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，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 512×512，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 函数在整数像素网格上插值，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

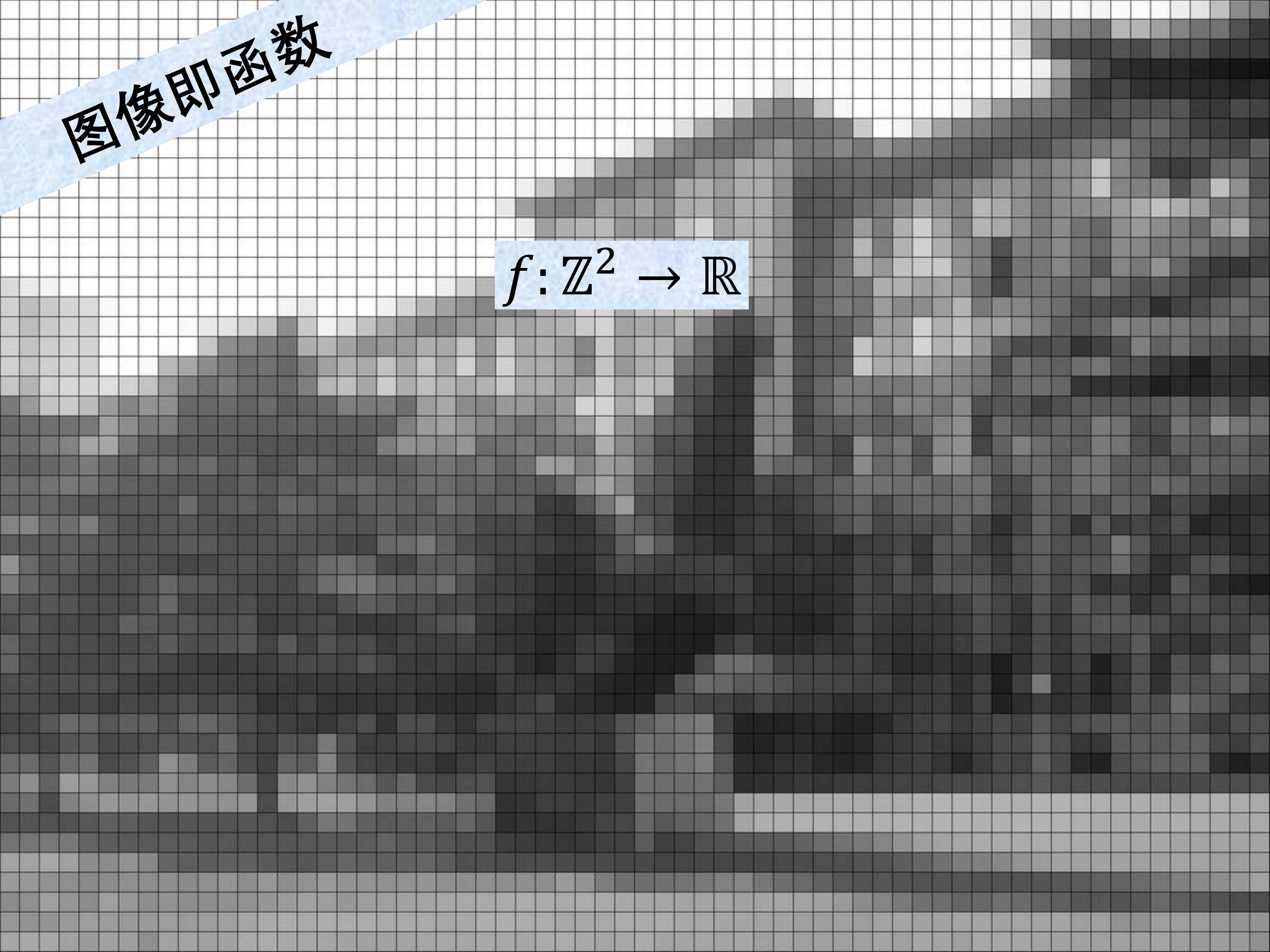
本节主题：

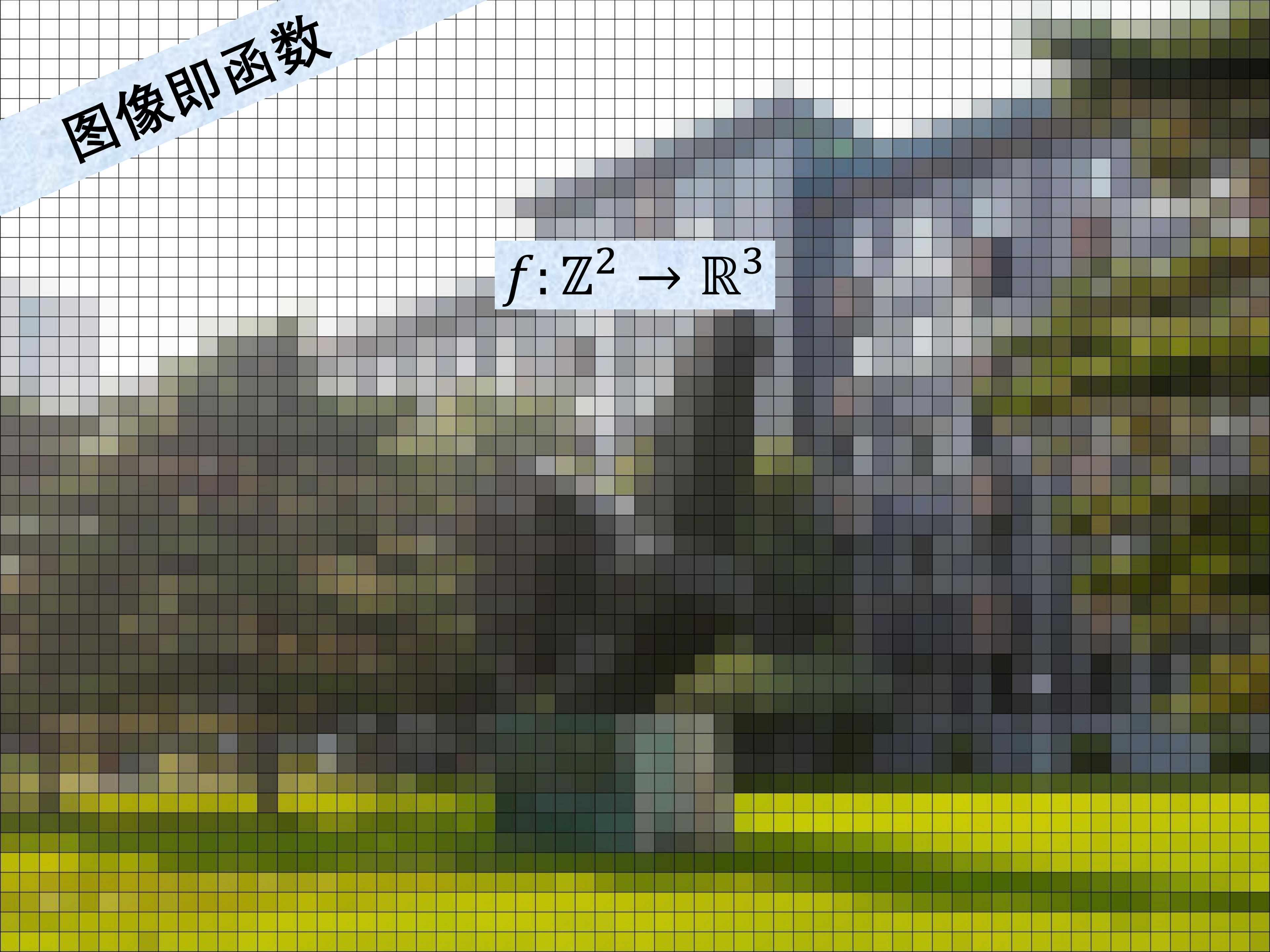
生物视觉与色彩

图像滤波

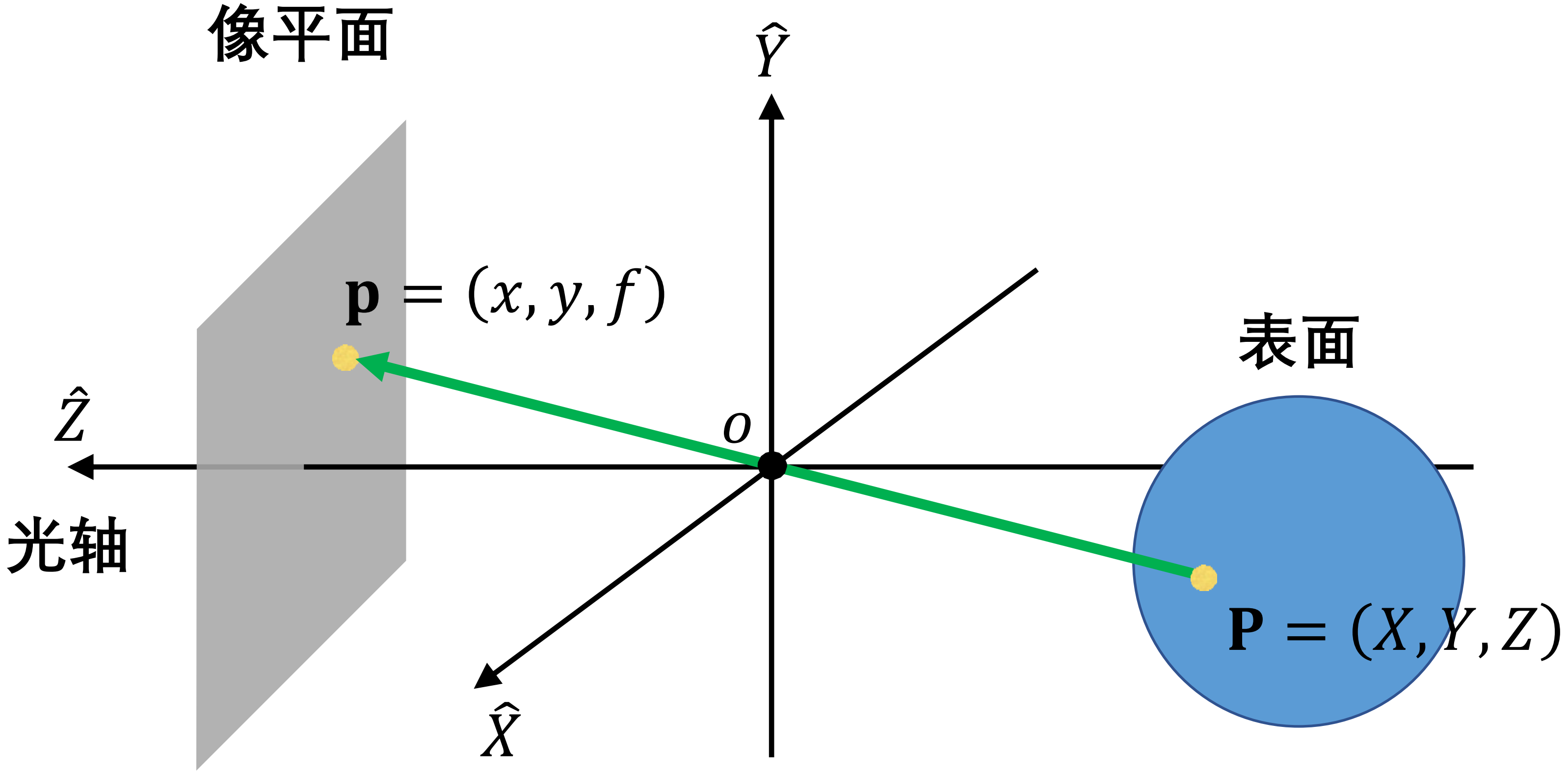
什么是图像？


图像即函数

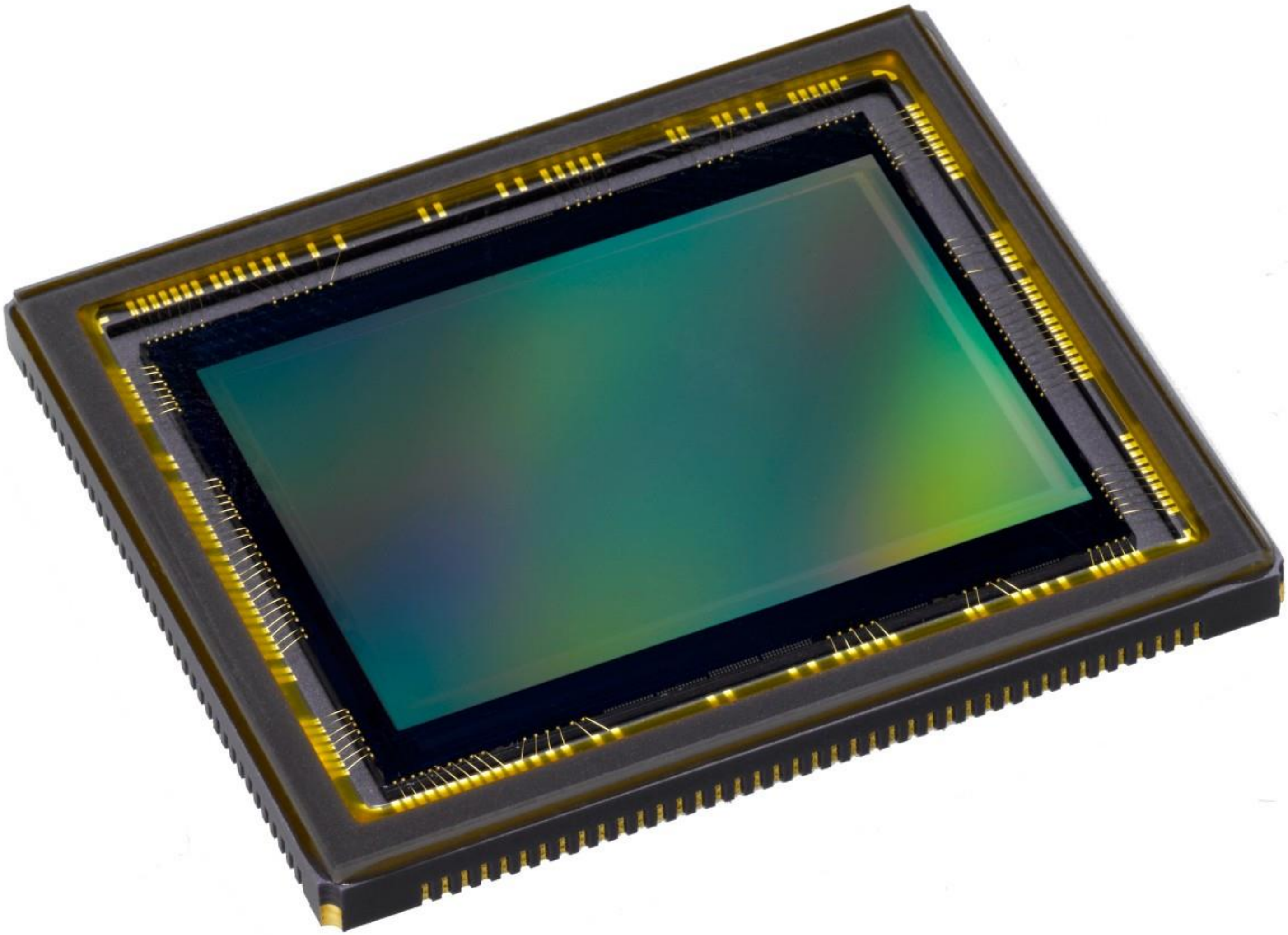

图像即函数
$f: \mathbb{Z}^2 \to \mathbb{R}$


图像即函数
$f: \mathbb{Z}^2 \rightarrow \mathbb{R}^3$

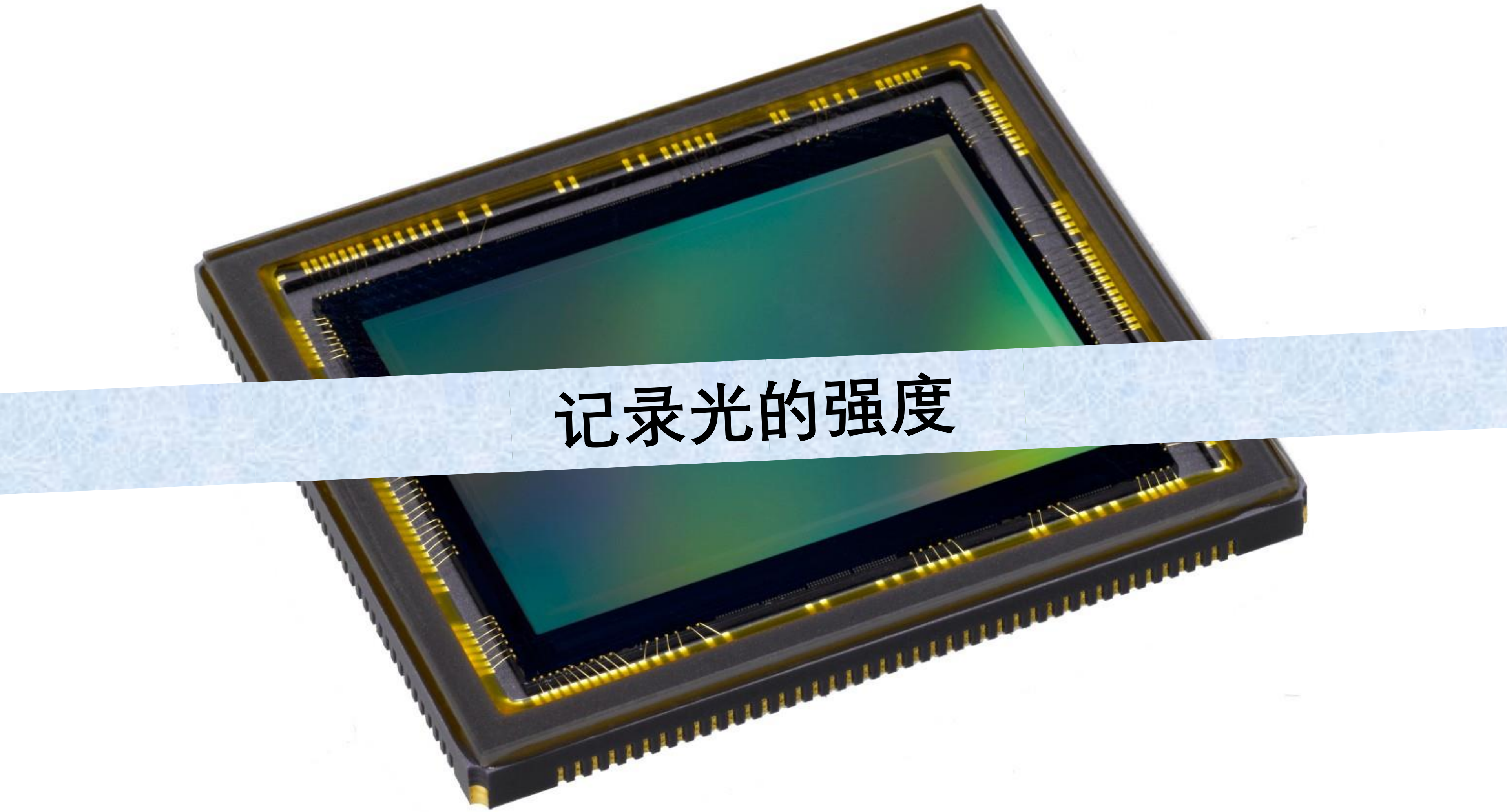

图像即函数
$f: \mathbb{R}^2 \to \mathbb{R}^3$



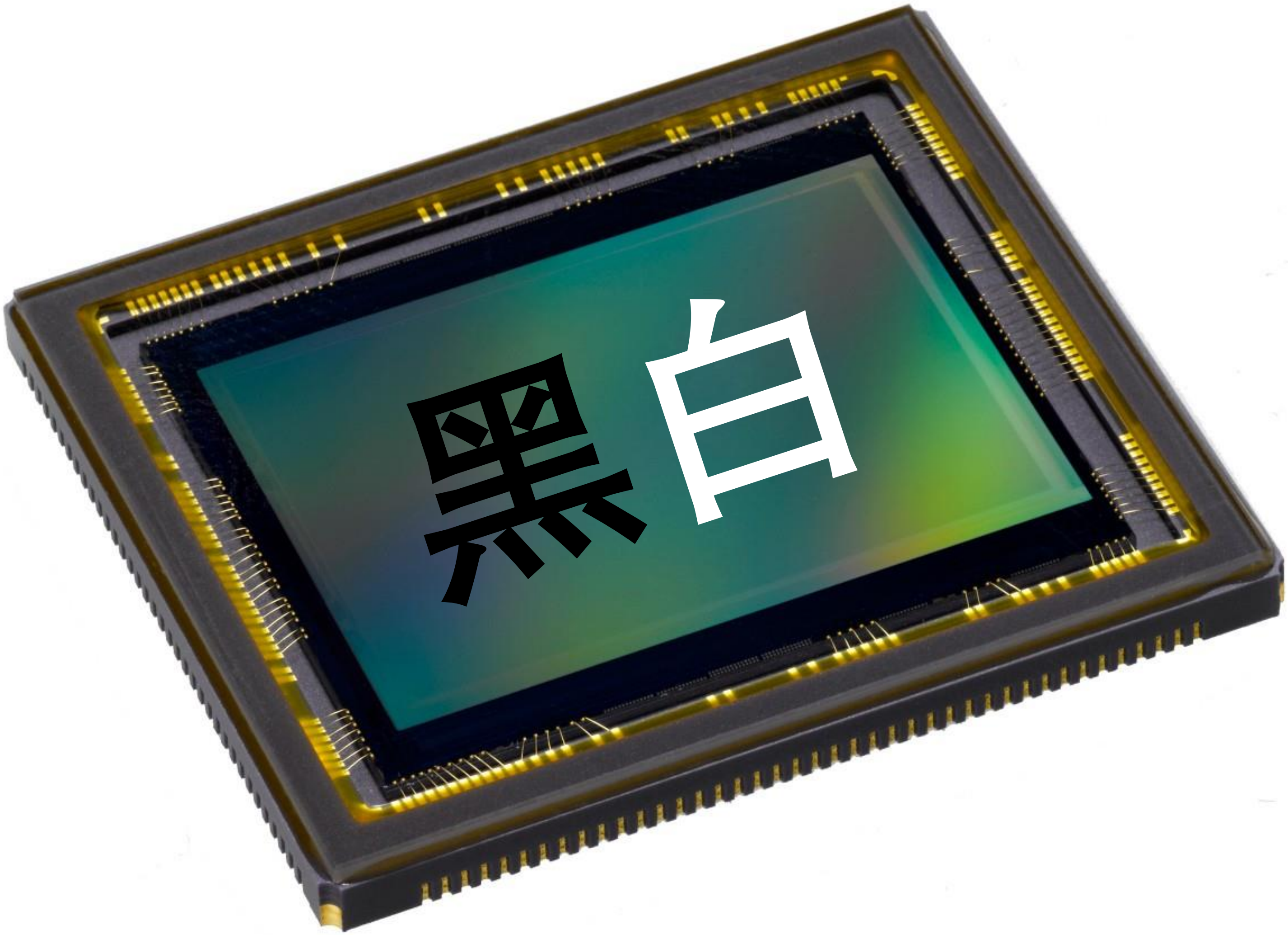
像平面
$\hat{Y}$
$\mathbf{p} = (x, y, f)$
表面
$\hat{Z}$
o
光轴
$\mathbf{P} = (X, Y, Z)$
$\hat{X}$



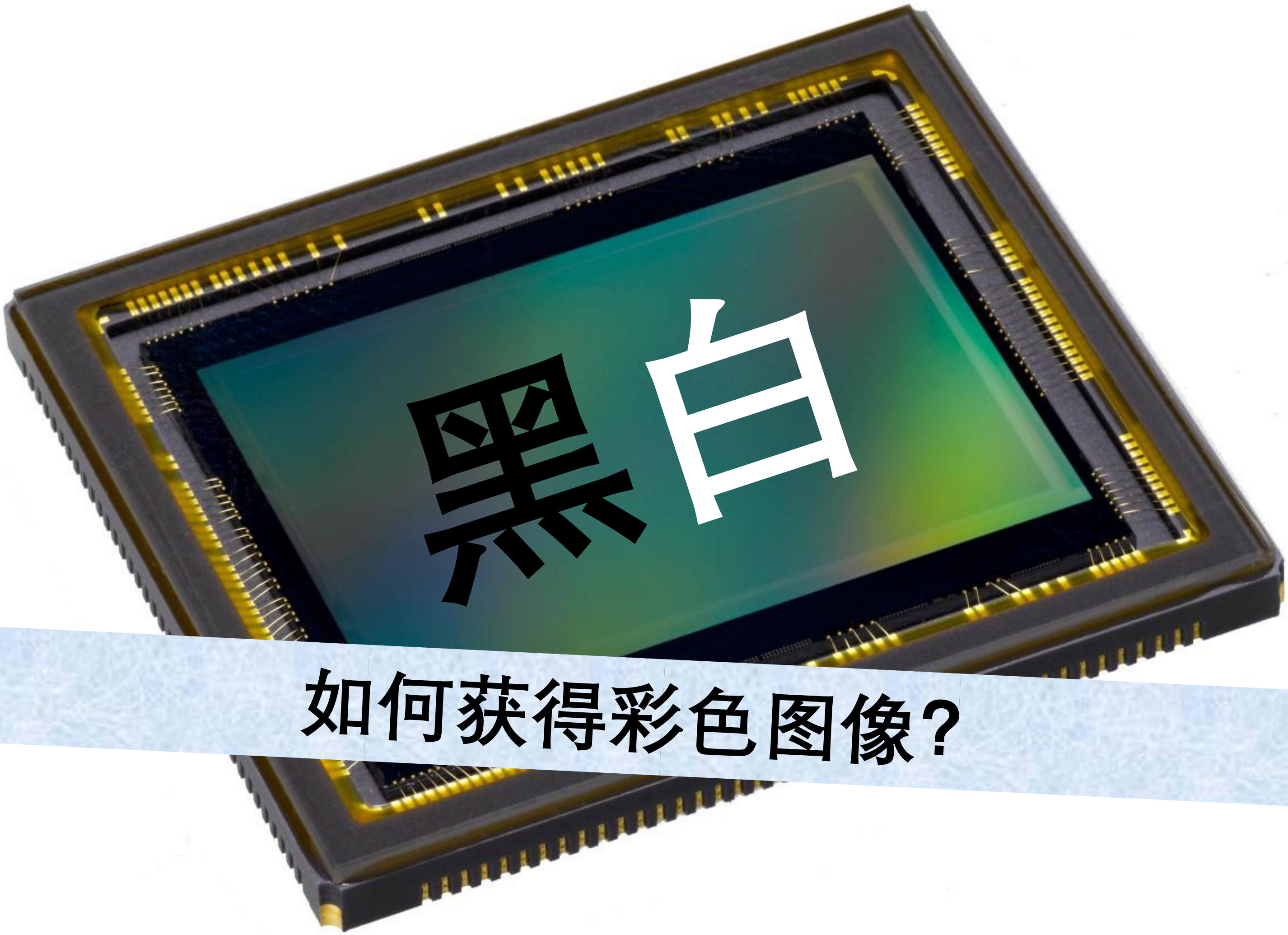
电荷耦合器件（CCD）



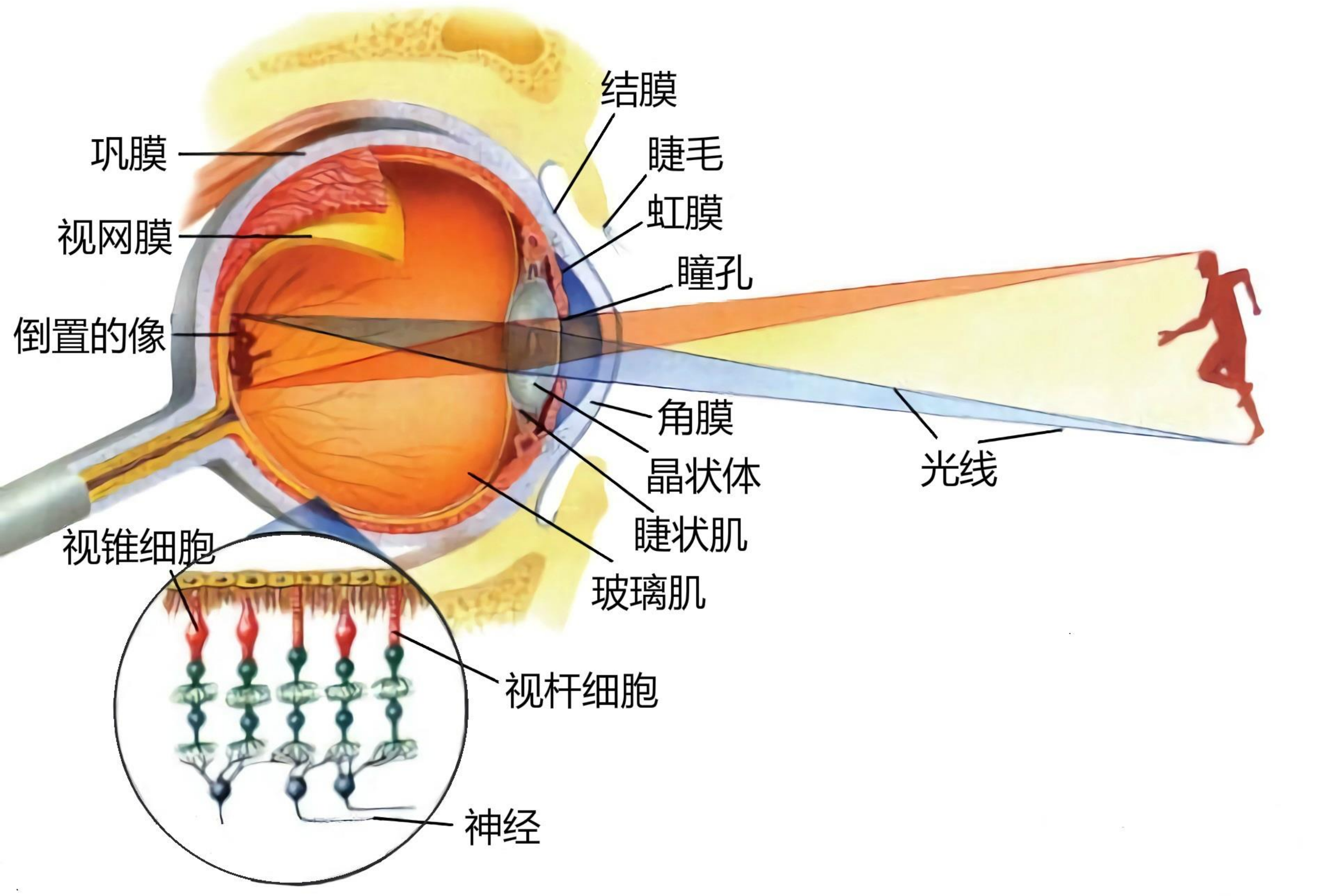
电荷耦合器件（CCD）

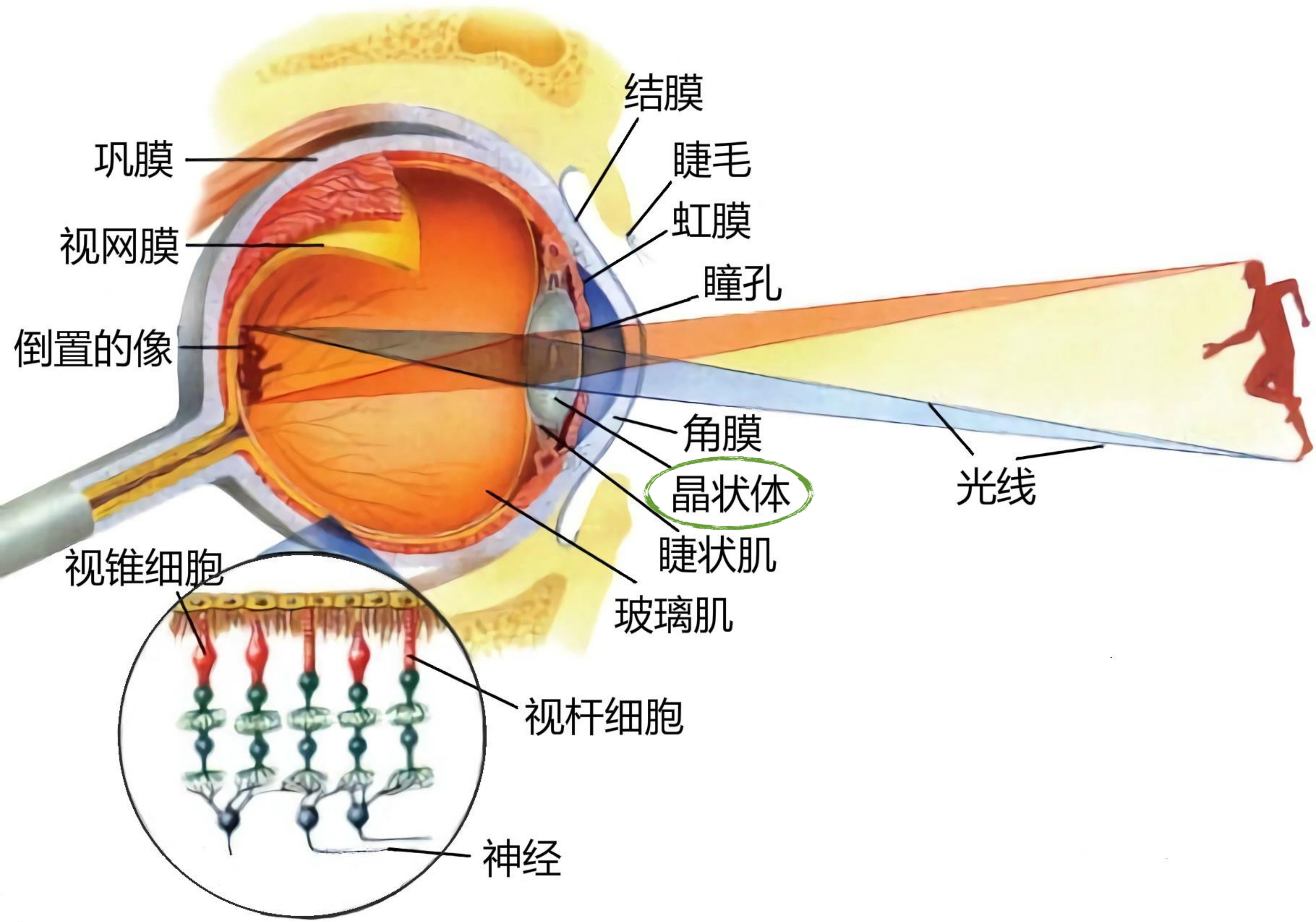


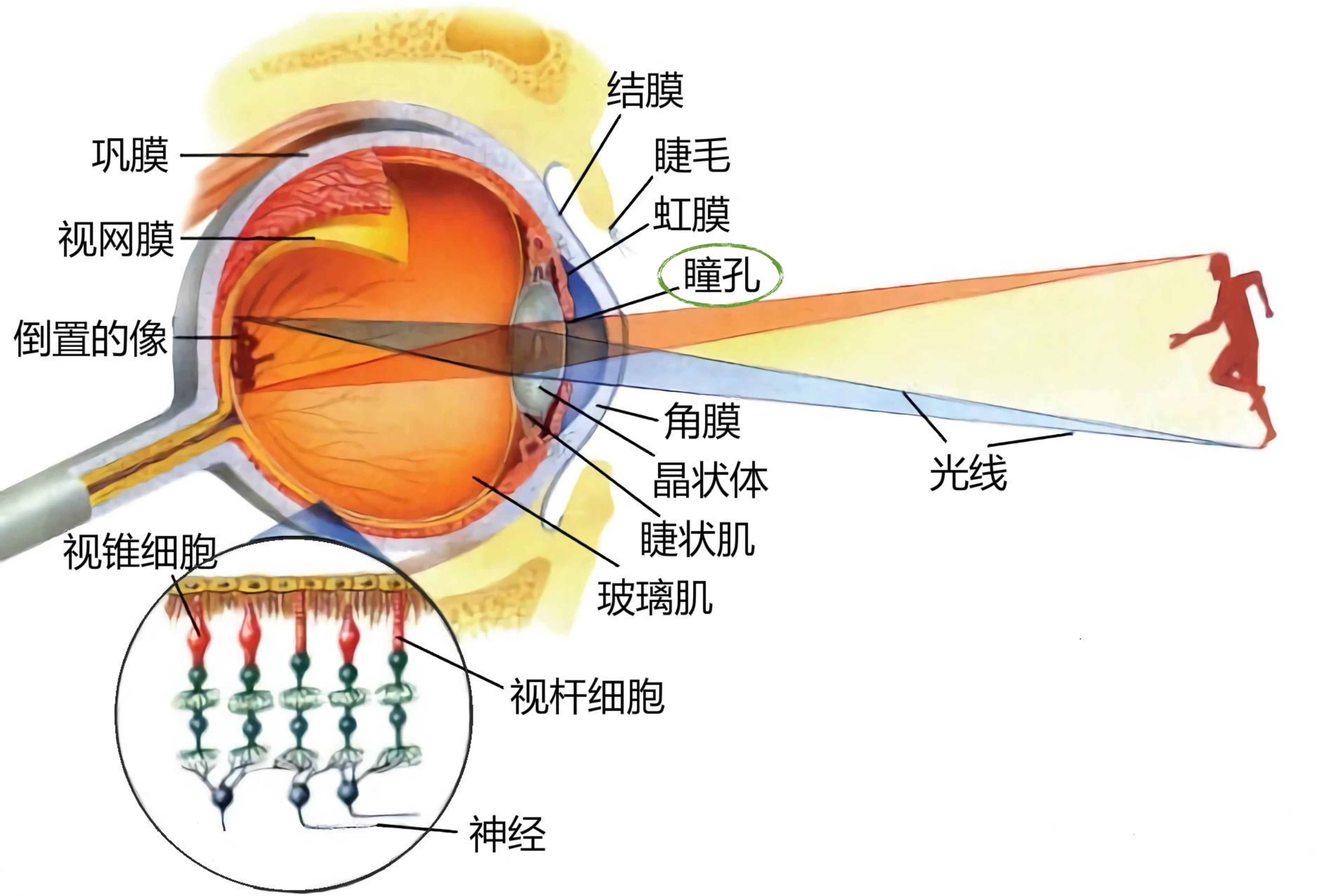
电荷耦合器件（CCD）



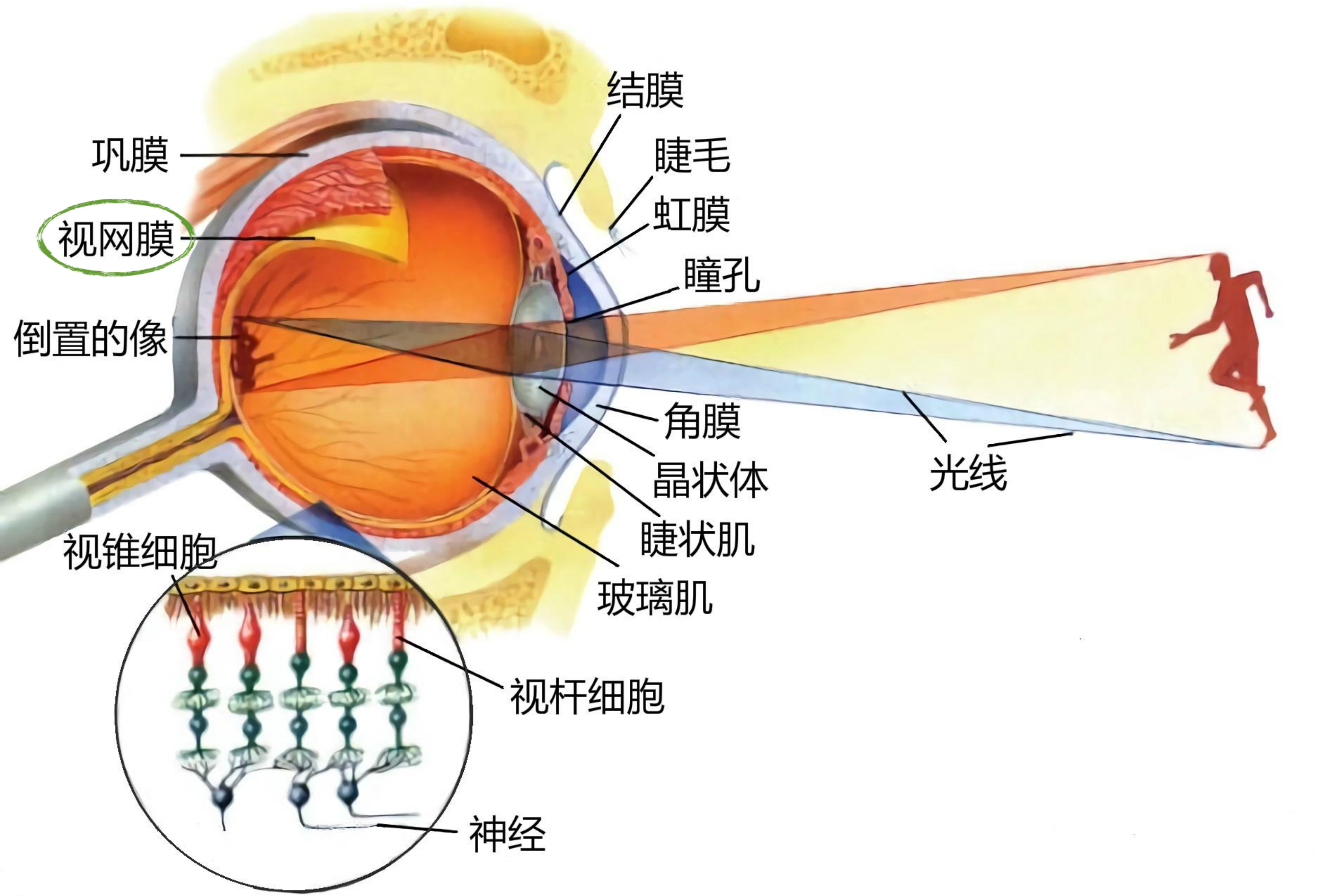
电荷耦合器件（CCD）

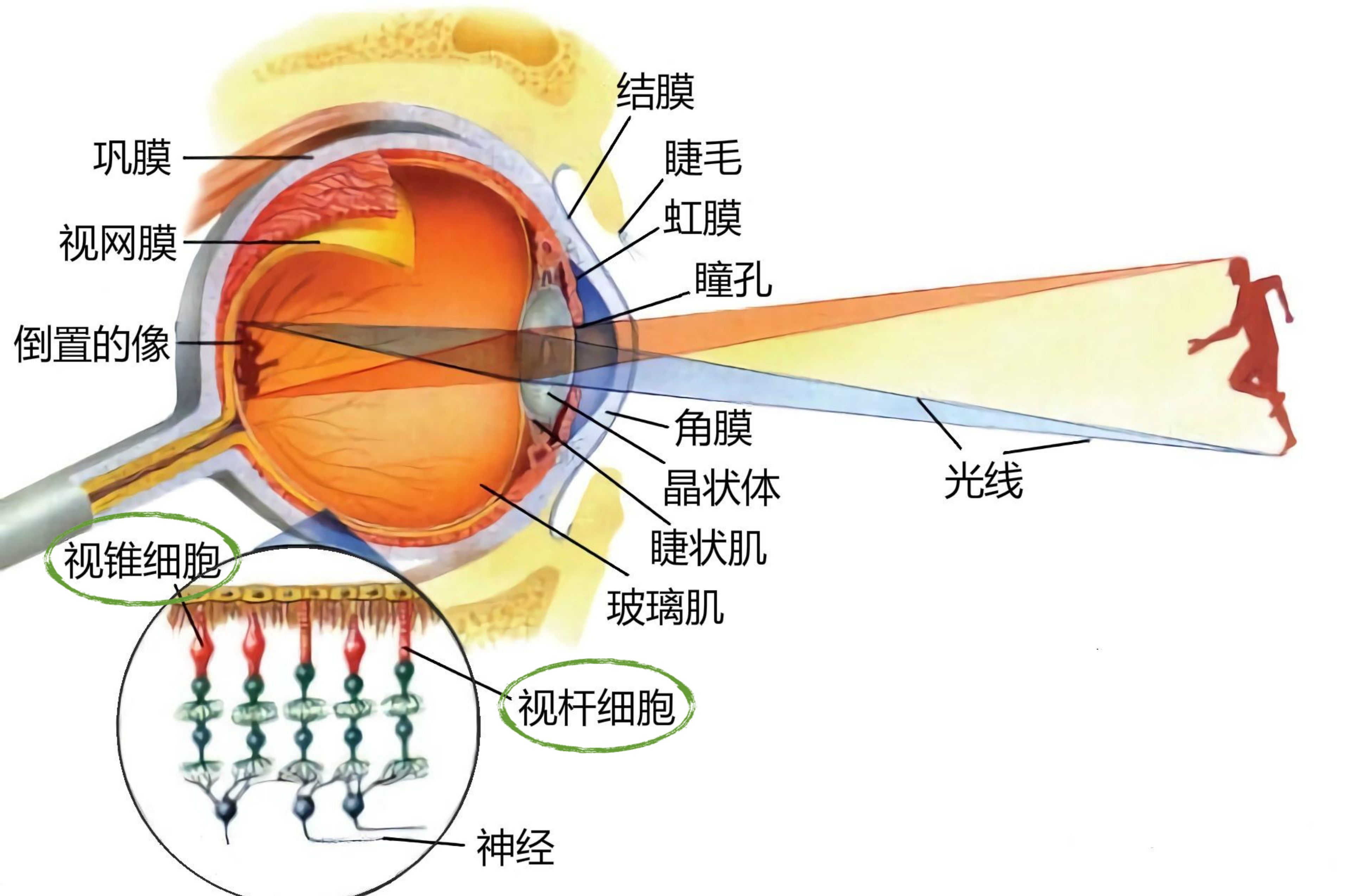

结膜
巩膜
睫毛
虹膜
视网膜
瞳孔
倒置的像
角膜
晶状体
光线
睫状肌
视锥细胞
玻璃肌
视杆细胞
神经

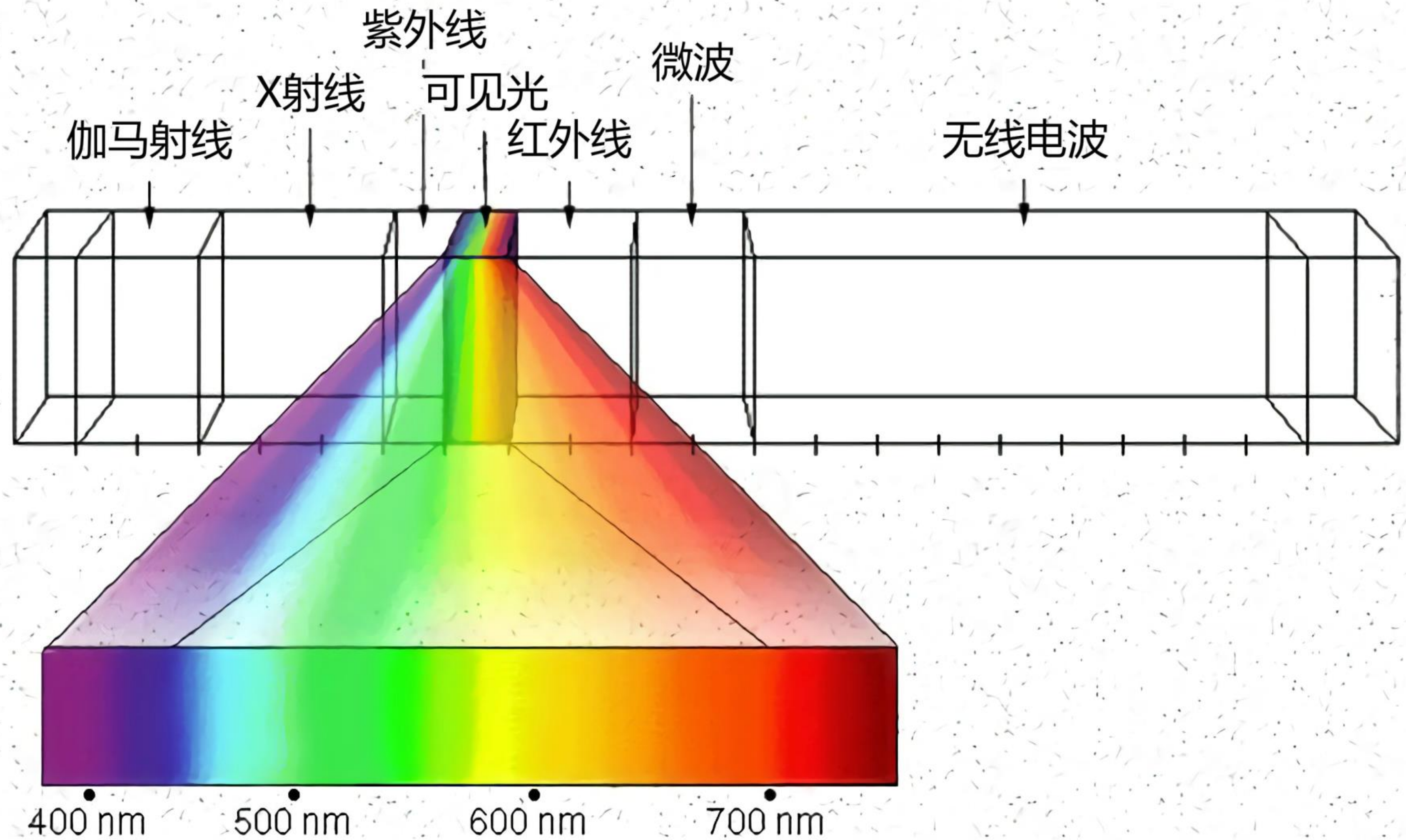

结膜
巩膜
睫毛
虹膜
视网膜
瞳孔
倒置的像
角膜
晶状体
光线
睫状肌
视锥细胞
玻璃肌
视杆细胞
神经

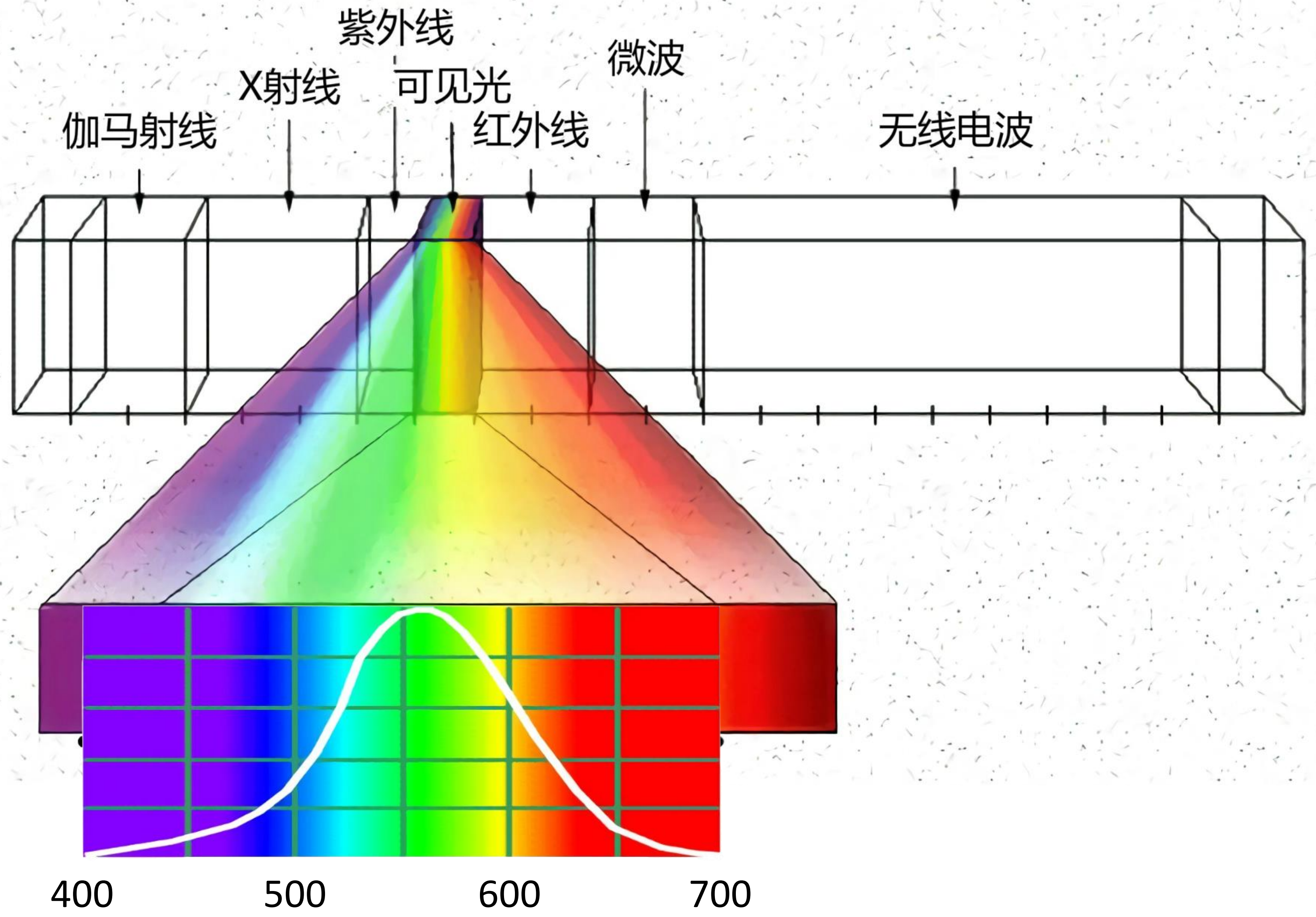


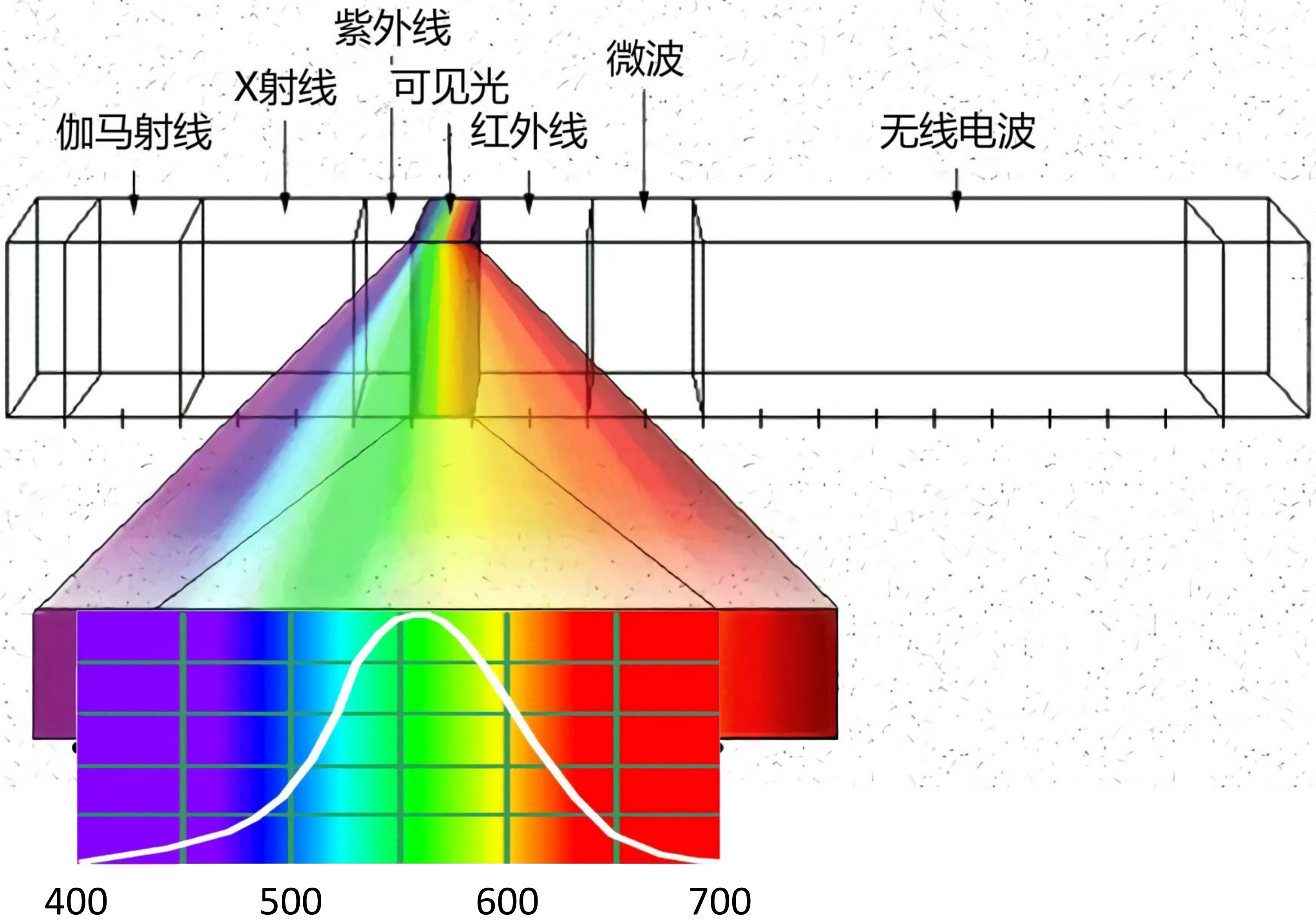
结膜
巩膜
睫毛
虹膜
视网膜
瞳孔
倒置的像
角膜
晶状体
光线
睫状肌
玻璃肌
视锥细胞
视杆细胞
神经


