# Lecture 3: Linear Filters

Professor Fei-Fei Li
Stanford Vision Lab

# What we will learn today?

- Images as functions
- Linear systems (filters)
- Convolution and correlation
- Discrete Fourier Transform (DFT)
- Sampling and aliasing

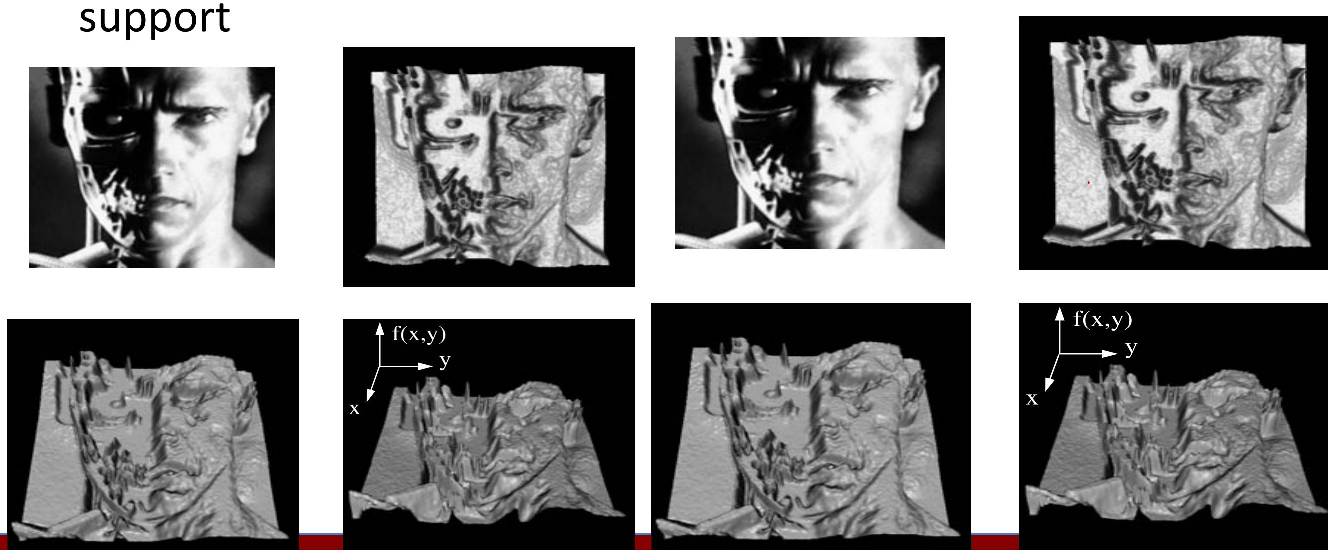
Some background reading:
Forsyth and Ponce, Computer Vision, Chapter 7 & 8
Jae S. Lim, Two-dimensional signal and image processing, Chapter 1, 4, 5

# Images as functions

- **An Image** as a function $f$ from $\mathrm{R}^2$ to $\mathrm{R}^M$:
  - $f(x, y)$ gives the **intensity** at position $(x, y)$ $M=1$
  - Defined over a rectangle, with a finite range:

$$f: \underbrace{[a,b] \times [c,d]}_{\text{Domain support}} \rightarrow \underbrace{[0,255]}_{\text{range}}$$

$R^2 \rightarrow R^1$



# Images as functions

- **An Image** as a function $f$ from $\mathrm{R}^2$ to $\mathrm{R}^M$:
  - $f( x, y )$ gives the **intensity** at position $( x, y )$
  - Defined over a rectangle, with a finite range:

$$f\colon \underbrace{[a,b] \times [c,d\,]}_{\text{Domain support}} \rightarrow \underbrace{[0,255]}_{\text{range}}$$

- A color image: $f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$

$R^2 \longrightarrow R^3$

color: RGB

HSV

Lab

# Images as discrete functions

- Images are usually **digital (discrete)**:
  - **Sample** the 2D space on a regular grid

- Represented as a matrix of integer values

pixel

$j$

$i$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 62 | 79 | 23 | 119 | 120 | 105 | 4 | 0 |
| 10 | 10 | 9 | 62 | 12 | 78 | 34 | 0 |
| 10 | 58 | 197 | 46 | 46 | 0 | 0 | 48 |
| 176 | 135 | 5 | 188 | 191 | 68 | 0 | 49 |
| 2 | 1 | 1 | 29 | 26 | 37 | 0 | 77 |
| 0 | 89 | 144 | 147 | 187 | 102 | 62 | 208 |
| 255 | 252 | 0 | 166 | 123 | 62 | 0 | 31 |
| 166 | 63 | 127 | 17 | 1 | 0 | 99 | 30 |

# Images as discrete functions

Cartesian coordinates

$$f[n,m] = \begin{bmatrix} \ddots & & \vdots & & \\ & f[-1,1] & f[0,1] & f[1,1] & \\ \ldots & f[-1,0] & \underline{f[0,0]} & f[1,0] & \ldots \\ & f[-1,-1] & f[0,-1] & f[1,-1] & \\ & & \vdots & & \ddots \end{bmatrix}$$

Notation for discrete functions

# Images as discrete functions

Array coordinates

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & \dots & a_{1M} \\ \vdots & \ddots & \vdots \\ a_{N1} & \dots & a_{NM} \end{bmatrix}$$

Matlab notation

# What we will learn today?

- Images as functions
- Linear systems (filters)
- Convolution and correlation
- Discrete Fourier Transform (DFT)
- Sampling and aliasing

Some background reading:
Forsyth and Ponce, Computer Vision, Chapter 7 & 8
Jae S. Lim, Two-dimensional signal and image processing, Chapter 1, 4, 5

# Systems and Filters

- **Filtering:**
  – Form a new image whose pixels are a combination original pixel values

**Goals:**

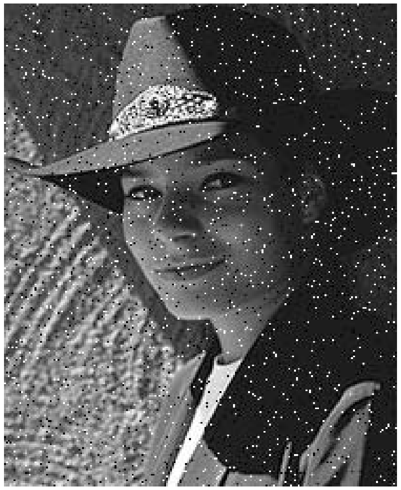
-Extract useful information from the images

- Features (edges, corners, blobs…)

- Modify or enhance image properties:

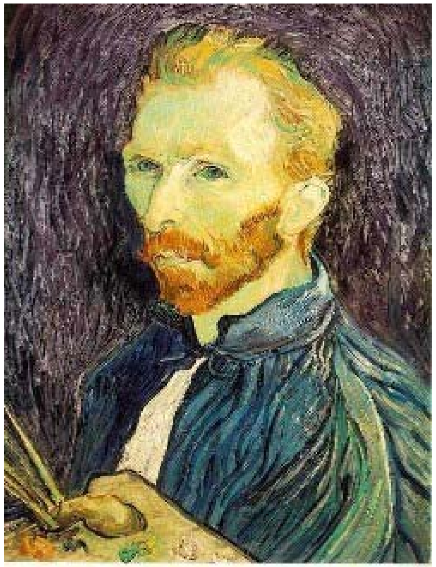
- super-resolution; in-painting; de-noising

## De-noising



Salt and pepper noise

## Super-resolution



## In-painting

Bertamio et al

# 2D discrete-space systems (filters)

$$f[n,m] \rightarrow \boxed{\text{System } \mathcal{S}} \rightarrow g[n,m]$$

$$g = \mathcal{S}[f], \quad g[n,m] = \mathcal{S}\{f[n,m]\}$$

$$f[n,m] \xrightarrow{\mathcal{S}} g[n,m]$$

# Filters: Examples

- 2D DS moving average over a 3 × 3 window of neighborhood

3x3 window

orig. im

filtered im

$$g[n,m] = \frac{1}{9} \sum_{k=n-1}^{n+1} \sum_{l=m-1}^{m+1} f[k,l]$$

normaliz'g const

$$= \frac{1}{9} \sum_{k=-1}^{1} \sum_{l=-1}^{1} f[n-k, m-l]$$

filter

$$h = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$(f * h)[m,n] = \frac{1}{9} \sum_{k,l} f[k,l]\, h[m-k, n-l]$$

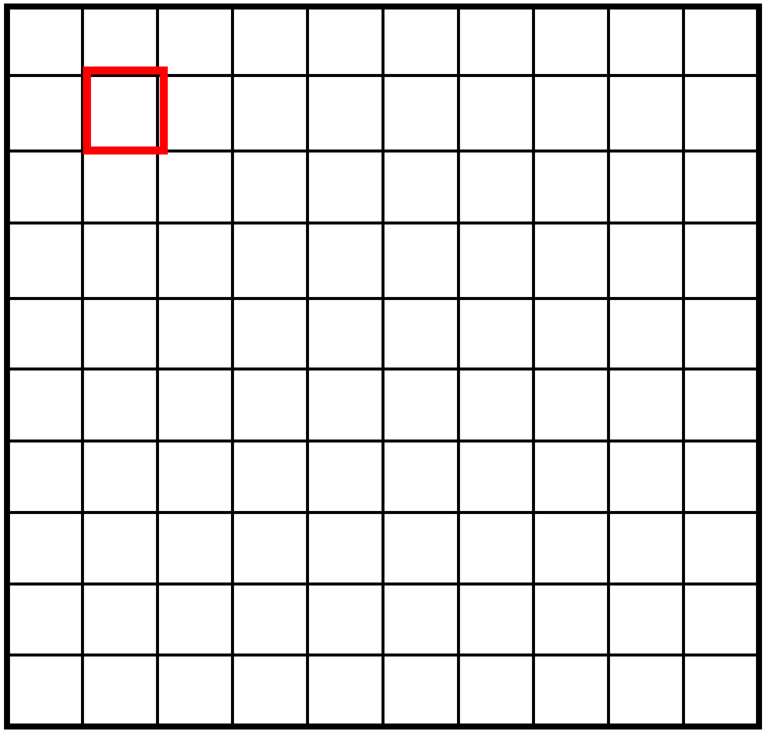
convolution

# Moving average

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$



Courtesy of S. Seitz

# Moving average

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

$$(f * g)[m, n] = \sum_{k,l} f[k, l]\, g[m-k, n-l]$$

# Moving average

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

$$(f * g)[m,n] = \sum_{k,l} f[k,l]\, g[m-k, n-l]$$

# Moving average

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

$$(f * g)[m,n] = \sum_{k,l} f[k,l]\, g[m-k, n-l]$$

# Moving average

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

$$(f * g)[m,n] = \sum_{k,l} f[k,l]\, g[m-k, n-l]$$

# Moving average

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$ “blur” or “smooth”

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | 30 | 20 | 10 | |
| | 0 | 20 | 40 | 60 | 60 | 60 | 40 | 20 | |
| | 0 | 30 | 60 | 90 | 90 | 90 | 60 | 30 | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | |
| | 0 | 20 | 30 | 50 | 50 | 60 | 40 | 20 | |
| | 10 | 20 | 30 | 30 | 30 | 30 | 20 | 10 | |
| | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | | | | |

$$(f * g)[m,n] = \sum_{k,l} f[k,l]\, g[m-k, n-l]$$

Source: S. Seitz

# Moving average

In summary:

- Replaces each pixel with an average of its neighborhood.

- Achieve smoothing effect (remove sharp features)

$$g[\cdot\,,\cdot\,] = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

# Moving average

# Filters: Examples

- Image segmentation based on a simple threshold:

$$g[n,m] = \begin{cases} 1, & f[n,m] > 10 \\ 0, & \text{otherwise.} \end{cases}$$

# Classification of systems

- Amplitude properties
  - Linearity
  - Stability
  - Invertibility
- Spatial properties
  - Causality
  - Separability
  - Memory
  - Shift invariance
  - Rotation invariance

filter

$\underbrace{\text{Linear Shift-Invariant}}_{\text{LSI}}$ Systems

# Shift-invariance

- If $f[n,m] \xrightarrow{\mathcal{S}} g[n,m]$ then

$$\boxed{f[n-n_0, m-m_0] \xrightarrow{\mathcal{S}} g[n-n_0, m-m_0]}$$

for every input image f[n,m] and shifts $n_0$,$m_0$

- Is the moving average shift invariant a system ?

