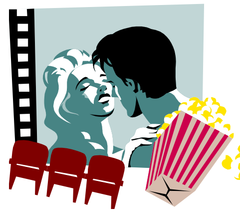
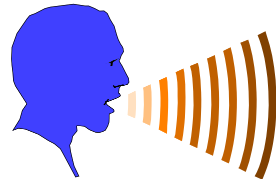





Image and Video Processing

Fourier Transform and Linear Filtering
Part 2: 2D Convolution

Yao Wang
Tandon School of Engineering, New York University

Outline of this lecture

- Part 1: 2D Fourier Transforms
- Part 2: 2D Convolution
- Part 3: Basic image processing operations: Noise removal, image sharpening, and edge detection using filtering

2D Convolution

- Continuous and discrete space convolution
  - Review of 1D convolution (continuous and discrete time)
  - 2D convolution (continuous and discrete space)
  - Separable filters
- Convolution theorem
- Frequency response of filters

Linear Convolution over Continuous Space

- 1D convolution

$$f(x)*h(x)=\int_{-\infty}^{\infty} f(x-\alpha)h(\alpha)d\alpha=\int_{-\infty}^{\infty} f(\alpha)h(x-\alpha)d\alpha$$

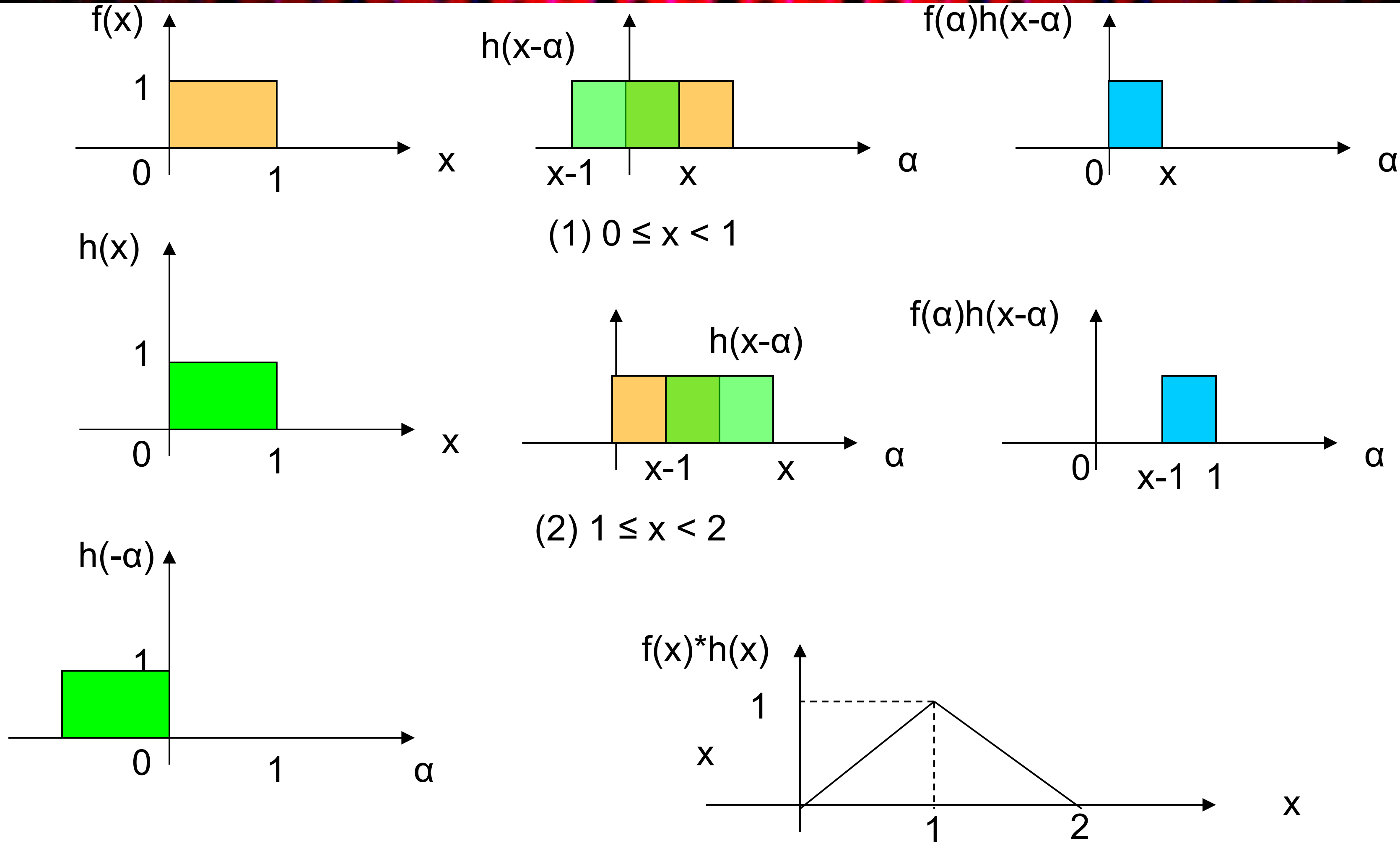
$$f(x)*\delta(x)=f(x),\quad f(x)*\delta(x-x_0)=f(x-x_0)$$

- 2D convolution

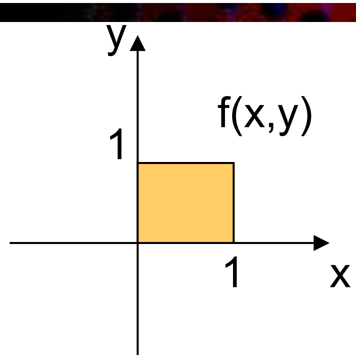
$$f(x,y)*h(x,y)=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x-\alpha,y-\beta)h(\alpha,\beta)d\alpha d\beta$$

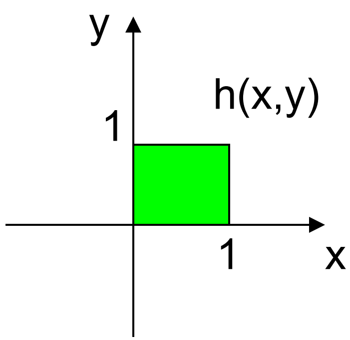
$$=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)h(x-\alpha,y-\beta)d\alpha d\beta$$

