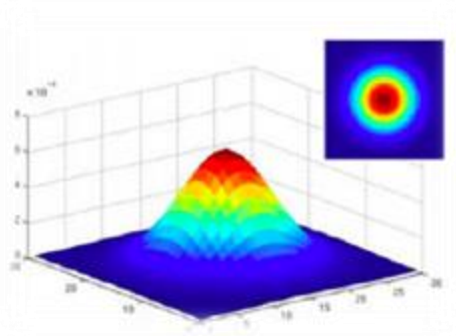
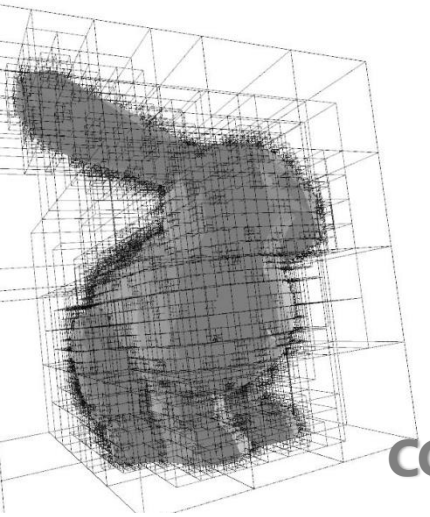
Image-Based Computer Graphics





Fundamentals of Image Processing

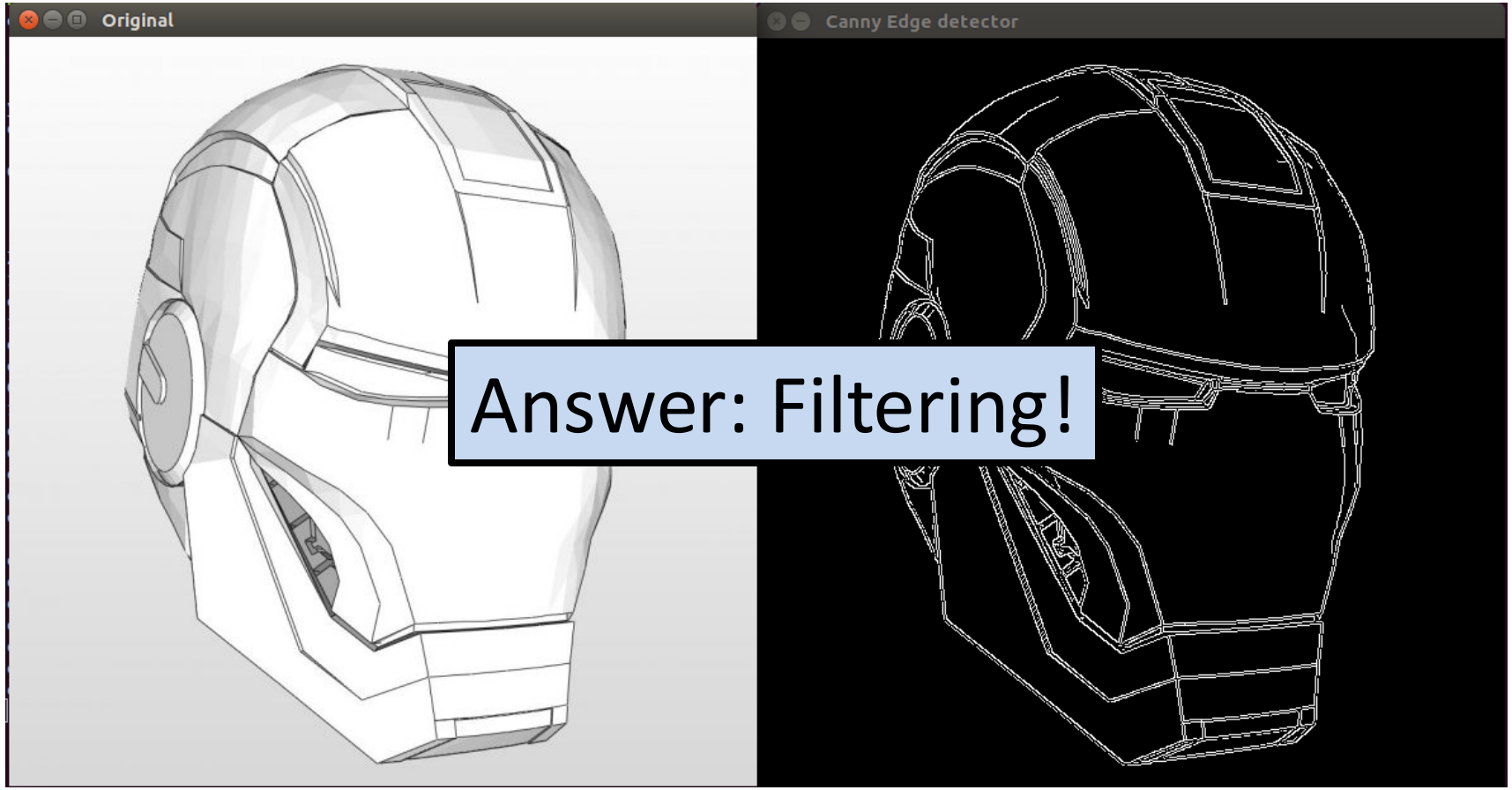
Morphology, Filtering and Pyramids

CGRA 352

How to remove the noises?

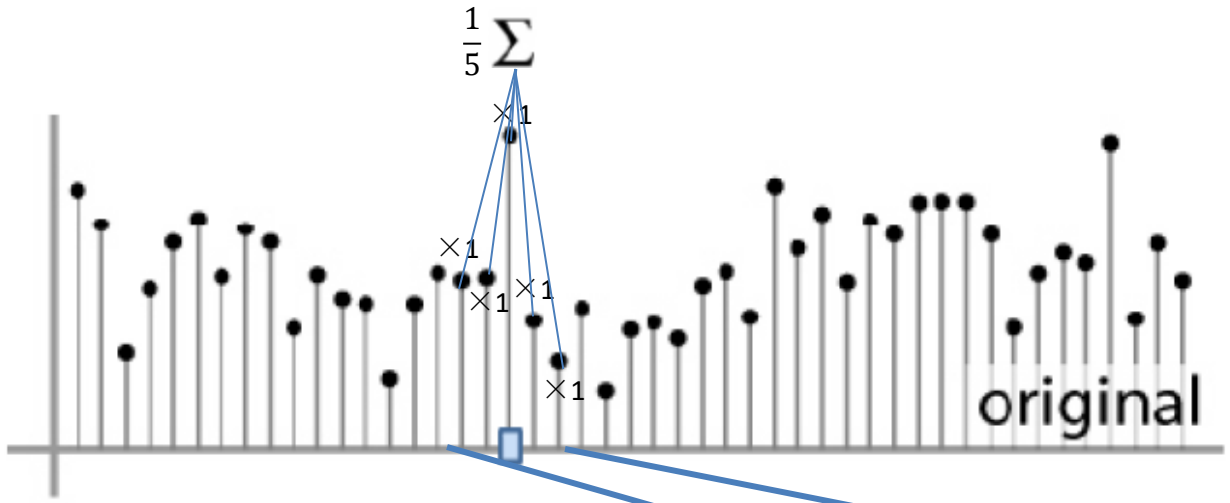
How to find the exactly same bird?

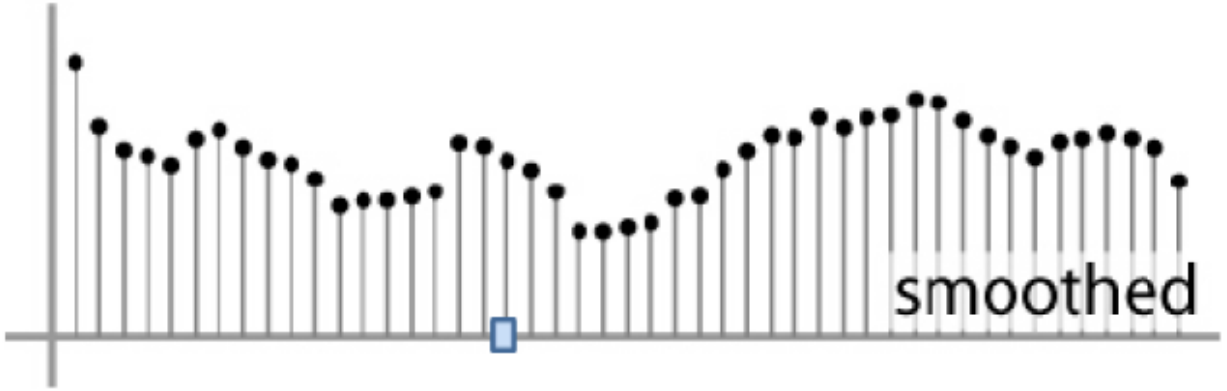
How to extract all these edges?



Denoising

- The simplest thing: replace each pixel by the average of neighbors.
- 1-D example:



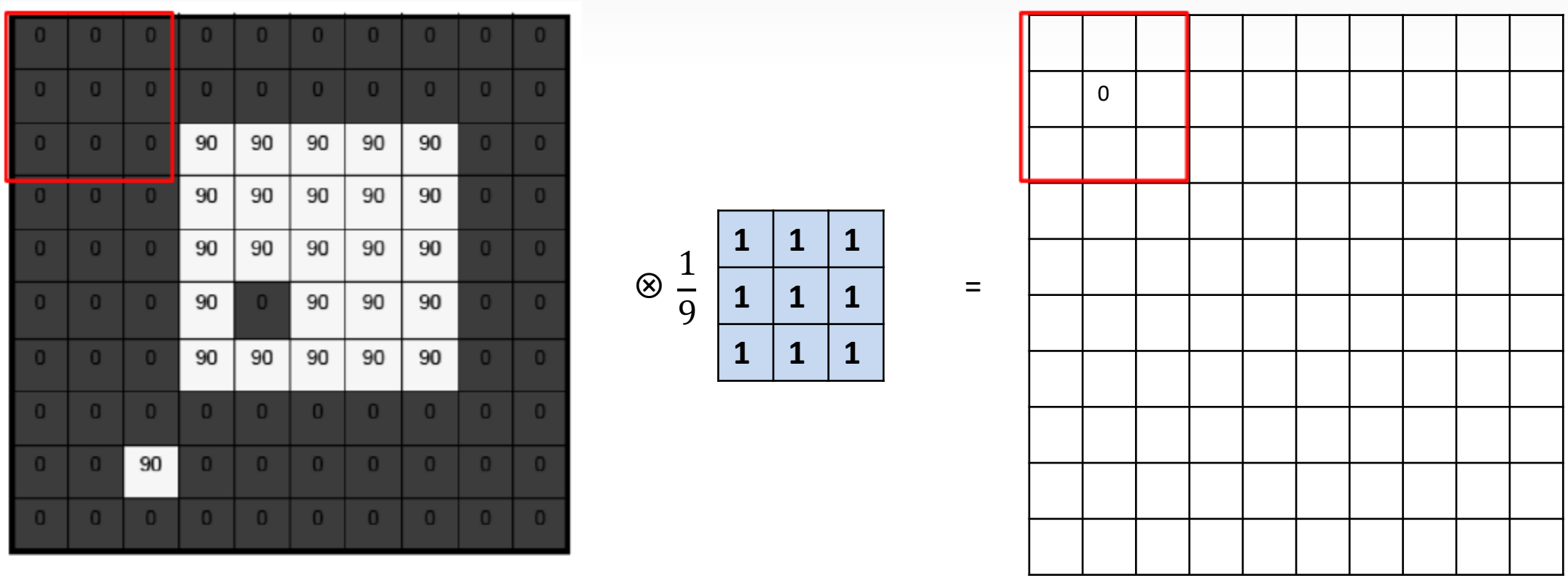


[Image from S. Marschner]

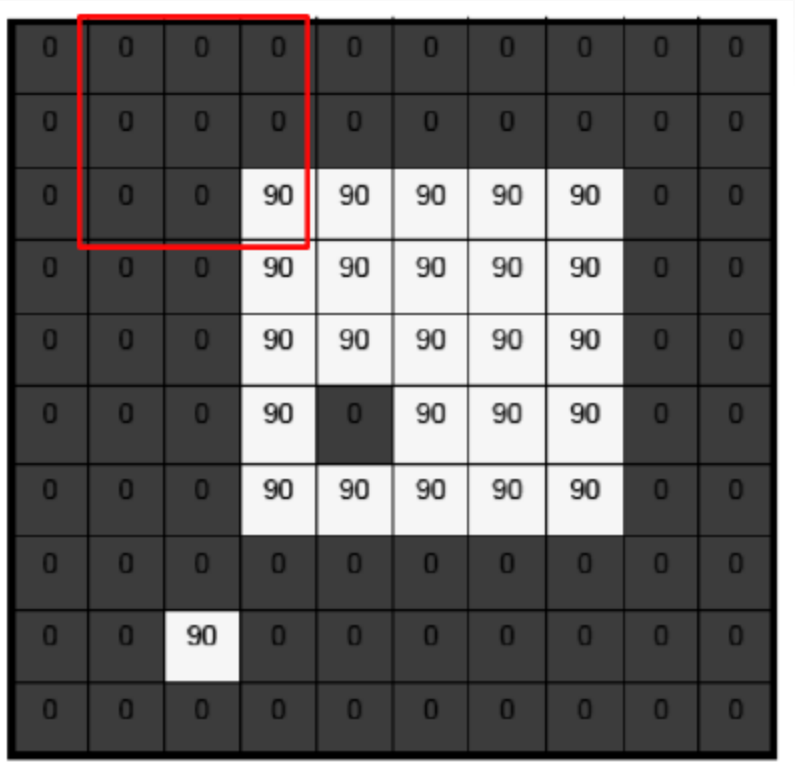
Filtering template $\frac{1}{5}$ 1 1 1 1 1

Center

Averaging in 2D image



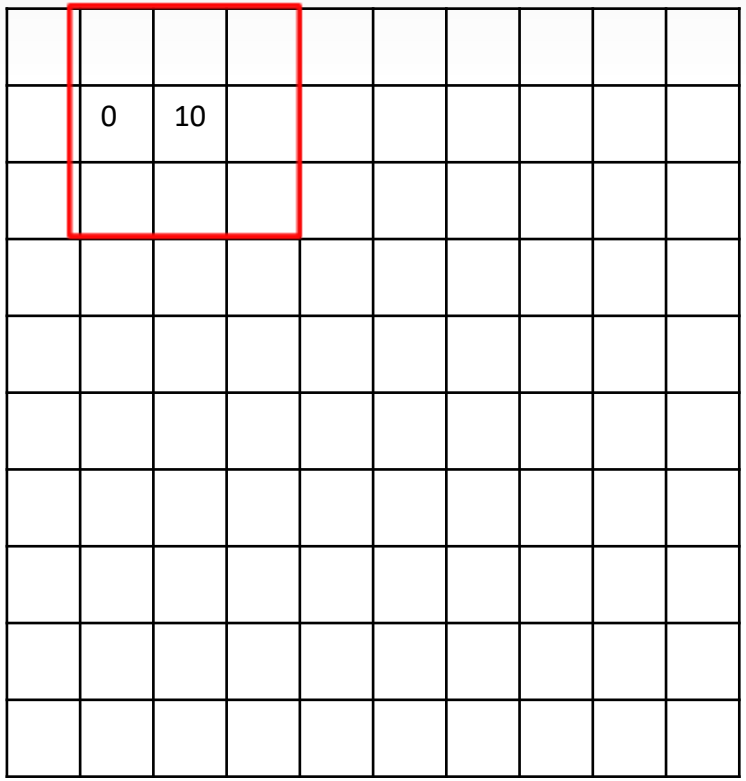
Averaging in 2D image



$\otimes \frac{1}{9}$

1	1	1
1	1	1
1	1	1

=



Averaging in 2D image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

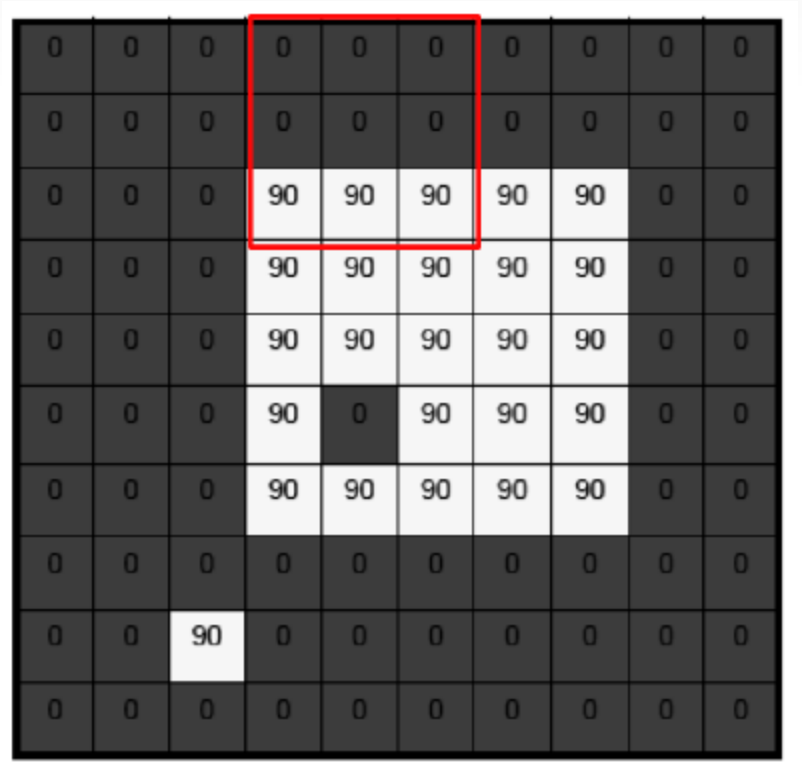
$\otimes \frac{1}{9}$

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

=

	0	10	20						

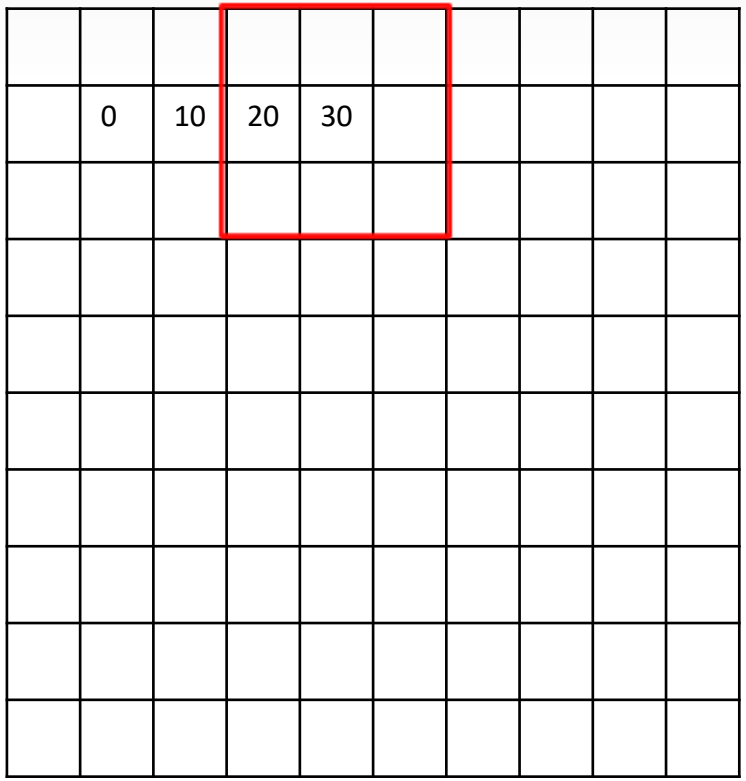
Averaging in 2D image



$\otimes \frac{1}{9}$

1	1	1
1	1	1
1	1	1

=



Averaging in 2D image

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$\otimes \frac{1}{9}$

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

=

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Filtering: Convolution

- For the averaging thing, it is equally weighted combinations of pixels in a small neighboring group:

$$G(i,j) = \frac{1}{(2k+1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} I(i+u, j+v)$$

- For general filtering, it is determined as a weighted sum of input pixel value

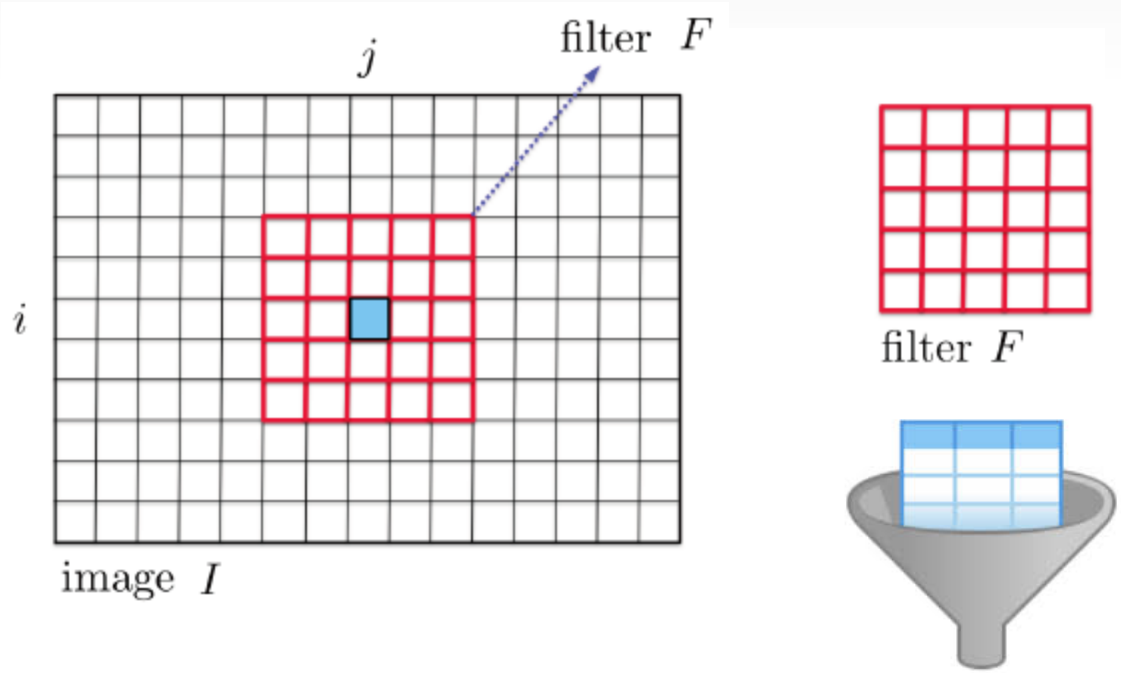
$$G(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v) * I(i+u, j+v)$$