结膜
巩膜
睫毛
虹膜
视网膜
瞳孔
倒置的像
角膜
晶状体
光线
睫状肌
视锥细胞
玻璃肌
视杆细胞
神经


结膜
巩膜
睫毛
虹膜
视网膜
瞳孔
倒置的像
角膜
晶状体
光线
睫状肌
玻璃肌
视锥细胞
视杆细胞
神经

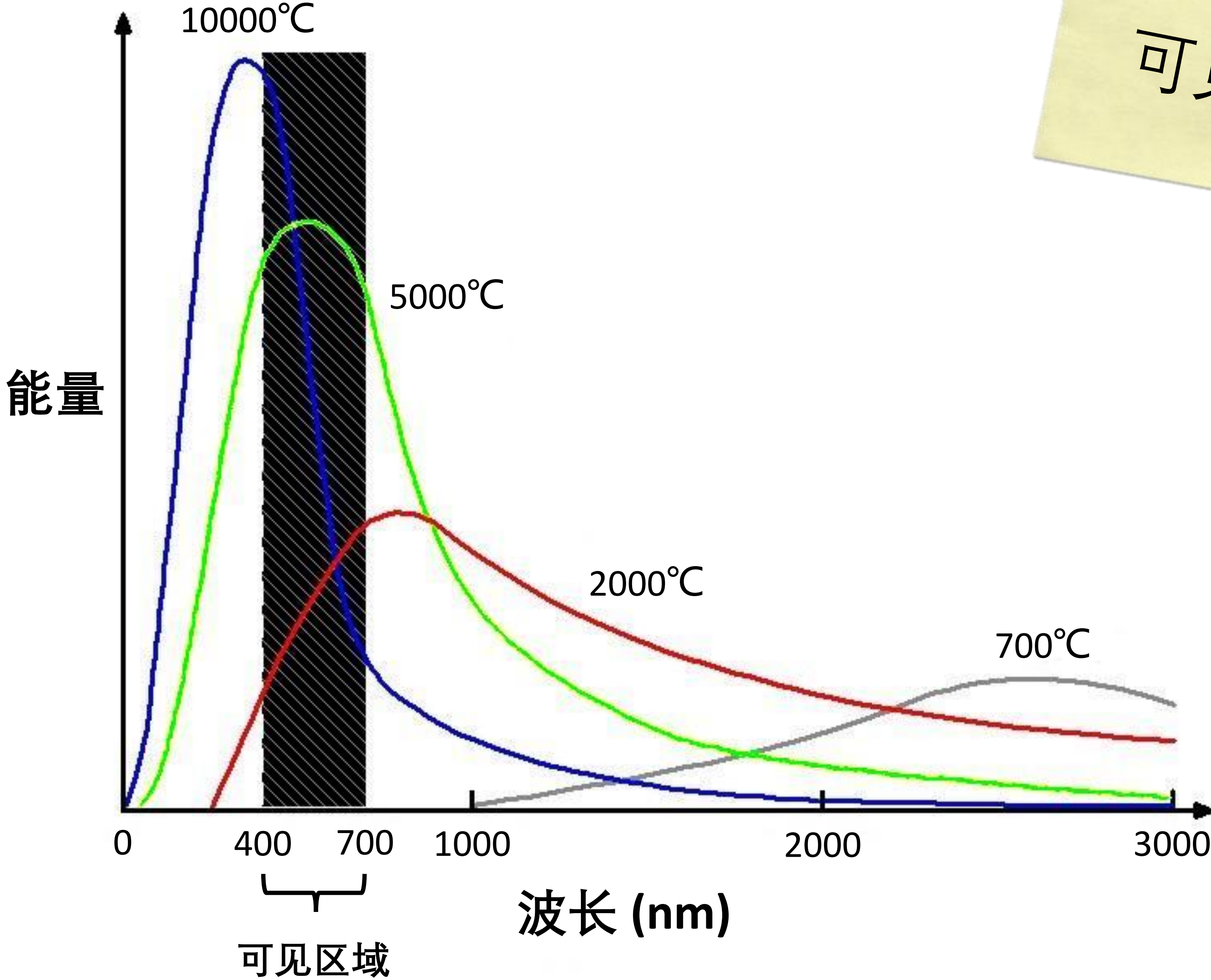

紫外线
X射线
可见光
微波
伽马射线
红外线
无线电波
400 nm
500 nm
600 nm
700 nm


紫外线
微波
X射线
可见光
伽马射线
红外线
无线电波
400
500
600
700

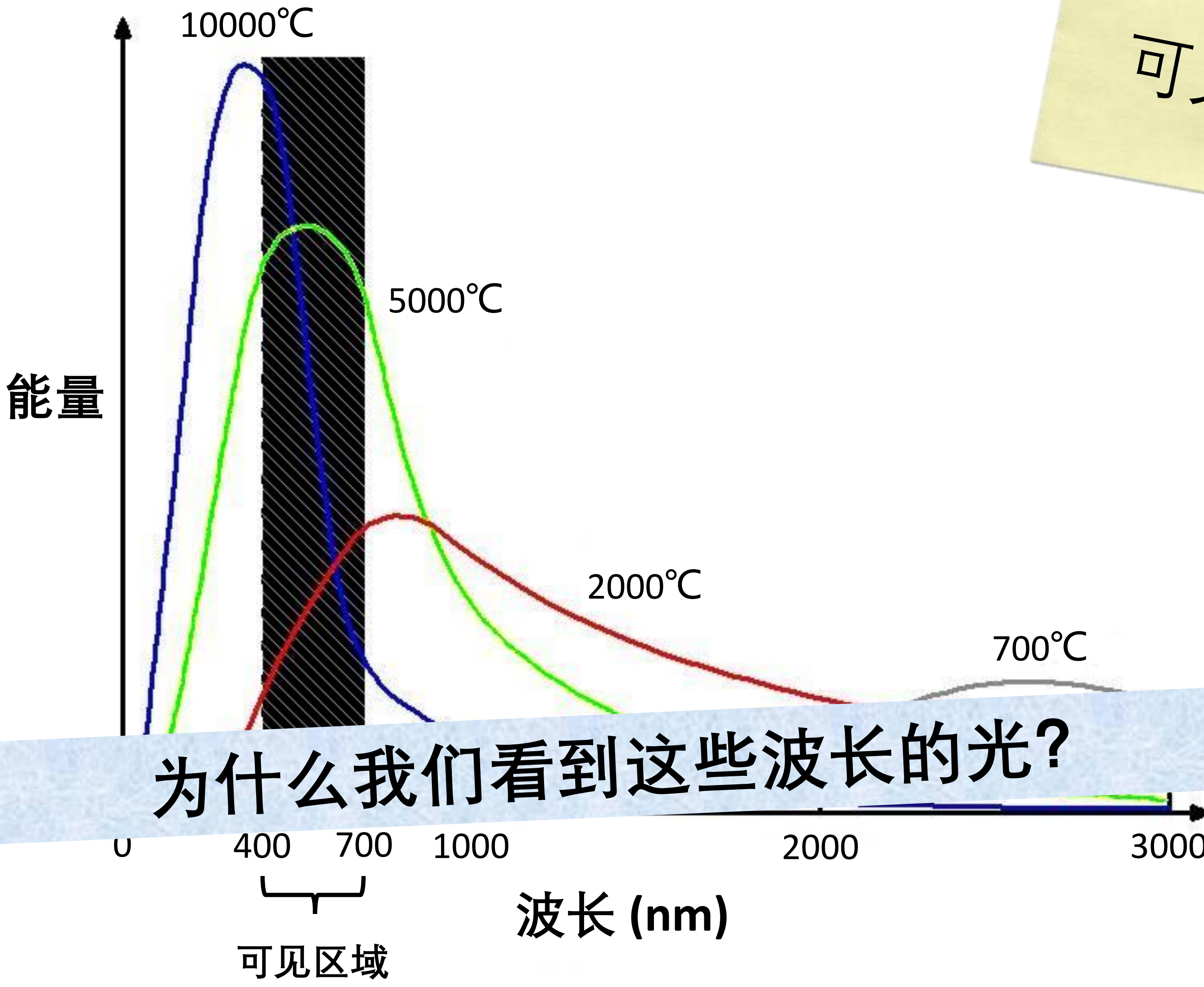


人眼光度敏感函数

可见光

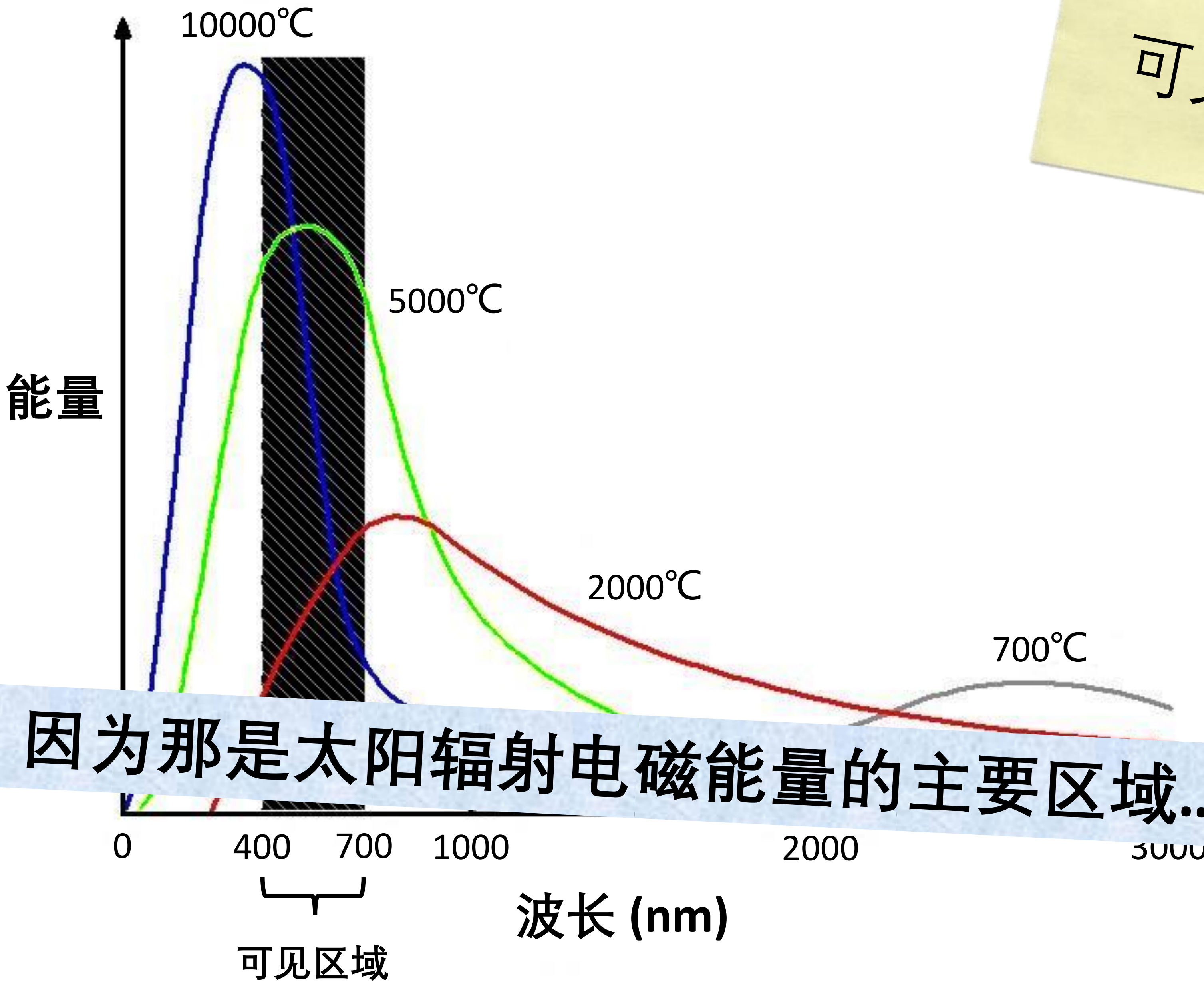


可见光

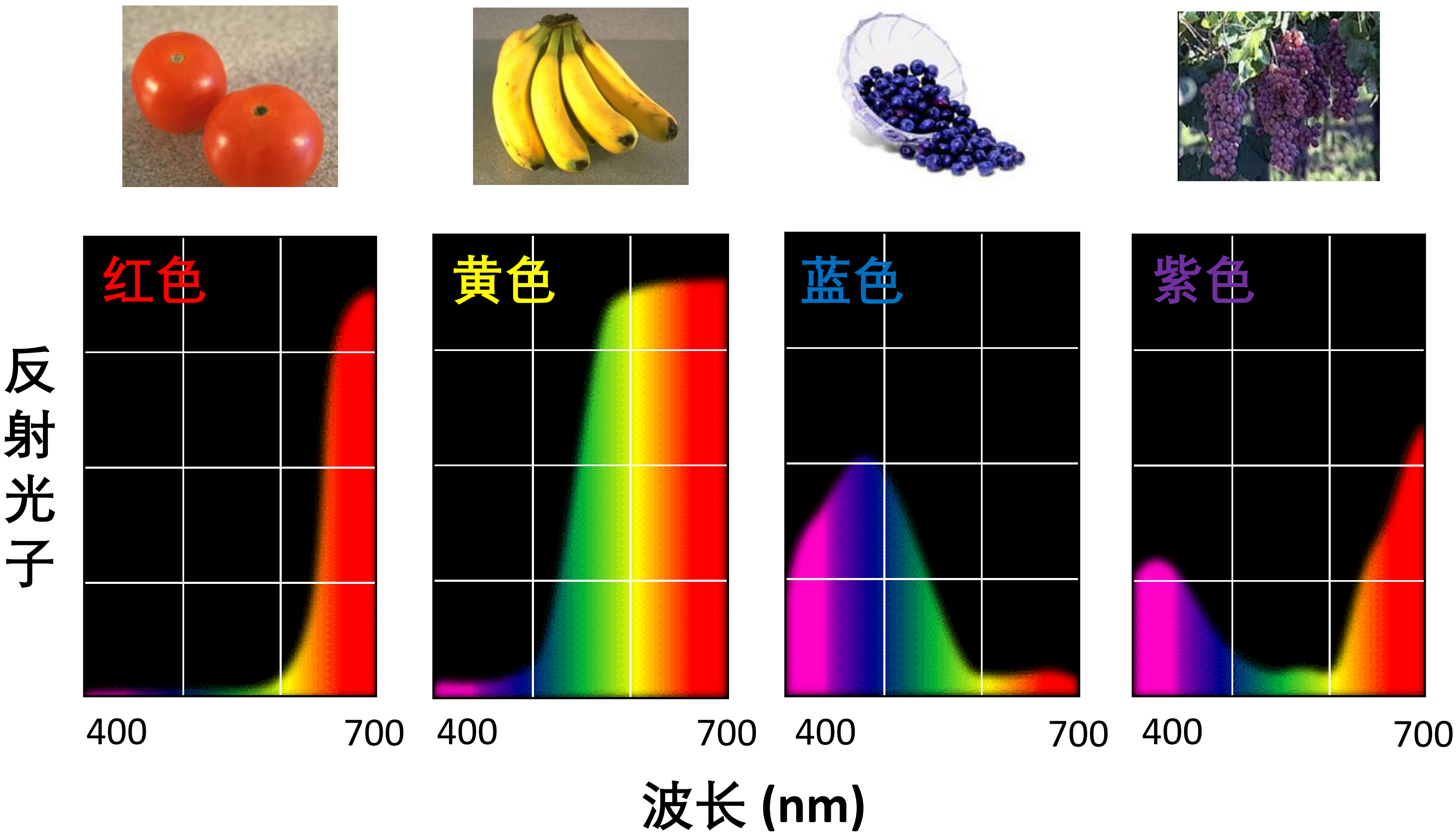


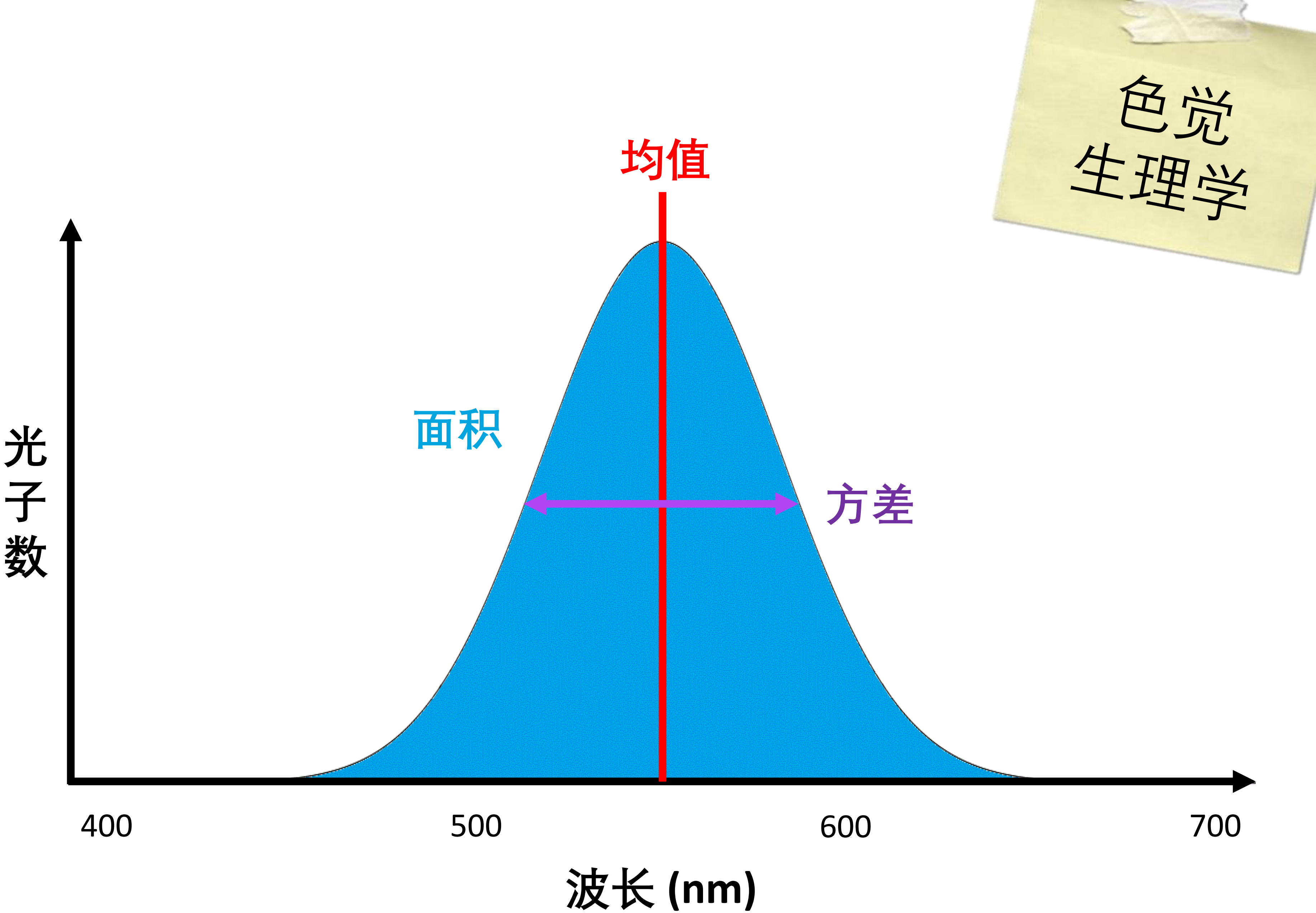
为什么我们看到这些波长的光?

可见光

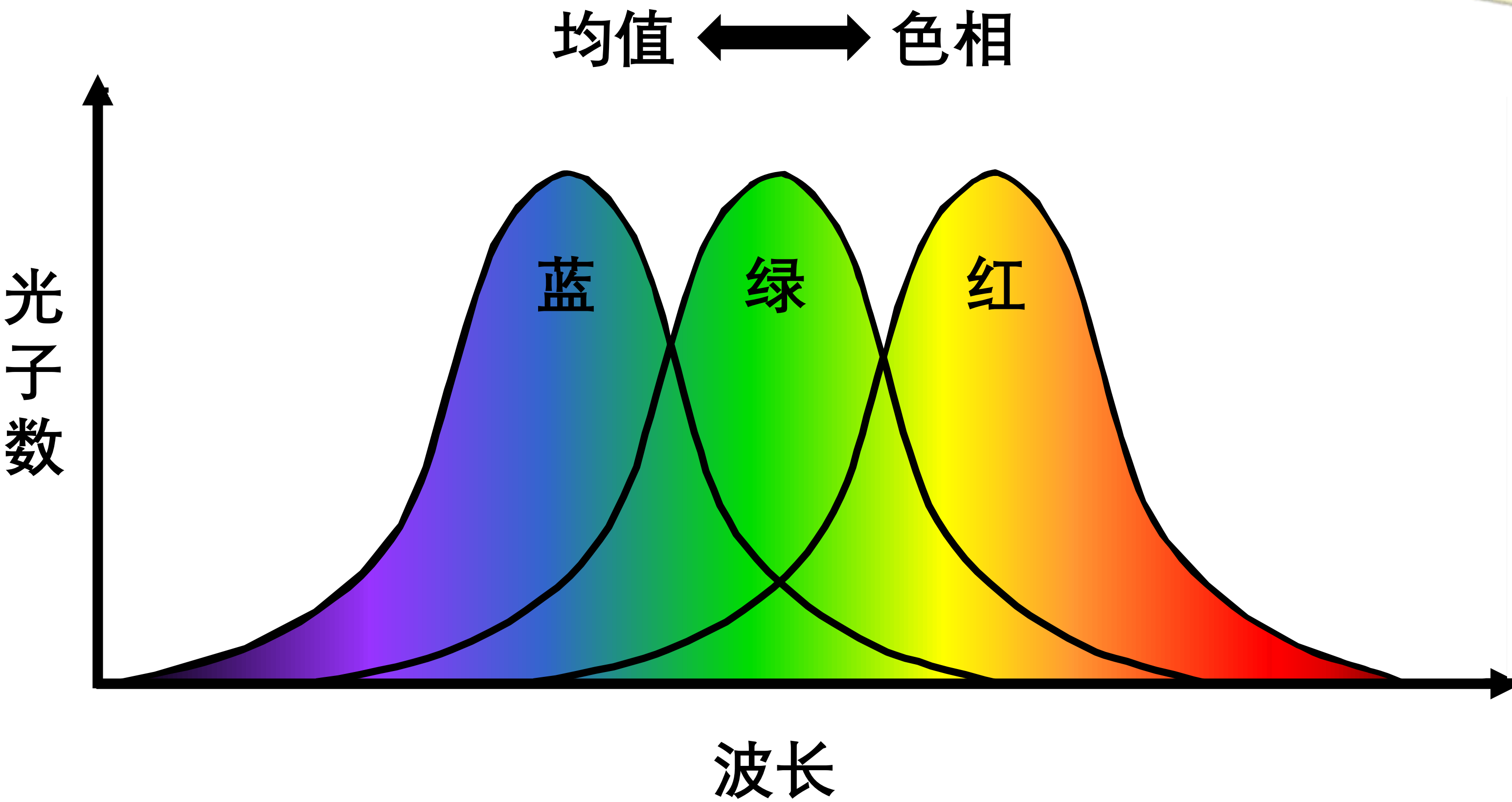


因为那是太阳辐射电磁能量的主要区域…


红色
黄色
蓝色
紫色
反射光子
400
700
400
700
400
700
400
700
波长 (nm)

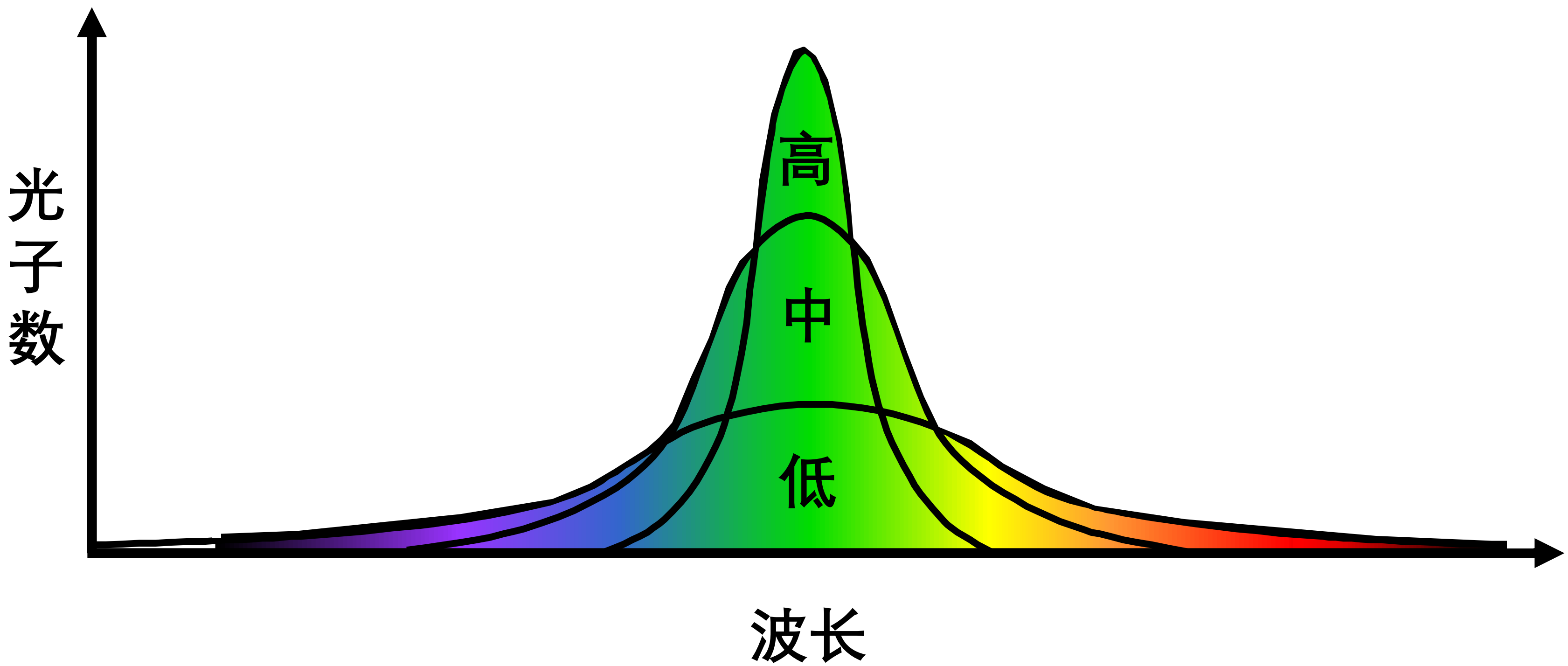

色觉
生理学
均值
面积
方差
光子数
400
500
600
700
波长 (nm)

