

Chapter 5

Image Analysis and Filtering in the Frequency Domain

- The analysis of the previous chapters were made in the spatial domain in which the characteristics of the image and the masks were described in terms of (x,y) locations.
- In this chapter, we investigate images in frequency domain, which offers a powerful means for image understanding, enhancement and especially filtering.
- The ideas of smoothing by **reducing high frequency contents** or **sharpening** by increasing the magnitudes of high frequency components relative to the **low frequency components** come from the concepts directly related to image characterization in terms of its frequency contents.
- In fact the ideas of **filtering** discussed in Chapter 4 is considerably more appealing and intuitive in the frequency domain.

5.1 Periodic Functions and Spatial Frequency

A periodic function such as a sinusoid consists of a pattern or cycle that repeats in both positive and negative directions. The length of this cycle is called the period P , and the reciprocal of the period denote by u is called the frequency. If the variation is in space (rather than for example in time), then P is the distance and u is the spatial frequency.

A periodic variation is also characterized by two other parameters –an amplitude A and a phase ϕ , as shown in Fig. 5.1. A one-dimensional image can be described by a gray level function

$$f(x) = A \sin\left(\frac{2\pi}{P}x + \phi\right) = A \sin(2\pi u x + \phi) \quad (5.1)$$

where $f(x)$ is the gray level, and u is the spatial frequency.

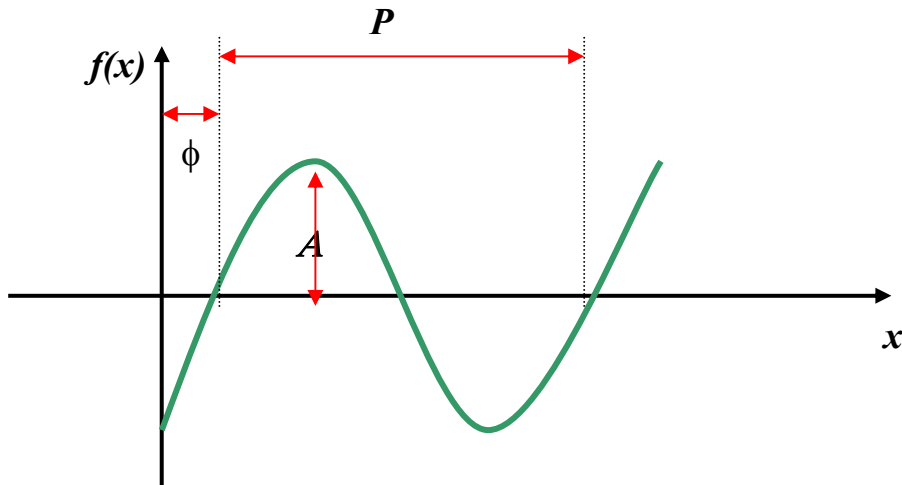


Fig. 5.1 A sinusoidal variation of gray level.

- The sinusoid can be represented by an image, as shown in Fig. 5.2 where Fig. 5.1a shows a sinusoid with frequency of $u = 4$, amplitude of $A = 255$, and phase of $\phi = 0$ on a 512 by 512 image. This means that the gray level values are now from 0 to 64. Note that there is no variation along the y axis

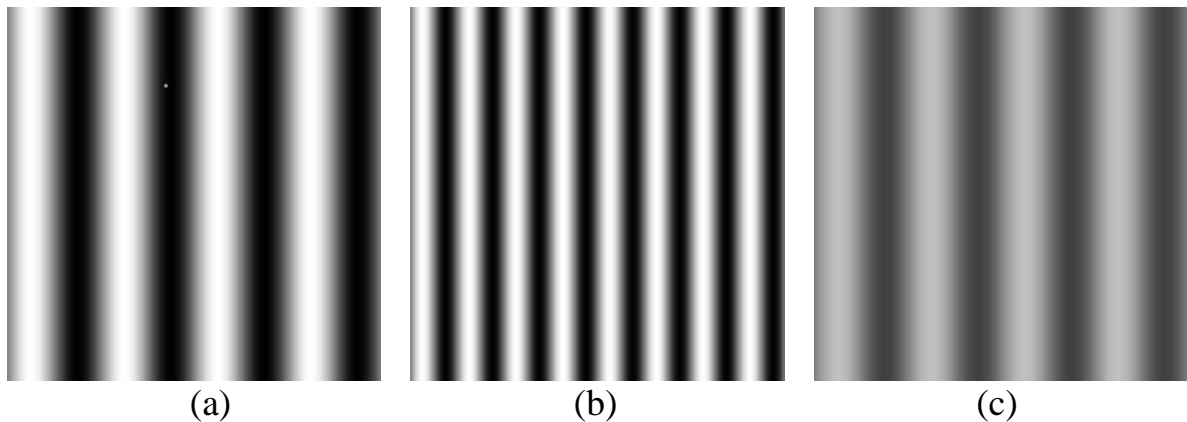
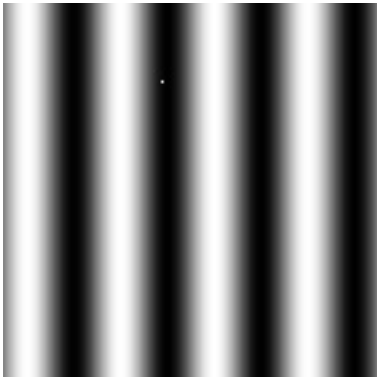
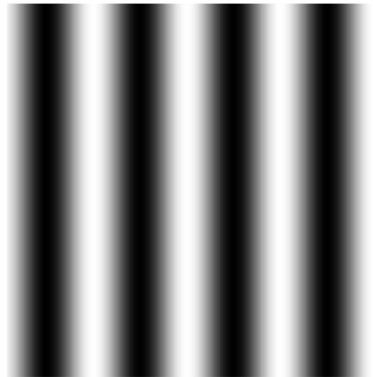


Fig. 5.2 (a) $u = 4, A = 255$, (b) $u = 8, A = 255$ (c) $u = 4, A = 64$.

Fig. 5.3a and Fig 5.3b show the sinusoidal image with phase $\phi = 0$, and $\phi = 90$ degrees, respectively. Fig 5.3c shows a sinusoidal image where the sinusoidal variations are in vertical direction. Finally Fig 5.3 (d) shows an image whose gray level values change along horizontal axis with a frequency of $u = 8$ and along the vertical axis with a frequency of $v = 4$.



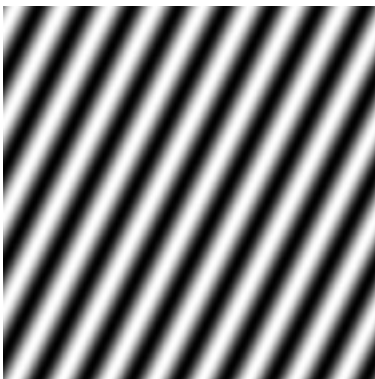
$\phi = 0, u=4, v=0$



$\phi = 90, u = 4, v = 0$



$\phi = 0, u = 0, v = 4$



$\phi = 0, u = 8, v=4$

- Image analysis in frequency domain is based on the fundamental work of Fourier. Fourier showed that any period function can be expressed as the sum of sine and cosine of various amplitudes and frequencies, i.e.

$$f(x) = \sum_{u=0}^{\infty} a_u \cos\left(\frac{2\pi u x}{P}\right) + \sum_{u=0}^{\infty} b_u \sin\left(\frac{2\pi u x}{P}\right) \quad (5.2)$$

where u is the number of cycles fitting into one period. Equation (5.2) is called the Fourier series. For example, consider a square wave with a period of $P=5$ given Fig. 5.2.

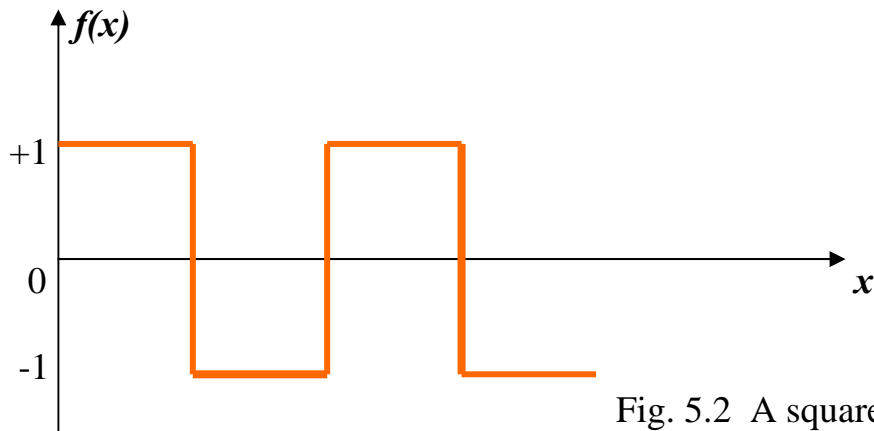
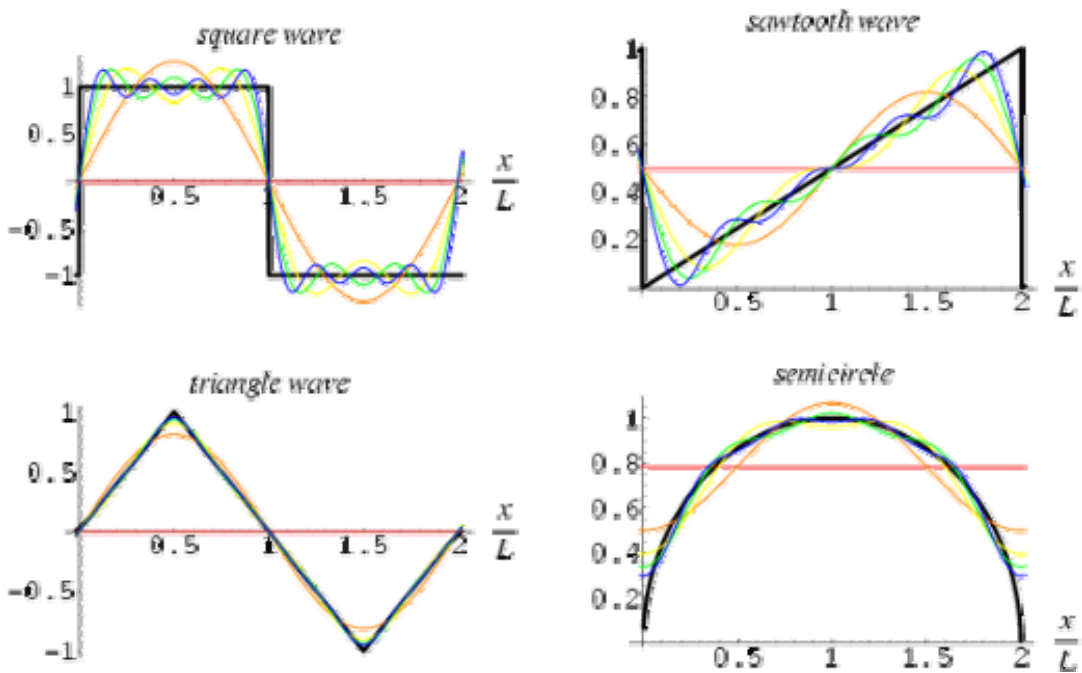


