

Computer Vision

Linear Filtering and Edge Detection

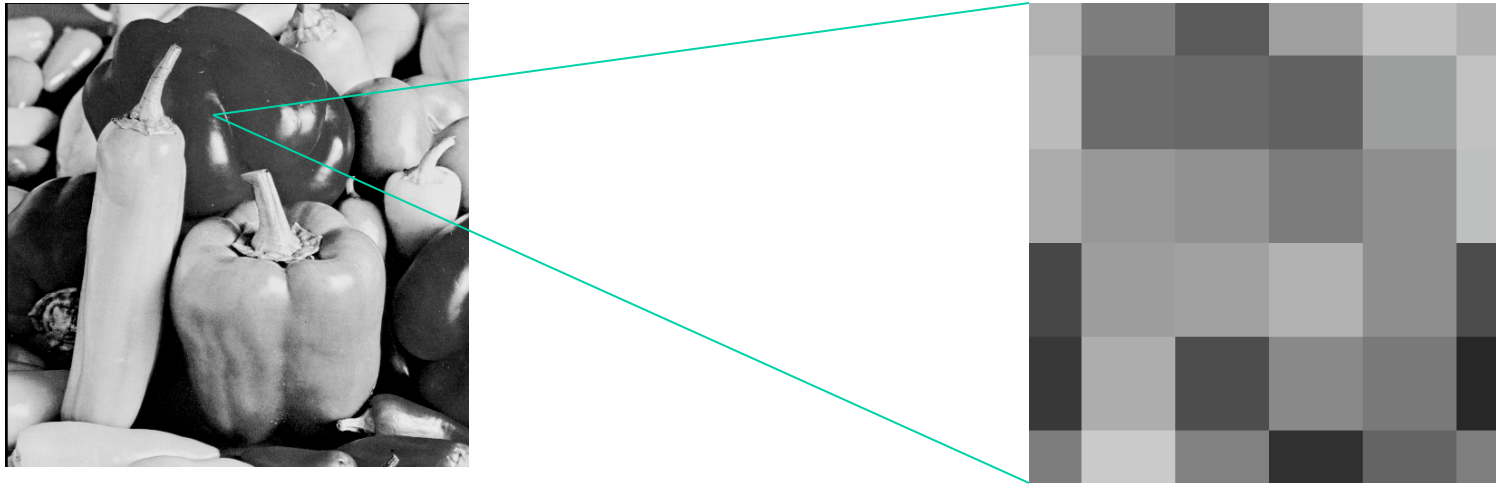
Professor Hager
<http://www.cs.jhu.edu/~hager>

Outline

- Image noise models
- Filtering by Convolution
- Properties of Convolution
- Derivative Operators

Goal: To understand the properties of common linear and nonlinear filtering operations on gray-scale images as a basis for many solutions in computer vision.

IMAGE NOISE



Cameras are not perfect sensors *and*
Scenes never quite match our expectations

Noise Models

- Noise is commonly modeled using the notion of “additive white noise.”
 - Images: $I(u,v,t) = I^*(u,v,t) + n(u,v,t)$
 - Note that $n(u,v,t)$ is independent of $n(u',v',t')$ unless $u'=u, v'=v, t'=t$.
 - Typically we assume that n (noise) is independent of image location as well --- that is, it is i.i.d
 - Typically we assume the n is zero mean, that is $E[n(u,v,t)]=0$
- A typical noise model is the Gaussian (or normal) distribution parametrized by μ and σ
$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right),$$
- This implies that no two images of the same scene are ever identical

Gaussian
Noise:
 $\sigma=1$



9/28/09

Gaussian
Noise:
 $\sigma=16$



9/28/09

Properties of Noise Processes

- Properties of temporal image noise:

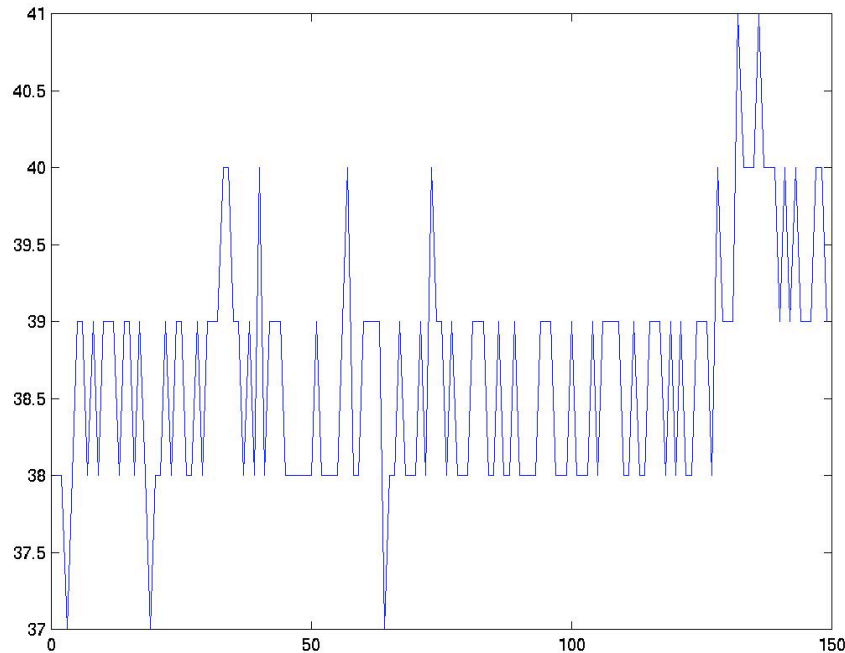
$$\text{Mean } \mu(i,j) = \sum I(u,v,t)/n$$

$$\text{Standard Deviation } \sigma_{i,j} = \text{Sqrt}(\sum (\mu(u,v) - I(u,v,t))^2/n)$$

$$\text{Signal-to-noise Ratio } \frac{\mu(i,j)}{\sigma_{i,j}}$$

Image Noise

- An experiment: take several images of a static scene and look at the pixel values



mean = 38.6
std = 2.99

$\text{Snr} = 38.6 / 2.99 \approx 13$
 $\text{max snr} = 255 / 3 \approx 85$

PROPERTIES OF TEMPORAL IMAGE NOISE (i.e., successive images)

If standard deviation of grey values at a pixel is σ for a pixel for a single image, then the laws of statistics states that for independent sampling of grey values, for a temporal average of n images, the standard deviation is:

$$\frac{\sigma}{\text{Sqrt}(n)}$$

For example, if we want to double the signal to noise ratio, we could average 4 images.

Temporal vs. Spatial Noise


- It is common to assume that:
 - spatial noise in an image is consistent with the temporal image noise
 - the spatial noise is independent and identically distributed
- Thus, we can think of a neighborhood of the image itself as approximated by an additive noise process
- Averaging is a common way to reduce noise
 - instead of temporal averaging, how about spatial?
- For example, for a pixel in image I at i, j

$$I'(i, j) = 1/9 \sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} I(i', j')$$

Convolution

Convolution is the generalization of this “averaging” process. As we’ll see, it can do more than average.

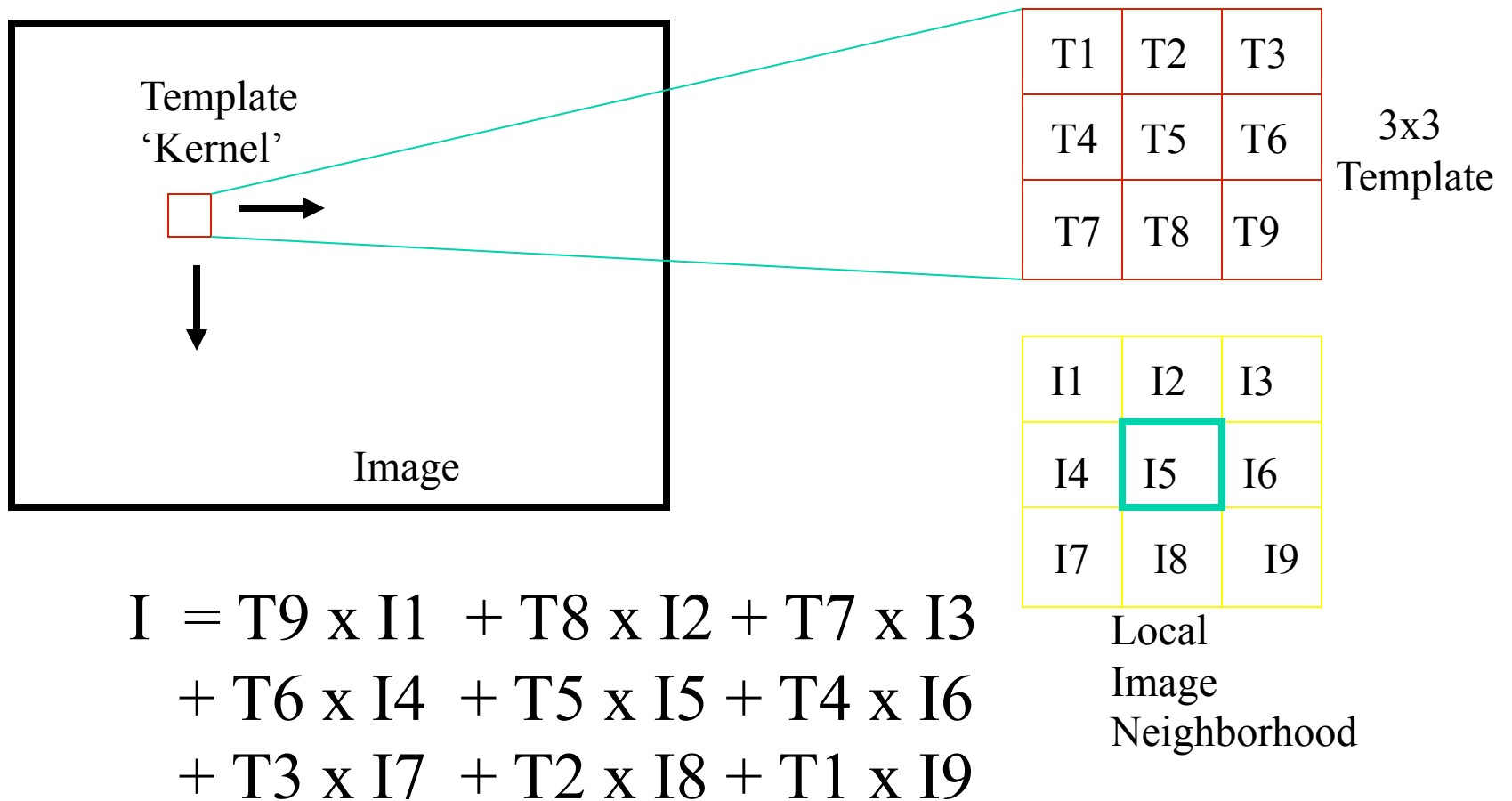
$$(I * K)(x,y) = \sum_i \sum_j I(x-i,y-j) K(i,j)$$



Note these indices
run backwards --- this
can sometimes fool you
down the road!

We often write $I' = I * K$ to represent the convolution of I by K . K is referred to as the *kernel* of the convolution (or sometimes the “stencil” in the discrete case). It is also the impulse response of the filter.

DISCRETE CONVOLUTION



Convolution

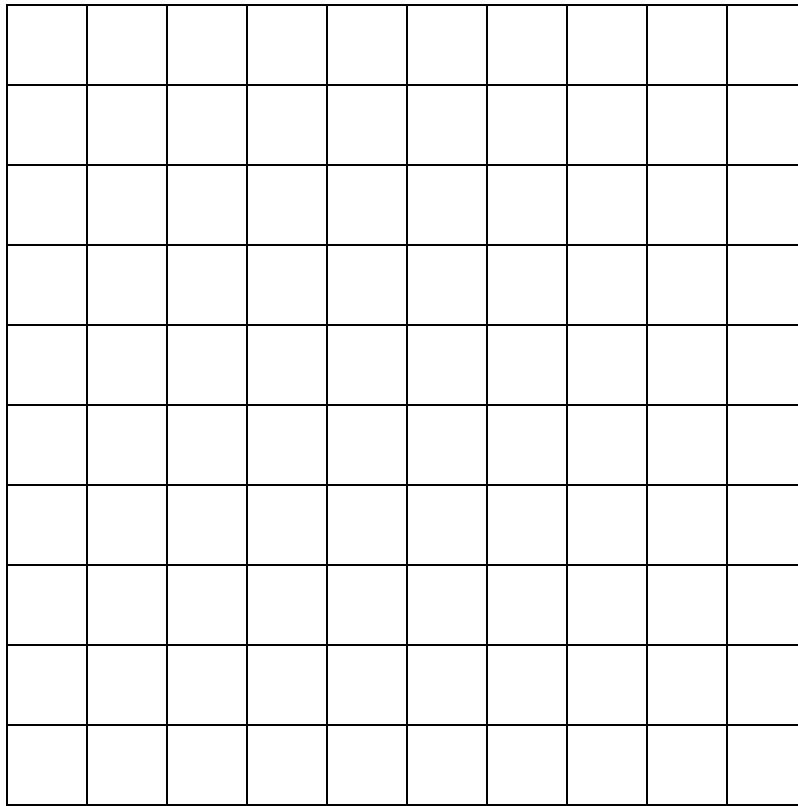


Image (I)

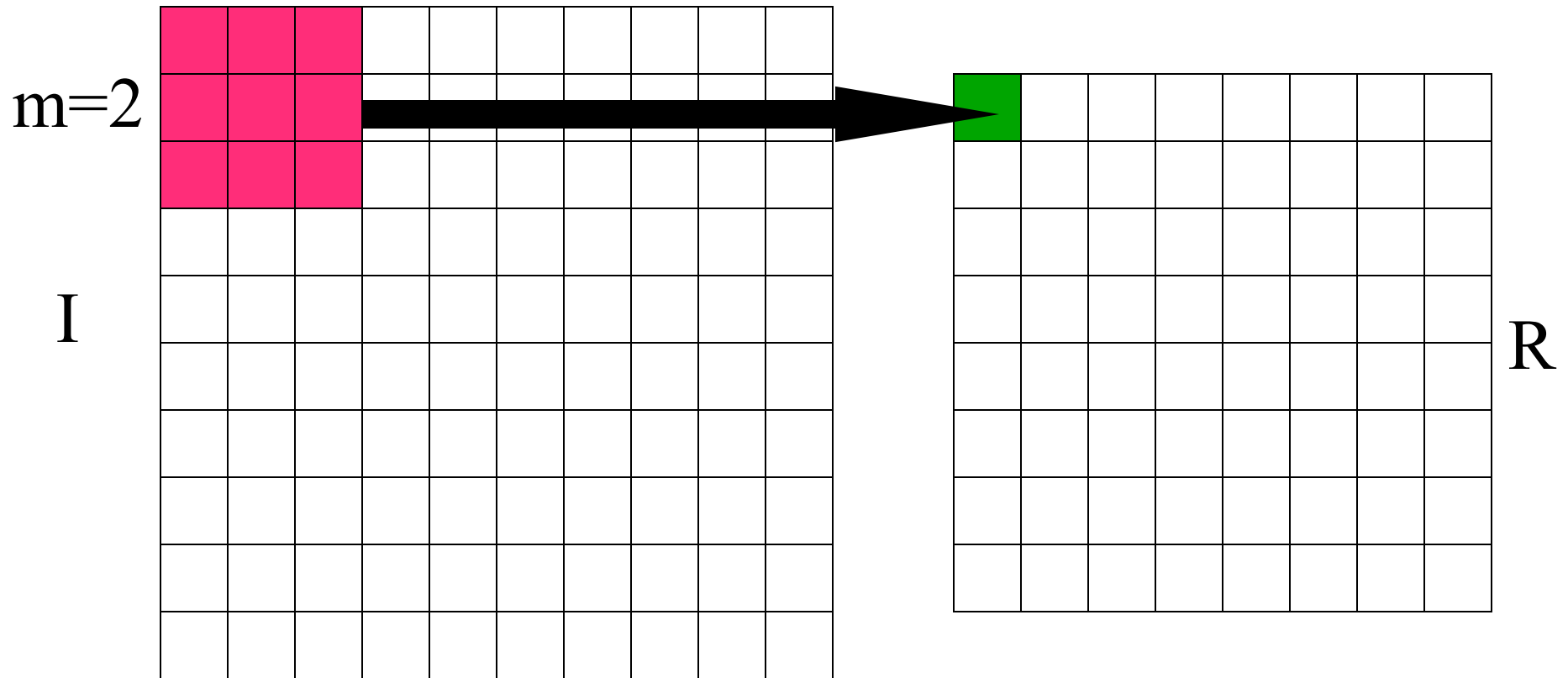
*

1	2	1
2	4	2
1	2	1

Kernel (K)

Note: Typically Kernel is relatively small in vision applications.

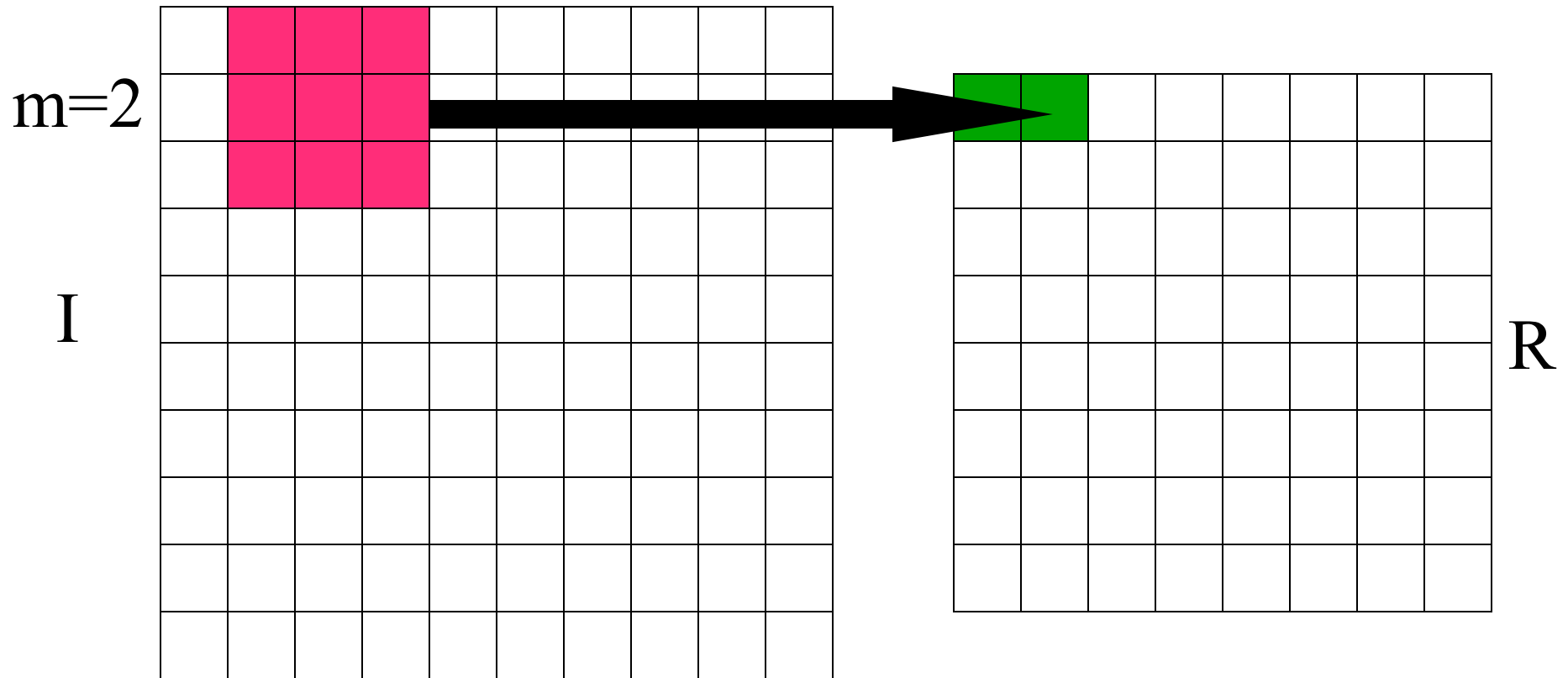
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

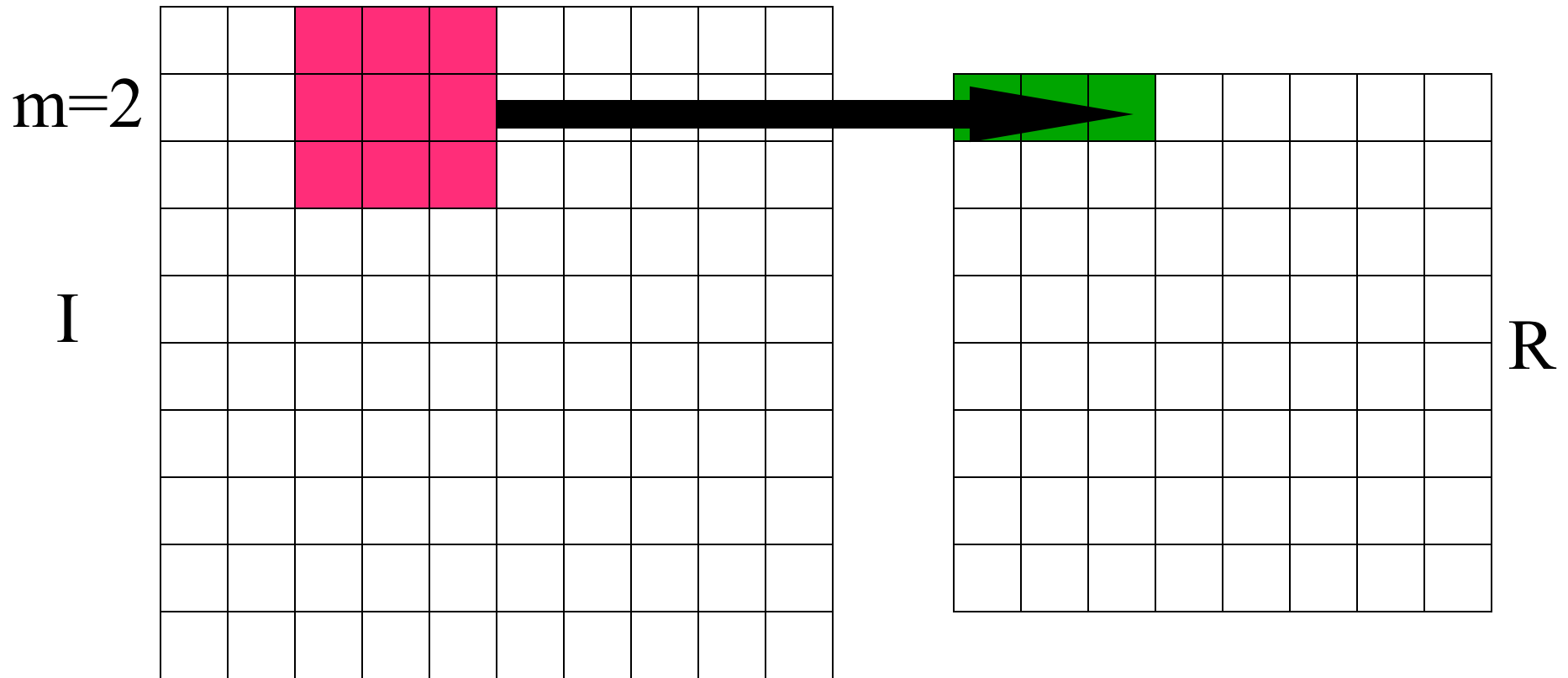
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

