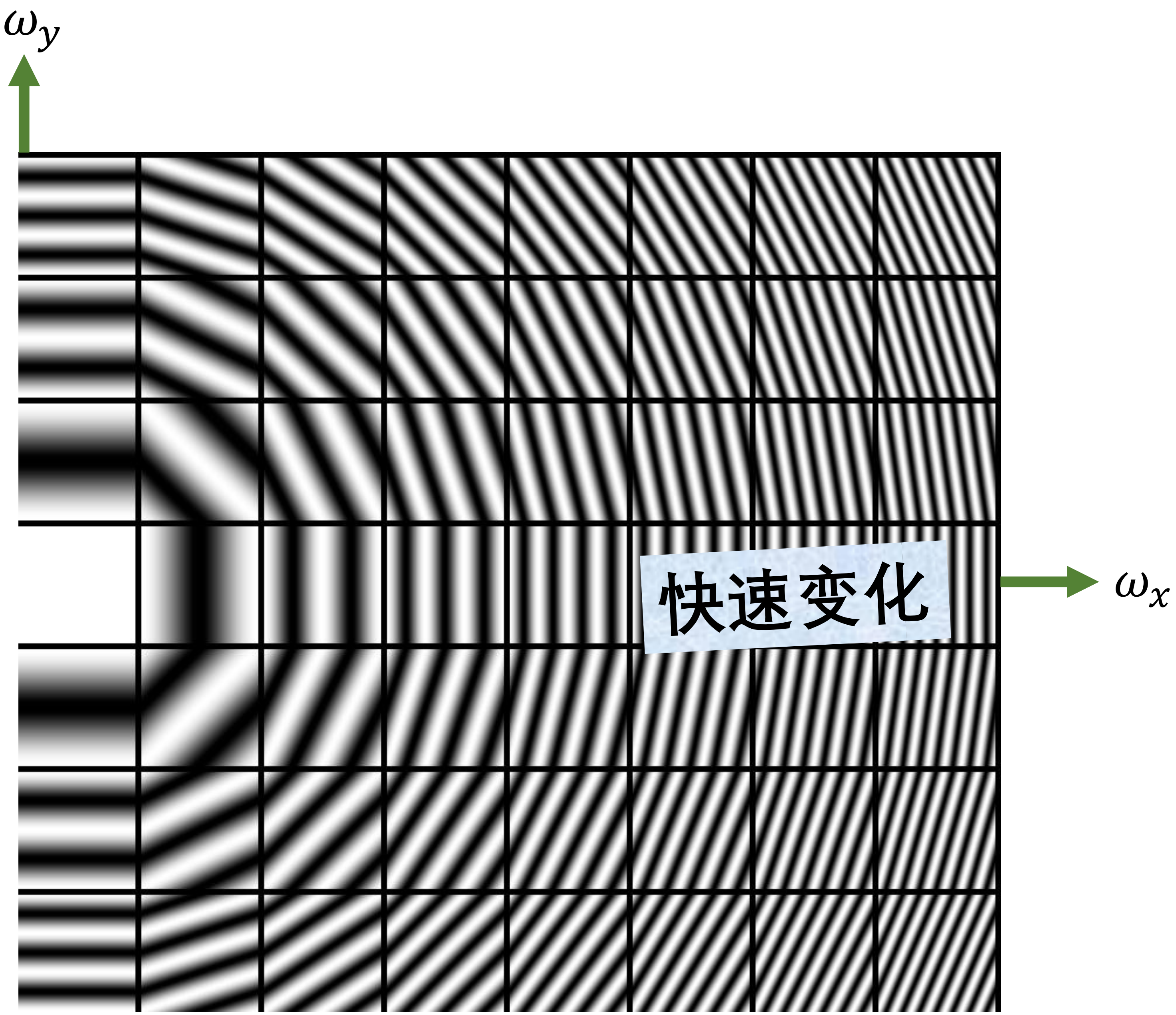


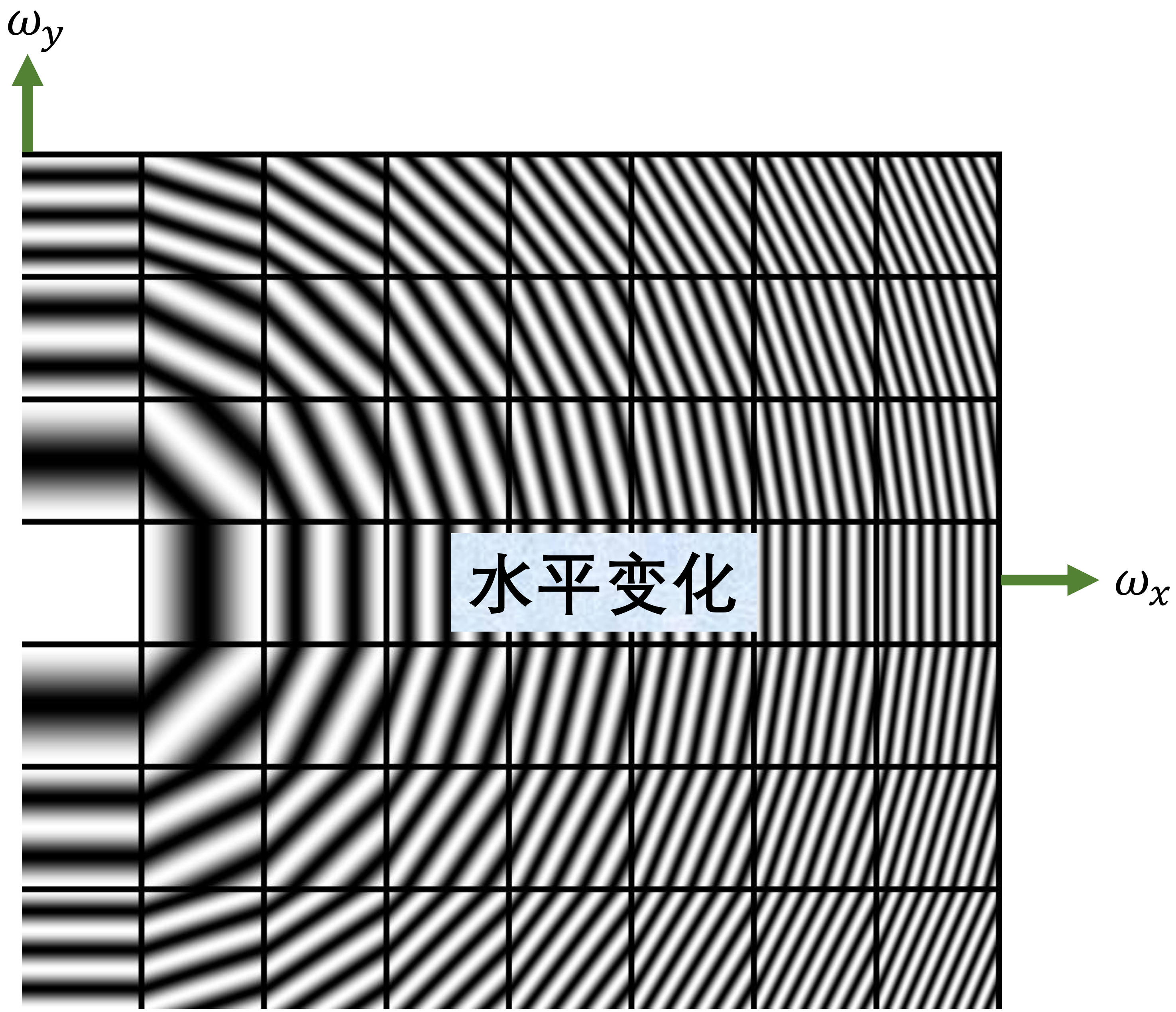
2D 怎么办?

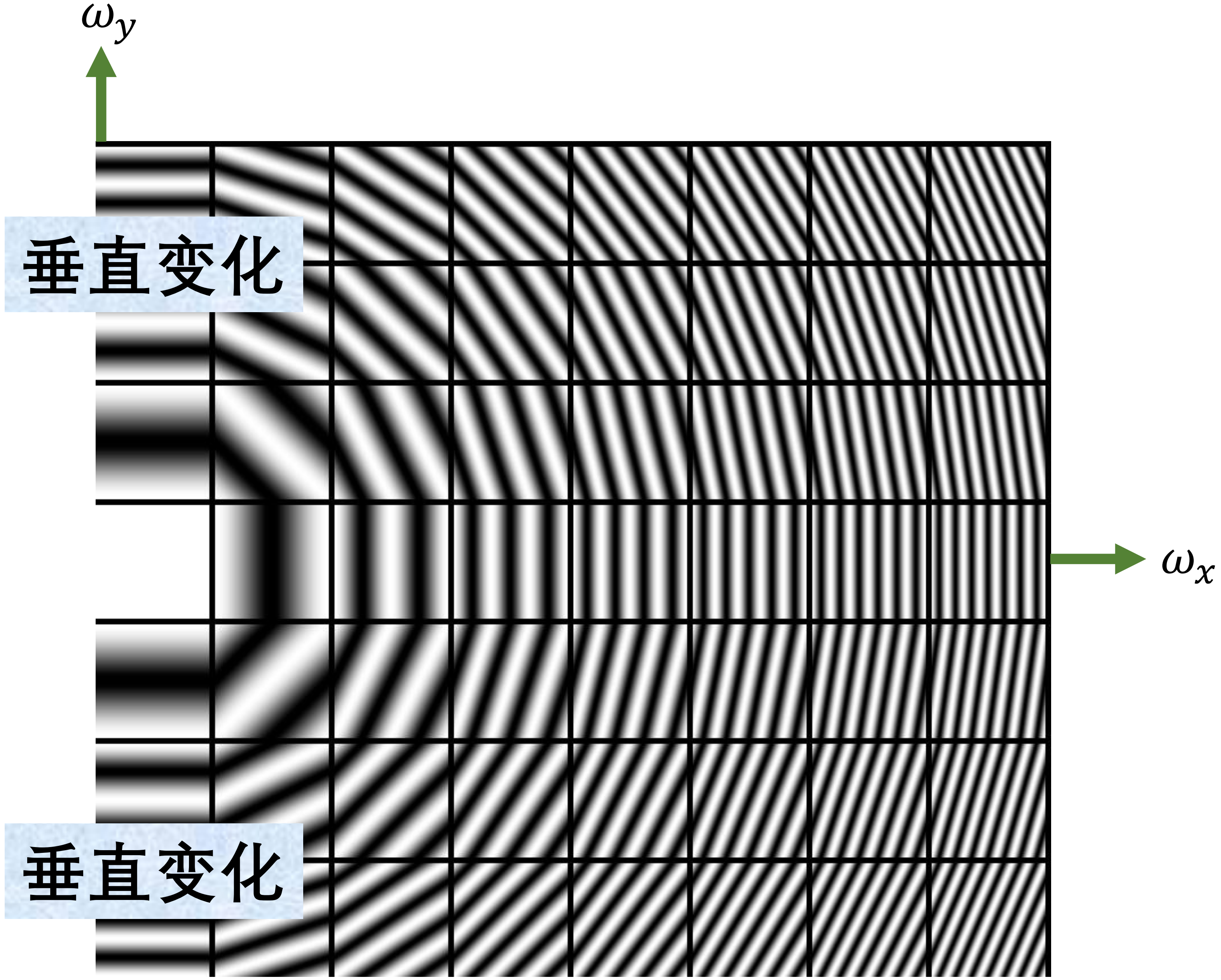

ω_y
ω_x

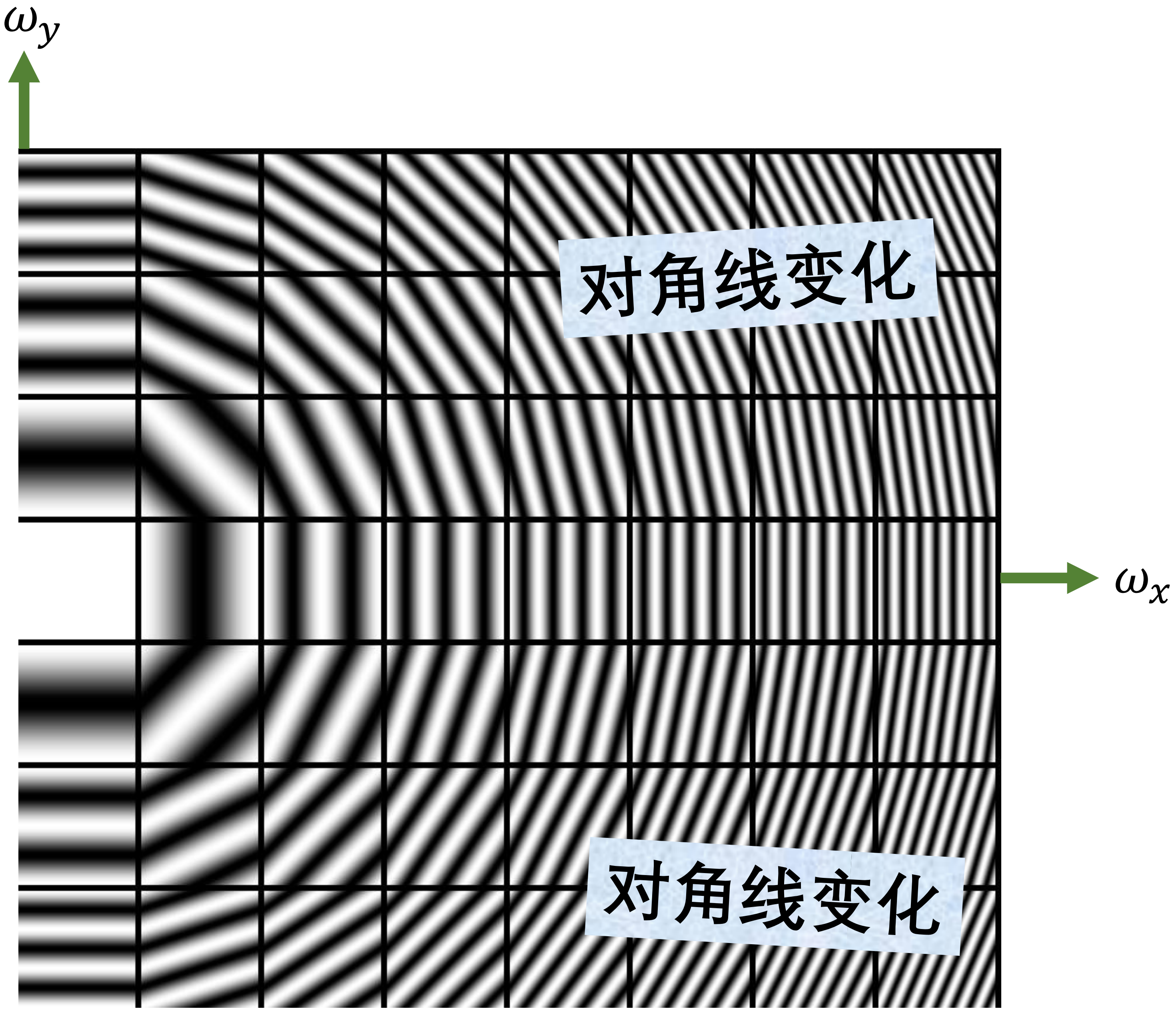

ω_y
ω_x
直流分量

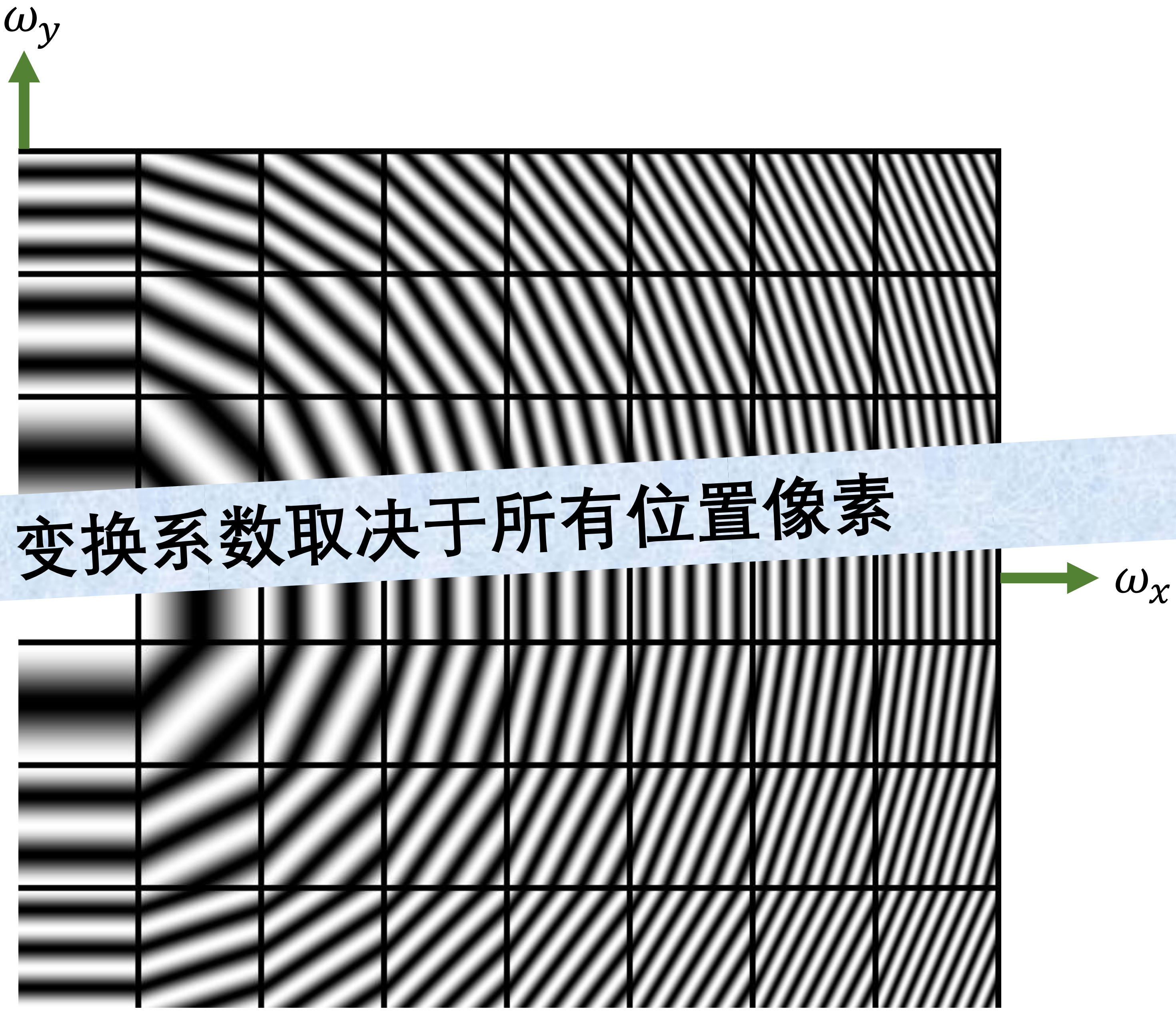

ω_y
ω_x
缓慢变化


ω_y
快速变化
ω_x


ω_y
水平变化
ω_x


ω_y
垂直变化
ω_x
垂直变化


ω_y
ω_x
对角线变化
对角线变化



变换系数取决于所有位置像素

有哪些频率？

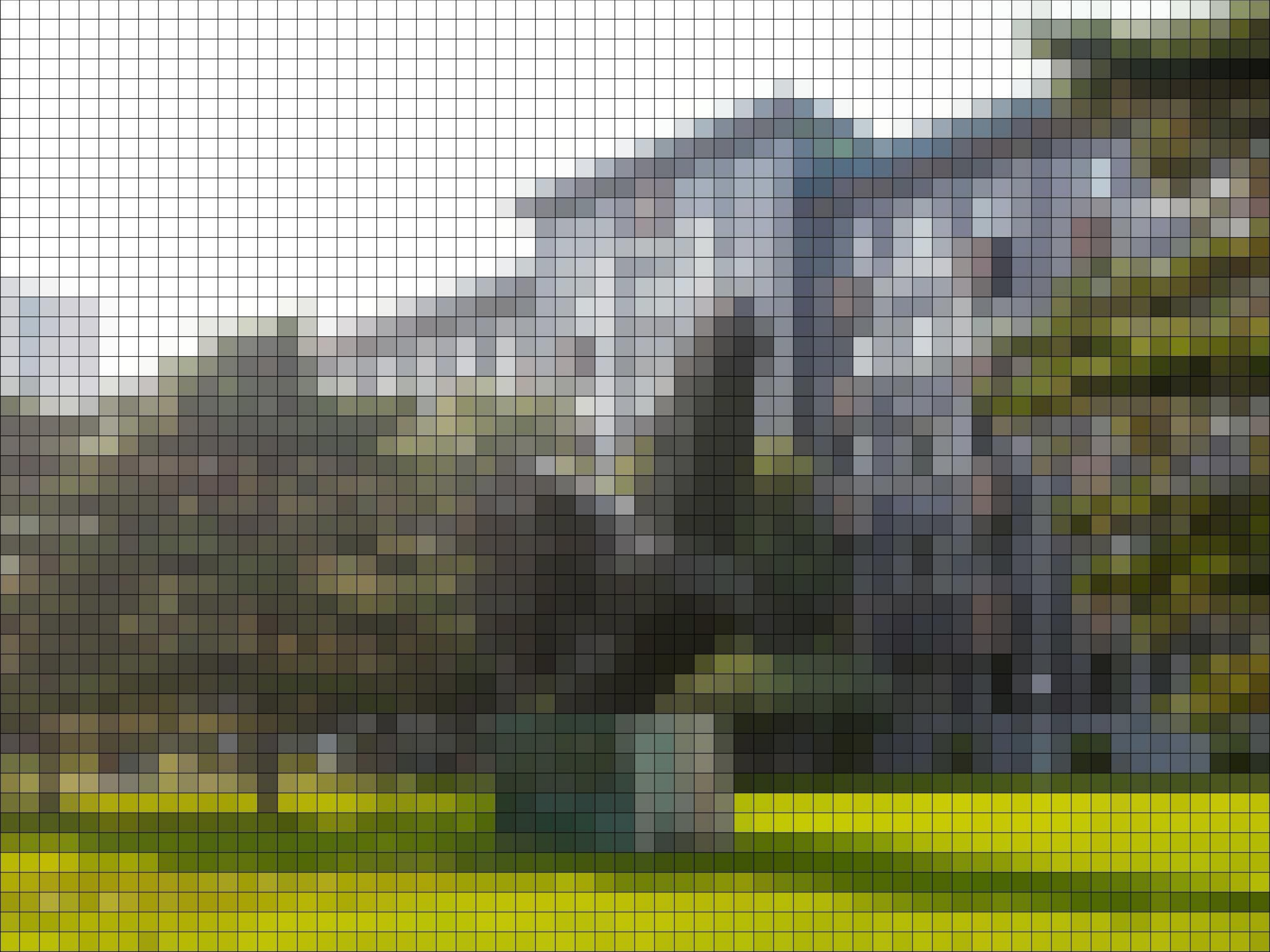
2D傅里叶变换

$$F(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-i2\pi(\omega_x x + \omega_y y)} dx dy$$

2D傅里叶逆变换

$$f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(\omega_x,\omega_y)e^{i2\pi(\omega_x x+\omega_y y)}d\omega_x d\omega_y$$

我们的母校



DFT

Discrete Fourier Transform

离散傅里叶变换

离散傅里叶变换

$$F[u,v] = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f[x,y] e^{-i2\pi\left(x\frac{u}{M}+y\frac{v}{N}\right)}$$

其中

$$u = 0, \dots, M-1$$
$$v = 0, \dots, N-1$$

离散傅里叶逆变换

$$f[x,y] = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F[u,v]e^{i2\pi\left(x\frac{u}{M}+y\frac{v}{N}\right)}$$

其中

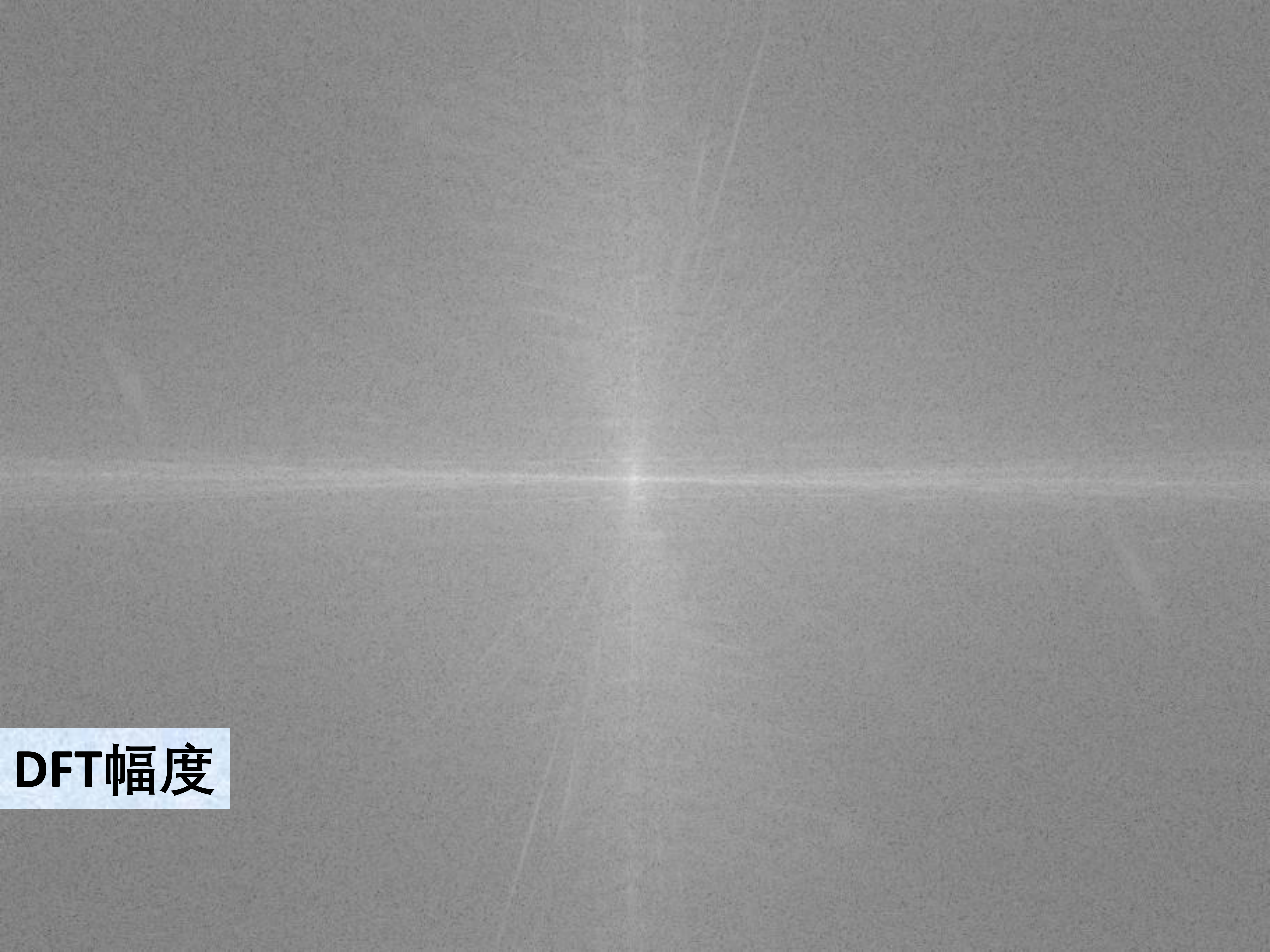
$$u = 0, \dots, M-1$$
$$v = 0, \dots, N-1$$

