Fourier Theory

and

Filtering in spectral and spatial domains

Image processing methods may be broadly divided into two categories:

Real space methods

-- which work by directly processing the input pixel array.

Fourier space methods

-- which work by firstly deriving a new representation of the input data by performing a *Fourier transform*, which is then processed, and finally, an *inverse Fourier transform* is performed on the resulting data to give the final output image.

What do frequencies mean in an image?

If an image has large values at *high* frequency components then the data is changing rapidly on a short distance scale. (*e.g.,* a page of text).

If the image has large *low* frequency components then the large-scale features of the picture are more important (*e.g.* a single fairly simple object which occupies most of the image).

Fourier Theory

The tool, which converts a spatial (real space) description of an image into one in terms of its frequency components, is called the Fourier transform. The new version is usually referred to as the Fourier space description of the image.

The corresponding *inverse* transformation which turns a Fourier space description back into a real space one is called the inverse Fourier transform.

1D Case:

Considering a continuous function *f*(*x*) of a single variable *x* representing distance. The Fourier transform of that function is denoted *F*(*u*), where *u* represents spatial frequency is defined by:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi xu} dx$$

Note: In general *F*(*u*) will be a complex quantity *even though* the original data is purely real.

The meaning of this is that, not only is the magnitude of each frequency present important, but that its phase relationship is too.

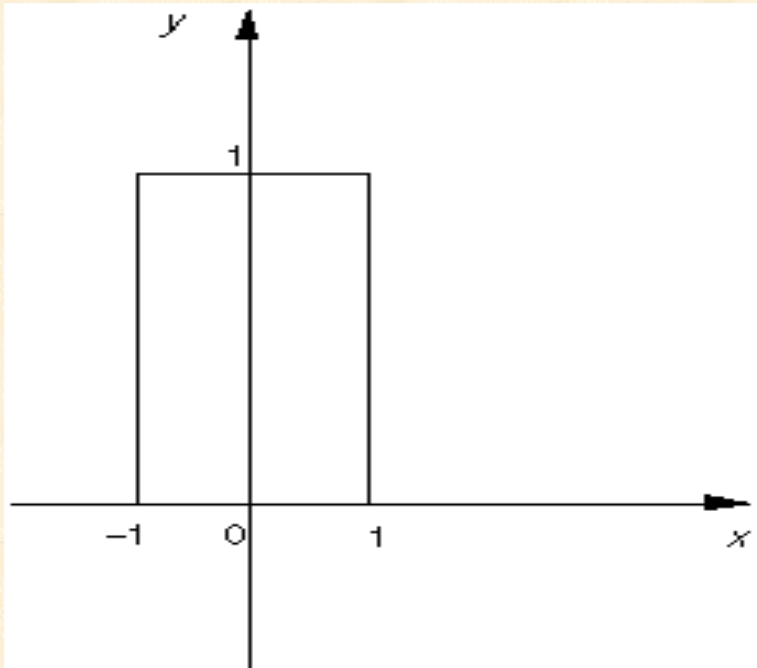
The inverse Fourier transform for regenerating *f*(*x*) from *F*(*u*) is given by:

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi xu} du$$

which is rather similar, except that the exponential term has the opposite sign.

Let's see how we compute a Fourier Transform: consider a particular function *f*(*x*) defined as :

$$f(x) = \begin{cases} 1, & \text{if } |x| \le 1 \\ 0, & \text{otherwise} \end{cases}$$

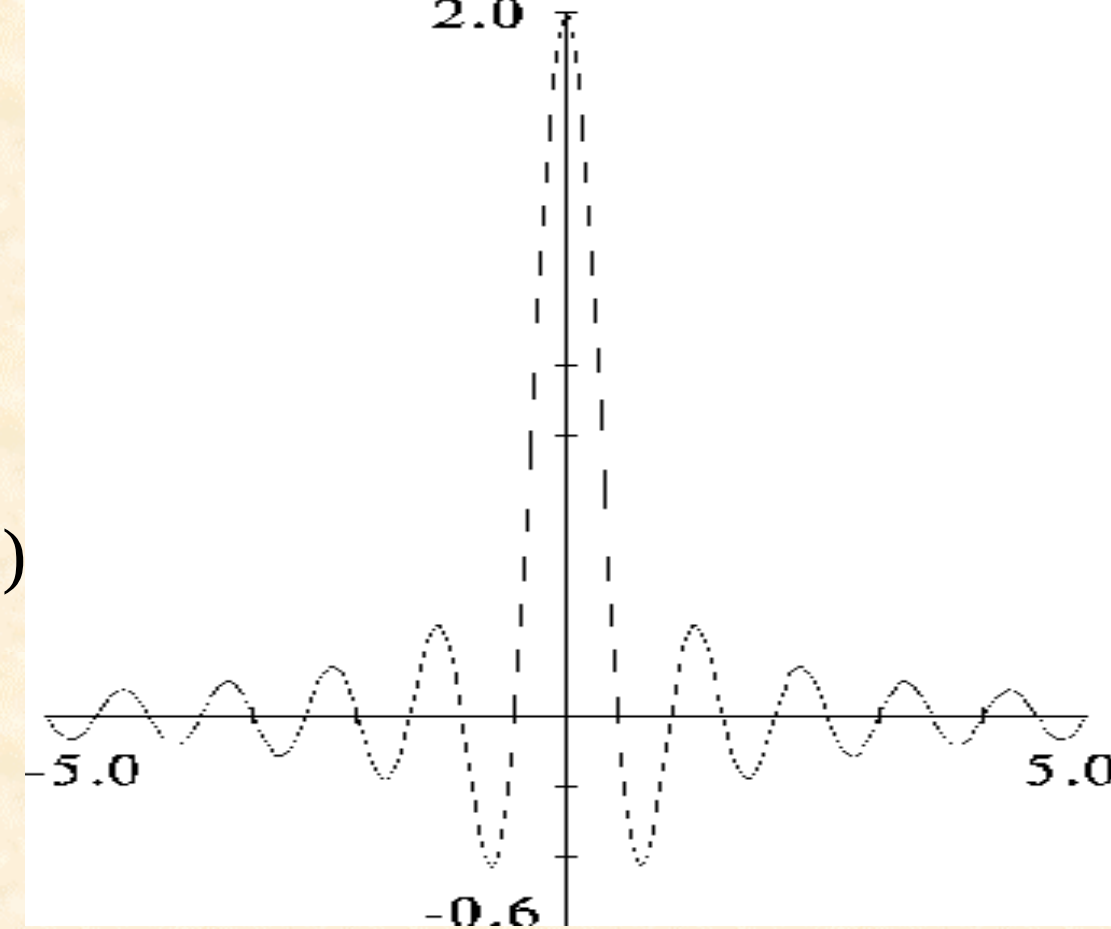


$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi xu} dx$$

$$= \int_{-1}^{1} e^{-j2\pi ux} dx$$

$$= \frac{-1}{j2\pi u}(e^{j2\pi ux} - e^{-j2\pi ux})$$

$$= \frac{\sin 2\pi u}{\pi u}$$



In this case *F*(*u*) is purely real, which is a consequence of the original data being symmetric in *x* and -*x*. This function is often referred to as the *Sinc function*

2D Case - continuous

If *f*(*x*,*y*) is a function, for example the brightness in an image, its Fourier transform is given by:

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(ux+vy)} dxdy$$

and the inverse transform is:

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{j2\pi(ux+vy)} dudv$$

Discrete Case - 1-D

Images are digitized - discrete sequence of numbers. Thus, we need a *discrete* formulation of the Fourier transform, which takes such regularly spaced data values, and returns the value of the Fourier transform for a set of values in frequency space which are equally spaced.

Replacing the integral by a summation, does this quite naturally, gives the *discrete Fourier transform* or DFT for short.

