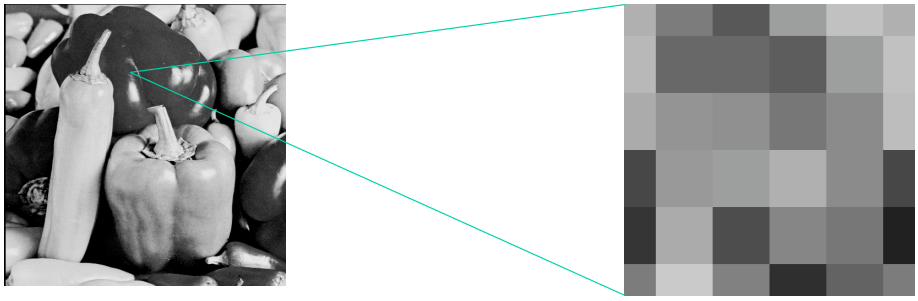
Computer Vision
Linear Filtering and Edge Detection

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

Outline

- Image noise
- Filtering by Convolution
- Properties of Convolution
- Derivative Operators
- The Canny Edge Detector

IMAGE NOISE



Cameras are not perfect sensors
Scenes never quite match our expectations

Noise Models

- Noise is commonly modeled using the notion of "additive white noise."
  - Scalar example: $I(x) = I^*(x) + n(x)$
  - Images: $I(i,j,t) = I^*(i,j,t) + n(i,j,t)$
  - Note that $n(i,j,t)$ is independent of $n(i',j',t')$ unless $i'=i, j'=j, t'=t$.
  - Typically we assume that n (noise) is independent of image location --- that is, it is i.i.d

Properties of Noise Processes

- Properties of temporal image noise:

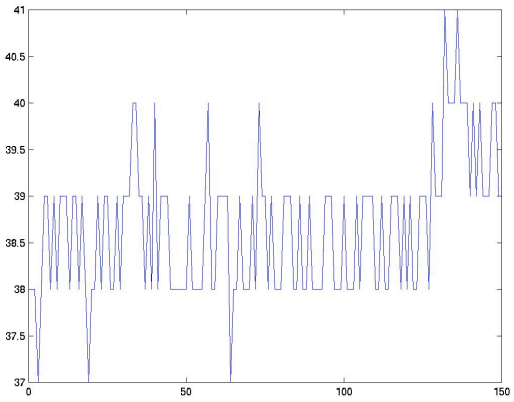
$$\text{Mean} \quad \mu(i,j) = \sum_t I(i,j,t)$$

$$\text{Standard Deviation} \quad \sigma_{i,j} = \text{Sqrt}\left(\sum_t (\mu - I(i,j,t))^2 \right)$$

$$\text{Signal-to-noise Ratio} \quad \frac{\sigma_{I*}}{\sigma_{i,j}}$$

Image Noise

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



mean = 38.6
std = 2.99

max snr = 255/3 = 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:

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

Temporal vs. Spatial Noise

- It is common to assume that:
  - spatial noise in an image is consistent with the temporal image noise
  - the noise is independent and identically distributed

- Thus, we can think of the image itself as an additive noise process

Gaussian Noise: sigma=1

10/15/04

Gaussian Noise: sigma=16

10/15/04

How to reduce noise

- Averaging is a common way to reduce noise
  - instead of temporal averaging, how about spatial?
- 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

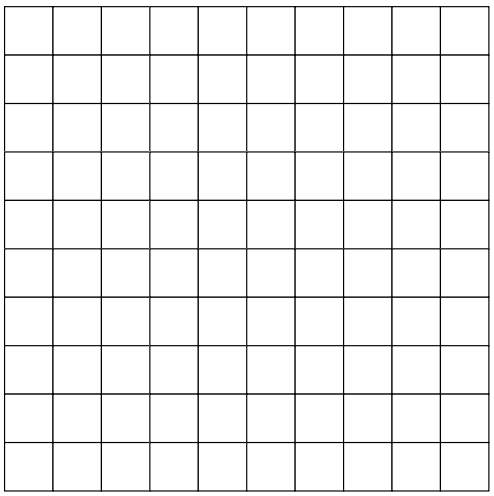


Image (I)

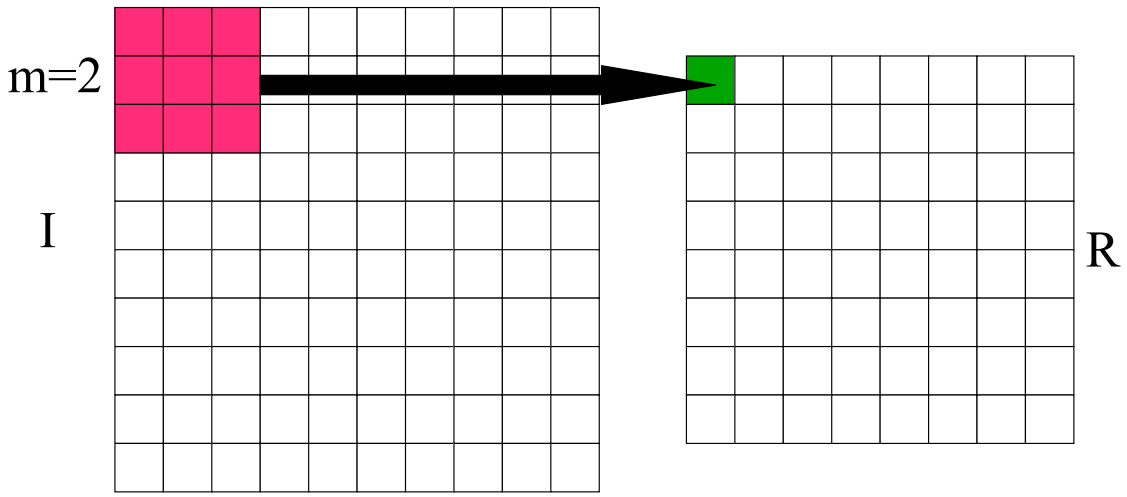
*

1	2	1
-1	-2	-1

Kernel (K)

Note: Typically Kernel is relatively small in vision applications.