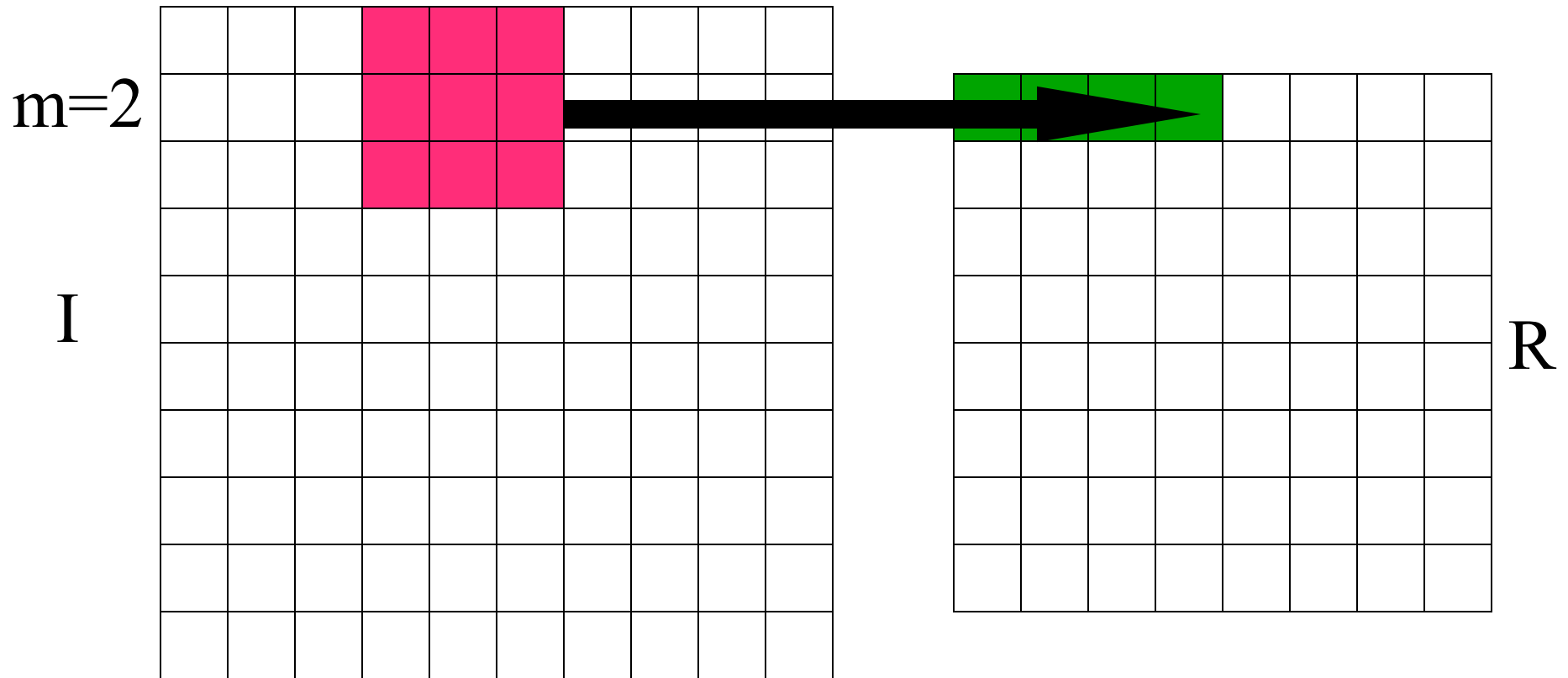
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

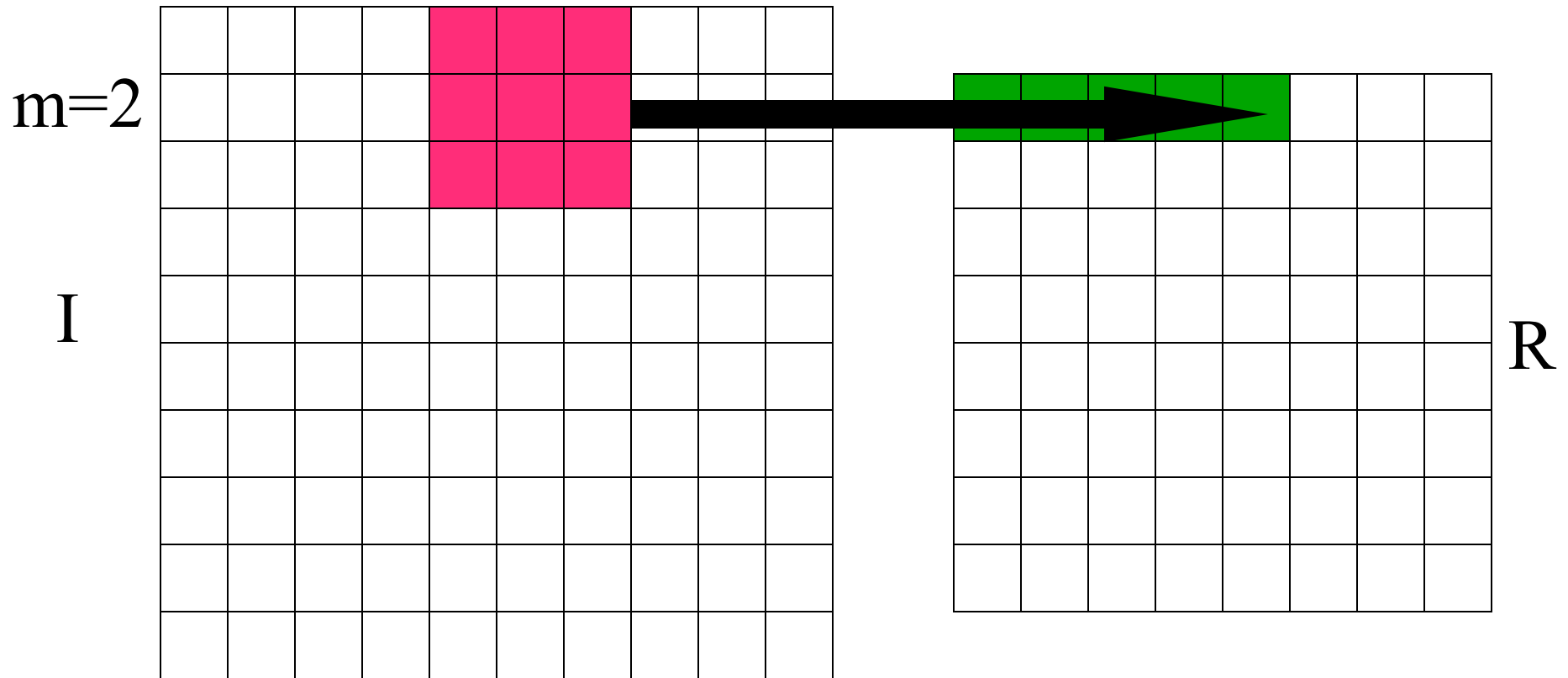
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

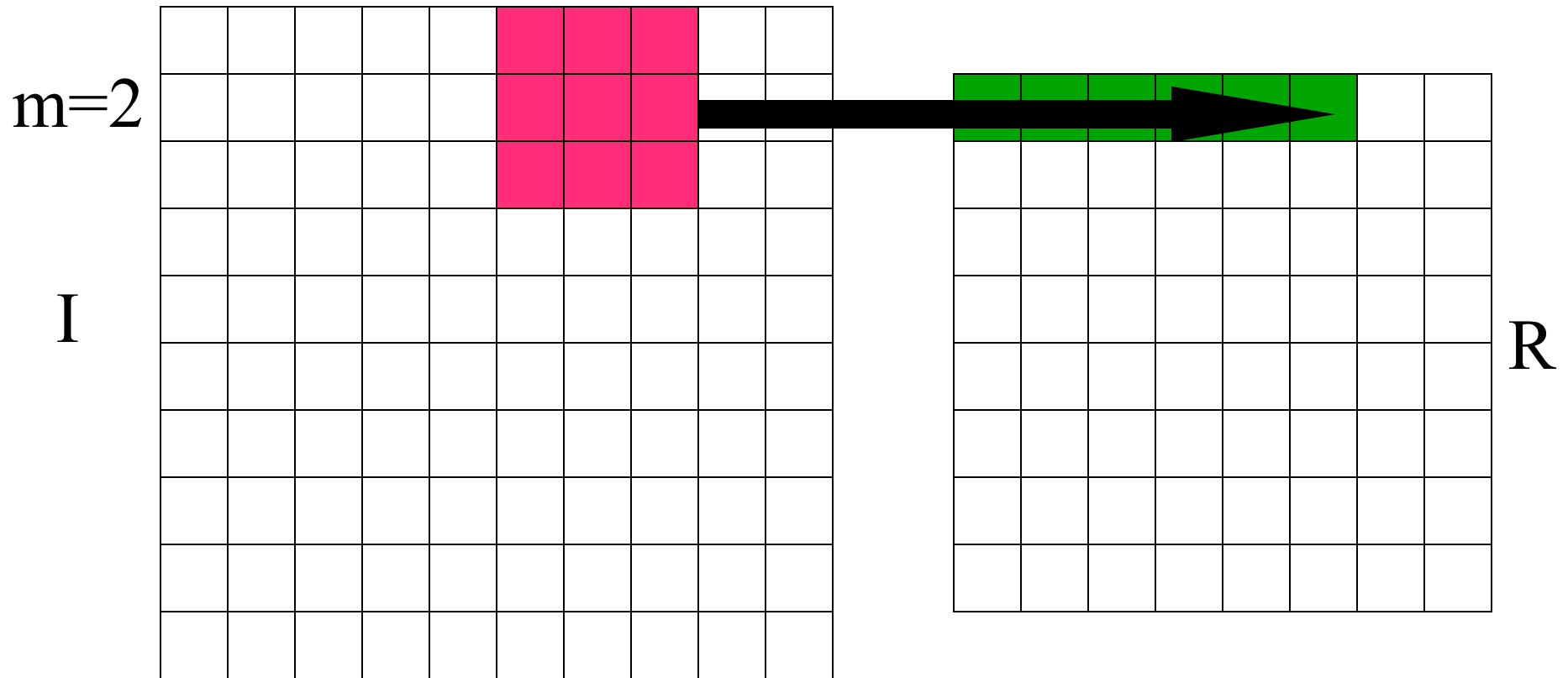
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

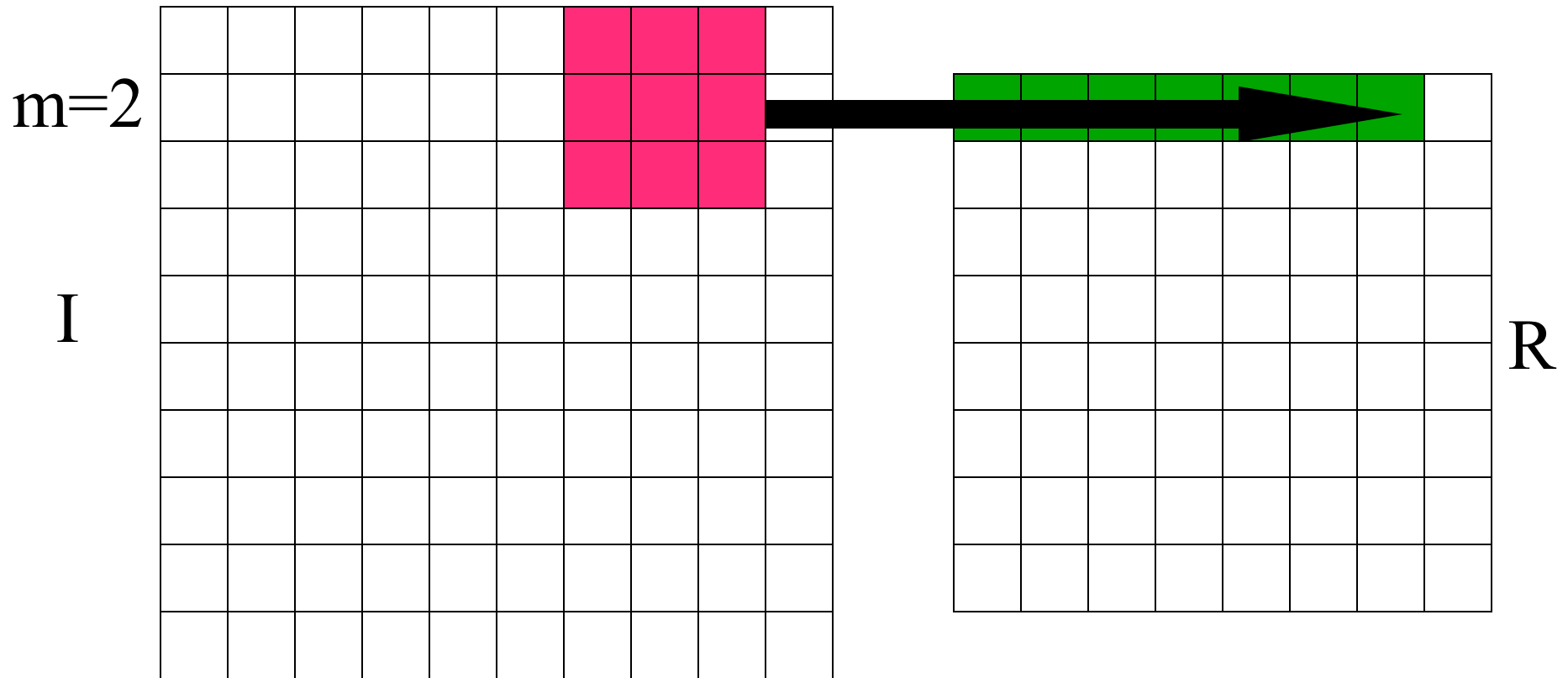
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

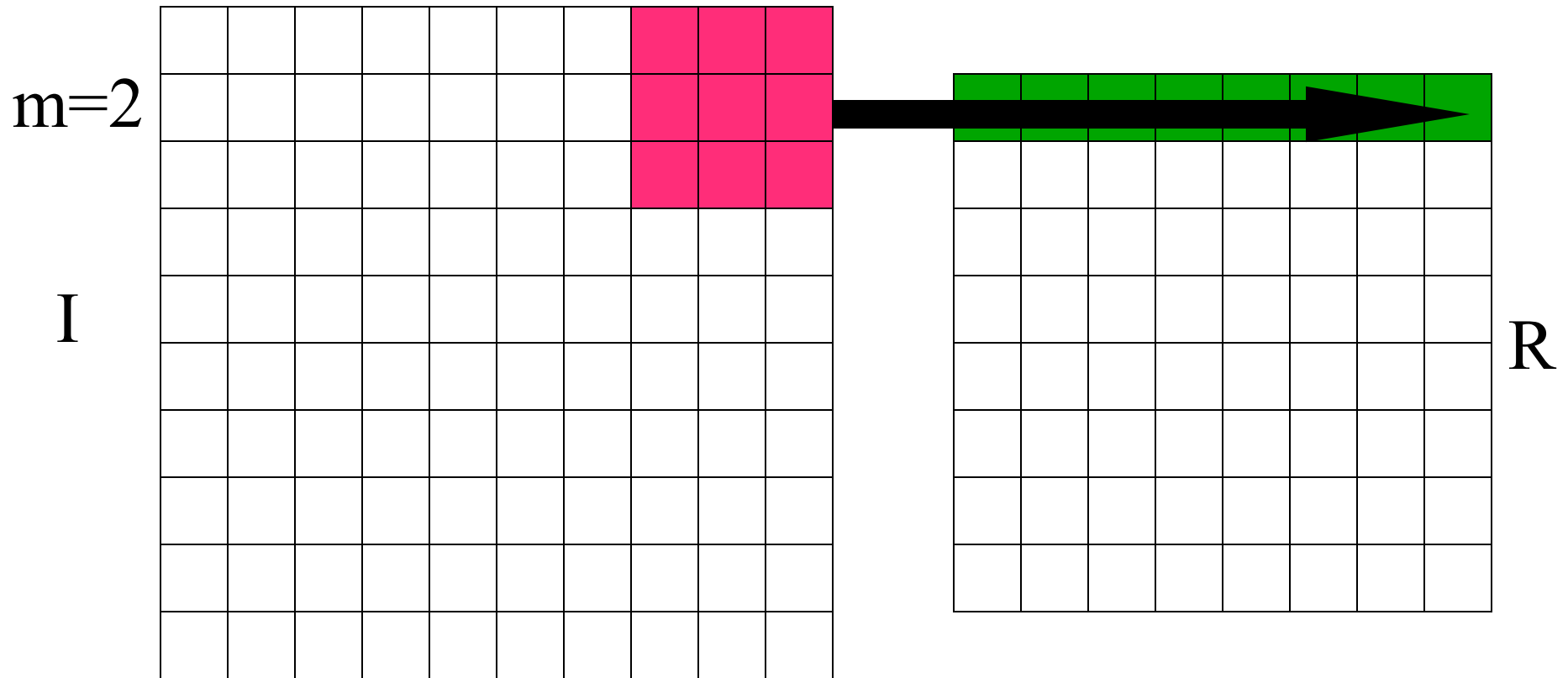
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

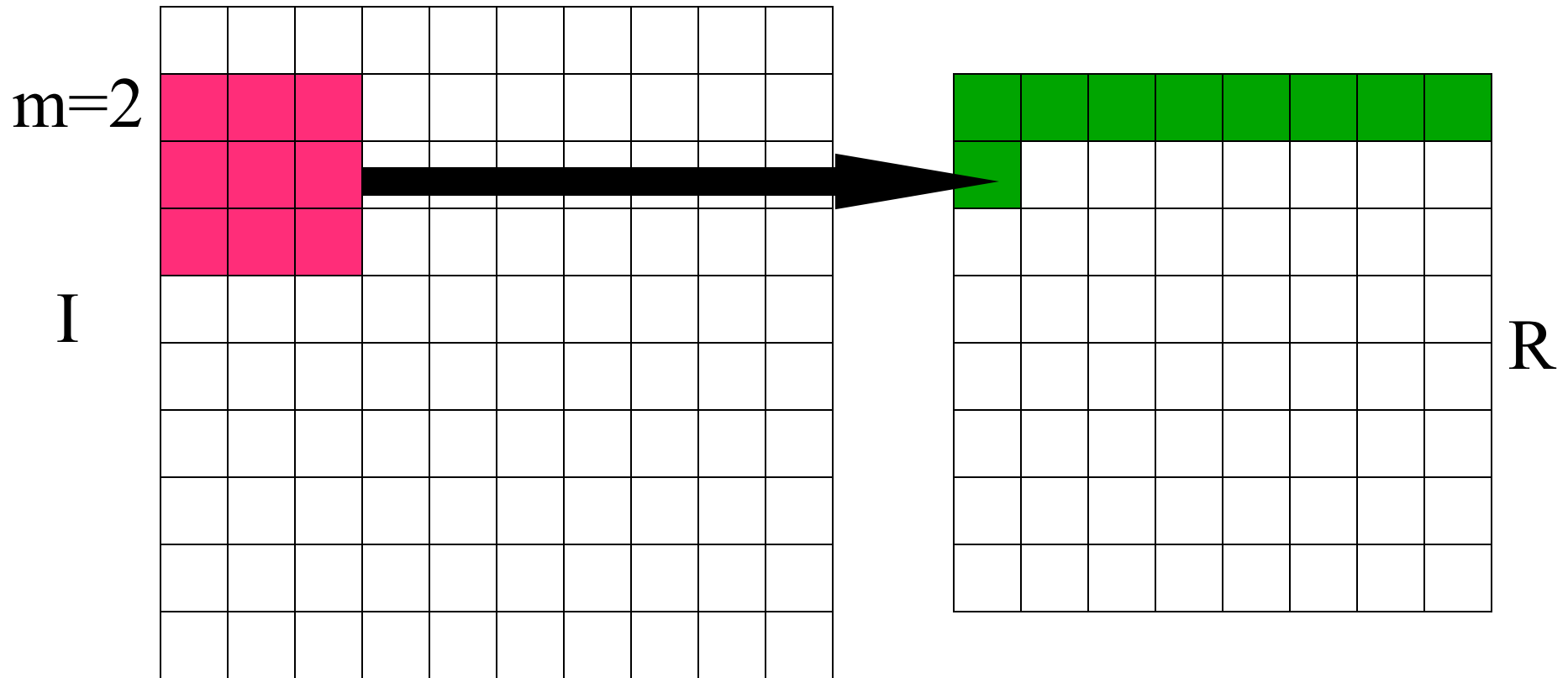
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

