ME/CS 132:
Introduction to
Vision-based Robot Navigation

Low-level Image Processing

Larry Matthies

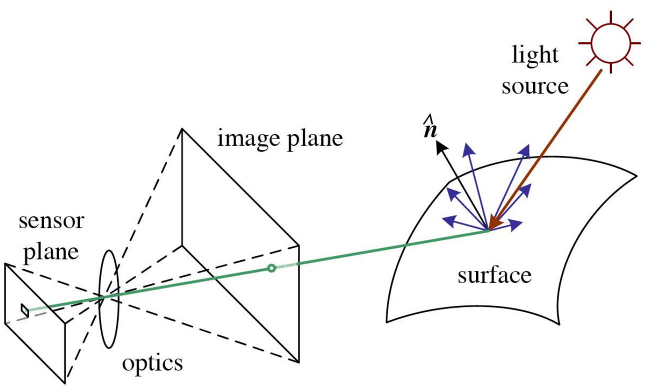
lhm@jpl.nasa.gov, 818-354-3722

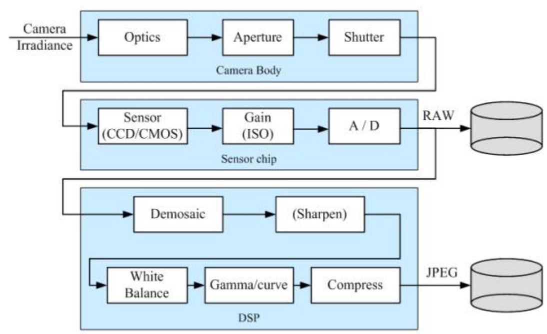
Announcements

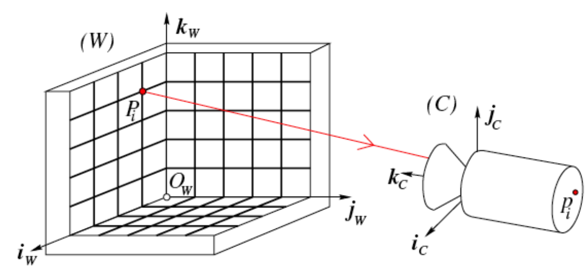
- First homework grading is done
- Second homework is due next Tuesday
- Third homework will be due Feb 8; combines material on outlier detection and egomotion
- (Next quarter)

Recap and Where We're Going: 1st Module (Done)

- Image formation, cameras, and camera calibration
  - Illumination and radiometry
    - Sources of light – sunlight, thermal emission, night sky glow
    - Propagation of light – reflection from surfaces, attenuation in media
  - Cameras
    - Basic optics
    - Camera architectures
    - Image detectors – materials, architectures, performance, for various regions of the EM spectrum
  - Geometric camera modeling and calibration



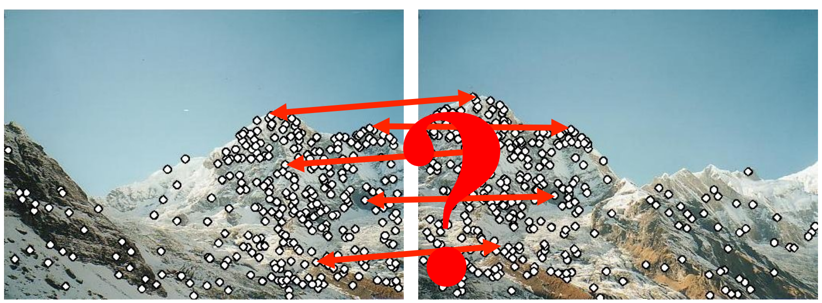


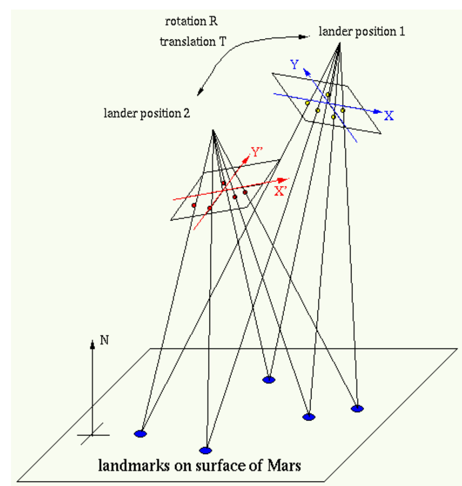


Recap and Where We're Going: 2nd Module (Midway Through)

- Visual motion estimation and 3-D perception
  - Low-level image processing
  - Feature detection, matching, and outlier detection
  - Pose estimation (egomotion) and visual odometry
  - Dense range imaging with stereo vision
  - Other range sensors, range data analysis, introduction to robots

- 1-week mini-project on visual localization using a stereo camera head to match landmark points





Recap and Where We're Going: 3rd Module

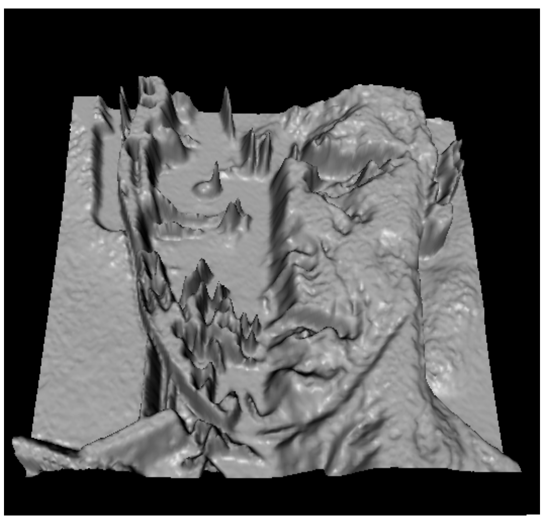
- State estimation, localization, and mapping
  - Introduction to estimation
  - Linear Kalman filter
  - Extended Kalman filter
  - Particle filters and the UKF
  - Simultaneous localization and mapping

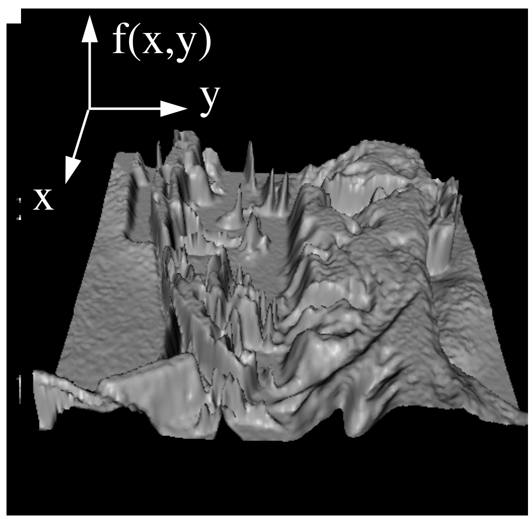
- 1-week mini-project on stereo-based SLAM for localization and occupancy grid mapping with ladar

This Lecture

- Point operators
- Neighborhood operators
- Edge detection
- Image pyramids
- Geometric image transformations
- Reading material:
  - Today: Szeliski ch. 3 and sections 4.2-4.3
  - Next Tuesday: Szeliski ch. 11

Images as Functions





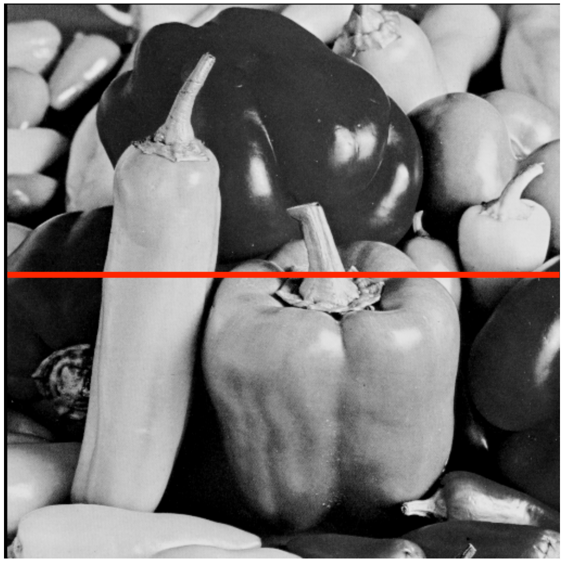
Interpret image either as:

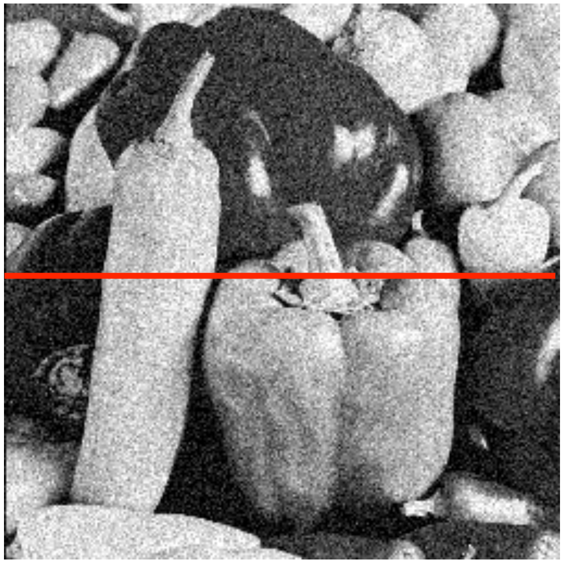
- Continuous image *f(x,y)*
- Discrete array *f[x,y]*

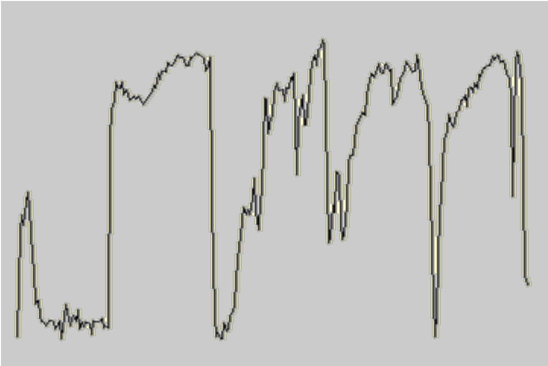
Images are affected by:

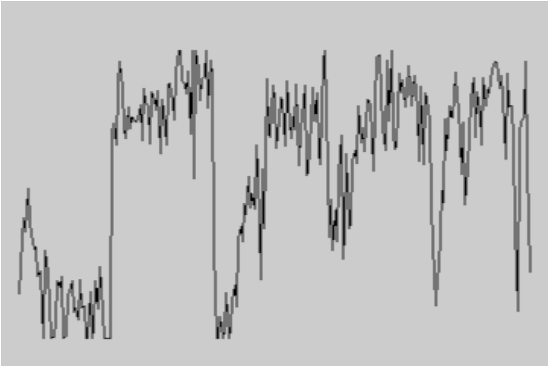
- Noise
- Nonlinear distortions of intensity and geometry

Image Noise







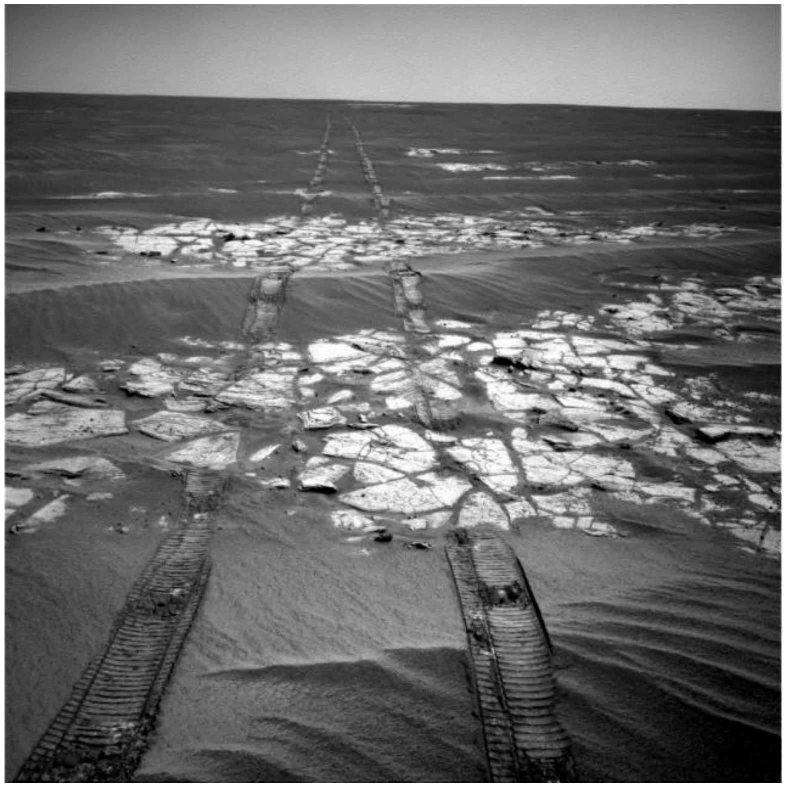


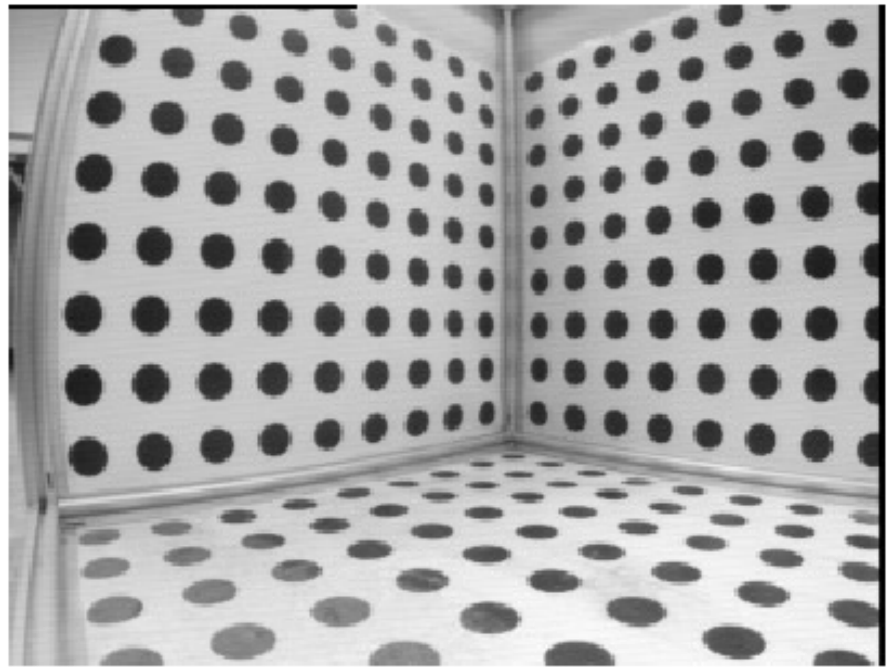
$$f(x,y) = \overbrace{\hat{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:

$$\eta(x,y) \sim \mathcal{N}(\mu, \sigma)$$

Image Distortions





Point Operators

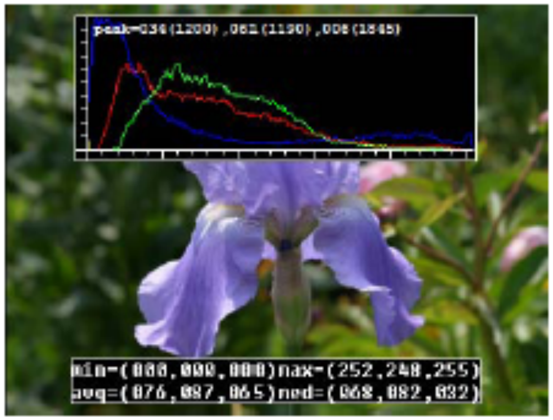
- Output pixel = function of one input pixel in one or more images

$$g(x) = h(f(x)) \text{ or } g(x) = h(f_0(x), \ldots, f_n(x))$$

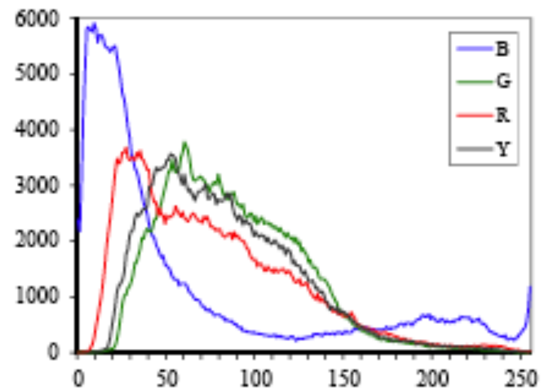
$$g(i, j) = h(f(i, j))$$

- Examples
  - Monadic linear: bias and gain adjustment, $g(x) = a\,f(x) + b$
  - Monadic nonlinear: gamma correction, $g(x) = [f(x)]^{1/\gamma}$
  - Dyadic linear: image blending, $g(x) = (1 - \alpha)\,f_0(x) + \alpha\,f_1(x)$