# Is the moving average system is shift invariant?

$F[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

$G[x, y]$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | 30 | 20 | 10 | |
| | 0 | 20 | 40 | 60 | 60 | 60 | 40 | 20 | |
| | 0 | 30 | 60 | 90 | 90 | 90 | 60 | 30 | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | |
| | 0 | 20 | 30 | 50 | 50 | 60 | 40 | 20 | |
| | 10 | 20 | 30 | 30 | 30 | 30 | 20 | 10 | |
| | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | | | | |

# Is the moving average system is shift invariant?

$$f[n,m] \quad \xrightarrow{\mathcal{S}} \quad g[n,m] = \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[n-k, m-l]$$

$$f[n-n_0, m-m_0]$$

$$\xrightarrow{\mathcal{S}} \quad g[n,m] = \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[n-k, m-l]$$

$$= \quad \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[(n-n_0)-k, (m-m_0)-l]$$

$$= \quad g[n-n_0, m-m_0]$$

Yes!

# Linear Systems (filters)

$$f(x,y) \rightarrow \boxed{\mathcal{S}} \rightarrow g(x,y)$$

- Linear filtering:
  - Form a new image whose pixels are a weighted sum of original pixel values
  - Use the same set of weights at each point
- **S** is a linear system (function) iff it *S satisfies*

$$\mathcal{S}[\alpha f_1 + \beta f_2] = \alpha \mathcal{S}[f_1] + \beta \mathcal{S}[f_2]$$

superposition property

# Linear Systems (filters)

$$f(x,y) \rightarrow \boxed{\mathcal{S}} \rightarrow g(x,y)$$

- Is the moving average a linear system?

$$f(n,m) \xrightarrow{S} g(n,m) = \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f(n-k, m-l)$$

$$S(\alpha f) = \frac{1}{9}\sum\sum \alpha f(n-k, m-l)$$

$$= \alpha \cdot \left[\frac{1}{9}\sum\sum f(n-k, m-l)\right]$$

$$= \alpha \cdot S(f)$$

- Is thresholding a linear system?

$$g(n,m) = \begin{cases} 1 & f(n,m) > T \\ 0 & \text{otherwise} \end{cases}$$

  – f1[n,m] + f2[n,m] > T
  – f1[n,m] < T
  – f2[n,m]<T

No!

# LSI (linear *shift invariant*) systems

delta func. $\delta[n,m] = \begin{cases} 1 & n=m=0 \\ 0 & \text{otherwise} \end{cases}$

**Impulse response**

$$\delta_2[n,m] \to \boxed{\mathcal{S}} \to h[n,m]$$

$$\delta_2[n-k,m-l] \to \boxed{\mathcal{S}\ (\mathrm{SI})} \to h[n-k,m-l]$$

# LSI (linear *shift invariant*) systems

***Example:*** impulse response of the 3 by 3 moving average filter:

$$h[n,m] = \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1}\delta_2[n-k,m-l]$$

$$= \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

h

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

# LSI (linear *shift invariant*) systems

**An LSI system is completely specified by its impulse response.**

sifting property of the delta function

$$f[n,m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l]\,\delta_2[n-k, m-l]$$

superposition

$$\rightarrow \boxed{\mathcal{S}\ \text{LSI}} \rightarrow \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l]\,h[n-k, m-l]$$

$$\delta_2[n,m] \rightarrow \boxed{\mathcal{S}} \rightarrow h[n,m]$$

Discrete convolution

$$= f[n,m] ** h[n,m]$$

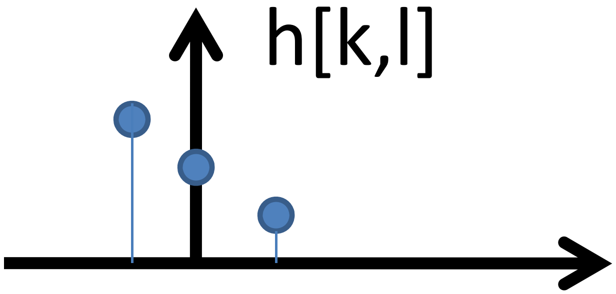
# What we will learn today?

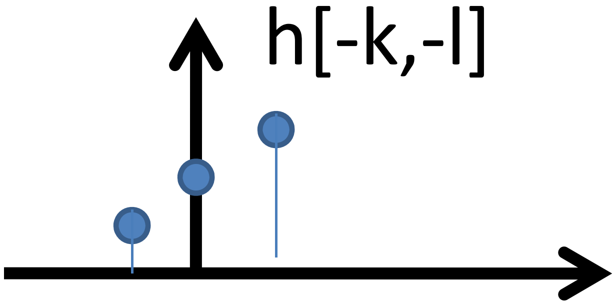
- Images as functions
- Linear systems (filters)
- Convolution and correlation
- Discrete Fourier Transform (DFT)
- Sampling and aliasing

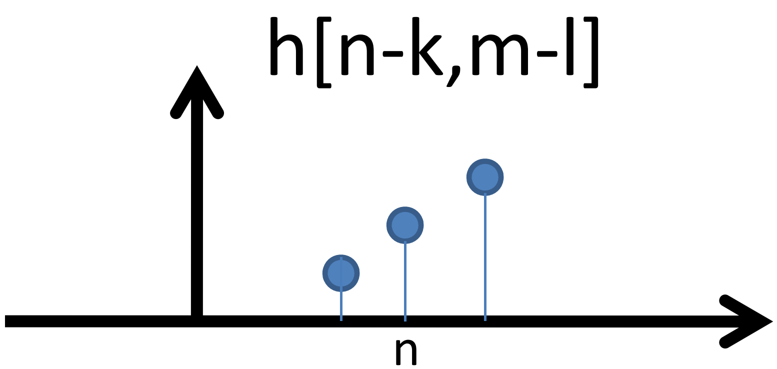
Some background reading:
Forsyth and Ponce, Computer Vision, Chapter 7 & 8
Jae S. Lim, Two-dimensional signal and image processing, Chapter 1, 4, 5

# Discrete convolution

- Fold h[n,m] about origin to form h[−k,−l]
- Shift the folded results by n,m to form h[n − k,m − l]
- Multiply h[n − k,m − l] by f[k, l]
- Sum over all k,l
- Repeat for every n,m

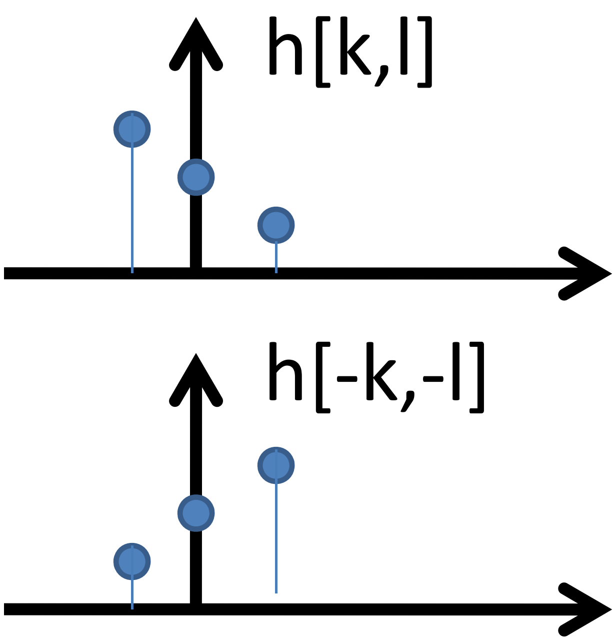


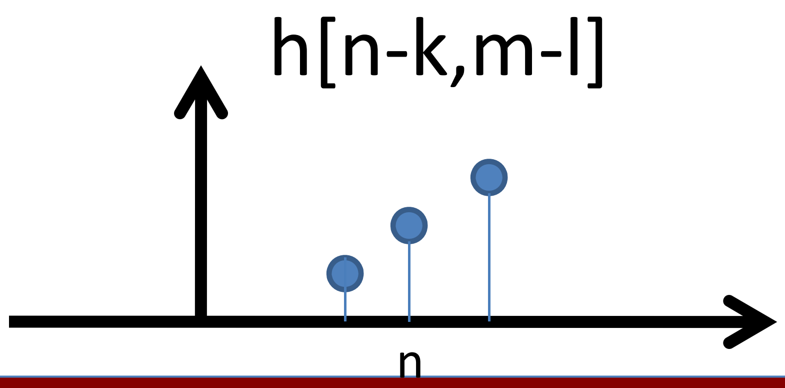



# Discrete convolution

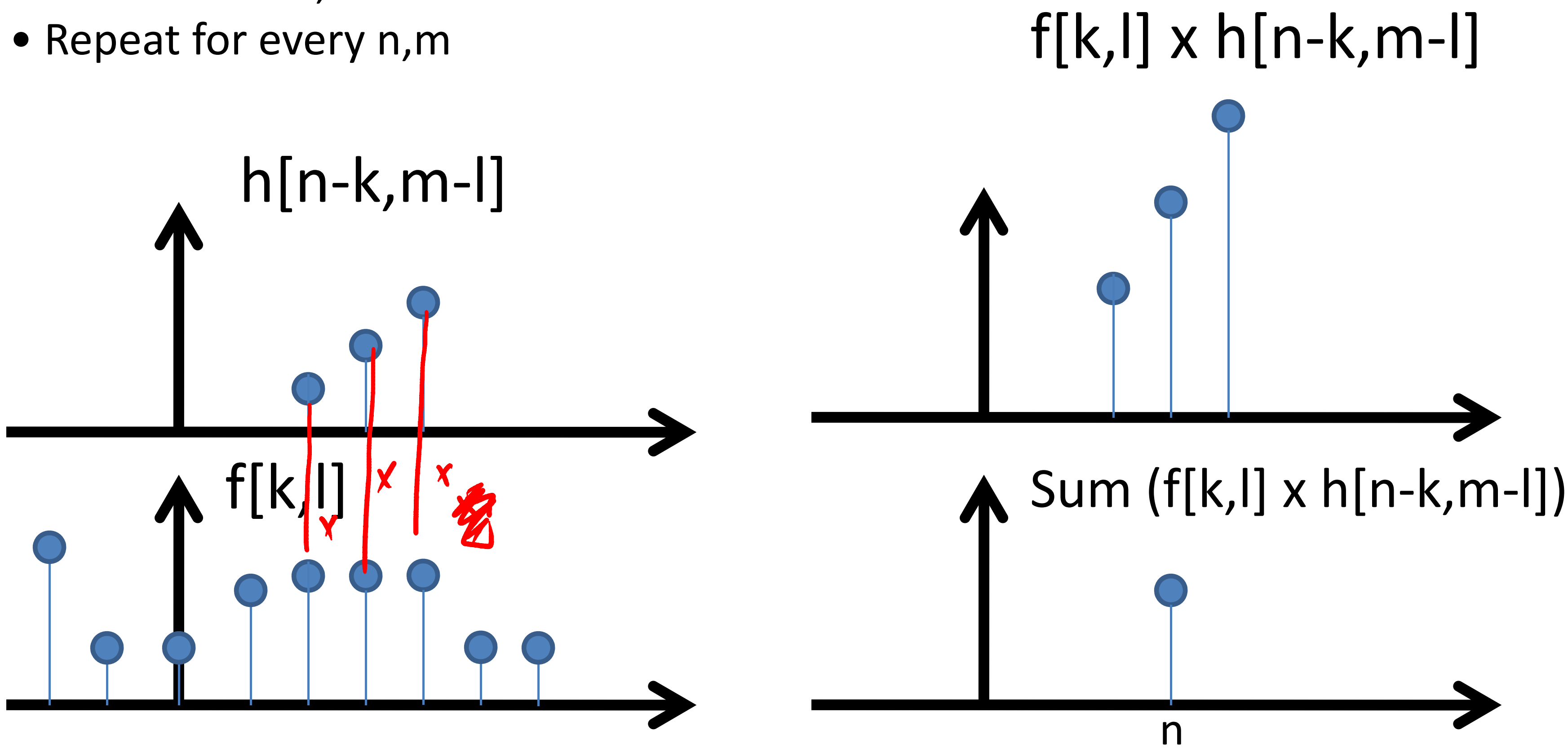
- Fold h[n,m] about origin to form h[−k,−l]
- Shift the folded results by n,m to form h[n − k,m − l]
- Multiply h[n − k,m − l] by f[k, l]
- Sum over all k,l
- Repeat for every n,m



