# CMPSCI 670: Computer Vision

## Linear filtering

University of Massachusetts, Amherst

September 22, 2014

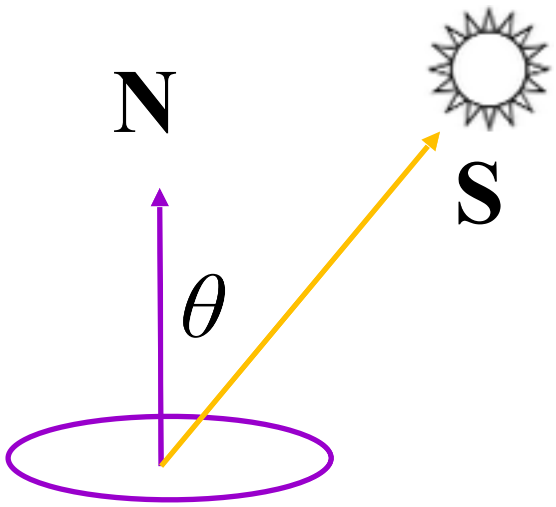
Instructor: Subhransu Maji

Slides credit: L. Lazebnik and others

# Today

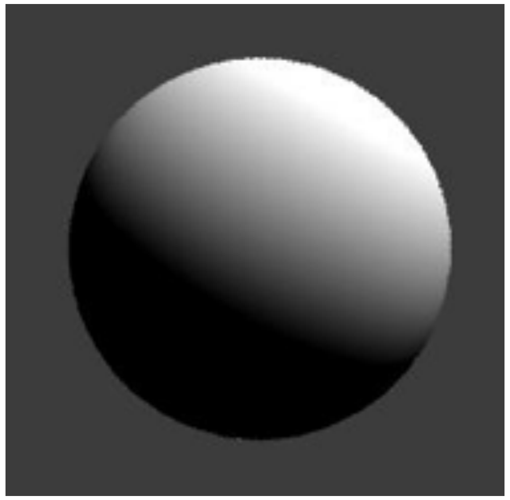
- **Administrivia:**
  - Anyone had problems with submitting homework via edlab should email their homework to me (**smaji@cs.umass.edu**)
  - Late submission policy
    - Everyone has two late days for the entire semester. Beyond that you lose 15% of the homework per day.
  - *Office hours this week:* Thursday 3:45 - 4:45, CS 274
- Today's lecture
  - Conclude photometric stereo, aka, shape from shading
  - Linear filtering

# Diffuse reflection: Lambert's law




$$B = \rho(\mathbf{N} \cdot \mathbf{S})$$

$$= \rho \left\| \mathbf{S} \right\| \cos\theta$$



*B*: radiosity (total power leaving the surface per unit area)
*ρ*: albedo (fraction of incident irradiance reflected by the surface)
*N*: unit normal
*S*: source vector (magnitude proportional to intensity of the source)

# Photometric stereo (shape from shading)

- Can we reconstruct the shape of an object based on shading cues?

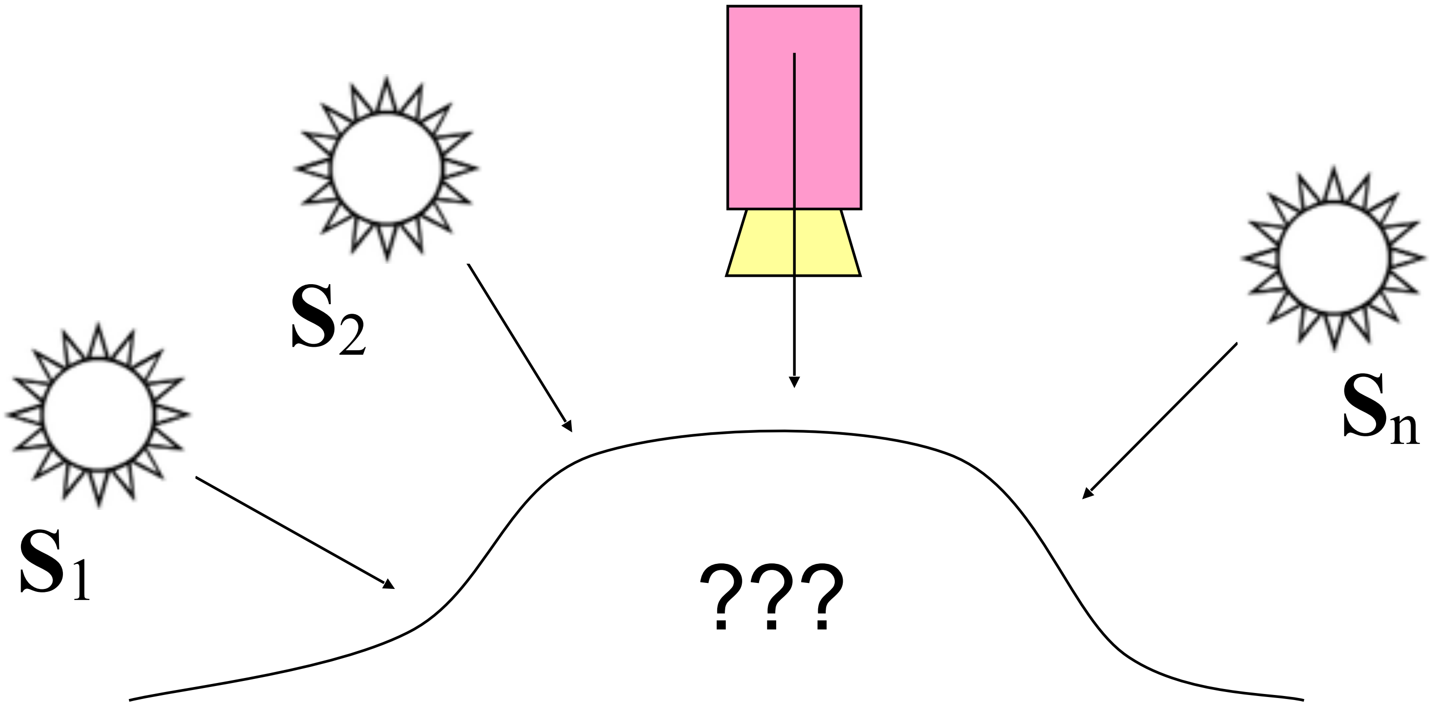
Luca della Robbia,
*Cantoria*, 1438

# Photometric stereo

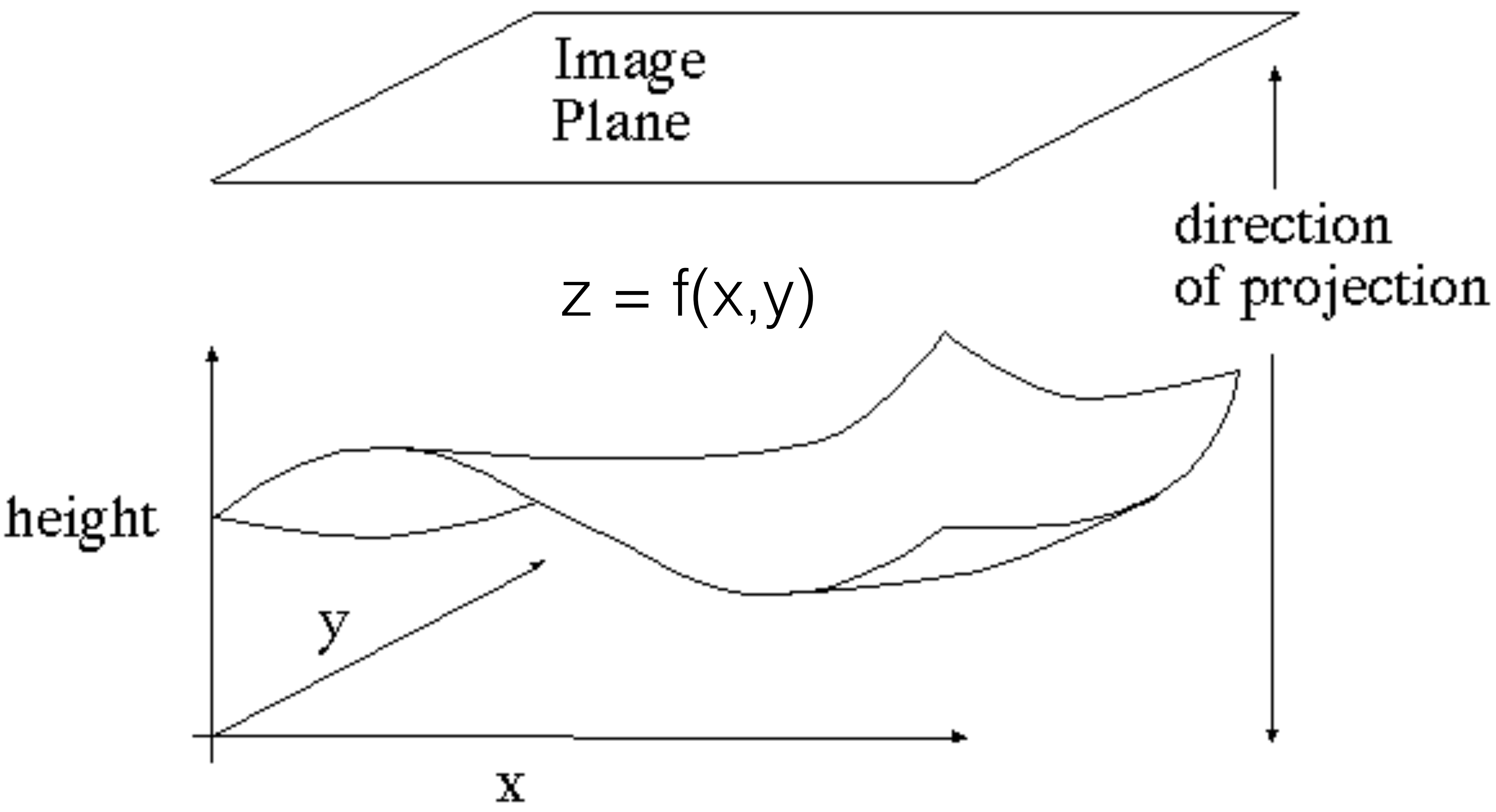
Assume:

- A Lambertian object
- A ***local shading model*** (each point on a surface receives light only from sources visible at that point)
- A set of ***known*** light source directions
- A set of pictures of an object, obtained in exactly the same camera/object configuration but using different sources
- Orthographic projection

Goal: reconstruct object shape and albedo



# Surface model: Monge patch




F&P 2nd ed., sec. 2.2.4

# Image model

- **Known:** source vectors $\mathbf{S}_j$ and pixel values $I_j(x,y)$
- **Unknown:** surface normal $\mathbf{N}(x,y)$ and albedo $\rho(x,y)$
- Assume that the response function of the camera is a linear scaling by a factor of $k$
- Lambert's law:

$$
\begin{aligned}
I_j(x,y) &= k\rho(x,y)\left(\mathbf{N}(x,y)\cdot\mathbf{S}_j\right) \\
&= \left(\rho(x,y)\mathbf{N}(x,y)\right)\cdot(k\mathbf{S}_j) \\
&= \mathbf{g}(x,y)\cdot\mathbf{V}_j
\end{aligned}
$$

F&P 2nd ed., sec. 2.2.4

# Least squares problem

- For each pixel, set up a linear system:

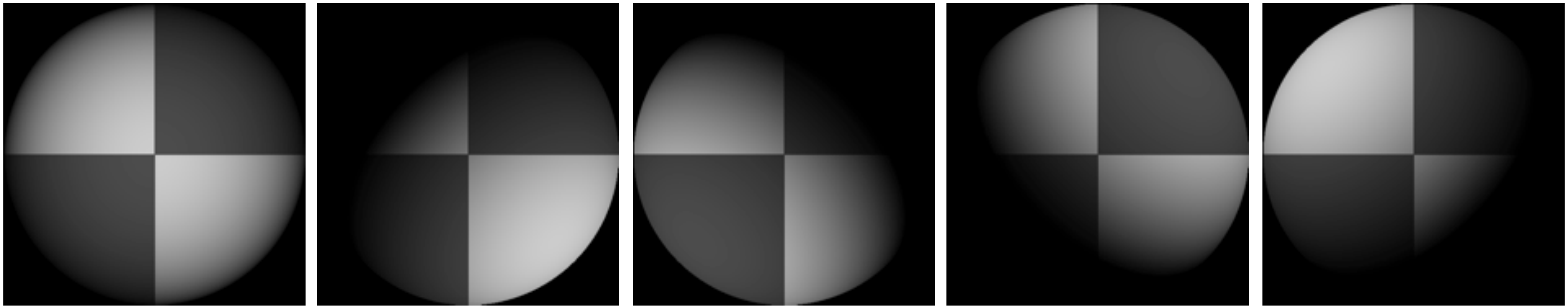
$$\begin{bmatrix} I_1(x,y) \\ I_2(x,y) \\ \vdots \\ I_n(x,y) \end{bmatrix} = \begin{bmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \\ \vdots \\ \mathbf{V}_n^T \end{bmatrix} \mathbf{g}(x,y)$$