In 1-D it is convenient now to assume that *x* goes up in steps of 1, and that there are *N* samples, at values of *x* from 0 to *N*-1. So the DFT takes the form:

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{-j2\pi ux/N}; \; u = 0,1,...., (N-1)$$

while the inverse DFT is:

$$f(x) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) e^{j2\pi ux/N}; \; x = 0,1,...., (N-1)$$

NOTE: Minor changes from the continuous case are a factor of 1/*N* in the exponential terms, and also the factor 1/*N* in front of the forward transform which does not appear in the inverse transform.

The *2D DFT* works in a similar way. So for an NxM grid in *x* and *y* we have:

$$F(u, v) = \frac{1}{NM} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x, y) e^{-j2\pi(xu/N + yv/M)}$$

$$u = 0,1,...., N-1; v = 0,1,...., M-1$$

and

$$f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u, v) e^{j2\pi(xu/N + yv/M)}$$

$$x = 0,1,...., N-1, y = 0,1,....., M-1$$

Often *N*=*M*, and it is then it is more convenient to redefine *F*(*u*,*v*) by multiplying it by a factor of *N*, so that the forward and inverse transforms are more symmetrical:

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} f(x, y) e^{-j2\pi(xu/N + yv/M)}$$

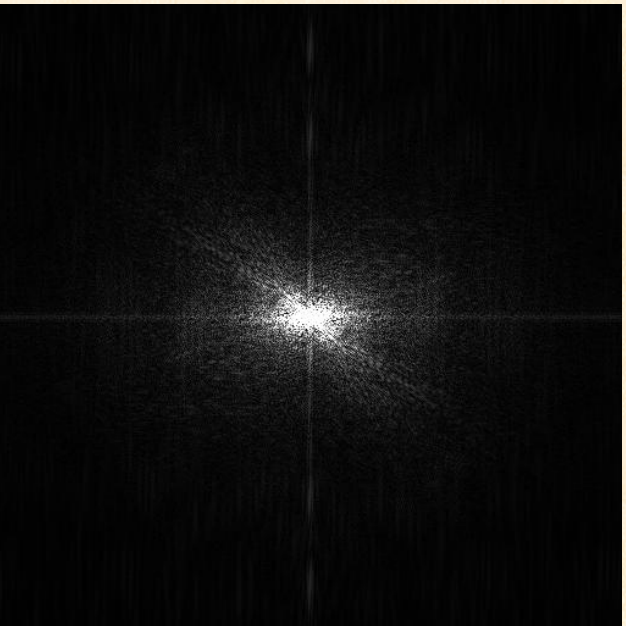
$$u, v = 0,1,..., N - 1$$

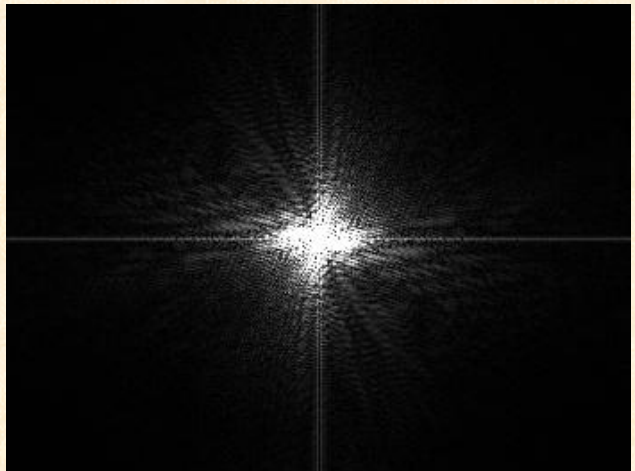
and

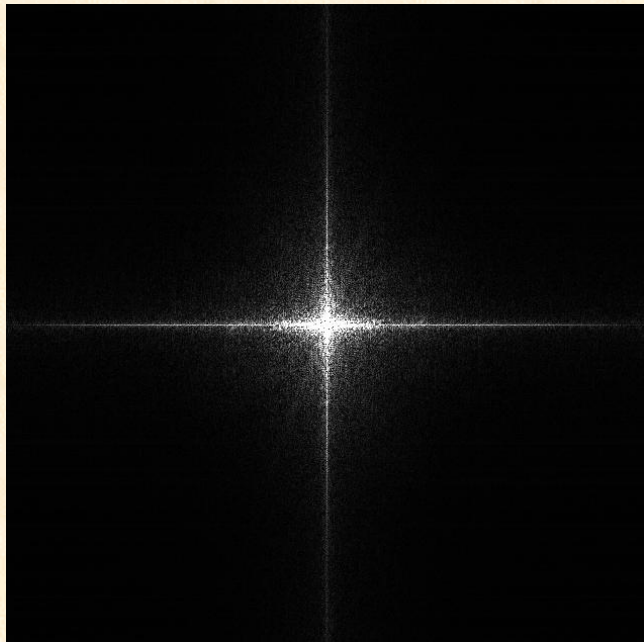
$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} F(u, v) e^{j2\pi(xu/N + yv/M)}$$

$$x, y = 0,1,..., N - 1$$

Digital images and their 2-D DFT's







FILTERING

LOW PASS FILTER

A 2-D ideal lowpass filter (LPF) is one whose transfer function satisfies the relation

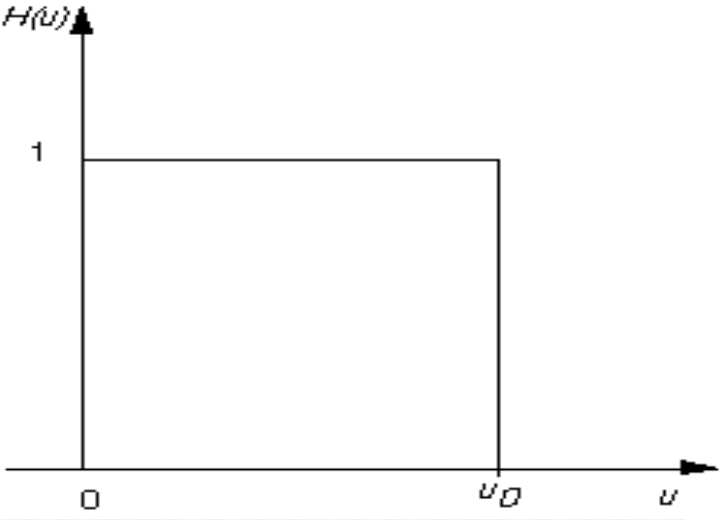
$$H(u, v) = \begin{cases} 1 & \text{if } D(u, v) \leq D_o \\ 0 & \text{if } D(u, v) > D_o \end{cases}$$

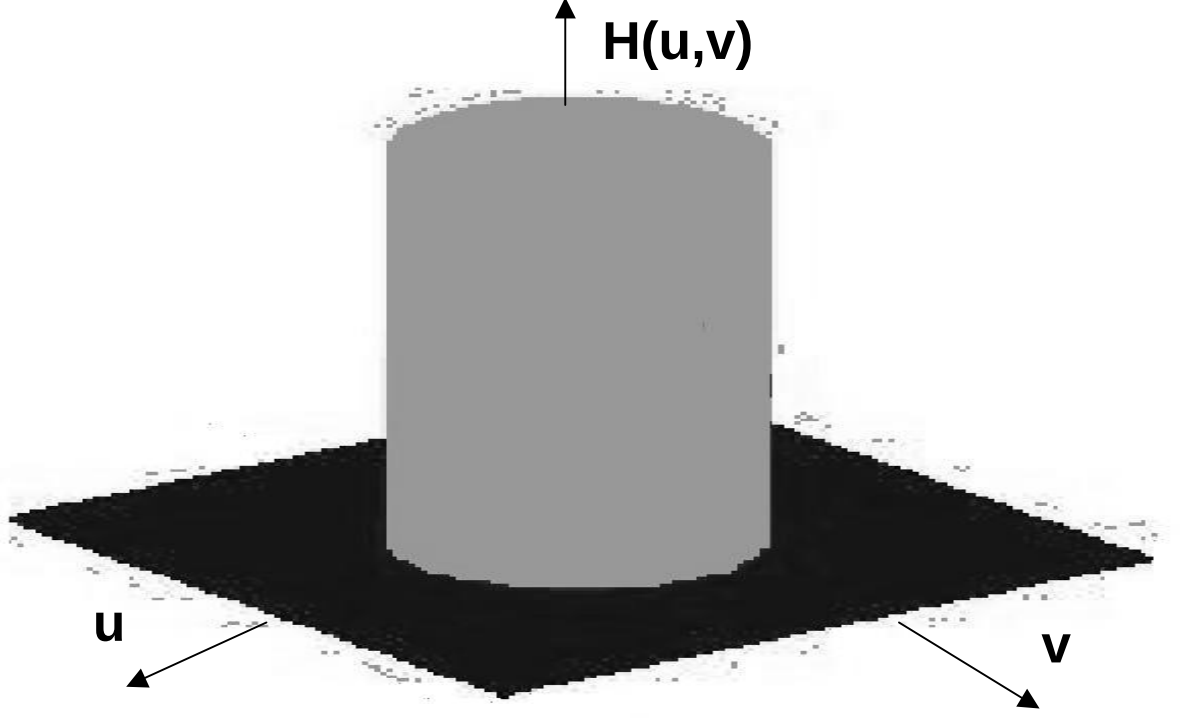
where, D_o is a specified non-negative quantity, and D(u, v) is the distance from point (u,v) to the origin of the frequency plane, that is,