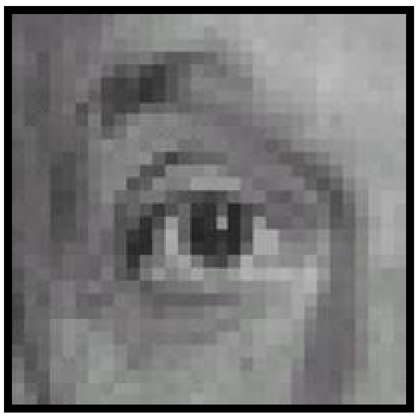



# Discrete convolution

- Fold h[n,m] about origin to form h[−k,−l]
- Shift the folded results by n,m to form h[n − k,m − l]
- Multiply h[n − k,m − l] by f[k, l]
- Sum over all k,l
- Repeat for every n,m



# Convolution in 2D - examples

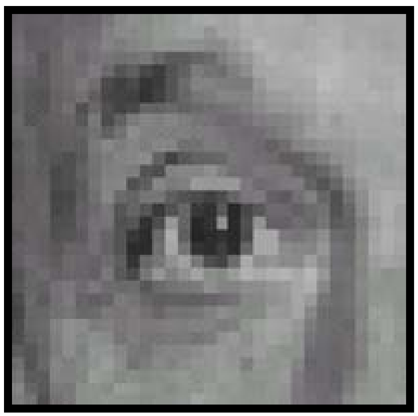


Original

$$** \quad \overset{h}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}} \quad = \quad ?$$

Courtesy of D Lowe

# Convolution in 2D - examples



Original

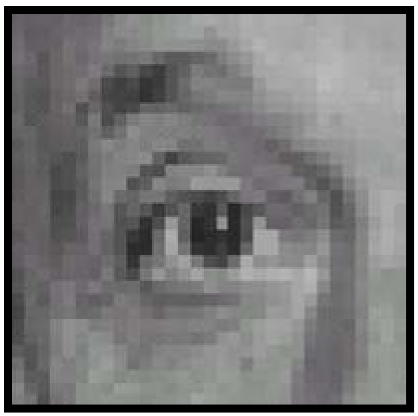
| | | |
|---|---|---|
| •0 | •0 | •0 |
| •0 | •1 | •0 |
| •0 | •0 | •0 |

=

Filtered
(no change)

Courtesy of D Lowe

# Convolution in 2D - examples



Original

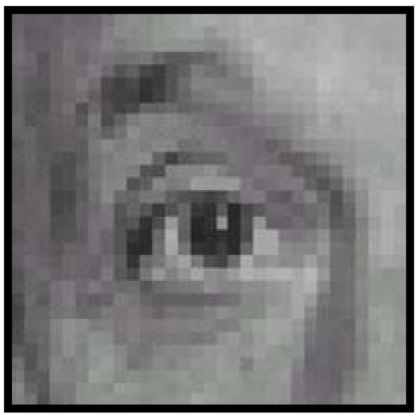
| | | |
|---|---|---|
| •0 | •0 | •0 |
| •0 | •0 | •1 |
| •0 | •0 | •0 |

=

?

Courtesy of D Lowe

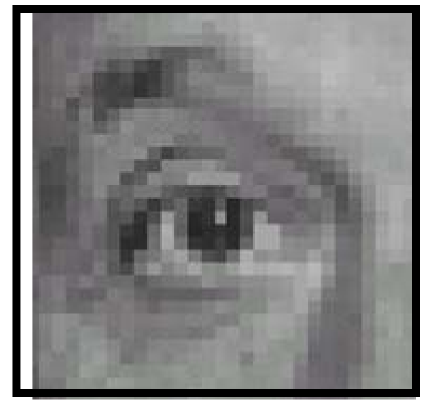
# Convolution in 2D - examples



Original

| | | |
|---|---|---|
| •0 | •0 | •0 |
| •0 | •0 | •1 |
| •0 | •0 | •0 |

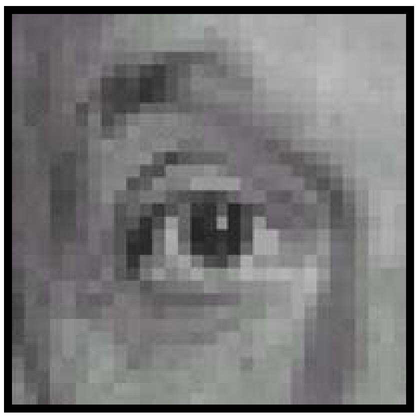
=



Shifted right
By 1 pixel

Courtesy of D Lowe

# Convolution in 2D - examples



Original

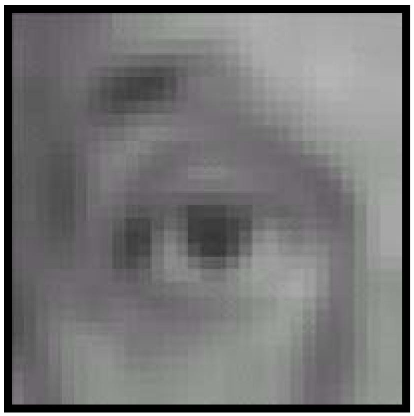
$$\frac{1}{9}\begin{array}{|c|c|c|}\hline \bullet 1 & \bullet 1 & \bullet 1 \\ \hline \bullet 1 & \bullet 1 & \bullet 1 \\ \hline \bullet 1 & \bullet 1 & \bullet 1 \\ \hline \end{array} = \textbf{?}$$

Courtesy of D Lowe

# Convolution in 2D - examples

Original

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



Blur (with a box filter)

Courtesy of D Lowe

# Convolution in 2D - examples

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

(Note that filter sums to 1)

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\text{unchanged}} + \overbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}}^{\text{“details of the image”}}$$

Courtesy of D Lowe

- What does blurring take away?


−

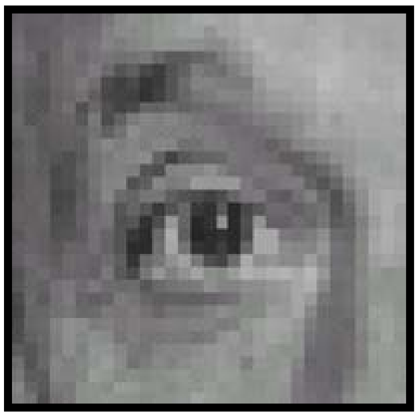

=


  - Let’s add it back:


+ a

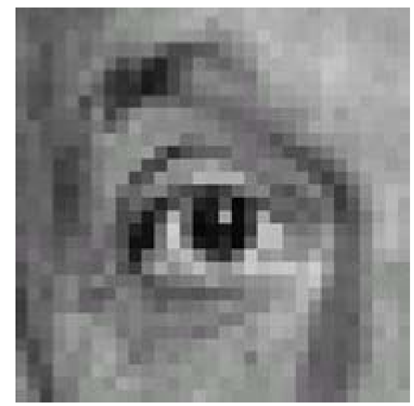

=

# Convolution in 2D – Sharpening filter



Original

$$* \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$



Courtesy of D Lowe

**Sharpening filter:** Accentuates differences with local average

# Convolution properties

- **Commutative property:**

$$f ** h = h ** f$$

- **Associative property:**

$$(f ** h_1) ** h_2 = f ** (h_1 ** h_2)$$

- **Distributive property:**

$$f ** (h_1 + h_2) = (f ** h_1) + (f ** h_2)$$

The order doesn't matter! $h_1 ** h_2 = h_2 ** h_1$

# Convolution properties

- **Shift property:**

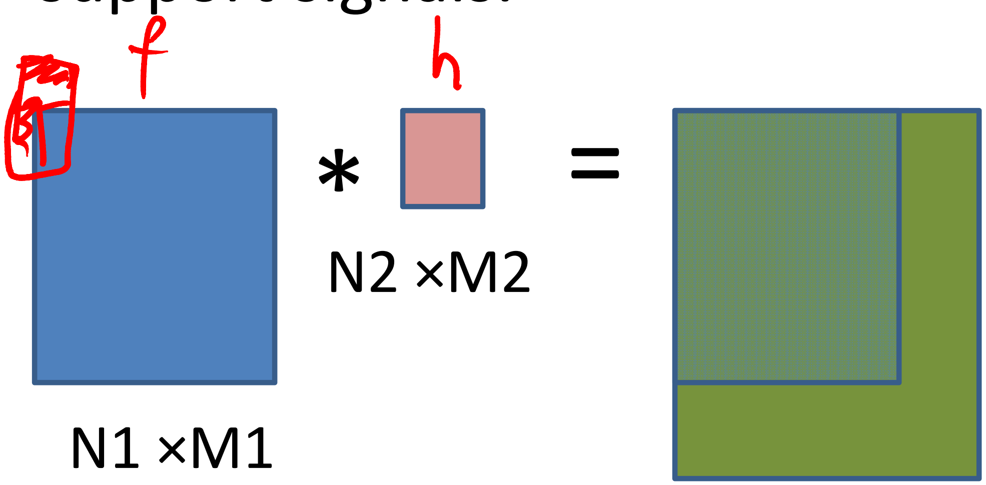
$$f[n,m] ** \delta_2[n-n_0, m-m_0] = f[n-n_0, m-m_0]$$

- **Shift-invariance:**

$$\begin{aligned} g[n,m] &= f[n,m] ** h[n,m] \\ &\implies f[n-l_1, m-l_1] ** h[n-l_2, m-l_2] \\ &\qquad = g[n-l_1-l_2, m-l_1-l_2] \end{aligned}$$

# Image support and edge effect

- A computer will only convolve **finite support signals.**
  - That is: images that are zero for n,m outside some rectangular region
- MATLAB's conv2 performs 2D DS convolution of finite-support signals.

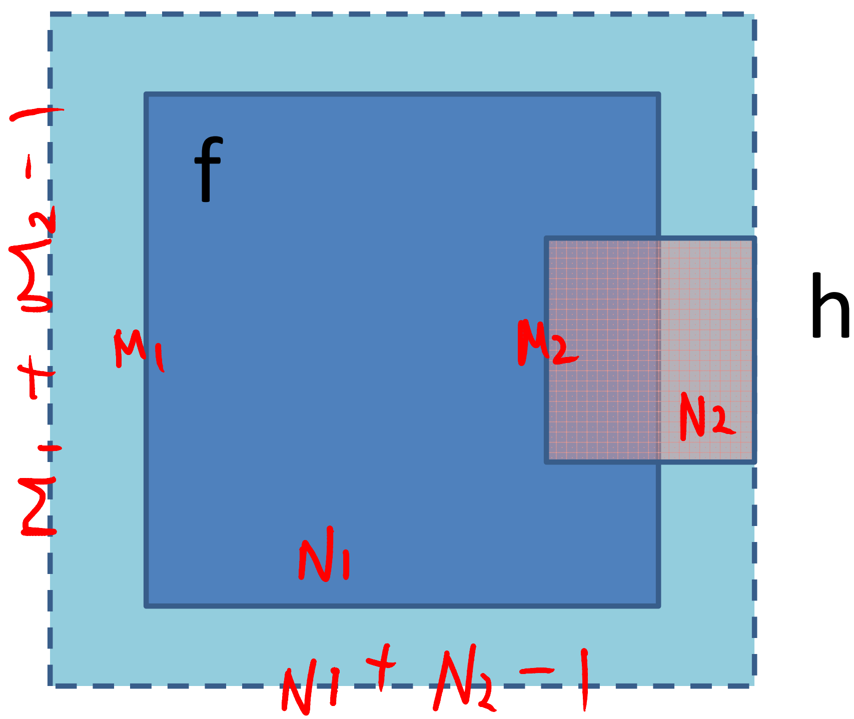



$(N1 + N2 - 1) \times (M1 + M2 - 1)$

# Image support and edge effect

•A computer will only convolve **finite support signals.**

• What happens at the edge?