$(n \times 1)$ known — $(n \times 3)$ known — $(3 \times 1)$ unknown

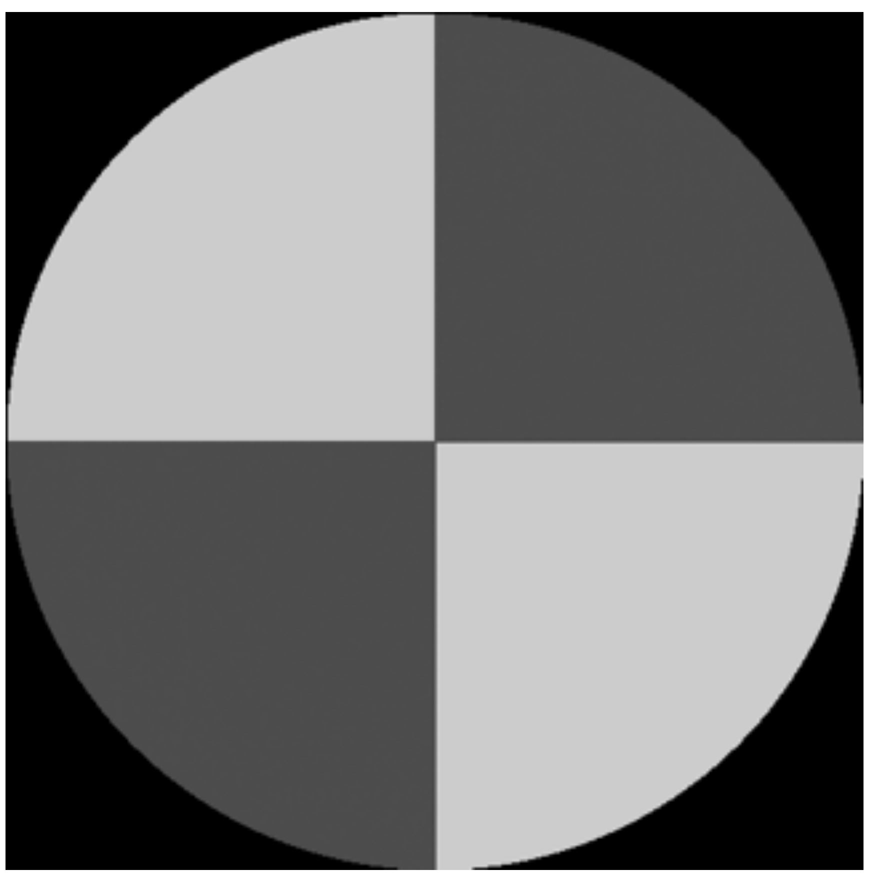
- Obtain least-squares solution for $\mathbf{g}(x,y)$ (which we defined as $\mathbf{N}(x,y)\ \rho(x,y)$)
- Since $\mathbf{N}(x,y)$ is the unit normal, $\rho(x,y)$ is given by the magnitude of $\mathbf{g}(x,y)$
- Finally, $\mathbf{N}(x,y) = \mathbf{g}(x,y)\ /\ \rho(x,y)$

F&P 2nd ed., sec. 2.2.4

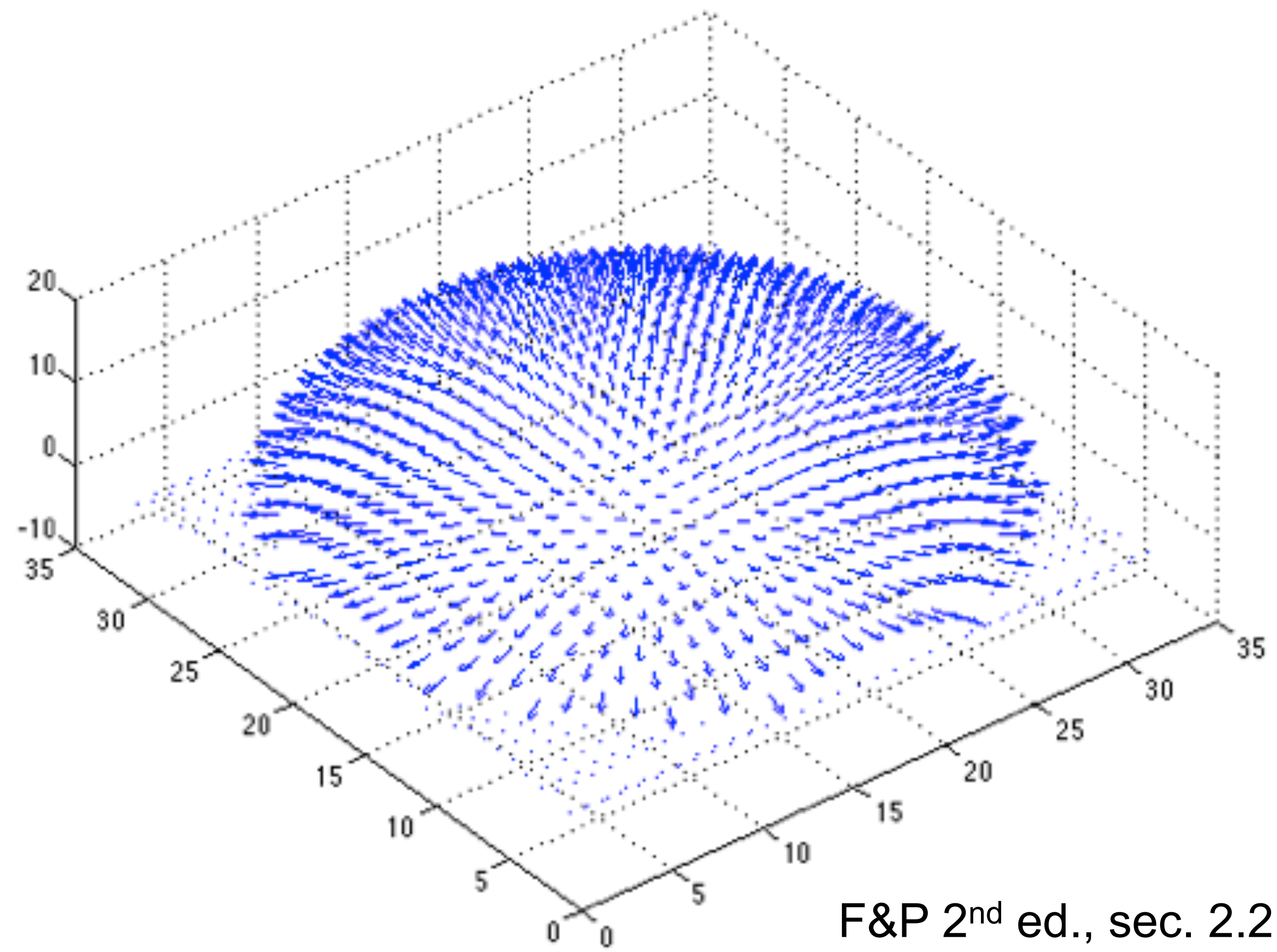
# Example



Recovered albedo



Recovered normal field




F&P 2$^{nd}$ ed., sec. 2.2.4

# Recovering a surface from normals

Recall the surface is written as

$$(x, y, f(x, y))$$

This means the normal has the form:

$$\mathbf{N}(x, y) = \frac{1}{\sqrt{f_x^2 + f_y^2 + 1}} \begin{pmatrix} f_x \\ f_y \\ 1 \end{pmatrix}$$

If we write the estimated vector *g* as

$$\mathbf{g}(x, y) = \begin{pmatrix} g_1(x, y) \\ g_2(x, y) \\ g_3(x, y) \end{pmatrix}$$

Then we obtain values for the partial derivatives of the surface:

$$f_x(x, y) = g_1(x, y) / g_3(x, y)$$
$$f_y(x, y) = g_2(x, y) / g_3(x, y)$$

# Recovering a surface from normals

*Integrability*: for the surface $f$ to exist, the mixed second partial derivatives must be equal:

$$\frac{\partial}{\partial y}(g_1(x,y)/g_3(x,y)) =$$

$$\frac{\partial}{\partial x}(g_2(x,y)/g_3(x,y))$$

(in practice, they should at least be similar)

We can now recover the surface height at any point by integration along some path, e.g.

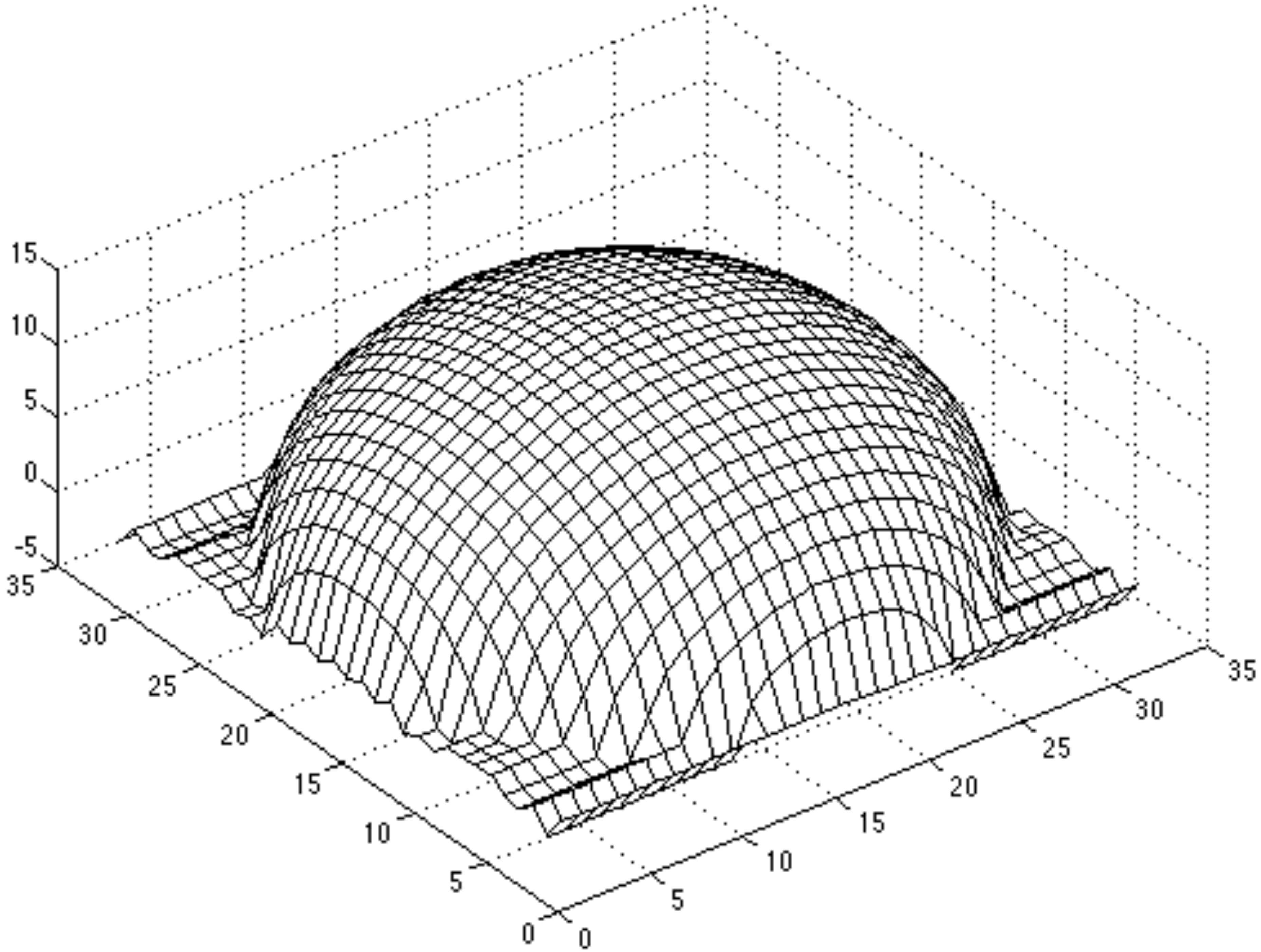
$$f(x,y) = \int_0^x f_x(s,y)ds +$$

$$\int_0^y f_y(x,t)dt + C$$

(for robustness, should take integrals over many different paths and average the results)