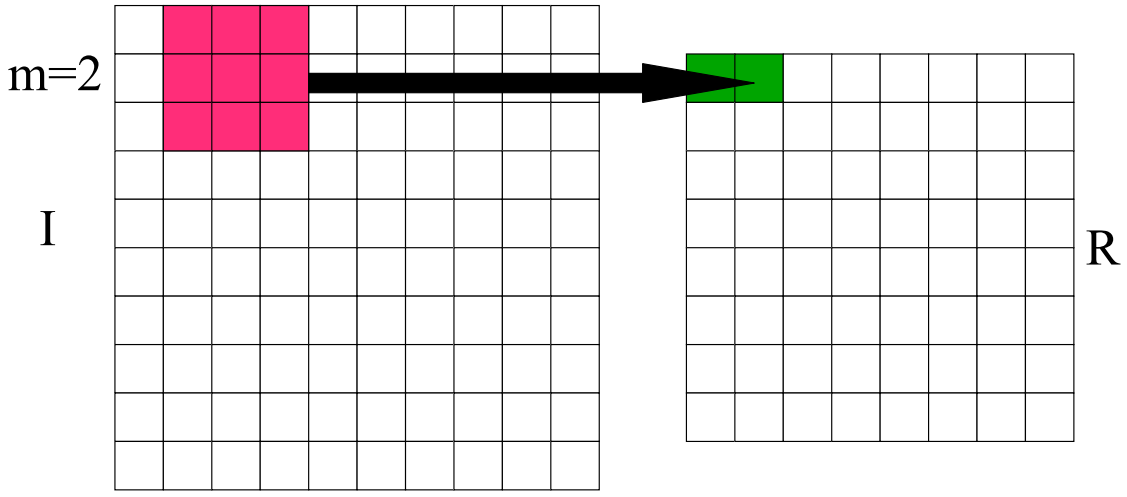
Convolution: R= K*I



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



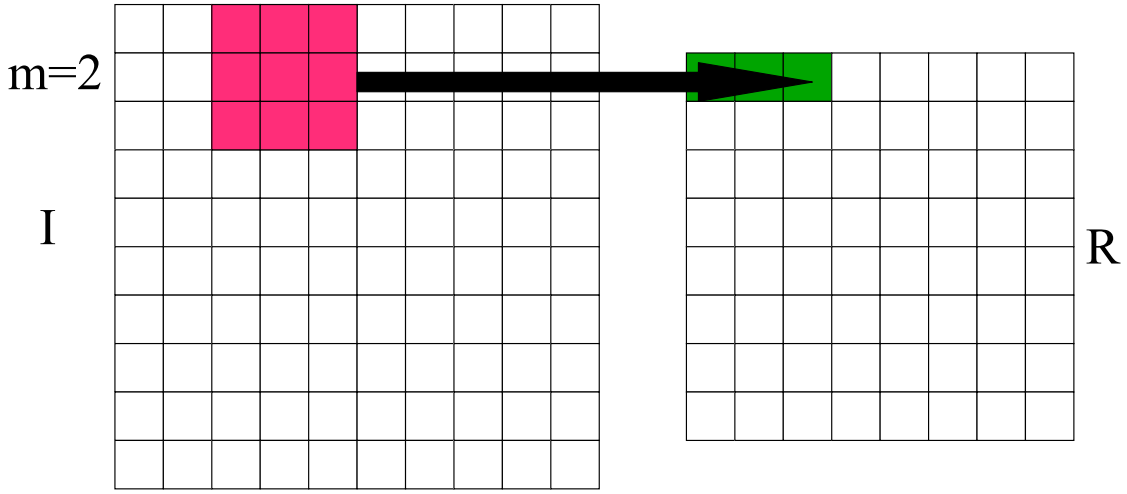
Kernel size
is m+1 by m+1

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

10/15/04

CS 461, Copyright G.D. Hager
borrowed from D. Kriegman

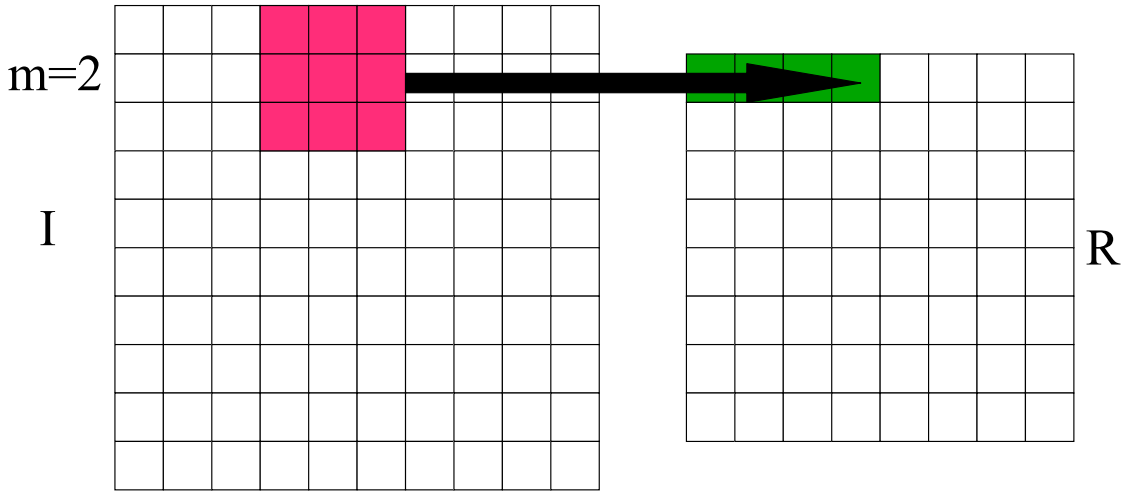
Convolution: R= K*I



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



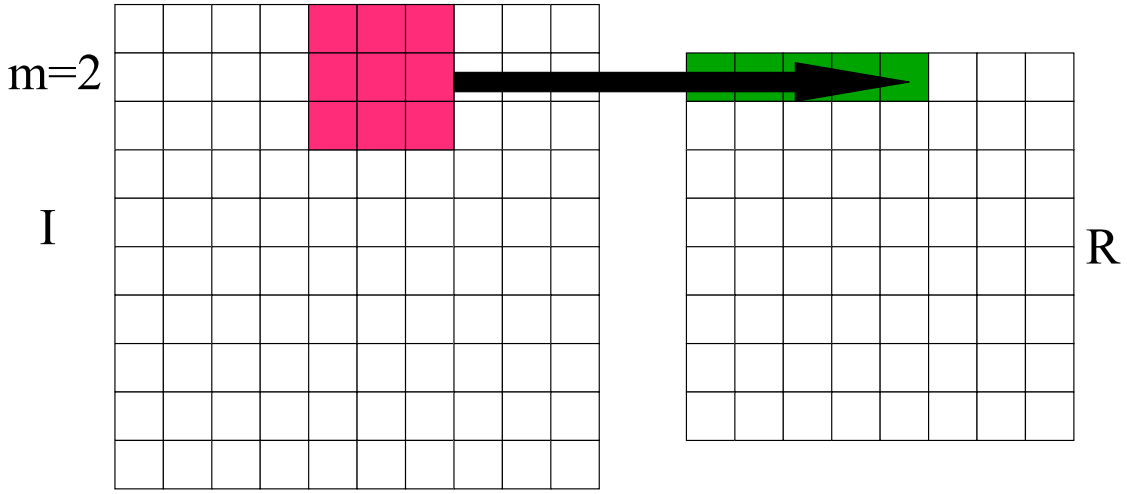
Kernel size
is m+1 by m+1

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

10/15/04

CS 461, Copyright G.D. Hager
borrowed from D. Kriegman

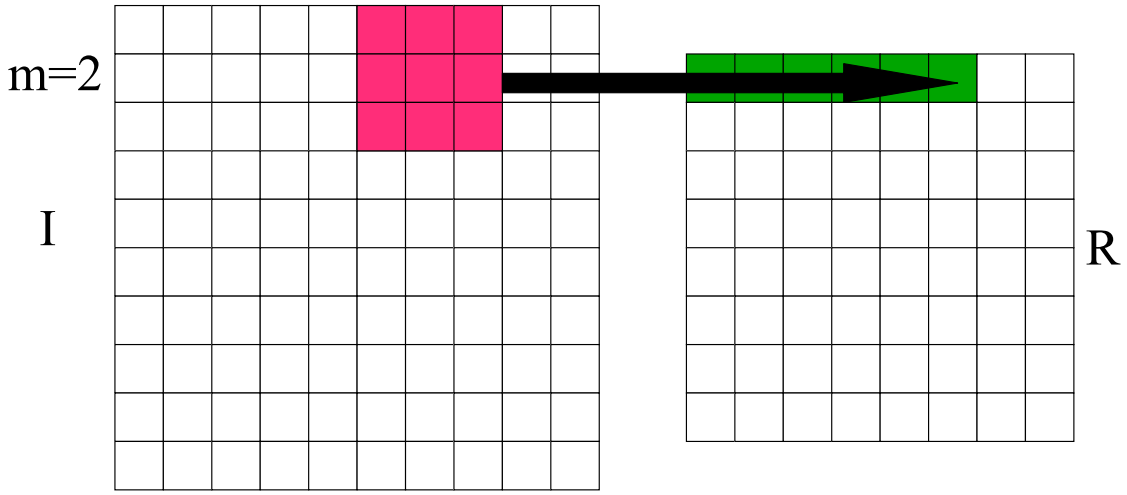
Convolution: R= K*I



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



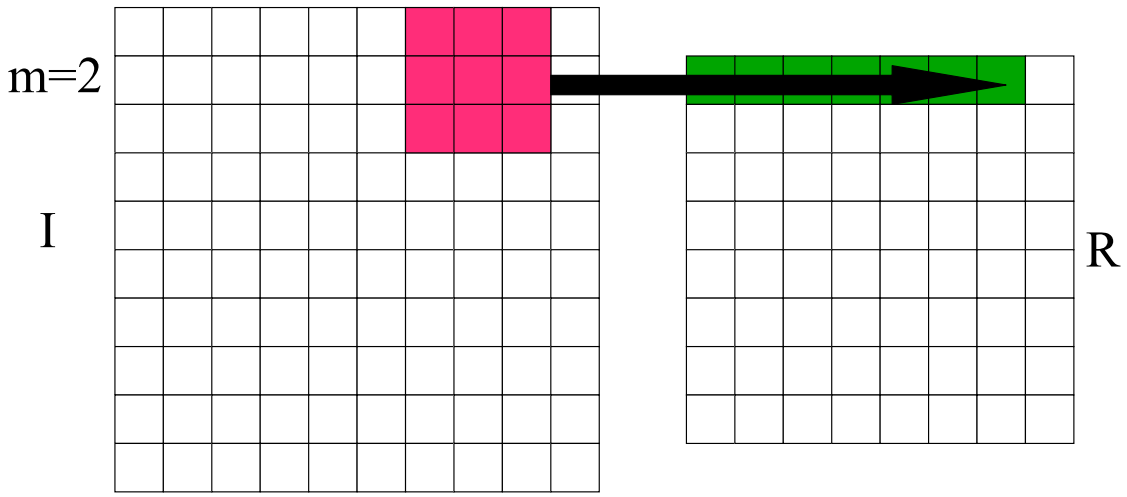
Kernel size
is m+1 by m+1

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

10/15/04

CS 461, Copyright G.D. Hager
borrowed from D. Kriegman

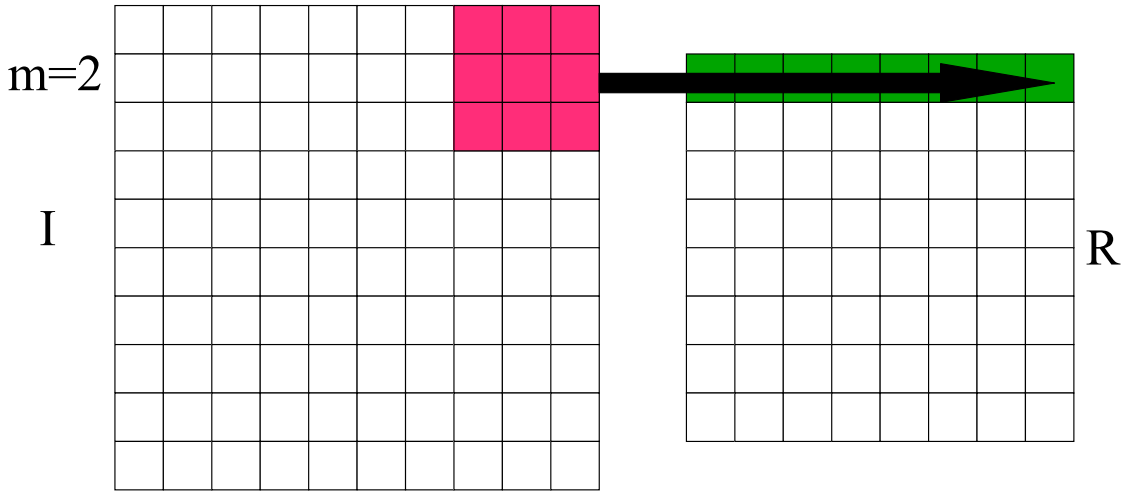
Convolution: R= K*I



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



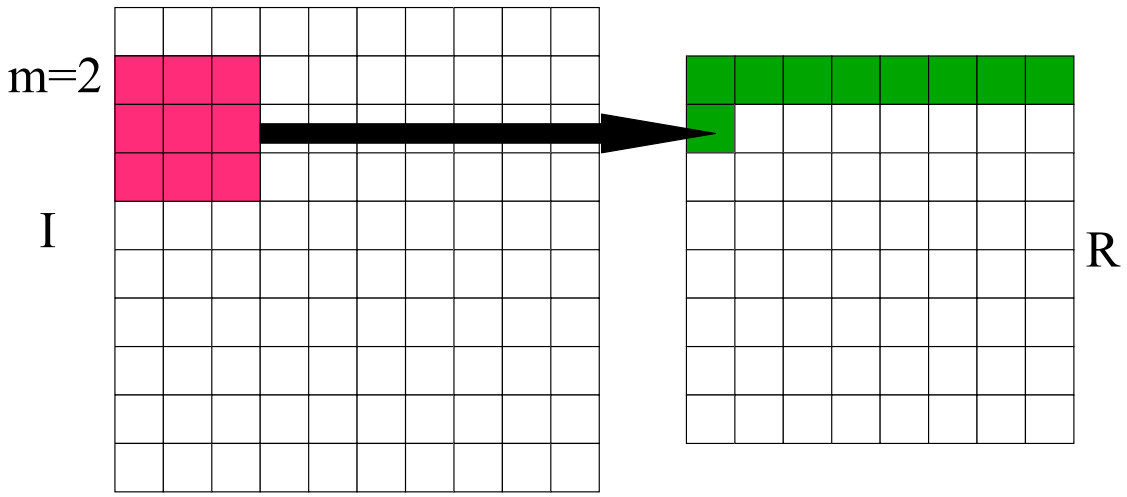
Kernel size
is m+1 by m+1

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

10/15/04

CS 461, Copyright G.D. Hager
borrowed from D. Kriegman

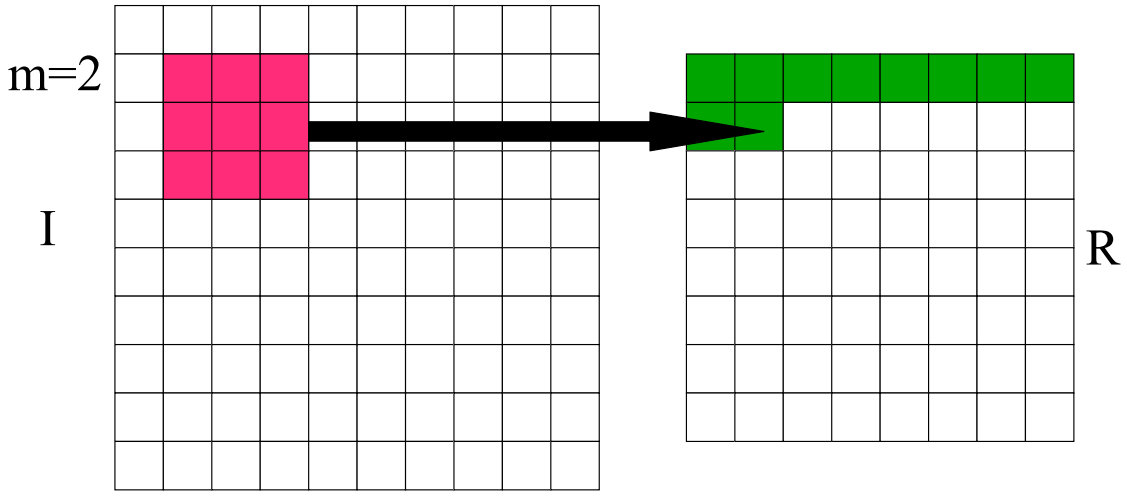
Convolution: R= K*I



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



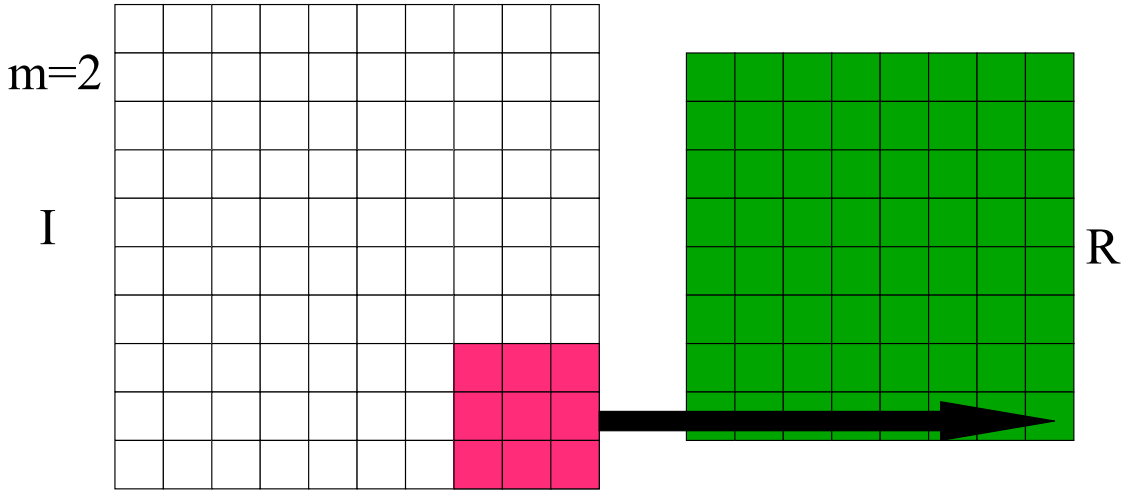
Kernel size
is m+1 by m+1

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

10/15/04

CS 461, Copyright G.D. Hager
borrowed from D. Kriegman

Convolution: R= K*I

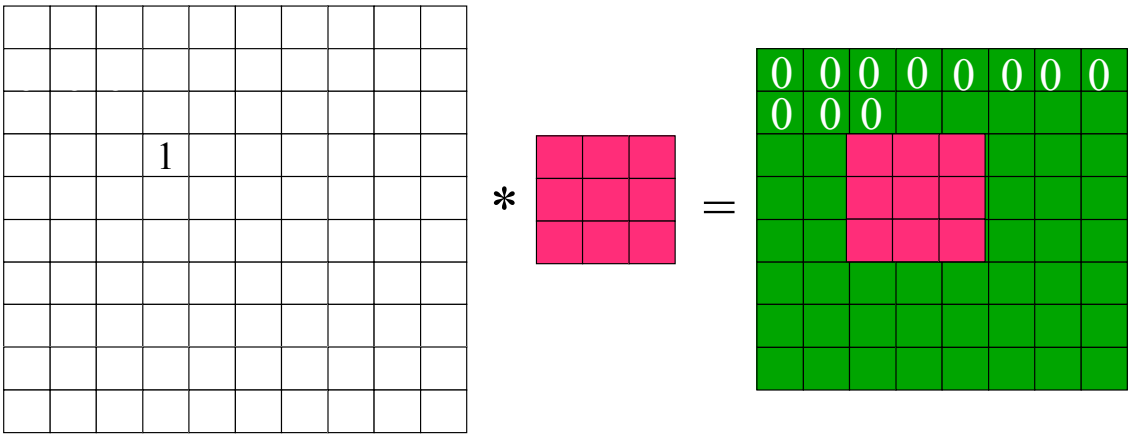


Kernel size
is m+1 by m+1

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

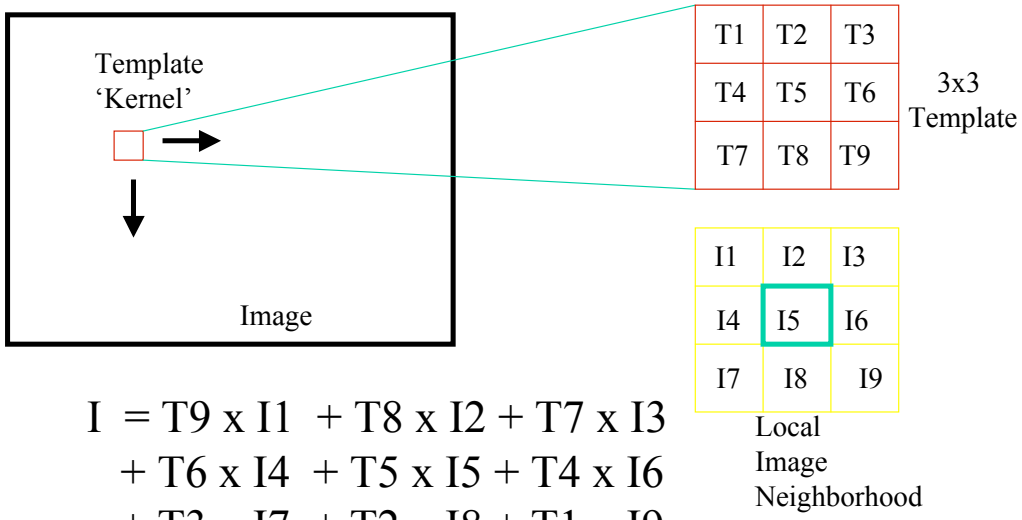
borrowed more D. Kriegman

Impulse Response



borrowed more D. Kriegman

DISCRETE CONVOLUTION



$$I = T9 \times I1 + T8 \times I2 + T7 \times I3 + T6 \times I4 + T5 \times I5 + T4 \times I6 + T3 \times I7 + T2 \times I8 + T1 \times I9$$

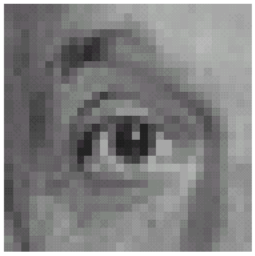
Convolution Formula

We often write I' = I*B to represent the convolution of I by B. B is referred to as the *kernel* of the convolution (or sometimes the "stencil" in the discrete case).

