

COS 429: COMPUTER VISION

Linear Filters and Edge Detection

- convolution
- shift invariant linear system
- Fourier Transform
- Aliasing and sampling
- scale representation
- edge detection
- corner detection

Reading: Chapters 7, 8



丹青不知老將去，富貴於我如浮雲。千載誰知意氣存，
我亦少陵情結盡。投筆如何/區區眉！一在至九年 孟頫



Linear Filters

- **Linear filtering:**
 - Form a new image whose pixels are a weighted sum of original pixel values, using the same set of weights at each point

Convolution

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

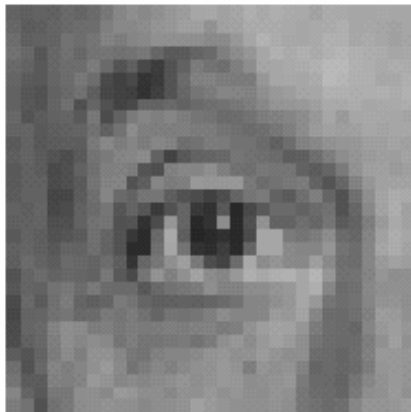
- Represent the linear weights as an image, F
- F is called the **kernel**
- Operation is called **convolution**
 - Center origin of the kernel F at each pixel location
 - Multiply weights by corresponding pixels
 - Set resulting value for each pixel

- Image, R , resulting from convolution of F with image H , where u, v range over kernel pixels:

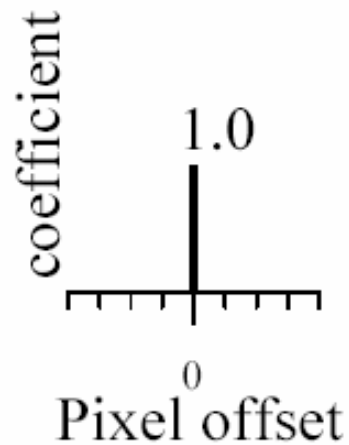
$$R_{ij} = \sum_{u,v} H_{i-u, j-v} F_{uv}$$

Warning: the textbook mixes up H and F

Linear filtering (warm-up slide)



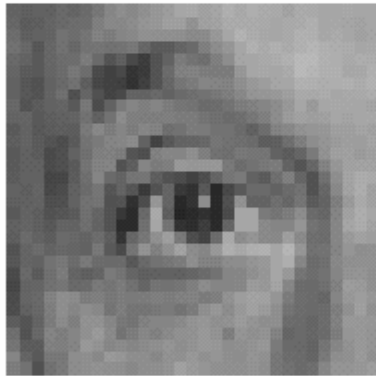
original



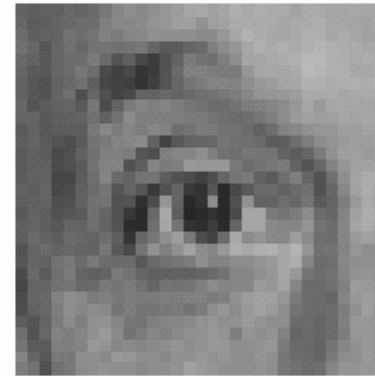
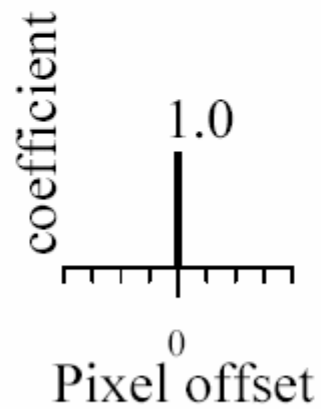
?

Slide credits for these examples: Bill Freeman, David Jacobs

Linear filtering (warm-up slide)

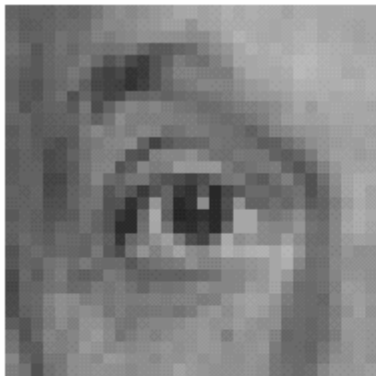


original

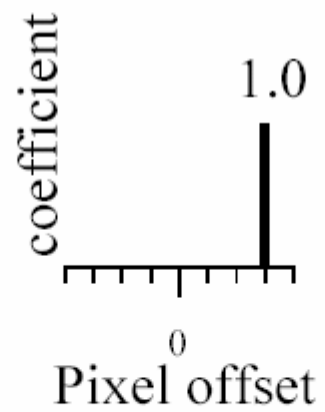


Filtered
(no change)

Linear filtering

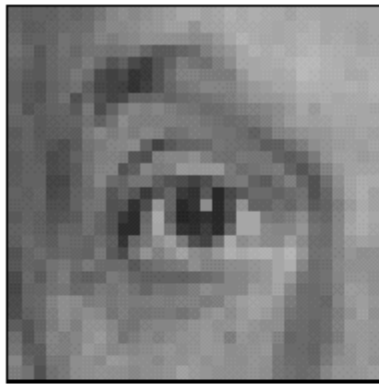


original

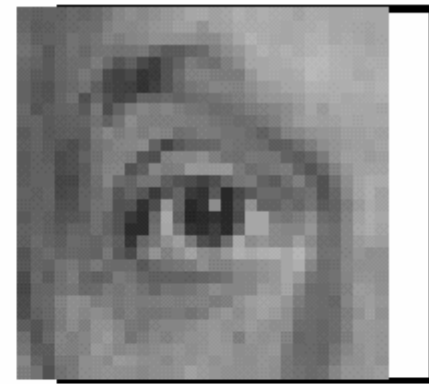
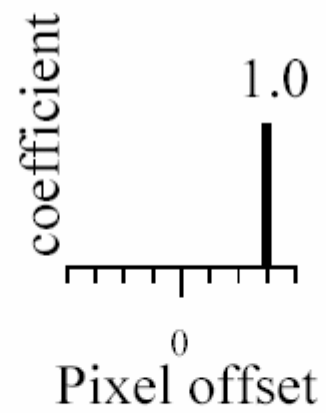


?

shift

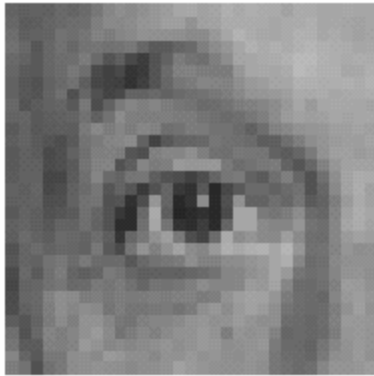


original

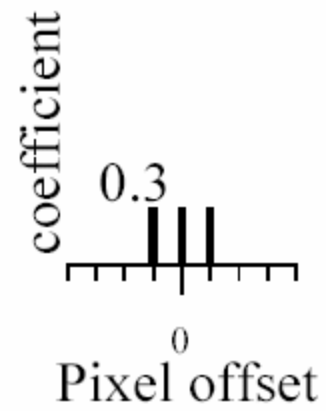


shifted

Linear filtering

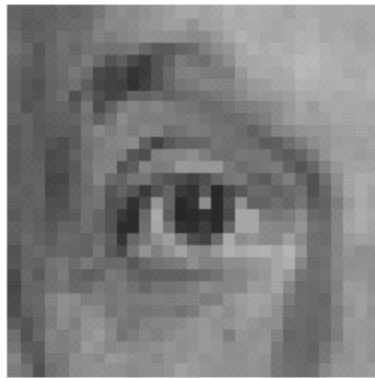


original

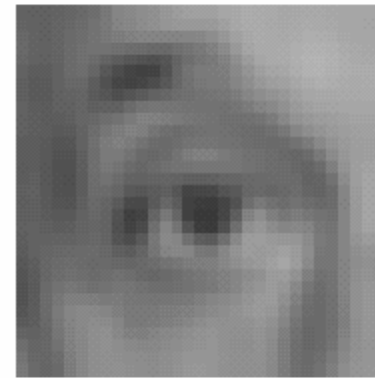
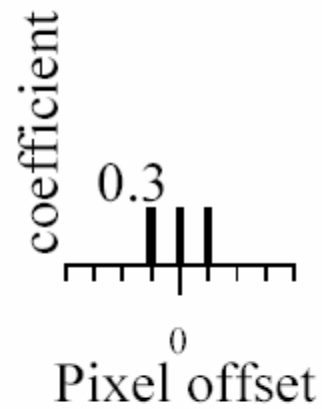


?

Blurring

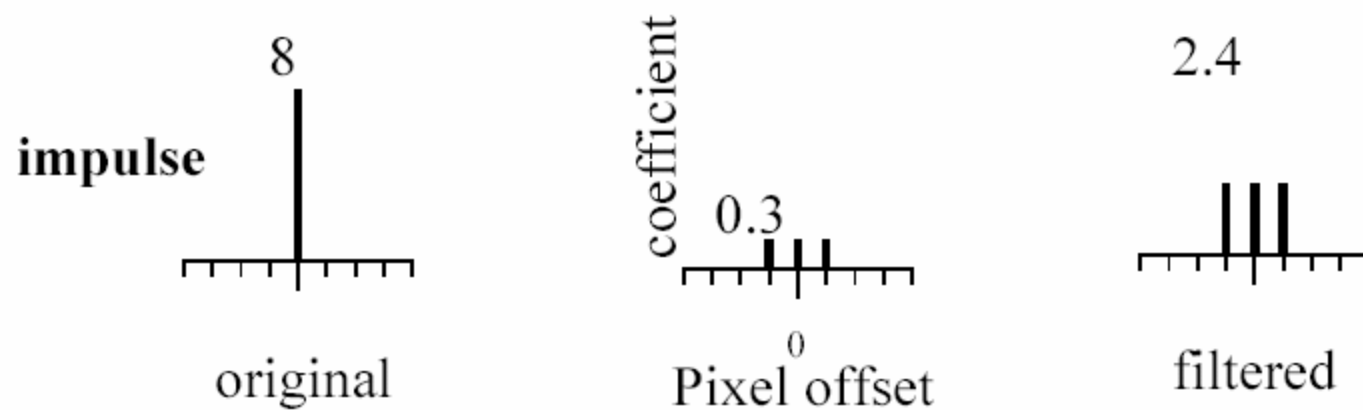


original

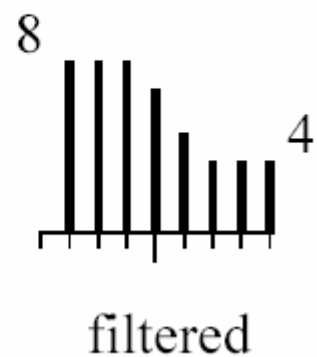
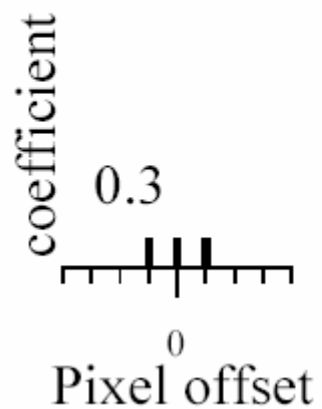
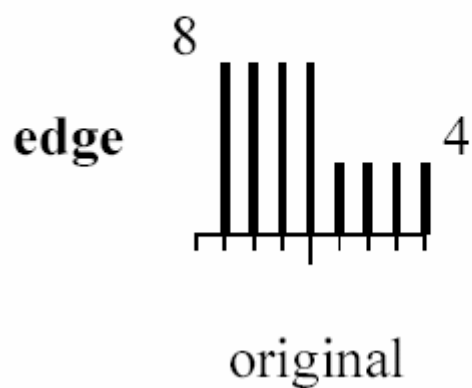
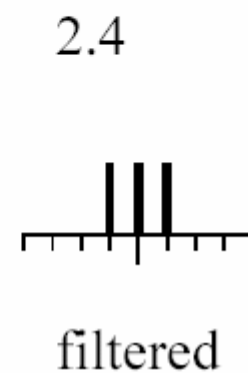
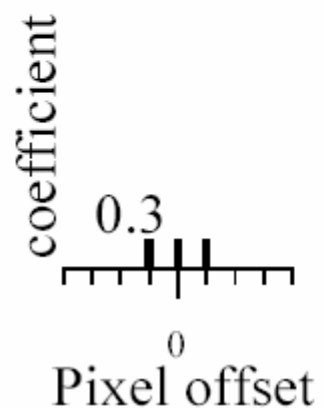
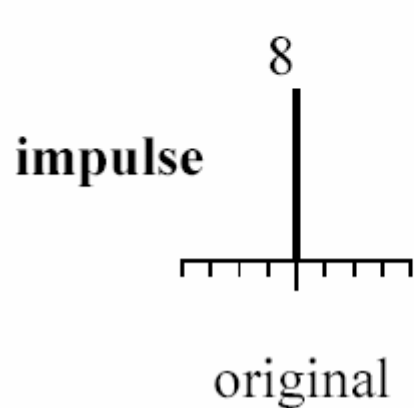


Blurred (filter applied in both dimensions).

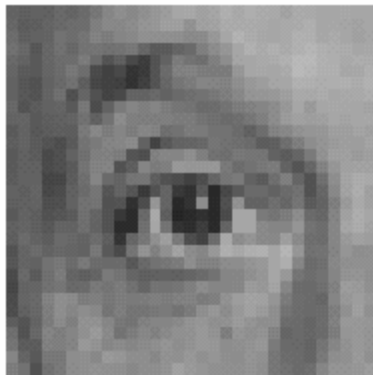
Blur examples



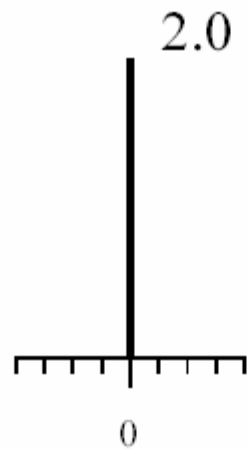
Blur examples



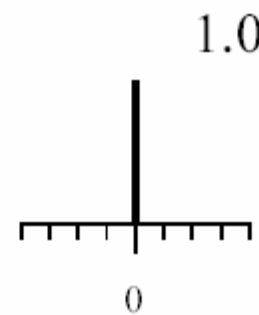
Linear filtering (warm-up slide)



original

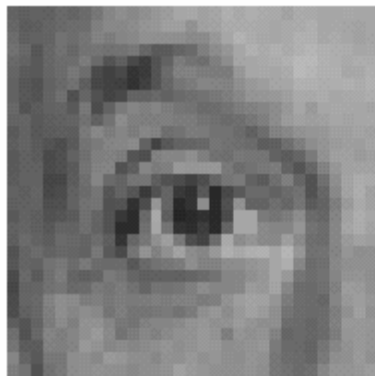


—

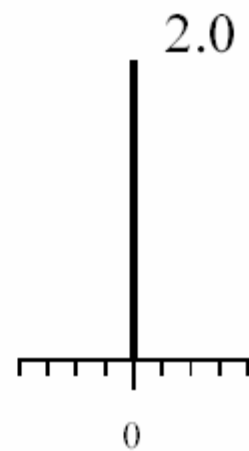


?