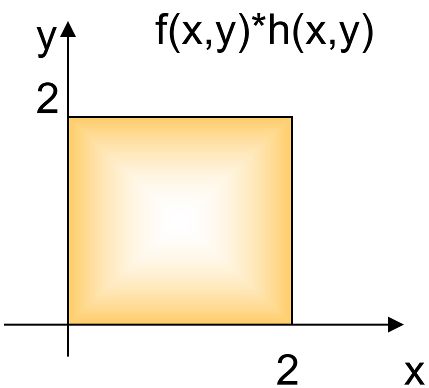
Examples of 1D Convolution

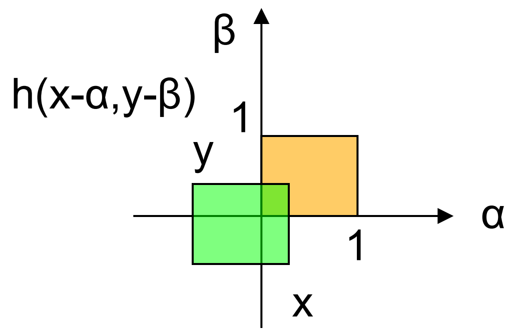


Example of 2D Convolution

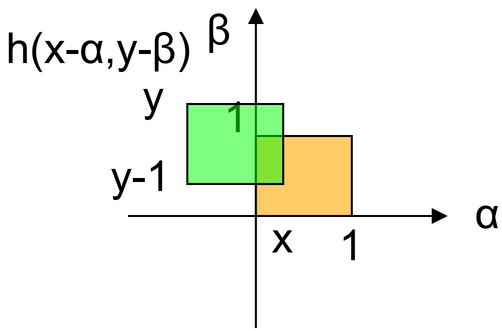




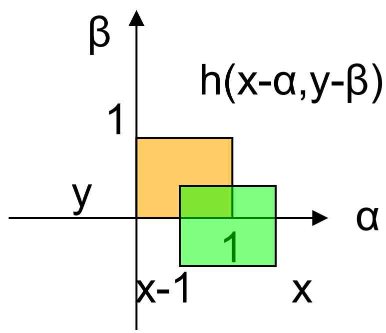




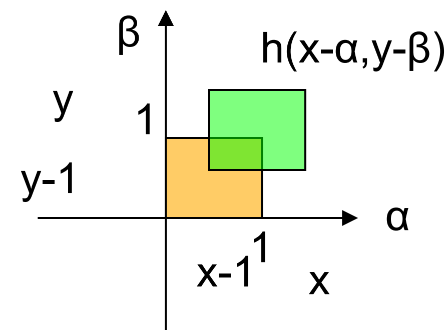
(1) $0<x\le 1,\ 0<y\le 1$
$g(x,y)=x*y$



(2) $0<x\le 1,\ 1<y\le 2$
$g(x,y)=x*(2-y)$



(3) $1<x\le 2,\ 0<y\le 1$
$g(x,y)=(2-x)*y$



(4) $1<x\le 2,\ 1<y\le 2$
$g(x,y)=(2-x)*(2-y)$

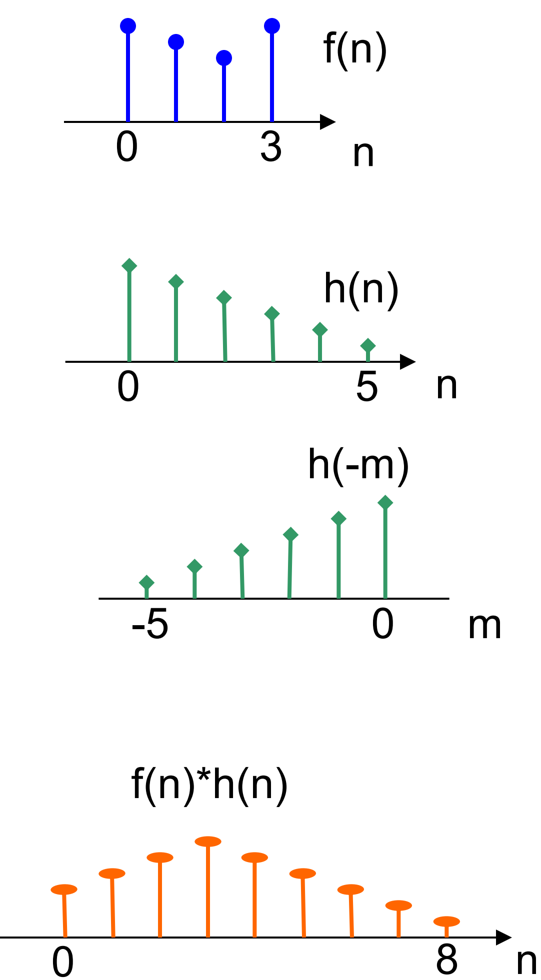
Convolution of 1D Discrete Signals

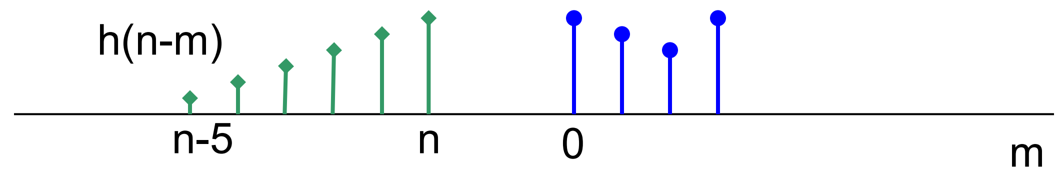
- Definition of convolution

$$f(n) * h(n) = \sum_{m=-\infty}^{\infty} f(n-m)h(m) = \sum_{m=-\infty}^{\infty} f(m)h(n-m)$$

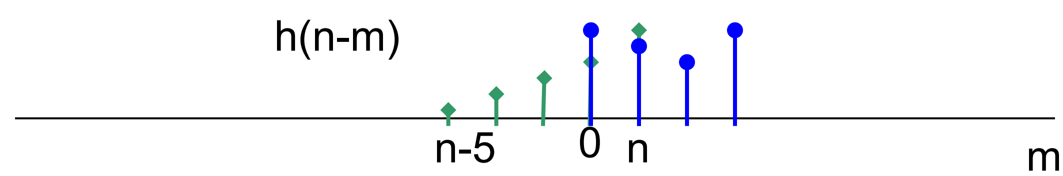
- The convolution with h(n) can be considered as the weighted average in the neighborhood of f(n), with the filter coefficients being the weights
  - sample f(n-m) is multiplied by h(m) (h(m) needs to be flipped)
- Signal length before and after filtering
  - Original signal length: N
  - Filter length: K
  - Filtered signal length: N+K-1

