# 计算机视觉

## 频率分析 下

中国传媒大学
COMMUNICATION UNIVERSITY OF CHINA

## 本节主题：

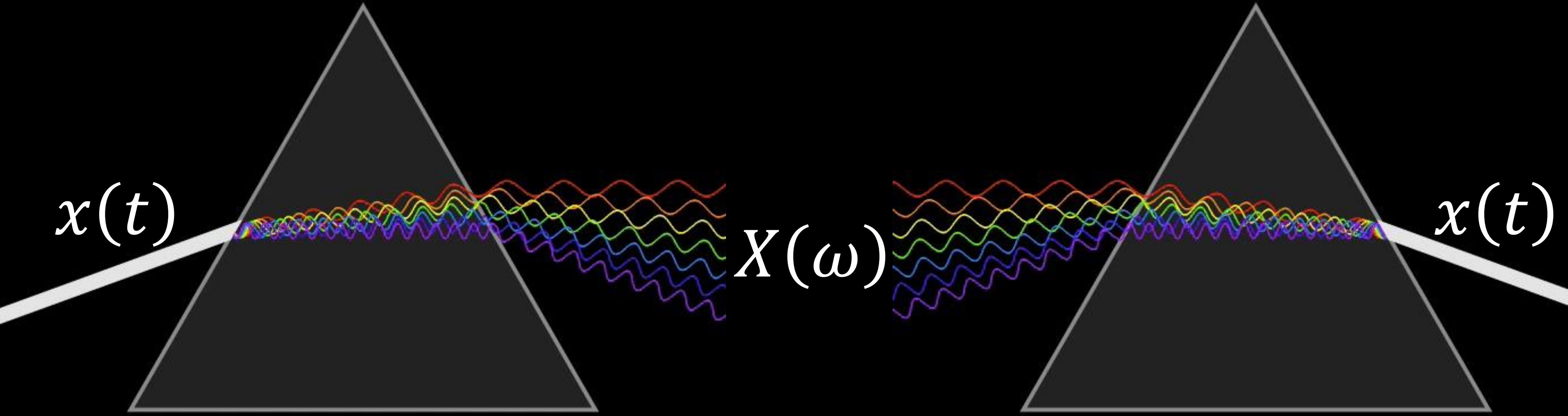
傅里叶变换

## 本节主题：

傅里叶变换

信号混叠

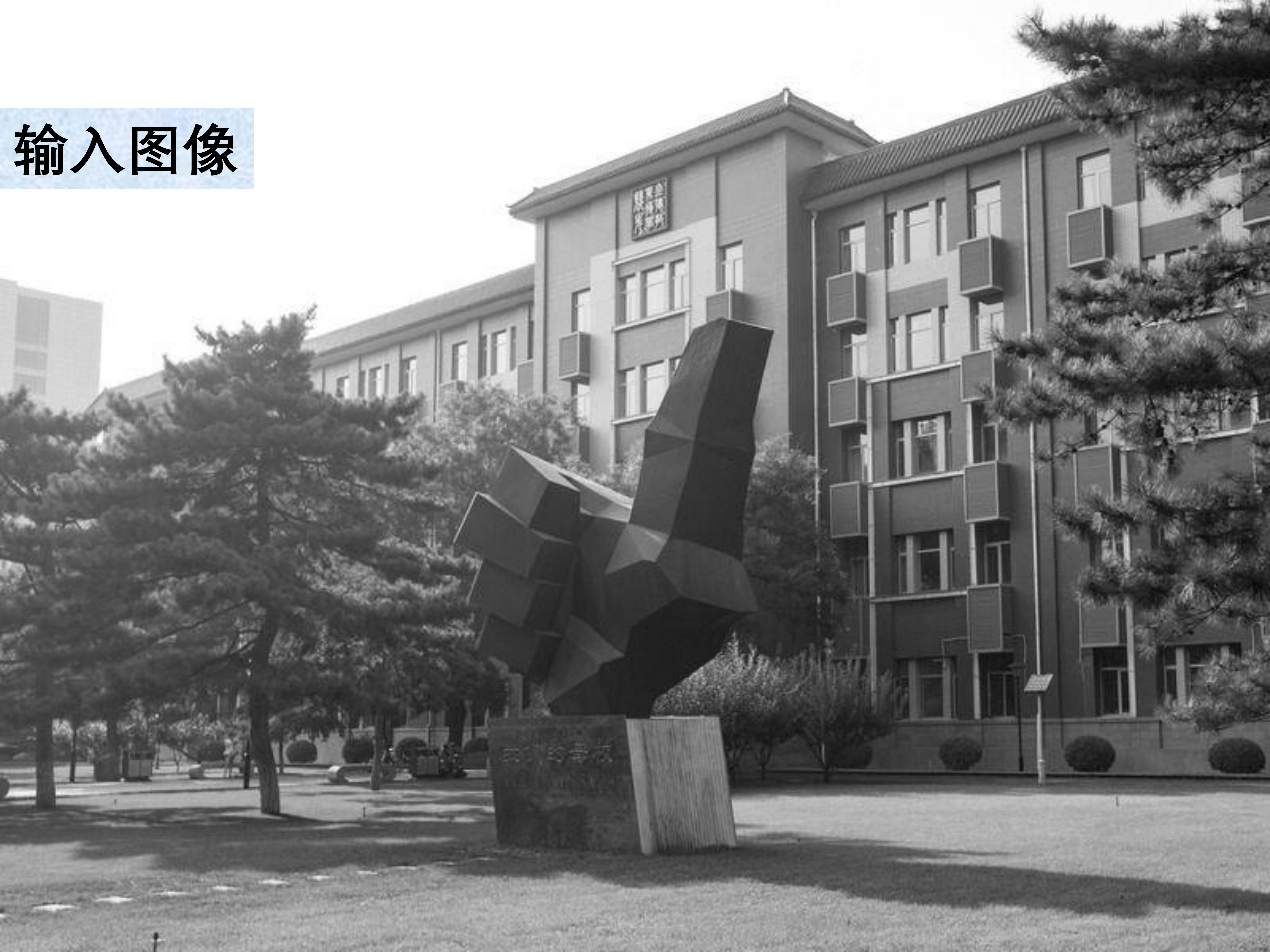
回顾



傅里叶分析

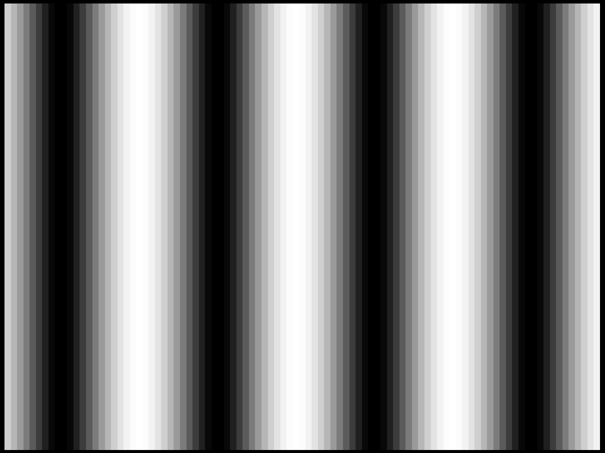
傅里叶合成

## 输入图像

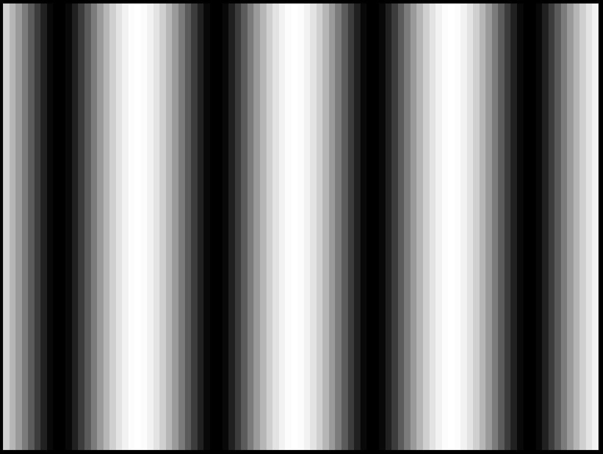


二

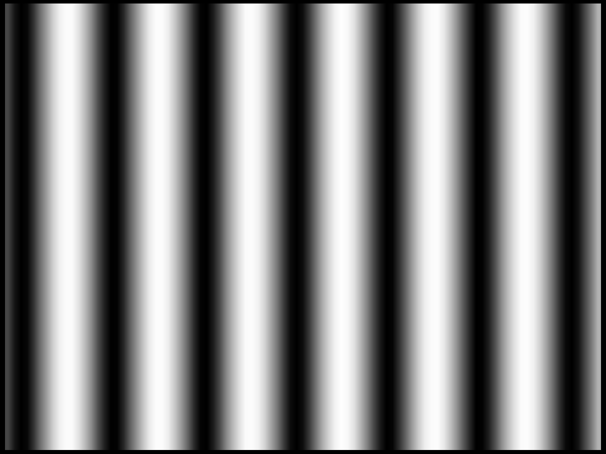
$$=$$

$\alpha_0$ $+$ $\alpha_1$ 

$=$

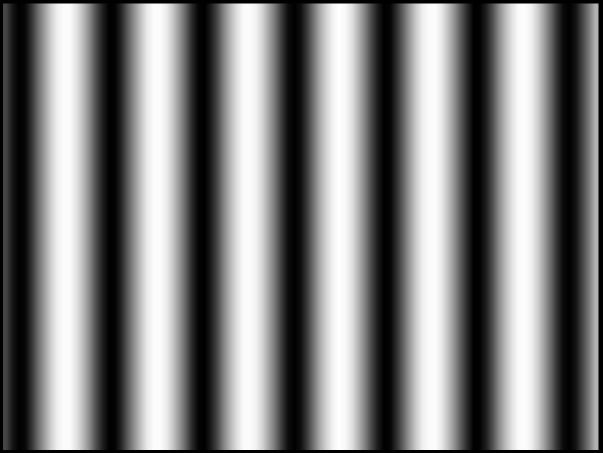


$+ \quad \alpha_2$

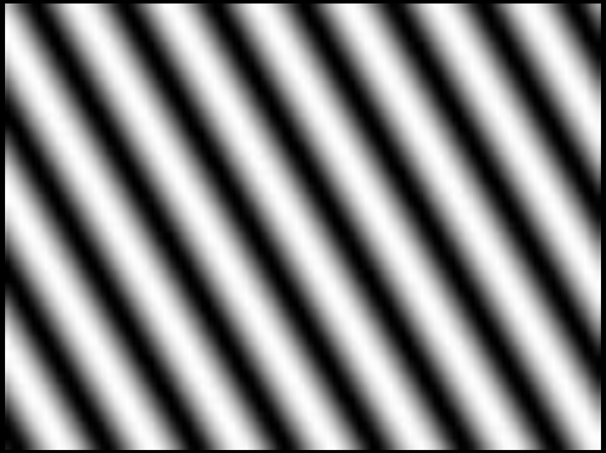


$+$

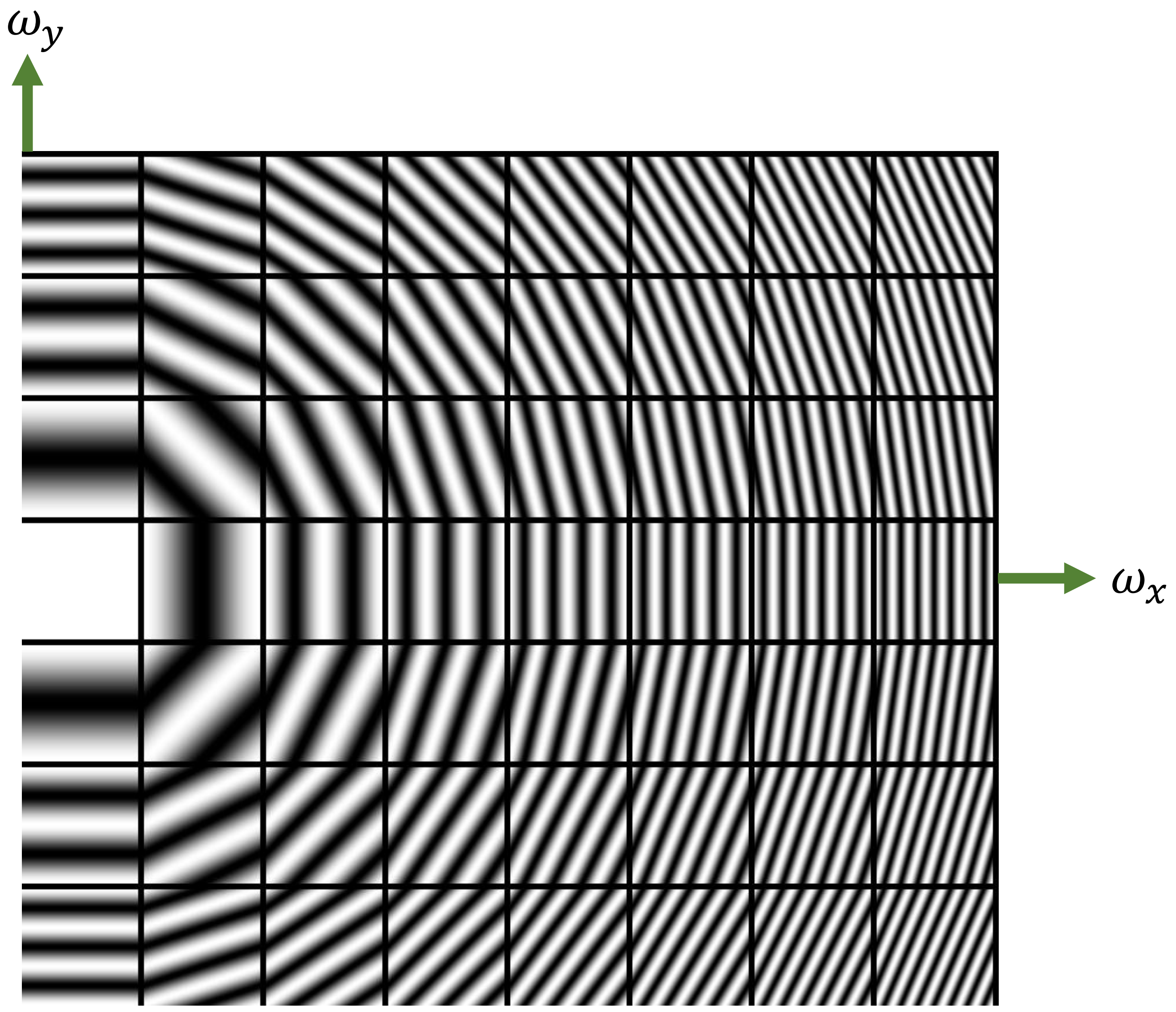
$$=$$

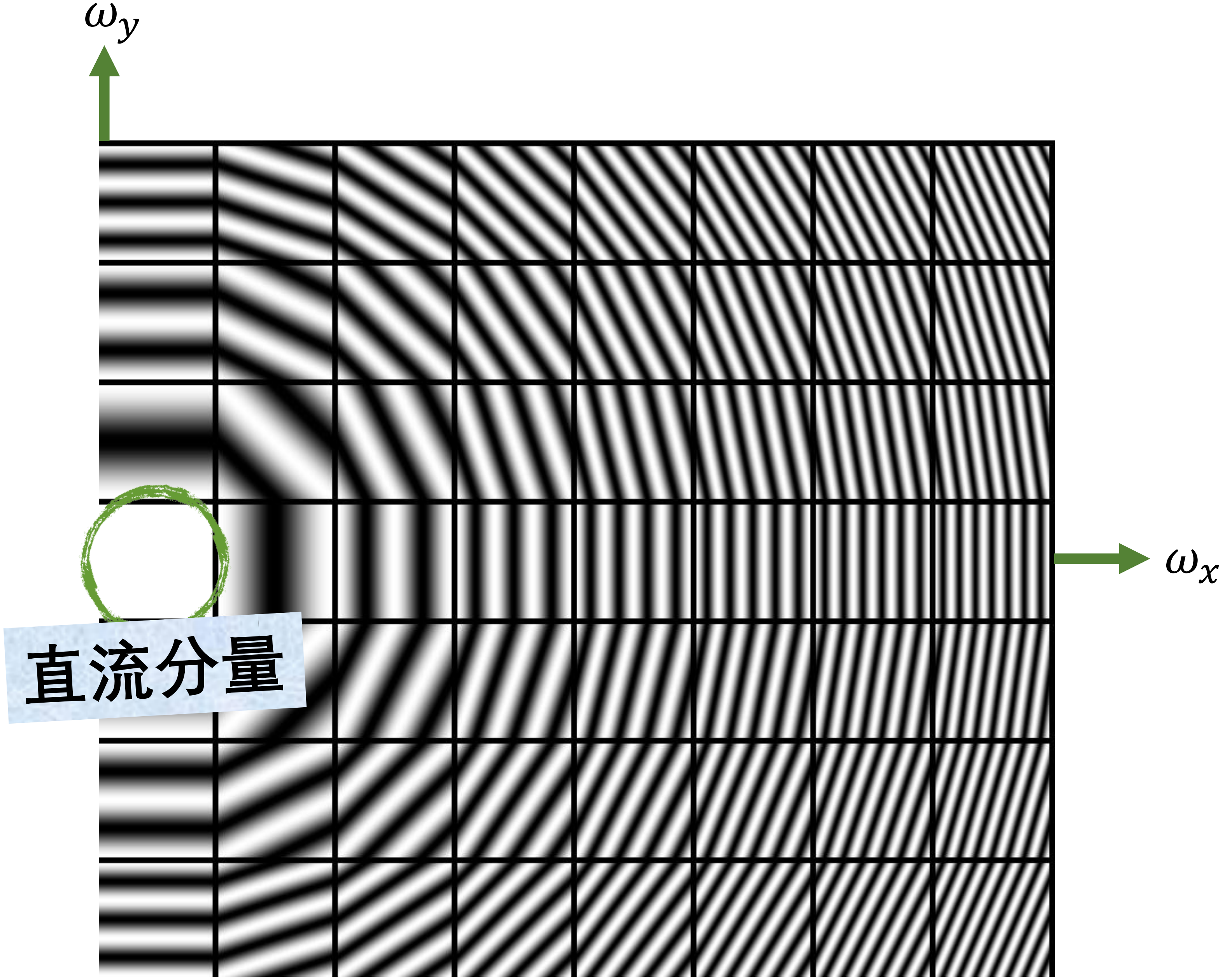


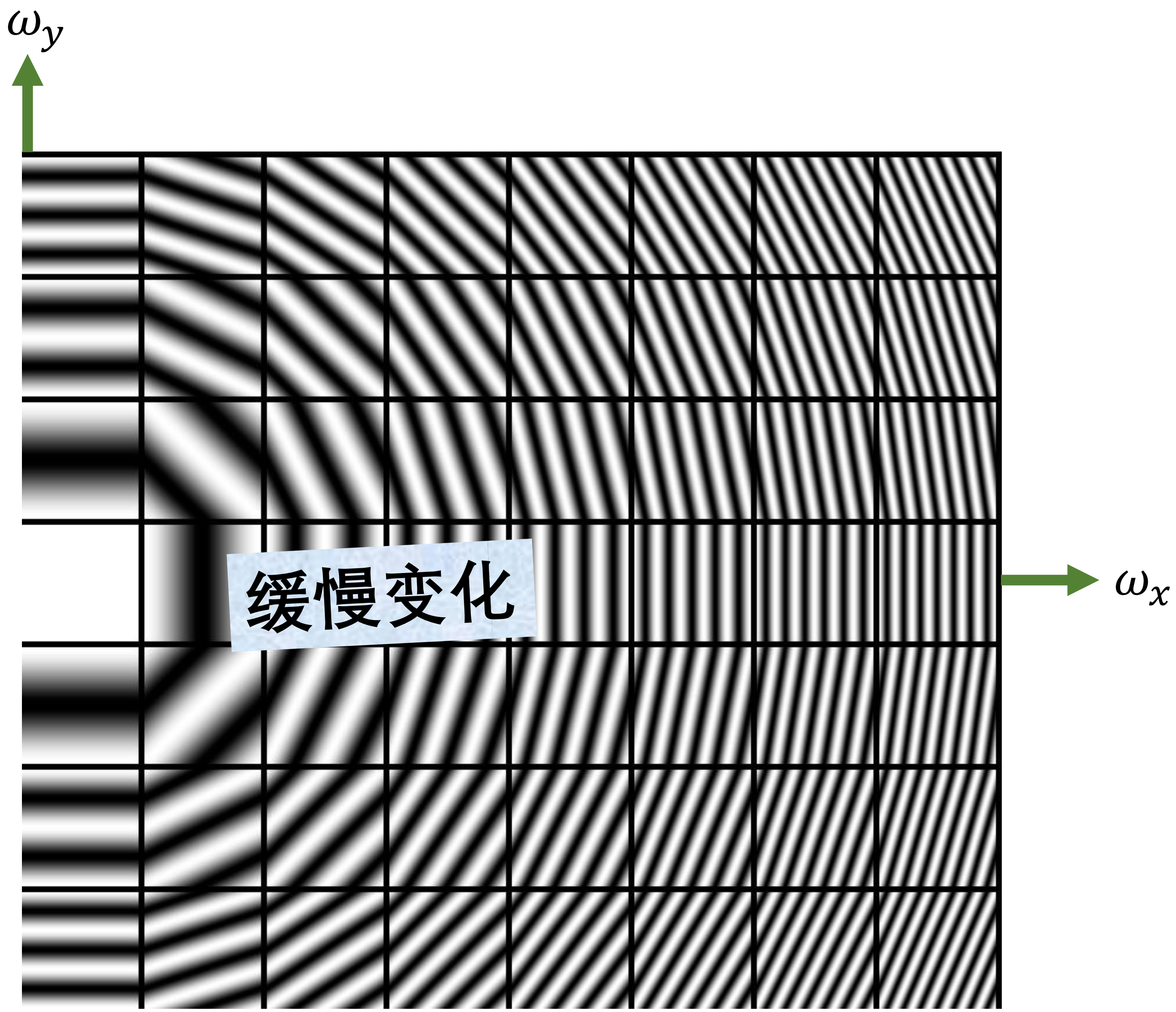
$$+ \quad \alpha_3$$

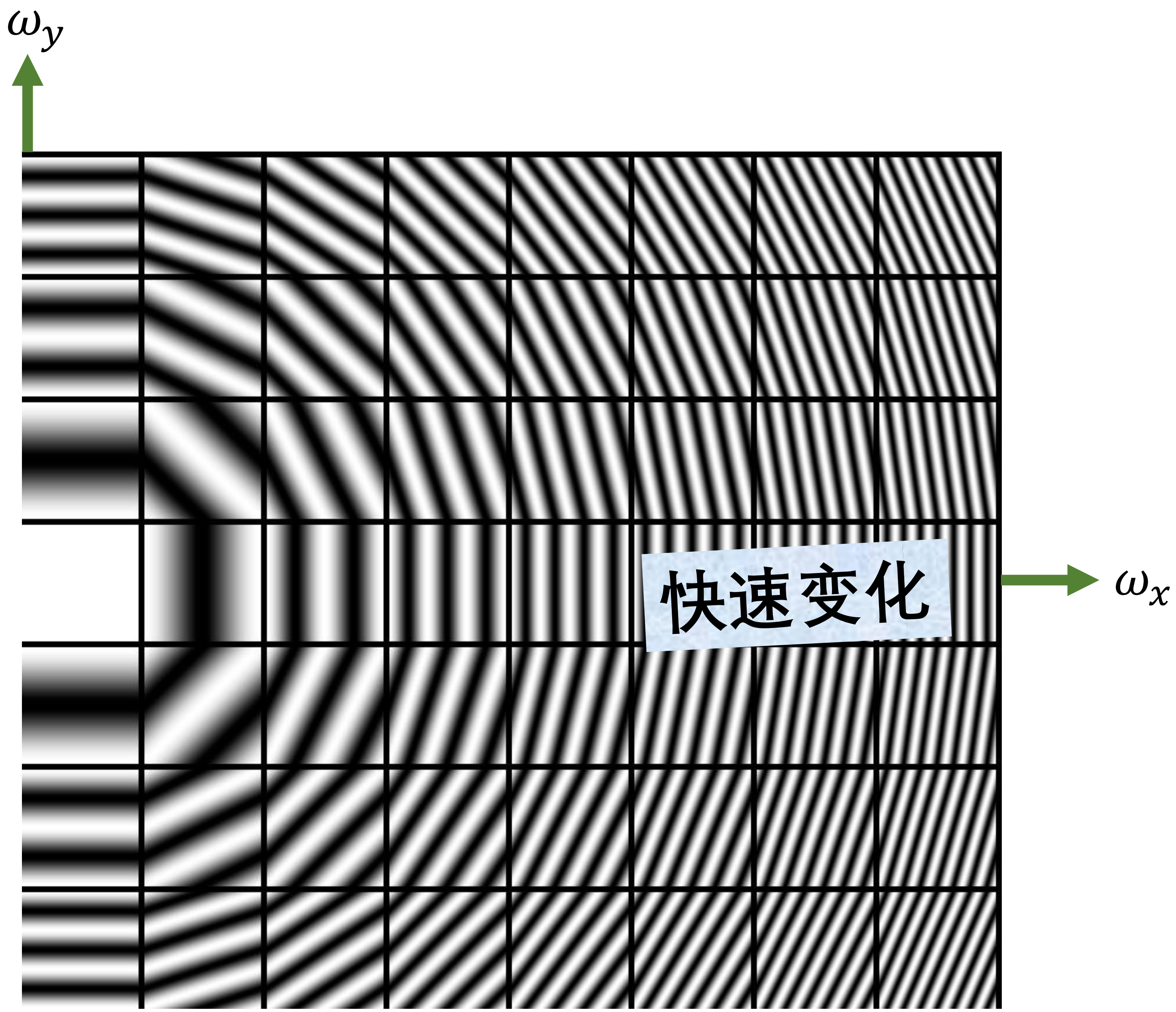


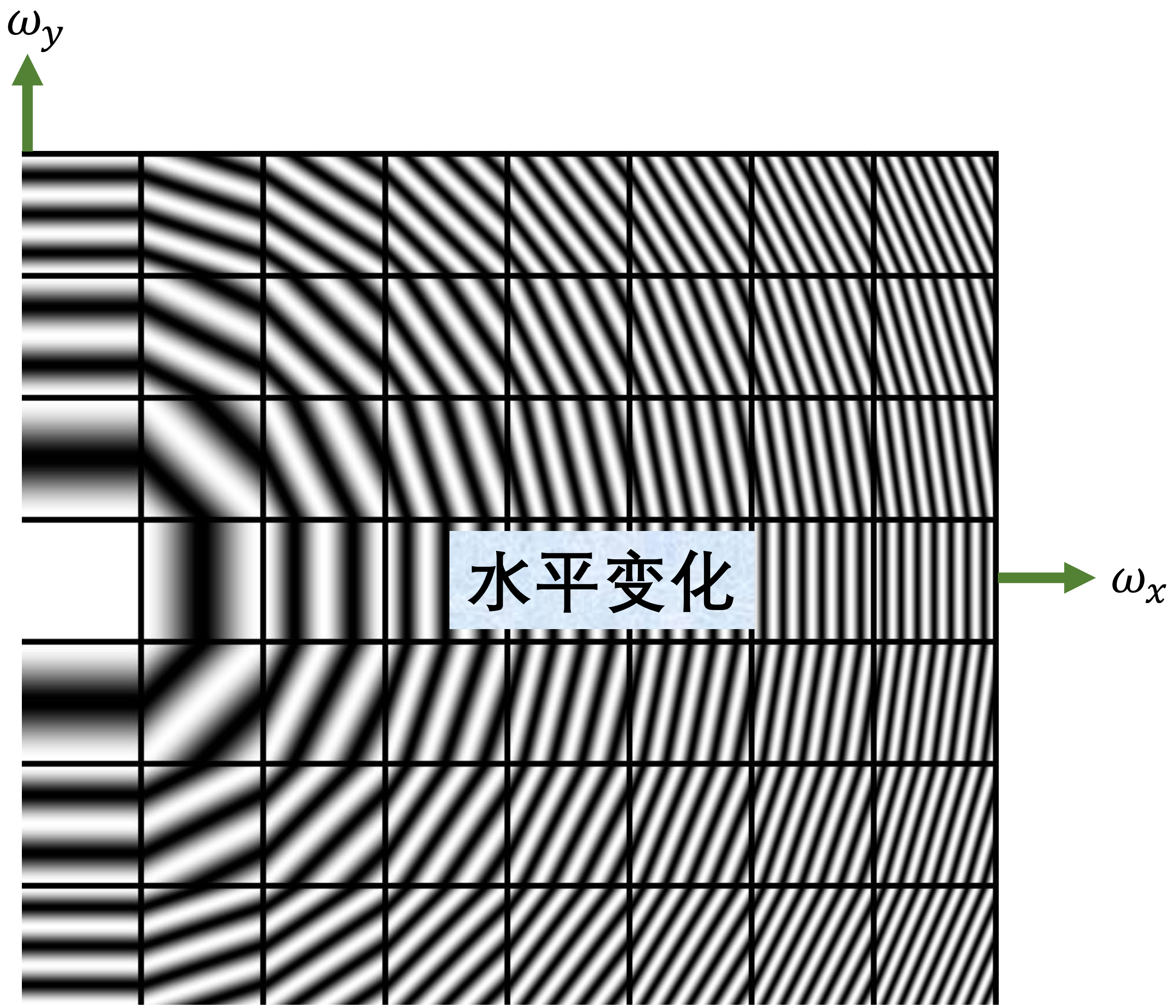
$$+ \quad \cdots$$

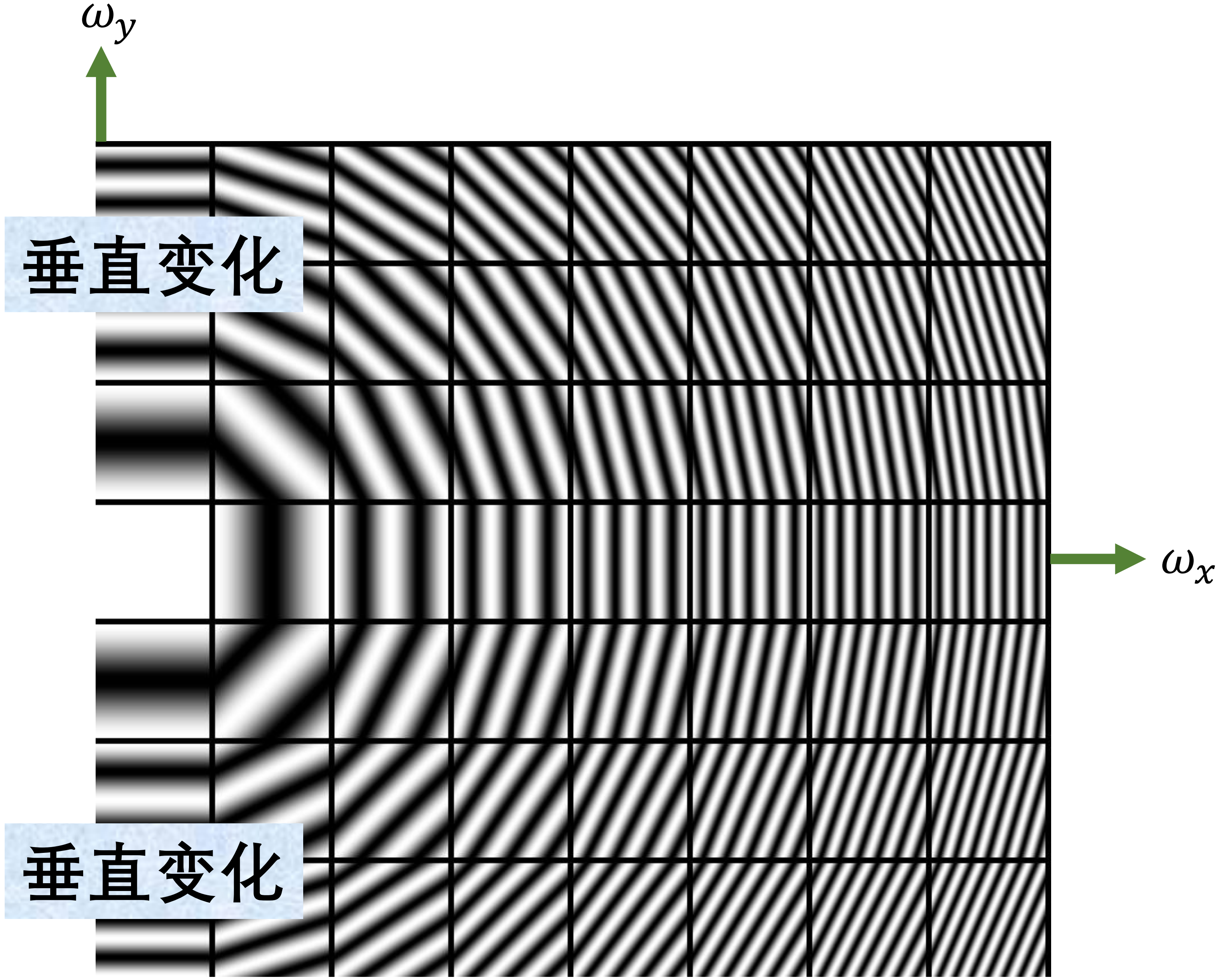

$\omega_y$
$\omega_x$

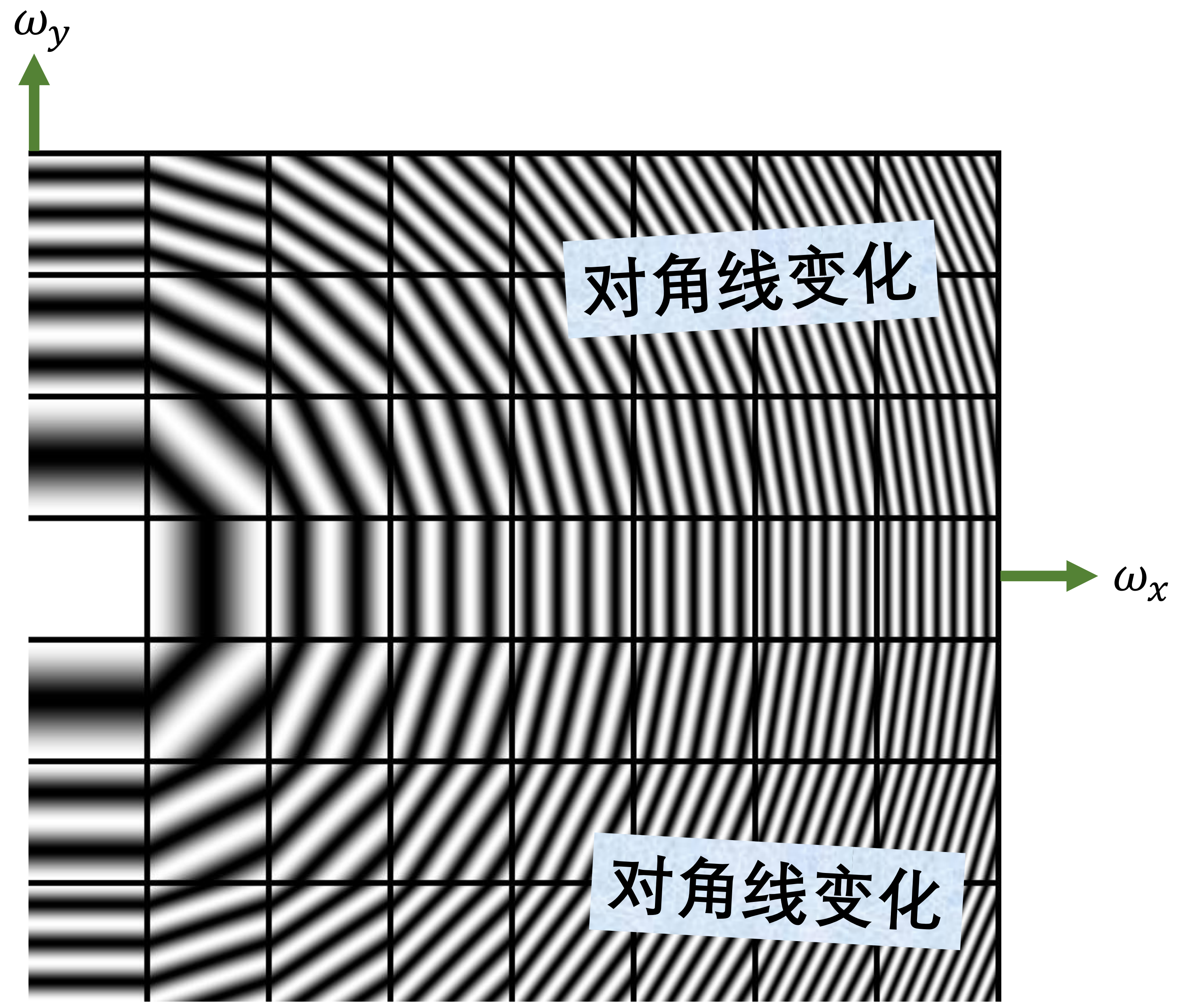

$\omega_y$
$\omega_x$
直流分量


$\omega_y$
$\omega_x$
缓慢变化

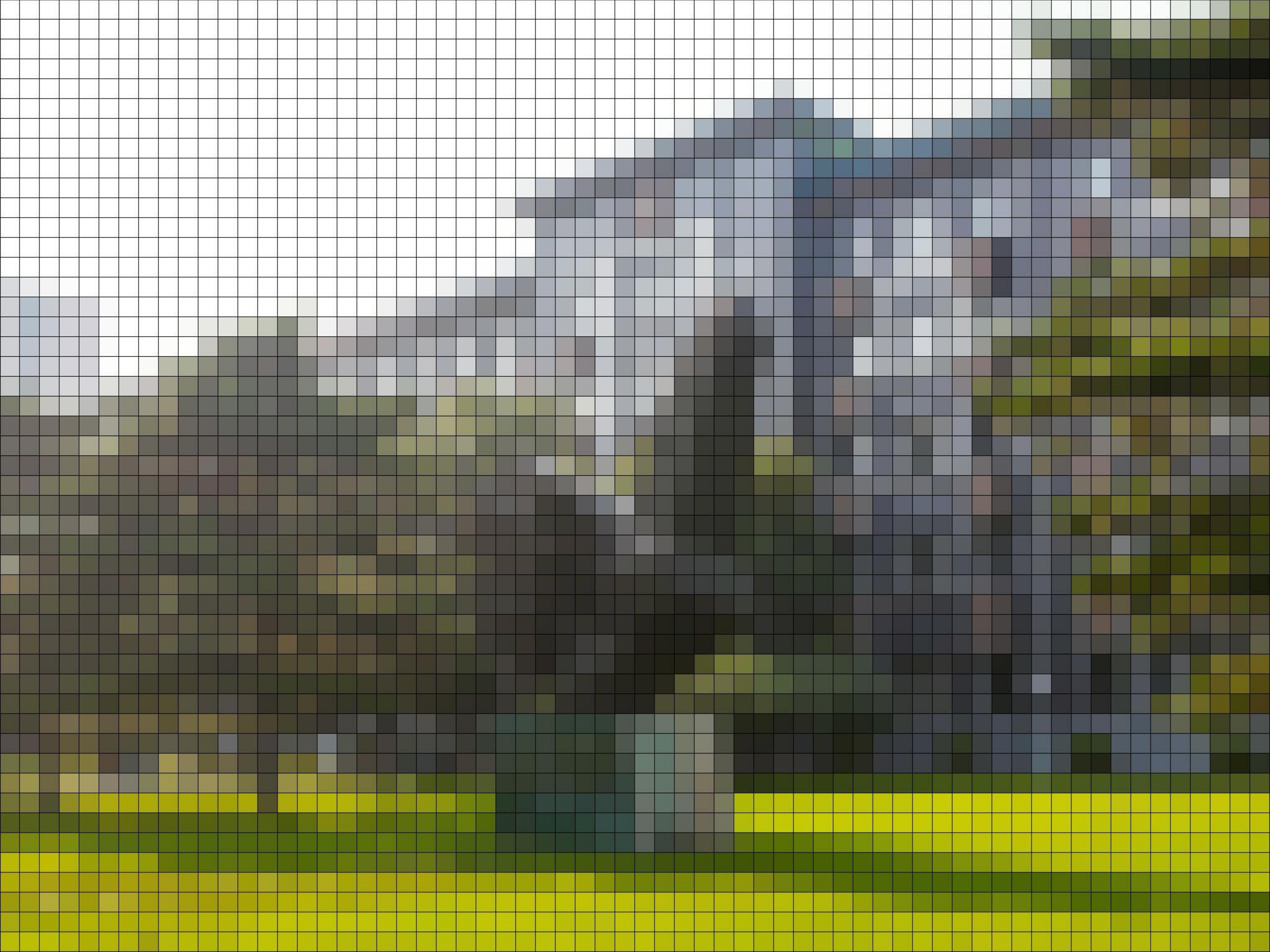

$\omega_y$
快速变化
$\omega_x$


$\omega_y$
水平变化
$\omega_x$


$\omega_y$
垂直变化
垂直变化
$\omega_x$


$\omega_y$
$\omega_x$
对角线变化
对角线变化

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

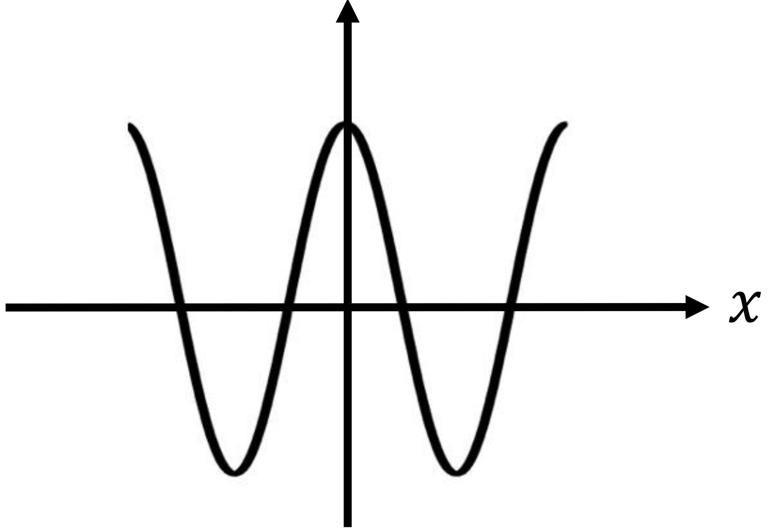
$$f[x,y] = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F[u,v] e^{i2\pi\left(x\frac{u}{M}+y\frac{v}{N}\right)}$$

**其中**

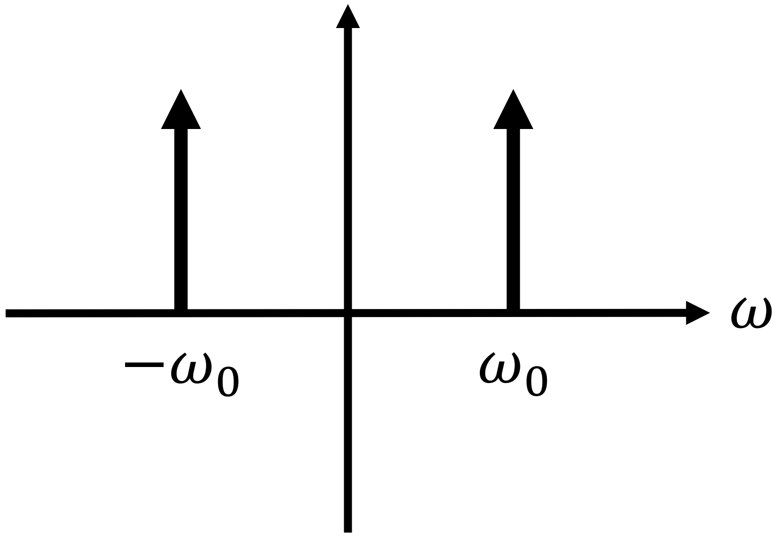
$$u = 0, \ldots, M-1$$
$$v = 0, \ldots, N-1$$

## 傅里叶变换对

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

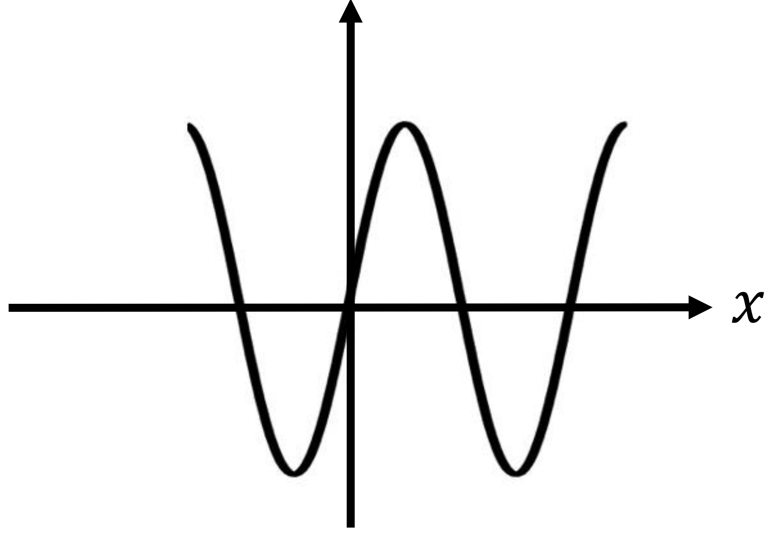


$\frac{1}{2}\left(\delta(\omega-\omega_0)+\delta(\omega+\omega_0)\right)$

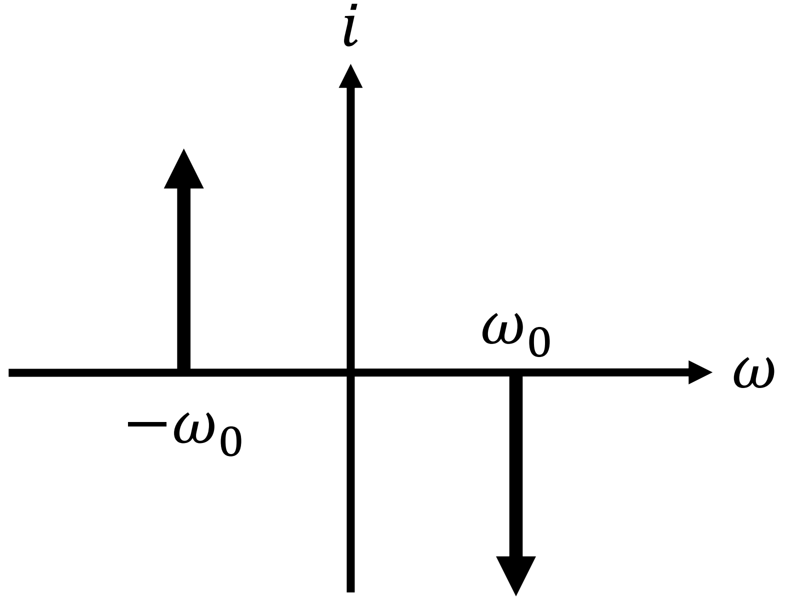


$$\cos(2\pi\omega_0 x)=\frac{1}{2}\left(e^{i2\pi\omega_0 x}+e^{-i2\pi\omega_0 x}\right)$$

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

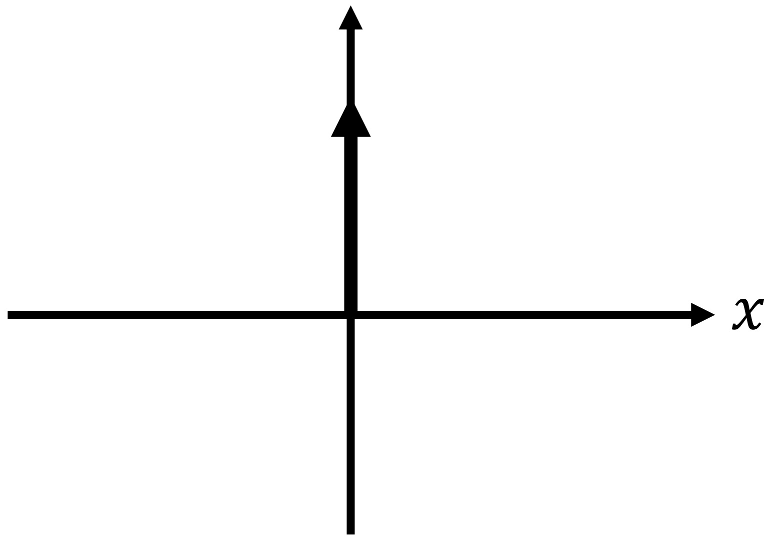


$\frac{1}{2}i\big(\delta(\omega+\omega_0)-\delta(\omega-\omega_0)\big)$

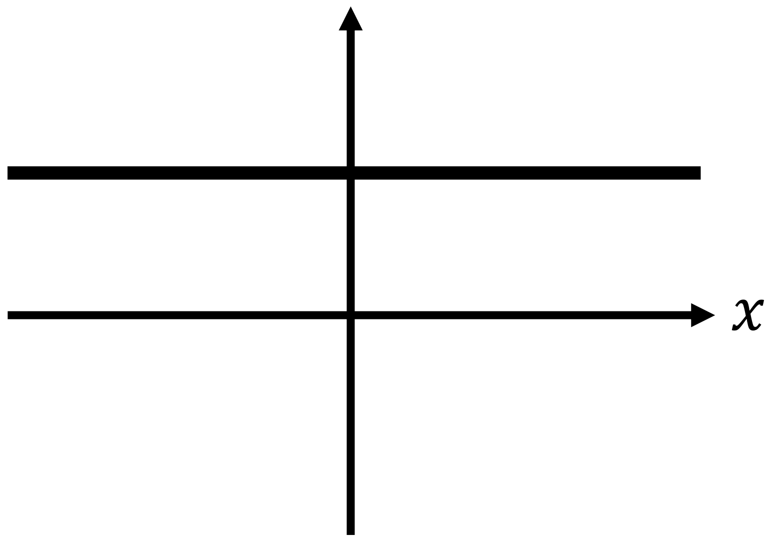


$$\sin(2\pi\omega_0 x) = \frac{1}{2i}\left(e^{i2\pi\omega_0 x} - e^{-i2\pi\omega_0 x}\right)$$

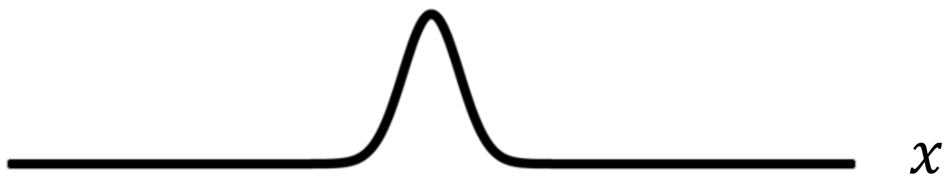
$f(x) = \delta(x)$



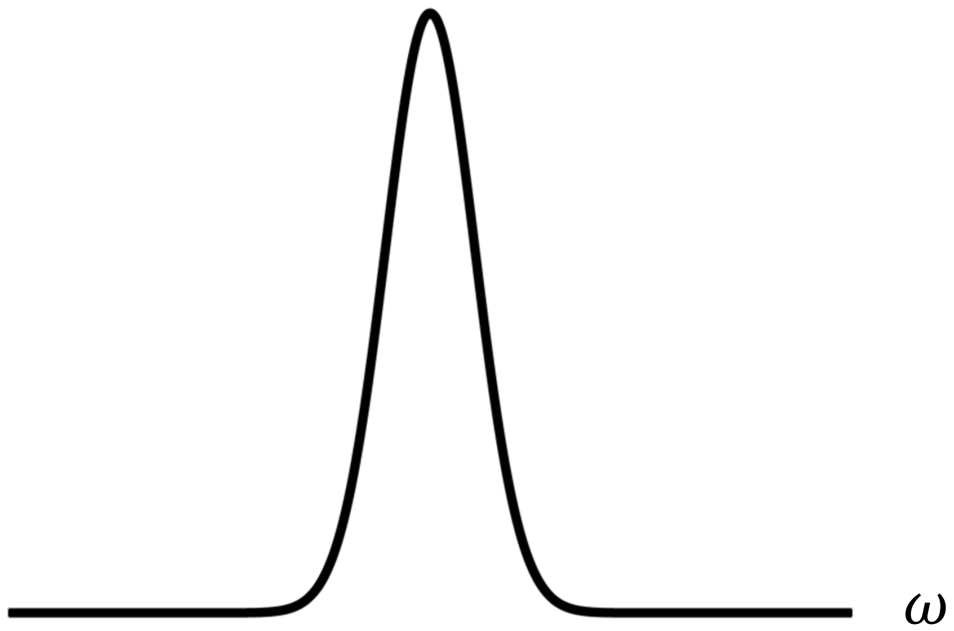
$F(\omega) = 1$



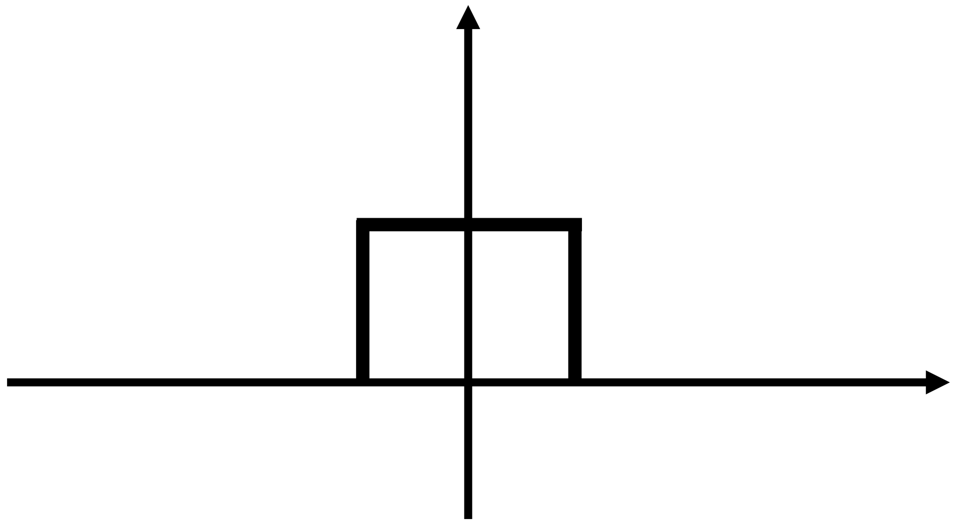
$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$



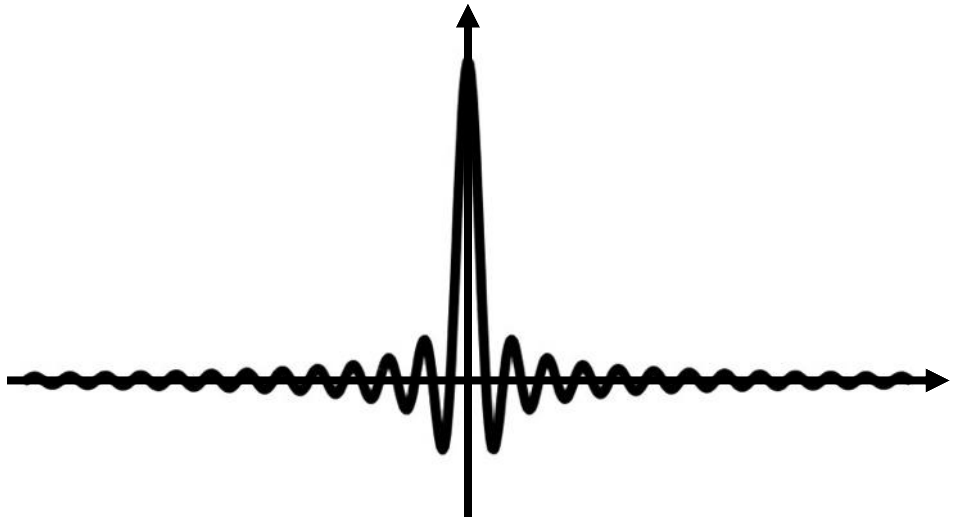
$$G(\omega) = e^{-\frac{(2\pi\omega)^2\sigma^2}{2}}$$



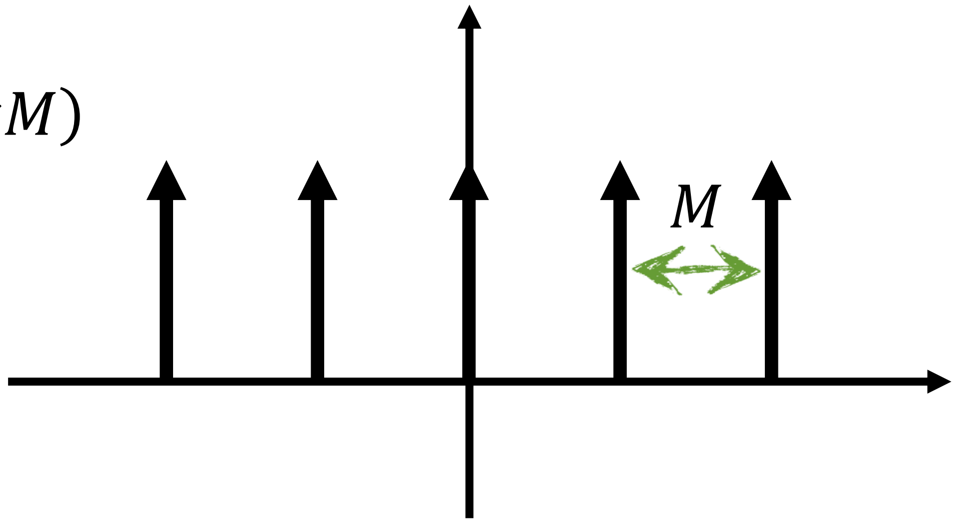
$\mathrm{box}(x)$



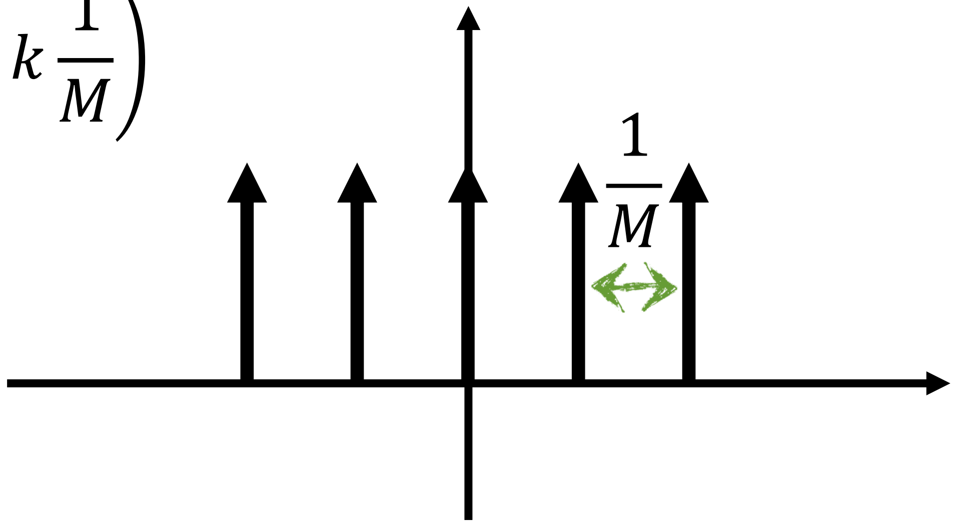
$\mathrm{sinc}(\omega)$



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



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



| | 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 | $f(x) * g(x)$ | $F(\omega) G(\omega)$ |

|  | 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 | $f(x) * g(x)$ | $F(\omega) G(\omega)$ |

## 证明:
## 卷积定理

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

## 证明:
## 卷积定理

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} \left[ \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau \right] e^{-i2\pi\omega x} dx$$

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} \underbrace{\left[ \int_{-\infty}^{\infty} f(\tau) g(x-\tau) d\tau \right]}_{\text{卷积}} e^{-i2\pi\omega x} dx$$

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} \left[ \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau \right] e^{-i2\pi\omega x} dx$$

傅里叶变换

## 证明：卷积定理

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau\right]e^{-i2\pi\omega x}dx$$

改变积分顺序

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} \left[ \int_{-\infty}^{\infty} f(\tau) g(x-\tau) d\tau \right] e^{-i2\pi\omega x} dx$$

改变积分顺序

$$= \int_{-\infty}^{\infty} f(\tau) \left[ \int_{-\infty}^{\infty} g(x-\tau) e^{-i2\pi\omega x} dx \right] d\tau$$

证明： $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} \left[ \int_{-\infty}^{\infty} f(\tau) g(x-\tau) d\tau \right] e^{-i2\pi\omega x} dx$$

改变积分顺序

$$= \int_{-\infty}^{\infty} f(\tau) \left[ \underline{\int_{-\infty}^{\infty} g(x-\tau) e^{-i2\pi\omega x} dx} \right] d\tau$$

**证明：** $F\{f(x)*g(x)\}(\omega)=F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f*g\}(\omega)$$

$$=\int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty}f(\tau)g(x-\tau)d\tau\right]e^{-i2\pi\omega x}dx$$

改变积分顺序

$$=\int_{-\infty}^{\infty}f(\tau)\left[\int_{-\infty}^{\infty}g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

**看着眼熟吗?**

## 傅里叶变换性质

|  | 空间域 | 频域 |
|---|---|---|
| **移位** | $f(x - x_0)$ | $e^{-i2\pi\omega x_0}F(\omega)$ |

证明: 卷积定理

**证明:** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau\right]e^{-i2\pi\omega x}dx$$

改变积分顺序

$$= \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

**看着眼熟吗?**

**证明：** $F\{f(x)*g(x)\}(\omega)=F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f*g\}(\omega)$$

$$=\int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty}f(\tau)g(x-\tau)d\tau\right]e^{-i2\pi\omega x}dx$$

改变积分顺序

$$=\int_{-\infty}^{\infty}f(\tau)\left[\int_{-\infty}^{\infty}g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

**看着眼熟吗？**

**移位信号的傅立叶变换**

证明：  $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau\right]e^{-i2\pi\omega x}dx$$

改变积分顺序

$$= \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

**看着眼熟吗？**

$$e^{-i2\pi\omega\tau}G(\omega)$$

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

移位定理

$$e^{-i2\pi\omega\tau}G(\omega)$$

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

移位定理

$$e^{-i2\pi\omega\tau}G(\omega)$$

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

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

移位定理

$$e^{-i2\pi\omega\tau}G(\omega)$$

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

提出公因子

**证明：** $F\{f(x)*g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f*g\}(\omega)$$

$$= \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

移位定理

$$e^{-i2\pi\omega\tau}G(\omega)$$

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

提出公因子

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

## 证明: 卷积定理

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

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

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

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

证明： $F\{f(x)*g(x)\}(\omega)=F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f*g\}(\omega)$$

$$=\int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

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

看着眼熟吗？

**证明：** $F\{f(x)*g(x)\}(\omega)=F(\omega)G(\omega)$

证明:
卷积定理

$$F\{f*g\}(\omega)$$

$$=\int_{-\infty}^{\infty} f(\tau)\left[\int_{-\infty}^{\infty} g(x-\tau)e^{-i2\pi\omega x}dx\right]d\tau$$

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

**看着眼熟吗？**

**傅里叶变换**

**证明：** $F\{f(x) * g(x)\}(\omega) = F(\omega)G(\omega)$

证明:
卷积定理

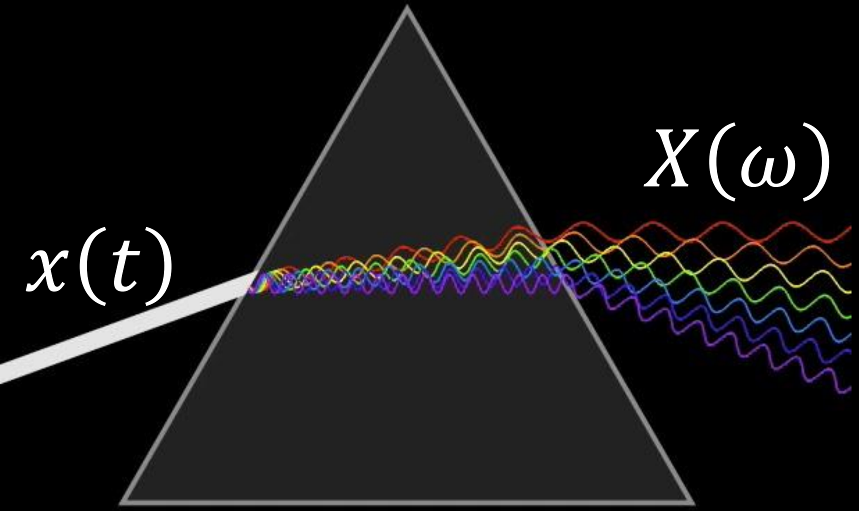
$$F\{f * g\}(\omega)$$

$$= \int_{-\infty}^{\infty} f(\tau) \left[ \int_{-\infty}^{\infty} g(x-\tau) e^{-i2\pi\omega x} dx \right] d\tau$$