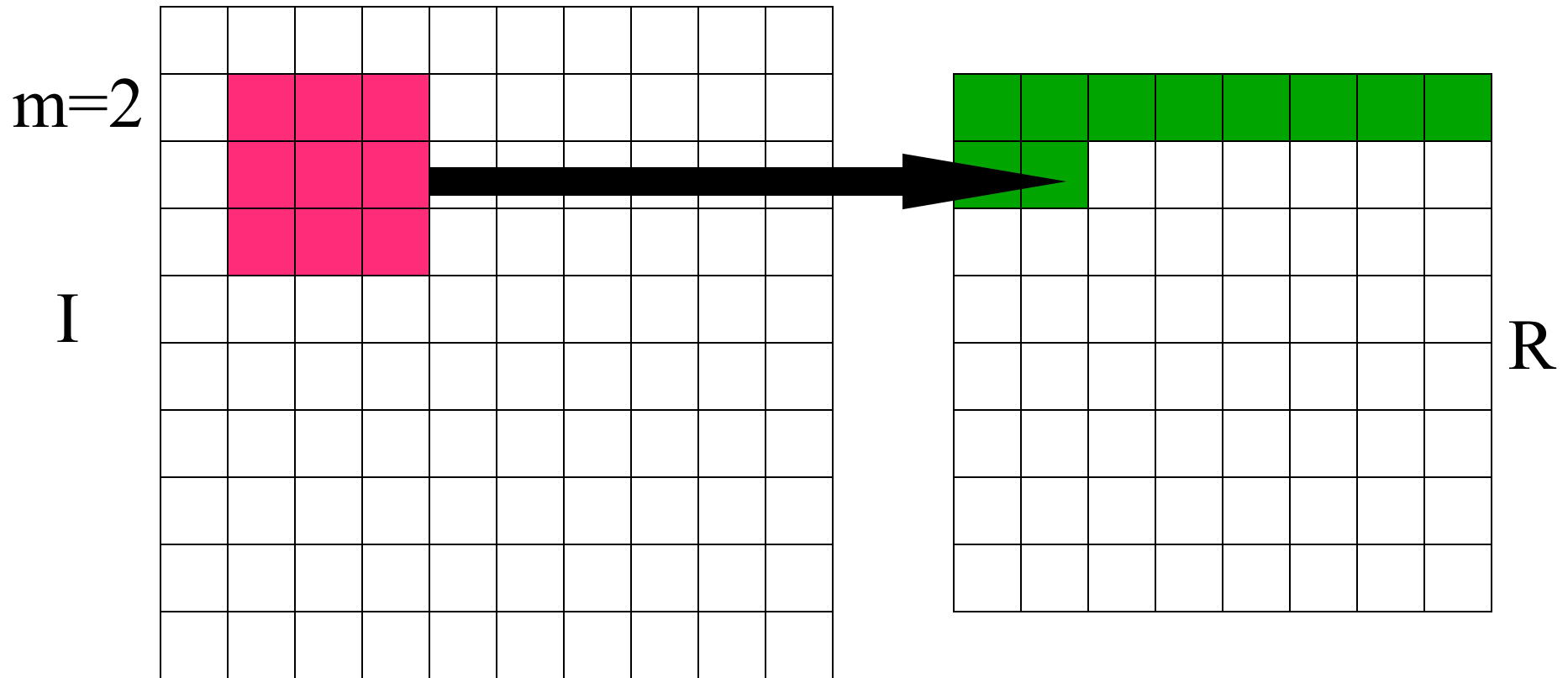
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

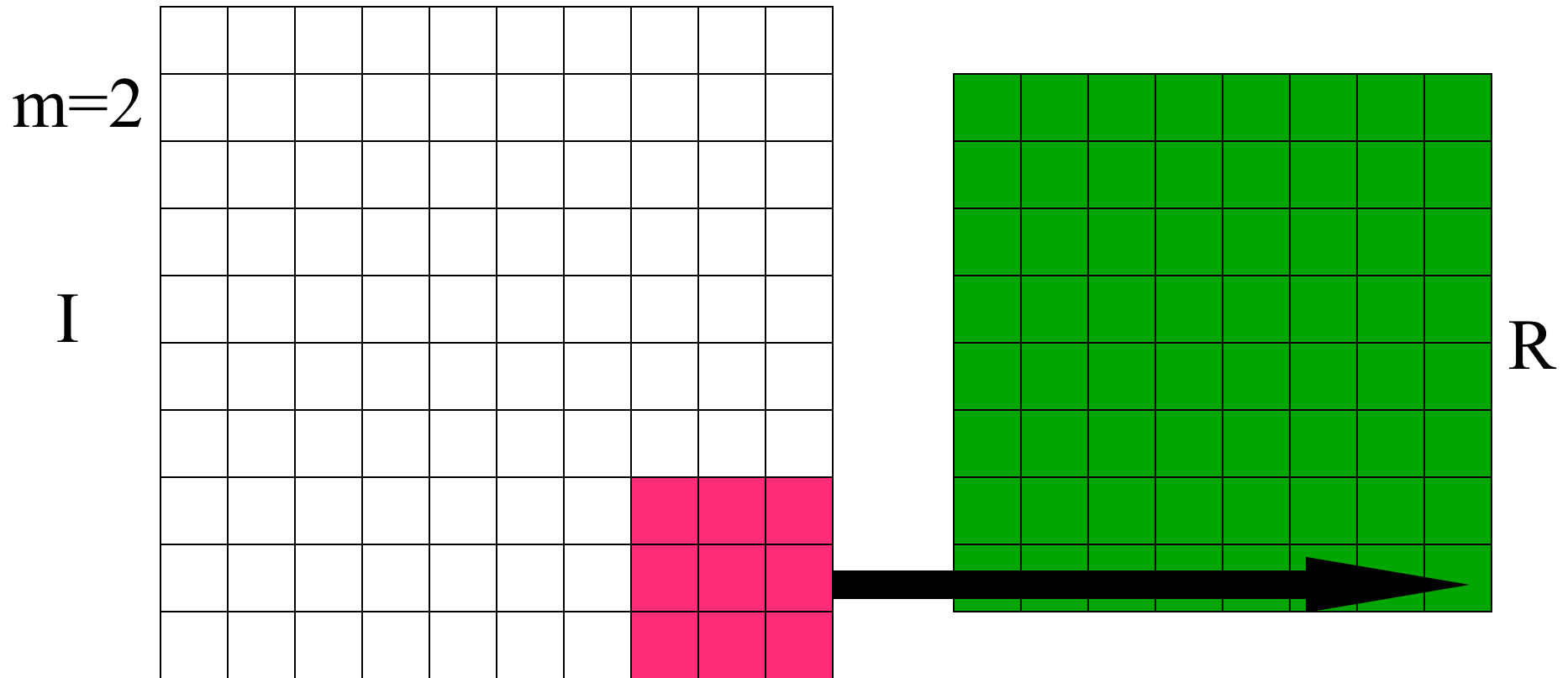
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

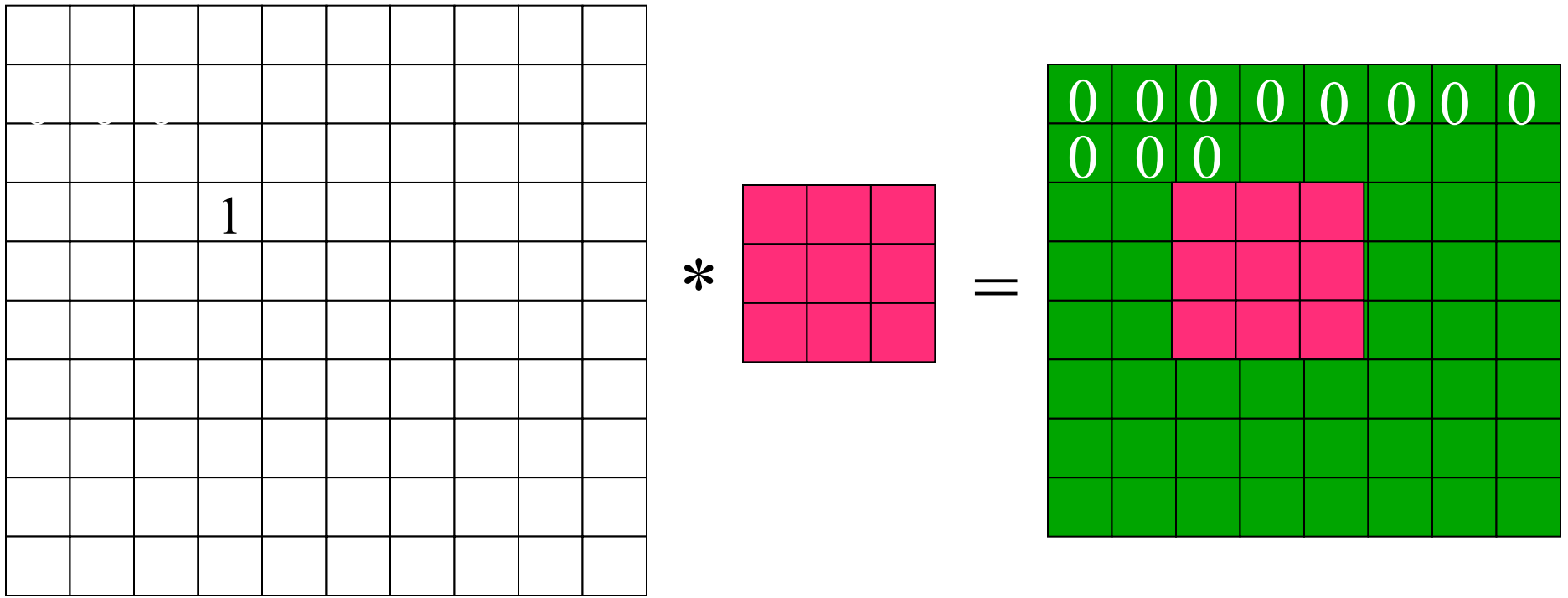
Convolution: $R = K * I$



Kernel size
is $m+1$ by $m+1$

$$R(i, j) = \sum_{h=-m/2}^{m/2} \sum_{k=-m/2}^{m/2} K(h, k) I(i - h, j - k)$$

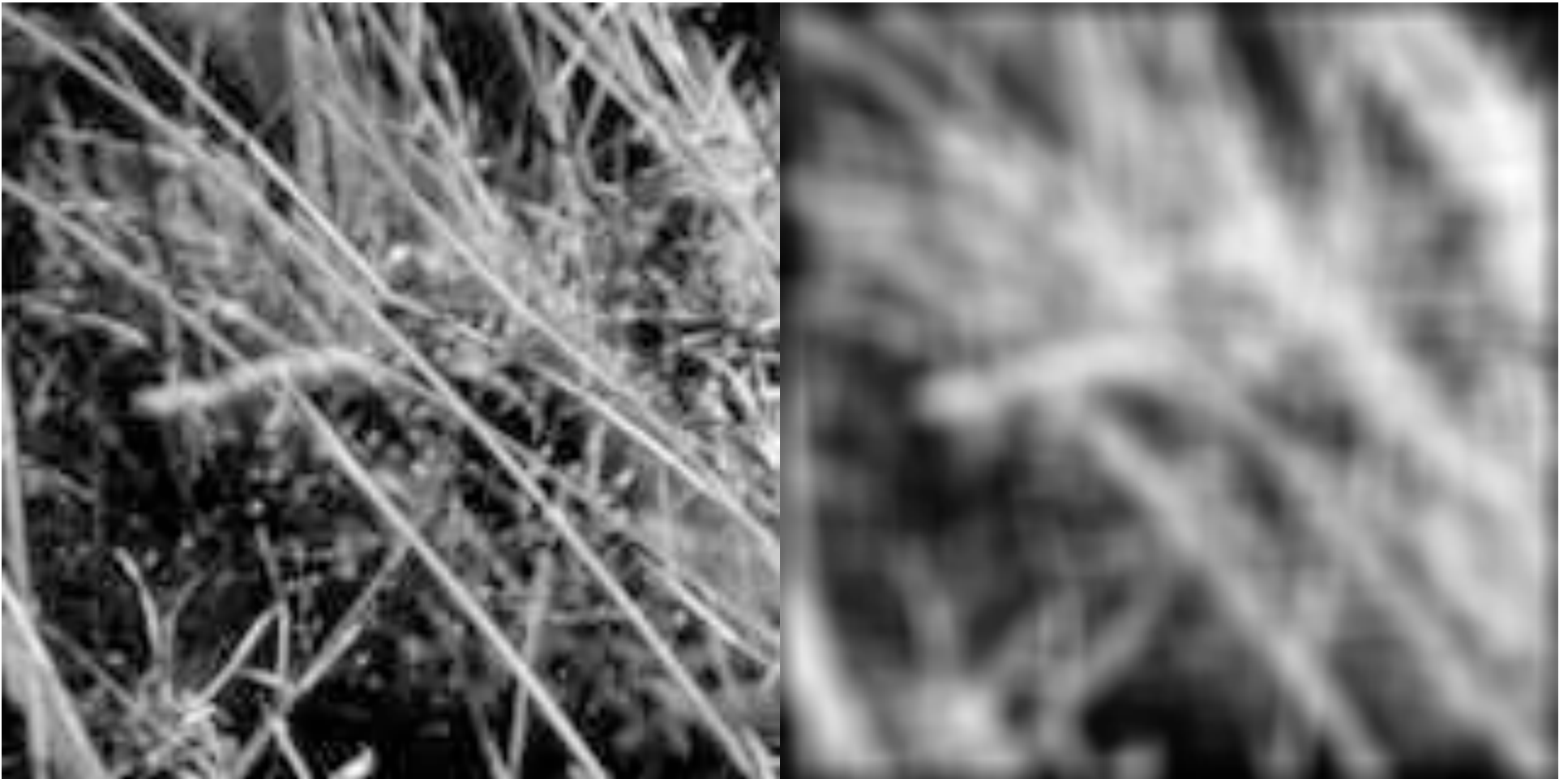
Impulse Response



Note that the “self-replicating” property of the impulse response is what require the “flip” in the template when it is applied

Smoothing by Averaging

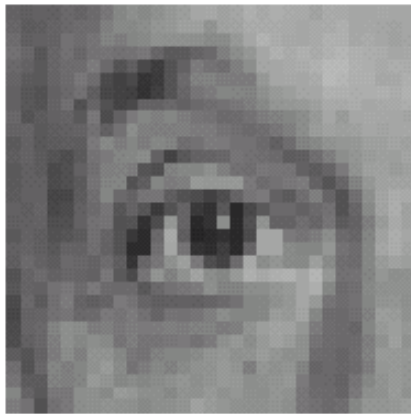
Kernel: 



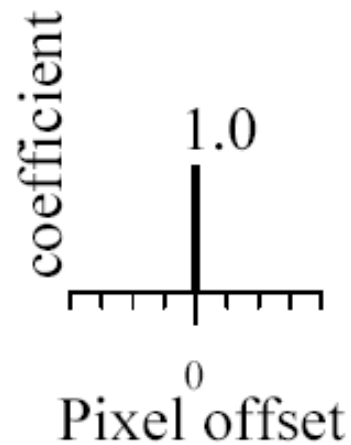
9/28/09

CS 461, Copyright © D. Hager
Borrowed from D. K. Fiegner

Linear filtering (warm-up slide)

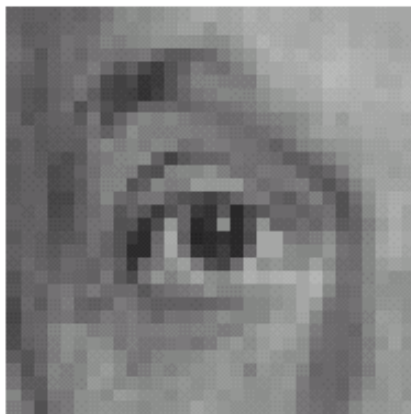


original

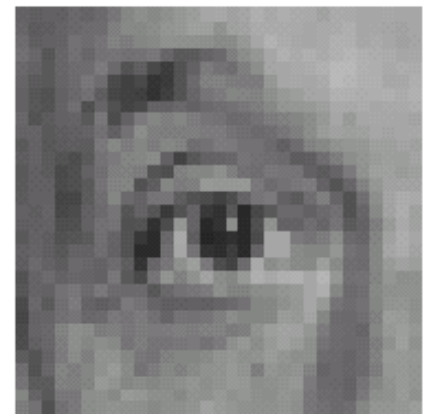
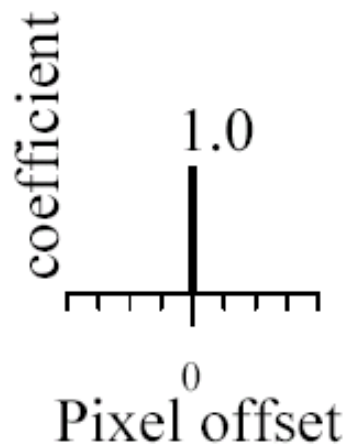


?

Linear filtering (warm-up slide)

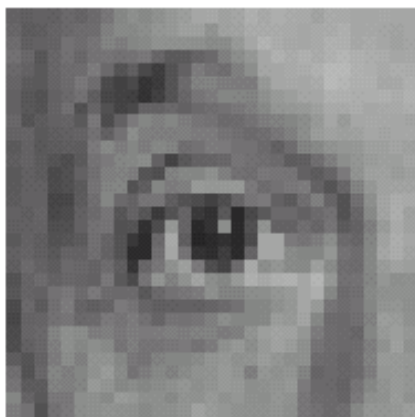


original

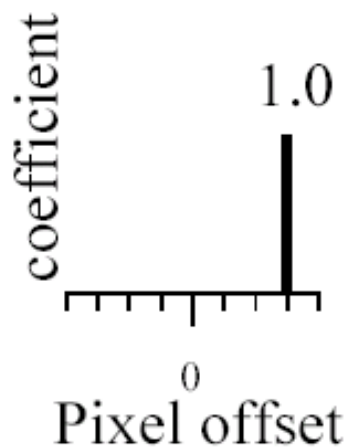


Filtered
(no change)

Linear filtering

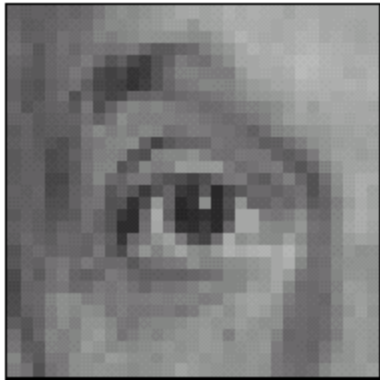


original

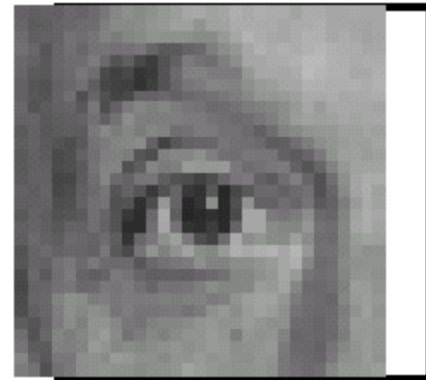
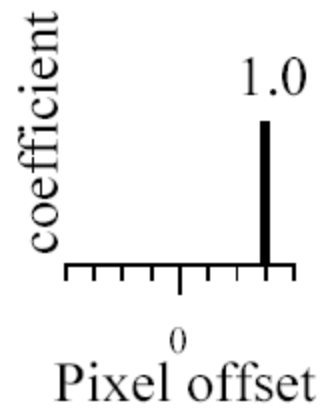


?

shift

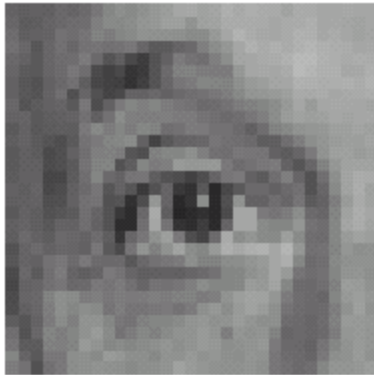


original

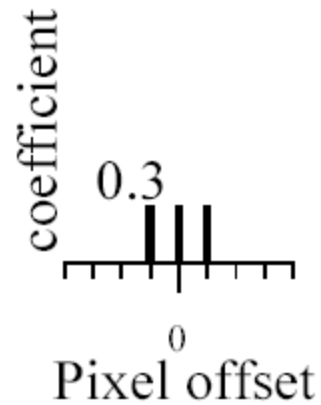


shifted

Linear filtering

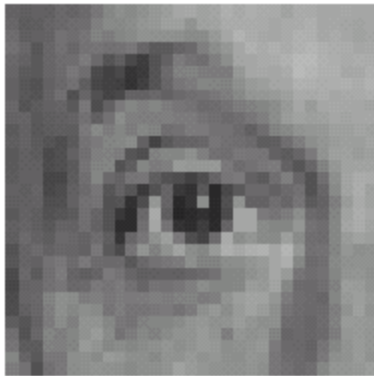


original

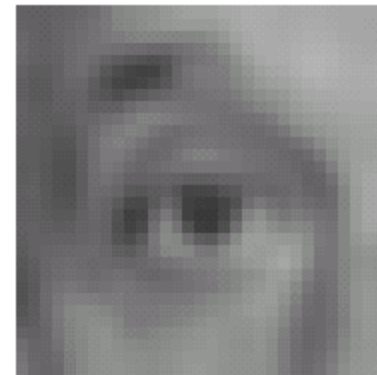
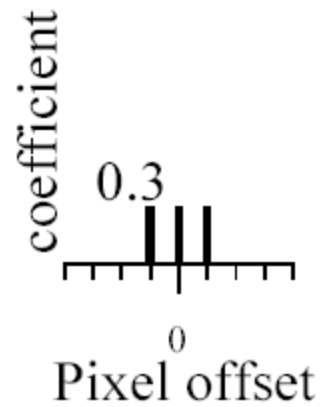


?

Blurring

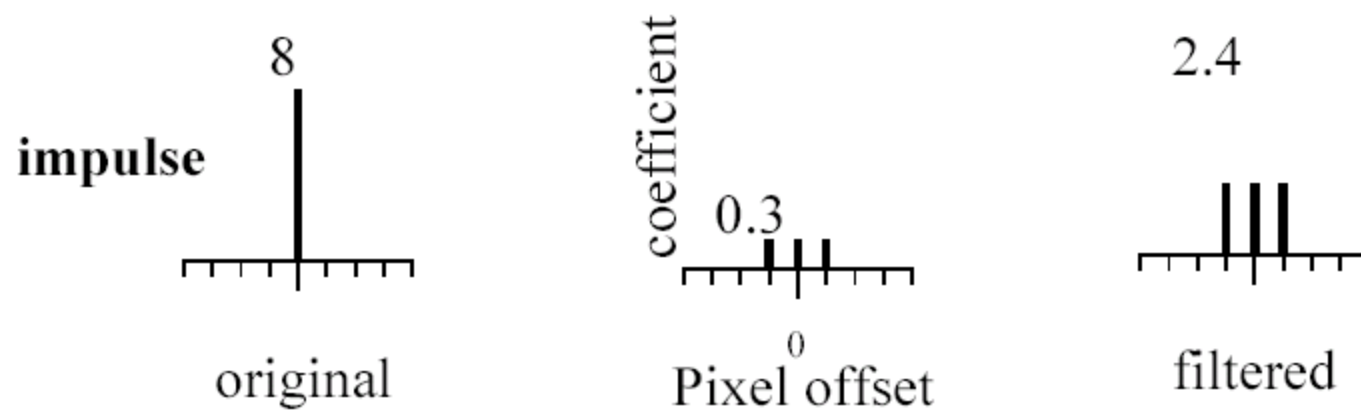


original

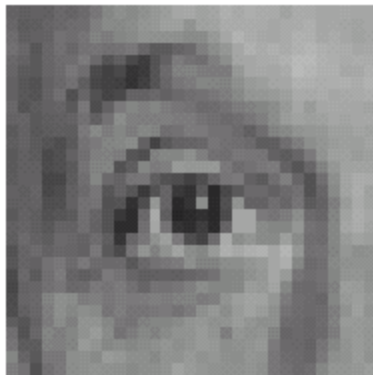


Blurred (filter applied in both dimensions).

