

Computer Vision

Ch.9 Object Detection #1

Prof. Po-Yueh Chen (陳伯岳)

E-mail: pychen@cc.ncue.edu.tw

Ext: 8440

NCUE CSIE

Edge Detection & Line Extraction

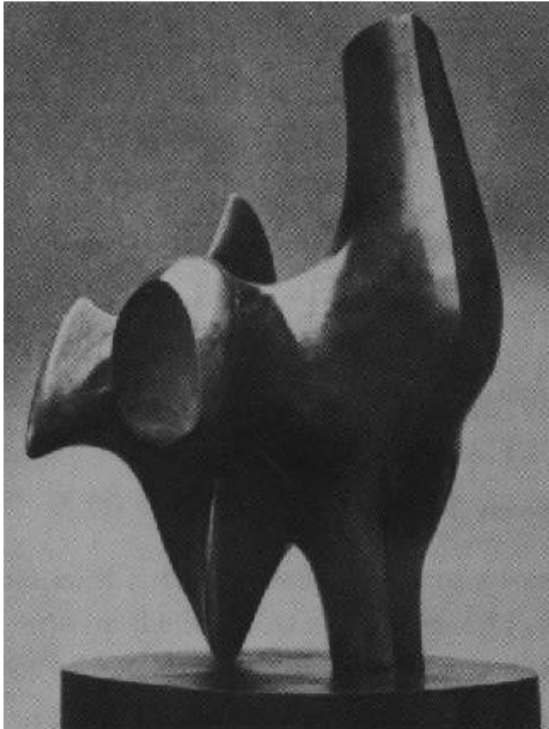
➤ **Background about image edge**

➤ **Image gradient**

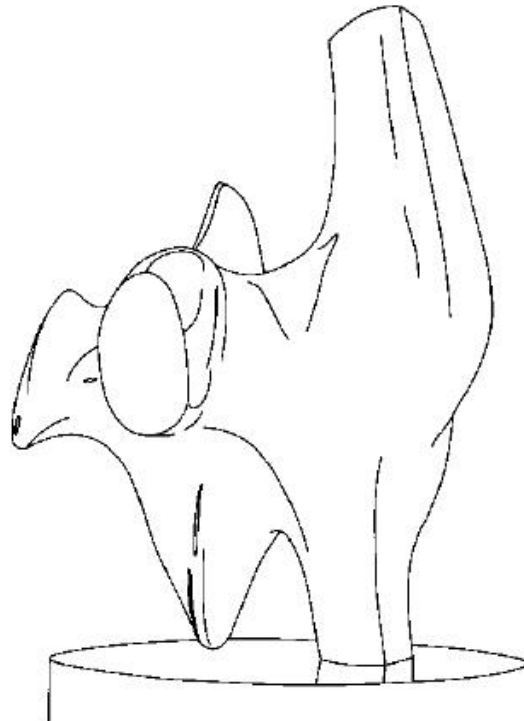
- Canny edge detector
- Hough line transform for line extraction

Edge Detection (1/9)

- ✓ The goal of edge detection is to convert a 2D image into a **set of curves**.
 - It extracts **salient features** of the scene
 - Edges are more **compact** than pixels



Image



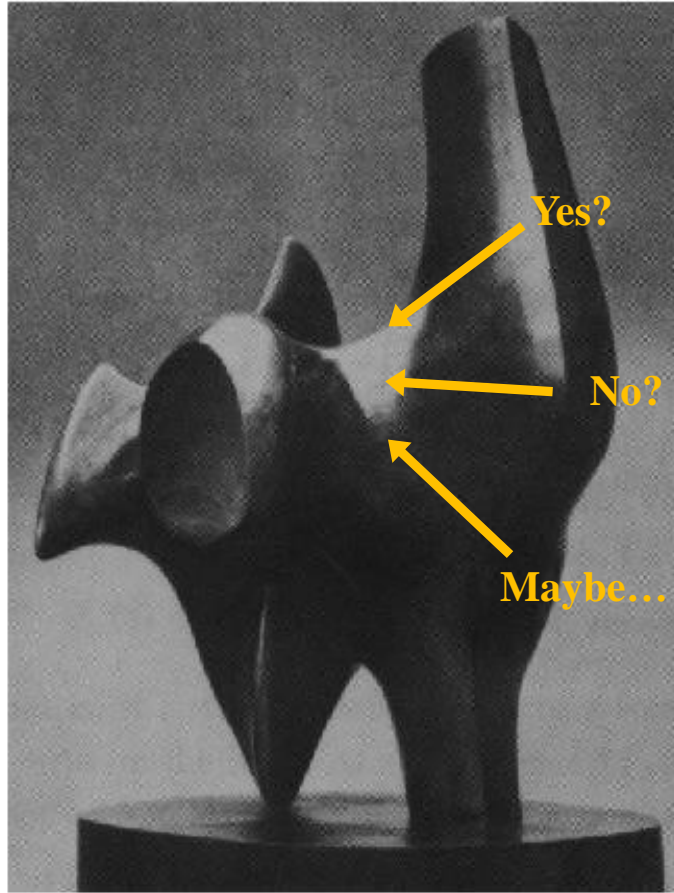
A set of curves

- We don't need to know all the pixel values of the image.
- With only the edges, we can know what the object is.

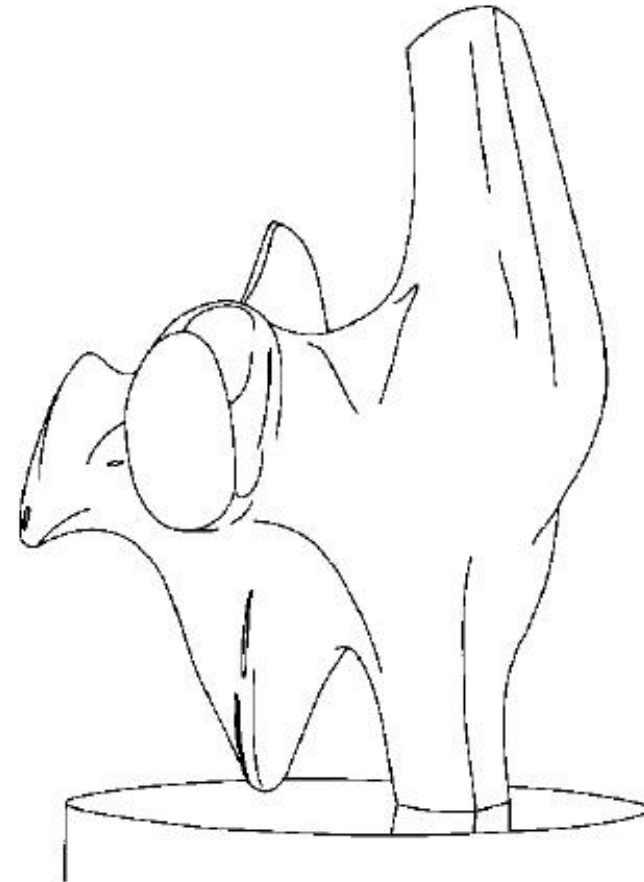
➡ *A statue of an animal (?)*

Edge Detection (2/9)

- So, How to tell that a pixel is on an **edge**?



Image



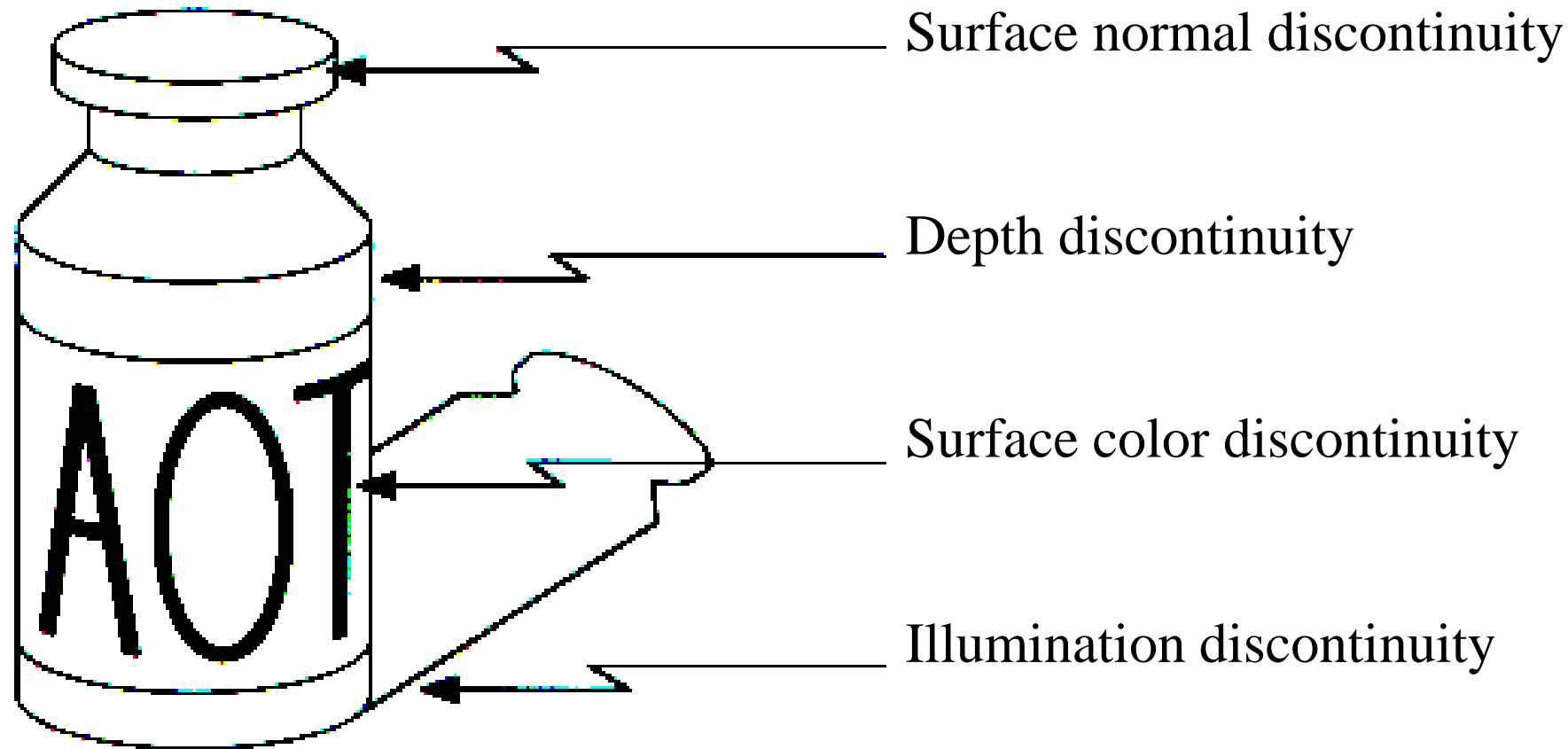
A set of curves

Edge Detection (3/9)

