

# **Fundamentals of Image Processing**

**Spatial Filtering**

# Image Filtering

- Image filtering: computes a function of a *local neighborhood* at each pixel position
- Called “Local operator,” “Neighborhood operator,” or “Window operator”
- $f$ : image → image
- Uses:
  - Enhance images
    - Noise reduction, smooth, resize, increase contrast, recolor, artistic effects, etc.
  - Extract features from images
    - Texture, edges, distinctive points, etc.
  - Detect patterns
    - Template matching, e.g., eye template

# Filtering

- The name “filter” is borrowed from frequency domain processing (next week’s topic)
- Accept or reject certain frequency components
- Fourier (1807):  
Periodic functions could be represented as a weighted sum of sines and cosines

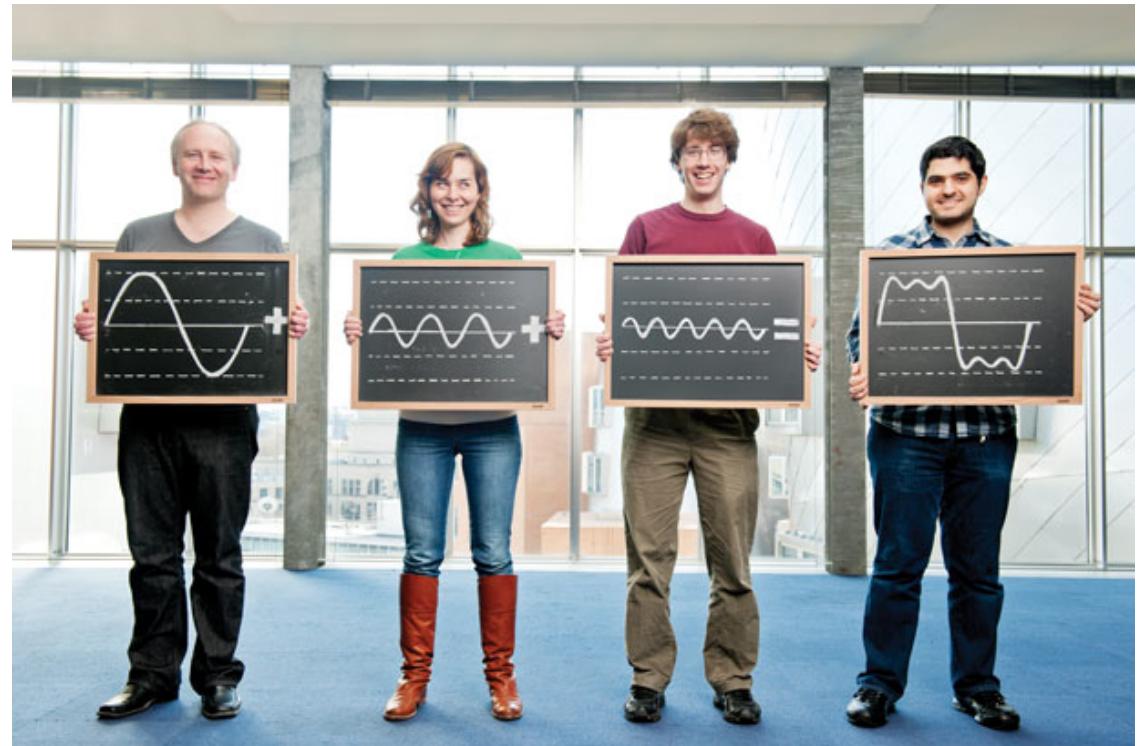
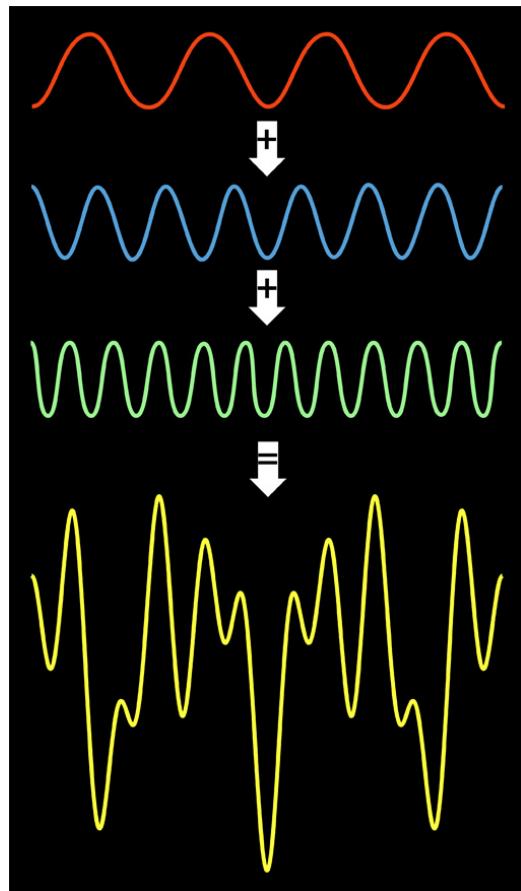


Image courtesy of Technology Review

# Signals

- A signal is composed of low and high frequency components



low frequency components: smooth /  
piecewise smooth

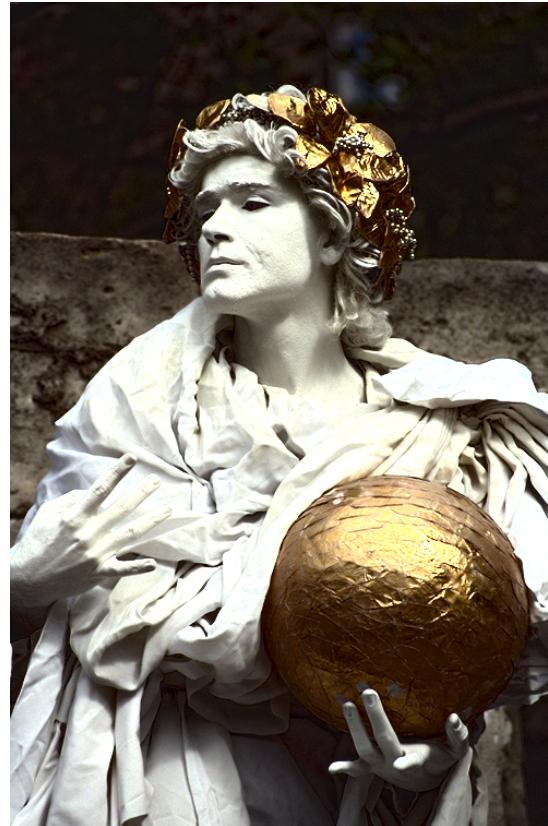
Neighboring pixels have similar brightness values  
You're within a region

high frequency components: oscillatory  
Neighboring pixels have different brightness values  
You're either at the edges or noise points

# Low/high frequencies vs. fine/coarse-scale details



Original image

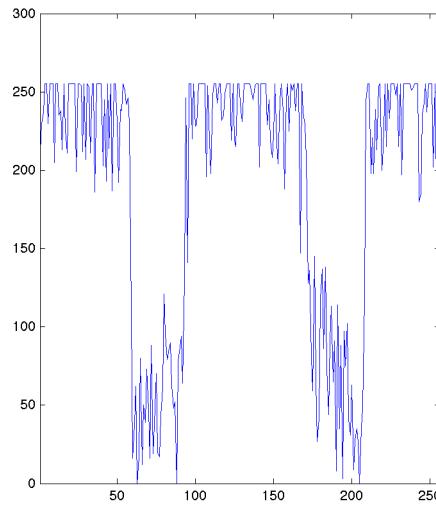
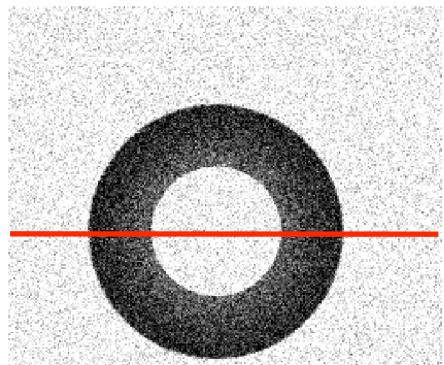
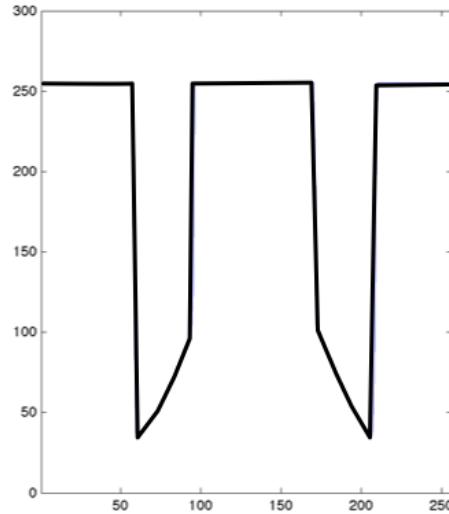
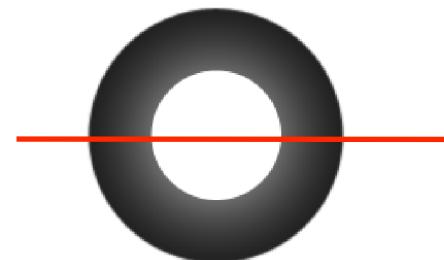


Low-frequencies  
(coarse-scale details)  
boosted



High-frequencies  
(fine-scale details)  
boosted

# Signals – Examples



# Motivation: noise reduction

- Assume image is degraded with an additive model.
- Then,

Observation = True signal + noise

Observed image = Actual image + noise

low-pass filters      high-pass filters



smooth the image

# Common types of noise

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



Original



Salt and pepper noise

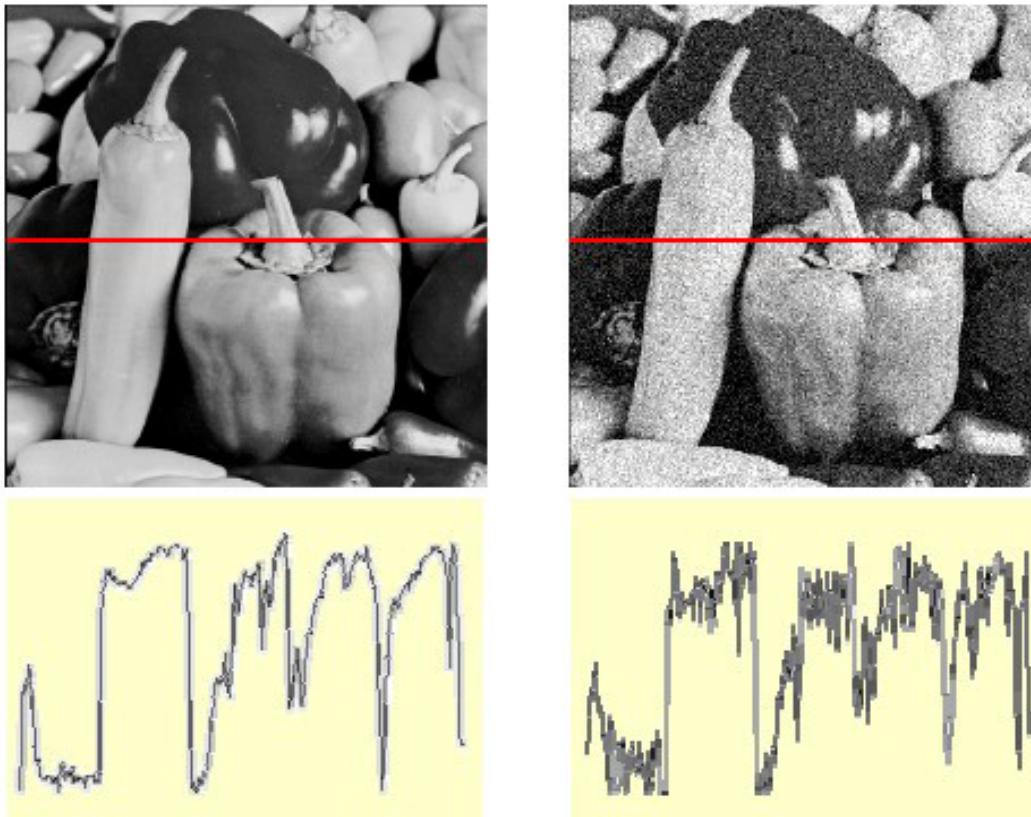


Impulse noise



Gaussian noise

# Gaussian noise



$$f(x, y) = \overbrace{\hat{f}(x, y)}^{\text{Ideal Image}} + \overbrace{\eta(x, y)}^{\text{Noise process}}$$