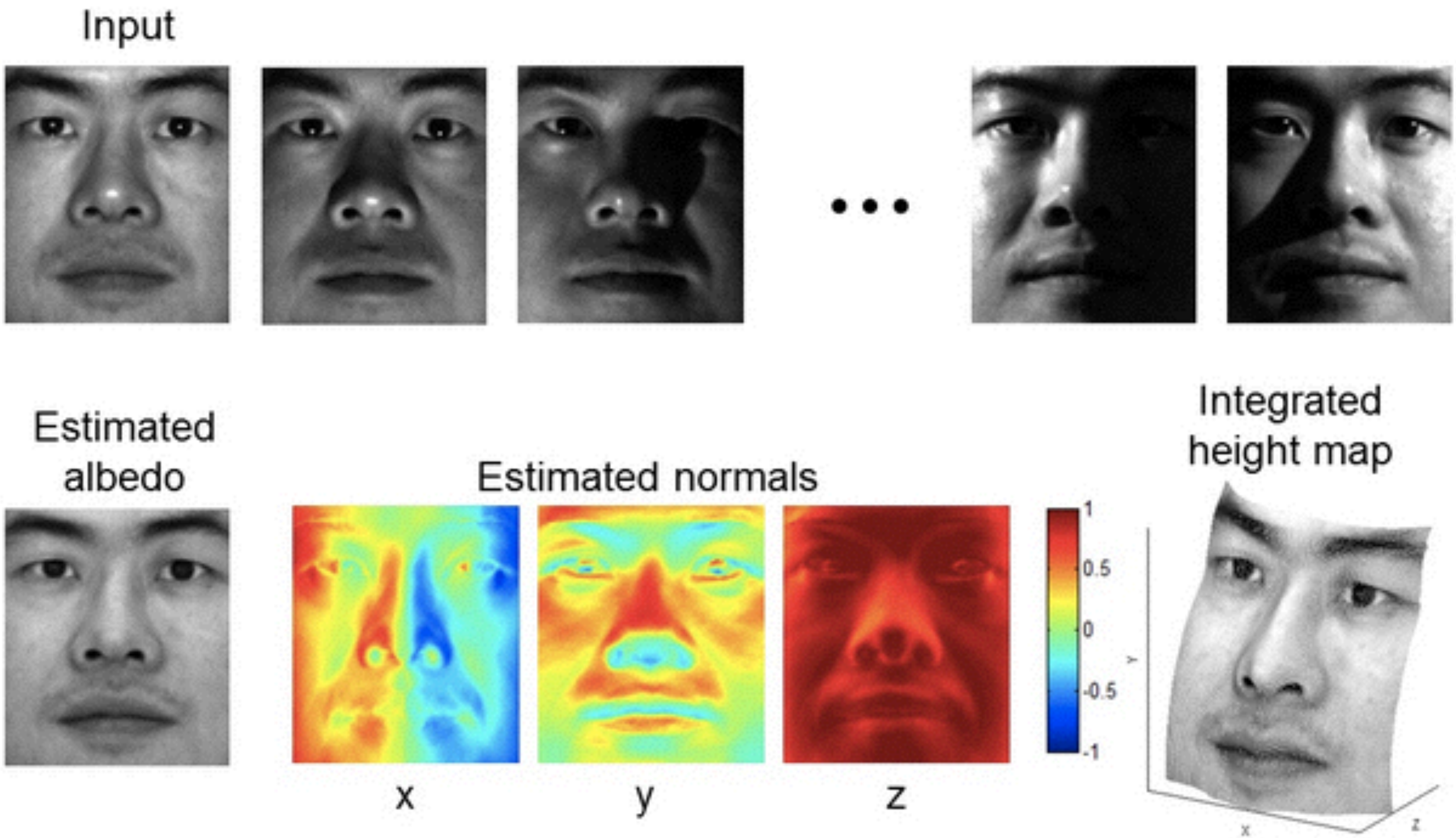
F&P 2nd ed., sec. 2.2.4

# Surface recovered by integration



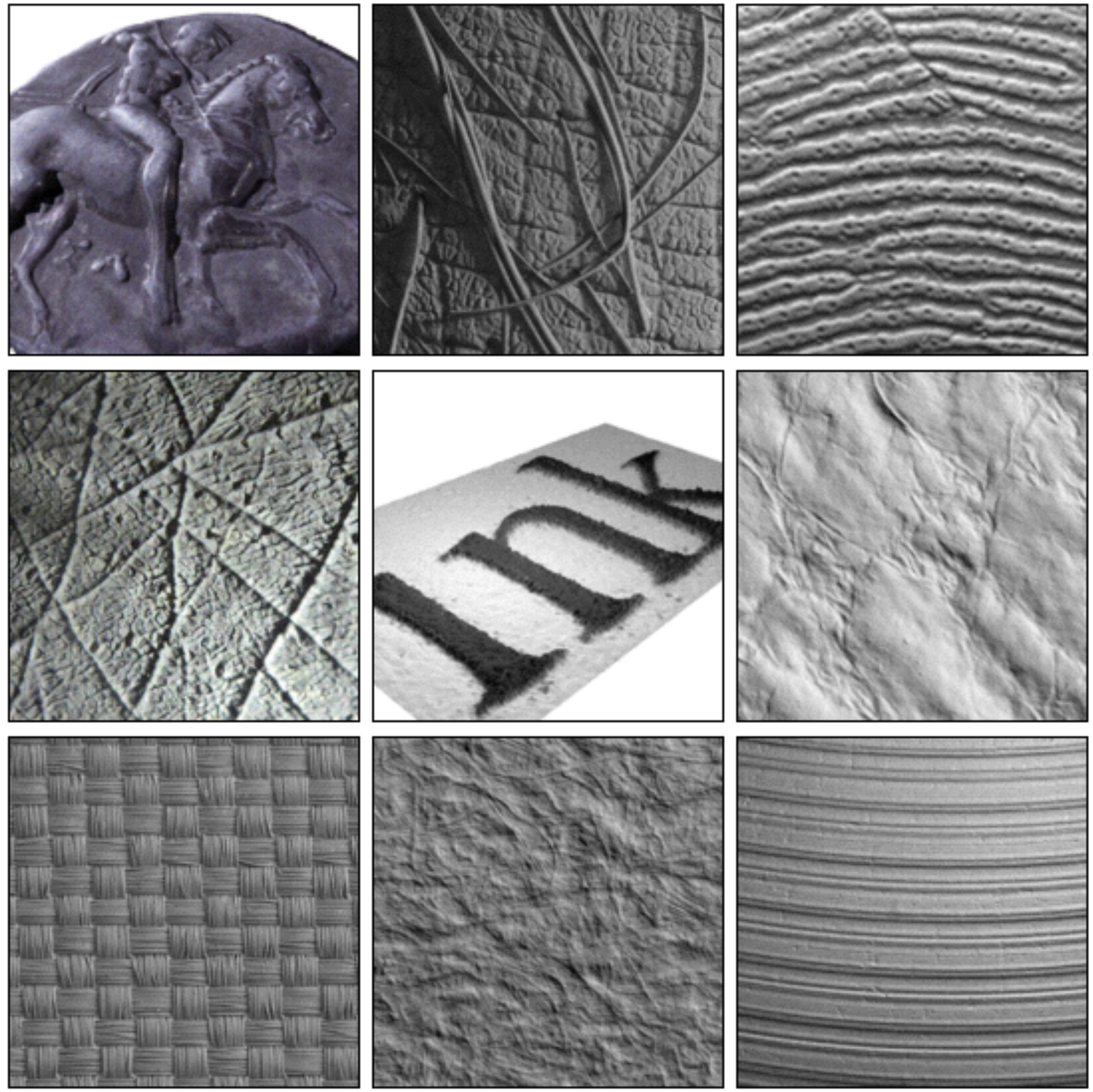

F&P 2nd ed., sec. 2.2.4

# Homework 2: Photometric stereo



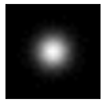
# Application

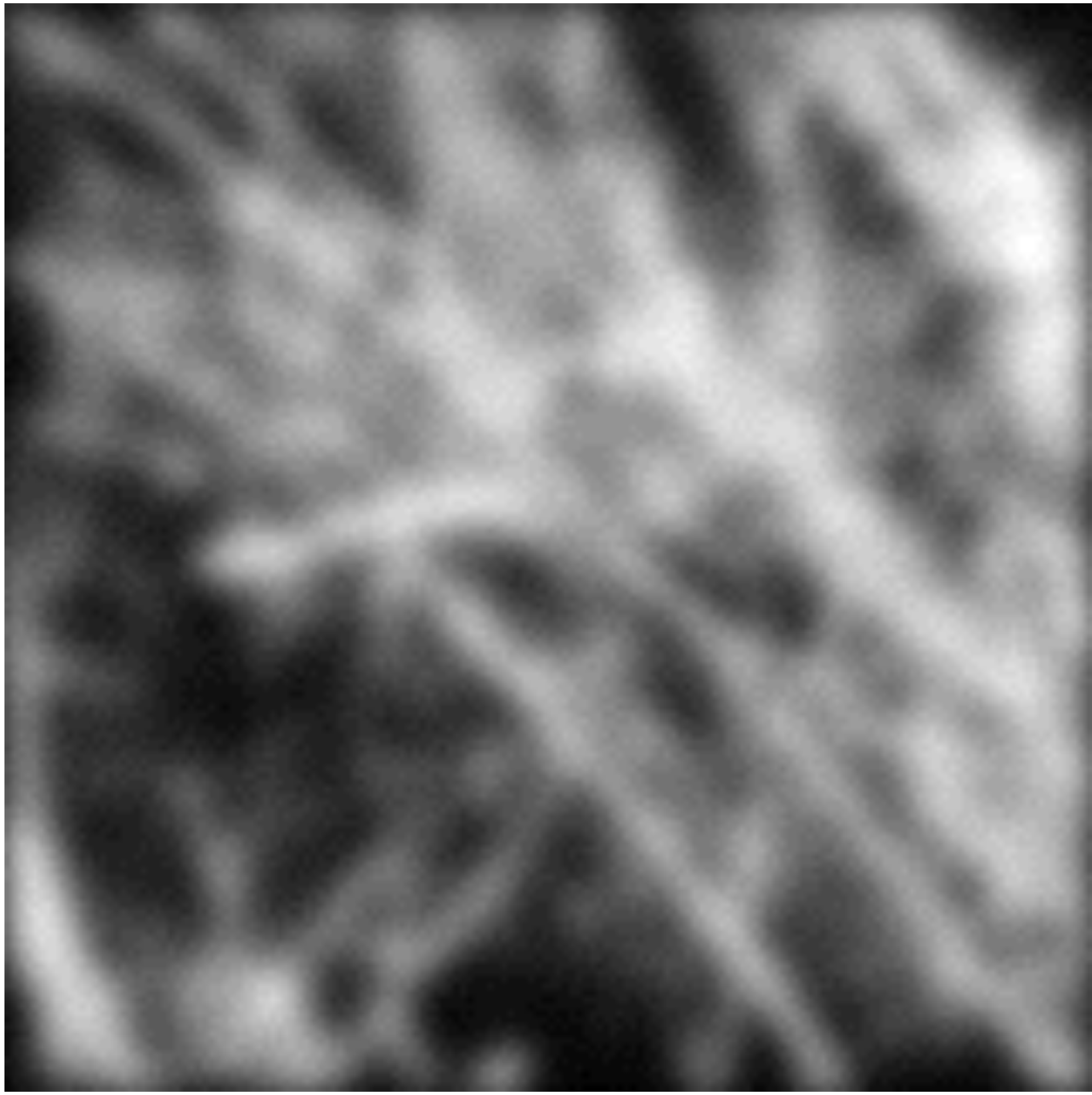
GELSIGHT




https://www.youtube.com/watch?v=S7gXih4XS7A

# Linear filtering





# Motivation: Image de-noising

- How can we reduce noise in a photograph?

# Moving average

- Let’s replace each pixel with a ***weighted*** average of its neighborhood
- The weights are called the ***filter kernel***
- What are the weights for the average of a 3x3 neighborhood?

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

“box filter”

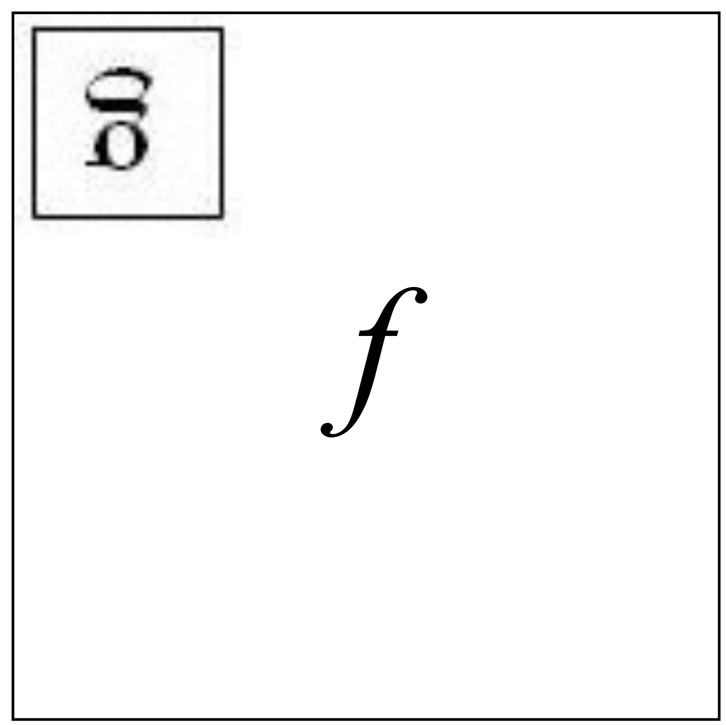
Source: D. Lowe 

# Defining convolution

- Let *f* be the image and *g* be the kernel. The output of convolving *f* with *g* is denoted *f* * *g*.

