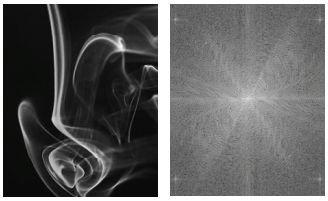


# Chapter 7:

# The Fourier Transform



## Introduction

- The Fourier transform allows us to perform tasks that would be impossible to perform any other way.
- It is more efficient to use the Fourier transform than a spatial filter for a large filter.
- The Fourier transform also allows us to isolate and process particular image frequencies.



## 1D Discrete Fourier Transform

- Definition of the One-Dimensional DFT

$$\mathbf{f} = [f_0, f_1, f_2, \ldots, f_{N-1}] \qquad \mathbf{F} = [F_0, F_1, F_2, \ldots, F_{N-1}],$$

$$F_u = \frac{1}{N}\sum_{x=0}^{N-1} \exp\left[-2\pi i \frac{xu}{N}\right] f_x \qquad (7.2)$$

- can be expressed as a matrix multiplication $F = \mathcal{F}f$,

$$\mathcal{F}_{m,n} = \frac{1}{N}\exp\left[-2\pi i \frac{mn}{N}\right] \Rightarrow \mathcal{F}_{m,n} = \frac{1}{N}\omega^{mn} \quad N \times N \text{ matrix}$$

$$\omega = \exp\left[\frac{-2\pi i}{N}\right] \Rightarrow \mathcal{F} = \frac{1}{N}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^1 & \omega^2 & \omega^3 & \omega^4 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega^8 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \omega^{12} & \cdots & \omega^{3(N-1)} \\ 1 & \omega^4 & \omega^8 & \omega^{12} & \omega^{16} & \cdots & \omega^{4(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \omega^{4(N-1)} & \cdots & \omega^{(N-1)^2} \end{bmatrix}$$



## 1D Discrete Fourier Transform

Example: Suppose $\boldsymbol{f}$ = [1, 2, 3, 4] so that $N$ = 4.

Find DFT coefficients F of f.

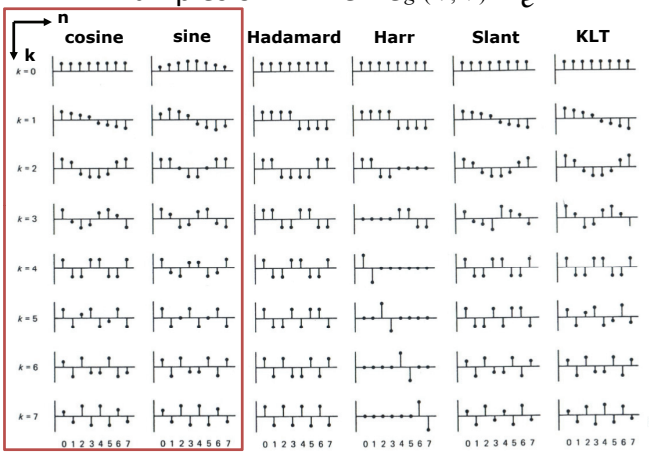
$$\begin{aligned} \omega &= \exp\left[\frac{-2\pi i}{4}\right] \\ &= \exp\left[-\frac{\pi i}{2}\right] \\ &= \cos\left(-\frac{\pi}{2}\right) + i\sin\left(-\frac{\pi}{2}\right) \\ &= -i. \end{aligned} \Rightarrow \mathcal{F} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & (-i)^2 & (-i)^3 \\ 1 & (-i)^2 & (-i)^4 & (-i)^6 \\ 1 & (-i)^3 & (-i)^6 & (-i)^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}$$

$$\mathbf{F} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 10 \\ -2+2i \\ -2 \\ -2-2i \end{bmatrix}$$



## Examples of 1-D Kernels

$$g(n,k) = e^{\frac{-j2\pi nk}{N}}$$




## THE INVERSE DFT

$$F_u = \frac{1}{N}\sum_{x=0}^{N-1} \exp\left[-2\pi i \frac{xu}{N}\right] f_x. \qquad (7.2)$$

$$f_x = \sum_{u=0}^{N} \frac{1}{} \exp\left[2\pi i \frac{xu}{N}\right] F_u \qquad (7.3)$$

✓ If you compare Equation (7.3) with Equation 7.2 you will see that there are really a few differences:

1. There is no scaling factor $1/N$
2. The sign inside the exponential function has been changed to positive.
3. The index of the sum is $u$, instead of $x$