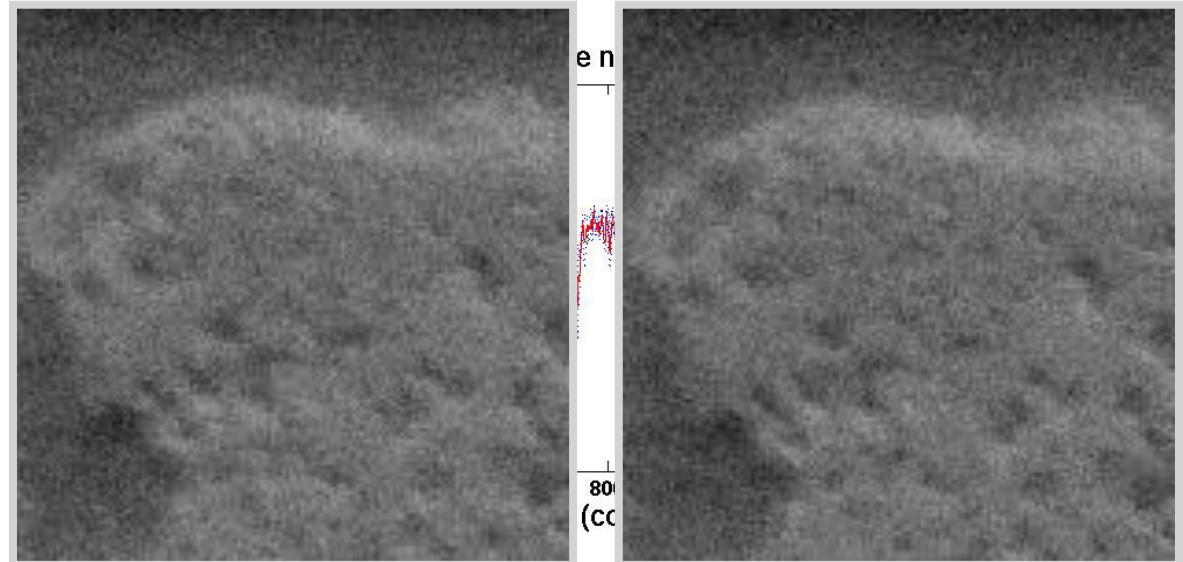
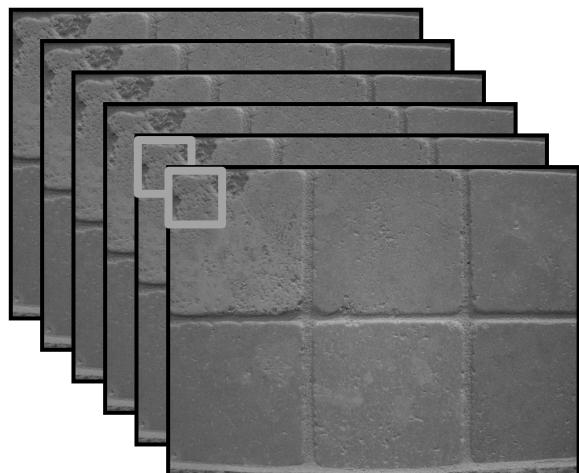
Gaussian i.i.d. ("white") noise:  
 $\eta(x, y) \sim \mathcal{N}(\mu, \sigma)$

```
>> noise = randn(size(im)).*sigma;  
>> output = im + noise;
```

What is the impact of the sigma?

Slide credit: M. Hebert

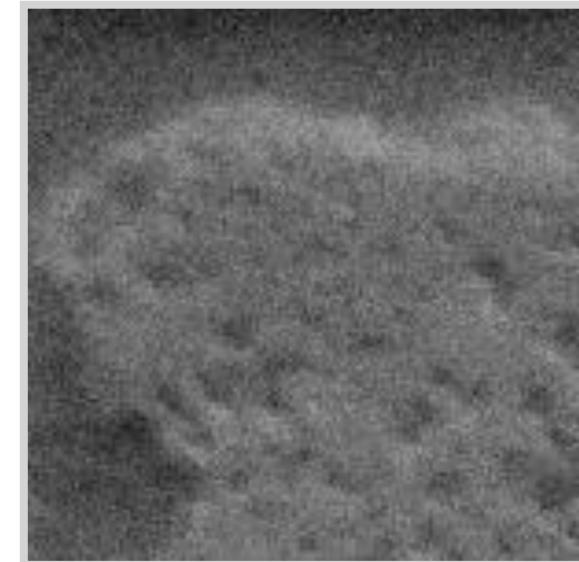
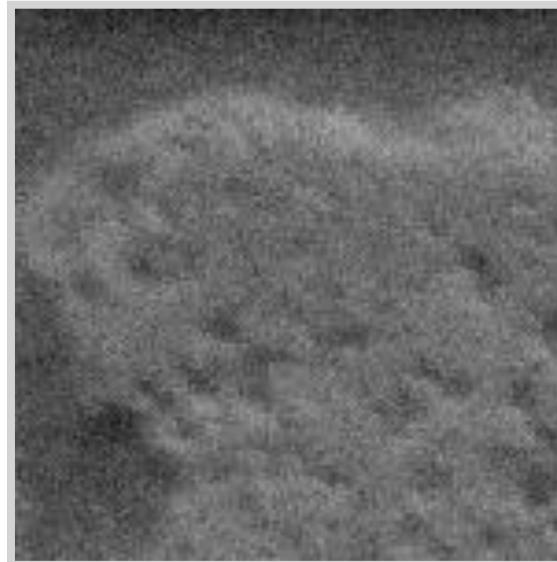
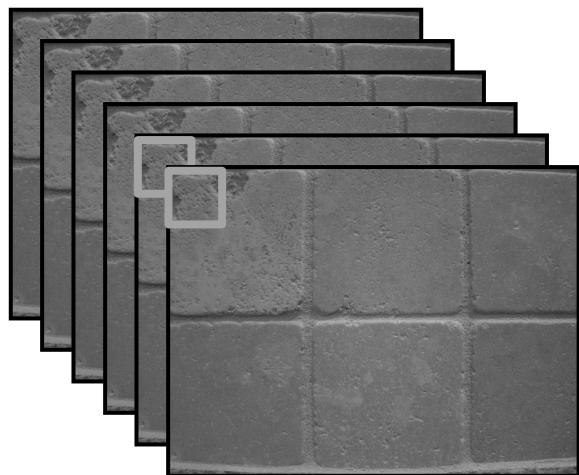
# Motivation: noise reduction



- Make multiple observations of the same static scene
- Take the average
- Even multiple images of the same static scene will not be identical.

Adapted from: K. Grauman

# Motivation: noise reduction



- Make multiple observations of the same static scene
- Take the average
- Even multiple images of the same static scene will not be identical.
- What if we can't make multiple observations?  
**What if there's only one image?**

Adapted from: K. Grauman

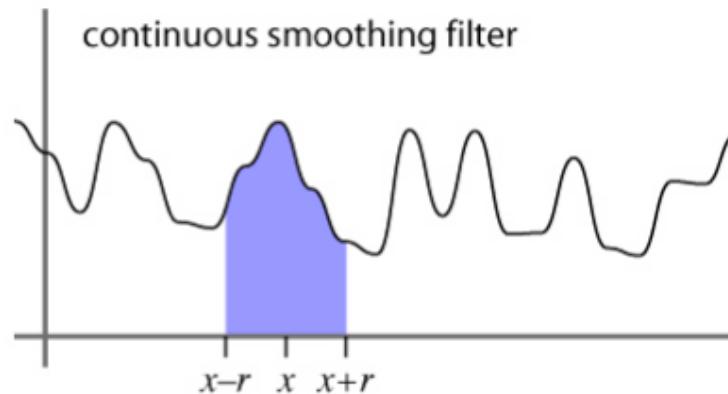
# Image Filtering

说明：像素的邻域内的像素  
可以视为是该像素的  
多个抽样

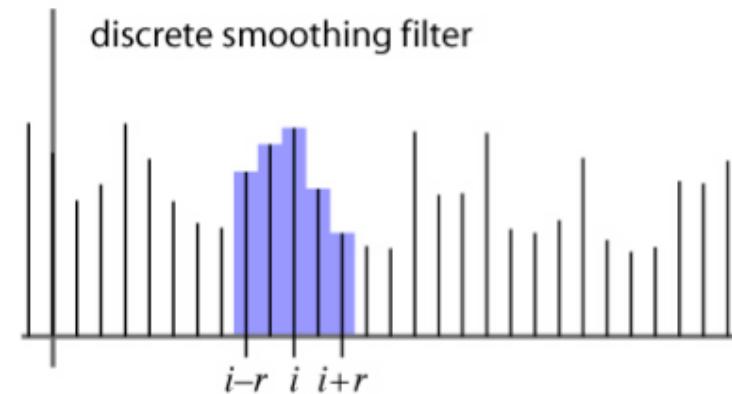
- Idea: Use the information coming from the neighboring pixels for processing
- Design a transformation function of the local neighborhood at each pixel in the image
  - Function specified by a “filter” or mask saying how to combine values from neighbors.
- Various uses of filtering:
  - Enhance an image (denoise, resize, etc)
  - Extract information (texture, edges, etc)
  - Detect patterns (template matching)

# Filtering

- Processing done on a function
  - can be executed in continuous form (e.g. analog circuit)
  - but can also be executed using sampled representation
- Simple example: smoothing by averaging



积分



求和

# Linear filtering

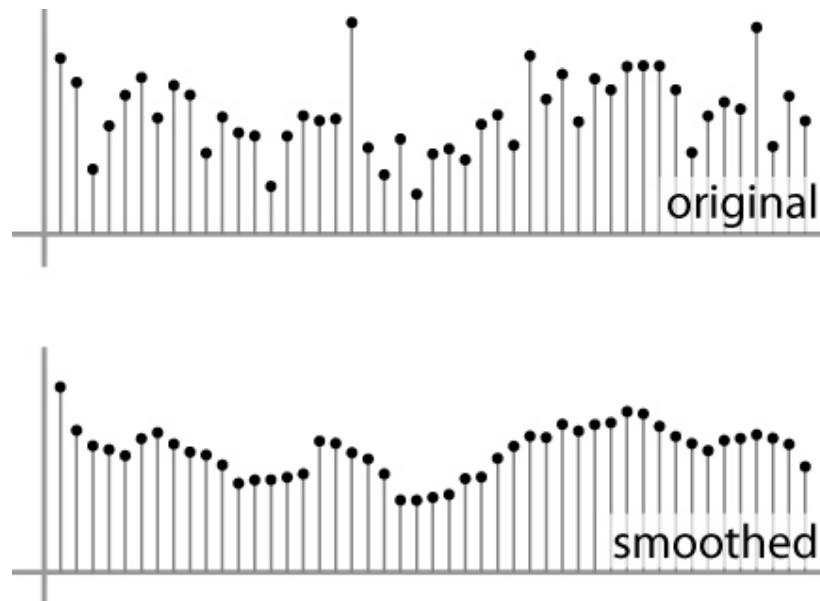
- Filtered value is the linear combination of neighboring pixel values.
- Key properties
  - linearity:  $\text{filter}(f + g) = \text{filter}(f) + \text{filter}(g)$
  - shift invariance: behavior invariant to shifting the input
    - delaying an audio signal
    - sliding an image around
- Can be modeled mathematically by *convolution*

# **First attempt at a solution**

- Let's replace each pixel with an average of all the values in its neighborhood
- Assumptions:
  - Expect pixels to be like their neighbors (spatial regularity in images)
  - Expect noise processes to be independent from pixel to pixel

# First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:



# Convolution warm-up

- Same moving average operation, expressed mathematically:

$$b_{\text{smooth}}[i] = \frac{1}{2r + 1} \sum_{j=i-r}^{i+r} b[j]$$

# Discrete convolution

- Simple averaging:

$$b_{\text{smooth}}[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} b[j]$$

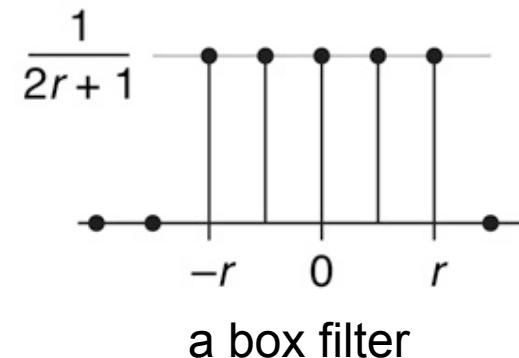
- every sample gets the same weight
- Convolution: same idea but with **weighted average**

$$(a \star b)[i] = \sum_j a[j]b[i-j]$$

- each sample gets its own weight (normally zero far away)
- This is all convolution is: it is a **moving weighted average**

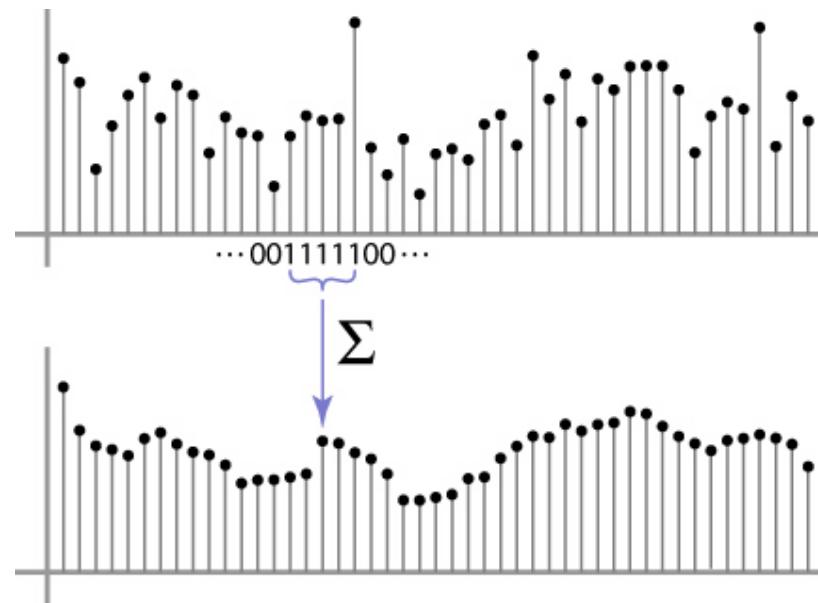
# Filters

- Sequence of weights  $a[j]$  is called a *filter*
- Filter is nonzero over its *region of support*
  - usually centered on zero: support radius  $r$
- Filter is *normalized* so that it sums to 1.0
  - this makes for a weighted average, not just any old weighted sum
- Most filters are *symmetric* about 0
  - since for images we usually want to treat left and right the same