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

Convention:
kernel is “flipped”




- MATLAB functions: conv2, filter2, imfilter

Source: F. Durand 

# Key properties

- **Linearity:** filter($f_1$ + $f_2$) = filter($f_1$) + filter($f_2$)
- **Shift invariance:** same behavior regardless of pixel location: filter(shift($f$)) = shift(filter($f$))
- Theoretical result: any linear shift-invariant operator can be represented as a convolution

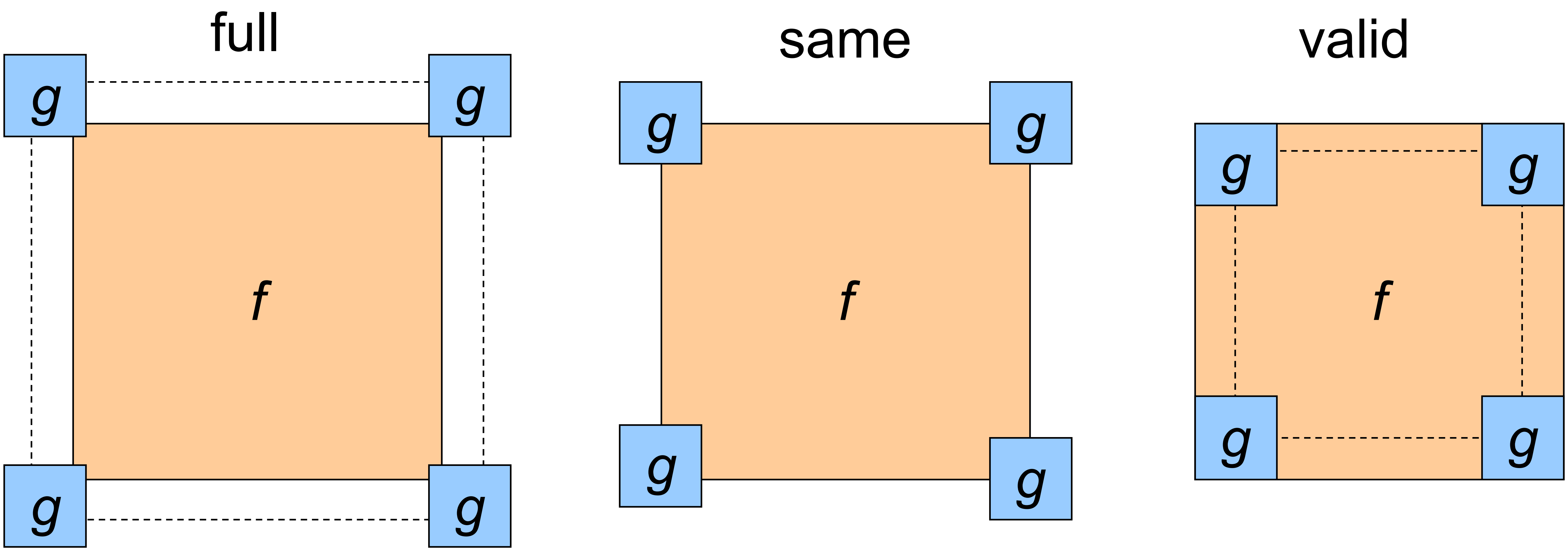
# Properties in more detail

- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k (a * b)$
- Identity: unit impulse $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$, $a * e = a$

# Annoying details

What is the size of the output?

- MATLAB: filter2(g, f, ***shape***)
  - *shape* = 'full': output size is sum of sizes of f and g
  - *shape* = 'same': output size is same as f
  - *shape* = 'valid': output size is difference of sizes of f and g



# Annoying details

## What about near the edge?

- the filter window falls off the edge of the image
- need to extrapolate
- methods:
  - clip filter (black)
  - wrap around
  - copy edge
  - reflect across edge

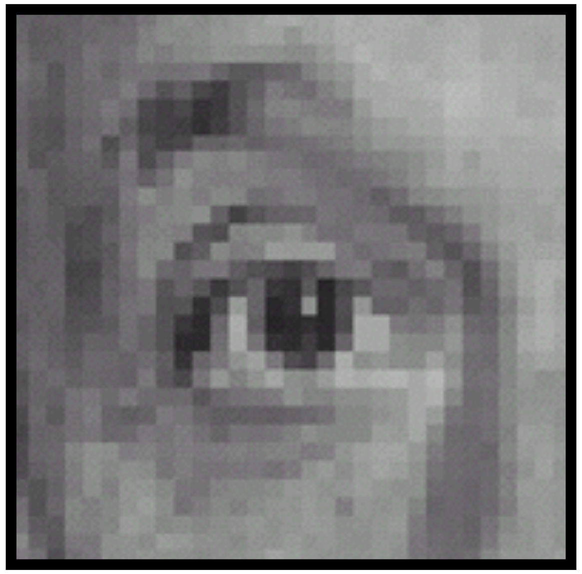
# Annoying details

## What about near the edge?

- the filter window falls off the edge of the image
- need to extrapolate
- methods (MATLAB):
  - clip filter (black): imfilter(f, g, 0)
  - wrap around: imfilter(f, g, 'circular')
  - copy edge: imfilter(f, g, 'replicate')
  - reflect across edge: imfilter(f, g, 'symmetric')

Source: S. Marschner

# Practice with linear filters

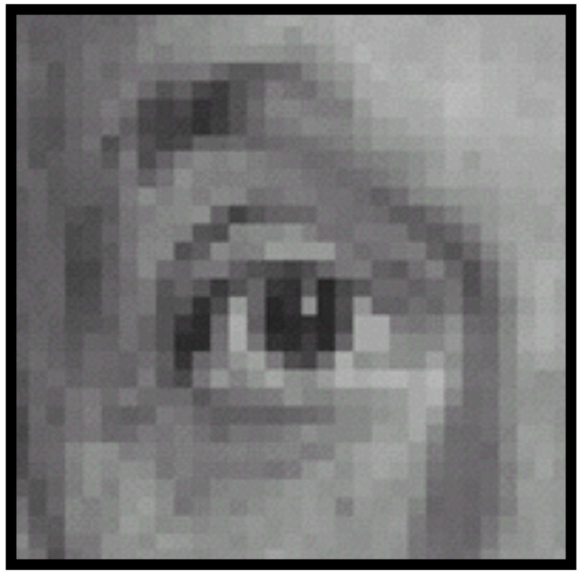


Original

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

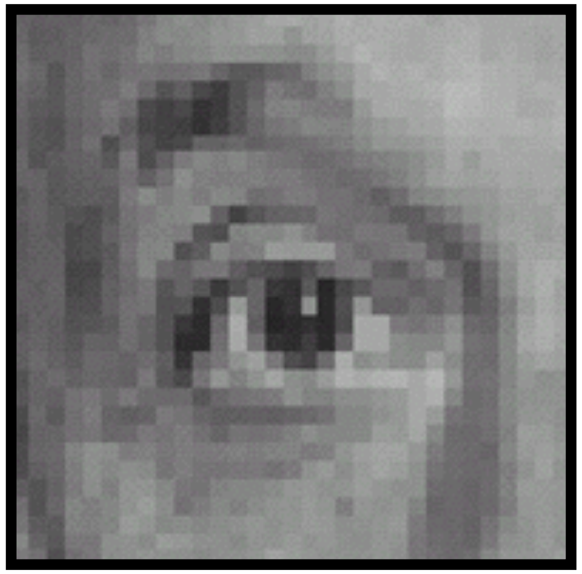
?

# Practice with linear filters



Original

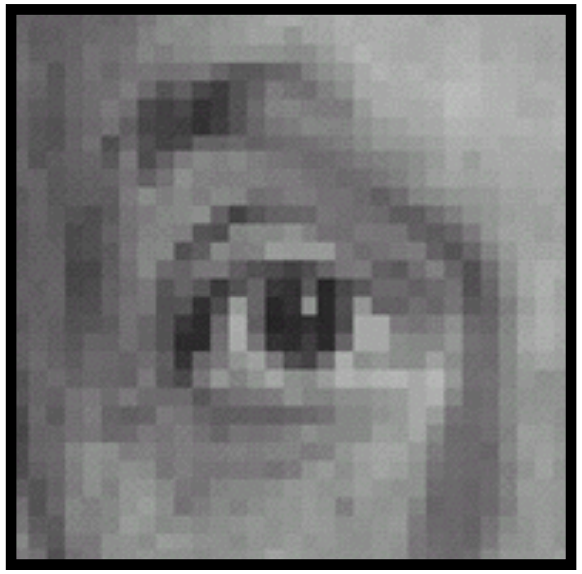
| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |



Filtered
(no change)

Source: D. Lowe

# Practice with linear filters



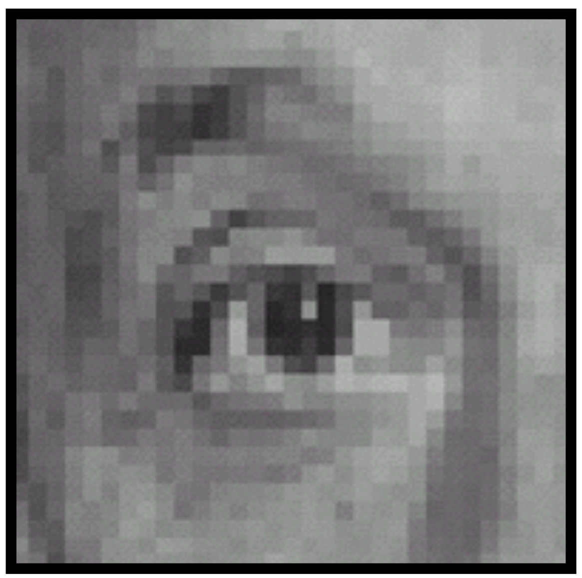
Original

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 0 |

?

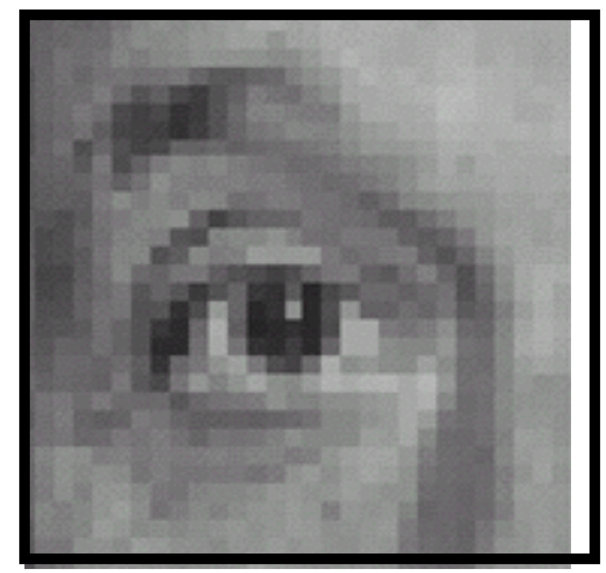
Source: D. Lowe

# Practice with linear filters



Original

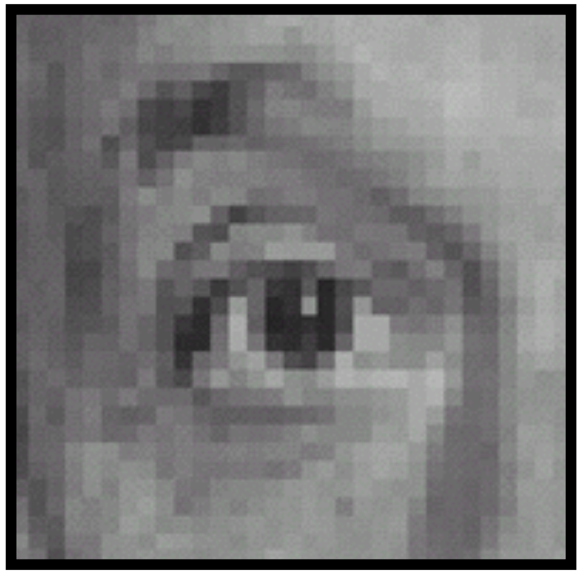
| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 0 | 0 |



Shifted *left*
By 1 pixel

Source: D. Lowe

# Practice with linear filters

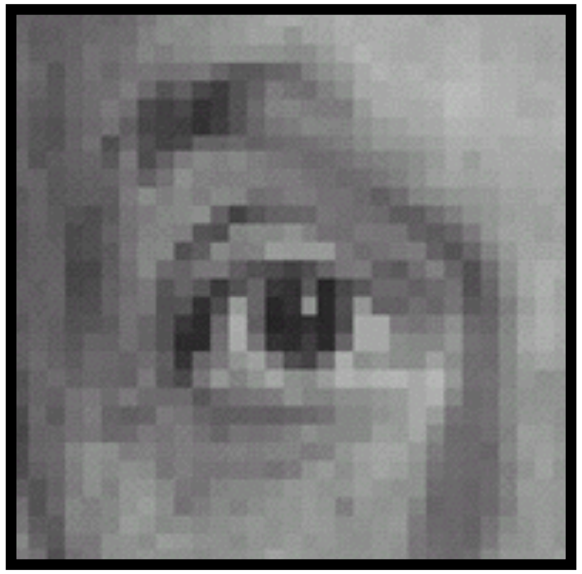


Original

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

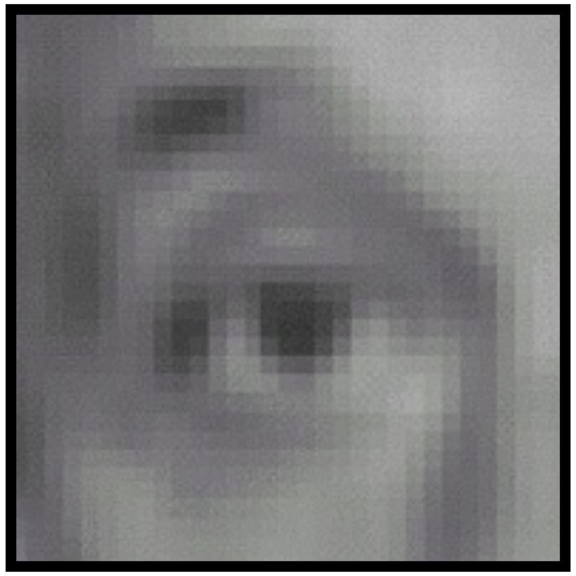
?

