

Derivatives, Edges and Pyramids

Computer Vision I (CSCI 3240U)

Faisal Z. Qureshi

<http://vclab.science.ontariotechu.ca>



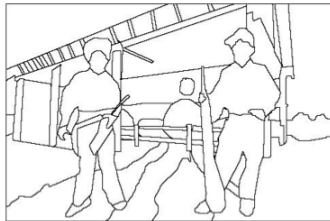
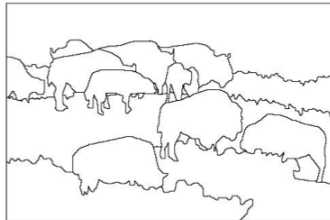
Week 4 at a glance

- ▶ **Lecture 1: Derivatives and gradients (Sec. 3.2.3)**
 - ▶ Finite differences and Taylor series
 - ▶ Gradient magnitude and orientation
 - ▶ Noise, and why we smooth first
 - ▶ Evaluating an edge map
- ▶ **Lecture 2: Canny and multiscale representations (Sec. 7.2.1, 3.5)**
 - ▶ Canny: non-maximum suppression and hysteresis
 - ▶ Sampling and aliasing
 - ▶ Gaussian and Laplacian pyramids
 - ▶ Coarse-to-fine

Reading: Szeliski Sec. 3.2.3, 7.2.1, 3.5. Exercises w04a, w04b, w04c. **Labs 4 and 5** build directly on this.

Lecture 1: Derivatives and edges

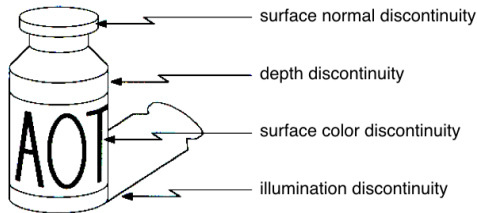
Edge detection



What is an edge

Edges (a rapid change in intensity) are caused by a variety of factors:

- ▶ Changes in appearance and texture (reflectance changes)
- ▶ Object boundaries and depth discontinuities
- ▶ Changes in surface orientation
- ▶ Shadows (illumination changes)
- ▶ The image cannot tell you which



Edge detection

- ▶ Identify sudden changes (discontinuities) in an image
- ▶ Edges encode semantic and shape information
- ▶ Edges are more compact than pixels

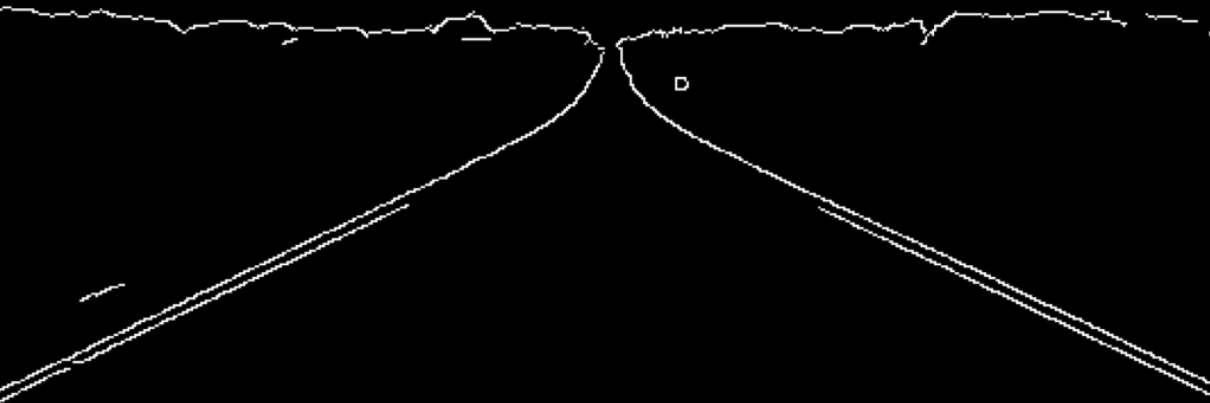


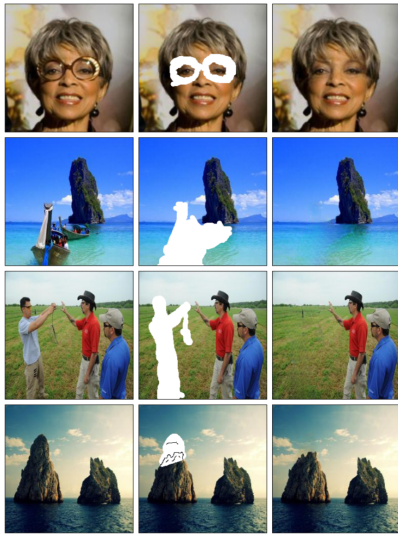
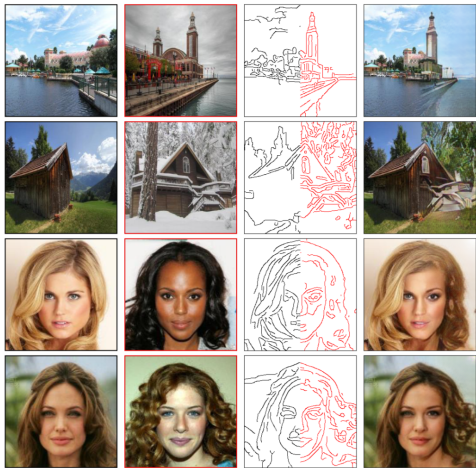
An artist's line drawing

[Source: D. Lowe]

Uses of edge detection

- ▶ Object recognition
- ▶ Scene understanding
- ▶ 3D scene reconstruction



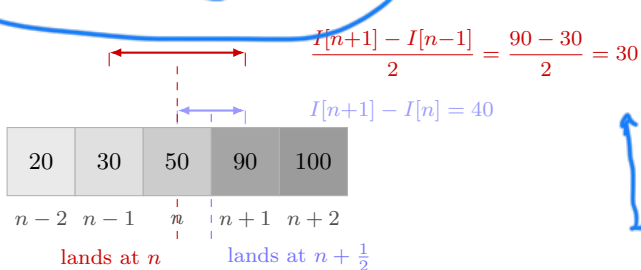


<https://vclab.ca/projects/edge-connect/edge-connect.html>

Derivatives of a discrete image

Finite difference is off centre: it really measures the slope halfway between the two pixels. Average the step to the right with the step to the left and it recentres, giving the **central difference**:

$$\frac{\partial I}{\partial x} \approx \frac{I[m, n+1] - I[m, n-1]}{2} \quad \text{kernel } \frac{1}{2} \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