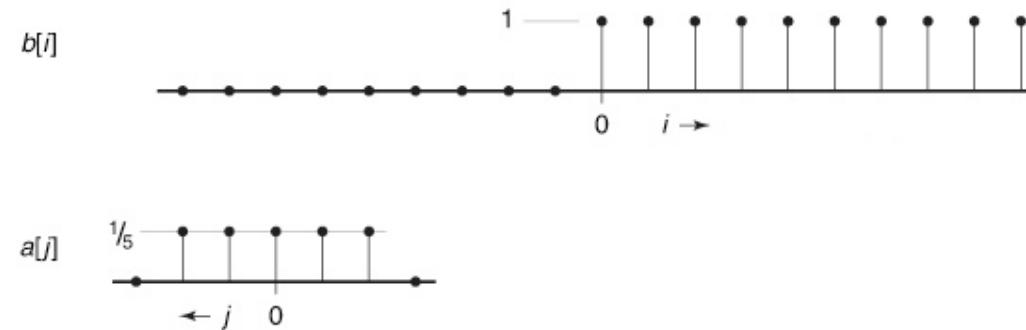
# Convolution and filtering

- Can express sliding average as convolution with a *box filter*
- $a_{\text{box}} = [..., 0, 1, 1, 1, 1, 1, 1, 0, ...]$



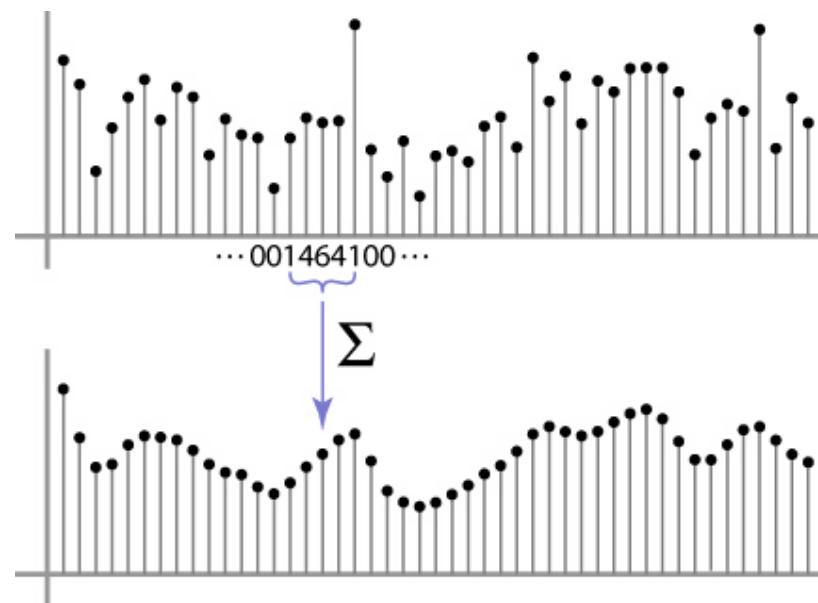
Slide credit: S. Marschner

# Example: box and step



# Convolution and filtering

- Convolution applies with any sequence of weights
- Example: bell curve (gaussian-like)  $[..., 1, 4, 6, 4, 1, ...]/16$



# And in pseudocode...

```
function convolve(sequence a, sequence b, int r, int i )  
    s = 0  
    for j = -r to r  
        s = s + a[j] b[i - j]  
    return s
```

# Key properties

- **Linearity:**  $\text{filter}(f_1 + f_2) = \text{filter}(f_1) + \text{filter}(f_2)$
- **Shift invariance:**  $\text{filter}(\text{shift}(f)) = \text{shift}(\text{filter}(f))$ 
  - same behavior regardless of pixel location, i.e. the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.
- 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  $e = [..., 0, 0, 1, 0, 0, ...]$ ,  
 $a * e = a$

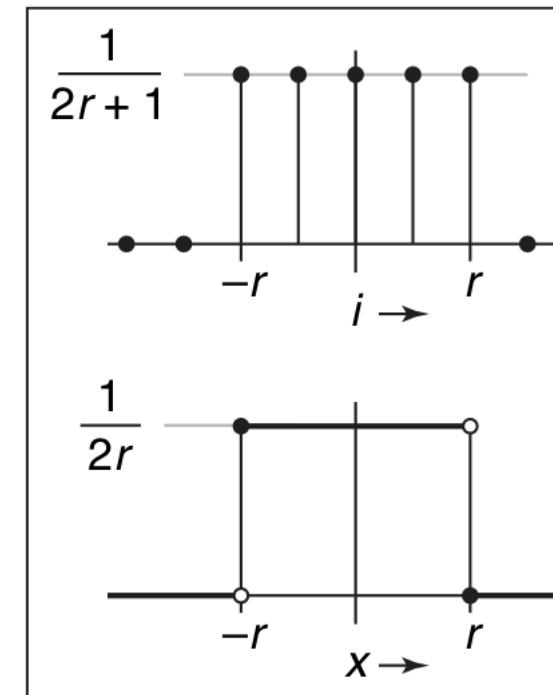
# **A gallery of filters**

- **Box filter**
  - Simple and cheap
- **Tent filter**
  - Linear interpolation
- **Gaussian filter**
  - Very smooth antialiasing filter

# Box filter

$$a_{\text{box},r}[i] = \begin{cases} 1/(2r+1) & |i| \leq r, \\ 0 & \text{otherwise.} \end{cases}$$

$$f_{\text{box},r}(x) = \begin{cases} 1/(2r) & -r \leq x < r, \\ 0 & \text{otherwise.} \end{cases}$$

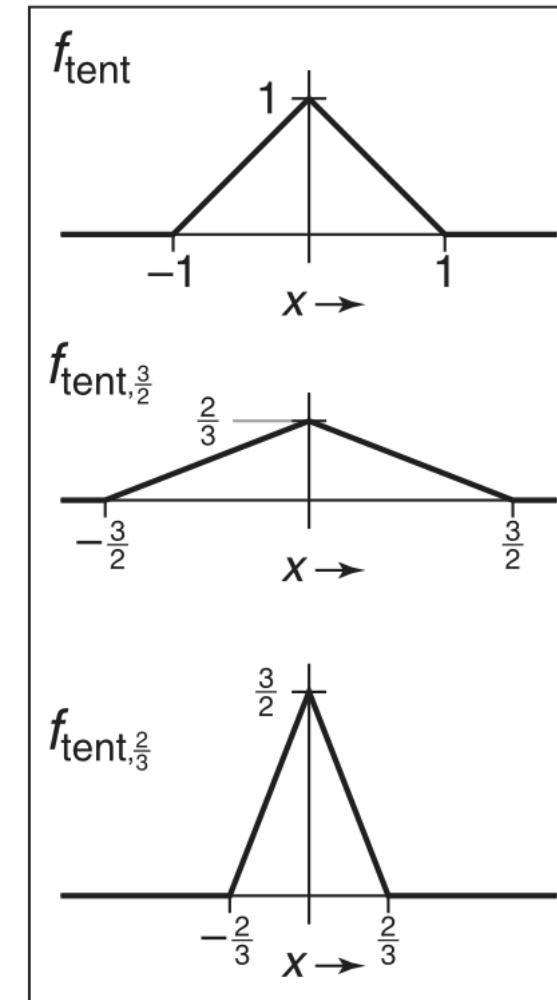


Slide credit: S. Marschner

# Tent filter

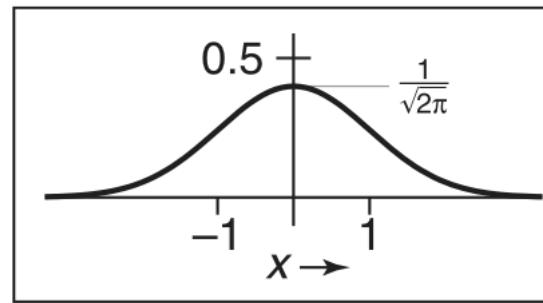
$$f_{\text{tent}}(x) = \begin{cases} 1 - |x| & |x| < 1, \\ 0 & \text{otherwise;} \end{cases}$$

$$f_{\text{tent},r}(x) = \frac{f_{\text{tent}}(x/r)}{r}.$$



Slide credit: S. Marschner

# Gaussian filter



$$f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

# Discrete filtering in 2D

- Same equation, one more index

$$(a \star b)[i, j] = \sum_{i', j'} a[i', j']b[i - i', j - j']$$

- now the filter is a rectangle you slide around over a grid of numbers
  - Usefulness of associativity
  - often apply several filters one after another:  $((a * b_1) * b_2) * b_3$
  - this is equivalent to applying one filter:  $a * (b_1 * b_2 * b_3)$

# And in pseudocode...

```
function convolve2d(filter2d a, filter2d b, int i, int j)
    s = 0
    r = a.radius
    for  $i' = -r$  to  $r$  do
        for  $j' = -r$  to  $r$  do
             $s = s + a[i'][j']b[i - i'][j - j']$ 
    return s
```

# Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

			0							

# Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

			0	10						

# Moving Average In 2D

$$F[x, y]$$

$$G[x, y]$$

# Moving Average In 2D

$F[x, y]$

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

# Moving Average In 2D

$$F[x, y]$$

$$G[x, y]$$

# Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	0	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
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	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 Correlation Filtering

- Center filter  $g$  at each pixel in image  $f$
- Multiply weights by corresponding pixels
- Set resulting value in output image  $h$
- $g$  is called a ***filter***, ***mask***, ***kernel***, or ***template***
- Linear filtering is sum of dot product at each pixel position
- Filtering operation called ***cross-correlation***

# Correlation filtering

Say the averaging window size is  $2k+1 \times 2k+1$ :

$$G[i, j] = \frac{1}{(2k+1)^2} \underbrace{\sum_{u=-k}^k}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{v=-k}^k}_{\text{Loop over all pixels in neighborhood around image pixel } F[i, j]} F[i + u, j + v]$$

*Attribute uniform weight to each pixel*

*Loop over all pixels in neighborhood around image pixel  $F[i, j]$*

1. 如果滤波器对称，即  $h(u, v) = h(-u, -v)$ , 则卷积滤波和相关滤波等价
2. 将相关滤波器  $h(u, v)$  按中心逆时针旋转 180 度再作卷积，和相关滤波具有等价效果
3. 使用卷积是为了应用有关频域处理的卷积定理（第四章）

Now generalize to allow different weights depending on neighboring pixel's relative position:

对比：1) 卷积： $H(u, v) \rightarrow F(i-u, j-v)$

2) 相关： $H(u, v) \rightarrow F(i+u, j+v)$

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] \underbrace{F[i + u, j + v]}_{\text{Non-uniform weights}}$$

# Correlation filtering

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i + u, j + v]$$

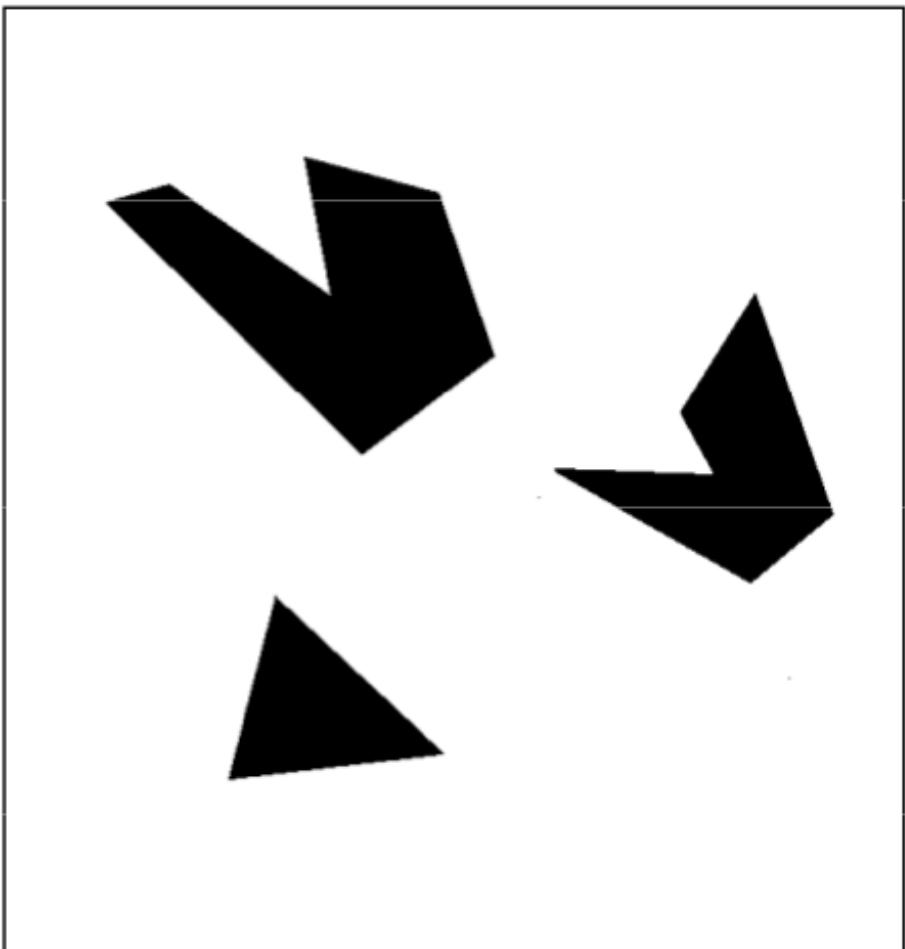
This is called cross-correlation, denoted

$$G = H \otimes F$$

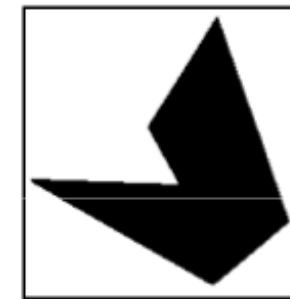
Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter “kernel” or “mask”  $H[u, v]$  is the prescription for the weights in the linear combination.