我们的母校

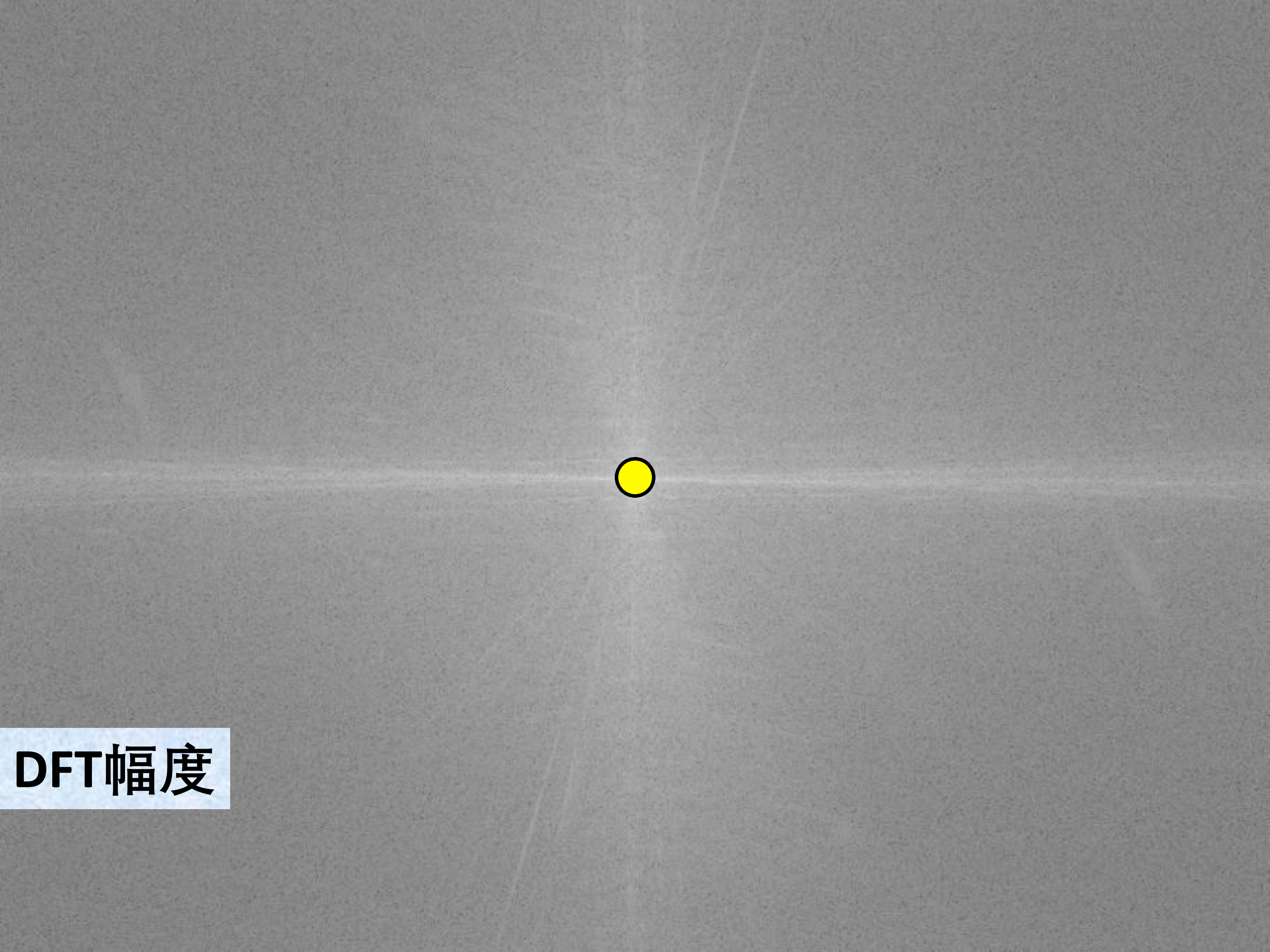
图像信号在两个维度上都是周期性的

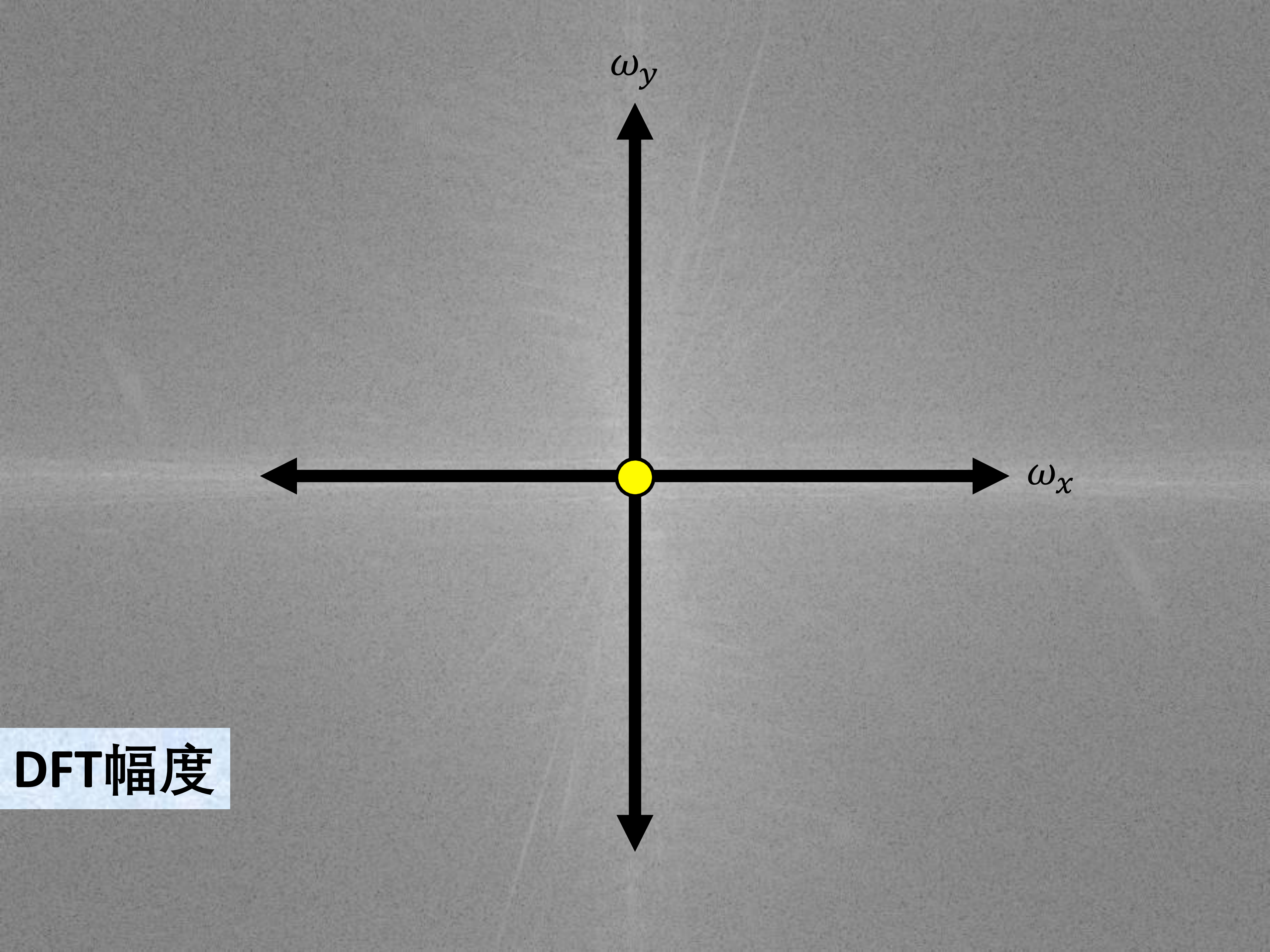
图像信号在两个维度上都是周期性的

输入图像
DBRS
Hilton

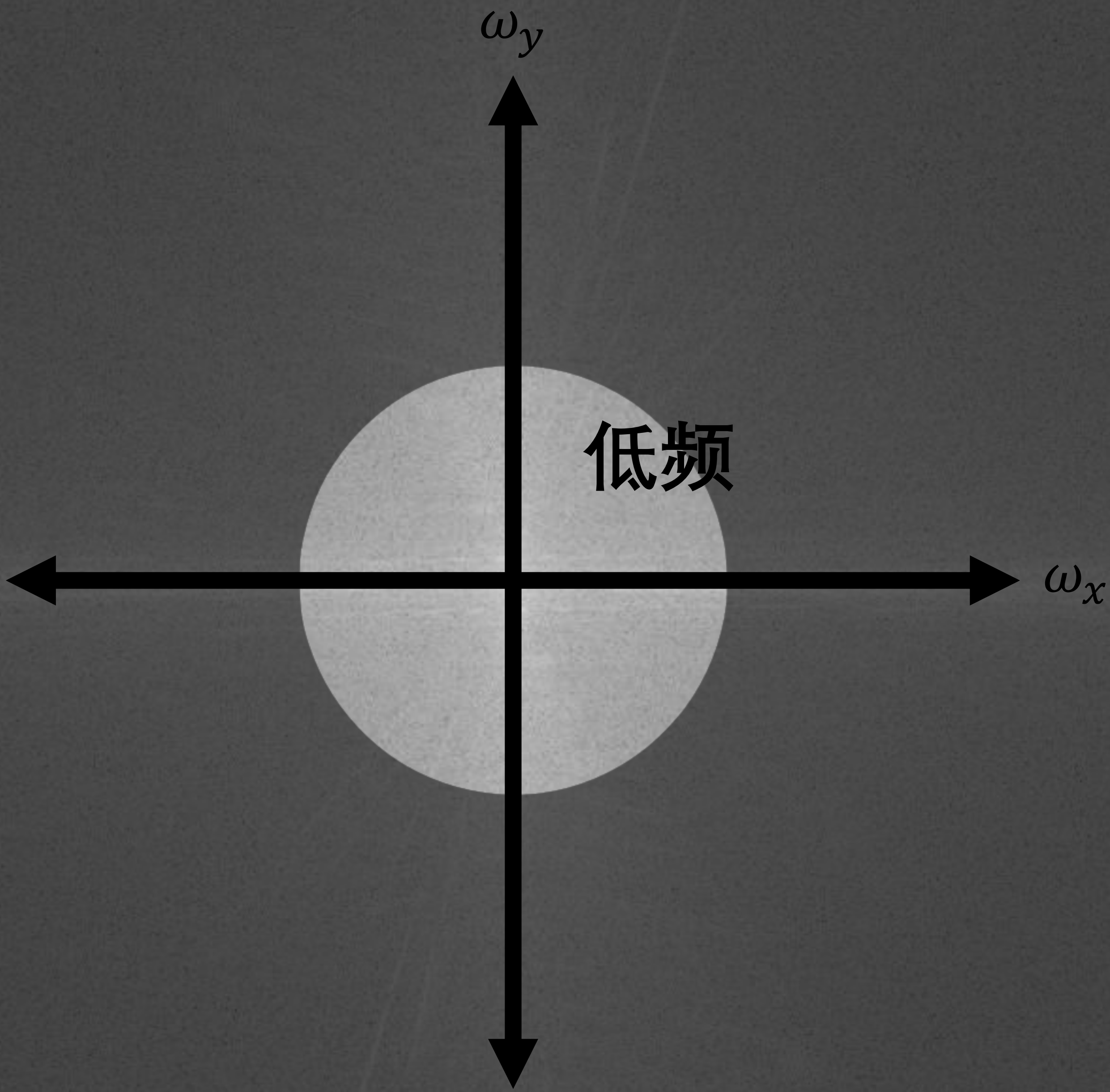


DFT幅度

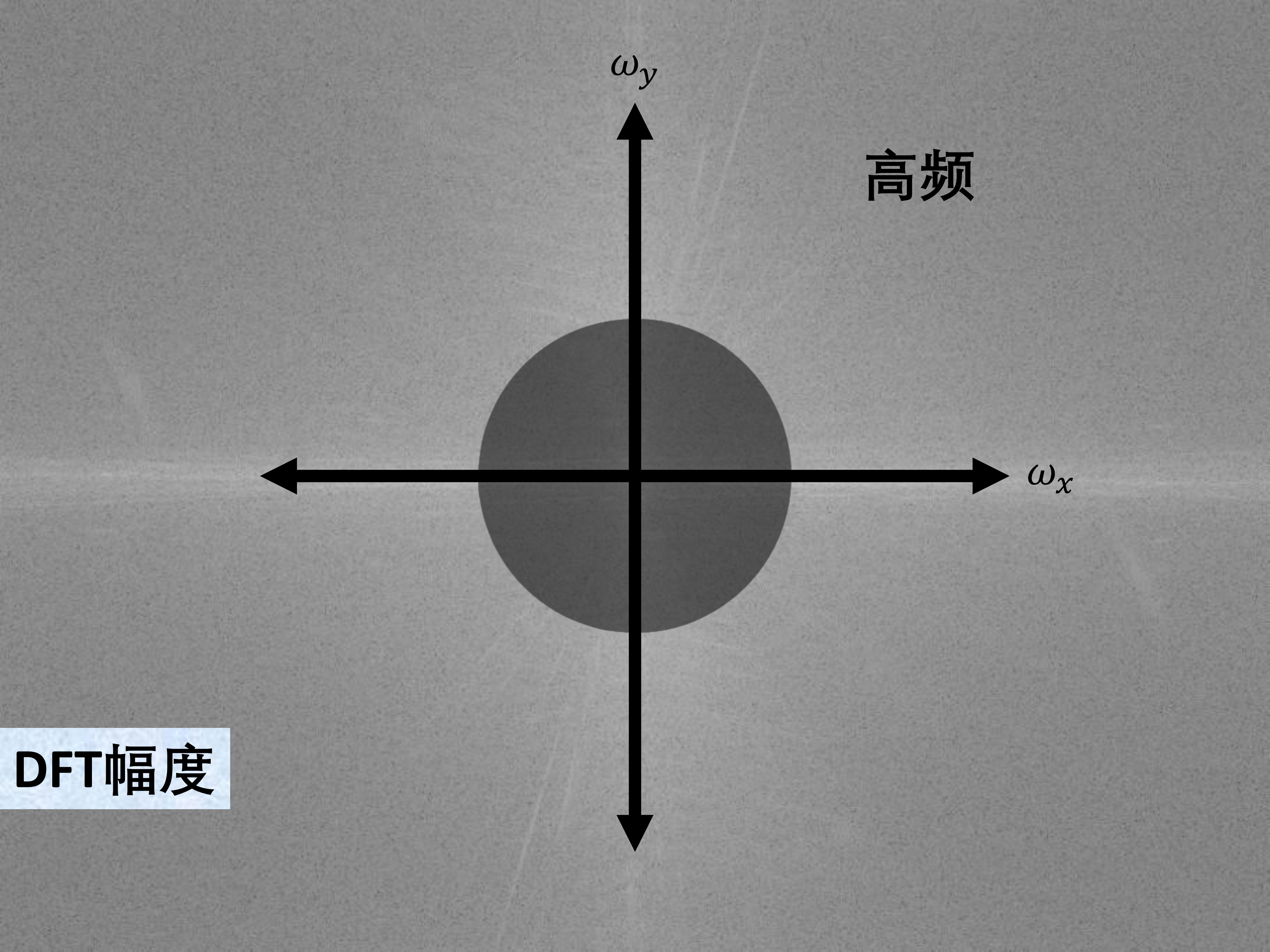

DFT幅度



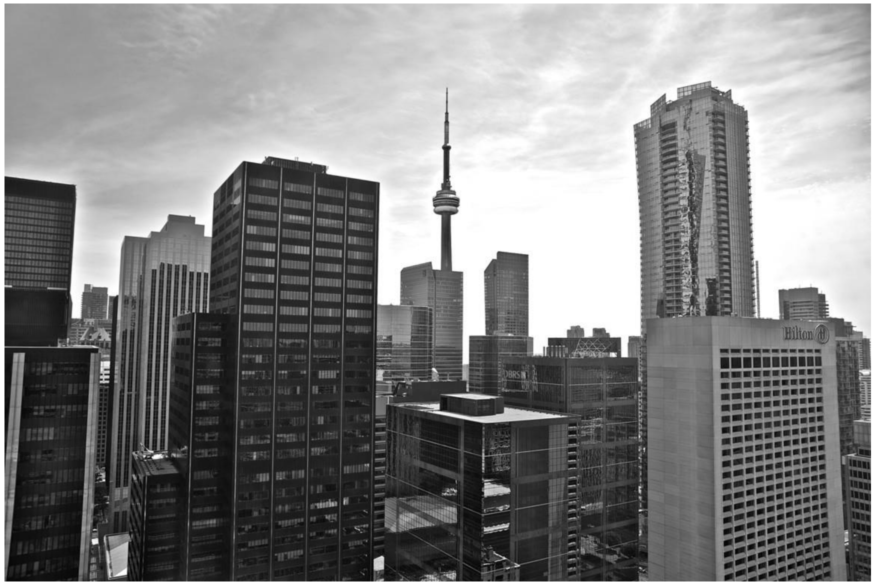
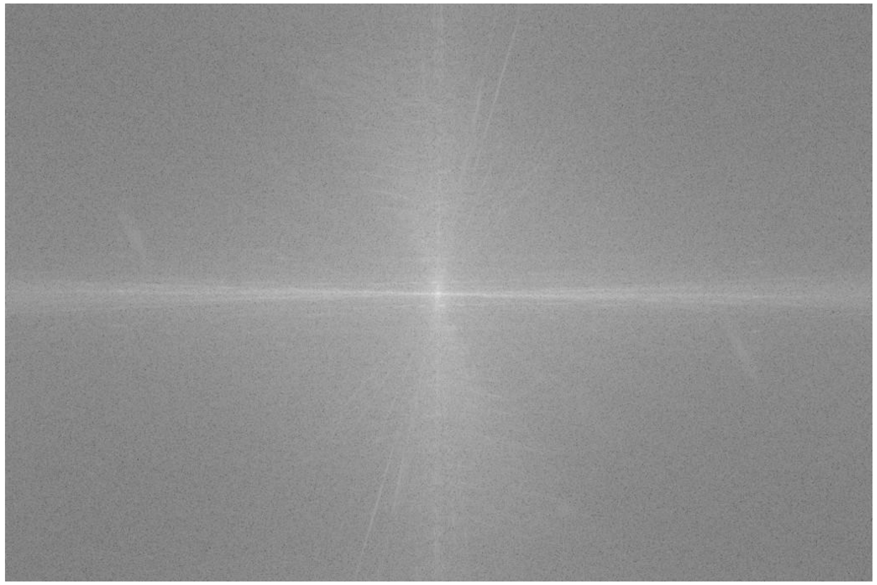
DFT幅度

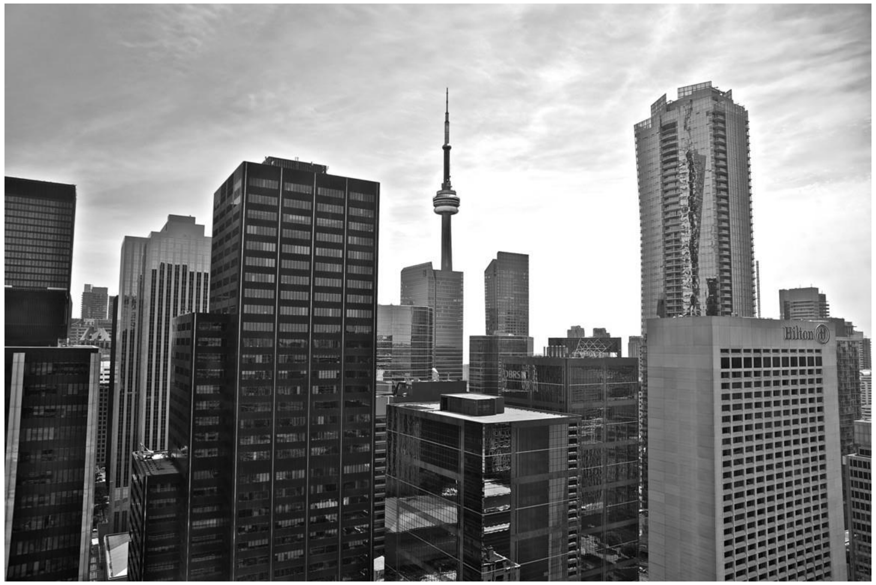


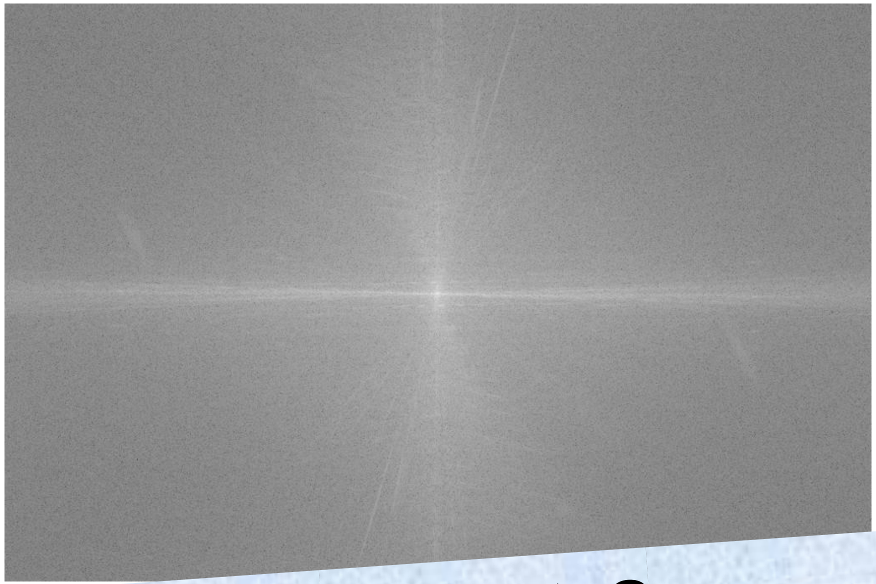
DFT幅度



ω_y
高频
ω_x
DFT幅度

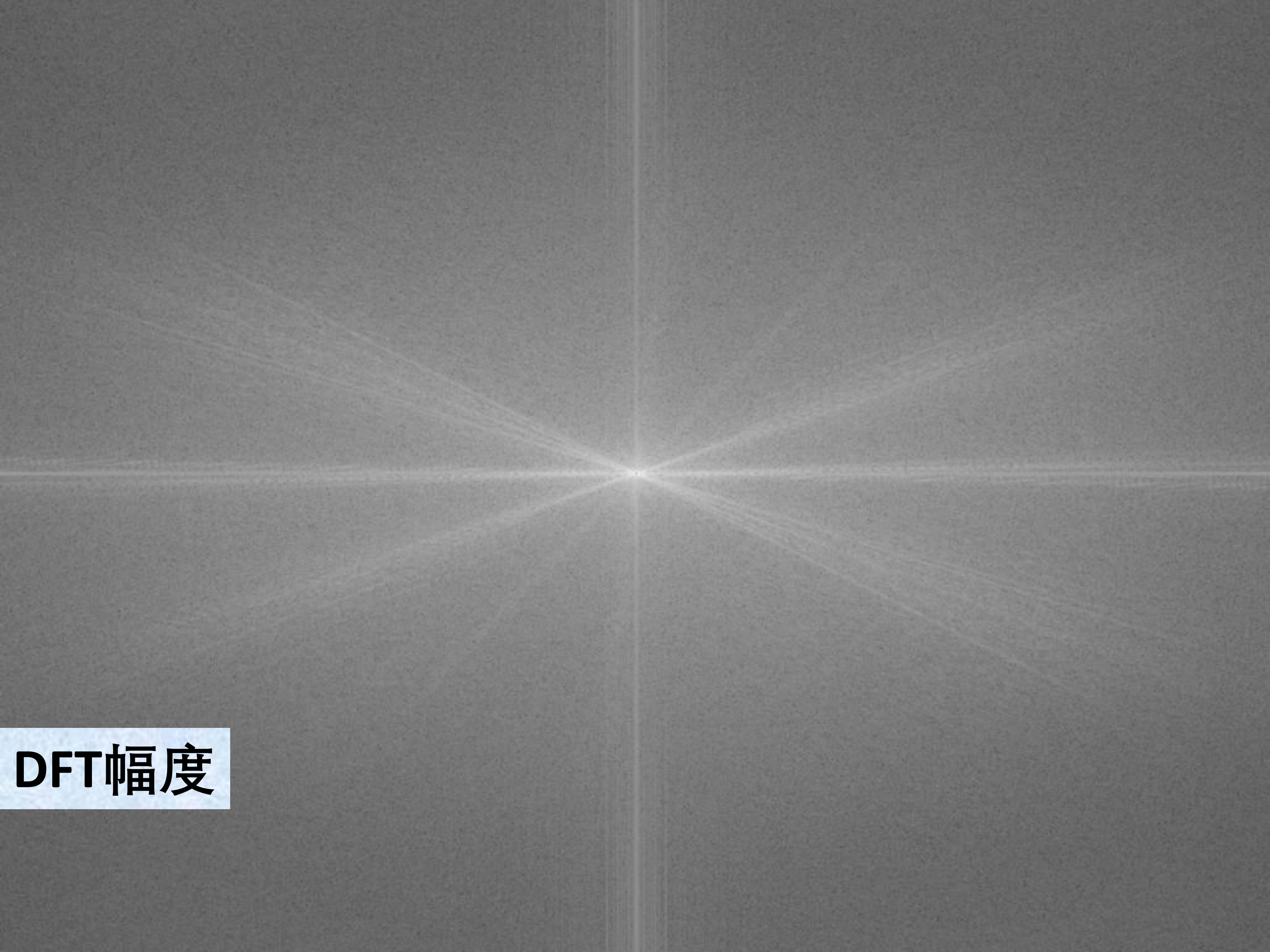

Hilton


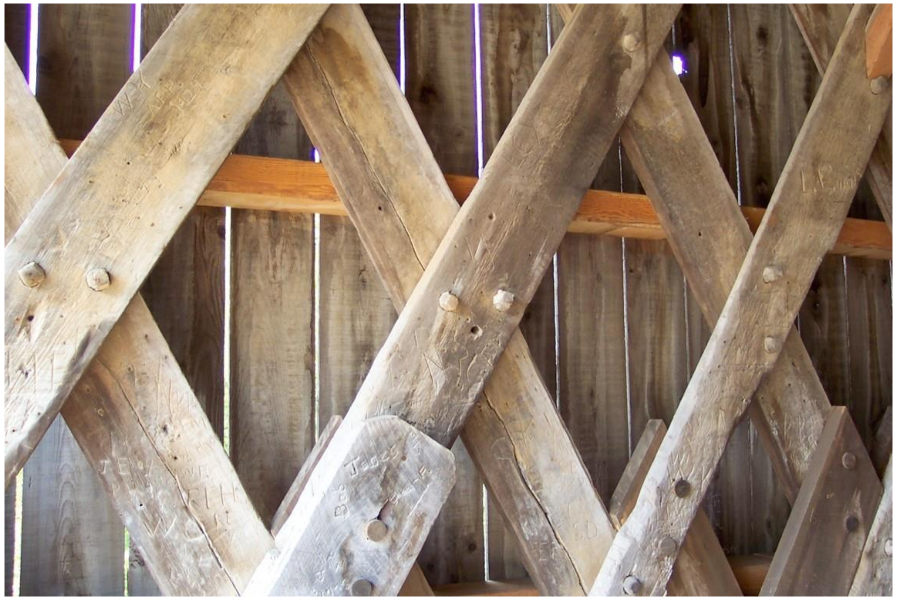
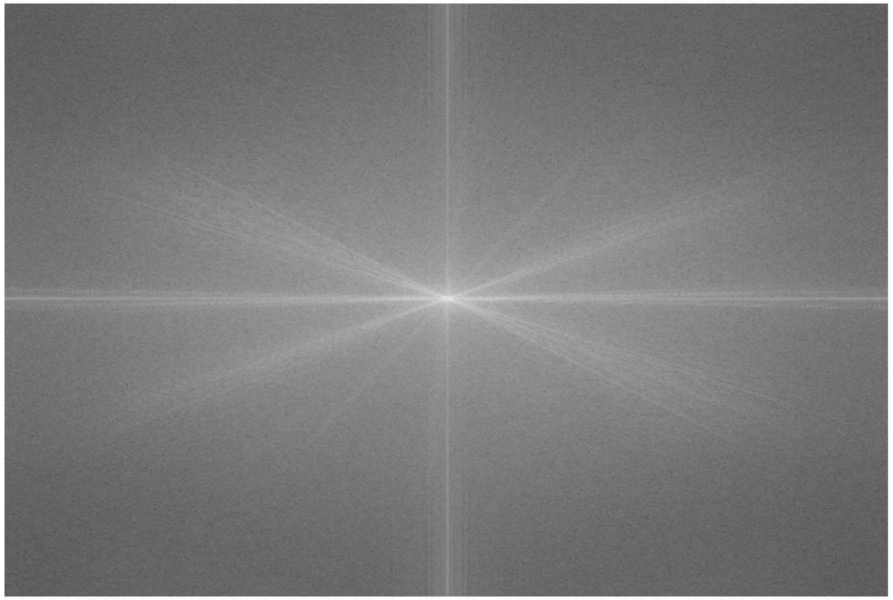


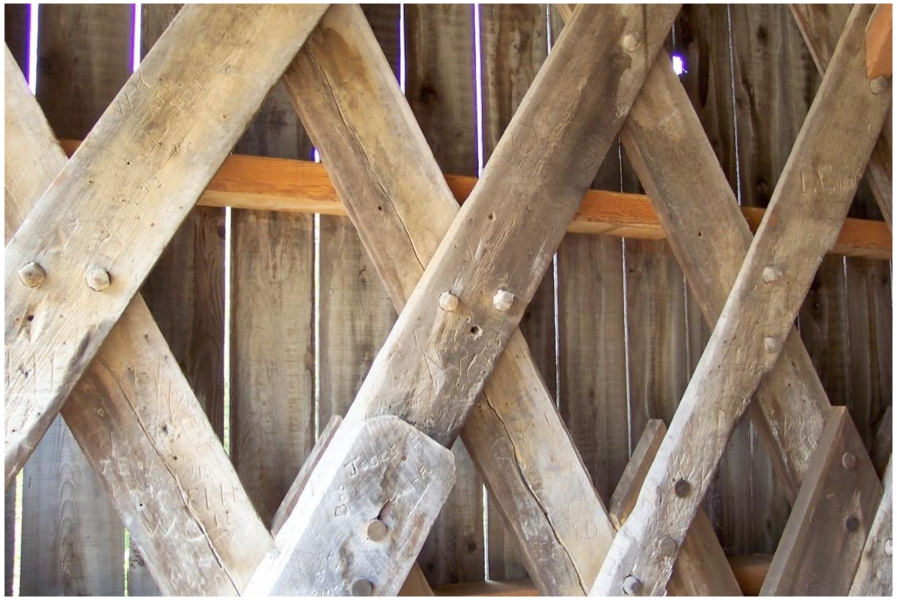


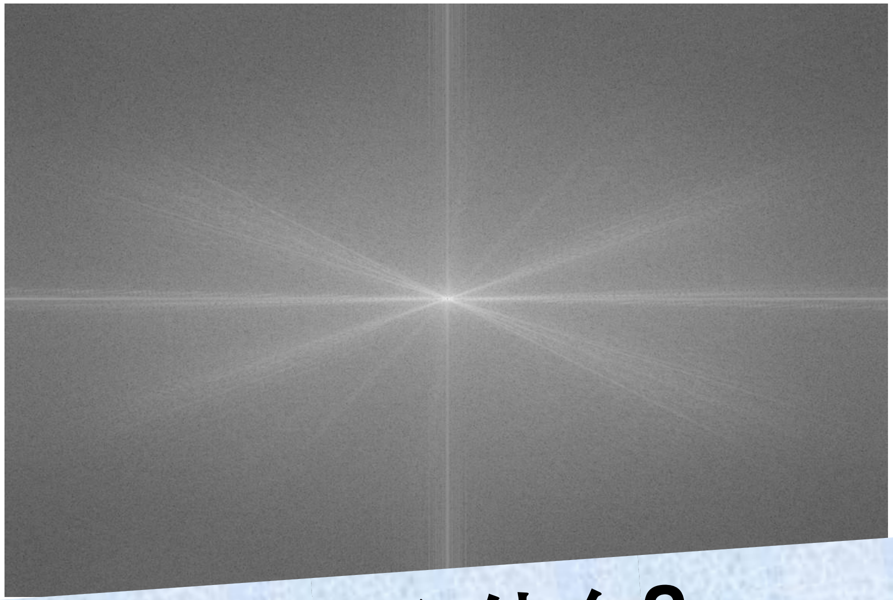
DFT中水平线在输入图像中对应什么?