## IDFT

$$f = \mathcal{F}^{-1}F$$

$$\mathcal{F}^{-1} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \overline{\omega}^1 & \overline{\omega}^2 & \overline{\omega}^3 & \overline{\omega}^4 & \cdots & \overline{\omega}^{N-1} \\ 1 & \overline{\omega}^2 & \overline{\omega}^4 & \overline{\omega}^6 & \overline{\omega}^8 & \cdots & \overline{\omega}^{2(N-1)} \\ 1 & \overline{\omega}^3 & \overline{\omega}^6 & \overline{\omega}^9 & \overline{\omega}^{12} & \cdots & \overline{\omega}^{3(N-1)} \\ 1 & \overline{\omega}^4 & \overline{\omega}^8 & \overline{\omega}^{12} & \overline{\omega}^{16} & \cdots & \overline{\omega}^{4(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \overline{\omega}^{N-1} & \overline{\omega}^{2(N-1)} & \overline{\omega}^{3(N-1)} & \overline{\omega}^{4(N-1)} & \cdots & \overline{\omega}^{(N-1)^2} \end{bmatrix} \qquad \overline{\omega} = \frac{1}{\omega} = \exp\left[\frac{2\pi i}{N}\right]$$

```
a =

     1     2     3     4     5     6

>> fft(a')

ans =

  21.0000
  -3.0000 + 5.1962i
  -3.0000 + 1.7321i
  -3.0000
  -3.0000 - 1.7321i
  -3.0000 - 5.1962i
```

CENGAGE Learning

## Properties of DFT: **LINEARITY**

- ✓ This is a direct consequence of the definition of the DFT as a matrix product.
- ✓ Suppose f and g are two vectors of equal length, and $p$ and $q$ are scalars, with $h = pf + qg$.
- ✓ If F, G, and H are the DFT's of f, g, and h, respectively, we have $H = pF + qG$

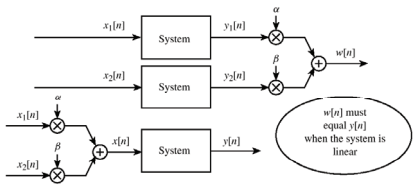



**Figure 5.17** Testing linearity by checking the interchange of operations.

CENGAGE Learning

## LINEAR SYSTEM EXAMPLES

- Examples: Let α=1, β=2
  - $x_1[n]$ ={2 3 5} (-1 ≤ n ≤ 1), otherwise x[n] = 0
  - $x_2[n]$ ={3 2 1} (-1 ≤ n ≤ 1), otherwise x[n] = 0
  - $Q1: y[n] = x[n]^2$
  - $Q2: y[n] = x[-n]$

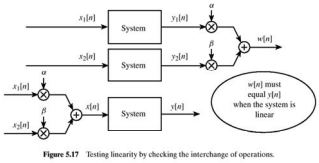
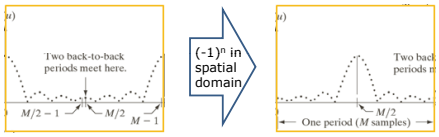



**Figure 5.17** Testing linearity by checking the interchange of operations.

CENGAGE Learning

## Properties of DFT: **SHIFTING**

- ✓ Suppose we multiply each element $x_n$ of a vector $x$ by $(-1)^n$. In other words, we change the sign of every second element.
- ✓ Let the resulting vector be denoted x'. The DFT X' of x' is equal to the DFT X of x with the swapping of the left and right halves




CENGAGE Learning

## Properties of DFT: Shifting

```
>> x = [2 3 4 5 6 7 8 1];

>> x1=(-1).^[0:7].*x

x1 =

     2    -3     4    -5     6    -7     8    -1

>> X=fft(x')

  36.0000
  -9.6569 + 4.0000i
  -4.0000 - 4.0000i
   1.6569 - 4.0000i
   4.0000
   1.6569 + 4.0000i
  -4.0000 + 4.0000i
  -9.6569 - 4.0000i
```

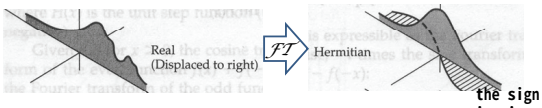
```
>> X1=fft(x1')

X1 =

   4.0000
   1.6569 + 4.0000i
  -4.0000 + 4.0000i
  -9.6569 - 4.0000i
  36.0000
  -9.6569 + 4.0000i
  -4.0000 - 4.0000i
   1.6569 - 4.0000i
```

Notice that the first four elements of X are the last four elements of X1 and vice versa.

CENGAGE Learning

## Properties of DFT

- **CONJUGATE SYMMETRY**
  - For real-valued signal f(x,y), $F(u) = F^*(-u)$




the sign of the imaginary part is opposite when u < 0

- **CONVOLUTION THEOREM**
  - convolution in the spatial domain is equivalent to multiplication in the frequency domain.

x*h <----> XH, xh <---->X*H

x(n) → h(n) → x(n)*h(n)