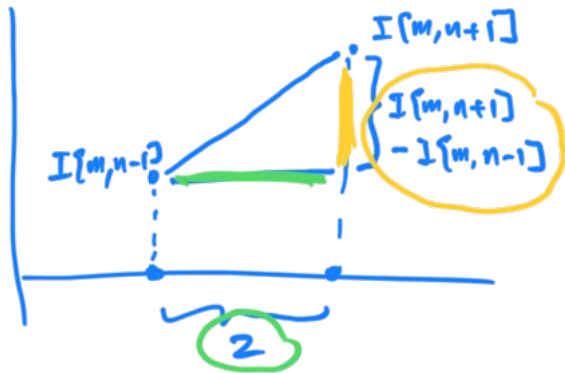


Derivatives are just filters. Everything from Week 3 applies.

$$\frac{\partial I}{\partial x} = \frac{I[m, n+1] - I[m, n-1]}{2}$$

$$= \frac{1}{2} \left(\underbrace{(I[m, n+1])}_{\text{red}} \underbrace{(1)}_{\text{green}} + \underbrace{(I[m, n])}_{\text{red}} \underbrace{(0)}_{\text{green}} + \underbrace{(I[m, n-1])}_{\text{red}} \underbrace{(-1)}_{\text{green}} \right)$$

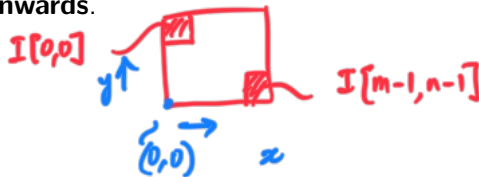
$$\Rightarrow \frac{1}{2} (1 \quad 0 \quad -1)$$



One convention, fixed here and used all term

Every derivative kernel printed in this course is a **convolution** kernel, and y increases **downwards**, as it has since Week 1.

- ▶ Correlate with the same kernel instead and that component changes sign.
- ▶ With this convention the signs are the physical ones: $I_x > 0$ where the intensity grows to the **right**, $I_y > 0$ where it grows **downwards**.



Write it on the same page as the Week 1 conventions. Both w04a and w08a print these kernels and both apply them as convolutions, so a sign slip here propagates straight into your answers.

How to derive the filters that compute derivatives

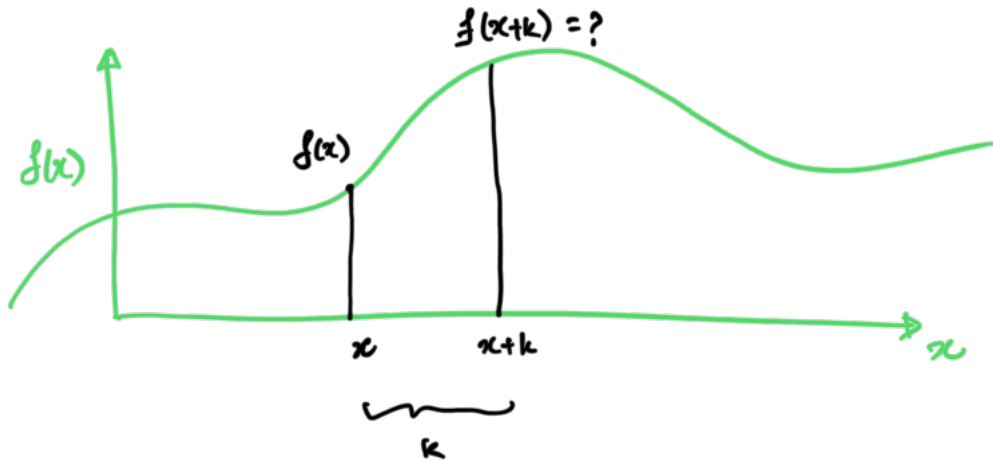
We know I only at whole pixels. Taylor's series writes a neighbour k pixels away in terms of the derivatives **at** x :

$$I(x+k) = I(x) + \underbrace{k I'(x)}_{\partial I / \partial x} + \frac{k^2}{2!} I''(x) + \frac{k^3}{3!} I'''(x) + \frac{k^4}{4!} I^{(4)}(x) + \dots$$

Each filter is a weighted sum of neighbours, which numerical analysis calls a **stencil**. To build one:

1. Write each neighbour as a Taylor series.
2. Choose the weights so every term cancels except the derivative you want.
3. The first term that survives is the **error**.

The simplest case is the finite difference we started with: one neighbour, one term dropped. We derive it next, then work up to the kernels the labs use.



From Taylor's series to the forward difference

Take a single neighbour, $k = 1$:

$$I(x+1) = I(x) + I'(x) + \frac{1}{2}I''(x) + \frac{1}{6}I'''(x) + \dots$$

Now solve for the derivative:

$$I'(x) = \underbrace{I(x+1) - I(x)}_{\text{pixels we have}} - \underbrace{\left(\frac{1}{2}I''(x) + \frac{1}{6}I'''(x) + \dots\right)}_{\text{dropped, so this is the error}}$$

Keep the first bracket, throw the tail away, and you have the **forward difference**.

Every stencil is this one move. More neighbours give more weights to play with, so more of the tail cancels instead of being dropped.

From Taylor's series to the central difference

Put $k = \pm 1$:

$$I(x \pm 1) = I \pm I' + \frac{1}{2}I'' \pm \frac{1}{6}I''' + \frac{1}{24}I^{(4)} \pm \dots$$

Subtract, and the even terms cancel:

$$\frac{I(x+1) - I(x-1)}{2} = I' + \underbrace{\frac{1}{6}I''' + \dots}_{\text{leftover, error}}$$

Add, and the odd terms cancel:

$$I(x+1) - 2I(x) + I(x-1) = I'' + \underbrace{\frac{1}{12}I^{(4)} + \dots}_{\text{leftover, error}}$$

In each line, the left side uses only pixel values: it is the stencil. The right side is what the stencil actually computes, the derivative we want plus leftover terms.

$$I[x+1] = I[x] + I'[x] + \frac{1}{2} I''[x] + \frac{1}{6} I'''[x] + \dots$$

$$I[x-1] = I[x] - I'[x] + \frac{1}{2} I''[x] - \frac{1}{6} I'''[x] + \dots$$

$$I[x+1] - I[x-1] = +2 I'[x] + \frac{1}{3} I'''[x] + \dots$$

$$\Rightarrow I'[x] = \frac{1}{2} (I[x+1] - I[x-1]) - \frac{1}{6} I'''[x] + \dots$$

error

$$I'[x] \quad \frac{1}{2} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Why central and not forward

The forward difference uses one side only:

$$I(x+1) - I(x) = I' + \frac{1}{2}I'' + \dots$$

- ▶ Forward: the error depends on I'' , so it is exact only for **straight lines**.
- ▶ Central: the error depends on I''' , so it is exact for **parabolas** too.

Both cost one subtraction. With pixel spacing h , the forward error shrinks like h and the central error like h^2 .