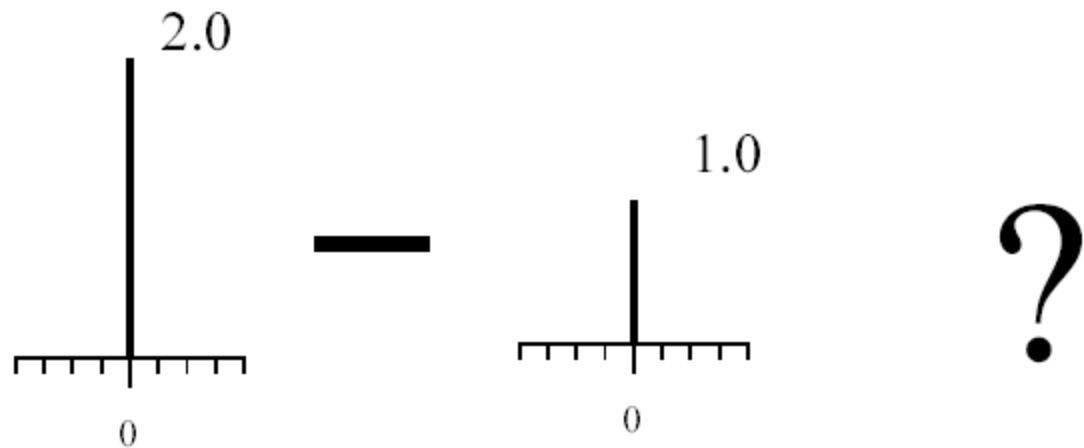
Blur examples



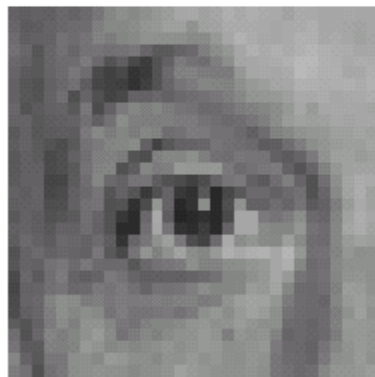
Linear filtering (warm-up slide)



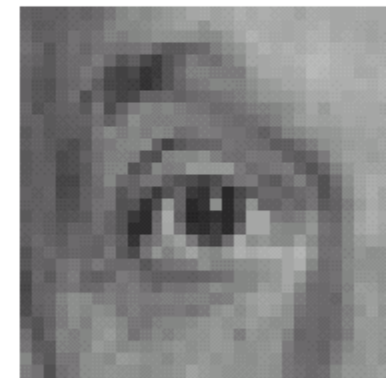
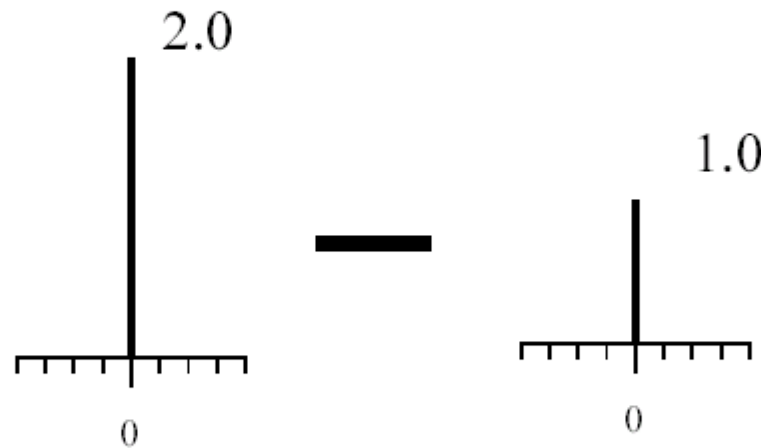
original



Linear filtering (no change)

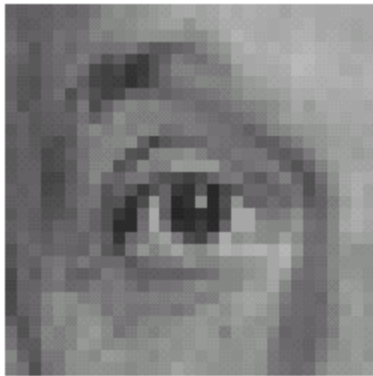


original

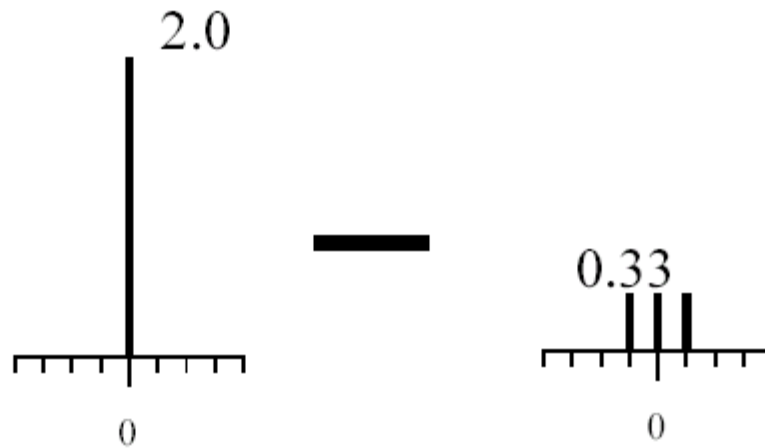


Filtered
(no change)

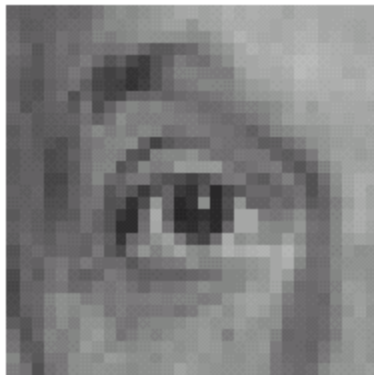
Linear filtering



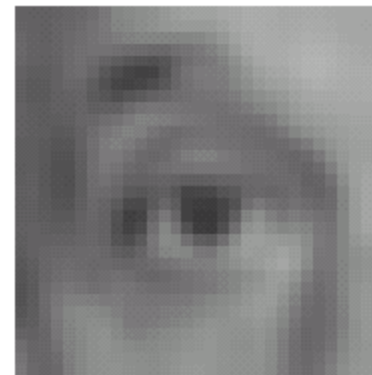
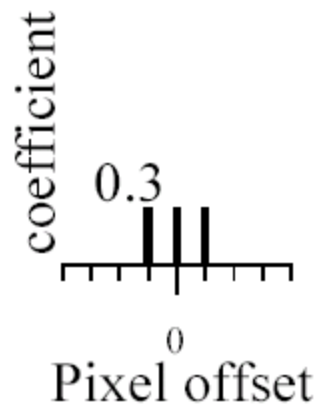
original



(remember blurring)

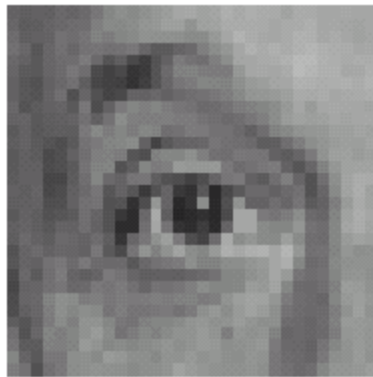


original

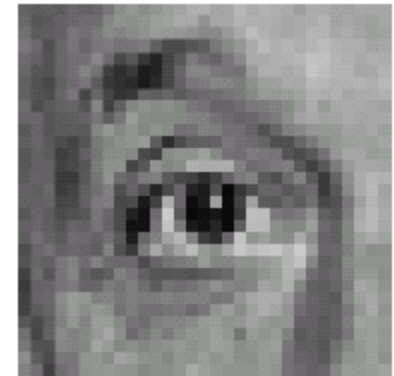
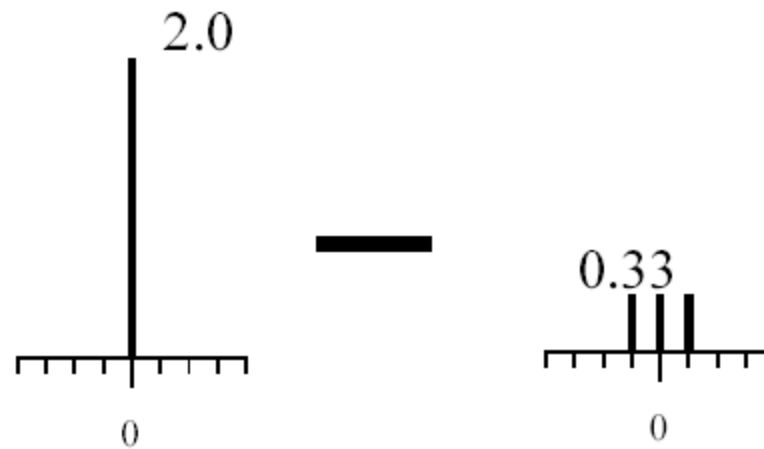


Blurred (filter
applied in both
dimensions).

Sharpening

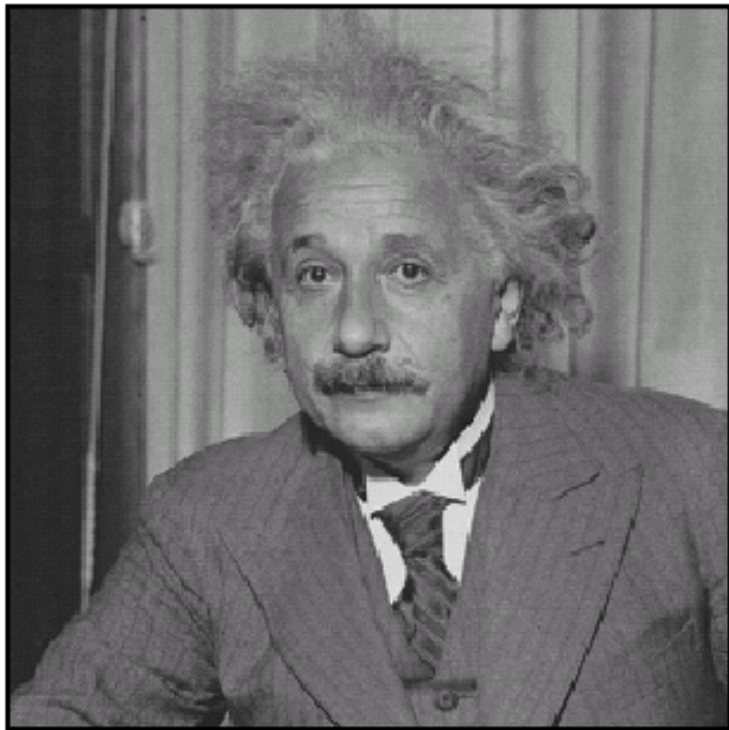


original

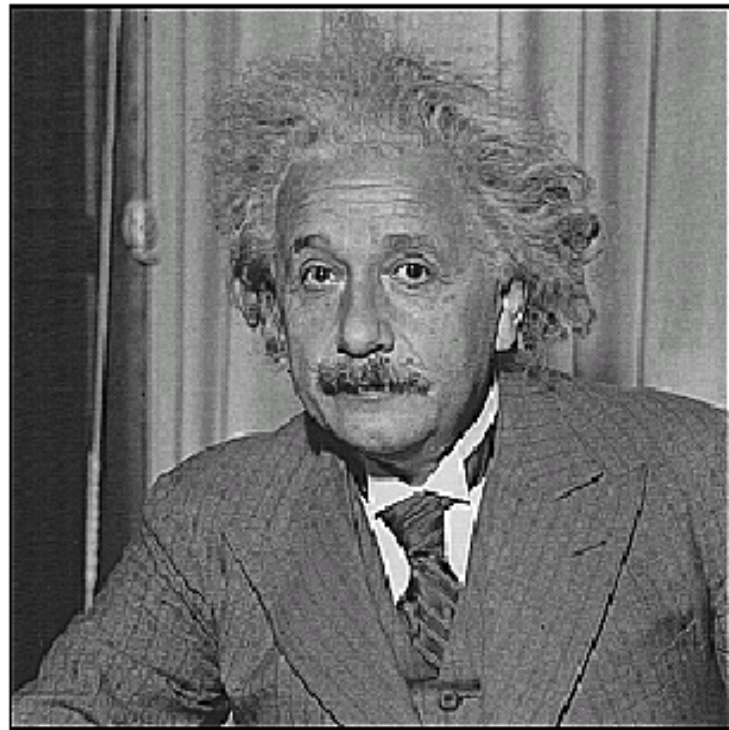


Sharpened
original

Sharpening



before



after

9/28/09

CS 461, Copyright © D. Karger
borrowed from D. Karger

How to Reduce Noise

- For a pixel in image I at I,j

$$I'(i, j) = 1/9 \sum_{i'=i-1}^{i+1} \sum_{j'=j-1}^{j+1} I(i', j')$$

- Computing this for every pixel location is the *convolution* of the image I with the *template* (or *kernel*) consisting of a 3x3 array of 1/9's.
- Note that is this $O(n^2m^2)$ for an nxn image and mxm template.
- Note we have to normalize the template to 1 to make sure we don't introduce any scaling into the image.
- Quiz question: what image structures are exactly preserved by this form of averaging?

Some Convolution Facts

- Convolution is
 - Associative
 - Commutative
 - A linear operator
- With a fixed kernel, convolution is also *shift invariant*, meaning if we translate the signal, we translate (but do not otherwise change) the response.
 - Linearity plus shift invariance can be used to derive convolution!
- We are using a *discrete convolution*; we will see this is not always consistent with an underlying *continuous convolution* that we may wish to implement
- Convolution is formally defined on unbounded images and kernels.
 - padding schemes:
 - pad with zeros (same size vs. full size)
 - compute only legal values

} time for a Matlab demo!

Another View of Convolution

- Suppose we consider the convolution template as a “vector”:
 - $T = [T_1, T_2, T_3 \dots T_n]$
- Likewise, consider a region of the image to which the convolution is applied as a vector
 - $I = [I_1, I_2, \dots I_n]$
- Then the value of the convolution at a point is just the “dot product”
 - $v = \text{sum}(\text{fliplr}(T) .* I)$
- Thus, we can also think of convolution as a kind of “pattern match” where regions of the image that are “similar” to T respond more strongly than those that are dissimilar (up to a scale factor)
- We’ll see this makes sense when thinking of Fourier transforms...

Understanding Convolution

- Another way to think about convolution is in terms of how it changes the *frequency distribution* in the image.
- Recall the *Fourier* representation of a function
 - $F(u) = \int f(x) e^{-2\pi i u x} dx$
 - recall that $e^{-2\pi i u x} = \cos(2\pi u x) - i \sin(2\pi u x)$
 - Also we have $f(x) = \int F(u) e^{2\pi i u x} du$
 - $F(u) = |F(u)| e^{i \Phi(u)}$
 - a decomposition into magnitude ($|F(u)|$) and phase $\Phi(u)$
 - If $F(u) = a + i b$ then
 - $|F(u)| = (a^2 + b^2)^{1/2}$ and $\Phi(u) = \text{atan2}(a,b)$
 - $|F(u)|^2$ is the *power spectrum*
- Questions: what function takes many many many terms in the Fourier expansion?

Understanding Convolution

Discrete Fourier Transform (DFT)

$$F[u, v] \equiv \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I[x, y] e^{-\frac{2\pi j}{N} (xu+yv)}$$

Inverse DFT

$$I[x, y] \equiv \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F[u, v] e^{\frac{+2\pi j}{N} (ux+vy)}$$

Implemented via the “Fast Fourier Transform” algorithm (FFT)

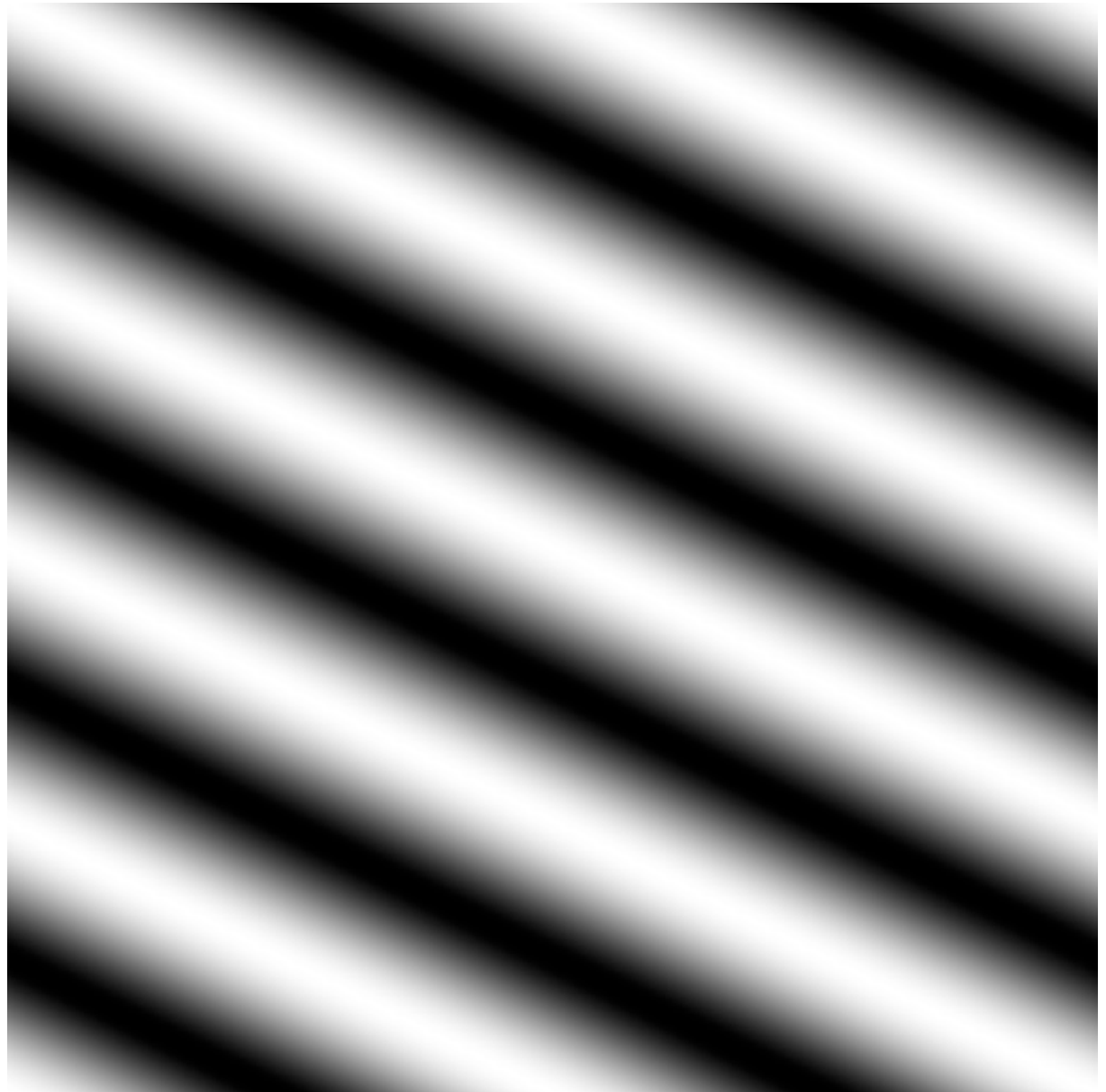
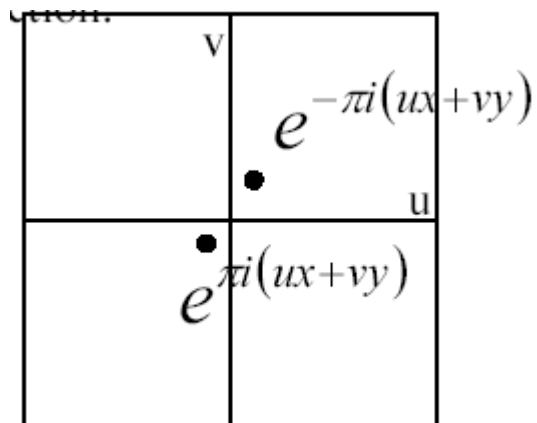
Fourier basis element

$$e^{-i2\pi(ux+vy)}$$

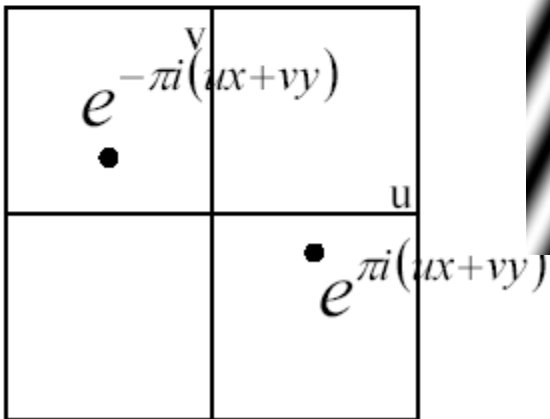
Transform is sum of orthogonal basis functions

Vector (u,v)

