

# Computer Vision and Deep Learning

Ch 04

강명묵

# Contents

**1. Edge Detection**

**2. Gradient( $\nabla$ ) as Edge Detection**

**3. Laplacian( $\nabla^2$ ) as Edge Detection**

**4. Canny Edge Detection**

# Edge Detection

**Edge Detection** : 윤곽선을 검출하는 기술로써, 영상에서 객체의 경계를 찾기 위한 기법

**Edge Operator** : 이미지에서 **Edge Position, magnitude, orientation**을 추출하는데 활용된다.

## Three performance criteria in Canny Edge Detection

### 1. Good Detection

Edge point의 marking 실패율을 낮출수록 좋다.  
(SNR을 올릴수록 좋음)

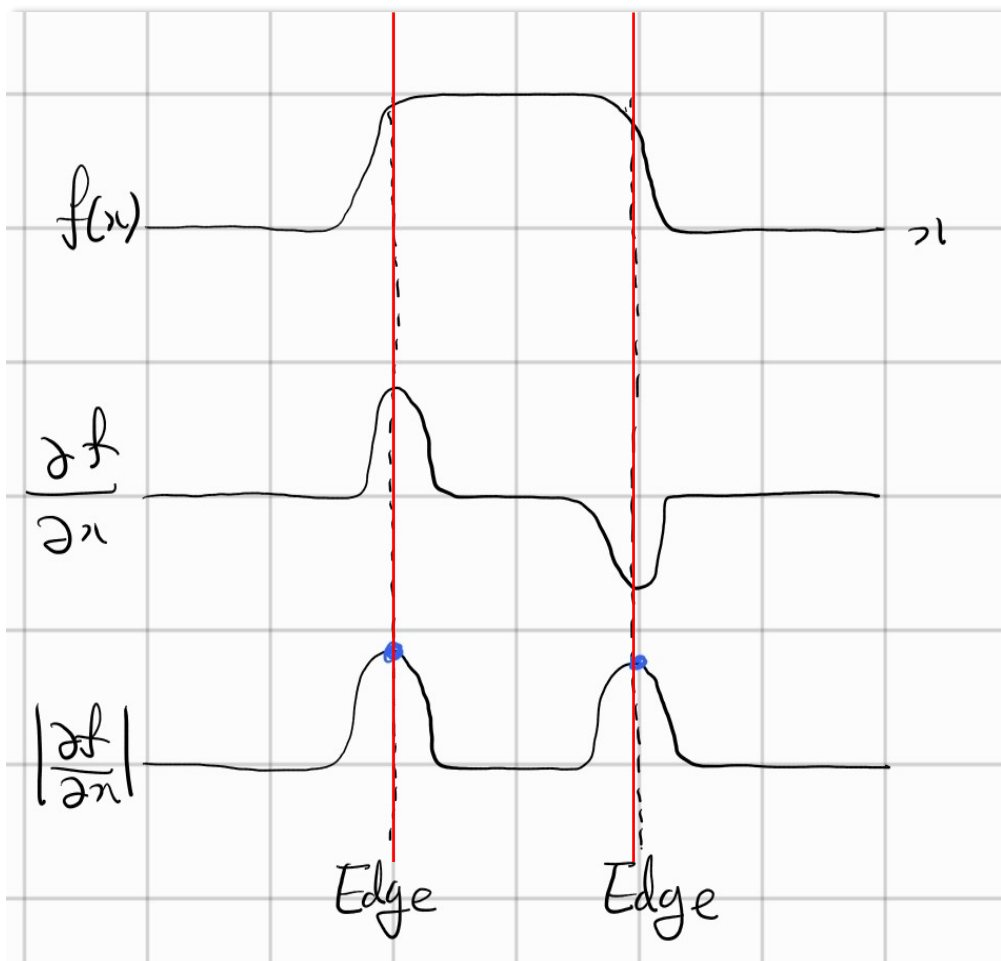
### 2. Good Localization

Edge로 판단한 지점과 실제 Edge의 중심이 가까울 수록 좋다.

### 3. Only one response to a single Edge

Edge에 대해 단 하나의 응답을 해야 한다.  
(NMS 적용)

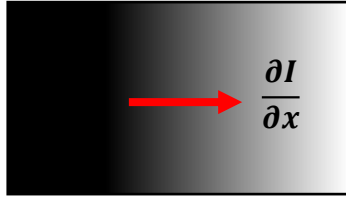
# 1D Edge Detection



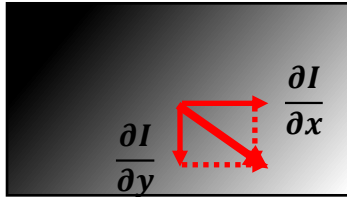
## Gradient( $\nabla$ )

1. Edge의 크기와 방향, 위치를 얻어낼 수 있다.
2. Edge 검출에 용이한 Threshold를 찾아야 한다.
3. Convolution을 두 번 수행한다.  
x축, y축
4. non-linear operator이다.  
각 픽셀에 대한 미분 값이 픽셀 값 자체에 의존하기 때문

# 2D Edge Detection



$$\nabla I = \left[ \frac{\partial I}{\partial x}, 0 \right]$$



$$\nabla I = \left[ \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right]$$