Five-point stencil for I'

The central difference left $\frac{1}{6}I'''$ behind. Reach out to the neighbours two pixels away and we have a second weight to play with, enough to cancel that term too. Subtract as before, once at each distance:

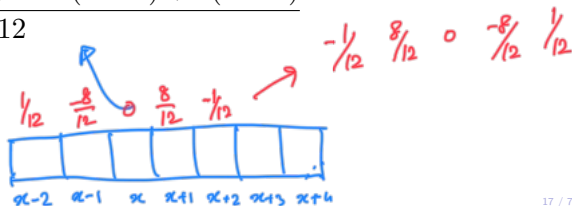
$$I(x+1) - I(x-1) = 2I' + \frac{1}{3}I''' + \dots$$

$$I(x+2) - I(x-2) = 4I' + \frac{8}{3}I''' + \dots$$

The far pair carries **eight times** as much I''' as the near pair, so $8\times$ the first line minus the second wipes it out, leaving $12I'$:

$$\underline{\underline{I'}} \approx \frac{-I(x+2) + 8I(x+1) - 8I(x-1) + I(x-2)}{12}$$

The first term left over is now $-\frac{1}{30}I^{(5)}$.



Where $-\frac{1}{30}I^{(5)}$ comes from

Subtracting cancels every **even** order, whatever the weights, so only odd orders can survive. Keep one more of them in each line:

$$I(x+1) - I(x-1) = 2I' + \frac{1}{3}I''' + \frac{1}{60}I^{(5)} + \dots$$

$$I(x+2) - I(x-2) = 4I' + \frac{8}{3}I''' + \frac{8}{15}I^{(5)} + \dots$$

The weights 8 and -1 were chosen to kill I''' . They do nothing for $I^{(5)}$, the next odd order, which therefore survives:

$$8 \cdot \frac{1}{60} - \frac{8}{15} = \frac{2}{15} - \frac{8}{15} = -\frac{2}{5} \quad \implies \quad 8 \text{ (first)} - \text{(second)} = 12I' - \frac{2}{5}I^{(5)}$$

Divide by 12 and the stencil equals $I' - \frac{1}{30}I^{(5)}$.

You cancel what you aim at, and inherit the next one along.

Five-point stencil for I''

The second derivative came from **adding** one neighbour each side, and it left $\frac{1}{12}I^{(4)}$ behind. Add at both distances:

$$I(x+1) + I(x-1) = 2I + I'' + \frac{1}{12}I^{(4)} + \dots$$

$$I(x+2) + I(x-2) = 2I + 4I'' + \frac{4}{3}I^{(4)} + \dots$$

The far pair carries **sixteen times** as much $I^{(4)}$, so $16\times$ the first line minus the second cancels it, leaving $30I + 12I''$:

$$I'' \approx \frac{-I(x+2) + 16I(x+1) - 30I(x) + 16I(x-1) - I(x-2)}{12}$$

The first term left over is now $-\frac{1}{90}I^{(6)}$.

Where $-\frac{1}{90}I^{(6)}$ comes from

Adding cancels every **odd** order, so only even orders can survive. Keep one more of them in each line:

$$I(x+1) + I(x-1) = 2I + I'' + \frac{1}{12}I^{(4)} + \frac{1}{360}I^{(6)} + \dots$$

$$I(x+2) + I(x-2) = 2I + 4I'' + \frac{4}{3}I^{(4)} + \frac{8}{45}I^{(6)} + \dots$$

The weights 16 and -1 were chosen to kill $I^{(4)}$, and again $I^{(6)}$ is left standing:

$$16 \cdot \frac{1}{360} - \frac{8}{45} = \frac{2}{45} - \frac{8}{45} = -\frac{2}{15} \quad \implies \quad 16(\text{first}) - (\text{second}) = 30I + 12I'' - \frac{2}{15}I^{(6)}$$

Subtract the $30I$, divide by 12, and the stencil equals $I'' - \frac{1}{90}I^{(6)}$.

Two more neighbours would cancel $I^{(6)}$ and leave $I^{(8)}$. On smooth data that is progress; on pixels it mostly amplifies noise.

The stencils as convolution kernels

	three-point	five-point
I'	$\frac{1}{2} \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$	$\frac{1}{12} \begin{pmatrix} -1 & 8 & 0 & -8 & 1 \end{pmatrix}$
error	$\frac{1}{6} I'''$	$-\frac{1}{30} I^{(5)}$
I''	$\begin{pmatrix} 1 & -2 & 1 \end{pmatrix}$	$\frac{1}{12} \begin{pmatrix} -1 & 16 & -30 & 16 & -1 \end{pmatrix}$
error	$\frac{1}{12} I^{(4)}$	$-\frac{1}{90} I^{(6)}$

Convolution flips the kernel, so the weight on $I(x+2)$ is written **first**. The I'' kernels are symmetric, so the flip does not change them.

In 2D, and exercise w04b

A partial derivative holds the other coordinate fixed, so the 1D stencils apply unchanged:

- ▶ I_x and I_{xx} : run the stencil **along the row** through the pixel.
- ▶ I_y and I_{yy} : run it **down the column**.

The two together form a **cross** centred on the pixel. w04b marks exactly that cross on an 8×8 image and asks for all four derivatives at pixel $(3, 3)$ with the five-point stencils.

Keep y downwards: $I(y + 1)$ is the pixel **below**. Reversing it flips the sign of I_y and leaves the other three alone.

Wider is not always better

The five-point stencils are exact for polynomials up to degree 4, but images are not polynomials. Their larger weights also amplify noise. For independent pixel noise of standard deviation σ , a stencil with weights w outputs noise of $\sigma\sqrt{\sum w^2}$:

	three-point	five-point
I'	0.71σ	0.95σ
I''	2.45σ	3.13σ

On the real pixels of w04b the two stencils disagree noticeably. Try both.

Next: smooth while you differentiate.

Sobel

Central differences are noisy. Smooth along the perpendicular direction at the same time:

$$S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Note S_x is separable:

$$S_x = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -1 \end{pmatrix}$$

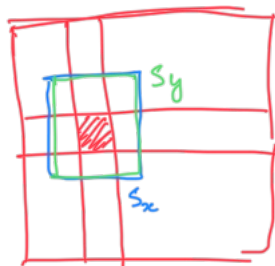
a smoothing kernel across, a difference kernel along.

Sign conventions differ between textbooks. Flipping a kernel negates that component and rotates the gradient direction by 180° . Magnitude is unaffected. Ours is the one fixed two slides back: these are convolution kernels, y downwards.

Gradient magnitude and orientation

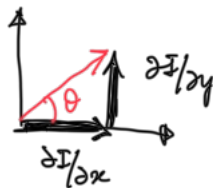
$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$$

$$\|\nabla I\| = \sqrt{I_x^2 + I_y^2} \quad \theta = \text{atan2}(I_y, I_x)$$



- ▶ $\|\nabla I\|$: **how strong** the edge is.
- ▶ θ : the direction of **steepest increase**, which is **perpendicular** to the edge.
- ▶ Use `atan2`, not `arctan`, or you lose the quadrant.

