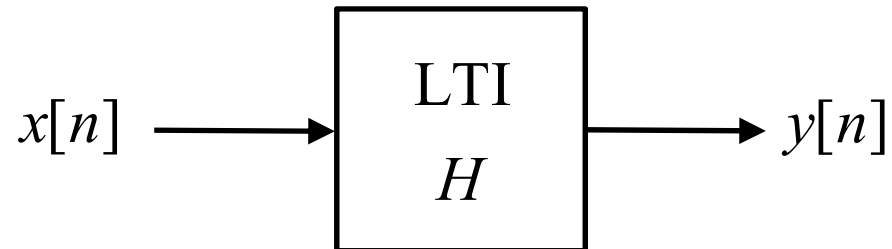


# **Module 5**

## **Linear Image Filtering**

- **Wraparound and Linear Convolution**
- **Linear Image Filters**
- **Linear Image Denoising**
- **Linear Image Restoration (Deconvolution)**

# REVIEW OF 1D LTI SYSTEMS



- Operator notation:  $y[n] = H \{x[n]\}$
- In English, this is read: “ $y[n]$  is the output of the system  $H$  when  $x[n]$  is the input.”

# THE MOST IMPORTANT PROPERTIES

- **Impulse response:** when the input is  $\delta[n]$ , the output is  $h[n]$ .

$$h[n] = H\{\delta[n]\}$$

- **Homogeneity:** if  $y[n] = H\{x[n]\}$  and  $c$  is a constant, then

$$H\{cx[n]\} = cH\{x[n]\} = cy[n].$$

- In other words, the action of the system **commutes** with multiplication by constants.

- **Superposition:** if  $y_1[n] = H\{x_1[n]\}$  and  $y_2[n] = H\{x_2[n]\}$ , then

$$H\{x_1[n] + x_2[n]\} = H\{x_1[n]\} + H\{x_2[n]\} = y_1[n] + y_2[n].$$

- In other words, the action of the system **commutes** with sums.

# IMPORTANT LTI PROPERTIES

- Together, homogeneity and superposition are called **linearity**.
- Together, they imply that the action of a **linear** system commutes with linear combinations:

$$H \{c_1 x_1[n] + c_2 x_2[n]\} = c_1 y_1[n] + c_2 y_2[n].$$

- **Translation Invariance:** if  $y[n] = H\{x[n]\}$  and  $n_0$  is an integer constant, then

$$H \{x[n - n_0]\} = y[n - n_0].$$

- In other words, the action of the system **commutes** with (time) shifts.
- Also called **time invariance** or **shift invariance**.

# 1D Linear Convolution

- As we have seen, any 1D discrete-time signal  $x[n]$  can be written as a linear combination of the translates of  $\delta[n]$ :

$$x[n] = \cdots + x[-1]\delta[n+1] + x[0]\delta[n] + x[1]\delta[n-1] + \cdots$$

- Here, it is **important** to realize that  $x[-1]$ ,  $x[0]$ ,  $x[1]$ , etc., are constants; they are **numbers**.
- So, if  $x[n]$  is the input to an LTI system  $H$ , the output is

$$\begin{aligned} y[n] &= H \{x[n]\} \\ &= H \{ \cdots + x[-1]\delta[n+1] + x[0]\delta[n] + x[1]\delta[n-1] + \cdots \} \\ &= \cdots + x[-1]H \{ \delta[n+1] \} + x[0]H \{ \delta[n] \} + x[1]H \{ \delta[n-1] \} + \cdots \\ &= \cdots + x[-1]h[n+1] + x[0]h[n] + x[1]h[n-1] + \cdots \end{aligned}$$

# $\Sigma$ Notation

- To save time and paper, we can write this **exact same thing** using “capital Sigma do-loops”:

$$x[n] = \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]$$

$$\begin{aligned} y[n] &= H \{x[n]\} \\ &= H \left\{ \sum_{k=-\infty}^{\infty} x[k]\delta[n-k] \right\} \\ &= \sum_{k=-\infty}^{\infty} x[k]H \{ \delta[n-k] \} \\ &= \sum_{k=-\infty}^{\infty} x[k]h[n-k] \end{aligned}$$

- This is called **linear convolution**; written  $y[n] = x[n] * h[n]$ .

# Interpretation

- For each  $n$ , the output signal  $y[n]$  is a number.
- This number is given by the dot product of the input  $x[n]$  with a **flipped-and-shifted version** of the impulse response:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \langle x[k], h[-k - (-n)] \rangle$$

- Another way to think of it:
  - Let the input be  $x[n] = 2\delta[n] + 3\delta[n-1] + 4\delta[n-2]$ .
  - We can think of this as a sum of three input signals.
  - For  $2\delta[n]$ , the output is  $2h[n]$ .
  - For  $3\delta[n-1]$ , the output is  $3h[n-1]$ .
  - For  $4\delta[n-2]$ , the output is  $4h[n-2]$ .
  - The total output is the sum of these: that's **convolution**.

# More About 1D Linear Convolution

- Continuous-time version:

$$y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\theta)h(t - \theta)d\theta.$$

- Computing 1D linear convolution in the transform domain:

$$Y(e^{j\omega}) = X(e^{j\omega})H(e^{j\omega})$$

discrete time

$$Y(z) = X(z)H(z)$$

$$Y(\Omega) = X(\Omega)H(\Omega)$$

continuous time

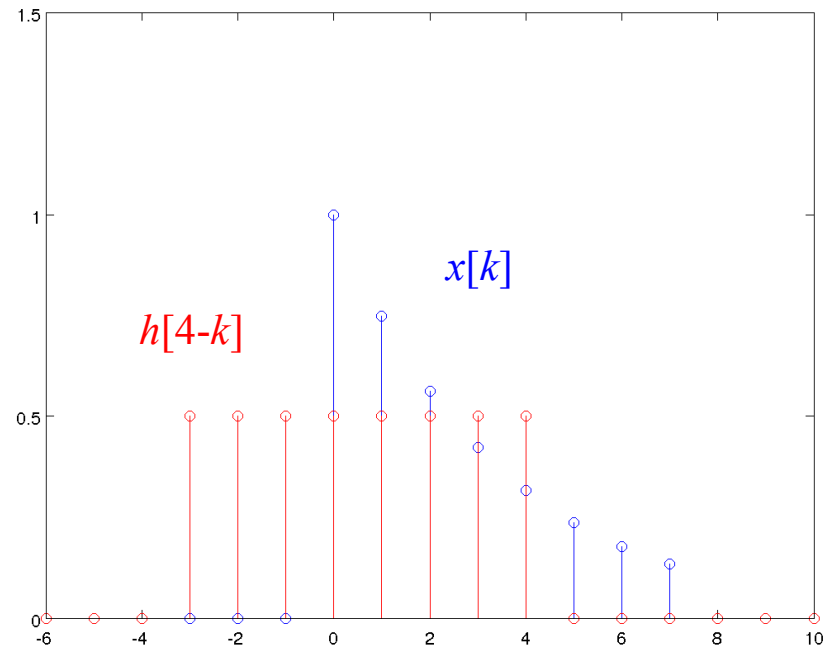
$$Y(s) = X(s)H(s)$$

# An Important Idea

- Suppose  $x[n] = (\frac{3}{4})^n$ ,  $0 \leq n \leq 7$ , and zero otherwise.
- Let  $h[n] = \frac{1}{2}$ ,  $0 \leq n \leq 7$ , and zero otherwise.
- Then, according to the convolution formula,

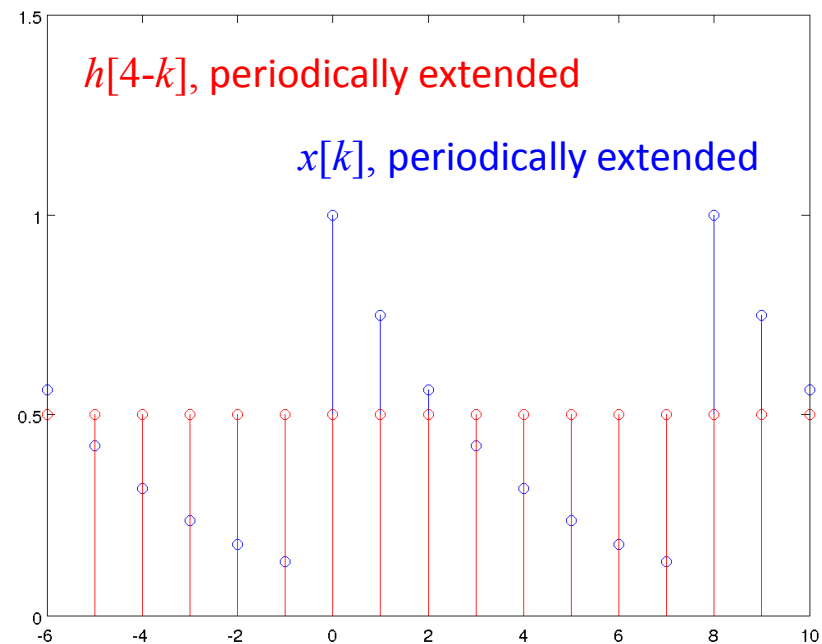
$$y[4] = \sum_{k=-\infty}^{\infty} x[k]h[4-k]$$

- Notice that the product  $x[k]h[4-k]$  is zero for  $k < 0$  because  $x[k]$  is zero there.
- Similarly, the product is zero for  $k > 4$  because  $h[4-k]$  is zero there.
- In the linear convolution sum, we get **zero** for the product in places  $k$  where one of the signals “hangs over.”



- Now suppose we try to compute this same convolution by multiplying the 8-point DFT's  $X[k]$  and  $H[k]$ .
- Recall that, to the DFT, a finite-length signal is **one period** of a **periodic** signal.
- So the picture will be different this time!

- Because the signals are now **periodically extended**, there will no longer be zeros in the sum at places where one of the signals “hangs over.”
- The number we get for  $y[4]$  this way will **not** be the same as what we got by linear convolution on the last page.
- It is something **different** – it is called **wraparound convolution**.
- More on this in a minute...



# *n*D Convolution

- In  $n$ D, a discrete LTI system  $H$  has an impulse response  $h[\mathbf{p}]$ , where  $\mathbf{p} \in \mathbb{Z}^n$ .
- For an  $n$ D input  $x[\mathbf{p}]$ , the output  $y[\mathbf{p}]$  is given by the  $n$ D discrete linear convolution

$$y[\mathbf{p}] = x[\mathbf{p}] * h[\mathbf{p}] = \sum_{\mathbf{q} \in \mathbb{Z}^n} x[\mathbf{q}]h[\mathbf{p} - \mathbf{q}].$$

- Note that this has the same form as the 1D version.
- For continuous space with  $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ , the linear convolution is given by

$$y(\mathbf{v}) = x(\mathbf{v}) * h(\mathbf{v}) = \int_{\mathbb{R}^n} x(\mathbf{w})h(\mathbf{v} - \mathbf{w})d\mathbf{w}.$$

# The 2D Case

- Now let's go back to our usual notation and write out the formulas from the last page for the special case of 2D.
- Let  $H(m, n)$  be the impulse response of a discrete 2D LTI system.
- Let the input be  $I(m, n)$ .
- The output is given by the 2D discrete linear convolution

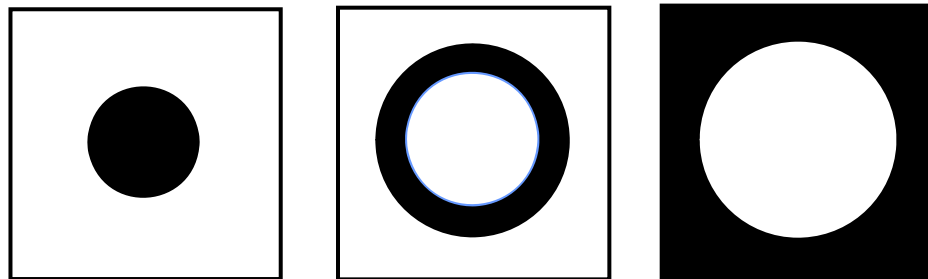
$$J(m, n) = I(m, n) * H(m, n) = \sum_{p=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} I(p, q)H(m - p, n - q).$$

- For continuous space, the 2D linear convolution is given by

$$J_C(x, y) = I_C(x, y) * H_C(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I_C(\alpha, \beta)H_C(x - \alpha, y - \beta)d\alpha d\beta.$$

## 2D Wraparound Convolution

- Modifying the DFT of an image changes its appearance. For example, multiplying a DFT by a **zero-one mask** predictably modifies image appearance:



- But is this linear convolution?

# Multiplying 2D DFTs

- What happens when two arbitrary 2D DFTs are multiplied together pointwise?

$$\tilde{\mathbf{J}} = \tilde{\mathbf{I}}_1 \otimes \tilde{\mathbf{I}}_2 \quad \text{or} \quad \tilde{\mathbf{J}} = \tilde{\mathbf{I}}_1 \Delta \tilde{\mathbf{I}}_2$$

- The answer has profound consequences in image processing.
- Pointwise division can be treated as multiplication by the reciprocal of  $\tilde{\mathbf{I}}_2$ , but special handling is needed if  $\tilde{\mathbf{I}}_2$  contains values that are zero or nearly zero.

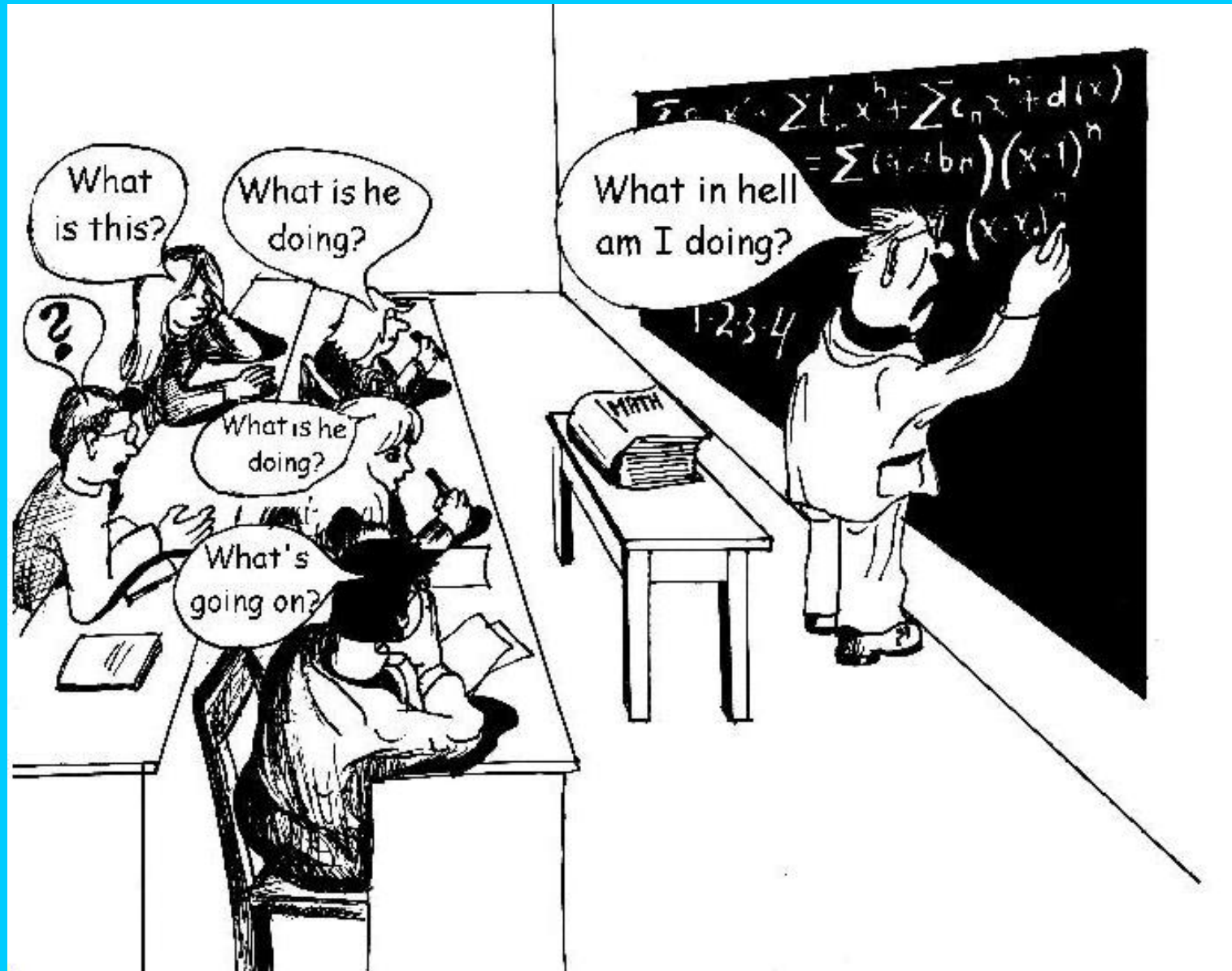
# Multiplying DFTs

- Consider the **product**

$$\tilde{\mathbf{J}} = \tilde{\mathbf{I}}_1 \otimes \tilde{\mathbf{I}}_2$$

- This has inverse DFT

$$\begin{aligned} \mathbf{J}(m, n) &= \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \tilde{\mathbf{J}}(u, v) W_M^{-um} W_N^{-vn} \\ &= \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \tilde{\mathbf{I}}_1(u, v) \tilde{\mathbf{I}}_2(u, v) W_M^{-um} W_N^{-vn} \\ &= \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \left\{ \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} \mathbf{I}_1(p, q) W_M^{up} W_N^{vq} \right\} \left\{ \sum_{r=0}^{M-1} \sum_{s=0}^{N-1} \mathbf{I}_2(r, s) W_M^{ur} W_N^{vs} \right\} W_M^{-um} W_N^{-vn} \end{aligned}$$



$$= \frac{1}{MN} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} I_1(p, q) \sum_{r=0}^{M-1} \sum_{s=0}^{N-1} I_2(r, s) \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} W_M^{u(p+r-m)} W_N^{v(q+s-n)}$$

$$\star = \frac{1}{MN} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} I_1(p, q) \sum_{r=0}^{M-1} \sum_{s=0}^{N-1} I_2(r, s) \cdot MN \cdot \delta(p+r-m, q+s-n)$$

$$\star \star = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} I_1(p, q) I_2 \left[ (m-p)_M, (n-q)_N \right]$$

$$= \sum_{r=0}^{M-1} \sum_{s=0}^{N-1} I_1 \left[ (m-r)_M, (n-s)_N \right] I_2(r, s)$$

**Note:**  $(p)_N = p \bmod N$

$$= I_1(m, n) \circledast I_2(m, n)$$

$I_1 \circledast I_2$  is the **wraparound convolution** of  $I_1$  with  $I_2$

The steps  $\star$  and  $\star \star$  are explained on the next 3 pages.

- To understand ★ on page 5.17, let

$$\ominus = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} W_M^{u(p+r-m)} W_N^{v(q+s-n)} = \underbrace{\sum_{u=0}^{M-1} W_M^{u(p+r-m)}}_{\odot} \underbrace{\sum_{v=0}^{N-1} W_N^{v(q+s-n)}}_{\ominus}$$

- If  $p = m-r$  and  $q = n-s$ , then  $p+r-m = q+s-n = 0$ , so

$$\odot = \sum_{u=0}^{M-1} W_M^0 \sum_{v=0}^{N-1} W_N^0 = \sum_{u=0}^{M-1} 1 \sum_{v=0}^{N-1} 1 = MN$$

- A useful sum formula:  $\sum_{n=A}^B a^n = \frac{a^A - a^{B+1}}{1-a}$ , provided  $a \neq 1$ .

- Now, if  $p \neq m-r$ , then  $p+r-m \neq 0$ , so with  $A = 0$ ,  $B = M-1$ , and  $a = W_M^{p+r-m}$ ,

$$\begin{aligned} \odot &= \sum_{u=0}^{M-1} W_M^{u(p+r-m)} = \frac{W_M^0 - W_M^{M(p+r-m)}}{1 - W_M^{p+r-m}} \\ &= \frac{1 - e^{j2\pi M(p+r-m)/M}}{1 - e^{j2\pi(p+r-m)/M}} = \frac{1 - e^{j2\pi(\text{integer})}}{1 - e^{j2\pi(\text{not integer})}} \\ &= \frac{1-1}{1 - (\text{not } 1)} = \frac{0}{\text{not zero}} = 0. \end{aligned}$$

- Understanding ★ ...
- Similarly, if  $q \neq n-s$ , then  $q+s-n \neq 0$ , so  $\textcircled{\ominus} = 0$ .
- All together,

$$\textcircled{\ominus} = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} W_M^{u(p+r-m)} W_N^{v(q+s-n)} = \begin{cases} MN, & p = m - r \text{ and } q = n - s, \\ 0, & \text{otherwise,} \end{cases}$$

$$= MN\delta(p + r - m, q + s - n),$$

which establishes the equality ★ on page 5.17.

- To understand ★★ on page 5.17, observe that

$$\begin{aligned}
& \sum_{r=0}^{M-1} \sum_{s=0}^{N-1} \mathbf{I}_2(r, s) \cdot MN \cdot \delta(p + r - m, q + s - n) \\
&= MN \sum_{r=0}^{M-1} \sum_{s=0}^{N-1} \mathbf{I}_2(r, s) \delta[r - (m - p), s - (n - q)] \\
&= MN \mathbf{I}_2(m - p, n - q).
\end{aligned}$$

- Now,  $0 \leq m, p \leq M - 1$ . So  $(1 - M) \leq m - p \leq (M - 1)$ .
- Likewise,  $0 \leq n, q \leq N - 1$ . So  $(1 - N) \leq n - q \leq (N - 1)$ .
- In other words,  $m - p$  can be outside the range 0 to  $M - 1$  and  $n - q$  can be outside the range 0 to  $N - 1$ , i.e., **outside** the bounds of the original  $\mathbf{I}_2$  image.
- But here  $\mathbf{I}_2$  must be interpreted as the IDFT of  $\tilde{\mathbf{I}}_2$ , which is the **periodic extension** of the original  $\mathbf{I}_2$  image.
- Thus, pixels outside the range  $[0, M-1]$ ,  $[0, N-1]$  must be taken from the periodic extension. In terms of the original  $\mathbf{I}_2$  image, this is conveniently written using modular arithmetic as  $MN \mathbf{I}_2[(p - m)_M, (q - n)_N]$ .