**Gradient Magnitude**

$$S = \|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

**Gradient Orientation**

$$\theta = \tan^{-1}\left(\frac{\partial I}{\partial y} / \frac{\partial I}{\partial x}\right)$$

# Discrete Gradient( $\nabla$ ) Operator

$$f'(x) = \frac{\lim_{\partial x \rightarrow 0} \frac{f(x+\partial x) - f(x)}{\partial x} + \lim_{\partial x \rightarrow 0} \frac{f(x) - f(x-\partial x)}{\partial x}}{2}$$

↳ 영상처리에서는 Discrete!

$$\Rightarrow f'(x) = \frac{f(x+1) - f(x) + f(x) - f(x-1)}{2} = \frac{1}{2} (f(x+1) - f(x-1))$$

⇒ 픽셀간의 상대적인 기운기가 중요한 것이지 기운기의 크기가 중요한 것은 아님.

⇒  $f(x+1) - f(x-1)$ 을 구현하는 필터 ⇒

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

1<sup>nd</sup> derivative

# Discrete Gradient( $\nabla$ ) Operator

-1	0
0	1

**Roberts\_x**

↕ Orthogonal!

0	-1
1	0

**Roberts\_y**

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

**Prewitt\_x**

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

**Prewitt\_y**

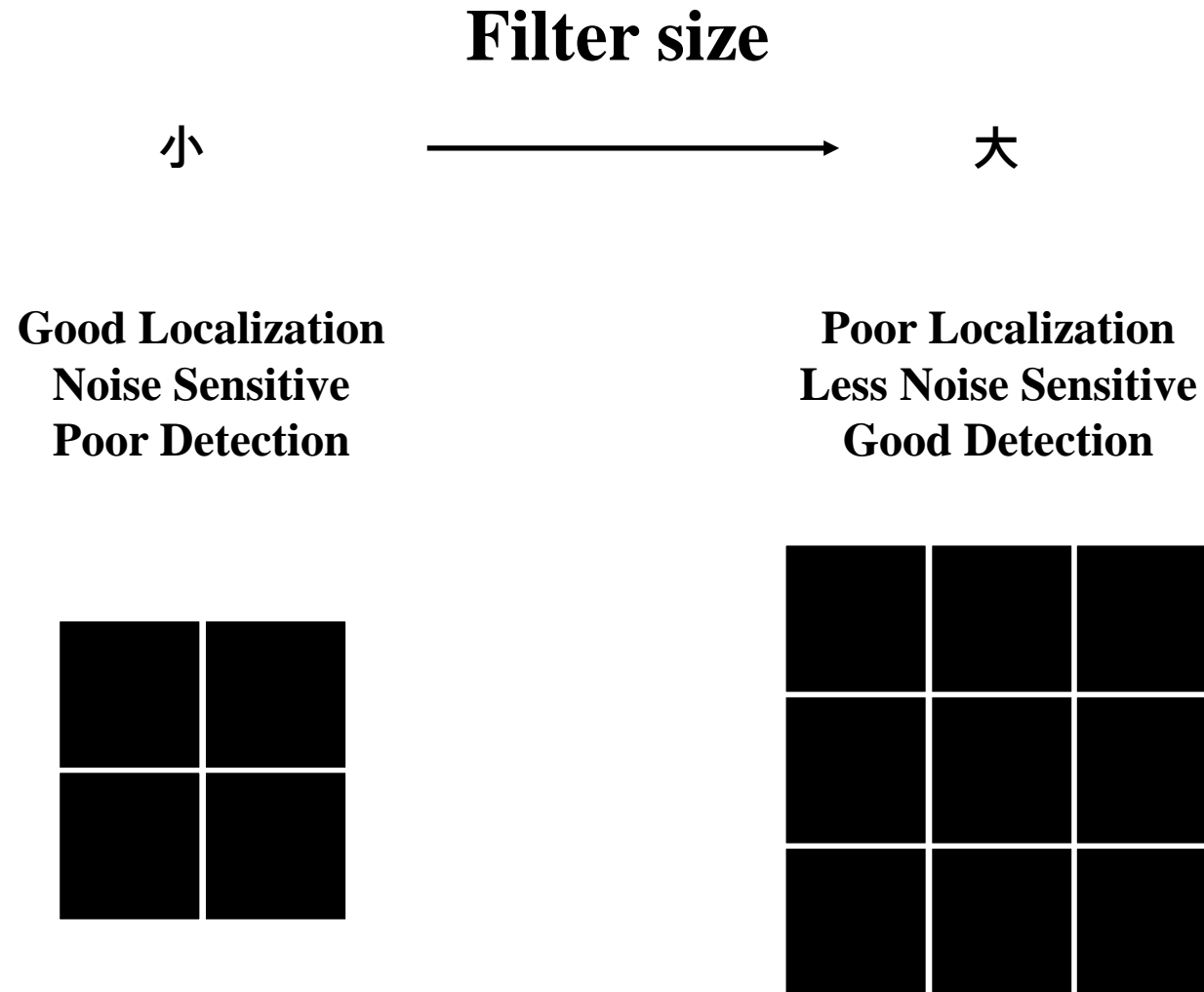
-1	0	1
-2	0	2
-1	0	1

**Sobel\_x**

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

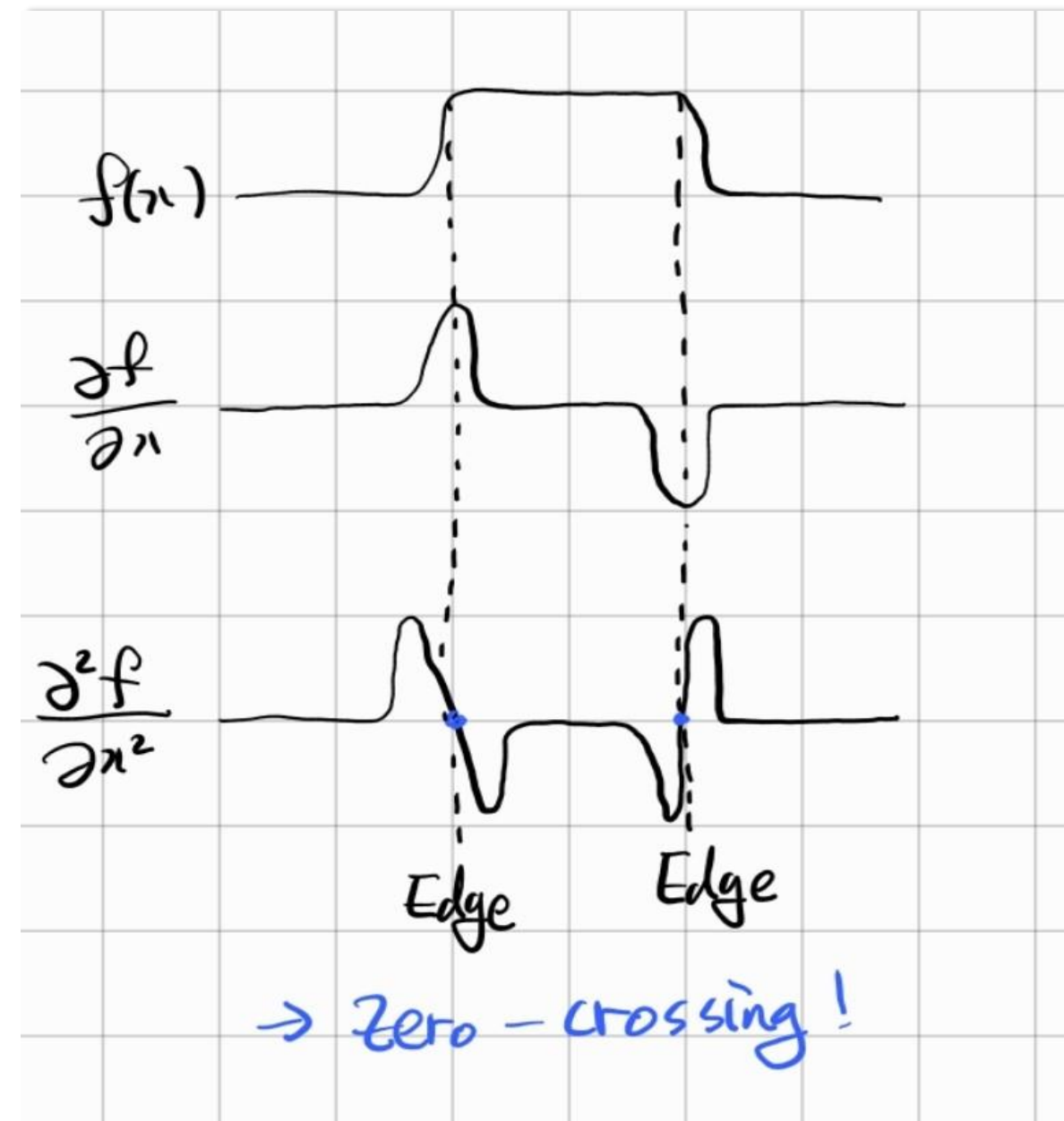
**Sobel\_y**

# Discrete Gradient( $\nabla$ ) Operator





# Discrete Laplacian( $\nabla^2$ ) Operator



## Laplacian( $\nabla^2$ )

1. Edge의 위치를 얻어낼 수 있다.

### 2. Zero-Crossing

모든 기울기의 합이 0이다.

3. Convolution을 한 번 수행한다.

### 4. Linear-operator이다.

Laplacian은 주변 픽셀에 대한 합으로 나타낼 수 있고, 이 합의 계수는 고정된 상수이기 때문

# Discrete Laplacian( $\nabla^2$ ) Operator

$$f'(x) = \frac{f(x+\delta) - f(x)}{\delta} = \frac{f(x+1) - f(x)}{\delta}$$

→ 차분.

