

# 计算机视觉

## 图像滤波



中国传媒大学

COMMUNICATION UNIVERSITY OF CHINA

# 作业 1

2115530096 计算机视觉  
作业 1: 图像形成  
最后期限: 2023 年 10 月 18 日 23:59  
(占期末成绩 10%)

此次作业是为了确保学生能够读取图像, 操纵像素, 并生成结果。作业必须**独立完成**。在使用 Python 函数时, 如需帮助, 请在加载库后, 在命令窗口输入“help(库名.函数名)”以获取说明文档。最后, 在操作图像时, 确保使用合适的类型转换 (即 float32 和 uint8)。

请将所有图像、程序打包到“你的姓名\_学号\_a1.zip”文件, 在最后期限前通过邮件发送到 [lifang8902@cuc.edu.cn](mailto:lifang8902@cuc.edu.cn), 每迟交 1 天扣 3 分。要求可以调用 a1\_script.py 输出全部结果。

1. 输入 (2分):
  - 1) 在 <http://sipi.usc.edu/database/database.php?volume=misc> 选择一张彩色图像, 尺寸不大于  $512 \times 512$ , 下载到 Python 工作目录;
  - 2) 创建 Python 文件, 并命名为“a1\_script.py”;
  - 3) 使用 cv2.imread 读取图像, 存储在变量 im 中, 并使用 cv2.imshow 显示;
  - 4) 将 im 转换成灰度图 (grayscale), 并显示, 详见 cv2.cvtColor;
2. 图像二维变换 (6分+2分, 共8分):
  - 1) 创建函数 my\_similarity(im, dx, dy, theta, s), 实现将图像沿 x 轴平移 dx 像素, 沿 y 轴平移 dy 像素 (注意区分 x、y 和行列), 逆时针旋转 theta 度, 并缩放 s 倍。要求首先计算变换矩阵, 然后通过矩阵乘法找到每个像素变换后的坐标 (注意使用齐次坐标), 最后使用 cv2.remap 函数在整数像素网络上插值, 获得变换后的图像。注意使用**逆卷绕 (inverse warping)**;
  - 2) 在“a1\_script.py”中依次调用上述 my\_similarity 函数, 适当设置输入参数使得能完整显示**全部像素** (即变换后所有像素坐标为正), 并显示结果图像。

# 本节主题：

生物视觉与色彩



# 本节主题：

生物视觉与色彩  
图像滤波

什么是图像？



# 图像即函数

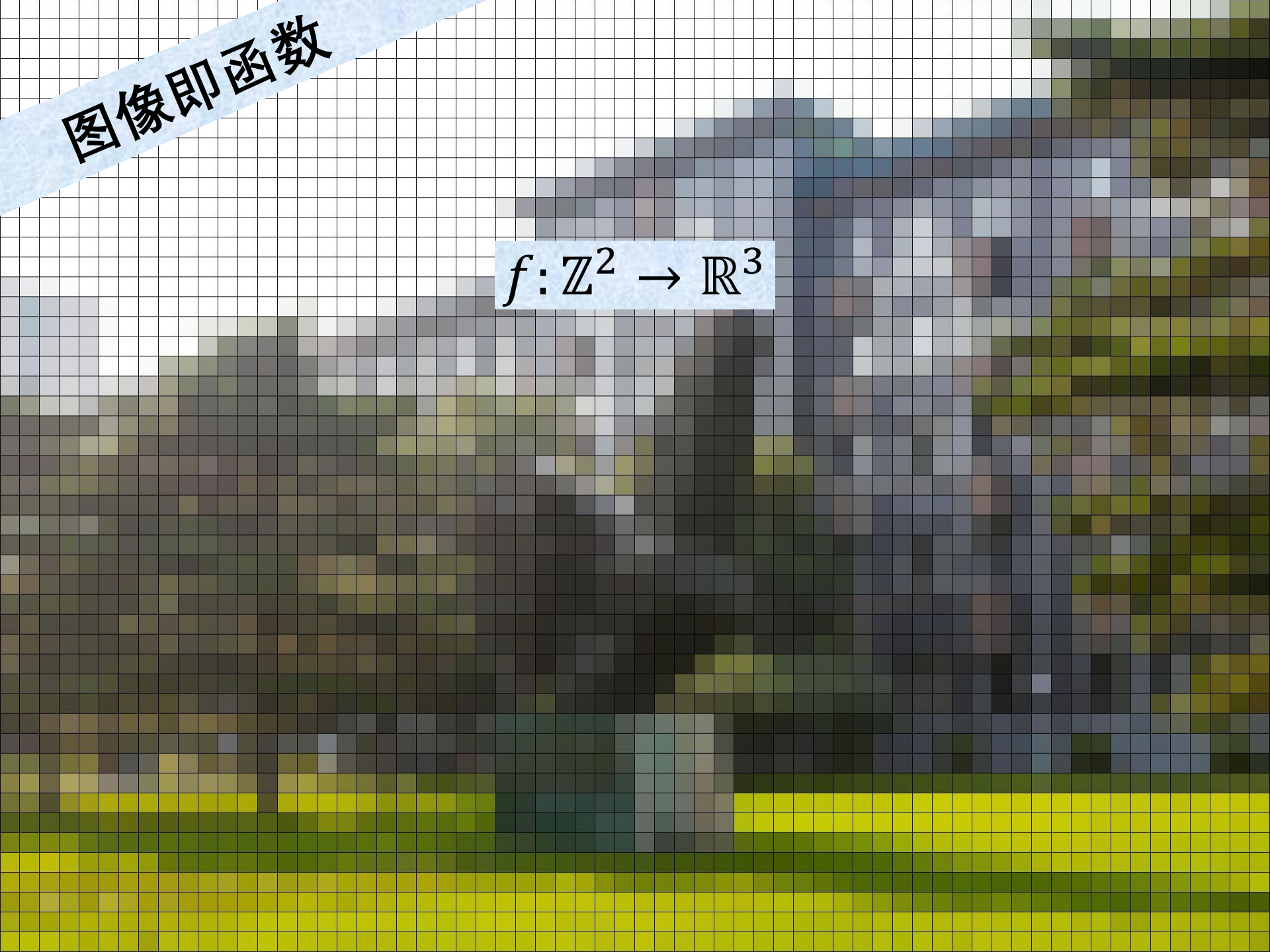


# 图像即函数

$$f: \mathbb{Z}^2 \rightarrow \mathbb{R}$$

# 图像即函数

$$f: \mathbb{Z}^2 \rightarrow \mathbb{R}^3$$

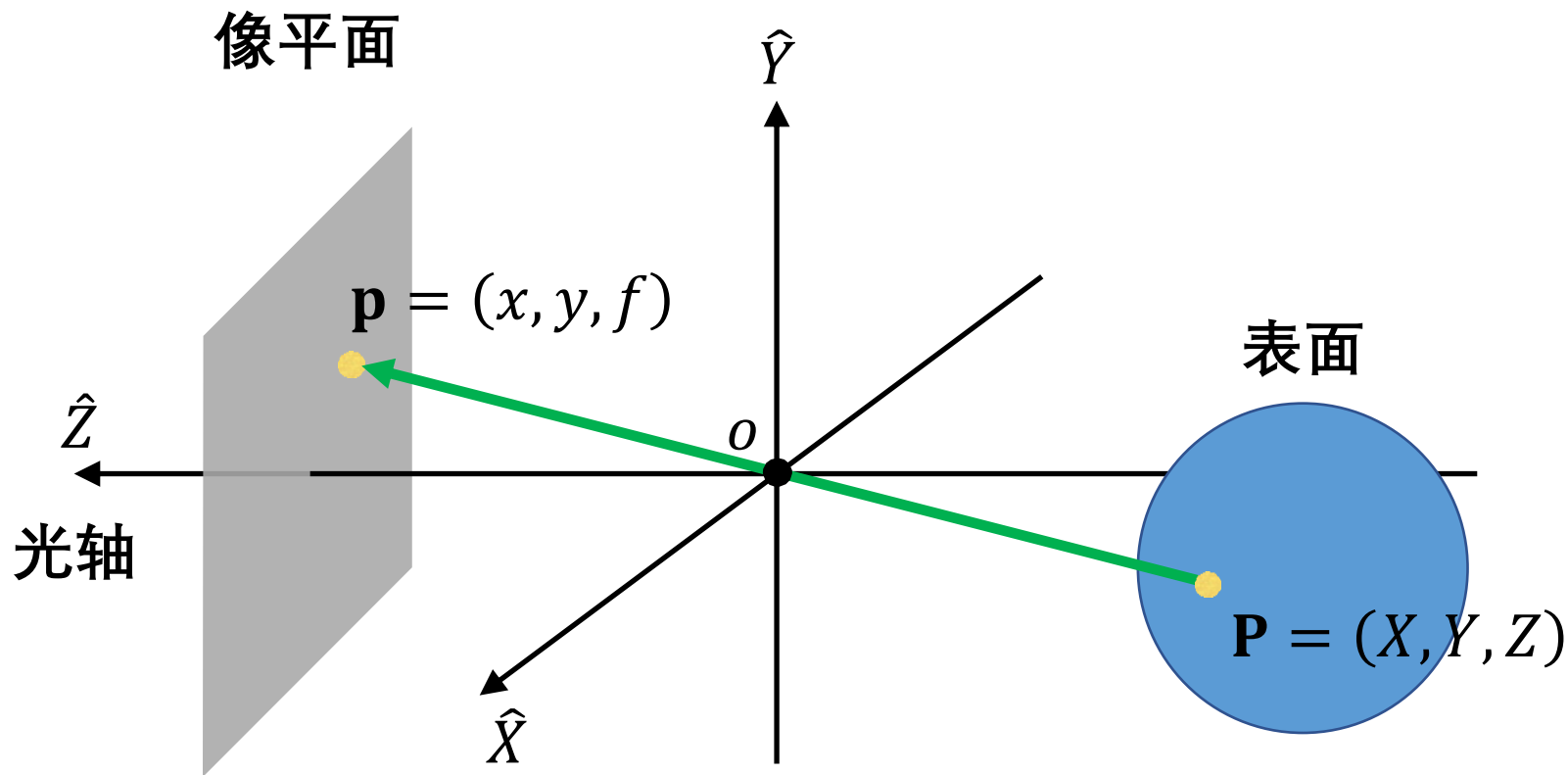




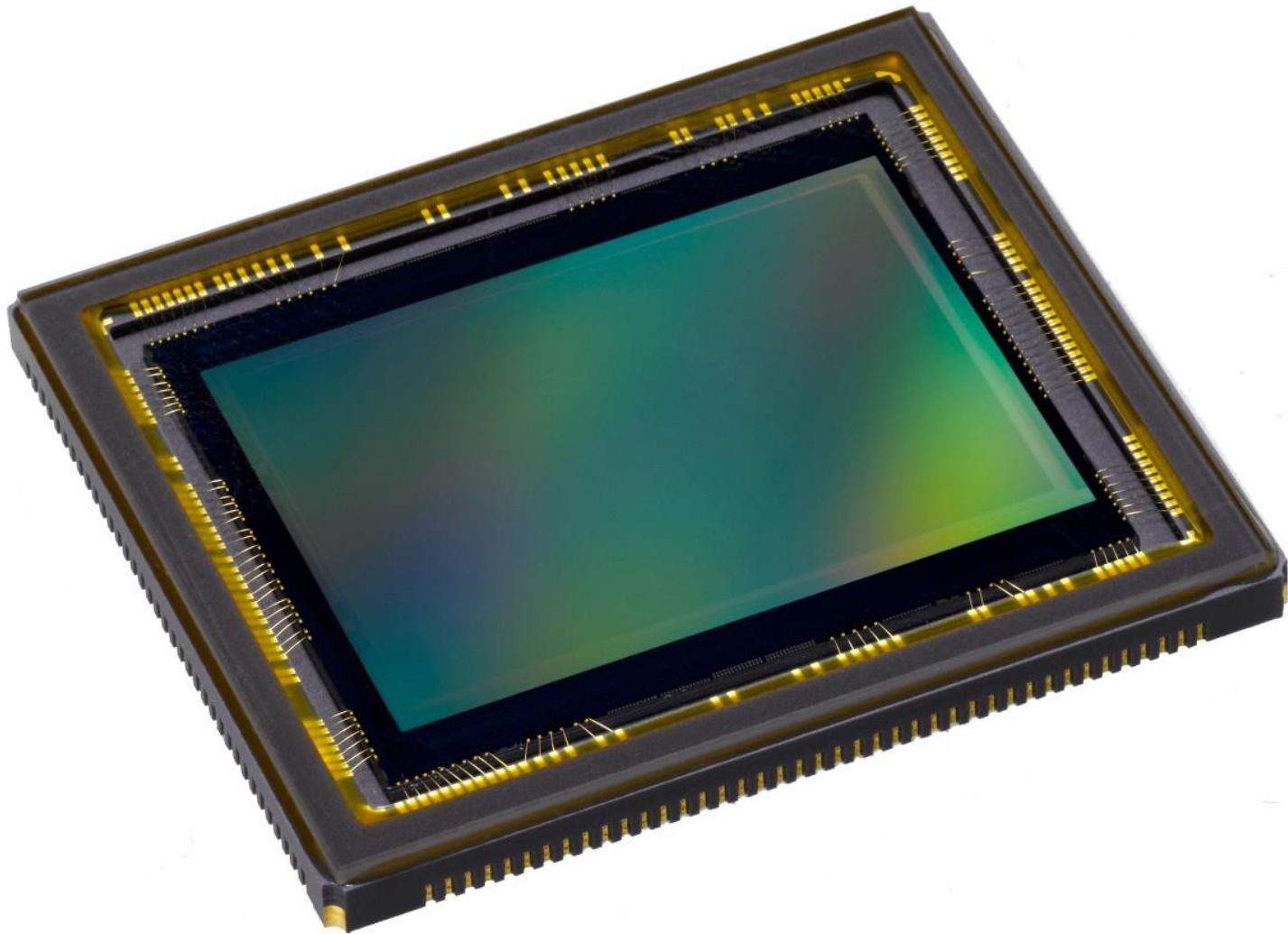
# 图像即函数

$$f: \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

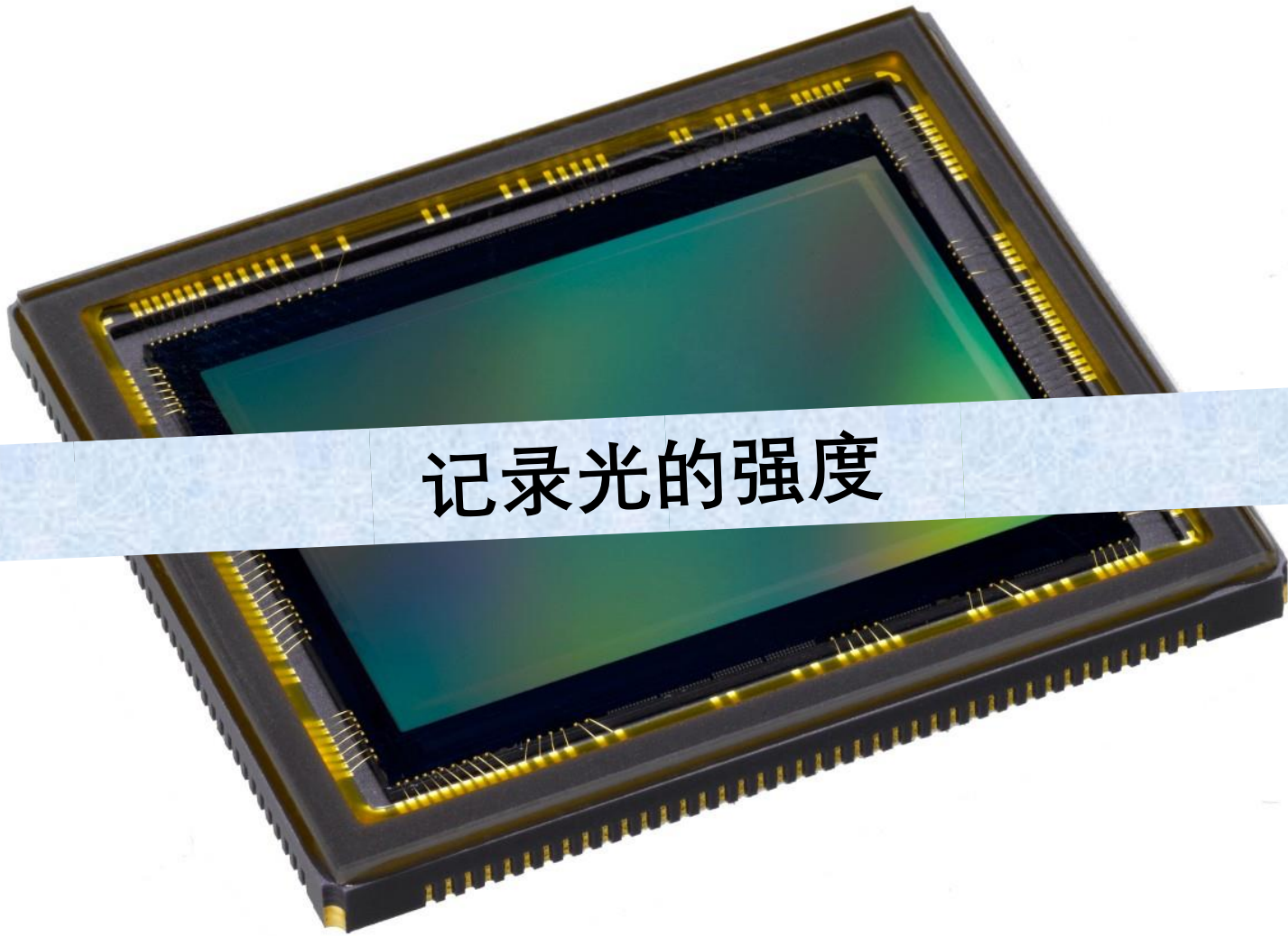






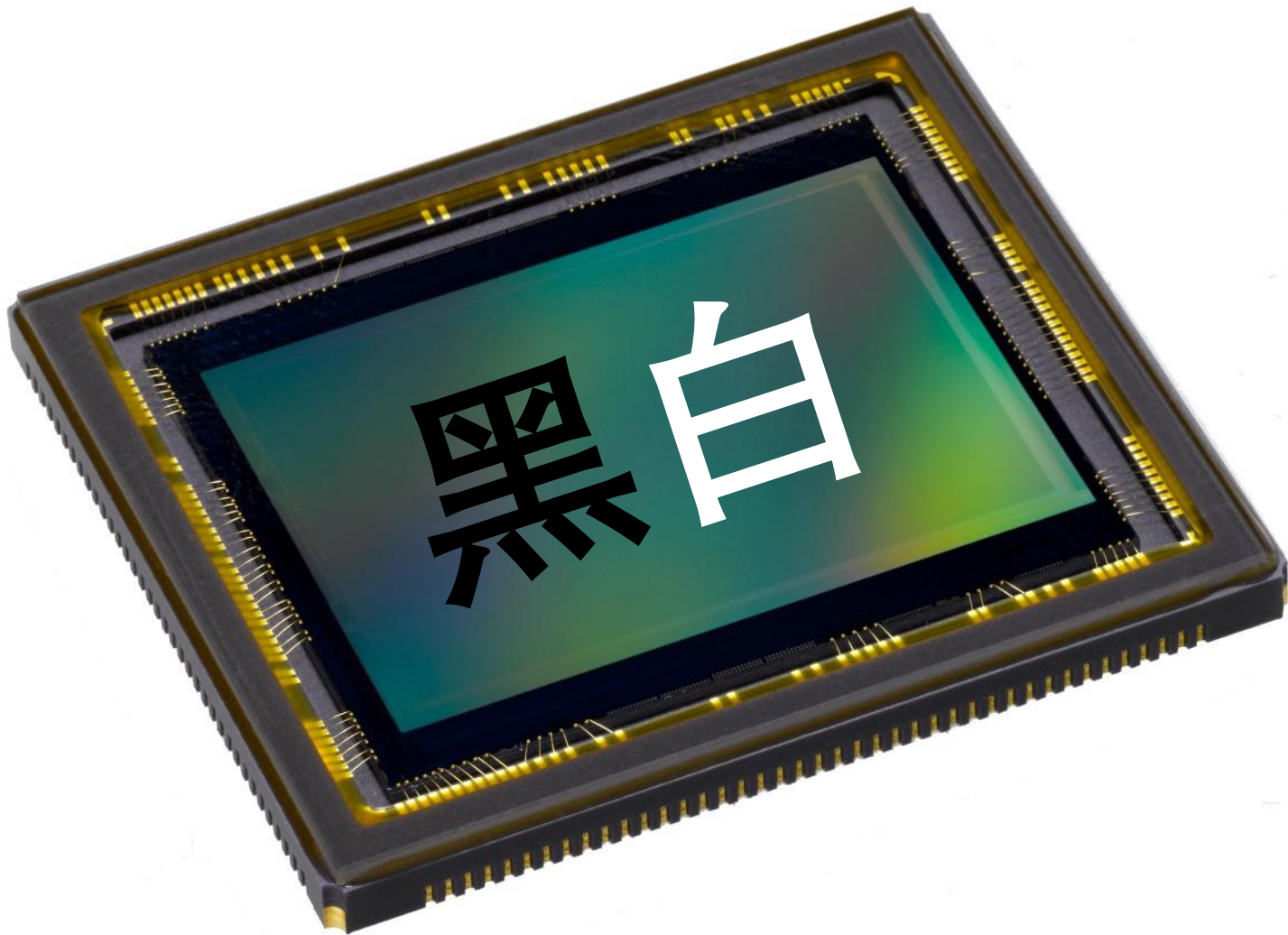


**电荷耦合器件 (CCD)**

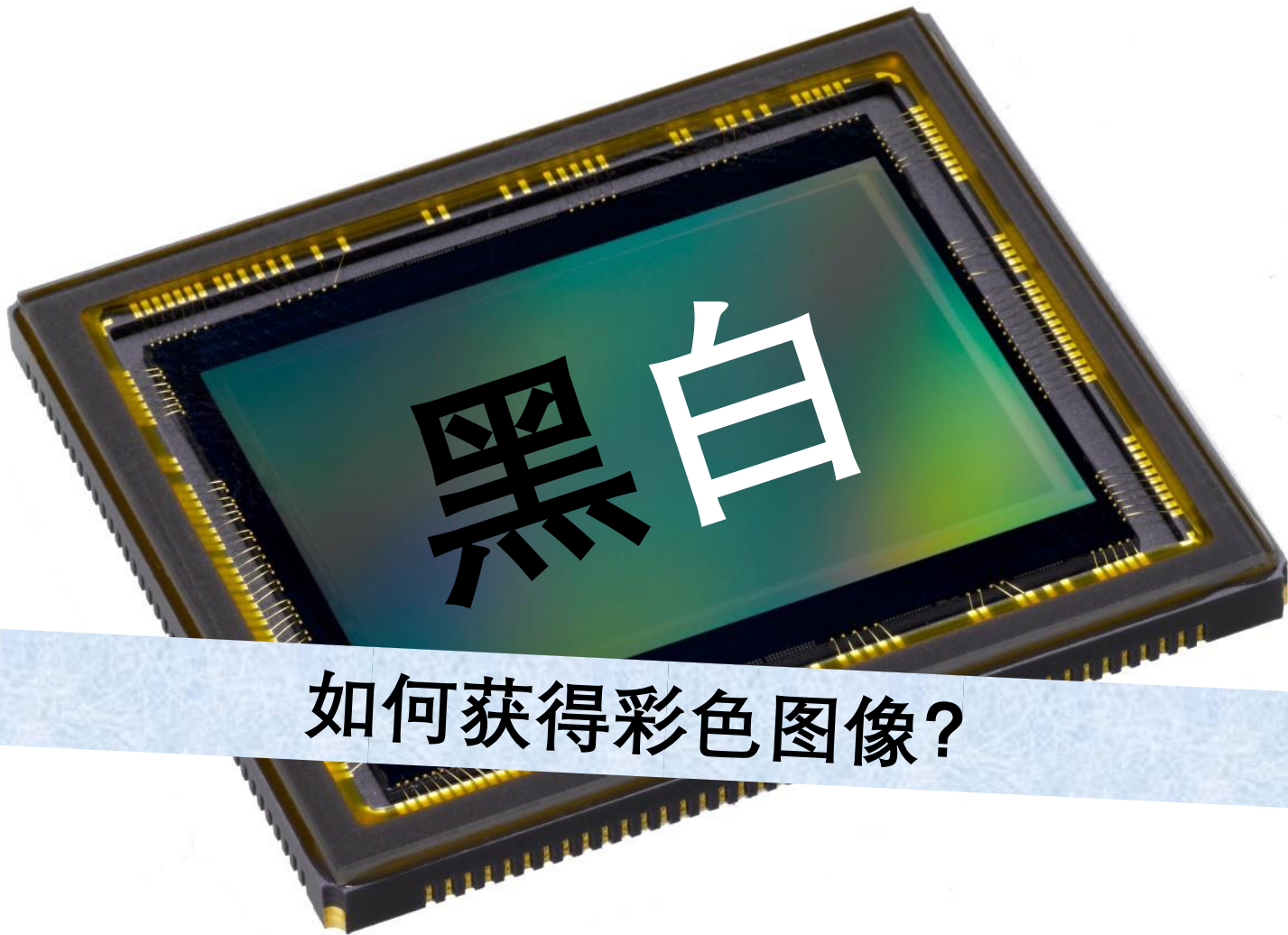


记录光的强度

电荷耦合器件 (CCD)



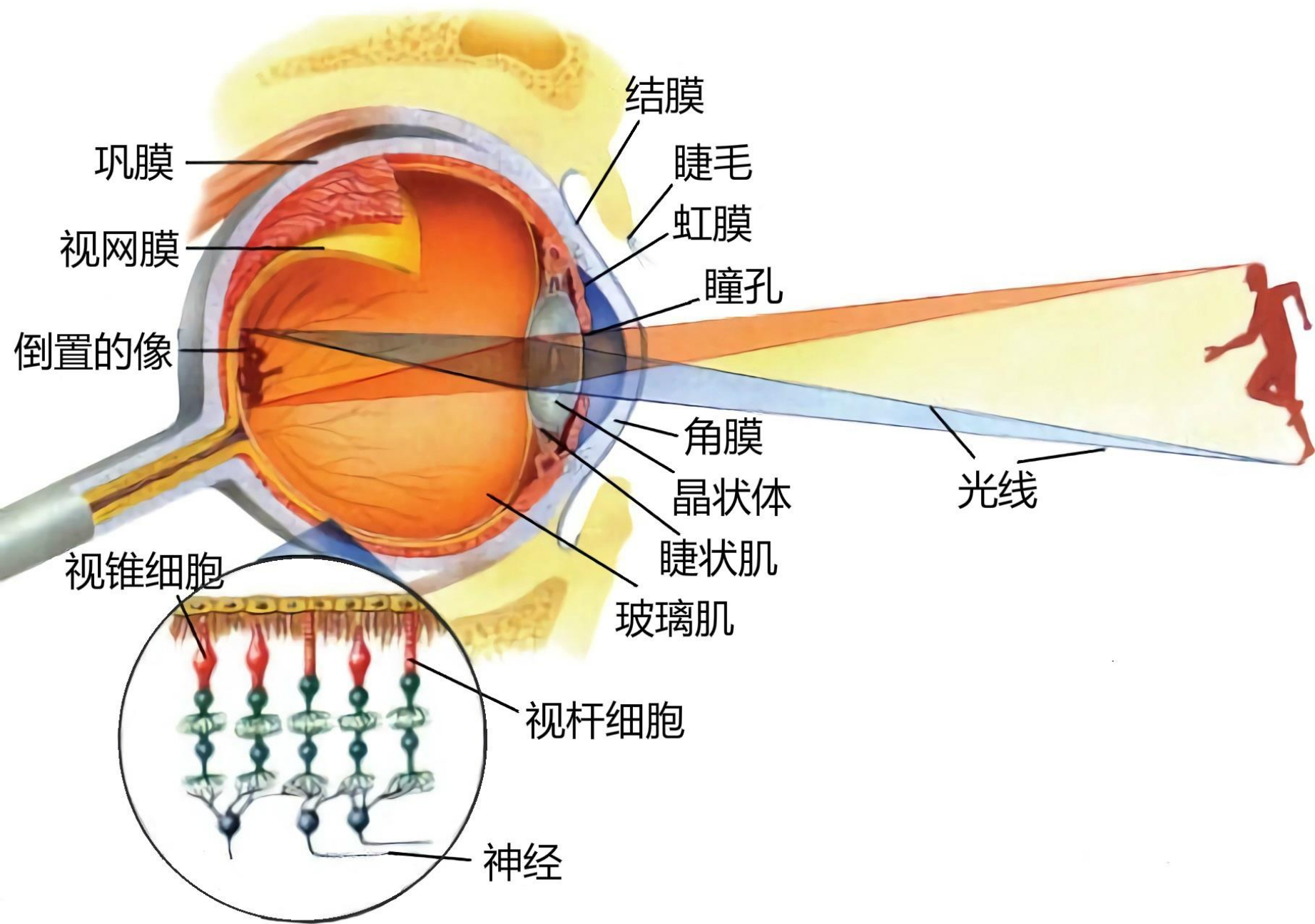
电荷耦合器件 (CCD)

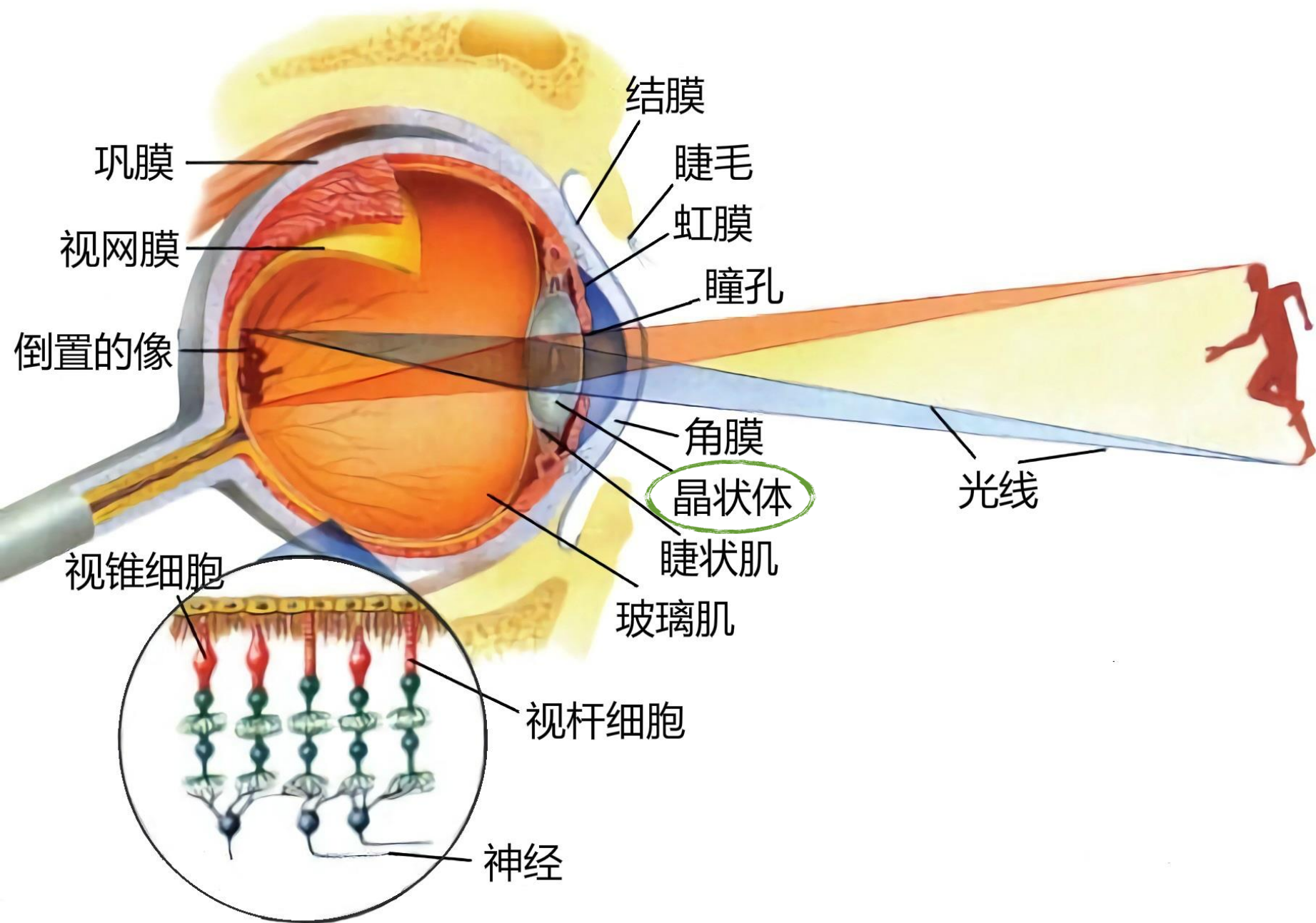


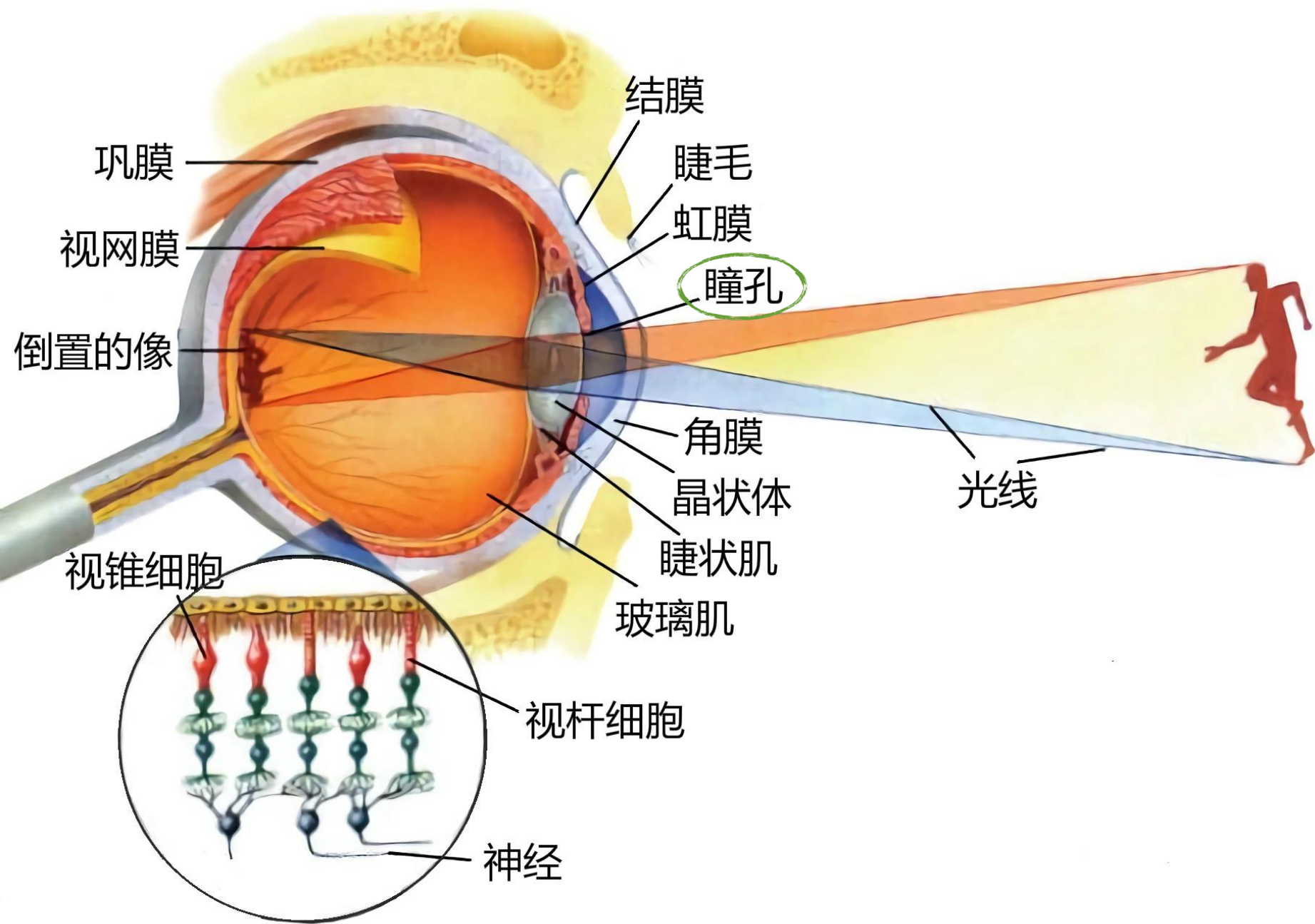
如何获得彩色图像？

电荷耦合器件 (CCD)

