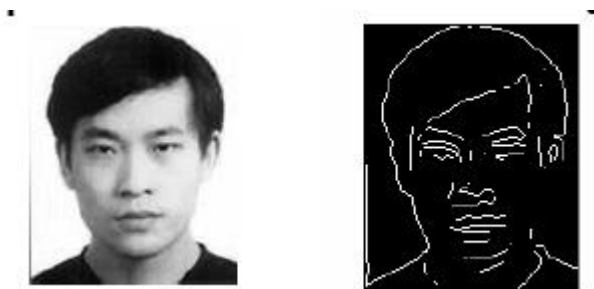


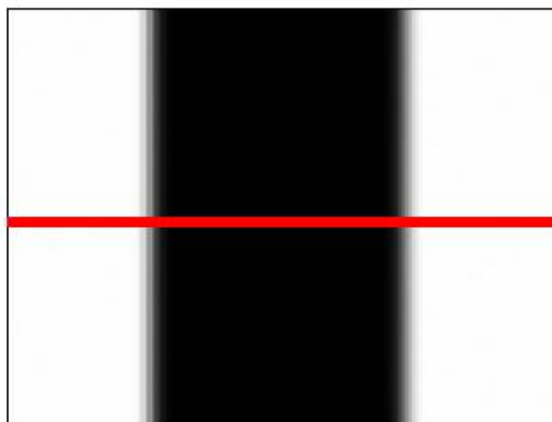
# 边缘提取

- 边缘定义
  - 图像中亮度变化明显的点

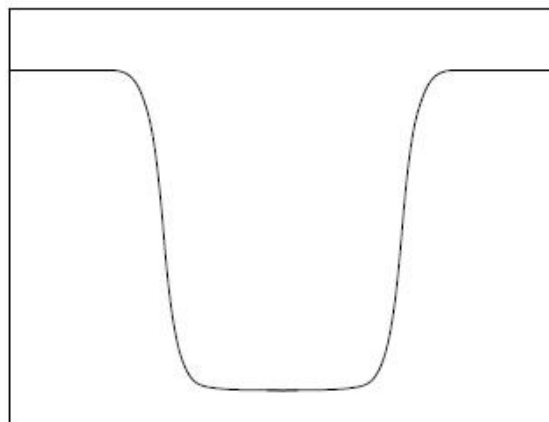


- 对于图像理解有重要作用
- 可减少数据量（素描、卡通）

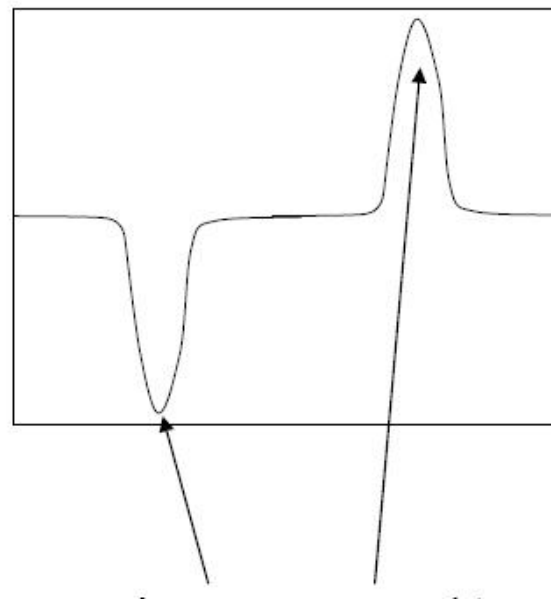
image

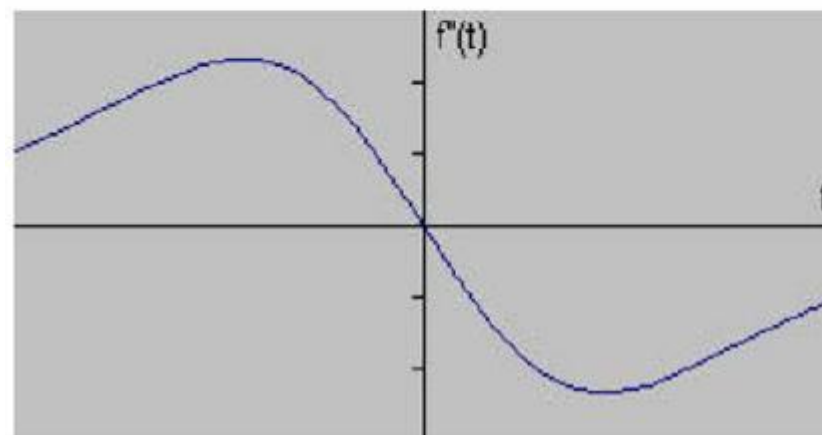
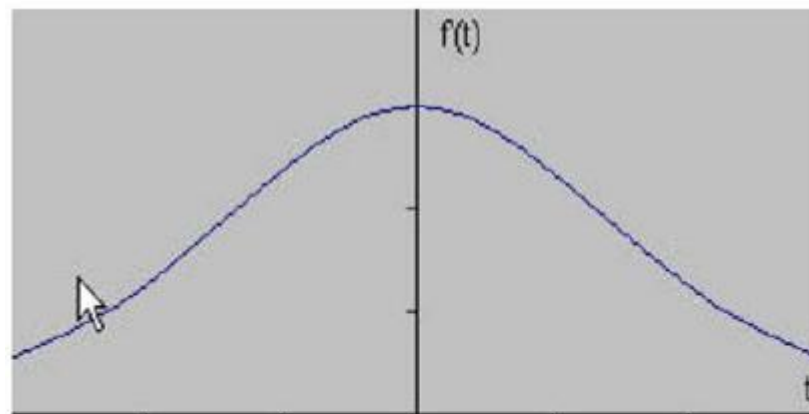
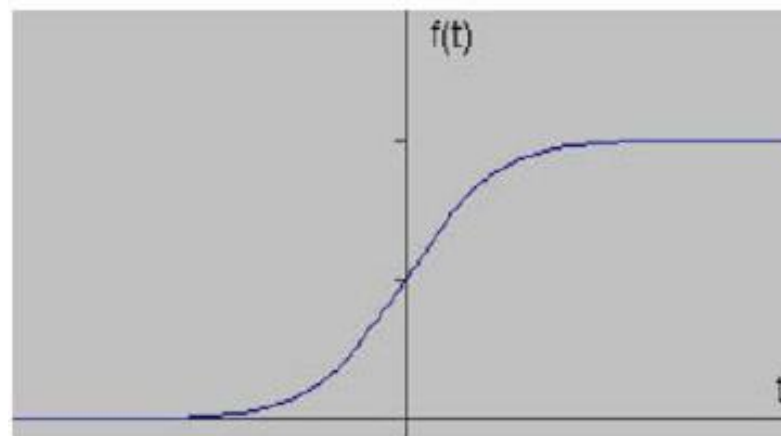