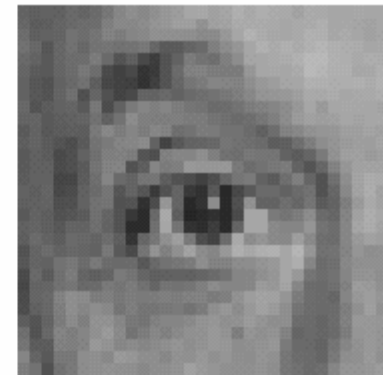
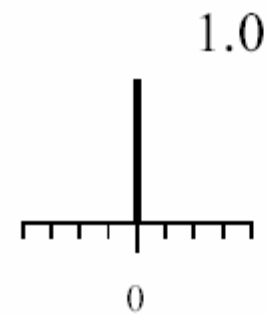
Linear filtering (no change)



original

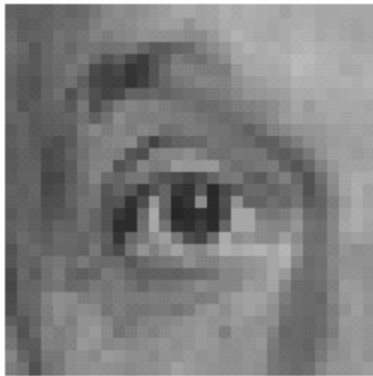


—

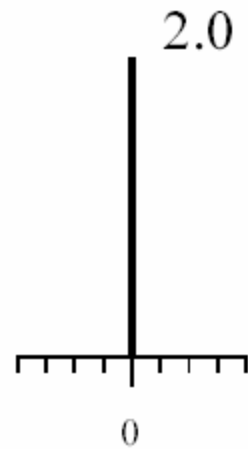


Filtered
(no change)

Linear filtering



original

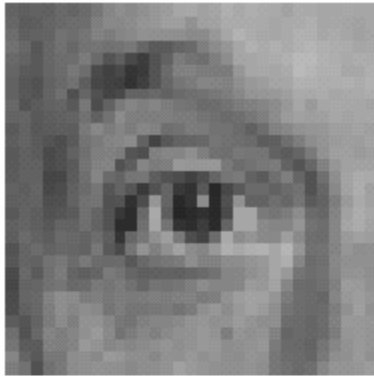


—

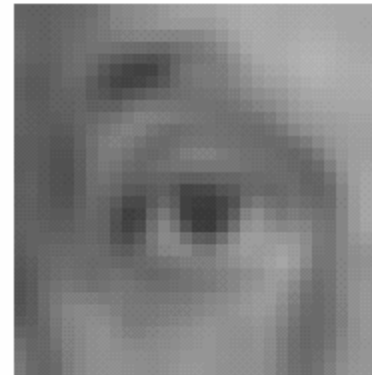
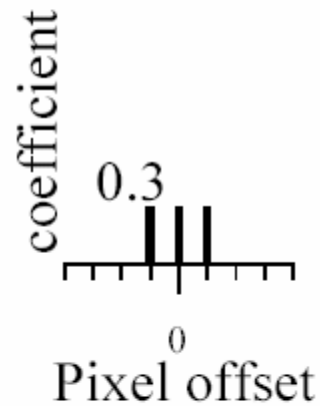


?

(remember blurring)

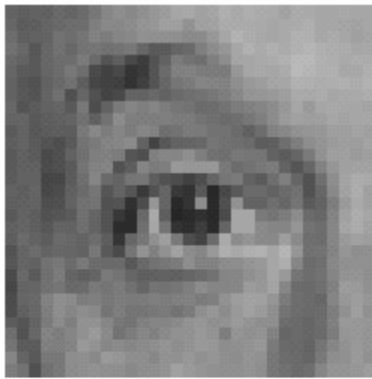


original

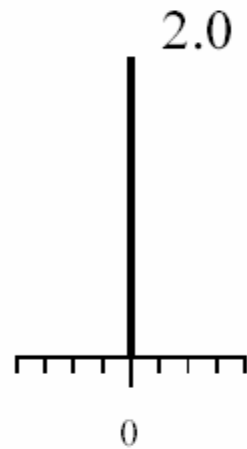


Blurred (filter applied in both dimensions).

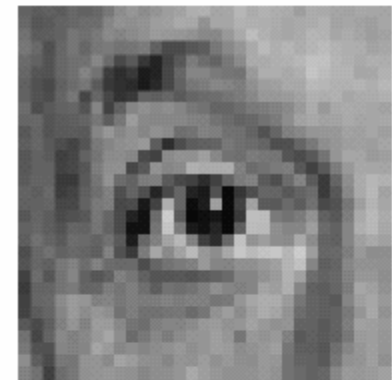
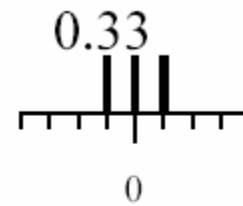
Sharpening



original

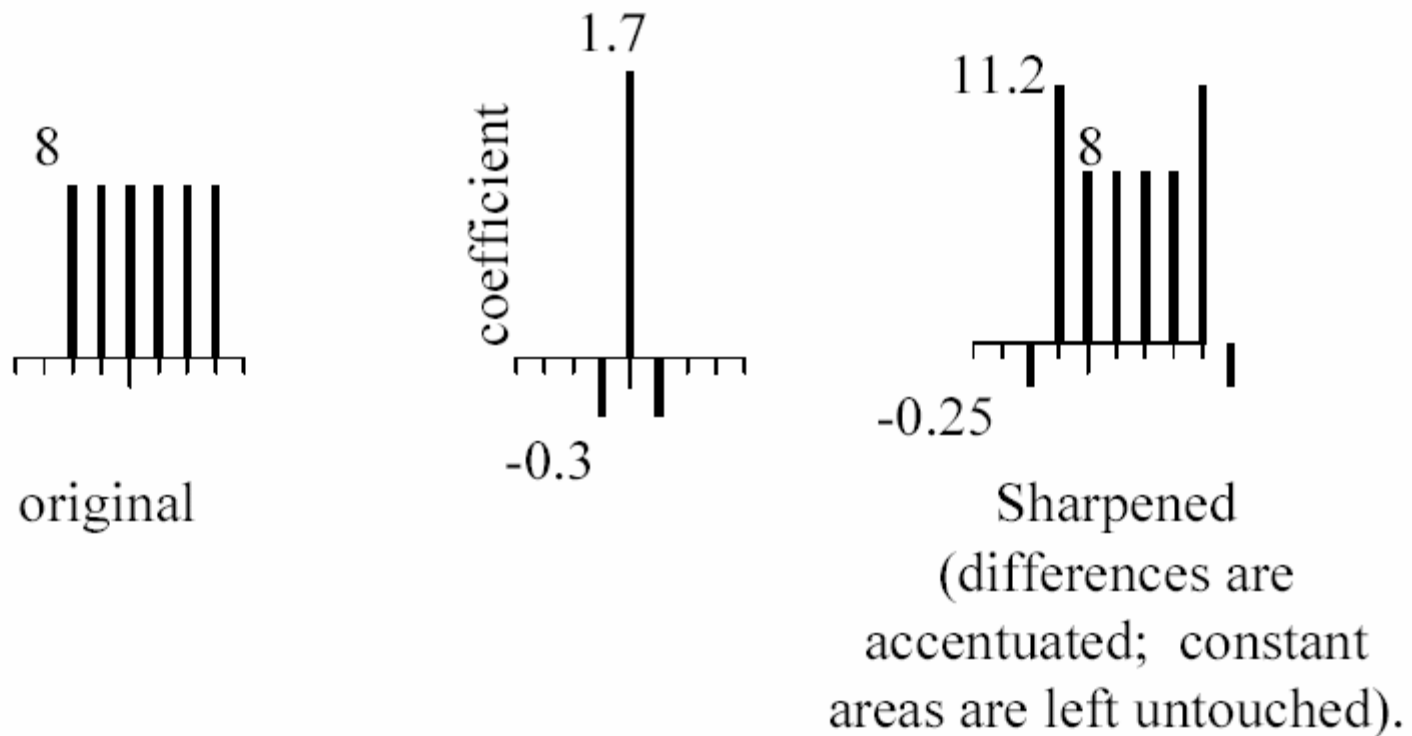


—

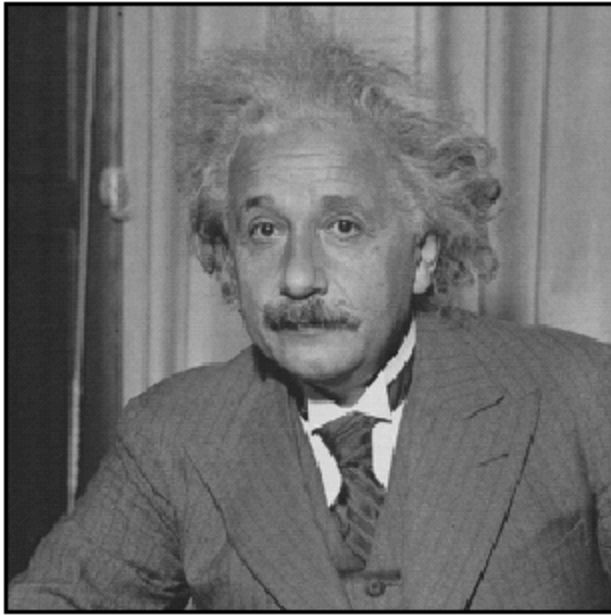


Sharpened
original

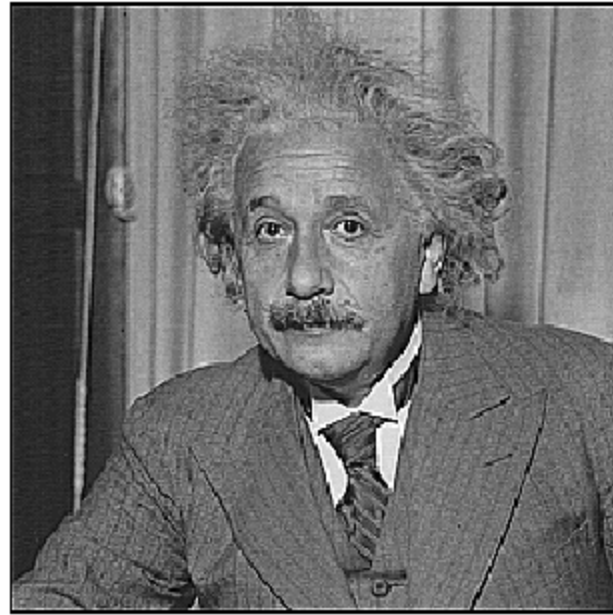
Sharpening example



Sharpening

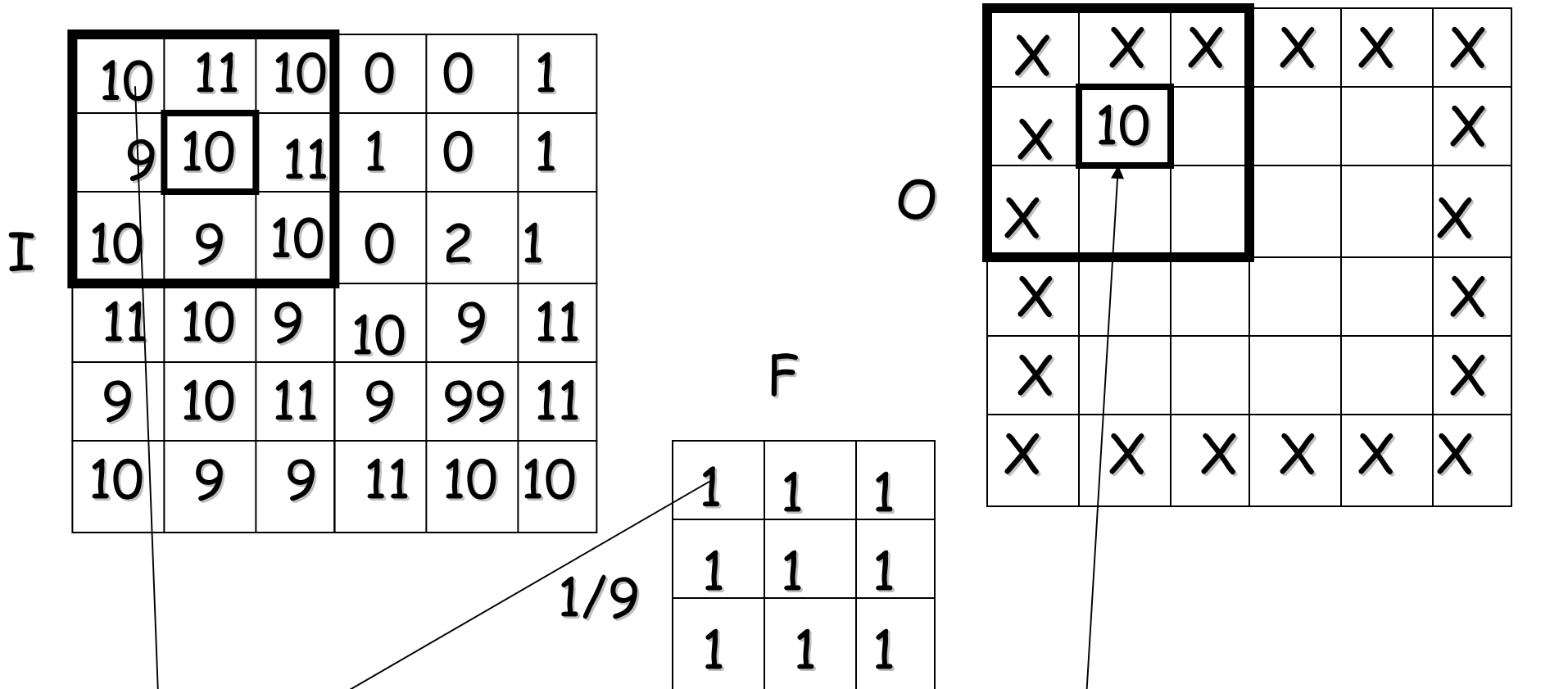


before



after

Convolution



$$1/9.(10 \times 1 + 11 \times 1 + 10 \times 1 + 9 \times 1 + 10 \times 1 + 11 \times 1 + 10 \times 1 + 9 \times 1 + 10 \times 1) = 1/9.(90) = 10$$

Average filter (box filter)

- Mask with positive entries, that sum to 1.
- Replaces each pixel with an average of its neighborhood.
- If all weights are equal, it is called a **box** filter.