# Wraparound Convolution

- The summation

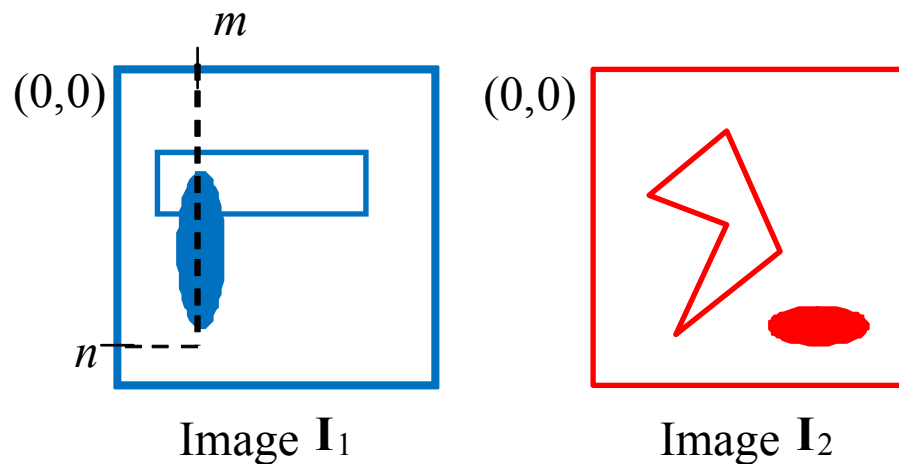
$$\begin{aligned} J(m, n) &= I_1(m, n) \circledast I_2(m, n) \\ &= \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} I_1(p, q) I_2 \left[ (m-p)_M, (n-q)_N \right] \end{aligned}$$

is also sometimes called **cyclic convolution**, **circular convolution**, or **periodic convolution**.

- It is an inner product between one sequence and a (doubly) reversed, shifted, and **periodically extended** version of the other.
- Mathematically, the periodic extension is accounted for by writing the indices of  **$I_2$  modulo- $M, N$** .

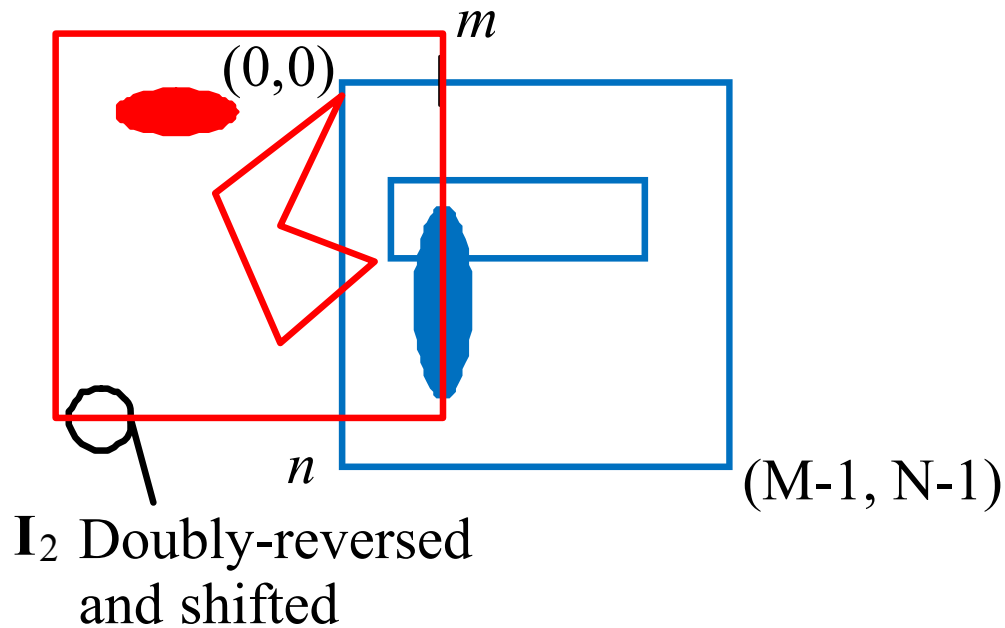
# Understanding Wraparound Convolution

- Consider **hypothetical images**  $\mathbf{I}_1$  and  $\mathbf{I}_2$

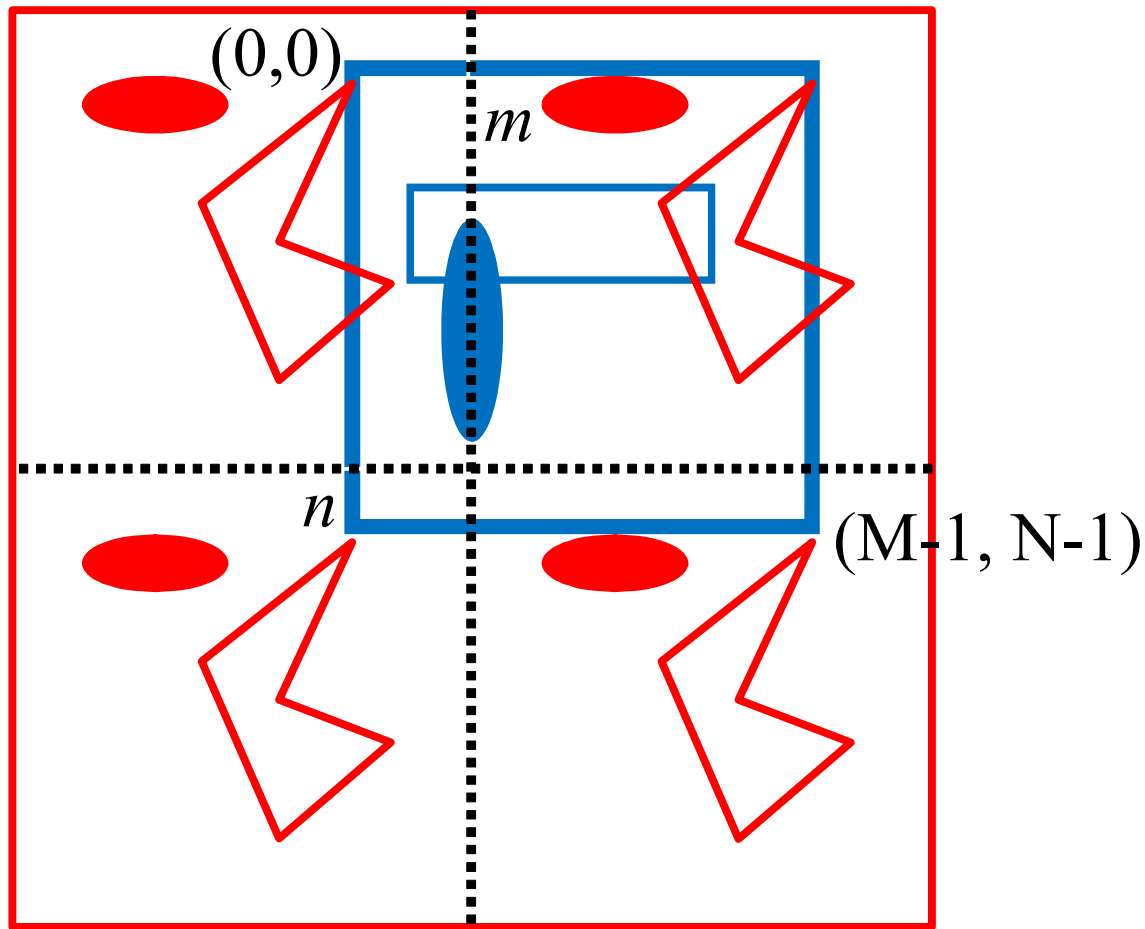


for which we wish to compute the **cyclic convolution** at  $(m, n)$  in the spatial domain (without DFTs).

- Without wraparound:



- For linear convolution, we would now compute the dot product of these two.
- But for wraparound convolution, the modulo arithmetic will periodically extend  $\mathbf{I}_2$ .



Overlay of periodic extension of shifted  $\mathbf{I}_2$   
 Summation occurs over  $0 \leq p \leq M-1, 0 \leq q \leq N-1$   
 i.e., over the blue  $\mathbf{I}_1$  image.

# Computation of Wraparound Convolution

- Direct computation of

$$\begin{aligned} J(m, n) &= I_1(m, n) \otimes I_2(m, n) \\ &= \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} I_1(p, q) I_2 \left[ (m-p)_M, (n-q)_N \right] \end{aligned}$$

is simple but expensive.

- For an  $M \times N$  image:
  - for **each** of the  $MN$  pixels in **J**, **MN additions and MN multiplies are required to compute  $J(m, n)$ .**
  - or  $(MN)(MN)$  multiply-add operations in total.
  - for  $M=N=512$ , this is  $2^{26} = 6.9 \times 10^{10}$

# DFT Computation of Wraparound Convolution

- Because of **FFT**, computing  $\otimes$  in the DFT domain is much faster, provided that  $M, N = \text{powers of } 2$ .
- Simply put,

$$\mathbf{J} = \mathbf{I}_1 \otimes \mathbf{I}_2 = \text{IFFT}_{M \times N} \left[ \text{FFT}_{M \times N} [\mathbf{I}_1] \otimes \text{FFT}_{M \times N} [\mathbf{I}_2] \right]$$

- Computing an  $(M \times N)$  FFT is  $\mathcal{O}[MN \cdot \log(MN)]$ , so computation of  $\otimes$  is as well.
- We will now discover that  $\otimes$  must be modified in order to make it useful in most applications.

# 2D LINEAR CONVOLUTION

- Wraparound convolution is a consequence of the **periodic DFT**.

- If two **DSFTs** are multiplied together:

$$\tilde{J}_D(U, V) = \tilde{I}_{D1}(U, V) \tilde{I}_{D2}(U, V)$$

then useful **linear convolution** results:

$$J(m, n) = I_1(m, n) * I_2(m, n)$$

- **Wraparound convolution** is an **artifact** of sampling the DSFT – which causes **spatial periodicity**.

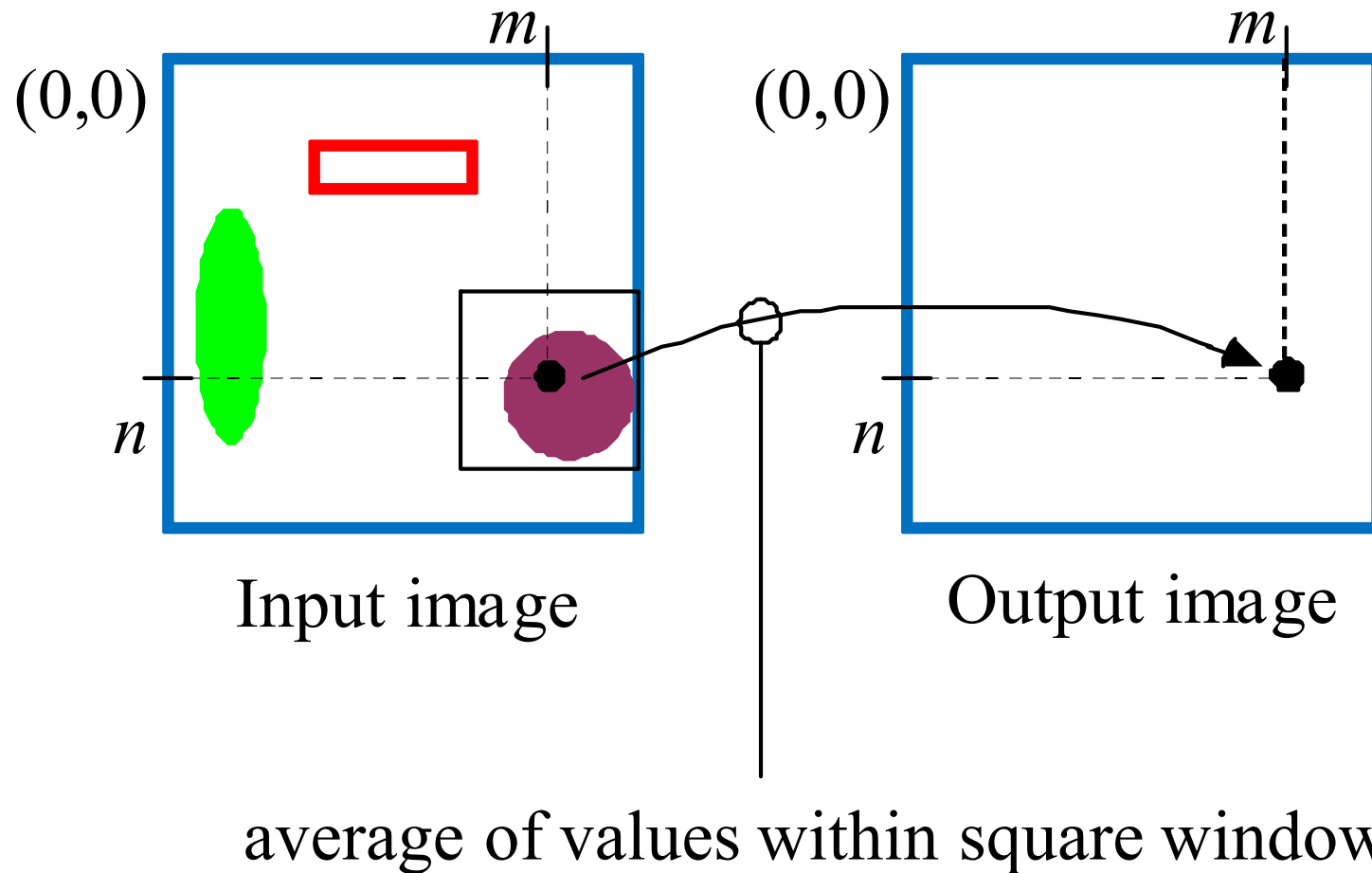
# About Linear Convolution

- Most of circuit theory, optics, and analog filter theory is based on **linear convolution**.
- And ... (linear) digital filter theory is based on the concept of **discrete linear convolution**.
- Fortunately, **wraparound convolution** can be used to compute **linear convolution**.

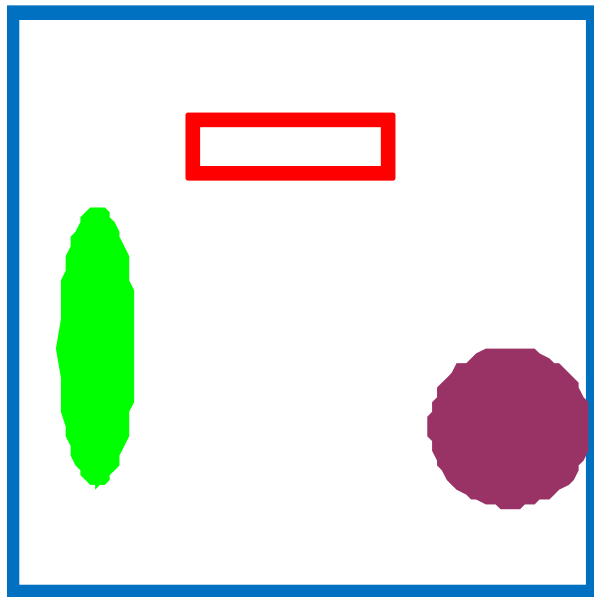
# Undesirability of Wraparound

- A very simple type of linear convolution is the **local average operation** (or averaging filter).
- Each image pixel is replaced by the **average** of its neighbors within a window:

# Depiction of Average Filtering



# Computation of Average Filtering



Input image

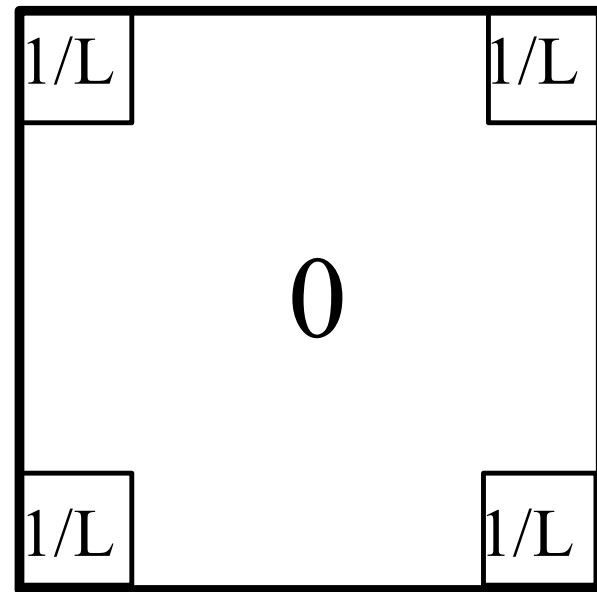
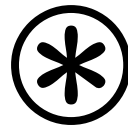
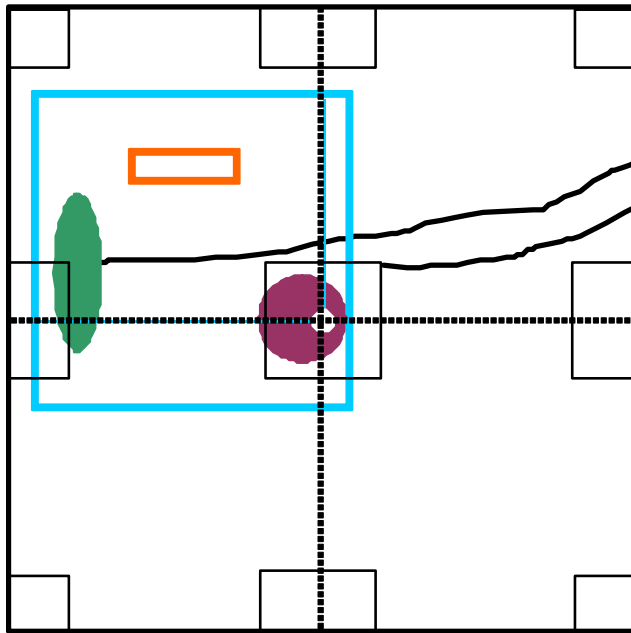


Image of square

The average filter operation may be expressed (at most points) as the wraparound convolution of the image with an image of a square with intensity  $1/L$ , where  $L = \#$  pixels in the square

# Wraparound Effect



Opposite sides  
of the image  
are being averaged  
together !

- Near the image borders, however, wraparound effects occur.
- Usually, it is desirable to average **only neighboring pixels** ...
- The effect is **much worse** if the filter is large.
- If the filter is **small**, then this can sometimes be fixed by trimming off the borders of the averaged image.

# Linear Convolution by Zero Padding

- Adapting wraparound convolution to do linear convolution is **conceptually simple**.
- Accomplished by **padding** the two image arrays with **zero values**.
- Then, the periodic extension wraps in only zero values.
- **Typically**, both image arrays are doubled in size, both vertically and horizontally:

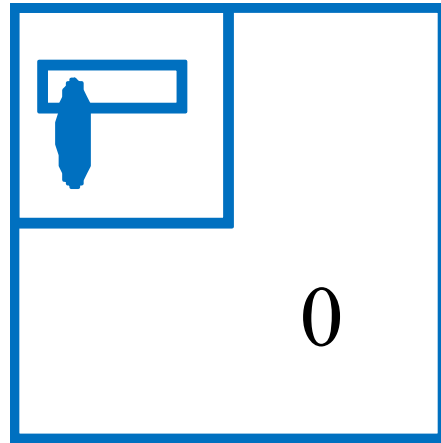


Image  $\mathbf{I}_1$   
(zero padded)

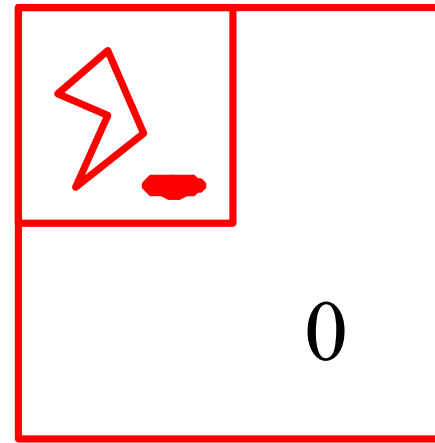
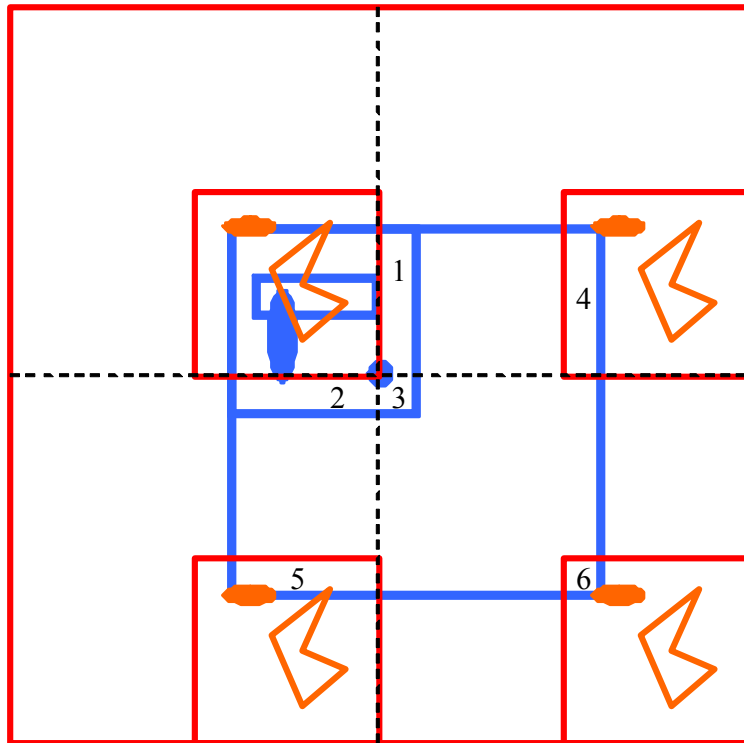


Image  $\mathbf{I}_2$   
(zero padded)

$2M \times 2N$  zero padded images

- **Wraparound eliminated**, since the zero padded, periodically extended, twice-reversed, and twice-shifted  $\mathbf{I}_2$  image will now wrap in only zero pixels over the original  $\mathbf{I}_1$  image.
- Can be seen by looking at the overlaps when computing the convolution at a point  $(m, n)$ :

# Wraparound Convolution of Zero Padded Images



Linear convolution by zero padding

- Remember, the wraparound convolution sum runs only over the **blue** square:  
( $0 \leq p \leq 2M-1$ ,  $0 \leq q \leq 2N-1$ ).
- In the areas 1, 2, 3 the wraparound effect is eliminated because the zero-padded  $\mathbf{I}_2$  image (red) is zero.
- In the areas 4, 5, 6 the wraparound effect is eliminated because the zero-padded  $\mathbf{I}_1$  image (blue) is zero.
- This makes the result the same as performing linear convolution – as we did on p. 5.23.