# Correlation filtering

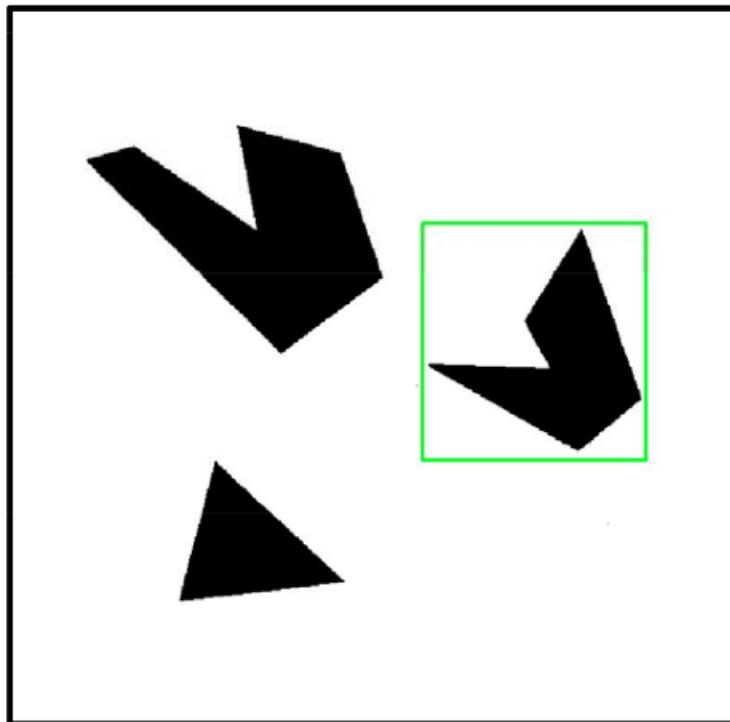


Scene

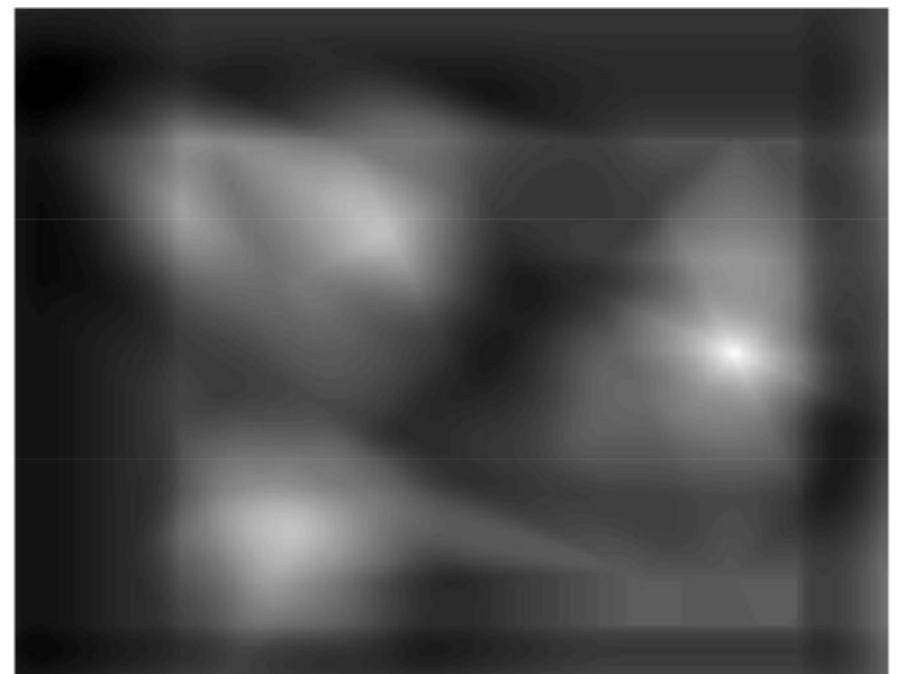


Template (mask)

# Correlation filtering



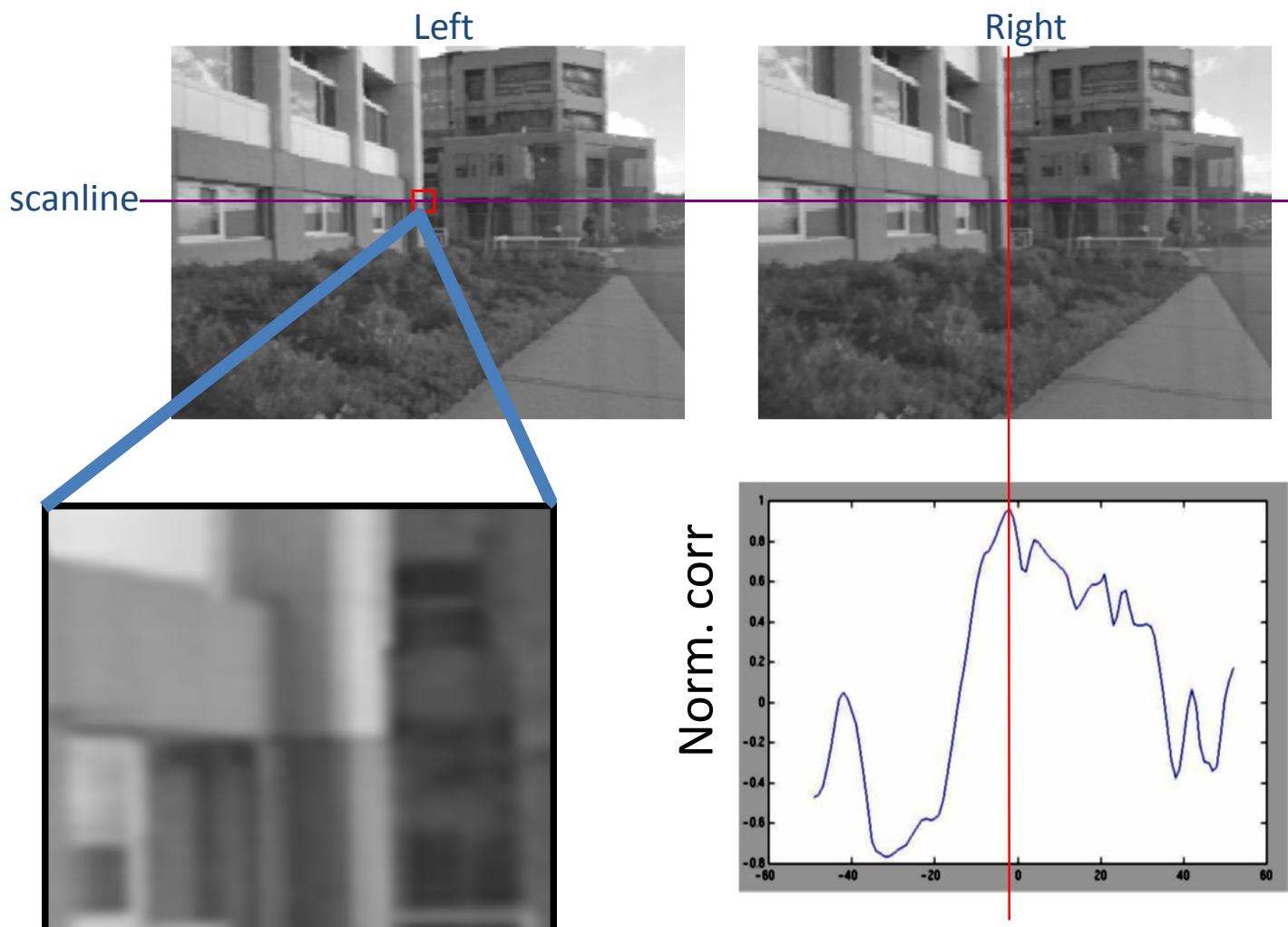
Detected template



Correlation map

可用于检测和定位一个  
由kernel表示的pattern，  
被检测pattern处响应强

# Cross correlation example



Slide credit: Fei-Fei Li

# 有关模板匹配 (1)

---

可以直接使用  $G(i, j) = \sum_{u=-k}^k \sum_{v=-l}^l H(u, v)F(i+u, j+v)$  吗？

考虑下图在车图中匹配车轮的例子，如果车轮匹配到一个各像素灰度值都较大的区域，可能会有较大的相应输出，得到错误结果。



# 有关模板匹配 (2)

---

相关匹配公式需要改造，可以使用教材中的相关计算，或

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

其中  $H(u, v)$  为匹配模板，而分母为图像内容 要对匹配的部分作归一化

# Averaging filter

- What values belong in the kernel  $H$  for the moving average example?

$$F[x, y] \quad \otimes \quad H[u, v]$$

The input image  $F[x, y]$  is a 9x9 grid of values. The central pixel at position (4,4) has a value of 90 and is highlighted with a red box. All other pixels have a value of 0.

$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & ? & 1 \\ 1 & 1 & 1 \end{matrix}$$

“box filter”

The kernel  $H[u, v]$  is a 3x3 matrix with 1's in all positions except the center, which is a question mark. To its left, the value  $\frac{1}{9}$  is written, indicating it is a box filter with a uniform weight of  $\frac{1}{9}$ .

$$G[x, y]$$

The output image  $G[x, y]$  is a 9x9 grid. The central pixel at position (4,4) has a value of 30 and is highlighted with a red box. All other pixels have a value of 0.