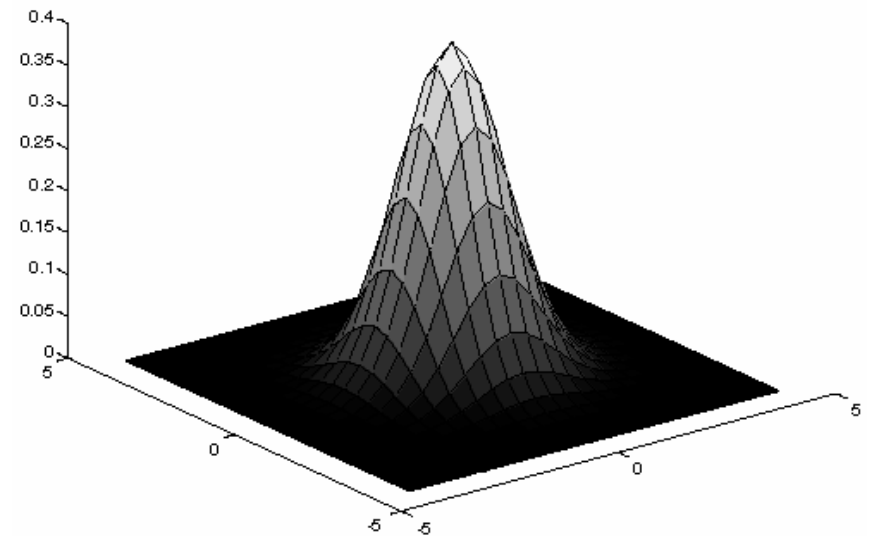
1 — 9	1	1	1
	1	1	1
	1	1	1

Example: Smoothing with a box filter



Smoothing with a Gaussian

- Smoothing with a box actually doesn't compare at all well with a defocussed lens
- Most obvious difference is that a single point of light viewed in a defocussed lens looks like a fuzzy blob; but the averaging process would give a little square.

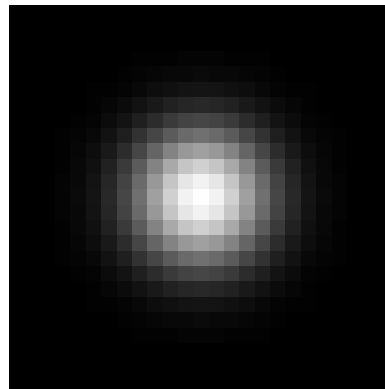
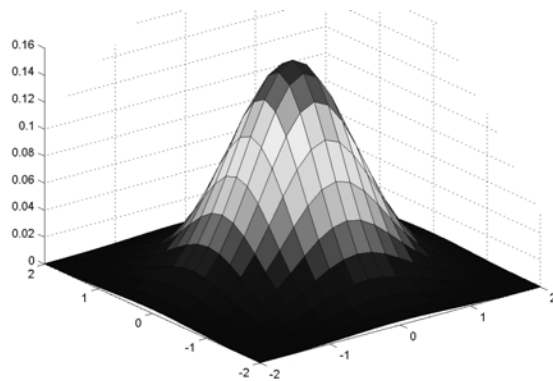


- A Gaussian gives a good model of a fuzzy blob
- It closely models many physical processes (the sum of many small effects)

Slide credit: David Lowe (UBC)

Gaussian Kernel

- Idea: Weight contributions of neighboring pixels by nearness



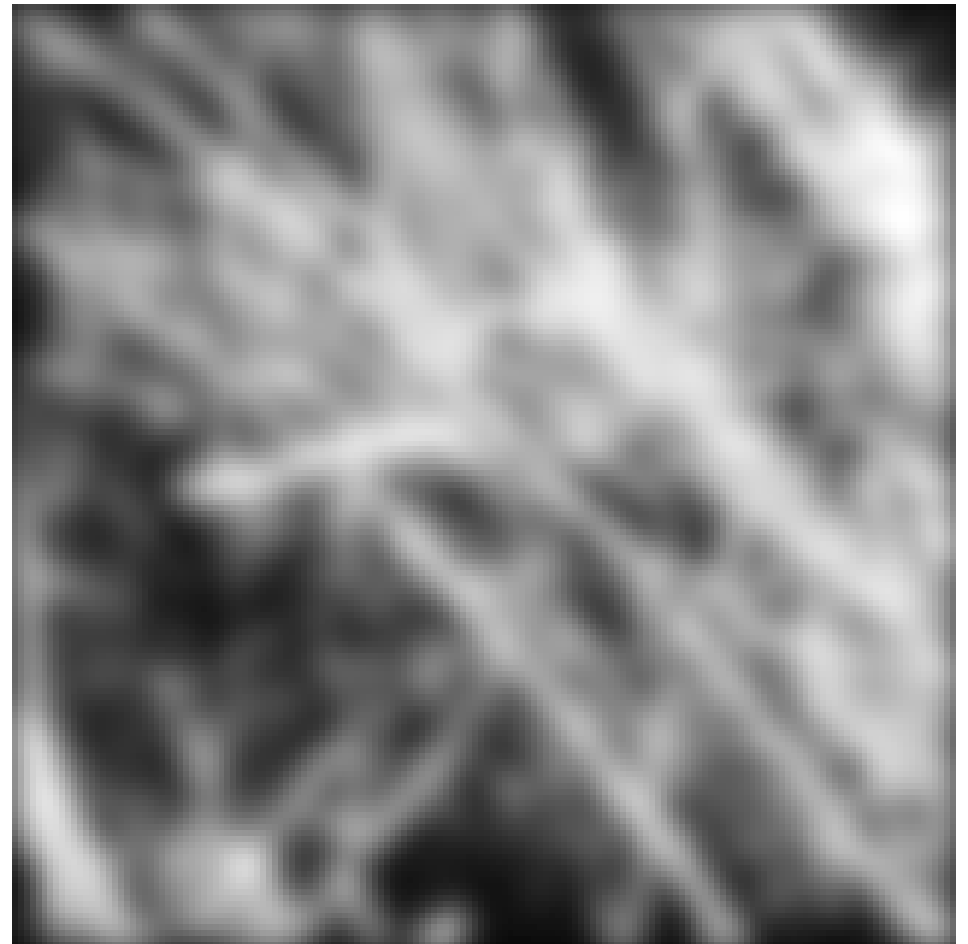
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

- Constant factor at front makes volume sum to 1 (can be ignored, as we should normalize weights to sum to 1 in any case).

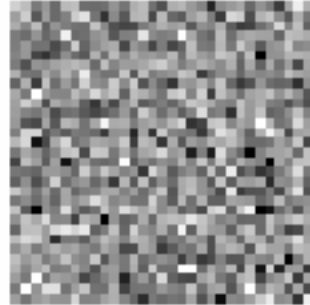
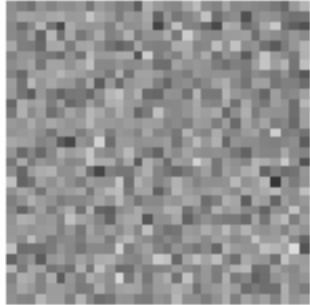
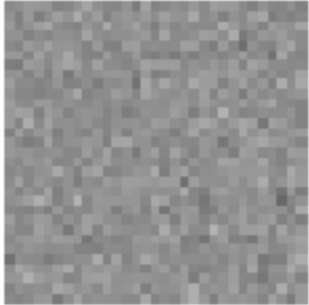
Smoothing with a Gaussian



$\sigma=0.05$

$\sigma=0.1$

$\sigma=0.2$



no
smoothing

Smoothing reduces pixel noise:



$\sigma=1$ pixel

Each row shows smoothing with Gaussians of different width; each column shows different amounts of Gaussian noise.



$\sigma=2$ pixels

Efficient Implementation

- Both the BOX filter and the Gaussian filter are **separable** into two 1D convolutions:
 - First convolve each row with a 1D filter
 - Then convolve each column with a 1D filter.

Differentiation and convolution

- Recall, for 2D function, $f(x,y)$:

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

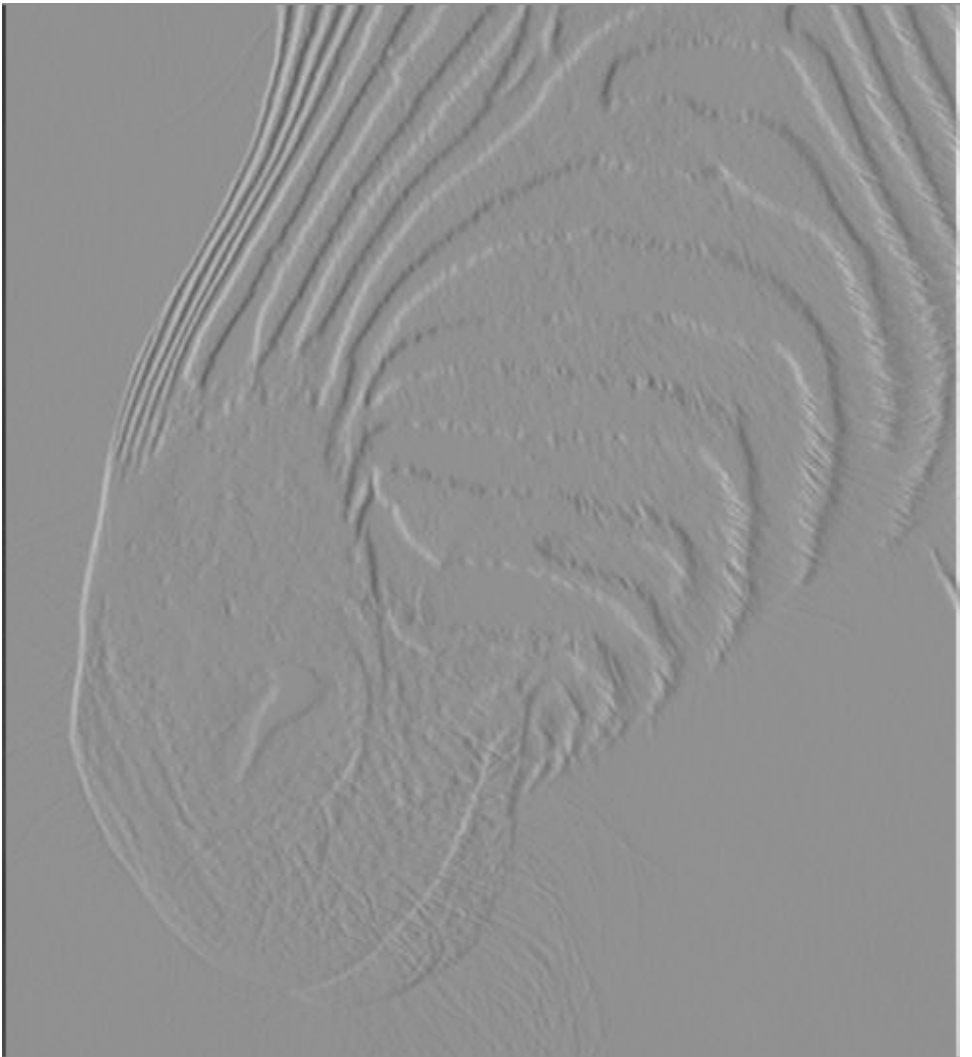
- This is linear and shift invariant, so must be the result of a convolution.

- We could approximate this as

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

(which is obviously a convolution)

Vertical gradients from finite differences



Shift invariant linear systems

- 3 properties
 - Superposition
 - Scaling
 - Shift invariance

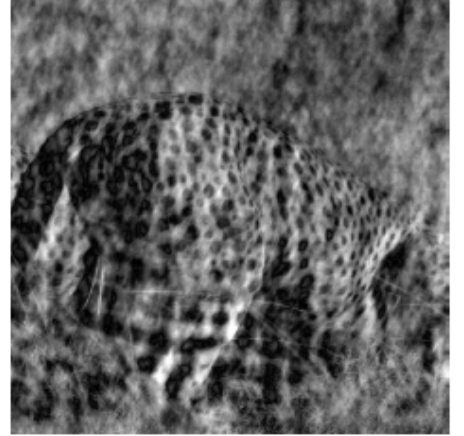
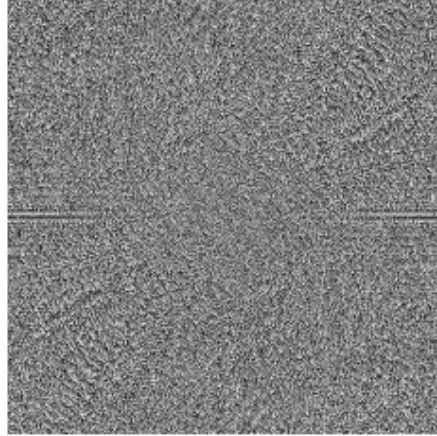
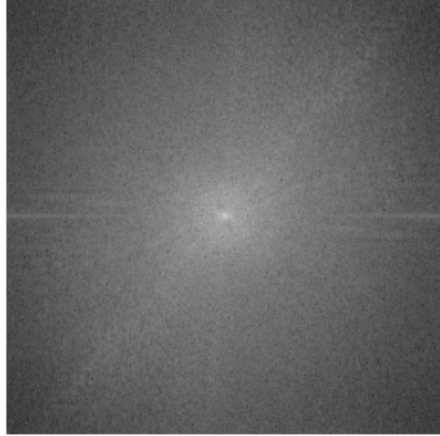
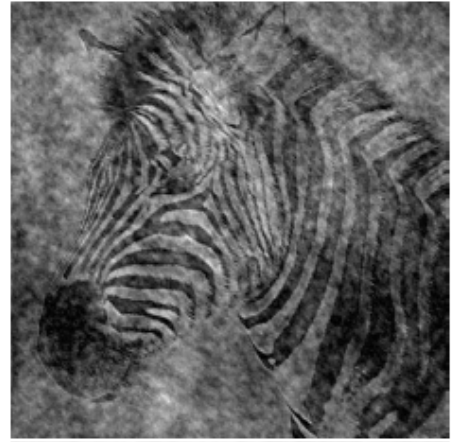
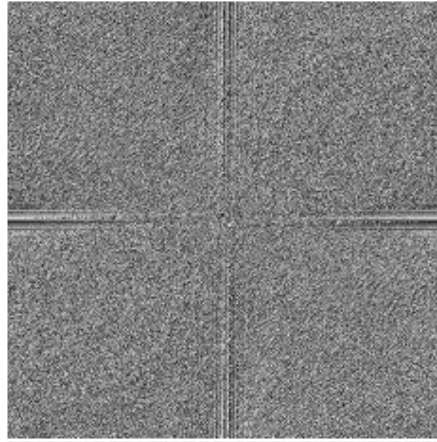
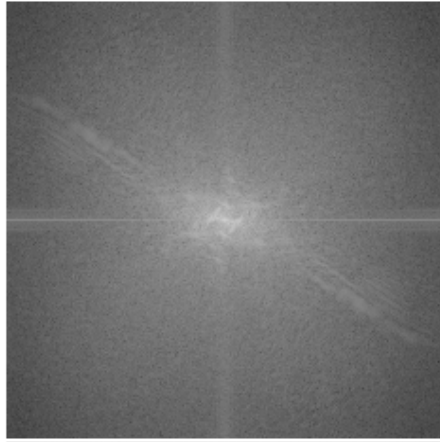
Discrete convolution