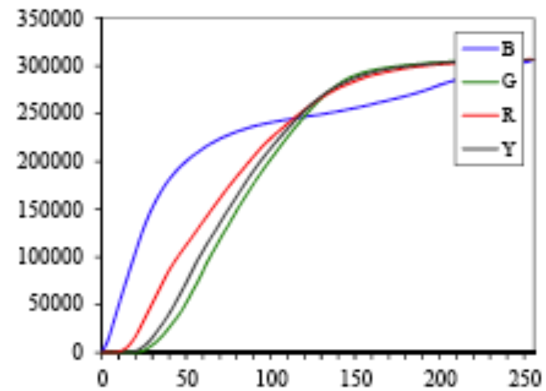
Point Operators: Histogram Equalization



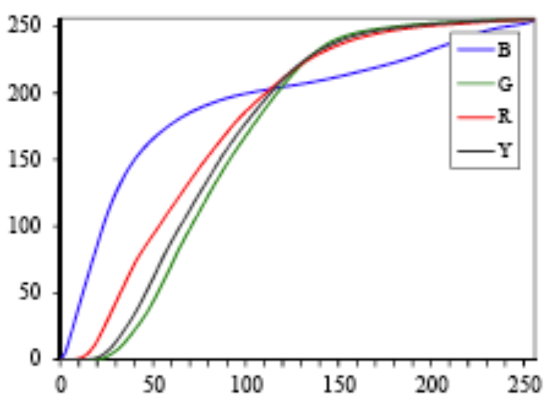
(a)



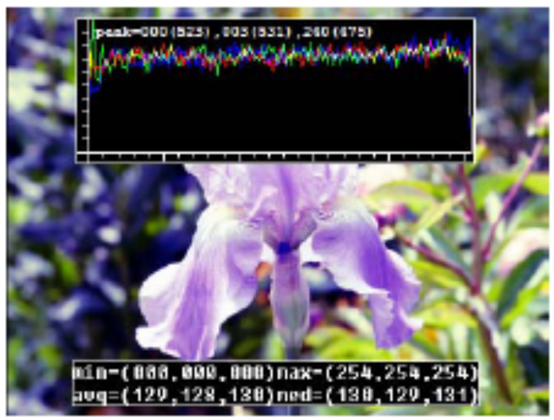
(b)



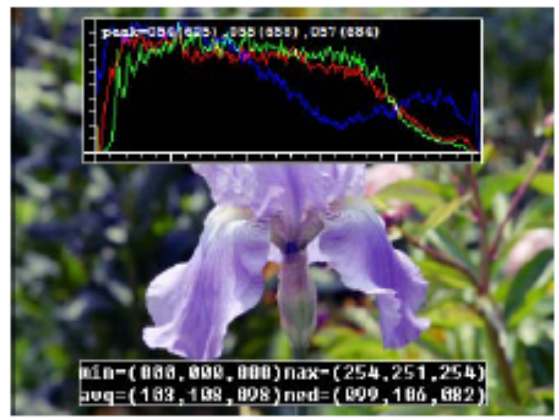
(c)



(d)

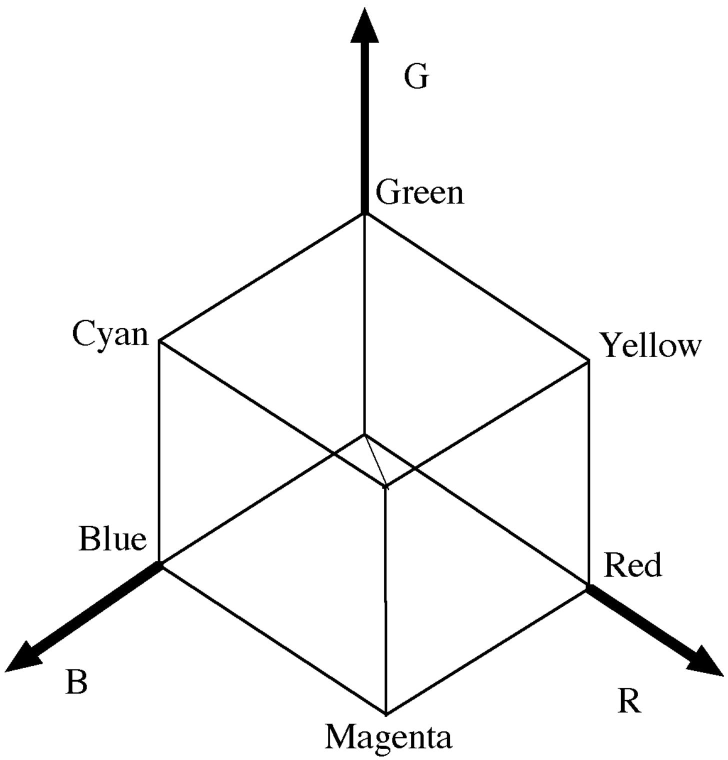


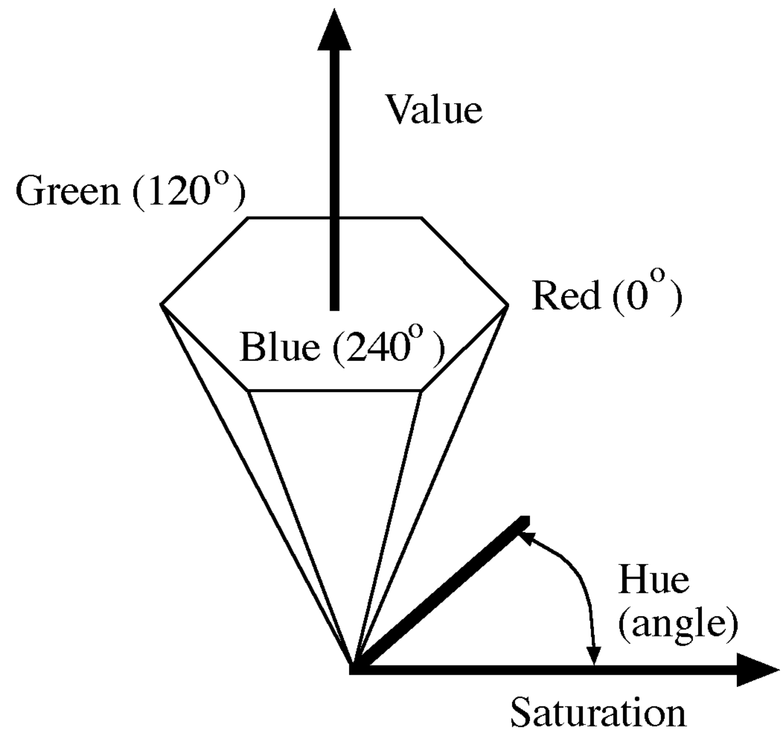
(e)



(f)

Point Operators: Color Space Conversion (e.g. RGB to HSV)

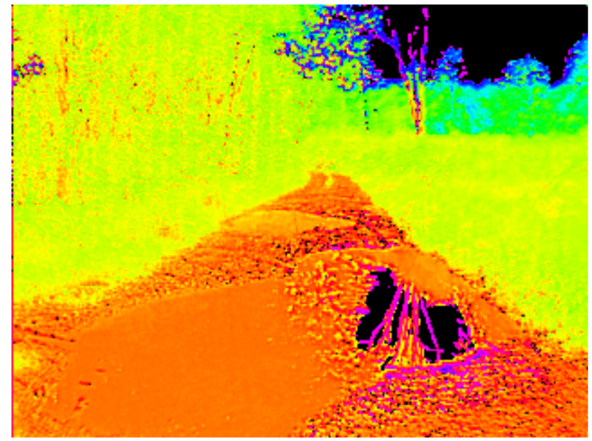




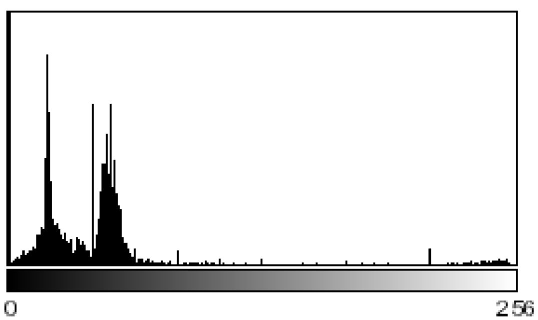
$$HSV(i, j) = f\,(\ RGB(i, j)\ )$$

Application of Color Space Conversion: Simple Segmentation

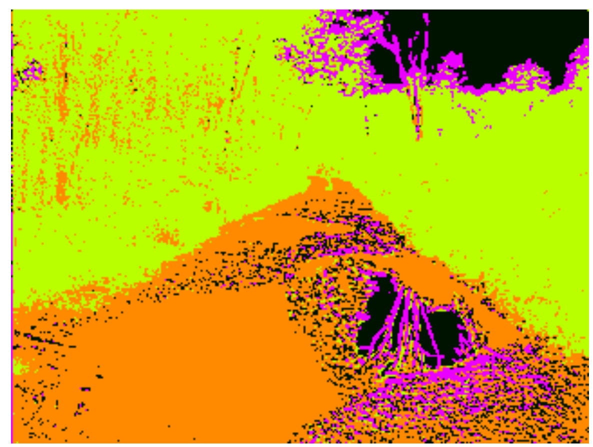
(a)



(b)



(c)



(d)

Neighborhood Operators (or Window Operators, or Local Operators)

- Slide one small window of numbers over the image and compute some function, replacing the central pixel in the image with the output of the function.

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Linear Filters

$$g(i,j) = \sum_{k,l} f(i+k, j+l) h(k,l)$$

- *h()* is called the *filter*, or *kernel*, or *mask*; entries in *h()* are the filter coefficients. Abbreviated notation:

$$g = f \otimes h$$ Correlation

- Can also be written:

$$g(i,j) = \sum_{k,l} f(i-k, j-l) h(k,l) = \sum_{k,l} f(k,l) h(i-k, j-l)$$

$$g = f * h$$ Convolution

Some Basic Properties of Convolution

- Commutative:

$$f * g = g * f$$

- Associative:

$$f * (g * h) = (f * g) * h$$

- Distributes over addition:

$$f * (g + h) = f * g + f * h$$

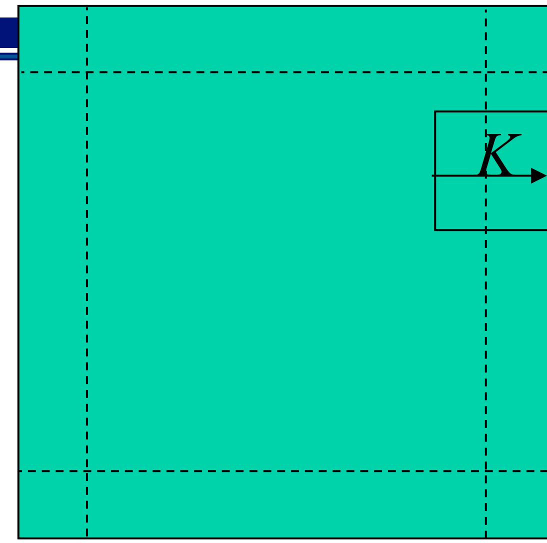
- Differentiation:

$$(f * g)' = f' * g = f * g'$$

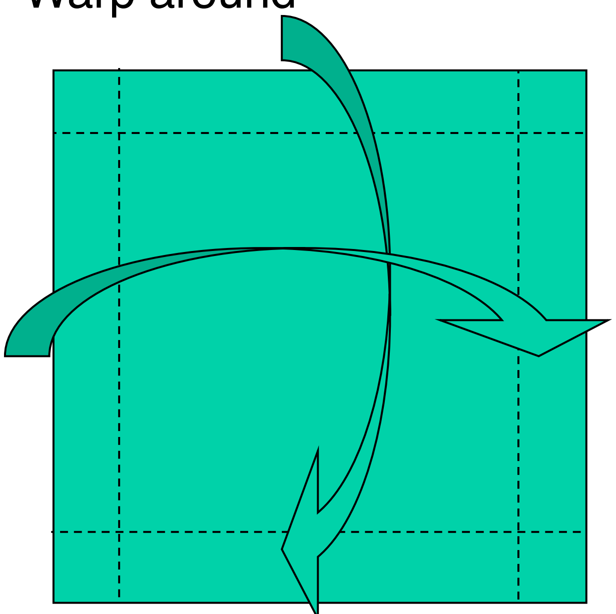
- Shift invariant

Neighborhood Operators: Border Effects

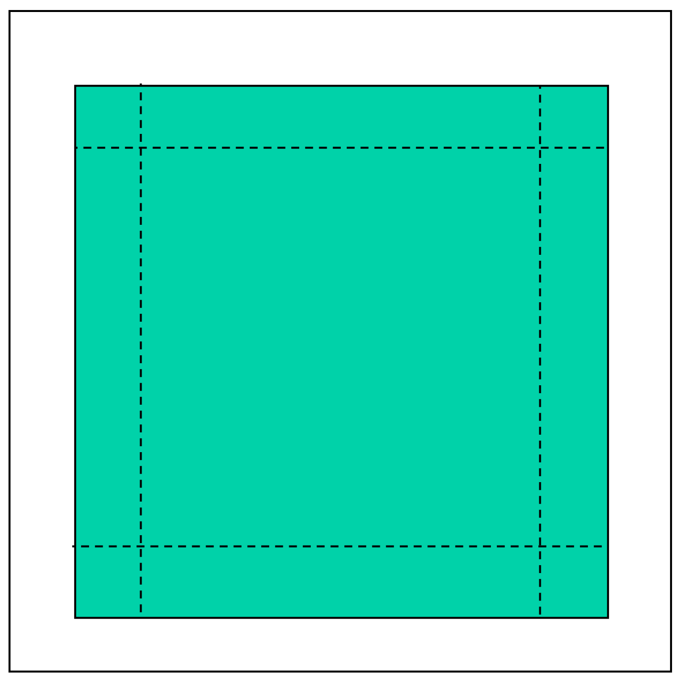
- When applying convolution with a *KxK* kernel, the result is undefined for pixels closer than *K* pixels to the border of the image
- Options:



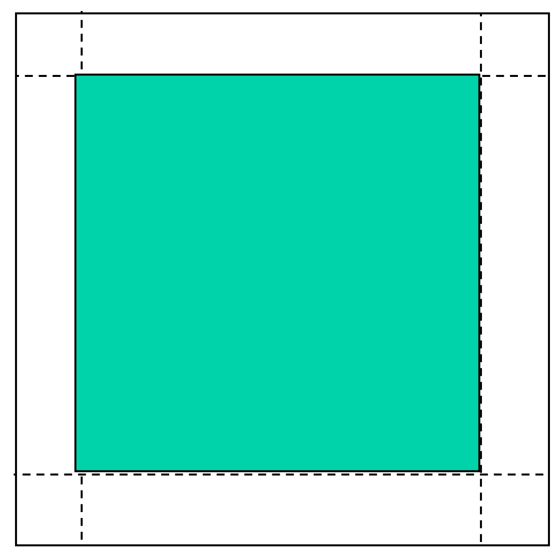
Warp around



Expand/Pad



Crop



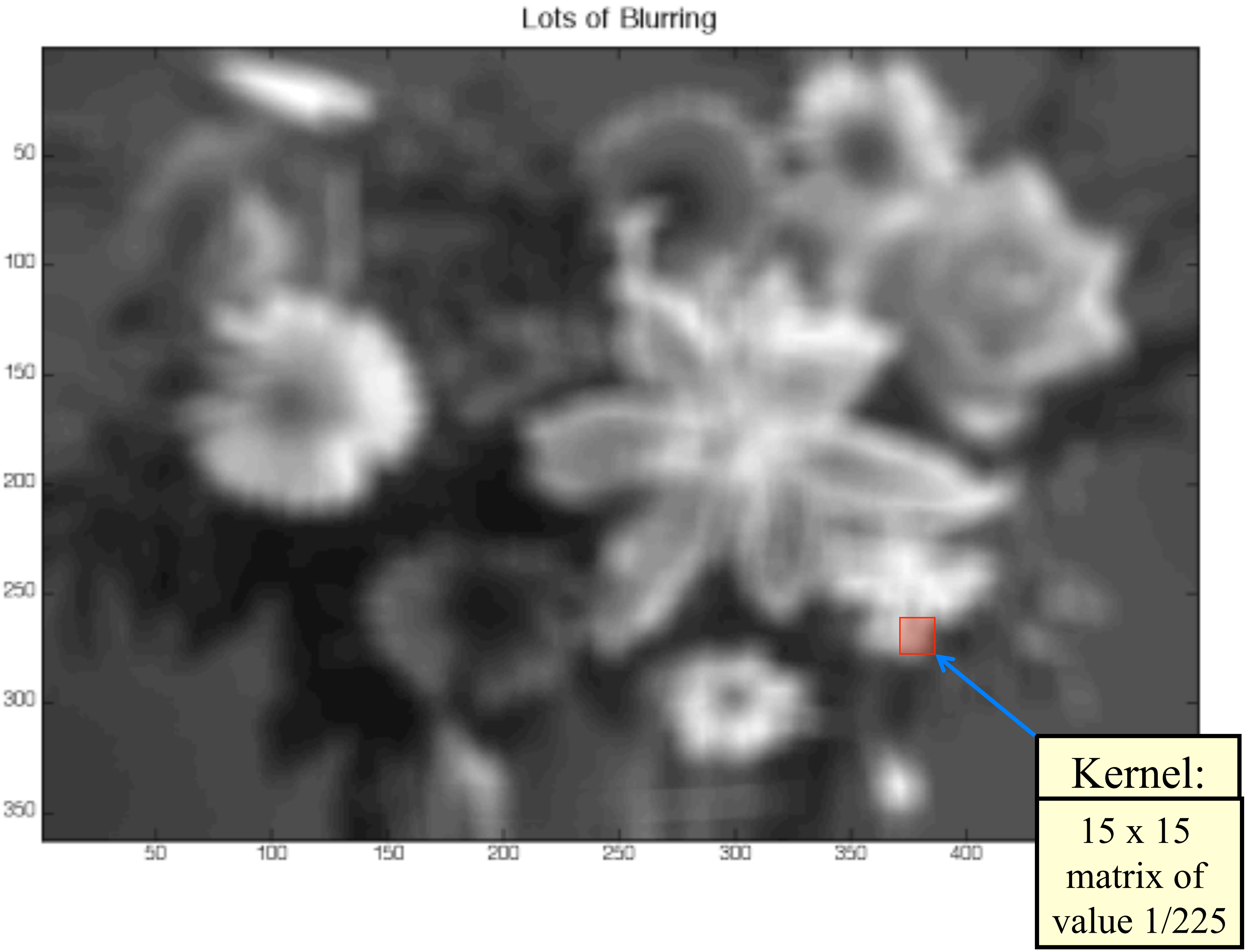
Original Image

Kernel:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

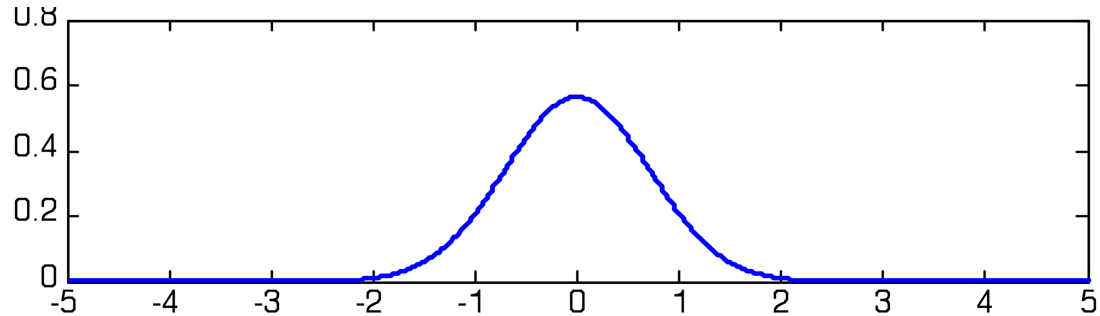
Kernel:

