IMAGE PROCESSING LABORATORY | SPRING ‘15

ASSIGNMENT - 4

GROUP-7

ANKIT BANSAL | 12EC35024
AMRUT RAJKARNE | 12EC35017

ASSIGNMENT – 4
FREQUENCY FILTERING

Aim:

Write C++/Image-J modular functions to perform the following operations on the **512 × 512 grayscale test images**, e.g. lena_gray_512.jpg, jetplane.jpg, lake.jpg, livingroom.jpg, mandril_gray.jpg, pirate.jpg, walkbridge.jpg.

1. **FFT2** (takes input image filename as the argument; gives 2D FFT coefficients as output)

2. **IFFT2** (takes 2D FFT coefficients as input argument; gives the back-projected/ reconstructed image as output)

3. Perform **Ideal**, **Gaussian**, and **Butterworth** low-pass and high-pass filtering, taking cut-off frequency, D0, and image filename as input arguments) respectively with

Display the (shifted) magnitude spectrums of the input, the filter and the filtered output.

You may make use of the tracker/slider function to choose images, filter types and cut-off frequencies.

Theory:

Frequency domain filtering is much more straightforward in terms of filtering operation. In this case, we compute the Fourier transform of the image, do the filtering operation and take the inverse transform to get back the filtered image.

Filtering:

For ***lowpass* filtering**, high frequency components are removed resulting in blurring of the image.

Mathematically, we can write :

$$G(u,v) = H(u,v)*F(u,v)$$

Where F(u,v) is the fourier transform of the input image. We need to select the H(u,v) filter that will reduce the high frequency components.

Ideal LPF will have following transfer function:

$$H(u,v) = 1 \text{ …. if } D(u,v) < D_o$$

$$0 \text{ … if } D(u,v) > D_o$$

In case of **Butterworth LPF**,

$$H(u,v) = 1/ [1 + D(u,v)/ D_o]$$

For ***highpass* filtering**, removal of low frequency components cause sharpening of the image.

The filter transfer function H(u,v) will have the following relation :

Ideal HPF: $H(u,v) = 0 \ldots.$ if $D(u,v) < D_o$

$1 \ldots$ if $D(u,v) > D_o$

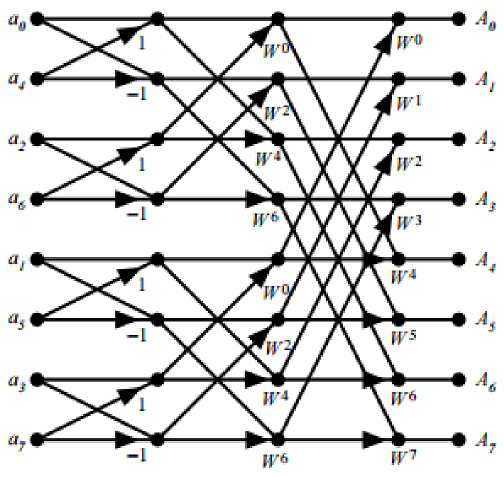
Butterworth HPF:

$$H(u,v) = 1/ [1 + D_o /D(u,v)]$$

Fast Fourier Transform:

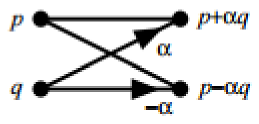
The FFT is a fast algorithm for computing the DFT. If we take the 2-point DFT and 4-point DFT and generalize them to 8-point, 16-point ... 2r -point, we get the FFT algorithm. To compute the DFT of an N-point sequence using equation (1) would take **O(N^2)** multiplies and adds. The FFT algorithm computes the DFT using **O(N log N)** multiplies and adds. There are many variants of the FFT algorithm.

The **"decimation in-time" FFT algorithm** for sequences whose length is a power of two (N = 2r for some integer r). Below is a diagram of an 8-point FFT, where W = W8 = $e^{-i\pi/4}$ = (1 − i)/√2:



Butterflies and Bit-Reversal.

The FFT algorithm decomposes the DFT into $\log_2 N$ stages, each of which consists of N/2 butterfly computations. Each butterfly takes two complex numbers p and q and computes from them two other numbers, $p + \alpha q$ and $p - \alpha q$, where α is a complex number. Below is a diagram of a butterfly operation.



In the diagram of the 8-point FFT above, note that the inputs aren't in normal order: a0, a1, a2, a3, a4, a5, a6, a7, they're in the bizarre order: a0, a4, a2, a6, a1, a5, a3, a7. Why this sequence?

Below is a table of j and the index of the jth input sample,

j	0	1	2	3	4	5	6	7
n_j	0	4	2	6	1	5	3	7
j base 2	000	001	010	011	100	101	110	111
n_j base 2	000	100	010	110	001	101	011	111

Bit Reversal

The pattern is obvious if j and n_j are written in binary (last two rows of the table).

Observe that each n_j is the bit-reversal of j. The sequence is also related to breadth-first traversal of a binary tree. It turns out that this FFT algorithm is simplest if the input array is rearranged to be in bit-reversed order

General FFT and IFFT Algorithm for $N = 2^r$.