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

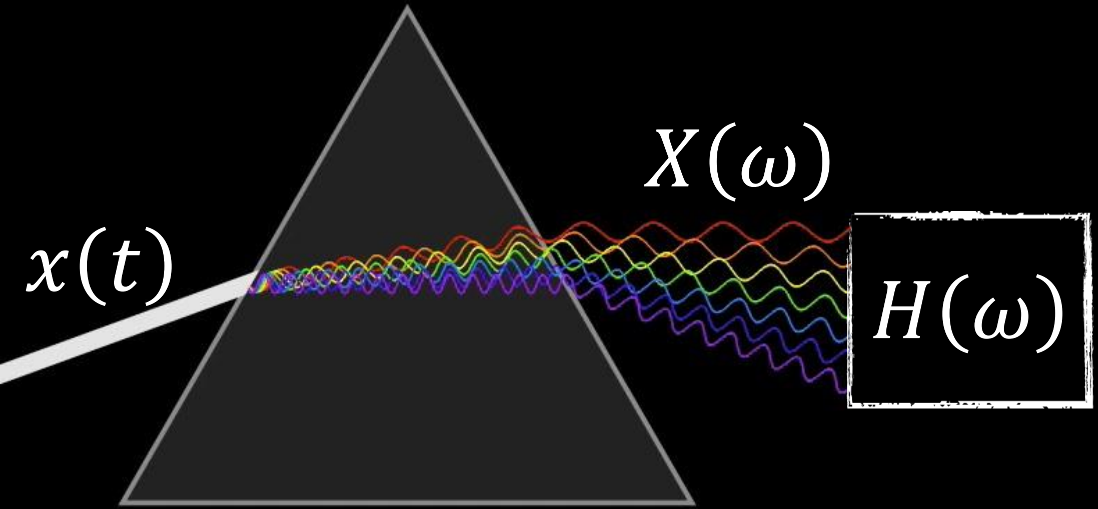
$$= F(\omega)G(\omega)$$

卷积定理



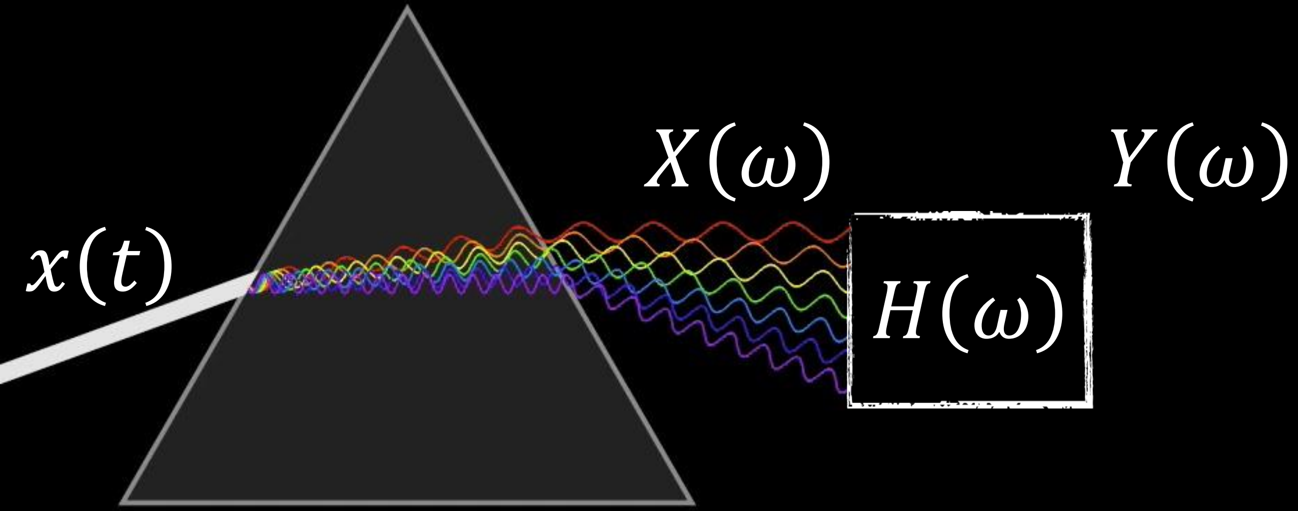
傅里叶分析

卷积定理



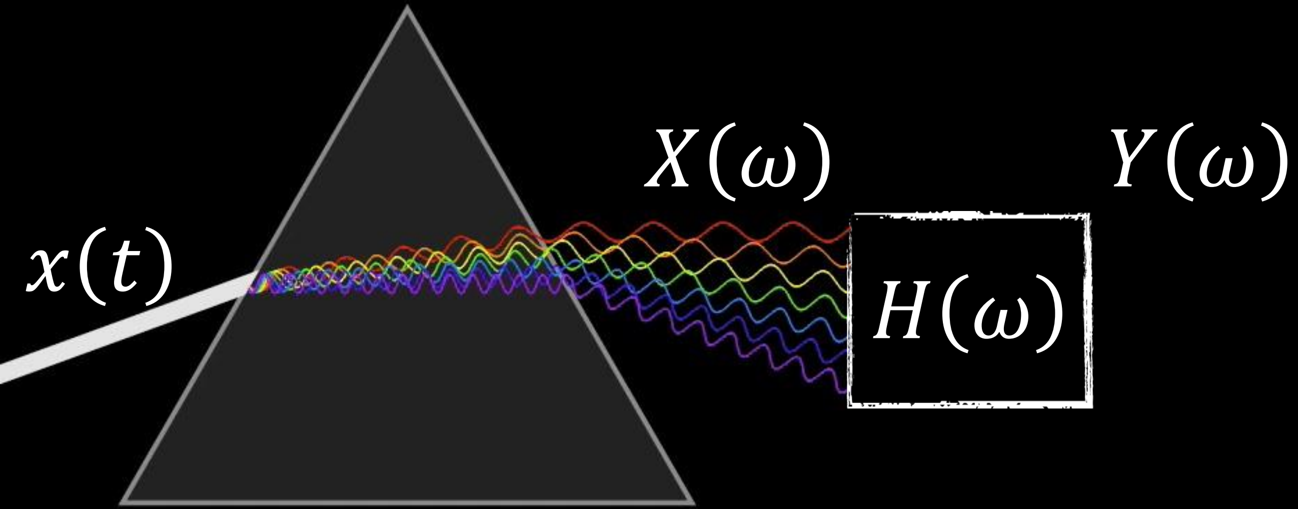
傅里叶分析

卷积定理



傅里叶分析

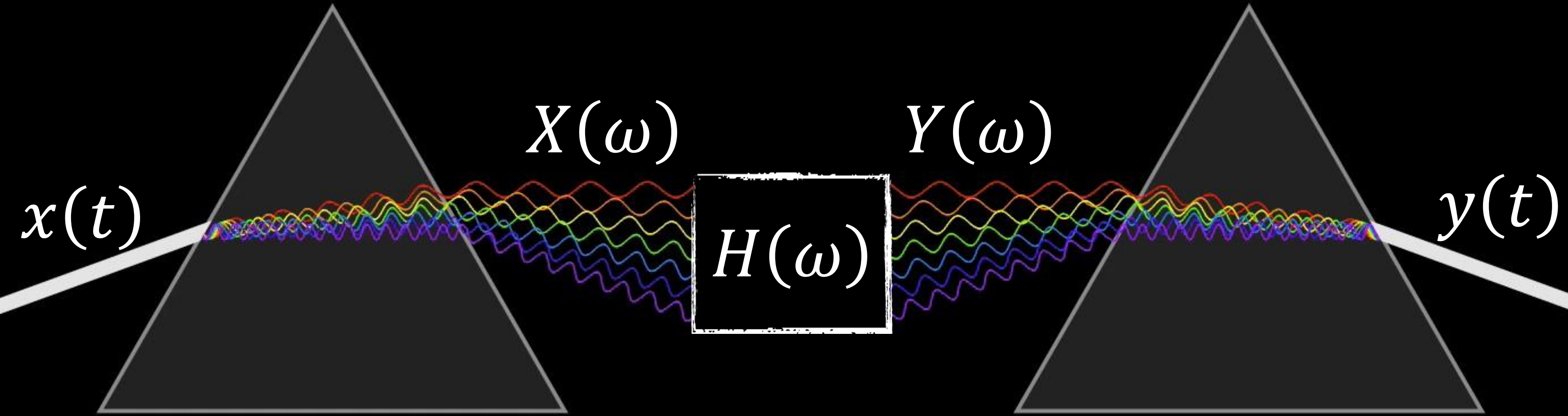
卷积定理



傅里叶分析

$$Y(\omega) = H(\omega)X(\omega)$$

卷积定理



傅里叶分析

傅里叶合成

$$Y(\omega) = H(\omega)X(\omega)$$

$$O\left(N^2\right)$$

卷积的时间复杂度

$$O(N \log N)$$

快速傅里叶变换的时间复杂度

## 卷积定理

空间域 $f \quad * \quad g \quad = \quad h$

## 卷积定理

$$\begin{tikzcd}
\text{空间域} & f \arrow[d, "\mathrm{FT}"] & * & g \arrow[d, "\mathrm{FT}"] & = & h \\
& {} & & {} & &
\end{tikzcd}$$

## 卷积定理

$$\begin{tikzcd}
\text{空间域} & f \arrow[d, "\mathrm{FT}"] & * & g \arrow[d, "\mathrm{FT}"] & = & h \\
\text{频域} & F & & G & &
\end{tikzcd}$$

## 卷积定理

$$\begin{tikzcd} \text{空间域} & f \arrow[d, "\text{FT}"] & * & g \arrow[d, "\text{FT}"] & = & h \\ \text{频域} & F & \times & G & & \end{tikzcd}$$

## 卷积定理

$$\begin{tikzcd} \text{空间域} & f \arrow[d, "\text{FT}"] & * & g \arrow[d, "\text{FT}"] & = & h \\ \text{频域} & F & \times & G & = & H \end{tikzcd}$$

## 卷积定理

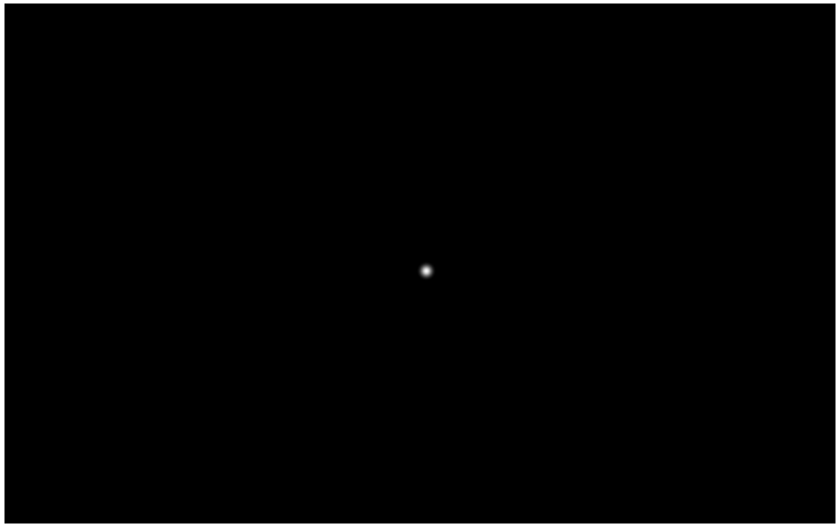
$$\begin{tikzcd}
\text{空间域} & f \arrow[d, "\text{FT}"] & * & g \arrow[d, "\text{FT}"] & = & h \\
\text{频域} & F & \times & G & = & H \arrow[u, "\text{IFT}"']
\end{tikzcd}$$

$O(N^2)$ vs. $O(N\log N)$

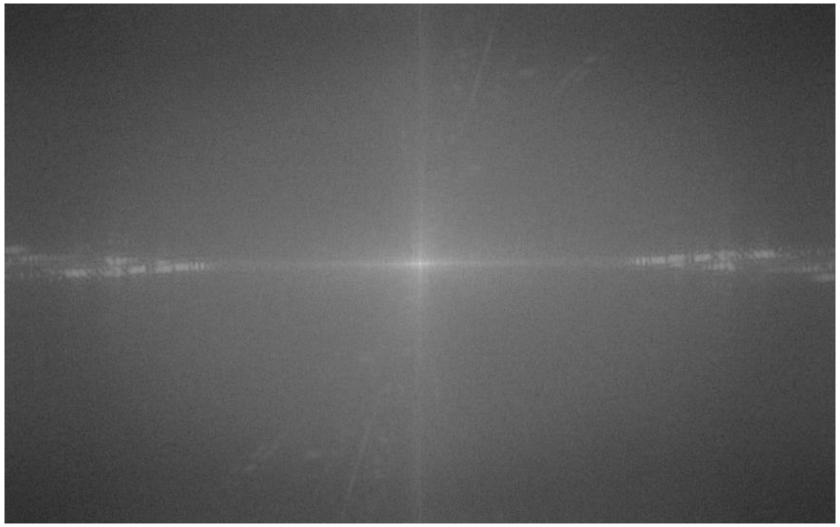
卷积的时间复杂度　　基于FFT的卷积时间复杂度

## 空间域

## 卷积定理实例

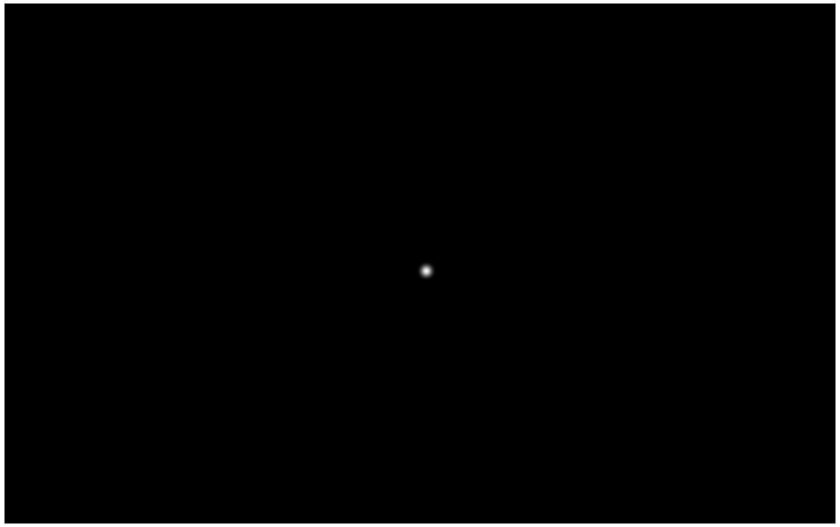


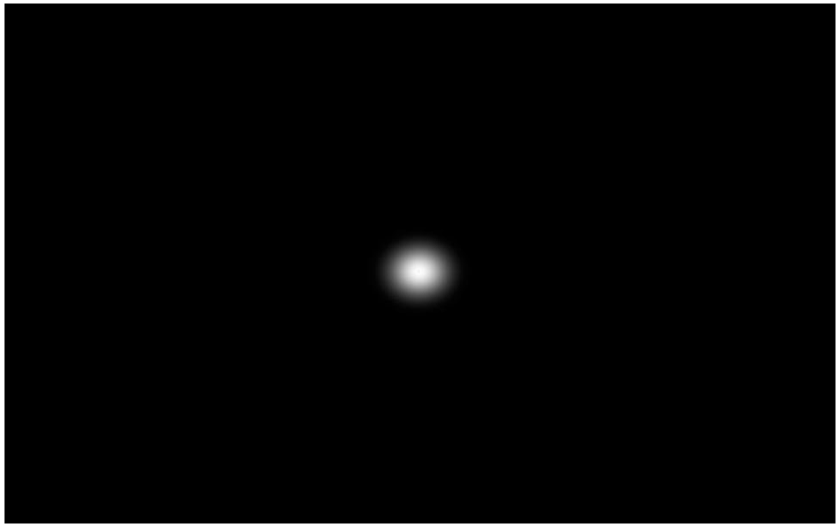
空间域 频域



*

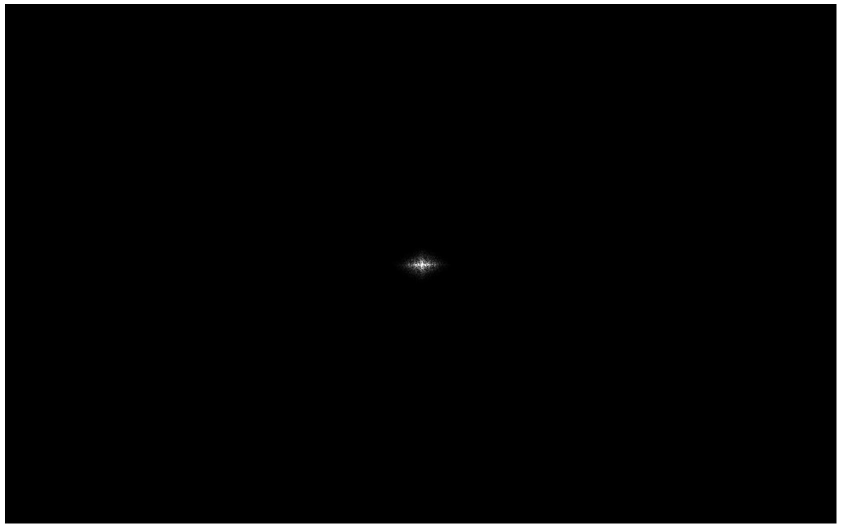
×





空间域

频域



Python时间

## 卷积定理

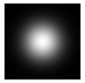
$$\begin{tikzcd}
\text{空间域} & f \arrow[d, "\text{FT}"] & * & g \arrow[d, "\text{FT}"] & = & h \\
\text{频域} & F & \times & G & = & H \arrow[u, "\text{IFT}"]
\end{tikzcd}$$

## 频域滤波

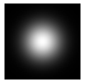
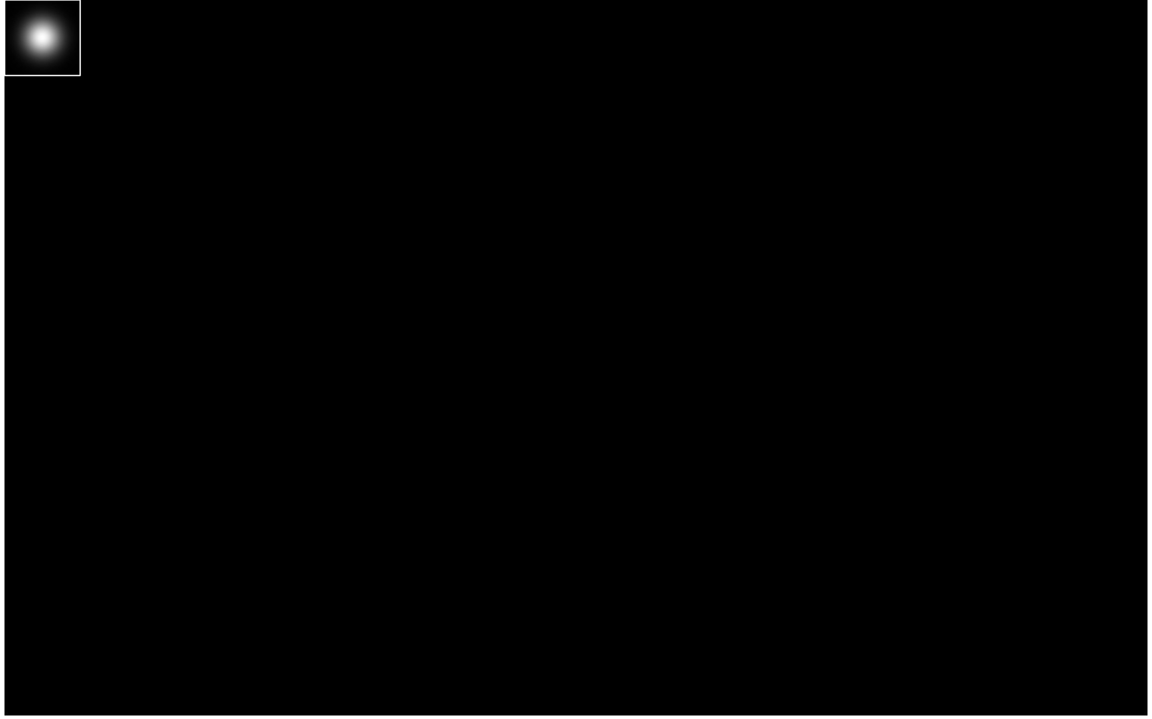
```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCA
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                                cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                        cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

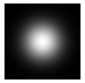
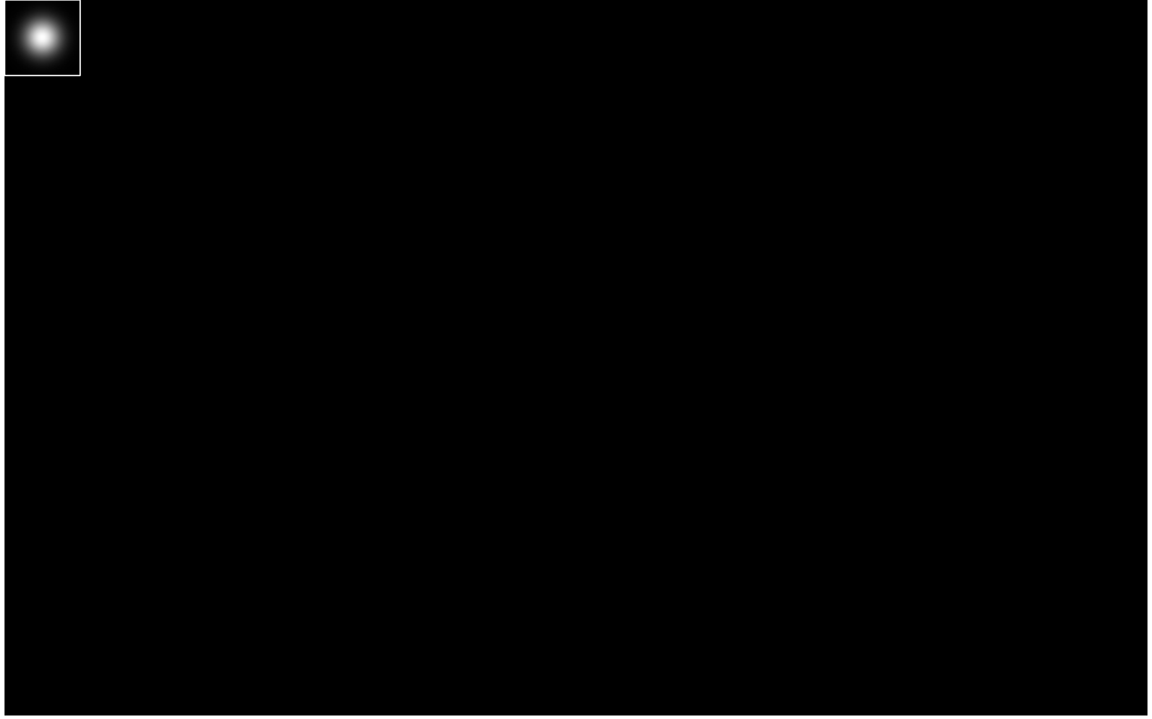
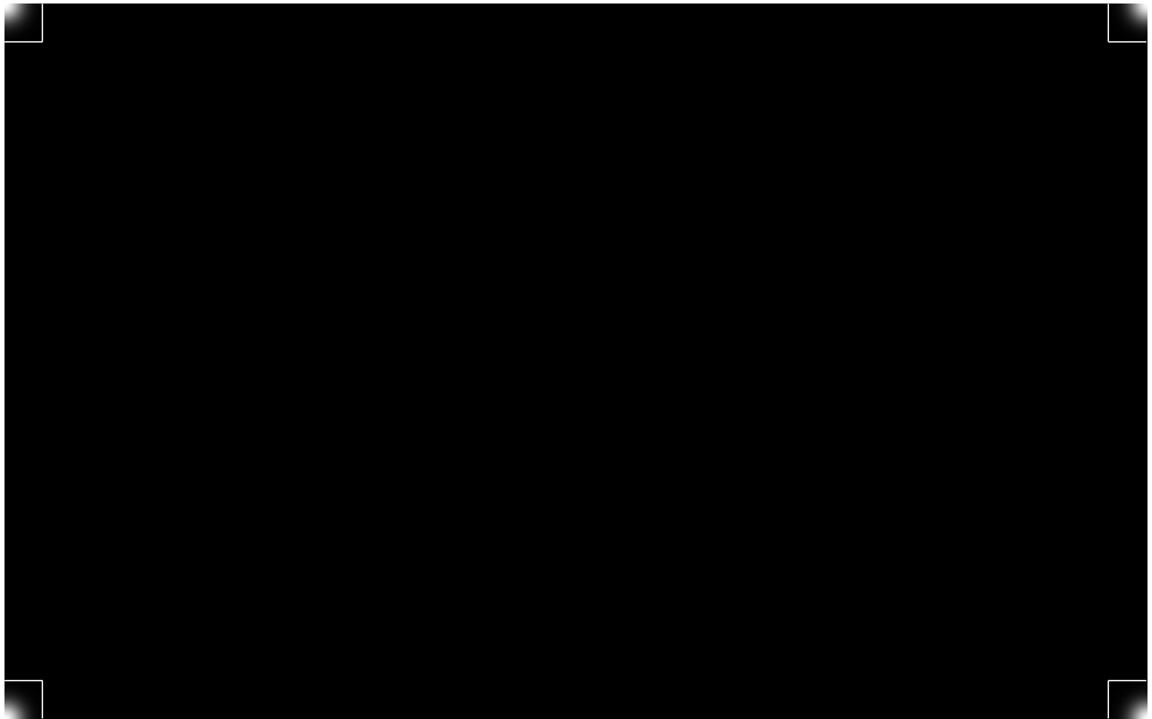
```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                               cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```



```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                              cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                        cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

## 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**

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

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                     cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                          cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

**DFT假设：图像信号是周期的**

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

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

                    (1, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                     cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

**DFT假设：图像信号是周期的**

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                          cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)
```

DFT

```
im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                     cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                                cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                     cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

逆DFT

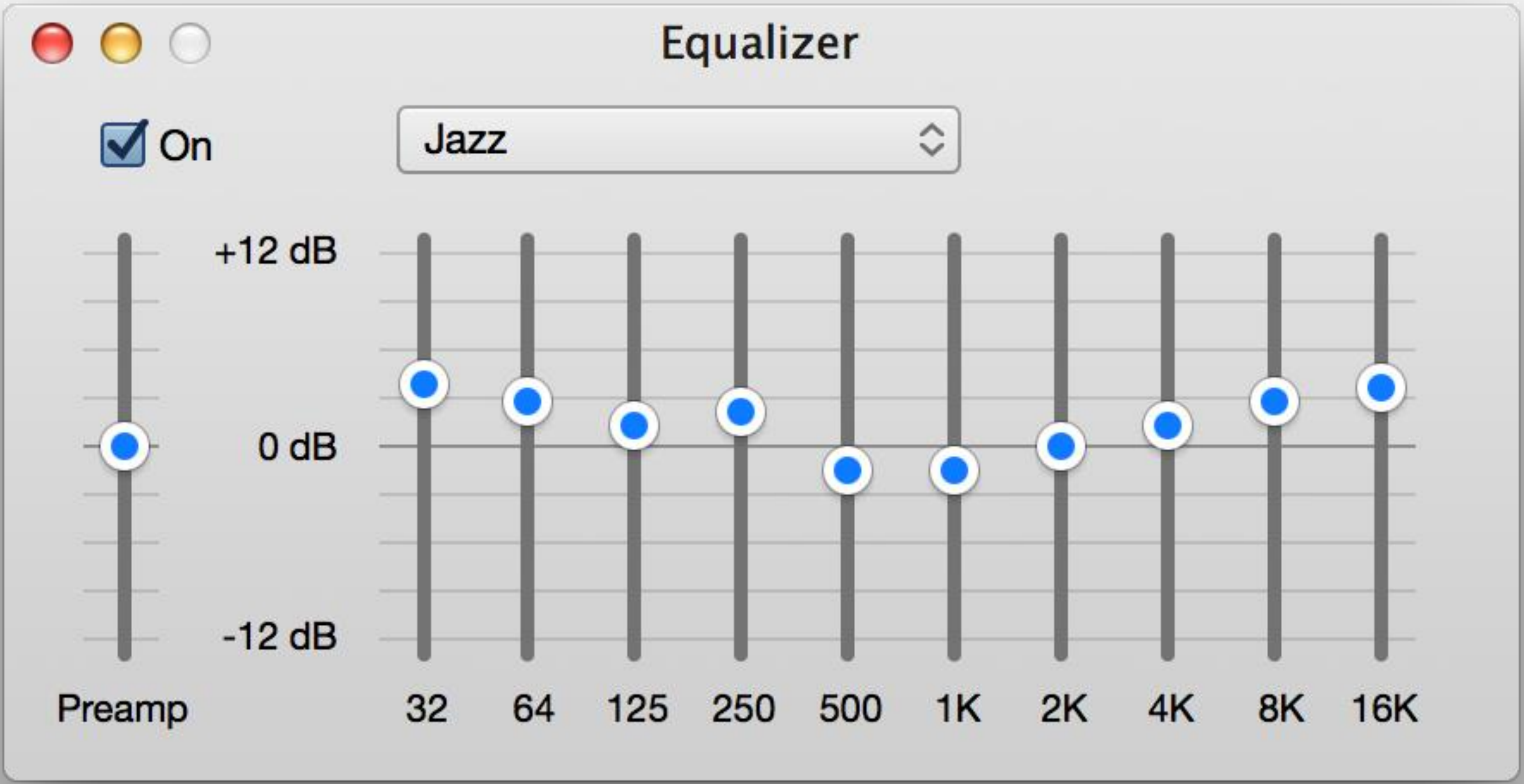
```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCALE)
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                         cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

## 频域滤波

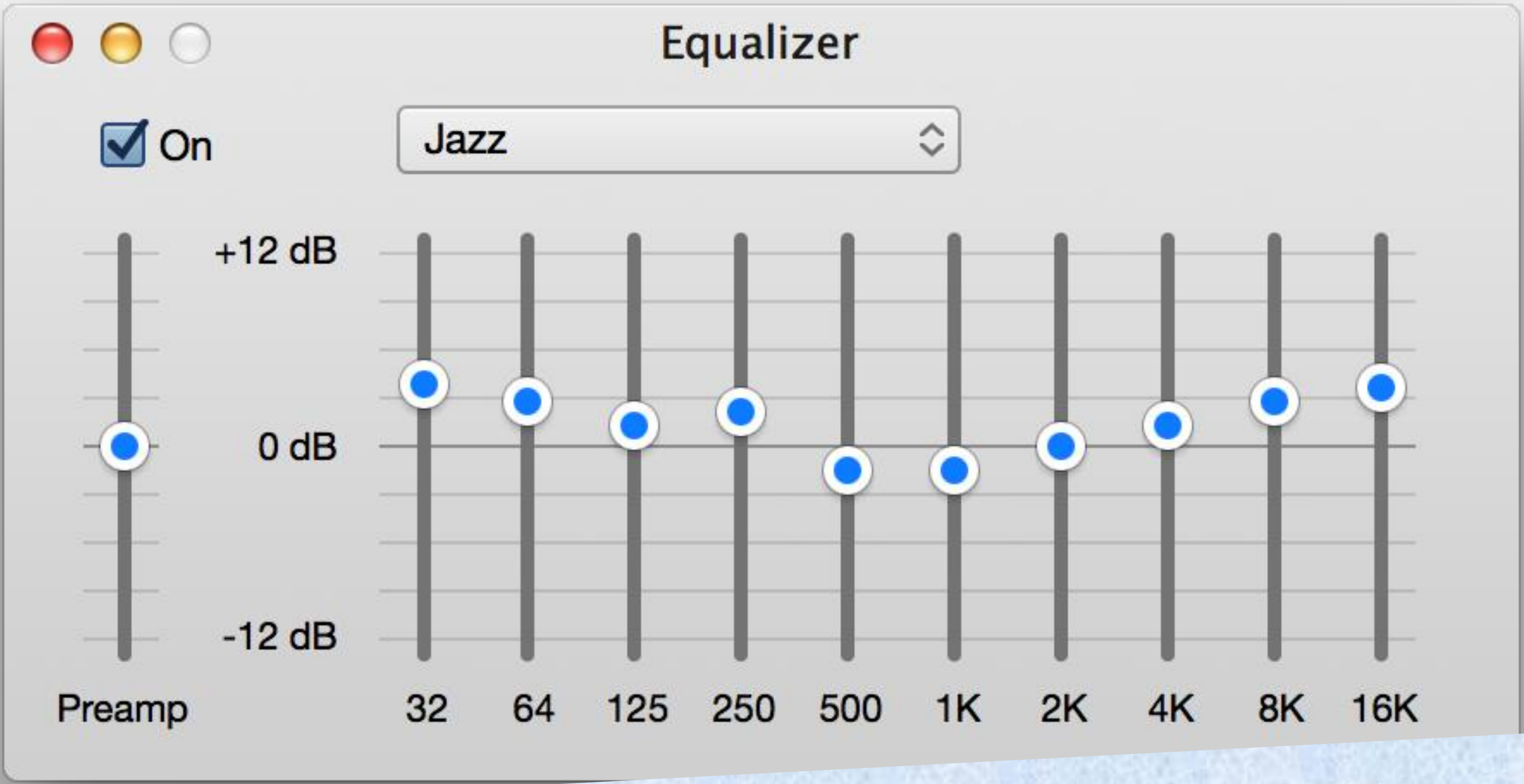
```
im = cv2.imread('colosseum.jpg', cv2.IMREAD_GRAYSCA
im = im.astype(float)/255

sigma = 10
width = 2*sigma*3+1
f = cv2.getGaussianKernel(width, sigma)
f = f @ f.T
f = np.pad(f, ((0, im.shape[0] - width), (0, im.shape[1] - width)))
f = np.roll(f, shift=-3*sigma, axis=(0, 1))

im_dft = np.fft.fft2(im, im.shape)
f_dft = np.fft. fft2(f, im.shape)

im_f_dft = im_dft * f_dft
im_f = np.real(np.fft.ifft2(im_f_dft))

im_f = cv2.normalize(im_f, None, 0, 255,
                                cv2.NORM_MINMAX, dtype=cv2.CV_8U)
cv2.imshow('Frequency Domain Filtering', im_f), cv2.waitKey(0)
```

# Python时间

已结束


Equalizer
On
+12 dB
0 dB
-12 dB
Preamp
32
64
125
250
500
1K
2K
4K
8K
16K

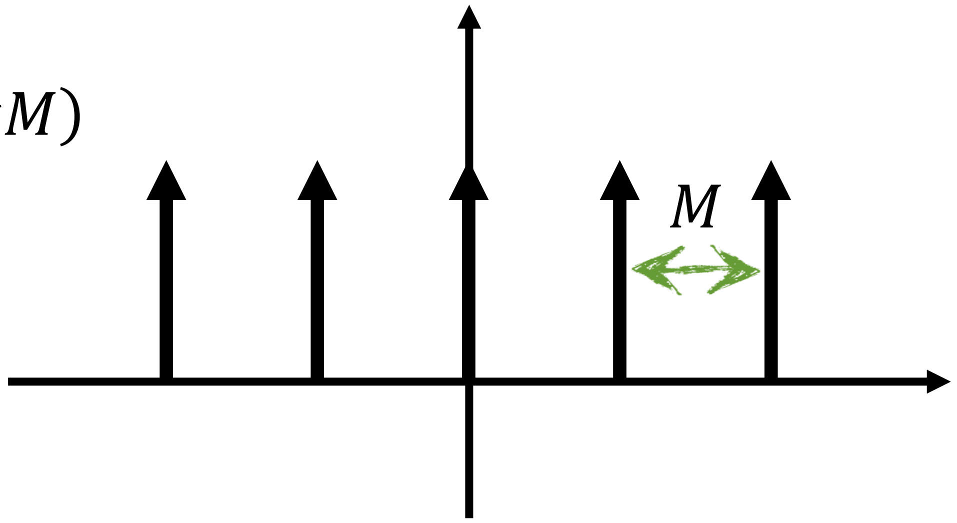
Jazz


Equalizer
On
Jazz
+12 dB
0 dB
-12 dB
Preamp
32
64
125
250
500
1K
2K
4K
8K
16K

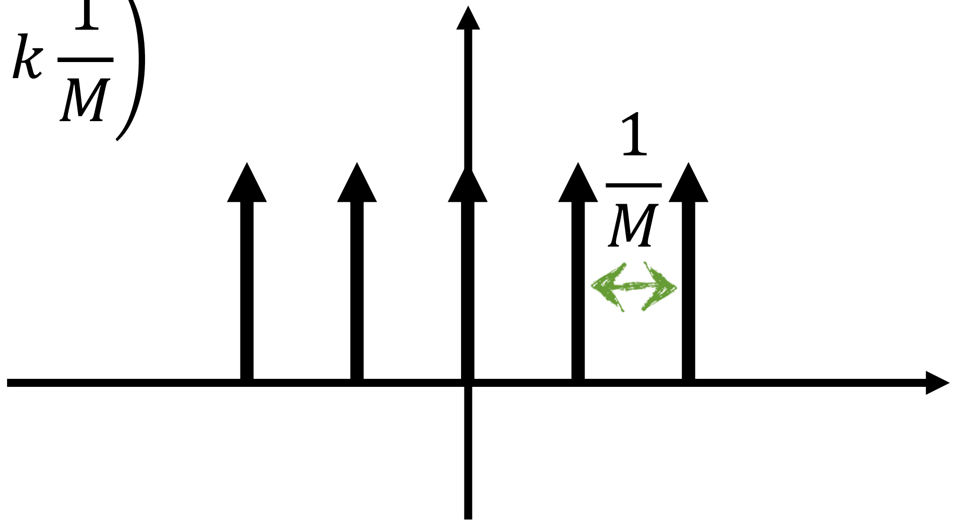
增强某些频率

## 梳状滤波器用于采样连续函数

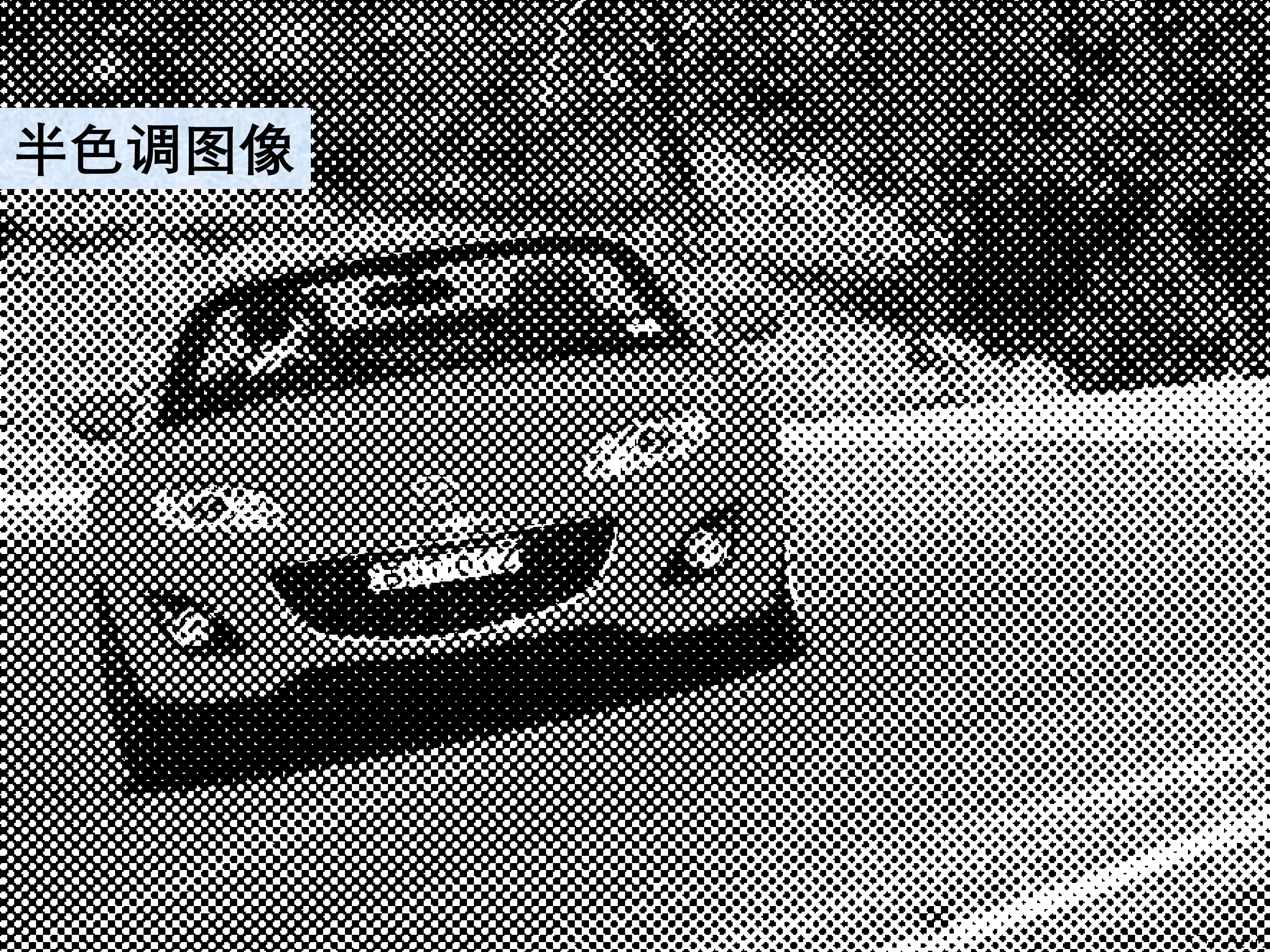
$$\mathrm{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$



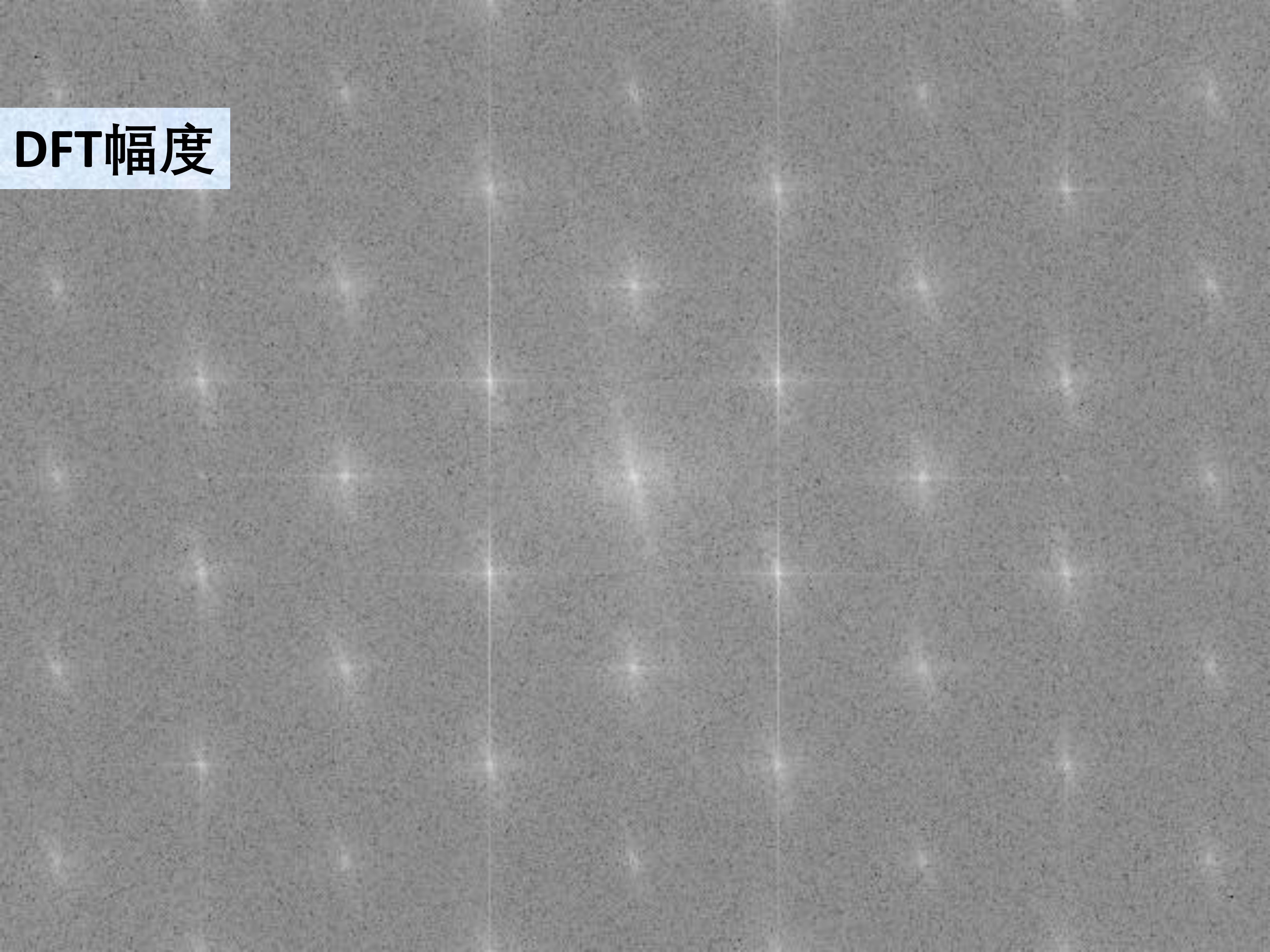
$$\mathrm{comb}(\omega) = \frac{1}{M} \sum_{k=-\infty}^{\infty} \delta\left(\omega - k\frac{1}{M}\right)$$



## 半色调图像

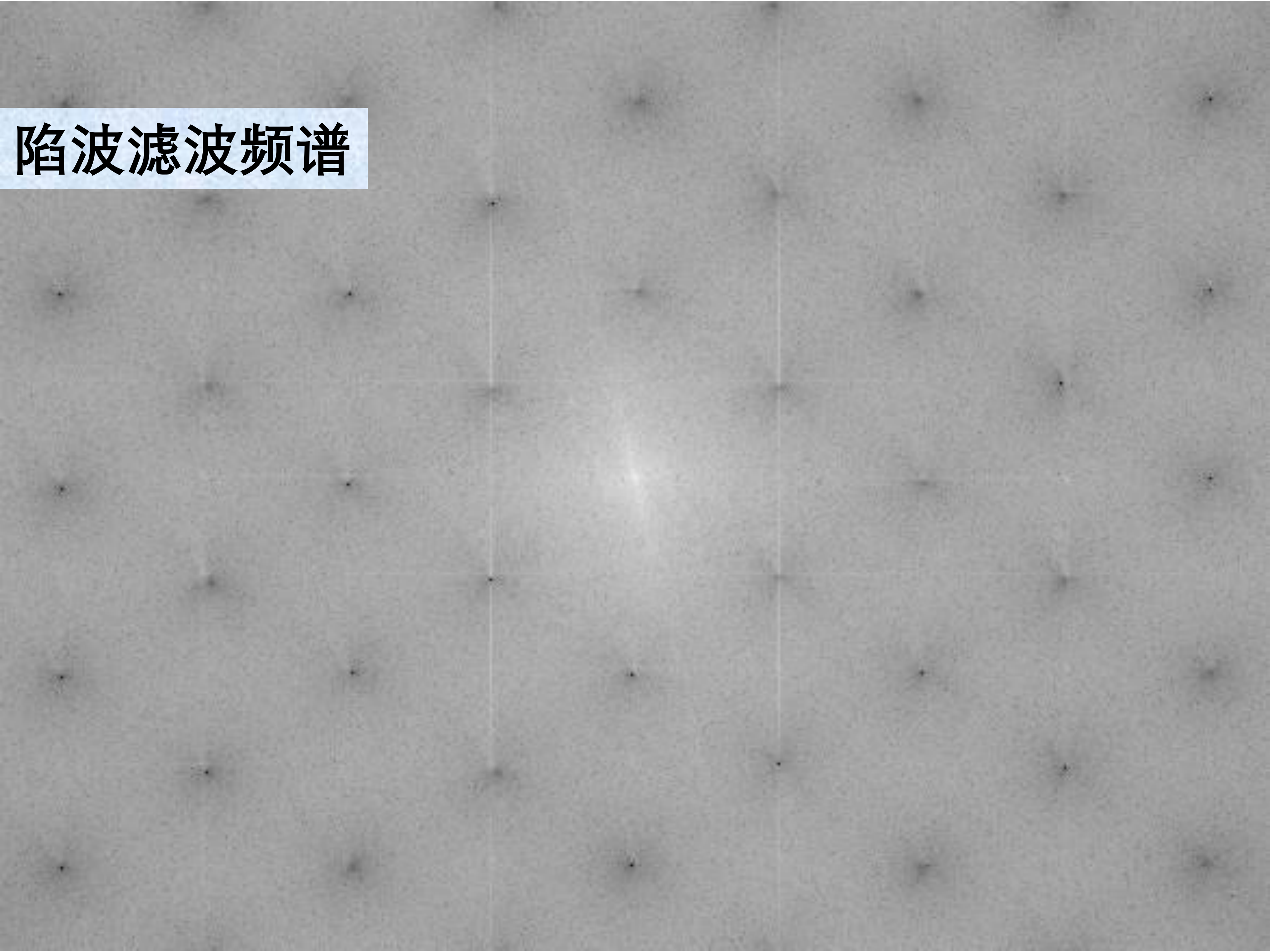


你能看清车牌吗？

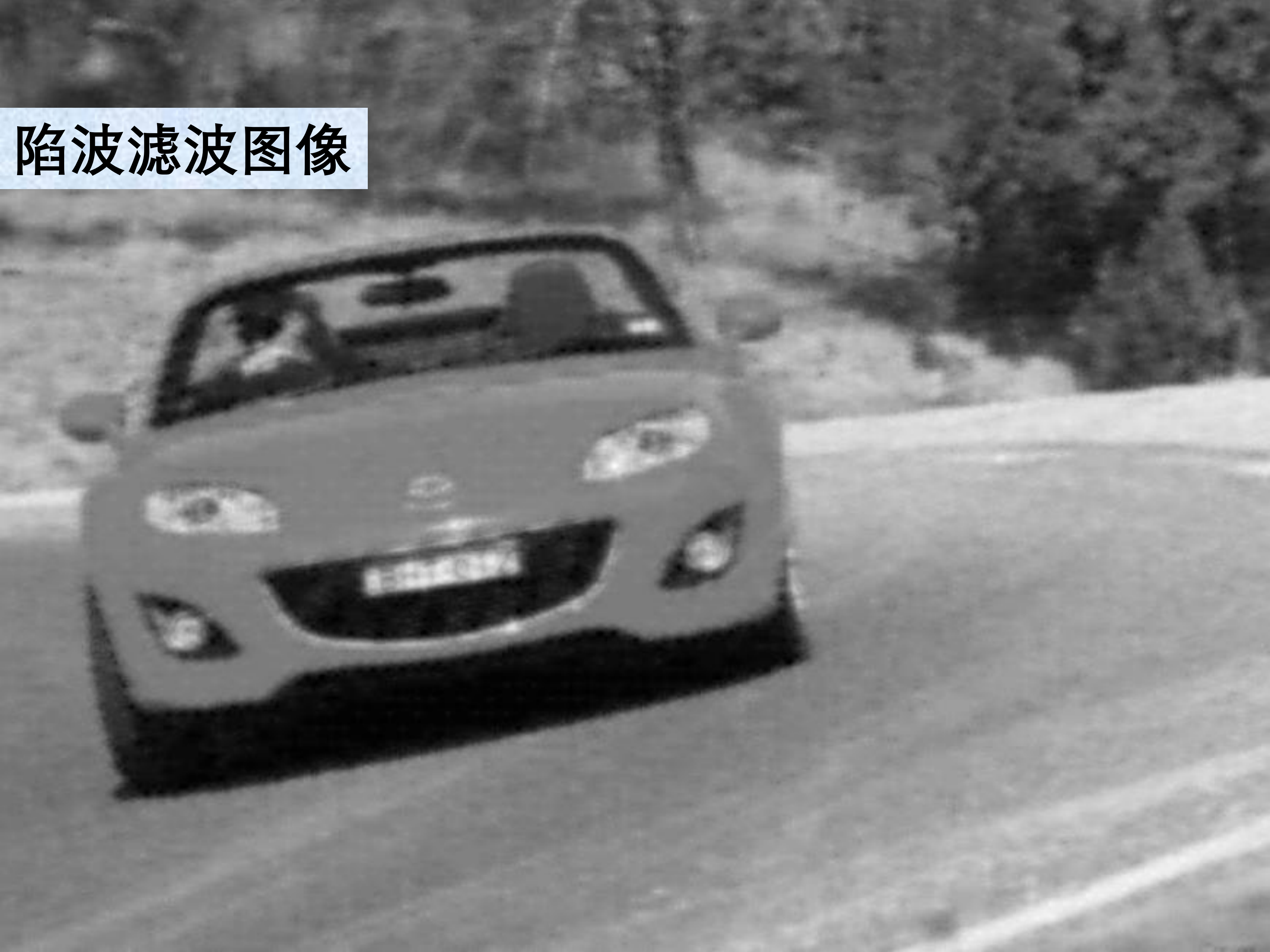


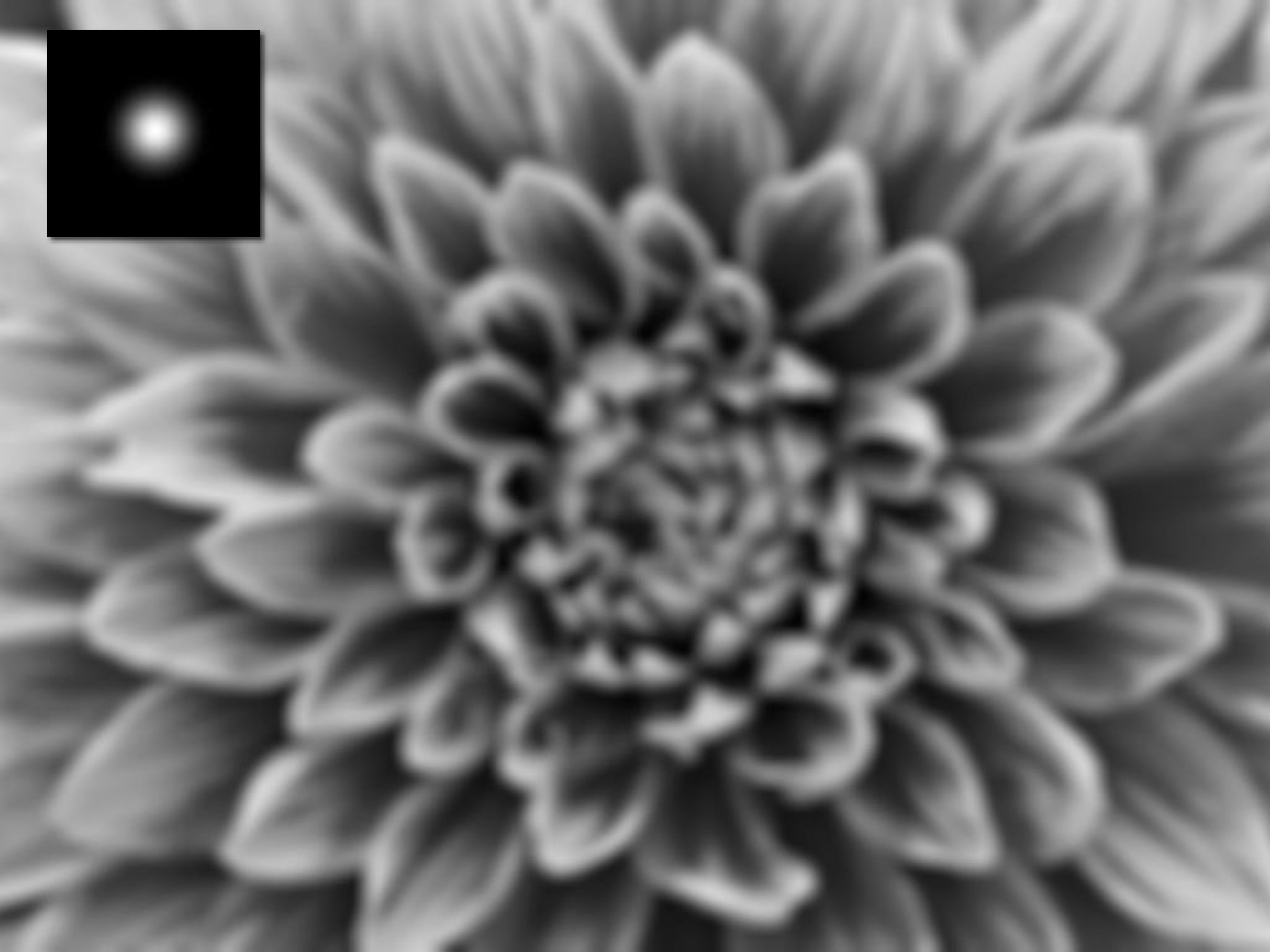
## DFT幅度

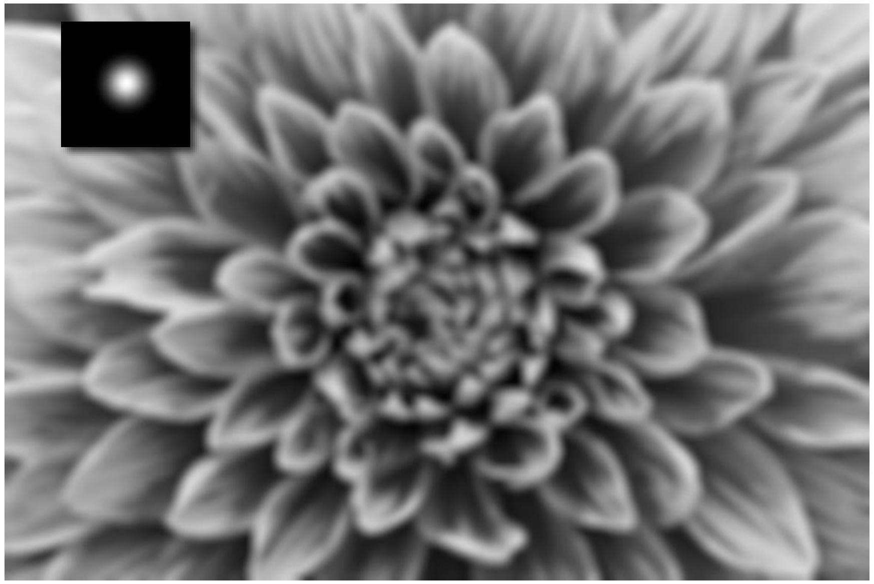
## 陷波滤波频谱

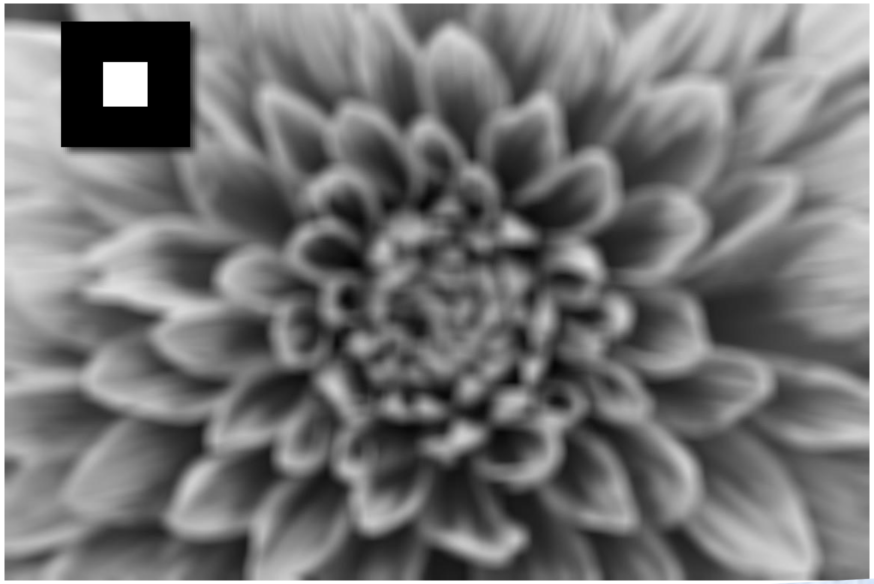


## 陷波滤波图像

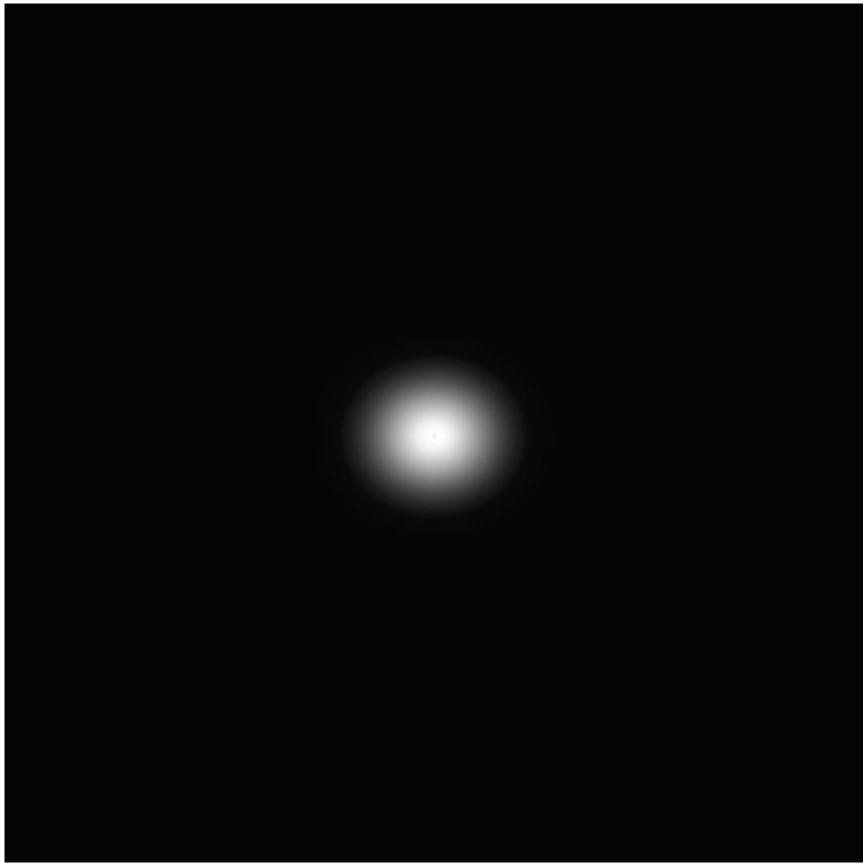




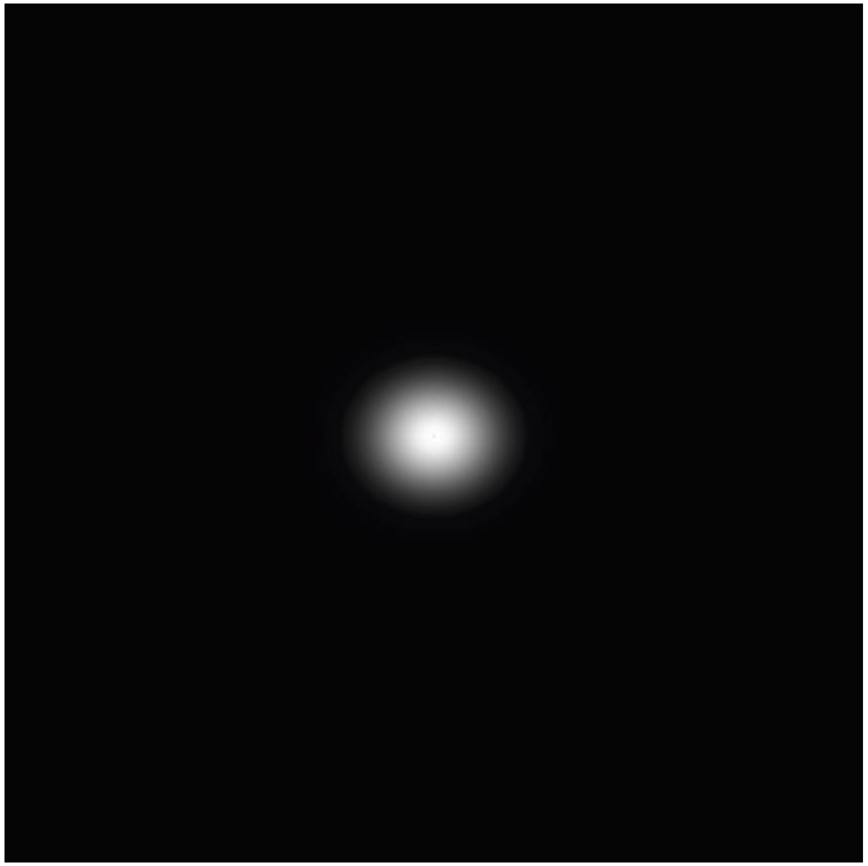


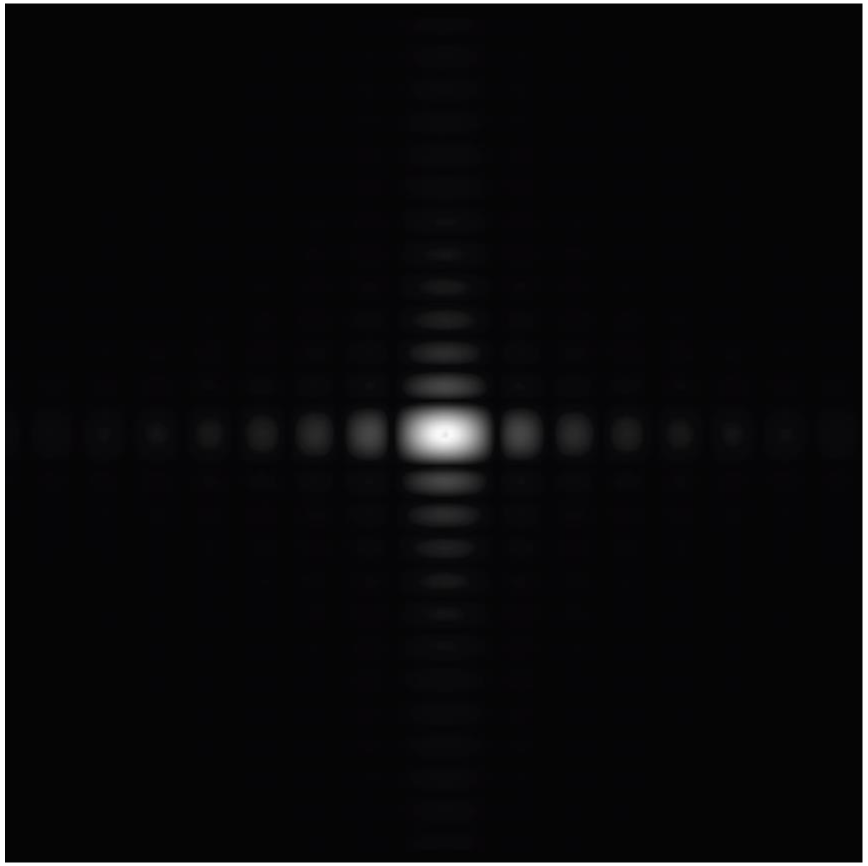


为什么高斯滤波后的图像比方框滤波后的图像更平滑?

