BIL 717
Image Processing

Feb. 15, 2016

Linear Filtering
Edge Detection

Erkut Erdem
Hacettepe University
Computer Vision Lab (HUCVL)

Today

- Linear Filtering
  - Review
  - Gauss filter
  - Linear diffusion

- Edge Detection
  - Review
  - Derivative filters
  - Laplacian of Gaussian
  - Canny edge detector

Today

- Linear Filtering
  - Review
  - Gauss filter
  - Linear diffusion

- Edge Detection
  - Review
  - Derivative filters
  - Laplacian of Gaussian
  - Canny edge detector

Filtering

- The name “filter” is borrowed from frequency domain processing
- Accept or reject certain frequency components
- <u>Fourier (1807):</u> Periodic functions could be represented as a weighted sum of sines and cosines

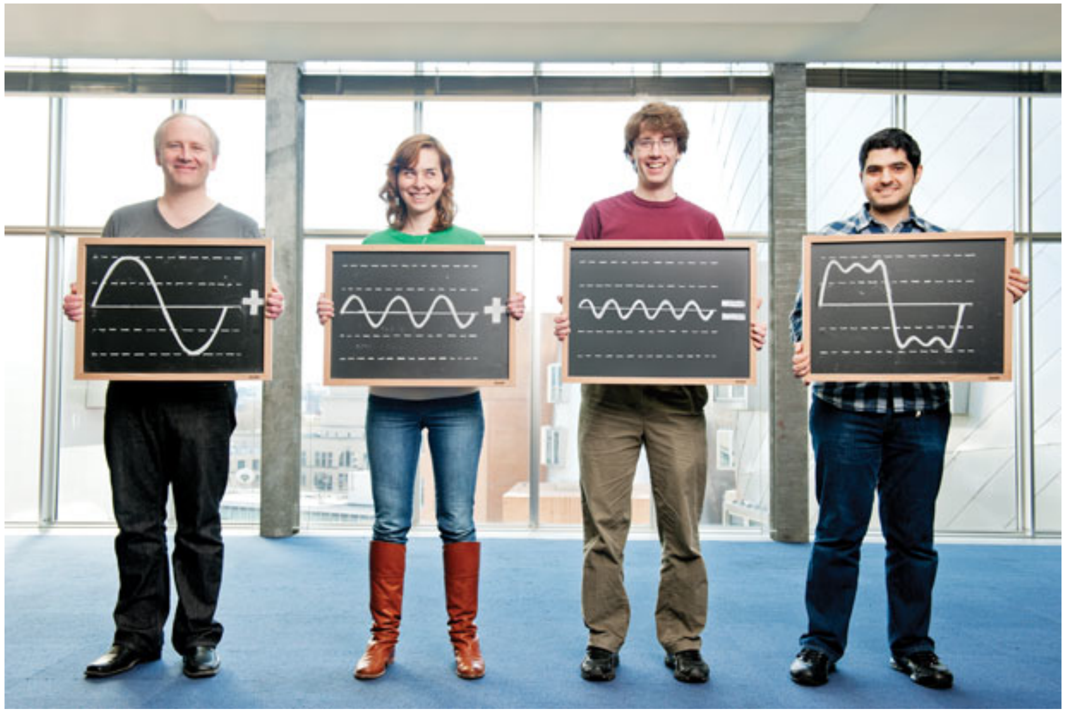
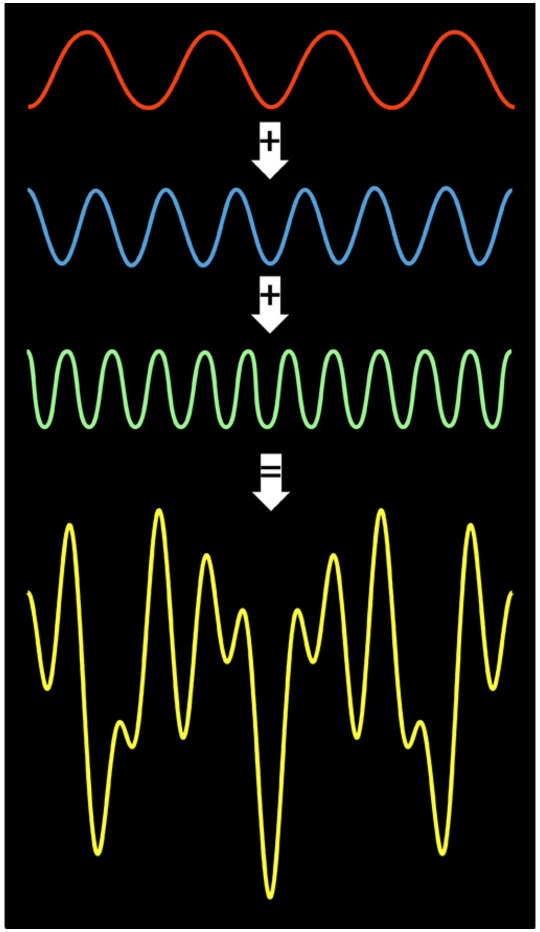


Image courtesy of Technology Review

Signals

- A signal is composed of low and high frequency components



low frequency components: smooth / piecewise smooth

Neighboring pixels have similar brightness values

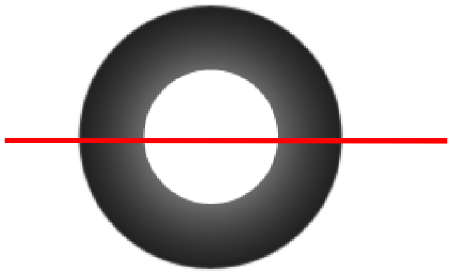
You're within a region

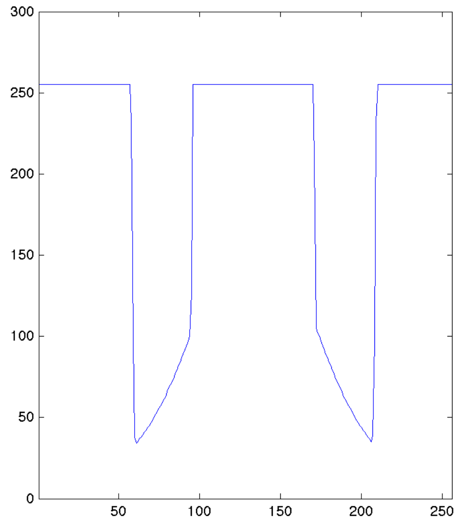
high frequency components: oscillatory

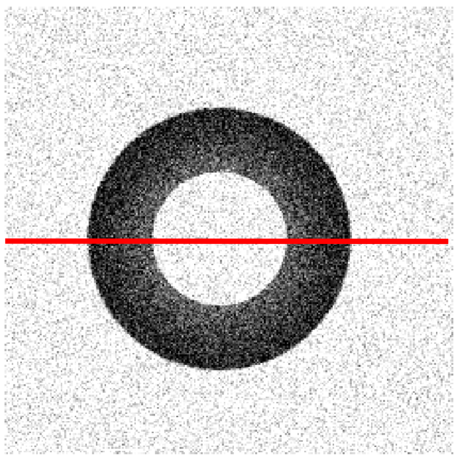
Neighboring pixels have different brightness values

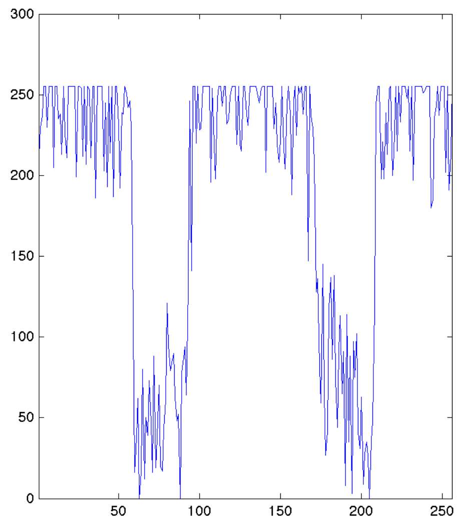
You're either at the edges or noise points

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

↓

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

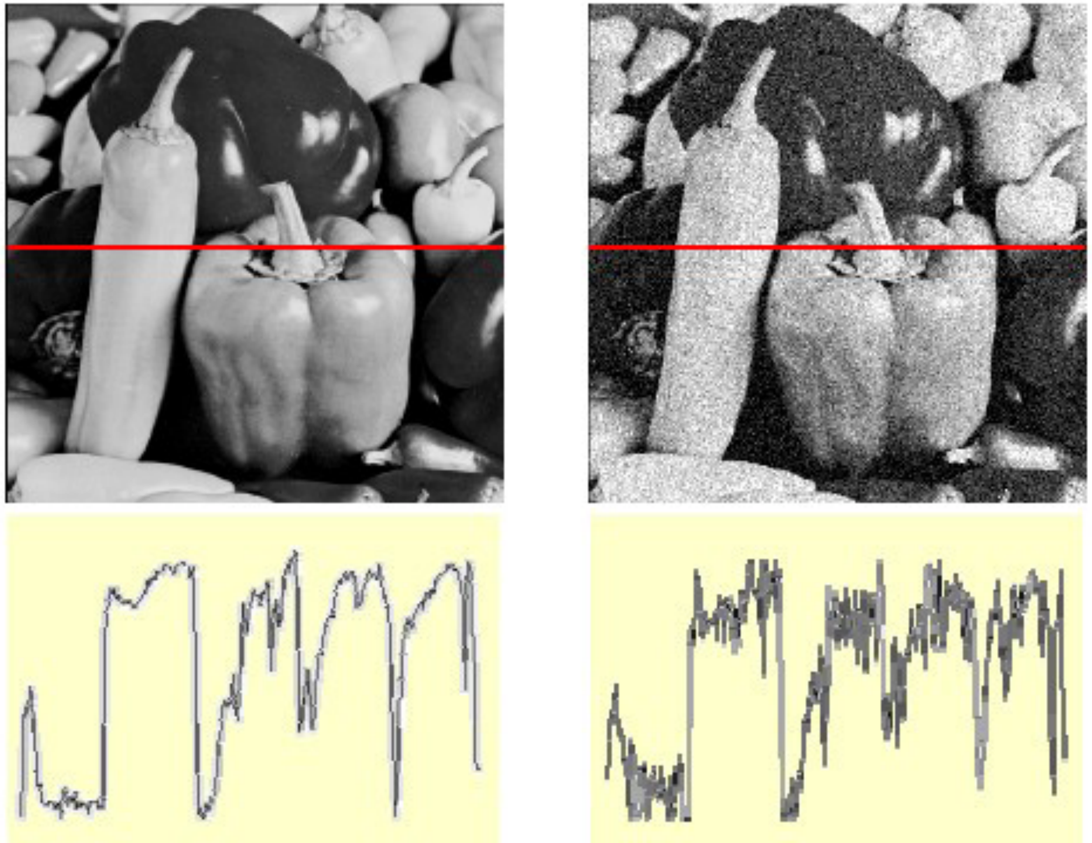
Salt and pepper noise

Impulse noise

Gaussian noise

Slide credit: S. Seitz

Gaussian noise



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

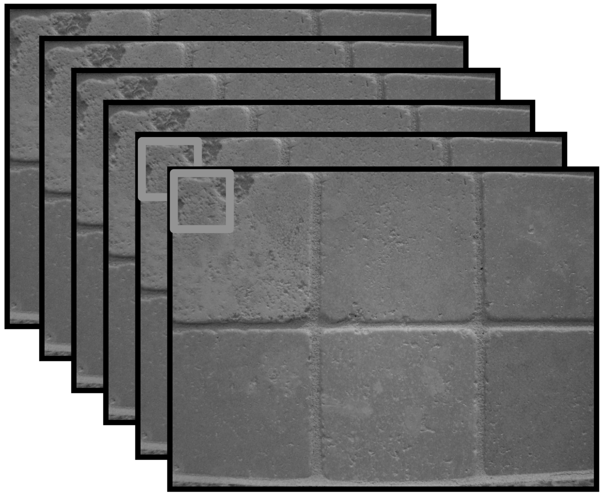
Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim \mathcal{N}(\mu, \sigma)$

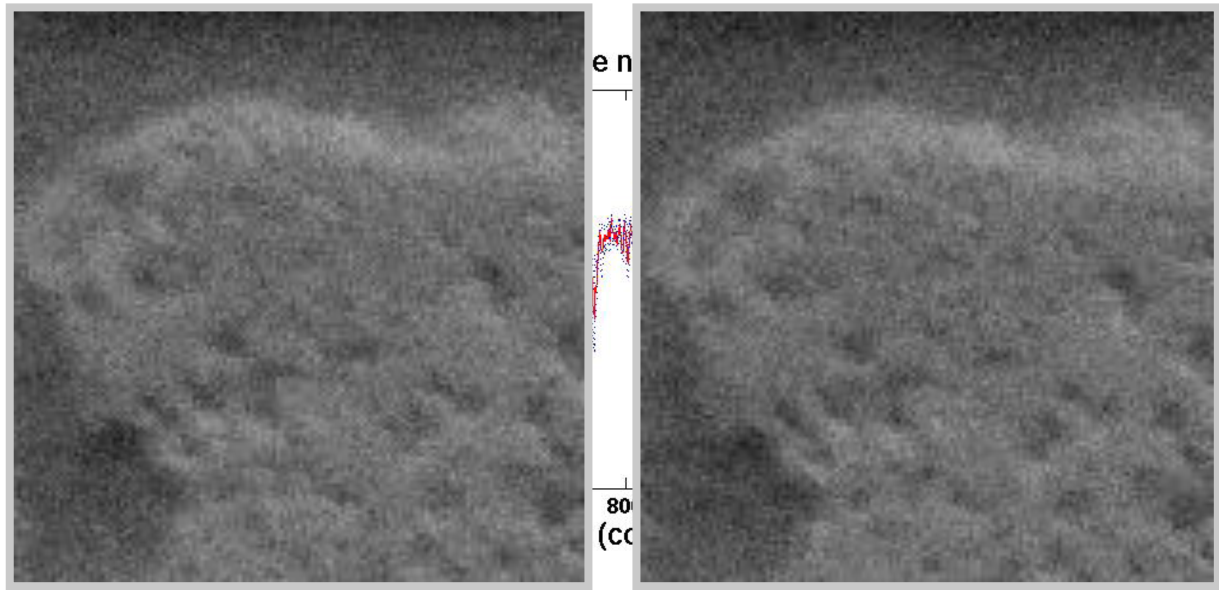
```
>> noise = randn(size(im)).*sigma;
>> output = im + noise;
```

What is the impact of the sigma?

Slide credit: M. Hebert

Motivation: noise reduction

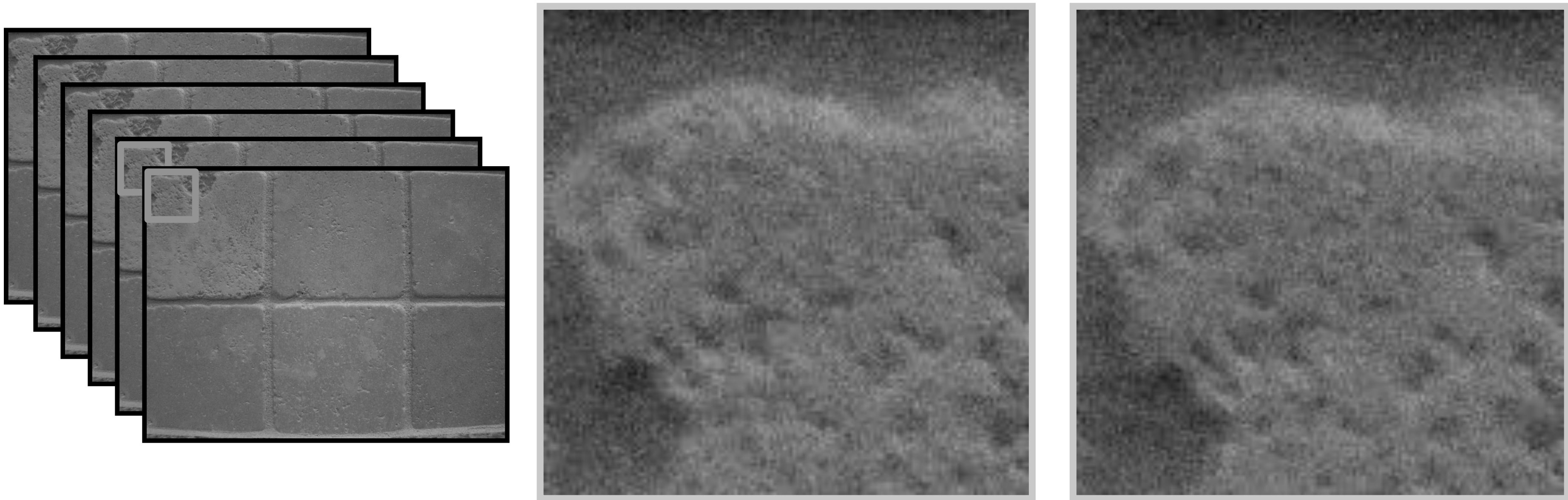




- Make multiple observations of the same <u>static</u> scene
- Take the average
- Even multiple images of the same static scene will not be identical.