$$G = H \otimes F$$

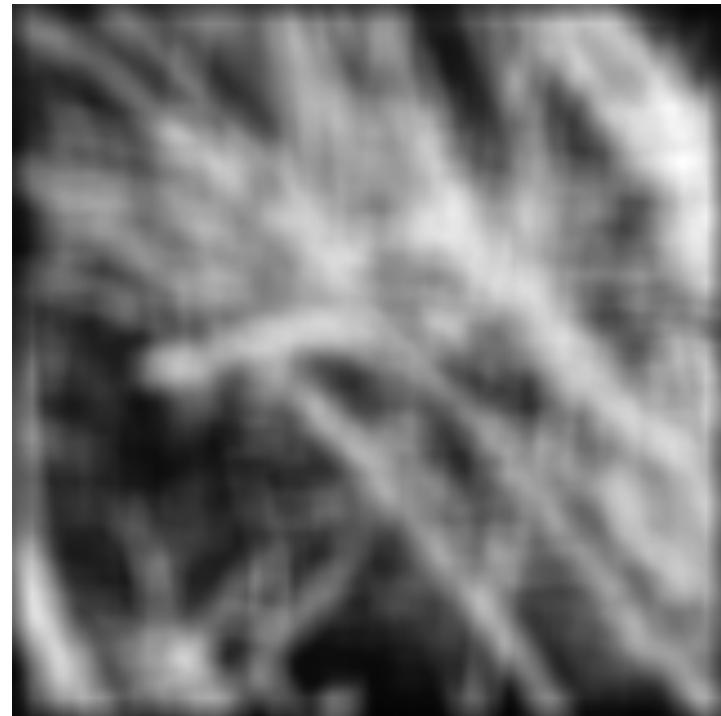
# Smoothing by averaging



depicts box filter:  
white = high value, black = low value



original



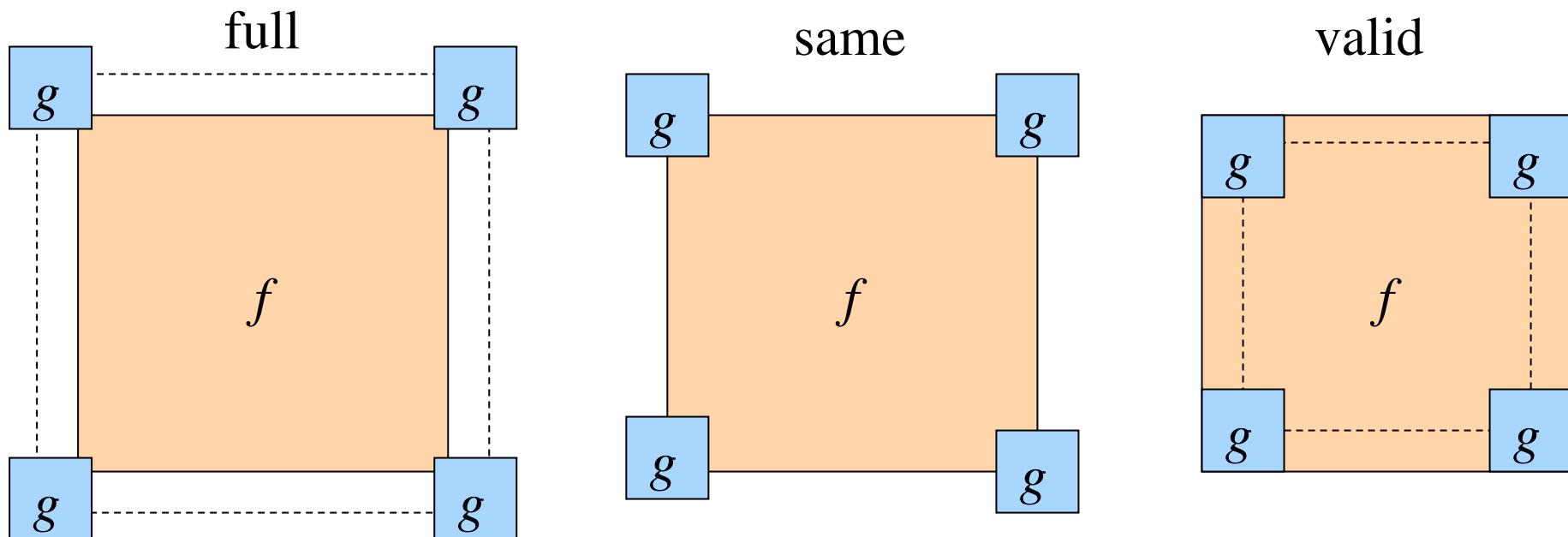
filtered

What if the filter size was  $5 \times 5$  instead of  $3 \times 3$ ?

Slide credit: K. Grauman

# Boundary issues

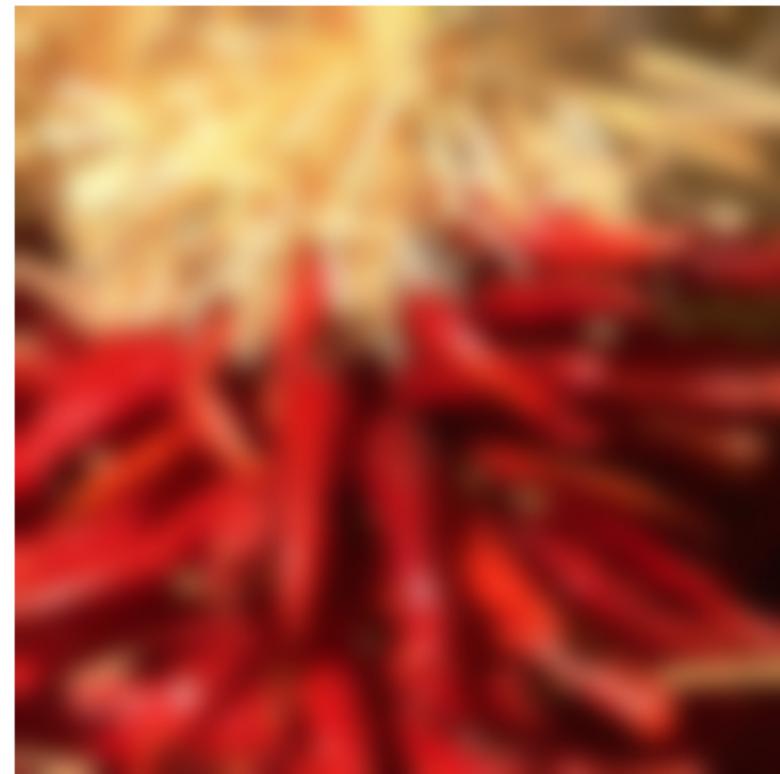
- What is the size of the output?
- MATLAB: output size / “shape” options
  - *shape* = ‘full’: output size is sum of sizes of  $f$  and  $g$
  - *shape* = ‘same’: output size is same as  $f$
  - *shape* = ‘valid’: output size is difference of sizes of  $f$  and  $g$



Slide credit: S. Lazebnik

# Boundary issues

- 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



Slide credit: S. Marschner

# Boundary issues

- 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')`

规则 : 1 ) 低通濾波器 - weights = 1

2 ) 高通濾波器 - weights = 0

# Gaussian filter

- What if we want nearest neighboring pixels to have the most influence on the output?

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

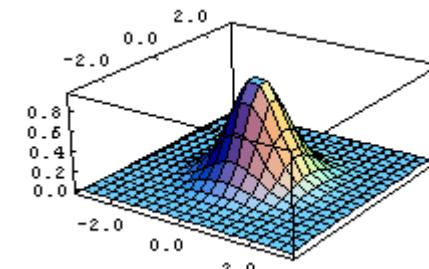
$F[x, y]$

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

$H[u, v]$

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

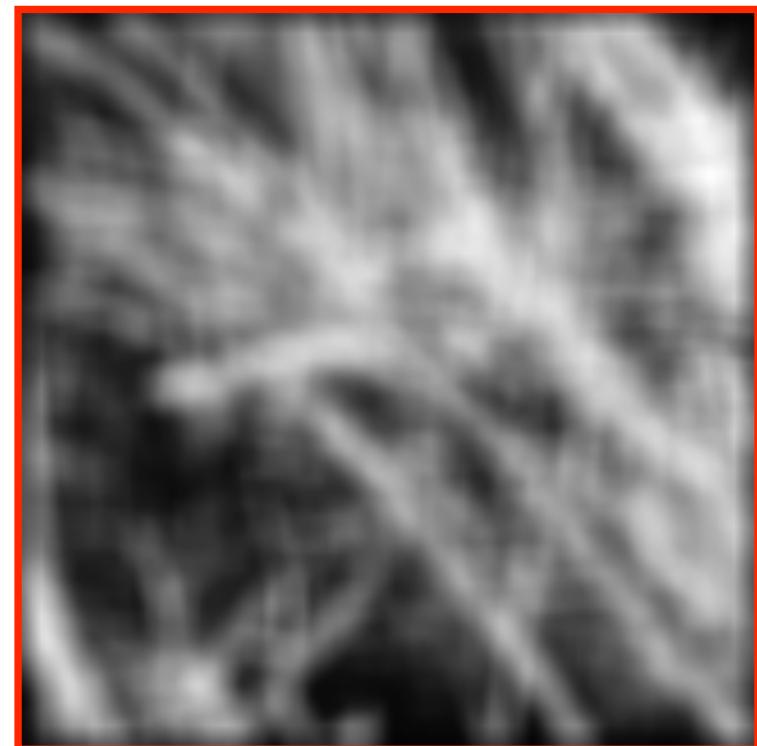
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



- Removes high-frequency components from the image (“low-pass filter”).

Slide credit: S. Seitz

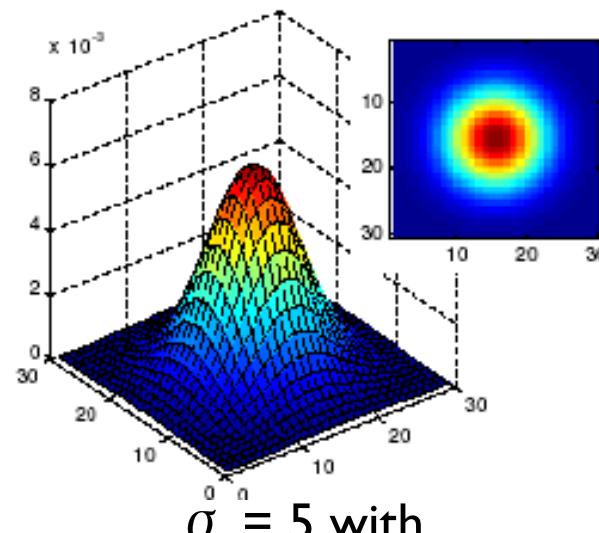
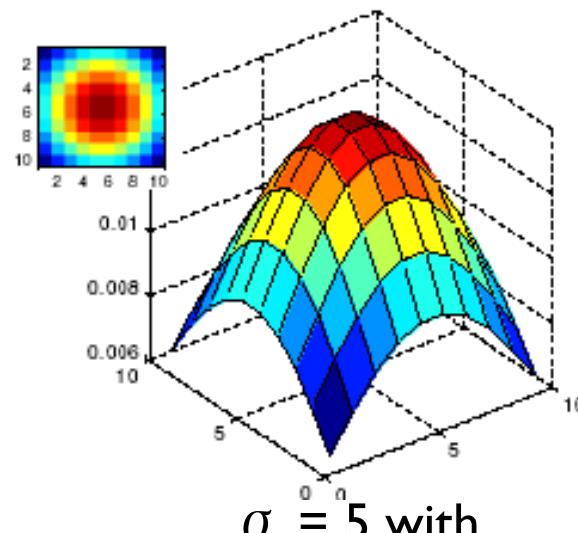
# Smoothing with a Gaussian



Slide credit: K. Grauman

# Gaussian filters

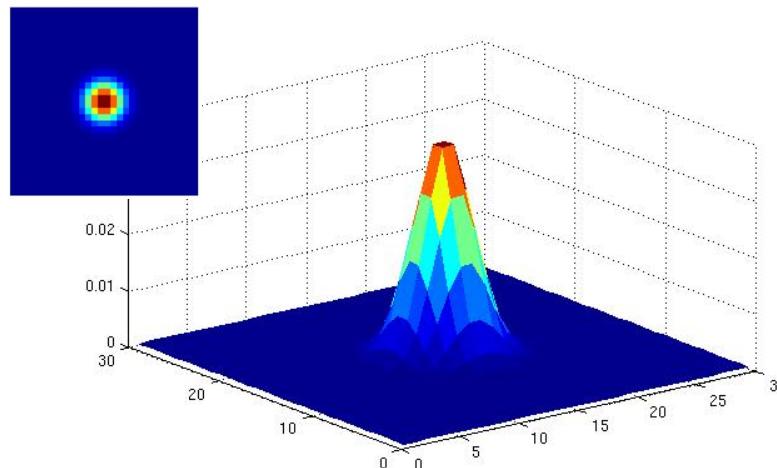
- What parameters matter here?
- **Size** of kernel or mask
  - Note, Gaussian function has infinite support, but discrete filters use finite kernels



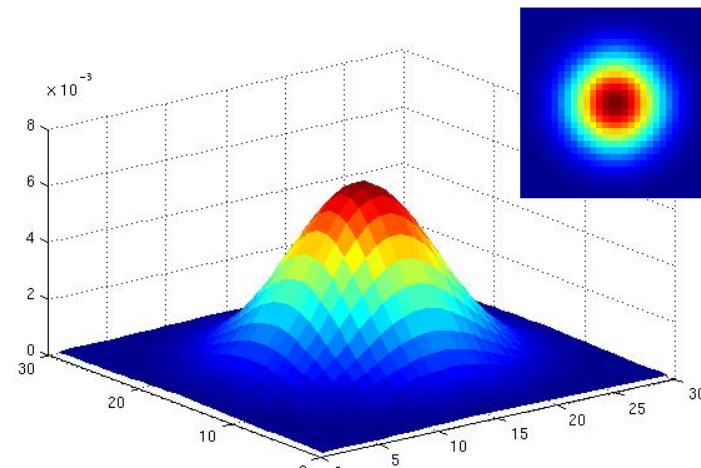
Slide credit: K. Grauman

# Gaussian filters

- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing



$\sigma = 2$  with  
30 x 30 kernel



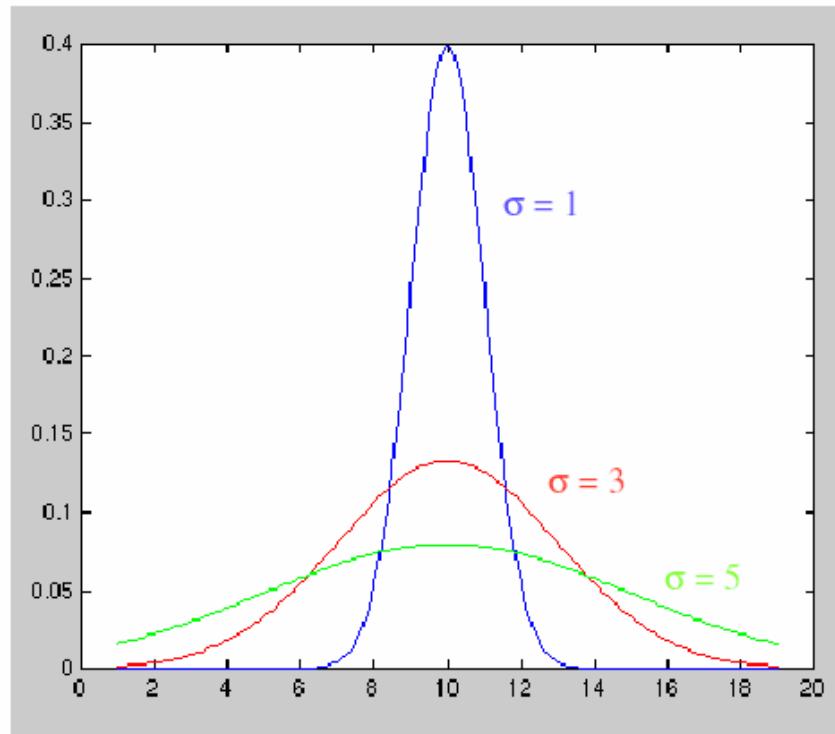
$\sigma = 5$  with  
30 x 30 kernel

# Choosing kernel width

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

经验公式

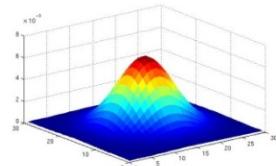
Effect of  $\sigma$



# Matlab

```
>> hsize = 10;  
>> sigma = 5;  
>> h = fspecial('gaussian', hsize, sigma);
```

```
>> mesh(h);
```



```
>> imagesc(h);
```



```
>> outim = imfilter(im, h); % correlation
```

```
>> imshow(outim);
```