# Practice with linear filters



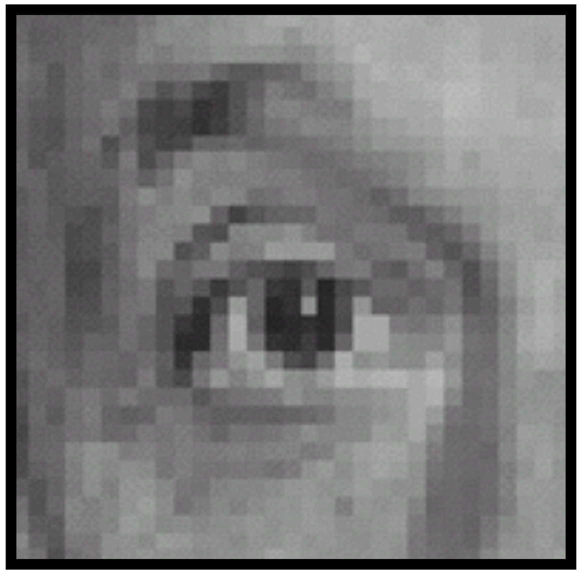
Original

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



Blur (with a box filter)

# Practice with linear filters



Original

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 0 |

-

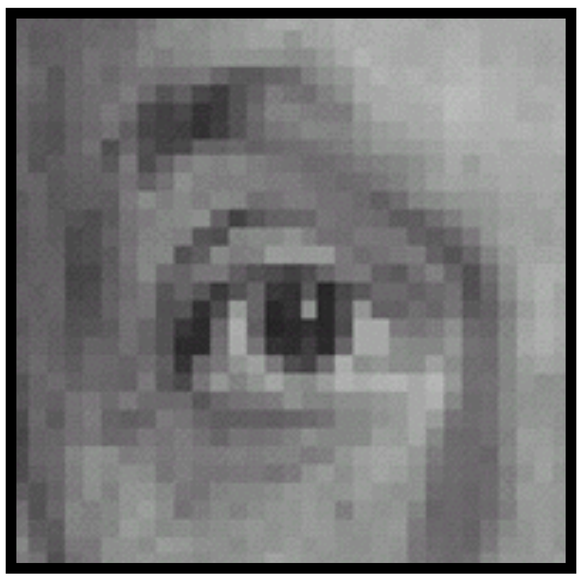
$\frac{1}{9}$

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

?

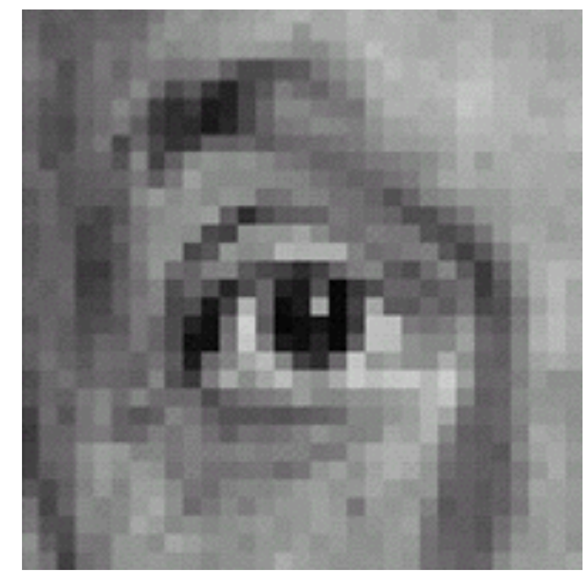
(Note that filter sums to 1)

# Practice with linear filters



Original

$$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

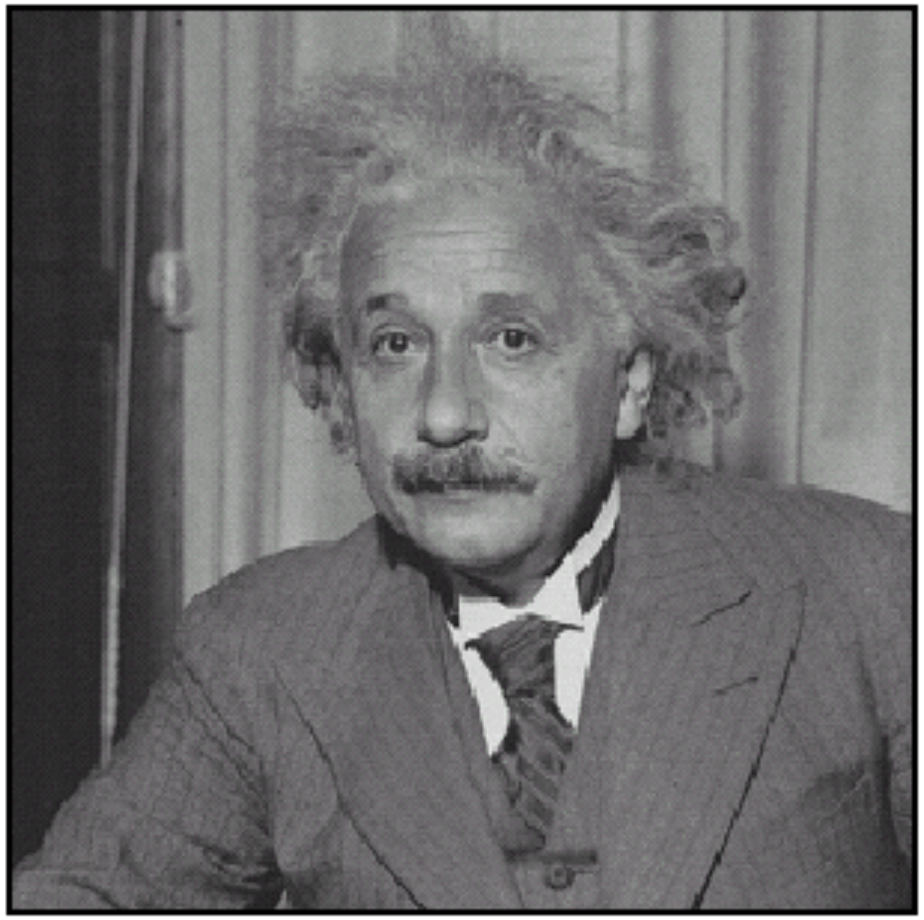


**Sharpening filter**

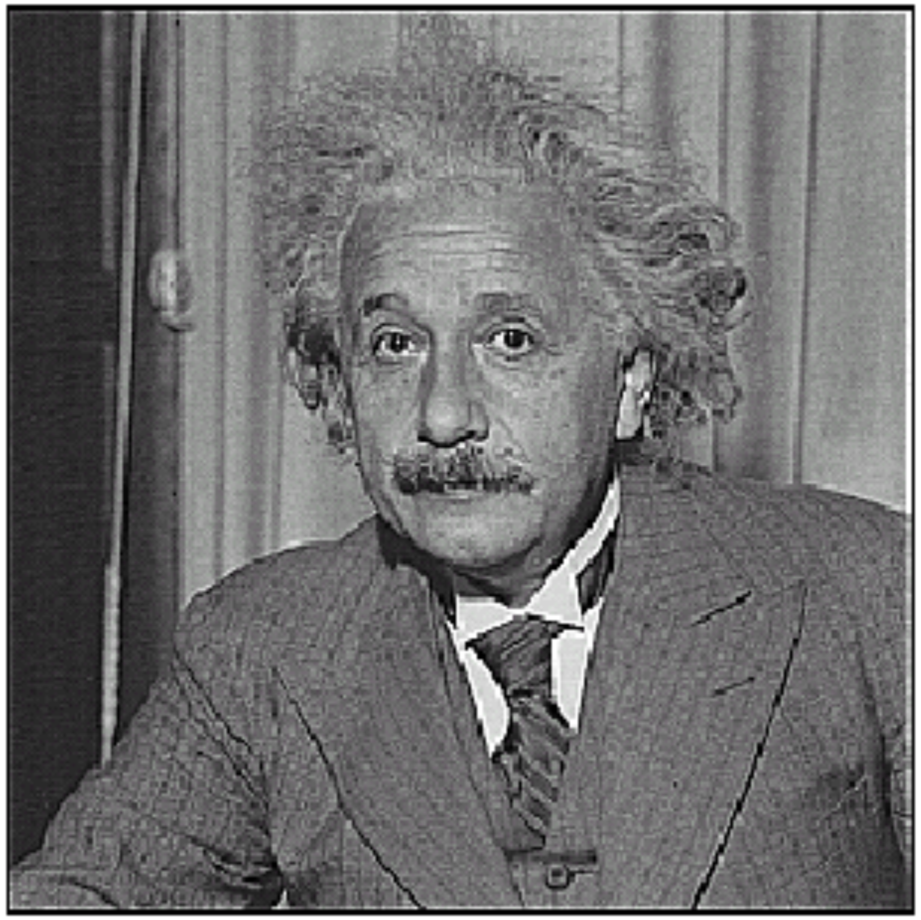
- Accentuates differences with local average

Source: D. Lowe

# Sharpening



before



after

Source: D. Lowe

# Sharpening

What does blurring take away?


−


=

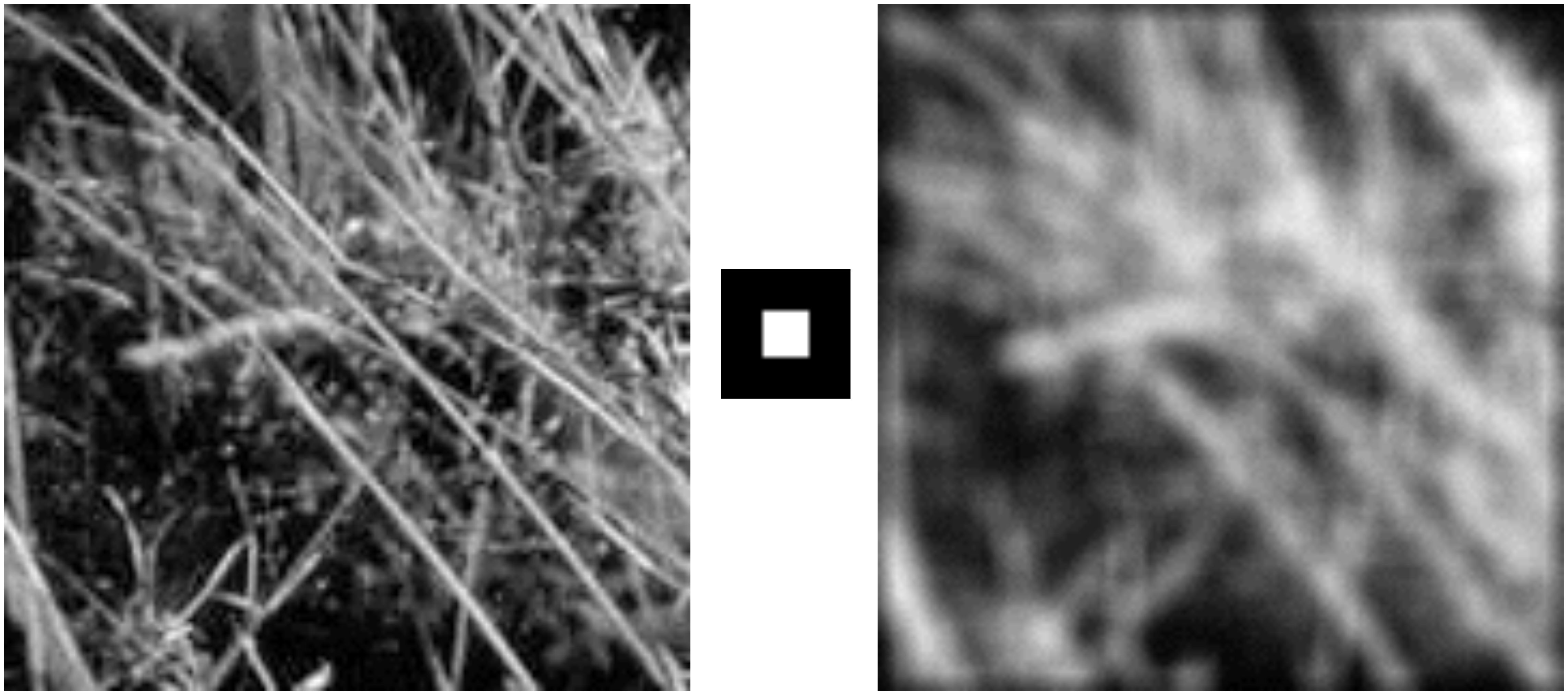

Let’s add it back:


+


=

# Smoothing with box filter revisited

- What's wrong with this picture?
- What's the solution?