Example of 1D Discrete Convolution

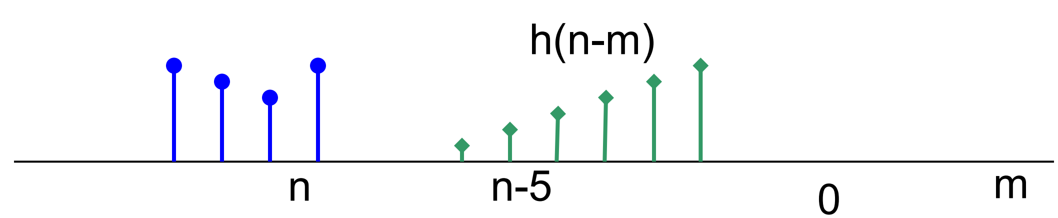




(a) n < 0, g(n) = 0



(b) 0 ≤ n ≤ 8, g(n) > 0



(c) n > 8, g(n) = 0

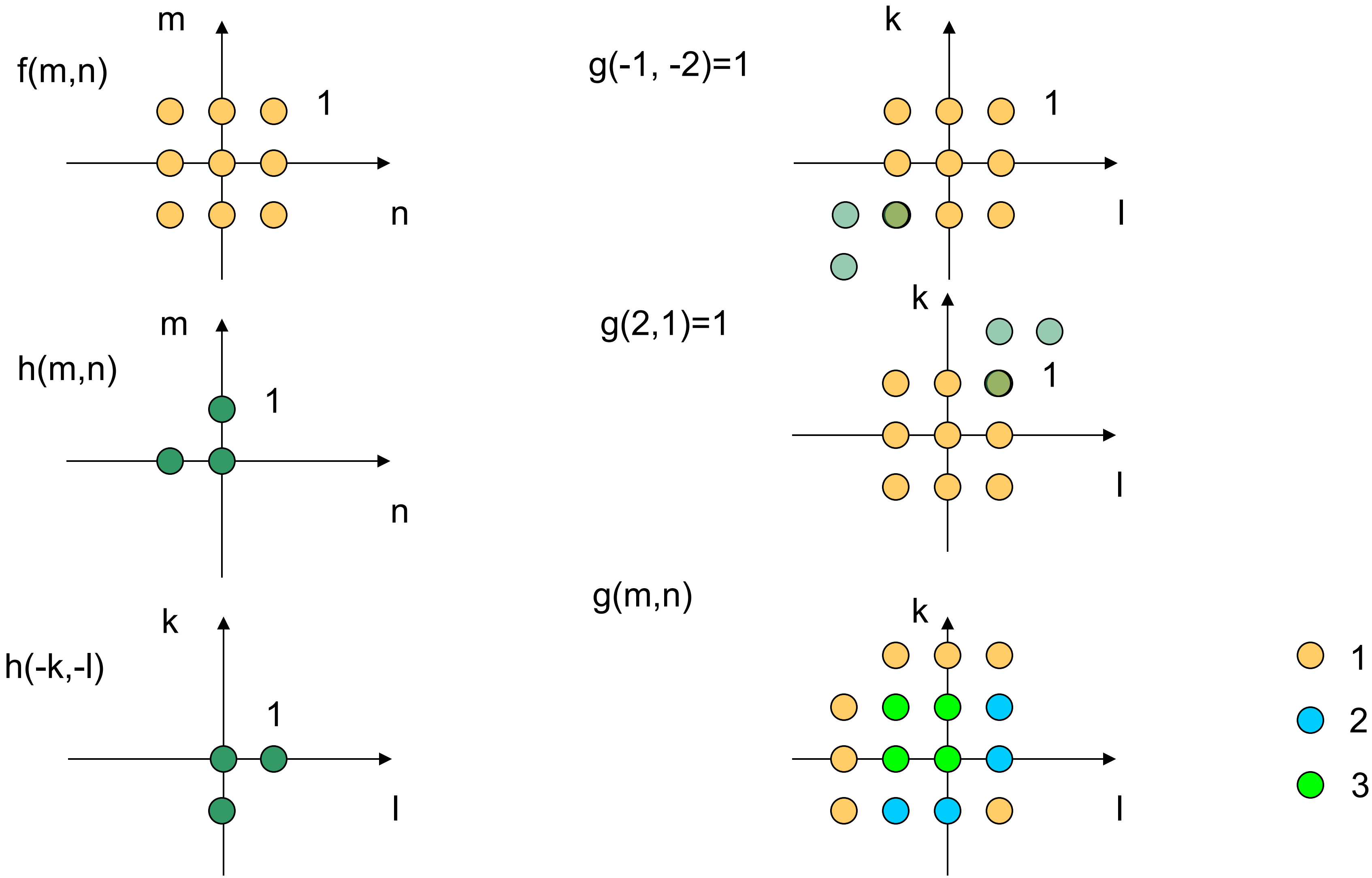
Convolution of 2D Discrete Signals

$$g(m,n) = f(m,n) * h(m,n) = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(m-k,n-l)h(k,l)$$

$$= \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f(k,l)h(m-k,n-l)$$

- Each new pixel g(m,n) is a weighted average of its neighboring pixels in the original image:
  - Pixel f(m-k,n-l) is weighted by h(k,l)
- We may use matrices to represent both signal (F) and filter (H) and use F*H to denote the convolution

Example of 2D Discrete Convolution



Example: Averaging and Weighted Averaging

100	100	100	100	100
100	200	205	203	100
100	195	200	200	100
100	200	205	195	100
100	100	100	100	100

$\frac{1}{9}\times$

1	1	1
1	1	1
1	1	1

$\frac{1}{16}\times$

1	2	1
2	4	2
1	2	1

100	100	100	100	100
100	144	167	145	100
100	167	200	168	100
100	144	166	144	100
100	100	100	100	100

100	100	100	100	100
100	156	176	158	100
100	174	201	175	100
100	156	175	156	100
100	100	100	100	100

Only those pixels that are highlighted are computed by convolution. Boundary pixels are kept as the original ones.

Example

Original image

Average filtered image

$\frac{1}{9}\times$

1	1	1
1	1	1
1	1	1

Weighted Average filtered image

$\frac{1}{16}\times$

1	2	1
2	4	2
1	2	1

Example: Edge Detection

1	0	-1
1	0	-1
1	0	-1

flip

-1	0	1
-1	0	1
-1	0	1

100	100	100	100	100
100	200	205	203	100
100	195	200	200	100
100	200	205	195	100
100	100	100	100	100

-1	0	1
-2	0	2
-1	0	1

flip

1	0	-1
2	0	-2
1	0	-1

	-205	-8	205	
	-310	-3	310	
	-205	0	205	

	-310	-11	310	
	-410	-8	410	
	-310	5	310	

Example of Sobel Edge Detector

Original image

Filtered image by H_x

-1	-2	-1
0	0	0
1	2	1

Filtered image by H_y

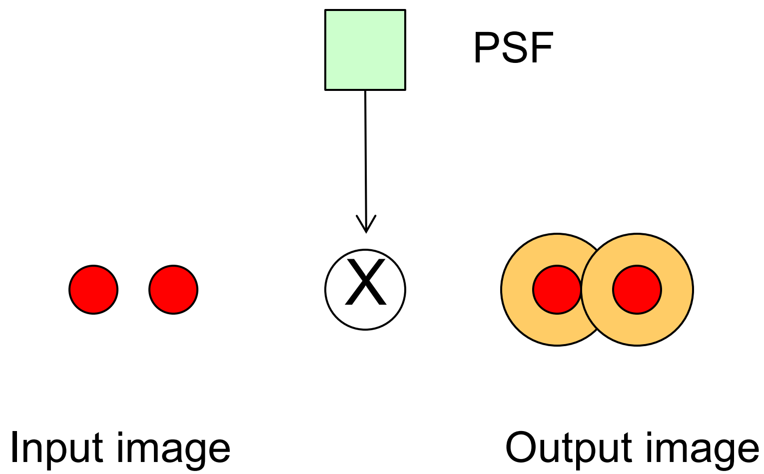
-1	0	1
-2	0	2
-1	0	1

What does h(m,n) mean?

- Any operation that is linear and shift invariant can be described by a convolution with a filter h(m,n)!
- h(m,n) is the impulse response of the system (i.e. output of the system to an impulse input $\delta(m,n)$)
- Better known as point spread function, indicating how a single point (i.e. an impulse) in the original image would be spread out in the output image

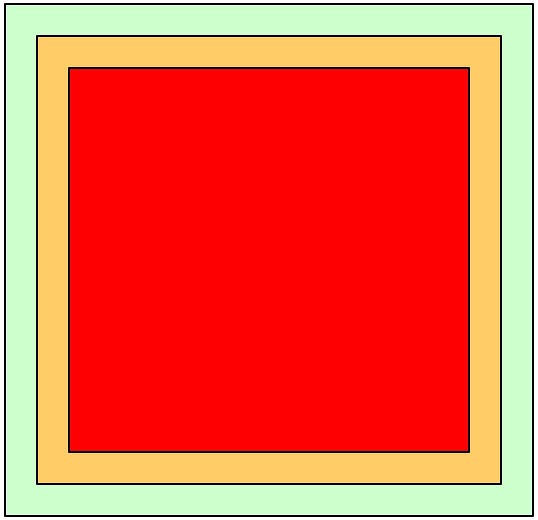
Point Spread Function

- The point spread function of an imaging system (e.g. a camera or a medical imaging system) describes the resolution of the system:
  - Two object points cannot be separated if they are closer than the support of the point spread function!



Boundary of Filtered Image

- An image of size MxN convolving with a filter of size KxL will yield an image of size (M+K-1,N+L-1)
- If the filter is symmetric with (2k+1)x(2k+1) samples, the convolved image should have an extra boundary of thickness k on each side outside the original image (outer boundary, green). The values along the outer boundary depend on the assumed pixel values outside the original image
- Filtered values in the inner boundary (orange) of k pixels inside the original image also depend on the assumed pixel values outside the original image
- Filtered values in the valid region (red) only depend on the available image values.