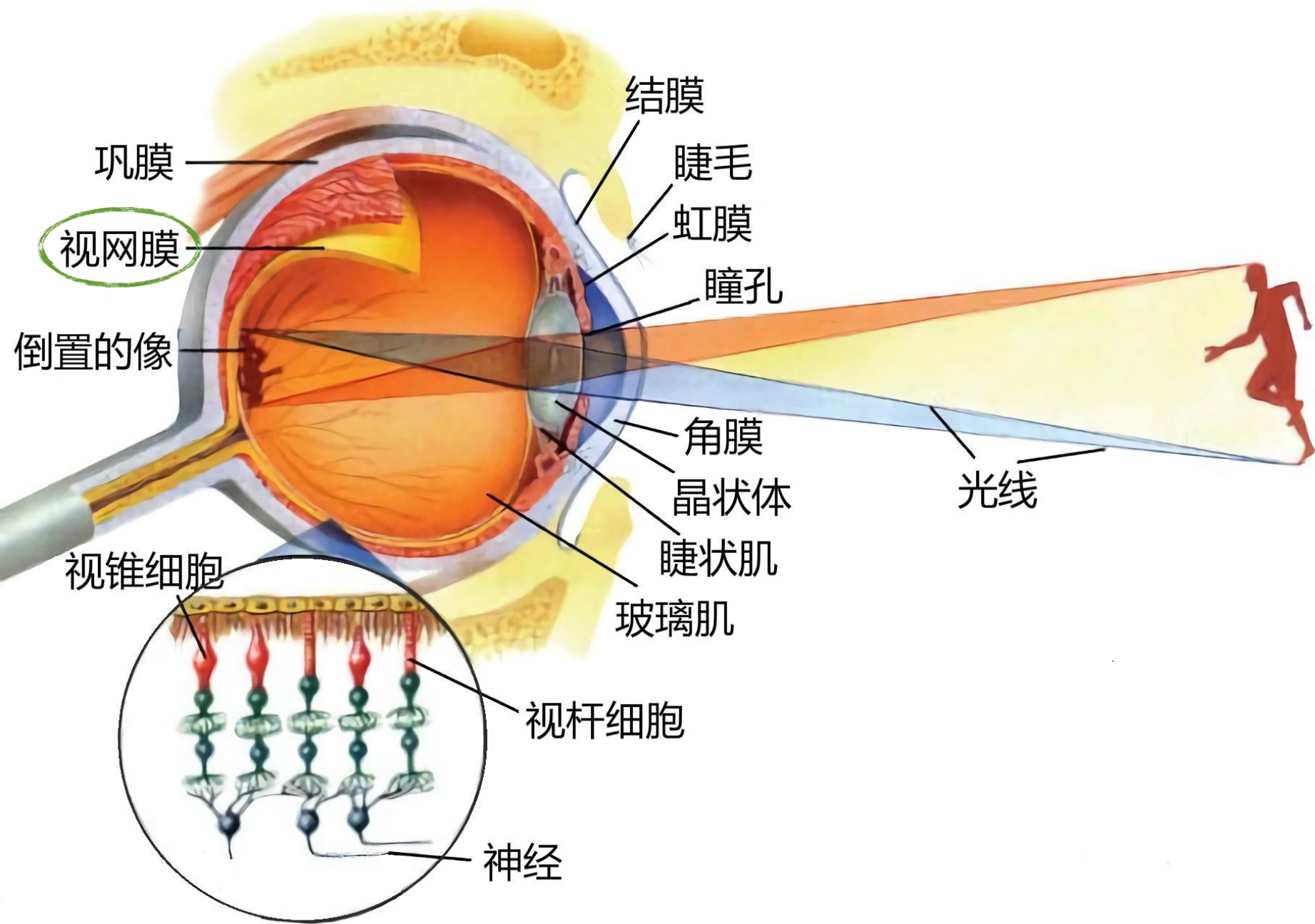




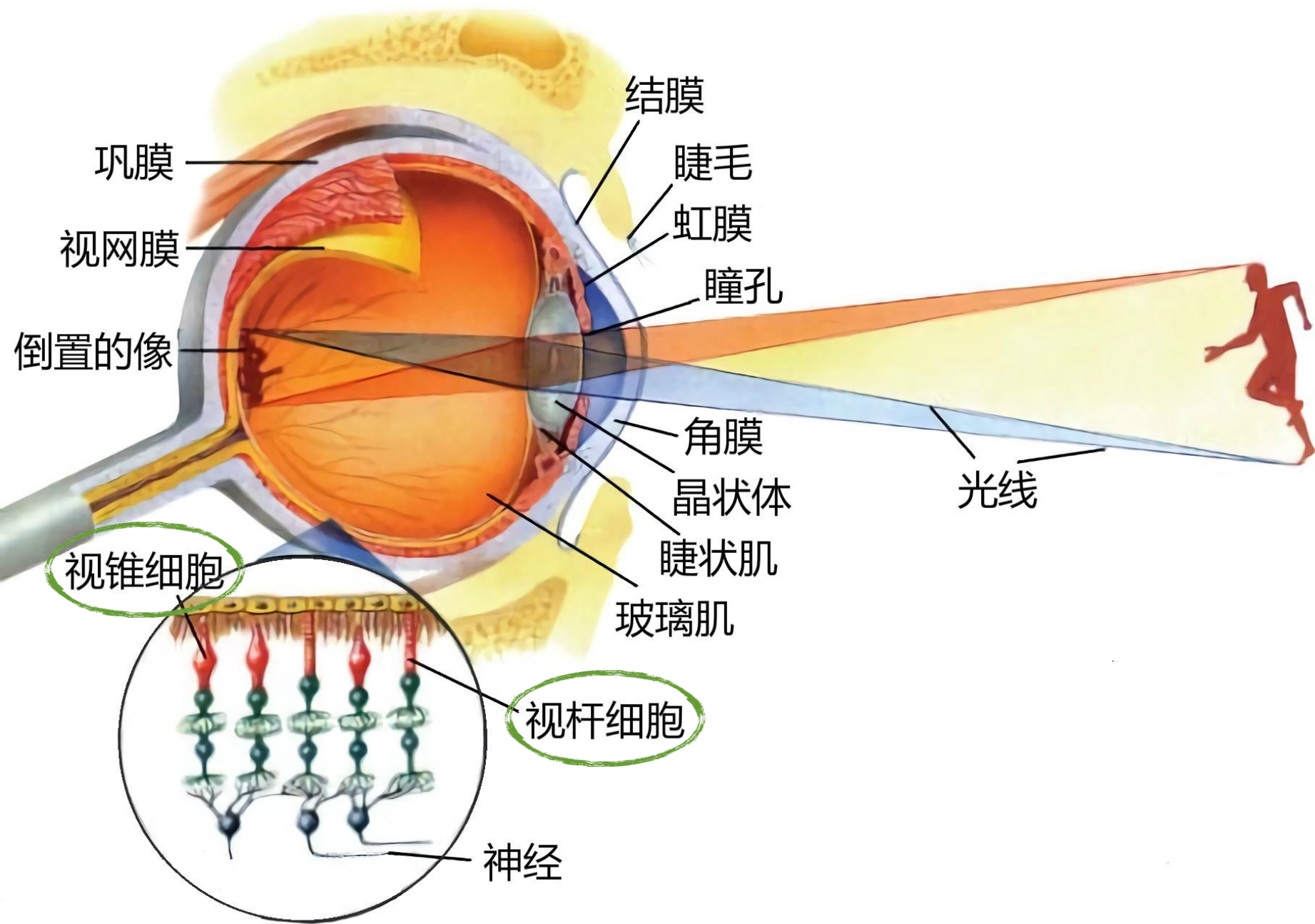


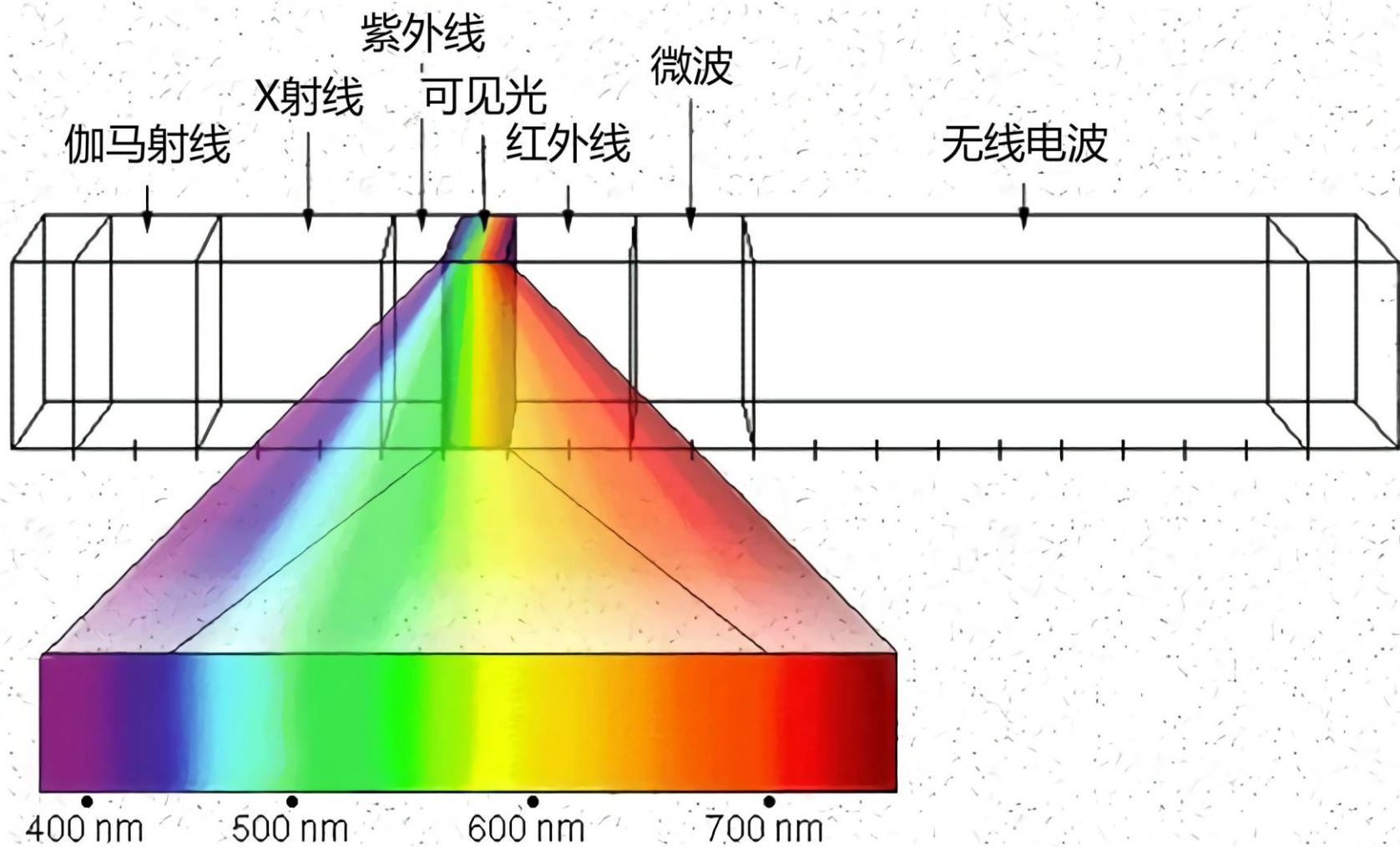


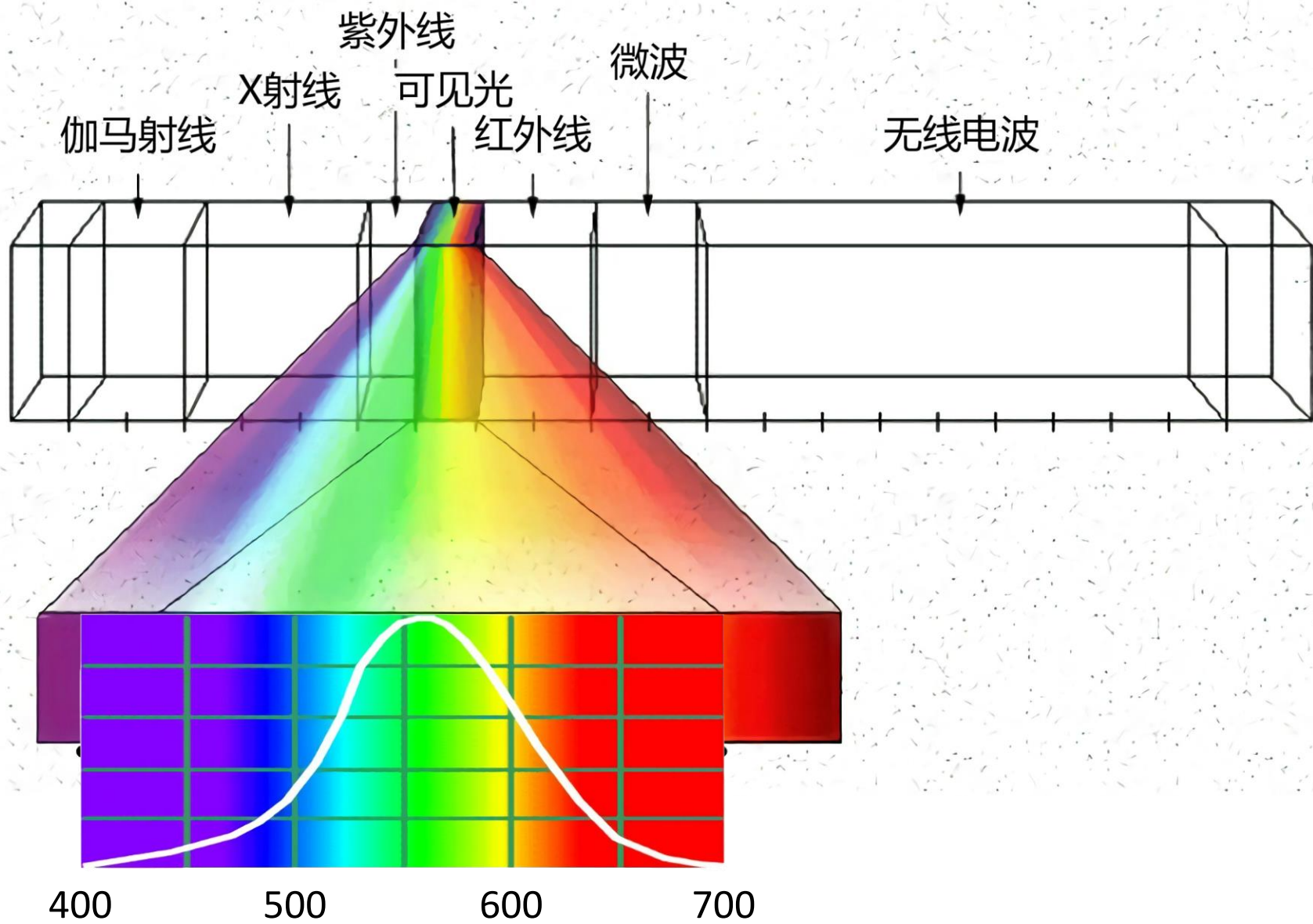




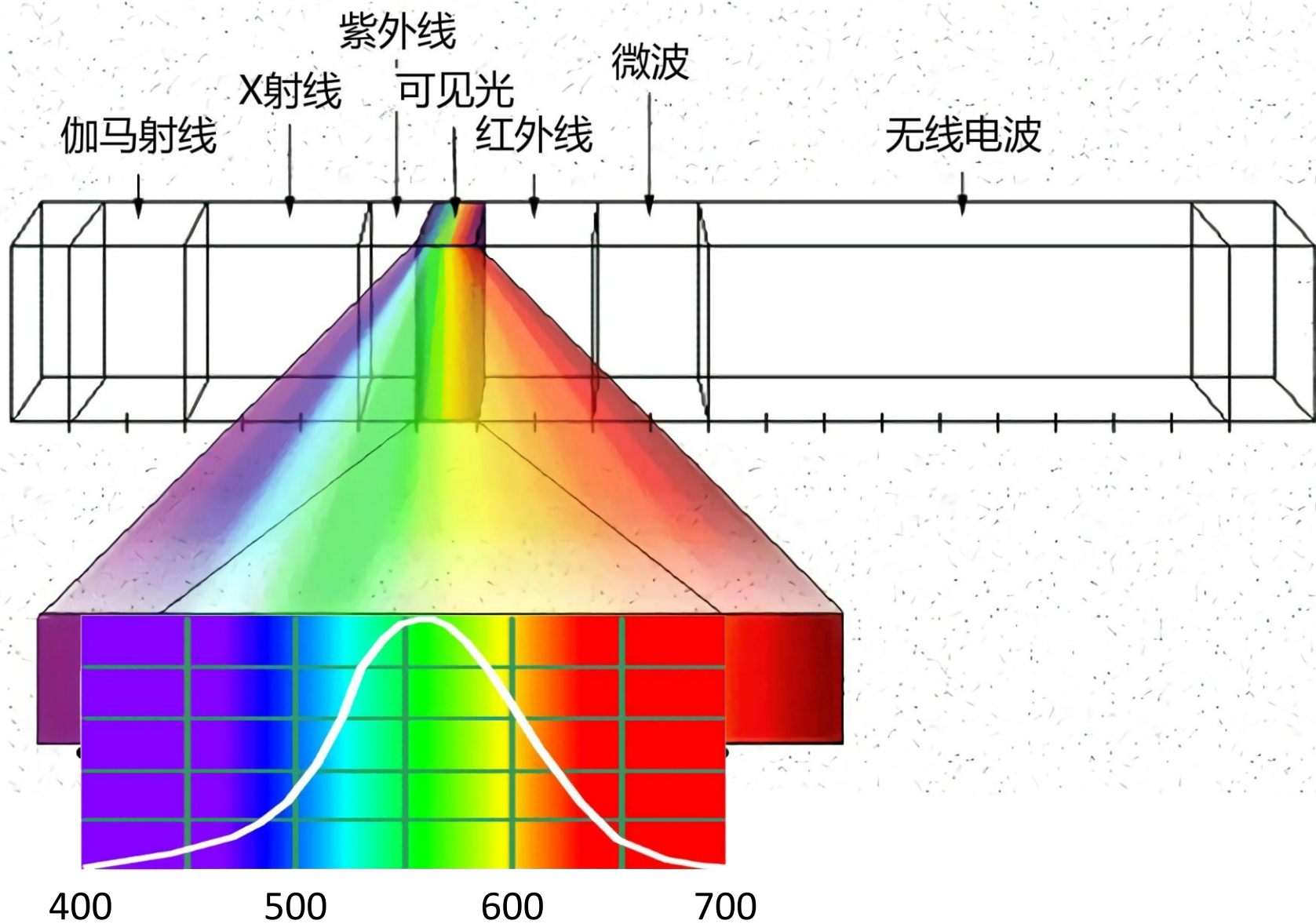




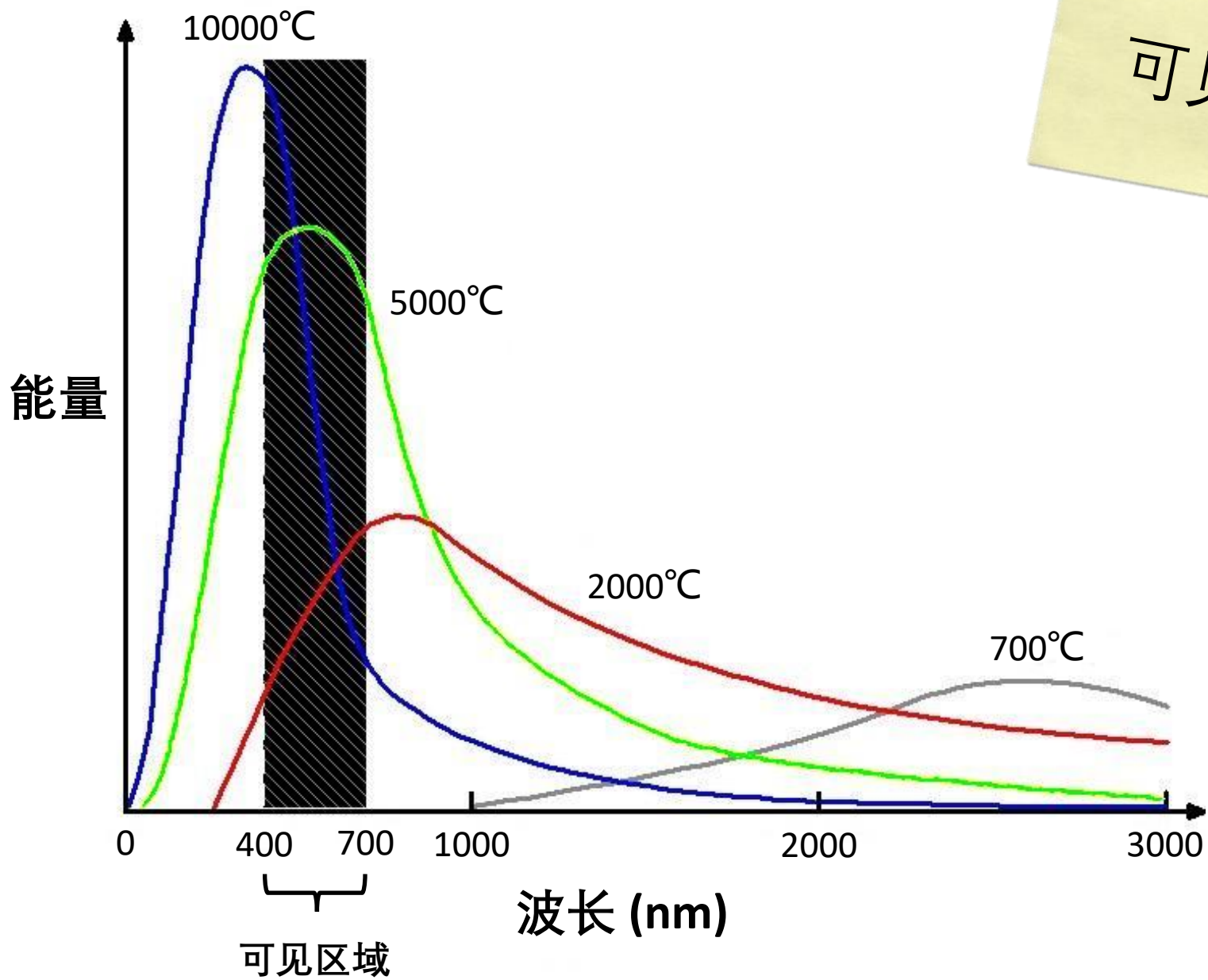


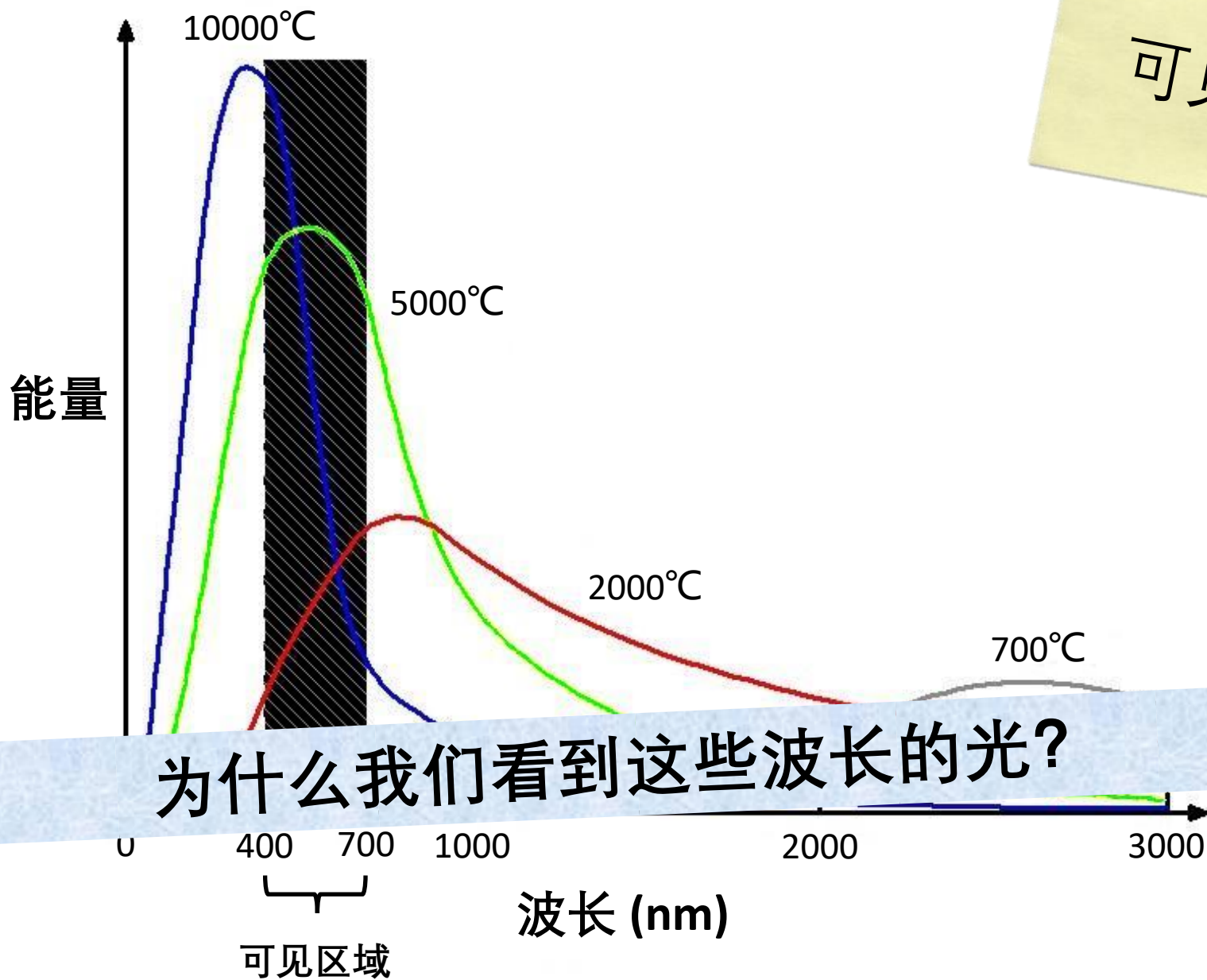




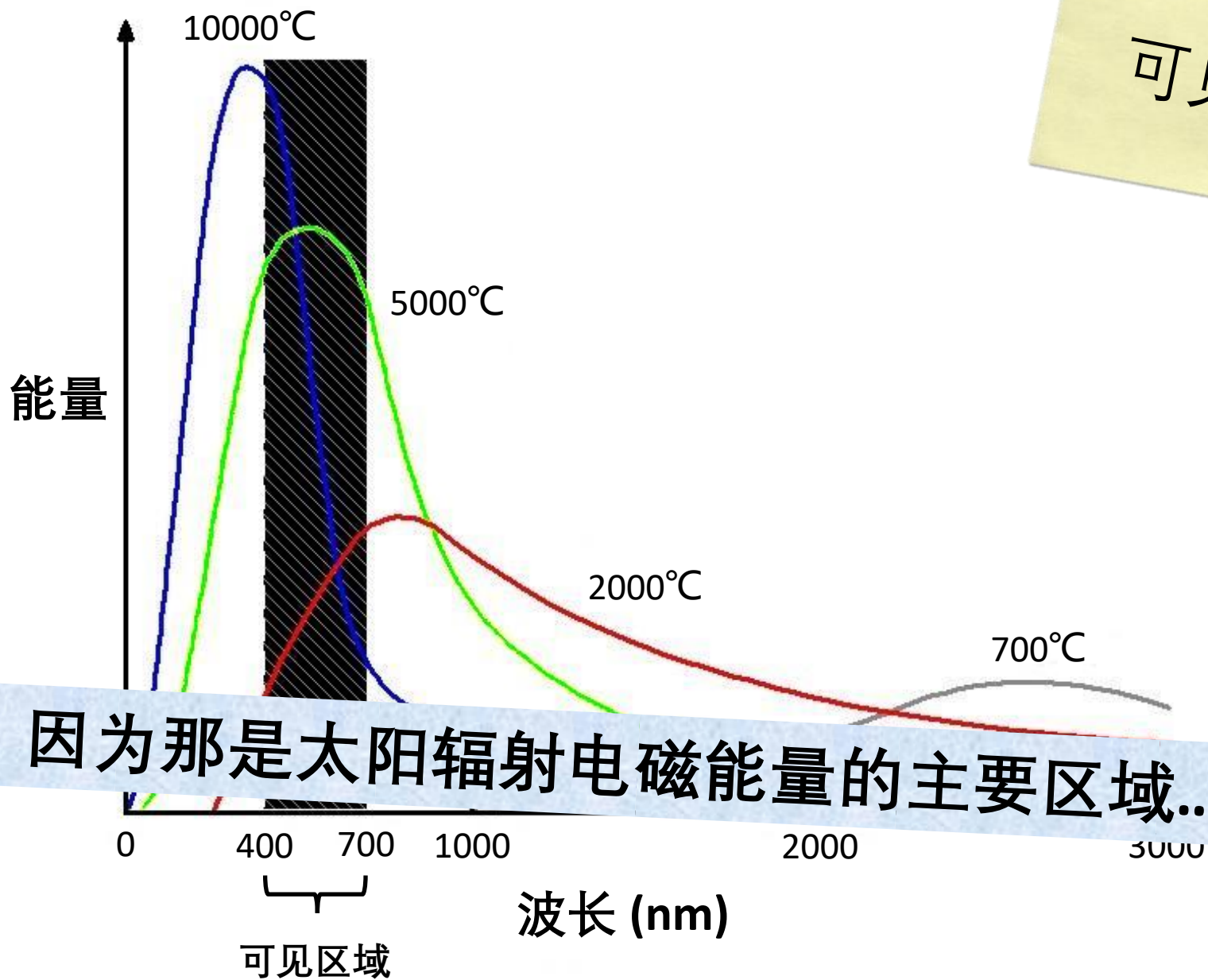


人眼光度敏感函数



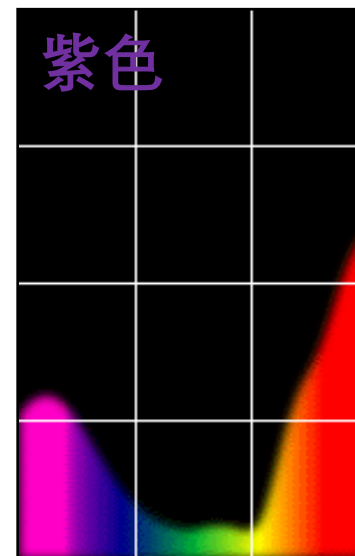
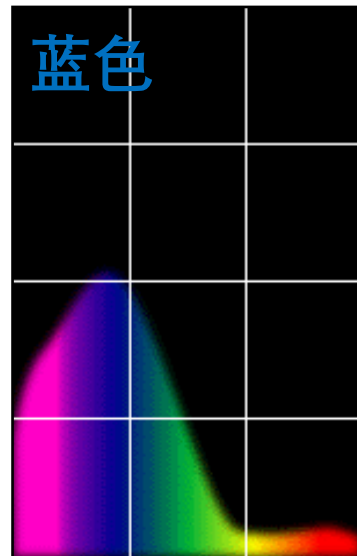
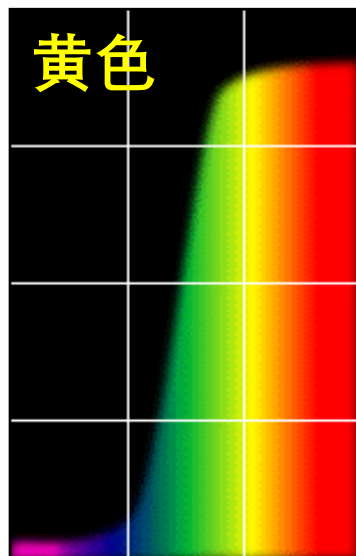
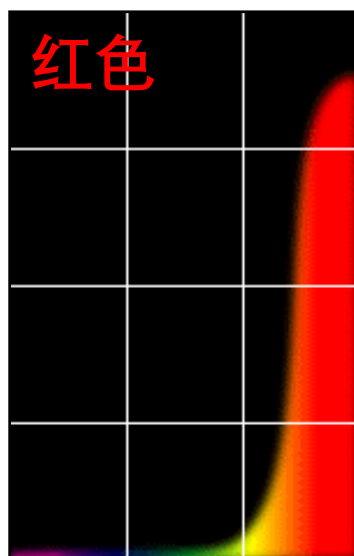


为什么我们看到这些波长的光？





反射光子



400

700

400

700

400

700

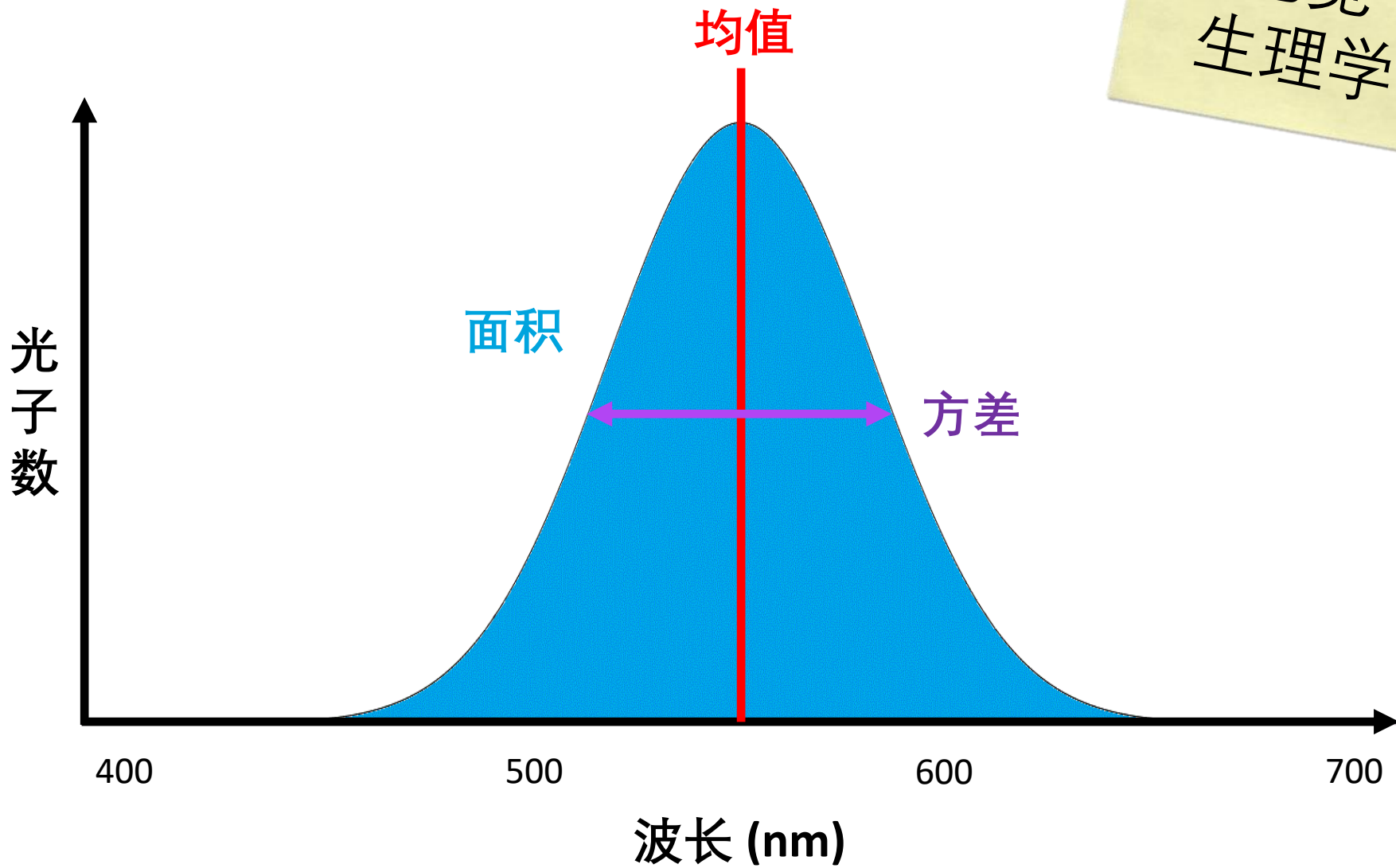
400

700

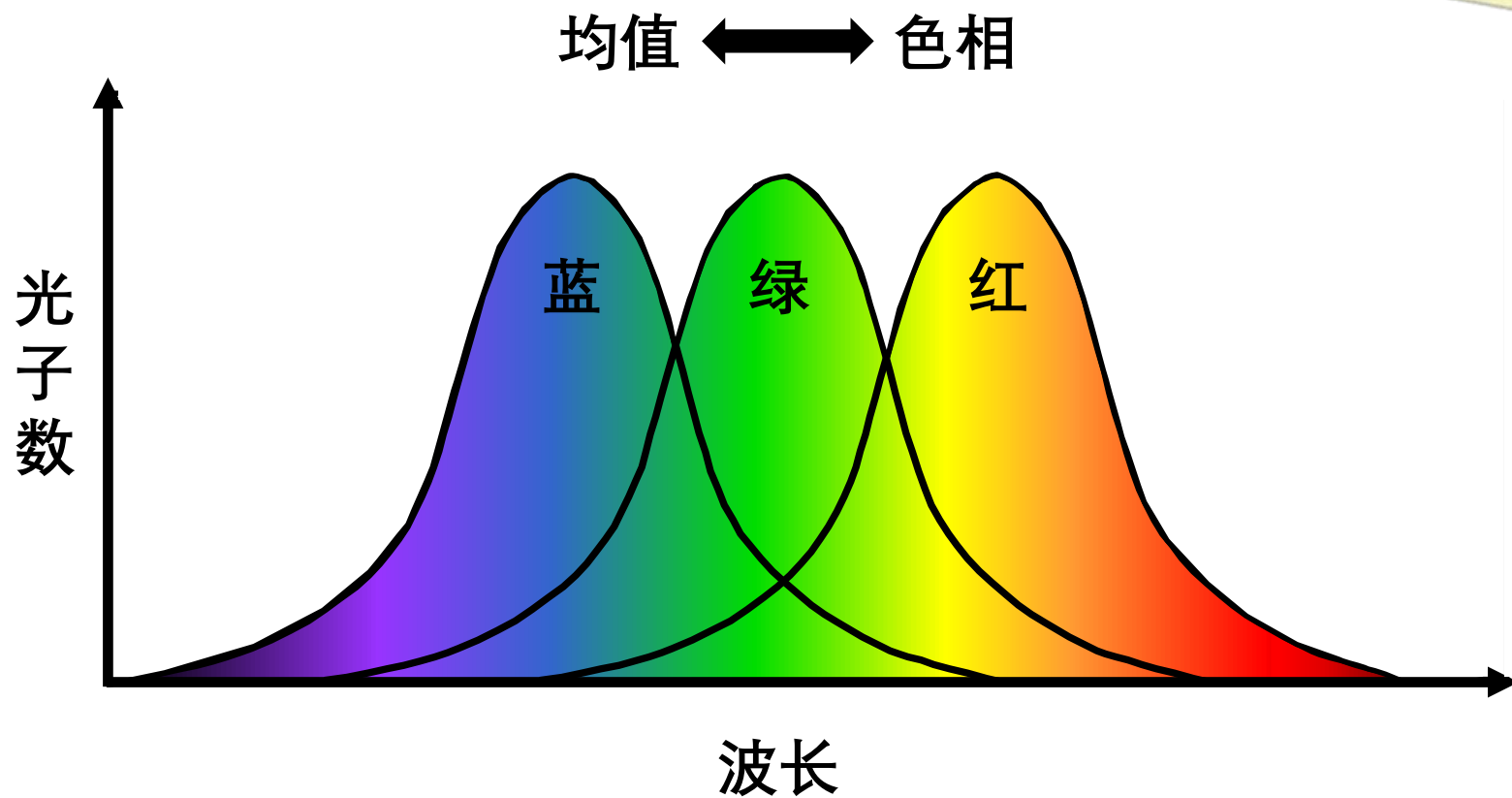
波长 (nm)



色觉  
生理学

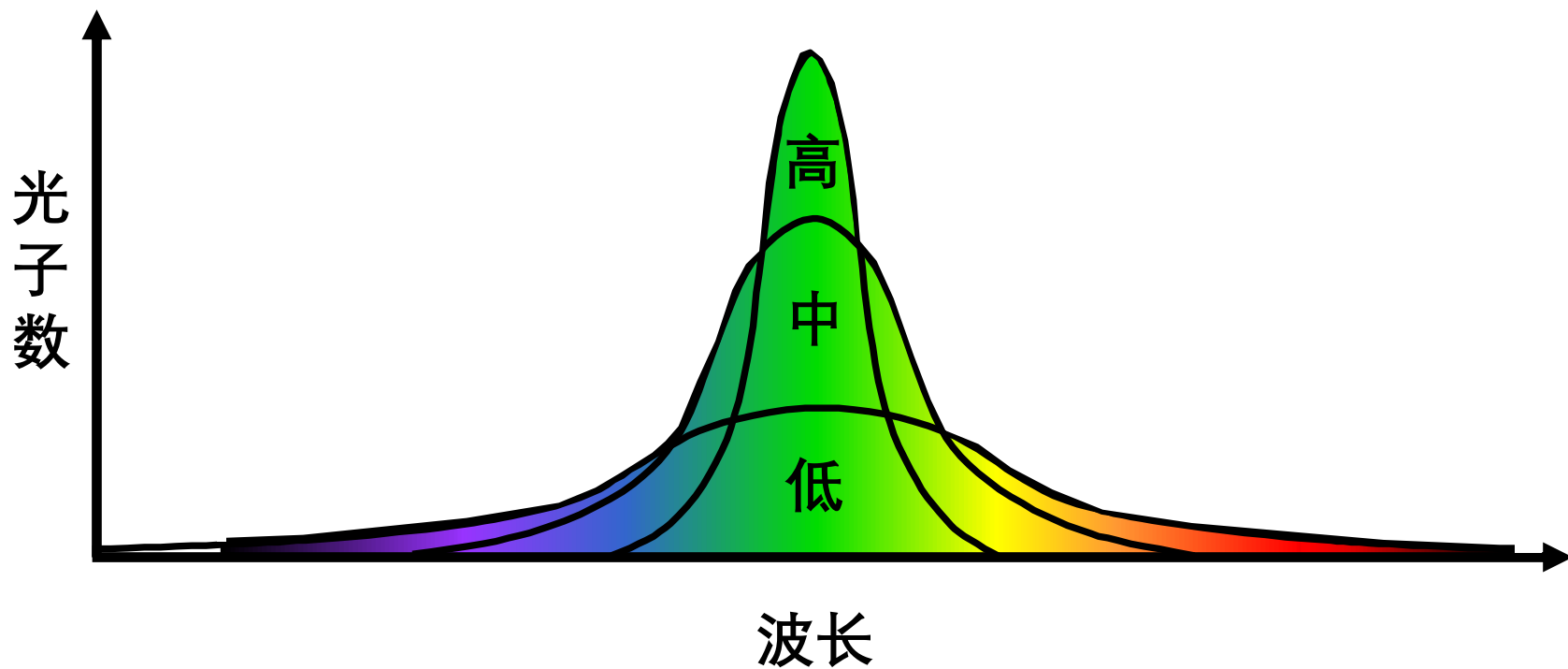


色觉  
生理学

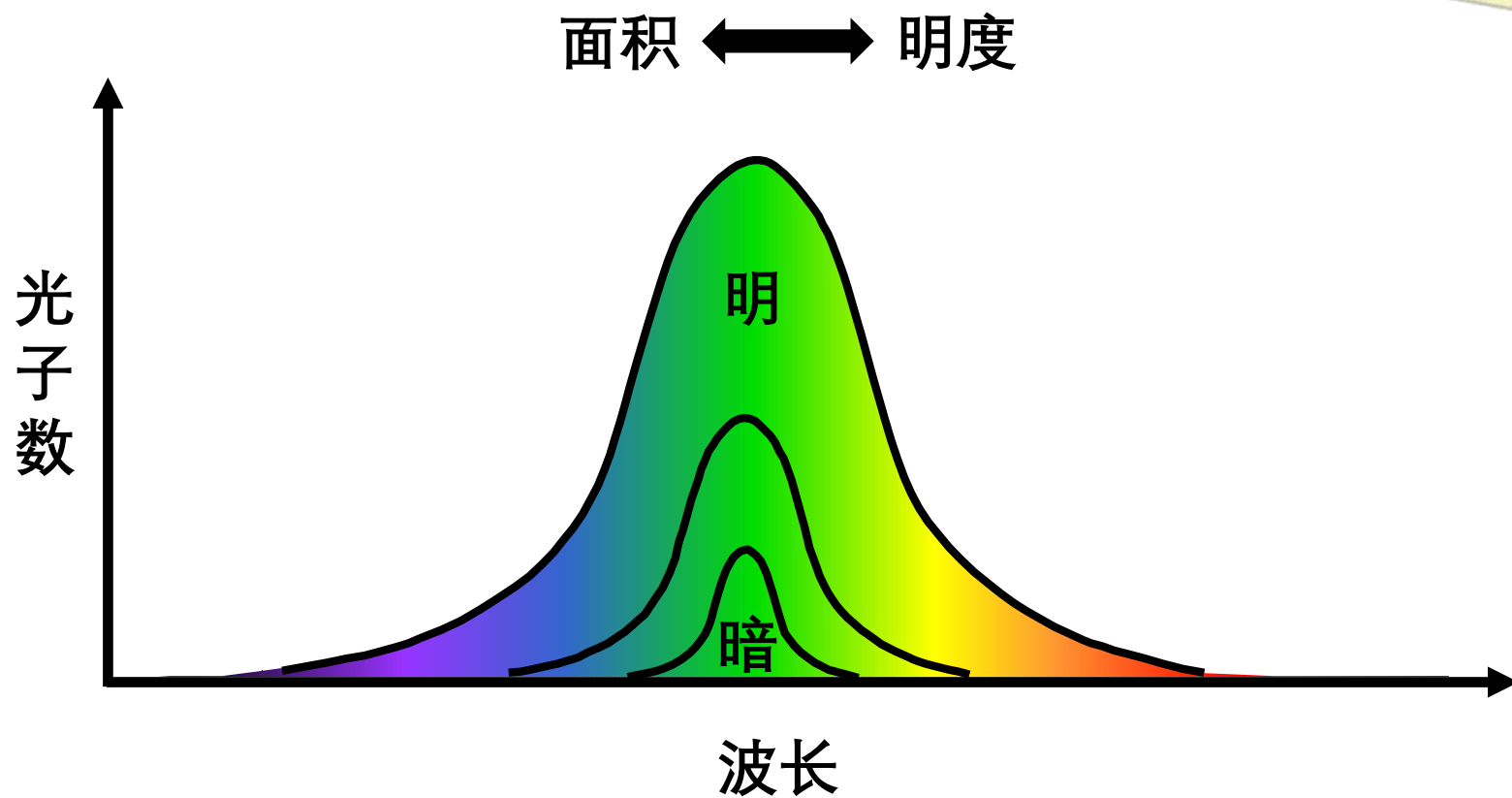


色觉  
生理学

方差  $\longleftrightarrow$  饱和度

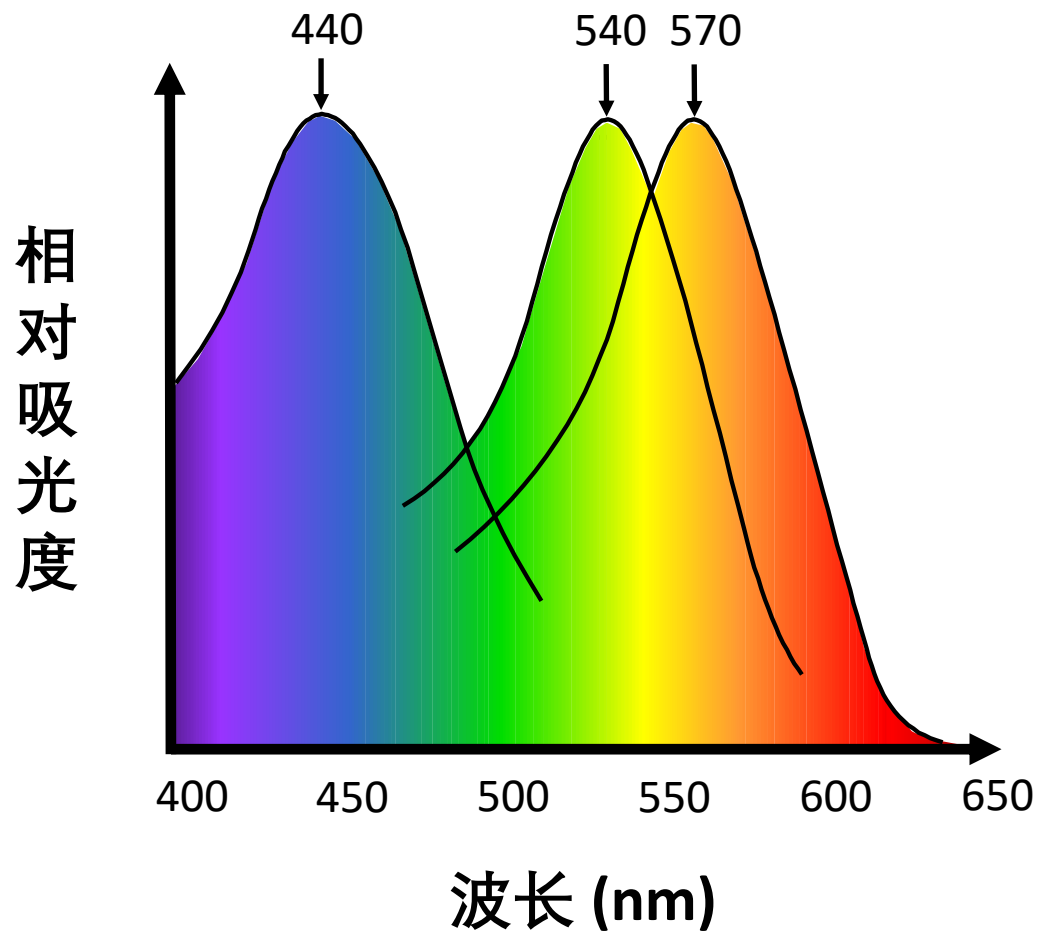


色觉  
生理学

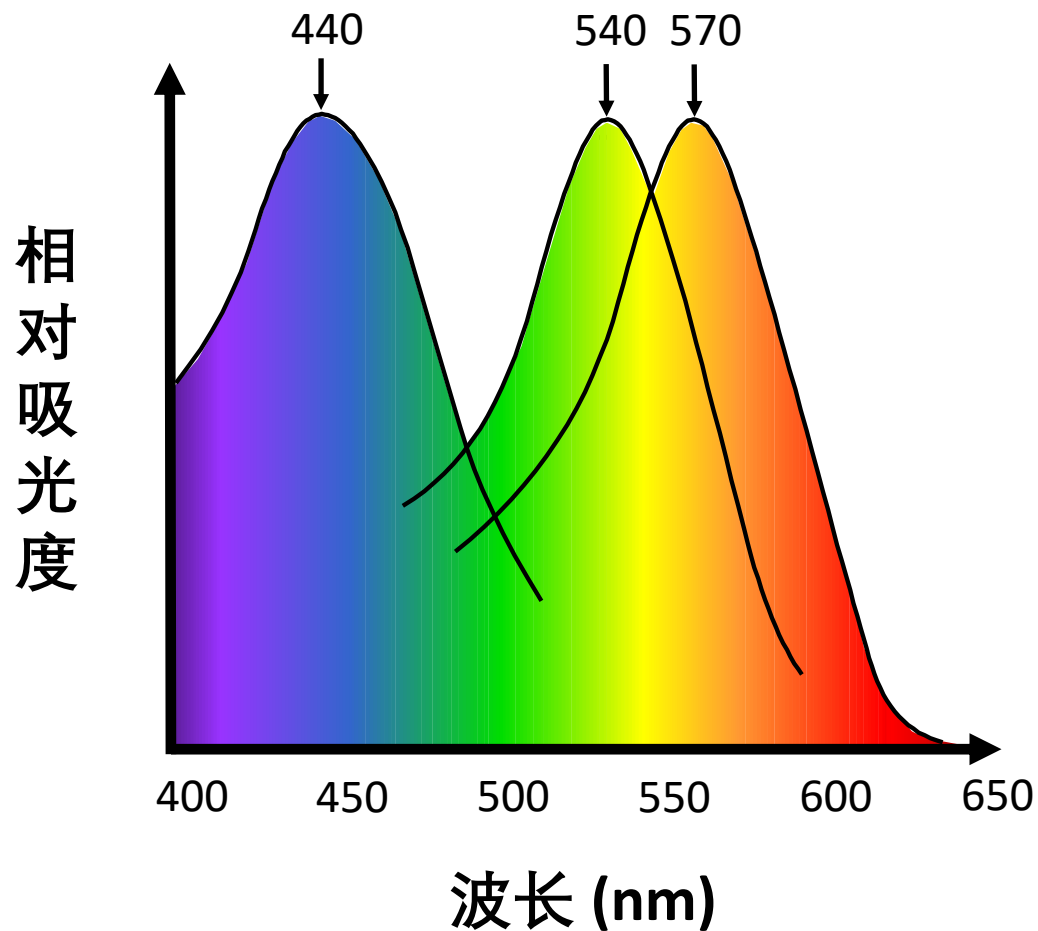


色觉  
生理学

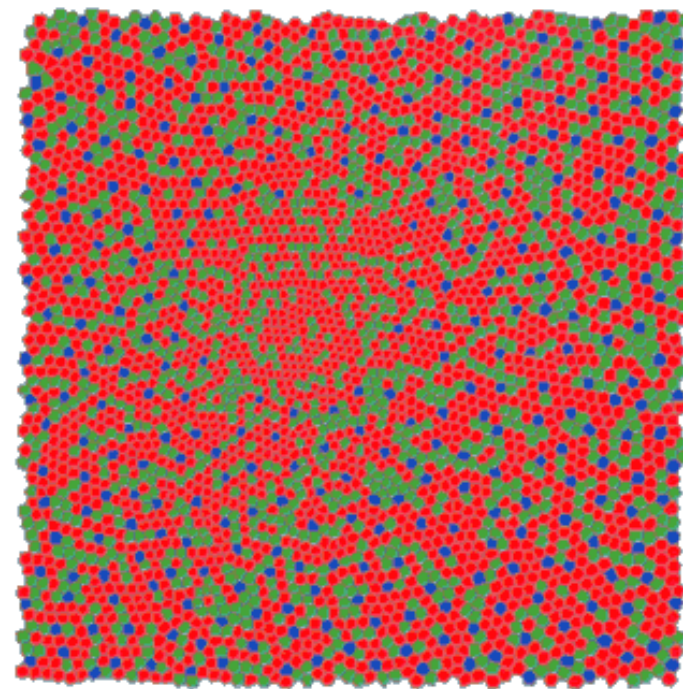
### 三种视锥细胞



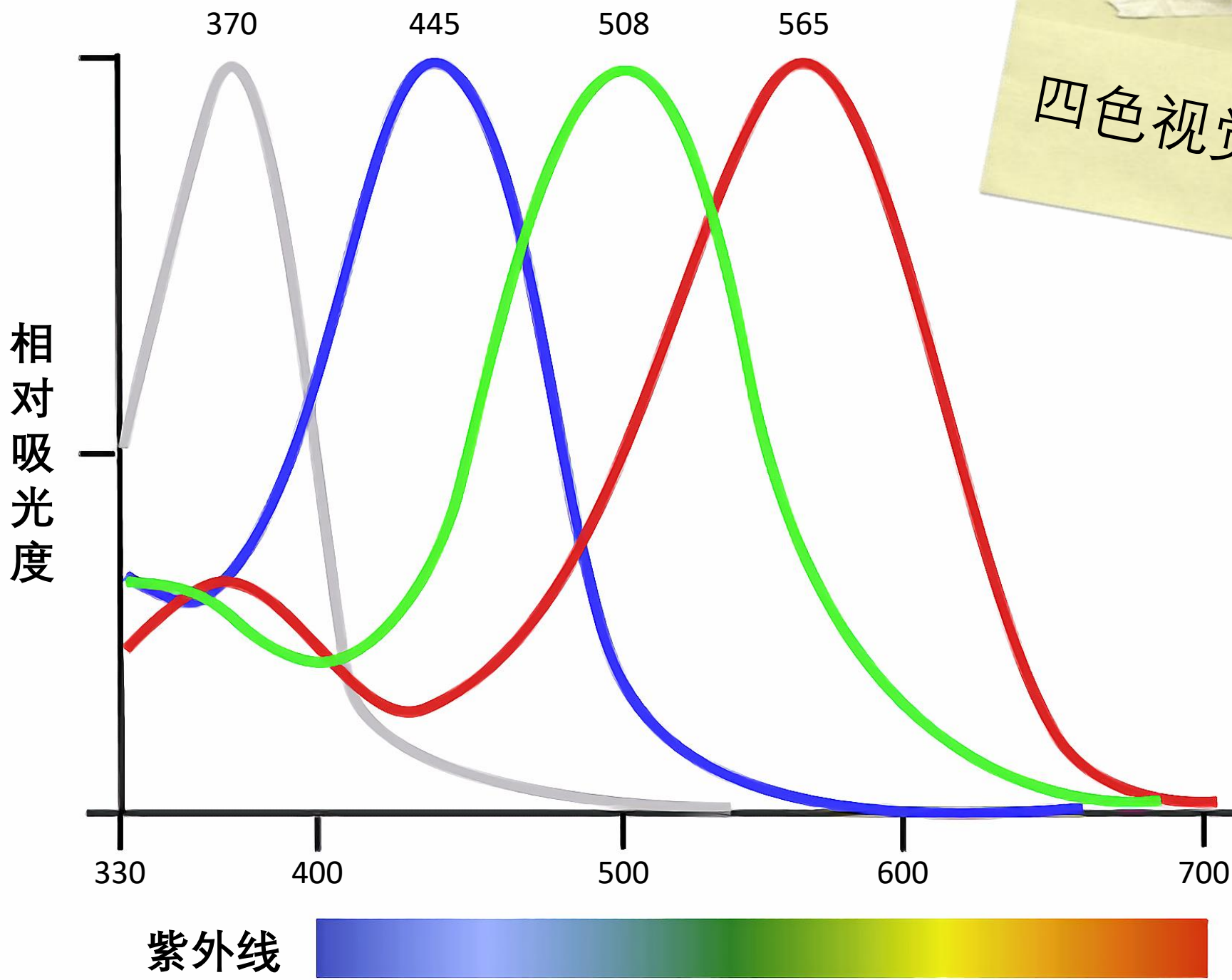
### 三种视锥细胞



### 视锥细胞镶嵌

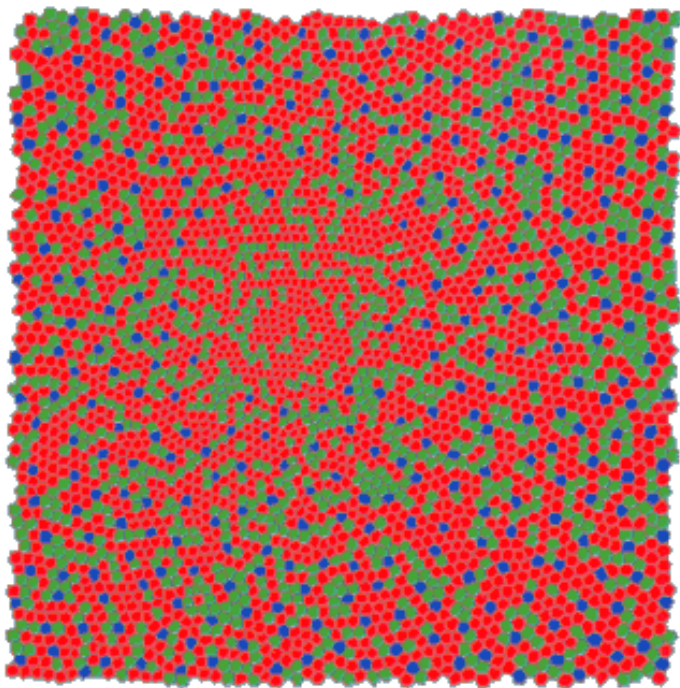


四色视觉



相机色彩  
感知

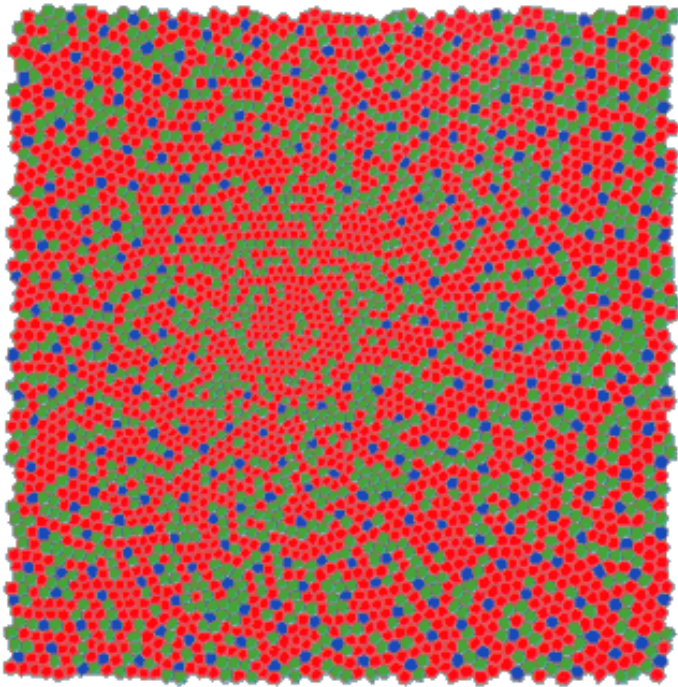
## 视锥细胞镶嵌



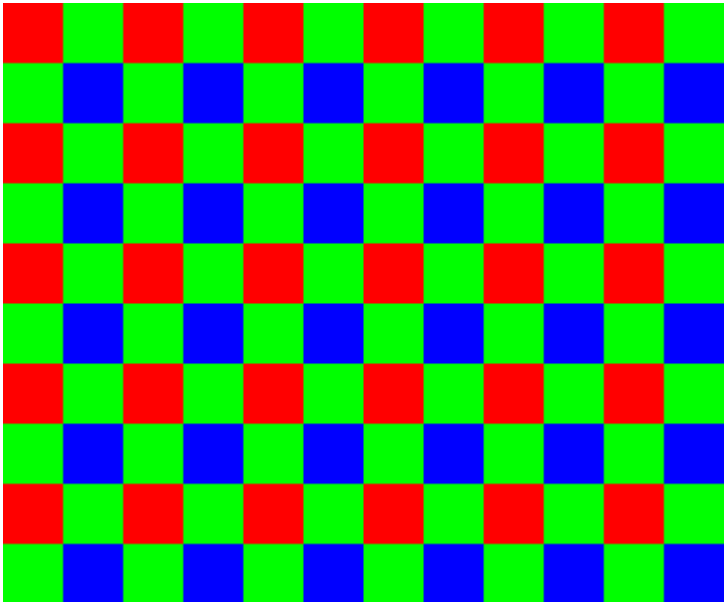


相机色彩感知

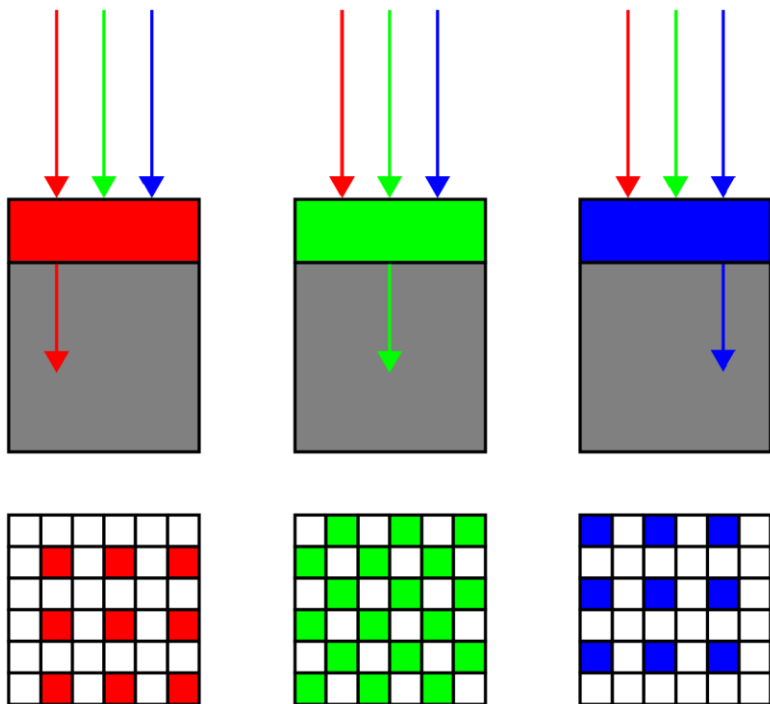
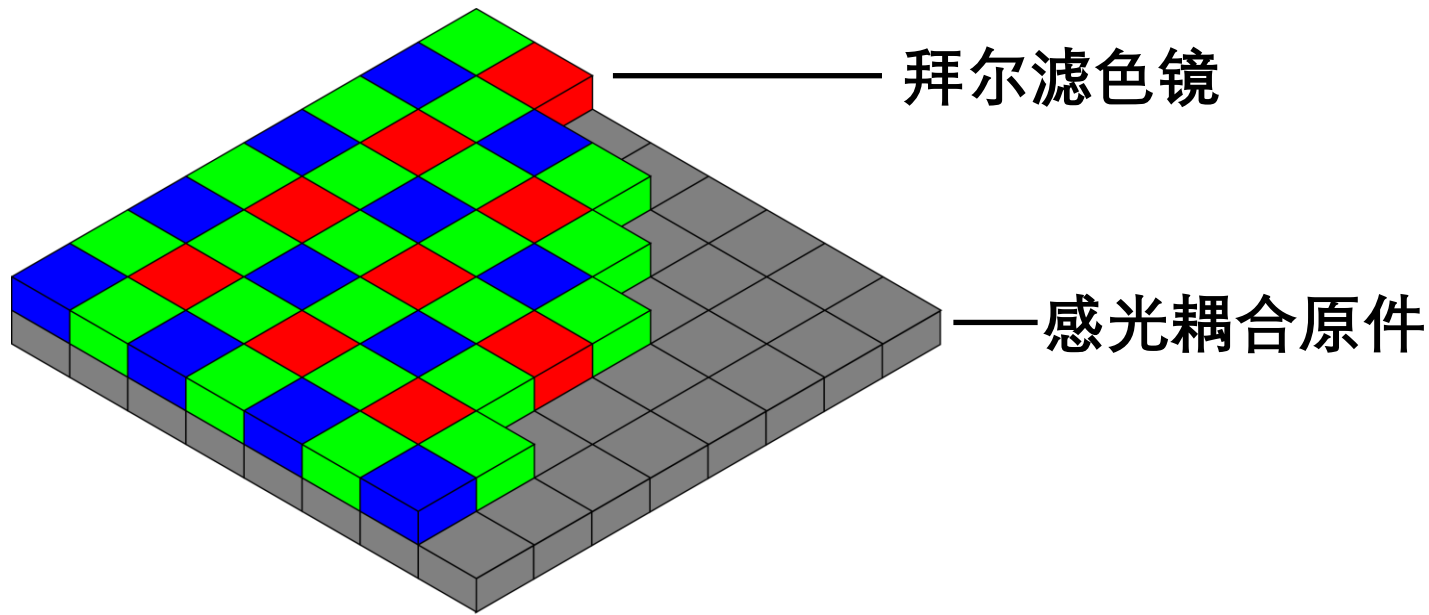
视锥细胞镶嵌