➤ Origin of Edges

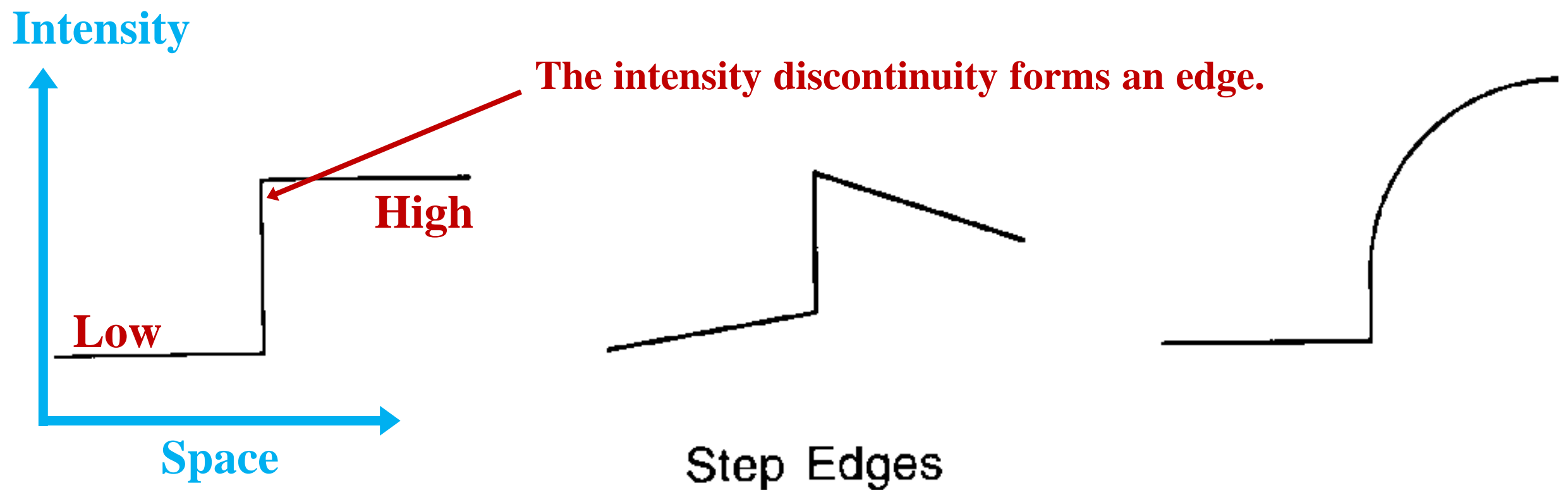
- ✓ Edges are caused by a **variety of factors**

✓ **In fact, discontinuity is where change occurs.**



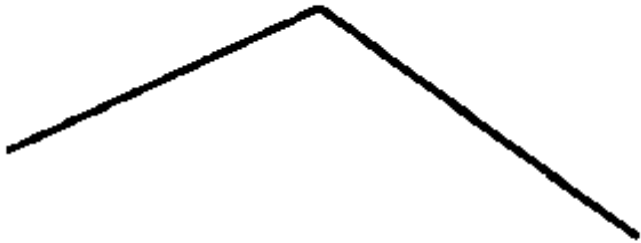
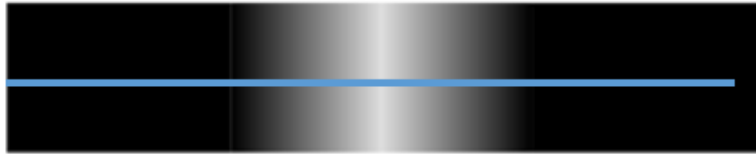
Edge Detection (4/9)

➤ Profiles of common edges



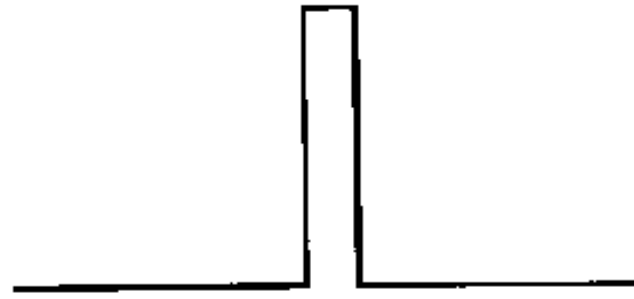
★ Edge Detection (5/9)

➤ Profiles of common edges



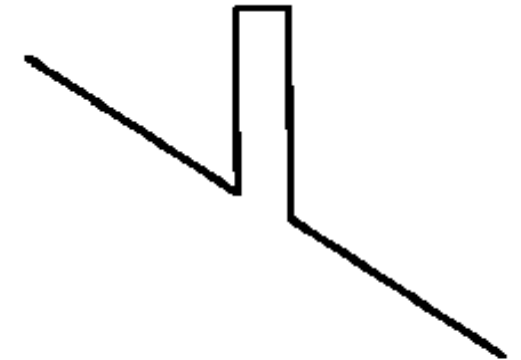
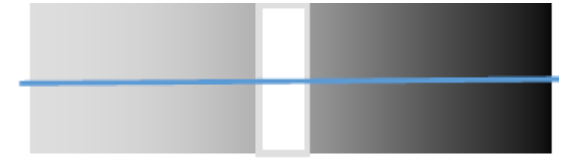
Roof Edge

- ✓ The intensity increases from two sides to the middle, resulting in an edge.



Line Edges

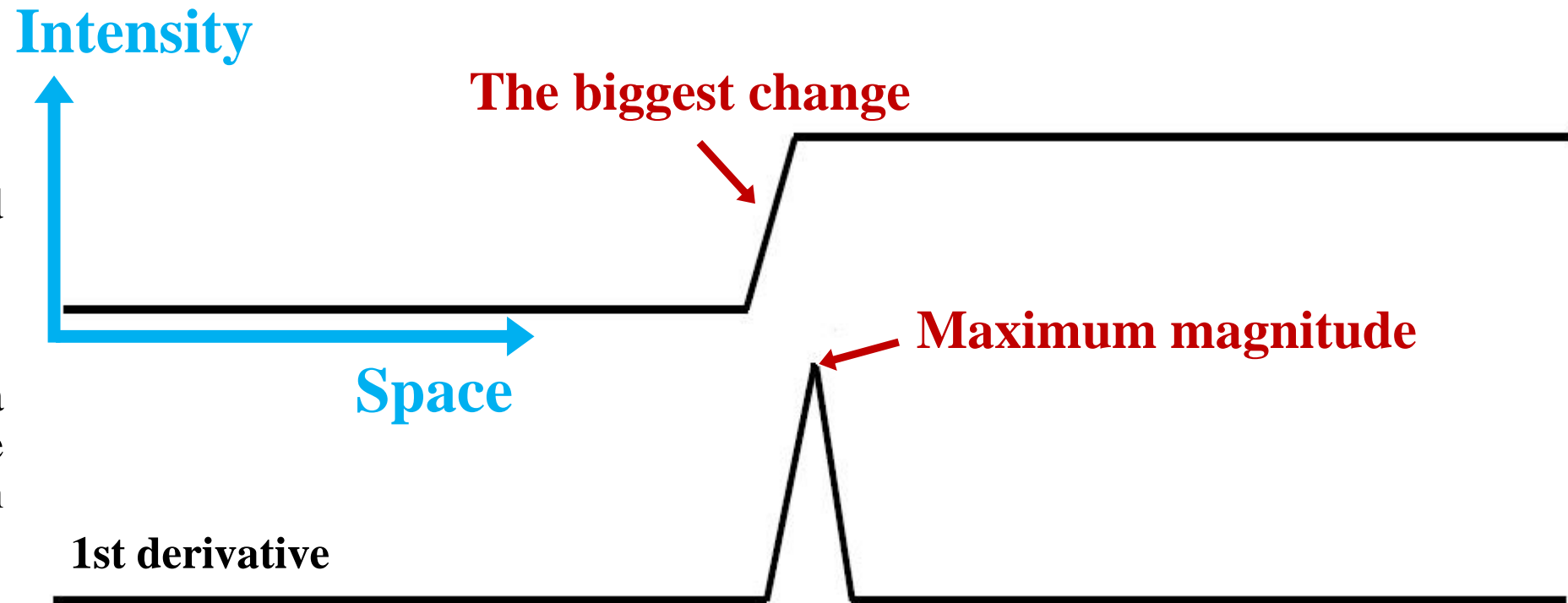
- ✓ There is a bright line of high intensity in the image.



Edge Detection (6/9)

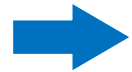
➤ Edge is Where **Change** Occurs

- ✓ Changes can be measured by **derivative in 1D**.
- ✓ The biggest change of a signal can be found at the **maximum magnitude** in its **1st derivative**.