色觉
生理学

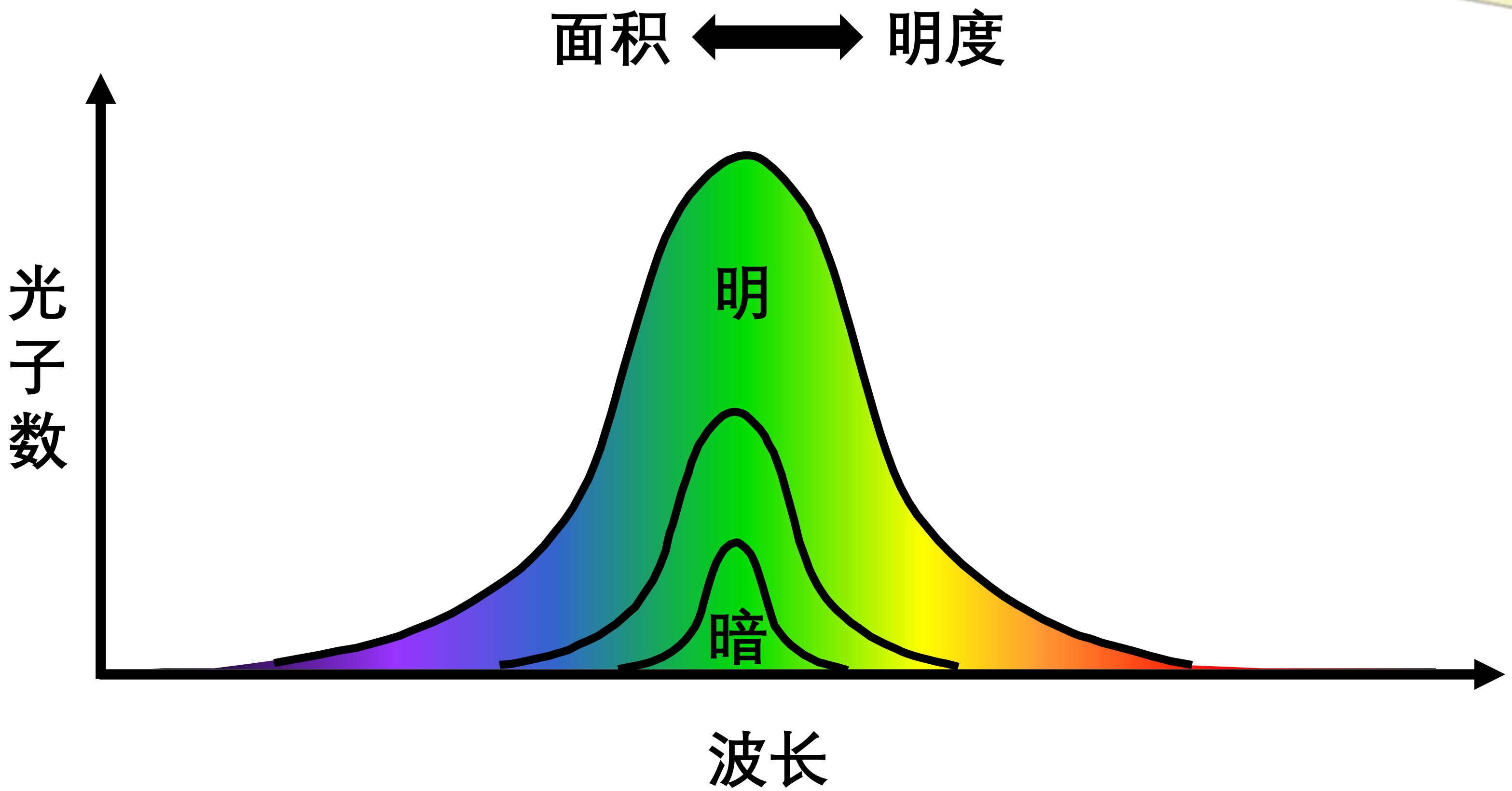


色觉生理学

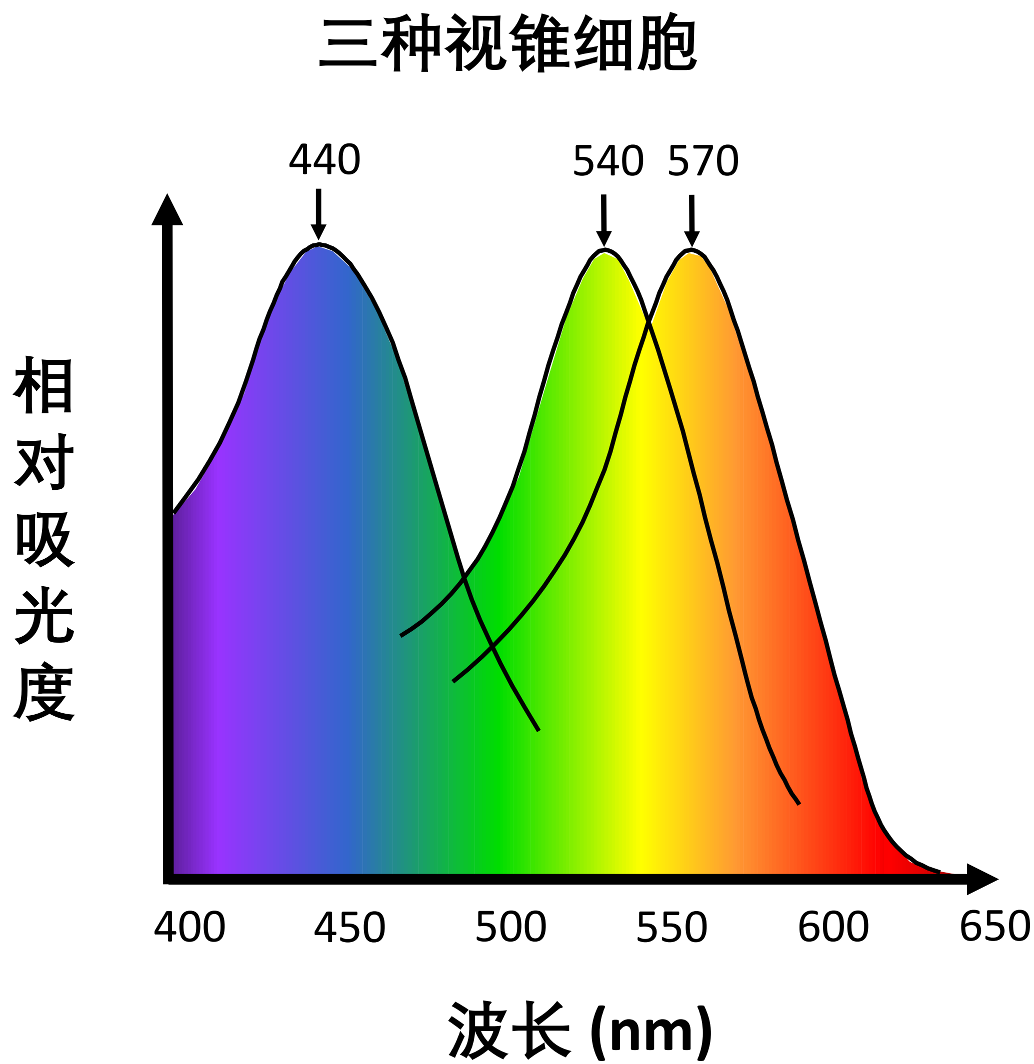
方差 ⟷ 饱和度



色觉
生理学

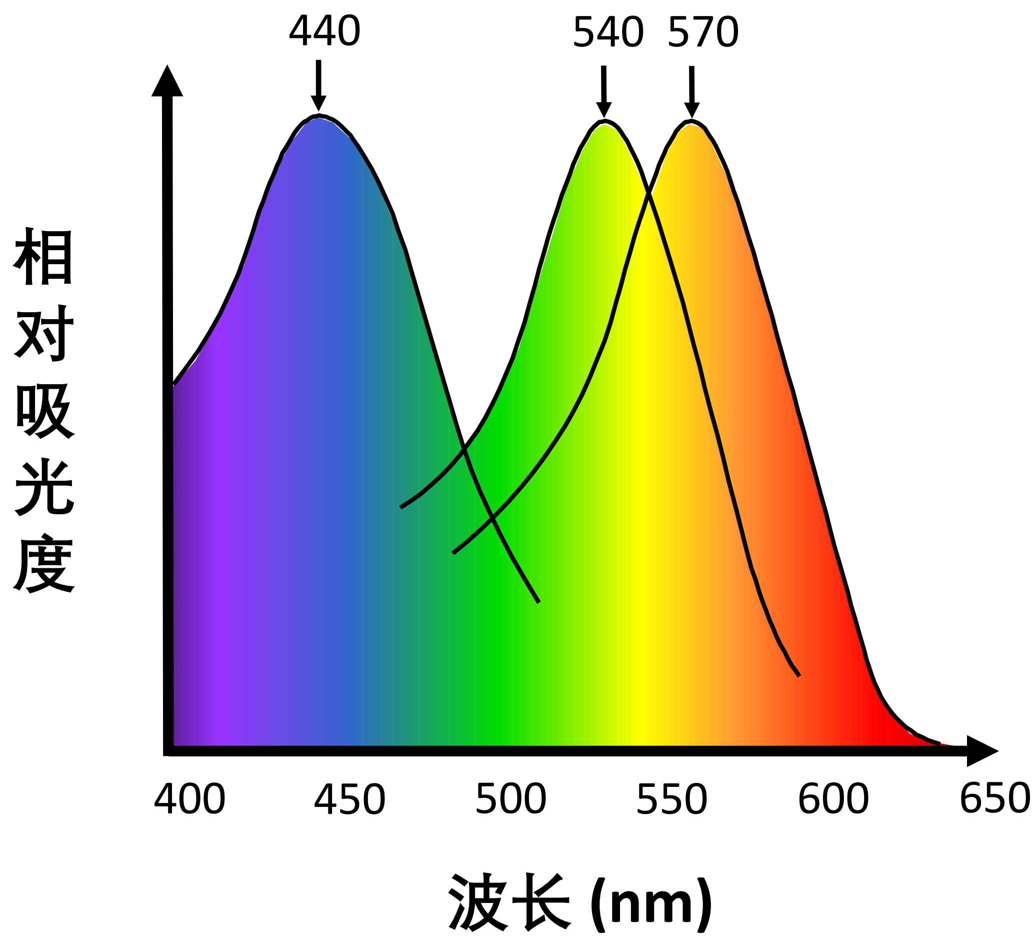


色觉生理学

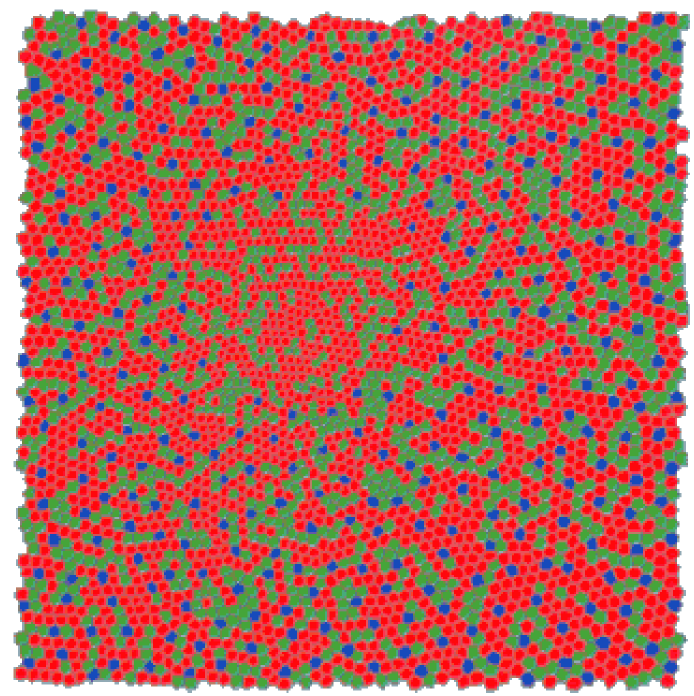


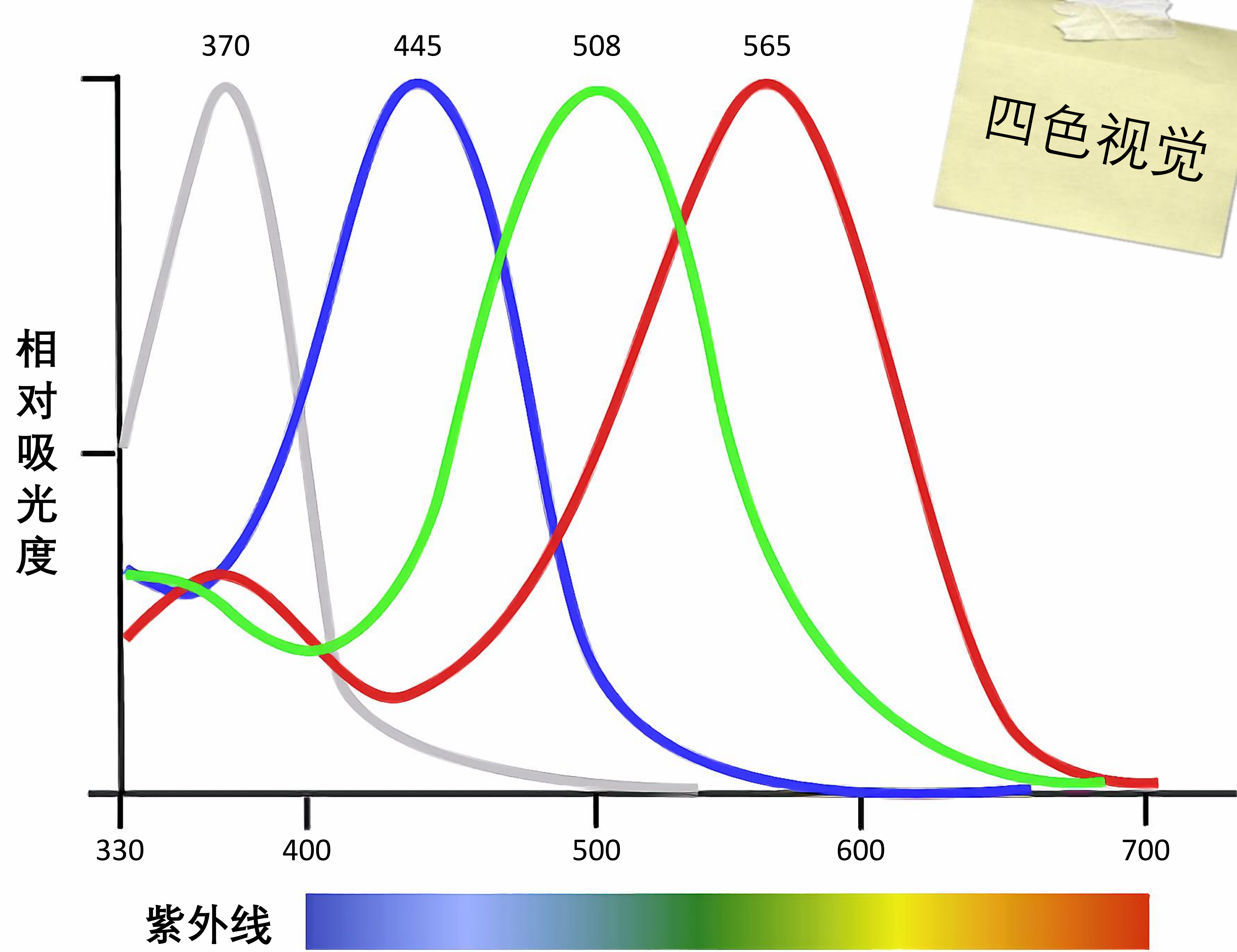
色觉
生理学

三种视锥细胞



视锥细胞镶嵌

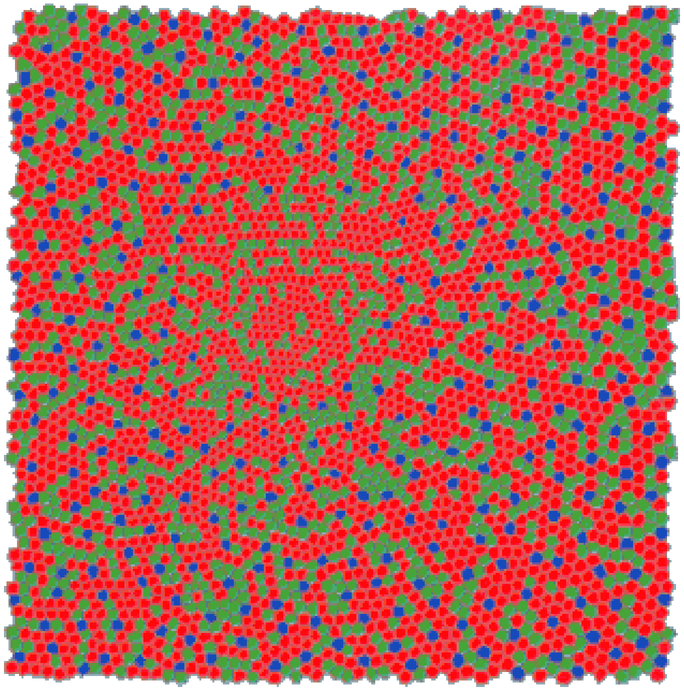




四色视觉
370
445
508
565
相对吸光度
330
400
500
600
700
紫外线

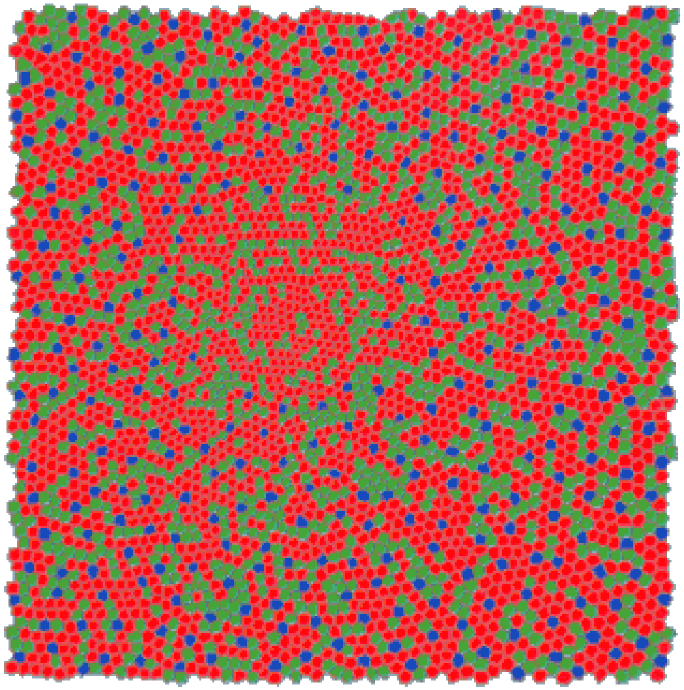
相机色彩感知

视锥细胞镶嵌

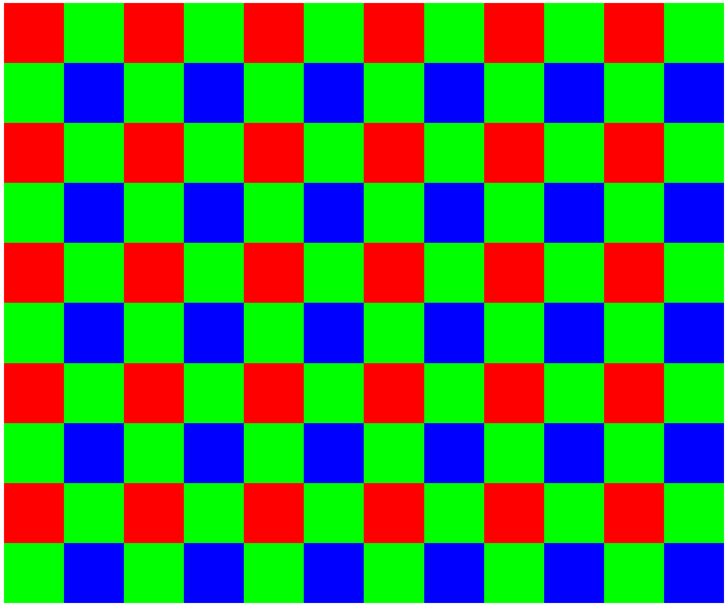


相机色彩感知

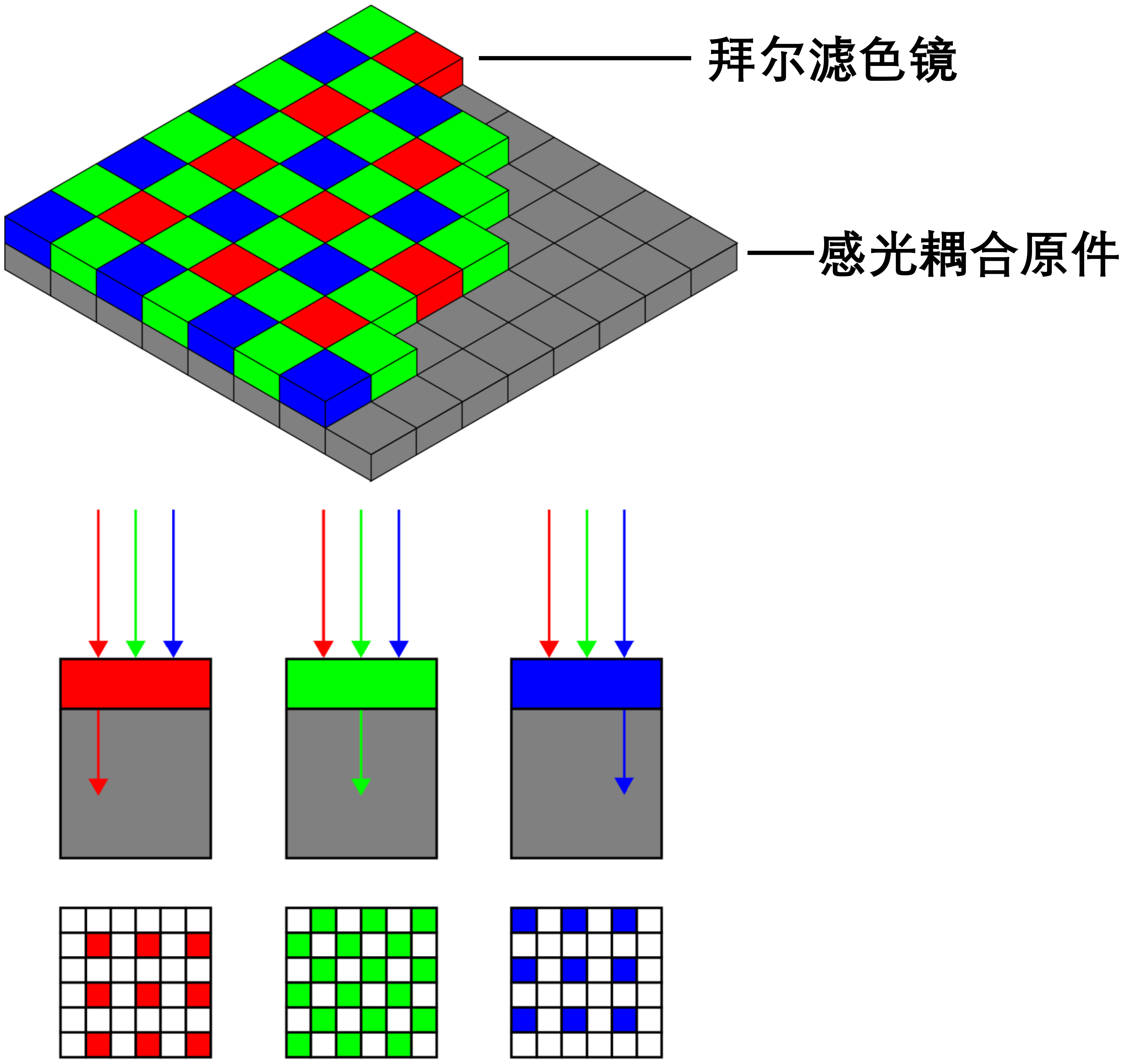
视锥细胞镶嵌

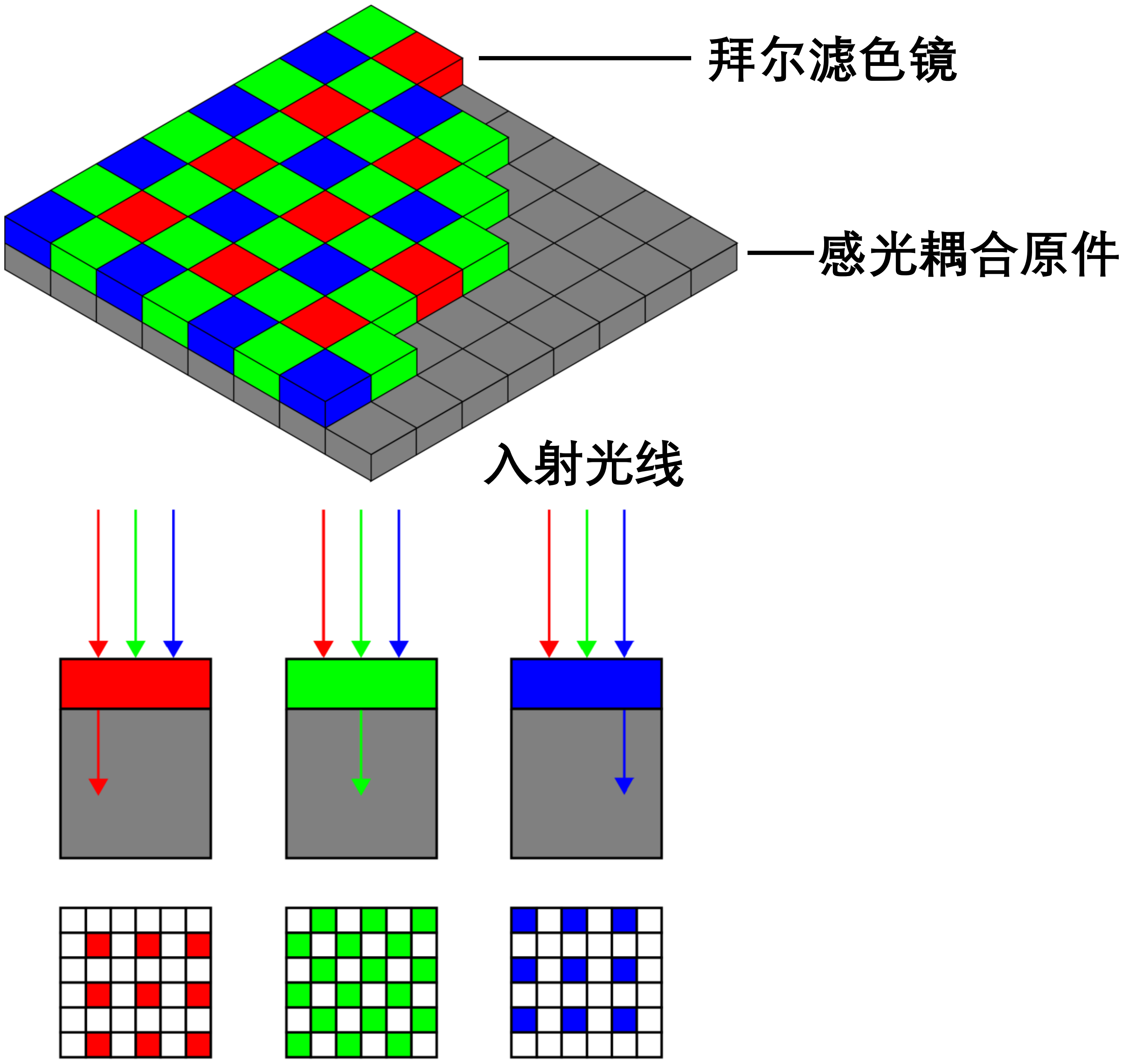


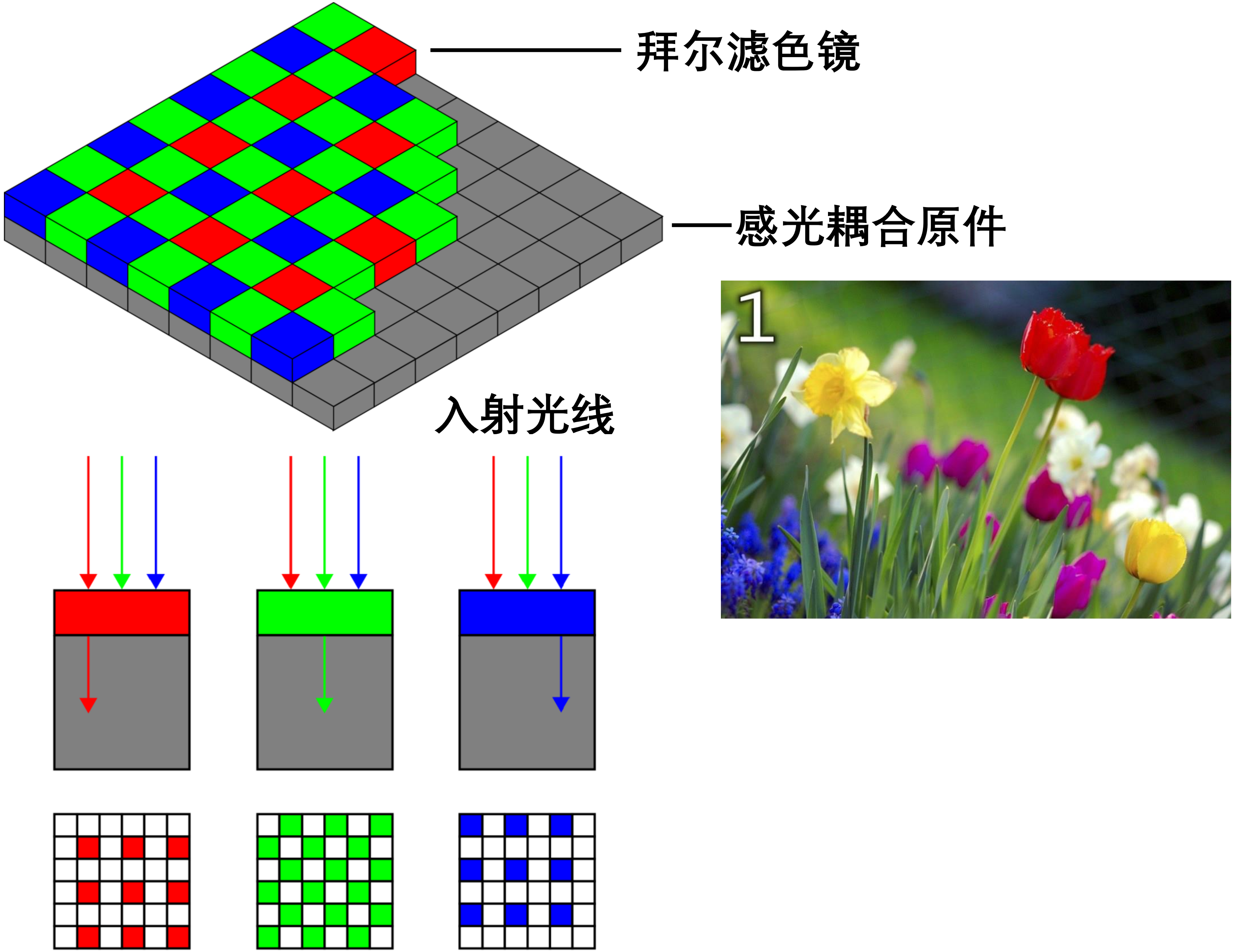
拜尔滤色镜

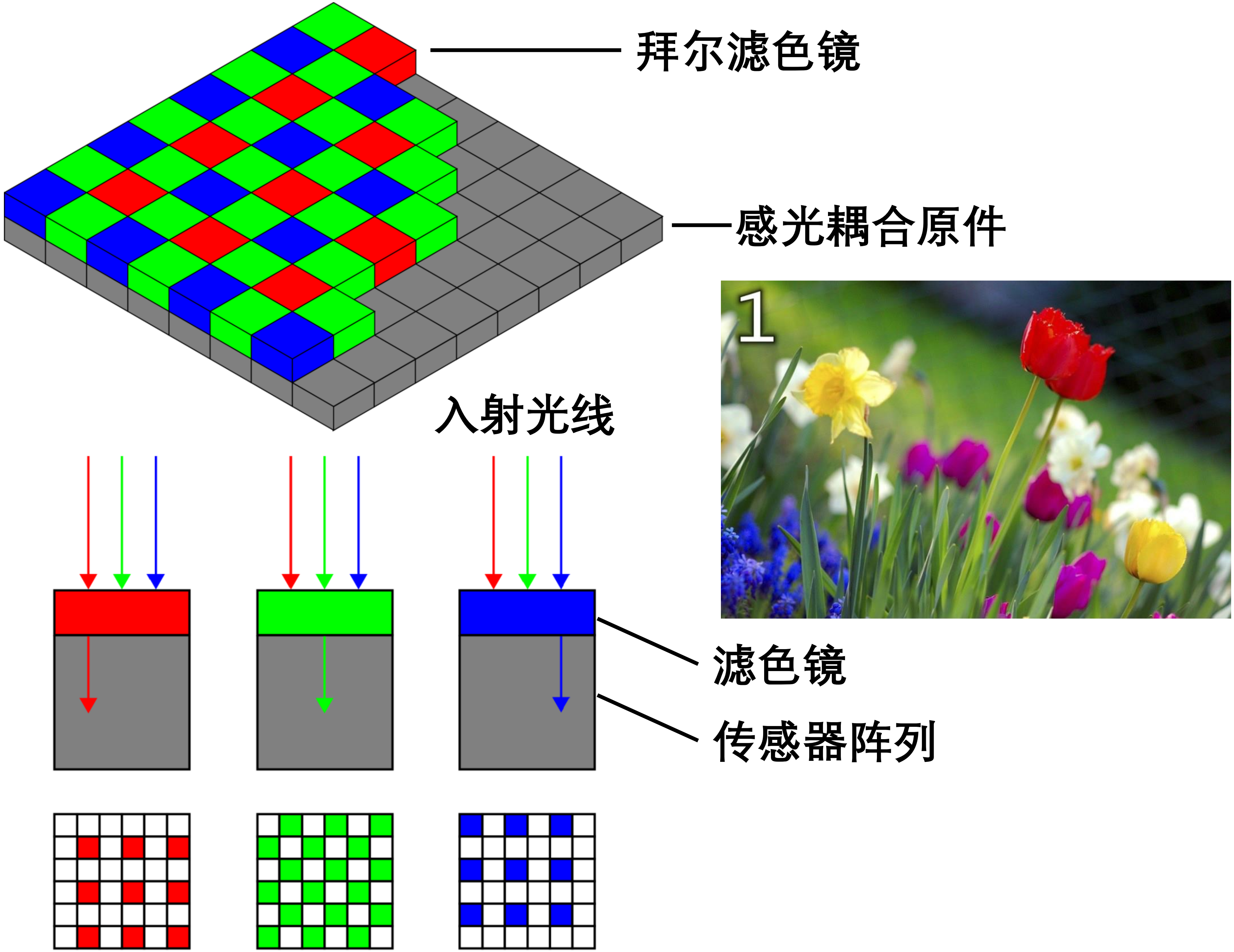


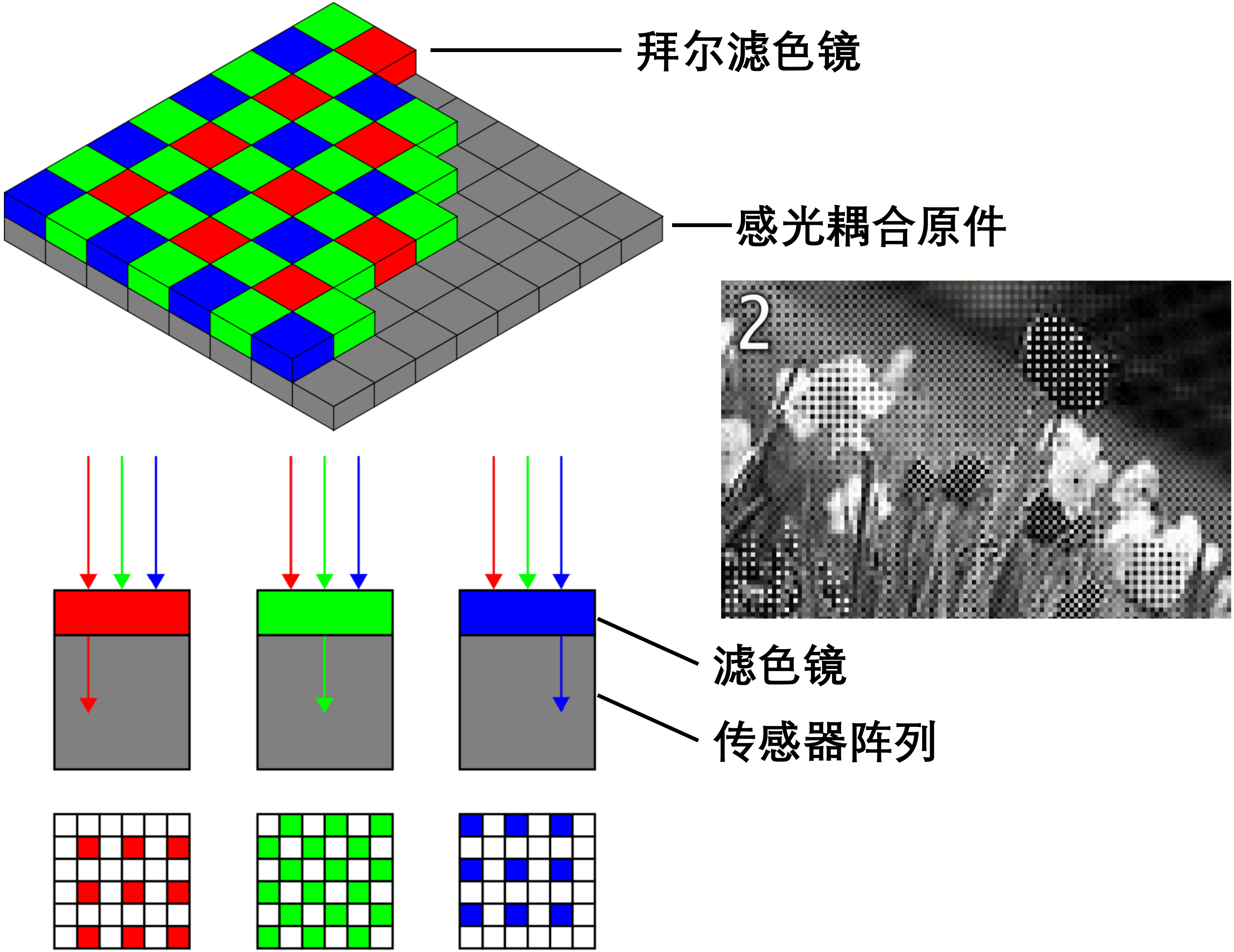
绿色：50%
红色：25%
蓝色：25%

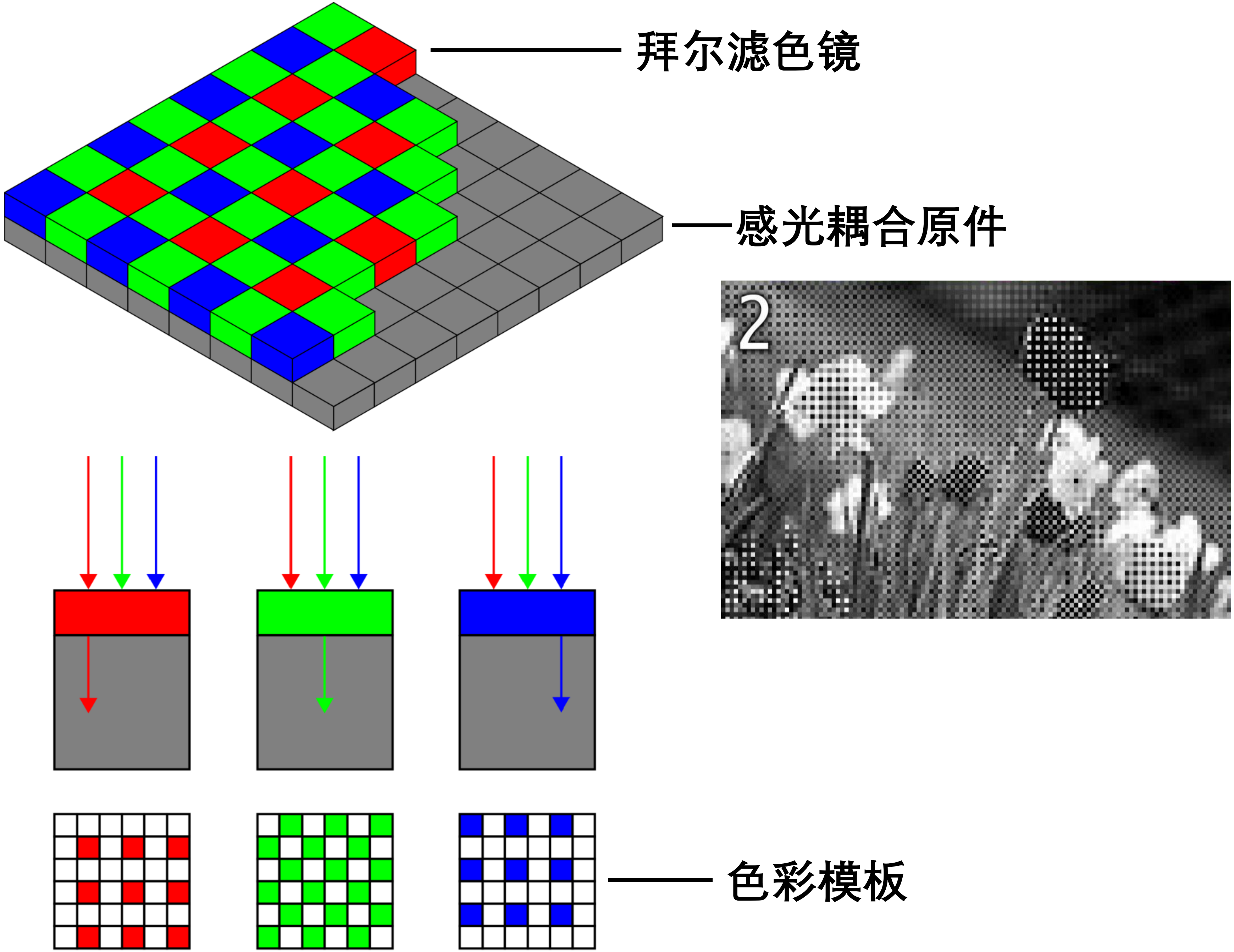

拜尔滤色镜
感光耦合原件

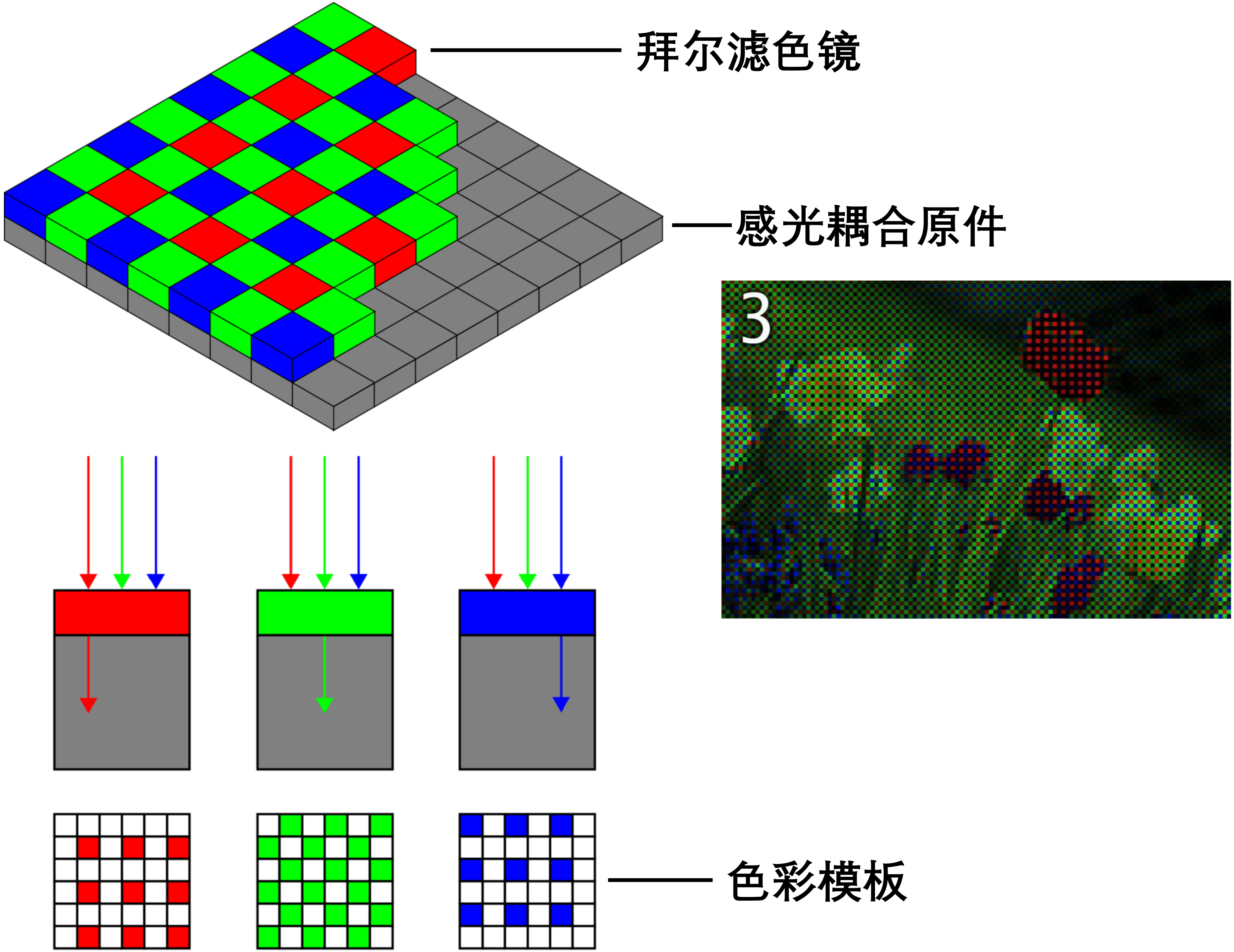

拜尔滤色镜
感光耦合原件
入射光线

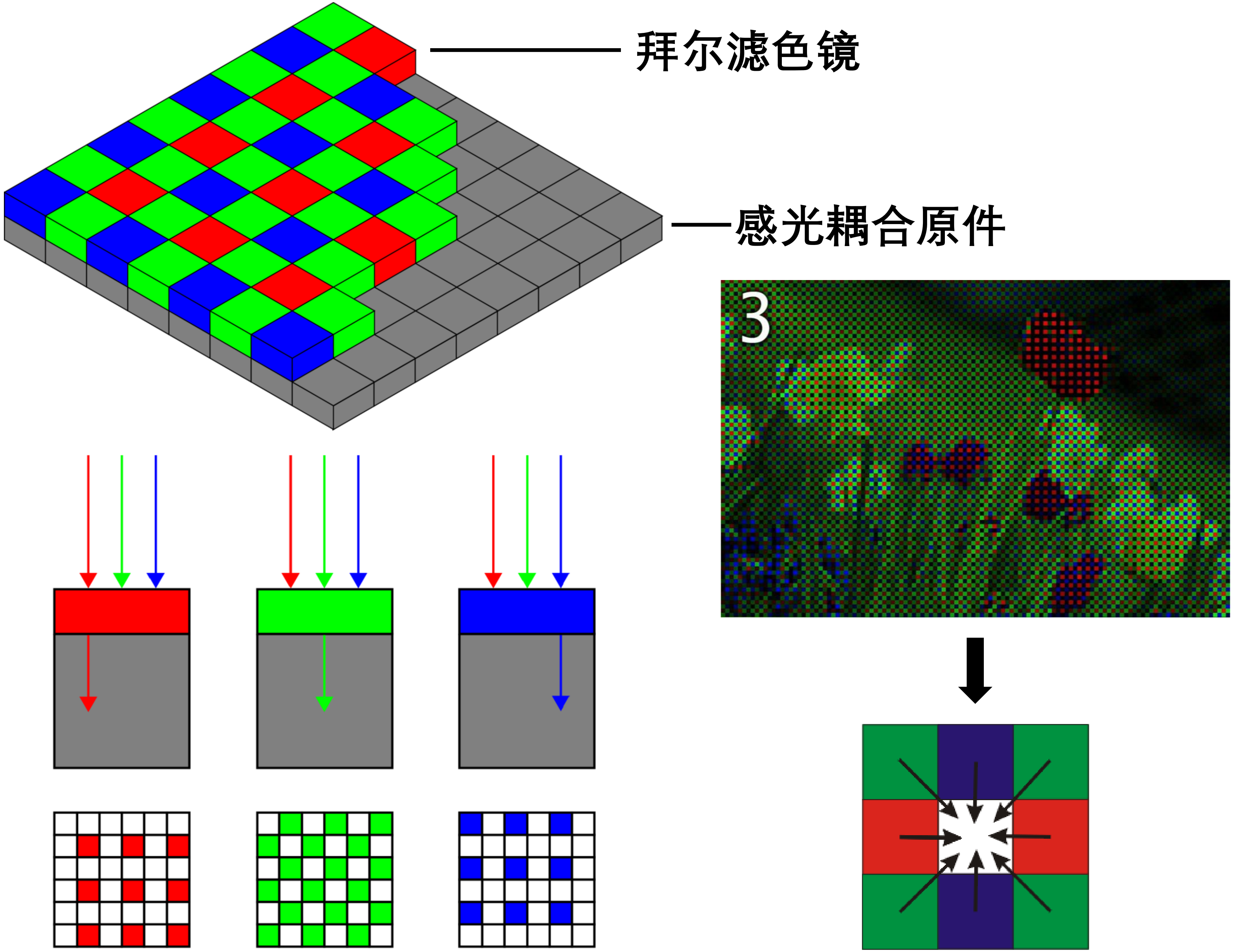

拜尔滤色镜
感光耦合原件
入射光线
1

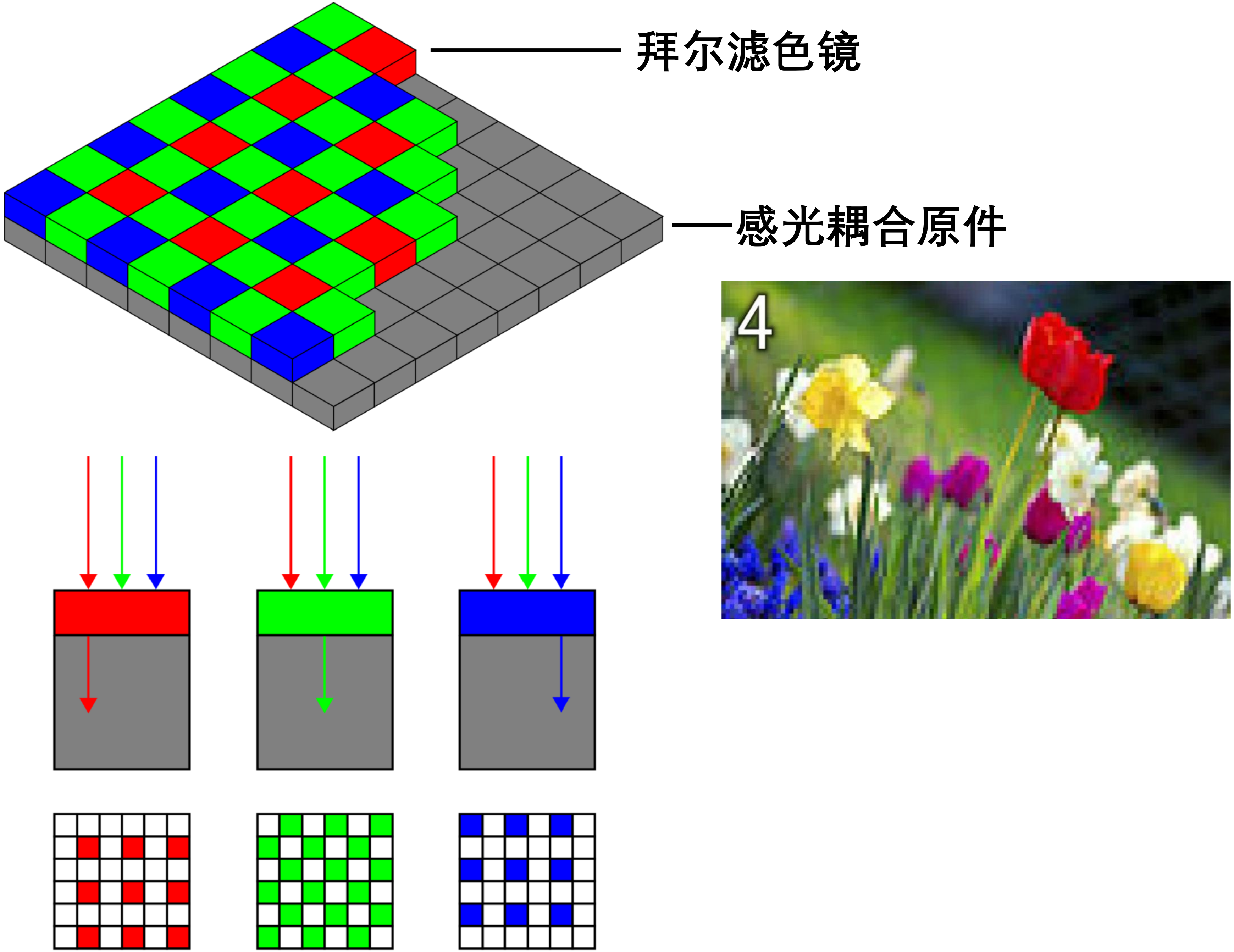

拜尔滤色镜
感光耦合原件
入射光线
1
滤色镜
传感器阵列


拜尔滤色镜
感光耦合原件
2
滤色镜
传感器阵列


拜尔滤色镜
感光耦合原件
2
色彩模板

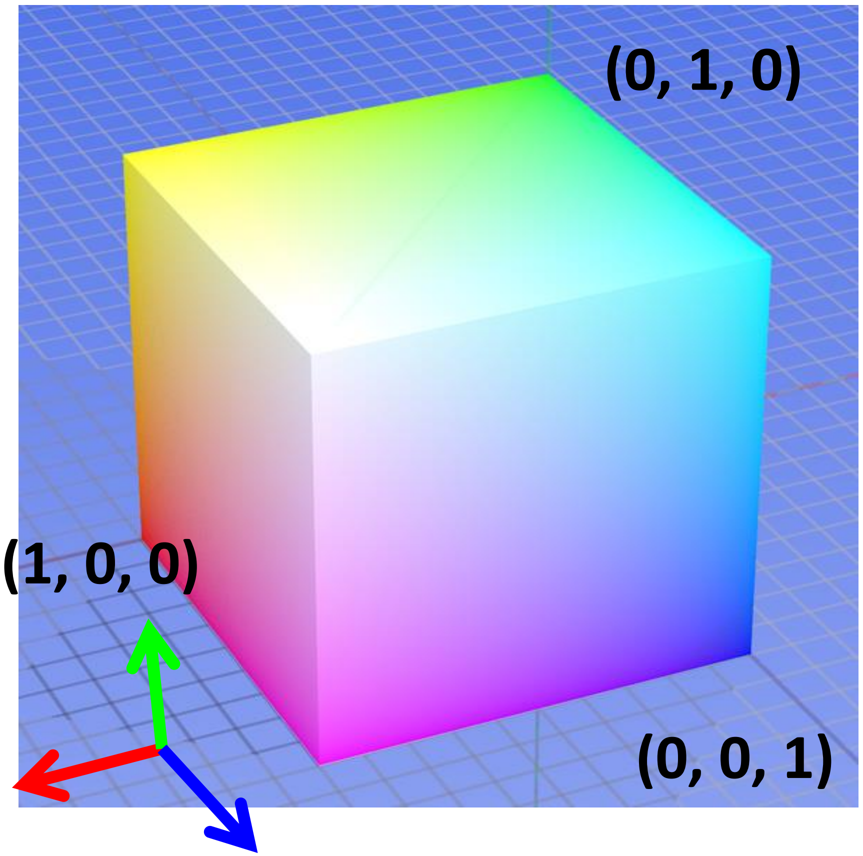

拜尔滤色镜
感光耦合原件
3
色彩模板


拜尔滤色镜
感光耦合原件
3


拜尔滤色镜
感光耦合原件


如何表示颜色？

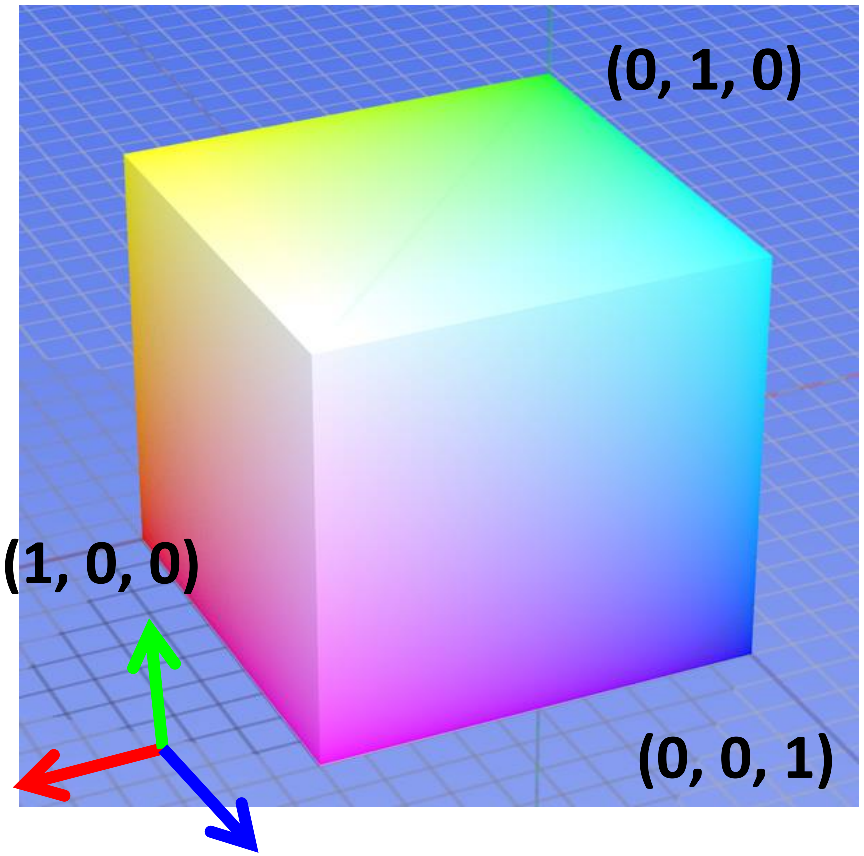
色彩空间
RGB



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

$256 \times 256 \times 256$
≈ 1677 万色

色彩空间
RGB

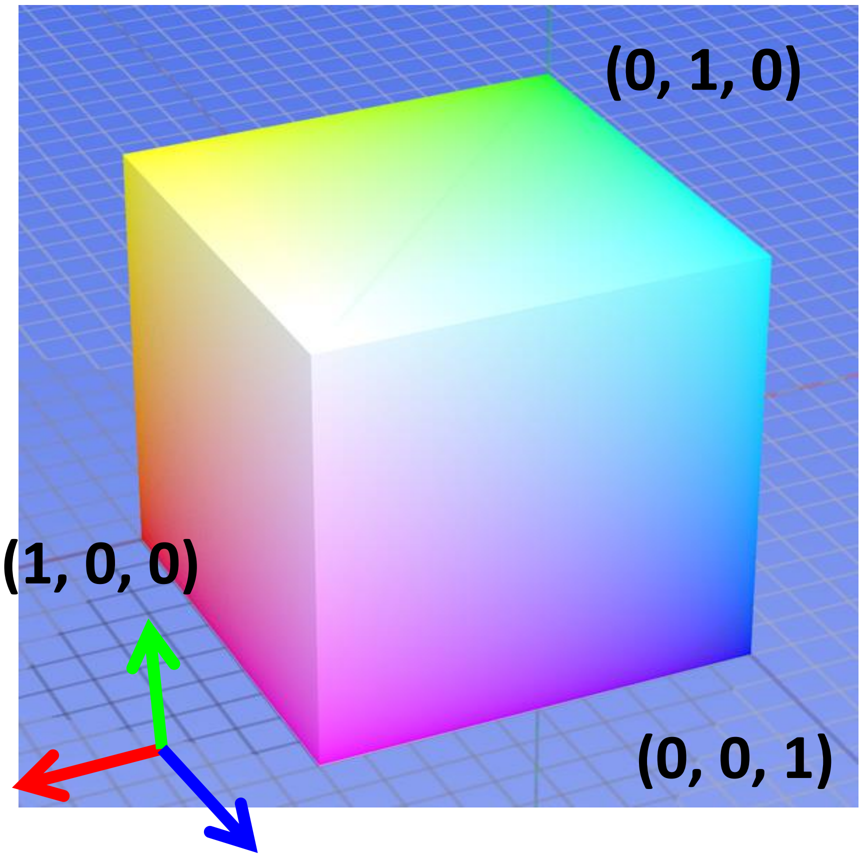


$$8\ \text{bit} \times 3 = 24\ \text{bit}$$

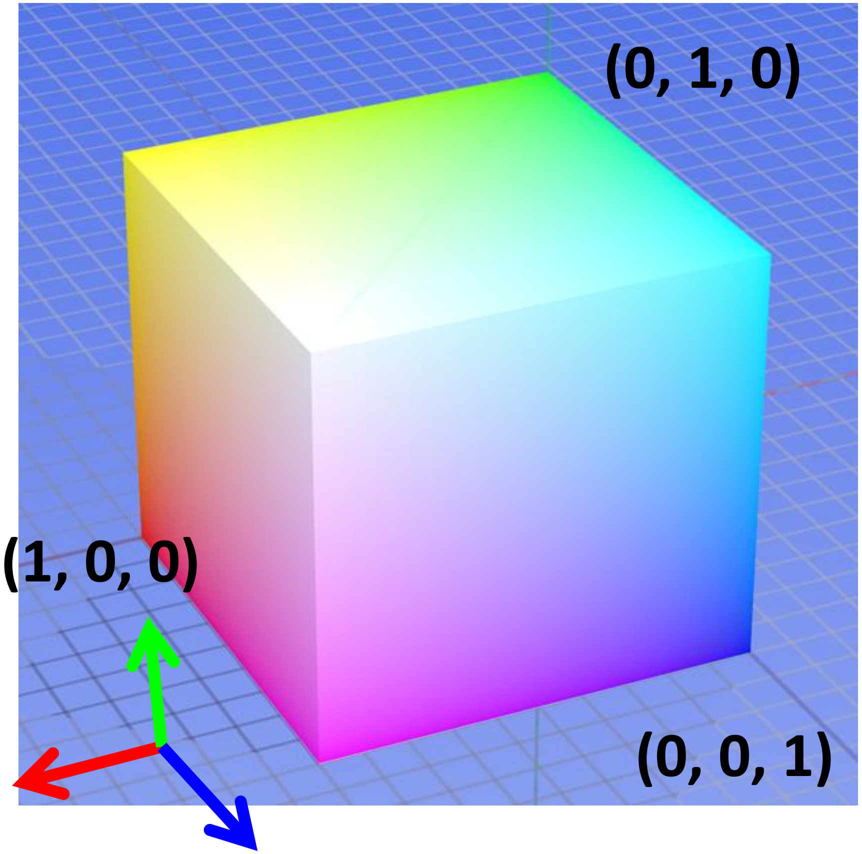
$$256 \times 256 \times 256$$
$$\approx 1677\ \text{万色}$$

24位真彩色

色彩空间
RGB



色彩空间 RGB

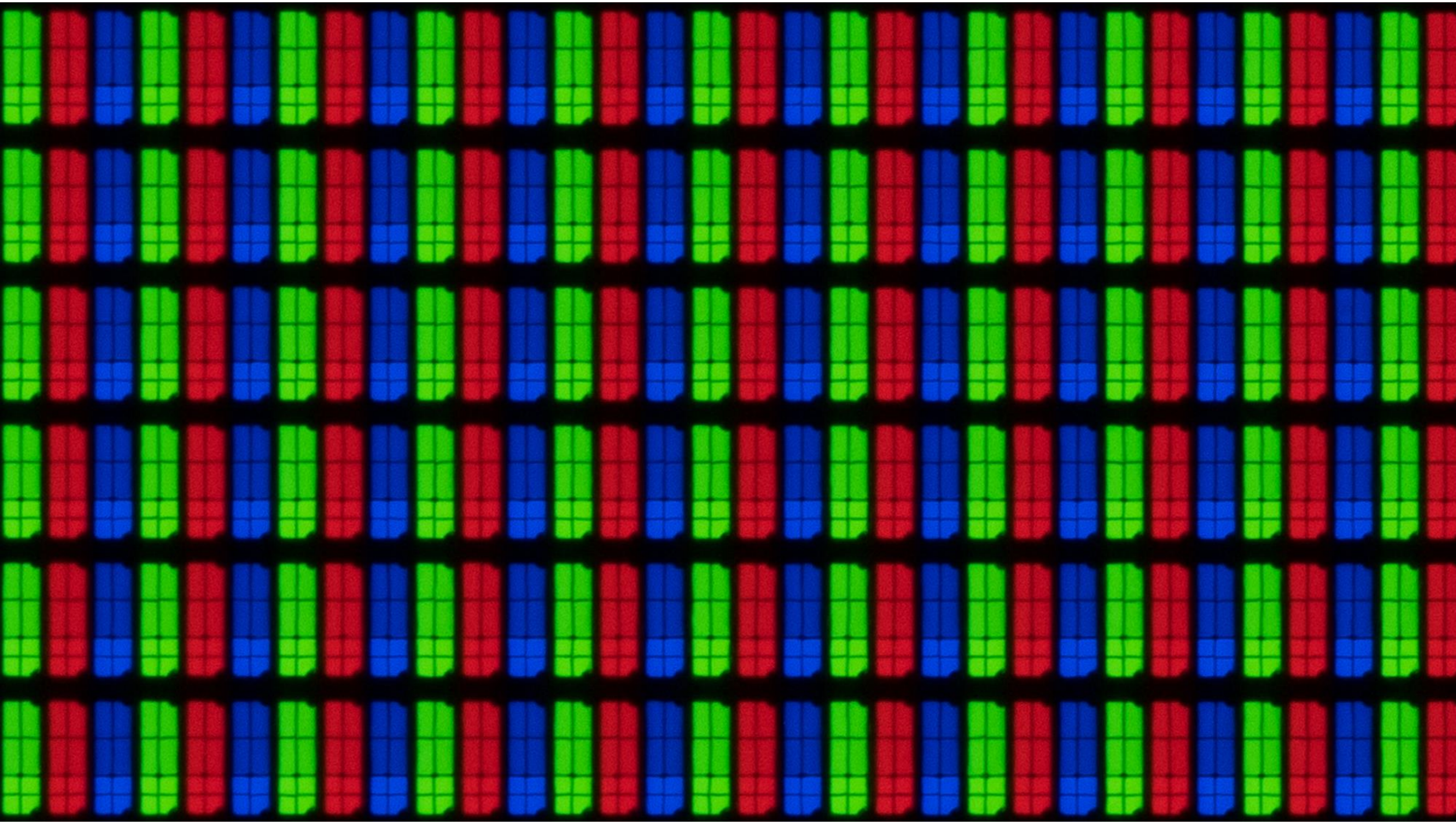


缺点：通道间相关性强
非感知

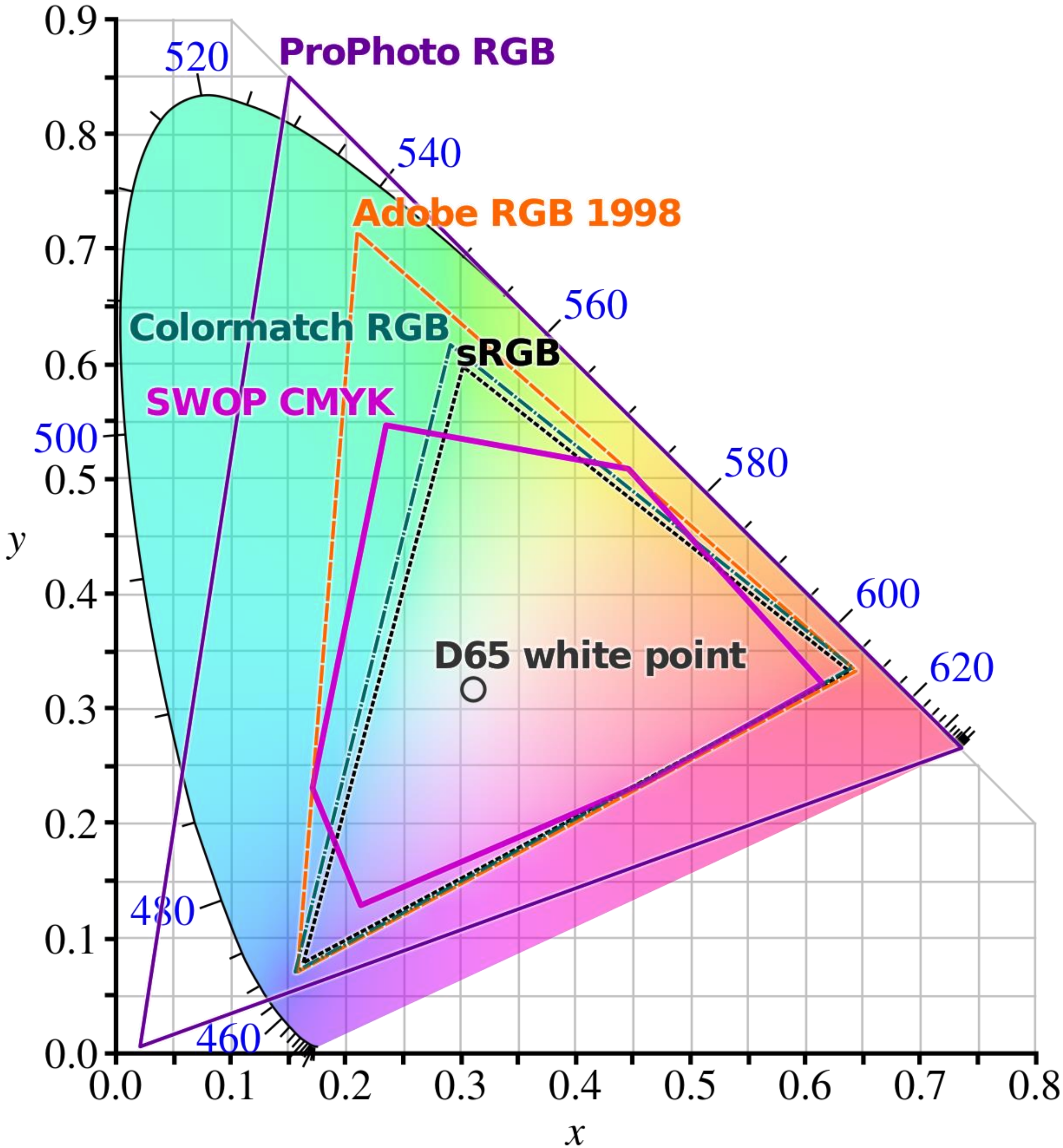
R
(**G** = 0, **B** = 0)