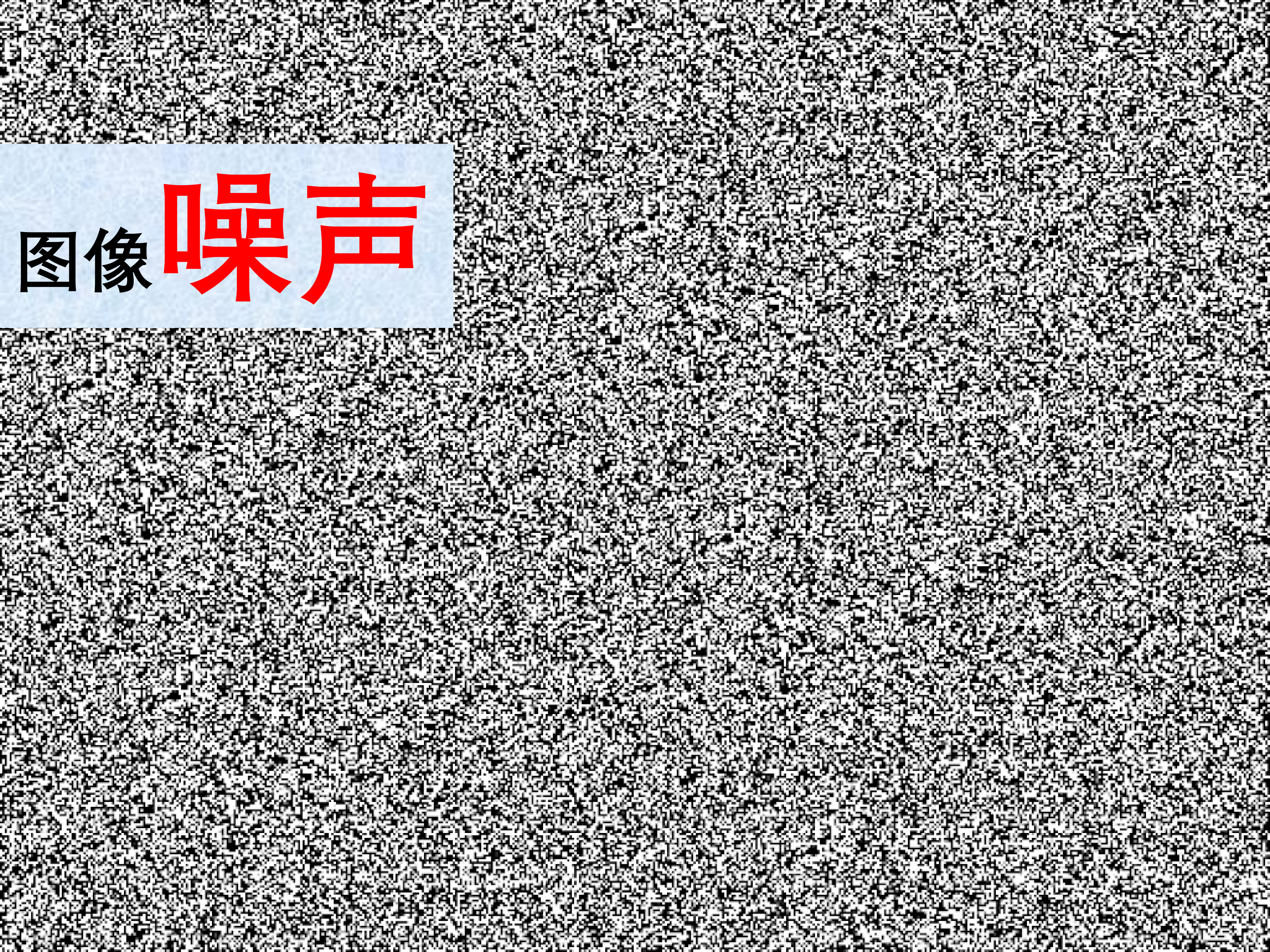


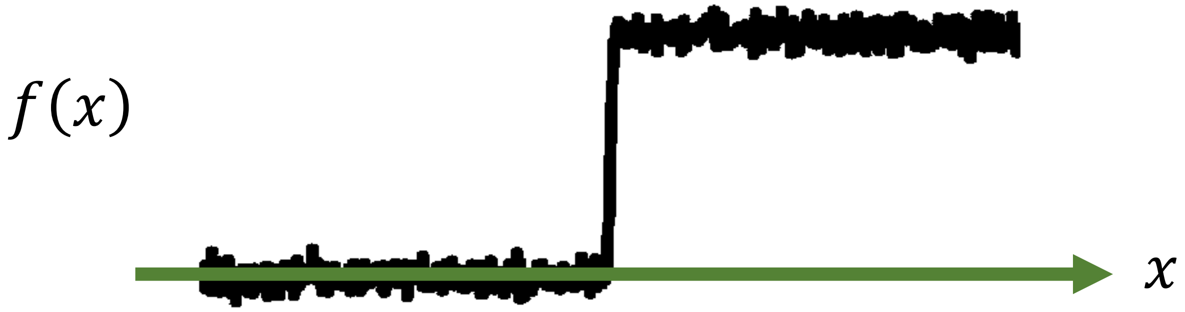
高斯傅里叶幅度

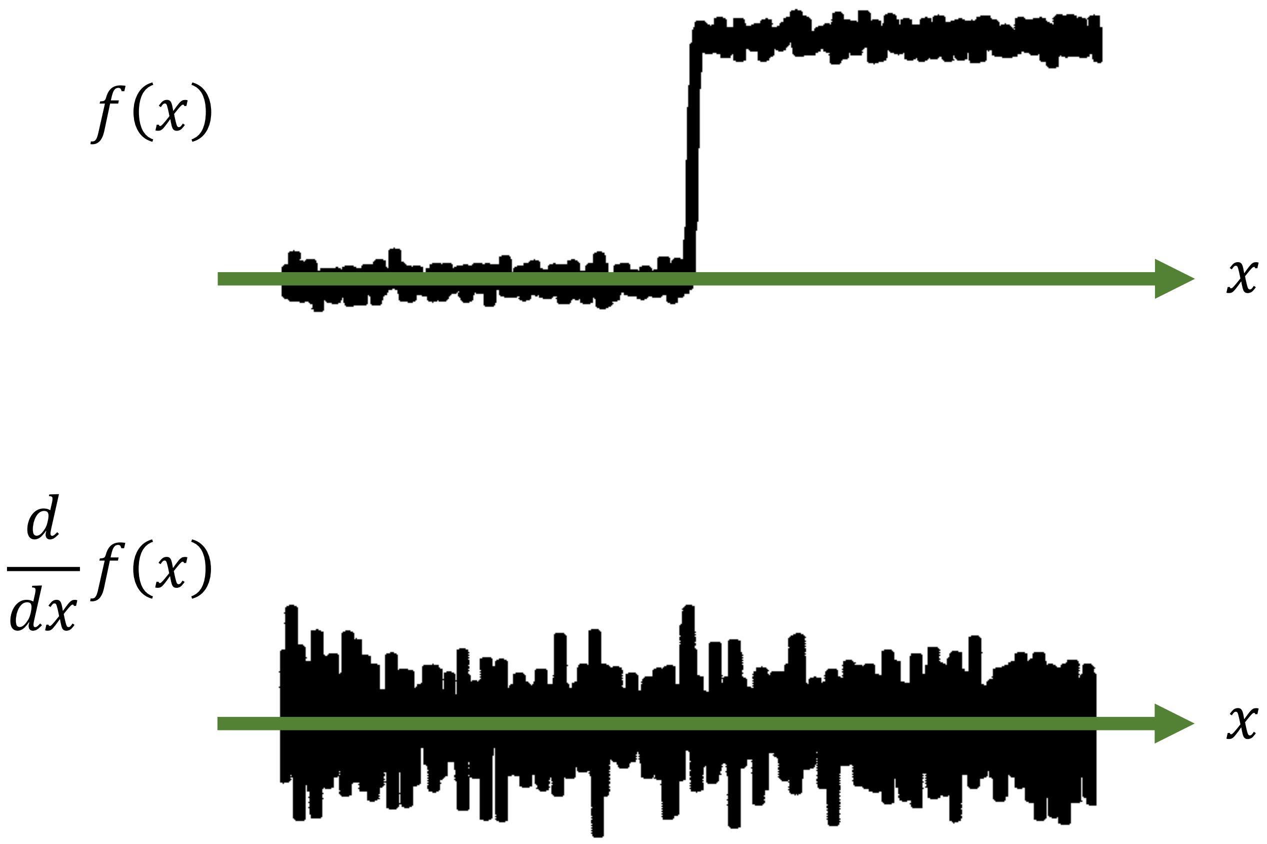

高斯傅里叶幅度

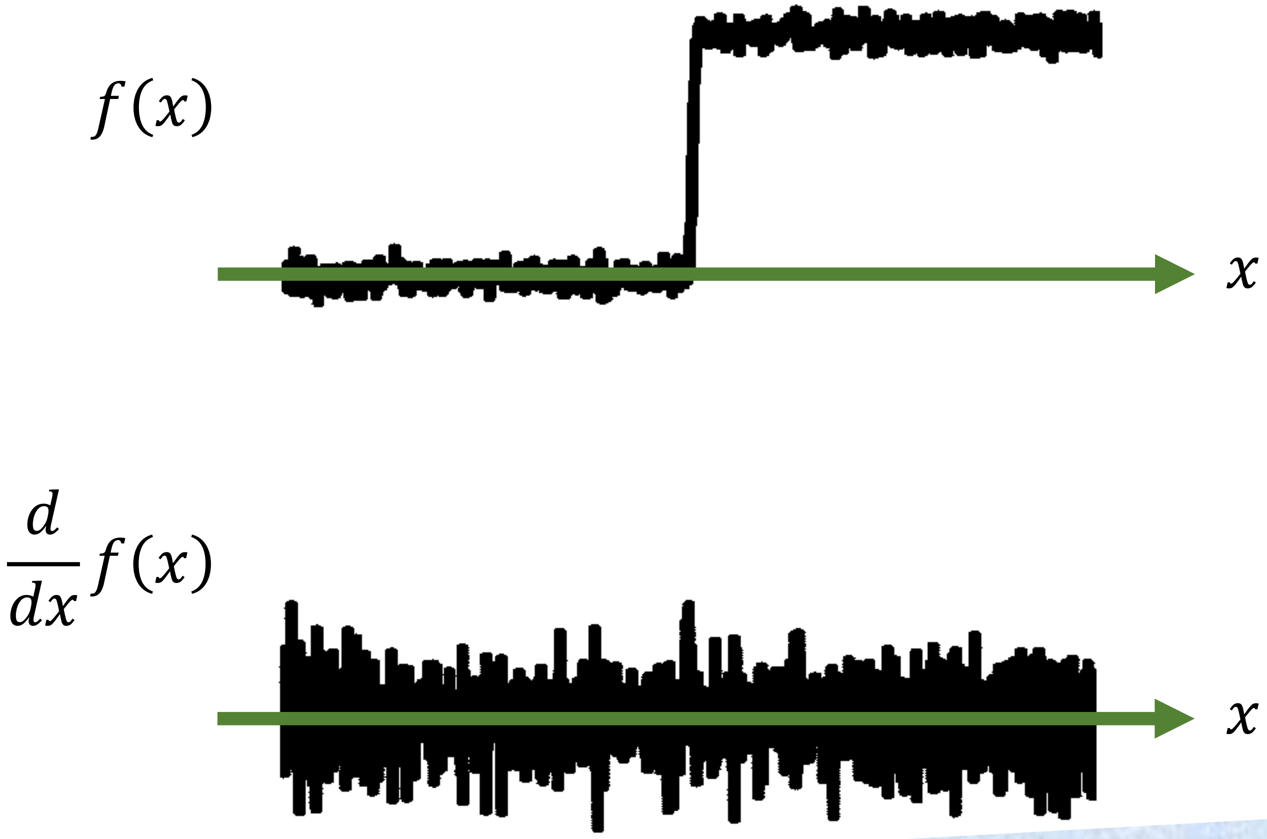

方框傅里叶幅度

## 图像噪声

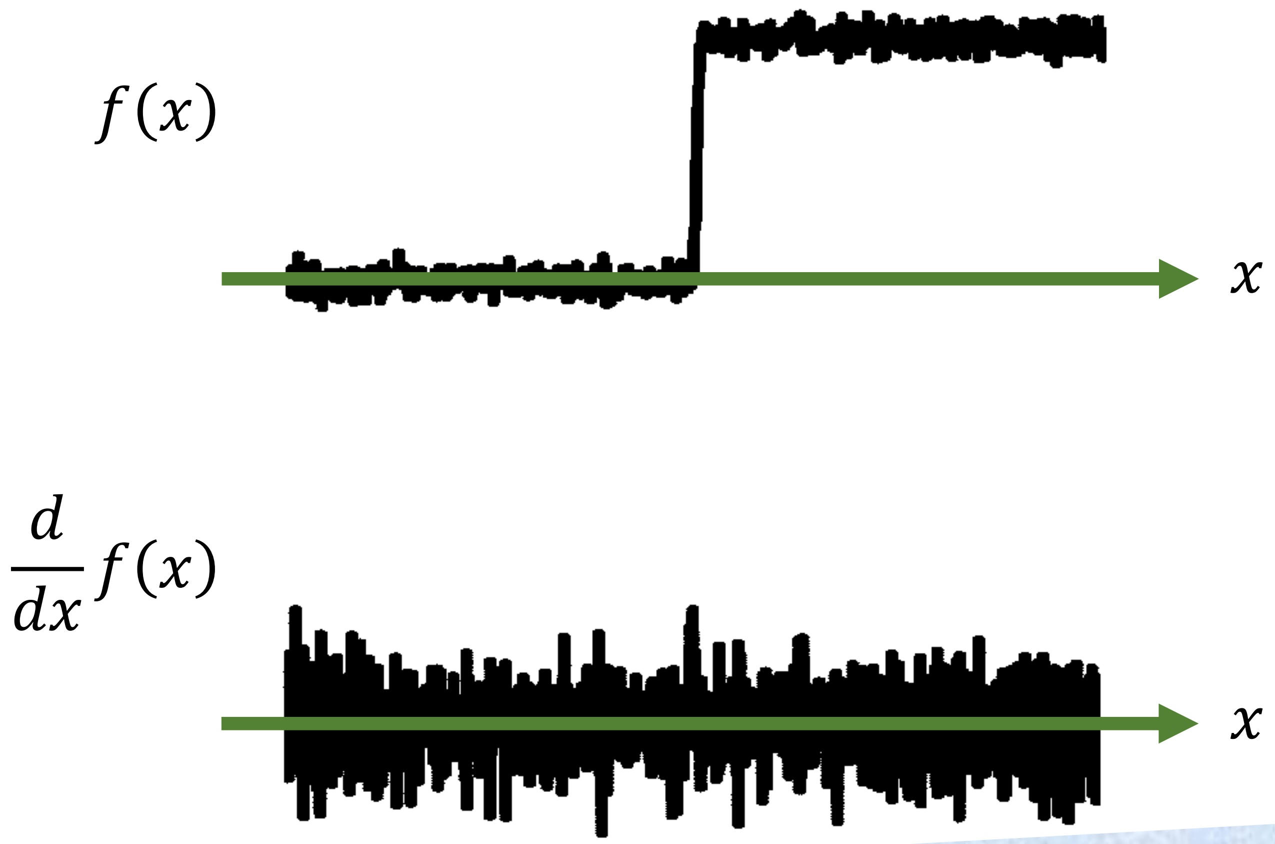



$f(x)$
$x$


$f(x)$
$x$
$\frac{d}{dx}f(x)$
$x$

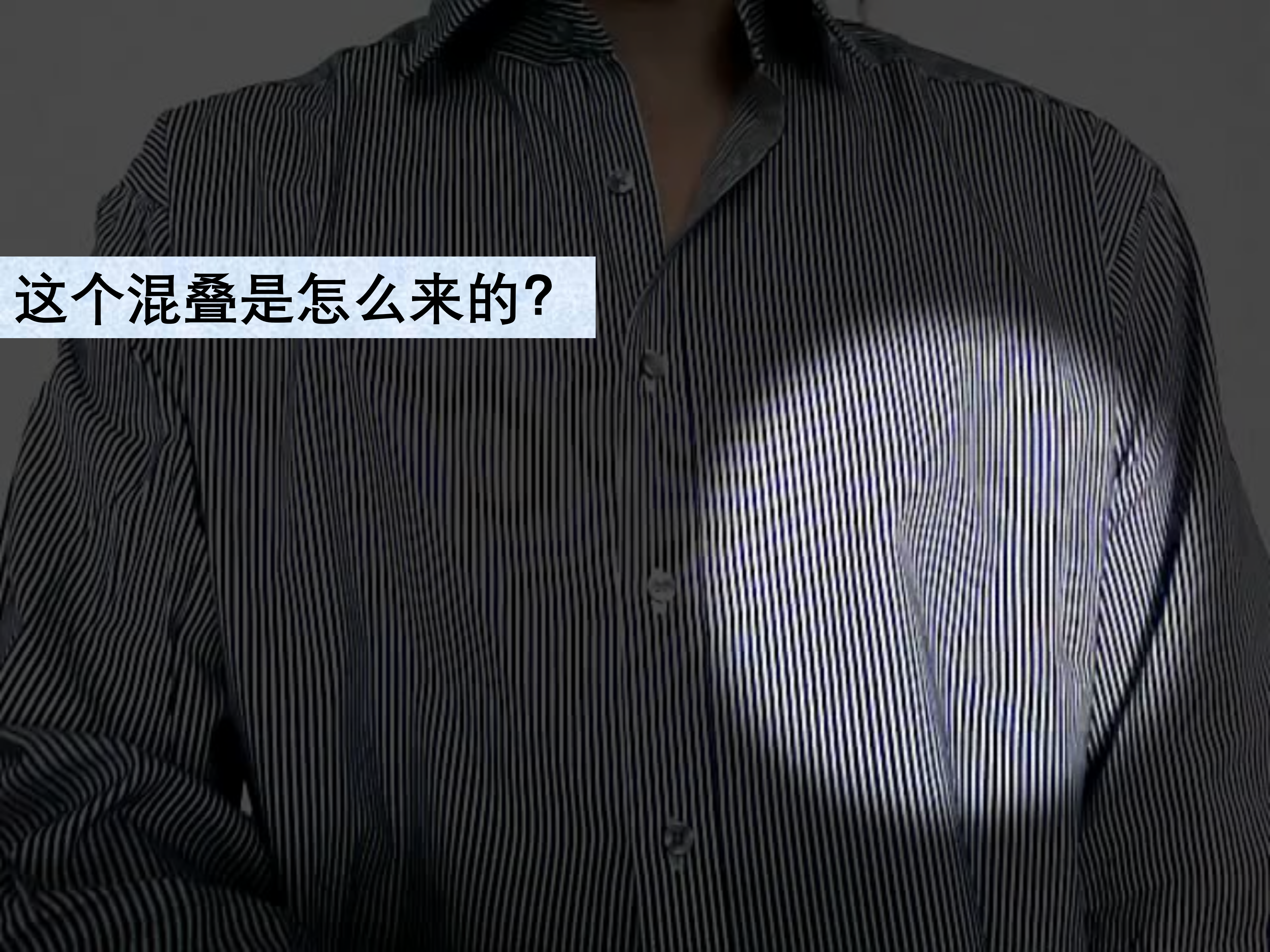


为什么微分会放大噪声?

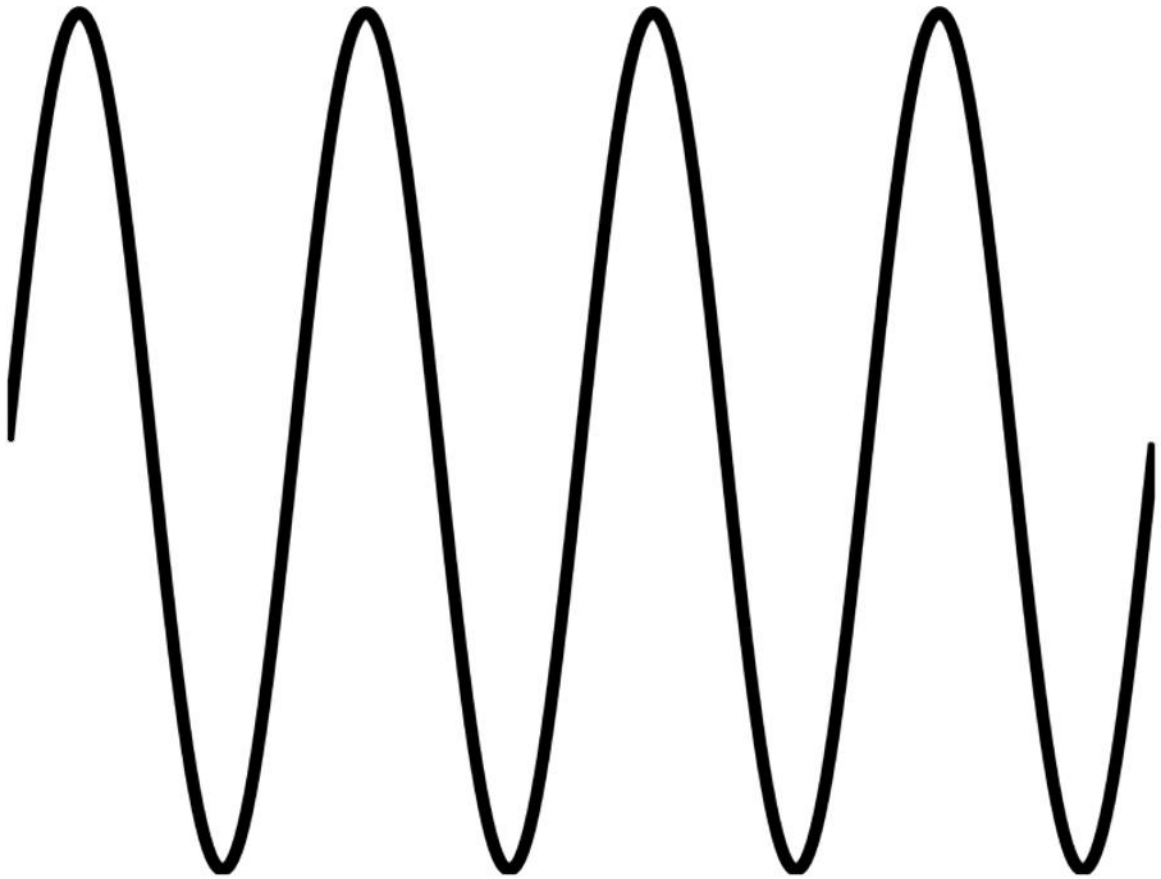


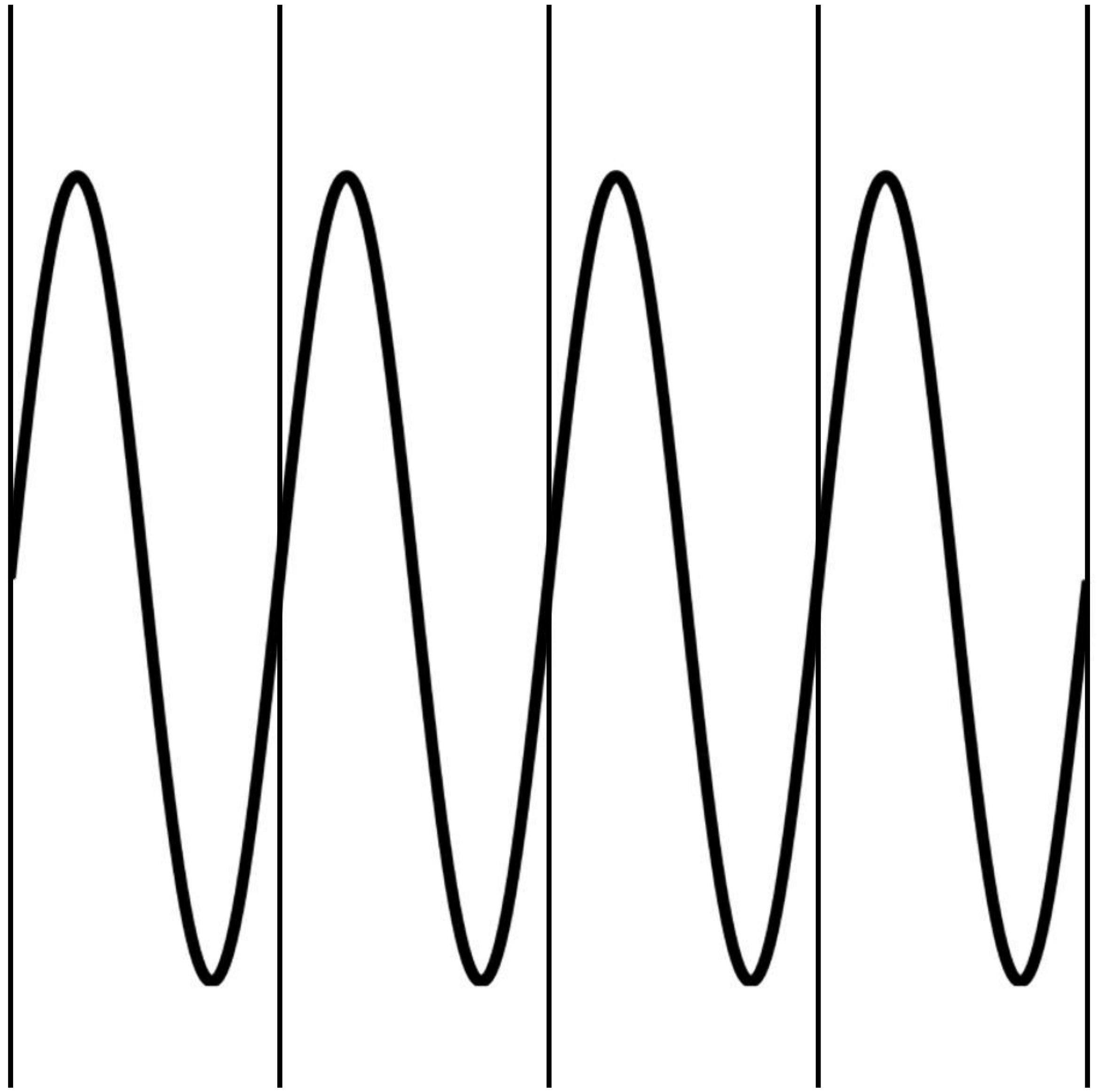
## 为什么微分会放大噪声?

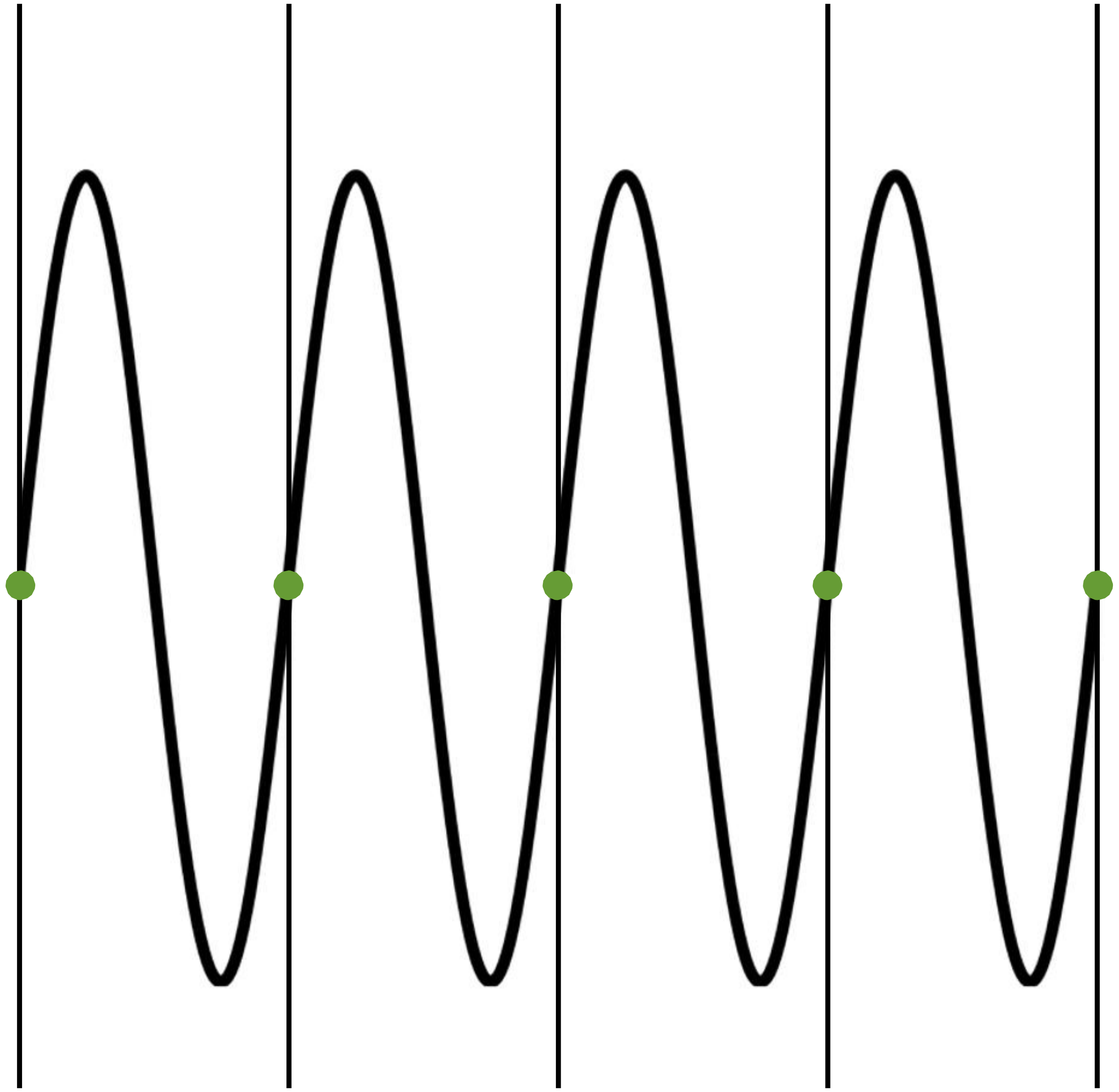
$$\mathcal{F}\left\{\frac{d^n f(x)}{dx^n}\right\} = (i2\pi\omega)^n F(\omega)$$

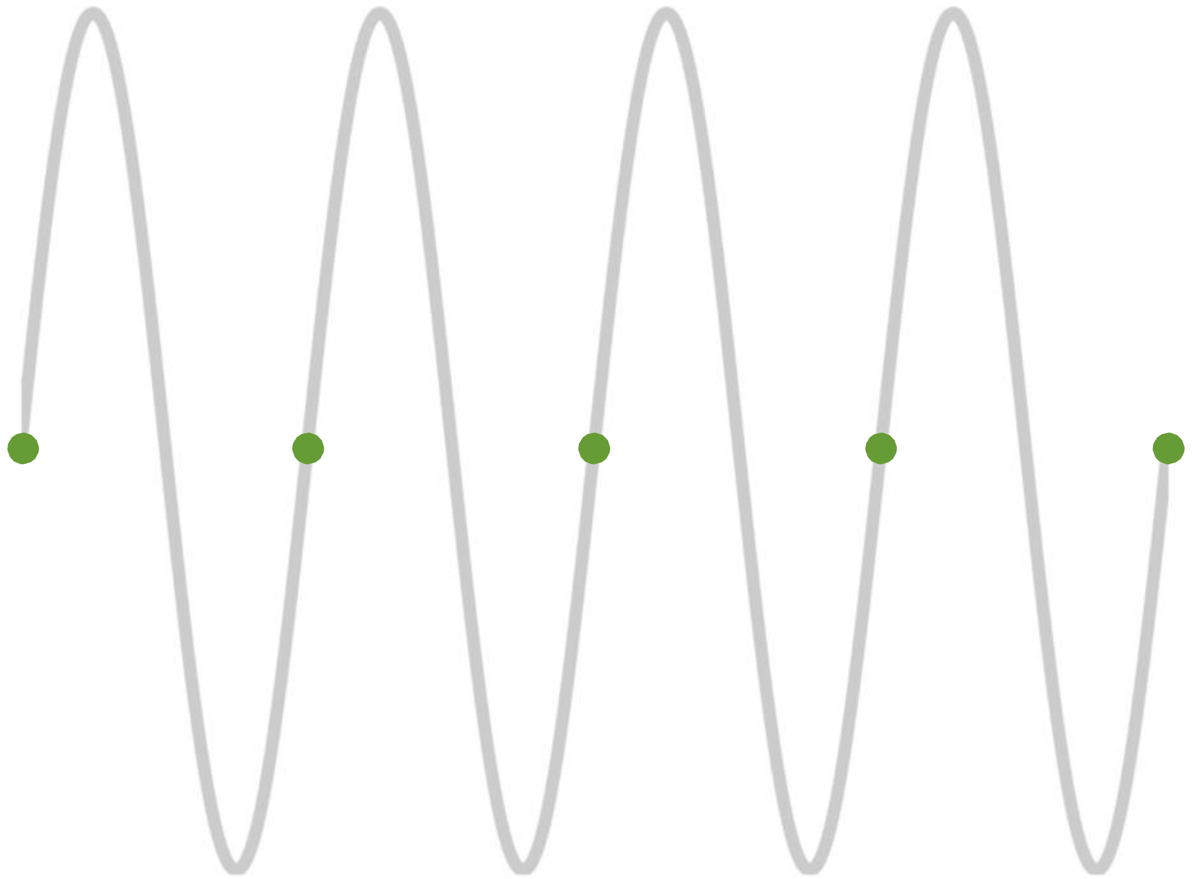


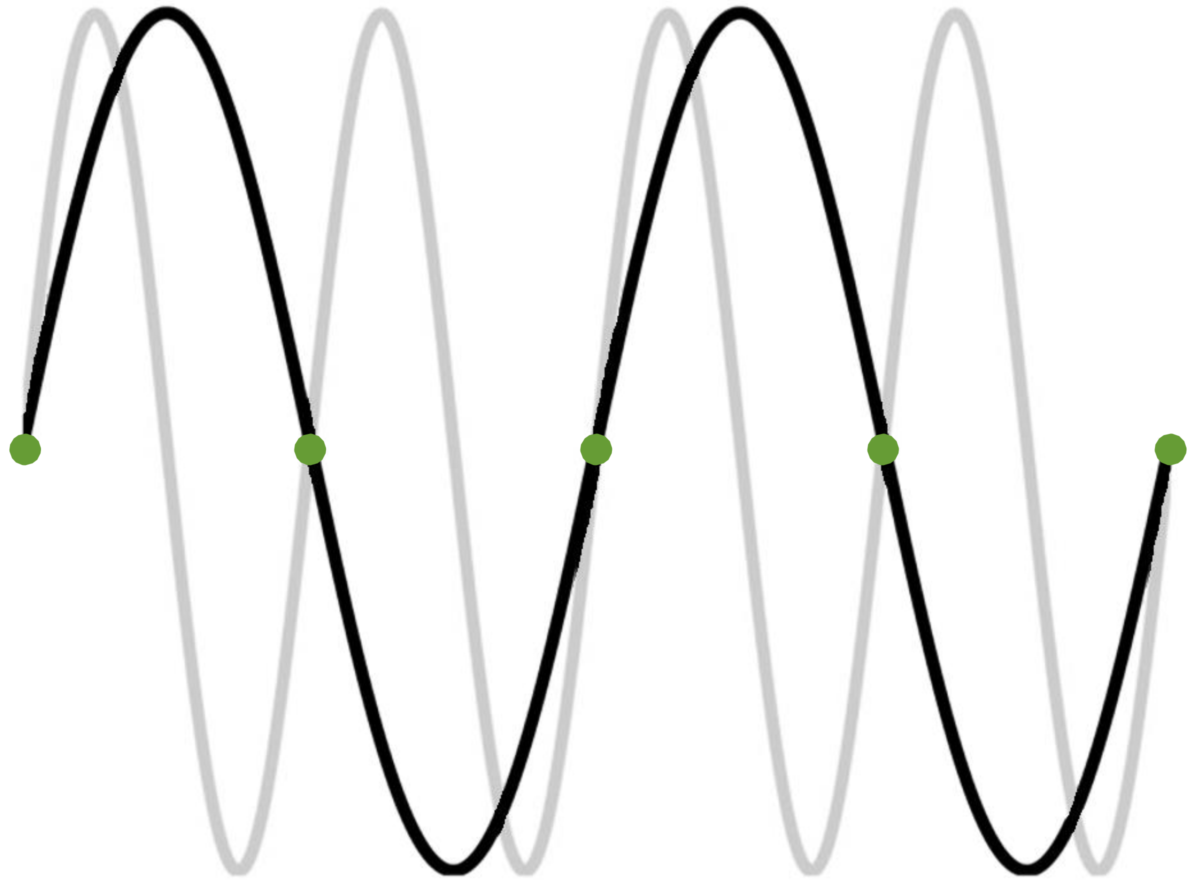
这个混叠是怎么来的？



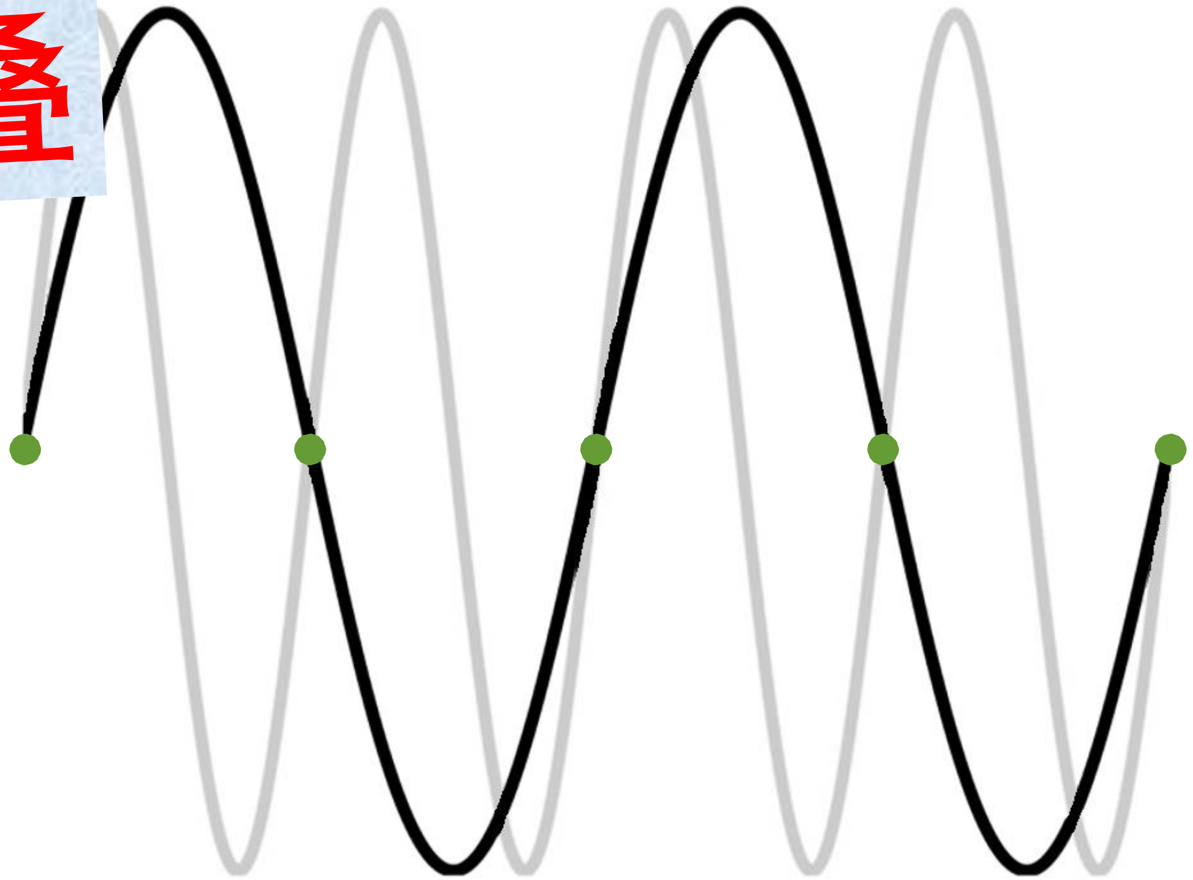




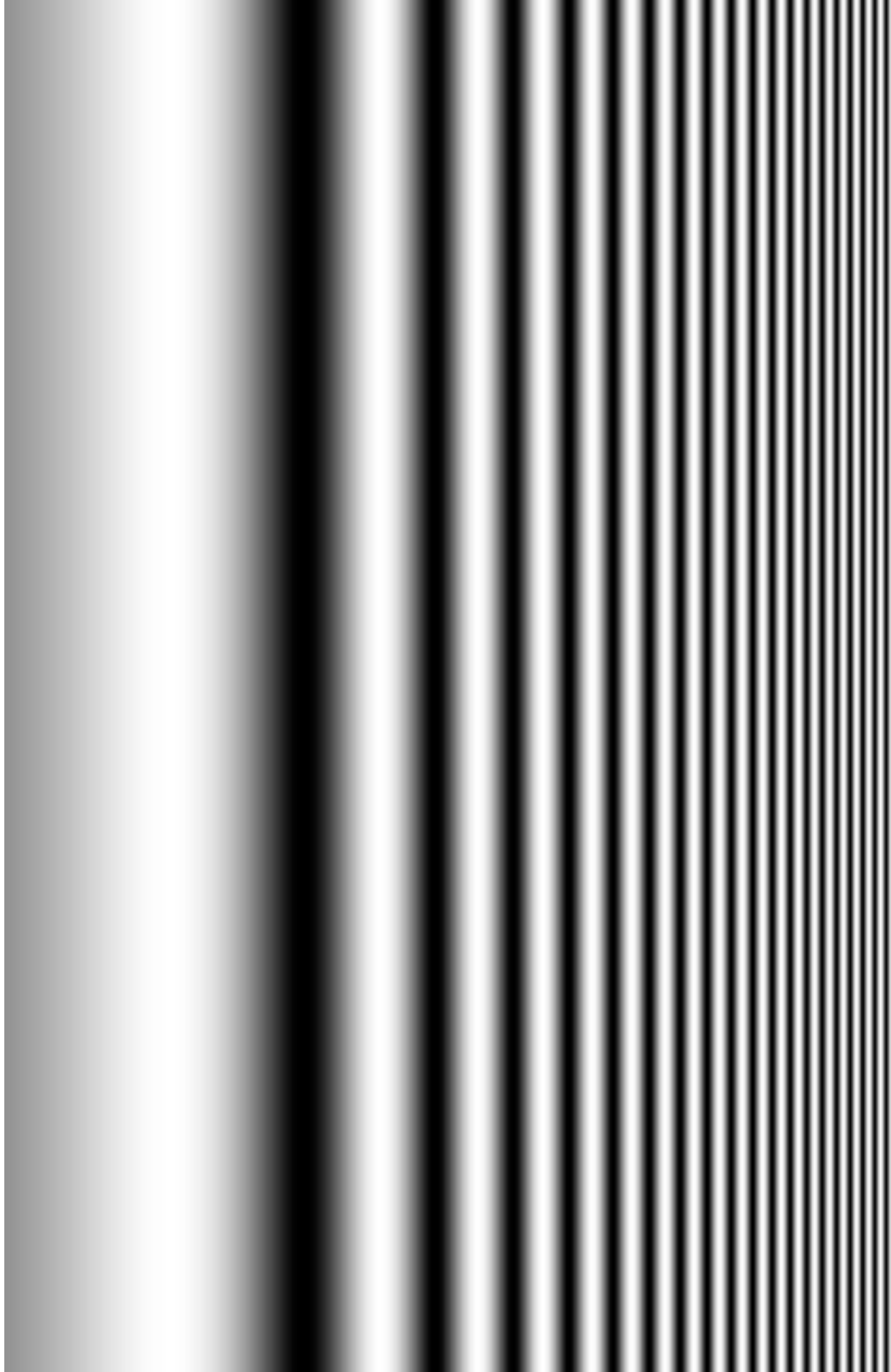




## 混叠

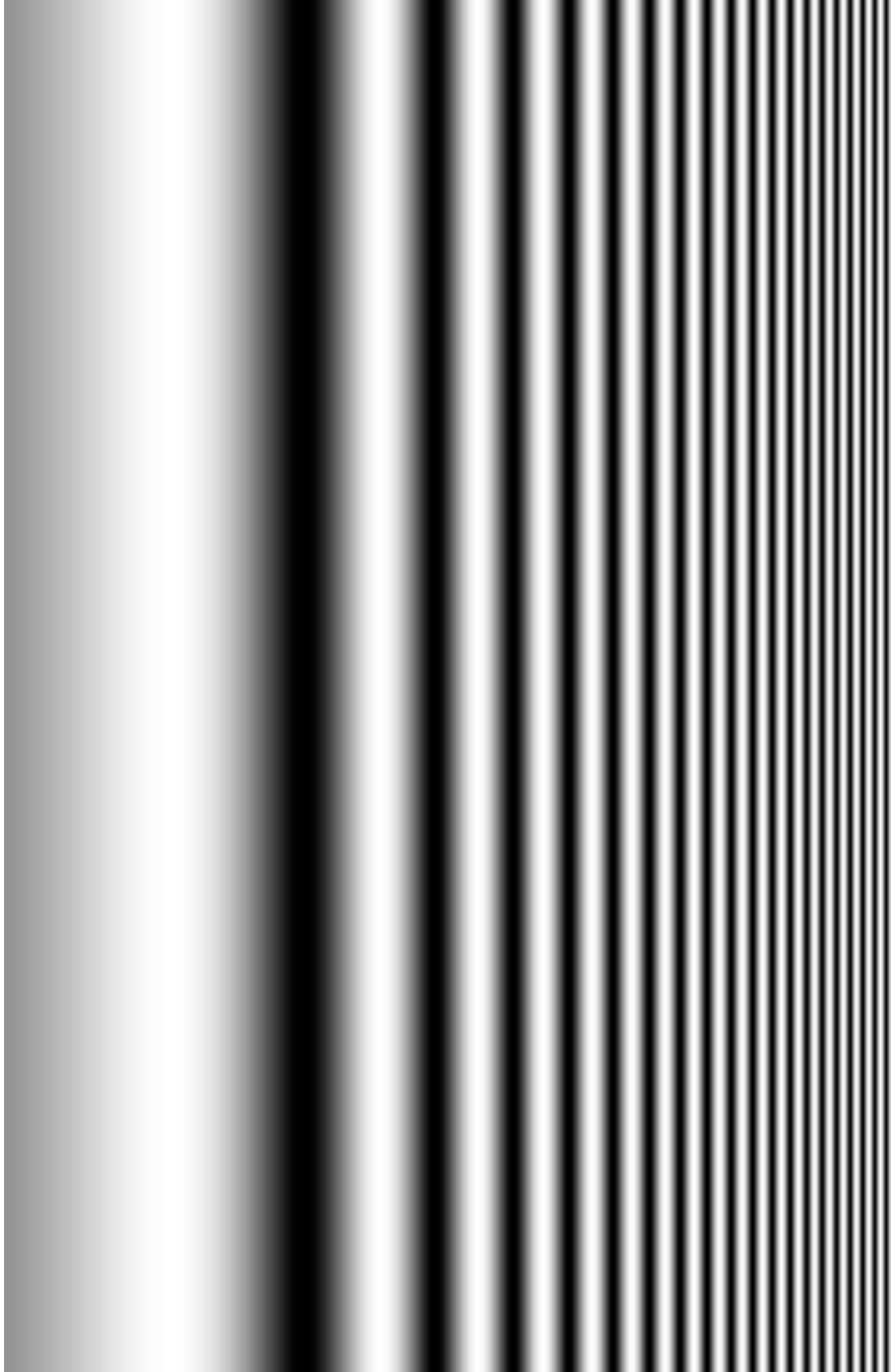


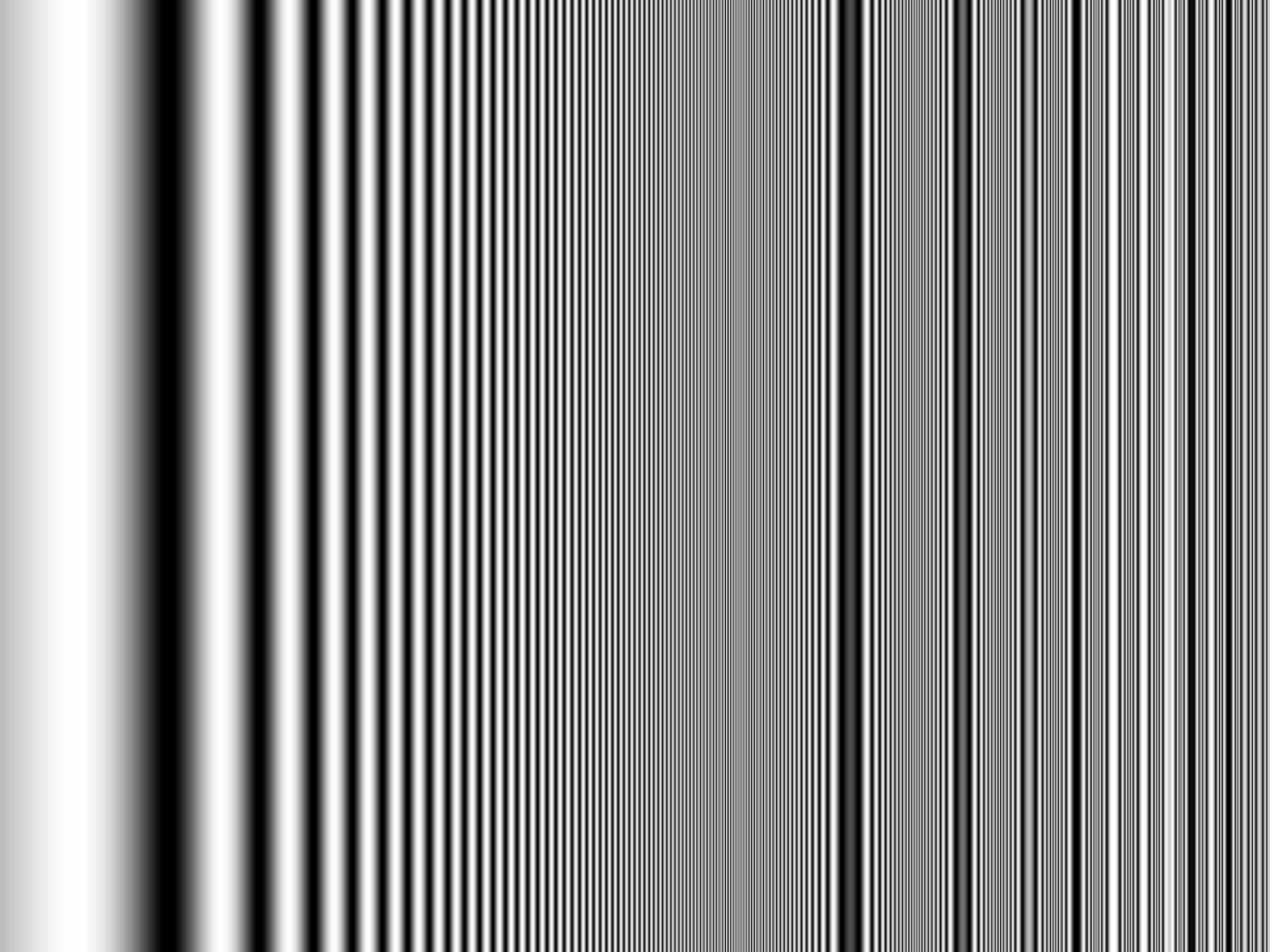
$$f(x, y) = \sin(2^x x)$$

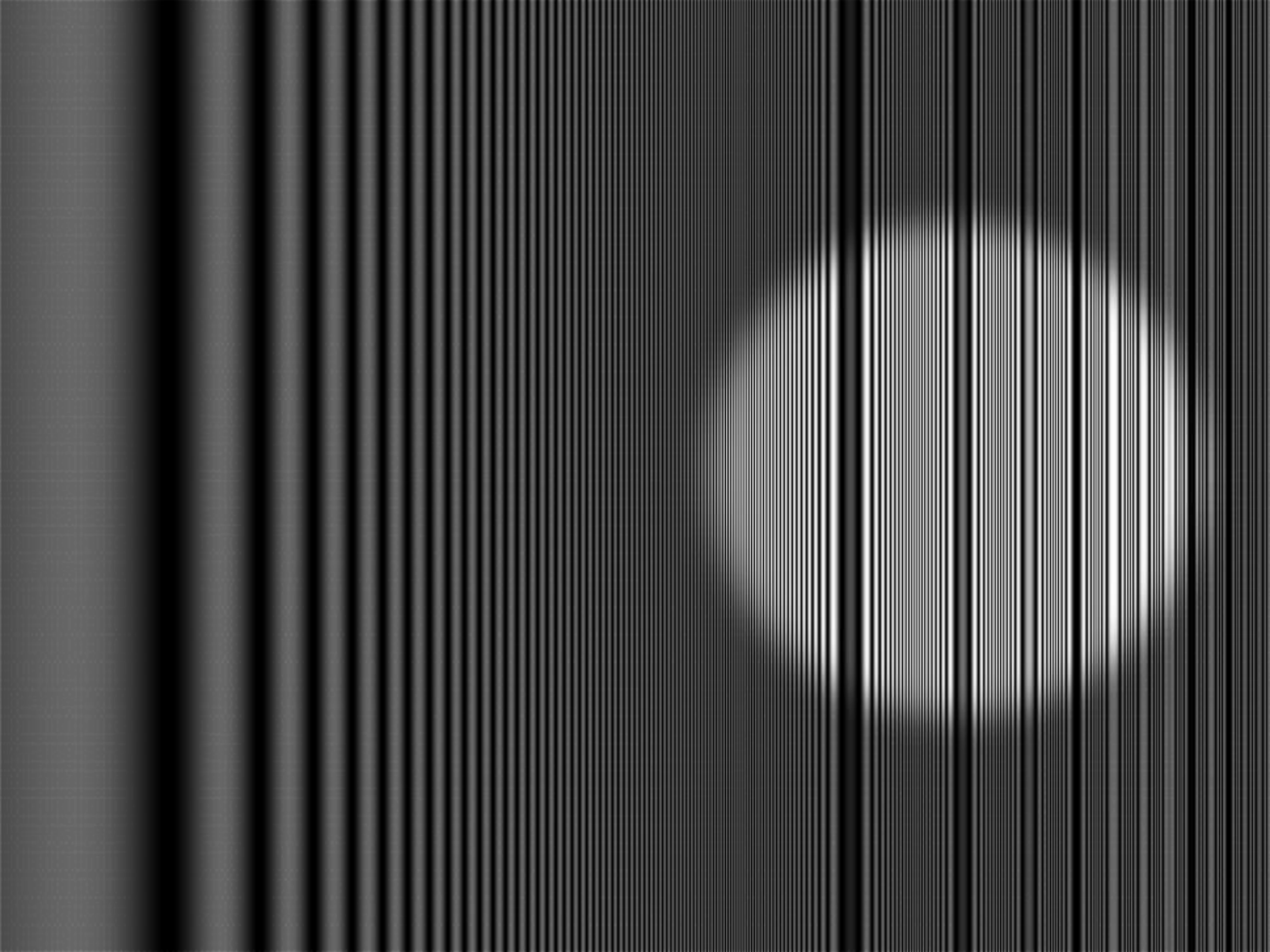


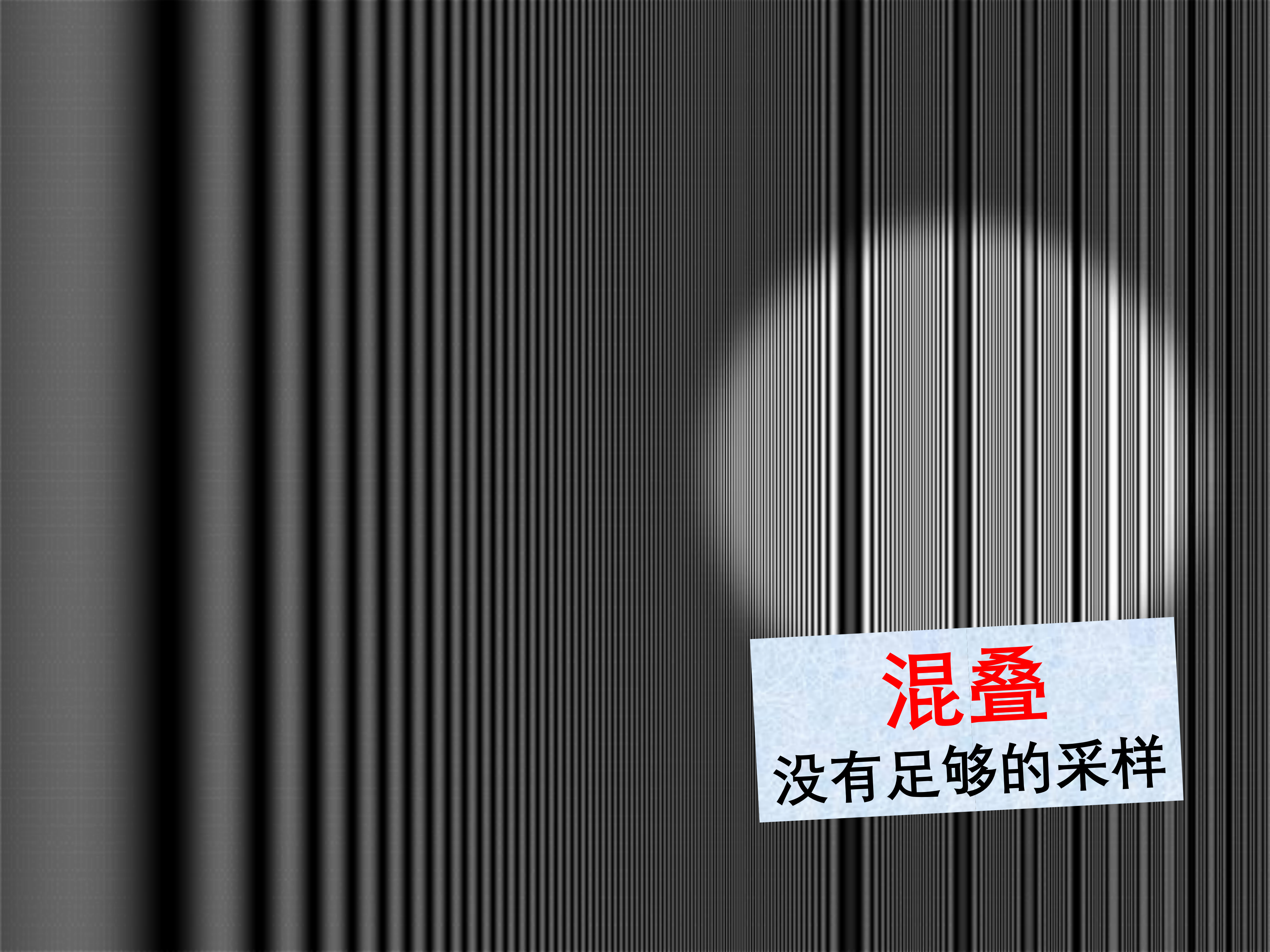
$$f(x,y) = \sin(2^x x)$$

频率随着$x$增加







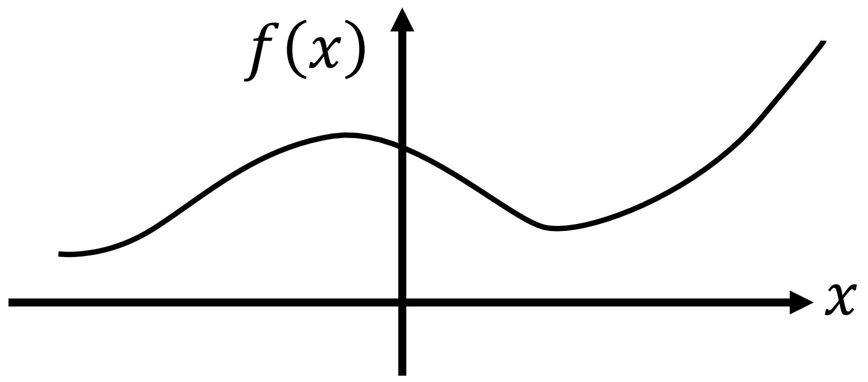


## 混叠

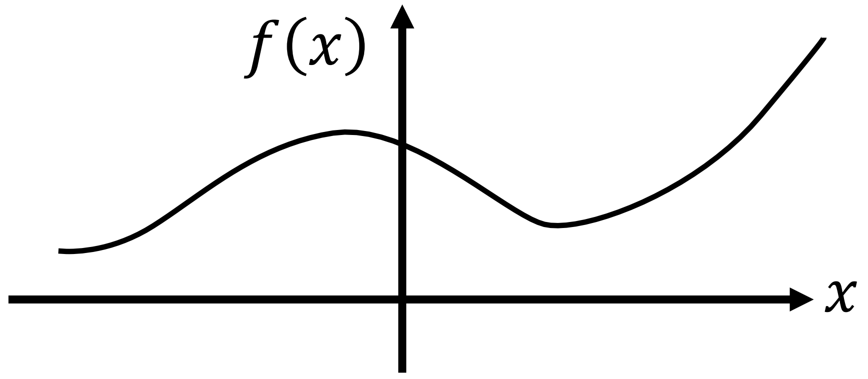
没有足够的采样

## 在频域中看混叠

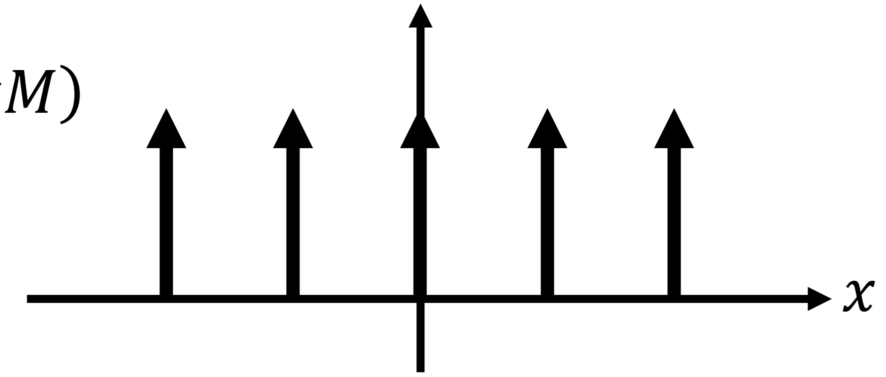
空间域



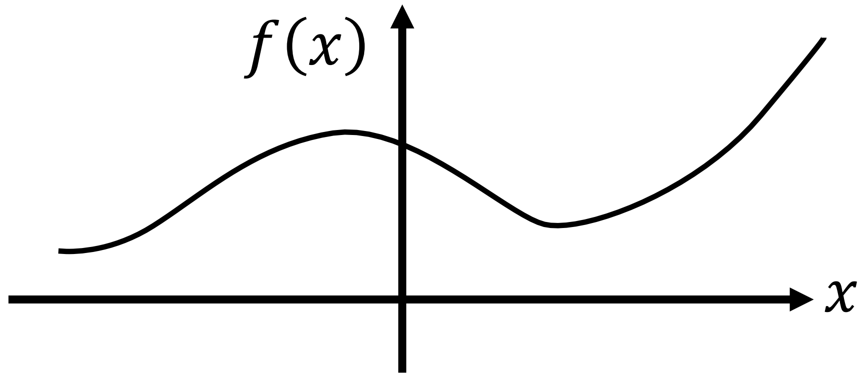
空间域



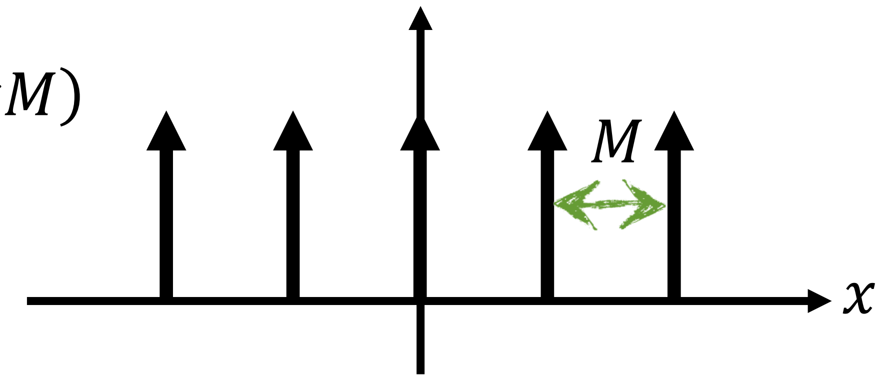
$$\mathrm{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$



