CS9840
Machine Learning in Computer Vision
Olga Veksler

Lecture 3

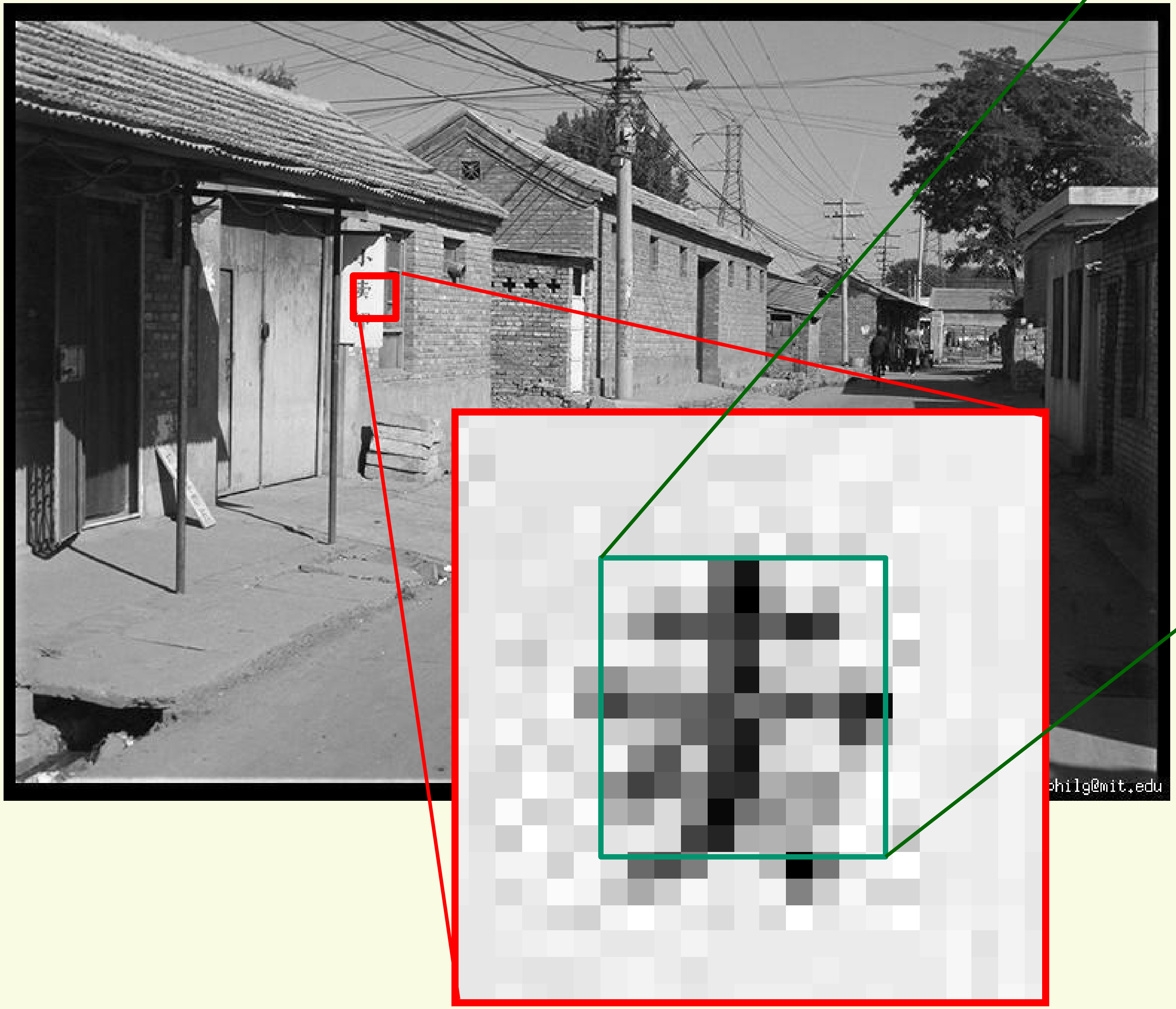
A Few Computer Vision Concepts

Some Slides are from Cornelia, Fermüller, Mubarak Shah, Gary Bradski, Sebastian Thrun, Derek Hoiem

Outline

- Computer Vision Concepts
  - Filtering
  - Edge Detection
  - Image Features
  - Template matching based on
    - Correlation
    - SSD
    - Normalized Cross Correlation
  - Motion and Optical Flow Field

Digital Grayscale Image

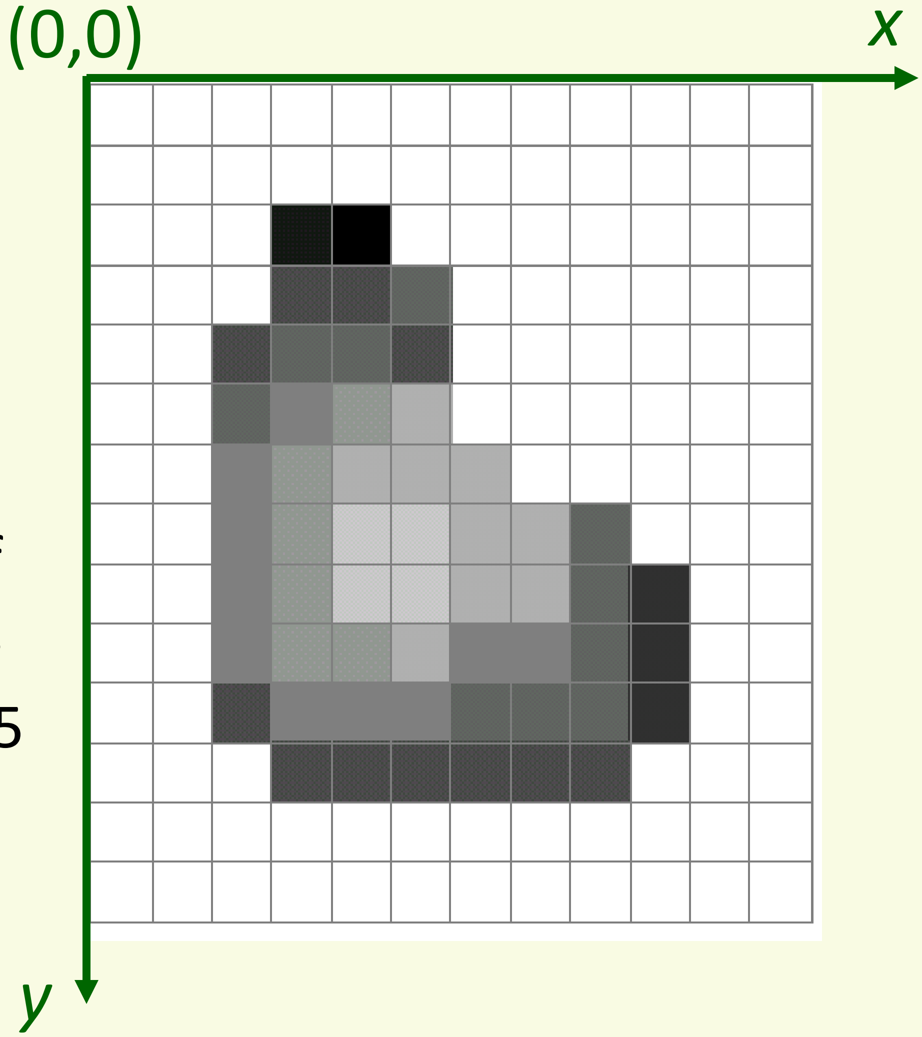


10	9	54	7	54	72
13	52	26	42	6	57
8	2	50	23	54	9
22	76	57	86	24	86
9	54	57	26	65	59
35	68	98	65	45	78
5	0	34	7	86	7

Slide Credit: D. Hoeim

Digital Grayscale Image

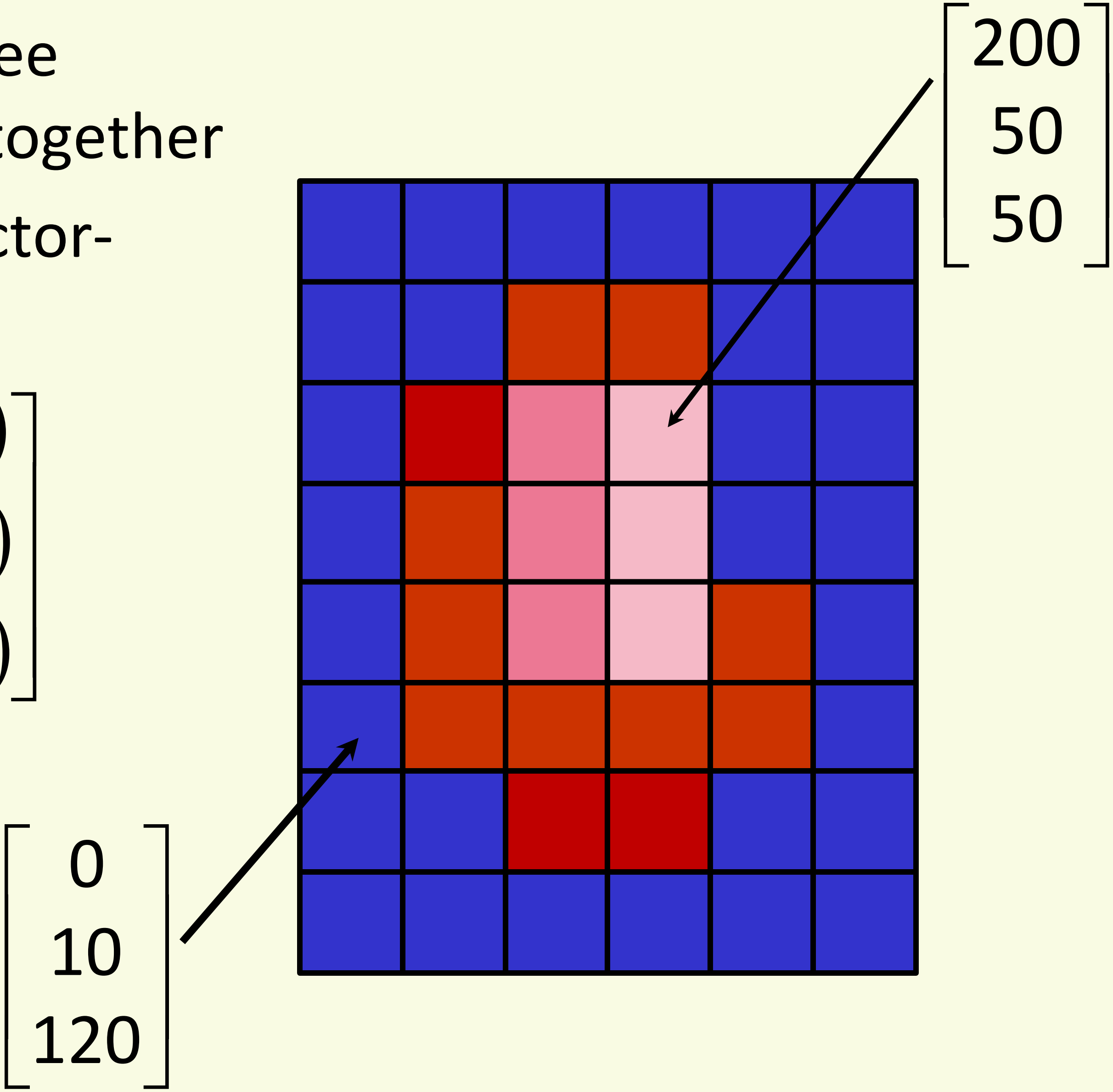
- Image is array *f*(*x*,*y*)
  - approximates continuous function *f*(*x*,*y*) from R^2 to R:
- *f*(*x*,*y*) is the **intensity** or **grayscale** at position (*x*,*y*)
  - proportional to brightness of the real world point it images
  - standard range: 0, 1, 2,…., 255



Digital Color Image

- Color image is three functions pasted together
- Write this as a vector-valued function:

$$f(x,y)=\begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$



$$\begin{bmatrix} 200 \\ 50 \\ 50 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 10 \\ 120 \end{bmatrix}$$

Digital Color Image

- Can consider color image as 3 separate images: R, G, B

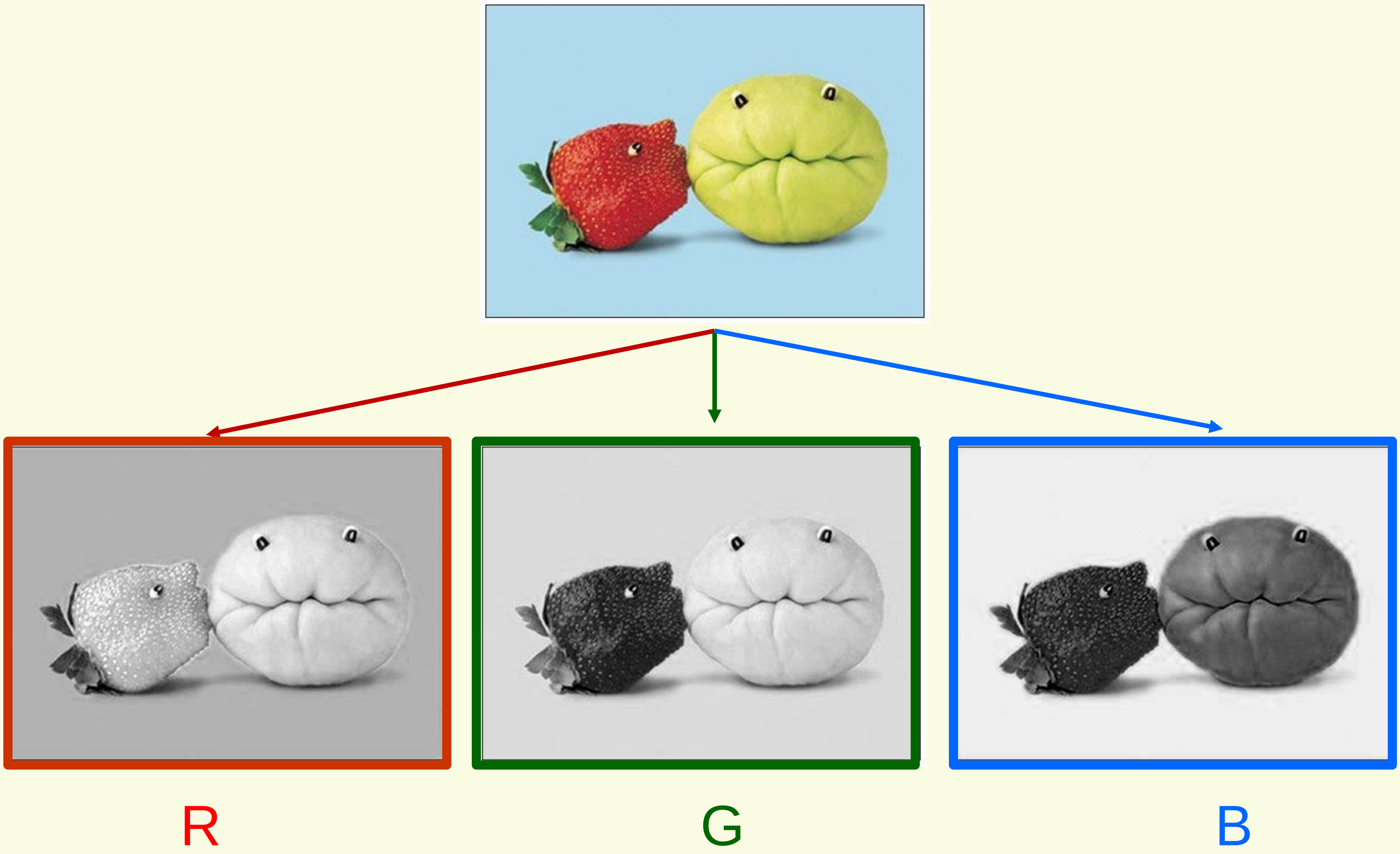


Image filtering

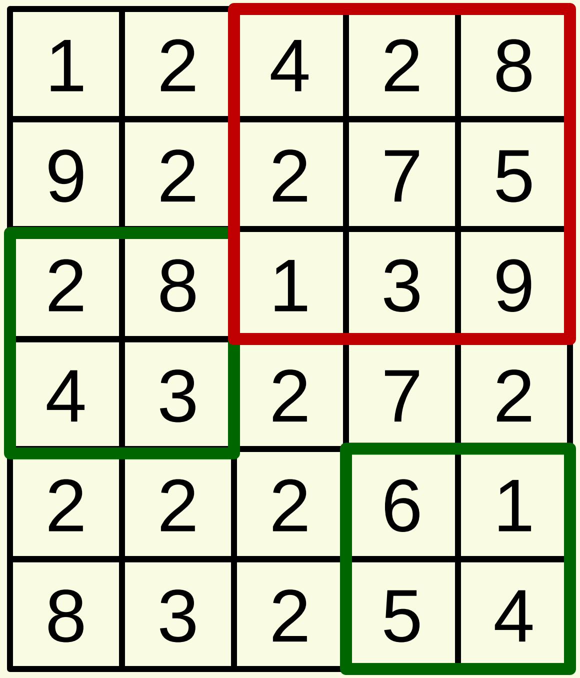
- Given $f(x,y)$ filtering computes a new image $g(x,y)$
- As a function of local neighborhood at each position (x,y)

$$g(x,y) = f(x,y)+f(x-1,y)\times f(x,y-1)$$

- Linear filtering: function is a weighted sum (or difference) of pixel values

$$g(x,y) = f(x,y) + 2\times f(x-1,y-1) - 3\times f(x+1,y+1)$$

- Many applications:
  - Enhance images
    - denoise, resize, increase contrast, …
  - Extract information from images
    - Texture, edges, distinctive points …
  - Detect patterns
    - Template matching



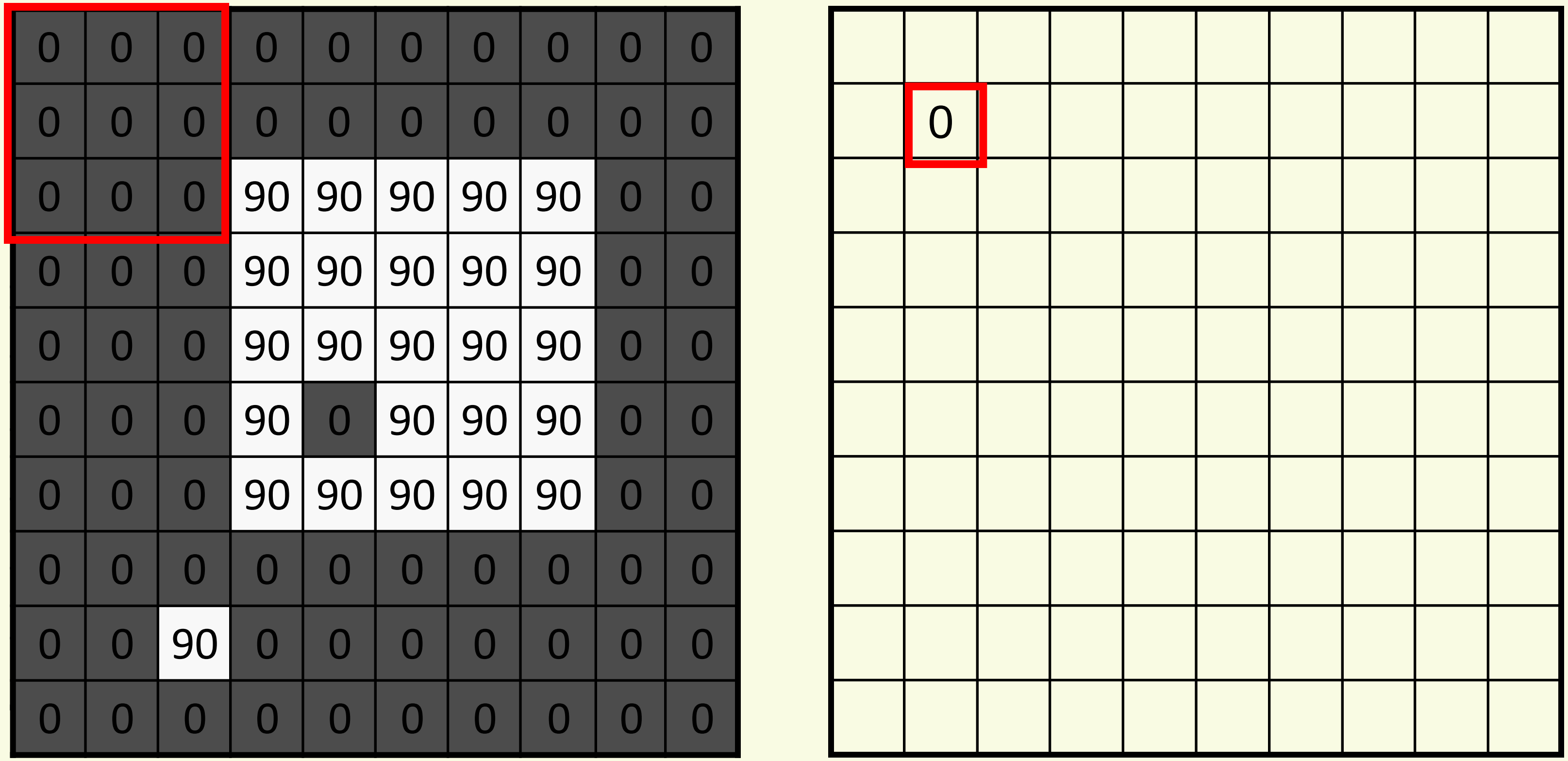
1	2	4	2	8
9	2	2	7	5
2	8	1	3	9
4	3	2	7	2
2	2	2	6	1
8	3	2	5	4