# 2D Linear Convolution by DFT

- Let  $\mathbf{I}_1$  and  $\mathbf{I}_2$  be  $M \times N$  digital images.
- We desire the **linear** convolution  $\mathbf{J} = \mathbf{I}_1 * \mathbf{I}_2$ .
- Let  $\mathbf{I}'_1$  and  $\mathbf{I}'_2$  be  $2M \times 2N$  zero padded versions of  $\mathbf{I}_1$  and  $\mathbf{I}_2$ .
- Compute:  
$$\mathbf{J}' = \mathbf{I}'_1 \circledast \mathbf{I}'_2 = \text{IFFT}_{2M \times 2N} \left\{ \text{FFT}_{2M \times 2N} [\mathbf{I}'_1] \otimes \text{FFT}_{2M \times 2N} [\mathbf{I}'_2] \right\}$$
- Then  $\mathbf{J}'$  is the  $2M \times 2N$  **wraparound** convolution of  $\mathbf{I}'_1$  and  $\mathbf{I}'_2$ .
- It contains the desired **linear** convolution  $\mathbf{J} = \mathbf{I}_1 * \mathbf{I}_2$ .
- But, we usually want the filtered image:
  - to be the same size as the original,
  - to not be shifted relative to the original.
- So, which part of  $\mathbf{J}'$  should we keep?
- The answer depends on the details of the application...

# Which part of J' Should We Keep?

- It will take some work to answer this question – because the answer depends on exactly what we are trying to do.
- Some textbooks (e.g., Gonzalez and Woods) give complicated rules for this. Invariably, the rules depend on specific assumptions about what you are trying to do.
- For us, it will be better to develop an understanding of what is going on from the **fundamentals**.
- Then you will be able to figure out the right answer no matter what the particular situation.
- As usual, it will be helpful to look at the 1D case first...

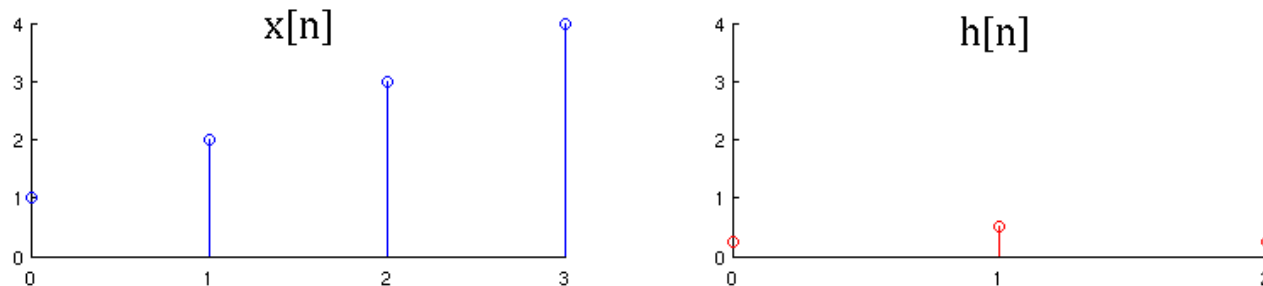
# Finite-Length 1D Linear Convolution

- Let  $H$  be a 1D LTI causal 3-point weighted average filter with

$$h[n] = \frac{1}{4}\delta[n] + \frac{1}{2}\delta[n-1] + \frac{1}{4}\delta[n-2].$$

- Let the input be the 4-point signal

$$x[n] = \delta[n] + 2\delta[n-1] + 3\delta[n-2] + 4\delta[n-3].$$

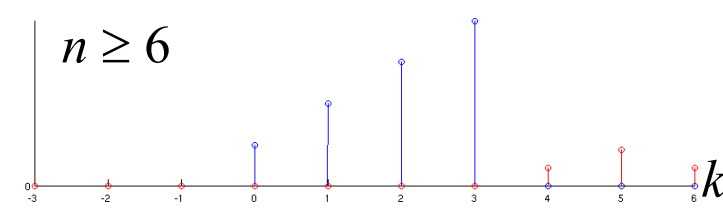
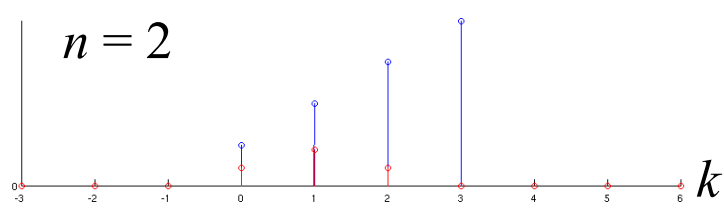
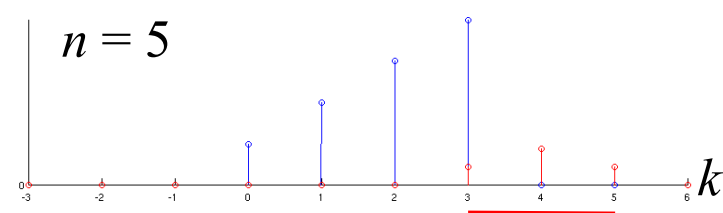
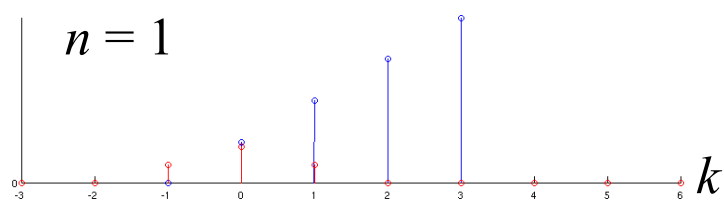
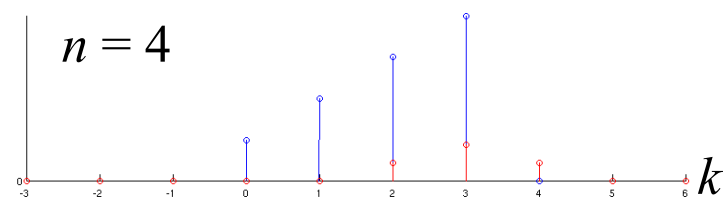
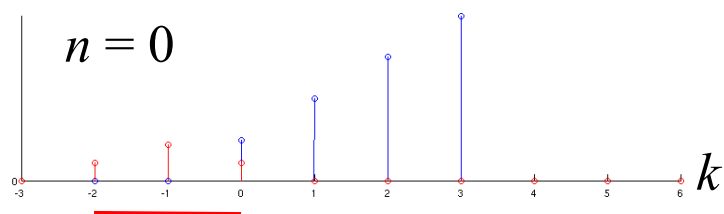
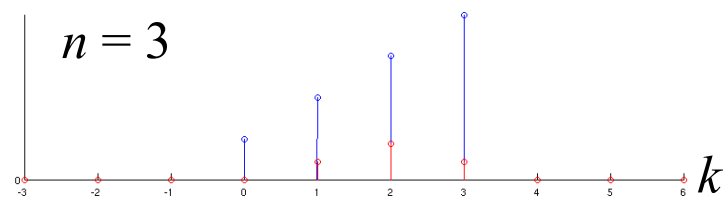
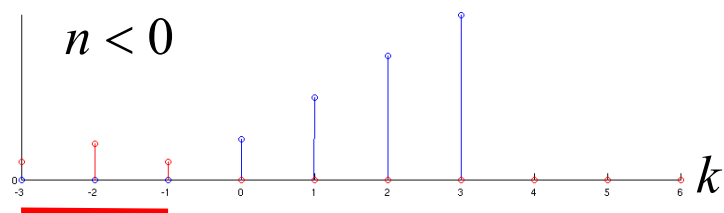


- Let  $N_1 = \text{length}(x[n]) = 4$  and  $N_2 = \text{length}(h[n]) = 3$ .
- The system output is the 1D linear convolution

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k].$$

$x[k]$

$h[n-k]$

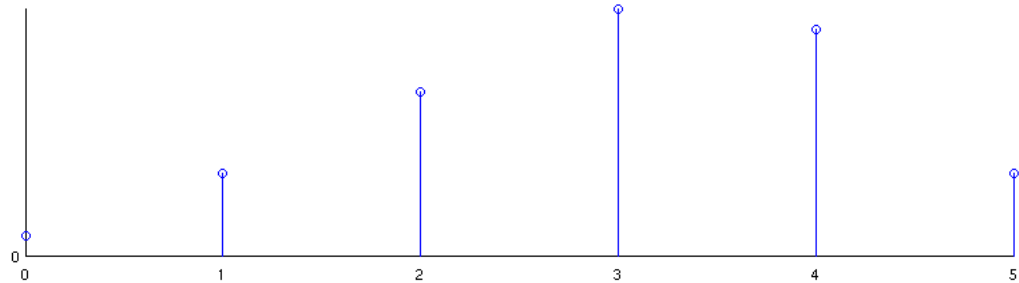


- There is nonzero overlap for  $n = 0, 1, 2, 3$ ; i.e., once for each sample in  $x[n]$ , and for  $n = 4, 5$ ; i.e., once for each sample in  $h[n]$  **but** the last.
- After that there is **no** overlap.
- So the length of the convolution is  $N_1 + N_2 - 1 = 6$ .

- More precisely, computing  $y[n]$  means computing the dot product of  $x[k]$  and  $h[n-k]$  for each  $n$ .
- Consider values of  $n$  starting at the far left ( $-\infty$ ) and going right:
  - for  $n < 0$ , the rightmost sample of  $h[n-k]$  does not yet reach the leftmost sample of  $x[k]$ . So the dot product is **zero**.
  - starting at  $n = 0$ , the rightmost sample of  $h[n-k]$  starts to overlap with the graph of  $x[k]$ , so we get **nonzero** in general.
  - this situation continues for  $N_1$  values of  $n$ , as the rightmost sample of  $h[n-k]$  progresses to overlap each sample in  $x[k]$ . Thus, in general we get a nonzero dot product for  $0 \leq n \leq N_1-1$ ; that is, for exactly  $N_1$  values of  $n$ .
  - then, at  $n = N_1$ , the rightmost sample of  $h[n-k]$  hangs over the right edge of the graph of  $x[k]$ . But we still get a nonzero dot product in general.
  - at  $n = N_1+1$ , the rightmost **two** samples of  $h[n-k]$  hang over the right edge of the graph of  $x[k]$ , but the dot product is still nonzero in general.
  - this situation continues until **all but one sample** of  $h[n-k]$  hang over. After that, the graph of  $h[n-k]$  is entirely past the graph of  $x[k]$  and the dot product is again zero.
- So, counting this up, we see that in general the dot product can be nonzero one time for each sample in  $x[k]$  and one time for each sample in  $h[n-k]$  **except the last one...** because once the last one hangs over there is no overlap.
- So, the convolution of two finite-length sequences with lengths  $N_1$  and  $N_2$  has a length that is given by  $N_1 + N_2 - 1$ .

- Matlab:

```
>> xn = [1 2 3 4];  
>> hn = [0.25 0.5 0.25];  
>> yn = conv(xn, hn);  
>> length(yn)  
ans = 6  
>> stem([0:5], yn);
```



- To re-compute this same example using the 1D DFT and circular convolution, we need to zero pad both sequences to a length of at least  $N_1 + N_2 - 1 = 6$ :

```
>> xprime = [xn zeros(1,2)];  
>> hprime = [hn zeros(1,3)];  
>> yprime = ifft(fft(xprime).*fft(hprime));  
>> max(abs(yn-yprime))  
ans = 2.2204e-16  
>> % output signal is the same as before
```

# Notes on this 1D Example

- You can zero pad to a length  $> N_1 + N_2 - 1$ . Especially in 2D, the DFT may run faster if the padded length is a power of 2.
  - if you do this, the linear convolution still has length  $N_1 + N_2 - 1$ . It is contained in the first (leftmost)  $N_1 + N_2 - 1$  samples of the result sequence.
- Interpreting  $y[n]$  as the weighted 3-point average of  $x[n]$ :
  - There are **edge effects** on both ends of  $y[n]$ : zeros are averaged in where the graph of  $h[n-k]$  hangs over the graph of  $x[k]$  ( $n = 0, 1, 4, 5$ ).
  - Because the filter is **causal**, it is **not** a centered average. For example,  $y[4] = 0.25x[2] + 0.5x[3] + 0.25x[4]$ .
  - In other words, there is **delay** (the filter introduces nonzero phase).

# Notes 1D Example...

- Often, the edge effects are not a concern in 1D applications.
  - the impulse response  $h[n]$  is often short compared to the signal  $x[n]$ .
  - for example, applying a 13-point average to one minute of digital audio at 44 kHz:  $x[n]$  has length  $2.64 \times 10^6$ . But the edge effects impact only the first 7 samples and the last 7.
- A reasonable time delay is also okay in many 1D applications:
  - audio CD player, MP3 player
- A reasonable time delay **may** be of no concern in some image/video applications:
  - DVD/Blu ray player, cable set top box, youtube...

# Extending these Ideas to 2D

- Let  $\mathbf{I}_1$  be a  $M_1 \times N_1$  digital image.
- Let  $\mathbf{I}_2$  be a  $M_2 \times N_2$  digital image.
- As before, we desire the **linear** convolution  $\mathbf{J} = \mathbf{I}_1 * \mathbf{I}_2$ .
- Extending the ideas we just saw in 1D, we can zero pad both images to a size of  $M \times N$ , where  $M = M_1 + M_2 - 1$  and  $N = N_1 + N_2 - 1$ .
- Call the zero padded images  $\mathbf{I}'_1$  and  $\mathbf{I}'_2$ .
- Then, the 2D linear convolution is given by

$$\mathbf{J} = \mathbf{I}'_1 \circledast \mathbf{I}'_2 = \text{IFFT}_{M \times N} \left\{ \text{FFT}_{M \times N} [\mathbf{I}'_1] \otimes \text{FFT}_{M \times N} [\mathbf{I}'_2] \right\}.$$

- It has size  $M \times N$ , i.e.,  $(M_1 + M_2 - 1) \times (N_1 + N_2 - 1)$ .

- In general, the 2D FFT is faster if the number of rows and the number of columns are both powers of 2.
- So there is a tradeoff between using the minimum size FFTs that will eliminate the wraparound effect and using larger FFTs where the number of rows/cols are powers of 2.
- This tradeoff depends on the details of the particular FFT implementation and can be complicated to analyze.
- In rapid prototyping and in research/exploratory work, it is common to skip the analysis and zero pad both images up to the next power of 2, both horizontally and vertically.
- As before, call the zero padded size  $M \times N$ , which are now the next powers of 2 that are  $\geq (M_1 + M_2 - 1)$  and  $(N_1 + N_2 - 1)$ .
- We then get  $\mathbf{J}' = \mathbf{I}'_1 \circledast \mathbf{I}'_2 = \text{IFFT}_{M \times N} \left\{ \text{FFT}_{M \times N} [\mathbf{I}'_1] \otimes \text{FFT}_{M \times N} [\mathbf{I}'_2] \right\}$ .
- The linear convolution  $\mathbf{J} = \mathbf{I}_1 * \mathbf{I}_2$  still has size
 
$$(M_1 + M_2 - 1) \times (N_1 + N_2 - 1).$$
- It is contained in the top  $N_1 + N_2 - 1$  rows and left  $M_1 + M_2 - 1$  columns of the image  $\mathbf{J}'$ .

# Some Notes on Zero Padding

- So why did it say on page 5.36 to zero pad both images to size  $2M \times 2N$  ?
  - If the original images  $I_1$  and  $I_2$  are the same size (i.e., if both are  $M \times N$ ), then the **minimum** FFT size to eliminate the wraparound effect is  $(2M-1) \times (2N-1)$ .
  - **Often**, the original sizes  $M$  and  $N$  **are** powers of 2. In this case,  $2M \times 2N$  is the smallest FFT size that **both** eliminates the wraparound effect **and** gives the “power of 2” speedup in the FFT computations.
- If the original sizes  $M$  and  $N$  are **not** powers of 2, then it will generally be slightly faster to zero pad to a size of  $(2M-1) \times (2N-1)$  instead of  $2M \times 2N$ .

# Practical Considerations

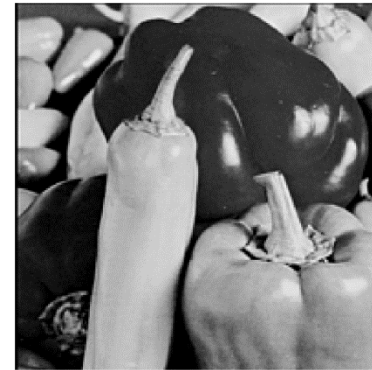
- When you convolve two images that each have 8-bit pixels, the result image generally has pixels that are **outside** the range [0, 255].
  - For example, if  $\mathbf{I}_1$  and  $\mathbf{I}_2$  are both  $256 \times 256$  and are both constant images with  $\mathbf{I}_1(m, n) = \mathbf{I}_2(m, n) = 255$ , then the maximum pixel value in  $\mathbf{J} = \mathbf{I}_1 * \mathbf{I}_2$  will be  $256^2 \cdot 255^2 = 4.2615 \times 10^9$  !
  - So you have to do a full-scale contrast stretch before you can look at the result image (or do practically **anything** useful with it).
- When computing the linear (or circular) convolution of two **real** images using FFTs, numerical roundoff errors may cause the result image to have a small but nonzero imaginary part.
  - If  $\mathbf{I}_1$  and  $\mathbf{I}_2$  are both real, then so is the convolution. You should discard any nonzero imaginary part.

# 2D Linear Convolution by FFT: Example 1

- Use FFTs to compute the linear convolution of:



$I_1$   
256 × 256 Lena



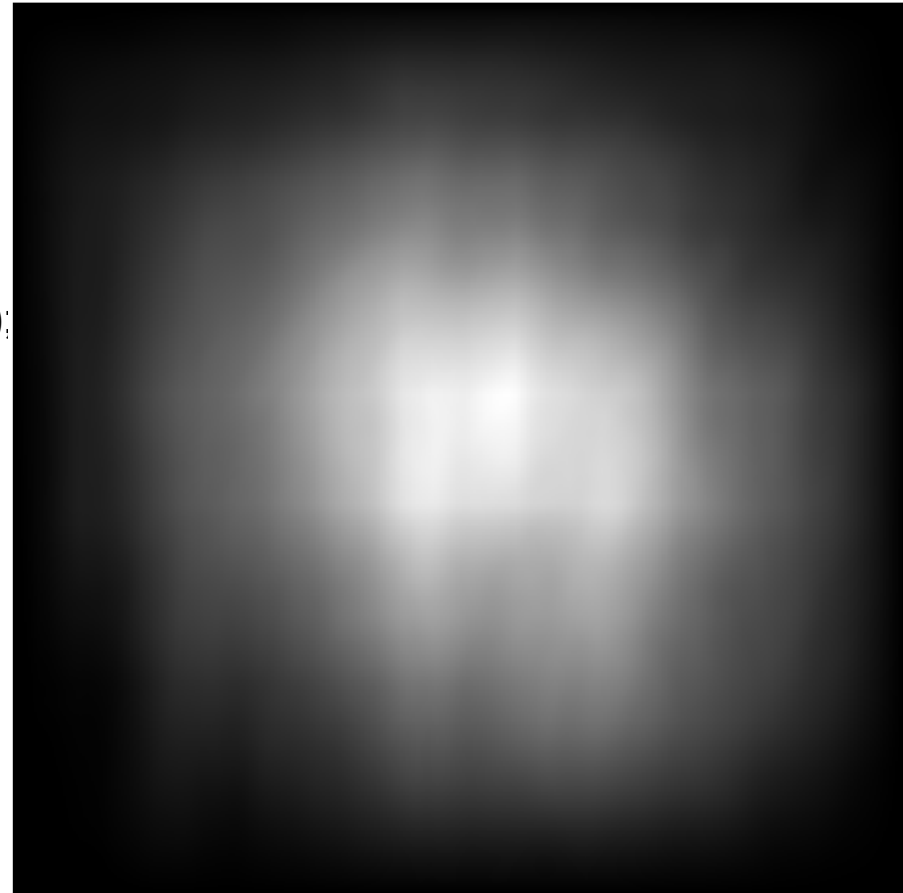
$I_2$   
200 × 200 Peppers

- Zero pad both images to size  $256 + 200 - 1 = 455$  rows/cols.

- Matlab:

```
[Lena,junk] = fread(fidLena,[256,256],'uchar');
[Peppers,junk] = fread(fidPeppers,[256,256],'uchar');
Lena = Lena'; Peppers = Peppers';
Peppers200 = Peppers(1:200,1:200);
Padsz = 256 + 200 - 1;
ZPLena = zeros(Padsz,Padsz);
ZPLena(1:256,1:256) = Lena;
ZPPeppers200 = zeros(Padsz,Padsz);
ZPPeppers200(1:200,1:200) = Peppers200;
J = real(ifft2(fft2(ZPLena) .* fft2(ZPPeppers200)));
JJ = stretch(J);
```

Result:



- Check:

```
J2 = conv2(Lena,Peppers200);
JJ2 = stretch(J2);
% exact same result
```

## Comments on Example 1:

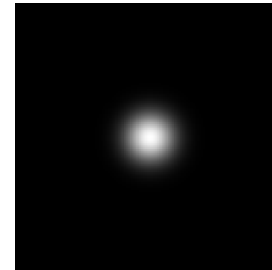
- The result is correct...
- But not very useful!
- In most cases, we aren't really interested in convolving two "typical" images directly.
- Rather, what we usually **are** interested in doing with linear convolution is applying a 2D LTI filter that will modify the image in a predictable way.
- This implies that we must **design** the filter:
  - design  $\mathbf{I}_2$  to be the filter impulse response, or
  - design  $\tilde{\mathbf{I}}_2$  to be the filter frequency response.