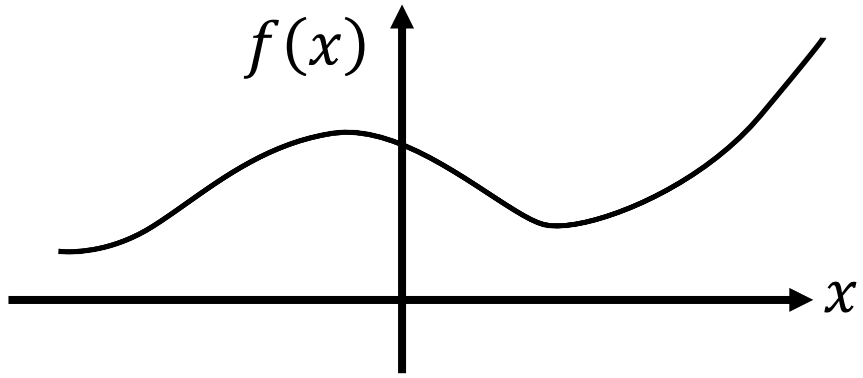
空间域



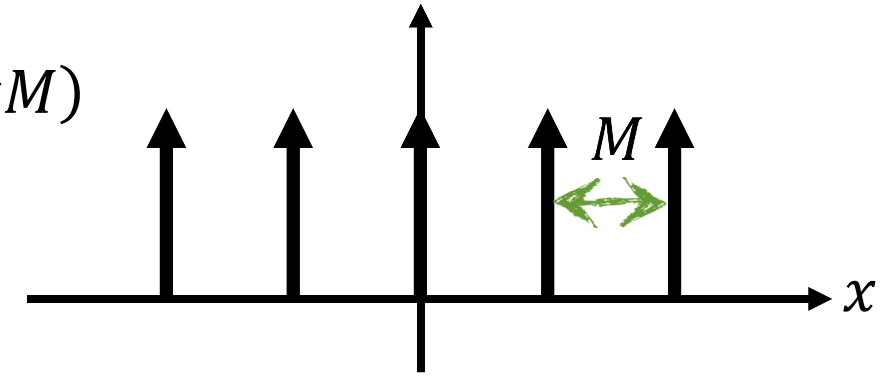
$$\mathrm{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$



空间域



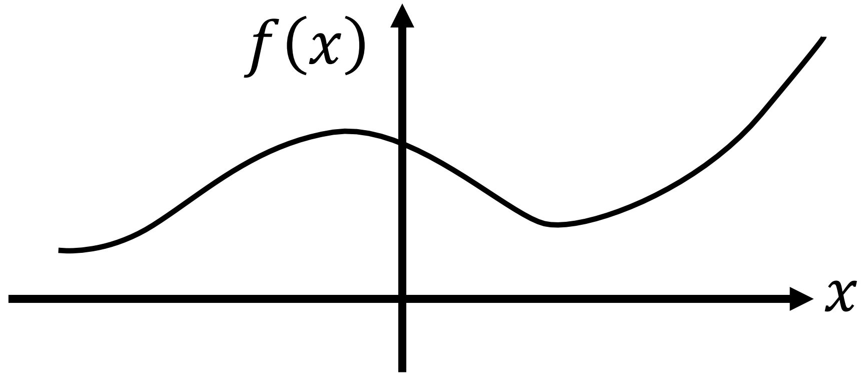
$$\mathrm{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$

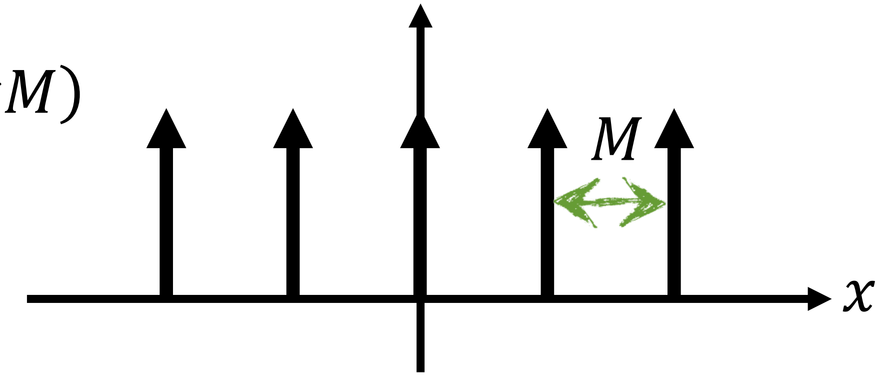


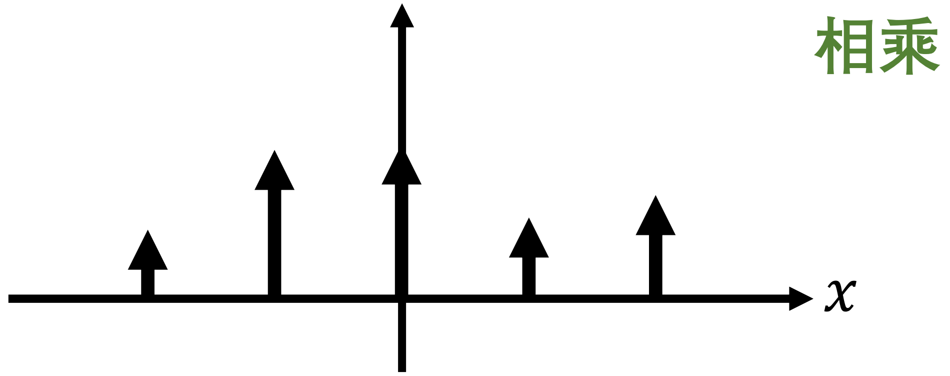
相乘

空间域

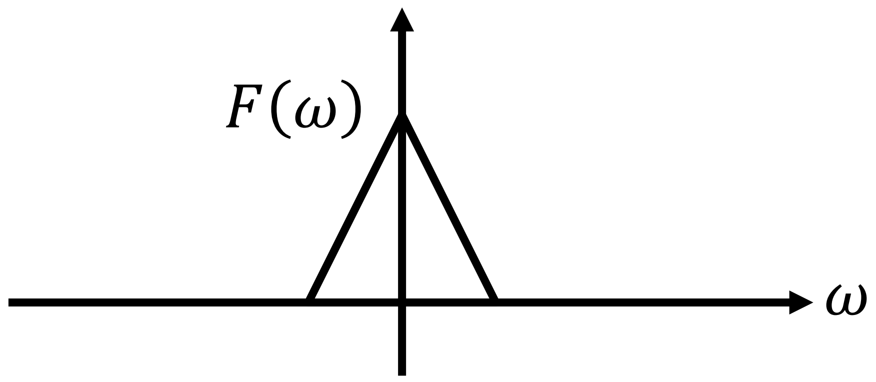
$$\mathrm{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$





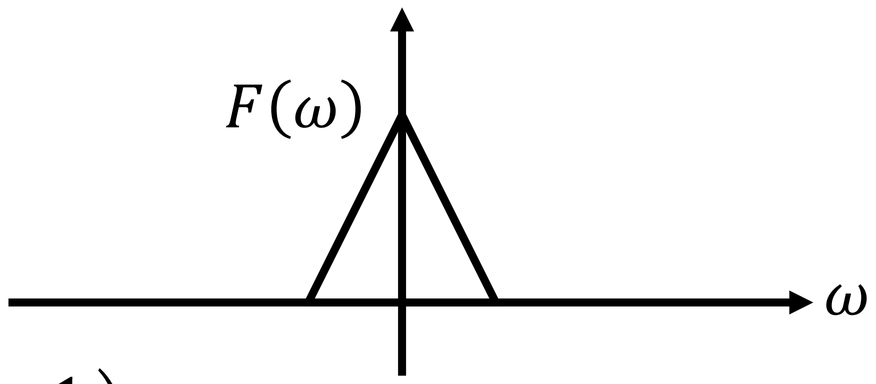


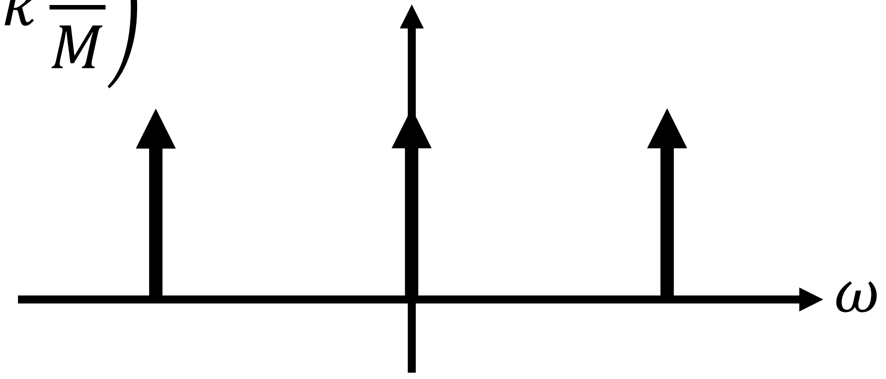
## 频域



## 频域

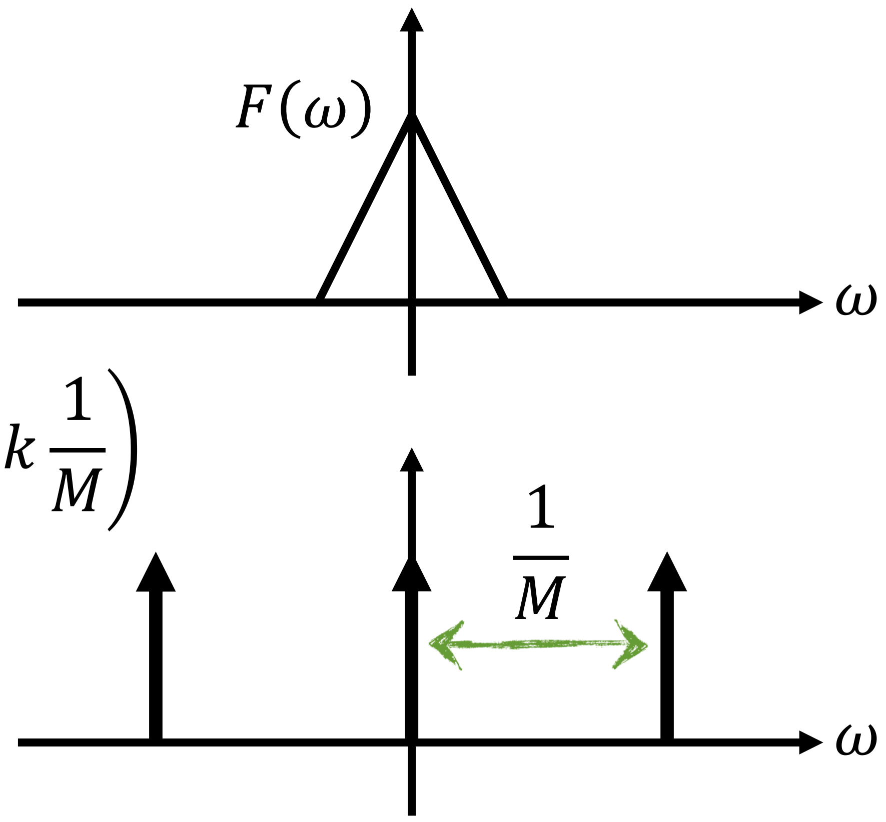
$$\mathrm{comb}(\omega) = \frac{1}{M} \sum_{k=-\infty}^{\infty} \delta\left(\omega - k\frac{1}{M}\right)$$





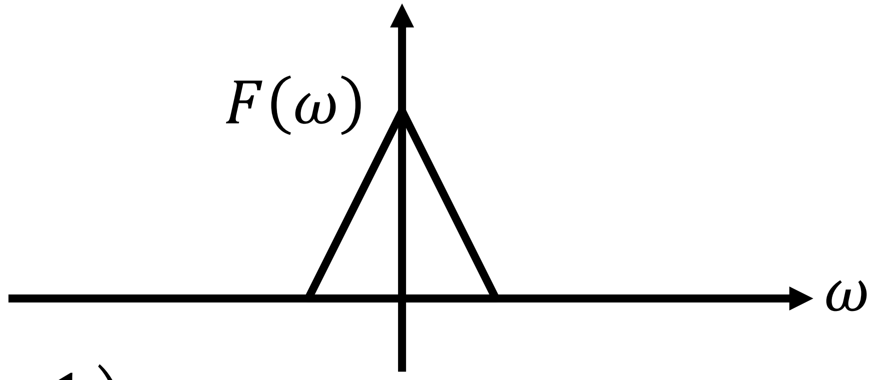
## 频域

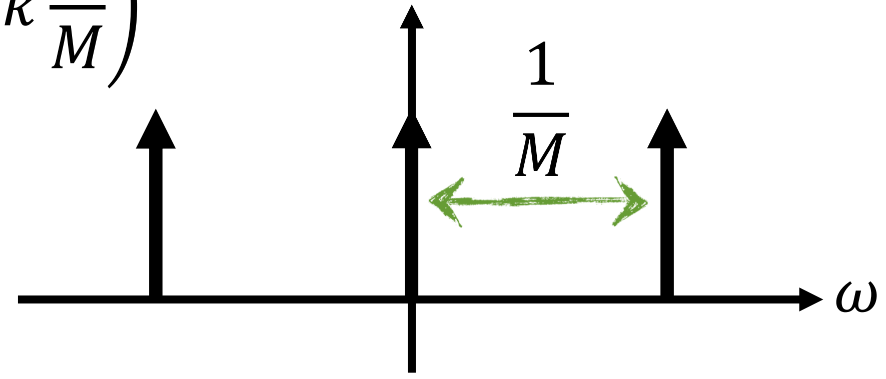
$$\text{comb}(\omega) = \frac{1}{M}\sum_{k=-\infty}^{\infty}\delta\left(\omega - k\frac{1}{M}\right)$$



## 频域

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

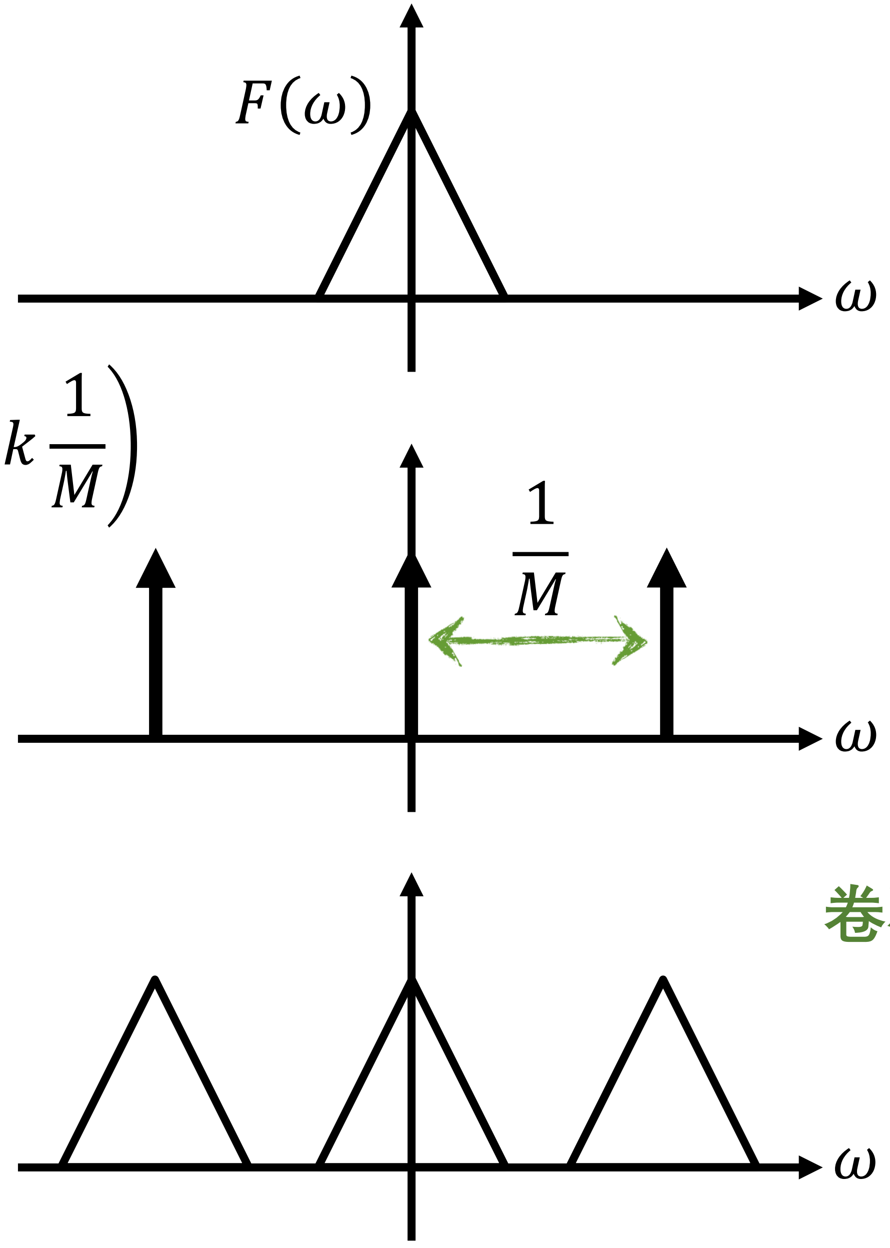


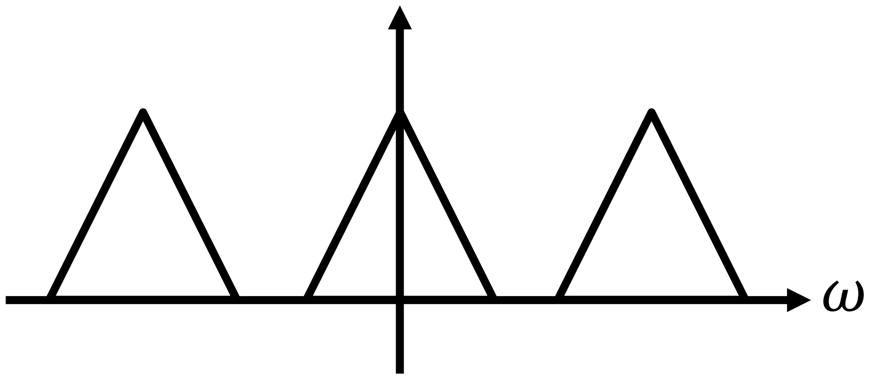


卷积

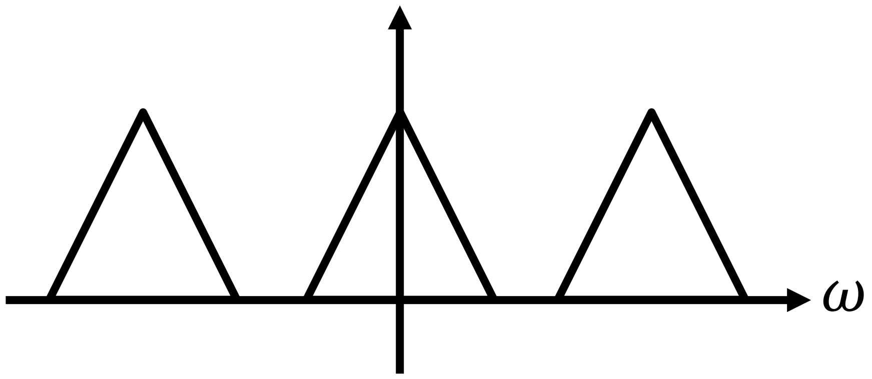
## 频域

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



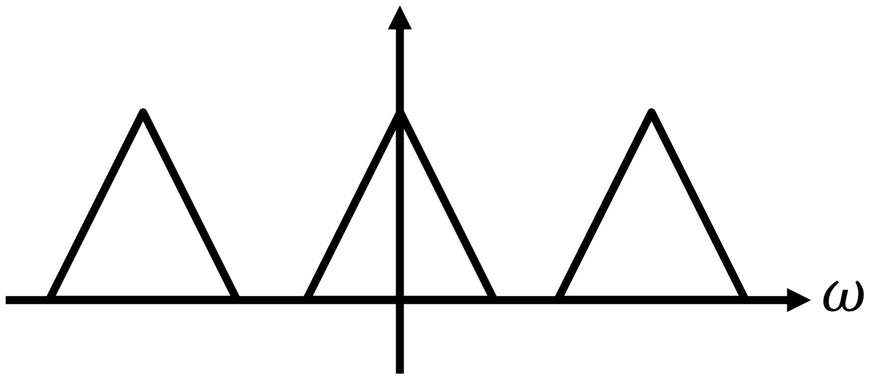


能准确恢复原始信号！



能准确恢复原始信号！

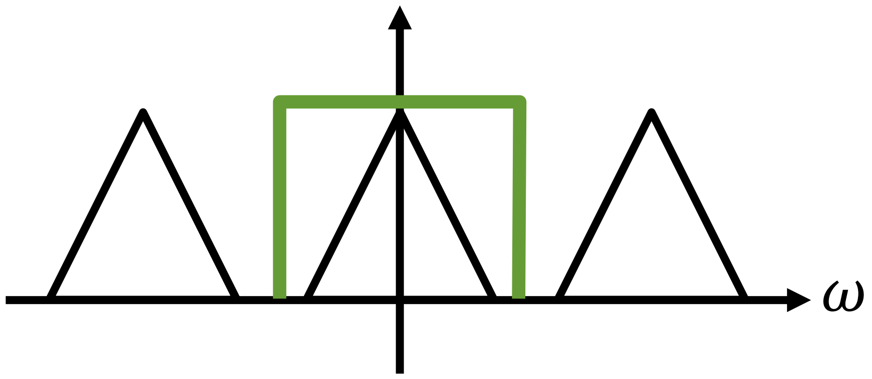
怎么做?



能准确恢复原始信号!

怎么做?

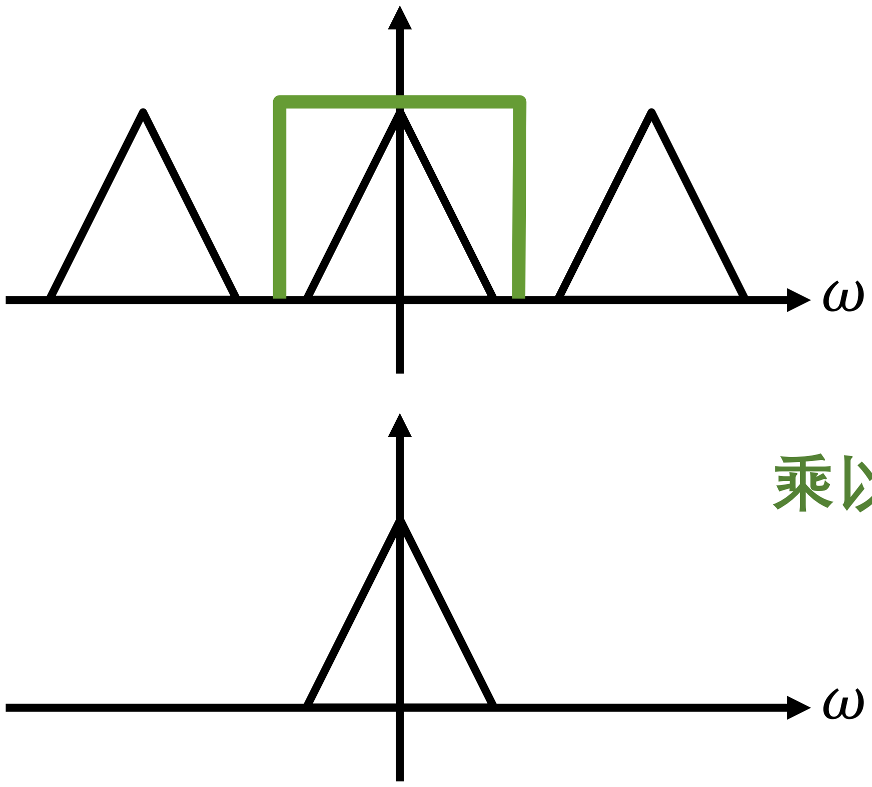
乘以方框滤波器



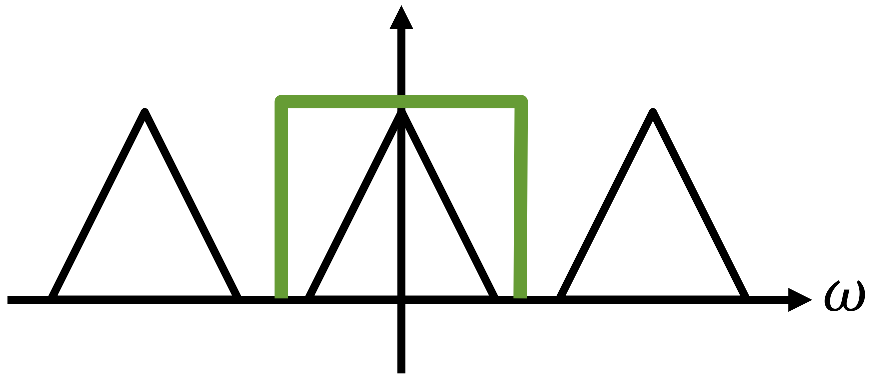
能准确恢复原始信号!

怎么做?

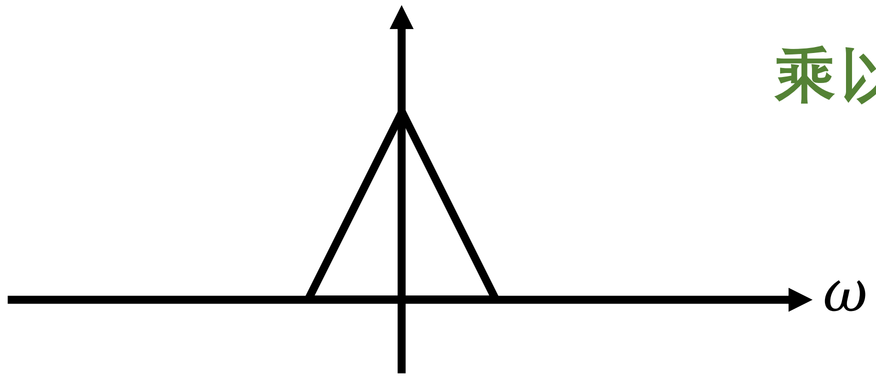
乘以方框滤波器



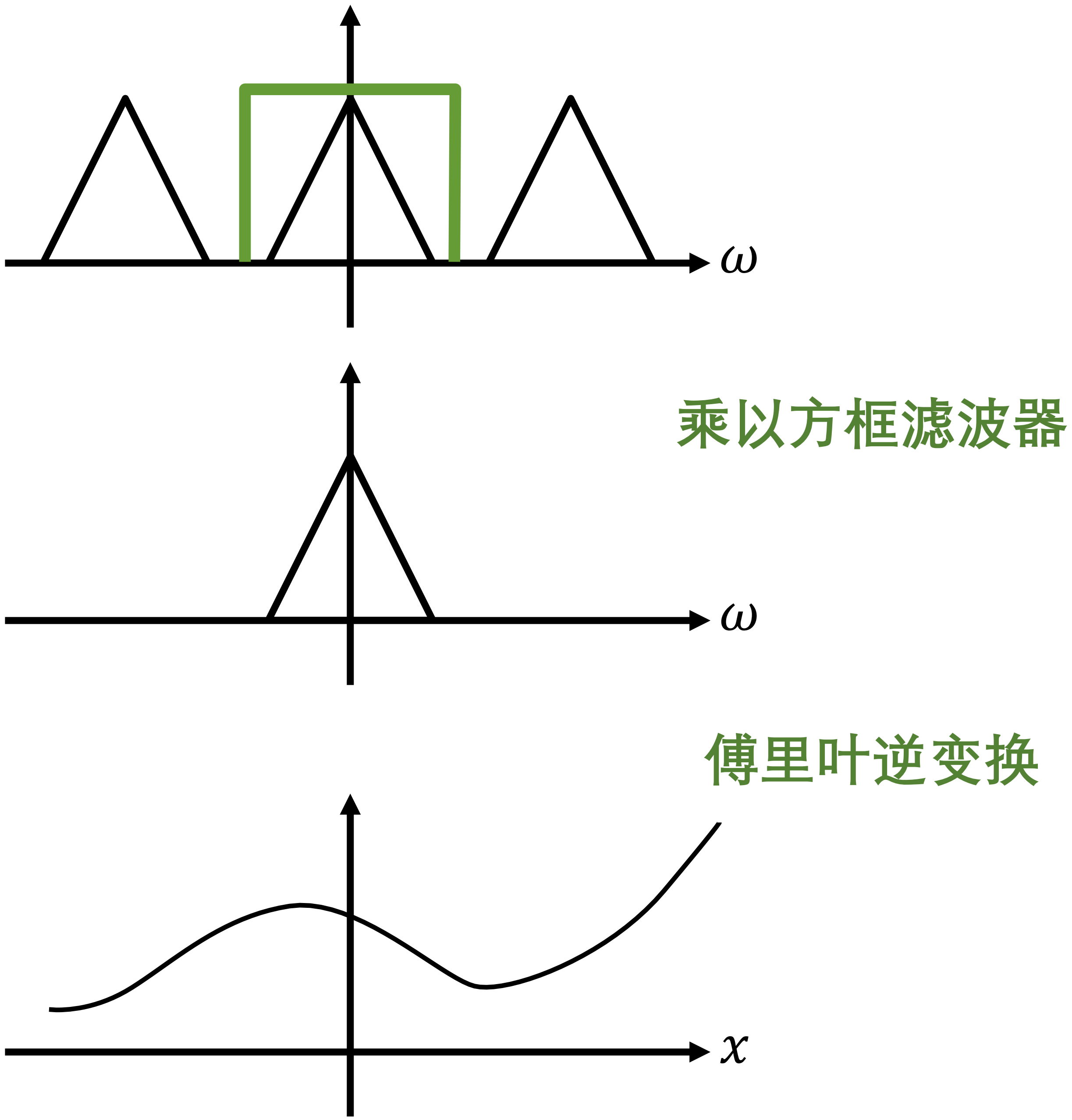
乘以方框滤波器



乘以方框滤波器

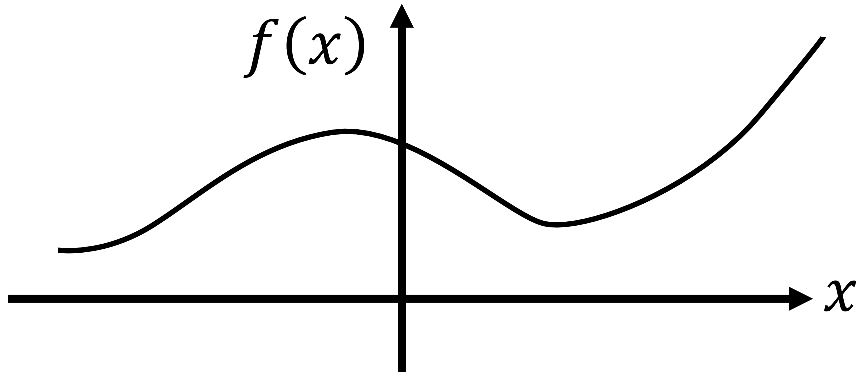


傅里叶逆变换

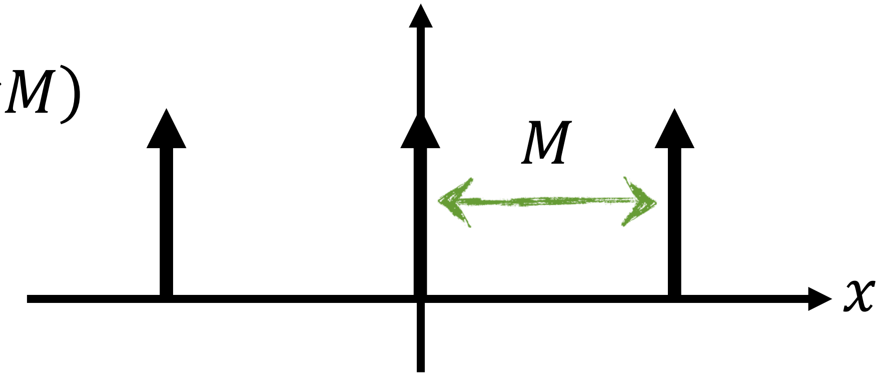

ω
乘以方框滤波器
ω
傅里叶逆变换
x

## 如果我们降低采样率

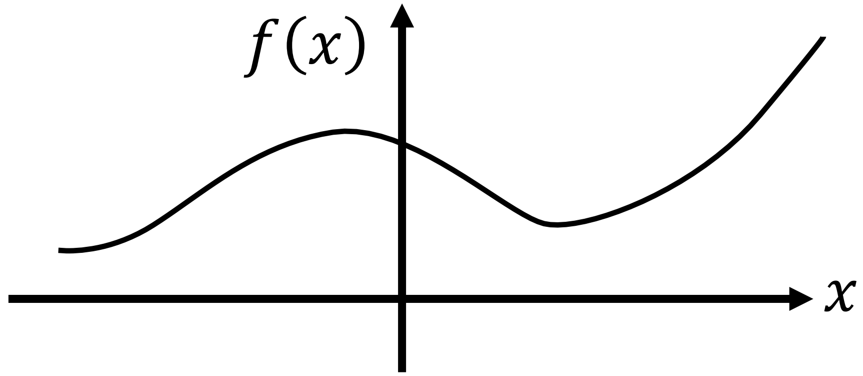
空间域



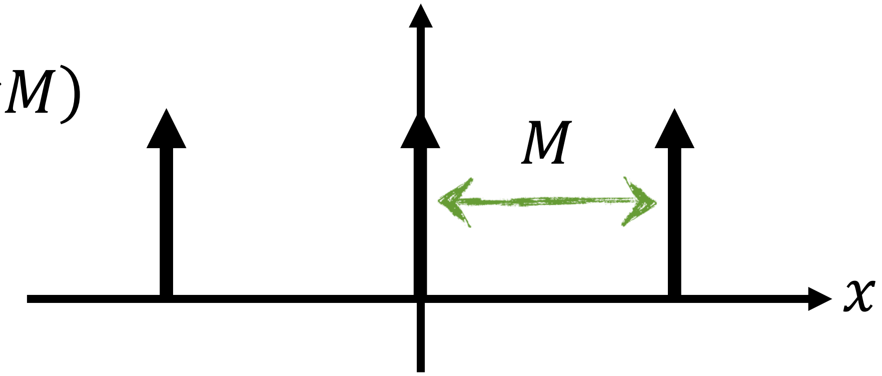
$$\text{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$



## 空间域



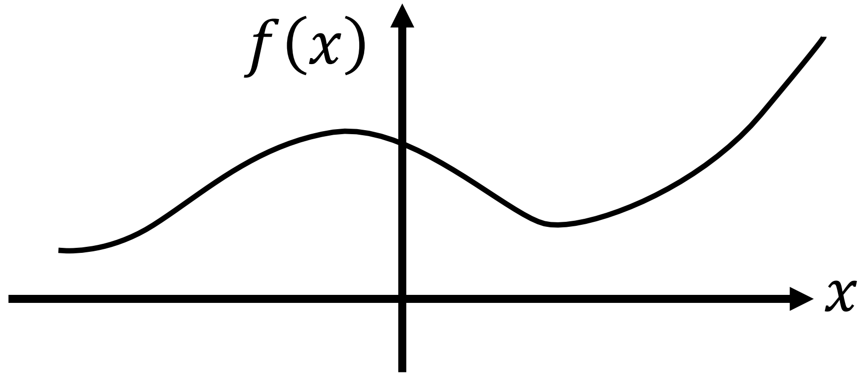
$$\mathrm{comb}(x) = \sum_{k=-\infty}^{\infty} \delta(x - kM)$$

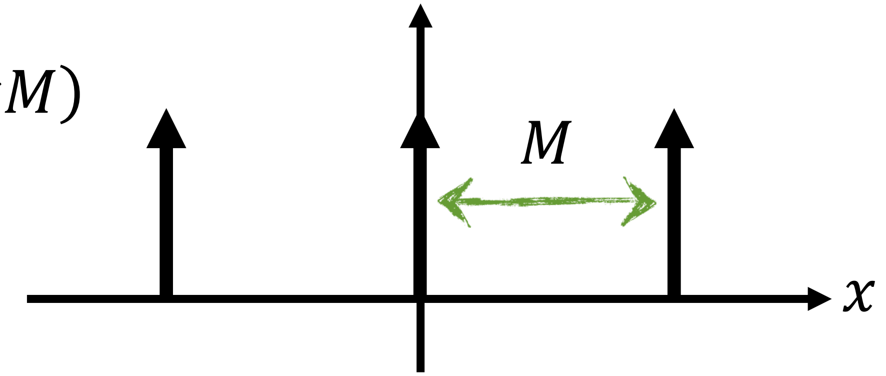


相乘

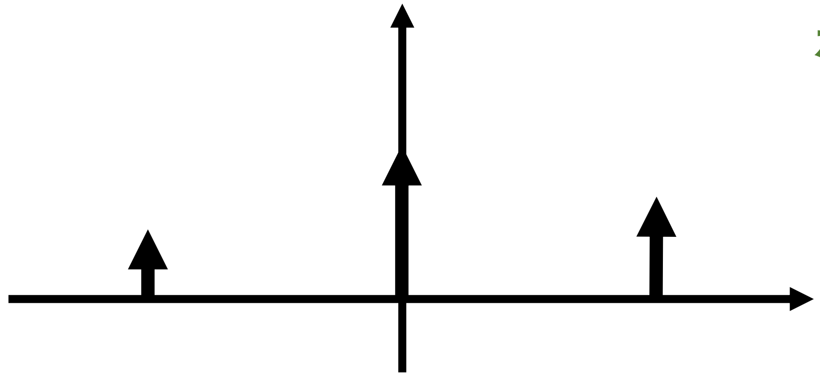
空间域

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

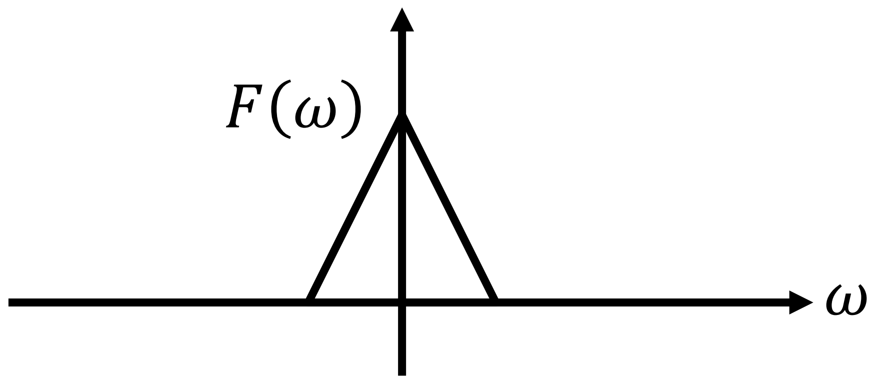




相乘

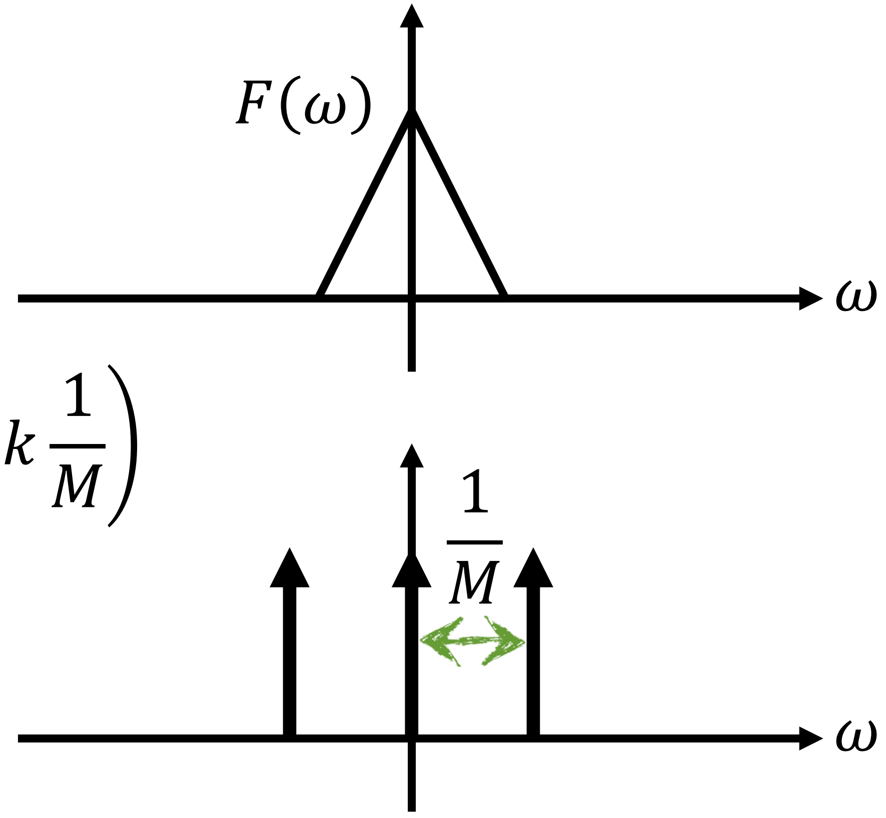


频域



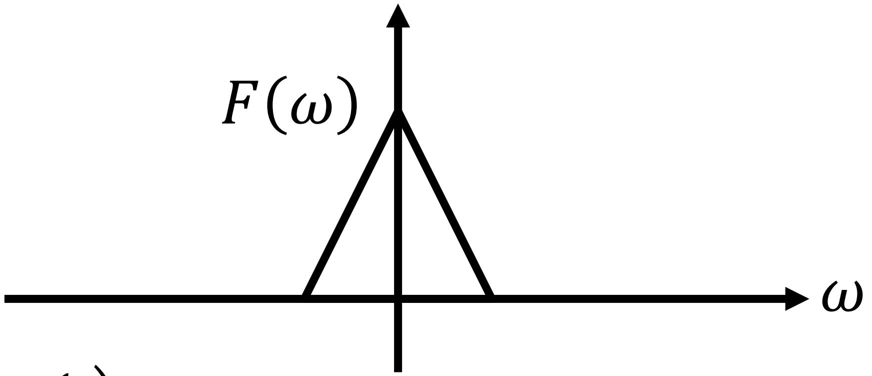
频域

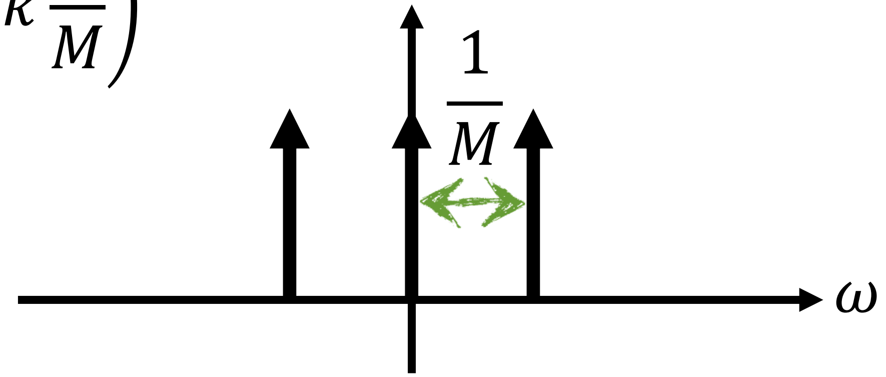
$$\text{comb}(\omega) = \frac{1}{M}\sum_{k=-\infty}^{\infty}\delta\left(\omega - k\frac{1}{M}\right)$$



## 频域

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

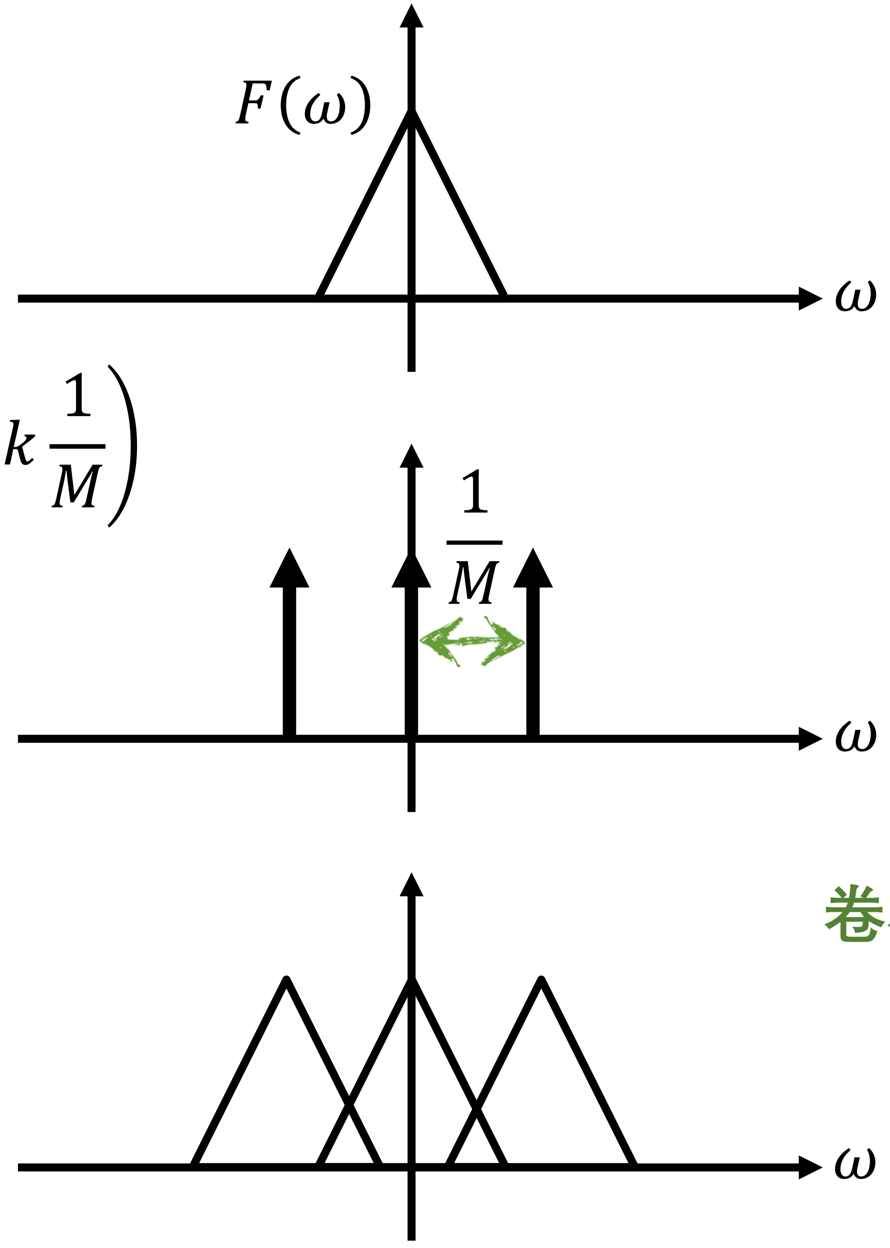


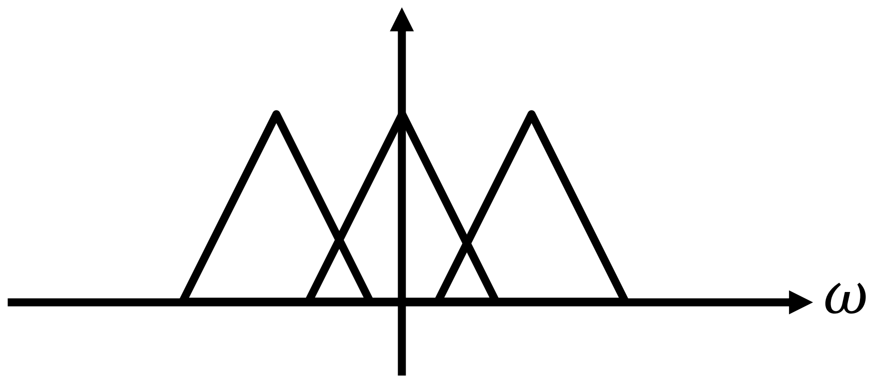


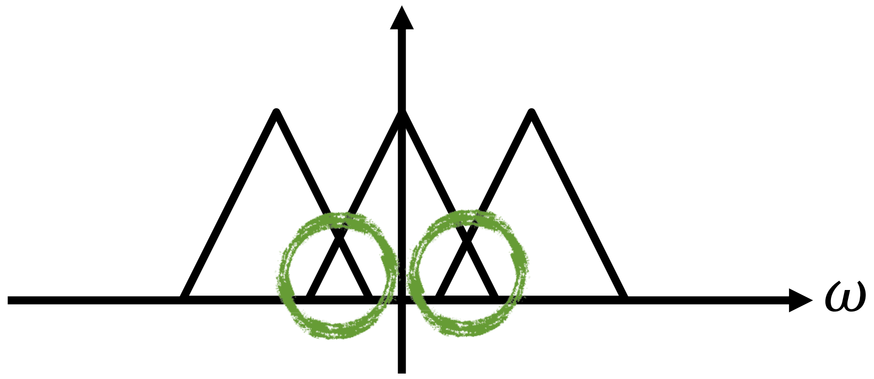
卷积

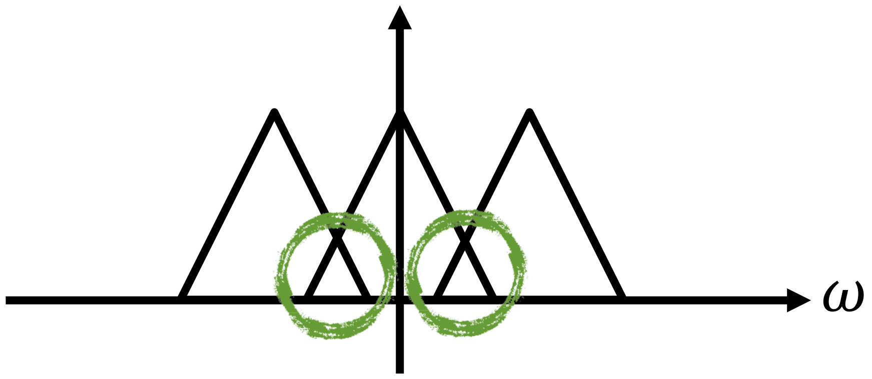
## 频域

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



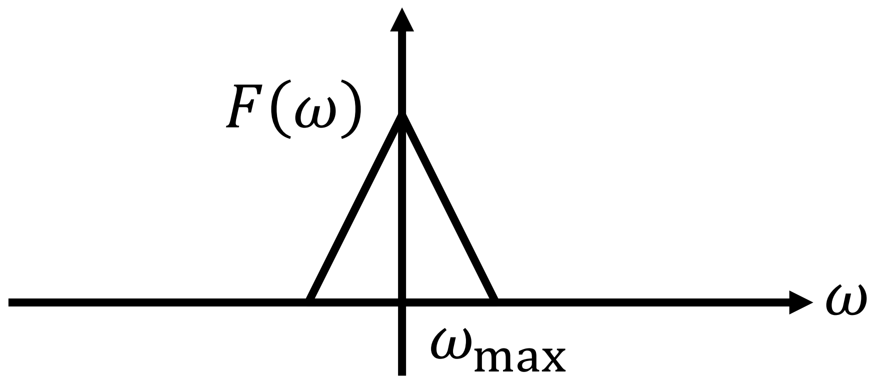

$\omega$

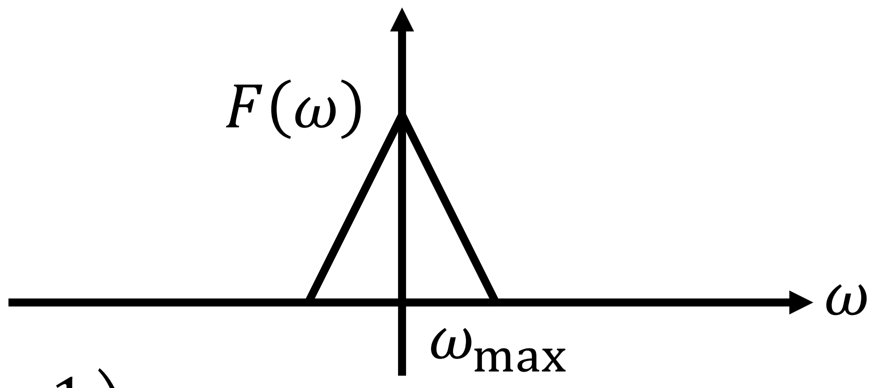

$\omega$



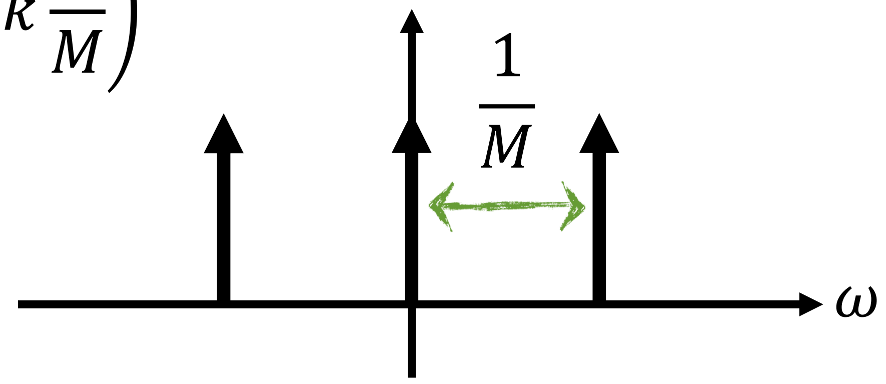
信号**混叠**了！

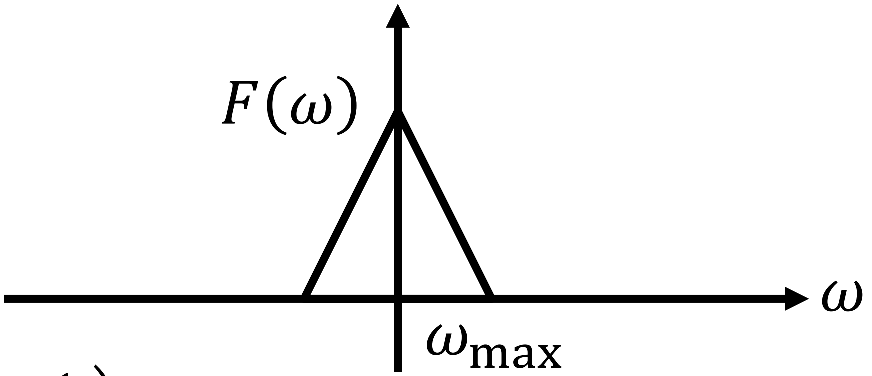
## 奈奎斯特
## 采样定理


$F(\omega)$
$\omega$
$\omega_{\max}$

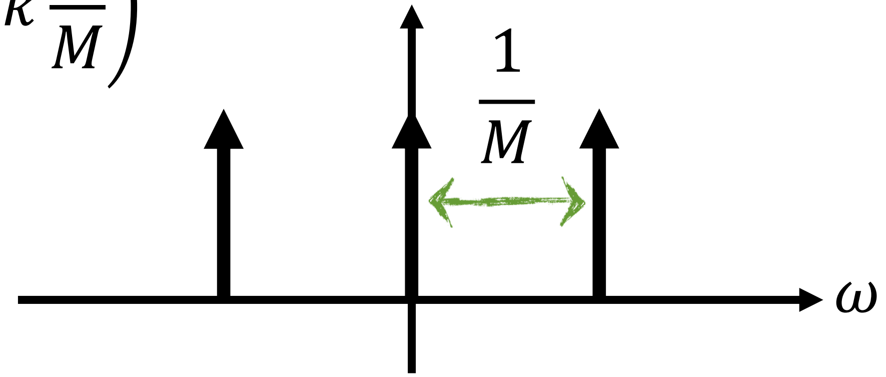


$$\mathrm{comb}(\omega) = \frac{1}{M}\sum_{k=-\infty}^{\infty}\delta\left(\omega - k\,\frac{1}{M}\right)$$



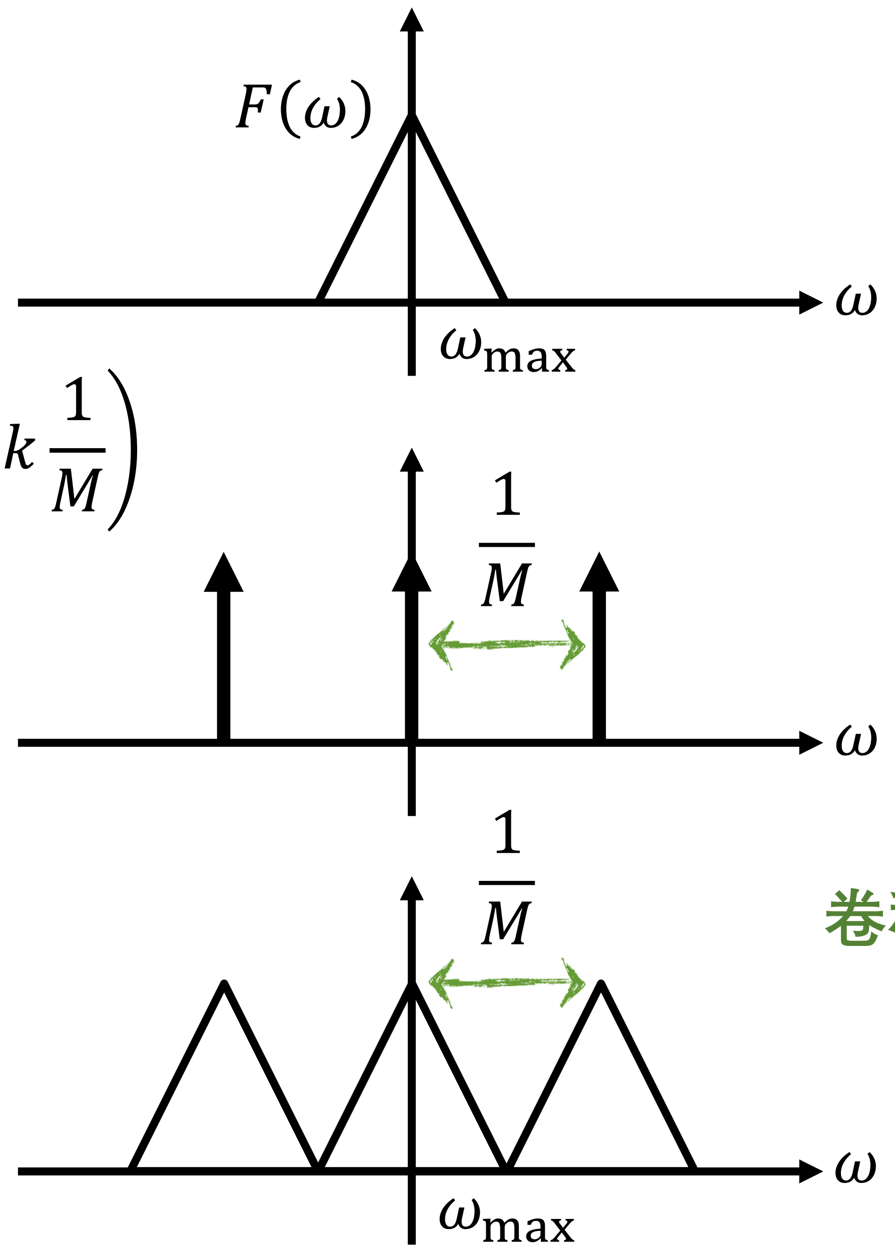


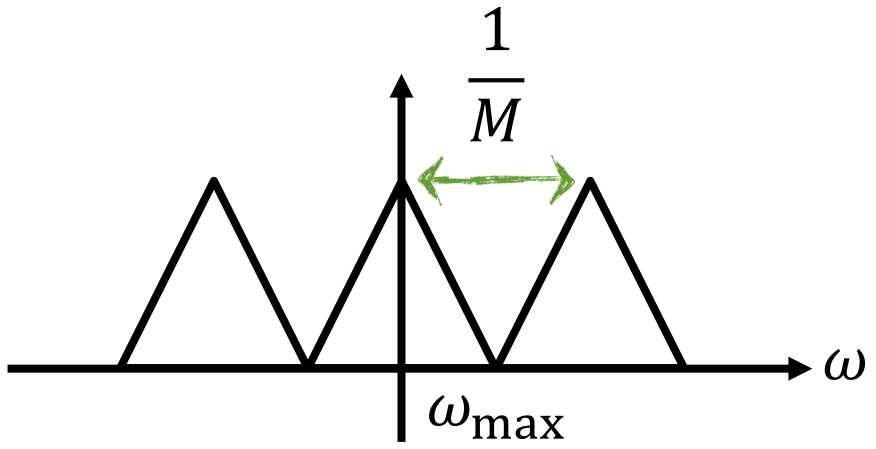
$$\mathrm{comb}(\omega) = \frac{1}{M}\sum_{k=-\infty}^{\infty}\delta\left(\omega - k\frac{1}{M}\right)$$



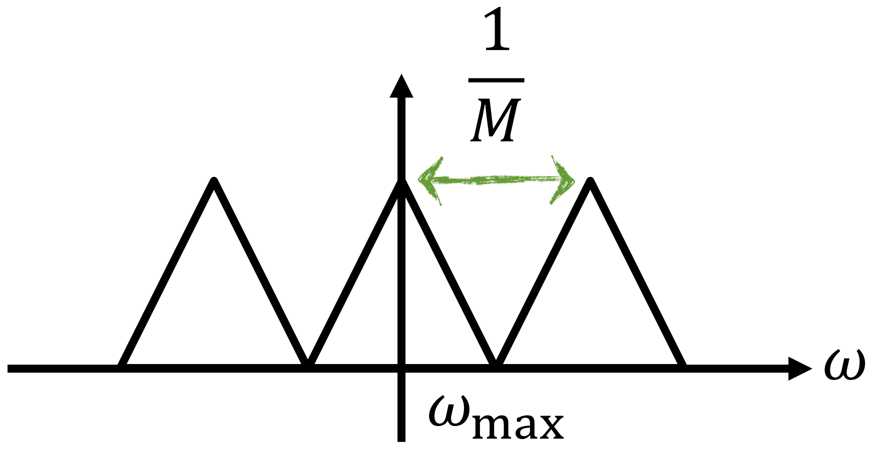
卷积

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

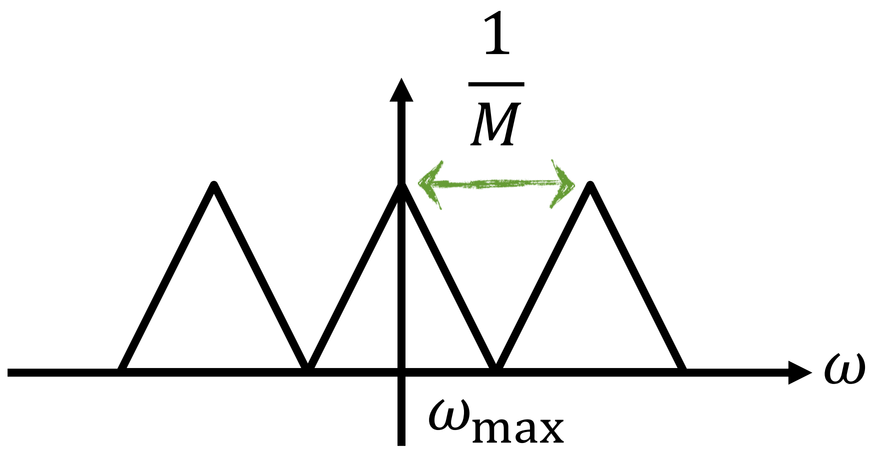




什么时候不出现**混叠**?