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

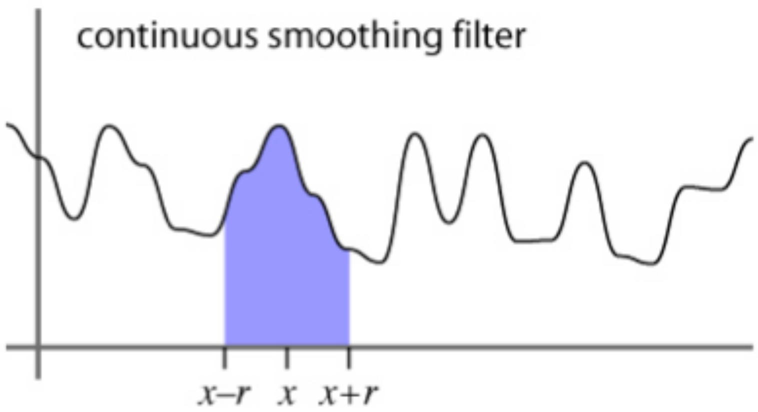
Image Filtering

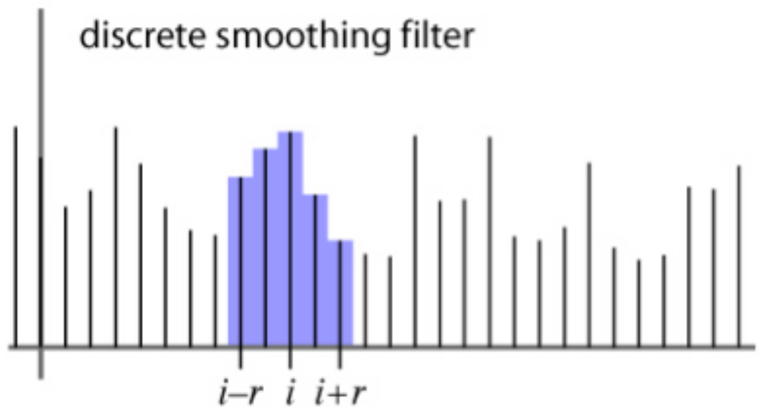
- Idea: Use the information coming from the neighboring pixels for processing
- Design a transformation function of the local neighborhood at each pixel in the image
  - Function specified by a “filter” or mask saying how to combine values from neighbors.
- Various uses of filtering:
  - Enhance an image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

Adapted from: K. Grauman

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





Slide credit: S. Marschner

Linear filtering

- Filtered value is the linear combination of neighboring pixel values.
- Key properties

– linearity: filter($f + g$) = filter(f) + filter(g)

– shift invariance: behavior invariant to shifting the input

  - delaying an audio signal
  - sliding an image around

- Can be modeled mathematically by convolution

Adapted from: S. Marschner

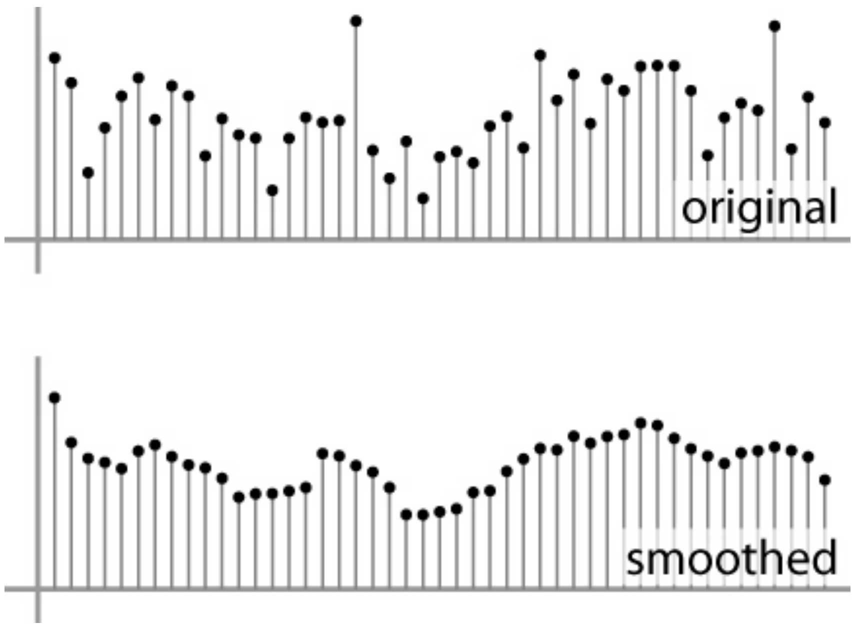
First attempt at a solution

- 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

Slide credit: S. Marschner, K. Grauman

First attempt at a solution

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



Slide credit: S. Marschner

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**

Slide credit: S. Marschner

Filters

- Sequence of weights $a[j]$ is called a *filter*
- Filter is nonzero over its *region of support*

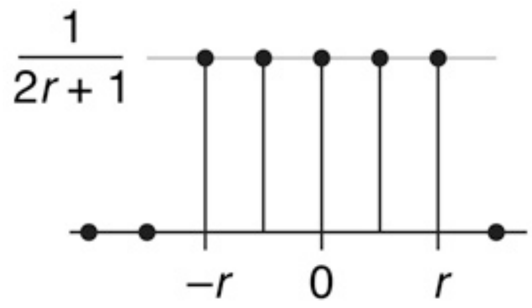
– usually centered on zero: support radius r

- 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

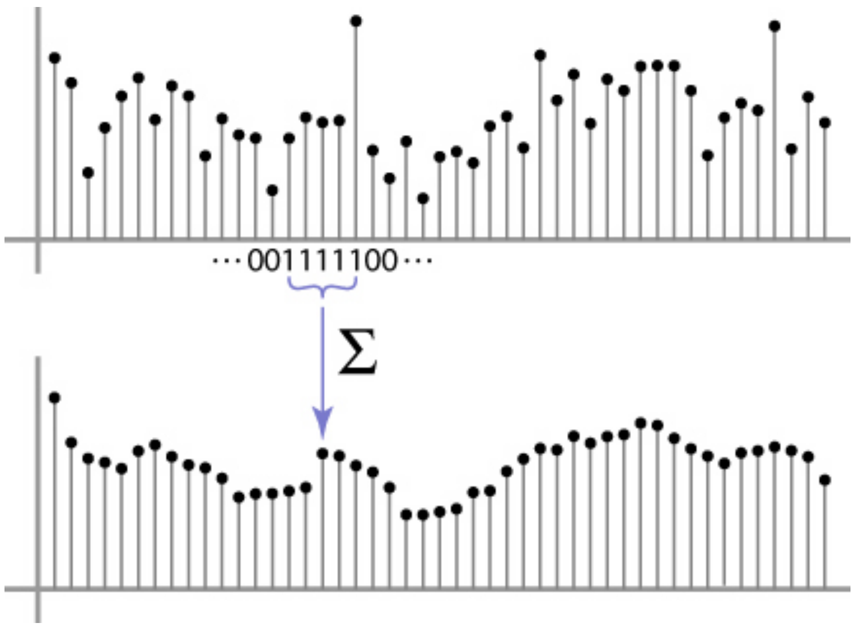


a box filter

Slide credit: S. Marschner

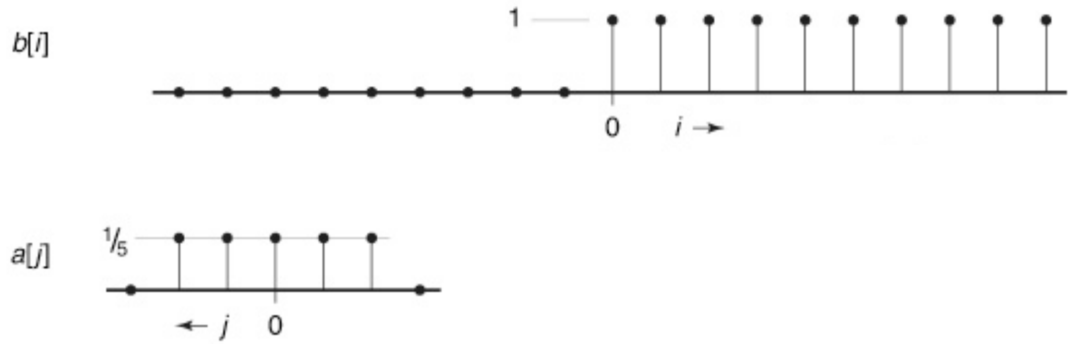
Convolution and filtering

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



Slide credit: S. Marschner

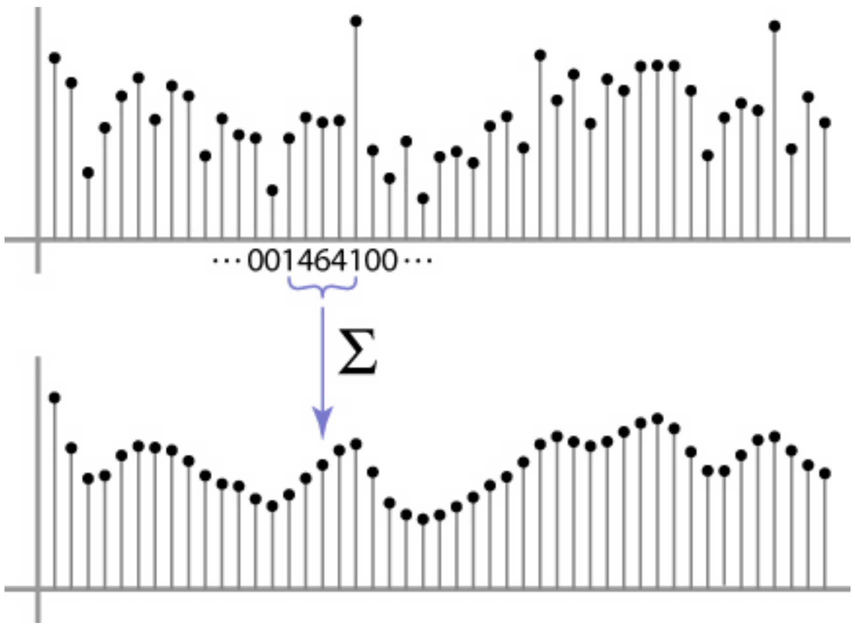
Example: box and step



Slide credit: S. Marschner

Convolution and filtering

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



Slide credit: S. Marschner

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

Slide credit: S. Marschner

Key properties

- **Linearity:** filter($f_1 + f_2$) = filter(f_1) + filter(f_2)
- **Shift invariance:** filter(shift(f)) = shift(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

Slide credit: S. Lazebnik

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 $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$,
  $a * e = a$

Slide credit: S. Lazebnik

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: $a * (b_1 * b_2 * b_3)$

Slide credit: S. Marschner

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

Slide credit: S. Marschner

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								

Slide credit: S. Seitz

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							

Slide credit: S. Seitz

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	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	

Slide credit: S. Seitz

Averaging filter

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

$F[x, y]$

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

$\otimes$

$H[u, v]$

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

"box filter"

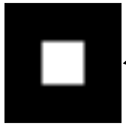
$G[x, y]$

	0	10	20	30	30				

$$G = H \otimes F$$

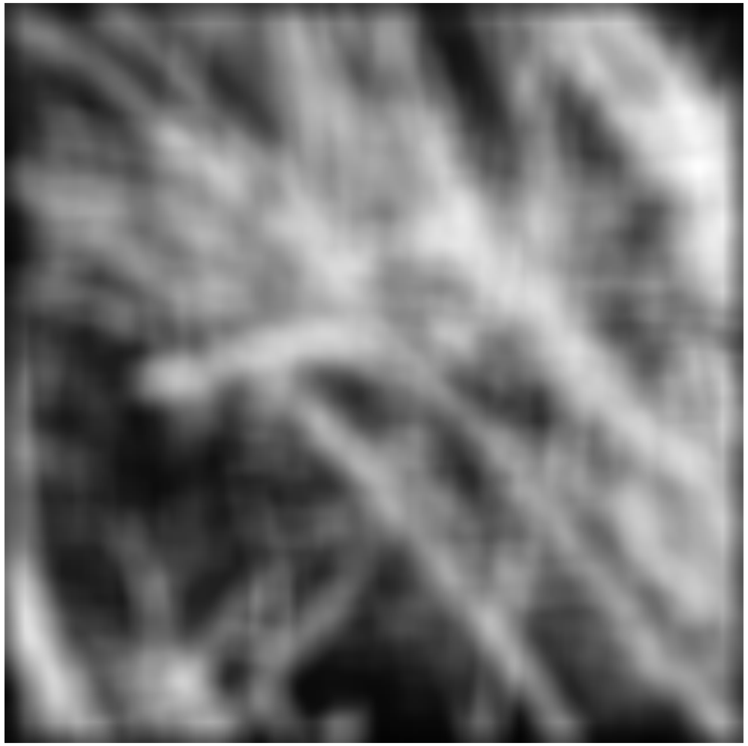
Slide credit: K. Grauman

Smoothing by averaging



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

original



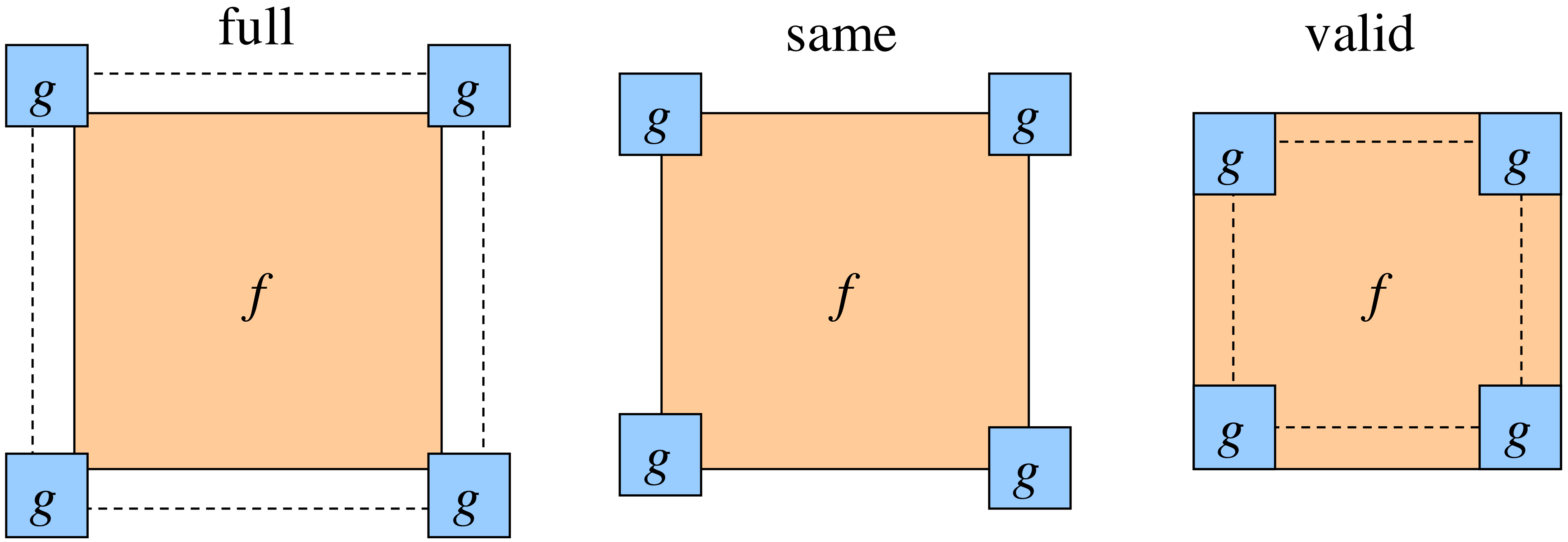
filtered

What if the filter size was 5 x 5 instead of 3 x 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’)`

Slide credit: S. Marschner

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	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

$F[x, y]$

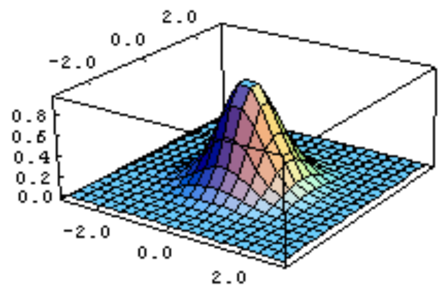
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$H[u, v]$

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

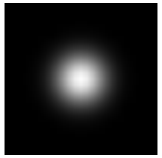
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

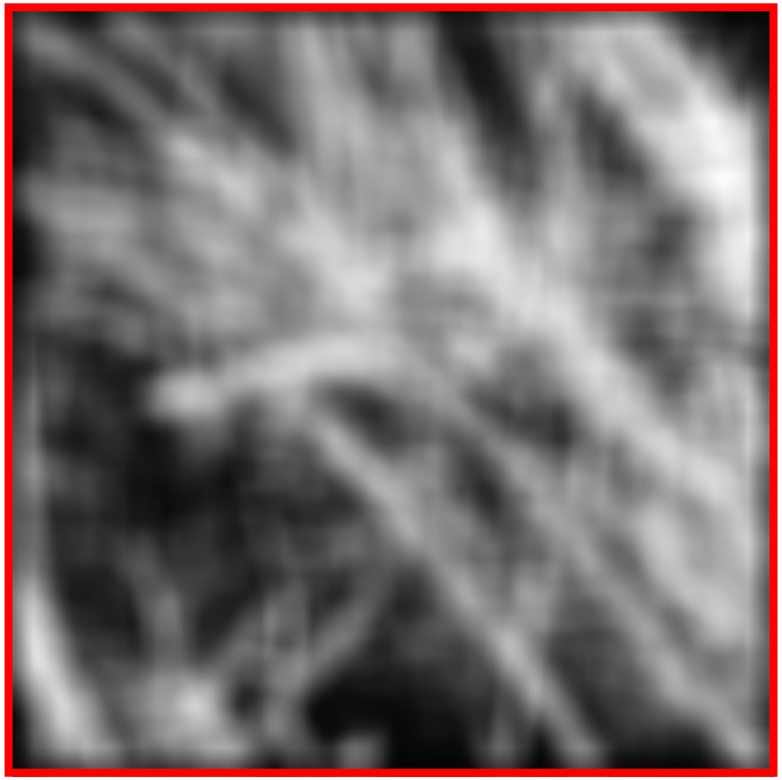


- Removes high-frequency components from the image ("low-pass filter").