Orange+Red: original image size
Red: Valid part of the output image (does not depend on pixels outside the original image)
Orange: inner boundary
Green: outer boundary

Boundary Treatment: Zero Padding

- MxN image convolved with KxL filter -> (M+K-1)x(N+L-1) image
- Filtered values at the outer and inner boundary of the output image depend on the assumed value of the pixels outside the original image
- Zero padding: Assuming pixel values are 0 outside the original image

0	0	0	0	0
0	200	205	203	0
0	195	200	200	0
0	200	205	195	0
0	0	0	0	0

Actual image pixels

Extended pixels

$\frac{1}{9} \times$

1	1	1
1	1	1
1	1	1

Outer boundary

200/9	(200+205)/9	…	…	…
(200+195)/9	(200+205+195 +200)/9	…	…	…
…	---	…	…	…
…	…	…	…	…
…	…	…	…	…

Inner boundary

Boundary Treatment: Symmetric Extension

- Assuming pixels values outside the image are the same as their mirroring pixels inside the image
  - Lead to less discontinuity in the filtered image along the outer and inner boundaries

200	200	205	203	203
200	200	205	203	203
195	195	200	200	200
200	200	205	195	195
200	200	205	195	195

Actual image pixels

Extended pixels

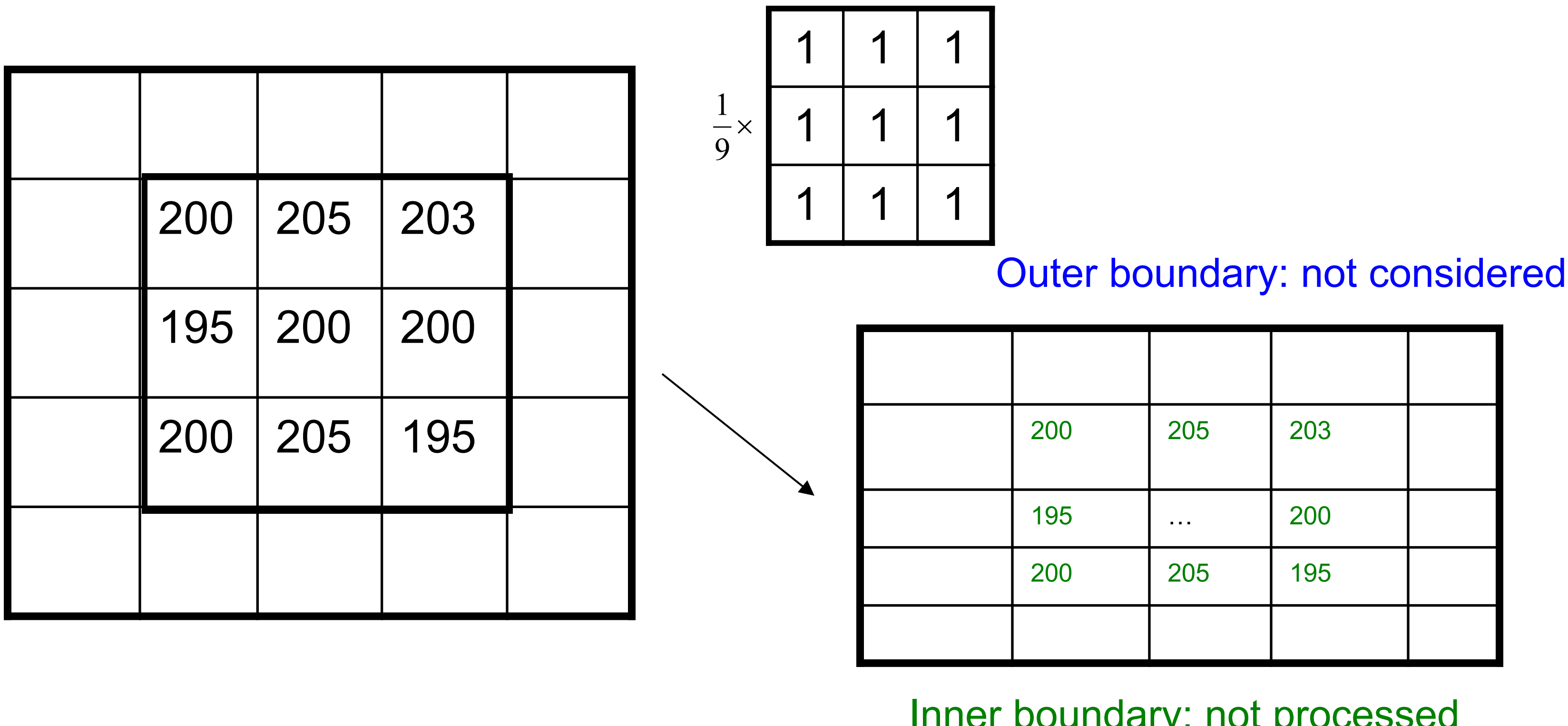
$\frac{1}{9}\times$

1	1	1
1	1	1
1	1	1

(200+200 +200+200)/9	(200+200+205 +200+200+205)/9	…	…	…
(200+200 +200+200 +195+195)/9	(200+200+205 +200+200+205 +195+195+200)/9	…	…	…
…	---	…	…	…
…	…	…	…	…
…	…	…	…	…

Simplified Boundary Treatment

- Filtered image size=original image size
- Only compute in the valid region.
  - Assign 0 (for high/band pass filters) or keep original value (for low pass) in the inner boundary
- Resulting image is correct only in the “valid” region



Example: Simplified Boundary Treatment

100	100	100	100	100
100	200	205	203	100
100	195	200	200	100
100	200	205	195	100
100	100	100	100	100

$\frac{1}{9}\times$

1	1	1
1	1	1
1	1	1

100	100	100	100	100
100	144	167	145	100
100	167	200	168	100
100	144	166	144	100
100	100	100	100	100

$\frac{1}{16}\times$

1	2	1
2	4	2
1	2	1

100	100	100	100	100
100	156	176	158	100
100	174	201	175	100
100	156	175	156	100
100	100	100	100	100

Sample Matlab Program (With Simplified Boundary Treatment)

```
% readin bmp file
x = imread('lena.bmp');
[xh xw] = size(x);
y = double(x);
```

```
% define 2D filter
h = ones(5,5)/25;
[hh hw] = size(h);
hhh = (hh - 1) / 2;
hhw = (hw- 1) / 2;
```

```
% linear convolution, assuming the filter is non-separable (although this example filter is separable)
z = y; %or z=zeros(xh,xw) if not low-pass filter
for m = hhh + 1:xh - hhh,
              %skip first and last hhh rows to avoid boundary problems
   for n = hhw + 1:xw - hhw,
              %skip first and last hhw columns to avoid boundary problems
      tmpy = 0;
      for k = -hhh:hhh,
         for l = -hhw:hhw,
            tmpv = tmpv + y(m - k,n – l)* h(k + hhh + 1, l + hhw + 1);
                          %h(0,0) is stored in h(hhh+1,hhw+1)
         end
      end
      z(m, n) = tmpv;
%for more efficient matlab coding, you can replace the above loop with
              z(m,n)=sum(sum(y(m-hhh:m+hhh,n-hhw:n+hhw).*flip(h)))
   end
end
```

Separable Filters

- A filter is separable if $h(m, n)=h_x(m)h_y(n)$.
- Matrix representation

$$H = h_x h_y^T$$

  – Where h_x and h_y are column vectors
- Example

$$H_x = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}; \quad H_y = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Separable Filtering

- If h(m,n) is separable, the 2D convolution can be accomplished by first applying 1D filtering along each row using $h_y(n)$, and then applying 1D filtering to the intermediate result along each column using the filter $h_x(n)$ (or column filtering followed by row filtering)
- Proof

$$
\begin{aligned}
f(m,n) * h(m,n) &= \sum_k \sum_l f(m-k, n-l) h_x(k) h_y(l) \\
&= \sum_k \left(\sum_l f(m-k, n-l) h_y(l) \right) h_x(k) \\
&= \sum_k g_y(m-k, n) h_x(k) \\
&= (f(m,n) * h_y(n)) * h_x(m)
\end{aligned}
$$

Results Using Sobel Filters

Original image

Filtered image by H_x

Filtered image by H_y

$$H_x = \frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}; \quad H_y = \frac{1}{4}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

What do Hx and Hy do?