The entries of the **weight kernel** or **mask** *F(u, v)* are called the **filter coefficients.**

This operation is also called **CONVOLUTION**, just as in deep learning

Filtering templates

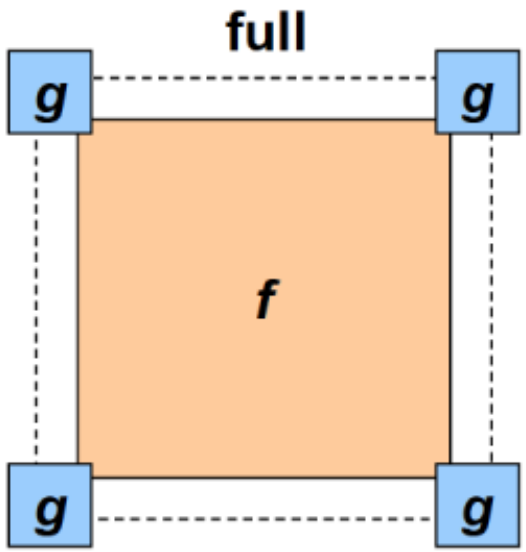
- We can define different templates for different situations

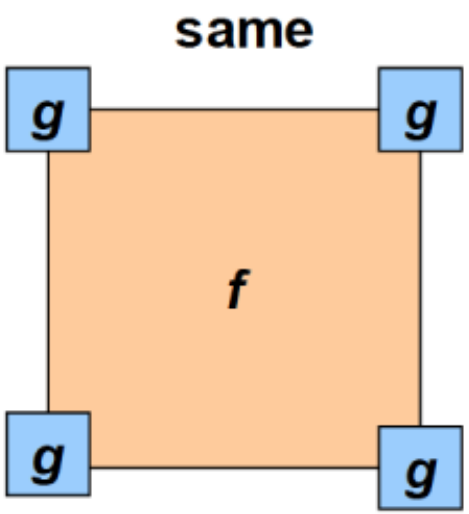


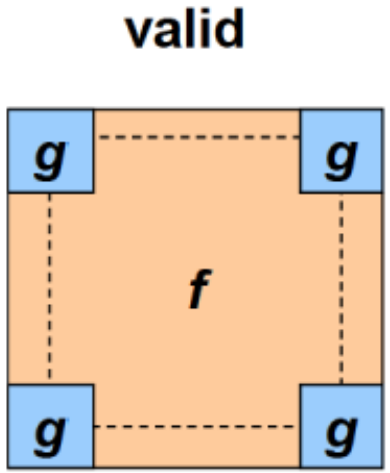
Filter output only one pixel value

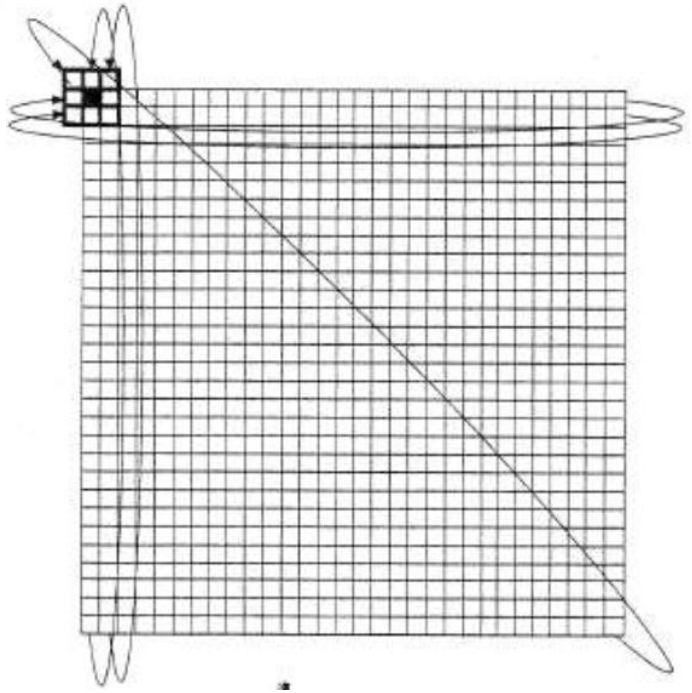
Filtering templates

- What about boundary pixels?
  - Different ways to handle it.



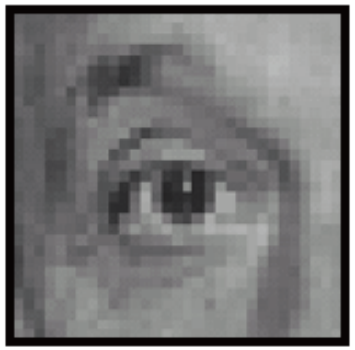






Recycle or just pad with zeros

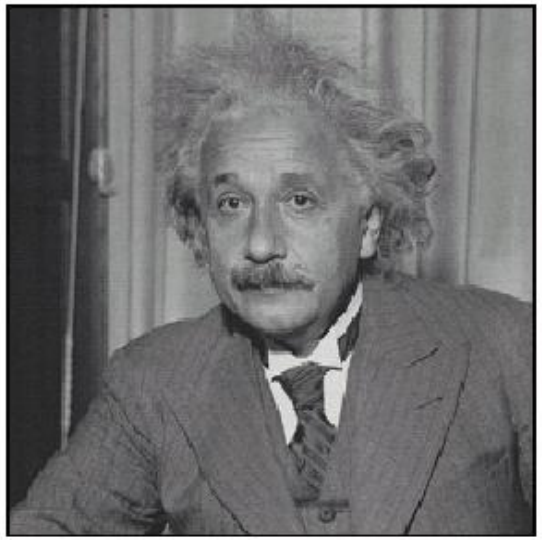
Filtering Applications



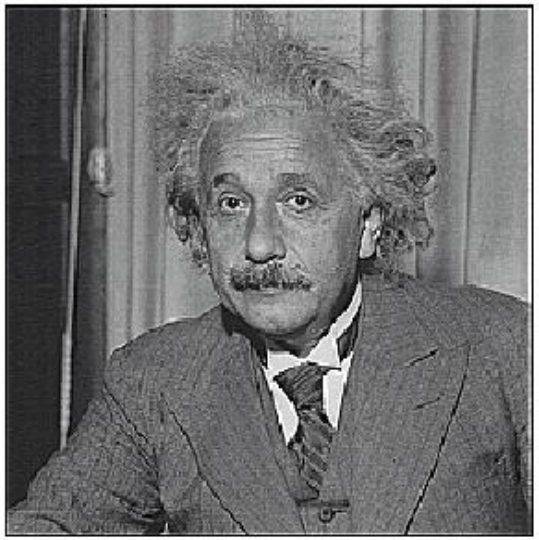
Original

$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) =$$

Sharpening filter
(accentuates edges)



before

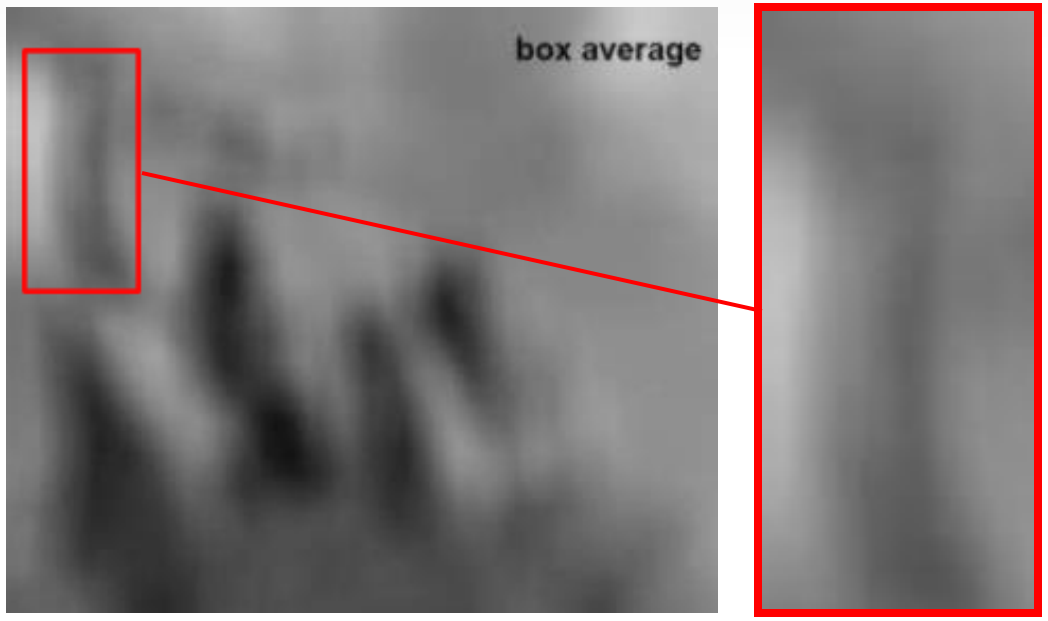


after

[Source: D. Lowe]

Filtering Applications

- Smoothing by averaging (or box averaging)



- We actually have a mathematically correct smoothing filter
  - Gaussian filter

Filtering Applications

- Gaussian filter kernel:
  - 3 by 3 version:

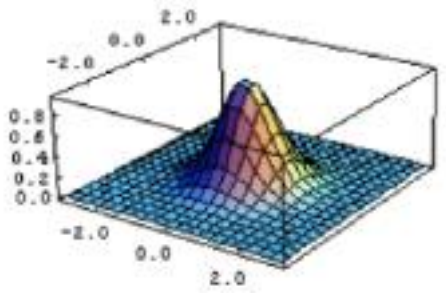
$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$F(i,j)$

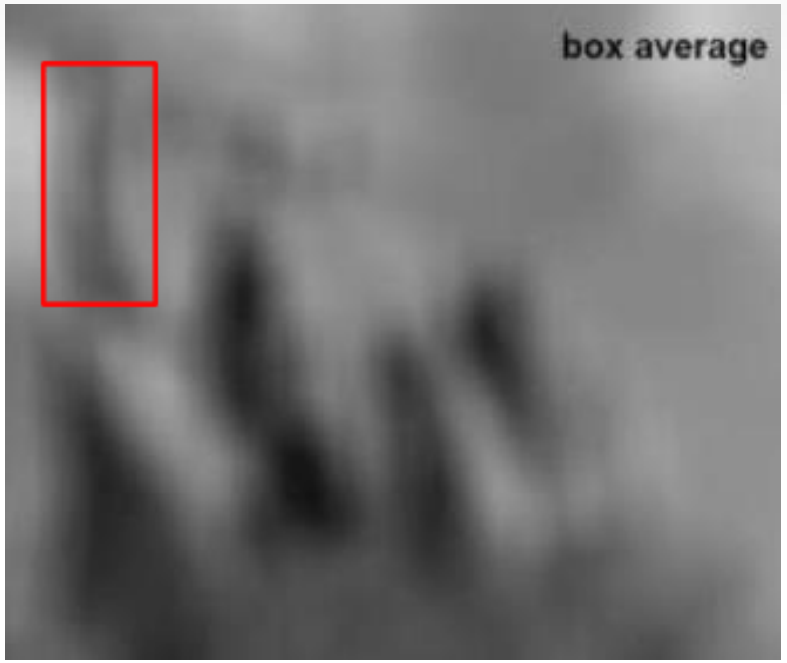
- Removes high-frequency components from the image (low-pass filter)
  - **In Fourier Transform** of an image, the 2D image is treated as a 2D signal
    - decomposed into components in different frequency
    - Sharp edges, where the image changes rapidly, are modeled by high frequencies
    - Applying Gaussian filter can remove the high frequencies, but box filter cannot.

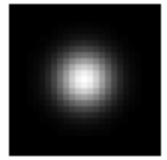
This kernel is an approximation of a 2d Gaussian function:

$$h(u,v) = \frac{1}{2\pi\sigma^2}e^{-\frac{u^2+v^2}{\sigma^2}}$$



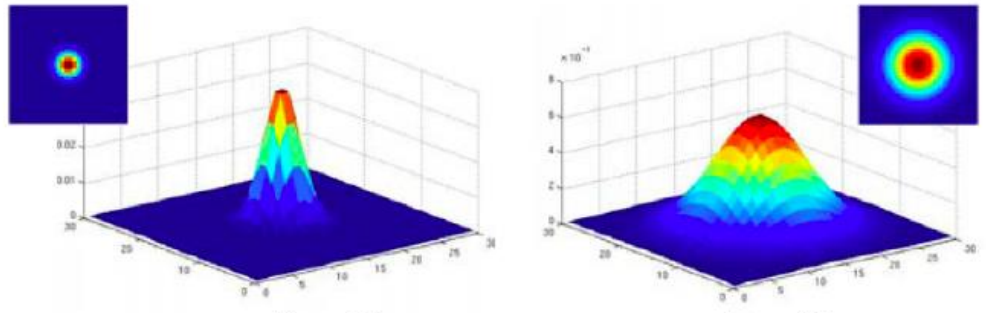
Smoothing by Gaussian





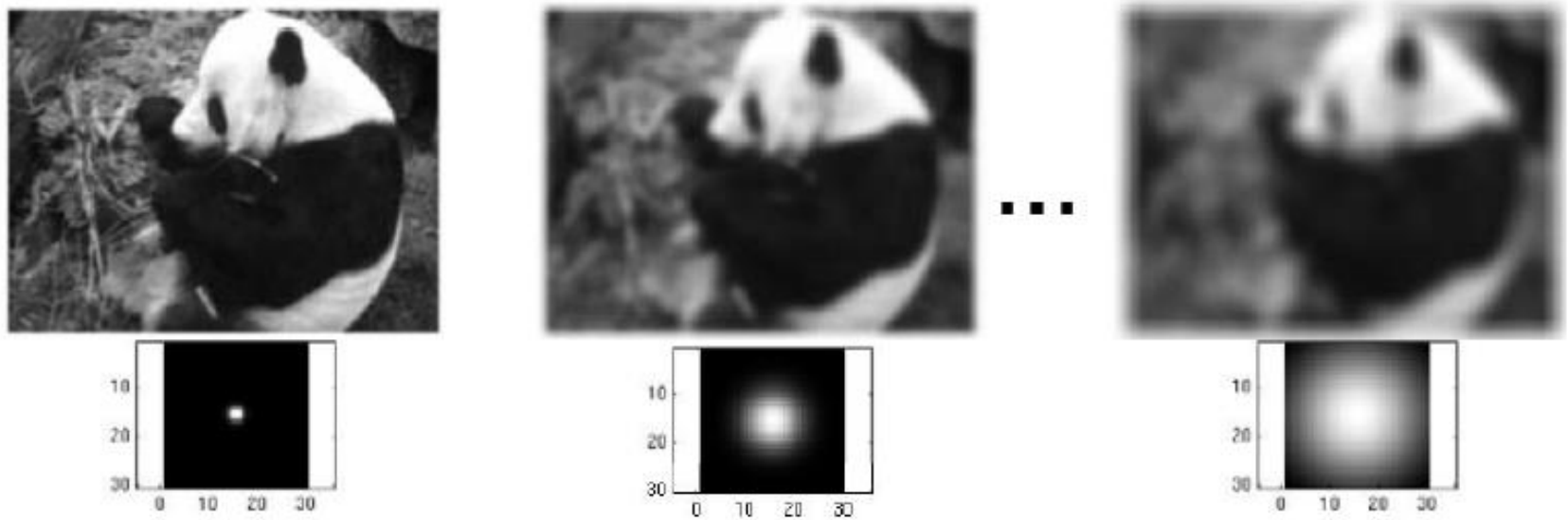
Smoothing by Gaussian

- Different parameters for the Gaussian functions
  - We can get different level of smoothness



σ = 2 with 30 x 30 kernel

σ = 5 with 30 x 30 kernel



[Source: K. Grauman]

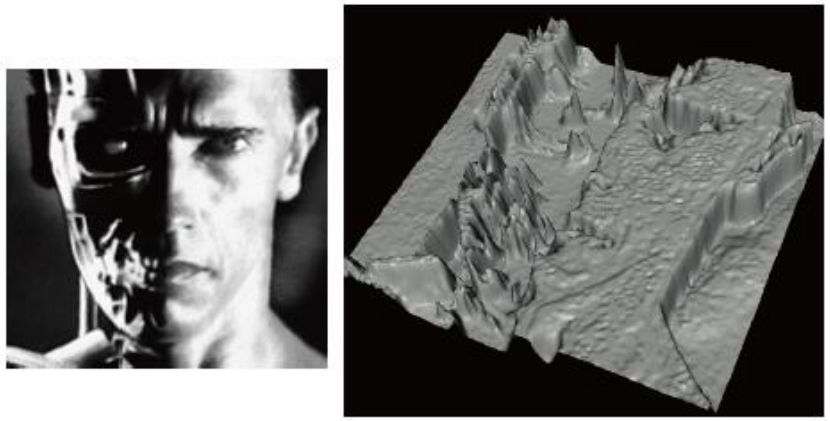
Edge Detection

- Classical definition of the edge detection problem: localization of large local changes in the grey level image → large graylevel gradients
  - Can be extended to color images

- Goal: identify objects in images
  - also feature extraction, 3D reconstruction, motion recognition, image restoration, registration
- Contours are very important perceptual cues!
  - They provide a first saliency map for the interpretation of image semantics

Edge Detection

- What causes an edge?
  - Depth discontinuity, object boundaries
  - Change in surface orientation: shape
  - Cast shadows
- Edges look like steep cliffs



Steep cliffs mean large function value changes

Pixel value changes: Derivatives

- If image f was continuous, then compute the partial derivative as

$$\frac{df(x,y)}{dx} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

- In the discrete domain, 1st-order forward discrete derivative:

$$\frac{df(x,y)}{dx} \approx \frac{f(x+1,y) - f(x,y)}{1}$$

- The partial derivative of image *f(x,y)* with respect to the variable *x* can be computed as:

$$(-1) * f(x,y) + (1) * f(x+1,y)$$

Filter for getting partial derivatives

- It can just be computed by applying the filter:

-1	1

$$(-1) * f(x,y) + (1) * f(x+1,y)$$

- Partial derivative for y:

-1
1

- The horizontal derivative using the filter $[-1,1]$

Filter for getting partial derivatives

- It can just be computed by applying the filter:

-1	1

$$(-1) * f(x,y) + (1) * f(x+1,y)$$

- Partial derivative for y:

-1
1

- The vertical derivative using the filter $[-1,1]^T$

Filter for getting partial derivatives

-1 | 1

-1
1

Other approximations

- Other templates for getting approximated derivatives

Roberts

$$\begin{pmatrix} [0] & 1 \\ -1 & 0 \end{pmatrix} \quad \begin{pmatrix} [1] & 0 \\ 0 & -1 \end{pmatrix}$$

Central Difference

$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & [0] & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & -1 & 0 \\ 0 & [0] & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Prewitt

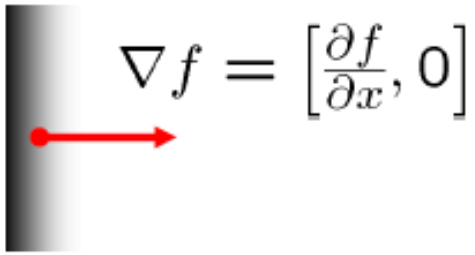
$$\begin{pmatrix} -1 & 0 & 1 \\ -1 & [0] & 1 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -1 & -1 \\ 0 & [0] & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

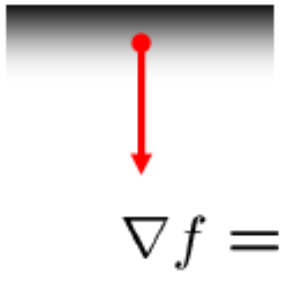
Sobel

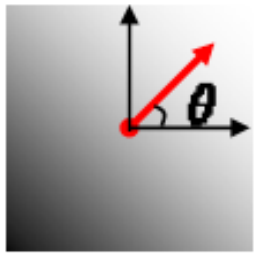
$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & [0] & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} -1 & -2 & -1 \\ 0 & [0] & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Image Gradient

- The gradient of an image: $\nabla f = [\frac{df}{dx}, \frac{df}{dy}]$
- The gradient points in the direction of most rapid change in intensity







- The gradient direction (orientation of edge normal) is given by:

$$\theta = \tan^{-1}(\frac{df}{dy} / \frac{df}{dx})$$

Image Gradient

- The **edge strength** is given by the magnitude:

$$||\nabla f|| = \sqrt{\left(\frac{df}{dx}\right)^2 + \left(\frac{df}{dy}\right)^2}$$