Slide credit: S. Seitz

Smoothing with a Gaussian

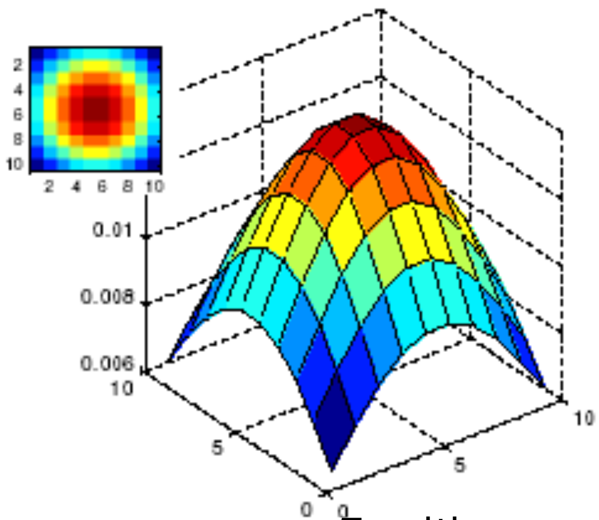




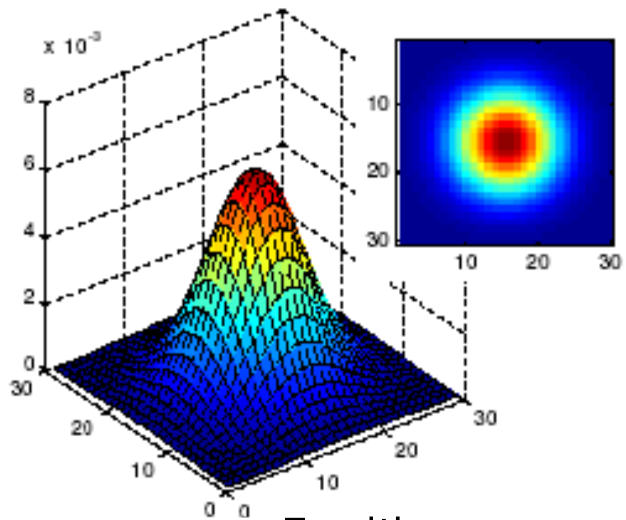
Slide credit: K. Grauman

Gaussian filters

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



σ = 5 with
10 x 10 kernel

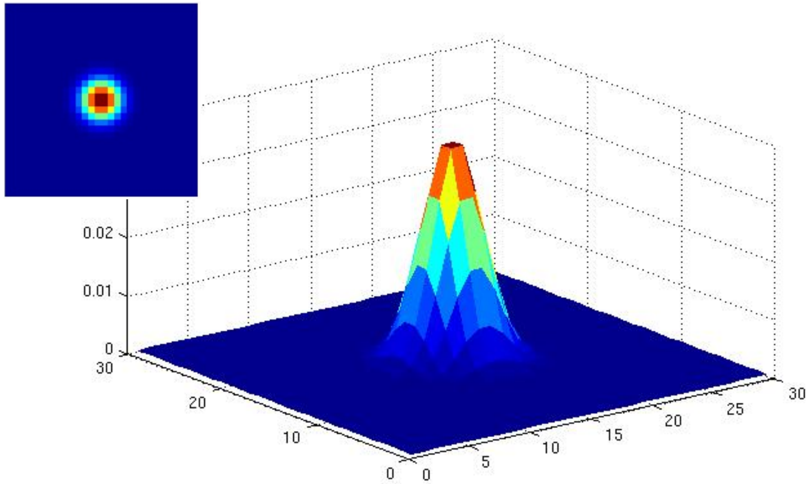


σ = 5 with
30 x 30 kernel

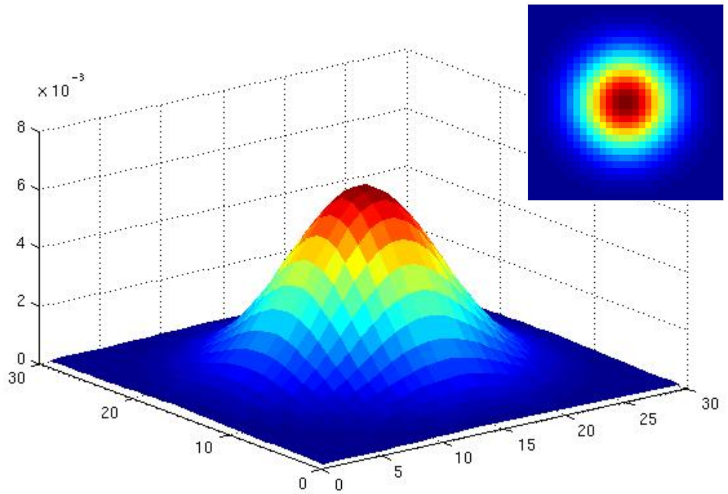
Slide credit: K. Grauman

Gaussian filters

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



σ = 2 with
30 x 30 kernel



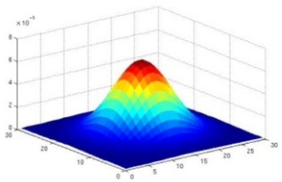
σ = 5 with
30 x 30 kernel

Slide credit: K. Grauman

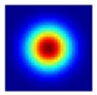
Matlab

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

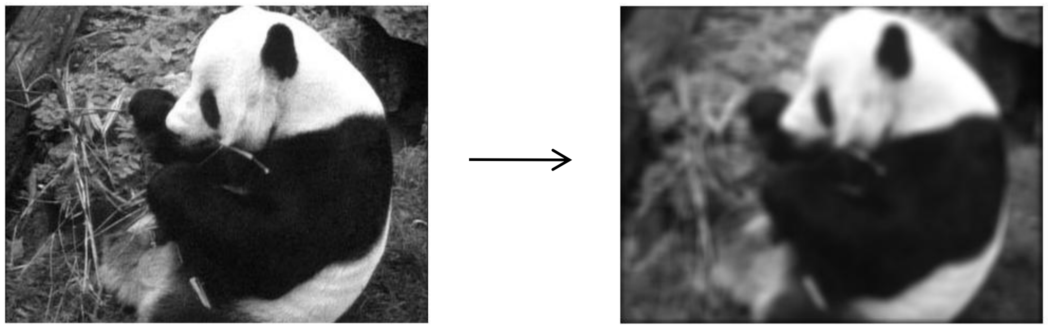
```
>> mesh(h);
```



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



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

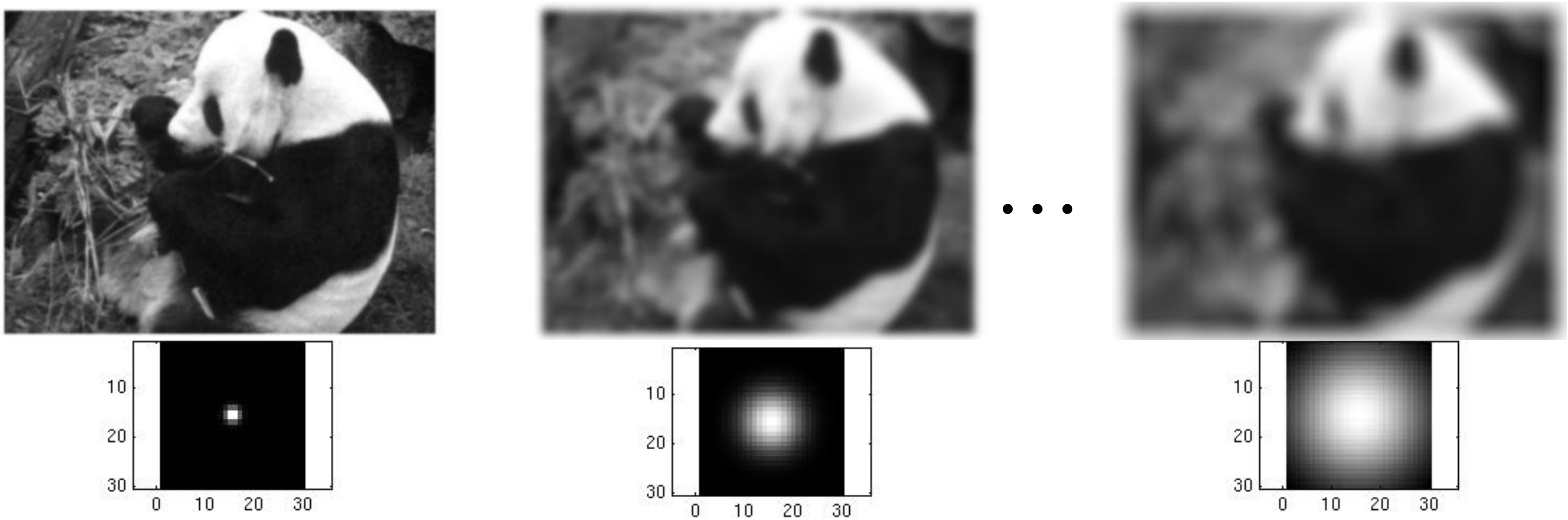


outim

Slide credit: K. Grauman

Smoothing with a Gaussian

Parameter σ 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

Properties of smoothing filters

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

Slide credit: K. Grauman

Linear Diffusion

- Let $f(x)$ denote a grayscale (noisy) input image and $u(x, t)$ be initialized with $u(x,0) = u^0(x) = f(x)$.
- The linear diffusion process can be defined by the equation:

$$\frac{\partial u}{\partial t} = \nabla \cdot (\nabla u) = \nabla^2 u$$

where $\nabla \cdot$ denotes the divergence operator. Thus,

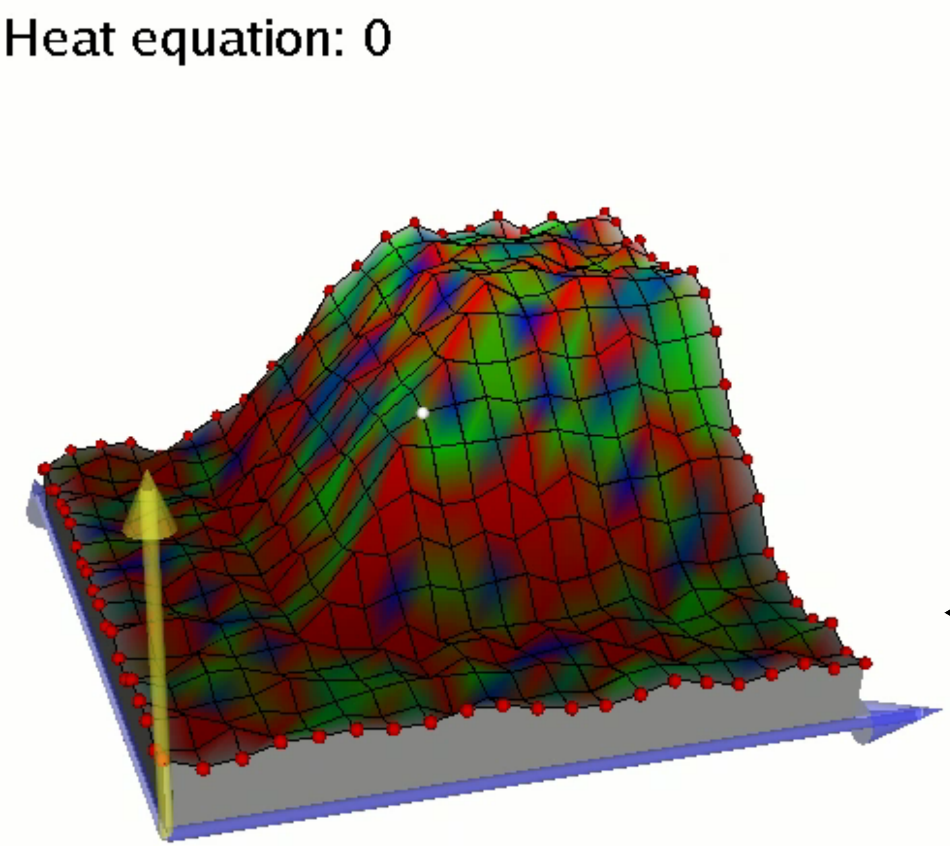
$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Linear Diffusion (cont'd.)

- Diffusion process as an evolution process.
- Artificial time variable t denotes the *diffusion time*
- Input image is smoothed at a constant rate in all directions.
  - $u^0(x)$: initial image,
  - $u(x, t)$: the evolving images under the governed equation representing the successively smoothed versions of the initial input image $f(x)$.
- Diffusion process creates a *scale space* representation of the given image f, with $t > 0$ being the scale.

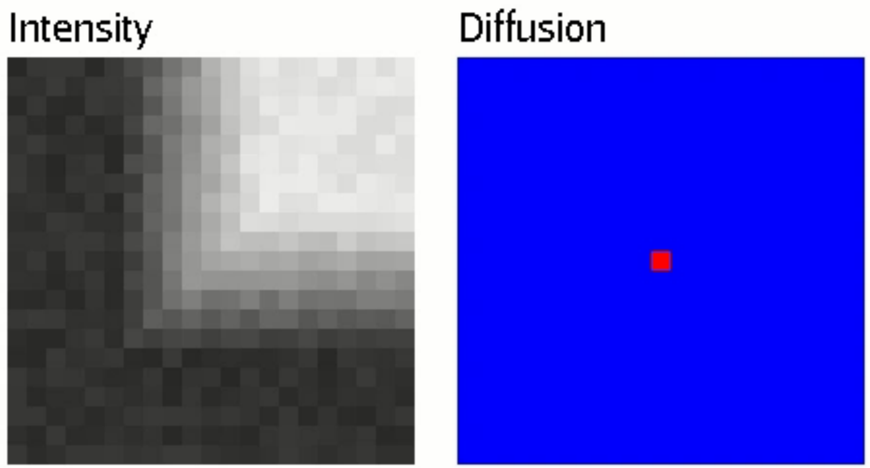
Linear Diffusion (cont'd.)

$$\frac{\partial u}{\partial t} = \nabla \cdot (\nabla u) = \nabla^2 u$$



red: active areas
blue: inactive area

gray-level image



influence of the central pixel on the other pixels (red: high, blue: low)

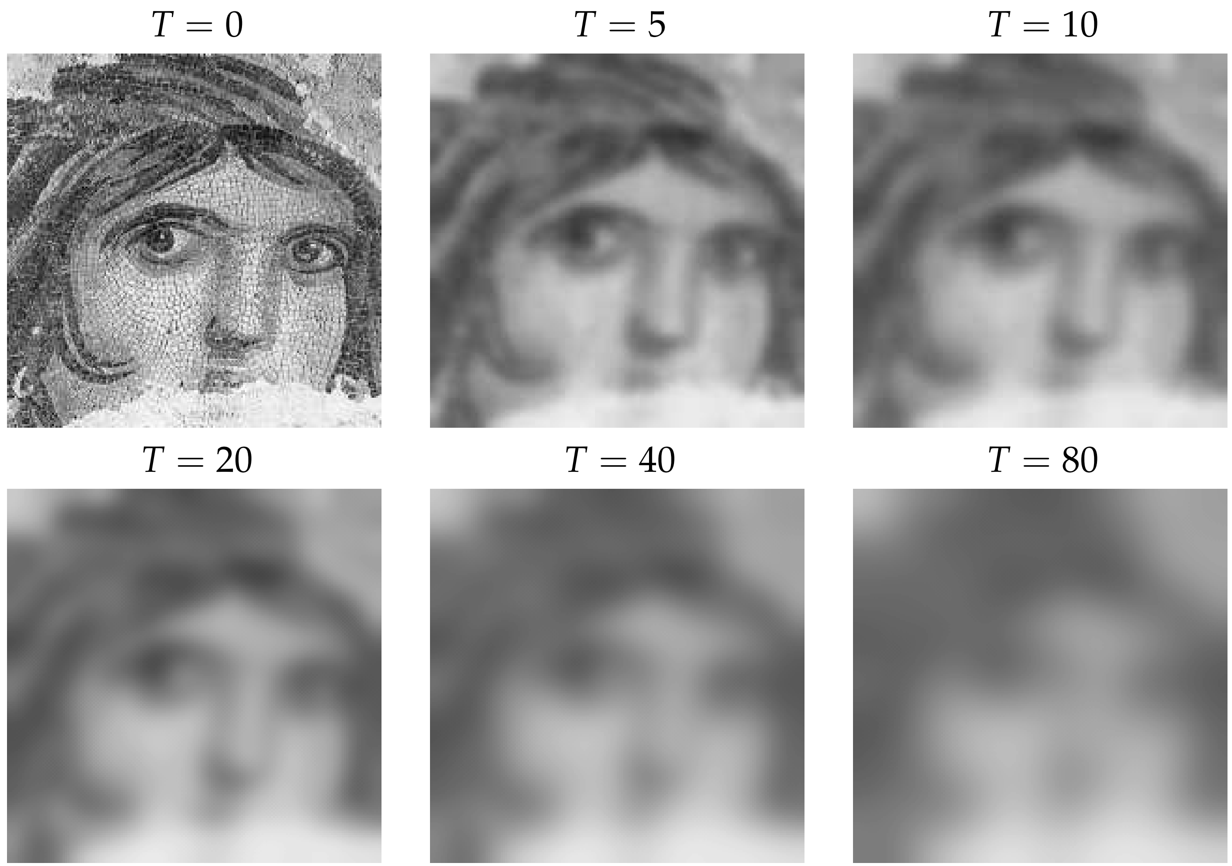
Credit: S. Paris

Linear Diffusion (cont'd.)

- As we move to coarser scales,
  - Evolving images become more and more simplified
  - Diffusion process removes the image structures at finer scales.

Linear Diffusion (cont'd.)

- As we move to coarser scales,
  - Evolving images become more and more simplified
  - Diffusion process removes the image structures at finer scales.



Linear Diffusion and Gaussian Filtering

- The solution of the linear diffusion can be explicitly estimated as:

$$u(x,T) = \left(G_{\sqrt{2T}} * f\right)(x)$$

with $G_\sigma(x) = \frac{1}{2\pi\sigma^2} exp\left(-\frac{|x|^2}{2\sigma^2}\right)$

- Solution of the linear diffusion equation is equivalent to a proper convolution of the input image with the Gaussian kernel $G_\sigma(x)$ with standard deviation $\sigma = \sqrt{2T}$
- The higher the value of T, the higher the value of σ, and the more smooth the image becomes.