intensity function  
(along horizontal scanline)



first derivative





# 一阶导数

- |      |   |     |
|------|---|-----|
| -1/2 | 0 | 1/2 |
|------|---|-----|

    4    152    148    149

$$I'(x) = -1/2 \cdot I(x-1) + 0 \cdot I(x) + 1/2 \cdot I(x+1).$$

$$(I(x) - I(x-1) + I(x+1) - I(x)) / 2$$

- 0.5   -1.5   73    72      -1.5

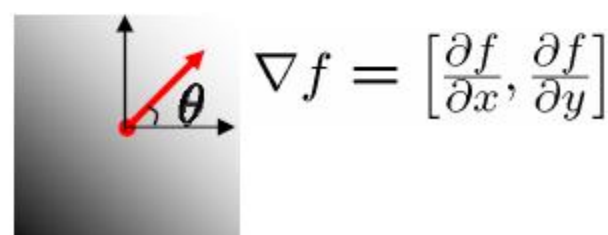
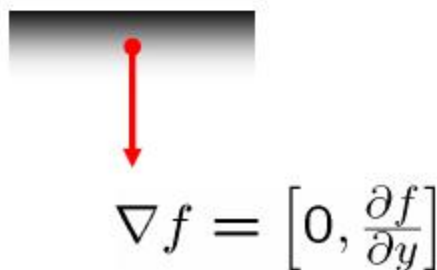
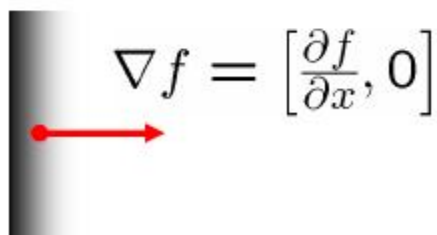
# 梯度

Definition of gradient

$$\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \quad \theta = \arg\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right)$$

where  $\theta$  is the direction of the gradient



# 梯度算子

- 梯度通过一个二维列向量来定义

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

- 向量的模值

$$\begin{aligned} \nabla f &= \text{mag}(\nabla f) \\ &= \left[ G_x^2 + G_y^2 \right]^{\frac{1}{2}} \\ &= \left[ \left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2 \right]^{\frac{1}{2}} \end{aligned}$$

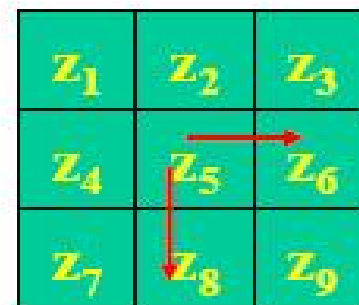
# 梯度算子

- 考虑一个3x3的图像区域， $z$ 代表灰度级，上式在点 $z_5$ 的 $\nabla f$ 值可用数字方式近似。

$$G_x = \frac{\partial f}{\partial x} \quad \text{用 } (z_6 - z_5) \text{ 近似}$$

$$G_y = \frac{\partial f}{\partial y} \quad \text{用 } (z_8 - z_5) \text{ 近似,}$$

组合为：



$$\nabla f \approx \left[ (z_6 - z_5)^2 + (z_8 - z_5)^2 \right]^{\frac{1}{2}}$$

# 梯度算子

- 向量模值的近似计算

用绝对值替换平方和平方根有：

$$\begin{aligned}\nabla f &= \left[ |G_x|^2 + |G_y|^2 \right]^{\frac{1}{2}} \\ &\approx |G_x| + |G_y|\end{aligned}$$



- 微分过滤器的原理

$$\nabla f \approx |z_6 - z_5| + |z_8 - z_5|$$

另外一种计算方法是使用交叉差：

$$\begin{aligned}\nabla f &\approx [(z_9 - z_5)^2 + (z_8 - z_6)^2]^{1/2} \\ &\approx |z_9 - z_5| + |z_8 - z_6|\end{aligned}$$



# 梯度算子

- 微分滤波器模板系数设计
  - Roberts交叉梯度算子
  - Prewitt梯度算子
  - Sobel梯度算子

# 微分滤波器模板系数设计

- Roberts交叉梯度算子

$$\nabla f \approx |z_9 - z_5| + |z_8 - z_6|$$



✓ 梯度计算由两个模板组成，第一个求得梯度的第一项，第二个求得梯度的第二项，然后求和，得到梯度。

✓ 两个模板称为Roberts交叉梯度算子

-1	0
0	1

0	-1
1	0

# 微分滤波器模板系数设计

- Prewitt 梯度算子——3x3的梯度模板

$$\nabla f \approx \begin{vmatrix} (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3) \\ (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7) \end{vmatrix} +$$

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

$z_1$	$z_2$	$z_3$
$z_4$	$z_5$	$z_6$
$z_7$	$z_8$	$z_9$

# 微分滤波器模板系数设计

- Sobel 梯度算子——3x3的梯度模板

$$\nabla f \approx \left| (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \right| + \left| (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \right|$$