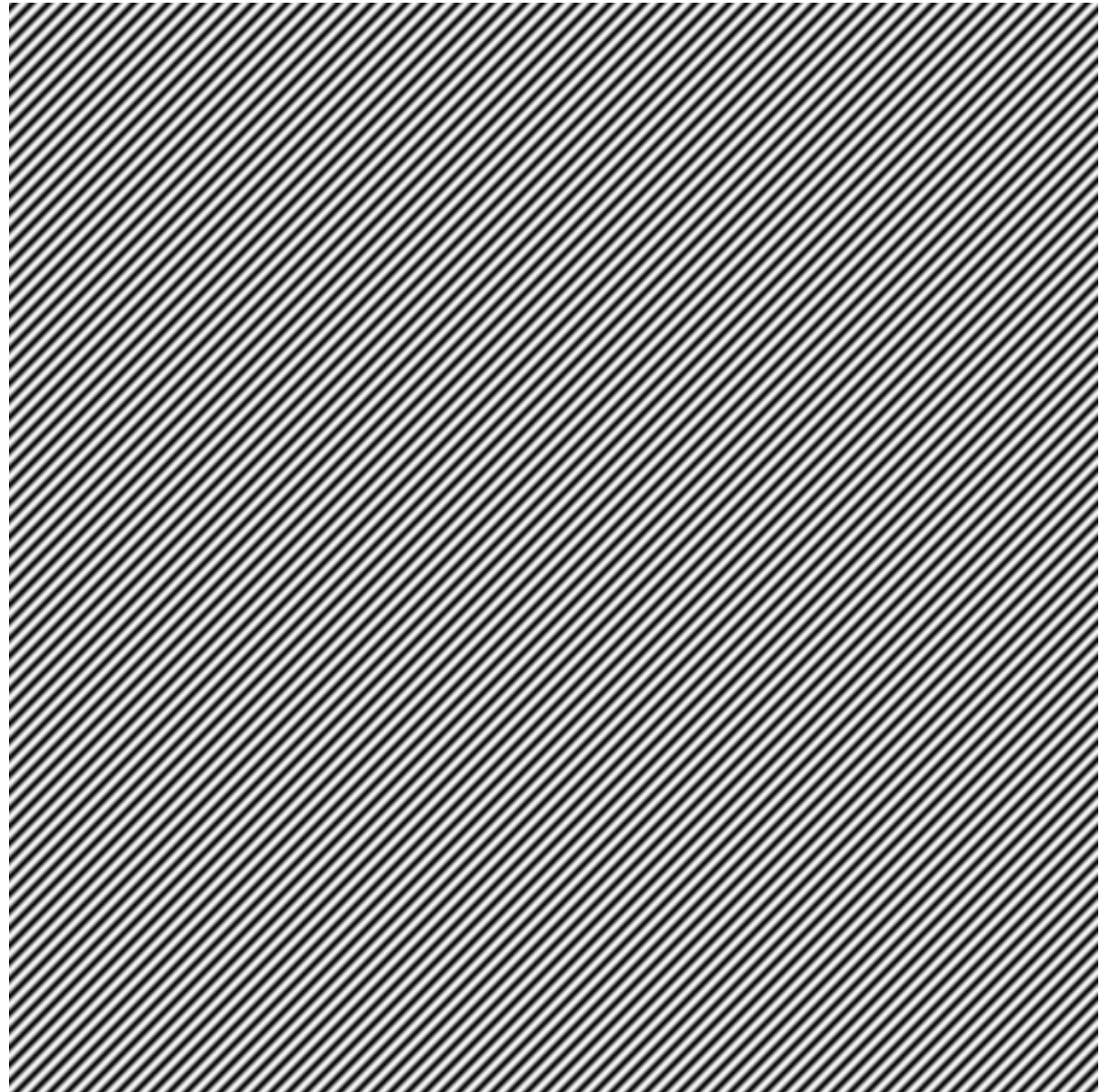
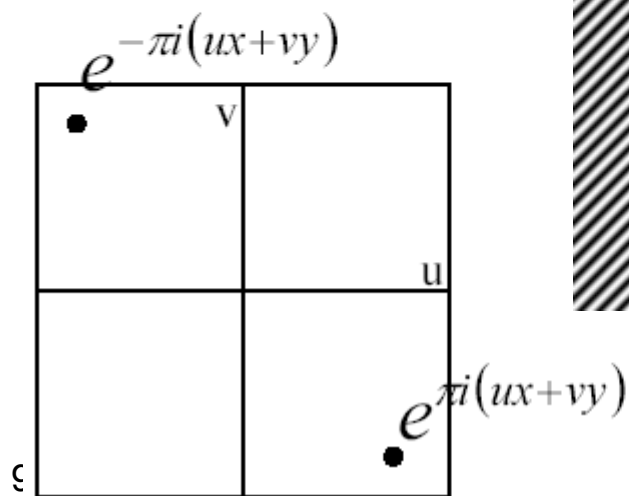
- Magnitude gives frequency
- Direction gives orientation.



Here u and v are
larger than in the
previous slide.

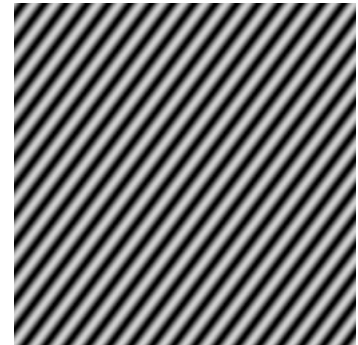
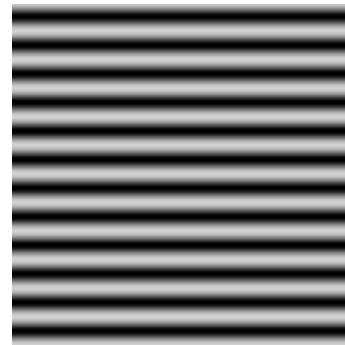
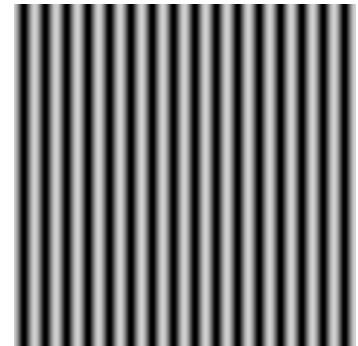


And larger still...

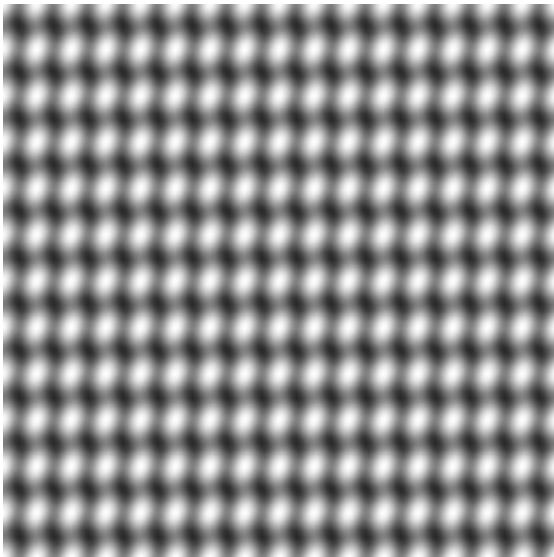


The Fourier “Hammer”

“Power Spectrum”

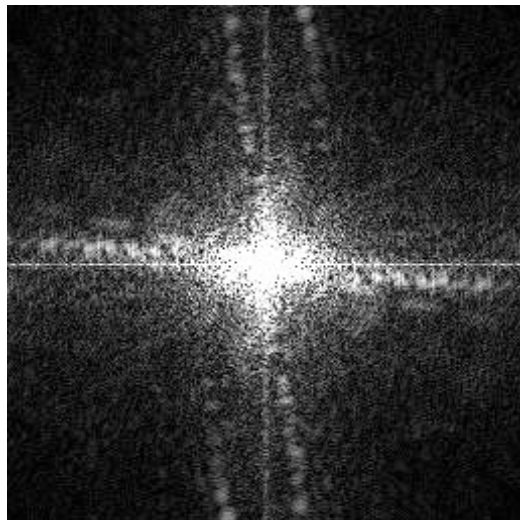


Linear Combination:

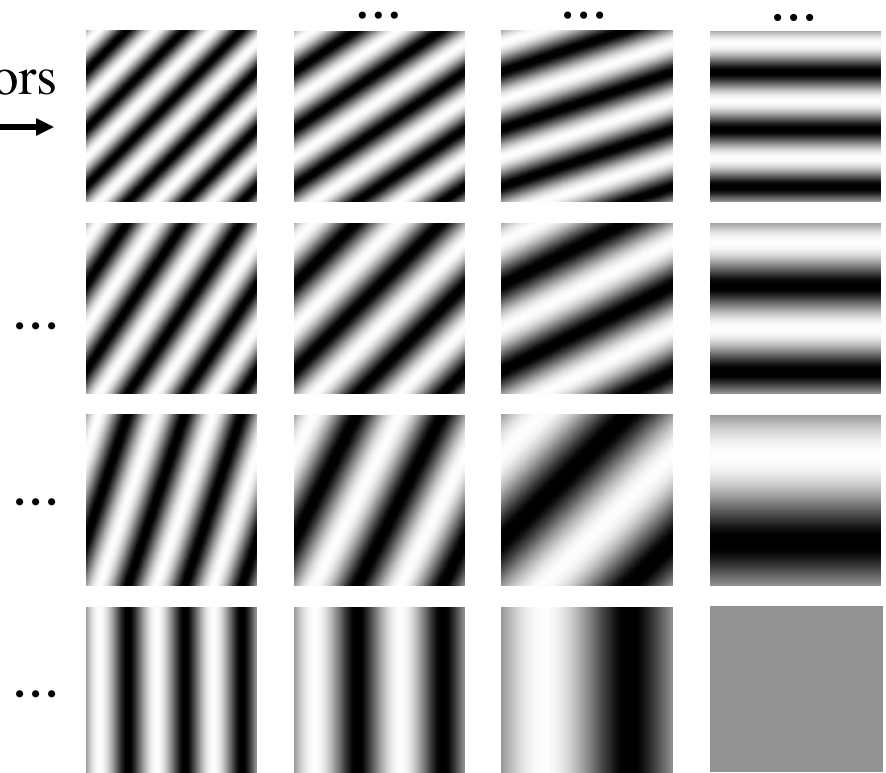


Basis vectors

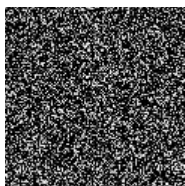
Frequency Decomposition



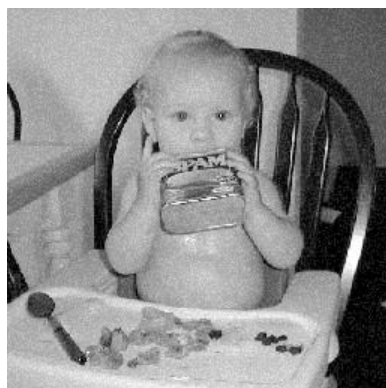
All Basis Vectors
→
←
Example



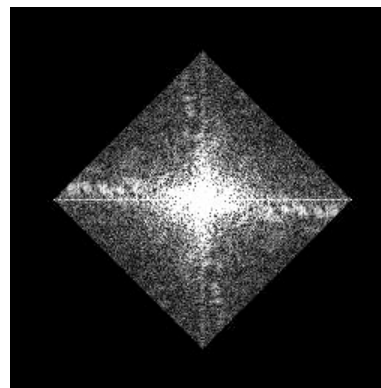
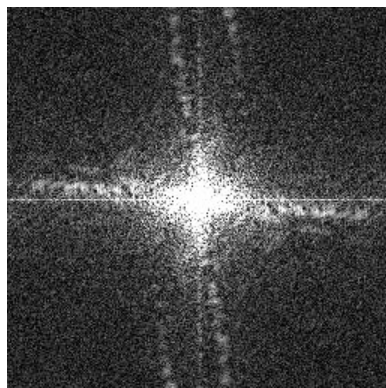
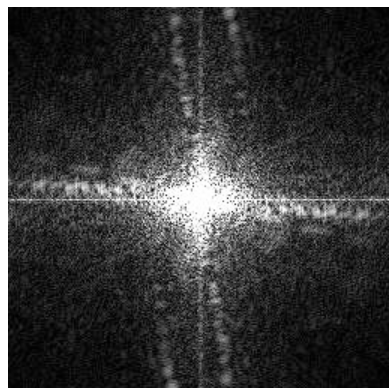
intensity \sim that frequency's coefficient



Using Fourier Representations



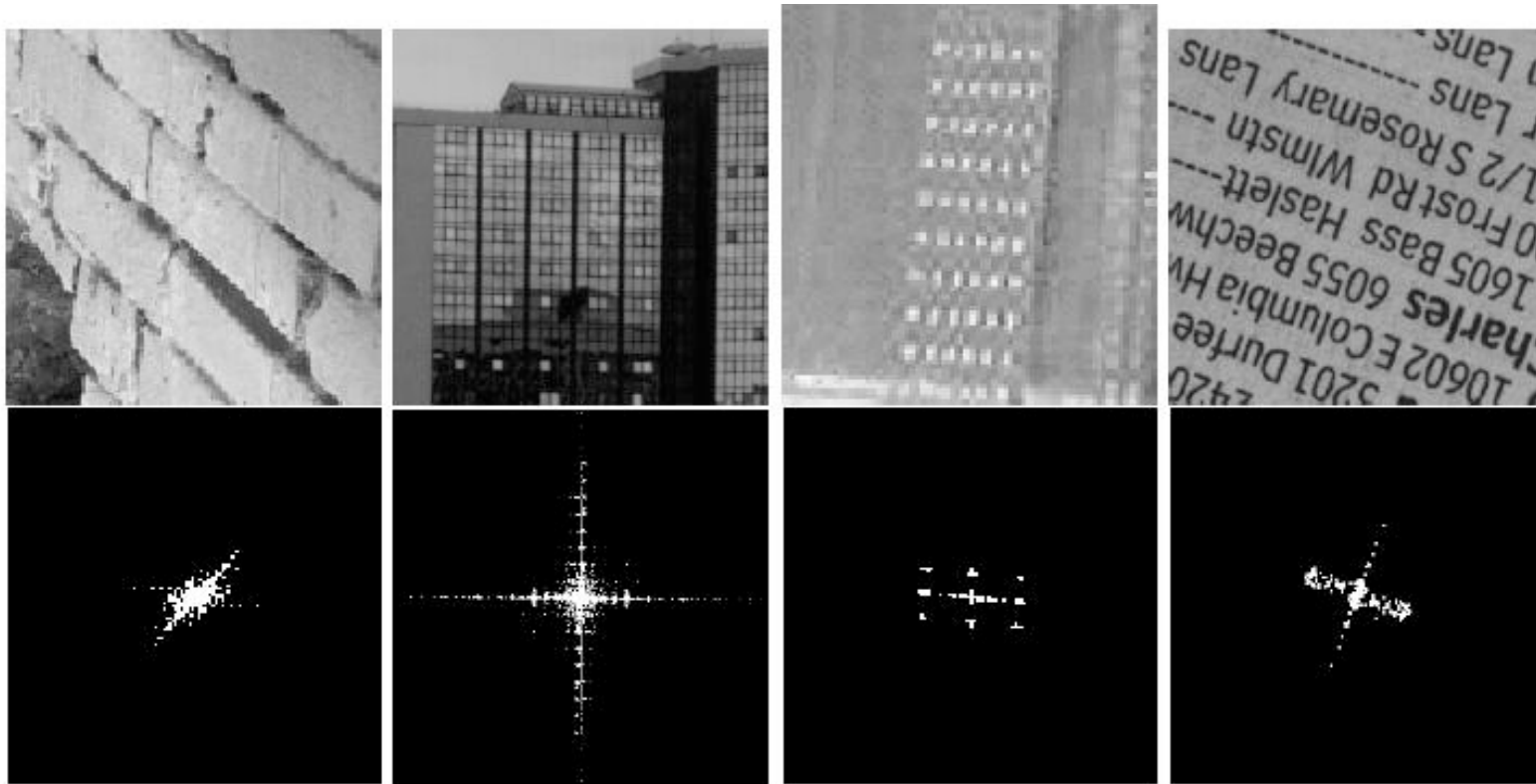
Smoothing



Data Reduction: only use *some* of the existing frequencies

Using Fourier Representations

Dominant Orientation



Limitations: not useful for local segmentation

Phase and Magnitude

$$e^{it} = \cos t + i \sin t$$

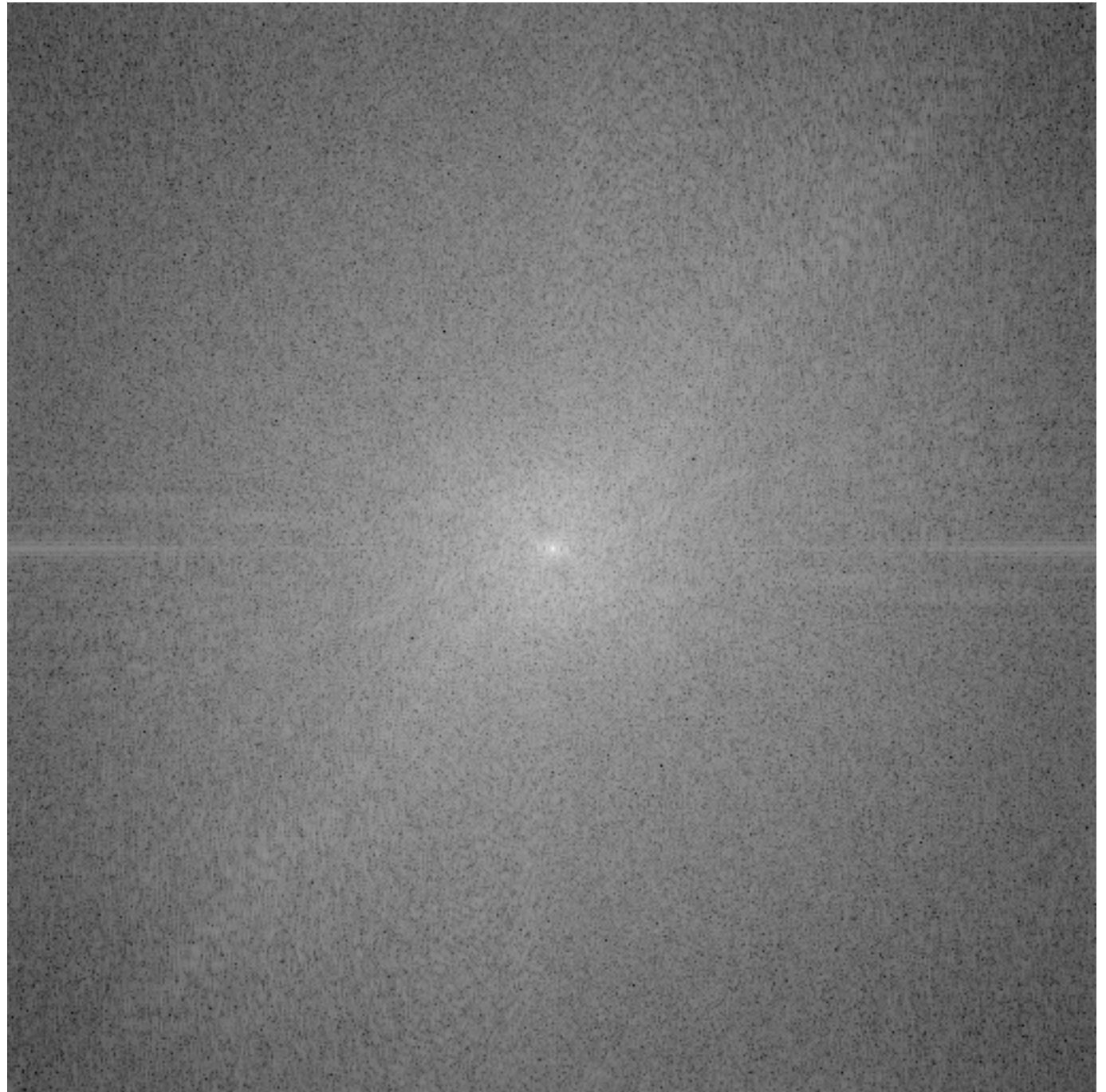
- Fourier transform of a real function is complex with real (R) and imaginary (I) components
 - difficult to plot, visualize
 - instead, we can think of the phase and magnitude of the transform
- Phase is the phase of the complex transform
 - $p(u) = \text{atan}(I(u)/R(u))$
- Magnitude is the magnitude of the complex transform
 - $m(u) = \text{sqrt}(R^2(u) + I^2(u))$
- Curious fact
 - all natural images have about the same magnitude transform
 - hence, phase seems to matter, but magnitude largely doesn't
- Demonstration
 - Take two pictures, swap the phase transforms, compute the inverse - what does the result look like?