Computation Cost: Non-Separable vs. Separable Filtering

- Suppose: Image size M*N, filter size K*L. Ignoring outer boundary pixels
- Non-separable filtering
  - Weighted average on each pixel: K*L mul; K*L – 1 add.
  - For all pixels: M*N*K*L mul; M*N*(K*L-1) add.
  - When M=N, K=L: M^2K^2 mul + $M^2(K^2-1)$ add.
- Separable filtering:
  - Each pixel in a row: L mul; L-1 add.
  - Each row: N*L mul; N*(L-1) add.
  - M rows: M*N*L mul; M*N*(L-1) add.
  - Each pixel in a column: K mul; K-1 add.
  - Each column: M*K mul; M*(K-1) add.
  - N columns: N*M*K mul; N*M*(K-1) add.
  - Total: M*N*(K+L) mul; M*N*(K+L-2) add.
  - When M=N, K=L: $2M^2K$ mul; $2M^2(K-1)$ add.
    - Significant savings if K (and L) is large!

MATLAB Function: conv2( )

- Matlab functions
  - C=conv2(H,F,shape)
    - Shape='full' (Default): C includes both outer and inner boundary, using zero padding
    - Shape="same": C includes the inner boundary, using zero padding
    - Shape="valid': C includes the convolved image without the inner boundary, computed without using pixels outside the original image
  - C=conv2(h1,h2,F,shape)
    - Separable filtering with h1 for column filtering and h2 for row filtering

Python Function: conv2( )

- Python functions
  - C=scipy.signal.convolve2d(*in1*, *in2*, *mode)*
    - mode ='full' (Default): C includes both outer and inner boundary, using zero padding
    - mode="same": C includes the inner boundary, using zero padding
    - mode="valid': C includes the convolved image without the inner boundary, computed without using pixels outside the original image

Notes about implementation

- Input image needs to be converted to float or double
  - MATLAB imread and Python cv2.imread return unsigned characters
  - *Never do numerical operations on unsigned character type!*
- Output image value may not be in the range of (0,255) and may not be integers
- To display or save the output image properly
  - Renormalize to (0,255) using a two-pass operation
    - First pass: save directly filtered value in an intermediate floating-point array
    - Second pass: find minimum and maximum values of the intermediate image, renormalize to (0,255) and rounding to integers
      - F= round((F1-fmin)*255/(fmax-fmin))
  - To display the unnormalized image directly in MATLAB, use imagesc(img). Or imshow(img, [ ]).
  - In python: cv2.imshow('fig_name',img), cv2.waitKey() or plt.imshow(img) (check Lecture note 1: ContrastEnhancement page 24)

Convolution Theorem

- Convolution Theorem

$$f * h \Leftrightarrow F \times H, \quad f \times h \Leftrightarrow F * H$$

- Proof

$$g(m,n) = f(m,n) * h(m,n) = \sum_k \sum_l f(m-k, n-l) h(k,l)$$

FT on both sides

$$G(u,v) = \sum_{m,n} \sum_{k,l} f(m-k, n-l) h(k,l) e^{-j2\pi(mu+nv)}$$

$$= \sum_{m,n} \sum_{k,l} f(m-k, n-l) e^{-j2\pi((m-k)u+(n-l)v)} h(k,l) e^{-j2\pi(ku+lv)}$$

$$= \sum_{m,n} f(m-k, n-l) e^{-j2\pi((m-k)u+(n-l)v)} \sum_{k,l} h(k,l) e^{-j2\pi(ku+lv)}$$

$$= \sum_{m',n'} f(m', n') e^{-j2\pi(m'u+n'v)} \sum_{k,l} h(k,l) e^{-j2\pi(ku+lv)}$$

$$= F(u,v) \times H(u,v)$$

Another view of convolution theorem

$$f(m,n) * h(m,n) \quad \Leftrightarrow \quad F(u,v)H(u,v)$$

- F(u,v)H(u,v) = Modifying the signal’s each frequency component’ complex magnitude F(u,v) by H(u,v)

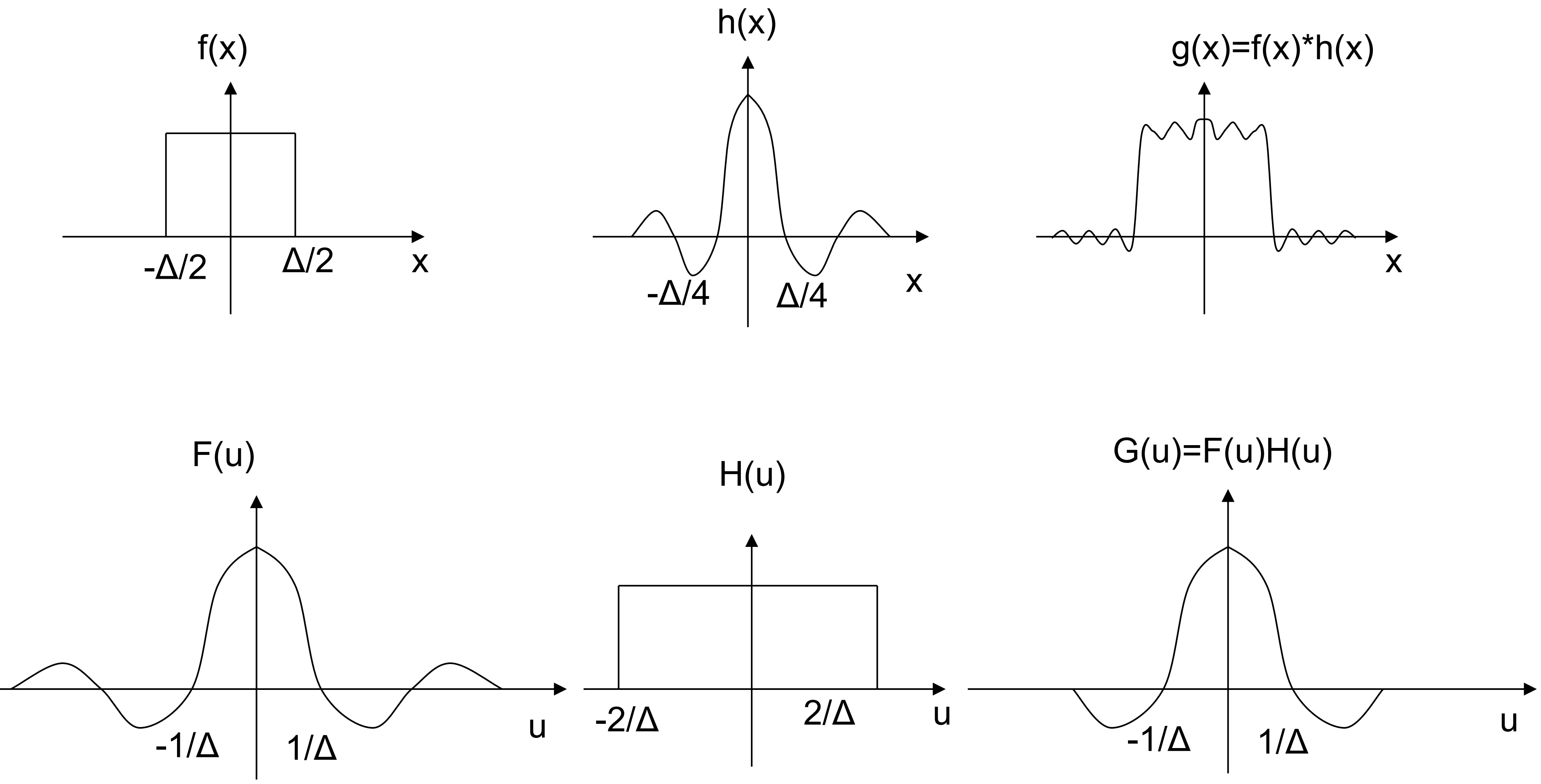
$$f(m,n) = \int_{-1/2}^{1/2}\int_{-1/2}^{1/2} F(u,v) e^{j2\pi(mu+nv)} du dv$$

↘

$$g(m,n) = \int_{-1/2}^{1/2}\int_{-1/2}^{1/2} F(u,v)H(u,v) e^{j2\pi(mu+nv)} du dv$$

- H(u,v) is also called Frequency Response of the 2D LSI system
  - exp{2π(um+vn)} -> H(u,v) exp{2π(um+vn)} = |H(u,v)|exp{2π(um+vn)+φ(H(u,v))}
  - Sinusoid (or complex exponential) input -> sinusoid (complex exponential) output!
  - H(u,v) describes how the magnitude and phase of a sinusoid input with frequency (u,v) are changed!