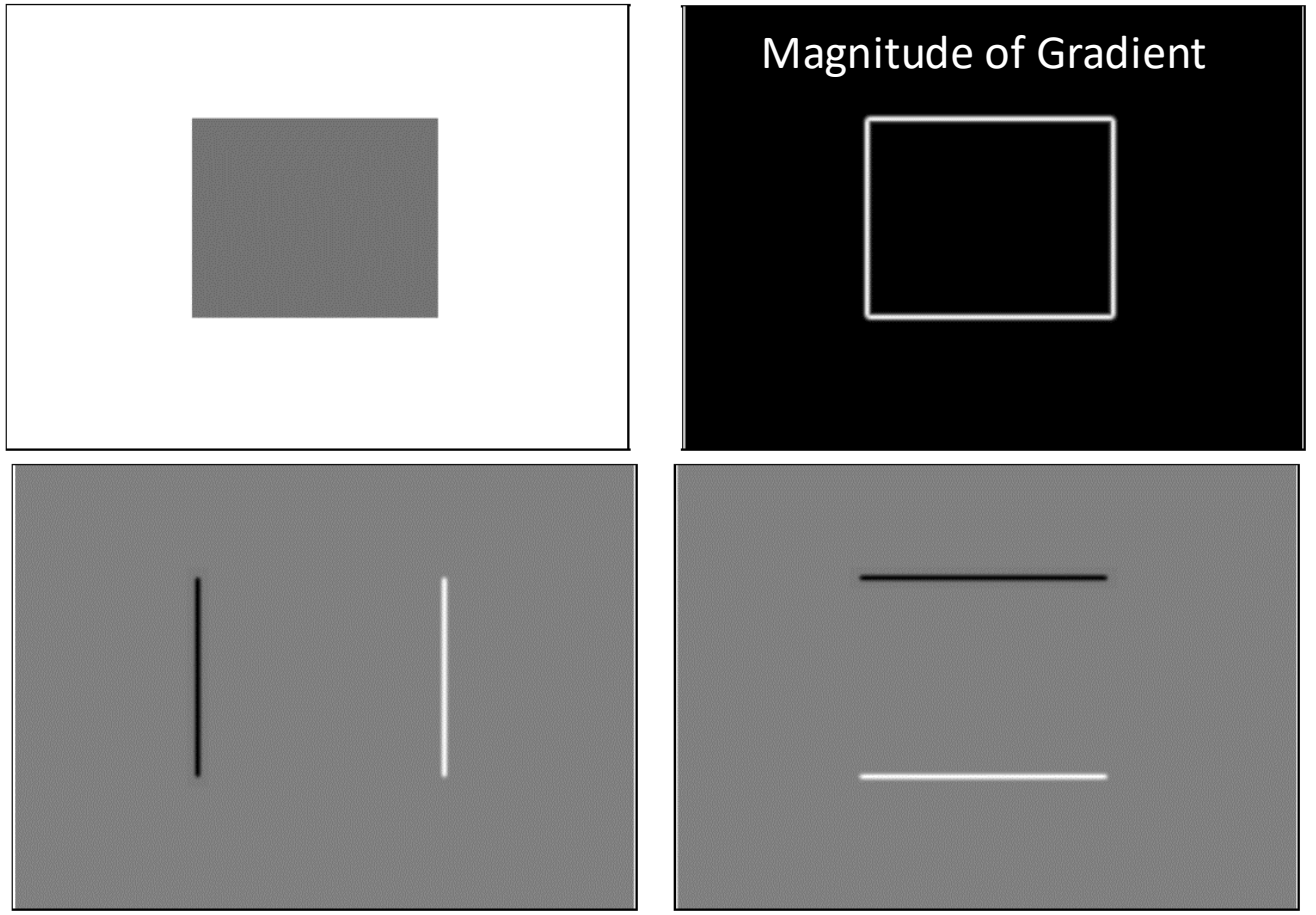


Image Gradient

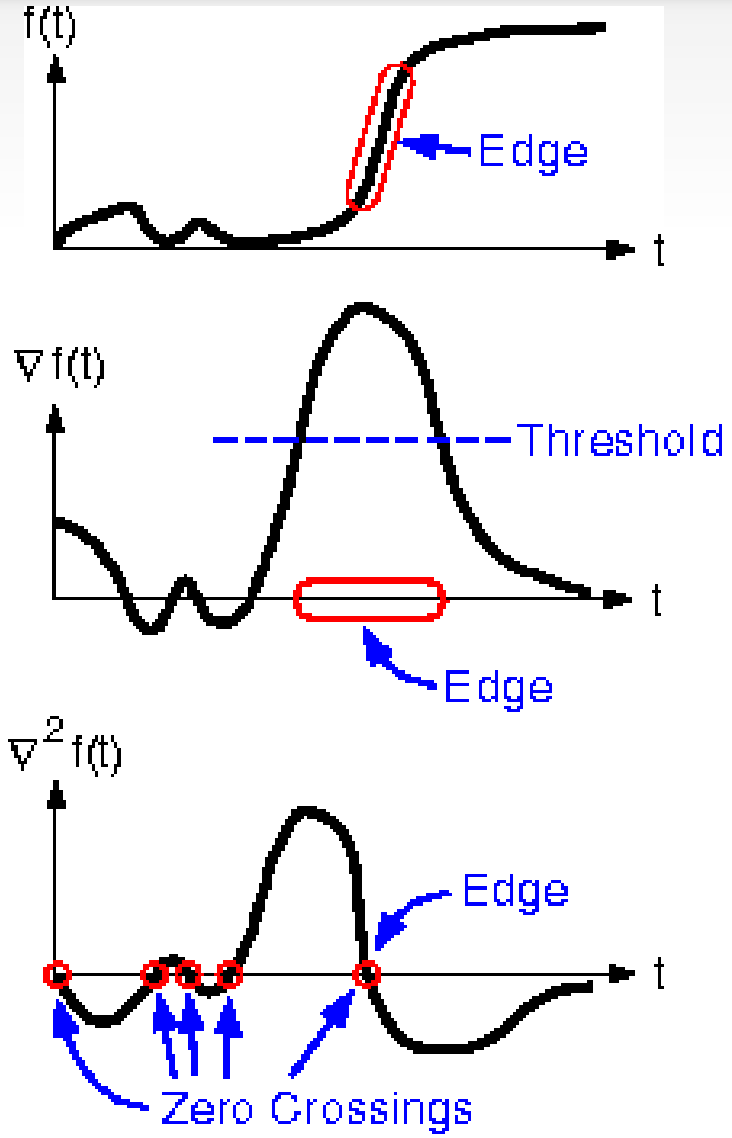
- Thresholding gradient magnitude to get edges:

Gradient Magnitude
(Gradient Map)

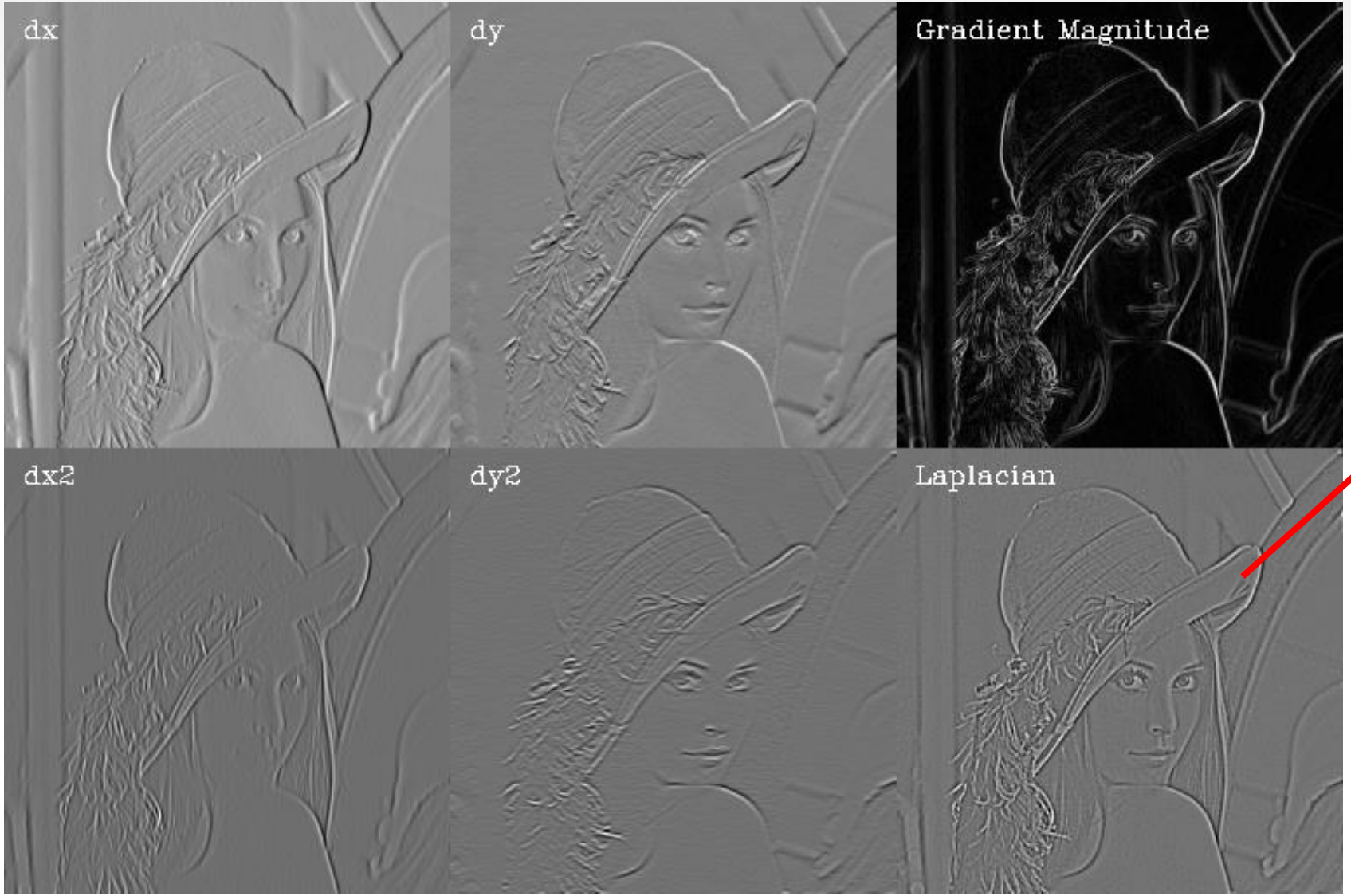
Edges

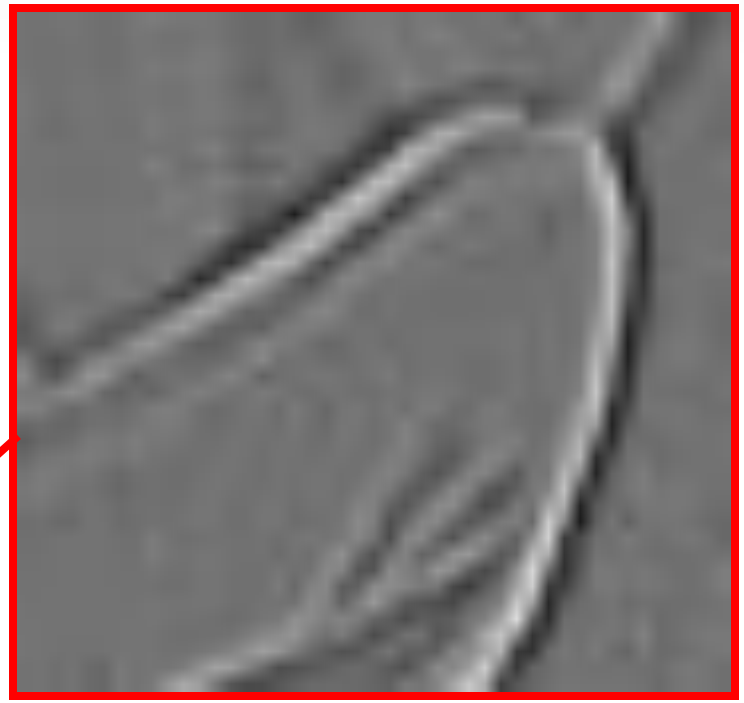
Laplacian filter – second derivatives

- However, directly thresholding the gradient magnitude, will get a “thick” edge
- Laplacian filter is for detecting edge by second derivative
- Zero crossing points will be the edge



Laplacian filter





Laplacian filter

First order derivative: $\frac{df(x)}{dx} = f(x+0.5) - f(x-0.5)$ ⇨ $\frac{df(x)}{dx} = f(x+0.5) - f(x-0.5)$

Second order derivative: $\frac{d\left(\frac{df(x)}{dx}\right)}{dx} = f(x+1) - 2f(x) + f(x-1)$

$$It\ is\ I(x+1) + I(x-1) - 2I(x)$$

Laplace filter

1	-2	1

For 2D

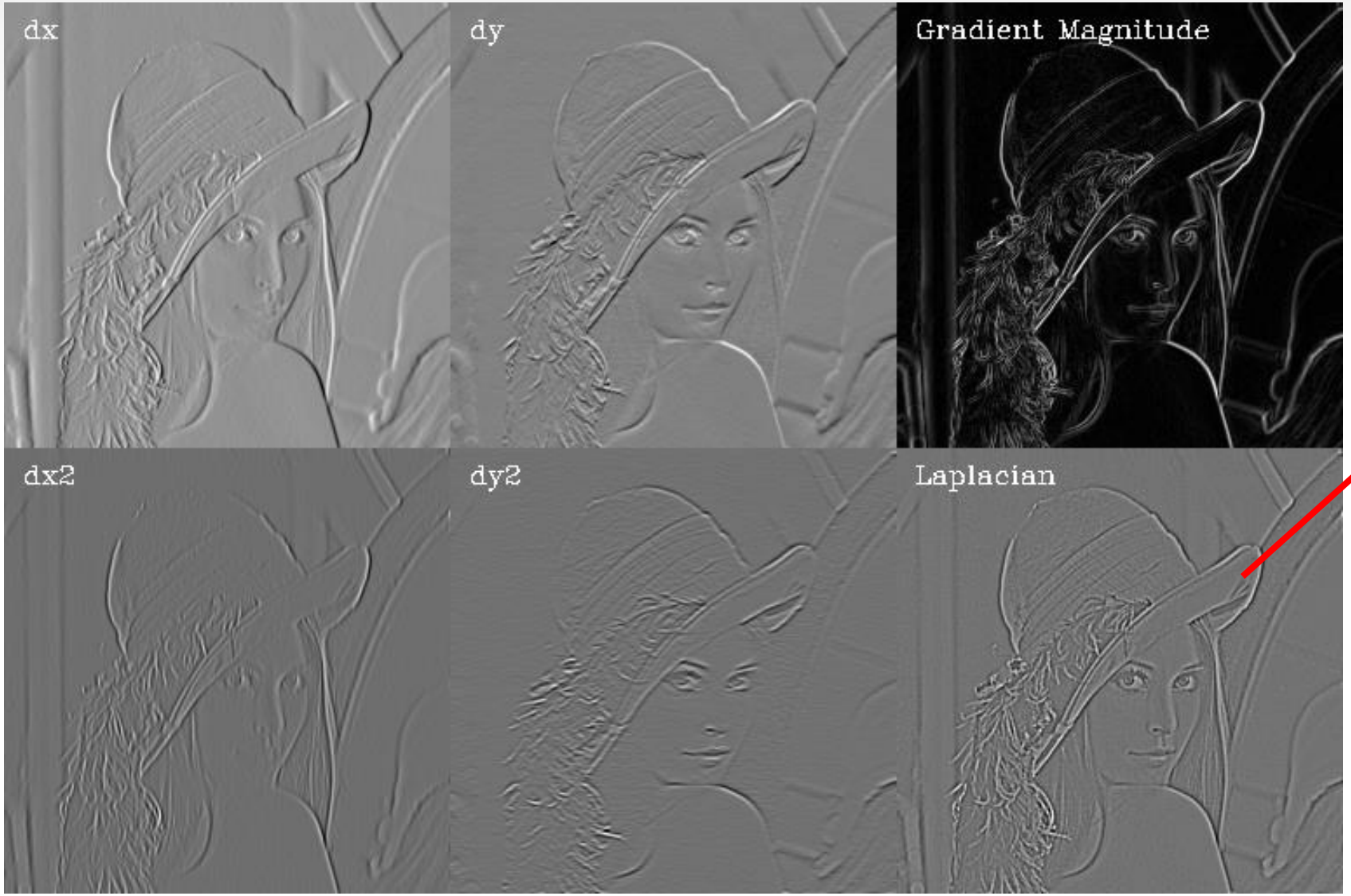
$$\Delta f = \frac{df^2}{d^2x} + \frac{df^2}{d^2y} = I(x+1,y) + I(x-1,y) - 2I(x,y) + I(x,y+1) + I(x,y-1) - 2I(x,y)$$

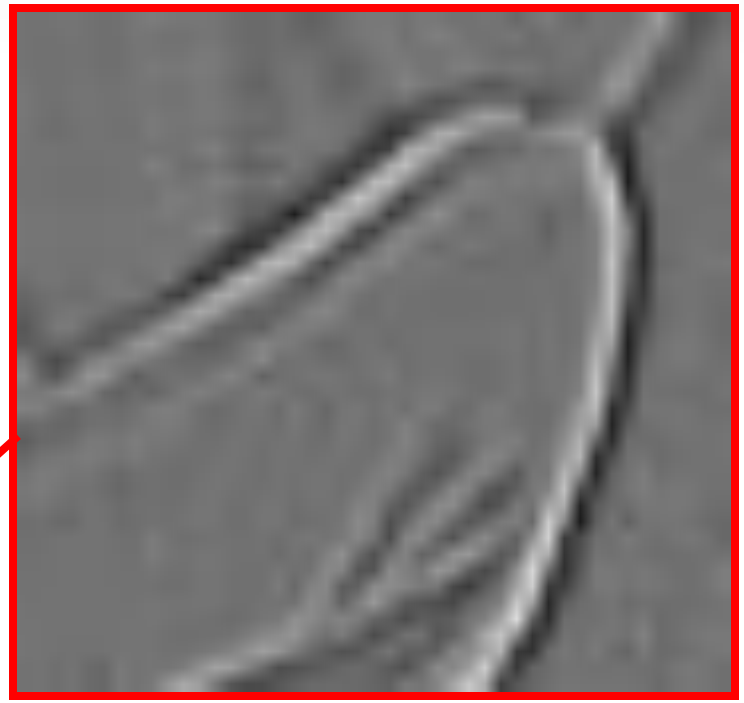
$$= I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y)$$

0	1	0
1	-4	1
0	1	0

2D Laplace filter

Laplacian filter





Today's topics

- Edge Detection
- Template Matching

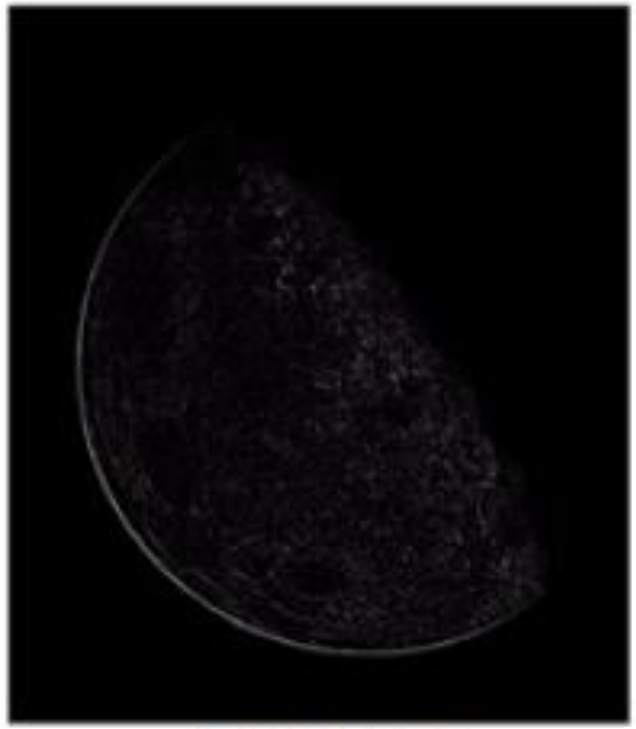
Laplacian filter

Filtered Result
(Absolute value visualisation)