9/28/09

CS 461 Copyright © D. Karger
Borrowed from D. Karger

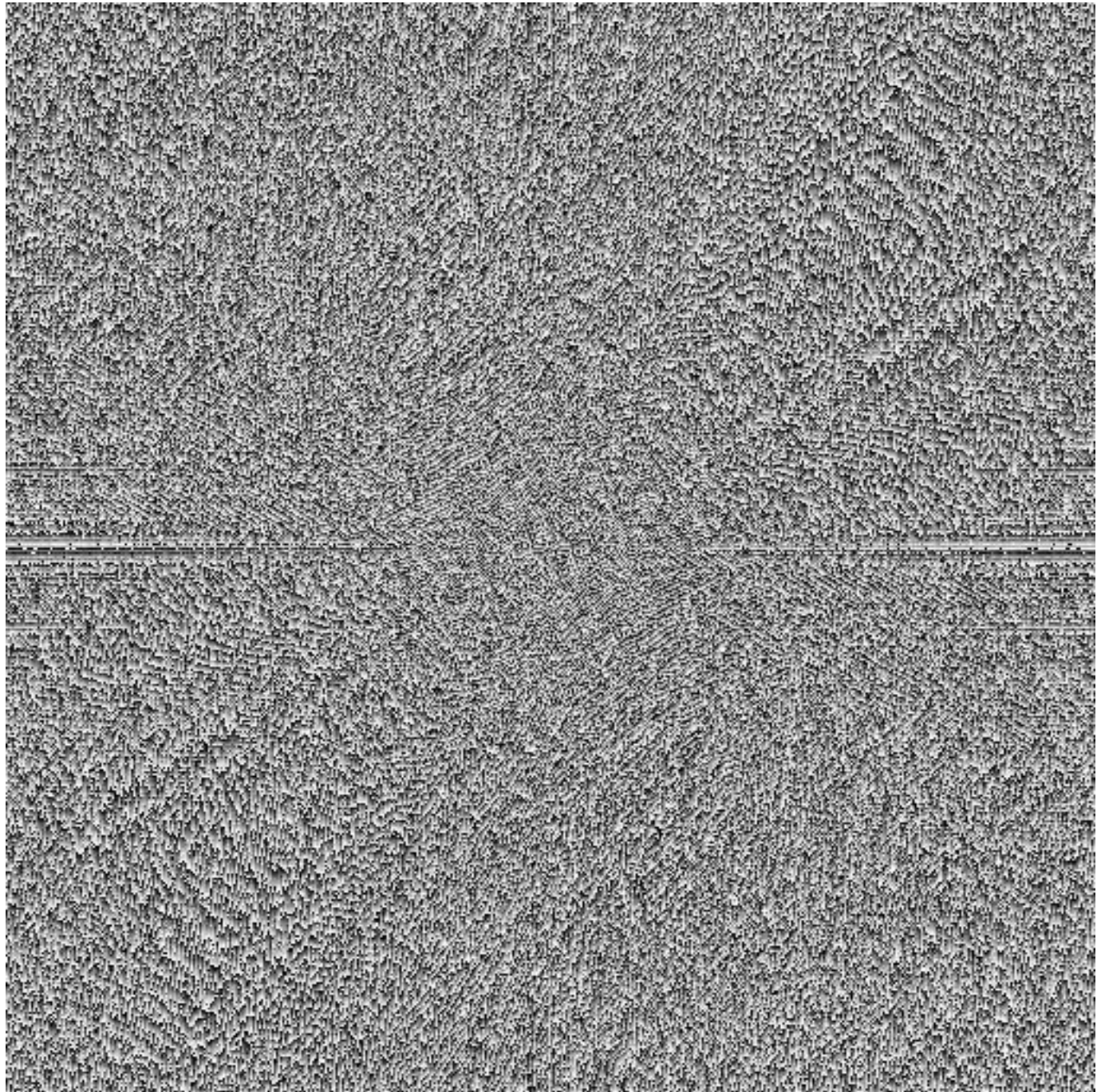
This is the
magnitude
transform
of the
cheetah pic



9/28/09

CS 461, Copyright © D. Hager
Borrowed from D. K. Regier

This is the
phase
transform
of the
cheetah pic



9/28/09

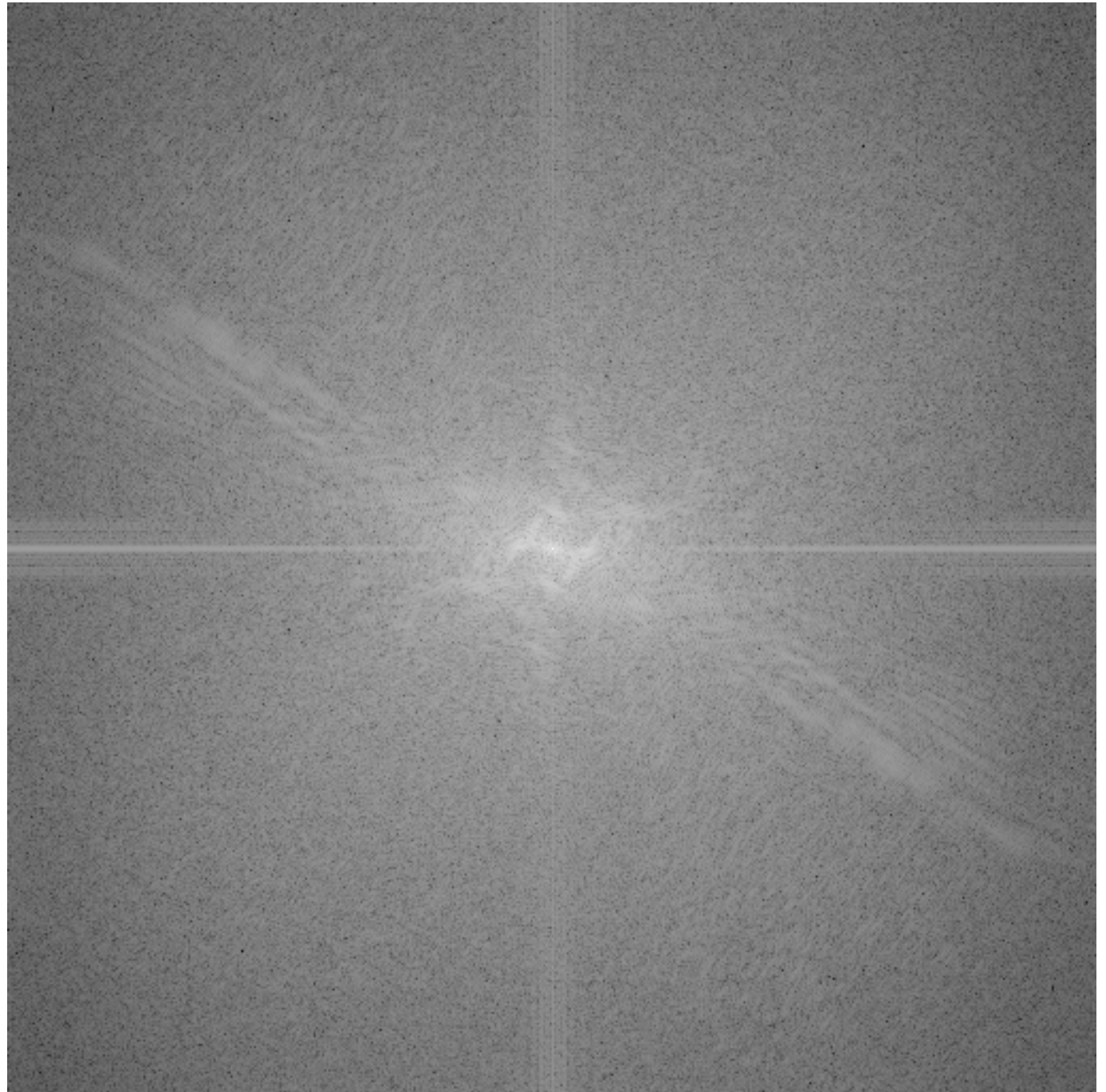
CS 461, Copyright © D. Hager
Borrowed from D. K. Regier



9/28/09

CS 461 Copyright © D. Hager
Borrowed from D. K. Regier

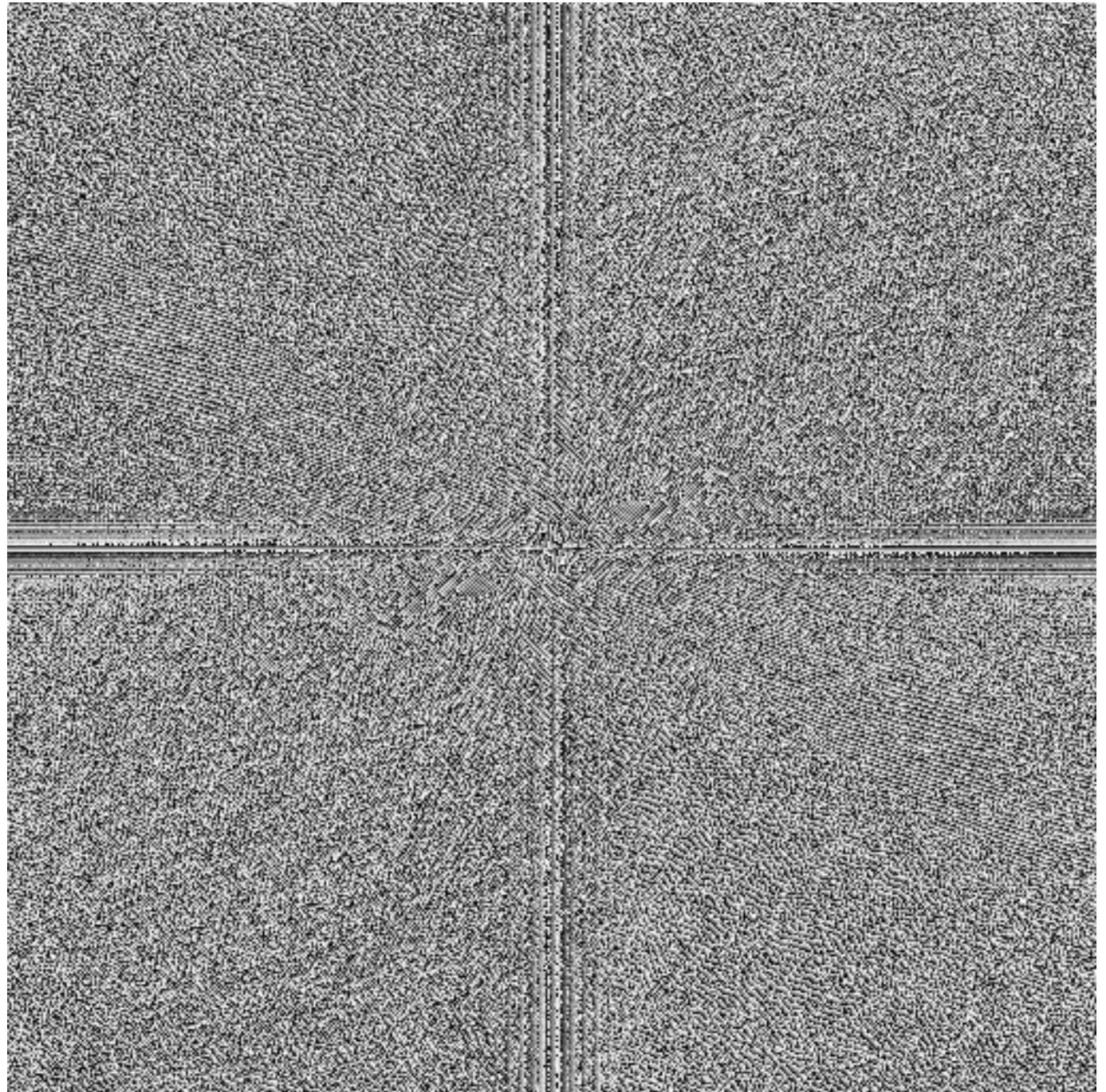
This is the
magnitude
transform
of the
zebra pic



9/28/09

CS 461, Copyright © D. Hager
Borrowed from D. K. Regier

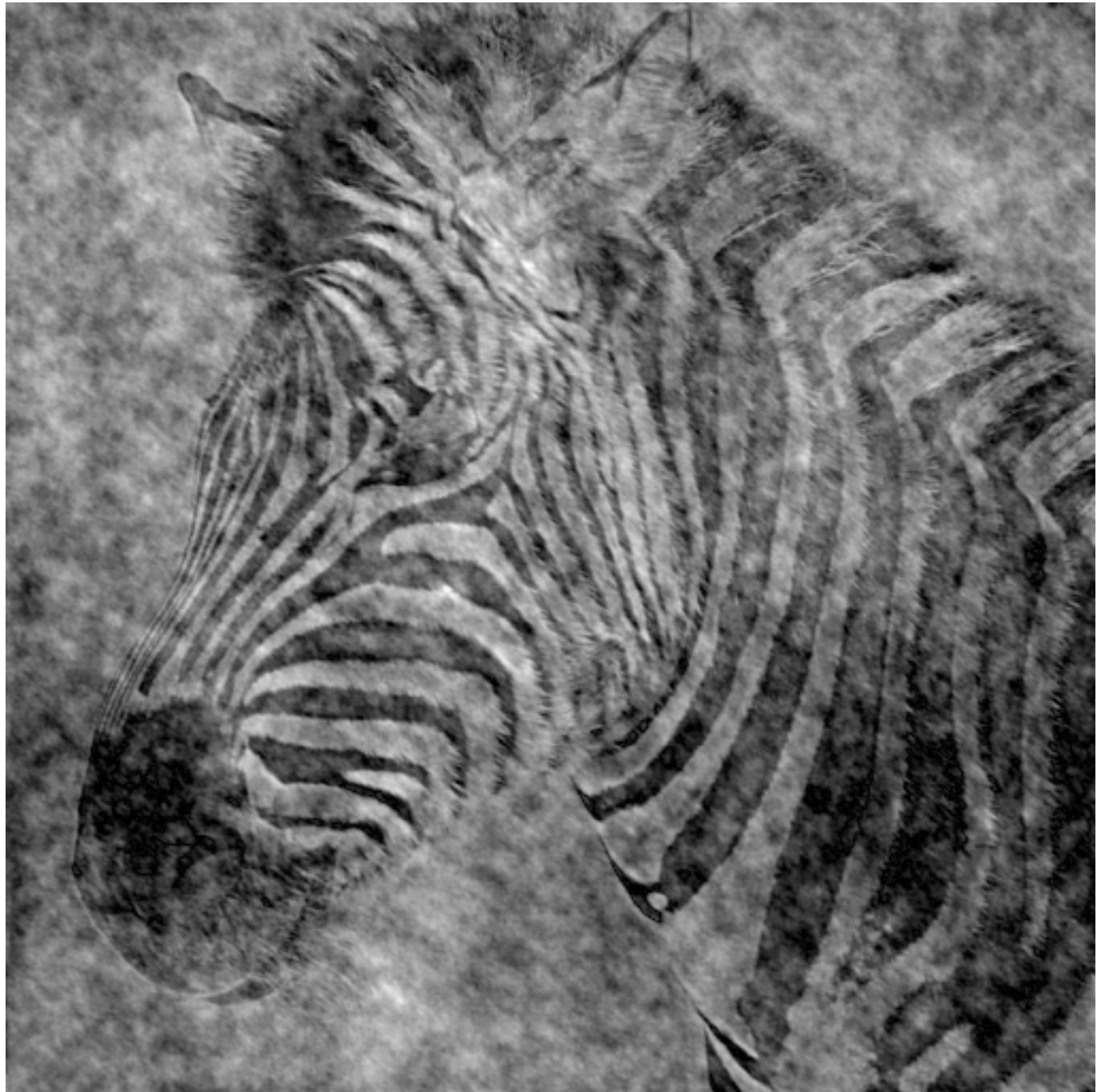
This is the
phase
transform
of the
zebra pic



9/28/09

CS 461, Copyright © D. Hager
Borrowed from D. K. Pong

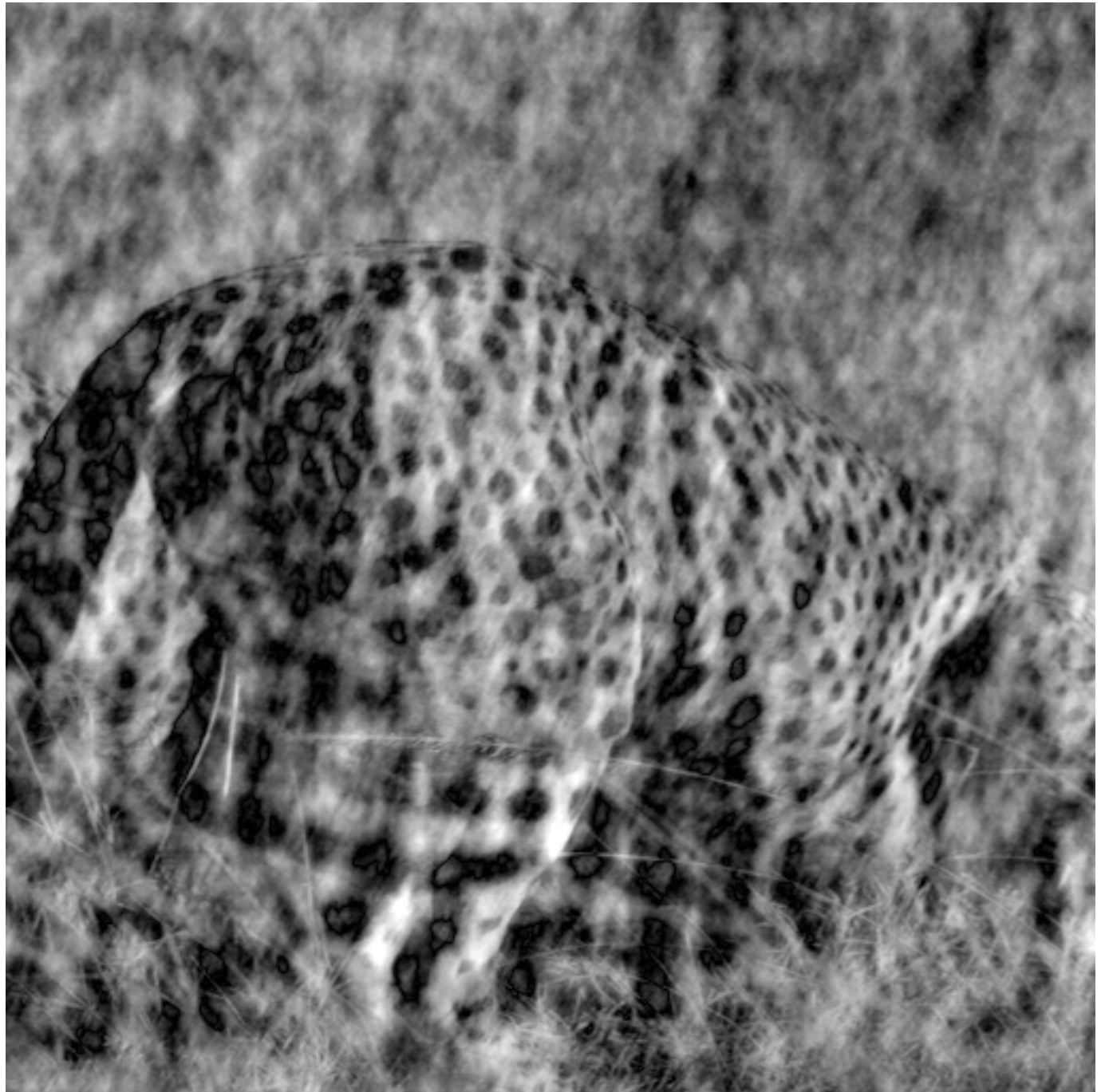
Reconstruction
with zebra
phase, cheetah
magnitude



9/28/09

CS 461, Copyright © D. Hager
Borrowed from D. K. Hegner

Reconstruction
with cheetah
phase, zebra
magnitude



9/28/09

CS 461 Copyright © D. Hager
Borrowed from D. K. Regier

The Fourier Transform and Convolution

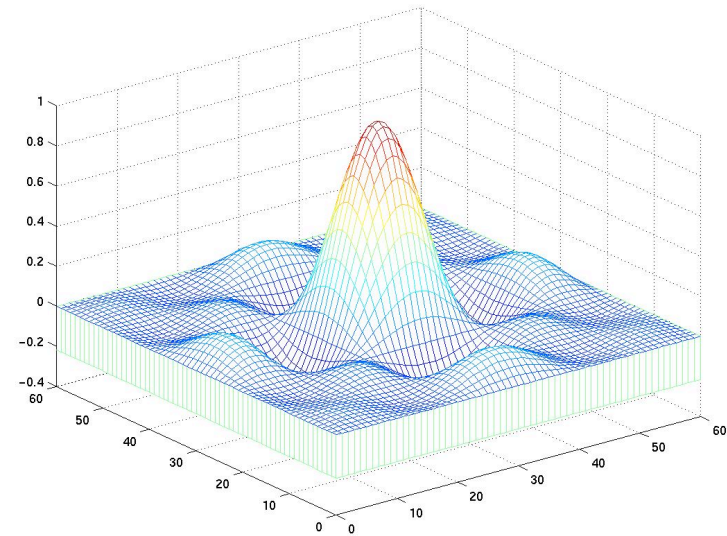
- If H and G are images, and $F(\cdot)$ represents Fourier transform, then

$$F(H * G) = F(H)F(G)$$

- Thus, one way of thinking about the properties of a convolution is by thinking of how it modifies the frequencies of the image to which it is applied.
- In particular, if we look at the power spectrum, then we see that convolving image H by G attenuates frequencies where G has low power, and amplifies those which have high power.
- This is referred to as the **Convolution Theorem**

The Properties of the Box Filter

$F(\text{mean filter}) =$



Thus, the mean filter enhances low frequencies but also has “side lobes” that admit higher frequencies

The Gaussian Filter: A Better Noise Reducer

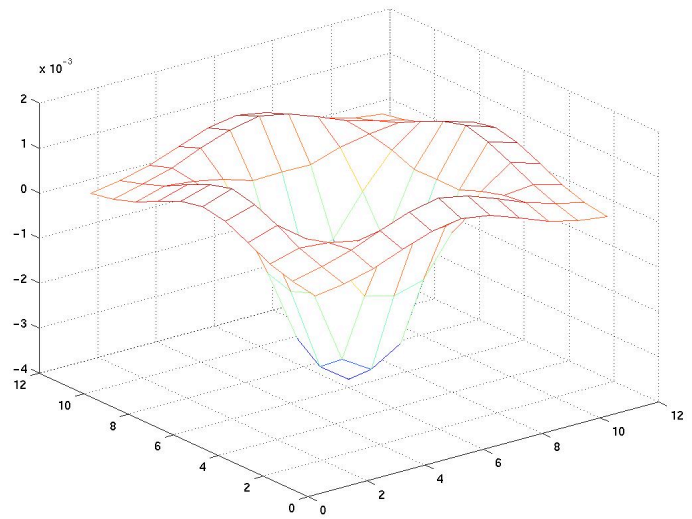
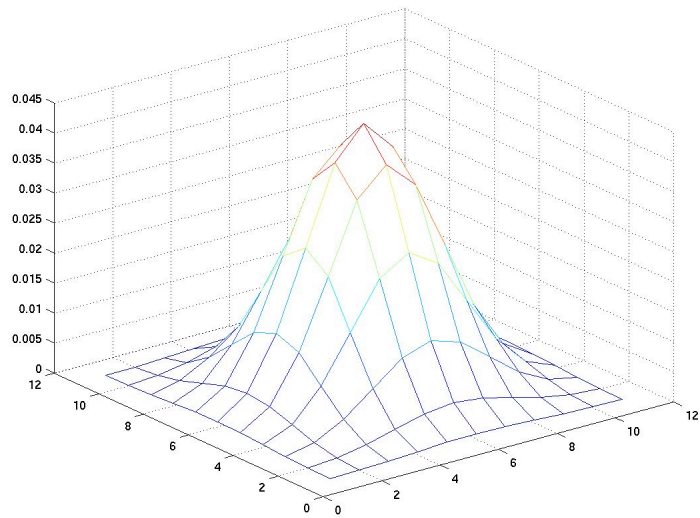
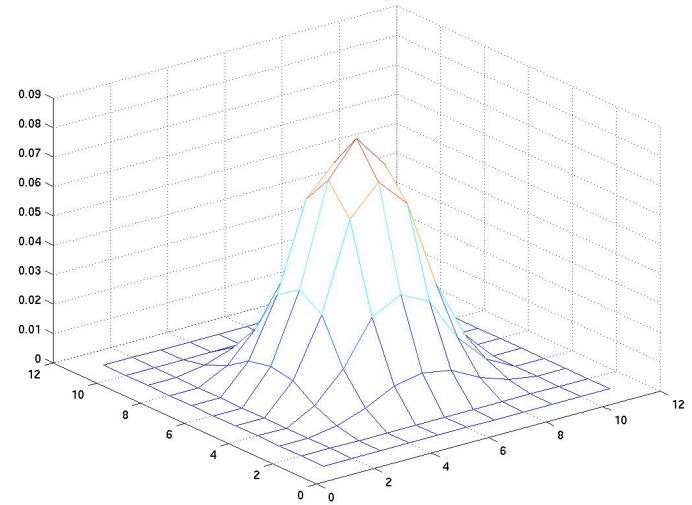
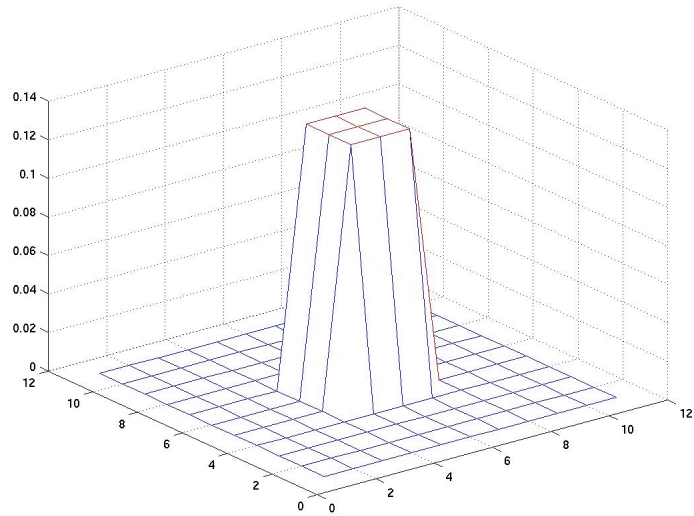
- Ideally, we would like an averaging filter that removes (or at least attenuates) high frequencies beyond a given range

$$g(i, j; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\left(i^2 + j^2\right)/2\sigma^2}$$

- It is not hard to show that the FT of a Gaussian is again a Gaussian.