输入图像

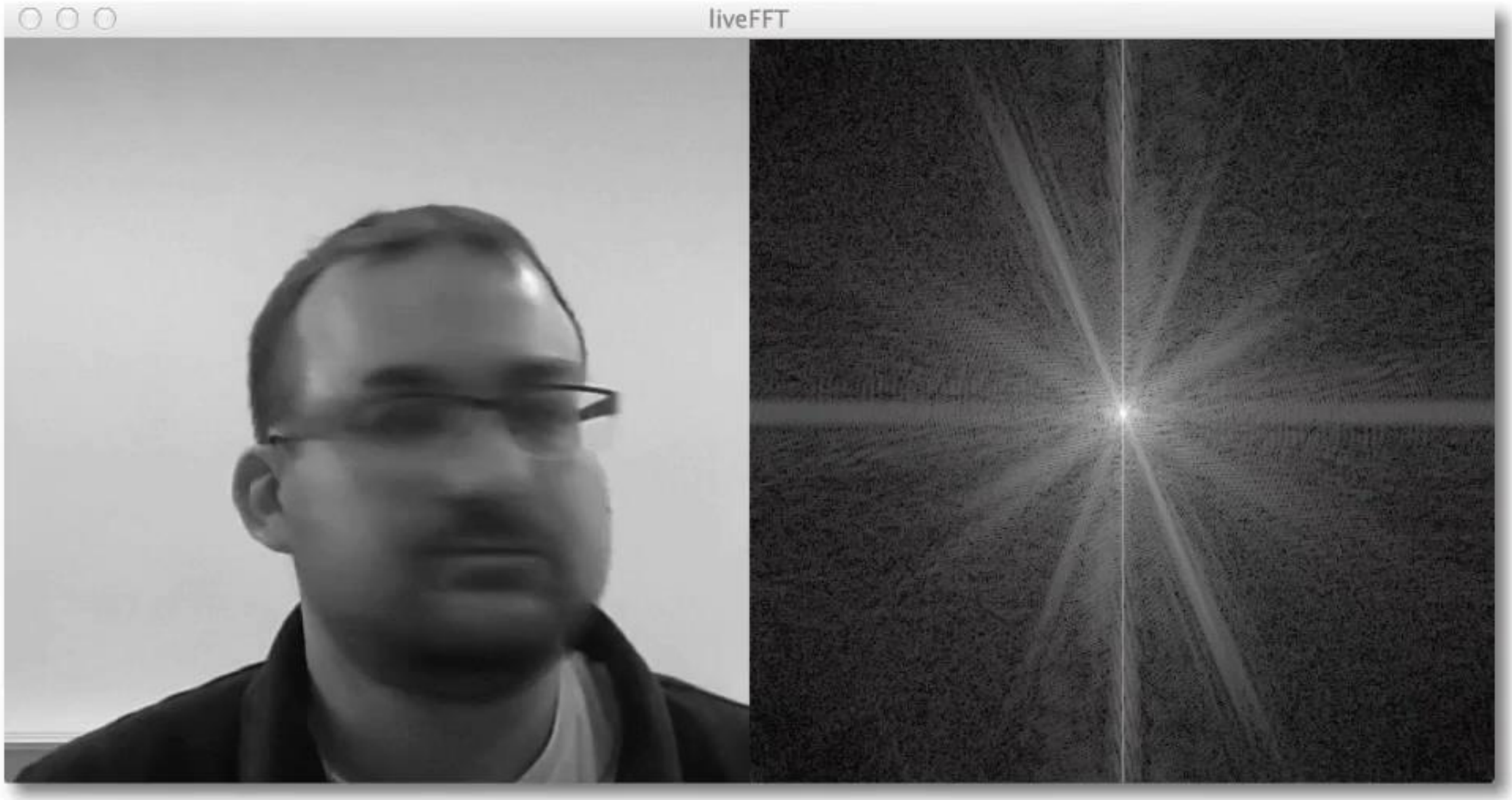

DFT幅度

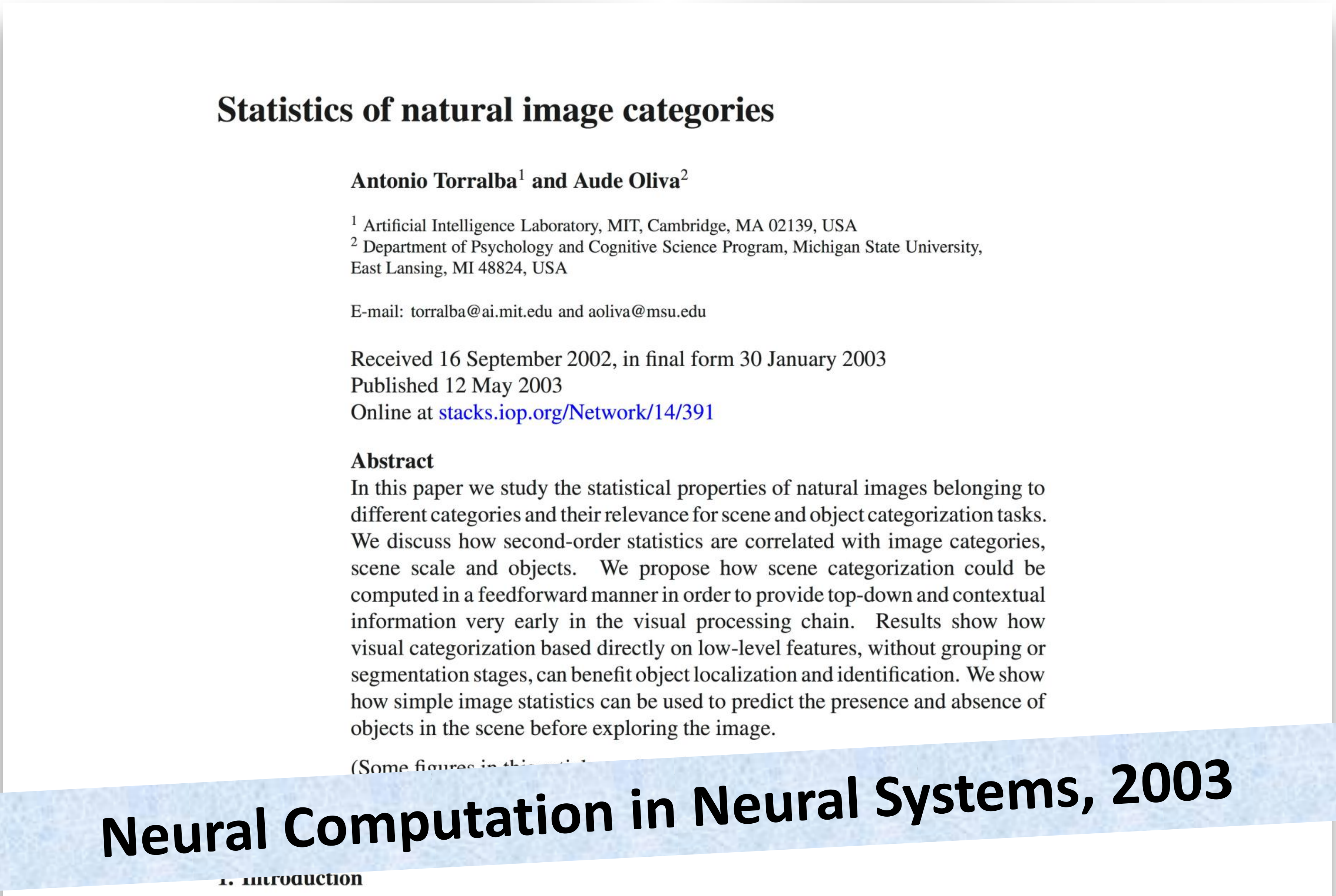







DFT中线条结构在输入图像中对应什么?


liveFFT



Statistics of natural image categories

Antonio Torralba[1] **and Aude Oliva**[2]

[1] Artificial Intelligence Laboratory, MIT, Cambridge, MA 02139, USA
[2] Department of Psychology and Cognitive Science Program, Michigan State University, East Lansing, MI 48824, USA

E-mail: torralba@ai.mit.edu and aoliva@msu.edu

Received 16 September 2002, in final form 30 January 2003
Published 12 May 2003
Online at stacks.iop.org/Network/14/391

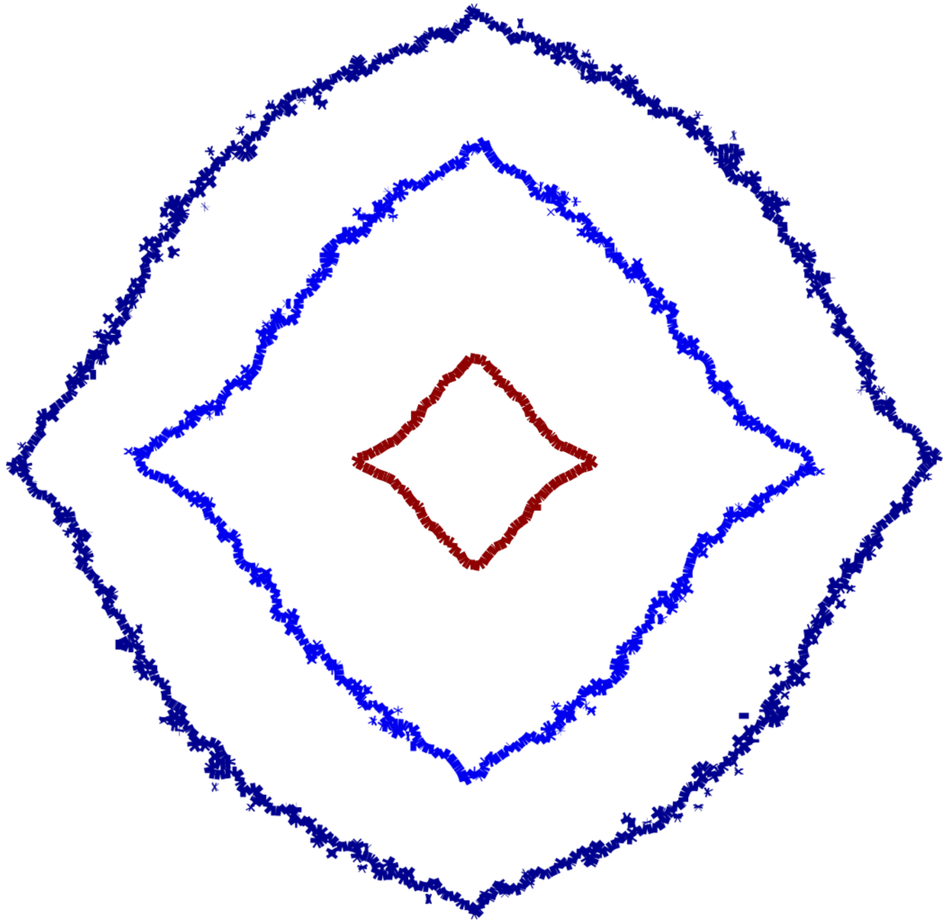
Abstract
In this paper we study the statistical properties of natural images belonging to different categories and their relevance for scene and object categorization tasks. We discuss how second-order statistics are correlated with image categories, scene scale and objects. We propose how scene categorization could be computed in a feedforward manner in order to provide top-down and contextual information very early in the visual processing chain. Results show how visual categorization based directly on low-level features, without grouping or segmentation stages, can benefit object localization and identification. We show how simple image statistics can be used to predict the presence and absence of objects in the scene before exploring the image.