Fig. 5.2 A square function

It can be shown that the square function can be expressed as

$$f(x) = \sum_{u=1}^{\infty} \frac{1}{u} \sin\left(\frac{2\pi u x}{5}\right); \quad i = 1, 3, 5, \dots \quad (5.3)$$

i.e. in (5.2) $a_u = 0$ for all u , and $b_u = \frac{1}{u}$ for odd i , $b_u = 0$ for even u . The mean value of the above function is zero (why?). (verify that (5.3) is in fact a square function by calculating (5.3) i up to 20 and plotting it.



The Fourier series (5.2) can be also expressed for a function of two variables x , and y as

$$f(x, y) = \sum_{u=0}^{\infty} \sum_{v=0}^{\infty} a_{u,v} \cos\left(\frac{2\pi(u x + v y)}{P}\right) + \sum_{u=0}^{\infty} \sum_{v=0}^{\infty} b_{u,v} \sin\left(\frac{2\pi(u x + v y)}{P}\right) \quad (5.4)$$

where u and v are the number of cycles fitting into a horizontal and a vertical period, respectively, and are called frequencies.

- Note that according to the Fourier series any image function $f(x, y)$ can be decomposed (or represented) by a number of sine and cosine images (similar to those in Fig. 5. 2 and 5.3).
- In (5.4) $a_{0,0}$ is the values of $f(x, y)$ when $u = v = 0$ (zero frequencies), that is the mean gray level of the image (why?).
- Higher values of frequencies u and v introduce fluctuations about this mean that are needed to represent gray level values across the image.
- The coefficients $a_{u,v}$ and $b_{u,v}$ determine the relative contributions of each harmonics to the image, and provide useful information on the special frequencies that are present in the image.

5.2 Fourier Transform

Fourier transform provides useful information about the frequency contents of an image, and is used to filter the noise, which is associated with the high frequency components in an image.

Complex Numbers: Since Fourier transform involves complex numbers, we give a brief review. A complex number c has a real part a and an imaginary part b and is expressed as

$$c = a + j b \quad (5.5)$$

where $j = \sqrt{-1}$. The complex number c has a magnitude r and a phase ϕ given by (see Fig 5.4).

$$r = |c| = \sqrt{a^2 + b^2}$$

$$\phi = \tan^{-1}\left(\frac{b}{a}\right) \quad (5.6)$$

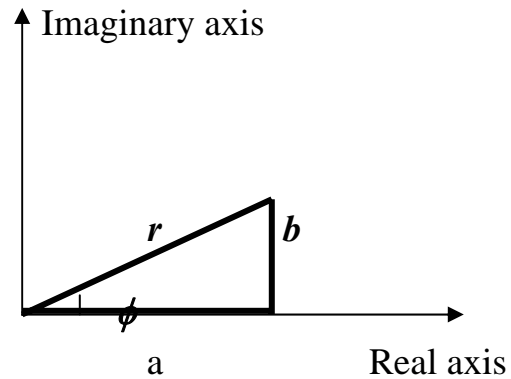


Fig. 5.4 Complex number representation

Another useful representation of a complex number is in the form of an exponential

$$c = r e^{j\phi} \quad (5.7)$$

Using the Euler formula $e^{j\phi} = \cos\phi + j\sin\phi$, we get from (5.6) and (5.7) $a = r \cos\phi$ and $b = r \sin\phi$. Furthermore, Euler formula allows us to express cosine and sine of an angle in terms of itself, i.e. we have (why?)

$$\cos\phi = \frac{1}{2}(e^{j\phi} + e^{-j\phi}) \quad \text{and} \quad \sin\phi = \frac{1}{2j}(e^{j\phi} - e^{-j\phi}) \quad (5.8)$$

- We now consider Fourier transform along x-axis only, i.e. one-dimensional transform. Let $f(x)$ be the gray levels at $x = 0, 1, 2, \dots, n-1$. The Fourier transform is defined as
-

$$F(u) = \frac{1}{n} \sum_{x=0}^{n-1} f(x) e^{-j(2\pi ux)/n}; \quad u = 0, 1, 2, \dots, n \quad (5.9)$$

where u is the frequency along horizontal axis. Note that $F(u)$ is only a function of u (not x).

Example 1: Consider $f(x)$ as shown in Fig. 5.5 below where there are only 4 pixels across x axis and their gray level is 120.

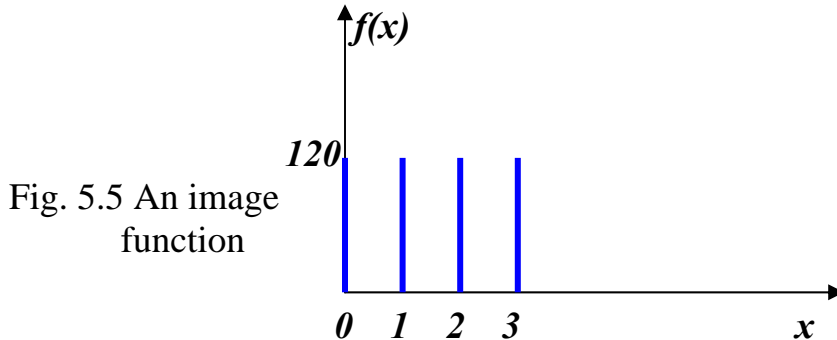


Fig. 5.5 An image function

The Fourier transform of $f(x)$ is computed from (5.9) as

$$F(0) = \frac{1}{4} \sum_{x=0}^3 f(x) e^{-j(2\pi 0x)/4} = \frac{1}{4} \sum_{x=0}^3 f(x) e^0 = 120$$

$$F(1) = \frac{1}{4} \sum_{x=0}^3 f(x) e^{-j(2\pi x)/4} = \frac{1}{4} \sum_{x=0}^3 f(x) e^{-j\pi x/2}$$

$$= \frac{1}{4} \left(120e^0 + 120e^{-j\pi/2} + 120e^{-j\pi} + 120e^{-j3\pi/2} \right) = 0$$

(Using Euler formula verify that the last expression is in fact zero). Similarly it can be shown that $F(2) = F(3) = 0$ (why?). The Fourier transform is shown in Fig. 5.5b, and it is seen that it has only one component, which is at the zero frequency. This is meaningful since the gray level values are constant across the image (Fig. 5.5), and there is no variation. This result can be generalized to one-dimensional images having any number of pixels (in general n), e.g. if $f(x) = l$, for $x = 0, 1, 2, \dots, n$ then $F(u) = l$ for $u = 0$, and $F(u) = 0$ for $u = 1, 2, \dots, n-1$ (why?).

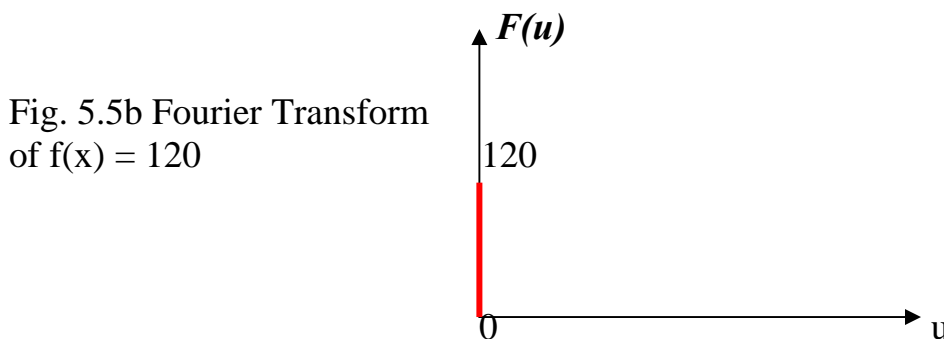


Fig. 5.5b Fourier Transform of $f(x) = 120$

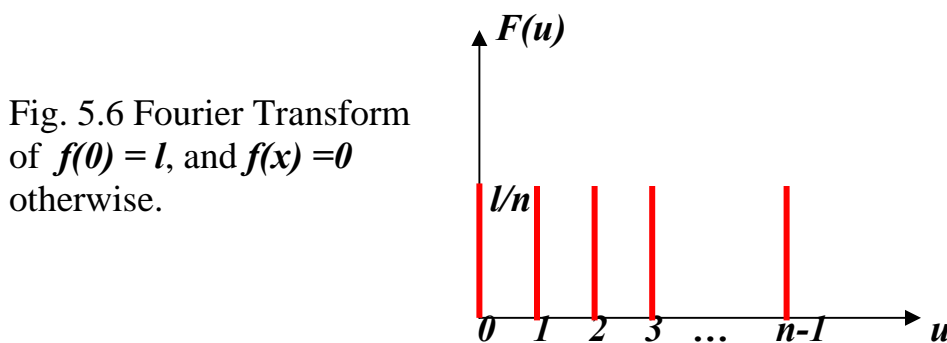
Example 2: Suppose that the image has one bright pixel with gray level l at $x = 0$, and all other pixels are black, i.e.

$$f(x) = \begin{cases} l & \text{for } x = 0 \\ 0 & \text{for } x = 1, 2, \dots, n-1 \end{cases}$$

The Fourier transform of the above image is obtained from (5.9) as

$$F(u) = \frac{1}{n} \sum_{x=0}^{n-1} f(x) e^{-j(2\pi ux)/n} = \frac{1}{n} f(0) e^{-j(2\pi u0)/n} = \frac{l}{n} \quad \text{for } u = 0, 1, \dots, n-1$$

The Fourier transform is plotted in Fig 5.6, and it has all frequency components (what is the explanation for this behavior?).