- What does this imply? $\text{FT}(e^{-\alpha x^2}) = \sqrt{\frac{\pi}{\alpha}} \cdot e^{-\frac{(\pi\xi)^2}{\alpha}}$

- Note that in general, we truncate --- a good general rule is that the width (w) of the filter is at least such that $w > 5 \sigma$.
Alternatively we can just stipulate that the width of the filter determines σ (or vice-versa).
- Note that in the discrete domain, we truncate the Gaussian, thus we are still subject to ringing like the box filter.



9/28/09

Smoothing by Averaging

Kernel: 

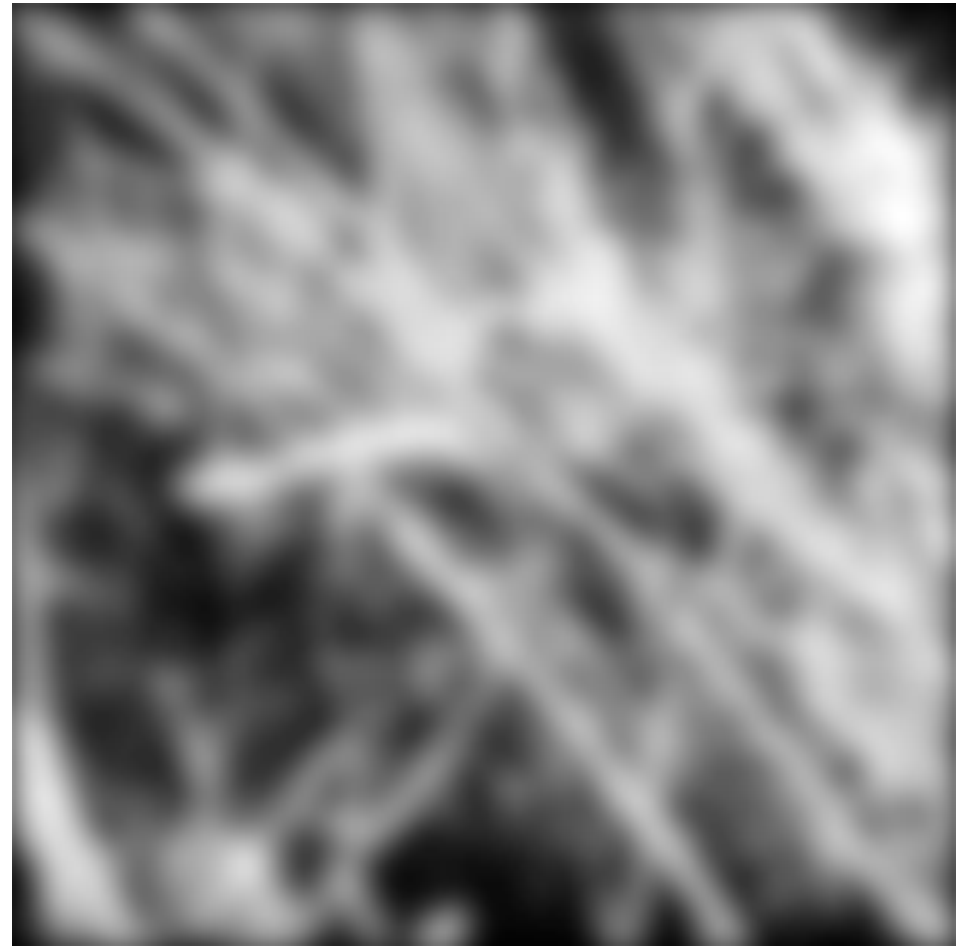


9/28/09

CS 461, Copyright © D. Hager
Borrowed from D. K. Pong

Smoothing with a Gaussian

Kernel: 



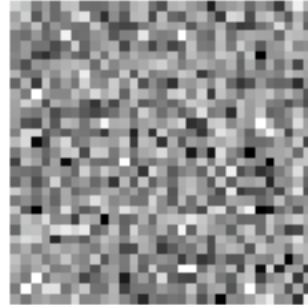
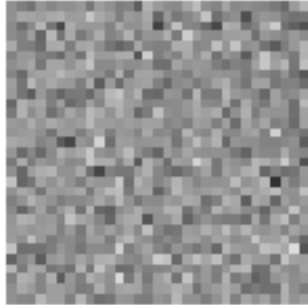
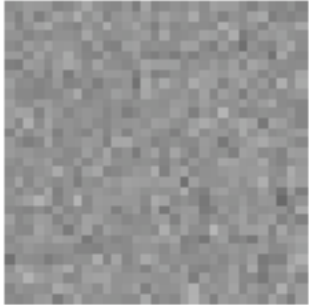
9/28/09

CS 461, Copyright © D. Hager
borrowed from D. K. Fregman

$\sigma=0.05$

$\sigma=0.1$

$\sigma=0.2$



no
smoothing



$\sigma=1$ pixel

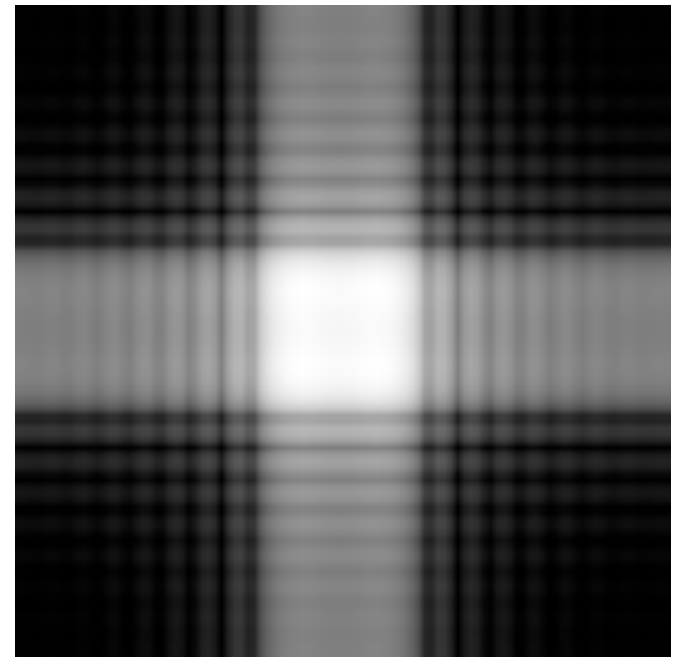
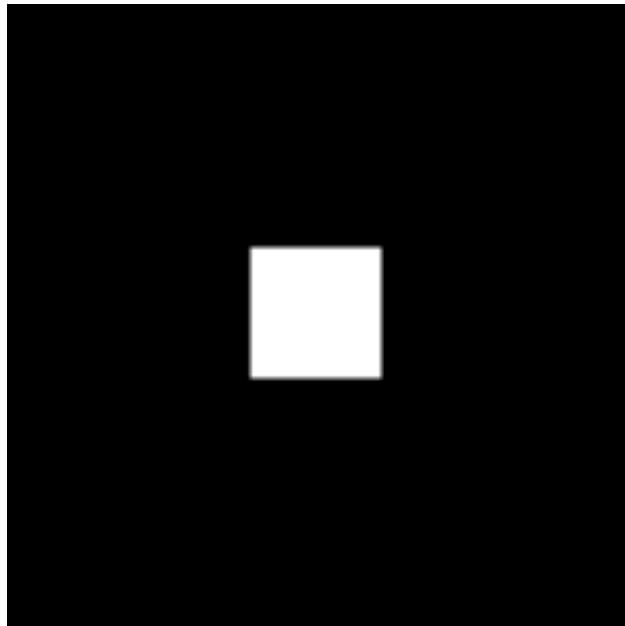


$\sigma=2$ pixels

The effects of smoothing

Each row shows smoothing with gaussians of different width; each column shows different realizations of an image of gaussian noise.

Why Not a Frequency Domain Filter?



Gabor Filters

- Fourier decompositions are a way of measuring “texture” properties of an image, but they are global
- Gabor filters are a “local” way of getting image frequency content

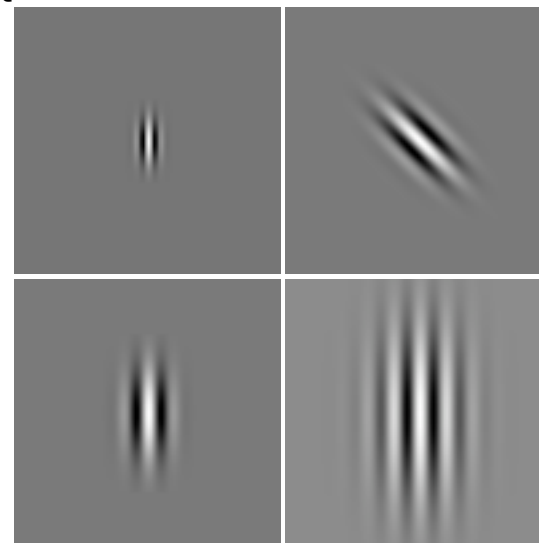
$g(x,y) = s(x,y) w(x,y)$ === a “sin” and a “weight”

$$s(x,y) = \exp(-i (2 \pi (x u + y v)))$$

$$w(x,y) = \exp(-1/2 (x^2 + y^2)/ \sigma^2)$$

Now, we have several choices to make:

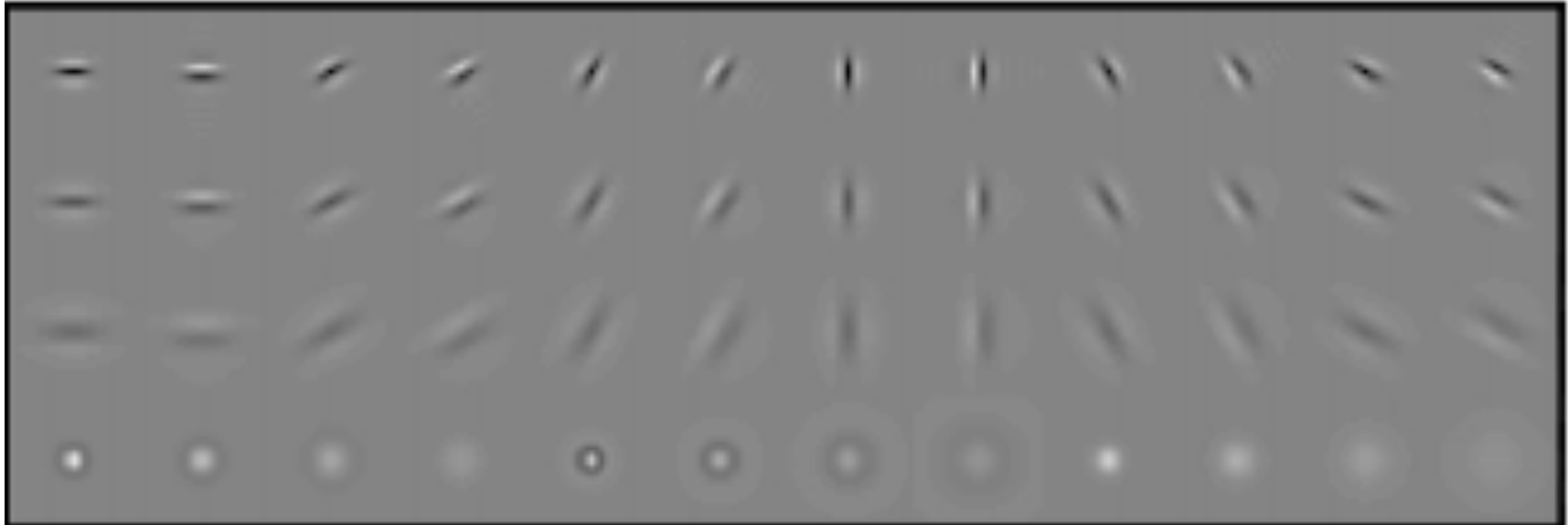
1. u and v defines frequency and orientation
2. σ defines scale (or locality)



Thus, Gabor filters for texture can be computationally expensive as we often must compute many scales, orientations, and frequencies

Another Texture Filter

- The Leung-Malik (LM Filter): set of edge and bar filters plus Gaussian and Laplacian of Gaussian (we'll see this shortly)



Computational Issues: Separability

- Recall that convolution is associative. Suppose I use the templates $g_x = \exp(-i^2/2 \sigma^2)$ and $B_y = g_y = \exp(-j^2/2 \sigma^2)$ Then
 - $g_x * (g_y * I) = (g_x * g_y) * I$
 - but, it is not hard to show that the first convolution is simply the 2-D Gaussian that we defined previously!
- In general, this means that we can “separate” the 2-D Gaussian convolution into 2 1-D convolutions with great computational cost savings.
- A good exercise is to show that the box filter is also separable.

Computational Issues: Minimizing Operations

- Note that for a 256 gray level image, we can *precompute* all values of the convolution and avoiding multiplication.
- For the box filter, we can implement any size using $4n$ additions per pixel.
- Also note that, by the central limit theorem, repeated box filter averaging yields approximations to a Gaussian filter.
- Finally, note that a sequence of filtering operations can be collapsed into one by associativity.
 - in general, this is not a win, but we'll see examples where it is ...

Other Types of “Noise”

- Digitization artifacts:
 - Blooming
 - Aliasing
 - Quantization
- Impulsive noise
 - randomly pick a pixel and randomly set to a value
 - saturated version is called salt and pepper noise
- Unanticipated/undesirable image structures
 - Also often called noise although it is a real repeatable signal.

Non-Noise Artifacts: Digitization Effects

- The “diameter” d of a pixel determines the highest frequency representable in an image by the Nyquist rate

$$f_{\text{samp}} = 1/d > 2f_{\text{max}} \Rightarrow f_{\text{max}} < 1/(2d)$$

- Real scenes may contain higher frequencies resulting in aliasing of the signal.
- In practice, this effect is often dominated by other digitization artifacts.
- This is issue when *resampling* an image
 - Rotating/warping
 - Resizing

Filter Pyramids

- Recall we can always filter with $G(\sigma)$ for any σ
- As a result, we can think of a continuum of filtered images as σ grows.
 - This is referred to as the “scale space” of the images. We will see this show up several times.
- As a related note, suppose I want to subsample images
 - subsampling reduces the highest frequencies
 - averaging reduces noise
 - Pyramids are a way of doing both

Gaussian Pyramid

- Algorithm:
 - 1. Filter with $G(\sigma=1)$
 - 2. Resample at every other pixel
 - 3. Repeat



Limitations of Linear Operators on Impulsive Noise



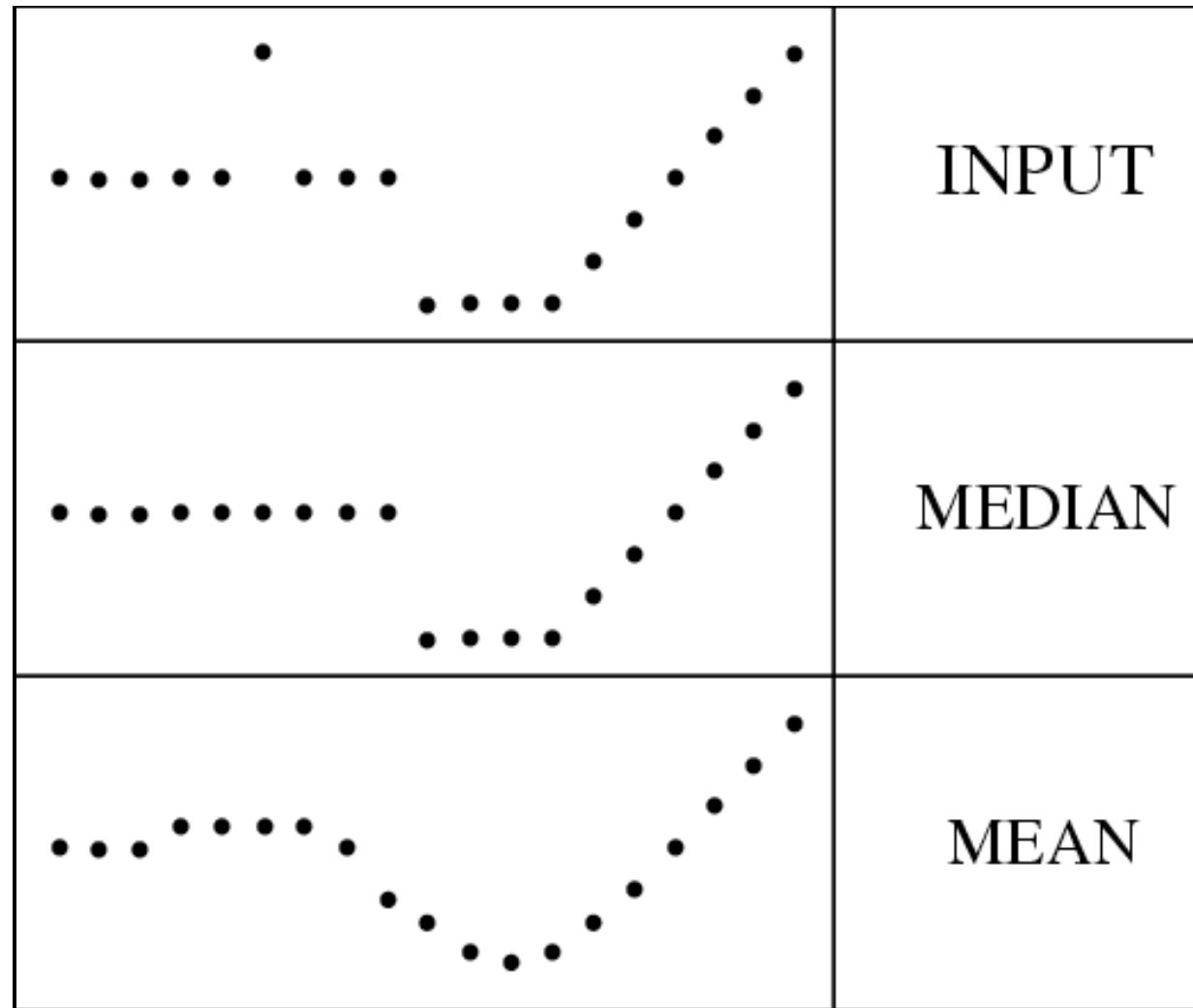
Nonlinear Filtering: The Median Filter

Suppose I look at the local statistics and replace each pixel with the *median* of it's neighbors:



Median filters : example

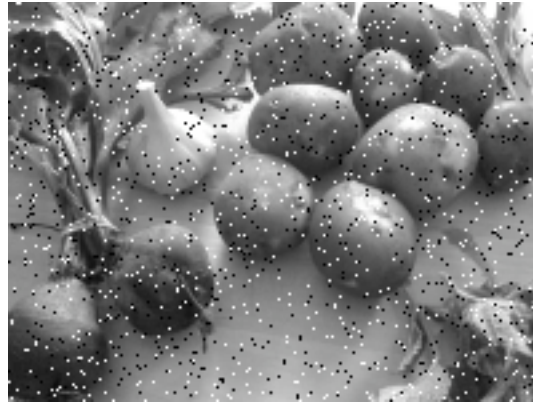
filters have width 5 :



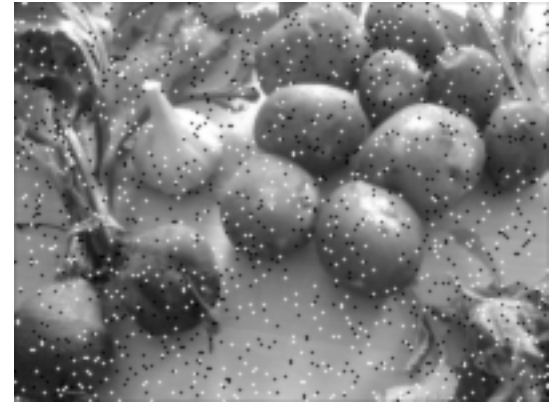
Median Filtering: Example



Original



Salt and Pepper



Gaussian Filter



Median Filter

Morphological Operators

Let S_t be the *translation* of a set of pixels S by t .

$$S_t = \{ x + t \mid x \in S \}$$

The *dilation* of a binary image A by a mask B^s is then

$$(f \oplus b)(x) = \sup_{z \in B^s} f(z - x)$$

The *erosion* of a binary image A by a mask E is

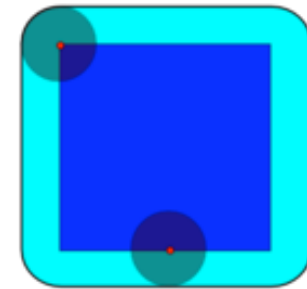
$$(f \ominus b)(x) = \inf_{y \in E} [f(y) - b(y - x)]$$

The *closing* of an image A by S is

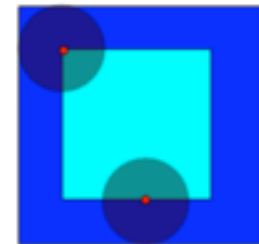
$$A \bullet S = (A \oplus S) \ominus S$$

The *opening* of an image A by S is

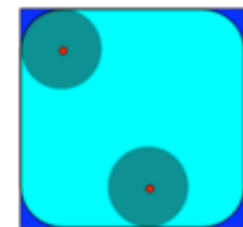
$$A \circ S = (A \ominus S) \oplus S$$



Dilation



Erosion



Opening

Gray-Scale Morphology



Opening



Closing

The “Poor Man’s” Closing

- Note that median (or more generally any order statistic) filtering is one way of achieving similar effects. On binary images this can be also implemented using the averaging filter

What Else Can You Do With Convolution?