0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04


Lots of Blurring
50
100
150
200
250
300
350
50
100
150
200
250
300
350
400
Kernel:
15 x 15
matrix of
value 1/225

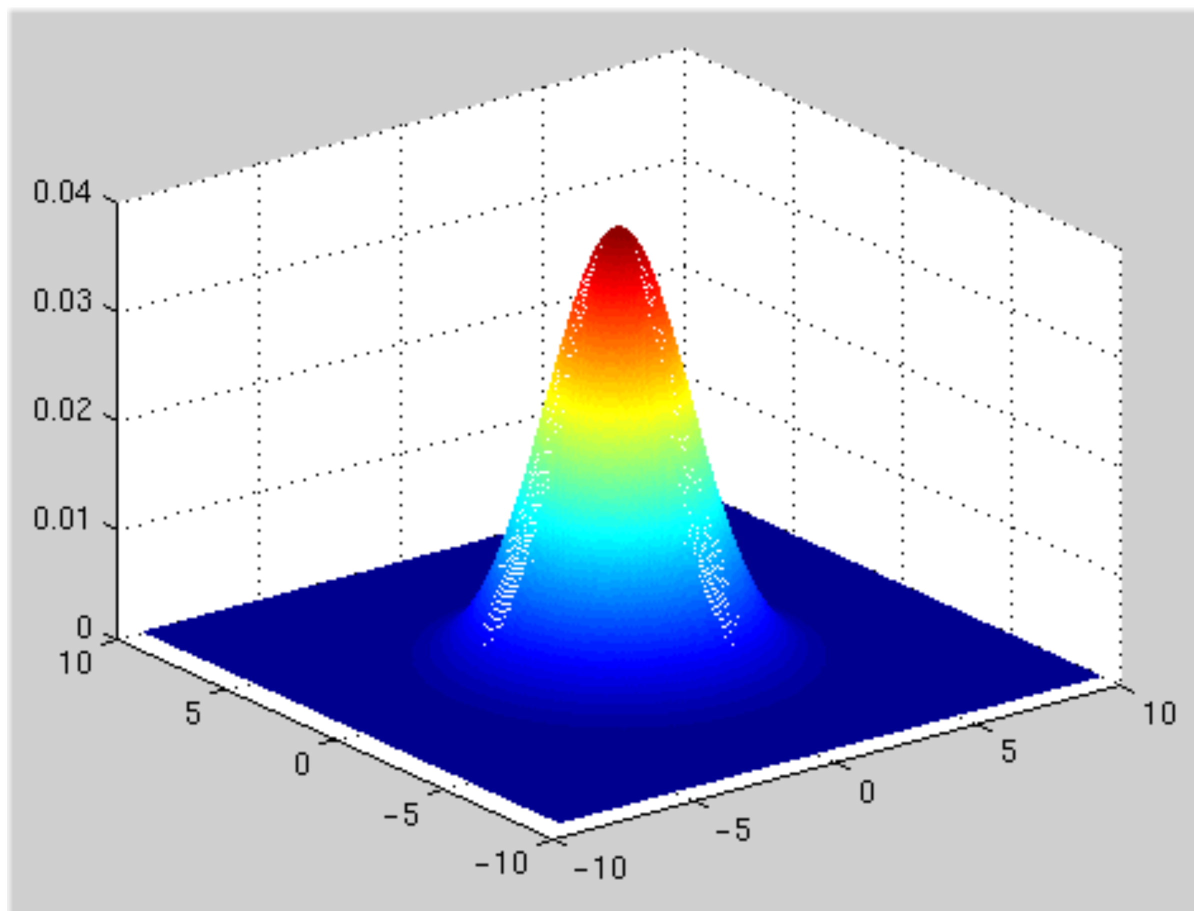
1-D:

$$g(x) = e^{-\frac{x^2}{2\sigma^2}}$$



2-D:

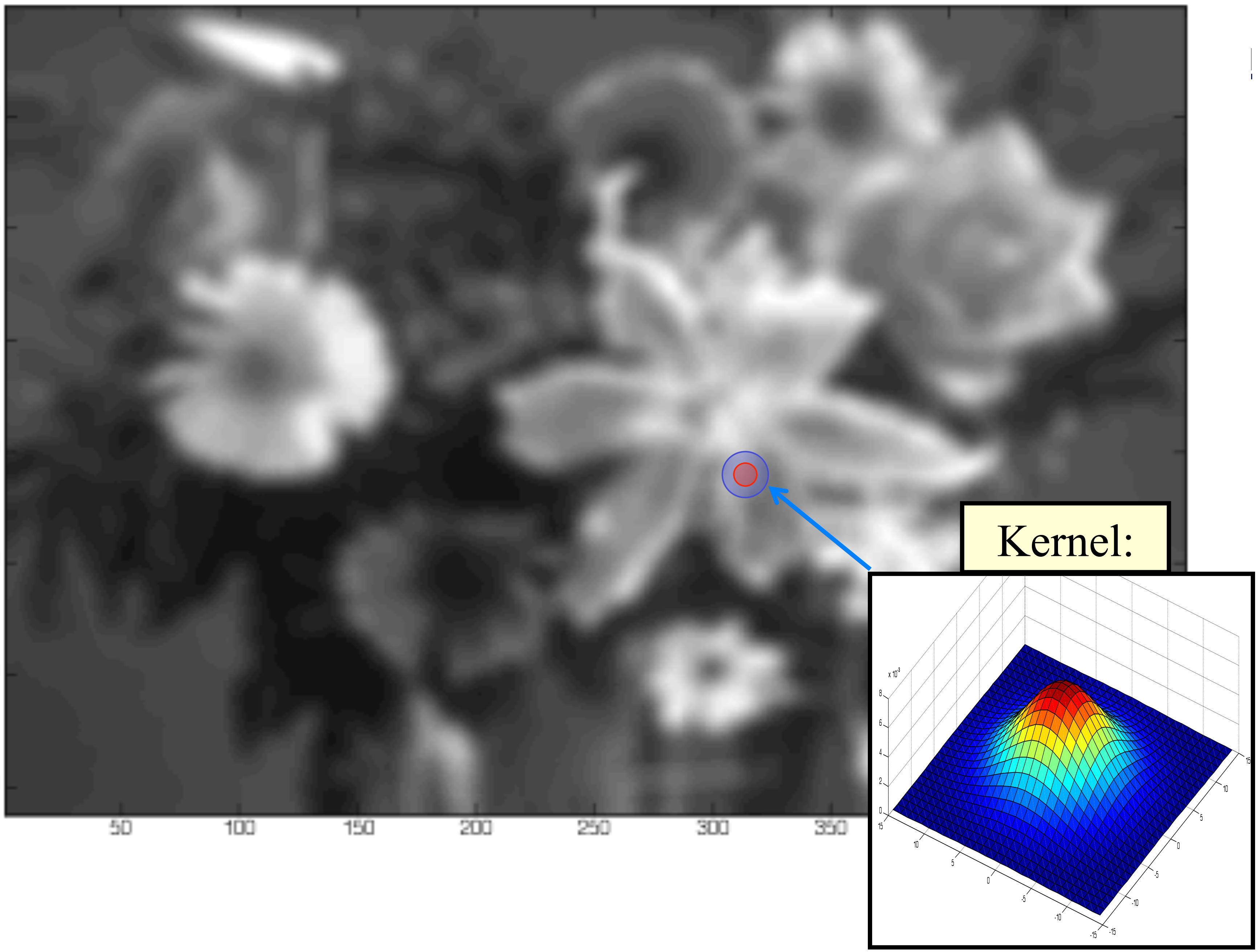
$$G(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}}$$



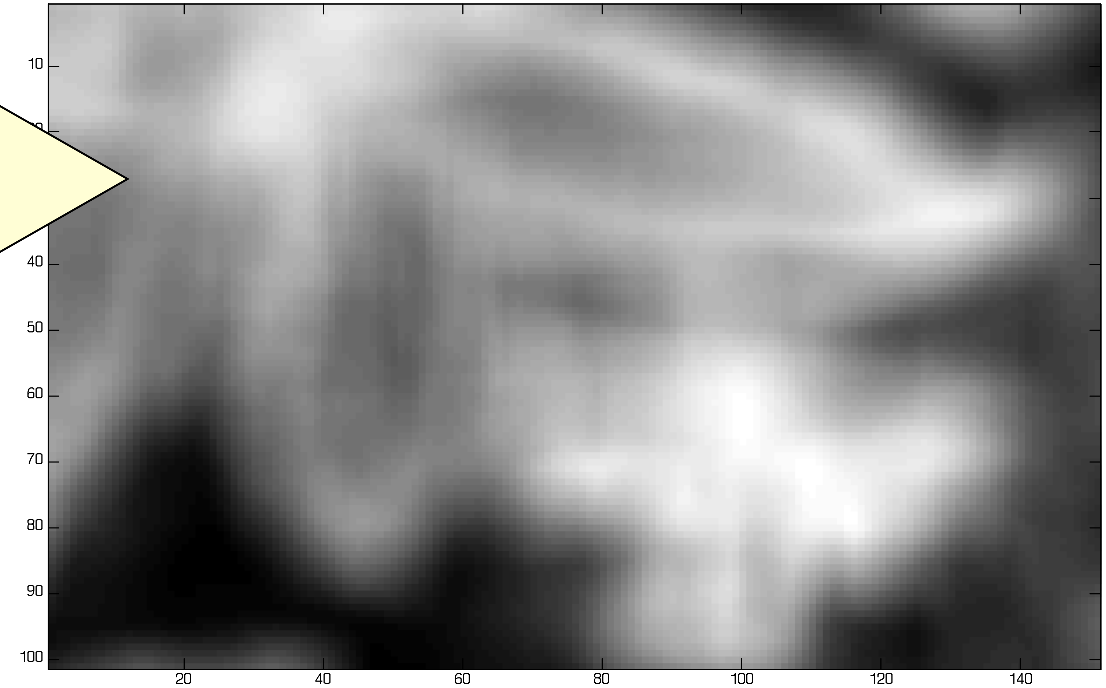
Slight abuse of notations:
We ignore the normalization constant such that

$$\int g(x)dx = 1$$

Gaussian Blurring, $\sigma = 5$



Simple Averaging



Gaussian Smoothing

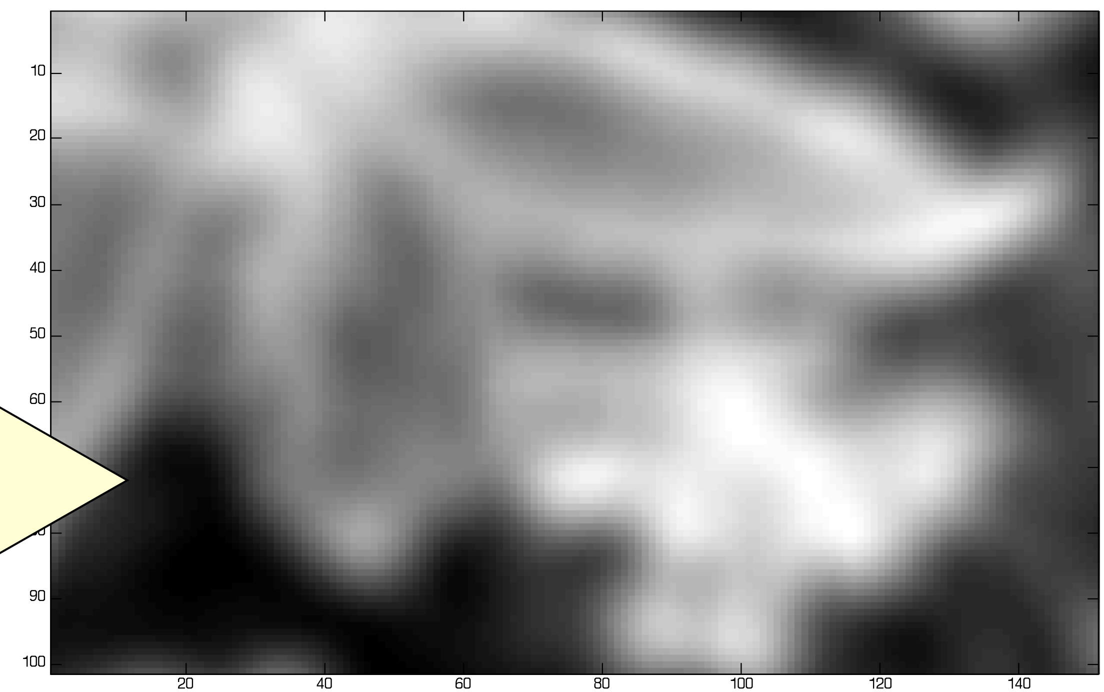
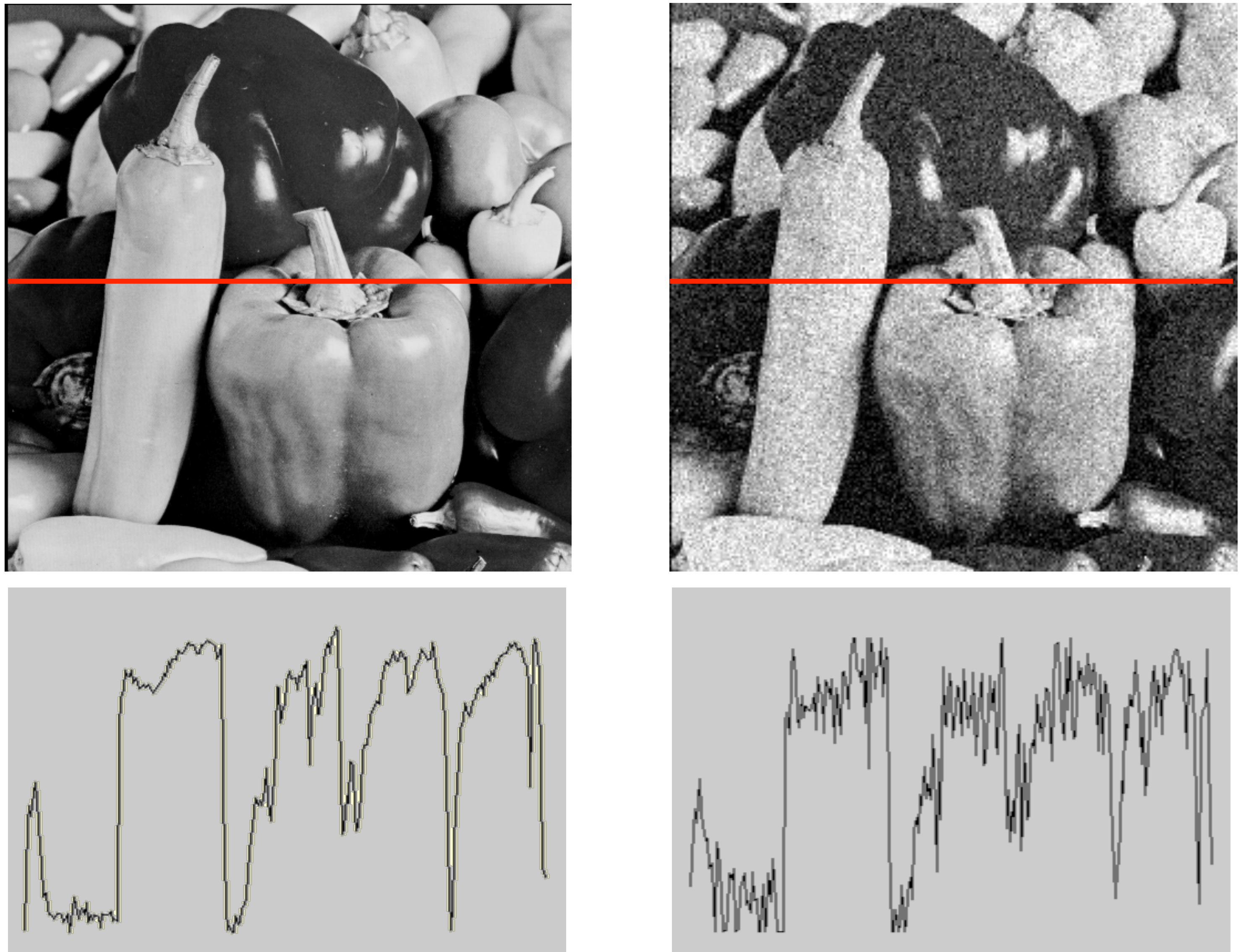