- 1D
- 2D
- “edge effects” in discrete convolution

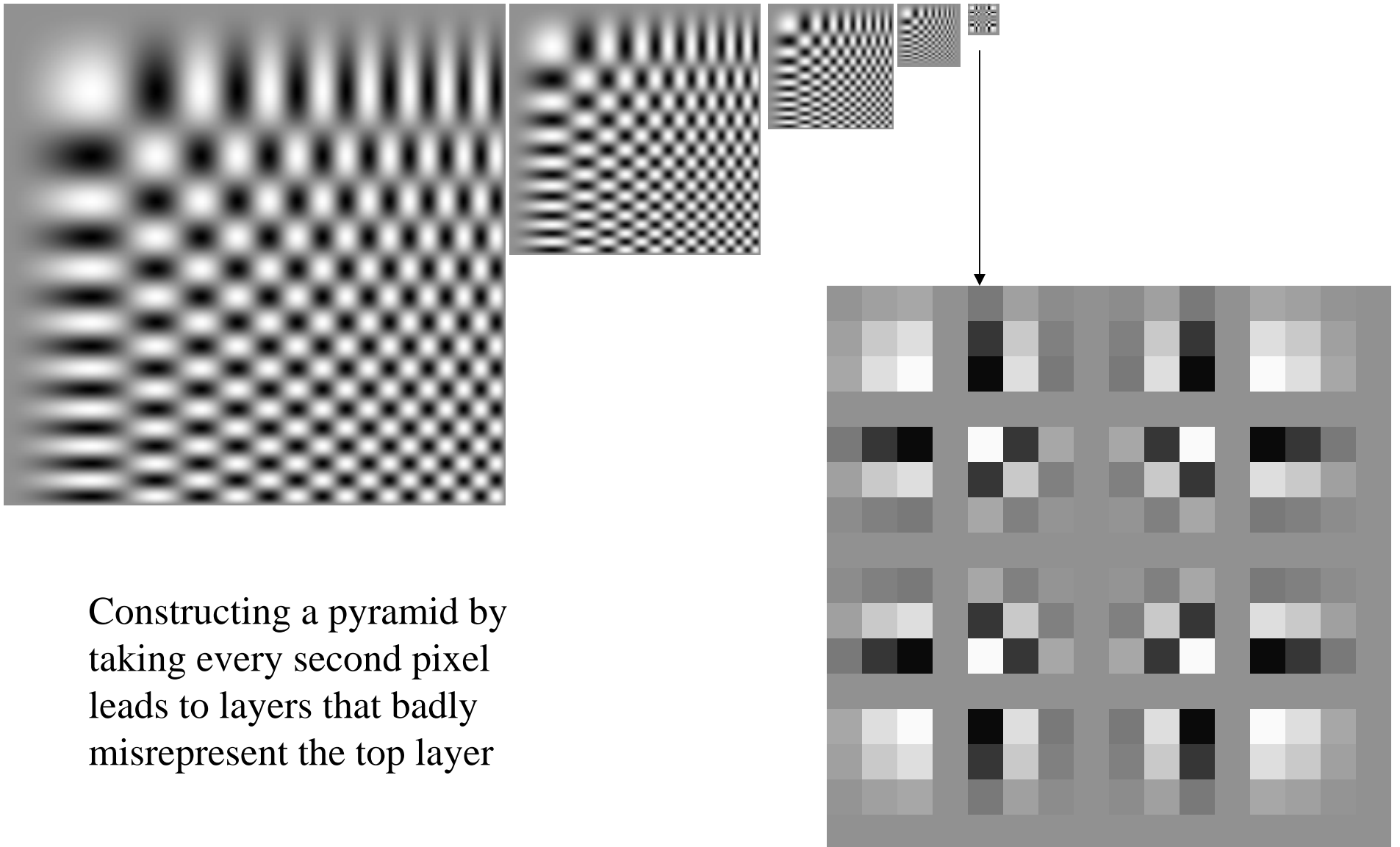
Spatial frequency and Fourier Transform

$$\mathcal{F}(g(x, y))(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) e^{-i2\pi(ux+vy)} dx dy$$

Function	Fourier transform
$g(x, y)$	$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) e^{-i2\pi(ux+vy)} dx dy$
$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \mathcal{F}(g(x, y))(u, v) e^{i2\pi(ux+vy)} du dv$	$\mathcal{F}(g(x, y))(u, v)$
$\delta(x, y)$	1
$\frac{\partial f}{\partial x}(x, y)$	$u\mathcal{F}(f)(u, v)$
$0.5\delta(x + a, y) + 0.5\delta(x - a, y)$	$\cos 2\pi au$
$e^{-\pi(x^2+y^2)}$	$e^{-\pi(u^2+v^2)}$
$box_1(x, y)$	$\frac{\sin u}{u} \frac{\sin v}{v}$
$f(ax, by)$	$\frac{\mathcal{F}(f)(u/a, v/b)}{ab}$
$\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \delta(x - i, y - j)$	$\sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \delta(u - i, v - j)$
$(f * *g)(x, y)$	$\mathcal{F}(f)\mathcal{F}(g)(u, v)$
$f(x - a, y - b)$	$e^{-i2\pi(au+bv)}\mathcal{F}(f)$
$f(x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta)$	$\mathcal{F}(f)(u \cos \theta - v \sin \theta, u \sin \theta + v \cos \theta)$



Sampling and aliasing



Constructing a pyramid by taking every second pixel leads to layers that badly misrepresent the top layer

Sampling in 1D

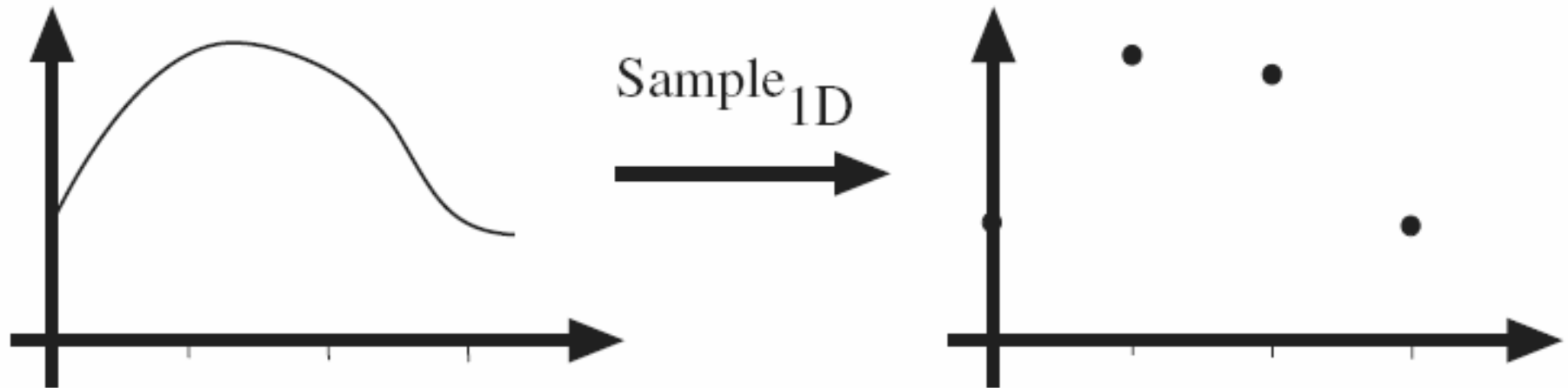
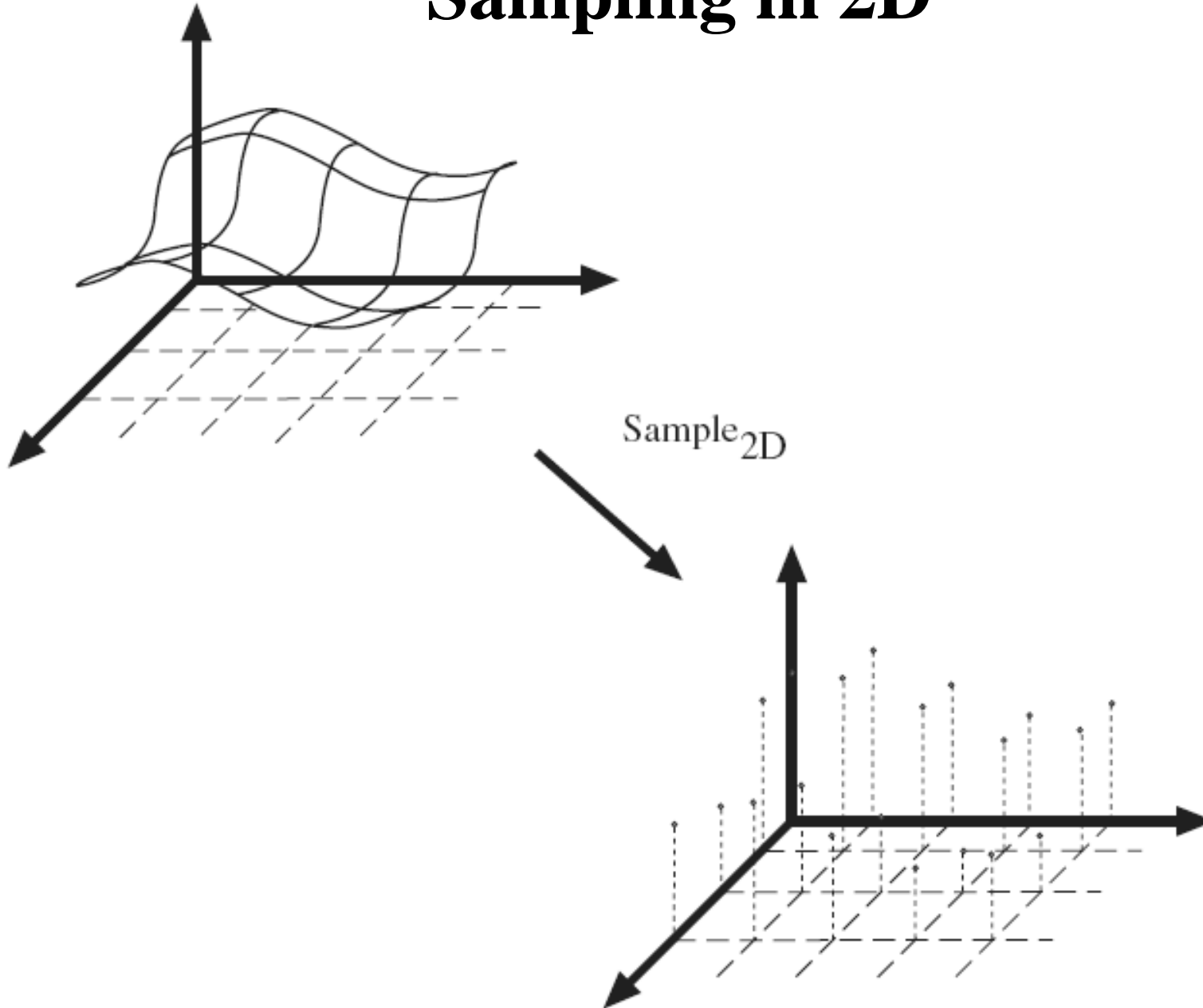
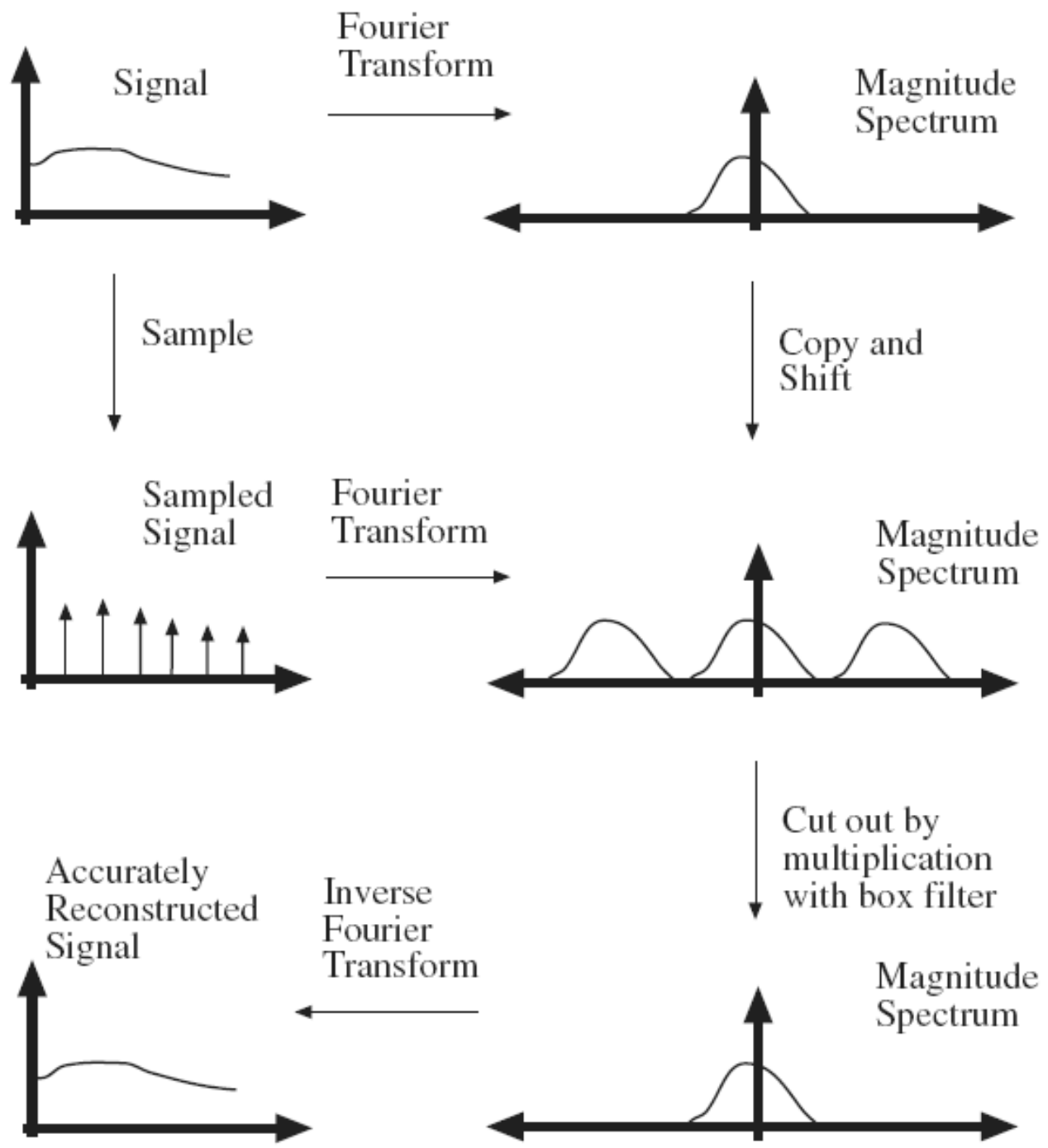
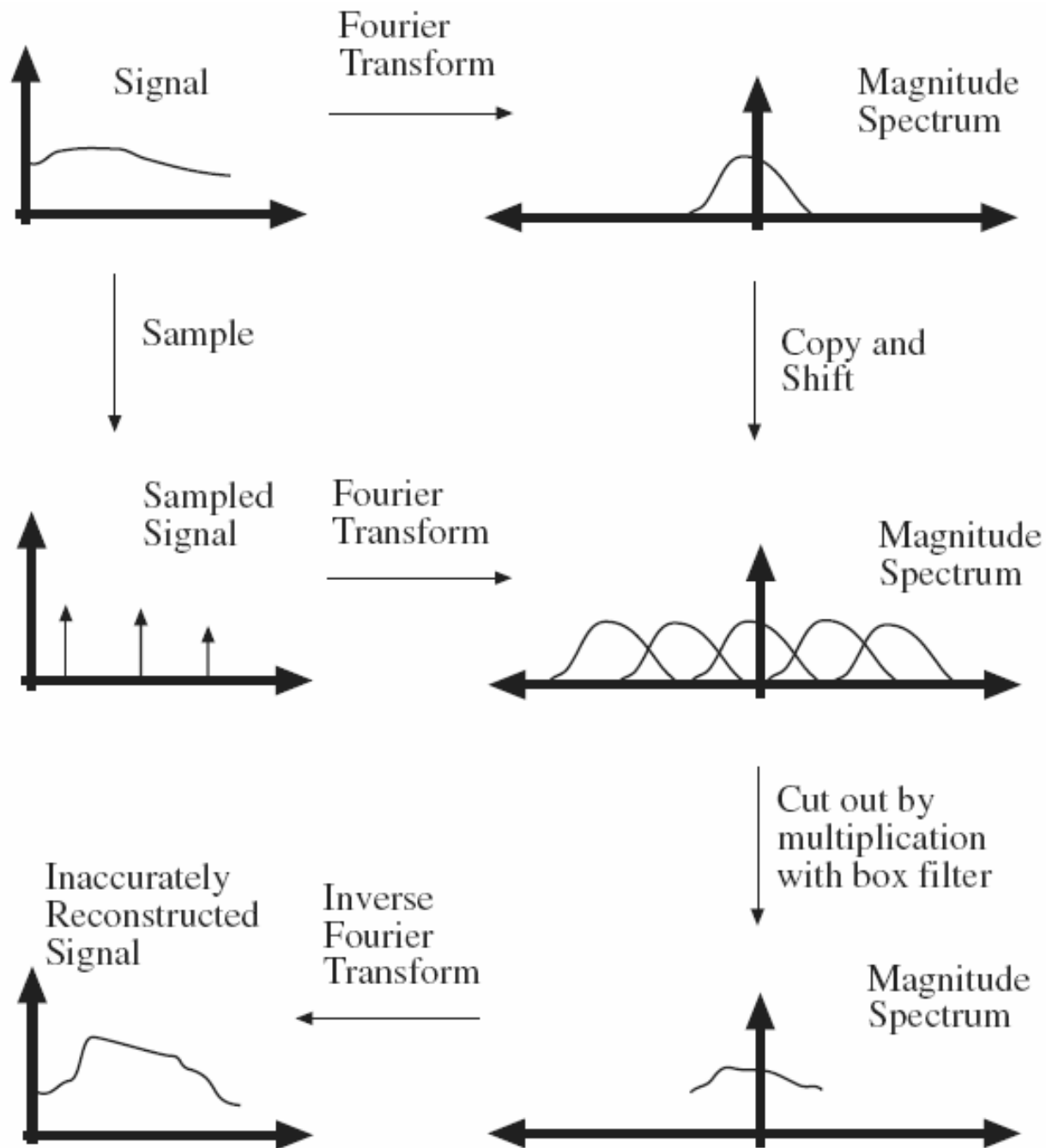