CENGAGE Learning

## 7.4 Properties of the One-Dimensional DFT

- **THE FAST FOURIER TRANSFORM** Appendix B

TABLE 7.1 *Comparison of FFT and direct arithmetic.*

| $2^n$ | Direct Arithmetic | FFT | Increase in Speed |
|---|---|---|---|
| 4 | 16 | 8 | 2.0 |
| 8 | 84 | 24 | 2.67 |
| 16 | 256 | 64 | 4.0 |
| 32 | 1024 | 160 | 6.4 |
| 64 | 4096 | 384 | 10.67 |
| 128 | 16384 | 896 | 18.3 |
| 256 | 65536 | 2048 | 32.0 |
| 512 | 262144 | 4608 | 56.9 |
| 1024 | 1048576 | 10240 | 102.4 |

CENGAGE Learning

## 2D Discrete Fourier Transform

$$F(u,v)=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)} \Leftrightarrow f(x,y)=\sum_{u=0}^{M-1}\sum_{v=0}^{N-1}F(u,v)e^{j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

f(x,y): pixel values of an image of size MxN
u, v: discrete variables

F(0,0) = ?

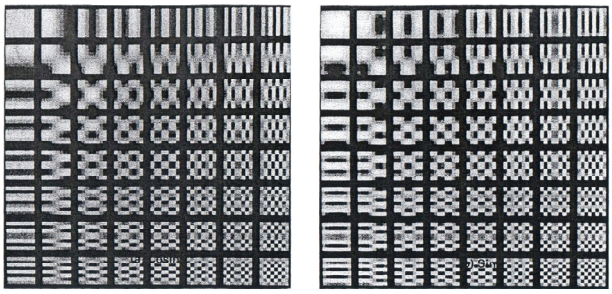
$$\begin{aligned}F(0,0)&=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)e^{-j2\pi\left(\frac{0x}{M}+\frac{0y}{N}\right)}\\&=\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}f(x,y)\\&=\bar{f}(x,y)\end{aligned}$$

CENGAGE Learning

## 8x8 2-D FT Coefficients

u
v

cosine sine



$$e^{j2\pi(ux+vy)}=\cos[2\pi(ux+vy)]+j\sin[2\pi(ux+vy)]$$

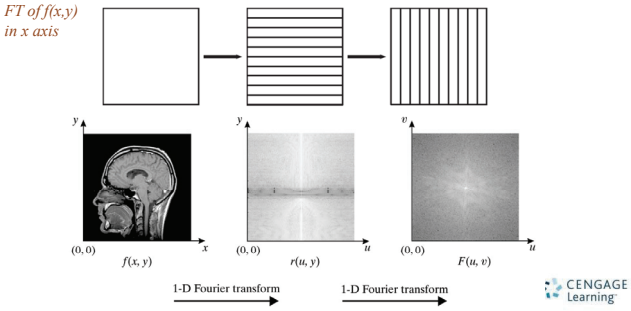
CENGAGE Learning

## Properties of the 2D DFT:Separability

- F(u,v) of f(x,y) can be calculated using two 1-D FT.

$$\int_{-\infty}^{\infty}r(u,y)e^{-j2\pi vy}dy=\int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty}f(x,y)e^{-j2\pi ux}dx\right]e^{-j2\pi vy}dy=\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}f(x,y)e^{-j2\pi(ux+vy)}dxdy=F(u,v)$$

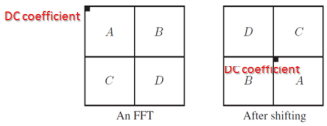
*FT of f(x,y) in x axis*




CENGAGE Learning

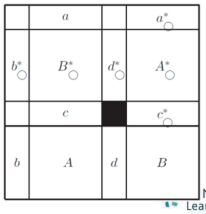
## Properties of the 2D DFT

- **SHIFTING** $f_p(x,y)$ by $(-1)^{x+y}$ before transform




- **CONJUGATE SYMMETRY**

$$F(u,v)=F^*(-u,-v)$$




NGAGE Learning

## How to Apply Convolution Theorem In Reality

✓ Suppose we wish to convolve an image $M$ with a spatial filter $S$

1. Pad $S$ with zeroes so that it is the same size as $M$; denote this padded result by $S'$ *: make the length of both data to fit $2^n$ for best performance of fft*
2. Form the DFTs of both $M$ and $S'$ to obtain $\mathcal{F}(M)$ and $\mathcal{F}(S')$.
3. Form the element-by-element product of these two transforms: $\mathcal{F}(M)\mathcal{F}(S')$
4. Take the inverse transform of the result: $\mathcal{F}^{-1}\{\mathcal{F}(M)\mathcal{F}(S')\}$

CENGAGE Learning

## Fourier Transforms in MATLAB

- ✓ `fft`: takes the DFT of a vector
- ✓ `ifft`: takes the inverse DFT of a vector
- ✓ `fft2`: takes the DFT of a matrix
- ✓ `ifft2`: takes the inverse DFT of a matrix
- ✓ `fftshift`: shifts a transform



## Fourier Transforms in MATLAB

e.g.

```
>> a=ones(8);
>> fft2(a)

ans =
    64     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

Note that the DC coefficient is indeed the sum of all the matrix values



## 7.6 Fourier Transforms in MATLAB

e.g.

*a: input*