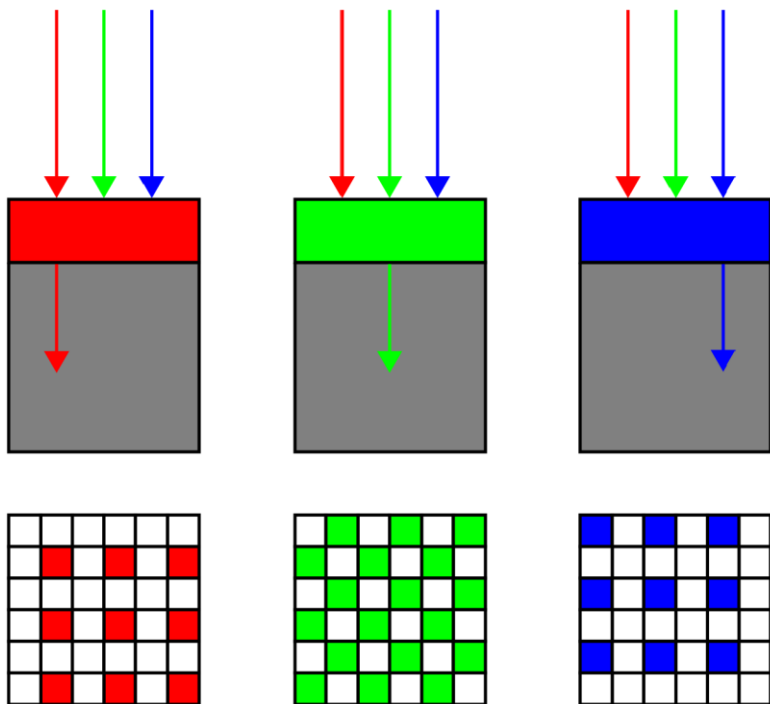
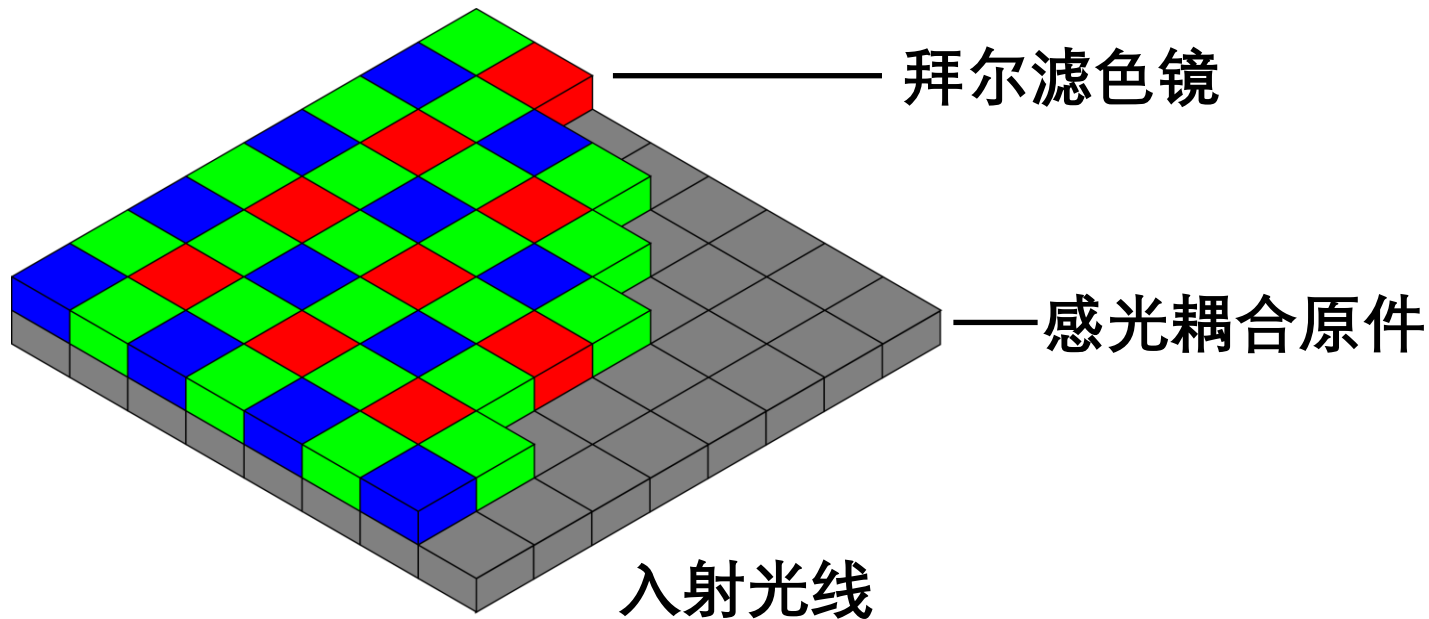


拜尔滤色镜



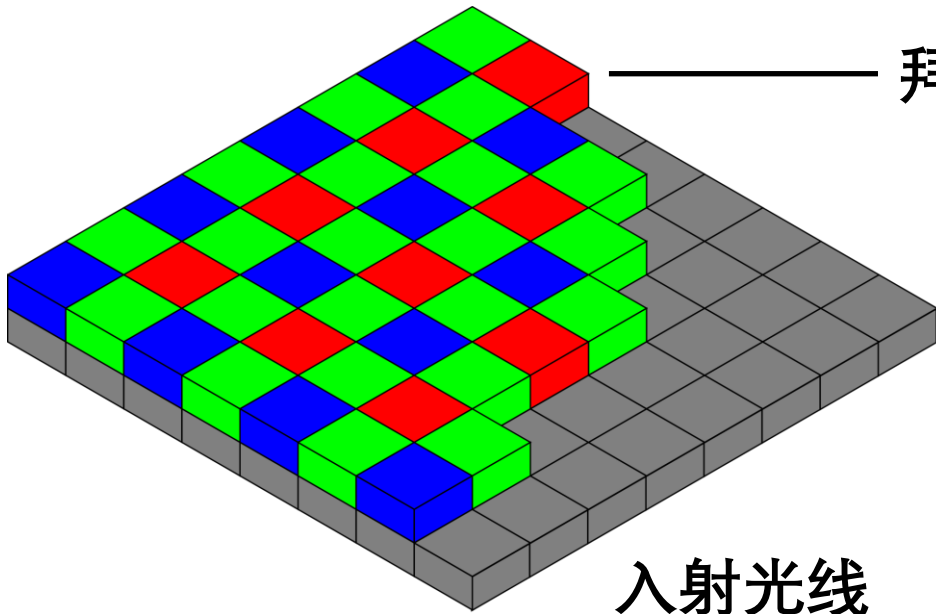
绿色：50%  
红色：25%  
蓝色：25%



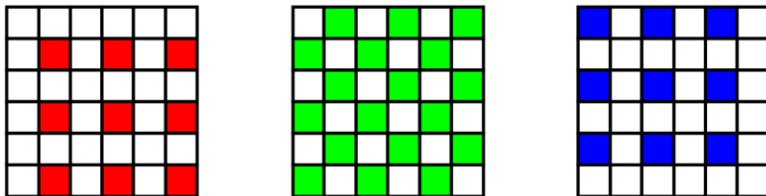
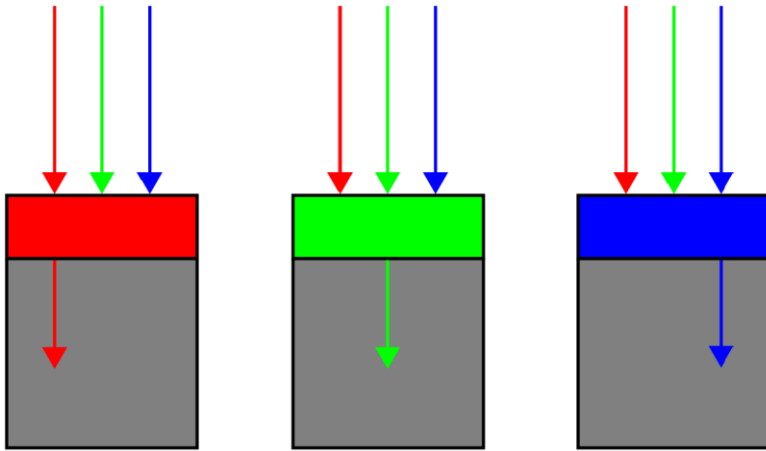


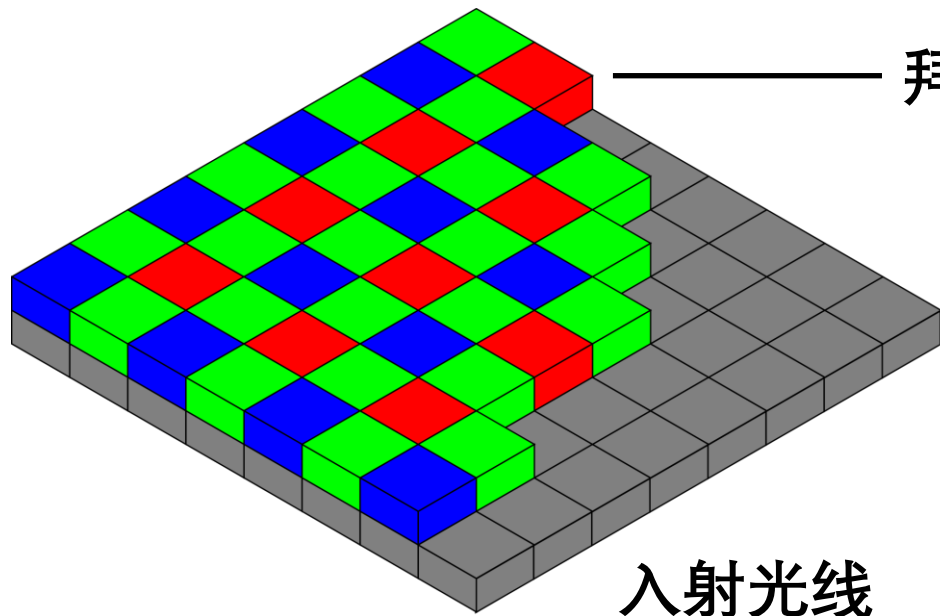
拜尔滤色镜

感光耦合原件



入射光线

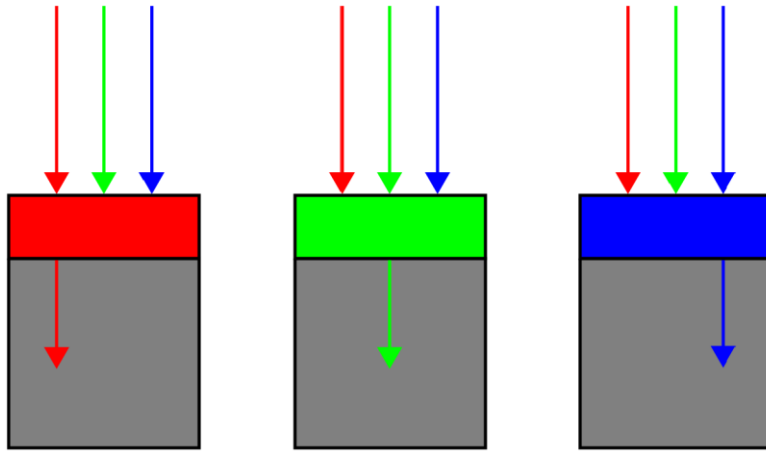




拜尔滤色镜

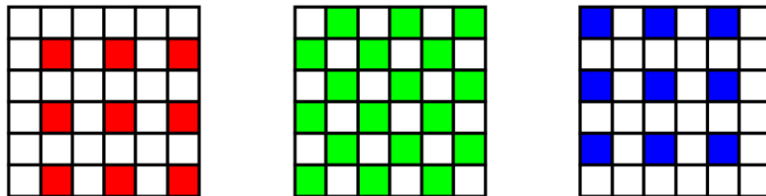
感光耦合原件

入射光线



滤色镜

传感器阵列



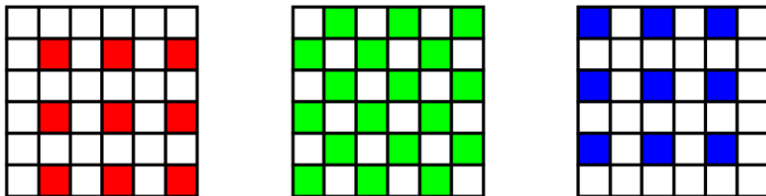
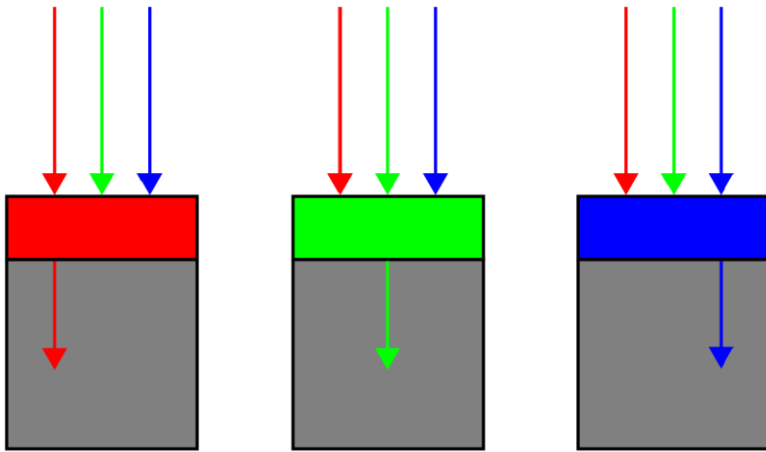
拜尔滤色镜

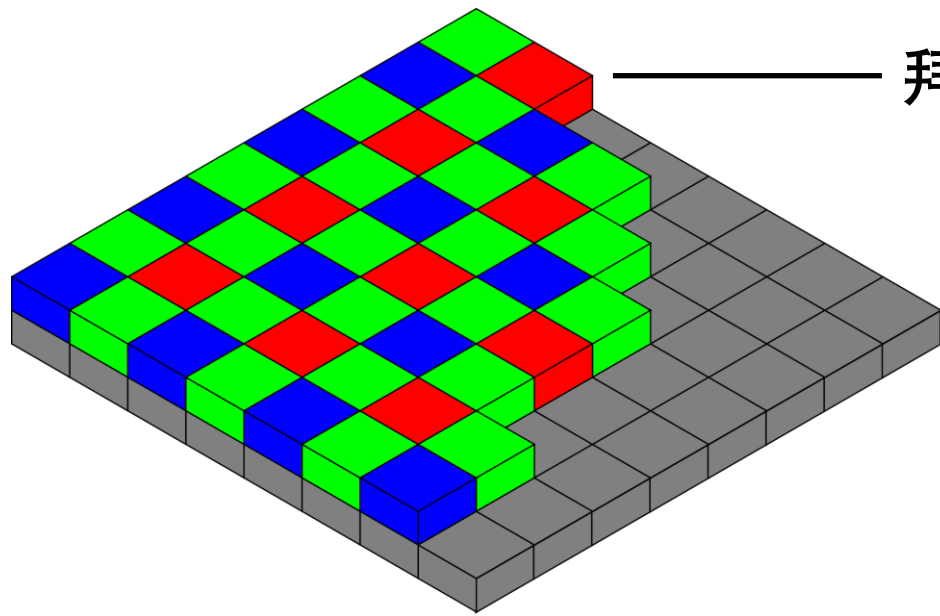
感光耦合原件



滤色镜

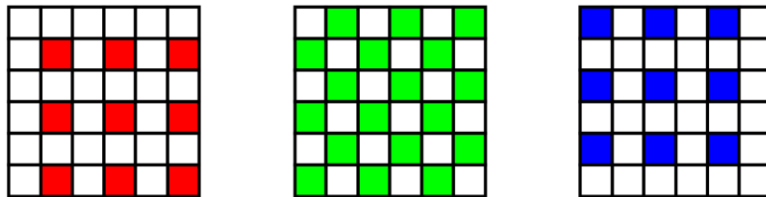
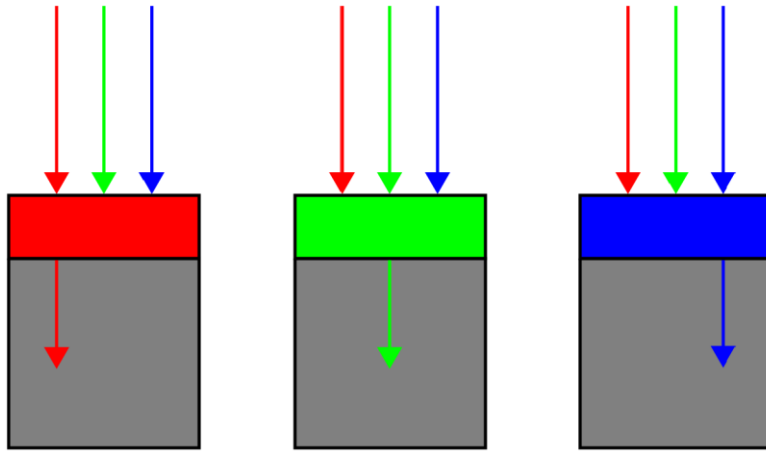
传感器阵列





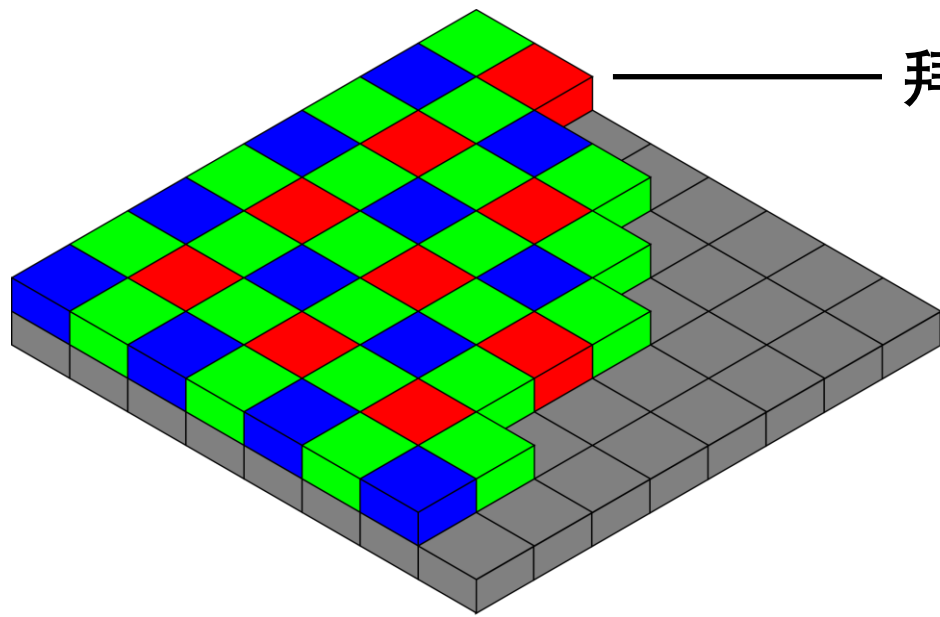
拜尔滤色镜

感光耦合原件



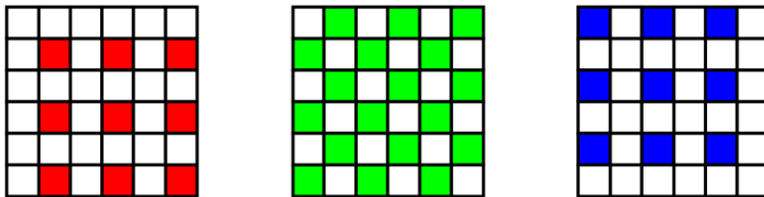
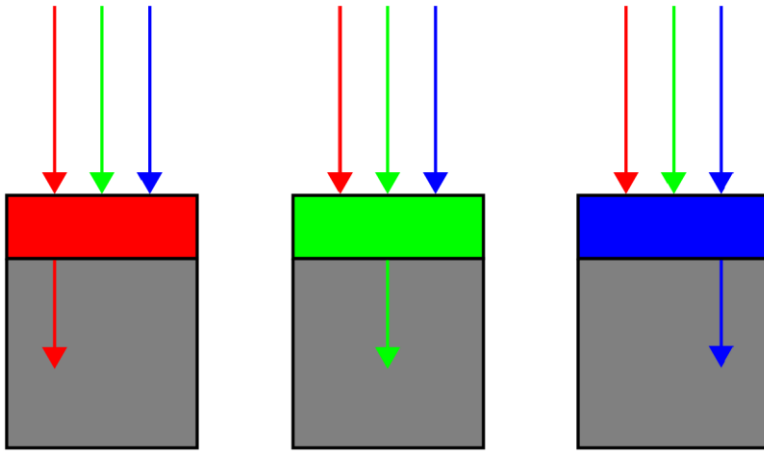
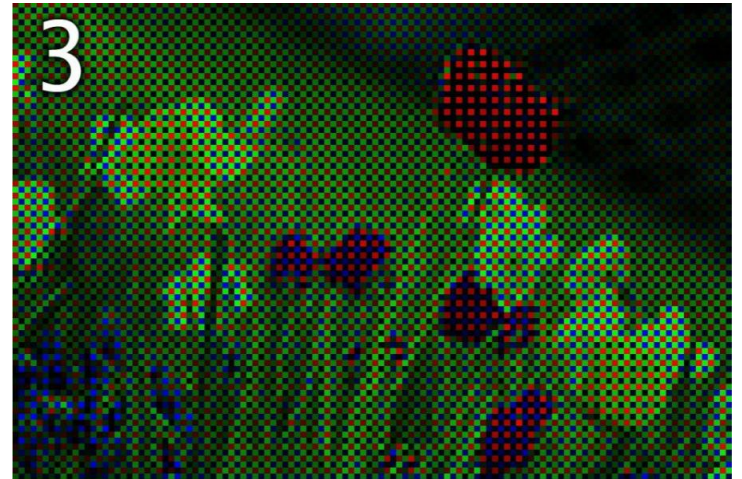
色彩模板





拜尔滤色镜

感光耦合原件

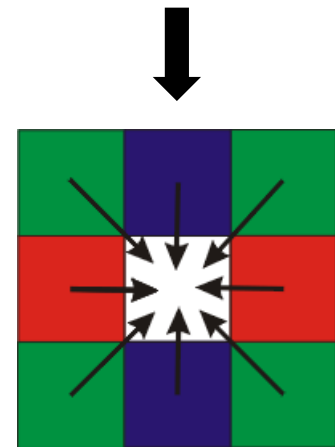
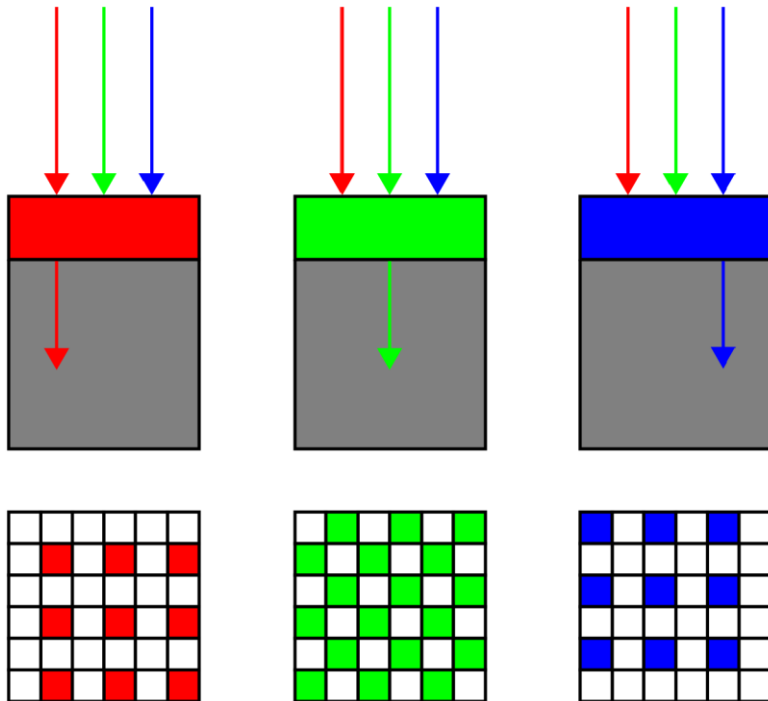
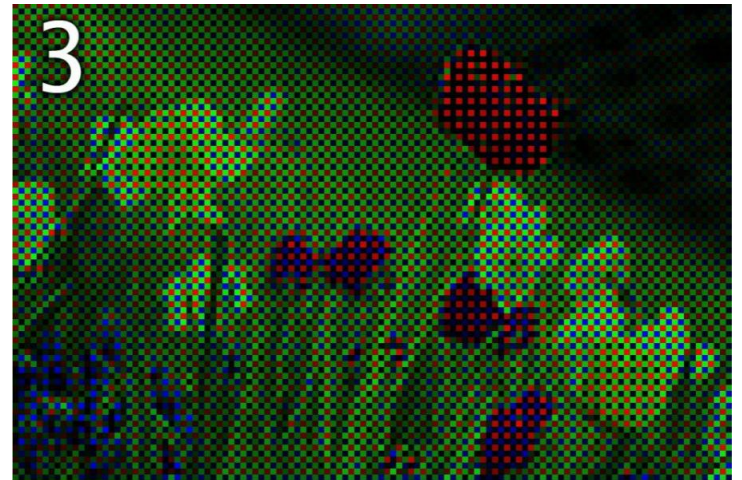
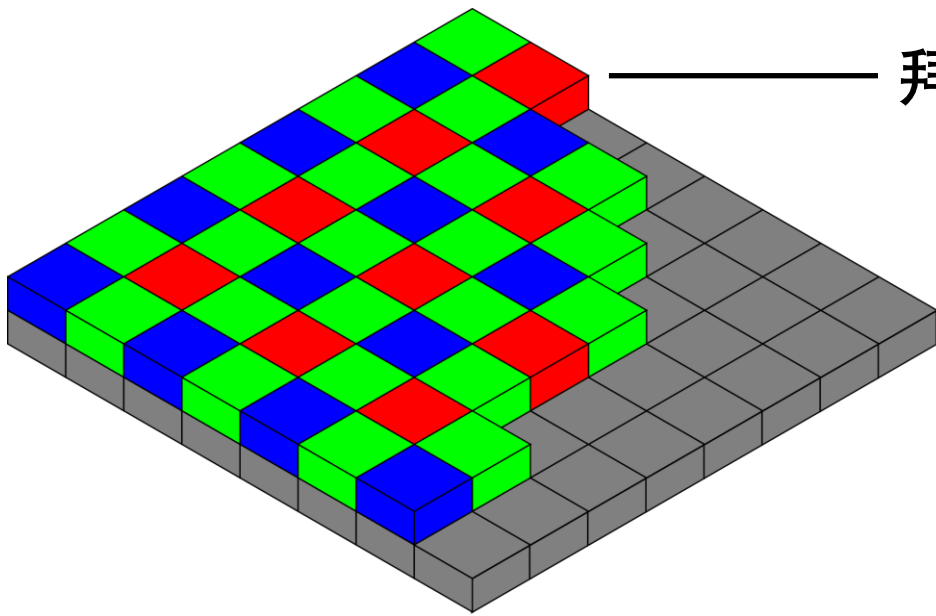


色彩模板



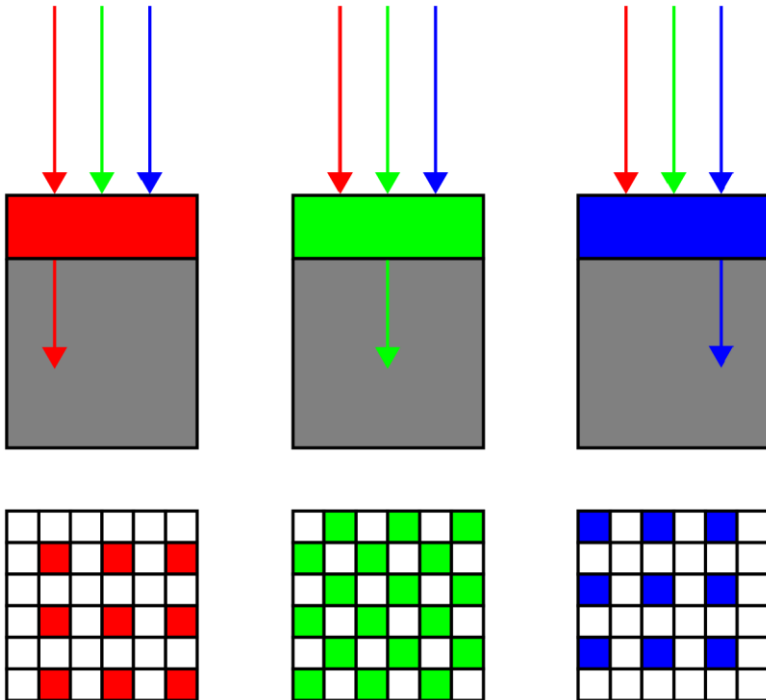
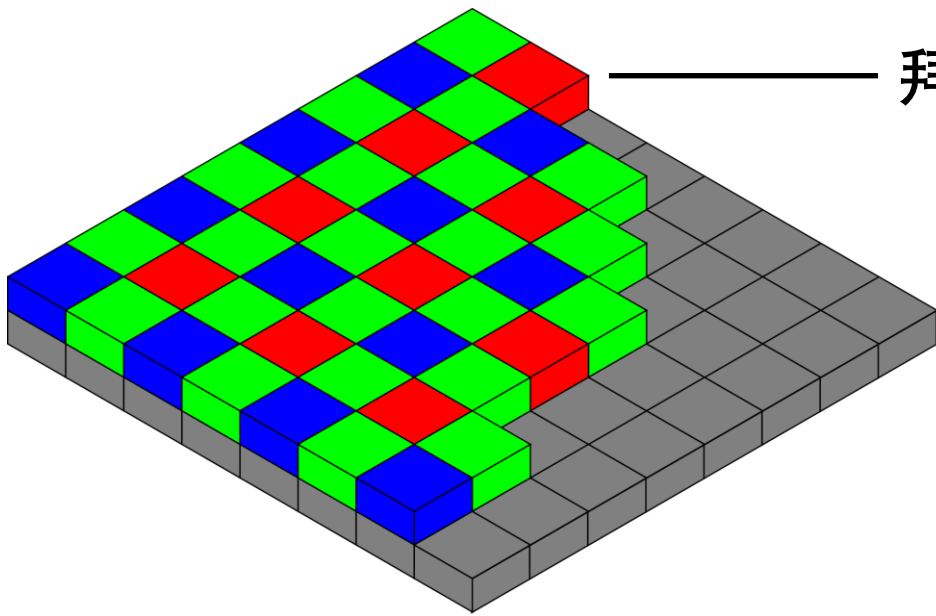
拜尔滤色镜

感光耦合原件



拜尔滤色镜

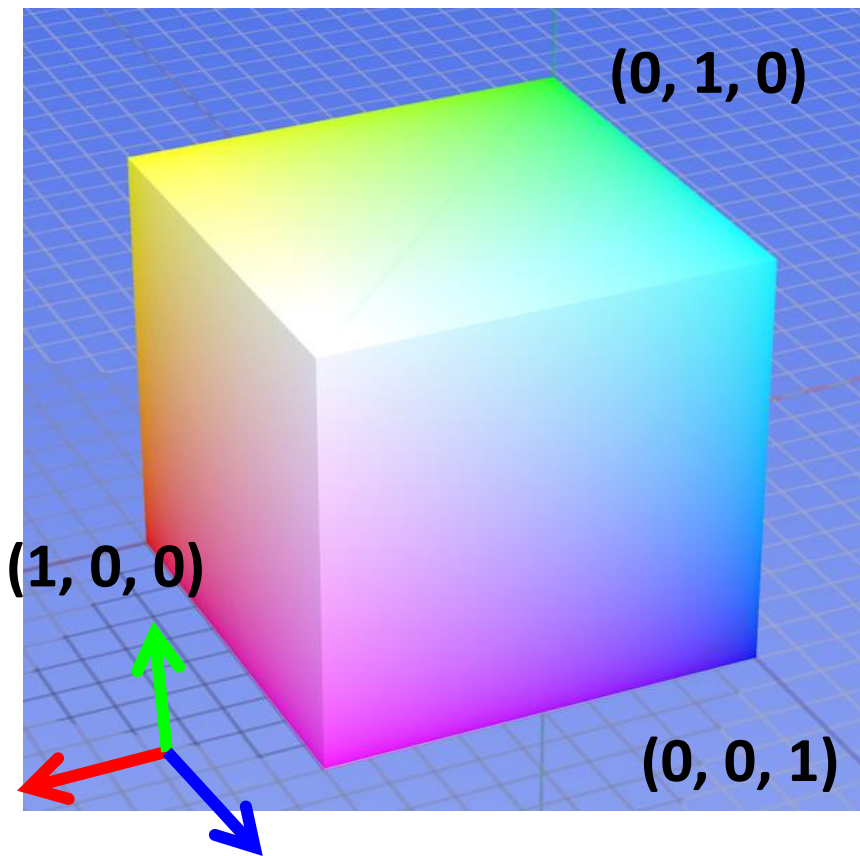
感光耦合原件



A brick wall is shown, illuminated by a light source that creates a spectrum of colors. The light is brightest in the center, where it appears white and yellow, and fades into blue on the left and red on the right. The brick pattern is visible throughout the image.

如何表示颜色？

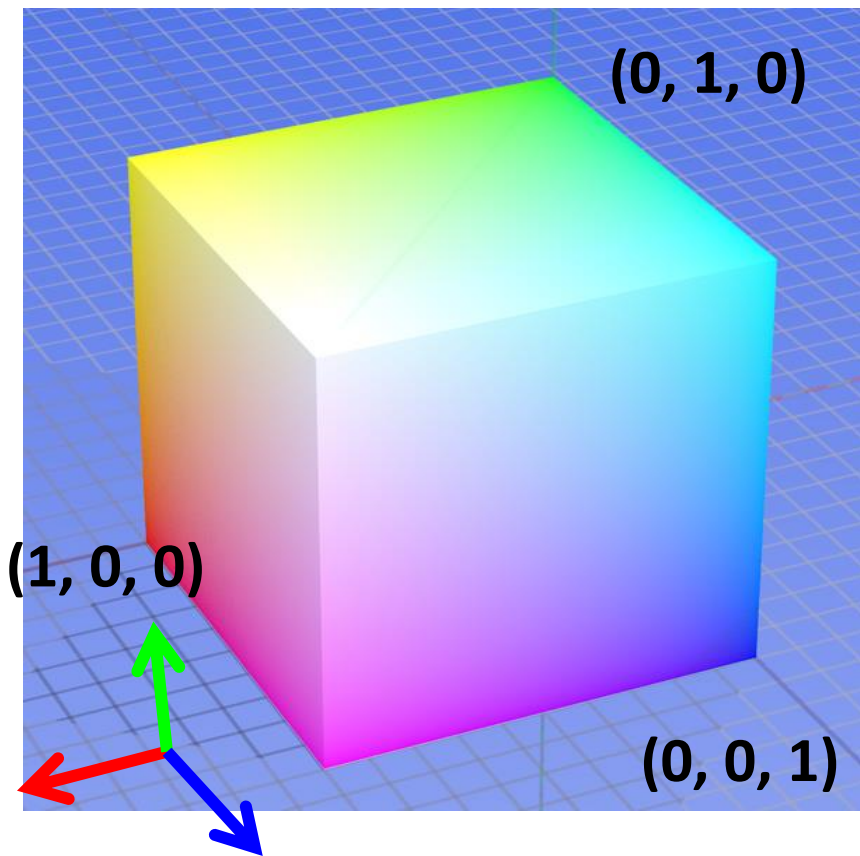
色彩空间  
RGB



$$8 \text{ bit} \times 3 = 24 \text{ bit}$$

$$256 \times 256 \times 256 \\ \approx 1677 \text{ 万色}$$

色彩空间  
RGB



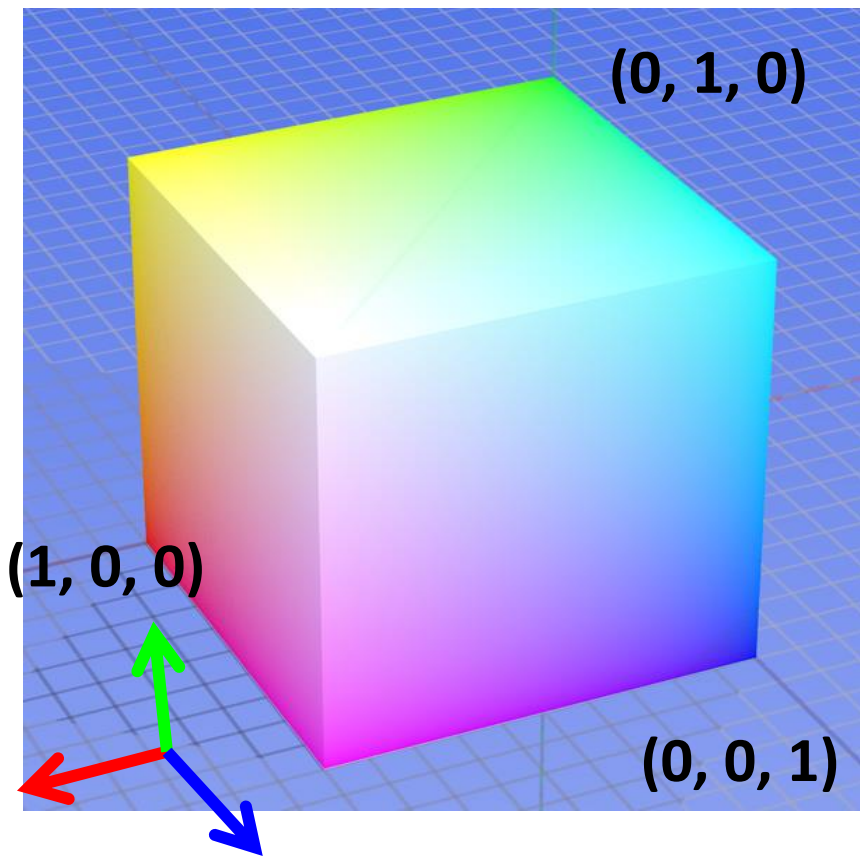
$$8 \text{ bit} \times 3 = 24 \text{ bit}$$

$$256 \times 256 \times 256 \\ \approx 1677 \text{ 万色}$$

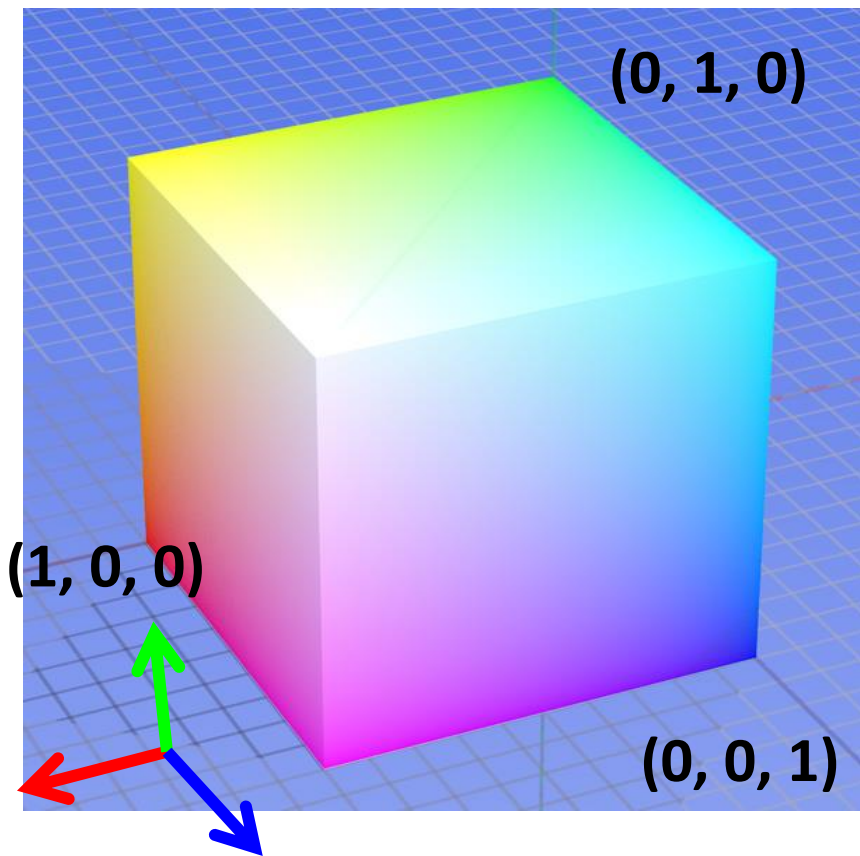
24位真彩色



色彩空间  
RGB



色彩空间  
RGB



缺点：通道间相关性强  
非感知



**R**  
(G = 0, B = 0)

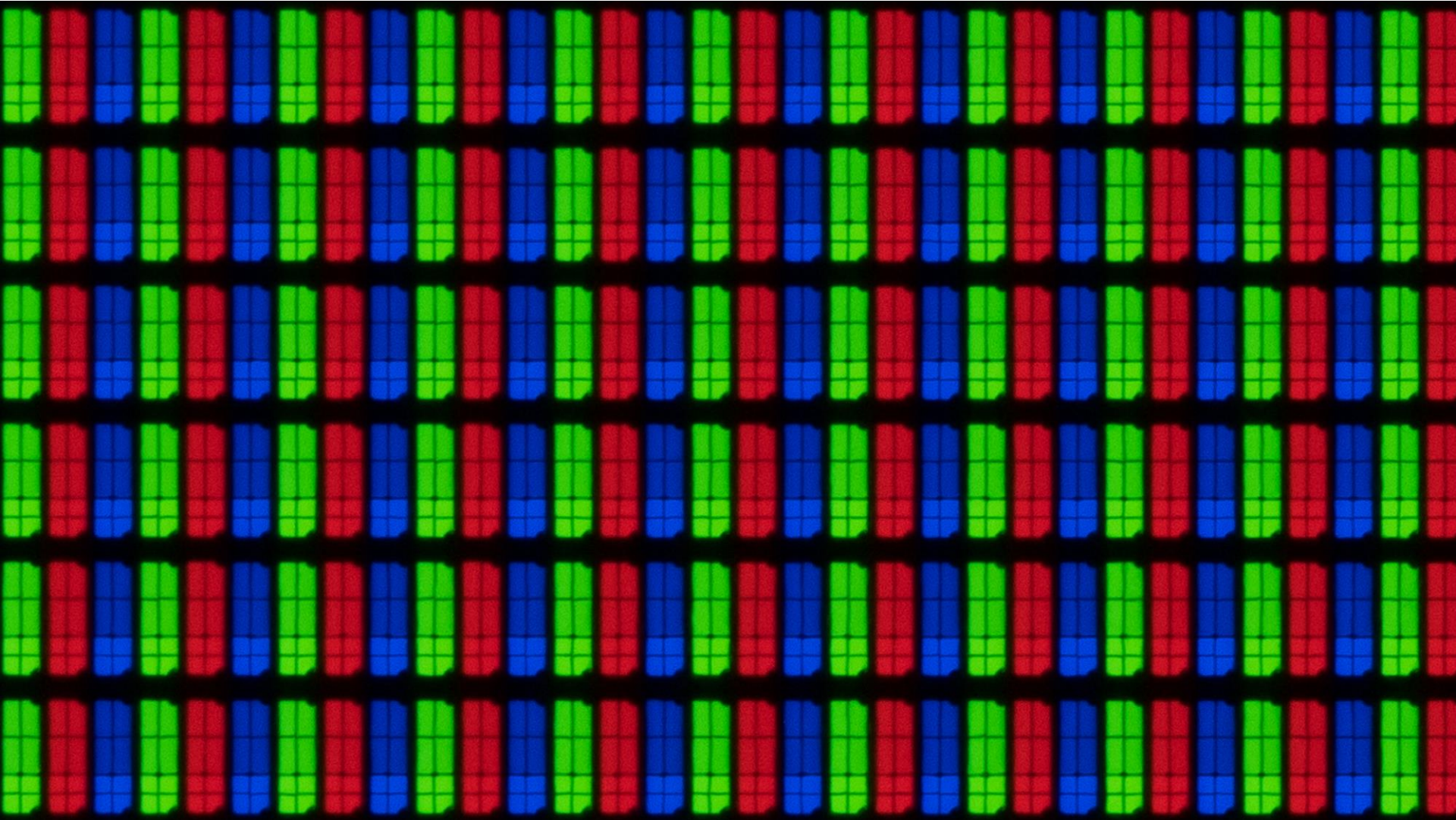


**G**  
(R = 0, B = 0)



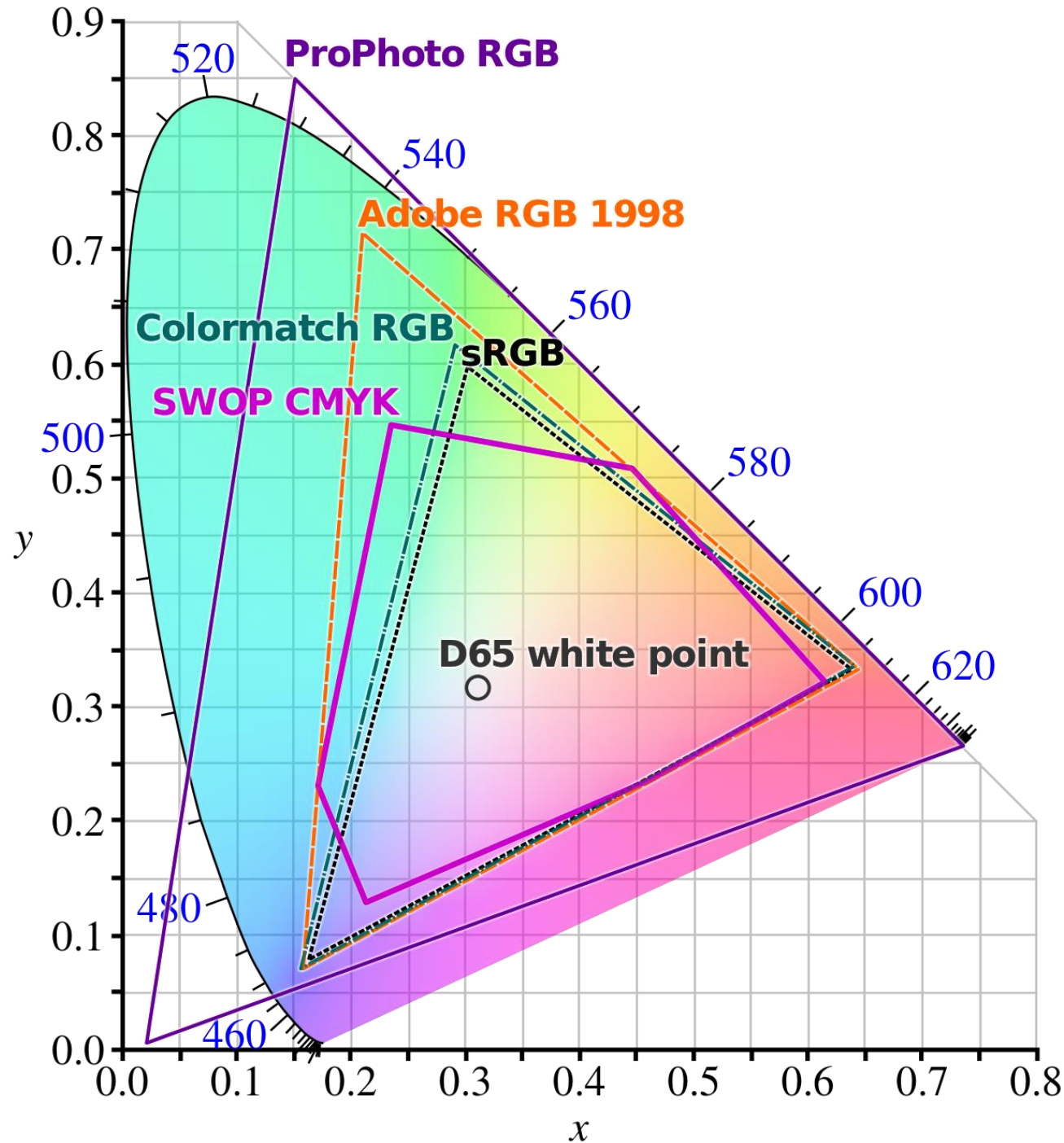
**B**  
(R = 0, G = 0)







RGB色域

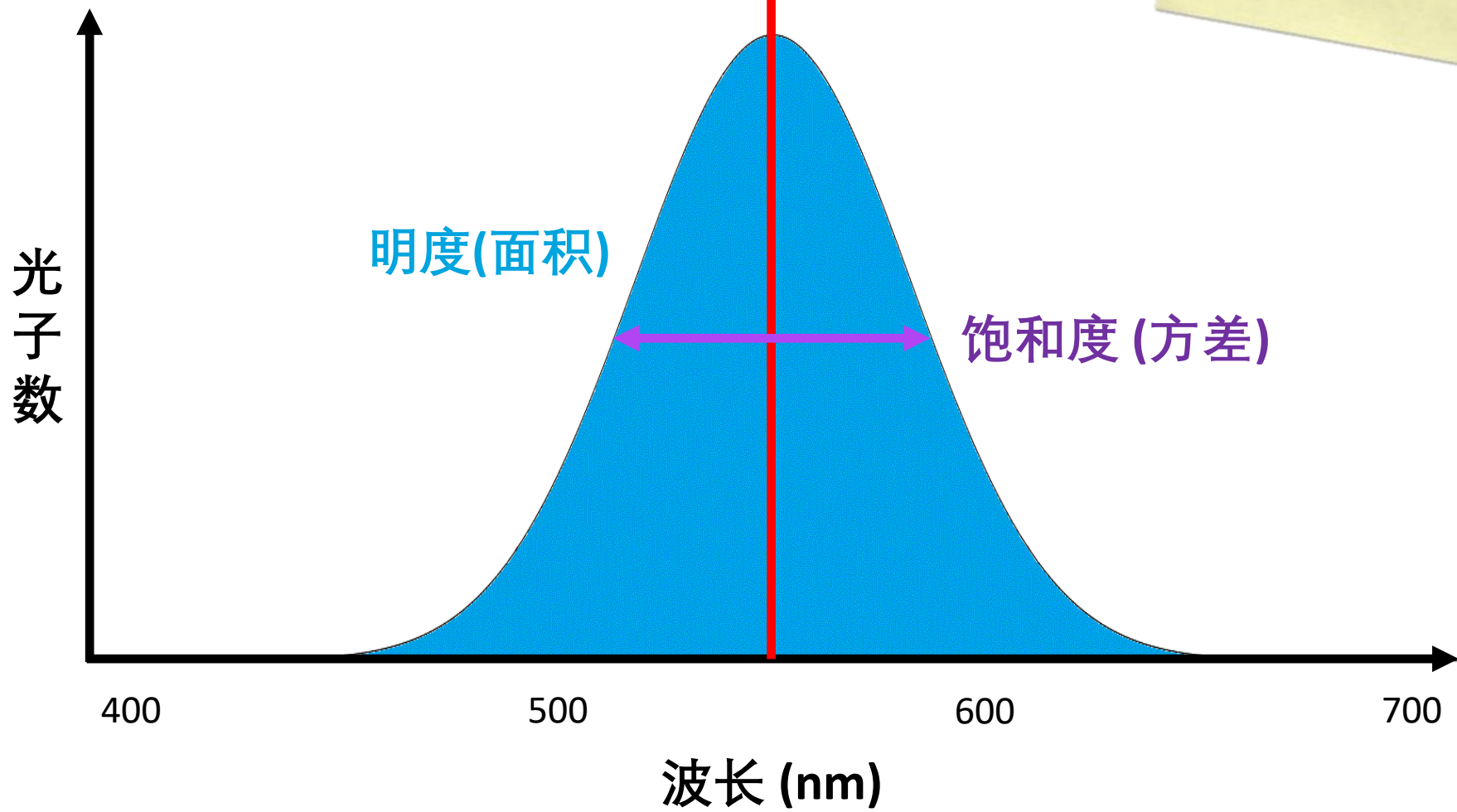


回顾

色相 (均值)