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

$z_1$	$z_2$	$z_3$
$z_4$	$z_5$	$z_6$
$z_7$	$z_8$	$z_9$

$$\begin{bmatrix} -1/2 & 0 & 1/2 \end{bmatrix}$$

$$\begin{bmatrix} -1/2 \\ 0 \\ 1/2 \end{bmatrix}$$

Sobel operator

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



Original



Roberts



Laplace

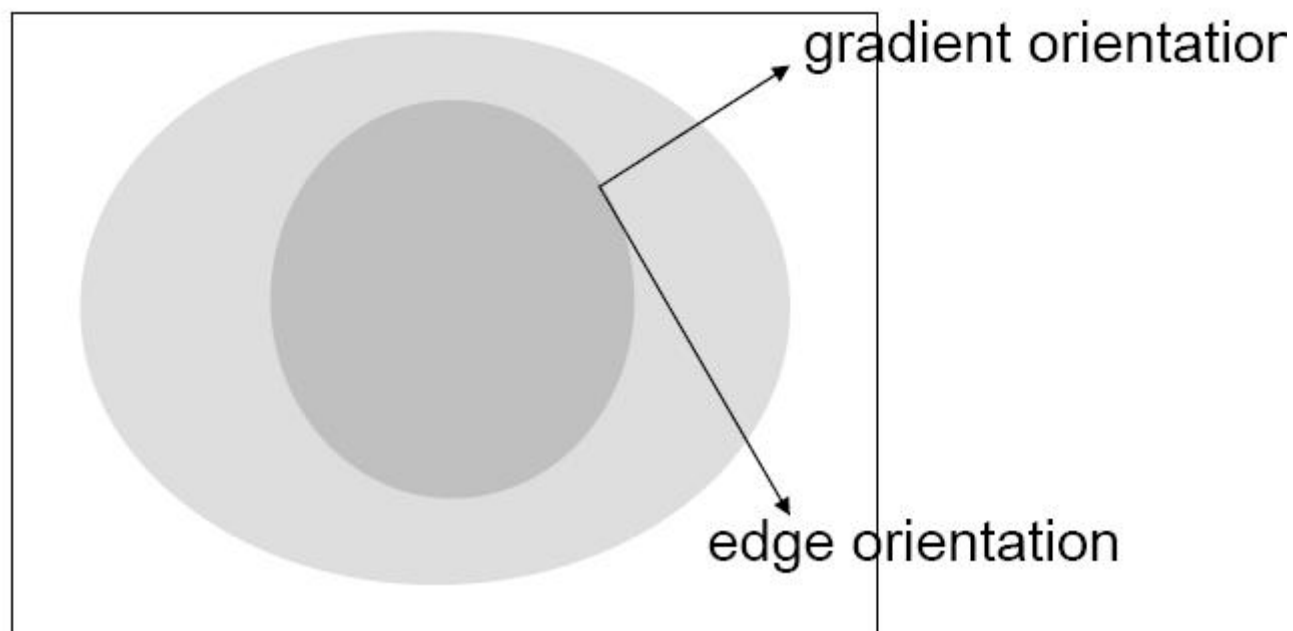


Sobel

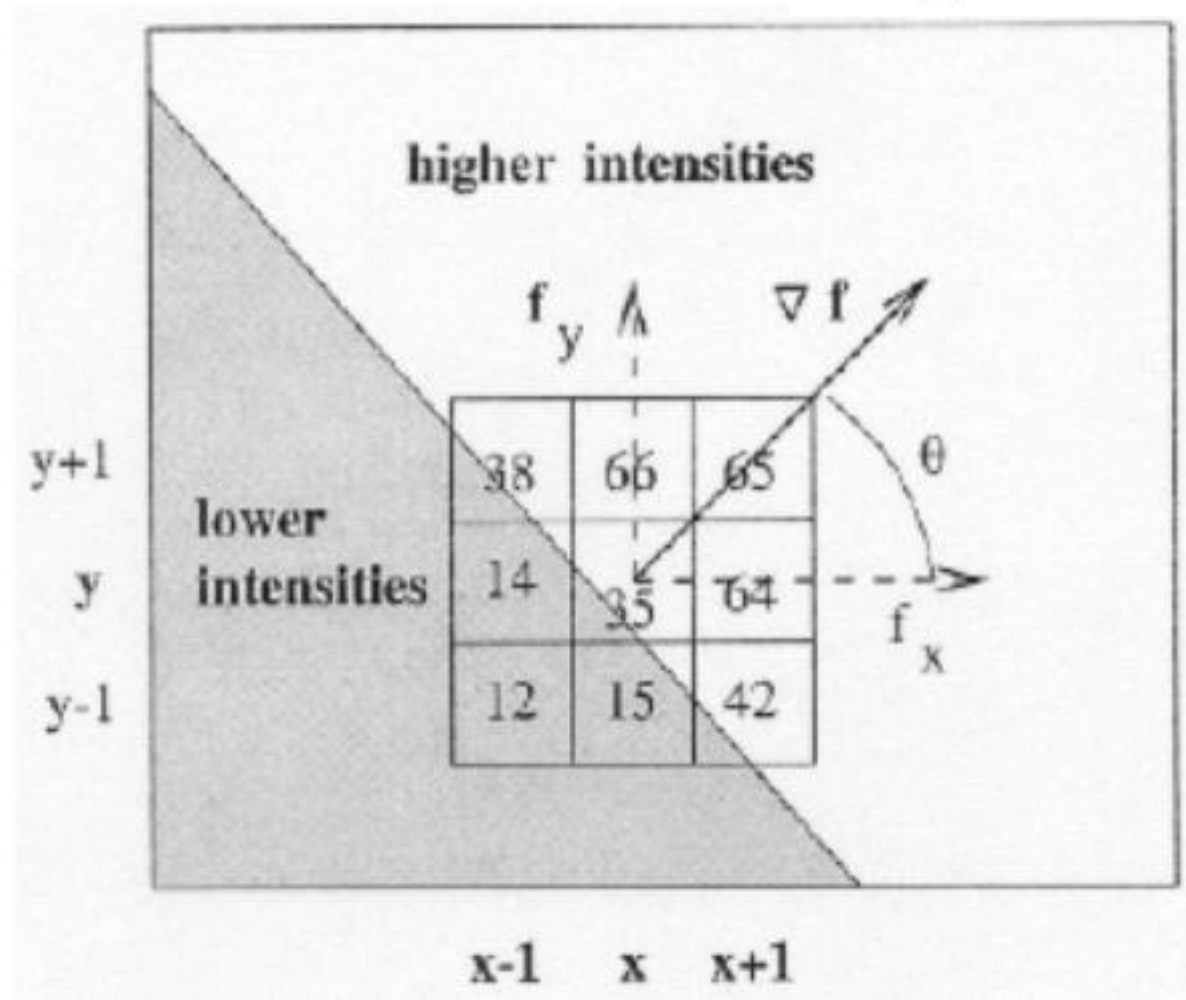
# Canny算法

- 计算每个像素的梯度  $\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$
- 计算梯度强度和方向
- 沿边缘垂直方向寻找梯度最大值
- 这些最大值所在位置即是 边缘