Numerical Implementation

- Solving the linear diffusion equation requires discretization in both spatial and time coordinates.
- Central differences for the spatial derivatives:

$$\frac{d^2 u_{i,j}}{dx^2} \approx \frac{u_{i+h_x,j} - 2u_{i,j} + u_{i-h_x,j}}{h_x^2}$$

$$\frac{d^2 u_{i,j}}{dy^2} \approx \frac{u_{i,j+h_y} - 2u_{i,j} + u_{i,j-h_y}}{h_y^2}$$

where $u_{i,j}$ denotes the gray value or the brightness of the evolving image at pixel location (i, j).

- We take $h_x = h_y = 1$ for a regular grid.

Numerical Implementation (cont'd.)

- Original model:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

- Space discrete version:

$$\frac{du_{i,j}}{dt} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}$$

- Space-time discrete version:

$$\frac{u_{i,j}^{k+1} - u_{i,j}^{k}}{\Delta t} = u_{i+1,j}^{k} + u_{i-1,j}^{k} + u_{i,j+1}^{k} + u_{i,j-1}^{k} - 4u_{i,j}^{k}$$

homogeneous Neumann boundary condition along the image boundary

$\Delta t \leq 0.25$ is required for numerical stability

Variational Regularization

- Variational regularization models formulate smoothing process as a functional minimization via which a noise-free approximation of a given image is to be estimated.
- With an additive model, $f(x) = u(x) + n(x)$
  - *f(x): original image*
  - *u(x): smoothed image*
  - *n(x): noise component*
- An example: Tikhonov energy functional

$$E(u) = \int_{\Omega} \left((u - f)^2 + \alpha |\nabla u|^2 \right) dx$$

Tikhonov energy functional

$$E(u) = \int_{\Omega} \Big(\underbrace{(u-f)^2}_{\text{data fidelity term}} + \underbrace{\alpha |\nabla u|^2}_{\text{regularization term}} \Big)\, dx$$

- $\Omega \subset \mathbf{R}^2$ is connected, bounded, open subset representing the image domain,
- f is an image defined on Ω,
- u is the smooth approximation of f ,
- $\alpha > 0$ is the scale parameter.

Variational Regularization and Diffusion Equations

- A strong relation between variational regularization methods and diffusion equations.
- The minimizing function *u* of the Tikhonov energy functional formally satisfies the Euler-Lagrange equation:

$$(u - f) - \alpha \nabla^2 u = 0$$

with the Neumann boundary condition $\left.\frac{\partial u}{\partial n}\right|_{\partial\Omega} = 0$

- can be rewritten as:

$$\frac{u - u^0}{\alpha} = \nabla^2 u \qquad \text{with} \qquad u^0 = f$$

implicit time discretization of the linear diffusion equation with a single time step ($T = \alpha$)

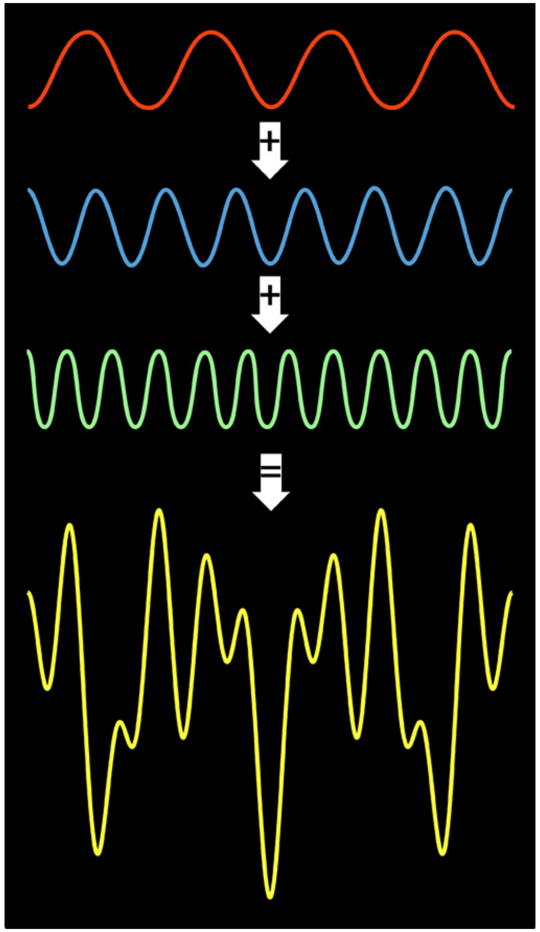
Today

- Linear Filtering
  - Review
  - Gauss filter
  - Linear diffusion

- Edge Detection
  - Review
  - Derivative filters
  - Laplacian of Gaussian
  - Canny edge detector

Signals and Images

- 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

Edge detection

- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels

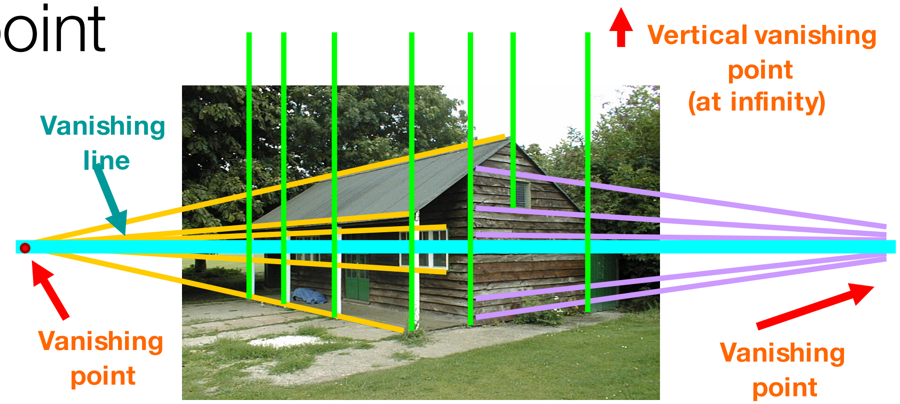
- **Ideal:** artist's line drawing (but artist is also using object-level knowledge)

Slide credit: D. Lowe

Why do we care about edges?

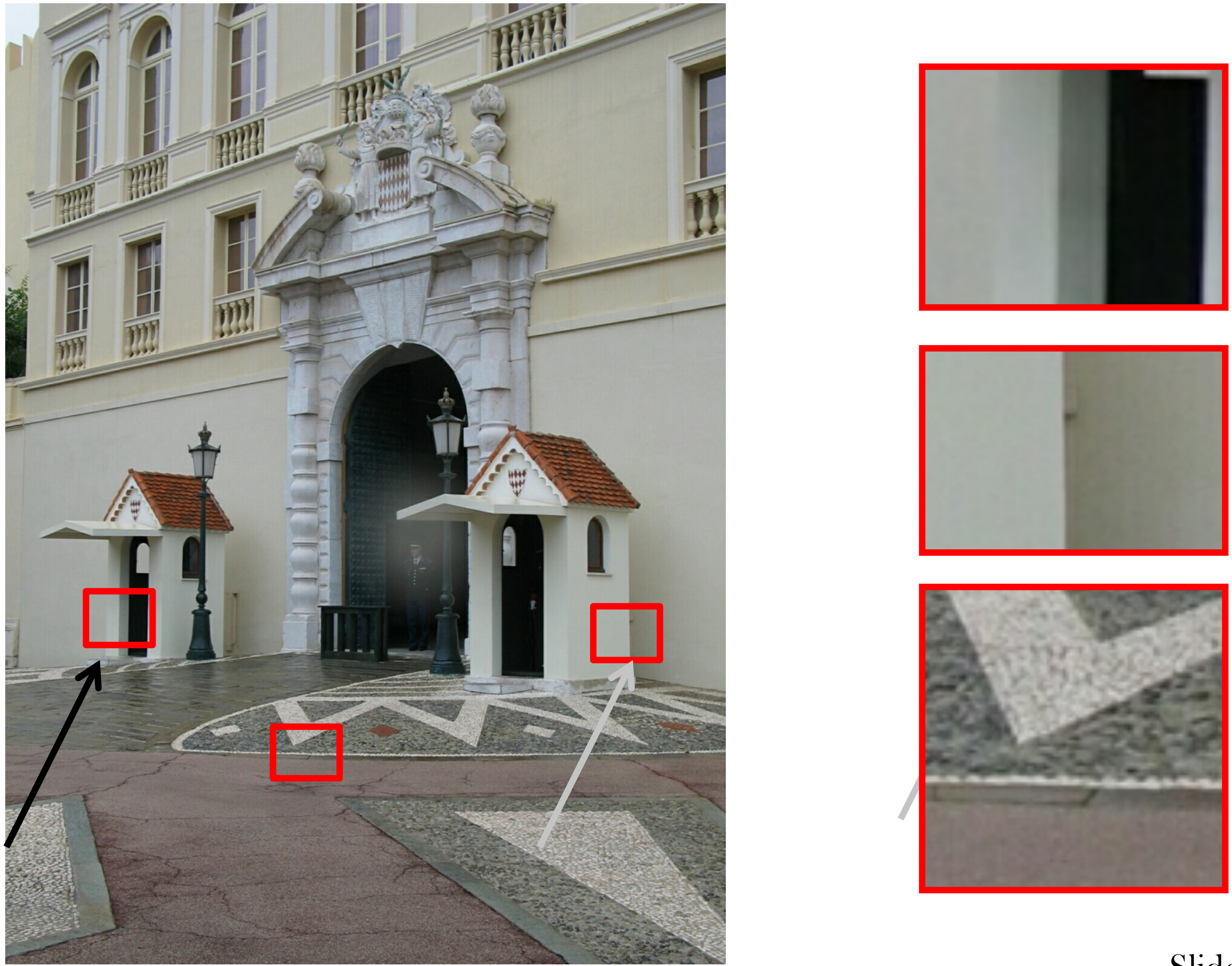
- Extract information, recognize objects

- Recover geometry and viewpoint



Source: J. Hays

Closeup of edges

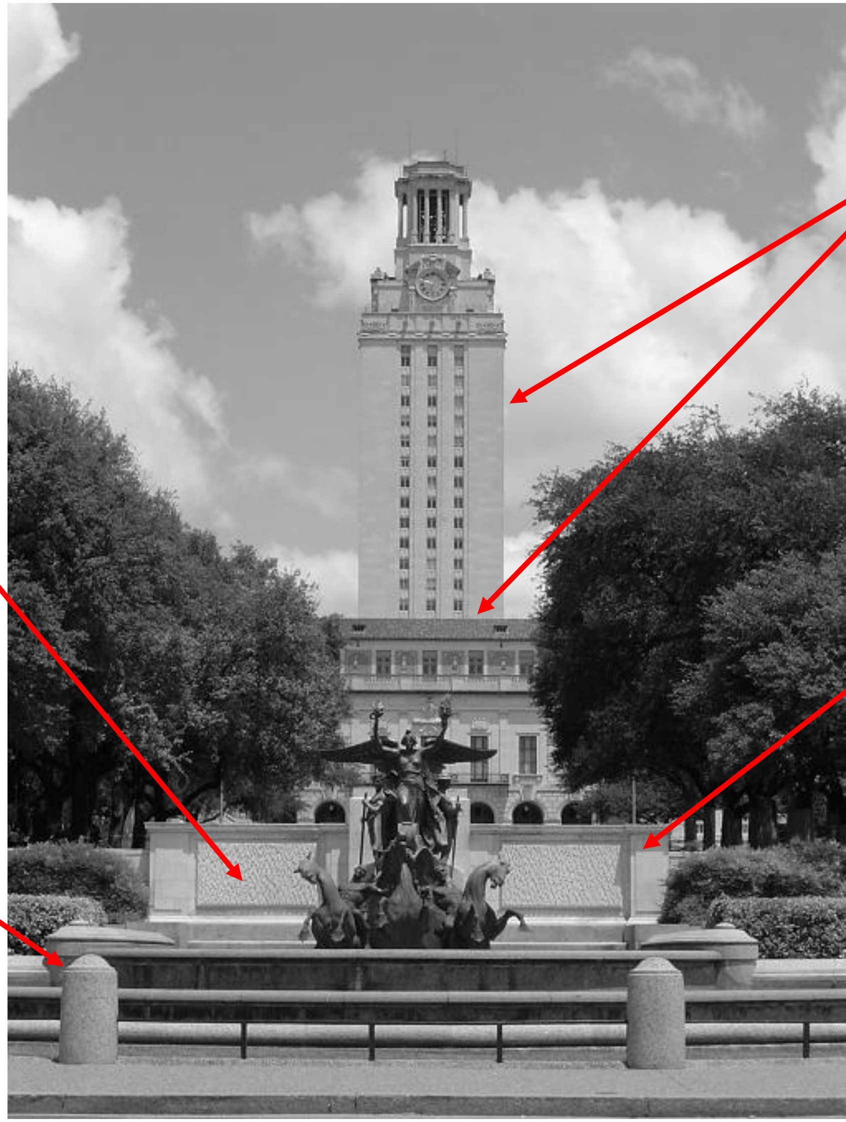


Slide credit: D. Hoiem

What causes an edge?

Reflectance change: appearance information, texture

Change in surface orientation: shape



Depth discontinuity: object boundary

Cast shadows

Slide credit: K. Grauman

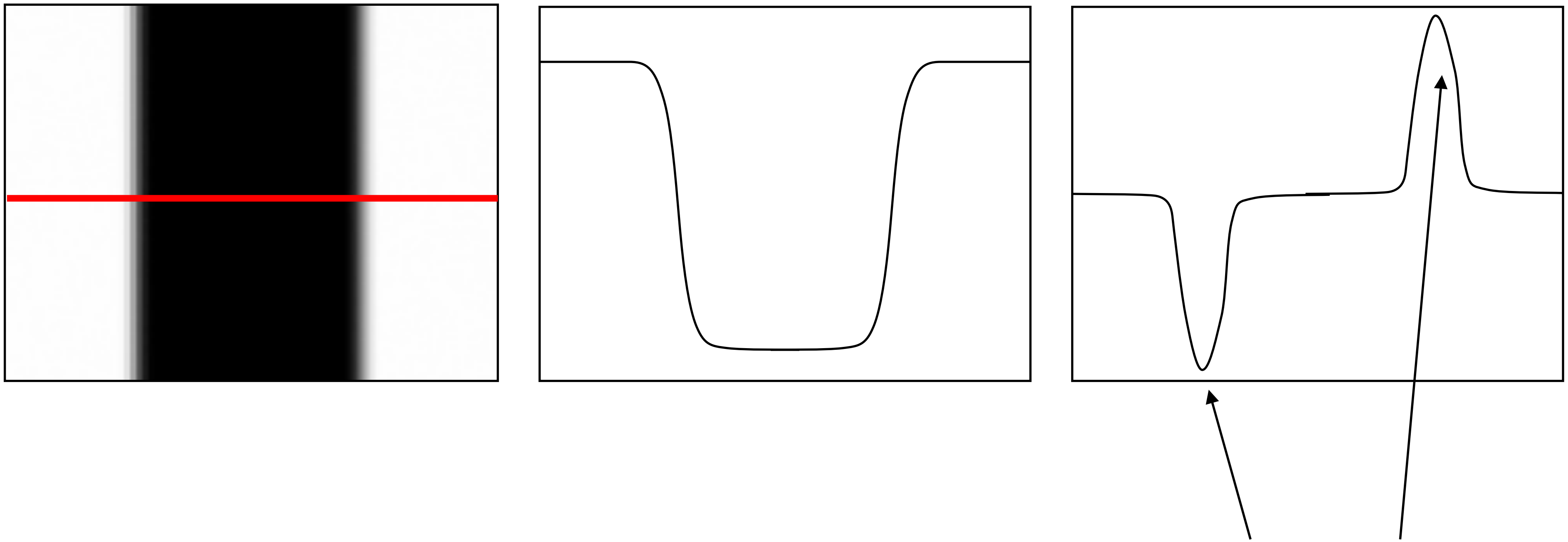
Characterizing edges

- An edge is a place of rapid change in the image intensity function

image

intensity function (along horizontal scanline)

first derivative



edges correspond to extrema of derivative

Slide credit: K. Grauman

Derivatives with convolution

For 2D function f(x,y), the partial derivative is:

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

For discrete data, we can approximate using finite differences:

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

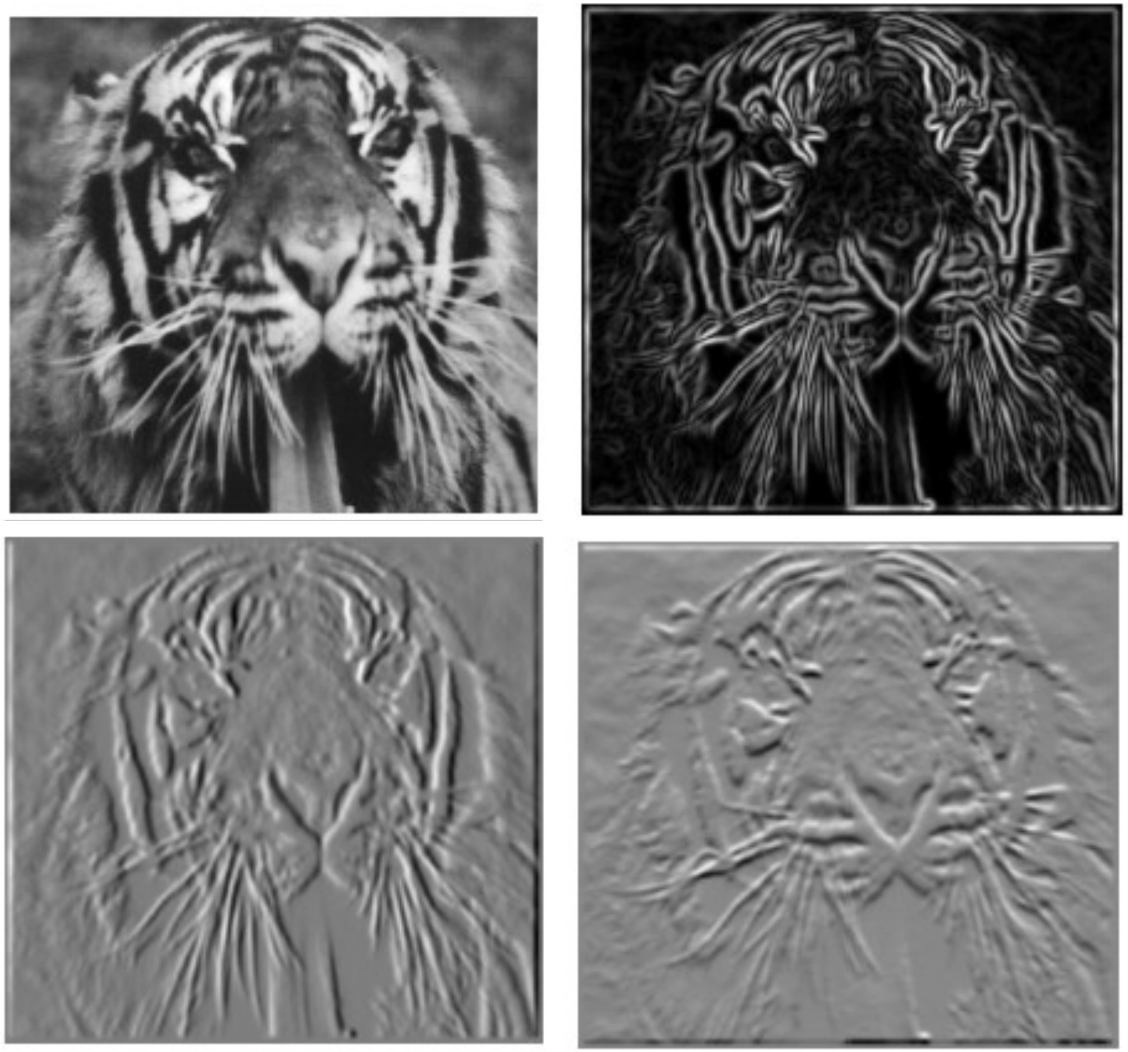
To implement above as convolution, what would be the associated filter?