$g(2,3) = 3 + 4 \times 8 = 35$

$g(4,5) = 4 + 5 \times 1 = 9$

$g(3,1) = 7 + 2\times 4 - 3\times 9 = -12$

Image Filtering: Moving Average



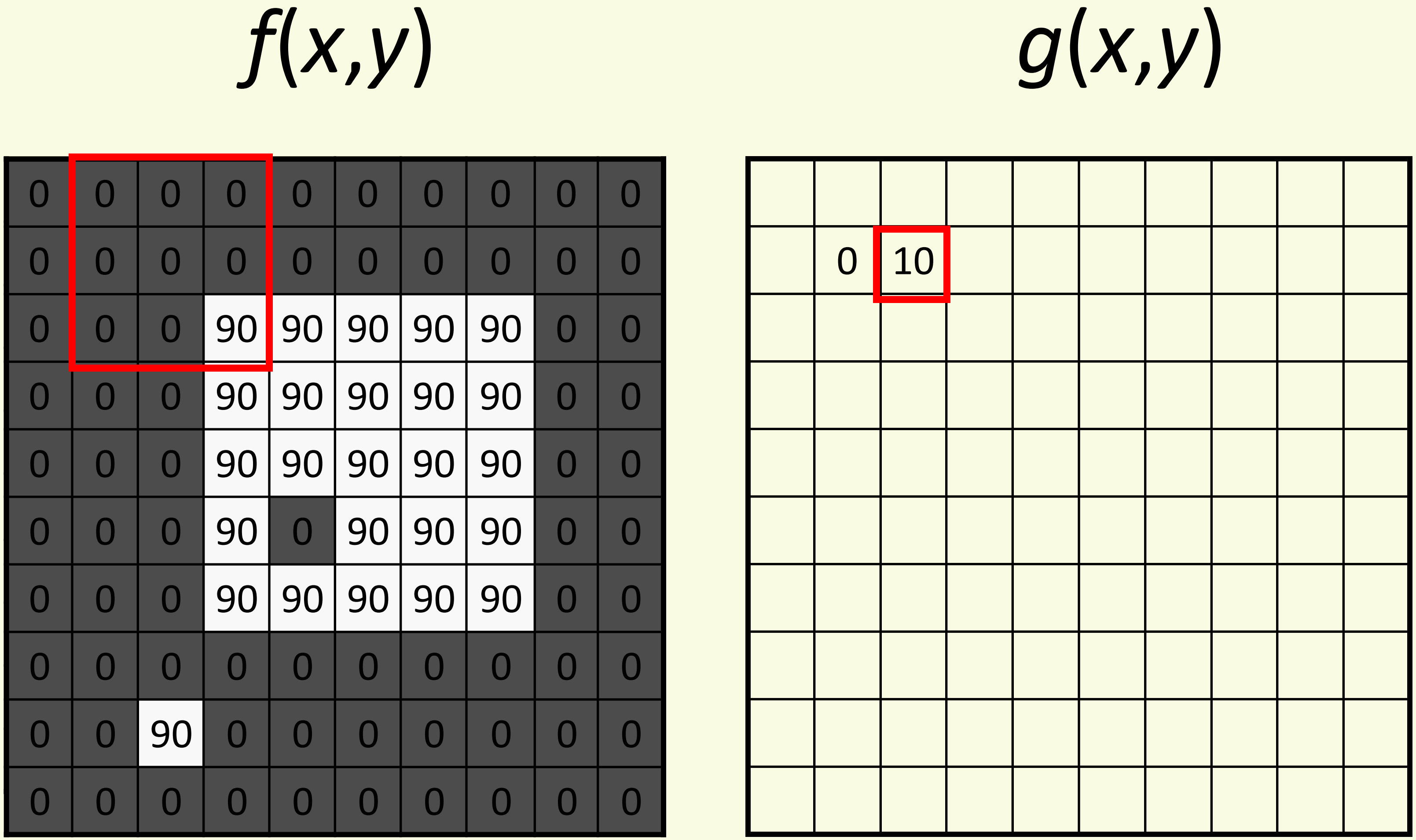
$f(x,y)$

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

$g(x,y)$

	0								

Image Filtering: Moving Average



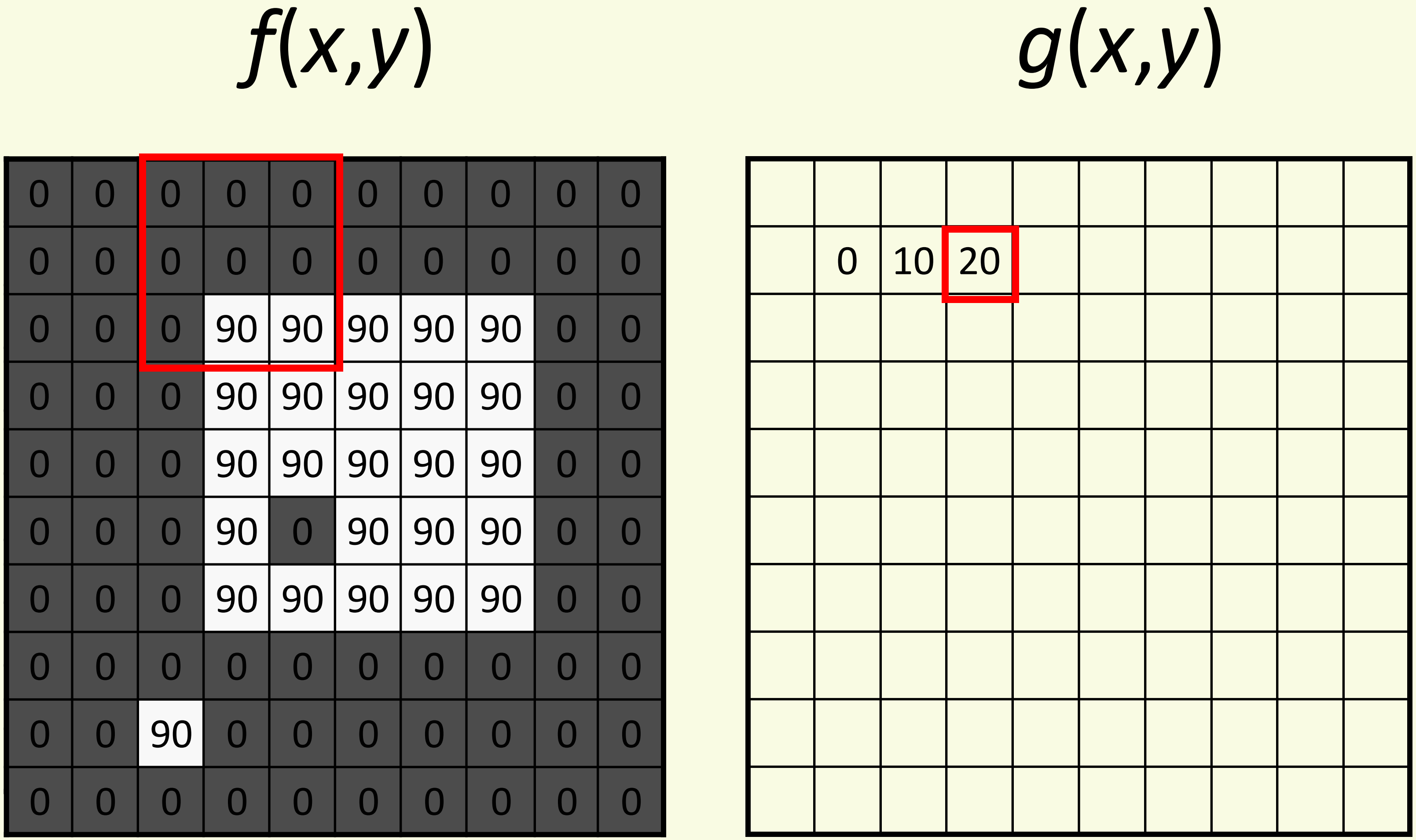
$f(x,y)$

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

$g(x,y)$

	0	10							

Image Filtering: Moving Average



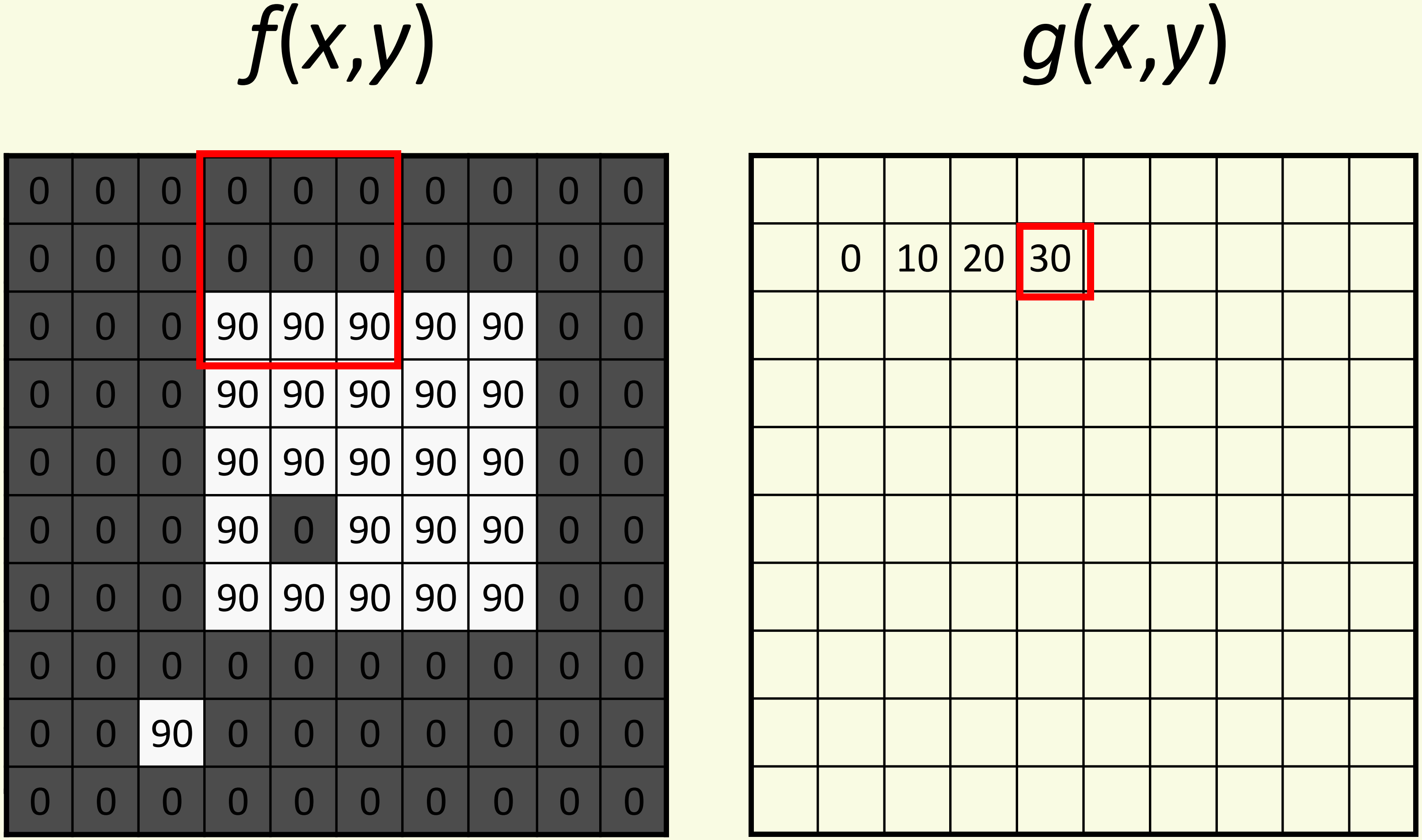
f(*x*,*y*)

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

g(*x*,*y*)

	0	10	20						

Image Filtering: Moving Average



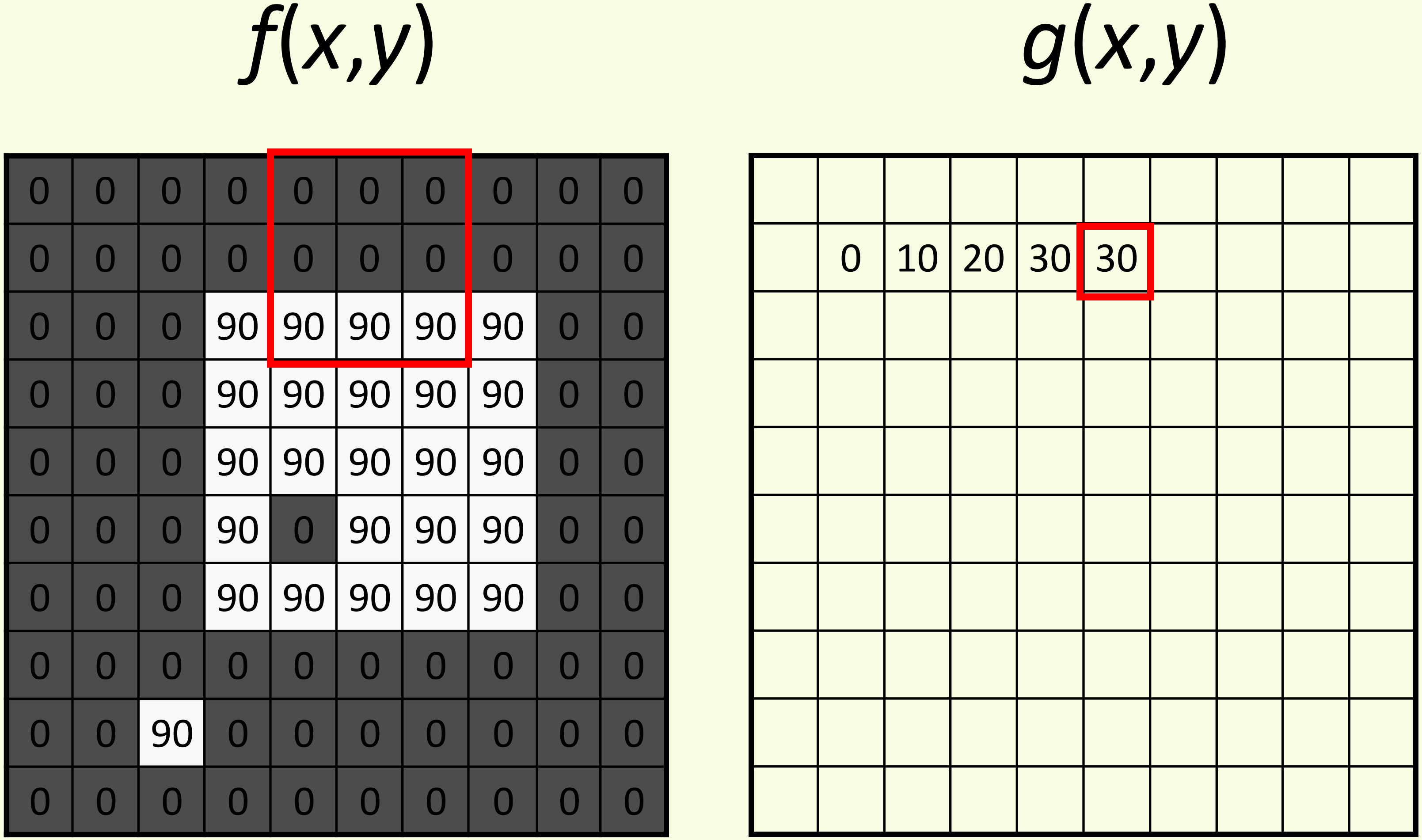
f(*x*,*y*)

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

g(*x*,*y*)

	0	10	20	30					

Image Filtering: Moving Average



$f(x,y)$

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

$g(x,y)$

	0	10	20	30	30				

Image Filtering: Moving Average

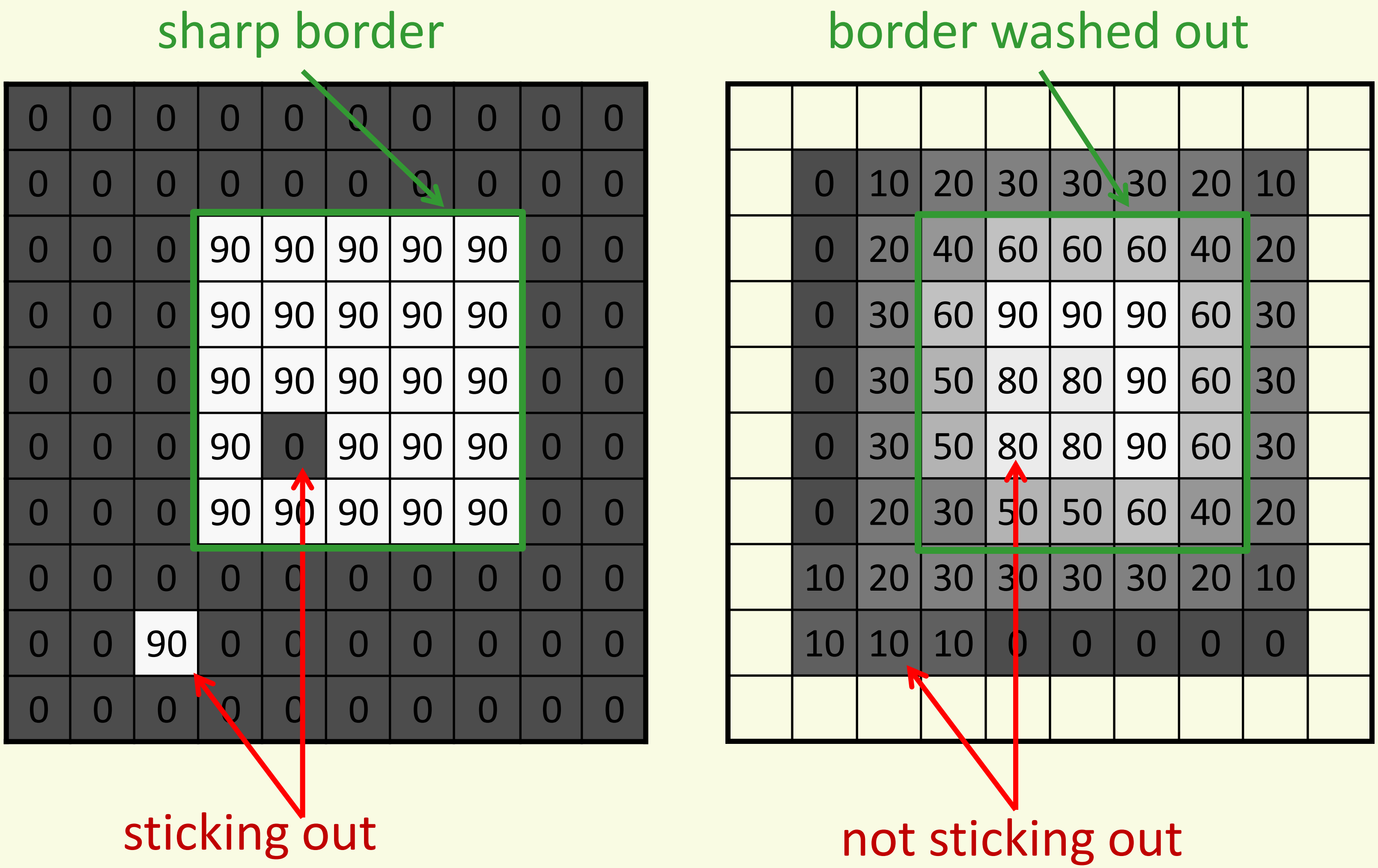
f(*x*,*y*)

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

g(*x*,*y*)

	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	

Image Filtering: Moving Average



f(*x*,*y*)

sharp border

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

sticking out

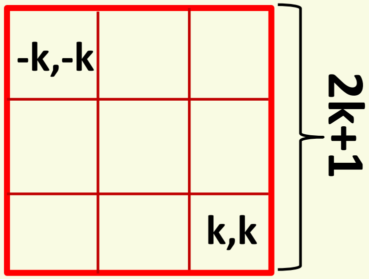
g(*x*,*y*)

border washed out

	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	

not sticking out

Correlation Filtering



- Write as equation, averaging window (2k+1)x(2k+1)

$$g(i,j)=\underbrace{\frac{1}{(2k+1)^2}}_{\text{uniform weight for each pixel}}\underbrace{\sum_{u=-k}^{k}\sum_{v=-k}^{k}f(i+u,j+v)}_{\textit{loop over all pixels in neighborhood around pixel } f\text{(i,j)}}$$

- Generalize by allowing different weights for different pixels in the neighborhood

$$g(i,j)=\sum_{u=-k}^{k}\sum_{v=-k}^{k}\underbrace{H[u,v]}_{\text{non-uniform weight for each pixel}}f(i+u,j+v)$$

Correlation filtering

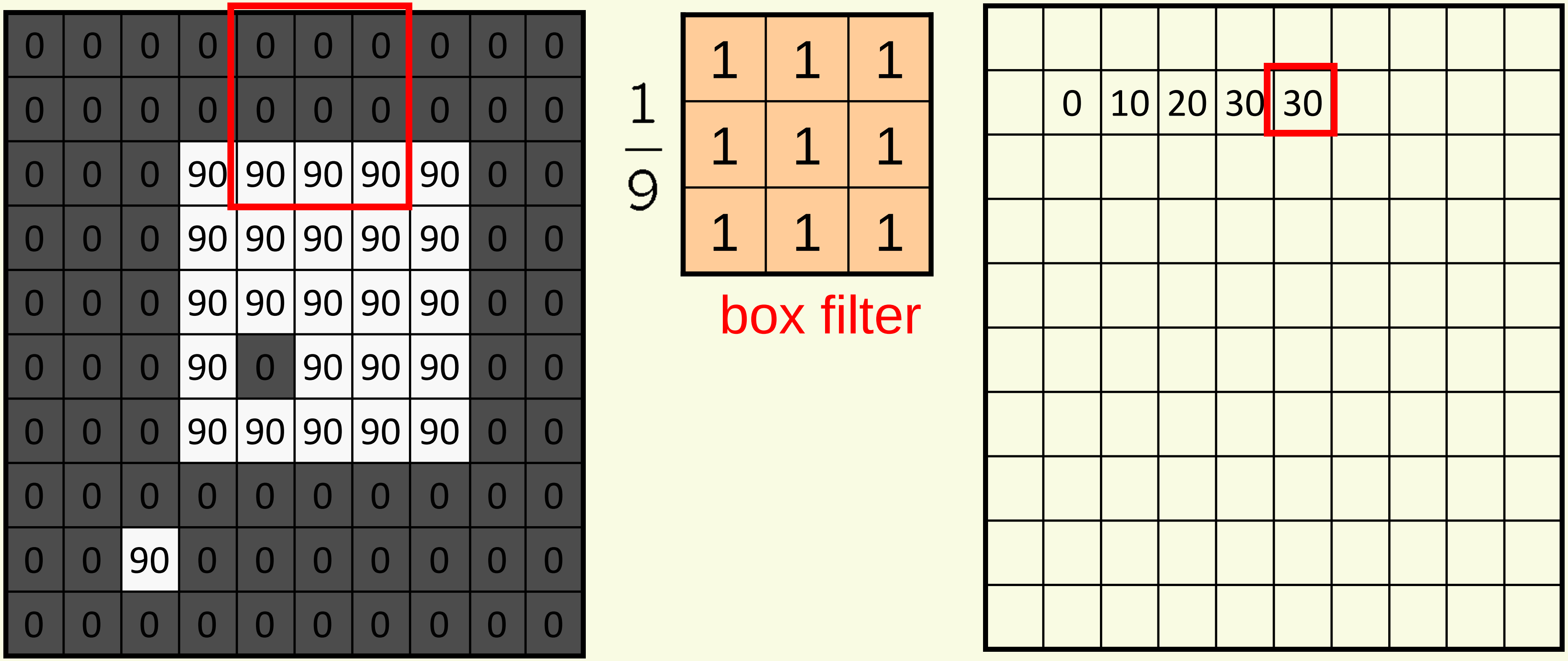
$$g(i,j)=\sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v]f(i+u,j+v)$$

- This is called cross-correlation, denoted $g = H \otimes f$
- Filtering an image: replace each pixel with a linear combination of its neighbors
- The filter kernel or mask H is gives the weights in linear combination

Averaging Filter

- What is kernel *H* for the moving average example?