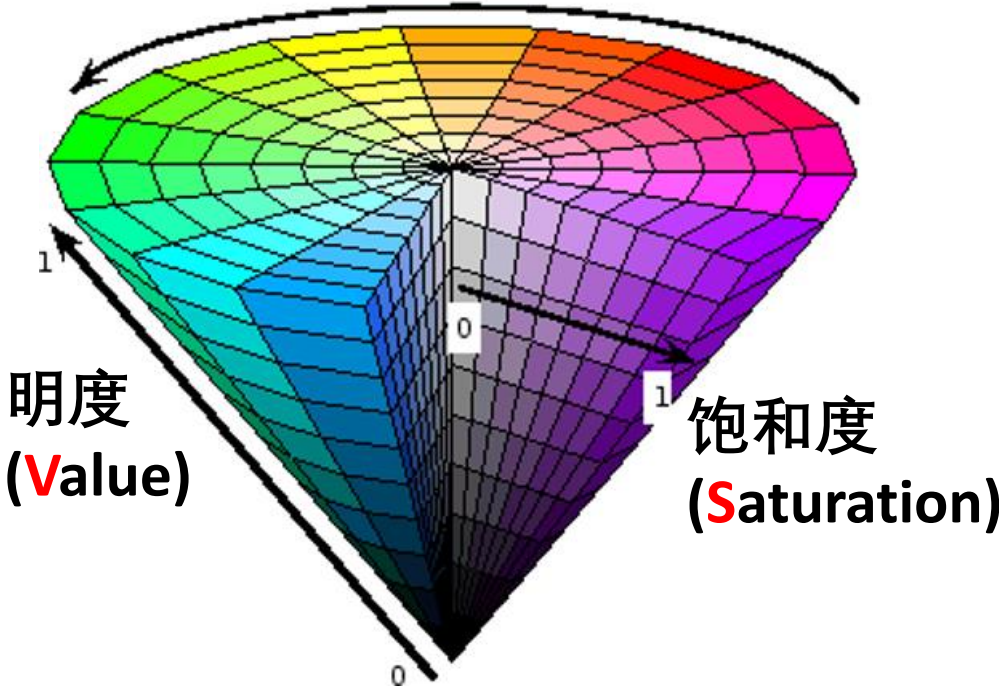
明度(面积)

饱和度 (方差)



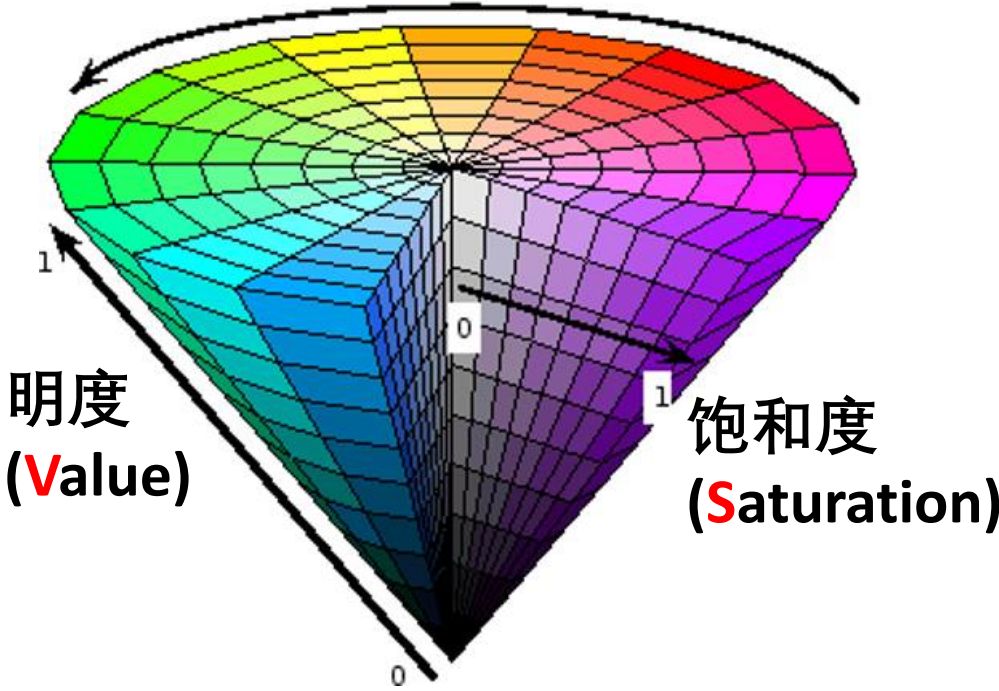
色彩空间  
HSV

色相 (Hue)



色彩空间  
HSV

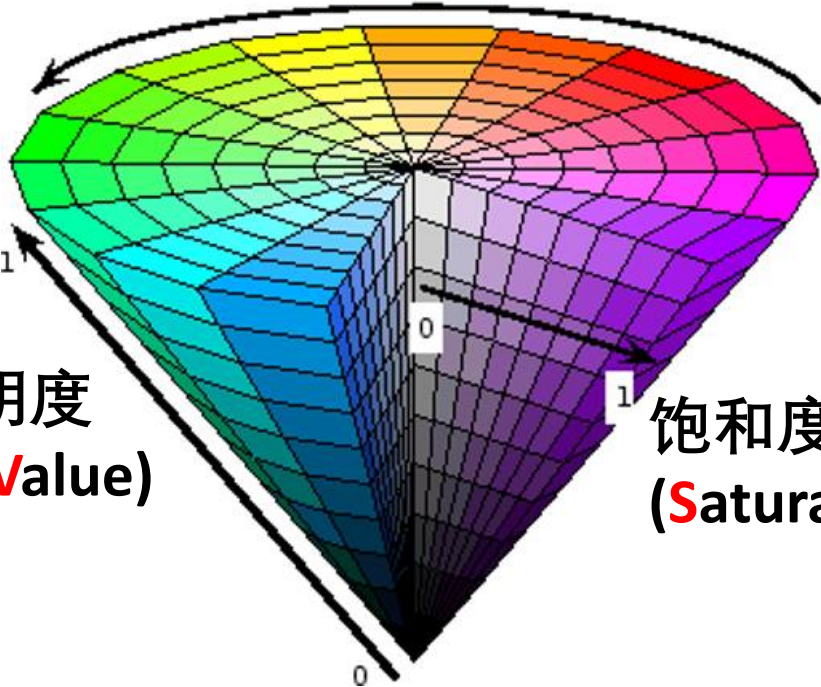
色相 (Hue)





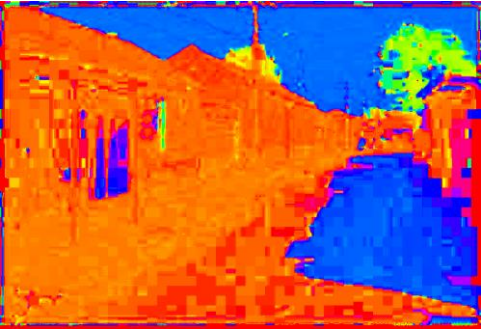
色彩空间  
HSV

色相 (Hue)



明度  
(Value)

饱和度  
(Saturation)



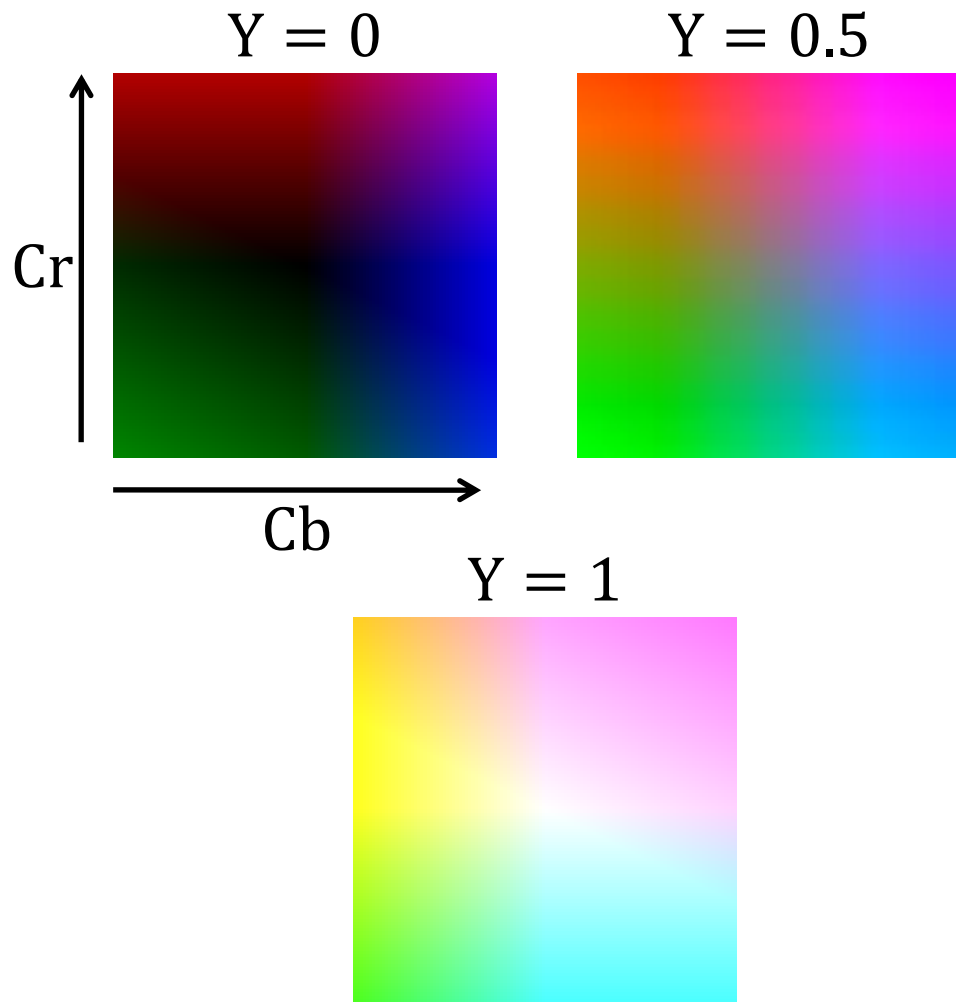
H  
(S = 1, V = 1)



S  
(H = 1, V = 1)



V  
(H = 1, S = 0)

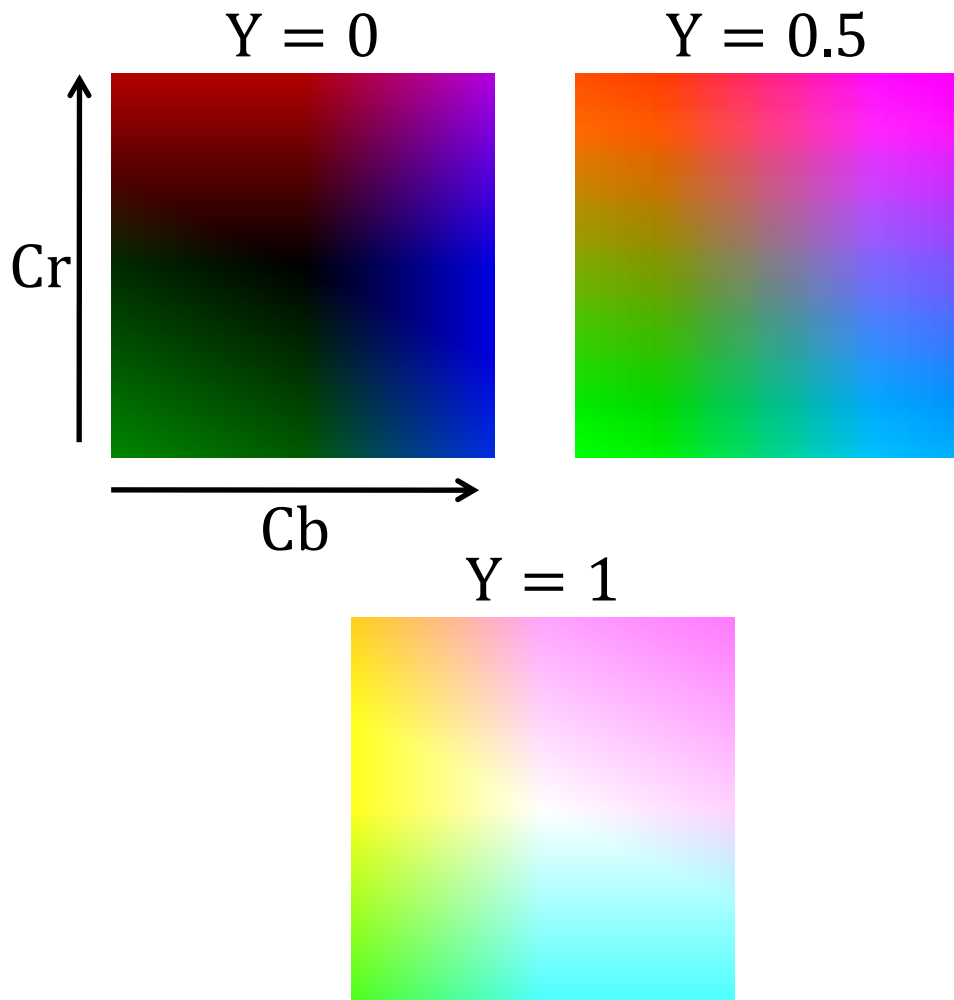


Y：明流，表示光的浓度且非线性

Cr：红色浓度偏移成分

Cb：蓝色浓度偏移成分

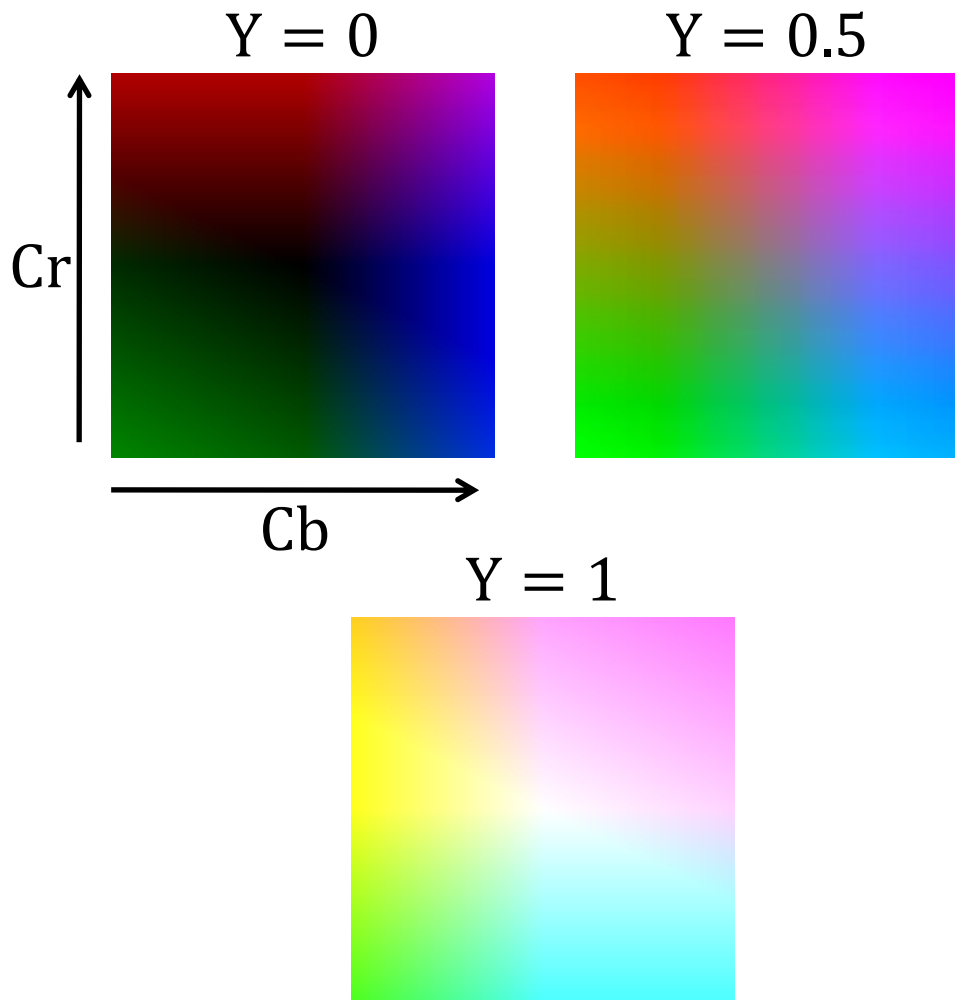




色彩空间  
YCrCb



Y：明流，表示光的浓度且非线性  
Cr：红色浓度偏移成分  
Cb：蓝色浓度偏移成分



色彩空间  
YCrCb



**Y**  
(Cr = 0.5, Cb = 0.5)



**Cb**  
(Y = 0.5, Cr = 0.5)

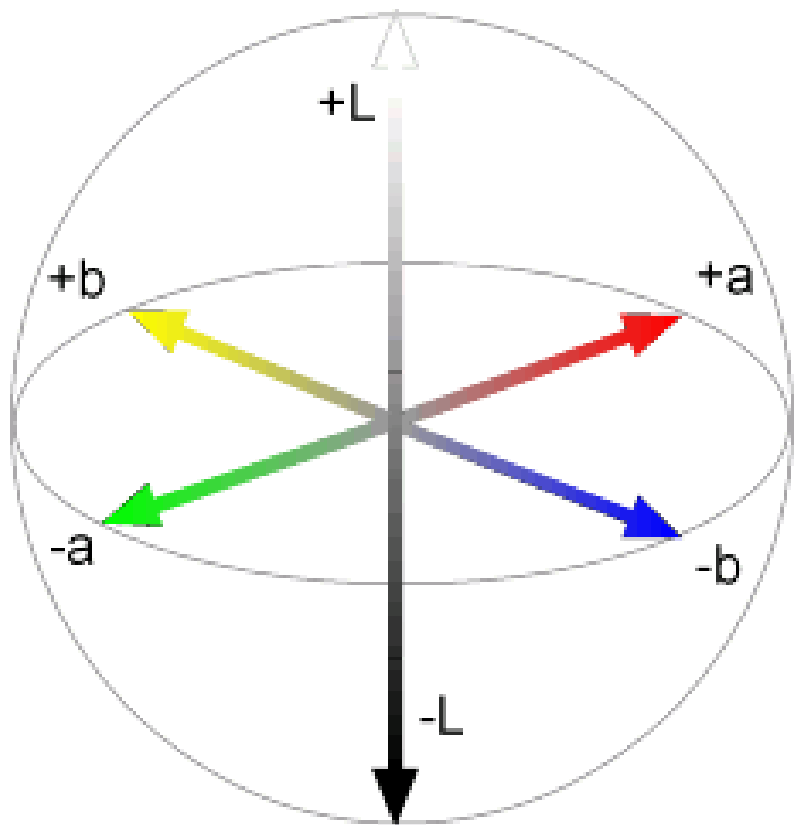


**Cr**  
(Y = 0.5, Cb = 0.5)

Y：明流，表示光的浓度且非线性  
Cr：红色浓度偏移成分  
Cb：蓝色浓度偏移成分

色彩空间  
CIELAB

“感知均匀”

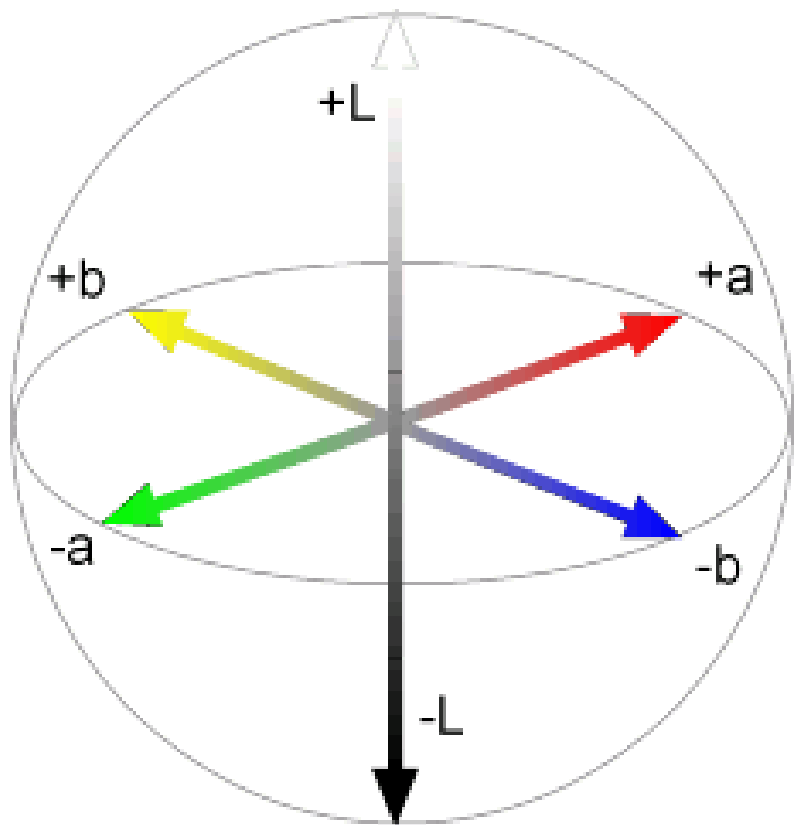


$$16 \text{ bit} \times 3 = 48 \text{ bit}$$

$$65536 \times 65536 \times 65536 \approx 281 \text{ 千亿色}$$

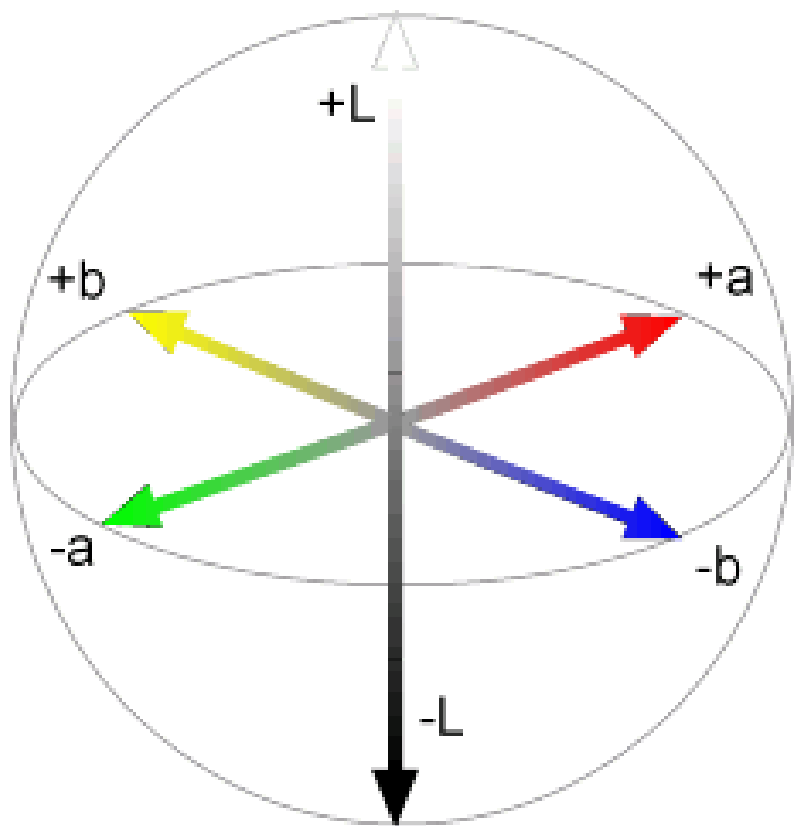
色彩空间  
CIELAB

“感知均匀”



# 色彩空间 CIELAB

“感知均匀”



**L**  
( $a = 0, b = 0$ )



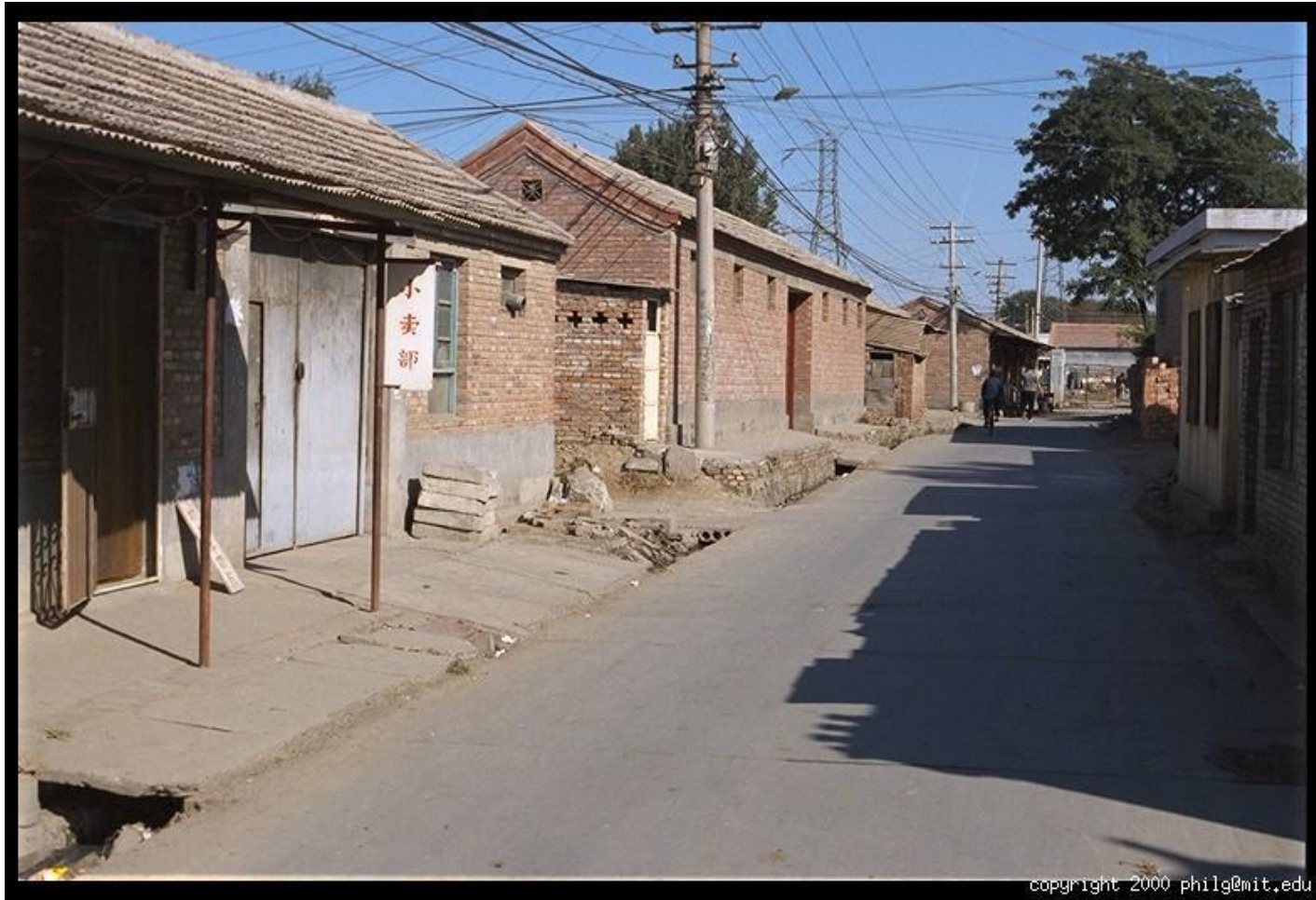
**a**  
( $L = 65, b = 0$ )