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

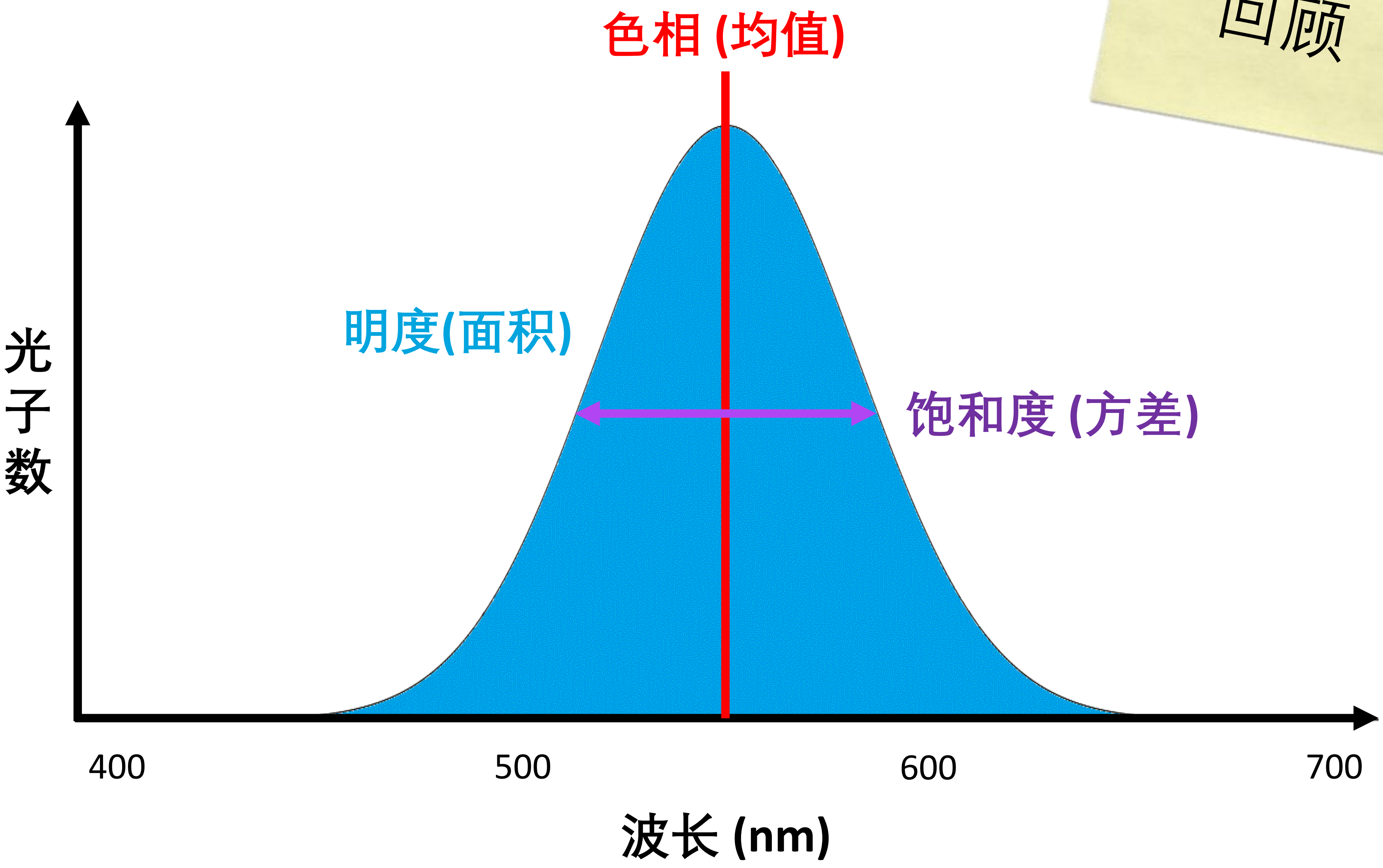
B
(**R** = 0, **G** = 0)



RGB色域

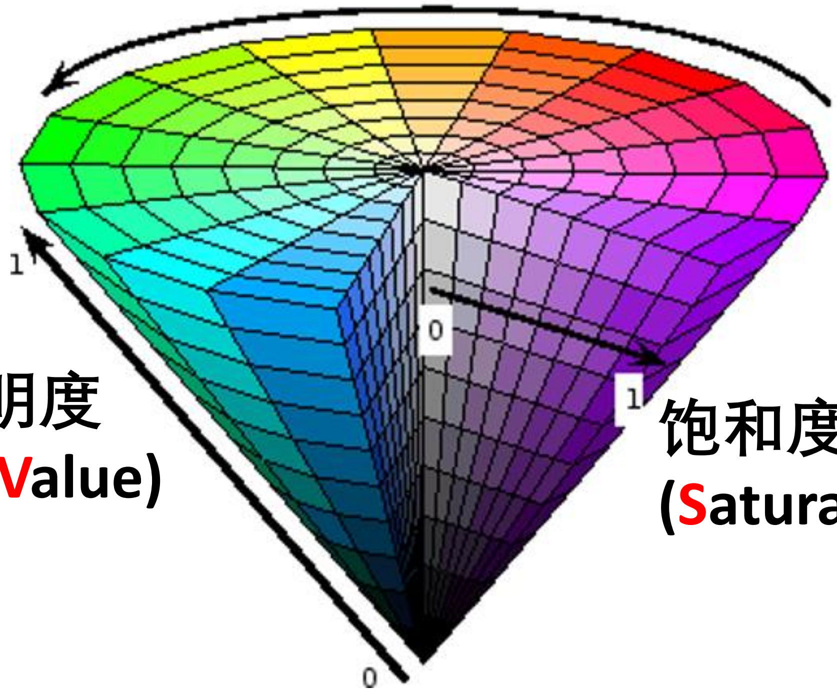

ProPhoto RGB
Adobe RGB 1998
Colormatch RGB
sRGB
SWOP CMYK
D65 white point
520
540
560
580
600
620
500
480
460
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
x
y

回顾



色彩空间 HSV

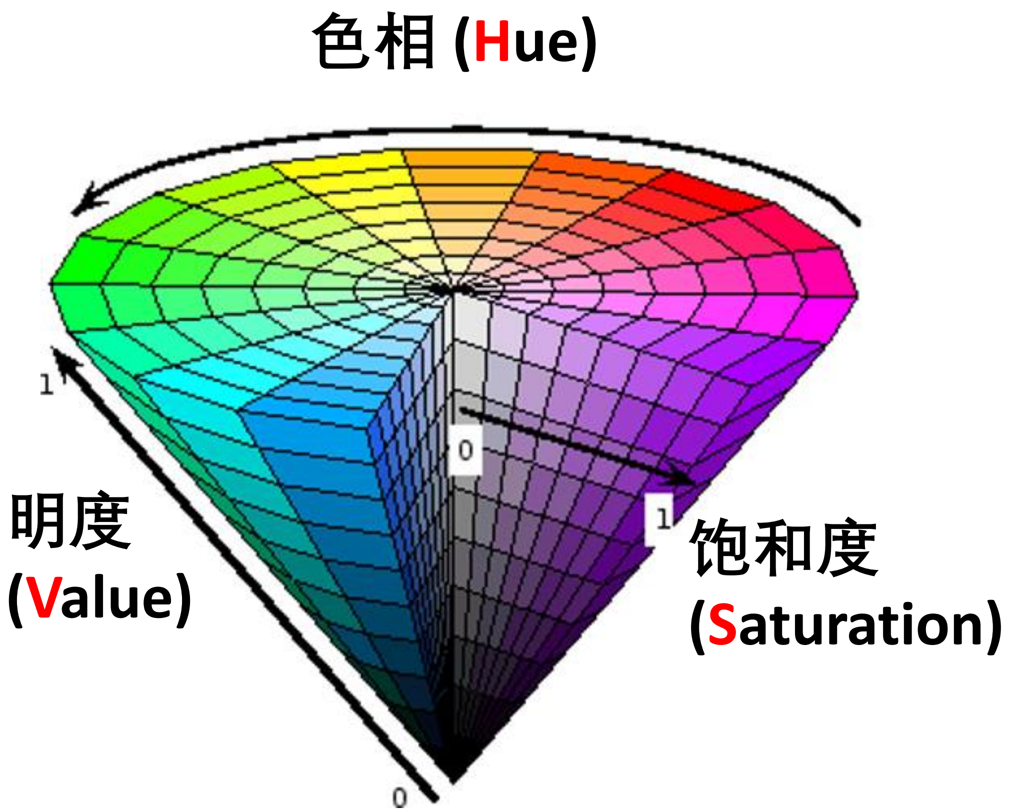
色相 (Hue)



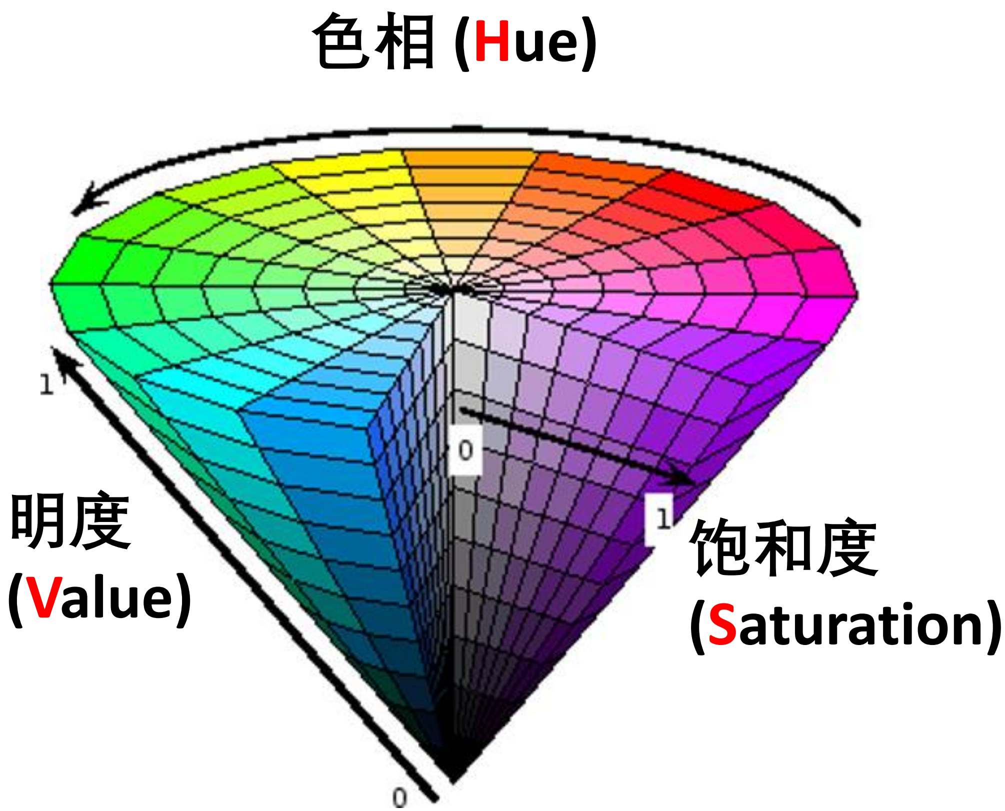
明度 (Value)

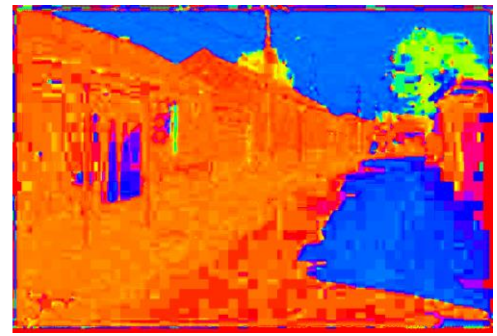
饱和度 (Saturation)

色彩空间 HSV



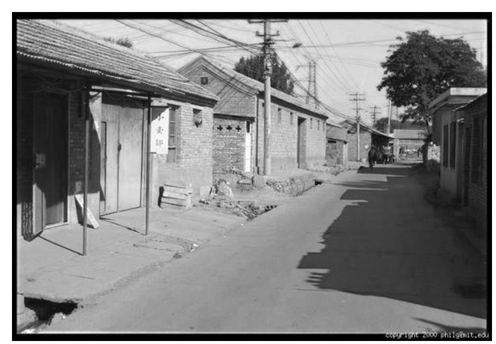
色彩空间 HSV





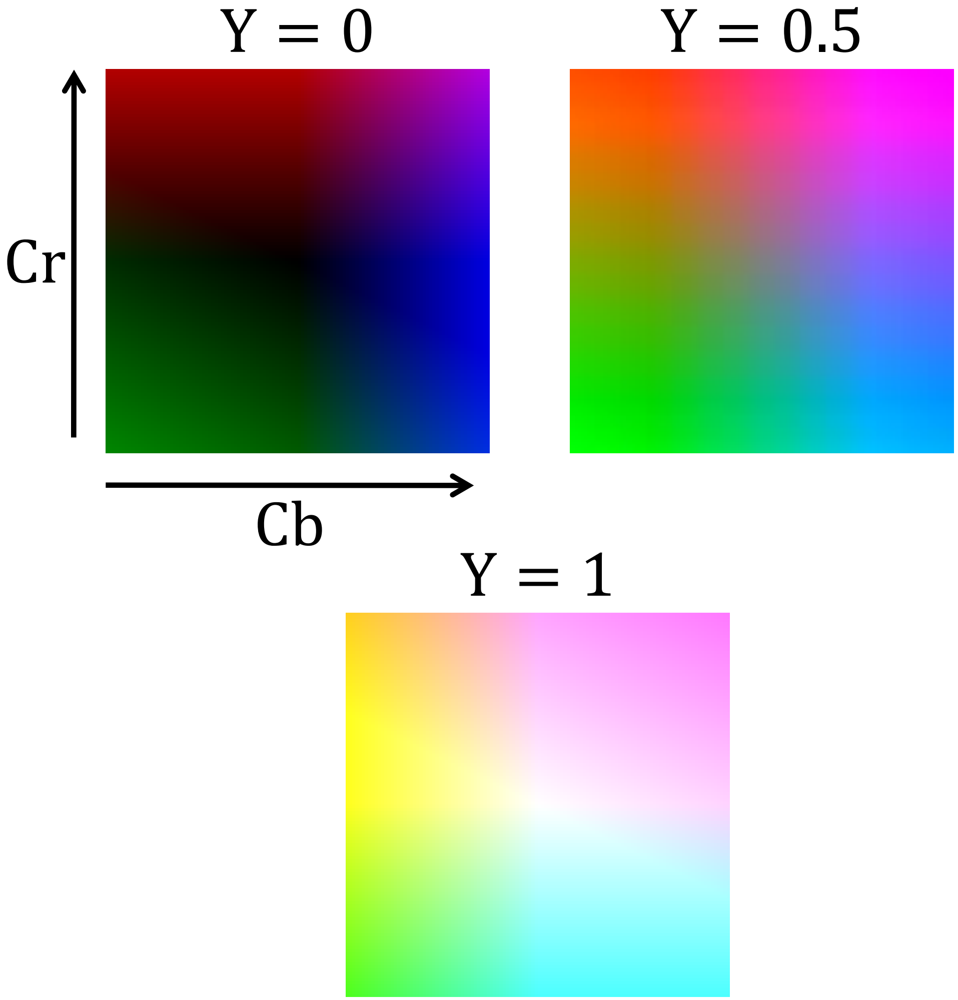
H
$(\mathbf{S} = 1, \mathbf{V} = 1)$

S
$(\mathbf{H} = 1, \mathbf{V} = 1)$



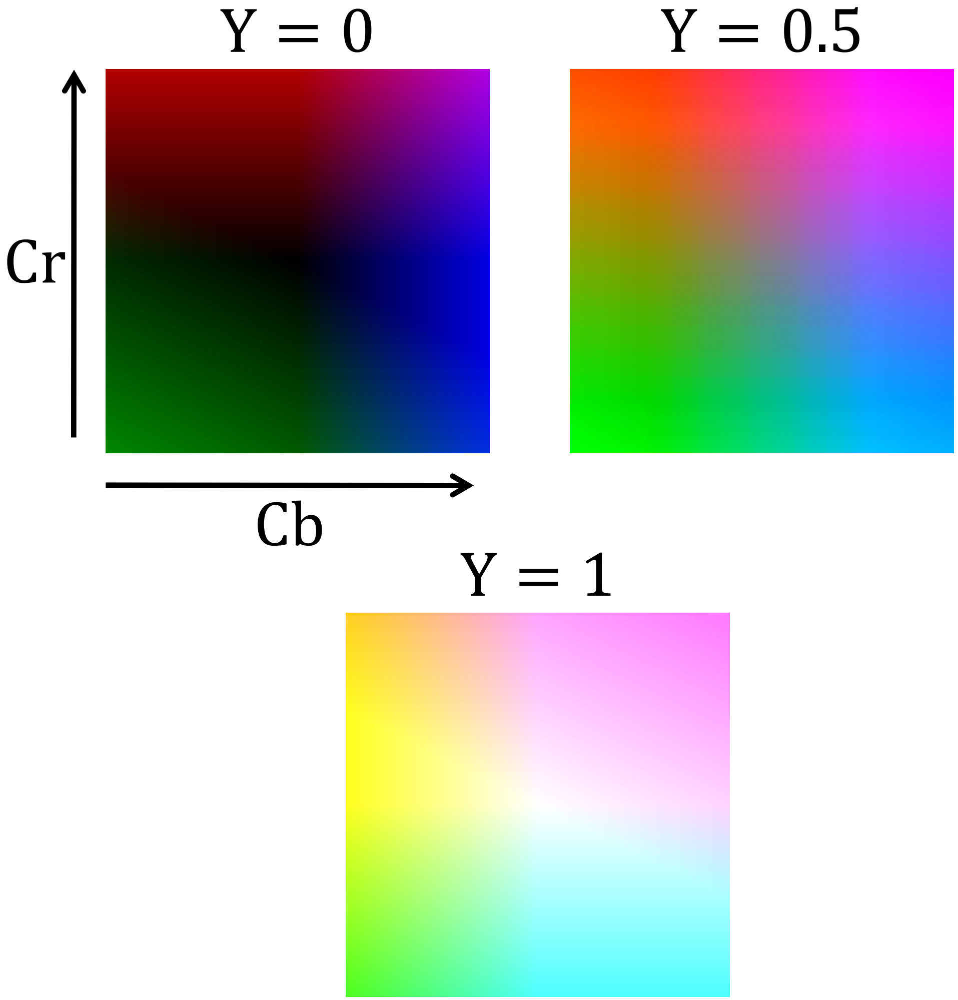
V
$(\mathbf{H} = 1, \mathbf{S} = 0)$

色彩空间 YCrCb



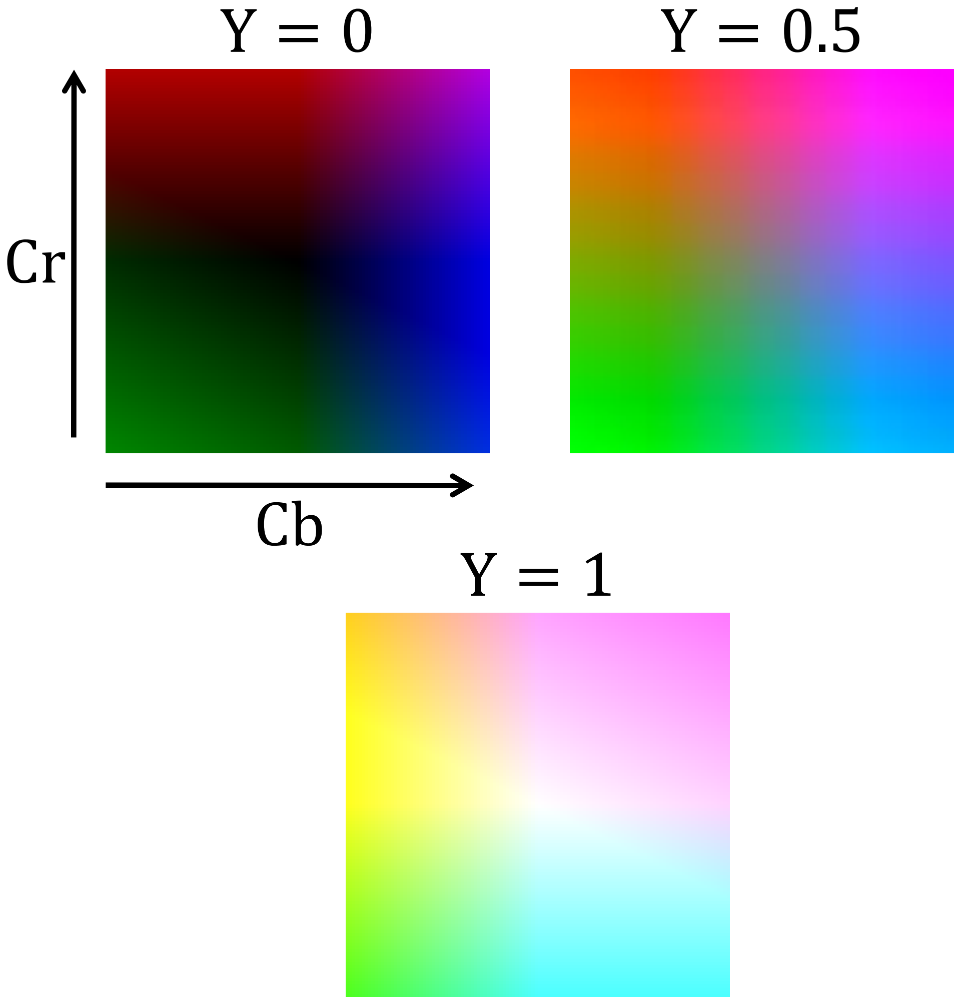
Y： 明流，表示光的浓度且非线性
Cr： 红色浓度偏移成分
Cb： 蓝色浓度偏移成分

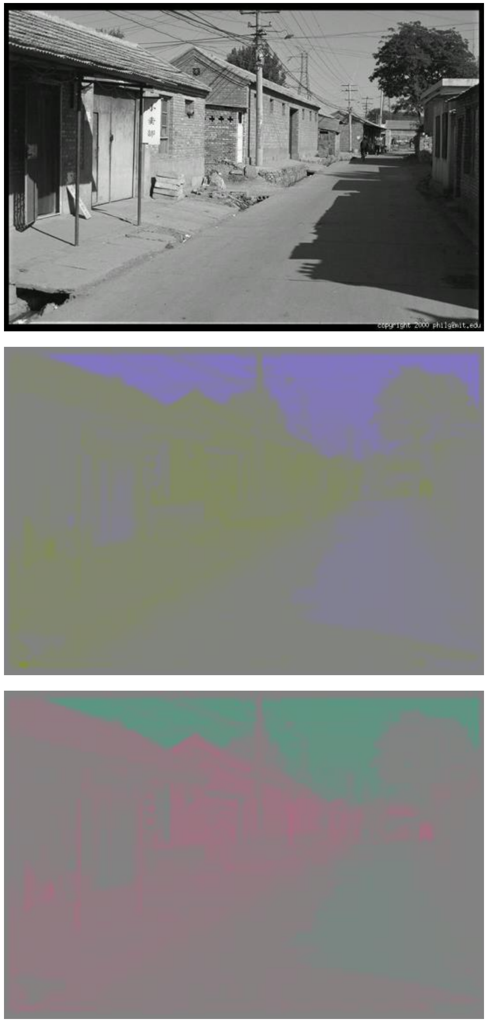
色彩空间 YCrCb



Y： 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





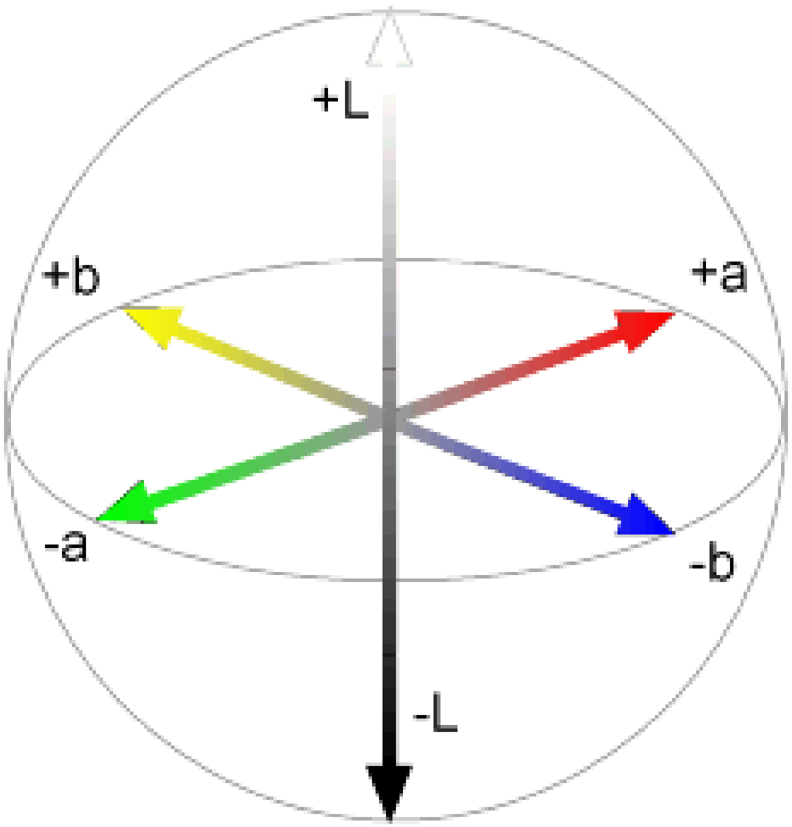
Y： 明流，表示光的浓度且非线性

Cr： 红色浓度偏移成分

Cb： 蓝色浓度偏移成分

色彩空间 CIELAB

“感知均匀”

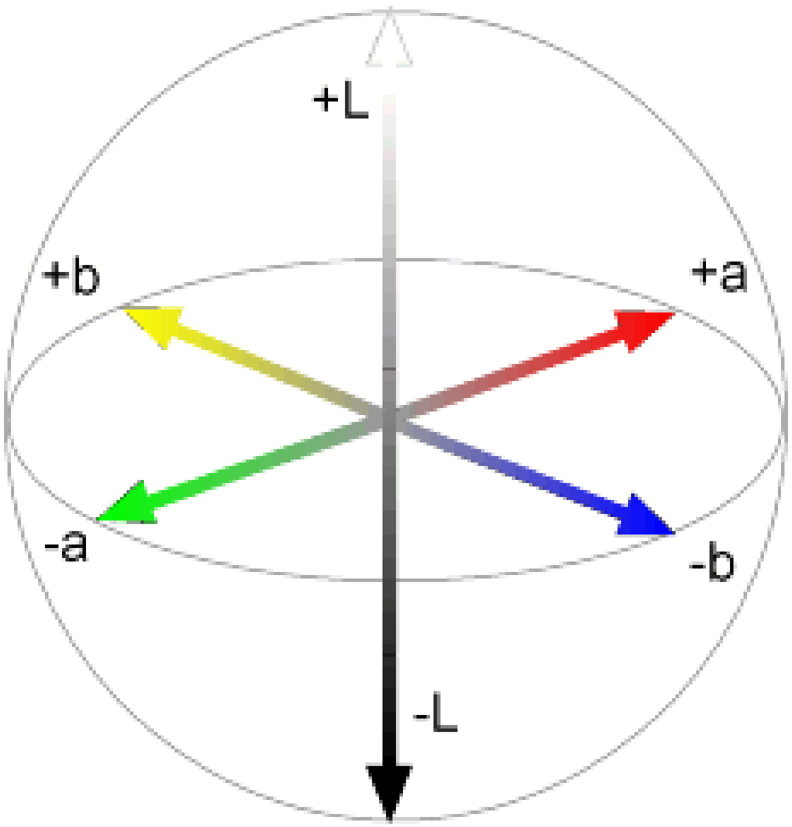


$$16 \text{ bit} \times 3 = 48 \text{ bit}$$

$$65536 \times 65536 \times 65536$$
$\approx$ 281 千亿色

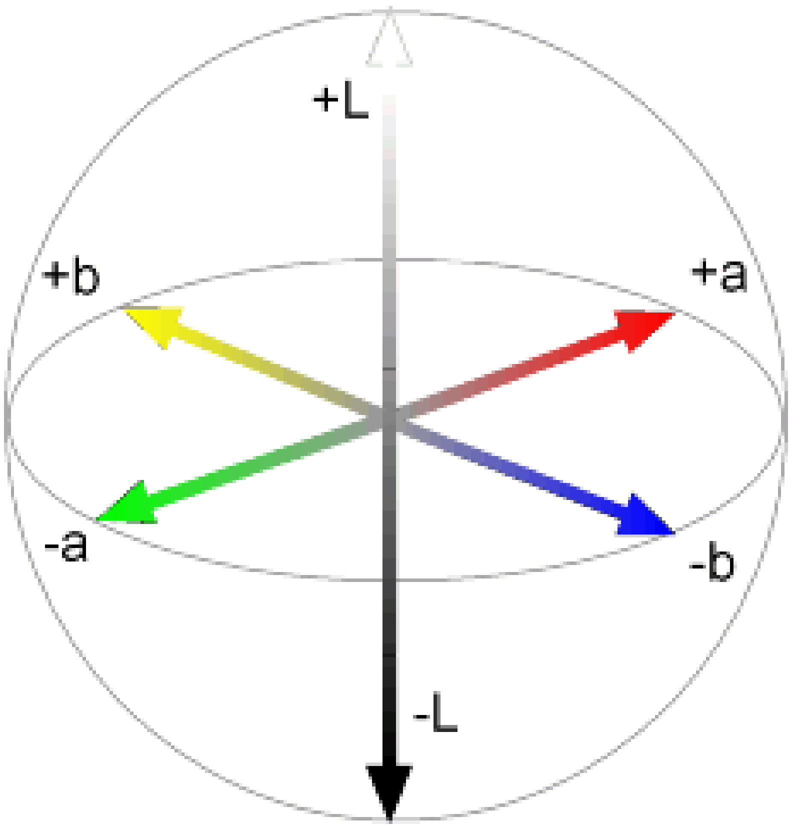
色彩空间 CIELAB

“感知均匀”



色彩空间 CIELAB

“感知均匀”



L

$(\mathbf{a} = 0, \mathbf{b} = 0)$

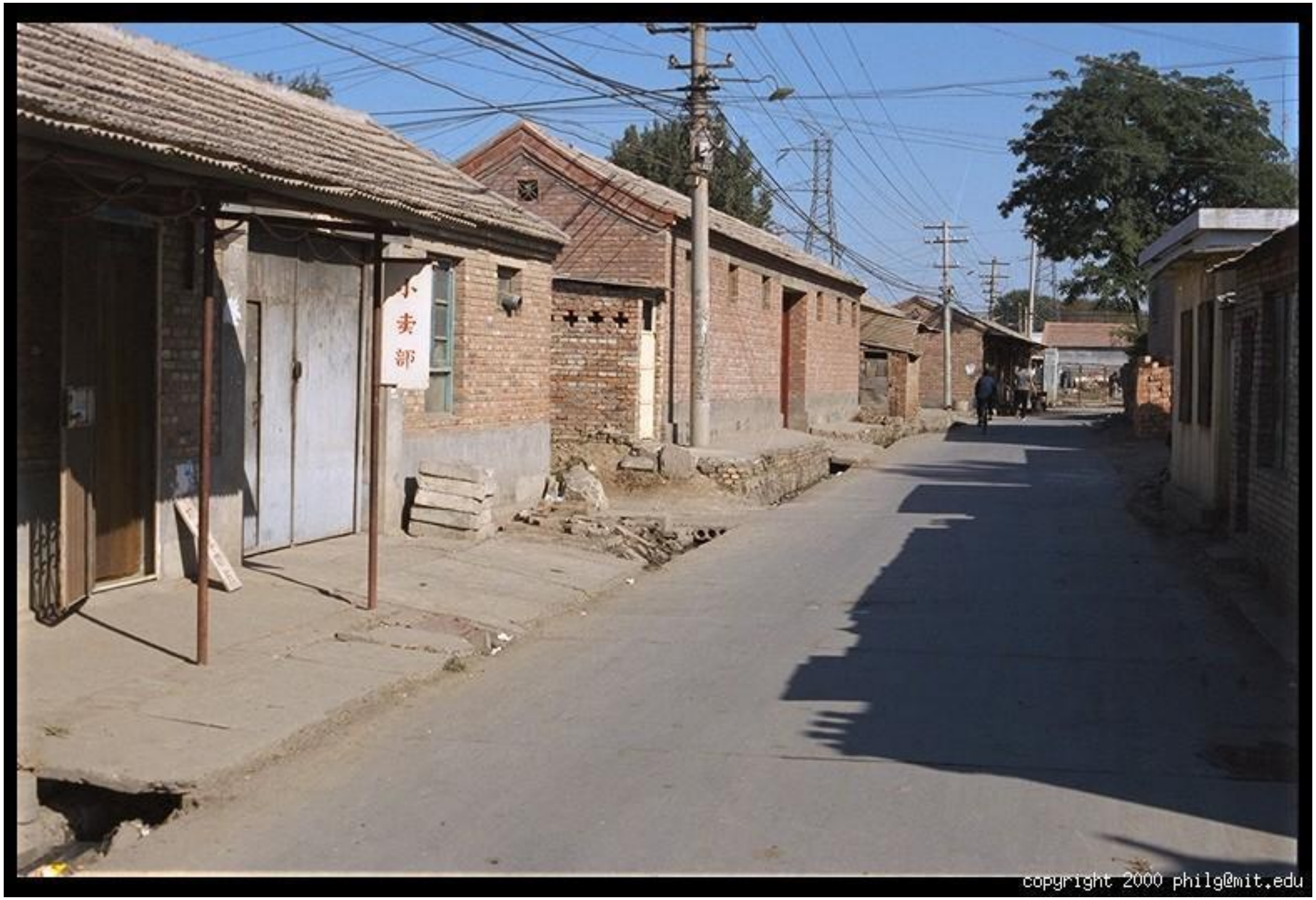
a

$(\mathbf{L} = 65, \mathbf{b} = 0)$

b

$(\mathbf{L} = 65, \mathbf{a} = 0)$

如果你不得不选择，你宁愿放弃**亮度**还是**色度**呢?



原图

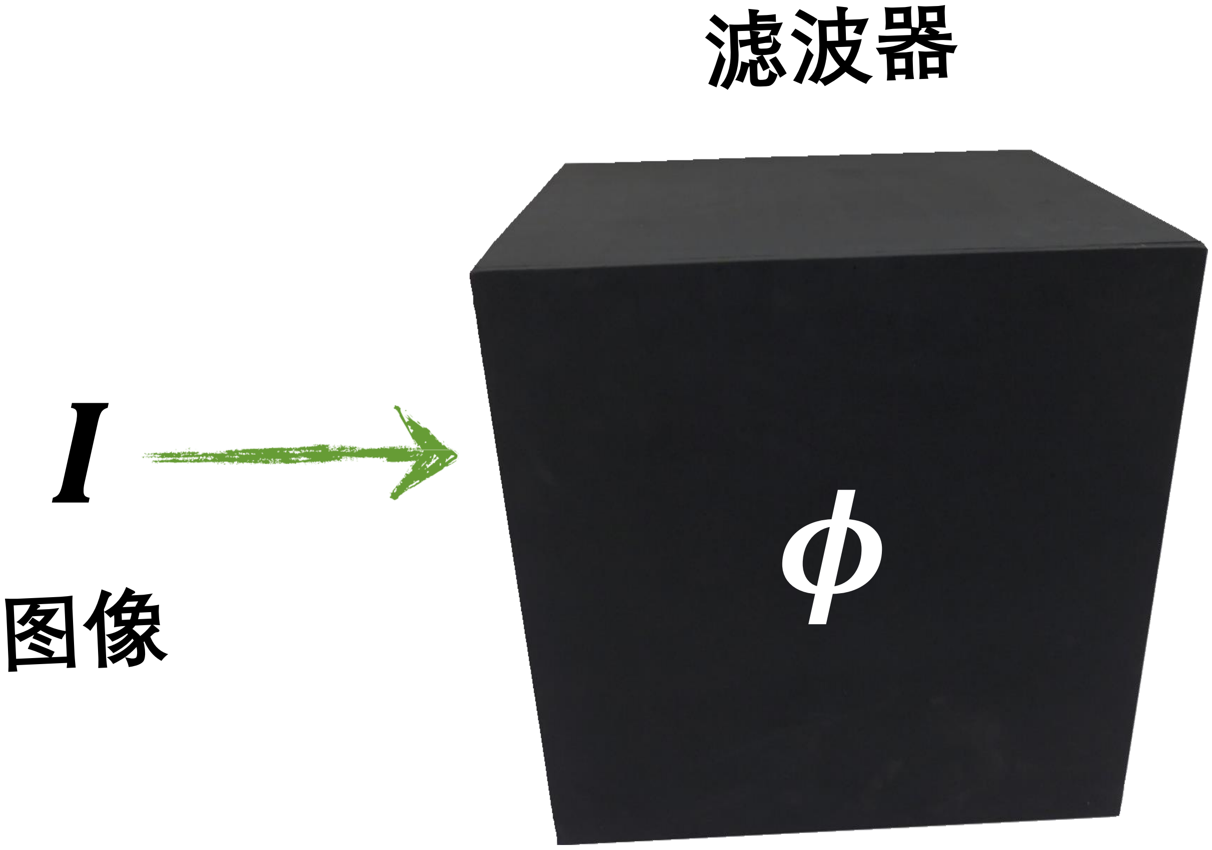
仅显示颜色——恒定强度

仅显示强度——恒定色彩

仅显示强度——恒定色彩

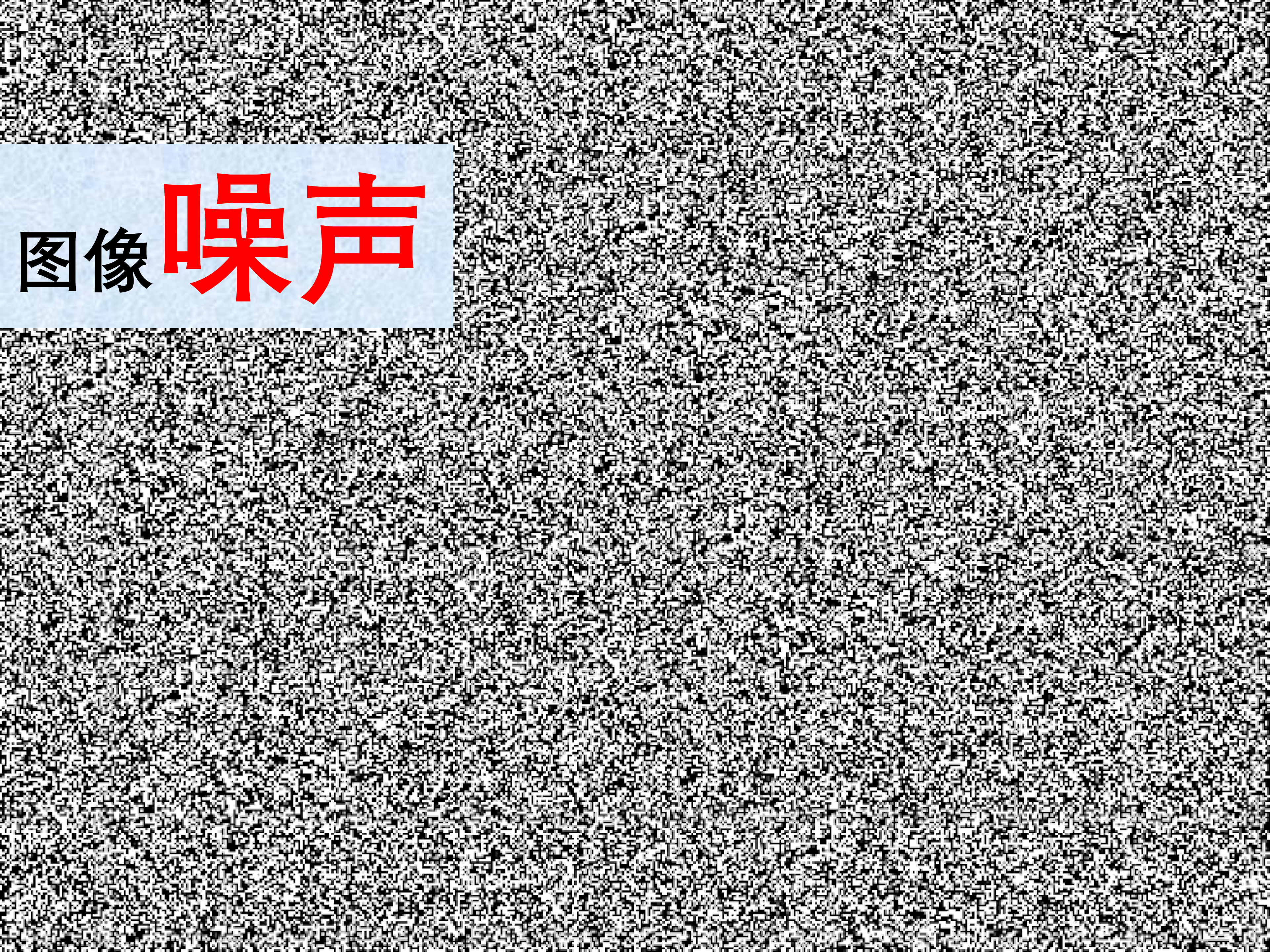
图像滤波

滤波器


滤波器
I
图像
ϕ


滤波器
I
图像
ϕ
$\phi(I)$
图像

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

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

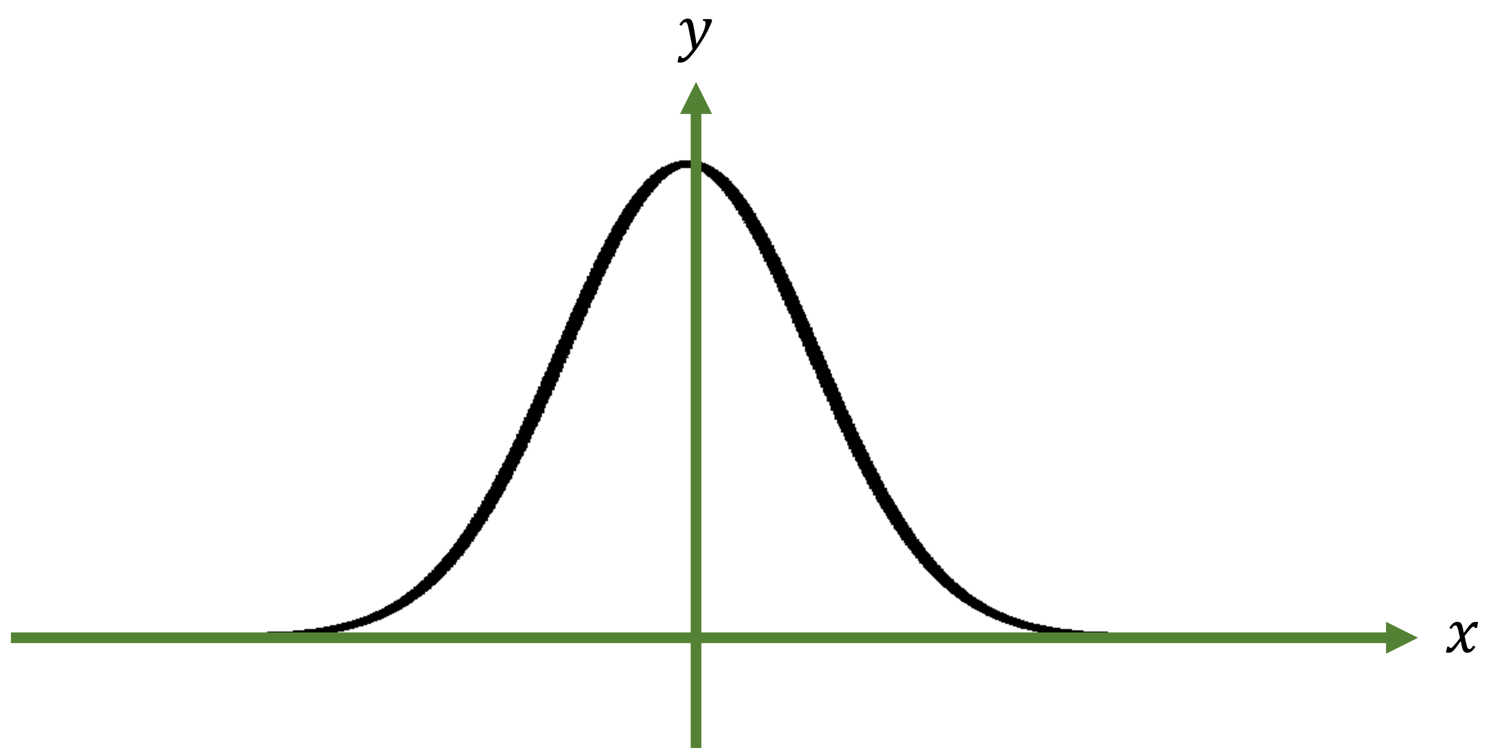
假设噪声独立同分布

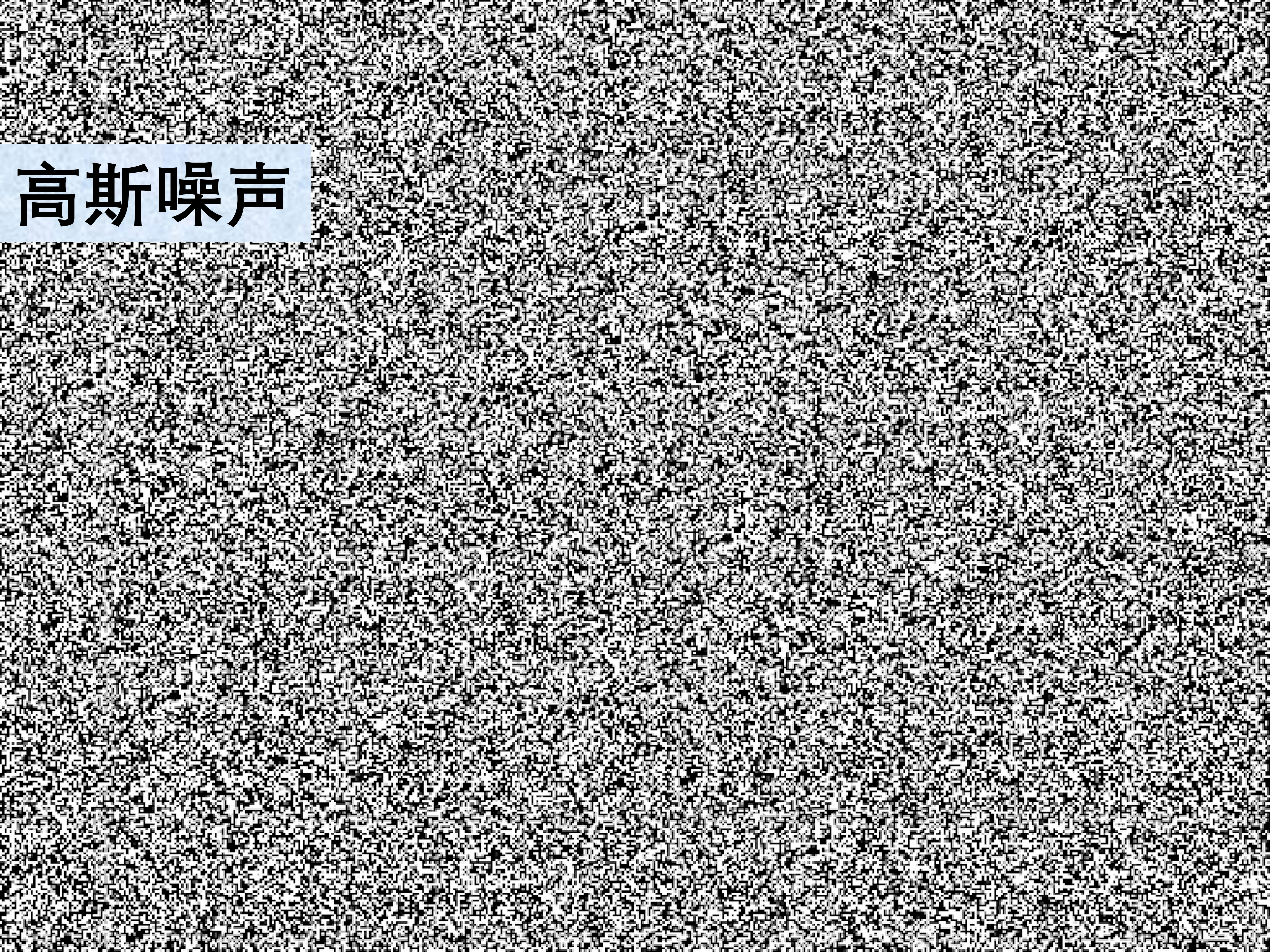
I.I.D

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

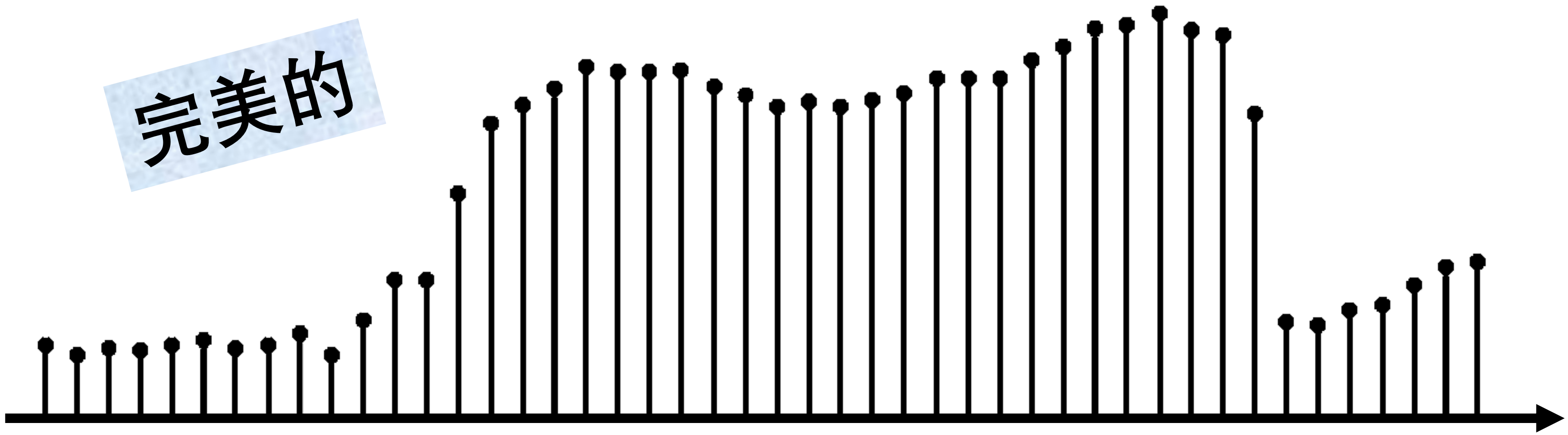
其中

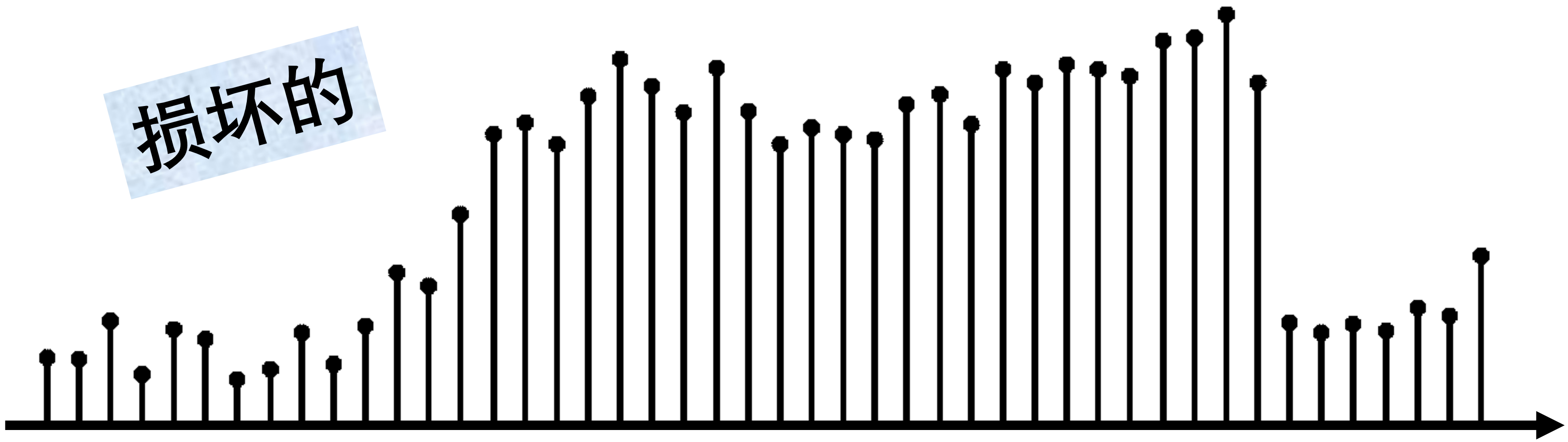
$$\eta(x,y) \sim \mathcal{N}(\mu = 0, \sigma)$$