```
>> a = [100 200; 100 200];
>> a = repmat(a,4,4)

ans =
     100   200   100   200   100   200   100   200
     100   200   100   200   100   200   100   200
     100   200   100   200   100   200   100   200
     100   200   100   200   100   200   100   200
     100   200   100   200   100   200   100   200
     100   200   100   200   100   200   100   200
     100   200   100   200   100   200   100   200
     100   200   100   200   100   200   100   200

>> af = fft2(a)

ans =
    9600     0     0     0 -3200     0     0     0
       0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0
       0     0     0     0     0     0     0     0
```

*af: DFT of a*



## 7.6 Fourier Transforms in MATLAB

e.g.

*a: input*

```
>> a = [zeros(8,4) ones(8,4)]
   a =
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
     0     0     0     0     1     1     1     1
```

*af: DFT then shifted*

```
>> af=fftshift(fft2(a));
>> round(abs(af))
ans =
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     9     0    21    32    21     0     9
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```



## Showing the Fourier Spectrum in Matlab: fftshow

```
function fftshow(f,type)

% Usage:  FFTSHOW(F,TYPE)
%
% Displays the fft matrix F using imshow, where TYPE must be one of
% 'abs' or 'log'.  If TYPE='abs', then then abs(f) is displayed; if
% TYPE='log' then log(1+abs(f)) is displayed.  If TYPE is omitted, then
% 'log' is chosen as a default.
%
%  Example:
%    c=imread('cameraman.tif');
%    cf=fftshift(fft2(c));
%    fftshow(cf,'abs')
%

if nargin<2,
  type='log';
end

if (type=='log')
  fl = log(1+abs(f));
  fm = max(fl(:));
  imshow(im2uint8(fl/fm))
elseif (type=='abs')
  fa=abs(f);
  fm=max(fa(:));
  imshow(fa/fm)
else
  error('TYPE must be abs or log.');
end;
```

FIGURE 7.9 *A function to display a Fourier transform.*

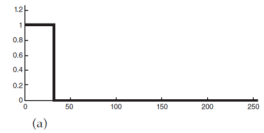


## DFT Example

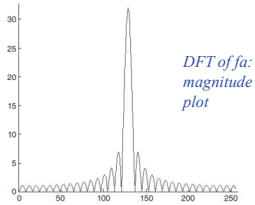
```
>> fa = [ones(1, 32), zeros(1, 224)];
>> fb = [ones(1, 64), zeros(1, 192)];
>> figure, plot(fa)
```

```
>> af = fft(fa)
>> aft = fftshift(af);
>> figure, plot(abs(aft))
```

*fa*




(a)




(b)

*DFT of fa: magnitude plot*

FIGURE 7.11 *A 1-D single edge signal* `fa` *and its FFT.*

## FIGURE 7.12

```
figure, plot(fb)
bf = fft(fb);
bft = fftshift(bf);
figure, plot(abs(bft))
```

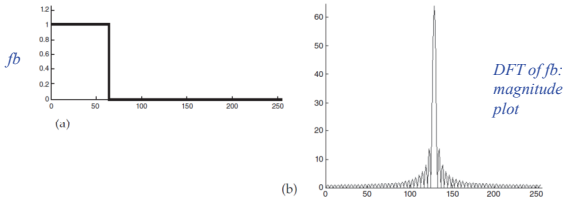



FIGURE 7.12 *A 1-D single edge signal* `fb` *and its FFT.*



## 2D DFT Example: Vertical Edge

```
>> a=[zeros(256,128) ones(256,128)];
```

```
>> af=fftshift(fft2(a));
>> fftshow(af,'log')
```

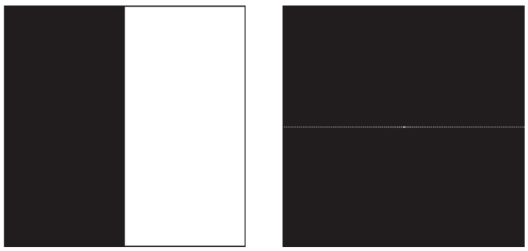


FIGURE 7.10 *A single edge and its DFT.*



## 2D DFT Example: Square Edge

```
>> a=zeros(256,256);
>> a(78:178,78:178)=1;
>> imshow(a)
>> af=fftshift(fft2(a));
>> figure,fftshow(af,'abe')
```

FIGURE 7.13 *A box and its DFT.*



## 2D DFT Example: Diagonal Edge

```
>> [x,y]=meshgrid(1:256,1:256);
>> b=(x+y<329)&(x+y>182)&(x-y>-67)&(x-y<73);
>> imshow(b)
>> bf=fftshift(fft2(b));
>> figure,fftshow(bf,'log')
```