- 寻找梯度最大之处



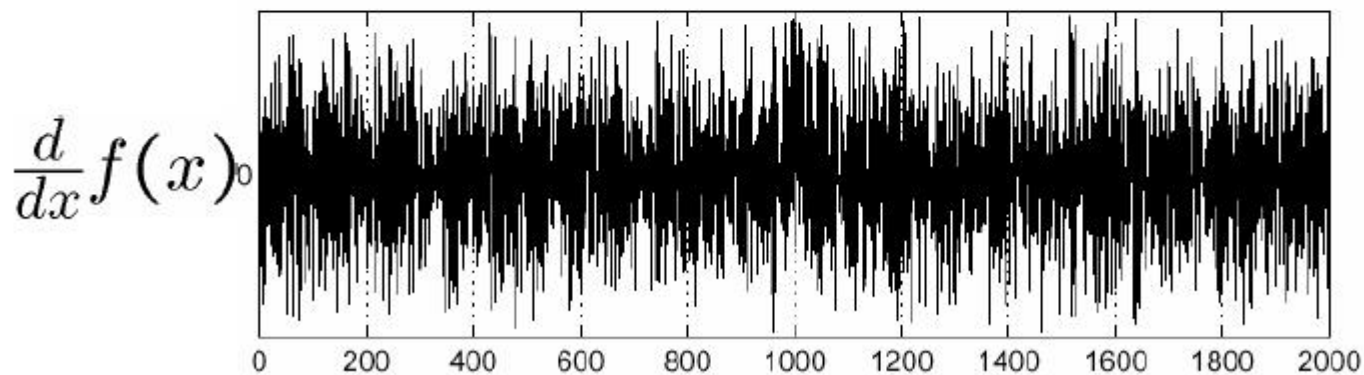
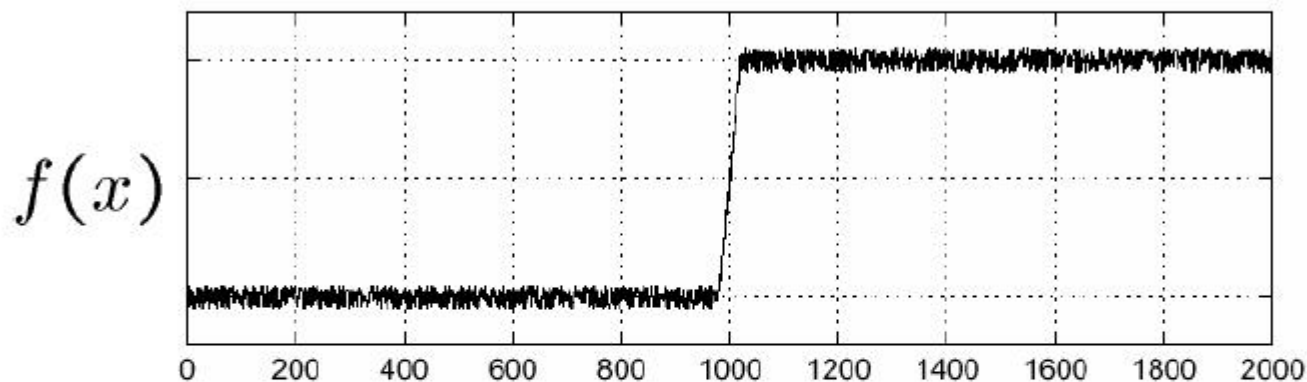
$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



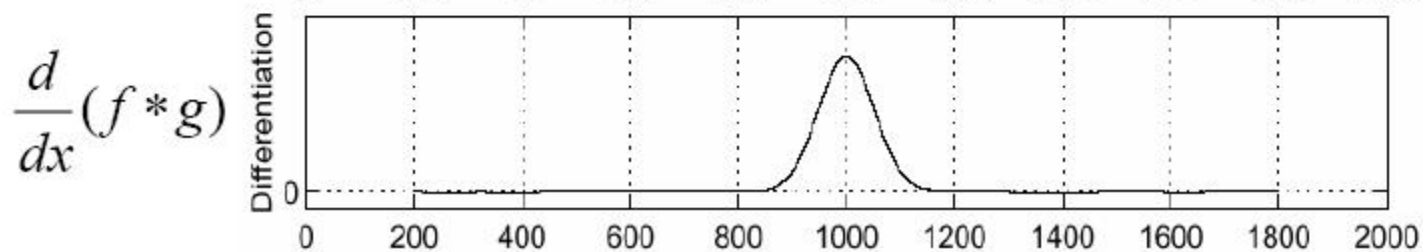
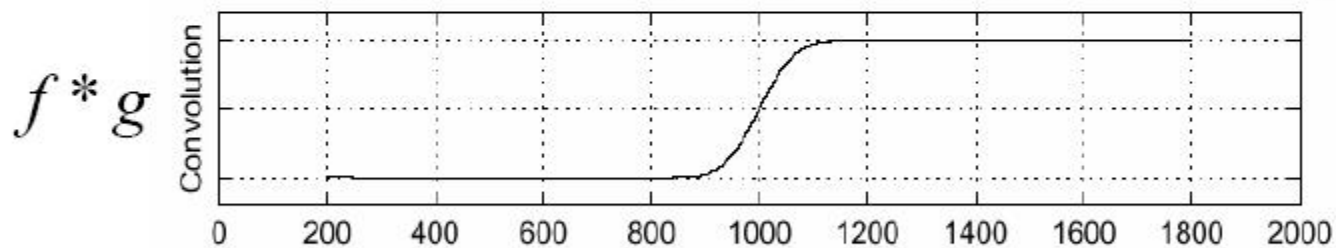
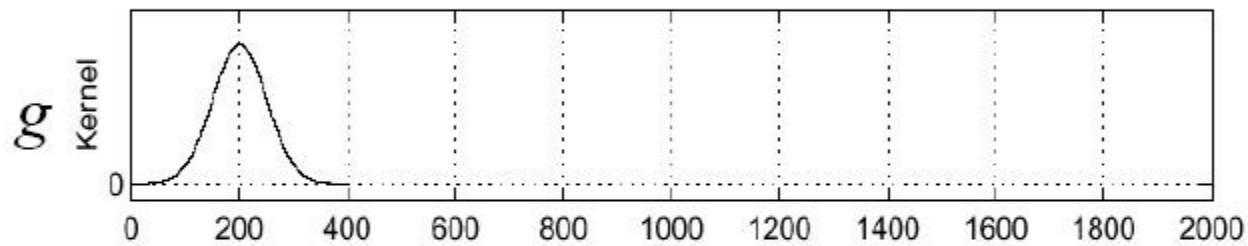
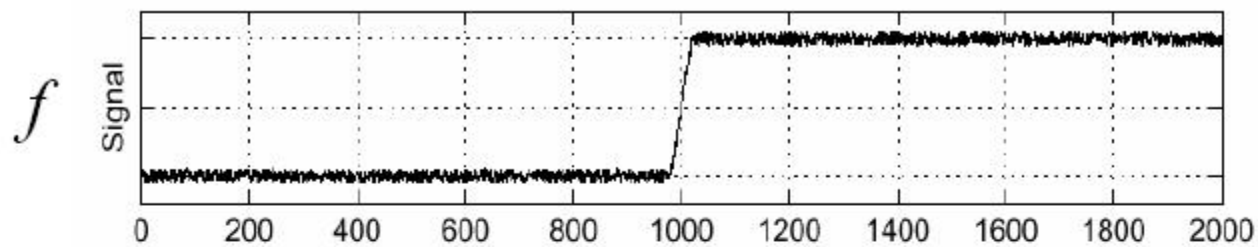


# Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal

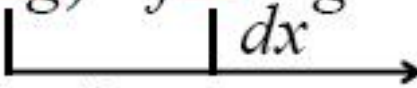


Sigma = 50

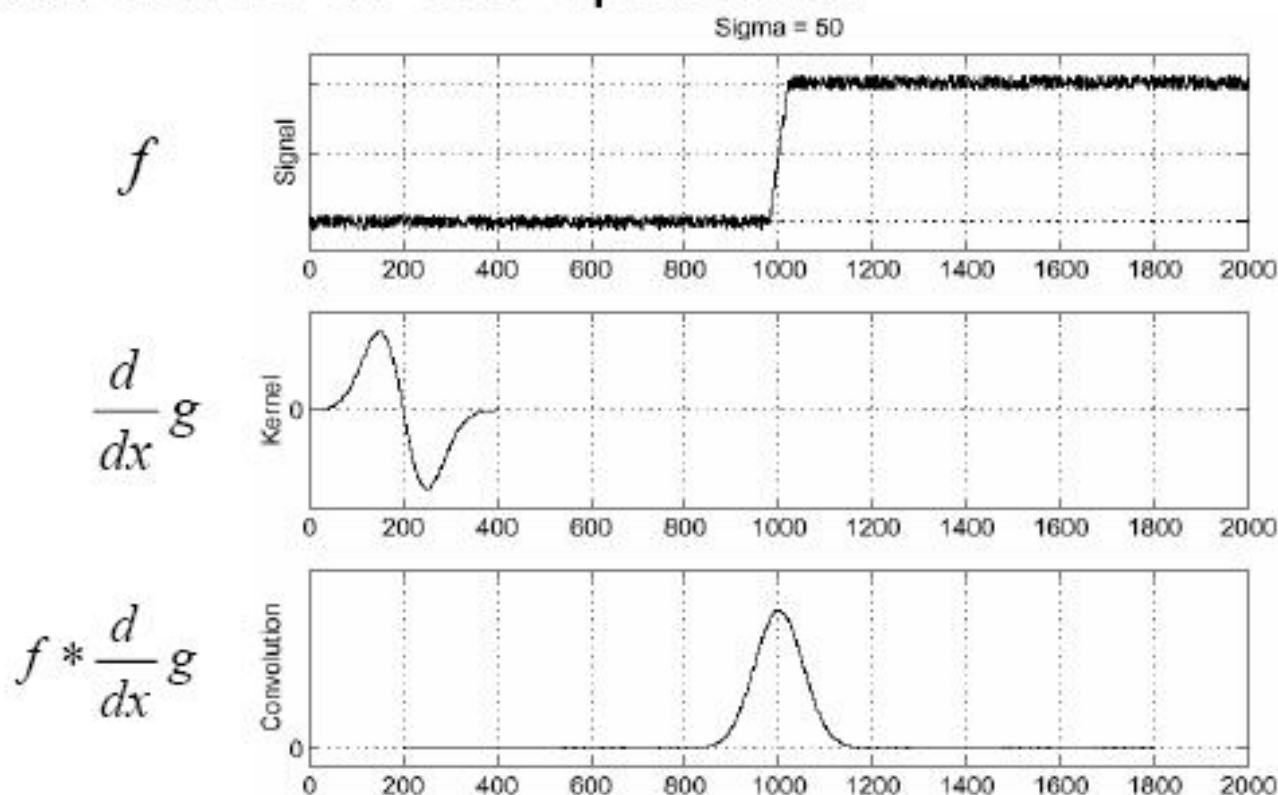


- Differentiation of convolution has the following property:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$



- This saves us one operation:



# 基于多方向Sobel算子的图像边缘检测

四个方向Sobel算子

$$A = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix};$$

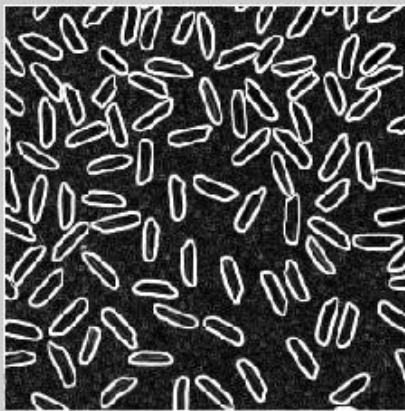
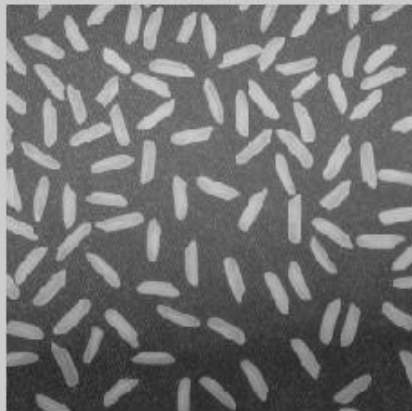
$$B = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix};$$

$$U = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix};$$

$$V = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix};$$

原图像

联合Sobel算子A、B、U、V进行边缘检测结果



# 基于多方向Sobel算子的图像边缘检测

```
1 % 联合Sobel算子A、B、U、V进行边缘检测结果
2 - close all;clear;clc;
3 - I=imread('rice.png');%m*n的单通道灰度图
4 - subplot(1,2,1)
5 - imshow(I)
6 - title('原图像');
7 - [h,w]=size(I);
8 - I_reshape=zeros(h+2,w+2);%创建一个上下左右均多一个像素宽的零矩阵
9 - I_reshape(2:h+1,2:w+1)=I;%把I矩阵拷贝到新创建矩阵中间
10 - g=zeros(h,w);%用于保存结果的矩阵
11 - A=[-1 -2 -1;
12     0 0 0;
13     1 2 1];
14 - B=[-1 0 1;
15     -2 0 2;
16     -1 0 1];
17 - U=[-2 -1 0;
18     -1 0 1;
19     0 1 2];
20 - V=[0 1 2;
21     -1 0 1;
22     -2 -1 0];
23 %I在I_reshape矩阵中纵向方向的范围是j=2到j=h+1，其中j=1和j=h+2对应0像素值
24 %I在I_reshape矩阵中横向方向的范围是i=2到i=w+1，其中i=1和i=w+2对应0像素值
25 - for j=2:h+1
26 -     for i=2:w+1
27 -         aa=0;
28 -         bb=0;
29 -         uu=0;
30 -         vv=0;
31 -         for r=-1:1
32 -             for s=-1:1
33 -                 aa=aa+I_reshape(j+r,i+s)*A(r+2,s+2);%在3*3领域进行卷积操作
34 -                 bb=bb+I_reshape(j+r,i+s)*B(r+2,s+2);%在3*3领域进行卷积操作
35 -                 uu=uu+I_reshape(j+r,i+s)*U(r+2,s+2);%在3*3领域进行卷积操作
36 -                 vv=vv+I_reshape(j+r,i+s)*V(r+2,s+2);%在3*3领域进行卷积操作
37 -             end
38 -         end
39 -         t=max(abs(aa),abs(bb));
40 -         t=max(t,abs(uu));
41 -         t=max(t,abs(vv));
42 -         g(j-1,i-1)=t;%取最大值作为模值
43 -     end
44 - end
45 - subplot(1,2,2)
46 - g=uint8(g);
47 - imshow(g)
48 - title('联合Sobel算子A、B、U、V进行边缘检测结果');
```

