Chapter 7.1
Edge Detection

Image Processing and Computer Vision

LE Thanh Sach
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

1. **Point Detection**

2. **Line Detection**

3. **Edge Detection**

4. **Laplacian of Gaussian (LoG)**

Point Detection

1. Filter the input image $f(x, y)$ with Laplacian H_{lap}, i.e., compute $g(x, y) = f(x, y) * H_{lap}(i, j)$
2. Detect isolated points (x, y) if they satisfy: $|g(x, y)| \geq T$. Where, T is a threshold value.

Laplacian kernel H_{lap}:

$$H_{lap} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Line Detection

1. Filter the input image $f(x, y)$ with all following masks for detecting horizontal, vertical, $\pm 45^0$-oriented lines. This process results $g_i(x, y), i = 1..4$. You can design new masks for other lines with new orientation.
2. Chose a orientation i for point (x, y) by selecting the largest $g_i(x, y), i = 1..4$.
3. Do thresholding with a certain T (input) to obtain lines.

Some kernels:

$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$ Horizontal	$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$ Vertical
$\begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix}$ $+45^0$	$\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix}$ -45^0

Edge Detection

Definition

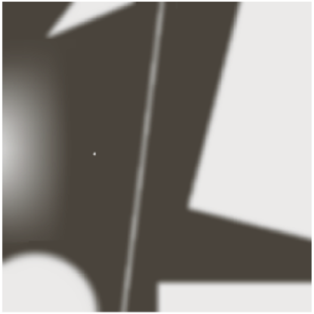
Edge is a set of connected pixels that lie on the boundary between two regions.

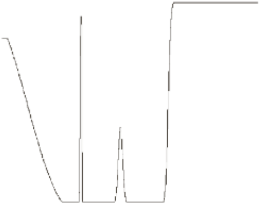
Properties

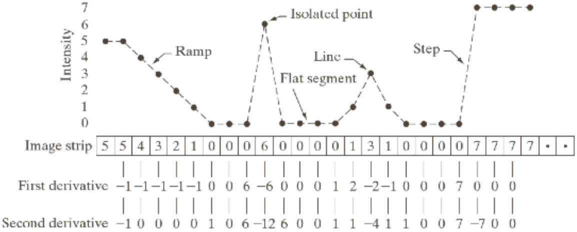
1. There is "meaningful" transitions in gray-levels at edge.
2. So, first-order and second-order derivatives can be used to detect the transition.

Edge Detection

Examples of Derivatives: image, a line profile, first and second-order derivatives.





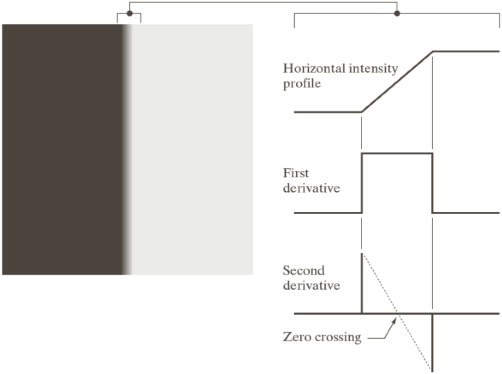


Edge Detection

Model of edges:

1. Left: **Clear edge or Ideal edge**, ideally represented as a step
2. Middle: **Blurred edge**, ideally represented as a ramp
3. Right: **A blurred bright edge**, ideally represented as a roof.

Edge Detection

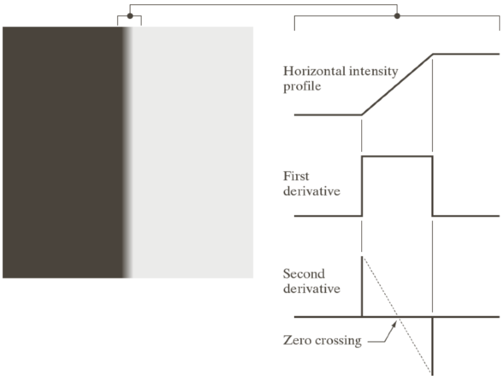


Edge with first-order derivatives

Edge consists of points where the module of the gradient vector is greater than a threshold.