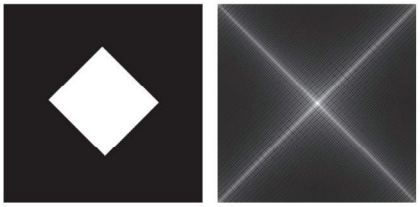


FIGURE 7.14 *A rotated box and its DFT.*



## 2D DFT Example: Circular Edge

```
>> [x,y]=meshgrid(-128:127,-128:127);
>> z=sqrt(x.^2+y.^2);
>> c=(z<15);
```

```
>> cf=fftshift(fft2(c));
>> fftshow(cf,'log')
```

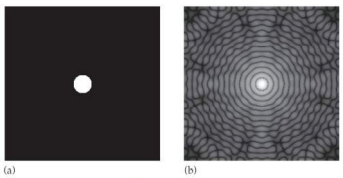
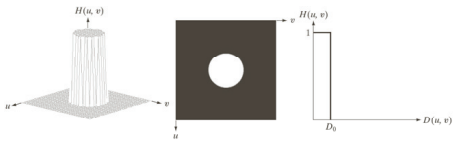



FIGURE 7.15 *(a) A circle and (b) its DFT.*

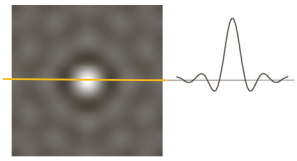


## Filtering in the Frequency Domain: Ideal LPF

$$H(u,v)=\begin{cases}1 & \text{if } D(u,v)\le D_0\\ 0 & \text{if } D(u,v)>D_0\end{cases}$$




ideal LPF in spatial domain



## Ideal LPF Example (contd.)

```
>> cm=imread('cameraman.tif');
>> cf=fftshift(fft2(cm));
>> figure,fftshow(cf,'log')
```

*ideal LPF*

```
>> cfl=cf.*c;
>> figure,fftshow(cfl,'log') % shown in the next page
```

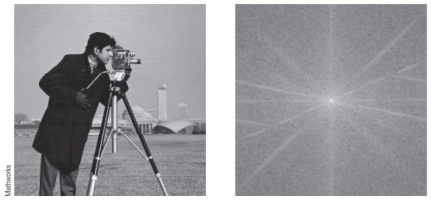


FIGURE 7.16 *The "cameraman" image and its DFT.*



## Ideal LPF Example

```
>> cfli=ifft2(cfl);
>> figure,fftshow(cfli,'abs')
```

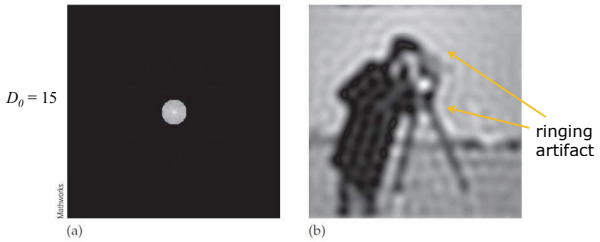



FIGURE 7.17 *Applying ideal low-pass filtering. (a) Ideal filtering on the DFT. (b) After inversion.*



## Ideal LPF

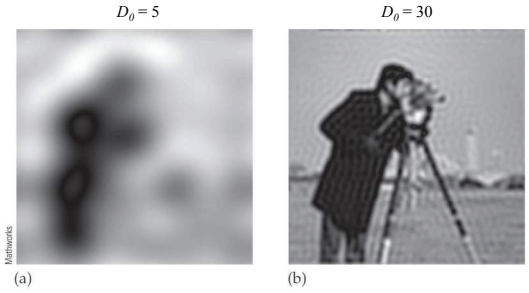


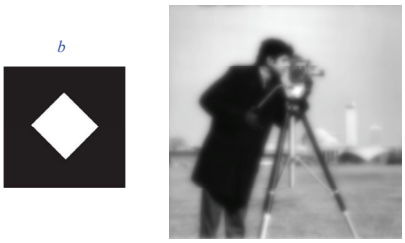

FIGURE 7.18 *Ideal low-pass filtering with different cutoffs. (a) Cutoff of 5. (b) Cutoff of 30.*



## 7.8 Filtering in the Frequency Domain

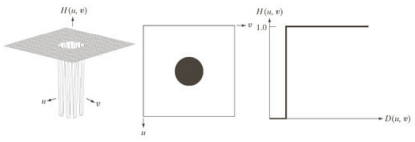
```
>> cfl = cf.*b
>> cfli = ifft2(cfl);
>> figure, fftshow(cfli, 'abs')
```

*b*





## Filtering in the Frequency Domain: Ideal HPF

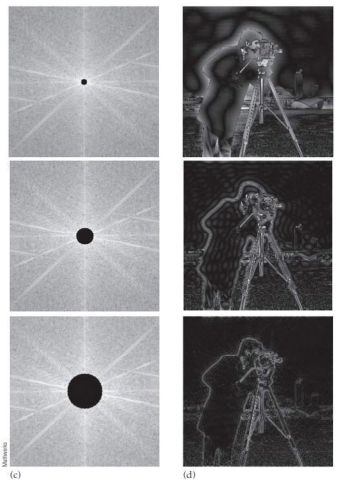


```
>> [x,y]=meshgrid(-128:127,-128:127);
>> z=sqrt(x.^2+y.^2);
>> c=(z>15);
```

```
>> cfh=cf.*c;
>> figure,fftshow(cfh,'log')
```