Source: D. Forsyth 

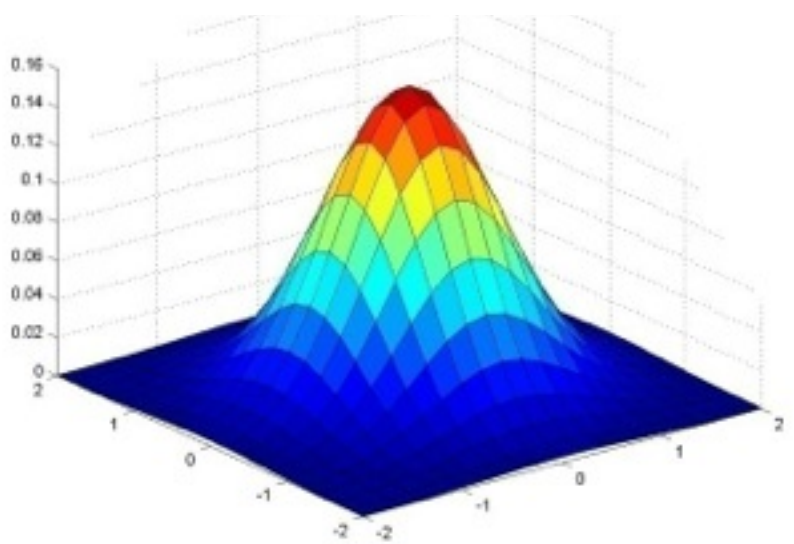
# Smoothing with box filter revisited

- What's wrong with this picture?
- What's the solution?
  - To eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to the center

"fuzzy blob"

# Gaussian Kernel

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$



| | | | | |
|---|---|---|---|---|
| 0.003 | 0.013 | 0.022 | 0.013 | 0.003 |
| 0.013 | 0.059 | 0.097 | 0.059 | 0.013 |
| 0.022 | 0.097 | 0.159 | 0.097 | 0.022 |
| 0.013 | 0.059 | 0.097 | 0.059 | 0.013 |
| 0.003 | 0.013 | 0.022 | 0.013 | 0.003 |

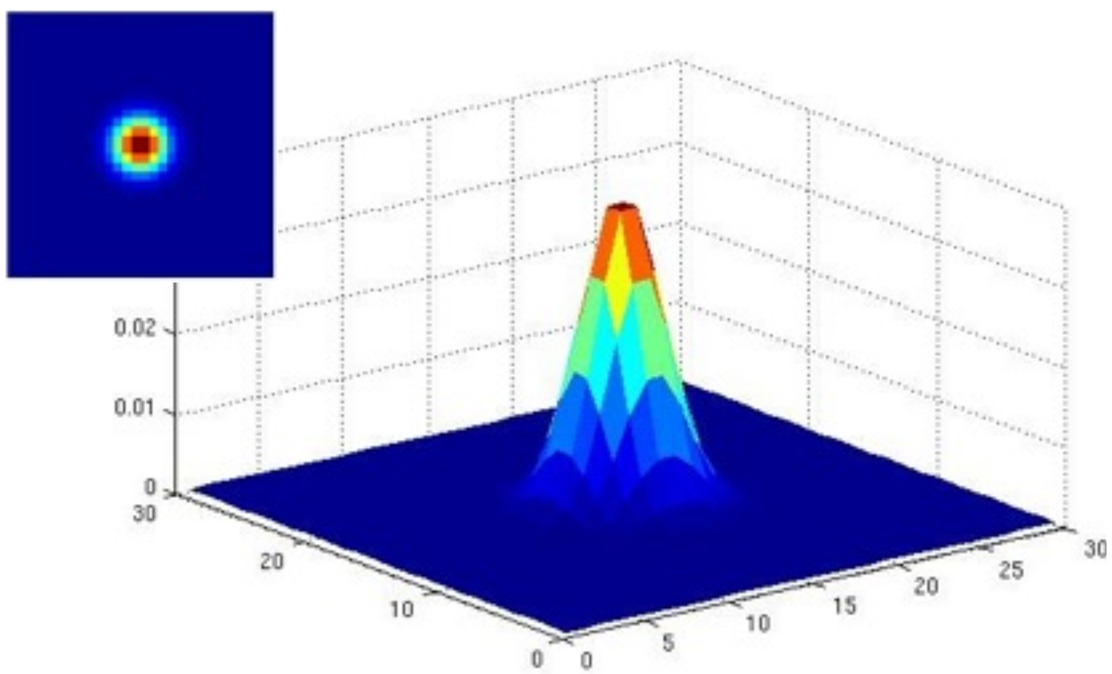
5 x 5, $\sigma = 1$

- Constant factor at front makes volume sum to 1 (can be ignored when computing the filter values, as we should renormalize weights to sum to 1 in any case)

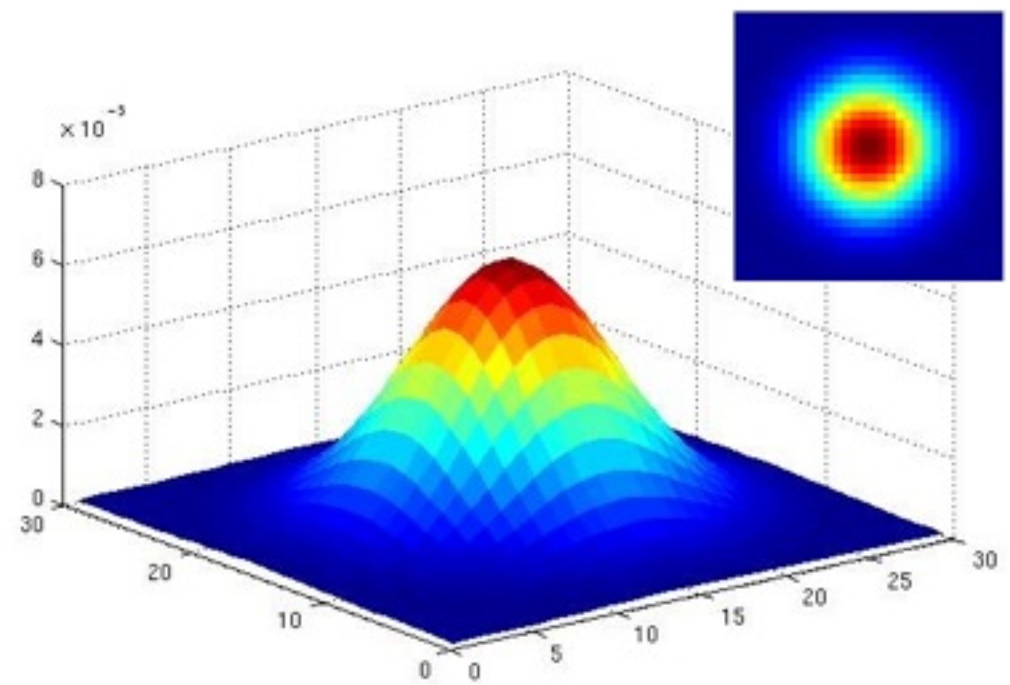
Source: C. Rasmussen

# Gaussian Kernel

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$




σ = 2 with 30 x 30
kernel

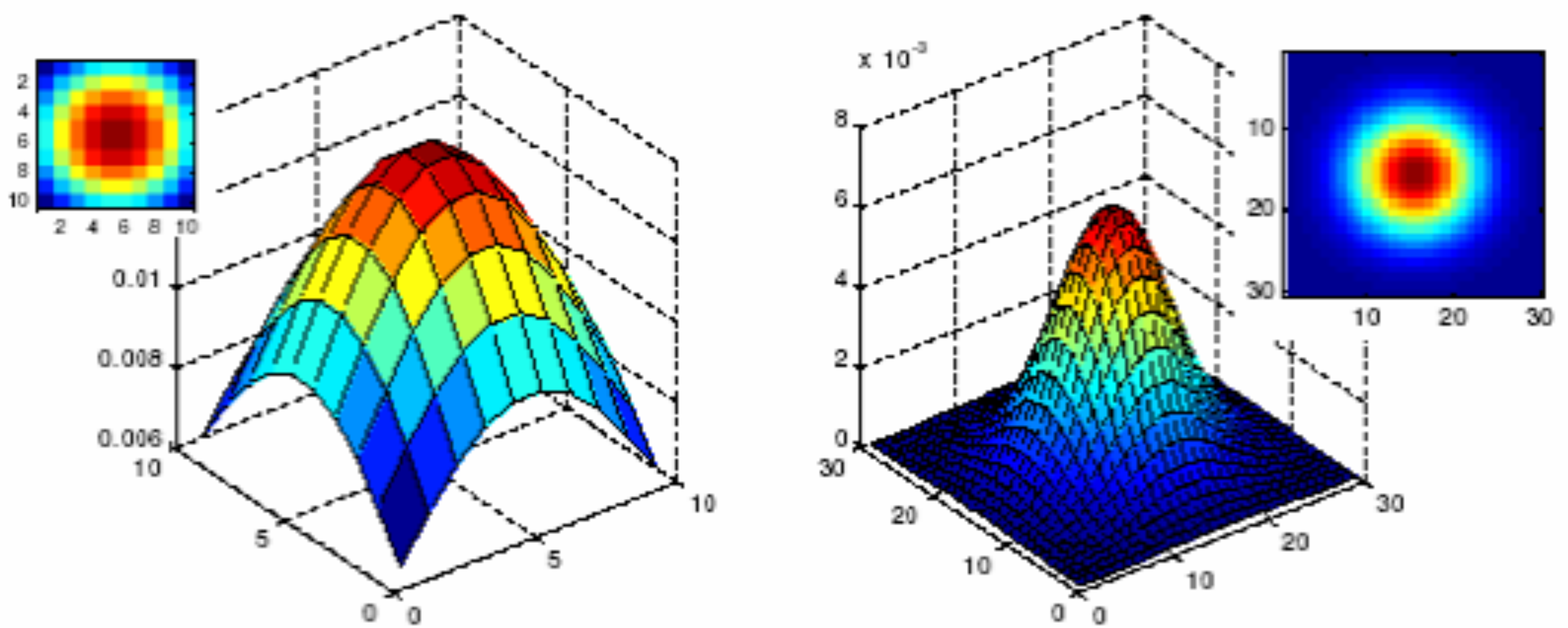



σ = 5 with 30 x 30
kernel

- Standard deviation σ: determines extent of smoothing

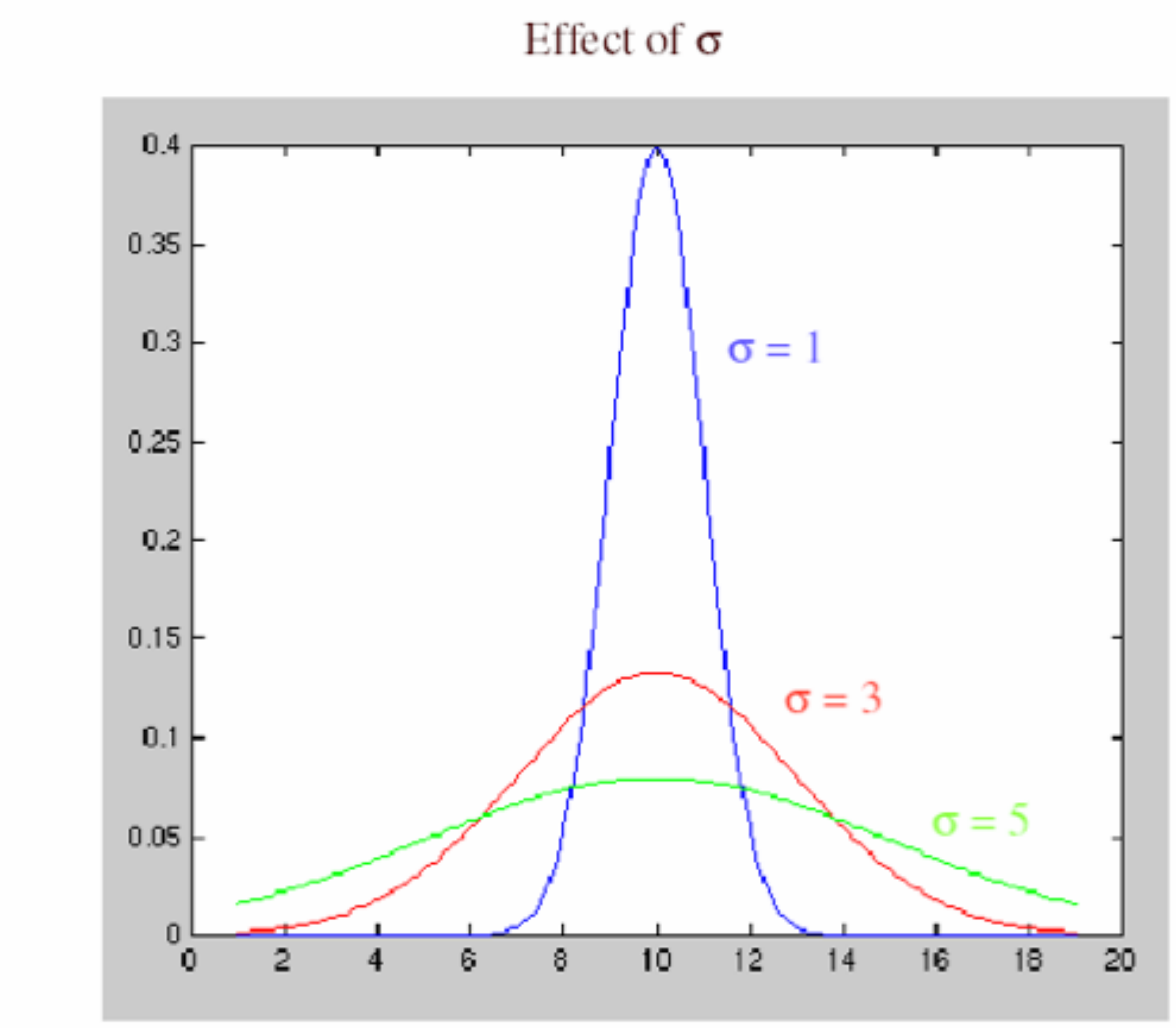
# Choosing kernel width

- The Gaussian function has infinite support, but discrete filters use finite kernels