In general Fourier transform has both real and imaginary parts as the following example demonstrates.

Example 3: Consider the image function $f(x) = 100$ for $x = 0, 1$ and $f(x) = 0$ for $x = 2, 3$. The Fourier transform is computed from (5.9), which after simplification gives (why?)

$$F(0) = 50, F(1) = 25 - j25, F(2) = 0 \text{ and } F(3) = 25 + 25j.$$

The magnitude is computed from (5.6) as

$$|F(0)| = 50, |F(1)| = 35.5, |F(2)| = 0 \text{ and } |F(3)| = 35.5$$

The phase can also be computed using (5.6).

Note: Slowly changing gray levels tend to have low frequencies, whereas abruptly changing gray levels have high components.

- If the Fourier transform $F(u)$ is given and the image $f(x)$ is to be found, then we use the **inverse Fourier transform** given by

$$f(x) = \sum_{u=0}^{n-1} F(u) e^{j(2\pi ux)/n} ; \quad x=0, 1, 2, \dots, n-1 \quad (5.10)$$

Note the similarity between (5.9) and (5.10).

Example 4: Let $F(0) = 120$, and $F(u) = 0$ for $u = 1, 2, \dots, n-1$. The inverse Fourier transform is

$$f(0) = \sum_{u=0}^3 F(u) e^{j(2\pi u \cdot 0)/4} = \sum_{u=0}^3 F(u) e^0 = 120$$

Similarly, the other components are found to be (why?) $f(1) = f(2) = f(3) = 120$ which are expected (why?).

- In general image are **two dimensional** $f(x,y)$. In this case the Fourier transform is

$$F(u,v) = \frac{1}{n} \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} f(x,y) e^{-j2\pi(ux+vy)/n} \quad (5.11)$$

where v is the frequency along vertical axis. In case the image is not square, n must be replaced by height h and width w .

The **inverse Fourier transform for the two dimension image** is

$$f(x,y) = \frac{1}{n} \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} F(u,v) e^{j2\pi(ux+vy)/n} \quad (5.12)$$

Spectra of an Image: Since $F(u,v)$ is a complex quantity, we can write it as

$$F(u,v) = R(u,v) + j I(u,v) \quad (5.13)$$

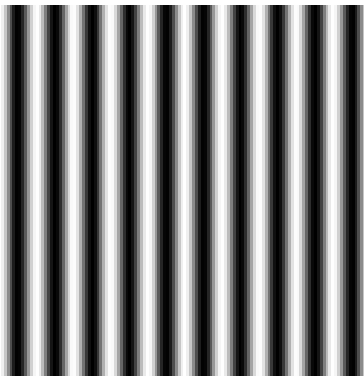
Where $R(u, v)$ and $I(u, v)$ are the real and imaginary parts of $F(u, v)$. The following two quantities are important:

Amplitude spectrum: $|F(u, v)| = \sqrt{R^2(u, v) + I^2(u, v)}$.

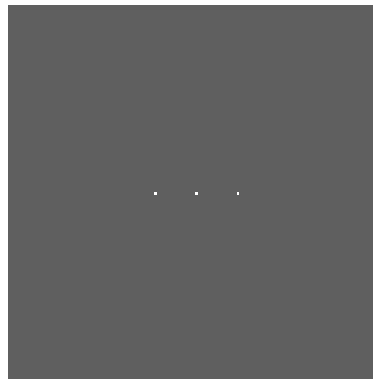
Phase spectrum: $\phi = \tan^{-1}\left(\frac{I(u, v)}{R(u, v)}\right)$.

Power spectrum : $P(u, v) = |F(u, v)|^2 = R^2(u, v) + I^2(u, v)$

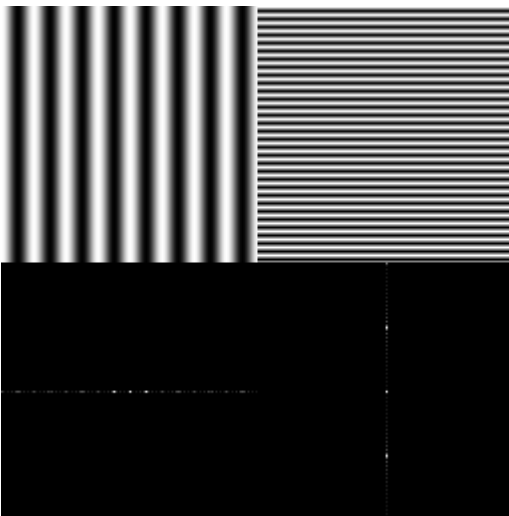
The phase spectrum is less significant than the amplitude spectrum for image analysis.



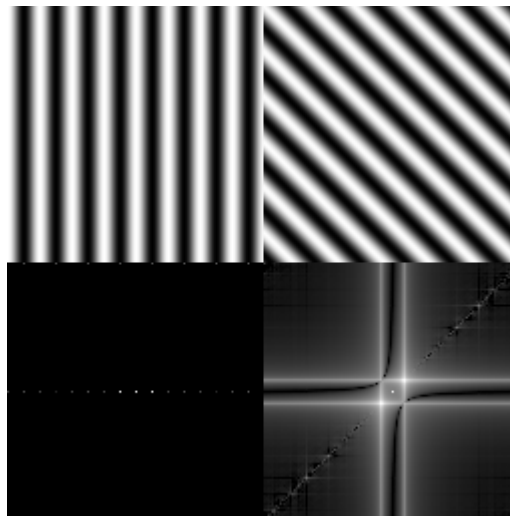
Strips $f(x,y)$



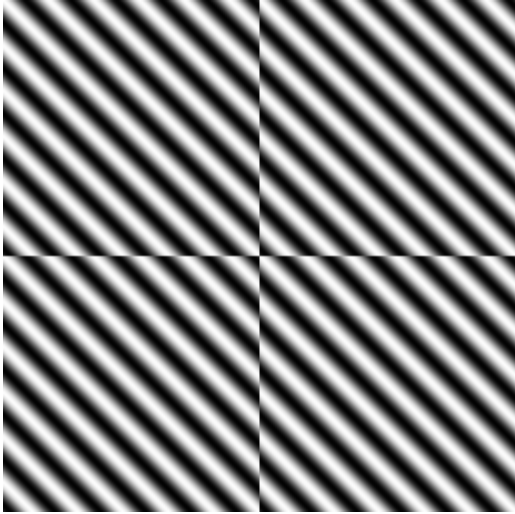
Strips $|F(u,v)|$ (magnitude)



**Top - Strips $f(x,y)$
Bottom Strips $|F(u,v)|$**



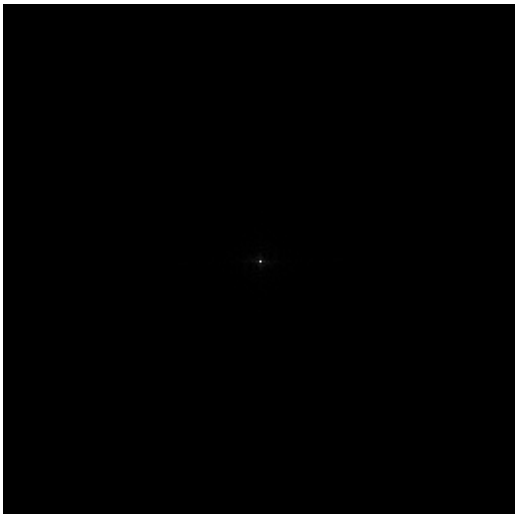
**Top - Strips $f(x,y)$
Bottom Strips $|F(u,v)|$**