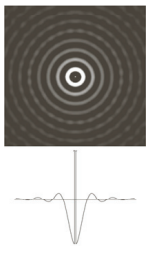
```
>> cfhi=ifft2(cfh);
>> figure,fftshow(cfhi,'abs')
```






## Ideal HPF

$$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \le D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases}$$



ringing artifact

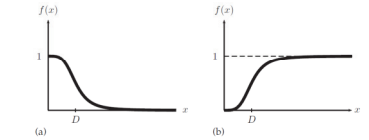
## Butterworth Filtering

- ✓ Ideal filter simply **cuts off** the Fourier coefficients at the specified distance from the center.
- ✓ It has the disadvantage of introducing unwanted artifacts (**ringing**) into the result
- ✓ One way of avoiding ringing artifacts is to use as a filter matrix, a circle with a less sharp cutoff**.**
- ✓ Butterworth and Gaussian filters

CENGAGE Learning

## nth Order Butterworth LPF & HPF

$$f(x) = \frac{1}{1 + (x/D)^{2n}} \qquad f(x) = \frac{1}{1 + (D/x)^{2n}}$$




*Lower n makes smoother transition.*

FIGURE 7.22 *Butterworth filter functions with $n = 2$. (a) Low pass. (b) High pass.*

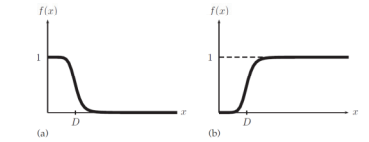



FIGURE 7.23 *Butterworth filter functions with $n = 4$. (a) Low pass. (b) High pass.*

CENGAGE Learning

## Butterworth in Matlab

```
function out=lbutter(im,d,n)
% LBUTTER(IM,D,N) creates a low-pass Butterworth filter
% of the same size as image IM, with cutoff D, and order N
%
% Use:
%   x=imread('cameraman.tif');
%   l=lbutter(x,25,2);
%
height=size(im,1);
width=size(im,2);
[x,y]=meshgrid(-floor(width/2):floor((width-1)/2),-floor(height/2): ...
      floor((height-1)/2));
out=1./(1+(sqrt(2)-1)*((x.^2+y.^2)/d^2).^n);
```

```
function out=hbutter(im,d,n)
% HBUTTER(IM,D,N) creates a high-pass Butterworth filter
% of the same size as image IM, with cutoff D, and order N
%
% Use:
%   x=imread('cameraman.tif');
%   l=hbutter(x,25,2);
%

out=1-lbutter(im,d,n);
```

CENGAGE Learning

## Butterworth LPF

```
>> bl = lbutter(c,15,1);
>> cfbl = cf.*bl;
>> figure, fftshow(cfbl, 'log');
```

```
>> cfbli = ifft2(cfbl);
>> figure, fftshow(cfbli, 'abs')
```

*freq. spectrum after applying Butterworth LPF*

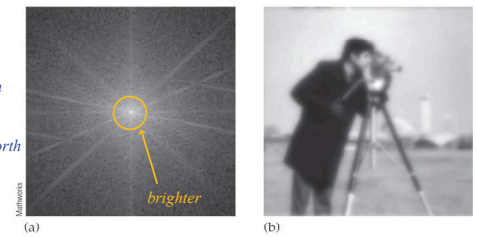



FIGURE 7.26 *Butterworth low-pass filtering. (a) The DFT after Butterworth low-pass filtering. (b) The resulting image.*

CENGAGE Learning

## Butterworth HPF

```
>> bh=hbutter(cm,15,1);
>> cfbh=cf.*bh;
>> figure,fftshow(cfbh,'log')
```

```
>> cfbhi=ifft2(cfbh);
>> figure,fftshow(cfbhi,'abs')
```

*freq. spectrum after applying Butterworth HPF*

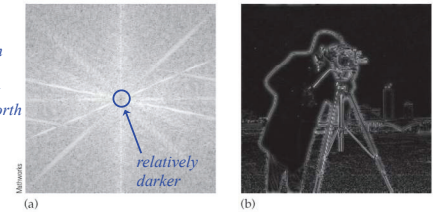



FIGURE 7.27 *Butterworth high-pass filtering. (a) The DFT after Butterworth high-pass filtering. (b) The resulting image.*

CENGAGE Learning

## Gaussian Filtering

```
>> g1=mat2gray(fspecial('gaussian',256,10));
>> cg1=cf.*g1;
>> fftshow(cg1,'log')
>> g2=mat2gray(fspecial('gaussian',256,30));
>> cg2=cf.*g2;
>> figure,fftshow(cg2,'log')
```

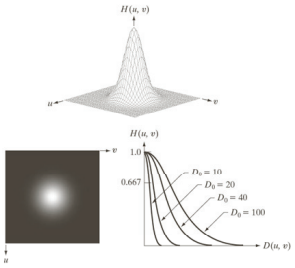
```
>> g=fspecial('gaussian',256,10);
>> format long, max(g(:)), format

ans =

   0.00158757552679
```

- The most popular filter mask.
- A wider function, with a large standard deviation, will have a low maximum.