• zero "padding"
• edge value replication
• mirror extension
• more (beyond the scope of this class)

-> Matlab conv2 uses zero-padding

# Cross correlation

Cross correlation of two 2D signals f[n,m] and g[n,m]

$$r_{fg}[k,l] \triangleq \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n,m]\, g^*[n-k, m-l]$$

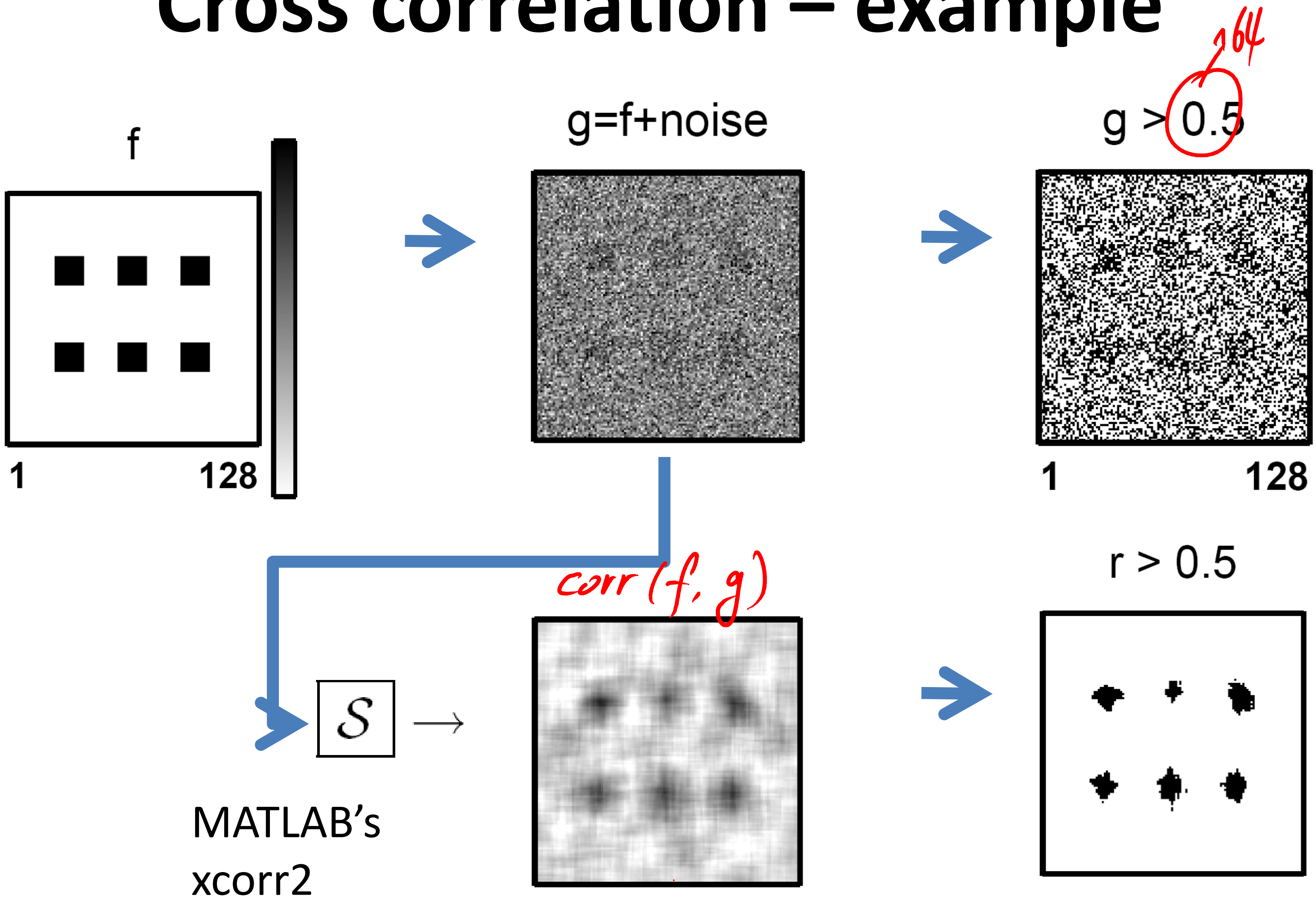
$$= \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} f[n+k, m+l]\, g^*[n,m], \quad k,l \in \mathbb{Z},$$

(k, l) is called the **lag**

- Equivalent to a convolution without the flip

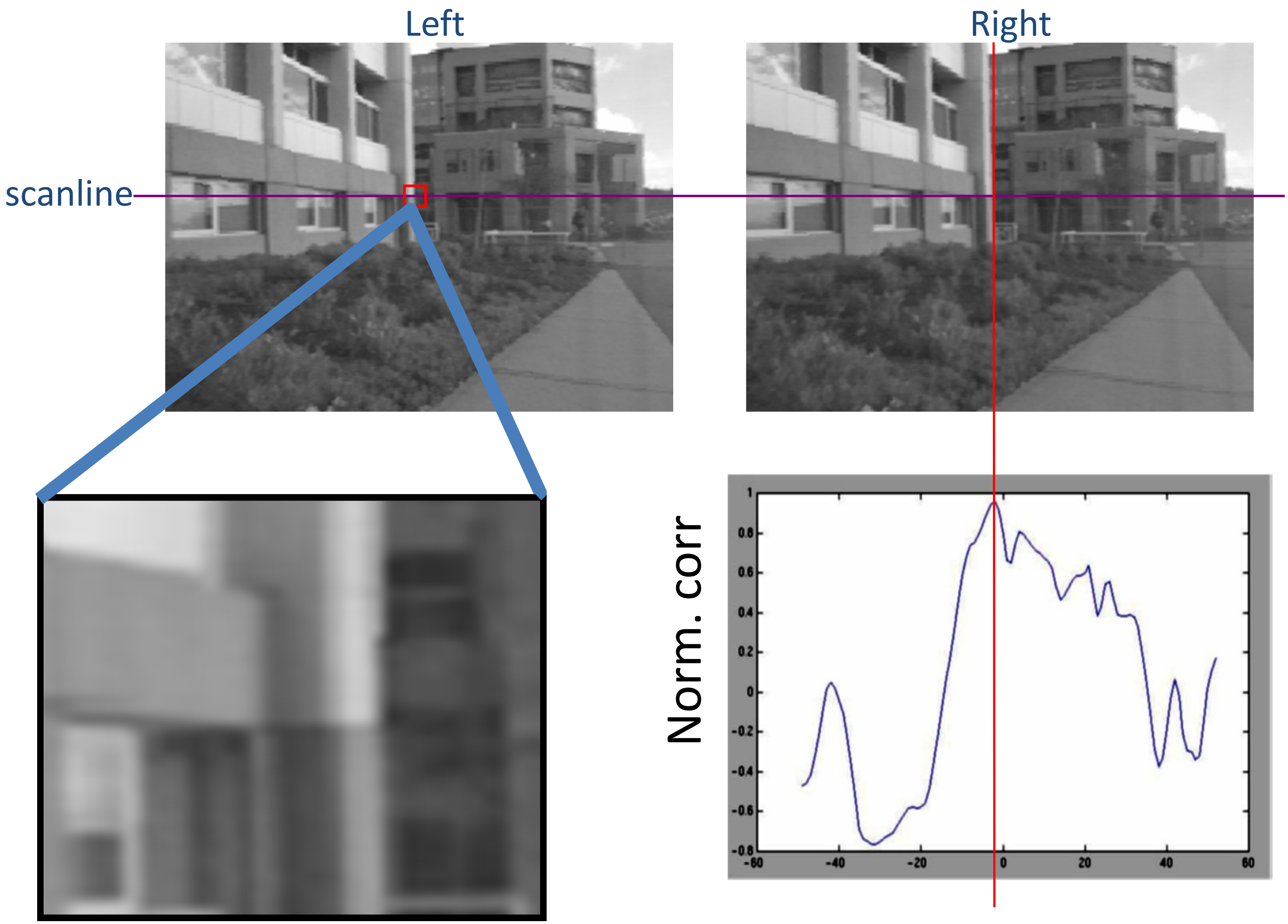
$$r_{fg}[n,m] = f[n,m] ** g^*[-n,-m]$$

# Cross correlation – example




Courtesy of J. Fessler

# Cross correlation – example



# Convolution vs. Correlation

- A **<u>convolution</u>** is an integral that expresses the amount of overlap of one function as it is shifted over another function.
  - convolution is a filtering operation

- **<u>Correlation</u>** compares the ***similarity*** *of* ***two*** *sets of* ***data.*** Correlation computes a measure of similarity of two input signals as they are shifted by one another. The correlation result reaches a maximum at the time when the two signals match best .
  - correlation is a measure of relatedness of two signals

# What we will learn today?

- Images as functions
- Linear systems (filters)
- Convolution and correlation
- Discrete Fourier Transform (DFT)
- Sampling and aliasing

Some background reading:
Forsyth and Ponce, Computer Vision, Chapter 7 & 8
Jae S. Lim, Two-dimensional signal and image processing, Chapter 1, 4, 5

# 2D Discrete-Signal Fourier Transform (DSFT)

2D discrete-signal Fourier transform (DSFT) of a 2D discrete-space signal g[n,m]:

frequency space

signal space

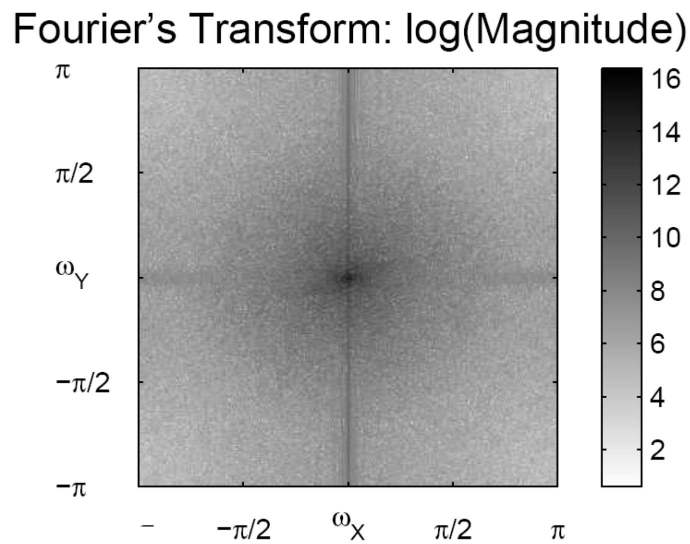
$$G(\omega_X, \omega_Y) = \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} g[n,m]\, e^{-\imath(\omega_X n + \omega_Y m)}$$

Inverse 2D DSFT:

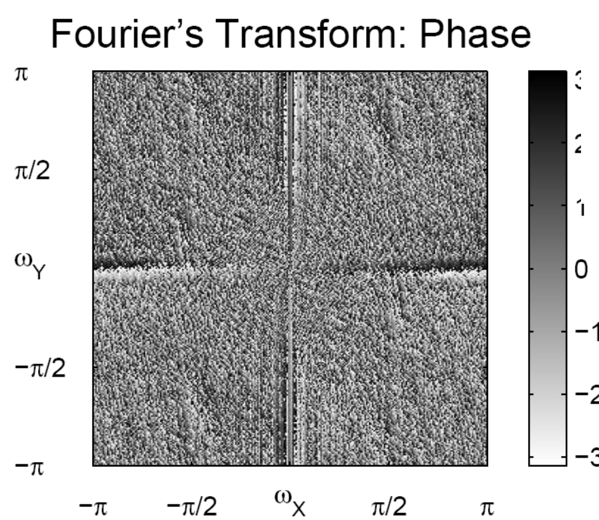
$$g[n,m] = \frac{1}{(2\pi)^2} \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} G(\omega_X, \omega_Y)\, e^{\imath(\omega_X n + \omega_Y m)}\, d\omega_X\, d\omega_Y$$

For memory refresher: Forsyth and Ponce, Ch 8


DSFT $\longleftrightarrow$




$|G(\omega_X, \omega_Y)|$




$e^{\imath \angle G(\omega_X, \omega_Y)}$

# DSFT – properties

**Shift:**

original FFT(g) phase shifted

$$g[n - n_0, m - m_0] \longleftrightarrow G(\omega_X, \omega_Y)\, e^{-\imath(\omega_X n_0 + \omega_Y m_0)}$$

**Convolution:**

$$f[n, m] ** h[n, m] \longleftrightarrow F(\omega_X, \omega_Y)\, H(\omega_X, \omega_Y)$$

**Delta function:**

$$\delta_2[n, m] \longleftrightarrow 1$$

$$\delta_2[n - n_0, m - m_0] \longleftrightarrow e^{-\imath(\omega_X n_0 + \omega_Y m_0)}$$

# Example: DSFT of moving average filter

$$g[n,m] = \frac{1}{9}\sum_{k=-1}^{1}\sum_{l=-1}^{1} f[n-k, m-l]$$

$$(f*h)[m,n] = \frac{1}{9}\sum_{k,l} f[k,l]\, h[m-k, n-l]$$

h

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

$$F(\omega_X, \omega_Y)\, H(\omega_X, \omega_Y)$$

$$= \frac{1}{9}\sum_{n=-1}^{1}\sum_{m=-1}^{1} e^{-\imath\omega_X n}\, e^{-\imath\omega_Y m}$$

$$= \frac{1}{9}[1 + 2\cos\omega_X][1 + 2\cos\omega_Y]$$

orig def!