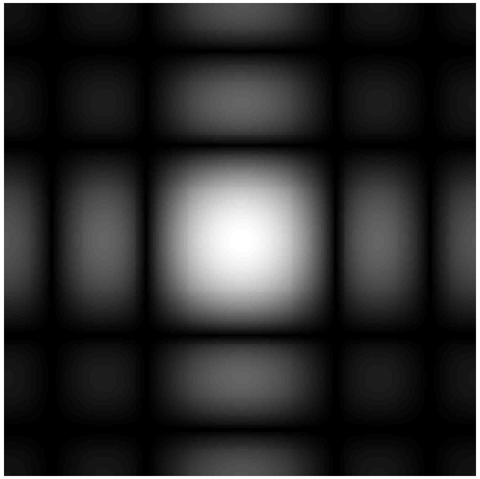
Explanation of Convolution in the Frequency Domain



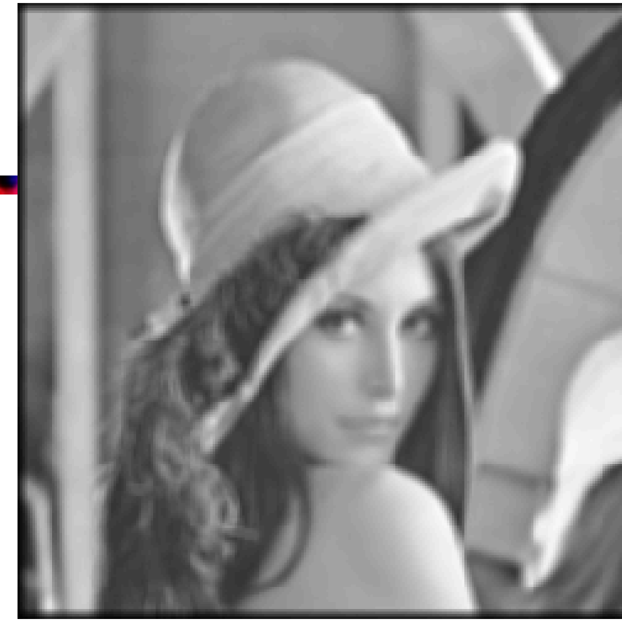
Example

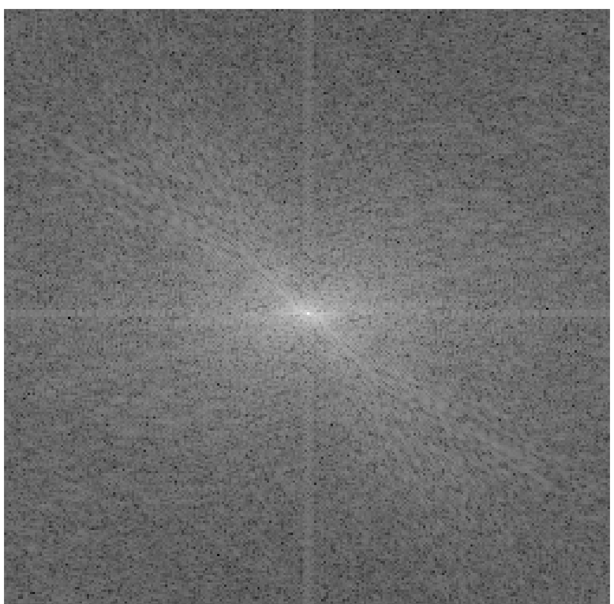
- Given a 2D filter, determine its frequency response. Apply to a given image, show original image and filtered image in pixel and freq. domain

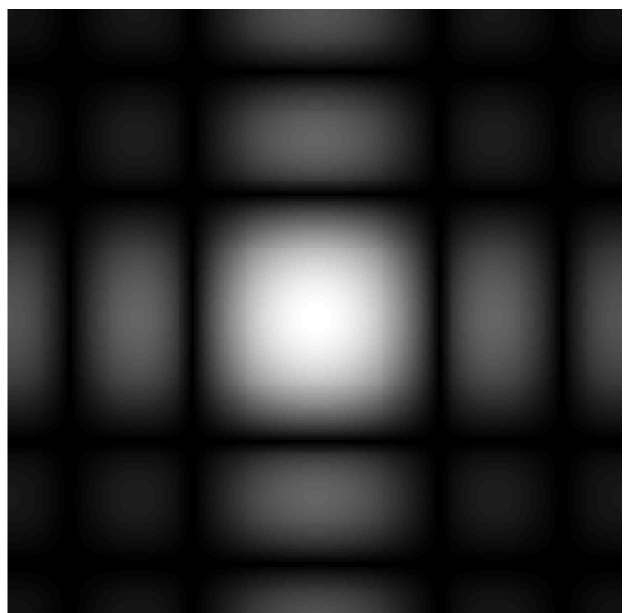
$$h = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