CENGAGE Learning

## Gaussian LPF

$$H(u,v) = e^{-\frac{D^2(u,v)}{2\sigma^2}} = e^{-\frac{D^2(u,v)}{2D_0^2}}$$



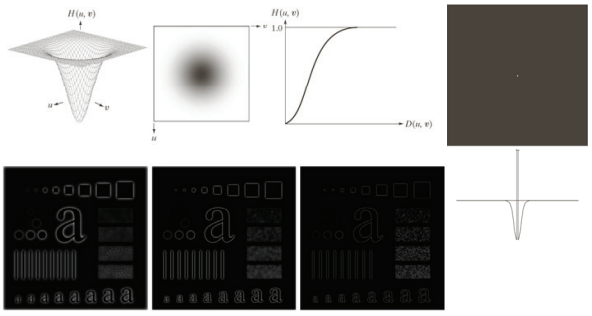

- no ringing artifact
- good for medical image proc.

a b
c d
e f

**FIGURE 4.48** (a) Original image. (b)–(f) Results of filtering using GLPFs with cutoff frequencies at the radii shown in Fig. 4.41. Compare with Figs. 4.42 and 4.45.

## Gaussian HPF

$$H(u,v) = 1 - e^{-\frac{D^2(u,v)}{2D_0^2}}$$



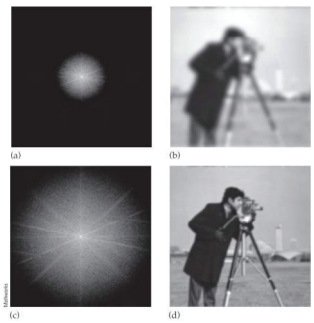



a b c

**FIGURE 4.56** Results of highpass filtering the image in Fig. 4.41(a) using a GHPF with $D_0$ = 30, 60, and 160, corresponding to the circles in Fig. 4.41(b). Compare with Figs. 4.54 and 4.55.

CENGAGE Learning

## Gaussian LPF Examples




```
>> cg11=ifft2(cg1);
>> cg12=ifft2(cg2);
>> fftshow(cg11,'abs');
>> fftshow(cg12,'abs');
```

**FIGURE 7.28** *Applying a Gaussian low-pass filter in the frequency domain. (a) $\sigma = 10$. (b) Resulting image. (c) $\sigma = 30$. (d) Resulting image.*

CENGAGE Learning

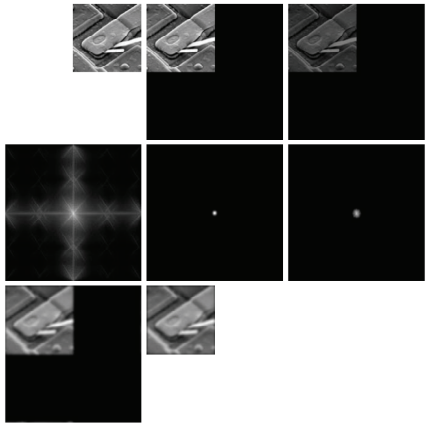
## Gaussian HPF Examples

```
>> h1=1-g1;
>> h2=1-g2;
>> ch1=cf.*h1;
>> ch2=cf.*h2;
>> chi1=ifft2(ch1);
>> chi2=ifft2(ch2);
>> fftshow(chi1,'abs')
>> figure,fftshow(chi2,'abs')
```


**FIGURE 7.29** *Applying a Gaussian high-pass filter in the frequency domain. (a) Using $\sigma = 10$. (b) Using $\sigma = 30$.*

CENGAGE Learning

## Procedure of Filtering in Frequency Domain



a b c
d e f
g h

**FIGURE 4.36**
(a) An $M \times N$ image, $f$. (b) Padded image, $f_p$ of size $P \times Q$. (c) Result of multiplying $f_p$ by $(-1)^{x+y}$. (d) Spectrum of $F_p$. (e) Centered Gaussian lowpass filter, $H$, of size $P \times Q$. (f) Spectrum of the product $HF_p$. (g) $g_p$, the product of $(-1)^{x+y}$ and the real part of the IDFT of $HF_p$. (h) Final result, $g$, obtained by cropping the first $M$ rows and $N$ columns of $g_p$.