**b**  
( $L = 65, a = 0$ )



如果你不得不选择，你宁愿放弃亮度  
还是色度呢？



原图



仅显示颜色——恒定强度



copyright 2000 philg@mit.edu

仅显示强度——恒定色彩



因此在绝大多数计算机视觉任务中只使用灰度图

仅显示强度——恒定色彩



# 图像滤波

# 濾波器



滤波器



$I$

图像

$\phi$



滤波器





# 图像噪声



原始图像





椒盐噪声





# 高斯噪声



# 图像噪声

$$I(x, y) = I_{\text{ideal}}(x, y) + \eta(x, y)$$

# 图像噪声

$$I(x, y) = I_{\text{ideal}}(x, y) + \eta(x, y)$$

其中

$$\eta(x, y) \sim \mathcal{N}(\mu = 0, \sigma)$$

# 图像噪声

$$I(x, y) = I_{\text{ideal}}(x, y) + \eta(x, y)$$

其中

$$\eta(x, y) \sim \mathcal{N}(\mu = 0, \sigma)$$

假设噪声独立同分布



# 图像噪声

$$I(x, y) = I_{\text{ideal}}(x, y) + \eta(x, y)$$

其中

$$\eta(x, y) \sim \mathcal{N}(\mu = 0, \sigma)$$

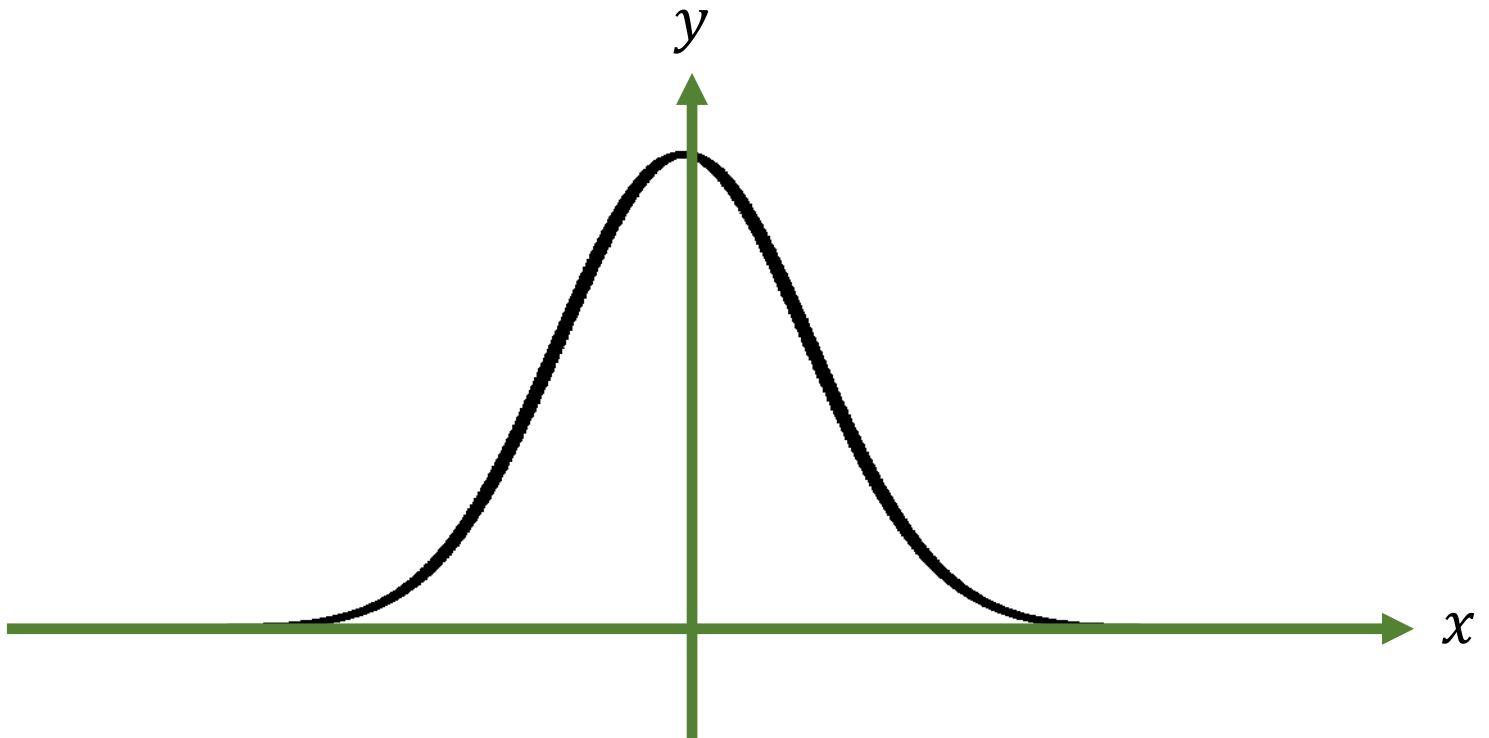
假设噪声独立同分布

I.I.D

$$I(x, y) = I_{\text{ideal}}(x, y) + \eta(x, y)$$

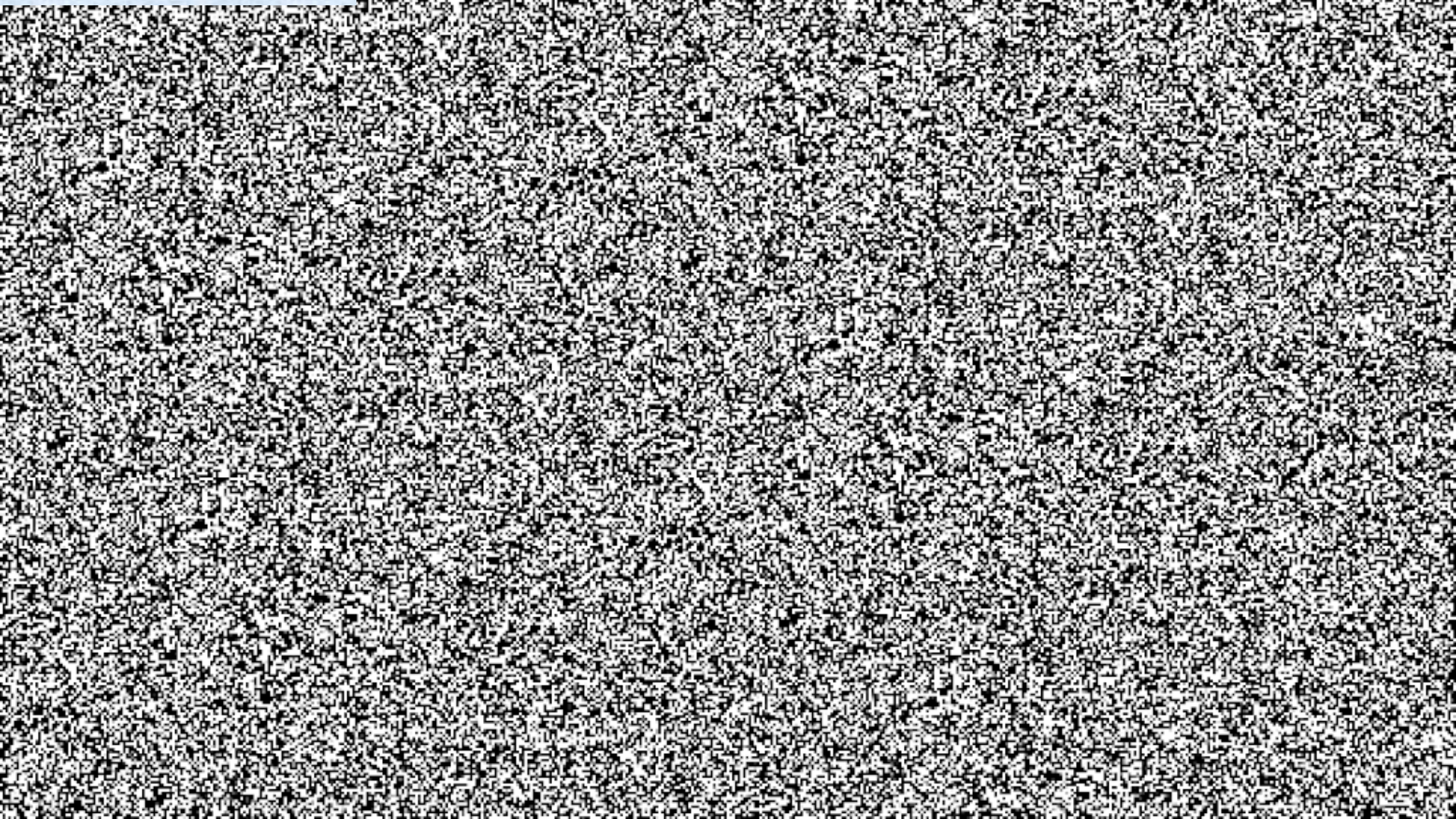
其中

$$\eta(x, y) \sim \mathcal{N}(\mu = 0, \sigma)$$



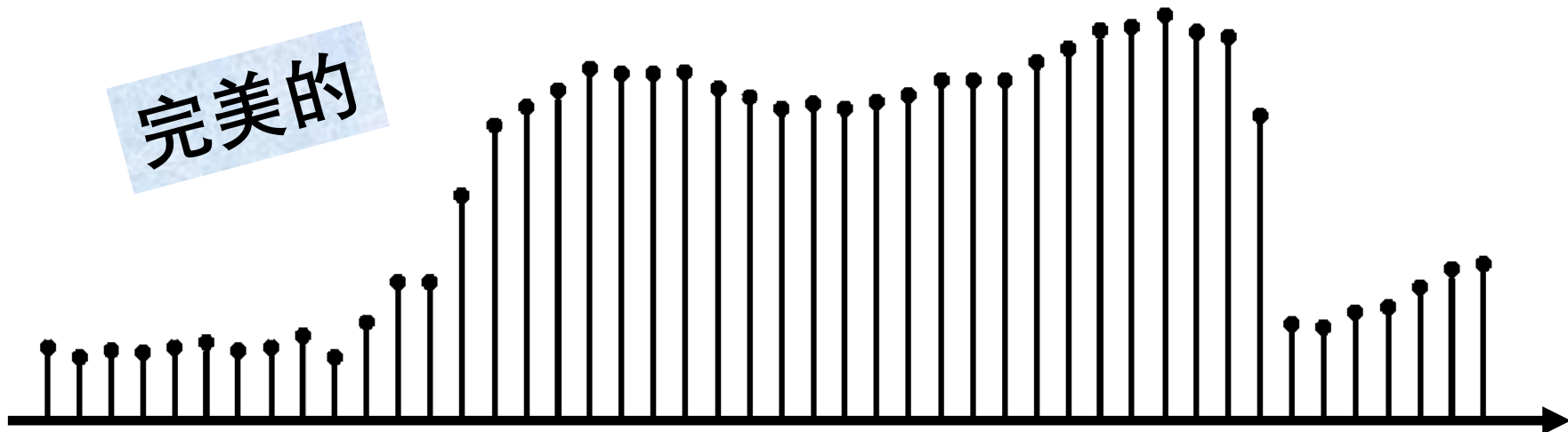


# 高斯噪声

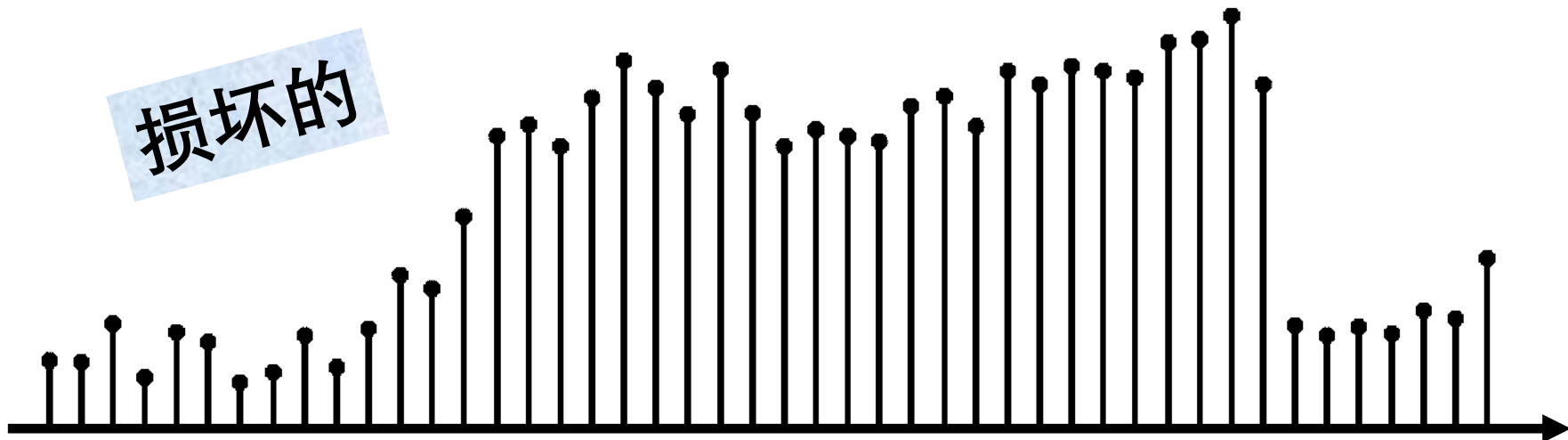




完美的

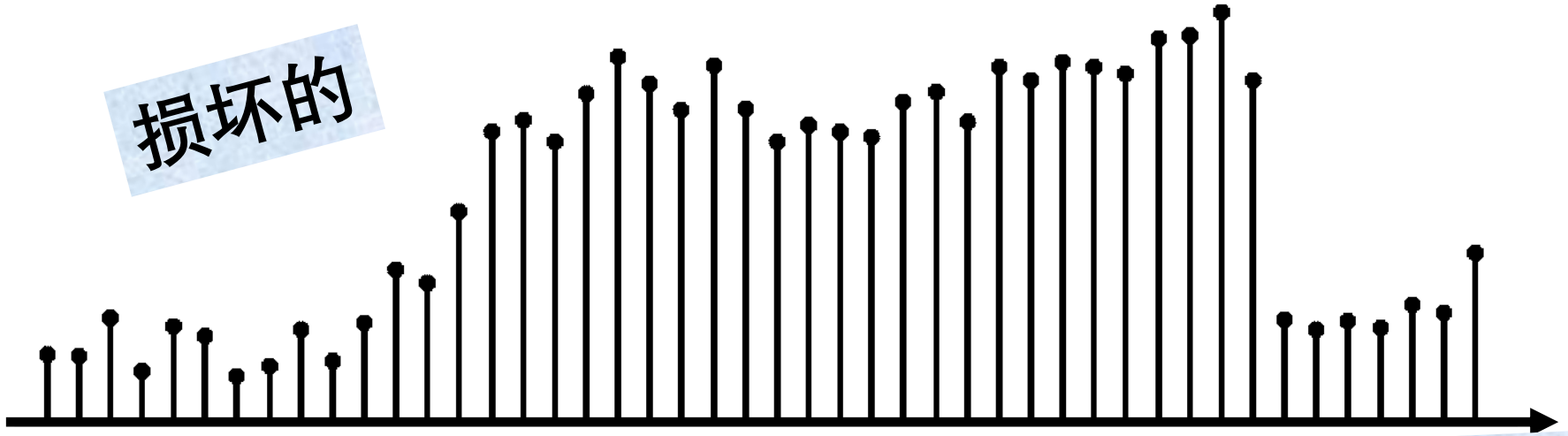


损坏的



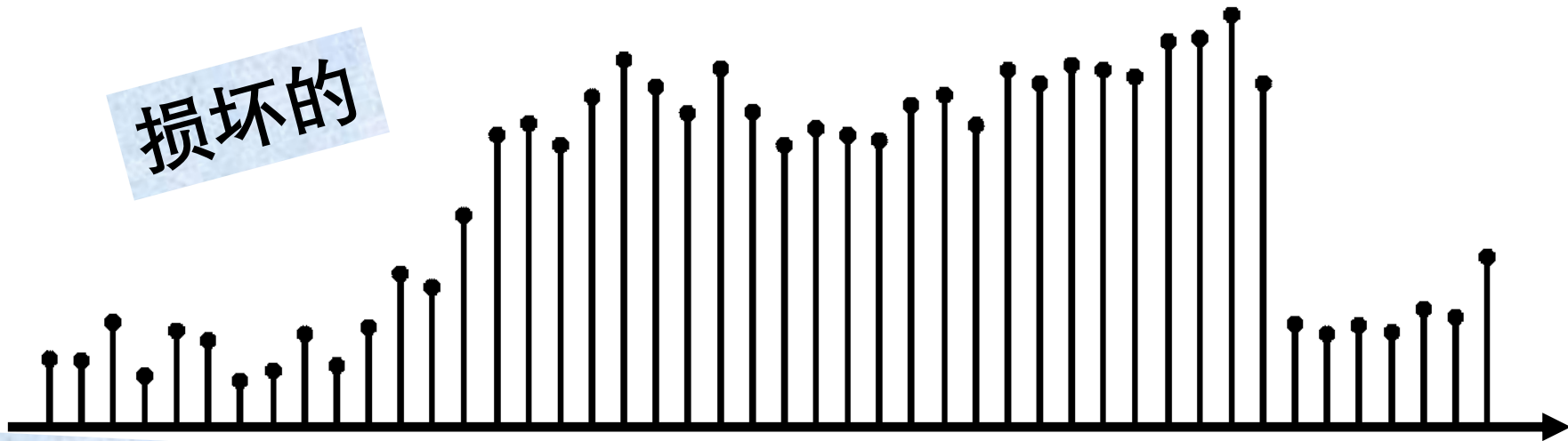


损坏的



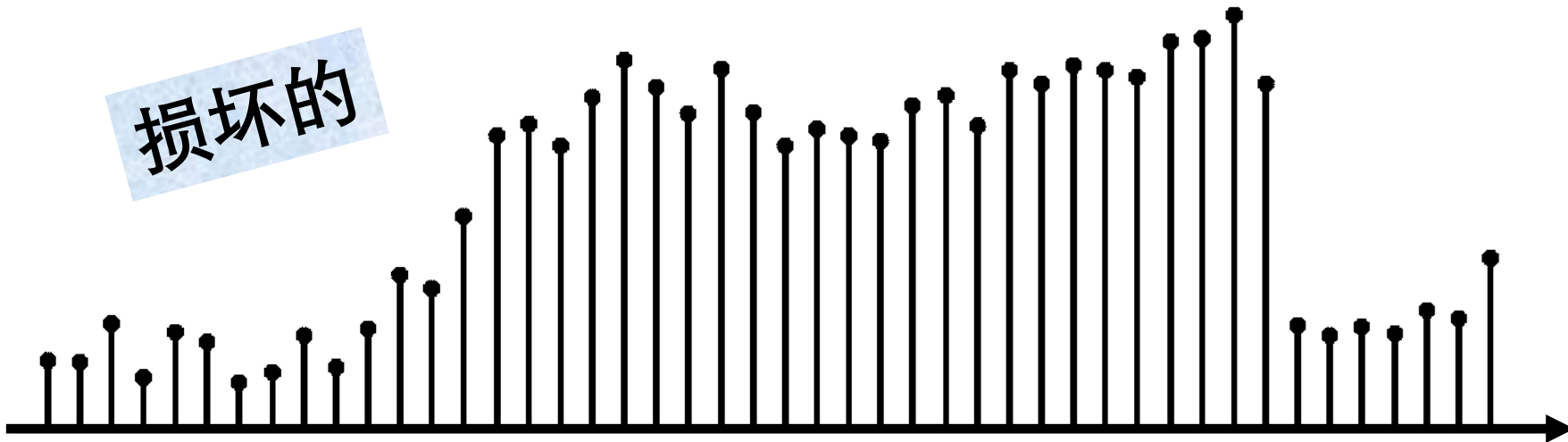
怎样才能消除噪声？

损坏的



假设噪声是I.I.D且相邻像素是相似的

损坏的

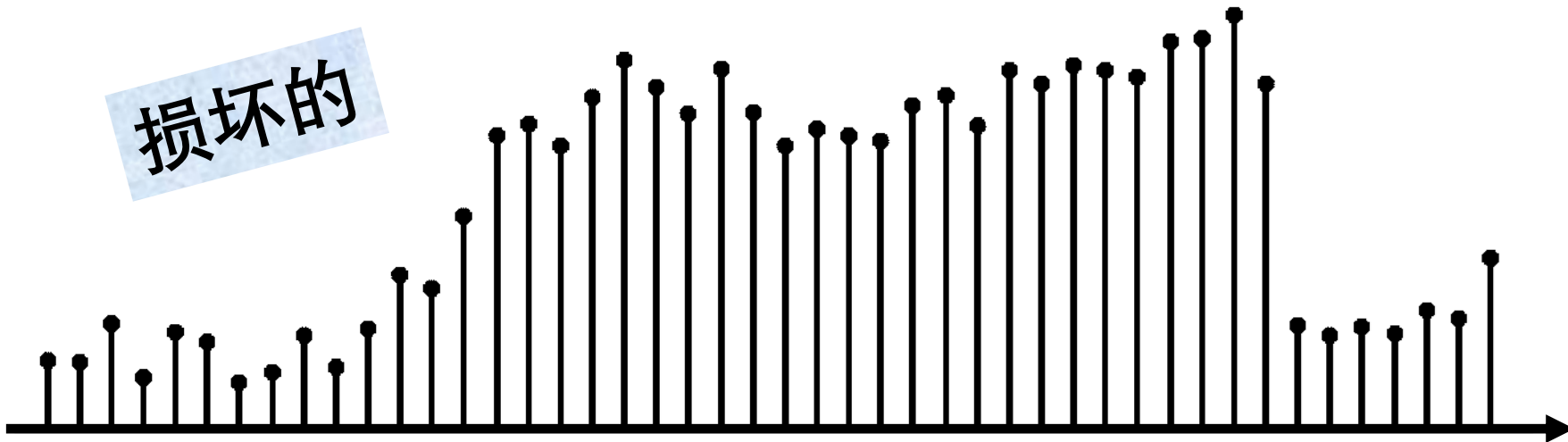


$$I(x, y) = I_{\text{ideal}}(x, y) + \eta(x, y)$$

其中

$$\eta(x, y) \sim \mathcal{N}(\mu = 0, \sigma)$$

损坏的

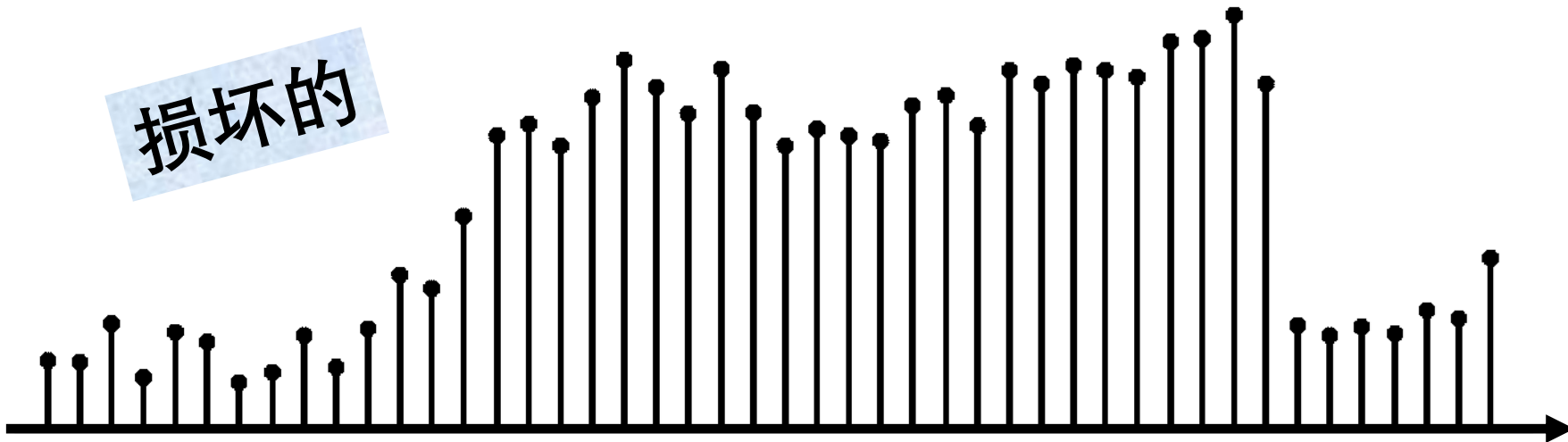


$$I(x, y) = I_{\text{ideal}}(x, y) + \eta(x, y)$$

其中

$$\eta(x, y) \sim \mathcal{N}(\mu = 0, \sigma)$$

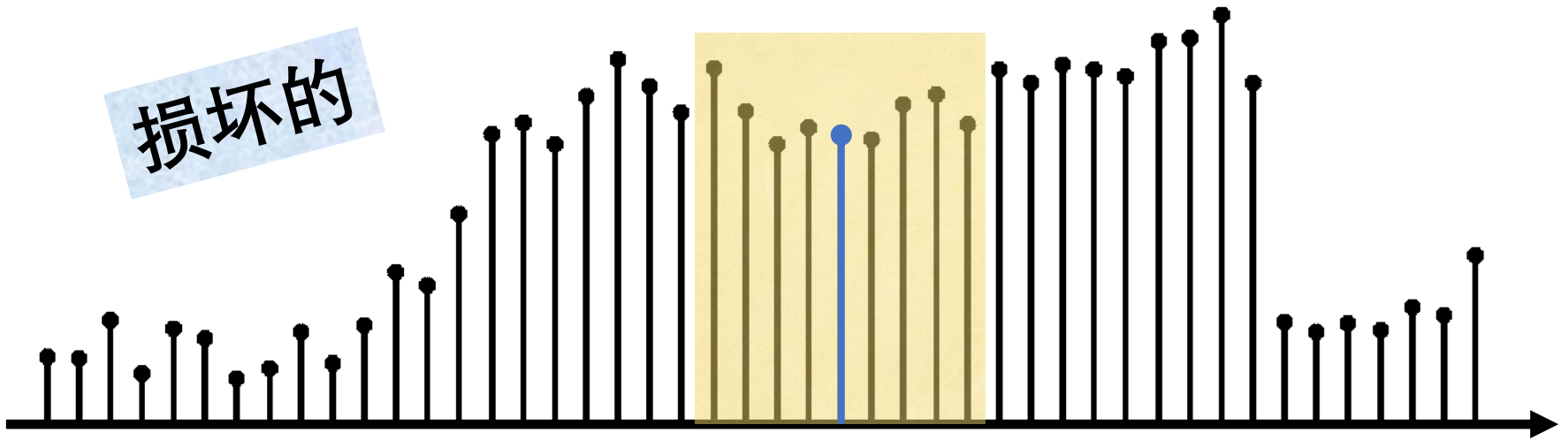
损坏的



让我们将每个像素替换为其相邻像素的平均值

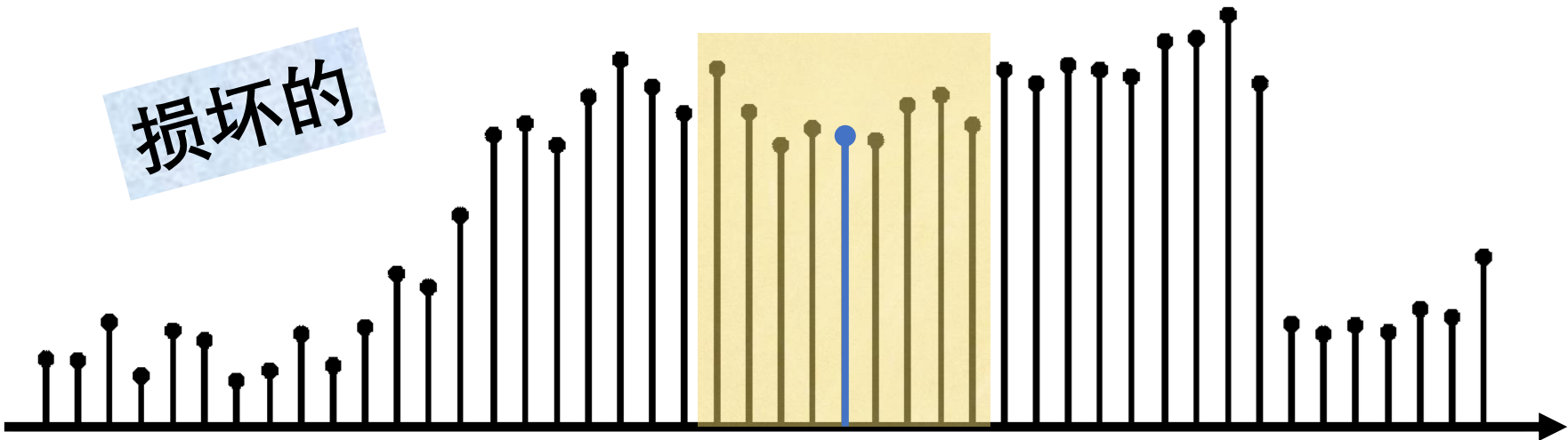


损坏的

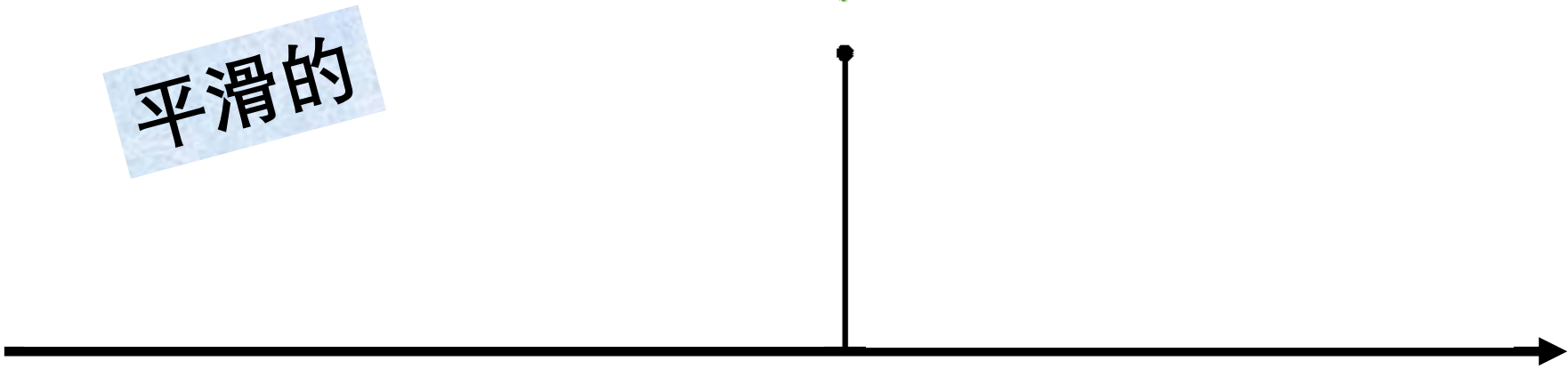


让我们将每个像素替换为其相邻像素的平均值

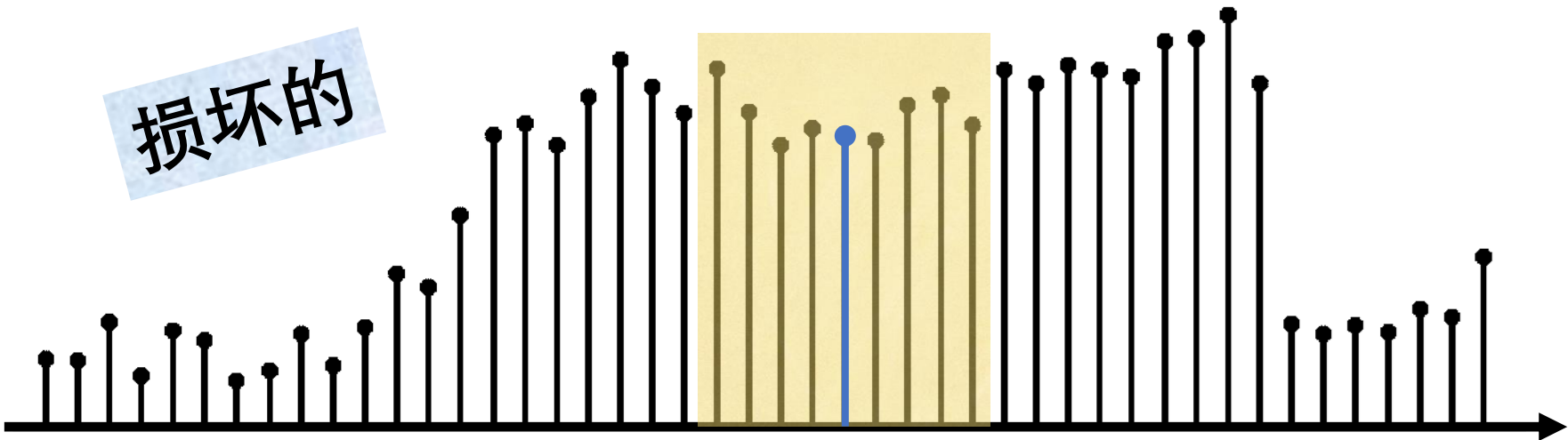
损坏的



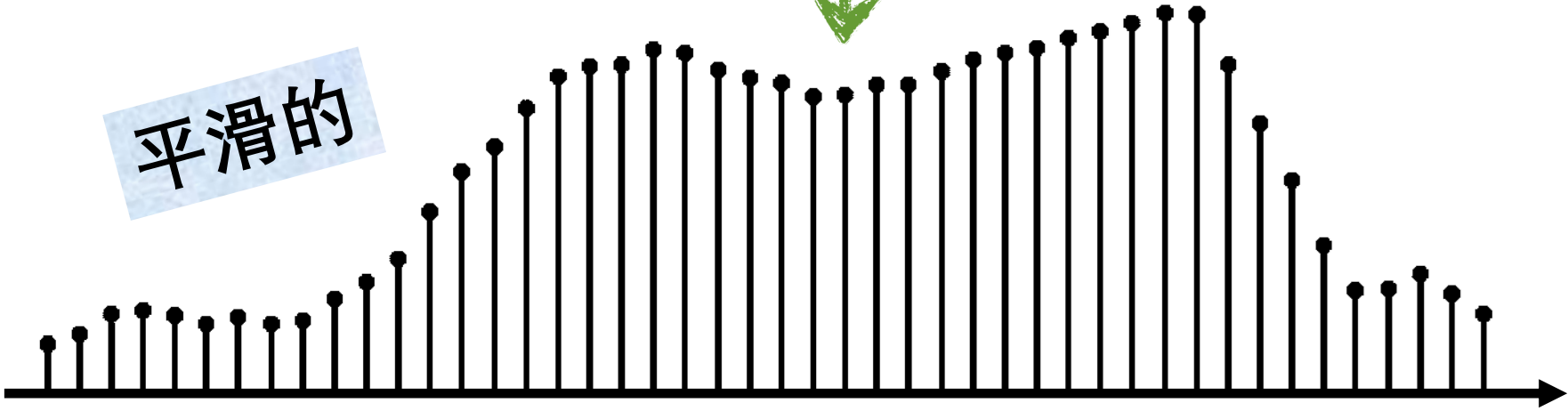
平滑的



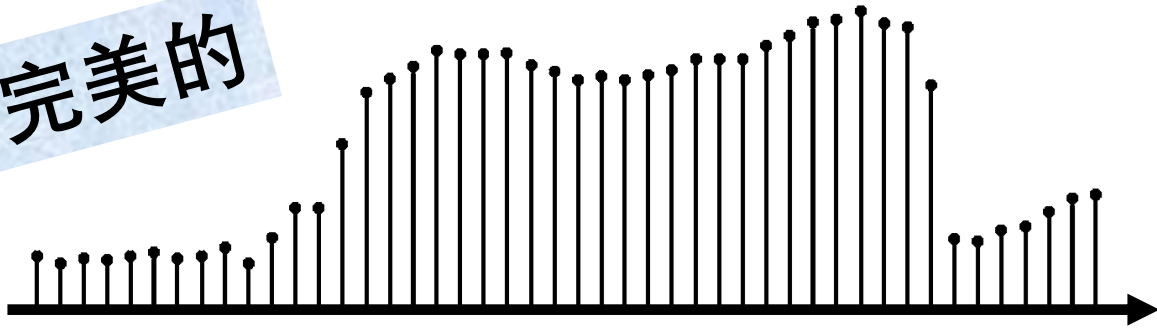
损坏的



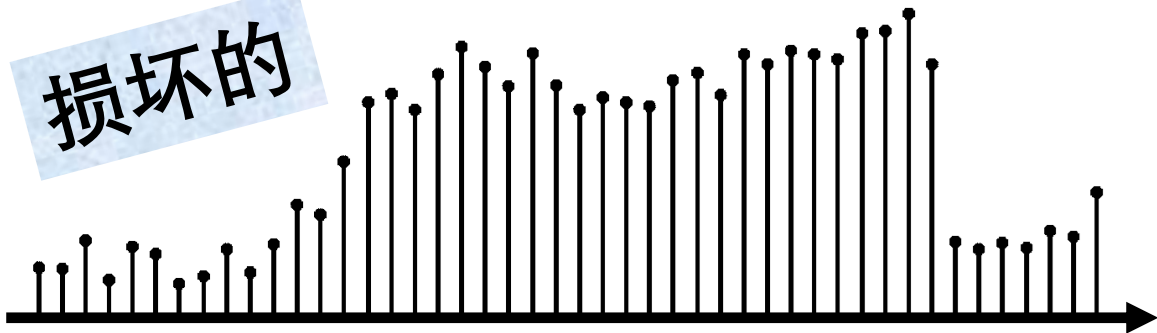
平滑的



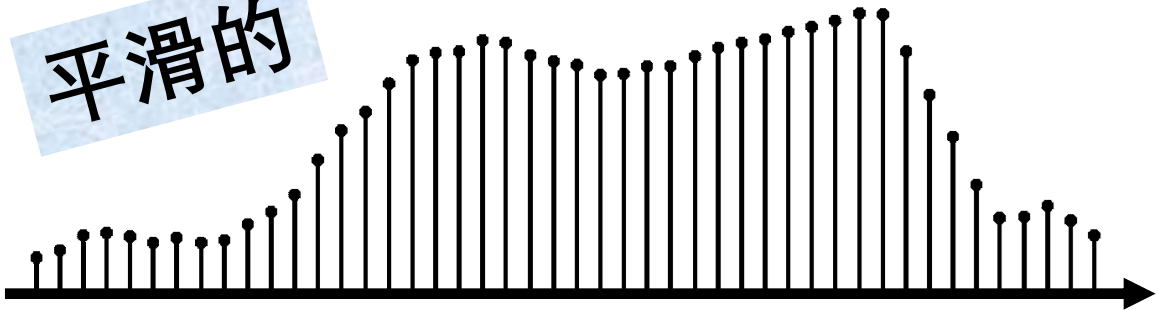
完美的



损坏的



平滑的



2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入



2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入


平滑的

2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入


平滑的

2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入


平滑的

2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入

	0								

平滑的

2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入

	0								

平滑的



2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入

	0	10							

平滑的

2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入

	0	10							

平滑的

2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入

	0	10	20						

平滑的

2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入

	0	10	20						

平滑的

2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入

	0	10	20	30					

平滑的



2D  
滑动平均

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

输入

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

平滑的

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \frac{1}{(2K + 1)^2} \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]$$

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \frac{1}{(2K + 1)^2} \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]$$

输出

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \frac{1}{(2K + 1)^2} \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]$$

输入

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \frac{1}{(2K + 1)^2} \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]$$

在邻域内像素中循环



令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \frac{1}{(2K + 1)^2} \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]$$

权重

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \frac{1}{(2K + 1)^2} \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]$$

如何根据像素在邻域内的位置来  
设置不同的权重？

令滑动平均的窗口大小为 $(2K + 1) \times (2K + 1)$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]G[u, v]$$

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v] G[u, v]$$

掩膜 (mask)、  
核函数 (kernel)  
或滤波器 (filter)

令滑动平均的窗口大小为 $(2K + 1) \times (2K + 1)$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]G[u, v]$$

该式叫作**互相关**，记作 $H = F \otimes G$

令滑动平均的窗口大小为 $(2K + 1) \times (2K + 1)$

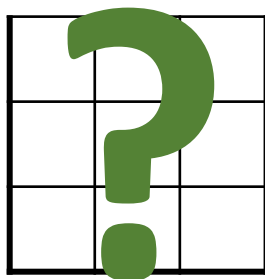
$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]G[u, v]$$

该式叫作**互相关**，记作 $H = F \otimes G$

将每个像素替换成其相邻像素的线性组合



# 均值滤波



$G[u, v]$



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

# 均值滤波

$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$G[u, v]$

$\otimes$

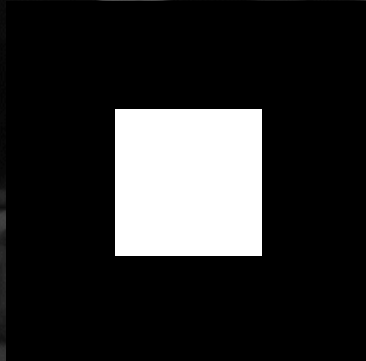
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

原始图像



原始图像



方框滤波器

# 平滑图像



$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$G[u, v]$



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$



$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

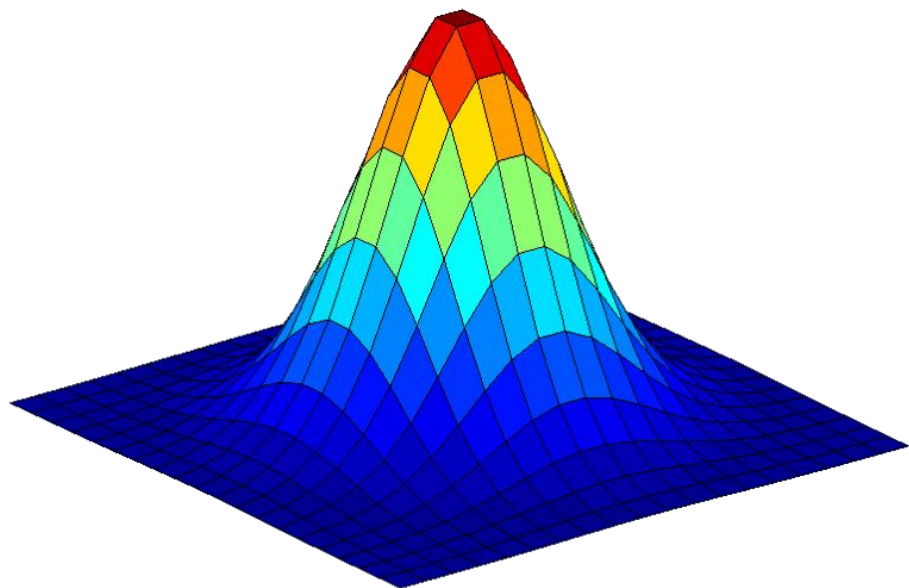
# 高斯滤波

$$\frac{1}{16}$$

1	2	1
2	4	2
1	2	1

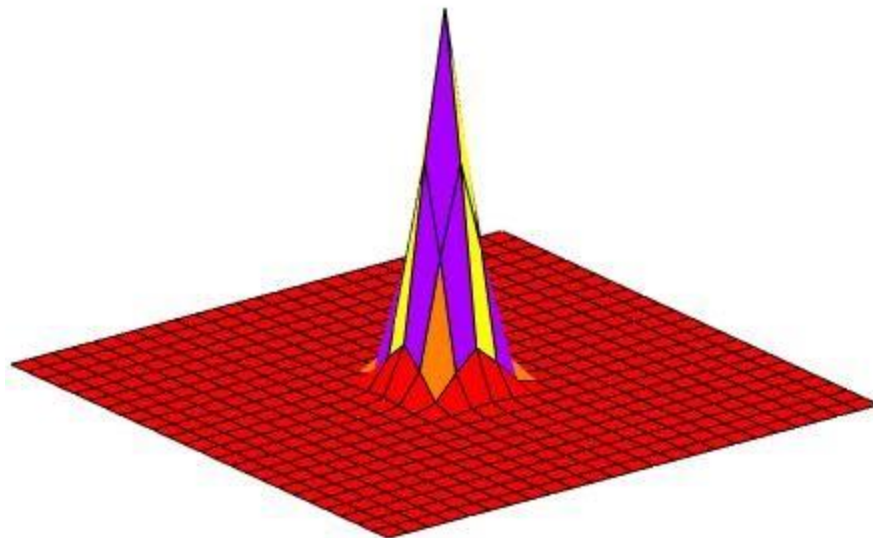
$G[u, v]$

$\approx$



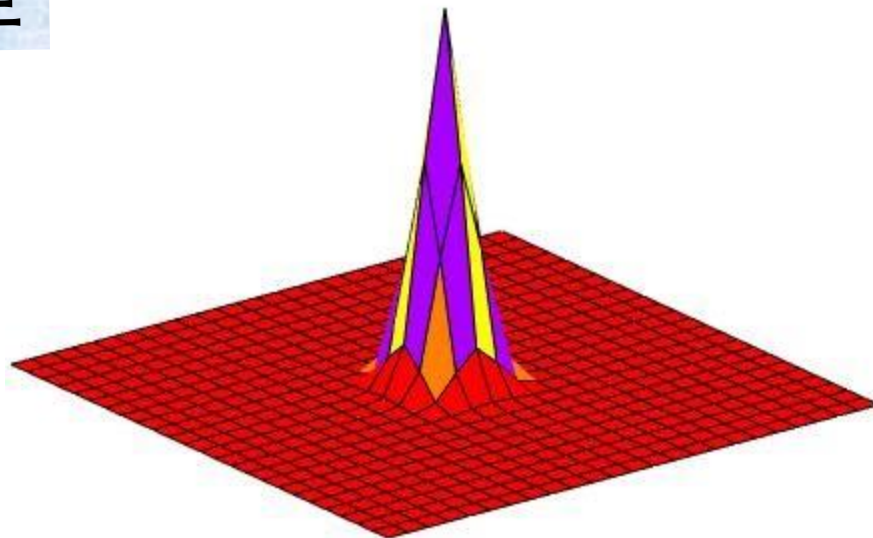
$$\frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

$$\frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$



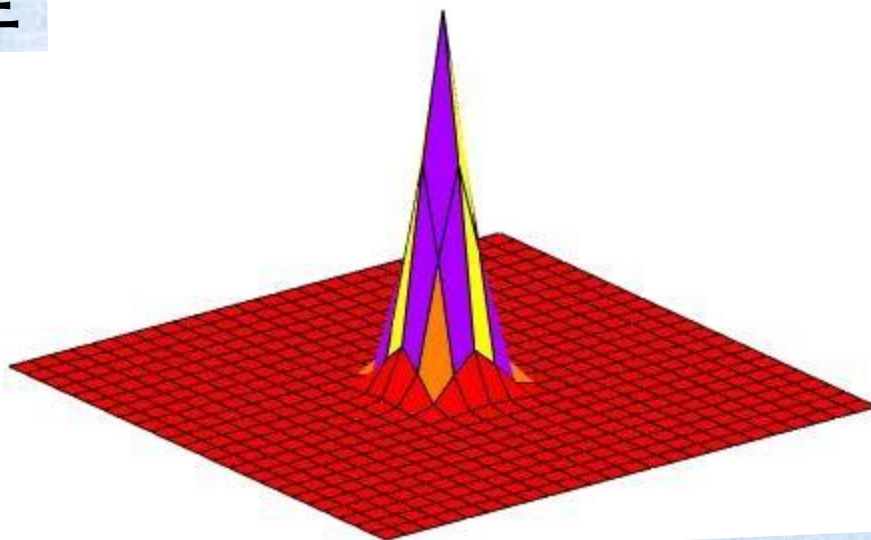
$$\frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

标准差



$$\frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

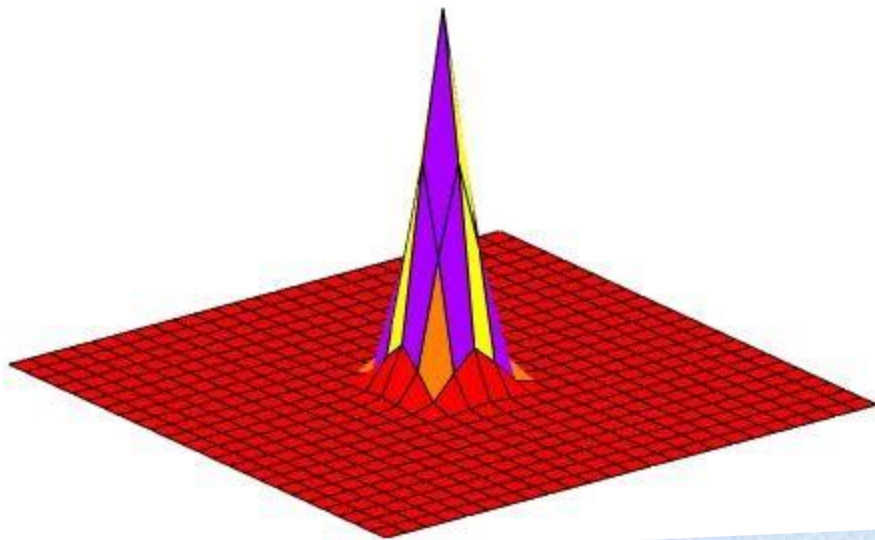
标准差



高斯函数是如何随 $\sigma$ 变化的？

# 归一化

$$\frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

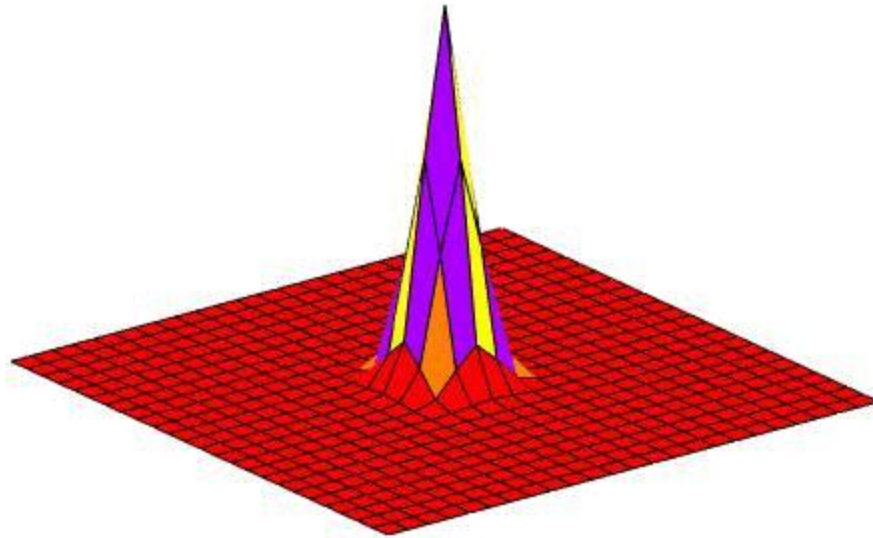


高斯函数是如何随 $\sigma$ 变化的？



$$\frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

$\sigma = 1$

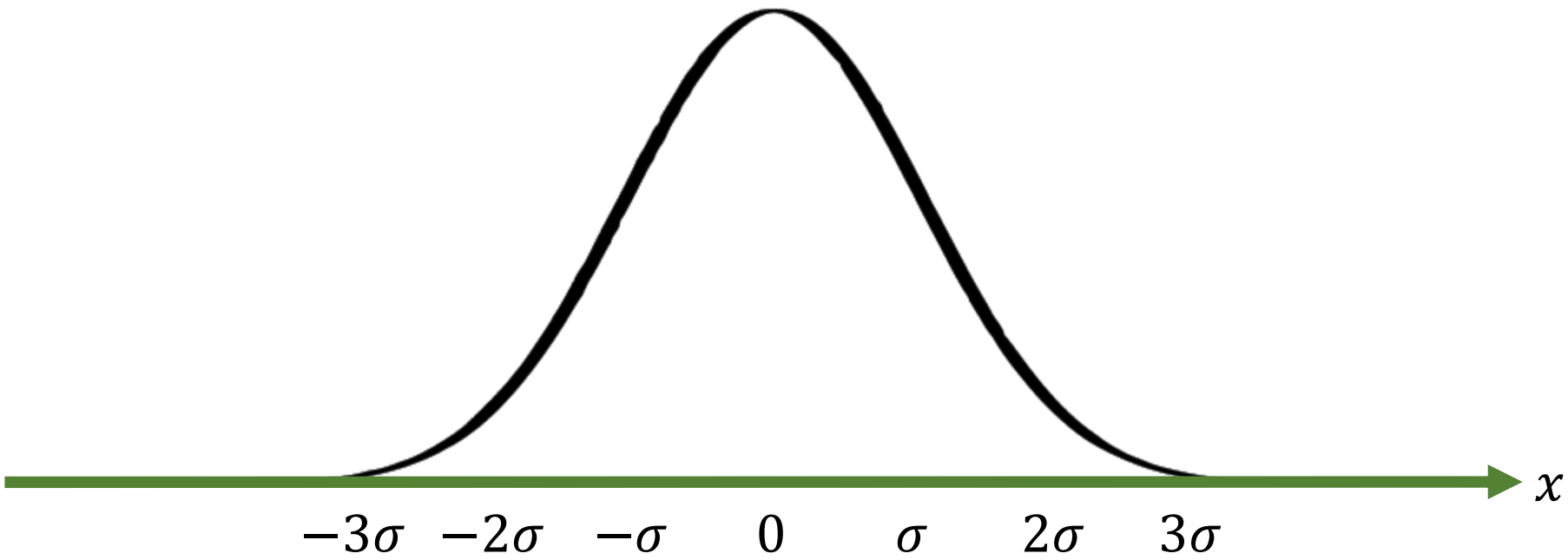


高斯滤波器有**无限长的支撑**

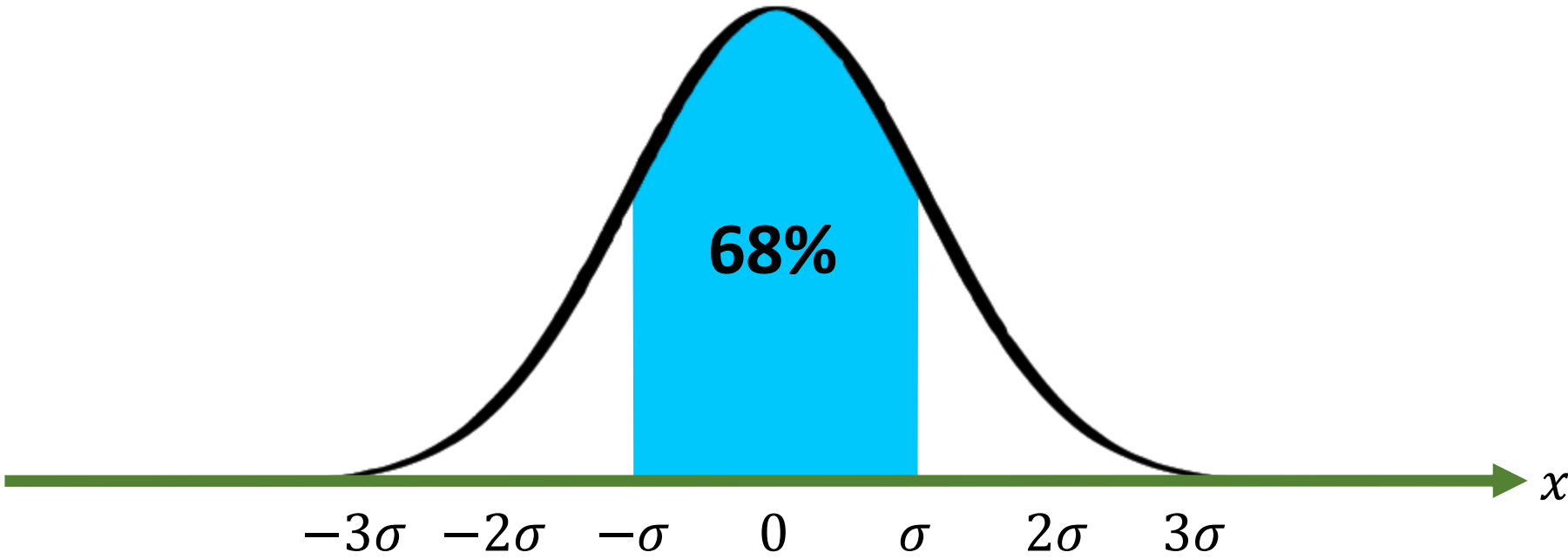
高斯滤波器有**无限长的支撑**

离散滤波器使用**有限大小的核函数**

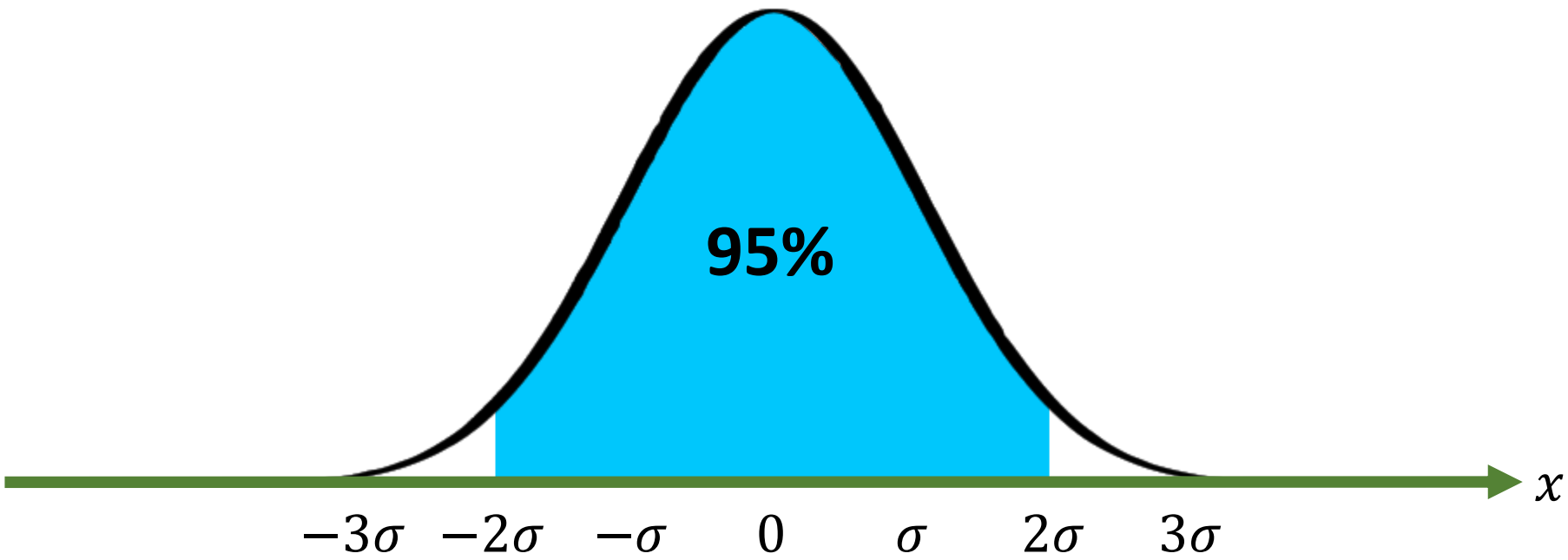
3- $\sigma$ 法则



3- $\sigma$ 法则

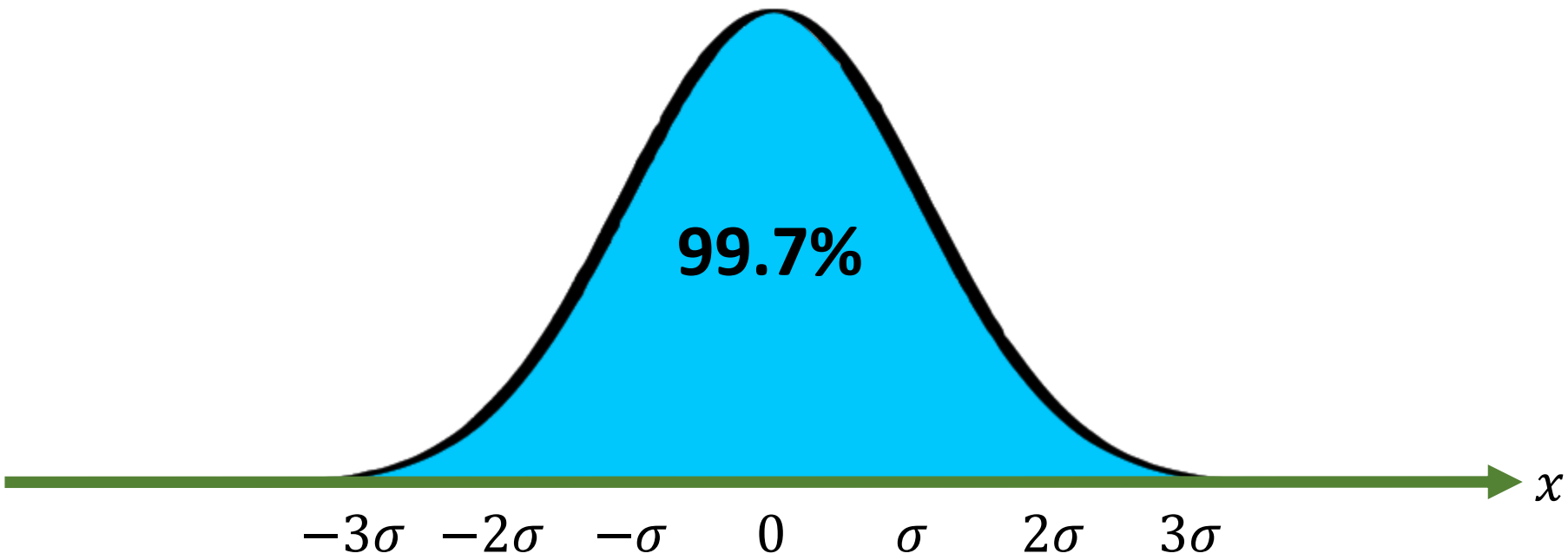


3- $\sigma$ 法则





3- $\sigma$ 法则



原始图像



原始图像



高斯滤波器

# 平滑图像



Python时间





```
>> sigma = 16
```

```
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
```

```
>> I = im + noise
```

```
>> I = (I - I.min()) / (I.max() - I.min())
```

```
>> cv2.imshow('noised', I)
```

```
>> cv2.waitKey(0)
```





```
>> sigma = 16
```

```
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
```

```
>> I = im + noise
```

```
>> I = (I - I.min()) / (I.max() - I.min())
```

```
>> cv2.imshow('noised', I)
```

```
>> cv2.waitKey(0)
```



```
>> sigma = 16
```

```
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
```

```
>> I = im + noise
```

```
>> I = (I - I.min()) / (I.max() - I.min())
```

```
>> cv2.imshow('noised', I)
```

```
>> cv2.waitKey(0)
```



```
>> sigma = 16
```

```
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
```

```
>> I = im + noise
```

```
>> I = (I - I.min()) / (I.max() - I.min())
```

```
>> cv2.imshow('noised', I)
```

```
>> cv2.waitKey(0)
```



```
>> sigma = 16
```

```
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
```

```
>> I = im + noise
```

```
>> I = (I - I.min()) / (I.max() - I.min())
```

```
>> cv2.imshow('noised', I)
```

```
>> cv2.waitKey(0)
```





```
>> sigma = 16
```

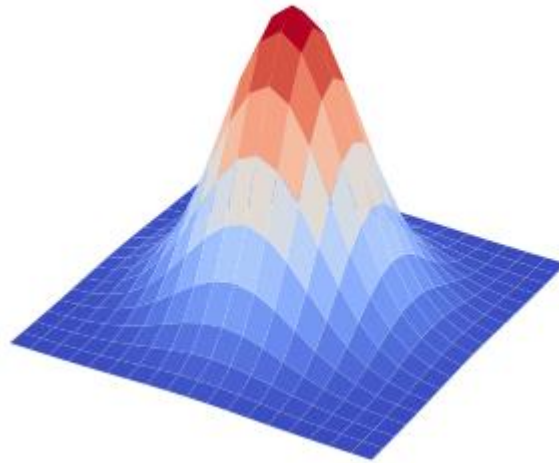
```
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
```

```
>> I = im + noise
```

```
>> I = (I - I.min()) / (I.max() - I.min())
```

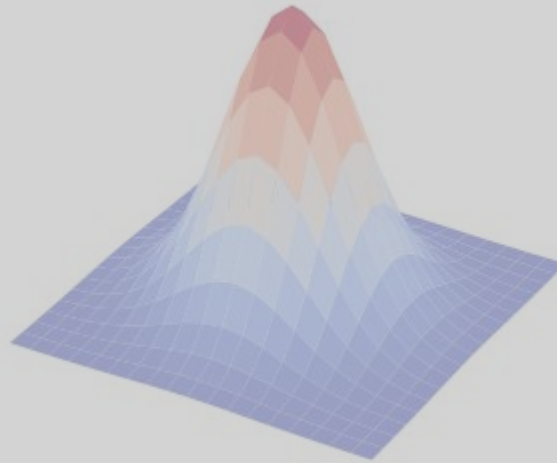
```
>> cv2.imshow('noised', I)
```

```
>> cv2.waitKey(0)
```

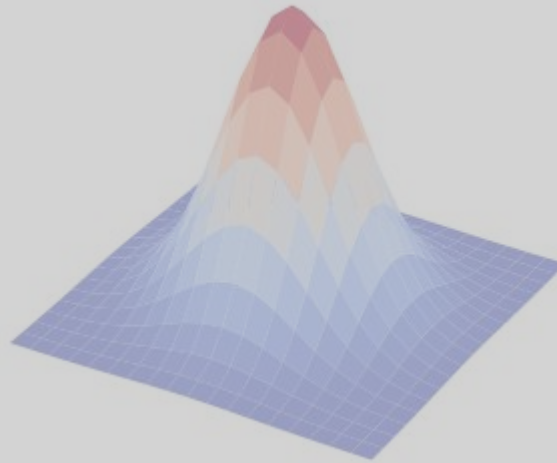


```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```

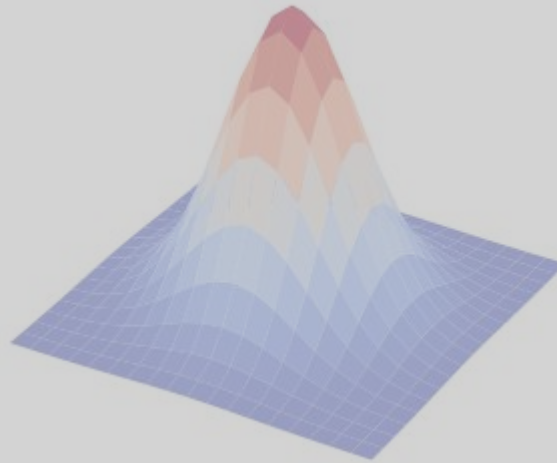




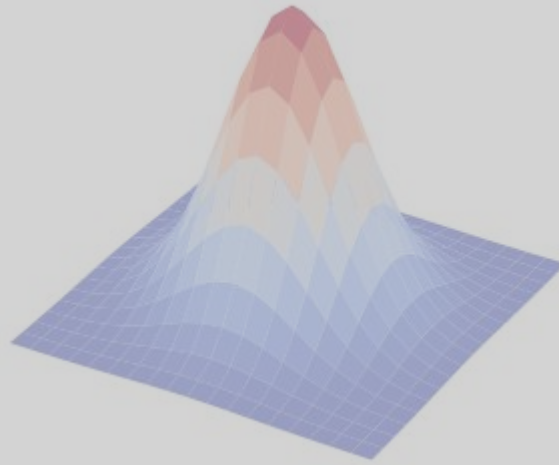
```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```



```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```

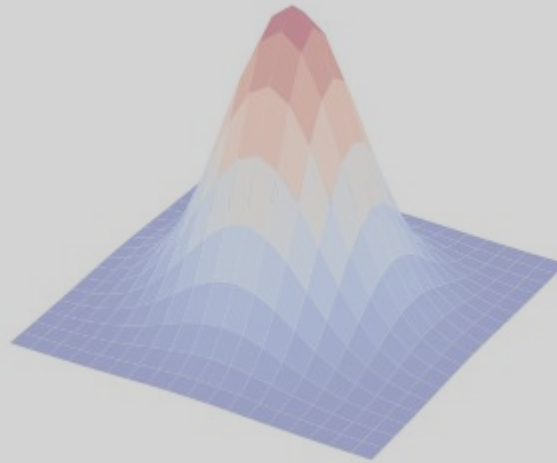


```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```



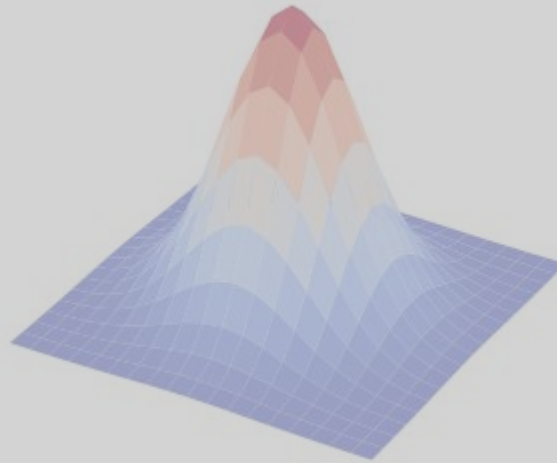
```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 10, 2
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```



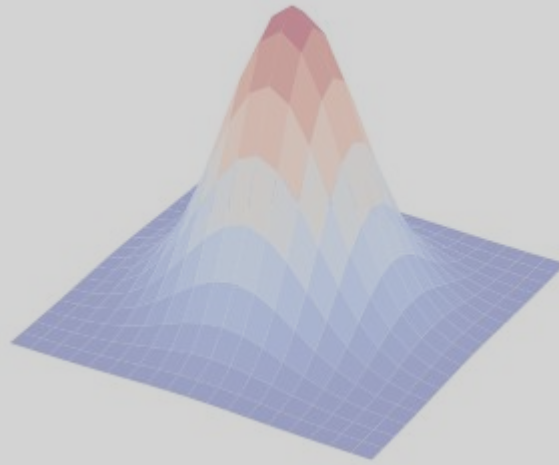


```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```



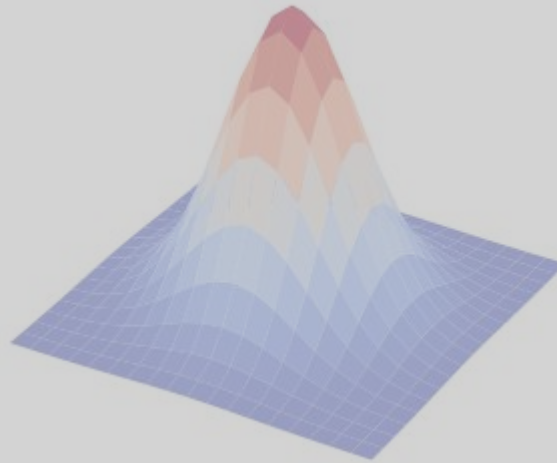


```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```

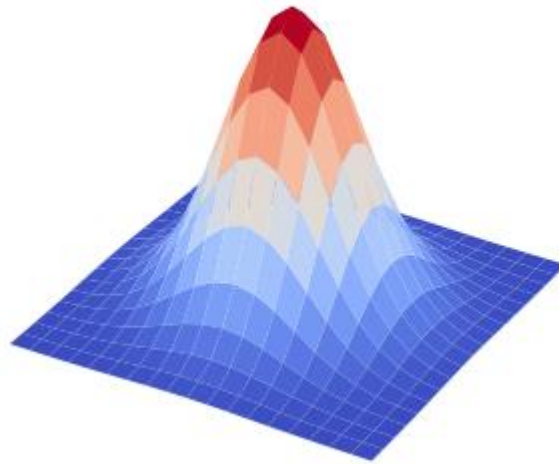


```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(10, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```

向下取整除



```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```



```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)
>>> ax.set_axis_off()
>>> pyplot.show()
```



```
>>> im = cv2.imread('Avengers.png', cv2.IMREAD_GRAYSCALE)
>>> I = cv2.filter2D(im, -1, g, cv2.BORDER_REFLECT)
>>> cv2.imshow('Avengers', I)
>>> cv2.waitKey(0)
```



```
>>> im = cv2.imread('Avengers.png', cv2.IMREAD_GRAYSCALE)
>>> I = cv2.filter2D(im, -1, g, cv2.BORDER_REFLECT)
>>> cv2.imshow('Avengers', I)
>>> cv2.waitKey(0)
```





```
>>> im = cv2.imread('Avengers.png', cv2.IMREAD_GRAYSCALE)
>>> I = cv2.filter2D(im, -1, g, cv2.BORDER_REFLECT)
>>> cv2.imshow('Avengers', I)
>>> cv2.waitKey(0)
```



**输出数据类型与输入一致**

```
>>> im = cv2.imread('Avengers.jpg', cv2.IMREAD_GRAYSCALE)
>>> I = cv2.filter2D(im, -1, g, cv2.BORDER_REFLECT)
>>> cv2.imshow('Avengers', I)
>>> cv2.waitKey(0)
```



```
>>> im = cv2.imread('Avengers.png', cv2.IMREAD_GRAYSCALE)
>>> I = cv2.filter2D(im, -1, g, cv2.BORDER_REFLECT)
>>> cv2.imshow('Avengers', I)
>>> cv2.waitKey(0)
```