- The gradient vector is **perpendicular** with the local edge passing that point

Edge Detection

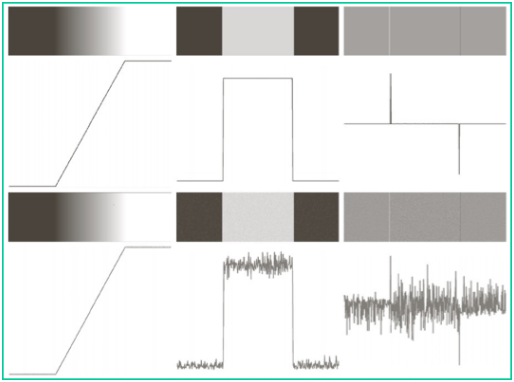


Edge with second-order derivatives

Edge consists of **zero-crossing points** in image filtered with second-order derivatives.

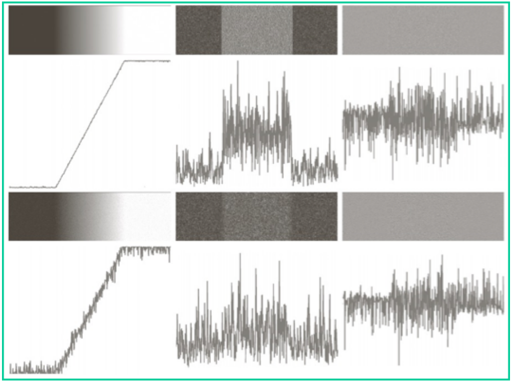
- Second-order derivatives create one **positive** response and another **negative** one for ramp edges.

Edge Detection with noise



- Rows: Row 1: no noise; Row 2: with Gaussian noise $(\mu = 0, \sigma = 0)$
- Cols: Col 1: a line profile; Col 2: **Fist-order derivative**; Col 3: **Second-order derivative**

Edge Detection with noise



- Rows: Row 1: with Gaussian noise $(\mu = 0, \sigma = 0.1)$; Row 2: with Gaussian noise $(\mu = 0, \sigma = 1.0)$
- Cols: Col 1: a line profile; Col 2: **Fist-order derivative**; Col 3: **Second-order derivative**

Edge Detection with noise

Properties

- Second-order derivative is more **sensitive to noise** compared with first-order derivative.
- However,
  - First-order derivatives provide **thick edges**
  - Second-order derivatives provide **thin edges** (via, zero-crossing)

Edge Detection and Laplacian

Question

Laplacian can provide the discontinuity in gray-levels. **Why is it not used in edge detection?**

Reasons

1. As a second-order derivative, it is unacceptably sensitive to noise
2. The magnitude of Laplacian provides double edges (one for positive and another one for negative response)
3. Laplacian can not provide edge direction

Therefore, Laplacian is directly suitable for sharpening images only.

Edge Detection and Laplacian

- Laplacian can provide thin edges via zeros-crossing detection. However, it is sensitive to noise.
- What will be happened if we remove noise before taking Laplician and then finding zeros-crossing?

Laplacian in edge detection

1. Perform noise removal will a Gaussian low-pass filter. The input image will be blurred.
2. Apply Laplacian to the resulting image.
3. Detect zero-crossing points to obtain edge points.