$f(x,y)$ $\quad$ $H[u,v] = ?$ $\quad$ $g(x,y)$



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

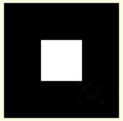
$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

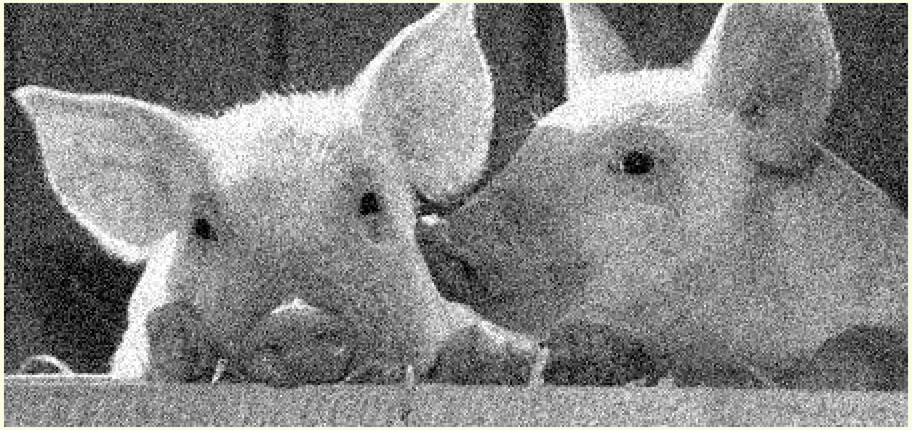
box filter

	0	10	20	30	30				

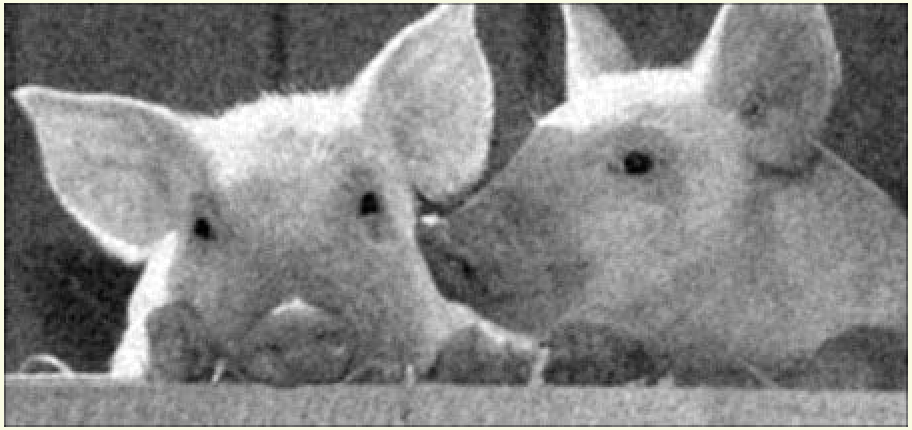
$$g = H \otimes f$$

Smoothing by Averaging

- Pictorial representation of box filter: 
  - white means large value, black means low value



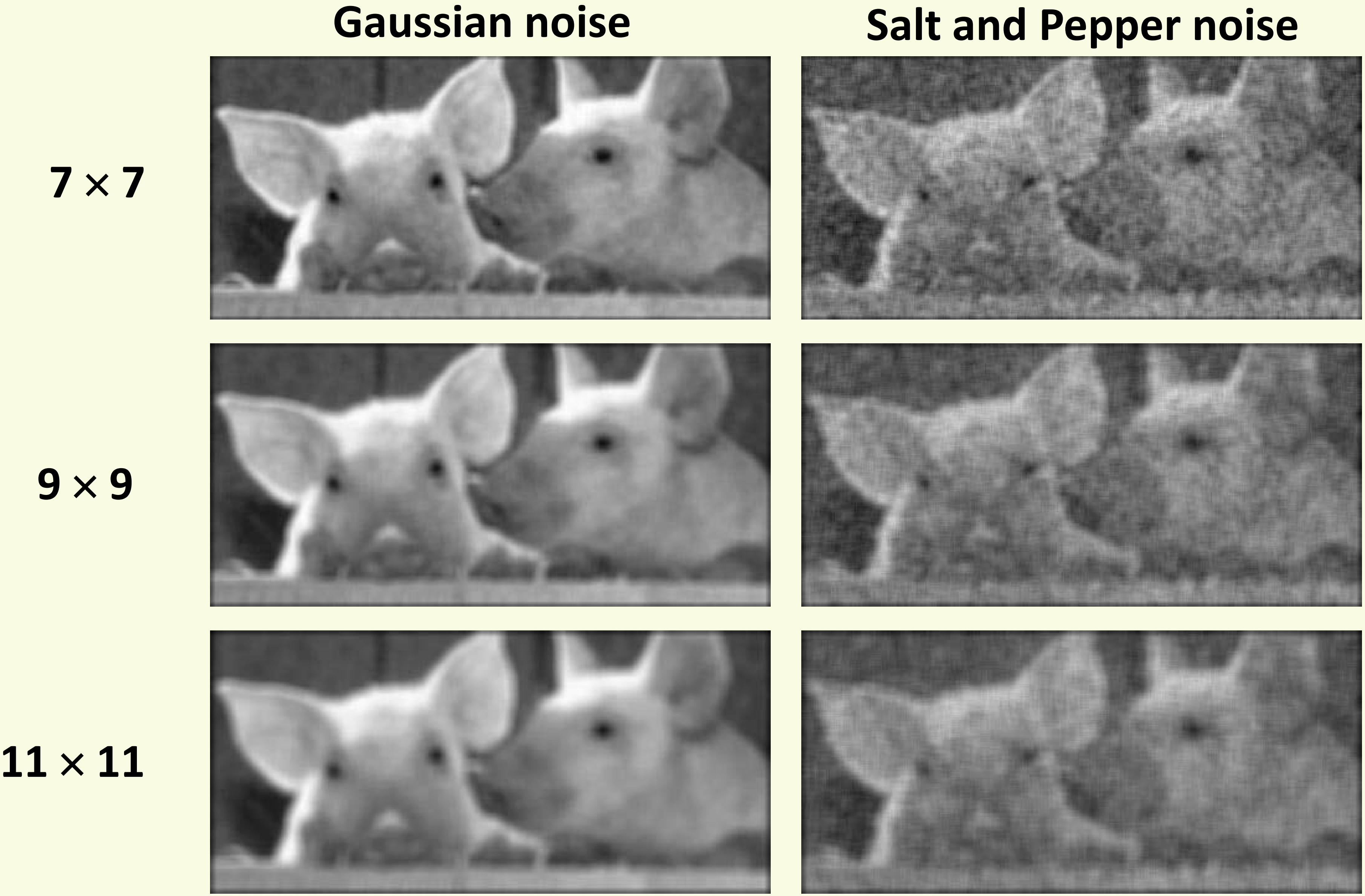
original



filtered

- What if the mask is larger than 3x3 ?

Effect of Average Filter



Gaussian noise

Salt and Pepper noise

7 × 7

9 × 9

11 × 11

Gaussian Filter

- May want nearest neighboring pixels to have the most influence

f(*x*,*y*)

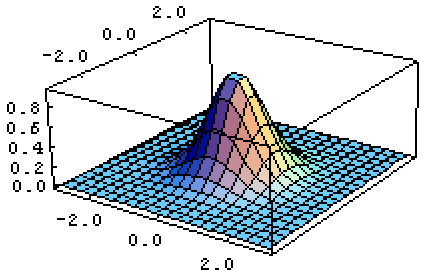
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

H[*u*,*v*]

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

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

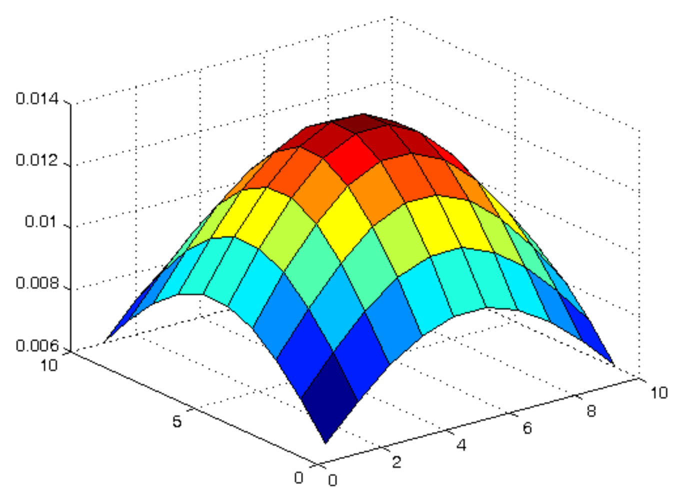
$$h(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



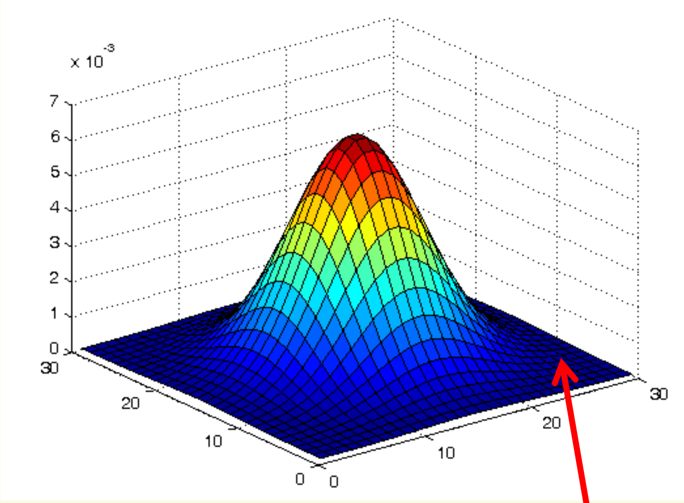
Gaussian Filters: Mask Size

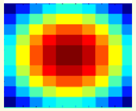
- Gaussian has infinite domain, discrete filters use finite mask
  - larger mask contributes to more smoothing

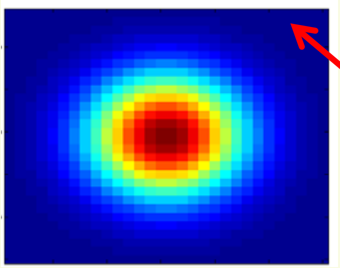
σ = 5 with 10 x 10 mask



σ = 5 with 30 x 30 mask





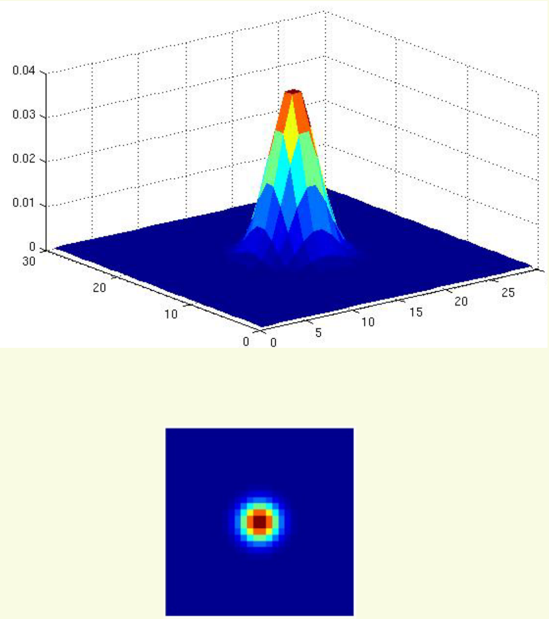


blue weights are so small they are effectively **0**

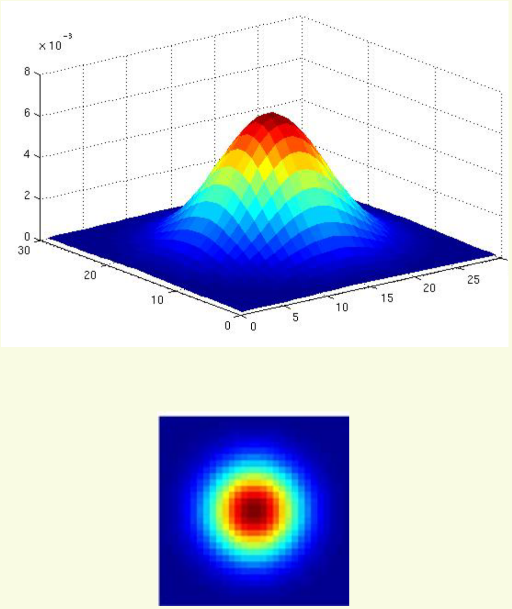
Gaussian filters: Variance

- Variance (*σ*) also contributes to the extent of smoothing
  - larger *σ* gives less rapidly decreasing weights → can construct a larger mask with non-negligible weights