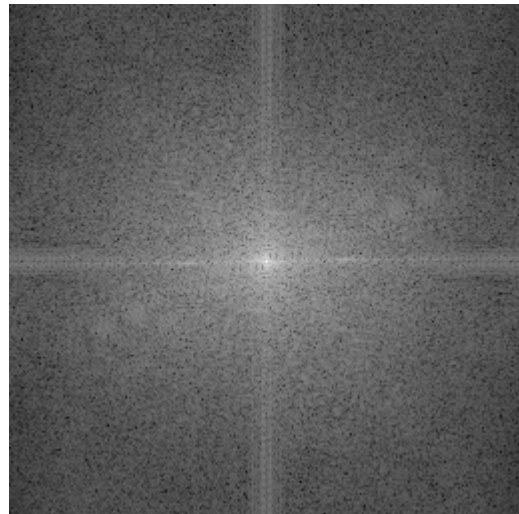
Strip as seen by FT



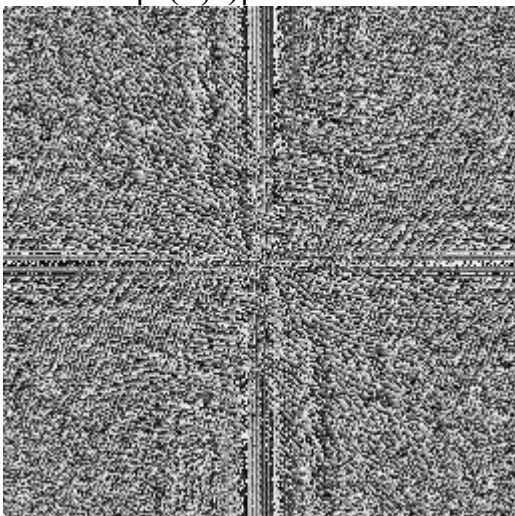
Clown $f(x,y)$



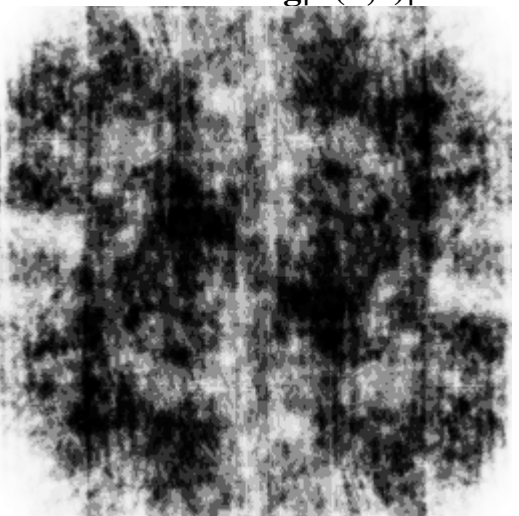
Clown $|F(u,v)|$



Clown $\log|F(u,v)|$



Clown $\phi(u,v)$



Clown $\text{Inv } |F(u,v)|$ (without phase).

5.3 Properties of Fourier Transform

Here we mention several useful properties of the Fourier transform:

Separability: A two-dimensional (2-D) Fourier transform can be computed as two 1-D transforms since (5.11) can be written as (why?)

$$F(\mathbf{u}, \mathbf{v}) = \frac{1}{n} \sum_{x=0}^{n-1} e^{-j2\pi ux/n} \sum_{y=0}^{n-1} f(x, y) e^{-j2\pi vy/n} \quad (5.14)$$

where the second summation is in fact a 1-D transform on $f(x, y)$ along the vertical axis. We denote the second summation by $F(x, \mathbf{v})$, (why is the second summation a function of \mathbf{x} and \mathbf{v} only?) and write (5.14) as

$$F(\mathbf{u}, \mathbf{v}) = \frac{1}{n} \sum_{x=0}^{n-1} e^{-j2\pi ux/n} F(x, \mathbf{v}) \quad (5.15)$$

Note that (5.15) is another 1-D transform. Therefore instead of writing a program to compute a 2-D Fourier transform, we write a simpler program for a 1-D transform and run it twice with different arguments (why is it better to do two 1-D transforms instead of one 2-D transform?). The same principles of separability applies to the inverse Fourier transform (why?).

(b) Periodicity and Symmetry: the Fourier transform and its inverse are periodic functions with the period n since

$$F(\mathbf{u}, \mathbf{v}) = F(\mathbf{u}+n, \mathbf{v}) = F(\mathbf{u}, \mathbf{v}+n) = F(\mathbf{u}+n, \mathbf{v}+n) \quad (5.16)$$

(Verify the above by replacing \mathbf{u} with $\mathbf{u}+n$ in (5.11), and similarly with $\mathbf{v}+n$, etc.) This property implies that only one period (0 to $n-1$) is needed to completely specify $F(\mathbf{u}, \mathbf{v})$ (why?). Furthermore, $F(\mathbf{u}, \mathbf{v})$ is symmetric since

$$|F(\mathbf{u}, \mathbf{v})| = |F(-\mathbf{u}, -\mathbf{v})| \quad (5.17)$$

(why?) To see the implications of periodicity and symmetry, for simplicity consider the case of 1-D transform in which (5.16) and (5.17) simplify to

$$F(u) = F(u+n) \quad \text{and} \quad |F(u)| = |F(-u)| \quad (5.18)$$

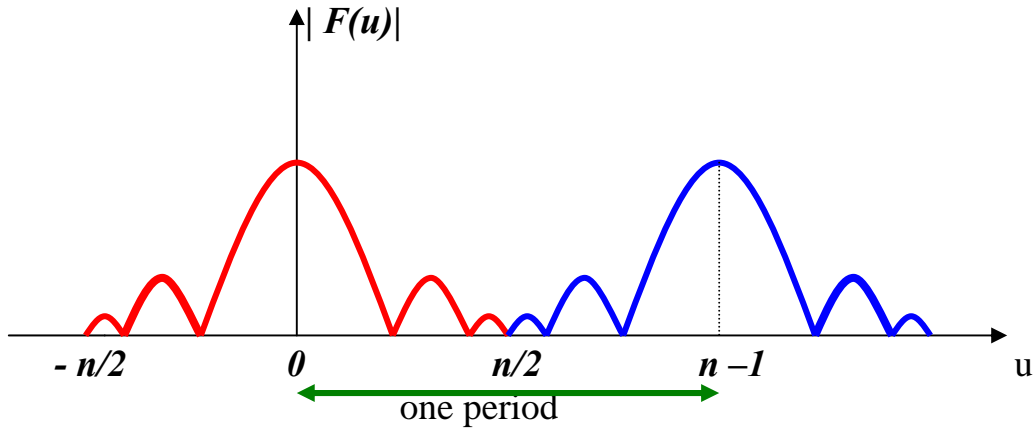


Fig. 5.7 Periodic and symmetric nature of the Fourier transform

Since Fourier transform is formulated in the interval 0 to $n-1$, the result produces to “back to back” half periods. To display a full period with the maximum value at the center (rather than at either side of the display frame), we must move the origin to $u = n/2$ as shown in Fig. 5.8.

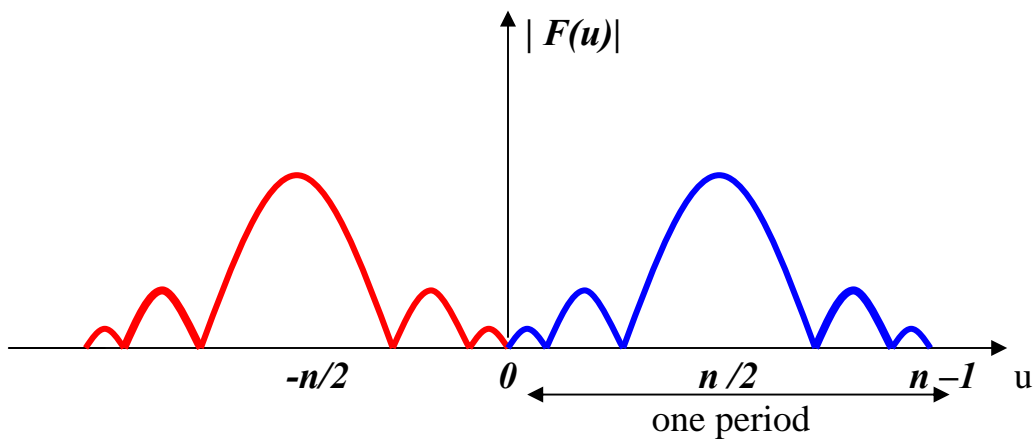


Fig. 5.7 The original is shifted to $u = n/2$.

Fig. 5.8 show the concept of periodicity and symmetry in 2-D as an image. Note that when the origin is unchanged the Fourier transform image appears in four corners of the $u-v$ display plane, as shown by a solid line square. In order to bring the Fourier transform image to the center of the display, $F(u,v)$ must be shifted to $u = n/2$ and $v = n/2$, as shown by the dotted square.

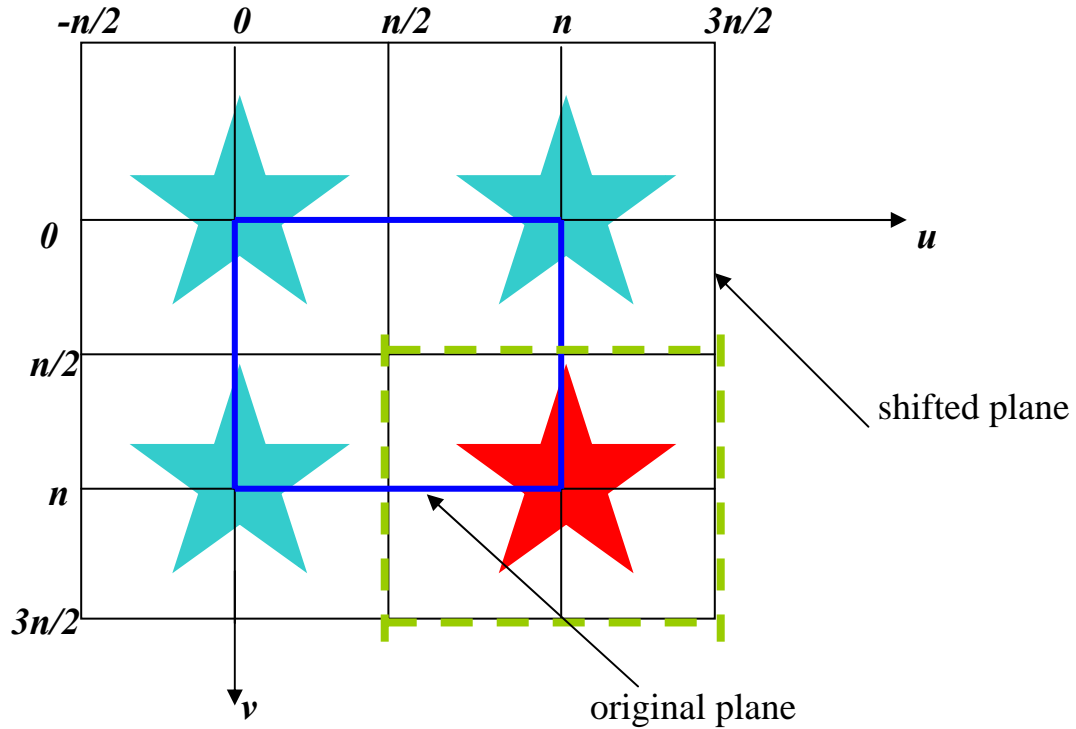


Fig. 5.8 Shifting of the origin to $u = v = n/2$

To implement the shift, we must determine the value of $F(u - n/2, v - n/2)$ using (5.11) as follows (verify this):

$$\begin{aligned}
 F(u - n/2, v - n/2) &= \frac{1}{n} \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} f(x, y) e^{-j2\pi((u-n/2)x + (v-n/2)y)/n} \\
 &= \frac{1}{n} \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} f(x, y) \cdot (-1)^{x+y} e^{-j2\pi(ux + vy)/n}
 \end{aligned}$$