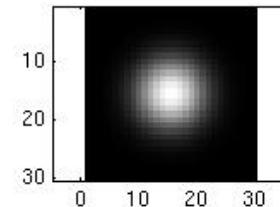
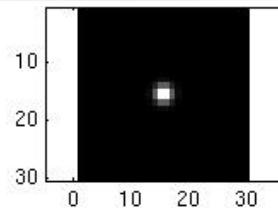


**outim**

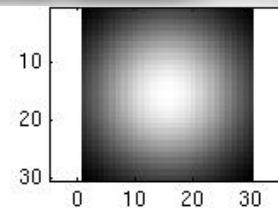
Slide credit: K. Grauman

# Smoothing with a Gaussian

Parameter  $\sigma$  is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



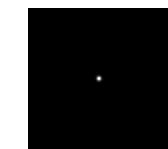
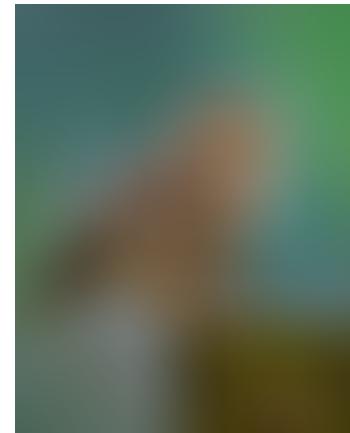
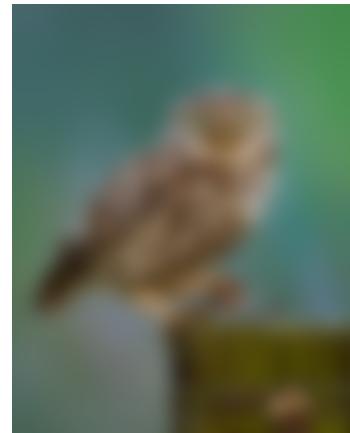
...



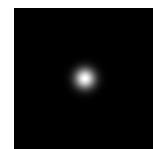
```
for sigma=1:3:10
    h = fspecial('gaussian', fsize, sigma);
    out = imfilter(im, h);
    imshow(out);
    pause;
end
```

Slide credit: K. Grauman

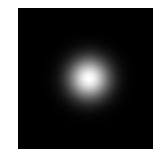
# Gaussian Filters



$\sigma = 1 \text{ pixel}$



$\sigma = 5 \text{ pixels}$



$\sigma = 10 \text{ pixels}$



$\sigma = 30 \text{ pixels}$

# Spatial Resolution and Color



original



R



G



B

Slide credit: C. Dyer

# Blurring the G Component

( 对G影响较大 )



original



processed



R



G



B

Slide credit: C. Dyer

# Blurring the R Component



original



processed



R



G



B

Slide credit: C. Dyer

# Blurring the B Component



original



processed



R



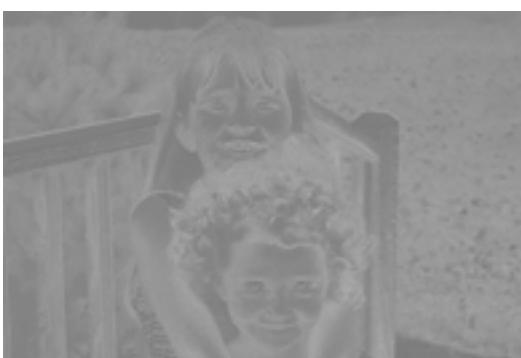
G



B

Slide credit: C. Dyer

# “Lab” Color Representation



- L A transformation of the colors into a color space that is more perceptually meaningful:
  - a: red-green,
  - b: blue-yellow
- a
- b

# Blurring L



original



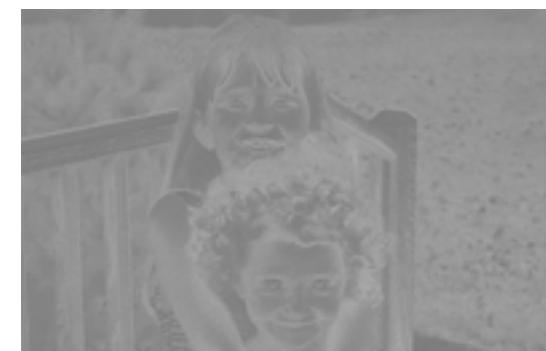
processed



L



a



b

# Blurring a



original



processed



L



a



b

Slide credit: C. Dyer

# Blurring b



original



processed



L



a



b

Slide credit: C. Dyer

滤波器可分离的条件：  
对于kernel  $h(u,v)$ ，有：  
 $h(u,v) = h(u)h(v)$

# Separability

- In some cases, filter is separable, and we can factor into two steps:
  - Convolve all rows 用 $h(v)$
  - Convolve all columns 用 $h(u)$

# Separability of the Gaussian filter

$$\begin{aligned} G_\sigma(x, y) &= \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}} \\ &= \left( \frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}} \right) \left( \frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}} \right) \end{aligned}$$

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

# Separability example

2D convolution  
(center location only)

$$\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix} * \begin{matrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{matrix}$$

The filter factors  
into a product of 1D  
filters:

$$\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix} = \begin{matrix} 1 \\ 2 \\ 1 \end{matrix} \times \begin{matrix} 1 & 2 & 1 \end{matrix}$$

Perform convolution  
along rows:

$$\begin{matrix} 1 & 2 & 1 \end{matrix} * \begin{matrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{matrix} = \begin{matrix} 11 \\ 18 \\ 18 \end{matrix}$$

Followed by convolution  
along the remaining column:

# Why is separability useful?

- 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?
  - $O(n^2 m)$

# Properties of smoothing filters

- Smoothing
  - Values positive
  - Sum to 1  $\rightarrow$  constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove “high-frequency” components; “low-pass” filter

# Filtering an impulse signal

对于脉冲图像F：

1. 和H相关滤波—> H旋转180度
2. 和H卷积滤波—> H本身

What is the result of filtering the impulse signal (image)  $F$  with the 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



a	b	c
d	e	f
g	h	i

$$H[u, v]$$

$$F[x, y]$$


$$G[x, y]$$

# Convolution

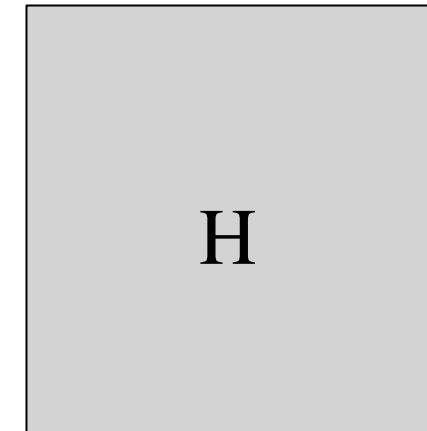
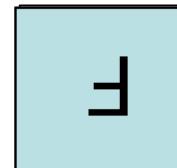
- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v]F[i - u, j - v]$$

$$G = H \star F$$



*Notation for  
convolution  
operator*



# Convolution vs. Correlation

- A **convolution** is an integral that expresses the amount of overlap of one function as it is shifted over another function.
  - convolution is a filtering operation
- **Correlation** compares the **similarity** of **two** sets of **data**.  
Correlation computes a measure of similarity of two input signals as they are shifted by one another. The correlation result reaches a maximum at the time when the two signals match best .  

---

  - correlation is a measure of relatedness of two signals

# Convolution vs. correlation

Convolution

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v]F[i - u, j - v]$$

$$G = H \star F$$

Cross-correlation

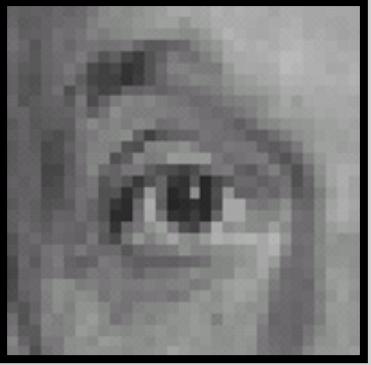
$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v]F[i + u, j + v]$$

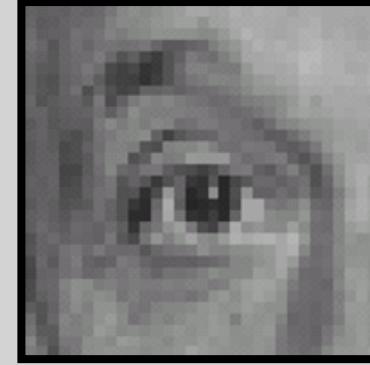
$$G = H \otimes F$$

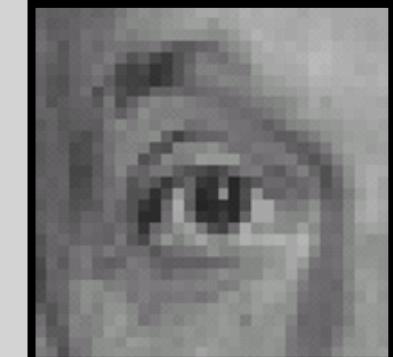
For a Gaussian or box filter, how will the outputs differ?

If the input is an impulse signal, how will the outputs differ?

# Predict the outputs using correlation filtering

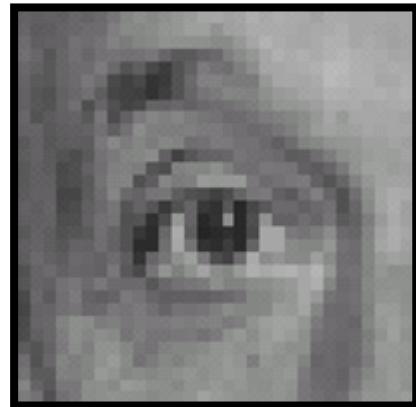

$$\text{Input Image} * \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} = ?$$


$$\text{Input Image} * \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{matrix} = ?$$


$$\text{Input Image} * \begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} = ?$$

Slide credit: K. Grauman

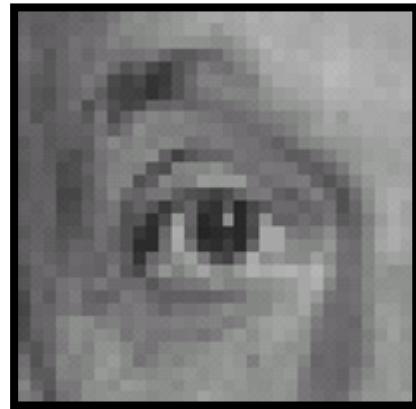
# Practice with linear filters



0	0	0
0	1	0
0	0	0

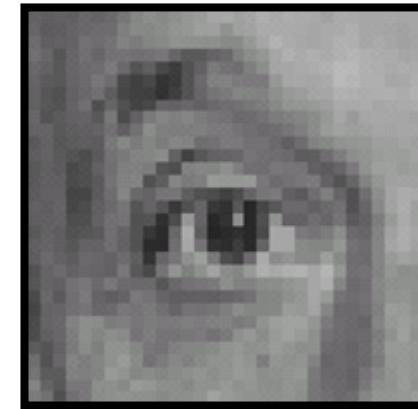
?

# Practice with linear filters



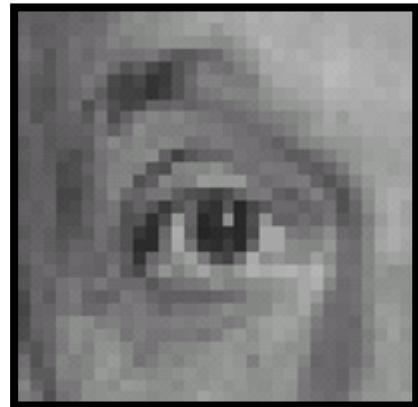
Original

0	0	0
0	1	0
0	0	0



Filtered  
(no change)

# Practice with linear filters

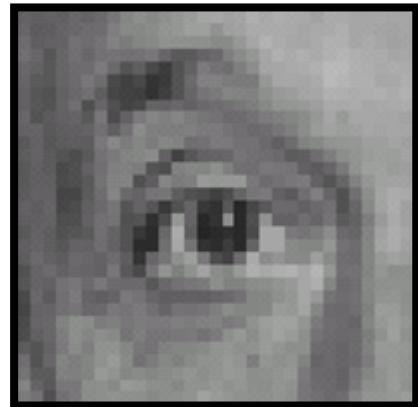


Original

0	0	0
0	0	1
0	0	0

?