根据算法思想，把程序转换为  
VC+OpenCV形式  
或Python+OpenCV形式

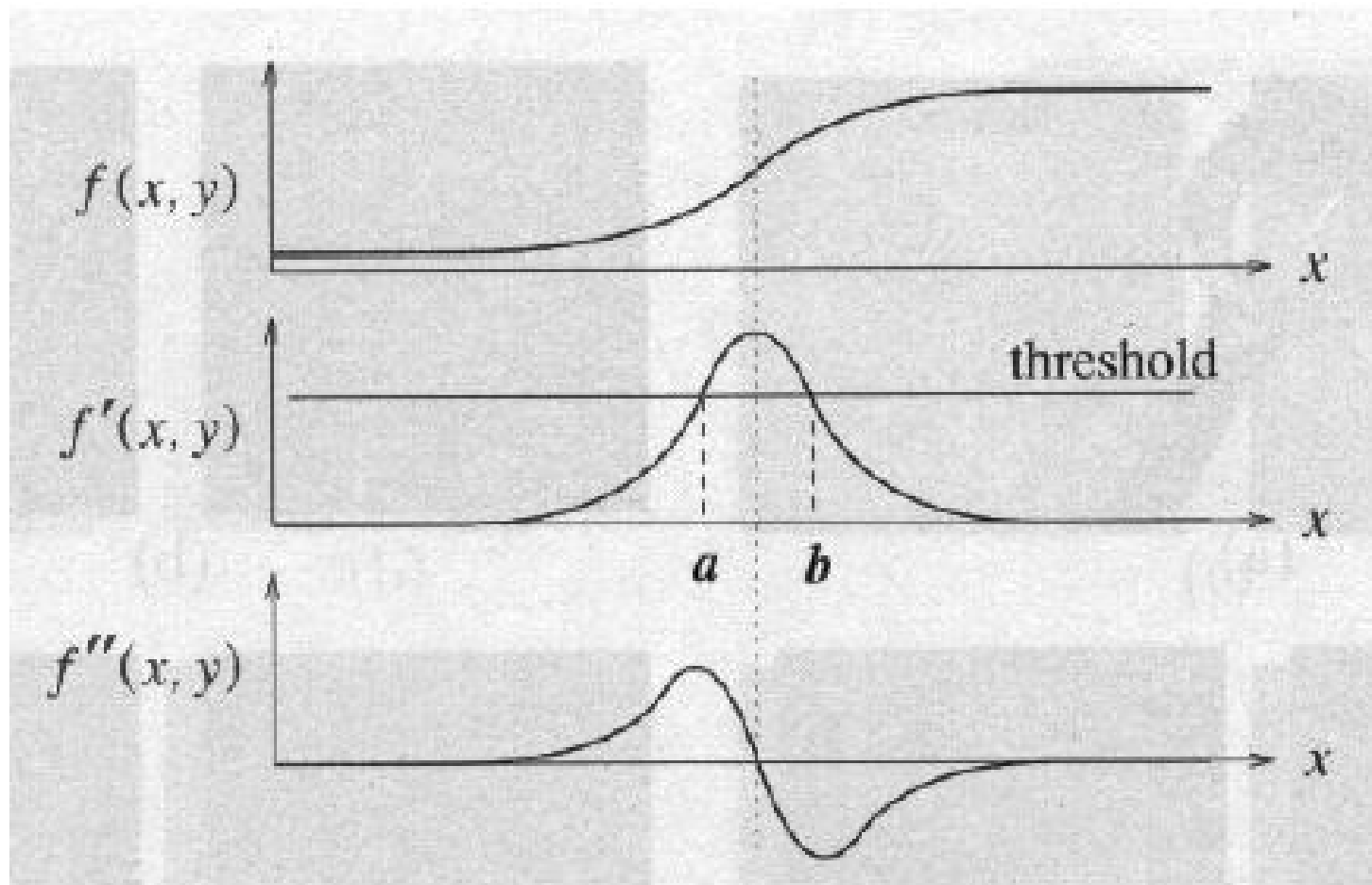
## 滤波操作注意事项

- ① 输入图像为单通道无符号整型；
- ② 新建两个变量，一个存储输入图像，一个存储结果图像，每次操作的滤波结果存入结果图像；
- ③ 注意边界，不要越界，防止内存溢出，访问出错；
- ④ 所有计算是在浮点型状态下进行；
- ⑤ 最后要把结果转换为0-255的整型，再传给结果图像。

```
void CBasicMethod::MySobel(Mat InputImg, Mat &Gray32Mat, float *Operator, int OperatorSize)
{
    //InputImg是灰度图像

    Gray32Mat.create(InputImg.size(), CV_32F); //CV_16S - 16-bit signed integers ( -32768..32767 )
    Mat<uchar> dataMat = InputImg;
    Mat<float> dataResultMat = Gray32Mat;
    int oh = OperatorSize / 2; //算子Operator的半径
    int ow = OperatorSize / 2;
    for (int j = 0; j < InputImg.rows; j++)
    {
        for (int i = 0; i < InputImg.cols; i++)
        {
            float datavalue = 0;
            for (int p = -oh; p <= oh; p++)
            {
                for (int q = -ow; q <= ow; q++)
                {
                    if (p + j < 0 || p + j > InputImg.rows - 1 || q + i < 0 || q + i > InputImg.cols - 1)
                    {
                        continue;
                    }
                    datavalue += dataMat(j + p, i + q) * Operator[(p + oh) * OperatorSize + (q + ow)];
                }
            }
            dataResultMat(j, i) = saturate_cast<float>(datavalue);
        }
    }
}
```