- Thus far, we've only considered convolution kernels that are smoothing filters.
- Consider the following kernel:
 - $[1, -1]$



What Else Can You Do With Convolution?

- Thus far, we've only considered convolution kernels that are smoothing filters.
- Consider the following kernel:
 - $[1; -1]$



Physical causes of edges

1. Object boundaries
2. Surface normal discontinuities
3. Reflectance (albedo) discontinuities
4. Lighting discontinuities

Object Boundaries



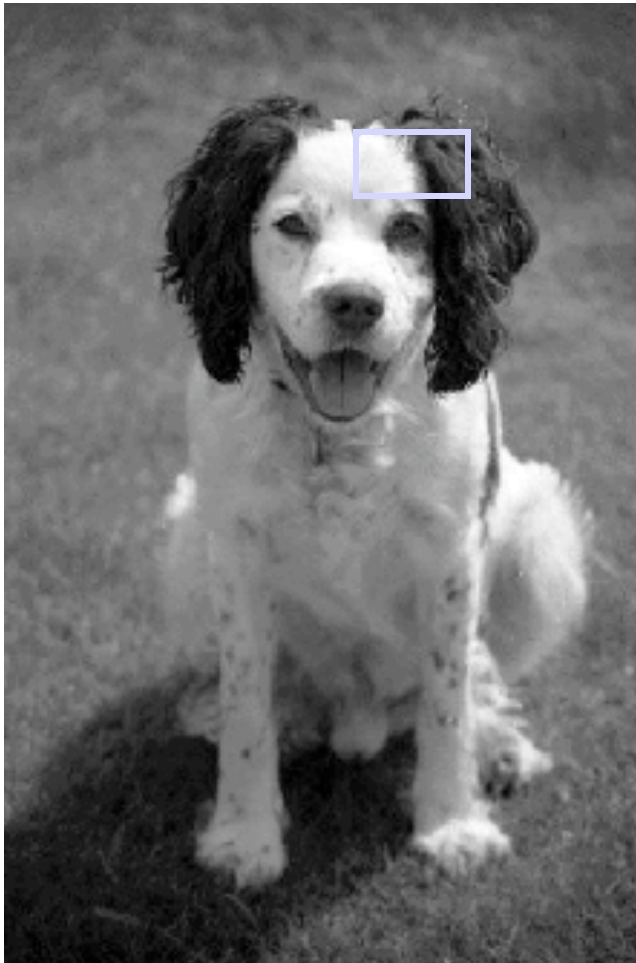
Surface normal discontinuities



9/26/09

CS 461, Copyright G.D. Hager

Boundaries of material properties



9/26/09



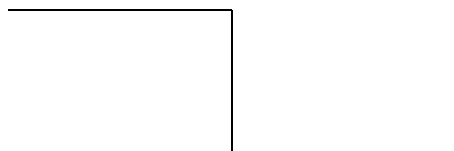
Boundaries of lighting



9/26/09

CS 461, Copyright G.D. Hager

Edge Types



Step



Ridge

Which of these do you suppose
a derivative filter detects best?



Roof

The Image Gradient

- Recall from calculus for a function of two variables $f(x,y)$:
 - The gradient: points in the direction of maximum increase.
 - Its magnitude is proportional to the rate of increase.
 - The total derivative in the direction n is $\text{dot}(n, \text{grad}(f))$
- The kernel $[1,-1]$ is a way of computing the x derivative
- The kernel $[1;-1]$ is a way of computing the y derivative

Some Other Interesting Kernels

The Roberts Operator

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

The Prewitt Operator

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Some Other Interesting Kernels

The Sobel Operator

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

The Laplacian Operator

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

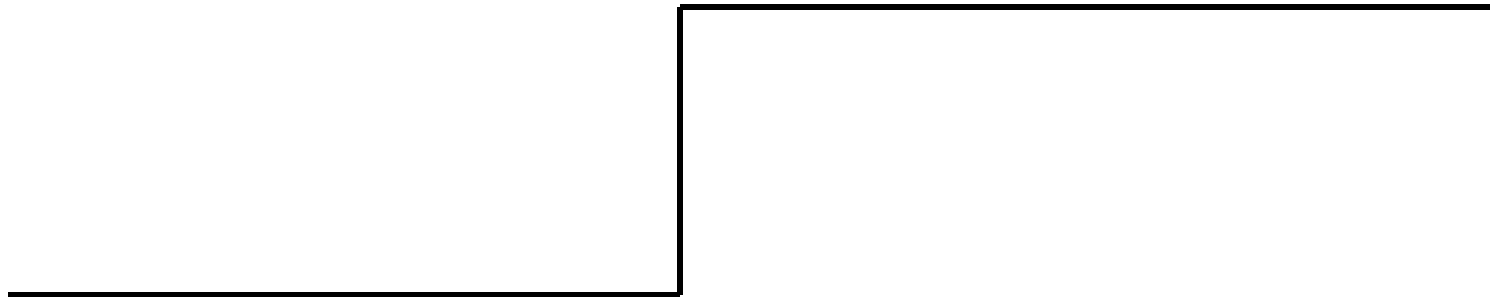
A good exercise: derive the Laplacian from 1-D derivative filters.

Note the Laplacian is rotationally symmetric!

Edge is Where Change Occurs

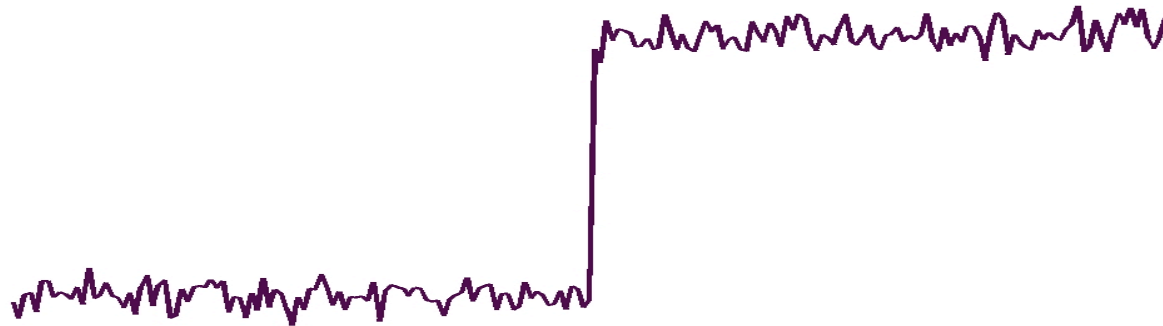
1-D

- Change is measured by derivative in 1D
 - Biggest change, derivative has maximum magnitude
 - Or 2nd derivative is zero.



Noisy Step Edge

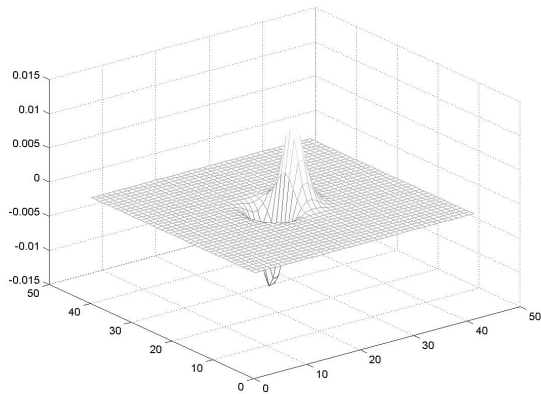
- Derivative is high everywhere.
- Must smooth before taking gradient.



Smoothing Plus Derivatives

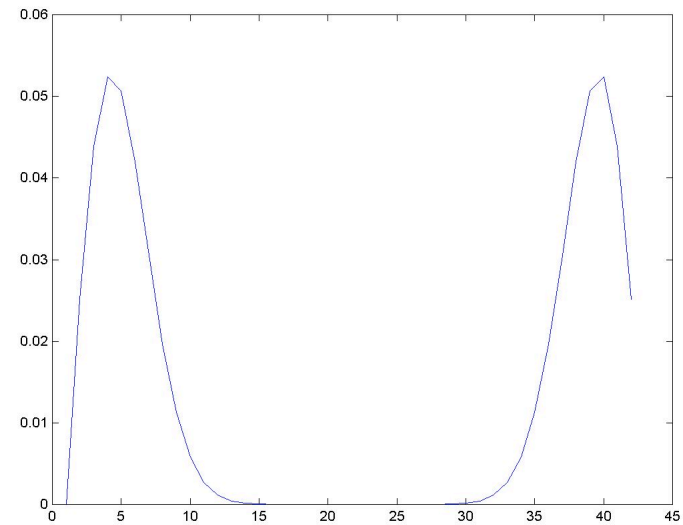
- One problem with differences is that they by definition reduce the signal to noise ratio (can you show this?)
- Recall smoothing operators (the Gaussian!) reduce noise.
- Hence, an obvious way of getting clean images with derivatives is to combine derivative filtering and smoothing: e.g.
 - $(I * G) * D_x = I * (D_x * G)$

The Fourier Spectrum of DOG



Derivative of a Gaussian

PS of central slice



Properties of the DoG operator

- Now, going back to the directional derivative:
 - $D_u(f(x,y)) = f_x(x,y)u_1 + f_y(x,y)u_2$
- Now, including a Gaussian convolution, we see
 - $D_u[G*I] = D_u[G]*I = [u_1G_x + u_2G_y]*I = u_1G_y*I + u_2G_x*I$
- The two components $I*G_x$ and $I*G_y$ are the *image gradient*
- Note the directional derivative is maximized in the direction of the gradient
- (note some authors use DoG as “Difference of Gaussian” which we’ll run into soon)

Algorithm: Simple Edge Detection

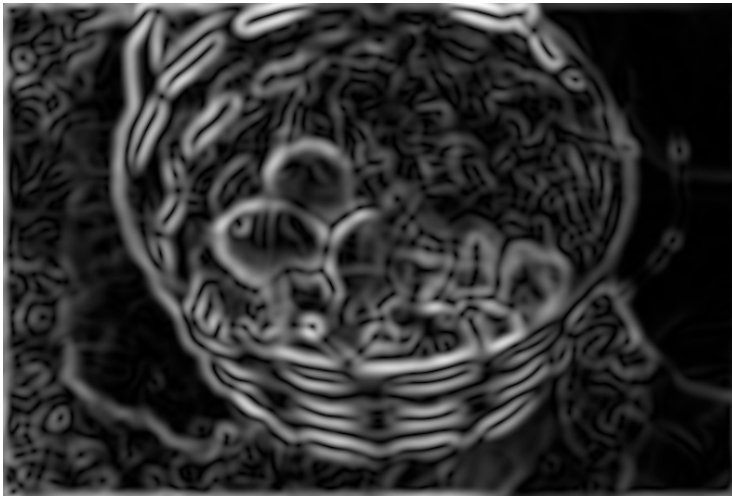
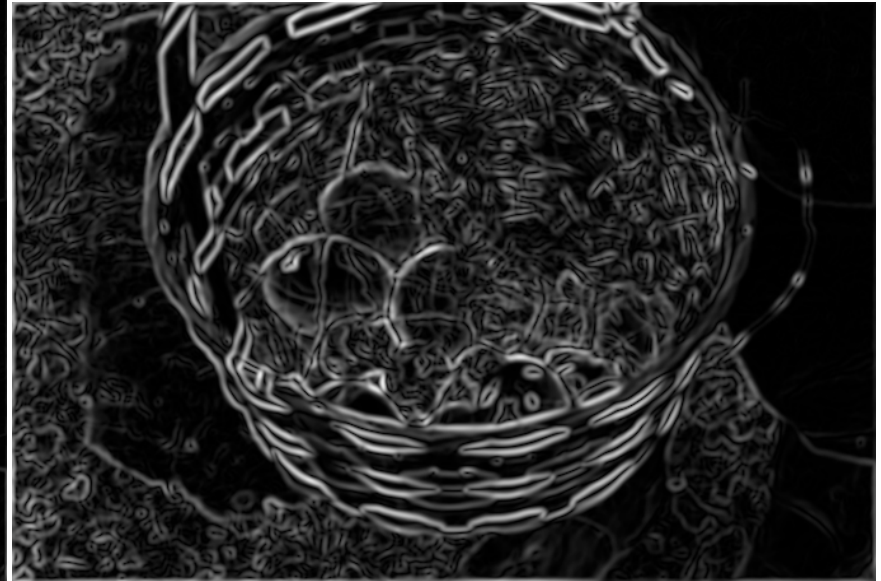
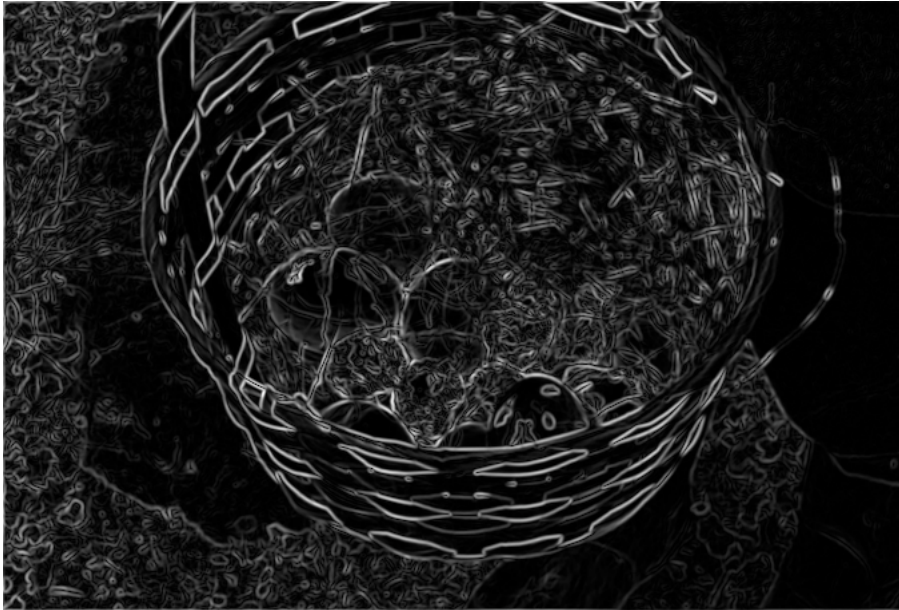
1. Compute $I_x = I_g * (G(\sigma) * G(\sigma)' * [1, -1; 1, -1])$
2. Compute $I_y = I_g * (G(\sigma) * G(\sigma)' * [1, -1; 1, -1]')$
3. Compute $I_{mag} = \text{sqrt}(I_x.^* I_x + I_y.^* I_y)$
4. Threshold: $I_{res} = I_{mag} > \tau$

It is interesting to note that if we wanted an edge detector for a specific direction of edges, we can simply choose the appropriate projection (weighting) of the component derivatives.

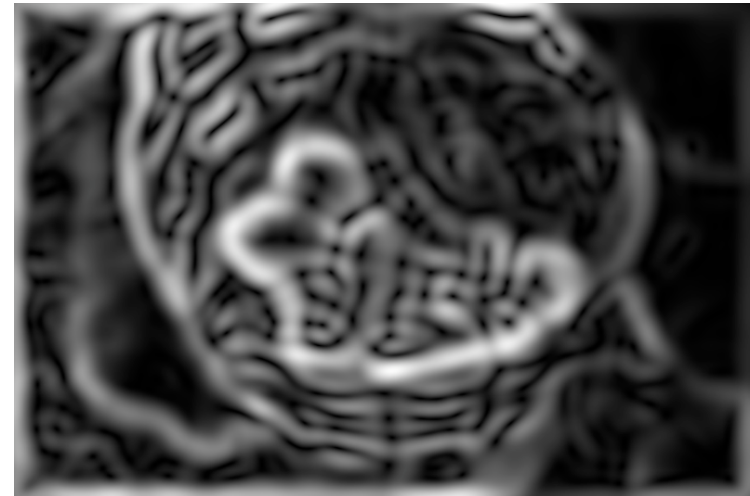
$\sigma = 1$

Example

$\sigma = 2$



$\sigma = 5$



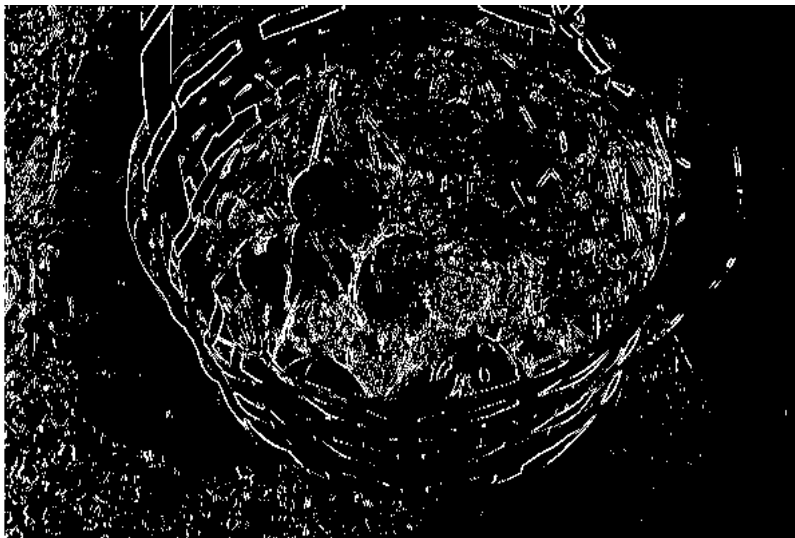
$\sigma = 10$

9/28/09

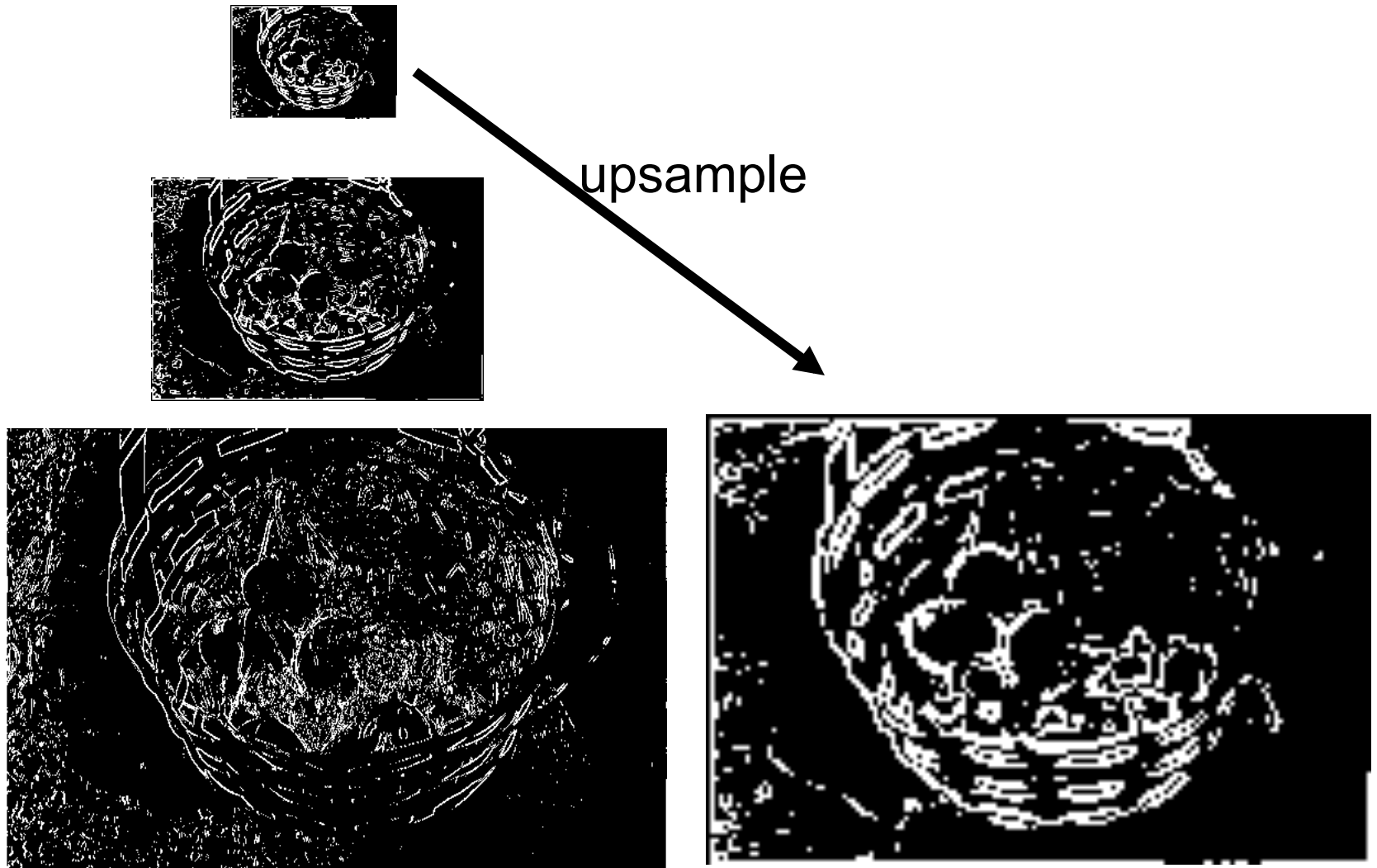
Laplacian Pyramid Algorithm

- Create a Gaussian pyramid by successive smoothing with a Gaussian and down sampling
- Set the coarsest layer of the Laplacian pyramid to be the coarsest layer of the Gaussian pyramid
- For each subsequent layer $n+1$, compute
 - $L(n+1) = G(n+1) - \text{upsample}(G(n))$
- In general, the idea of using a series of Gaussians with different values of σ leads to the idea of *scale space* in computer vision
$$L(\sigma) = I * G(\sigma)$$
 - i.e. a continuous family of images

Laplacian of Gaussian Pyramid



Laplacian of Gaussian Pyramid



Summary

- Linear filters as
 - Smoothing
 - Texture extraction
 - Gradient operators
 - Laplacian
- Fourier transforms
 - What they represent
 - Convolution Theorem
- Computational Issues
 - Padding
 - Using associative property to regroup
 - Using analytical means to simplify convolutions
 - Separability