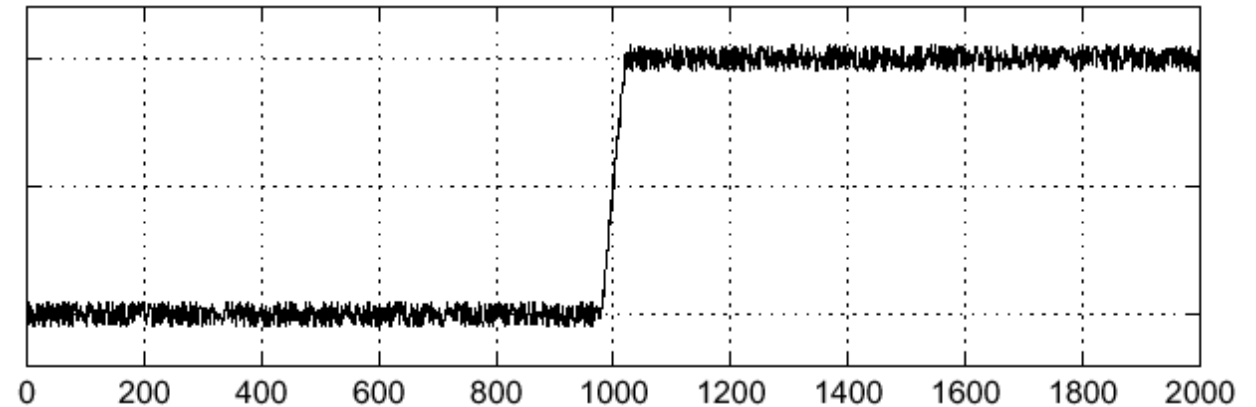


Edge Detection (7/9)

- **Effects of noise** ✓ Consider a single row(or column) of the image
- Plotting intensity as a function of position gives a signal



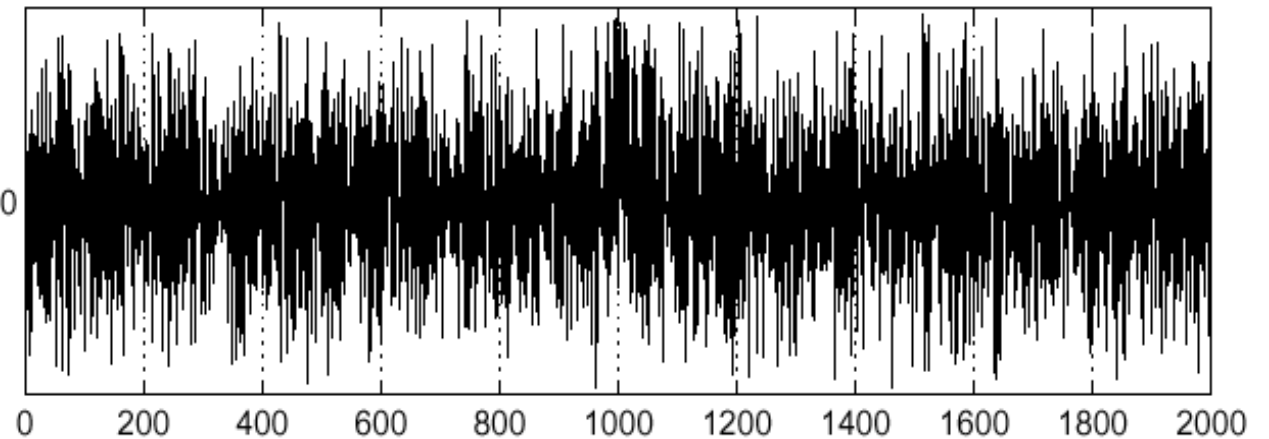
$f(x)$



Step edge with noise

Now, it is hard to know where is the edge, since **derivatives are sensitive to noise**.

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



1st derivative

Edge Detection (8/9)

➤ Solution: Smoothing first

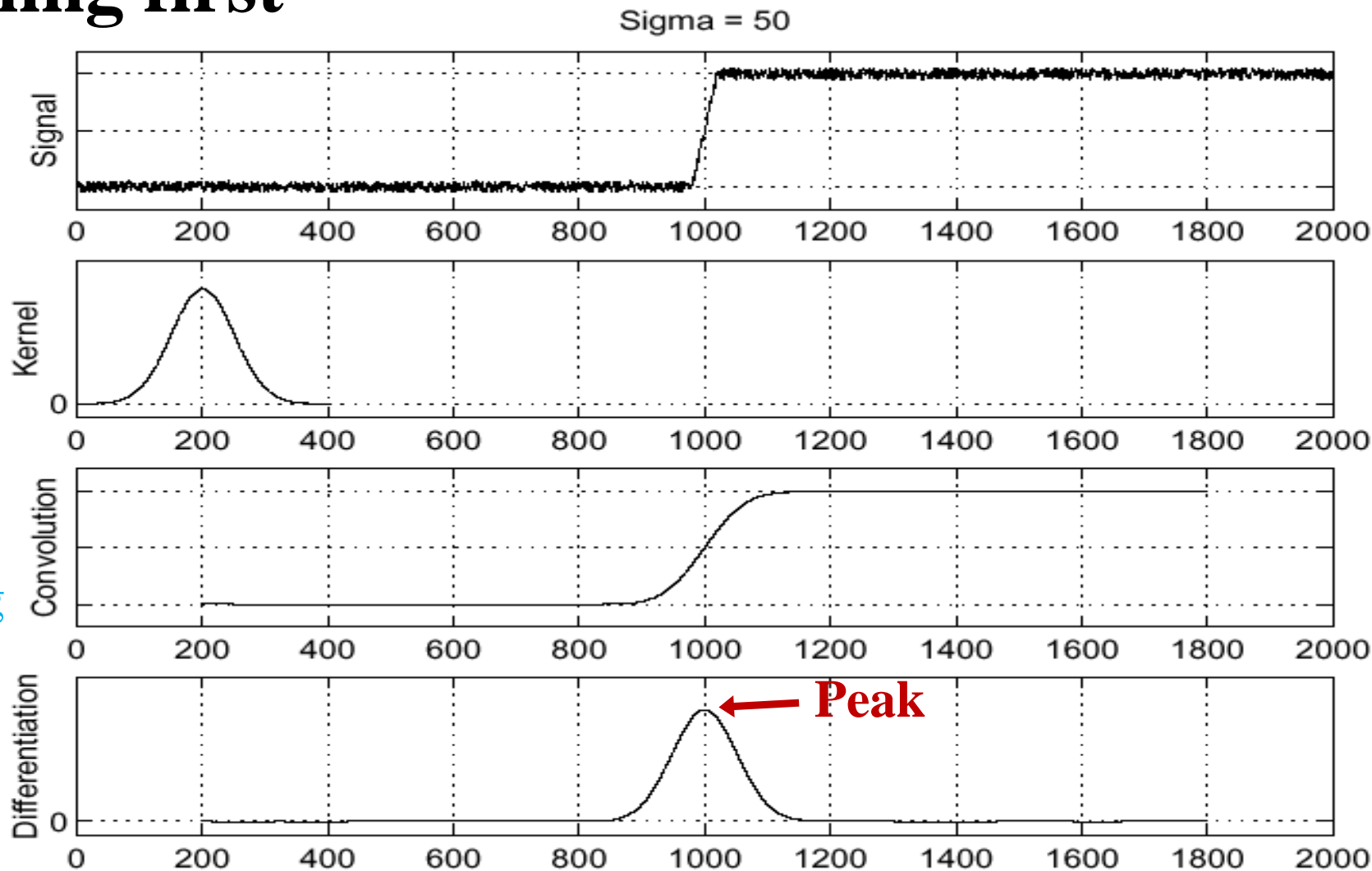
In the previous episode, we have learned several smoothing filters. Among them, Gaussian kernel is most commonly used.

f
Signal with noise

h
Gaussian Kernel

$h \times f$
The result of smoothing

$\frac{\partial}{\partial x}(h \times f)$
1st derivative of the smoothed signal



Edge Detection (9/9)

➤ Three Steps for Edge Detection

Step 1. Image smoothing for noise reduction.

Step 2. Detection of edge points.

- This is a **local operation** that extracts from an image all points that are potential **edge-point candidates**.

➤ **Find all edge-point candidates**

Step 3. Edge localization.

- The objective of this step is to select from the candidate points only the points that are **members of the set of points comprising an edge**.

➤ **Find the true edge points**

Image Gradient (1/7)

- Gradient is a tool for finding edge **strength** and **direction** at an arbitrary **location** (x, y) of an **image** f .
- The gradient is denoted by ∇f and defined as the vector:

$$\nabla f(x, y) \equiv \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix}$$

$g_x(x, y)$: gradient in x-direction
 $g_y(x, y)$: gradient in y-direction

➤ They are computed by the partial derivatives of f with respect to x and y .

Image Gradient (2/7)

- This vector points in the direction of the **maximum change rate** of f at (x, y) .

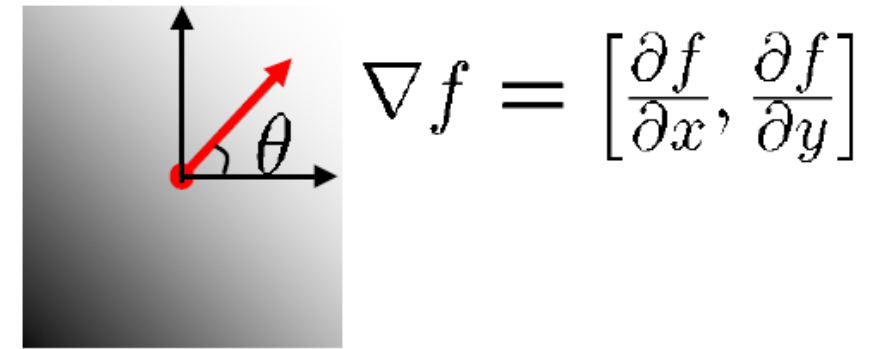
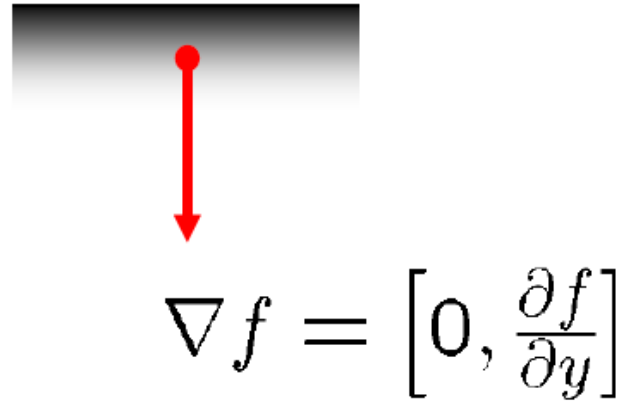
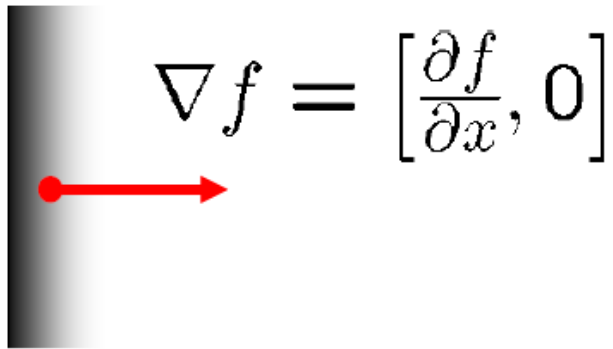


Image Gradient (3/7)

- The **magnitude** of this gradient vector is give by its **Euclidean vector norm**:

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

$$\nabla f(x, y) \equiv \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix}$$

- This is the value of the rate of change in the direction of the gradient vector at point (x, y) .