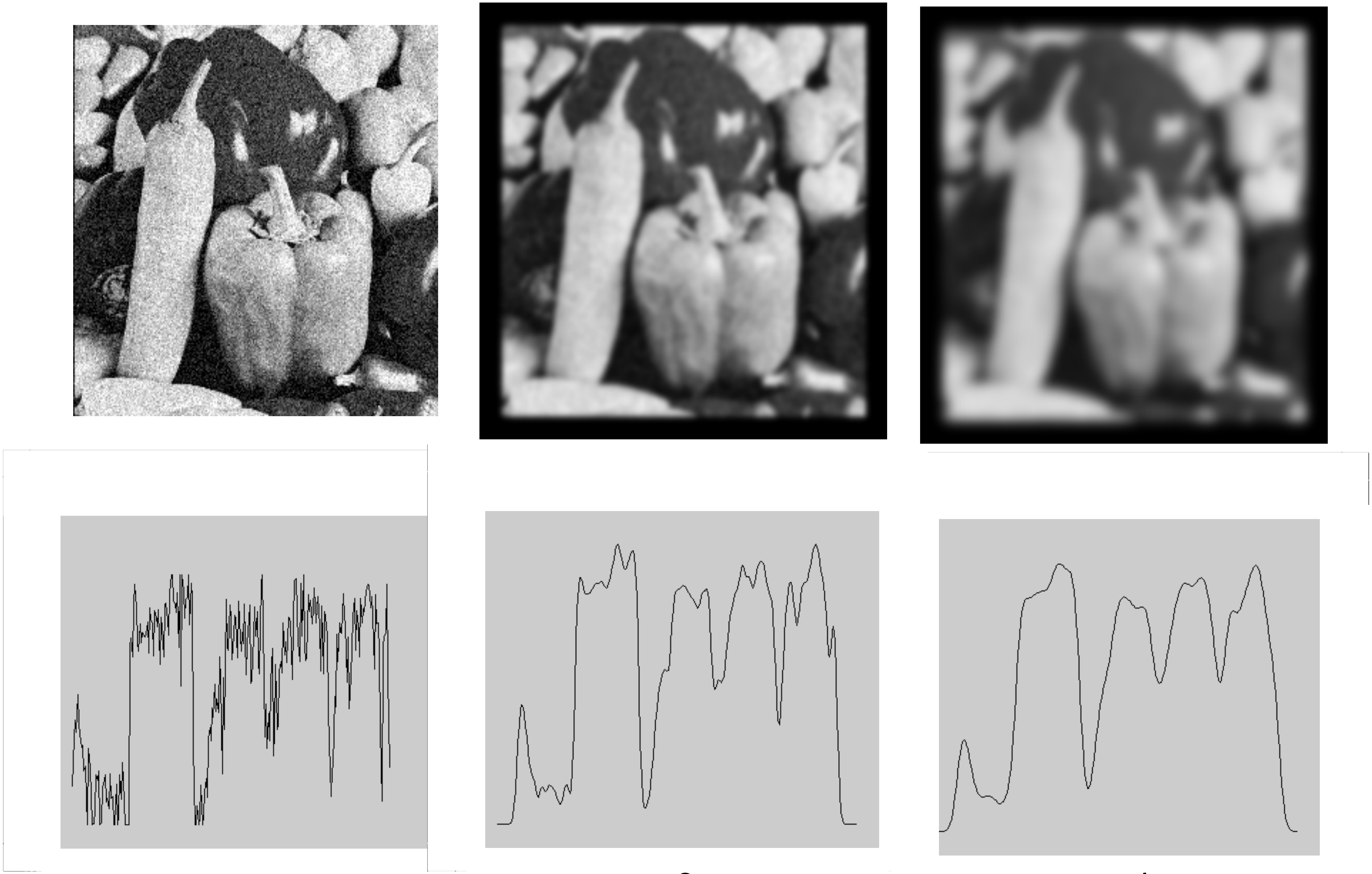
Image Noise



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

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

Gaussian Smoothing to Remove Noise



No smoothing $\sigma = 2$ $\sigma = 4$

Computational Issues for Filters: Separability and Moving Averages

$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{256}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

$$\frac{1}{K}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline \end{array} \qquad \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array} \qquad \frac{1}{16}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}$$

- When a *KxK* filter is equivalent to a *Kx1* and a *1xK* filter
- Reduces number of multiplies from K^2 to $2K$
- For box filter, moving average makes cost independent of K

Some Other (Nonlinear) Neighborhood Operators: Median Filter

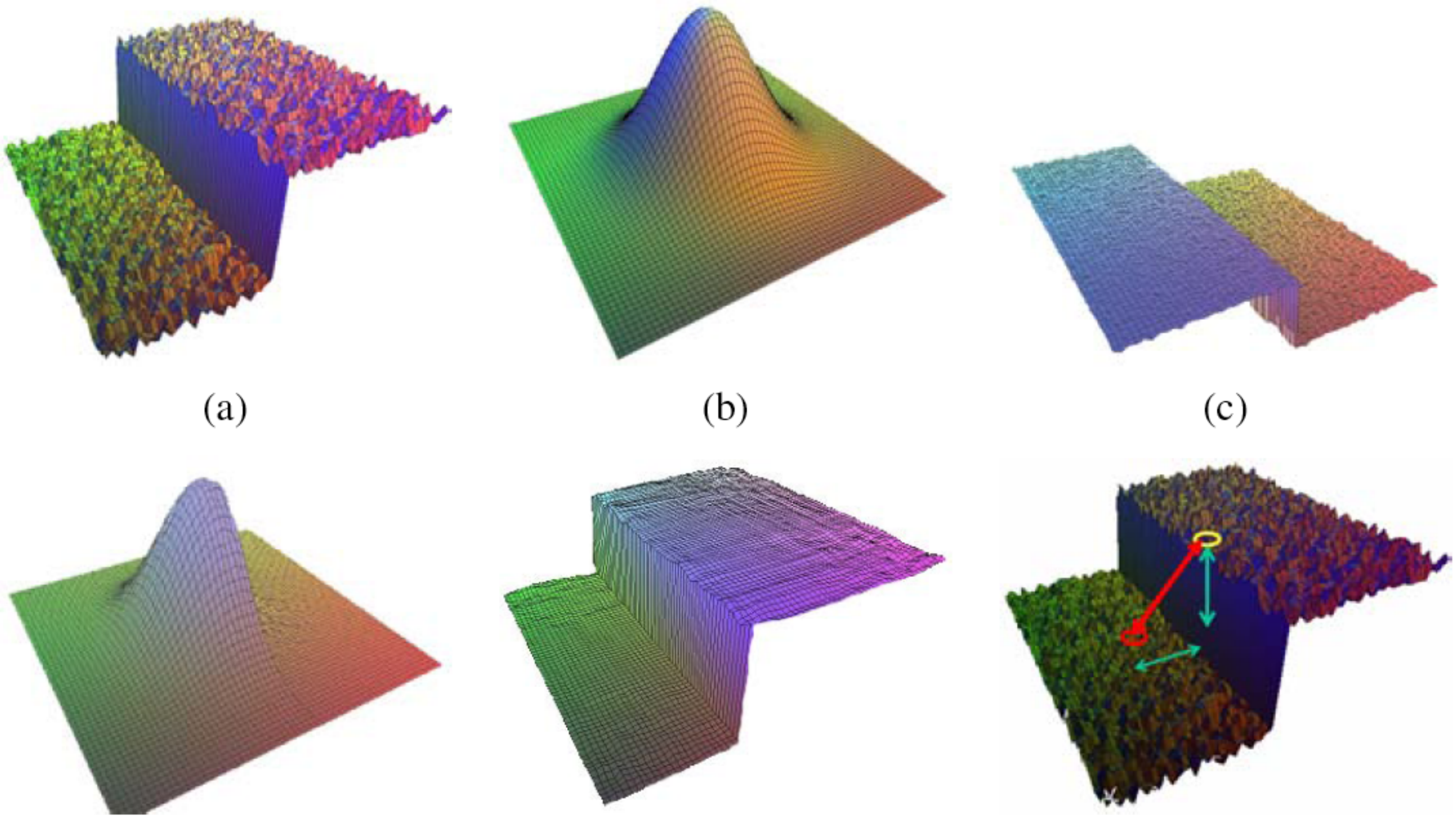
- Replace center pixel of KxK window with the mean value of all pixels in the window
- Good for removing large noise spikes without blurring image

Original image

Gaussian filtered

Median filtered

Some Other (Nonlinear) Neighborhood Operators: Bilateral Filter for Edge-Preserving Smoothing



(a) (b) (c)
(d) (e) (f)

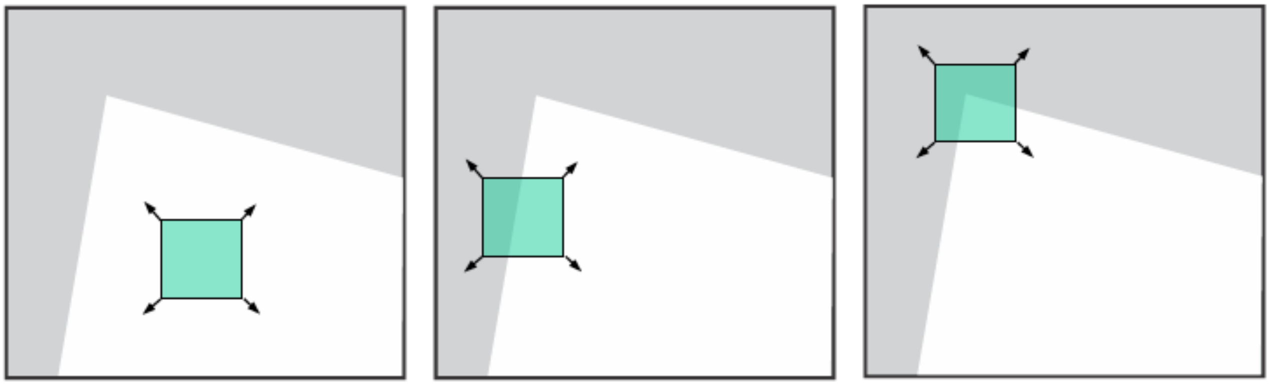
Some Other (Nonlinear) Neighborhood Operators: Bilateral Filter for Edge-Preserving Smoothing

Original image

15x15 box filter

15x15 bilateral filter

Other Useful Neighborhood Operators: Feature Detectors (Recall Earlier Lecture)



"flat" region:
no change in all
directions

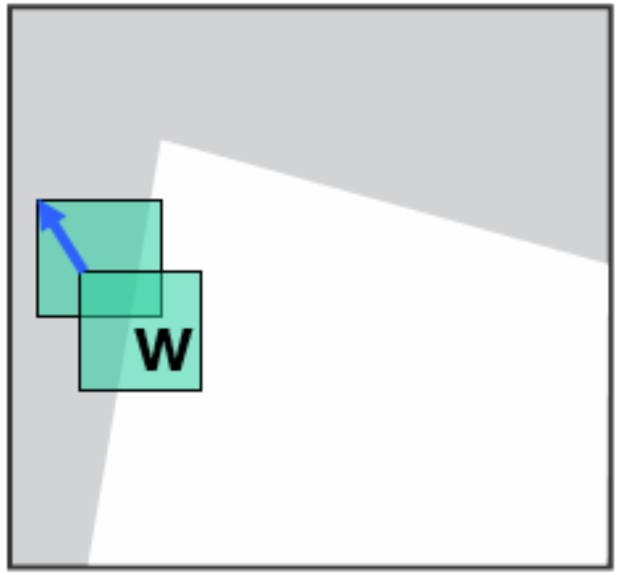
"edge":
no change along
the edge direction

"corner":
significant change
in all directions

$$H = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

(Implementation note: moving averages)

Other Useful Neighborhood Operators: Image Matching (Recall Earlier Lecture)



$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

- Sum squared difference (SSD) vs. sum absolute difference (SAD)
- Normalization: why? Many approaches.

Other Useful Neighborhood Operators

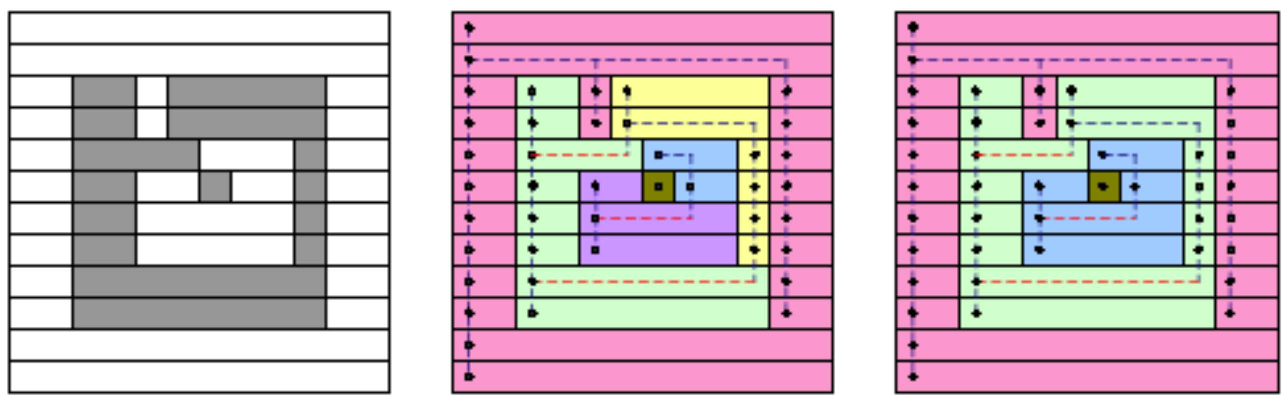
- Morphology

- Distance transform

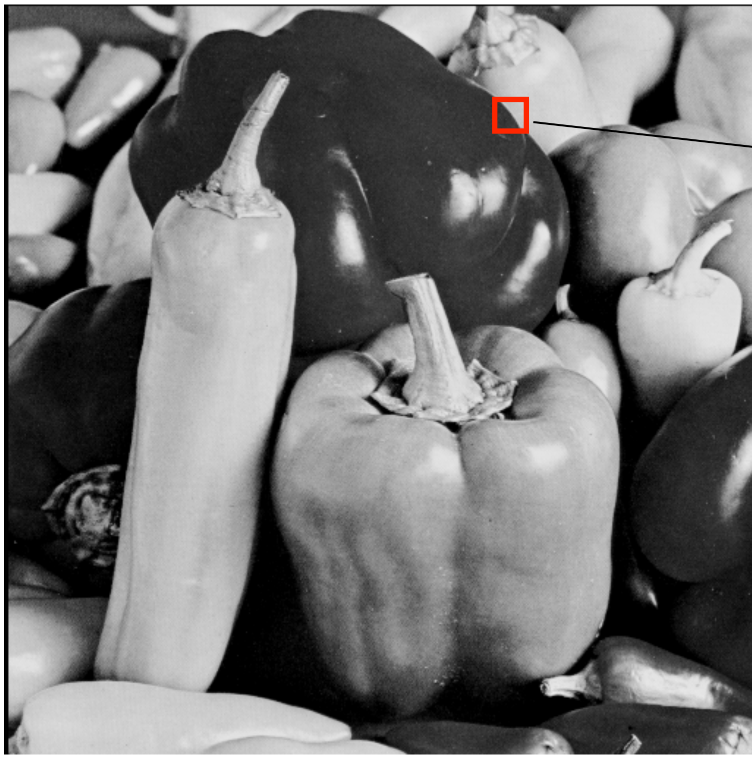
0	0	0	0	1	0	0
0	0	1	1	1	0	0
0	1	1	1	1	1	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

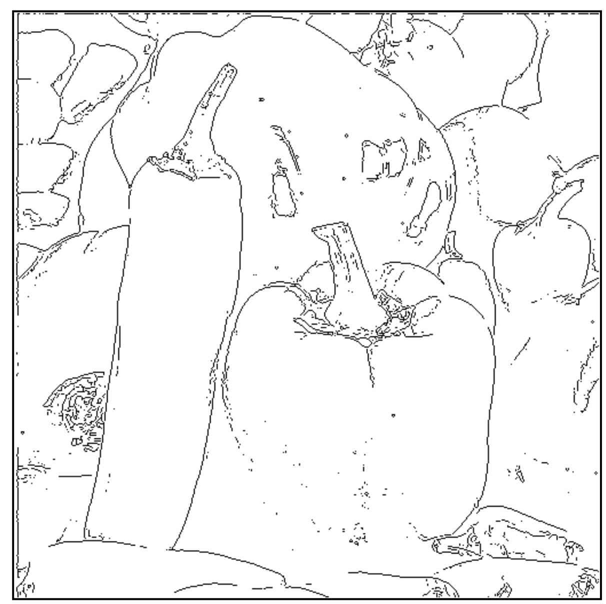
0	0	0	0	1	0	0
0	0	1	1	1	0	0
0	1	2	2	2	1	0
0	1	2	2	1	1	0
0	1	2	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

- Connected components



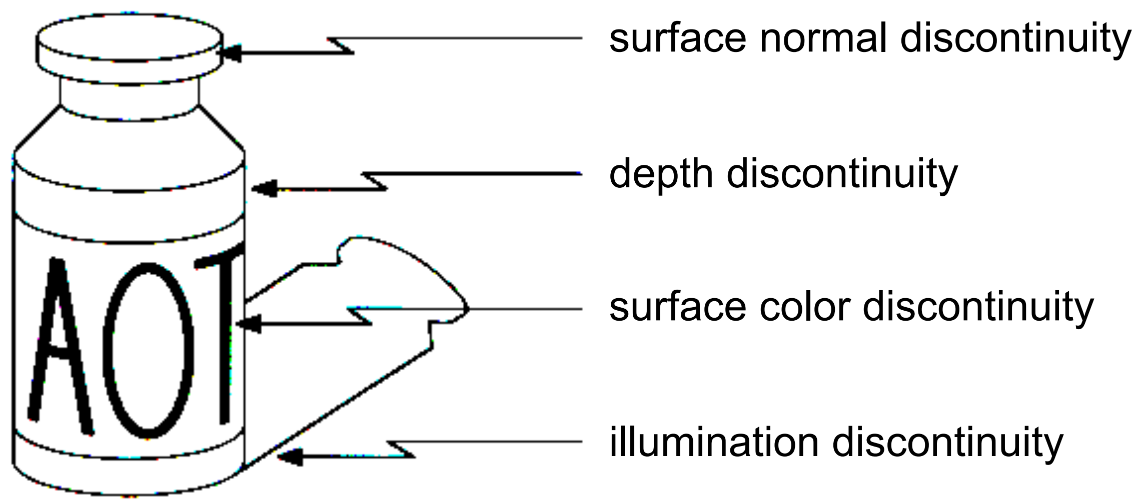
Edge Detection: What is an Edge?