```
>>> im = cv2.imread('Avengers.png', cv2.IMREAD_GRAYSCALE)
>>> I = cv2.filter2D(im, -1, g, cv2.BORDER_REFLECT)
>>> cv2.imshow('Avengers', I)
>>> cv2.waitKey(0)
```





靠近图像边缘时怎么办？



边界问题

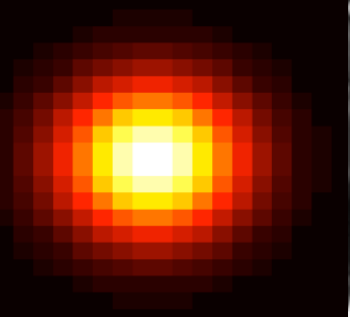




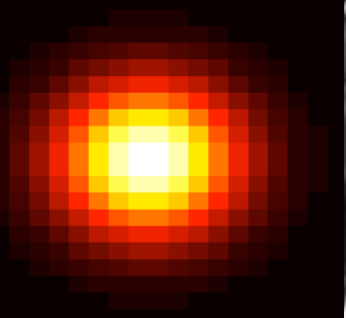
# 边界问题



边界问题



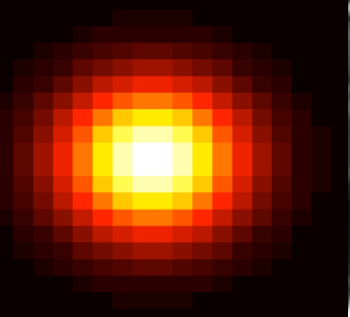
边界问题



裁剪滤波



边界问题

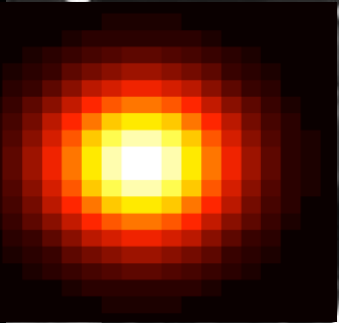


裁剪滤波

`cv.BORDER_CONSTANT`

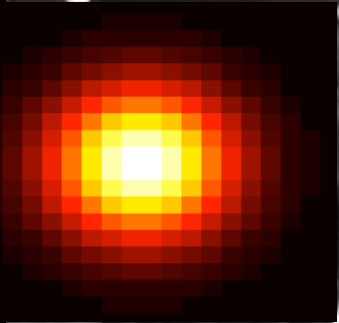


边界问题



循环滤波

边界问题

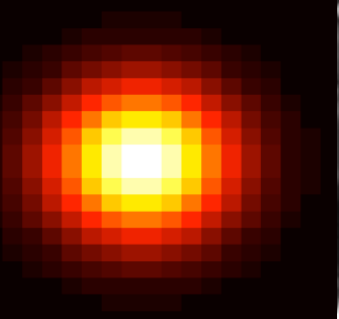


循环滤波

cv.BORDER\_WARP

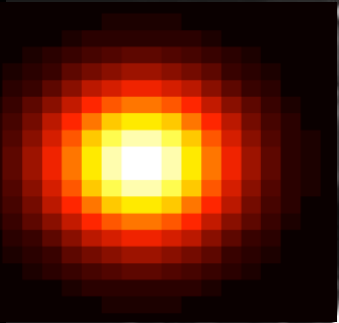


边界问题



复制滤波

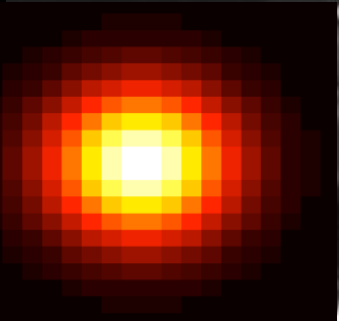
边界问题



复制滤波

`cv2.BORDER_REPLICATE`

边界问题



对称滤波

边界问题



对称滤波

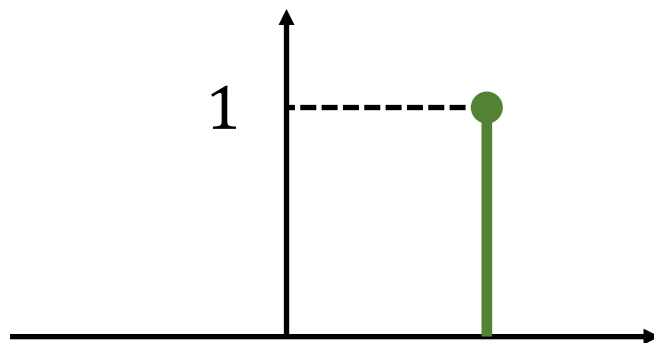
cv2.BORDER\_REFLECT



# Python时间



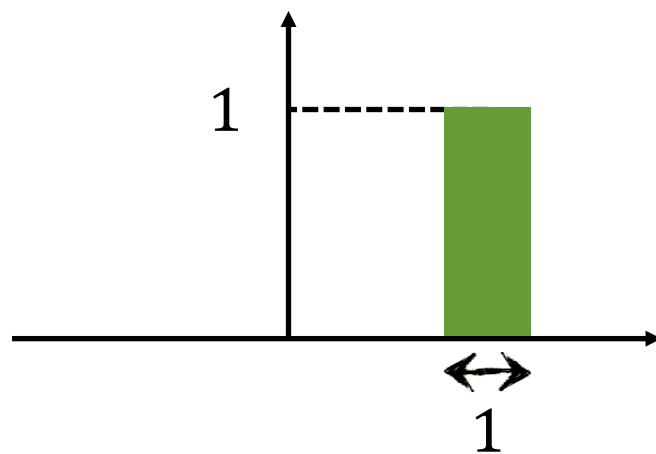
# 离散 冲击响应



在单个位置上值为1的函数

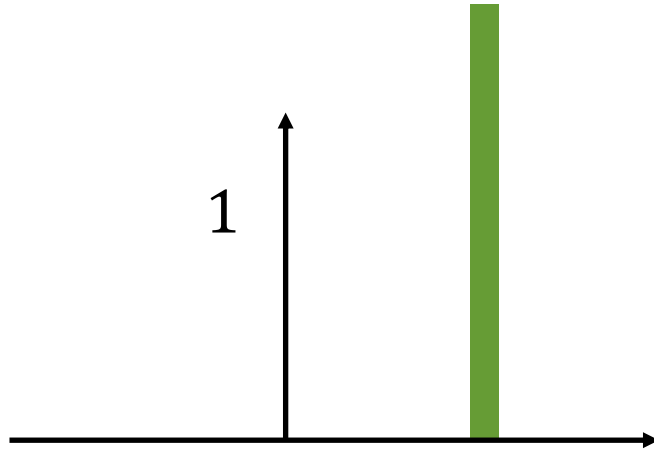
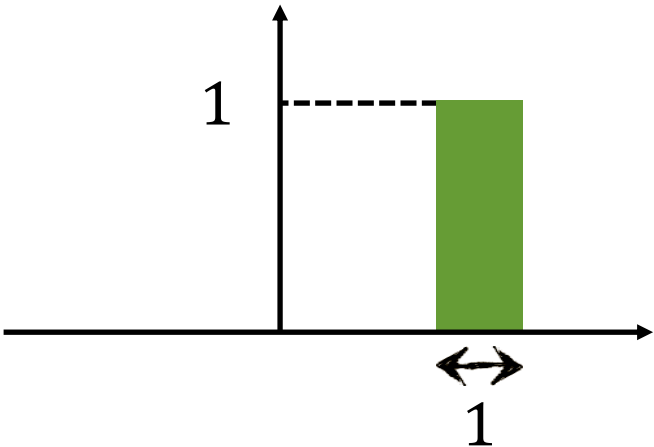


离散  
冲击响应



在单个位置上值为1的函数

连续  
冲击响应



一个非常窄和非常高的函数，在极限处有一个单位面积

对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=


$H[x, y]$



对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0						

$H[x, y]$

对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0					

$H[x, y]$

对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i				

$H[x, y]$



对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i				

$H[x, y]$

对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i	h			

$H[x, y]$

对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i	h			

$H[x, y]$

对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i	h	g		

$H[x, y]$

对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i	h	g	0	0
0	0					

$H[x, y]$

对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i	h	g	0	0
0	0	f				

$H[x, y]$



对冲冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i	h	g	0	0
0	0	f	e	d	0	0
0	0	c	b	a	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$H[x, y]$

对冲击响应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i	h	g	0	0
0	0	f	e	d	0	0
0	0	c	b	a	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$H[x, y]$

滤波后输出反转了!

对冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=


$H[x, y]$

如何避免输出反转呢？

对冲击响  
应滤波

a	b	c
d	e	f
g	h	i

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=


$H[x, y]$

如何避免输出反转呢？

反转滤波器

对冲击响  
应滤波

c	b	a
f	e	d
i	h	g

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=


$H[x, y]$

如何避免输出反转呢？

反转滤波器

对冲击响  
应滤波

i	h	g
f	e	d
c	b	a

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=


$H[x, y]$

如何避免输出反转呢？

反转滤波器



对冲击响  
应滤波

i	h	g
f	e	d
c	b	a

$G[u, v]$

$\otimes$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$

=

		a	b	c		
		d	e	f		
		g	h	i		

$H[x, y]$

如何避免输出反转呢？

反转滤波器

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

该式叫作卷积，记作  $H = F * G$

令滑动平均的窗口大小为 $(2K + 1) \times (2K + 1)$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]G[u, v]$$

该式叫作**互相关**，记作 $H = F \otimes G$



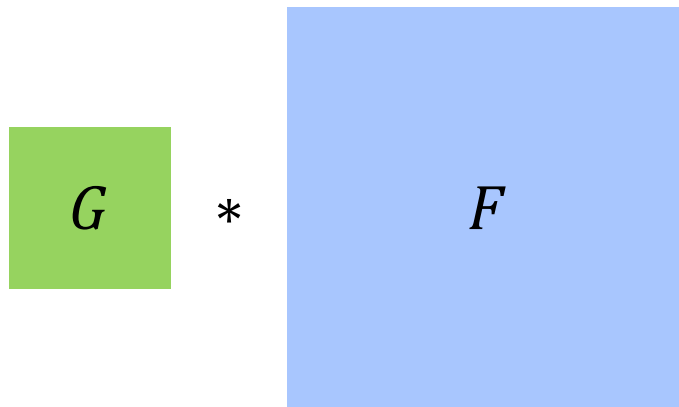
令滑动平均的窗口大小为  $(2K + 1) \times (2K + 1)$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x + u, y + v]G[u, v]$$

该式叫作互相关，记作  $H = F \otimes G$

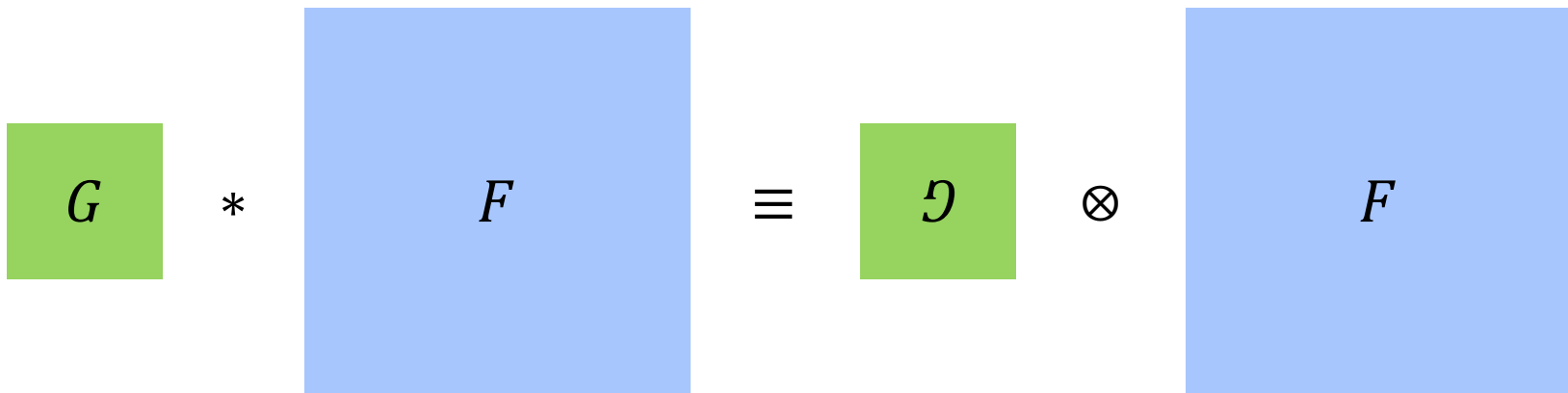
$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

该式叫作卷积，记作 $H = F * G$



$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

该式叫作卷积，记作 $H = F * G$



# 卷积性质

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

$$F[x, y] * \delta[x, y] = F[x, y]$$

# 卷积性质

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

## 线性运算

$$G * (\alpha F_1[x, y] + \beta F_2[x, y]) = \alpha H_1[x, y] + \beta H_2[x, y]$$

# 卷积性质

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

平移不变性

$$G * F[x - \alpha, y - \beta] = H[x - \alpha, y - \beta]$$



# 卷积性质

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

## 分配律

$$G * (E[x, y] + F[x, y]) = (G * E[x, y]) + (G * F[x, y])$$

# 卷积性质

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

## 结合律

$$(E[x, y] * F[x, y]) * G[x, y] = E[x, y] * (F[x, y] * G[x, y])$$

# 卷积性质

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

## 结合律

$$(E[x, y] * F[x, y]) * G[x, y] = E[x, y] * (F[x, y] * G[x, y])$$

依次应用若干个滤波器  
相当于应用一个滤波器

# 卷积性质

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

## 交换律

$$F[x, y] * G[x, y] = G[x, y] * F[x, y]$$

证明略...





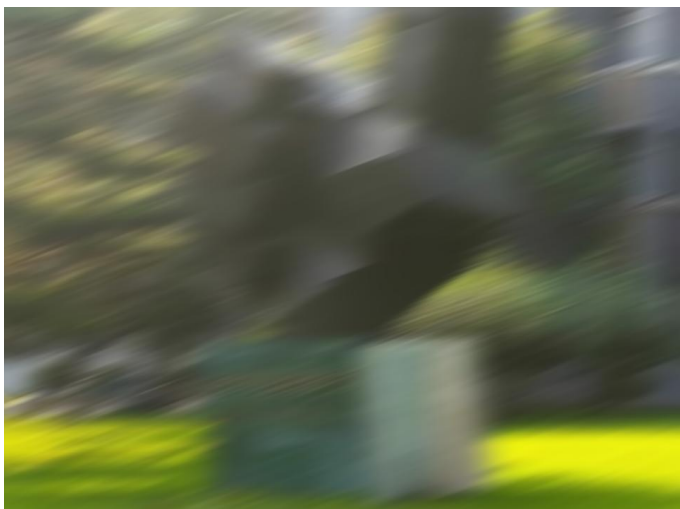


Canon





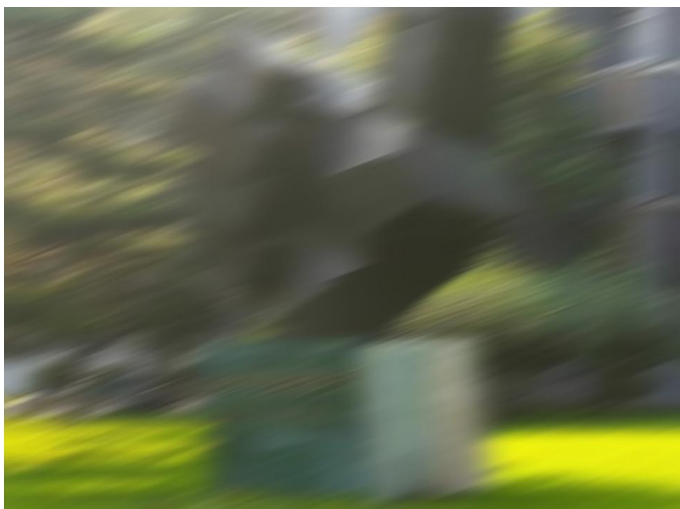
运动模糊



=

模糊图像

运动模糊



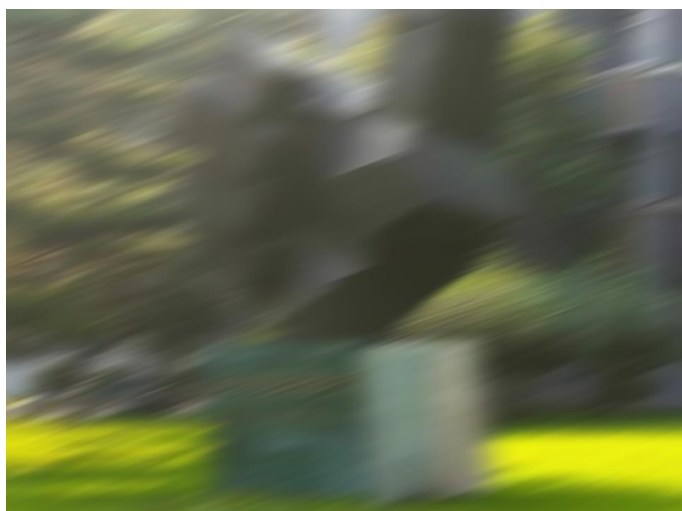
模糊图像

=



完美图像

运动模糊



模糊图像

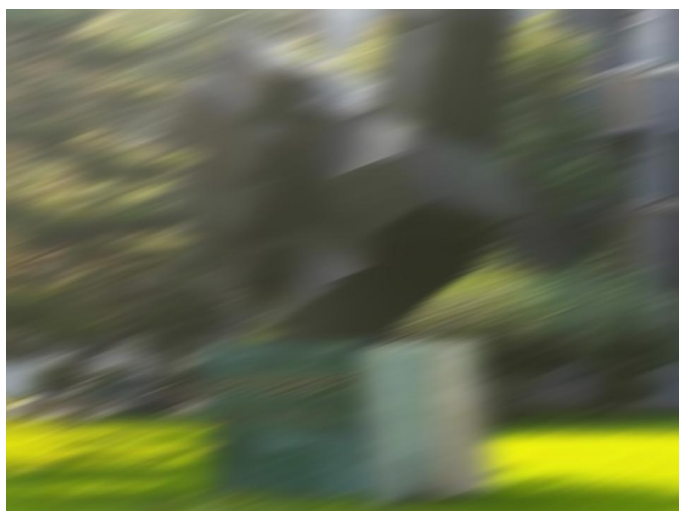
=



完美图像

\*

运动模糊



模糊图像

=



完美图像

\*



核函数

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

卷积每像素需要多少个操作？



$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

卷积每像素需要多少个操作？

$$(2K + 1)^2$$

假设滤波器可以改写成

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

该滤波器称作可分离的

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

该滤波器称作可分离的

可以让卷积**更快!**

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

---

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

---

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$



假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

---

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

---

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

---

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

眼熟吗？

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

1D水平卷积



假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

眼熟吗？

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

**1D垂直卷积**

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

先水平滤波再垂直滤波

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

卷积每像素需要多少个操作？

假设滤波器可以改写成

$$F[x, y] = U[x]V[y]$$

$$H * F = \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]F[x - u, y - v]$$

代入滤波器定义

$$= \sum_{v=-\infty}^{\infty} \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]V[y - v]$$

提出因子

$$= \sum_{v=-\infty}^{\infty} V[y - v] \sum_{u=-\infty}^{\infty} H[u, v]U[x - u]$$

卷积每像素需要多少个操作?

$$2(2K + 1)$$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K F[x - u, y - v]G[u, v]$$

