Image Processing Lab

Frequency Filtering

GROUP-10

Submitted by:

Aditya Narayan (13EC35010)

Adarsh Kumar Kosta (13EC35008)

Objective

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

Ideal_LPF	**Ideal_HPF**
Gaussian_LPF	**Gaussian_HPF**
Butterworth_LPF	**Butterworth_HPF**

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

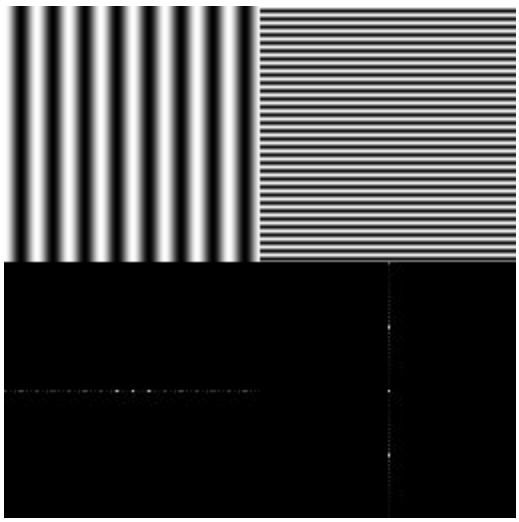
You may make use of the tracker/slider function to choose images, filter types and cut-off frequencies.

Introduction

Fourier Analysis of Images

The Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

The Fourier Transform is used in a wide range of applications, such as image analysis, image filtering, image reconstruction and image compression.



The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image. The number of frequencies corresponds to the number of pixels in the spatial domain image, i.e. the image in the spatial and Fourier domain are of the same size.

For a square image of size N×N, the two-dimensional DFT is given by:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j)\, e^{-\imath 2\pi (\frac{ki}{N} + \frac{lj}{N})}$$

where f(a,b) is the image in the spatial domain and the exponential term is the basis function corresponding to each point F(k,l) in the Fourier space. The equation can be interpreted as: the value of each point F(k,l) is obtained by multiplying the spatial image with the corresponding base function and summing the result.

The basis functions are sine and cosine waves with increasing frequencies, i.e. F(0,0) represents the DC-component of the image which corresponds to the average brightness and F(N-1,N-1) represents the highest frequency.

In a similar way, the Fourier image can be re-transformed to the spatial domain. The inverse Fourier transform is given by:

$$f(a,b) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} F(k,l)\, e^{\imath 2\pi (\frac{ka}{N} + \frac{lb}{N})}$$

Note the *1/N*^{2} normalization term in the inverse transformation. This normalization is sometimes applied to the forward transform instead of the inverse transform, but it should not be used for both.

To obtain the result for the above equations, a double sum has to be calculated for each image point. However, because the Fourier Transform is separable, it can be

$$F(k,l) = \frac{1}{N} \sum_{b=0}^{N-1} P(k,b)\, e^{-\imath 2\pi \frac{lb}{N}}$$

written as:
where
Using these two formulas, the spatial domain image is first transformed into an intermediate image using N one-dimensional Fourier Transforms. This intermediate image is then transformed into the final image, again using N one-dimensional

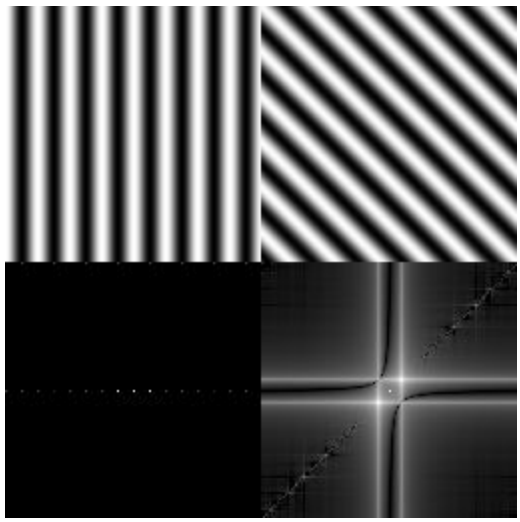
$$P(k,b) = \frac{1}{N} \sum_{a=0}^{N-1} f(a,b)\, e^{-\imath 2\pi \frac{ka}{N}}$$

Fourier Transforms. Expressing the two-dimensional Fourier Transform in terms of a series of 2N one-dimensional transforms decreases the number of required computations.