$$D(u, v) = (u^2 + v^2)^{1/2}$$

Low frequency components are responsible for the slowly varying characteristics of an image, such as overall contrast and average intensity.

It blurs the image, since it de-enhances edges and other sharp details in an image which contribute to the high frequency components.





Input Image(Lena)

Low Pass Filtered Image

HIGH PASS FILTER

A 2-D ideal highpass filter (HPF) is one whose transfer function satisfies the relation

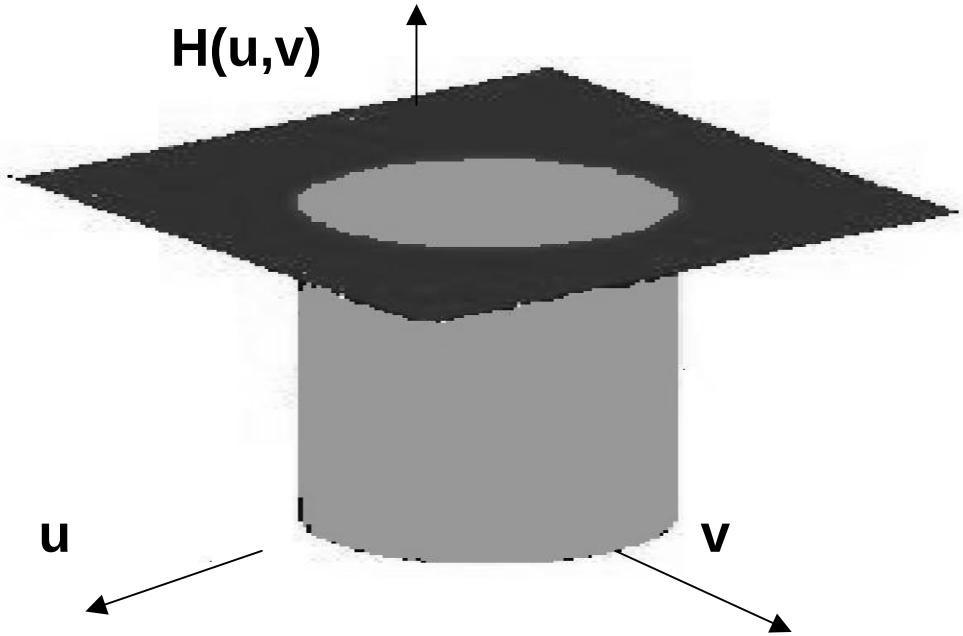
$$H(u, v) = \begin{cases} 0 & \text{if } D(u, v) \leq D_o \\ 1 & \text{if } D(u, v) > D_o \end{cases}$$

Where D_o is the cutoff distance measured from the origin of the frequency plane and D(u, v) is given by

$$D(u, v) = (u^2 + v^2)^{1/2}$$

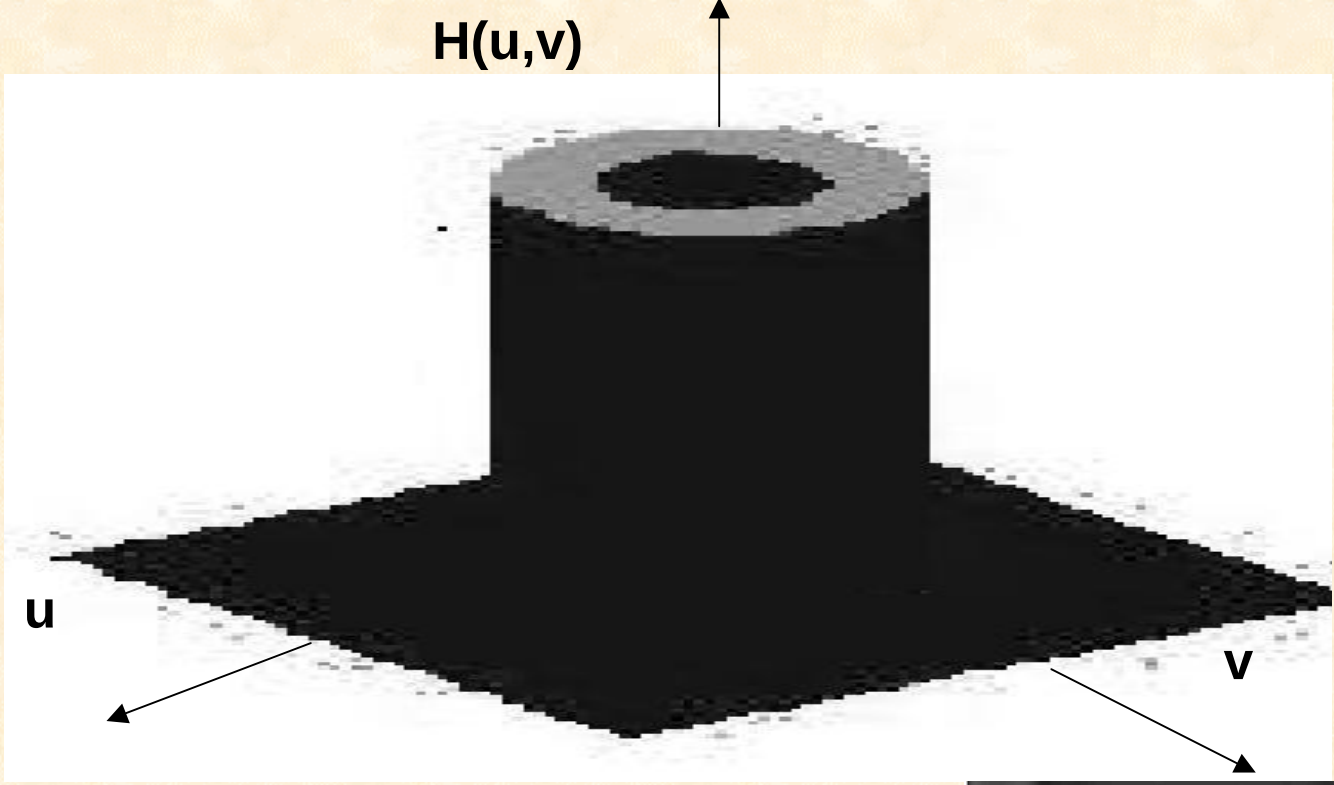
High-frequency components characterize edges and other sharp details.

Highpass filtering causes a loss in the low frequency components in the image, the smaller gray level variations in the image are removed in the output. The output image will have sharpened edges and other sharp details.