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

# Why is DFT important?

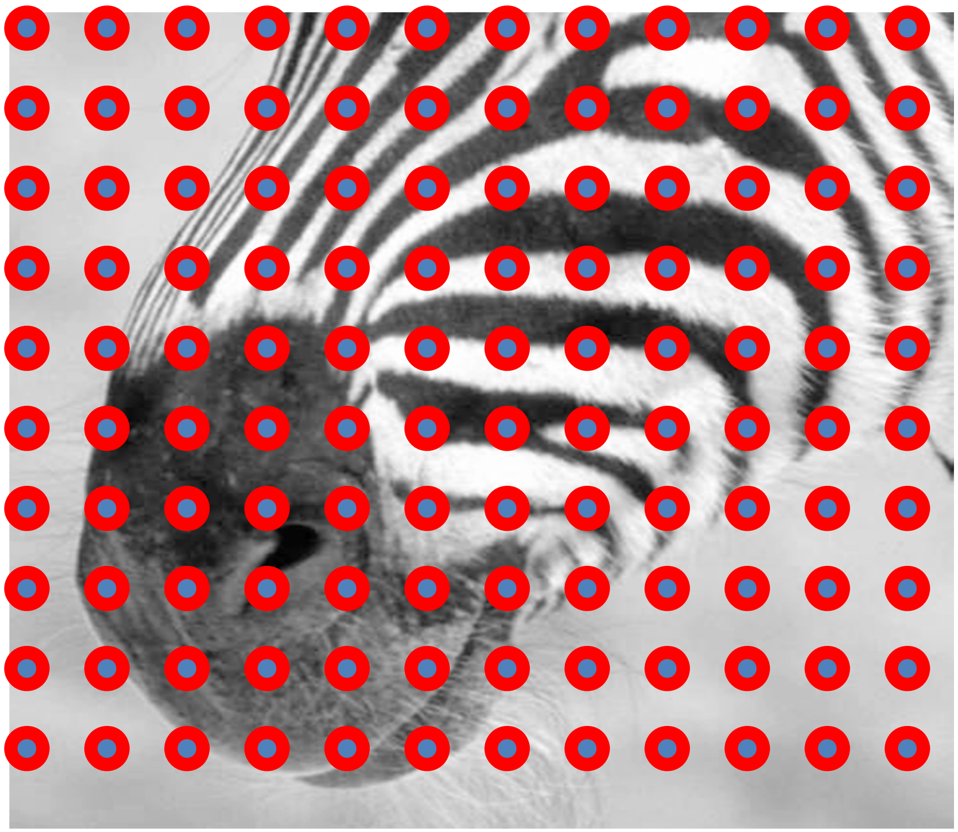
- Perform efficient linear convolution as product of DFTs
- Each DFT can be implemented using the FFT (Fast Fourier Transform ) [see appendix for details]

# What we will learn today?

- Images as functions
- Linear systems (filters)
- Convolution and correlation
- Discrete Fourier Transform (DFT)
- Sampling and aliasing

Some background reading:
Forsyth and Ponce, Computer Vision, Chapter 7 & 8
Jae S. Lim, Two-dimensional signal and image processing, Chapter 1, 4, 5

# Sampling



Throw away every other row and column
to create a 1/2 size image

# Sampling

- Down-sampling operation:

(trivial form of image compression)

$$
g[n,m] = f[2n,2m] = \begin{bmatrix}
\ddots & & \vdots & & \\
 & f[-2,2] & f[0,2] & f[2,2] & \\
\dots & f[-2,0] & f[0,0] & f[2,0] & \dots \\
 & f[-2,-2] & f[0,-2] & f[2,-2] & \\
 & & \vdots & & \ddots
\end{bmatrix}
$$

# Why is a multi-scale representation useful?

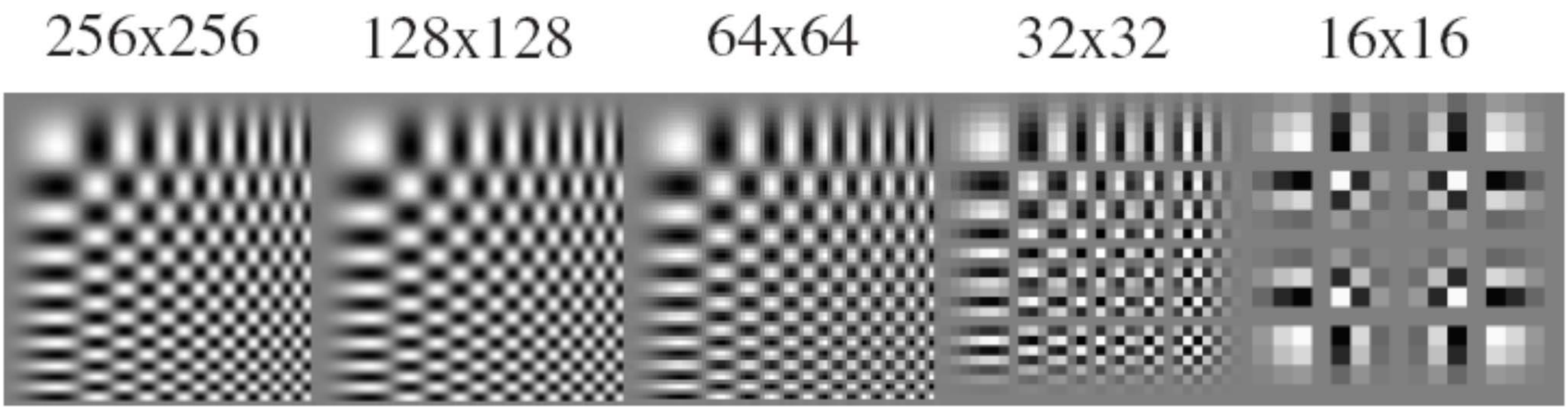
- Find template matches at all scales
  - e.g., when finding hands or faces, we don't know what size they will be in a particular image
  - Template size is constant, but image size changes

- Efficient search for correspondence
  - look at coarse scales, then refine with finer scales

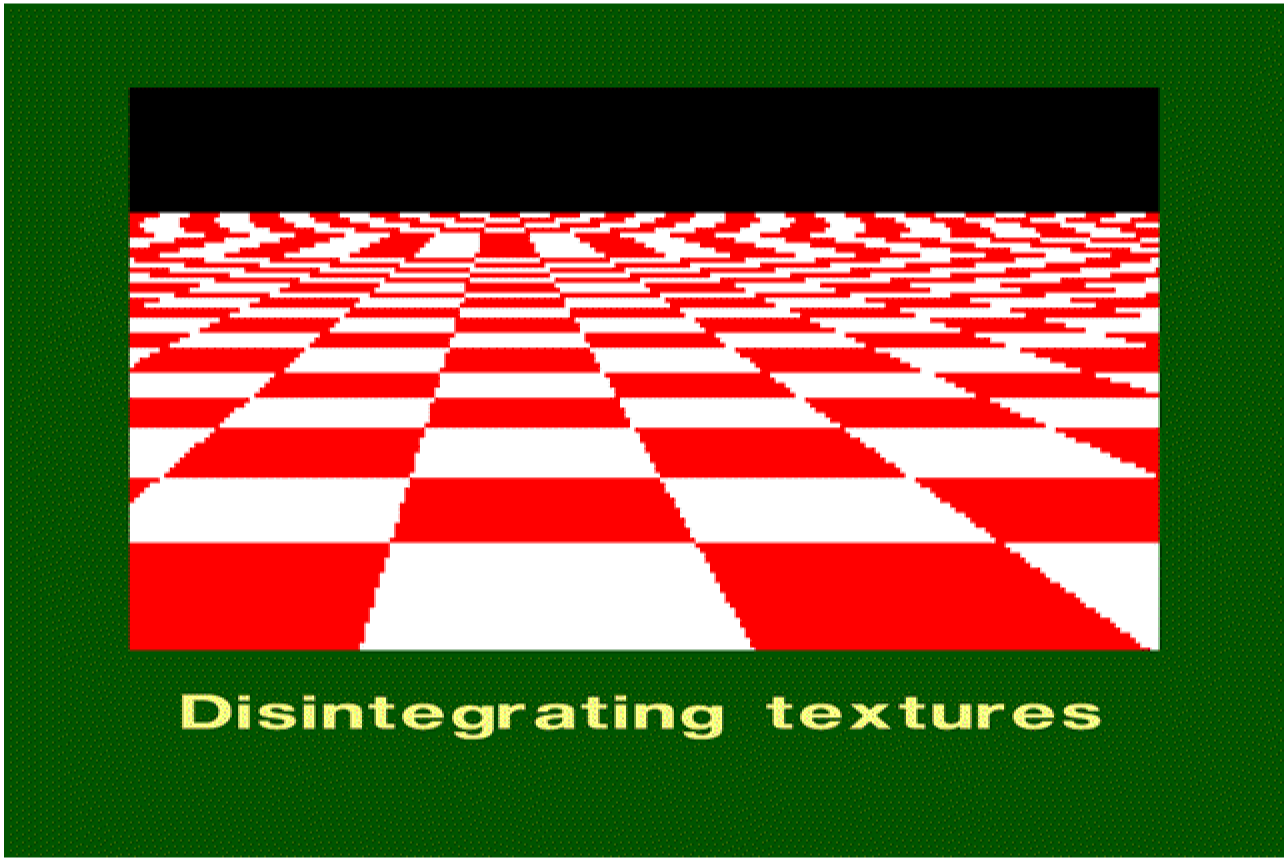
- Examining all levels of detail
  - Find edges with different amounts of blur
  - Find textures with different spatial frequencies (levels of detail)

Slide credit: David Lowe (UBC)