## 2D Linear Convolution by FFT: Example 2

- Use FFTs to apply a  $128 \times 128$  LTI Gaussian low pass filter.
- Impulse resp:  $H(r) = e^{-r^2 / (2\sigma^2)}$ , where  $r = \sqrt{(m - 64)^2 + (n - 64)^2}$ .



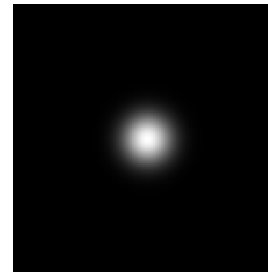
**I<sub>1</sub>**  
256 × 256 Lena



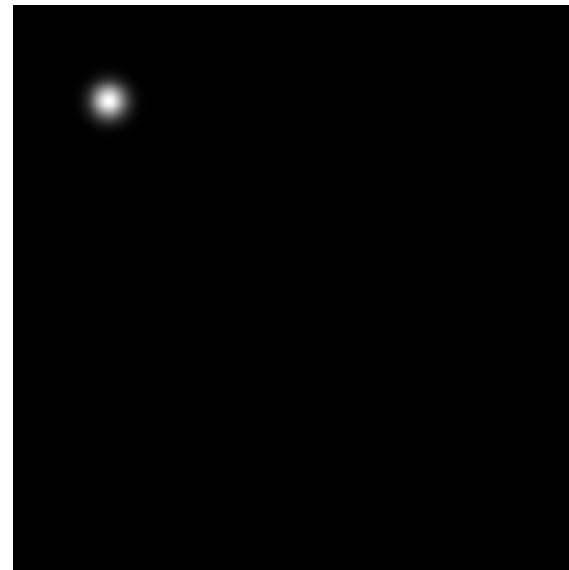
**I<sub>2</sub> = H**  
128 × 128 Gaussian  
 $\sigma = 8$

- Zero pad both images to size  $256 + 128 - 1 = 383$  rows/cols.

```
[COLS,ROWS] = meshgrid(0:127,0:127);  
COLS = COLS - 64;  
ROWS = ROWS - 64;  
R = sqrt(ROWS .^ 2 + COLS .^ 2);  
sigma = 8;  
h = exp(-R.^2/(2*sigma^2));
```



```
Padsiz = 256 + 128 - 1;  
ZPLena = zeros(Padsiz,Padsiz);  
ZPLena(1:256,1:256) = Lena;  
ZPh = zeros(Padsiz,Padsiz);  
ZPh(1:128,1:128) = h;
```



```
y1 = real(iff2(fft2(ZPLena) .* fft2(ZPh)));  
yy1 = stretch(y1);
```



- Usually, we want the result to be the same size as the original image, so:

```
y3 = y1(1:256,1:256);  
yy3 = stretch(y3);
```



## Comments on Example 2:

- This result is more useful than the one in Example 1!
  - In Example 2, we **did** get a low pass filtered Lena!
- But there are two main problems:
  - the filtered Lena image (i.e., the convolution result) is not the same size as the original.
  - since the main lobe of  $\mathbf{H}$  is not centered around pixel  $(0,0)$ , the filter introduces a nontrivial phase shift.
  - In other words, the convolution result has a **spatial shift** relative to the original image.
- One approach to fixing these two problems is to simply “undo” the spatial shift and crop the result to the size of the original image.
- To understand this, we once again go back to 1D...

# 1D Linear Convolution Again

- Back on page 5.38, we had the 1D LTI causal 3-point weighted average filter with

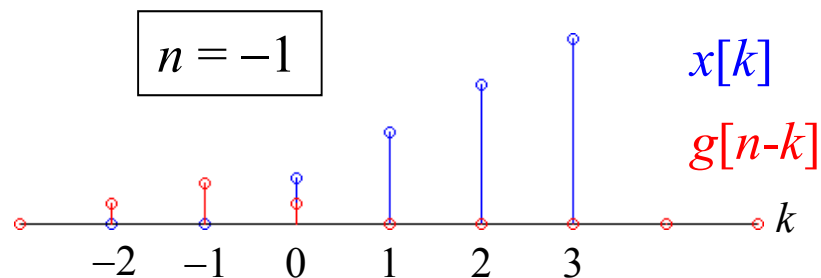
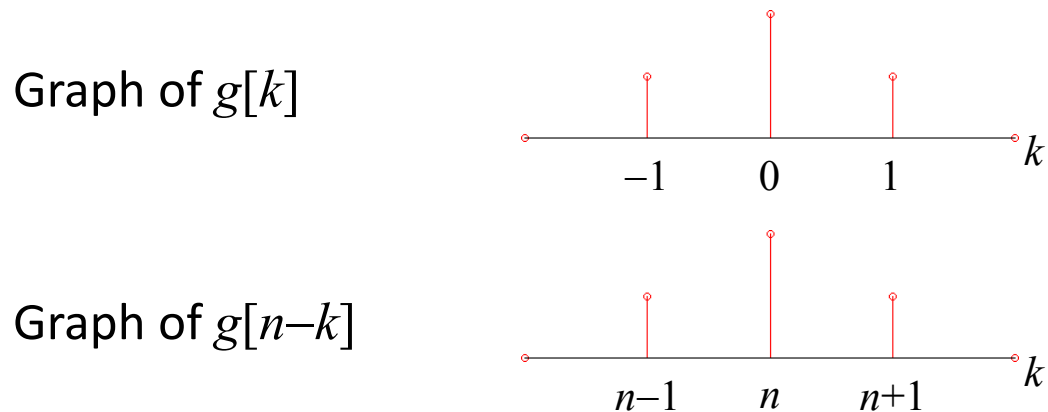
$$h[n] = \frac{1}{4}\delta[n] + \frac{1}{2}\delta[n-1] + \frac{1}{4}\delta[n-2].$$

- By shifting  $h[n]$  in time, we can turn this into a non-causal 3-point weighted average that **is centered**, so that it has zero phase and introduces no phase shift:

$$g[n] = h[n+1] = \frac{1}{4}\delta[n+1] + \frac{1}{2}\delta[n] + \frac{1}{4}\delta[n-1].$$

- Because  $g[n]$  is real and even, it's DTFT is also **real** and even.
  - This means that the phase  $\angle G(e^{j\omega})$  is **identically zero**.
  - So the filter  $G$  is not causal, but it introduces no phase shift between the input signal and the output signal.

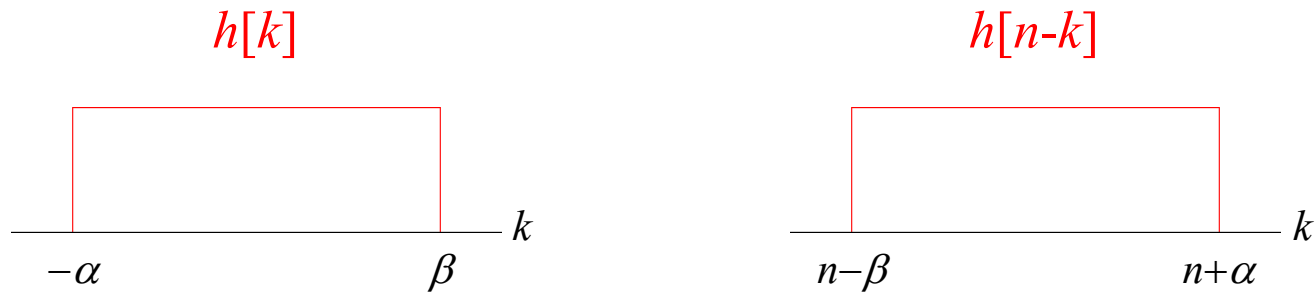
- Recall:  $x[n] = \delta[n] + 2\delta[n-1] + 3\delta[n-2] + 4\delta[n-3]$ .



- Now, the first nonzero overlap occurs when  $n = -1$  instead of  $n = 0$ .
- The linear convolution  $y[n] = x[n] * g[n]$  still has length  $N_1 + N_2 - 1 = 6$ .
- But now this corresponds to times  $n = -1$  to 4 **instead** of  $n=0$  to 5.
- In Matlab, **everything is still exactly the same as before.**

- So even though everything is exactly the same in Matlab, we have to remember that this time the first array element of  $y_n = \text{conv}(x_n, h_n)$  is for  $n = -1$ , not  $n=0$ !
- The length 4 weighted 3-point average signal corresponding to  $x[n]$  is obtained by taking  $y_n$  for  $n=0$  to 3 **only**.
- In Matlab, this is  $y_n(2 : 5)$ .
- Why?
  - The convolution  $y[n]$  has length 6 and goes from  $n = -1$  to  $n=4$ .
  - So, in Matlab, the first element of  $y_n$  is for  $n = -1$ , the second element is for  $n=0$ , and so on...
- Notice that  $y_n(2 : 5)$  is the same size as  $x[n]$  and is not shifted relative to  $x[n]$ .

- More generally, suppose we have a 1D LTI filter  $H$  with an impulse response  $h[n]$  that is nonzero from  $n = -\alpha$  to  $n = +\beta$ :



- If  $x[n]$  starts at  $n=0$ , then the **first** nonzero overlap in the linear convolution  $y[n] = x[n] * h[n]$  will occur at  $n = -\alpha$ .
- So the  $\alpha+1$ 'st nonzero sample of the convolution will be the one that corresponds to  $n=0$ .
- Thus, in the Matlab array `yn=conv(xn, hn)`, it is the element `yn(alpha+1)` that corresponds to  $n=0$ .
- To obtain an output sequence the same length as the input that is **not** shifted, we keep `length(x[n])` samples from `yn` starting at index  $\alpha+1$ .
- In the example on pages 5.56-5.57,  $\alpha = 1$  and `length(x[n]) = 4`, so we kept `yn(2:5)`.

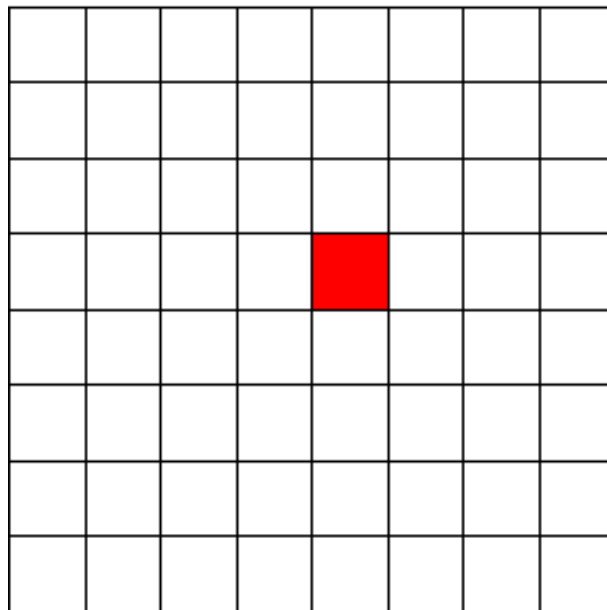
# Extension to 2D

- Let  $\mathbf{I}$  be a  $M_1 \times N_1$  digital image.
- We will design a  $M_2 \times N_2$  digital impulse response  $\mathbf{H}$ .
  - This time, the spatial origin  $(0,0)$  will **not** be located in the upper left corner of the  $\mathbf{H}$  image.
  - Let  $(m, n)$  indicate the **true** col/row spatial coordinates in the mathematical formula for the impulse response function  $\mathbf{H}$ .
  - Let  $(p, q)$  indicate zero-based col/row indexing in the  $\mathbf{H}$  image.
  - Let  $(r, s)$  indicate Matlab row/col indexing in the  $\mathbf{H}$  image.
- We desire the **linear** convolution  $\mathbf{J} = \mathbf{I} * \mathbf{H}$ .
- In computing the image pixels  $H(p, q)$ , suppose we place the mathematical origin  $(m, n) = (0,0)$  at  $(p, q) = (p_0, q_0)$ .
- Then we have horizontal offset  $\alpha_x = p_0$  and vertical offset  $\alpha_y = q_0$ .
- The upper left pixel of the  $\mathbf{H}$  image will be for  $(m, n) = (-p_0, -q_0)$ .
- The first pixel to **keep** will be  $(m, n) = (0,0)$ ,  $(p, q) = (p_0, q_0)$ , and Matlab row/col  $(r, s) = (q_0+1, p_0+1)$ .

- For example, suppose  $M_2 = N_2 = 8$  and  $(p_0, q_0) = (4,3)$ :

$p=4, m=0$ , Matlab column  $s=5$ .

the **H**  
image



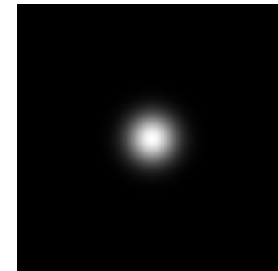
$q=3, n=0$ , Matlab row  $r=4$ .

- The first pixel of the convolution to keep will be the one at  $(p, q) = (4,3)$ , or Matlab row/col  $(r, s) = (4,5)$ .

## 2D Linear Convolution by FFT: Example 3

- Use FFTs to apply a  $128 \times 128$  LTI Gaussian low pass filter.
- The result must be the same size as the original image and must not be shifted relative to the original.
- We place the origin  $(m, n) = (0,0)$  at  $(p, q) = (64,64)$ .
- Impulse response:

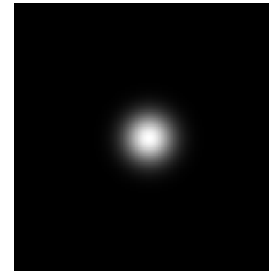
$$\begin{aligned} H(m, n) &= \exp\left[-\frac{m^2 + n^2}{2\sigma^2}\right] \\ &= \exp\left[-\frac{(p - 64)^2 + (q - 64)^2}{2\sigma^2}\right] \end{aligned}$$



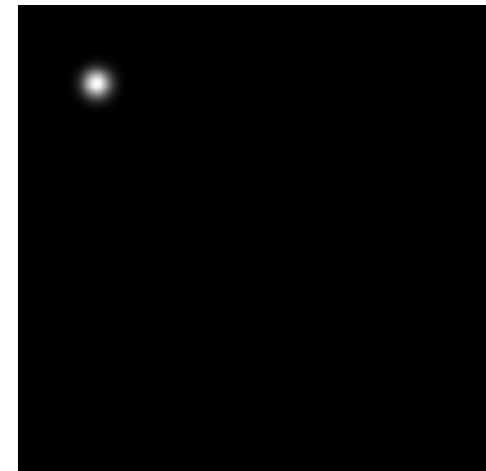
$H(p, q)$   
 $128 \times 128$  Gaussian  
 $\sigma = 8$

- We have  $(p_0, q_0) = (64,64)$ . So the first pixel to keep is  $(p, q) = (64,64)$ , which is Matlab row/col  $(r, s) = (65,65)$ .

```
[COLS,ROWS] = meshgrid(0:127,0:127);  
sigma = 8;  
h = exp(-((ROWS-64).^ 2 + (COLS-64).^ 2)/(2*sigma^2));
```



```
Padsiz = 256 + 128 - 1;  
ZPLena = zeros(Padsiz,Padsiz);  
ZPLena(1:256,1:256) = Lena;  
ZPh = zeros(Padsiz,Padsiz);  
ZPh(1:128,1:128) = h;
```



```
y1 = real(ifft2(fft2(ZPLena) .* fft2(ZPh)));  
y3 = stretch(y1(65:320,65:320));
```



# A Common Approach

- For a  $M \times N$  digital image  $\mathbf{I}$ , it is common to also design the impulse response  $\mathbf{H}$  to be  $M \times N$ .
- Assuming  $M$  and  $N$  are even, we place the spatial origin  $(m, n) = (0,0)$  at  $(p, q) = (M/2, N/2)$ , which is Matlab row/col  $(r, s) = (N/2 + 1, M/2 + 1)$ .
- This gives offsets  $p_0 = M/2$  and  $q_0 = N/2$ .
- Usually,  $\mathbf{I}$  and  $\mathbf{H}$  are both zero padded to size  $2M \times 2N$  to obtain the zero padded images  $\mathbf{I}'$  and  $\mathbf{H}'$ .
- Then,

$$\mathbf{J}' = \mathbf{I}' \circledast \mathbf{H}' = \text{IFFT}_{2M \times 2N} \left\{ \text{FFT}_{2M \times 2N} [\mathbf{I}'] \otimes \text{FFT}_{2M \times 2N} [\mathbf{H}'] \right\}.$$

- The desired  $M \times N$  image  $\mathbf{J}$  is then taken from  $\mathbf{J}'$  starting with pixel  $(p, q) = (M/2, N/2)$ .
- In Matlab row/col coordinates, this is

$$\mathbf{J} = \mathbf{J}' \left( \frac{N}{2} + 1 : \frac{3N}{2}, \frac{M}{2} + 1 : \frac{3M}{2} \right).$$

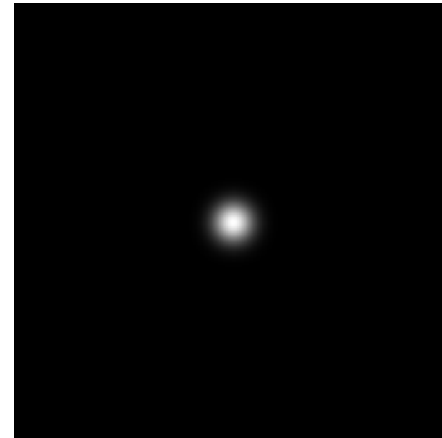
## 2D Linear Convolution by FFT: Example 4

- Input image  $\mathbf{I}$  is  $256 \times 256$  Lena.
- Use FFTs to apply a  $256 \times 256$  Gaussian low pass LTI filter with space constant  $\sigma = 8$ .
  - We have  $M = N = 256$ .
  - For  $\mathbf{H}$ , we place the origin  $(m, n) = (0,0)$  at  $(p, q) = (128,128)$ . This is Matlab row/col  $(r, s) = (129,129)$ .
  - This gives horizontal/vertical offsets  $p_0 = q_0 = 128$ .
  - With  $\sigma = 8$ , we compute the  $\mathbf{H}$  image according to

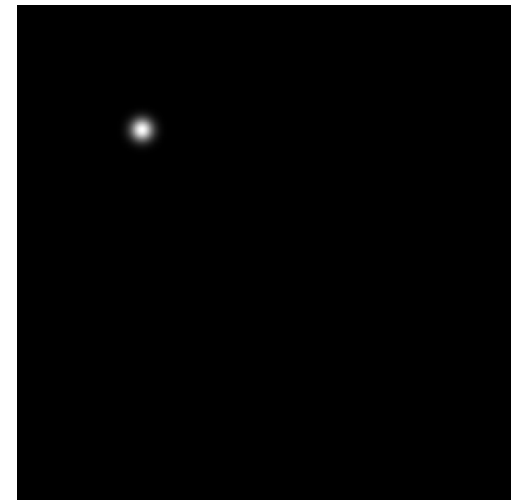
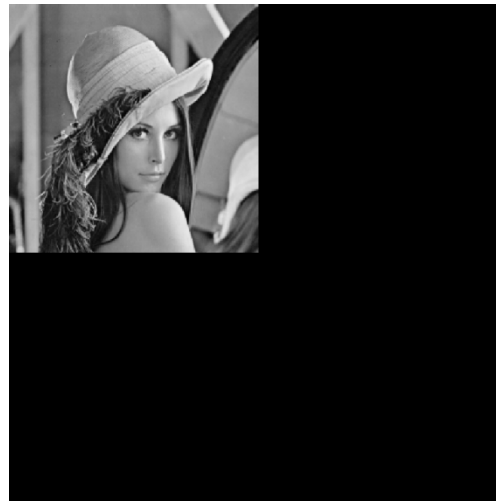
$$H(p, q) = \exp\left[-\frac{m^2 + n^2}{2\sigma^2}\right] = \exp\left[-\frac{(p-128)^2 + (q-128)^2}{2\sigma^2}\right]$$

- For Matlab, remember that  $(\text{row}, \text{col}) = (r, s) = (q+1, p+1)$ .
- we zero pad both images to size  $2M \times 2N = 512 \times 512$ .
- The desired  $256 \times 256$  image  $\mathbf{J}$  is then taken from  $\mathbf{J}'$  starting with pixel  $(p, q) = (128,128)$ ; for Matlab this is  $(r, s) = (129,129)$ .

```
[COLS,ROWS] = meshgrid(0:255,0:255);  
sigma = 8;  
h = exp(-((ROWS-128).^ 2 + (COLS-128).^ 2)/(2*sigma^2));
```



```
Padsizе = 512;  
ZPLena = zeros(Padsizе,Padsizе);  
ZPLena(1:256,1:256) = Lena;  
ZPh = zeros(Padsizе,Padsizе);  
ZPh(1:256,1:256) = h;
```



```
y = real(ifft2(fft2(ZPLena) .* fft2(ZPh)));  
yy = stretch(y(129:384,129:384));
```