- Local maxima of rate of change of intensity

Origin of Edges



- Many factors
- Sometimes care which factor applies; sometimes can determine that

Image Derivatives: 1-D Case

- We want to compute, at each pixel (x,y) the derivatives:
- In the discrete case we could take the difference between the left and right pixels:

$$\frac{\partial I}{\partial x} \approx I(i+1,j) - I(i-1,j)$$

- Convolution of the image by

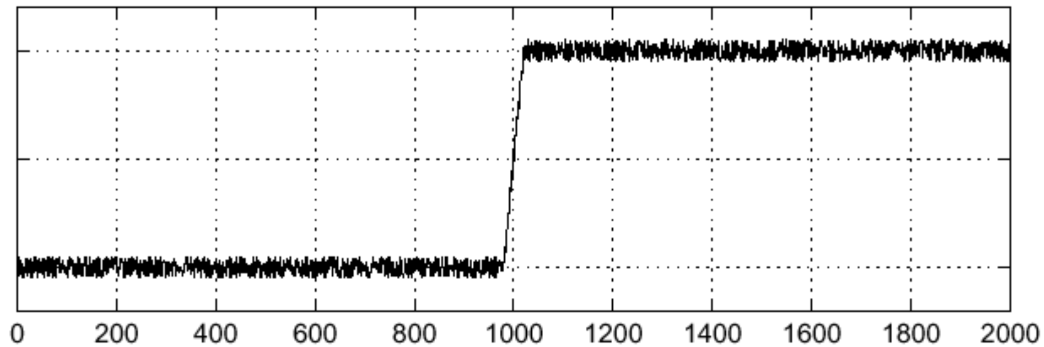
$$\partial_x = \boxed{-1 \mid 0 \mid 1}$$

- Problem: Increases noise

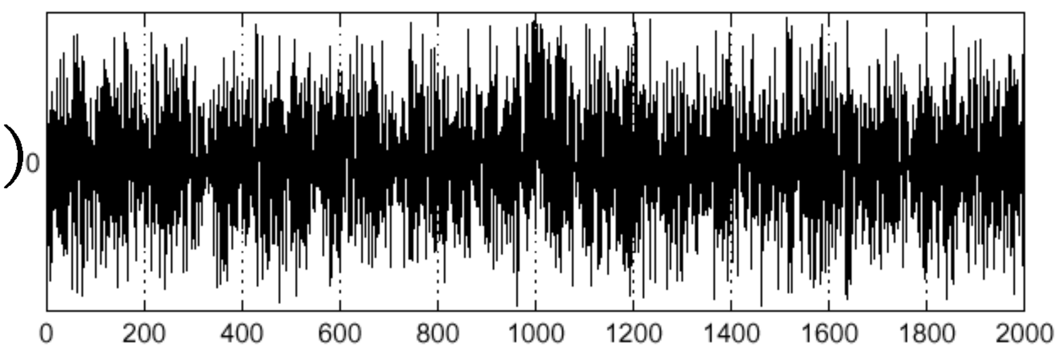
$$\underbrace{I(i+1,j) - I(i-1,j)}_{\text{Difference between Actual image values}} = \underbrace{\hat{I}(i+1,j) - \hat{I}(i-1,j)}_{\text{True difference (derivative)}} \underbrace{+ n_+ + n_-}_{\text{Twice the amount of noise as in the original image}}$$

Edges in 1-D:
Effects of Noise

$f(x)$



$\frac{d}{dx}f(x)$



Where is the edge?

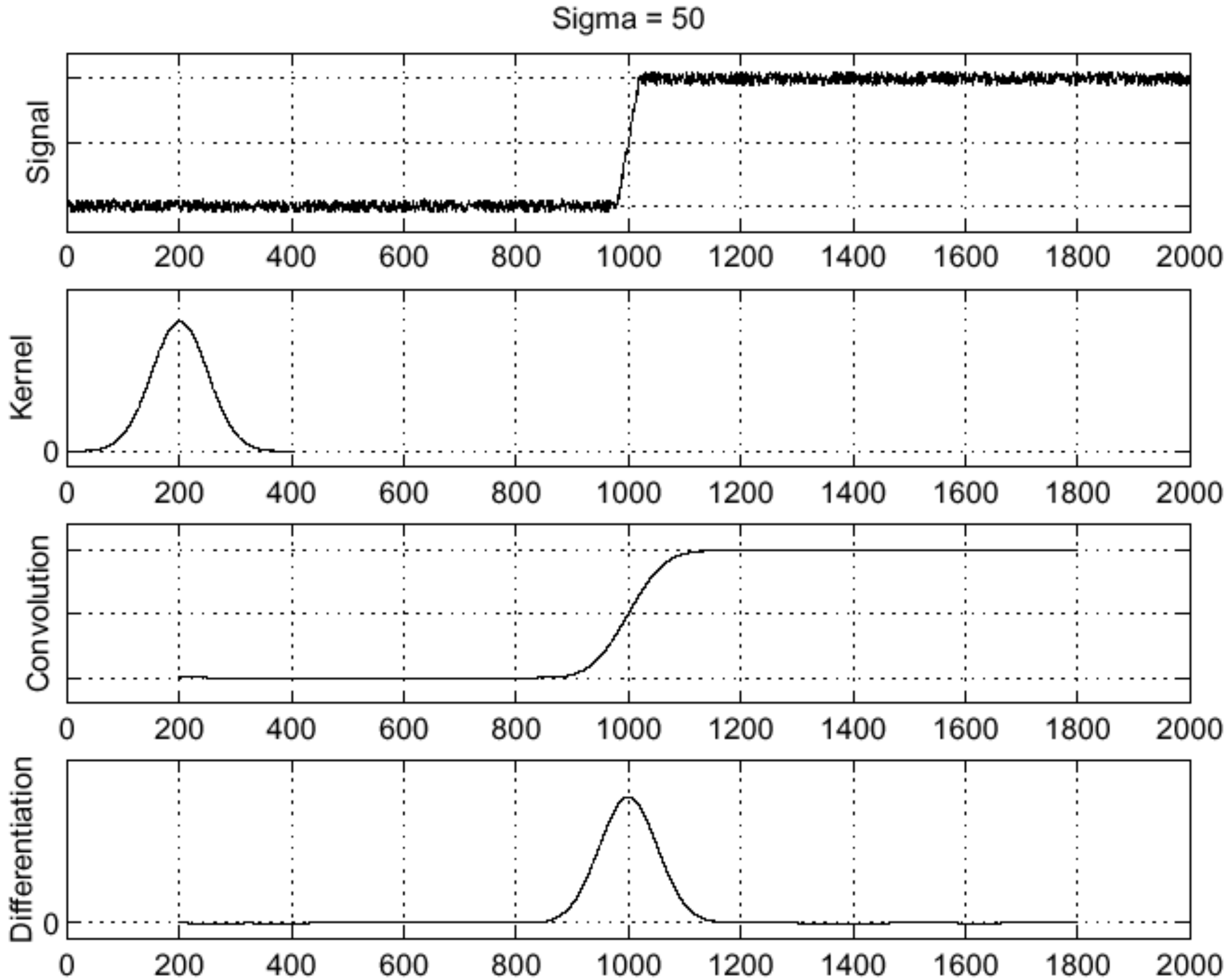
Solution: Smooth First

$f(x)$

$h(x)$

$h * f$

$\frac{\partial}{\partial x}(h * f)$



Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h * f)$

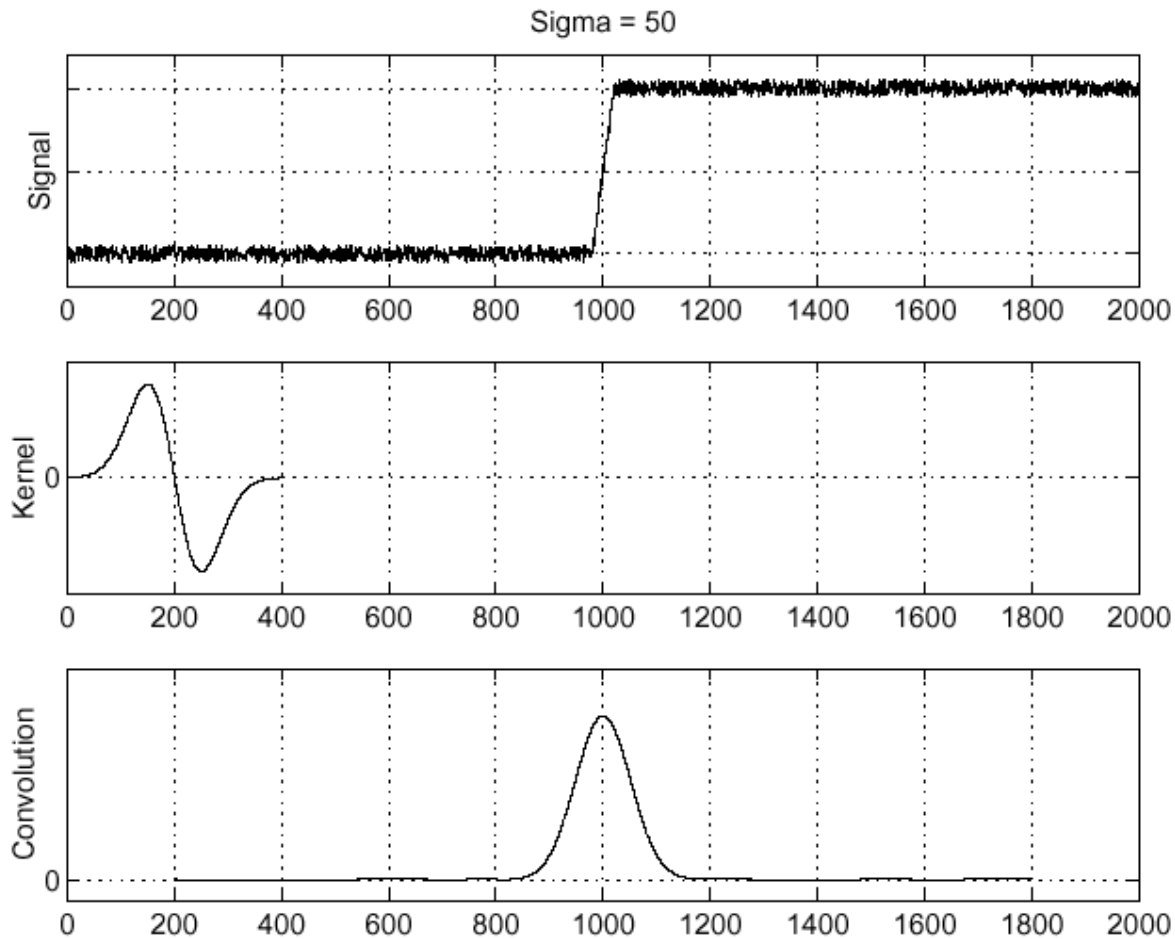
Derivative Property of Convolution: Saves One Step

$$\frac{\partial}{\partial x}(h * f) = \left(\frac{\partial}{\partial x} h\right) * f$$

$f(x)$

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

$\left(\frac{\partial}{\partial x} h\right) * f$

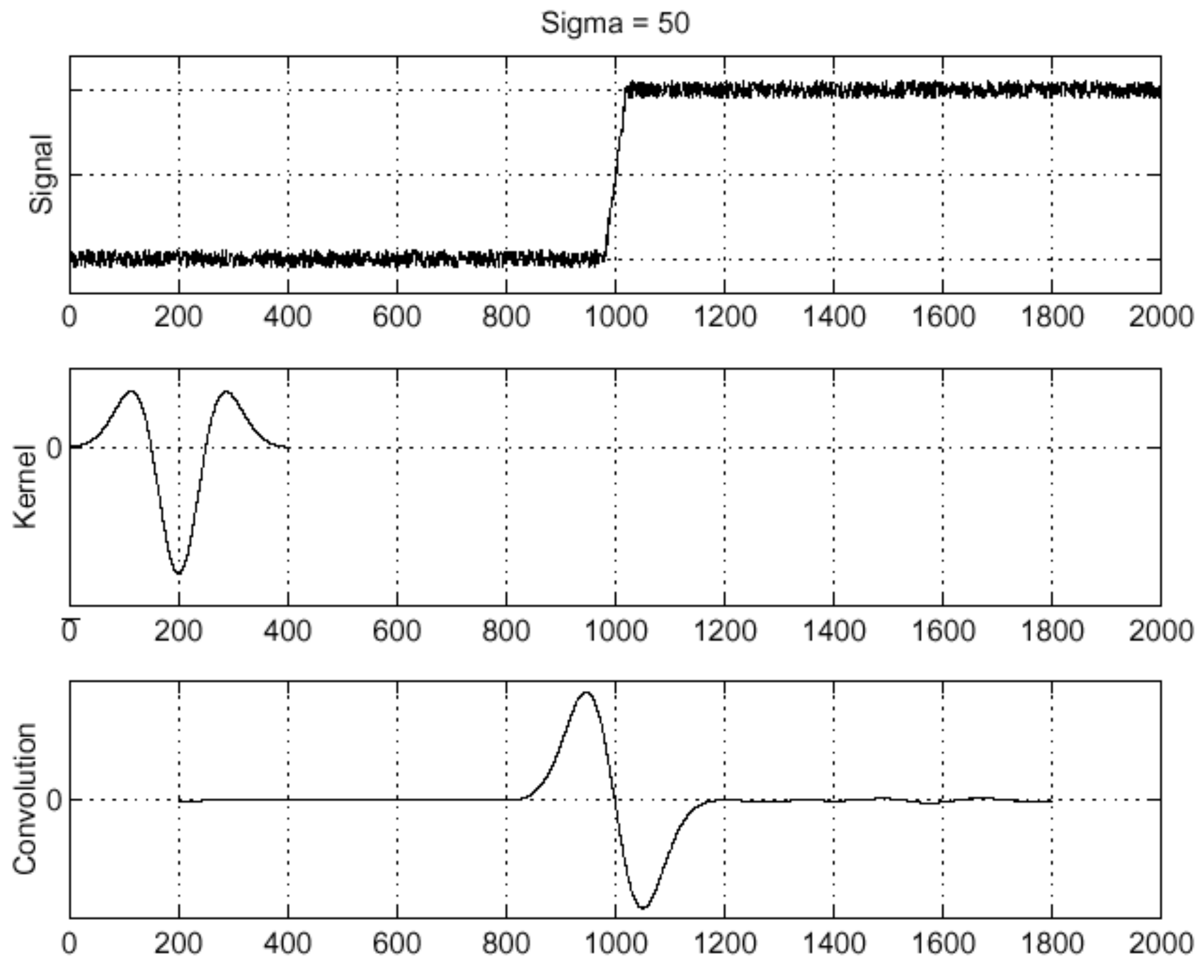


Laplacian of Gaussian

$$f(x)$$

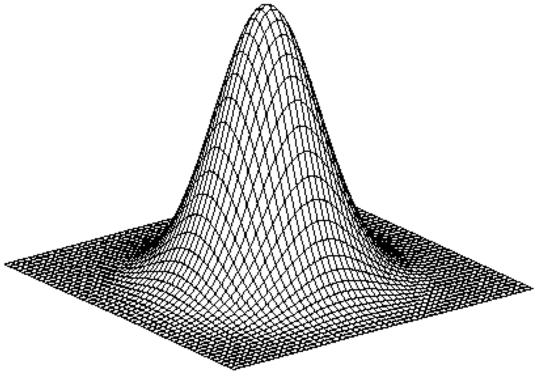
$$\frac{\partial^2}{\partial x^2} h$$

$$\left(\frac{\partial^2}{\partial x^2} h\right) * f$$



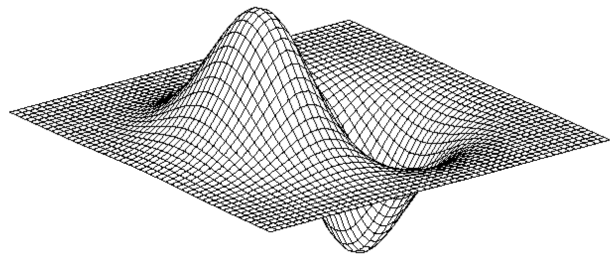
Where is the edge? Zero crossing of bottom graph