Image Gradient (4/7)

- Note that $M(x, y)$, $\|\nabla f(x, y)\|$, $g_x(x, y)$, $g_y(x, y)$ are arrays of the same size as the original image f .
- It is common to refer to $M(x, y)$ as the **gradient image**, or simply the **gradient**.



Original image f



Gradient image M

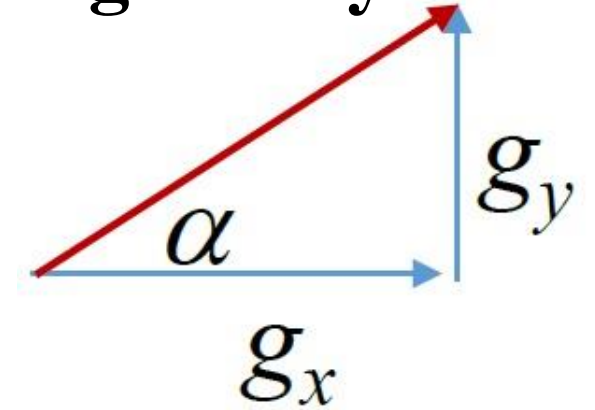
Image Gradient (5/7)



$$\nabla f(x, y) \equiv \begin{bmatrix} g_x(x, y) \\ g_y(x, y) \end{bmatrix}$$

- The direction of the gradient vector at point (x, y) is given by

$$\alpha(x, y) = \tan^{-1} \left(\frac{g_y(x, y)}{g_x(x, y)} \right)$$



- Angles are measured in the counterclockwise direction with respect to the x-axis.
- $\alpha(x, y)$ is also of the same size as the original image f .

Image Gradient (6/7)

- The direction of an edge at point (x, y) is orthogonal to the direction, $\alpha(x, y)$, of the gradient.

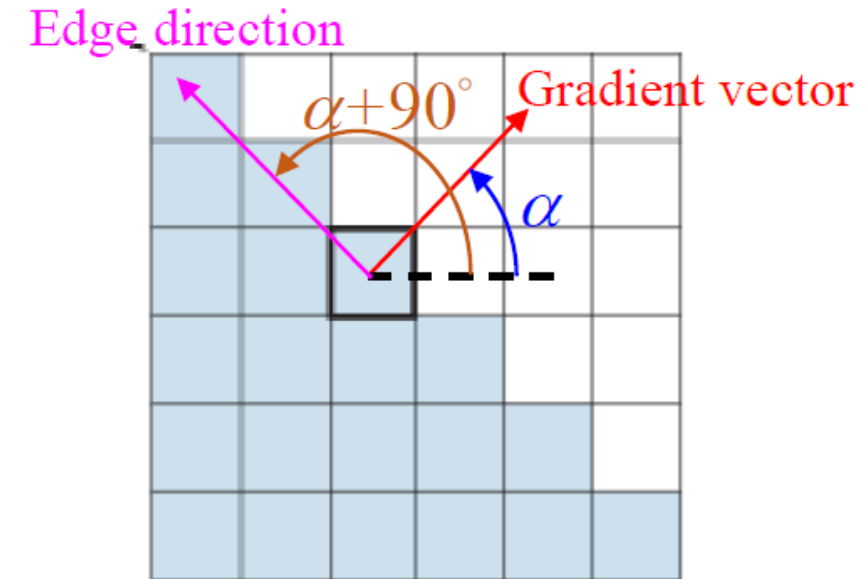
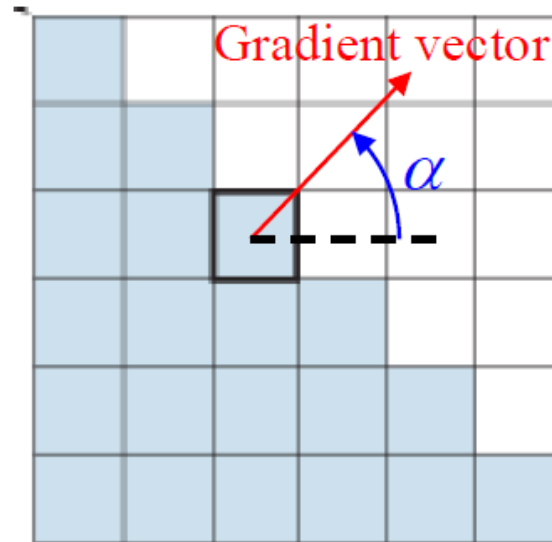
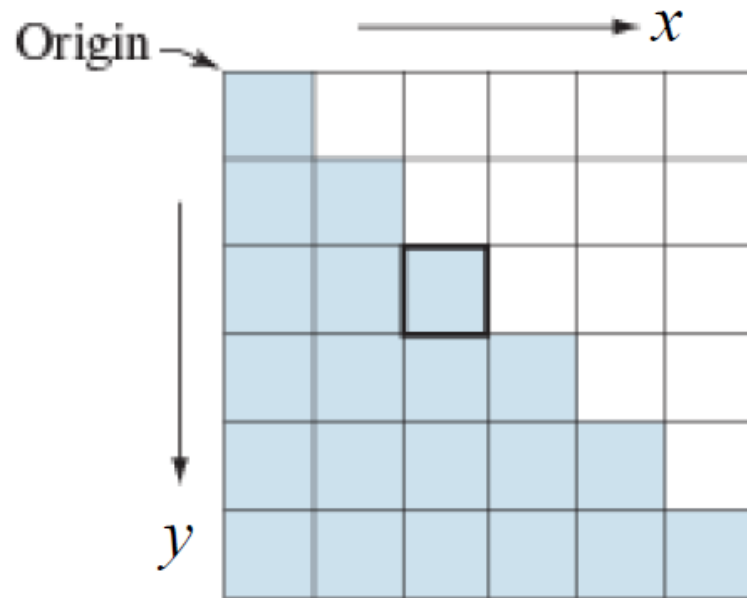


Image Gradient (7/7)

- **Edge strength** can be estimated by the magnitude of the gradient.

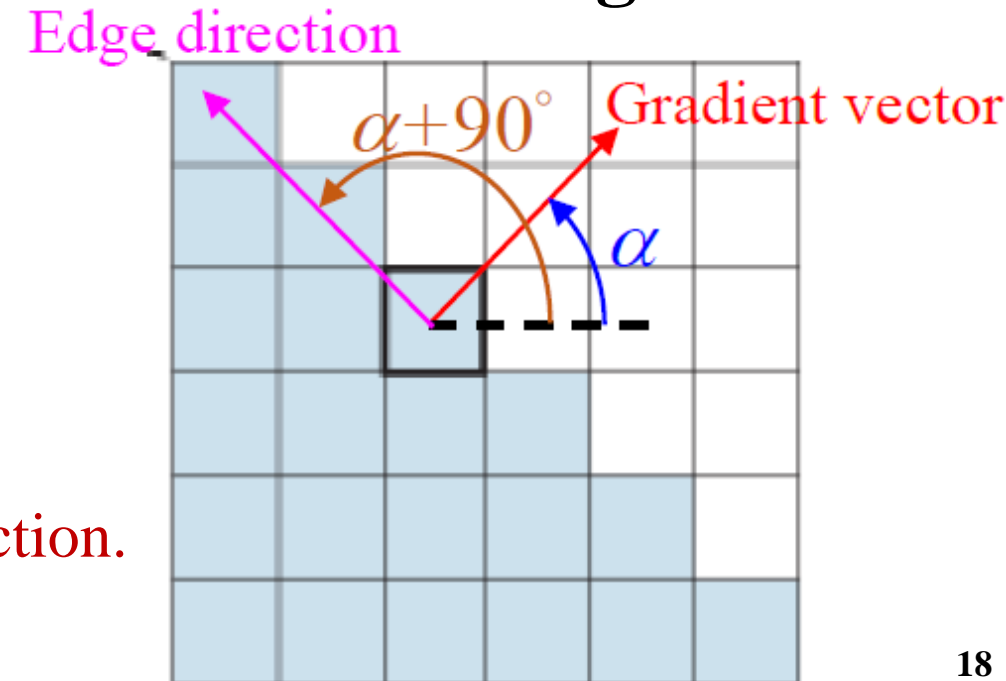
$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

- **Edge direction** can be estimated by the direction of the gradient.

$$\alpha(x, y) = \tan^{-1} \left(\frac{g_y(x, y)}{g_x(x, y)} \right)$$

$$\text{Edge direction} = \alpha + 90^\circ$$

The edge direction is orthogonal to the gradient direction.



Gradient Operator (1/23)

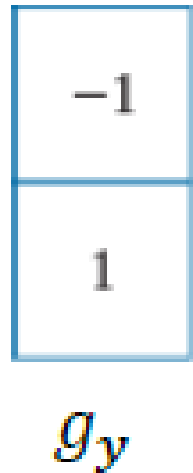
- Obtaining the gradient of an image requires computing the partial derivatives in x - and y -directions ($\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$) at every pixel location.
- We can use the following **digital approximation**:

$$g_x = \frac{\partial f(x,y)}{\partial x} \approx f(x+1, y) - f(x, y)$$

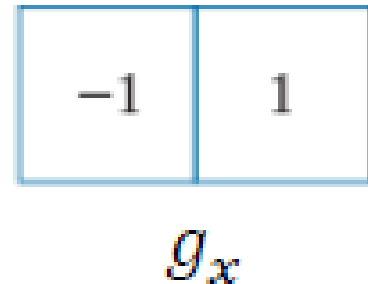
$$g_y = \frac{\partial f(x,y)}{\partial y} \approx f(x, y+1) - f(x, y)$$

Gradient Operator (2/23)

- The two equations can be implemented with 1D kernels.