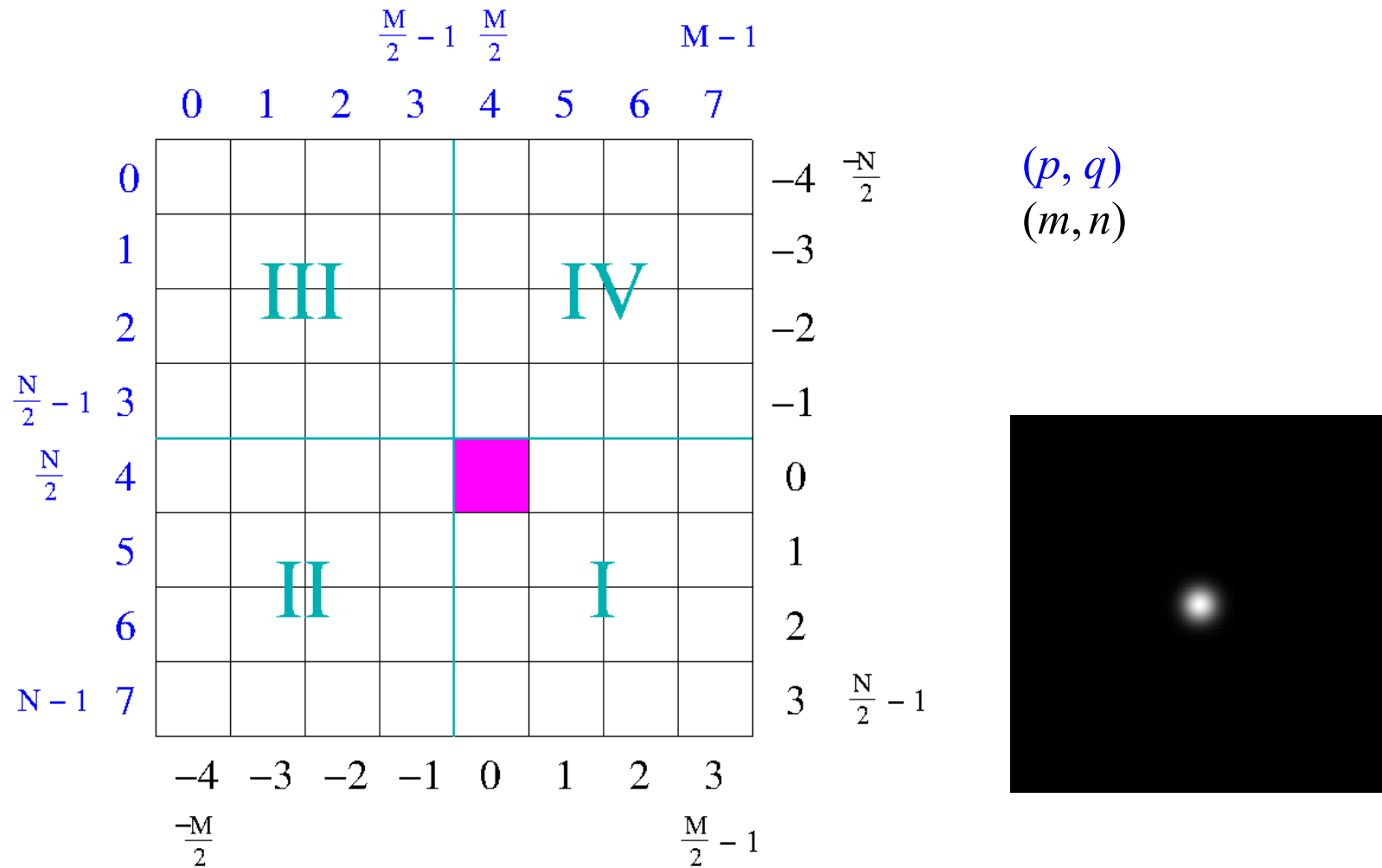


# Zero Phase Impulse Response

- Another way to prevent the output image from being shifted relative to the original is to design a true zero phase impulse response directly.
- There are two main requirements for this:
  1. The impulse response must be real and even:
    - $H(-m, -n) = H(m, n)$
    - $\text{Im}[H(m, n)] = 0$
  2. The spatial origin  $(m, n) = (0,0)$  must be placed on the upper left pixel  $(p, q) = (0,0)$  of the impulse response image **H**.
- Because the DFT implies that the image is periodic, this means that the main lobe of the impulse response will be split across the four corners of the image **H**.
- As we will see, this creates some issues when we zero pad the **H** image to implement linear convolution with FFTs.

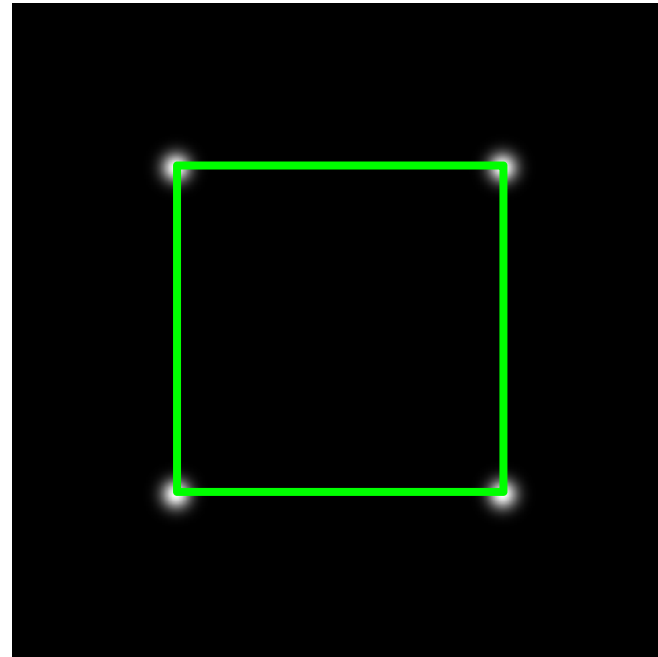
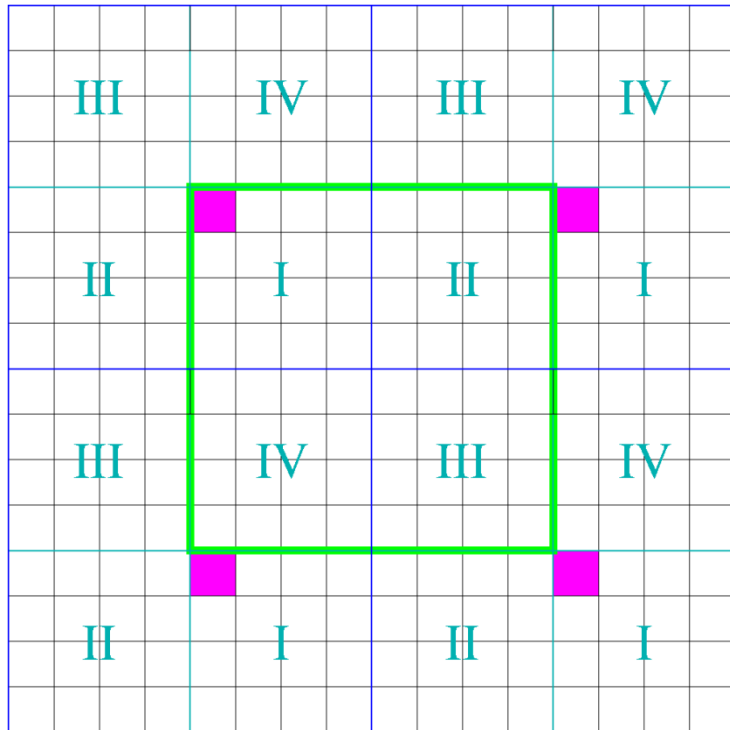
- To understand how this works, let's think about it in terms of Example 4 on pages 5.64-5.65.
- Input image  $\mathbf{I}$  is  $256 \times 256$ . So  $M = \text{\#cols} = 256$  and  $N = \text{\#rows} = 256$ .
- We want to use FFTs to apply a  $256 \times 256$  Gaussian low pass LTI filter with space constant  $\sigma = 8$ .
- Recall: we have **three** coordinate systems:
  - $(p, q)$ : zero based col/row in the  $\mathbf{H}$  image.
  - $(m, n)$ : “true” col/row that we use in the math to compute the  $\mathbf{H}$  image.
  - $(r, s)$ : Matlab row/col:  $(r, s) = (q+1, p+1)$ .
- To design a true zero-phase impulse response image  $\mathbf{H}$ , we need for the  $(m, n)$  coordinate system and the  $(p, q)$  coordinate system to “line up.”
- This means that the offsets are  $p_0 = q_0 = 0$ .
- It will be useful to think of the  $\mathbf{H}$  image as being divided into quadrants.
- Let's start by looking at a small  $\mathbf{H}$  image with  $M = N = 8$ .

- Here is the division of the **H** image into quadrants I-IV, with the coordinate systems as they were in Example 4:



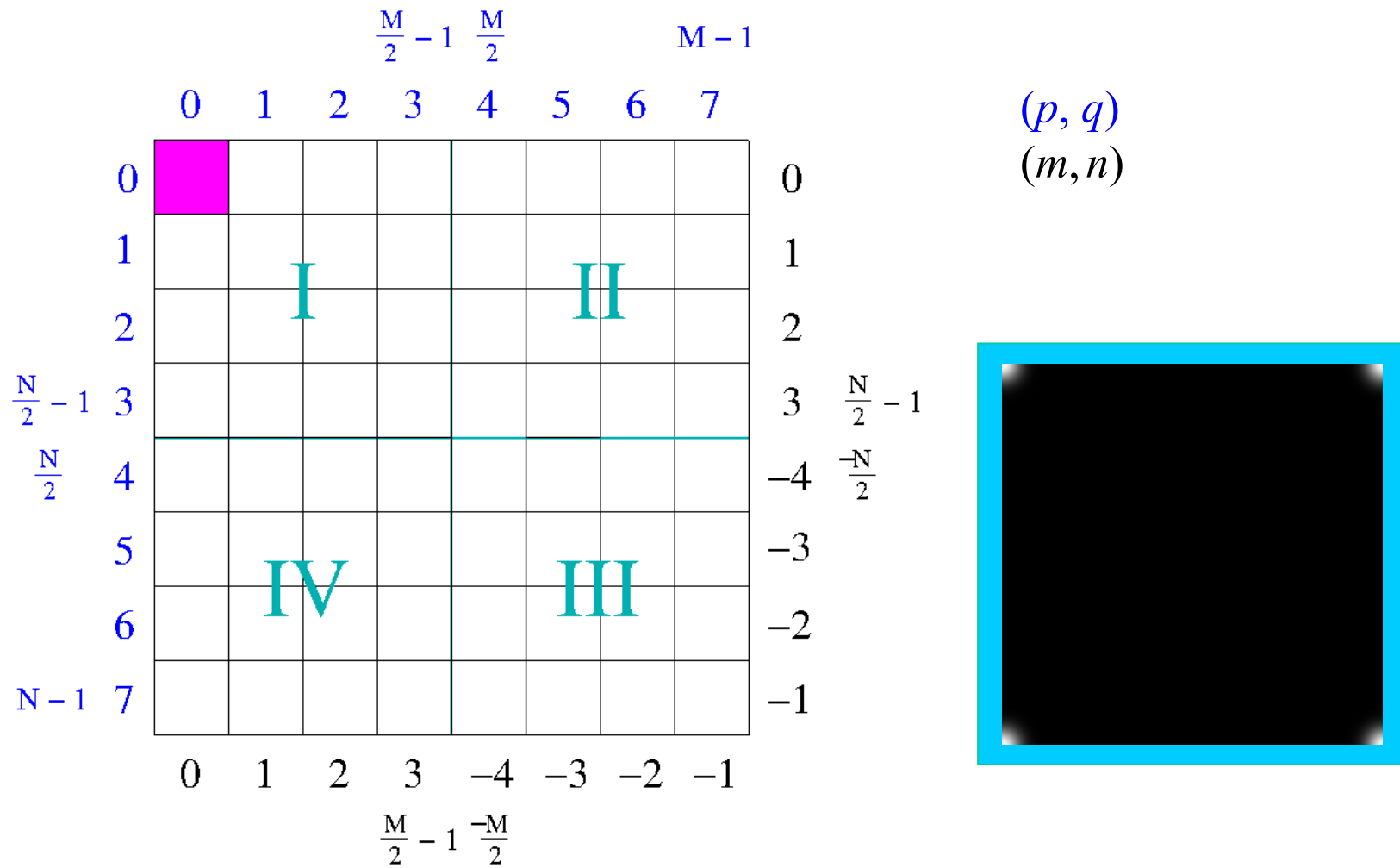
- The DFT implies that this is one period of a periodic image.

- Here is the periodic extension:



- The part in the green box is a zero phase image, with  $(m, n) = (0, 0)$  in the upper left corner.
- Notice that this splits the main lobe of the impulse response across the four corners of the image in the green box.
- To get this image, we have to permute the quadrants around...

- Here is the zero phase image in the green box on the last page:



- You can write a program to design this directly.
- Or you can do the design as in Example 4, and then shift the quadrants around:
  - Can write a program to do this directly.
  - Or take the FFT, multiply by  $(-1)^{u+v}$ , then invert.
  - In Matlab, you can call `fftshift` on the **H image**.

```
[COLS,ROWS] = meshgrid(0:255,0:255);
sigma = 8;
h = exp(-((ROWS-128).^2 + (COLS-128).^2)/(2*sigma^2));
h2 = fftshift(h);
```

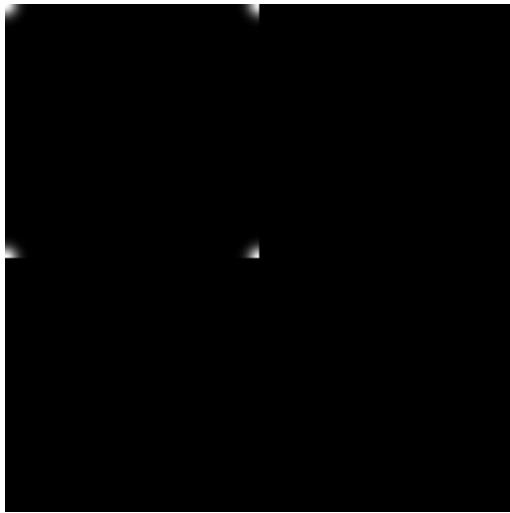
- If you take the FFT, multiply it pointwise with the FFT of the image, and invert, then you will get circular convolution **without any phase shift**.

```
ifft2(fft2(Lena).*fft2(h2));
```

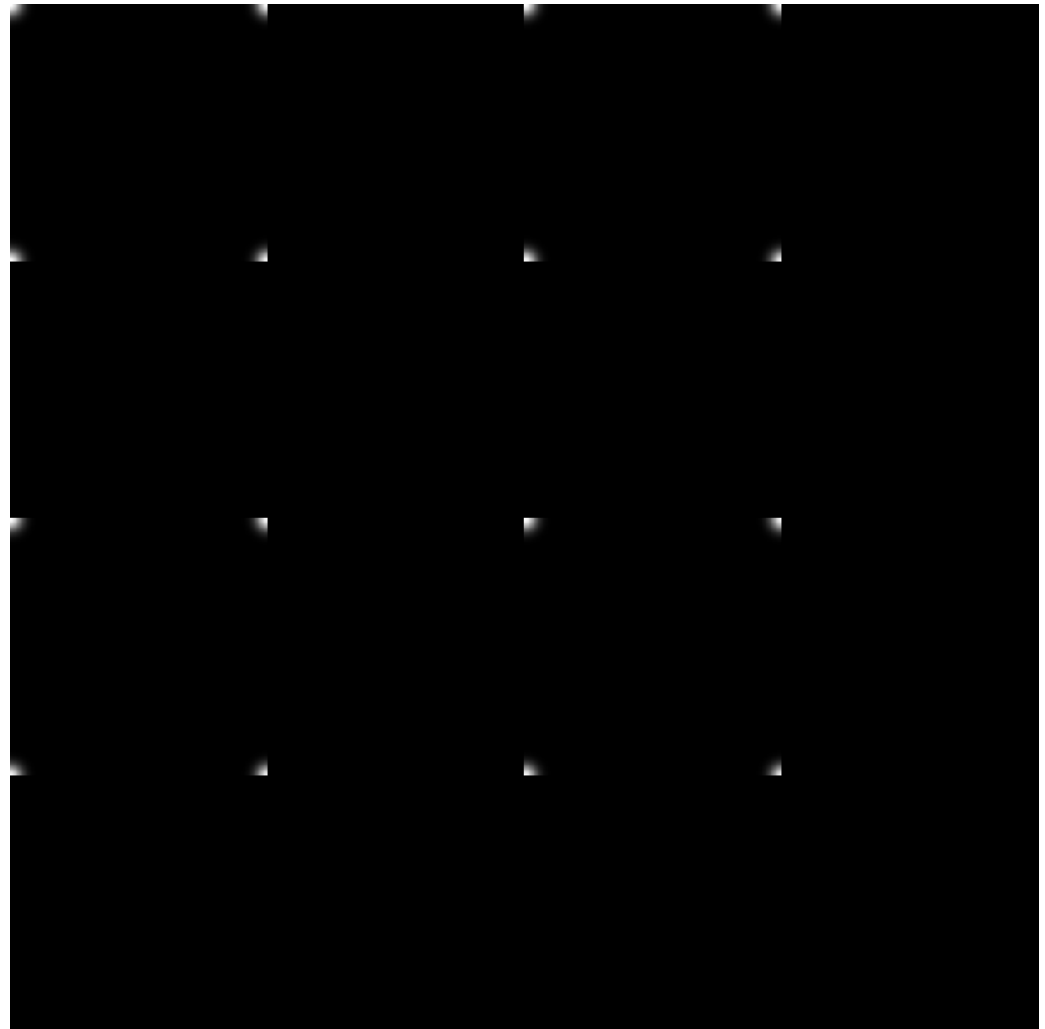


- However, our previous zero padding scheme will not work with the zero phase impulse response.

zero padded

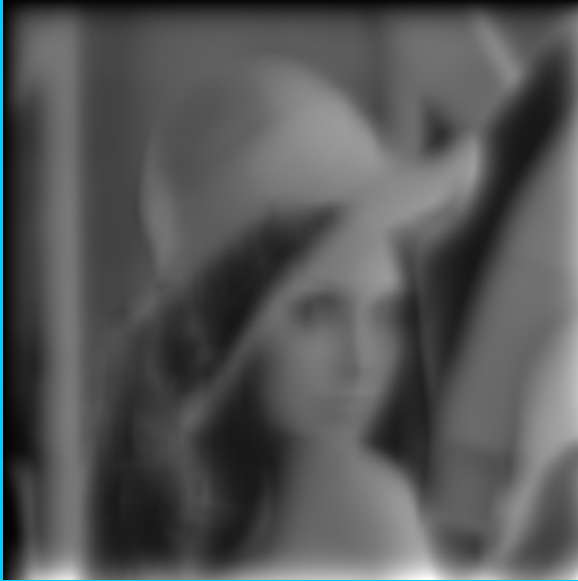


periodic extension

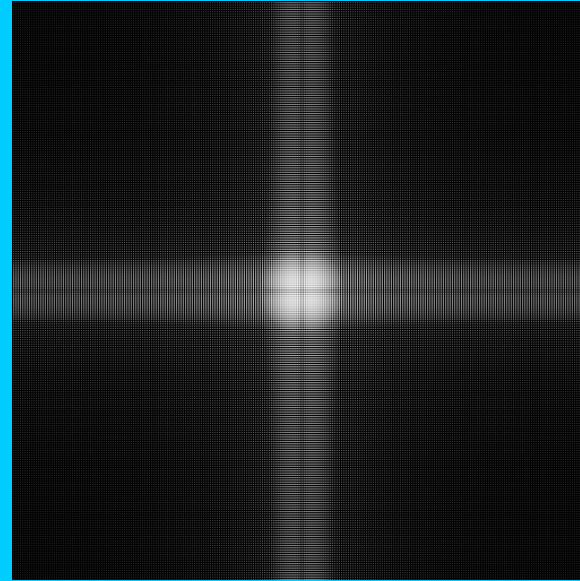


- The quadrants of the main lobe don't fit together correctly.
- This is **not** the filter that we set out to design.
- Will give **incorrect** results.