Slide credit: K. Grauman

Partial derivatives of an image

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

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



-1	1

-1
1

Which shows changes with respect to x?

Slide credit: K. Grauman

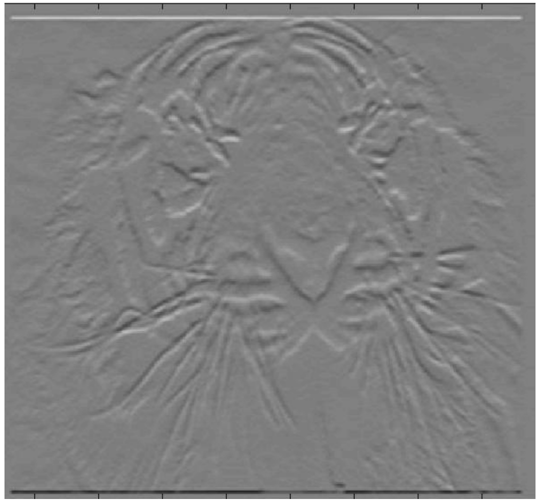
Assorted finite difference filters

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

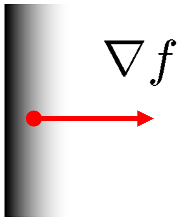
```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```



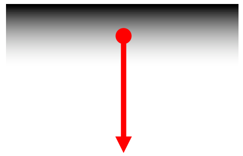
Slide credit: K. Grauman

Image gradient

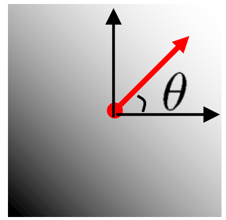
- The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

- 

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



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



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

The gradient points in the direction of most rapid increase in intensity

- How does this direction relate to the direction of the edge?

The gradient direction is given by $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

The *edge strength* is given by the gradient magnitude

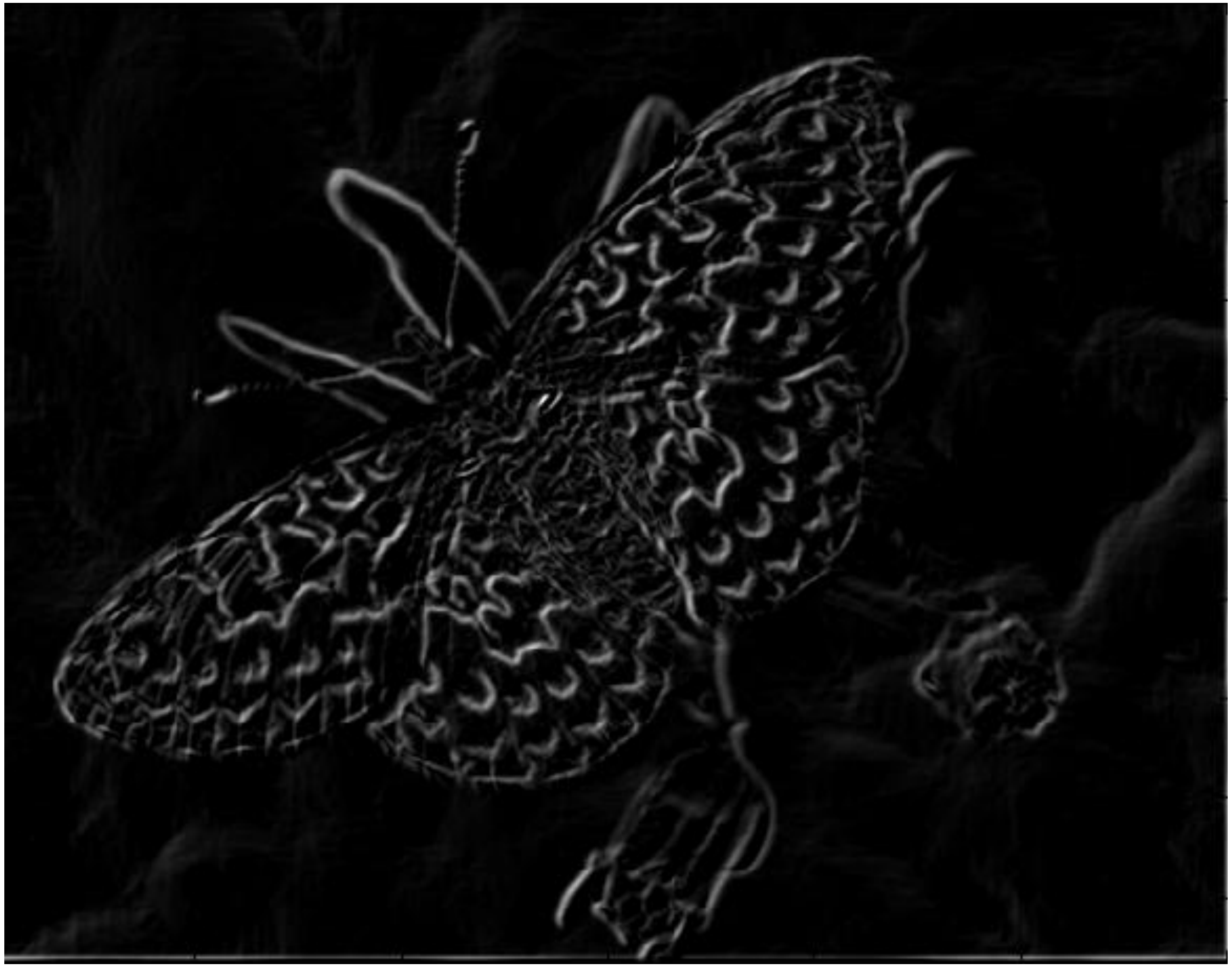
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Slide credit: S. Seitz

Original Image

Slide credit: K. Grauman

Gradient magnitude image



Slide credit: K. Grauman

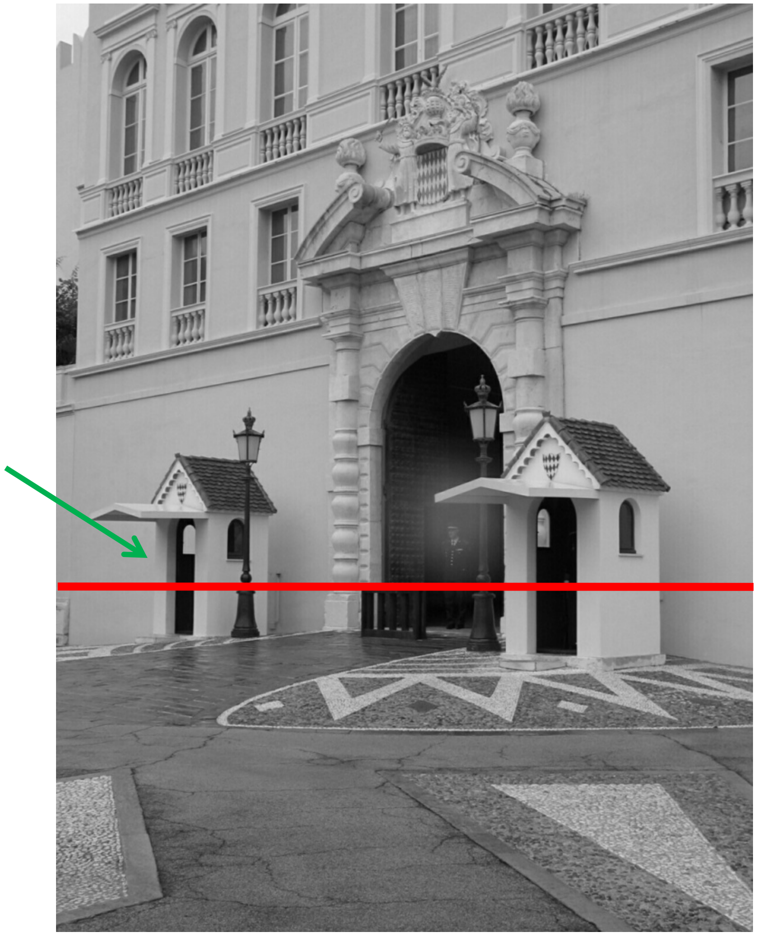
Thresholding gradient with a lower threshold

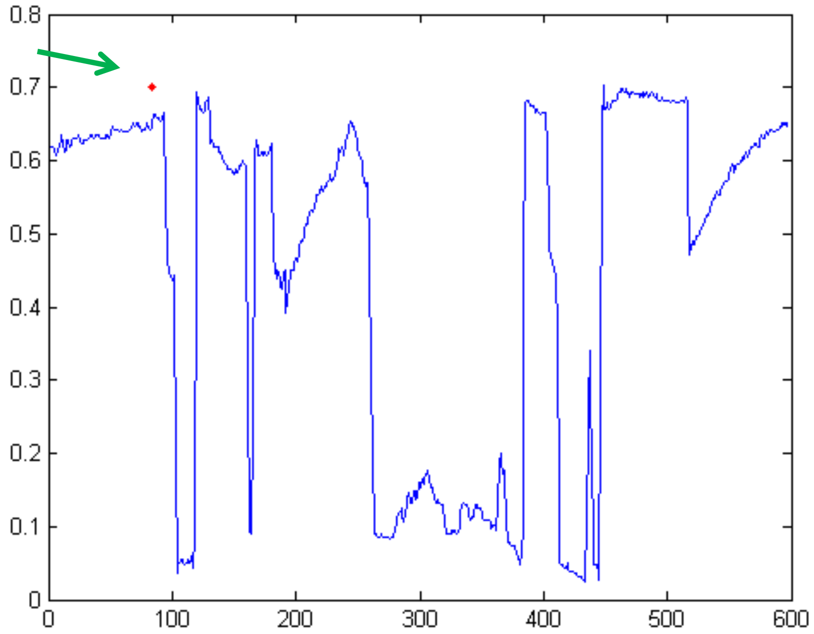
Slide credit: K. Grauman

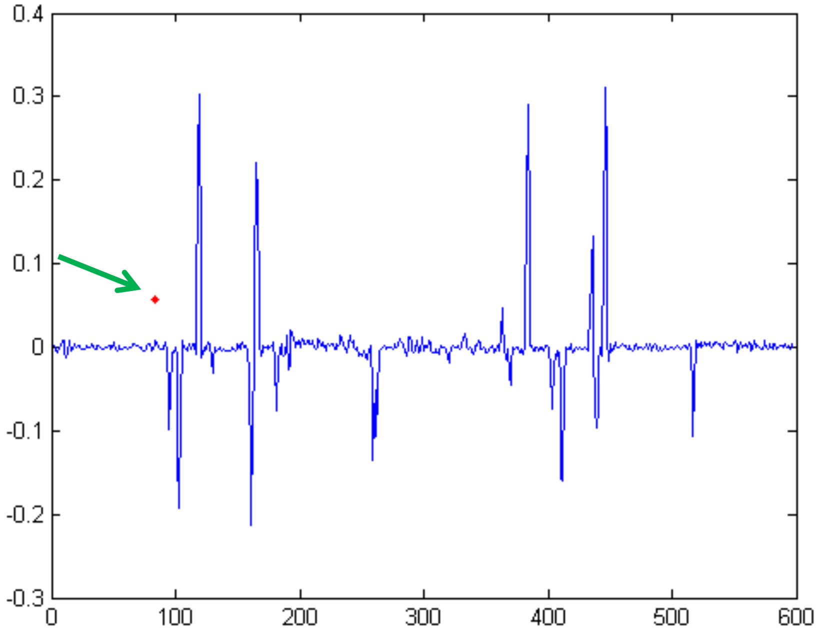
Thresholding gradient with a higher threshold

Slide credit: K. Grauman

Intensity profile

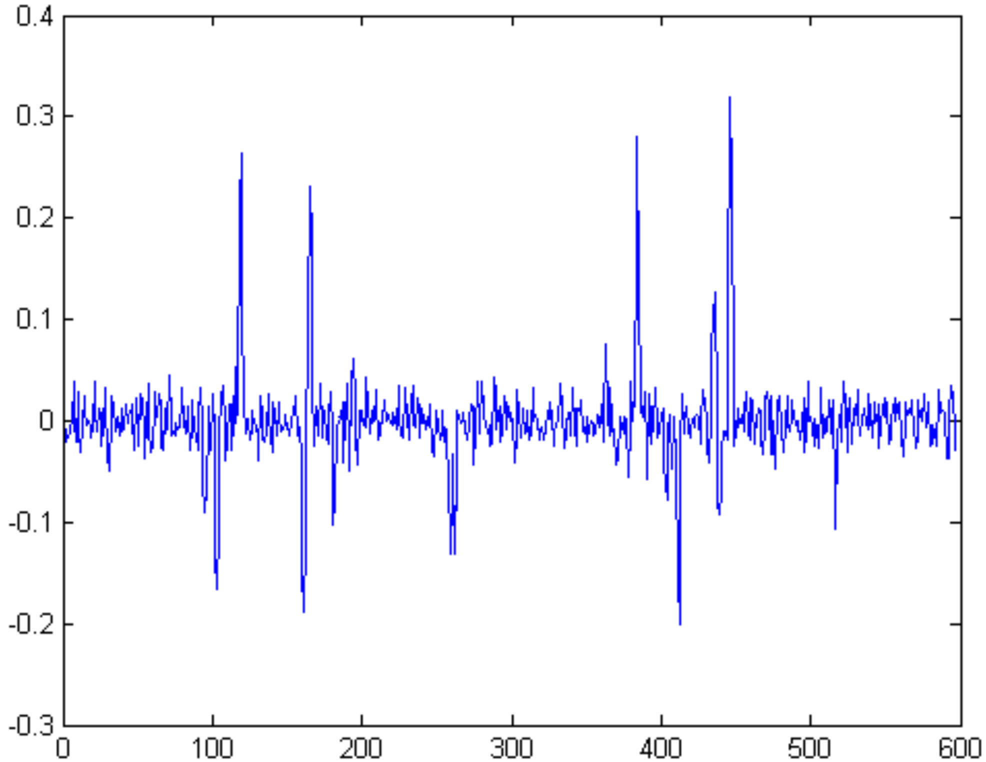






Slide credit: D. Hoiem

With a little Gaussian noise

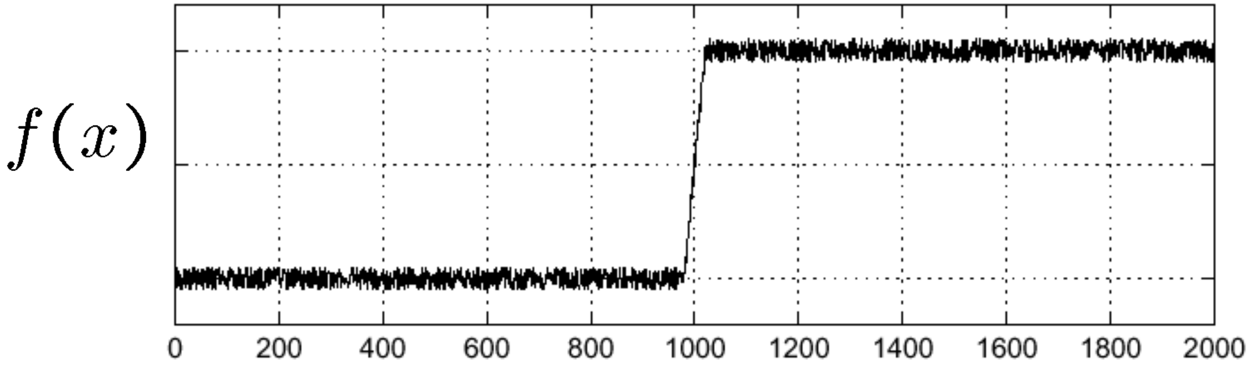


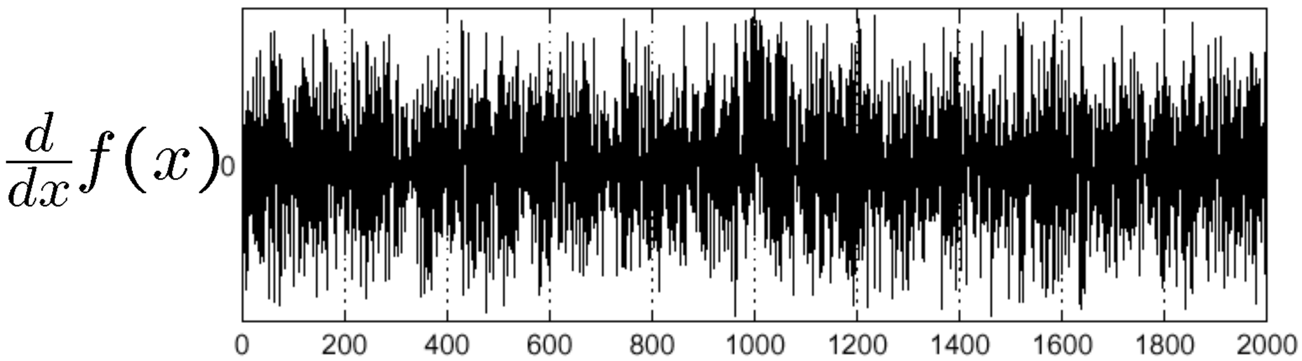
Gradient

Slide credit: D. Hoiem

Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal





Where is the edge?

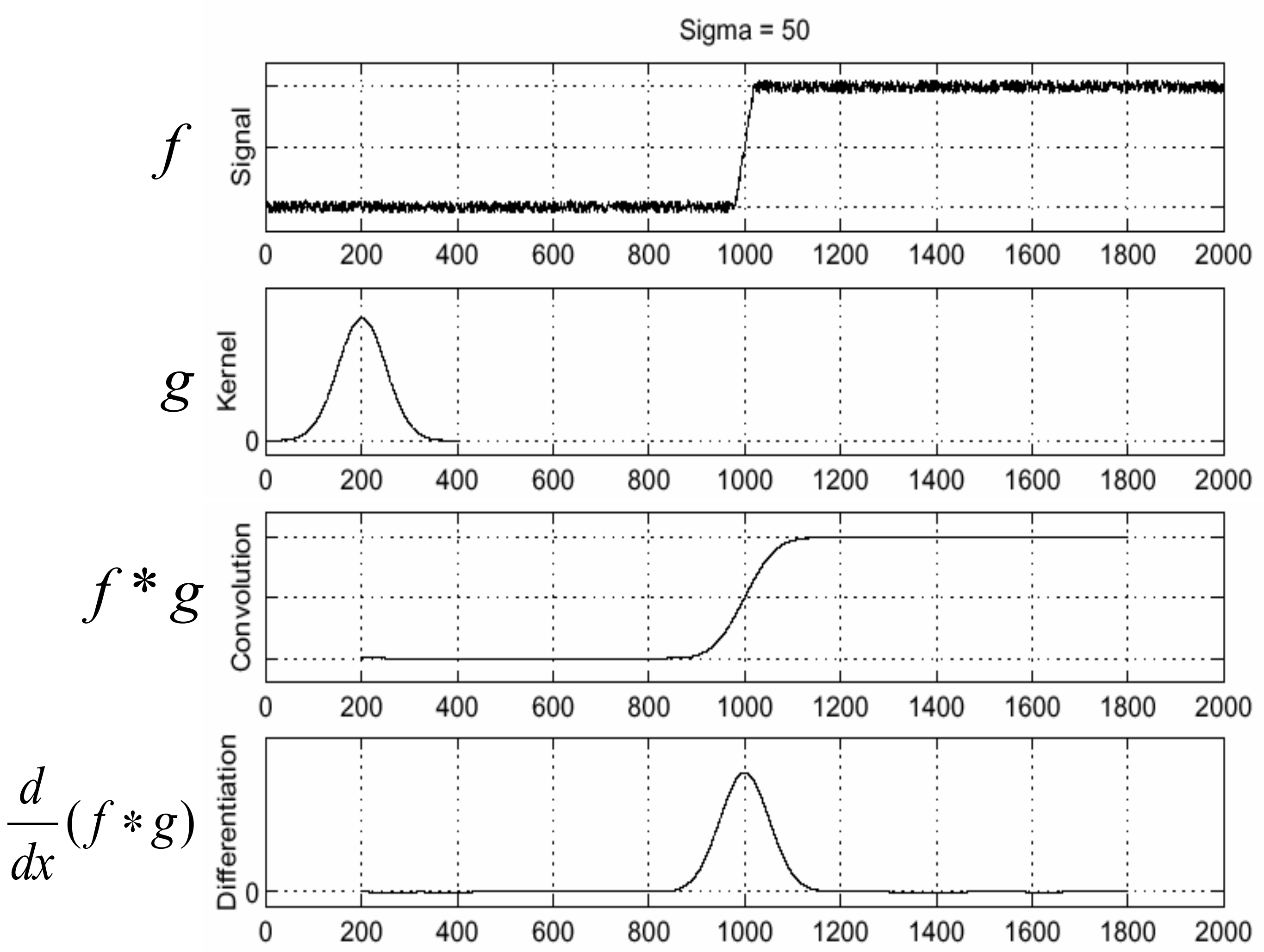
Slide credit: S. Seitz

Effects of noise

- Difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- What can we do about it?

Slide credit: D. Forsyth

Solution: smooth first

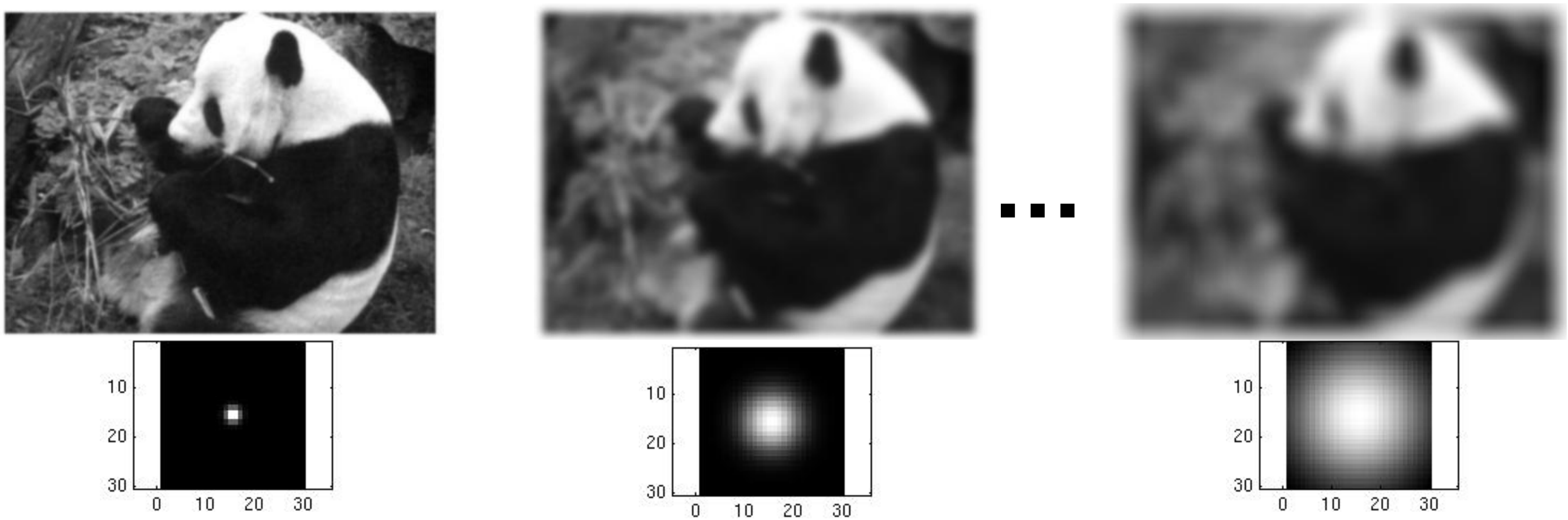