Input Image(Lena)

High Pass Filtered Image



Input Image(Lena)

Band-Pass Filtered Image

Comparison of the effects of filtering

Low Pass

Band Pass

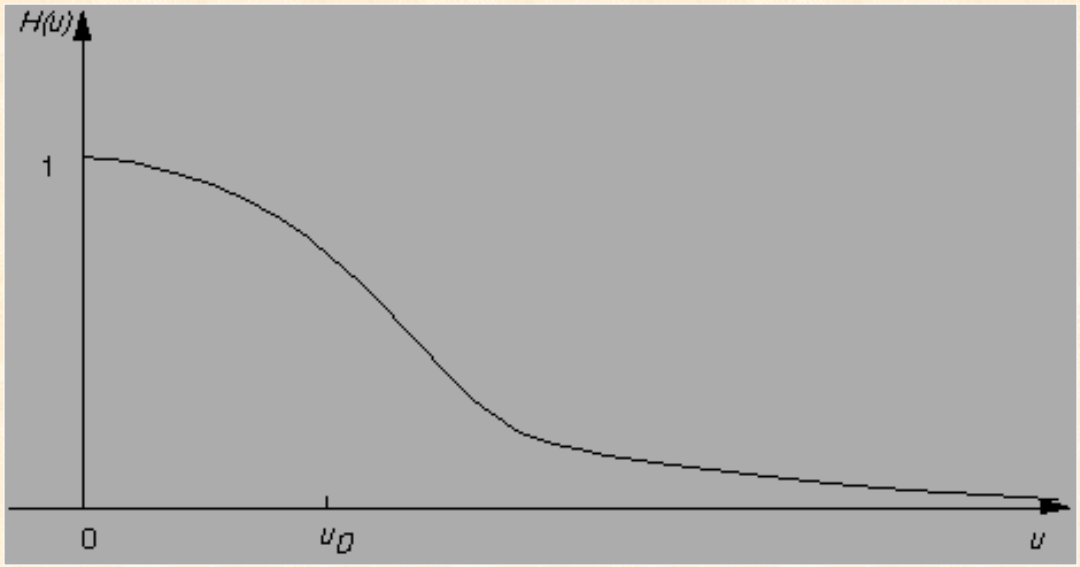
High Pass

Low Pass Butterworth Filter

Another filter sometimes used is the *Butterworth lowpass filter (BLPF)*. In this case, *H*(*u*,*v*) takes the form:

$$H(u, v) = \frac{1}{1 + [(u^2+v^2) / w_0^2]^n}$$

where, *n* is called the order of the filter. This keeps some of the high frequency information, as illustrated by the second order one-dimensional Butterworth filter shown in the figure below:



This is one way of reducing the blurring effect of an ILPF.

Wiener Filter: Adaptive Inverse Filter

Purpose: To Remove noise and/or bluriness in the image.

Estimate the local mean and variance in the neighborhood around each pixel

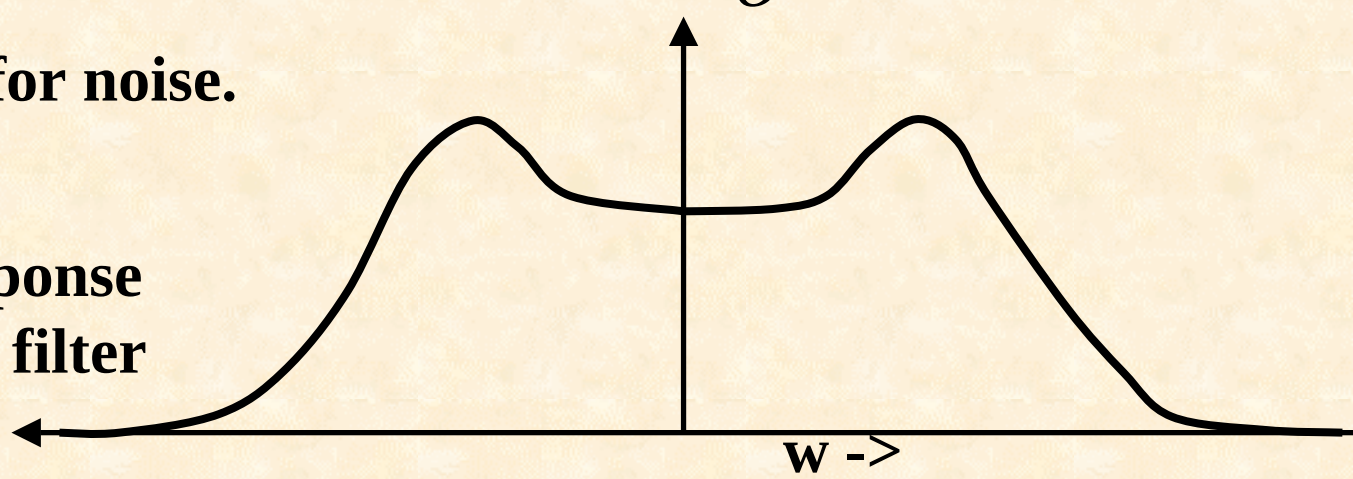
$$\mu = (1/MN)\sum f(x,y) \quad \sigma^2 = (1/MN)\sum f^2(x,y) - \mu^2$$

Wiener filter formulation, for no blur:

$$w(x,y) = \mu + \frac{\sigma^2 - n_v^2}{\sigma^2}(f(x,y) - \mu)$$

Where n_v is the standard deviation for noise.

Typical response For Wiener filter



Convolution

Several important optical effects can be described in terms of convolutions. Let us examine the concepts using 1D continuous functions.

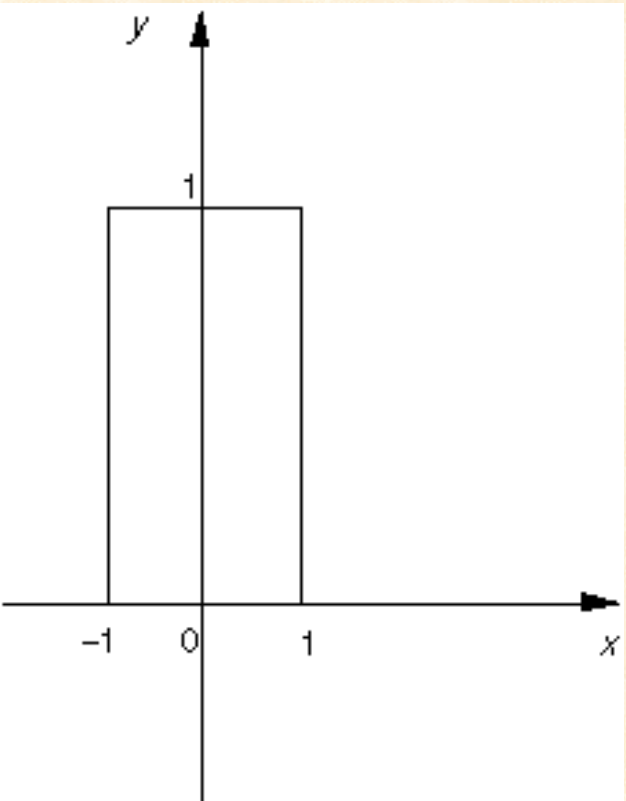
The convolution of two functions *f*(*x*) and *g*(*x*), written *f*(*x*)g*(*x*), is defined by the integral**

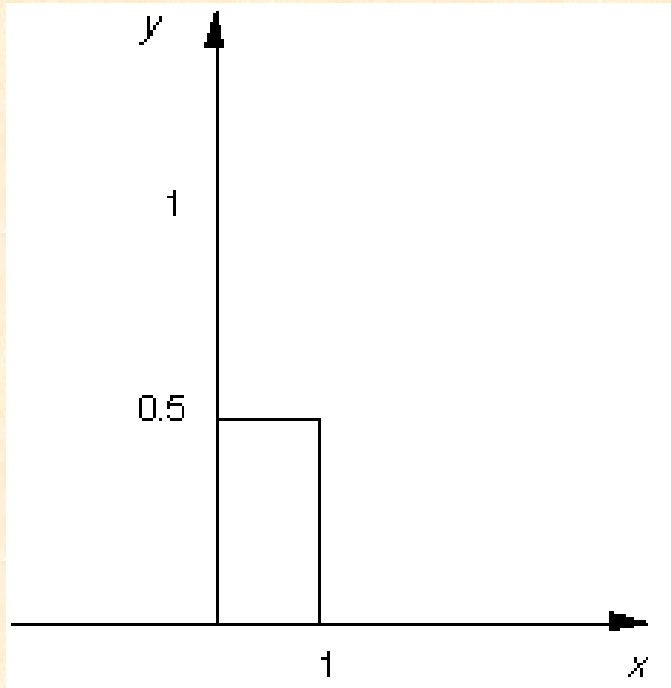
$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\alpha) g(x - \alpha) d\alpha$$