奈奎斯特采样定理 $$\frac{1}{M} \geq 2\omega_{\max}$$



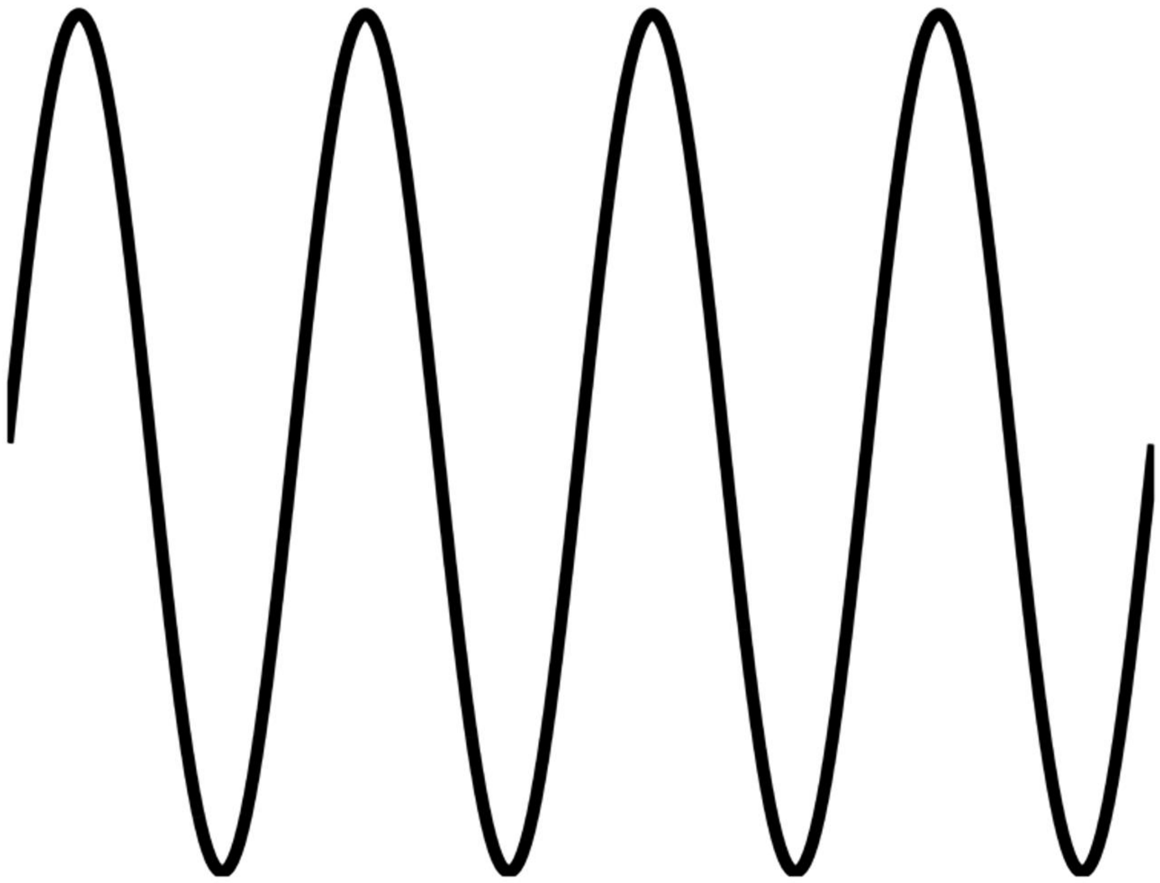
奈奎斯特采样定理 $$\frac{1}{M} \geq 2\omega_{\max}$$

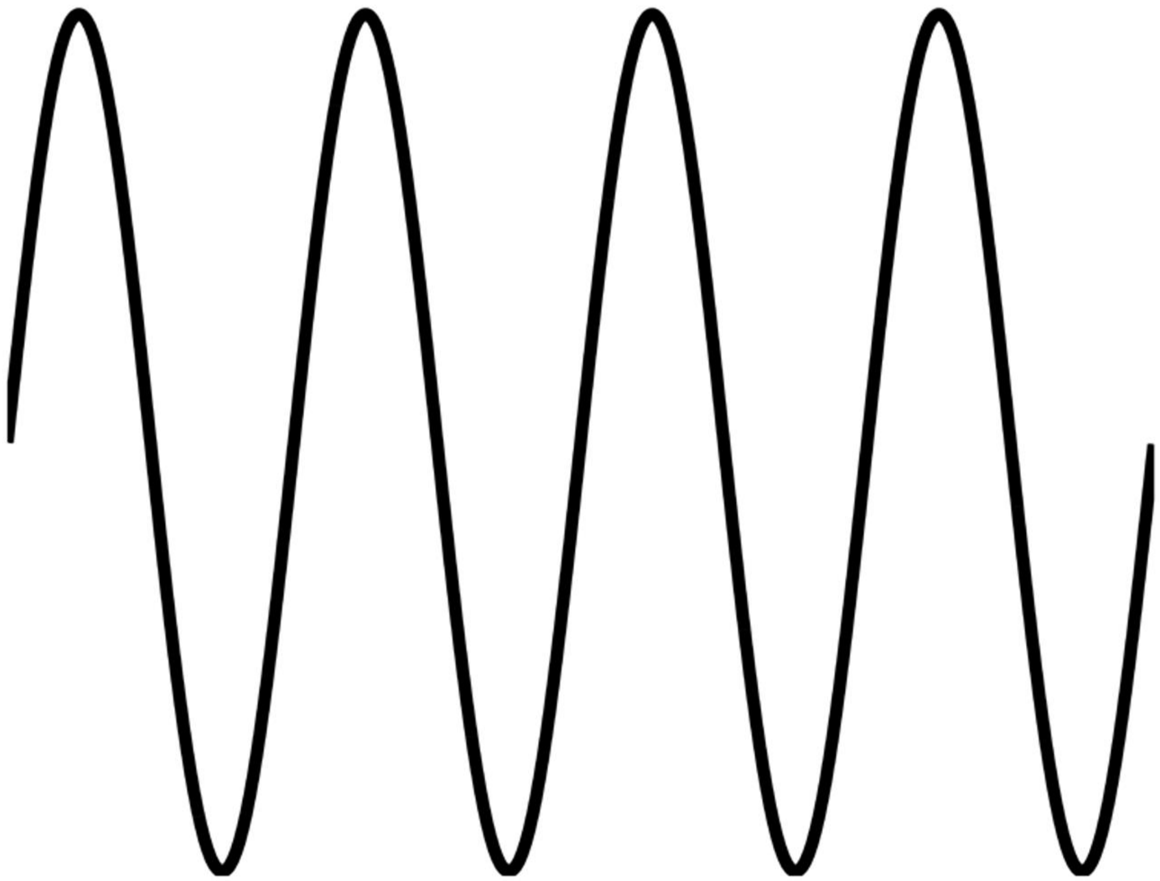
采样频率必须大于信号最高频率的两倍

## 如何避免
## 混叠？

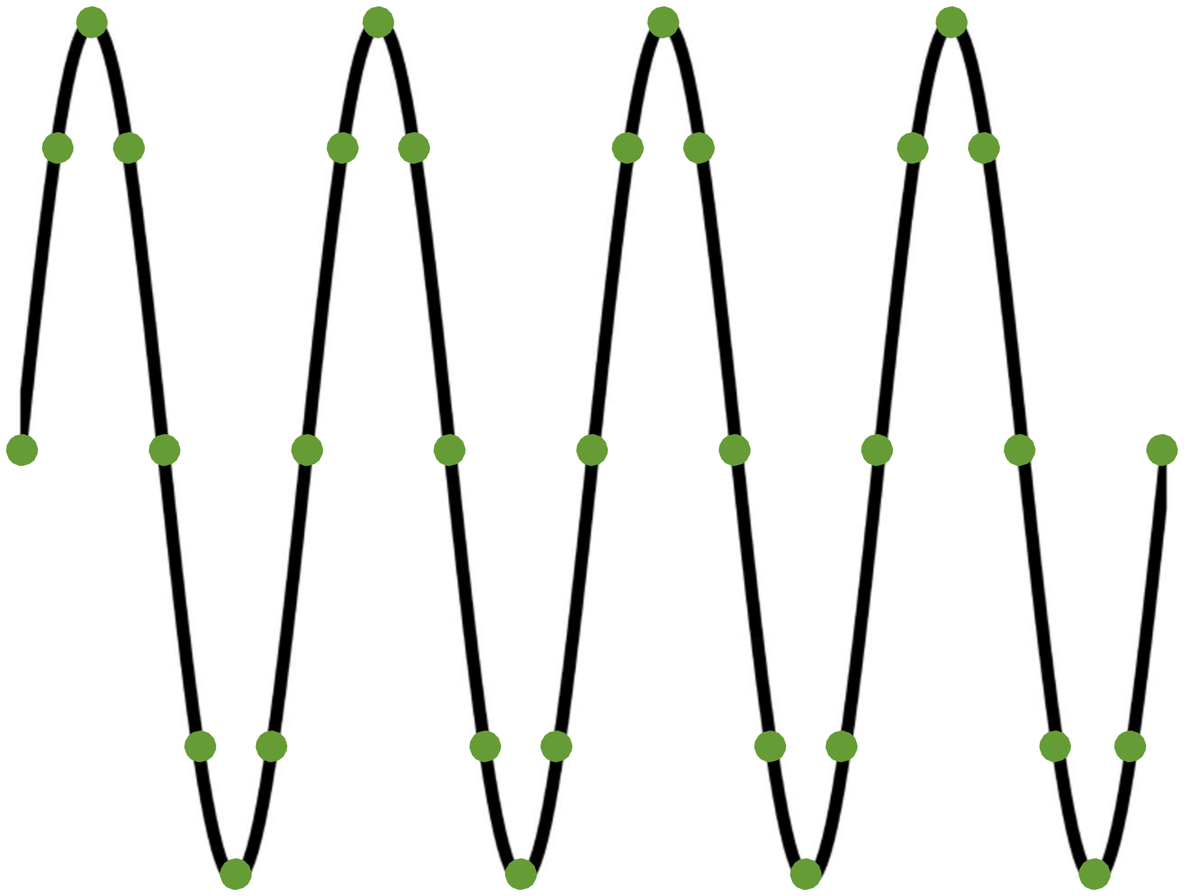
## 步骤1

调整采样率

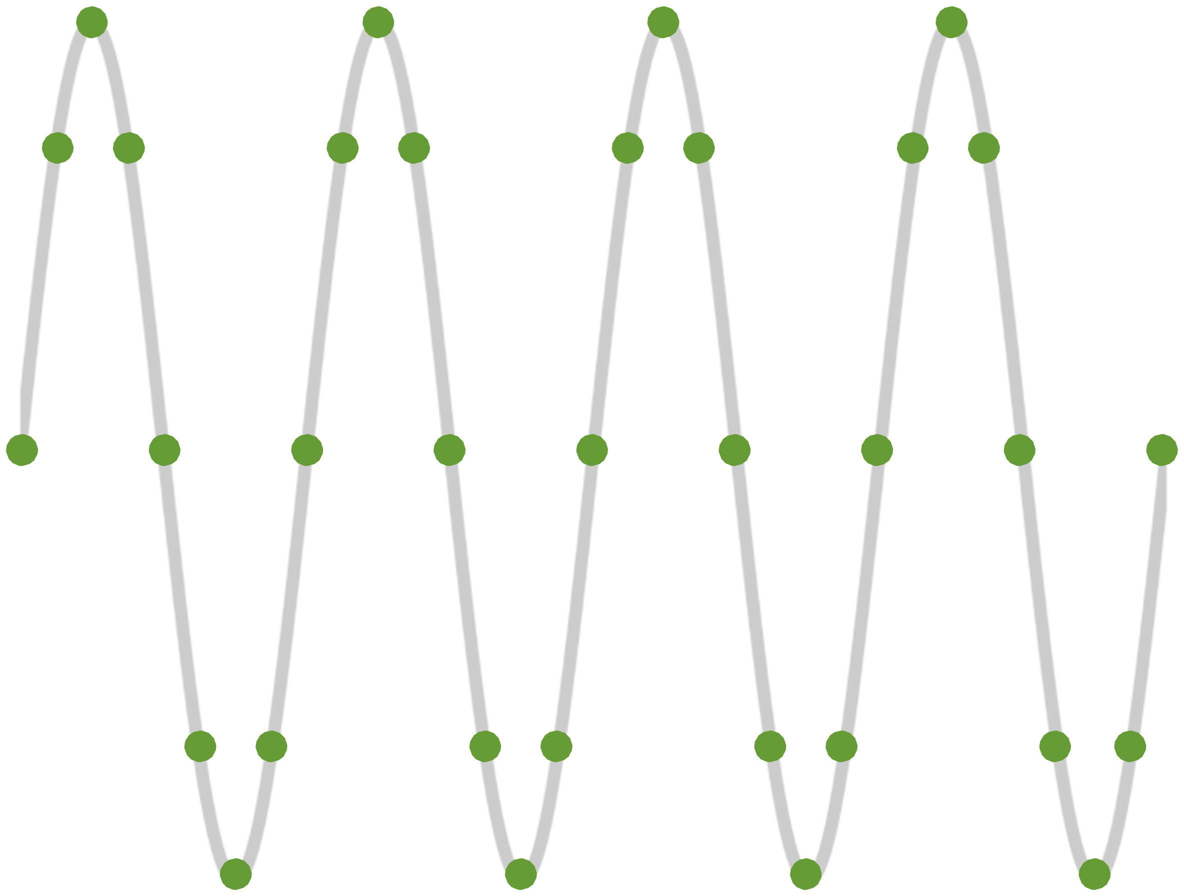




用更高频率采样

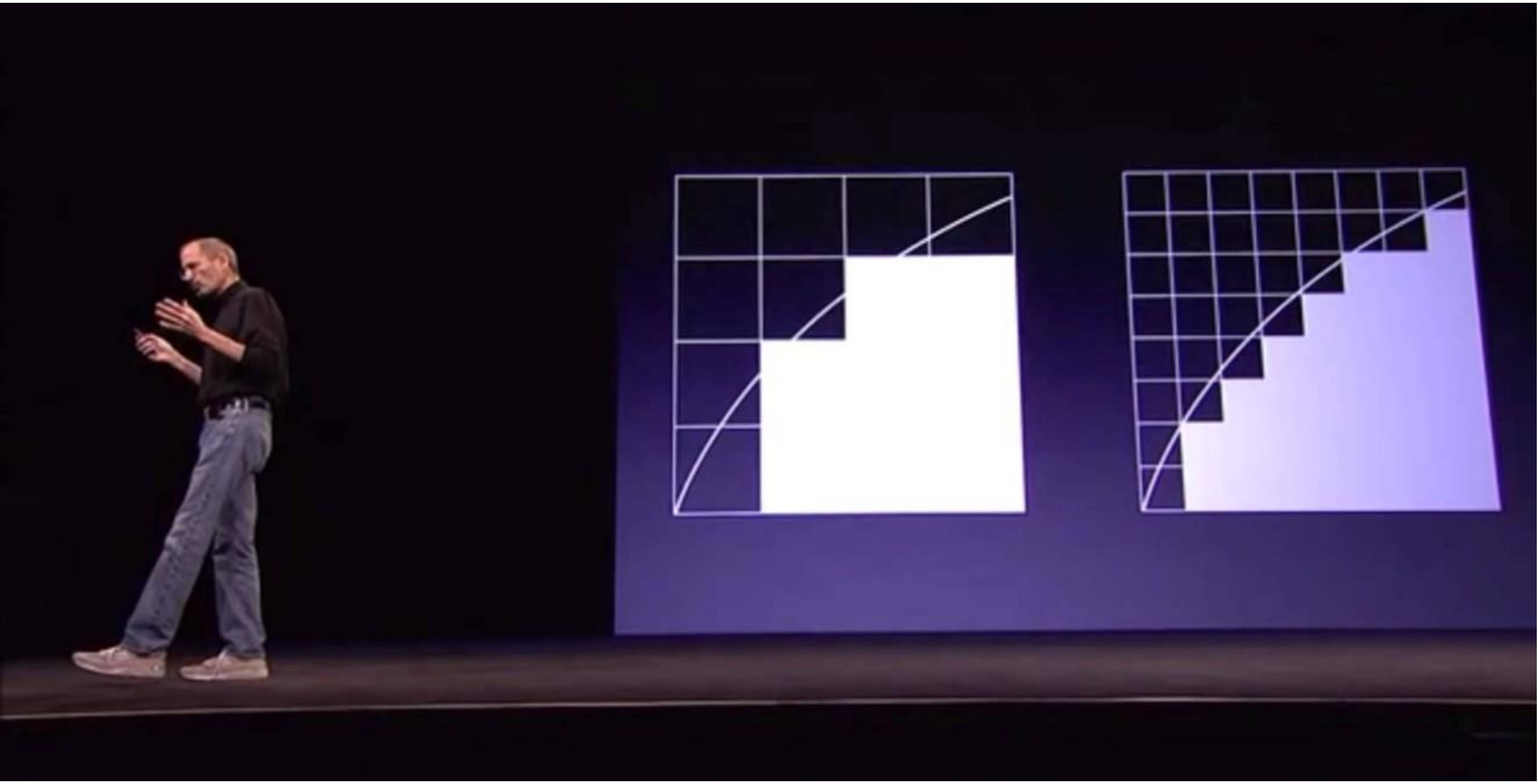


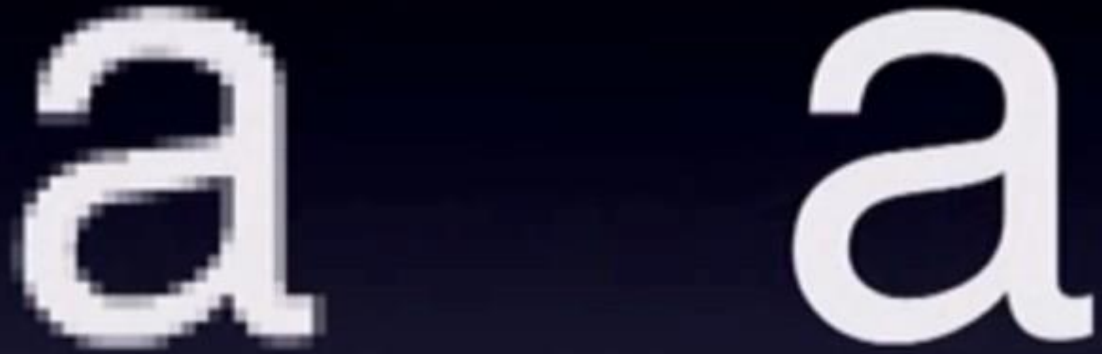
用更高频率采样

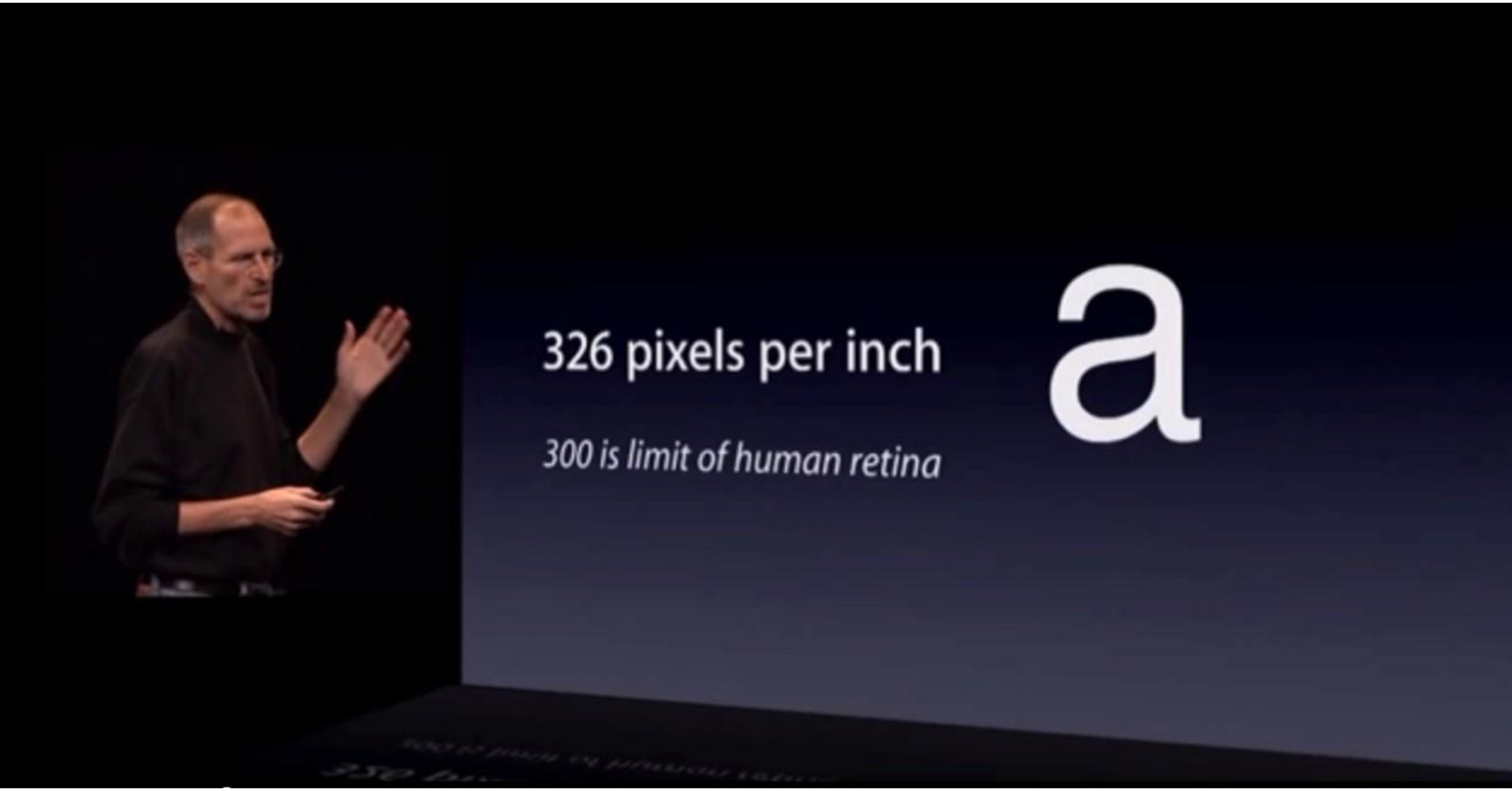


用更高频率采样

Retina display




a
a



326 pixels per inch

*300 is limit of human retina*

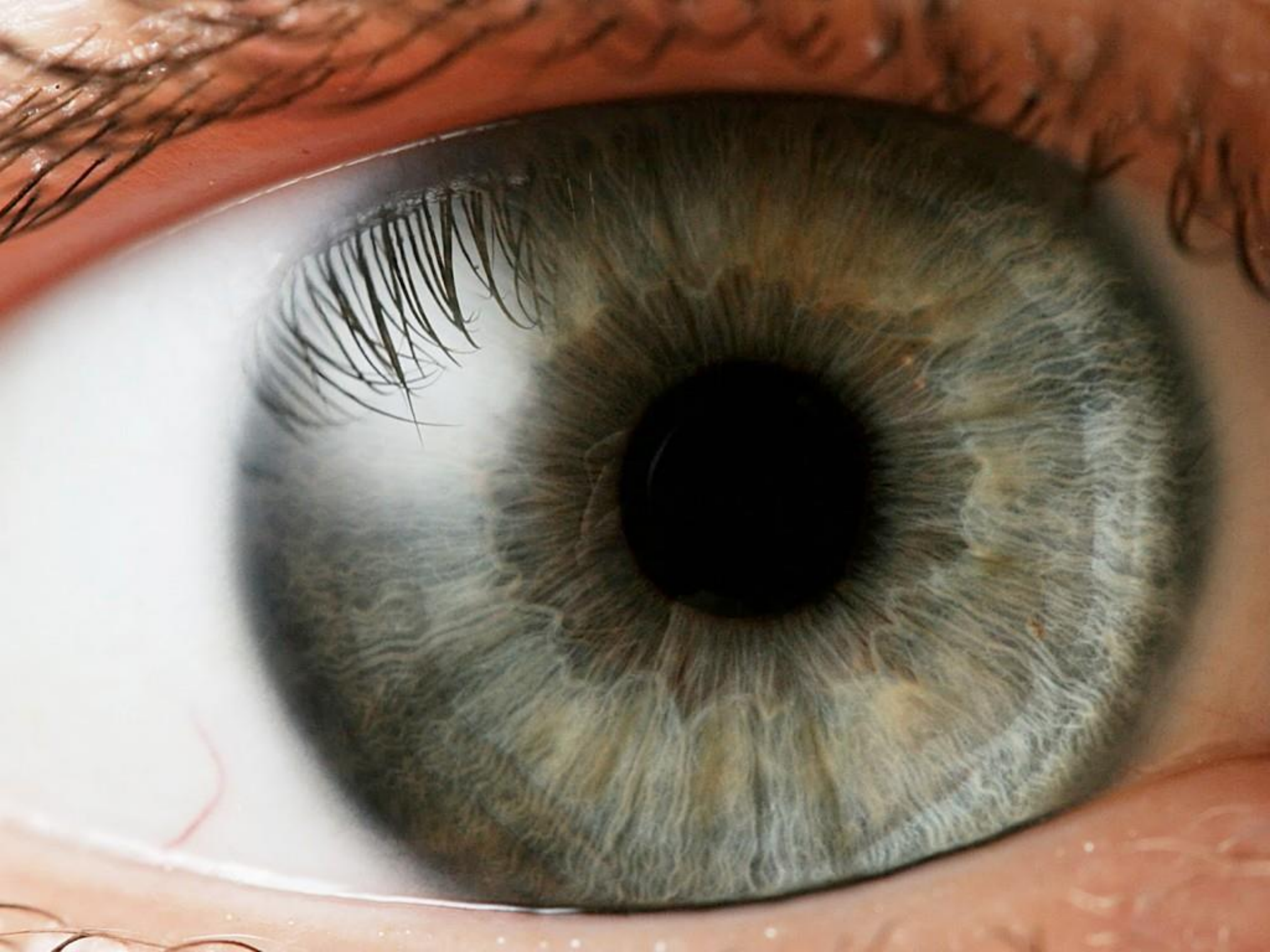
## 步骤2

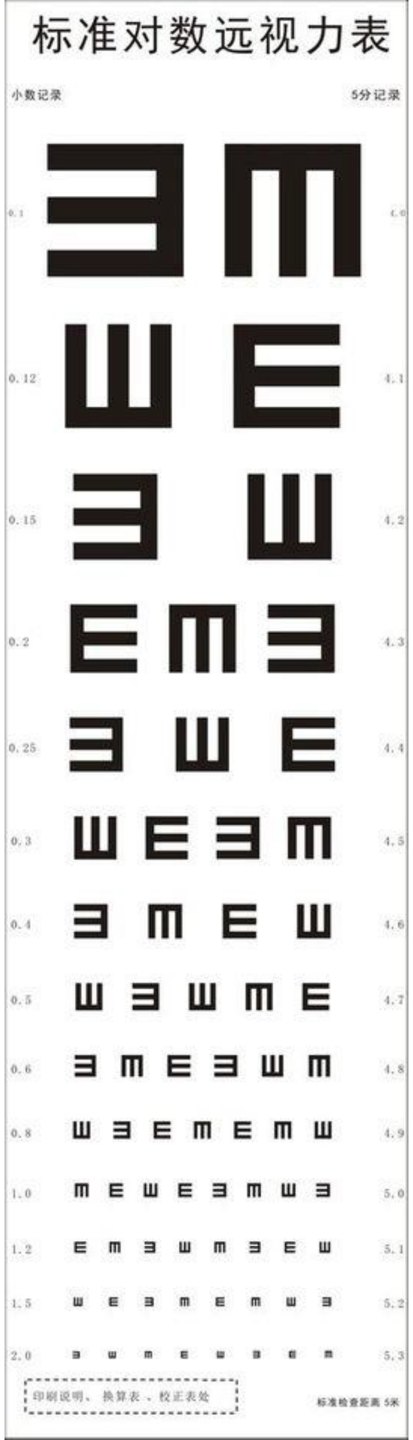
预滤波图像

## 去除一些高频成分

## 去除一些高频成分

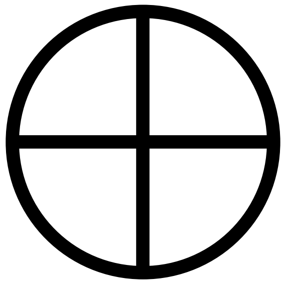
在采样前平滑信号



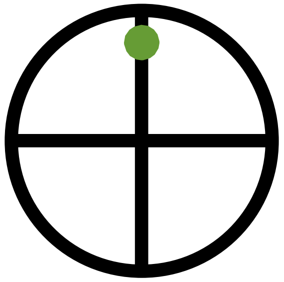

标准对数远视力表
小数记录
5分记录
0.1
4.0
0.12
4.1
0.15
4.2
0.2
4.3
0.25
4.4
0.3
4.5
0.4
4.6
0.5
4.7
0.6
4.8
0.8
4.9
1.0
5.0
1.2
5.1
1.5
5.2
2.0
5.3
印刷说明、换算表、校正表处
标准检查距离 5米

透镜起到低通滤波器的作用

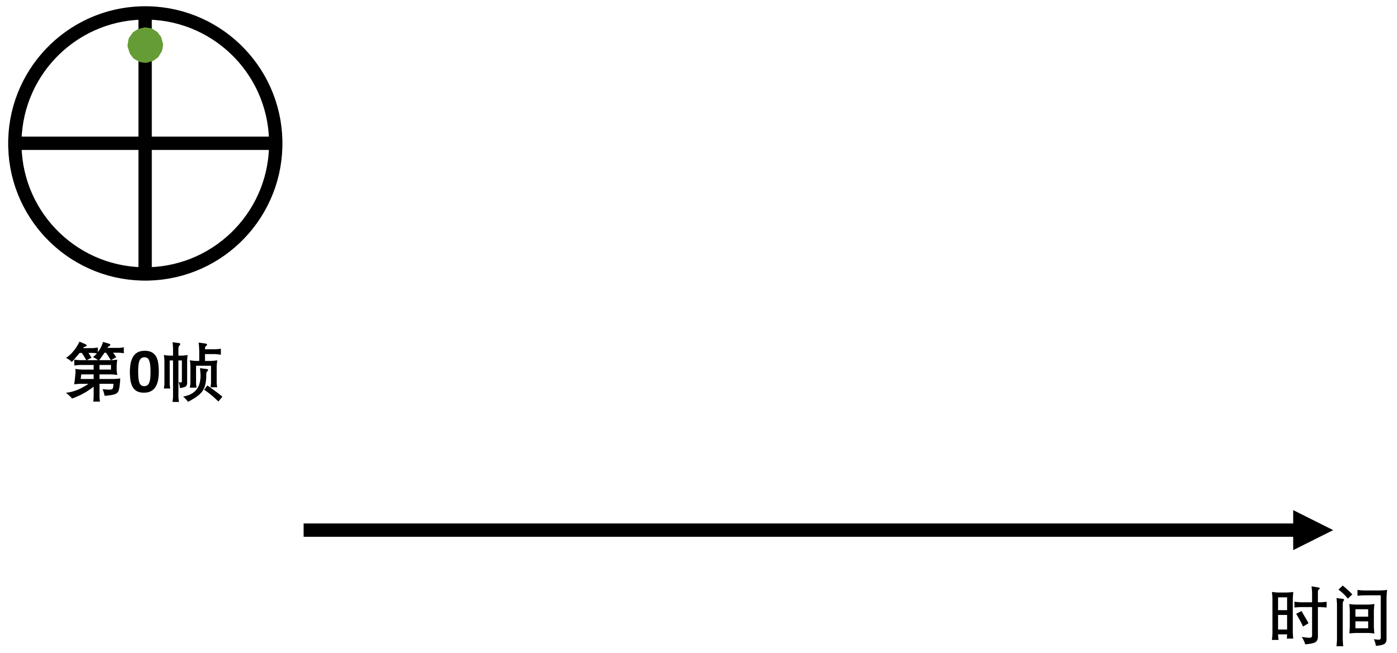
车轮是朝哪个方向转动的?

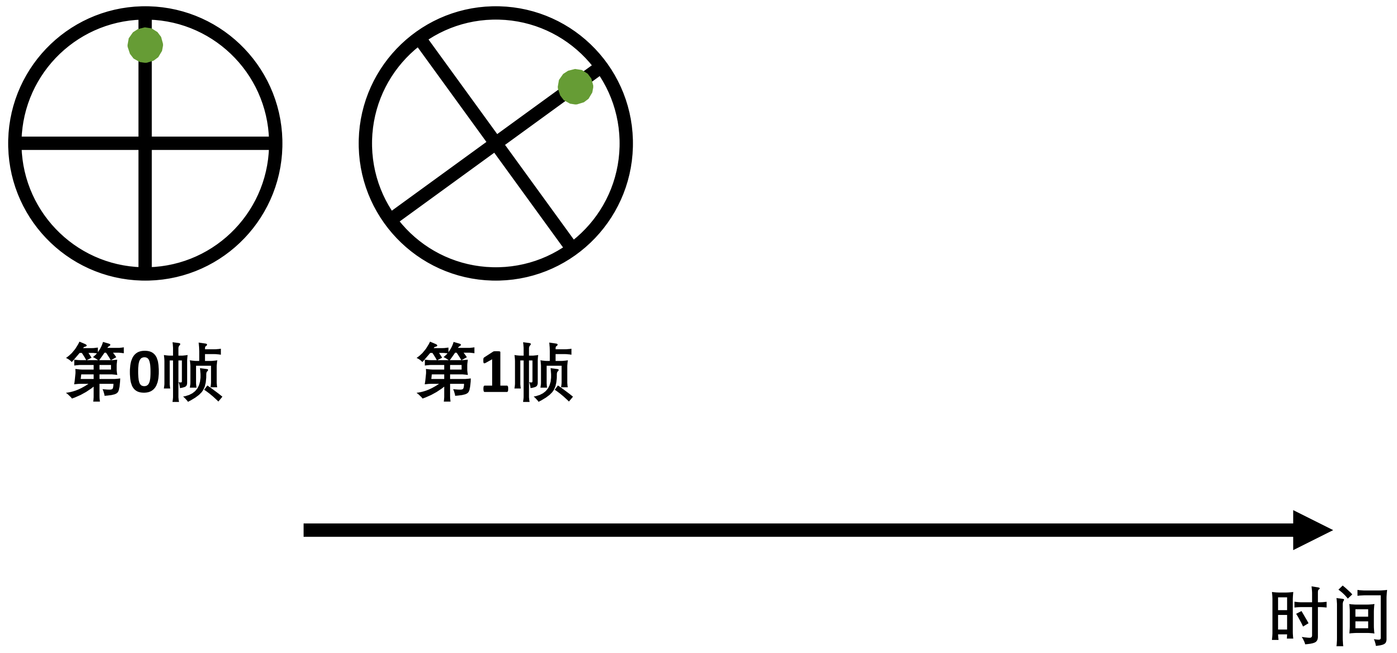


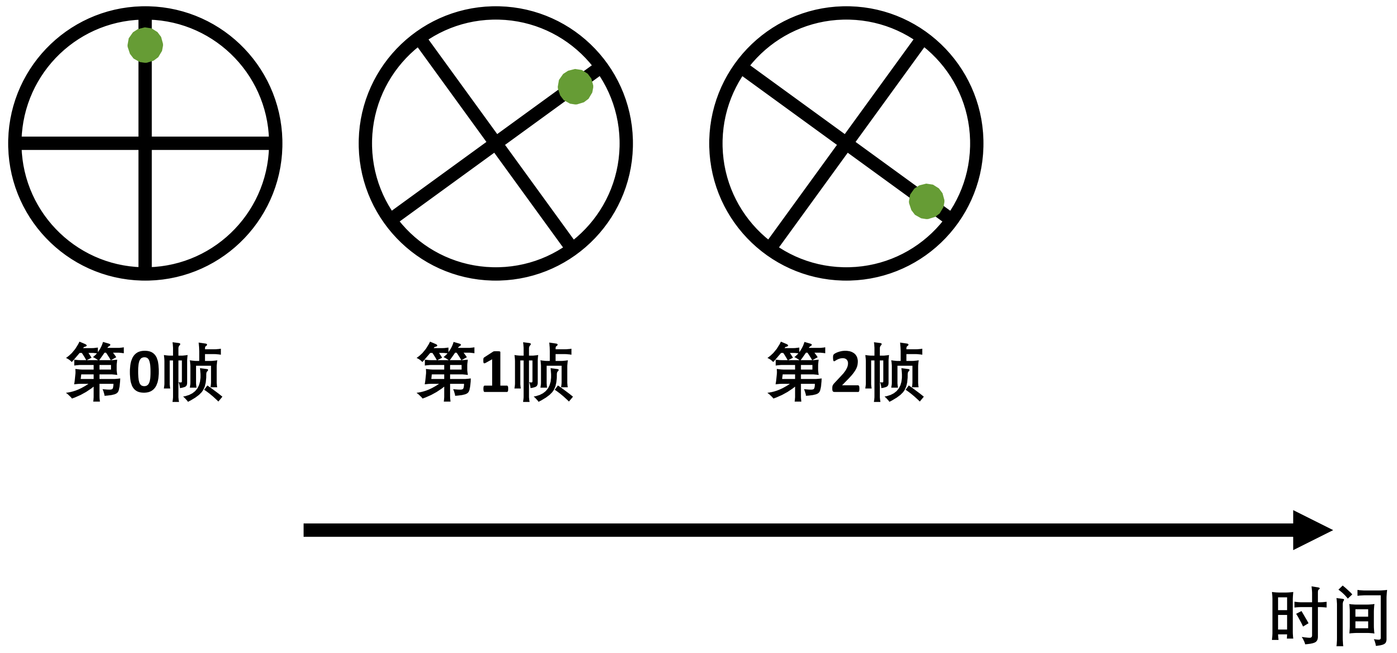
第0帧

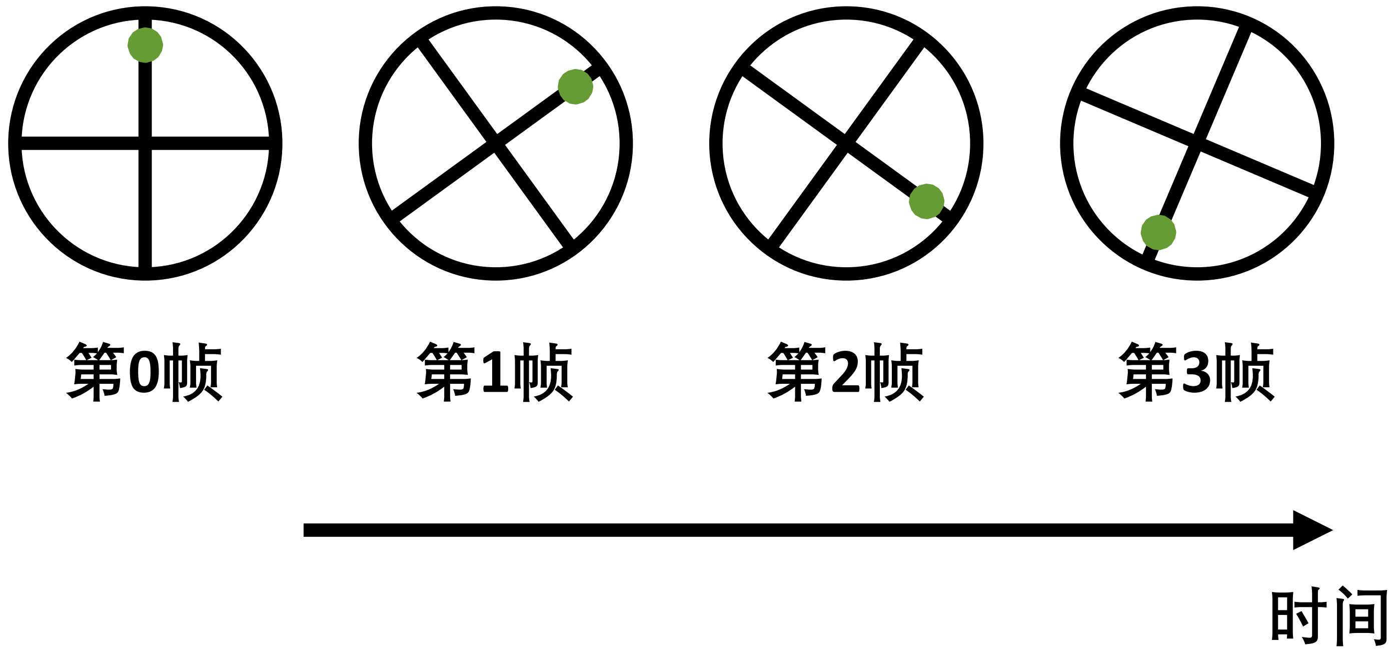


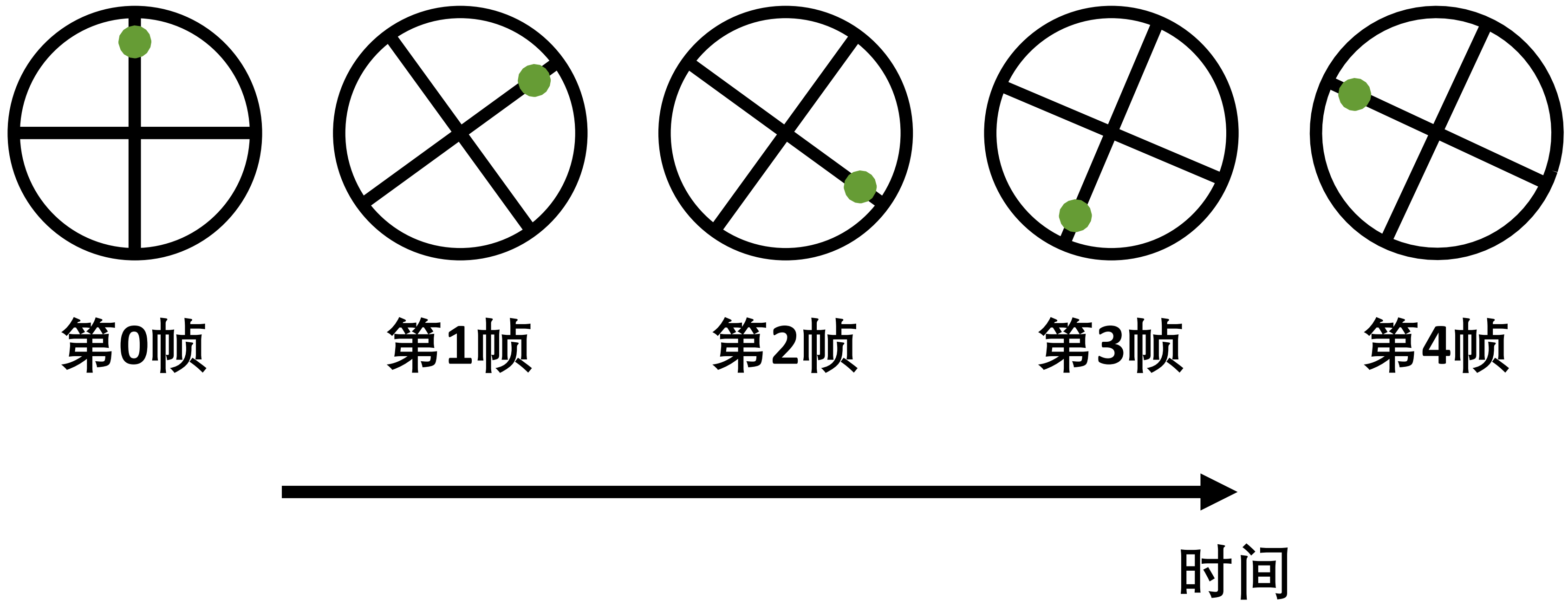
**第0帧**

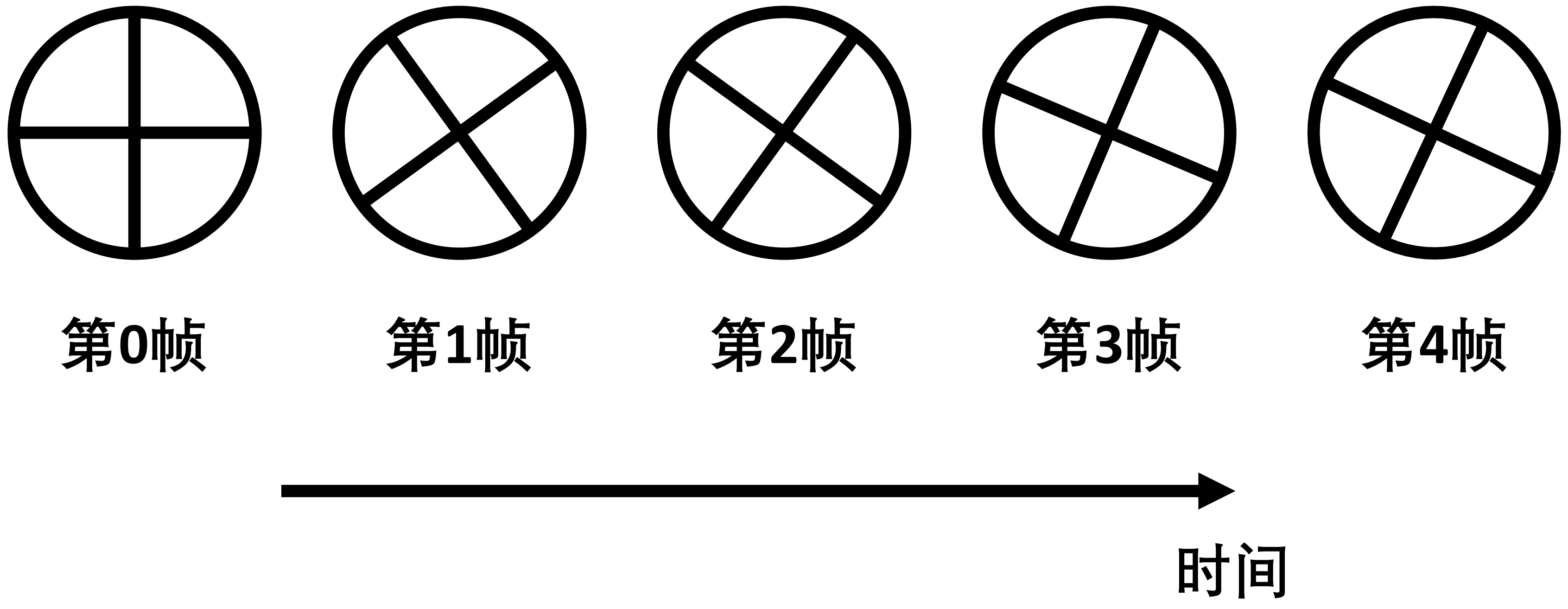

第0帧
时间

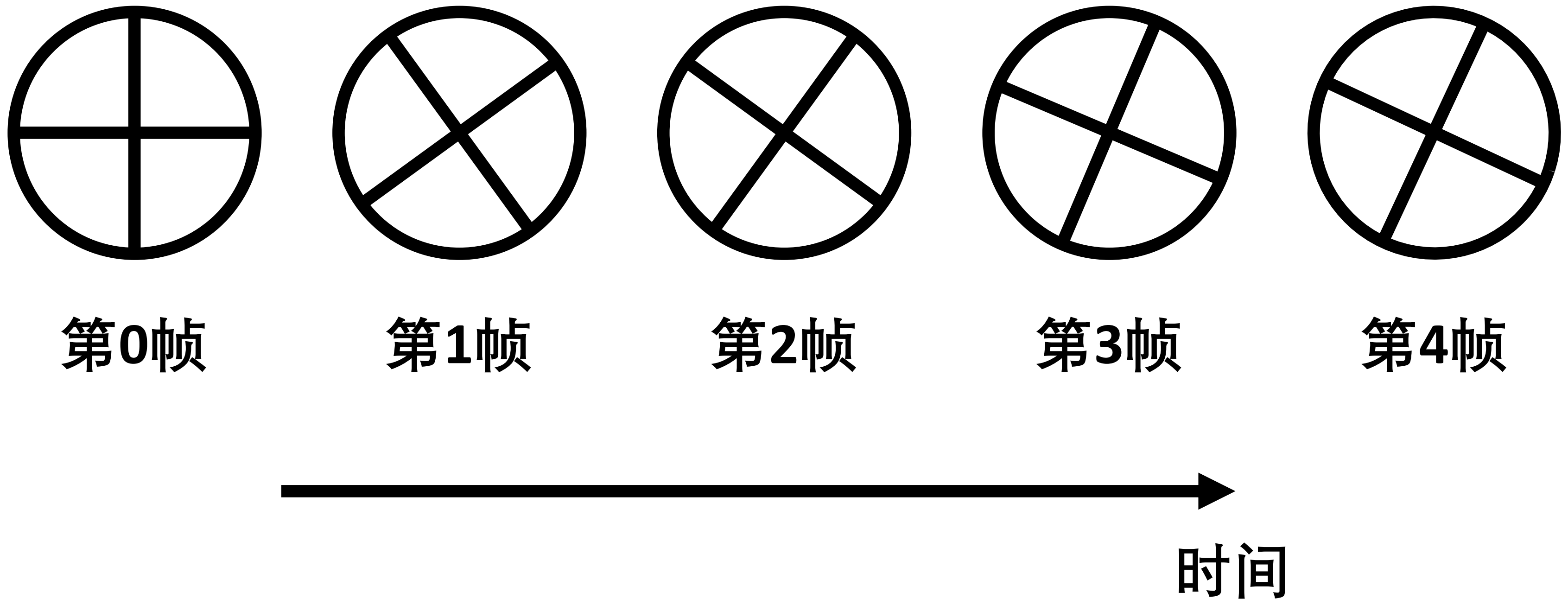

第0帧
第1帧
时间


第0帧
第1帧
第2帧
时间

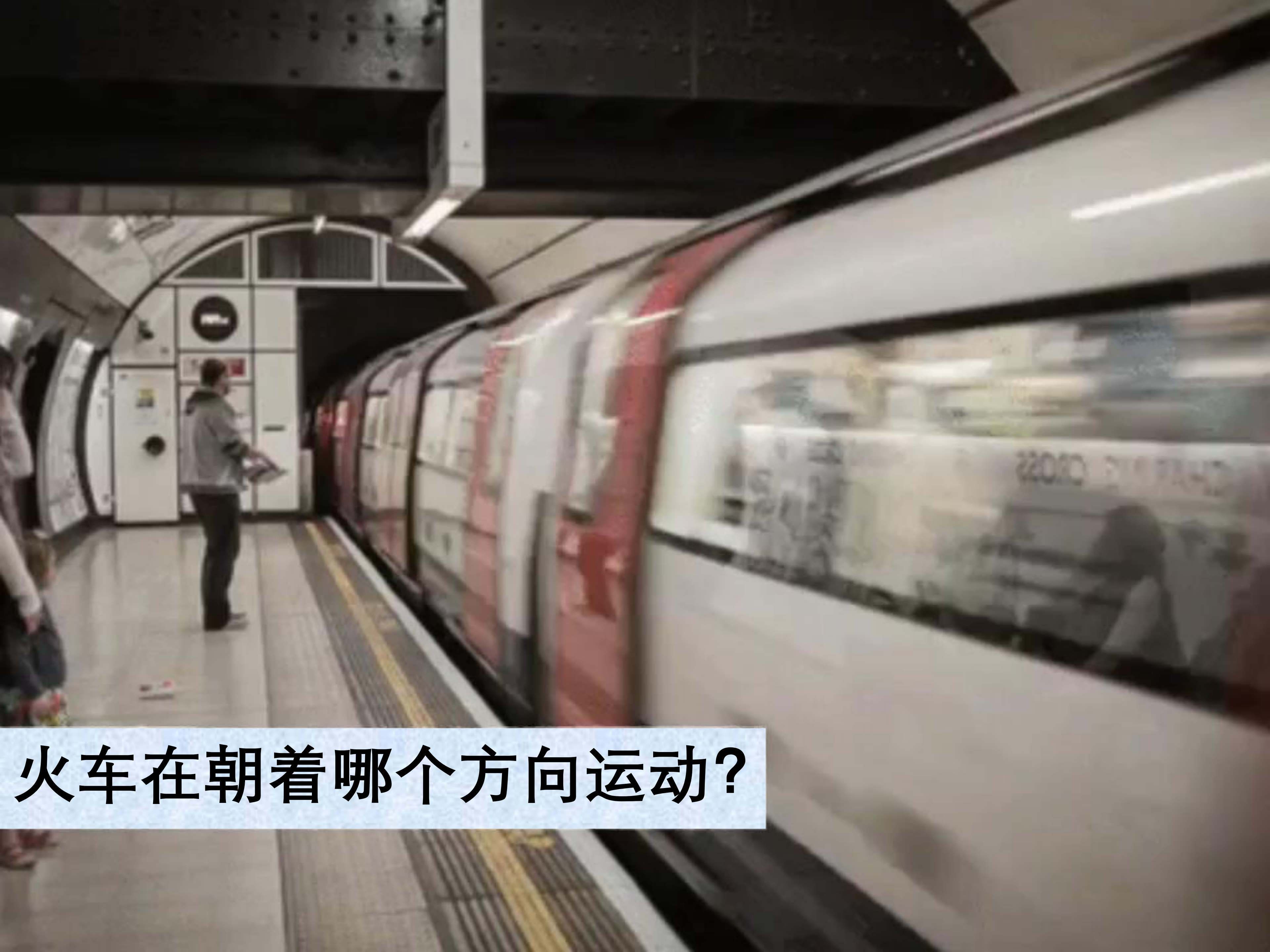

第0帧
第1帧
第2帧
第3帧
时间


第0帧
第1帧
第2帧
第3帧
第4帧
时间


第0帧
第1帧
第2帧
第3帧
第4帧
时间

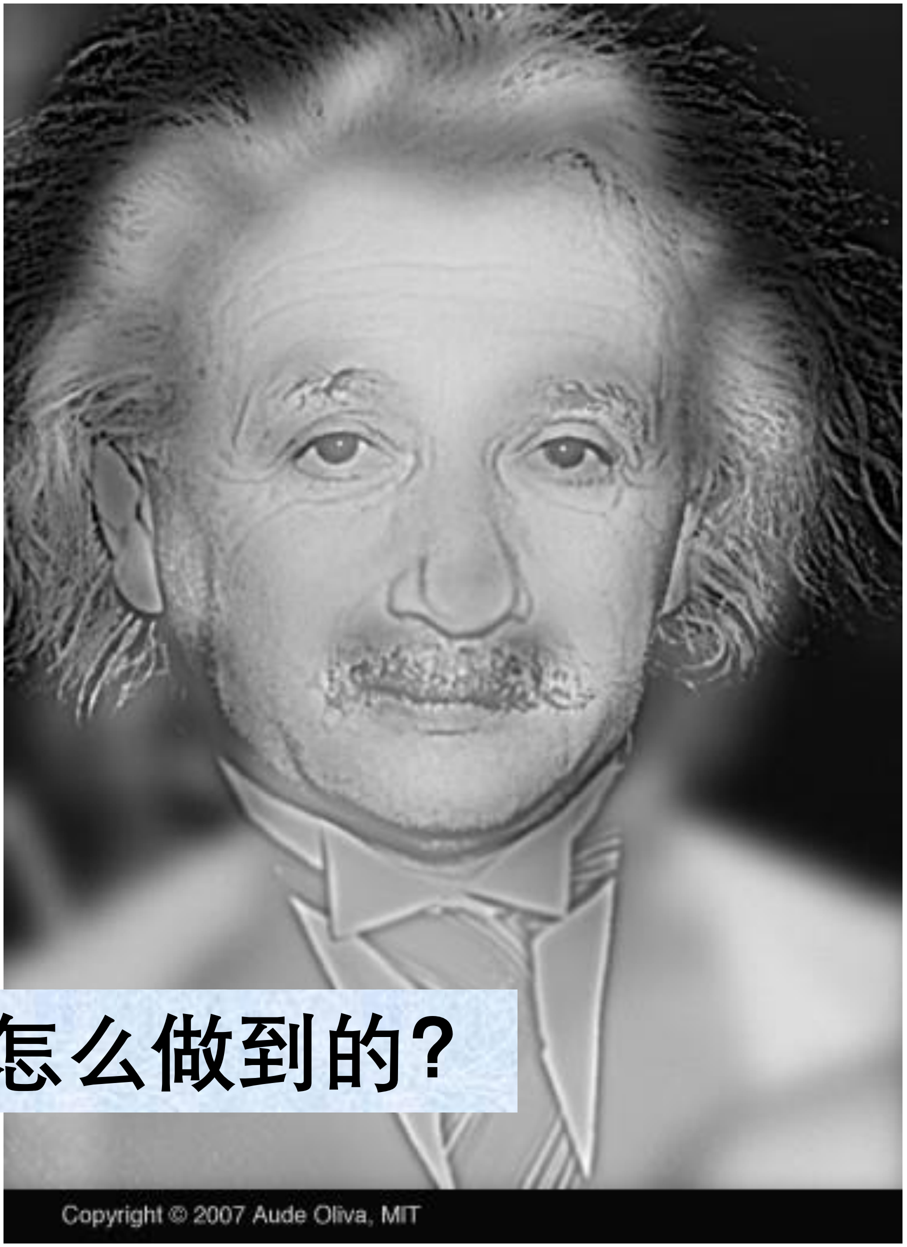


没有了点，轮子似乎缓慢地向后旋转！

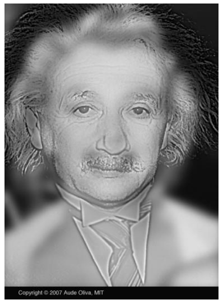


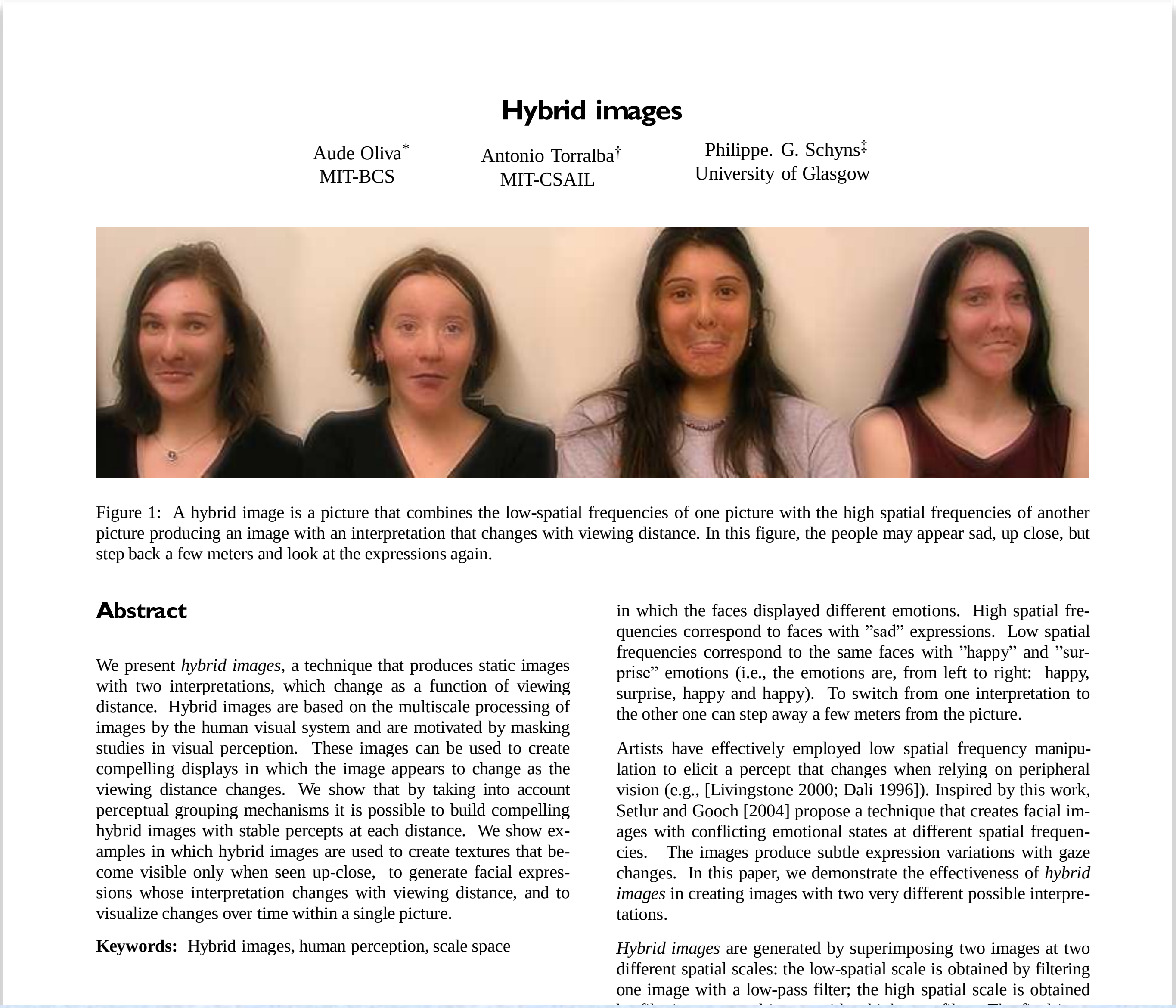
火车在朝着哪个方向运动?

07-17-2017 10:13:06


Copyright © 2007 Aude Oliva, MIT

这是怎么做到的?


Copyright © 2007 Aude Oliva, MIT



## Hybrid images

Aude Oliva[*] MIT-BCS
Antonio Torralba[†] MIT-CSAIL
Philippe. G. Schyns[‡] University of Glasgow

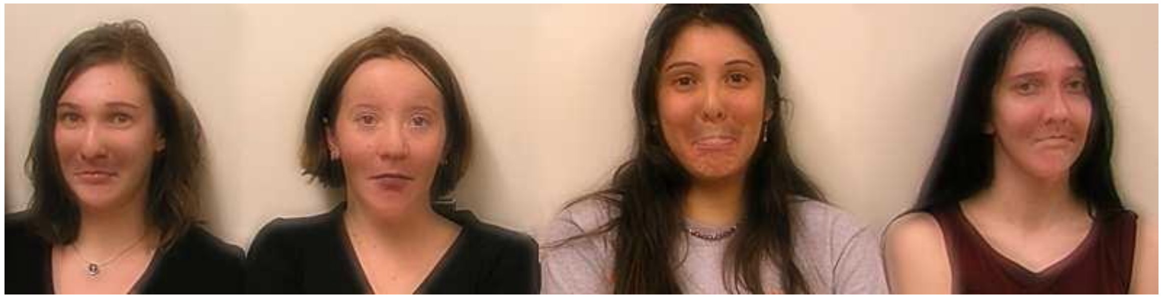


Figure 1: A hybrid image is a picture that combines the low-spatial frequencies of one picture with the high spatial frequencies of another picture producing an image with an interpretation that changes with viewing distance. In this figure, the people may appear sad, up close, but step back a few meters and look at the expressions again.

### Abstract

We present *hybrid images*, a technique that produces static images with two interpretations, which change as a function of viewing distance. Hybrid images are based on the multiscale processing of images by the human visual system and are motivated by masking studies in visual perception. These images can be used to create compelling displays in which the image appears to change as the viewing distance changes. We show that by taking into account perceptual grouping mechanisms it is possible to build compelling hybrid images with stable percepts at each distance. We show examples in which hybrid images are used to create textures that become visible only when seen up-close, to generate facial expressions whose interpretation changes with viewing distance, and to visualize changes over time within a single picture.

**Keywords:** Hybrid images, human perception, scale space

in which the faces displayed different emotions. High spatial frequencies correspond to faces with ”sad” expressions. Low spatial frequencies correspond to the same faces with ”happy” and ”surprise” emotions (i.e., the emotions are, from left to right: happy, surprise, happy and happy). To switch from one interpretation to the other one can step away a few meters from the picture.