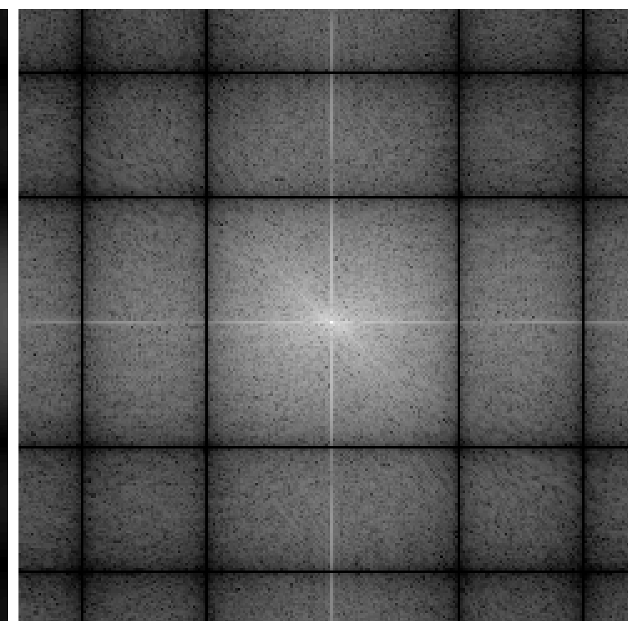


$$h = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$





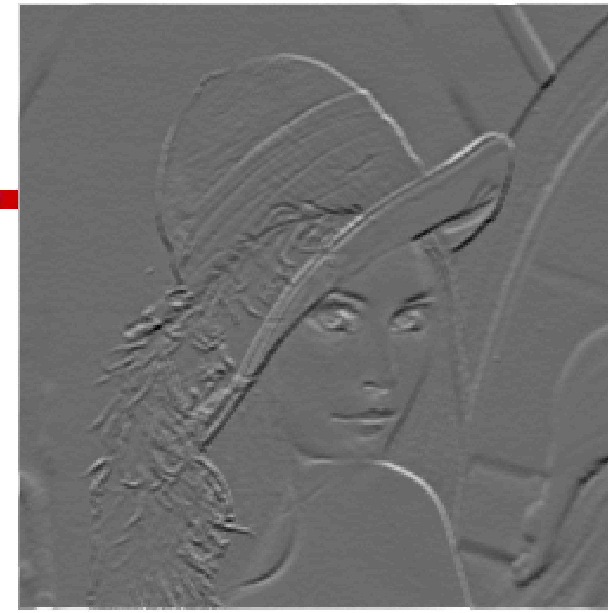


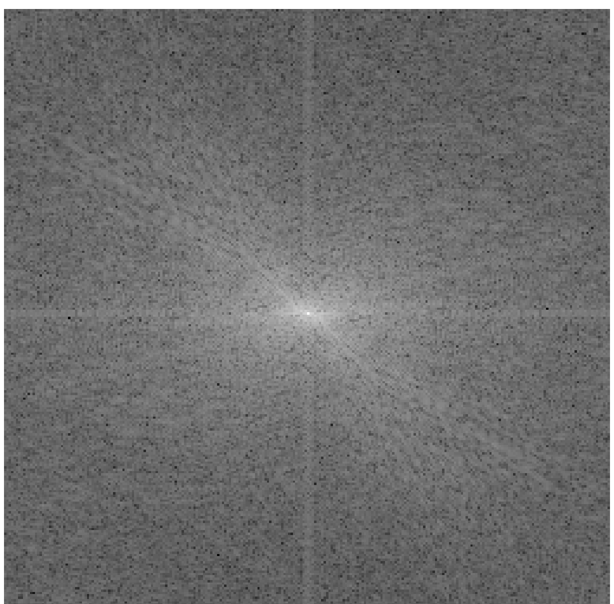


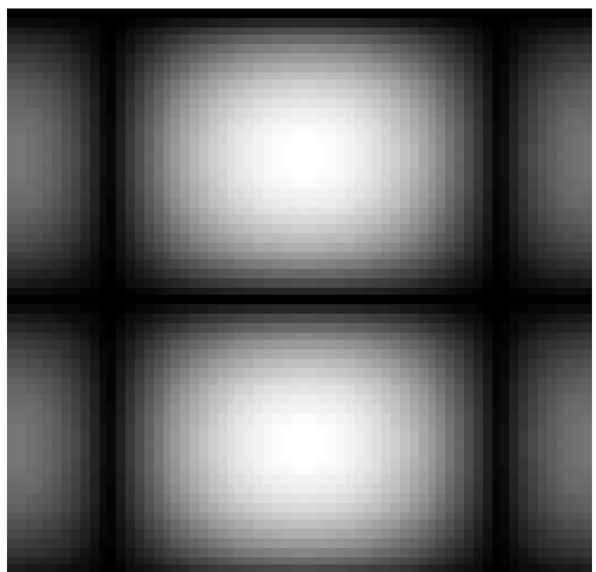
Matlab Program Used

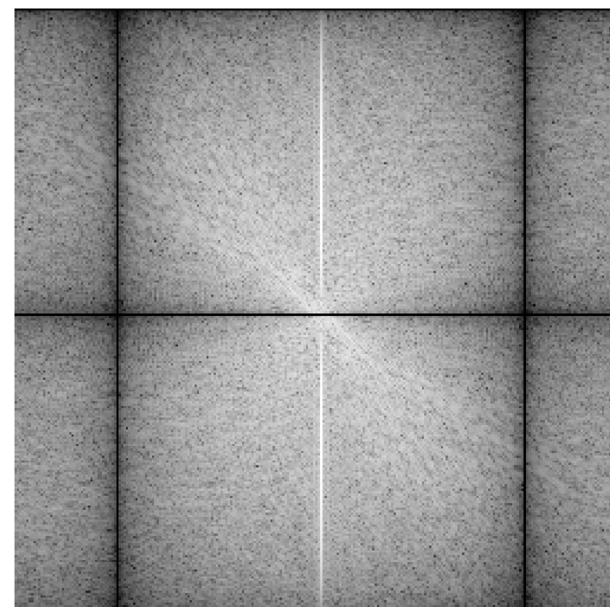
```
x = imread('lena256.bmp');
figure(1); imshow(x);
f = double(x);
ff=abs(fft2(f));
figure(2); imagesc(fftshift(log(ff+1))); colormap(gray);truesize;axis off;
h = ones(5,5)/9;
hf=abs(freqz2(h));
figure(3);imagesc((log(hf+1)));colormap(gray);truesize;axis off;
y = conv2(f, h);
figure(4);imagesc(y);colormap(gray);truesize;axis off;
yf=abs(fft2(y));
figure(5);imagesc(fftshift(log(yf+1)));colormap(gray);truesize;axis off;
```