FIGURE 7.8: Sampling in 1D takes a function, and returns a vector whose elements are values of that function at the sample points, as the top figures show. For our purposes, it is enough that the sample points be integer values of the argument. We allow the vector to be infinite dimensional, and have negative as well as positive indices.

Sampling in 2D







Aliasing!

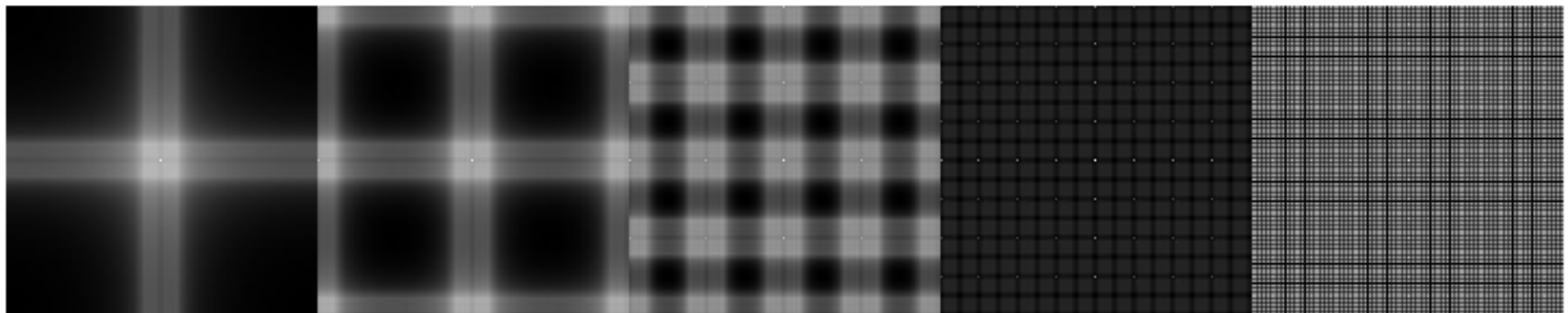
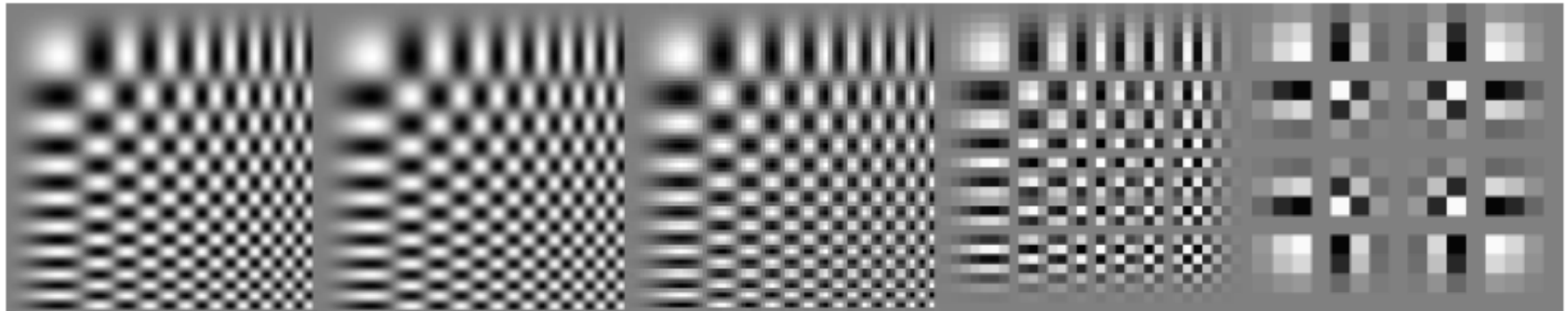
256x256

128x128

64x64

32x32

16x16



Smoothing and resampling

- Nyquist's theorem

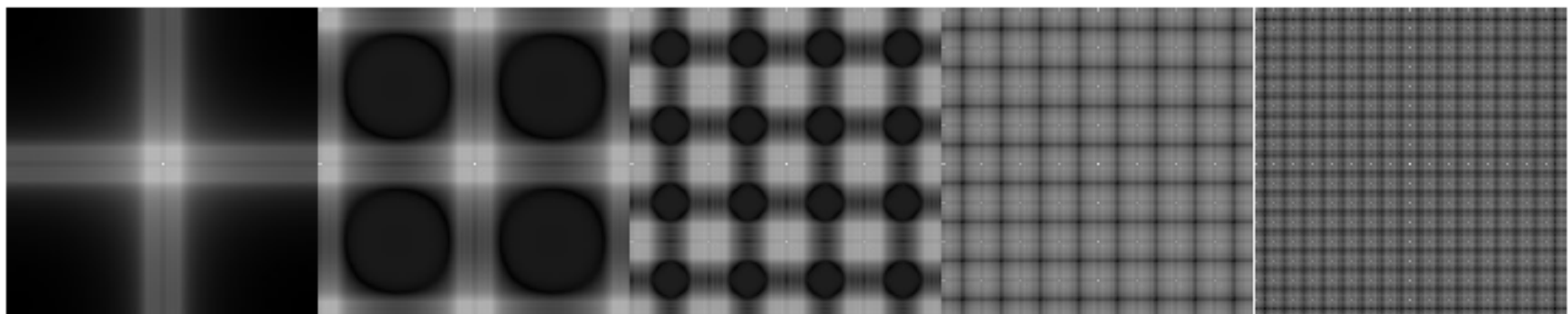
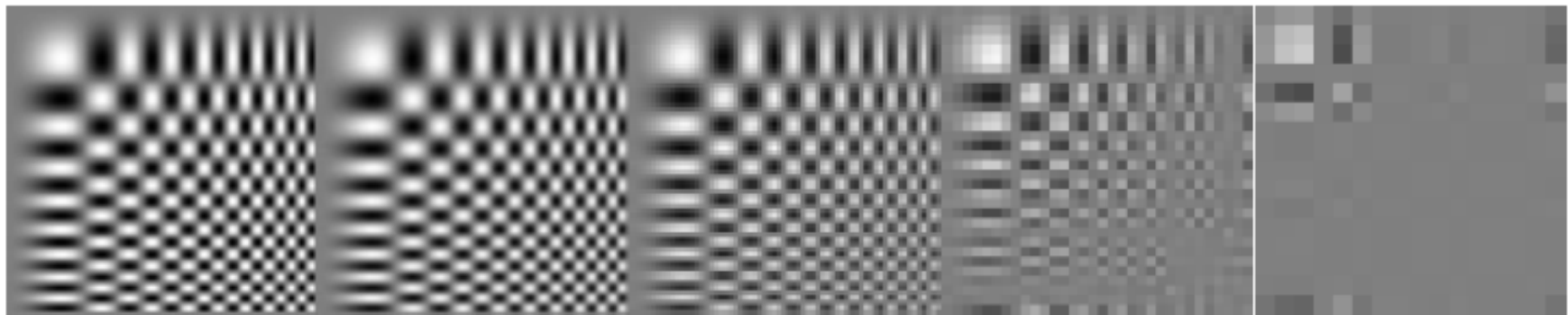
256x256

128x128

64x64

32x32

16x16



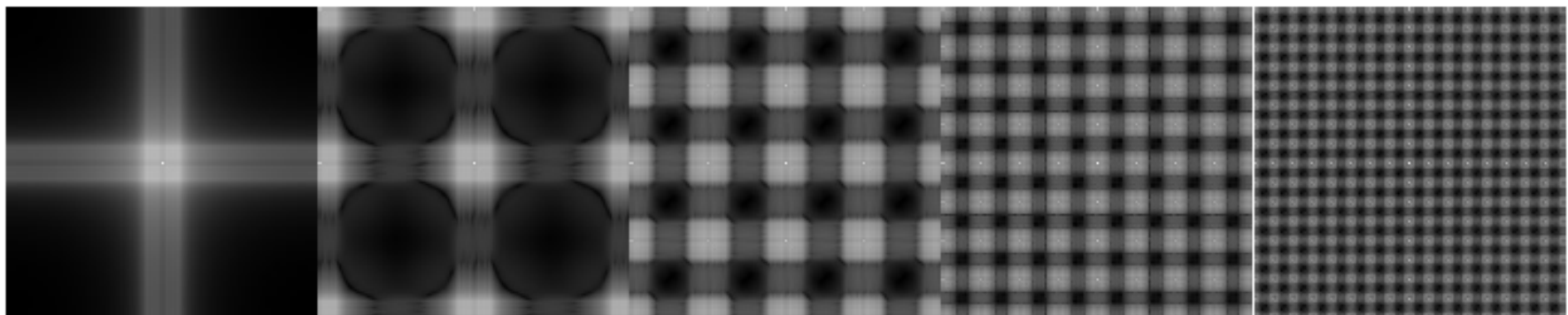
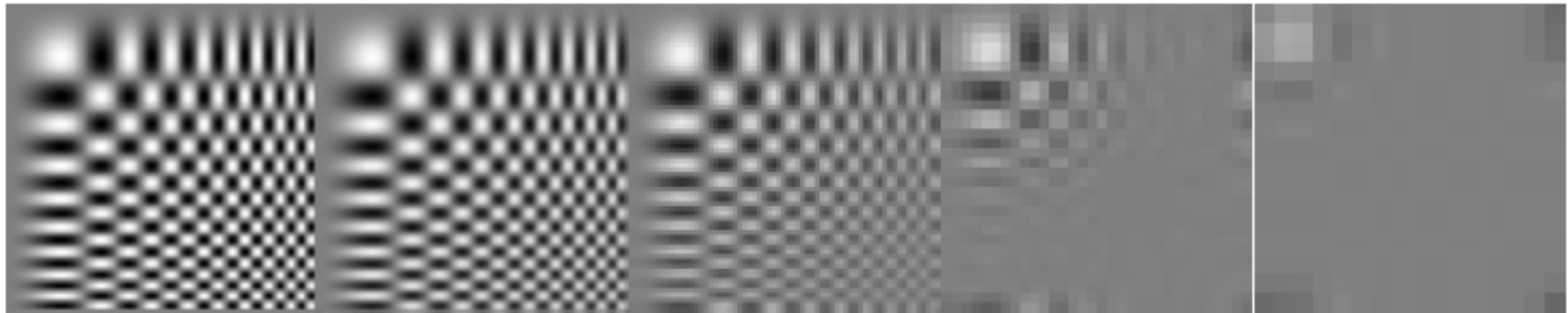
256x256