# Practice with linear filters



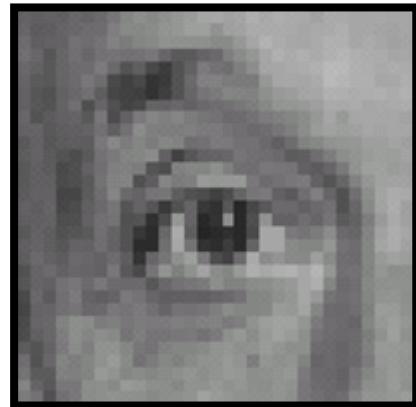
Original

0	0	0
0	0	1
0	0	0



Shifted left  
by 1 pixel with  
correlation

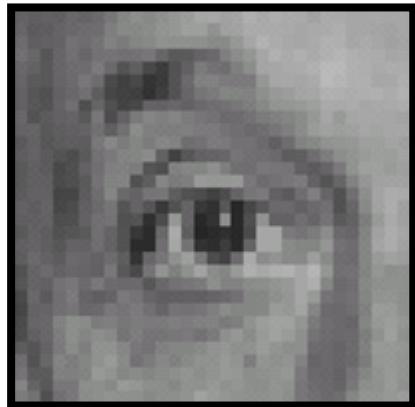
# Practice with linear filters



$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

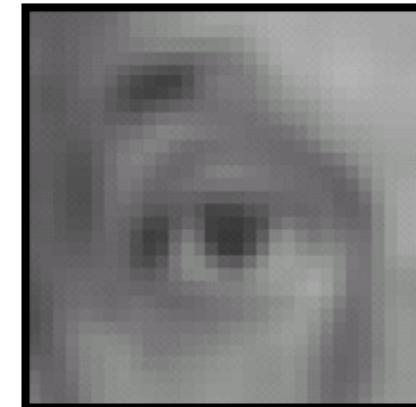
?

# Practice with linear filters



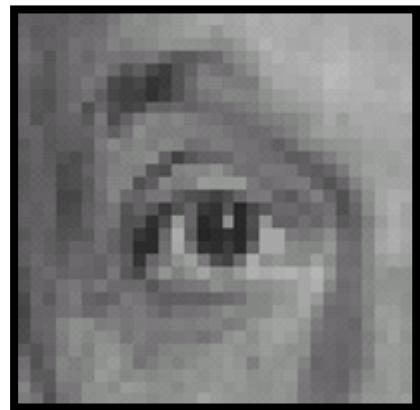
Original

$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$



Blur (with a  
box filter)

# Practice with linear filters



$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

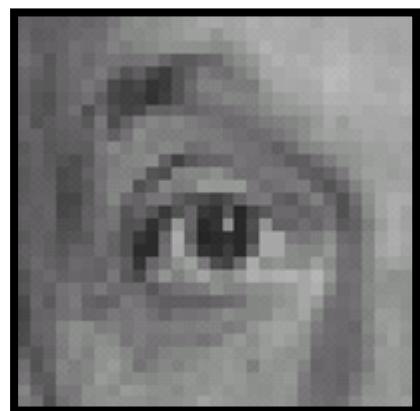
$$- \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

?

Original

# Practice with linear filters

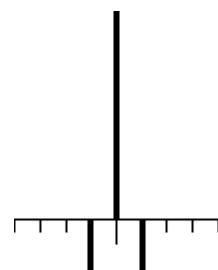
original - smoothed = detail  
original + detail = sharpening  
->  $2 \cdot \text{original} - \text{smoothed}$



Original

0	0	0
0	2	0
0	0	0

$$- \frac{1}{9} \begin{array}{|ccc|} \hline 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ \hline \end{array}$$

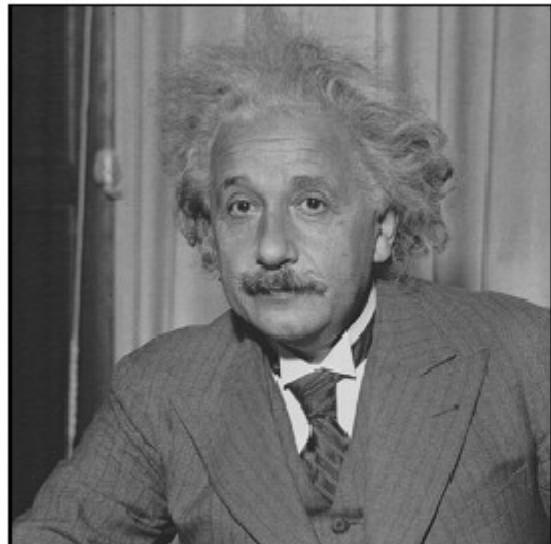


Sharpening filter:  
accentuates differences with  
local average

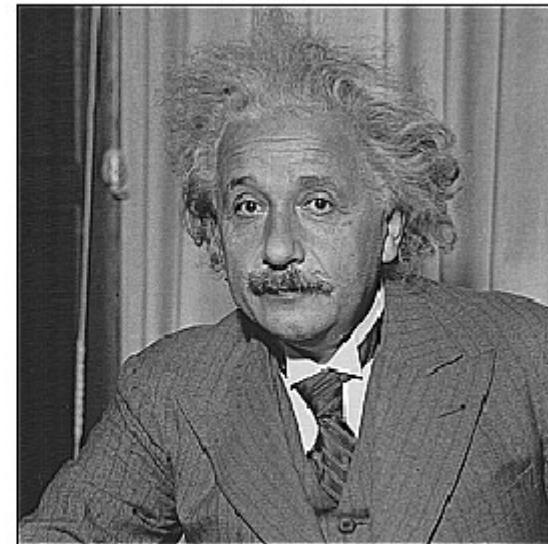
original + original - smoothed  
= original + detail  
= sharpening

Slide credit: D. Lowe

# Filtering examples: sharpening



before



after

Slide credit: K. Grauman

# Sharpening

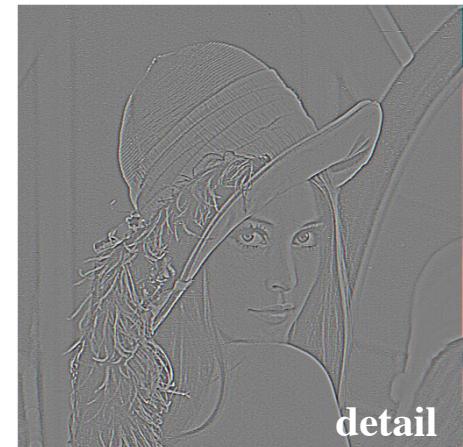
- What does blurring take away?



-



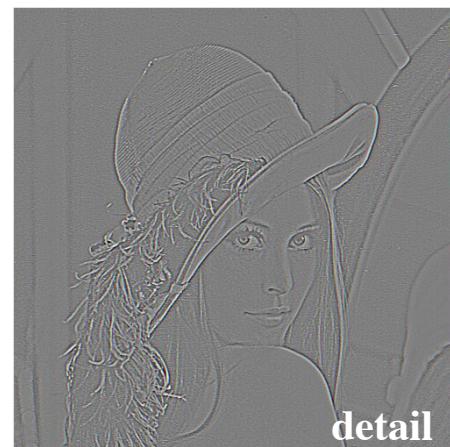
=



Let's add it back:



+



=

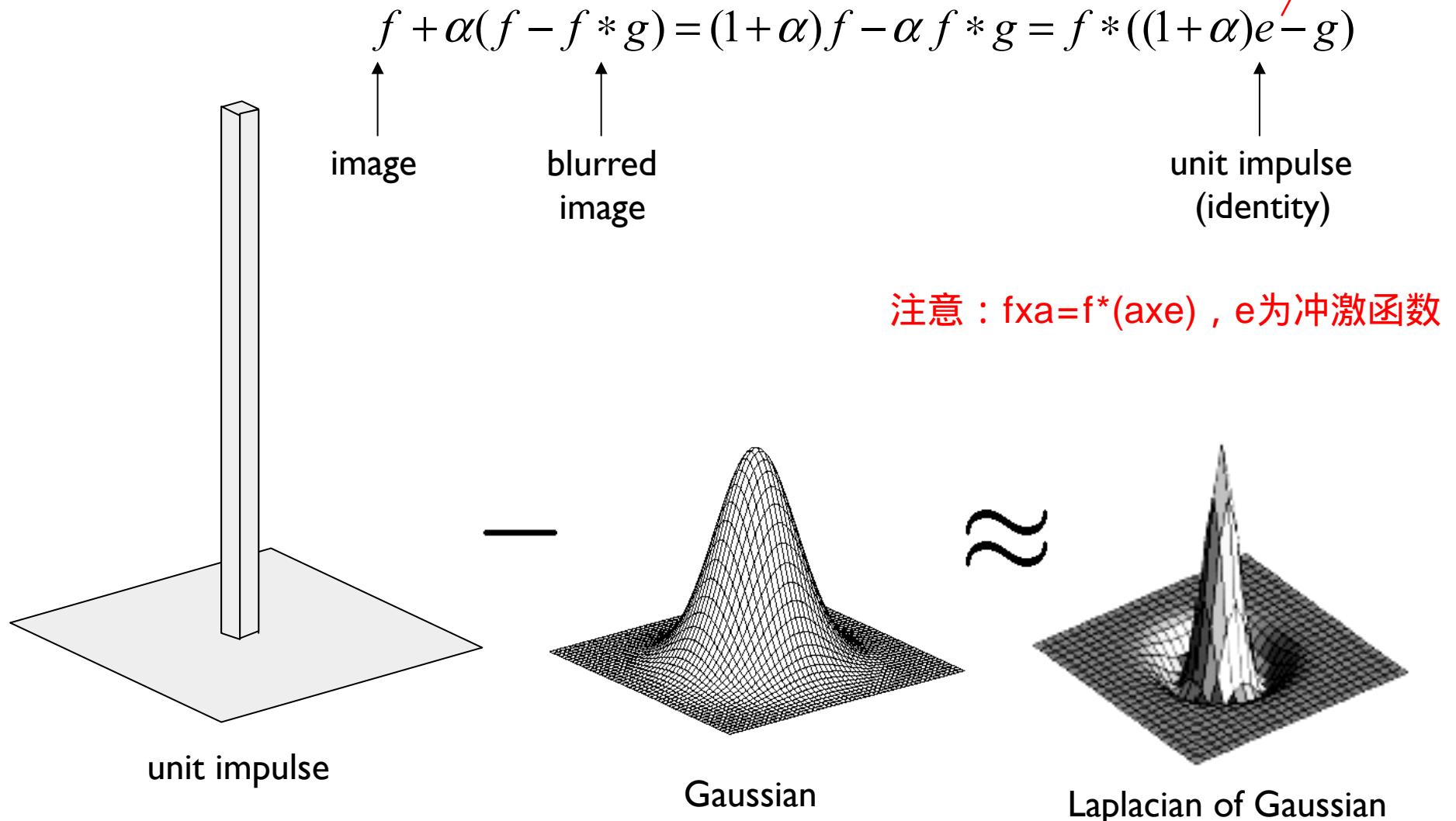


Slide credit: S. Lazebnik

Sharpening through blurring: Unsharp Masking

# Unsharp mask filter

滤波器核为一个冲激I  
减去一个平滑滤波器g  
例如高斯，效果是一个  
Laplacian of Gaussian



Slide credit: S. Lazebnik

# Sharpening using Unsharp Mask Filter



Original



Filtered result

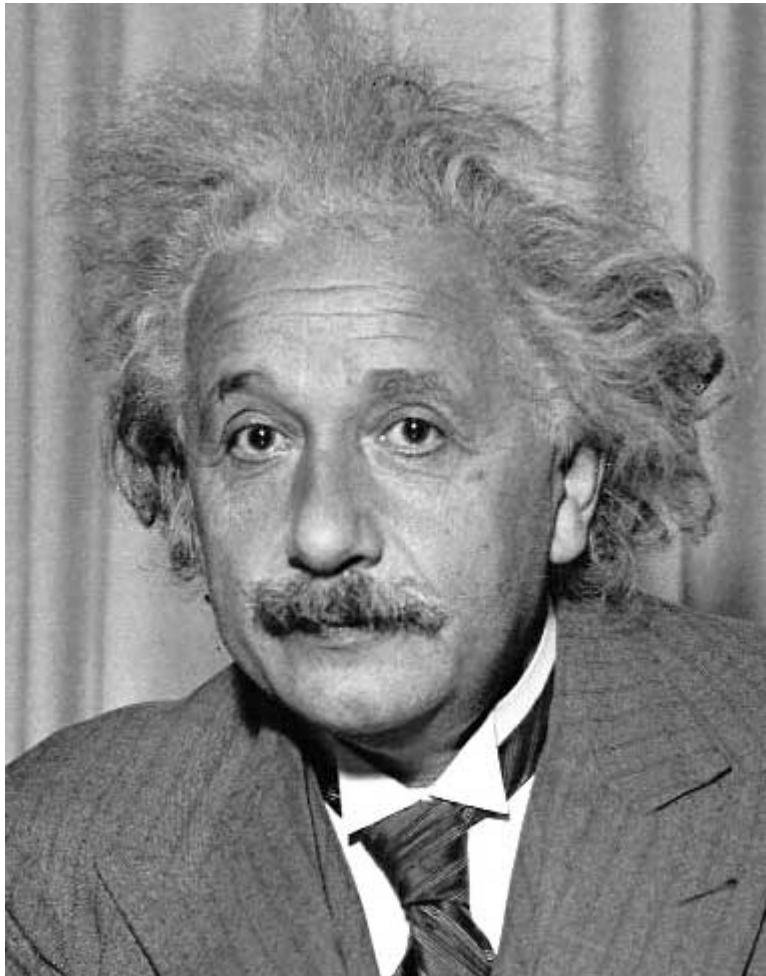
Slide credit: C. Dyer

# Unsharp Masking



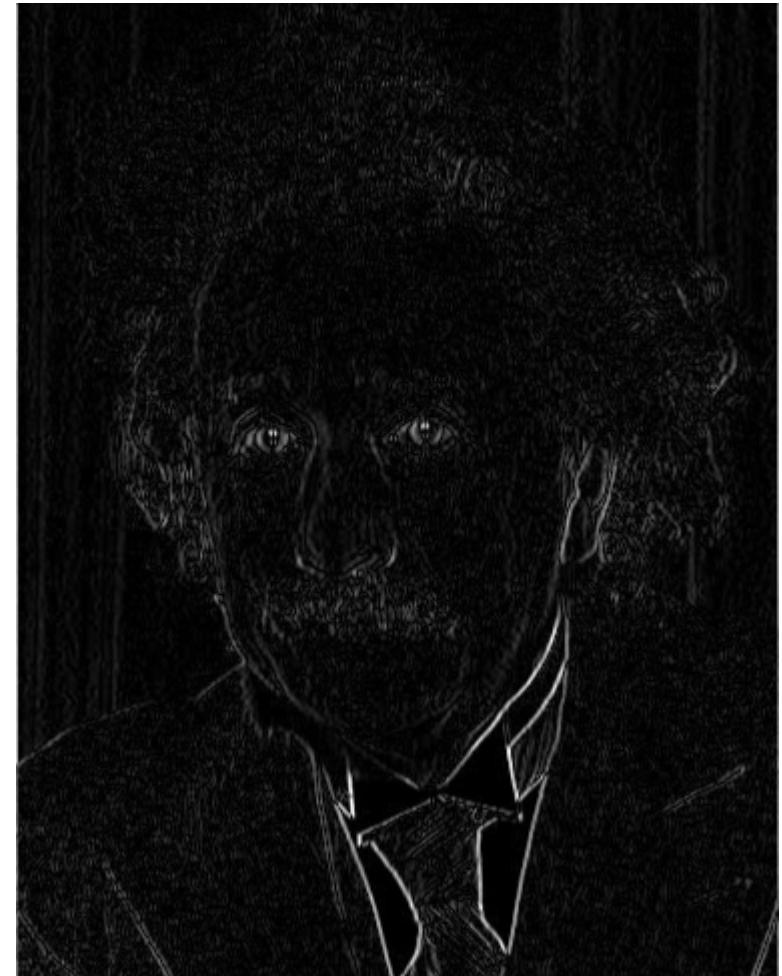
Slide credit: C. Dyer

# Other filters



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

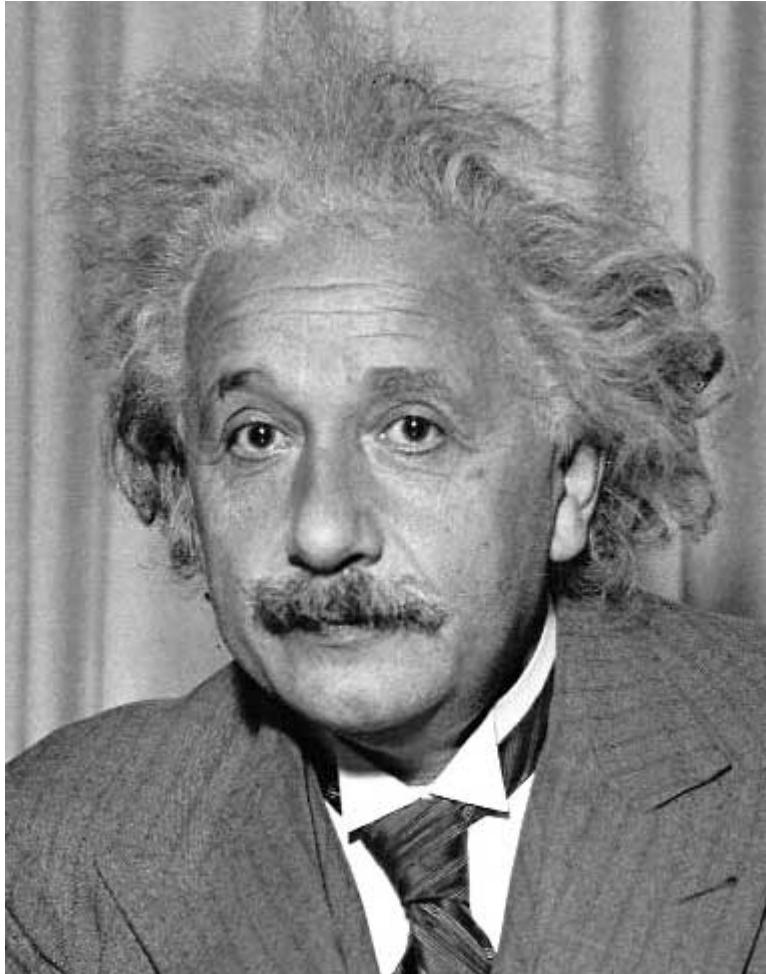
Sobel



Vertical Edge  
(absolute value)

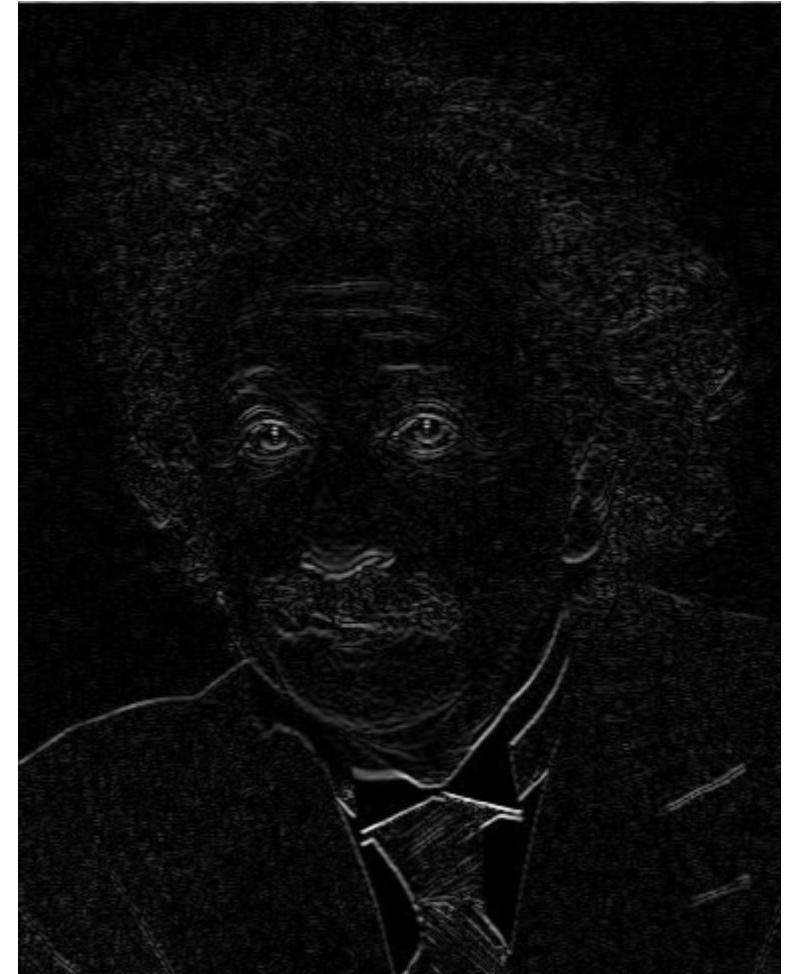
Slide credit: J. Hays

# Other filters



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

Sobel

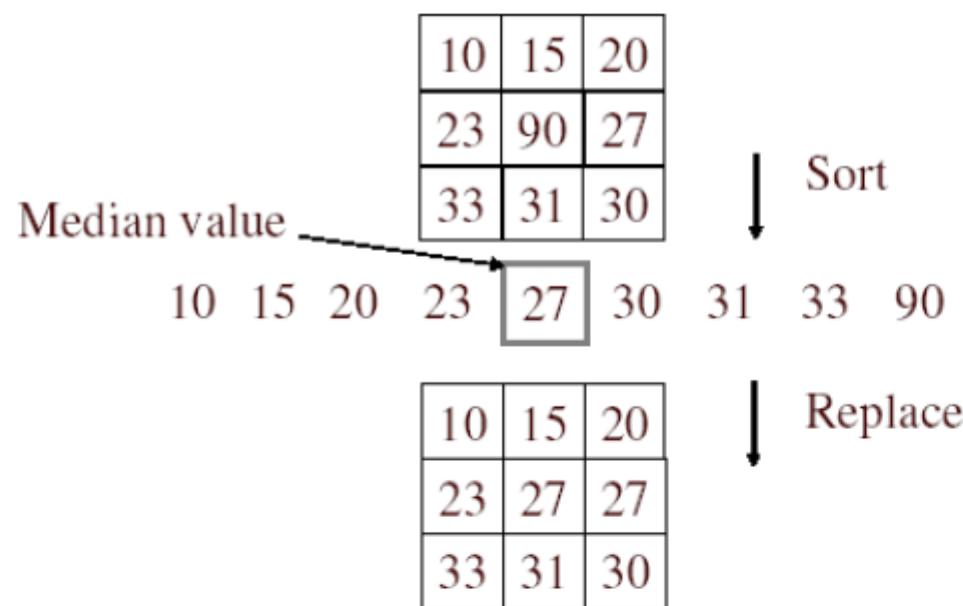


Horizontal Edge  
(absolute value)

# Median filters

- A **Median Filter** operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?
- Is a median filter a kind of convolution?

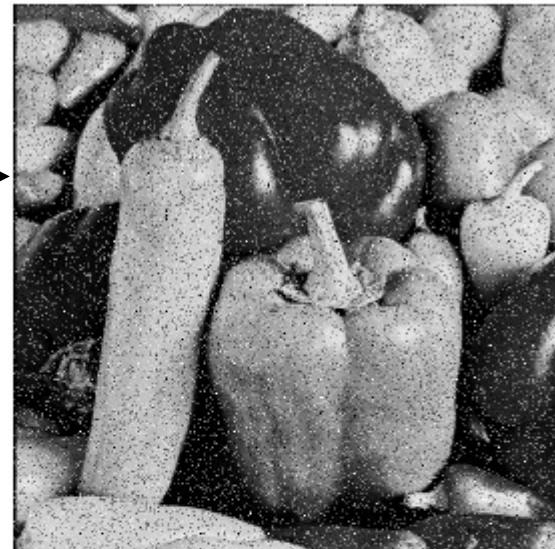
# Median filter



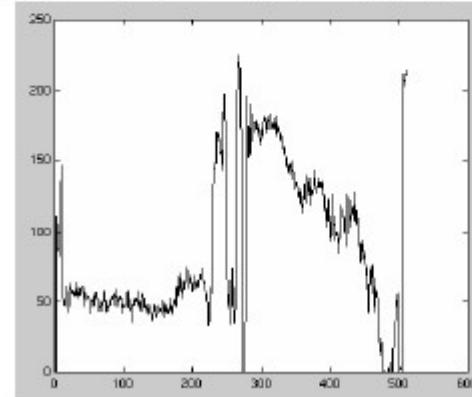
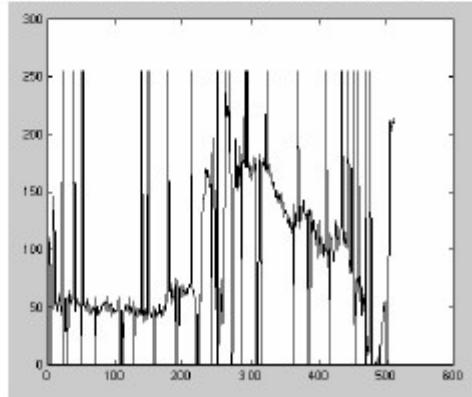
- No new pixel values introduced
- Removes spikes: good for impulse, salt & pepper noise
- Non-linear filter

# Median filter

Salt and  
pepper  
noise



Median  
filtered



Plots of a row of the image

Matlab: output im = medfilt2(im, [h w]);

Slide credit: M. Hebert

# Median filter

- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers
  - Median filter is edge preserving

filters have width 5 :