Laplacian filter

- A simple enhancement using Laplacian filter: Sharpen

Original Image



Laplacian Filtered Image

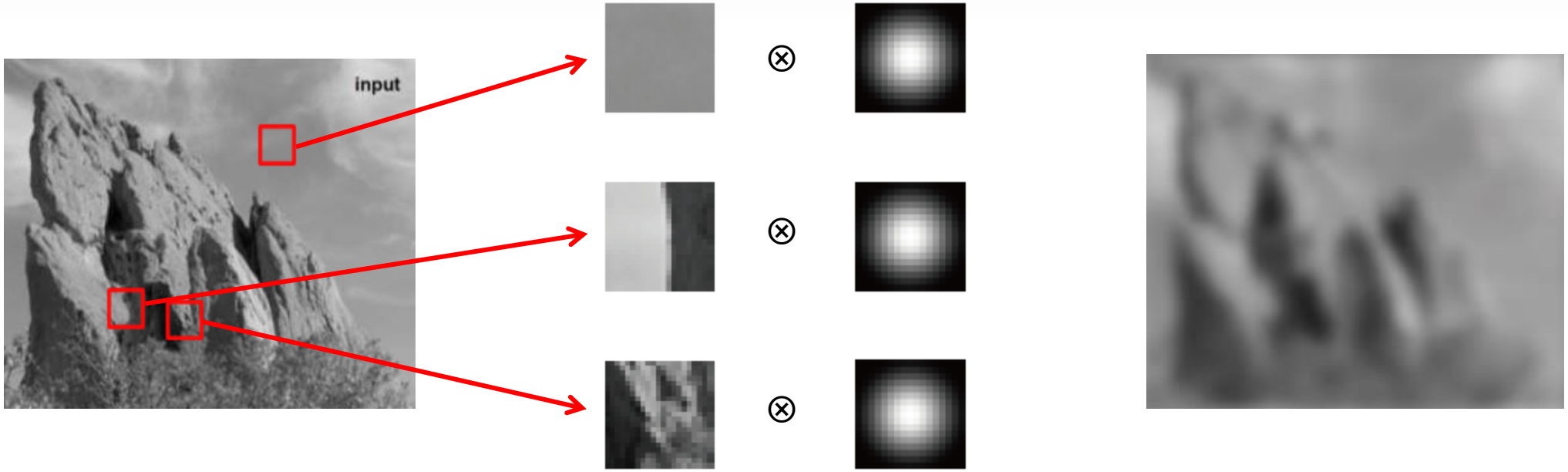
0	1	0
1	-4	1
0	1	0

2D Laplace filter

Sharpened Image

Bilateral filtering

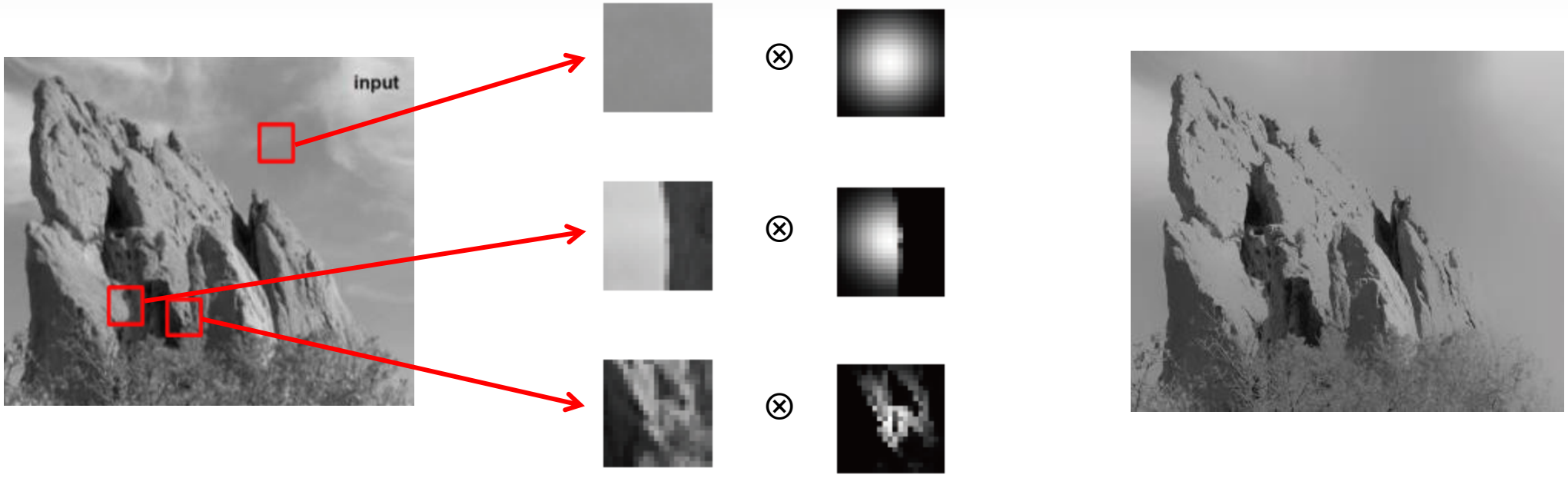
- Blur Comes from Averaging across Edges for Gaussian Filter



Same Gaussian kernel everywhere.

Bilateral filtering

- Bilateral Filter: No Averaging across Edges



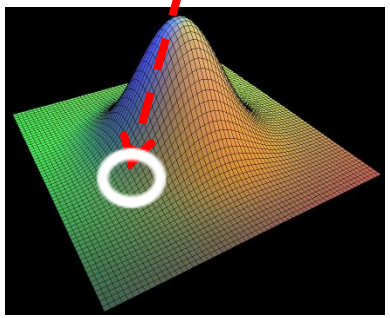
The kernel shape depends on the image content.

Bilateral Filter Definition

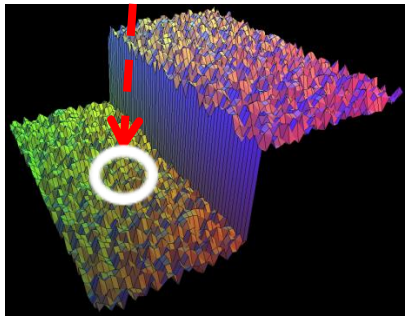
- Same idea: **weighted average of pixels**

$$W_{\boldsymbol{p}} = \sum_{\boldsymbol{q}\in N} G_s(||\boldsymbol{p}-\boldsymbol{q}||)G_r\left(\left||I_p - I_q\right|\right)$$

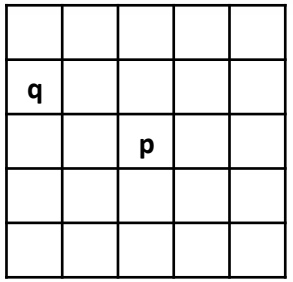
$$B(\boldsymbol{p}) = 1/W_{\boldsymbol{p}} \sum_{\boldsymbol{q}\in N} G_s(||\boldsymbol{p}-\boldsymbol{q}||)G_r\left(\left||I_p - I_q\right|\right) I_{\boldsymbol{q}}$$

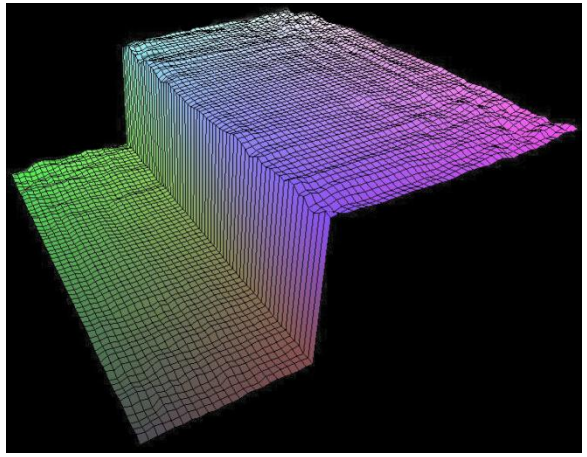


Spatial distance weight

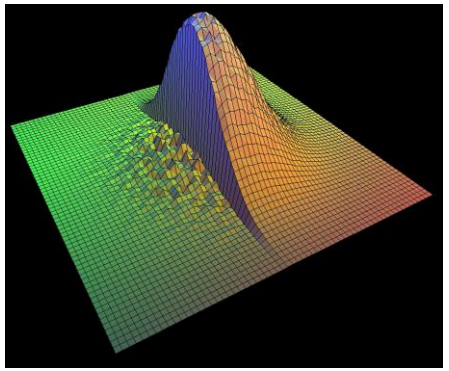


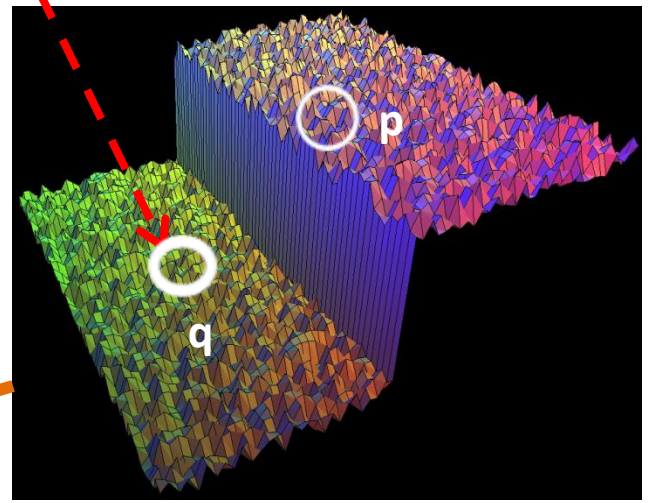
Color difference weight





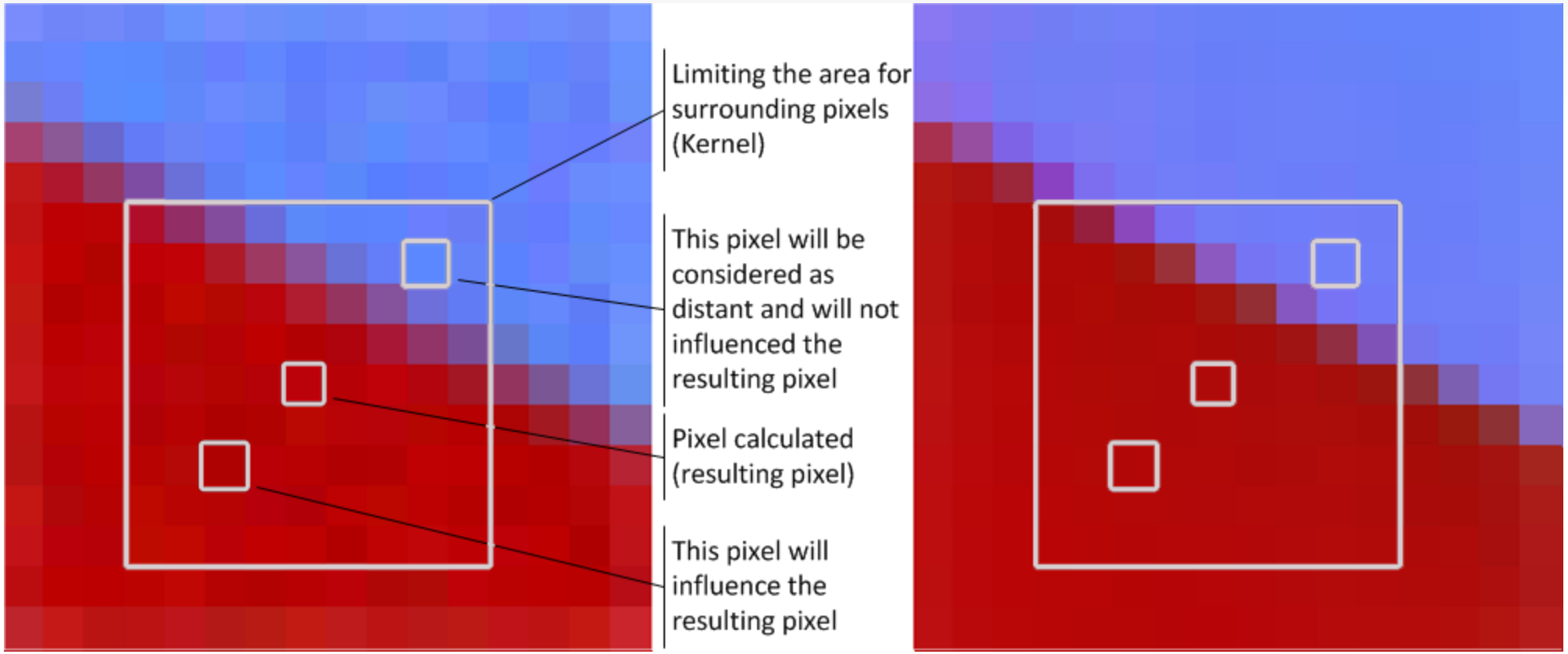
Output





Input

Bilateral Filter



Output

Edge Detector

Canny edge detector

The Process of Canny edge detection algorithm can be broken down to 4 steps:

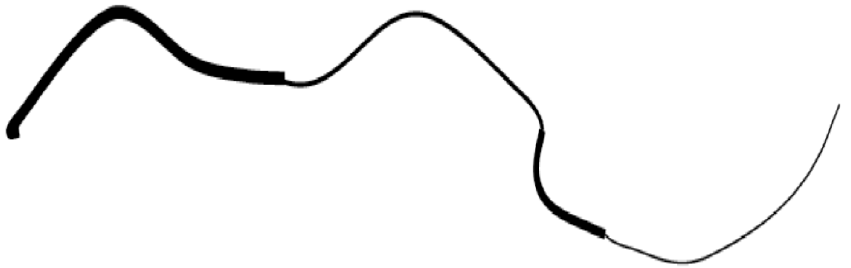
- Apply Gaussian filter to smooth the image in order to remove the noise
- Find the intensity gradients of the image, and apply non-maximum suppression to get rid of spurious response to edge detection

(You can directly use zero-crossing points after Laplacian filtering to do the same thing)

- Apply double threshold to determine potential edges
- Track edge by hysteresis: Finalize the detection of edges by suppressing all the other edges that are weak and not connected to strong edges.

Why double thresholding and hysteresis?

Track edges by hysteresis (depending on the system's history):
Use a high threshold to start edge curves, and a low threshold to continue them



[Source N. Snavely]

original image

high threshold
(strong edges)

low threshold
(weak edges)

hysteresis threshold

[Source Fei-Fei]

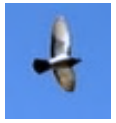
Bilateral Filter Result

Input

Output

Template Matching

- Problem: given a small image, how to find a most similar region with it?



30	50	80
20	30	50
20	30	30

Where is it?

0	10	20	30	30	30	20	10
0	20	40	60	60	60	40	20
0	30	60	90	90	90	60	30
0	30	50	80	80	90	60	30
0	30	50	80	80	90	60	30
0	20	30	50	50	60	40	20
10	20	30	30	30	30	20	10
10	10	10	0	0	0	0	0

Template Matching

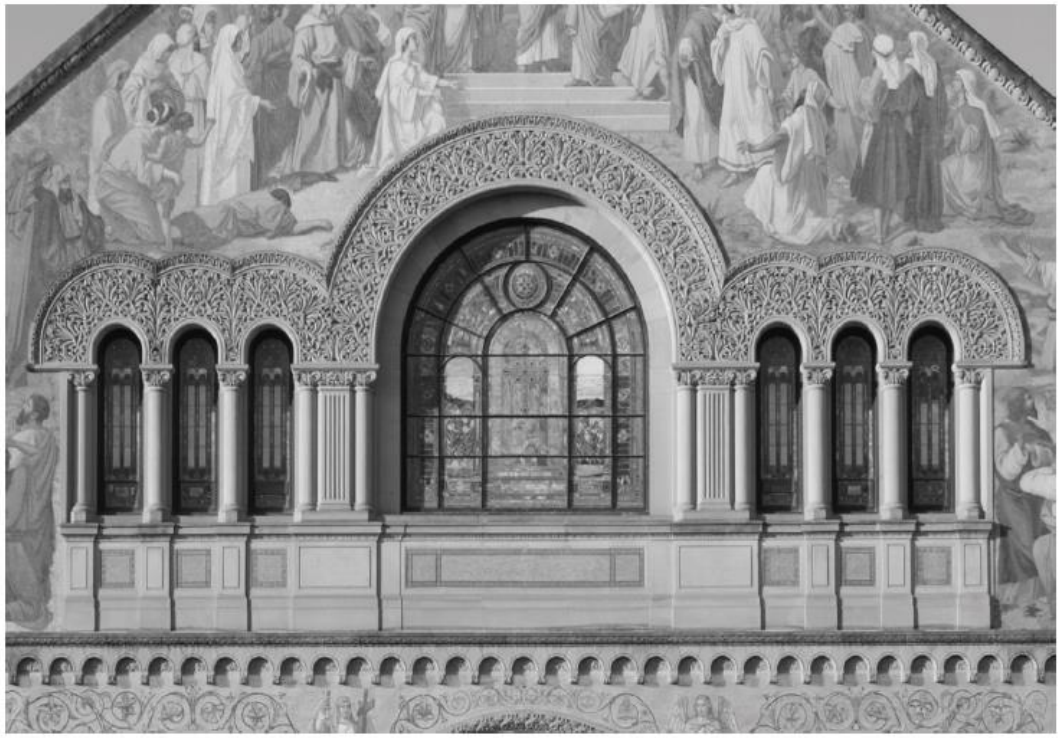
- How to measure the similarity between two patches?
  - Is SSD (Sum of Squares Distance) a good choice?
  - We may only care about the structural or geometrical features
  - Think of two patches as two high-dimensional vectors, we are measuring the similarity of the two vectors!

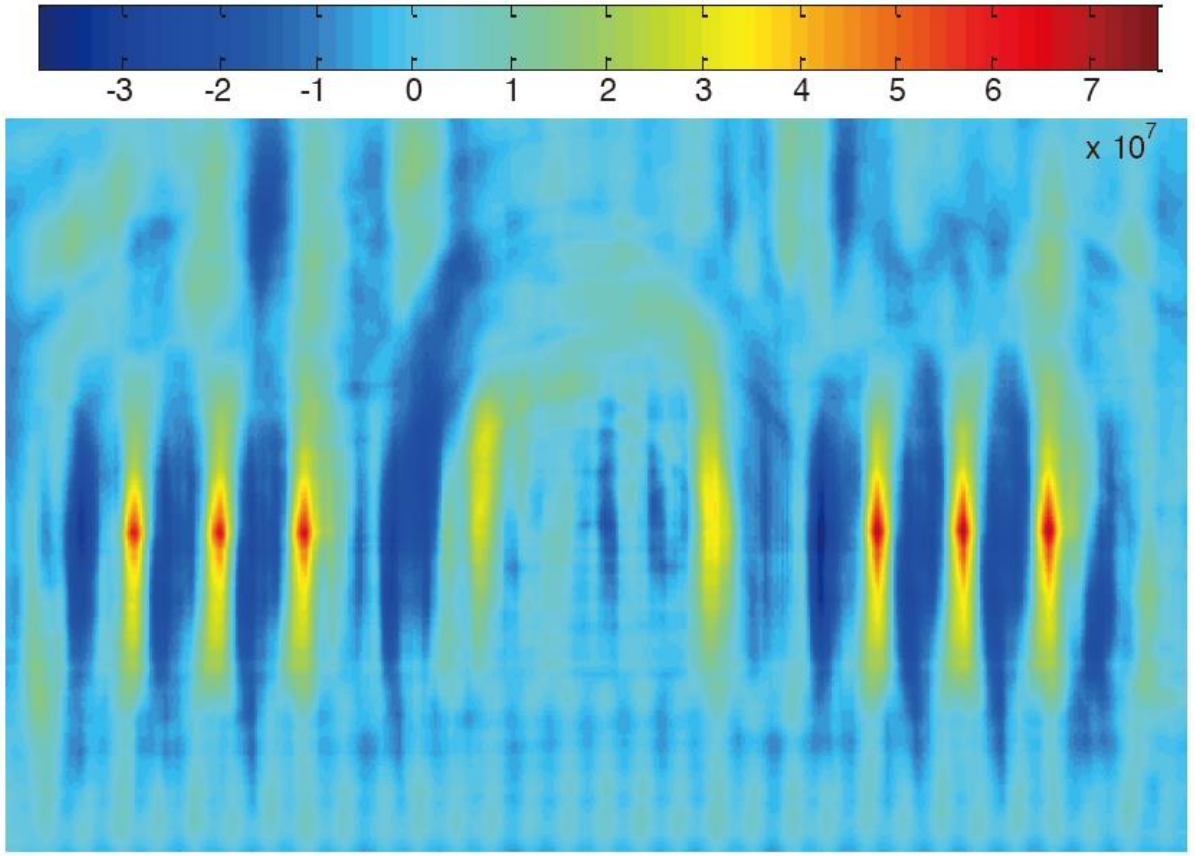
- Taking the small image as the filtering template, and doing **normalized cross correlation (NCC)**

$$C(i,j) = \frac{1}{|F||I_{i,j}|} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F(u,v) * I(i+u, j+v)$$