Laplacian of Gaussian (LoG)

Step 1 and 2 in the above algorithm is equivalent to filtering image with a LoG mask

Laplacian of Gaussian (LoG)

A Gaussian function $G(x, y)$

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- σ : standard deviation. **This parameter decides the degree of blurring in output image, if the input image is convoluted with this function**

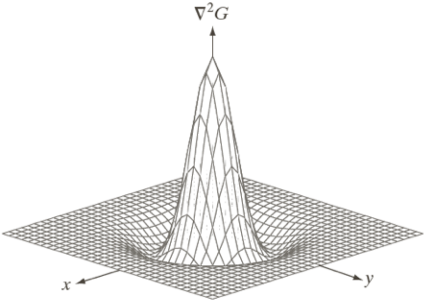
Laplacian of Gaussian (LoG)

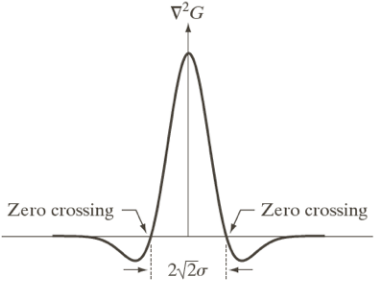
Laplacian of Gaussian (LoG)

$$\nabla^2 G(r) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}}$$

- LoG $\equiv$ Laplacian of function $G(x, y)$
- LoG $\equiv \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$

Laplacian of Gaussian (LoG)





0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

Laplacian of Gaussian (LoG)

Properties

1. Other name: Mexican hat, because of its shape
2. Zero-crossing point in LoG: $x^2 + y^2 = 2\sigma^2$
3. Radius from the origin to zero-crossing point: $r = \sqrt{2}\sigma$
4. Kernel of LoG given above: just an example. It can be approximated by any size and any coefficients.
5. **Sum of all coefficients of the kernel must be 0**

Generation of LoG's kernel

How can you generate LoG's kernel?

Laplacian of Gaussian (LoG)

Properties: Linearity

$$
\begin{aligned}
g(x, y) &= \left[\nabla^2 G(x, y)\right] * f(x, y) \\
&= \nabla^2 \left[G(x, y) * f(x, y)\right]
\end{aligned}
$$

Marr-Hildreth Algorithm

Marr-Hildreth Algorithm

1. **Filter** the input image $f(x,y)$ with Gaussian low-pass filter by kernel size $n \times n$ to obtain the output $g(x,y)$.
2. **Compute** Laplacian of $g(x,y)$ to obtain $g_L(x,y)$
3. **Find** zero-crossing points in $g_L(x,y)$

LoG

Step 1 and 2 can be implemented as applying LoG on the input image.

Marr-Hildreth Algorithm

Power of Marr-Hildreth Algorithm

Marr-Hildreth Algorithm can remedy the following problems in edge detection:

1. Intensity changes are not independent of image scale $\Rightarrow$ use different kernel' size.
2. Edges are sensitive to noise, especially true for second-order derivative $\Rightarrow$ use Gaussian low-pass filter

Questions

1. How can you obtain the kernel's size?
2. How can you detect zero-crossing points?

Marr-Hildreth Algorithm

How can you obtain the kernel's size?

- Volume of a Gaussian function inside of circle $radius = 3\sigma$ is 99.7%
- $\Rightarrow$ Kernel size $n \times n$, where n an odd numer $\geq 6\sigma$

Marr-Hildreth Algorithm

How can you detect zero-crossing points?

1. Perform thresholding of the magnitude of LoG image, i.e. $|g_l(x,y)|$, with a value T.

$$g_l(x,y) = \begin{cases} -1 & \text{if } (g_l(x,y) < 0) \text{ and } |g_l(x,y)| > T \\ 1 & \text{if } (g_l(x,y) > 0) \text{ and } |g_l(x,y)| > T \\ 0 & \text{ortherwise} \end{cases}$$

2. Apply a mask 3×3 at each pixel on $g_l(x,y)$.

NW	N	NE
W	C	E
SW	S	SE

3. Detect the difference on the sign at opposing corners, i.e., (W, E), (N, S), (NW, SE), and (SW, NE).
4. If any pair of corners results a difference on the sign, then $g_l(x,y)$ is an edge point.