2-D Edge Detection Filters



Gaussian

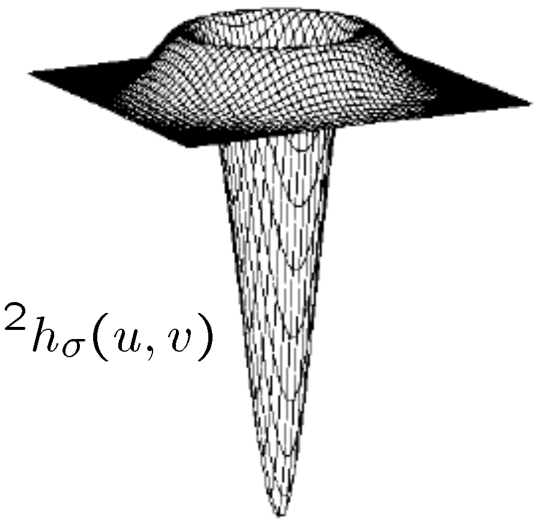
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian



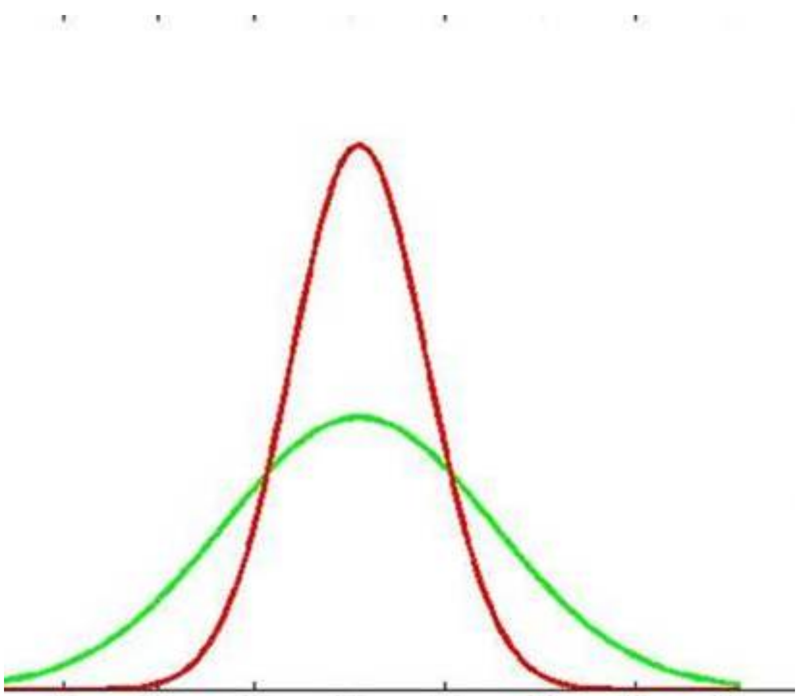
$$\nabla^2 h_\sigma(u,v)$$

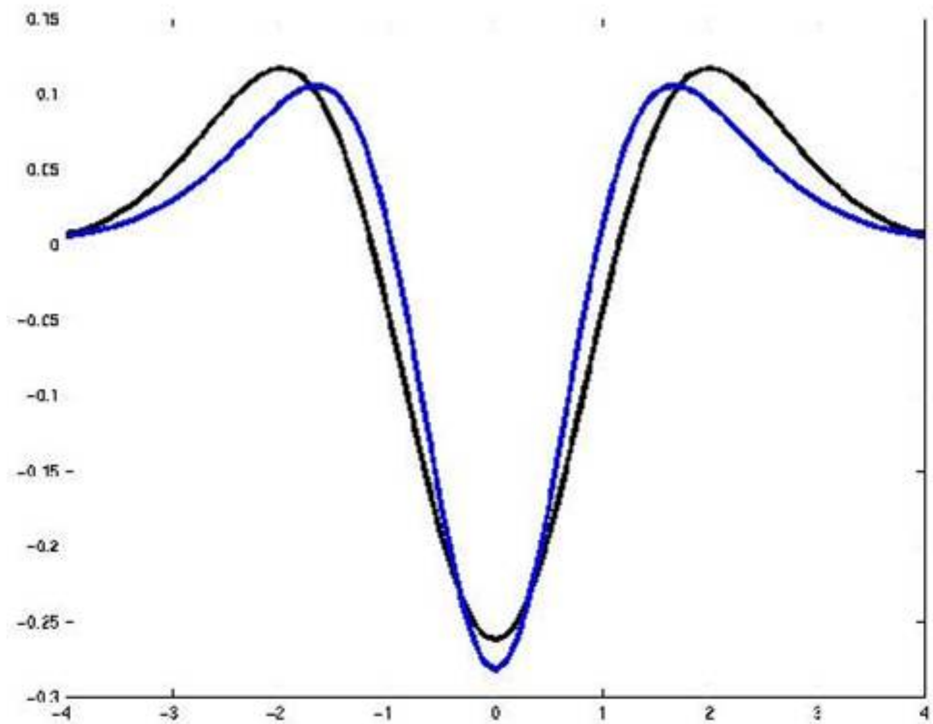
∇^2 is the **Laplacian** operator:

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

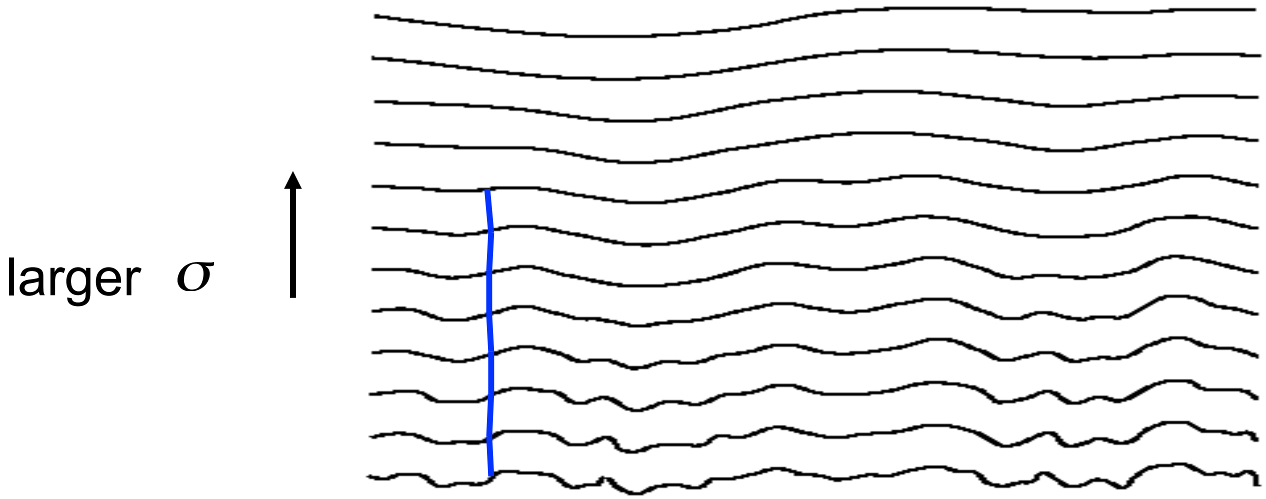
DOG Approximation to LOG

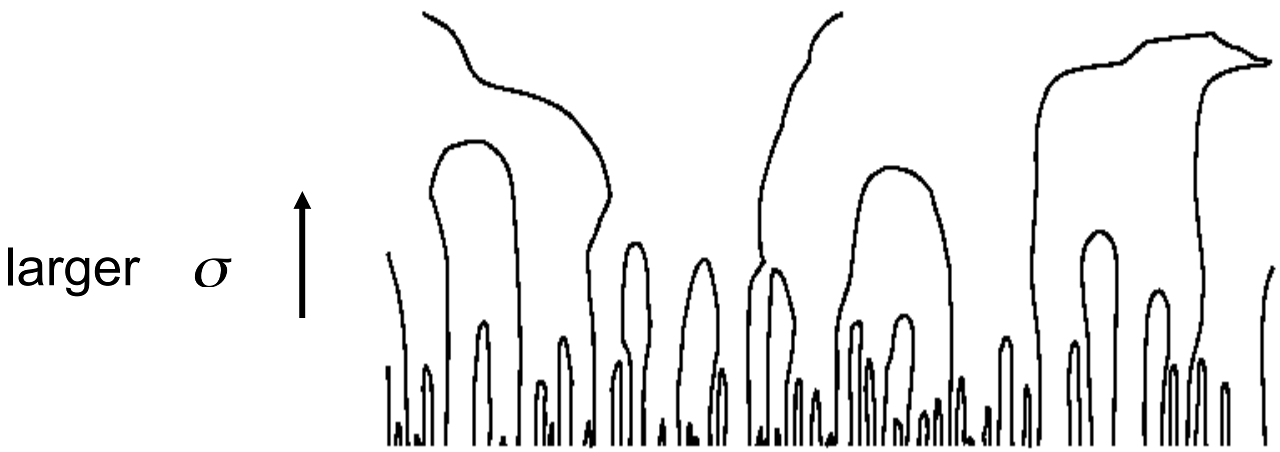
$$\nabla^2 G_\sigma \approx G_{\sigma_1} - G_{\sigma_2}$$



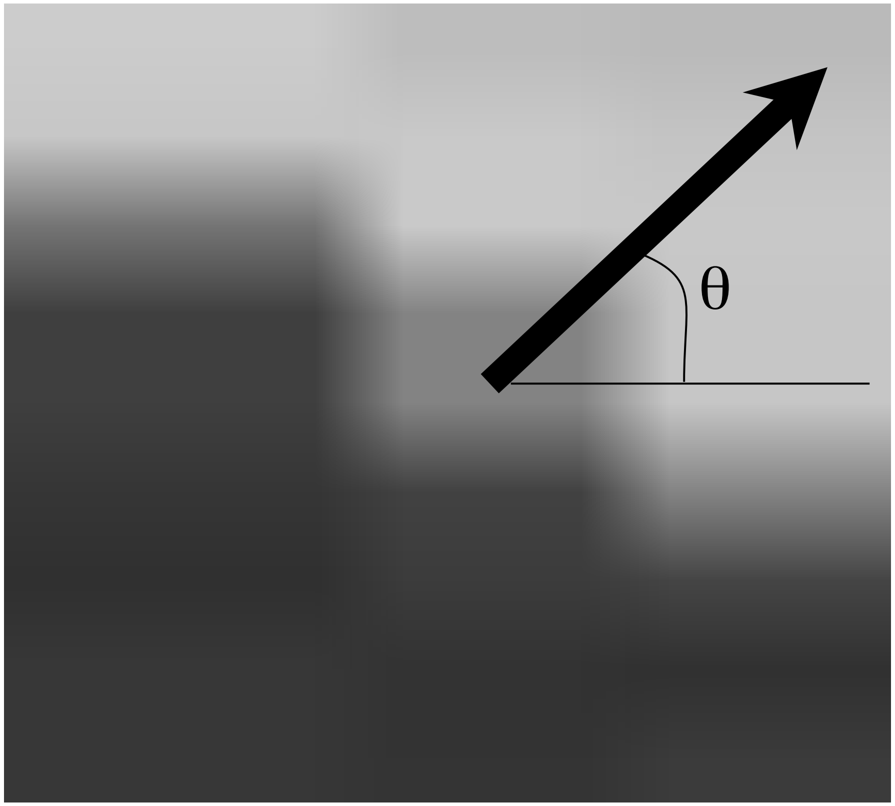


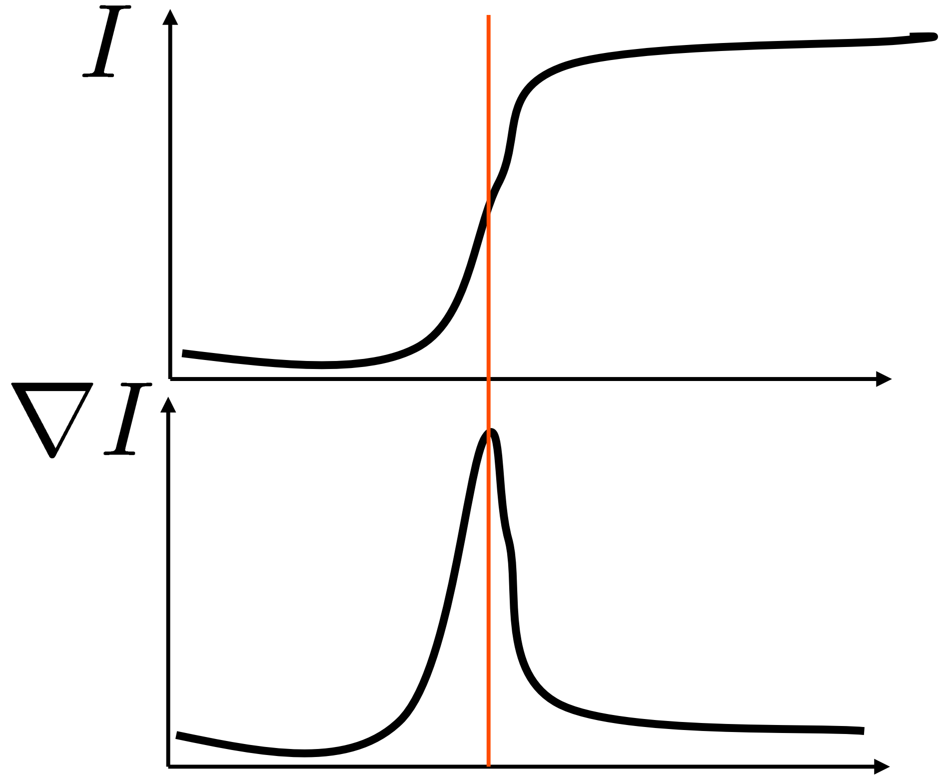
The Effect of Scale on Edge Detection





Scale space (Witkin 83)





Edge pixels are at local maxima of gradient magnitude
Gradient computed by convolution with Gaussian derivatives
Gradient direction is always perpendicular to edge direction

$$\frac{\partial I}{\partial x} = G_\sigma^x * I \qquad \frac{\partial I}{\partial y} = G_\sigma^y * I$$

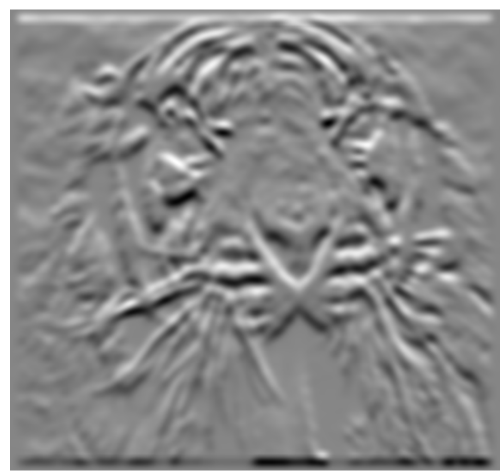
$$|\nabla I| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \quad \theta = atan2\left(\frac{\partial I}{\partial y}, \frac{\partial I}{\partial x}\right)$$

Applying the Gradient Magnitude Operator

$$I$$

$$|\nabla I| = \sqrt{\frac{\partial I}{\partial x}^2 + \frac{\partial I}{\partial y}^2}$$

$$\frac{\partial I}{\partial x}$$



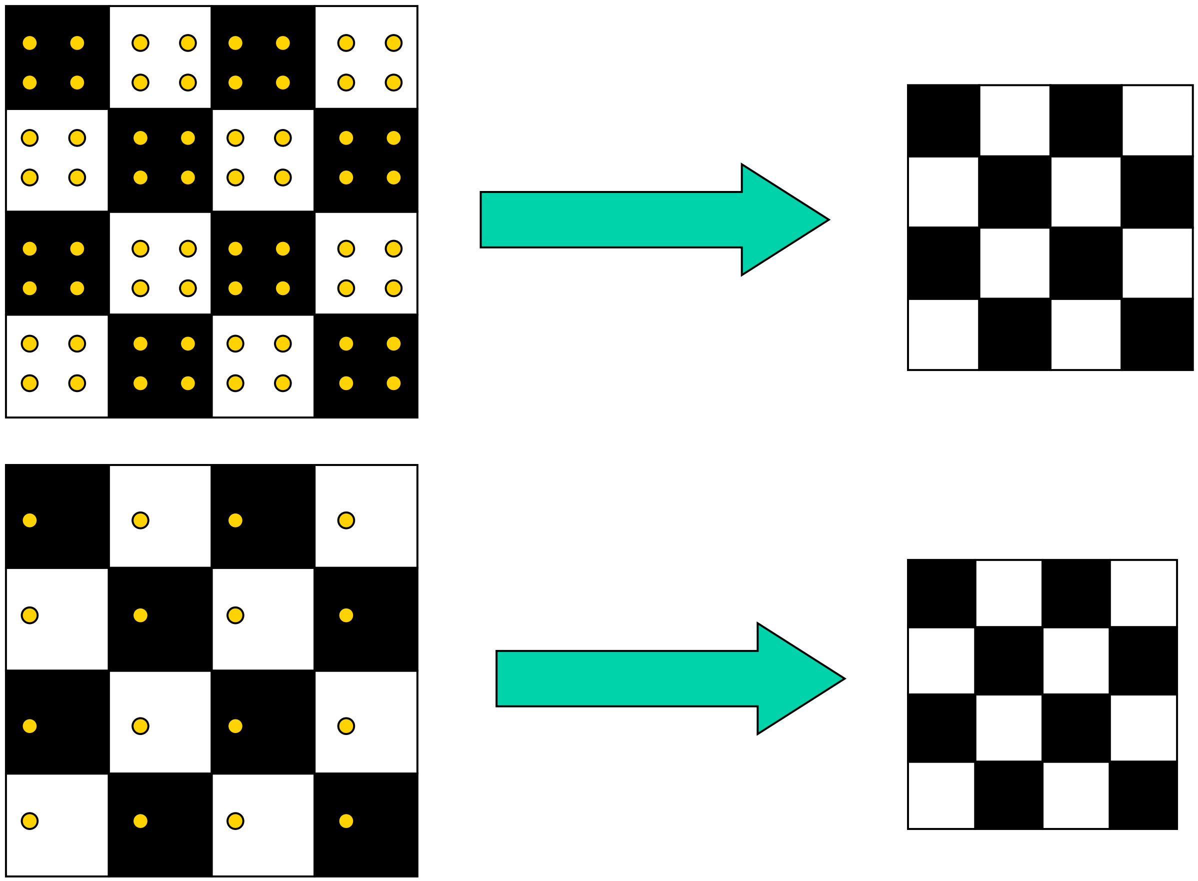
$$\frac{\partial I}{\partial y}$$

Different Thresholds Applied to the Gradient Magnitude

Additional steps:

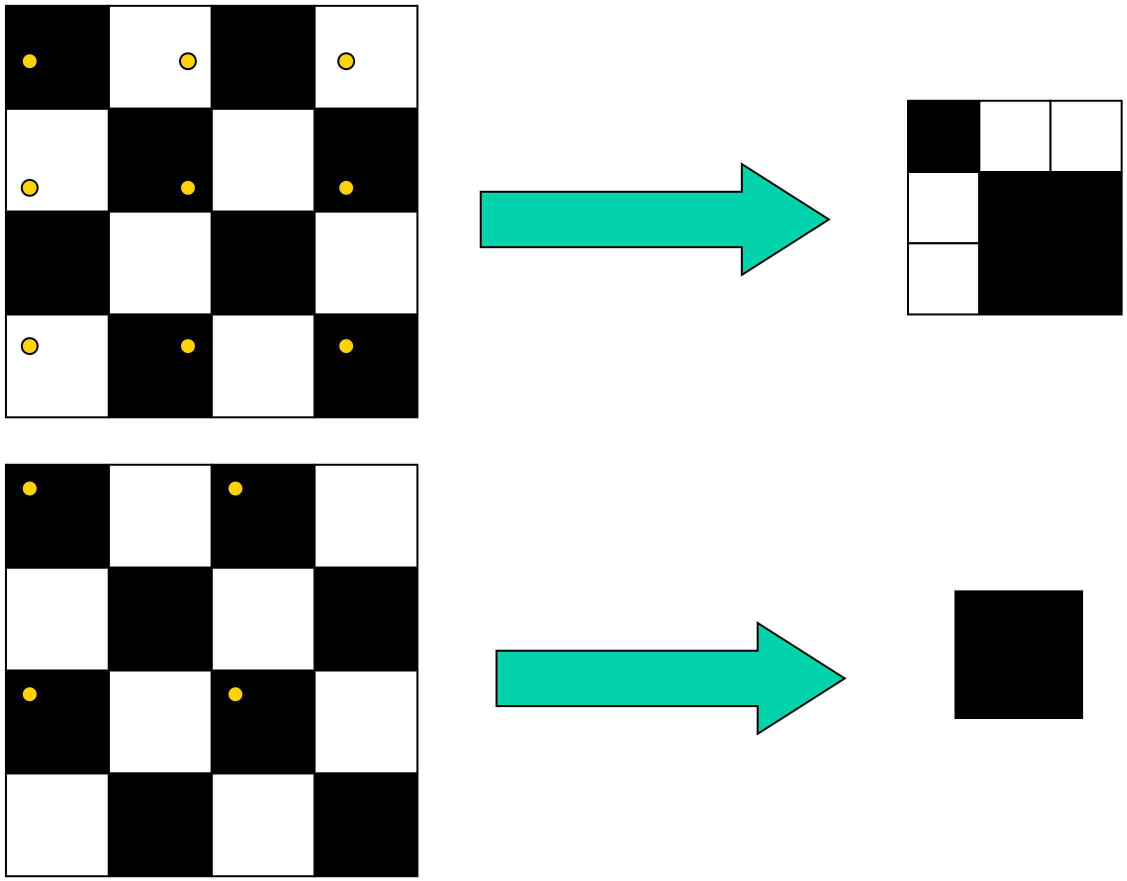
- Thresholding with hysteresis
- Thinning (non-maximum suppression)
- Linking

Sampling an Image



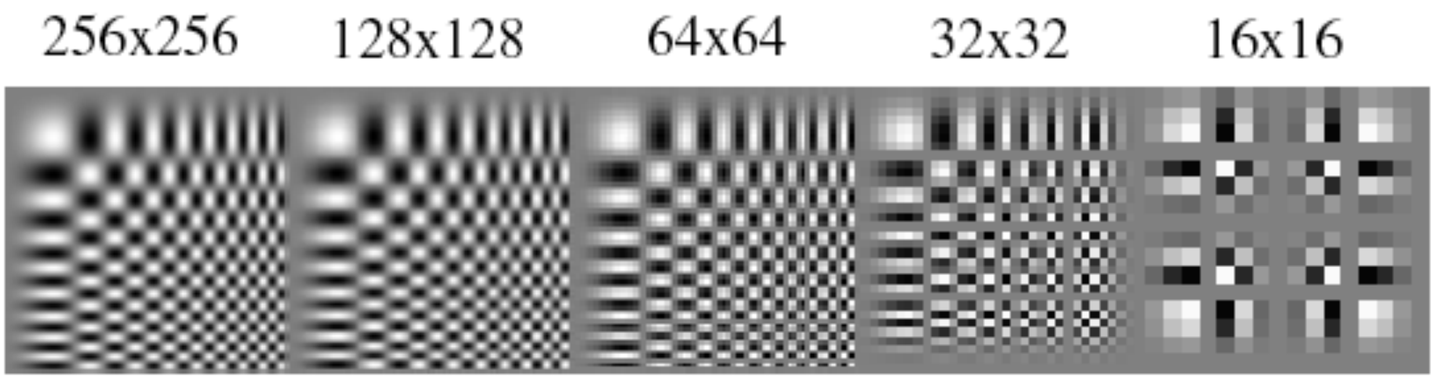
Examples of GOOD sampling

Undersampling and Aliasing



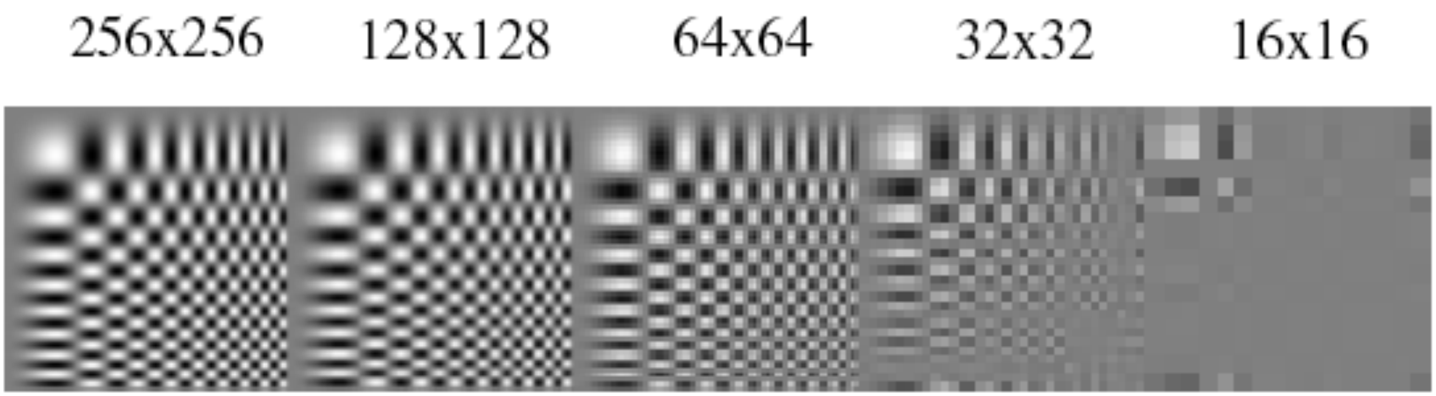
Examples of BAD sampling -> Aliasing

Downsampling and Aliasing



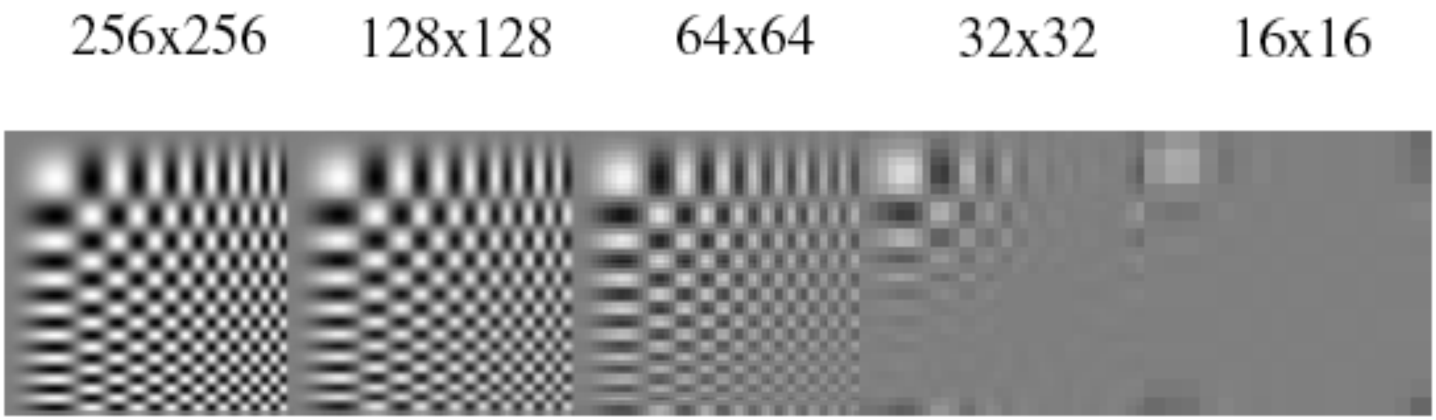
Sample every other pixel to go left to right

Downsampling with Smoothing (Gaussian, 1 Sigma)



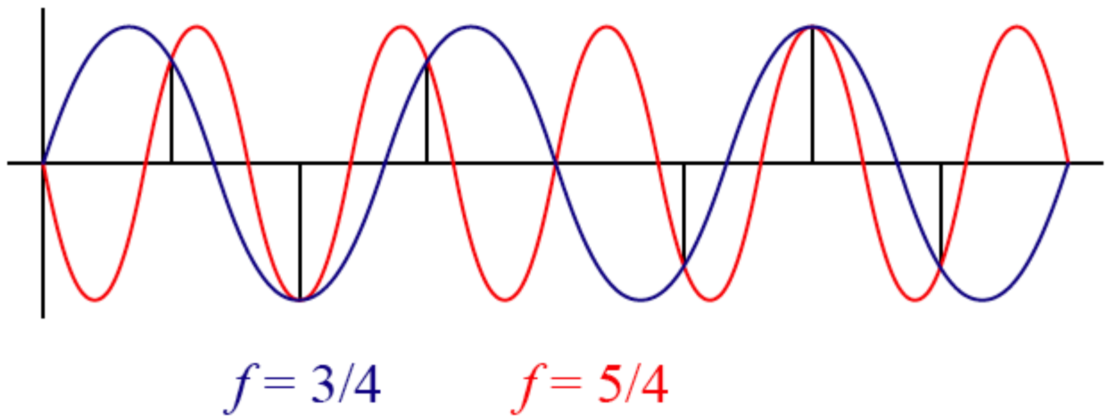
Sample every other pixel to go left to right

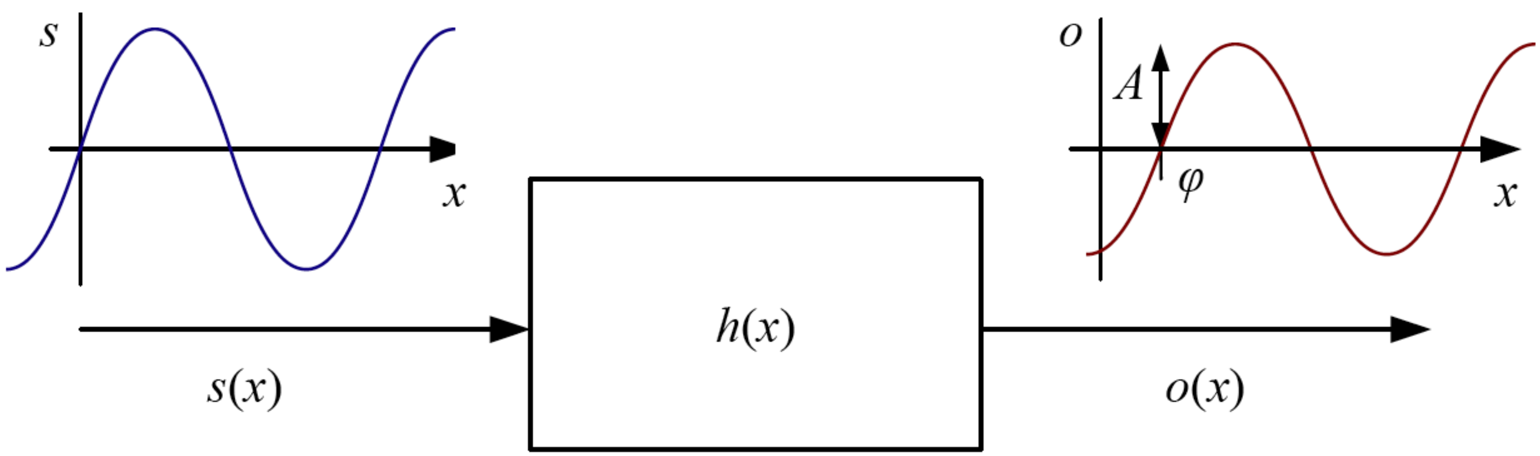
Downsampling with Smoothing (Gaussian, 1.4 Sigma)



Sample every other pixel to go left to right

Intuitive Introduction to Fourier Transforms

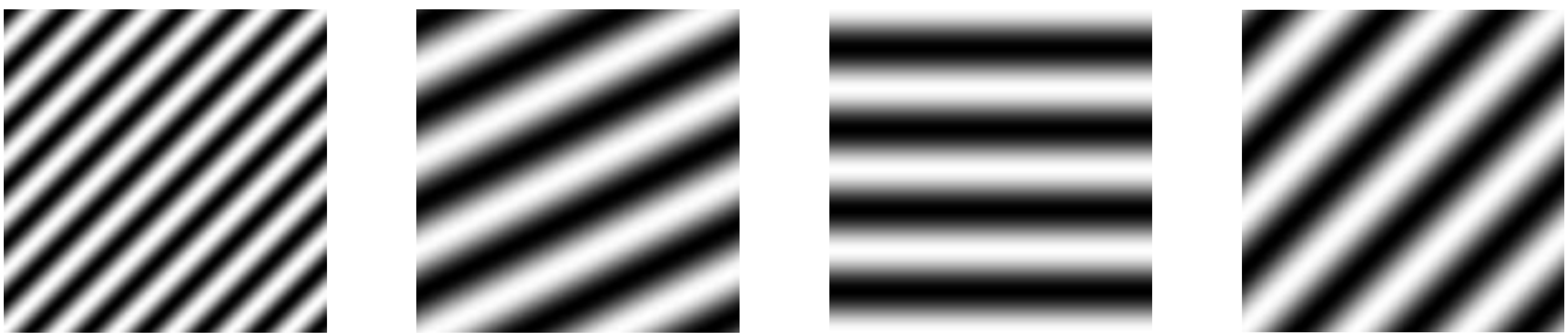




Intuition for Two Dimensions

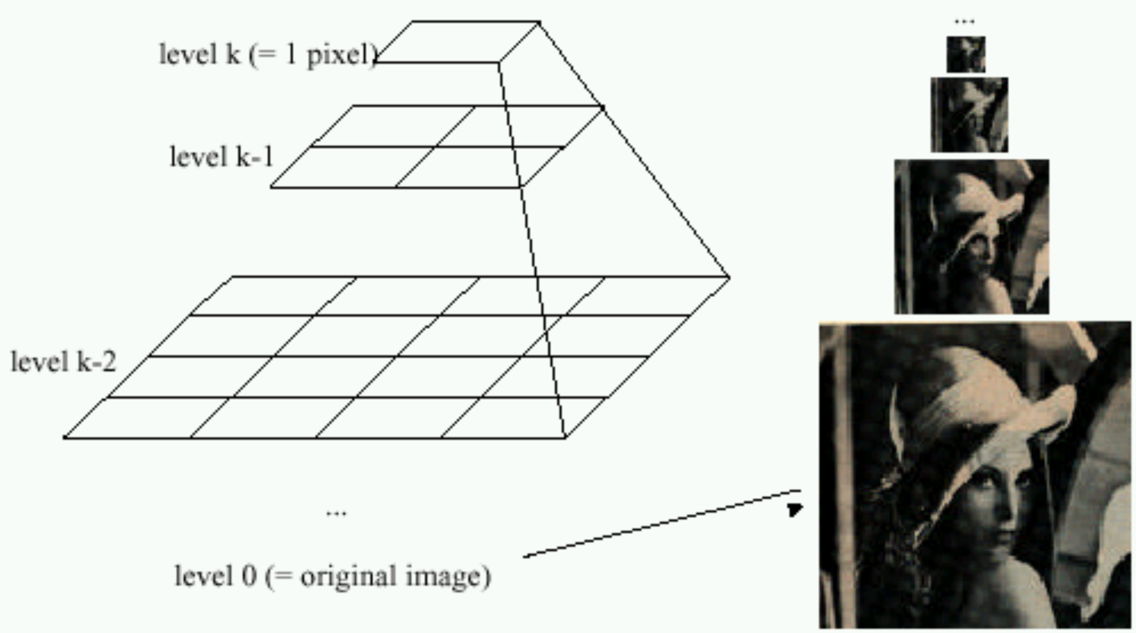
"dot"