Where, $|I_{i,j}| = \sqrt{\sum_{u,v \in [-k,k]} I(i+u, j+v)^2}$

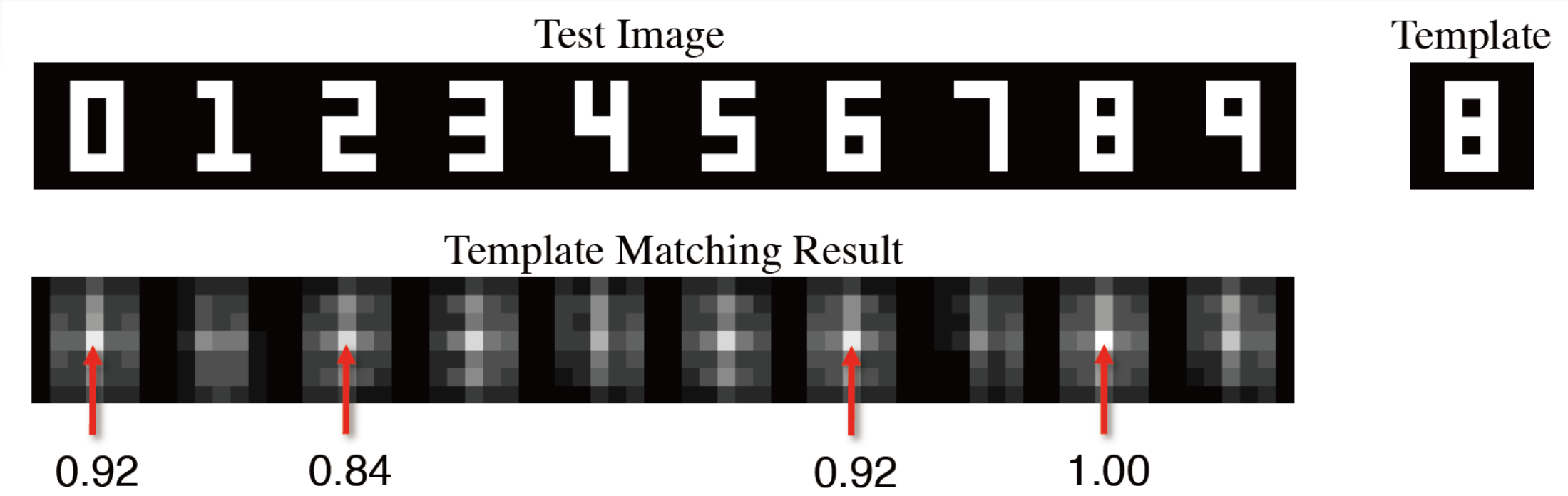
Template Matching





Red spots show the candidate positions

Template Matching



Convolutional neural networks

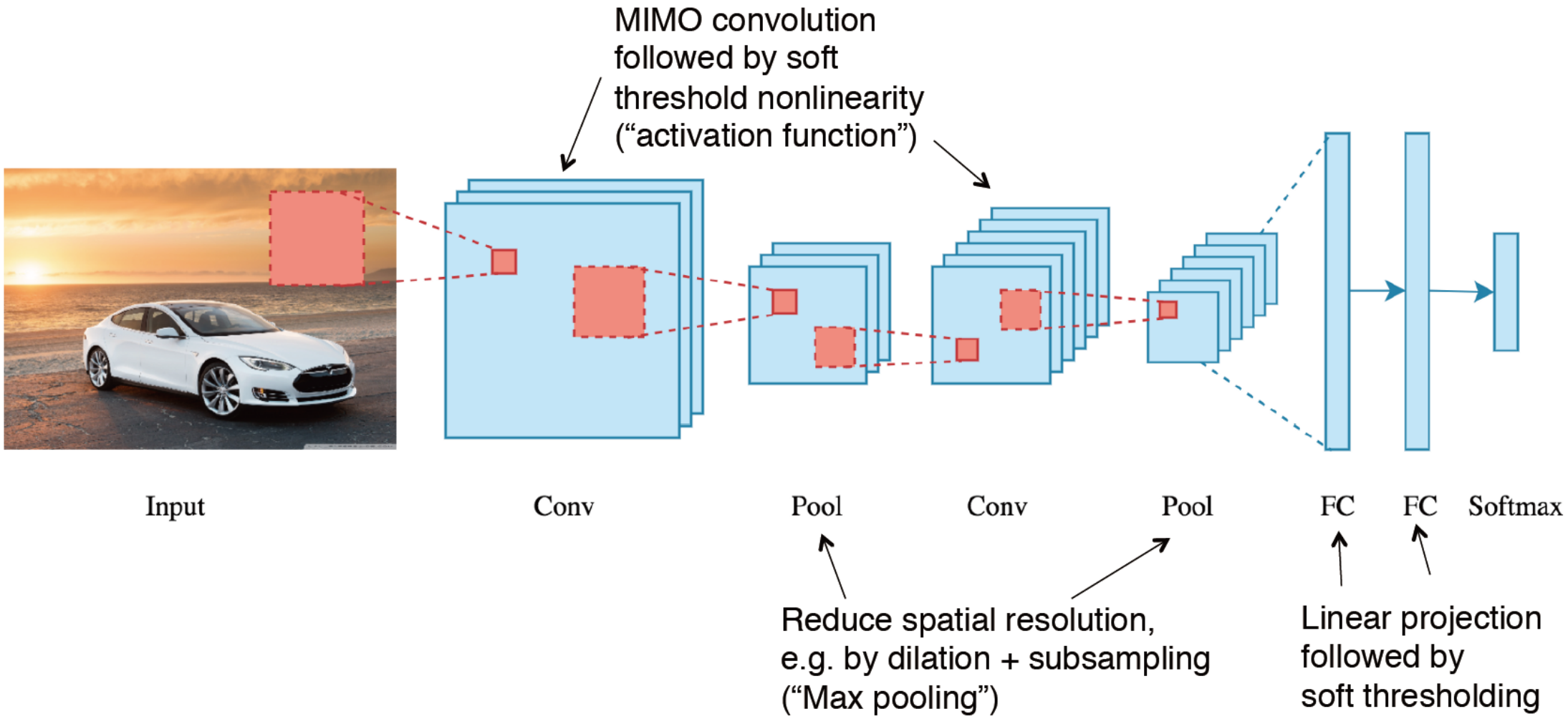
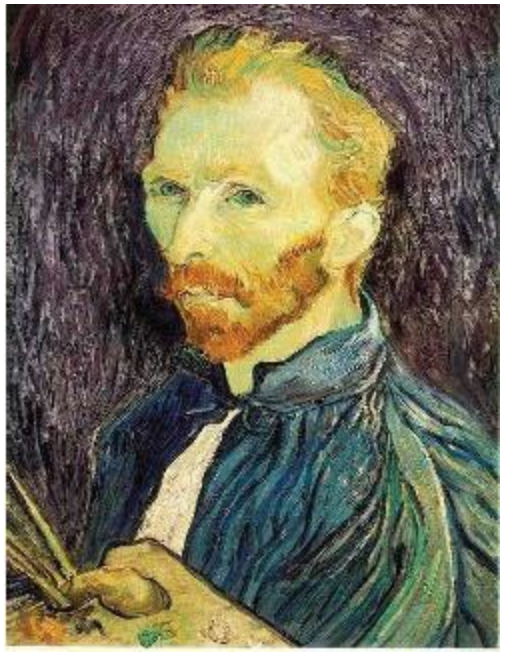


Image Sub-Sampling

- Idea: Throw away every other row and column to create a ½ size image



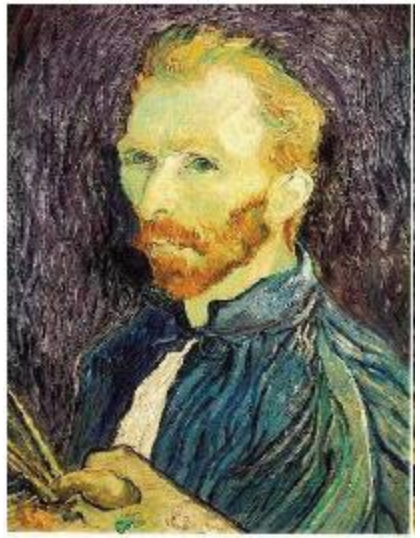
1/2

1/4

[Source: S. Seitz]

Image Sub-Sampling

- Why does this look so bad?



1

1/2 (2x zoom)

1/4 (4x zoom)

[Source: S. Seitz]

Even worse for synthetic images

- Resize this image by factor 2
- If take every other column and every other row(1st, 3rd,5th,etc)

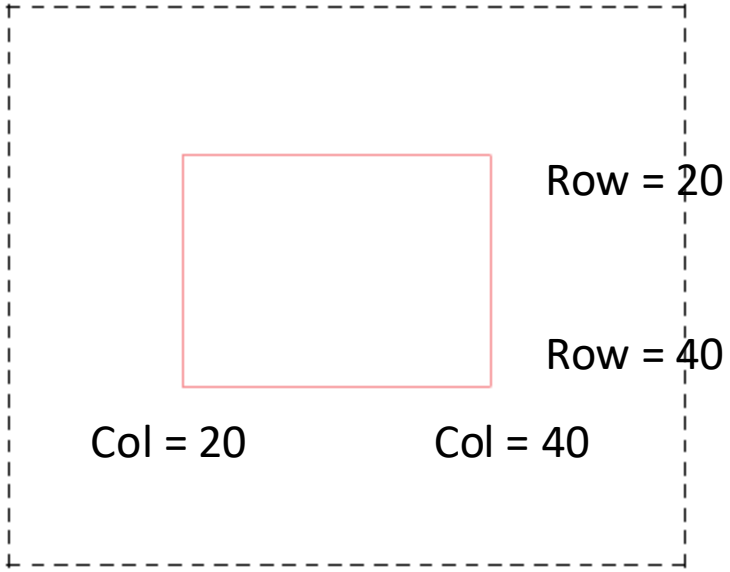


Figure: Dashed line denotes the border of the image (it's not part of the image)

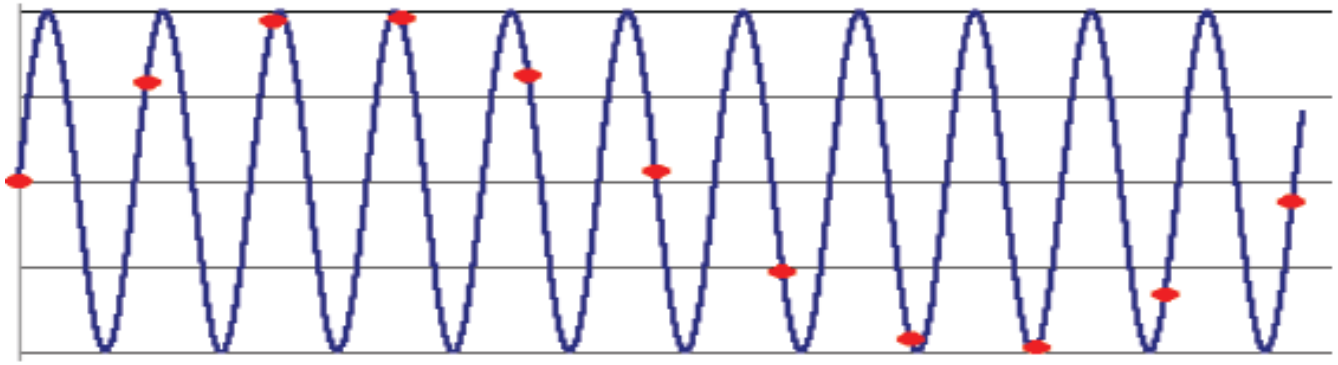
Even worse for synthetic images

- Resize this image by factor 2
- If take every other column and every other row(1^{st}, 3^{rd},5^{th},etc)
- Where is the red rectangle?

Figure: Dashed line denotes the border of the image (it's not part of the image)

Aliasing

- Occurs when your sampling rate is not high enough to capture the amount of detail in your image

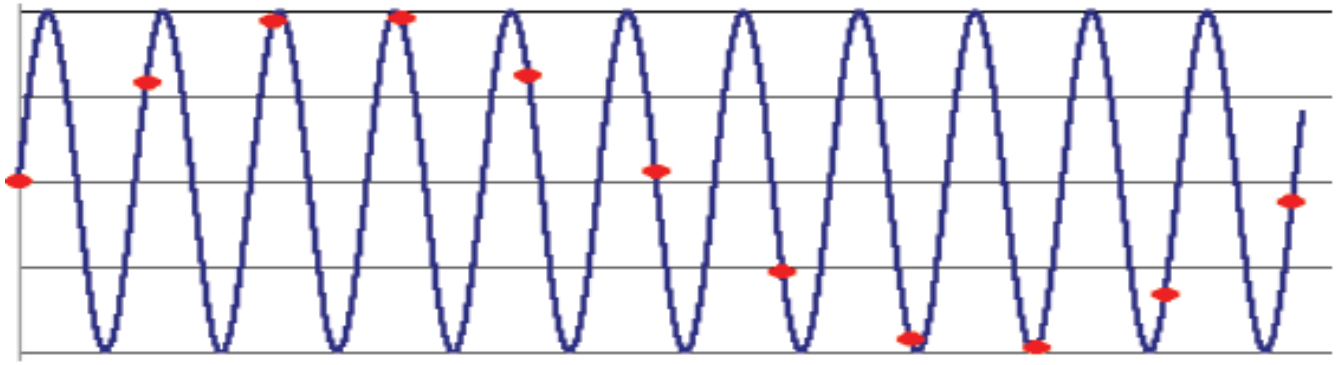


- To do sampling right, need to understand the structure of your signal/image

[Source: R. Urtasun]

Aliasing

- Occurs when your sampling rate is not high enough to capture the amount of detail in your image

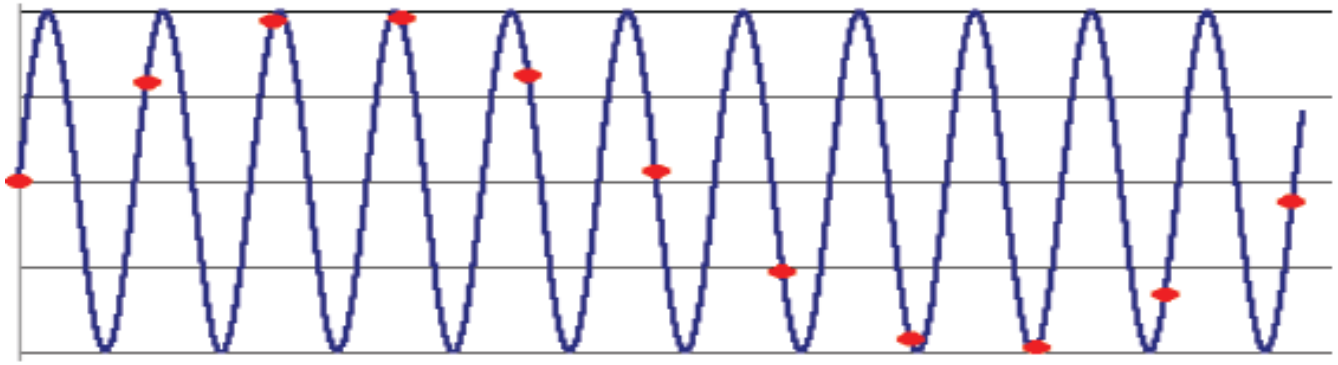


- To do sampling right, need to understand the structure of your signal/image
- The minimum sampling rate is called the Nyquist rate

[Source: R. Urtasun]

Aliasing

- Occurs when your sampling rate is not high enough to capture the amount of detail in your image



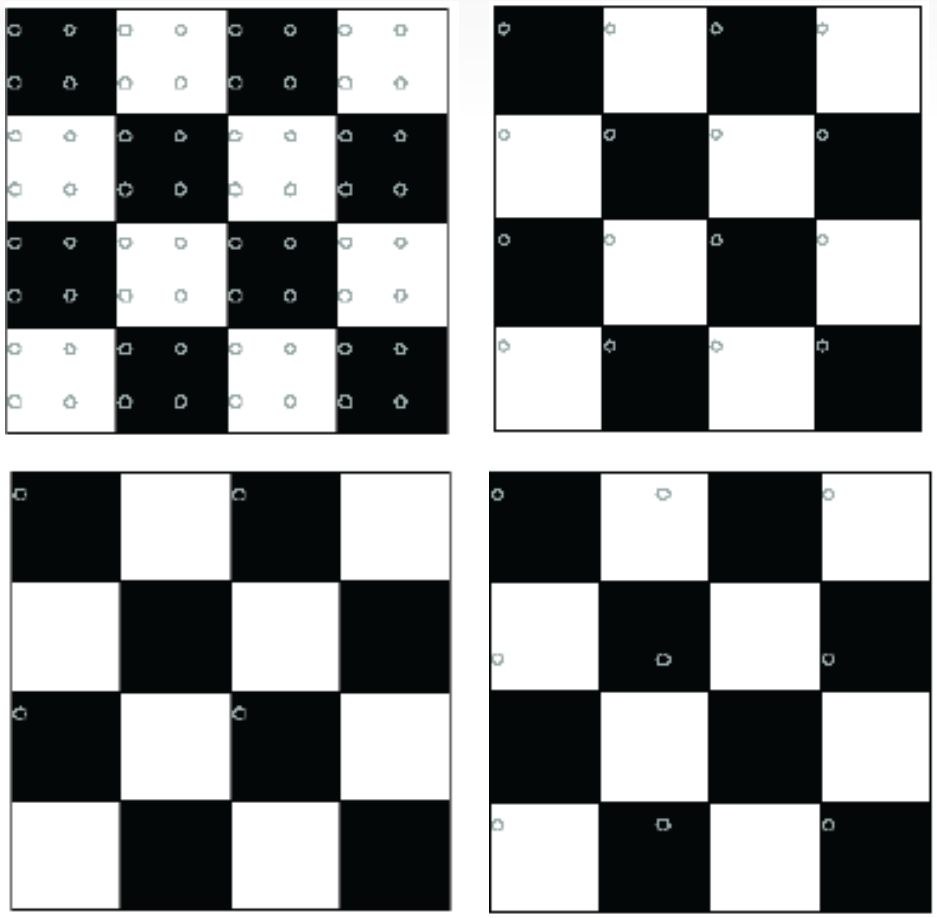
- To do sampling right, need to understand the structure of your signal/image
- The minimum sampling rate is called the Nyquist rate

[Source: R. Urtasun]

Mr. Nyquist

- Harry Nyquist says that one should look at the frequencies of the signal.
- One should find the highest frequency (via Fourier Transform)
- To sample properly you need to sample with at least twice that frequency
- For those interested: http://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

2D example



Good sampling

Bad sampling

[Source:N. Snavely]

Down-sampling

- When downsampling by a factor of two, the original image has frequencies that are too high
- High frequencies are caused by sharp edges
- How can we fix this?

[Adopted from: R. Urtasun]

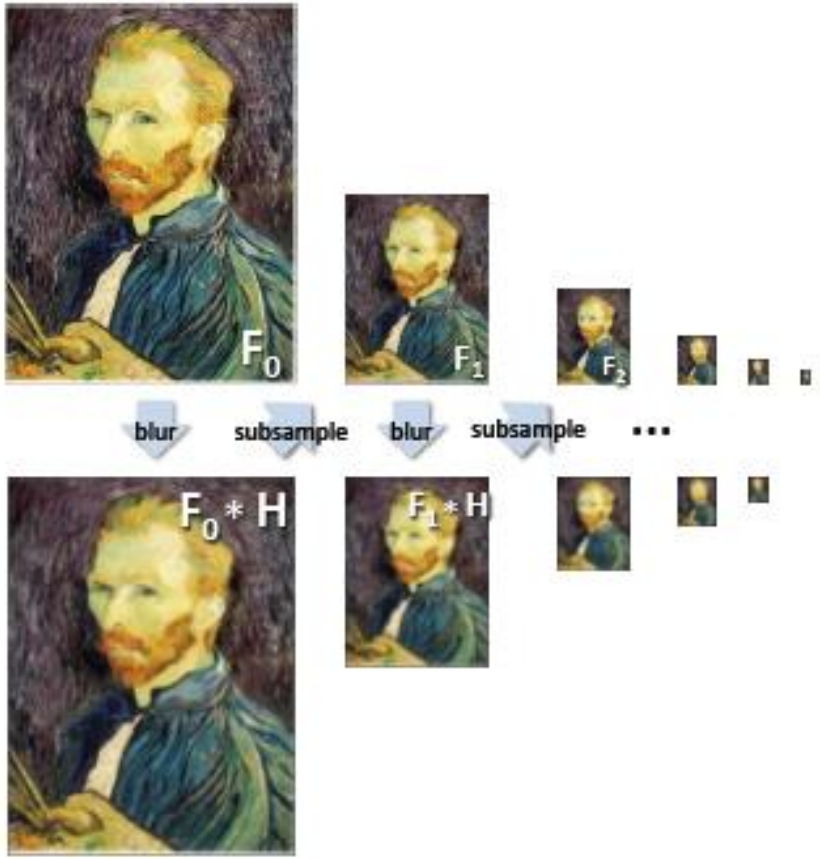
Down-sampling

- When downsampling by a factor of two, the original image has frequencies that are too high
- High frequencies are caused by sharp edges
- How can we fix this?

[Adopted from: R. Urtasun]

Gaussian pre-filtering

- Solution: Filter out the higher frequency data. Blur the image via Gaussian, then subsample. Very simple!



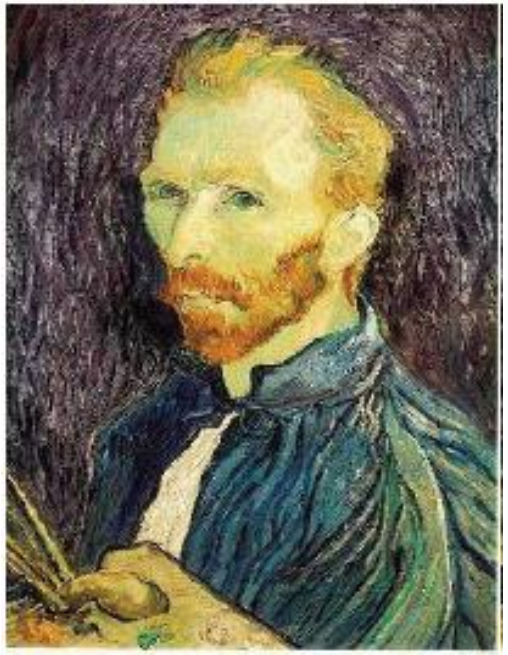
Subsampling with Gaussian pre-filtering

Gaussian 1/2

Gaussian 1/4

[Source: S. Seitz]

Compare to our result without



1/2 (2x zoom)

1/4 (4x zoom)

[Source: S. Seitz]

Where is the Chicken?