$$H_1 = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

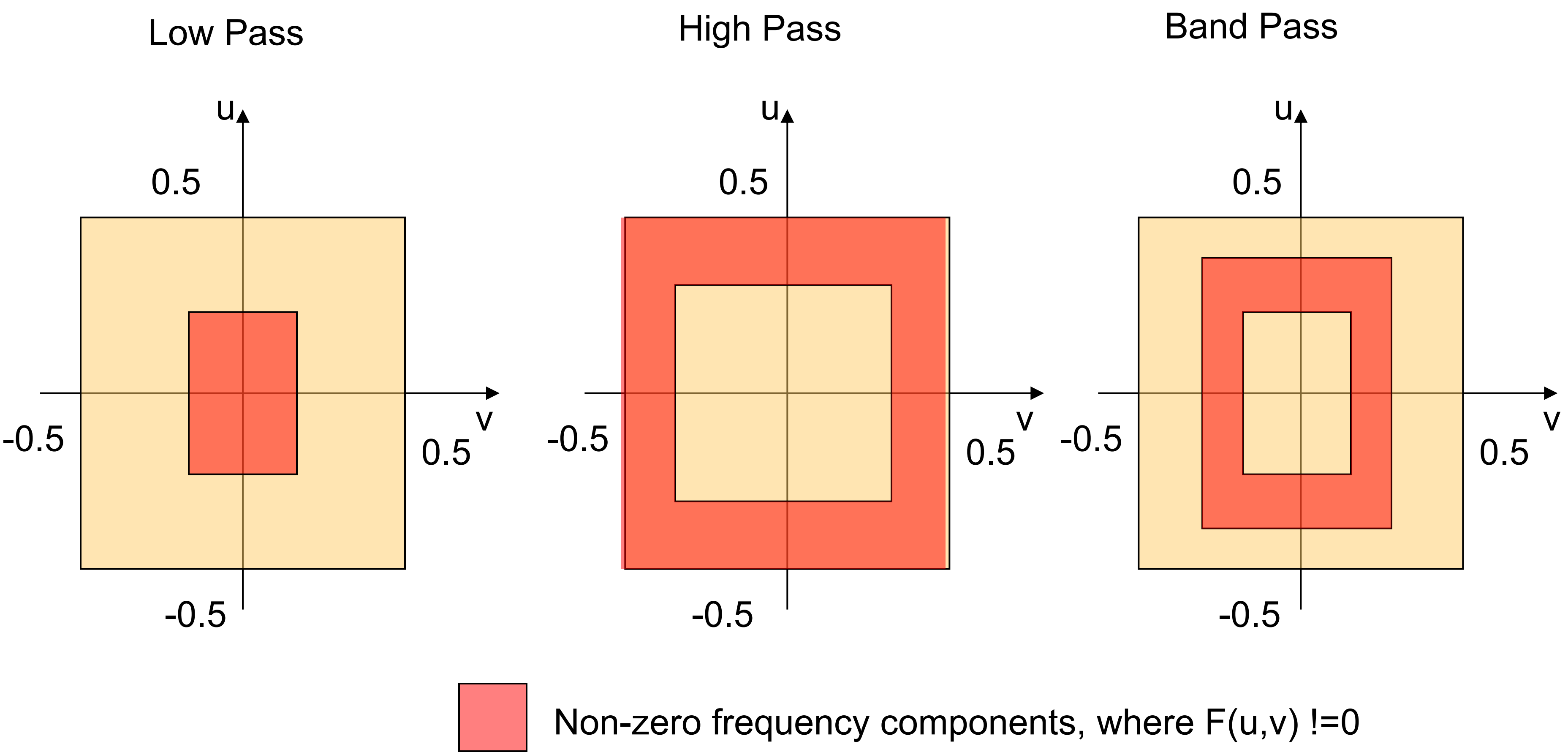








Typical Filter Types

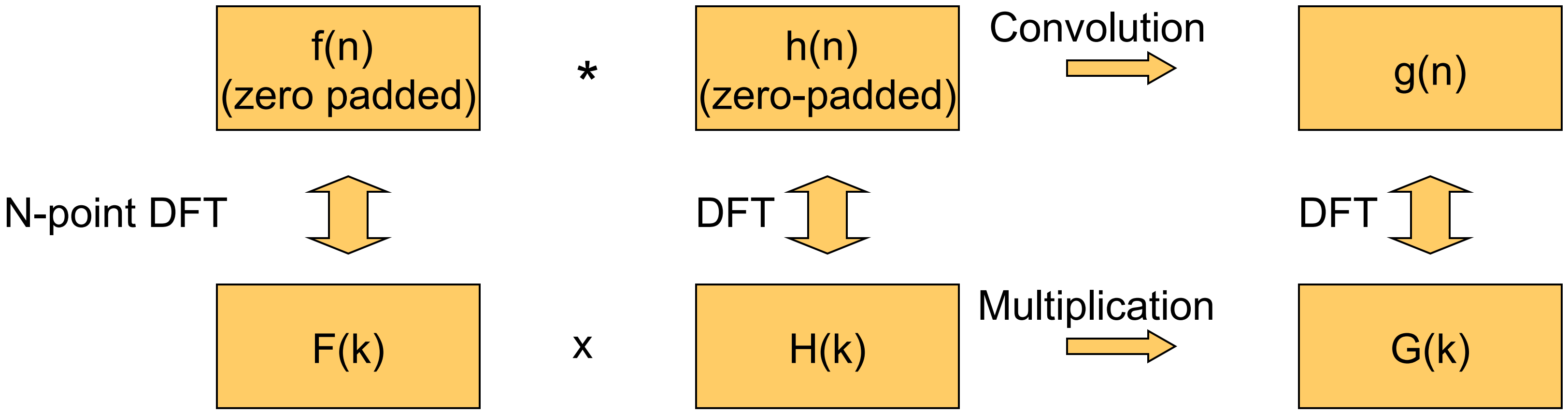


Filter Design

- The ideal low-pass, high-pass, band-pass filters not realizable
- Filter design (ECE-GY 6113 DSP1!)
  - Apply optimization technique to find the filter to match the desired frequency response as close as possible under some constraint (e.g. length)
  - FIR vs. IIR filters
  - Linear phase vs. non-linear phase
  - FIR: can realize linear phase
  - IIR filters can realize same transition band/attenuation with much shorter filter order than FIR!
  - Non-linear phase of IIR can be overcome through filtering forward and backward (filtfilt( ))!
- 1D filter design method can be used to design separable 2D filters
  - Not a focus of this class
- Image processing typically use short filters

Calculate Linear Convolution Using DFT

- 1D case
  - f(n) is length N_1, h(n) is length N_2
  - g(n) = f(n)*h(h) is length N = N_1+N_2-1.
  - To use DFT, need to **extend** f(n) and h(n) to length N by zero padding.
  - H(k) can be precalculated



Computing 2D Convolution Using 2D DFT

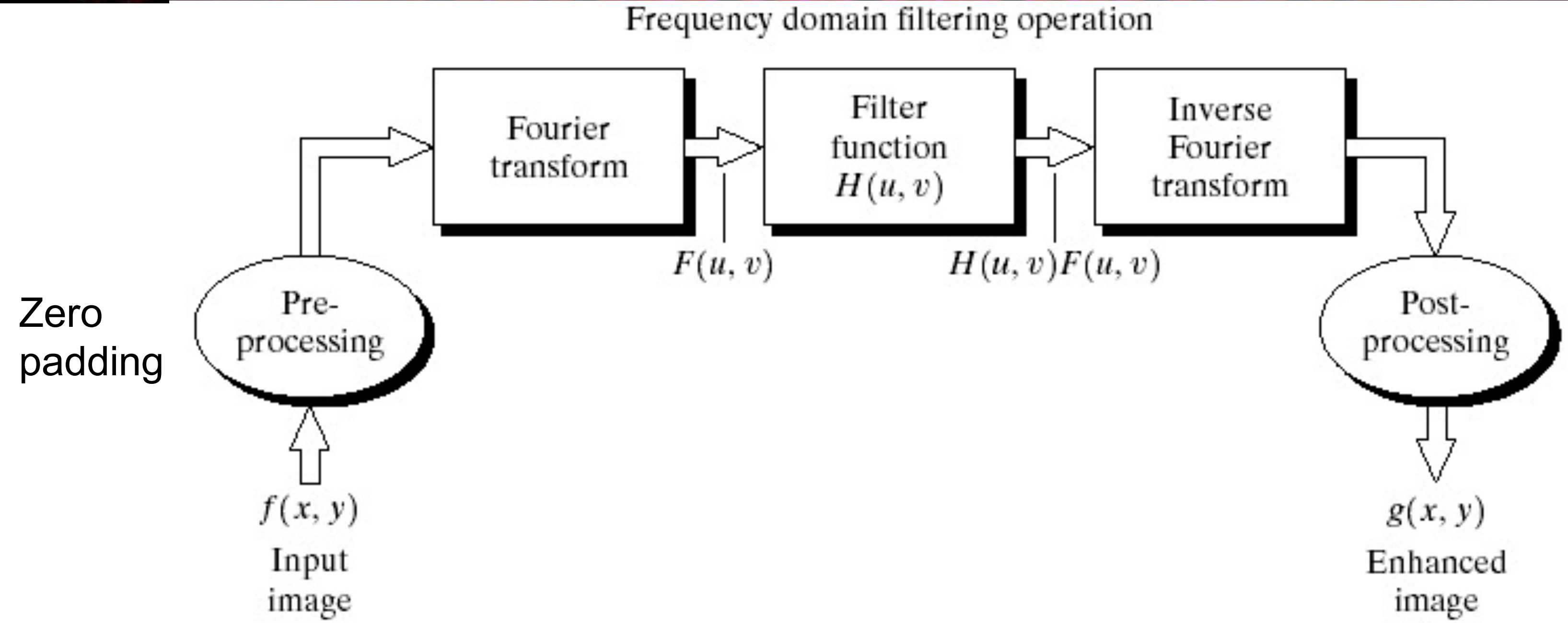


FIGURE 4.5 Basic steps for filtering in the frequency domain.

Relation between spatial and frequency domain operation:

$$g(x,y) = h(x,y) \otimes f(x,y) \quad \Leftrightarrow \quad G(u,v) = H(u,v)F(u,v)$$

$$h(x,y) = IDFT(H(u,v)), \quad H(u,v) = DFT(h(x,y)).$$

Image Filtering Using DFT

- Typically DFT size=image size. This corresponds to circular convolution, which differs from linear convolution at the inner boundaries. Only correct in the valid region.
- Circular convolution

$$f(n) \otimes h(n) = \sum_{k=0}^{N-1} f((n-k)\bmod(N))h(k)$$

$$f(n) \otimes h(n) \Leftrightarrow F_N(k) H_N(k)$$

- Image filters typically have short length to avoid boundary problems and ringing effect.
- For image filtering with short filters, it is more efficient to do convolution in the spatial domain directly.

Computation Complexity

- Image size MxM, Filter size KxK, assume M>>K
- Direct computation
  - Each pixel needs K^2 operations, total M^2 K^2
- Using FFT (length M)
  - Row transform: Each row: M log M, M rows: M^2 log M
  - Column transform: M^2 log M
  - Total 2 M^2 log M
- Convolution using 2D FFT
  - Need two transforms: 4 M^2 log M
- When M>>K, direct computation is just as good.

What you should know

- 2D linear convolution
  - weighted average of neighboring pixels
  - Filter=Point spread function (impulse response in 2D)
  - Separable filters: can filter row wise and then column wise
  - Computation of convolution: boundary treatment, separable filtering
- Convolution theorem
  - Convolution in space / time = Multiplication in FT domain
  - Frequency response of a linear system = FT of the filter
    - Describe how different frequency component of an input signal will be changed in magnitude and phase
  - Computation of convolution using DFT
- MATLAB function: conv2( ), freqz2( )
- Python: scipy.signal.convolve2d,scipy.signal.freqz( )