For example, let us take two top hat functions. Let f(x) and g(x) be two top hat functions defined as:

$$f(x) = \begin{cases} 1, & \text{if } |x| \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

$$g(x) = \begin{cases} 1/2, & \text{if } 0 \leq x \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

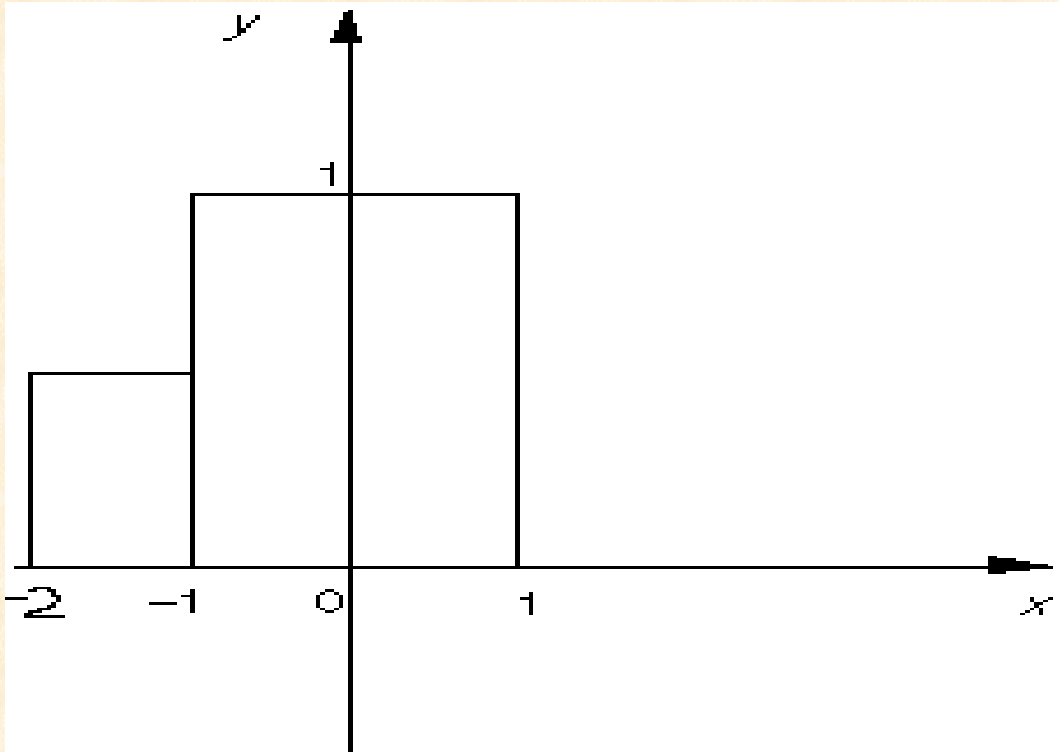




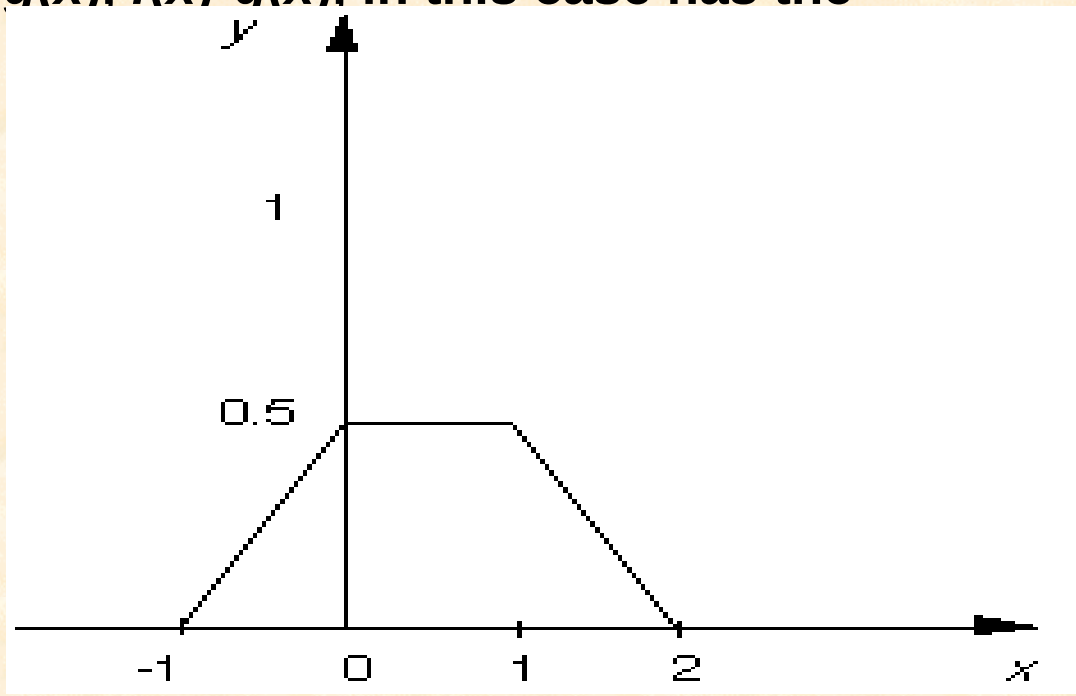
Steps:

- **Form g(-a),**
- **Form g (x - a) by shifting/sliding,**
- **For any given x, get the product of f(a). g(x-a), by finding the overlap of these two functions,**
- **Keep repeating the above step for all values of x**

Example below illustrates the situation, when x = -1:



Thus the convolution of *f*(*x*) and *g*(*x*), *f*(*x*)g*(*x*), in this case has the form :**



Mathematically the output of this convolution can be expressed by:

$$f(x) * g(x) = \begin{cases} (x+1)/2 & \text{if } -1 \le x \le 0 \\ 1/2 & \text{if } 0 \le x \le 1 \\ 1 - x/2 & \text{if } 1 \le x \le 2 \\ 0 & \text{otherwise} \end{cases}$$

Convolution in 2-D

$$1-D:\ m * f(x) = \int f(u)m(x-u)du$$

Continuous case:

$$2-D:\ m * f(x,y) = \int f(u,v)m(x-u,y-v)dudv$$

$$1-D:\ m * f(x) = \sum_i f(x-i)m(i)$$

Discrete case:

$$2-D:\ m * f(x,y) = \sum_i \sum_j f(x-i,y-j)m(i,j)$$

Correlation (discrete case):