$$f''(x) = \frac{f'(x) - f'(x-\delta)}{\delta} = f'(x) - f'(x-1)$$

$$= f(x+1) - 2f(x) + f(x-1)$$

filter →

1	-2	1
---	----	---

2<sup>nd</sup> derivative

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

0	0	0
1	-2	1
0	0	0

+

0	1	0
0	-2	0
0	1	0

$\nabla^2 =$

0	1	0
1	-4	1
0	1	0

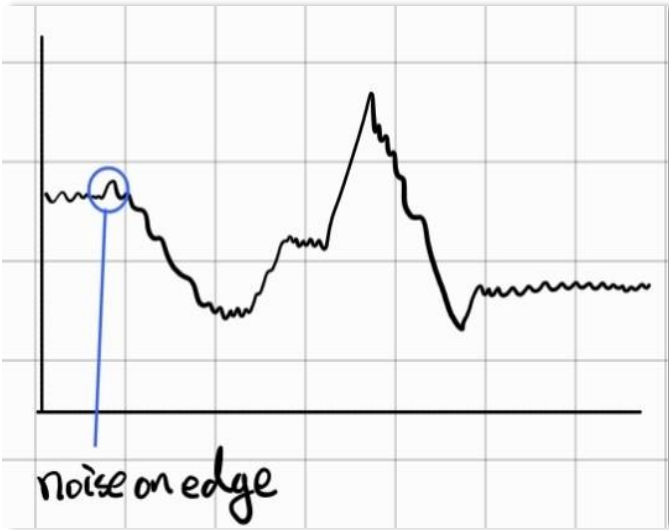


정확성 ↑

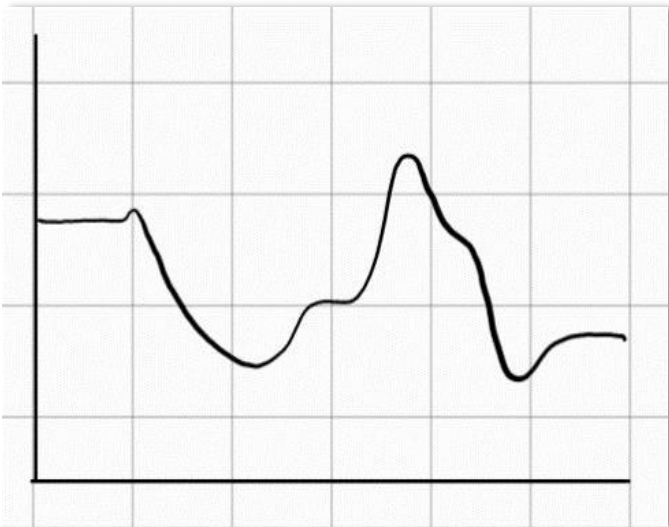
$\nabla^2 = \frac{1}{6}$

1	4	1
4	-20	4
1	4	1

# Operator Application

 $f$ 

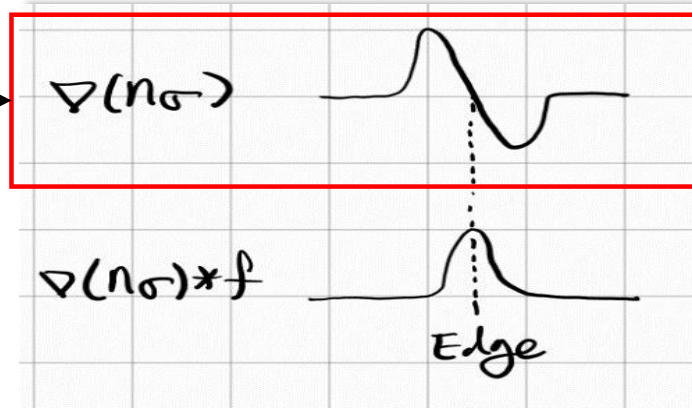
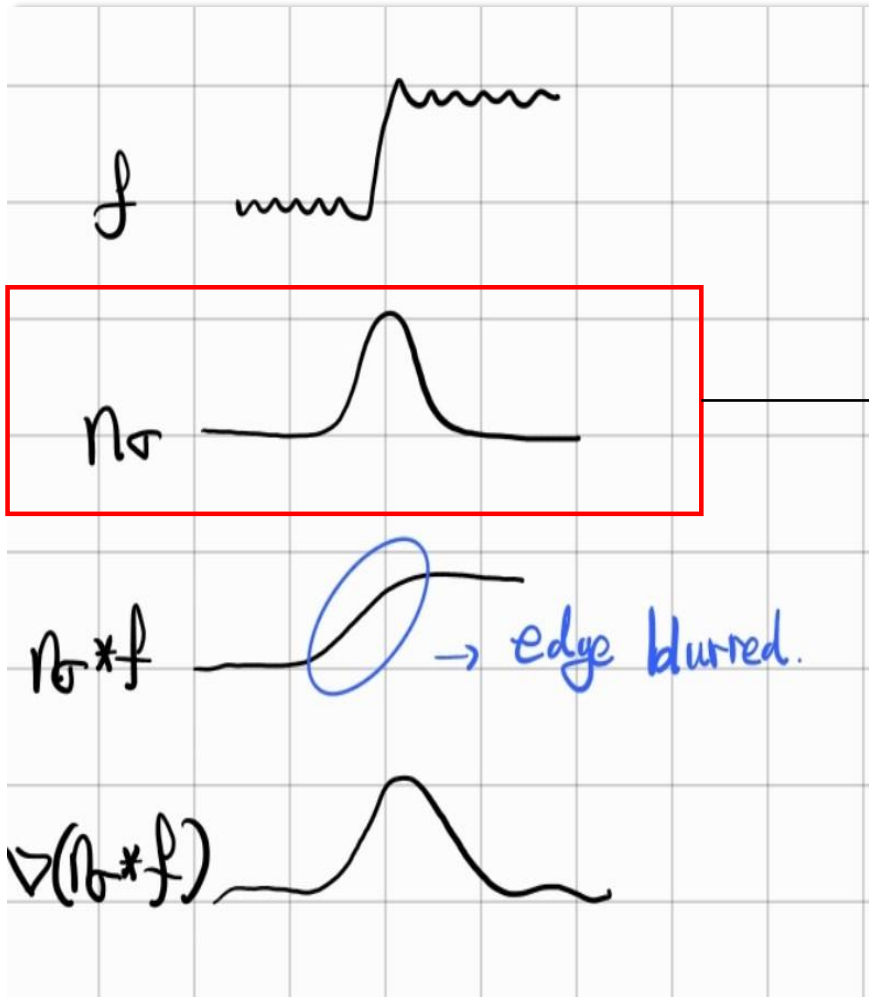
Noise때문에 어디서든 기울기가 빠르게 변화한다.