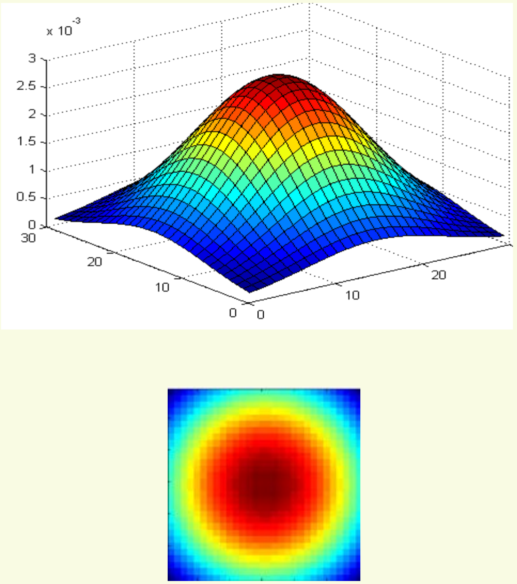
σ = 2 with 30 x 30 kernel



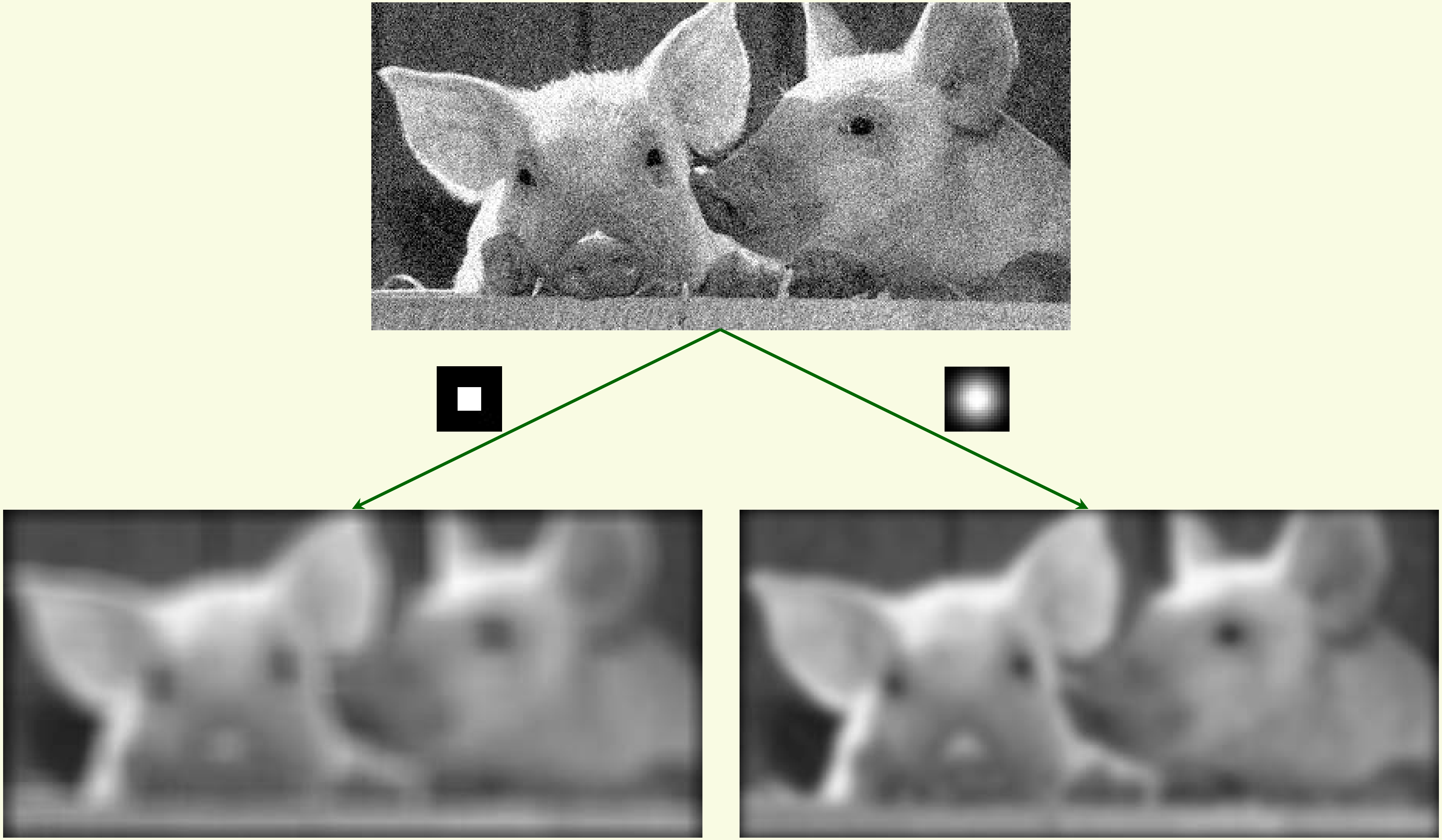
σ = 5 with 30 x 30 kernel



σ = 8 with 30 x 30 kernel



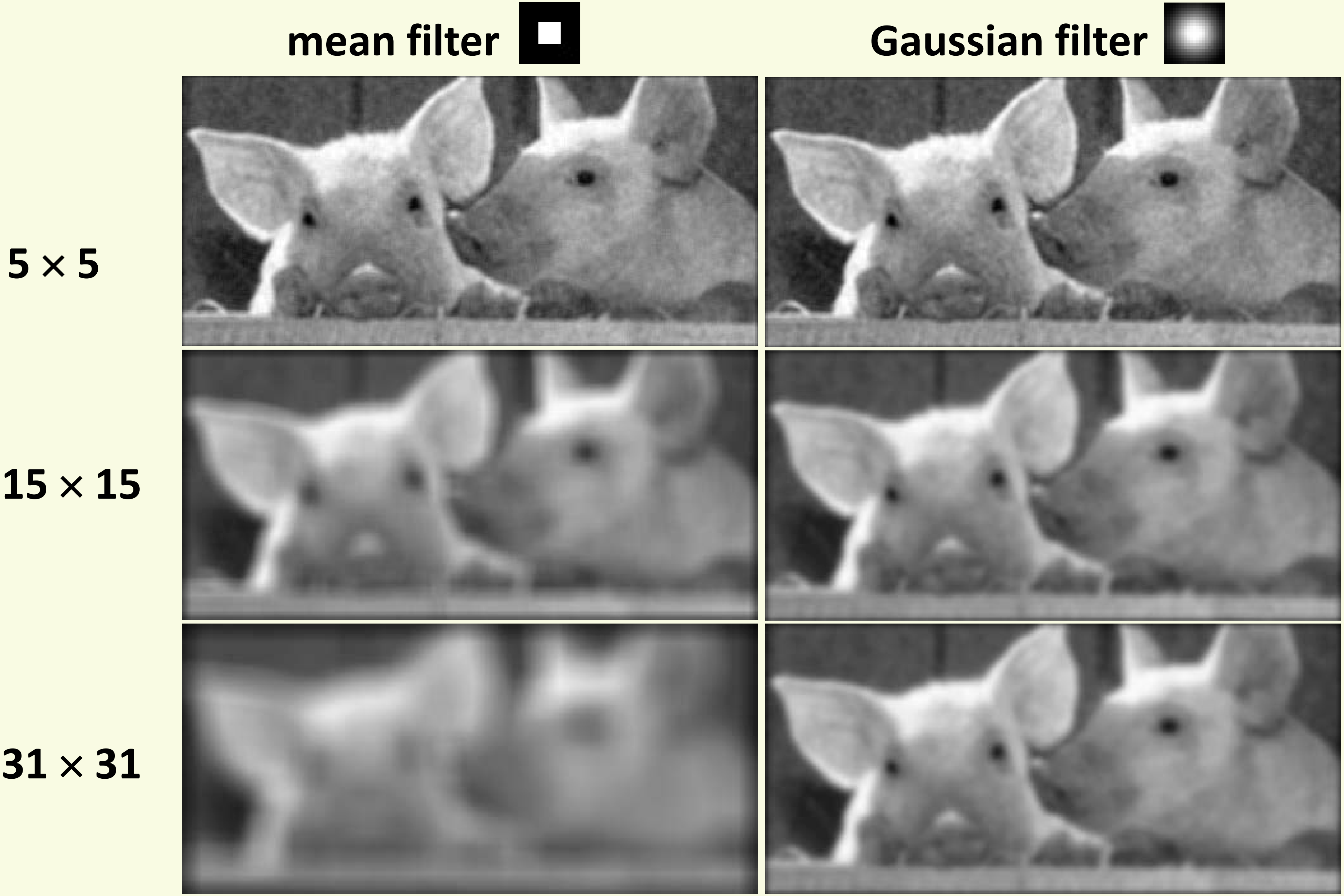
Average vs. Gaussian Filter



mean filter

Gaussian filter

More Average vs. Gaussian Filter



mean filter

Gaussian filter

5 × 5

15 × 15

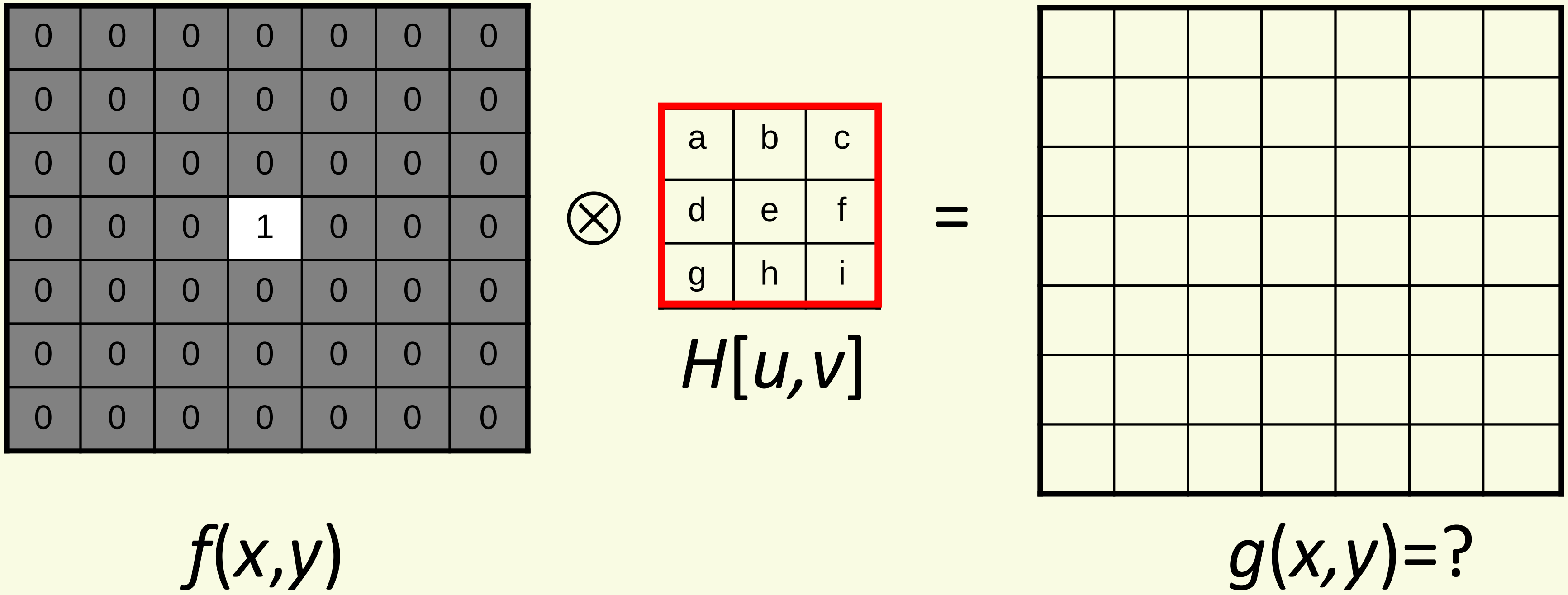
31 × 31

Properties of Smoothing Filters

- Values positive
- Sum to 1
  - constant regions same as input
  - overall image brightness stays unchanged
- Amount of smoothing proportional to mask size
  - larger mask means more extensive smoothing

Filtering an Impulse Signal

- What is the result of filtering the impulse signal (image) with arbitrary kernel *H*?



0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$f(x,y)$

$\otimes$

a	b	c
d	e	f
g	h	i

$H[u,v]$

=

$g(x,y)$=?

Filtering an Impulse Signal

- What is the result of filtering the impulse signal (image) with arbitrary kernel *H*?

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$f(x,y)$

$\otimes$

a	b	c
d	e	f
g	h	i

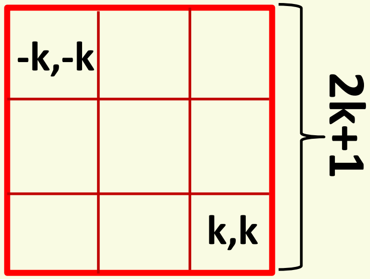
$H[u,v]$

=

		i	h	g		
		f	e	d		
		c	b	a		

$g(x,y)$=?

Convolution



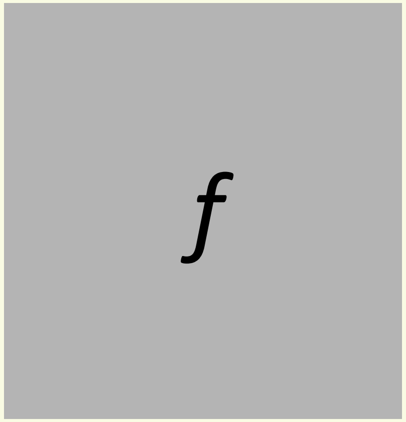
- Convolution:
  - Flip the mask in both dimensions
    - bottom to top, right to left
  - Then apply cross-correlation

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] f(i-u, j-v)$$

H

H

flipped



- Notation for convolution: $g = H*f$

Convolution vs. Correlation

- Convolution: g = H*f

$$g(i,j)=\sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v]f(i-u,j-v)$$

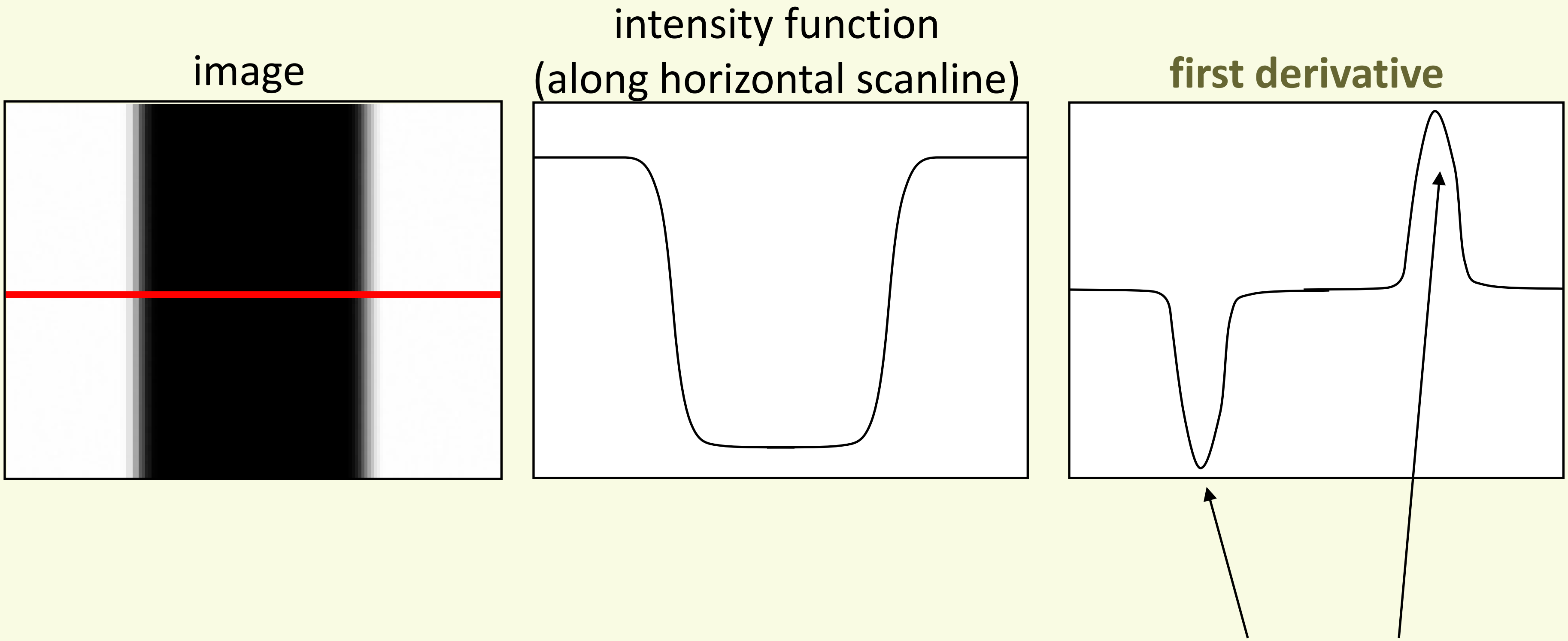
- Correlation: g = $H \otimes f$

$$g(i,j)=\sum_{u=-k}^{k}\sum_{v=-k}^{k} H[u,v]f(i+u,j+v)$$

- For Gaussian or box filter, how the outputs differ?
- If the input is an impulse signal, how the outputs differ?

Derivatives and Edges

- An edge is a place of rapid change in intensity



image

intensity function (along horizontal scanline)

first derivative

edges correspond to extrema of derivative

Derivatives with Convolution

- For 2D function *f*(*x*,*y*), partial derivative in horizontal direction

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

- For discrete data, approximate

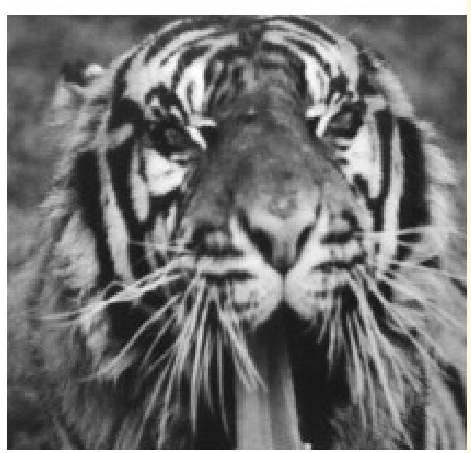
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

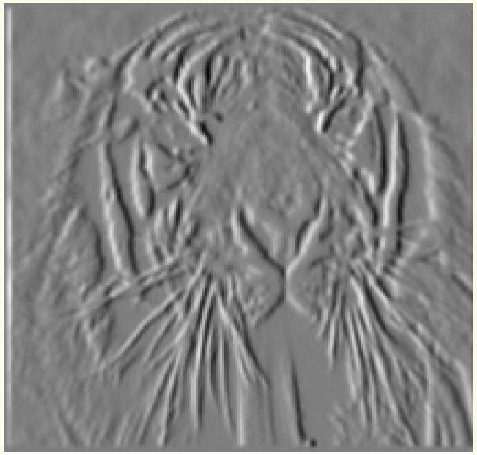
- Similarly, approximate vertical partial derivative (wrt y)

- How to implement as a convolution?

Image Partial Derivatives

Which is with respect to x?



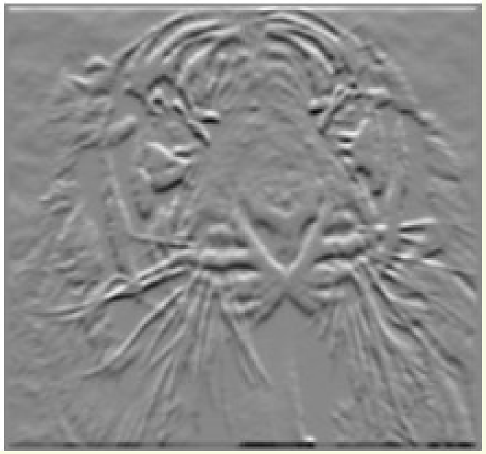


$$\frac{\partial f(x,y)}{\partial x}$$

-1	1

or

1	-1



$$\frac{\partial f(x,y)}{\partial y}$$

-1
1

or

1
-1

Finite Difference Filters

- Other filters for derivative approximation

Prewitt:

$$H_x = \frac{1}{6}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad H_y = \frac{1}{6}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Sobel:

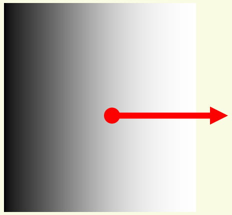
$$H_x = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad H_y = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Image Gradient

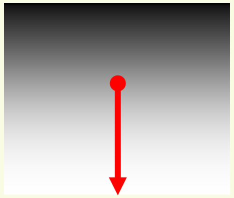
- Combine both partial derivatives into vector $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

  image gradient

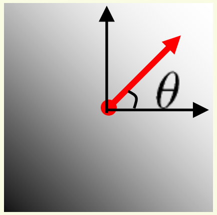
- Gradient points in the direction of most rapid increase in intensity



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



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



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

- **Direction** perpendicular to edge:

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

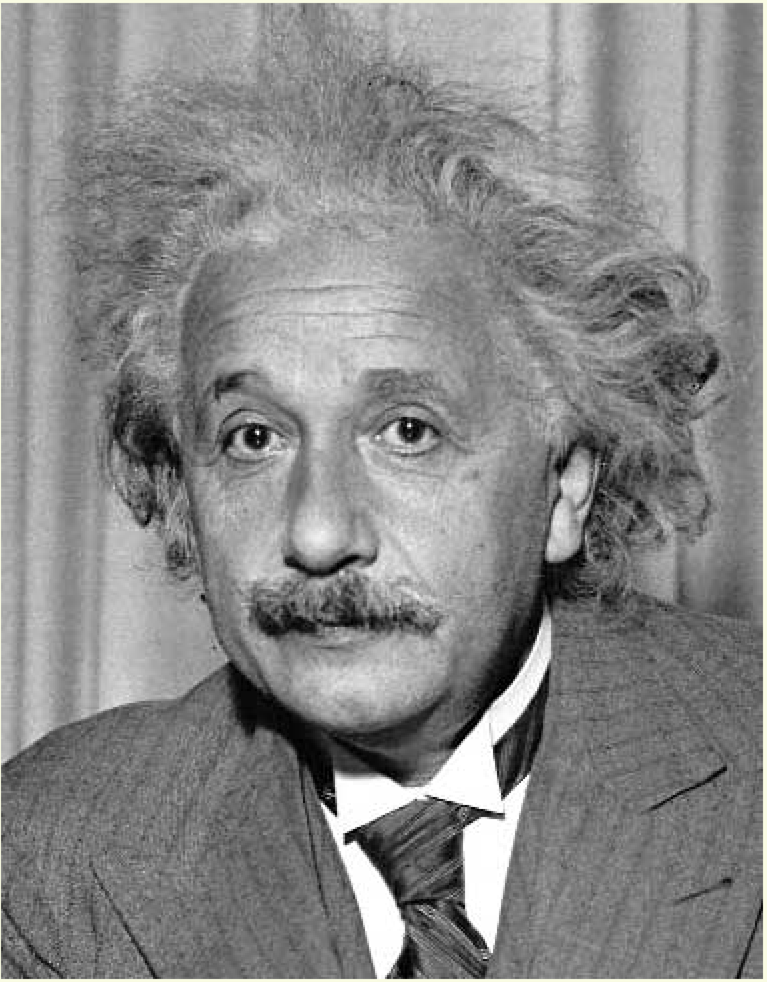
gradient orientation

- Edge **strength**

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

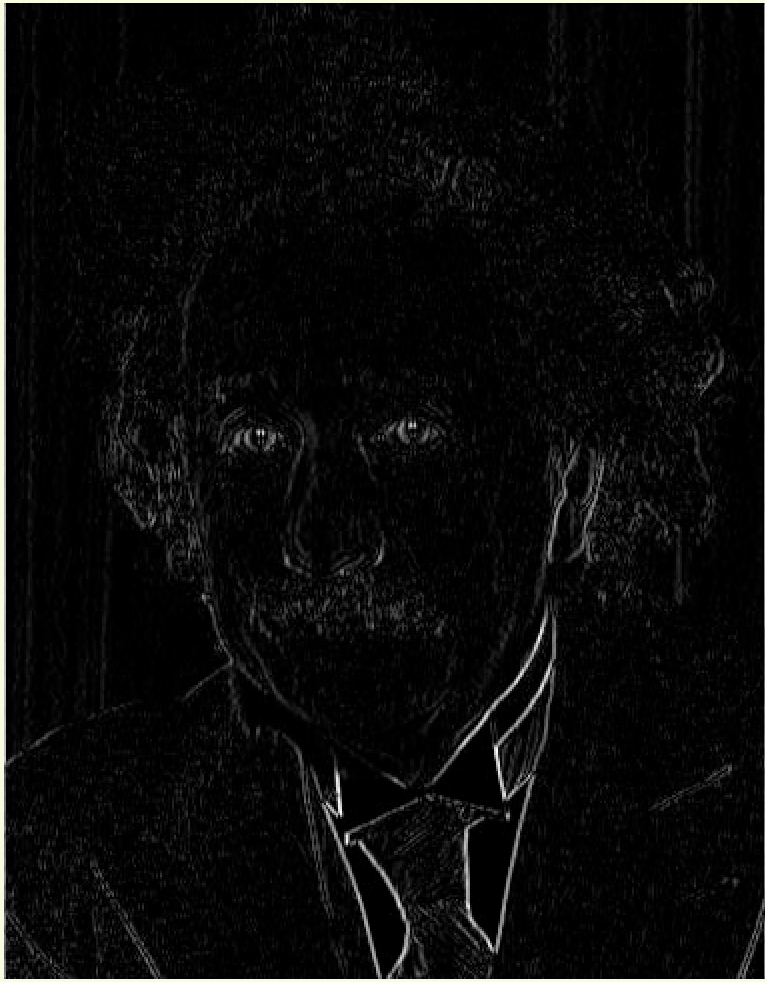
gradient magnitude

Sobel Filter for Vertical Gradient Component



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

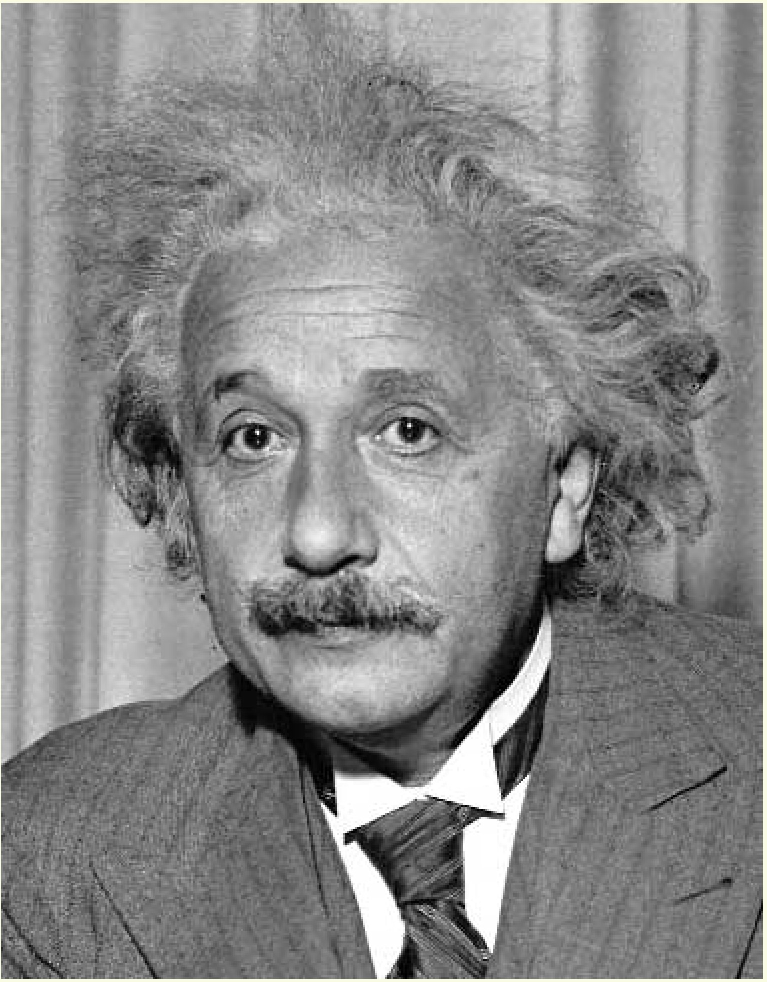
Sobel



Vertical Edge
(absolute value)