Artists have effectively employed low spatial frequency manipulation to elicit a percept that changes when relying on peripheral vision (e.g., [Livingstone 2000; Dali 1996]). Inspired by this work, Setlur and Gooch [2004] propose a technique that creates facial images with conflicting emotional states at different spatial frequencies. The images produce subtle expression variations with gaze changes. In this paper, we demonstrate the effectiveness of *hybrid images* in creating images with two very different possible interpretations.

*Hybrid images* are generated by superimposing two images at two different spatial scales: the low-spatial scale is obtained by filtering one image with a low-pass filter; the high spatial scale is obtained

# SIGGRAPH, 2006

透镜起到低通滤波器的作用

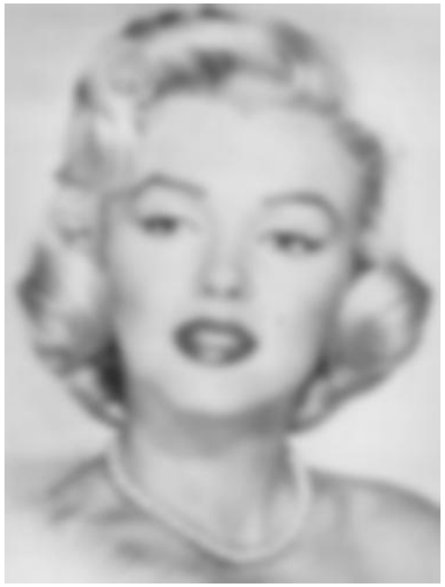
$* \quad g_1[x, y]$

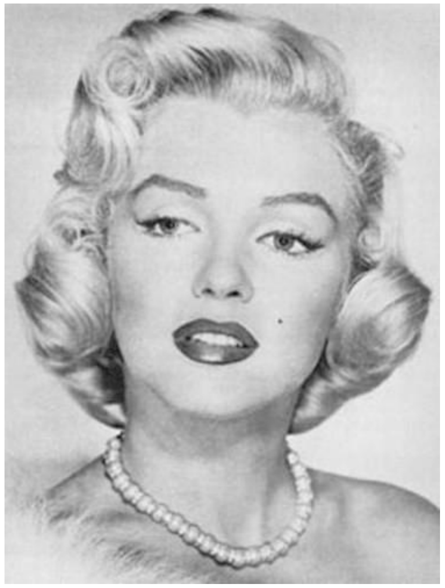
低通滤波器

$* \quad g_1[x, y]$

低通滤波器

$=$

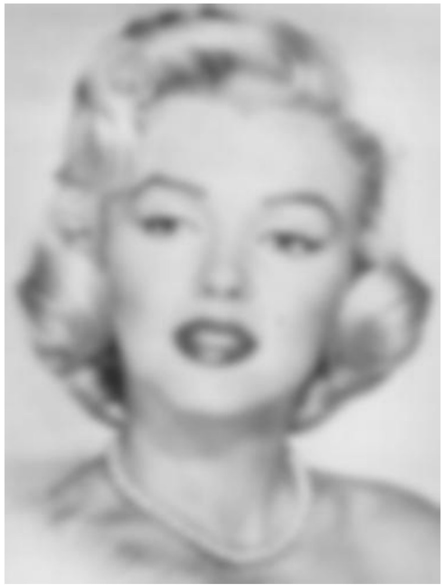


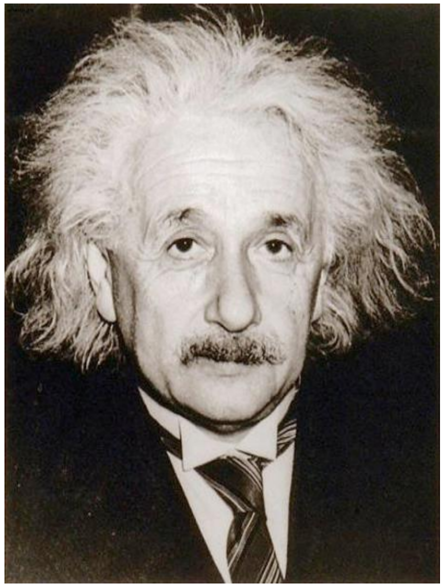


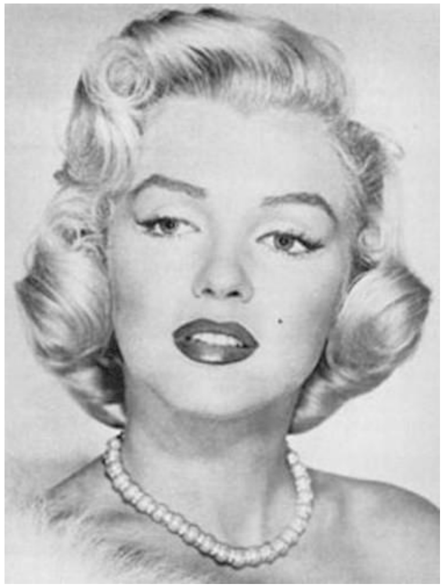
$* \quad g_1[x, y]$

低通滤波器

$=$



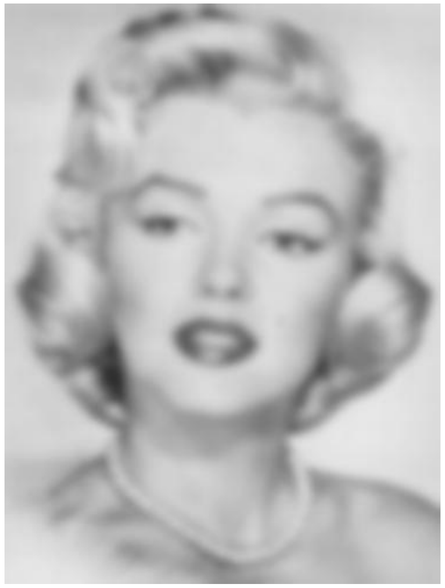


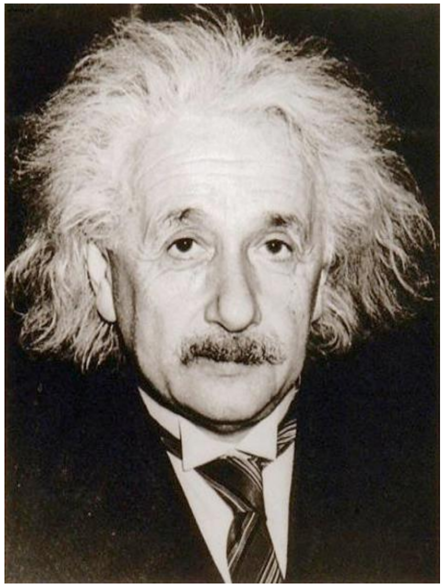


$* \quad g_1[x,y]$

低通滤波器

$=$





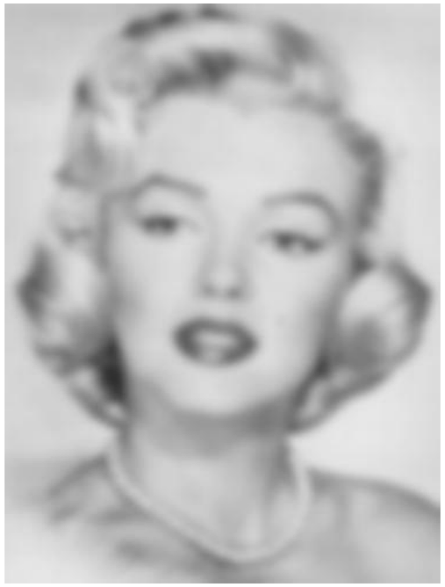
$* \, (\delta[x,y] - g_2[x,y])$

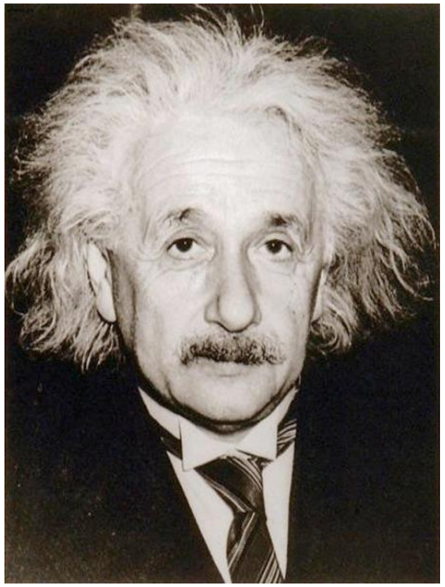
高通滤波器

$* \quad g_1[x,y]$

低通滤波器

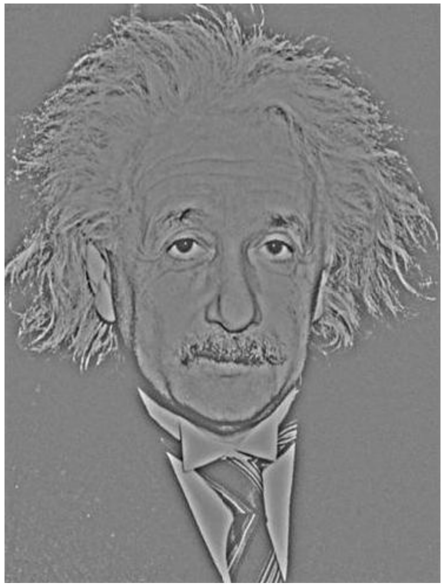
$=$





$* \left(\delta[x,y] - g_2[x,y]\right) =$

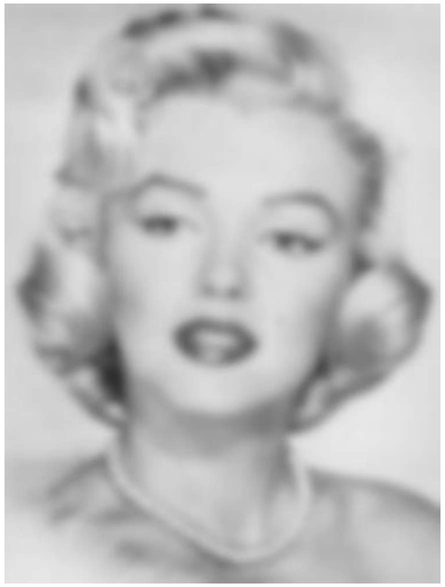
高通滤波器



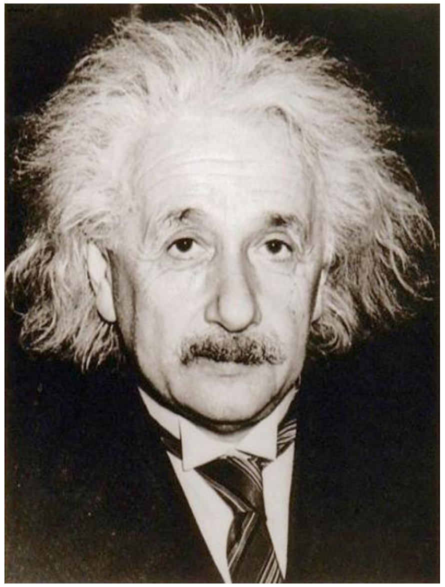
$* \quad g_1[x,y]$

低通滤波器

$=$



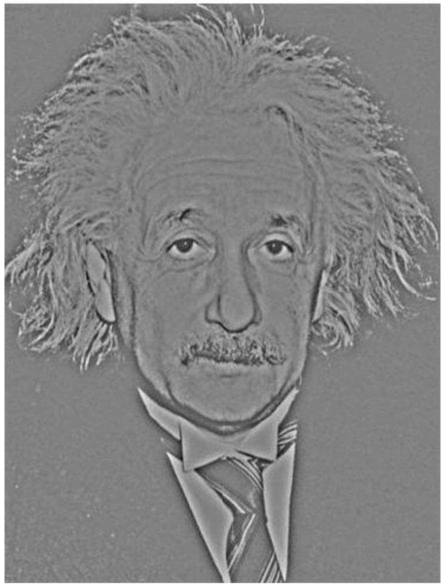
$+$



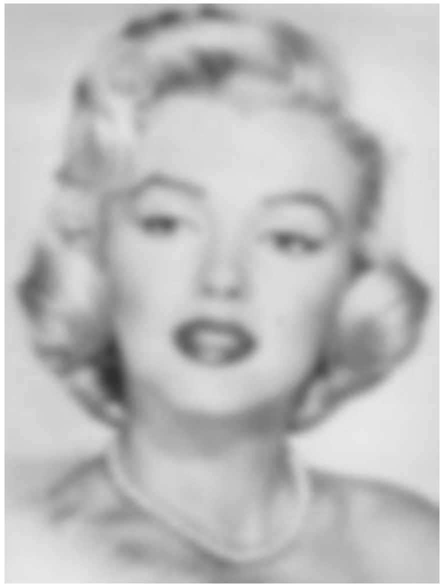
$* \, (\delta[x,y] - g_2[x,y])$

高通滤波器

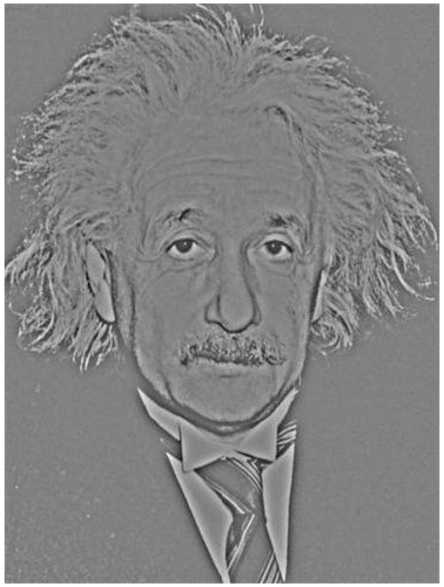
$=$



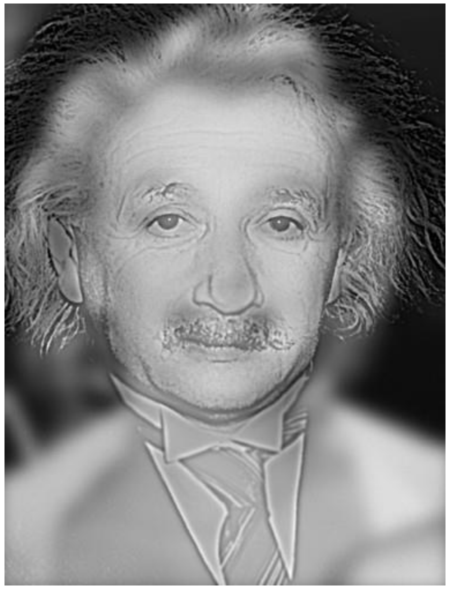
=



+



=

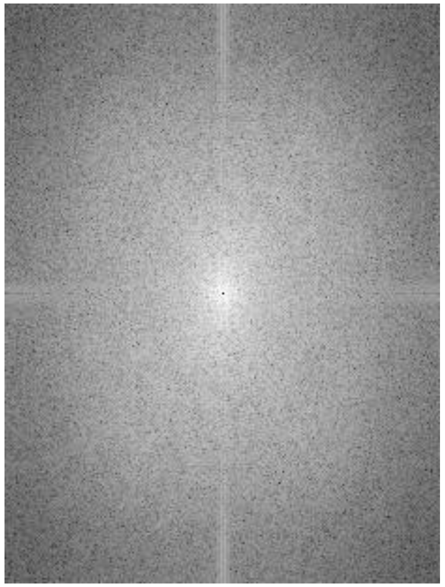


$[x, y]$ =

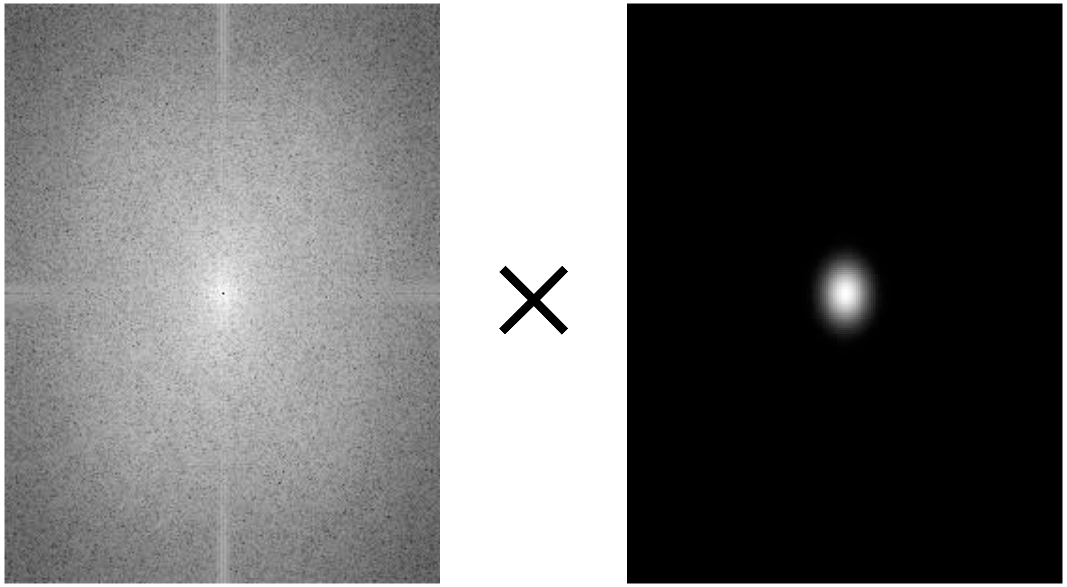
器

## 频域

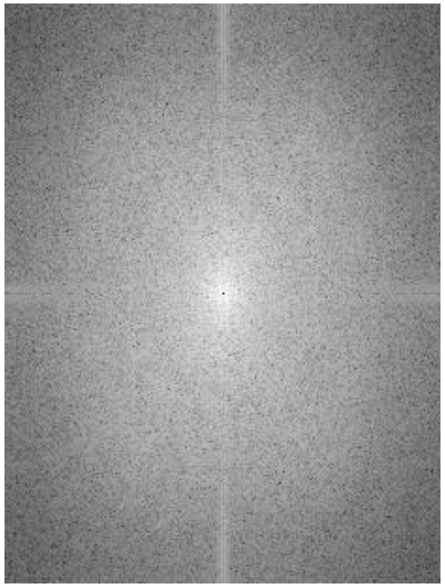
## 频域



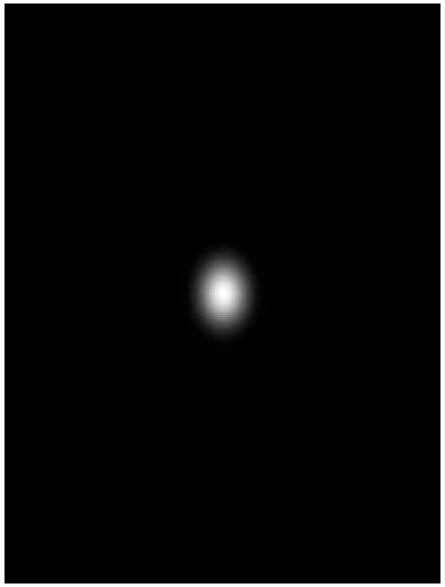
## 频域



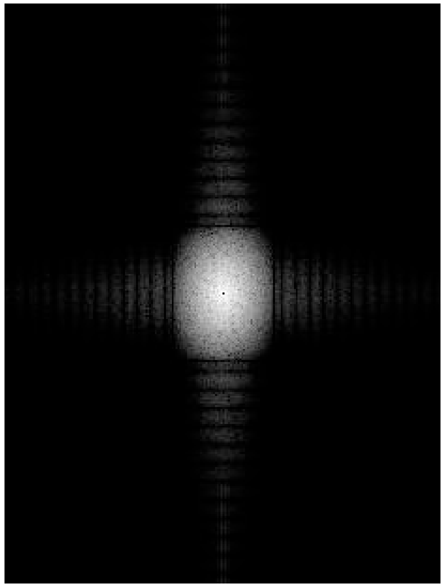
## 频域

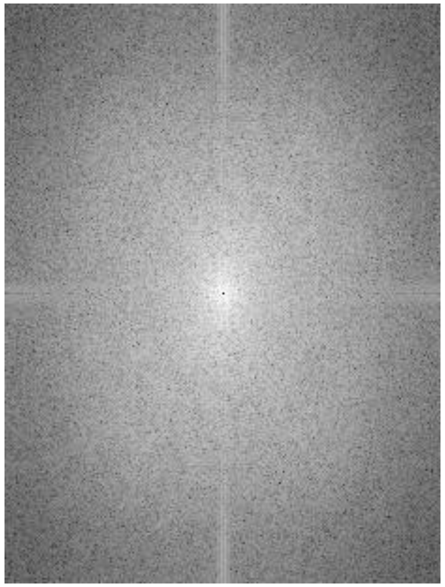


$\times$

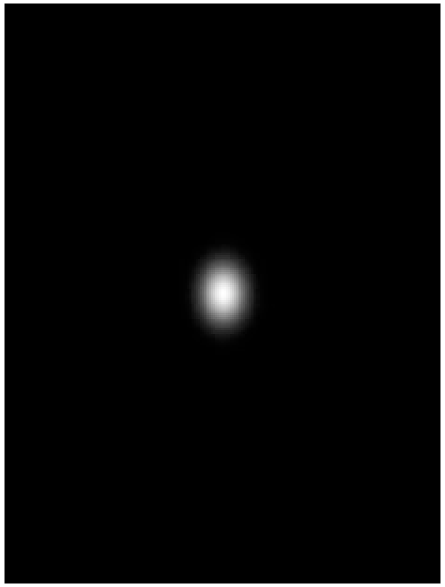


$=$

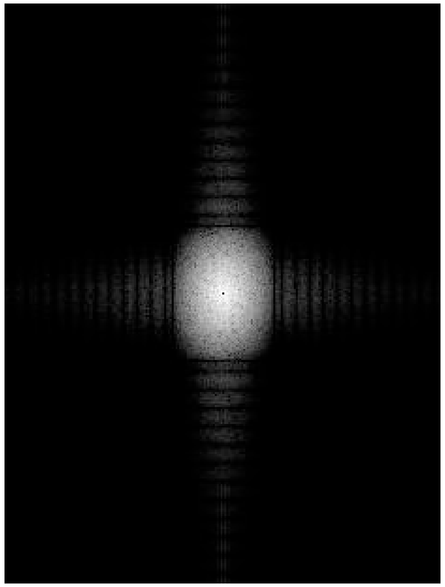


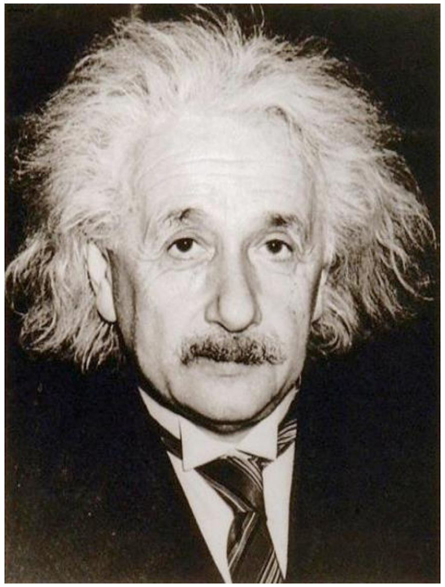


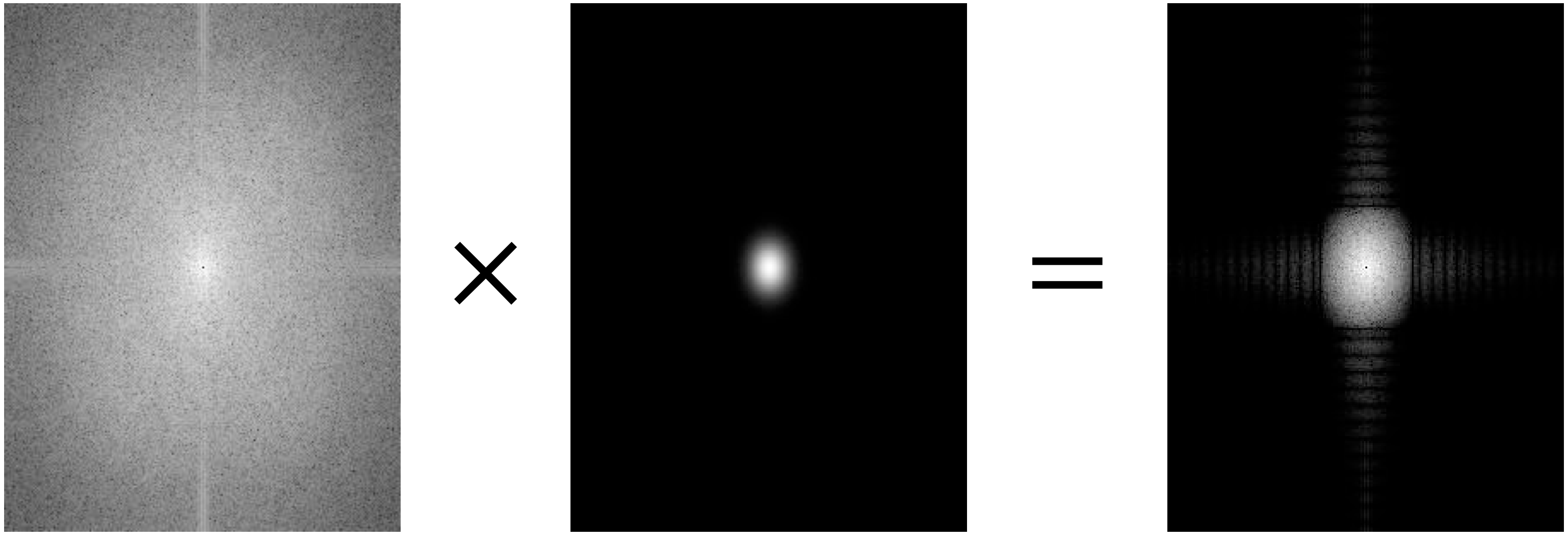
$\times$

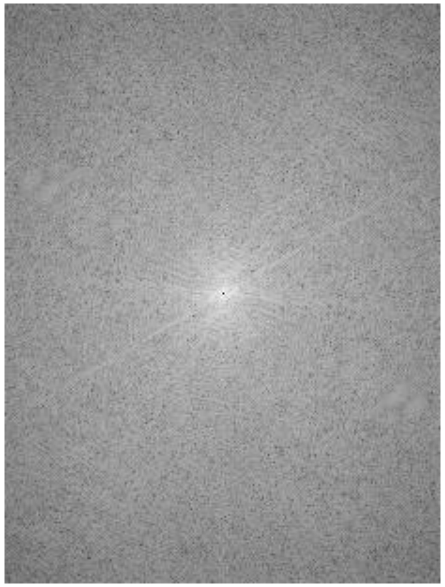


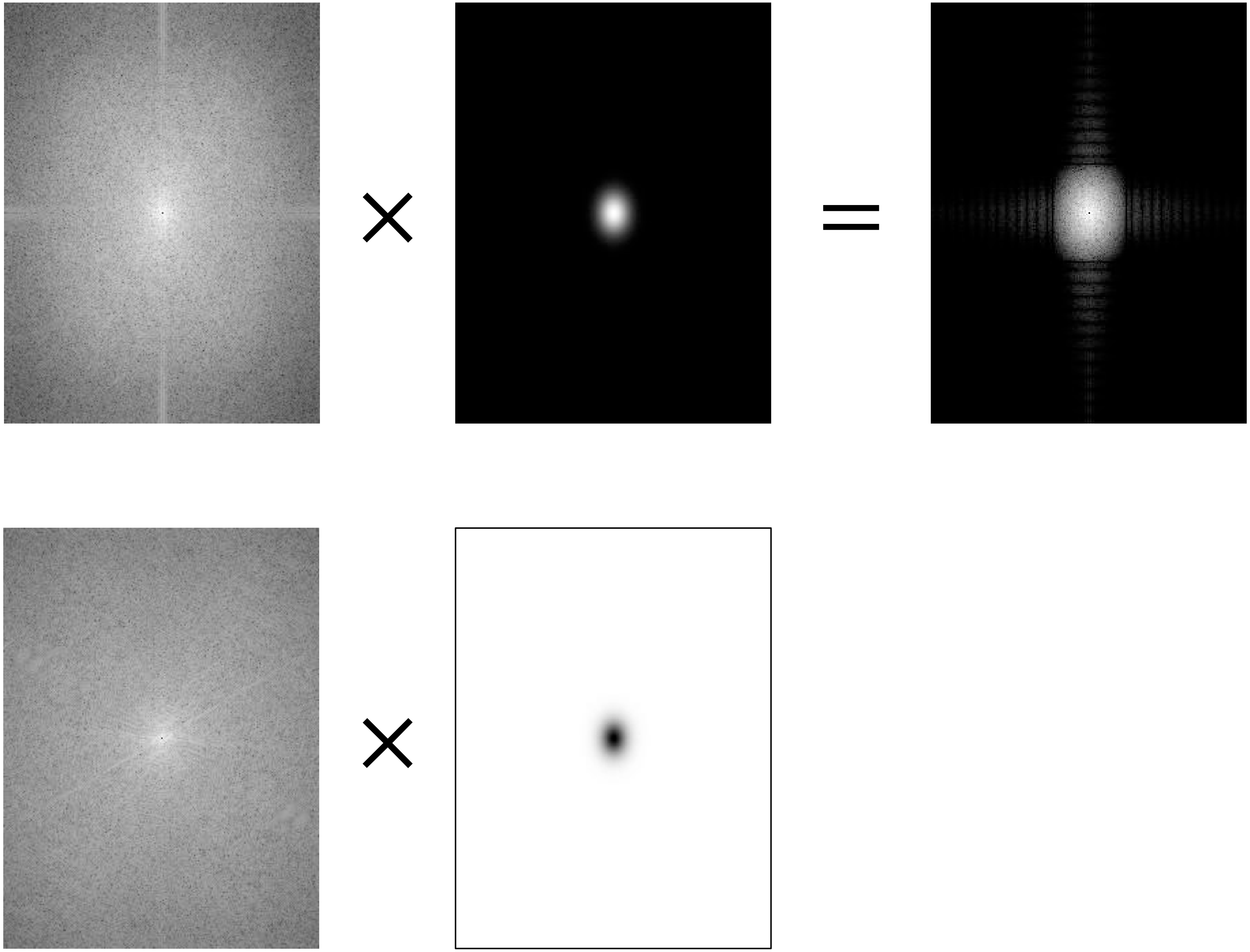
$=$

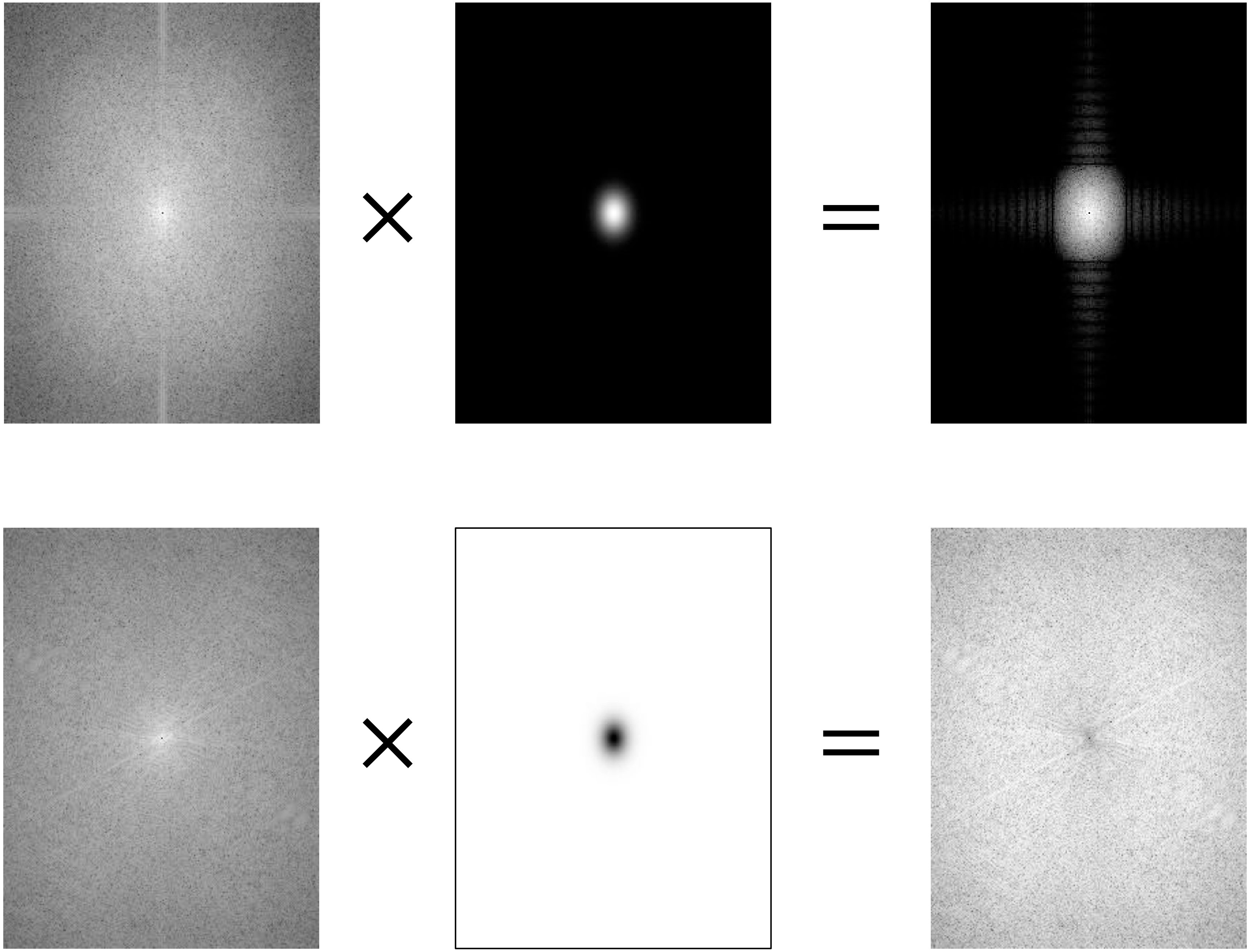


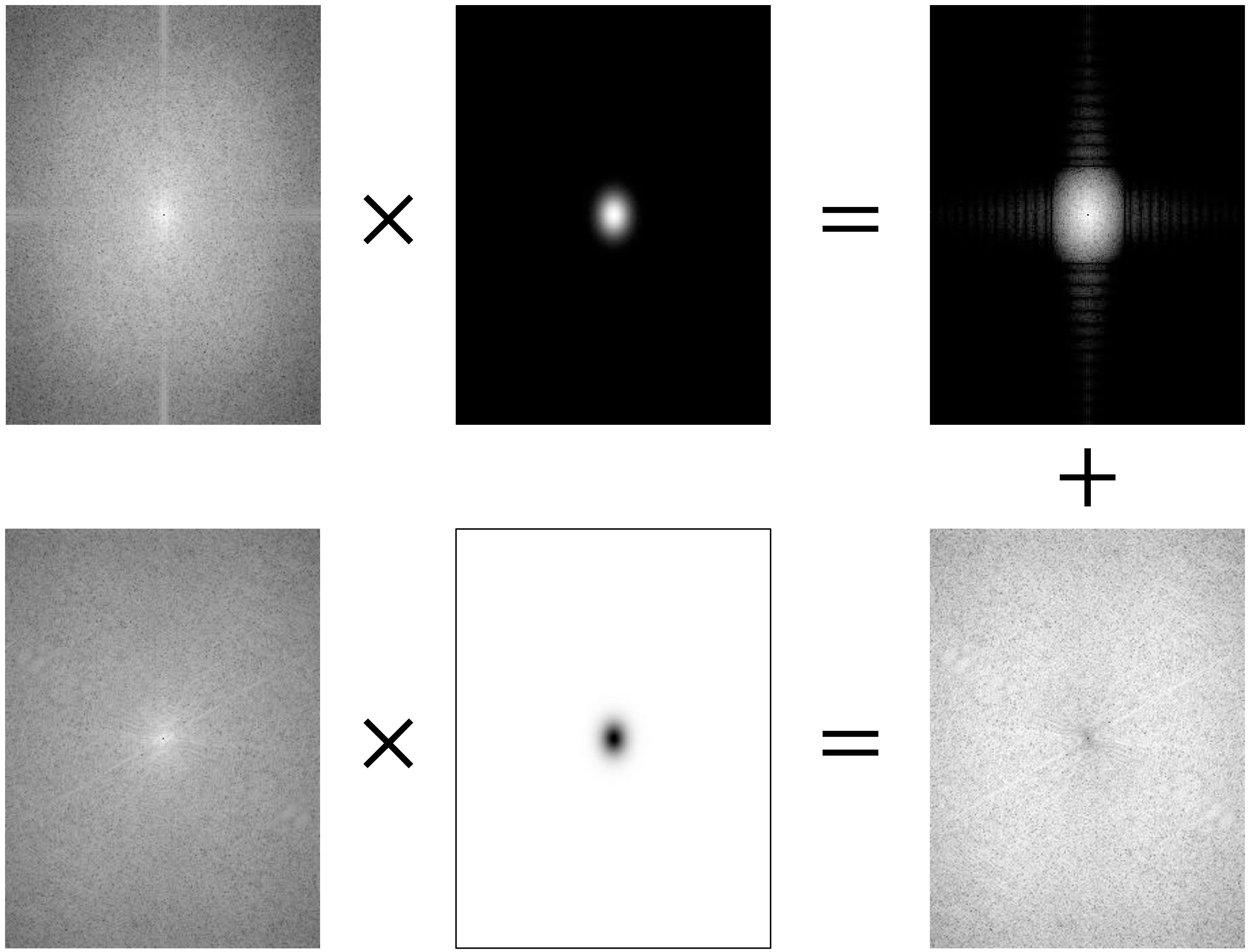




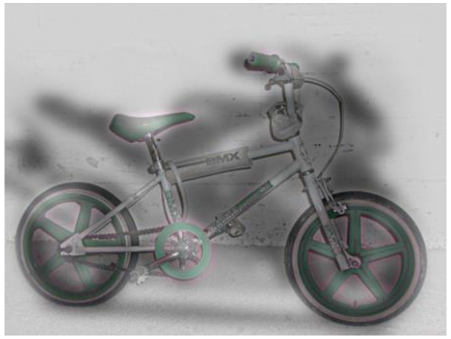




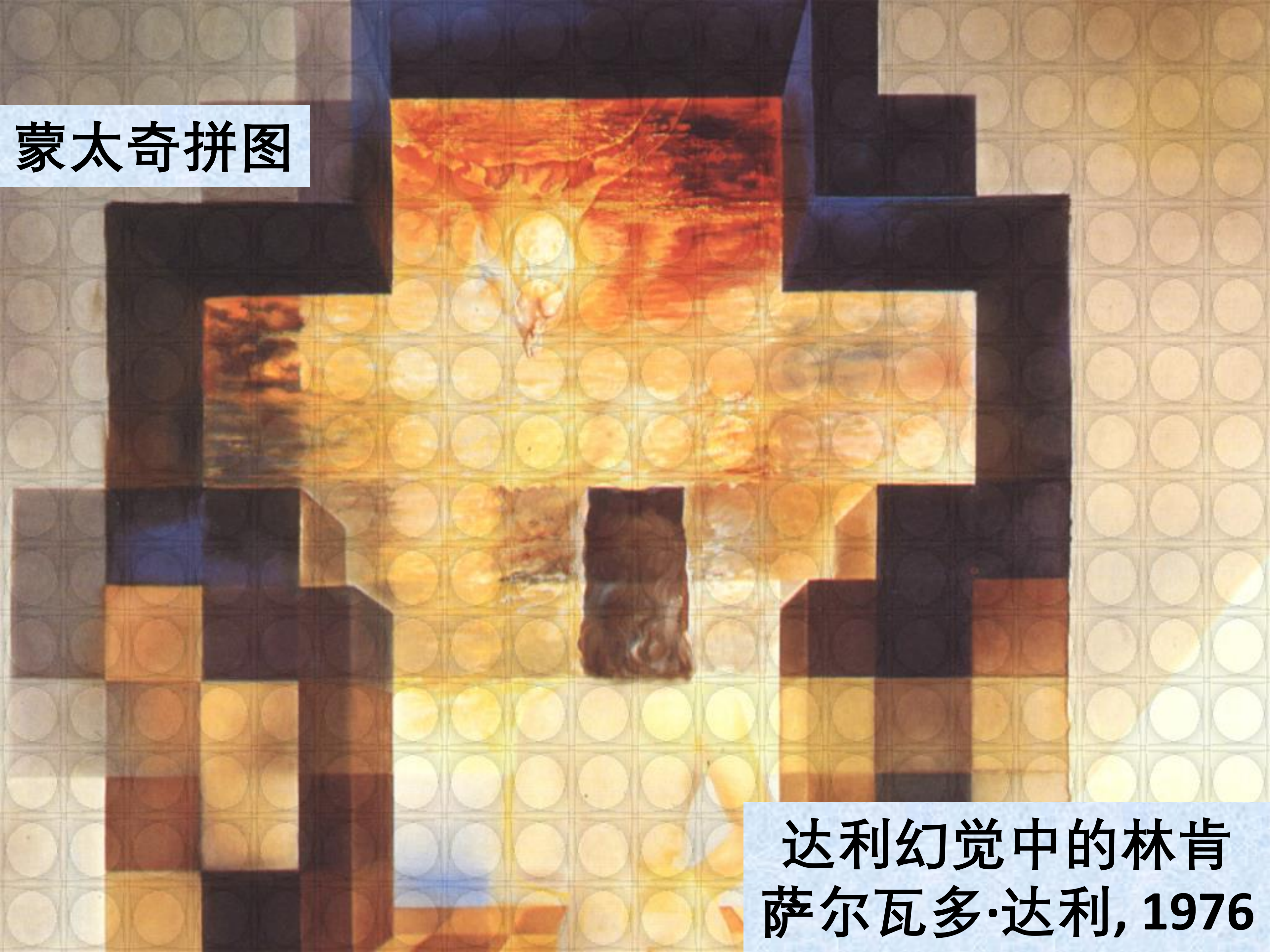




BMX
BMX
BMX

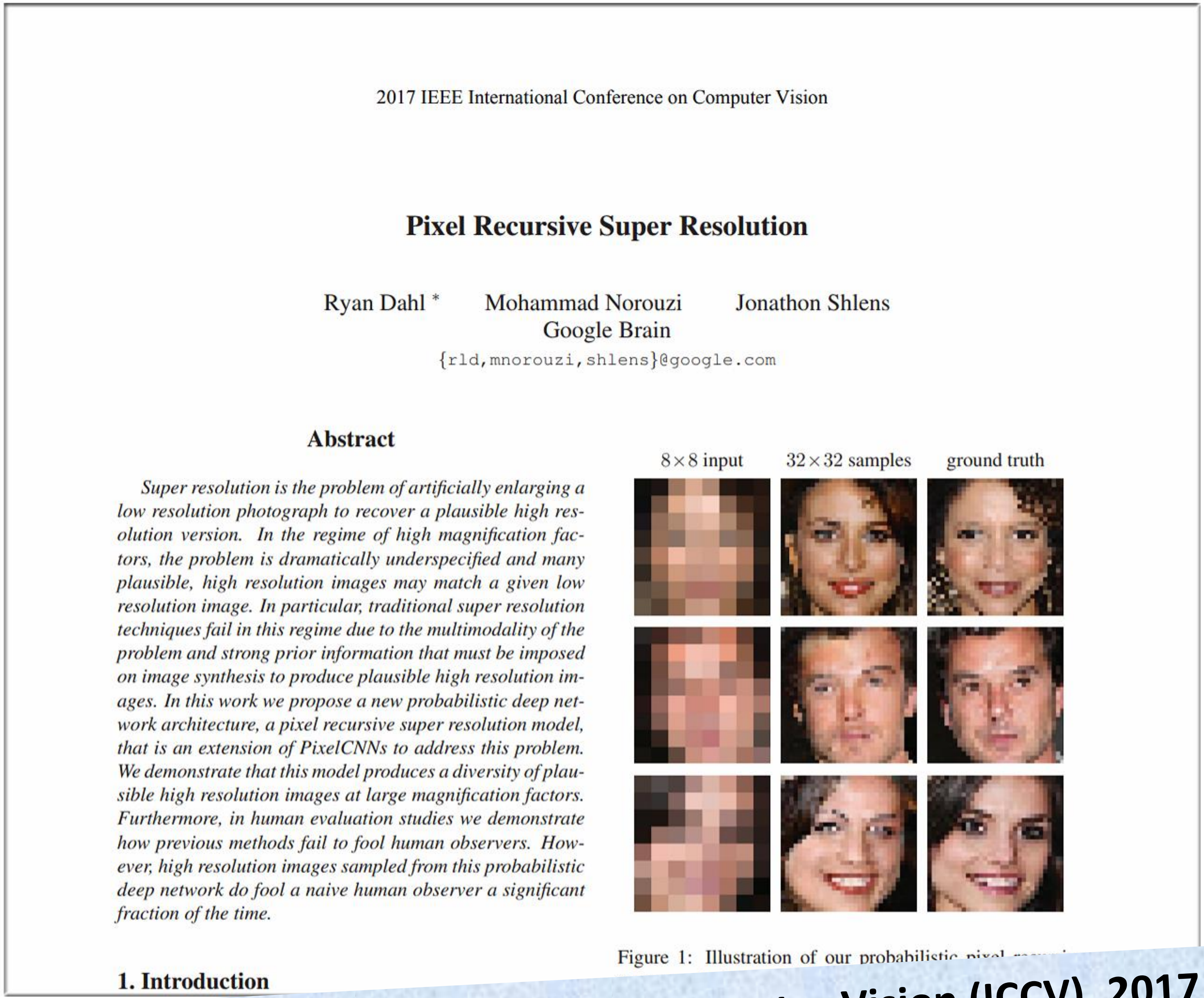

BMX

## 蒙太奇拼图



达利幻觉中的林肯
萨尔瓦多·达利, 1976

FEDERAL RESERVE NOTE
THE UNITED STATES OF AMERICA
IB625 [redacted] 38 A
SERIES 2006
Secretary of the Treasury.
FW A 8
FIVE DOLLARS
LINCOLN
5



2017 IEEE International Conference on Computer Vision

# Pixel Recursive Super Resolution

Ryan Dahl [*] Mohammad Norouzi Jonathon Shlens
Google Brain
{rld,mnorouzi,shlens}@google.com

## Abstract

*Super resolution is the problem of artificially enlarging a low resolution photograph to recover a plausible high resolution version. In the regime of high magnification factors, the problem is dramatically underspecified and many plausible, high resolution images may match a given low resolution image. In particular, traditional super resolution techniques fail in this regime due to the multimodality of the problem and strong prior information that must be imposed on image synthesis to produce plausible high resolution images. In this work we propose a new probabilistic deep network architecture, a pixel recursive super resolution model, that is an extension of PixelCNNs to address this problem. We demonstrate that this model produces a diversity of plausible high resolution images at large magnification factors. Furthermore, in human evaluation studies we demonstrate how previous methods fail to fool human observers. However, high resolution images sampled from this probabilistic deep network do fool a naive human observer a significant fraction of the time.*

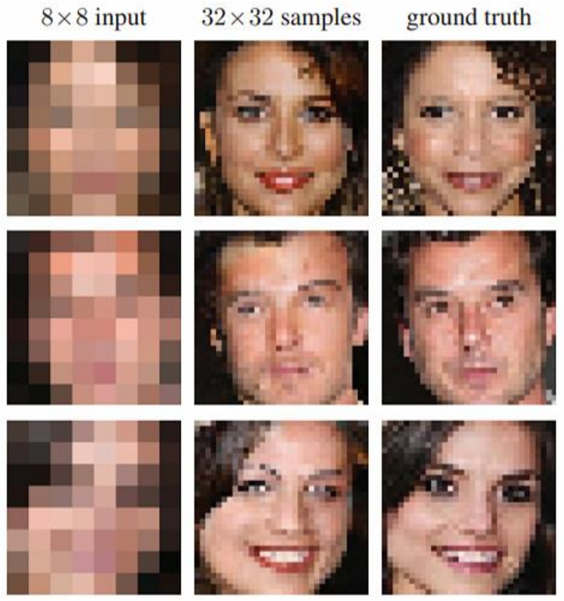
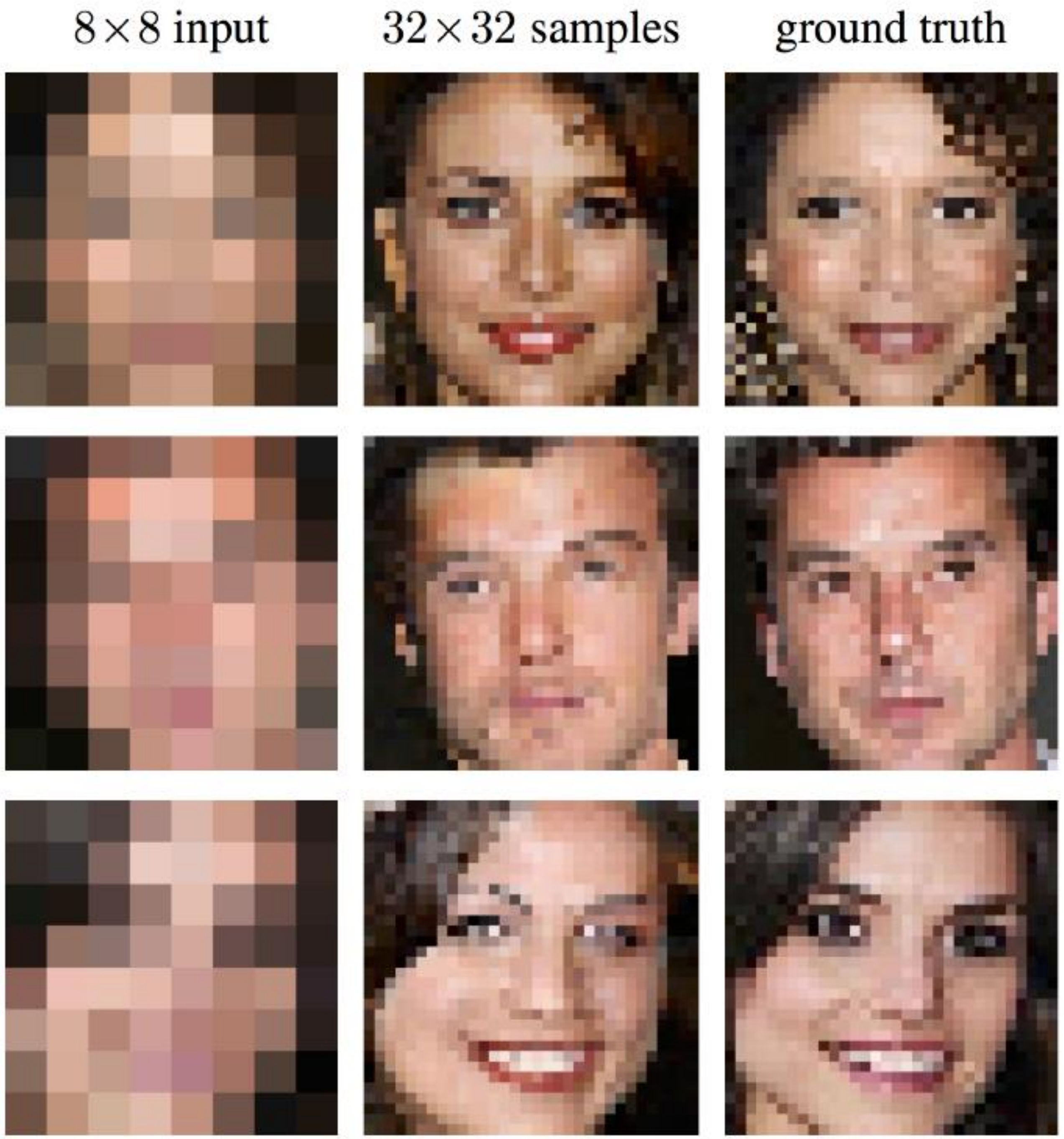


Figure 1: Illustration of our probabilistic pixel

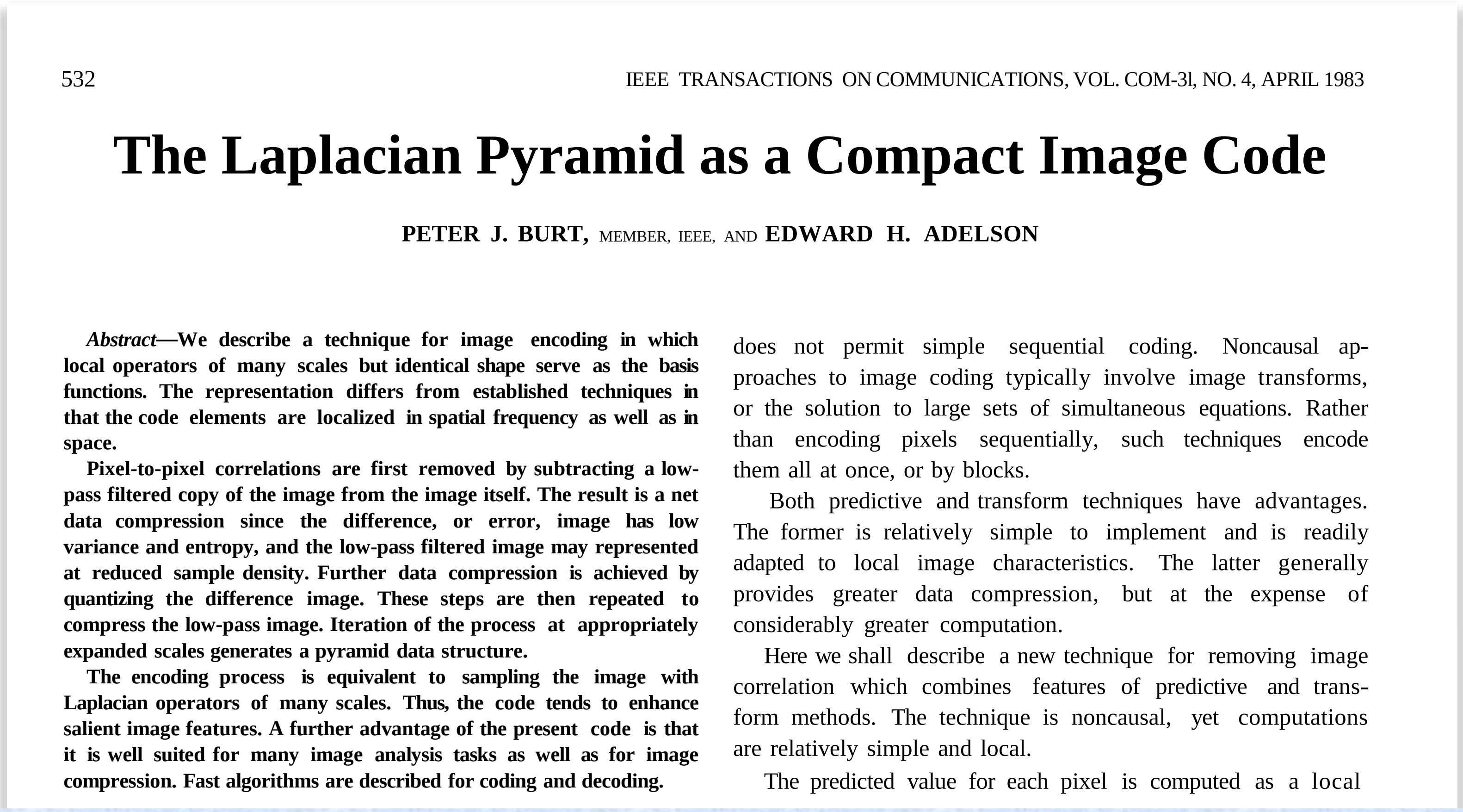
## 1. Introduction

**IEEE International Conference on Computer Vision (ICCV), 2017**

*odel that*
*hancing*
*respond*
*odeling*
*ent con-*
*details–*
*o repre-*
*rly mod-*
*solution*
*out. We*
*ng prior*
*r with a*
*evalua-*
*del look*
*aseline.*



## 图像金字塔



# The Laplacian Pyramid as a Compact Image Code

PETER J. BURT, MEMBER, IEEE, AND EDWARD H. ADELSON

***Abstract*—We describe a technique for image encoding in which local operators of many scales but identical shape serve as the basis functions. The representation differs from established techniques in that the code elements are localized in spatial frequency as well as in space.**

**Pixel-to-pixel correlations are first removed by subtracting a low-pass filtered copy of the image from the image itself. The result is a net data compression since the difference, or error, image has low variance and entropy, and the low-pass filtered image may represented at reduced sample density. Further data compression is achieved by quantizing the difference image. These steps are then repeated to compress the low-pass image. Iteration of the process at appropriately expanded scales generates a pyramid data structure.**

**The encoding process is equivalent to sampling the image with Laplacian operators of many scales. Thus, the code tends to enhance salient image features. A further advantage of the present code is that it is well suited for many image analysis tasks as well as for image compression. Fast algorithms are described for coding and decoding.**

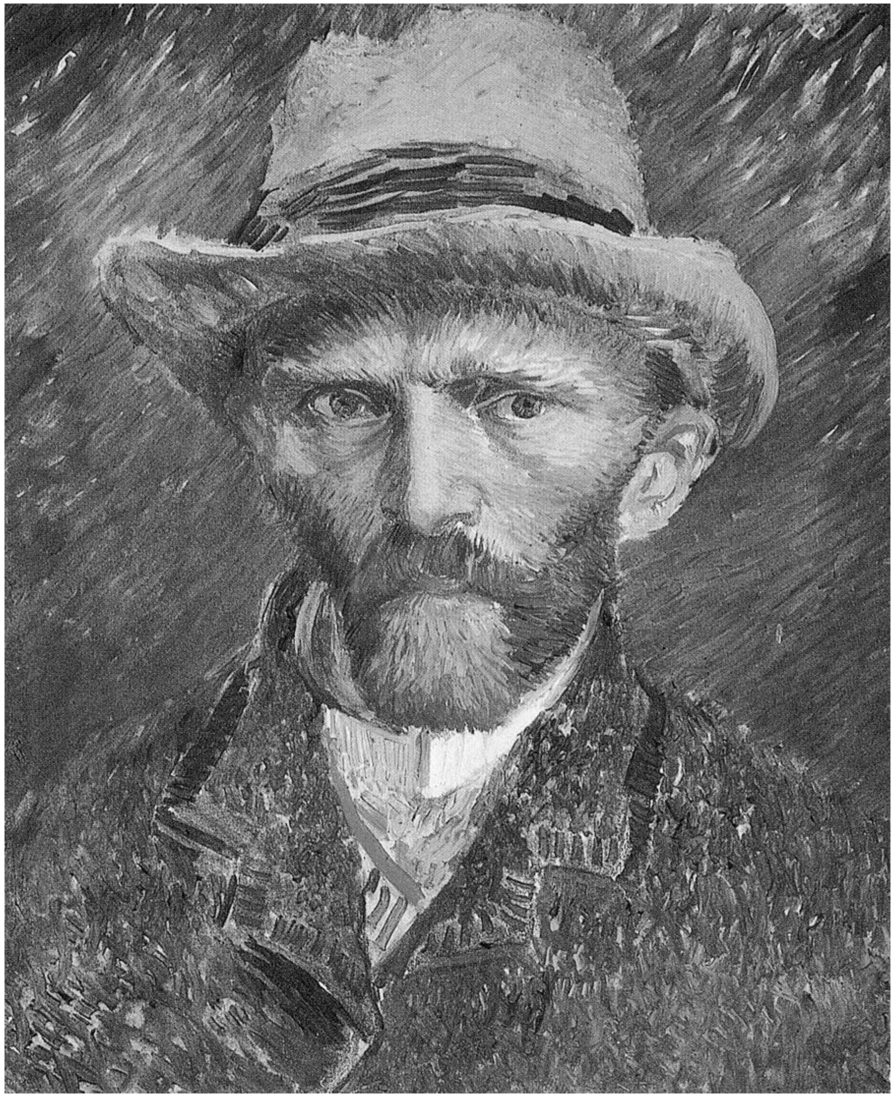
does not permit simple sequential coding. Noncausal approaches to image coding typically involve image transforms, or the solution to large sets of simultaneous equations. Rather than encoding pixels sequentially, such techniques encode them all at once, or by blocks.

Both predictive and transform techniques have advantages. The former is relatively simple to implement and is readily adapted to local image characteristics. The latter generally provides greater data compression, but at the expense of considerably greater computation.

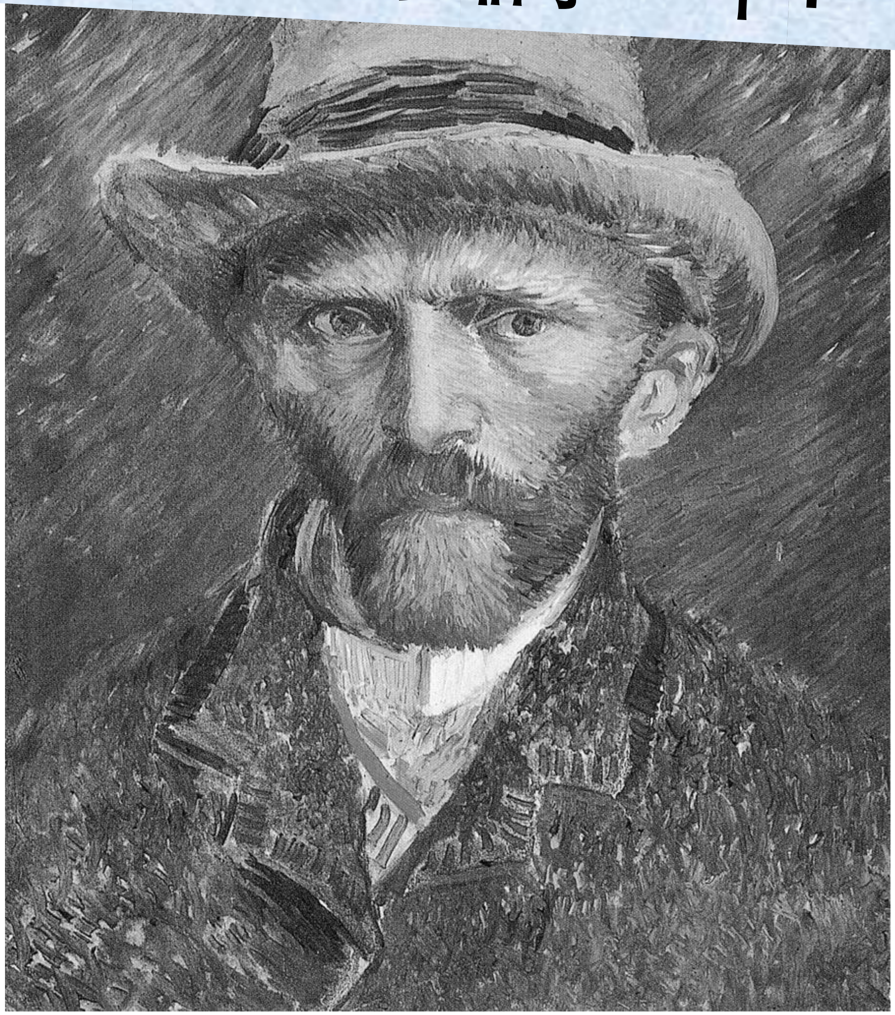
Here we shall describe a new technique for removing image correlation which combines features of predictive and transform methods. The technique is noncausal, yet computations are relatively simple and local.