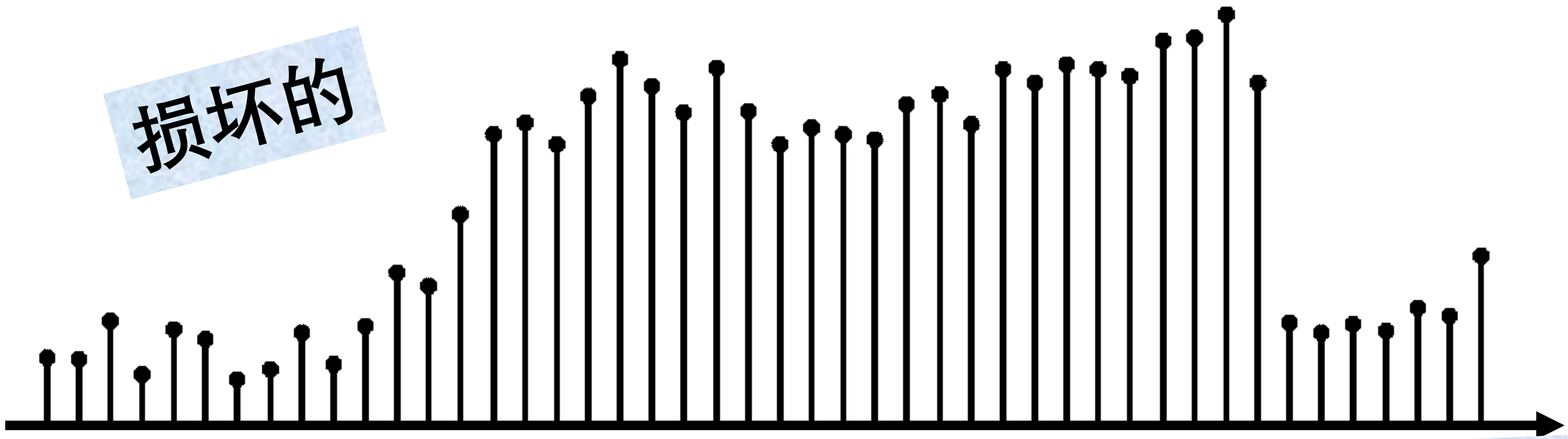




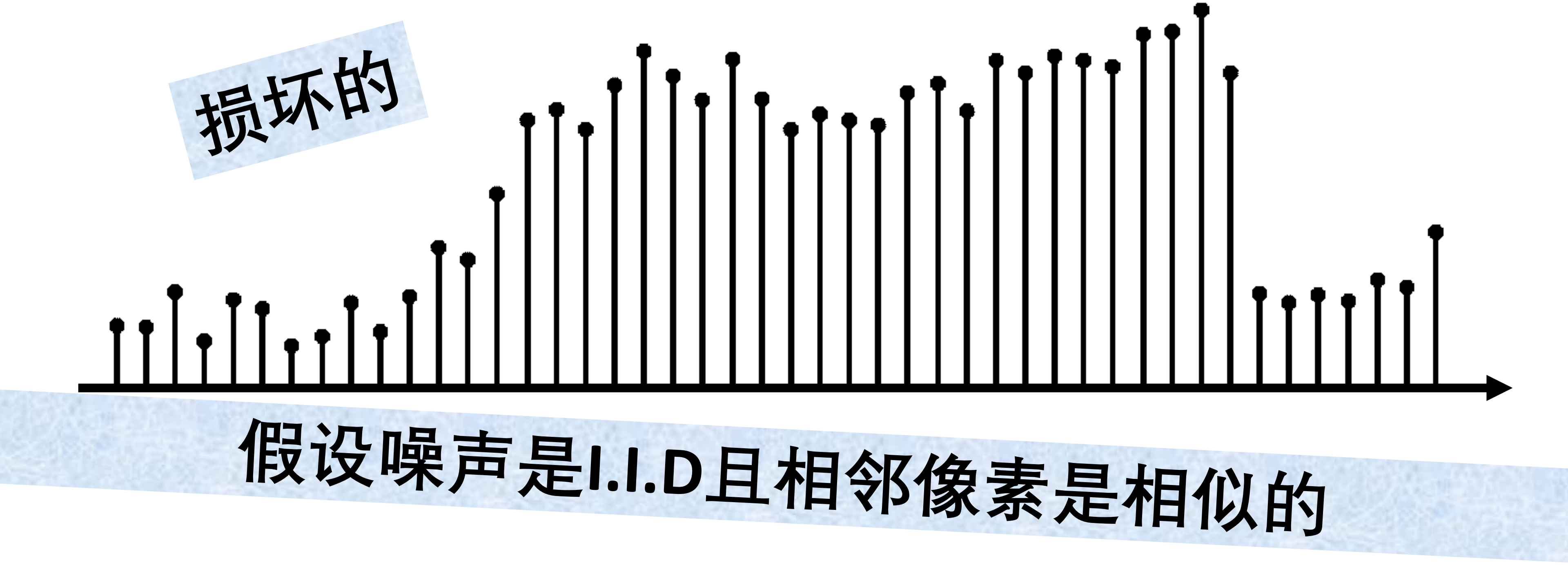
高斯噪声

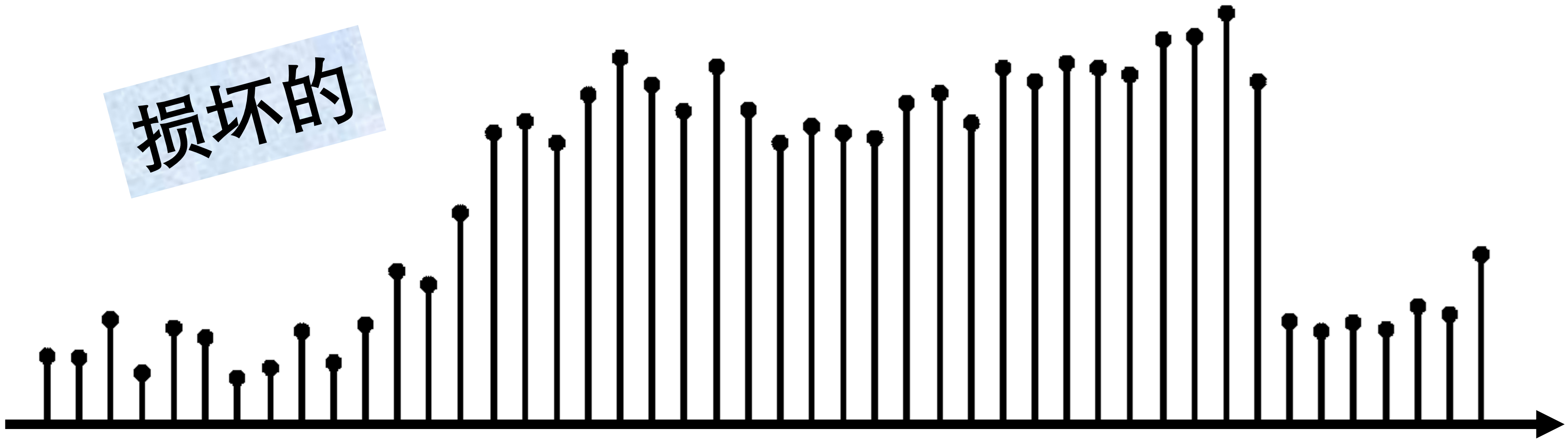

完美的


损坏的



怎样才能消除噪声？

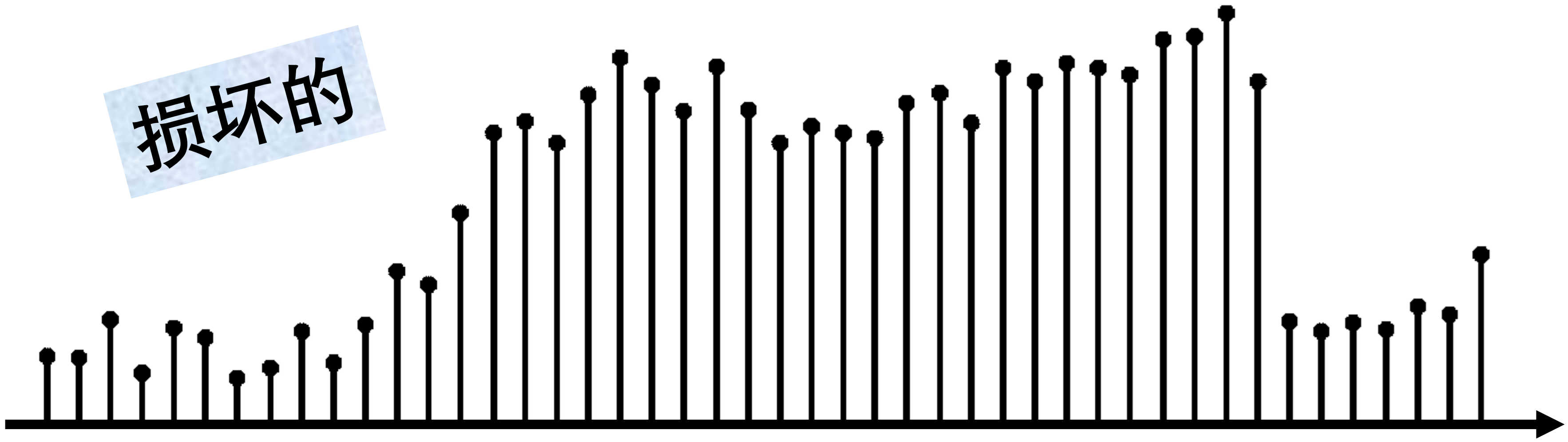

损坏的
假设噪声是I.I.D且相邻像素是相似的



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

其中

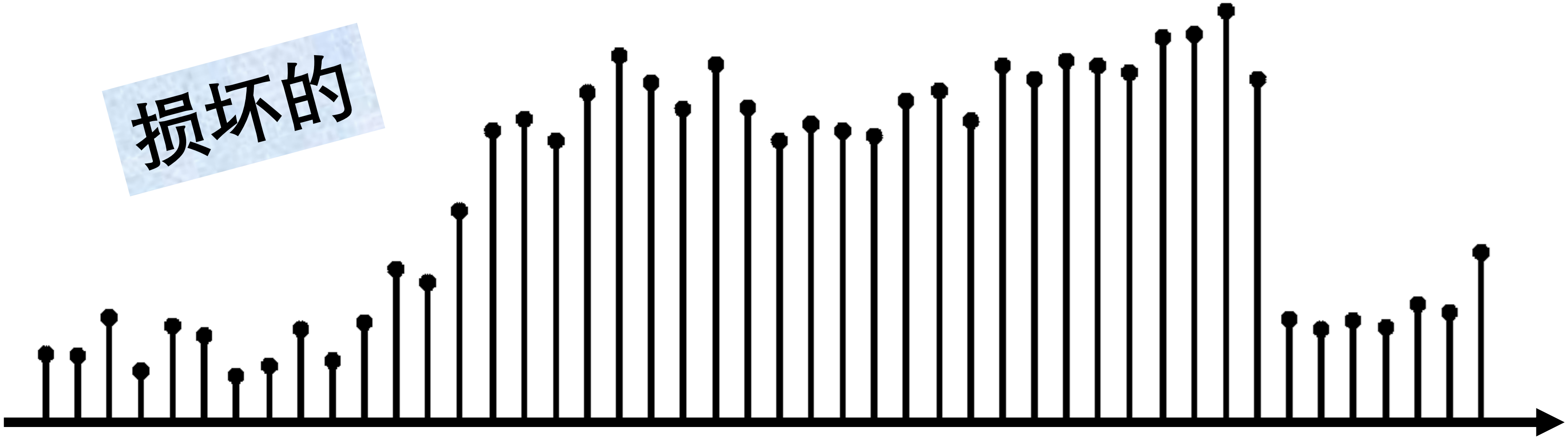
$$\eta(x,y) \sim \mathcal{N}(\mu = 0, \sigma)$$



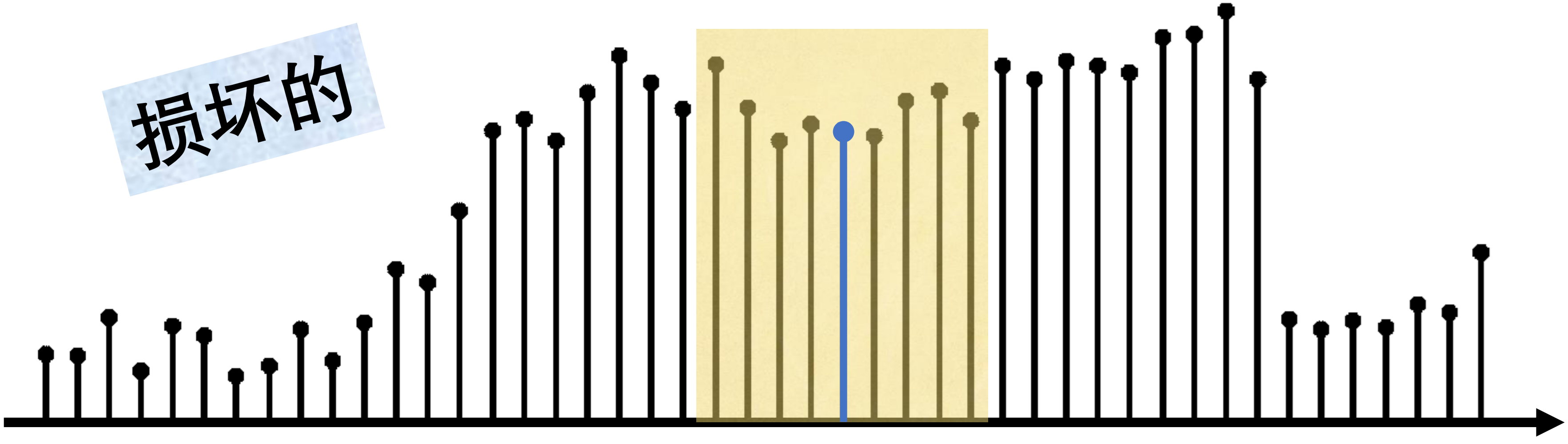
$$I(x,y) = I_{\text{ideal}}(x,y) + \eta(x,y)$$

其中

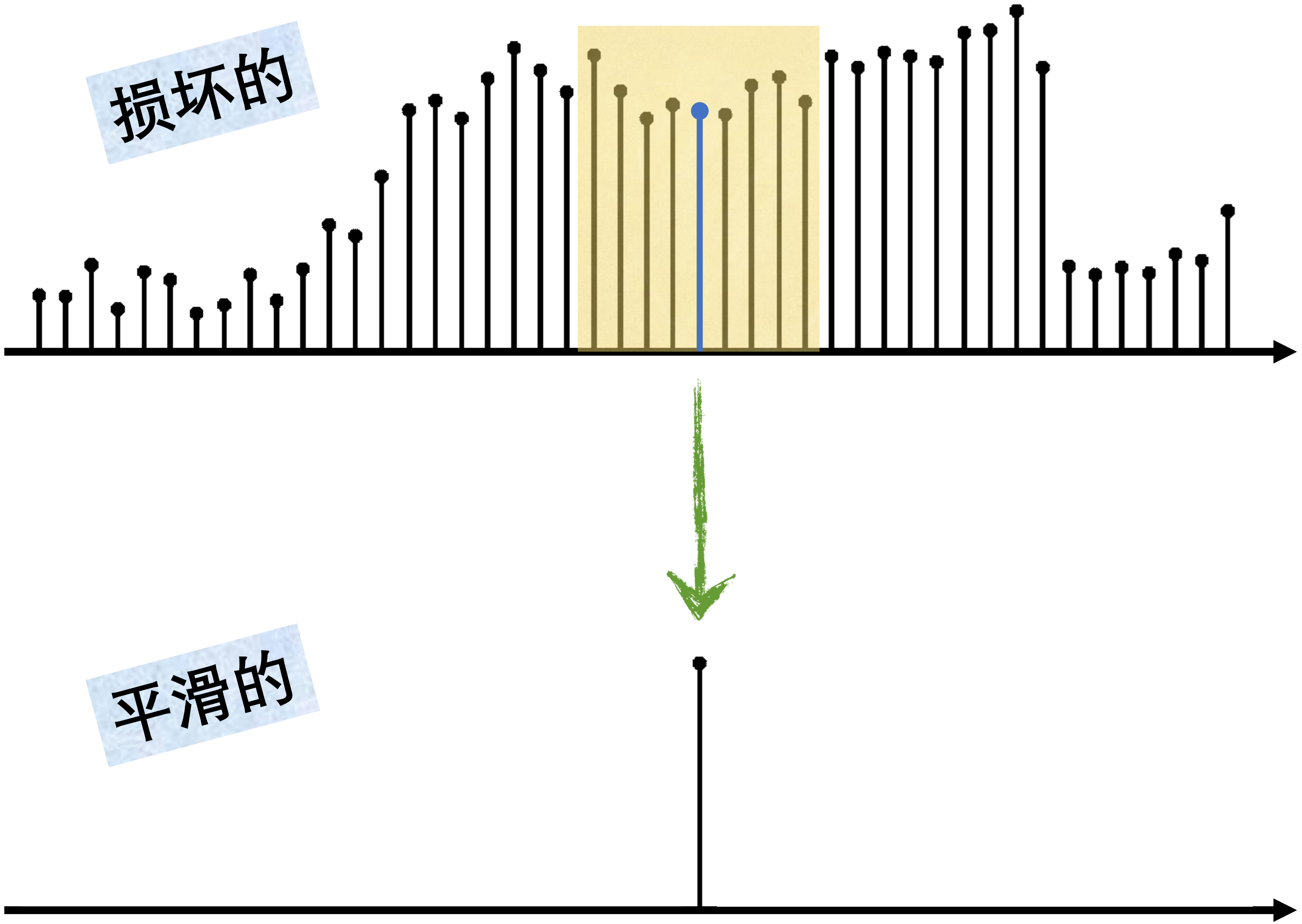
$$\eta(x,y) \sim \mathcal{N}(\mu = 0, \sigma)$$

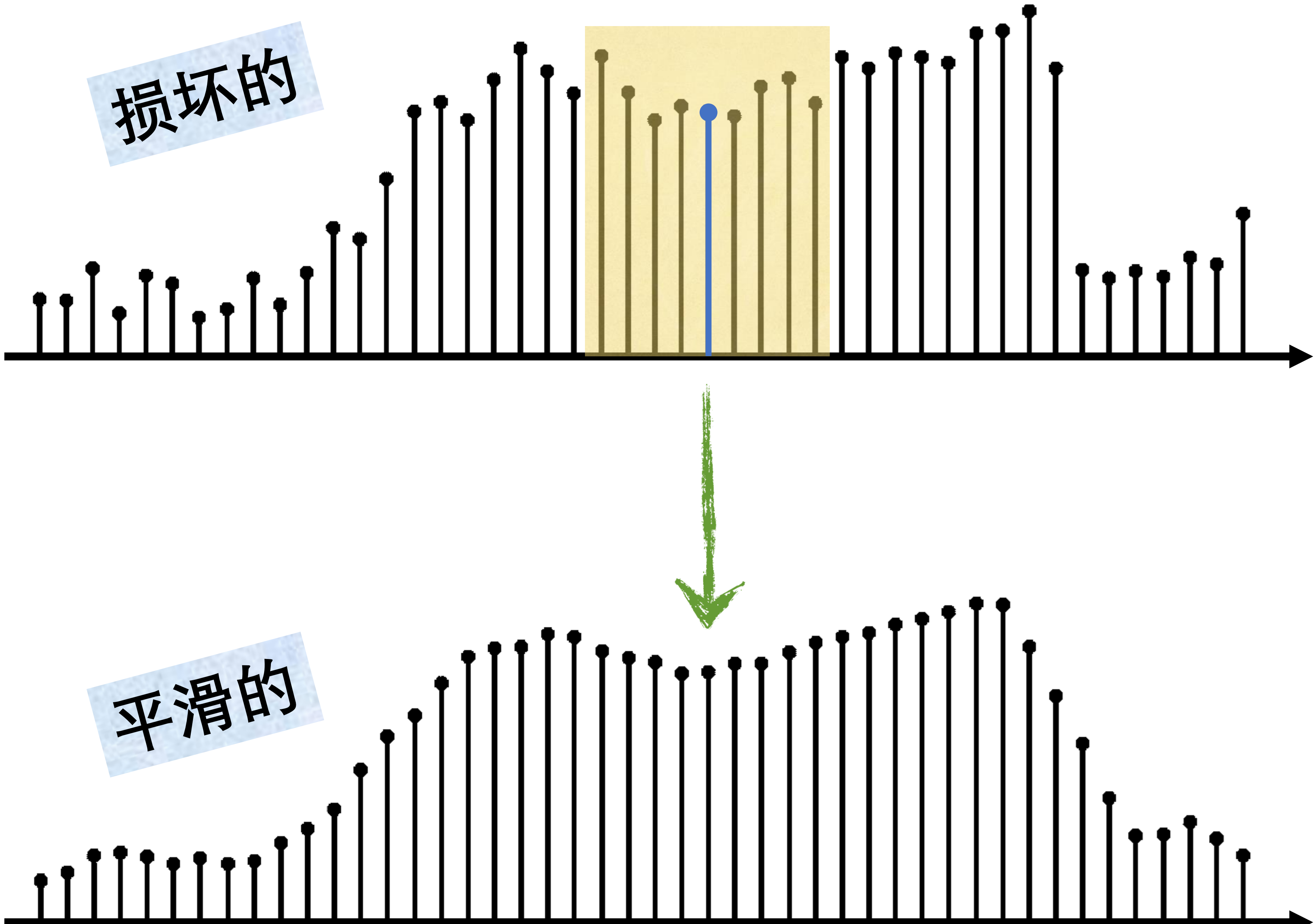


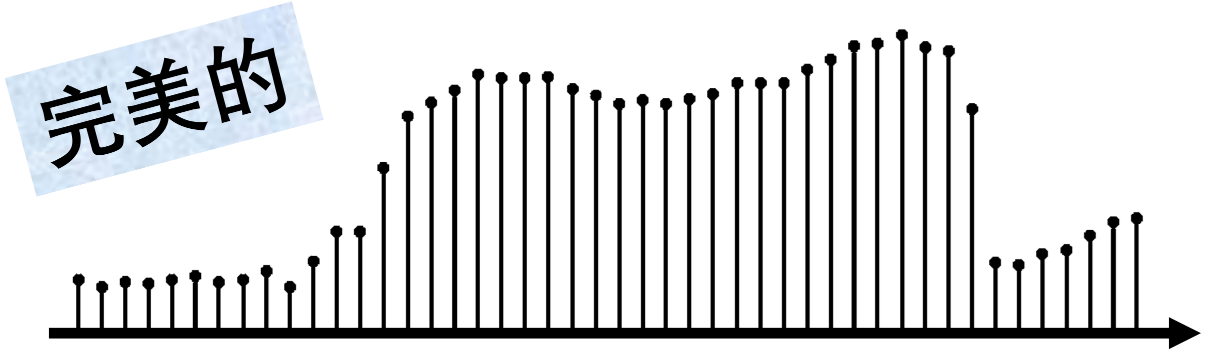
让我们将每个像素替换为其相邻像素的平均值

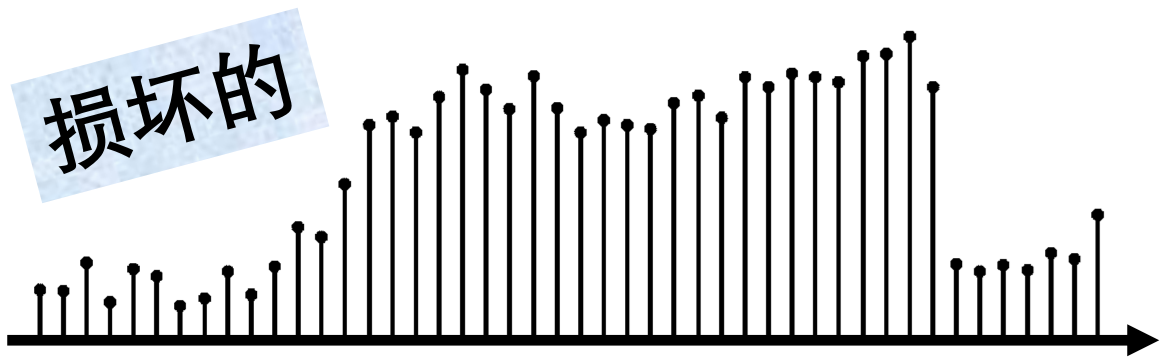


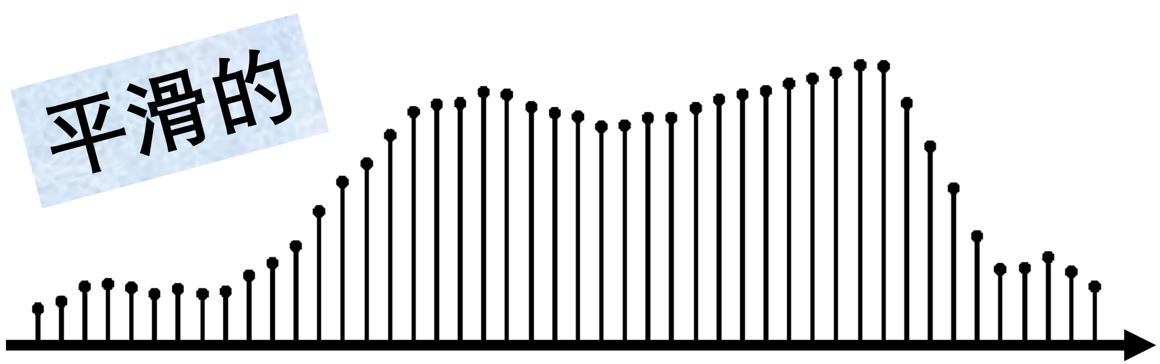
让我们将每个像素替换为其相邻像素的平均值


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

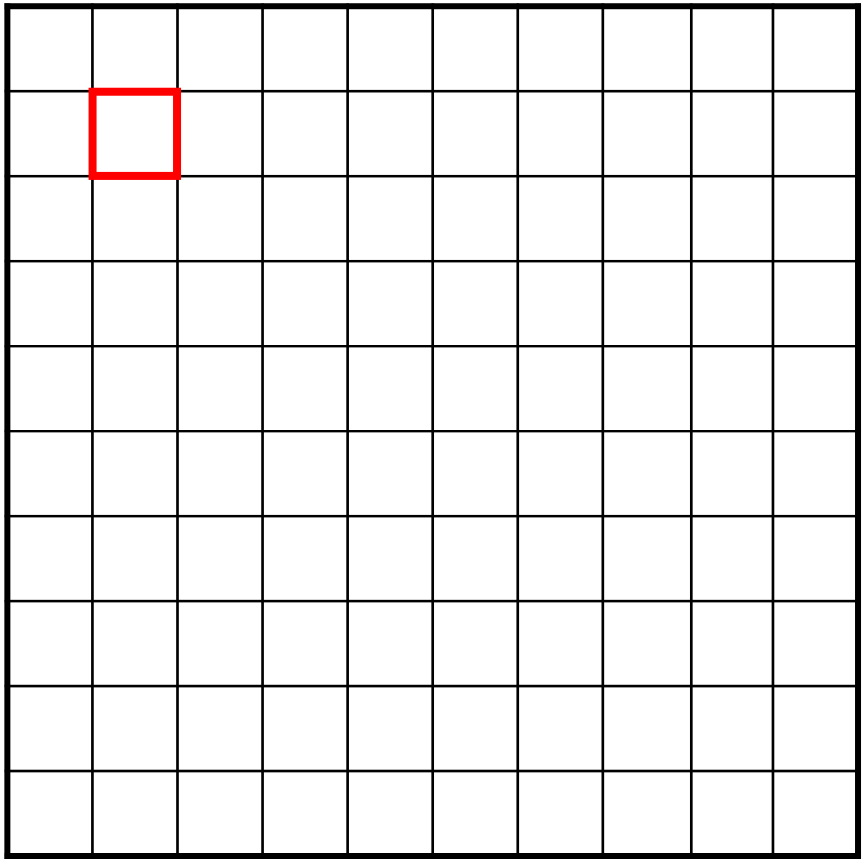
输入

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$H=F\otimes G$

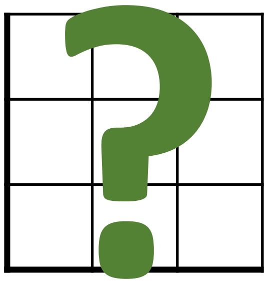
令滑动平均的窗口大小为$(2K+1) \times (2K+1)$

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

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



$G[u,v]$

$\otimes$

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

$F[x,y]$

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

$G[u, v]$

$\otimes$

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

$F[x, y]$

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

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

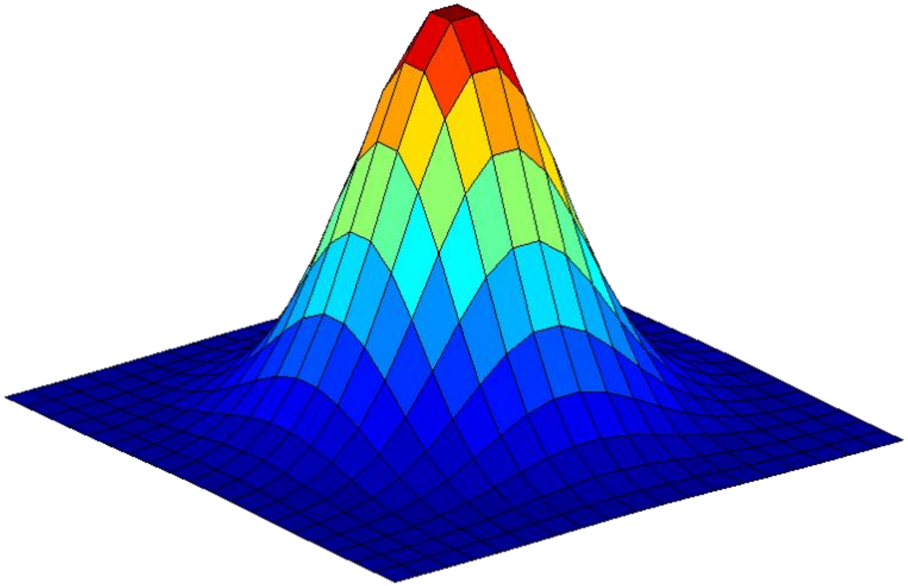
$F[x, y]$

高斯滤波

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \approx$$

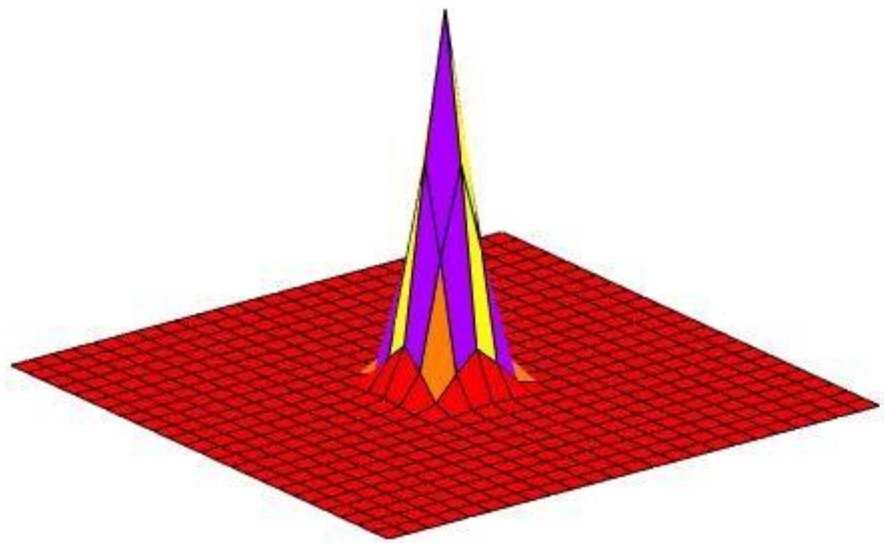
1	2	1
2	4	2
1	2	1

$G[u,v]$



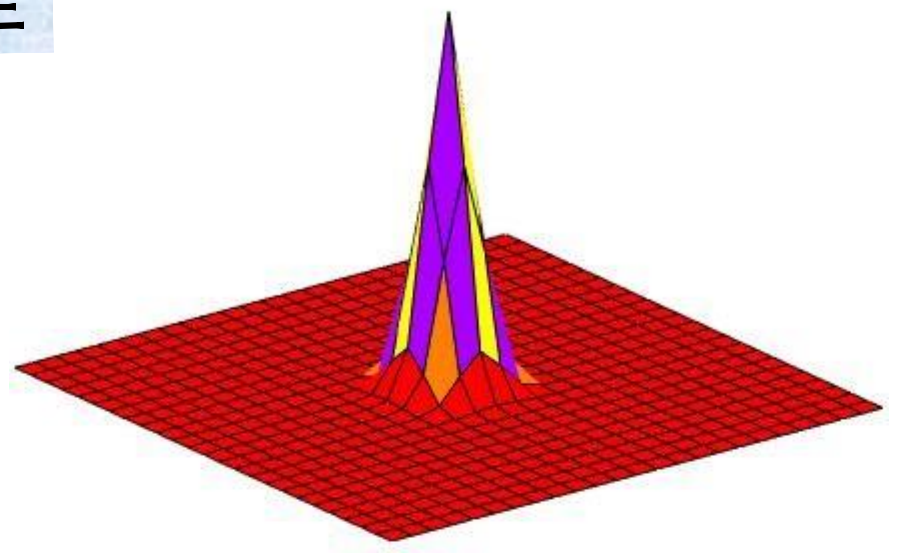
$$\frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

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



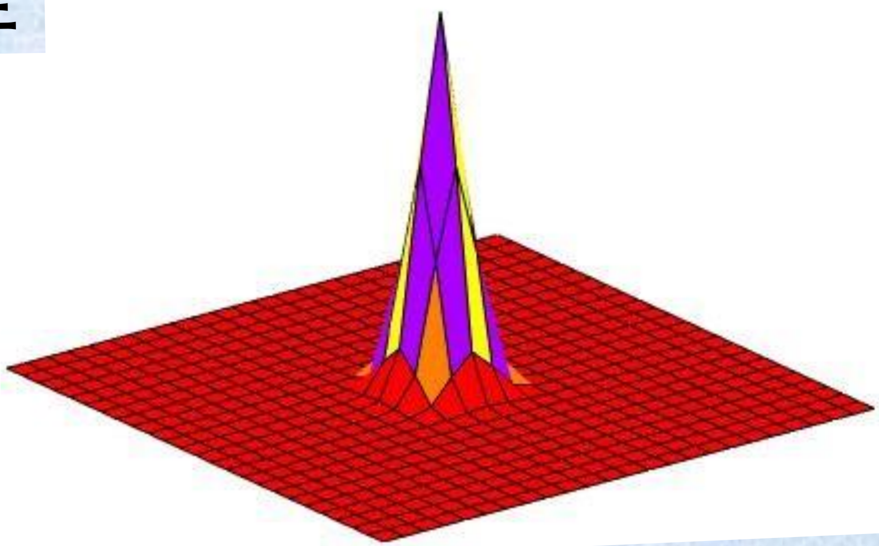
$$\frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

标准差



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

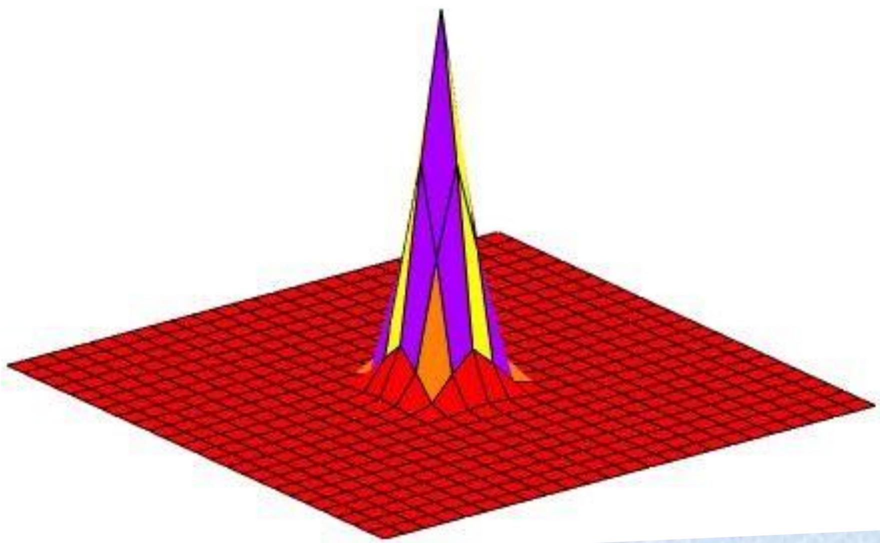
标准差



高斯函数是如何随σ变化的?

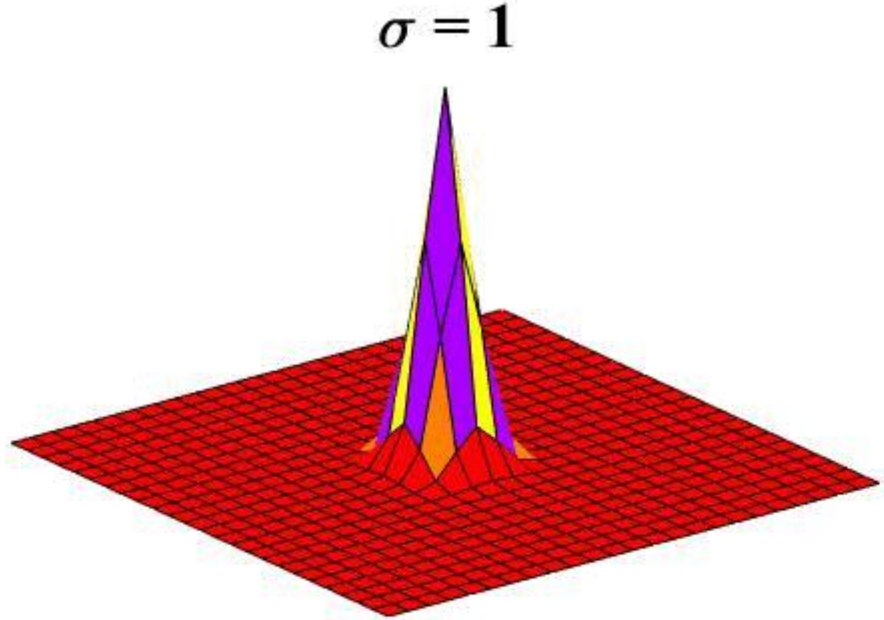
归一化

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



高斯函数是如何随σ变化的?

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

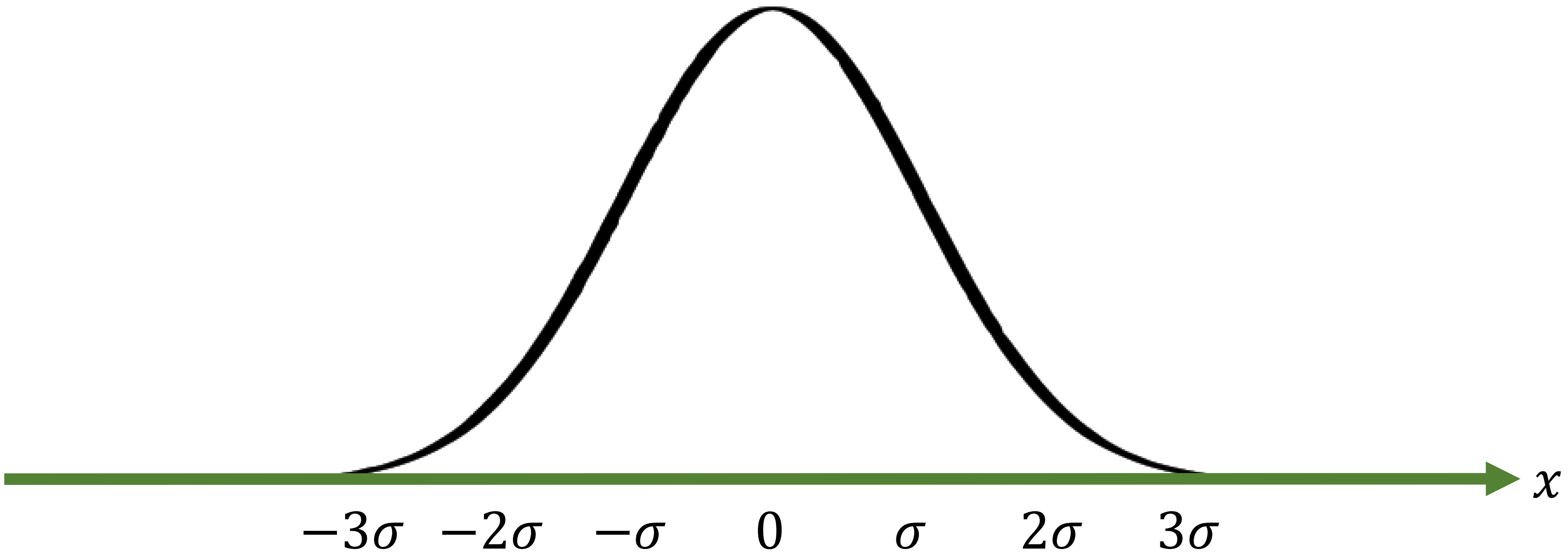


高斯滤波器有**无限长的支撑**

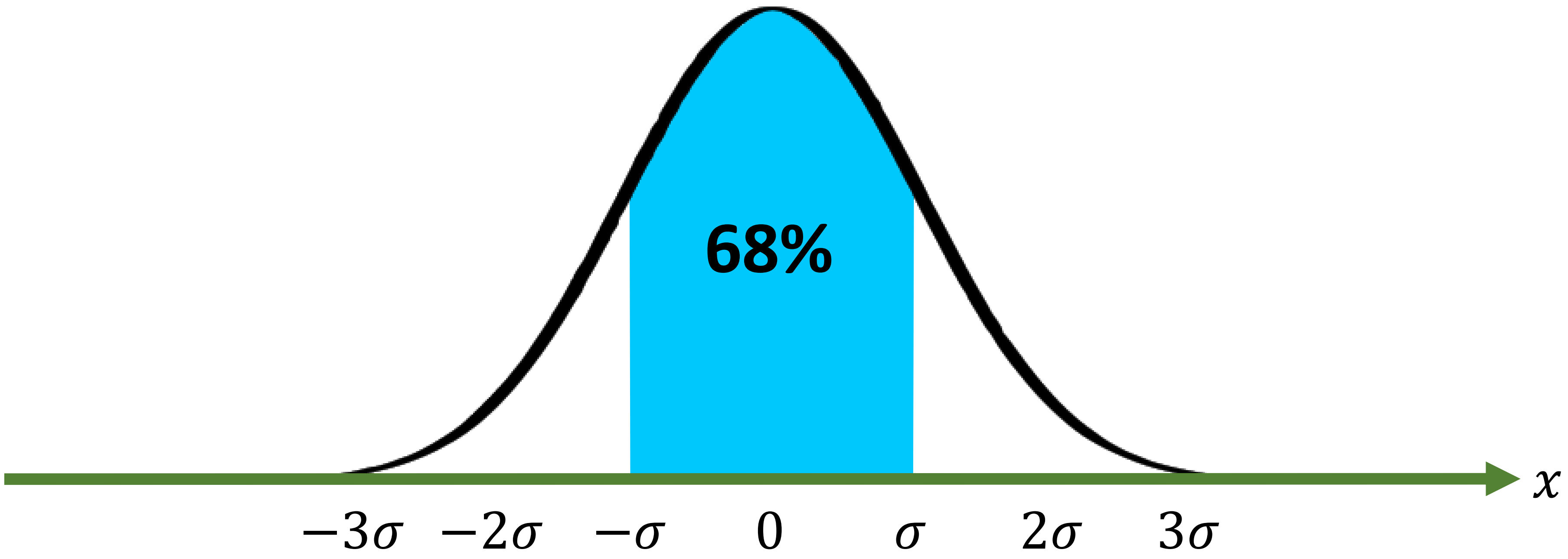
高斯滤波器有无限长的支撑

离散滤波器使用有限大小的核函数

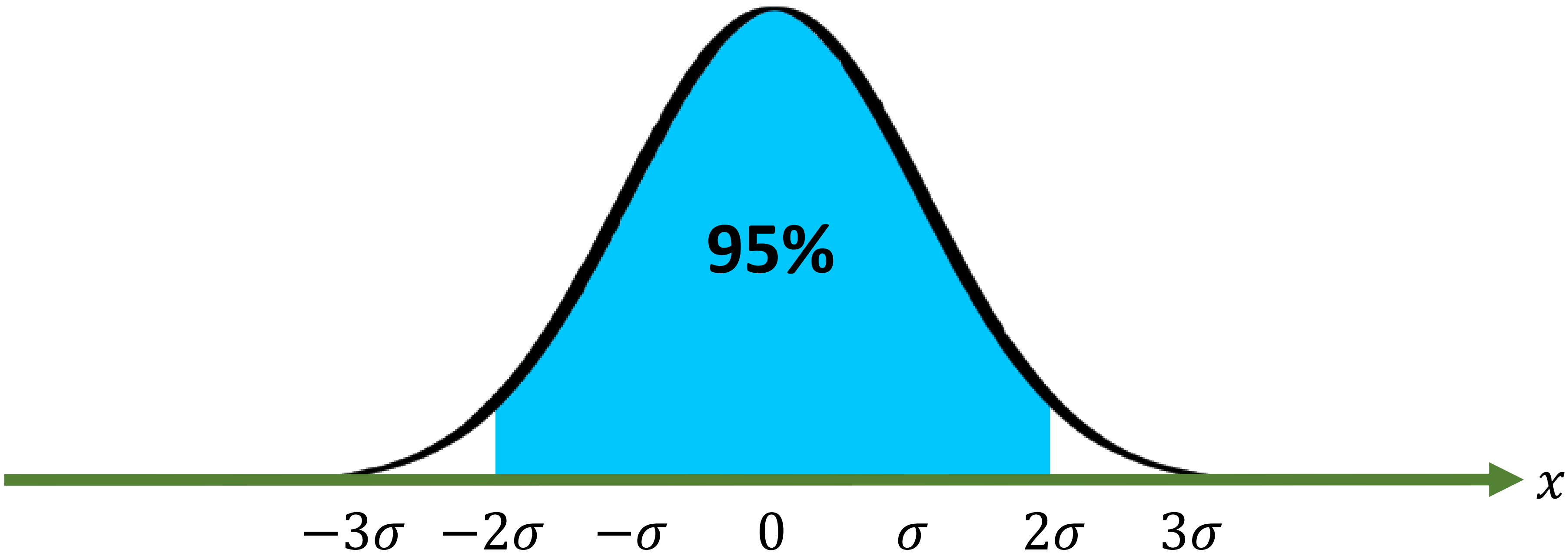
3-σ法则



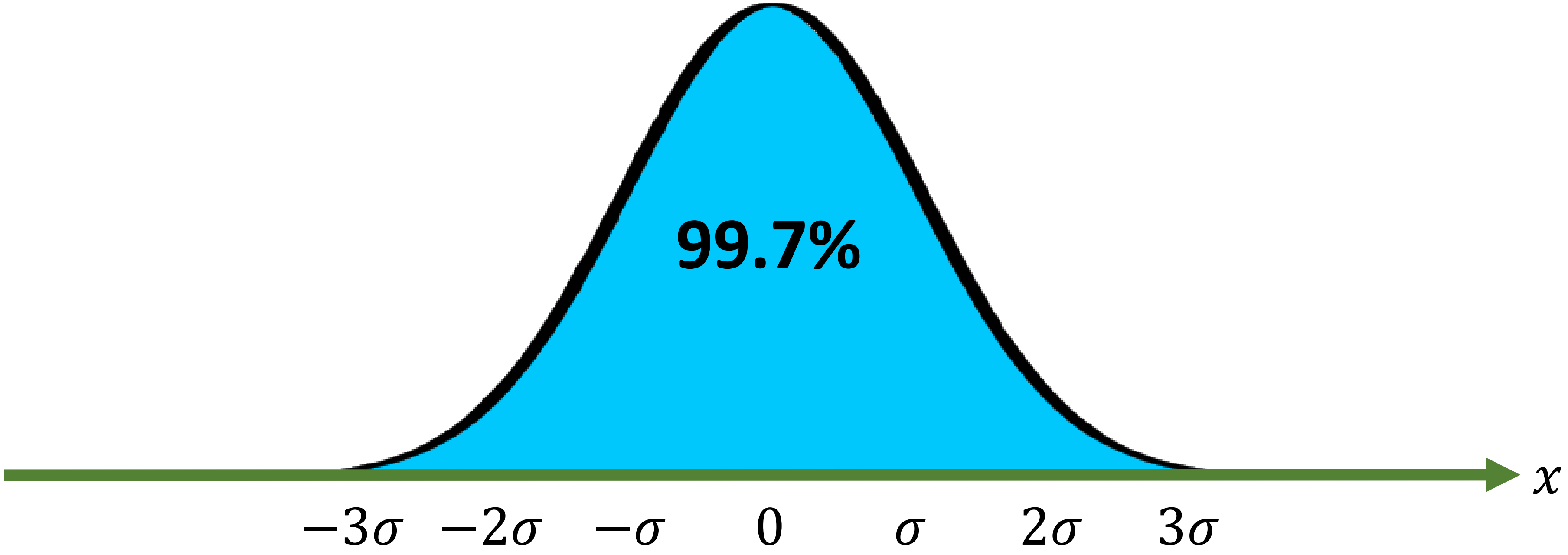
3-σ法则



3-σ法则



3-σ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)
```

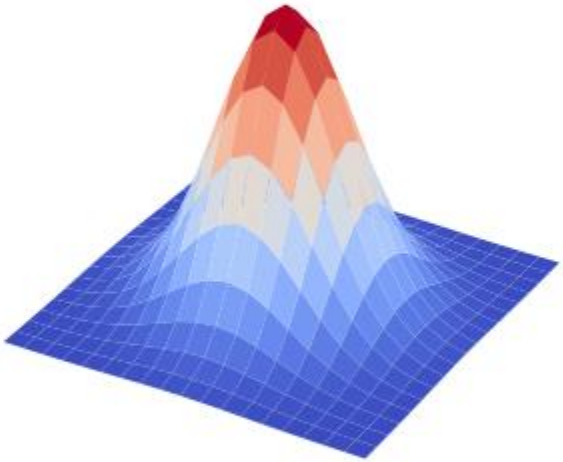
```
>> sigma = 16
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
>> I = im + noise
>> I = (I – I.min()) / (I.max() - I.min())
>> cv2.imshow('noised', I)
>> cv2.waitKey(0)
```

```
>> 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)
```

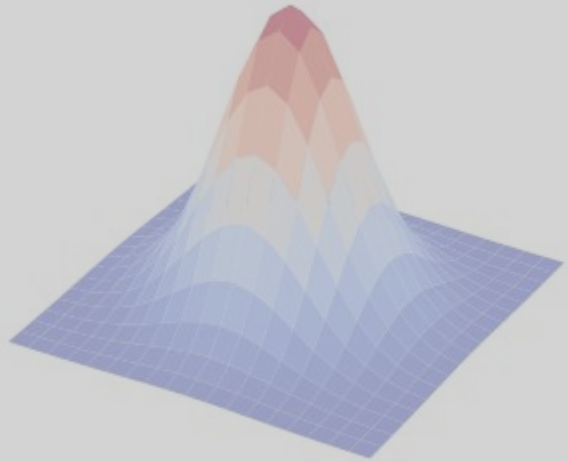
```
>> sigma = 16
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
>> I = im + noise
>> I = (I – I.min()) / (I.max() - I.min())
>> cv2.imshow('noised', I)
>> cv2.waitKey(0)
```

```
>> 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)
```

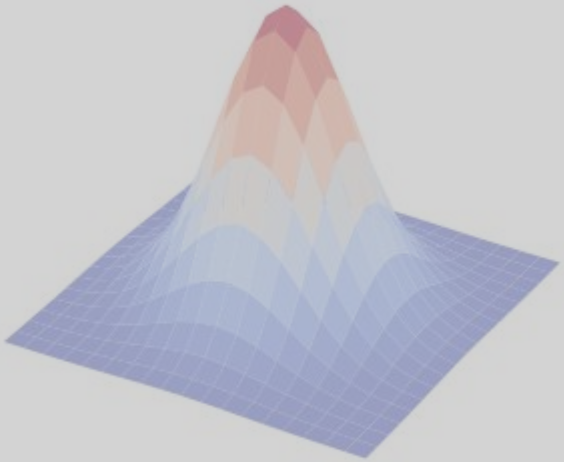
```
>> sigma = 16
>> noise = np.random.randn(im.shape[0], im.shape[1]) * sigma
>> I = im + noise
>> I = (I – I.min()) / (I.max() - I.min())
>> cv2.imshow('noised', I)
>> cv2.waitKey(0)
```



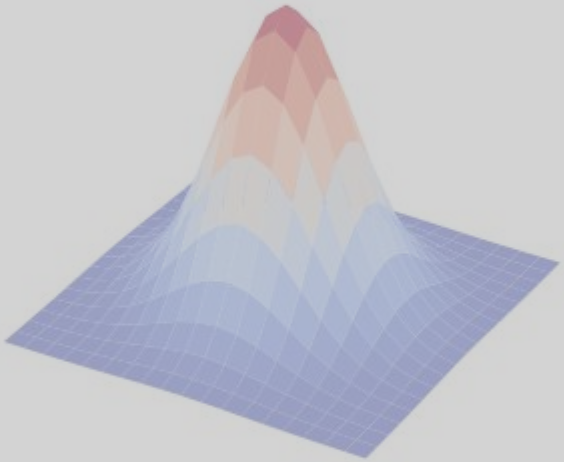
```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z,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