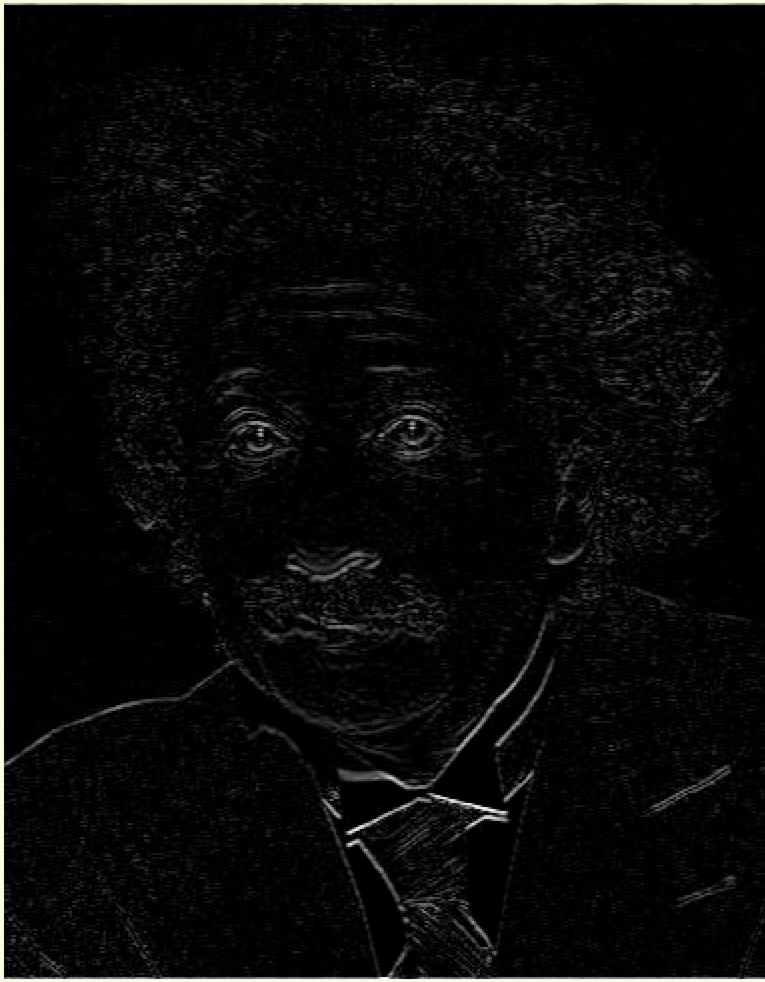
Slide Credit: D. Hoeim

Sobel Filter for Horizontal Gradient Component



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

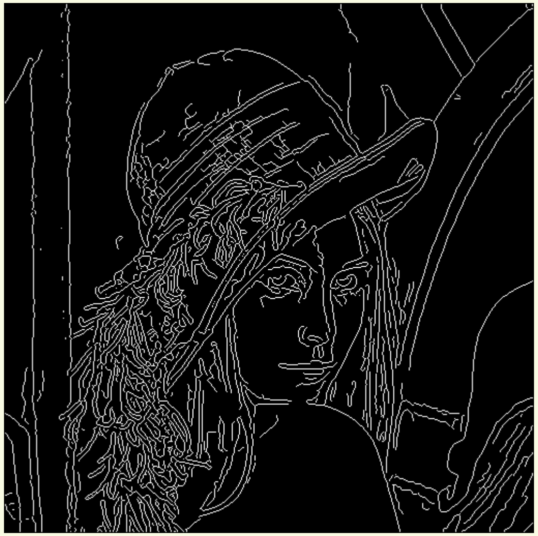
Sobel



Horizontal Edge
(absolute value)

Slide Credit: D. Hoeim

Edge Detection

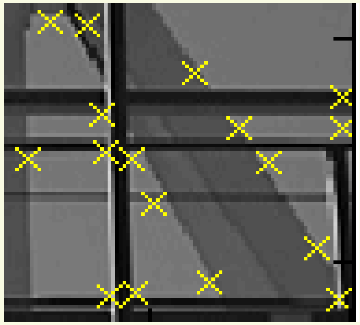


canny edge detector

- Smooth image
  - gets rid of noise and small detail
- Compute Image gradient (with Sobel filter, etc)
- Pixels with large gradient magnitude are marked as edges
- Can also apply non-maximum suppression to “thin” the edges and other post-processing

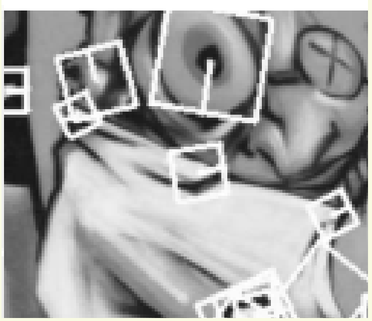
Image Features

- Edge features capture places where something interesting is happening
  - large change in image intensity
- Edges is just one type of image features or “interest points”
- Various type of corner features, etc. are popular in vision
- Other features:



corners

stable regions

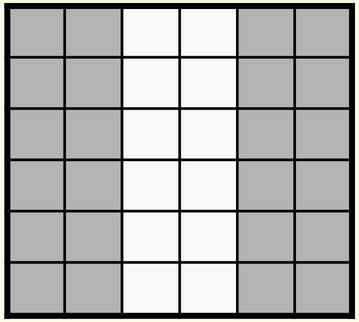


SIFT

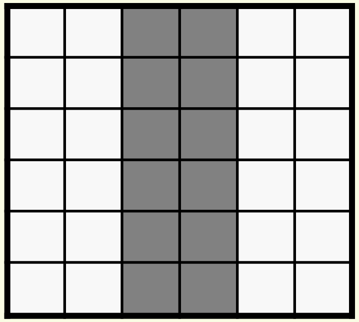
What does this Mask Detect?

- Masks “looks like” the feature it’s trying to detect

2	2	-4	-4	2	2
2	2	-4	-4	2	2
2	2	-4	-4	2	2
2	2	-4	-4	2	2
2	2	-4	-4	2	2



strong negative response

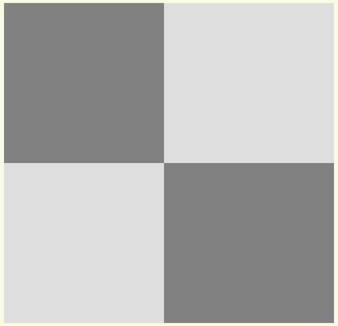


strong positive response

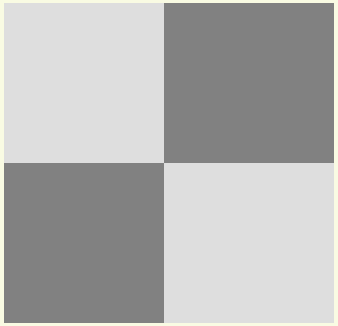
What Does this Mask Detect?

2	2	-2	-2
2	2	-2	-2
-2	-2	2	2
-2	-2	2	2

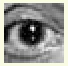
strong negative response

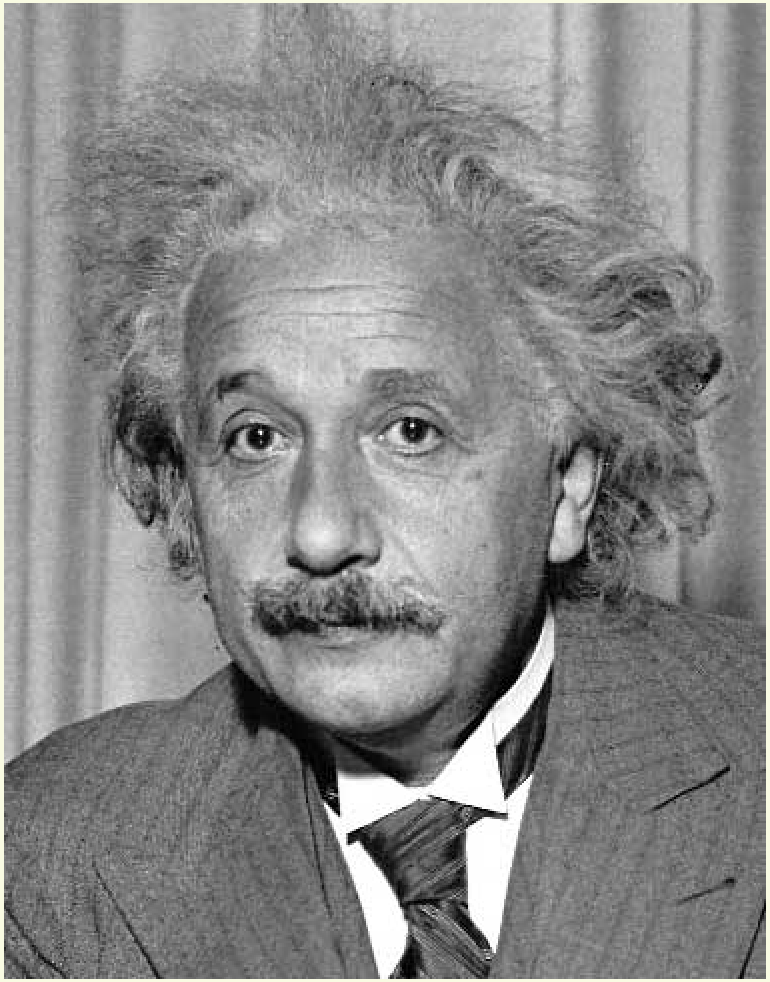


strong positive response



Template matching

- Goal: find  in image
- Main challenge: What is a good similarity or distance measure between two patches?
    - Correlation
    - Zero-mean correlation
    - Sum Square Difference
    - Normalized Cross Correlation



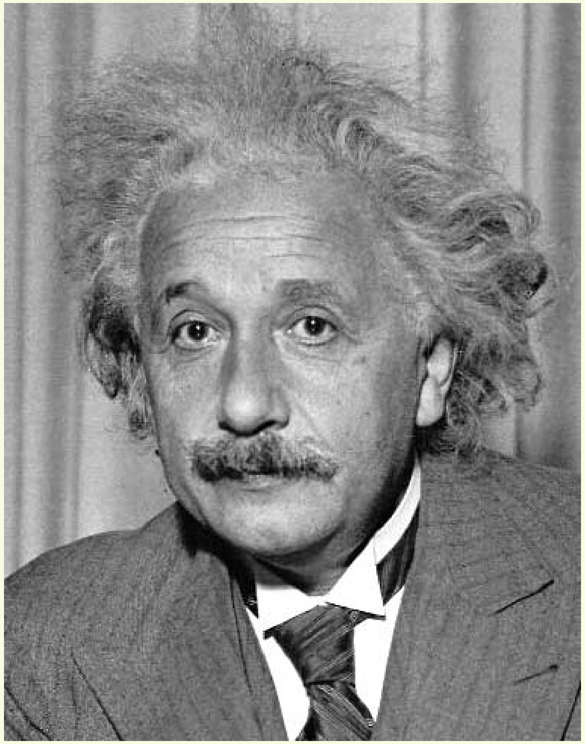
Slide Credit: D. Hoeim

Method 0: Correlation

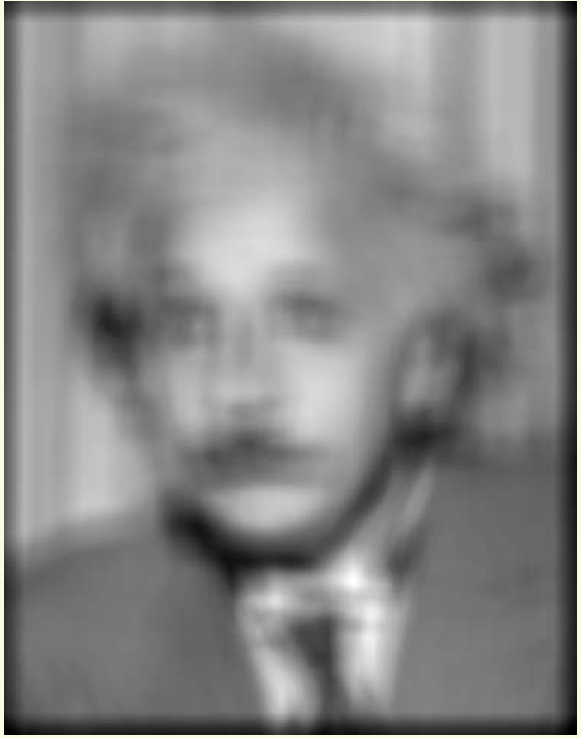
- Goal: find in image
- Filter the image with H = “eye patch”

$$g[m,n] = \sum_{k,l} H[k,l]\, f[m+k, n+l]$$

f = image
H = filter



Input



Filtered Image

What went wrong?

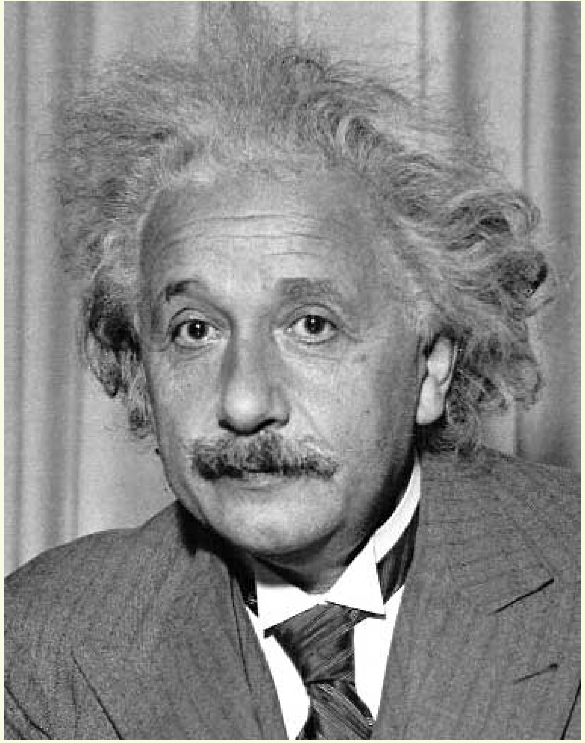
Slide Credit: D. Hoeim

Method 1: zero-mean Correlation

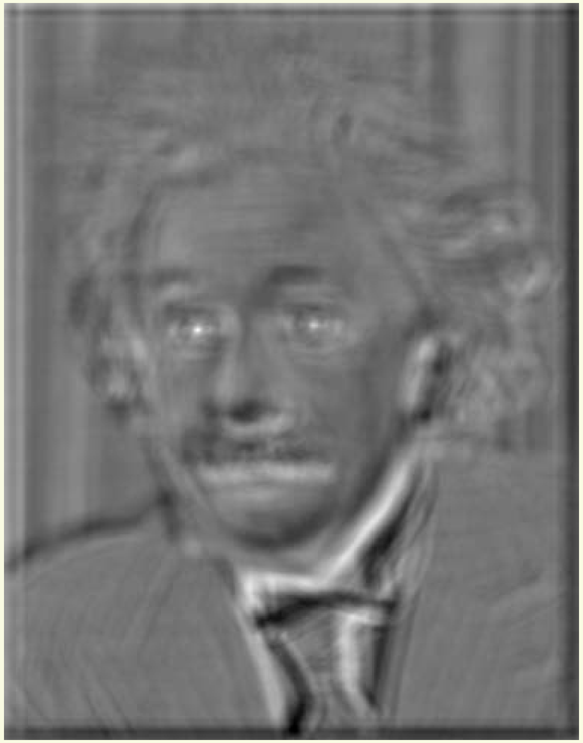
- Goal: find in image
- Filter the image with zero-mean eye

$$g[m,n] = \sum_{k,l} (H[k,l] - \overline{H})\,(f[m+k, n+l])$$

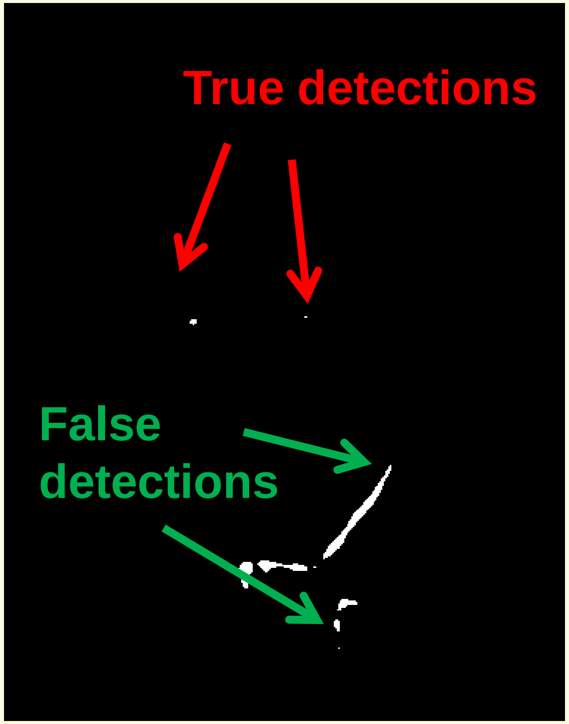
mean of template H



Input



Filtered Image (scaled)



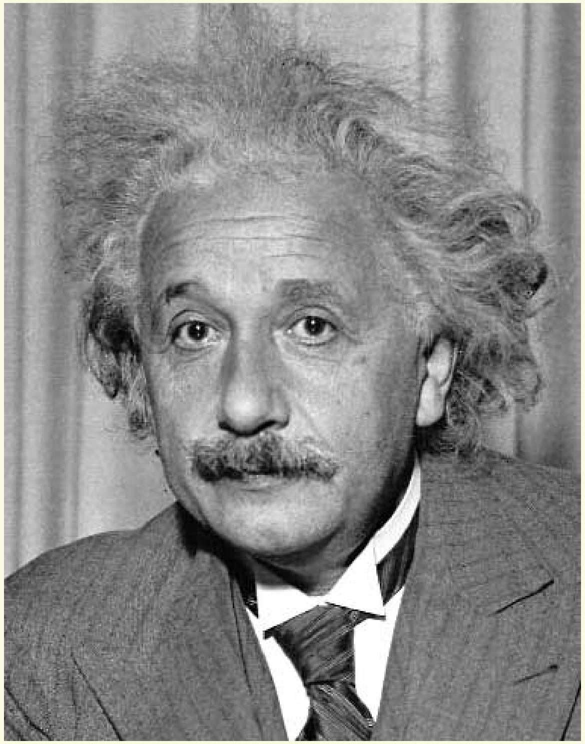
Thresholded Image

Slide Credit: D. Hoeim

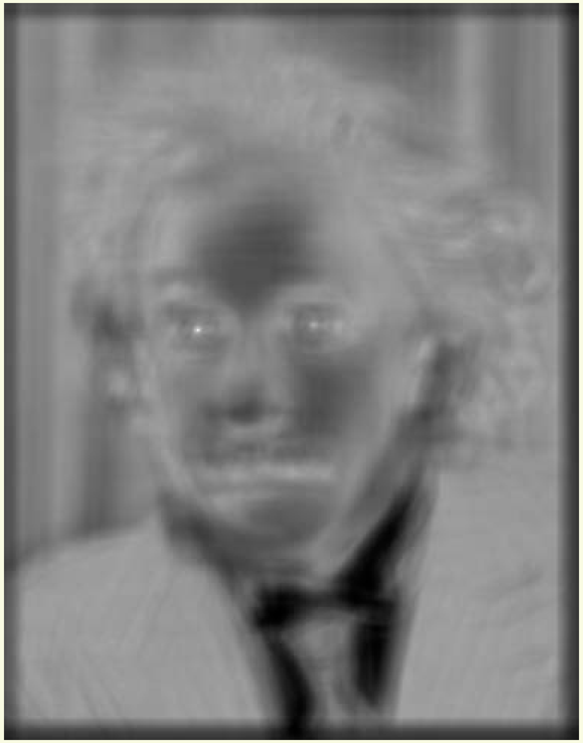
Method 3: Sum of Squared Differences

- Goal: find in image

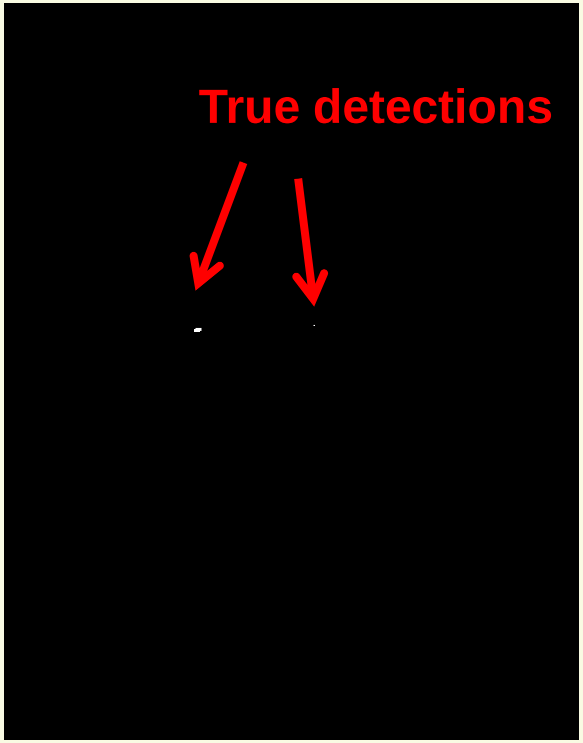
$$g[m,n] = \sum_{k,l} (H[k,l] - f[m+k,n+l])^2$$