(Some figures in thi

Neural Computation in Neural Systems, 2003

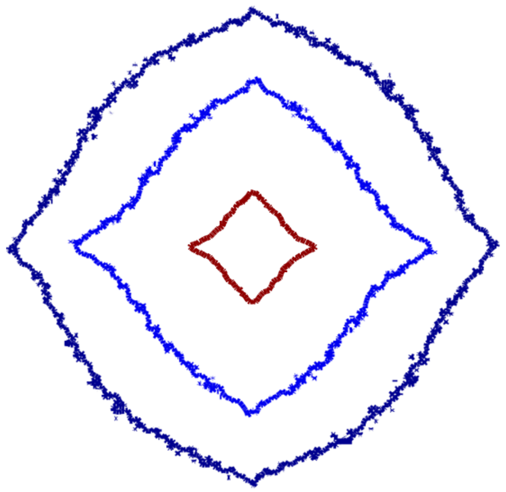
1. Introduction

平均
幅度谱

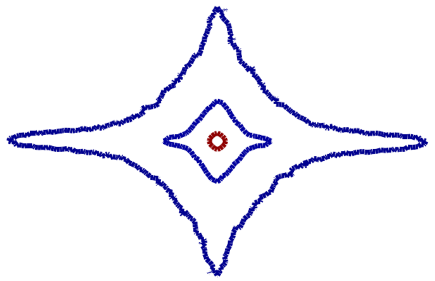


森林

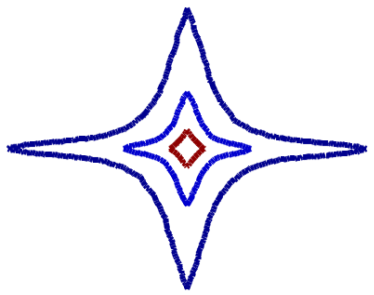
平均
幅度谱



森林



街道

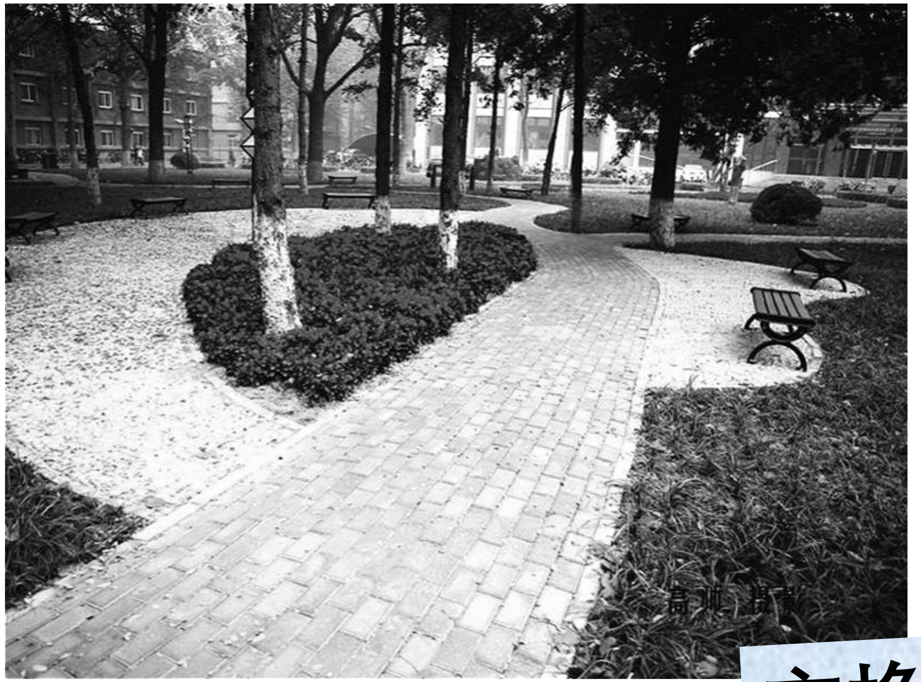


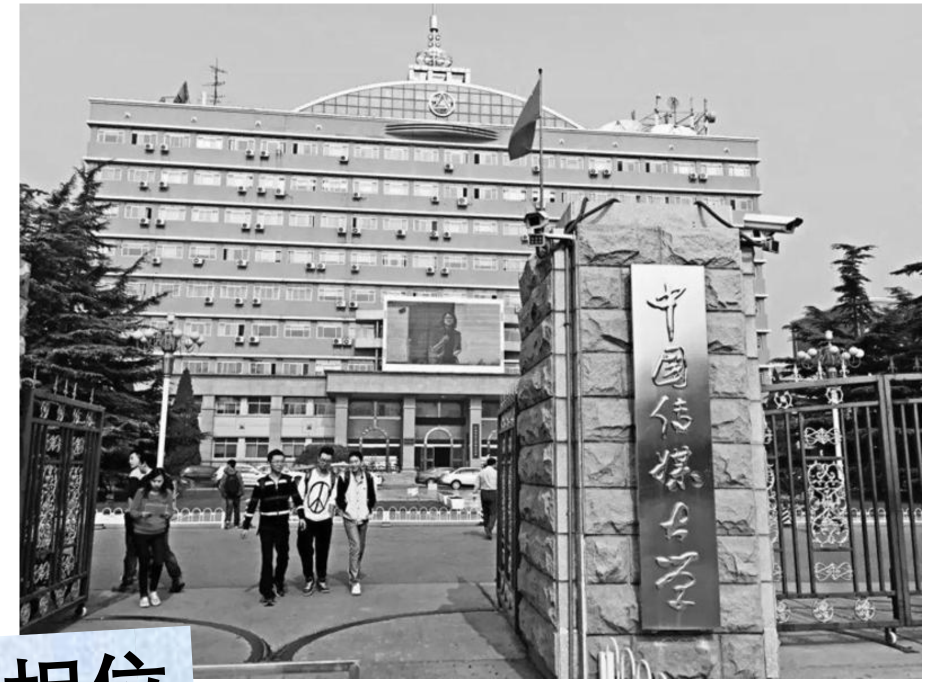
室内

高帅 摄影

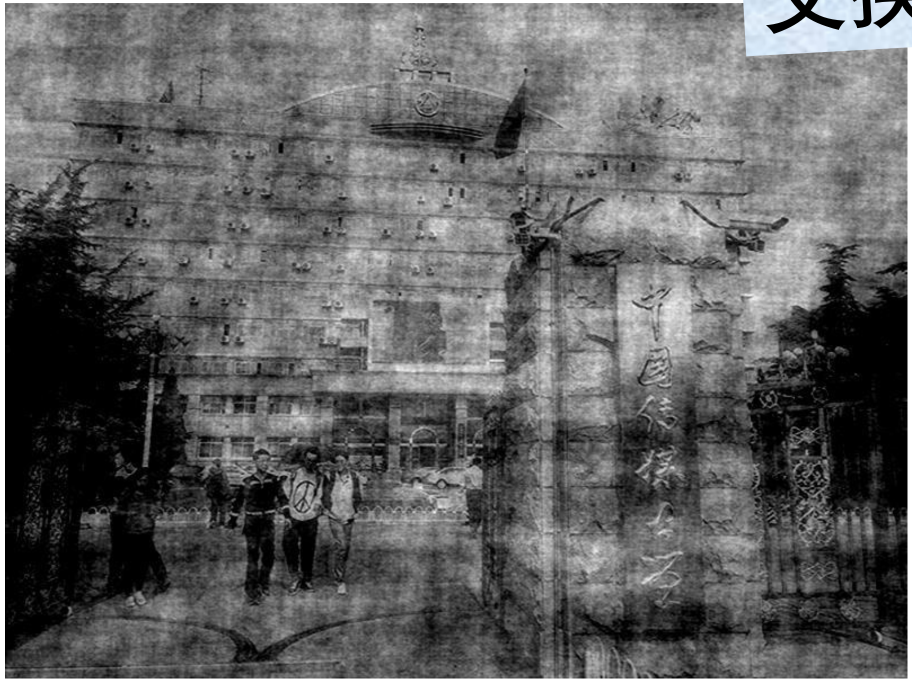
中国传媒大学

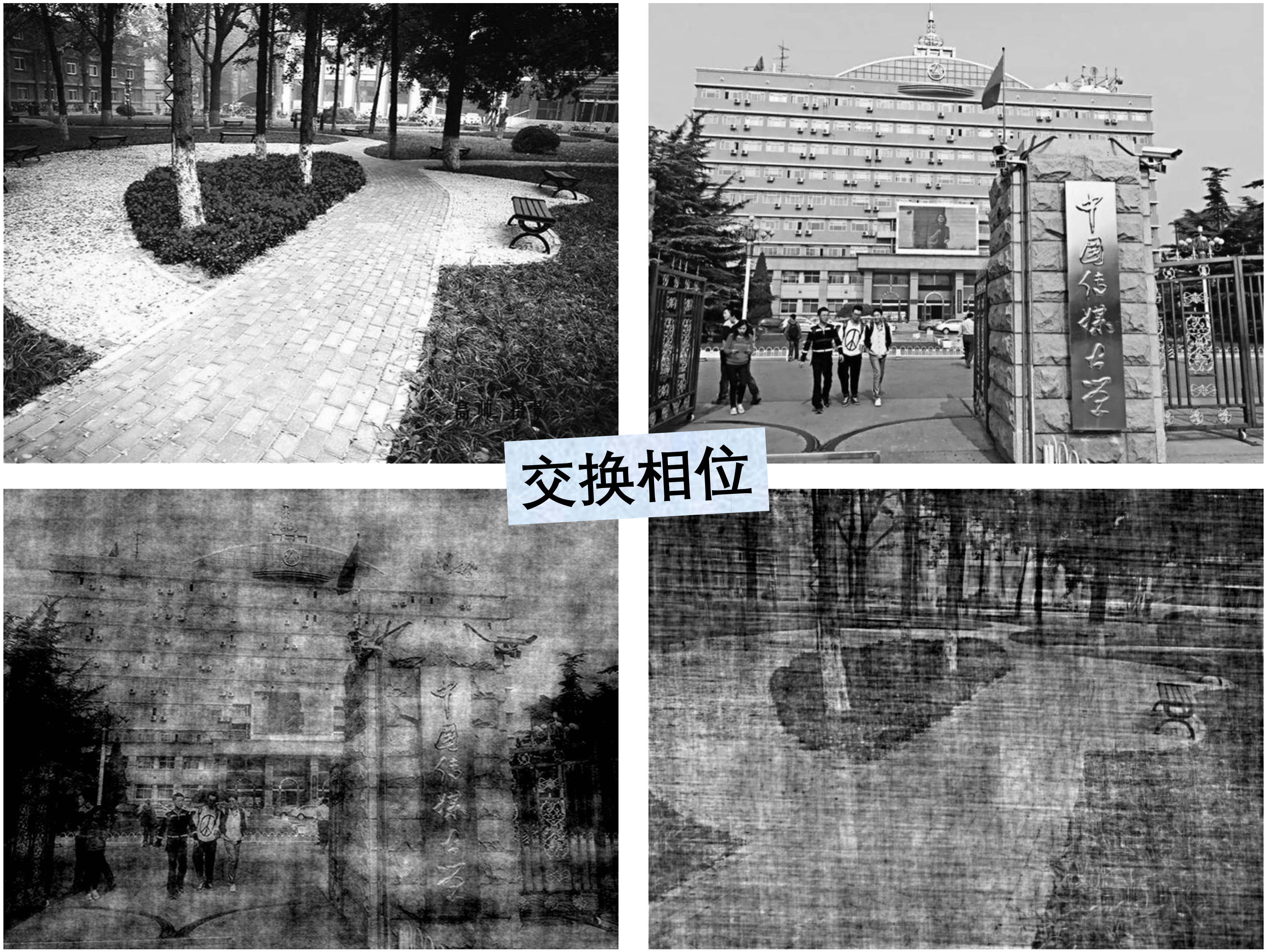
交换相位



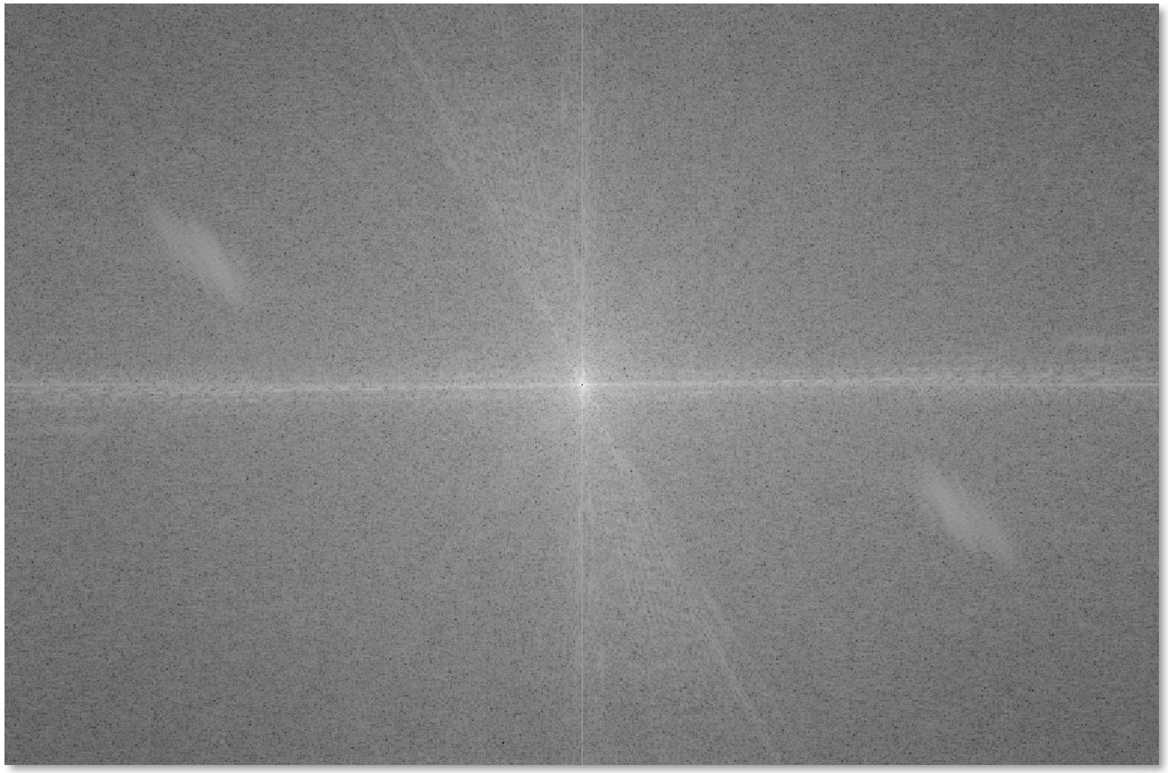


交换相位

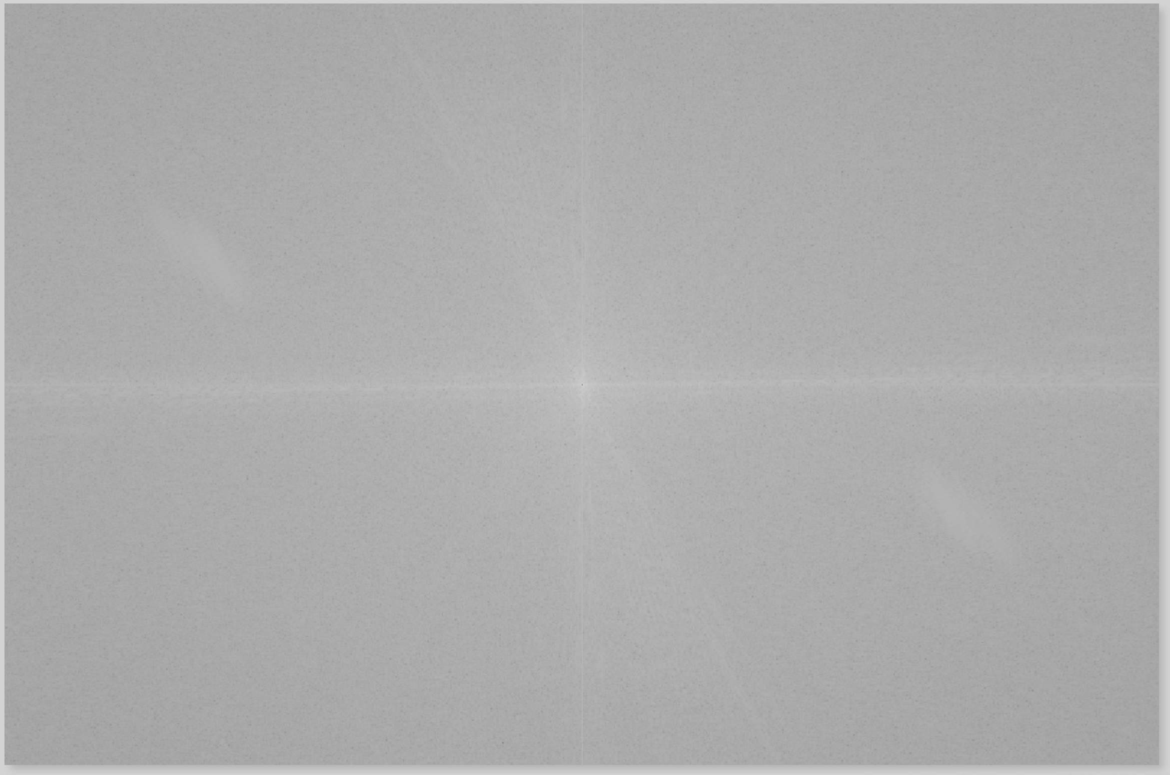



交换相位

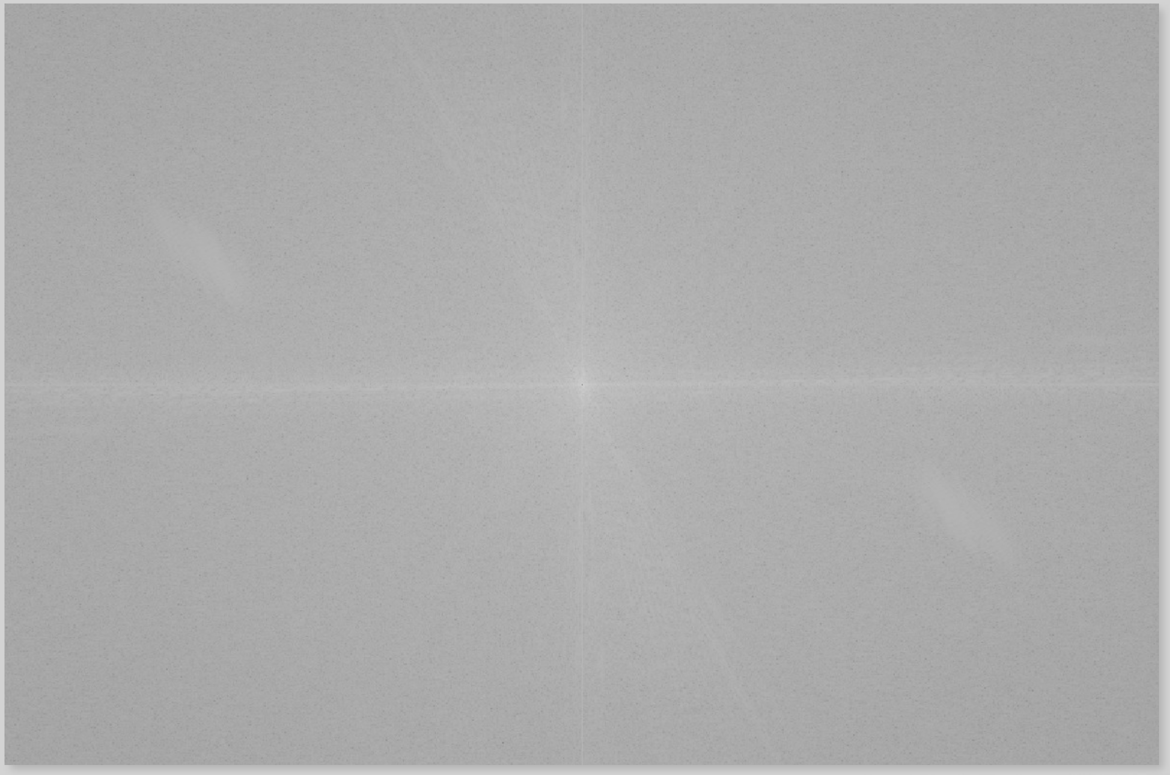
Python时间



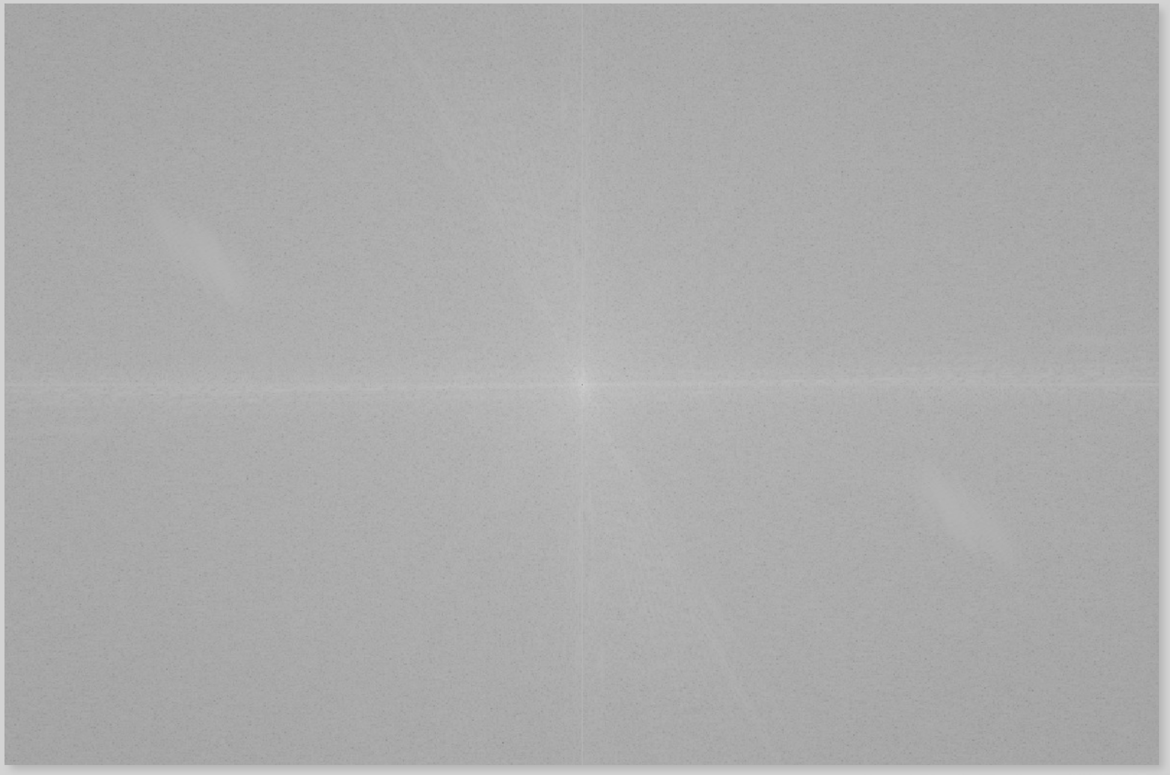
```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                          cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```



```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                            cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```

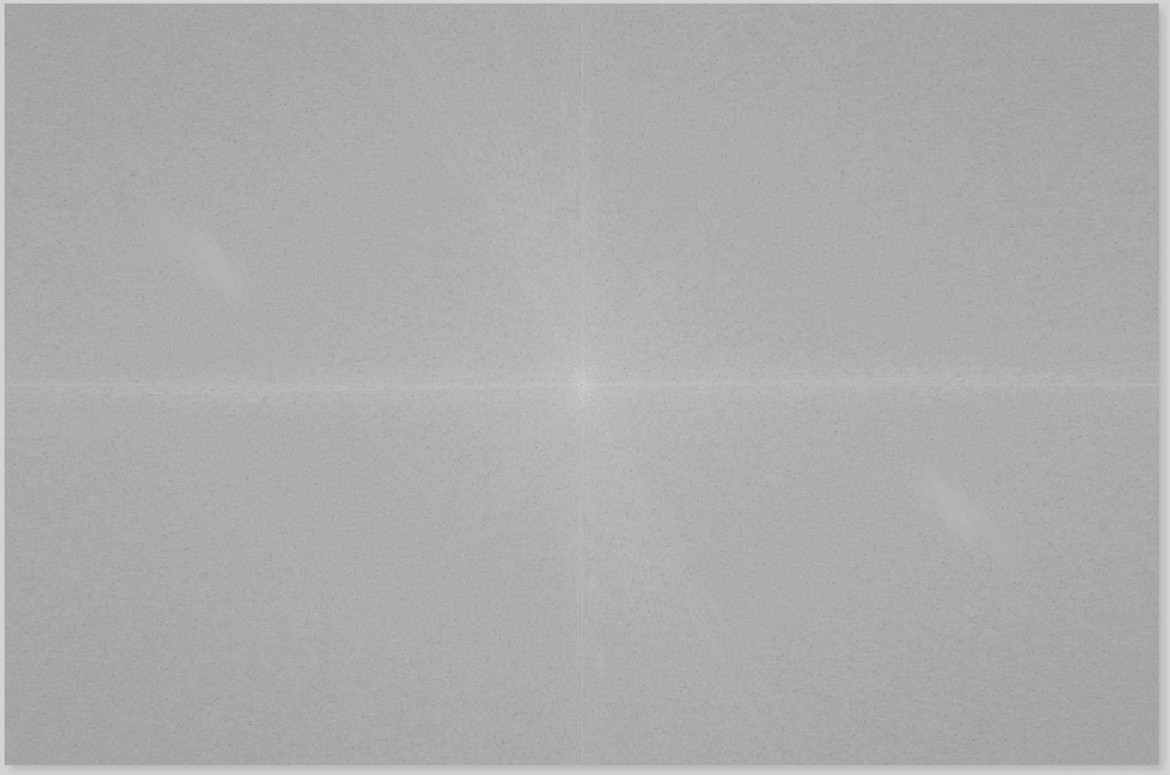


```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                                   cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```

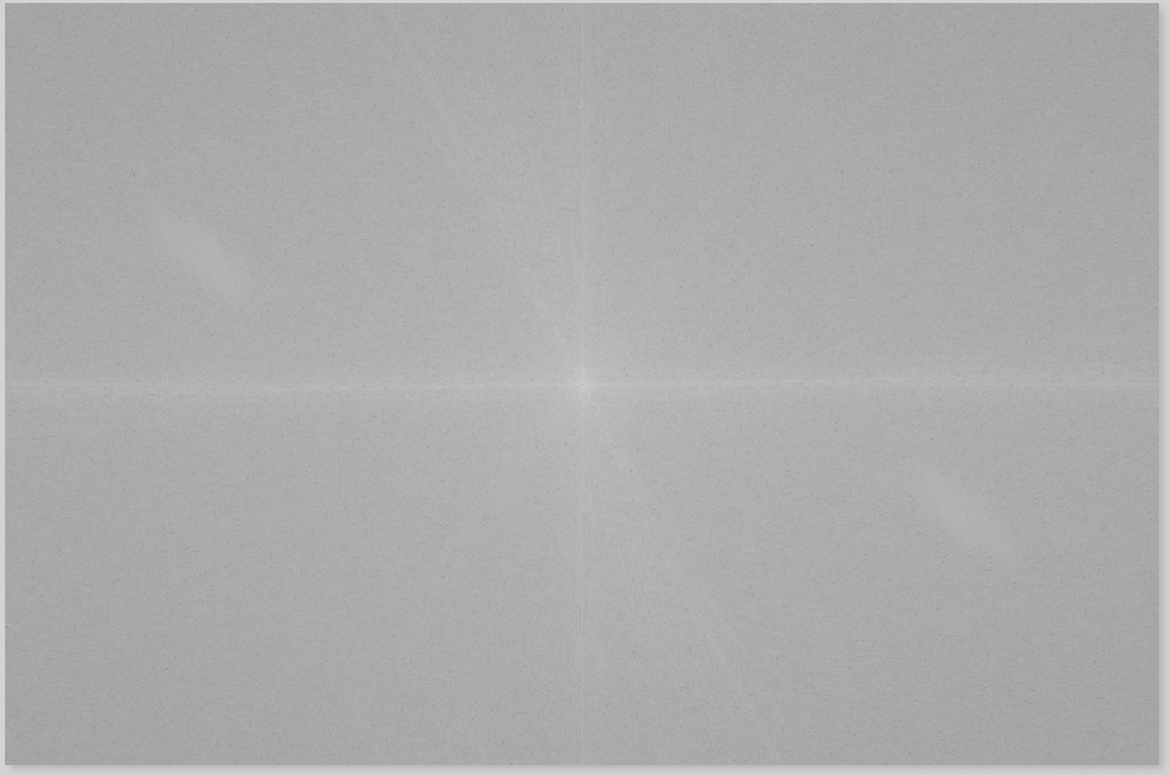


```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
[illegible]
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                           cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```

2D离散傅里叶变换

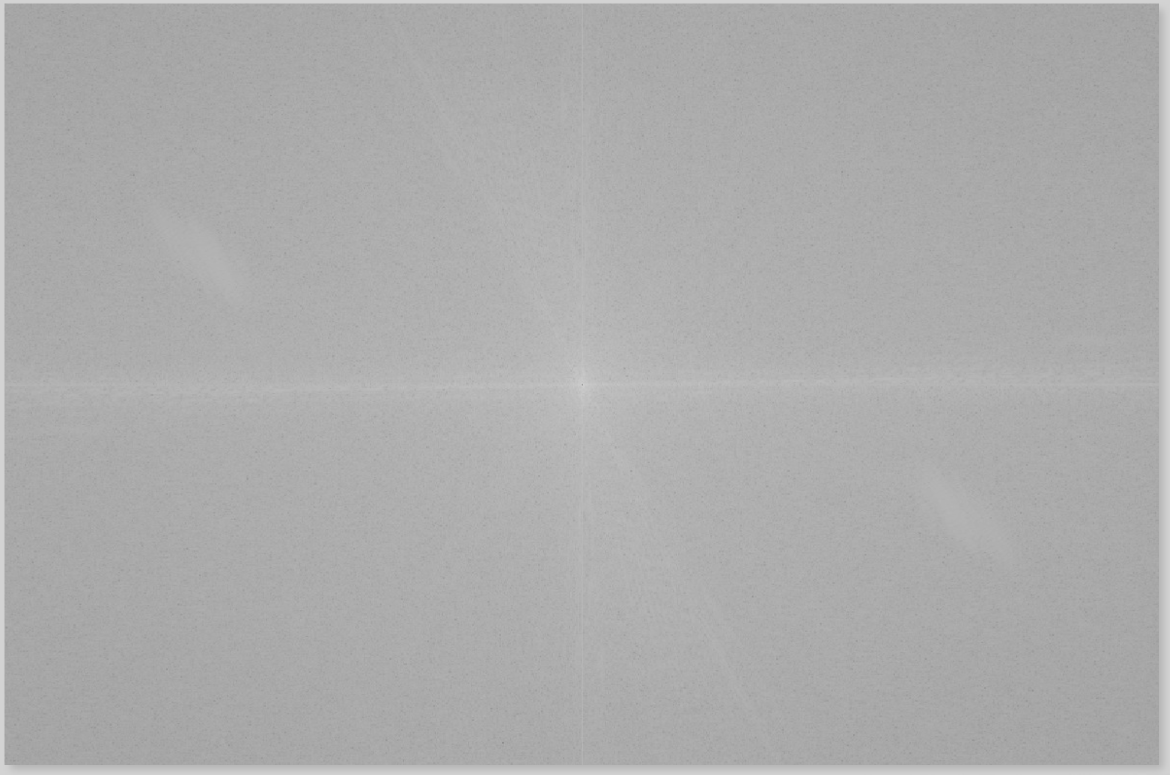


```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                            cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```

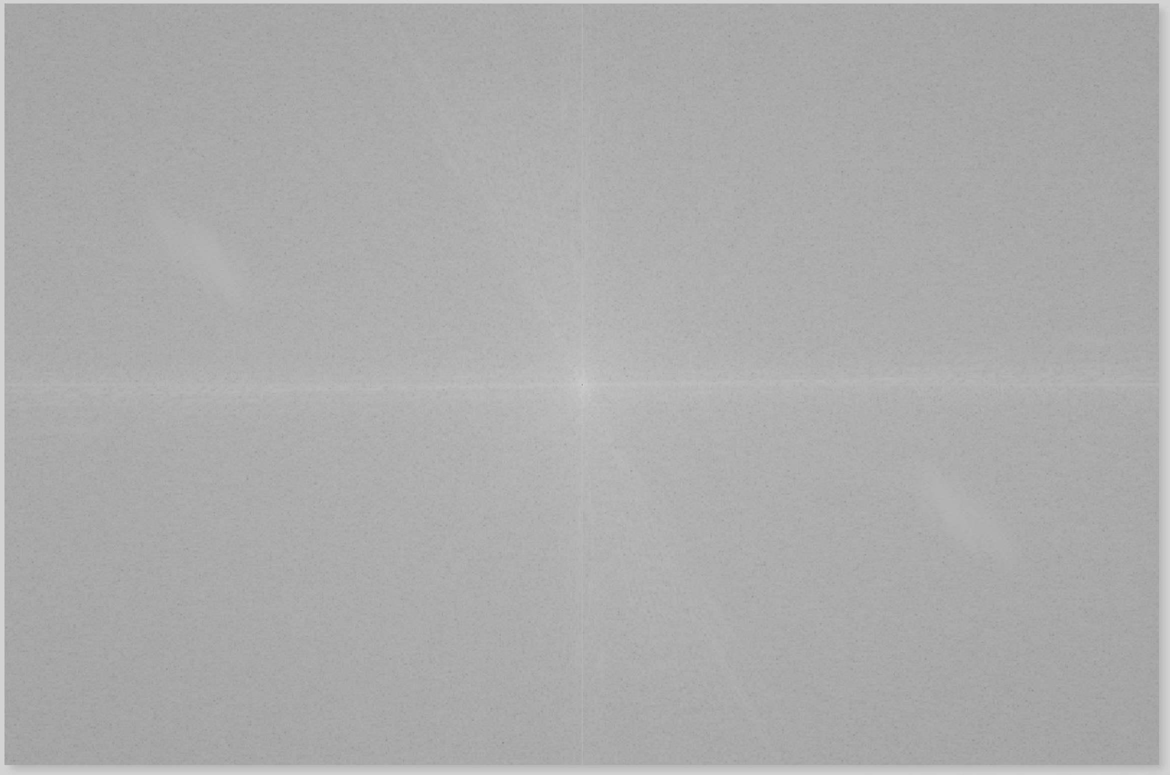


```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_ff
            e(im_fft, None, 0, 255,
                                   cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```

移除直流分量以提高可视化效果

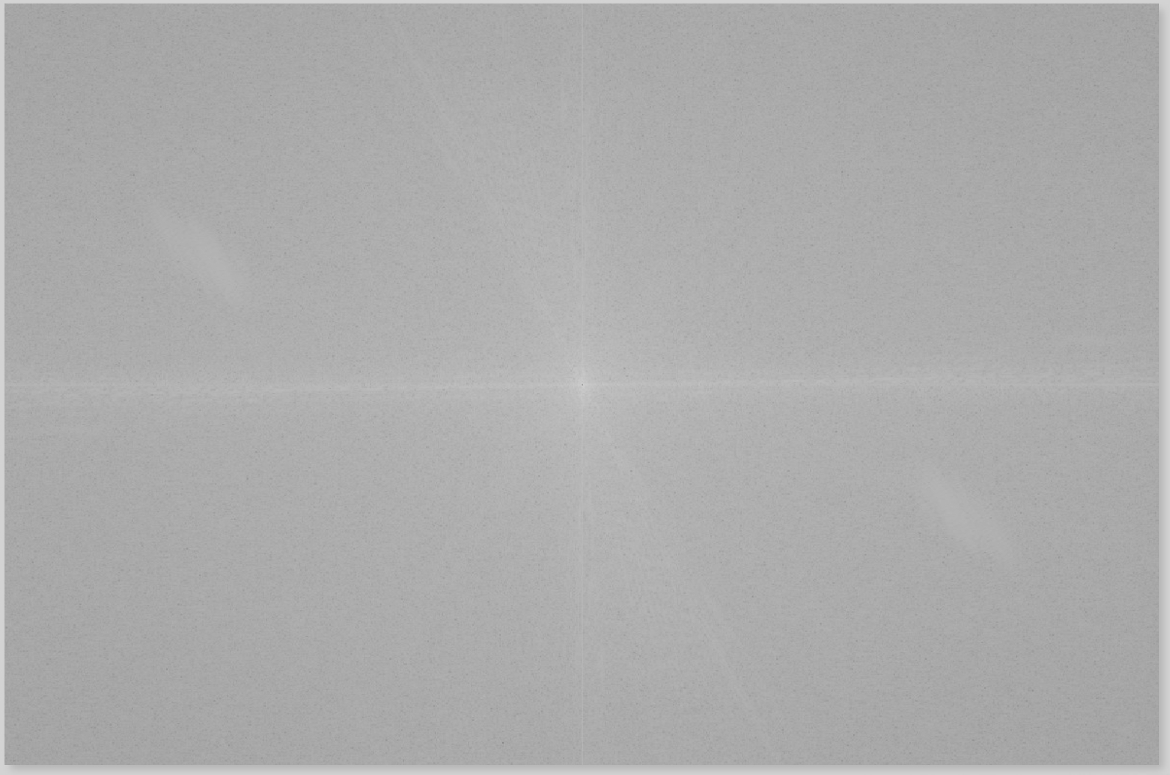


```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                           cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```



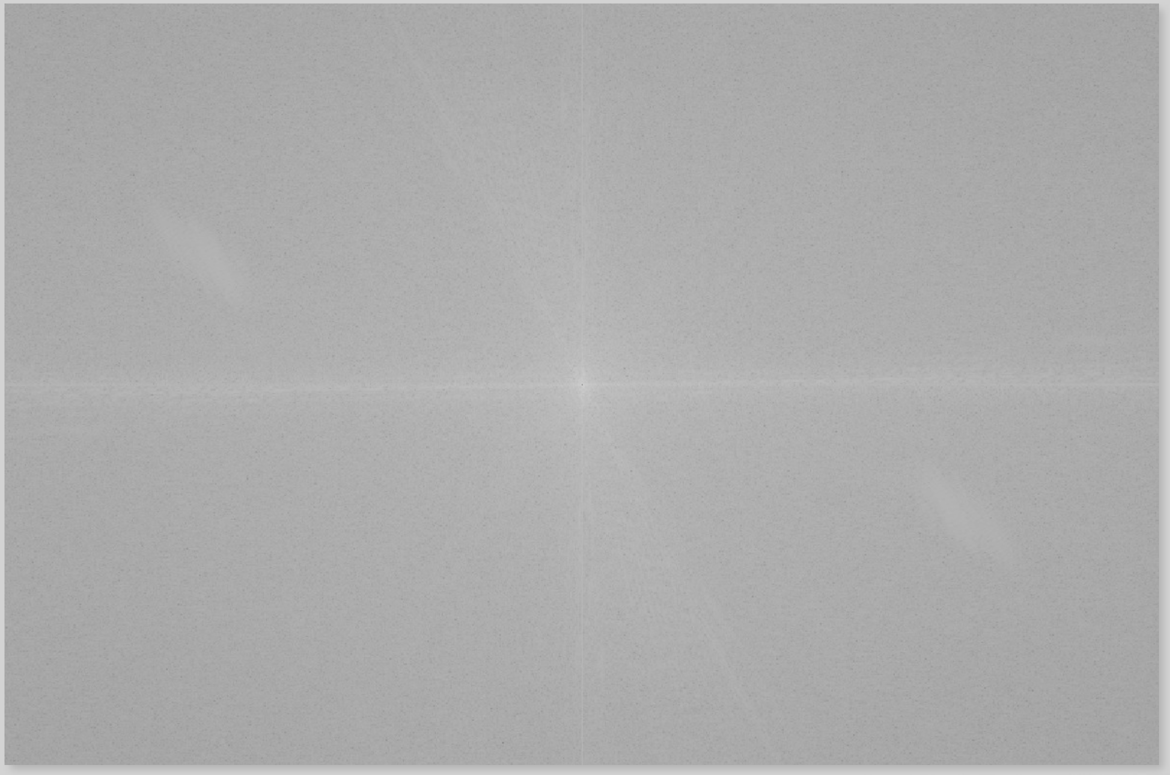
```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astyp
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                           cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```