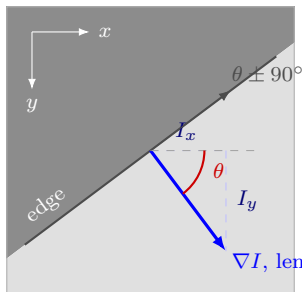
Edge orientation is $\theta \pm 90^\circ$.



$$\begin{aligned} \text{blue box} &\sim \frac{\partial I}{\partial x} \\ \text{green box} &\sim \frac{\partial I}{\partial y} \end{aligned}$$

The gradient at one pixel

The gradient points **across** the edge, from dark to light. The edge itself runs at right angles to it.



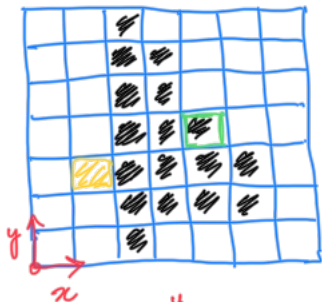
$$I_x = 3, \quad I_y = 4$$

$$\|\nabla I\| = \sqrt{3^2 + 4^2} = 5$$

$$\theta = \text{atan2}(4, 3) = 53^\circ$$

edge runs at $53^\circ \pm 90^\circ$

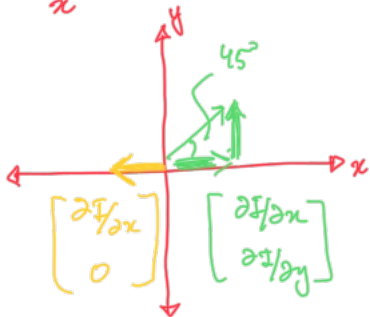
∇I , length $\|\nabla I\|$



$$\nabla I = \begin{bmatrix} \partial I / \partial x \\ \partial I / \partial y \end{bmatrix}$$

$$|\nabla I|^2 = \left(\partial I / \partial x \right)^2 + \left(\partial I / \partial y \right)^2$$

$$\theta = \arctan \left(\frac{\partial I / \partial y}{\partial I / \partial x} \right)$$



Data types: a real trap

Derivatives are **signed**.

Computing I_x on a uint8 array silently discards every negative value: you get one-sided edges and a magnitude map that is wrong everywhere the intensity decreases.

```
I = I.astype(np.float32)      # before differentiating
```

Convert back only for display. This costs one line and saves an afternoon.

Differentiation amplifies noise

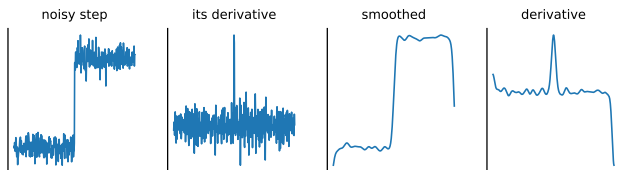
Derivative of a noisy image is dominated by noise. **Smooth first.**

By associativity,

$$\frac{\partial}{\partial x}(G_{\sigma} * I) = \left(\frac{\partial G_{\sigma}}{\partial x}\right) * I$$

So we can differentiate the *Gaussian* once, analytically, and convolve with that: one pass instead of two. This is the **derivative of Gaussian** filter.

σ controls **which edges are visible**: small σ finds texture, large σ finds only major boundaries.



Effect of noise on edge detection

Consider the following 1d image.

f

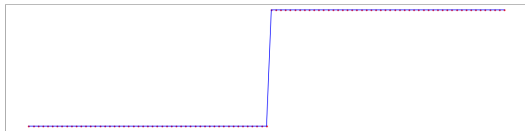
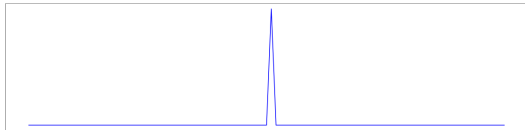


Image derivative successfully localizes the edge seen in this image.

$\frac{\partial}{\partial x} f$



Effect of noise on edge detection

Now consider the same 1d image, but this time corrupted with some noise. Notice that the edge is still seen in the image.

f

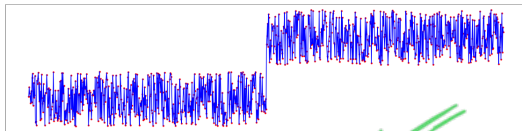


Image derivative, however, is unable to localize the edge in this case.

$\frac{\partial}{\partial x} f$

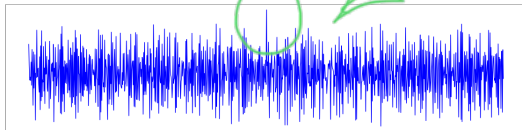
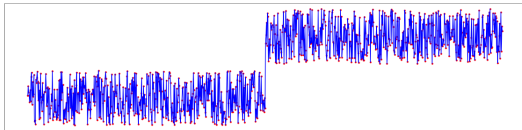


Image derivatives are highly sensitive to the presence of noise in the image. Use smoothing first to get rid of high-frequency components.

Smoothing to reduce the effect of noise

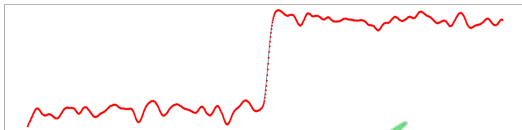
Given noisy image

f



Step 1: Perform Gaussian filtering

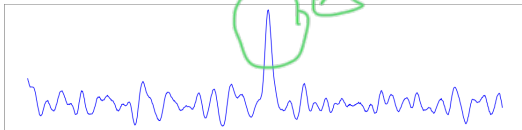
$G * f$



Step 2: Compute image derivative

$\left(\frac{\partial G}{\partial x}\right) * f$

$\frac{\partial}{\partial x}(G * f)$



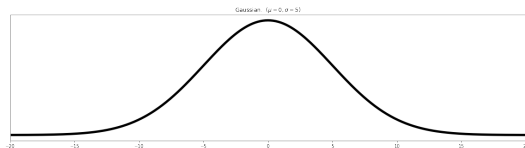
Smoothing to reduce the effect of noise

Recall that convolution (linear filtering) is associative, meaning

$$\frac{\partial}{\partial x}(G * f) = \left(\frac{\partial}{\partial x}G\right) * f$$

Gaussian

G



First derivative of Gaussian

$\frac{\partial}{\partial x}G$

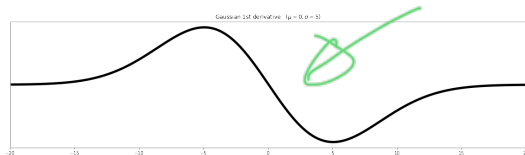
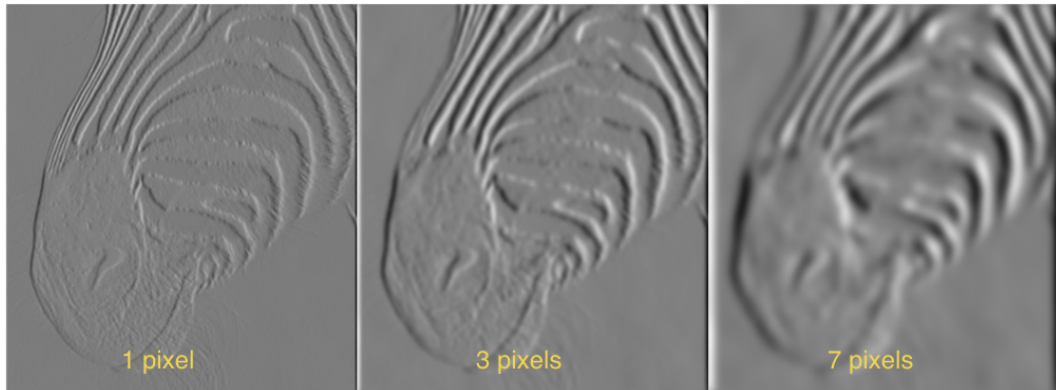


Image smoothing & edge localization

Smoothed derivative removes noise, but blur the edges, making edge localization challenging.