A vertical 1D kernel represented as a blue-outlined box containing two cells. The top cell contains the value -1 and the bottom cell contains the value 1. Below the box is the label g_y in a blue, italicized font.



A horizontal 1D kernel represented as a blue-outlined box containing two cells. The left cell contains the value -1 and the right cell contains the value 1. Below the box is the label g_x in a blue, italicized font.

- We can use these 1D kernels to perform **convolution** on an image for simple gradient estimation.
- However, these 1D kernels can detect vertical and horizontal edges only.

Gradient Operator (3/23)

- When diagonal edge direction is of interest, we need 2D kernels, e.g., Roberts operators.
- Kernels of size 2×2 are simple conceptually.
- But they are not as useful for computing edge direction as kernels that are symmetric about their centers.
- The smallest of such symmetric kernels are of size 3×3 .
- These kernels take into account the nature of the data on opposite sides of the center point, and thus carry more information regarding the direction of an edge.

| | | | |
|----|---|---|----|
| -1 | 0 | 0 | -1 |
| 0 | 1 | 1 | 0 |

Roberts

| | | | | | |
|----|----|----|----|---|---|
| -1 | -1 | -1 | -1 | 0 | 1 |
| 0 | 0 | 0 | -1 | 0 | 1 |
| 1 | 1 | 1 | -1 | 0 | 1 |

Prewitt

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

Sobel

Gradient Operator (4/23)

- The simplest **digital approximation** to the partial derivatives using kernels of size 3×3 are given by **Prewitt operators**.
- The difference between the first and third rows (columns) of the 3×3 region approximates the derivative in the y-direction (x-direction).
- For **Sobel operators**, as light variation uses a weight of 2 in the center coefficient.

| | | | | | |
|----|----|----|----|---|---|
| -1 | -1 | -1 | -1 | 0 | 1 |
| 0 | 0 | 0 | -1 | 0 | 1 |
| 1 | 1 | 1 | -1 | 0 | 1 |

Prewitt

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

Sobel

Gradient Operator (5/23)

- Compared to the Prewitt kernels, the Sobel kernels have better **noise-suppression** (smoothing) characteristics.
- The coefficients of all the kernels in the figure sum to **zero**.
 - Thus, it gives a response of **zero** in a area of **constant intensity**.
- The computation of gradient image requires **high computational** burden of **squares** and **square root**.

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

- An approach used frequently is to approximate the gradient magnitude by absolute value:

$$M(x, y) \approx |g_x(x, y)| + |g_y(x, y)|$$

Gradient Operator (6/23)

➤ Code

Syntax:

Sobel(src, dst, ddepth, dx, dy, ksize, scale, delta, borderType)

src – Source 8-bit or floating-point, 1-channel or 3-channel image.

dst – Output image of the same size and the same number of channels as src .

ddepth – Output image depth; the following combinations of src.depth() and ddepth are supported:

src.depth() = CV_8U, ddepth = -1/CV_16S/CV_32F/CV_64F

src.depth() = CV_16U/CV_16S, ddepth = -1/CV_32F/CV_64F

src.depth() = CV_32F, ddepth = -1/CV_32F/CV_64F

src.depth() = CV_64F, ddepth = -1/CV_64F

Gradient Operator (7/23)

➤ Code

Syntax:

`Sobel(src, dst, ddepth, dx, dy, ksize, scale, delta, borderType)`

dx (X order) – Order of the derivative x..

dy (Y order) – Order of the derivative y.

ksize – Border mode used to extrapolate pixels outside of the image, see `BorderTypes`

scale – Optional scale factor for the computed derivative values by default.

delta – Optional delta value that is added to the results prior to storing them in `dst`.

borderType – Pixel extrapolation method (see `borderInterpolate()` for details).

Gradient Operator (8/23)

➤ Sobel

X order, Y order – Orders of the derivative in x-and y-directions.

It can be 0, 1, and at most 2.

A 0 value indicates no derivative in that direction.

Sample code for sobelX and sobelY:
`Sobel(image, sobelX, CV_16S, 1, 0);`
`Sobel(image, sobelY, CV_16S, 0, 1);`

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

Sobel *y*

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

Sobel *x*

If you want to use this operator to approximate the 1st derivative in Y direction, set X order=0, and Y order=1.

Gradient Operator (9/23)

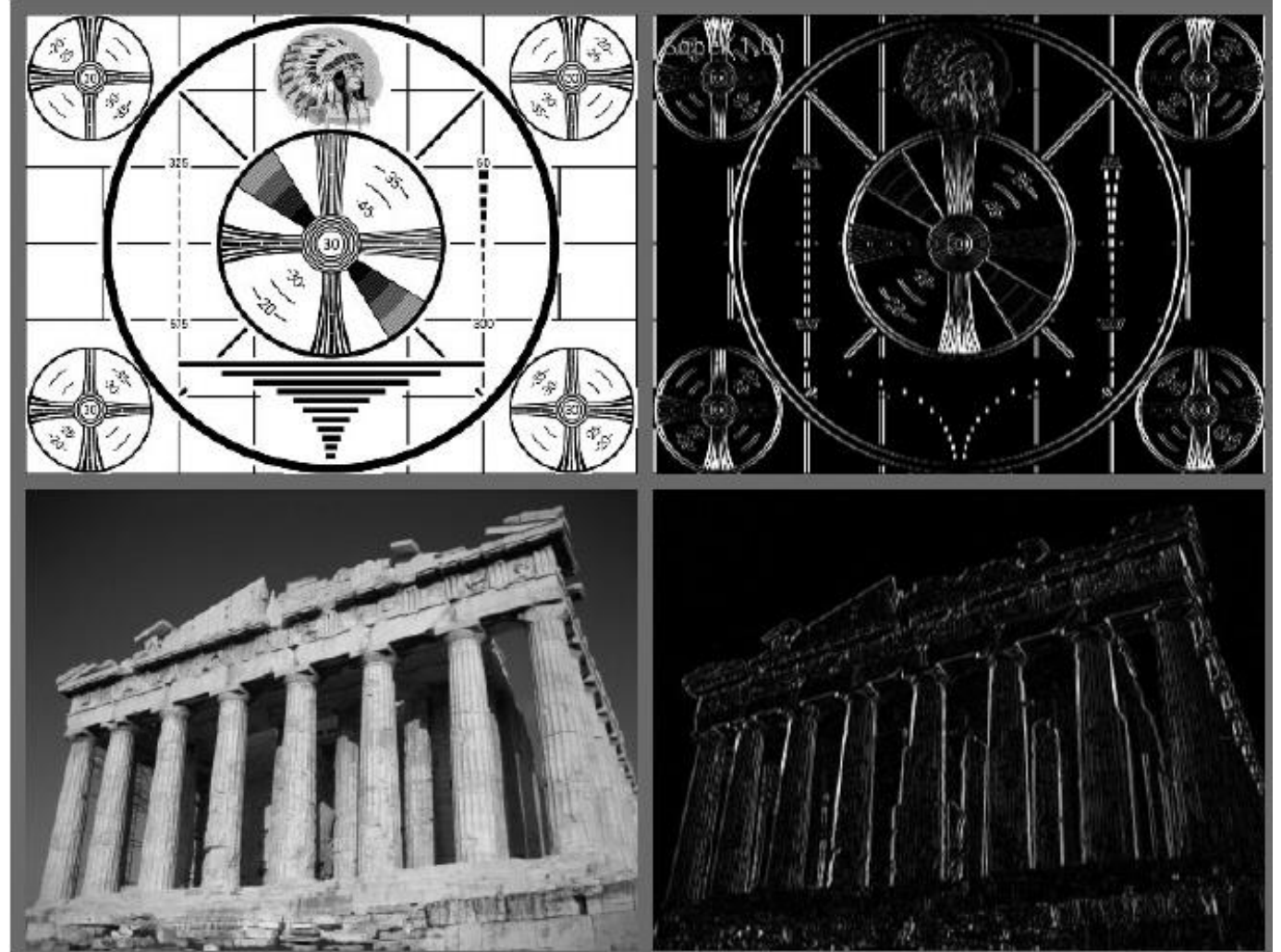
➤ Sobel

- The effect of the Sobel operator when used to approximate a **first derivative** in the **x -dimension**

`Sobel(image, sobelX, CV_16S, 1, 0);`

X order = 1
Y order = 0

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



Gradient Operator (10/23)