 $n_{\sigma} * f$ 

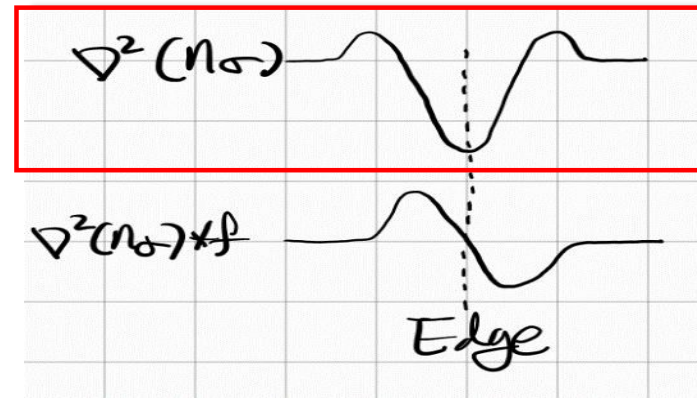
가우시안 스무딩을 통해 노이즈 완화

# Operator Application

$$\nabla(n_\sigma * f) = \nabla(n_\sigma) * f$$



$$\nabla^2(n_\sigma * f) = \nabla^2(n_\sigma) * f$$



# Canny Edge Detector

## Canny Edge Detector

최소 오류율, 위치 정확도, 한 두께라는 기준에 따라 목적 함수를 정의한 에지 검출 최적화 기법

## Non-Maximum Suppression

한 두께 에지를 출력하기 위한 기법이다.

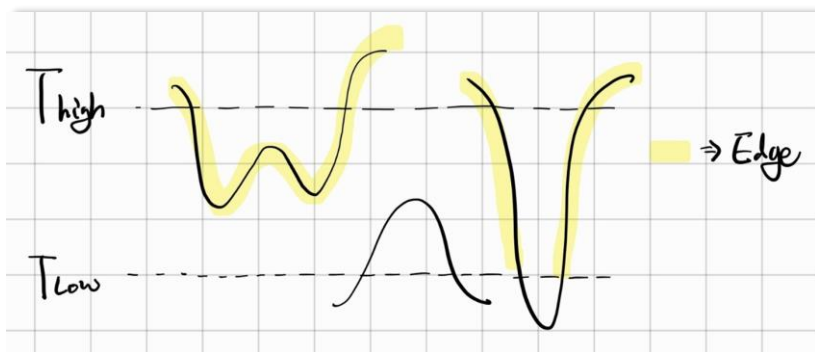
에지 방향에 수직인 두 이웃 화소의 에지보다 크면 에지로 살아남고, 그렇지 않으면 억제한다.

## Hysteresis thresholding

$\|\nabla I(x, y)\| < T_{Low}$  : not Edge

$\|\nabla I(x, y)\| > T_{high}$  : Edge

Then,  $T_{Low} < \|\nabla I(x, y)\| < T_{high}$  ?



# Canny Edge Detector

## Algorithm

### 1. $n_\sigma * I$

2D Gaussian 필터를 이용하여 이미지를 **스무딩**

### 2. $\nabla n_\sigma * I$

**Sobel** 연산자를 이용하여 **Gradient**를 구한다.

Gradient **magnitude** :  $\|\nabla n_\sigma * I\|$

Gradient **orientation** :  $\hat{n} = \frac{\nabla n_\sigma * I}{\|\nabla n_\sigma * I\|}$

### 3. Non-maximum suppression

두꺼운 에지 제거

### 4. Hysteresis thresholding

최종 에지 선정 ( $T_{high}$ 는  $T_{low}$ 의 2~3배 설정)