[Source: D. Forsyth]

Image smoothing & edge localization

- ▶ The gradient magnitude is large along a thick 'ridge'. How do we identify (localize) the actual edge pixels?
- ▶ How do we link edge pixels to form contours?



[Source: D. Forsyth]

Evaluating an edge map

In Lab 4 you score your edges against hand-labelled ground truth. Two counting rules:

$$\text{precision} = \frac{TP}{TP + FP}, \quad \text{recall} = \frac{TP}{TP + FN}$$

with a **tolerance**: a detection within τ pixels of the reference counts, because the reference was traced by a human.

Do not report accuracy. The road boundary is under half a percent of the image, so “no edge anywhere” scores over 99% accurate. On unbalanced problems accuracy is meaningless.



A warning about precision here

Your edge detector will *correctly* fire on kerbs, markings, cars and buildings. All are real edges, and none of them is the road boundary you are scoring against.

So the reference is a thin curve and your output is a dense map: precision is capped low **by the task**, not by your code.

In Lab 4 we therefore report

- ▶ **recall**: did the boundary survive?
- ▶ **selectivity**: how many candidates came with it?

Precision becomes meaningful in Week 7, once we fit *lines* and commit to an answer.

Lecture 1 recap

- ▶ Derivatives are filters; the stencils come from Taylor series.
- ▶ $\|\nabla I\|$ is edge strength; θ is perpendicular to the edge.
- ▶ Convert to float first; derivatives are signed.
- ▶ Smooth before differentiating; derivative-of-Gaussian does both at once.
- ▶ Score edges with recall and a tolerance; never accuracy.
- ▶ **Next lecture:** Canny, then multiscale representations.

Lecture 2: Canny and multiscale representations

Designing an edge detector

- ▶ An edge detector should find all “real” edges, ignoring noise and other artifacts
- ▶ An edge detector should have good localization property, i.e., the detected edges should be as close to real edges as possible.

Cues that we can use for developing a useful edge detector

- ▶ Intensity, color or texture differences
- ▶ Continuity or closure
- ▶ High-level or semantic knowledge

Canny edge detector

Canny, J., *A Computational Approach To Edge Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

- ▶ Most widely used edge-detector in computer vision
- ▶ Theoretical model: step-edges corrupted by Gaussian noise
- ▶ Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of signal-to-noise ratio and localization

Canny edge detector: key ideas

1. Non-maximum suppression. Keep a pixel only if its magnitude is a local maximum *along the gradient direction*. Ridges become one-pixel lines.

2. Hysteresis thresholding. Two thresholds $t_{\text{low}} < t_{\text{high}}$:

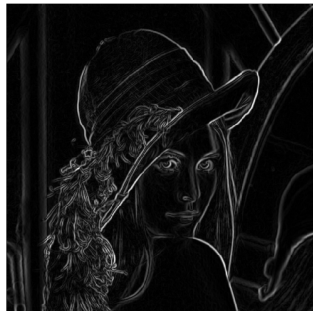
- ▶ above t_{high} : definitely an edge;
- ▶ below t_{low} : discard;
- ▶ between: keep **only if connected** to a strong edge.

That connectivity rule is what stops a fading lane marking from breaking into dashes.

Thresholding $\|\nabla I\|$ gives thick, broken, threshold-sensitive edges.

Canny edge detector - Edge pixel identification

Step 1 Identify edge pixels using gradient magnitude

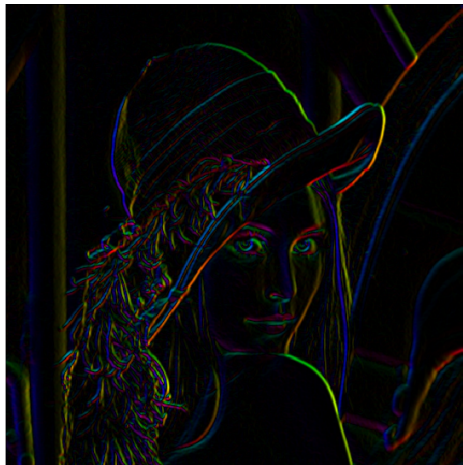


$$\frac{R+G+B}{3}$$

$$|\nabla I|^2$$

Canny edge detector - Edge orientations

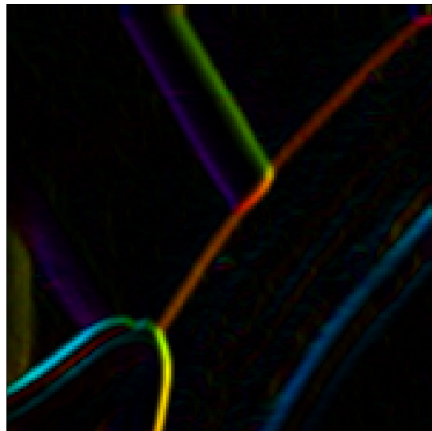
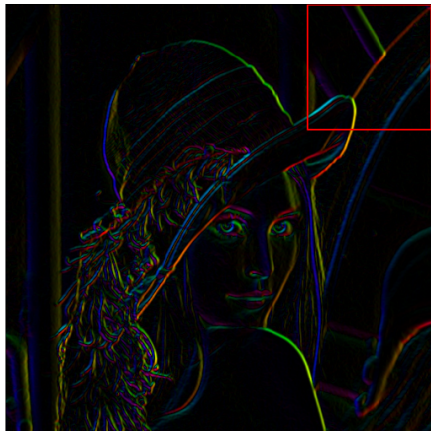
Step 2 Compute edge orientations using image gradient



θ

Canny edge detector - Edge localization

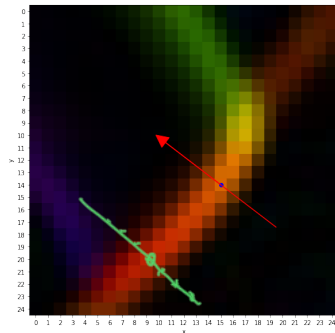
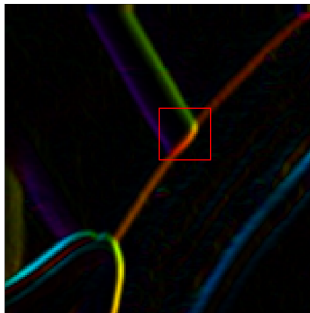
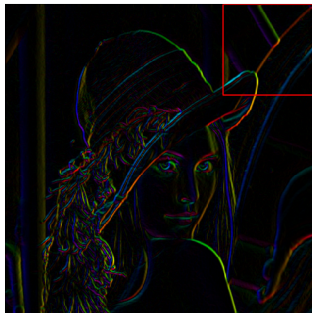
Step 3 In order to achieve good edge localization, we need to thin out the edges.