The above equation implies that in order to shift the origin of the $\mathbf{F}(\mathbf{u}, \mathbf{v})$, we must first multiply the image function $f(\mathbf{x}, \mathbf{y})$ by $(-1)^{x+y}$ and then perform the Fourier transform. This simply means that when $(\mathbf{x} + \mathbf{y})$ is odd, we change $f(\mathbf{x}, \mathbf{y})$ to $-f(\mathbf{x}, \mathbf{y})$, otherwise we leave it unchanged.

5.4 The Fast Fourier Transform (FFT)

The computation of a 2-D Fourier transform, require two 1-D transforms. A

1-D transform involves determining $F(u) = \frac{1}{n} \sum_{x=0}^{n-1} f(x)e^{-j(2\pi ux)/n}$ for each

u . This require n complex number multiplication of the exponential by $f(x)$ plus $n-1$ additions for each u , with the total complexity of $O(n^2)$, (why?).

Note that $e^{-j(2\pi ux)/n}$ can be computed once and be kept in an array (what will be the size of this array considering that there are n values for u and x ?)

Using FFT, the number of multiplication and additions will reduce to

$O(n \lg n)$. This is a considerable saving, e.g for $n = 512$, $n^2 = 262144$

whereas $n \lg n = 4608$, and for $n = 1024$, $n^2 = 1048576$ versus

$n \lg n = 10,240$.

- To perform an FFT, we require n to be a power of 2 and the image to be square. In case the image is not square, we pad it with zeros (black) to get the required size. A 1-D FFT of length n is performed by two FFTs of length $n/2$ each, as follows

$$\begin{aligned} nF(u) &= \sum_{x=0}^{n-1} f(x)e^{-j(2\pi ux)/n} \\ &= \sum_{x=0}^{n/2-1} f(2x)e^{-j2\pi u(2x)/n} + \sum_{x=0}^{n/2-1} f(2x+1)e^{-j2\pi u(2x+1)/n} \end{aligned} \quad (5.19)$$

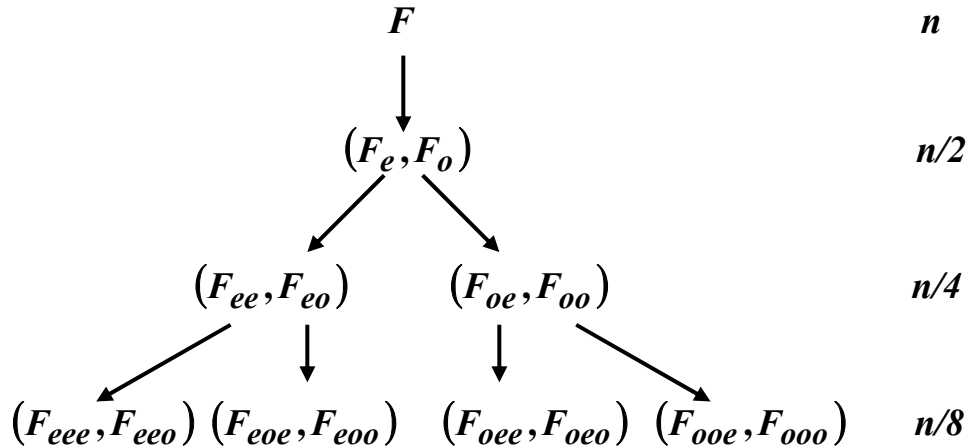
(Why? - verify the above for $n=4$). Furthermore, we can write (5.19) as

$$nF(u) = \sum_{x=0}^{n/2-1} f(2x)e^{-j2\pi ux/(n/2)} + Q^u \sum_{x=0}^{n/2-1} f(2x+1)e^{-j2\pi ux/(n/2)}$$

where $Q = e^{-j2\pi/n}$ is a constant complex number (verify the above). The above equation shows that the FFT contains two parts – one uses only even x and the other uses odd x . In other words

$$nF(u) = F_e(u) + Q^u F_o(u) \quad (5.20)$$

Furthermore, FFT can be computed recursively, as follows.



For example if $n = 16$, then

F_e uses $f(0), f(2), f(4), f(6), f(8), f(10), f(12), f(14)$

F_o uses $f(1), f(3), f(5), f(7), f(9), f(11), f(13), f(15)$

F_{ee} uses $f(0), f(4), f(8), f(12)$

F_{eo} uses $f(2), f(6), f(10), f(14)$

F_{eee} uses $f(0), f(8)$

F_{eeo} uses $f(4), f(12), \text{etc.}$

If n is a power of 2, then due to the recursive nature of the FFT algorithm its complexity becomes $O(n \lg n)$, (why?).

Bit Reversal : In order to determine which image data $f(x)$ is to be used for F_e, F_o, F_{ee}, F_{eo} , etc., we perform bit reversal, as the following table demonstrates

array index	binary representation	bit reversal	array index
0	0000	0000	0
1	0001	1000	8
2	0010	0100	4
3	0011	1100	12
4	0100	0010	2
5	0101	1010	10
6	0110	0110	6
7	0111	1110	14
⋮	⋮	⋮	⋮
15	1111	1111	15

Note that the bit reversal allows combining correct sequences for performing FFT. Thus before carrying out the FFT, the image function $f(x)$ must be reordered so that the resulting FFT is correct.

Note: Fourier transform and its inverse are different only in the sign of exponential, and therefore the same method can be used to perform both forward and inverse transform by supplying an argument -1 for the former and $+1$ for the latter.

Fourier Array: The FFT of $f(x)$, $x=0,1, \dots, n-1$ is stored in a 1-D array of size $2n$ since each value of $F(u)$ has a real and an imaginary part. The real and imaginary parts are stored in consecutive array locations, i.e. Real ($F(u)$) is stored in `FourierArray[2u]`, and Imaginary($F(u)$) is stored in `FourierArray[2u+1]`, $u = 0, 1, \dots, n-1$. Although the image function $f(x)$ is real, we allocate an `ImageArray` of size $2n$, and fill its alternate locations with $f(x)$. The odd number locations are set to zero. This allows the same function to be used for both forward and inverse transforms. Note 2-D transform is in fact two 1-D transforms as noted previously.

Windowing Effect: Fourier transform $F(u,v)$ is periodic, and implicit in this periodicity is that the image $f(x,y)$ is also periodic. Therefore when an image array of size n by n is presented to the Fourier transform, it assumes that $f(x,y)$ repeats its self as shown in Fig. 5.9



Fig. 5.9 Spatial discontinuities at borders due to periodic assumption

As seen there are discontinuities at the borders, and the Fourier transform sees it as a sudden change in the image and accommodates it through distortion of the spectrum. This effect is called windowing since FFT is presented with a window, and not as it expects with a plane extending to infinity in all directions.

The windowing effect is reduced by smoothing the image at edges. A number of smoothing functions have been presented. Barlett proposes that the image $f(x,y)$ be multiplied by the following function before taking the FFT.

$$W(r) = \begin{cases} 1 - (r/r_{\max}) & r \leq r_{\max} \\ 0 & r > r_{\max} \end{cases} \quad (5.21)$$

where r is the distance from the center of the image, and r_{\max} is its maximum value. This function has a cone shape. The Hanning window is defined by

$$W(r) = 0.5 - 0.5 \cos\left(\pi\left(1 - \frac{r}{r_{\max}}\right)\right) \quad (5.22)$$

Hanning window is somewhat smoother than Barlett window.

5.5 Java Implementation of FFT

- FFT is not supported by standard Java 2D.
- The `com.pearsoneduc.ip.op` package provides a single class called `ImageFFT`. Instances of this class can compute forward and inverse Fourier transform using FFT, and manage the handling of complex data.
- However, `ImageFFT` class can not be implemented as `BufferedImageOp`, instead it provides a set of methods that can be used to implement operations in the frequency domain as `BufferedImageOp` classes.
- Some of the methods of the `ImageFFT` class are as follows:

`ImageFFT(BufferedImage img)` - Creates an `ImageFFT` of image
`ImageFFT(BufferedImage img, int win)` - Creates an `ImageFFT` of image with win defined by one the following parameters :

Name	Value of win
<code>NO_WINDOW</code>	1
<code>BARLETT_WINDOW</code>	2
<code>HANNIG_WINDOW</code>	3

`BufferedImage getSpectrum()` - Returns the amplitude spectrum in an image form, shifted such that a zero frequency is at the center and scaled logarithmically

`BufferedImage getPhase(int u, int v)` - Returns the phase of Fourier transform at specified coordinates.

`Float getMagnitude(int u, int v)` - returns the magnitude of the Fourier transform at the specified frequencies.