128x128

64x64

32x32

16x16



Algorithm

Algorithm 7.1: Sub-sampling an Image by a Factor of Two

Apply a low-pass filter to the original image

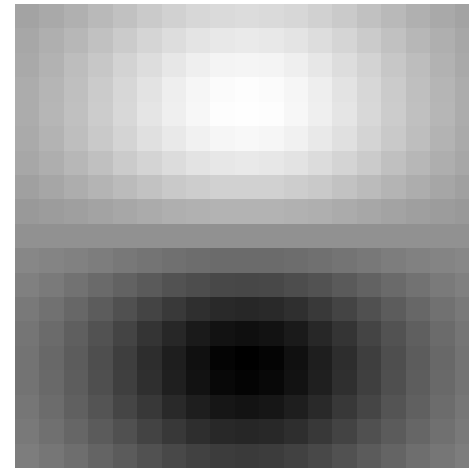
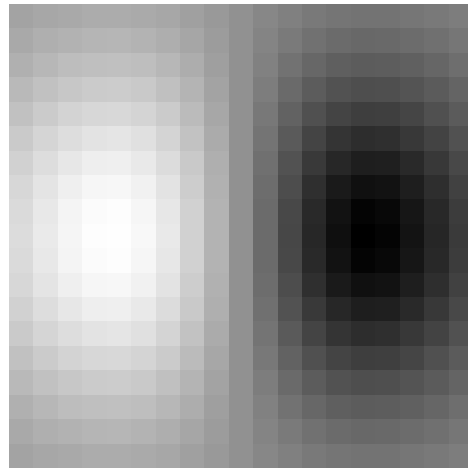
(a Gaussian with a σ of between one and two pixels is usually an acceptable choice).

Create a new image whose dimensions on edge are half those of the old image

Set the value of the i, j 'th pixel of the new image to the value of the $2i, 2j$ 'th pixel of the filtered image

Filters are templates

- Applying a filter at some point can be seen as taking a dot-product between the image and some vector
- Filtering the image is a set of dot products
- Insight
 - filters look like the effects they are intended to find
 - filters find effects they look like



Normalized correlation

- Think of filters as a dot product of the filter vector with the image region
 - Now measure the angle between the vectors

$$a \cdot b = |a| |b| \cos \theta$$

- Angle (similarity) between vectors can be measured by normalizing length of each vector to 1.
- Normalized correlation: divide each correlation by square root of sum of squared values (length)

Application: Vision system for TV remote control

- uses template matching

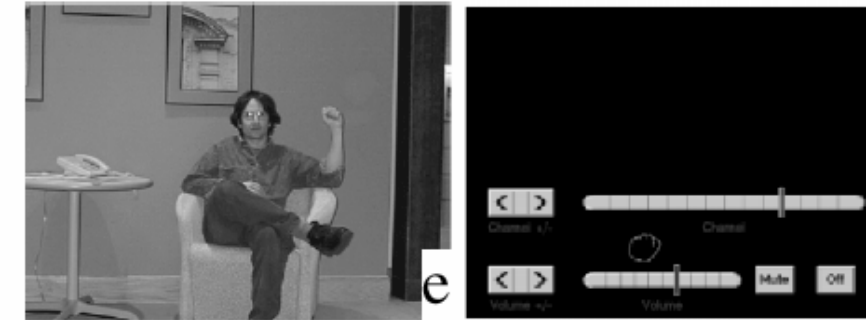
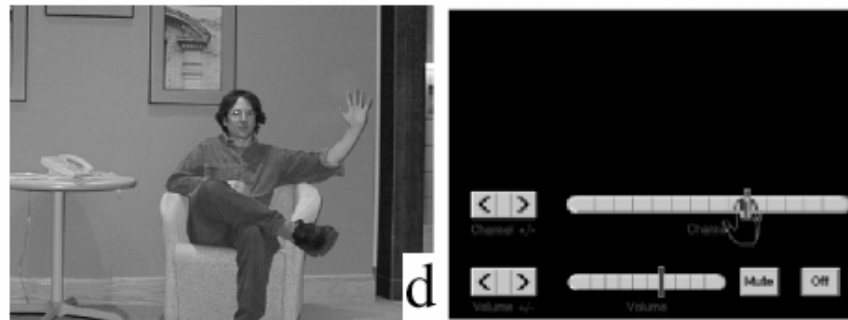
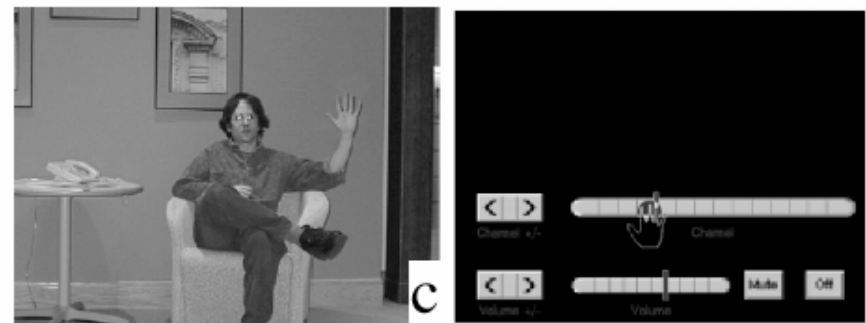
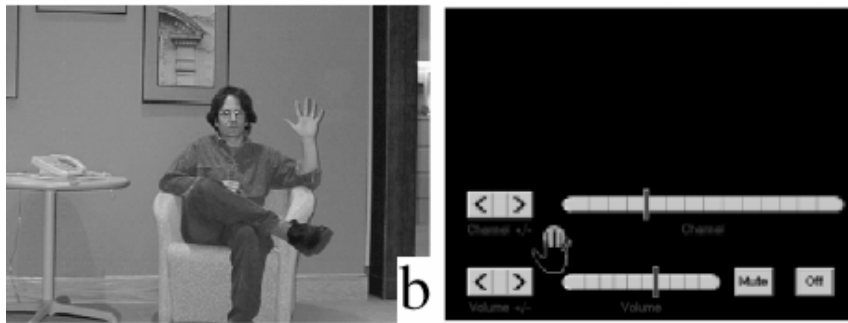


Figure from "Computer Vision for Interactive Computer Graphics," W.Freeman et al, IEEE Computer Graphics and Applications, 1998 copyright 1998, IEEE

We need scaled representations

- Find template matches at all scales
 - e.g., when finding hands or faces, we don't know what size they will be in a particular image
 - Template size is constant, but image size changes
- Efficient search for correspondence
 - look at coarse scales, then refine with finer scales
 - much less cost, but may miss best match
- Examining all levels of detail
 - Find edges with different amounts of blur
 - Find textures with different spatial frequencies (levels of detail)

The Gaussian pyramid

- Create each level from previous one:
 - smooth and sample
- Smooth with Gaussians, in part because
 - a Gaussian*Gaussian = another Gaussian
 - $G(x) * G(y) = G(\sqrt{x^2 + y^2})$



512

256

128

64

32

16

8



All the extra levels add very little overhead for memory or computation!

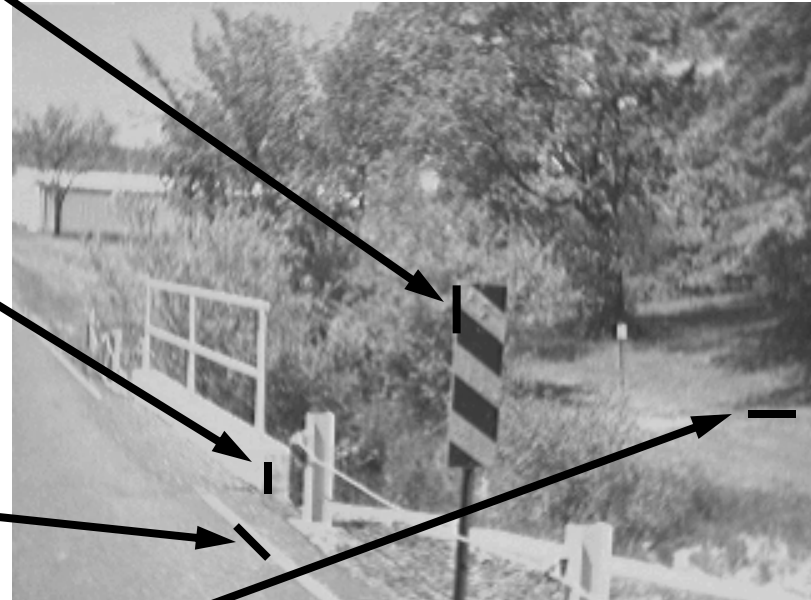
Edge and Corner Detection

- **Goal:** Identify sudden changes (discontinuities) in an image
- This is where most shape information is encoded
- **Example:** artist's line drawing (but artist is also using object-level knowledge)



What causes an edge?

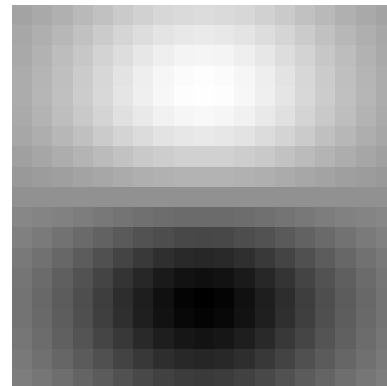
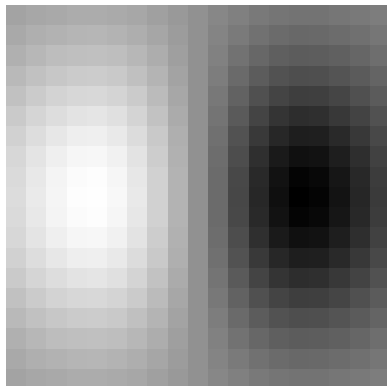
- Depth discontinuity
- Surface orientation discontinuity
- Reflectance discontinuity (i.e., change in surface material properties)
- Illumination discontinuity (e.g., shadow)



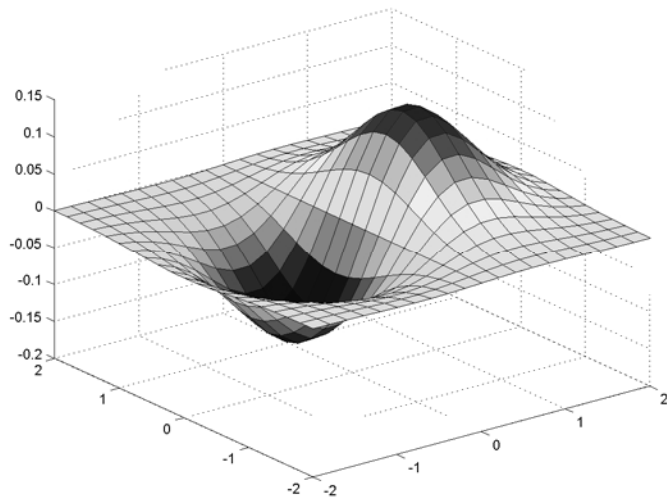
Smoothing and Differentiation

- **Edge:** a location with high gradient (derivative)
- Need smoothing to reduce noise prior to taking derivative
- Need two derivatives, in x and y direction.
- We can use derivative of Gaussian filters
 - because differentiation is convolution, and convolution is associative:

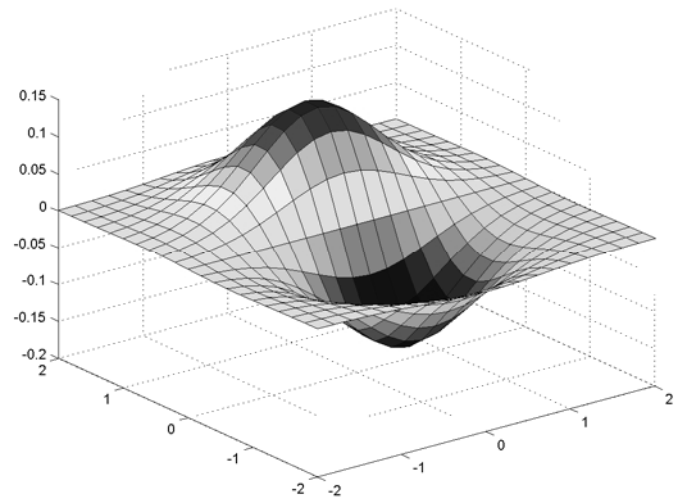
$$D * (G * I) = (D * G) * I$$



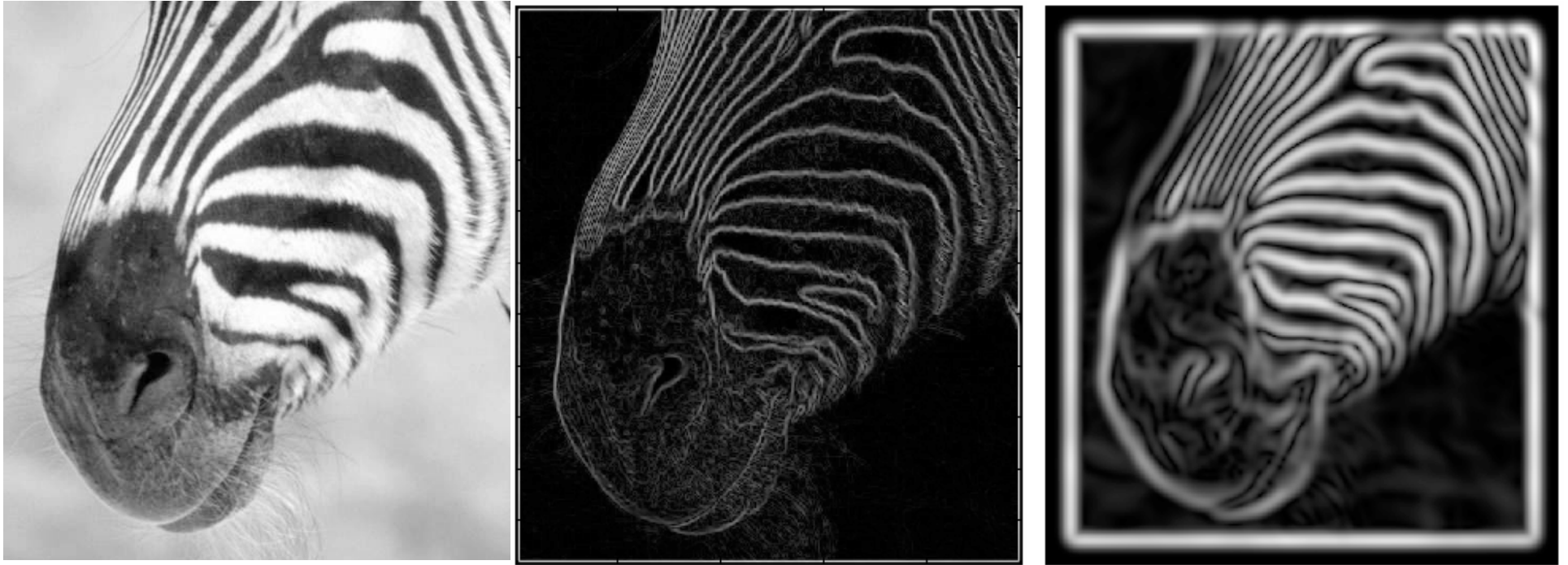
Derivative of Gaussian



$$\frac{\partial}{\partial x} G_{\sigma}$$



$$\frac{\partial}{\partial y} G_{\sigma}$$



Scale

Increased smoothing:

- Eliminates noise edges.
- Makes edges smoother and thicker.
- Removes fine detail.