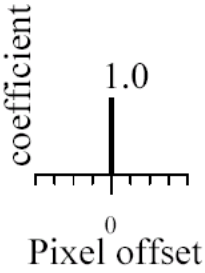
$$(I * B)(x,y) = \sum_i \sum_j I(x-i,jy-j) * B(i,j)$$

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

Linear filtering (warm-up slide)

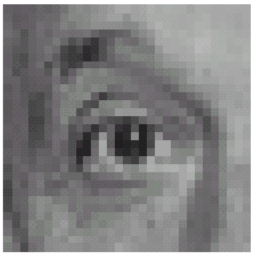


original

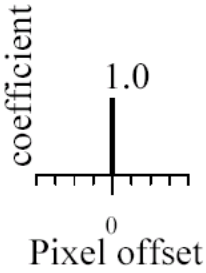


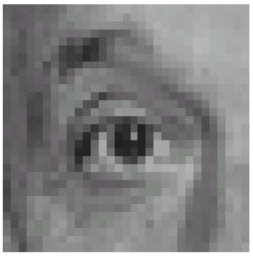
?

Linear filtering (warm-up slide)



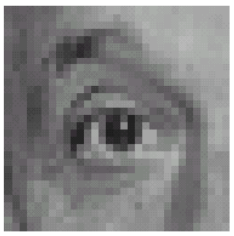
original



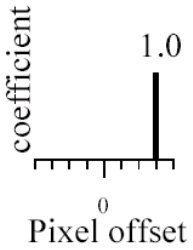


Filtered
(no change)

Linear filtering

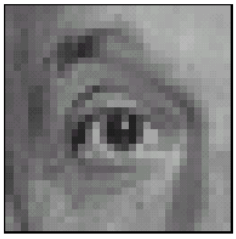


original

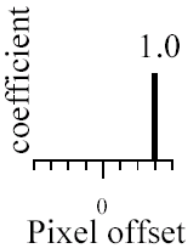


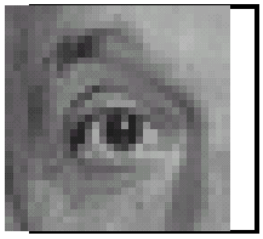
?

shift



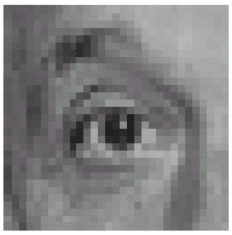
original



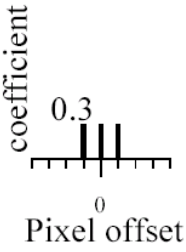


shifted

Linear filtering

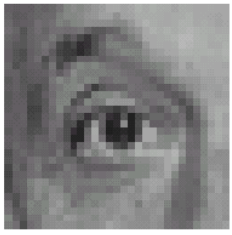


original

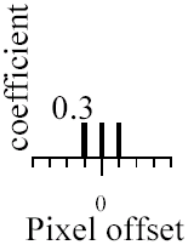


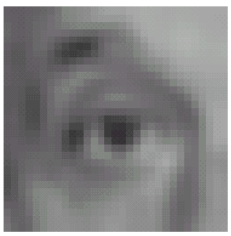
?

Blurring



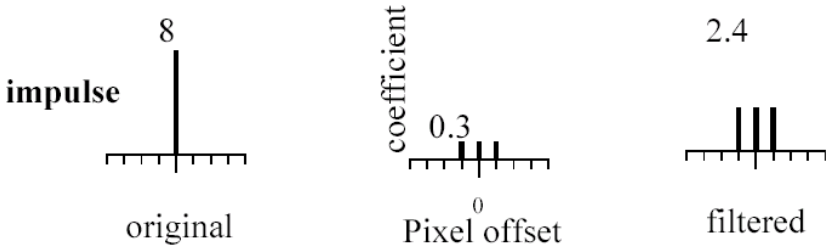
original





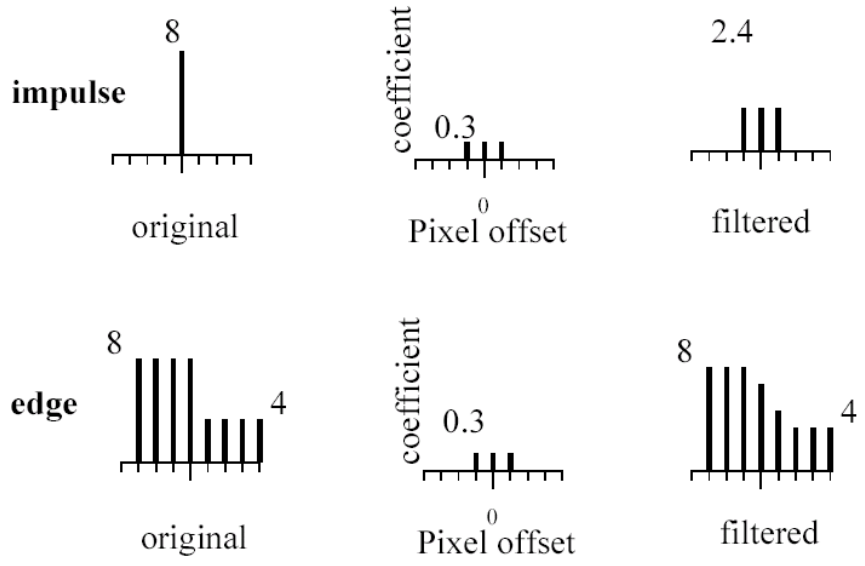
Blurred (filter applied in both dimensions).

Blur examples



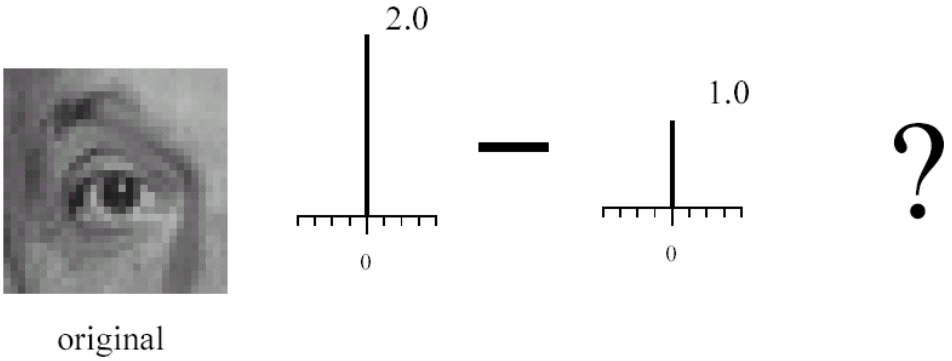
borrowed from D. Kriegman

Blur examples



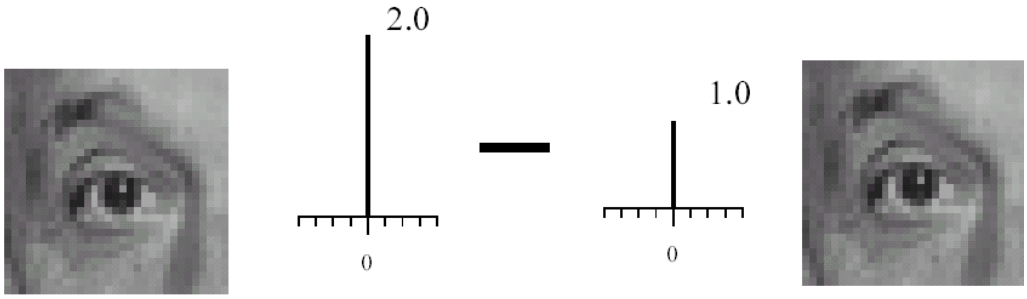
borrowed from D. Kriegman

Linear filtering (warm-up slide)



original

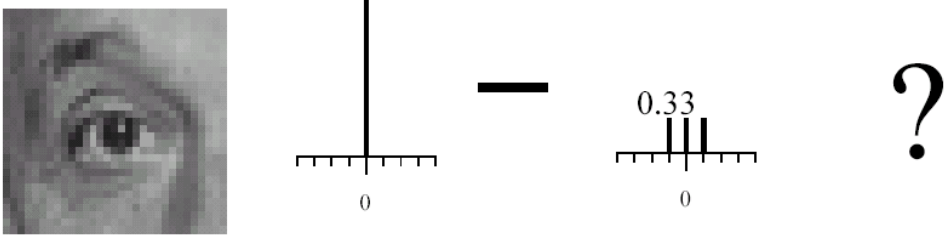
Linear filtering (no change)



original

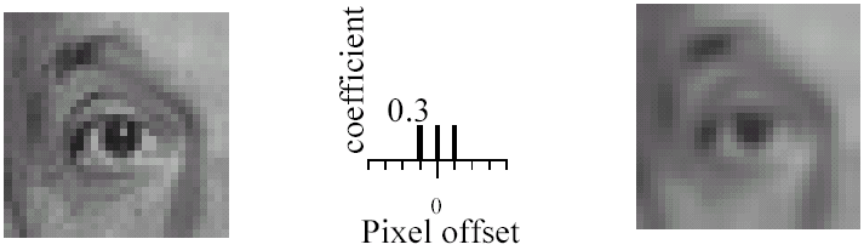
Filtered
(no change)

Linear filtering



original

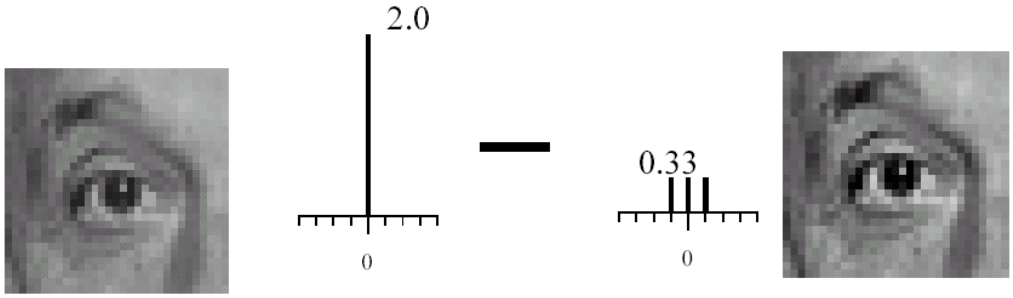
(remember blurring)



original

Blurred (filter applied in both dimensions).

Sharpening

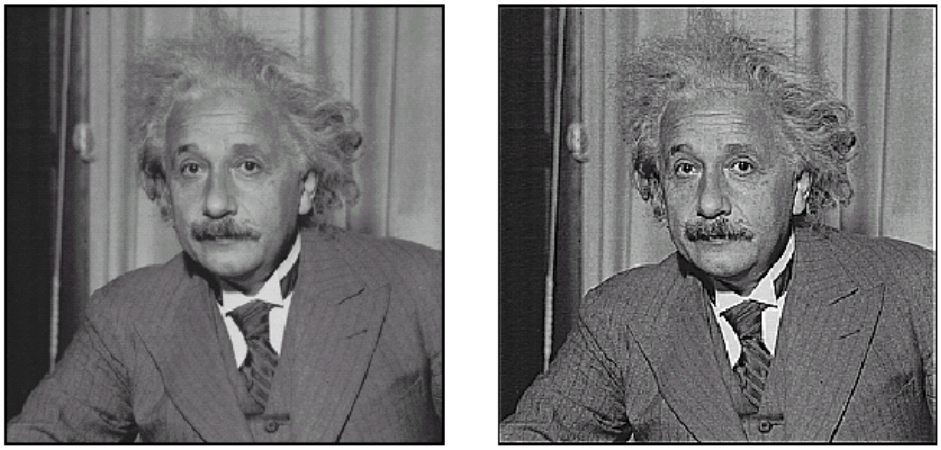


original

Sharpened original

borrowed from D. Kriegman

Sharpening



before **after**

borrowed from D. Kriegman

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'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.

Smoothing by Averaging

Kernel: 

Borrowed from D. Kriegman

Some Convolution Facts

- We often write I' = i*B to represent the convolution of I by B. B is referred to as the *kernel* of the convolution (or sometimes the "stencil" in the discrete case).
- Note convolution is
  - Associative
  - Commutative
  - A linear operator
- 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!