Canny edge detector - Edge localization

Step 3 In order to achieve good edge localization, we need to thin out the edges.

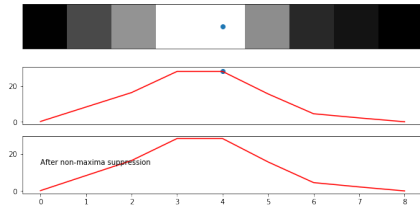
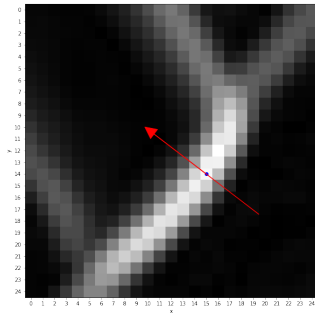
We will use non-maxima-suppression to achieve that. Non-maxima suppression needs to account for the orientation at each location.



Canny edge detector - Non-Maxima Suppression

Sample pixel values across the edge (using gradients) and only keep a pixel if it has the maximum value.

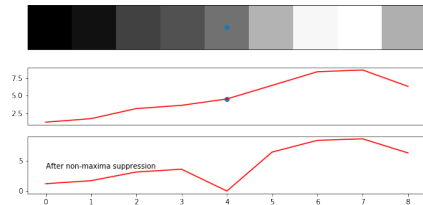
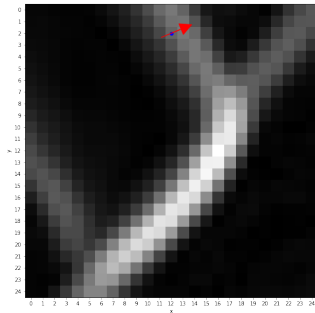
Pixel is **kept** since it has the largest gradient magnitude across the edge



Canny edge detector - Non-Maxima Suppression

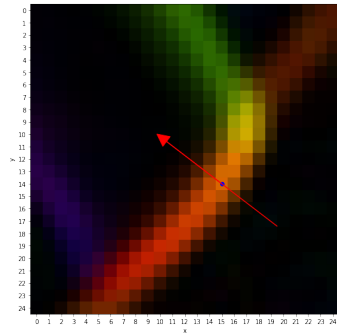
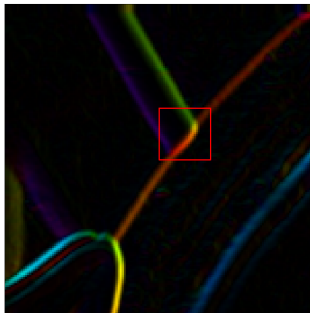
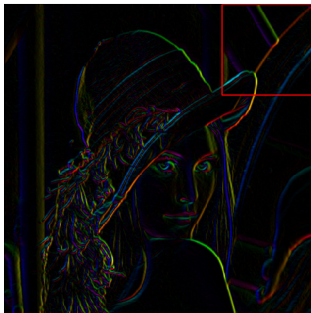
The same test on a pixel that is not on the ridge.

Pixel is **discarded** since it does not have the largest gradient magnitude across the edge



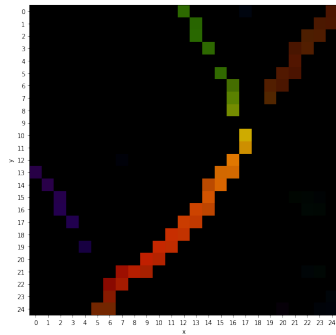
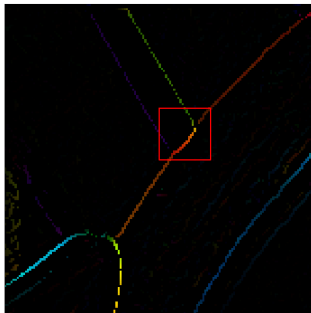
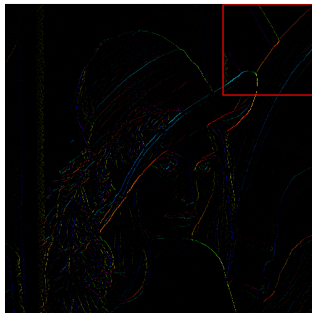
Canny edge detector - Non Maxima Suppression

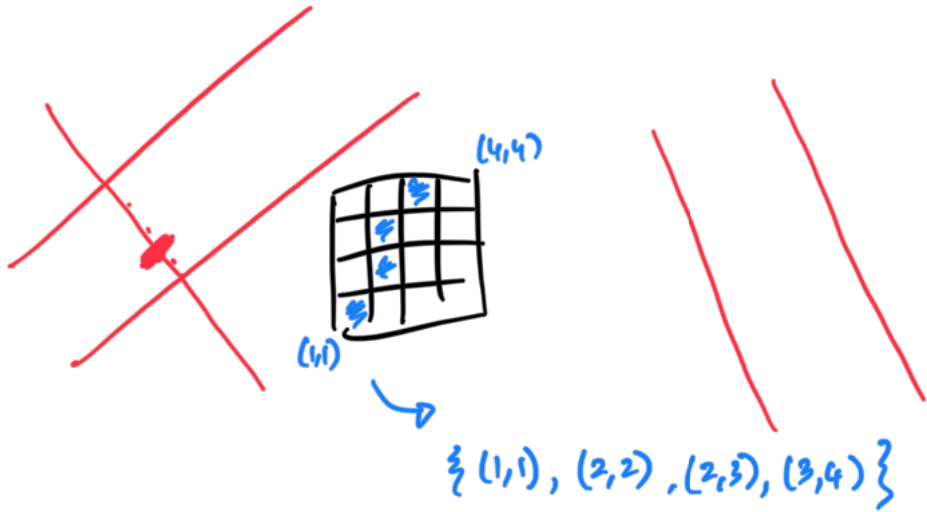
Before



Canny edge detector - Non Maxima Suppression

After



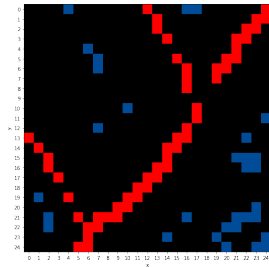
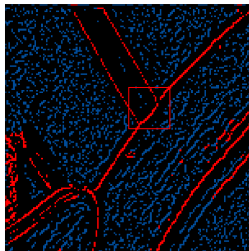
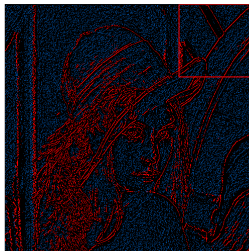


Canny edge detector - Discard non-edge pixels

Pick two thresholds t_1 and t_2 , where $t_1 < t_2$, and use these to discard non-edge pixels.