# Canny Edge Detector

## Step 1.

2D Gaussian 필터를 이용하여 이미지를 스무딩( $n_{\sigma} * I$ )

```
Mat Gaussian_Kernel(int size, float sigma) {  
    Mat kernel(size, size, CV_32F);  
  
    int kernelCenter = (size - 1) / 2;  
    float sum = 0.0;  
  
    for (int i = 0; i < size; i++) {  
        for (int j = 0; j < size; j++) {  
            int x = kernelCenter - i;  
            int y = kernelCenter - j;  
            kernel.at<float>(i, j) = exp(-(static_cast<float>(x * x) +  
                static_cast<float>(y * y)) / (2.0 * sigma * sigma)) / sqrt(2.0 * PI * sigma * sigma);  
            sum += kernel.at<float>(i, j);  
        }  
    }  
  
    kernel /= sum;  
    return kernel;  
}
```

```
Mat conv(Mat& img, Mat& kernel) {  
    int kernelCenter = (kernel.rows - 1) / 2;  
    Mat img_conv(img.cols - 2 * kernelCenter, img.rows - 2 * kernelCenter, CV_32F);  
  
    for (int i = kernelCenter; i < img.rows - kernelCenter; i++) {  
        for (int j = kernelCenter; j < img.cols - kernelCenter; j++) {  
            float value = 0;  
            for (int m = 0; m < kernel.rows; m++) {  
                for (int n = 0; n < kernel.cols; n++) {  
                    value += static_cast<float>(img.at<uchar>(i - kernelCenter + m,  
                        j - kernelCenter + n)) * kernel.at<float>(m, n);  
                }  
            }  
            img_conv.at<float>(i - kernelCenter, j - kernelCenter) = value;  
        }  
    }  
    return img_conv;  
}
```



Gaussian filter  $\sigma = 1$ , size = 3x3

# Canny Edge Detector

## Step 2.

**Sobel** 연산자를 이용하여 **Gradient**를 구한다.

Gradient **magnitude** :  $\|\nabla n_{\sigma} * \mathbf{I}\|$

Gradient **orientation** :  $\hat{n} = \frac{\nabla n_{\sigma} * \mathbf{I}}{\|\nabla n_{\sigma} * \mathbf{I}\|}$

```
Mat Sobel_operator(int x) {  
    Mat sobelKernelX = (Mat_<float>(3, 3) << -1, 0, 1, -2, 0, 2, -1, 0, 1);  
    Mat sobelKernelY = (Mat_<float>(3, 3) << -1, -2, -1, 0, 0, 0, 1, 2, 1);  
    if (x == 1) return sobelKernelX;  
    else return sobelKernelY;  
}
```

**Sobel\_x**

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

**Sobel\_y**

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

**Sobel\_combined**

```
Mat weight_x_y(Mat& img_x, Mat& img_y, float weight) {  
    Mat img_w = Mat::zeros(img_x.rows, img_x.cols, CV_32F);  
    for (int x = 0; x < img_x.cols; ++x) {  
        for (int y = 0; y < img_y.rows; ++y) {  
            img_w.at<float>(y, x) = img_x.at<float>(y, x) * weight + img_y.at<float>(y, x) * (1 - weight);  
        }  
    }  
    return img_w;  
}
```

**Gradient의  
크기와 방향 계산**

```
Mat calc_gradient_magnitude_orientation(Mat& img_x, Mat& img_y, int k) {  
    Mat mag = Mat::zeros(img_x.rows, img_x.cols, CV_32F);  
    Mat orient = Mat::zeros(img_x.rows, img_x.cols, CV_32F);  
  
    for (int x = 0; x < img_x.cols; ++x) {  
        for (int y = 0; y < img_y.rows; ++y) {  
            mag.at<float>(y, x) = sqrt(img_x.at<float>(y, x) * img_x.at<float>(y, x) + img_y.at<float>(y, x) * img_y.at<float>(y, x));  
            float a = 10 * atan2(img_y.at<float>(y, x), img_x.at<float>(y, x));  
            if (a < 0) a = -a;  
            orient.at<float>(y, x) = a;  
        }  
    }  
    if (k == 1) return mag;  
    else return orient;  
}
```



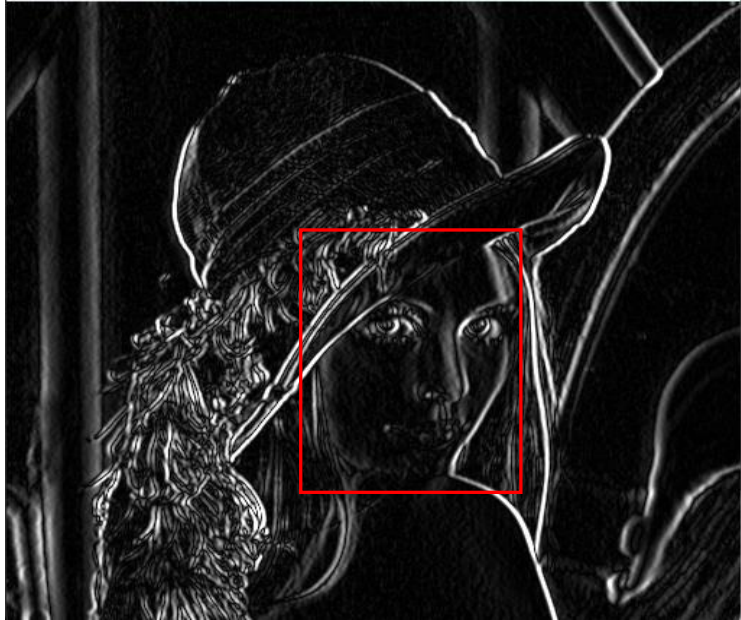
# Canny Edge Detector

## Step 2.

Sobel 연산자를 이용하여 **Gradient**를 구한다.