- Input image

[Source: S. Fidler]

Where is the Chicken?

- If I only take every other column (1st, 3rd, 5th, etc)
- Where is the chicken?

Where is the Chicken?

- Input image
- But Let's blur it,

[Source: S. Fidler]

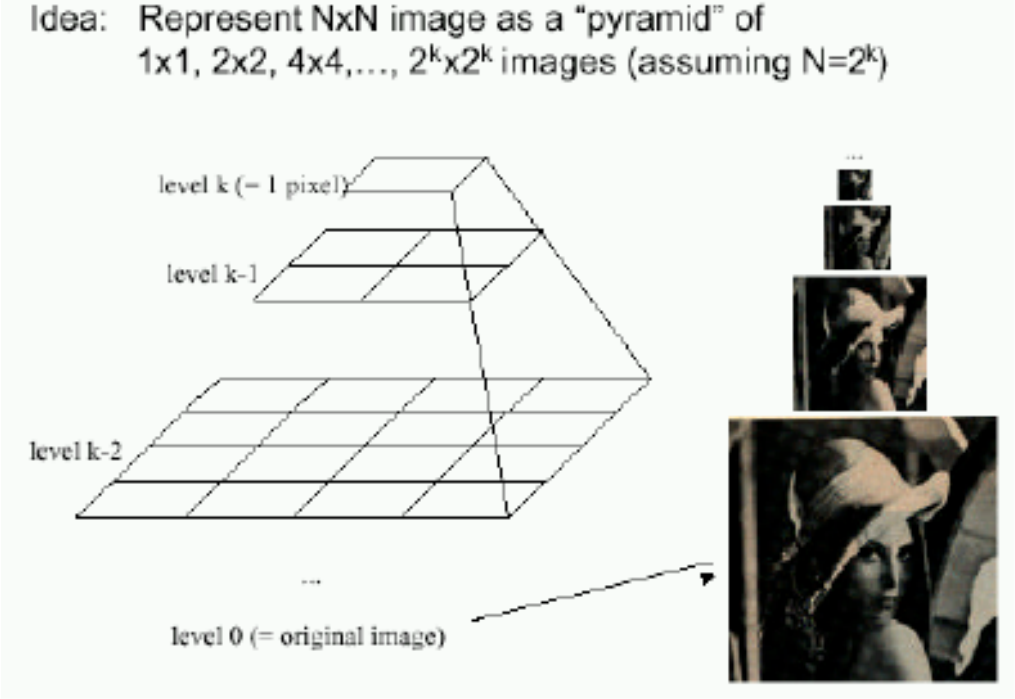
Where is the Chicken?

- Input image
- But Let's blur it,
- And now take every other column

[Source: S. Fidler]

Gaussian Pyramids [Burt and Adelson, 1983]

- A sequence of images created with Gaussian blurring and downsampling is called a Gaussian Pyramid
- For 3D rendering, a *mip map* [Williams, 1983] is a Gaussian Pyramid.



[Source: S. Seitz]

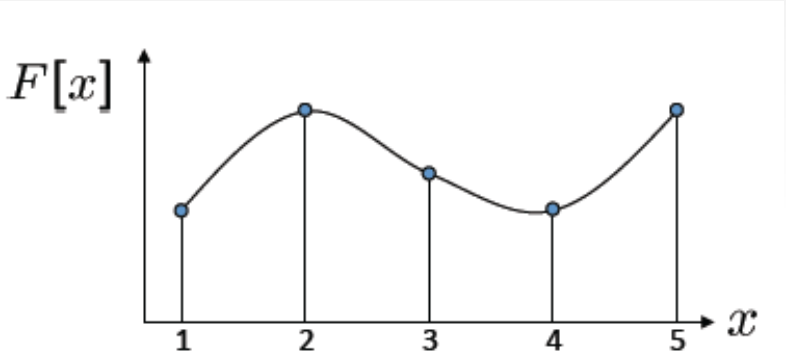
Image Up-Sampling

- This image is too small, how can we make it 10 times as big?

- Simplest approach: repeat each row and column 10 times

[Source:N. Snavely, R. Urtasun]

Interpolation



d = 1 in this example

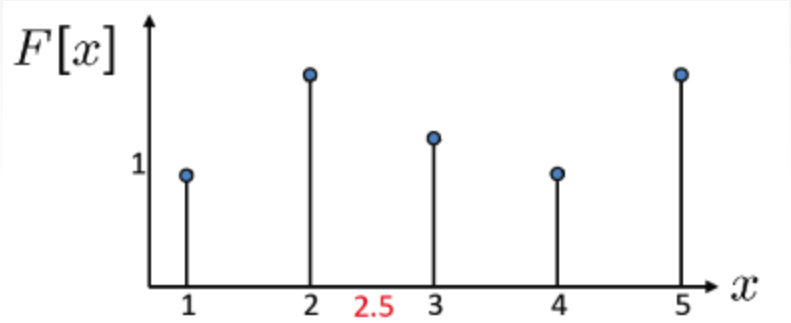
Recall: how a digital image is formed

$$F[x, y] = quantize\{f(\frac{x}{d}, \frac{y}{d})\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

[Source:N. Snavely, S. Seitz]

Interpolation

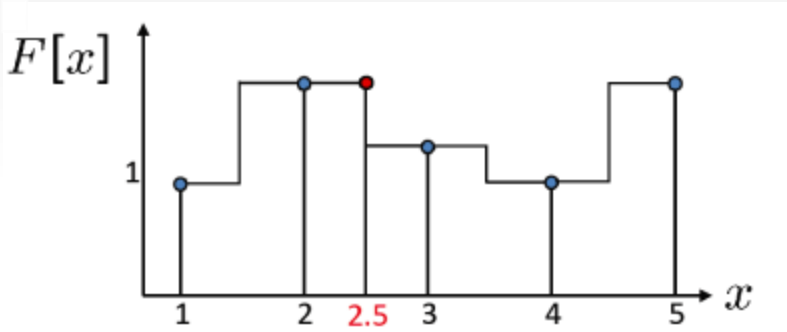


d = 1 in this example

- What if we don't know *f* ?

[Source:N. Snavely, S. Seitz]

Interpolation



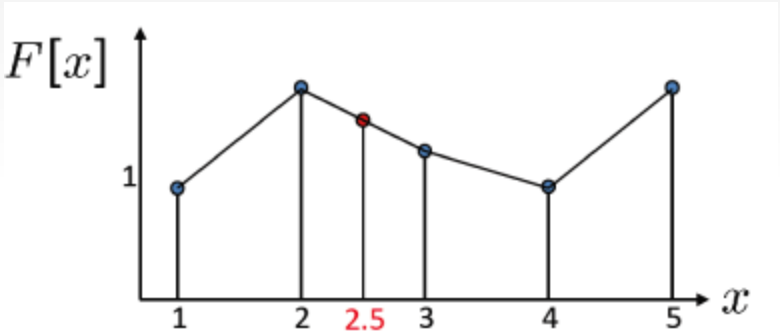
d = 1 in this example

What if we don't know *f* ?

- Guess an approximation: for example nearest-neighbor

[Source:N. Snavely, S. Seitz]

Interpolation



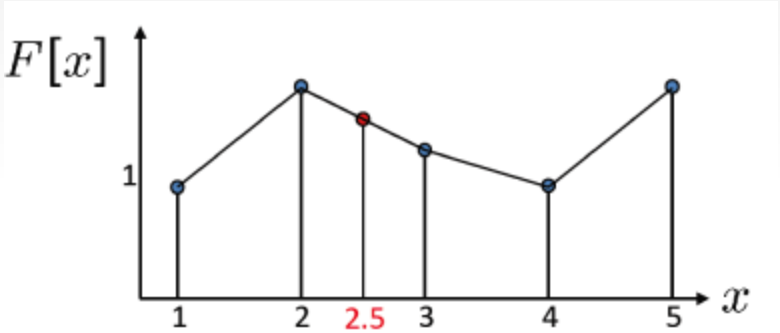
d = 1 in this example

What if we don't know *f* ?

- Guess an approximation: for example nearest-neighbor
- Guess an approximation: for example linear

[Source:N. Snavely, S. Seitz]

Interpolation



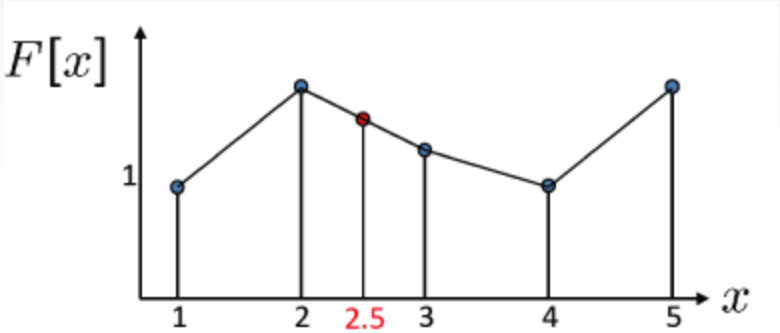
d = 1 in this example

What if we don't know *f* ?

- Guess an approximation: for example nearest-neighbor
- Guess an approximation: for example linear
- More complex approximations: cubic, B-splines

[Source:N. Snavely, S. Seitz]

Interpolation



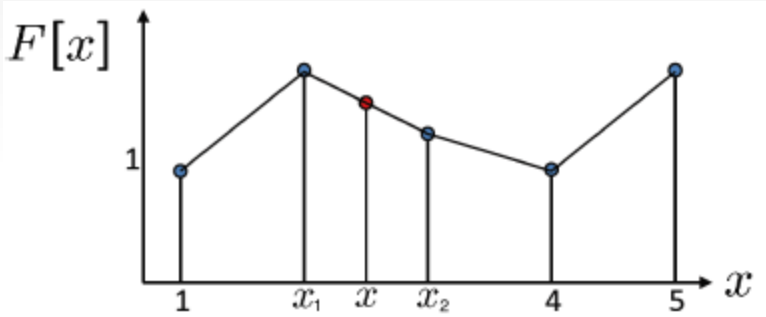
d = 1 in this example

What if we don't know *f* ?

- Guess an approximation: for example nearest-neighbor
- Guess an approximation: for example linear
- More complex approximations: cubic, B-splines
- But more isn't always better!

[Source:N. Snavely, S. Seitz]

Linear Interpolation



d = 1 in this example

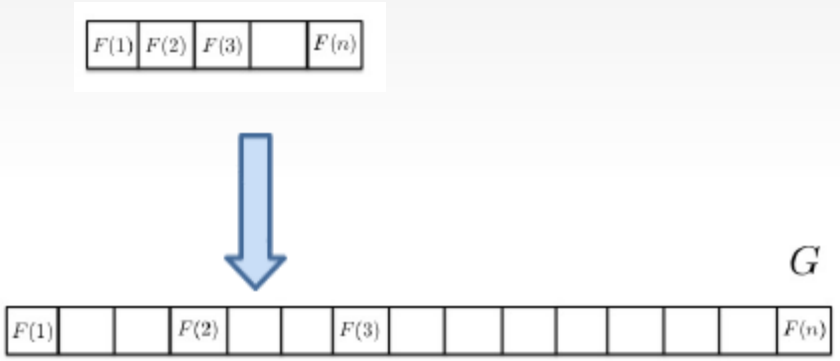
- Linear interpolation from our discretized F:

$$G(x) = \frac{x_2 - x}{x_2 - x_1} F(x_1) + \frac{x - x_1}{x_2 - x_1} F(x_2)$$

Interpolation: 1D Example

- Let’s make this signal triple length

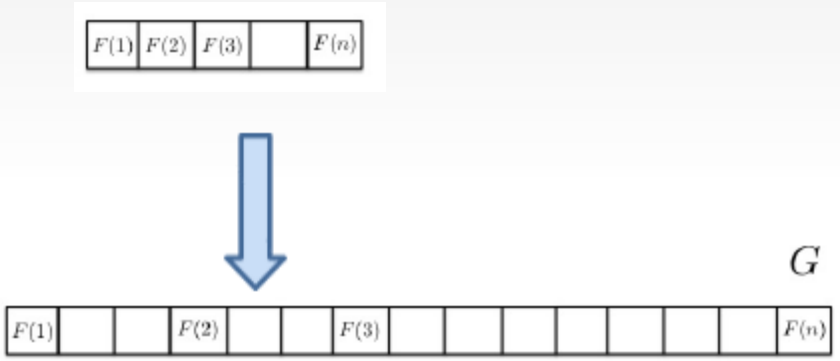
Interpolation: 1D Example



Make a vector *G* with *d* times the size of *F*

- Let's make this signal triple length (d=3)

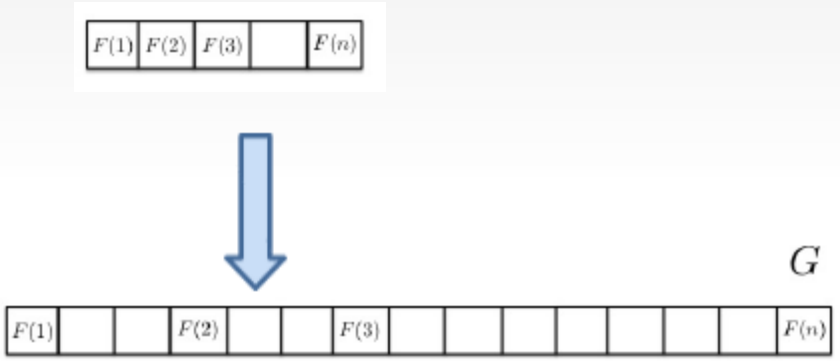
Interpolation: 1D Example



$$\text{if } \frac{i}{d} \text{ integer: } G(i) = F(i/d)$$

- Let's make this signal triple length (d=3)
- If *i/d* is an integer, just copy from the signal

Interpolation: 1D Example



$$\text{if } \frac{i}{d} \text{ integer: } G(i) = F(i/d)$$

$$\text{otherwise: } G(i) = \frac{x_2 - x}{x_2 - x_1} F(x_1) + \frac{x - x_1}{x_2 - x_1} F(x_2)$$

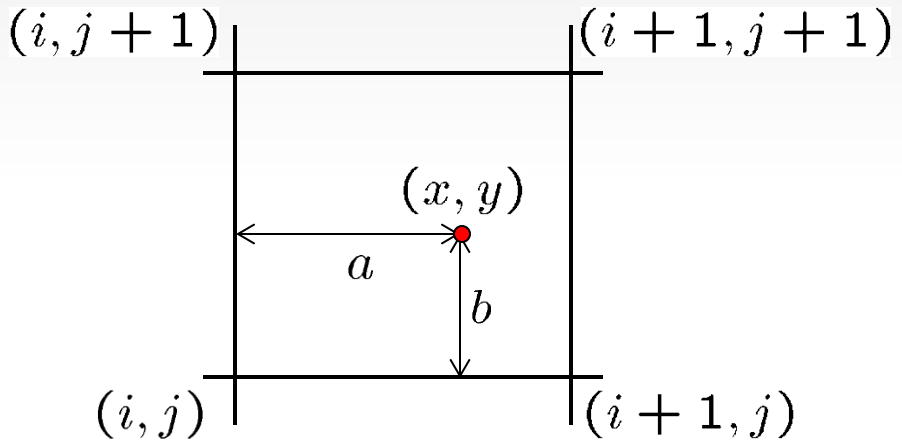
$$\text{where } \begin{aligned} x &= i/d \\ x_1 &= \lfloor i/d \rfloor \\ x_2 &= \lceil i/d \rceil \end{aligned}$$

- Let's make this signal triple length (d=3)
- If *i/d* is an integer, just copy from the signal
- Otherwise use the interpolation formula

2D Interpolation

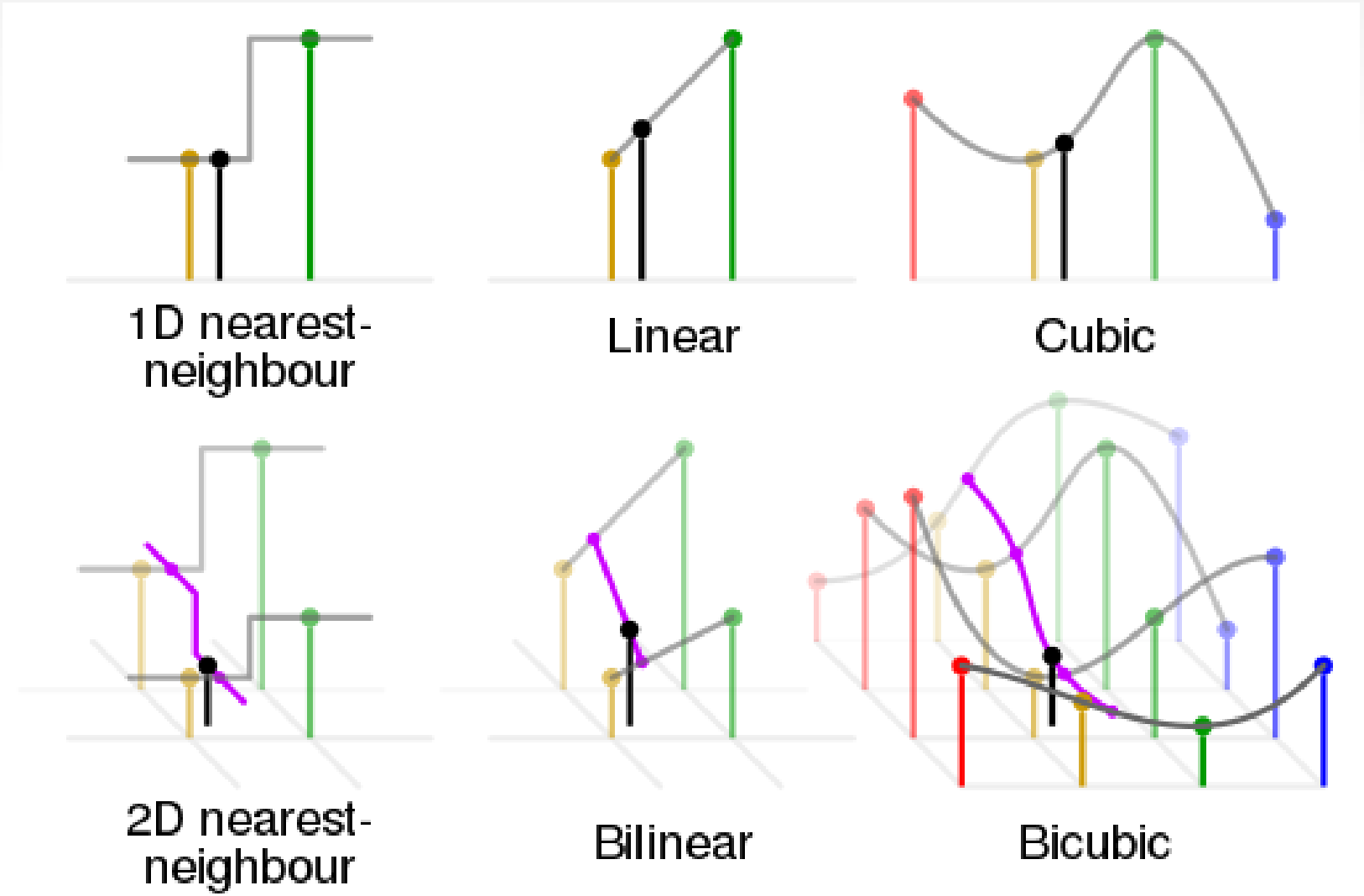
- Also nearest neighbor's value

- Better: bilinear interpolation; linear interpolation in x, then y, resulting in a quadratic interpolation.



$$
\begin{aligned}
f(x,y) = \quad & (1-a)(1-b) && f[i,j] \\
& +a(1-b) && f[i+1,j] \\
& +ab && f[i+1,j+1] \\
& +(1-a)b && f[i,j+1]
\end{aligned}
$$