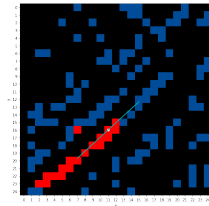
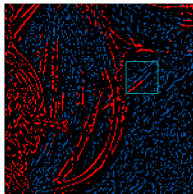
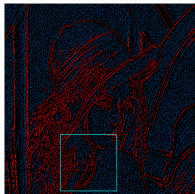
$$E(x, y) = \begin{cases} \text{not an edge,} & \text{when } |\nabla I(x, y)| < t_1 \\ \text{weak edge,} & \text{when } t_1 \leq |\nabla I(x, y)| \leq t_2 \\ \text{strong edge,} & \text{when } |\nabla I(x, y)| > t_2 \end{cases}$$

E captures the information about non-edge, weak-edge, and strong-edge pixels.



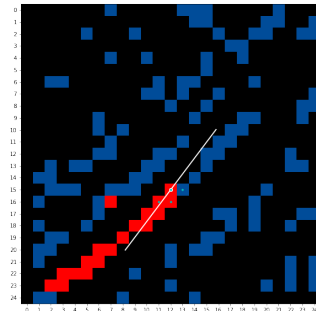
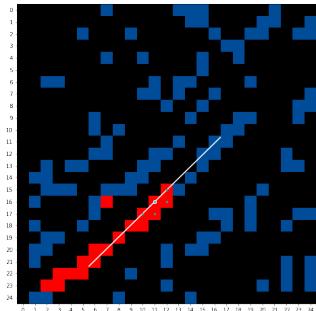
Canny edge detector - Hysteresis

- ▶ Start edges at “strong” edge pixels and continue these on “weak” pixels that fall along these edges.
- ▶ Use **tangent** at each pixel to identify neighbouring pixels that fall along the edge.



Canny edge detector - Hysteresis

Use edge tangent to identify neighbouring pixels that fall along the same edge.

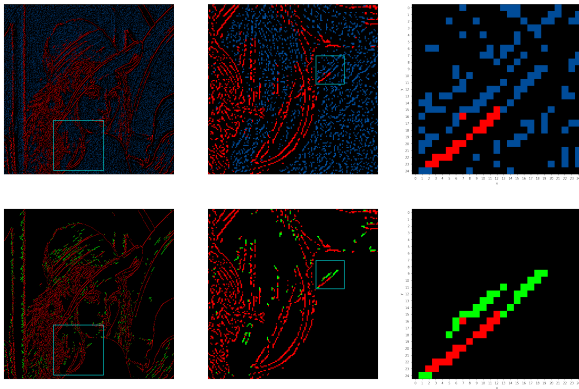


Tangent at each pixel is easily computed by rotating gradient at that pixel by 90 degrees.

Canny edge detector - Hysteresis

Weak pixels that **do not** fall along edges containing strong pixels are discarded.

In the figure below, notice that some blue pixels are turned off (set to black), while others are set to green. Both red and green pixels are edge pixels. Before (top row) and after (bottom row) performing hysteresis based edge identification.

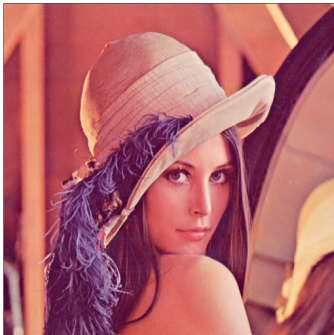


Canny edge detector - Open CV implementation

OpenCV includes an implementation of Canny edge detector.

```
import cv2
import numpy as np

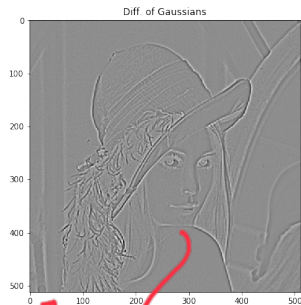
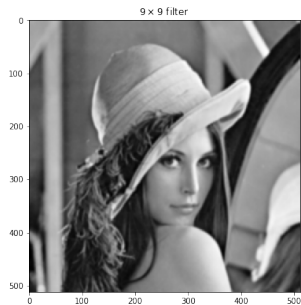
img_bgr = cv2.imread(filename)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
canny = cv2.Canny(img_gray, threshold1=60, threshold2=120, apertureSize=3, L2gradient=True)
```



Difference of Gaussian (DoG)

DoG can be used to approximate smoothed gradient of an image as seen below.

$$\frac{\partial}{\partial x} (G_{\sigma} * I) \approx (G_{\sigma_1} * I) - (G_{\sigma_2} * I)$$



not contours

Why scale matters

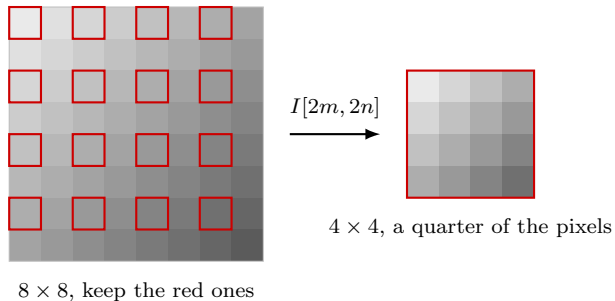
An object 10 m away and the same object 40 m away differ by a factor of four in image size. A filter tuned to one will miss the other.

Two options:

1. Keep the image, change the filter size. Cost grows with k .
2. Keep the filter, **change the image size**. Cost *falls* as we go coarser.

Option 2 is a **pyramid**.

Downsampling: keep every second sample



Nothing is averaged and nothing is smoothed. Three pixels in four are simply thrown away.

Downsampling done wrong

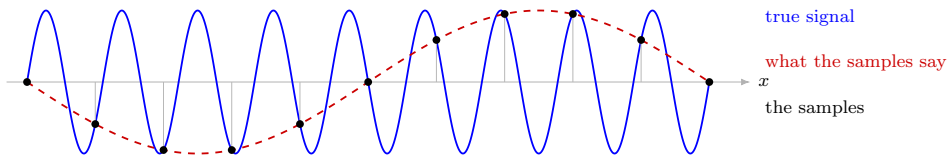
Taking every second pixel throws away half the samples, but the fine detail does not disappear: it **reappears as false coarse structure**.

This is **aliasing**. On a road image it is easy to see: dashed lane markings gain dashes that were never there, and railings develop moiré patterns.

The sampling theorem says we can only represent frequencies below half the sampling rate. Anything above must be **removed before** subsampling, not after.

Two signals, one set of samples

The blue signal turns nine tenths of a cycle between samples. The samples cannot tell that from one tenth of a cycle the other way, so the fine detail comes back as **false coarse structure**.



Every sample lands on both curves, so nothing in the data separates them.