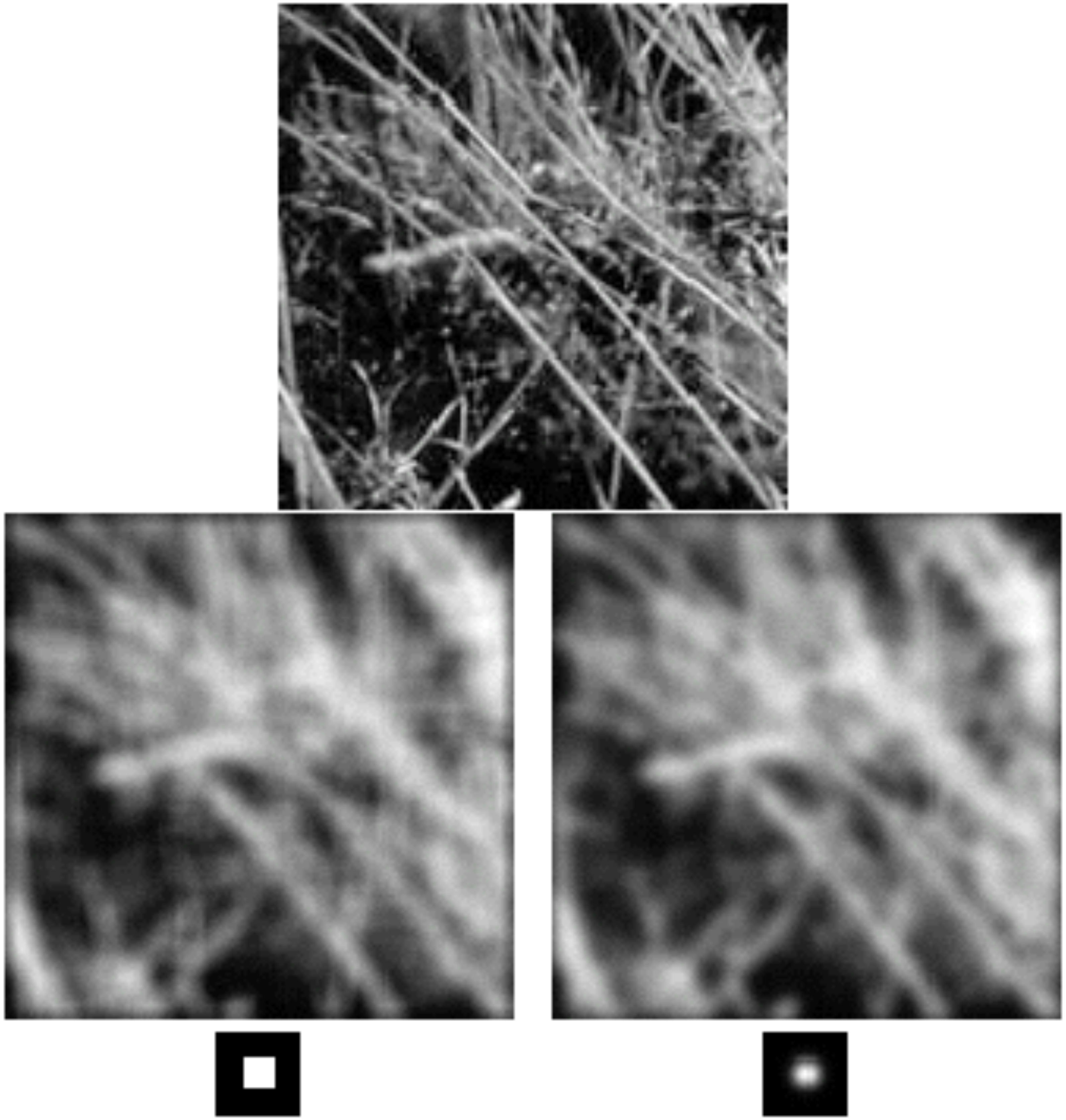


# Choosing kernel width

- Rule of thumb: set filter half-width to about $3\sigma$



# Gaussian vs. box filtering



# Gaussian filters

- Remove high-frequency components from the image (***low-pass filter***)
- Convolution with self is another Gaussian
  - So can smooth with small-$\sigma$ kernel, repeat, and get same result as larger-$\sigma$ kernel would have
  - Convolving two times with Gaussian kernel with std. dev. $\sigma$ is same as convolving once with kernel with std. dev. $\sigma\sqrt{2}$
- *Separable* kernel
  - Factors into product of two 1D Gaussians
  - Discrete example:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Source: K. Grauman

# Separability of the Gaussian filter

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}}\right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}}\right)$$

The 2D Gaussian can be expressed as the product of two functions, one a function of $x$ and the other a function of $y$

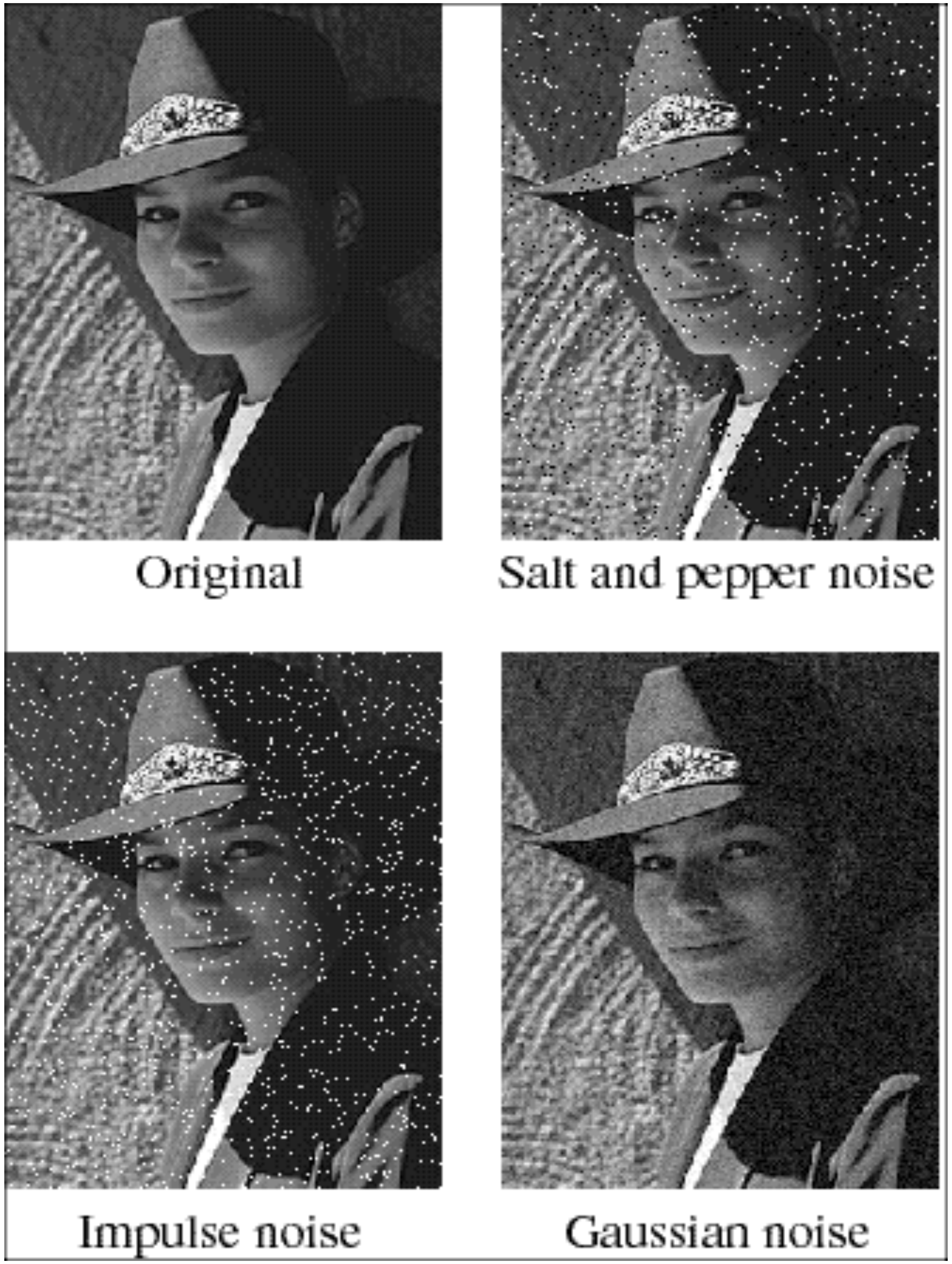
In this case, the two functions are the (identical) 1D Gaussian

Source: D. Lowe

# Why is separability useful?

- Separability means that a 2D convolution can be reduced to two 1D convolutions (one among rows and one among columns)
- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?
  - $O(n^2 m^2)$
- What if the kernel is separable?
  - $O(n^2 m)$

# Noise

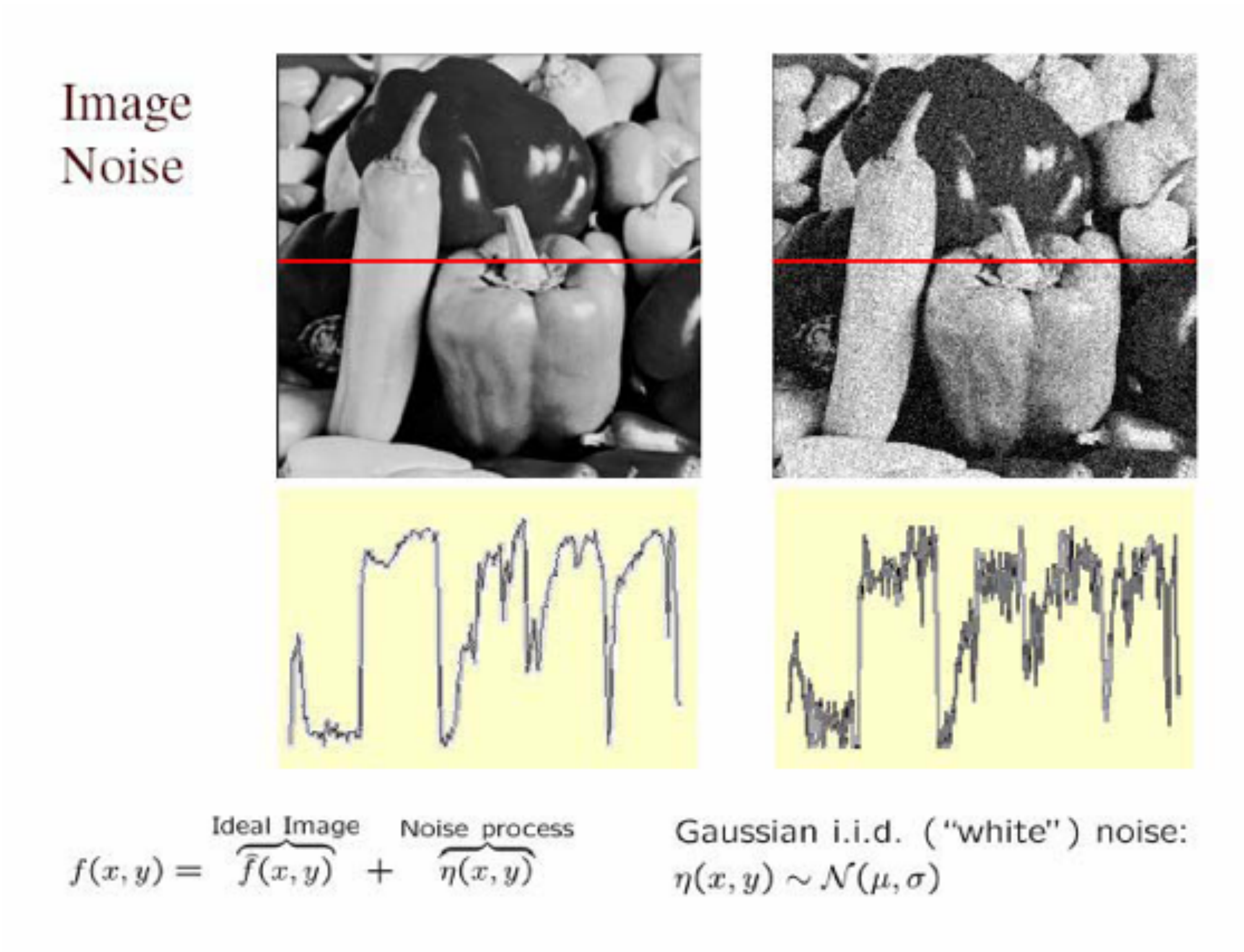



- **Salt and pepper noise**: contains random occurrences of black and white pixels
- **Impulse noise:** contains random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Source: S. Seitz

# Gaussian noise

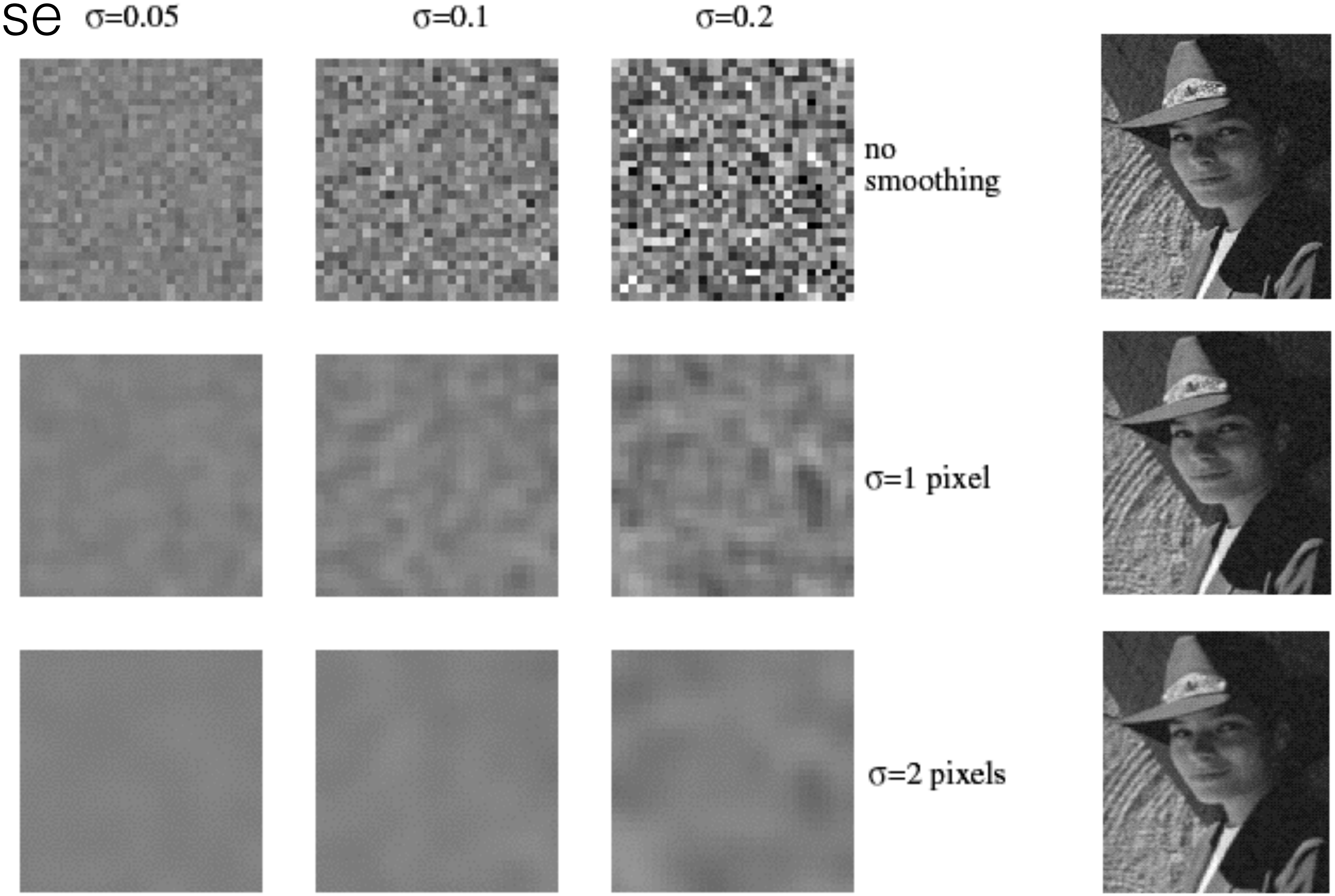
- Mathematical model: sum of many independent factors
- Good for small standard deviations
- Assumption: independent, zero-mean noise