The previously diagrammed algorithm for the 8-point FFT is easily generalized to any power of two. The input array is bit-reversed, and the butterfly coefficients can be seen to have exponents in arithmetic sequence modulo N. For example, for N = 8, the butterfly coefficients on the last stage in the diagram are W0, W1, W2, W3, W4, W5, W6, W7. That is, powers of W in sequence. The coefficients in the previous stage have exponents 0,2,4,6,0,2,4,6, which is equivalent to the sequence 0,2,4,6,8,10,12,14 modulo 8. And the exponents in the first stage are 1,-1,1,-1,1,-1,1,-1, which is equivalent to W raised to the powers 0,4,0,4,0,4,0,4, and this is equivalent to the exponent sequence 0,4,8,12,16,20,24,28 when taken modulo 8.

The width of the butterflies (the height of the ”X’s” in the diagram) can be seen to be 1, 2, 4, ... in successive stages, and the butterflies are seen to be isolated in the first stage (groups of 1), then clustered into overlapping groups of 2 in the second stage, groups of 4 in the 3rd stage, etc. The generalization to other powers of two should be evident from the diagrams for N = 4 and N = 8.

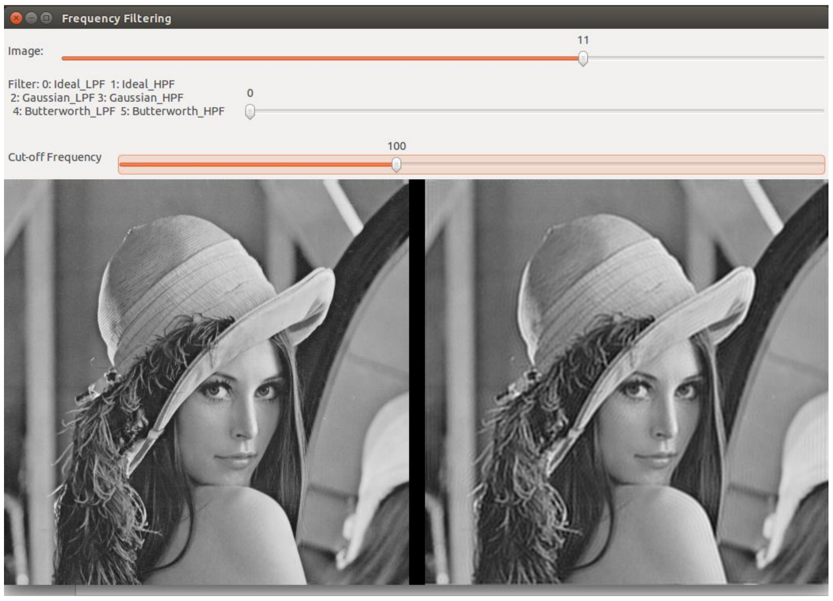
The inverse FFT (IFFT) is identical to the FFT, except one exchanges the roles of a and A, the signs of all the exponents of W are negated, and there’s a division by N at the end

Implemented Algorithm:

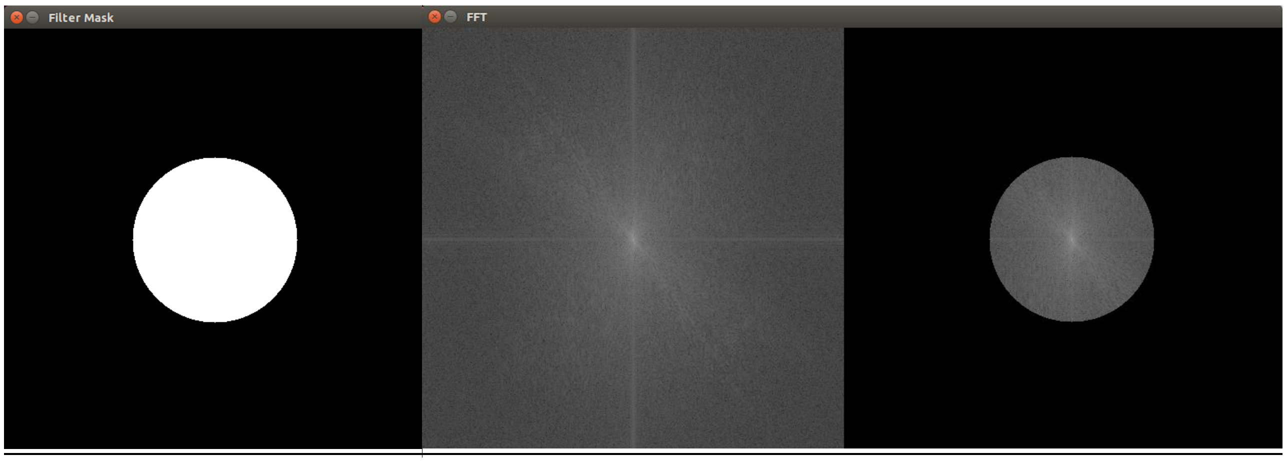
- To realize frequency domain filter, first we compute the Fourier Transform of the image using **FFT algorithm**, decimation in time
- Then we do the filtering operation where we **multiply the image 2-D array with the filter array element wise** and replace the gray level of each pixel of the image with the corresponding product
- Finally, we perform **the inverse Fourier Transform** of the resultant array to get back the filtered image.

Results:

- **Ideal LPF** –


Frequency Filtering
Image:
11
Filter: 0: Ideal_LPF 1: Ideal_HPF
2: Gaussian_LPF 3: Gaussian_HPF
4: Butterworth_LPF 5: Butterworth_HPF
0
Cut-off Frequency
100

Input vs Output



Filter Mask **FFT of Image** **Filtered Response**

Similarly the output results for the other filter types, magnitude spectrum of input, filter and filtered output are documented in the **'Result Images'** folder