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
  - $F(u) = \int\int f(x)\, e^{-2\pi i u x}\, dx$
  - recall that $e^{-2\pi i u x} = \cos(2\pi u x) - i \sin(2 \pi u x)$
  - Also we have $f(x) = \int\int F(u)\, e^{-2\pi i u x}\, du$
  - $F(u) = |F(u)|\, e^{i \Phi(u)}$
    - a decomposition into magnitude and phase
  - $|F(u)|^2$ 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

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

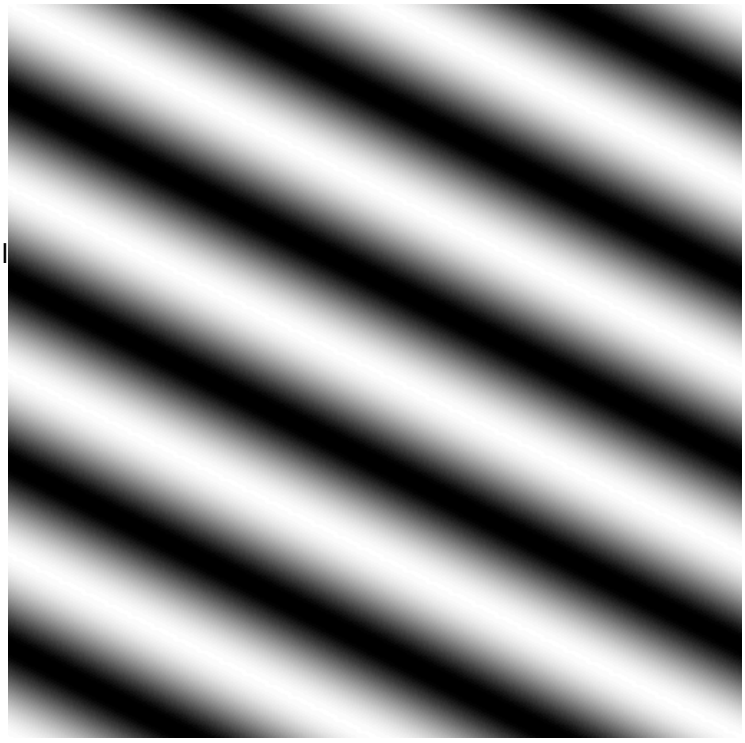
Implemented via the "Fast Fourier Transform" algorithm (FFT)

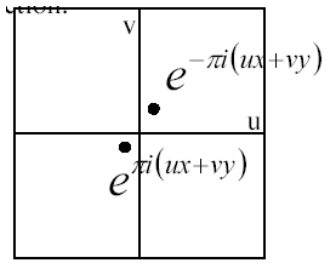
Fourier basis element

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

Transform is sum of orthogonal basis functions

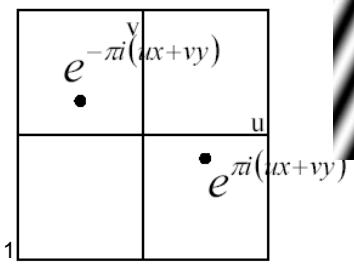
Vector (u,v)
- Magnitude gives frequency
- Direction gives orientation.





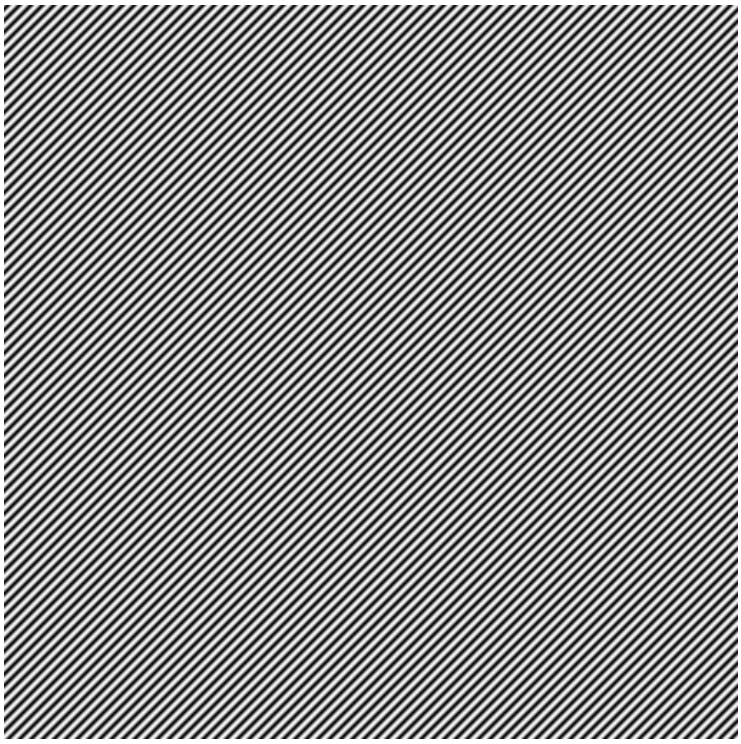
Borrowed from D. Kriegman

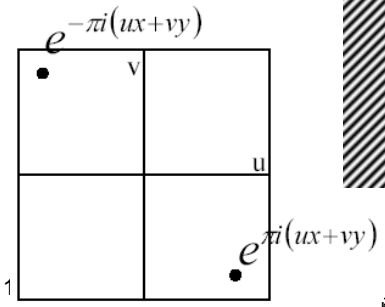
Here u and v are larger than in the previous slide.



borrowed from D. Kriegman

And larger still...

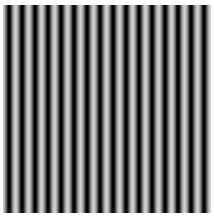


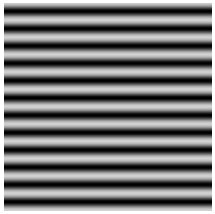


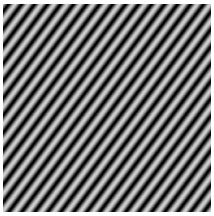
borrowed from D. Kriegman

The Fourier "Hammer"

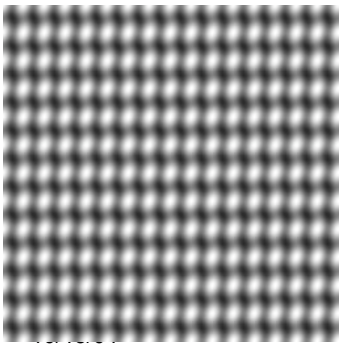
"Power Spectrum" →







Linear Combination:

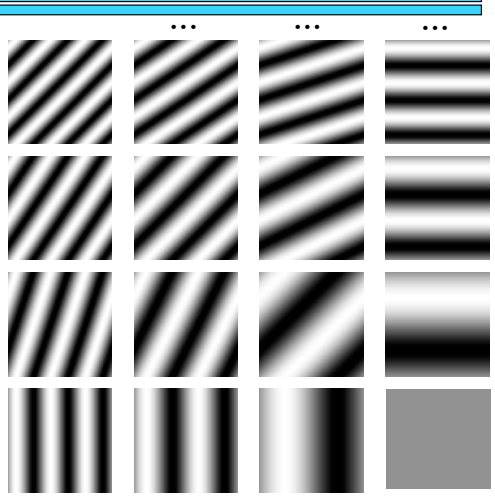


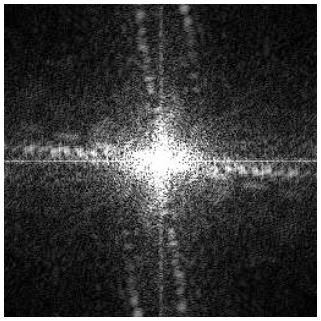
Basis vectors

Frequency Decomposition

All Basis Vectors →

← Example



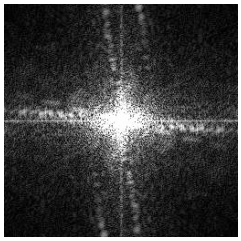
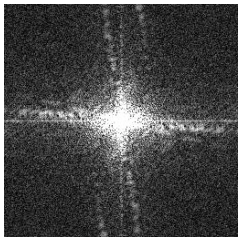
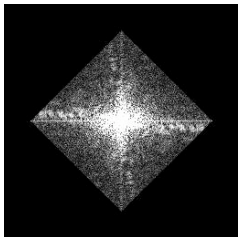


intensity ~ that frequency's coefficient

Using Fourier Representations



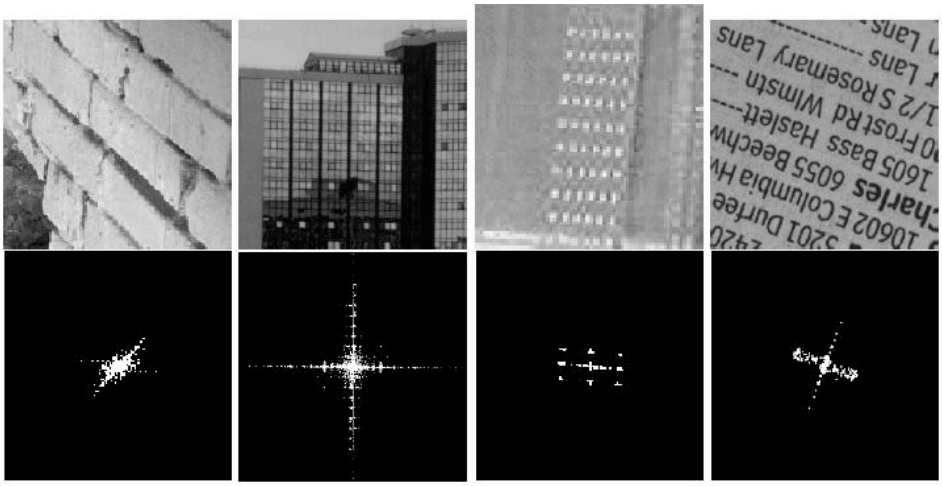
Smoothing

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

Using Fourier Representations

Dominant Orientation



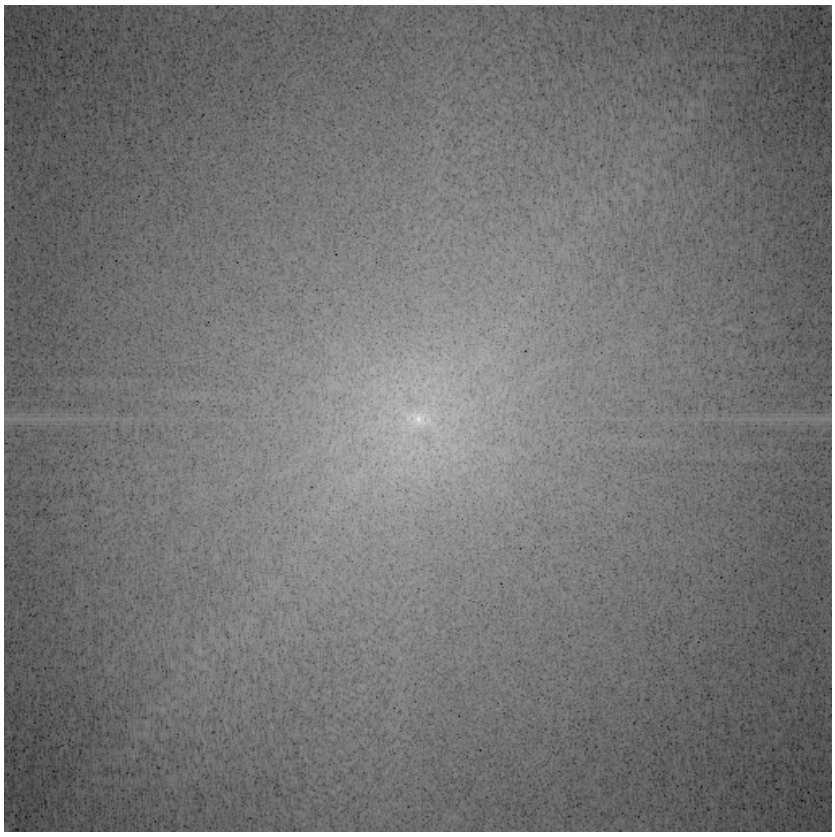
Limitations: not useful for local segmentation

Phase and Magnitude

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

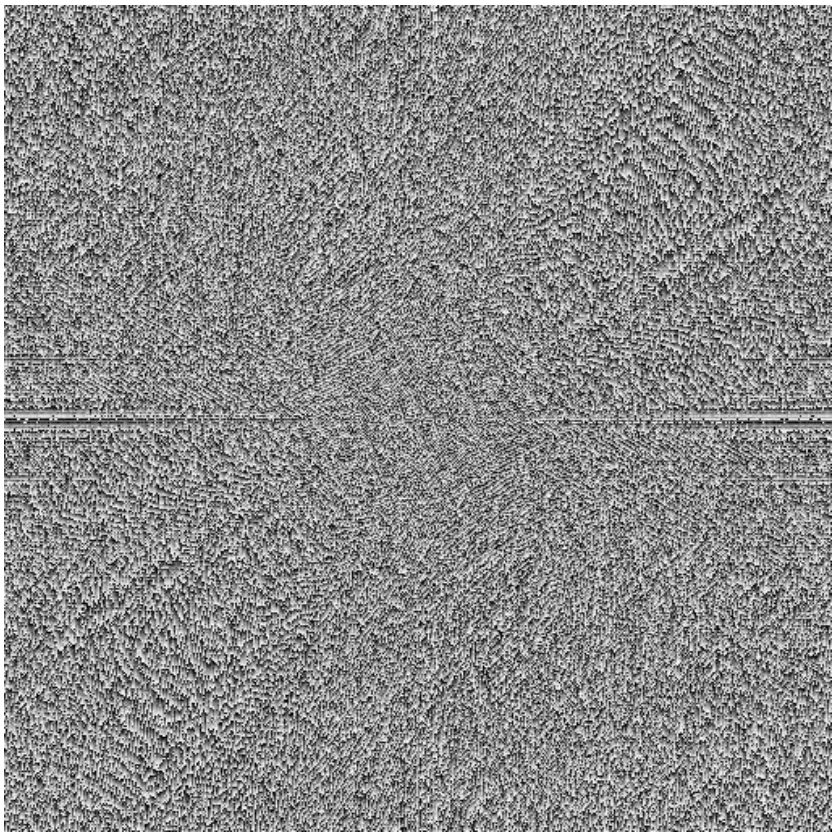
- Fourier transform of a real function is complex with real (R) and imaginary (I) components
  - difficult to plot, visualize
  - instead, we can think of the phase and magnitude of the transform
- Phase is the phase of the complex transform
  - $p(u) = \text{atan}(I(u)/R(u))$
- Magnitude is the magnitude 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?