Source: M. Hebert

# Reducing Gaussian noise

noise




Smoothing with larger standard deviations suppresses noise, but also blurs the image

# Reducing salt-and-pepper noise

3x3

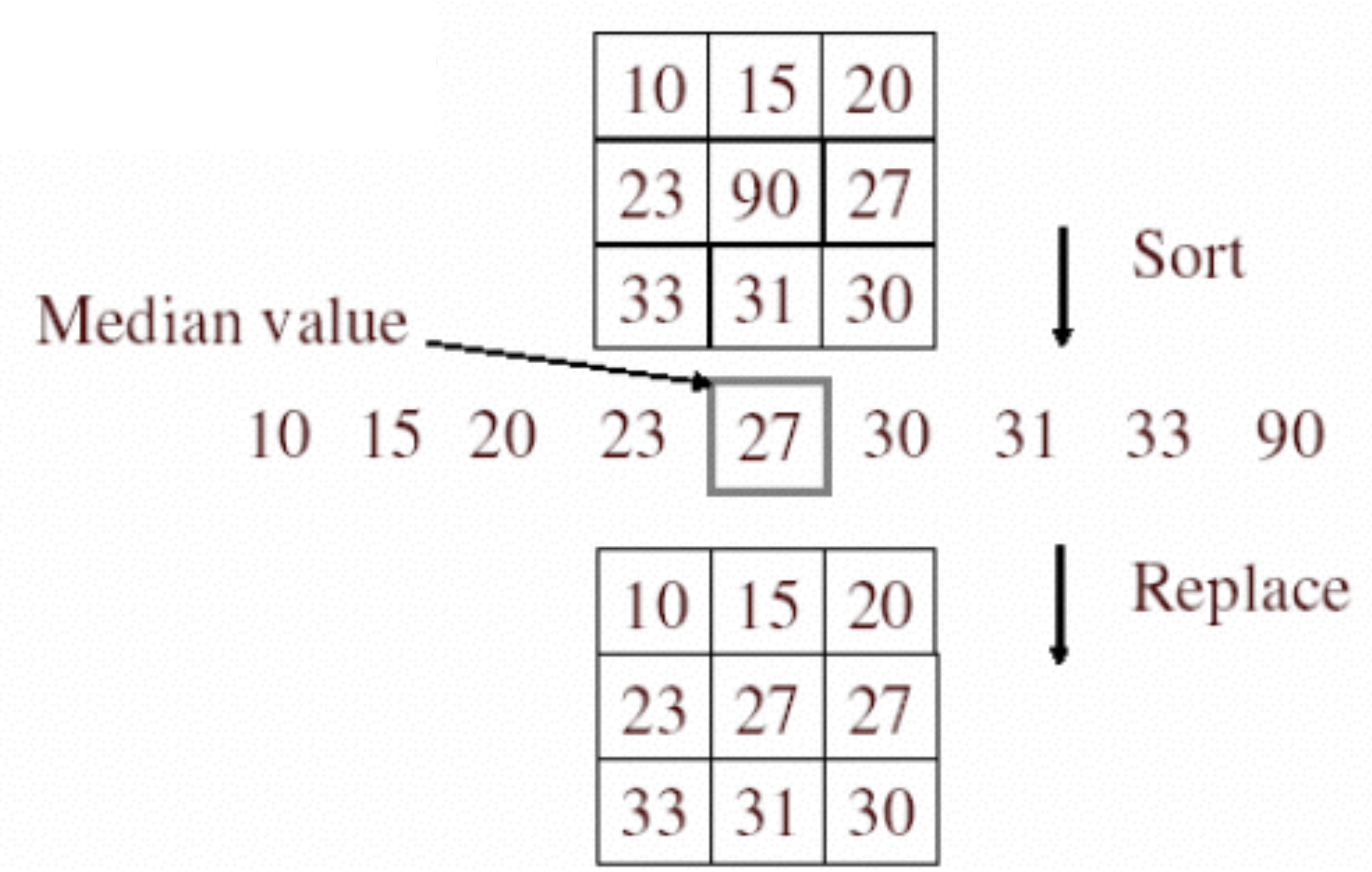
5x5

7x7

What's wrong with the results?

# Alternative idea: Median filtering

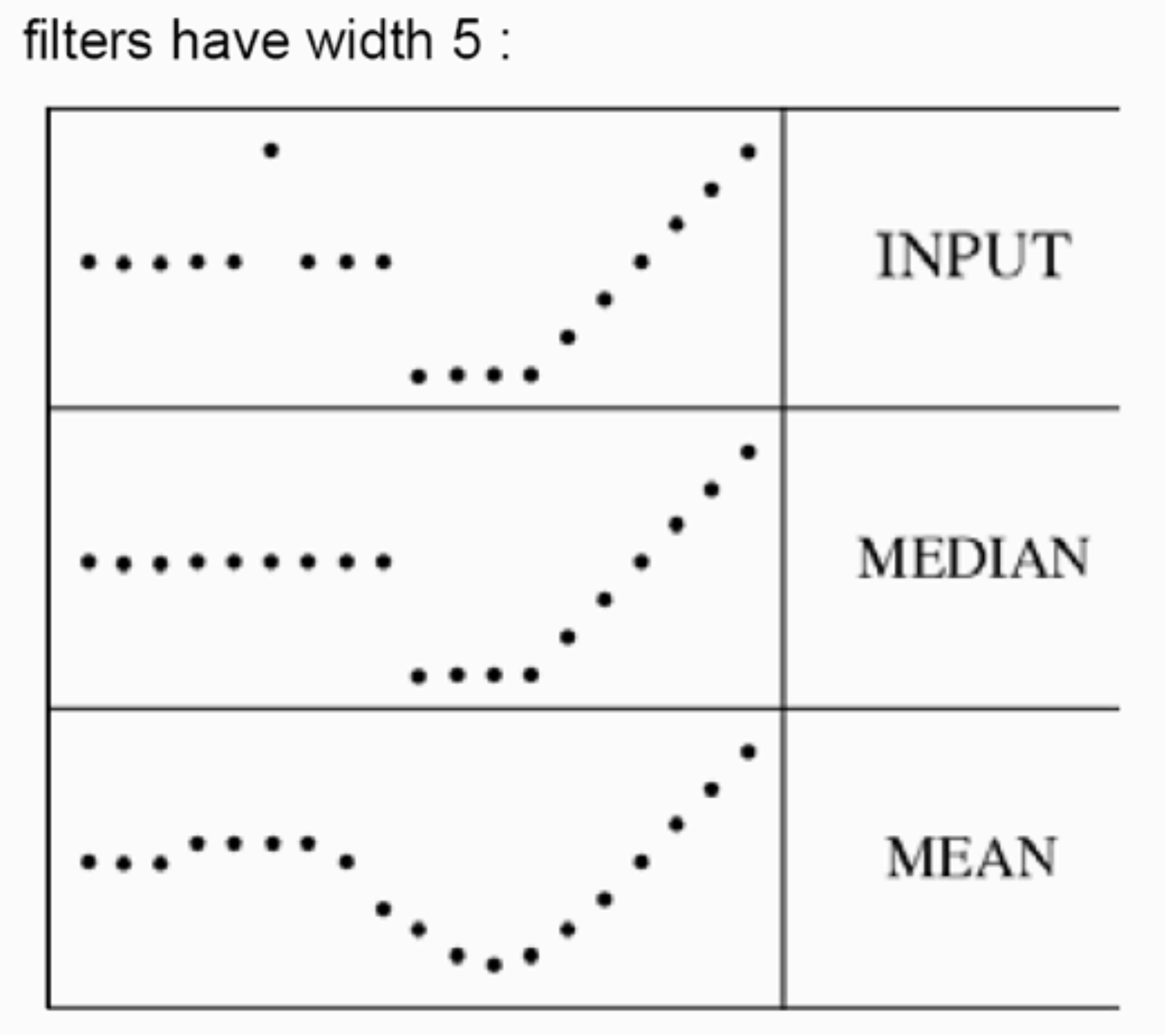
- A **median filter** operates over a window by selecting the median intensity in the window




- Is median filtering linear?

# Median filter

- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers

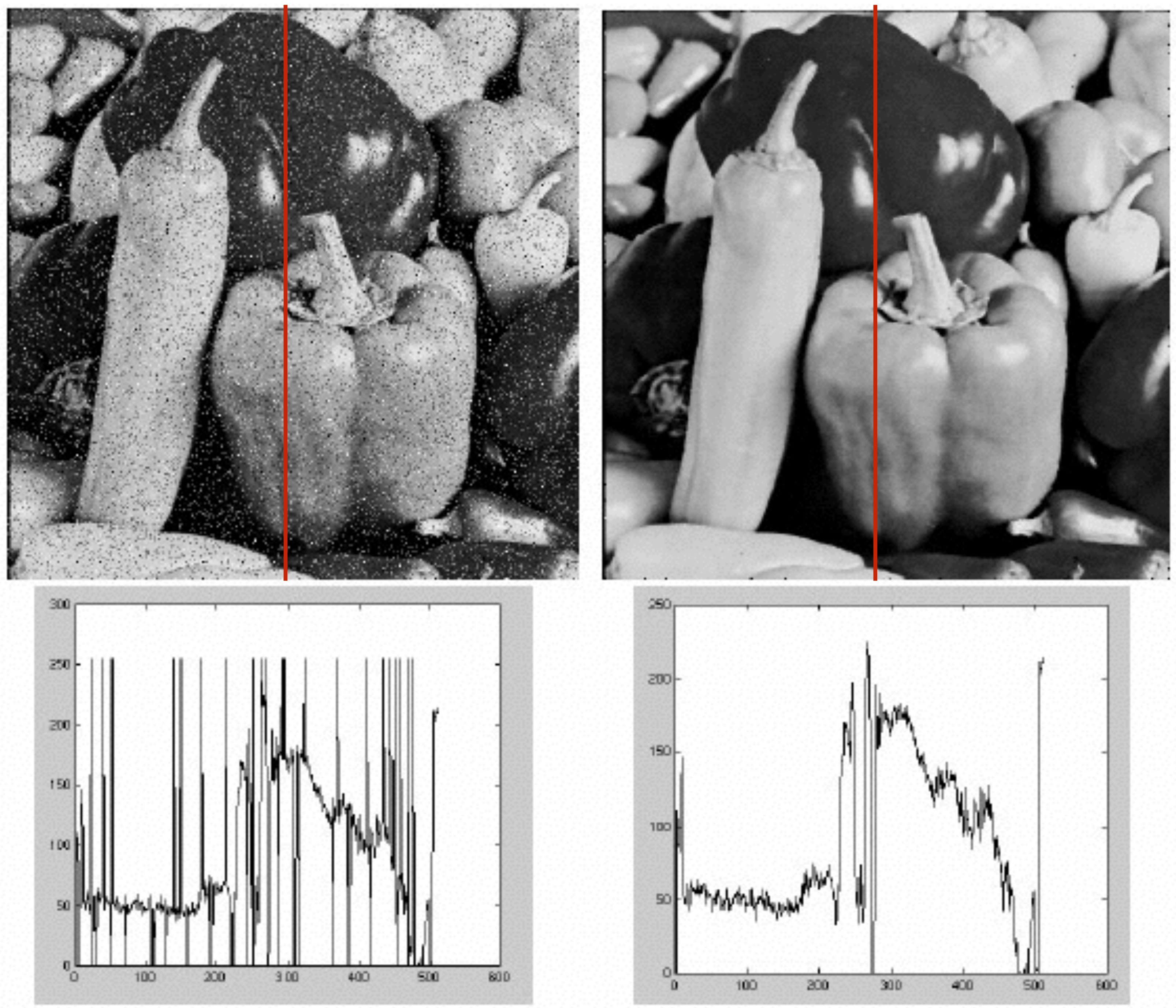



Source: K. Grauman

# Median filter

Salt-and-pepper noise

Median filtered




MATLAB: medfilt2(image, [h w])

Source: M. Hebert