➤ Sobel

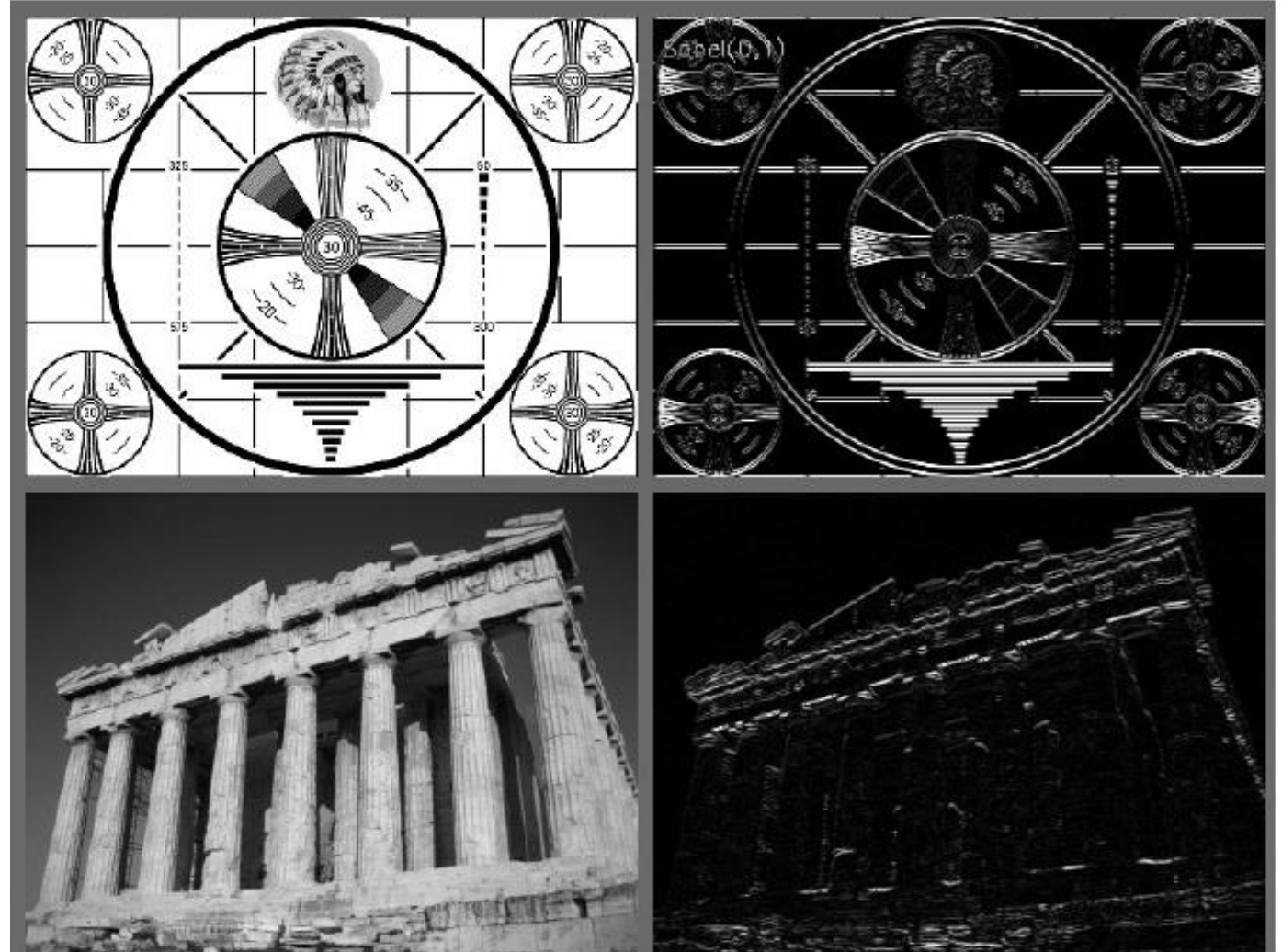
- The effect of the Sobel operator when used to approximate a **first derivative** in the **y-dimension**

`Sobel(image, sobelX, CV_16S, 1, 0);`

X order = 0

Y order = 1

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



Gradient Operator (11/23)

➤ Gradient magnitude and direction

Theoretically, gradient magnitude should be computed by L2 norm (Euclidean distance):

$$M(x, y) = \|\nabla f(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)}$$

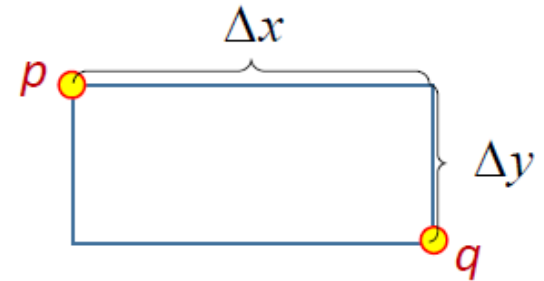
Sobel(image, **sobelX**, CV_16S, 1, 0);

Sobel(image, **sobelY**, CV_16S, 0, 1);

Due to computational efficiency, gradient magnitude can be approximated by L1 norm (Manhattan distance or city-block distance):

$$M(x, y) \approx |g_x(x, y)| + |g_y(x, y)|$$

sobel = abs(sobelX)+abs(sobelY); ➤ We can use this code to obtain a gradient image



Euclidean distance

$$D_E(p, q) = \sqrt{\Delta x^2 + \Delta y^2}$$

Manhattan distance

$$D_M(p, q) = \Delta x + \Delta y$$

Gradient Operator (12/23)

➤ Demo

```
Sobel(image,sobelX,CV_16S,1,0);  
Sobel(image,sobelY,CV_16S,0,1);  
sobel = abs(sobelX)+abs(sobelY);
```



Original Image

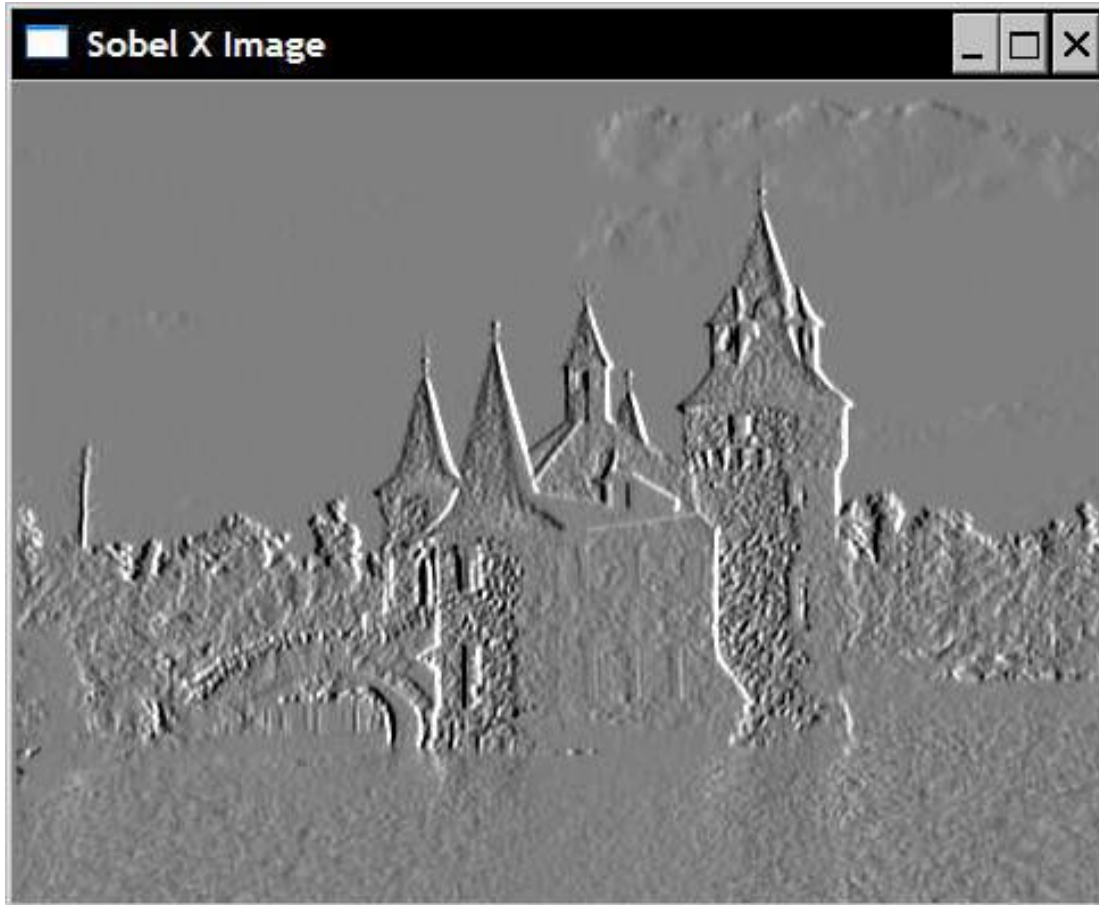


Gradient magnitude image

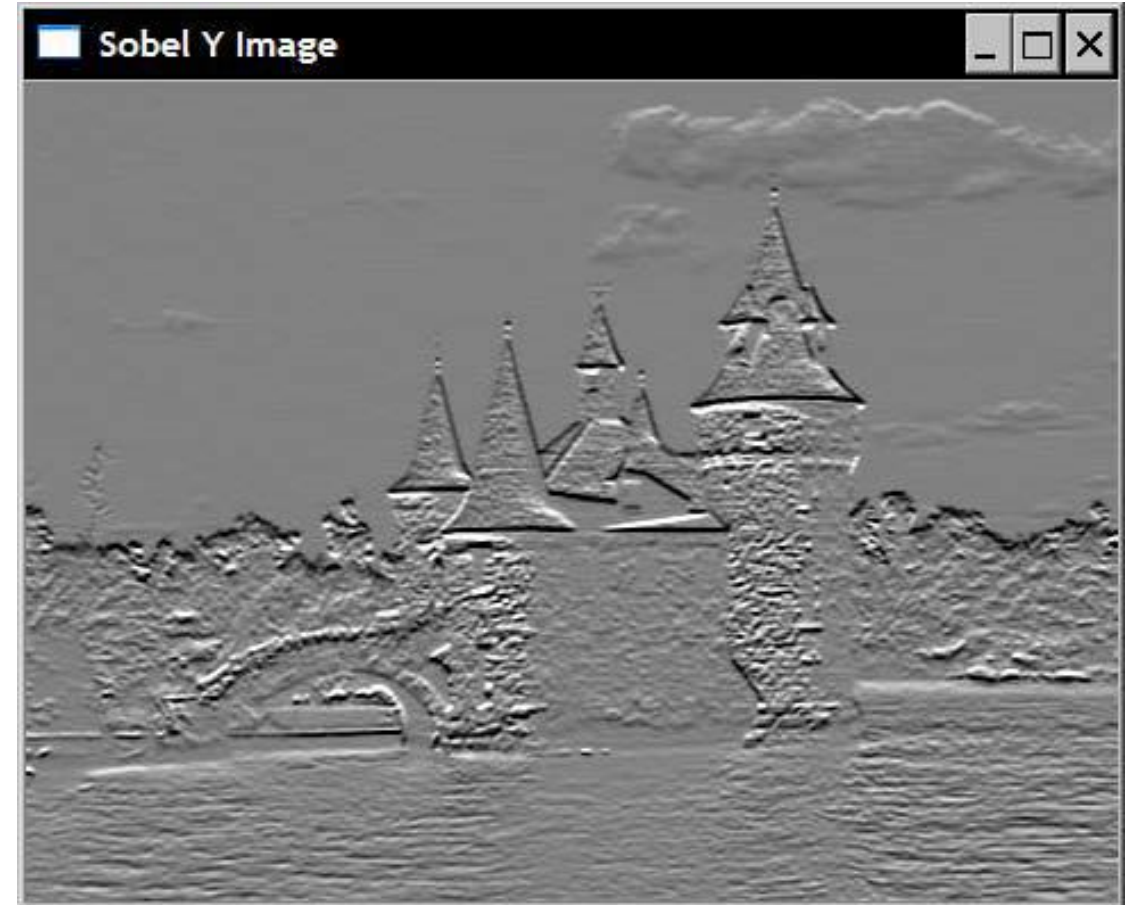
Gradient Operator (13/23)