Gradient **magnitude** :  $\|\nabla n_{\sigma} * I\|$

Gradient **orientation** :  $\hat{n} = \frac{\nabla n_{\sigma} * I}{\|\nabla n_{\sigma} * I\|}$



Sobel\_x\_conv



Sobel\_y\_conv



Sobel\_combined  
w = 0.5

# Canny Edge Detector

## Step 2.

Sobel 연산자를 이용하여 **Gradient**를 구한다.

Gradient **magnitude** :  $\|\nabla n_{\sigma} * \mathbf{I}\|$

Gradient **orientation** :  $\hat{n} = \frac{\nabla n_{\sigma} * \mathbf{I}}{\|\nabla n_{\sigma} * \mathbf{I}\|}$



**Gradient magnitude**



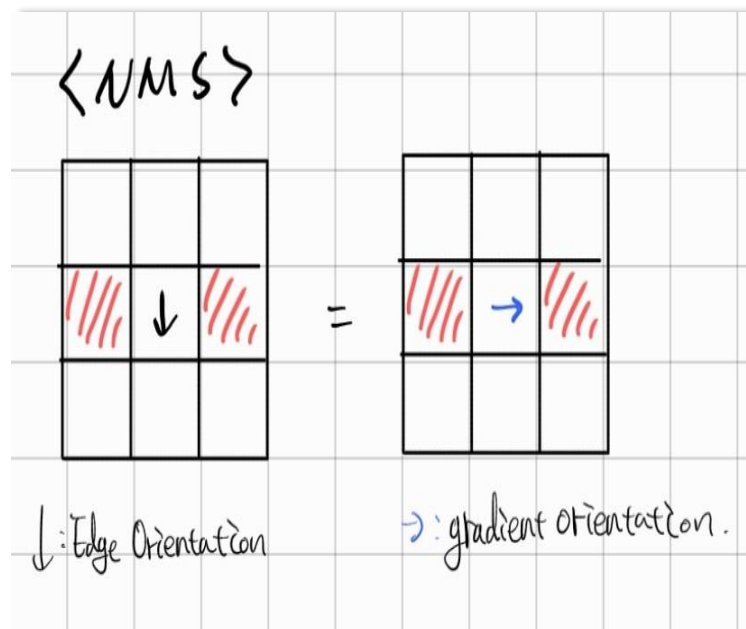
**Gradient orientation**

# Canny Edge Detector

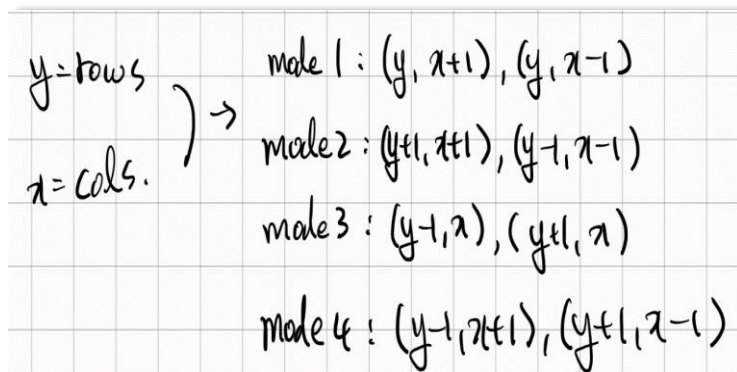
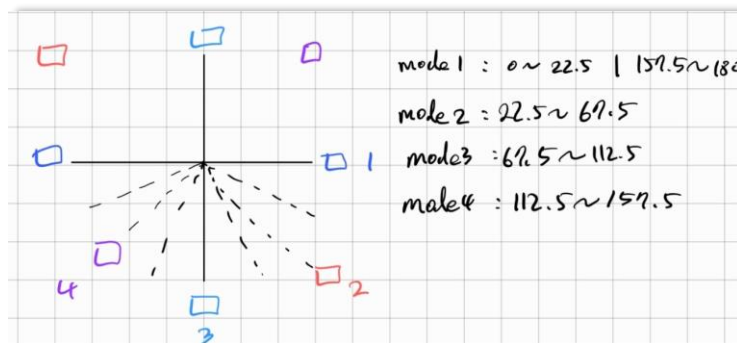
## Step 3.