卷积每像素需要多少个操作？

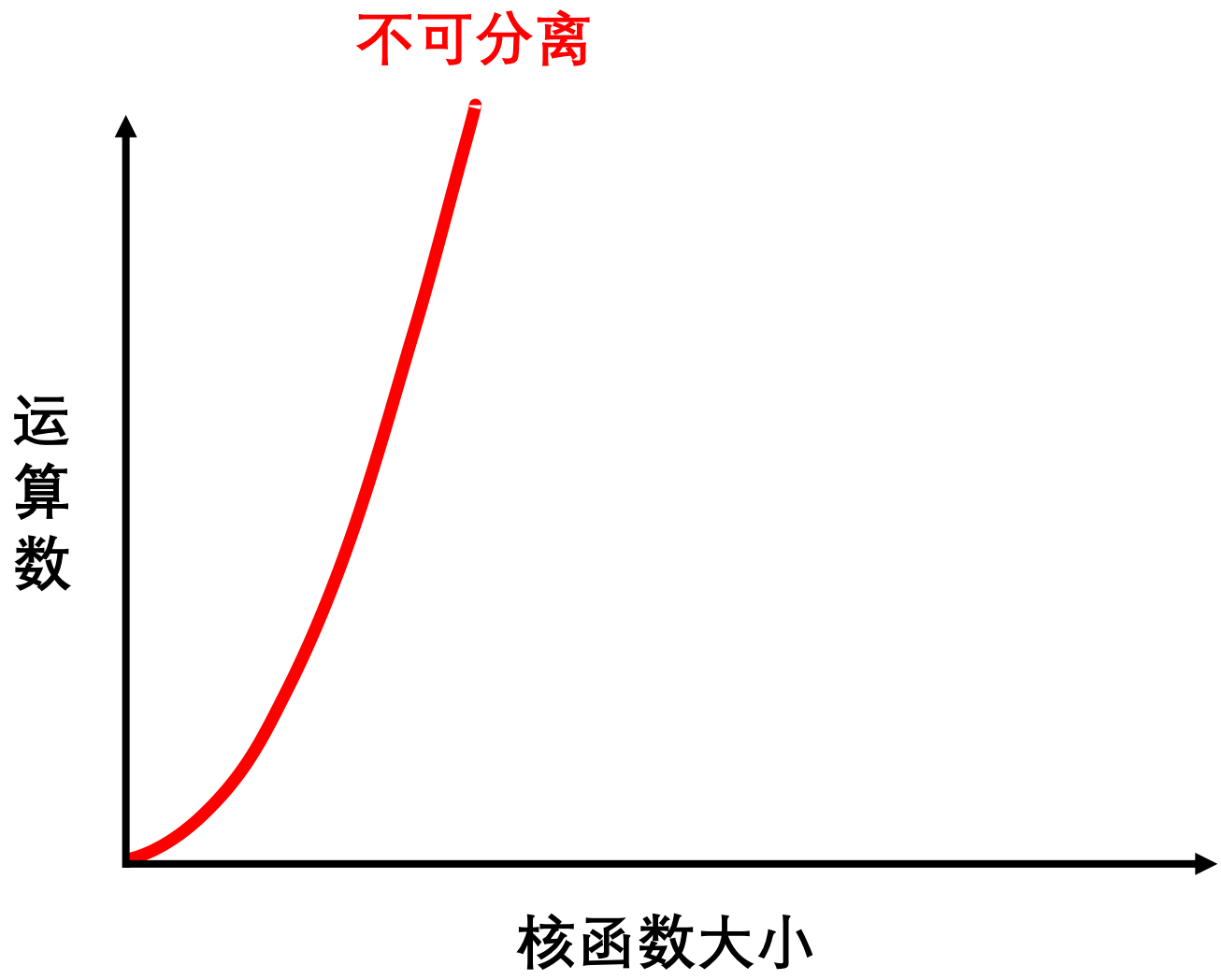
$$(2K + 1)^2$$

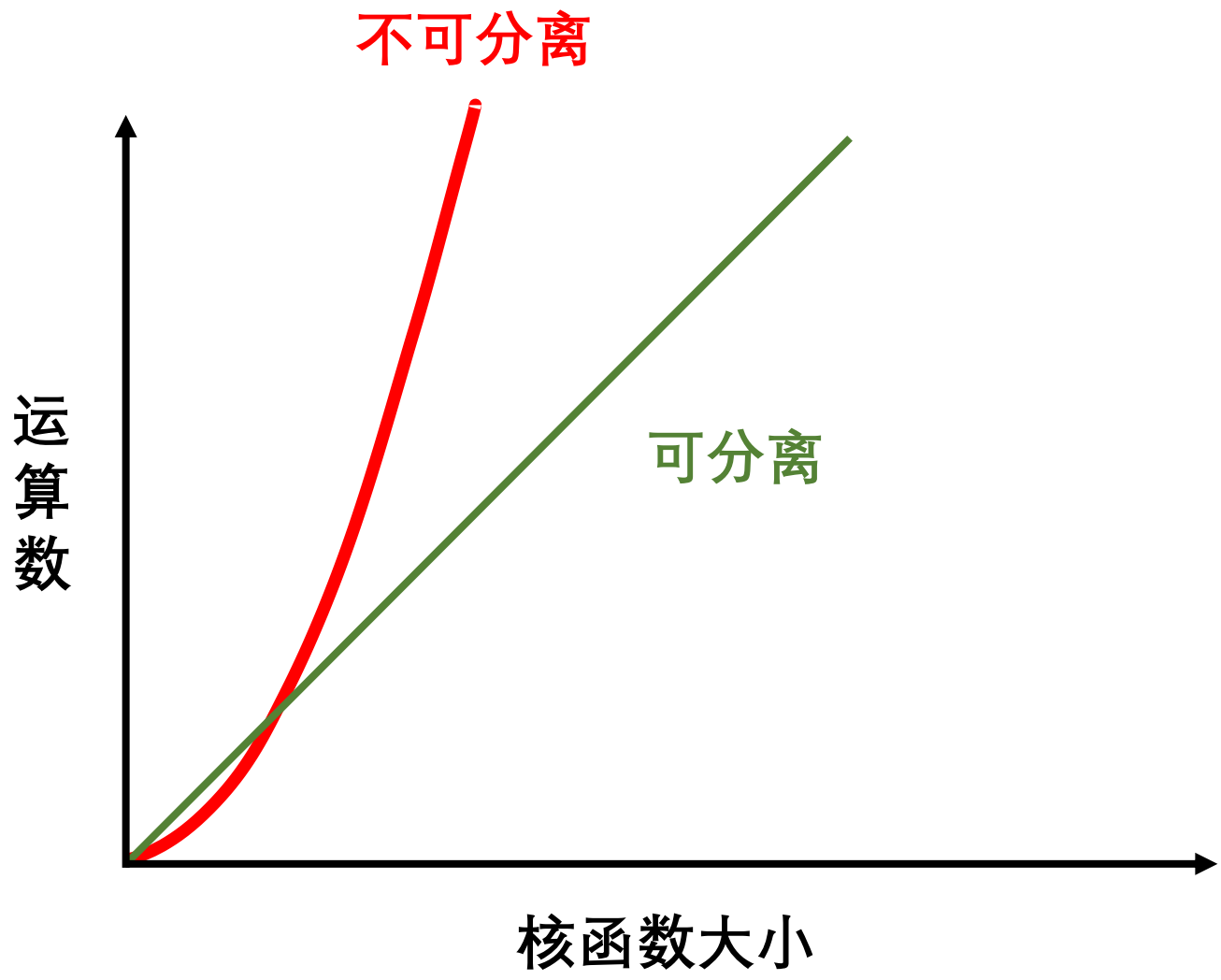


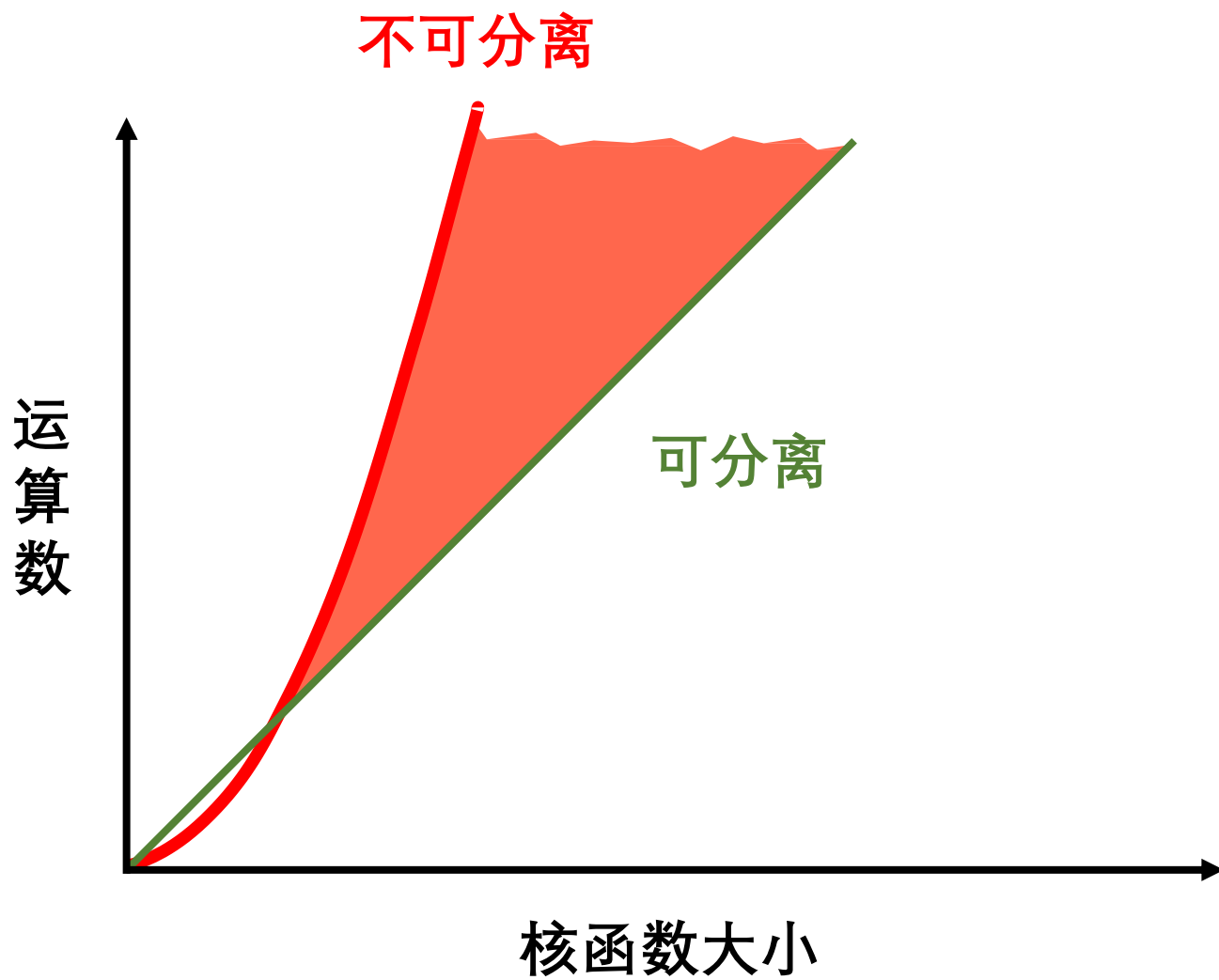


运算数

核函数大小







**证明2D高斯滤波器是可分离的**

**证明2D高斯滤波器是可分离的**

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

证明2D高斯滤波器是可分离的

$$\begin{aligned} G(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \\ &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \exp\left(-\frac{y^2}{2\sigma^2}\right) \end{aligned}$$



## 证明2D高斯滤波器是可分离的

$$\begin{aligned} G(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \\ &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= \left[ \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \right] \left[ \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{y^2}{2\sigma^2}\right) \right] \end{aligned}$$

## 证明2D高斯滤波器是可分离的

$$\begin{aligned} G(x, y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \\ &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right) \exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= \left[ \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right) \right] \left[ \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{y^2}{2\sigma^2}\right) \right] \\ &= G_1(x)G_1(y) \end{aligned}$$

回顾

互相关  $H = G \otimes F$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K G[u, v] F[x + u, y + v]$$

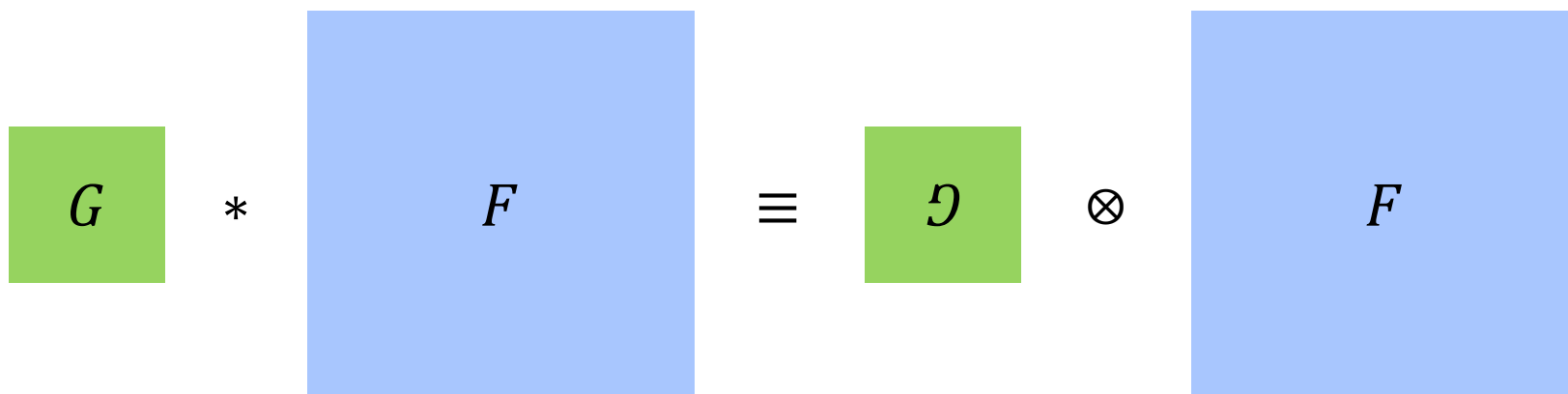
卷积  $H = G * F$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K G[u, v] F[x - u, y - v]$$

回顾

互相关  $H = G \otimes F$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K G[u, v] F[x + u, y + v]$$



卷积  $H = G * F$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K G[u, v] F[x - u, y - v]$$

回顾

互相关  $H = G \otimes F$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K G[u, v] F[x + u, y + v]$$

对于对称滤波器，输出有什么不同？

卷积  $H = G * F$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K G[u, v] F[x - u, y - v]$$



回顾

互相关  $H = G \otimes F$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K G[u, v] F[x + u, y + v]$$

对于对称滤波器，输出有什么不同？

没区别

卷积  $H = G * F$

$$H[x, y] = \sum_{u=-K}^K \sum_{v=-K}^K G[u, v] F[x - u, y - v]$$





实践

用例



图像锐化



输入

图像锐化



输入

图像锐化



输入



模糊的

图像锐化



输入



模糊的

图像锐化



输入

-



模糊的

=

# 图像锐化



输入

—



模糊的

=



“锐利的东西”

图像锐化



输入

=



模糊的

+



“锐利的东西”



# 图像锐化



模糊的

+



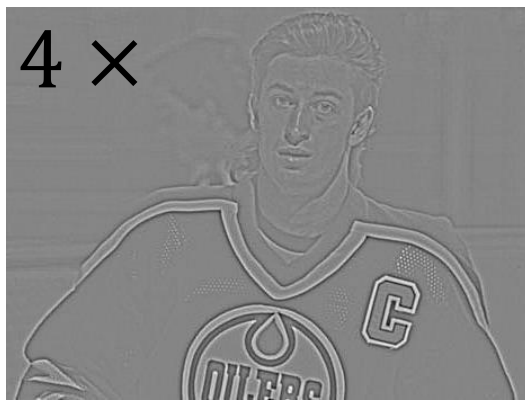
“锐利的东西”

图像锐化



模糊的

+



“锐利的东西”

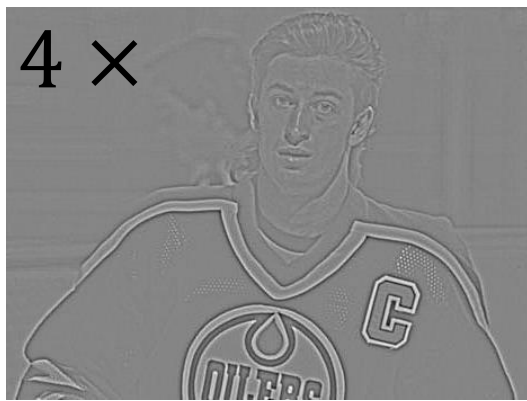
=

# 图像锐化



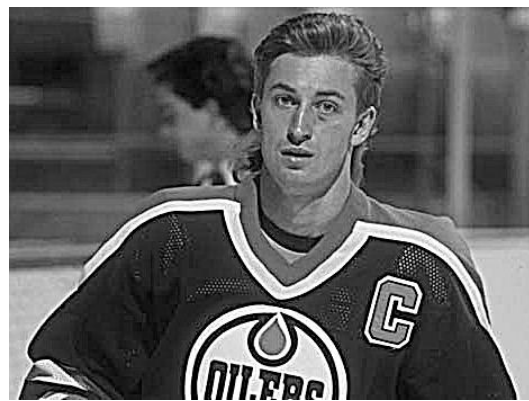
模糊的

+



“锐利的东西”

=



锐化的

图像锐化



输入



锐化的

# 预测

滤波器输出





输入

\*

0	0	0
0	1	0
0	0	0

=





输入

\*

0	0	0
0	1	0
0	0	0

=



没变化





中國傳媒大學  
COMMUNICATION UNIVERSITY OF CHINA

輸入

\*

0	0	0
0	0	1
0	0	0

=





输入

\*

0	0	0
0	0	1
0	0	0

=



向右平移1像素



输入

$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$





输入

$$* \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} =$$



模糊了

Image kernels explained v1 x  
setosa.io/ev/image-kernels/

[Back](#)

# Image kernels

Explained Visually


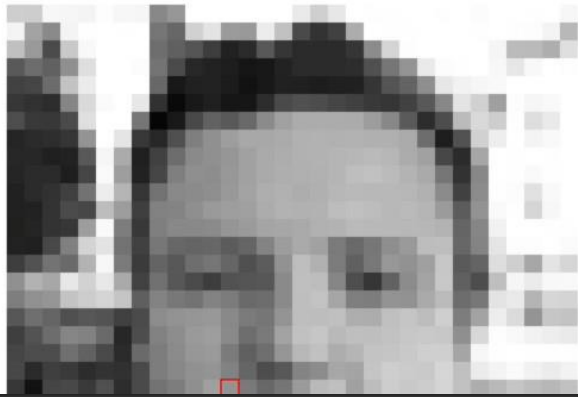
[Tweet](#) 137 [Like](#) [Share](#) 111

By [Victor Powell](#)

An image kernel is a small matrix used to apply effects like the ones you might find in Photoshop or Gimp, such as blurring, sharpening, outlining or embossing. They're also used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context the process is referred to more generally as "convolution" (see: [convolutional neural networks](#).)

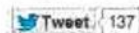
To see how they work, let's start by inspecting a black and white image. The matrix on the left contains numbers, between 0 and 255, which each correspond to the brightness of one pixel in a picture of a face. The large, granulated picture has been blown up to make it easier to see; the last image is the "real" size.

206	205	247	245	244	253	247	245	126	151	255	255	255	255	255	255	254	207	231	255	254	254	255	255	254	255	252	215	255	254	255	247
244	182	138	244	254	255	254	255	118	103	209	238	155	155	236	183	74	52	66	173	255	254	254	255	255	254	254	255	254	255	254	184
182	154	76	201	249	255	255	255	110	86	61	61	35	44	80	53	44	43	43	34	140	213	255	255	255	255	245	187	186	175	223	
90	109	87	144	223	255	255	252	117	75	41	35	31	24	25	36	45	44	46	81	116	148	234	252	254	253	248	231	248	255	254	
67	69	108	186	236	255	255	255	104	25	34	35	29	20	25	34	30	30	32	34	33	85	100	142	251	242	247	249	255	255	255	255
55	51	46	135	218	251	255	252	51	12	26	33	24	24	46	74	81	77	70	85	58	53	67	90	136	228	259	180	253	246	249	255
79	58	56	78	224	255	255	158	11	27	73	98	98	105	139	161	172	172	172	171	157	136	91	48	78	187	217	236	254	232	233	255
38	43	47	53	148	255	229	56	41	80	128	144	159	168	168	171	177	178	177	178	176	176	171	108	31	82	259	238	254	249	255	255
40	44	44	31	60	245	171	30	85	109	128	144	150	161	170	173	177	176	181	183	186	182	172	161	31	45	186	215	254	255	254	255
37	44	44	31	68	250	158	36	70	128	142	141	152	161	170	174	176	177	181	180	183	187	179	169	51	136	215	254	250	254	255	255
34	45	51	64	116	237	181	53	115	137	139	142	153	163	175	177	173	176	182	185	184	184	182	177	139	66	140	254	252	255	249	255
34	36	52	75	71	188	138	83	130	123	143	154	159	160	172	176	177	178	188	182	189	184	186	181	155	80	147	250	254	214	247	255
32	38	52	55	159	250	126	57	128	127	137	139	150	155	165	167	170	177	179	186	185	184	184	182	179	101	133	242	255	255	254	254
36	32	72	129	212	228	115	85	120	103	101	103	95	102	123	157	169	161	124	107	120	142	154	188	190	153	133	229	253	253	255	251
61	82	118	107	179	247	124	60	150	88	110	118	102	80	93	148	190	177	125	97	122	132	148	160	199	91	98	221	207	187	227	215
144	178	187	231	210	232	170	67	114	87	75	82	82	84	87	138	181	189	134	79	53	98	140	164	200	96	78	191	245	235	248	249
127	145	150	198	205	213	197	95	132	121	116	132	125	107	109	138	190	195	186	128	147	146	170	187	159	120	227	233	180	215	212	
87	112	101	80	87	65	75	641	147	150	152	137	124	119	148	190	188	182	174	173	182	187	188	207	126	162	238	219	148	188	195	