# Things can go wrong! -- Aliasing

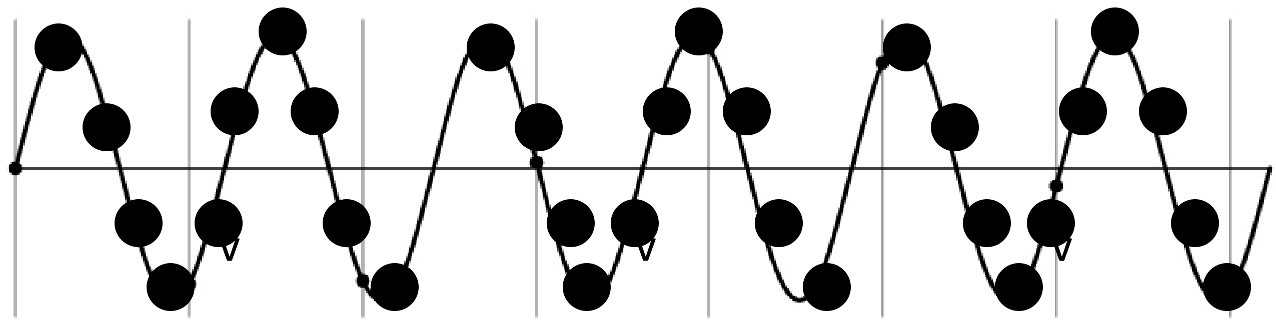


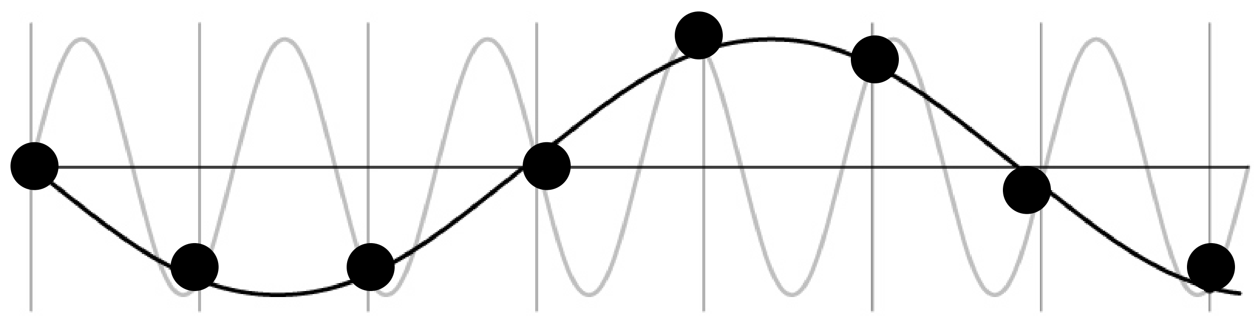
# Things can go wrong! -- Aliasing



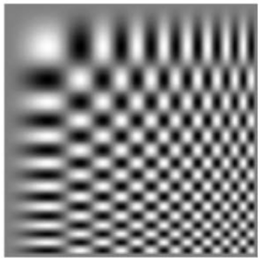
# Sampling Theorem (Nyquist)

- When sampling a signal at discrete intervals, the sampling frequency must be $\geq 2 \times f_{max}$
- $f_{max}$ = max frequency of the input signal.



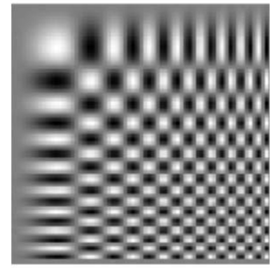
256x256



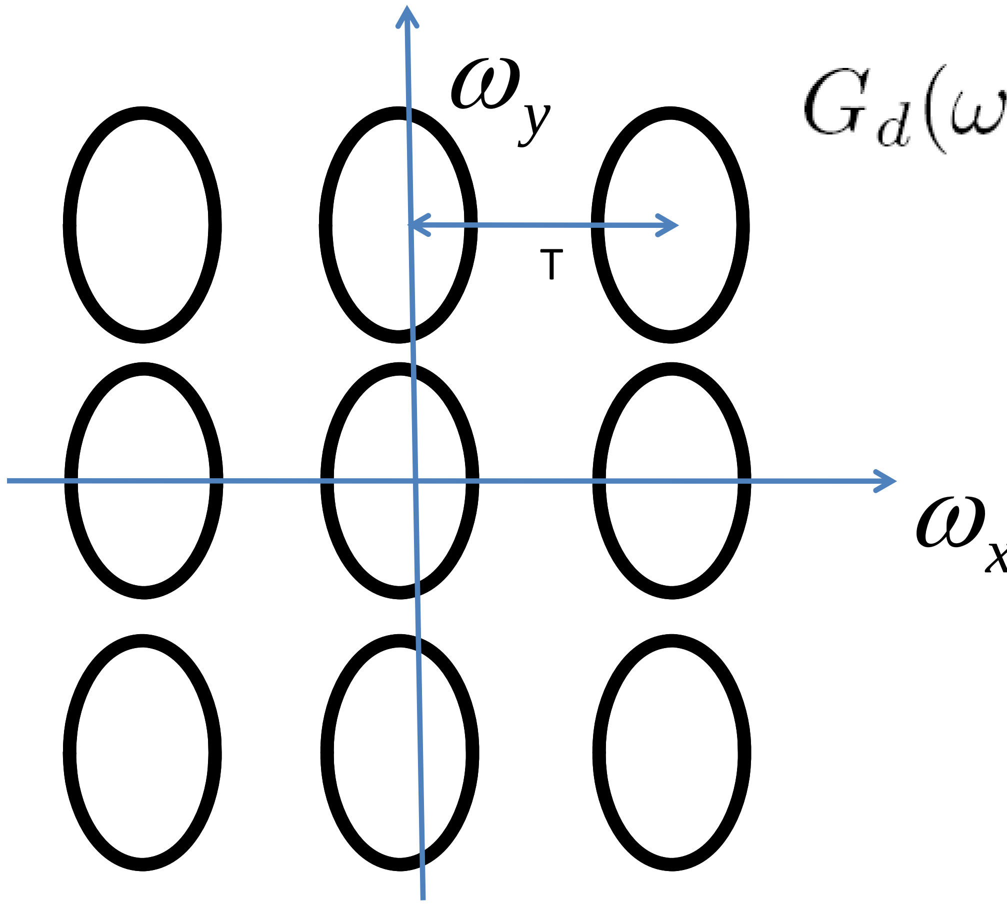
g [n,m]

Small sampling period N

64x64



$g_{sampled}$ [n,m]

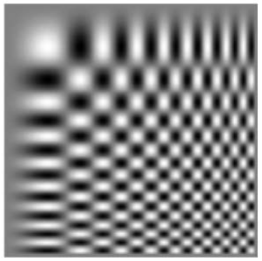



$G_d(\omega_X, \omega_Y)$

- N = sampling period
- T = periodicity of the replicas of the DFT (g)

$$T \sim 1/N$$

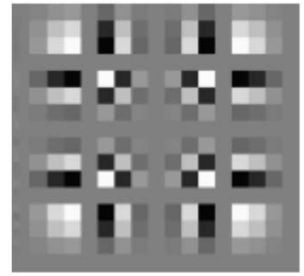
256x256



g [n,m]

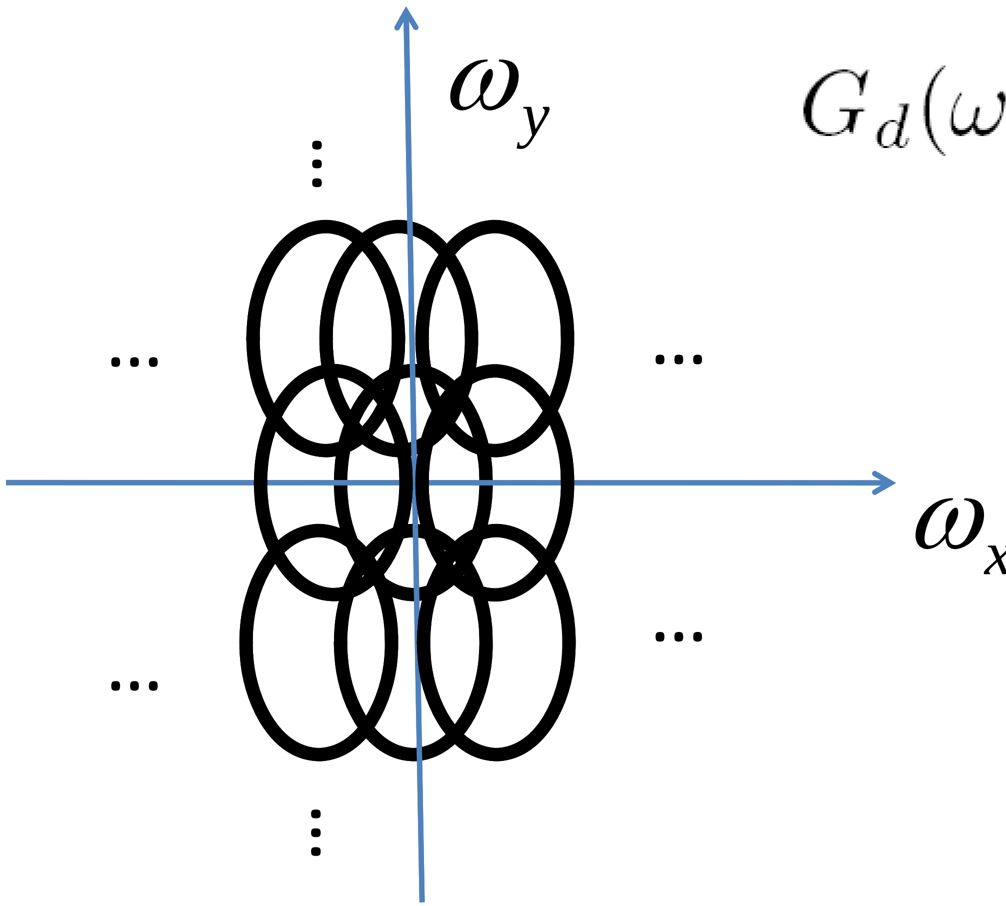
Large sampling period N

16x16



$g_{sampled}$ [n,m]

$G_d(\omega_X, \omega_Y)$




- N = sampling period
- T = periodicity of the replicas of the DFT (g)

T ~ 1/N

# Anti-aliasing

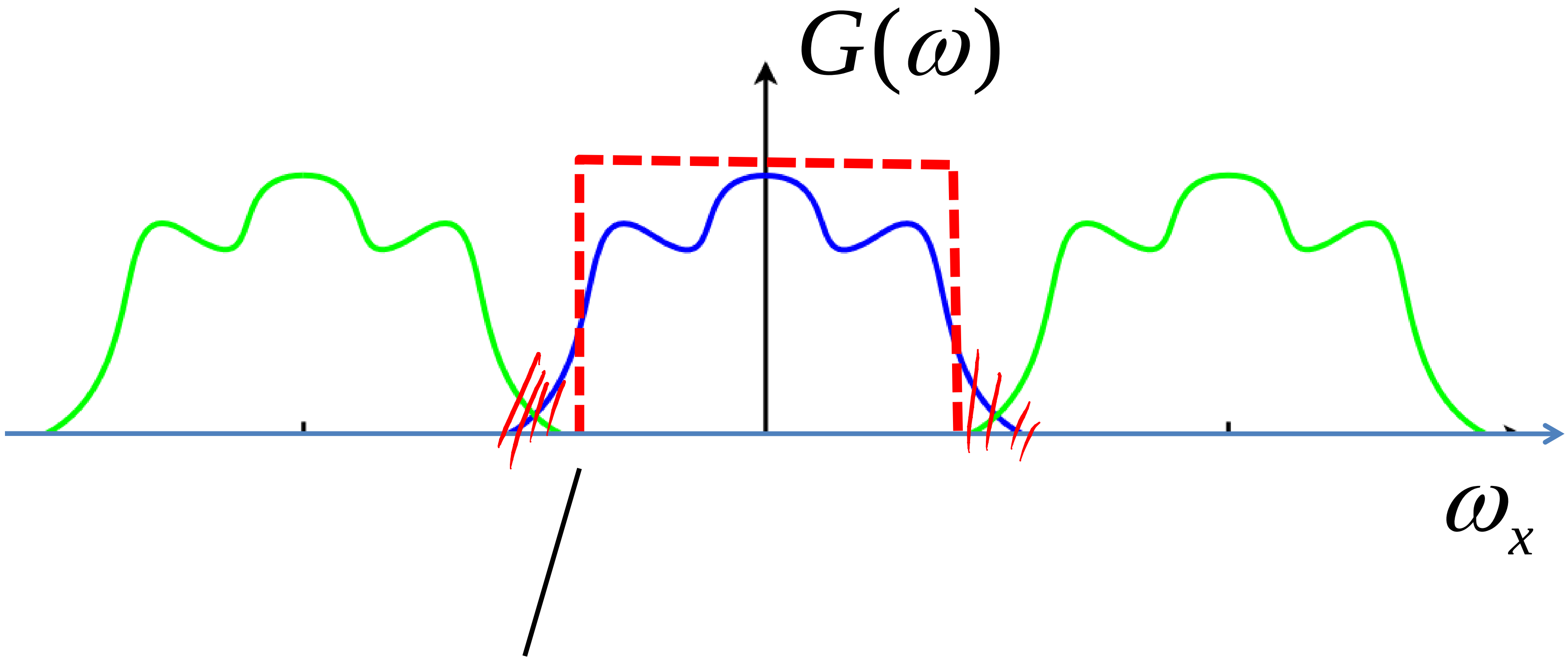
Solutions:

- Sample more often

- Get rid of all frequencies that are greater than half the new sampling frequency
  - Will lose information - but it's better than aliasing
  - Apply a smoothing filter to remove high frequencies

# Anti-aliasing

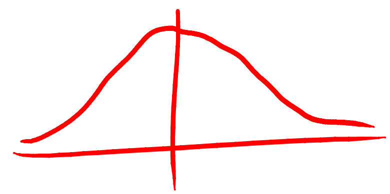
Apply a smoothing filter to remove high frequencies:




Low pass filter (smoothing)

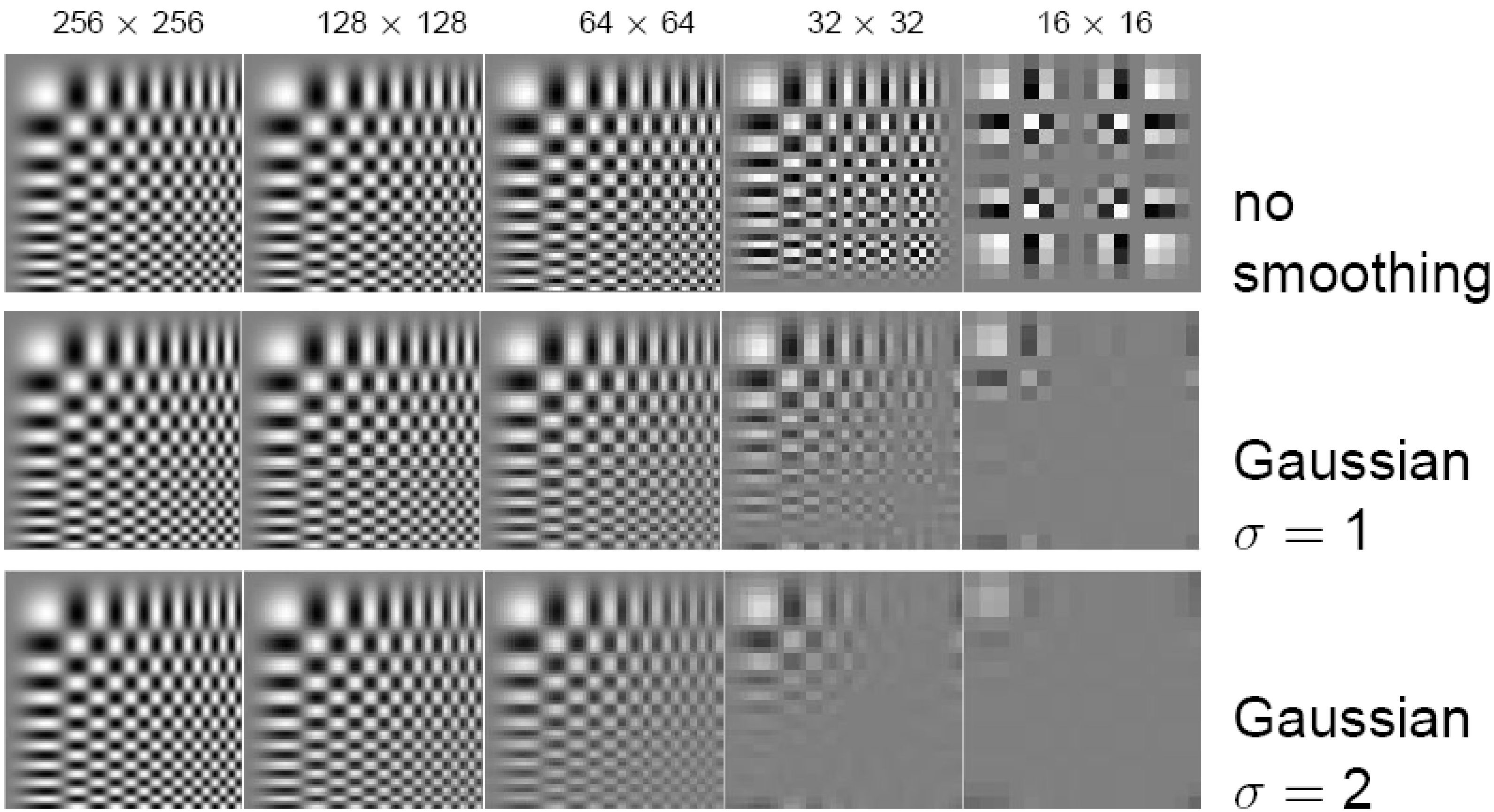
# Sampling algorithm

**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 $\sigma$ 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

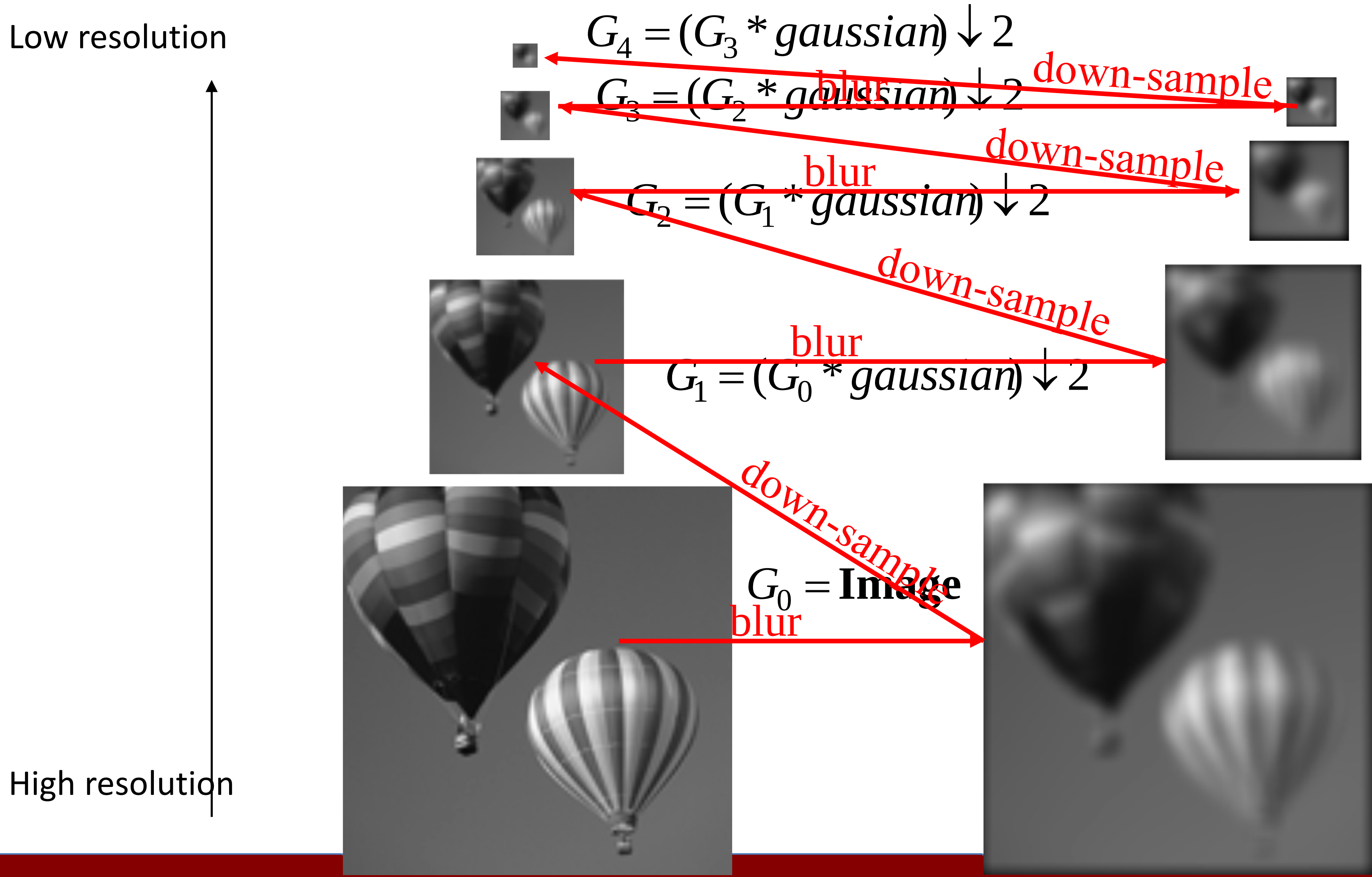
# Resampling with Prior Smoothing



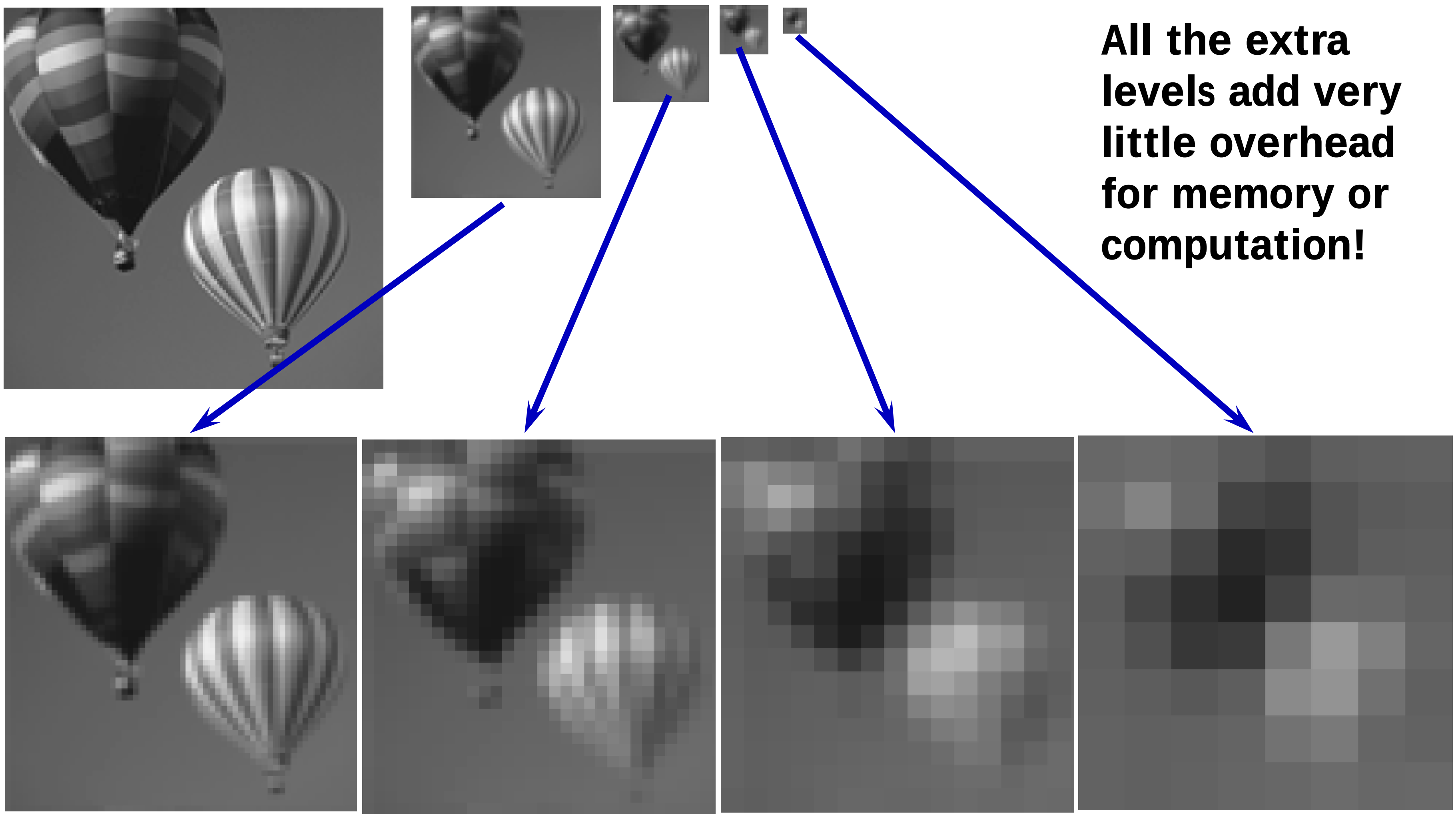

- Note: We cannot recover the high frequencies, but we can avoid artifacts by smoothing before resampling.