### Non-maximum suppression

두꺼운 에지 제거



Criteria : Gradient orientation



1.  $\text{Atan2} \rightarrow -\pi \sim \pi$

2. 각도의 절댓값 저장(  $0 \sim 180^\circ$  )

3. 각도에 따른 모드 설정

4. 모드에 따른 주변 픽셀  
고려 후 억제



# Canny Edge Detector

## Step 3.

## Non-maximum suppression

두꺼운 에지 제거



Img\_NMS

```
Mat NMS(Mat& mag, Mat& orient) {
    Mat img_NMS(mag.rows, mag.cols, CV_32F);
    Mat mag_p;
    int d = 1;
    int mode = 0;
    copyMakeBorder(mag, mag_p, d, d, d, d, BORDER_CONSTANT, 0);

    for (int x = d; x < mag_p.cols - d; ++x) {
        for (int y = d; y < mag_p.rows - d; ++y) {
            float value = orient.at<float>(y-d, x-d);

            if ((0 <= value && value < 22.5) || (157.5 <= value && value < 180)) mode = 1;
            else if (22.5 <= value && value < 67.5) mode = 2;
            else if (67.5 <= value && value < 112.5) mode = 3;
            else mode = 4;

            switch (mode) {
                case 1:
                    if ((mag_p.at<float>(y, x) < mag_p.at<float>(y, x-1)) || (mag_p.at<float>(y, x) < mag_p.at<float>(y, x+1))) img_NMS.at<float>(y-d, x-d) = 0;
                    else img_NMS.at<float>(y-d, x-d) = mag_p.at<float>(y, x);
                    break;
                case 2:
                    if ((mag_p.at<float>(y, x) < mag_p.at<float>(y+1, x+1)) || (mag_p.at<float>(y, x) < mag_p.at<float>(y-1, x-1))) img_NMS.at<float>(y-d, x-d) = 0;
                    else img_NMS.at<float>(y-d, x-d) = mag_p.at<float>(y, x);
                    break;
                case 3:
                    if ((mag_p.at<float>(y, x) < mag_p.at<float>(y-1, x)) || (mag_p.at<float>(y, x) < mag_p.at<float>(y+1, x))) img_NMS.at<float>(y-d, x-d) = 0;
                    else img_NMS.at<float>(y-d, x-d) = mag_p.at<float>(y, x);
                    break;
                case 4:
                    if ((mag_p.at<float>(y, x) < mag_p.at<float>(y-1, x+1)) || (mag_p.at<float>(y, x) < mag_p.at<float>(y+1, x-1))) img_NMS.at<float>(y-d, x-d) = 0;
                    else img_NMS.at<float>(y-d, x-d) = mag_p.at<float>(y, x);
                    break;
            }
        }
    }

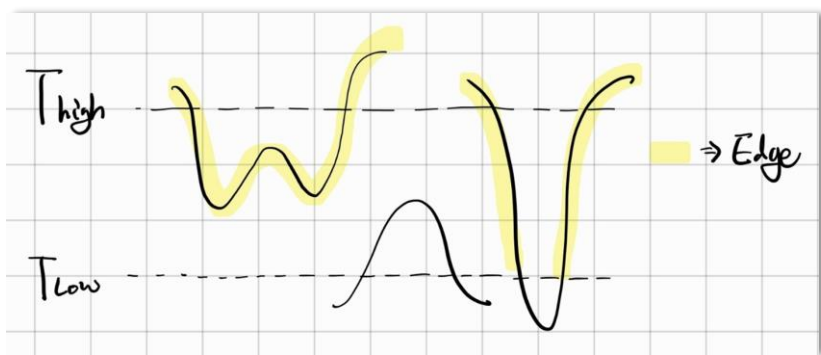
    return img_NMS;
}
```

# Canny Edge Detector

## Step 4.

### Hysteresis thresholding

최종 에지 선정 ( $T_{high}$ 는  $T_{low}$ 의 2~3배 설정)



$$T_{Low} < \|\nabla I(x, y)\| < T_{high}$$

$$\|\nabla I(x, y)\| > T_{high} : \text{Edge}$$

$$\|\nabla I(x, y)\| < T_{Low} : \text{not Edge}$$

```
Mat Hysteresis_Threshold(Mat& img_NMS, float lowThreshold, float highThreshold) {  
    Mat img_H = Mat::zeros(img_NMS.rows, img_NMS.cols, CV_8U); int d = 1;  
    Mat img_NMS_p; copyMakeBorder(img_NMS, img_NMS_p, d, d, d, d, BORDER_CONSTANT, 0);  
    int flag = 0;  
  
    for (int x = d; x < img_NMS_p.cols - d; ++x) {  
        for (int y = d; y < img_NMS_p.rows - d; ++y) {  
            if (img_NMS_p.at<float>(y, x) > highThreshold) { img_H.at<uchar>(y - d, x - d) = 255; }  
            else if (img_NMS_p.at<float>(y, x) >= lowThreshold) {  
                for (int m = -1; m <= 1; ++m) {  
                    for (int n = -1; n <= 1; ++n) {  
                        if (img_NMS_p.at<float>(y + m, x + n) >= highThreshold) {  
                            flag = true;  
                            break;  
                        }  
                    }  
                }  
                if (flag) break;  
            }  
            img_H.at<uchar>(y - d, x - d) = (flag) ? 255 : 0;  
        }  
    }  
    else {img_H.at<uchar>(y - d, x - d) = 0;}  
    flag = 0;  
}  
}  
return img_H;  
}
```

# Canny Edge Detector

Step 4.

Hysteresis thresholding

최종 에지 선정



**lowThreshold = 30**  
**HighThreshold = 60**



**lowThreshold = 30**  
**HighThreshold = 90**



**lowThreshold = 50**  
**HighThreshold = 150**

감사합니다.