83	83	109	134	131	107	40	78	121	141	154	158	138	110	123	163	193	188	183	191	190	194	189	201	189	142	216	233	249	242	238	234
88	78	78	113	86	74	43	107	128	138	111	154	124	96	111	149	184	183	173	181	183	187	201	207	208	163	247	254	255	254	254	254
72	44	43	59	46	52	49	74	126	138	145	148	131	102	77	88	133	143	167	164	187	206	203	202	215	182	238	244	251	242	236	243
85	20	89	75	59	80	48	74	118	138	143	161	147	123	104	119	155	186	182	181	188	208	208	204	215	193	174	185	187	188	188	183



# Image kernels

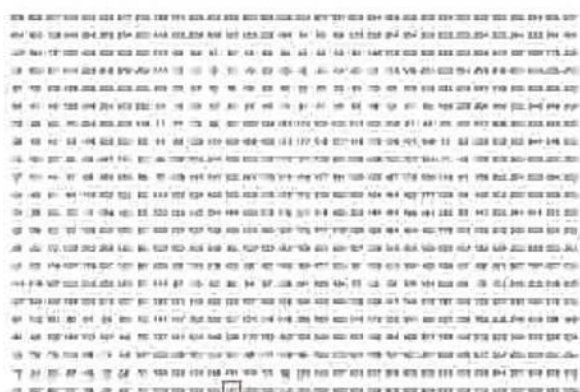
Explained Visually



By [Victor Powell](#)

An image kernel is a small matrix used to apply effects like the ones you might find in Photoshop or Gimp, such as blurring, sharpening, outlining or embossing. They're also used in machine learning for 'feature extraction', a technique for determining the most important portions of an image. In this context the process is referred to more generally as "convolution" (see: [convolutional neural networks](#)).

To see how they work, let's start by inspecting a black and white image. The matrix on the left contains numbers, between 0 and 255, which each correspond to the brightness of one pixel in a picture of a face. The large, granulated picture has been blown up to make it easier to see; the last image is the "real" size.







# 非线性滤波

椒盐噪声





椒盐噪声



10	15	20
23	90	27
33	31	30

10	15	20
23	90	27
33	31	30

## 将像素值排序

10 15 20 23 27 30 31 33 90

10	15	20
23	90	27
33	31	30

将像素值排序

10 15 20 23 27 30 31 33 90

用中值替换像素



10	15	20
23	90	27
33	31	30

将像素值排序

10 15 20 23 27 30 31 33 90

用中值替换像素

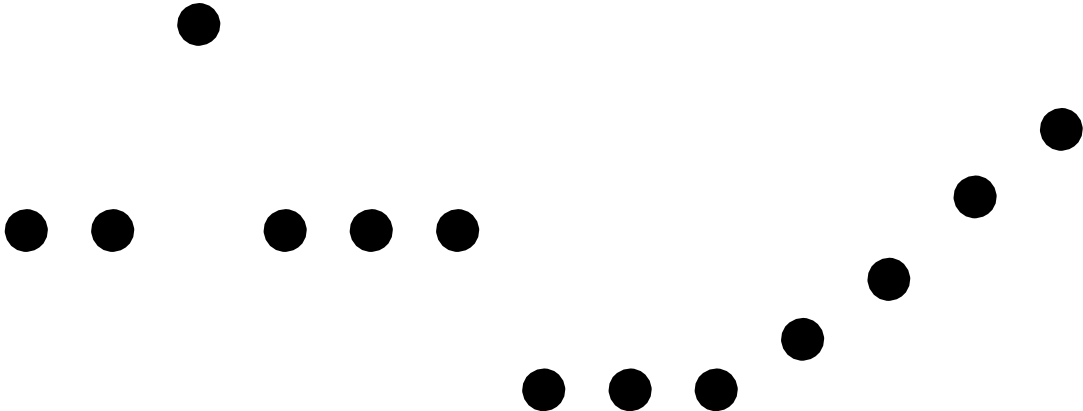


# 中值滤波



# 中值滤波

输入

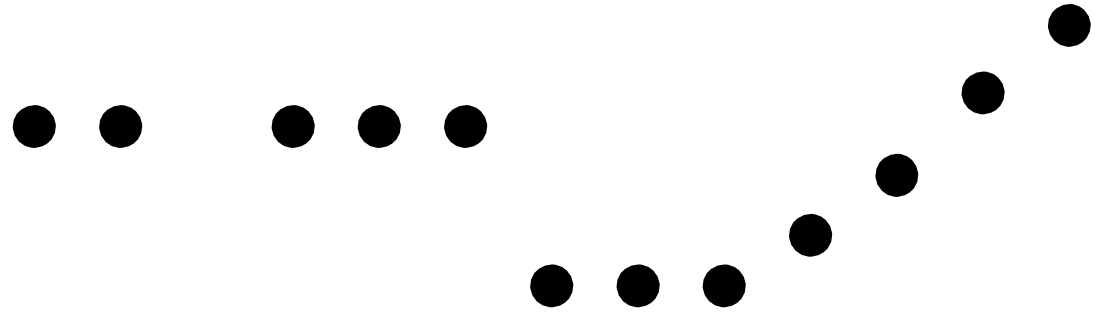


中值滤波

局外点

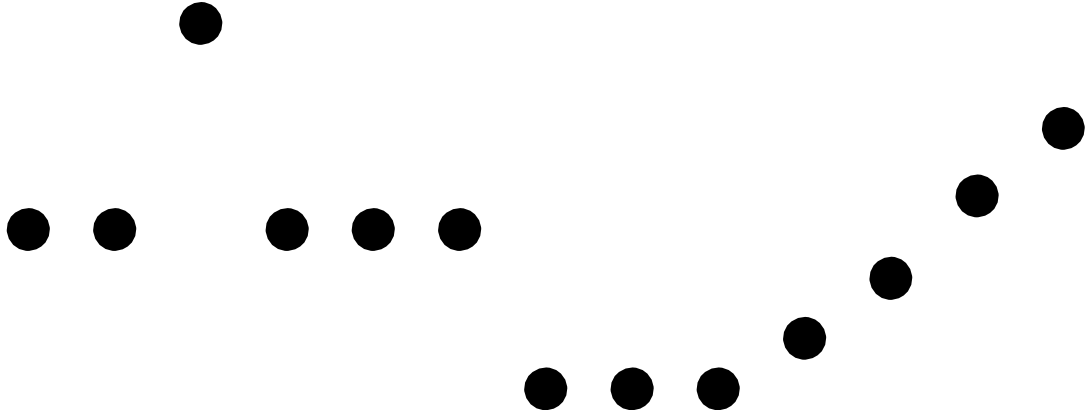


输入

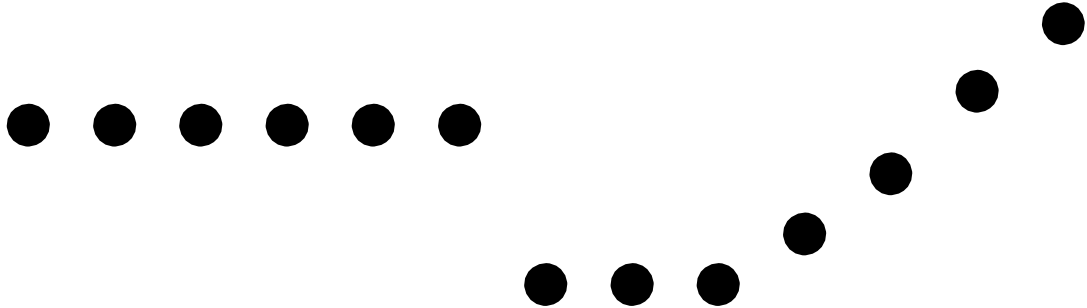


中值滤波

输入

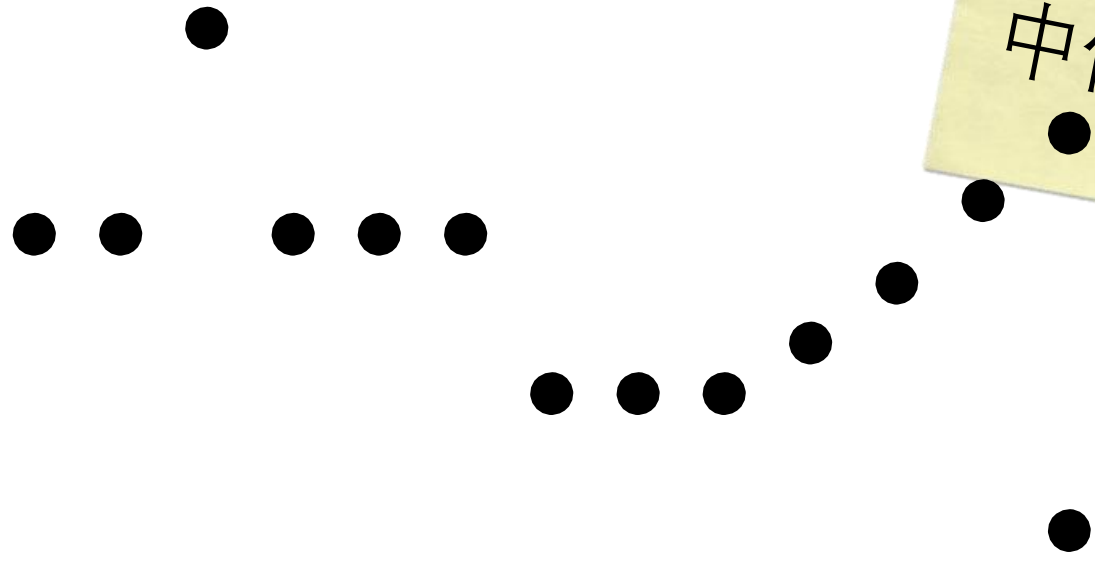


中值滤波

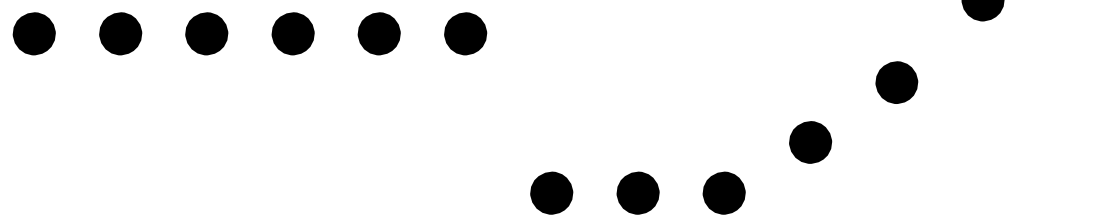




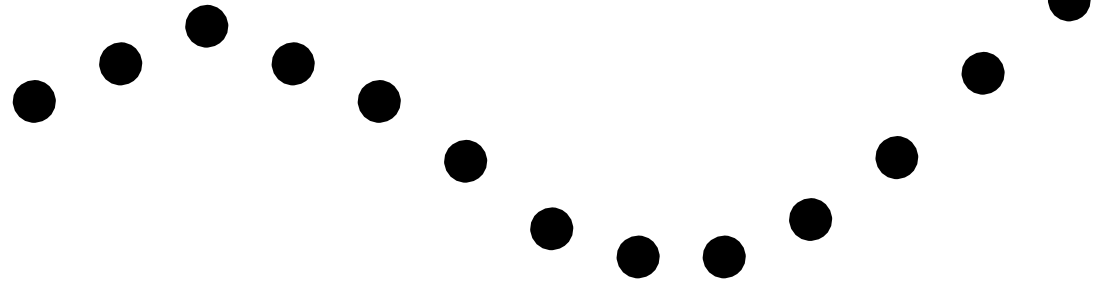
输入



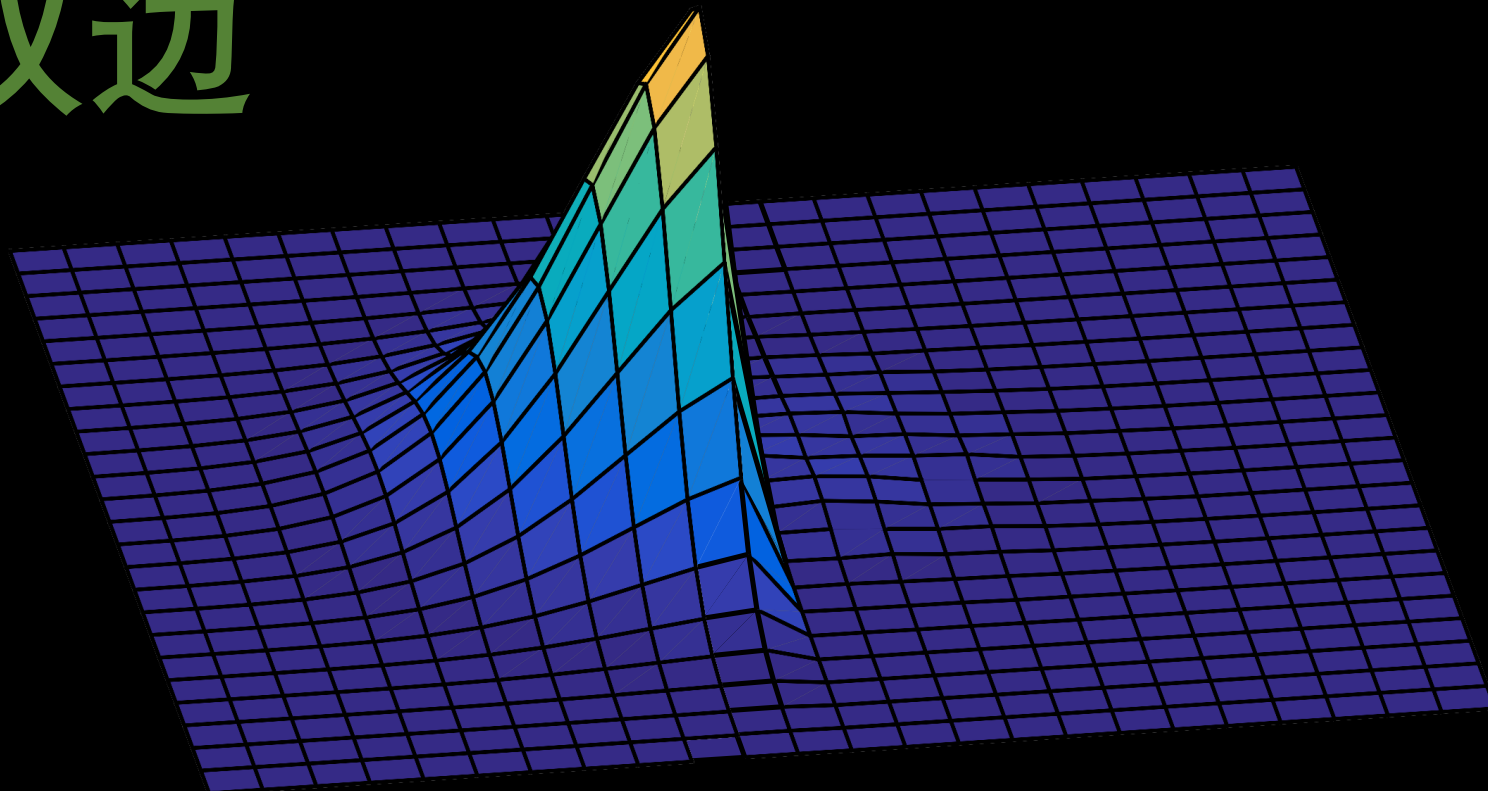
中值滤波



均值滤波



双边



滤波器

# Bilateral Filtering for Gray and Color Images

C. Tomasi \*

R. Manduchi

Computer Science Department  
Stanford University  
Stanford, CA 94305  
tomasi@cs.stanford.edu

Interactive Media Group  
Apple Computer, Inc.  
Cupertino, CA 95014  
manduchi@apple.com

## Abstract

*Bilateral filtering smooths images while preserving edges, by means of a nonlinear combination of nearby image values. The method is noniterative, local, and sim-*

*ple. It smooths gray and color images based on both their intensity and color. This method prevents blurring of edges, while still averaging within smooth regions. Anisotropic diffusion [12, 14] is a popular answer: local image variation is measured at every point, and pixel values are averaged from neighborhoods whose size and shape depend on local variation. Diffusion*

**IEEE International Conference on Computer Vision (ICCV) 1998**



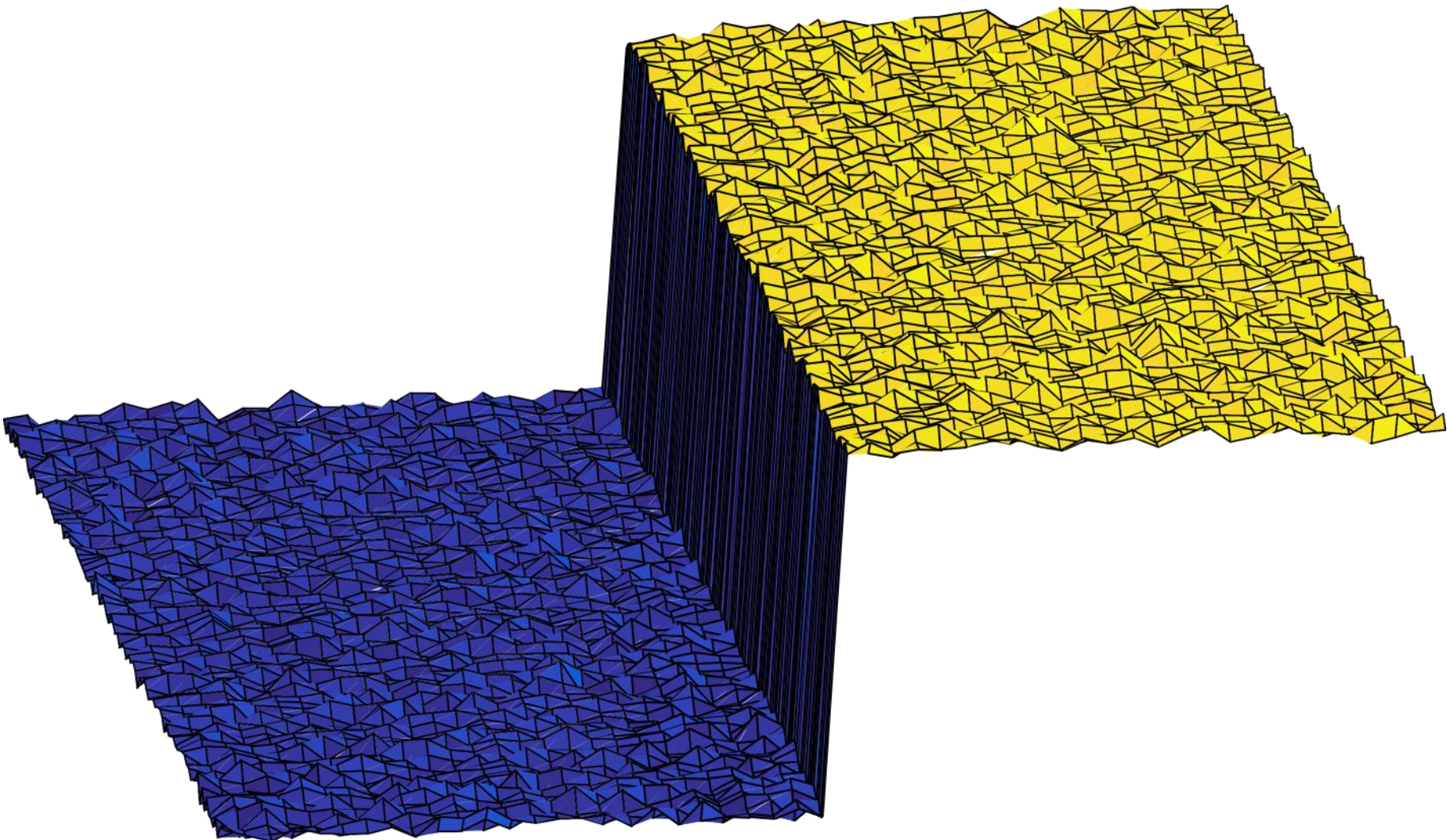


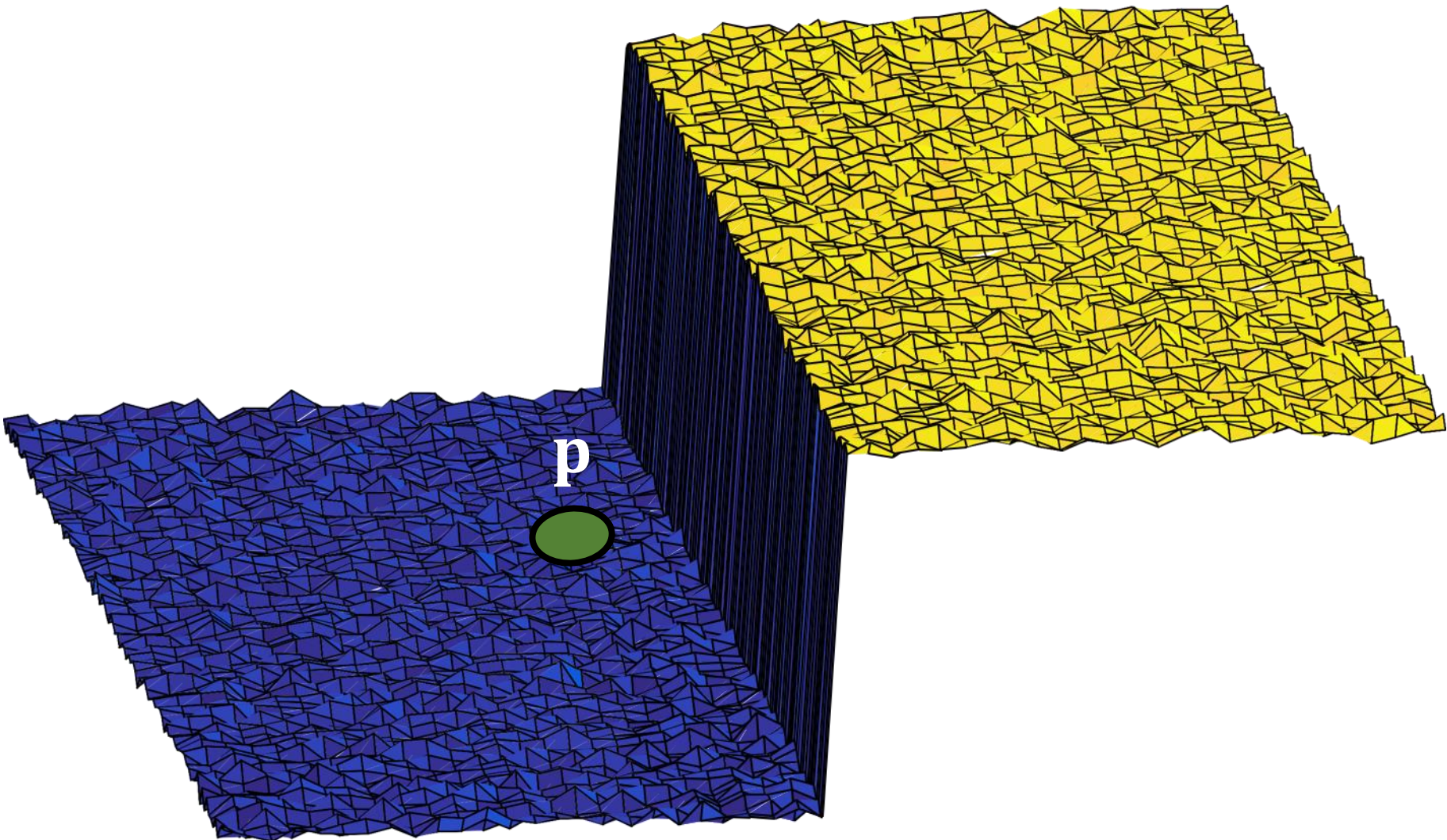
高斯平滑



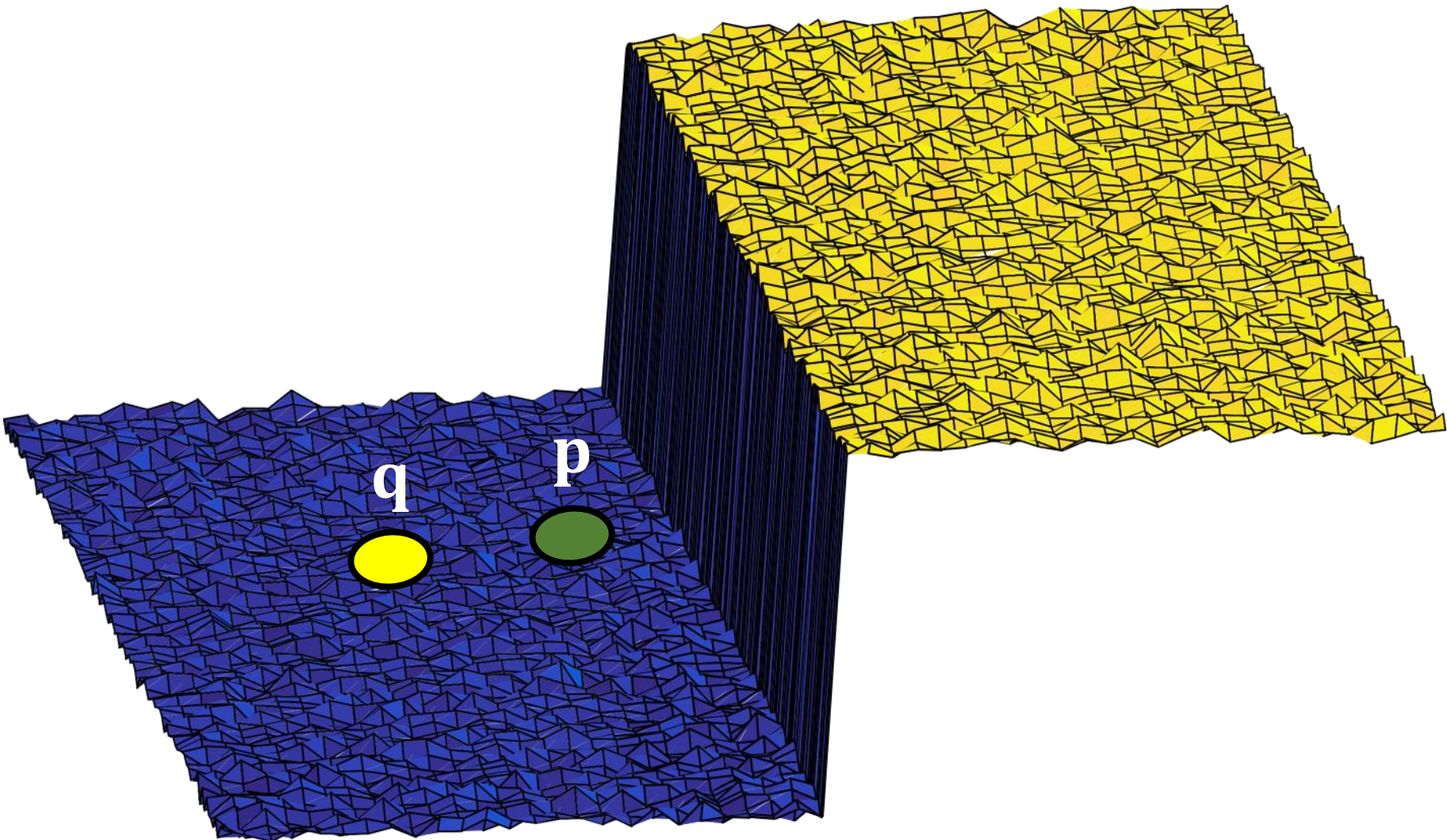
高斯平滑



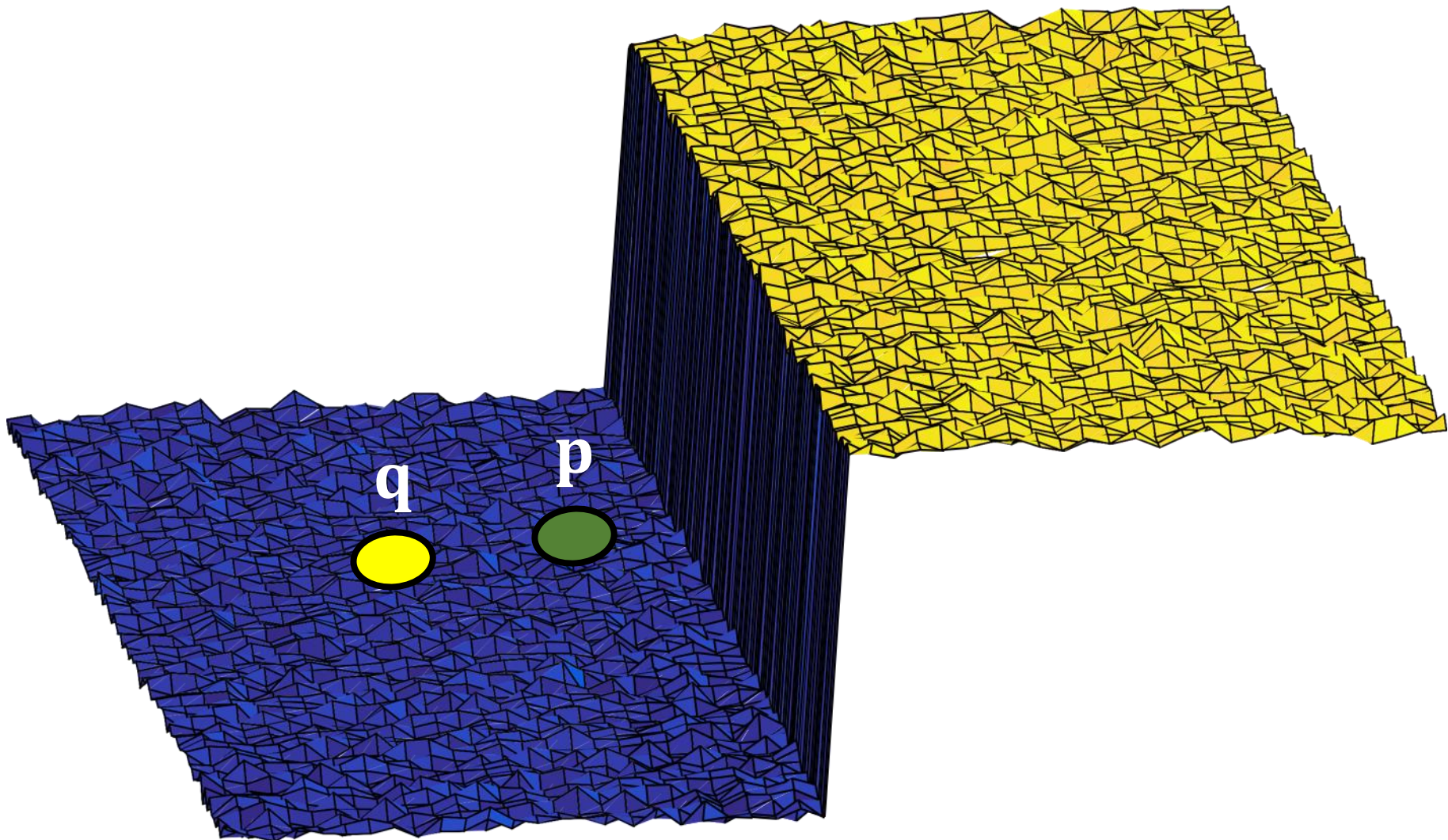






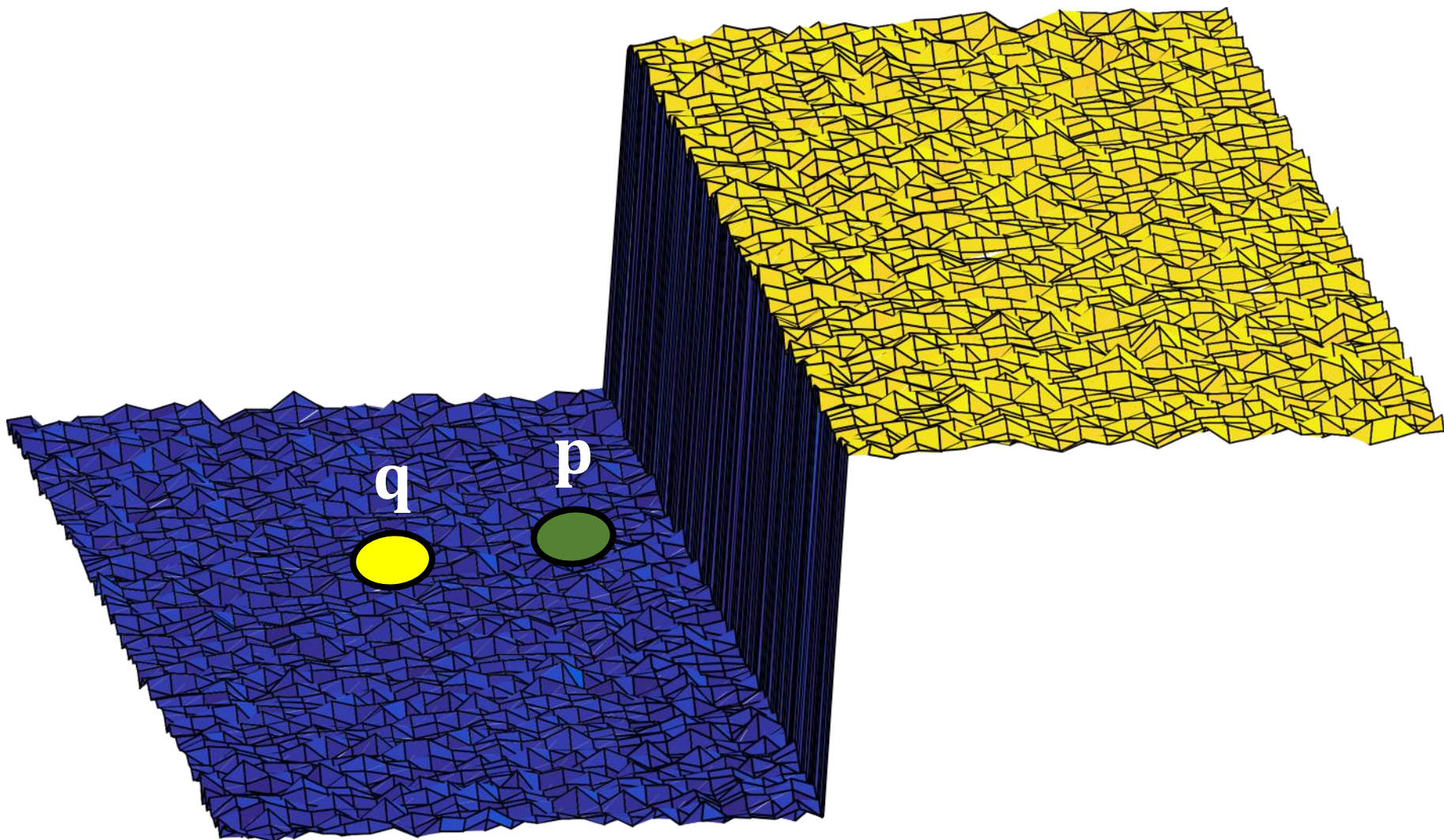


$$I'(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) I(\mathbf{q})$$





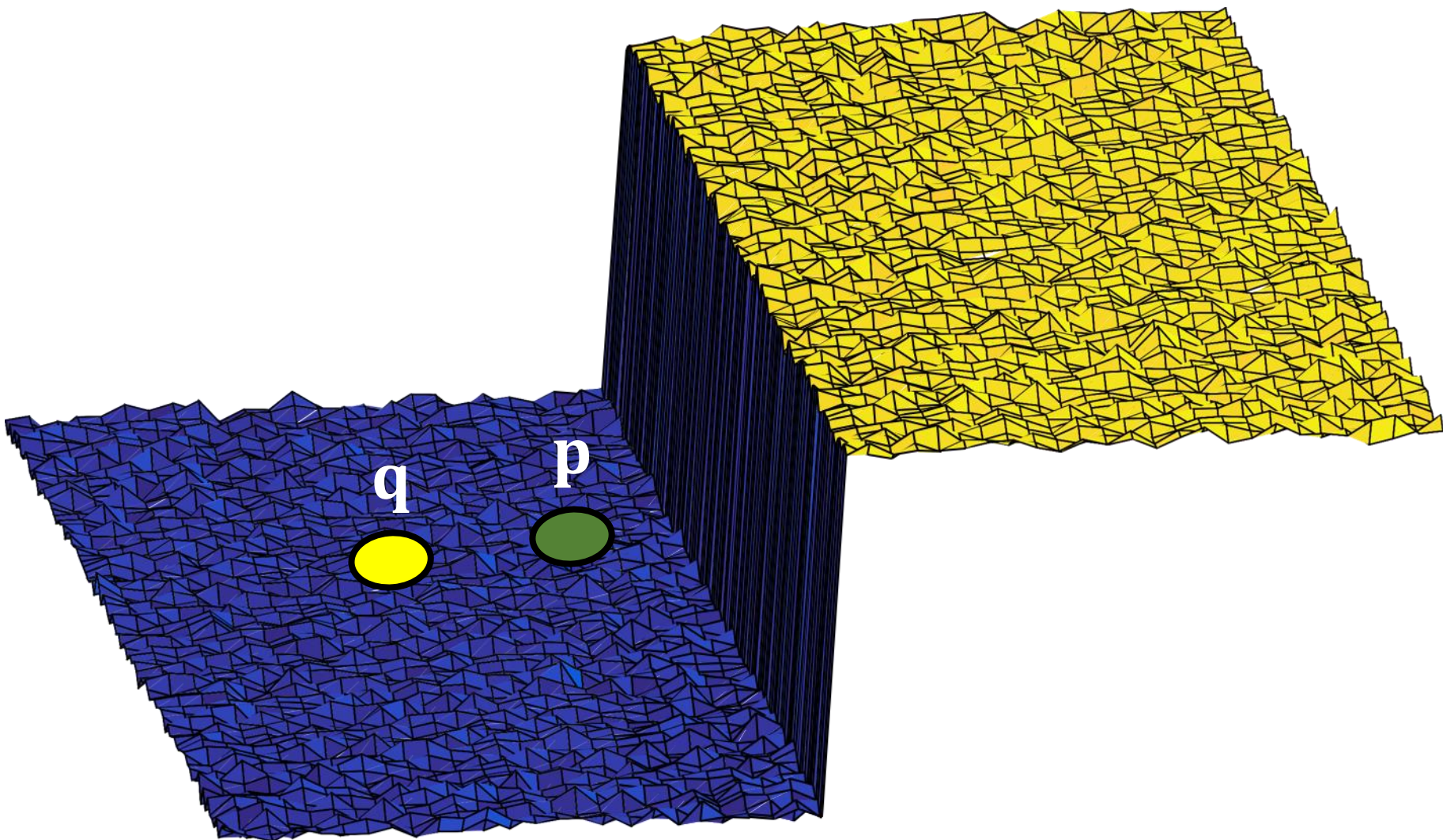
$$I'(\mathbf{p}) = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) I(\mathbf{q})$$





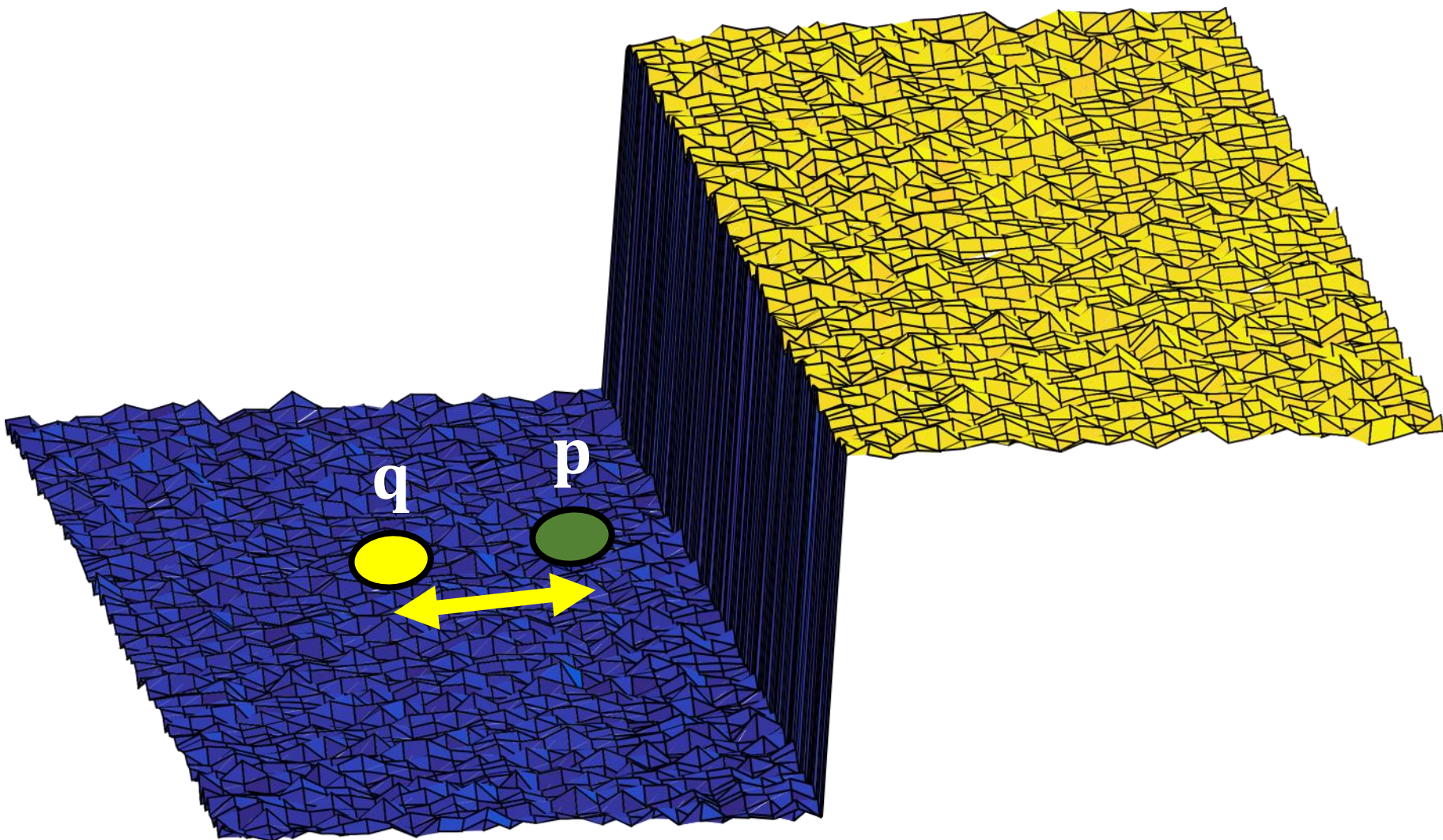
空间域权重

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



空间域权重

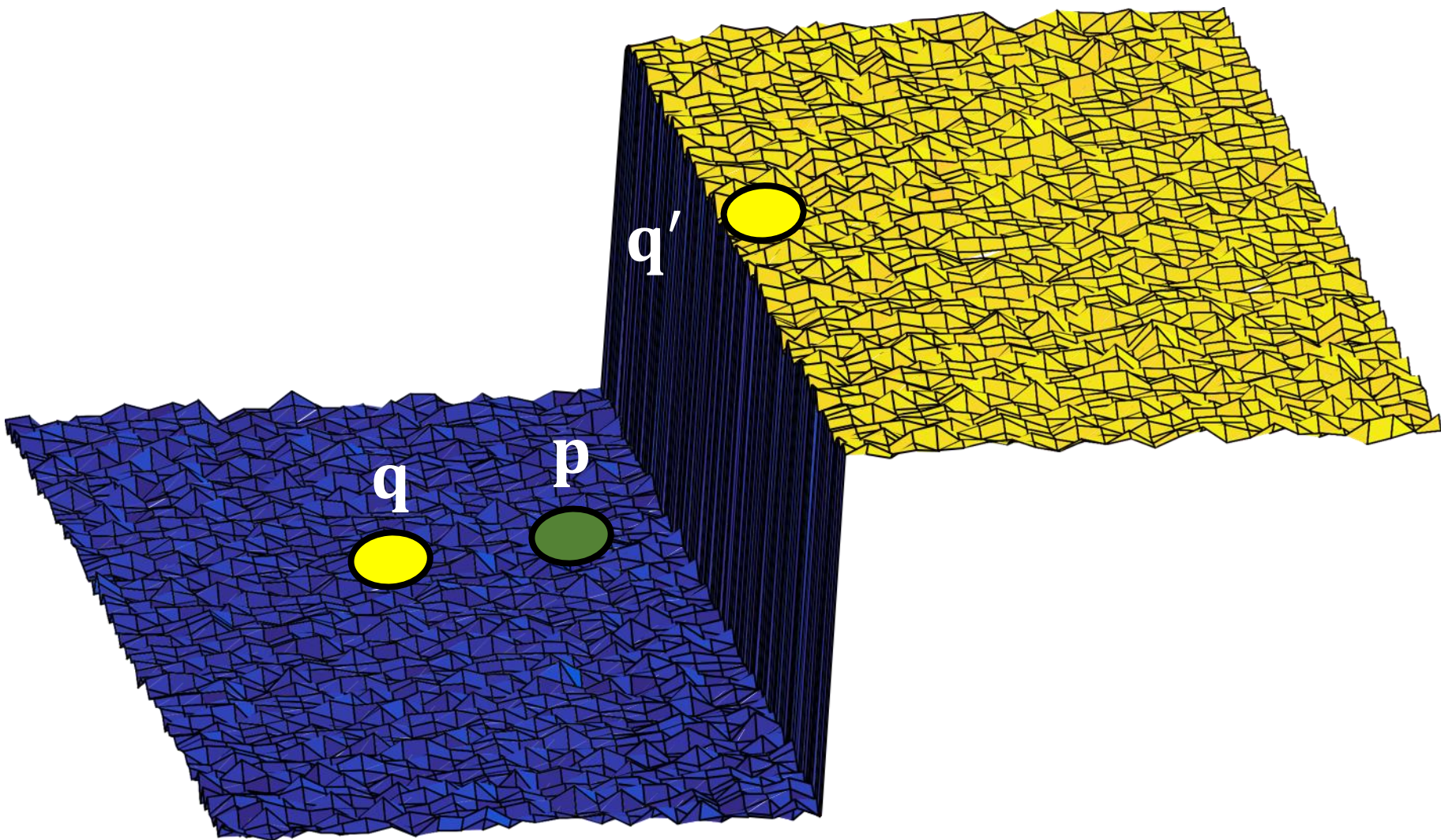
$$I'(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) I(\mathbf{q})$$





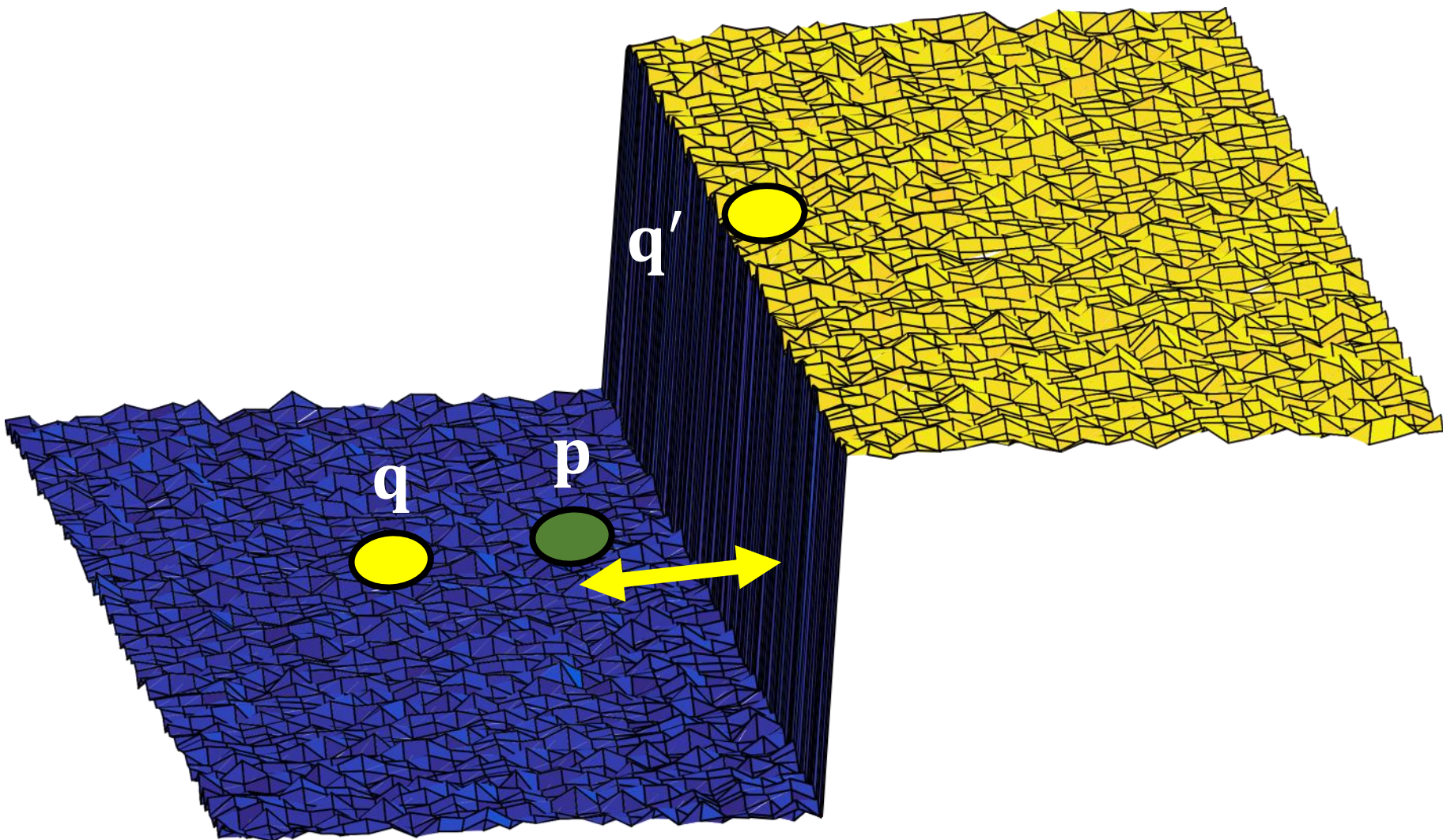
空间域权重

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



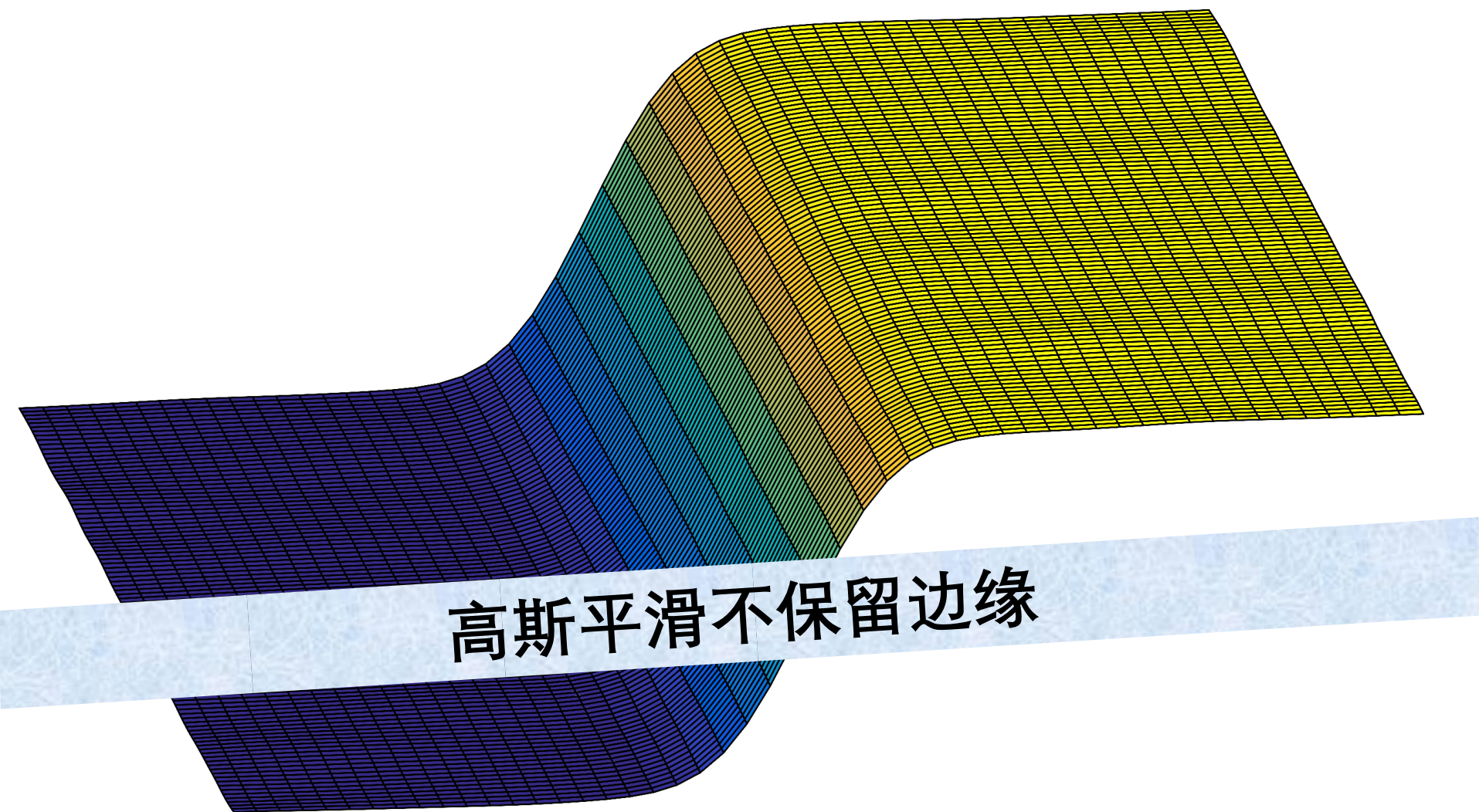
空间域权重

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

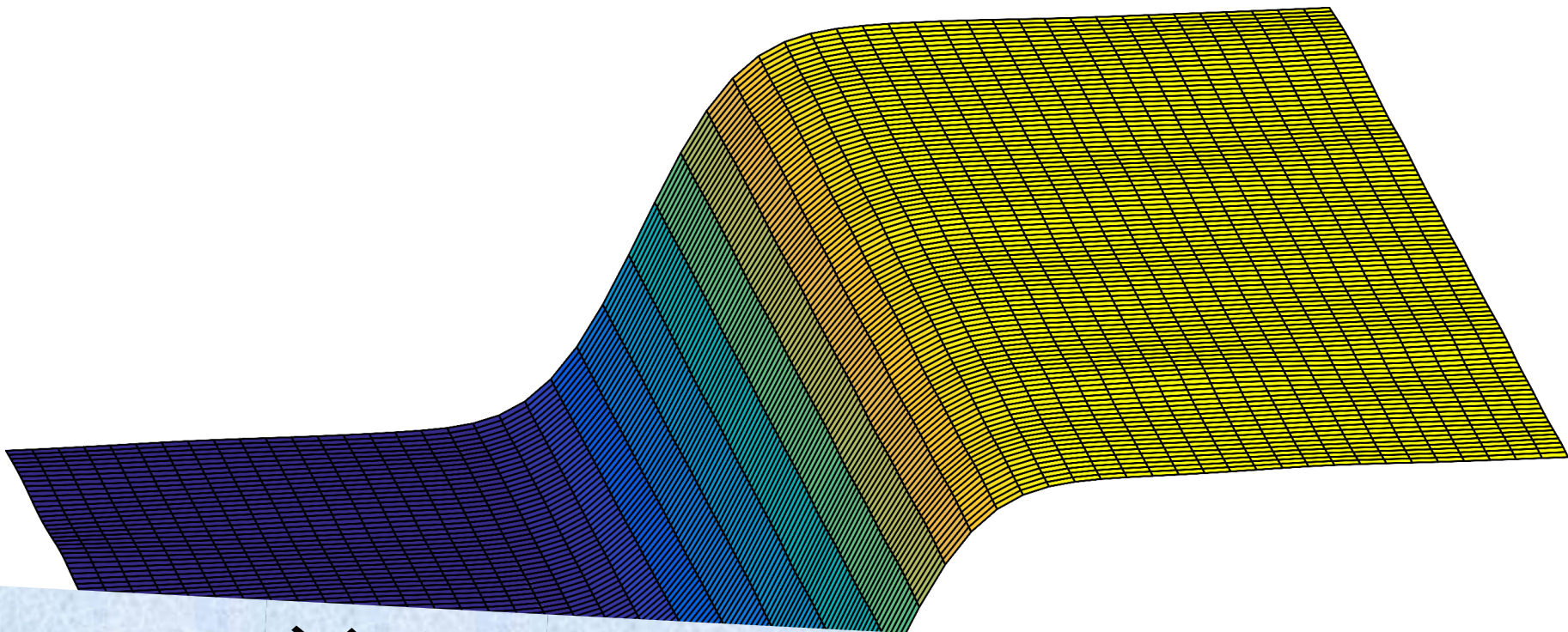




$$I'(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) I(\mathbf{q})$$



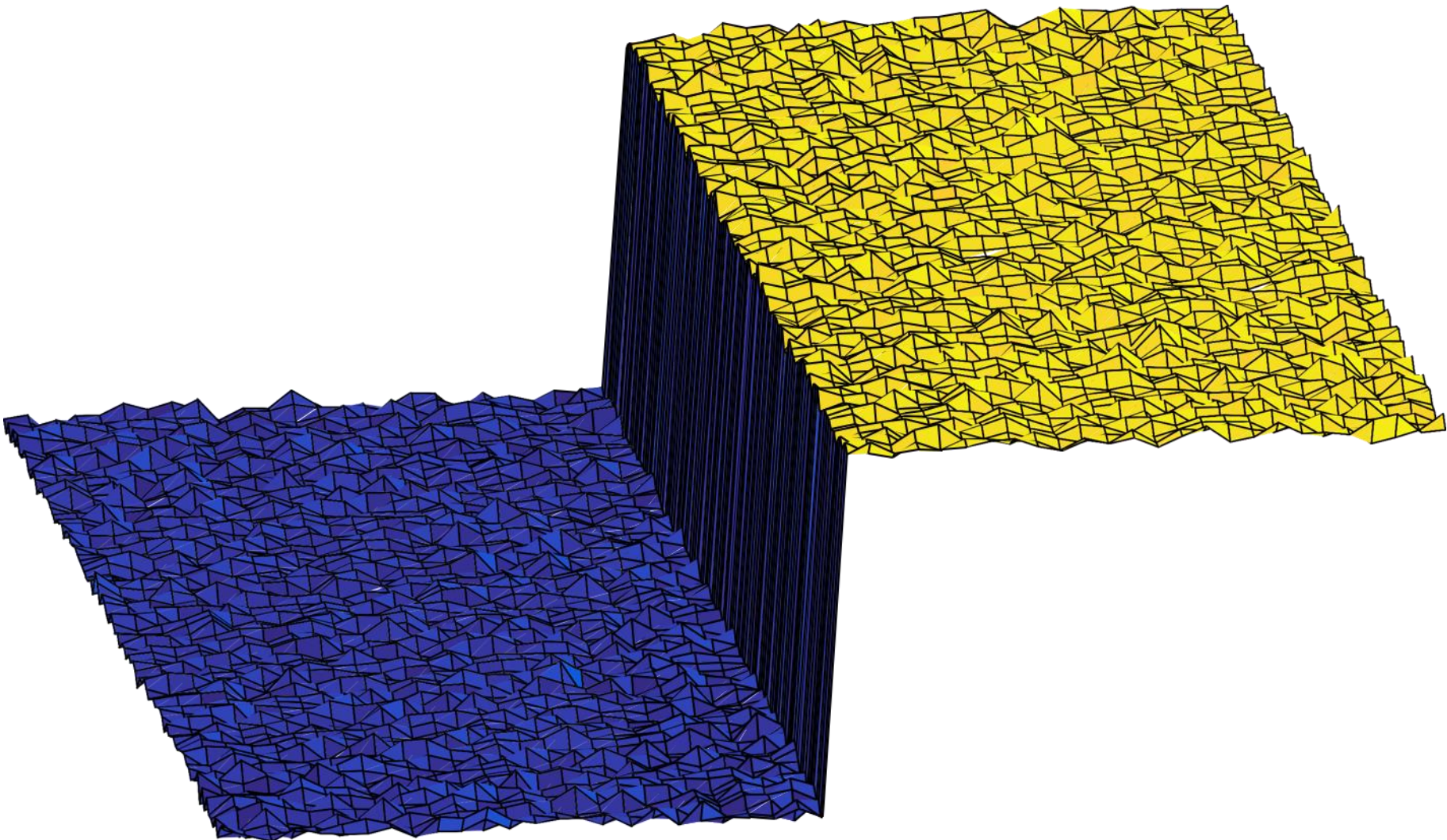
$$I'(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) I(\mathbf{q})$$



空间权重的形状在每个位置都相同

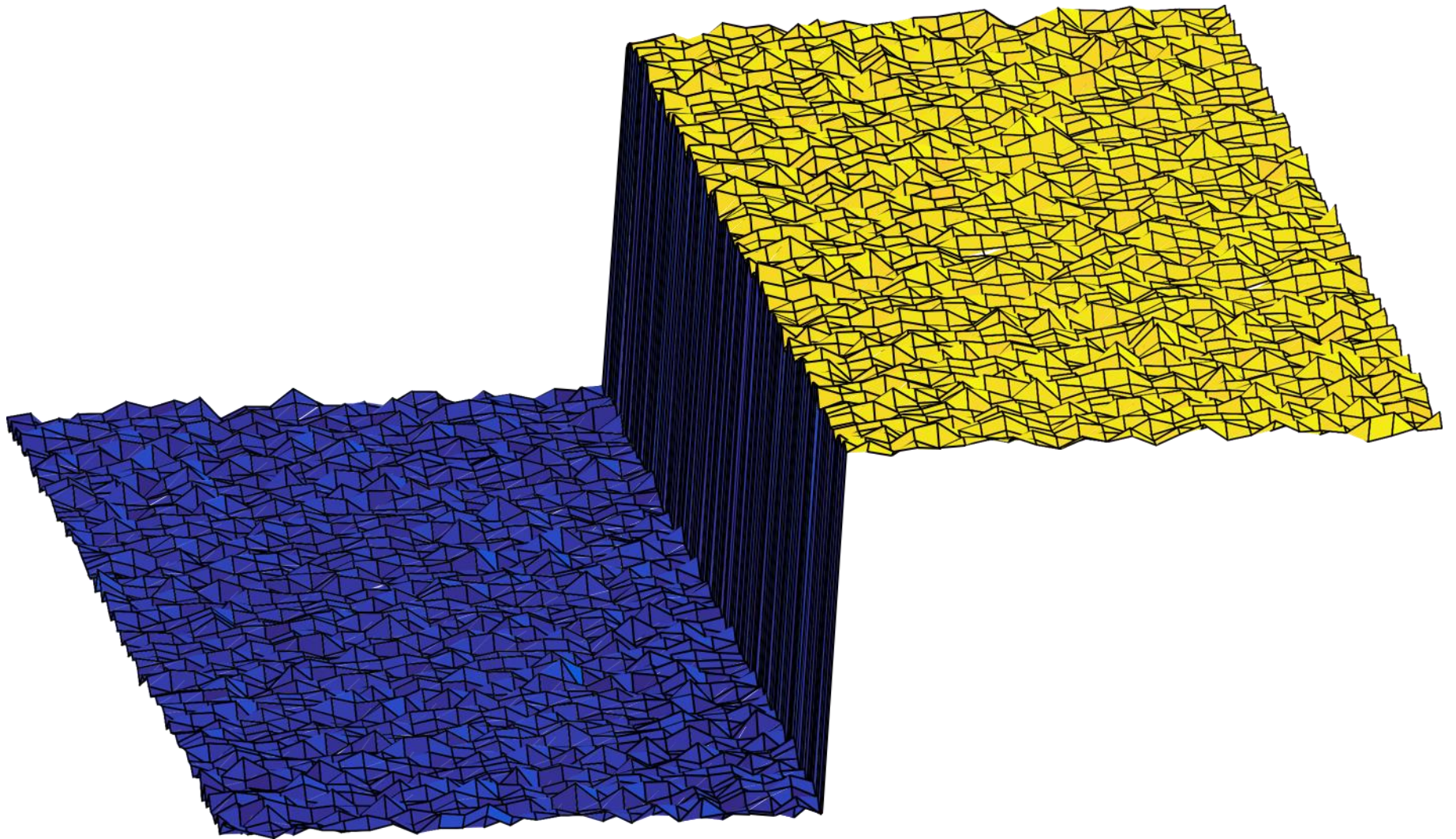


$$I'(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) I(\mathbf{q})$$

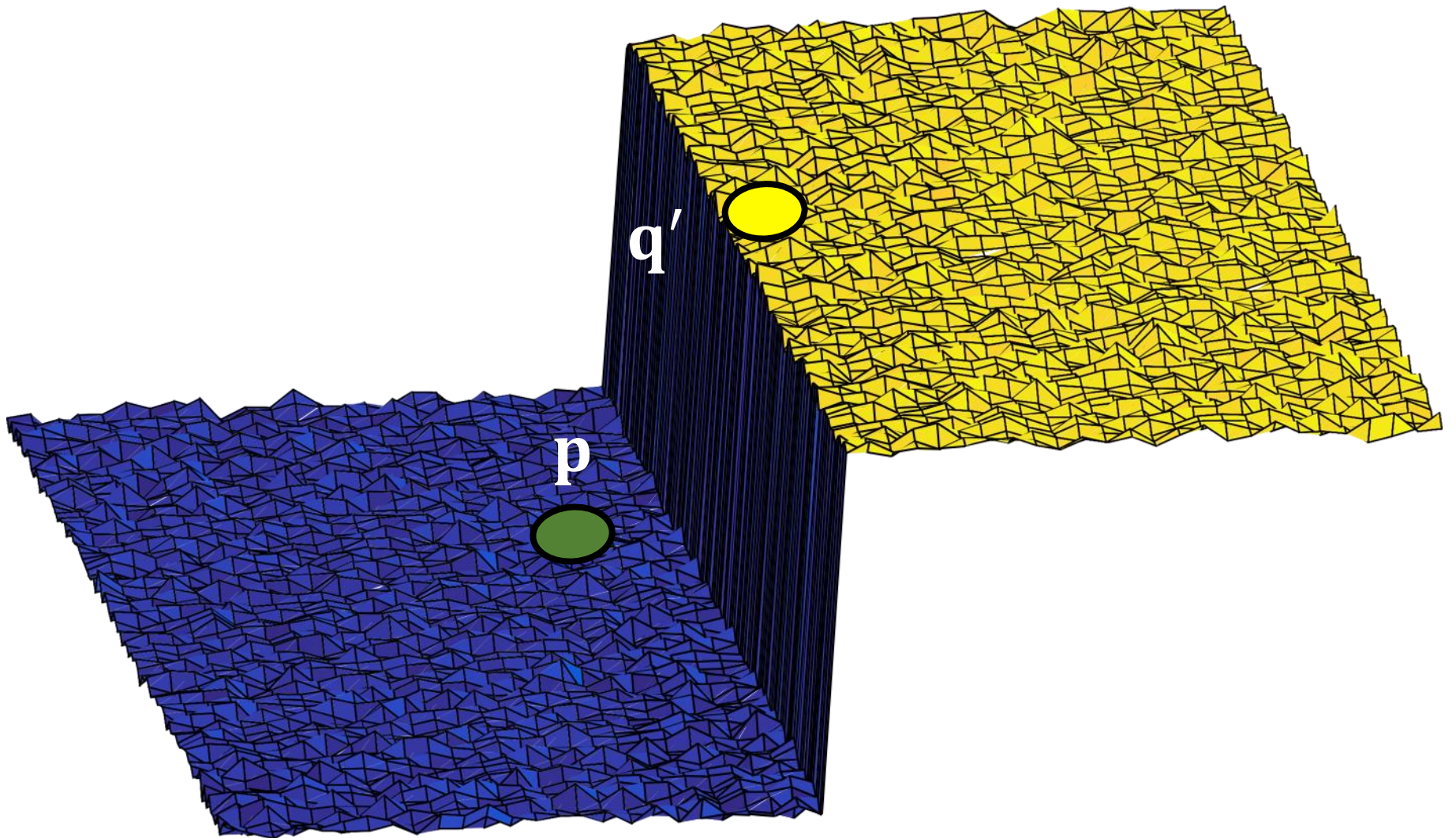




$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$

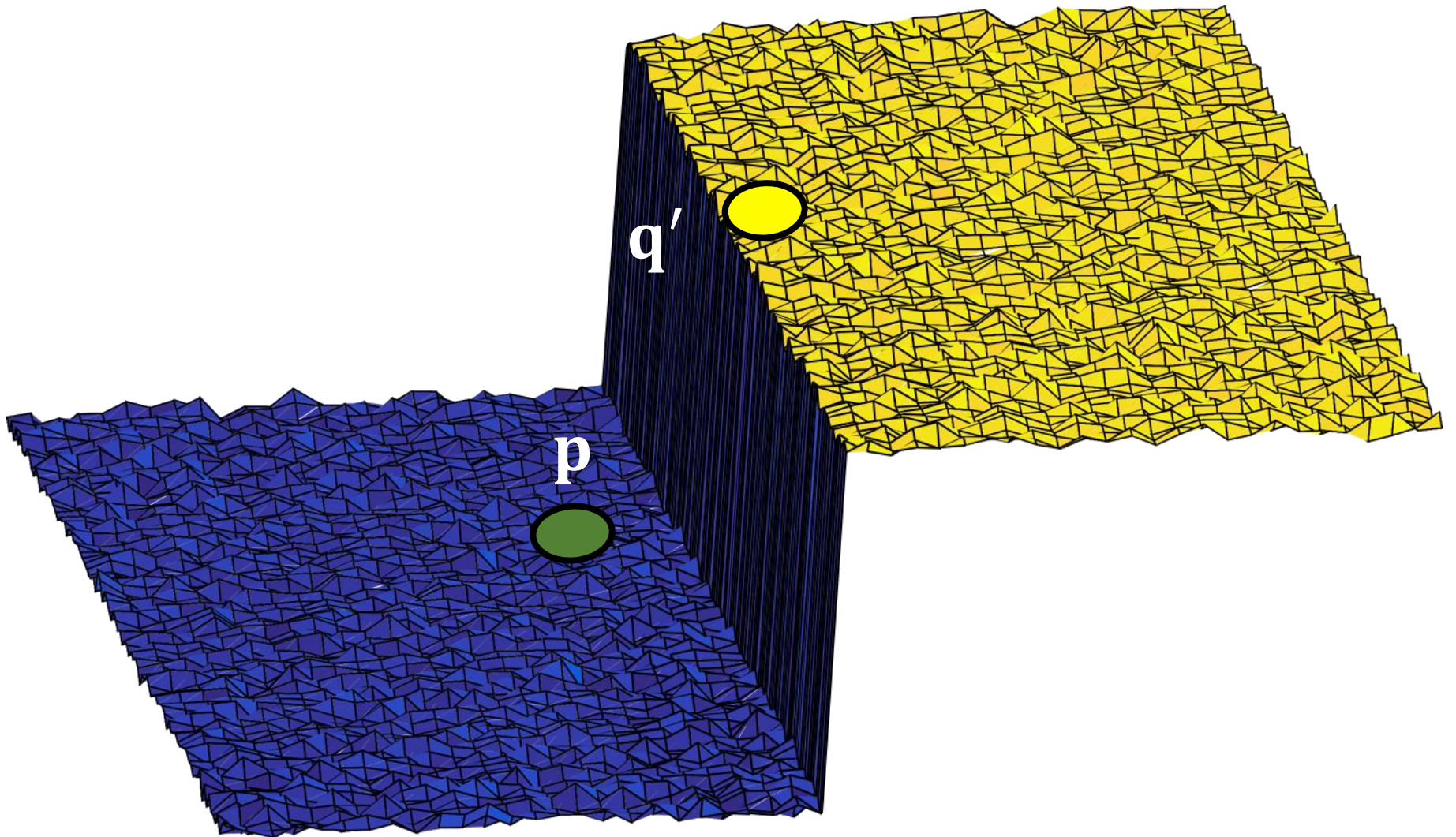


$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$



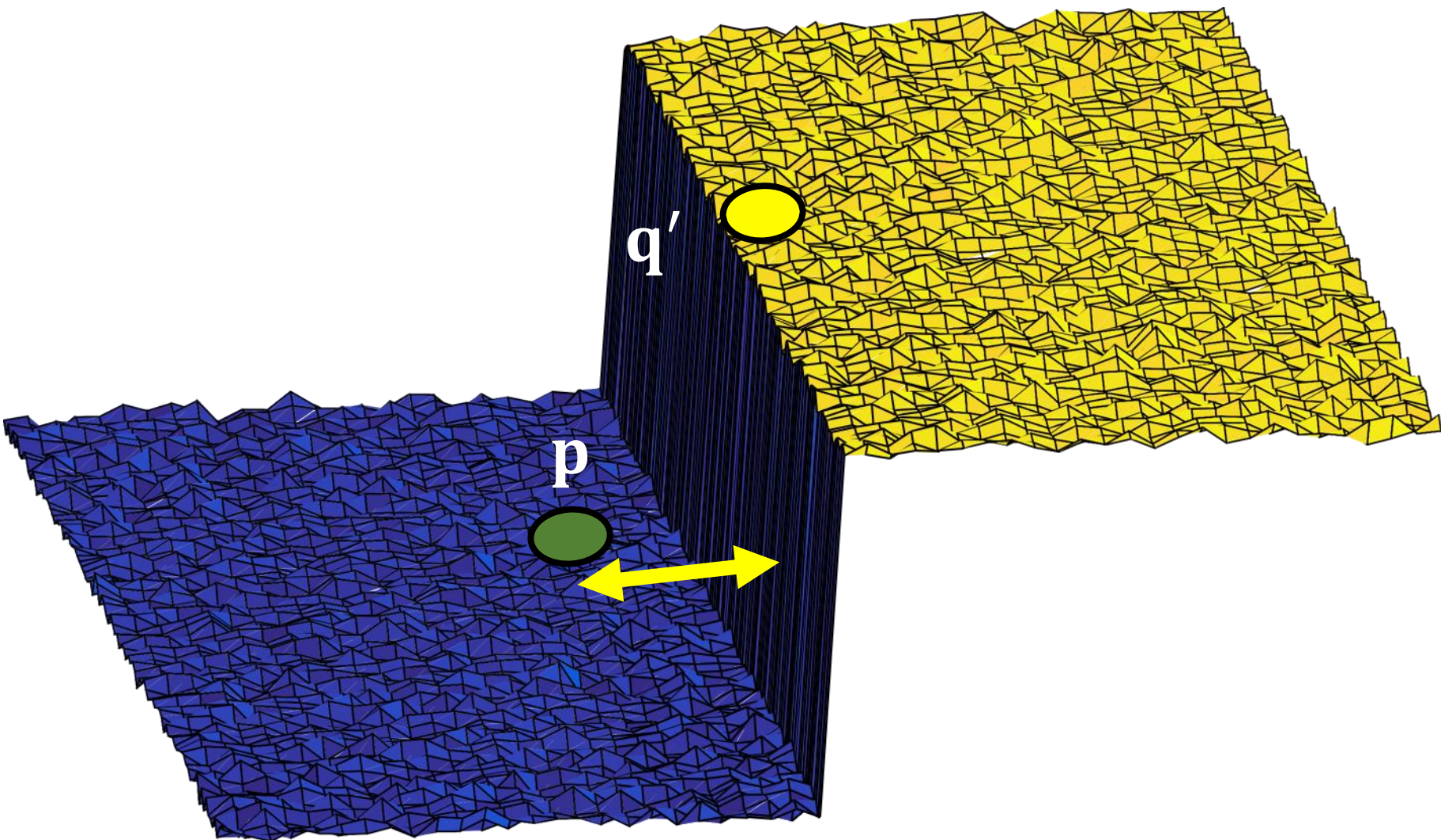


$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$



# 空间域权重

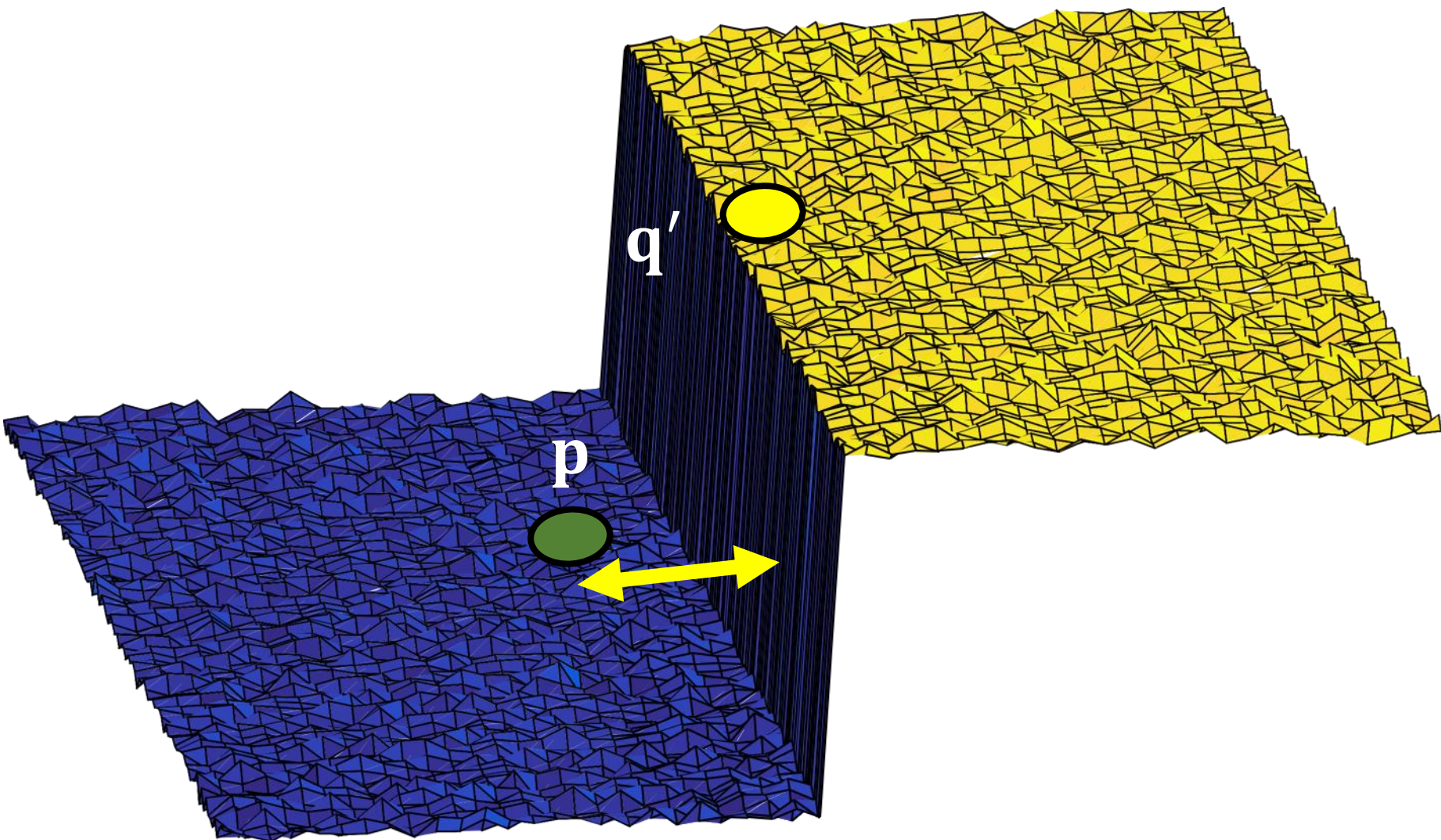
$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$





# 空间域权重

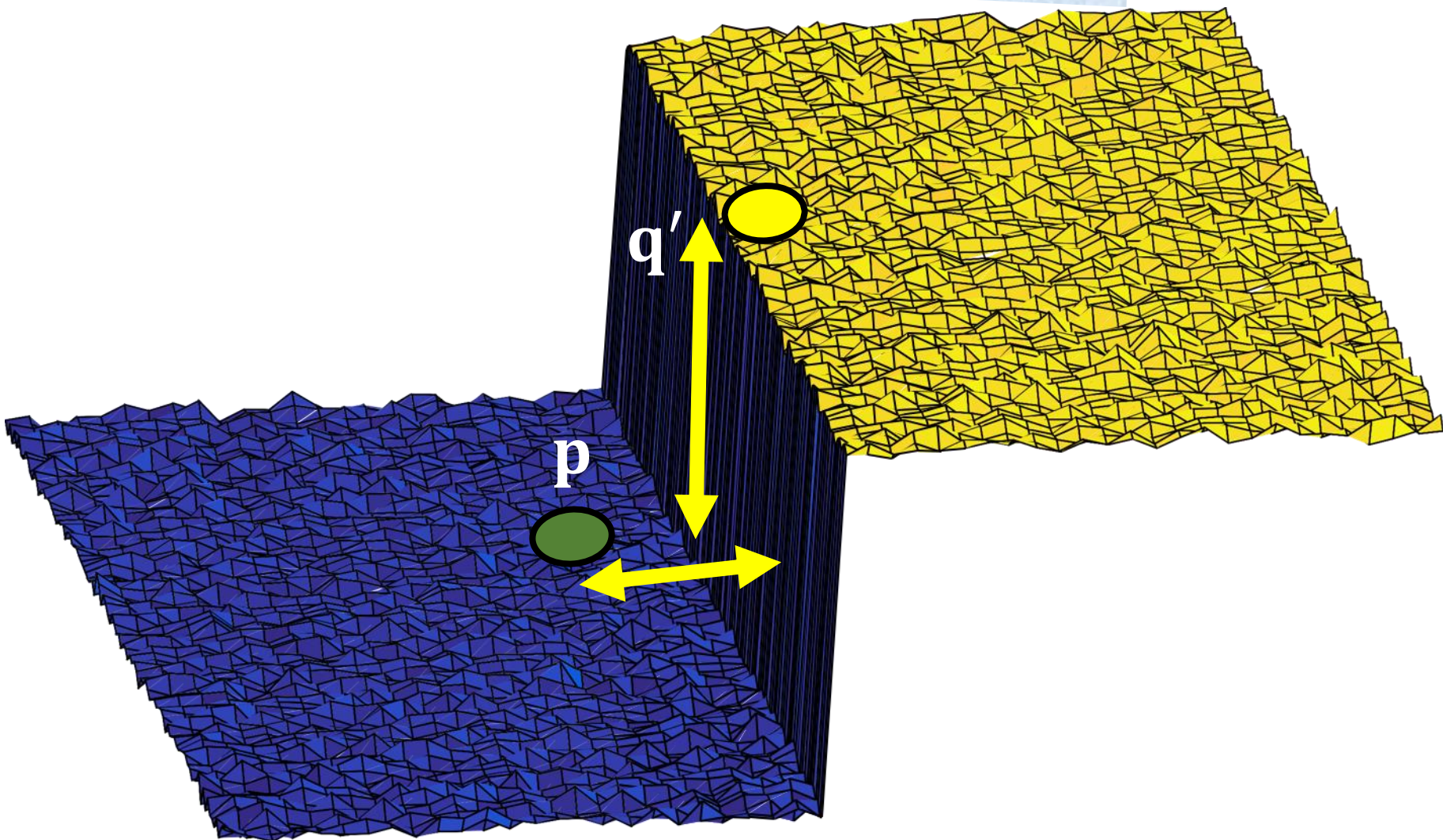
$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$



空间域权重

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

值域权重

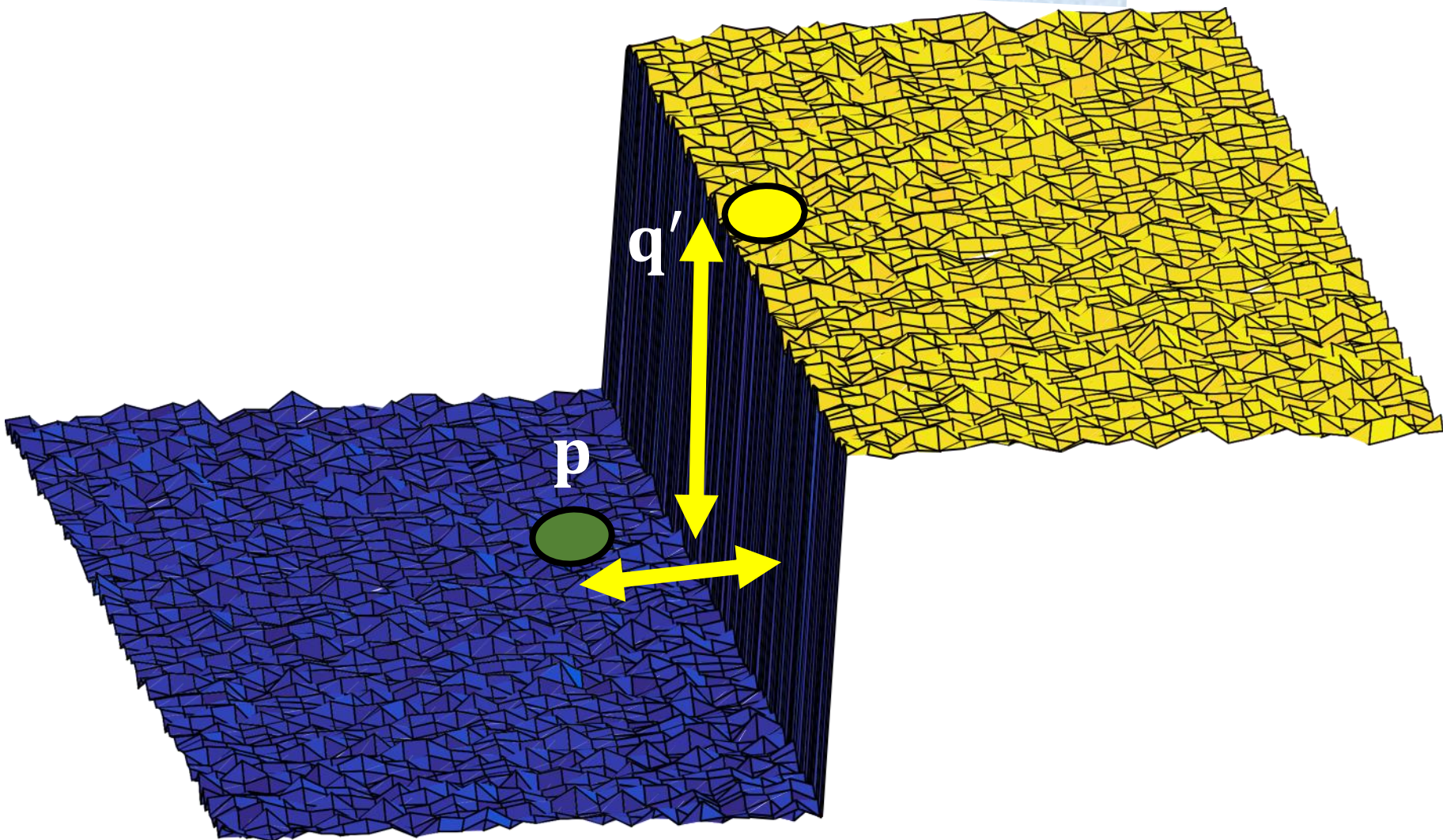




空间域权重

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

值域权重

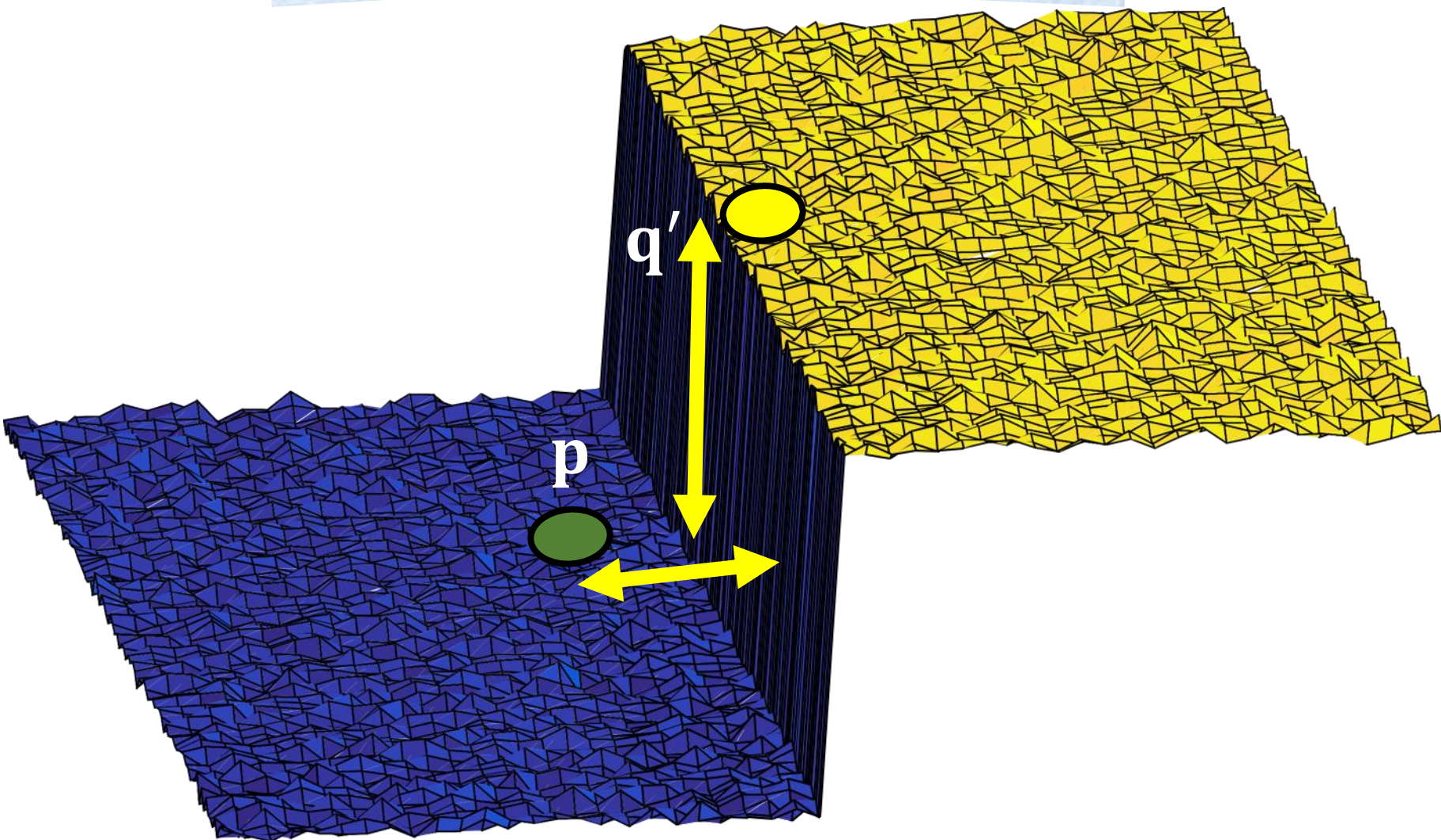


空间域权重

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

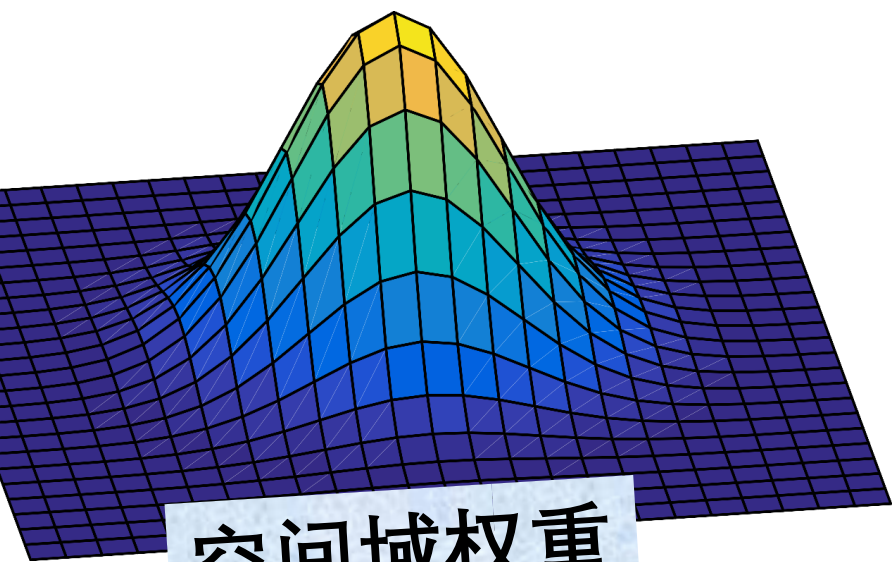
归一化

值域权重

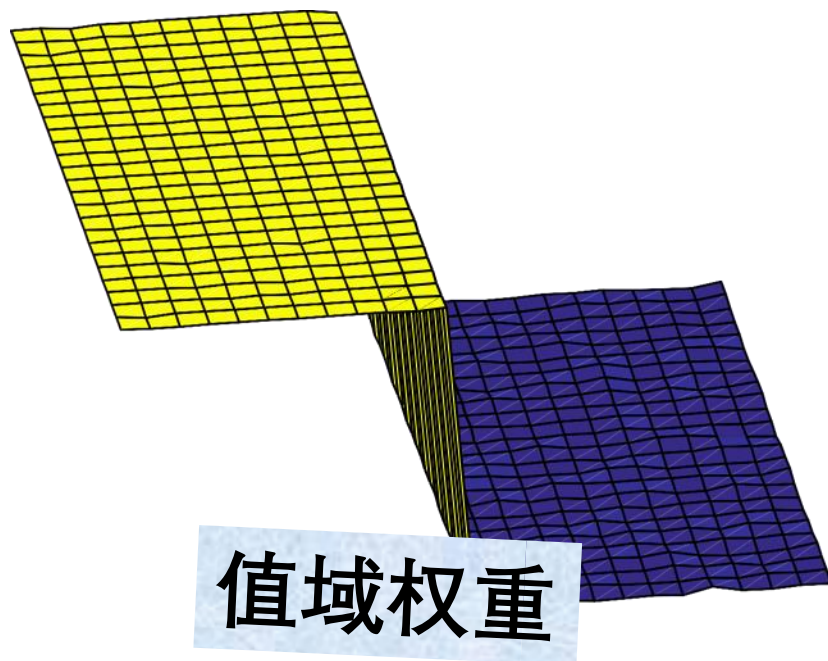




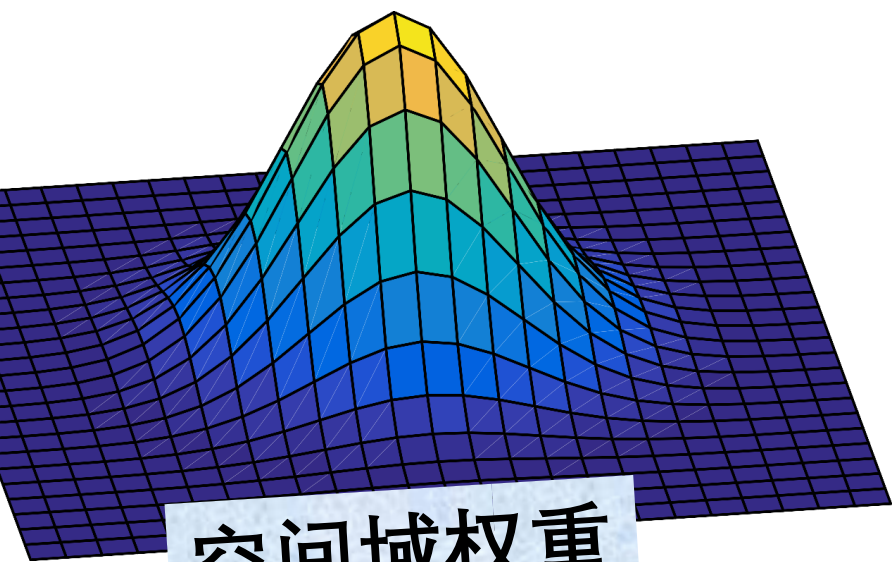
$$G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)$$



$$G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|)$$



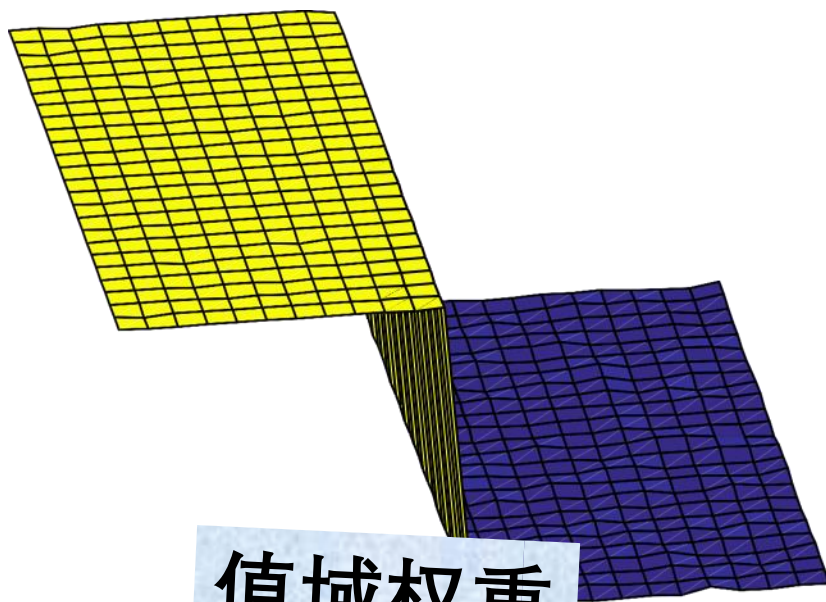
$$G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)$$



空间域权重

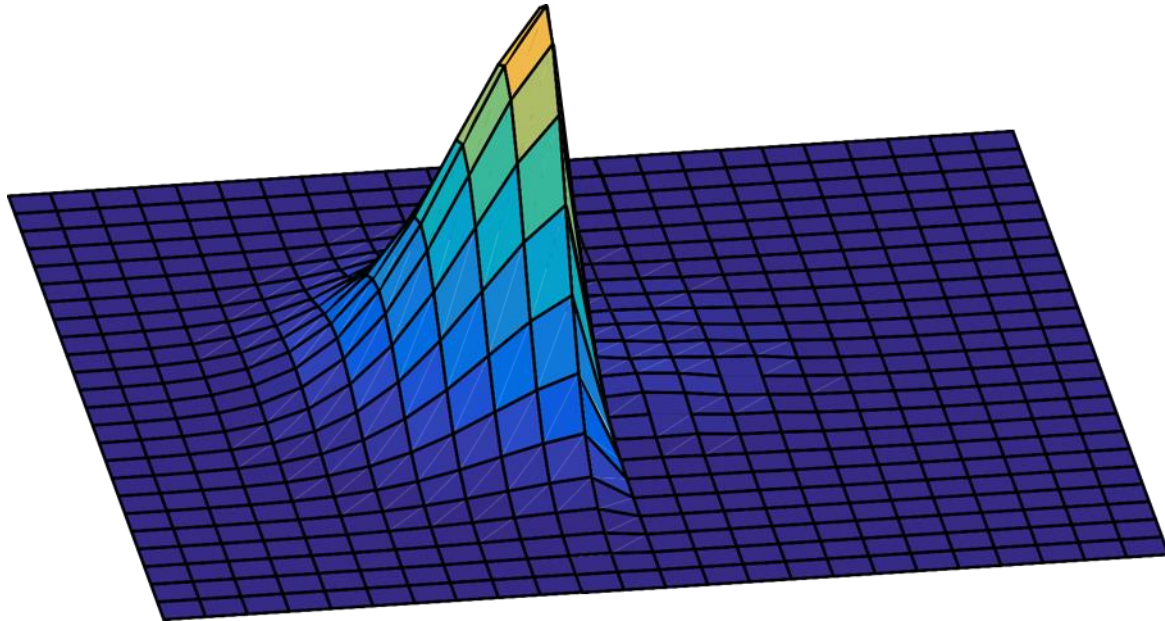
×

$$G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|)$$

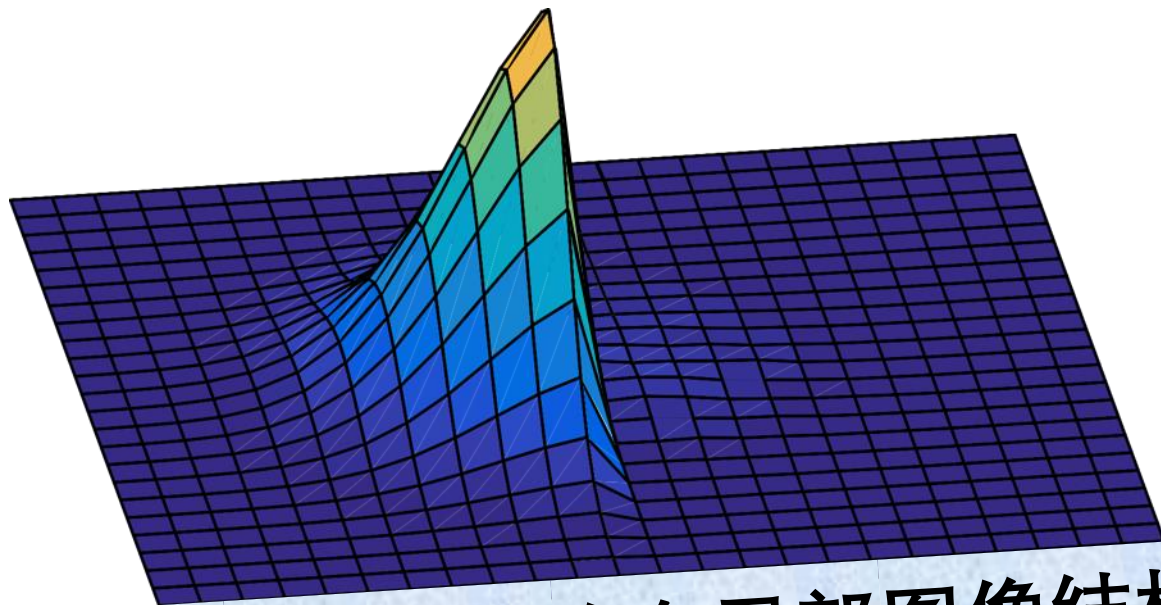


值域权重

$$w(\mathbf{p})G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|)$$



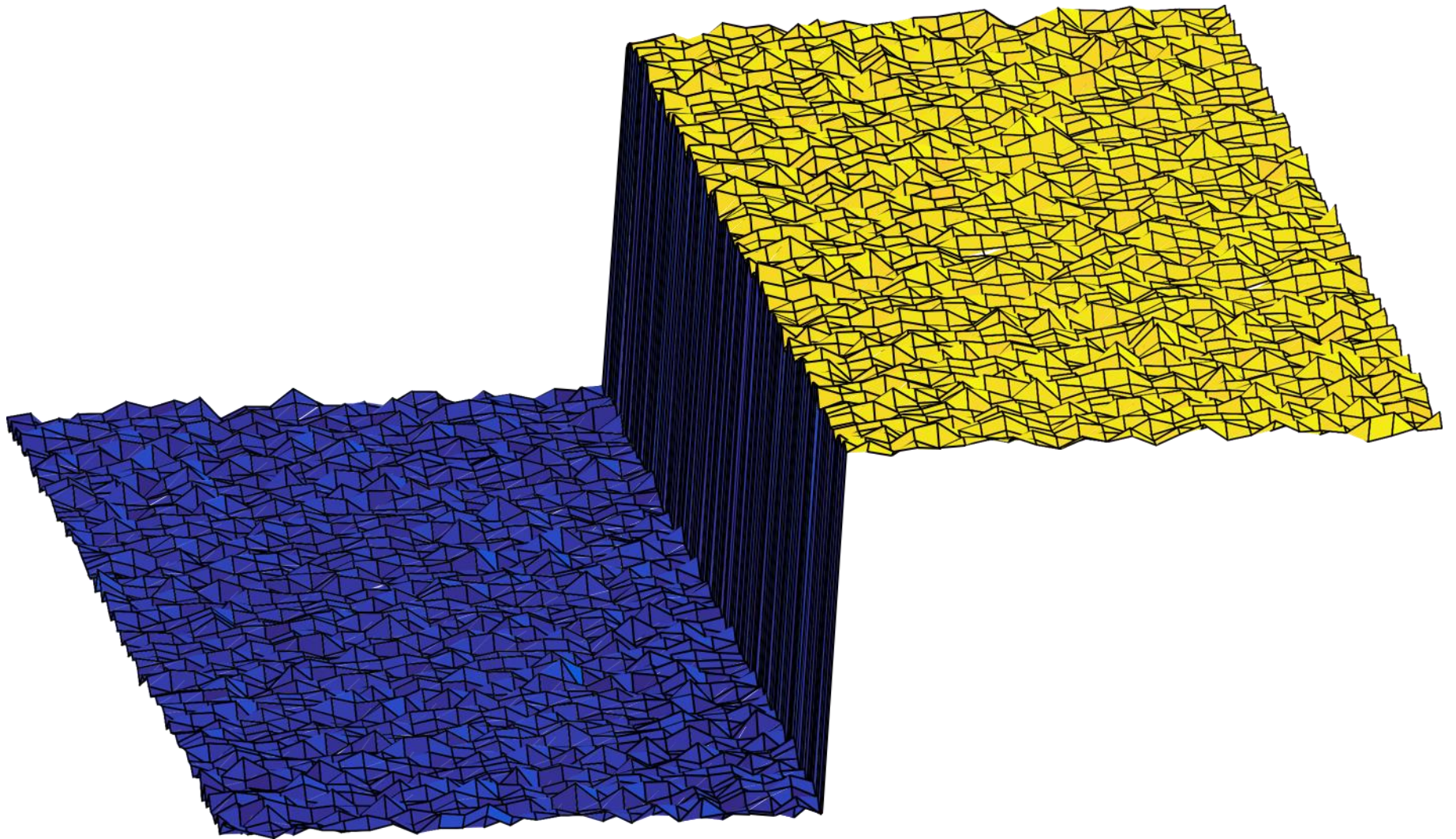
$$w(\mathbf{p})G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|)$$



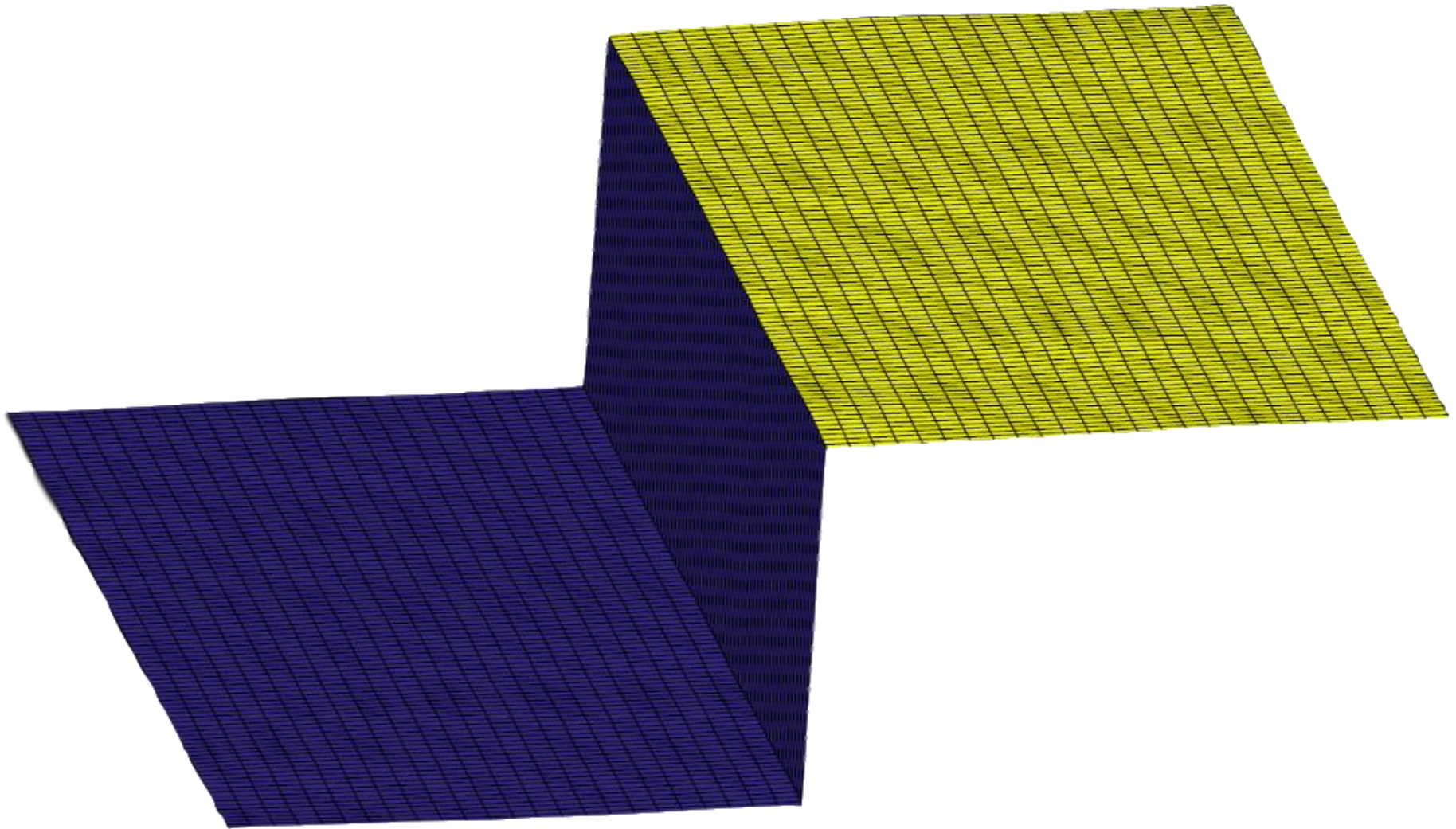
权重滤波器的形状符合局部图像结构



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

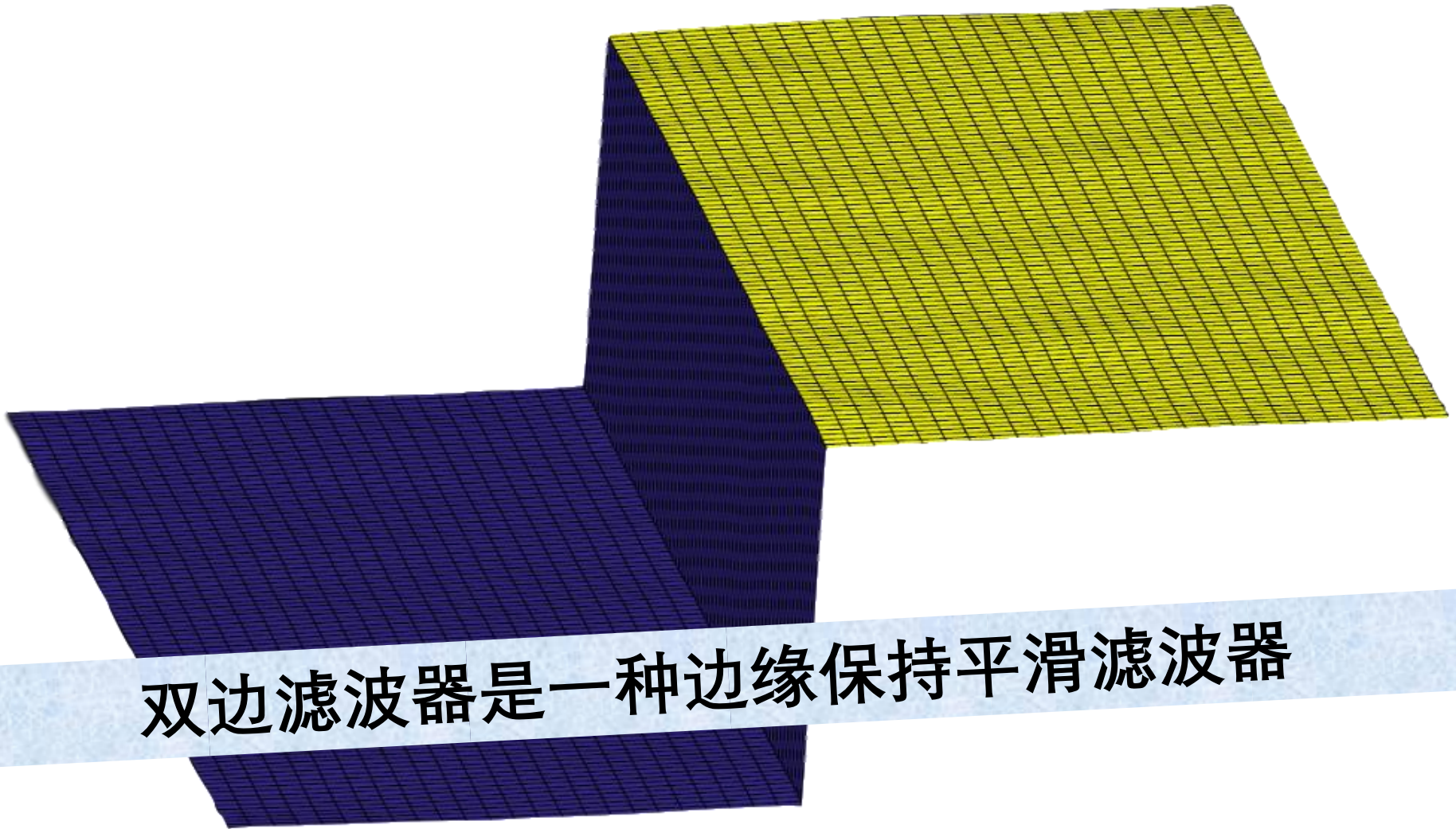


$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$





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



双边滤波器是一种边缘保持平滑滤波器



双边滤波





双边滤波