- To find edges, look for peaks in $\frac{d}{dx}(f * g)$

Slide credit: S. Seitz

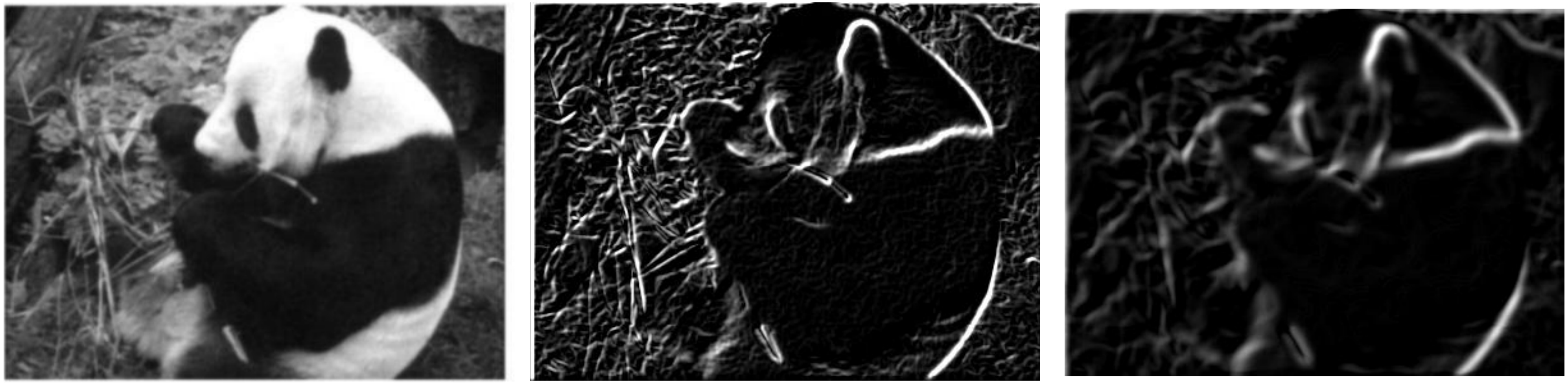
Smoothing with a Gaussian

Recall: parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



Slide credit: K. Grauman

Effect of σ on derivatives



σ = 1 pixel σ = 3 pixels

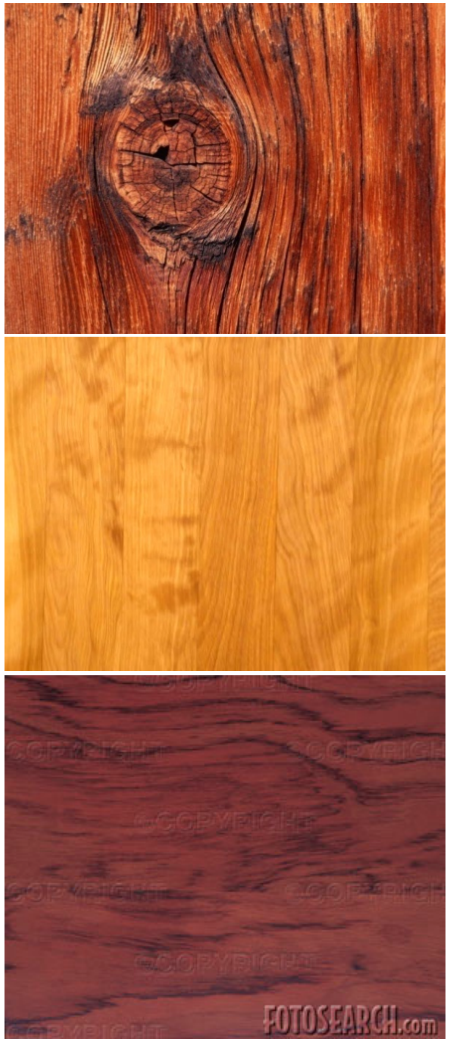
The apparent structures differ depending on Gaussian's scale parameter.

Larger values: larger scale edges detected
Smaller values: finer features detected

Slide credit: K. Grauman

So, what scale to choose?

It depends what we’re looking for.



Slide credit: K. Grauman

Smoothing and Edge Detection

- While eliminating noise via smoothing, we also lose some of the (important) image details.
  - Fine details
  - Image edges
  - etc.
- What can we do to preserve such details?
  - Use edge information during denoising!
  - This requires a definition for image edges.

Chicken-and-egg dilemma!

- Edge preserving image smoothing (Next week's topic!)

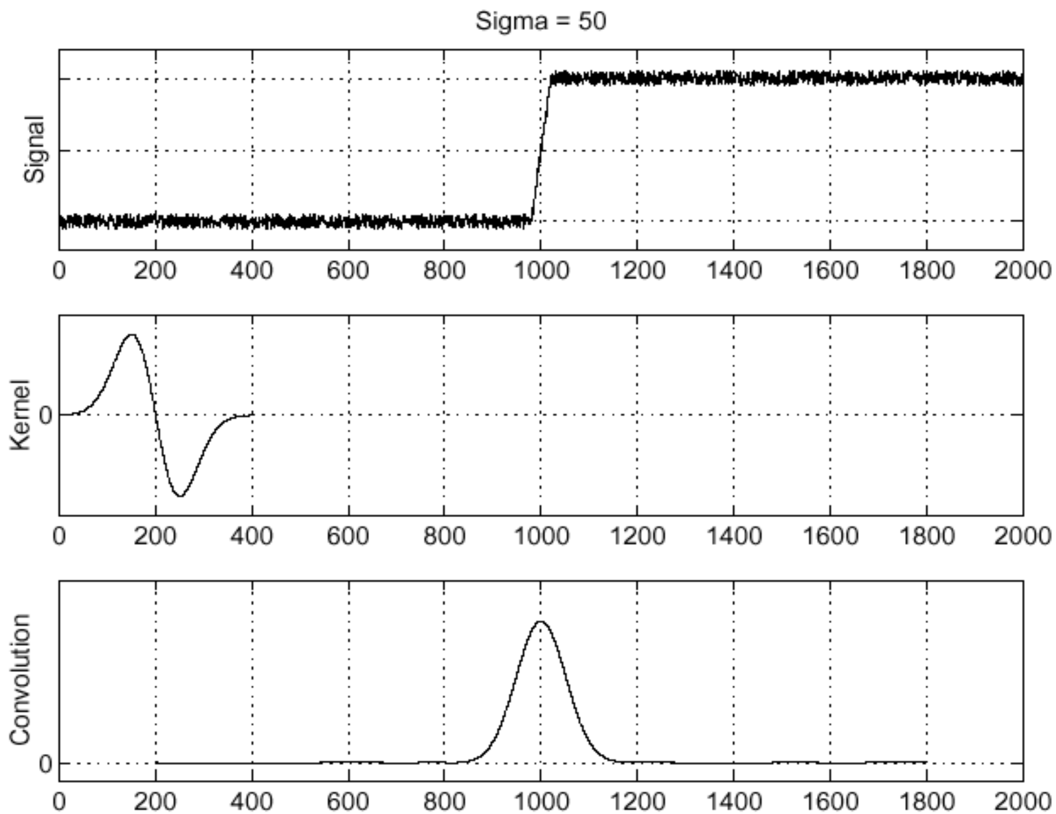
Derivative theorem of convolution

- Differentiation is convolution, and convolution is associative:
- This saves us one operation: $\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$

f

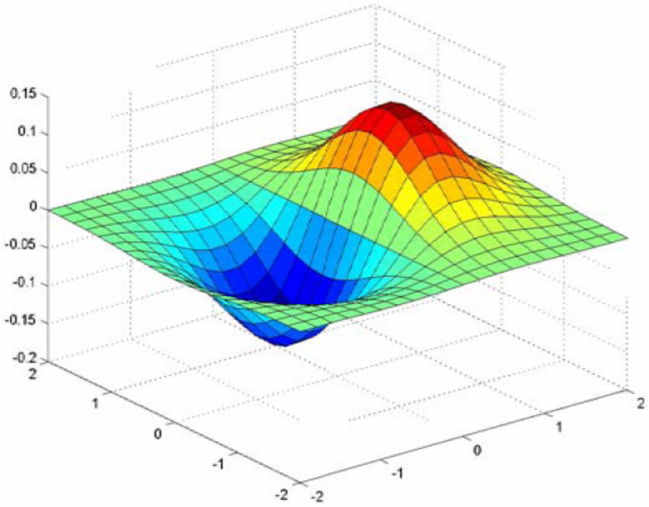
$\frac{d}{dx} g$

$f * \frac{d}{dx} g$

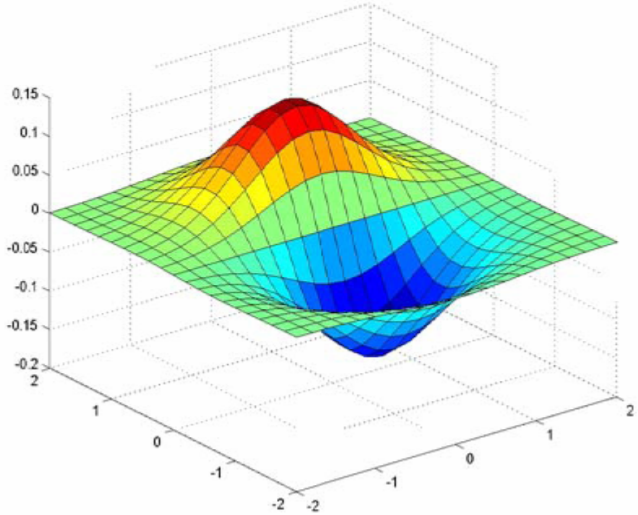


Slide credit: S. Seitz

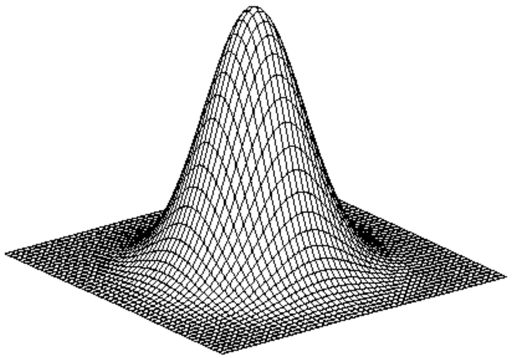
Derivative of Gaussian filter



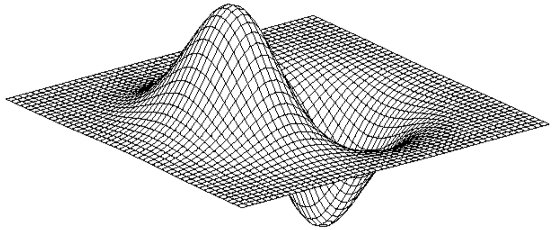
x-direction



y-direction

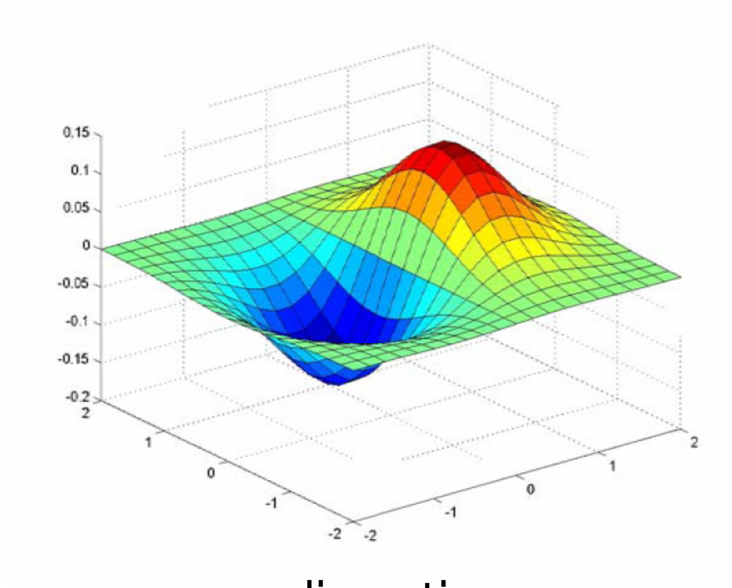


* [1 -1] =

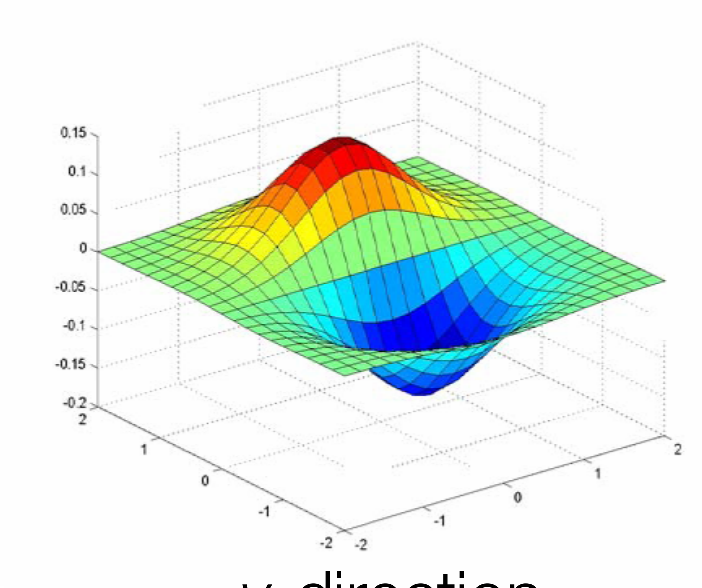


Slide credit: S. Lazebnik

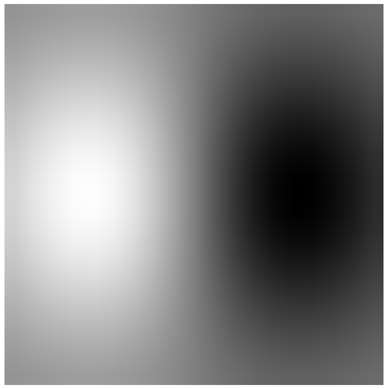
Derivative of Gaussian filter

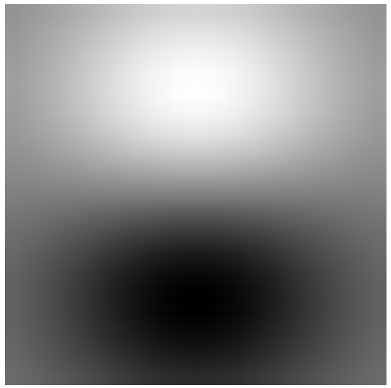


x-direction



y-direction

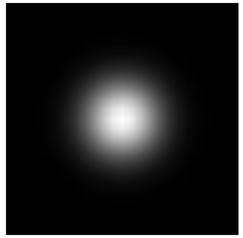




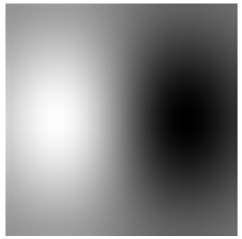
- Which one finds horizontal/vertical edges?

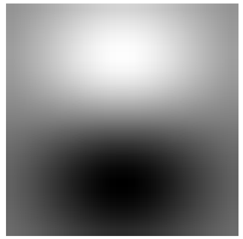
Slide credit: S. Lazebnik

Smoothing vs. derivative filters



- Smoothing filters
  - Gaussian: remove "high-frequency" components; "low-pass" filter
  - Can the values of a smoothing filter be negative?