幅度谱

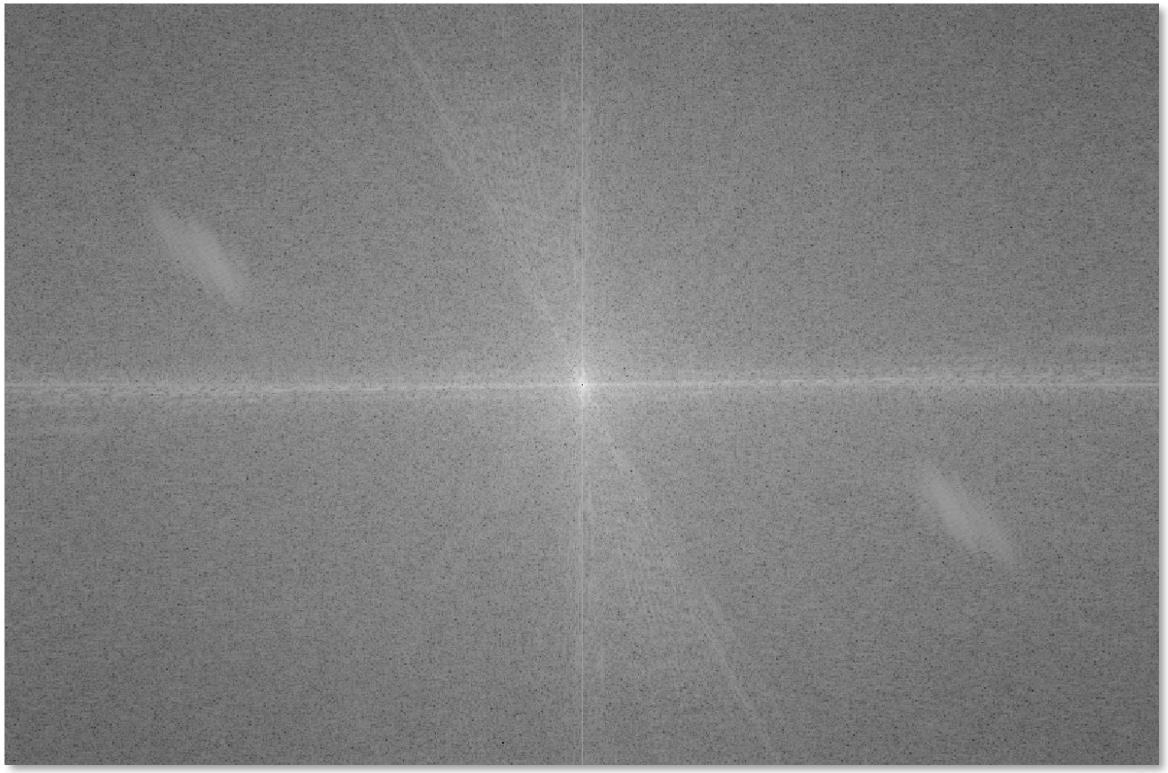


```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                          _MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```

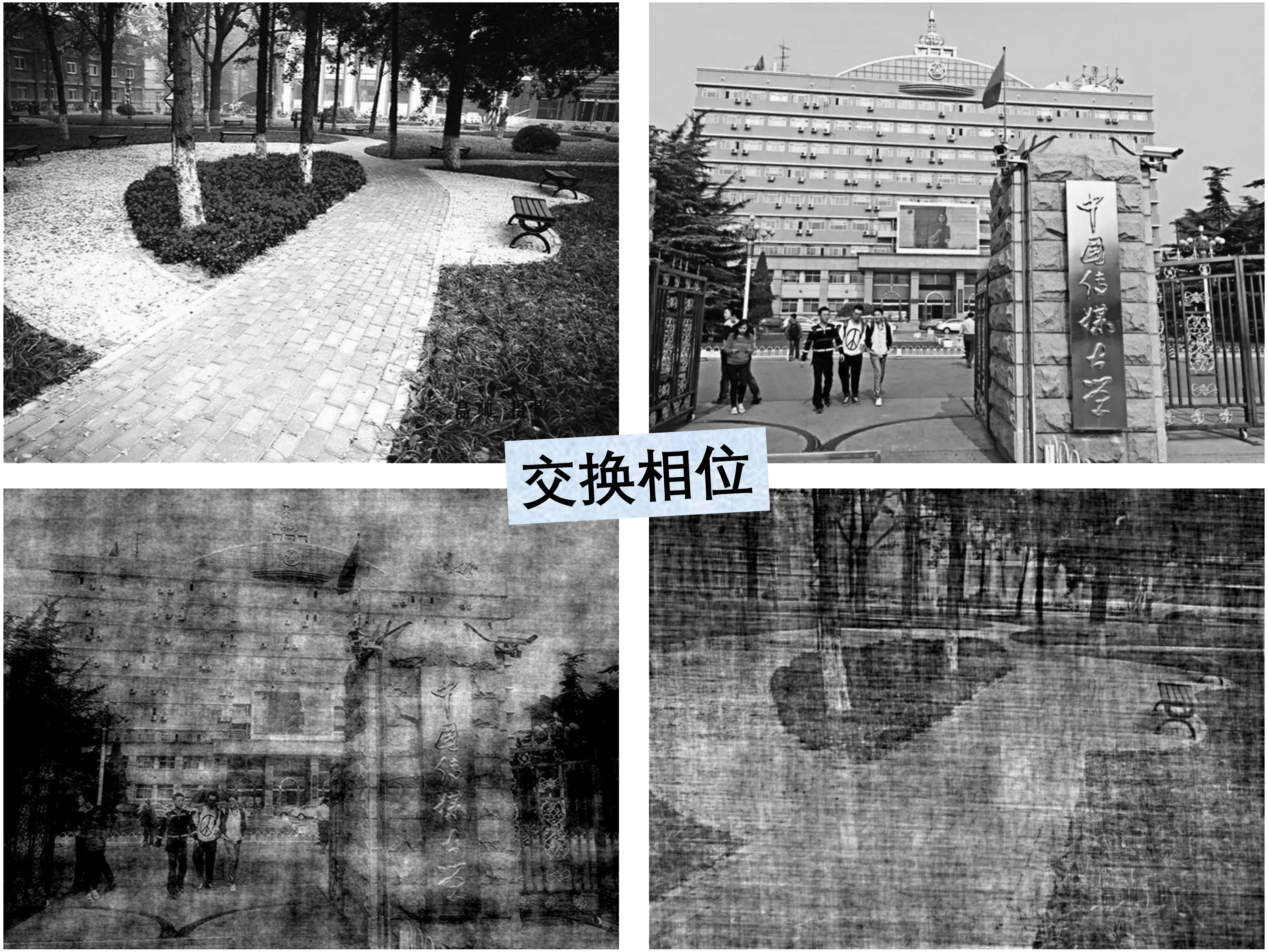
提高可视化效果



```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                                    cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```



```
>>> im = cv2.imread('lena.png', cv2.IMREAD_GRAYSCALE)
>>> im_fft = np.fft.fft2(im.astype(float))
>>> im_fft[0, 0] = 0
>>> im_fft = np.log(1 + np.abs(im_fft))
>>> im_fft = cv2.normalize(im_fft, None, 0, 255,
                                    cv2.NORM_MINMAX, dtype=cv2.CV_8U)
>>> cv2.imshow('fft', np.fft.fftshift(im_fft)), cv2.waitKey(0)
```


高帅 摄影
中国传媒大学
交换相位

```
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.ex                    (I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(       _phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
```

虚数单位

```
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
[illegible](I2_fft)*np.exp(1j*np.angle(I1_fft))
[illegible]
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
```

将一幅图像的幅度与另一幅图像的相位相结合

欧拉公式

$$Ae^{ik} = A(\cos k + i \sin k)$$

```
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
[illegible](I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_[illegible]
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
```

将一幅图像的幅度与另一幅图像的相位相结合

```
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*
[illegible]
_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
```

计算第一幅图像的幅度分量

```
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
[illegible](I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_a[illegible]
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
```

计算第二幅图的相位分量

```
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
                            l(np.fft.ifft2(abs2_phase1))
```

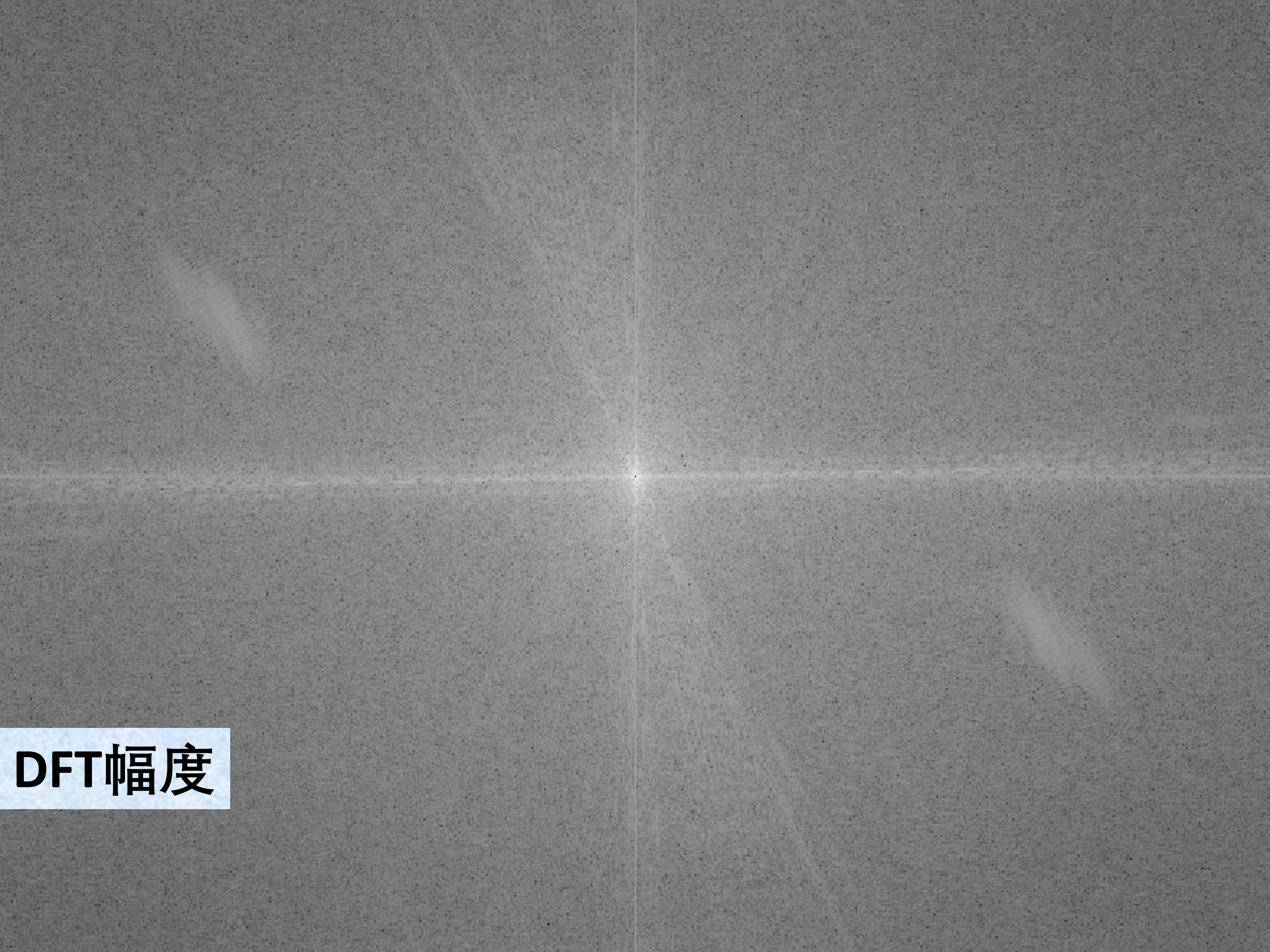
2D离散傅里叶逆变换

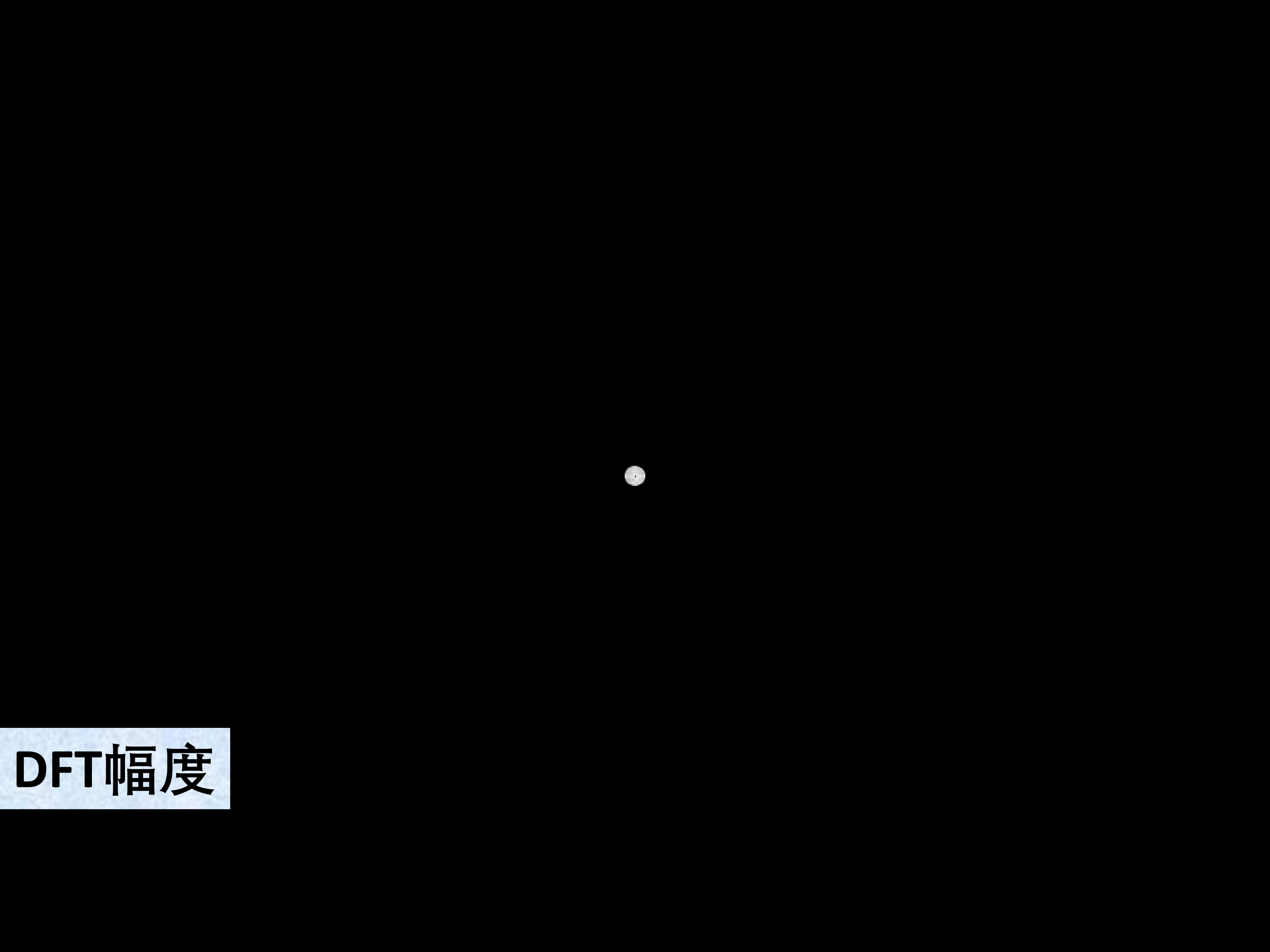
```
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
>>> I1 = cv2.imread('cuc-garden.png', cv2.IMREAD_GRAYSCALE)
>>> I2 = cv2.imread('cuc-gate.png', cv2.IMREAD_GRAYSCALE)
>>> I1_fft = np.ftt.fft2(I1.astype(float))
>>> I2_fft = np.ftt.fft2(I2.astype(float))
>>> abs1_phase2 = np.abs(I1_fft)*np.exp(1j*np.angle(I2_fft))
>>> abs2_phase1 = np.abs(I2_fft)*np.exp(1j*np.angle(I1_fft))
>>> I_abs1_phase2 = np.real(np.fft.ifft2(abs1_phase2))
>>> I_abs2_phase1 = np.real(np.fft.ifft2(abs2_phase1))
```