This is the magnitude transform of the cheetah pic



borrowed from D. Kriegman

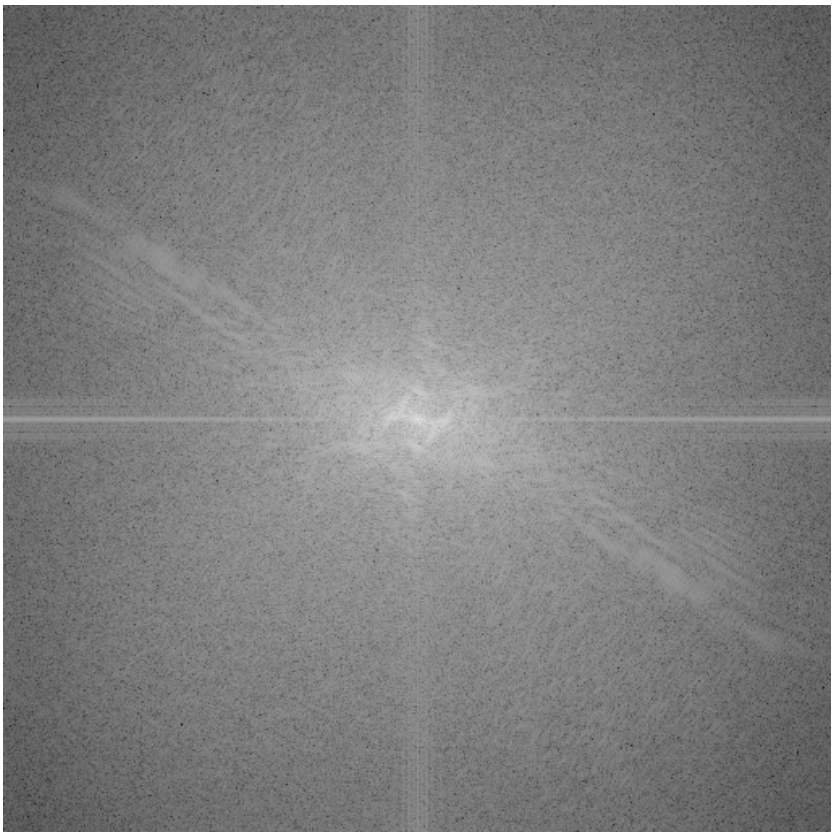
This is the phase transform of the cheetah pic



borrowed from D. Kriegman

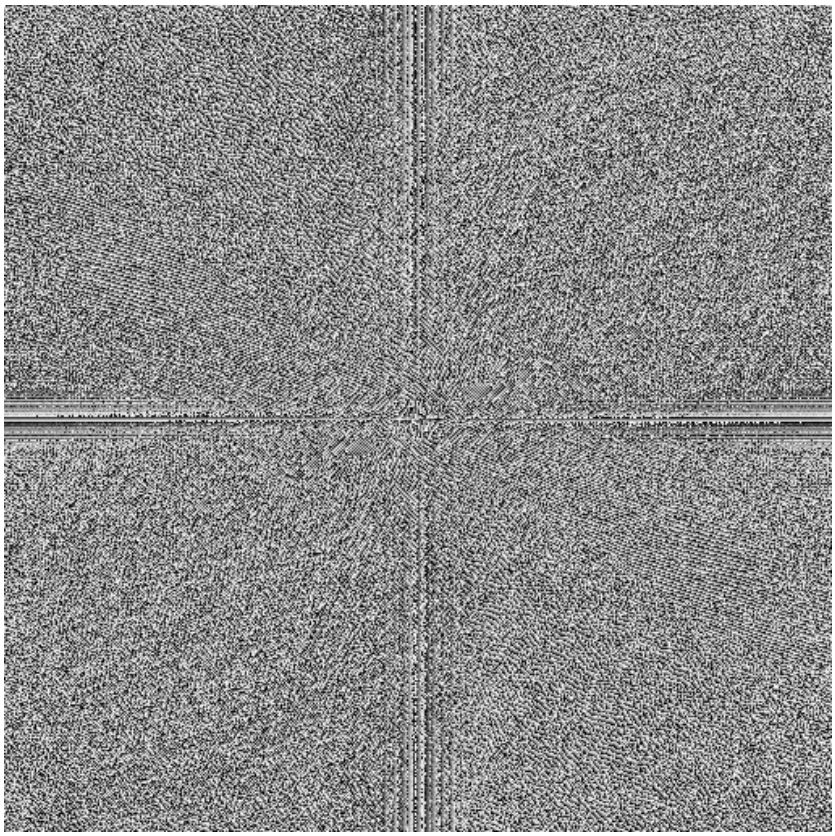
borrowed from D. Kriegman

This is the magnitude transform of the zebra pic



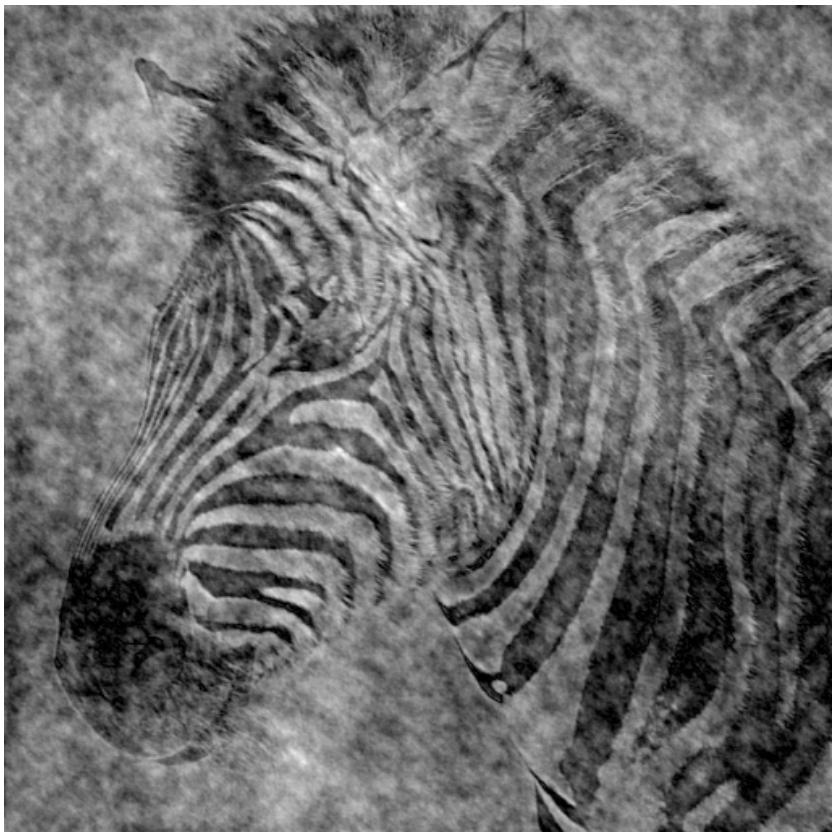
borrowed from D. Kriegman

This is the phase transform of the zebra pic



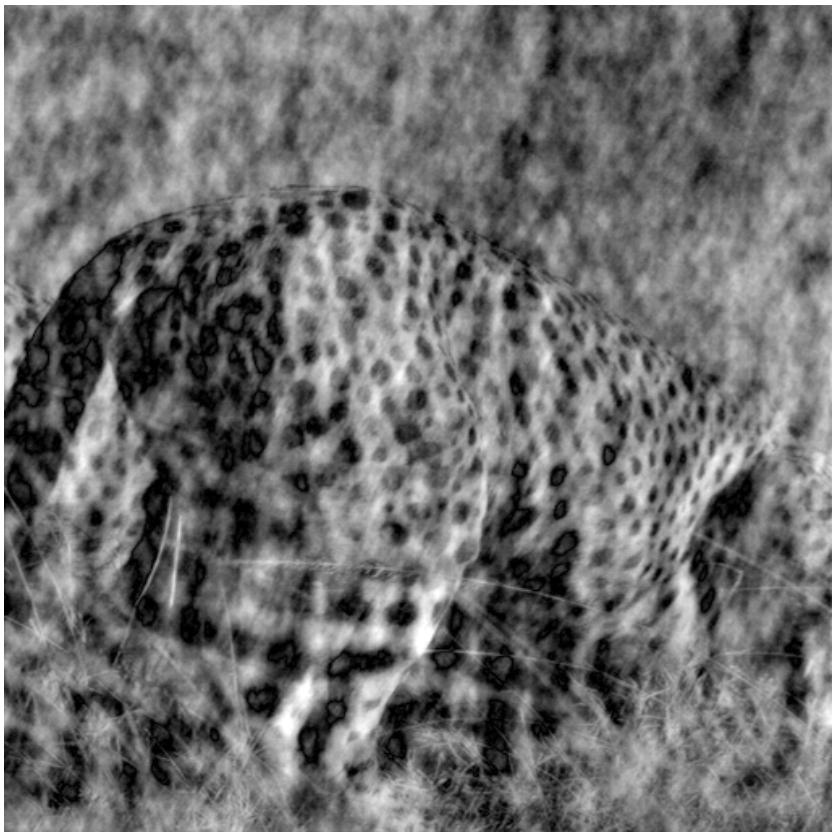
borrowed from D. Kriegman

Reconstruction with zebra phase, cheetah magnitude



borrowed from D. Kriegman

Reconstruction with cheetah phase, zebra magnitude



Borrowed from D. Kriegman

The Fourier Transform and Convolution

- If H and G are images, and F(.) represents Fourier transform, then

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

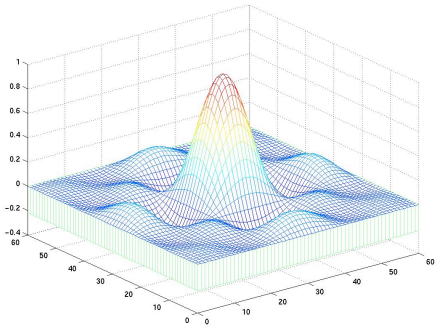
- Thus, one way of thinking about the properties of a convolution is by thinking of how it modifies the frequencies of the image to which it is applied.

- 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(mean filter) =



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

What a Box Filter Does

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

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

- It is not hard to show that the FT of a Gaussian is again a Gaussian. Hence, it operates as a low pass filter.

- 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 **σ.** 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.

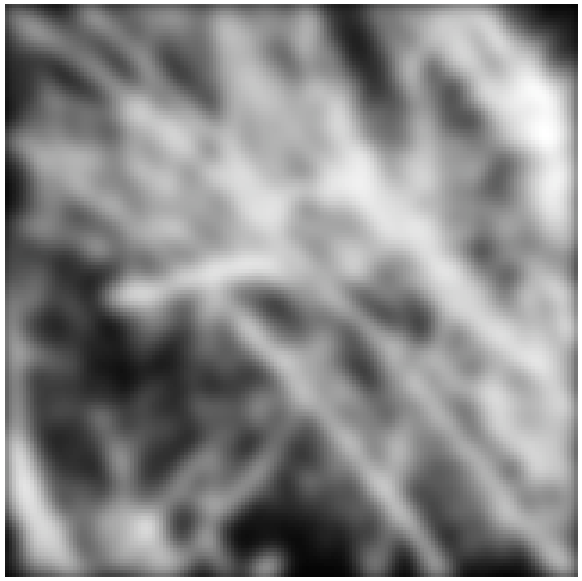
Smoothing by Averaging

Kernel: ■

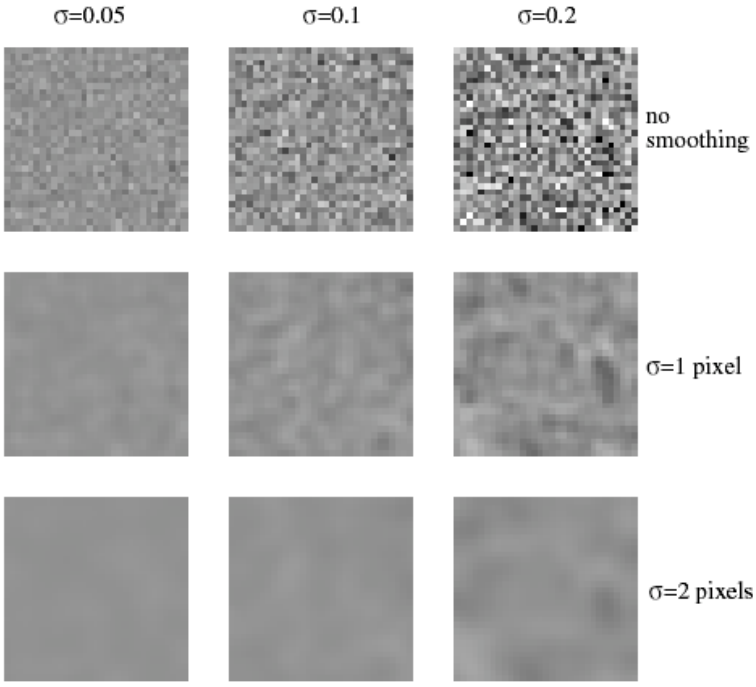
Borrowed from D. Kriegman

Smoothing with a Gaussian

Kernel: 