Interpolation



Morphological Operations

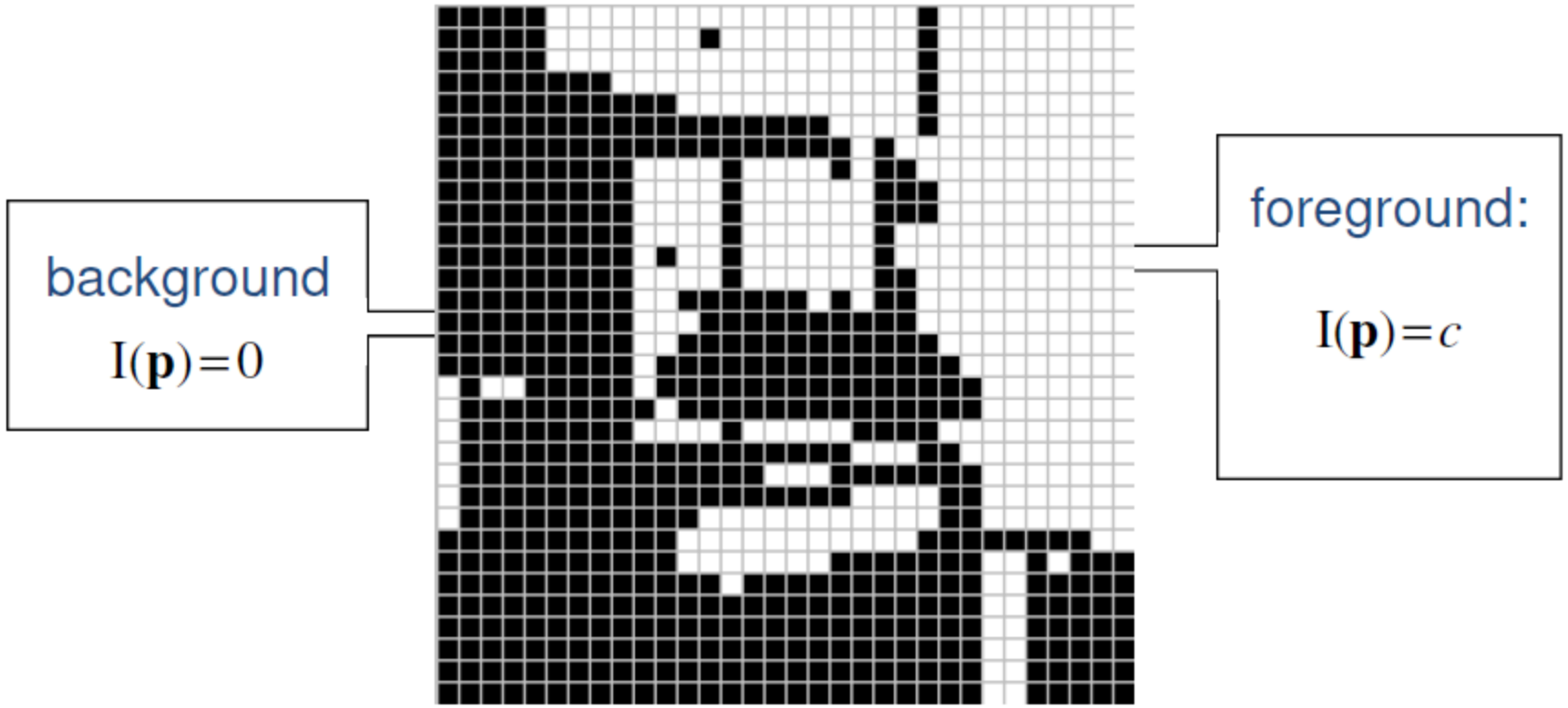
Finger Print Image Input

Morphology operation results

- Khawar Khurshid, 2012

Binary image

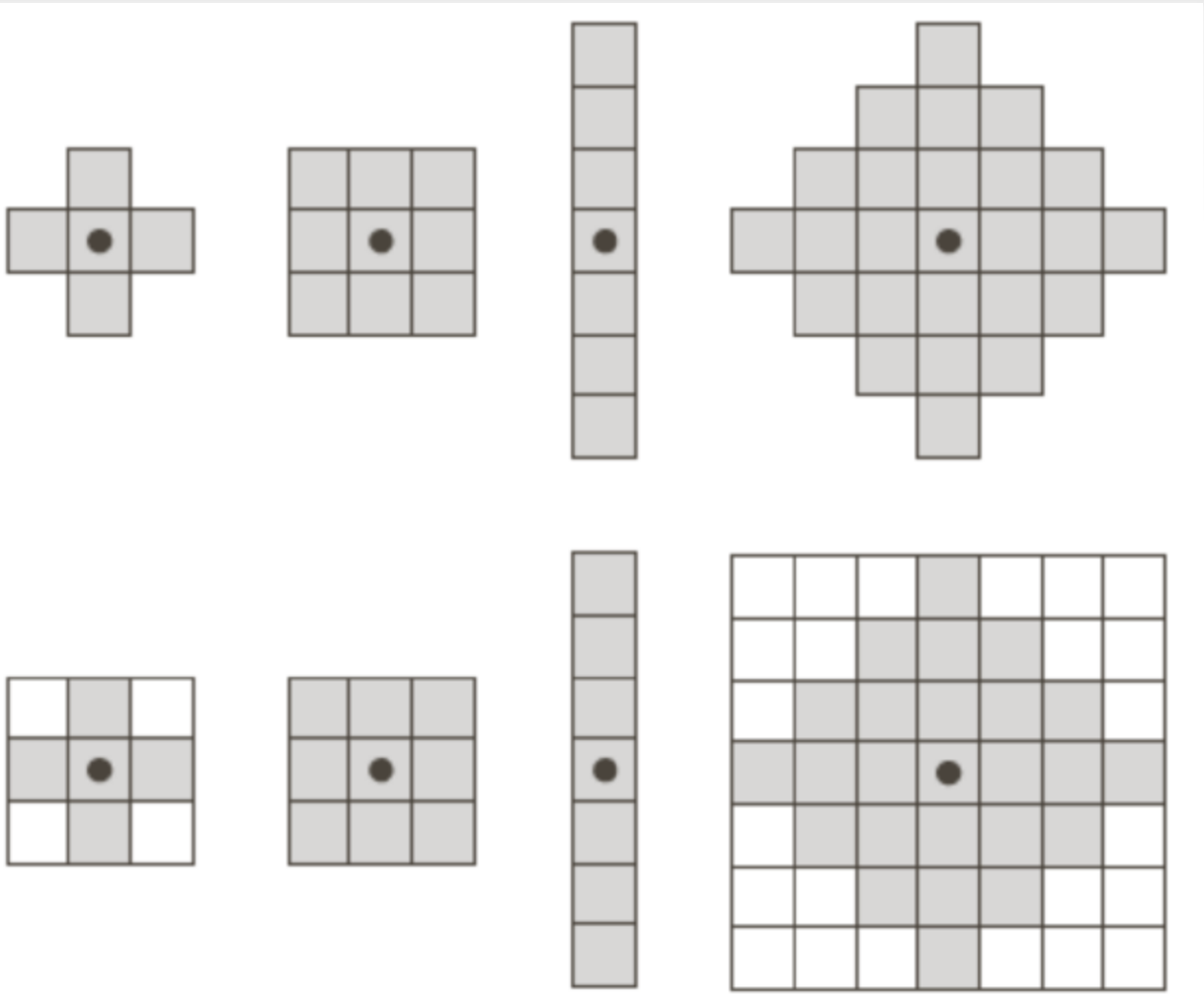
- Very useful, for it can be used to represent regions, or binary properties of pixels



Morphological Operations

"Probe" an image with a simple, pre-defined shape, drawing conclusions on how this shape fits or misses the shapes in the image.

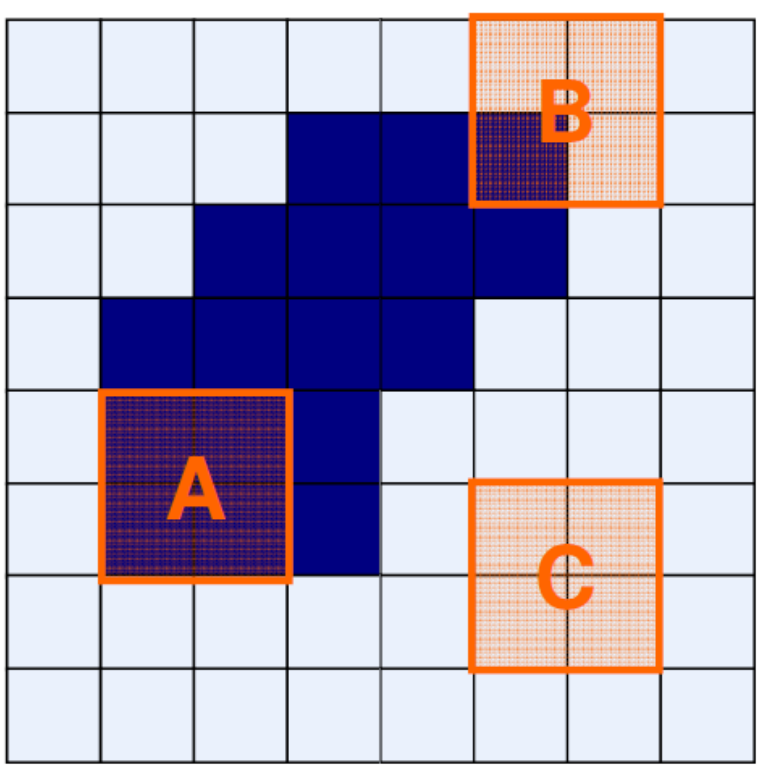
- This simple "probe" is called the structuring element, and is itself a binary image
- A structuring element is a small image – used as a moving window

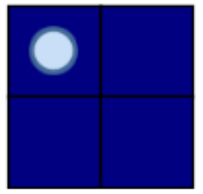


Fit and Hit

- Fit: All *of the pixels* in the structuring element cover *foreground pixels* in the image

- Hit: Any *one pixel* in the structuring element covers a *foreground pixel* in the image





Structuring Element

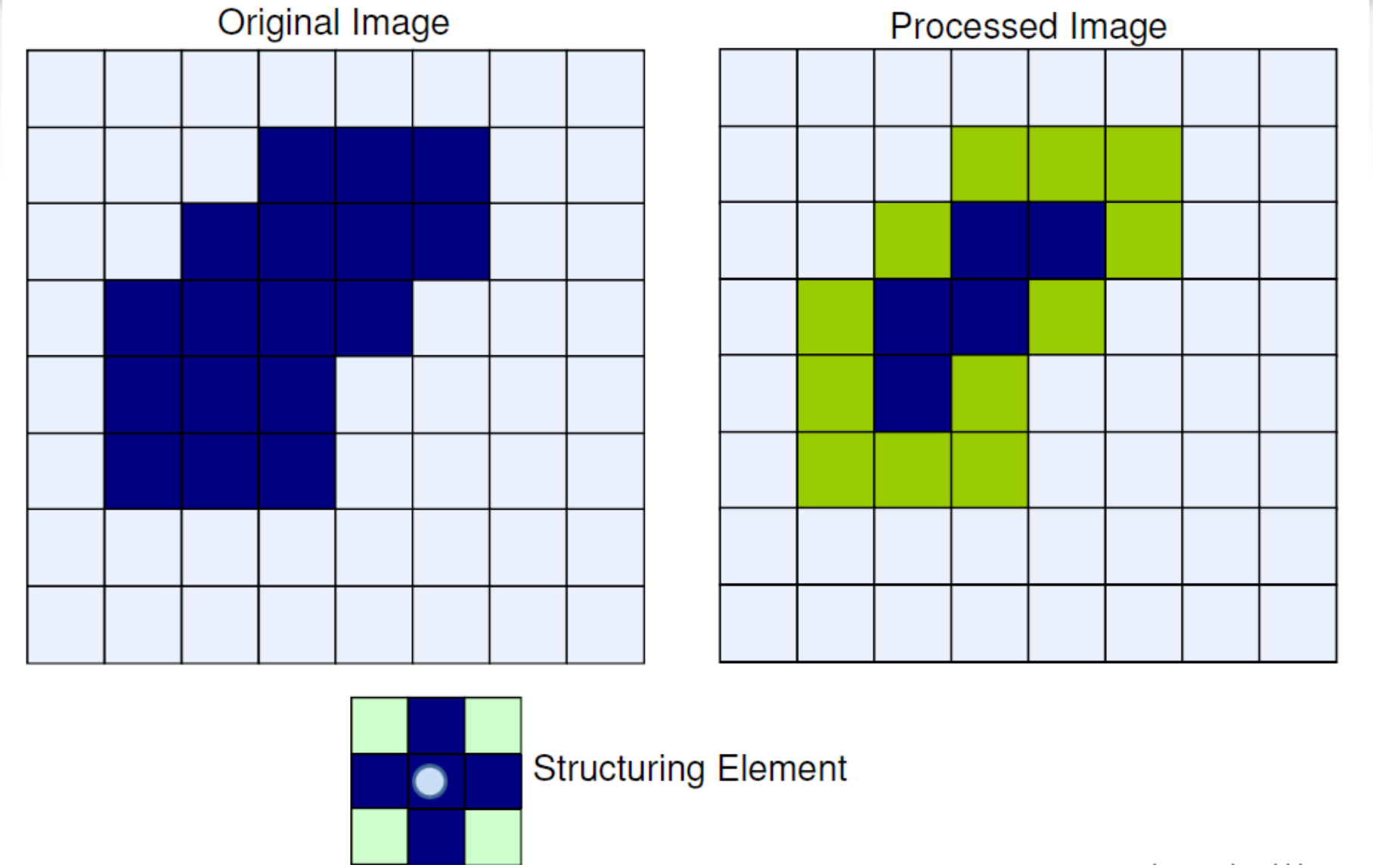
All morphological processing operations are based on these simple ideas

Erosion

- There are two basic morphological operations:
  - **erosion** and **dilation**
- Erosion of image *f* by structuring element *s* is given by:
  - The structuring element s is positioned with its origin at (x, y) and the new pixel value is determined using the rule:

$$g(x, y) = \begin{cases} 1 \text{ if } s \text{ fits } f \\ 0 \text{ otherwise} \end{cases}$$

Erosion

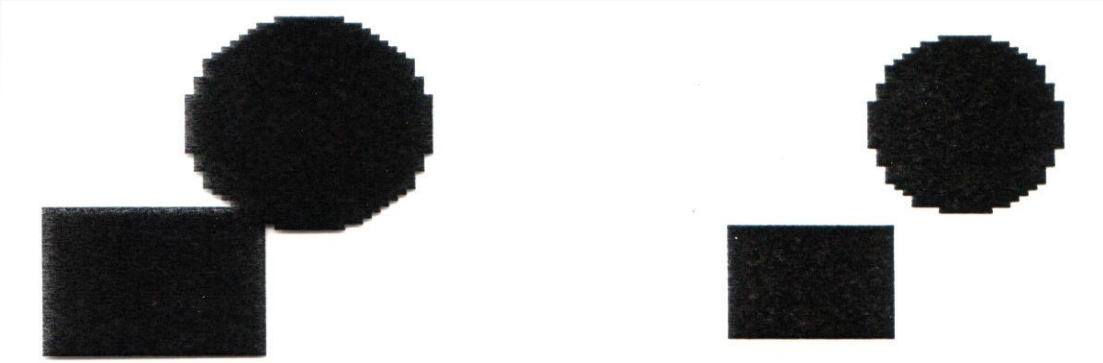


Erosion

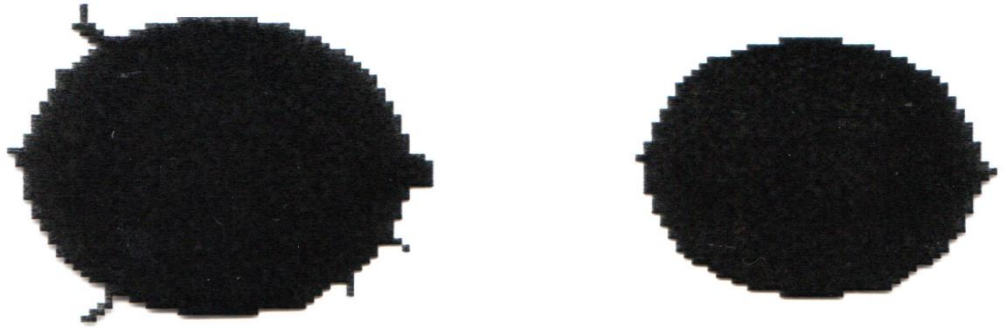
- Effects
  - Shrinks the size of foreground (1-valued) objects
  - Smoothes object boundaries
  - Removes small objects
- Rule for Erosion
  - In a binary image, if any of the pixel (in the neighborhood defined by structuring element) is 0, then output is 0

Erosion

- Erosion can split apart joined objects



- Erosion can strip away extrusions

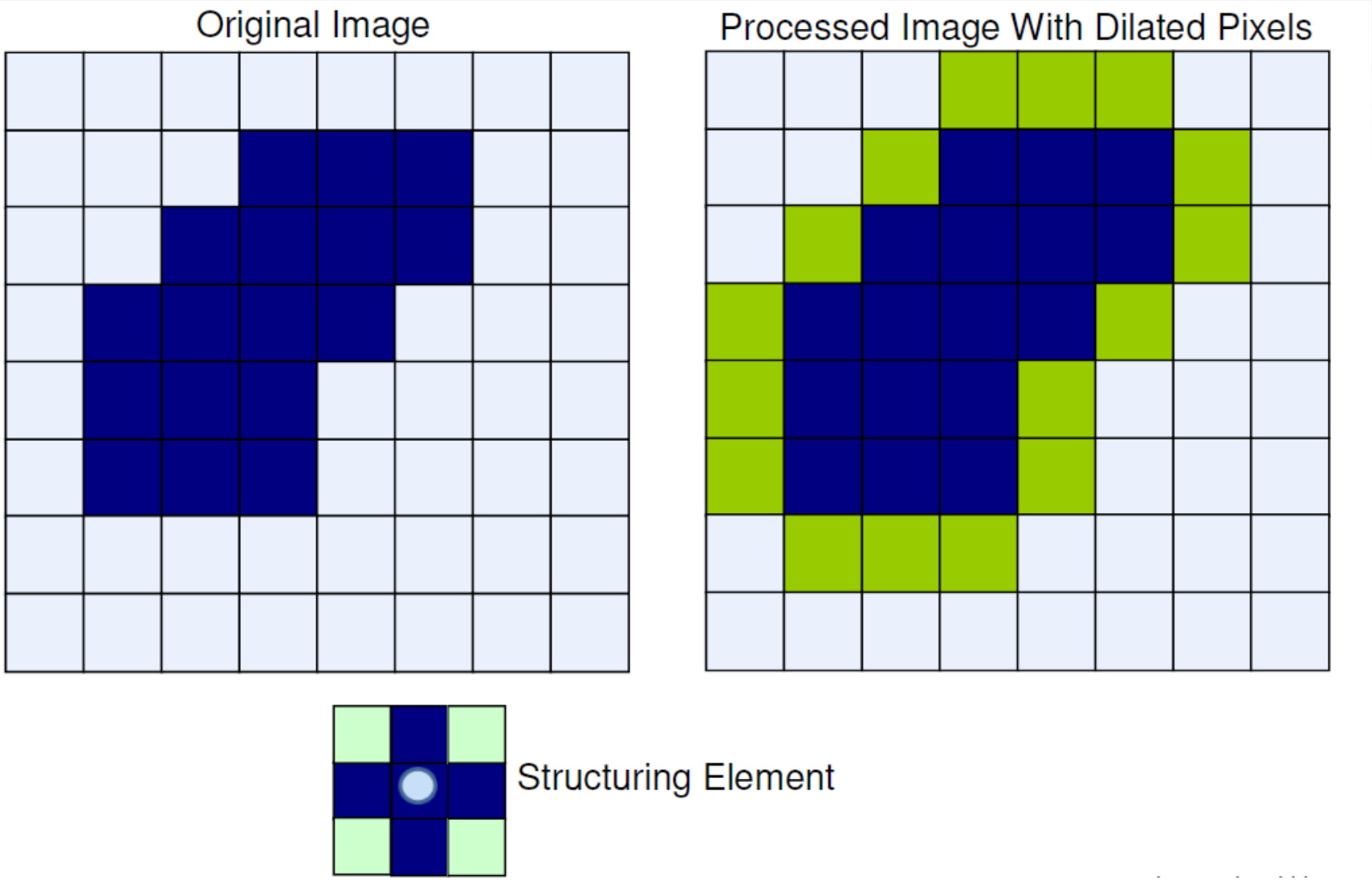


Dilation

- Dilation of image *f* by structuring element *s* is given by
  - The structuring element s is positioned with its origin at *(x, y)* and the new pixel value is determined using the rule:

$$g(x, y) = \begin{cases} 1 \text{ if } s \text{ hits } f \\ 0 \text{ otherwise} \end{cases}$$

Dilation


Original Image
Processed Image With Dilated Pixels
Structuring Element

Dilation

- Effects
  - Expands the size of foreground (1-valued) objects.
  - Smoothes object boundaries.
  - Closes holes and gaps.
- Rule for Dilation
  - In a binary image, if any of the pixel (in the neighborhood defined by structuring element) is 1, then output is 1.

Dilation

- Dilation can repair breaks

- Dilation can repair intrusions

Compound Operations

- The most widely used of these *compound operations* are:
  - **Opening**
  - **Closing**

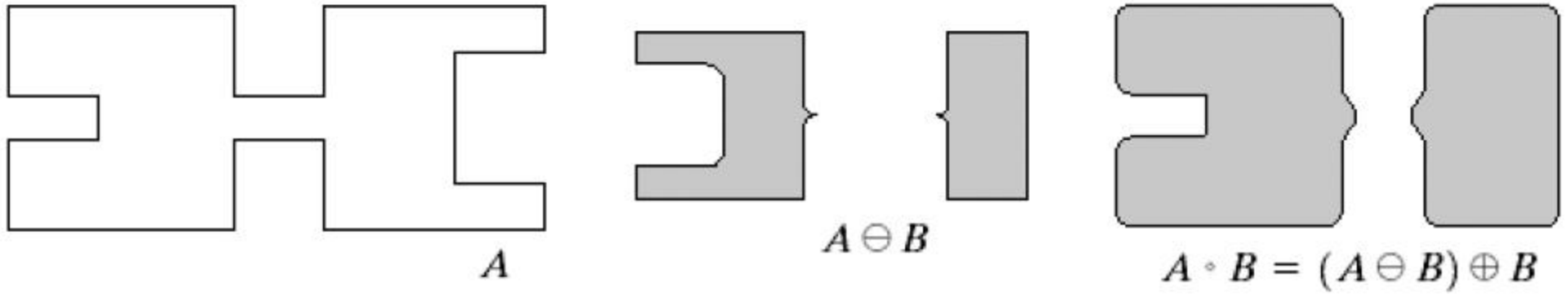
- **Try to remove noises, or fix intrusions/extrusions, BUT keep the original sizes of shapes**

- Opening and Closing also hold the duality property.