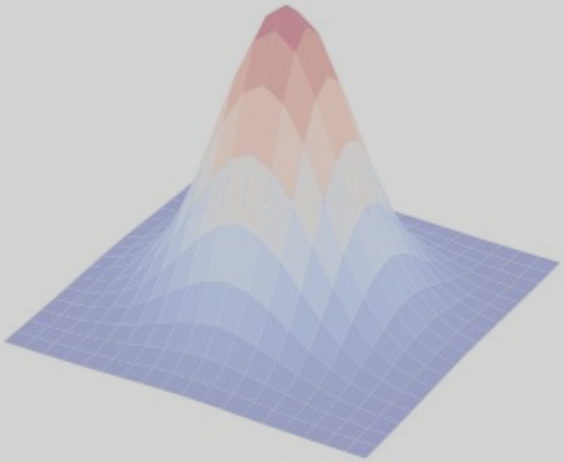
```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z,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

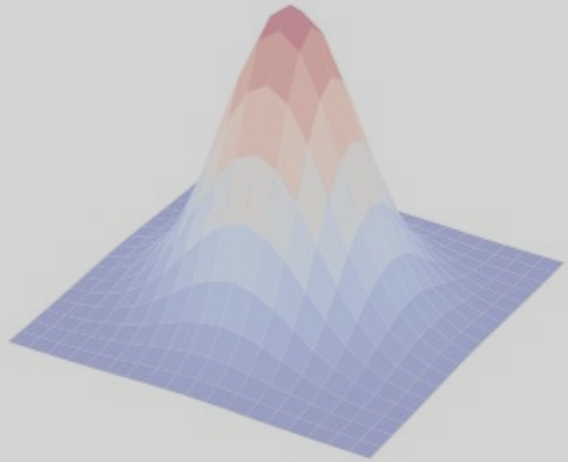


```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z,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



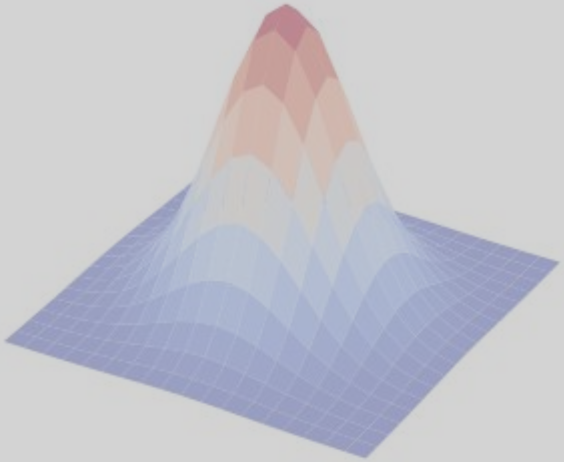
```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 10, 2
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z,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

转置

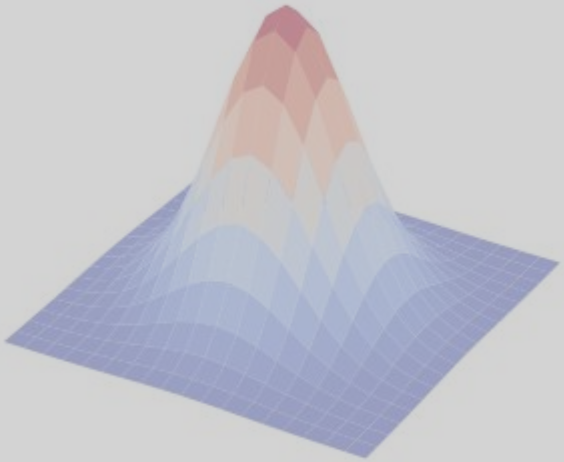


矩阵乘法

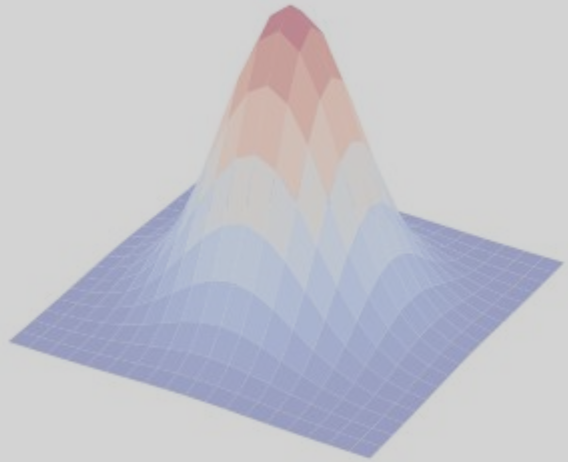
```
>>> from matplotlib import pyplot, cm
>>> width, sigm
>>> h = cv                    nel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)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

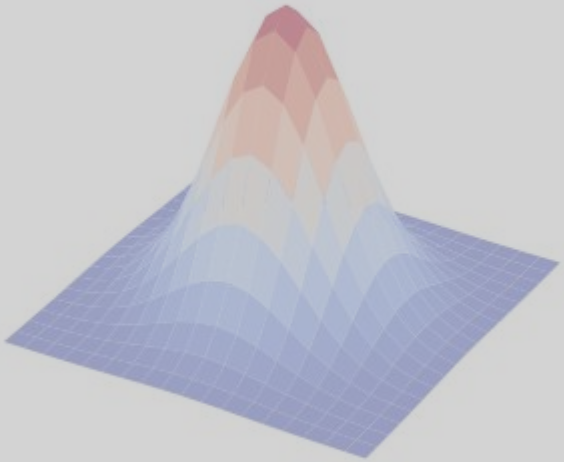


```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKernel(width, sigma)
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z,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

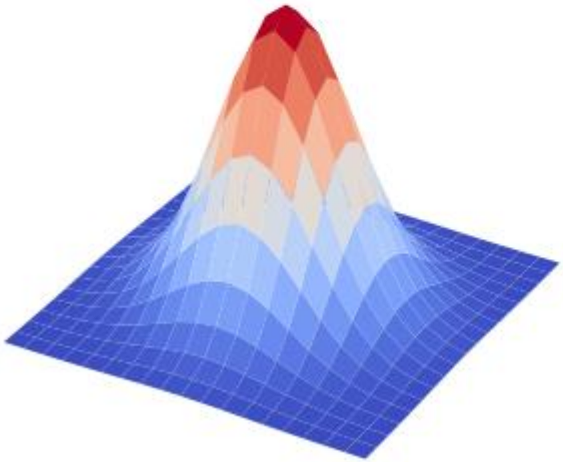


```
>>> from matplotlib import pyplot, cm
>>> width, sigma = 19, 3
>>> h = cv2.getGaussianKern
>>> Z = h @ h.T
>>> X = Y = np.arange(-width//2, width//2, 1)
>>> X, Y = np.meshgrid(X, Y)
>>> fig, ax = pyplot.subplots(subplot_kw={'projection': '3d'})
>>> surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm)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

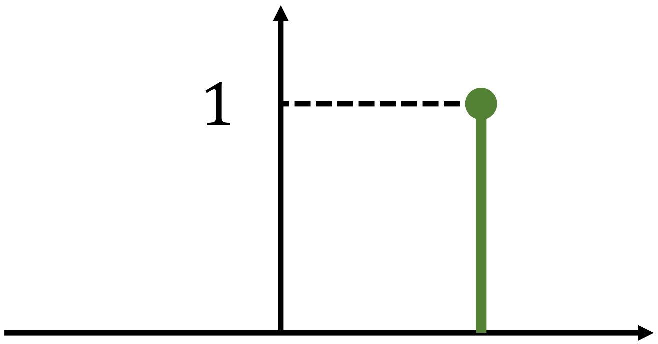
边界问题

对称滤波

cv2.BORDER_REFLECT

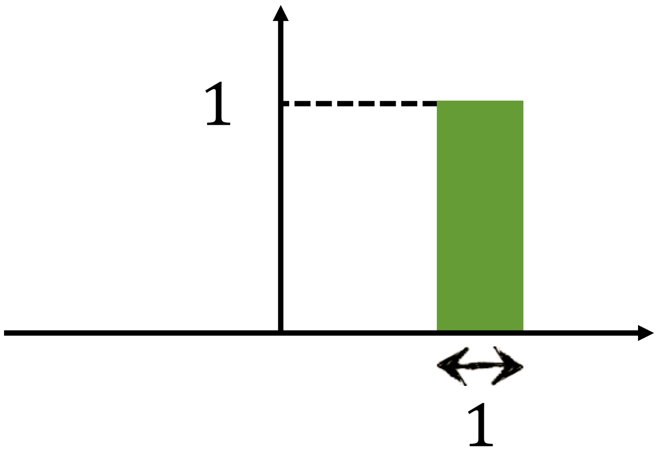
Python时间

离散
冲击响应



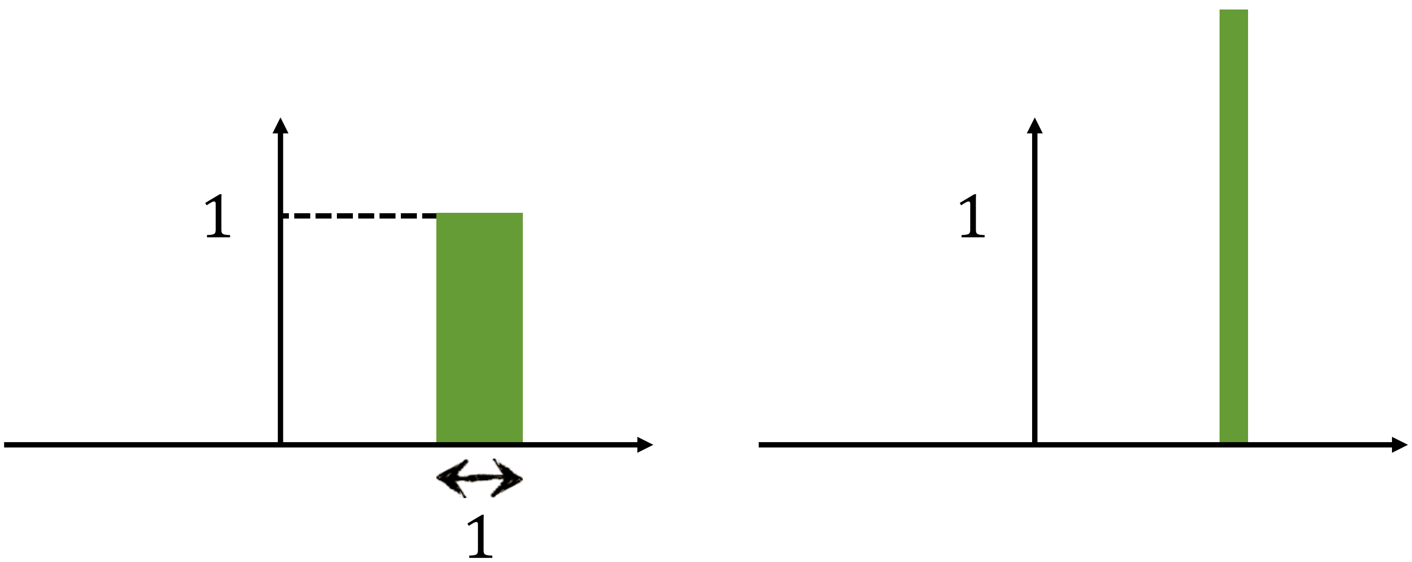
在单个位置上值为1的函数

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

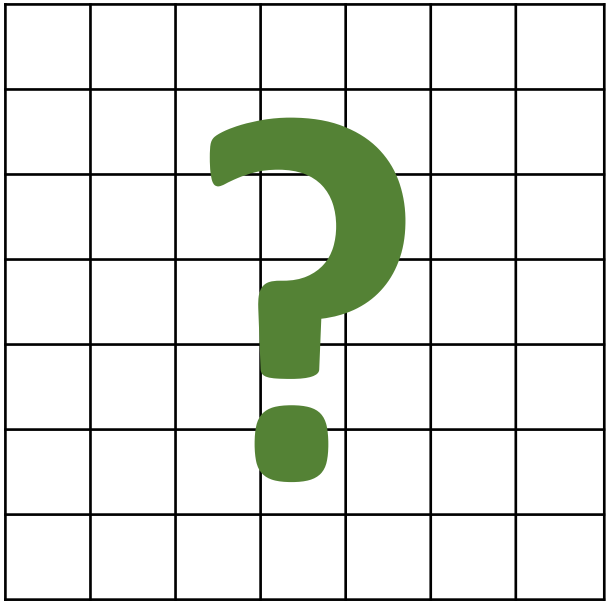
$G[u,v]$

$\otimes$

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

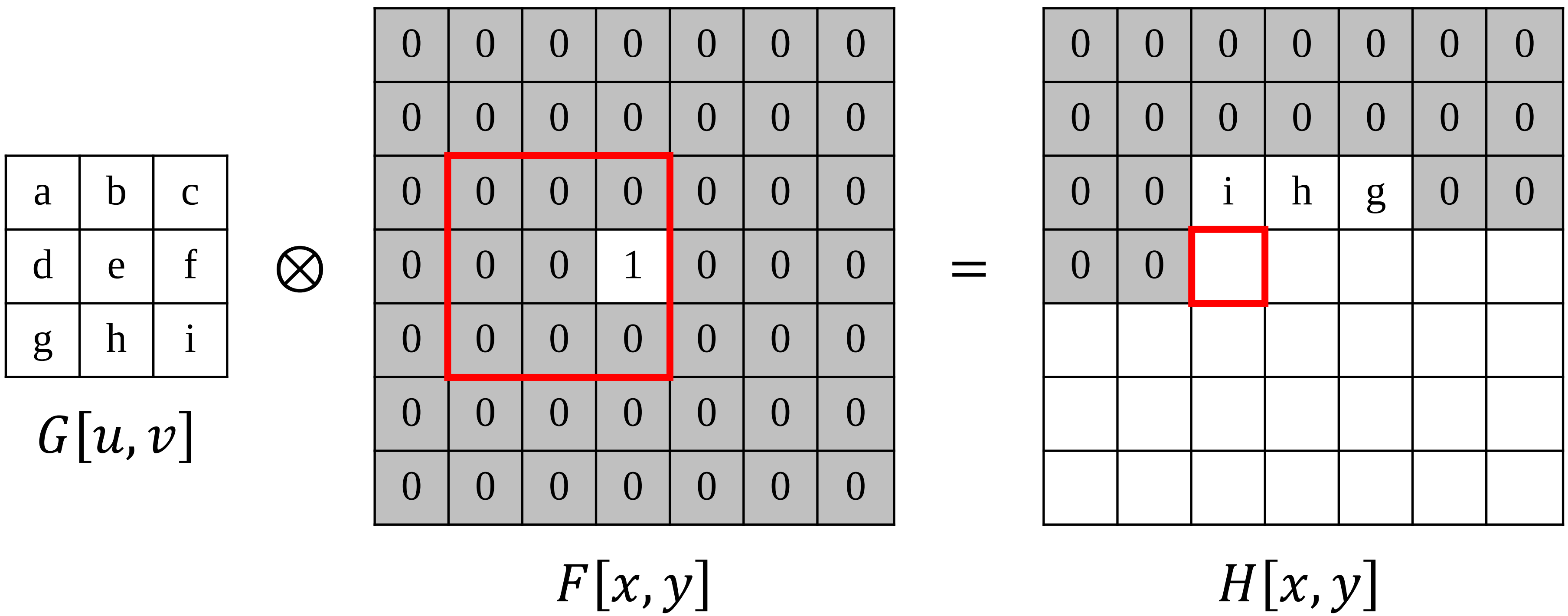
$F[x,y]$

$=$

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$H=F\otimes G$

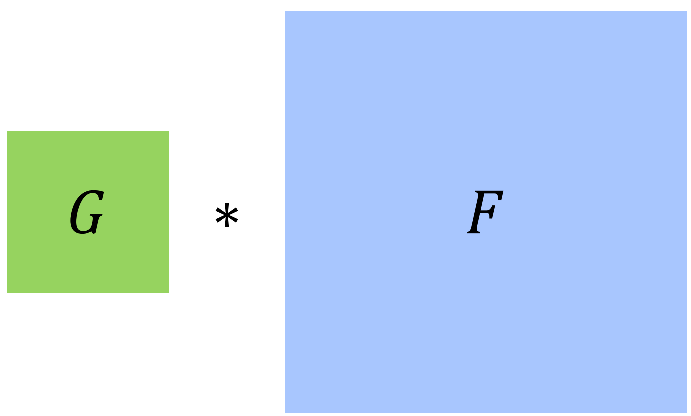
令滑动平均的窗口大小为$(2K+1)\times(2K+1)$

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

该式叫作互相关，记作$H=F\otimes G$

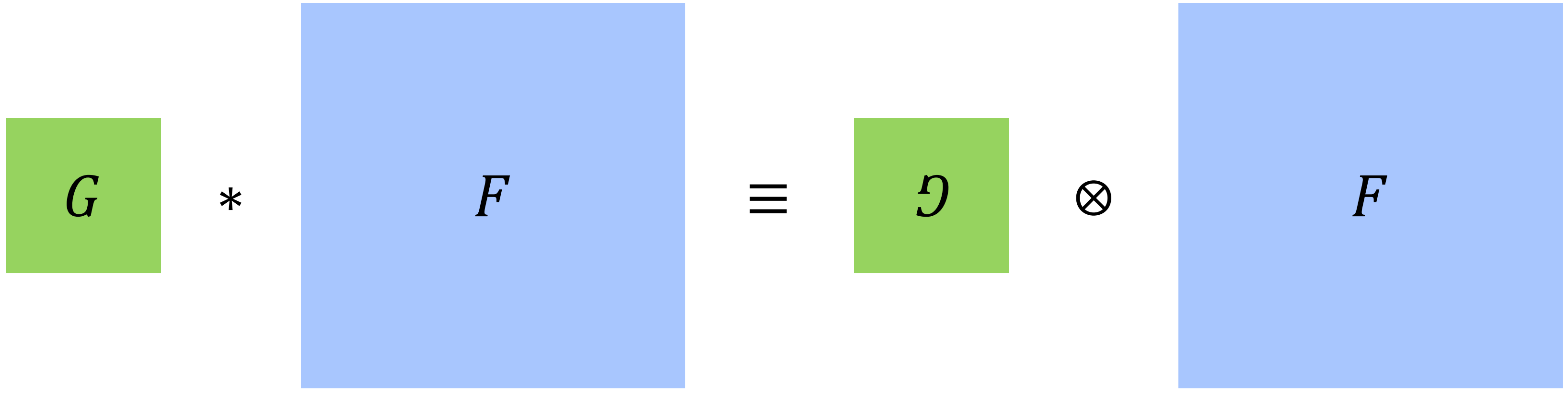
$$H[x,y] = \sum_{u=-K}^{K} \sum_{v=-K}^{K} F[x-u, y-v] G[u,v]$$

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

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

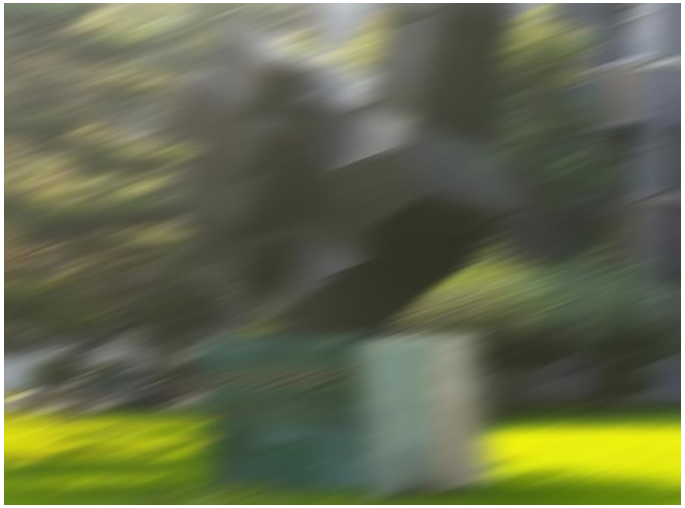
交换律

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

证明略…


OFF
ON
RAW JPEG
MENU
INFO.
START STOP
AF-ON
LOCK
Canon

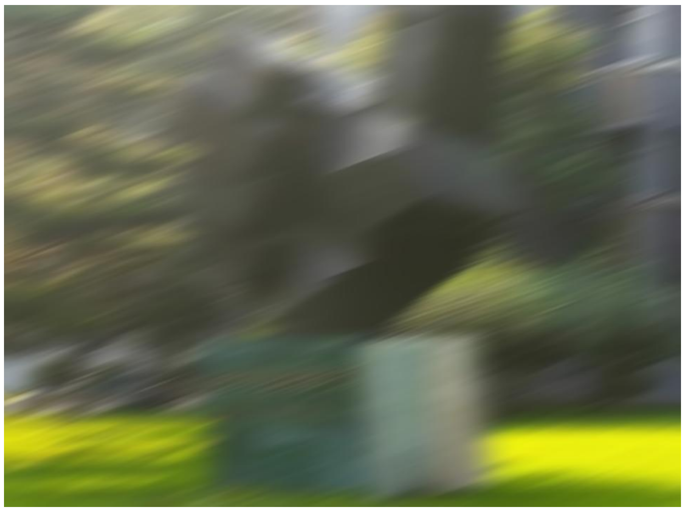
运动模糊



=

模糊图像

运动模糊

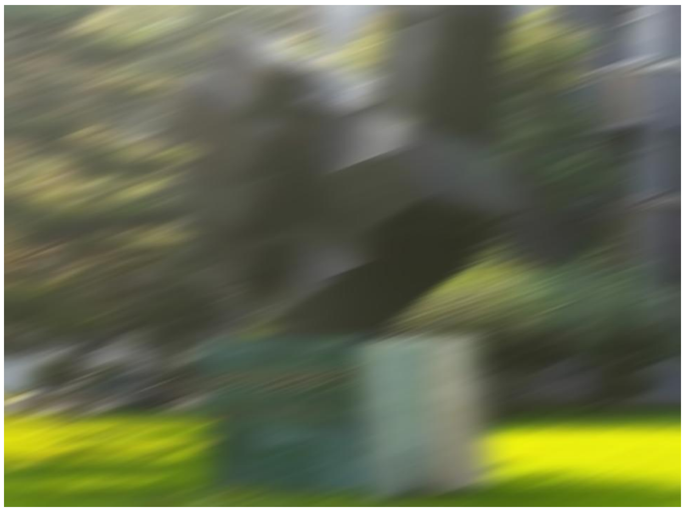


=

模糊图像

完美图像

运动模糊



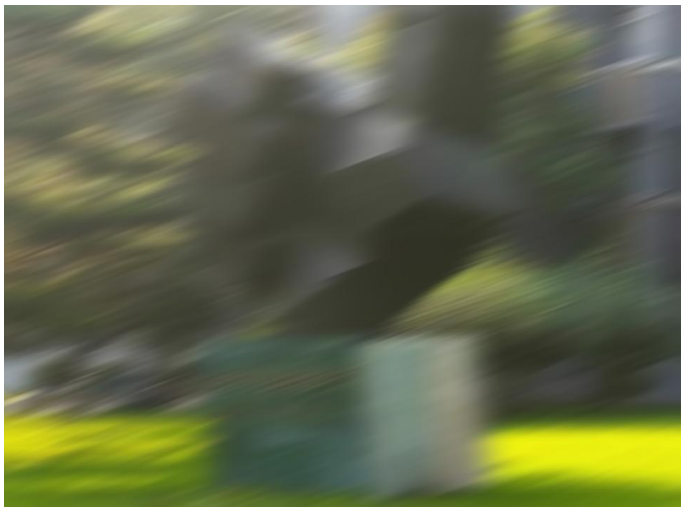
=

*

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?

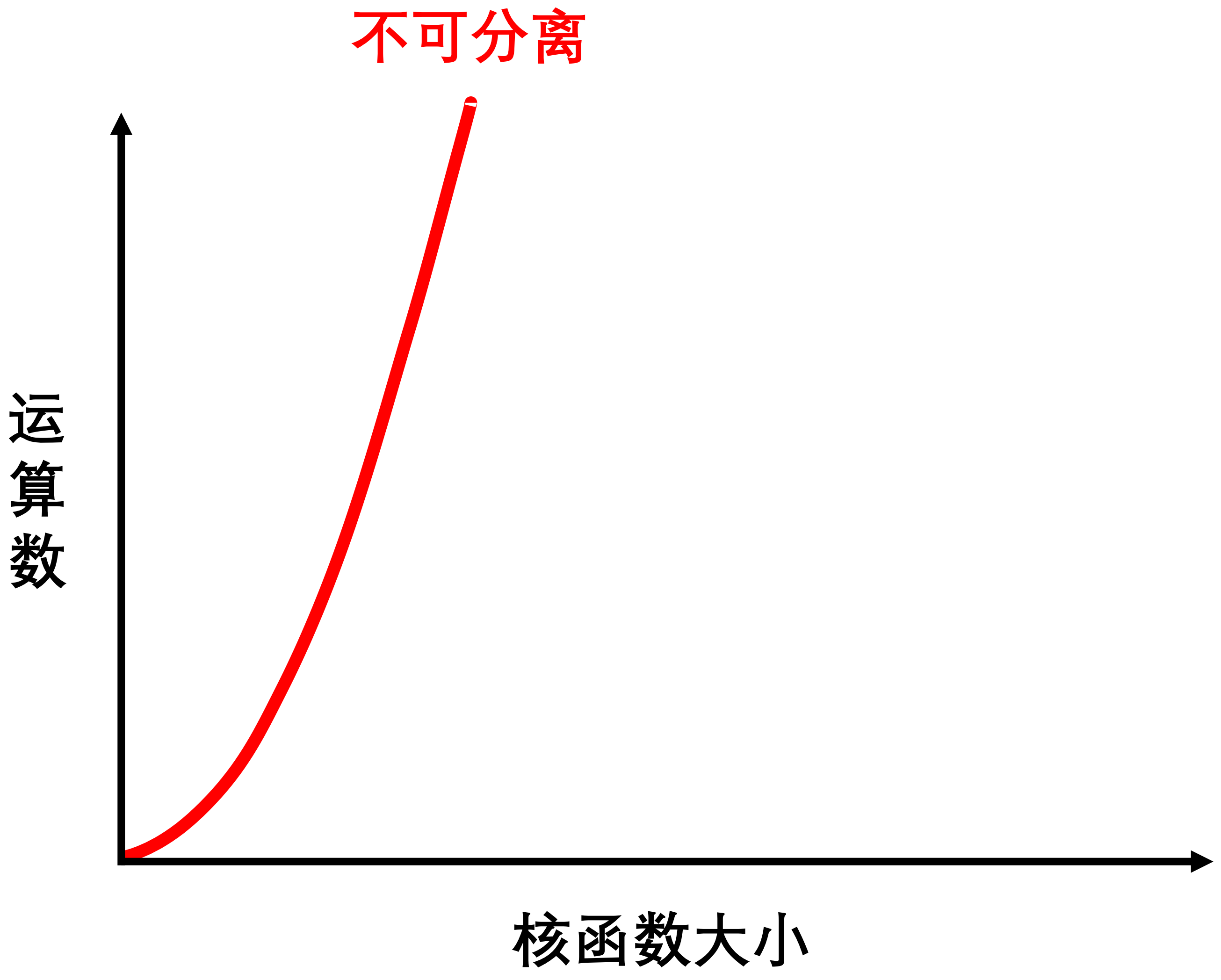
$$2(2K+1)$$

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

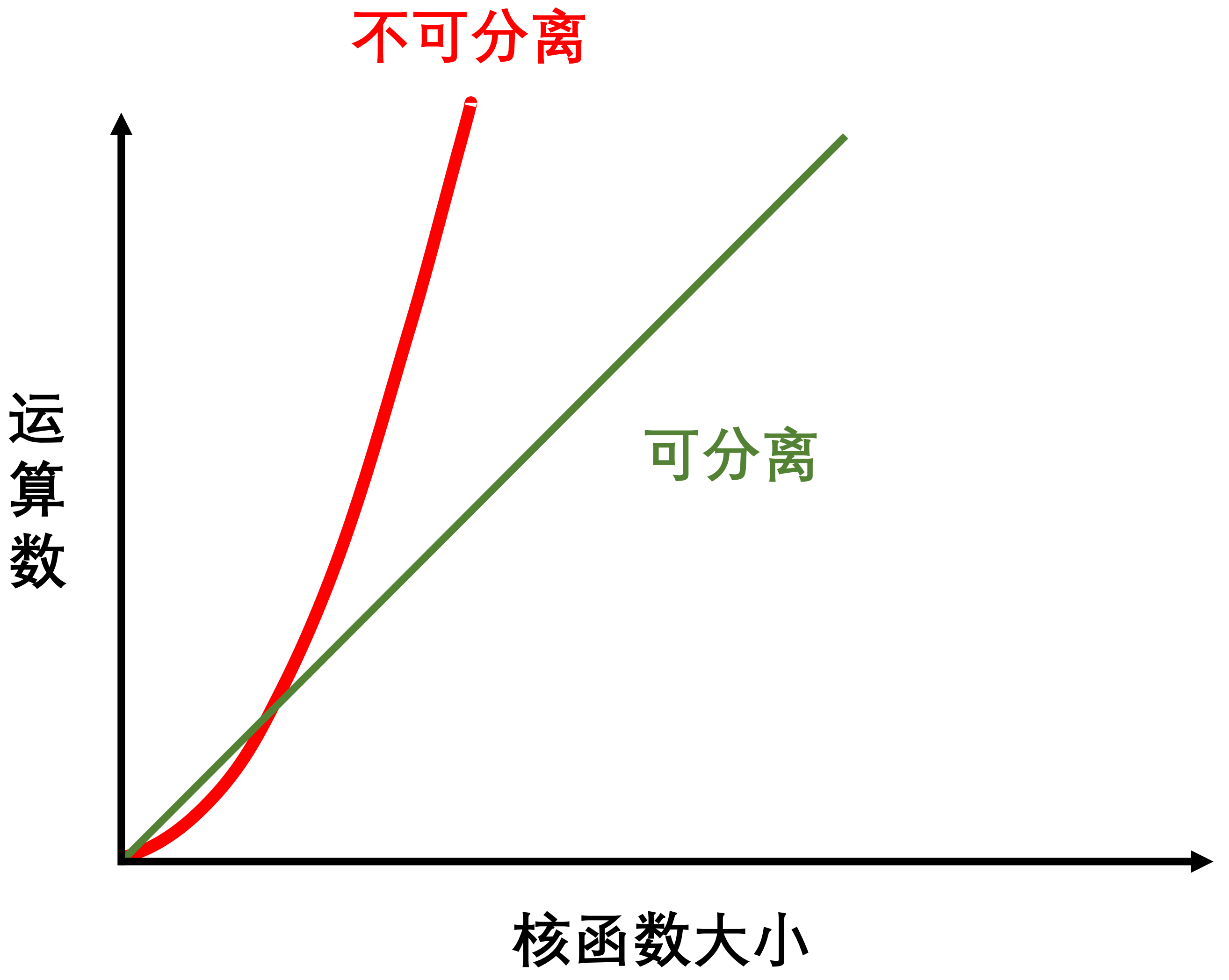
卷积每像素需要多少个操作?

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

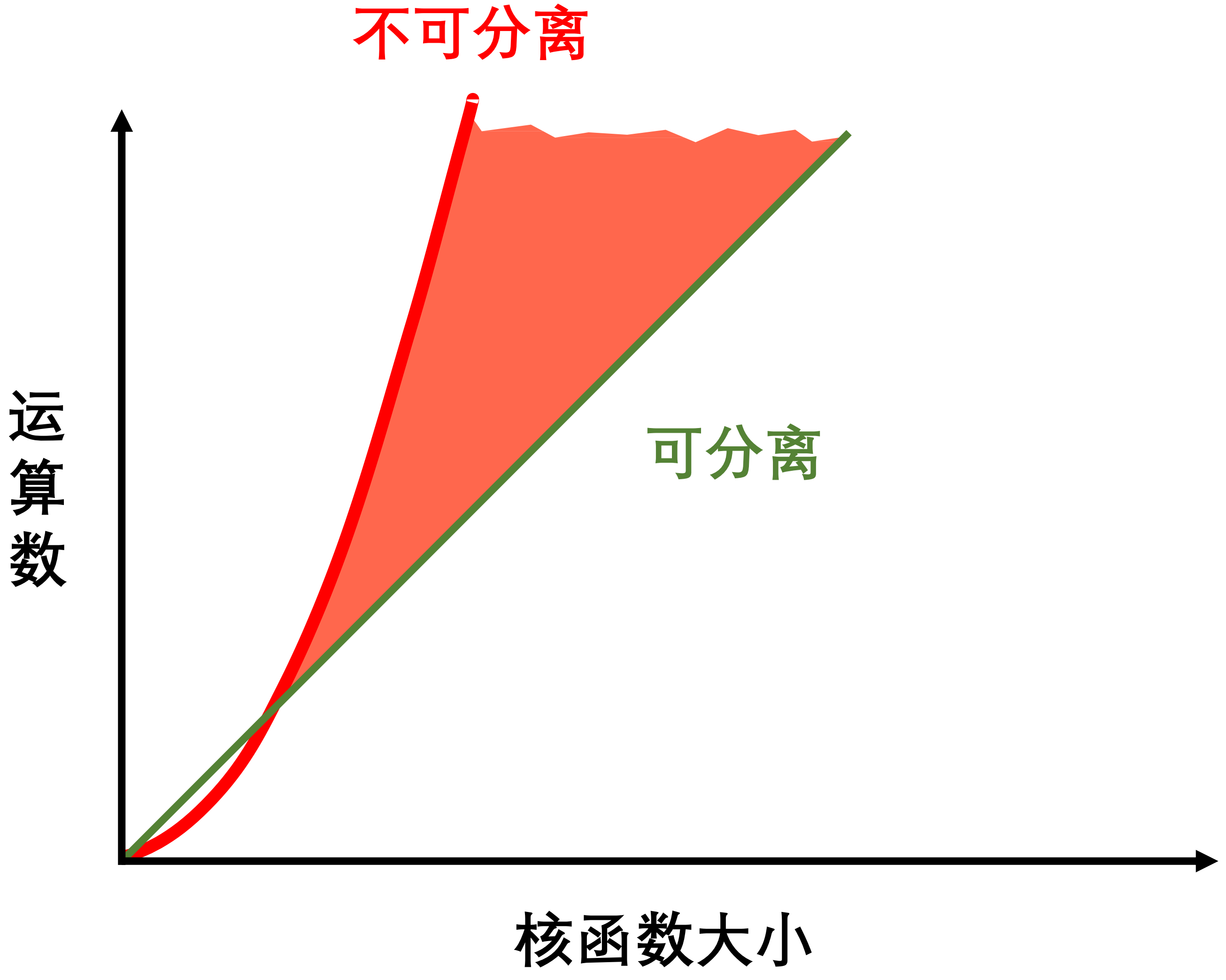

运算数
核函数大小



不可分离
运算数
核函数大小



不可分离
可分离
运算数
核函数大小


不可分离
运算数
可分离
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

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

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

证明2D高斯滤波器是可分离的

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

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

$$= \left[\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)\right] \left[\frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{y^2}{2\sigma^2}\right)\right]$$

证明2D高斯滤波器是可分离的

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

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

$$= \left[\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{x^2}{2\sigma^2}\right)\right]\left[\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{y^2}{2\sigma^2}\right)\right]$$

$$= G_1(x)G_1(y)$$

回顾

互相关 $H = G \otimes F$

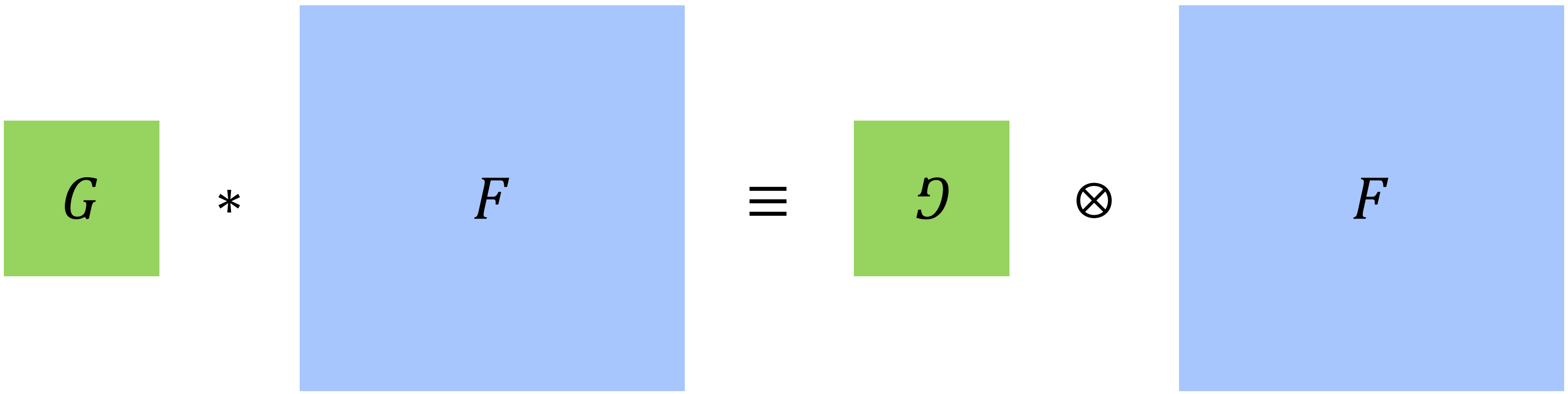
$$H[x,y] = \sum_{u=-K}^{K} \sum_{v=-K}^{K} G[u,v] F[x+u, y+v]$$

卷积 $H = G * F$

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

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

输入

$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} = \ ?$$

输入

*

0	0	0
0	0	1
0	0	0

=

向右平移1像素

$$* \ \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \ ?$$

输入

$$* \ \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$

输入

模糊了

Image kernels explained v ×

setosa.io/ev/image-kernels/

Back

Image kernels

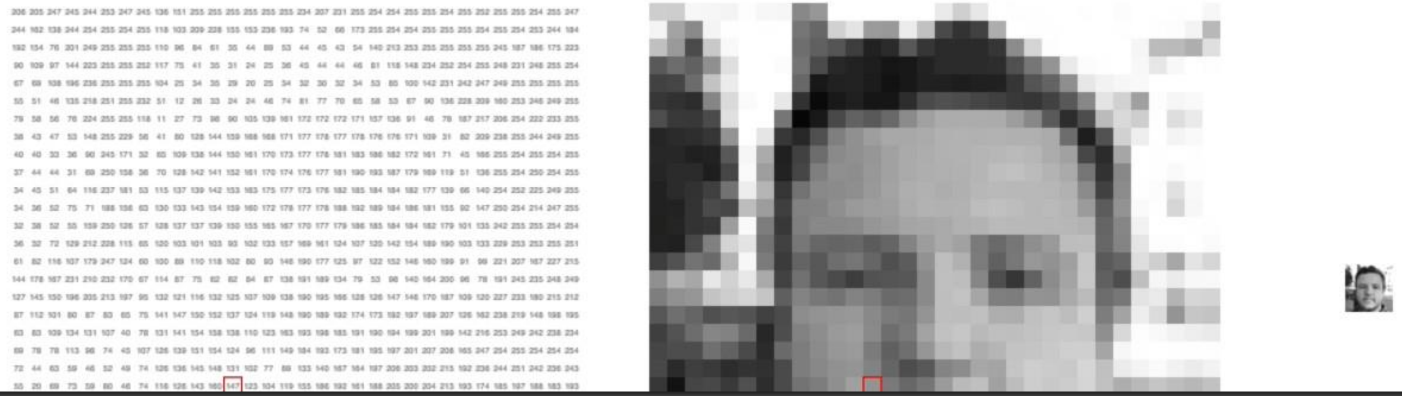
Explained Visually

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.



tinyurl.com/image-kernels

Image kernels

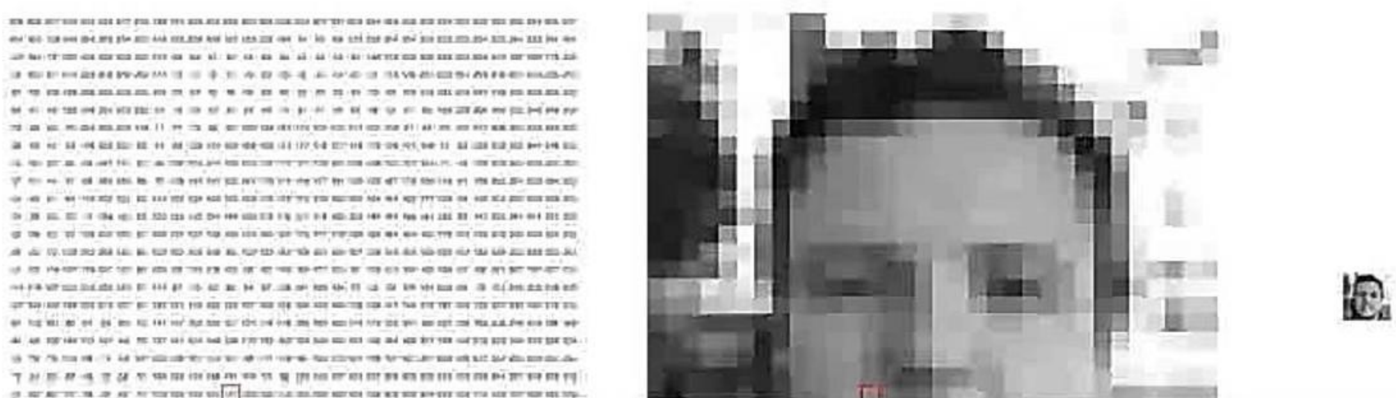
Explained Visually

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.



tinyurl.com/image-kernels

Image kernels

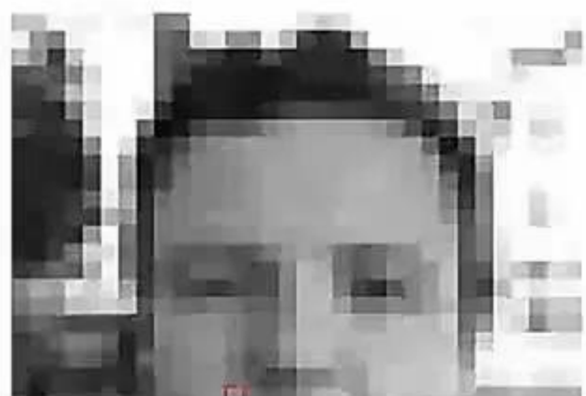
Explained Visually

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.