Edge Detection

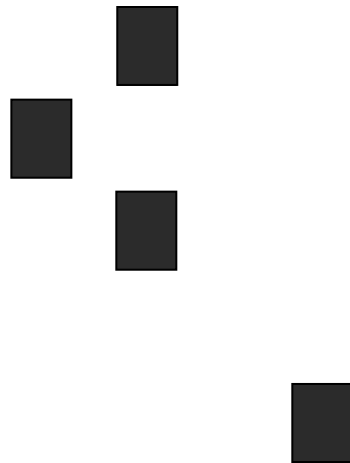
- Criteria for optimal edge detection:
 - Good detection: the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)
 - Good localization: the edges detected must be as close as possible to the true edges.
 - Single response constraint: the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge

Edge Detection

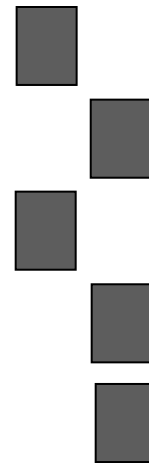
- Examples:



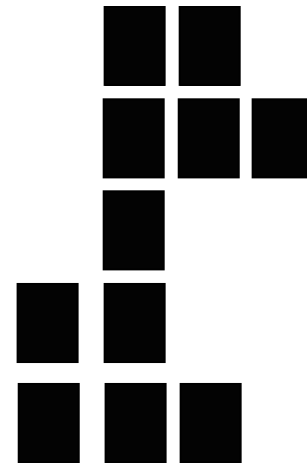
True edge



Poor robustness to noise



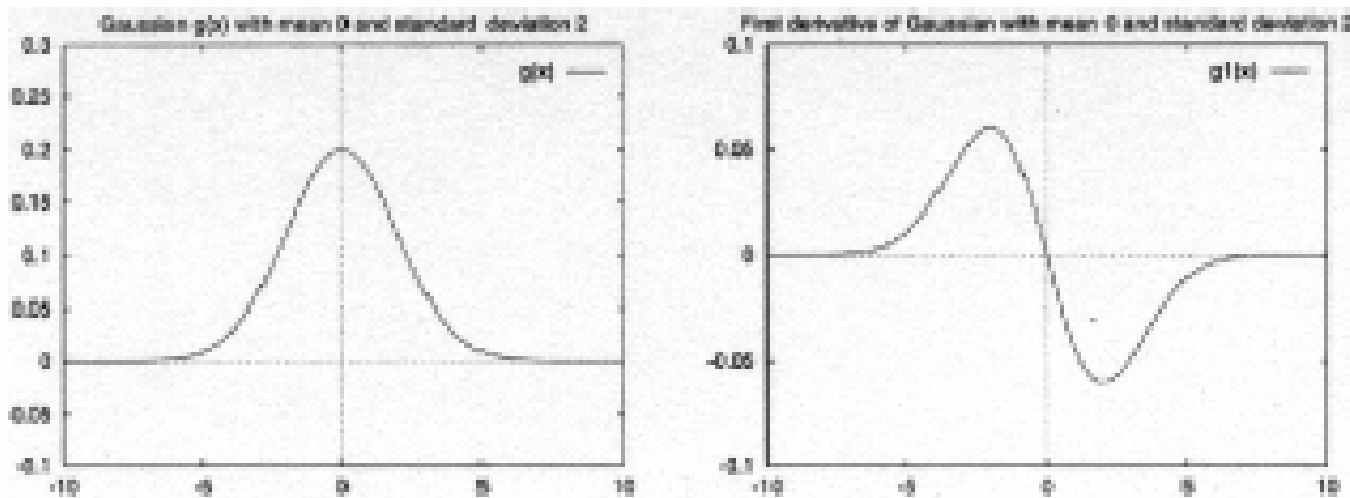
Poor localization



Too many responses

Canny Edge Detection

- The Canny edge detector:
 - This is probably the most widely used edge detector in computer vision.
 - Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of *signal-to-noise* ratio and localization.
 - His analysis is based on "step-edges" corrupted by "additive Gaussian noise".



Canny Edge Detection

Steps:

1. Smooth image w/ Gaussian filter
 2. Apply derivative of Gaussian
 3. Find magnitude and orientation of gradient
 4. 'Non-maximum suppression'
 - Thin multi-pixel wide "ridges" down to single pixel width
 5. 'Hysteresis': Linking and thresholding
 - Low, high edge-strength thresholds
 - Accept all edges over low threshold that are connected to edge over high threshold
- Matlab: `edge(I, 'canny')`

Canny Edge Detector

First Two Steps

- Smoothing

$$S = I * g(x, y) = g(x, y) * I$$

$$g(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Derivative

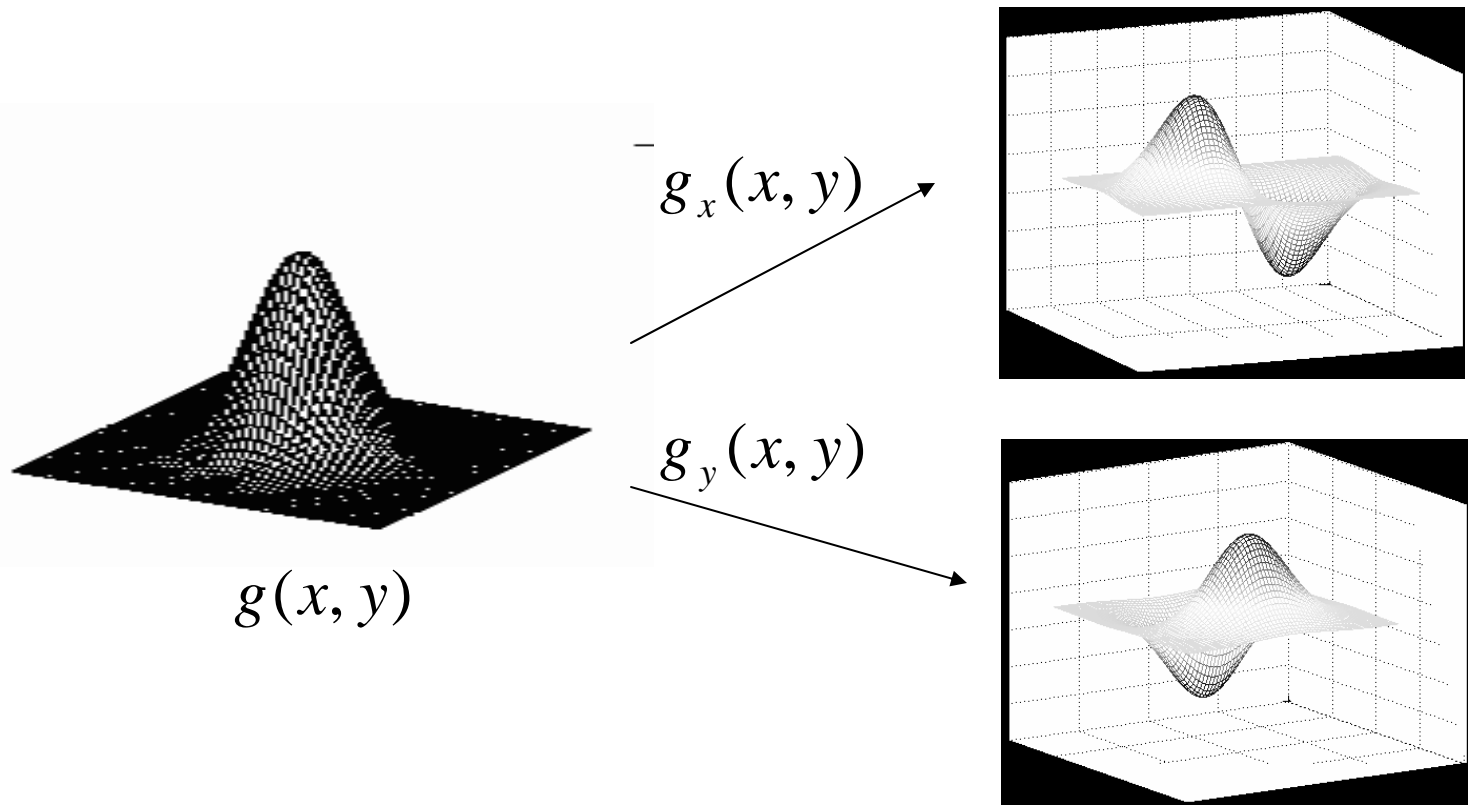
$$\nabla S = \nabla(g * I) = (\nabla g) * I$$

$$\nabla S = \begin{bmatrix} g_x \\ g_y \end{bmatrix} * I = \begin{bmatrix} g_x * I \\ g_y * I \end{bmatrix}$$