Input



1- sqrt(SSD)



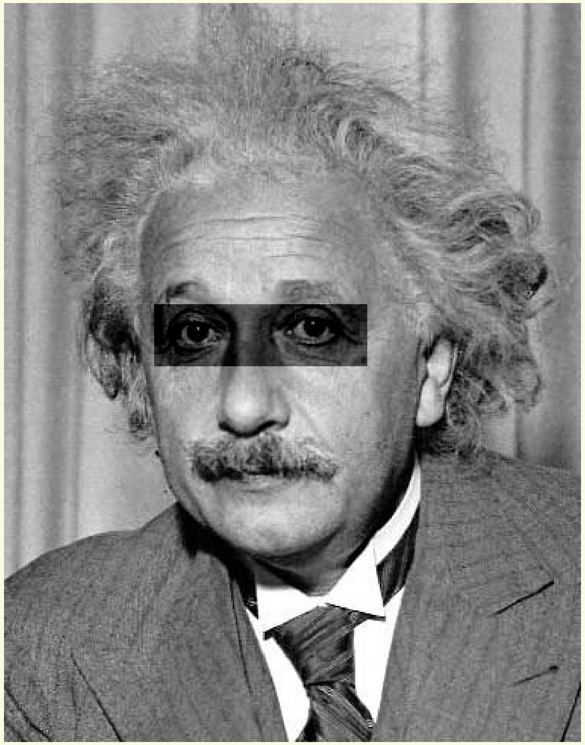
True detections

Thresholded Image

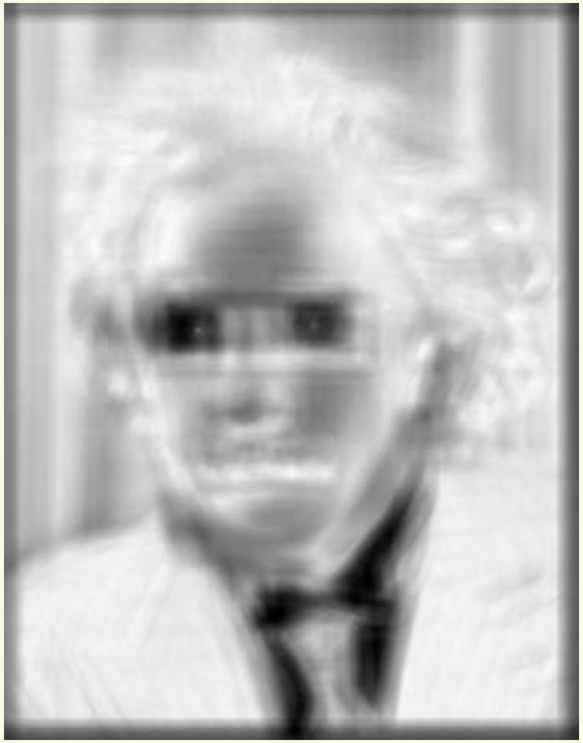
Slide Credit: D. Hoeim

Problem with SSD

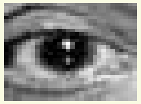
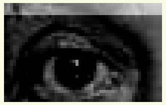
- SSD is sensitive to changes in brightness

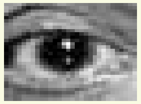
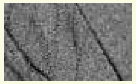


Input



1- sqrt(SSD)

( - )^2 = large

( - )^2 = medium

Slide Credit: D. Hoeim

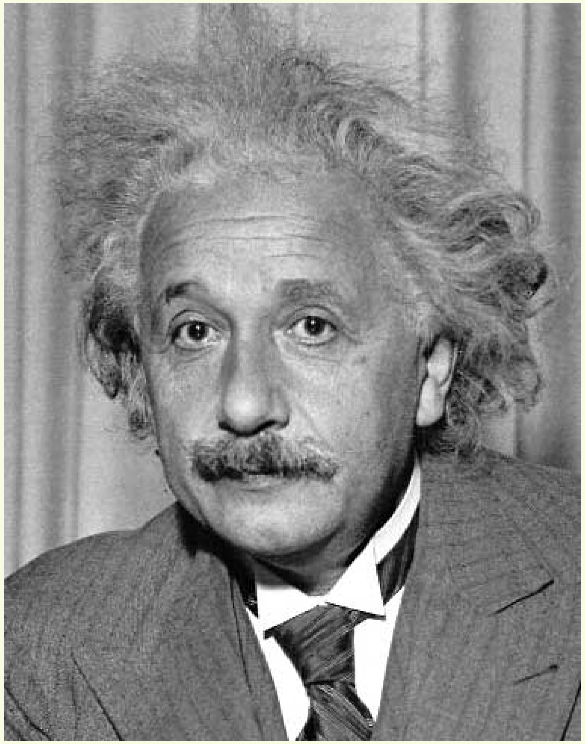
Method 3: Normalized Cross-Correlation

- Goal: find in image

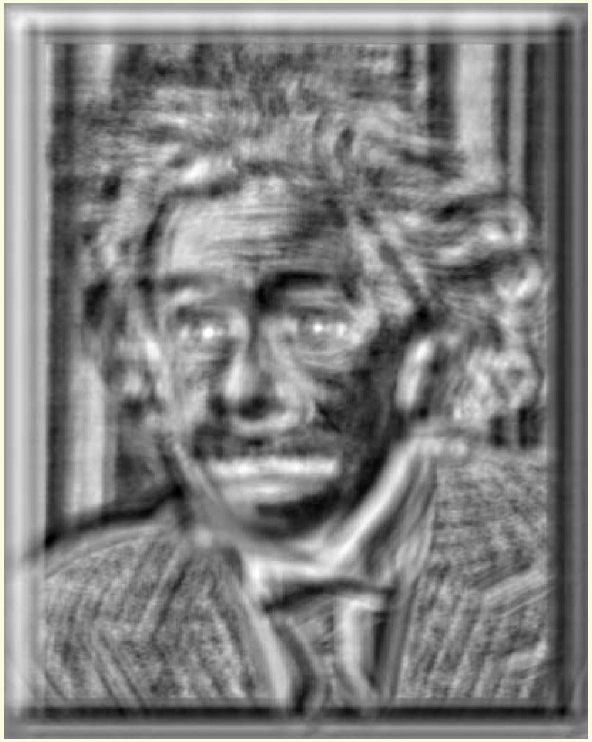
$$g[m,n]=\frac{\sum_{k,l}(H[k,l]-\overset{\substack{\text{mean template}\\ \downarrow}}{\overline{H}})(f[m+k,n+l]-\overset{\substack{\text{mean image patch}\\ \downarrow}}{\bar{f}_{m,n}})}{\left(\sum_{k,l}(H[k,l]-\overline{H})^2\sum_{k,l}(f[m+k,n+l]-\bar{f}_{m,n})^2\right)^{0.5}}$$

Slide Credit: D. Hoeim

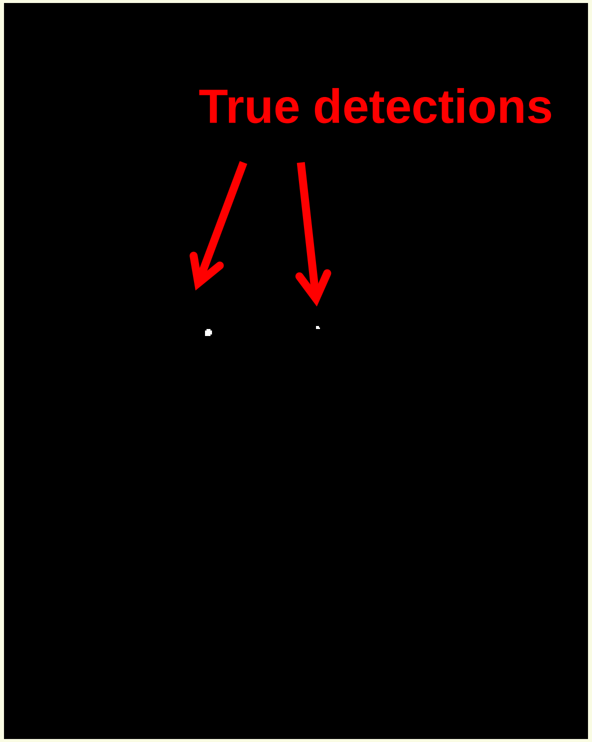
Method 3: Normalized Cross-Correlation



Input



Normalized X-Correlation



True detections

Thresholded Image

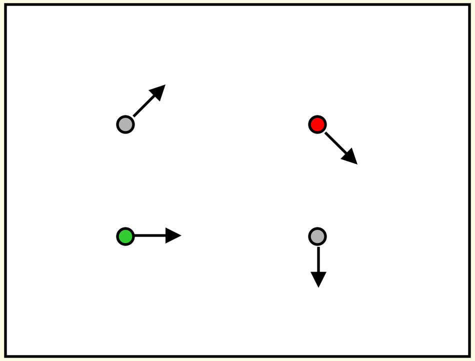
Slide Credit: D. Hoeim

Comparison

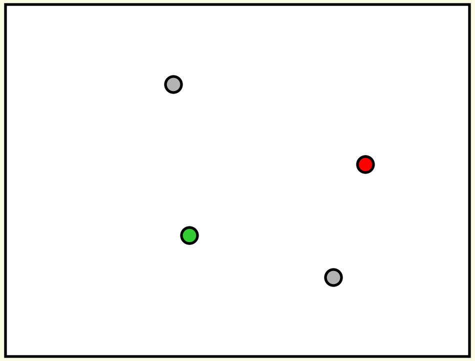
- Zero-mean filter: fastest but not a great matcher
- SSD: next fastest, sensitive to overall intensity
- Normalized cross-correlation: slowest, but invariant to local average intensity and contrast

Slide Credit: D. Hoeim

Optical flow



first image I_1

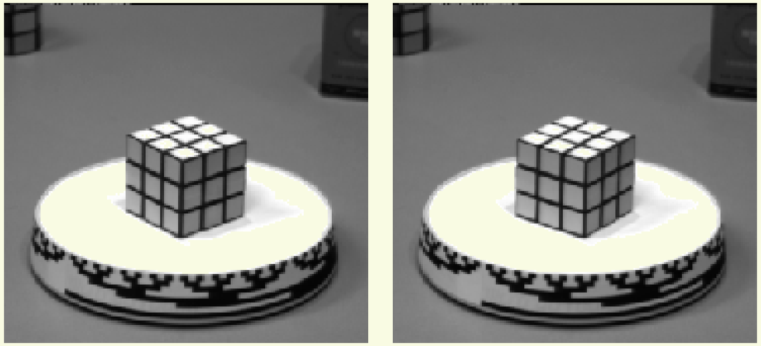


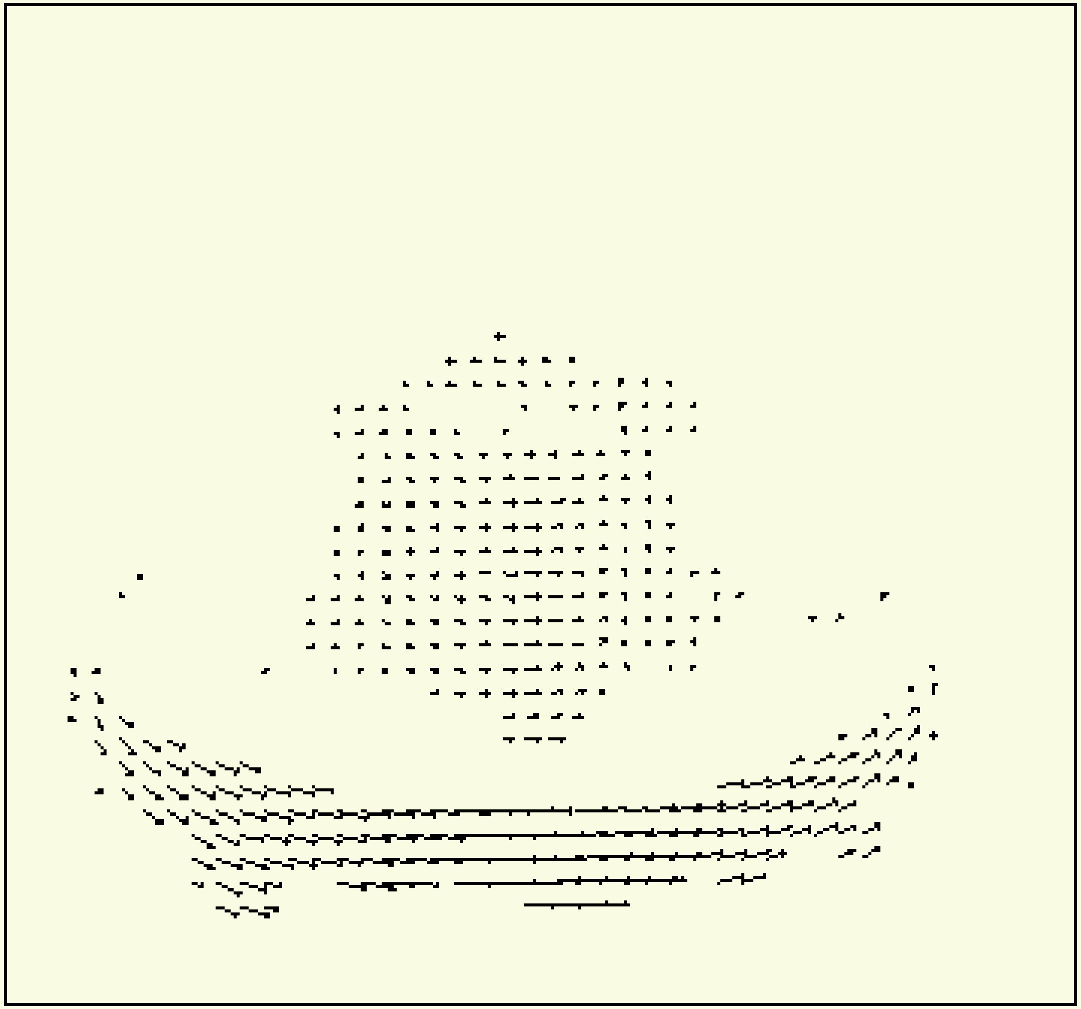
second image I_2

- How to estimate pixel motion from image I_1 to image I_2 ?
  - Solve pixel correspondence problem
    - given a pixel in I_1 , find pixels with similar color in I_2

- Frequently made assumptions
  - **color constancy:** a point in I_1 looks the same in I_2
    - For grayscale images, this is **brightness constancy**
  - **small motion**: points do not move very far
- This is called the **optical flow** problem

Optical Flow Field





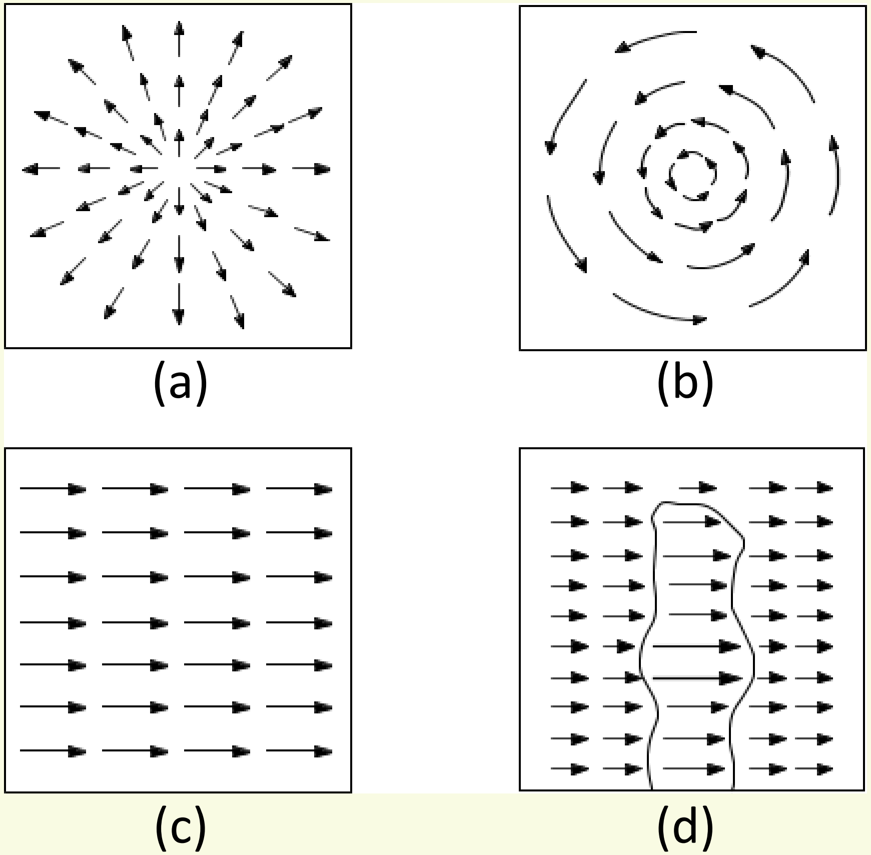
Optical Flow and Motion Field

- Optical flow field is the apparent motion of brightness patterns between 2 (or several) frames in an image sequence
- Why does brightness change between frames?
- Assuming that illumination does not change:
  - changes are due to the RELATIVE MOTION between the scene and the camera
  - There are 3 possibilities:
    - Camera still, moving scene
    - Moving camera, still scene
    - Moving camera, moving scene

Motion Field (MF)

- The MF assigns a velocity vector to each pixel in the image
- These velocities are induced by the relative motion between the camera and the 3D scene
- The MF is the *projection* of the 3D velocities on the image plane

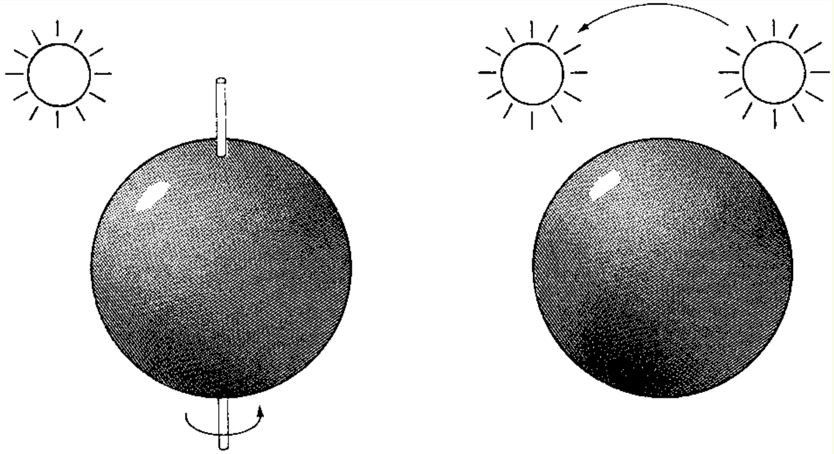
Examples of Motion Fields



(a) Translation perpendicular to a surface. (b) Rotation about axis perpendicular to image plane. (c) Translation parallel to a surface at a constant distance. (d) Translation parallel to an obstacle in front of a more distant background.