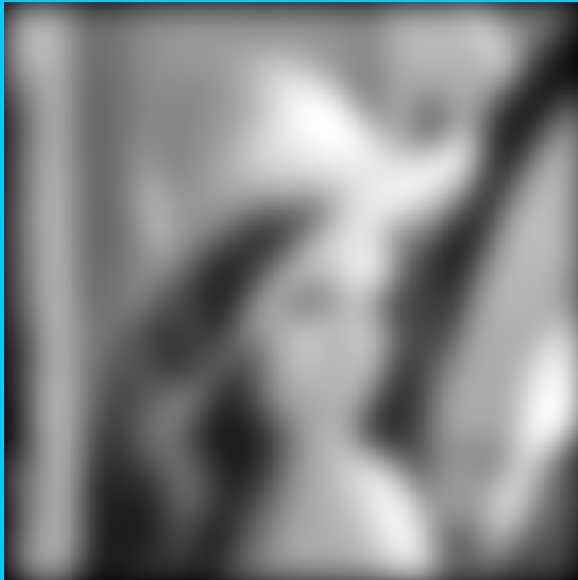
**Incorrect** filtered image



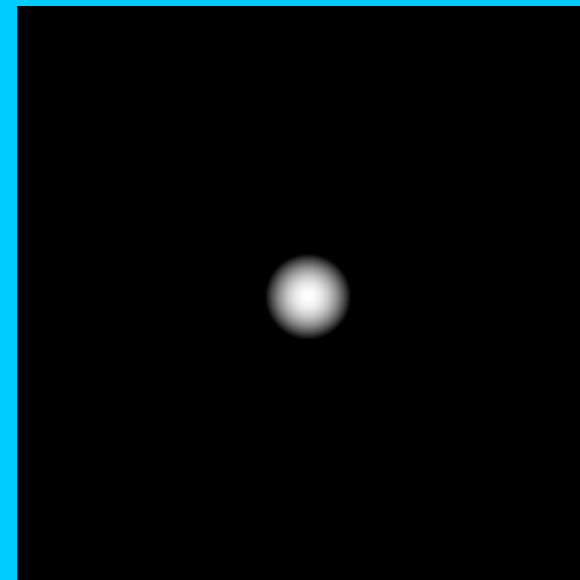
**Incorrect** frequency response



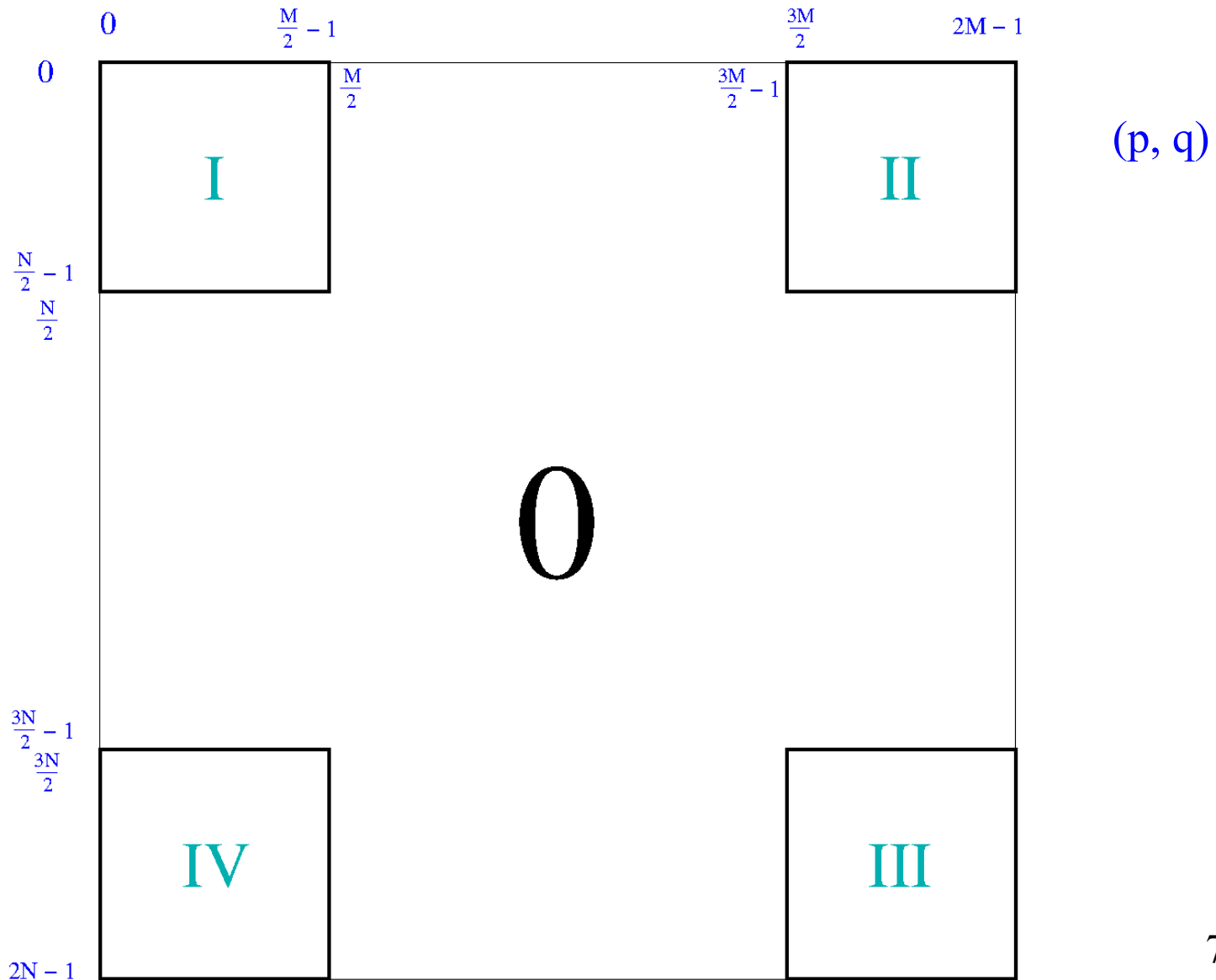
**Correct** filtered image



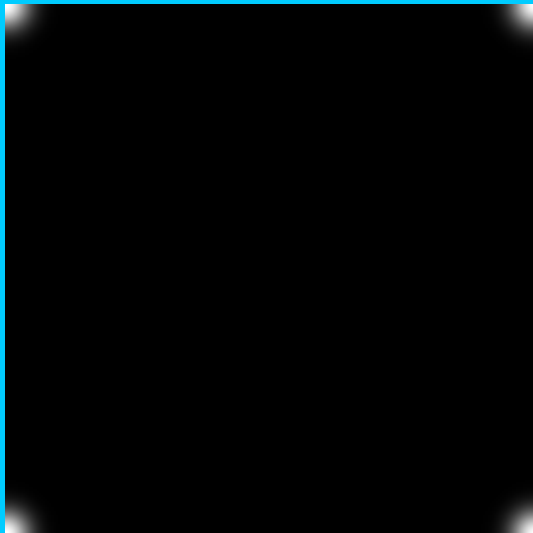
**Correct** frequency response



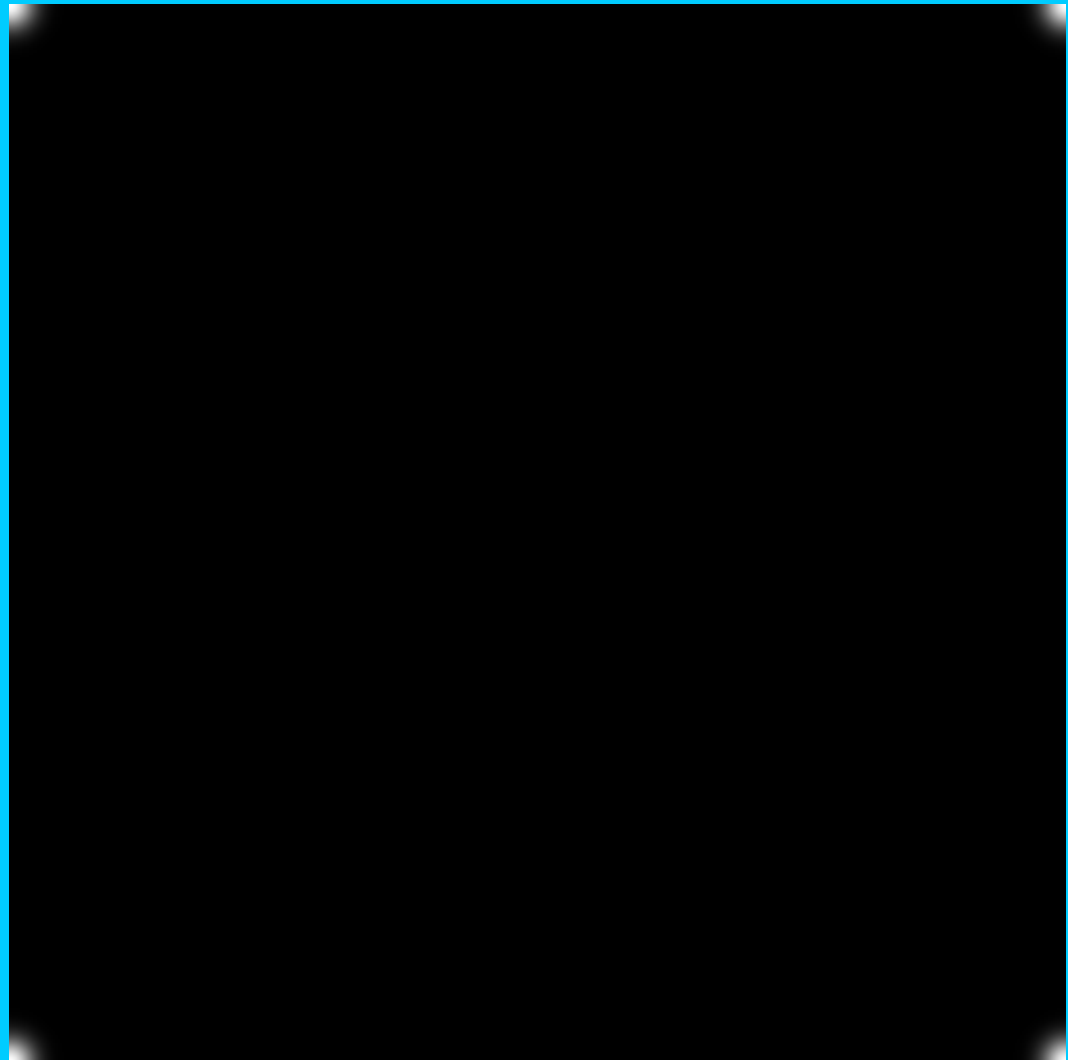
- To get the correct zero padded filter image, you must place the four quadrants of the zero phase impulse response in a  $2M \times 2N$  zero image:



zero phase  
impulse  
response



correctly zero padded



## 2D Linear Convolution by FFT: Example 5

- Use FFTs to apply a  $256 \times 256$  true zero phase Gaussian low pass LTI filter with space constant  $\sigma = 8$ .

```
ZPLena = zeros(512,512);
ZPLena(1:256,1:256) = Lena;
[COLS,ROWS] = meshgrid(0:255,0:255);
h = exp(-((ROWS-128).^ 2 + (COLS-128).^ 2)/(2*sigma^2));
h2 = fftshift(h);
h2ZP = zeros(512,512);
h2ZP(1:128,1:128) = h2(1:128,1:128);
h2ZP(1:128,385:512) = h2(1:128,129:256);
h2ZP(385:512,1:128) = h2(129:256,1:128);
h2ZP(385:512,385:512) = h2(129:256,129:256);
yz = ifft2(fft2(ZPLena).*fft2(h2ZP));
yz = yz(1:256,1:256);
```



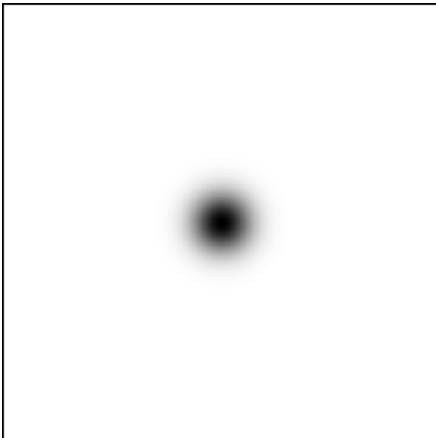
# Frequency Response Design

- Often, it is desirable to design the frequency response  $\tilde{\mathbf{H}}$  directly (instead of the impulse response  $\mathbf{H}$ ).
- Normally, the frequency response will be designed at a size of  $M \times N$  (same size as the image) so that the frequencies will be the same as those in the DFT of the image.
- The usual way of doing this is to fill in the  $M \times N$  DFT array  $\tilde{\mathbf{H}}$  from a designed formula for the frequency response (this is called **frequency sampling**).
- However, to implement linear convolution by multiplication of DFTs, what we need is a  $2M \times 2N$  DFT array that can be pointwise multiplied with the DFT of the zero padded image.
- There is no simple analytical way to get this from the designed  $M \times N$  DFT array  $\tilde{\mathbf{H}}$ .

- Common procedure for direct design of the frequency response:
  1. Fill in the  $M \times N$  DFT array  $\tilde{\mathbf{H}}$  from the design equation.
    - Usually done with the **centered** DFT.
    - Then call fftshift to “un-center” it.
  2. Take the inverse FFT to get the zero phase  $M \times N$  impulse response  $\mathbf{H}$ .
  3. Then **either**:
    - Proceed as in Example 5, **or**
    - Shift the impulse response to place the origin  $(m,n) = (0,0)$  in the center at  $(p,q) = (M/2, N/2)$  and proceed as in Example 4.
      - In Matlab, the shifting can be done by calling fftshift.

## 2D Linear Convolution by FFT: Example 6

- Use frequency sampling to apply a high pass Gaussian LTI filter to the  $256 \times 256$  Lena image.
- Hi pass Gaussian design equation:  $\tilde{H}_D(U, V) = 1 - e^{-(U^2 + V^2)/2\sigma^2}$



*% Follow the steps on p. 5.78*

```
ZPLena = zeros(512,512);
```

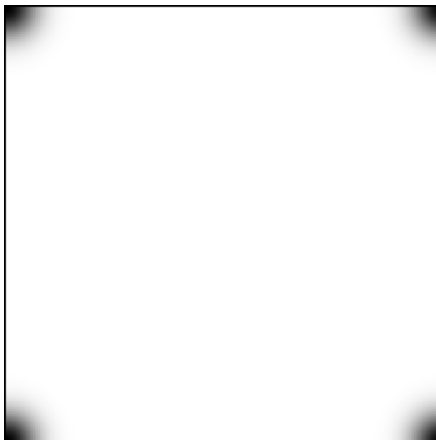
```
ZPLena(1:256,1:256) = Lena;
```

*% Fill in the centered DFT array for the filter from design eqn:*

```
[COLS,ROWS] = meshgrid(0:255,0:255);
```

```
sigma = 12;
```

```
H = 1 - exp(-((ROWS-128).^ 2 + (COLS-128).^ 2)/(2*sigma^2));
```



*% call fftshift to un-center the filter DFT array:*

```
Hshift = fftshift(H);
```

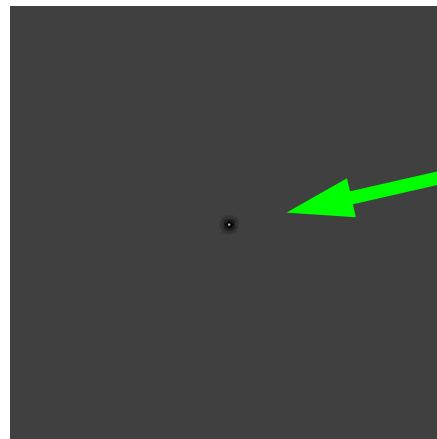
- Use frequency sampling...

```
% ifft to get zero-phase impulse resp:
hzero = ifft2(Hshift);
disp = stretch(log(1 + stretch(hzero)));
```

```
% shift the zero-phase impulse response
% to proceed as in Example 4:
h = fftshift(hzero);
```

```
% compute filtered result as in Example 4:
ZPh = zeros(512,512);
ZPh(1:256,1:256) = h;
```

```
y = real(ifft2(fft2(ZPLena) .* fft2(ZPh)));
yy = stretch(y(129:384,129:384));
```

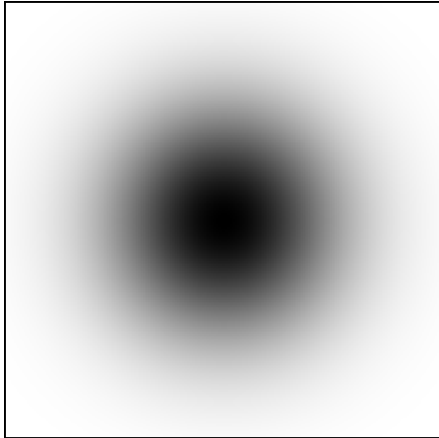


The interesting part is here; may be small and hard to see on your monitor!

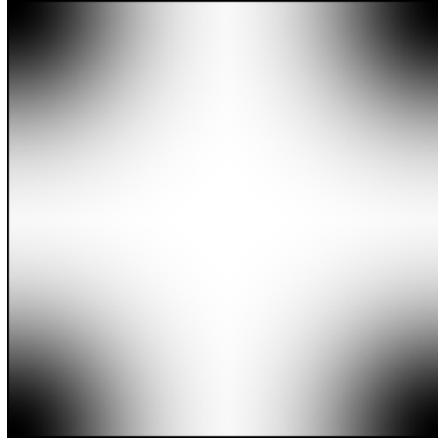


- Run again with  $\sigma = 48$ :

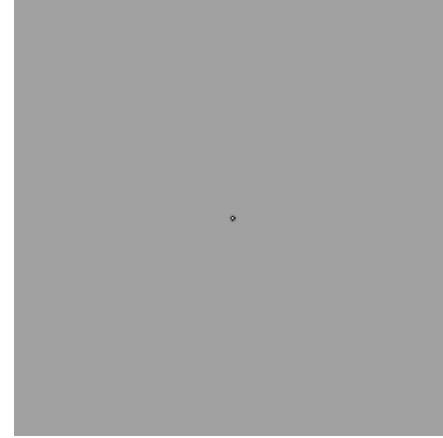
centered freq resp



un-centered freq resp



impulse resp



output

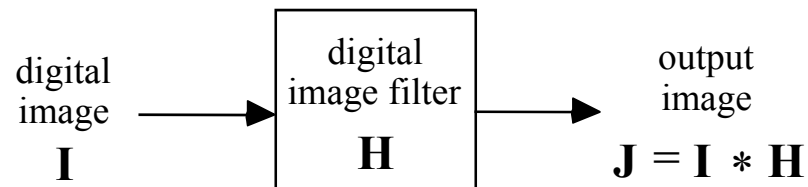
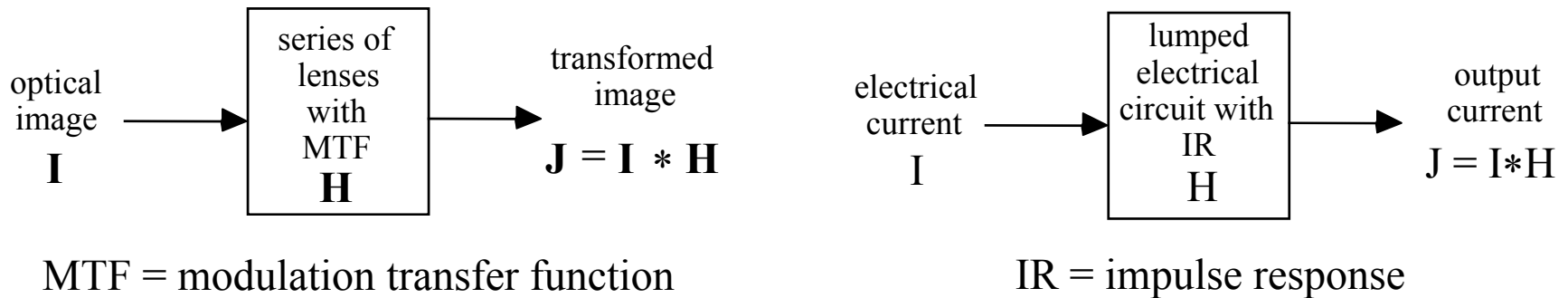


original



# LINEAR IMAGE FILTERING

- A process that transforms a signal or image **I** by linear convolution is a type of **linear system**.



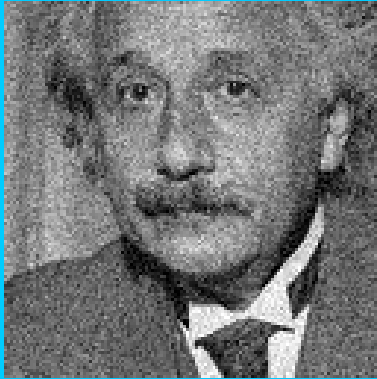
Of interest to us

# Goals of Linear Image Filtering

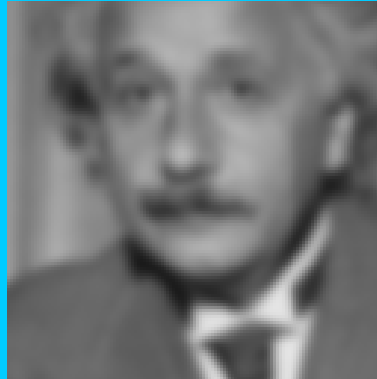
- Process sampled, quantized images to **transform** them into
  - images of **better quality** (by some criteria)
  - images with certain features **enhanced**
  - images with certain features **de-emphasized** or **eradicated**

# Some Specific Goals

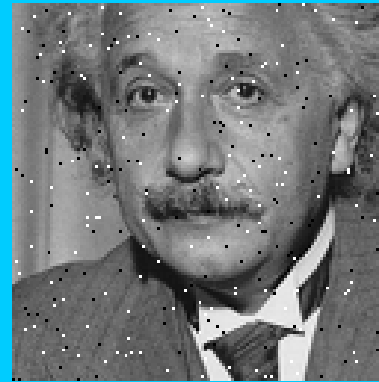
- smoothing - remove noise from bit errors, transmission, etc
- deblurring - increase **sharpness** of blurred images
- sharpening - emphasize significant features, such as **edges**
- combinations of these



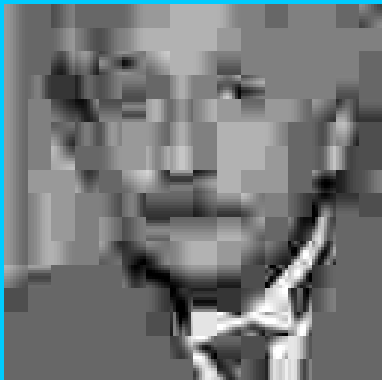
Gaussian white noise



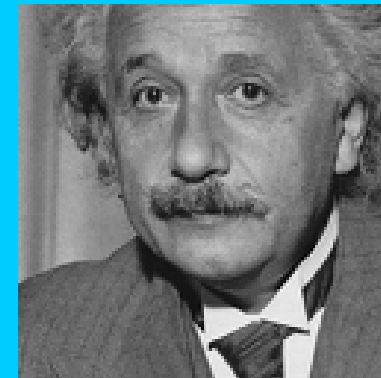
blur



impulse noise

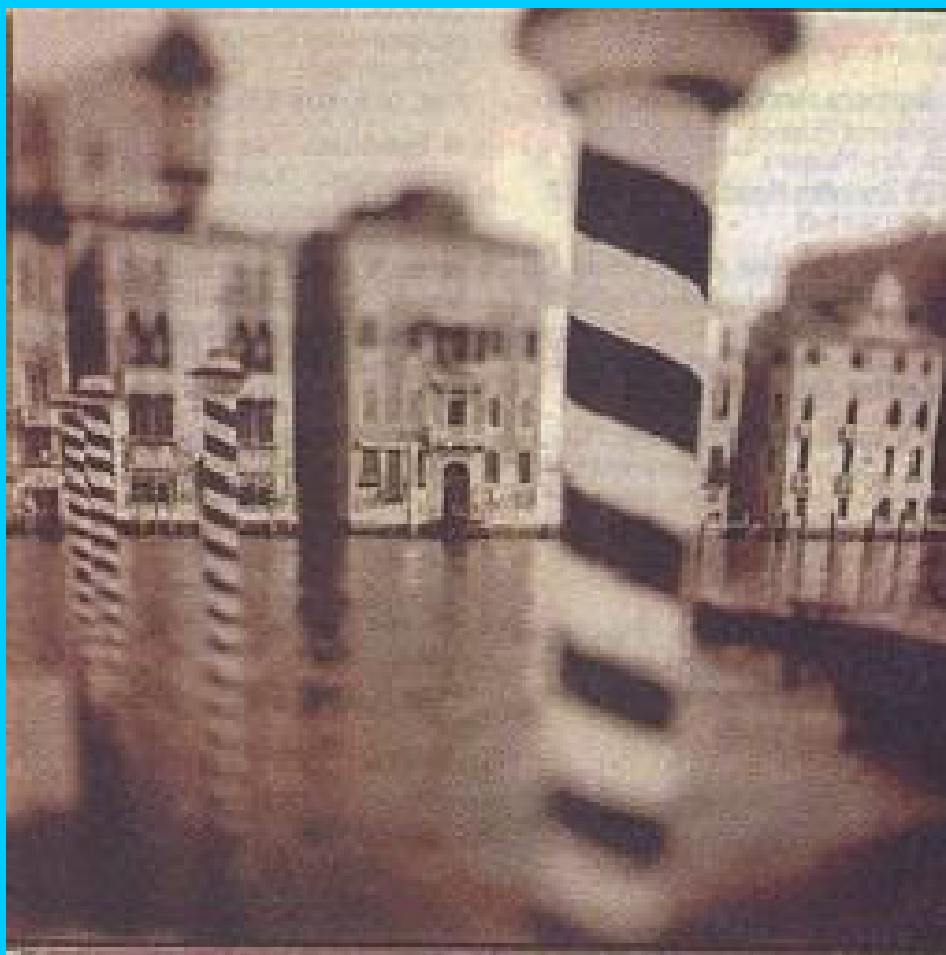


JPEG compression



Albert

## Variety of Image Distortions



A Tough One!