Aliasing on a real frame



1 The solid lane marking breaks into dashes. **2** The grass turns into speckle that changes with every frame. **3** The railing and tram tracks flicker: rails vanish, others double. Blurring first (bottom) keeps all three honest.

Downsampling done right

- ▶ `pyr_down`: blur with G_σ , then keep every second sample. **The order is not optional.** Blur first, subsample second.
- ▶ Going the other way:
`pyr_up`: insert zeros between samples, then blur, then scale by 4

The factor of 4 compensates for the energy lost to the inserted zeros in 2D. Getting it wrong is the usual cause of a reconstruction that is *nearly* right.

$$I^0 \in \mathbb{R}^{256 \times 256}$$

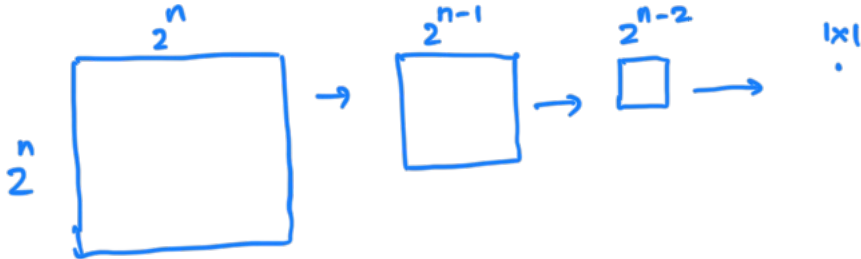
$$I_g = \text{Blur}(I_g)$$

$$I' = I_g[:, ::2], ::2]$$

The Gaussian pyramid

$$\mathbf{g}_0 = I, \quad \mathbf{g}_{i+1} = \text{pyr_down}(\mathbf{g}_i)$$

- ▶ Each level is half the width and half the height, a quarter of the pixels.
- ▶ Total storage is $1 + \frac{1}{4} + \frac{1}{16} + \dots \rightarrow \frac{4}{3}$ of the original. A pyramid is cheap.
- ▶ Each level is a **progressively blurrier** copy. Level i has lost everything above its Nyquist frequency, so the levels are not independent; they overlap heavily.



Four levels of a Gaussian pyramid

Drawn at true relative size. Level 0 holds 192,000 pixels and level 3 holds 3,008, yet the whole pyramid costs only $1.33\times$ level 0. The car survives every level; the overhead wires are gone by g_2 .

g_0 512×375



g_1 256×188



g_2 128×94



g_3 64×47



The Laplacian pyramid

Store what each level *adds* instead:

$$\mathbf{L}_i = \mathbf{g}_i - \text{pyr_up}(\mathbf{g}_{i+1})$$

keeping the coarsest Gaussian level $\mathbf{g}_n$ at the top.

- ▶ $\mathbf{L}_i$ is a **band-pass** image: the detail present at level i but not at level $i + 1$.
- ▶ Mostly near zero, so it compresses well.

It is invertible:

$$\mathbf{g}_{i-1} = \mathbf{L}_{i-1} + \text{pyr_up}(\mathbf{g}_i)$$

Start at the top and walk down. This is exercise w04c, and Task 4 of Lab 5 checks it numerically; the reconstruction error should be floating-point noise.

3	4
9	3

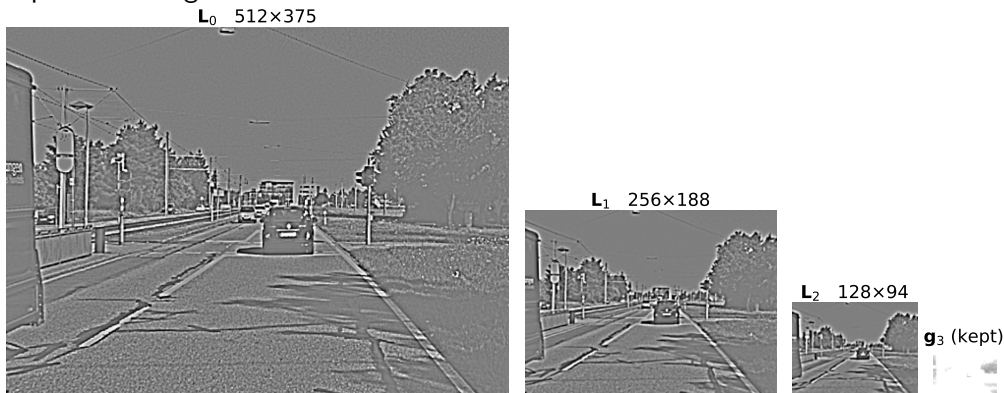


3	0	4	0
0	0	0	0
9	0	3	0
0	0	0	0

* w

The same frame, taken apart

Mid grey is zero. Around **60%** of every level sits within ± 5 of zero, which is why a Laplacian pyramid compresses and a Gaussian one does not. Only the coarsest level, g_3 , is kept as an image.



Keep the levels in floating point. Collapse this pyramid and g_0 comes back bit for bit. Round each level to uint8 on the way and the error is no longer floating-point noise.

Scale and edges

Run an edge detector at every pyramid level and you will find:

- ▶ fine texture disappears first;
- ▶ large structures survive to coarse levels;
- ▶ but thin structures (a lane marking a few pixels wide) vanish after two or three halvings.

In Lab 5 you measure this: recall against pyramid level. Level 0 to level 1 costs very little recall for a **fourfold** reduction in pixels processed. After that it degrades quickly.

The lesson is not “coarser is better”, it is **match the scale to the size of the thing you are looking for**.

Coarse-to-fine

The general recipe, used again for optical flow (Week 8) and stereo (Week 10):

1. Solve the problem at a coarse level, where it is cheap and the search range is small.
2. Upsample the answer.
3. Use it to **initialise** the next finer level, and refine.

The assumption: *nothing appears at the fine scale that left no trace at the coarse scale.*

That assumption is exactly what fails for thin structures, which you will have just measured.

Week 4 recap

- ▶ Derivatives are filters; smooth before differentiating.
- ▶ Gradient magnitude and orientation; orientation is perpendicular to the edge.
- ▶ Canny adds non-maximum suppression and hysteresis.
- ▶ Score edges with recall and a tolerance; never accuracy.
- ▶ Blur before subsampling, or alias.
- ▶ Gaussian pyramid = successive blurred copies; Laplacian pyramid = the differences, and it inverts exactly.
- ▶ Coarse-to-fine is a recurring pattern in this course.

This week's work

- ▶ **Reading:** Szeliski Sec. 3.2.3, 7.2.1, 3.5.
- ▶ **Exercises:** w04a (Sobel by hand), w04b (Taylor stencils), w04c (Laplacian reconstruction).
- ▶ **Lab 4:** gradients, Canny, a region of interest, and recall/selectivity against KITTI ground truth.

Reminder: Midterm 1 is on **Oct 8** and covers Weeks 1–4. Week 5's bilateral filtering is *not* on it.

Copyright and License

©Faisal Z. Qureshi



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.