Optical Flow vs. Motion Field

- Optical Flow is the *apparent* motion of brightness patterns
- We equate Optical Flow Field with Motion Field
- Frequently works, but now always:



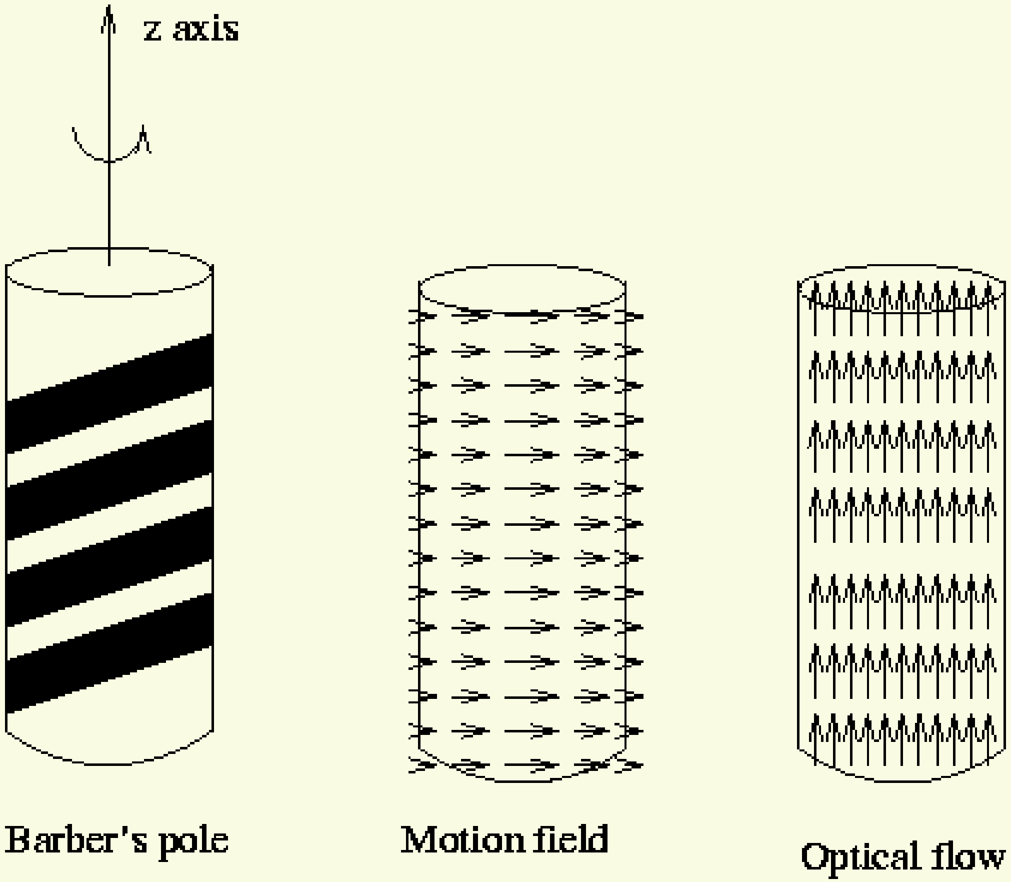
(a) (b)

(a) A smooth sphere is rotating under constant illumination. Thus the optical flow field is zero, but the motion field is not

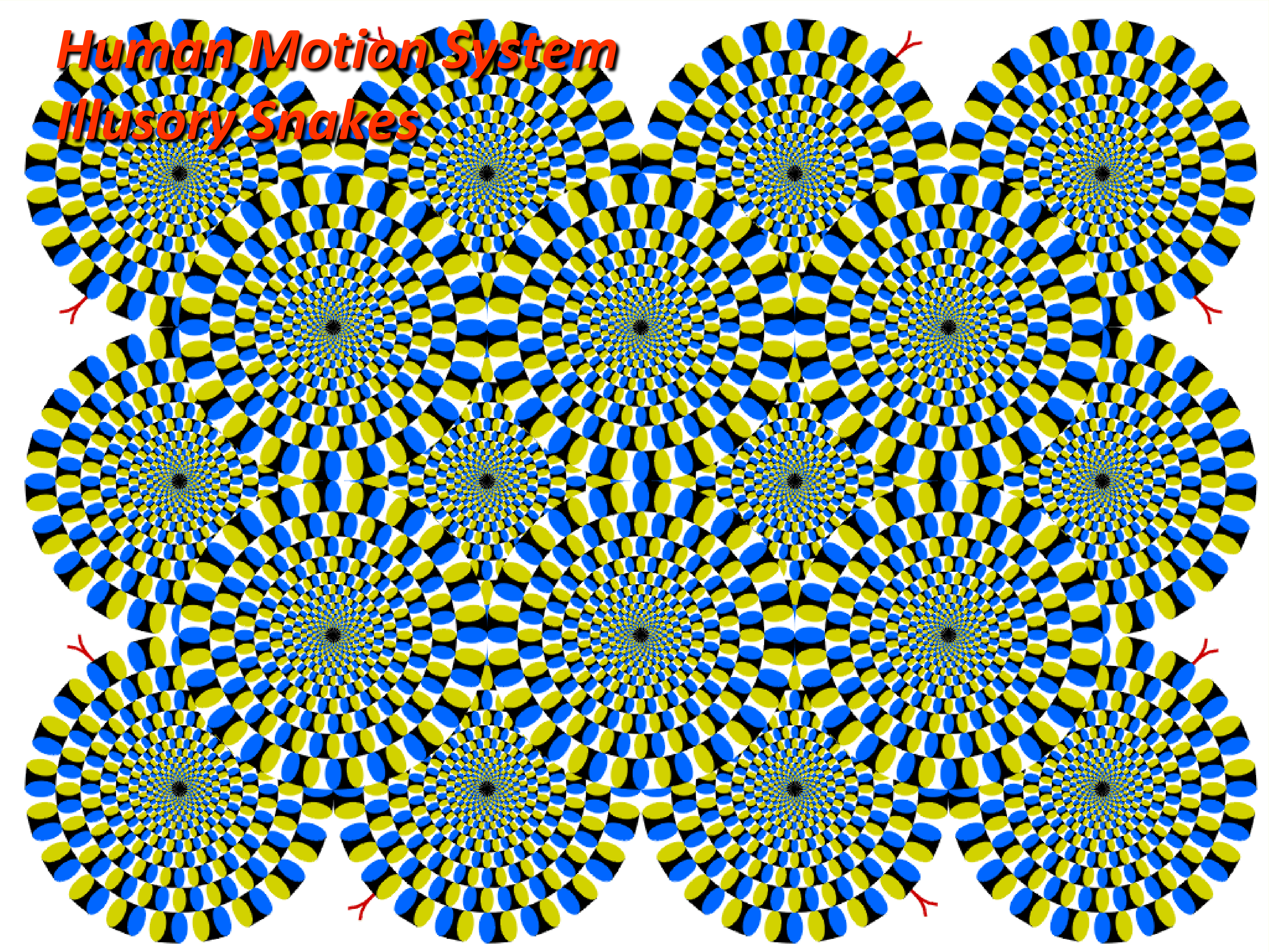
(b) A fixed sphere is illuminated by a moving source—the shading of the image changes. Thus the motion field is zero, but the optical flow field is not

Optical Flow vs. Motion Field

- Often (but not always) optical flow corresponds to the true motion of the scene

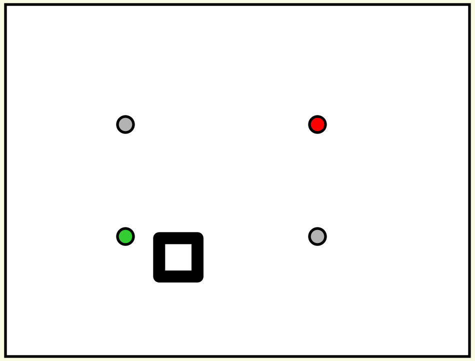


Human Motion System
Illusory Snakes

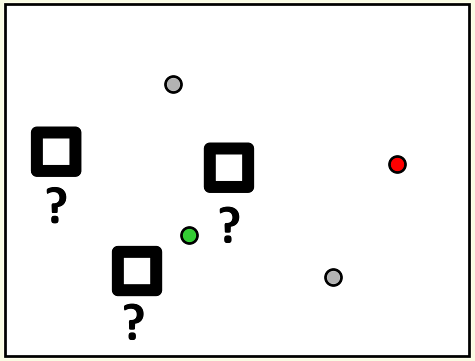


from Gary Bradski and Sebastian Thrun

Computing Optical flow: Direct Search



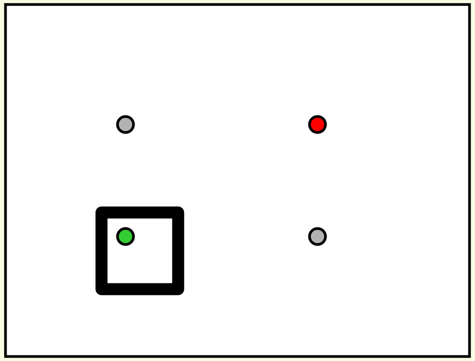
first image I_1



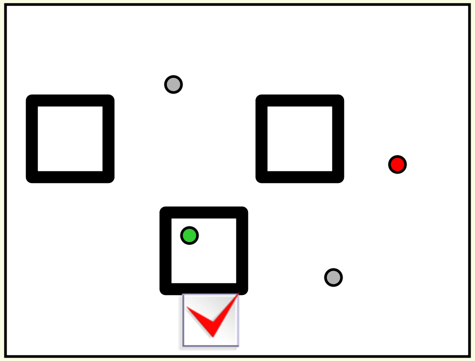
second image I_2

- Can perform direct search for pixel correspondence
- Individual pixels are not reliable to match

Computing Optical flow: Direct Search



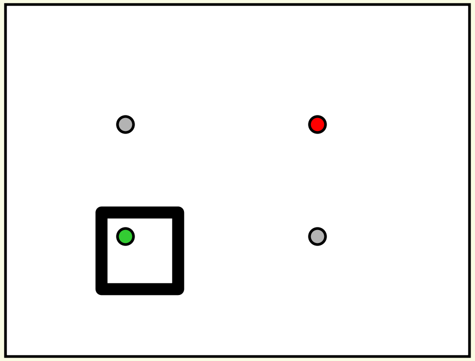
first image I_1



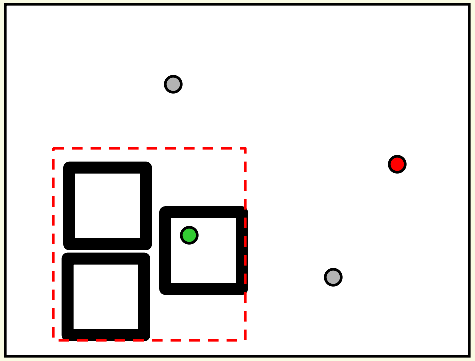
second image I_2

- Can perform direct search for pixel correspondence
- Individual pixels are not reliable to match
- For each pixel, take a patch of pixels around it, and match patches
  - Use any of template matching cost functions studied previously

Computing Optical flow: Direct Search



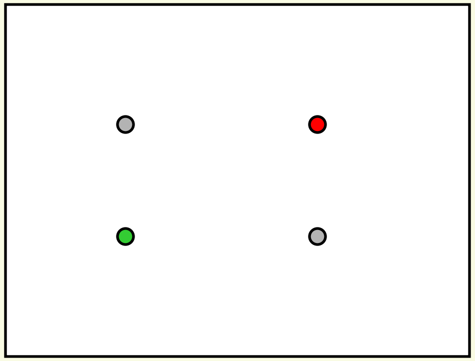
first image I_1



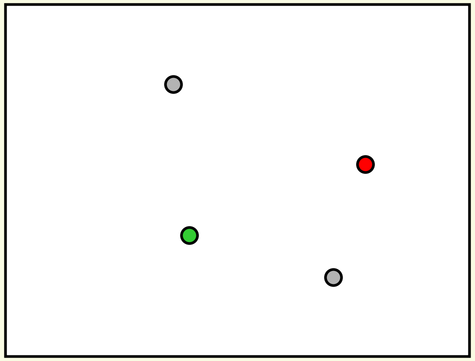
second image I_2

- Can perform direct search for pixel correspondence
- Individual pixels are not reliable to match
- For each pixel, take a patch of pixels around it, and match patches
    - Use any of template matching cost functions studied previously
- Assuming small motion lets us limit the search to a small area around pixel’s position in the first image

Computing Optical Flow without Direct Search



first image I_1



second image I_2

- Can find optical flow **without** direct search
  - Very small motion (not more than one pixel)
    - will relax this later
  - Color constancy
    - Can also be relaxed

Computing Optical Flow: Brightness Constancy Equation

- Let ***P*** be a moving point in 3D:
  - At time ***t***, ***P*** has coordinates $(\boldsymbol{X}(\boldsymbol{t}),\boldsymbol{Y}(\boldsymbol{t}),\boldsymbol{Z}(\boldsymbol{t}))$
  - Let $\boldsymbol{p}=(\boldsymbol{x}(\boldsymbol{t}),\boldsymbol{y}(\boldsymbol{t}))$ be the coordinates of its image at time ***t***
  - Let $\boldsymbol{E}(\boldsymbol{x}(\boldsymbol{t}),\boldsymbol{y}(\boldsymbol{t}),\boldsymbol{t})$ be the brightness at ***p*** at time ***t***.
- Brightness Constancy Assumption:
  - As ***P*** moves over time, $\boldsymbol{E}(\boldsymbol{x}(\boldsymbol{t}),\boldsymbol{y}(\boldsymbol{t}),\boldsymbol{t})$ remains constant

Computing Optical Flow: Brightness Constancy Equation

$$E(x(t), y(t), t) = Constant$$

- Taking derivative with respect to time:

$$\frac{dE(x(t), y(t), t)}{dt} = 0$$

- Rewriting:

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0$$

Computing Optical Flow: Brightness Constancy Equation

- This is one equation with two unknowns:

$$\frac{\partial E}{\partial x}\frac{dx}{dt}+\frac{\partial E}{\partial y}\frac{dy}{dt}+\frac{\partial E}{\partial t}=0$$

- Let's group some terms together:

$$\nabla E=\begin{bmatrix}\frac{\partial E}{\partial x}\\ \frac{\partial E}{\partial y}\end{bmatrix} \qquad \begin{bmatrix}u\\ v\end{bmatrix}=\begin{bmatrix}\frac{dx}{dt}\\ \frac{dy}{dt}\end{bmatrix} \qquad E_t=\frac{\partial E}{\partial t}$$

frame spatial gradient   optical flow   derivative across frames

- Equation becomes: $\nabla E\cdot\begin{bmatrix}u & v\end{bmatrix}=-E_t$

Computing Optical Flow: Brightness Constancy Equation

- Need to get more equations for a pixel: $\nabla E(p_i)\cdot\begin{bmatrix}u & v\end{bmatrix} = -E_t(p_i)$
- Idea: impose additional constraints
  - assume that the flow field is smooth locally
  - i.e. pretend the pixel's neighbors have the same (***u***,***v***)
    - If we use a 5x5 window, that gives us 25 equations per pixel!

$$\begin{bmatrix} E_x(p_1) & E_y(p_1) \\ E_x(p_2) & E_y(p_2) \\ \vdots & \vdots \\ E_x(p_{25}) & E_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} E_t(p_1) \\ E_t(p_2) \\ \vdots \\ E_t(p_{25}) \end{bmatrix}$$

matrix ***E***
25x2

vector ***d***
2x1

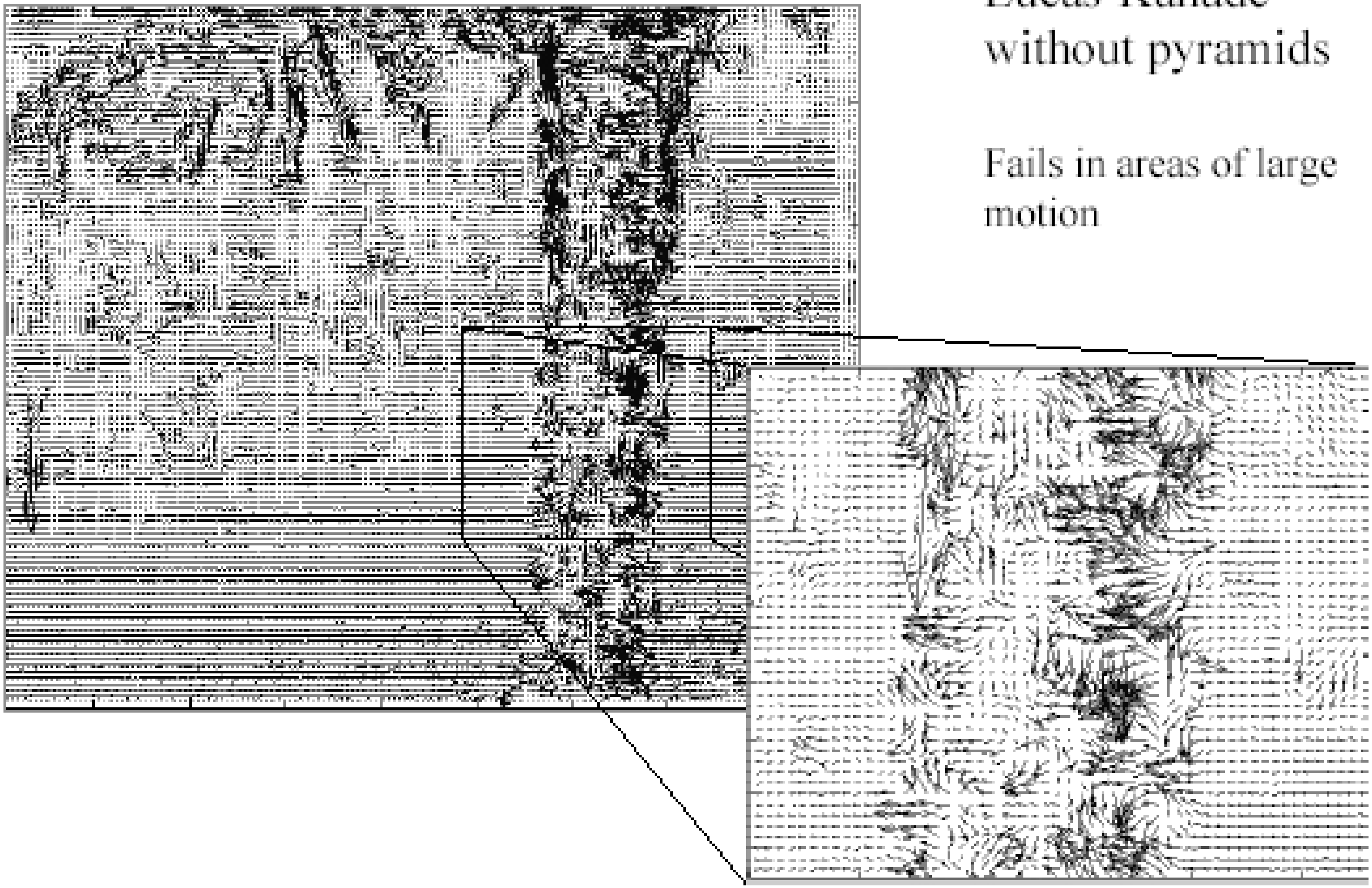
vector ***b***
25x1

Video Sequence

*

* Picture from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Optical Flow Results



Lucas-Kanade
without pyramids

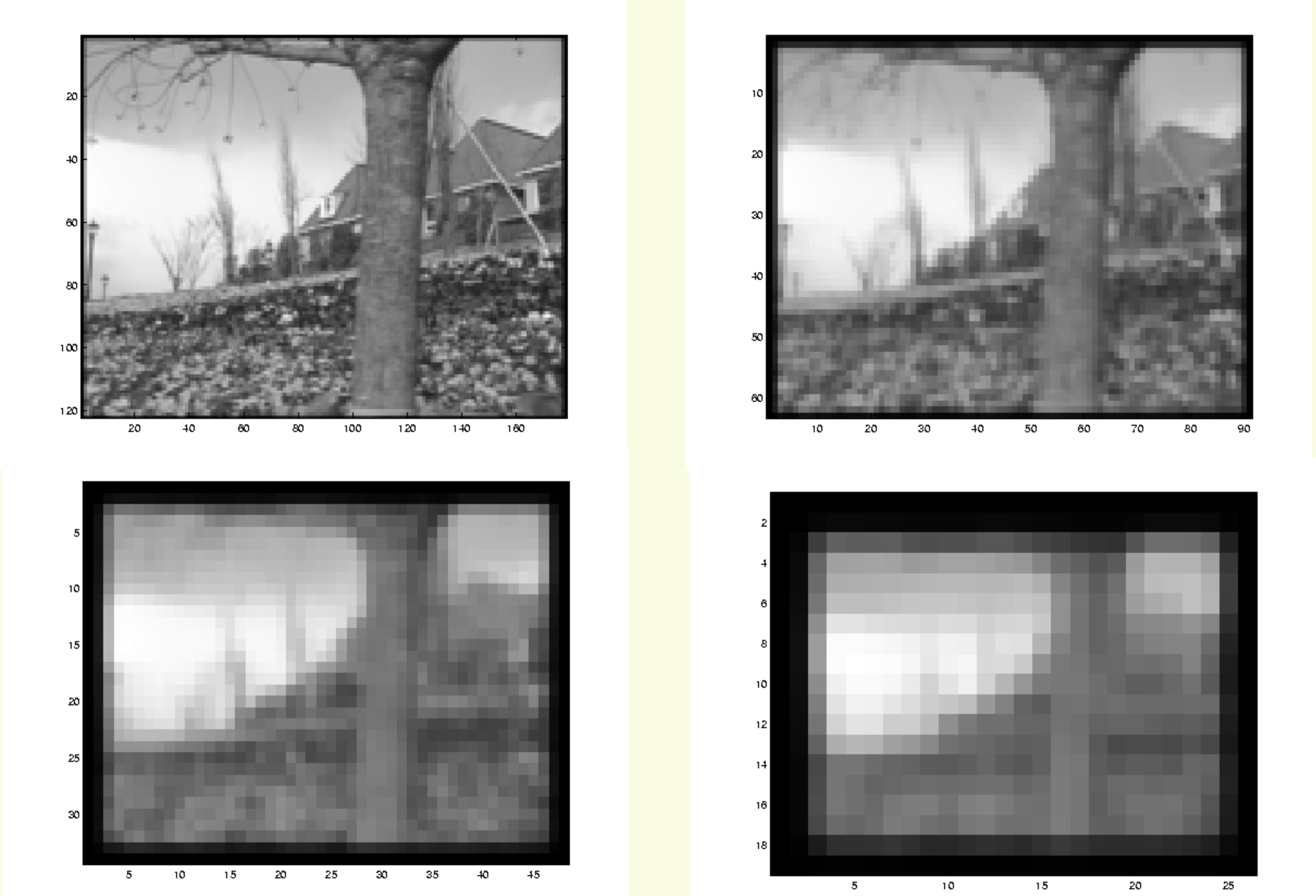
Fails in areas of large motion

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Revisiting the small motion assumption