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

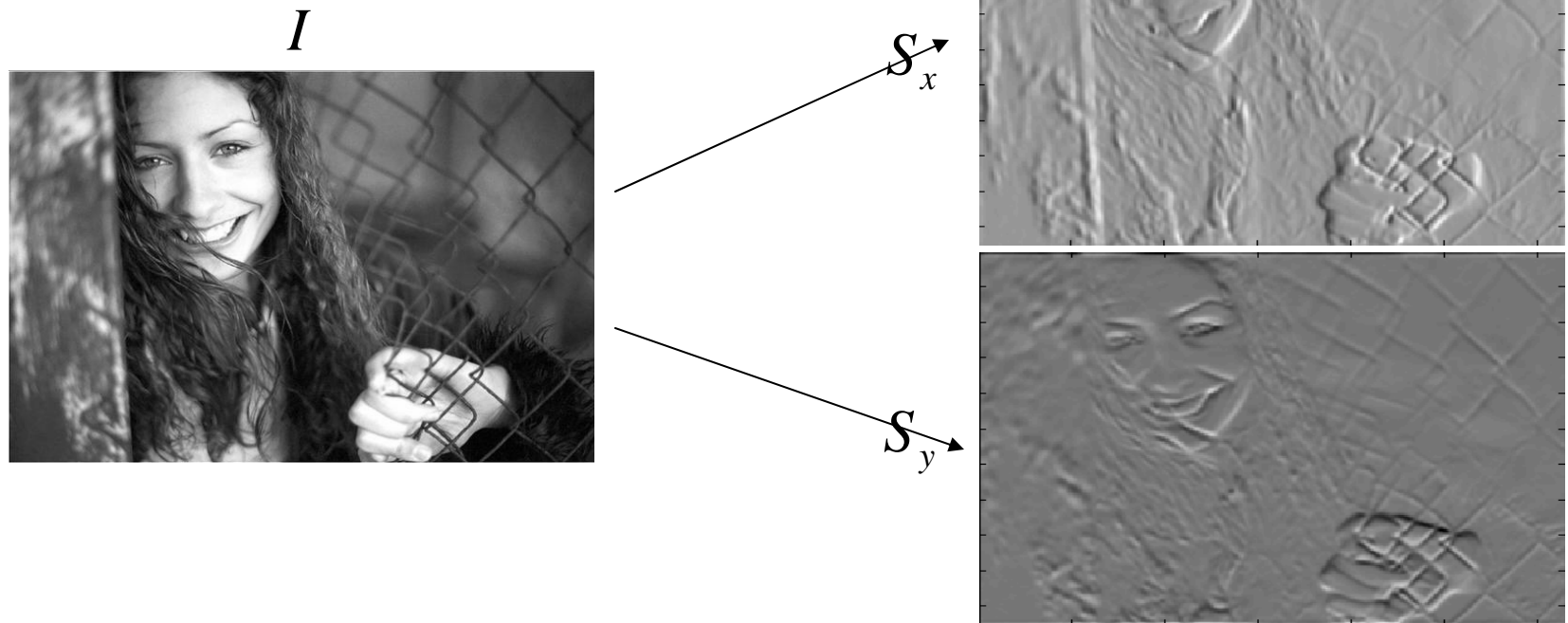
Canny Edge Detector

Derivative of Gaussian



Canny Edge Detector

First Two Steps



Canny Edge Detector

Third Step

- Gradient magnitude and gradient direction

(S_x, S_y) Gradient Vector

$$\text{magnitude} = \sqrt{(S_x^2 + S_y^2)}$$

$$\text{direction} = \theta = \tan^{-1} \frac{S_y}{S_x}$$



image

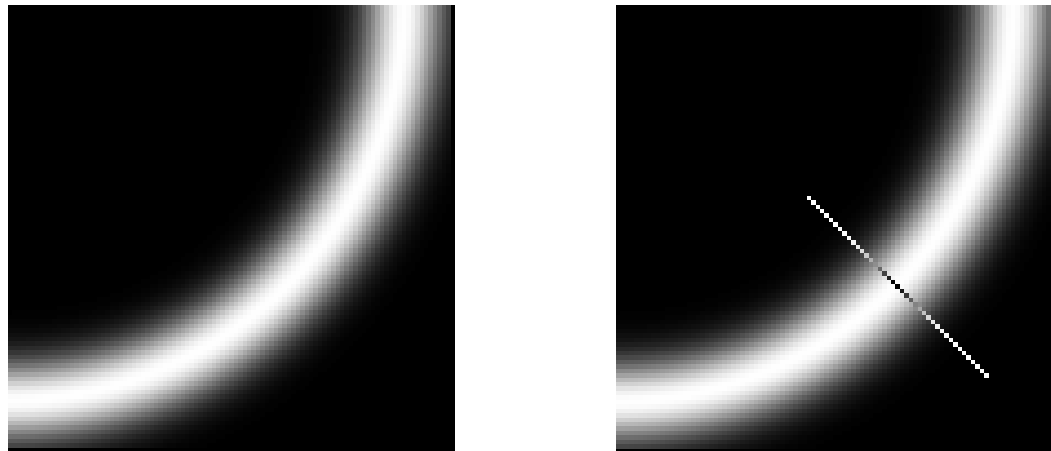


gradient magnitude

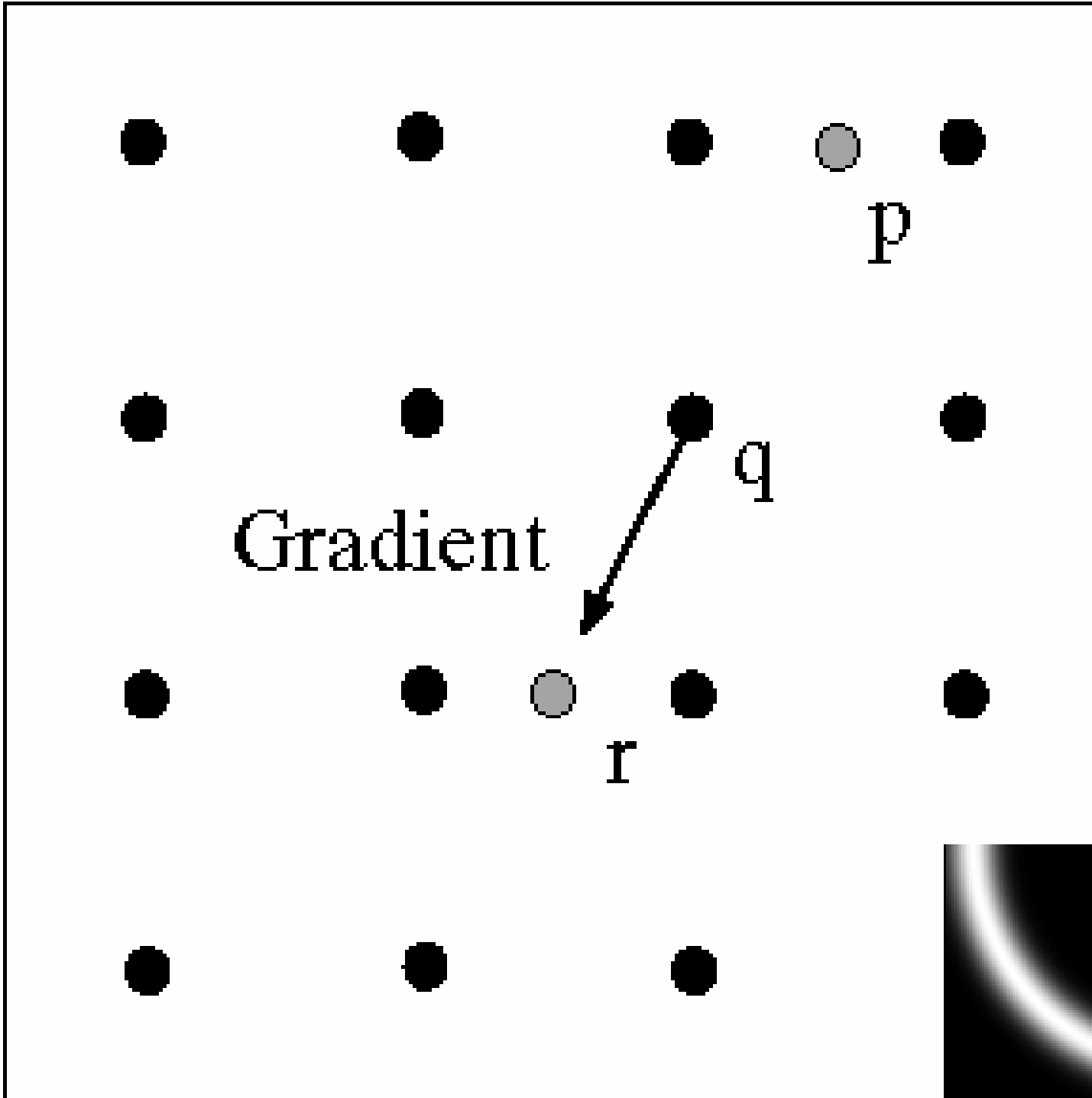
Canny Edge Detector

Fourth Step

- Non maximum suppression

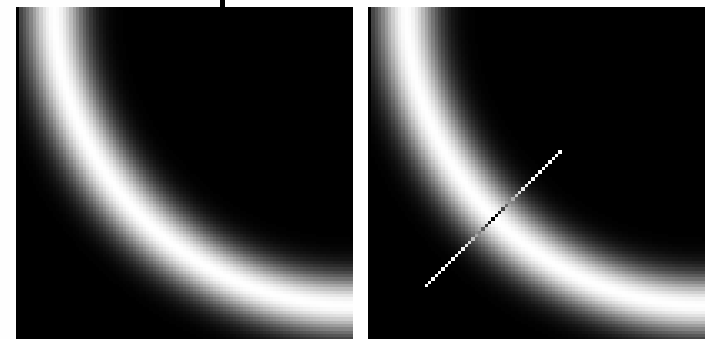


We wish to mark points along the curve where the **magnitude is biggest**. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?



**Non-maximum
suppression**

At q, the
value must
be larger
than values
interpolated
at p or r.



Examples: Non-Maximum Suppression



Original image



Gradient magnitude

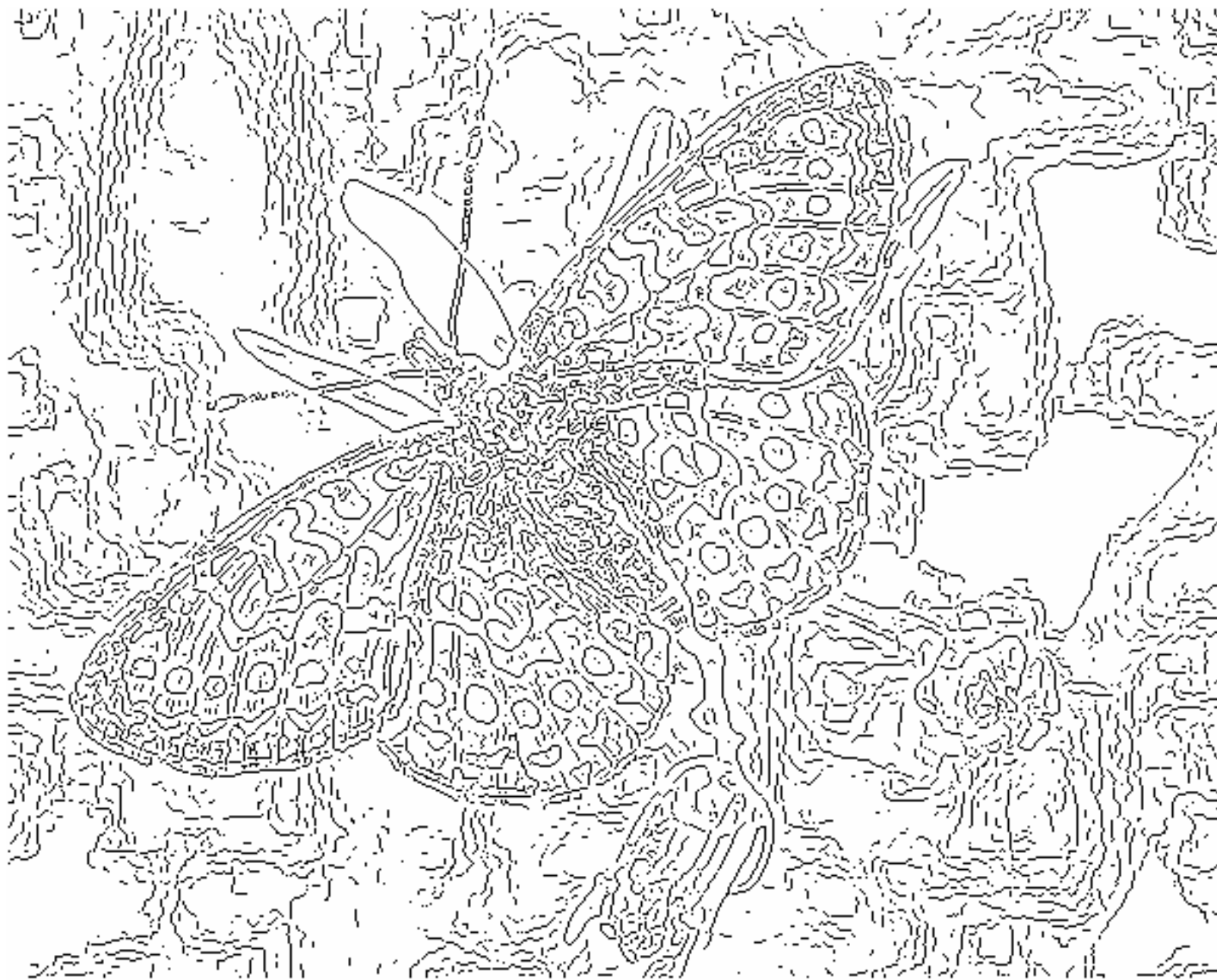


courtesy of G. Loy

Non-maxima
suppressed

Slide credit: Christopher Rasmussen





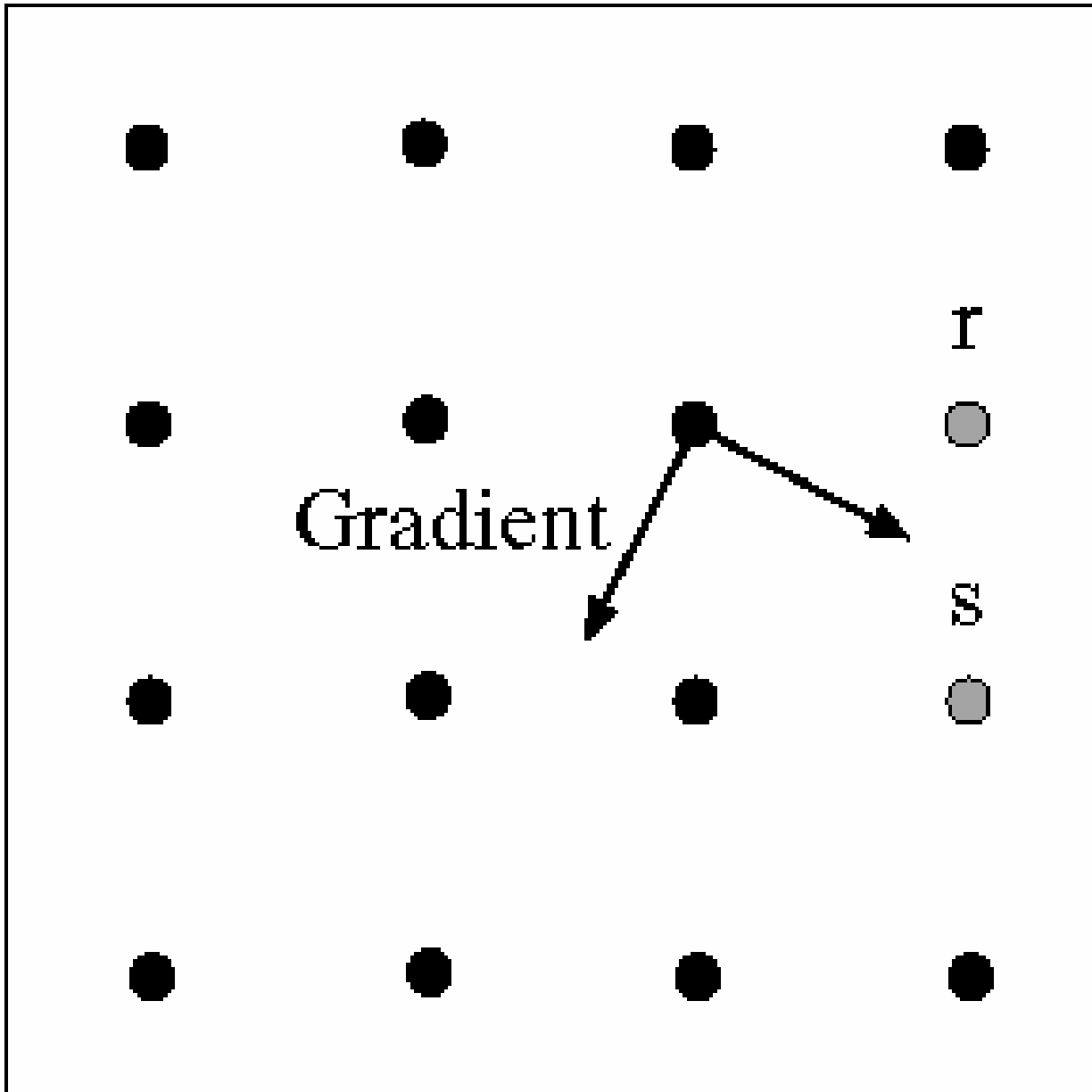
fine scale
high
threshold



coarse
scale,
high
threshold



coarse
scale
low
threshold



Linking to the next edge point

Assume the marked point is an edge point.

Take the normal to the gradient at that point and use this to predict continuation points (either r or s).

Canny Edge Detector

Step 5: Hysteresis Thresholding

- **Hysteresis:** A lag or momentum factor
- **Idea:** Maintain two thresholds k_{high} and k_{low}
 - Use k_{high} to find strong edges to start edge chain
 - Use k_{low} to find weak edges which continue edge chain
- Typical ratio of thresholds is roughly

$$k_{\text{high}} / k_{\text{low}} = 2$$

Example: Canny Edge Detection

Original image



gap is gone



Strong + connected weak edges

Strong edges only



Weak edges



courtesy of G. Loy

Example: Canny Edge Detection



Using Matlab with default thresholds

Demo

<http://www.cs.washington.edu/research/imagedatabase/demo/edge/>