borrowed from D. Kriegman



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

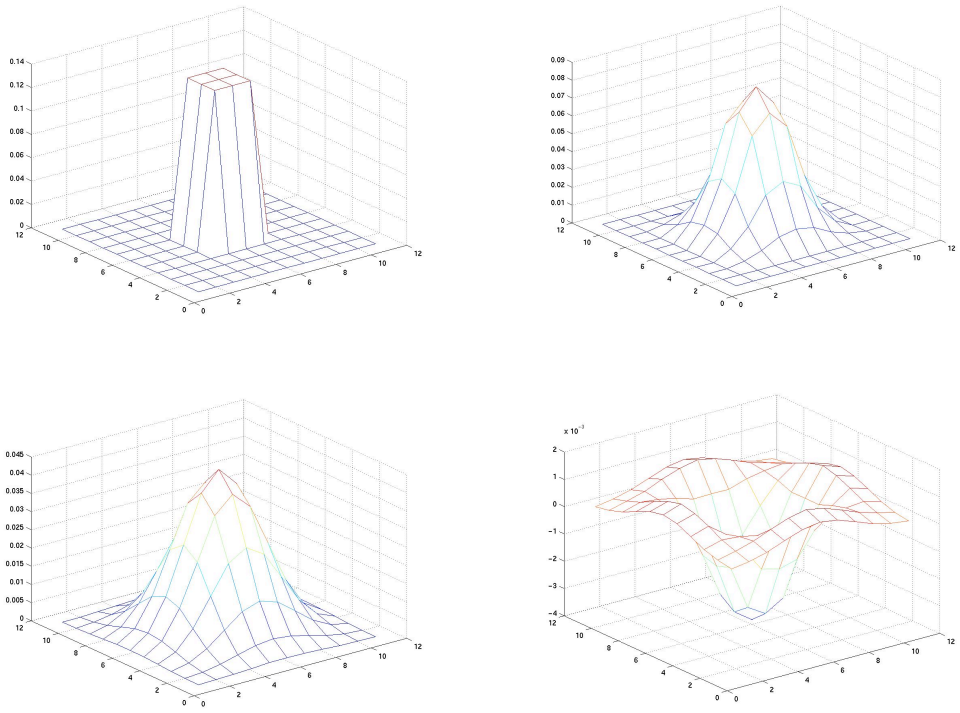
borrowed from D. Kriegman

Computational Issues: Separability

- Recall that convolution is associative. Suppose I use the templates gx = exp(-i^2/2 σ^2) and By = gy = exp(-j^2/2 σ^2) Then
  - gx * (gy * I) = (gx * gy) * 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

- Impulsive noise
  - randomly pick a pixel and randomly set to a value
  - saturated version is called salt and pepper noise
- Quantization effects
  - Often called noise although it is not statistical
- Unanticipated image structures
  - Also often called noise although it is a real repeatable signal.

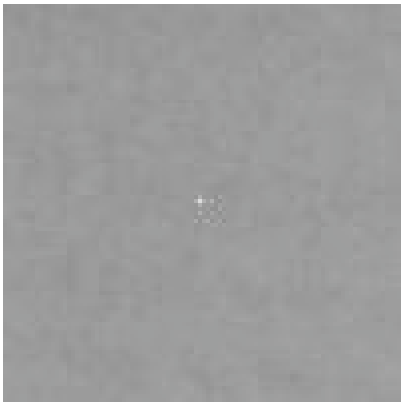
Digitization Effects

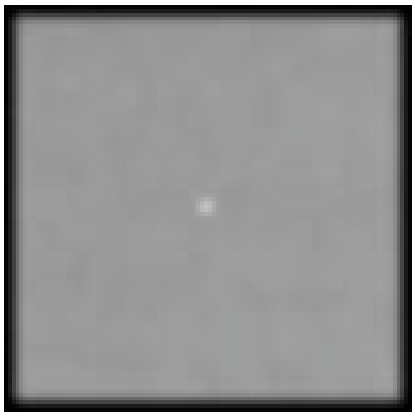
- The “diameter” d of a pixel determines the highest frequency representable in an image

$$l = 1/2d$$

- Real scenes may contain higher frequencies resulting in aliasing of the signal.
- In practice, this effect is often dominated by other digitization artifacts.
- One problem in particular is differing sampling rates between digitizer and camera readout of a row.

Limitations of Linear Operators



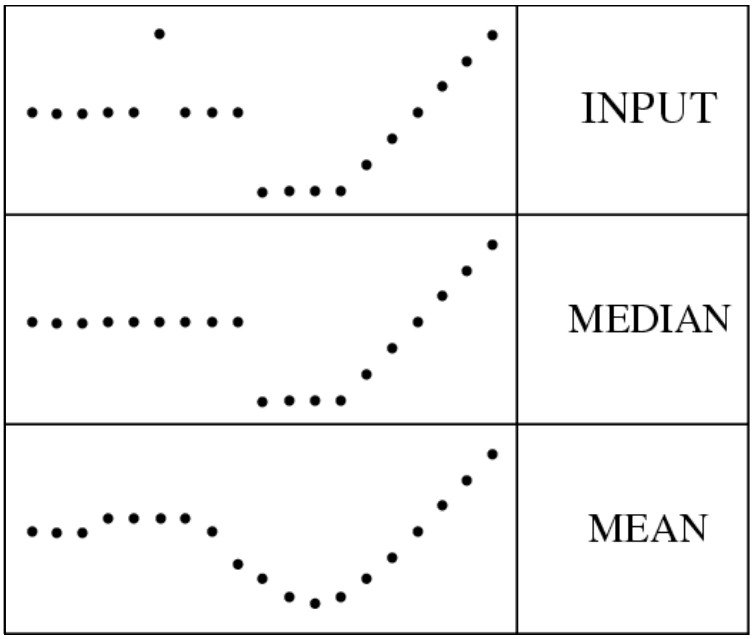


Nonlinear Filtering: The Median Filter

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 :

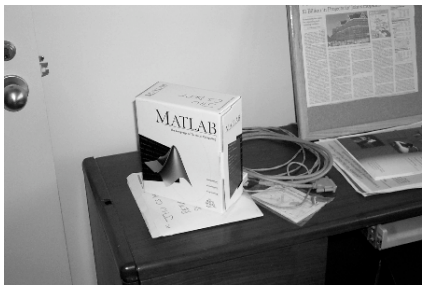


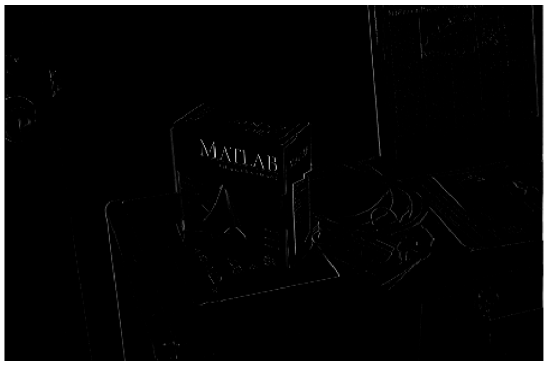
Yet Another View of Convolution

- Suppose we consider the convolution template as a “vector”:
  - T = [T1,T2,T3 .... Tn]
- Likewise, consider a region of the image to which the convolution is applied as a vector
  - I = [I1,I2,...In]
- Then the value of the convolution at a point is just the “dot product”
  - v = 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)
- Why does this make sense when thinking of Fourier transforms?

What Else Can You Do With Convolution?

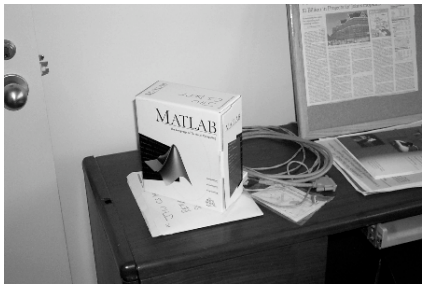
- Thus far, we’ve only considered convolution kernels that are smoothing filters.
- Consider the following kernel:
  - [1, -1]

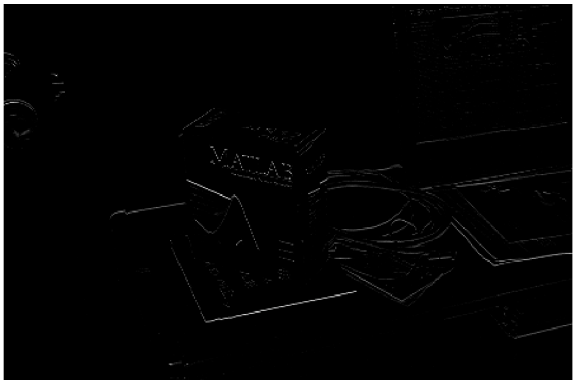




What Else Can You Do With Convolution?

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

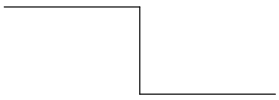




Physical causes of edges

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

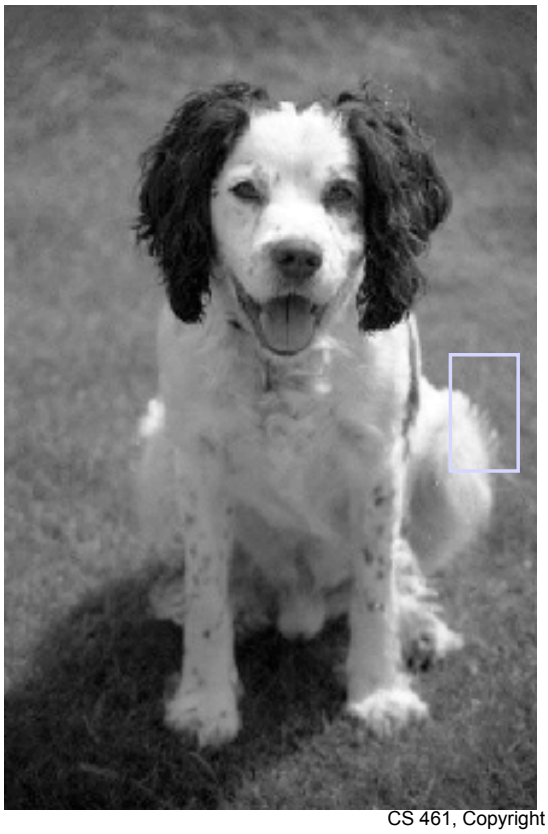
Edge Types



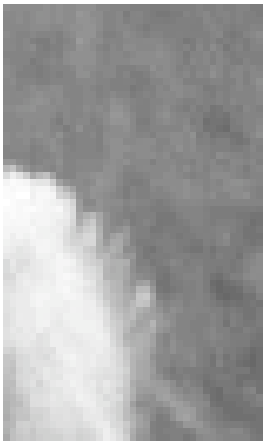
Step

Ridge

Roof

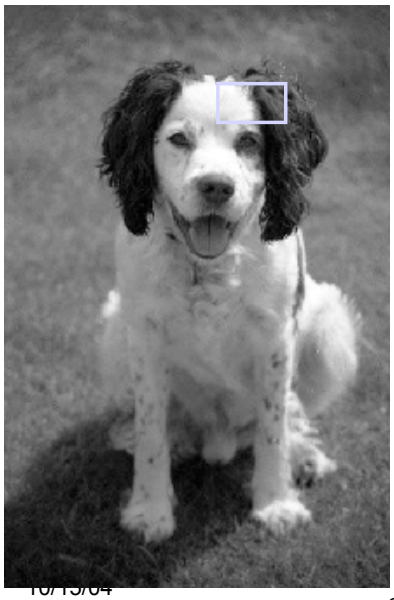


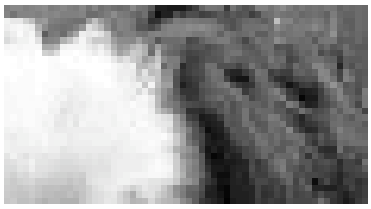
Object Boundaries



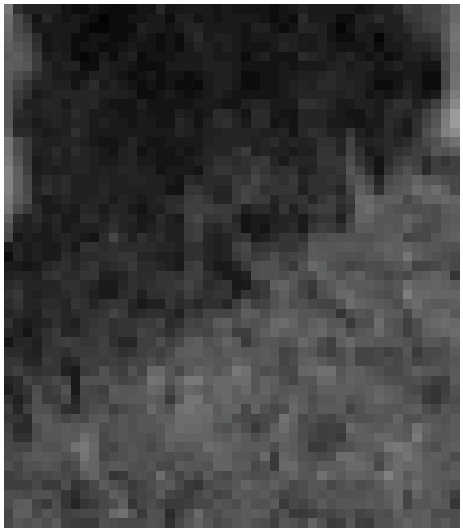
Surface normal discontinuities

Boundaries of material properties





Boundaries of lighting

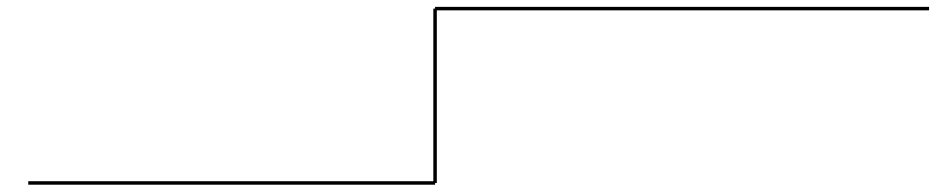


The Image Gradient

- 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 = n ¢ r 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

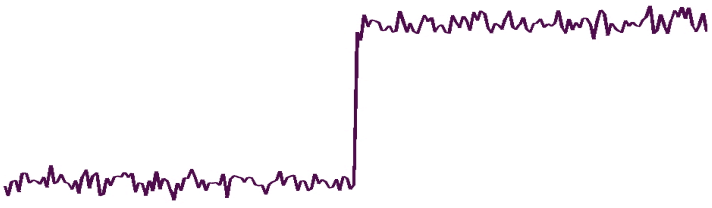
Edge is Where Change Occurs 1-D

- 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.



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 Kernals

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

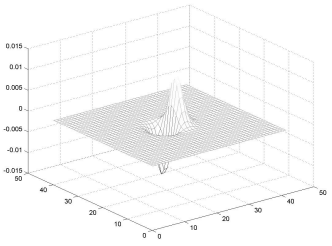
A good exercise: derive the Laplacian from 1-D derivative filters.