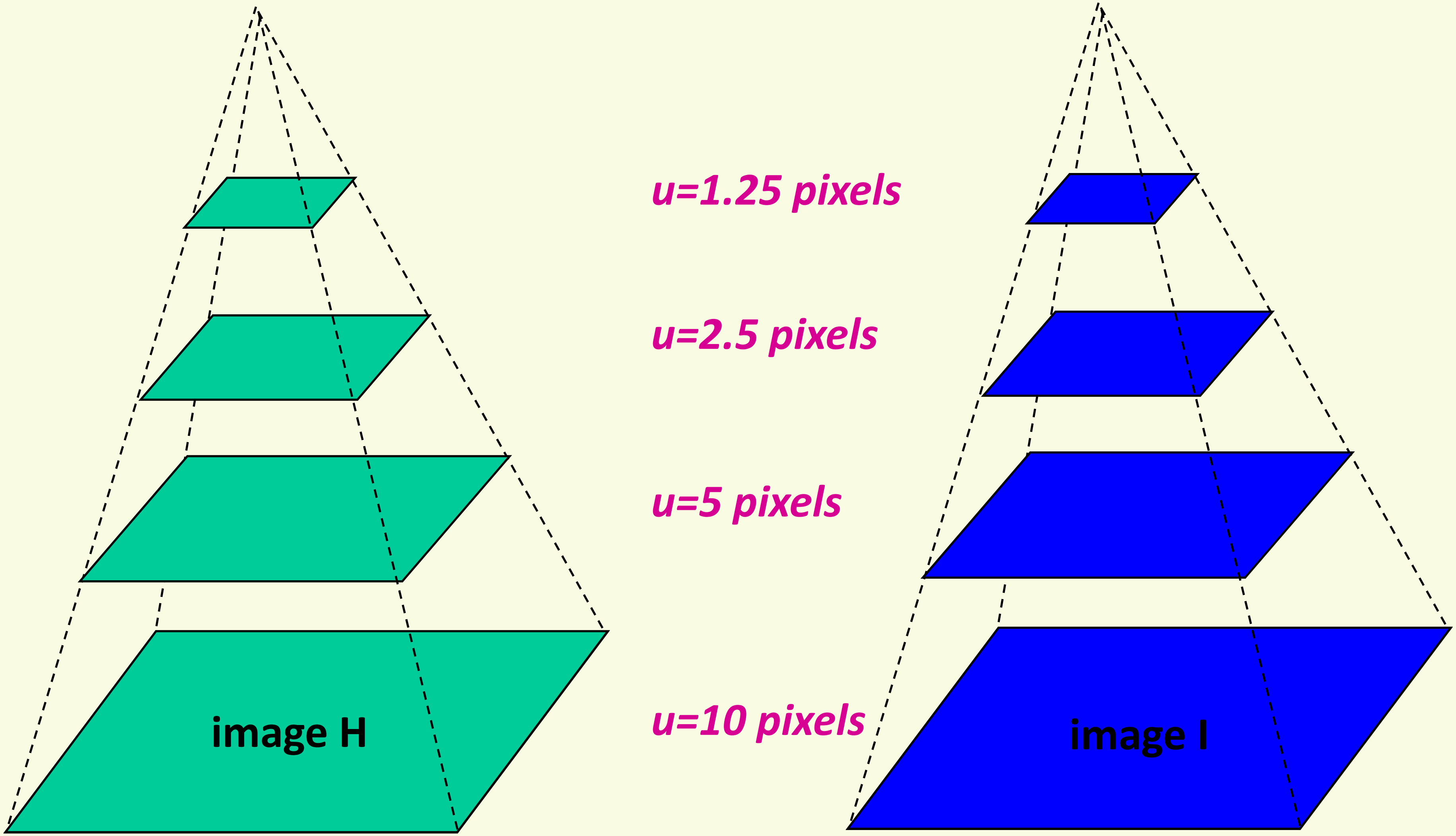
- Is this motion small enough?
  - Probably not—it's much larger than one pixel
  - How might we solve this problem?

Reduce the resolution!



motion is about 1 pixel

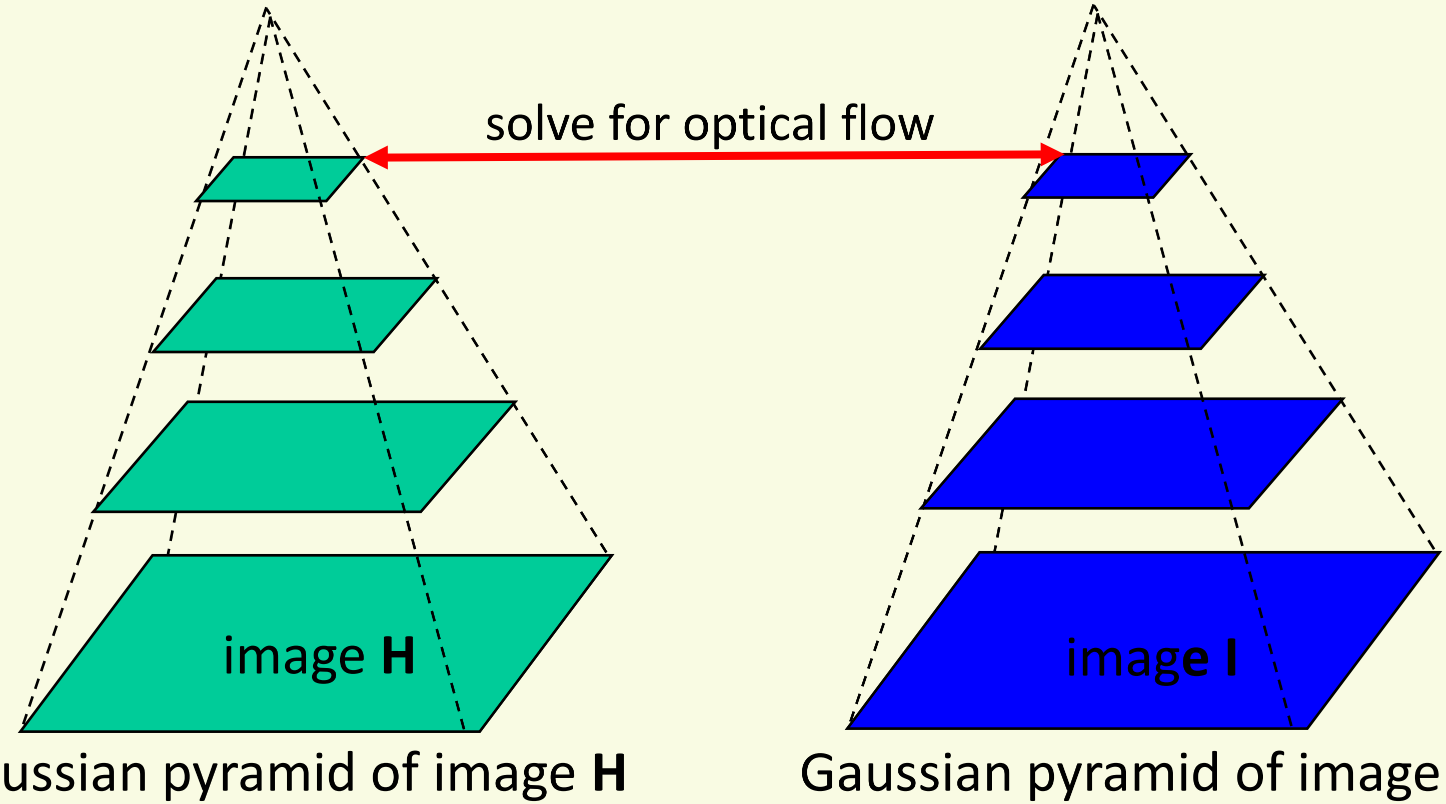
Coarse-to-fine optical flow estimation



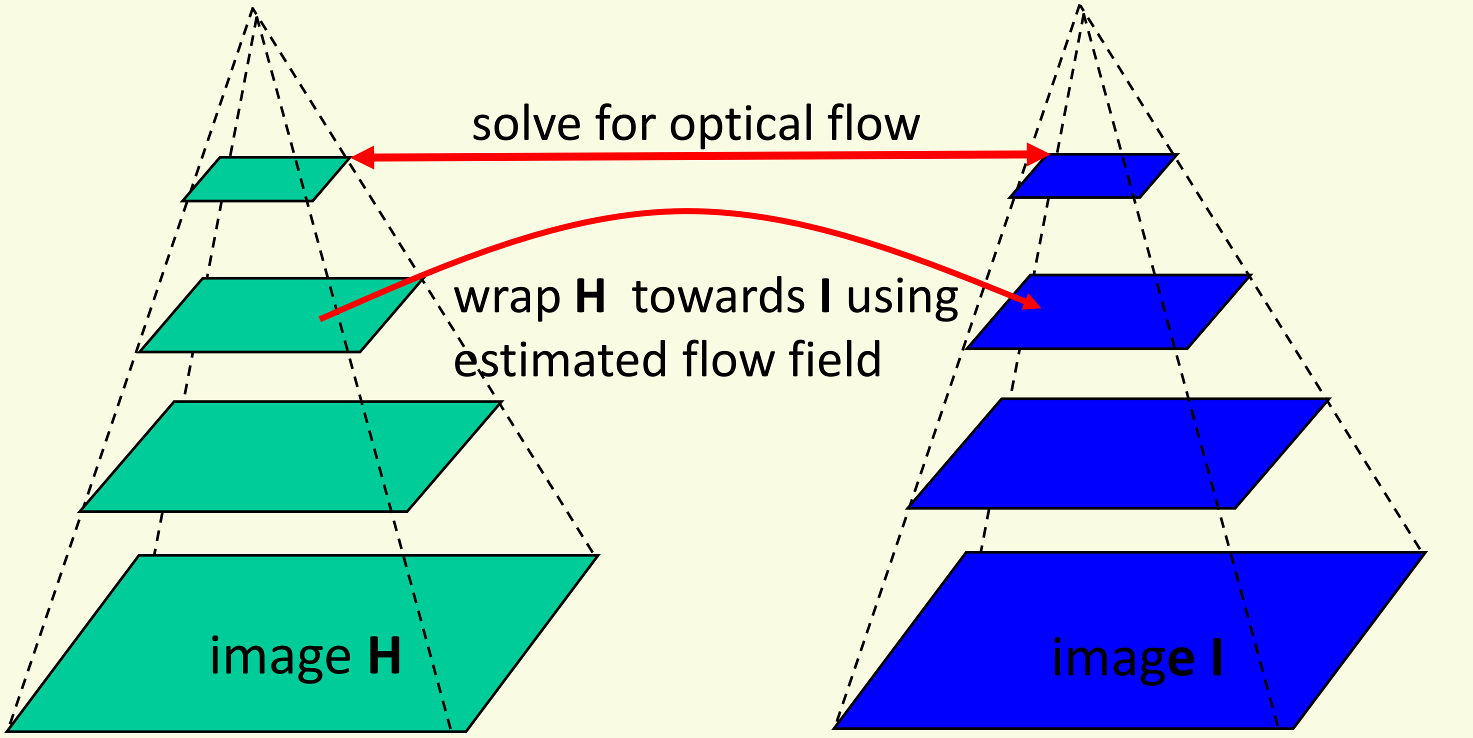
Gaussian pyramid of image H

Gaussian pyramid of image I

Iterative Lukas-Kanade Refinement


solve for optical flow
image **H**
image **I**
Gaussian pyramid of image **H**
Gaussian pyramid of image **I**

Iterative Lukas-Kanade Refinement

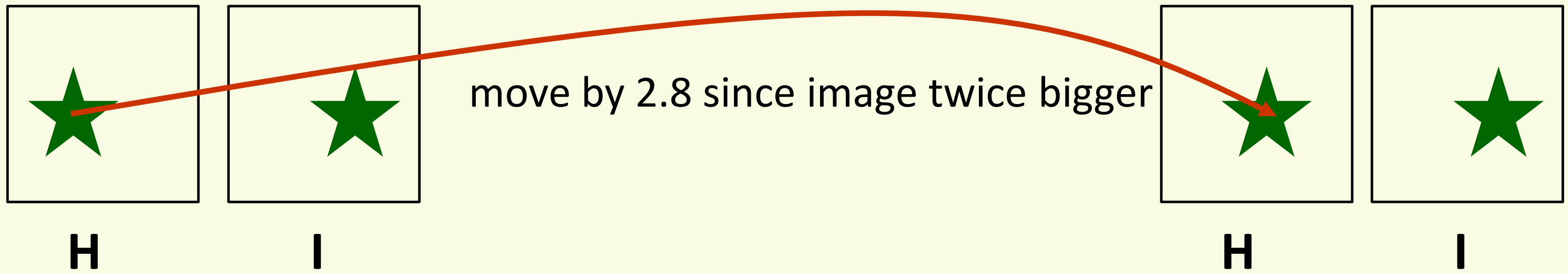


solve for optical flow

wrap **H** towards **I** using estimated flow field

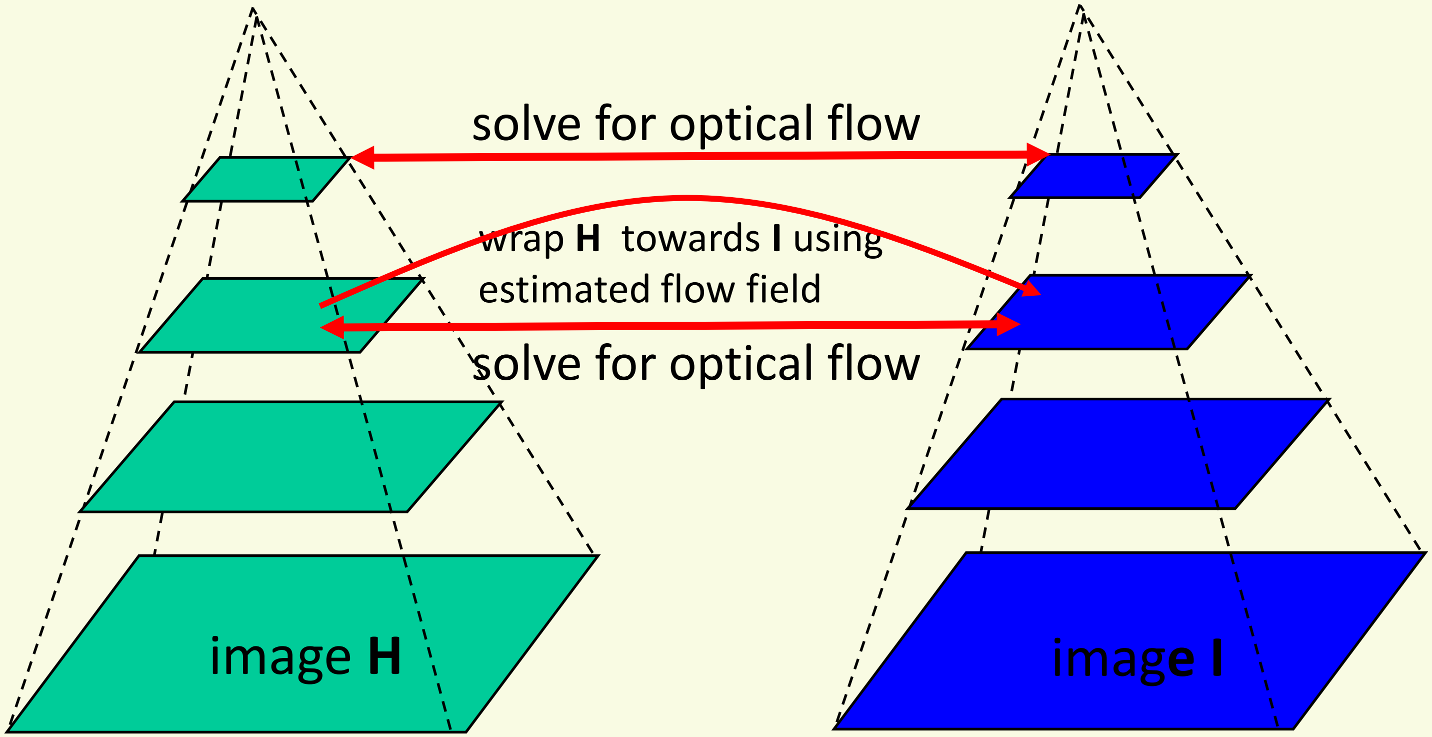
- Before wrapping, motion of 3.9 pixels
- Estimated flow is 1.4 pixels to the left

- After wrapping
- Residual motion is 1.1 pixels to the left



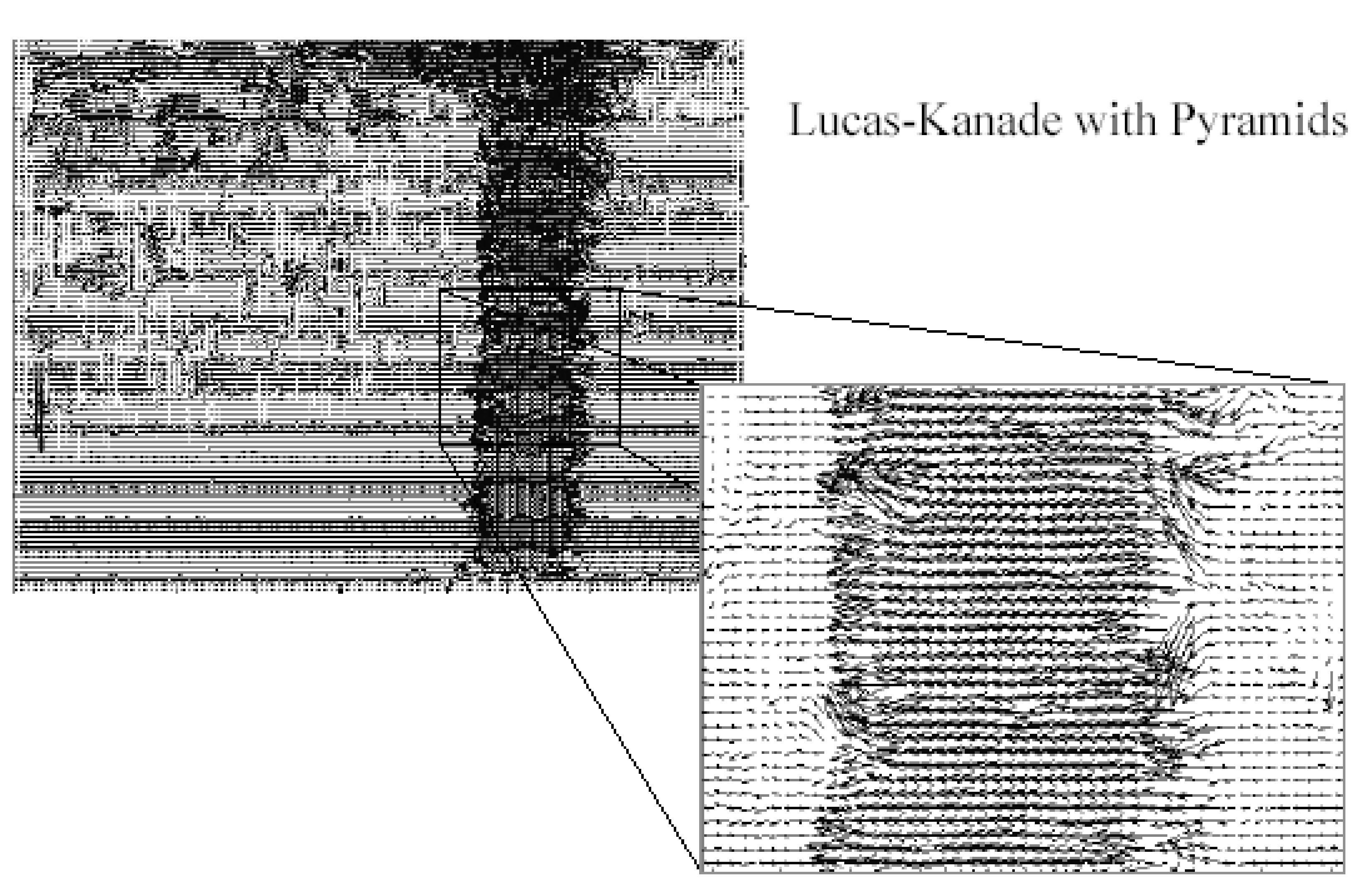
move by 2.8 since image twice bigger

Iterative Lukas-Kanade Refinement



- Continue iterations until reach the bottom of the pyramid
  - Solve for optical flow
  - Wrap **H** toward **I** using estimated optical flow

Optical Flow Results



* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Modern OF Algorithms

- A lot of development in the past 10 years
- See Middlebury Optical Flow Evaluation
  - http://vision.middlebury.edu/flow/
  - Dataset with ground truth