Python时间

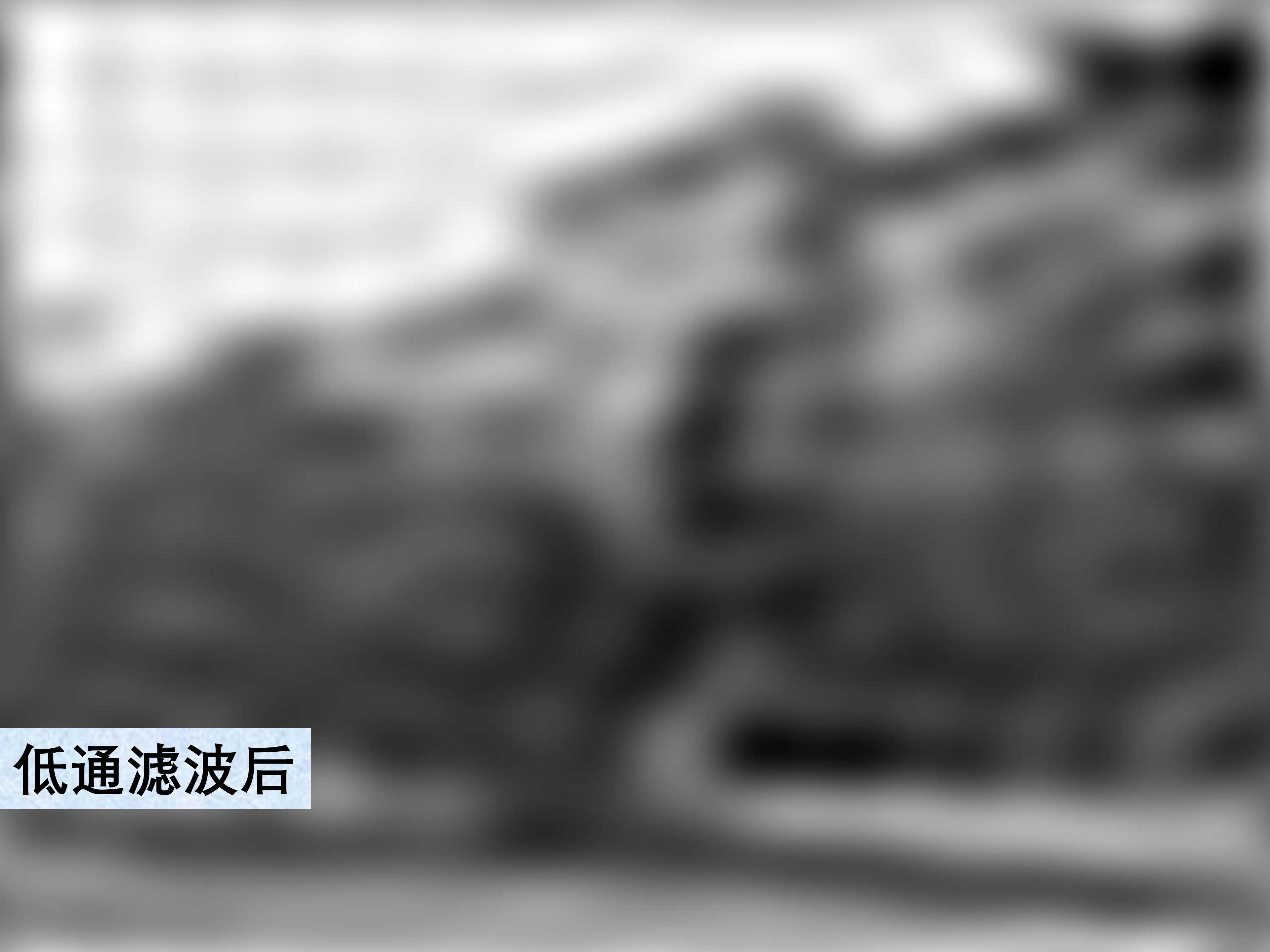
已结束

输入图像

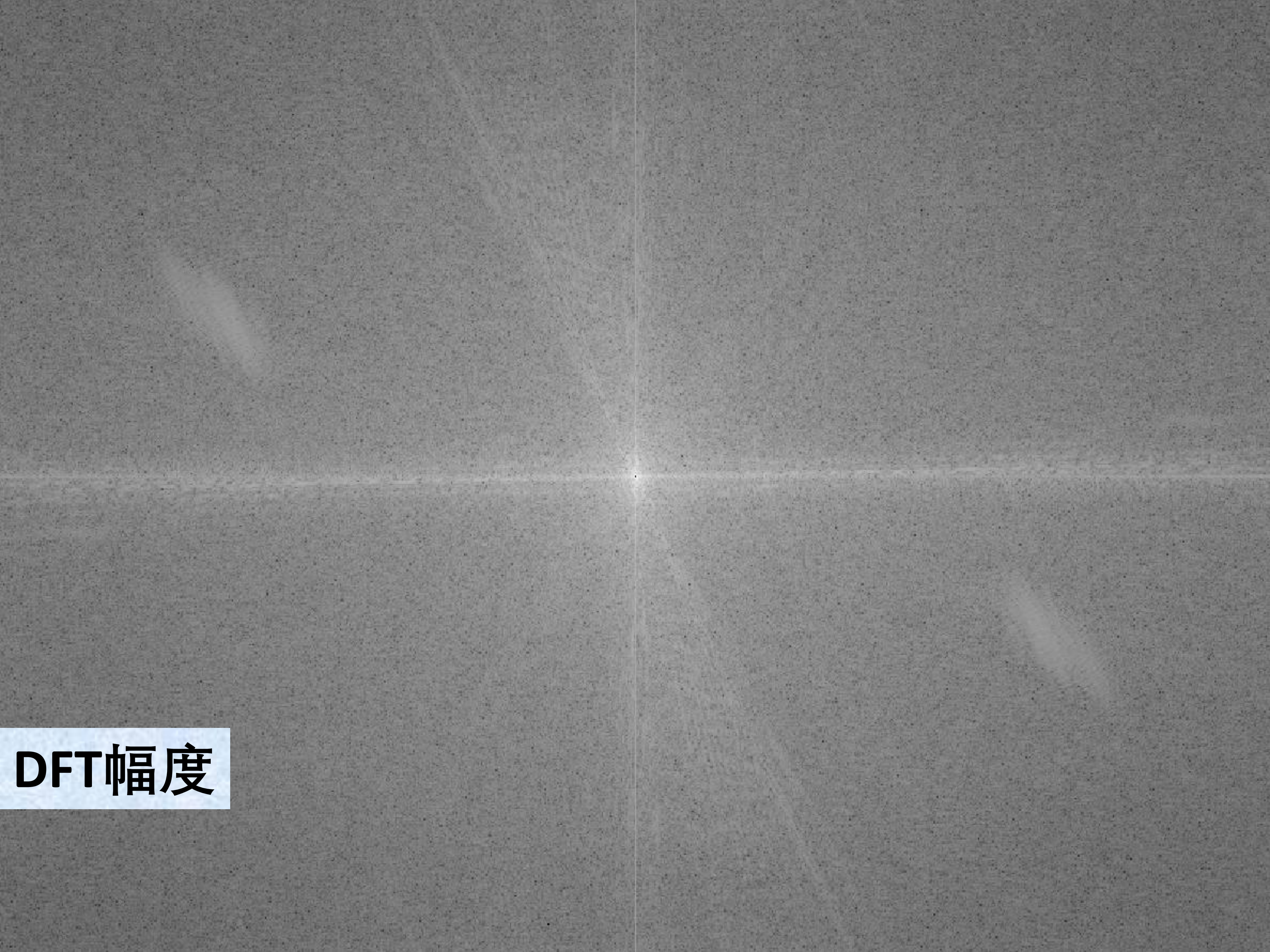

DFT幅度

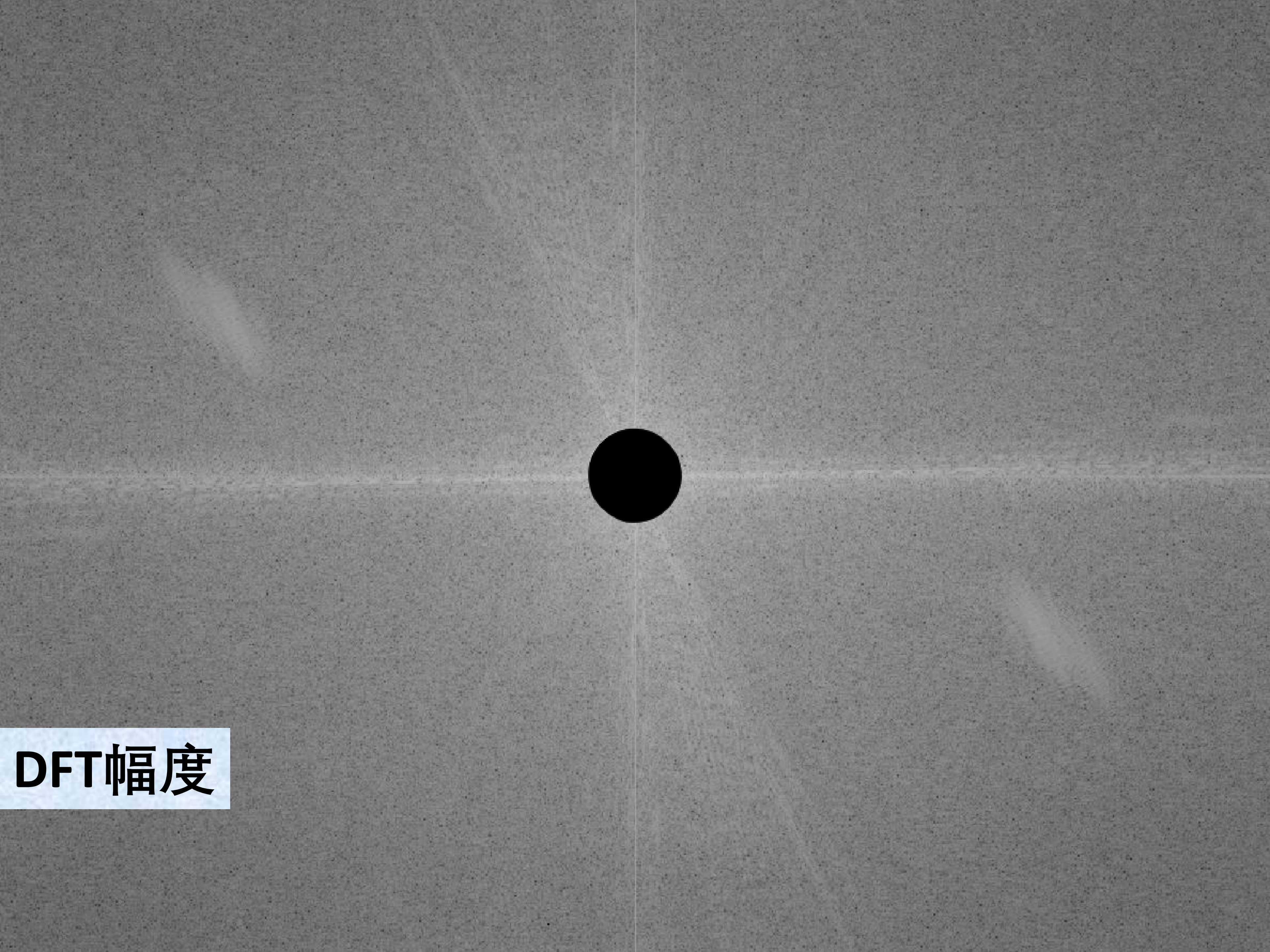


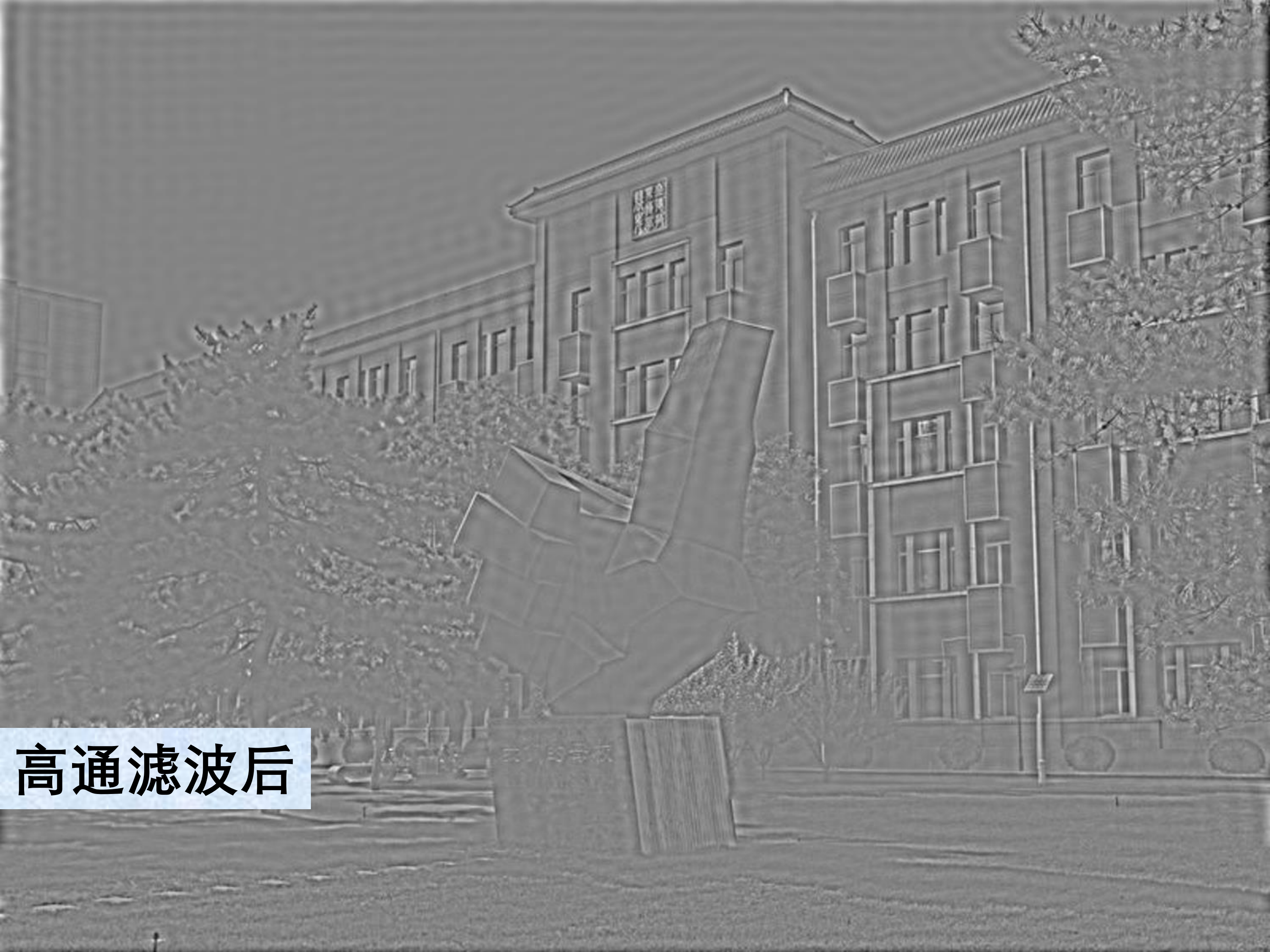
DFT幅度


低通滤波后

输入图像


DFT幅度

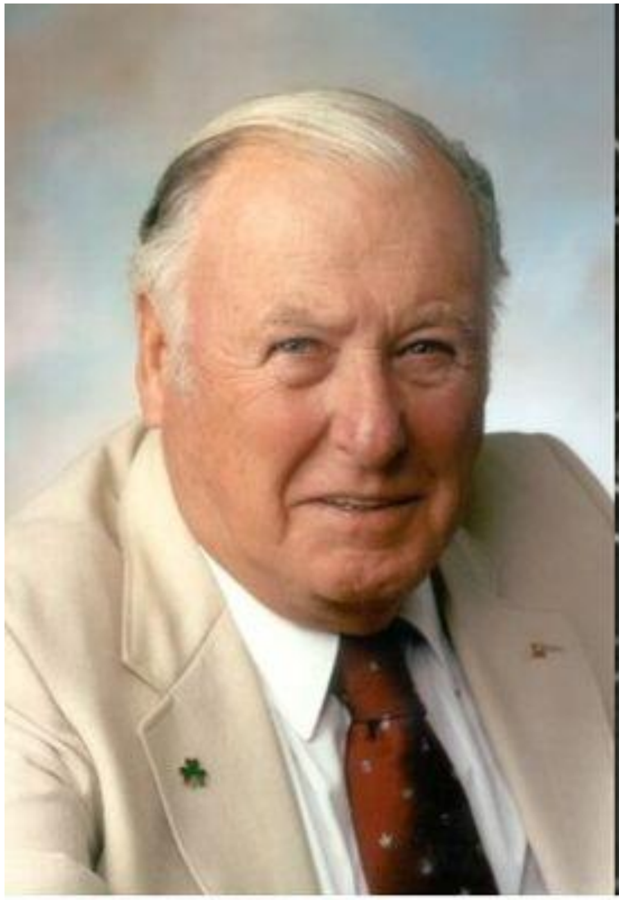

DFT幅度

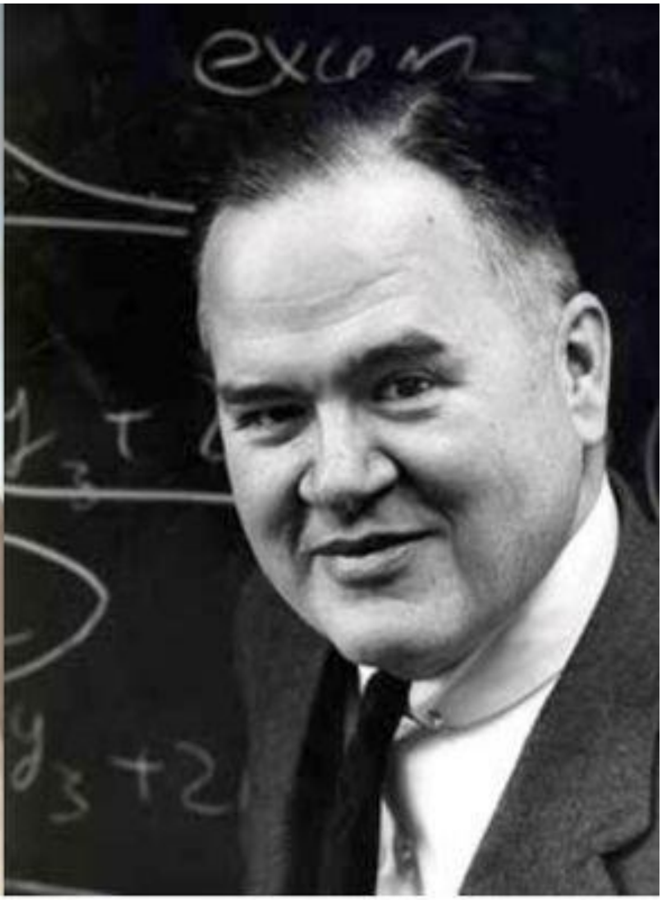


高通滤波后

FFT

Fast Fourier Transform
快速傅里叶变换


詹姆斯·威廉·库利
(1926-)


约翰·怀尔德·图基
(1915-2000)

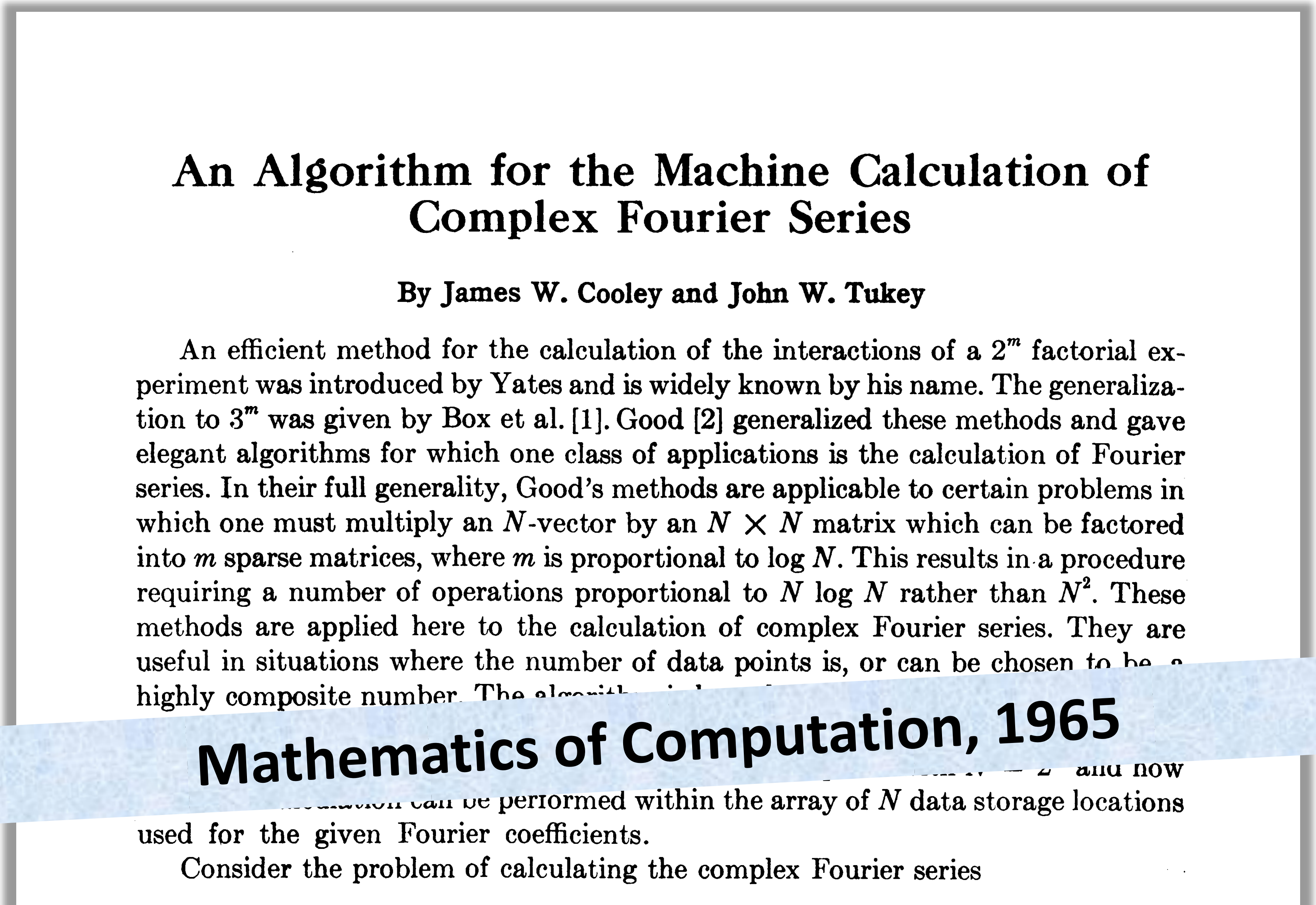
卡尔·弗里德里希·高斯

$$O(N^2)$$

离散傅里叶变换的时间复杂度

$$O(N \log N)$$

快速傅里叶变换的时间复杂度



An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to log N. This results in a procedure requiring a number of operations proportional to N log N rather than N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be a highly composite number. The algorit

and now

can be performed within the array of N data storage locations used for the given Fourier coefficients.

Consider the problem of calculating the complex Fourier series

Mathematics of Computation, 1965

from *SIAM News*, Volume 33, Number 4

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

Algos is the Greek word for pain. *Algor* is Latin, to be cold. Neither is the root for *algorithm*, which stems instead from al-Khwarizmi, the name of the ninth-century Arab scholar whose book *al-jabr wa'l muqabalah* devolved into today's high school algebra textbooks. Al-Khwarizmi stressed the importance of methodical procedures for solving problems. Were he around today, he'd no doubt be impressed by the advances in his eponymous approach.

Some of the very best algorithms of the computer age are highlighted in the January/February 2000 issue of *Computing in Science & Engineering*, a joint publication of the American Institute of Physics and the IEEE Computer Society. Guest editors Jack Don-garra of the University of Tennessee and Oak Ridge National Laboratory and Fran-cis Sullivan of the Center for Comput-ing Sciences at the Institute for Defense Analyses put togeth-er a list they call the "Top Ten Algorithms of the Century."

"We tried to assemble the 10 al-gorithms with the greatest influence on the development and practice of science and engineering in the 20th century," Dongarra and Sullivan write. As with any top-10 list, their selections—and non-selections—are bound to be controversial, they acknowledge. When it comes to picking the algorithmic best, there seems to be no best algorithm.

Without further ado, here's the CiSE top-10 list, in chronological order. (Dates and names associated with the algorithms should be read as first-order approximations. Most algorithms take shape over time, with many contributors.)

1946: John von Neumann, Stan Ulam, and Nick Metropolis, all at the Los Alamos Scientific Laboratory, cook up the Metropolis algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer's reputation for deterministic calculation, it's fitting that one of its earliest applications was the generation of random numbers.

In terms of wide-spread use, George Dantzig's simplex

1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**.

In terms of widespread application, Dantzig's algorithm is one of the most successful of all time: Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the "real" problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.

1950: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis

from *SIAM News*, Volume 33, Number 4

The Best of the 20th Century: Editors Name Top 10 Algorithms

By Barry A. Cipra

James Cooley

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories unveil the **fast Fourier Transform**.

Easily the most far-reaching algo-rithm in applied mathematics, the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of asteroids), but it was the Cooley–Tukey paper that made it clear how easily Fourier transforms can be computed. Like Quicksort, the FFT relies on a divide-and-conquer strategy to reduce an ostensibly $O(N^2)$ chore to an $O(N \log N)$ frolic. But unlike Quick- sort, the implementation is (at first sight) nonintuitive and less than straightforward. This in itself gave computer science an impetus to investigate the inherent complexity of computational problems and algorithms.

John Tukey

algorithm, also known as the **Monte Carlo method**.

The Metropolis algorithm aims to obtain approximate solutions to numerical problems with unmanageably many degrees of freedom and to combinatorial problems of factorial size, by mimicking a random process. Given the digital computer's reputation for deterministic calculation, it's fitting that one of its earliest applications was the generation of random numbers.

In terms of wide-spread use, George Dantzig's simplex

1947: George Dantzig, at the RAND Corporation, creates the **simplex method for linear programming**.

In terms of widespread application, Dantzig's algorithm is one of the most successful of all time: Linear programming dominates the world of industry, where economic survival depends on the ability to optimize within budgetary and other constraints. (Of course, the "real" problems of industry are often nonlinear; the use of linear programming is sometimes dictated by the computational budget.) The simplex method is an elegant way of arriving at optimal answers. Although theoretically susceptible to exponential delays, the algorithm in practice is highly efficient—which in itself says something interesting about the nature of computation.

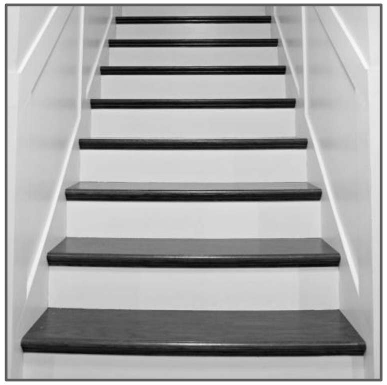
1950: Magnus Hestenes, Eduard Stiefel, and Cornelius Lanczos, all from the Institute for Numerical Analysis