$$1-D:\ m \bullet f(x) = \sum_i f(x+i)m(i)$$

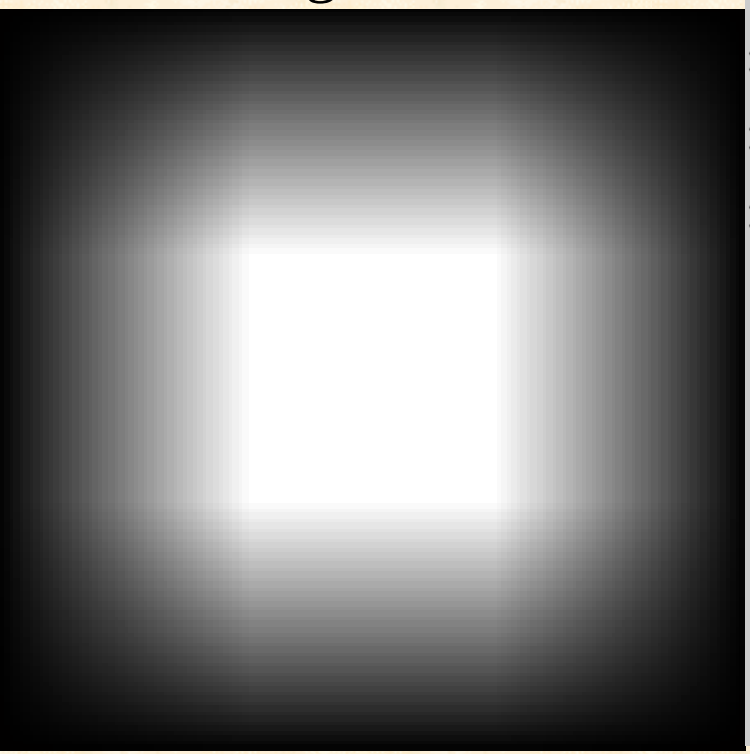
$$2-D:\ m \bullet f(x,y) = \sum_i \sum_j f(x+i,y+j)m(i,j)$$

If we take two functions f and g, as 2-D equivalent of 1-D top-hat functions, (call them as roof-top functions):

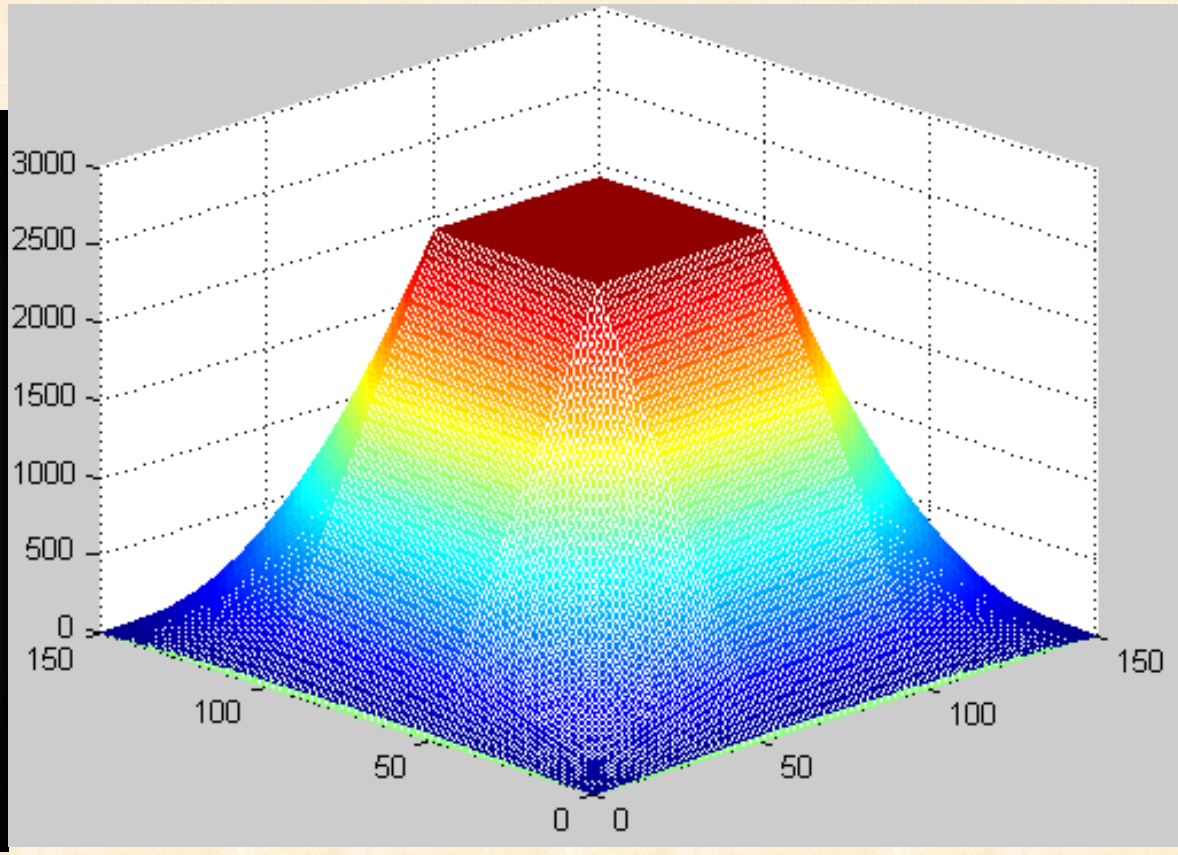
Then the 2D convolution of the functions will result in:

Results displayed as:

Image



Surface Plot



This computation is very time consuming and expensive for large size images.

If the image resolution is 32*32 or less than 64*64, it is recommended to use the above code (i.e. convolve in the spatial domain).

Generally most digital images are larger in size. Then for computational efficiency use the *convolution theorem*:

CONVOLUTION THEOREM

In 1-D :

$$f(x) * g(x) \quad \Leftrightarrow \quad F(u) \,.\, G(u)$$

and

$$f(x) \,.\, g(x) \quad \Leftrightarrow \quad F(u) * G(u)$$

in 2-D :

$$f(x, y) * g(x, y) \quad \Leftrightarrow \quad F(u, v) \,.\, G(u, v)$$

and

$$f(x, y) \,.\, g(x, y) \quad \Leftrightarrow \quad F(u, v) * G(u, v)$$

FFT algorithm exists which computes the Fourier transform of a digitized signal efficiently. Hence it is recommended to first transform the signals to the frequency domain, multiply and then compute the inverse transform to obtain the convolution. Since FFT is computationally efficient, this method works faster for large images/signals.

Equation for filtering: Use convolution theorem

Input Image to be filtered: f(x,y)
Output filtered Image: g(x,y)

Obtain g(x,y) using: $G(u,v) = H(u,v).F(u,v)$,

where, H(u,v) is the filter transfer function, F(u,v) is the DFT of the image, and G(u,v) is the DFT of the output filtered image. Obtain g(x,y), by IDFT of G(u,v).

For bandpass filtering, the transfer function is:

$$H(u, v) = \begin{cases} 1 & \text{if } D1 \le D(u, v) \le D_2 \\ 0 & \text{otherwise} \end{cases}$$

For bandstop filtering, the transfer function is:

$$H(u, v) = \begin{cases} 0 & \text{if } D1 \le D(u, v) \le D_2 \\ 1 & \text{otherwise} \end{cases}$$

Discrete Sampling as convolution

Assume f(x,y) to be continuous and the δ function has the property:

$$\int_{-\infty}^{\infty} a\delta(x) f(x) dx = af(0)$$

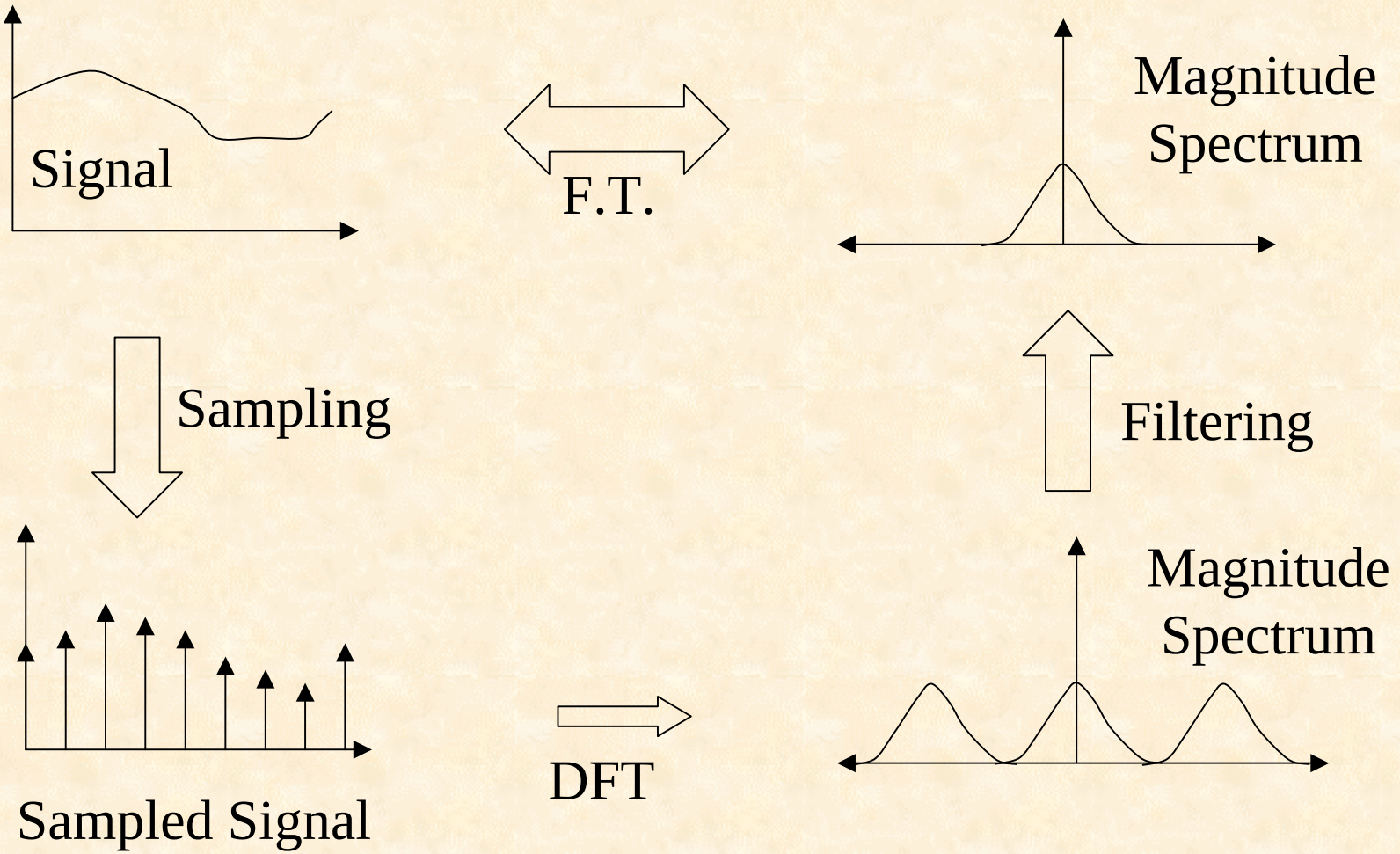
Then the discretized signal $f_d(x,y)$ is:

$$f_d(x,y) = \sum_{i=-\infty}^{\infty}\sum_{i=-\infty}^{\infty} f(i,j)\delta(x-i, y-j)$$

$$= f(x,y)[\sum_{i=-\infty}^{\infty}\sum_{i=-\infty}^{\infty}\delta(x-i, y-j)]$$

Multiply the signal by a set of δ functions, one at each sample point. This is called a ***comb*** function in 1D and ***bed of nails*** in 2D.

Sampling and Aliasing:



The Fourier transform of a sampled signal:

$$\mathrm{F}\{f_d(x,y)\} = \mathrm{F}\{f(x,y)[\sum_{i=-\infty}^{\infty}\sum_{i=-\infty}^{\infty}\delta(x-i,y-j)]\}$$

$$= \mathrm{F}\{f(x,y)\} * \mathrm{F}\{[\sum_{i=-\infty}^{\infty}\sum_{i=-\infty}^{\infty}\delta(x-i,y-j)]\}$$

$$= \mathrm{F}(u,v) * [\sum_{i=-\infty}^{\infty}\sum_{i=-\infty}^{\infty}\delta(u-i,v-j)]\}$$

$$= \sum_{i=-\infty}^{\infty}\mathrm{F}(u-i,v-j)$$

Smoothing - Spatial Domain

The simplest approach is *neighborhood averaging*, where each pixel is replaced by the average value of the pixels contained in some neighborhood about it. The simplest case is probably to consider the group of pixels centered on the given pixel, and to replace the central pixel value by the un-weighted (for weighted - Gaussian function is commonly used) average of these (nine, in case of 3*3 neighborhood) pixels.

For example, the central pixel in Figure below is replaced by the value 13 (the nearest integer to the average).

10	12	11
11	23	12
10	14	15

If any one of the pixels in the neighborhood has a faulty value due to noise, this fault will now be smeared over nine pixels as the image is smoothed. This tends to blur the image.

A better approach is to use a *median filter*.

A similar neighborhood around the pixel under consideration is used, but this time the pixel value is replaced by the *median* pixel value in the neighborhood.

Thus, if we have a 3*3 neighborhood, we write the 9 pixel values in sorted order, and replace the central pixel by the fifth highest value. For example, again taking the data shown in Figure above, the central pixel is replaced by the value 12.

This approach has two advantages.

- **Occasional spurious high or low values are not averaged in -- they are ignored**
- **The sharpness of edges is preserved. To see the latter, consider the pixel data shown in the next slide.**

10	10	20	20
10	10	20	20
10	10	20	20

When the neighborhood covers the left-hand nine pixels, the median value is 10; when it covers the right hand ones, the median value is 20; thus the edge is preserved.

- **If there are large amounts of noise in an image, more than one pass of median filtering may be useful to further reduce the noise.**

- **A rather different real space technique for smoothing is to average multiple copies of the image.**

- **The idea is that over several images, the noise will tend to cancel itself out if it is independent from one image to the next.**

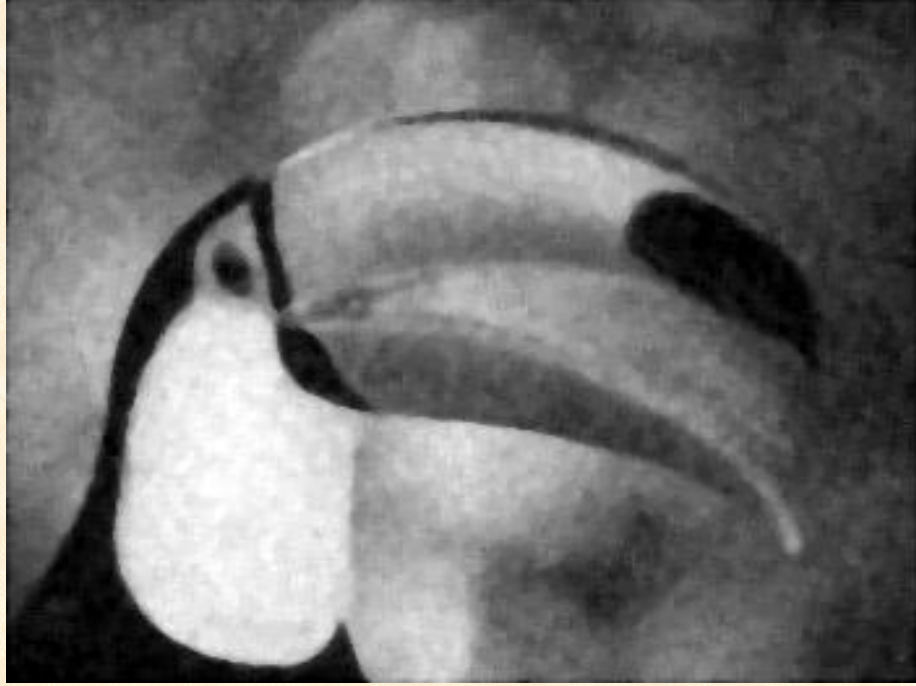
- **Statistically, we expect the effects of noise to be reduced by a factor $n^{-1/2}$, if we use *n* images. One particular situation where this technique is of use, is in low lighting conditions.**