The predicted value for each pixel is computed as a local

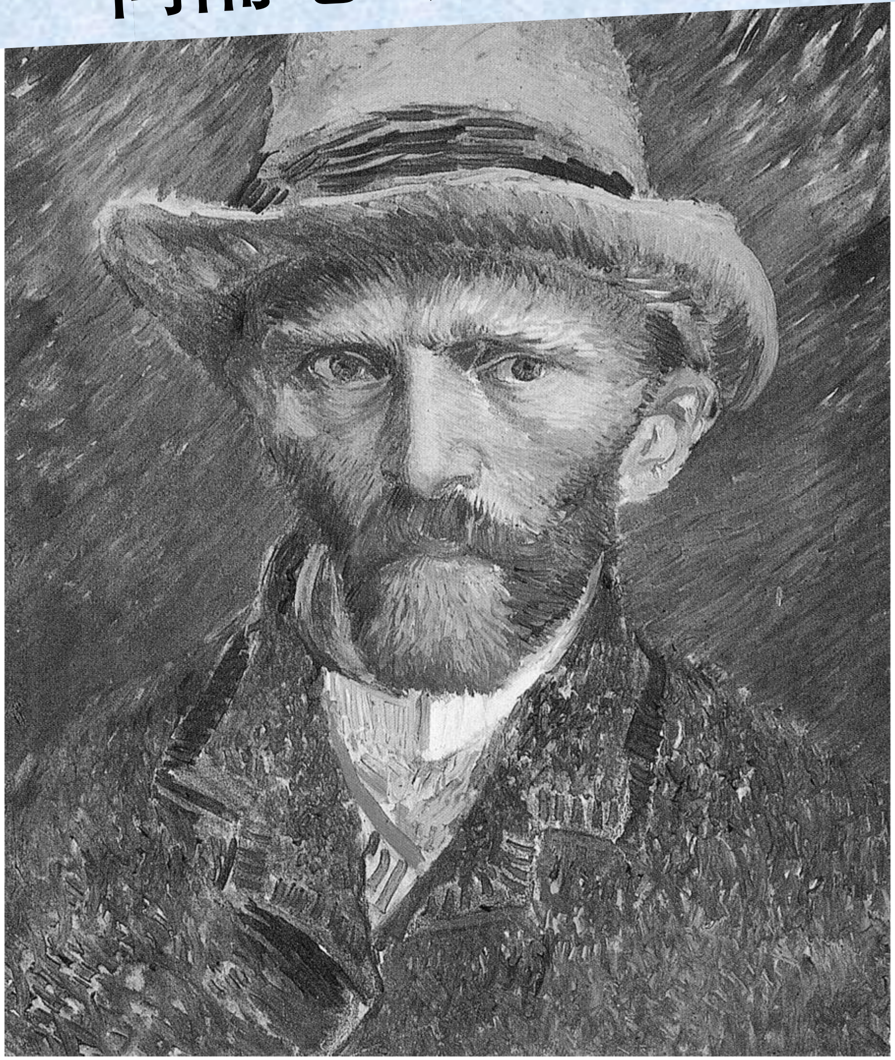
## IEEE Transactions on Communications, 1983



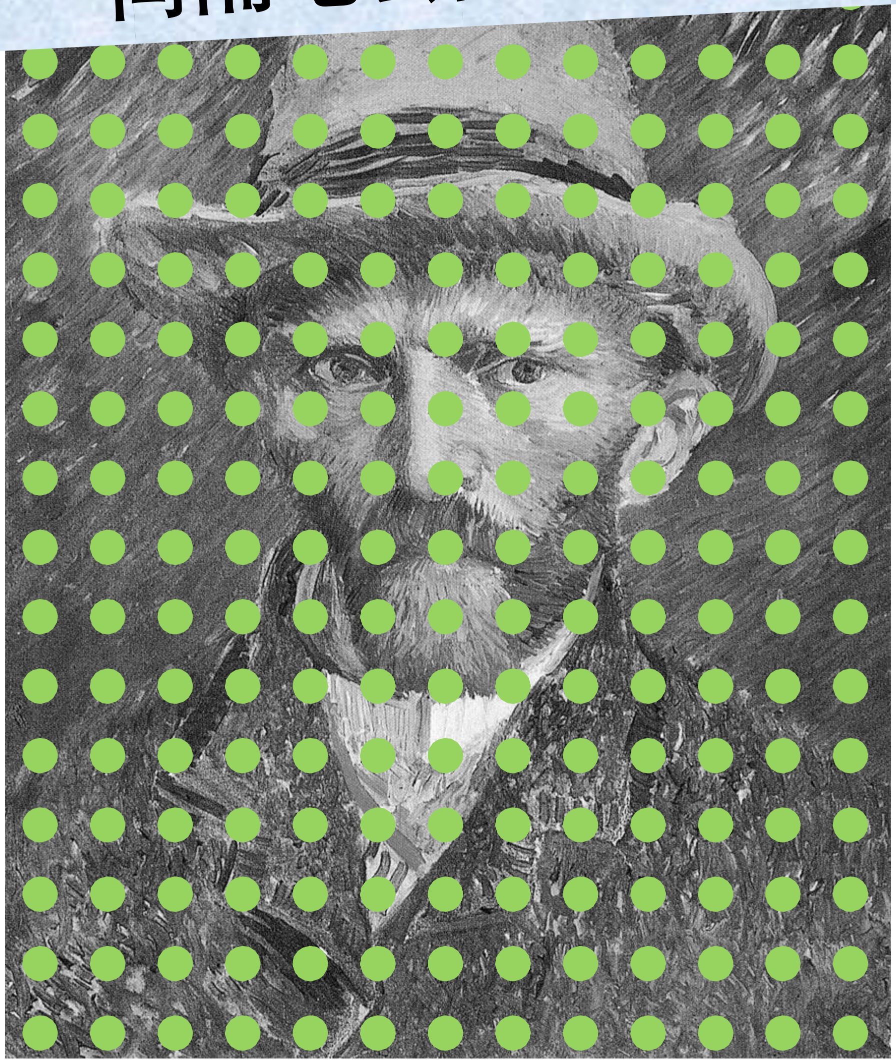
## 如何把尺寸缩小一半？

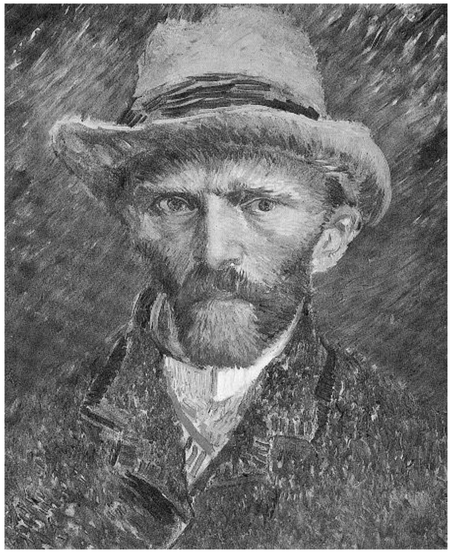


## 间隔地丢掉行和列

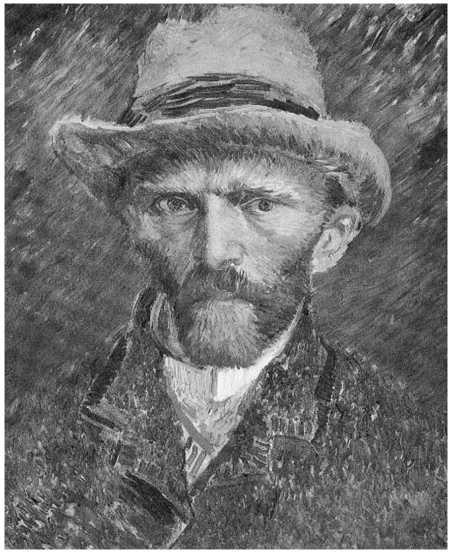


## 间隔地丢掉行和列

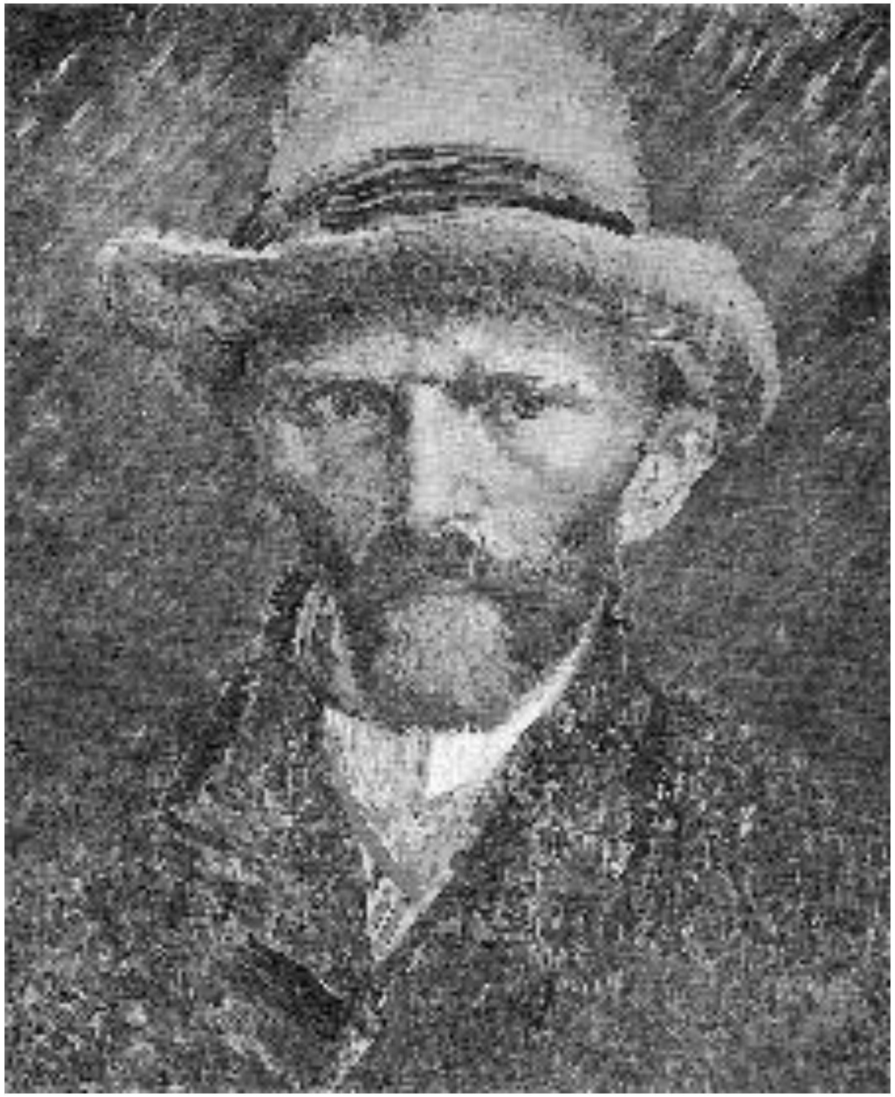




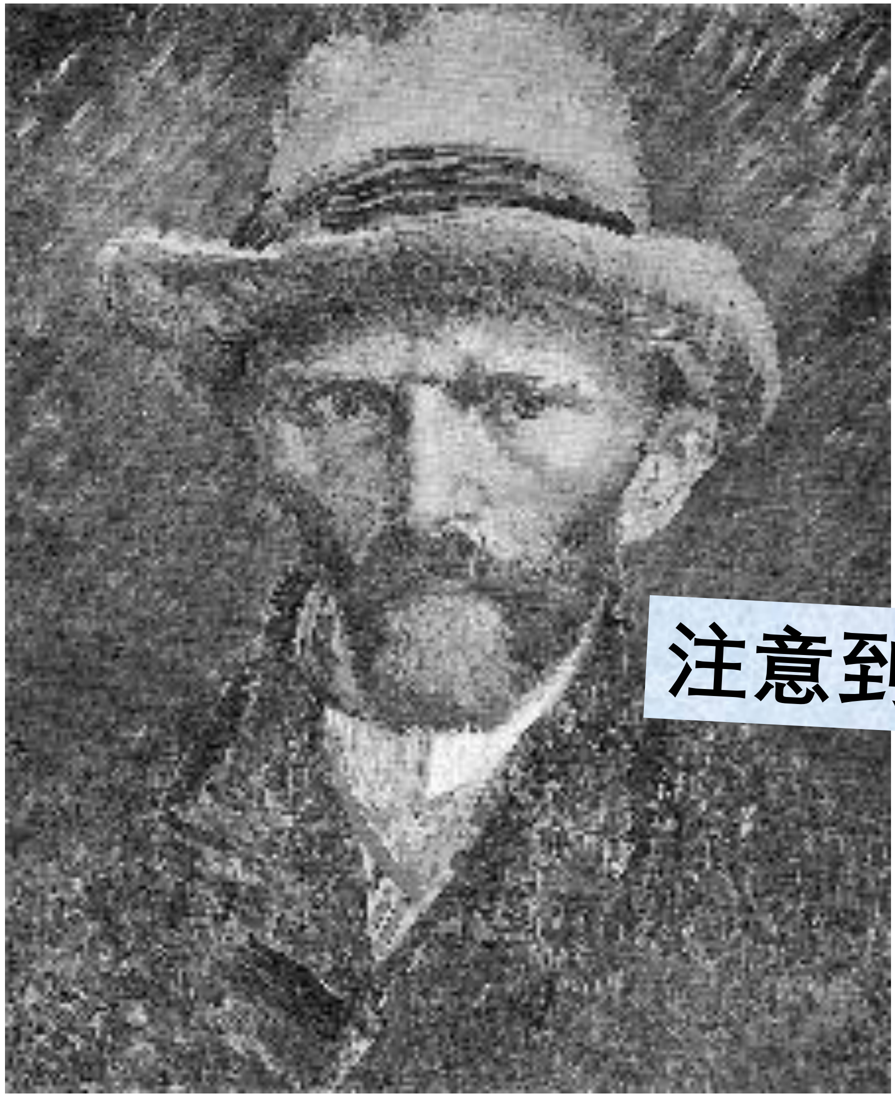
1/2尺寸


1/2尺寸

1/4尺寸

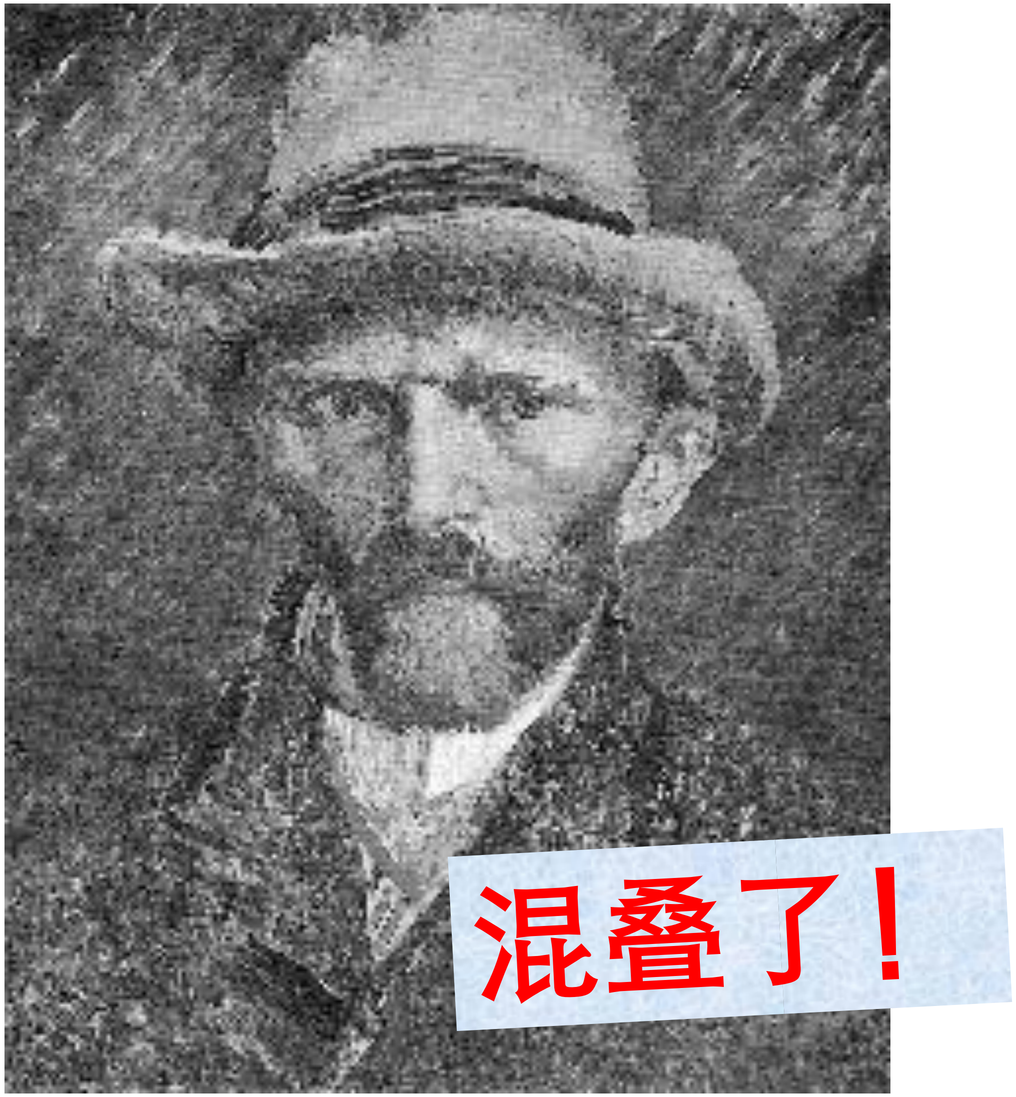


1/4尺寸 (放大4倍)



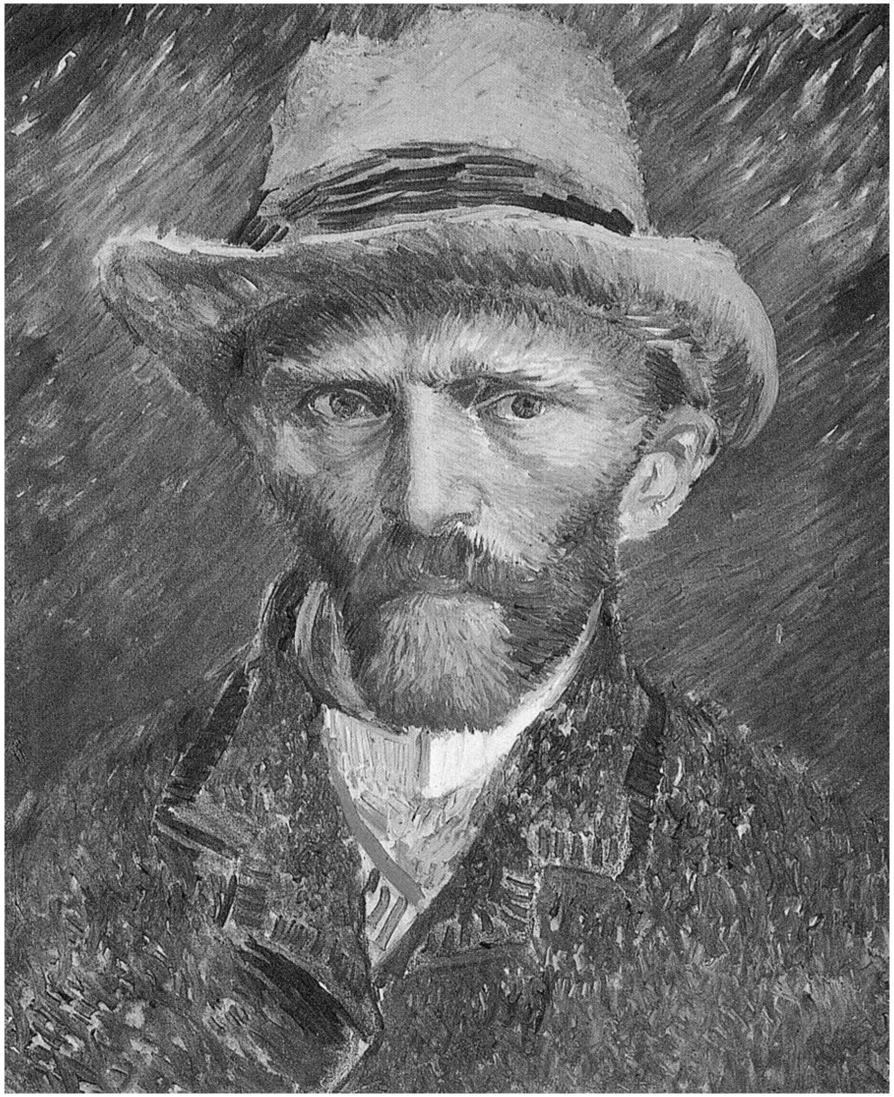
注意到什么了吗？

1/4尺寸 (放大4倍)

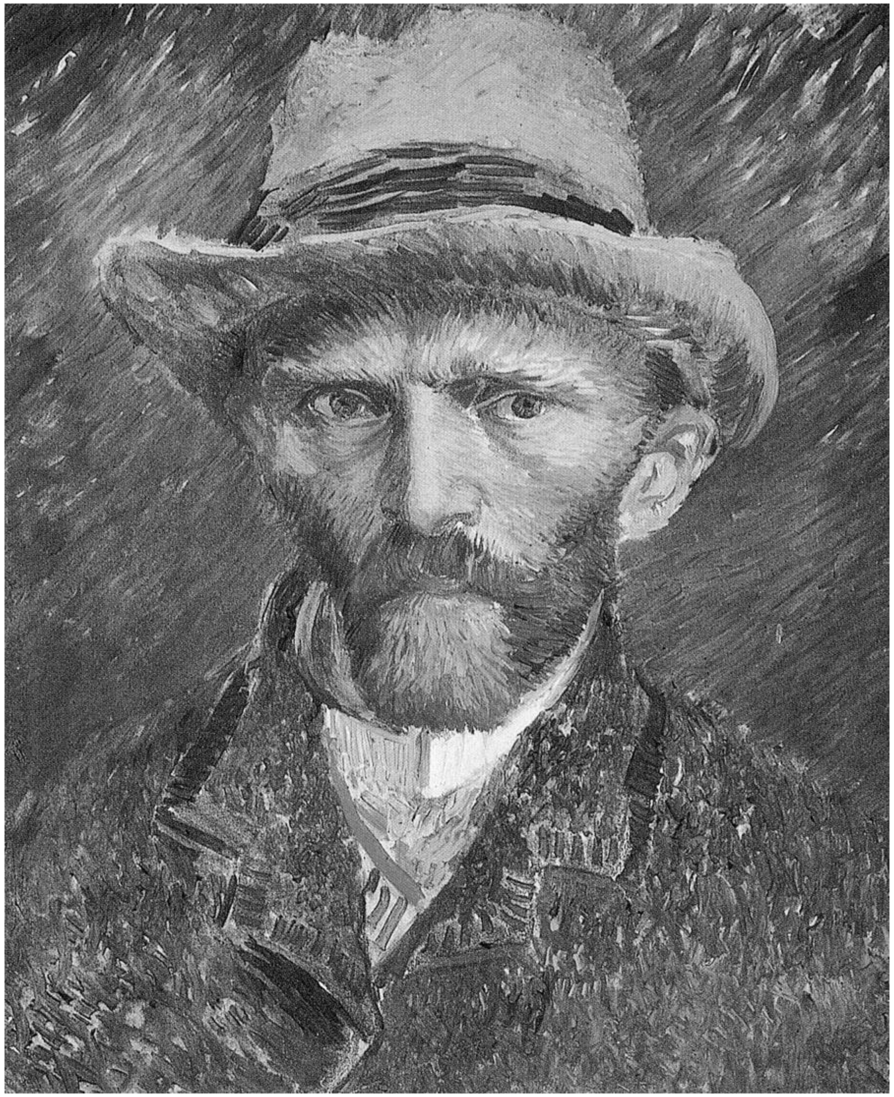


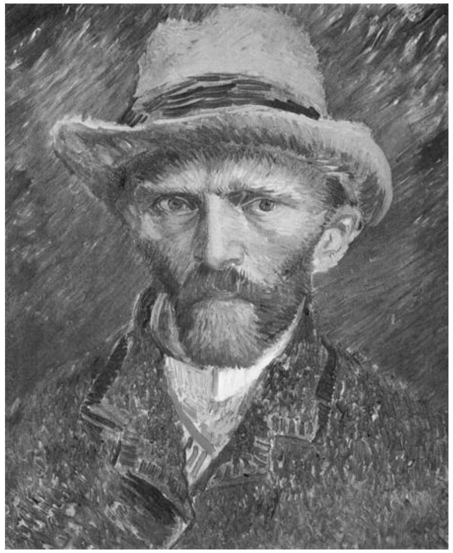
1/4尺寸 (放大4倍)

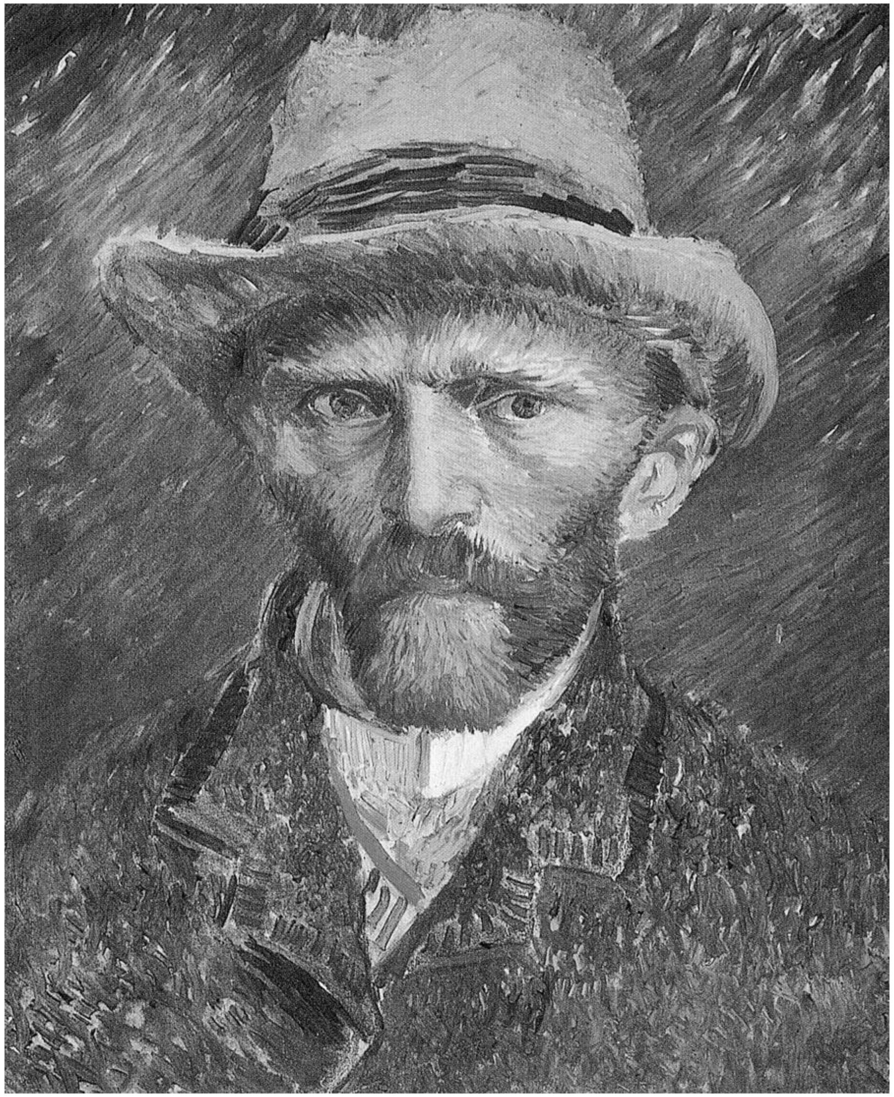
## 先平滑再进行下采样

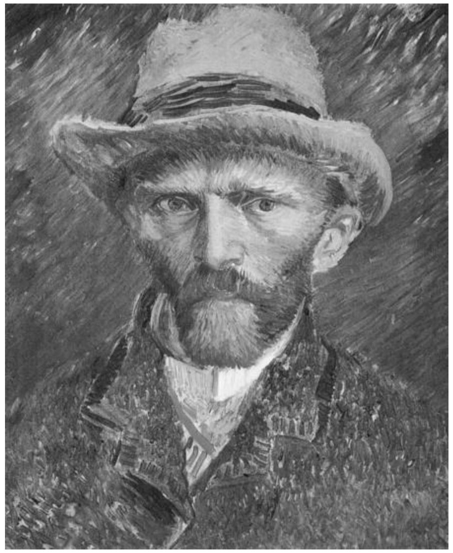


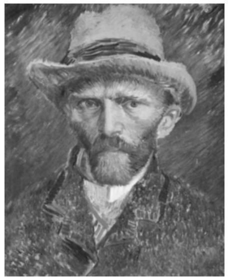
原图

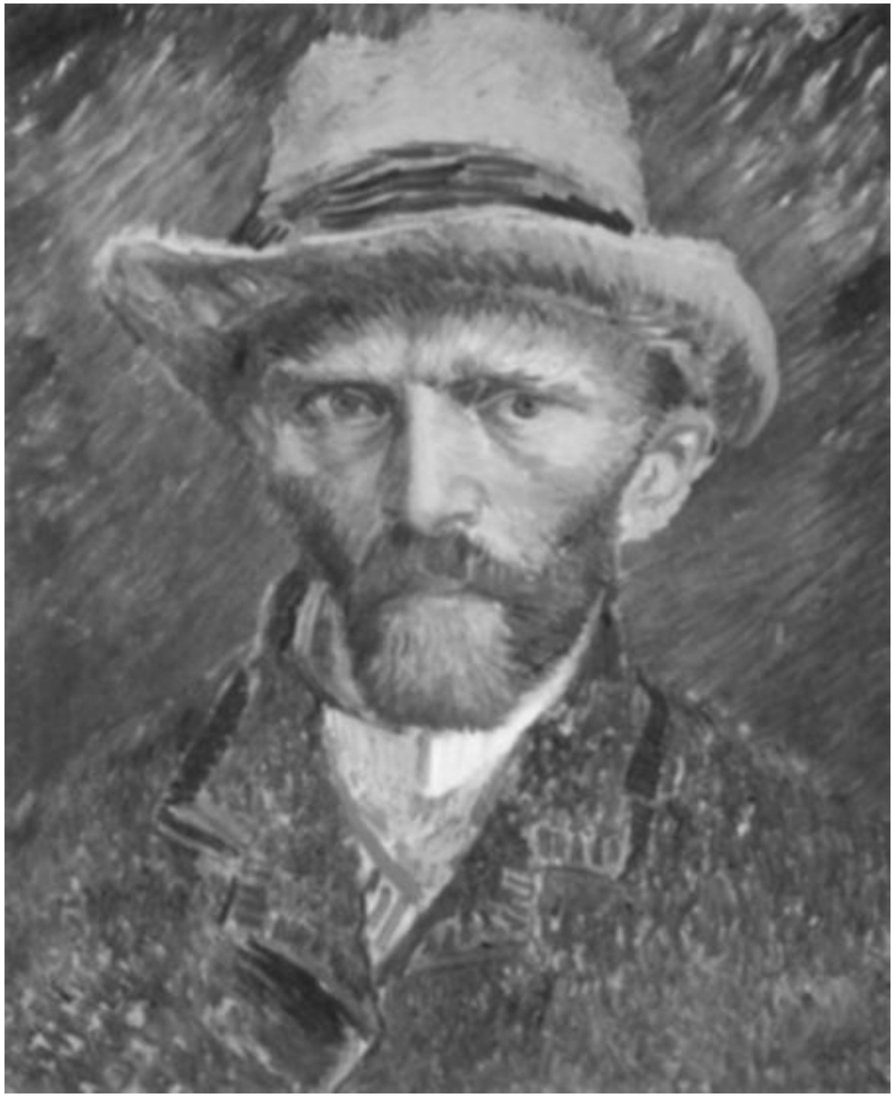

原图


1/2尺寸


原图


1/2尺寸


1/4尺寸



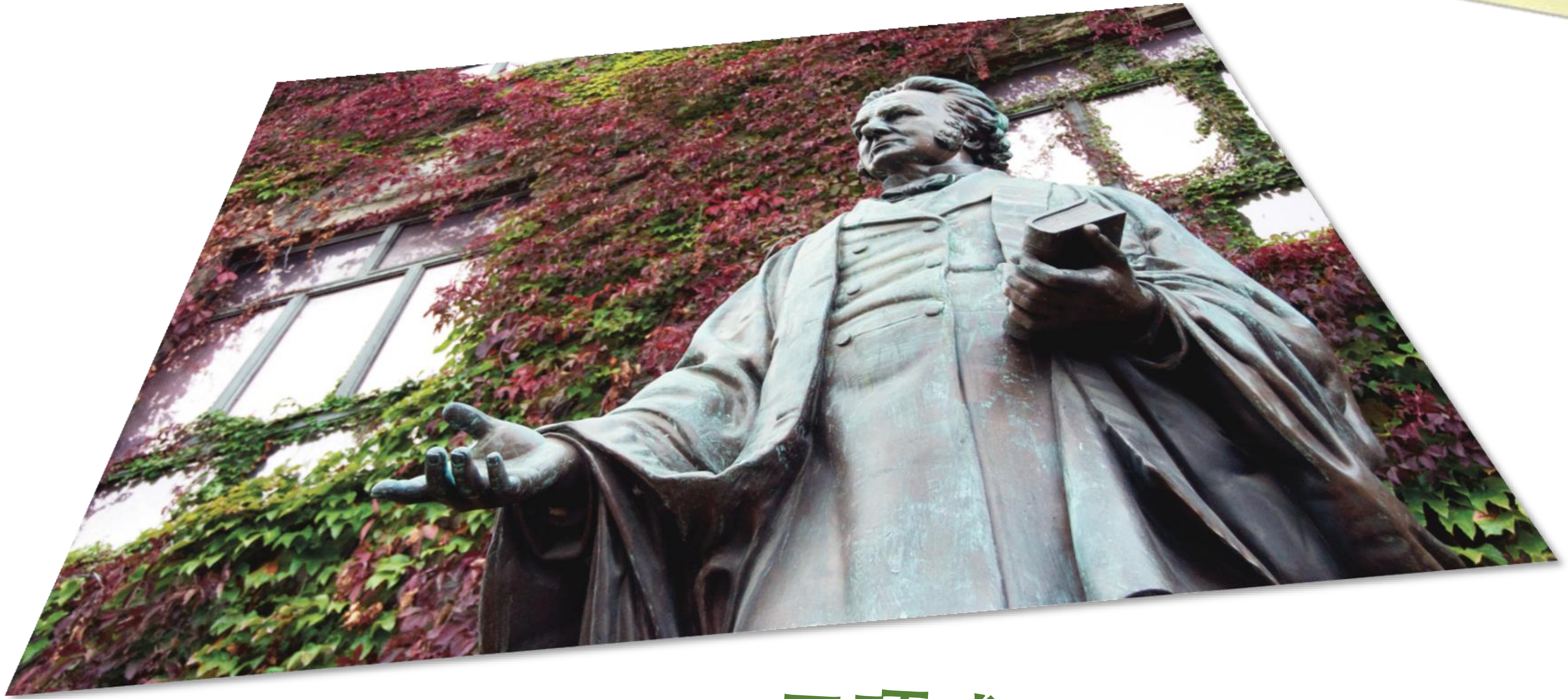
1/4尺寸 (放大4倍)

高斯
金字塔

高斯
金字塔

重复平滑和下采样

高斯
金字塔



二项式

重复平滑和下采样

## 二项式滤波器：

二项式系数仅使用整数提供了一个高斯系数的紧凑近似值

## 二项式滤波器：

二项式系数仅使用整数提供了一个高斯系数的紧凑近似值

$$[1\ 1]$$

## 二项式滤波器：

二项式系数仅使用整数提供了一个高斯系数的紧凑近似值

$$[1\ 1]$$

$$[1\ 1] * [1\ 1]$$

## 二项式滤波器：

二项式系数仅使用整数提供了一个高斯系数的紧凑近似值

$$[1\ 1]$$

$$[1\ 1] * [1\ 1]$$

$$[1\ 1] * [1\ 1] * [1\ 1]$$

## 二项式滤波器：

二项式系数仅使用整数提供了一个高斯系数的紧凑近似值

$$[1\ 1]$$

$$[1\ 1] * [1\ 1]$$

$$[1\ 1] * [1\ 1] * [1\ 1]$$

$$[1\ 1] * [1\ 1] * [1\ 1] * [1\ 1]$$

## 二项式滤波器：

　　二项式系数仅使用整数提供了一个高斯系数的紧凑近似值

$$[1\ 1]$$

$$[1\ 1] * [1\ 1]$$

$$[1\ 1] * [1\ 1] * [1\ 1]$$

$$[1\ 1] * [1\ 1] * [1\ 1] * [1\ 1]$$

$$\vdots$$

## 二项式滤波器：

二项式系数仅使用整数提供了一个高斯系数的紧凑近似值

$$[1\ 1] \qquad \sigma_1^2 = \frac{1}{4}$$

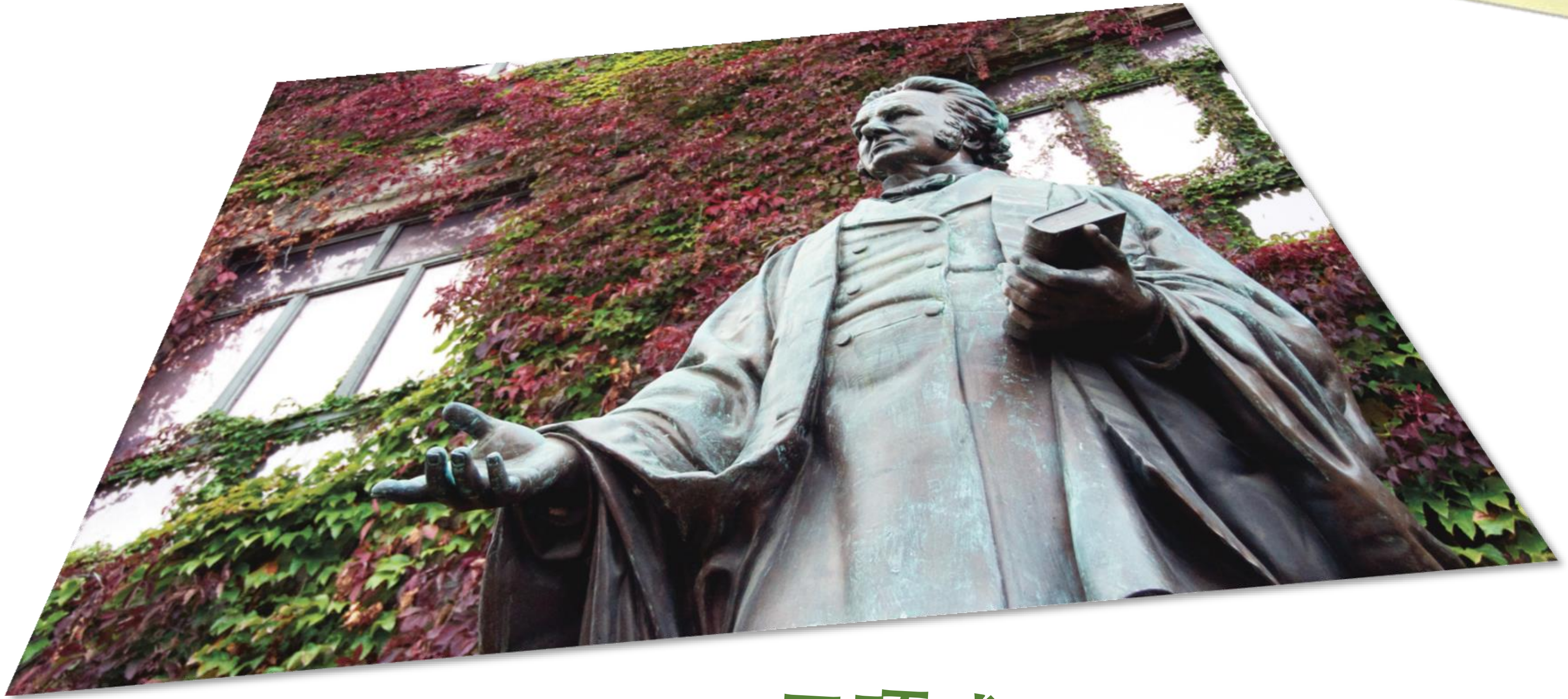
$$[1\ 2\ 1] \qquad \sigma_1^2 = \frac{1}{2}$$

$$[1\ 3\ 3\ 1] \qquad \sigma_1^2 = \frac{3}{4}$$

$$[1\ 4\ 6\ 4\ 1] \qquad \sigma_1^2 = 1$$

$$\vdots$$

高斯
金字塔



二项式

重复平滑和下采样

高斯
金字塔

二项式

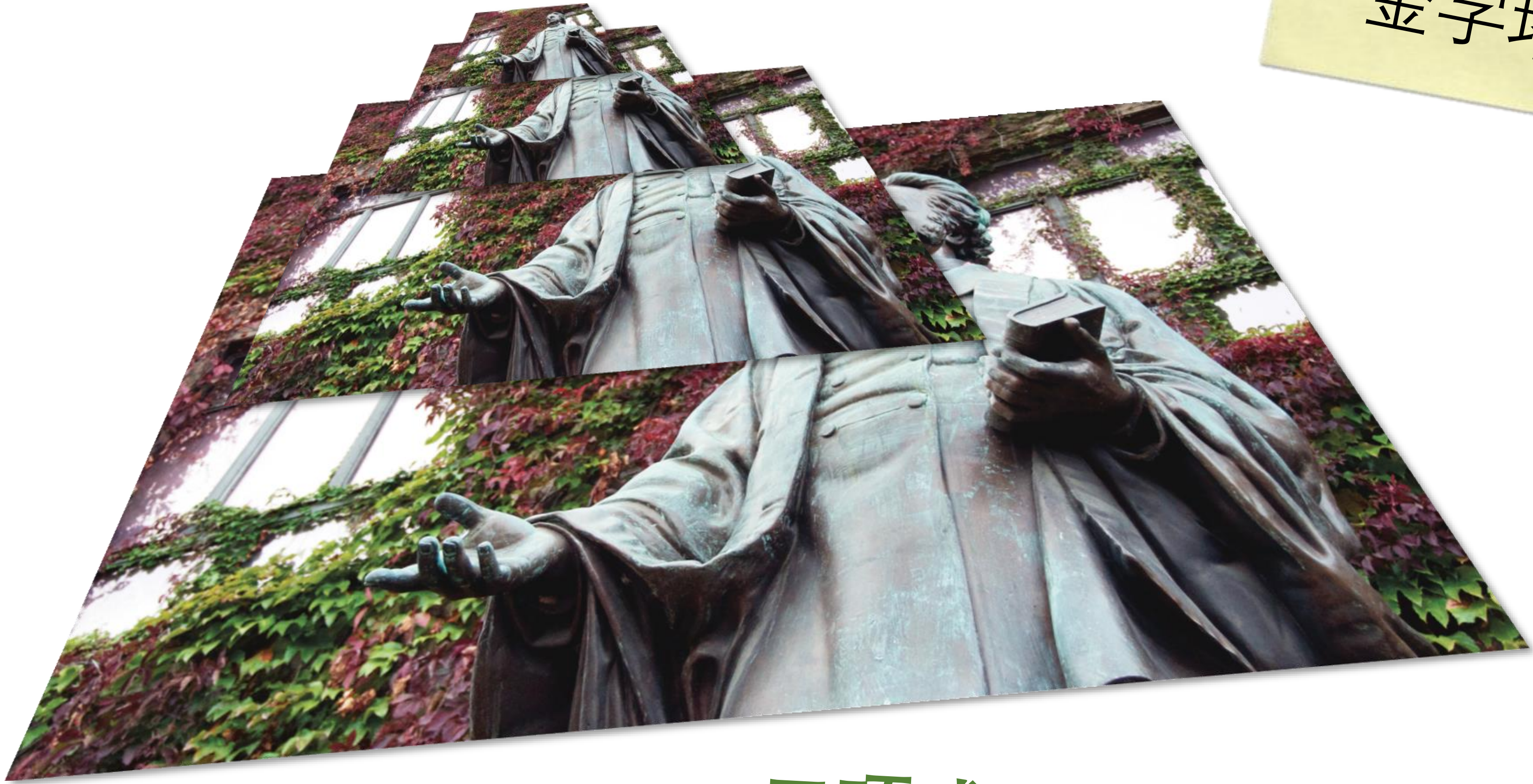
重复平滑和下采样

高斯
金字塔

二项式

重复平滑和下采样

高斯
金字塔

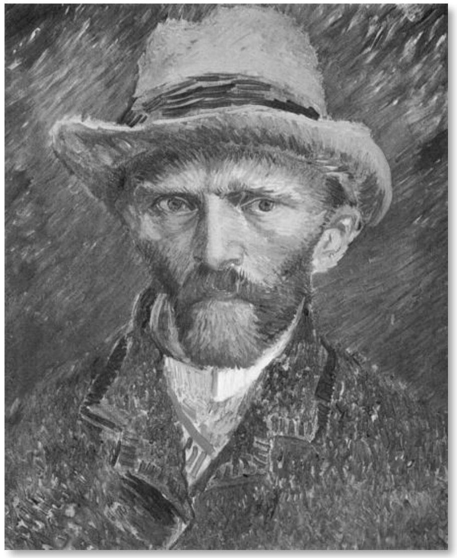


二项式

重复平滑和下采样

# Python时间

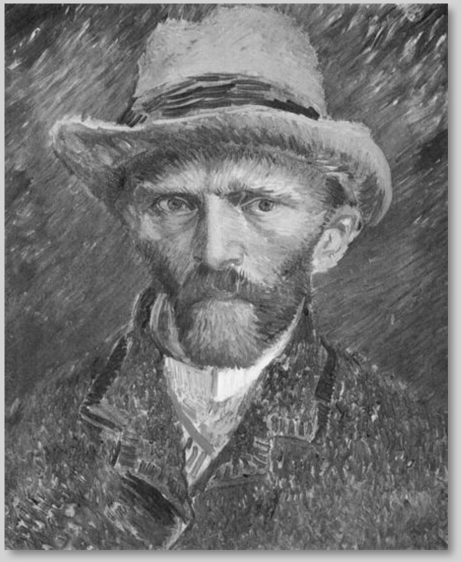
高斯
金字塔



**1/2尺寸**

```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smooth', im_s), cv2.waitKey(0)
```

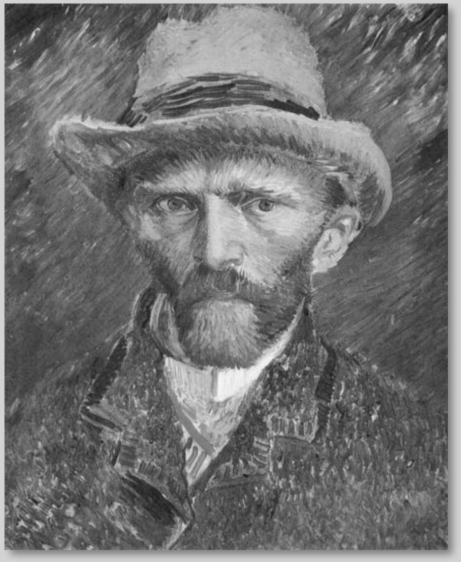
高斯
金字塔



## 1/2尺寸

```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smooth', im_s), cv2.waitKey(0)
```

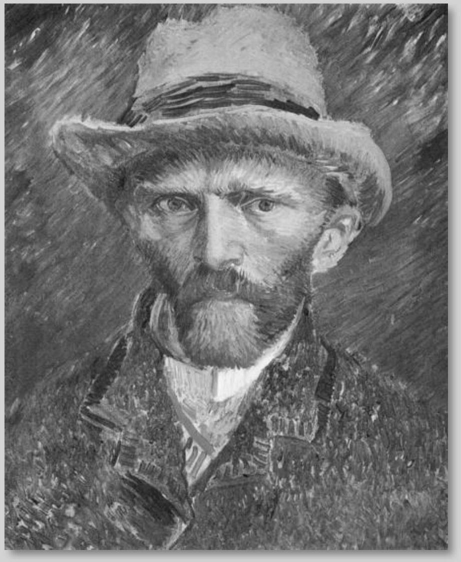
高斯
金字塔



1/2尺寸

```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smooth', im_s), cv2.waitKey(0)
```

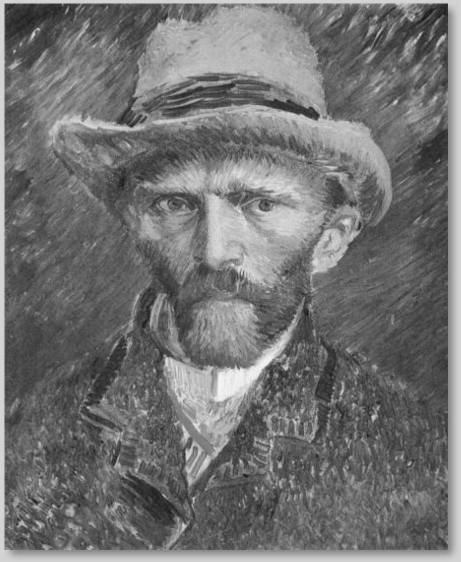
高斯
金字塔



1/2尺寸

```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smooth', im_s), cv2.waitKey(0)
```

高斯
金字塔

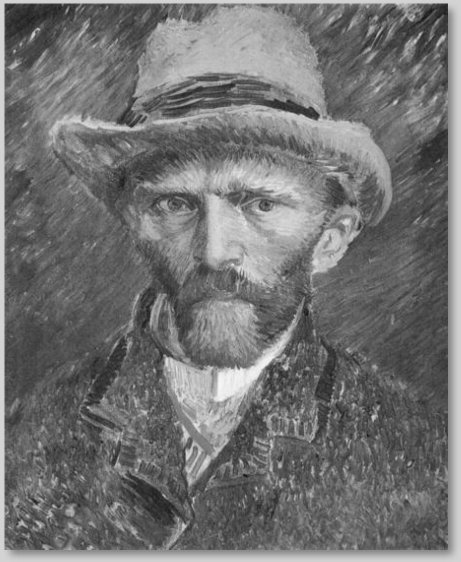


**1/2尺寸**

```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smooth', i                       waitKey(0)
```

**可分离滤波器**

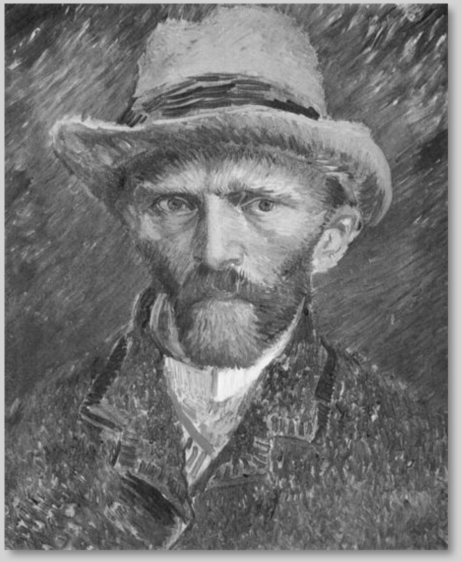
高斯
金字塔


**1/2尺寸**

```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smo                 cv2.waitKey(0)
```

**下采样**

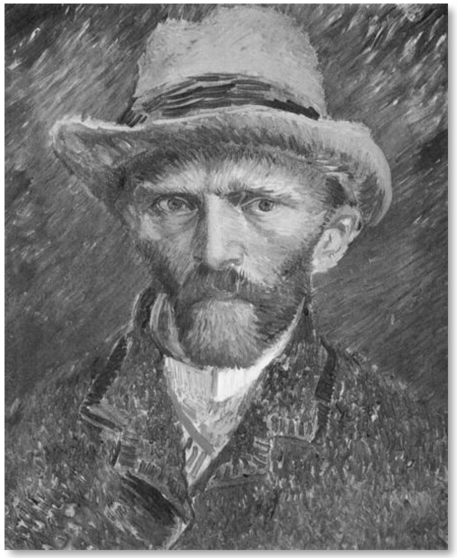
高斯
金字塔



1/2尺寸

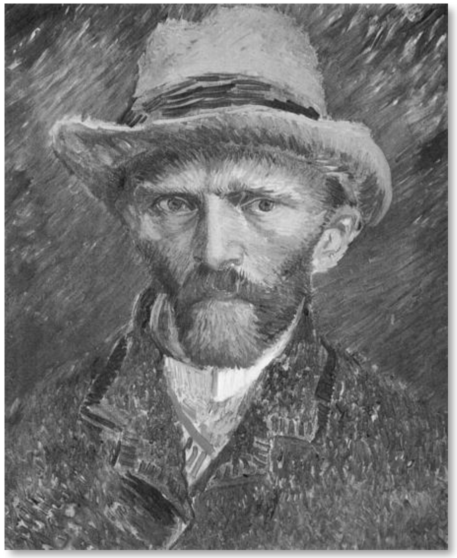
```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smooth', im_s), cv2.waitKey(0)
```

高斯
金字塔


**1/2尺寸**

```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smooth', im_s), cv2.waitKey(0)
```

高斯
金字塔

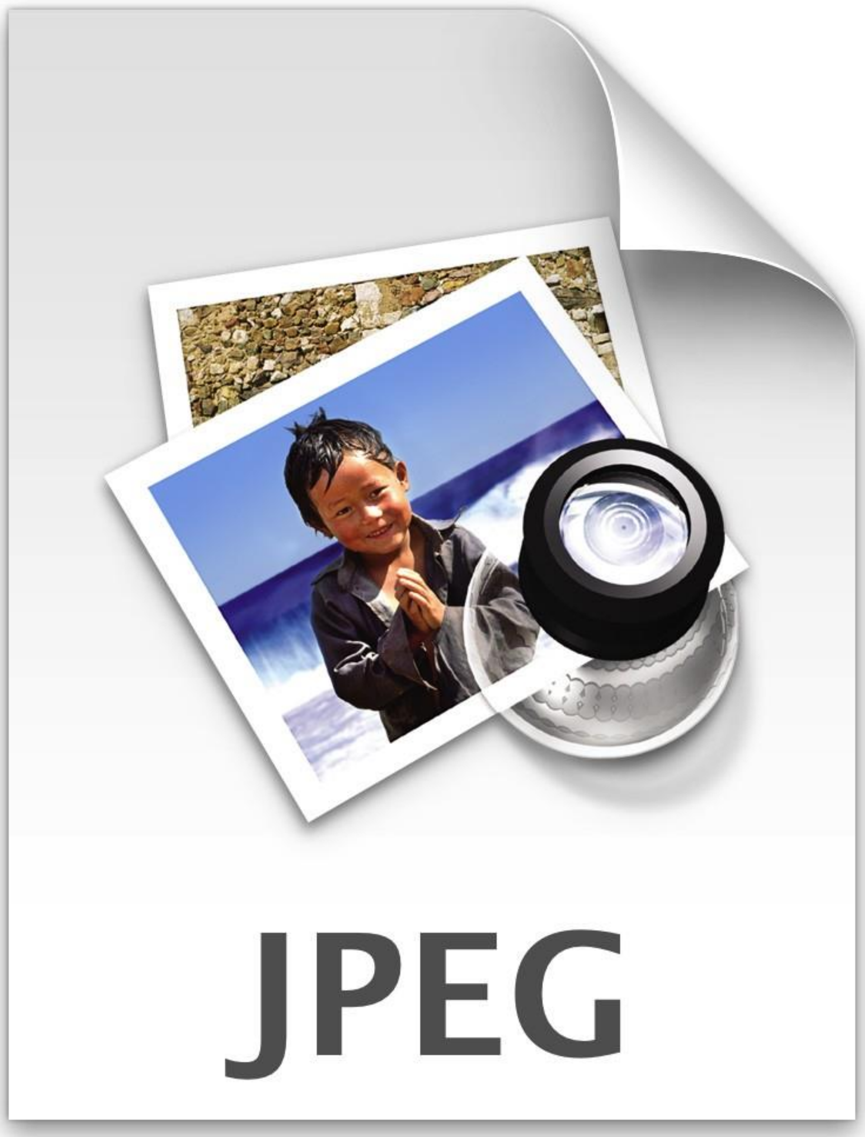

1/2尺寸

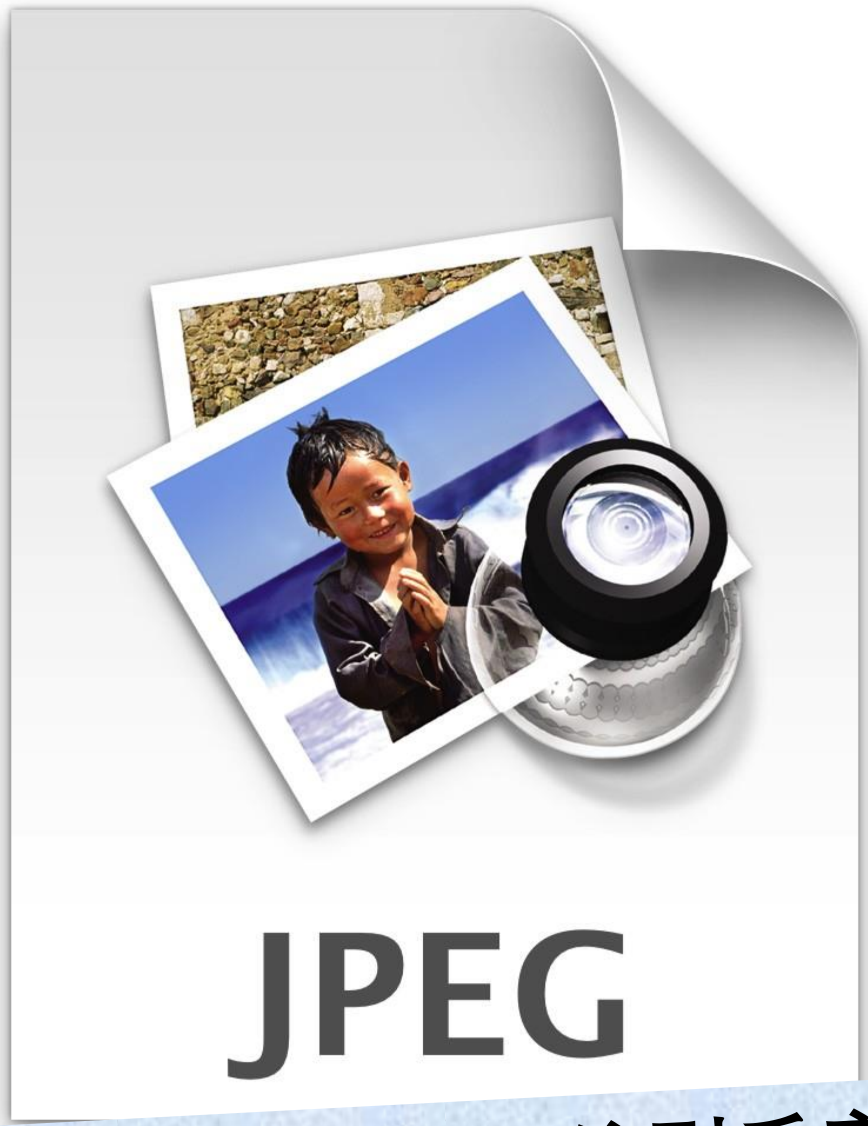
```
>>> f = (1/16)*np.array([1, 4, 6, 4, 1])  # Binomial filter
>>> tmp = cv2.filter2D(im, -1, f, cv2.BORDER_REFLECT)
>>> im_s = cv2.filter2D(tmp, -1, f.T, cv2.BORDER_REFLECT)
>>> im_s = im_s[::2, ::2]
>>> cv2.imshow('Smooth', im_s), cv2.waitKey(0)
```