  - What should the values sum to?
    - **One:** constant regions are not affected by the filter





- Derivative filters
  - Derivatives of Gaussian
  - Can the values of a derivative filter be negative?
  - What should the values sum to?
    - **Zero:** no response in constant regions
  - High absolute value at points of high contrast

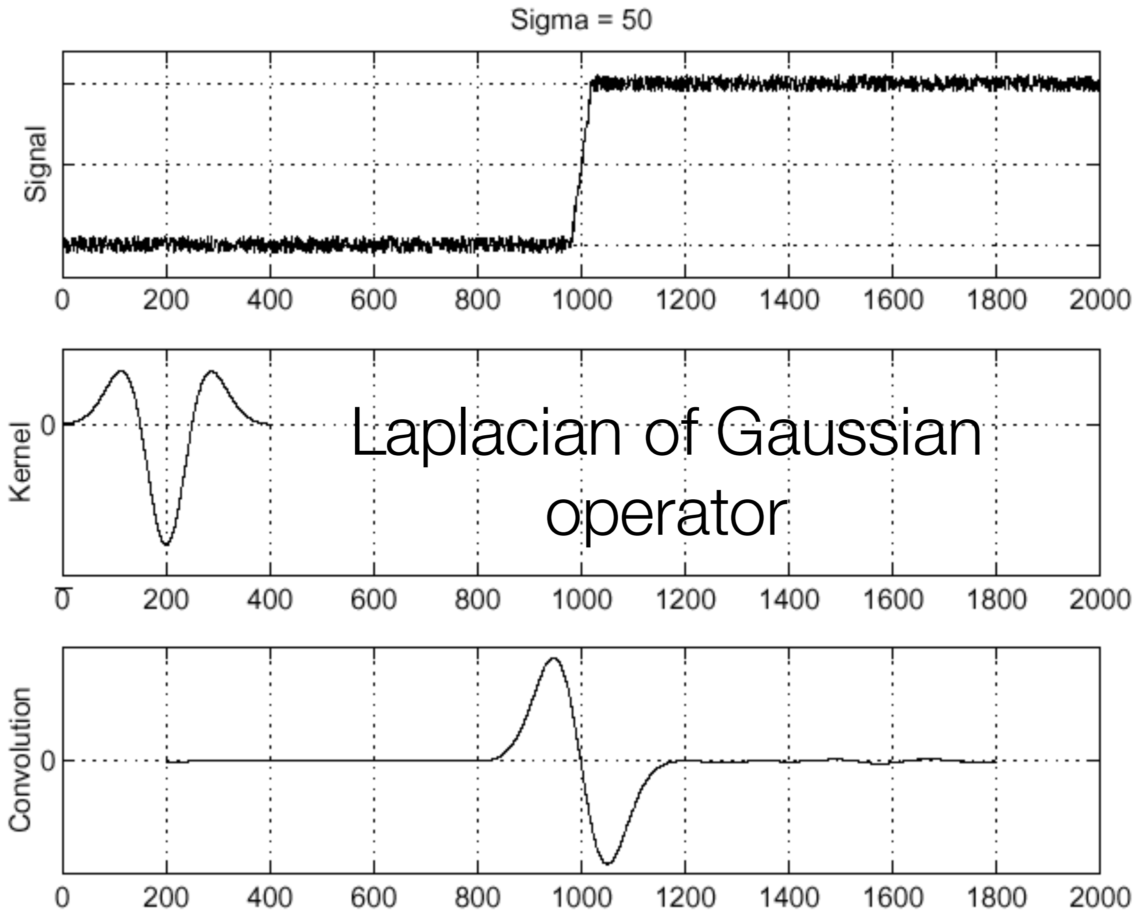
Slide credit: S. Lazebnik

Laplacian of Gaussian

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

$\frac{\partial^2}{\partial x^2}h$

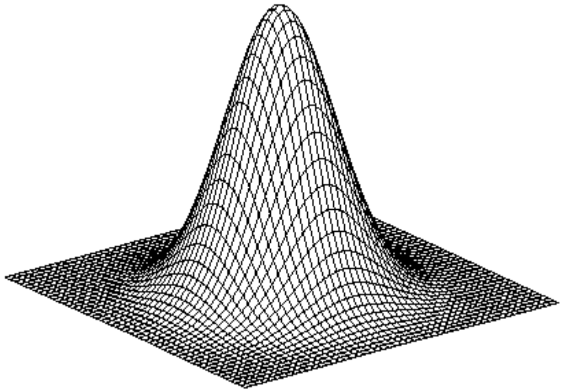
$(\frac{\partial^2}{\partial x^2}h) \star f$



Where is the edge? Zero-crossings of bottom graph

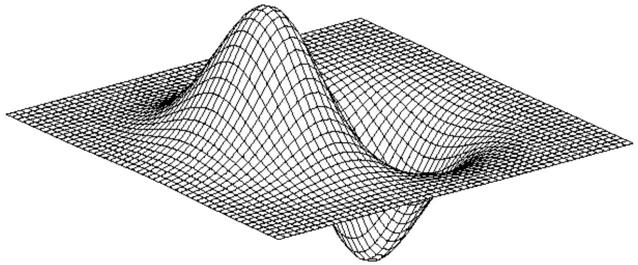
Slide credit: K. Grauman

2D edge detection filters



Gaussian

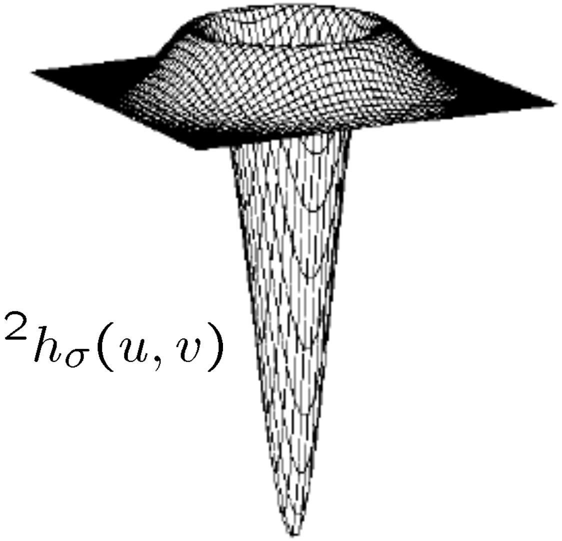
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian



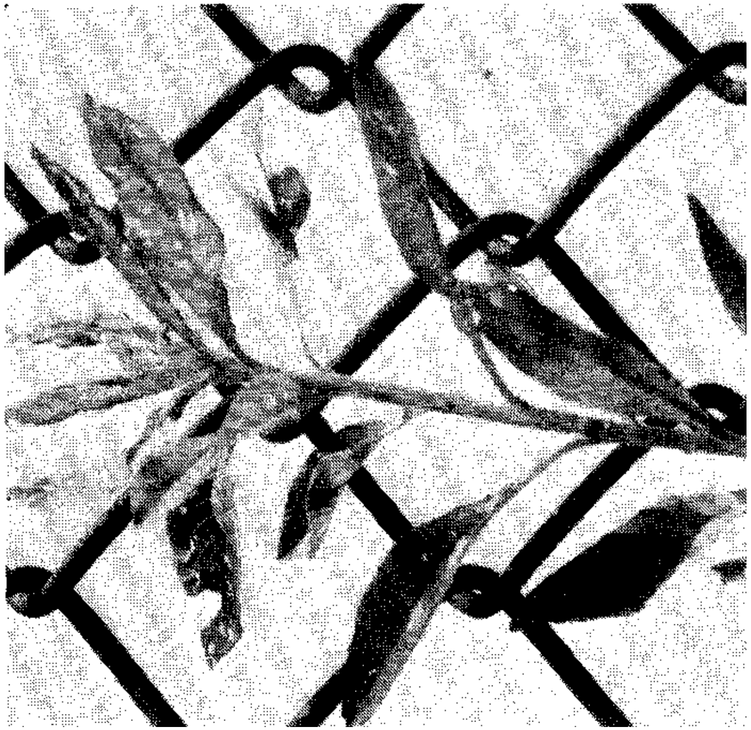
$$\nabla^2 h_\sigma(u,v)$$

- The Laplacian operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Slide credit: K. Grauman

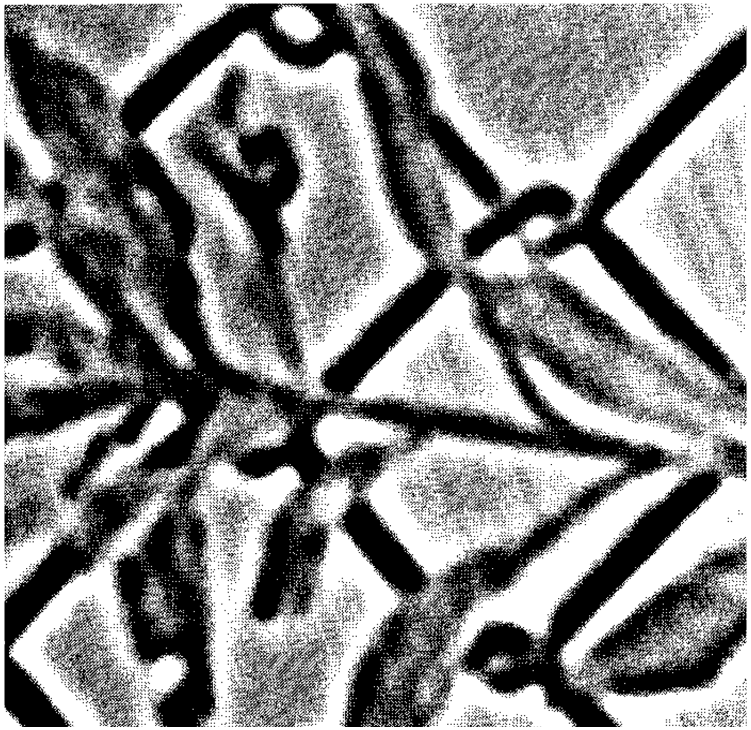
Laplacian of Gaussian



original image

Source: D. Marr and E. Hildreth (1980)

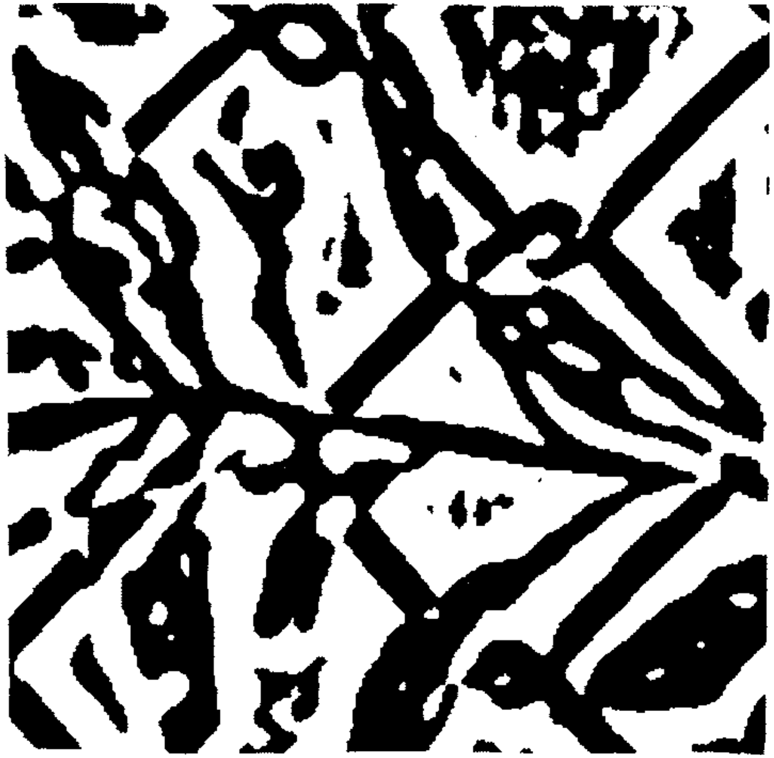
Laplacian of Gaussian



convolution with
$$\nabla^2 h_\sigma(u,v)$$

Source: D. Marr and E. Hildreth (1980)

Laplacian of Gaussian



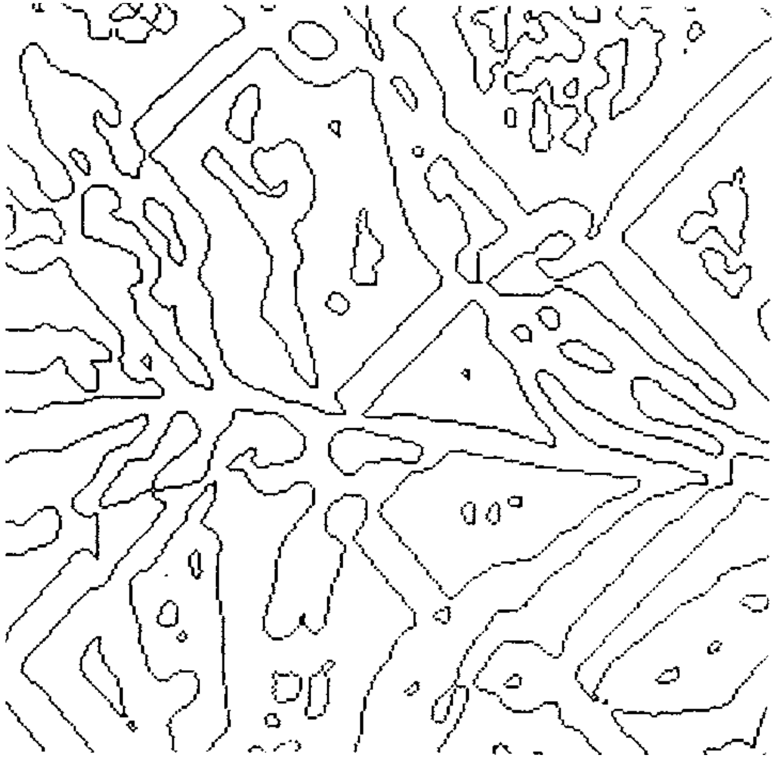
convolution with

$$\nabla^2 h_\sigma(u, v)$$

(pos. values – white, neg. values – black)

Source: D. Marr and E. Hildreth (1980)

Laplacian of Gaussian



zero-crossings

Source: D. Marr and E. Hildreth (1980)

Designing an edge detector

- Criteria for a good edge detector:
  - **Good detection:** the optimal detector should find all real edges, ignoring noise or other artifacts
  - **Good localization**
    - the edges detected must be as close as possible to the true edges
    - the detector must return one point only for each true edge point
- Cues of edge detection
  - Differences in color, intensity, or texture across the boundary
  - Continuity and closure
  - High-level knowledge

Slide credit: L. Fei-Fei

The Canny edge detector

original image (Lena)

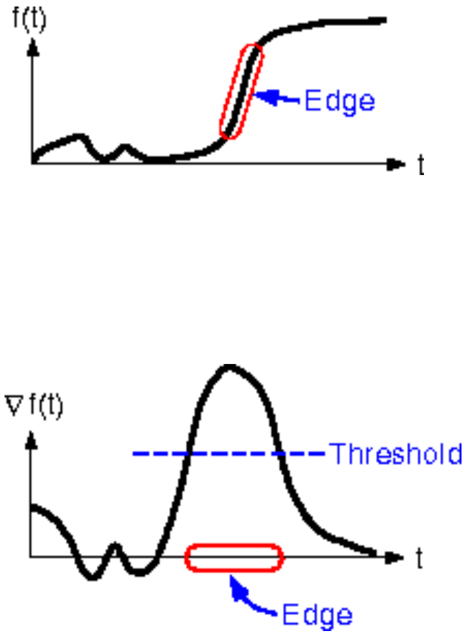