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

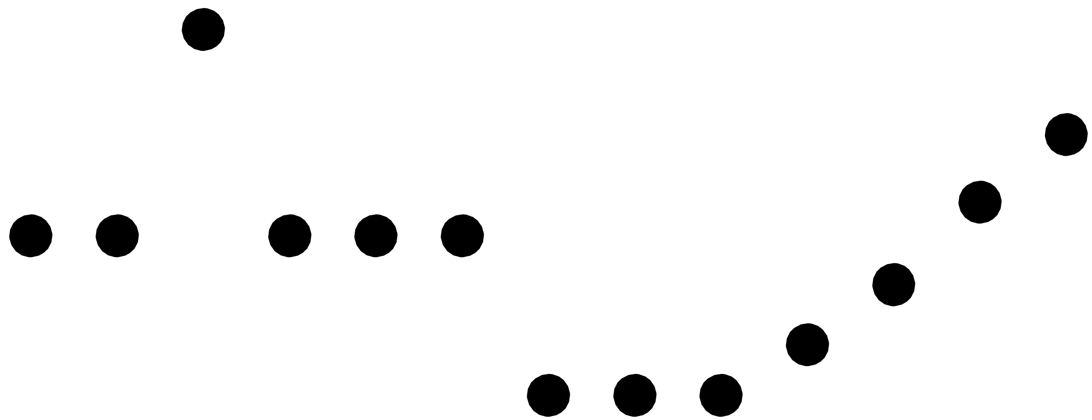
10	15	20	23	27	30	31	33	90

用中值替换像素

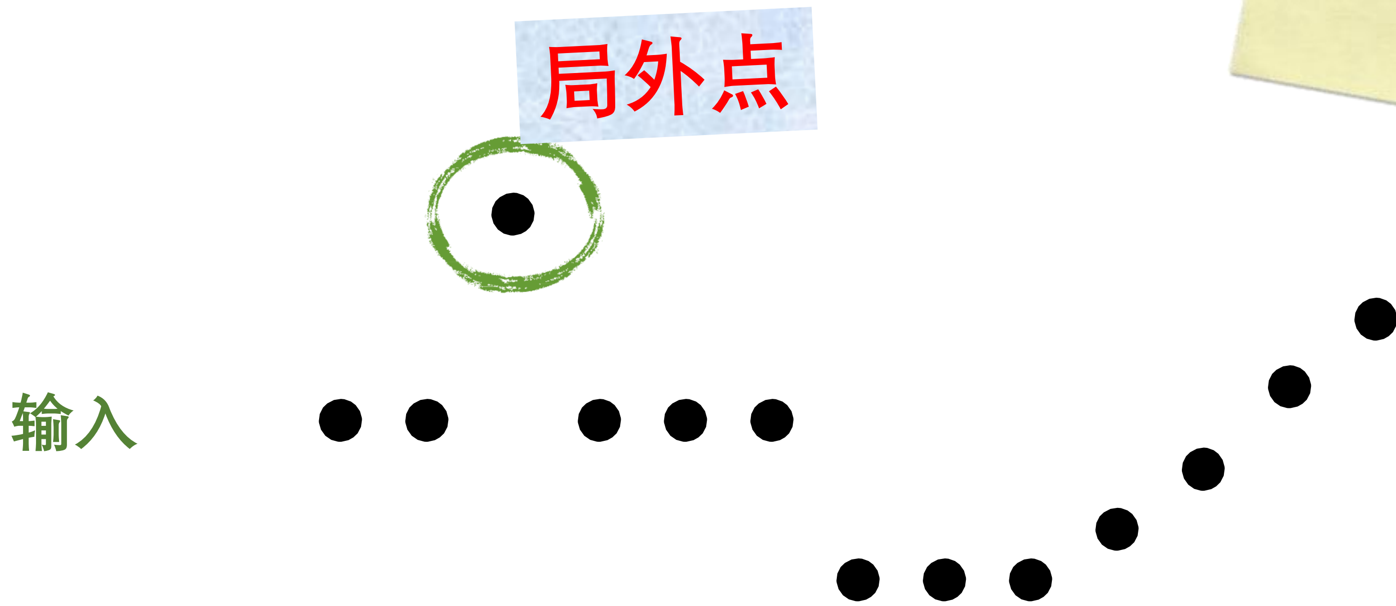
中值滤波

中值滤波

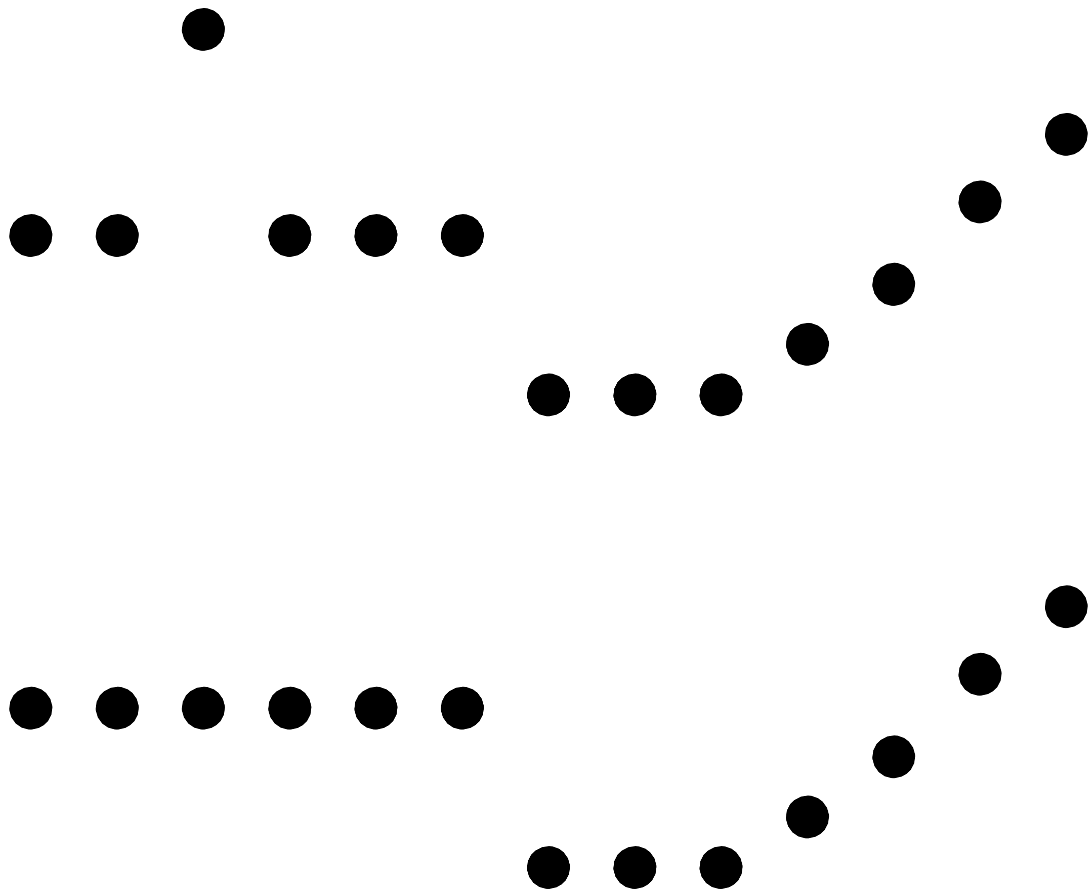
输入



中值滤波



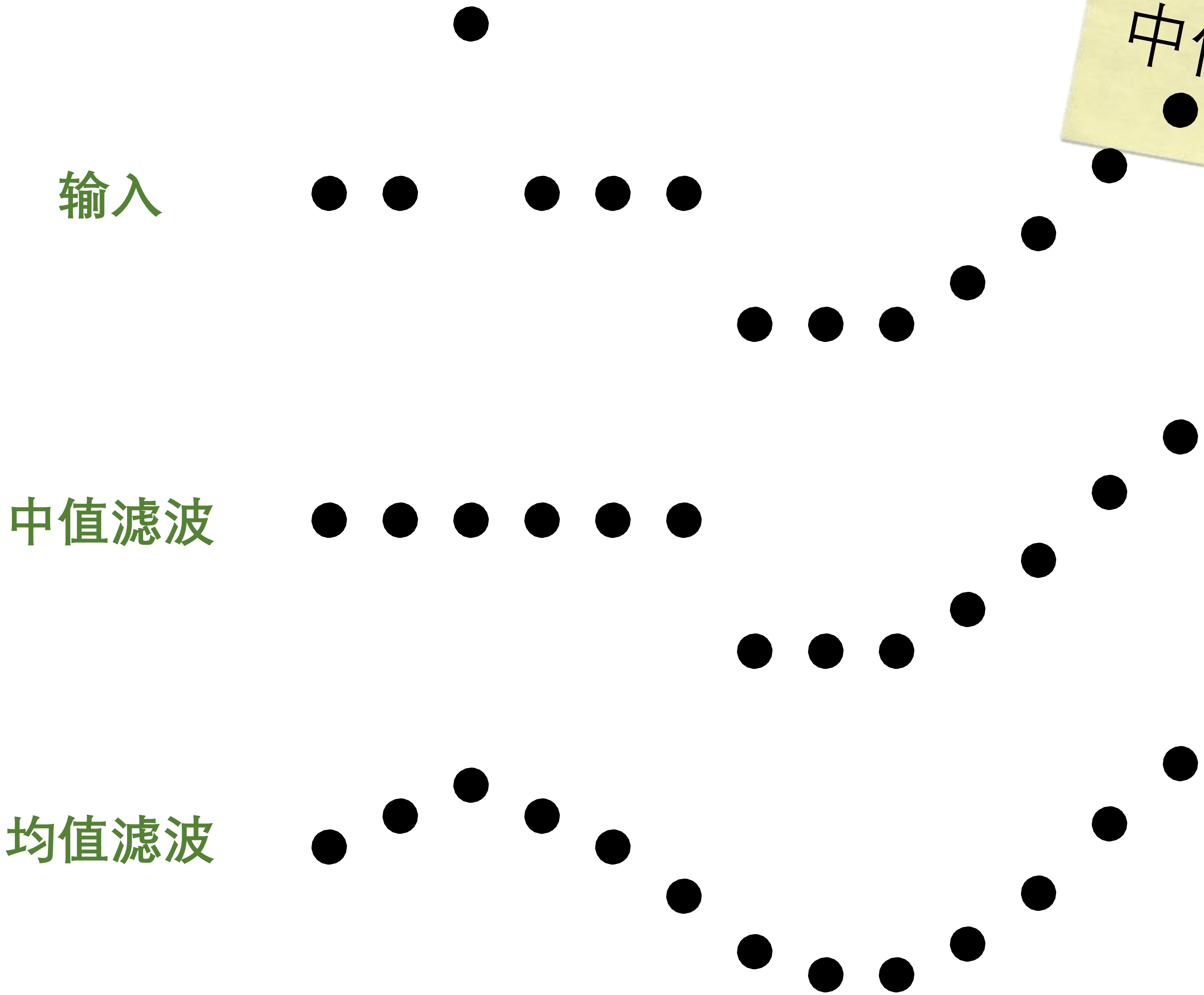
中值滤波



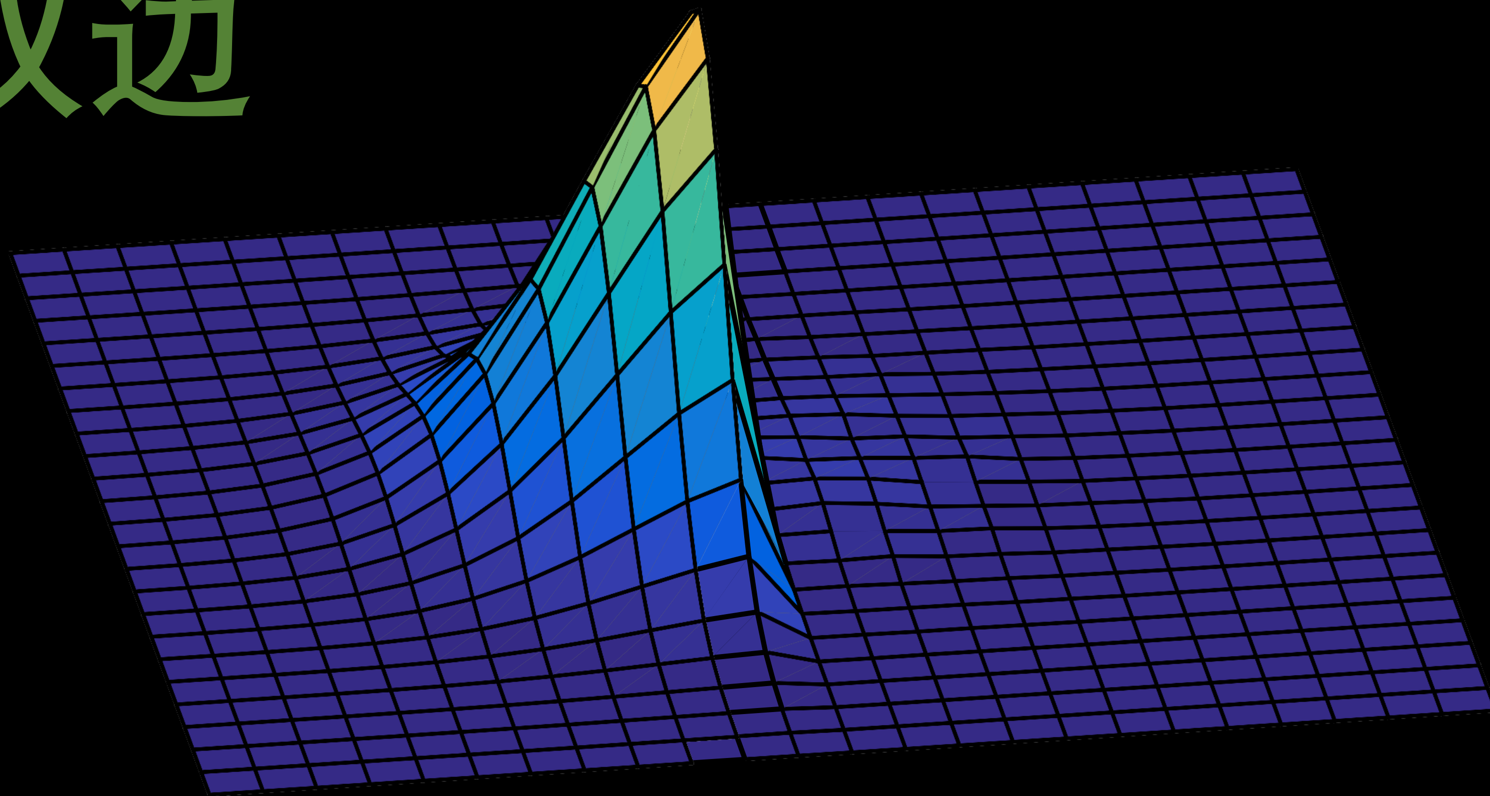
输入

中值滤波

中值滤波



双边



滤波器

Bilateral Filtering for Gray and Color Images

C. Tomasi *

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

R. Manduchi

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

Abstract

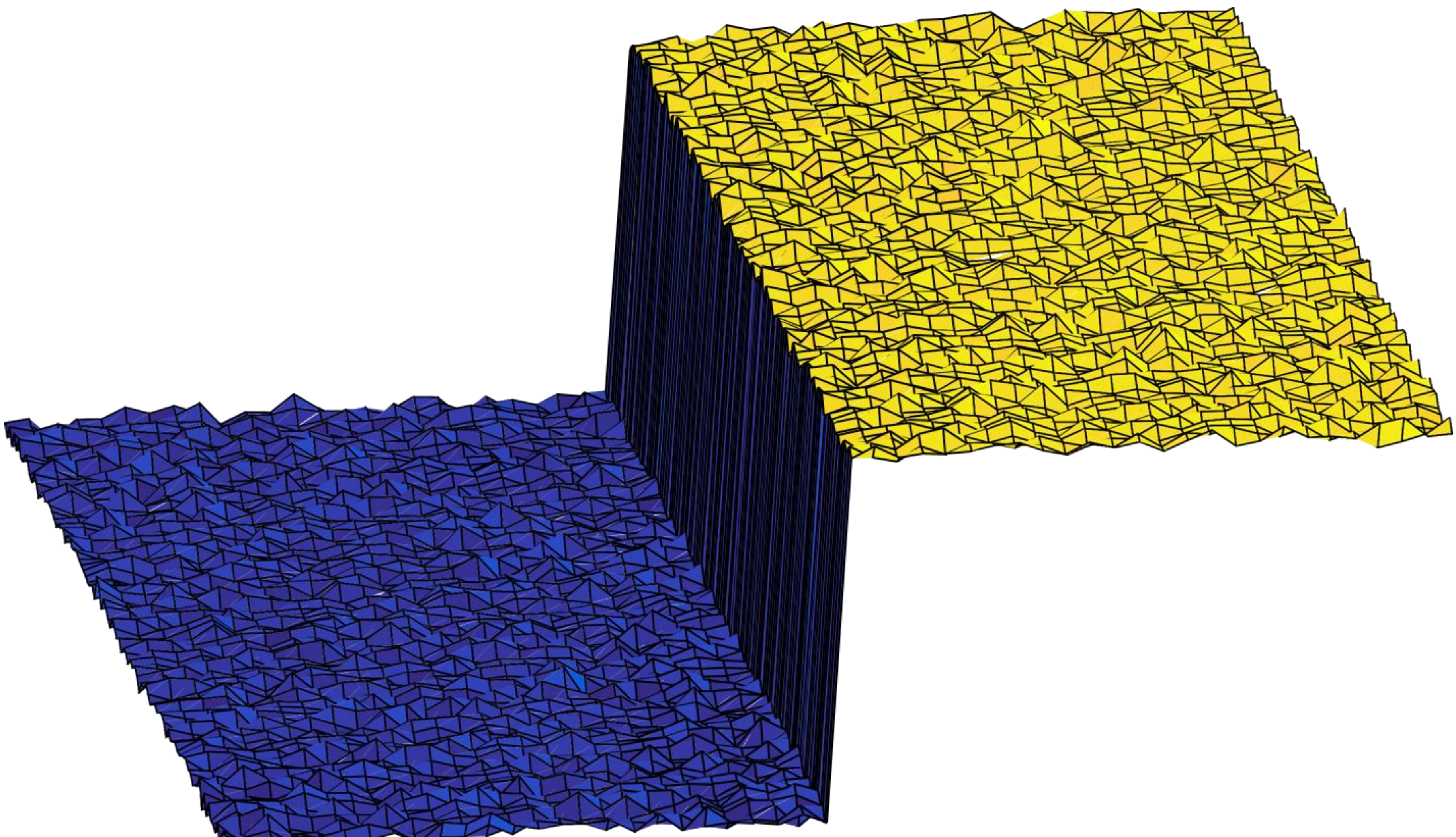
Bilateral filtering smooths images while preserving edges, by means of a nonlinear combination of nearby image values. The method is noniterative, local, and sim- ... levels or colors based on both their

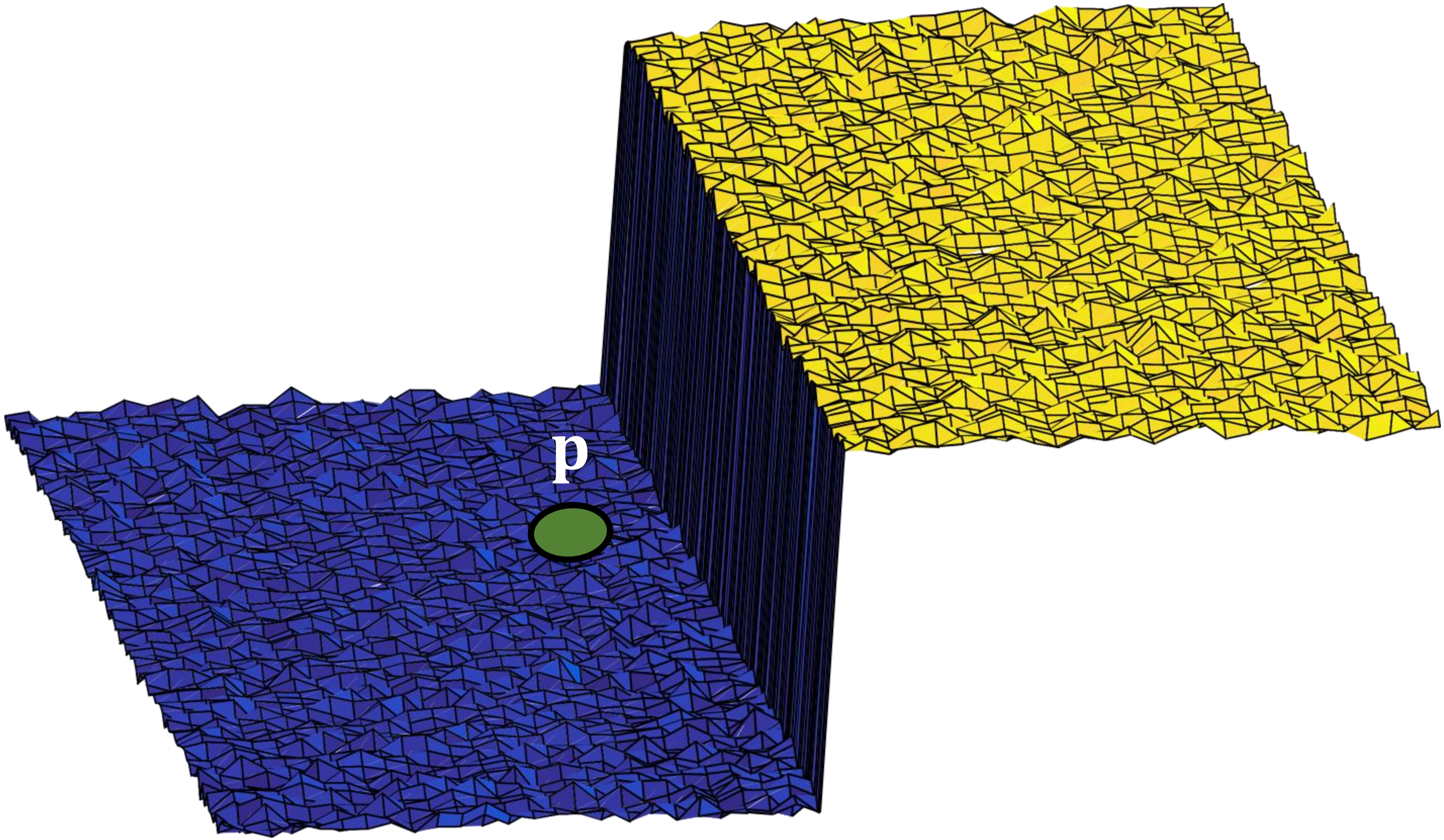
we prevent averaging across edges, while still averaging within smooth regions? Anisotropic diffusion [12, 14] is a popular answer: local image variation is measured at every point, and pixel values are averaged from neighborhoods whose size and shape depend on local variation. Diffusion

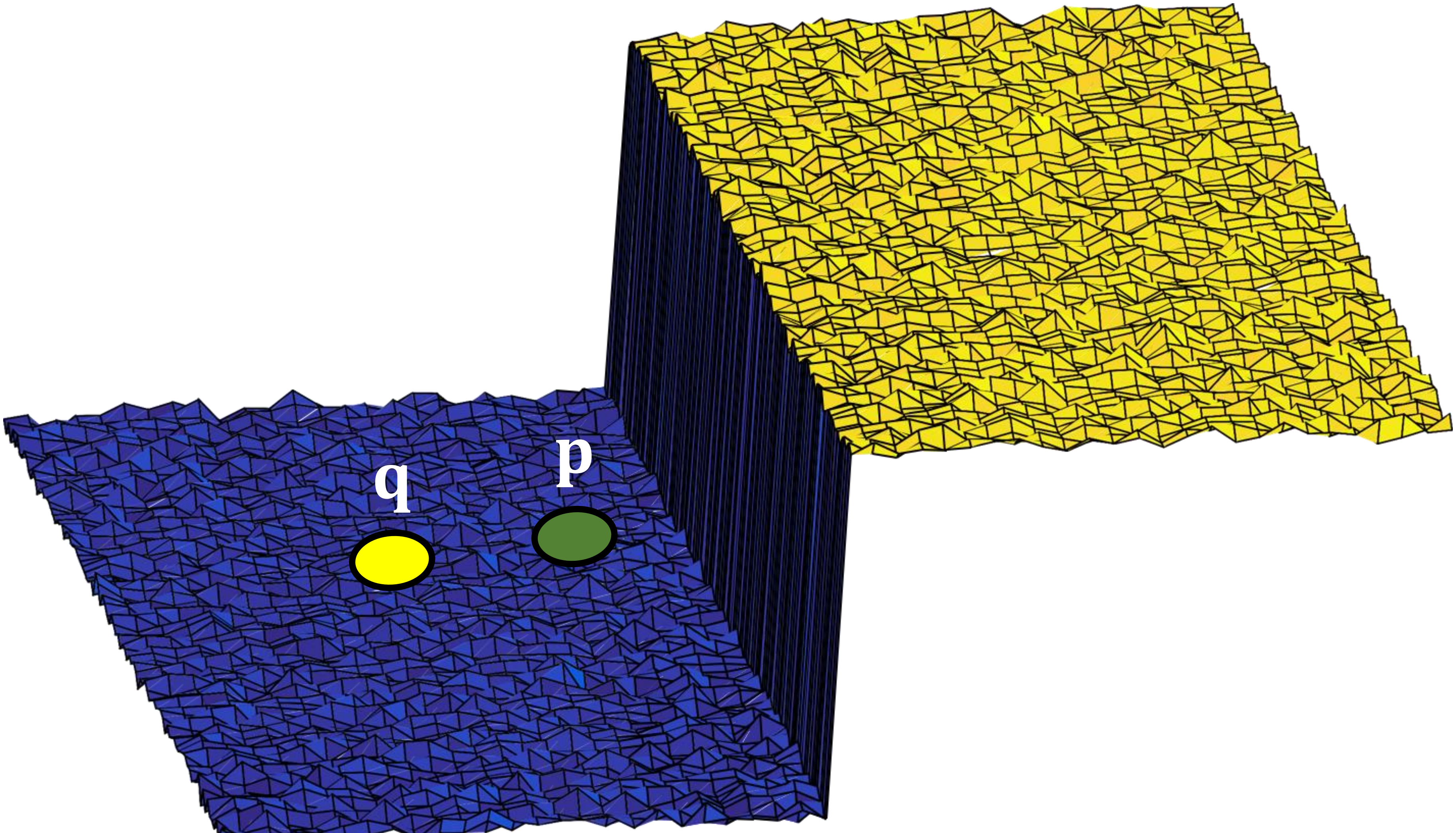
IEEE International Conference on Computer Vision (ICCV) 1998


高斯平滑

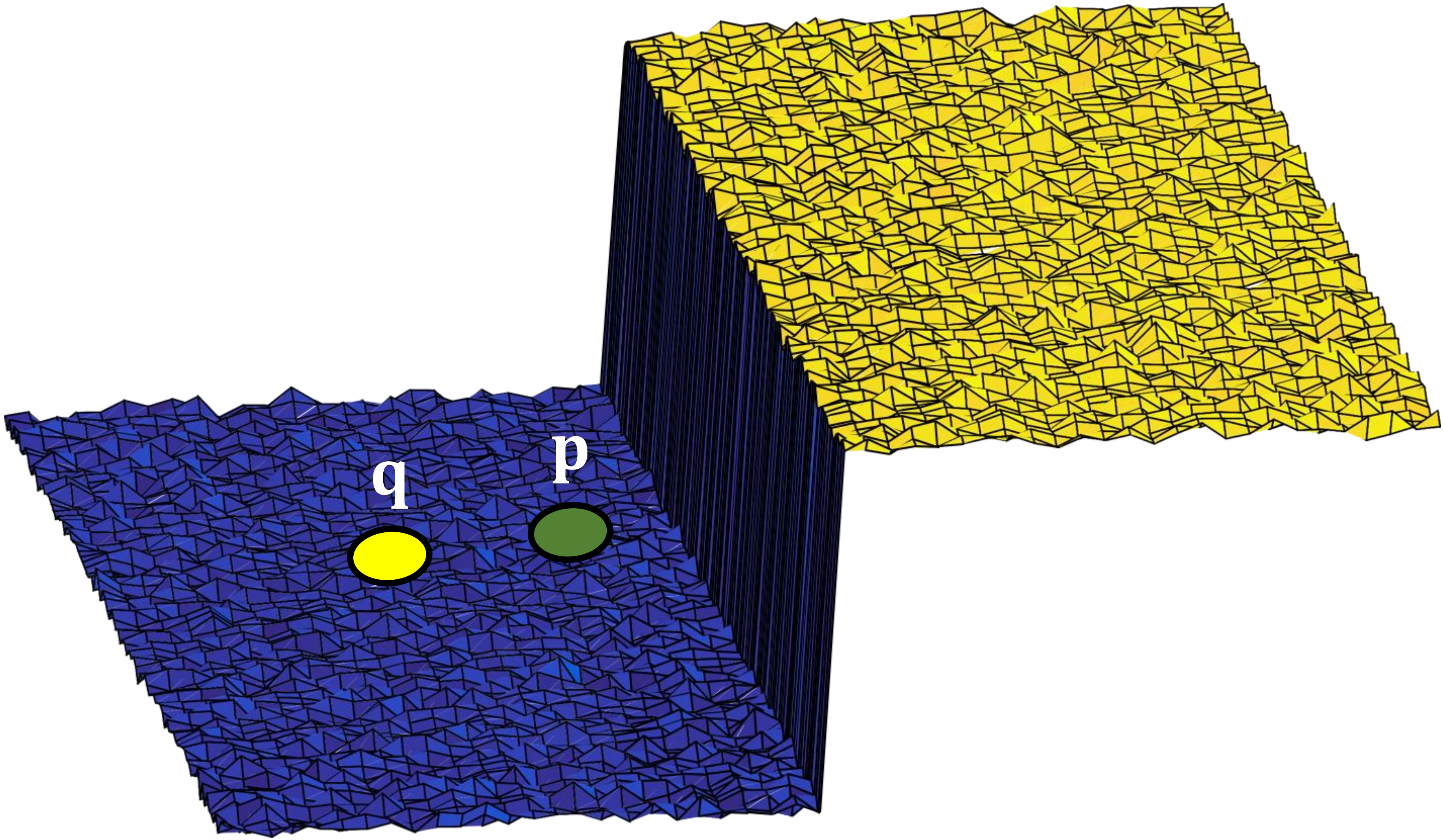
高斯平滑



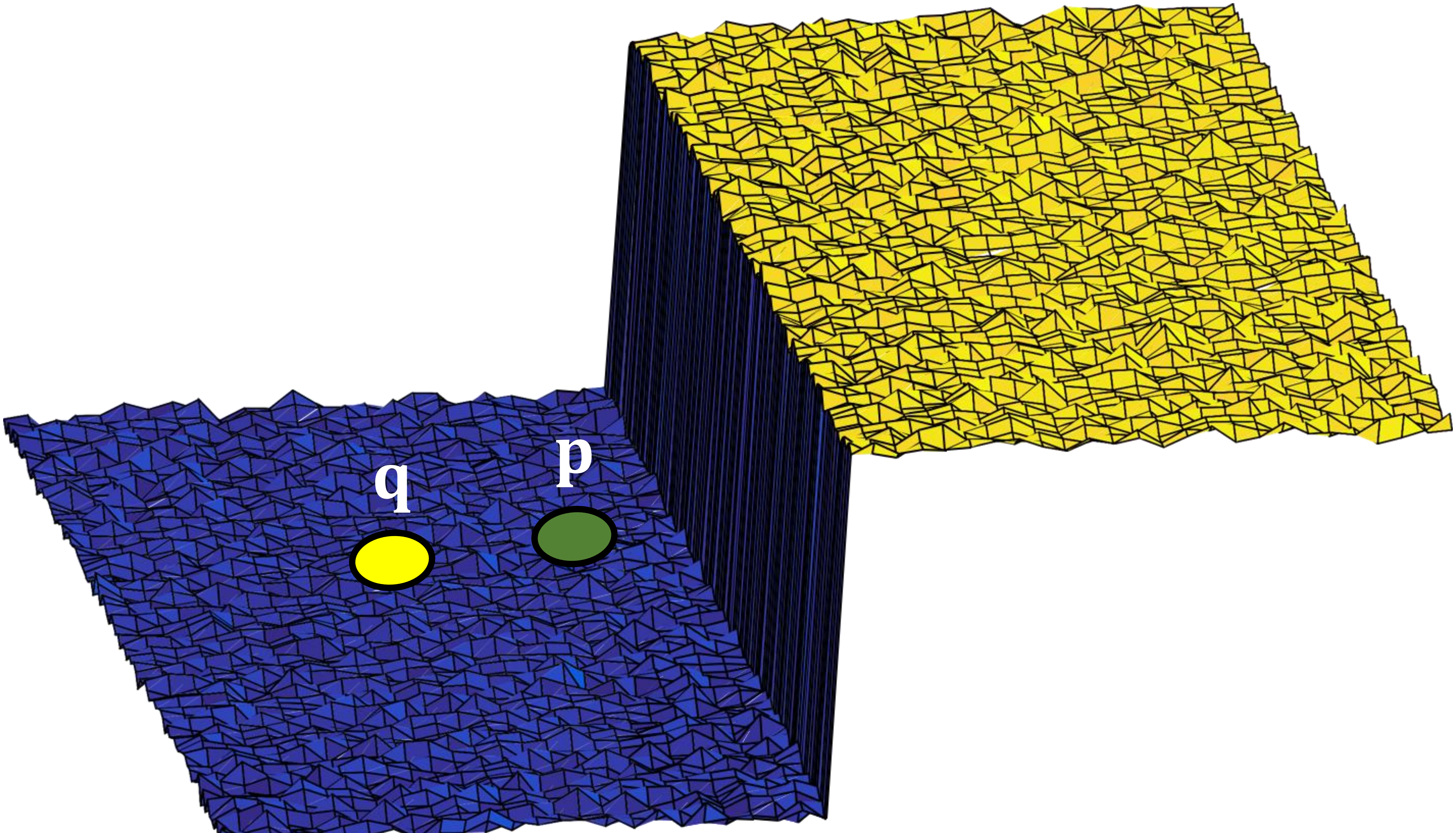

p


q
p

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

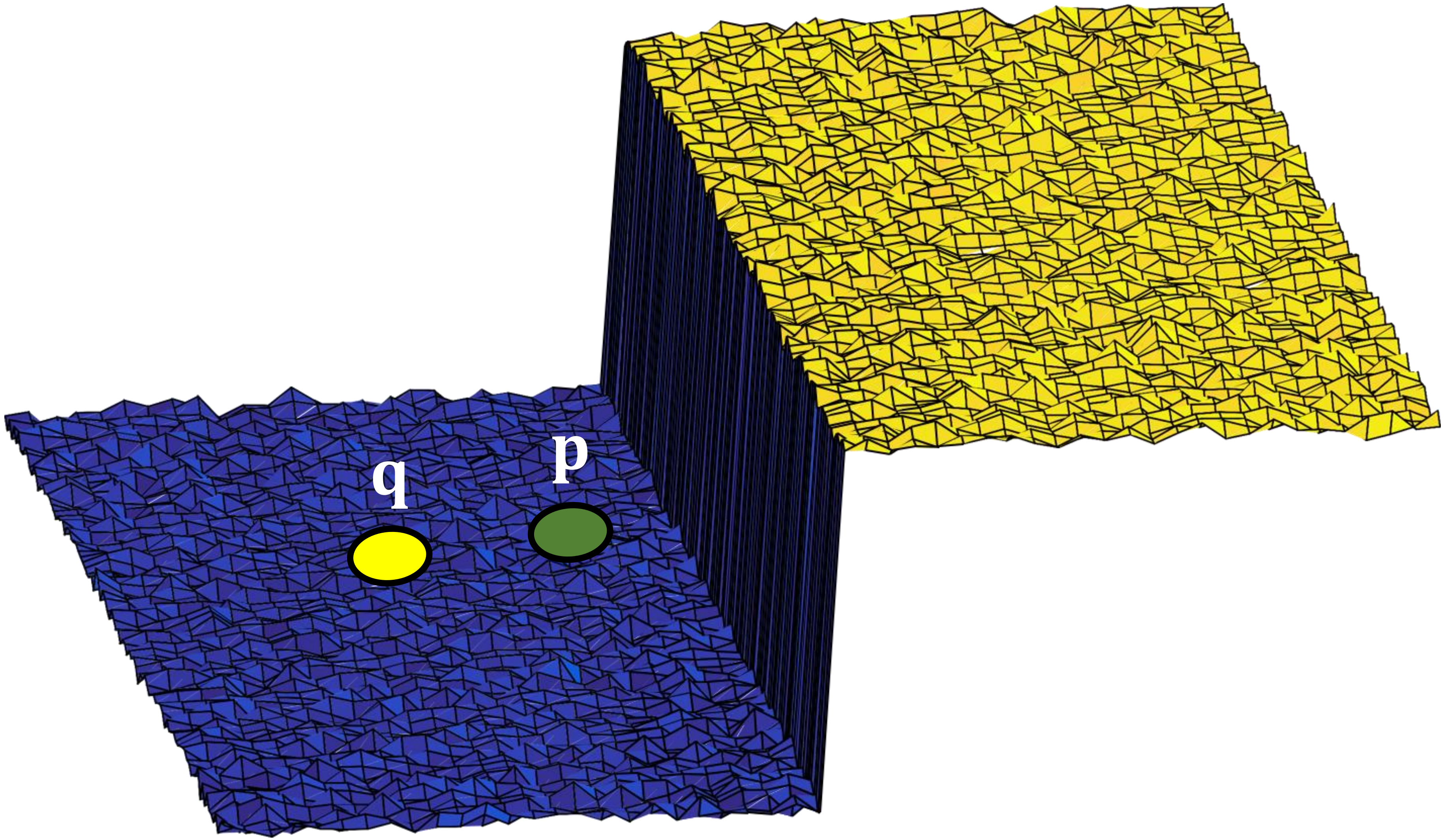


$$I'(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) I(\mathbf{q})$$



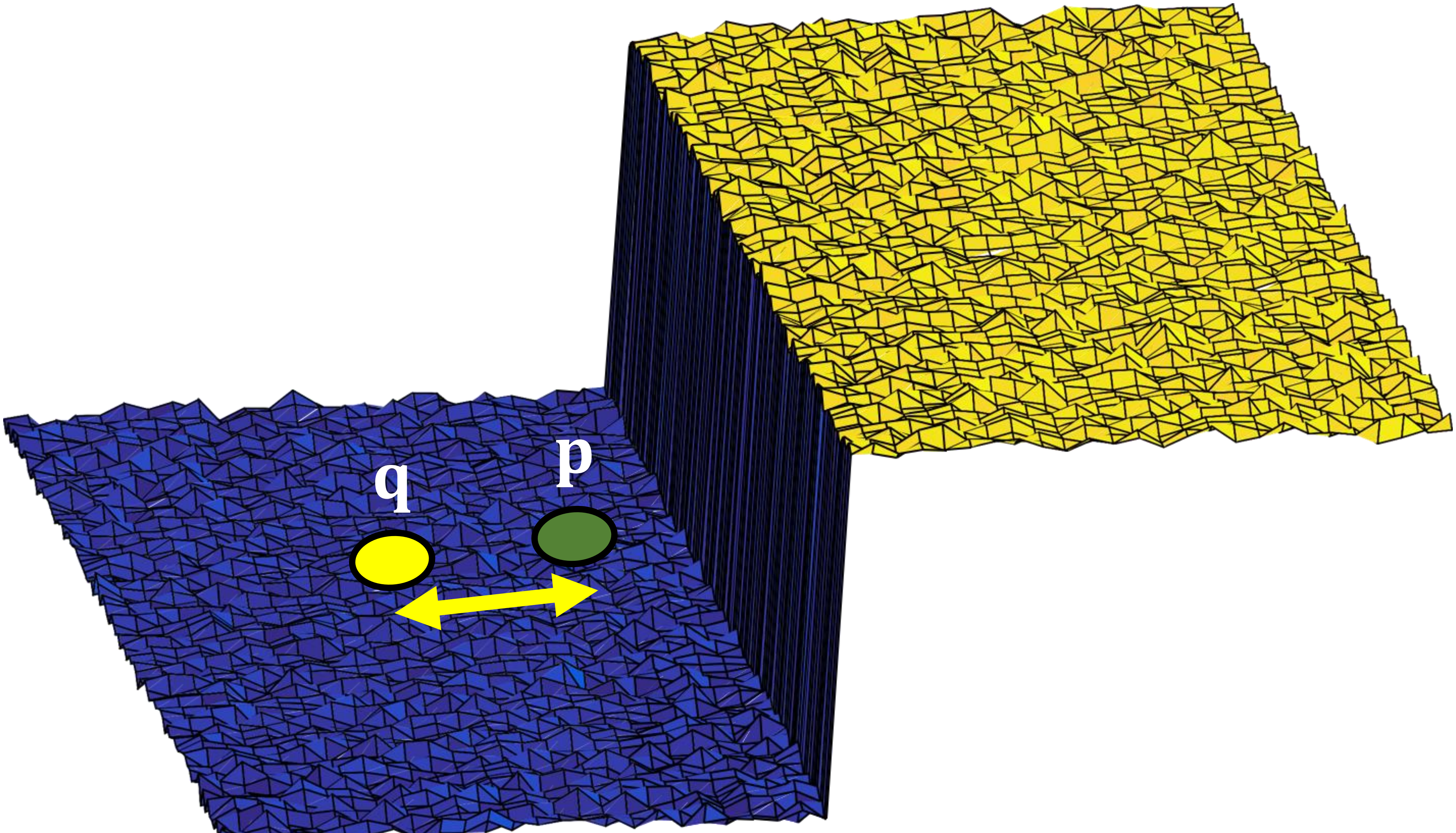
空间域权重

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



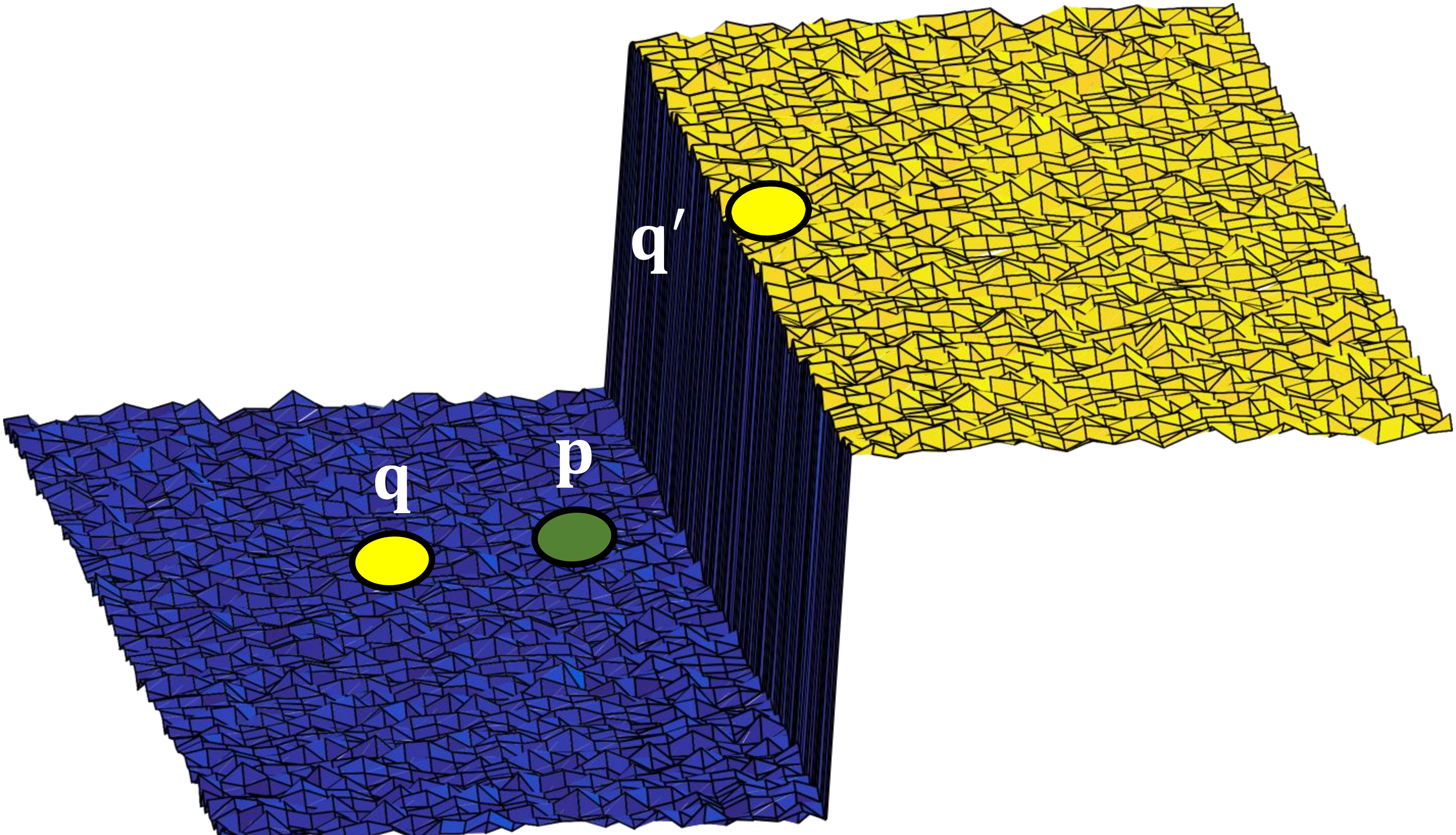
空间域权重

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



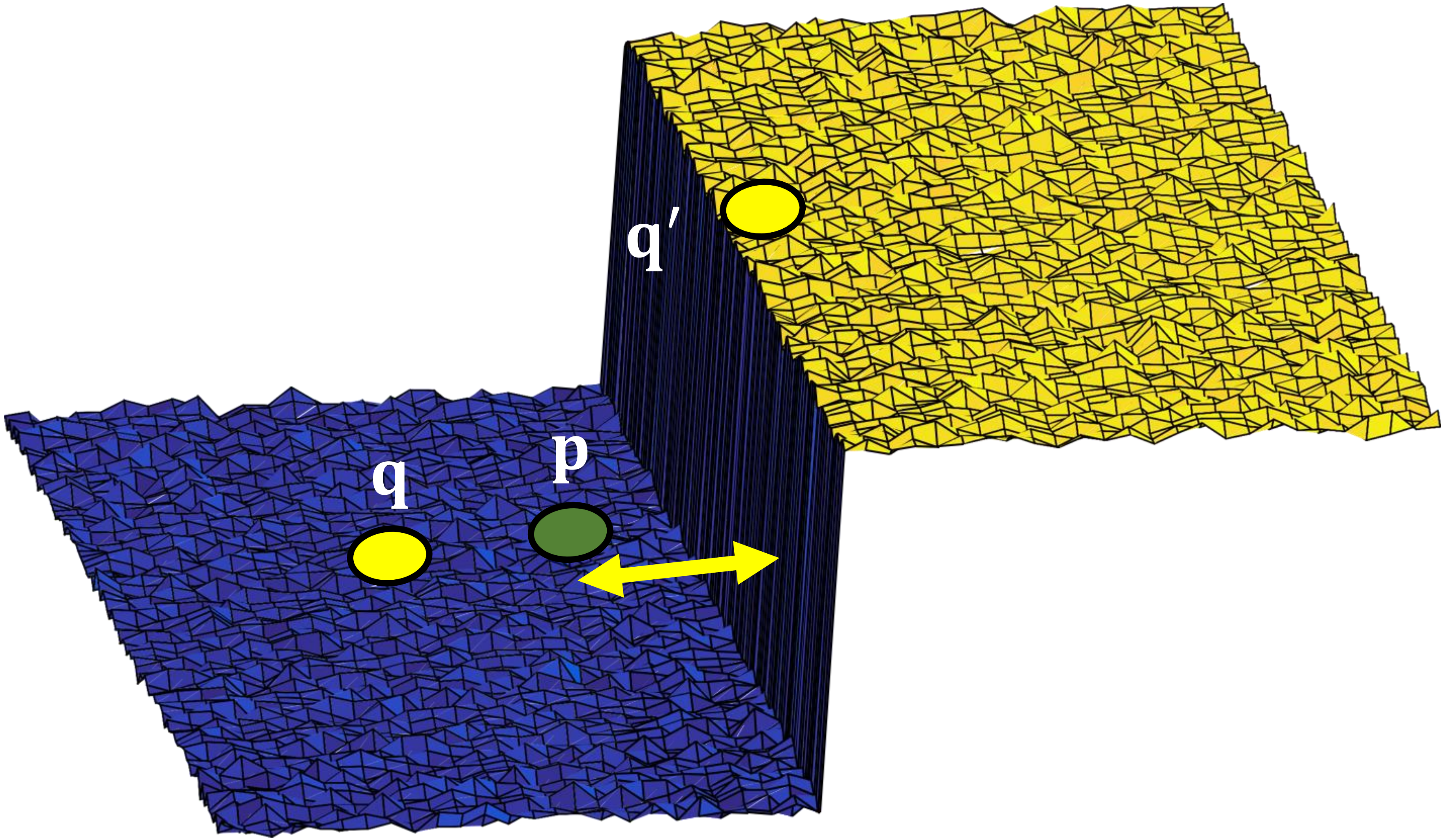
空间域权重

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

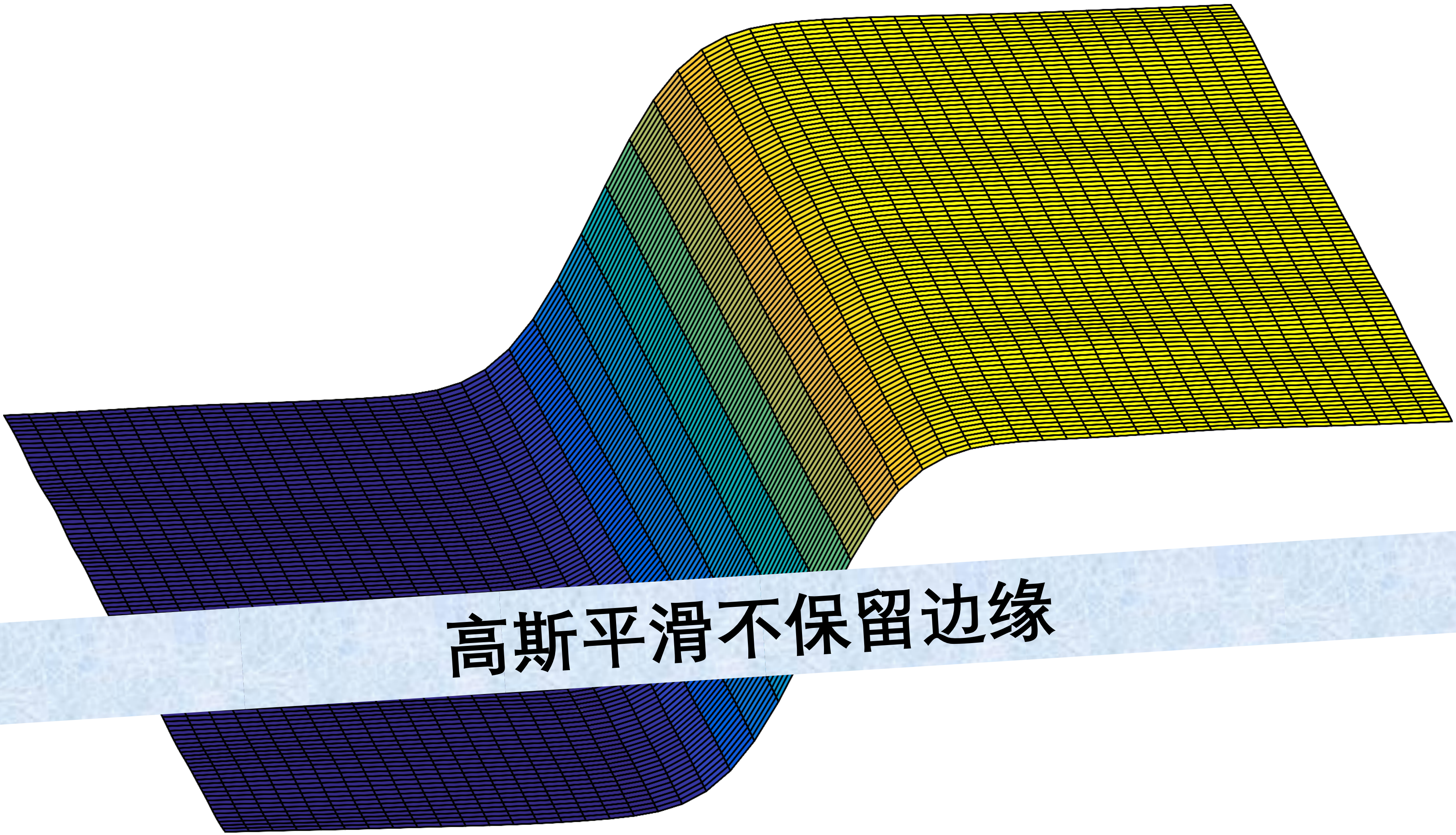


空间域权重

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

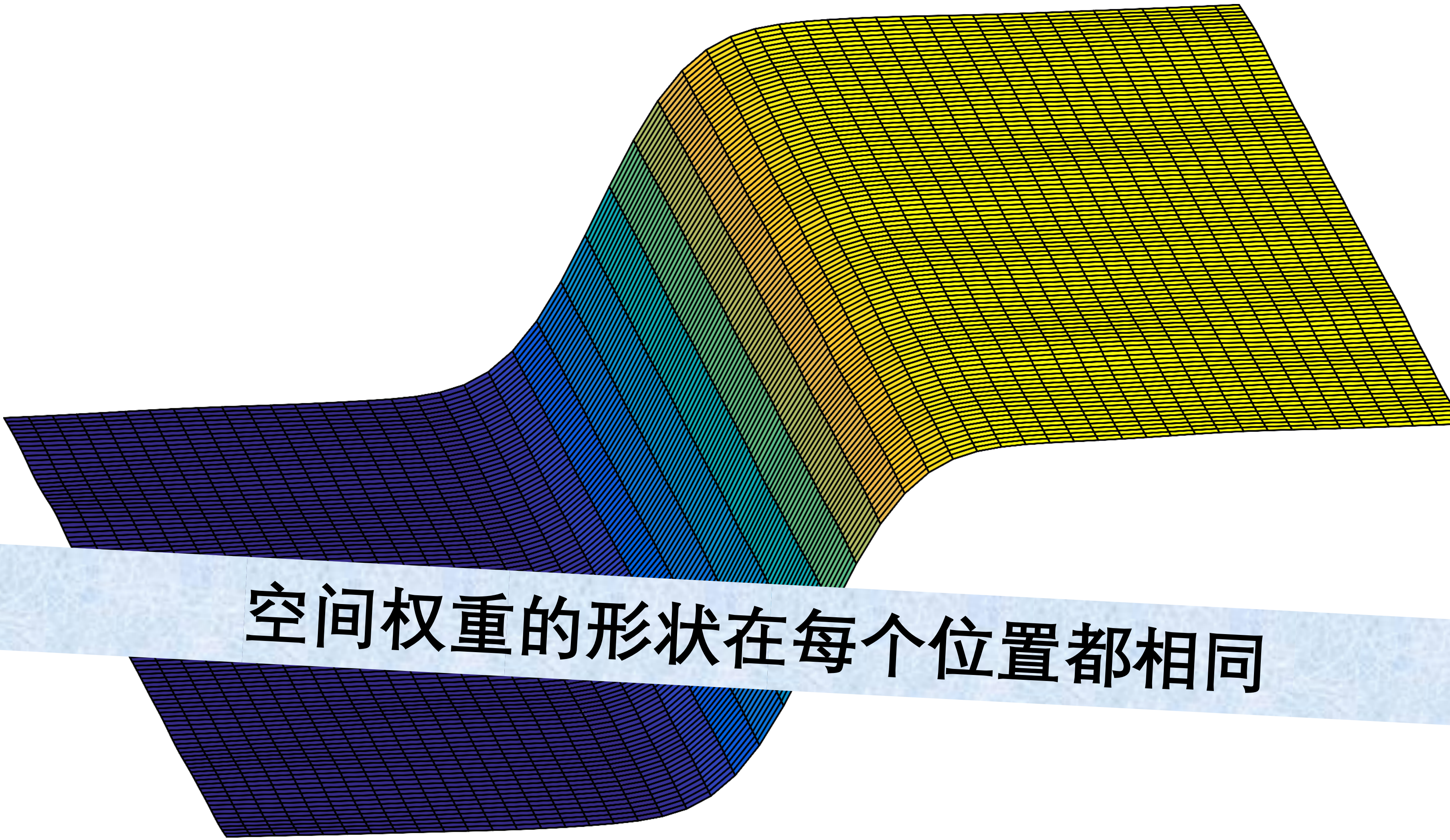


$$I'(\mathbf{p}) = \sum_{\mathbf{q}\in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)I(\mathbf{q})$$