The code below computes the spectrum using Hanning window and returns the spectrum as another image.

```
public static BufferedImage computeSpectrum (BufferedImage image)
throws FFTException {
    ImageFFT fft = new ImageFFT(image, ImageFFT.HANNIG_WINDOW);
    fft.transform();
    return fft.getSpectrum(); }
```

The above assumes that the supplied image is an 8-bit gray scale and throw an exception if this is not the case. When `ImageFFT` is created, the following happens: (a) Image data are copied into the internal storage of the `ImageFFT` object. The data is then padded with zeros if the dimensions are not power of two (why?), and finally the data are windowed. Note that a Fourier transform must always be called when using `ImageFFT`.

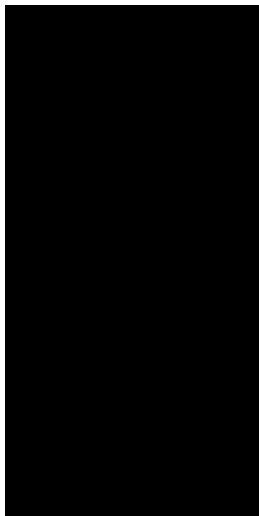
5.6 Spectra Display

The spectrum amplitude $|F(u,v)|$ is generally displayed as an 8-bit gray level image. Therefore a scaling is usually done to bring the value of $|F(u,v)|$ to the range of 0 to 255. Natural (non-synthetic) images mostly contain low frequency components which are concentrated around $u = v = 0$ in the center of the spectrum image with few higher frequency components. This results in a situation where most of the display window is black with heavy concentration of light pixels are in the center. In order to better view this high activity region, a logarithmic mapping of the following form is applied

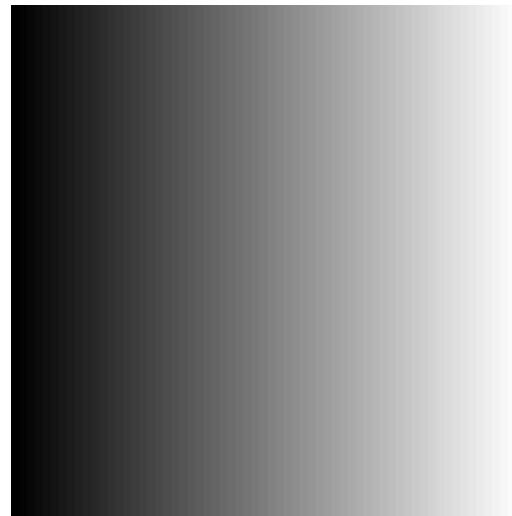
$$|F_{lg}(u,v)| = C \log(|F(u,v)| + 1) \quad (5.23)$$

where the constant C is chosen so that the results is an 8-bit value. The $+1$ is for preventing the case of undefined logarithm when $|F(u,v)|=0$.

Let us now examine a few synthetic and natural images. Figure 5.10 shows step type images, one with 2-step (half black, half white), and the other a 64-step image. The amplitude spectrum of the each image is shown below it. Note that because of abrupt change along x direction of $f(x,y)$ in Fig. 5.10a, the spectrum 5.10c contains a lot of both low and high frequency along the u axis in $|F(u,v)|$ plane. The image 5.10b has a slow variations in gray levels along the u axis, and therefore its spectrum has lower amplitudes (seen as low gray levels) along the u axis in Fig. 5.10d. However, because of the periodic nature of the 64-step image, we now observe some relatively strong components at several locations along the u axis in Fig. 5.10d.



(a)



(b)

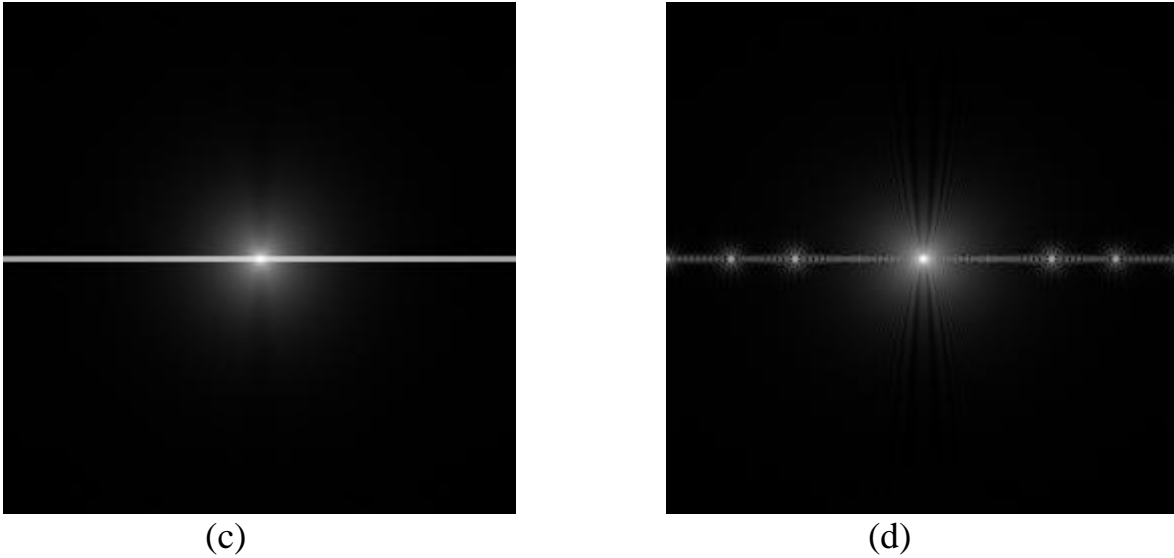


Fig. 5.10 (a) Binary 2-step image, (b) gray level 64-step image, (c) Spectrum of (a), and (d) spectrum of (b).

5.7 Frequency-Domain Filtering

- In the previous chapter we discussed several filtering techniques in the spatial domain to reduce noise and improve the quality of the image.
- Filtering in frequency domain is more effective for many types of noise where spatial domain filtering cannot achieve good results, such as Gaussian noise.
- More importantly, noise is best described as high a frequency signal and therefore the frequency domain techniques are more appropriate to describe and filter the noise.
- Finally, convolution in spatial domain is computationally more expensive than filtering the in frequency domain (why?).
- Filtering can be expressed as the pixel by pixel multiplication of the spectrum $\mathbf{F}(\mathbf{u},\mathbf{v})$ by the filter (mask) transfer function $\mathbf{M}(\mathbf{v},\mathbf{v})$

$$\mathbf{G}(\mathbf{u},\mathbf{v}) = \mathbf{F}(\mathbf{u},\mathbf{v}) \mathbf{M}(\mathbf{u},\mathbf{v}) \quad (5.24)$$

where $\mathbf{G}(\mathbf{u},\mathbf{v})$ is the output image.

- The filter transfer function $\mathbf{M}(\mathbf{u},\mathbf{v})$ must be specified to achieve a particular filtering objective, as we shall see.

- Most filters are **zero-phase-shifters**, meaning that they do not change the phase of the image.
- A **fundamental relation** exists between filtering in the spatial and frequency domains, which can be expressed as follows:

$$\begin{aligned} \text{Fourier}(f(x,y) \otimes m(x,y)) &= F(u,v) M(u,v) \\ \text{InvFourier}(F(u,v) M(u,v)) &= f(x,y) \otimes m(x,y) \end{aligned} \quad (5.25)$$

- This relation which we have quoted without proof, states that if the image $f(x,y)$ is convolved with the mask (filter) $M(x,y)$ and then Fourier transform is taken, the result will be equal to multiplication of the Fourier transform of the image by the Fourier transform of the filter. Similarly the inverse Fourier transform of the product is equal to the convolution. As a result, everything that can be achieved in the spatial domain, can be obtained in frequency domain, and vice versa.
- It turns out, however, that performing the operations in the frequency domain is computationally more efficient (require less operations) than in the spatial domain, when the dimension of the filter is high (e.g. higher than 5 by 5).
- However, the main reason for working in the frequency domain is that in many cases much better filtering can be achieved (why?)

5.7.1 Low Pass Filtering

A natural image contains many frequency components, i.e. from low to high values of u and v . However, the low frequency components generally have higher magnitudes, and many high frequency components are usually absent in a natural image.

- To demonstrate this fact consider the **300×450 image** of Quan given Fig. 5.11a. This image **is now padded with zeros to make it a 512×512 = 266144 pixel image** (why?). Suppose now that we perform Fourier transform on this padded image, and only retain components that are approximately in a circle of **125 pixel radius** around the origin in the u - v plane. We then set the remaining components approximately to zero (this is called a Butterworth low pass filter to be discussed shortly).

- This way we are retaining only $(125 \times 125 \times 3.14) / (512 \times 512) = 18.7\%$ of the frequency information. Fig. 11b shows the image after the inverse transformation. Despite the fact that more than 80% of the frequency domain information is removed, we see little degradation in the quality of the image.



(a)



(b)

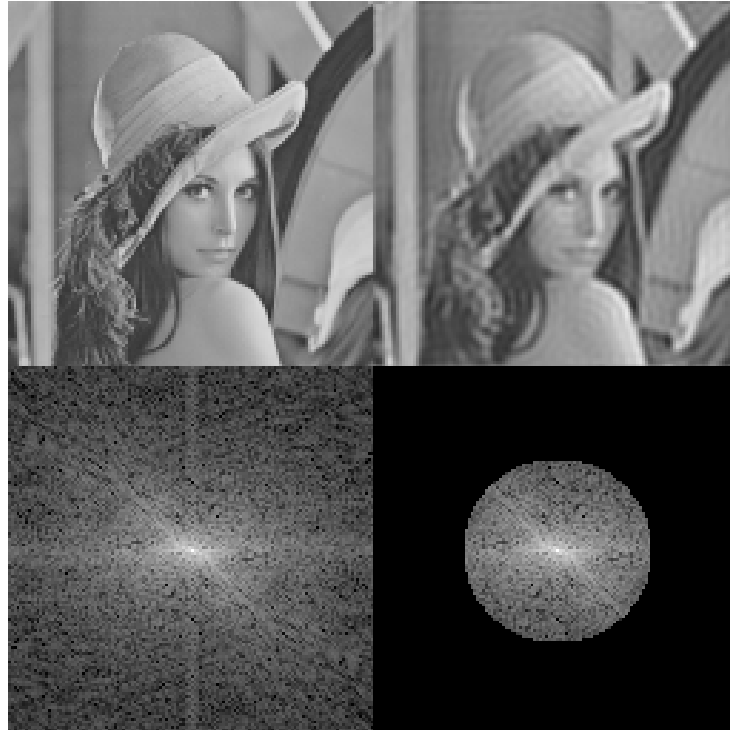
Fig. 11 (a) Original 300x450 pixel image, (b) 512x512 image after removing more than 80% of frequency components.