Slide credit: K. Grauman

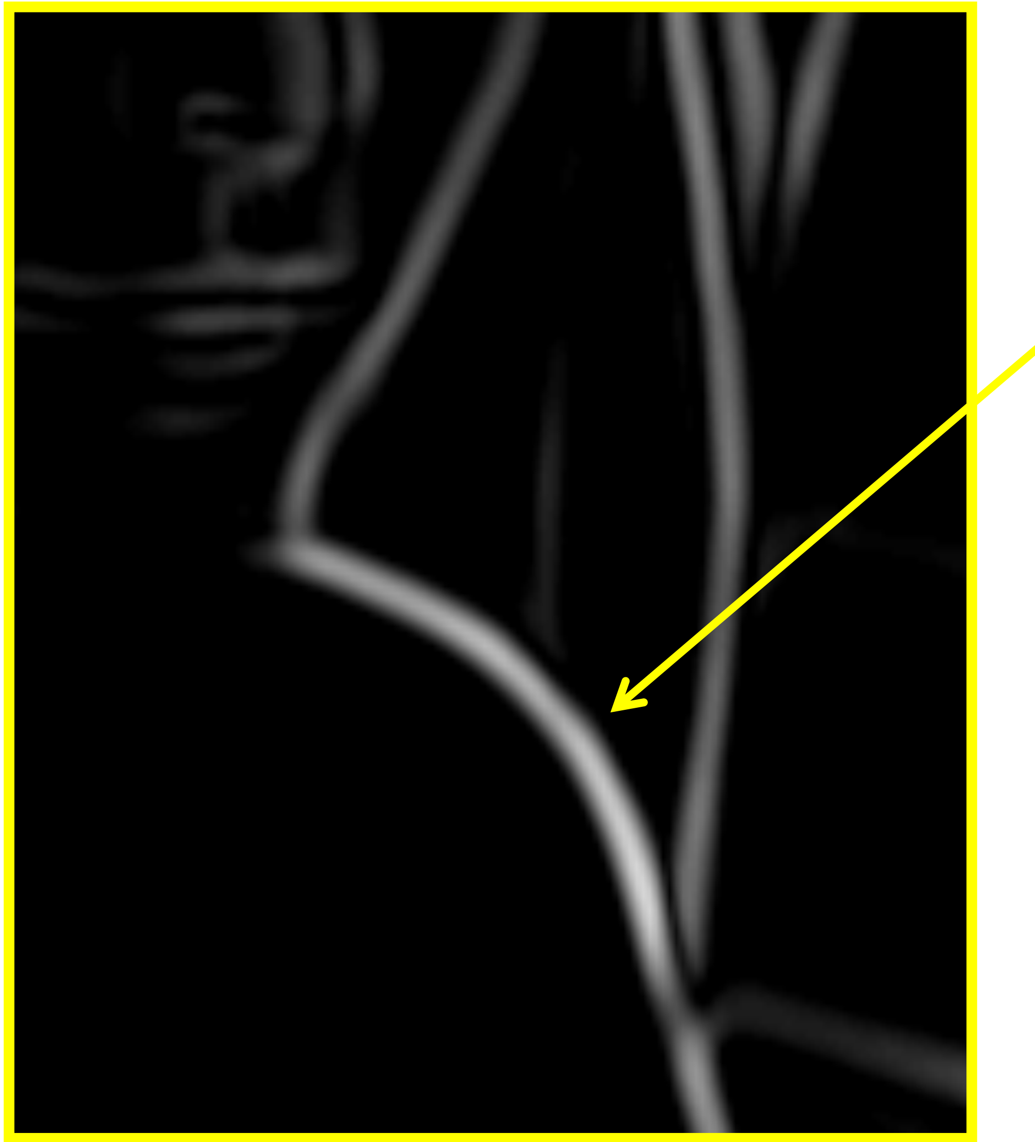
The Canny edge detector

thresholding

Slide credit: K. Grauman

The Canny edge detector

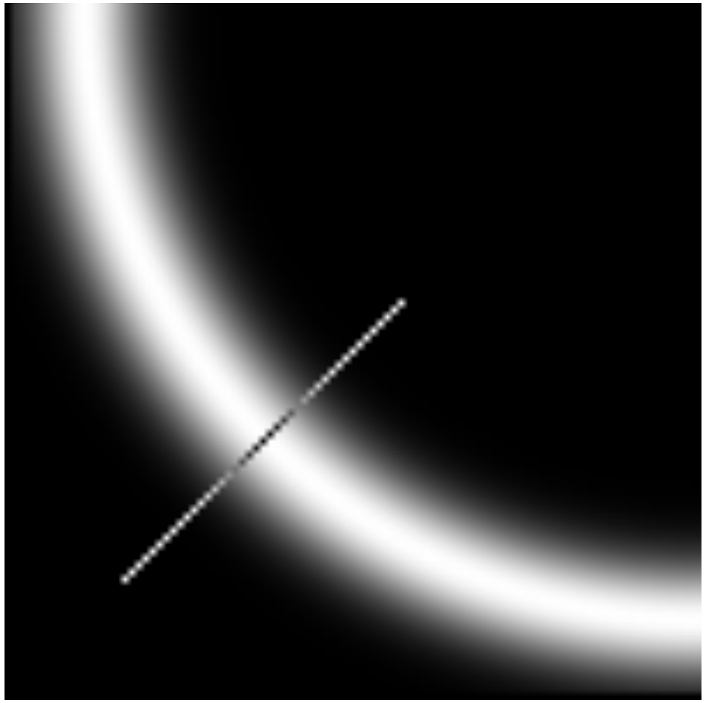


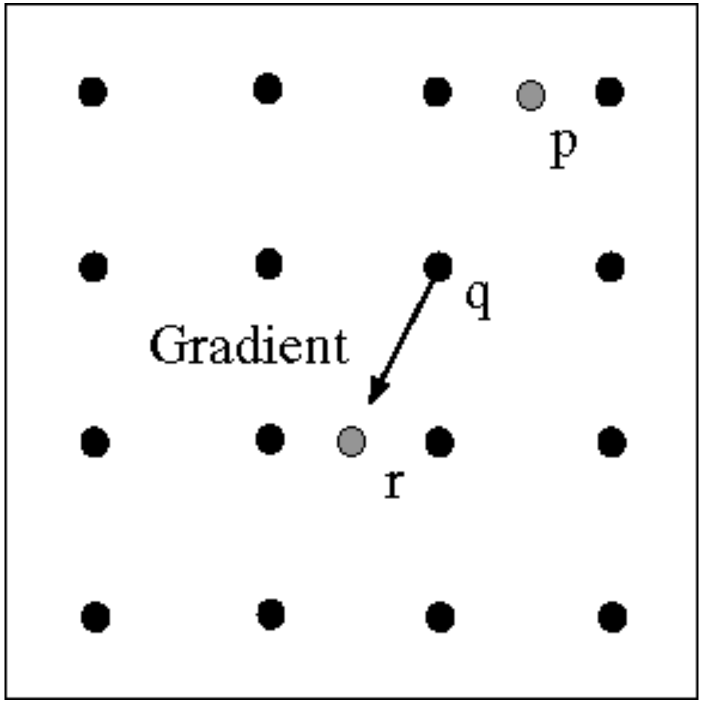


How to turn these thick regions of the gradient into curves?

Slide credit: K. Grauman

Non-maximum suppression





Check if pixel is local maximum along gradient direction, select single max across width of the edge

- requires checking interpolated pixels p and r

Slide credit: K. Grauman

The Canny Edge Detector

Problem: pixels along this edge didn't survive the thresholding

thinning
(non-maximum suppression)

Slide credit: K. Grauman

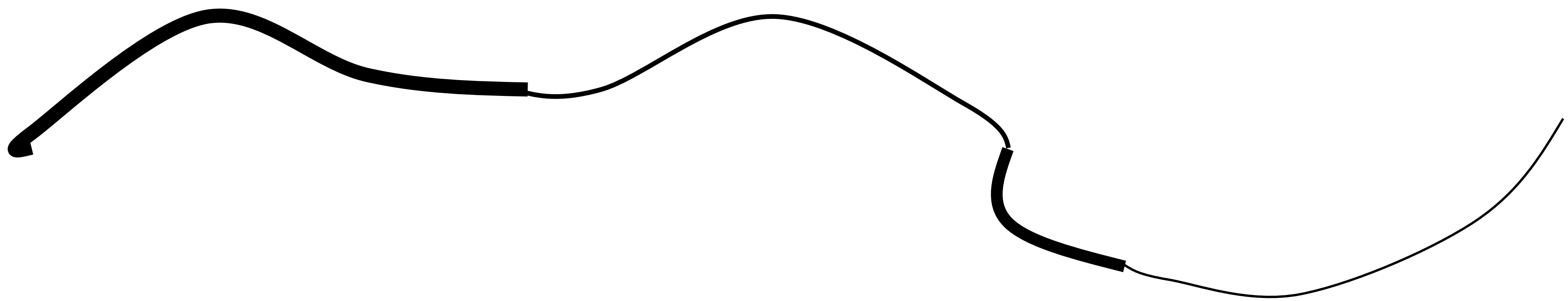
Hysteresis thresholding

- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels

Slide credit: J. Hays

Hysteresis thresholding

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



Slide credit: S. Seitz

Hysteresis thresholding

original image

high threshold
(strong edges)

low threshold
(weak edges)

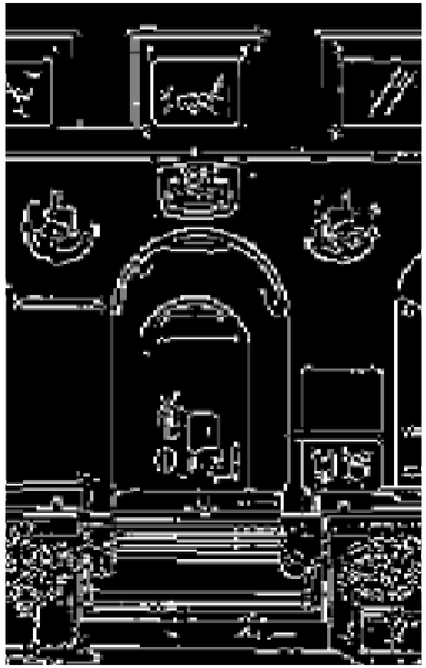
hysteresis threshold

Slide credit: L. Fei-Fei

Hysteresis thresholding

high threshold
(strong edges)

low threshold
(weak edges)



hysteresis threshold

Slide credit: L. Fei-Fei

Recap: Canny edge detector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. **Non-maximum suppression**:
   - Thin wide “ridges” down to single pixel width
4. **Linking and thresholding** (**hysteresis**):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

- MATLAB: `edge(image, ‘canny’);`

Slide credit: D. Lowe, L. Fei-Fei

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

Slide credit: S. Seitz

Low-level edges vs. perceived contours

Background

Texture

Shadows

Slide credit: K. Grauman

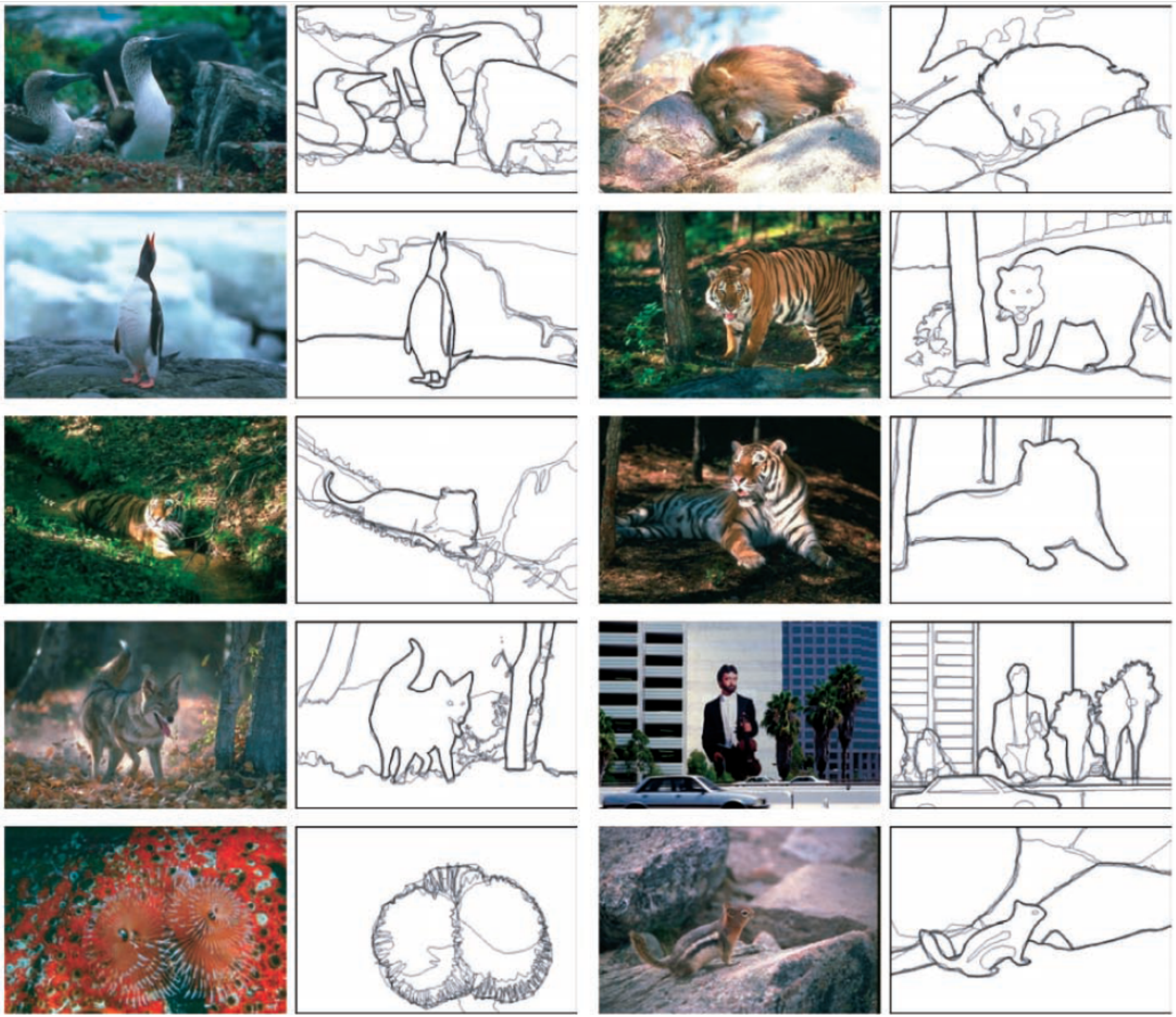
Edge detection is just the beginning…

image | human segmentation | gradient magnitude

- Berkeley segmentation database:
  http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

Source: S. Lazebnik

Learn from humans which combination of features is most indicative of a "good" contour?
[D. Martin et al. PAMI 2004]



Human-marked segment boundaries

Slide credit: K. Grauman