# Characterizing Linear Filters

- Any **linear** digital image filter can be characterized in one of two **equivalent** ways:

(1) The filter **impulse response**  $\mathbf{H} = [H(m, n)]$

(2) The filter **frequency response**  $\tilde{\mathbf{H}} = [\tilde{H}(u, v)]$

- These are a DFT pair:

$$\tilde{\mathbf{H}} = \text{DFT}[\mathbf{H}]$$

$$\mathbf{H} = \text{IDFT}[\tilde{\mathbf{H}}]$$

# Frequency Response

- The **frequency response** describes how the system affects each frequency in an image that is passed through the system.

- Since

$$\tilde{H}(u, v) = |\tilde{H}(u, v)| \exp\{j\angle\tilde{H}(u, v)\}$$

an image frequency component at  $(u, v) = (a, b)$  is amplified or attenuated by the amount  $|\tilde{H}(a, b)|$  and shifted by the amount  $\angle\tilde{H}(a, b)$

# Frequency Response Example

- Let the **input** to a system **H** be a cosine image:

$$I(m, n) = \cos \left[ 2\pi \left( \frac{b}{M} m + \frac{c}{N} n \right) \right] = \frac{1}{2} \left( W_M^{bm} W_N^{cn} + W_M^{-bm} W_N^{-cn} \right)$$

- If **H** is real, the **output** is

$$J(m, n) = H(m, n) * I(m, n) = \frac{1}{2} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} H(p, q) \left[ W_M^{-b(m-p)} W_N^{-c(n-q)} + W_M^{b(m-p)} W_N^{c(n-q)} \right]$$

$$= \frac{1}{2} W_M^{-bm} W_N^{-cn} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} H(p, q) W_M^{bp} W_N^{cq} + \frac{1}{2} W_M^{bm} W_N^{cn} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} H(p, q) W_M^{-bp} W_N^{-cq}$$

$$= \frac{1}{2} W_M^{-bm} W_N^{-cn} \tilde{H}(b, c) + W_M^{bm} W_N^{cn} \tilde{H}(-b, -c) = \frac{|\tilde{H}(b, c)| \cos \left[ 2\pi \left( \frac{b}{M} m + \frac{c}{N} n \right) + \angle \tilde{H}(b, c) \right]}{}$$

# Impulse Response

- The response of system **H** to the **unit impulse**

$$\delta(m,n) = \begin{cases} 1; & m = n = 0 \\ 0; & \text{else} \end{cases}$$

- An effective way to model responses since every input image is a **weighted sum of unit pulses**

$$I(m,n) = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} I(p,q) \delta(m-p, n-q) = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} I(p,q) \delta(m-p, n-q)$$

# Linear Filter Design

- Often a filter is to be designed according to **frequency-domain** specifications.
- **Models of linear distortion** in the **continuous domain** lead to **linear digital solutions**.

# Sampled Analog Specification

- Given an **analog** or **continuous-space** spec:

$$H_C(x, y) \stackrel{\mathcal{F}}{\Leftrightarrow} \tilde{H}_C(\Omega, \Lambda)$$

- Impulse Invariance ( $X=Y=1$ ):

$$H(m, n) = H_C(m, n) ; \quad \begin{array}{l} -\frac{M}{2} \leq m \leq \frac{M}{2} - 1 \\ -\frac{N}{2} \leq n \leq \frac{N}{2} - 1 \end{array} \quad (1)$$

- DFT frequency sampling:

$$\tilde{H}(u, v) = \tilde{H}_C\left(\frac{u}{M}, \frac{v}{N}\right) \quad (2)$$

# Low-Pass, Band-Pass, and High-Pass Filters

- The terms low-pass, band-pass, and high-pass are **qualitative descriptions** of a system's frequency response.
- "**Low-pass**" - attenuates all but the "lower" frequencies.
- "**Band-pass**" - attenuates all but an intermediate range of "middle" frequencies.
- "**High-pass**" - attenuates all but the "higher" frequencies.
- We have seen examples of these: the **zero-one** frequency masking results.

# Generic Uses of Filter Types

- **Low-pass filters** are typically used to
  - **smooth** noise
  - **blur** image details to emphasize gross features
- **High-pass filters** are typically used to
  - **enhance** image details and contrast
  - **remove** image blur
- **Bandpass filters** are usually **special-purpose**

# Example Low-Pass Filter

- The **Gaussian filter** with frequency response

$$\tilde{H}_C(\Omega, \Lambda) = \exp\left[-2(\pi\sigma)^2(\Omega^2 + \Lambda^2)\right]$$

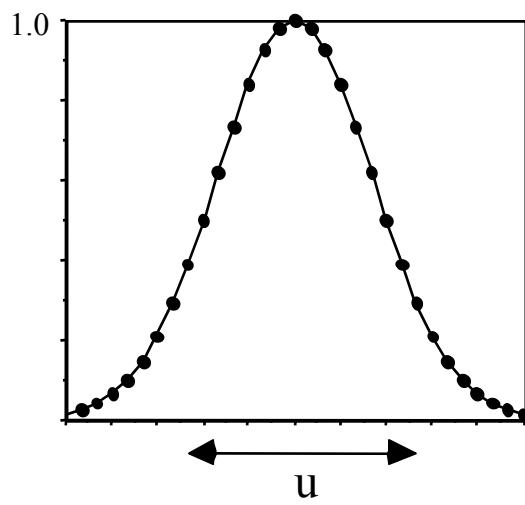
hence

$$\tilde{H}(u, v) = \exp\left\{-2\pi^2\sigma^2\left[\left(\frac{u}{N}\right)^2 + \left(\frac{v}{M}\right)^2\right]\right\}$$

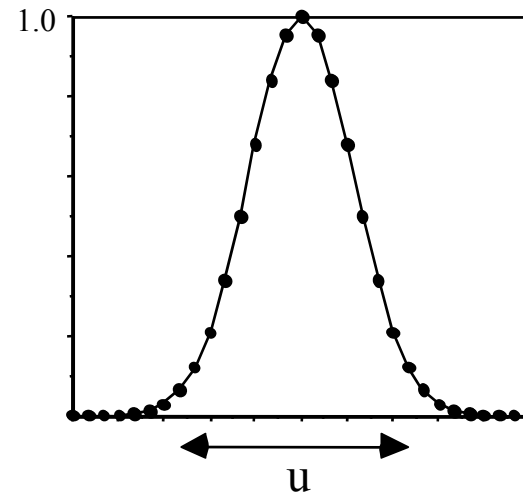
which quickly falls at larger frequencies.

- The Gaussian is an **important** low-pass filter.

# Gaussian Filter Profile



$N = 32, \sigma = 1$



$N = 32, \sigma = 1.5$

Plots show one row of DFT array ( $v = 0$ )

# Example Band-Pass Filter

- Can define a BP filter as the **difference of two LPFs** identical except for a scaling factor.
- A common choice in image processing is the **difference-of-gaussians (DOG)** filter:

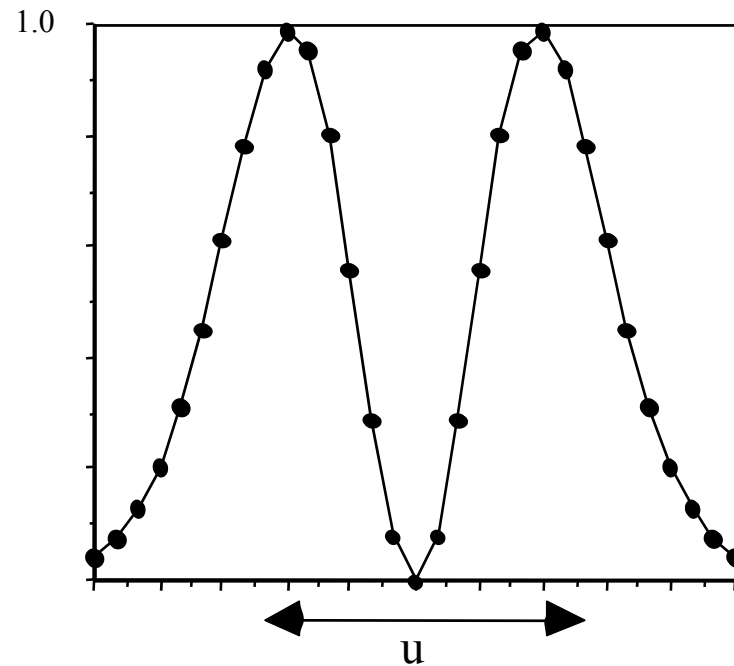
$$\tilde{H}_C(\Omega, \Lambda) = \exp\left[-2(\pi\sigma)^2(\Omega^2 + \Lambda^2)\right] - \exp\left[-2(K\pi\sigma)^2(\Omega^2 + \Lambda^2)\right]$$

hence

$$\tilde{H}(u, v) = \exp\left\{-2(\pi\sigma)^2\left[\left(\frac{u}{N}\right)^2 + \left(\frac{v}{M}\right)^2\right]\right\} - \exp\left\{-2(K\pi\sigma)^2\left[\left(\frac{u}{N}\right)^2 + \left(\frac{v}{M}\right)^2\right]\right\}$$

- **Typically,  $K \approx 1.5$ .**

# DOG Filter Profile



N=32

- DOG filters are very useful for image analysis – and in human visual modelling.

# Example High-Pass Filter

- The **Laplacian** filter is also important

$$\tilde{H}_C(\Omega, \Lambda) = A(\Omega^2 + \Lambda^2)$$

hence

$$\tilde{H}(u, v) = A \left[ \left( \frac{u}{N} \right)^2 + \left( \frac{v}{M} \right)^2 \right]$$

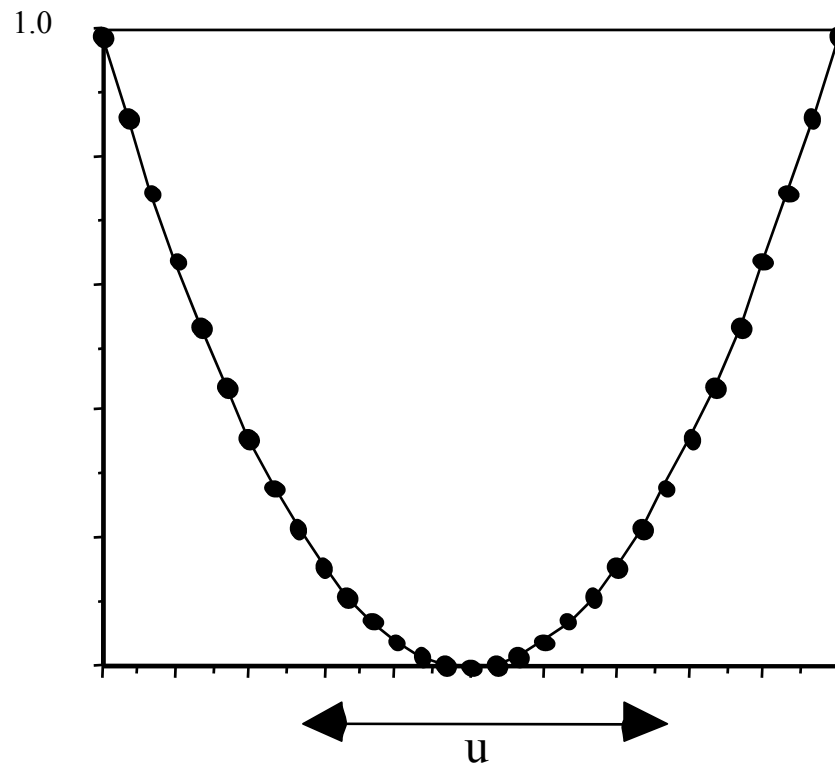
although this is a **severely truncated**

approximation! Best used **in combination** with another filter (later).

- An approximation to the Fourier transform of the **continuous Laplacian**:

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

# Laplacian Profile



$A = 4.5, N = 32$

# LINEAR IMAGE DENOISING

- **Linear image denoising** means a process that smooths noise **without** destroying the image information.
- The noise is usually modeled as **additive** or **multiplicative**.
- We consider **additive** noise now.
- **Multiplicative** noise is better handled by **homomorphic filtering** which uses a **nonlinear** preprocessing step.

# Additive White Noise Model

- Model **additive white noise** as an image  $\mathbf{N}$  with highly chaotic, unpredictable elements.
- Can be **thermal circuit noise, channel noise, sensor noise**, etc.
- Noise may effect the **continuous image** before sampling:

$$\underbrace{J_C(x,y)}_{\text{observed}} = \underbrace{I_C(x,y)}_{\text{original}} + \underbrace{N_C(x,y)}_{\text{white noise}}$$

# Zero-Mean White Noise

- The white noise is **zero-mean** if the limit of the average of  $P$  arbitrary noise image realizations vanishes as  $P \rightarrow \infty$ :

$$\frac{1}{P} \sum_{p=1}^P N_{C,p}(x, y) \rightarrow 0 \text{ for all } (x, y) \text{ as } P \rightarrow \infty$$

- **On average**, the noise falls around the value zero.\*

\*Strictly speaking, the noise is also "mean-ergodic."

# Spectrum of White Noise

- The noise energy spectrum is

$$\tilde{N}_C(\Omega, \Lambda) = \mathcal{F} \{N_C(\mathbf{x}, y)\}$$

- If the noise is white, then, on average, the energy spectrum will be flat (flat spectrum = 'white'):

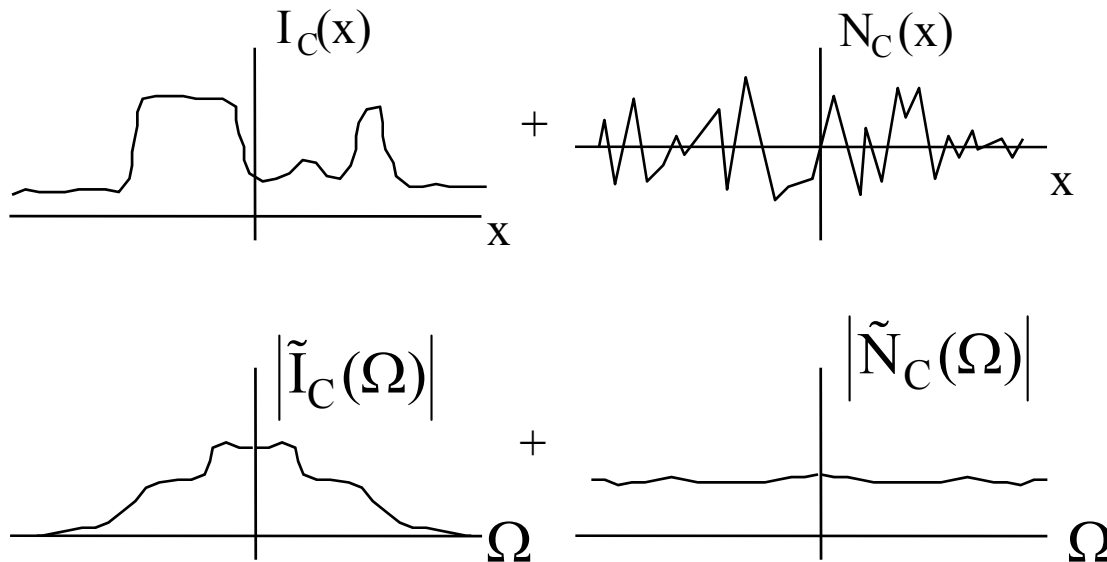
$$\frac{1}{P} \sum_{p=1}^P |\tilde{N}_{C,p}(\Omega, \Lambda)| \rightarrow \eta \text{ for all } (\Omega, \Lambda) \text{ as } P \rightarrow \infty$$

- **Note**:  $\eta^2$  is called the noise power.

# White Noise Model

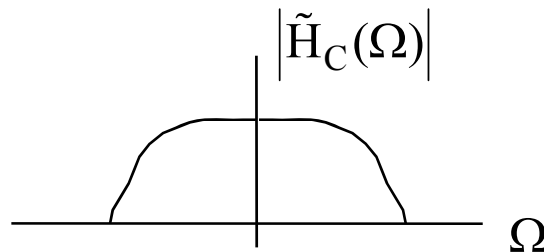
- White noise is an **approximate** model of additive **broadband** noise:

$$\underbrace{J_C(x,y)}_{\text{observed}} = \underbrace{I_C(x,y)}_{\text{original}} + \underbrace{N_C(x,y)}_{\text{broadband noise}}$$



# Linear Denoising

- **Objective**: Remove as much of the high-frequency noise as possible while **preserving** as much of the image spectrum as possible.
- Generally accomplished by a LPF of fairly wide bandwidth (images are fairly wideband):



# Digital White Noise

- We make a similar model for **digital zero-mean additive white noise**:

$$\underbrace{\mathbf{J}}_{\text{observed}} = \underbrace{\mathbf{I}}_{\text{original}} + \underbrace{\mathbf{N}}_{\text{noise}}$$

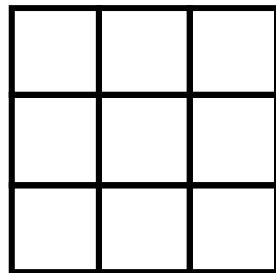
- **On average**, the elements of  $\mathbf{N}$  will be zero.
- The DFT of the noisy image is the sum of the DFTs of the original image and the noise image:

$$\underbrace{\tilde{\mathbf{J}}}_{\text{observed}} = \underbrace{\tilde{\mathbf{I}}}_{\text{original}} + \underbrace{\tilde{\mathbf{N}}}_{\text{noise}}$$

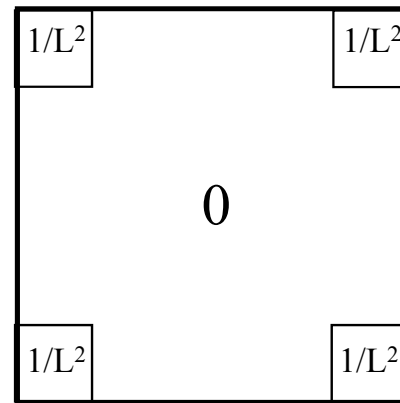
- **On average** the noise DFT will contain a **broad band** of frequencies.

# Denoising - Average Filter

- To smooth an image: replace each pixel in a noisy image by the **average** of its  $L \times L$  neighbors:



3x3 window



zero phase impulse response

# Average Filter Rationale

- Averaging elements **reduces the noise mean towards zero.**
- Also reduces the noise variance and the noise power  $\eta^2$  (actually, they are equal).
- The **window size** is usually an intermediate value to balance the tradeoff between noise smoothing and image smoothing.
- **Typical** average filter window sizes:  $L \times L = 3 \times 3$ ,  $5 \times 5$ , ...,  $15 \times 15$  (lots of smoothing), e.g. for a  $512 \times 512$  image.