Open

- The opening of image f by structuring element s, which is denoted by $f \circ s$ is simply an erosion followed by a dilation

$$f \circ s = (f \ominus s) \oplus s$$

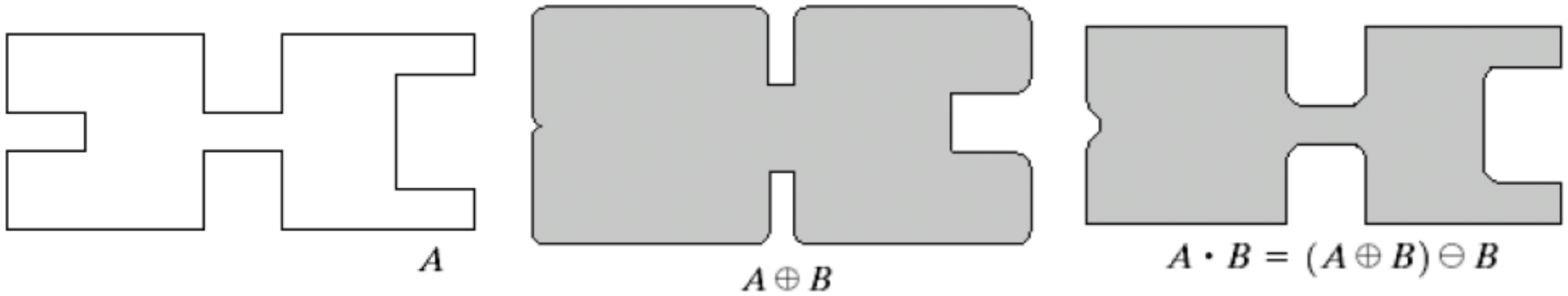


Breaks narrow joints
Removes ‘Salt’ noise

Close

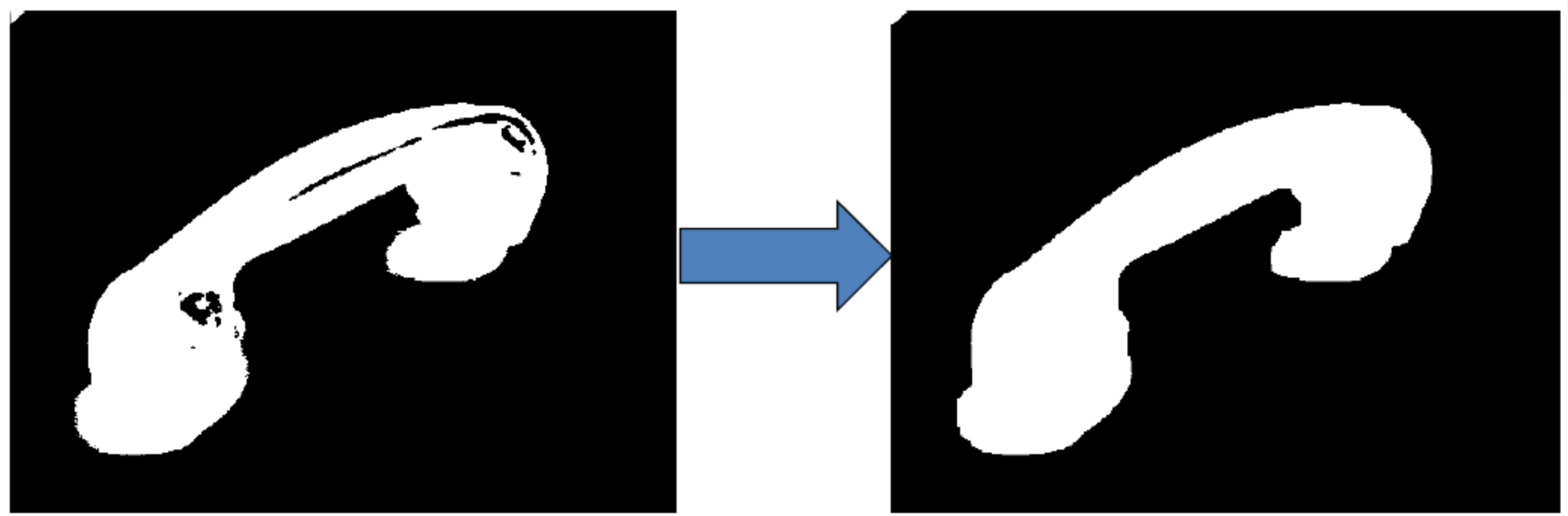
- The closing of image *f* by structuring element *s*, denoted by $f \bullet s$ is simply a dilation followed by an erosion

$$f \bullet s = (f \oplus s) \ominus s$$

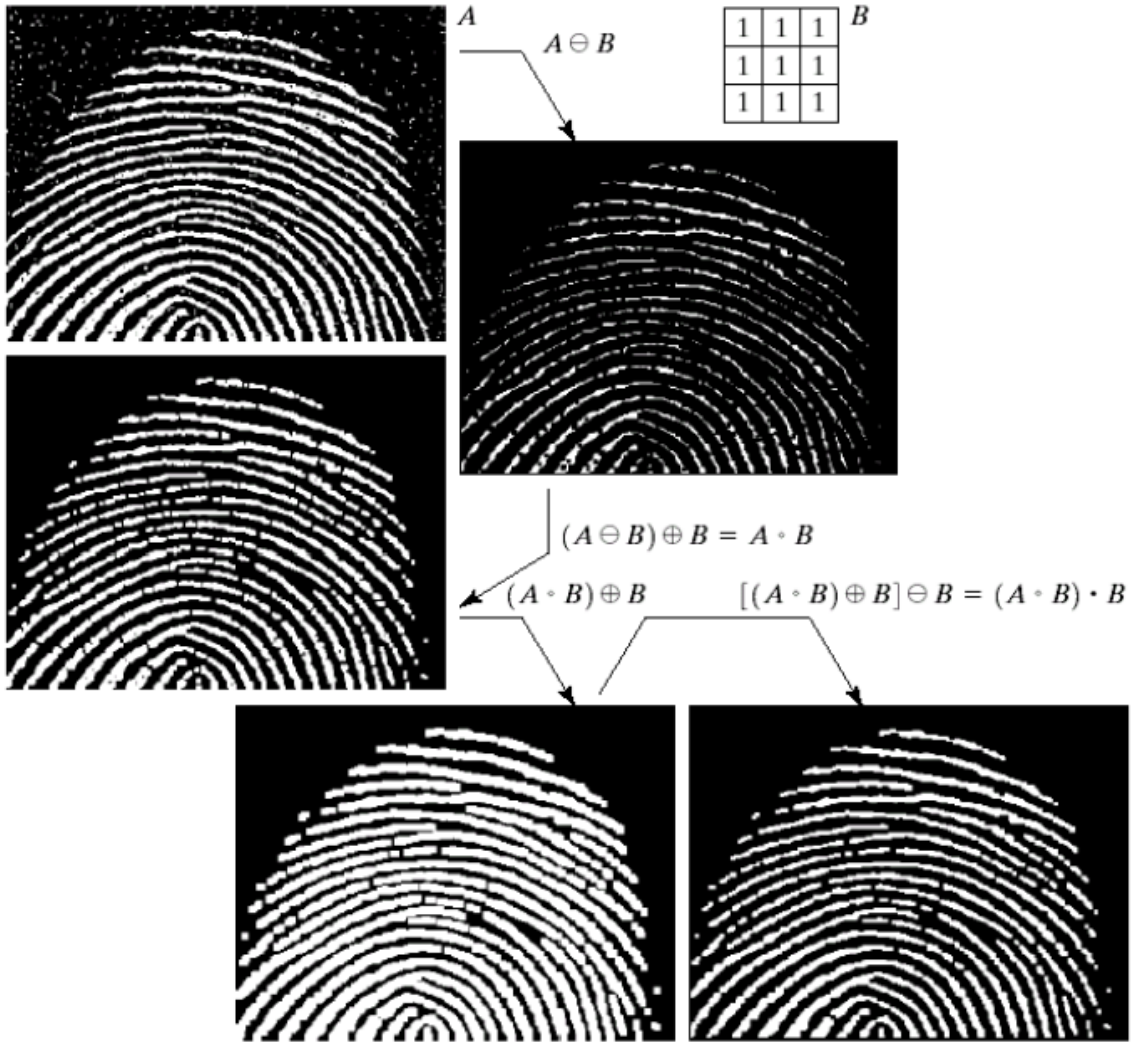


Eliminates small holes
- Fills gaps
- Removes ‘Pepper’ noise

Closing



A large example



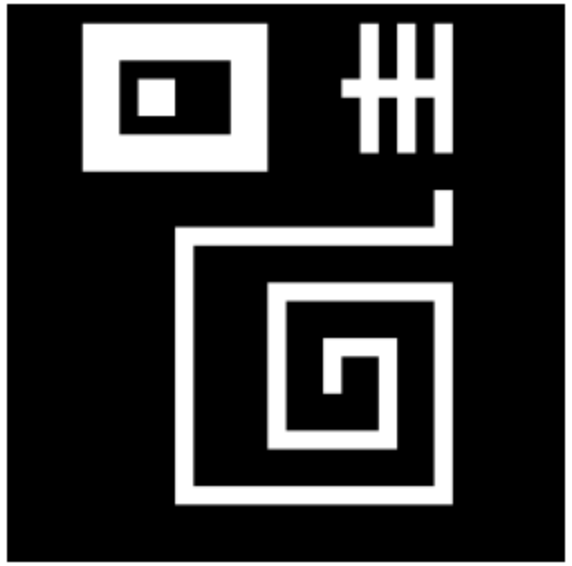
a b
d c
e f

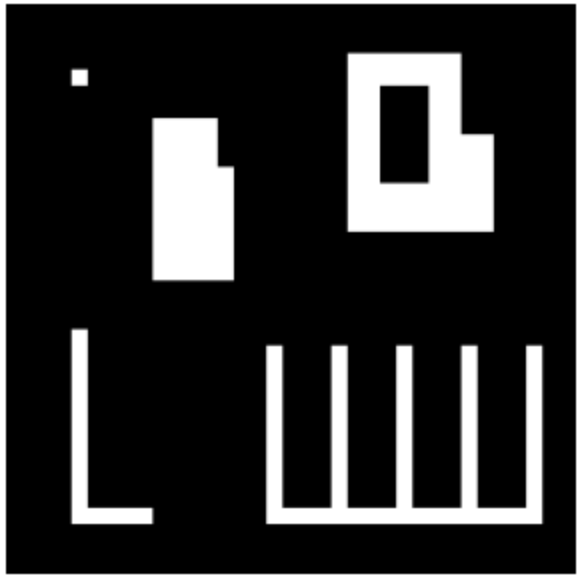
FIGURE 9.11
(a) Noisy image. (b) Structuring element. (c) Eroded image. (d) Opening of A. (e) Dilation of the opening. (f) Closing of the opening. (Original image courtesy of the National Institute of Standards and Technology.)

[Figure 9.11 of Gonzalez book: Digital Image Processing]

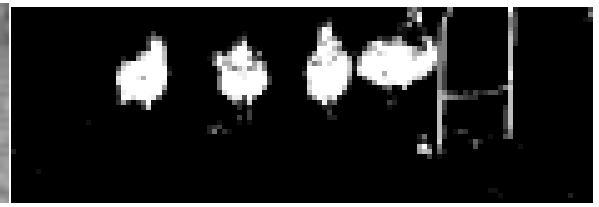
Connected components in binary images

- How many objects in the following images?





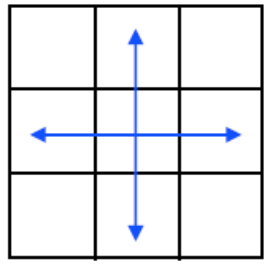
- Why do we care?
  - How many chickens in this image?



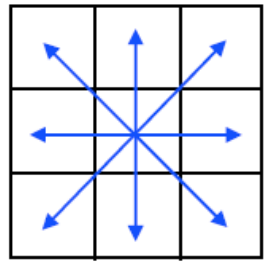
Connected components in binary images

- Define what is “connect”:

Neighborhoods:



4-neighbor metric



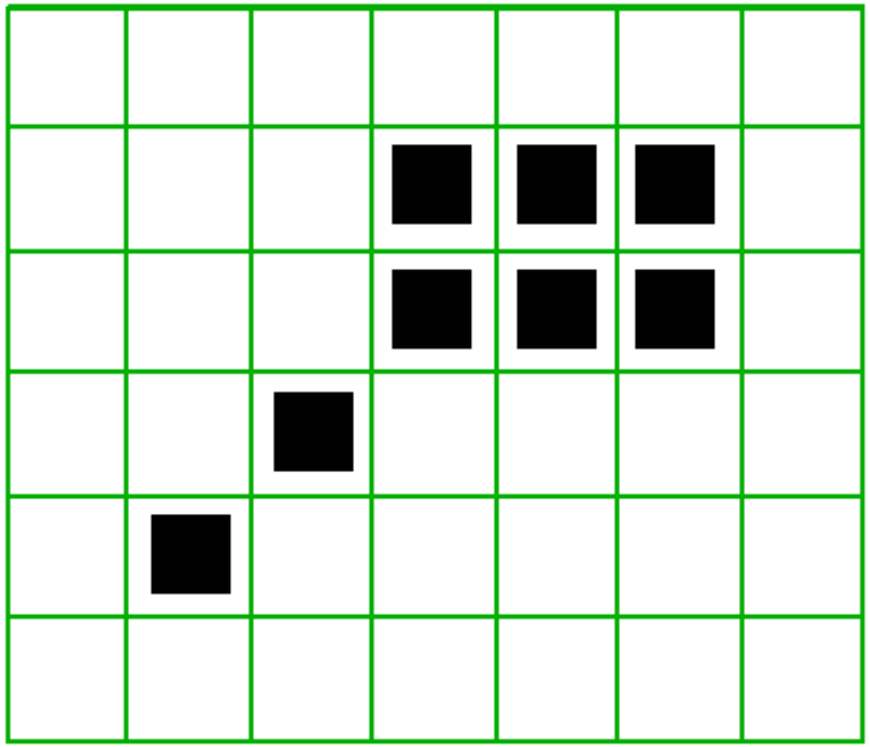
8-neighbor metric

Connected Components:

- S= the set of object pixels, where S is a Connected Component if:
  - **for each pixel pair (x1,y1) in S and (x2,y2) in S there is a path passing through X-neighbors in S. (here X = 4 or 8).**
- S may contain several connected components.

Connected Components

- Example



1 connected component-8

3 connected components-4

Algorithm for getting connected components

Connected Component Algorithm: Two passes over the image.

Pass 1:

Scan the image pixels from left to right and from top to bottom.

For every pixel P of value 1 (an object pixel), test top and left neighbors (4-neighbor metric).

- If 2 of the neighbors equals 0: assign a new mark to P.
- If 1 of the neighbors equals 1: assign the neighbor's mark to P.
- If 2 of the neighbors equals 1: assign the left neighbor's mark to P and note equivalence between 2 neighbor’s marks.

Algorithm for getting connected components

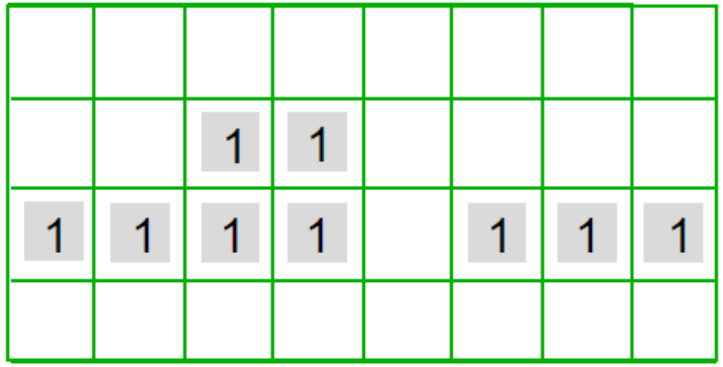
Connected Component Algorithm: Two passes over the image.

Pass 2:

- Divide all marks in to equivalence classes (marks of neighboring pixels are considered equivalent).
- Replace each mark with the number of it's equivalence class.

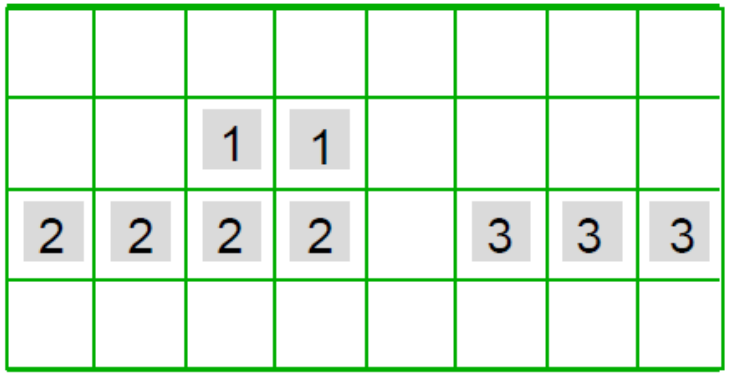
Algorithm for getting connected components

Original Binary image

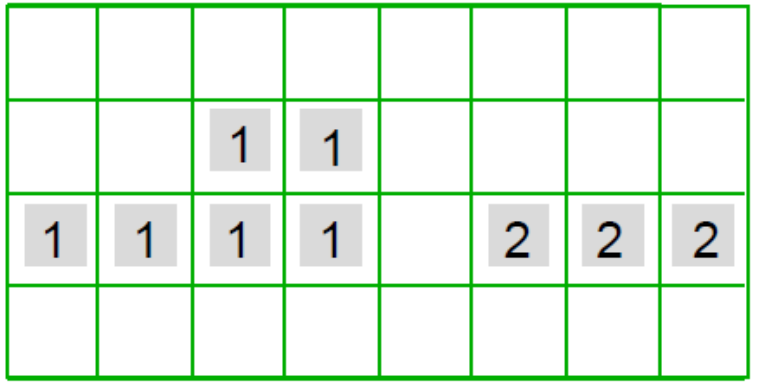


Equivalence Class number	Original mark
1	1,2
2	3

Pass 1:



Pass 2:



Euler number

- S= object pixels
- O= all other pixels
- Background= connected components of O that touch the edge of the image.
- Hole= connected components of O that is not in the background.
- Simply Connected Component = a component without holes.

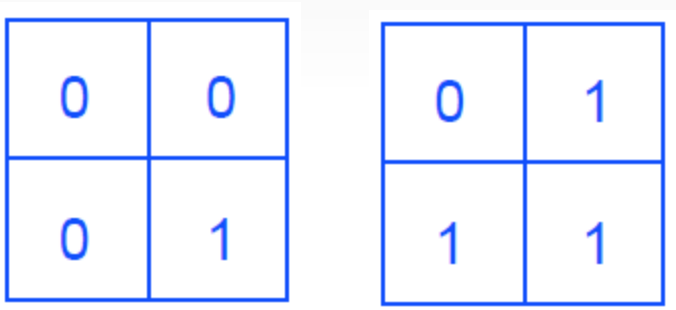
Euler Number = the number of objects minus the number of holes.

7 objects
3 holes

Euler = 4

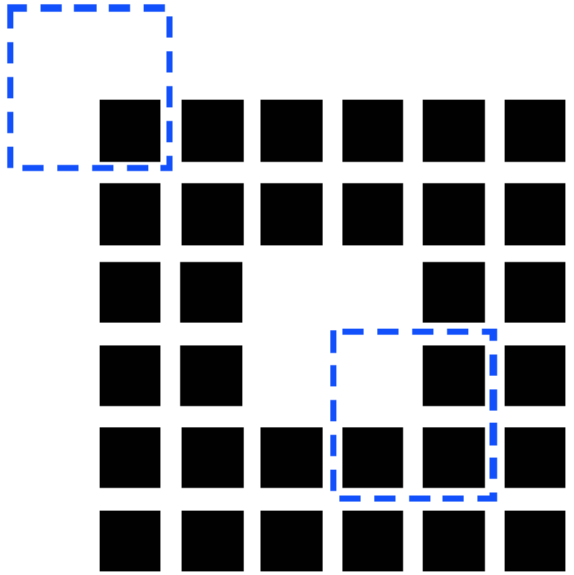
Euler Number

- It can be calculated by simply check the numbers of the following two patterns:



- If we have X pattern A and Y pattern B

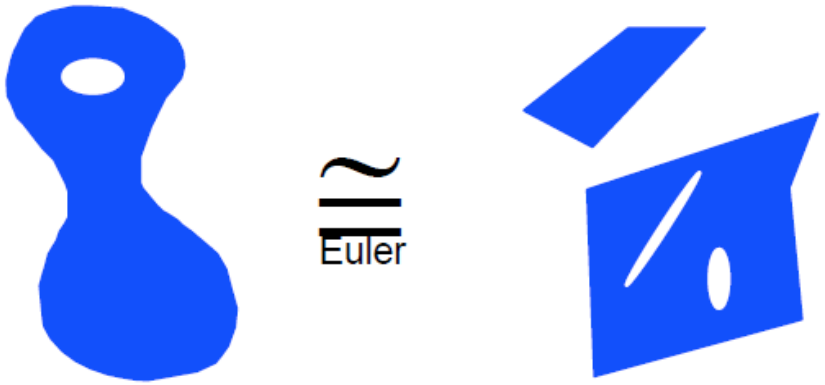
  Then, **Euler = X - Y**



Euler Number is a Shape Descriptor

- But not a very good one:

Euler Number = 0



Euler Number = 0
Objects = 1
Holes = 1