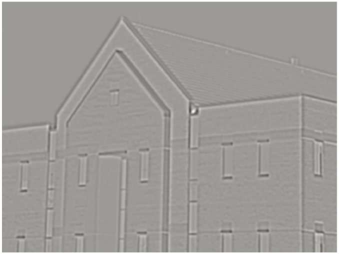
Laplacian of Gaussian (LoG): Illustration

Figure: Original image

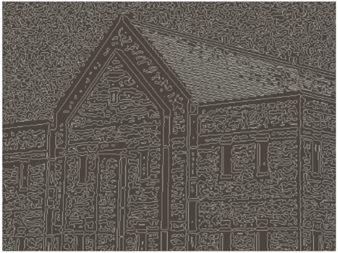
Laplacian of Gaussian (LoG): Illustration

Figure: Original image

Laplacian of Gaussian (LoG): Illustration



(a)

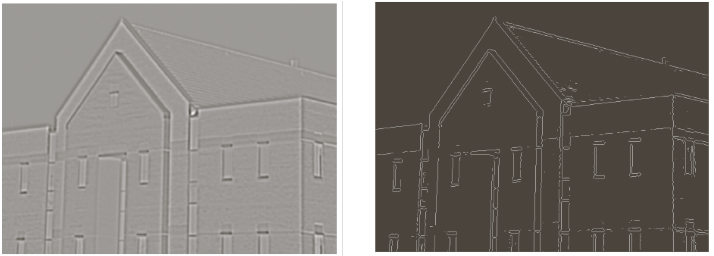


(b)

Figure: Marr-Hildreth Algorithm: (a): Result of Step 1 and 2, (b): Zero-crossing of (a), Threshold $= 0$

- Step 1 and 2: $\sigma = 4, n = 25$ (kernel's size: 25×25)
- Low threshold $\Rightarrow$ many edge points.

Laplacian of Gaussian (LoG): Illustration



(a) (b)

Figure: Marr-Hildreth Algorithm: (a): Result of Step 1 and 2, (b): Zero-crossing of (a), Threshold $= 4\%$ of maximum value in (a)

- Step 1 and 2: $\sigma = 4, n = 25$ (kernel's size: 25×25)
- Larger threshold $\Rightarrow$ provide strong edge only

Canny Edge Detection

Canny Edge Detection Algorithm

1. Smooth the input image with Gaussian low-pass filter
2. Compute the gradient magnitude angle images
3. Apply nonmaxima suppression to the gradient magnitude image.
4. Use double thresholding and connectivity analysis to detect and link edges

Canny Edge Detection

Step 1: Smooth the input image with Gaussian low-pass filter

1. Smooth the input image with Gaussian low-pass filter
2. Compute the gradient magnitude angle images
3. Apply nonmaxima suppression to the gradient magnitude image.
4. Use double thresholding to obtain strong and weak edge masks
5. Analyze the connectivity to detect and link edges

Canny Edge Detection

Step 1: Smooth the input image with Gaussian low-pass filter

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$
$$f_s(x, y) = f(x, y) * G(x, y)$$

- $f_s(x, y)$: a smoothed version of $f(x, y)$
- σ : decides the degree of smoothing
- $f_s(x, y)$: Gaussian noise has been removed

Canny Edge Detection

Step 2: Compute the gradient magnitude angle images

- Compute $g_x(x, y)$ and $g_y(x, y)$

$$g_x(x,y) = f_s(x,y) * H_x(x,y)$$
$$g_y(x,y) = f_s(x,y) * H_y(x,y)$$

  - $H_x(x, y)$, $H_y(x, y)$: any first-order derivative kernels, e.g., "standard" approximations kernels, Sobel, Roberts, Prewitts, etc.

- Compute gradient magnitude and angle images

$$M(x,y) = \begin{bmatrix} g_x(x,y) \\ g_y(x,y) \end{bmatrix}$$
$$\alpha(x,y) = tan^{-1}\left[\frac{g_y(x,y)}{g_x(x,y)}\right]$$

Canny Edge Detection

Step 3: Apply nonmaxima suppression to the gradient magnitude image.