➤ Demo

```
Sobel(image,sobelX,CV_16S,1,0);  
Sobel(image,sobelY,CV_16S,0,1);  
sobel = abs(sobelX)+abs(sobelY);
```



Sobel X image



Sobel Y image

Gradient Operator (14/23)

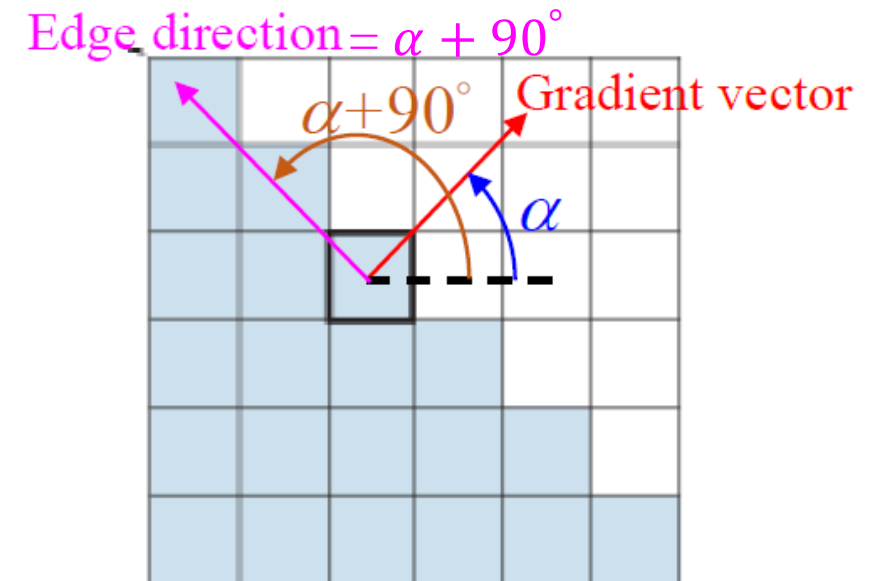
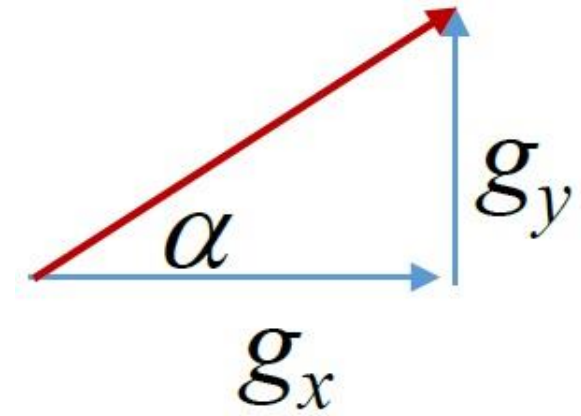
➤ Gradient magnitude and direction

$$\alpha(x, y) = \tan^{-1} \left(\frac{g_y(x, y)}{g_x(x, y)} \right)$$

Sobel(image, **sobelY**, CV_16S, 0, 1);

Sobel(image, **sobelX**, CV_16S, 1, 0);

Similarly, we can use **Sobel** operators to estimate the **gradient direction** and edge direction of each pixel.



Gradient Operator (15/23)

➤ Gradient magnitude image



Original Image



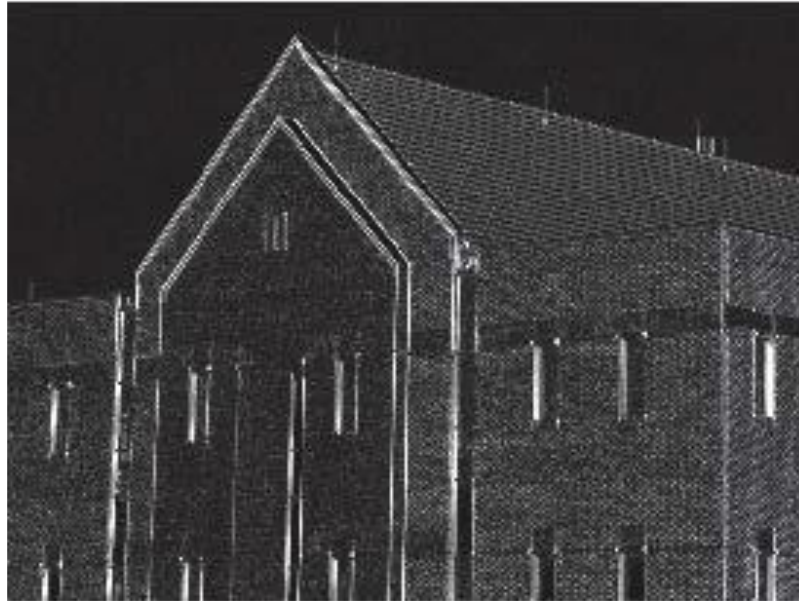
$$M(x, y) = |g_x| + |g_y|$$

Gradient Operator (16/23)

➤ Gradient magnitude image



Original Image



$\text{abs}(\text{sobel } X)$



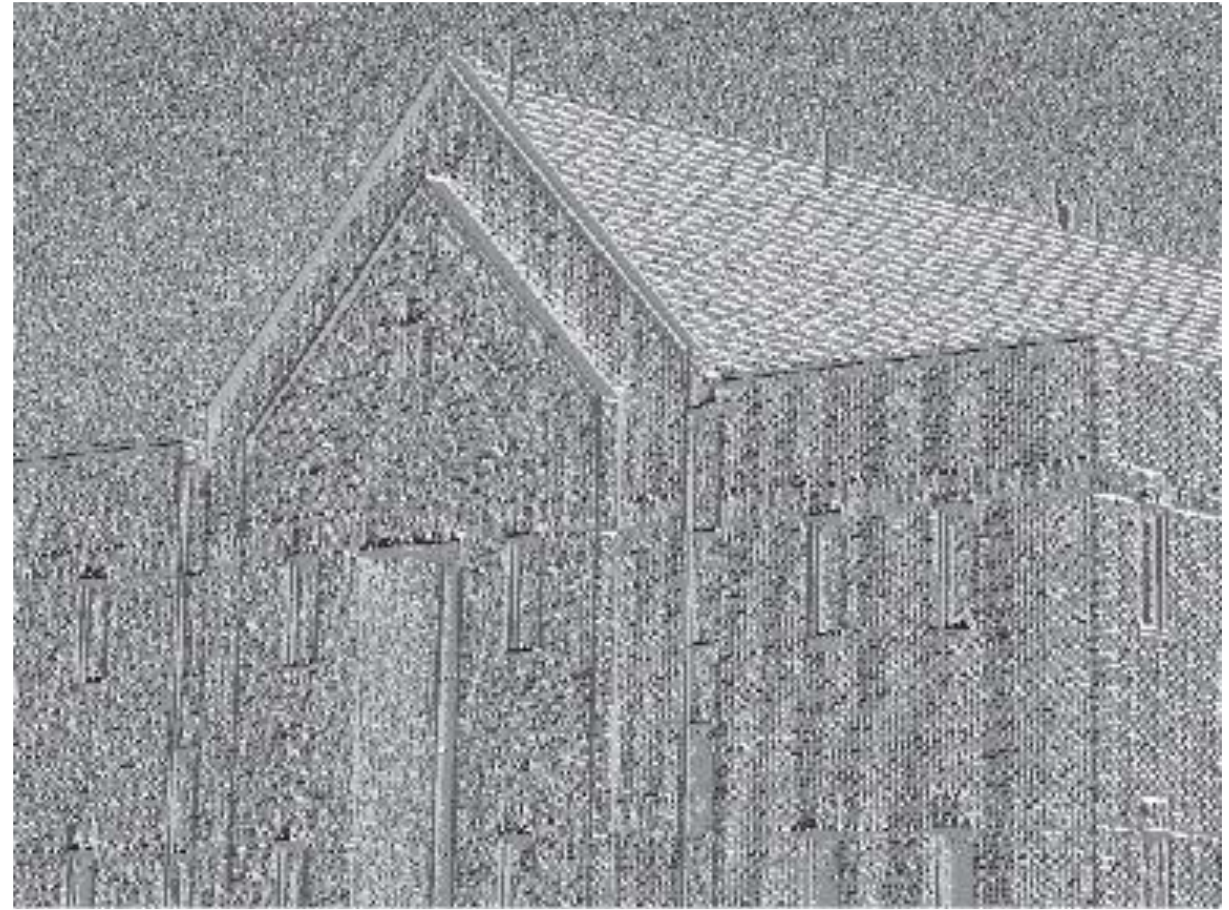
$\text{abs}(\text{sobel } Y)$

Gradient Operator (17/23)

- **Gradient magnitude image** ✓ In general, angle images are not as useful as gradient magnitude images for edge detection.