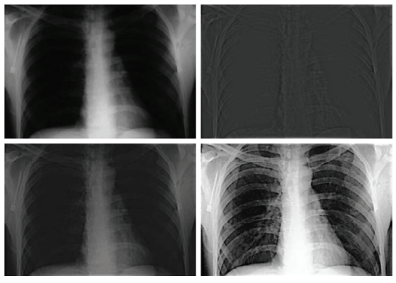
IGAGE ning

## High-Freq.-Emphasis Filter

$$g(x,y) = F^{-1}\{[k_1 + k_2 H_{HP}(u,v)]F(u,v)\}$$

original image

HFE filtered image



HP filtered image

HE of HFE filtered image

CENGAGE Learning

## Homomorphic Filtering

- An image $x(n_1,n_2)$ is created by "reflecting" the light from an object that has been "illuminated" by some light source.
- If the illuminating intensity $i(n_1,n_2)$ is a slowly varying component (controlling overall dynamic range = low freq. component), the reflecting source $r(n_1,n_2)$ is a fast varying component (affecting the local contrast = high freq. component),

$$x(n_1,n_2) = i(n_1,n_2)\cdot r(n_1,n_2)$$



## Homomorphic Filtering

- How about separating these two components and emphasizing/deemphasizing each one ?

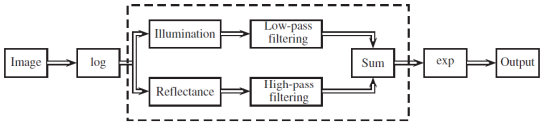
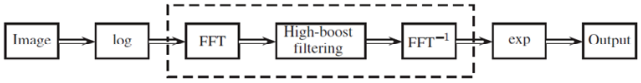



FIGURE 7.30 *A schema for homomorphic filtering.*

$$y(n_1,n_2) = \exp\left\{\begin{array}{l}\gamma_L \bullet LPF\left[\log x(n_1,n_2)\right] + \\ \gamma_H \bullet HPF\left[\log x(n_1,n_2)\right]\end{array}\right\}$$

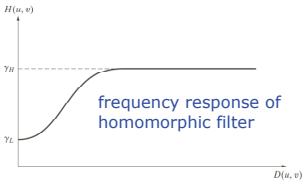


## Homomorphic Filtering




$\gamma_H > 1.0$

$\gamma_L < 1.0$

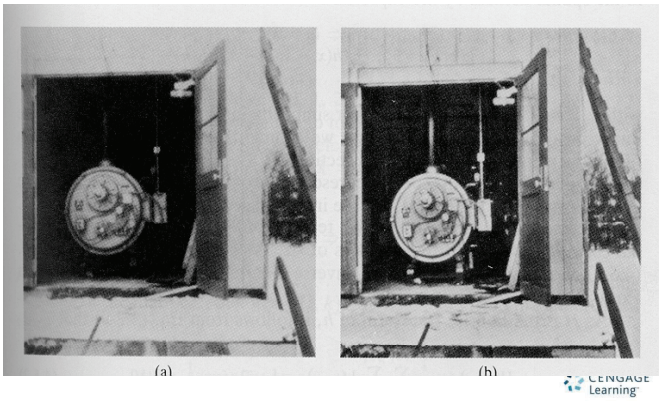



frequency response of homomorphic filter

can be approximated using the back form of HPF $H(u,v) = (\gamma_H - \gamma_L)\left[1 - e^{-c\left[\frac{D^2(u,v)}{D_0^2}\right]}\right] + \gamma_L$



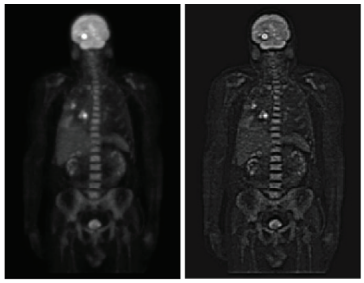
## Homomorphic Filtering



(a) (b)



## Homomorphic Filtering



a b

FIGURE 4.62 (a) Full body PET scan. (b) Image enhanced using homomorphic filtering. (Original image courtesy of Dr. Michael E. Casey, CTI PET Systems.)

- Reduced the effects of the dominant illumination components (hot spots, low freq. component).
- The reflectance components (edge, high freq. component) were sharpened considerably.
- Sharper in hot spots, brain, and skeleton
- More detailed information is shown.



## Homomorphic Filtering in Matlab

```
function res=homfilt(im,cutoff,order,lowgain,highgain)

% HOMFILT(IMAGE,FILTER) applies homomorphic filtering to the image IMAGE
% with the given parameters

u=im2uint8(im);

u(find(u==0))=1;
l=log(double(u));
ft=fftshift(fft2(l));
f=hb_butter(im,cutoff,order,lowgain,highgain);
b=f.*ft;
ib=abs(ifft2(b));
res=exp(ib);
```

FIGURE 7.32 *A function to apply homomorphic filtering.*