= A measure of image content at this frequency and orientation



Multiresolution Image Representations

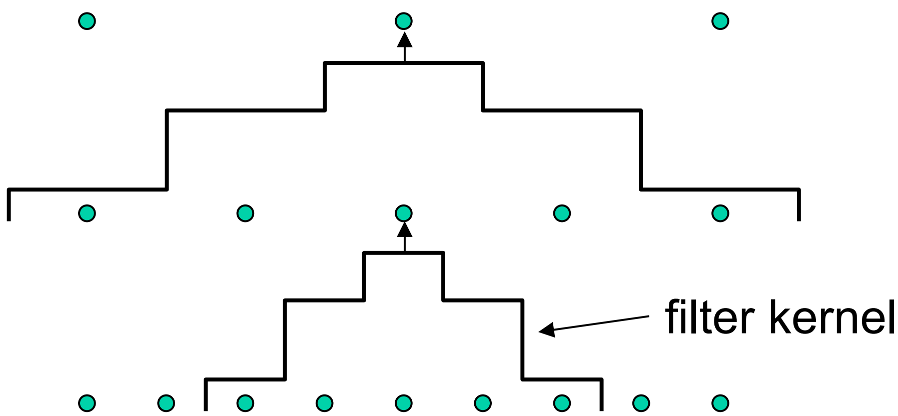
Idea: Represent NxN image as a "pyramid" of 1×1, 2×2, 4×4,…, $2^k\times2^k$ images (assuming $N=2^k$)



Known as a **Gaussian Pyramid** [Burt and Adelson, 1983]

Gaussian Pyramids have all sorts of applications in computer vision

Gaussian Pyramid Construction



Repeat

- Filter
- Subsample

Until minimum resolution reached

- can specify desired number of levels (e.g., 3-level pyramid)

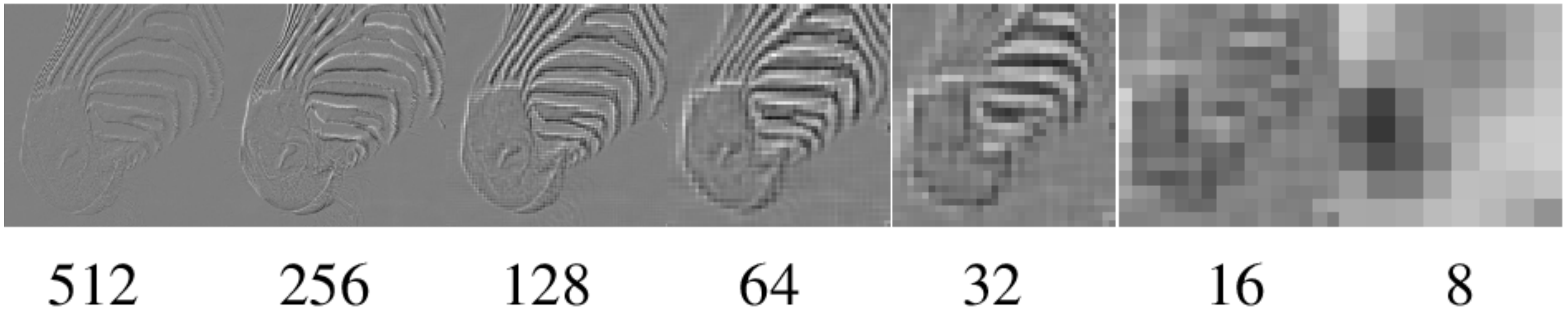
The whole pyramid is only 4/3 the size of the original image

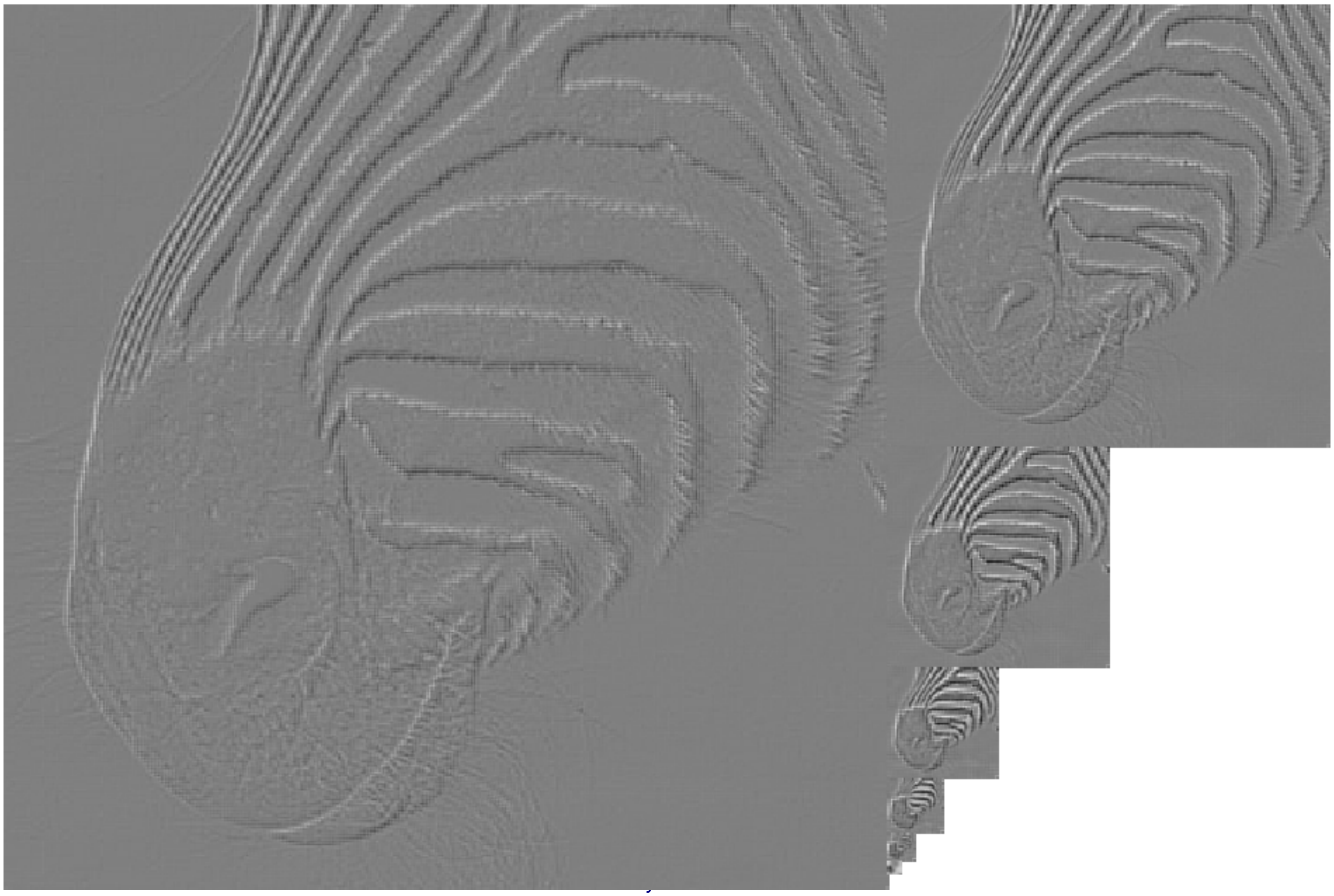
512 256 128 64 32 16 8

Slides by D.A. Forsyth

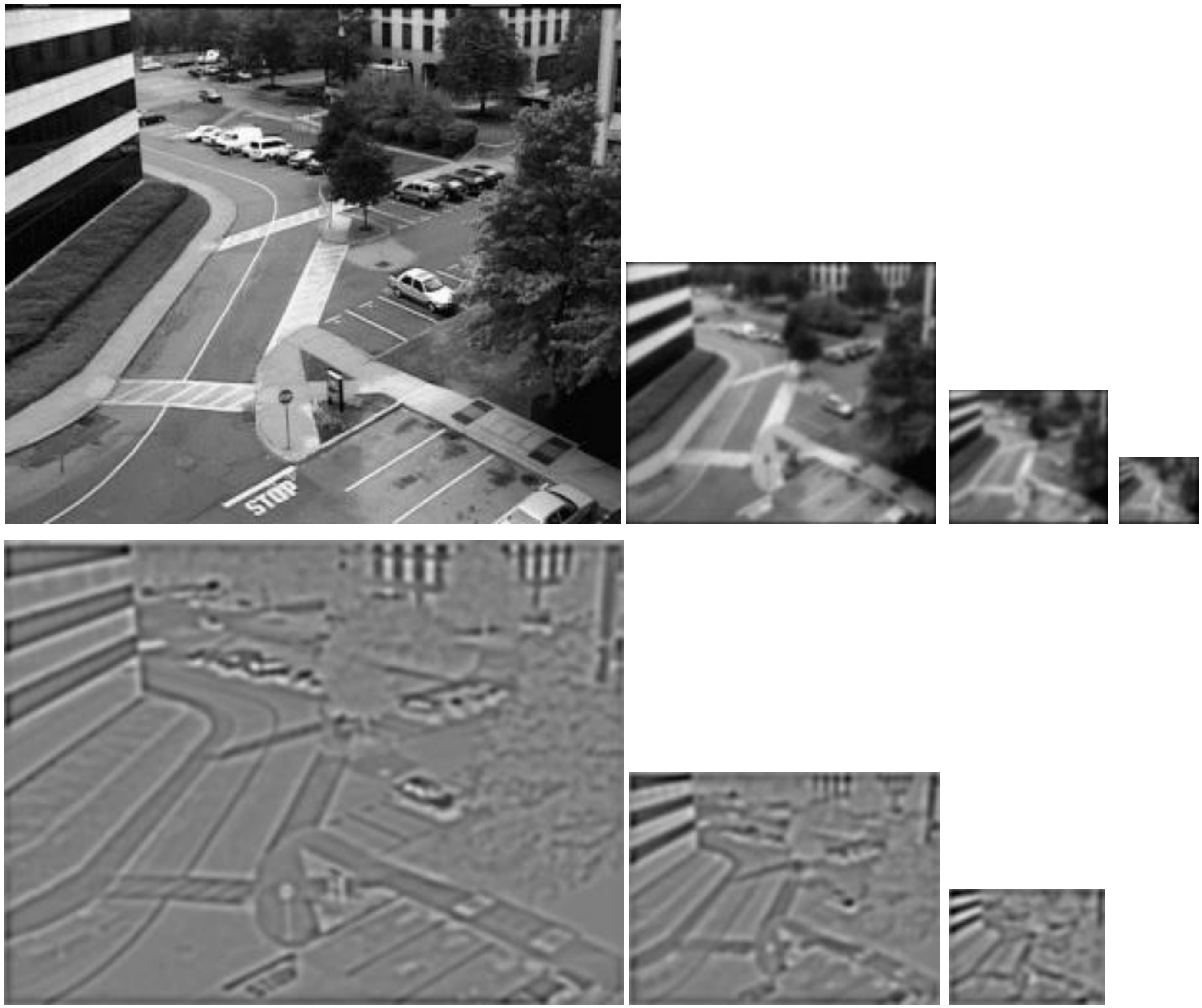
Laplacian Pyramid Construction

- Given input I
- Construct Gaussian pyramid $I^G{}_1,..,I^G{}_n$
- Take the difference between consecutive levels:
- $I^L{}_k = I^G{}_k - I^G{}_{k-1}$
- Image $I^L{}_k$ is an approximation of the Laplacian at scale number k
  - Laplacian is a band-pass filter: Both high frequencies (edges and noise) and low frequencies (slow variations of intensity across the image)

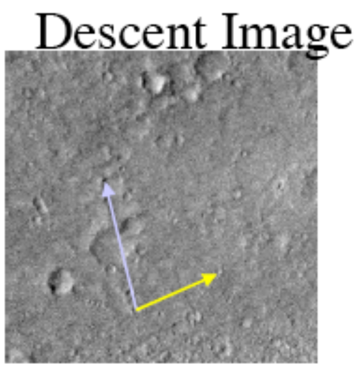

512
256
128
64
32
16
8

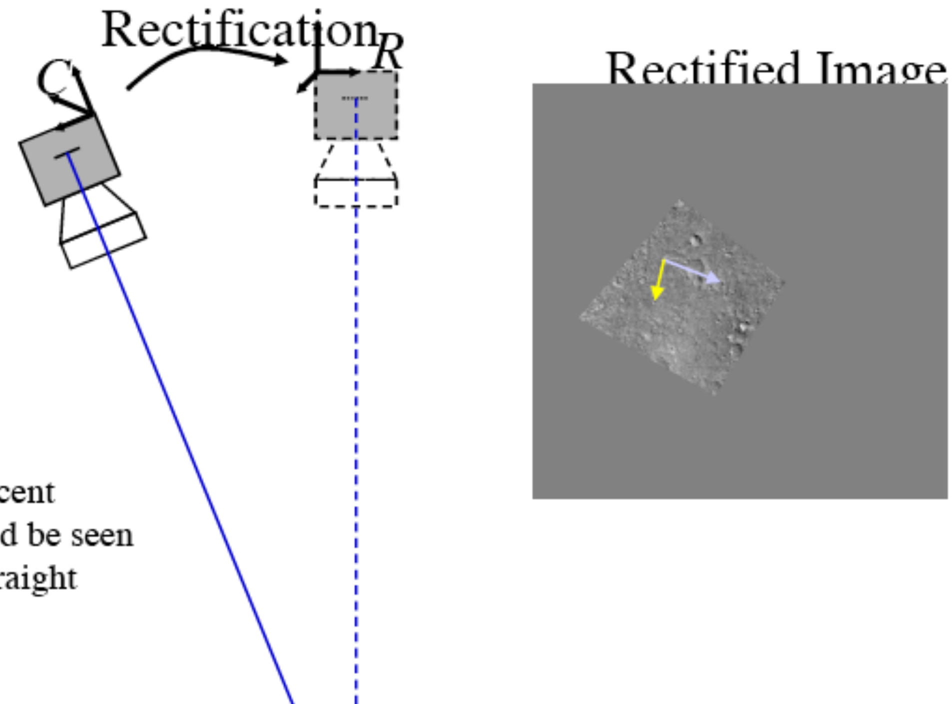


Another Example

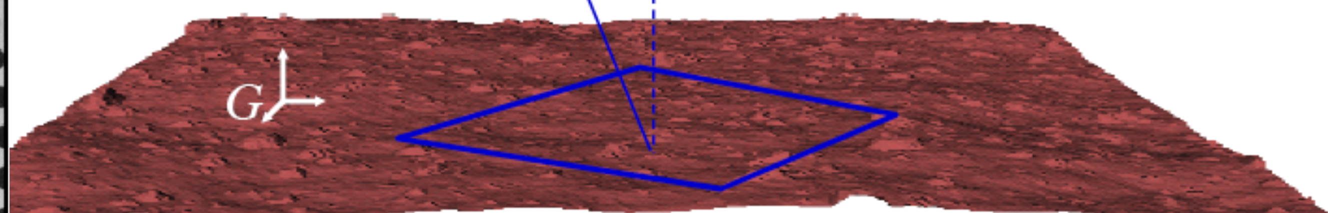


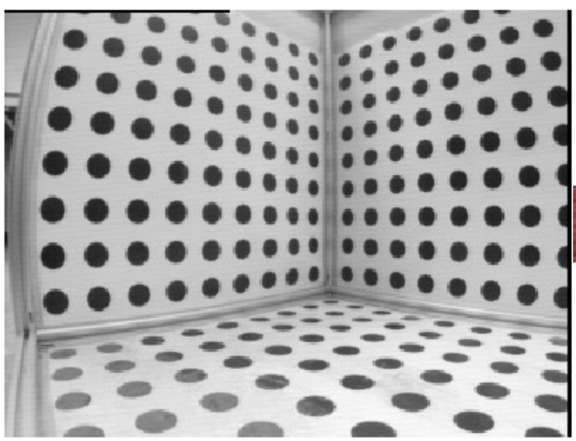
Geometric Image Transformations
(e.g. Barrel Distortion Correction, Rectification)





Rectification transforms a descent image into an image that would be seen by a virtual camera looking straight down.





Geometric Image Transformations (e.g. Barrel Distortion Correction, Rectification)

procedure *inverseWarp*$(f, h, \textbf{out } g)$:

For every pixel x' in $g(x')$

1. Compute the source location $x = \hat{h}(x')$
2. Resample $f(x)$ at location x and copy to $g(x')$

Algorithm 3.2 Inverse warping algorithm for creating an image $g(x')$ from an image $f(x)$ using the parametric transform $x' = h(x)$.

Figure 3.47 Inverse warping algorithm: (a) a pixel $g(x')$ is sampled from its corresponding location $x = \hat{h}(x')$ in image $f(x)$; (b) detail of the source and destination pixel locations.