迭代上述过程以实现额外的层次

## Python时间

已结束


JPEG



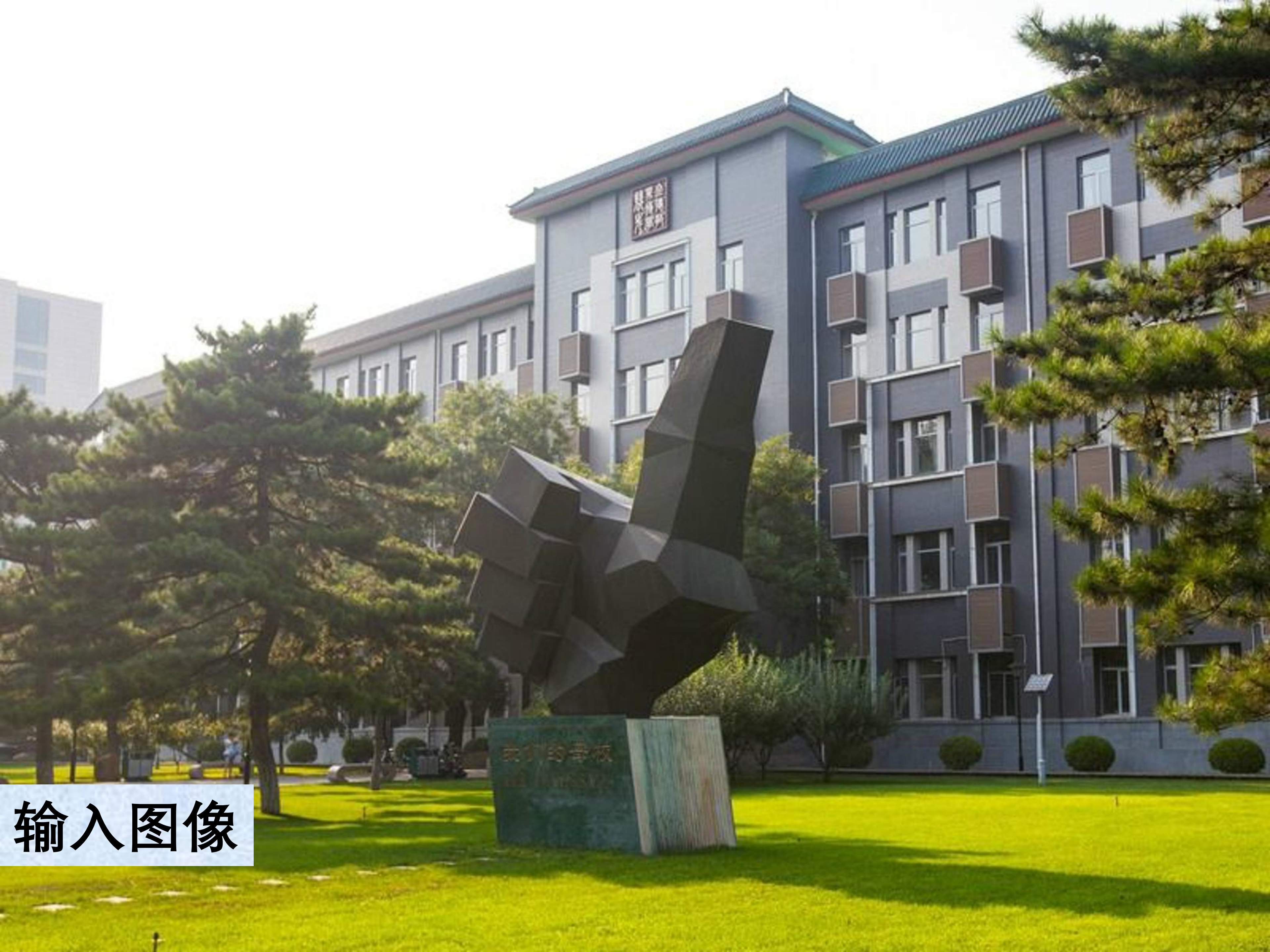
将图像从兆字节压缩到千字节

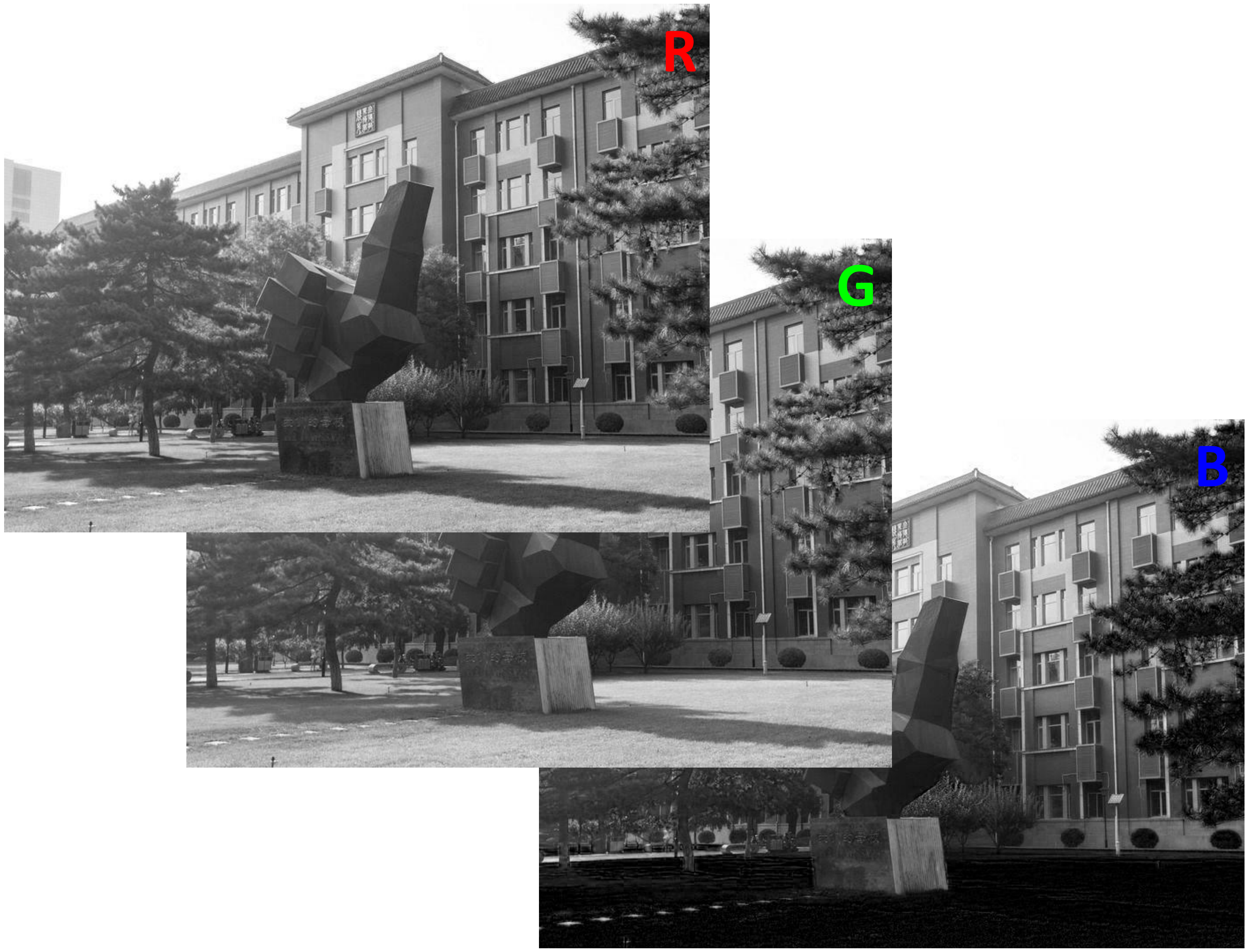
## 5

## 主要步骤

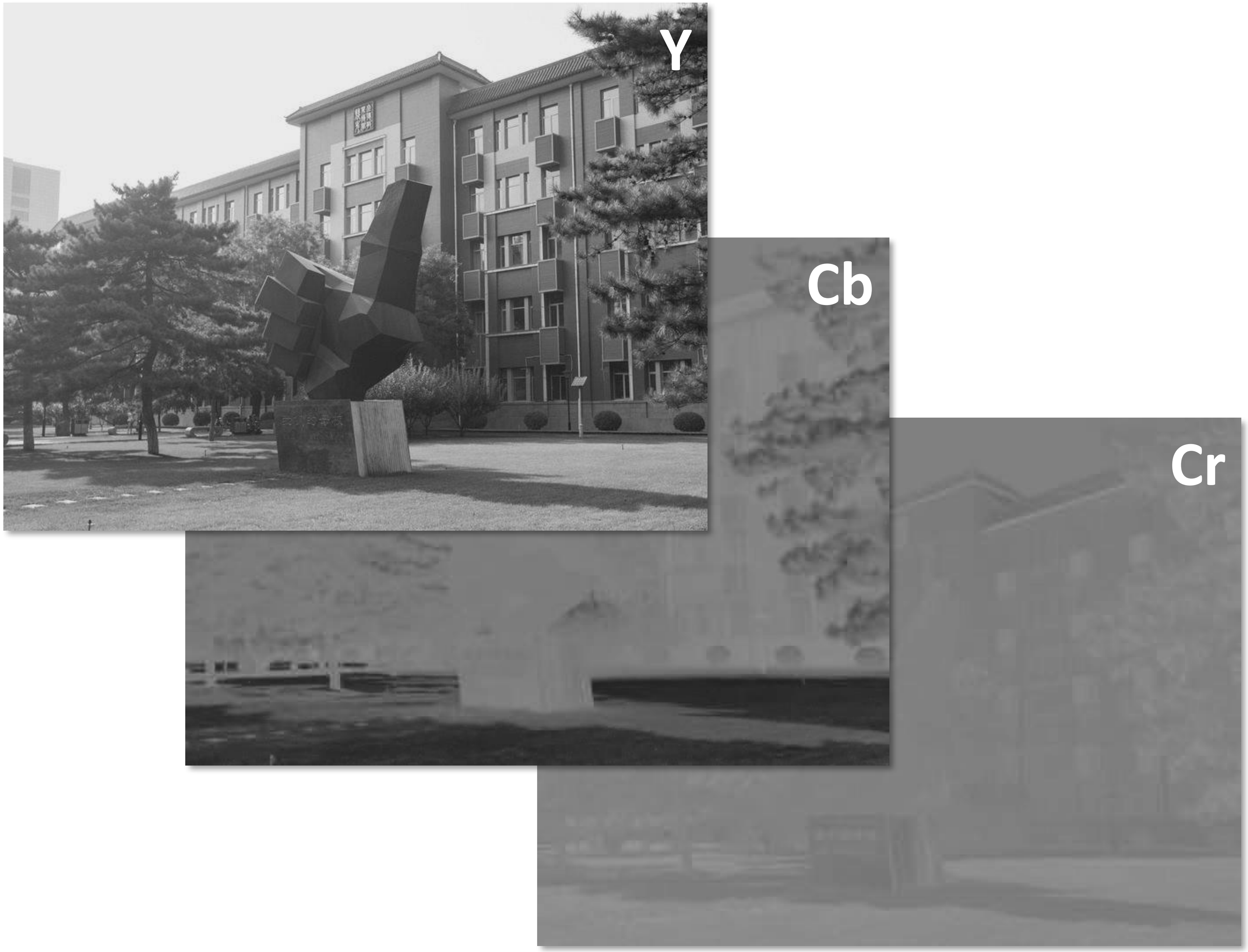
## 步骤1

转换颜色空间


输入图像

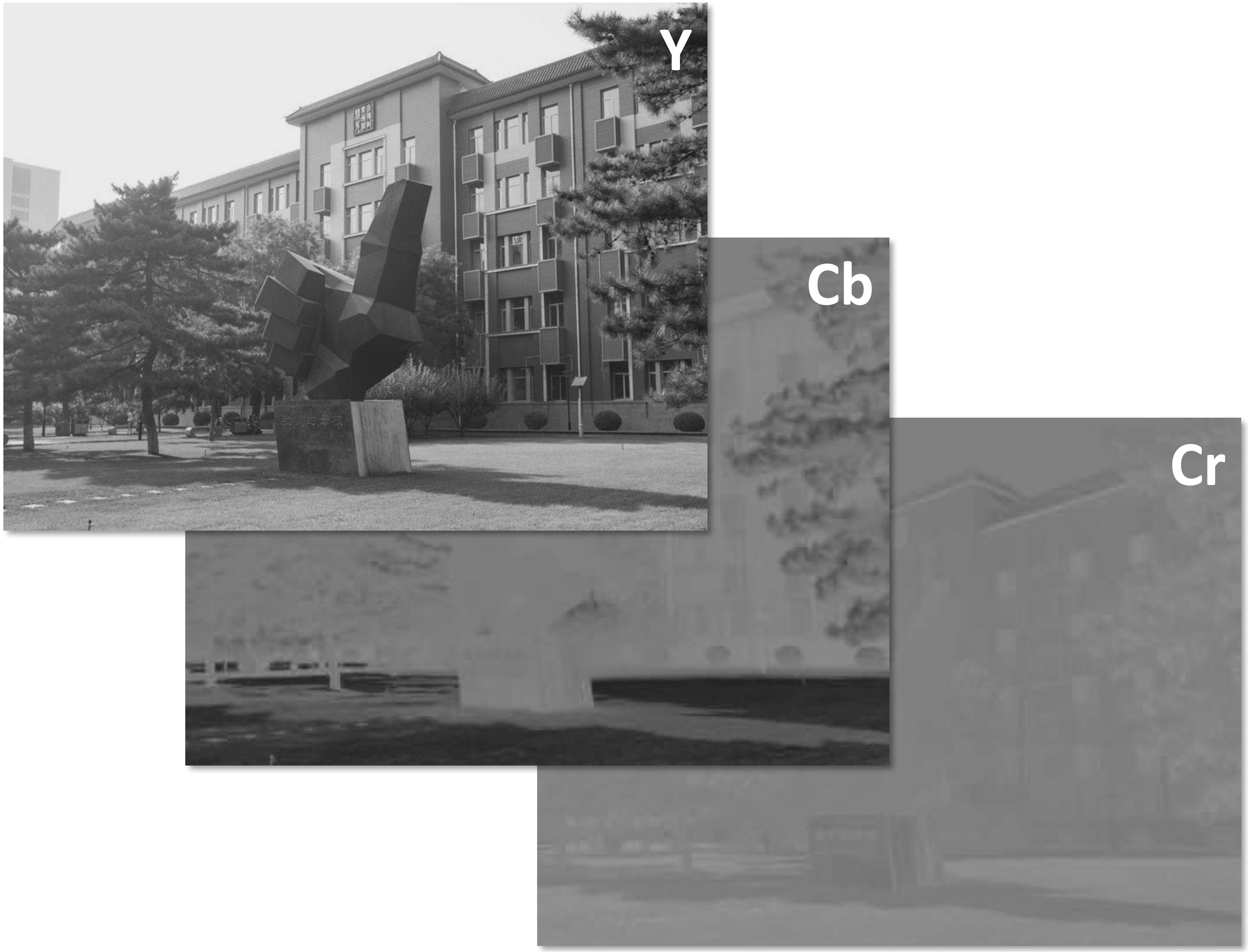

R
G
B

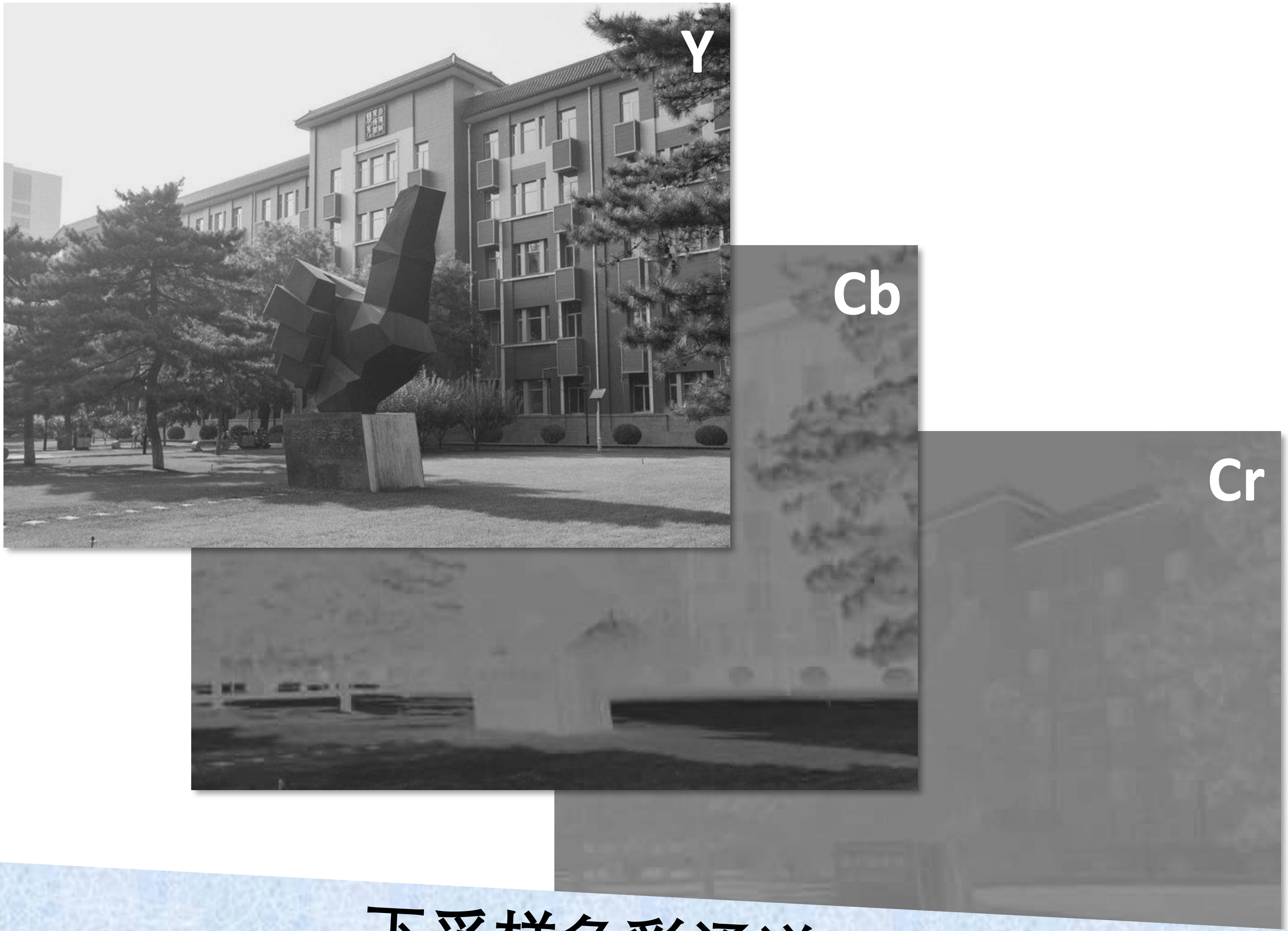
## 将RGB图像转换成YCrCb


Y
Cb
Cr

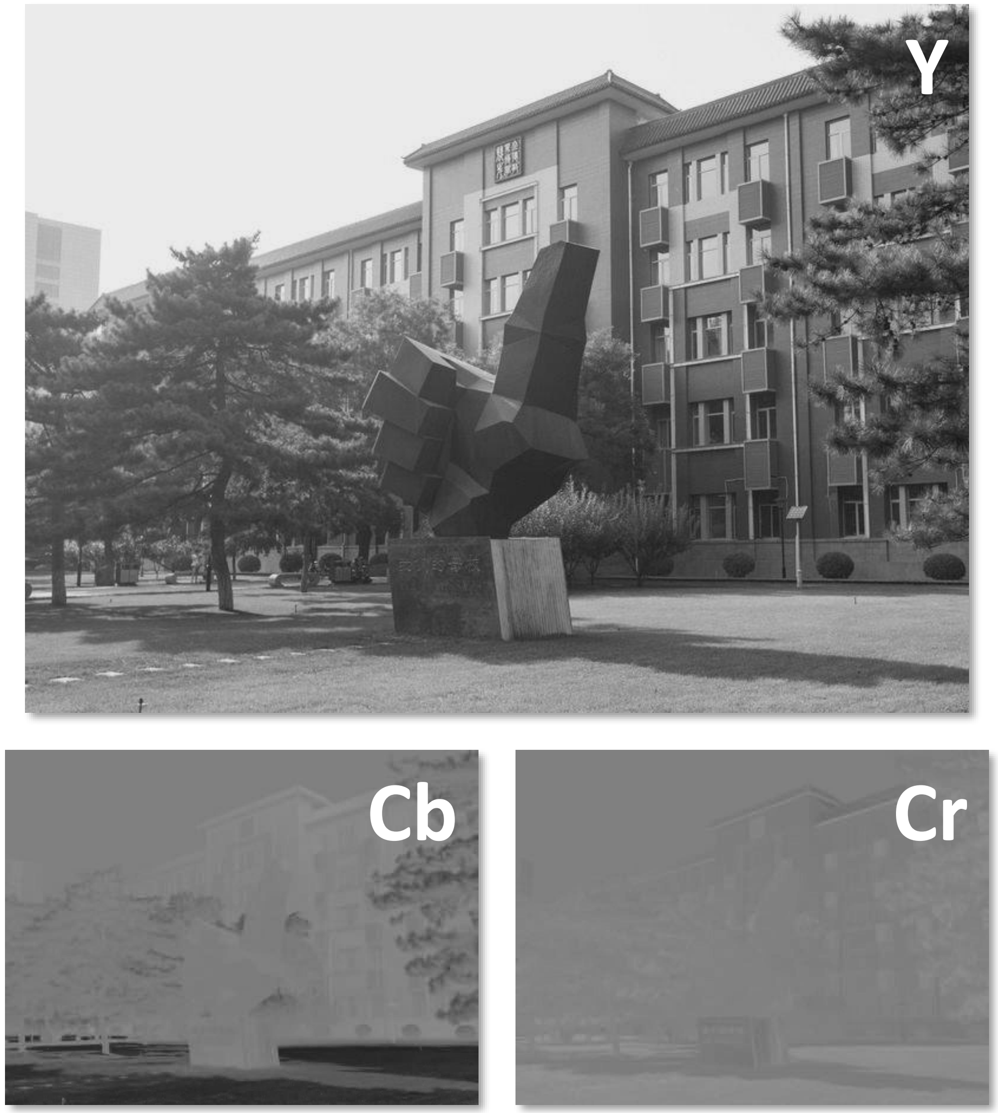
## 步骤2

重采样色彩通道


Y
Cb
Cr



下采样色彩通道至一半


Y
Cb
Cr

## 步骤3

对每个8 × 8块计算DCT

我们的母校

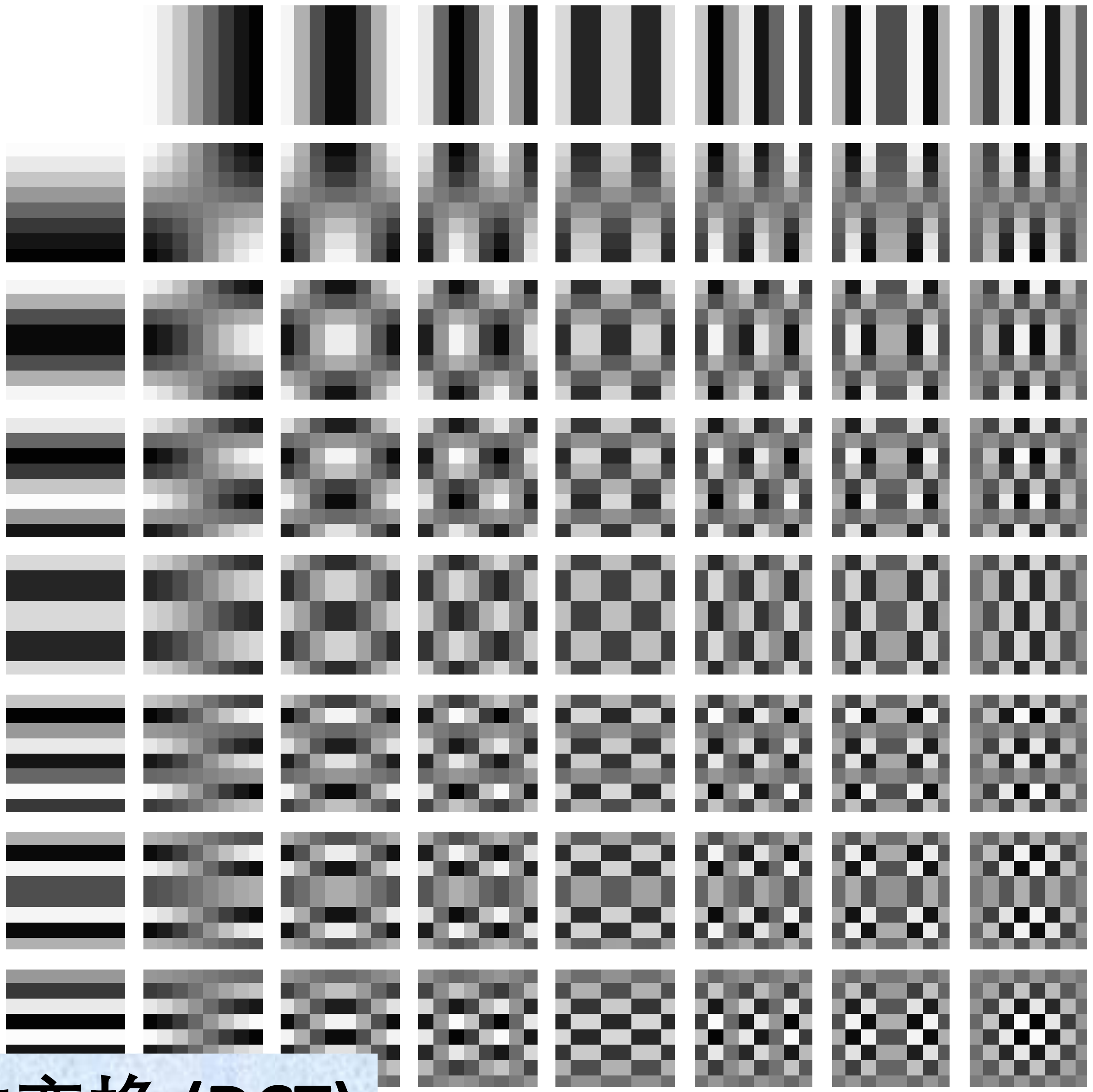
## 步骤3

**对每个8 × 8块计算DCT**

## 步骤3

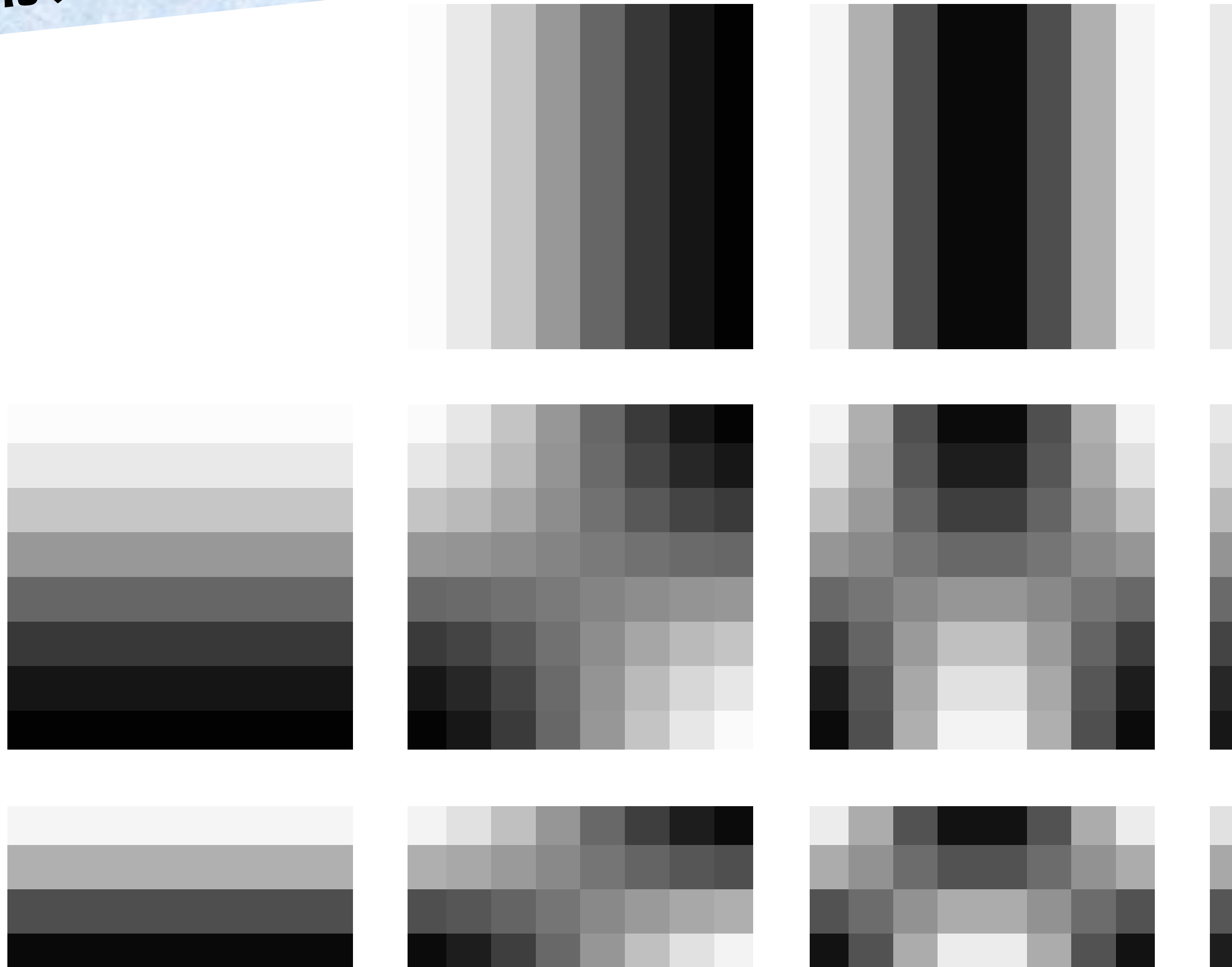
**对每个8 × 8块计算DCT**

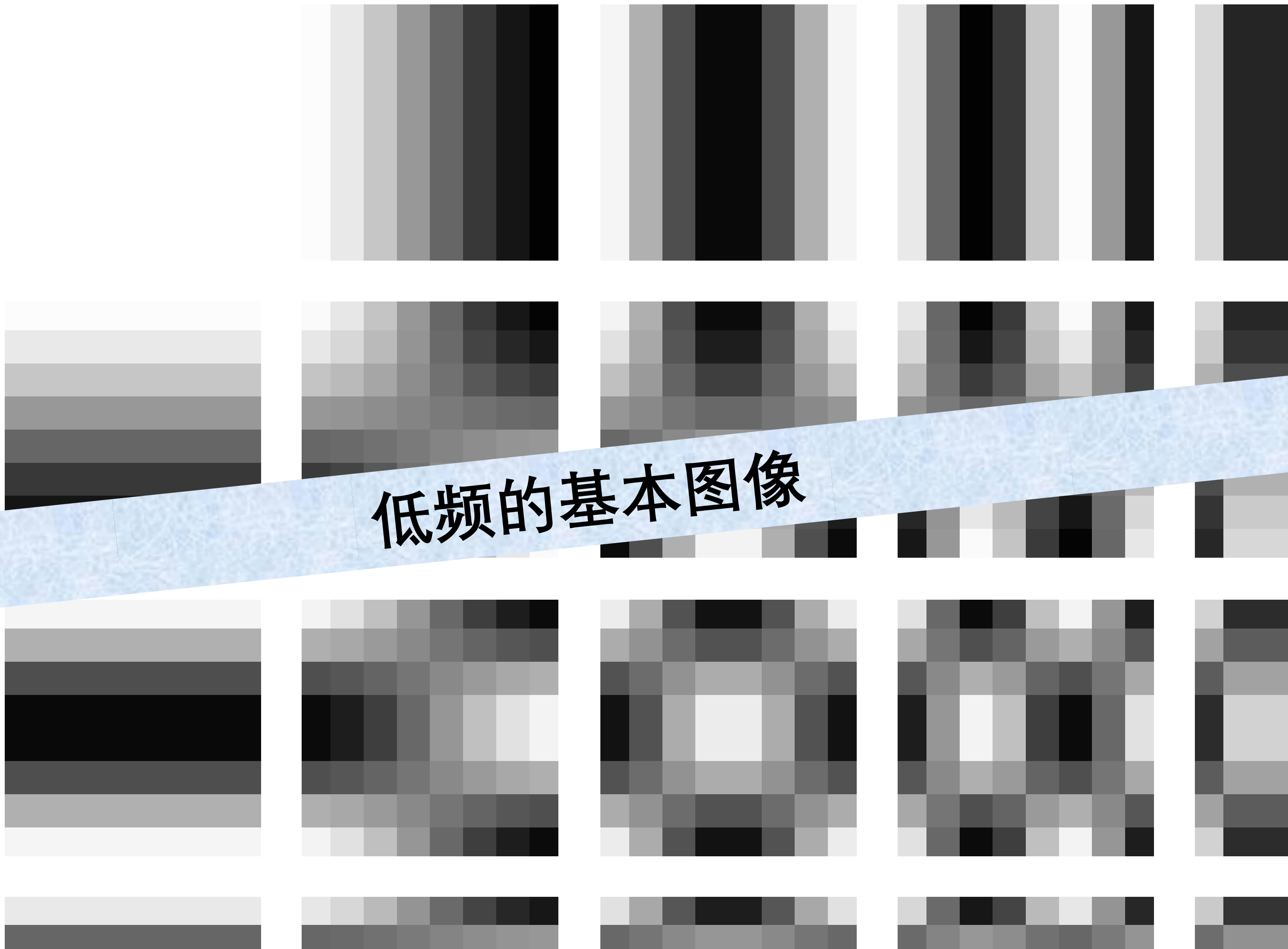
**将图像块映射到一个实数集**

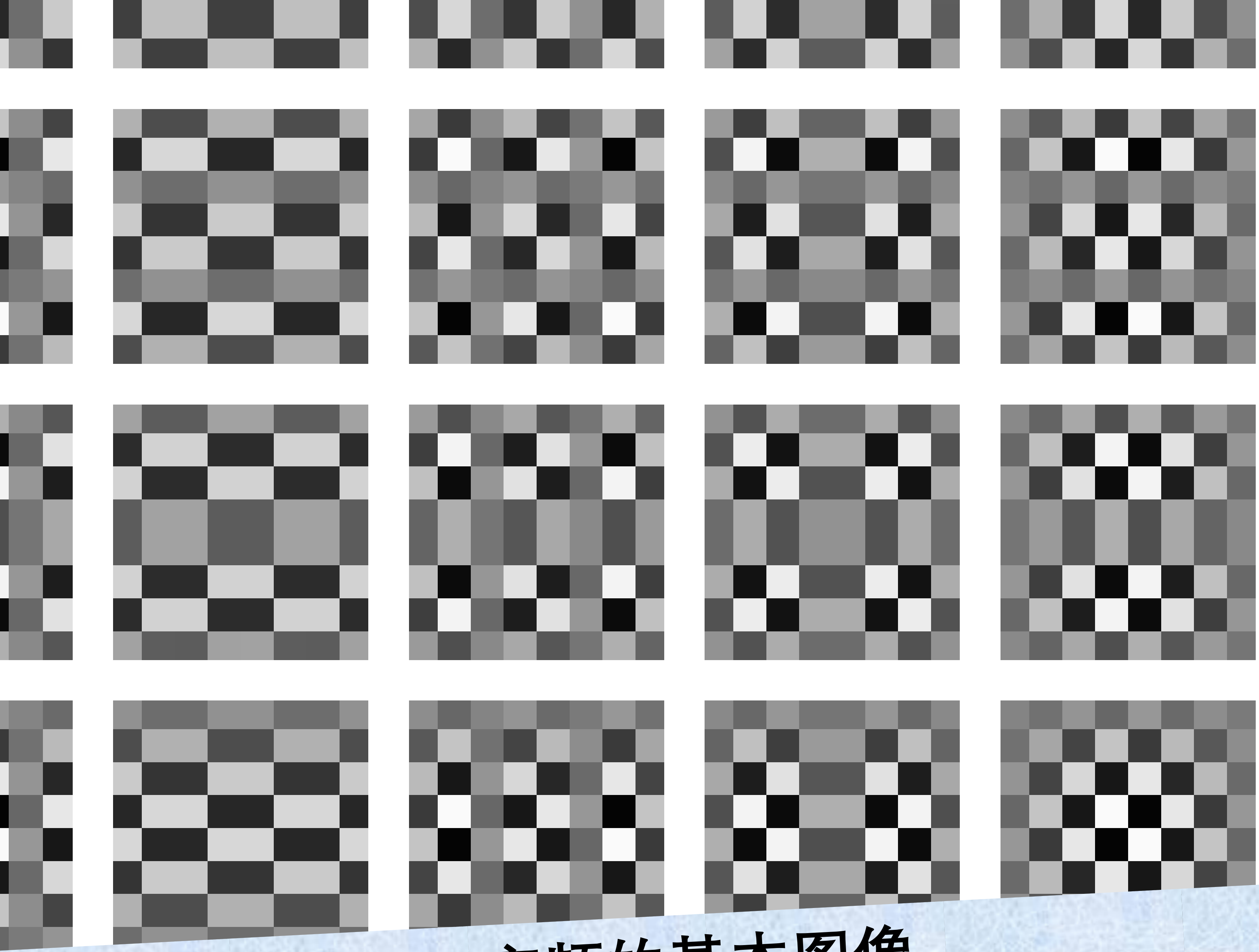


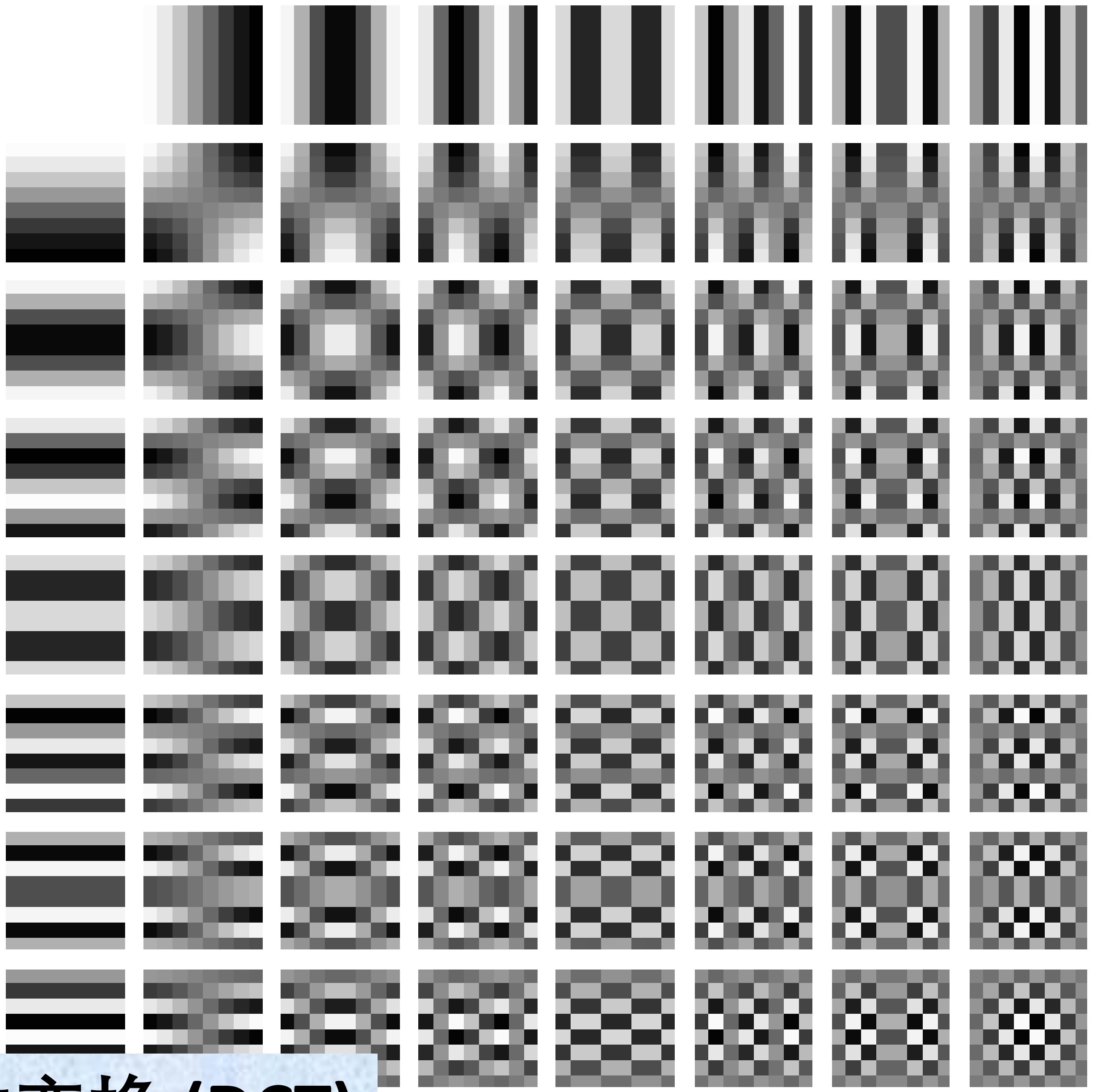
## 离散余弦变换 (DCT)

## 直流分量捕获平均强度

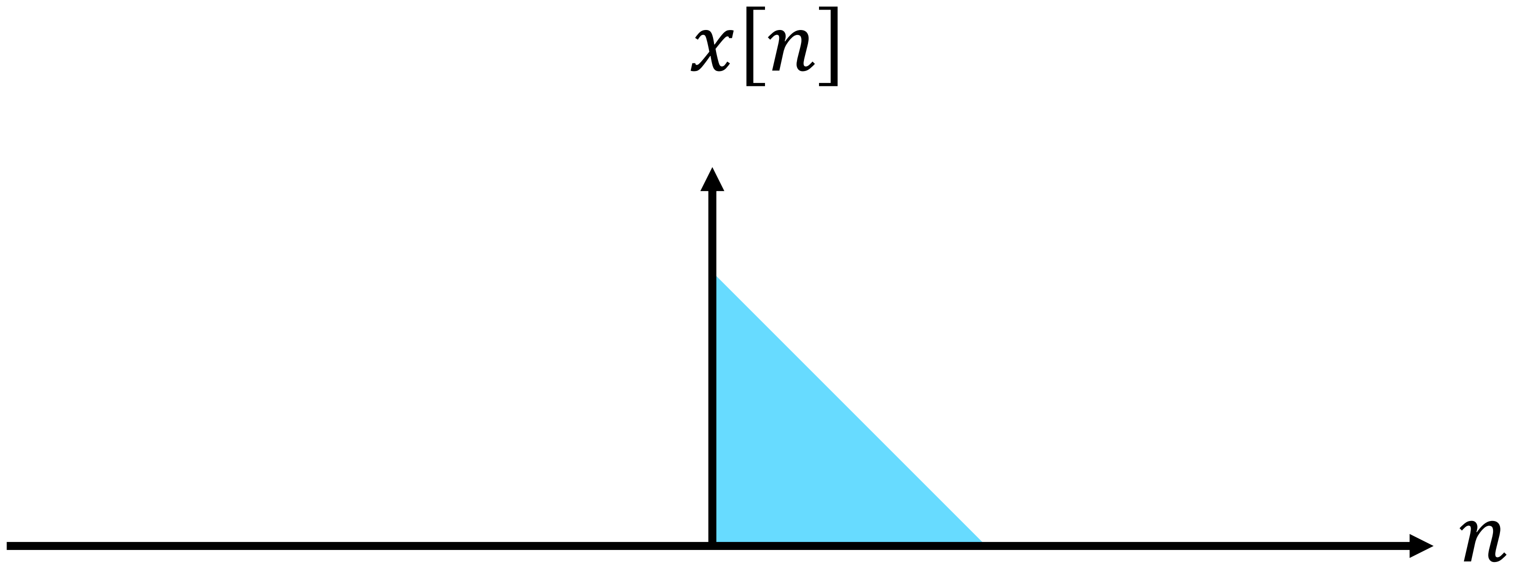


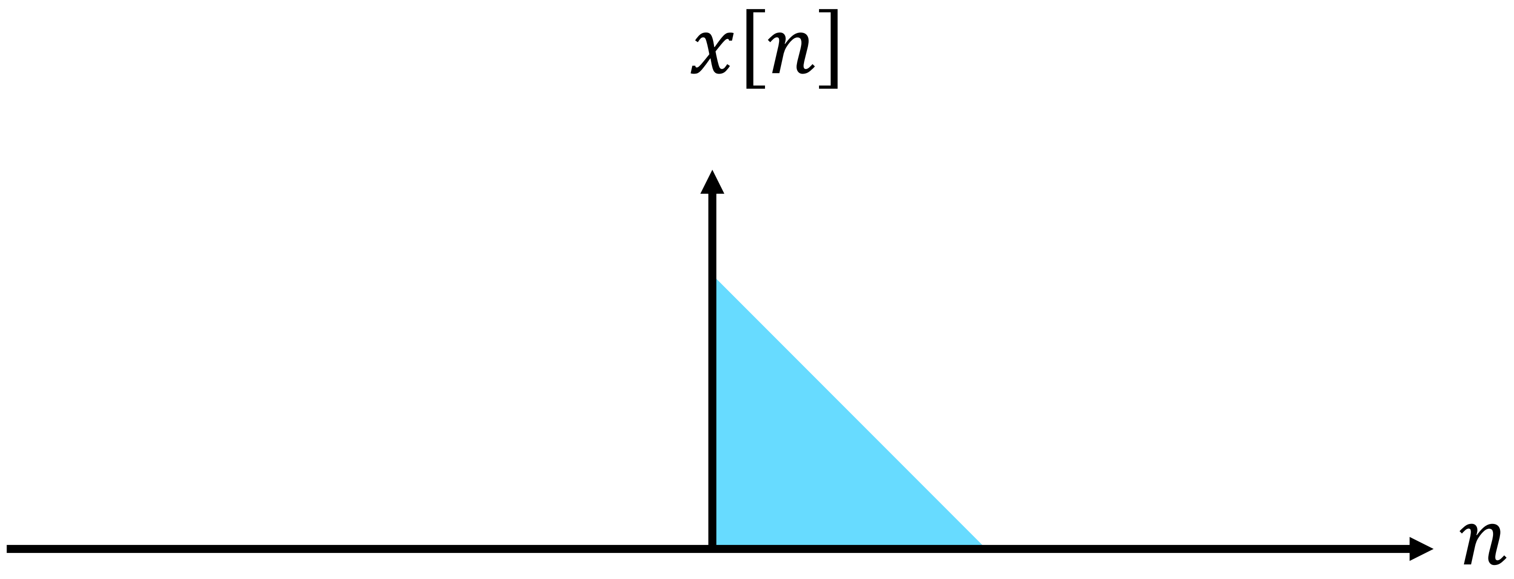

低频的基本图像


高频的基本图像

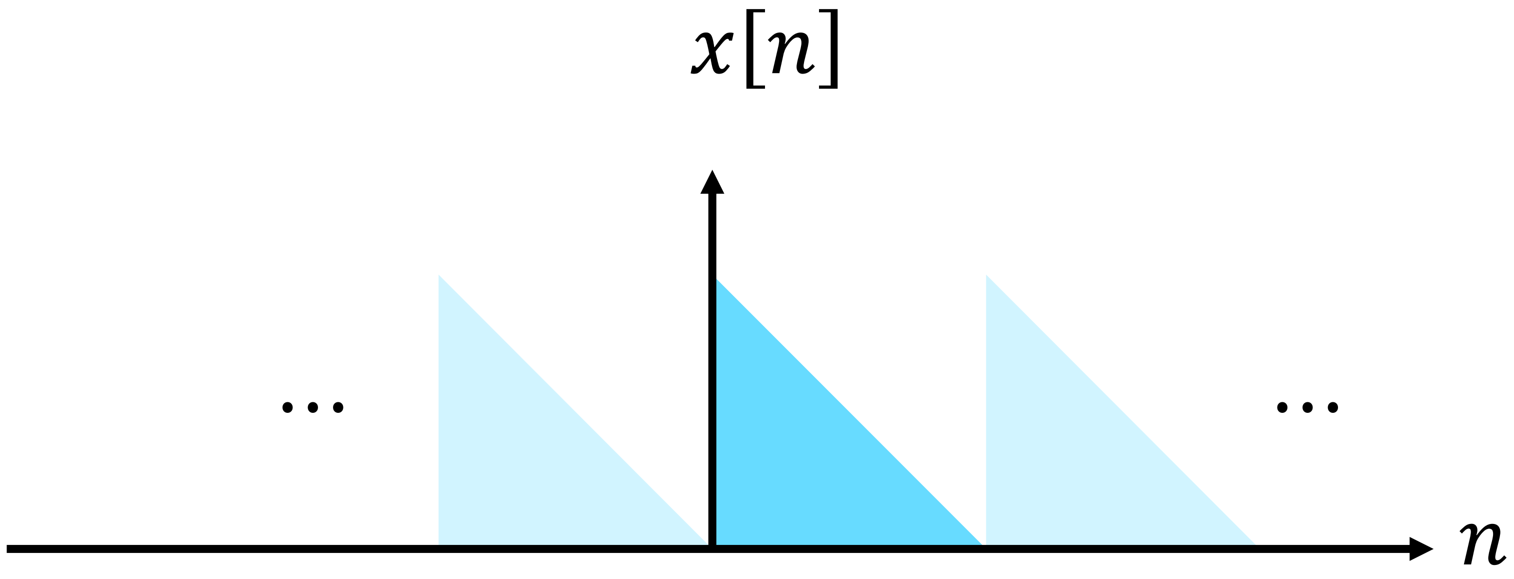


# 离散余弦变换 (DCT)

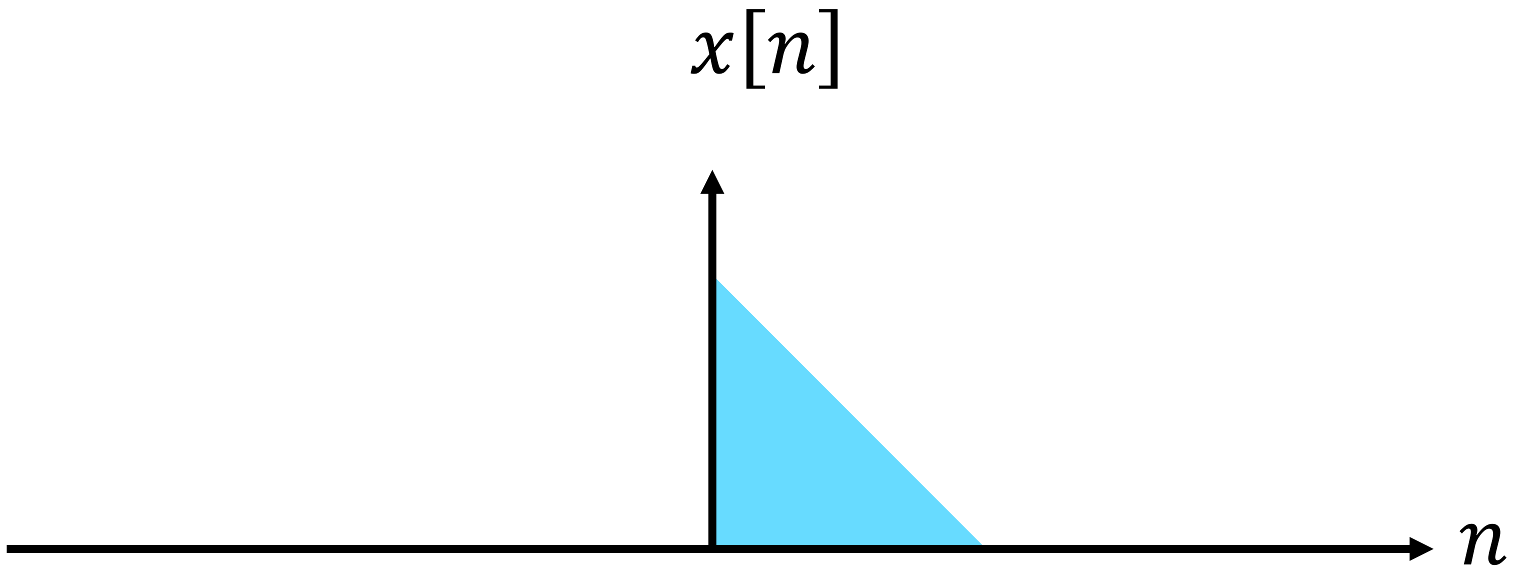

$x[n]$
$n$

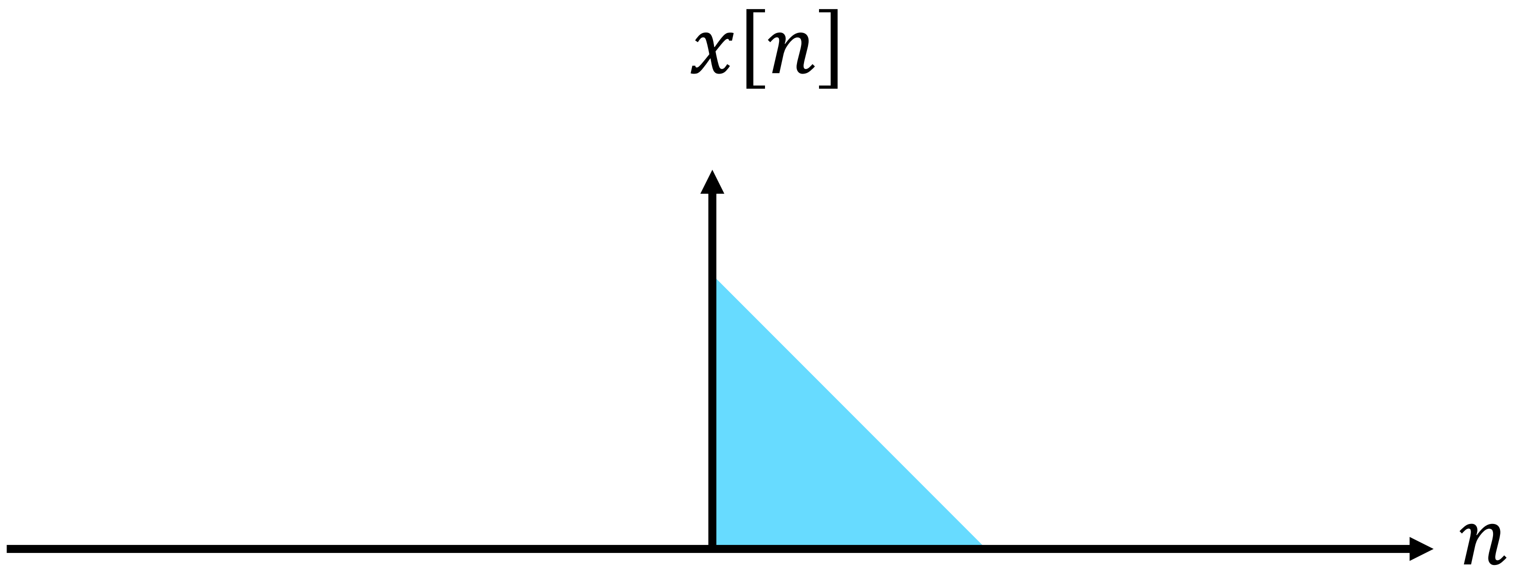


DFT假设超出边界的周期性扩展

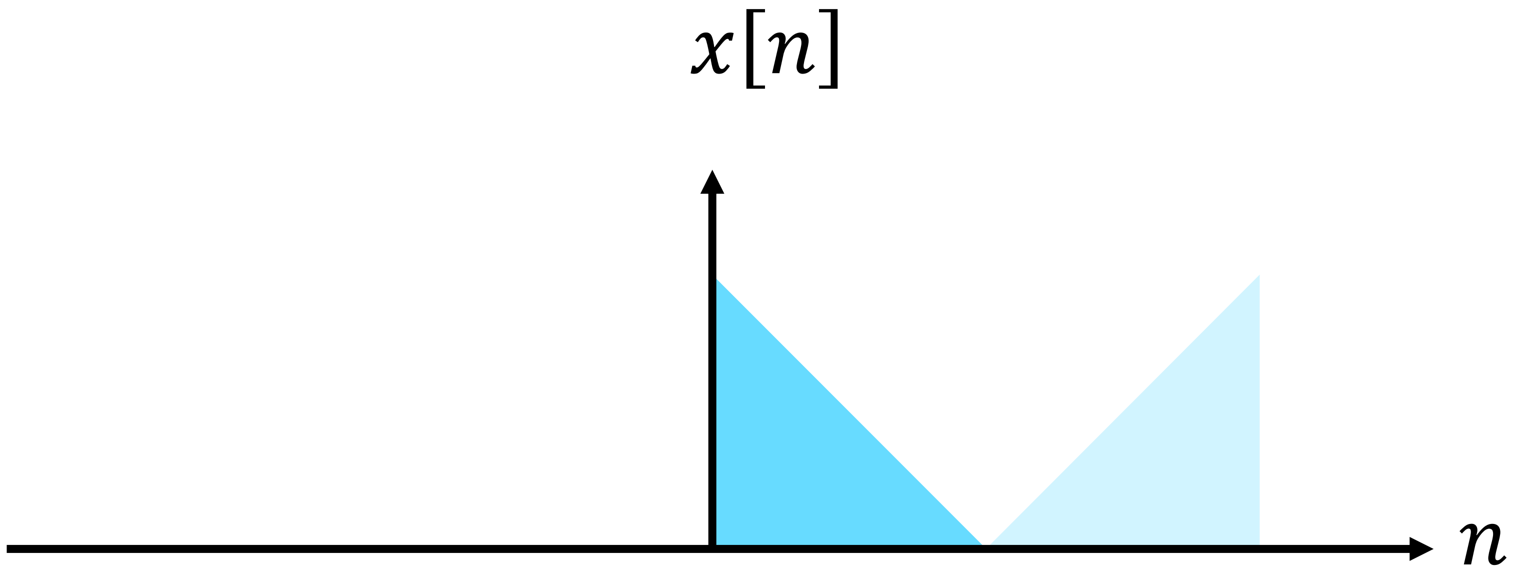


DFT假设超出边界的周期性扩展

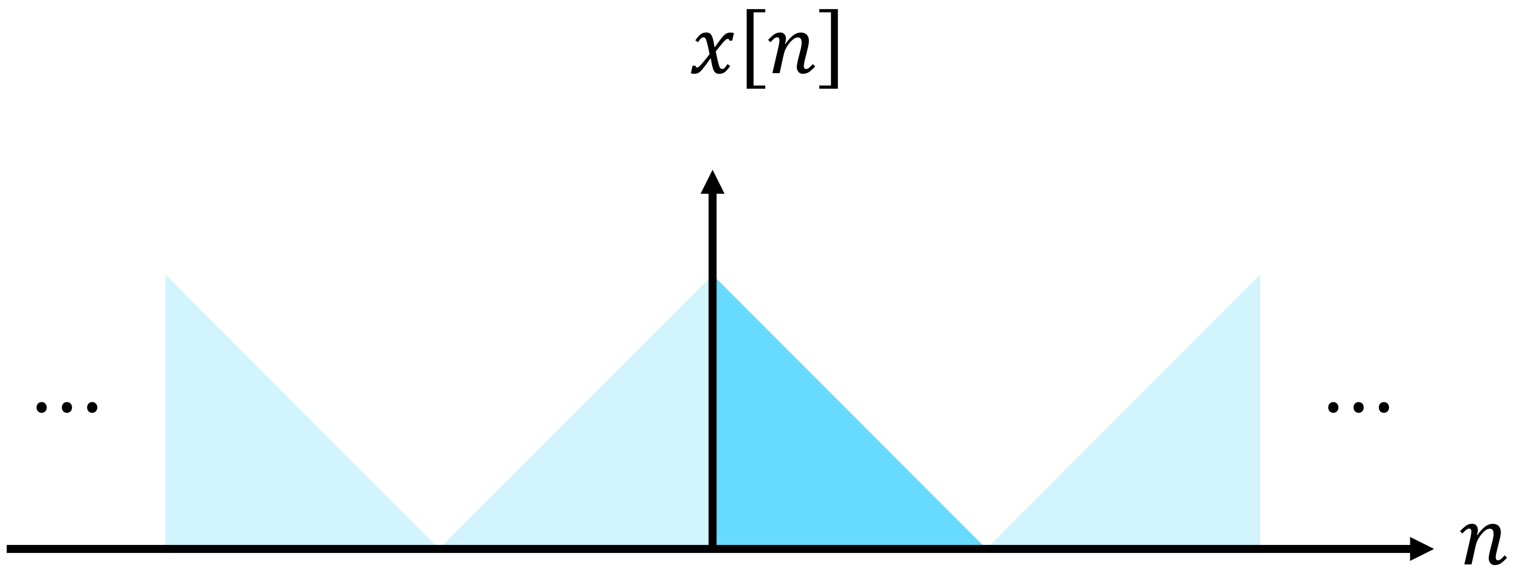

$x[n]$
$n$



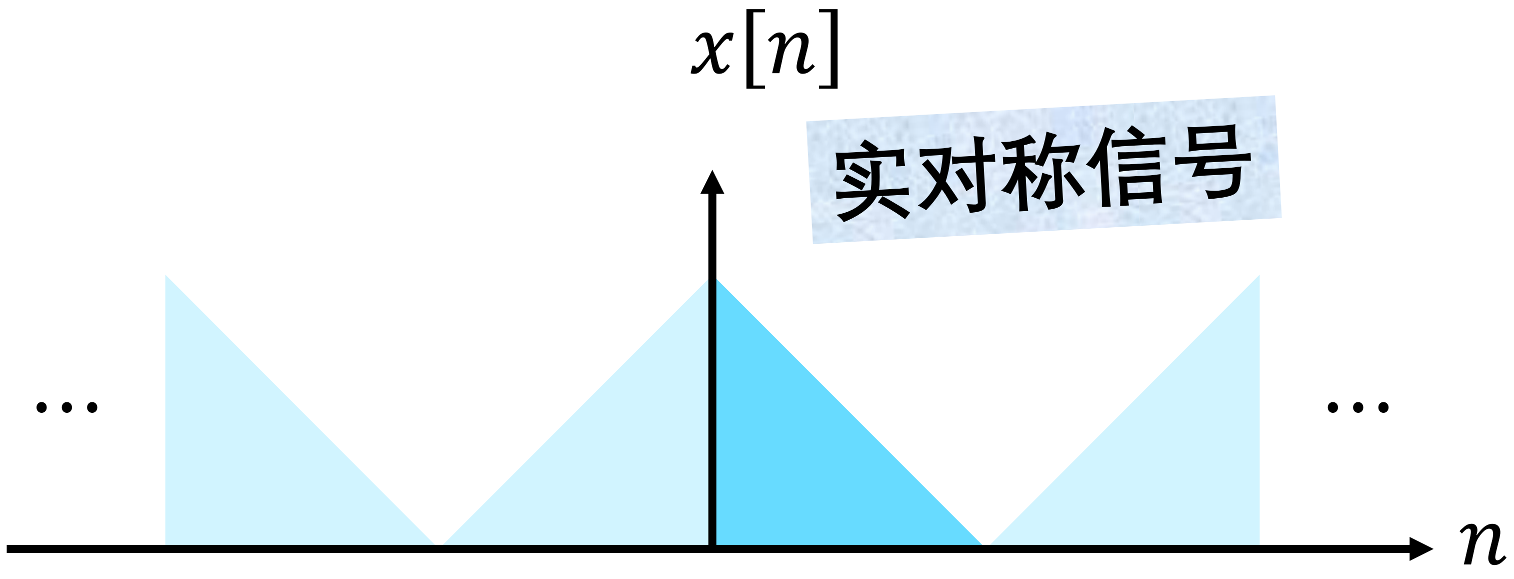
DCT加上其镜像，然后周期性扩展



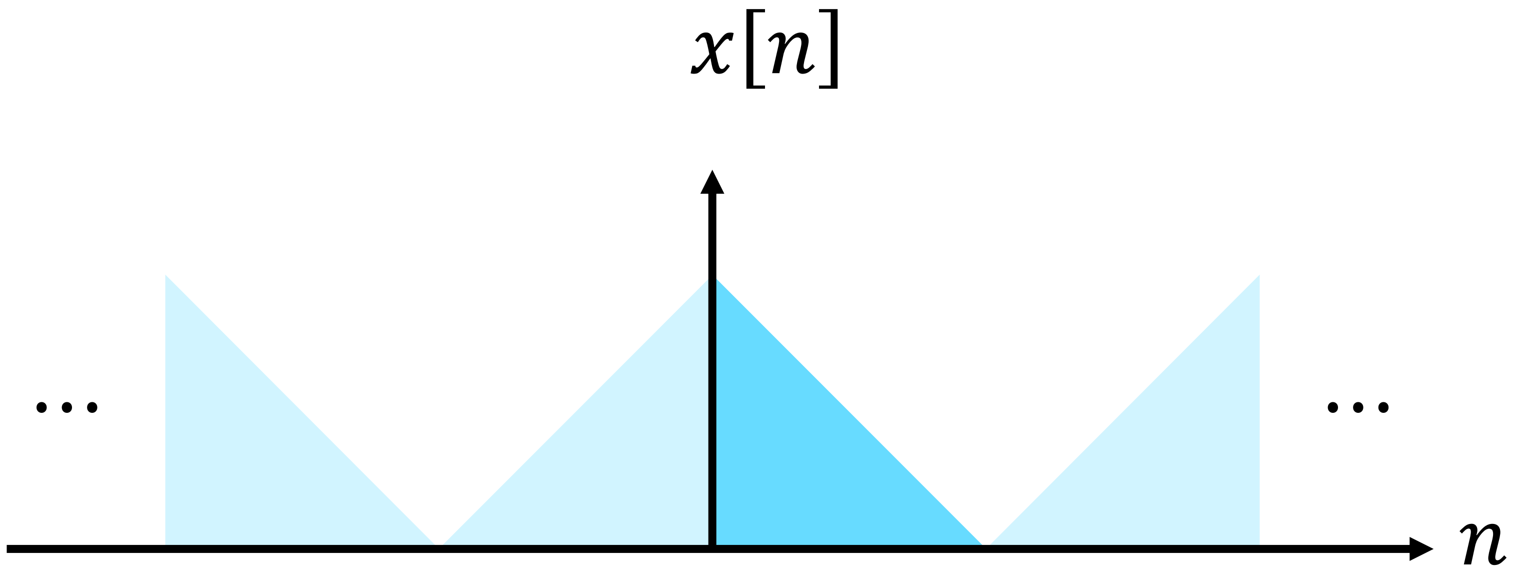
**DCT加上其镜像，然后周期性扩展**



DCT加上其镜像，然后周期性扩展



DCT加上其镜像，然后周期性扩展



## 在不使用复数的情况下执行DFT

## 步骤4

量化DCT系数

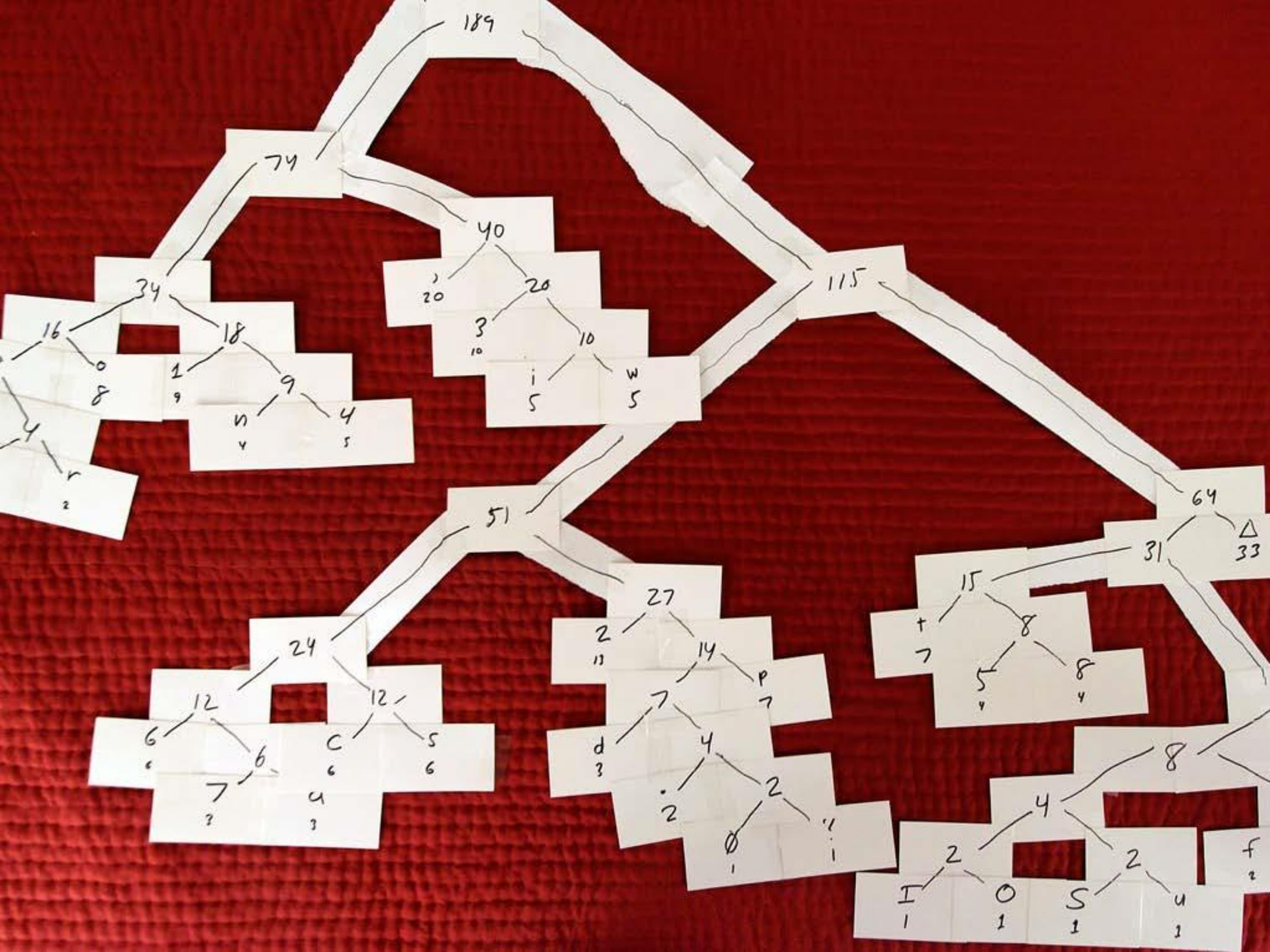
# 步骤4

## 量化DCT系数

量化水平取决于频率

## 步骤5

使用哈夫曼编码进行无损压缩



189
74
115
34
40
16
18
20
o
8
1
9
9
n
4
4
5
r
2
3
10
10
i
5
w
5
51
64
31
33
24
27
15
12
12
2
13
14
t
7
8
5
4
8
4
6
6
6
c
6
s
6
7
3
u
3
p
7
d
3
4
2
2
0
1
1
8
4
2
2
I
1
O
1
S
1
u
1
f
2