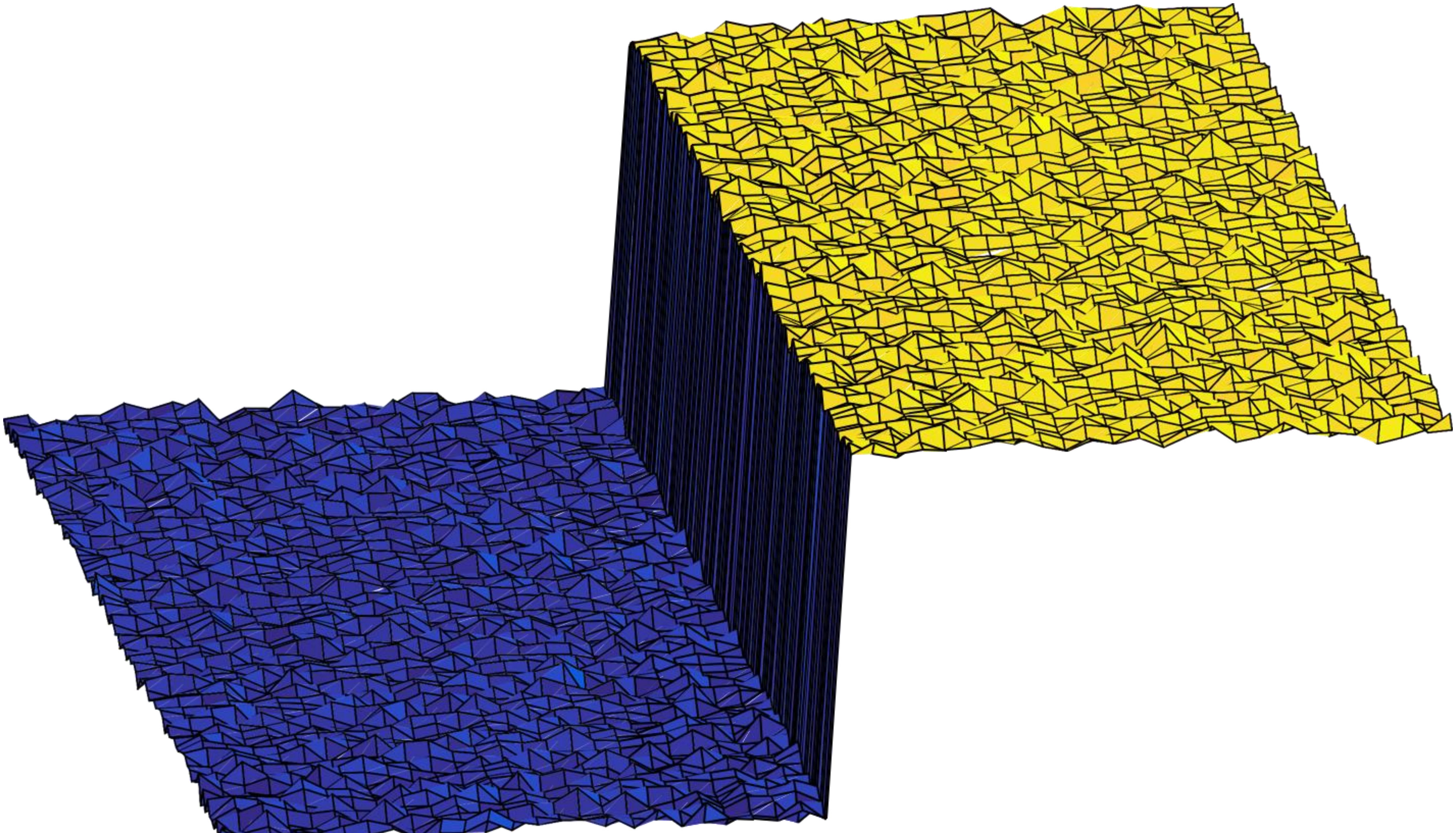
高斯平滑不保留边缘

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

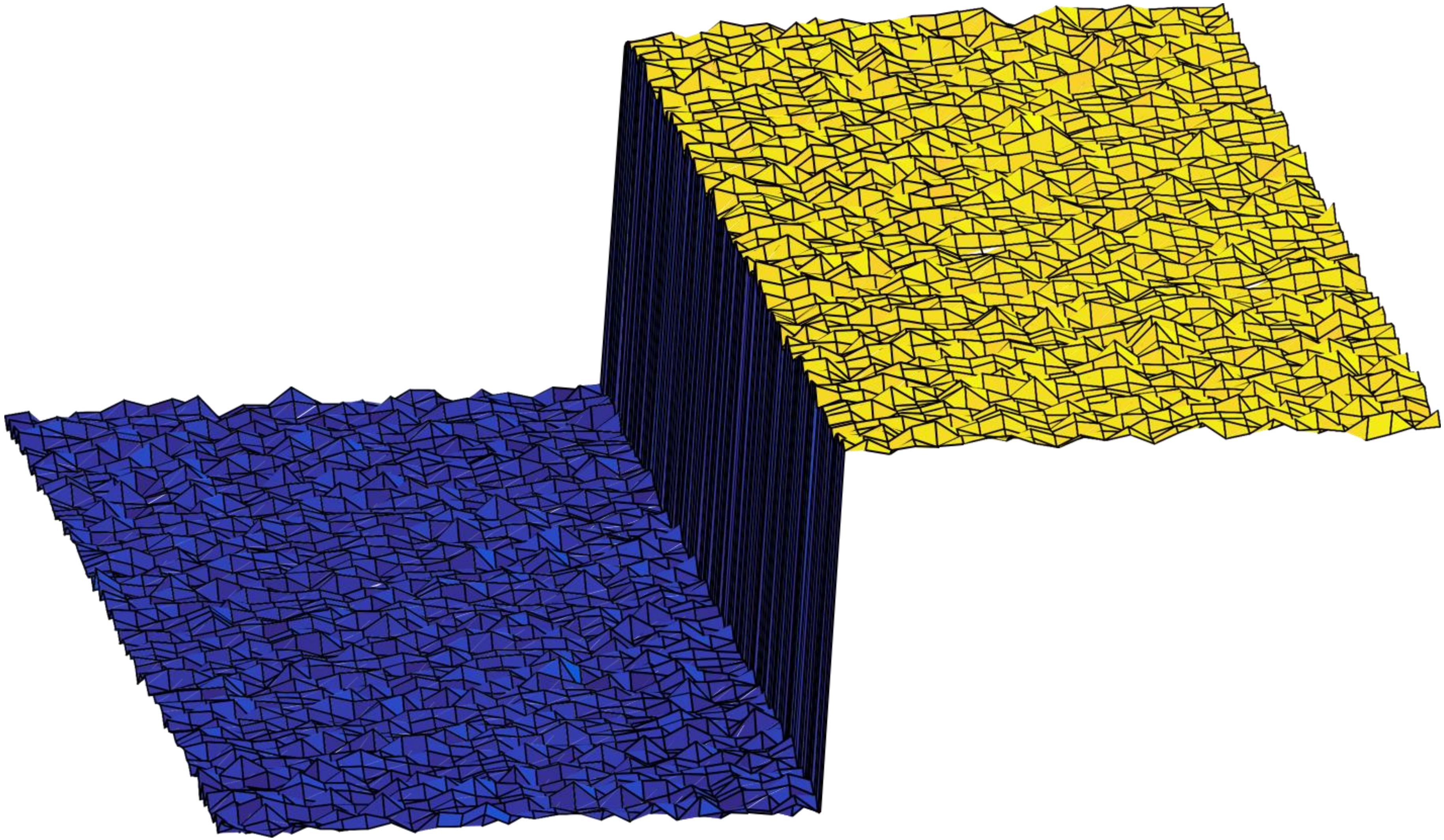


空间权重的形状在每个位置都相同

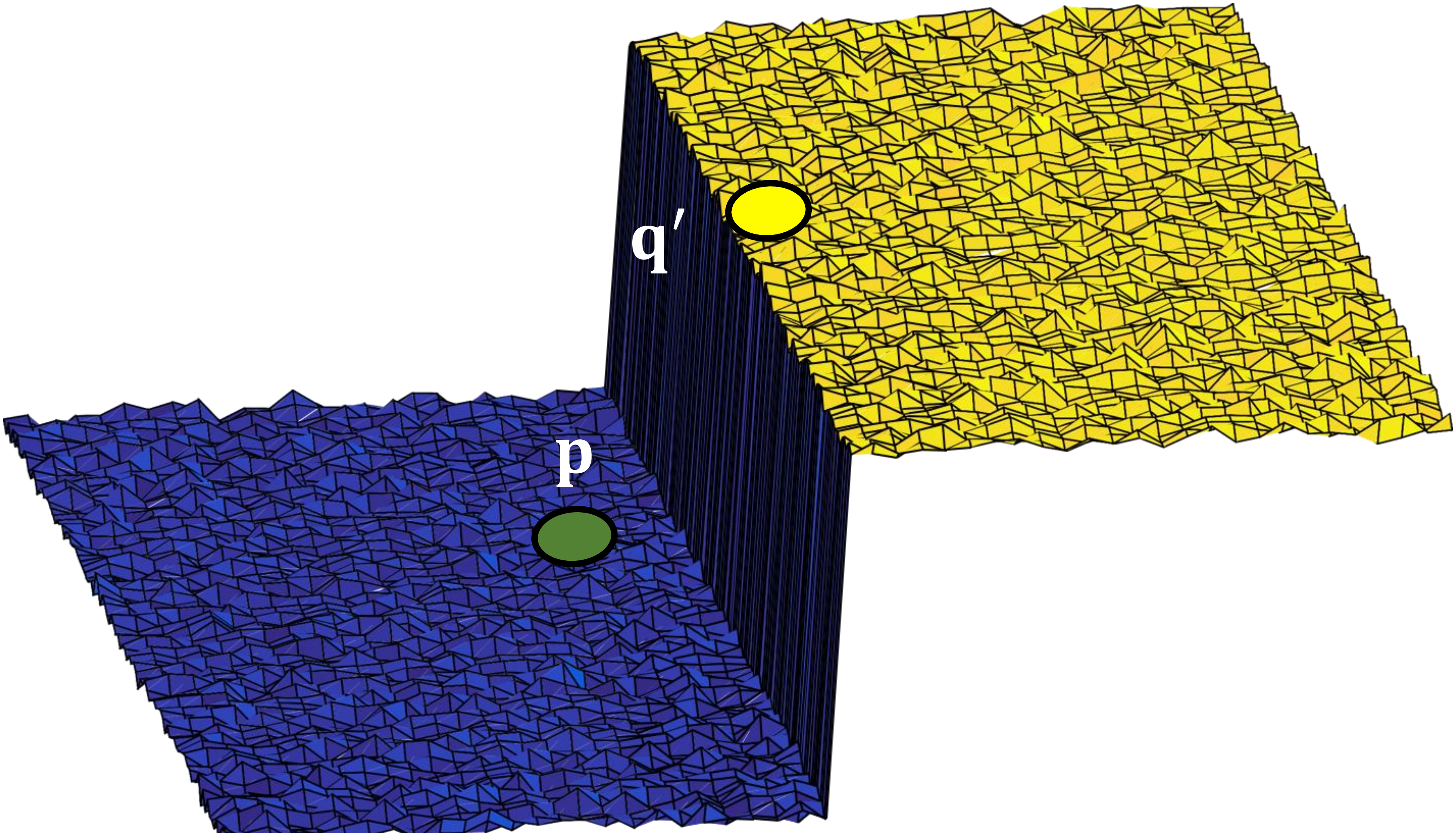
$$I'(\mathbf{p}) = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) I(\mathbf{q})$$



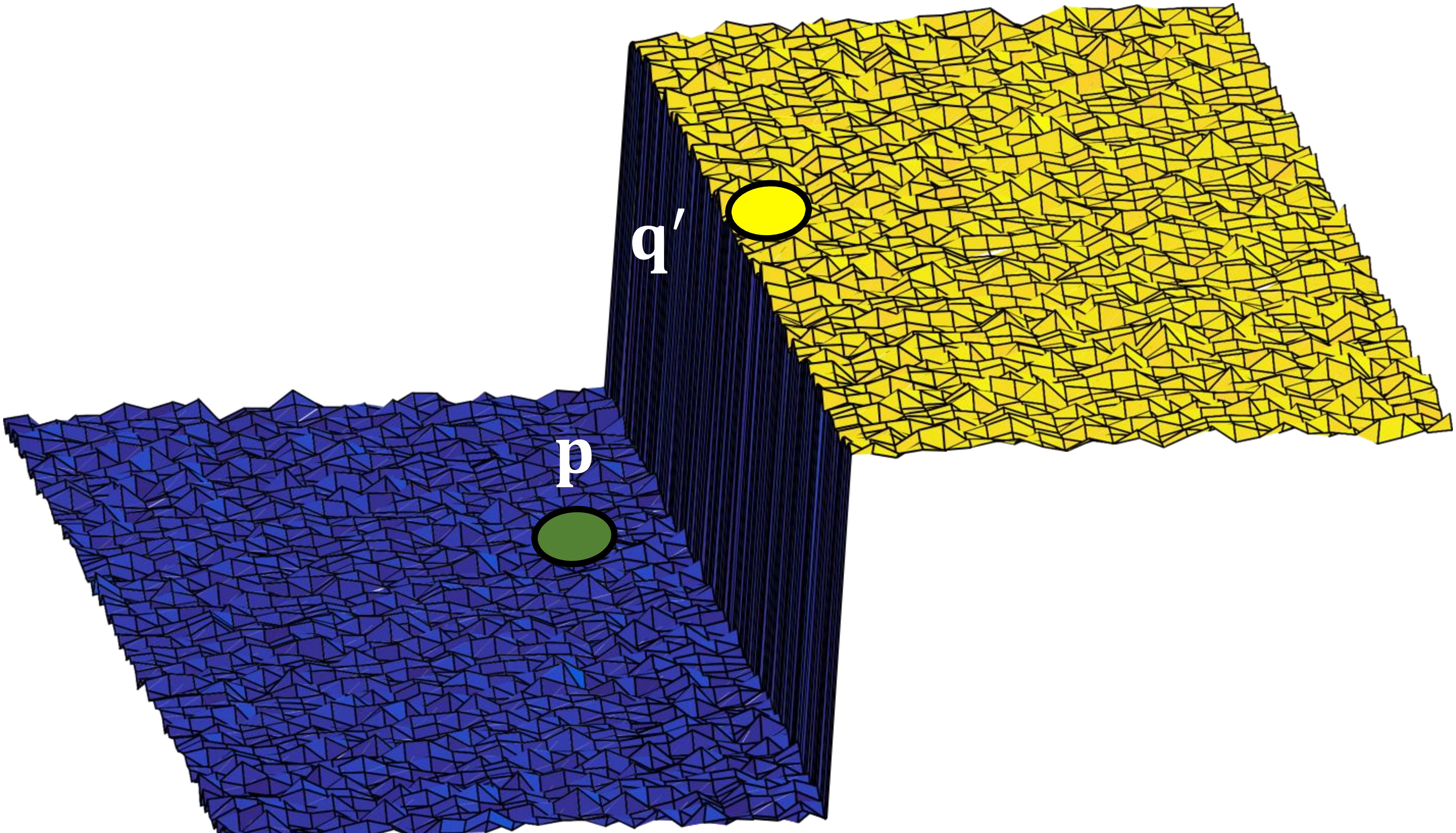
$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$



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

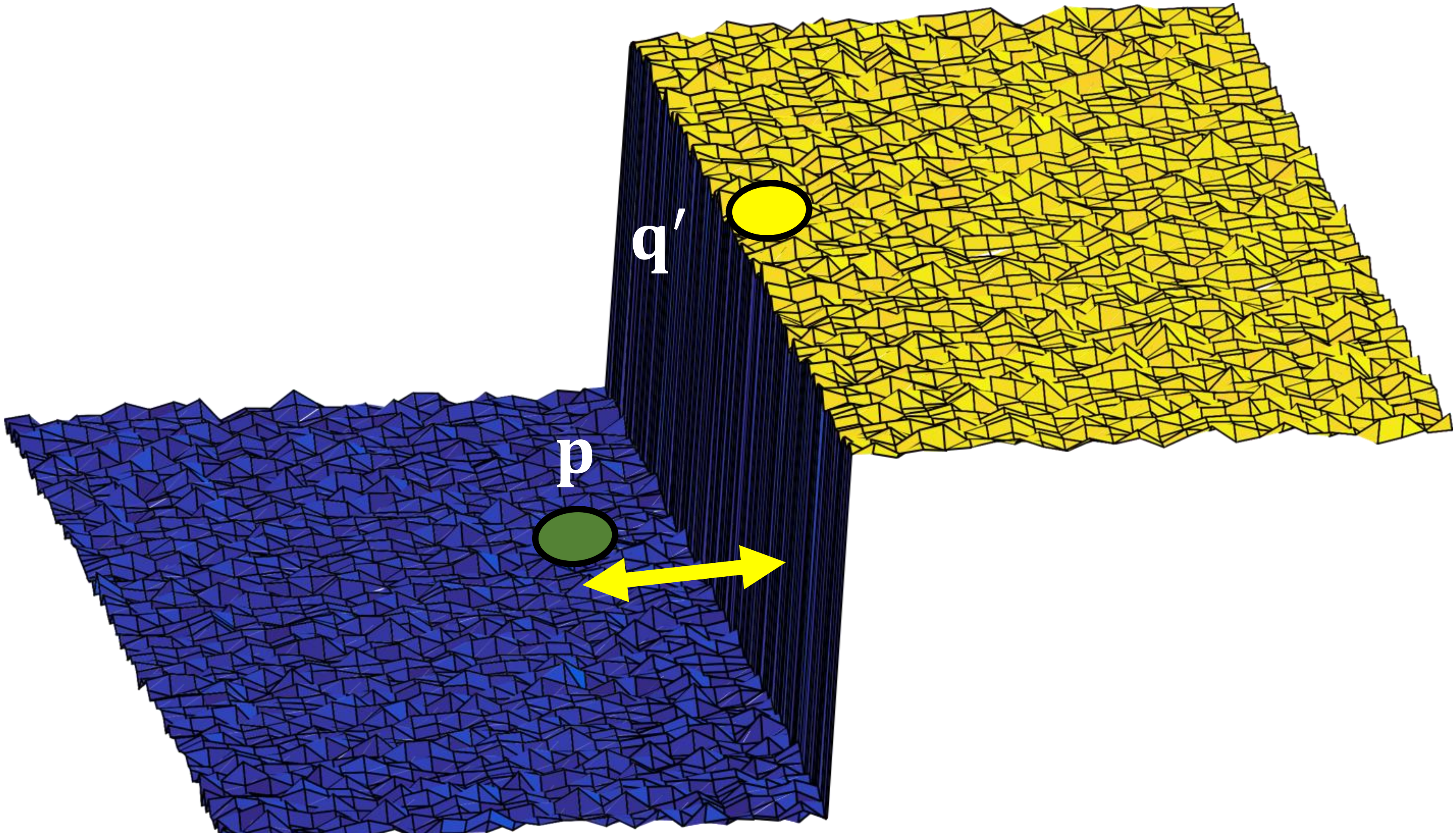


$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$



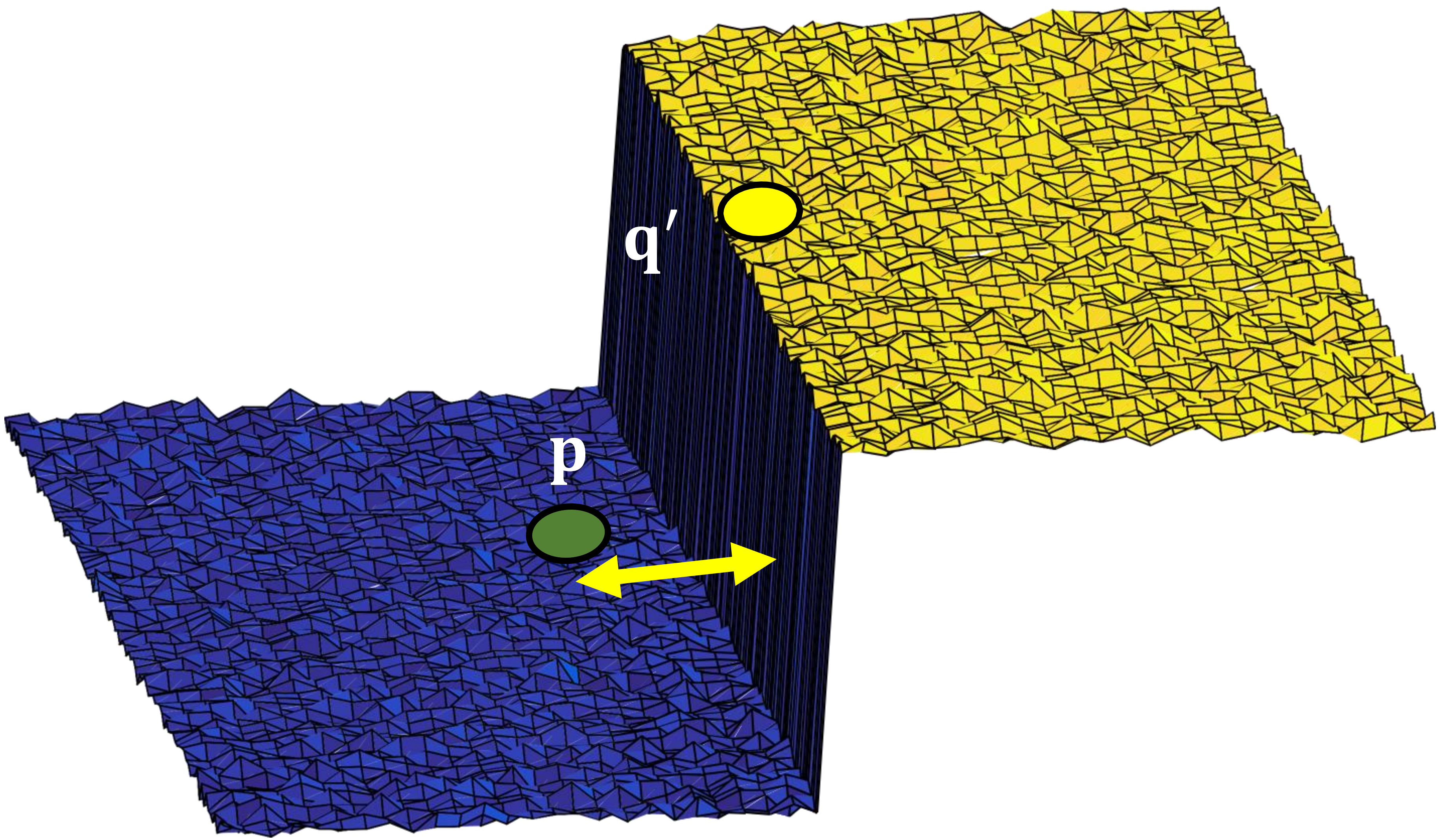
空间域权重

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



空间域权重

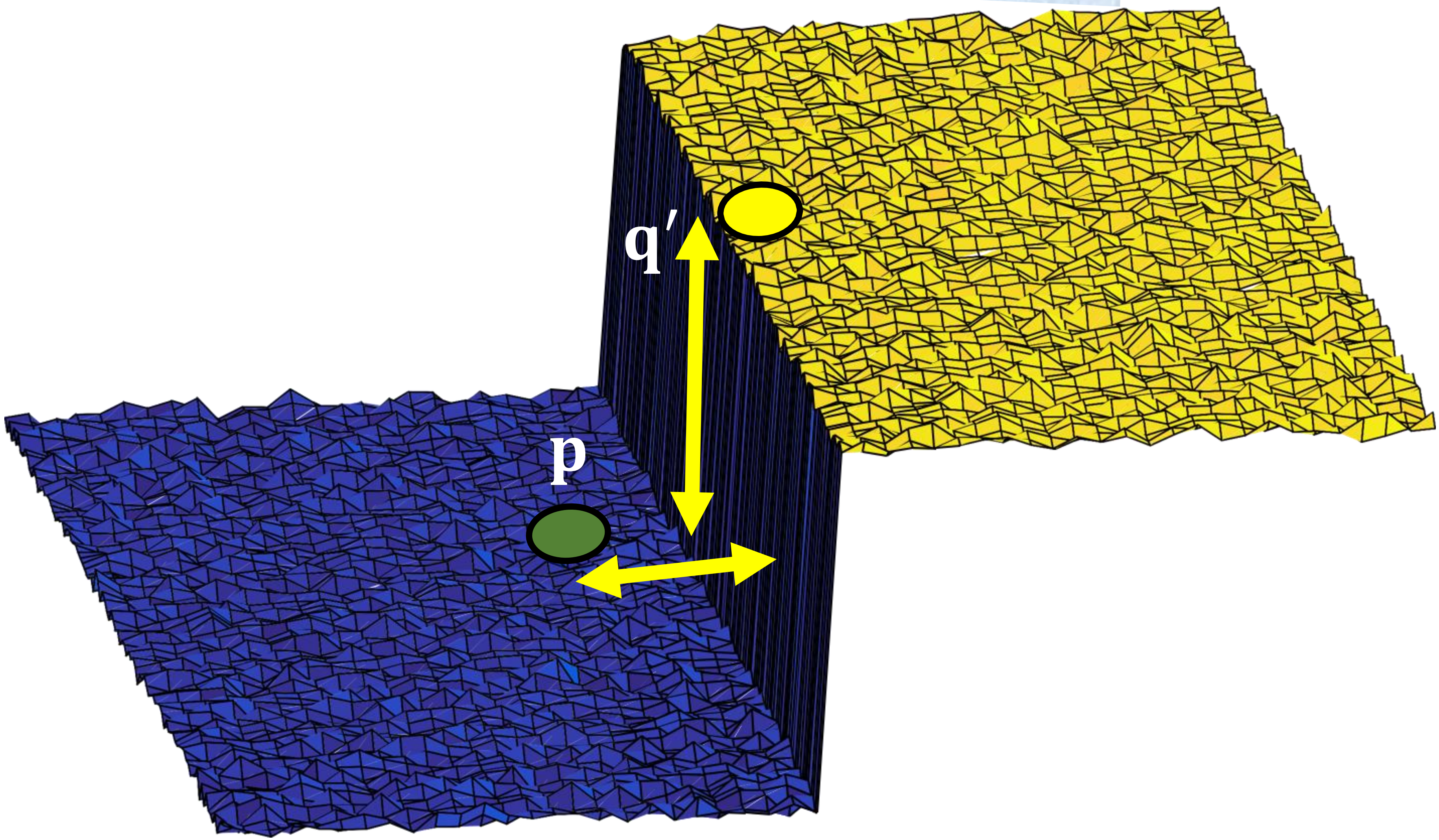
$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q}\in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)G_{\sigma_r}(\|I(\mathbf{p})-I(\mathbf{q})\|)I(\mathbf{q})$$



空间域权重

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

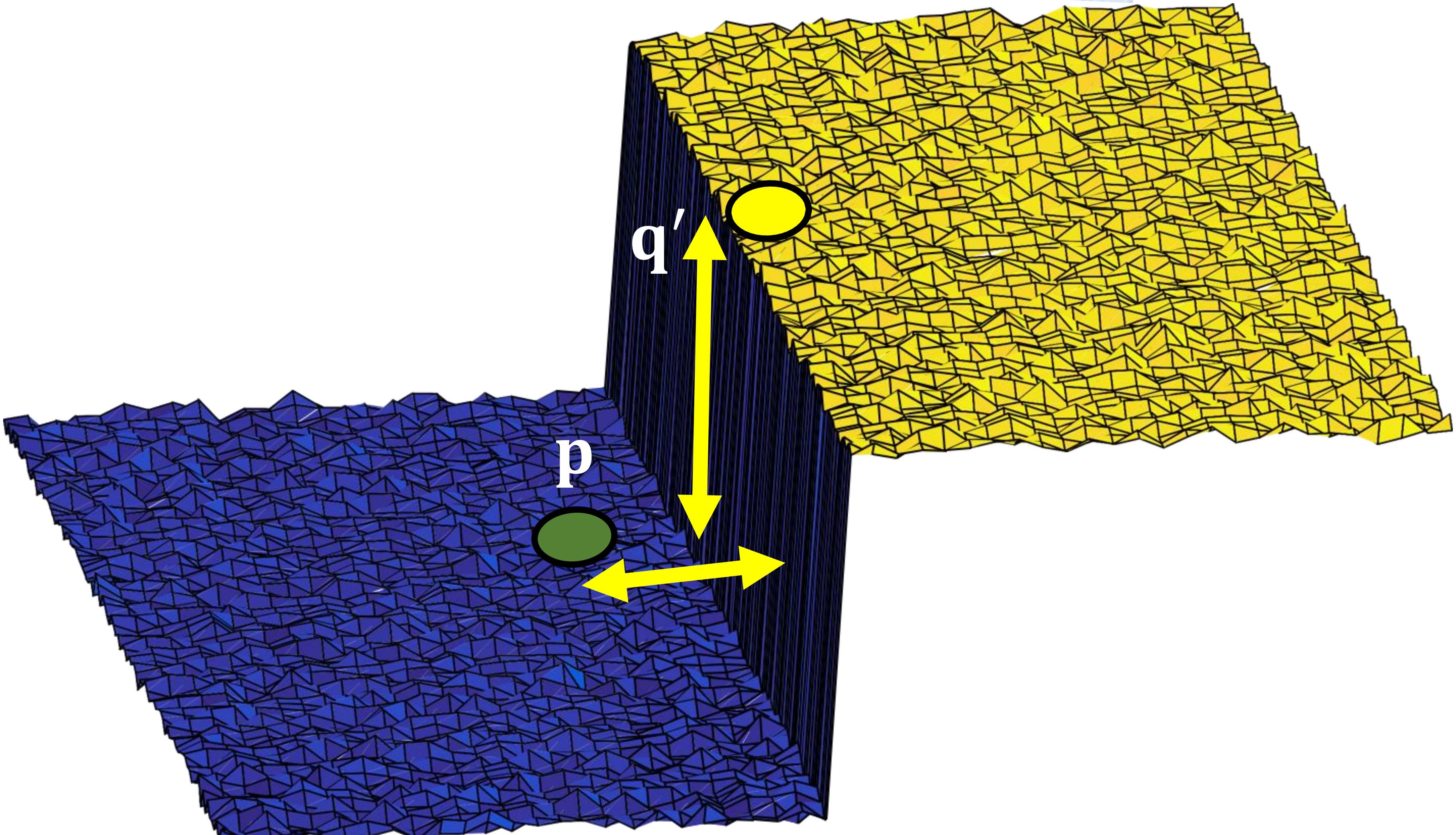
值域权重



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

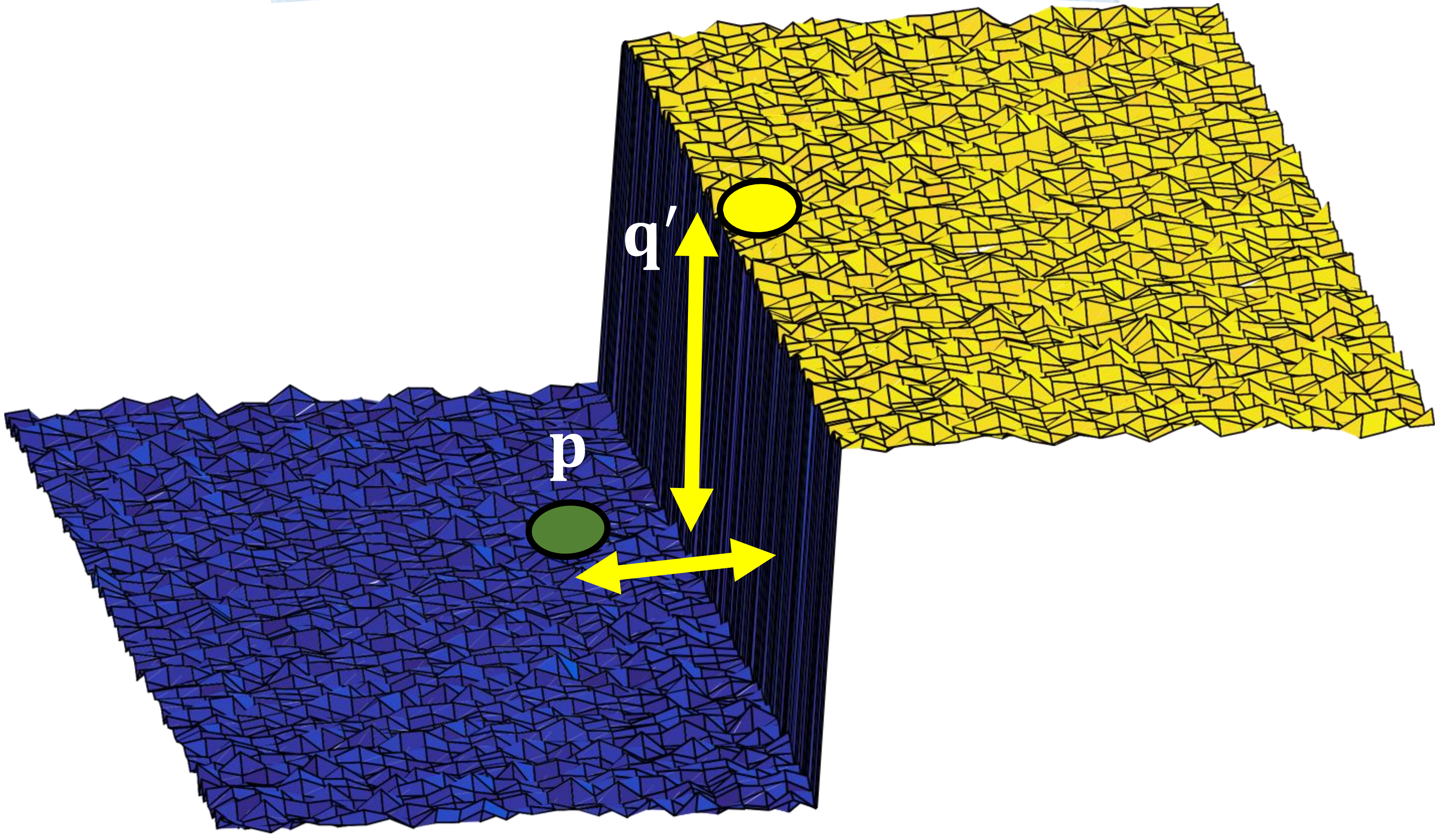


空间域权重

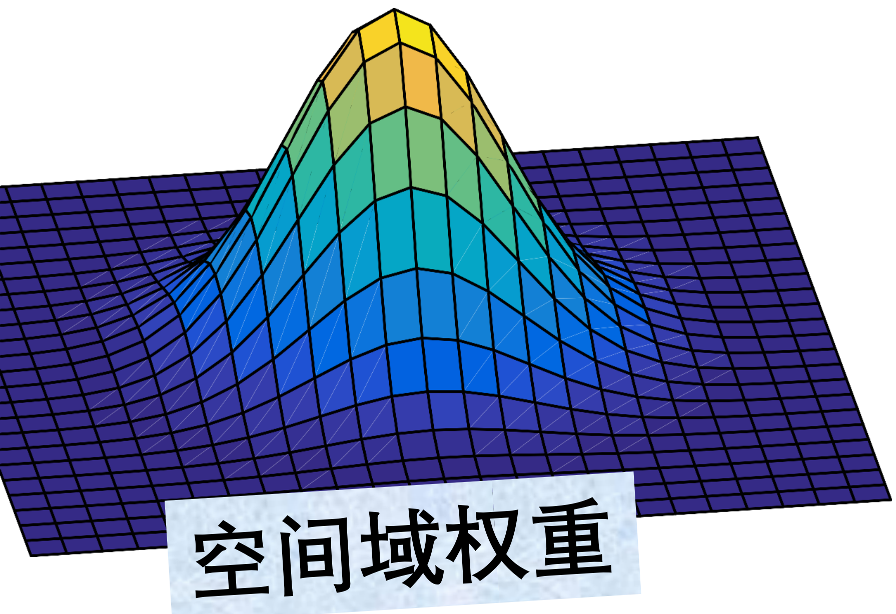
$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q}\in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}(\|I(\mathbf{p})-I(\mathbf{q})\|)\, I(\mathbf{q})$$

归一化

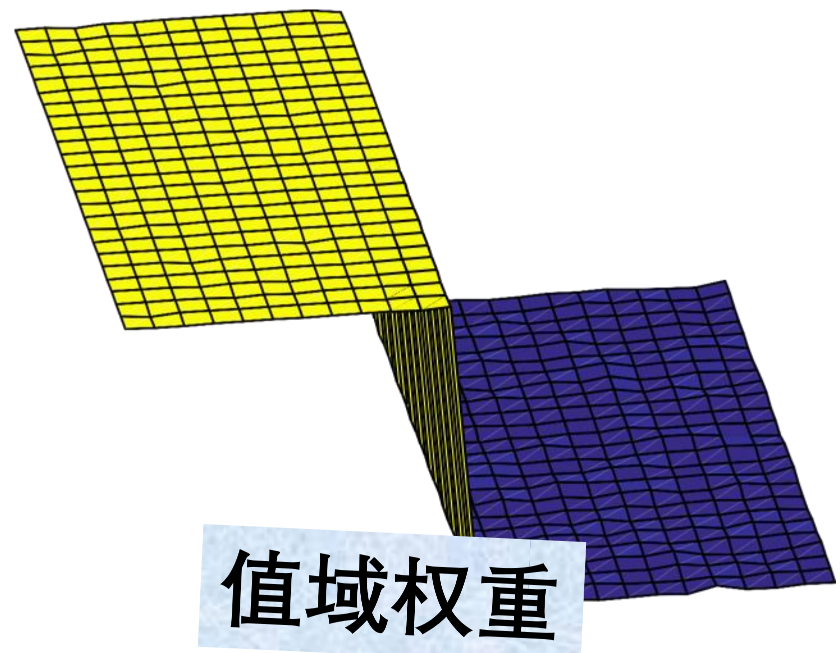
值域权重



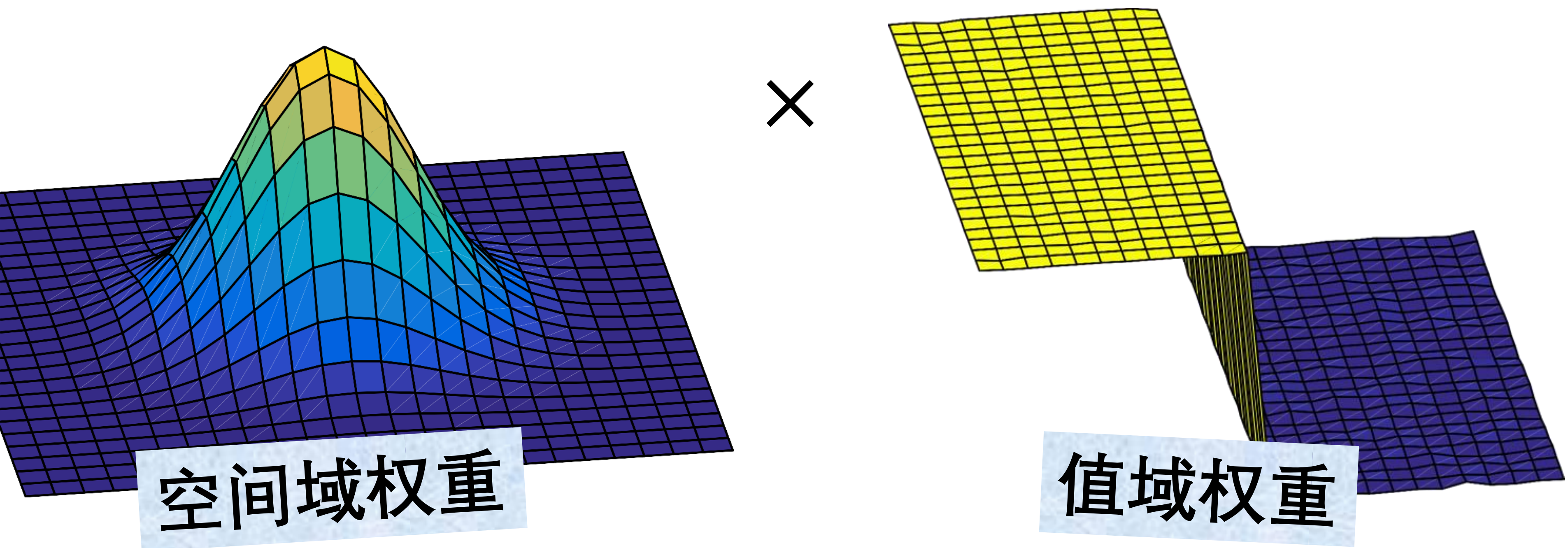
$G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)$



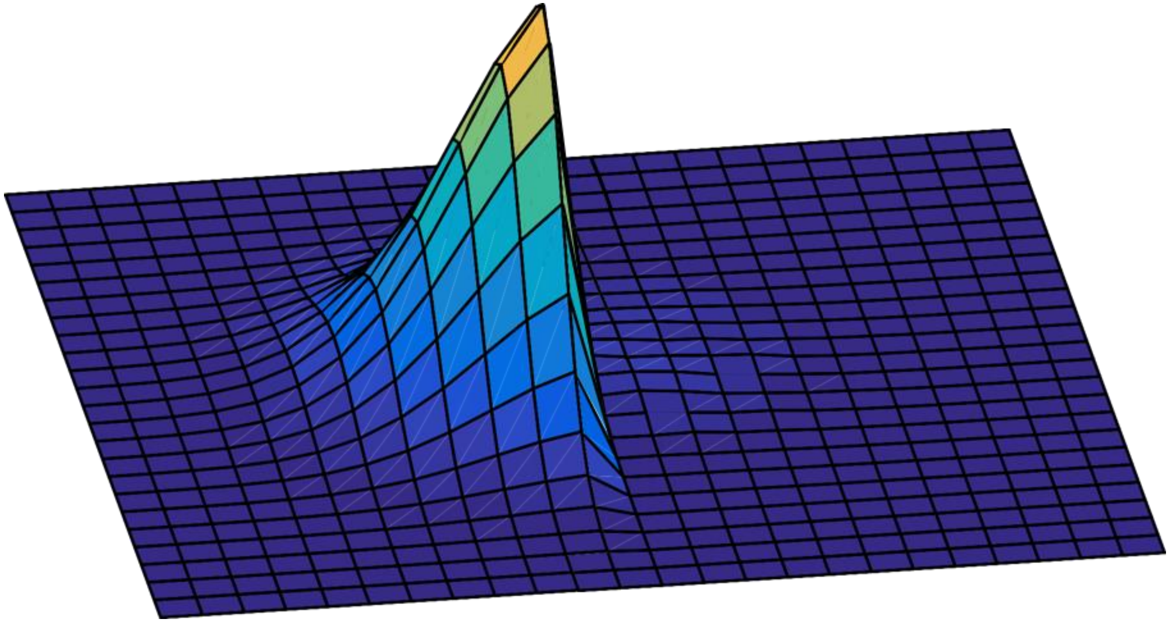
$G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|)$



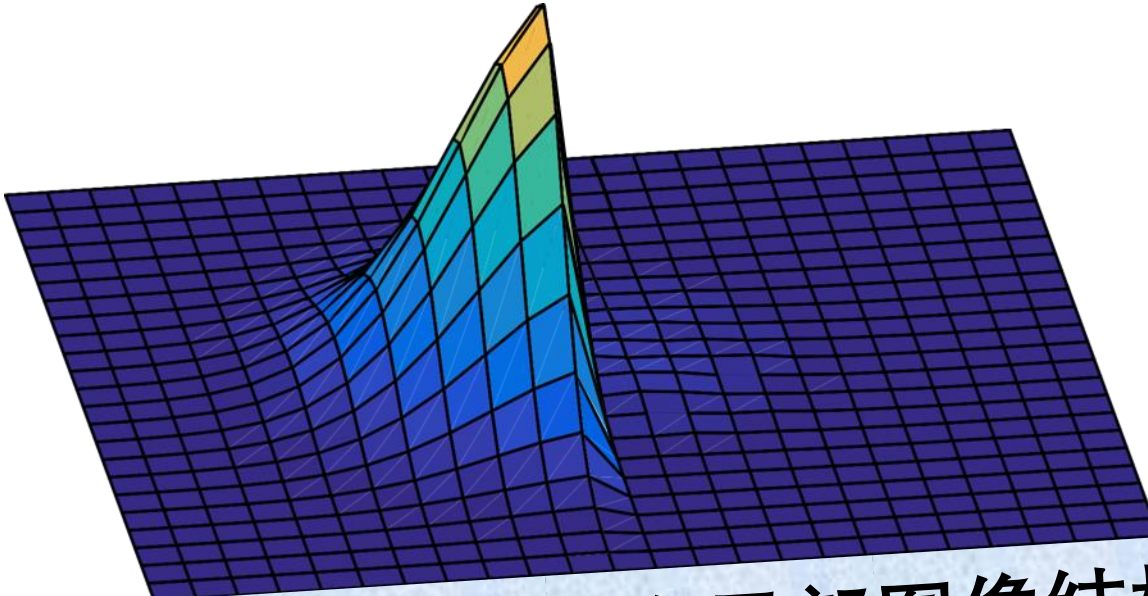
$$G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|) \times G_{\sigma_r}(\|I(\mathbf{p})-I(\mathbf{q})\|)$$



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

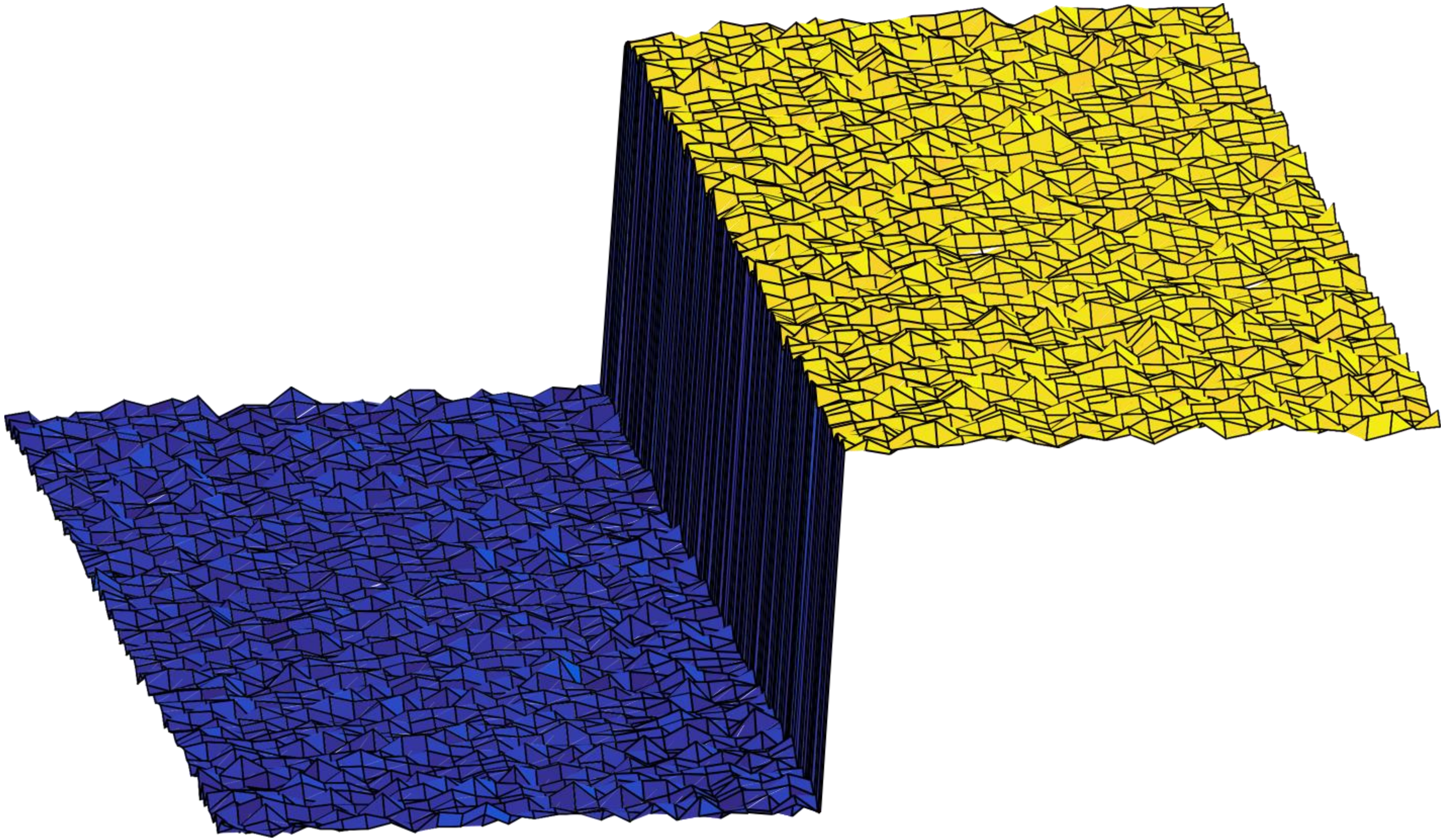


$$w(\mathbf{p})G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)G_{\sigma_r}(\|I(\mathbf{p})-I(\mathbf{q})\|)$$

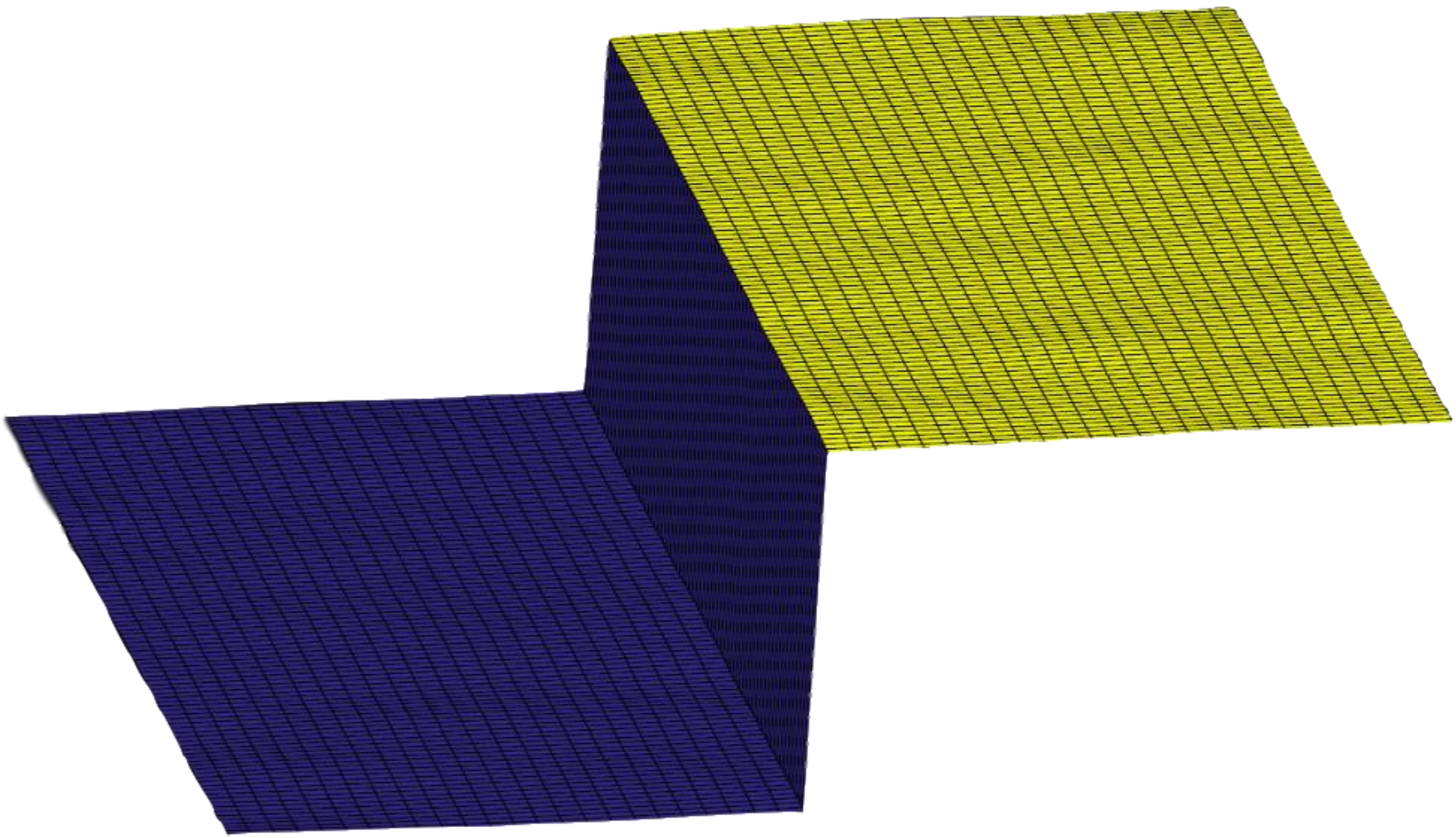


权重滤波器的形状符合局部图像结构

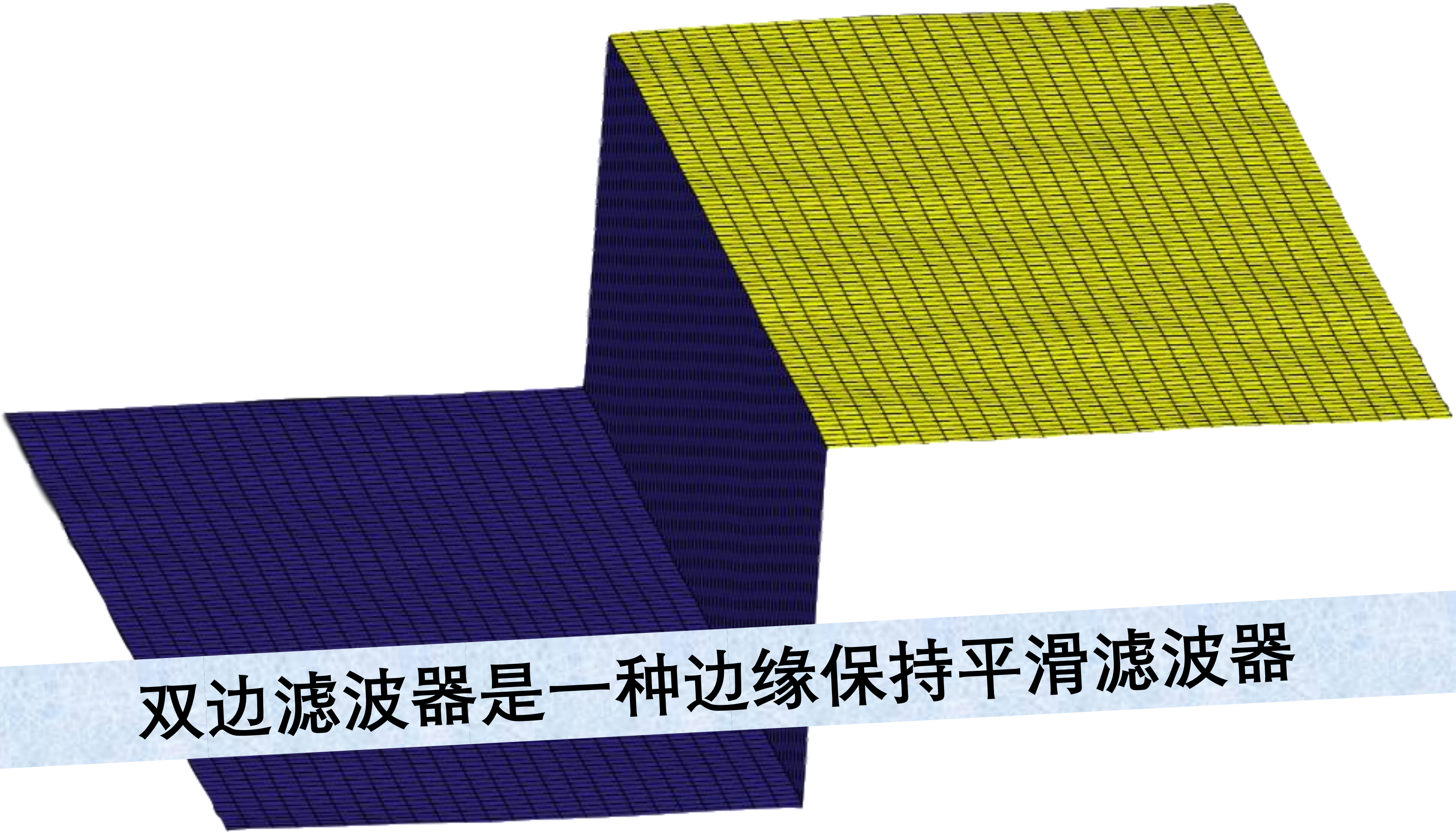
$$I'(\mathbf{p}) = w(\mathbf{p}) \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I(\mathbf{p}) - I(\mathbf{q})\|) I(\mathbf{q})$$



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



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



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