The underlying idea of nonmaxima suprression

if a point is not a local maxima, then supress (remove, stop, etc) it.

- Edges will pass points that are **local maxima** in gradient magnitude image, ie., $M(x, y)$.
- $\Rightarrow$ Remove (supress) points that are not local maxima.
- $\equiv$ **nonmaxima suprression**

Canny Edge Detection

Step 3: Apply nonmaxima suppression to the gradient magnitude image.

Questions

What does **local** mean?

- **local** $\equiv$ local points involving in edge.
- for a point (x, y) in $M(x, y)$, **which neighbor points are edge local points?**
- $\Rightarrow$ need gradient angle

Gradient vector at a point is **perpendicular** to local edge at that point.

Canny Edge Detection

Step 3: Apply nonmaxima suppression to the gradient magnitude image.

1. Discrete gradient angle values into small rangles.
2. Find direction d_k that is closest to $\alpha(x, y)$
3. Find local neighbors on edge using d_k, referred to as N_1 and N_2
4. Compute nonmaxima suppressed image $g_N(x, y)$

$$g_N(x, y) = \begin{cases} 0 & \text{if } [M(x, y) < N_1] \,\&\, [M(x, y) < N_2] \\ M(x, y) & \text{otherwise} \end{cases}$$

Canny Edge Detection

Step 3: Apply nonmaxima suppression to the gradient magnitude image.

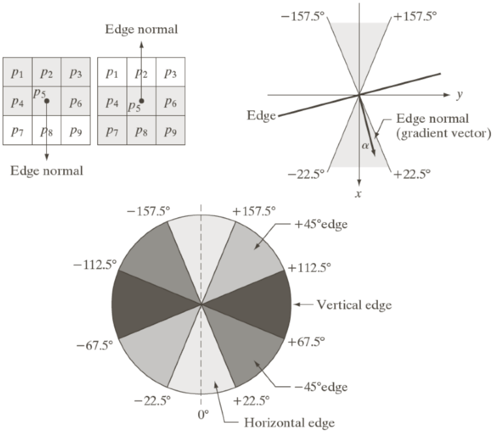


Figure: Demonstration for 4 directions: horizontal, vertical, $\pm 45^0$

Canny Edge Detection

Step 4: Use double thresholding to obtain strong and weak edge masks

1. Do thresholding with high and low threshold value T_H and T_L respectively.

$$g_{NH}(x,y) = g_N(x,y) \geq T_H$$
$$g_{NL}(x,y) = g_N(x,y) \geq T_L$$

2. Eliminate points in $g_{NL}(x,y)$ that has been indicated in $g_{NH(x,y)}$

$$g_{NL}(x,y) = g_{NL}(x,y) - g_{NH}(x,y)$$

- $g_{NH}(x,y)$: strong edge
- $g_{NL}(x,y)$: weak edge

Canny Edge Detection

Step 5: Analyze connectivity and to detect and link edges

1. Create an **edge map** that marks all non-zeros in $g_{NH}(x, y)$ as valid edge points.
2. For each pixel p that is non-zeros in $g_{NH}(x, y)$, do
   - Find all non-zeros pixels in $g_{NL}(x, y)$ that are connected to p via $4-$ or $8-$connectivity, mark corresponding points in **edge map** as valid pixels.

- **edge map** may contain edges thicker than 1 pixel.
- Apply edge-thinning algorithm to create thinner edge map, if needed.

Canny Edge Detection: Illustration

(a) (b)

Figure: Edge detection (a): Original image, (b): Thresholded gradient magnitude image - **thick edge**

Canny Edge Detection: Illustration

(a)

(b)

Figure: Edge detection (a): Marr-Hildreth Method, (b): Canny method - **better**