Noise is usually reflected in an image as high frequency components, and thus the purpose of a low pass filter is to remove such frequencies to clean the image. An ideal low pass filter has the following characteristics

$$M_{lp}(u,v) = \begin{cases} 1, & \text{for } r \leq r_0 \\ 0, & \text{for } r > r_0 \end{cases} \quad (5.26)$$

where $r = \sqrt{u^2 + v^2}$ and is r_0 constant. Equation (5.26) describes a circle in the $u-v$ plan with a radius of r_0 . When the above filter $M(u,v)$ is multiplied by the image Fourier transform $F(u,v)$, the frequency components inside the circle, which correspond to low frequencies, remain unchanged in the output image, and the component outside the circle are removed. The frequency r_0

is called the cut off frequency, and is specified by the user, e.g. in Fig 11, $r_0 = 125$.



- An ideal low pass filter blurs the image as result of removing higher frequencies. It has also another side effect and that is a phenomenon called ringing. The ringing effect is due to sharp and sudden change in the filter characteristics. We know from (5.25) that multiplication in the frequency domain is equivalent to convolution in spatial domain. Now the spatial domain characteristics of an ideal low pass filter is of the form $\frac{\sin r}{r}$ which is the equation of a wave (ripple). When this wave (called the sinc function) is convolved with the image, it produces the so called ringing effect. In order to avoid ringing, we can use a low pass filter with a transfer function that falls smoothly to zero. Butterworth has proposed the following filter transfer function for this purpose

$$M_{lp}(u,v) = \frac{1}{1 + (r/r_0)^{2p}} \quad (5.27)$$

where p is the order of the filter, e.g. $p=1$ is Butterworth filter of order 1. As the order increases this filter approaches an ideal filter, as shown in Fig. 5.12.

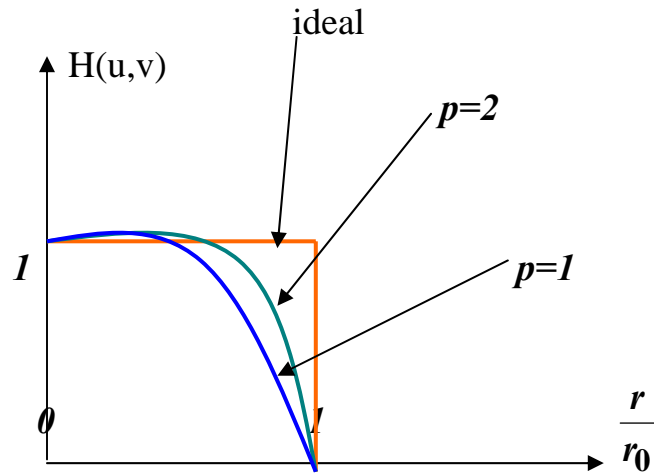
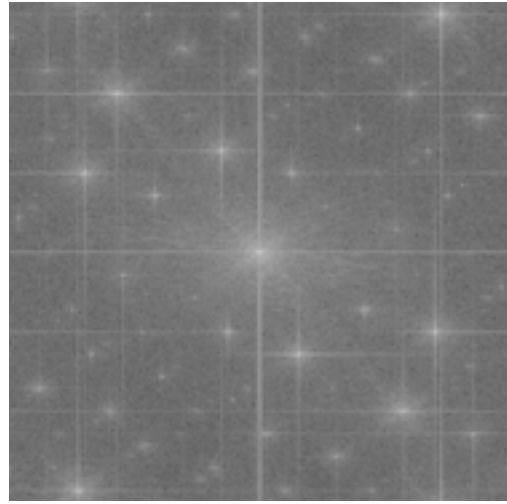


Fig. 5.12 Characteristics of low pass filters.

Fig. 5.13a shows Blonde corrupted by sinusoidal noise, where the Fourier spectrum is seen in Fig. 5.13b and shows the high frequency contents of the noise. Fig. 5.13c shows the image after applying a Butterworth low pass filter of order one with a normalized radius of 0.3. Normalized radius is defined as $r_0 / \max(u, v)$. Finally, Fig. 5.13d the Fourier spectrum of the filtered image. As seen, the high frequency components are removed.



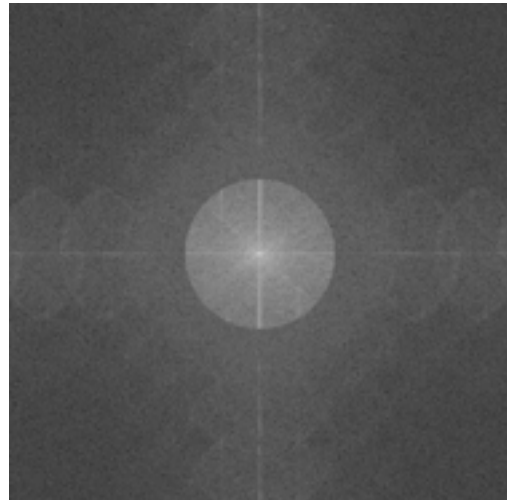
(a)



(b)



(c)



(d)

Fig. 5.13 (a) and (b) Sinusoidal noise corrupted image and its spectrum, (c) and (d) filtered image and its spectrum.

Note that a mean or median filter will not be effective, as the former just averages the noise and the signal (clean part of image), and the latter can pick many noise pixels. Fig. 5.14 shows that neither of these spatial domain filters is effective in reducing the noise which has spread through the fabric of the image.

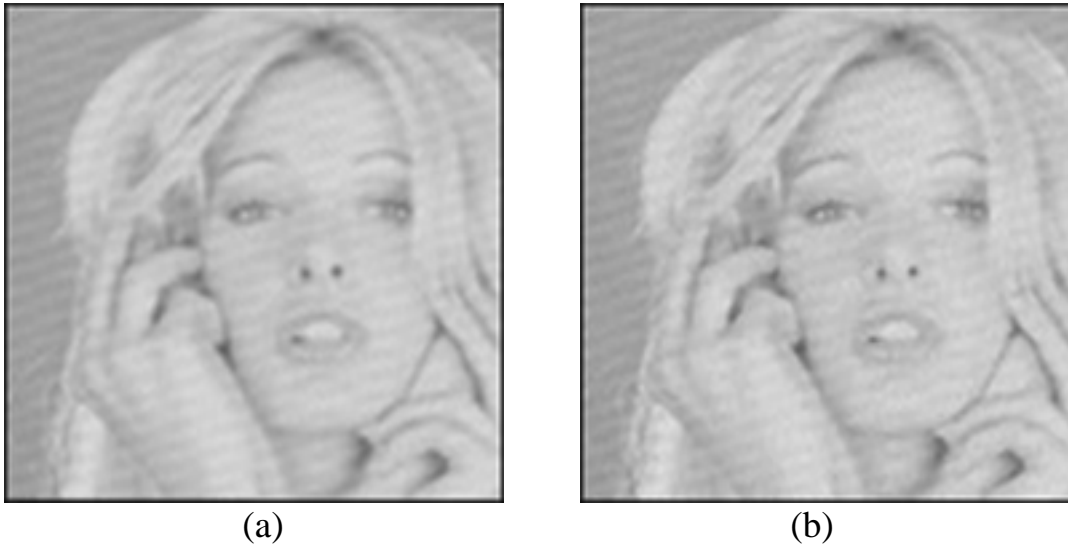
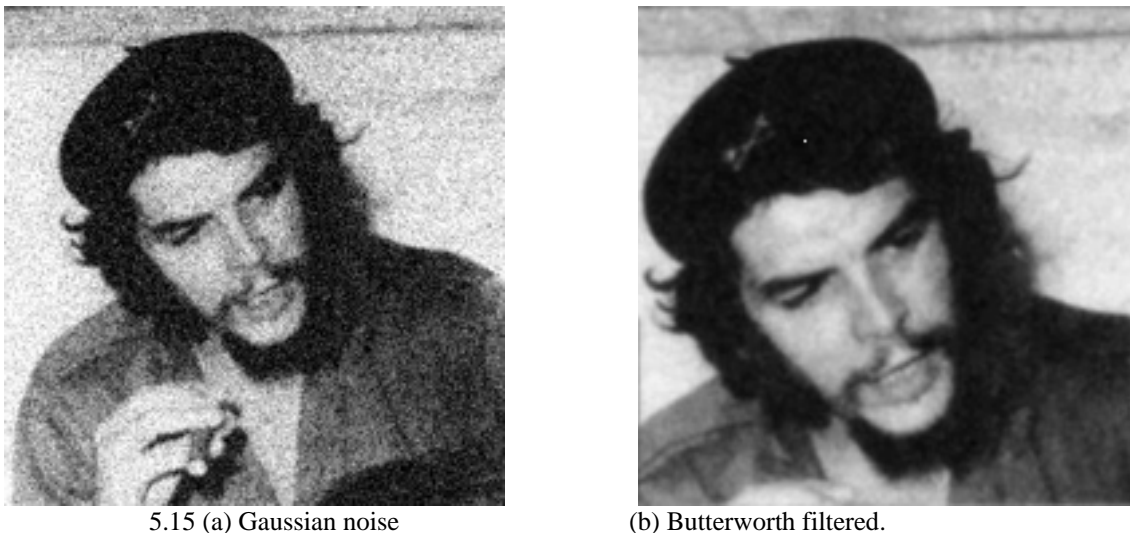


Fig. 5.14 (a) The result of mean filtering of the sinusoidal noise corrupted image of Fig. 13a, (b) median filtering of the image in Fig. 5.13a.