Original Image



Gradient angle image

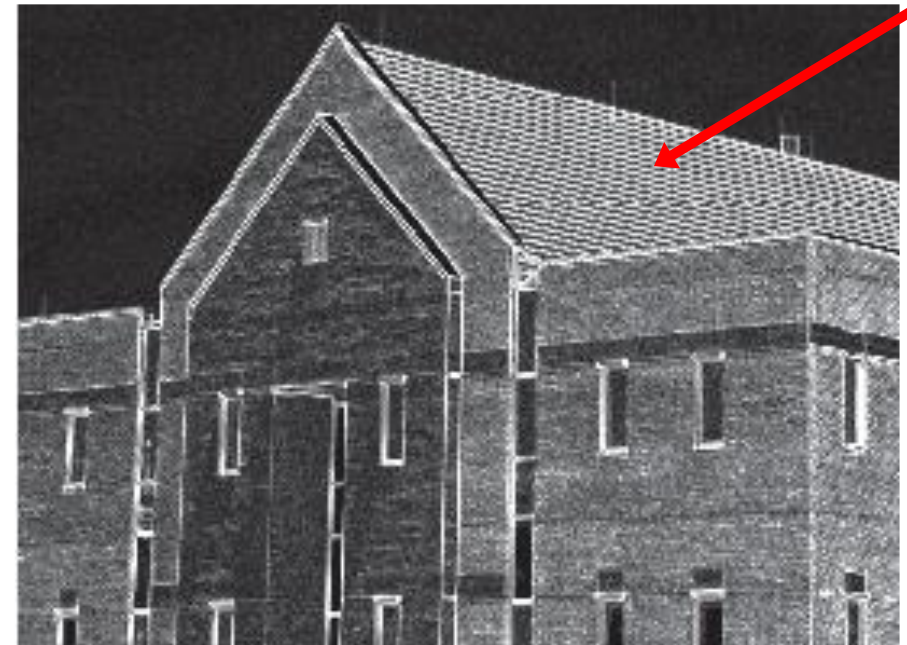
Gradient Operator (18/23)

➤ Gradient magnitude image

In this example, the original image is of reasonably high resolution.

At the distance the image was acquired, the contribution made to image detail by the wall bricks is significant.

One way to reduce fine detail is to smooth the image prior to computing the edges.



Bricks

Gradient Operator (19/23)

- Smoothed by a 5×5 average kernel



$$M(x, y) = |g_x| + |g_y|$$

Gradient Operator (20/23)

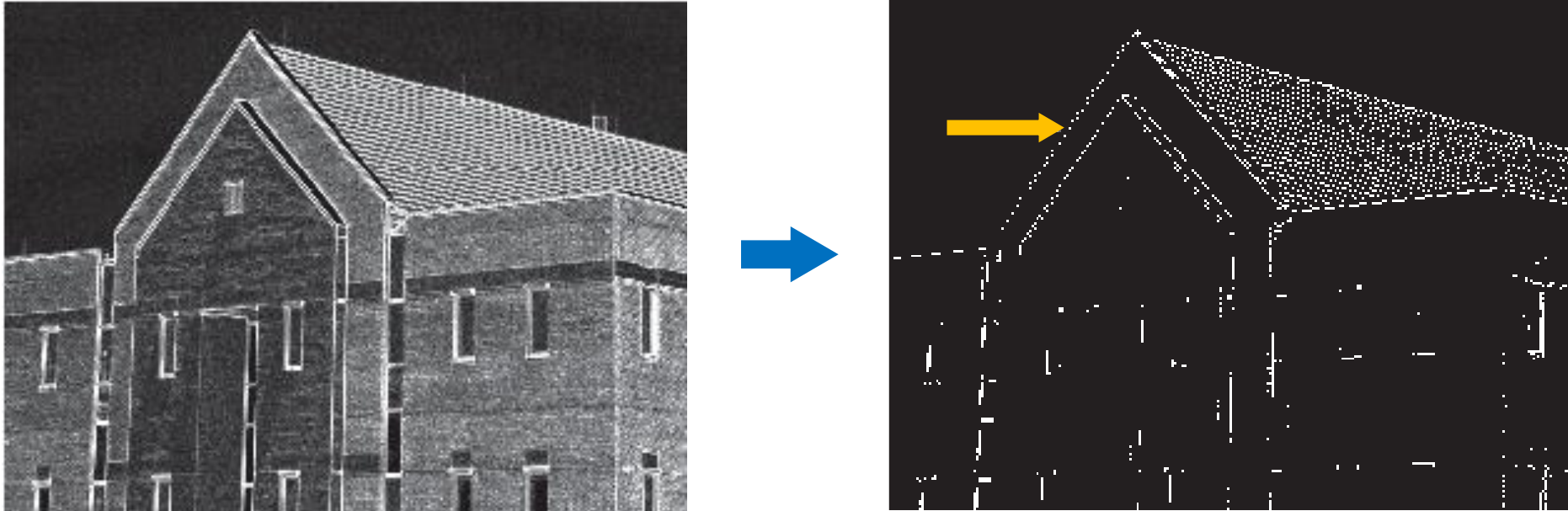
➤ No smoothing – Comparison with smoothed



$$M(x, y) = |g_x| + |g_y|$$

Gradient Operator (21/23)

➤ Combining the gradient with thresholding

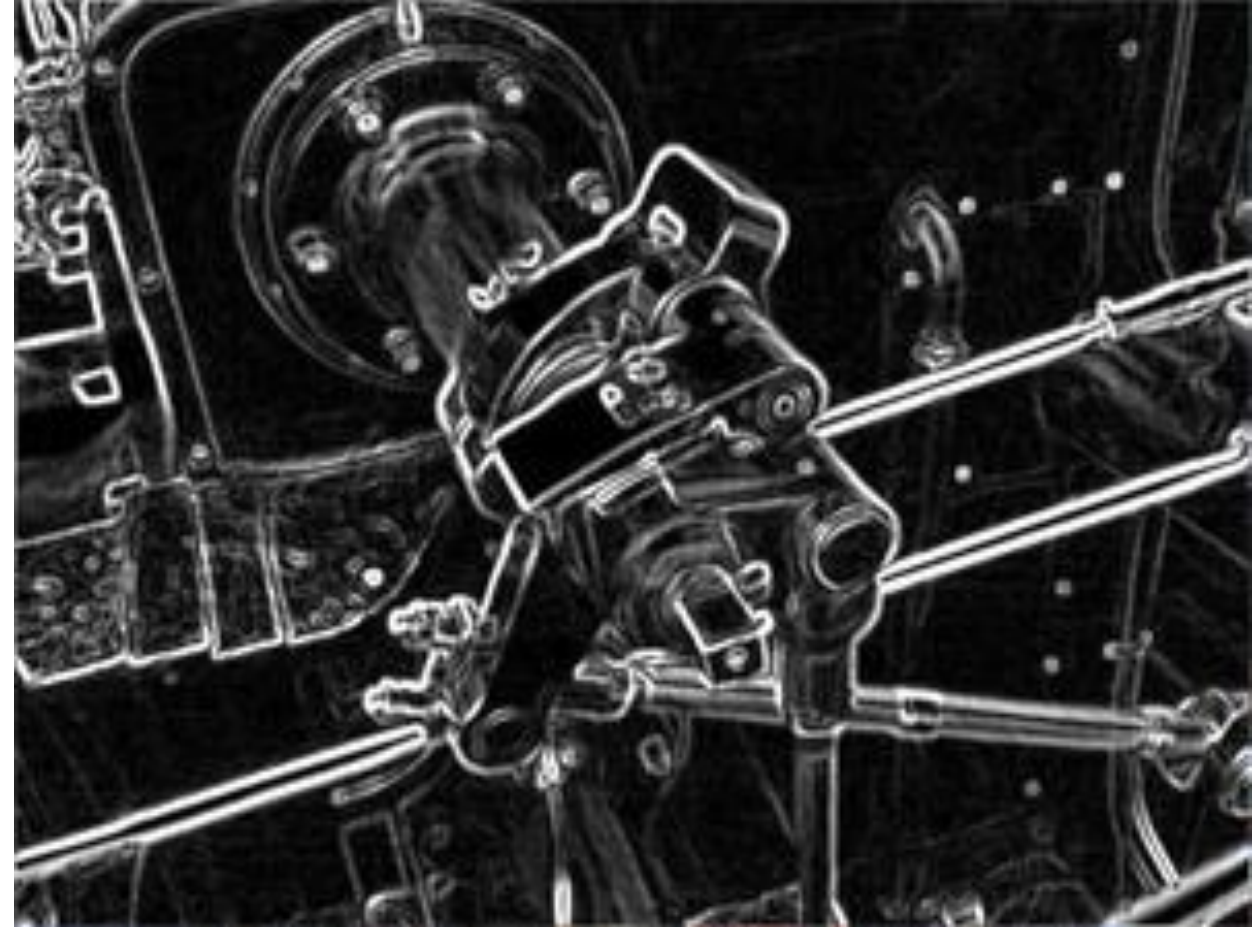


1. There are fewer edges in the threshold image.
2. Numerous edges are broken. E.g., the sloping line defining the far edge of the roof (see yellow arrow).

Gradient Operator (22/23)

➤ Scharr Filter

- ✓ Calculates the first x- or y- image derivative using Scharr operator.
- ✓ The downside of the approximation used for the Sobel operator is that it is less accurate for small kernels.
 - For large kernels, where more points are used in the approximation, this problem is less significant.
- ✓ The difficulty arises when you want to make image measurements that are approximations of *directional derivatives*.



Source:

https://upload.wikimedia.org/wikipedia/commons/thumb/d/d4/Valve_sobel_%283%29.PNG/300px-Valve_sobel_%283%29.PNG

Gradient Operator (23/23)

➤ Code

Syntax:

`Scharr(src, dst, ddepth, dx, dy, scale, delta=0, borderType)`

src – Input image.

dst – Output image of the same size and the same number of channels as src.

ddepth – output image depth (see Sobel() for the list of supported combination of src.depth() and ddepth).

scale – Optional scale factor for the computed derivative values by default.

delta – Optional delta value that is added to the results prior to storing them in dst.

borderType – Pixel extrapolation method (see borderInterpolate() for details).

Exercise #5

**Please make a comparison between the sobel and scharr.
(after smoothing by Gaussian)**

1. Sobel

➤ **LibreStock**

2. Scharr

➤ **Pixabay**

✓ **Note:** **.sln & *.ppt(or *.pptx)* are necessary and compress in a *.rar file.

Thanks!

Any questions?