测试

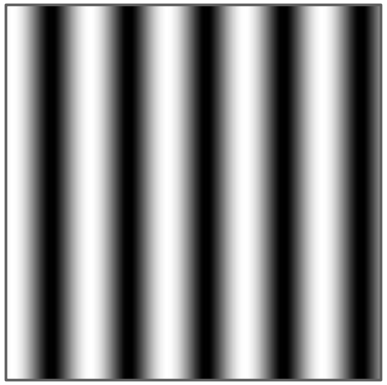
将空间域图像与傅立叶幅度图像匹配

将空间域图像与傅立叶幅度图像匹配

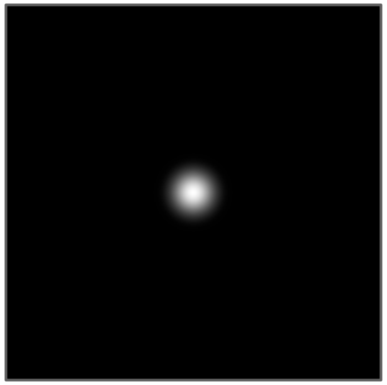
1



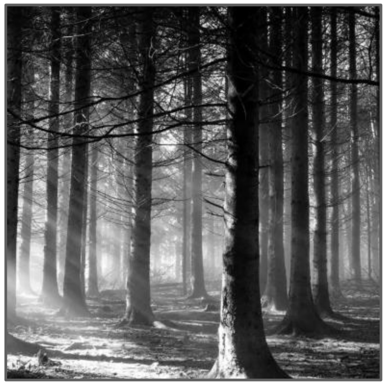
2

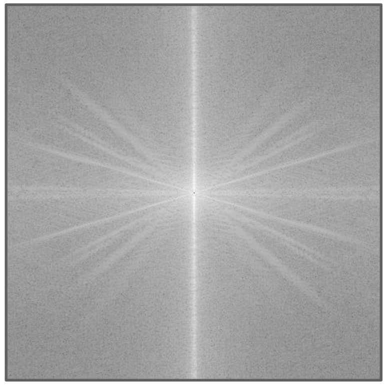


3

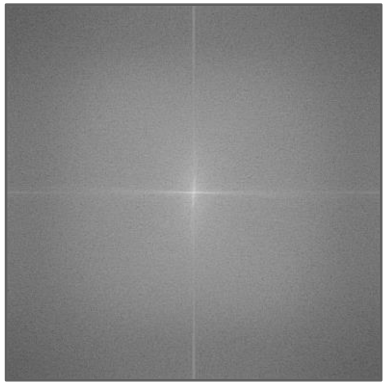


4

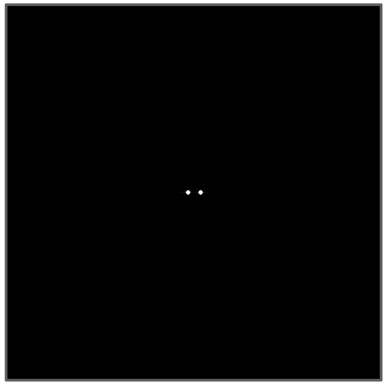




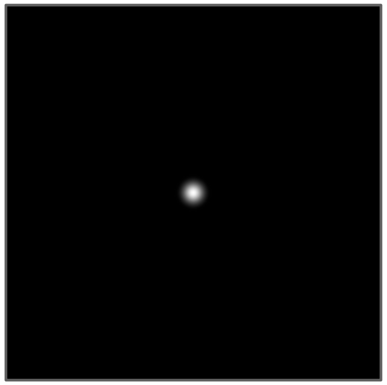
a



b

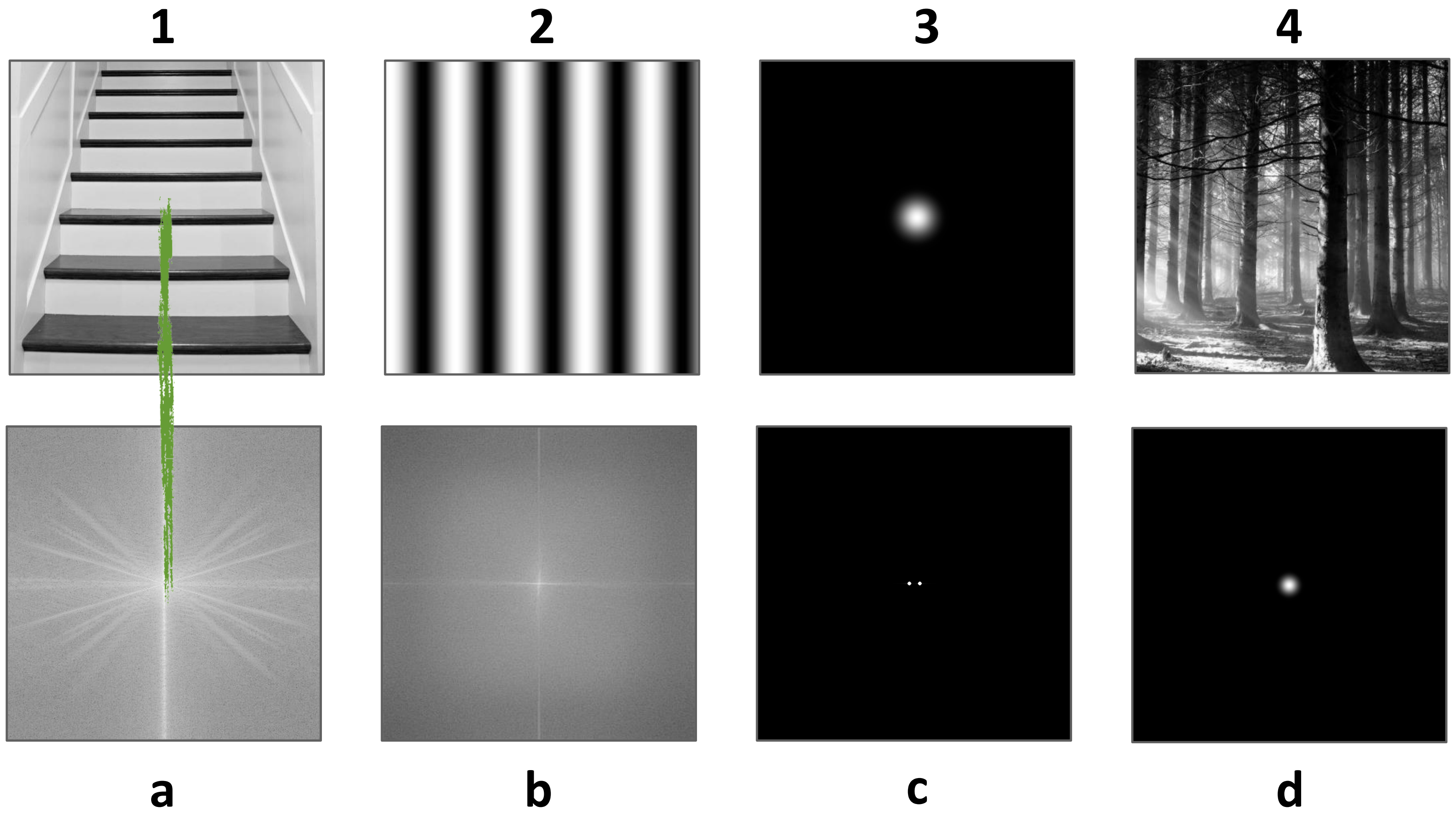


c

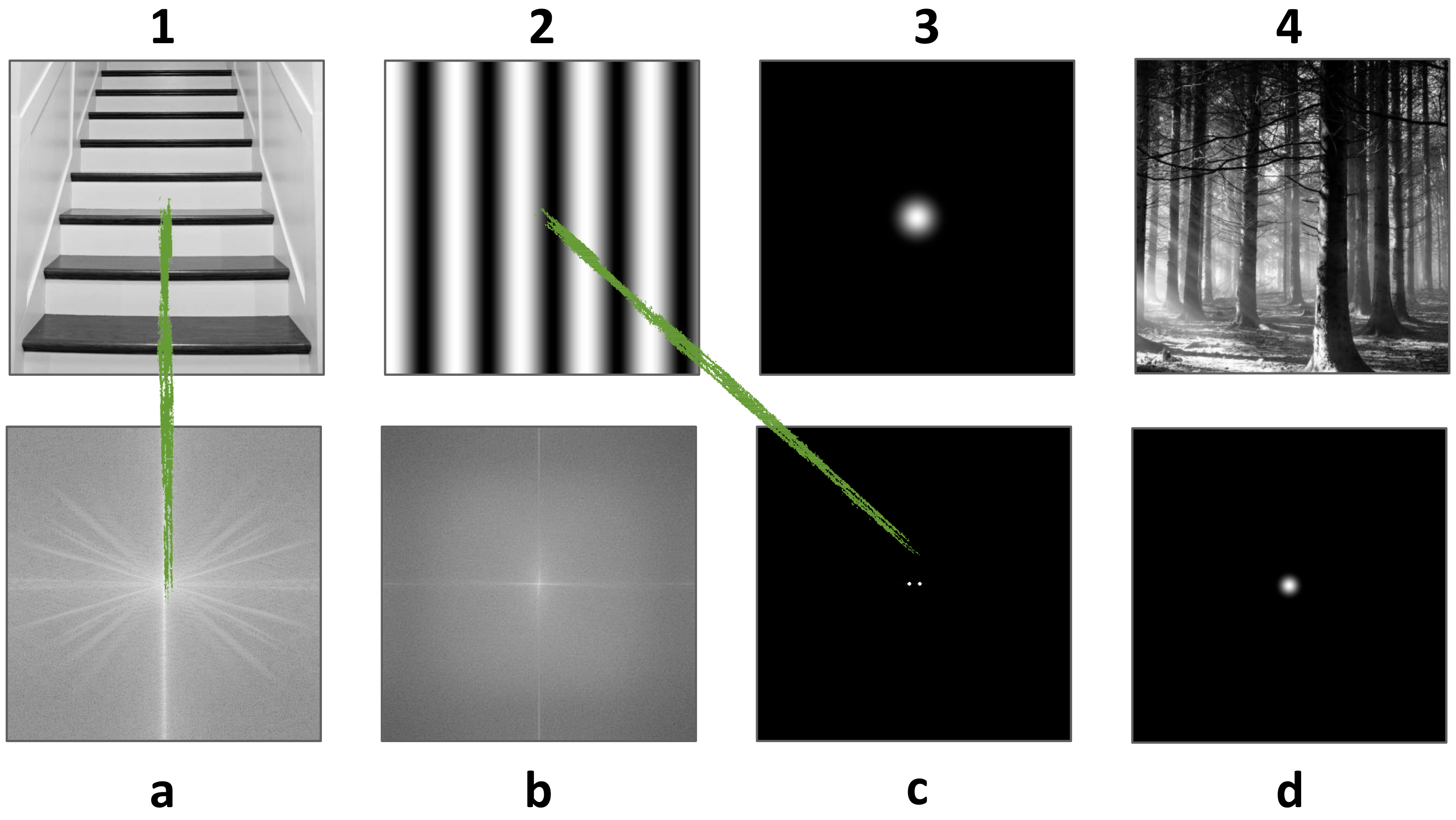


d

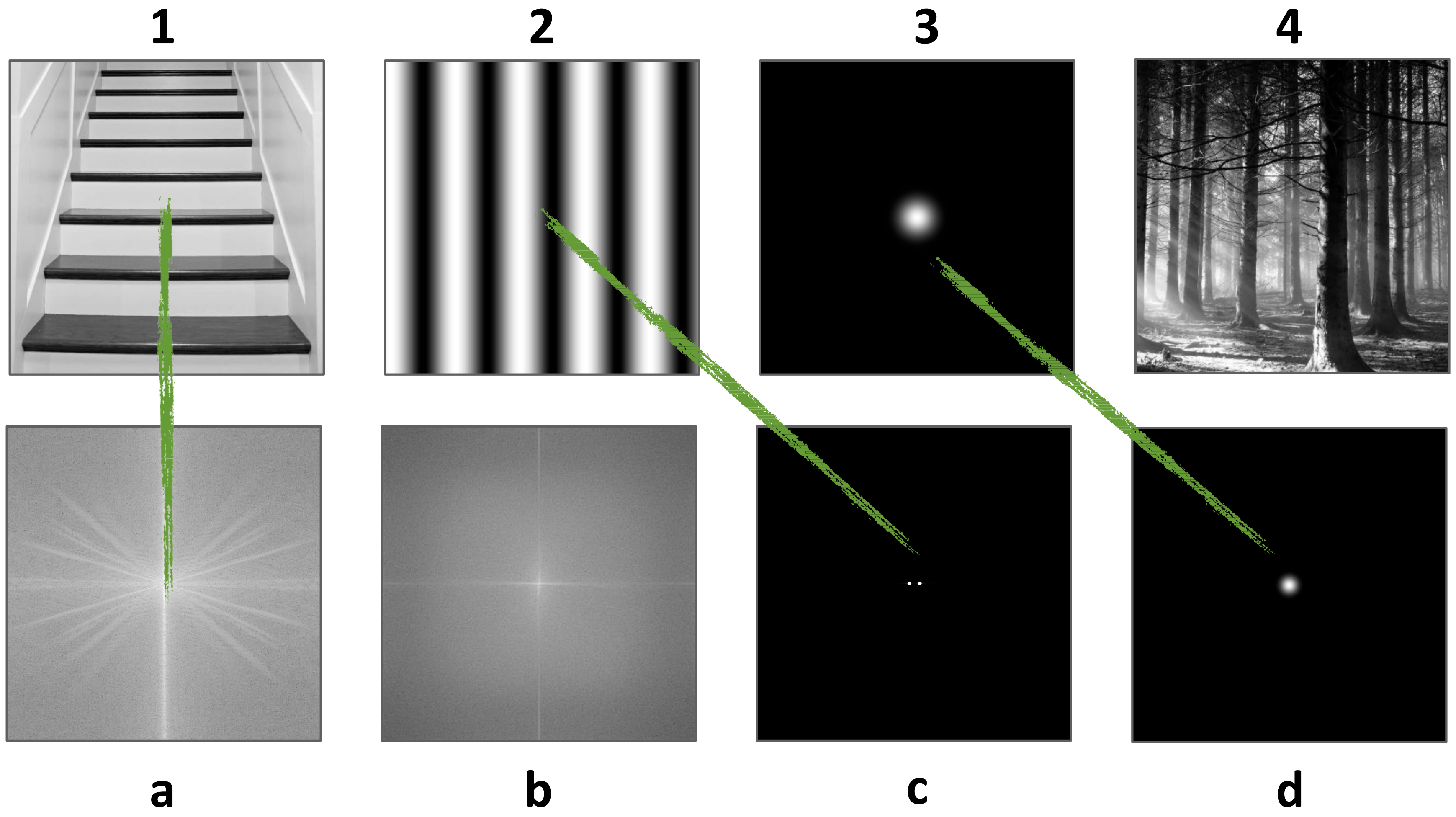
将空间域图像与傅立叶幅度图像匹配



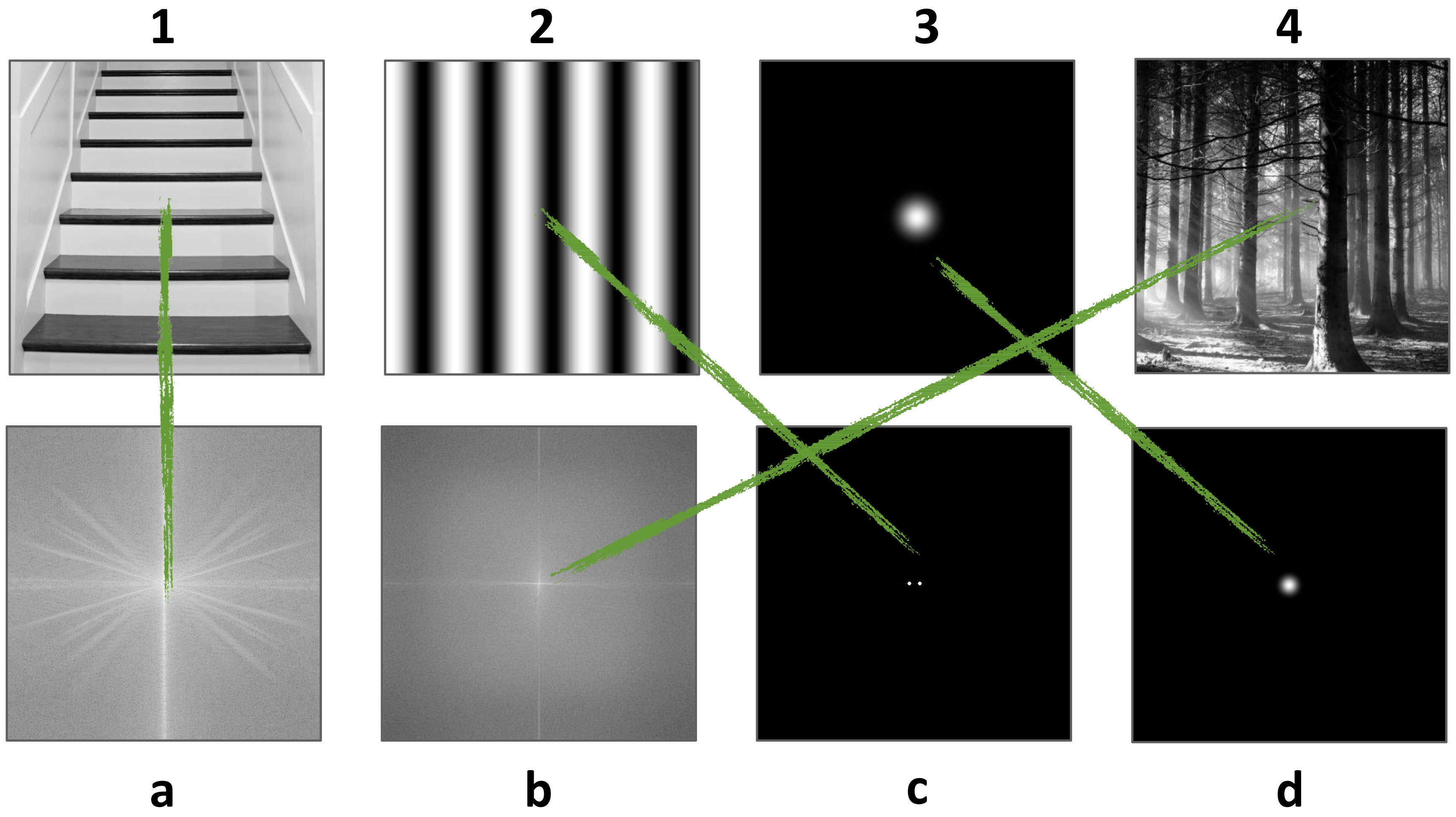
将空间域图像与傅立叶幅度图像匹配



将空间域图像与傅立叶幅度图像匹配

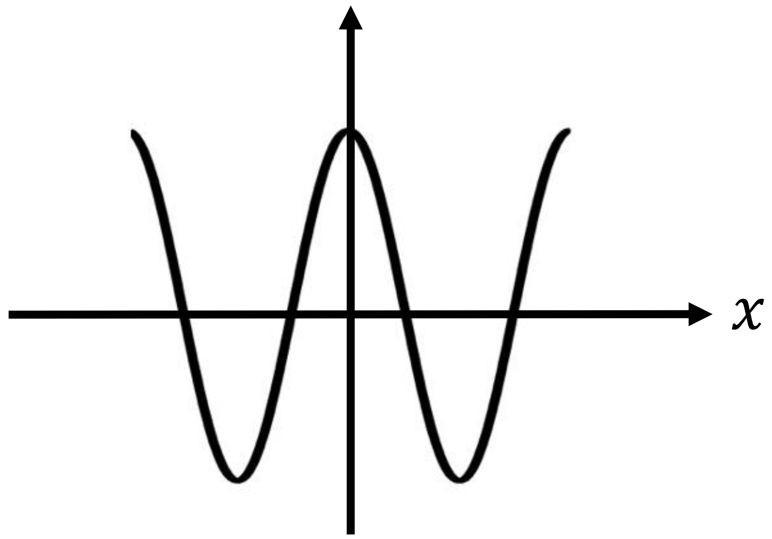


将空间域图像与傅立叶幅度图像匹配

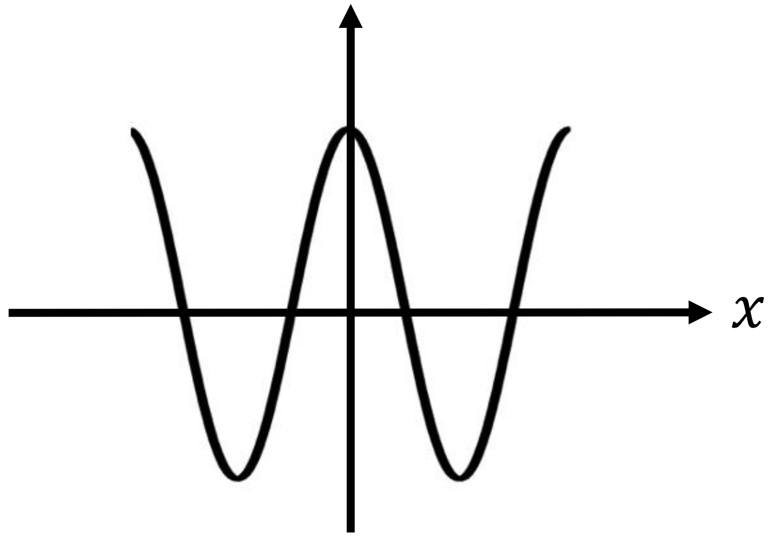


傅里叶变换对

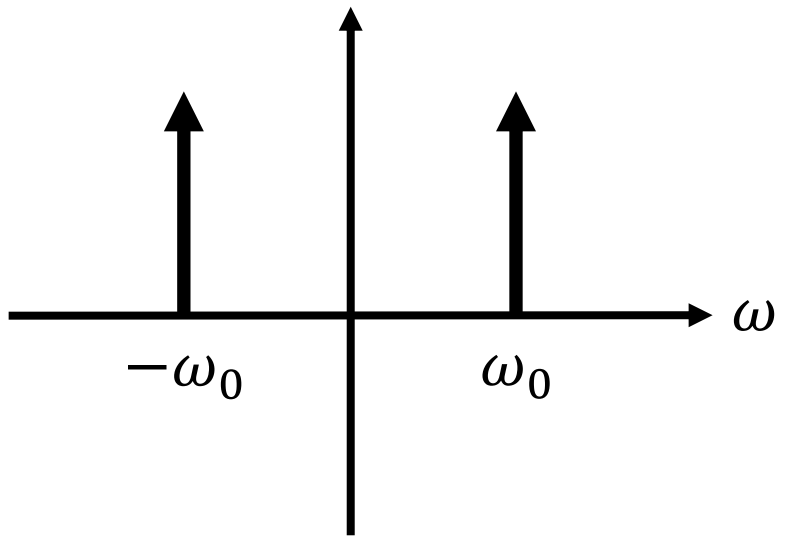
$\cos(2\pi\omega_0 x)$



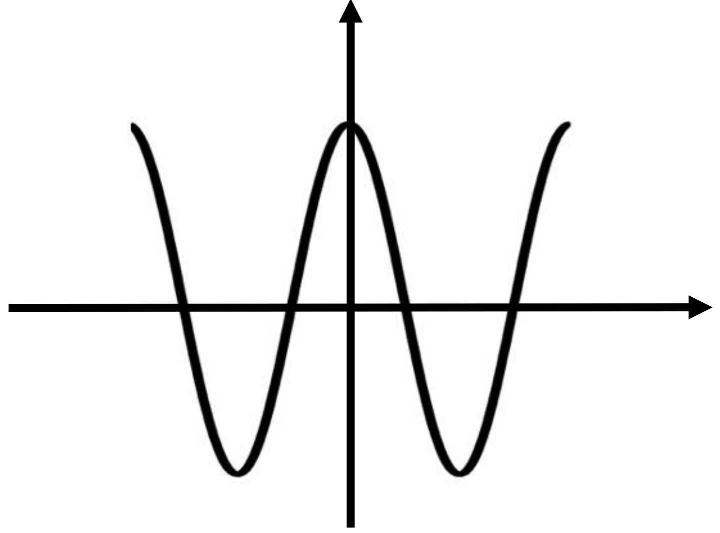
$\cos(2\pi\omega_0 x)$



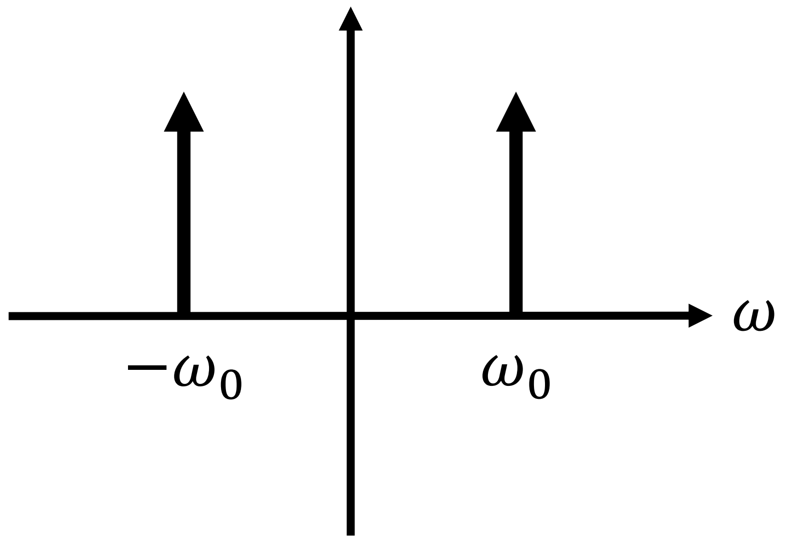
$\frac{1}{2}\big(\delta(\omega - \omega_0) + \delta(\omega + \omega_0)\big)$



$\cos(2\pi\omega_0 x)$

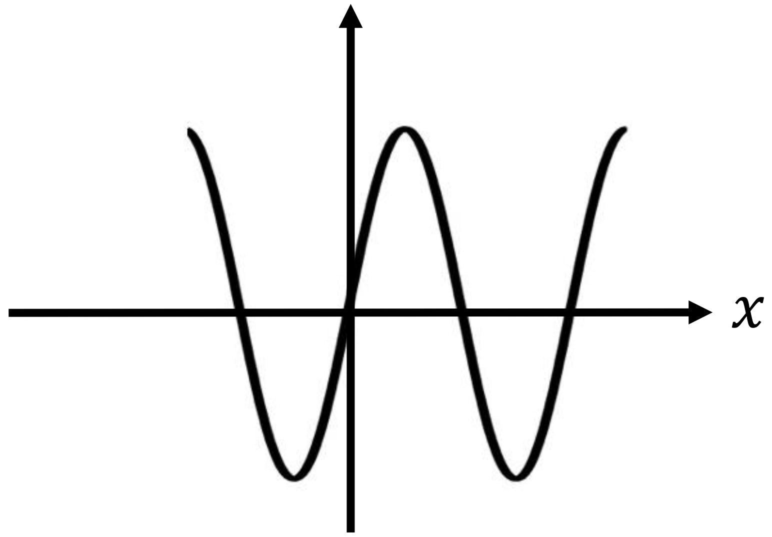


$$\frac{1}{2}\left(\delta(\omega-\omega_0)+\delta(\omega+\omega_0)\right)$$

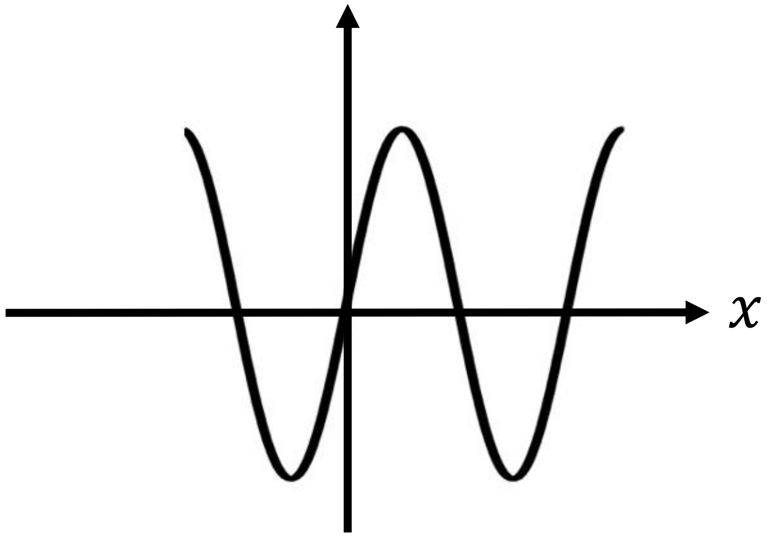


$$\cos(2\pi\omega_0 x)=\frac{1}{2}\left(e^{i2\pi\omega_0 x}+e^{-i2\pi\omega_0 x}\right)$$

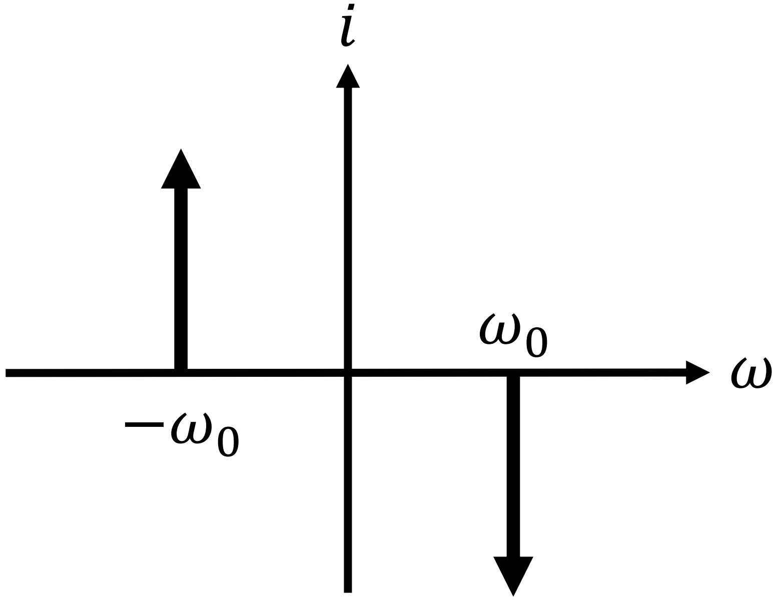
$\sin(2\pi\omega_0 x)$



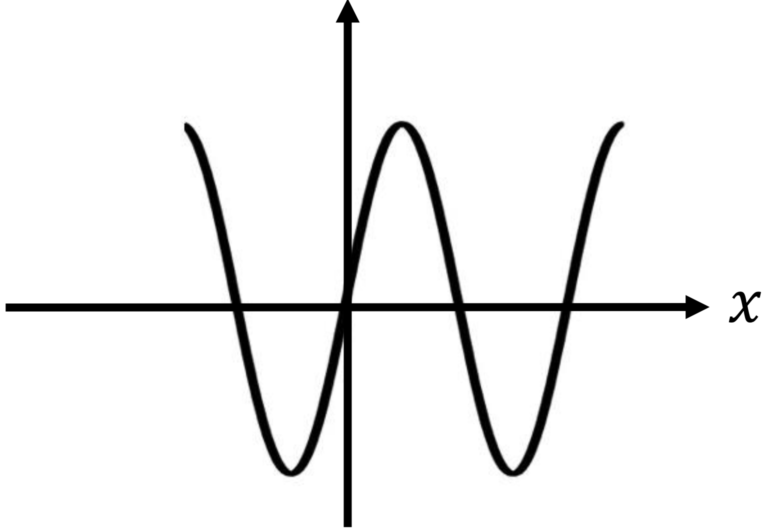
$\sin(2\pi\omega_0 x)$



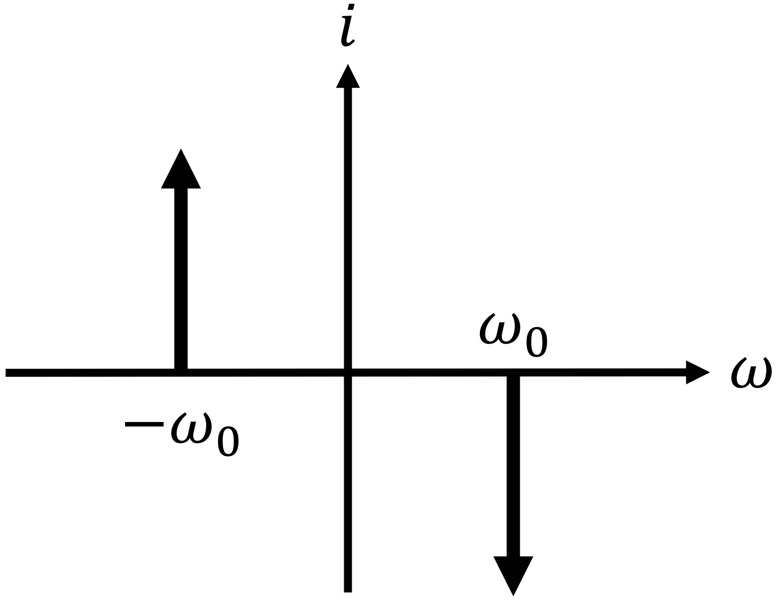
$\frac{1}{2}i\big(\delta(\omega+\omega_0)-\delta(\omega-\omega_0)\big)$



$$\sin(2\pi\omega_0 x)$$

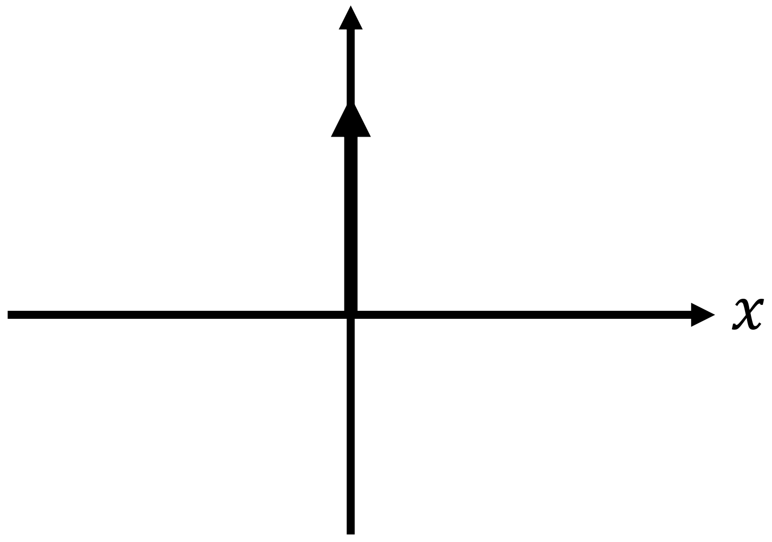


$$\frac{1}{2}i\big(\delta(\omega+\omega_0)-\delta(\omega-\omega_0)\big)$$

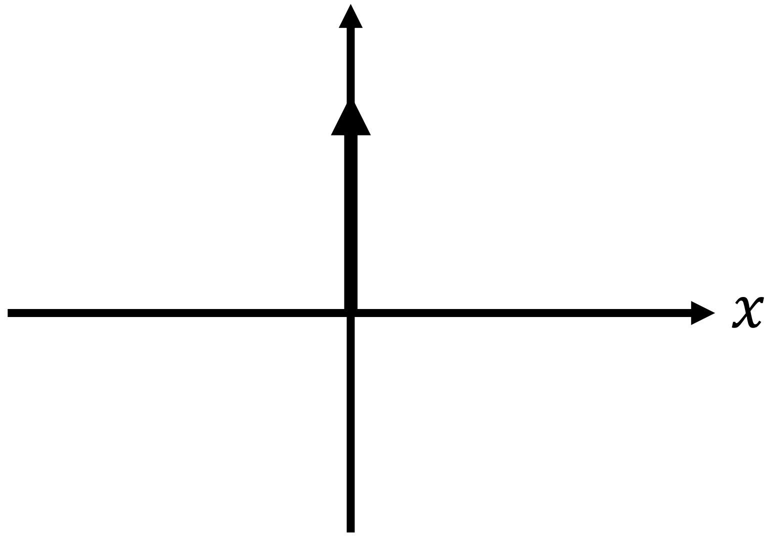


$$\sin(2\pi\omega_0 x)=\frac{1}{2i}\left(e^{i2\pi\omega_0 x}-e^{-i2\pi\omega_0 x}\right)$$