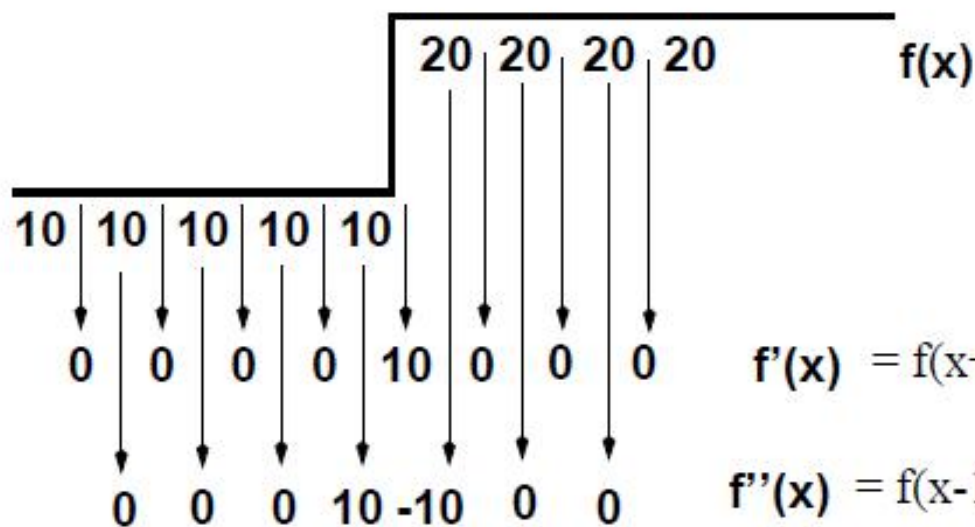
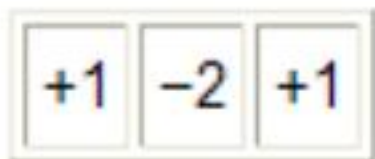
# 二阶导数



# 二阶导数

$$I''(x) = 1 \cdot I(x-1) - 2 \cdot I(x) + 1 \cdot I(x+1).$$

$$((I(x+1) - I(x)) - (I(x) - I(x-1)))$$



$$f'(x) = f(x+1) - f(x) \quad (\text{approximates } f'() \text{ at } x+1/2)$$

$$f''(x) = f(x-1) - 2f(x) + f(x+1) \quad (\text{approximates } f''() \text{ at } x)$$

zero-crossing



$$\text{mask } M = [-1, 2, -1]$$

$S_1$			12	12	12	12	12	24	24	24	24	24
$S_1$	$\otimes$	$M$	0	0	0	0	-12	12	0	0	0	0

(a)  $S_1$  is an upward step edge

$S_2$			24	24	24	24	24	12	12	12	12	12
$S_2$	$\otimes$	$M$	0	0	0	0	12	-12	0	0	0	0

(b)  $S_2$  is a downward step edge

$S_3$			12	12	12	12	15	18	21	24	24	24
$S_3$	$\otimes$	$M$	0	0	0	-3	0	0	0	3	0	0

(c)  $S_3$  is an upward ramp

$S_4$			12	12	12	12	24	12	12	12	12	12
$S_4$	$\otimes$	$M$	0	0	0	-12	24	-12	0	0	0	0

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

0	1	0
1	-4	1
0	1	0

- Approximating  $\nabla^2 f$ :

$$\frac{\partial^2 f}{\partial x^2} = f(i, j+1) - 2f(i, j) + f(i, j-1)$$

$$\frac{\partial^2 f}{\partial y^2} = f(i+1, j) - 2f(i, j) + f(i-1, j)$$

$$\nabla^2 f = -4f(i, j) + f(i, j+1) + f(i, j-1) + f(i+1, j) + f(i-1, j)$$

# 拉普拉斯算子

The diagram illustrates the 3x3 grid of values for  $f(x,y)$  and its neighbors. The central value is  $f(x,y) = -4$ . The values for the neighbors are:

- $f(x-1,y) = 1$
- $f(x,y-1) = 1$
- $f(x+1,y) = 1$
- $f(x,y+1) = 1$
- $f(x-1,y-1) = 0$
- $f(x-1,y+1) = 1$
- $f(x+1,y-1) = 1$
- $f(x+1,y+1) = 1$

The values for the corners are all 0.

a	b
c	d

**FIGURE 3.39**  
(a) Filter mask used to implement the digital Laplacian, as defined in Eq. (3.7-4).  
(b) Mask used to implement an extension of this equation that includes the diagonal neighbors. (c) and (d) Two other implementations of the Laplacian.

- (a) 执行离散拉普拉斯变换所用的滤波器掩模
- (b) 扩展的拉普拉斯掩模，包括了对角线邻域
- (c) 和(d)其他两种拉普拉斯的实现

0	1	0
1	-4	1
0	1	0

求和运算

8	8	8
8	8	8
8	8	8

1	6	6	6
1	6	6	6
1	6	6	6
3	3	3	5
4	4	4	9
4	4	4	9
4	4	4	9

8	8	8	1	6	6	6
8	0	-7	12	-5	0	6
8	-5	-12	16	-4	1	6
3	6	8	0	-8	-6	7
4	-1	4	-16	12	5	2
4	0	5	-12	7	0	2
4	4	4	9	2	2	2

填充后的图像

结果图像

0	1	0
1	-4	1
0	1	0

求和运算

8	8	8	1	6	6	6
8	8	8	1	6	6	6
8	8	8	1	6	6	6
3	3	3	5	7	7	7
4	4	4	9	2	2	2
4	4	4	9	2	2	2
4	4	4	9	2	2	2

8	8	8	1	6	6	6
8	0	-7	12	-5	0	6
8	-5	-12	16	-4	1	6
3	6	8	0	-8	-6	7
4	-1	4	-16	12	5	2
4	0	5	-12	7	0	2
4	4	4	9	2	2	2