Original Image

Median Filtered Image

Noisy Image

Original Image



Median Filtered Image

Noisy Image

Original Image

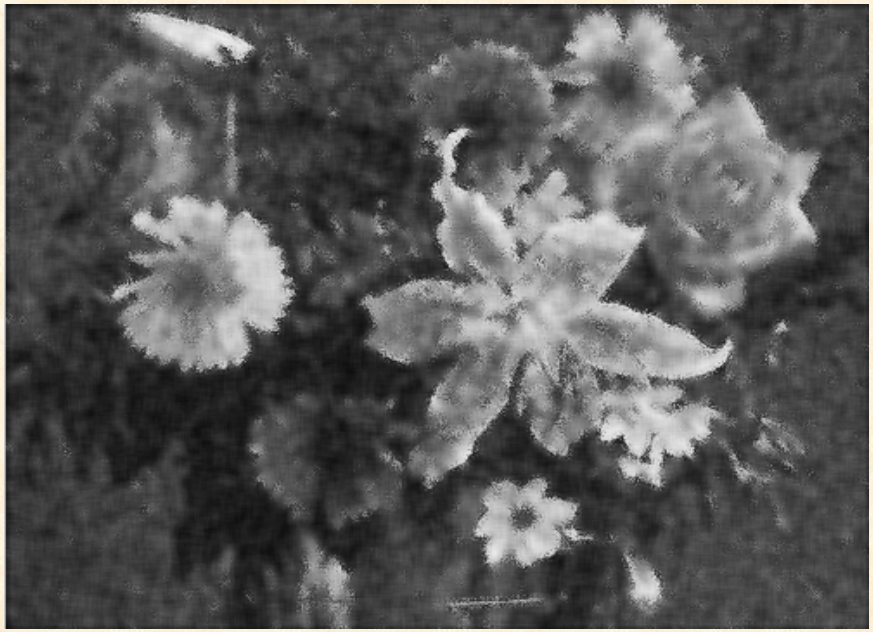
Noisy Image

Median Filtered Image

Original Image

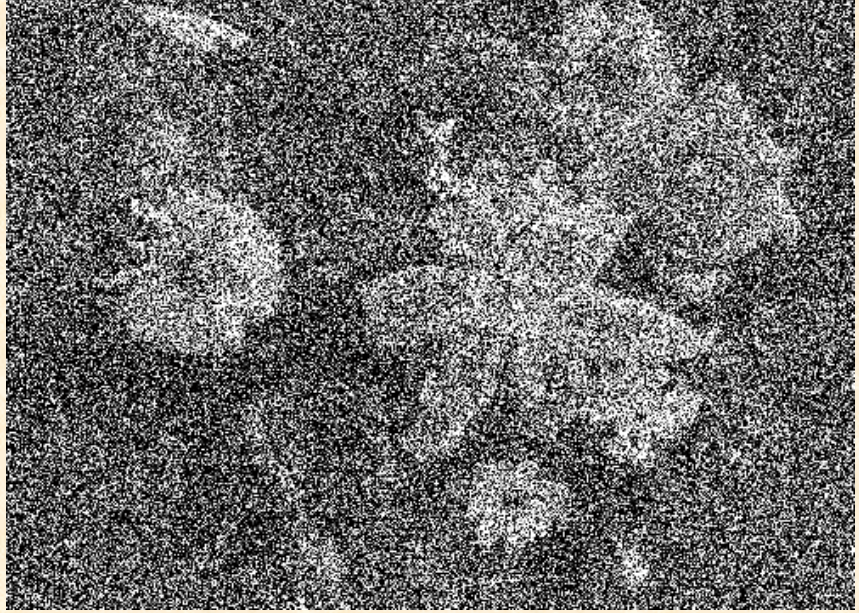
Noisy Image

Median Filtered Image

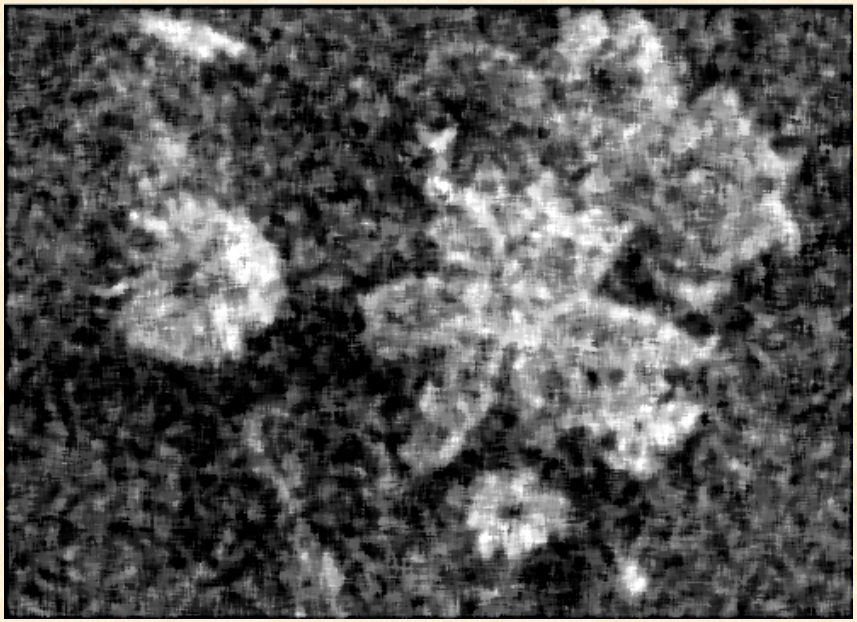


Filtered using Wiener Filter

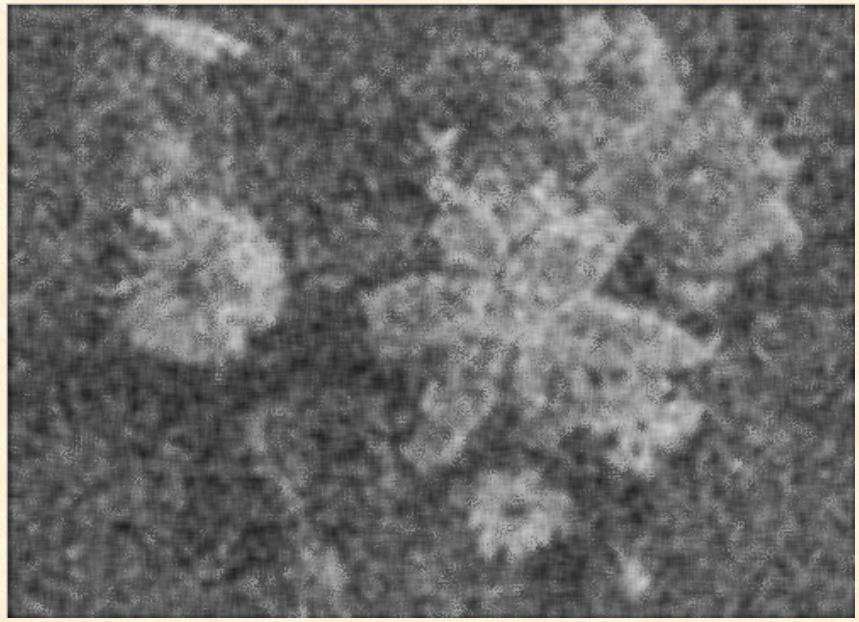
Original Image



Noisy Image



Median Filtered Image



Filtered using Wiener Filter

References

1. “Digital Image Processing”; R. C. Gonzalez and R. E. Woods; Addison Wesley; 1992+.

2. “Fundamentals of Digital Image Processing”; Anil K Jain; Prentice Hall of India; 1995+.

3. “Digital Image Processing and Computer Vision”; Robert J. Schallkoff; John Wiley and Sons; 1989+.

4. “ Pattern Recognition: Statistical. Structural and Neural Approaches”; Robert J. Schallkoff; John Wiley and Sons; 1992+.

5. “Algorithms for Image Processing and Computer Vision”; J. R. Parker; John Wiley and Sons; 1997+.