Image Source: Forsyth & Ponce

# The Gaussian Pyramid



# Gaussian Pyramid – Stored Information



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

Source: Irani & Basri

# Summary: Gaussian Pyramid

- Construction: create each level from previous one
  - Smooth and sample
- Smooth with Gaussians, in part because
  - a Gaussian*Gaussian = another Gaussian
  - $G(\sigma_1) * G(\sigma_2) = G(\mathrm{sqrt}(\sigma_1^{\,2} + \sigma_2^{\,2}))$
- Gaussians are low-pass filters, so the representation is redundant once smoothing has been performed.

  $\Rightarrow$ There is no need to store smoothed images at the full original resolution.

Slide credit: David Lowe

# Application: Vision system for TV remote control

- uses template matching

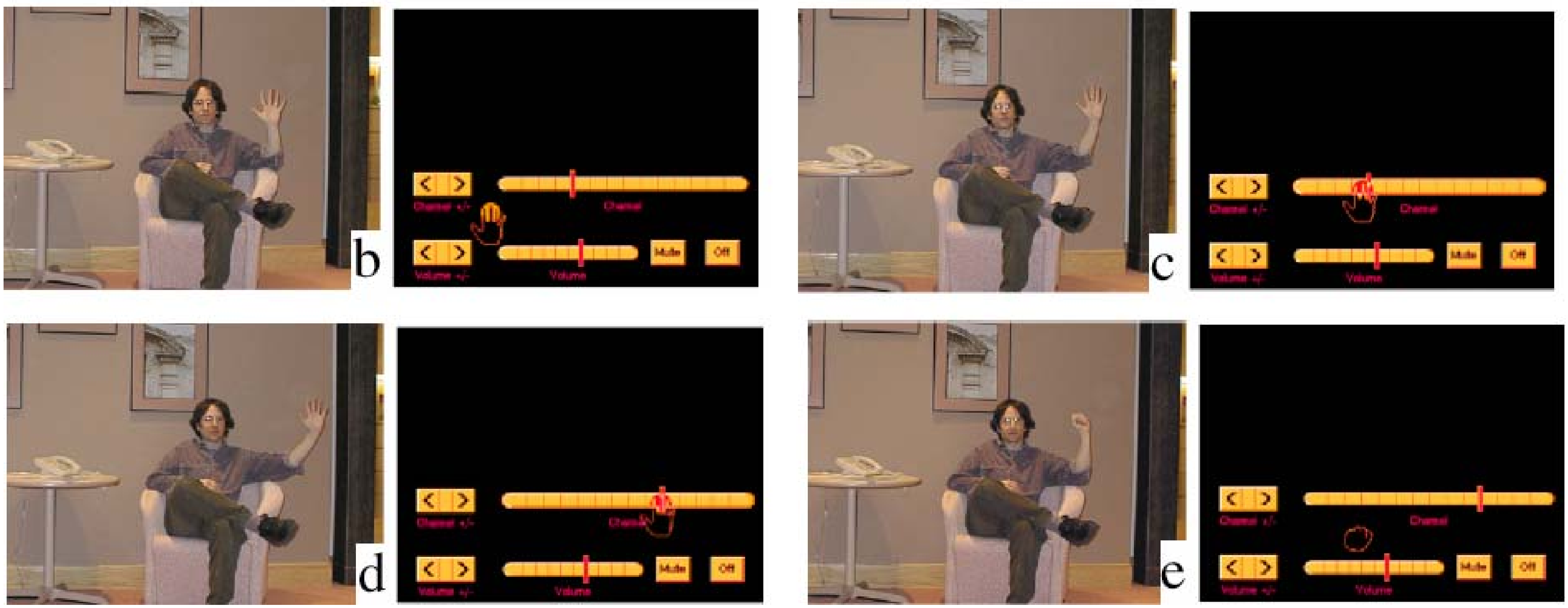




Figure from “Computer Vision for Interactive Computer Graphics,” W.Freeman et al, IEEE Computer Graphics and Applications, 1998 copyright 1998, IEEE

# What we have learned today?

- Images as functions
- Linear systems (filters)
- Convolution and correlation
- Discrete Fourier Transform (DFT)
- Sampling and aliasing

# Appendix

# Convergence

If g absolutely summable: $\sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} |g[n,m]| < \infty$

then

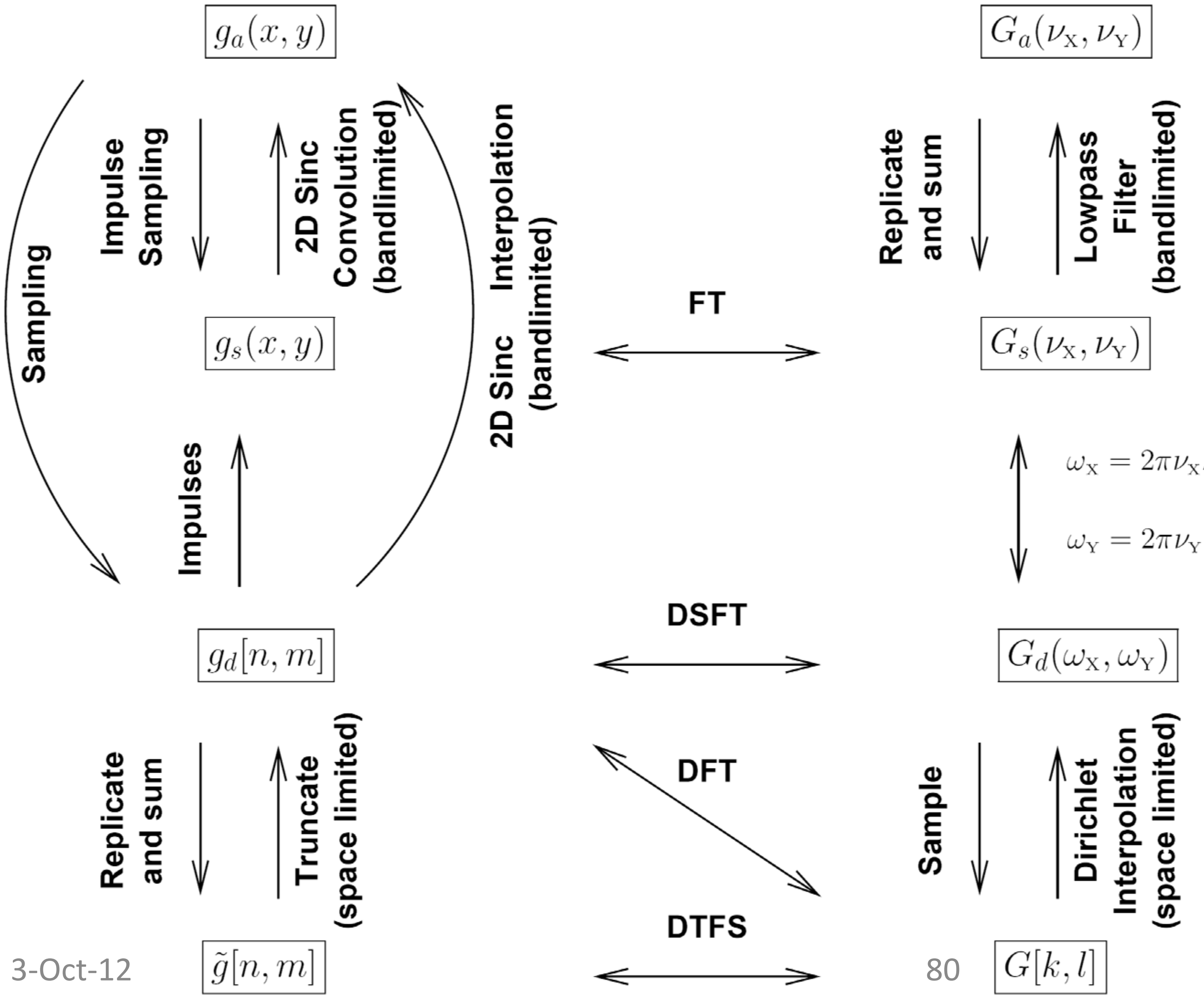
$$\lim_{N\to\infty} \boxed{\sum_{n=-N}^{N} \sum_{m=-N}^{N} g[n,m]\, e^{-\imath(\omega_X n + \omega_Y m)}} = G(\omega_X, \omega_Y)$$

If g is square summable (energy signal):

$$E_g \triangleq \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} |g[n,m]|^2 < \infty$$

$$\int_{-\pi}^{\pi} \int_{-\pi}^{\pi} \left| \boxed{G_N(\omega_X, \omega_Y)} - G(\omega_X, \omega_Y) \right|^2 d\omega_X \, d\omega_Y \to 0$$

$N \to \infty$


$g_a(x,y)$
$G_a(\nu_{\mathrm{X}},\nu_{\mathrm{Y}})$
Sampling
Impulse Sampling
2D Sinc Convolution (bandlimited)
2D Sinc Interpolation (bandlimited)
Replicate and sum
Lowpass Filter (bandlimited)
FT
$g_s(x,y)$
$G_s(\nu_{\mathrm{X}},\nu_{\mathrm{Y}})$
Impulses
$\omega_{\mathrm{X}} = 2\pi\nu_{\mathrm{X}}$
$\omega_{\mathrm{Y}} = 2\pi\nu_{\mathrm{Y}}$
DSFT
$g_d[n,m]$
$G_d(\omega_{\mathrm{X}},\omega_{\mathrm{Y}})$
Replicate and sum
Truncate (space limited)
DFT
Sample
Dirichlet Interpolation (space limited)
DTFS
$\tilde{g}[n,m]$
$G[k,l]$

# Fast Fourier transforms (FFT)

- Brute-force evaluation of the 2D DFT would require O((NM)2) flops

$$X[k,l] \triangleq$$

$$= \begin{cases} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x[n,m]\, e^{-\imath 2\pi (kn/N + lm/M)} & k = 0,\ldots,N-1, l = 0,\ldots,M-1 \\ 0, & \text{otherwise.} \end{cases}$$

- DFT is a **separable operation**
  → we can reduce greatly the computation

# Fast Fourier transforms (FFT)

- DFT is a **separable operation:**

$$X[k,l] = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} x[n,m]\, \mathrm{e}^{-\imath 2\pi (kn/N + lm/M)}$$

$$= \sum_{n=0}^{N-1} \mathrm{e}^{-\imath 2\pi kn/N} \left[ \sum_{m=0}^{M-1} x[n,m]\, \mathrm{e}^{-\imath 2\pi lm/M} \right]$$

- Apply the 1D DFT to each column of the image, and then apply the 1D DFT to each row of the result.

- Use the **fast Fourier transform (1-D FFT)** for these 1D DFTs!

# Fast Fourier transforms (FFT)

- FFT computational efficiency:

- **inner set of 1D FFTs** : N **O**(M logM)
- **outer set of 1D FFTs**: M **O**(N logN)

•**Total: O**(MN logMN) flops

- A critical property of FFT is that $N = 2^k$ with k =integer

If x = 512x512 image →
saving is a factor of 15000 relative to the brute-force 2D DFT!

Matlab: fft2 = fft(fft(x).').'

# FFT & Efficiency

- **General goal:** perform efficient linear convolution
- Perform convolution as product of DFTs
- **Pros:** DFT can be implemented using the FFT (fast fourier transform)
  - FFT is very efficient (fast!)
- **Cons:** DFT perform circular convolution
  - Compensate the wrap-around effect
- **Cons:** Online-memory storage
  - Use the overlap-add method or overlap-save method

# FFT & Efficiency

- Suppose we wish to convolve a **256 × 256** image with a **17 × 17** filter.
- The result will be **272×272**.
- The smallest prime factors of 272 is **2**.
- So one could pad to a **512 × 512 image**
- Note: only 28% of the final image would be the part we care about - the rest would be zero in exact arithmetic.

- Handling to a 512 × 512 image requires **much memory**

→ Use overlap-add method