Note the Laplacian is rotationally symmetric!

Smoothing Plus Derivatives

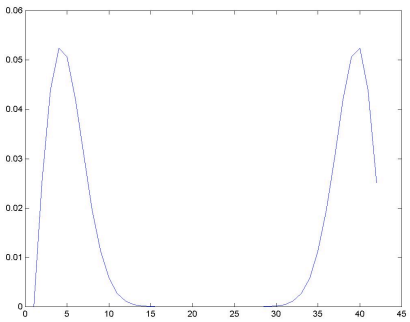
- One problem with differences is that they by definition reduce the signal to noise ratio (can you show this?)
- Recall smoothing operators (the Gaussian!) reduce noise.
- Hence, an obvious way of getting clean images with derivatives is to combine derivative filtering and smoothing: e.g.
  - $G * D_x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

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
  - $D_u[G*I] = D_u[G]*I = [u_1G_x + u_2G_y]*I = u_1G_x*I + u_2G_x*I$
- The two components G_x and 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 ....)

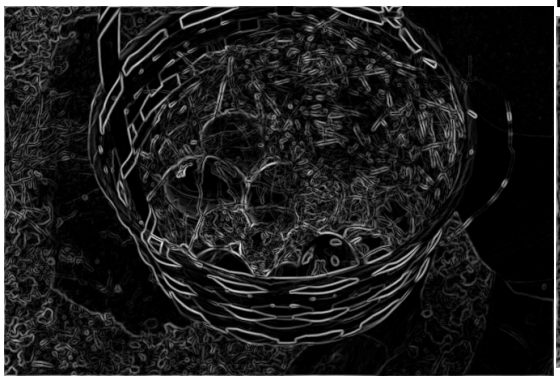
Algorithm: Simple Edge Detection

- 1. Compute $I_g = G(\sigma) *G(\sigma)' * I$
- 2. Compute $I_x = [-1,1;-1,1] * I_g$
- 3. Compute $I_y = [-1,1;-1,1]' * I_g$
- 4. Compute $I_{mag} = sqrt(I_x.* I_x + I_y .* I_y)$
- 5. 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.

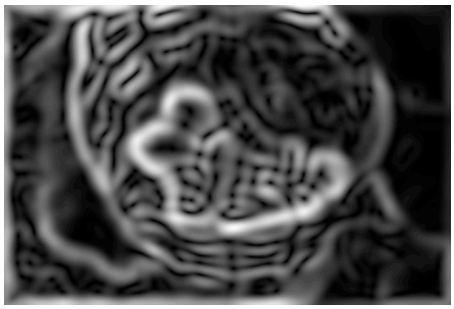
Example

sigma = 1



sigma = 2

sigma = 5



sigma = 10

Types of Edge Operators

1. Operators approximating derivatives using differences.
   - directional: Roberts, Prewitt, DoG, etc.
   - Rotationally invariant: Laplacian (sum of second derivatives)

2. Operators based on the zero crossing of the second derivative (e.g. Canny).

3. Operators that attempt to match a specific image profile.

Filter Pyramids

- An Exercise:
  - Suppose I have $G(\sigma)$ and I perform $G(\sigma) * G(\sigma) * I$
    - Hint: think about the convolution theorem and the FFT
- Suppose I want to subsample images
  - subsampling reduces the highest frequencies
  - averaging reduces noise
  - Can I average and resample and reduce noise while not loosing desirable frequencies?

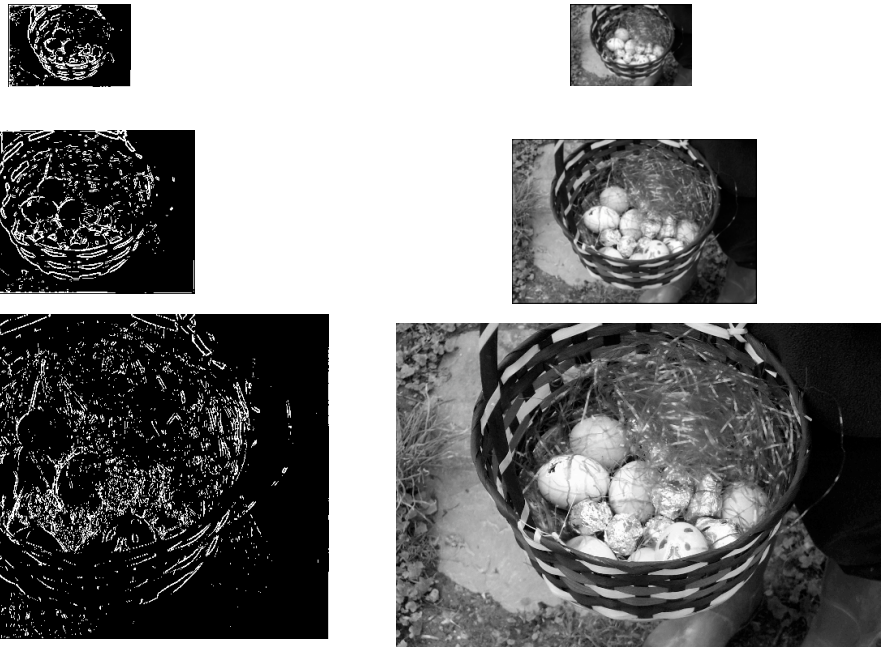
Gaussian Pyramid

- Algorithm:
  - 1. Filter with $G(\sigma)$
  - 2. Resample at every other pixel
  - 3. Repeat
- A common use of this is the Laplacian Pyramid

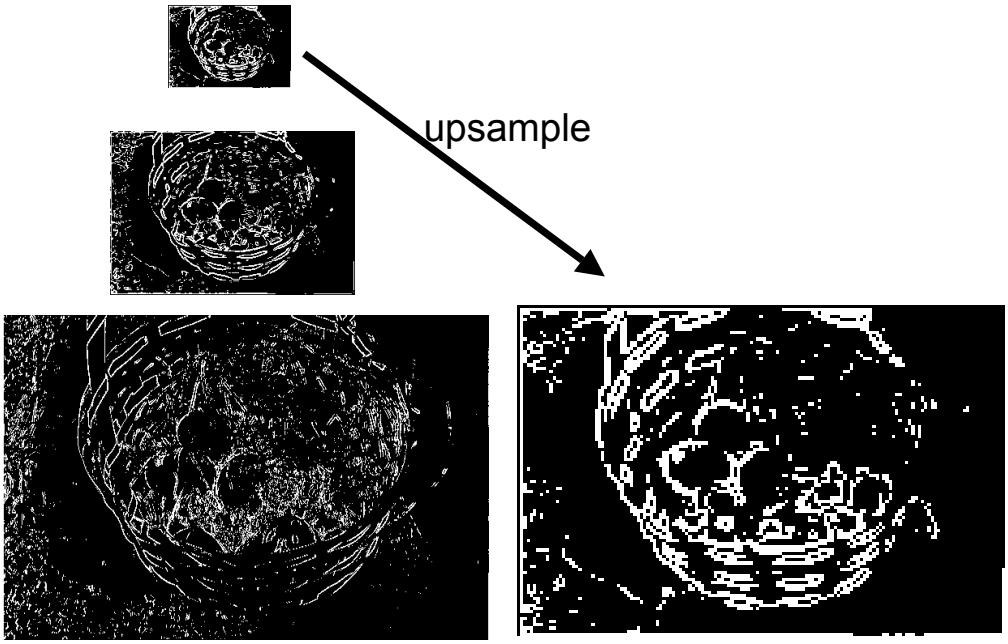
Laplacian Pyramid Algorithm

- 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) - upsample(G(n))

Laplacian of Gaussian Pyramid



Laplacian of Gaussian Pyramid



From Pixels to Edges

- Various operators can be used to *enhance* rapid contrast changes
- Detecting these contrast changes involves *thresholding* to separate noise from signal
- *Edges* are a result of *grouping* pixels (sometimes called “edgels”) into groups forming continuous curves.

Definitions:

Edge normal: Unit vector in direction of maximum intensity variation
Edge direction: Perpendicular to edge normal
Edge position: Image position of pixels of edge
Edge strength: Change in contrast along normal

From Pixels to Edges

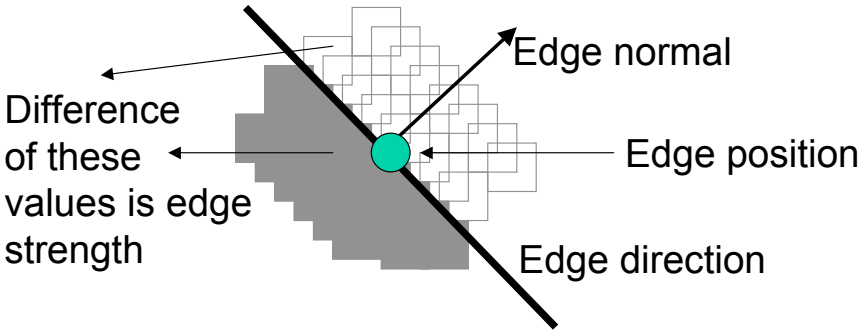
Definitions:

Edge normal: Unit vector in direction of maximum intensity variation

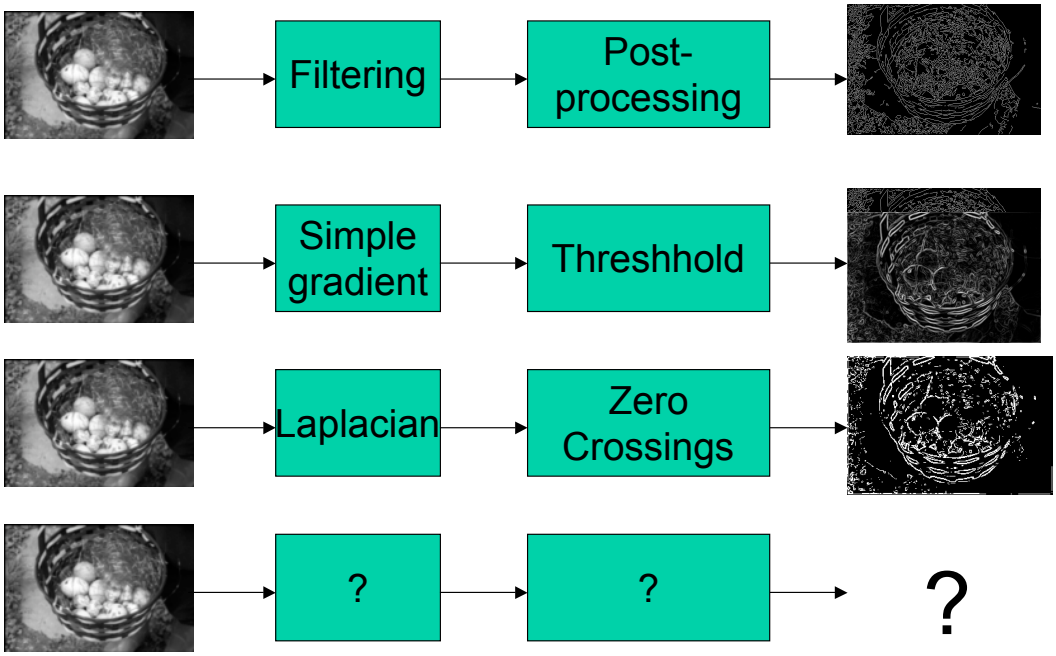
Edge direction: Perpendicular to edge normal

Edge position: Image position of pixels of edge

Edge strength: Change in contrast along normal



Edges: The Problem



What is optimal?

Canny Edge Detector

- The Plan:
  - Formulate an optimization problem for detection on 1-D signals
  - Generalize to 2D signals
  - Apply thresholding with hysteresis
  - Apply this operator at various scales

- The Assumptions:
  - Edge enhancement is linear
  - The edge model is step edges with amplitude A
  - Noise is additive, white and Gaussian

Optimization Criteria

- **Good Detection:** ***Minimize the probability of false positives and false negatives***
  - **Maximize the SNR**

$$\frac{A}{n_0}\Sigma(f) = \left| \frac{A\int_{-W}^{0} f(t)dt}{n_0\sqrt{\int_{-W}^{W} f^2(t)dt}} \right|$$

- **Good Localization:** ***Edgels detected should lie as close as possible to the true edge***
  - **Maximize 1/distance to edge center which leads to maximizing LOC**

$$\frac{A}{n_0}\Lambda(f') = \left| \frac{A\,|\,f'(0)\,|}{n_0\sqrt{\int_{-W}^{W} f'^2(t)dt}} \right|$$

Optimization Cont'd

- Consider the maximizing the product of both criteria
  - result is itself a step filter
  - step filters are noise amplifying!