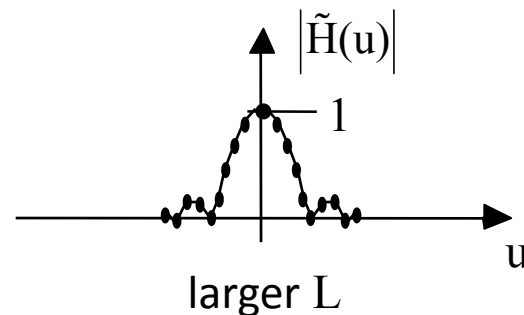
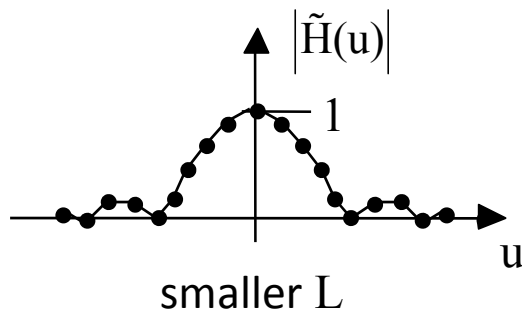
# Average Filter Rationale

- **Linear filtering the image** (with zero-padding assumed hereafter)

$$\mathbf{K} = \mathbf{H} * \mathbf{J} = \mathbf{H} * \mathbf{I} + \mathbf{H} * \mathbf{N}$$

$$\tilde{\mathbf{K}} = \tilde{\mathbf{H}} \otimes \tilde{\mathbf{J}} = \tilde{\mathbf{H}} \otimes \tilde{\mathbf{I}} + \tilde{\mathbf{H}} \otimes \tilde{\mathbf{N}}$$

will affect image / noise spectra in the same way:



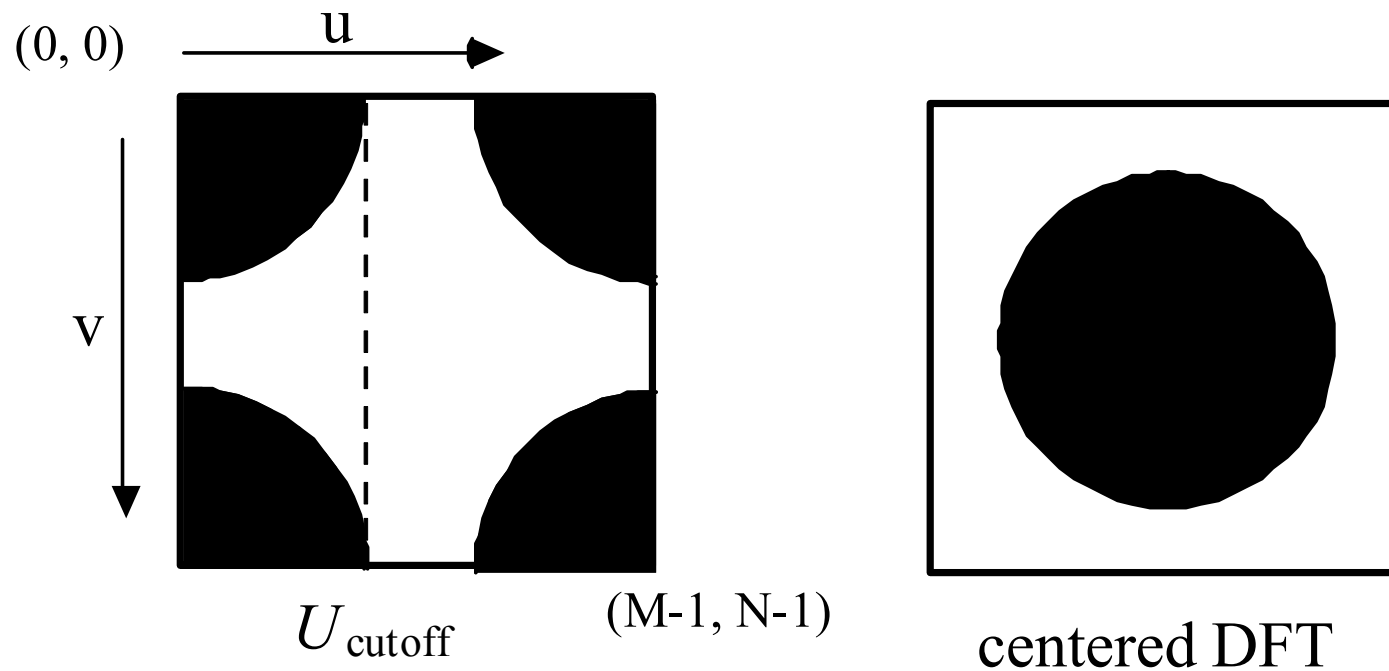
# Denoising – Ideal Low-Pass Filter

- Also possible to use an **ideal low-pass filter** by designing in the DFT domain:

$$\tilde{H}(u, v) = \begin{cases} 1 ; \text{ if } \sqrt{u^2 + v^2} \leq U_{\text{cutoff}} \\ 0 ; \text{ otherwise} \end{cases}$$

- Possibly useful if its possible to estimate the highest important **radial** frequency  $U_{\text{cutoff}}$  in the original image.

# Ideal LPF



# Denoising - Gaussian Filter

- The **isotropic** Gaussian filter is an **effective** smoother:

$$\tilde{H}(u, v) = \exp\left[-2\pi^2\sigma^2\left(\frac{u^2 + v^2}{N^2}\right)\right]$$

- It gives more weight to “**closer**” neighbors.
- DFT design: Set the **half-peak bandwidth** to  $U_{\text{cutoff}}$  by solving for  $\sigma$ :

$$\exp\left[-2\pi^2\sigma^2\left(\frac{U_{\text{cutoff}}^2}{N^2}\right)\right] = \frac{1}{2}$$

$$\Rightarrow \sigma = \left(\frac{N}{\pi U_{\text{cutoff}}}\right) \sqrt{\log \sqrt{2}} \approx 0.19 \left(\frac{N}{U_{\text{cutoff}}}\right)$$

# Measuring the Success of the Filter

- The observed image:  $\underbrace{\mathbf{J}}_{\text{observed}} = \underbrace{\mathbf{I}}_{\text{original}} + \underbrace{\mathbf{N}}_{\text{noise}}$
- The filtered (denoised) image:  $\mathbf{K} = \mathbf{H} * \mathbf{J}$
- There are two main **figures of merit** that have traditionally been used to measure or quantify the success of the denoising algorithm:
  - Mean squared error (MSE):

$$MSE(K) = \frac{\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} [K(m,n) - I(m,n)]^2}{MN}$$

Better Performance  
= **lower** MSE

# Measuring Denoising Success

- Improvement in signal-to-noise ratio (ISNR):

$$ISNR(\mathbf{K}) = 10 \log_{10} \frac{MSE(\mathbf{J})}{MSE(\mathbf{K})} \text{ dB}$$

Better Performance = <b>higher</b> ISNR
--

- Both MSE and ISNR are **widely** used to quantify denoising performance.
- Most of our existing theory of optimal signal processing is based on them.
- But they do have limitations.

# Limitations of MSE and ISNR

- Both are based on MSE, which does not always agree well with **human visual perception**.
  - In other words, the result that has better MSE (or better ISNR) may sometimes look **worse**.
- Example:
  - Let  $\mathbf{J}$  be a *Lena* image with terrible noise.
  - Suppose that some fantastic denoising algorithm produces an output image  $\mathbf{K}$  that is **exactly** the original *Lena* image, but shifted right by one column.
  - Even though the result looks **fantastic**, the MSE and ISNR will both be **terrible**.
- This has motivated recent work to develop new **perceptually based** figures of merit for images like the **structural similarity index (SSIM)**.

# LINEAR IMAGE DEBLURRING

- Often an image that is obtained digitally has already been corrupted by a linear process.
  - in other words, the optical image was corrupted **prior** to being captured by the camera.
- This may be due to **motion blur**, blurring due to **defocusing**, etc.
- We can model such an observed image as the result of a **linear convolution**:

so

$$\underbrace{J_C(x, y)}_{\text{observed}} = \underbrace{G_C(x, y)}_{\text{linear distortion}} * \underbrace{I_C(x, y)}_{\text{original}}$$

$$\underbrace{\tilde{J}_C(\Omega, \Lambda)}_{\text{observed}} = \underbrace{\tilde{G}_C(\Omega, \Lambda)}_{\text{linear distortion}} \cdot \underbrace{\tilde{I}_C(\Omega, \Lambda)}_{\text{original}}$$

# Digital Blur Function

- The **sampled image** will then be of the form (assuming sufficient sampling rate)

hence 
$$\mathbf{J} = \mathbf{G} * \mathbf{I}$$

$$\tilde{\mathbf{J}} = \tilde{\mathbf{G}} \otimes \tilde{\mathbf{I}}$$

- The distortion  $\mathbf{G}$  is **almost always low-pass** (blurring).
- Our goal is to use digital filtering to reduce blur – a VERY hard problem!

# Deblur - Inverse Filter

- Often it is possible to make an **estimate** of the distortion  $G$ .
- This may be possible by examining the **physics** of the situation.
- For example, **motion blur** (relative camera movement) is usually along **one direction**. If this can be determined, then a filter can be designed.
- The MTF of a camera can often be determined – and hence, a **digital deblur** filter designed.

# Deconvolution

- Reversing the linear blur  $\mathbf{G}$  is **deconvolution**. It is done using the **inverse filter** of the distortion:

$$\tilde{\mathbf{G}}_{\text{inverse}}(u, v) = \frac{1}{\tilde{\mathbf{G}}(u, v)}$$

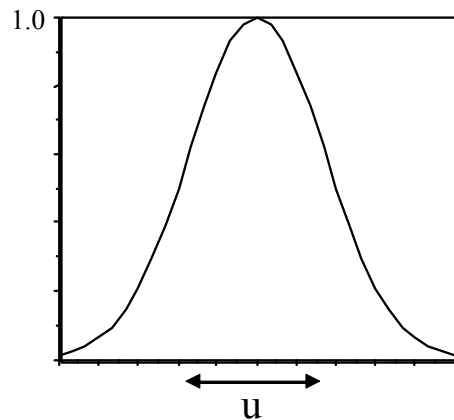
provided that  $\tilde{\mathbf{G}}(u, v) \neq 0$  for any  $(u, v)$ .

- Then the restored image is:

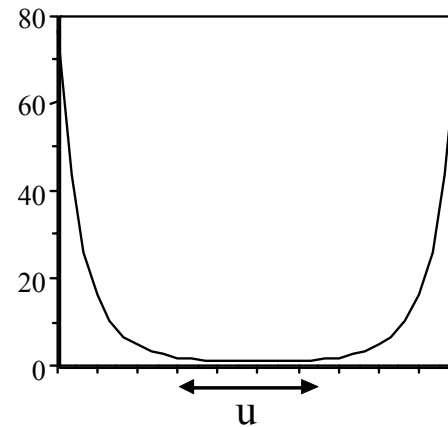
$$\tilde{\mathbf{K}} = \tilde{\mathbf{G}}_{\text{inverse}} \otimes \mathbf{G} \otimes \tilde{\mathbf{I}} = \tilde{\mathbf{I}} \quad !$$

# Blur Estimation

- An **estimate** of blur **G** might be obtainable.
- The inverse of a **low-pass** blur is **high-pass**:



Gaussian distortion



Inverse filter

- At high frequencies the designer must be **careful!**
- **Note:** The inverse takes value 1.0 at  $(u, v) = (0,0)$

# Deblur - Missing Frequencies

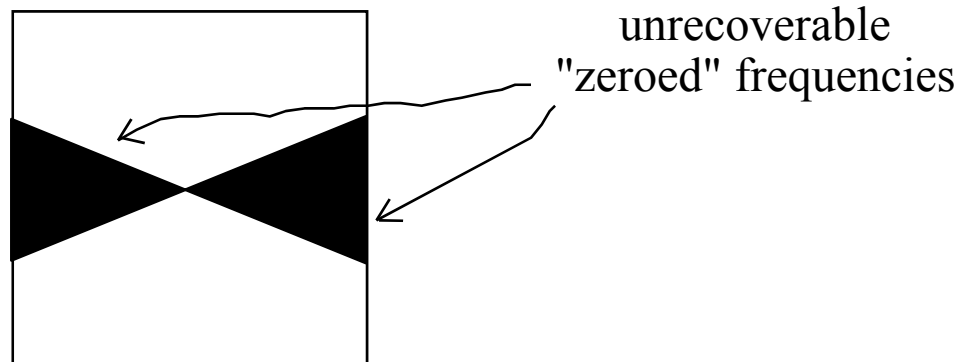
- Unfortunately, things are not always so "ideal" in the real world.
- Sometimes the blur frequency response takes **zero value(s)**.
- If

$\tilde{G}(u_0, v_0) = 0$  for some  $(u_0, v_0)$ , then  $\tilde{G}_{\text{inverse}}(u_0, v_0) = \infty$

which is **meaningless**.

# Zeroed Frequencies

- The reality: any frequencies that are **zeroed** by a linear distortion are **unrecoverable in practice** (at least by linear means) - lost forever!
- The best that can be done is to reverse the distortion at the **non-zero values**.
- Sometimes much of the frequency plane is lost. Some optical systems remove a **large** angular spread of frequencies:



# Pseudo-Inverse Filter

- The **pseudo-inverse filter** is defined by

$$\tilde{G}_{\text{p-inverse}}(u, v) = \begin{cases} 1/\tilde{G}(u, v) ; & \text{if } \tilde{G}(u, v) \neq 0 \\ 0 & ; \text{if } \tilde{G}(u, v) = 0 \end{cases}$$

- Thus **no attempt** is made to recover lost frequencies.
- The pseudo-inverse is set to **zero** in the known region of missing frequencies – a conservative approach.
- In this way spurious (noise) frequencies will be **eradicated**.

# Deblur in the Presence of Noise

- A **worse case** is when the image **I** is distorted both by linear blur **G** and additive noise **N**:

$$\mathbf{J} = \mathbf{G} * \mathbf{I} + \mathbf{N}$$

- This may occur, e.g., if an image is linearly distorted then sent over a noisy channel.
- The DFT:

$$\tilde{\mathbf{J}} = \tilde{\mathbf{G}} \otimes \tilde{\mathbf{I}} + \tilde{\mathbf{N}}$$

# Filtering a Blurred, Noisy Image

- Filtering with a linear filter  $\mathbf{H}$  will produce the result

$$\mathbf{K} = \mathbf{H} * \mathbf{J} = \mathbf{H} * \mathbf{G} * \mathbf{I} + \mathbf{H} * \mathbf{N}$$

or

$$\tilde{\mathbf{K}} = \tilde{\mathbf{H}} \otimes \tilde{\mathbf{J}} = \tilde{\mathbf{H}} \otimes \tilde{\mathbf{G}} \otimes \tilde{\mathbf{I}} + \tilde{\mathbf{H}} \otimes \tilde{\mathbf{N}}$$

- The problem is that **neither** a low-pass filter (to smooth noise, but won't correct the blur) nor a high-pass filter (the inverse filter, which will **amplify** the noise) will work.

# Failure of Inverse Filter

- If the inverse filter were used, then

$$\mathbf{K} = \mathbf{G}_{\text{inverse}} * \mathbf{J} = \mathbf{I} + \mathbf{G}_{\text{inverse}} * \mathbf{N}$$

or

$$\tilde{\mathbf{K}} = \tilde{\mathbf{G}}_{\text{inverse}} \otimes \tilde{\mathbf{J}} = \tilde{\mathbf{I}} + \tilde{\mathbf{G}}_{\text{inverse}} \otimes \tilde{\mathbf{N}}$$

- In this case the blur is corrected, **but** the restored image has **horribly amplified high-frequency noise** added to it.

# Wiener Filter

- The **Wiener filter** (after Norbert Wiener) or minimum-mean-square-error (MMSE) filter is a “best” linear approach.
- The Wiener filter for **blur G** and **white noise N** is

$$\tilde{G}_{\text{Wiener}}(u, v) = \frac{\tilde{G}^*(u, v)}{|\tilde{G}(u, v)|^2 + \eta^2}$$

- Often the noise power  $\eta$  is **unknown** or **unobtainable**. The designer will usually experiment with heuristic values for  $\eta$ .
- In fact, better **visual results** may often be obtained by using such heuristic values for  $\eta$  in the Wiener filter.

# Wiener Filter Rationale

- We won't derive the Wiener filter here. But:
- If  $\eta = 0$  (**no noise**), the Wiener filter reduces to the **inverse filter**:

$$\tilde{G}_{\text{Wiener}}(u, v) = \frac{\tilde{G}^*(u, v)}{|\tilde{G}(u, v)|^2} = \frac{1}{\tilde{G}(u, v)}$$

which is **highly desirable**.

# Wiener Filter Rationale

- If  $\tilde{G}(u, v) = 1$  for all  $(u, v)$  (**no blur**) the Wiener filter reduces to:

$$\tilde{G}_{\text{Wiener}}(u, v) = \frac{1}{1 + \eta^2}$$

which does **nothing except scale the variance** so that the MSE is minimized.

- So, the Wiener filter is not useful unless there is **blur**.

# Pseudo-Wiener Filter

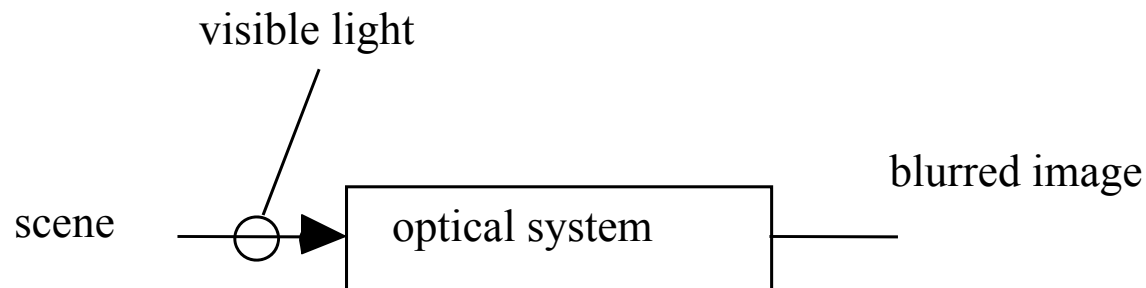
- Obviously, if there are **frequencies** zeroed by the linear distortion  $\mathbf{G}$  then it is best to define a **pseudo-Wiener filter**:

$$\tilde{G}_{\text{Wiener}}(u, v) = \begin{cases} \frac{\tilde{G}^*(u, v)}{|\tilde{G}(u, v)|^2 + \eta^2} & ; \text{if } \tilde{G}(u, v) \neq 0 \\ 0 & ; \text{if } \tilde{G}(u, v) = 0 \end{cases}$$

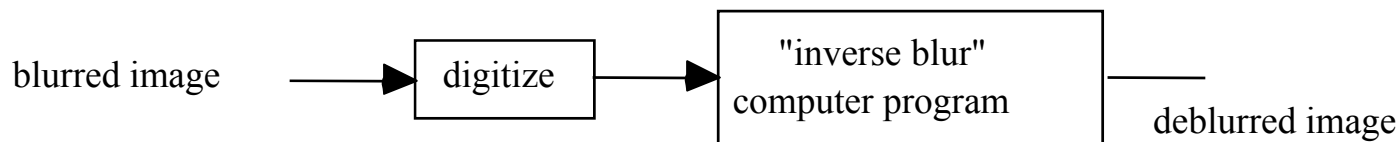
- Noise in the "missing region" of frequencies will be **eradicated**.

# APPLICATION EXAMPLE: OPTICAL SERIAL SECTIONING

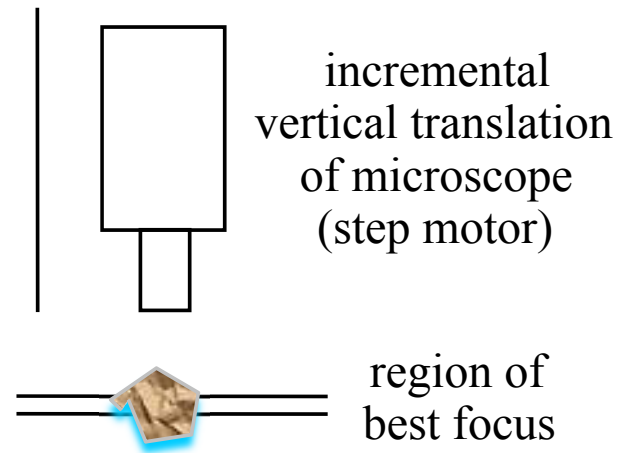
- **Optical** systems often **blur** images:



one possible solution:

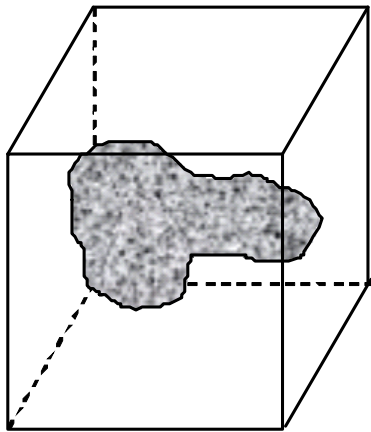


# Optical Sectioning Microscopy

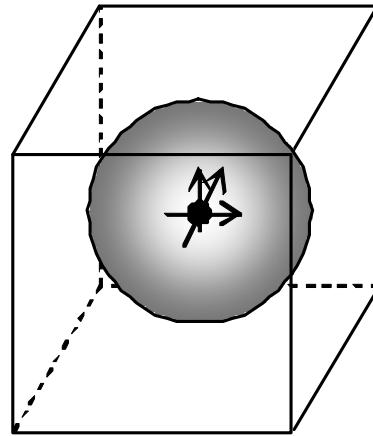


- A very **narrow-depth of field** microscope. One image taken at **each** focusing plane, giving a **sequence** of 2-D images - or **3-D image of optical density**.

# 3-D Image of Optical Density



3-D image of optical density

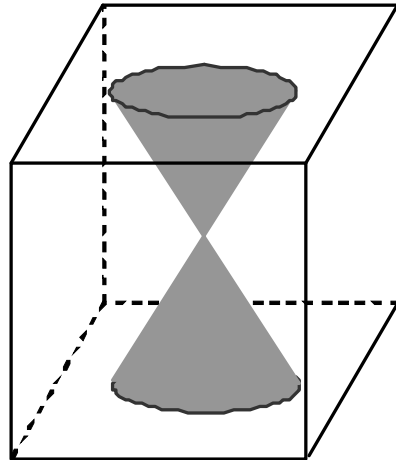


Magnitude of 3-D DFT

# 3-D Optical System Analysis

- In this system **three** effects occur:
  - (1) A **linear low-pass distortion G**.
  - (2) A large **biconic region of frequencies** aligned along the **optical axis is zeroed**.
  - (3) Approximately **additive white noise**.
- Items (1) and (2) shown using **principles of geometric optics**. Item (3) shown **empirically**.

# 3-D Biconic Spread of Lost Frequencies



Region of zeroed 3-D frequencies

- Note that DC  $(u, v, w) = (0, 0, 0)$  is zeroed also. Hence the **background level (AOD) is lost**.
- There is **no linear filtering way** to recover this biconic region of frequencies.

# 3-D Restoration

- So: the 3-D images are **blurred**, have a large 3-D region of **missing frequencies**, and are corrupted by low-level **white noise** added.
- The **processed results** show the efficacy of
  - pseudo-inverse filtering
  - pseudo-Wiener filteringapplied to two optical sectioned 3-D images:
  - a pollen grain
  - a pancreas Islet of Langerhans (collection of cells)

# Shroud of Turin Image



An intensely enhanced, denoised, deblurred, etc etc etc and **debated** image

# Comments

- Next we shall dispense with **frequency-domain concepts** and **linear filtering tools**.
- We shall now study **non-linear filtering methods...** onward to **Module 6**.