填充后的图像

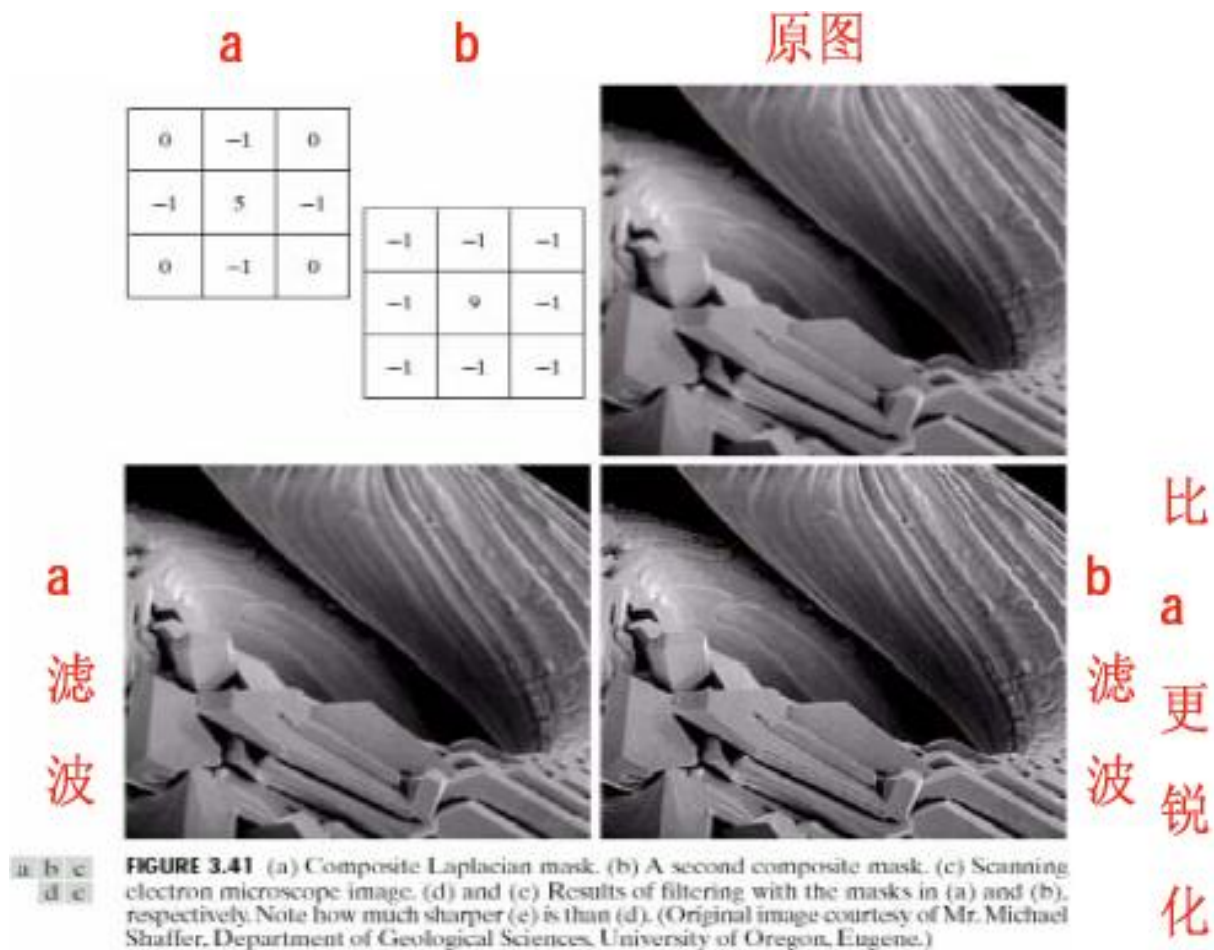
结果图像

# 拉普拉斯算子

拉普拉斯微分算子强调图像中灰度的突变,弱化灰度慢变化的区域。这将产生一幅把浅灰色边线、突变点叠加到暗背景中的图像。



# 拉普拉斯算子一例



# 拉普拉斯算子

将原始图像和拉普拉斯图像叠加在一起的方法可以保护拉普拉斯锐化处理的效果，同时又能复原背景信息。

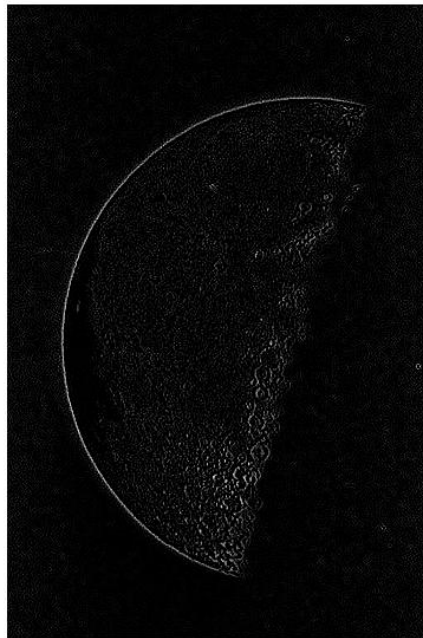
- 拉普拉斯变换对图像增强的基本方法

$$g(x, y) = \begin{cases} f(x, y) - \nabla^2 f(x, y) & (1) \\ f(x, y) + \nabla^2 f(x, y) & (2) \end{cases}$$

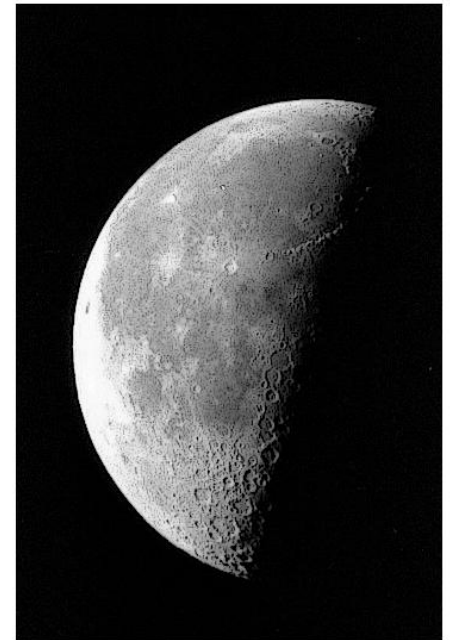
(1)用于拉普拉斯模板中心系数为负

(2)用于拉普拉斯模板中心系数为正

original image:moon.tif



g8





# 拉普拉斯算子

$$g(x, y) = f(x, y) - \nabla^2 f(x, y)$$

$$= f(x, y) - \{[f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 4f(x, y)\}$$

$$= 5f(x, y) - [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)]$$

$$g(x, y) = f(x, y) + \nabla^2 f(x, y)$$

$$= f(x, y) + \{[f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 4f(x, y)\}$$

$$= [f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)] - 3f(x, y)$$

# 拉普拉斯算子

可以用下面的掩模  
一次扫描来实现

0	-1	0
-1	5	-1
0	-1	0

0	1	0
1	-3	1
0	1	0