- Additional criterion: single response constraint:
  - detector should minimize the number of local maxima about an edge (recall what happens with step filter)
  - ***RESULT1: localization vs. detection***

$$\Sigma(f_w) = \sqrt{w}\Sigma(f) \;\; and \;\; \Lambda(f'_w) = \frac{1}{\sqrt{w}}\Lambda(f') \;\; where \;\; f_w(x) = f(x/w)$$

  - ***RESULT2: optimal detector is very close to the first derivative of a Gaussian.***

The Procedure

- Enhancement:
  - compute x and y derivatives using DoG's.
  - compute direction and magnitude of gradient (two images)

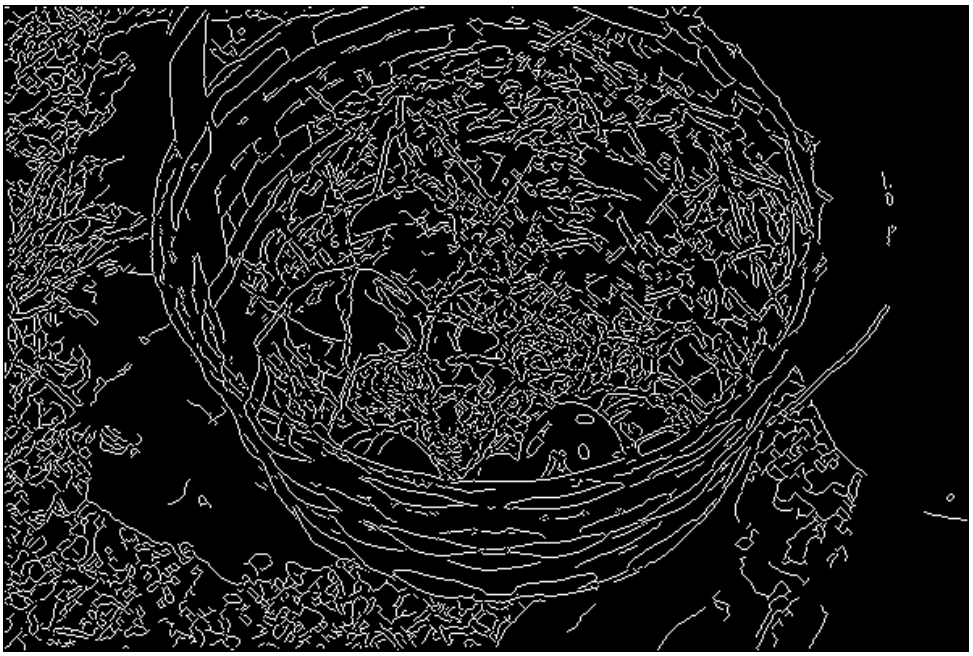
- Nonmaximal Suppression:
  - Sample along the gradient direction
  - If given pixel is smaller than neighbor, set it to zero

- Hysteresis Thresholding:
  - Starting from upper left, visit pixels until one exceeds t_{upper}
  - Follow chains of maxima in edge direction until value drops below t_{lower}
  - Mark and save all visited values as a connected contour

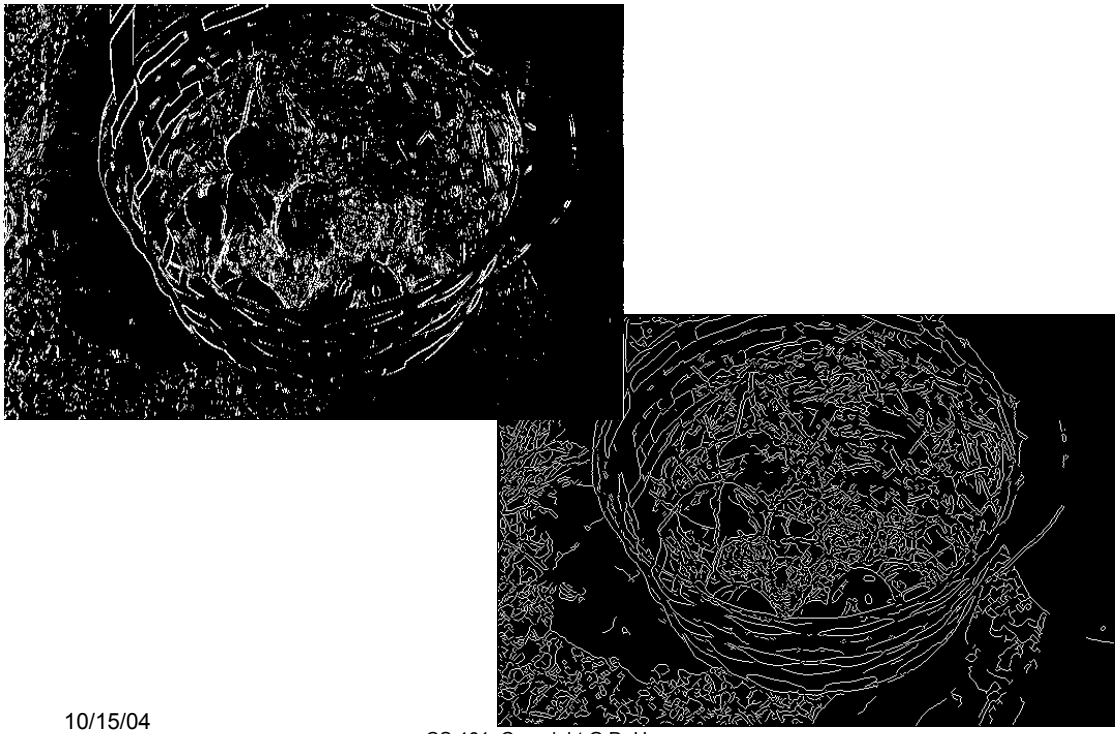
Hysteresis

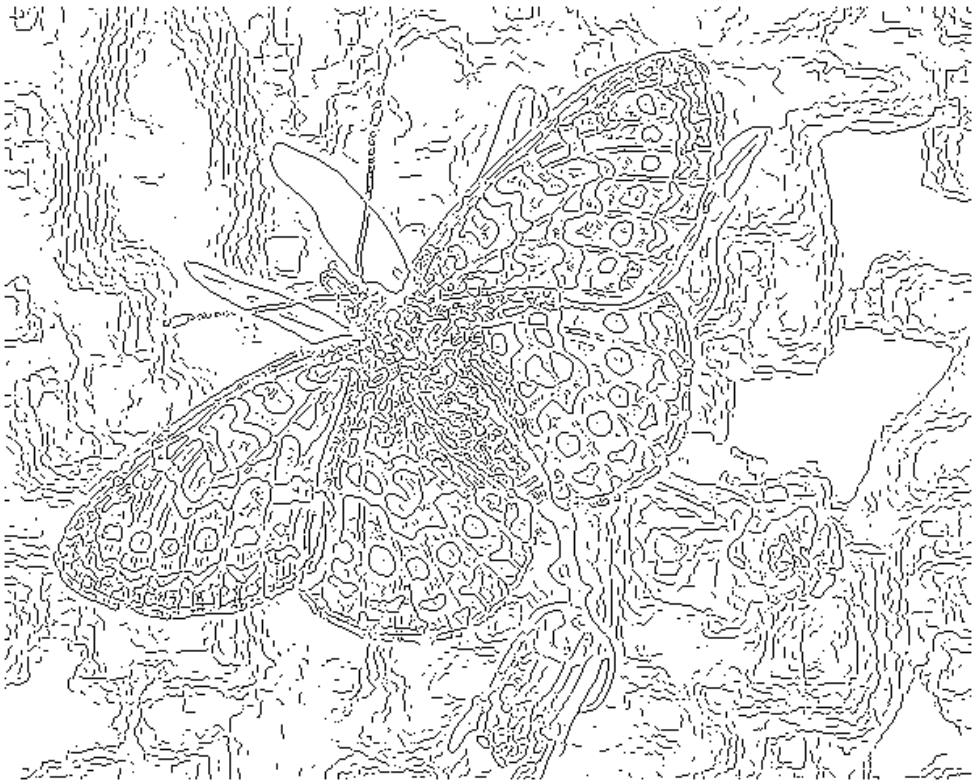
- Track edge points by starting at point where gradient magnitude > τ_{high}.
- Follow edge in direction orthogonal to gradient.
- Stop when gradient magnitude < τ_{low}.
    - i.e., use a high threshold to start edge curves and a low threshold to continue them.

Canny Output



Canny Comparison





fine scale high threshold

coarse scale, high high threshold

coarse
scale
Low high
threshold

Why is Canny so Dominant

- Still widely used after 20 years.
  1. Theory is nice (but end result same,).
  2. Details good (magnitude of gradient, non-max supression).
  3. Hysteresis an important heuristic.
  4. Code was distributed.

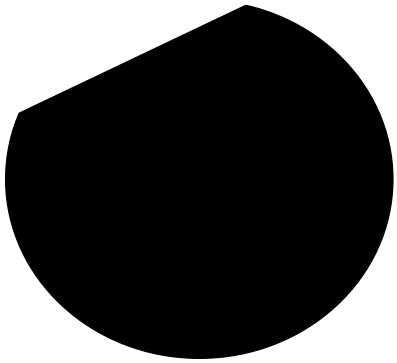
SubPixel Precision

- It is often hard to exactly localize the maximum of a function
- Many algorithms need sub-pixel precision
- Thus, it is common to apply a *second derivative* operator locally to locate the edge
  - note we know the edge direction, so we can compute second directional derivatives!

DO IT IN MATLAB GREG!

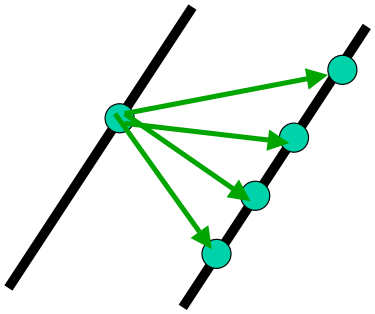
Corners

- Why are they important?



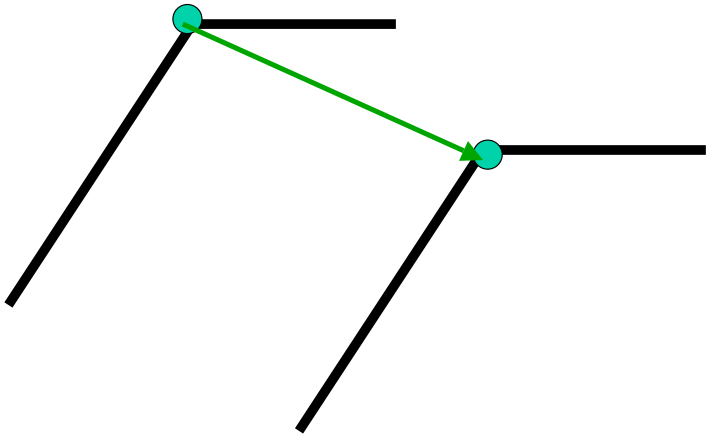
Corners contain more info than lines.

- A point on a line is hard to match.

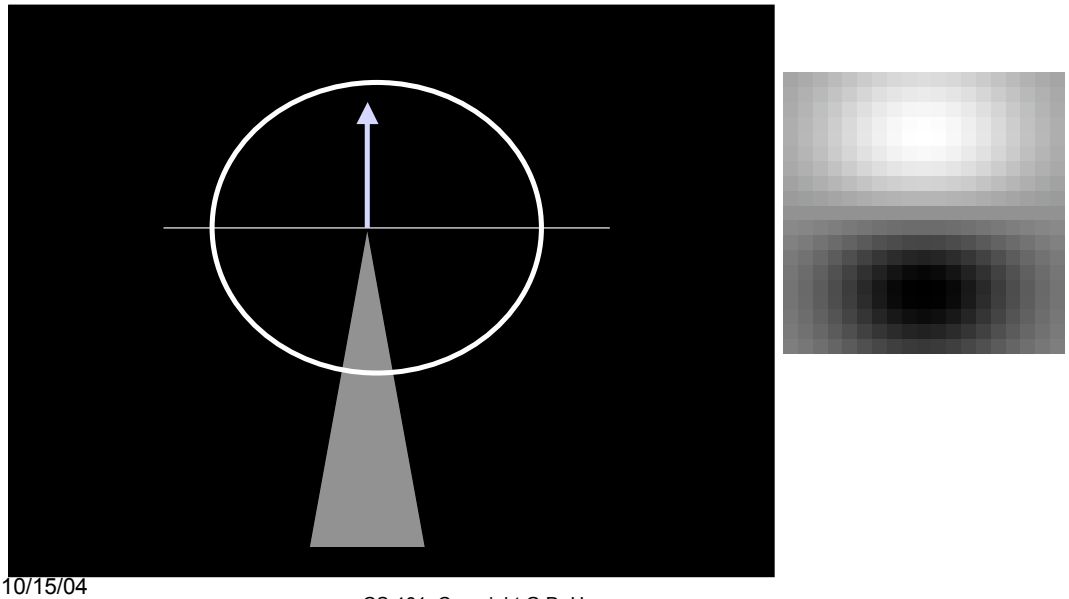


Corners contain more info than lines.

- A corner is easier to match



Edge Detectors Tend to Fail at Corners



Finding Corners

Intuition:

- Right at corner, gradient is ill defined.
- Near corner, gradient has two different values.

Another Use of Gradients: Detecting Corners

- Edges can be thought of as “1D” features
- Corners are “2D” features
- To make this precise, we need to think about the span of the gradients

$$C = \sum_N \nabla I(u) (\nabla I(u))' = R D R'$$

$$D = \mathrm{diag}(\lambda_1, \lambda_2)$$

$$c = \min(\lambda_1, \lambda_2)$$

Formula for Finding Corners

We look at matrix:

Sum over a small region, the hypothetical corner

Gradient with respect to x, times gradient with respect to y

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Matrix is symmetric

WHY THIS?

First, consider case where:

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

This means all gradients in neighborhood are:

(k,0) or (0, c) or (0, 0) (or off-diagonals cancel).

What is region like if:

1. $\lambda 1 = 0$?
2. $\lambda 2 = 0$?
3. $\lambda 1 = 0$ and $\lambda 2 = 0$?
4. $\lambda 1 > 0$ and $\lambda 2 > 0$?

General Case:

From Linear Algebra we haven't talked about it follows that since C is symmetric:

$$C = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

So every case is like one on last slide.

Corners Algorithm

- For every point (u,v)', compute C for a neighborhood about (u,v)
- Sort by the minimum singular value of C
- Read off locations starting with highest values and working down until enough locations are found or we run out of locations
  - optional: as we read down, discard corners that are within a small distance of corners that have appeared higher in the list

Edges Summary

- Filtering is a way of removing noise or suppressing/enhancing frequency content
- Typically, we combine some type of image derivative with smoothing
- Image gradients are the basic tool in 2D images
- Derivative of Gaussian is generally the gradient operator of choice
- Canny detector is probably the most widely used algorithm for performing edge detection