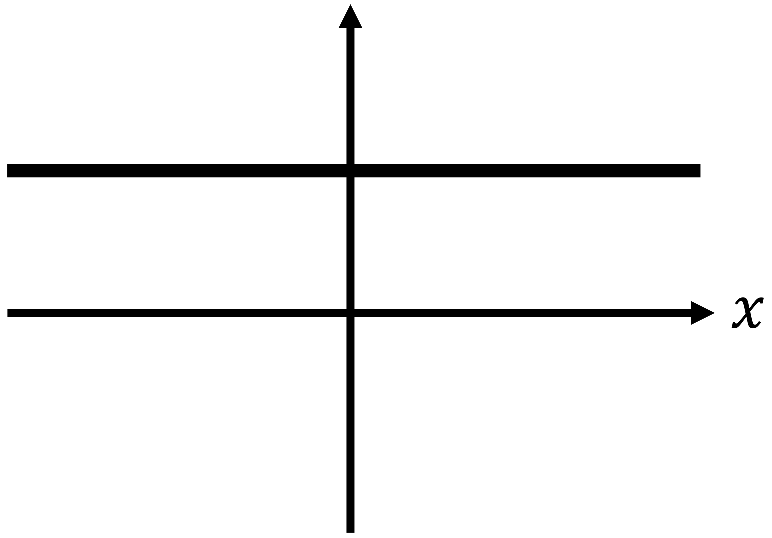
$$f(x) = \delta(x)$$



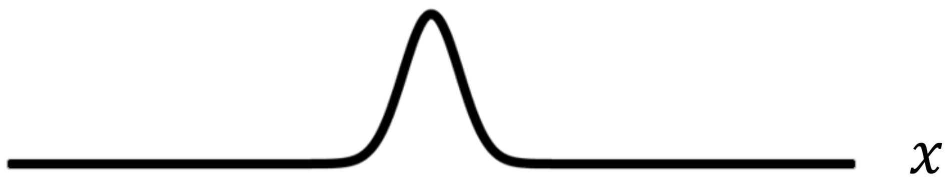
$f(x) = \delta(x)$



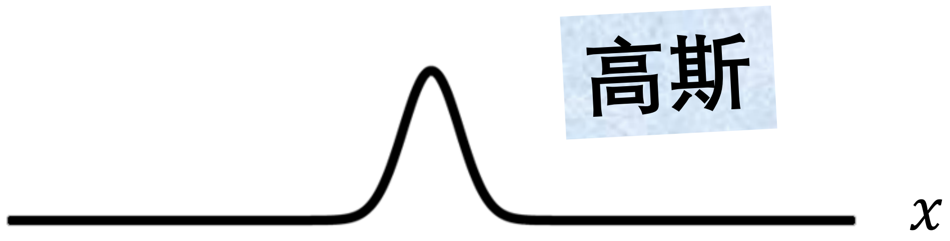
$F(\omega) = 1$



$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

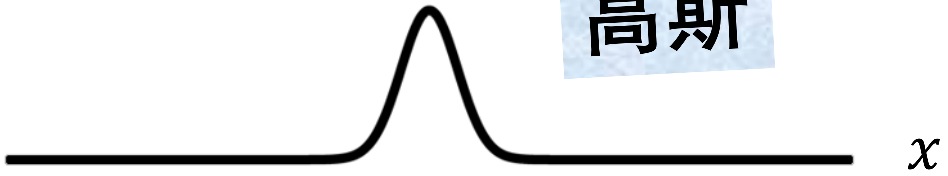


$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

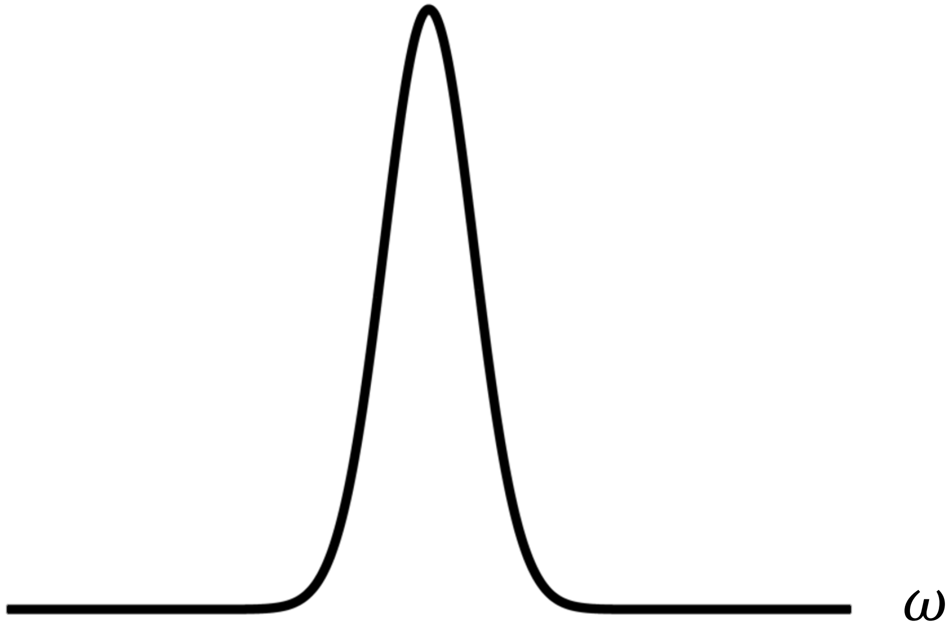


$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

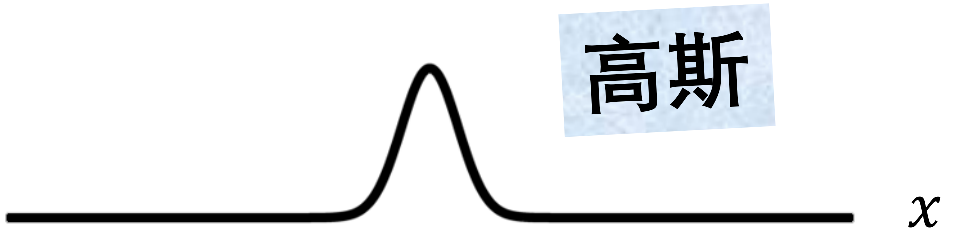
高斯



$$G(\omega) = e^{-\frac{(2\pi\omega)^2\sigma^2}{2}}$$

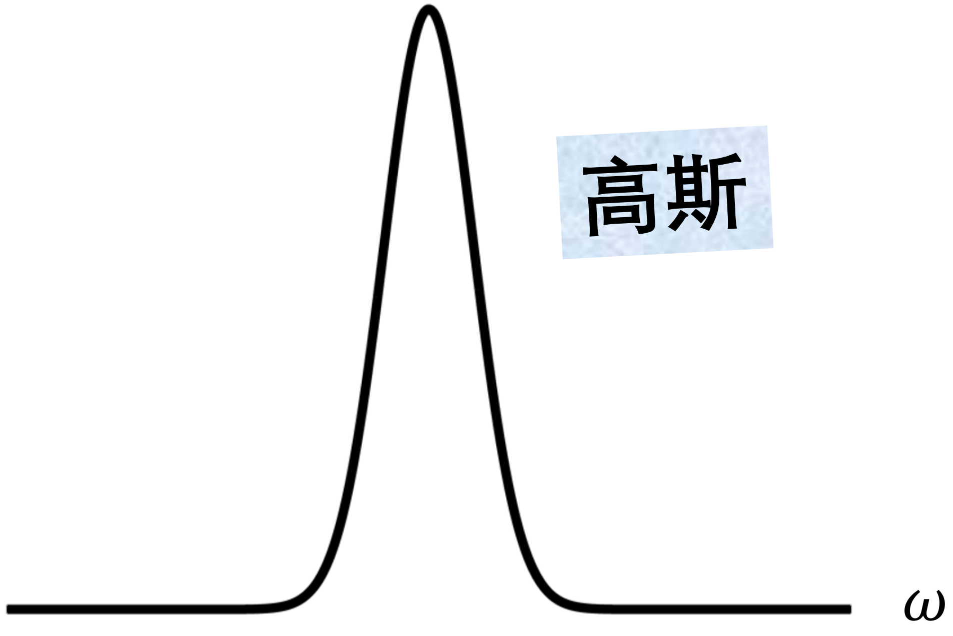


$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

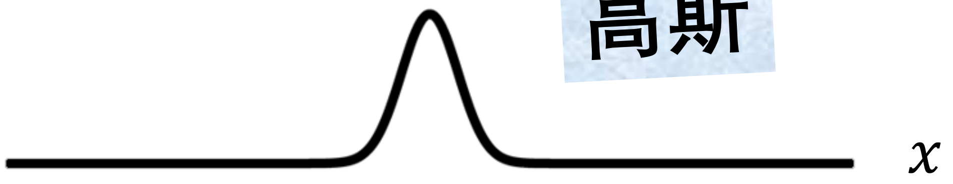


$$G(\omega) = e^{-\frac{(2\pi\omega)^2\sigma^2}{2}}$$

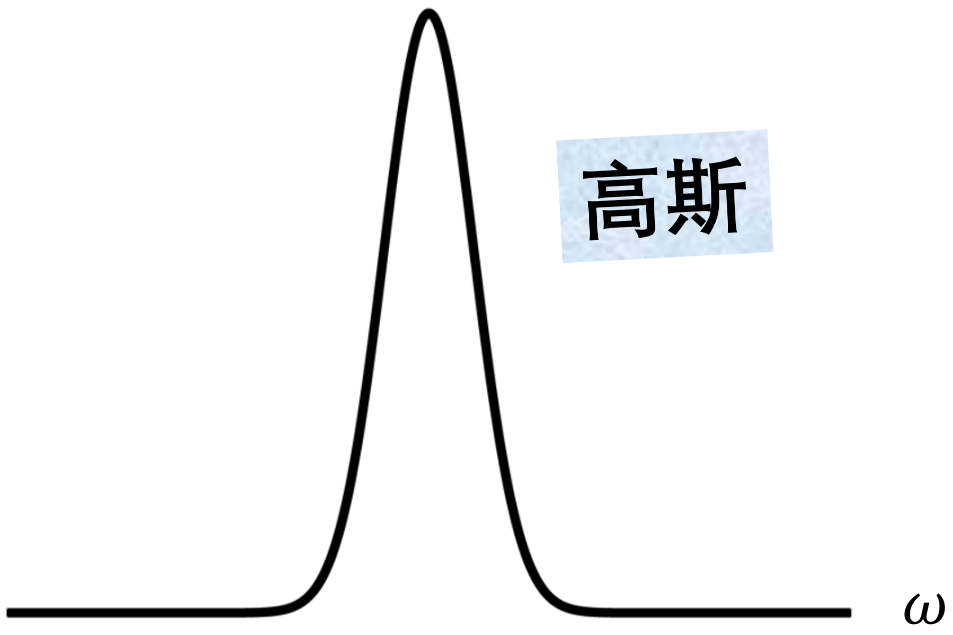
未归一化



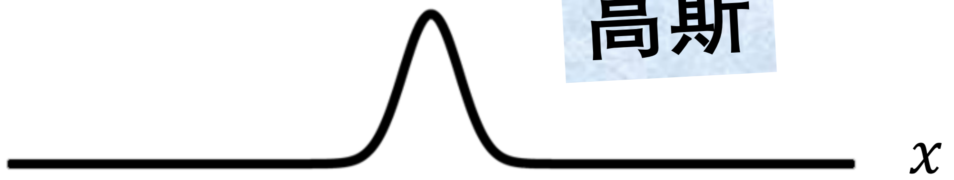
$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$



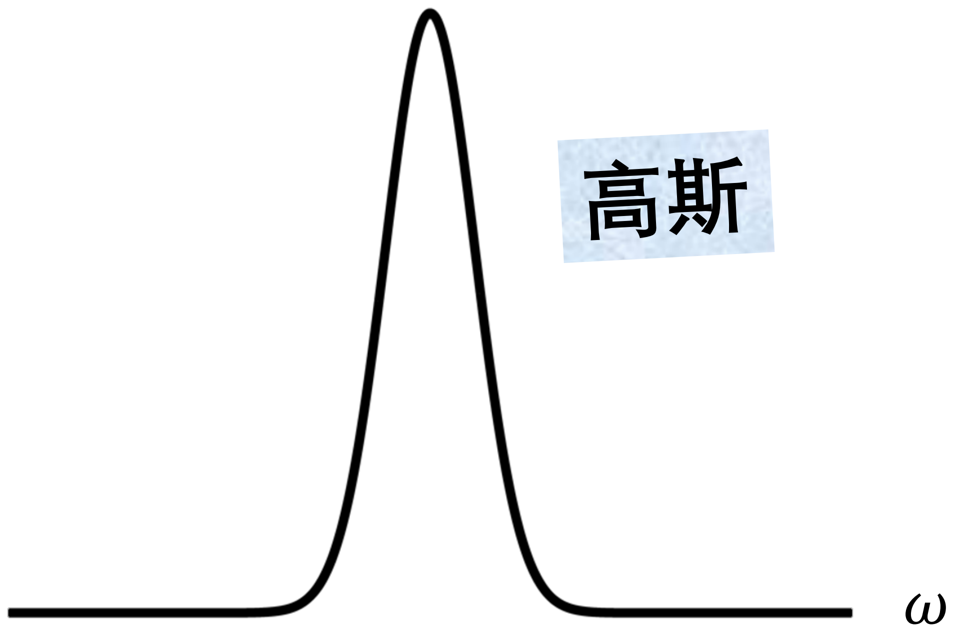
$$G(\omega) = e^{-\frac{(2\pi\omega)^2\sigma^2}{2}}$$



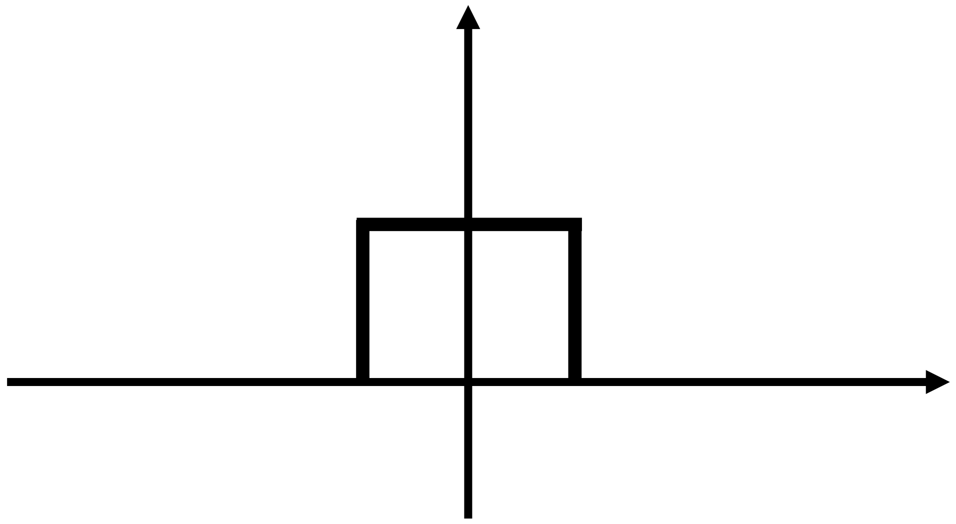
$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\boxed{\sigma^2}}}$$



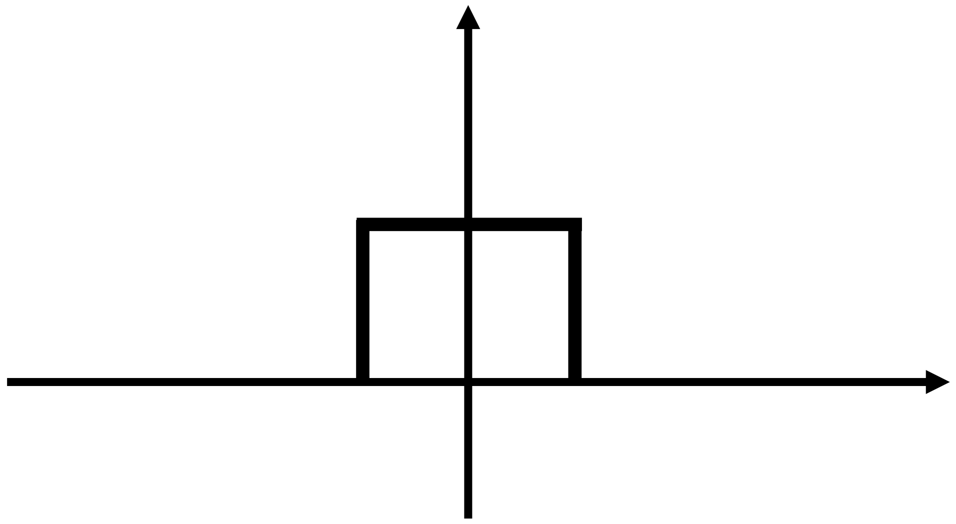
$$G(\omega) = e^{-\frac{(2\pi\omega)^2 \boxed{\sigma^2}}{2}}$$



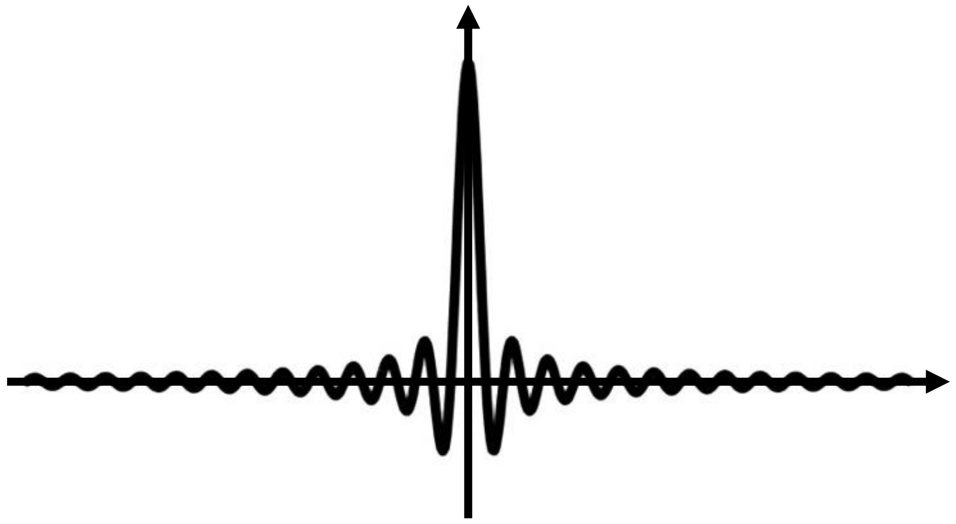
$\mathrm{box}(x)$



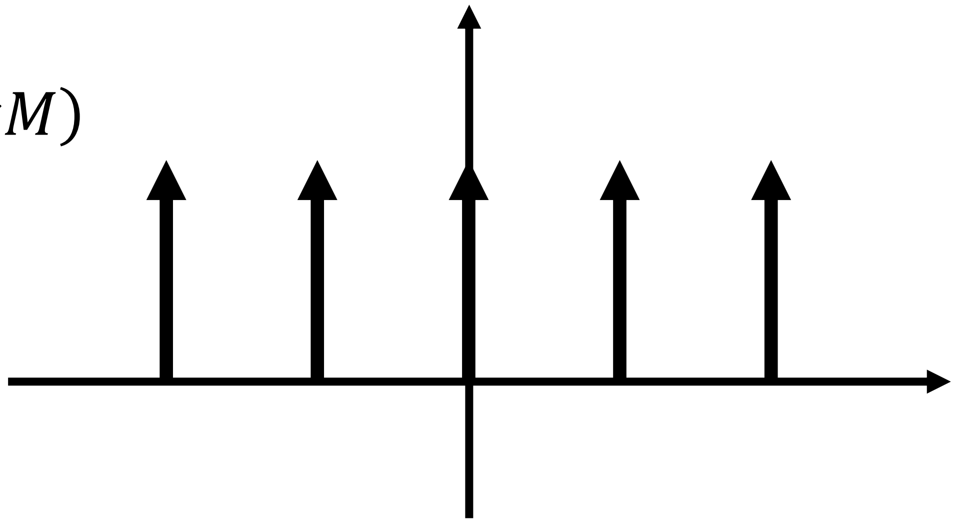
$\mathrm{box}(x)$



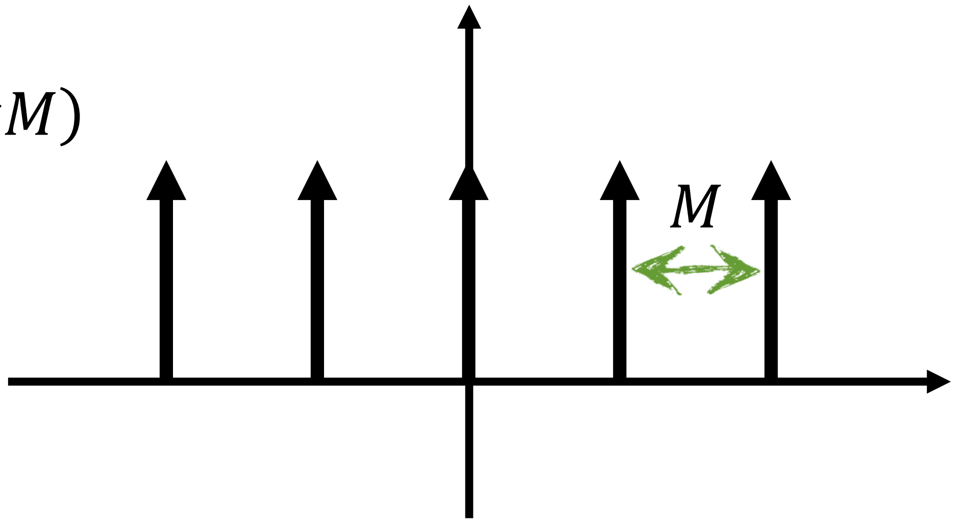
$\mathrm{sinc}(\omega)$



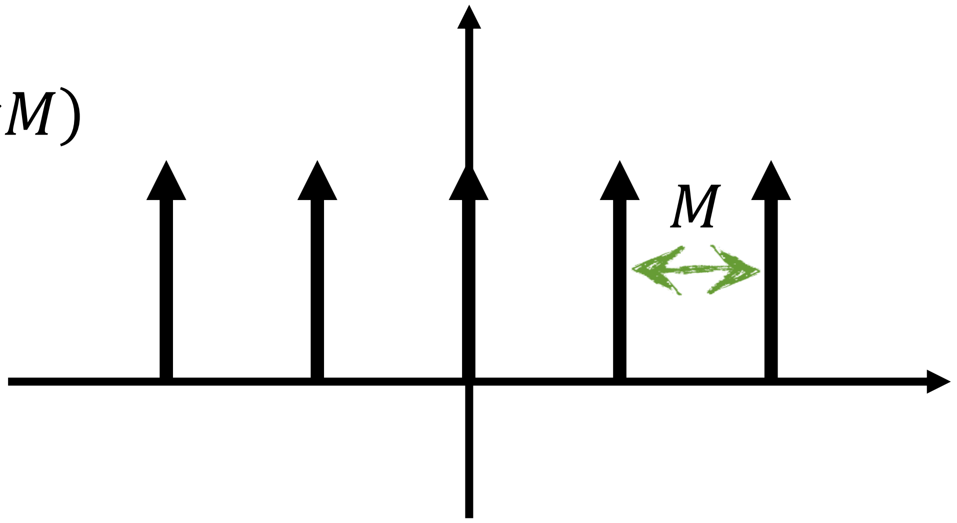
$$\mathrm{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$



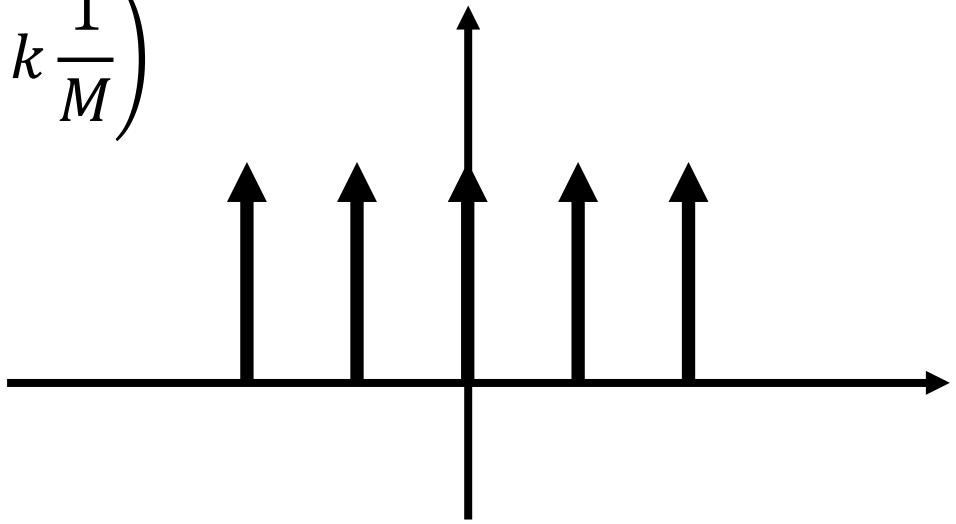
$$\text{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$



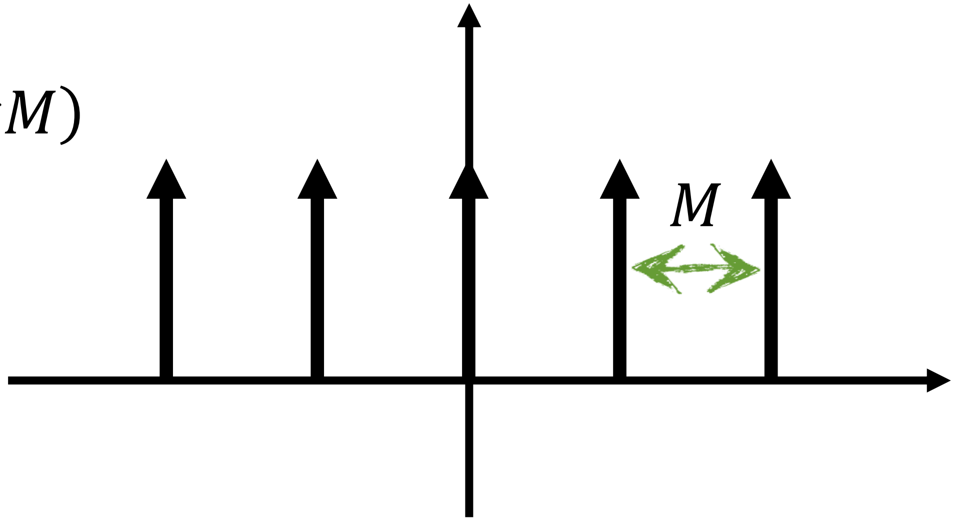
$$\text{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$



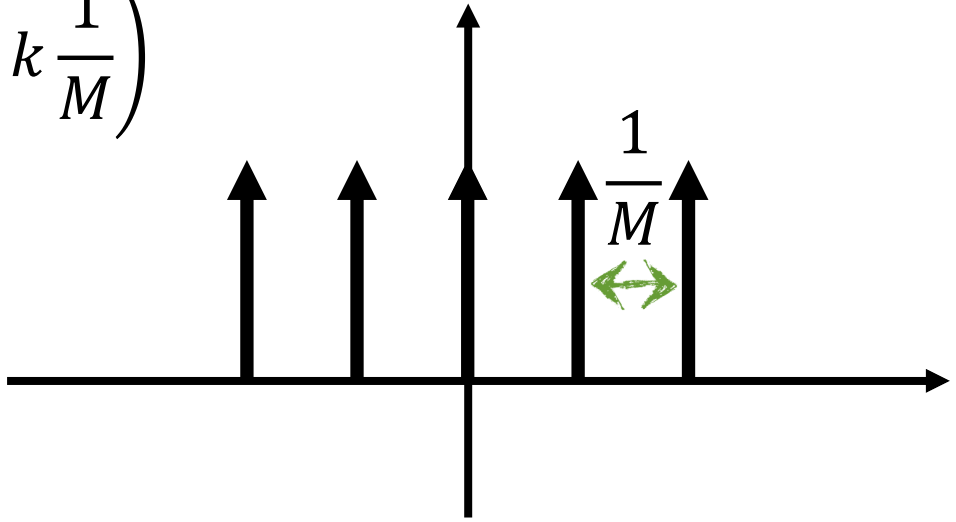
$$\text{comb}(\omega) = \frac{1}{M} \sum_{k=-\infty}^{\infty} \delta\left(\omega - k\frac{1}{M}\right)$$



$$\mathrm{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$



$$\mathrm{comb}(\omega) = \frac{1}{M} \sum_{k=-\infty}^{\infty} \delta\left(\omega - k\frac{1}{M}\right)$$



梳状滤波器用于
采样连续函数

傅里叶变换性质

傅里叶变换性质

空间域

频域

傅里叶变换性质

	空间域	频域
线性		

傅里叶变换性质

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	

傅里叶变换性质

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$

傅里叶变换性质

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放		

傅里叶变换性质

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	

傅里叶变换性质

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{\omega}{a}\right)$

傅里叶变换性质

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a\rvert} F\left(\frac{\omega}{a}\right)$
移位		

傅里叶变换性质

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{\omega}{a}\right)$
移位	$f(x - x_0)$	

傅里叶变换性质

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{\omega}{a}\right)$
移位	$f(x - x_0)$	$e^{-i2\pi\omega x_0} F(\omega)$

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a\rvert} F\left(\frac{\omega}{a}\right)$
移位	$f(x - x_0)$	$e^{-i2\pi\omega x_0} F(\omega)$
微分		

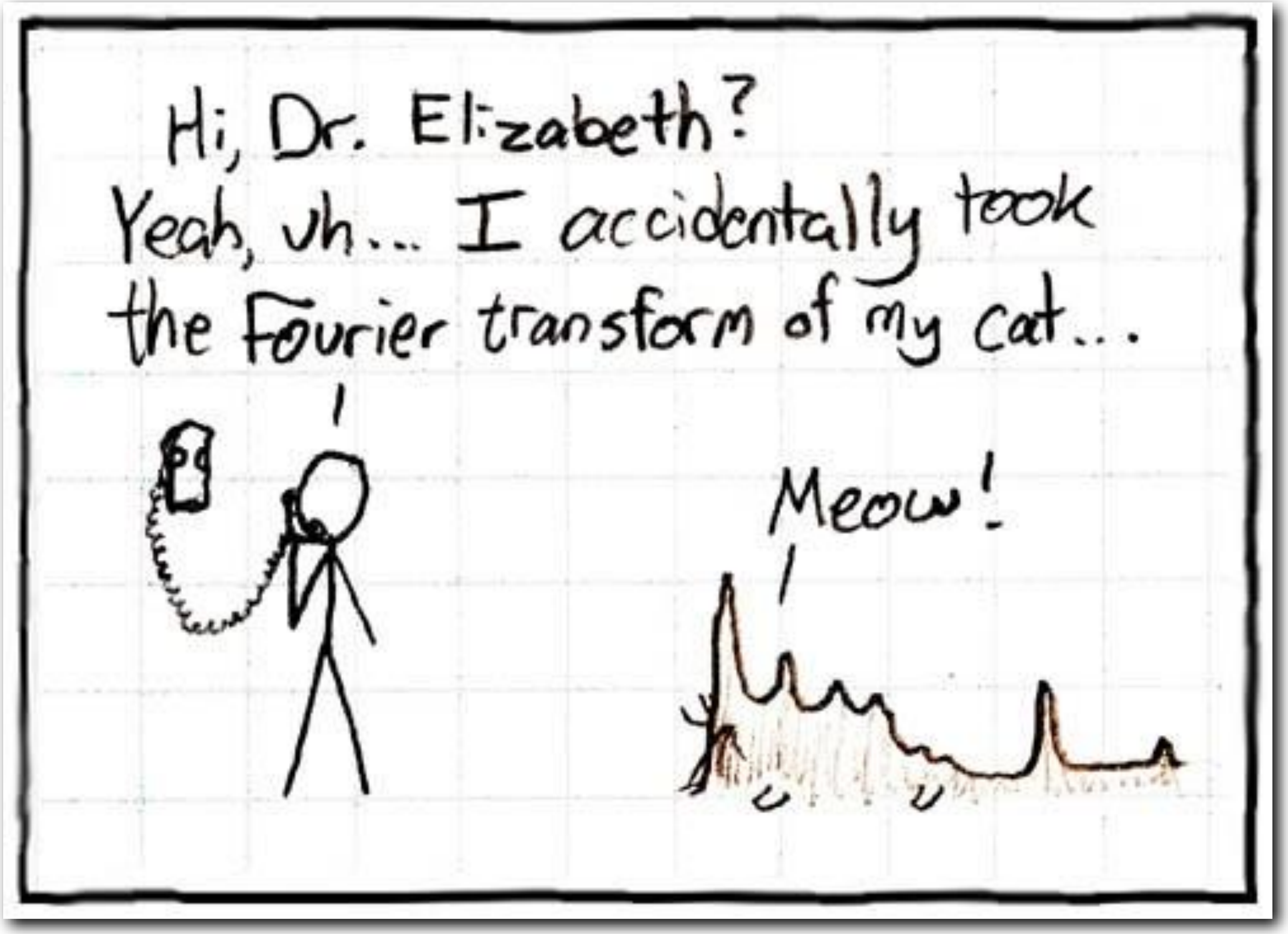
	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{\omega}{a}\right)$
移位	$f(x - x_0)$	$e^{-i2\pi\omega x_0} F(\omega)$
微分	$\frac{d^n f(x)}{dx^n}$	

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{\omega}{a}\right)$
移位	$f(x - x_0)$	$e^{-i2\pi\omega x_0} F(\omega)$
微分	$\frac{d^n f(x)}{dx^n}$	$(i2\pi\omega)^n F(\omega)$

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a\rvert} F\left(\frac{\omega}{a}\right)$
移位	$f(x - x_0)$	$e^{-i2\pi\omega x_0} F(\omega)$
微分	$\frac{d^n f(x)}{dx^n}$	$(i2\pi\omega)^n F(\omega)$
卷积		

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{\omega}{a}\right)$
移位	$f(x - x_0)$	$e^{-i2\pi\omega x_0} F(\omega)$
微分	$\frac{d^n f(x)}{dx^n}$	$(i2\pi\omega)^n F(\omega)$
卷积	$f(x) * g(x)$	

	空间域	频域
线性	$c_1 f(x) + c_2 g(x)$	$c_1 F(\omega) + c_2 G(\omega)$
缩放	$f(ax)$	$\frac{1}{\lvert a \rvert} F\left(\frac{\omega}{a}\right)$
移位	$f(x - x_0)$	$e^{-i2\pi\omega x_0} F(\omega)$
微分	$\frac{d^n f(x)}{dx^n}$	$(i2\pi\omega)^n F(\omega)$
卷积	$f(x) * g(x)$	$F(\omega)G(\omega)$


Hi, Dr. Elizabeth?
Yeah, uh... I accidentally took the Fourier transform of my cat...
Meow!