Finally, Fig. 5.15 shows Butterworth low pass filtering of an image corrupted by the Gaussian noise. The image of Che is taken many years ago with a low quality camera and under adverse environmental conditions (possibly cloudy and rainy weather). Note that the filtered image is zoomed in to show the details.



5.7.2 High Pass Filtering

A high pass filter blocks low frequency components, and retains high frequency contents of an image. The ideal high pass filter is

$$M_{hp}(u,v) = \begin{cases} 1, & \text{for } r > r_0 \\ 0, & \text{for } r \leq r_0 \end{cases} \quad (5.28)$$

where $r = \sqrt{u^2 + v^2}$ and is r_0 the cut-off frequency as before. As with the ideal low pass filter, this filter also gives rise to ringing, which can be avoided by the use of a Butterworth filter with the filter characteristics

$$M_{hp}(u,v) = \frac{1}{1 + (r_0 / r)^{2p}} \quad (5.29)$$

The characteristics of this filter is plotted in Fig. 5.16.

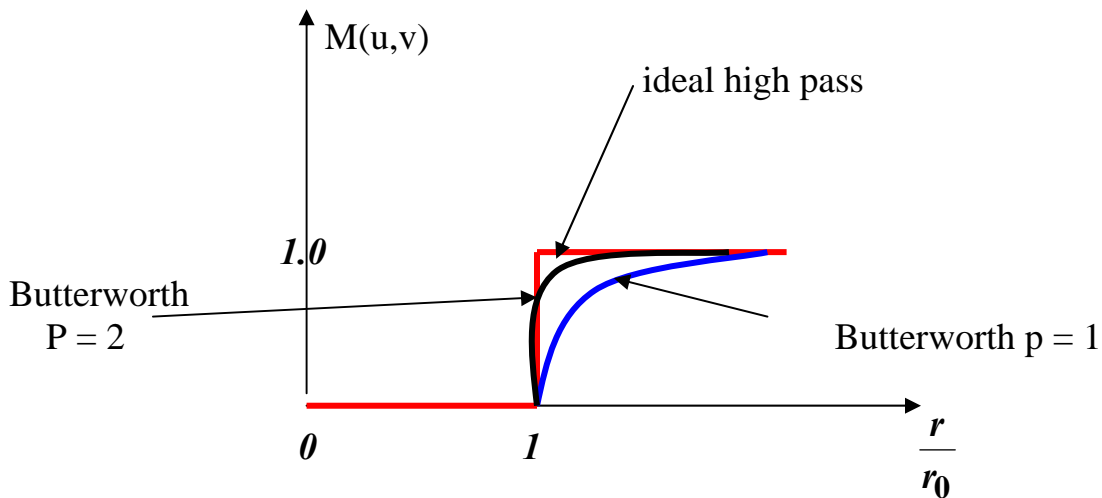


Fig. 5.16 Characteristics of high pass filters

Figure 5.17 shows the result of Butterworth low pass and high pass filter applied to Einstein, with a cut-off frequency of 0.5. As seen high frequencies contribute only to changes in the gray levels.

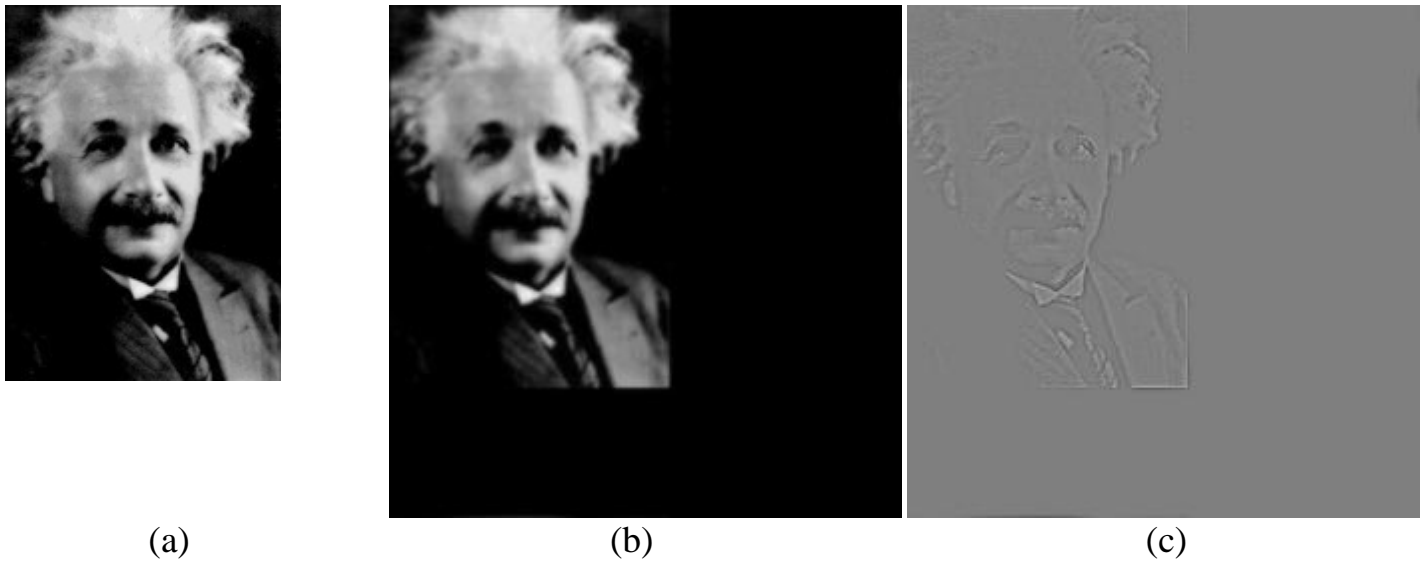
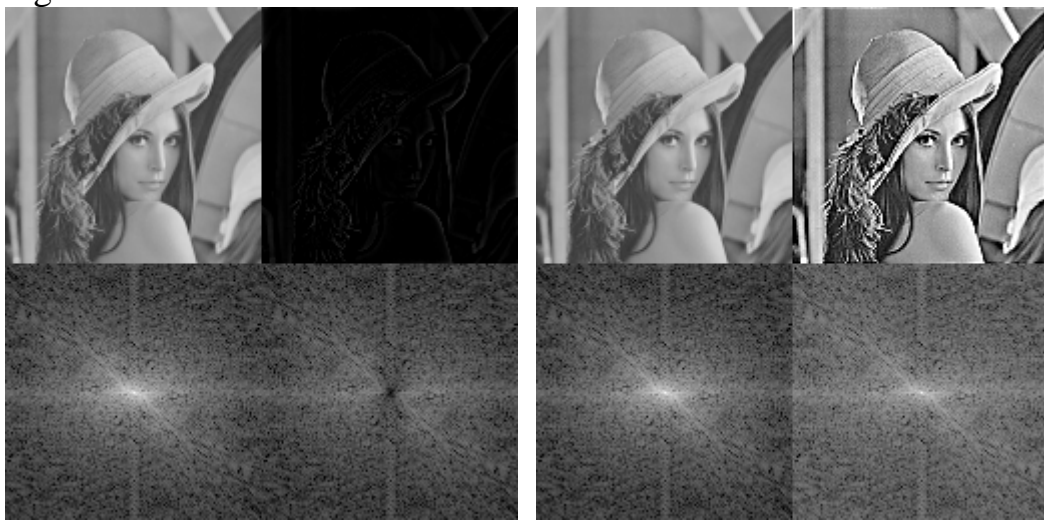


Fig. 5.17 (a) original image, (b) low passed image, and (c) high passed image.



5.7.3 Band Pass and Band Stop Filters

A band pass filter passes a specific range of frequencies, and rejects others. A band stop filter has the opposite characteristics, i.e. it rejects a specific range and passes all other frequencies. The transfer function of the band stop filter can be specified by a cutoff frequency r_0 and a bandwidth w as

$$M_{bs}(u, v) = \frac{1}{1 + (w/(r^2 - r_0^2))^{2p}} \quad (5.30)$$

Note that at frequencies close to r_0 , $M_{hp}(u, v)$ approaches zero (why?, and what is its value at $r = w$?). The band pass transfer function is

$$M_{bp}(u, v) = 1 - M_{bs} \quad (5.31)$$

(why is (5.31) a band pass filter?).

5.7.4 Implementation of Filters in Java

The ImageFFT class provides support filters discussed in Sections 5.7.1-5.7.3. For example for applying a Butterworth high pass filter of order 2 ($p = 2$), we have

```
ImageFFT fft = new ImageFFT(inputImage);
fft.transform();
fft.butterworthLowPassFilter(2, r);
fft.transform();
BufferedImage outputImage = fft.toImage(null);
```

The parameter r is the normalized radius of the filter. Instead of low pass filter in line 3 of the above code, the following can be used.

```
void idealLowPassFilter(double r)
void idealHighPassFilter(double r)
void idealBandPassFilter(double r, double w) //w is the bandwidth
void idealBandStopFilter(double r, double w)
void butterworthLowPassFilter(double r)
void butterworthLowPassFilter(int p, double r)
void butterworthHighPassFilter(double r)
void butterworthHighPassFilter(int p, double r)
void butterworthBandPassFilter(double r, double w)
void butterworthBandPassFilter(int p, double r, double w)
void butterworthBandStopFilter(double r, double w)
void butterworthBandStopFilter(int p, double r, double w)
```