Even with these computational savings, the ordinary one-dimensional DFT has N^2 complexity. This can be reduced to $Nlog_2 N$ if we employ the Fast Fourier Transform (FFT) to compute the one-dimensional DFTs. This is a significant improvement, in particular for large images. There are various forms of the FFT and most of them restrict the size of the input image that may be transformed, often to $N=2^n$ where n is an integer. The mathematical details are well described in the literature.

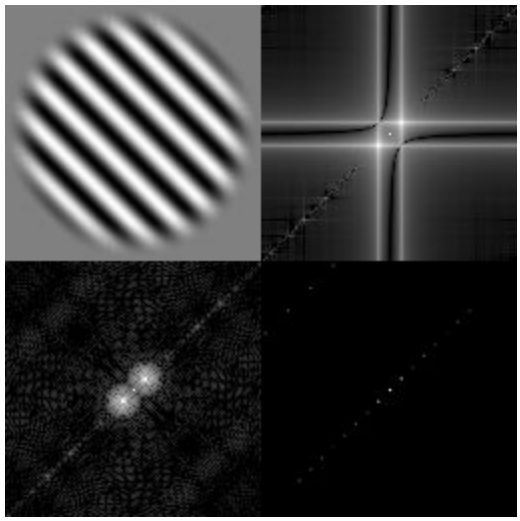
Rotation and Edge Effects

In general, rotation of the image results in equivalent rotation of its FT. To see that this is true, we will take the FT of a simple cosine and also the FT of a rotated version of the same function. The results can be seen by:

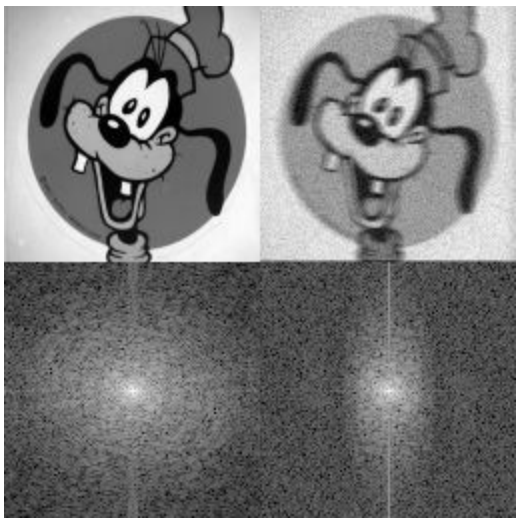


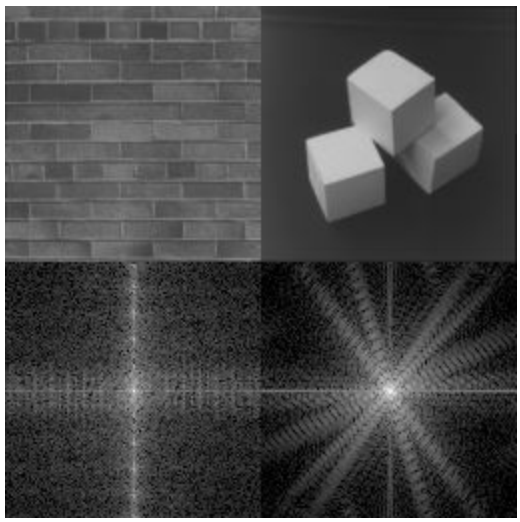
At first, the results seem rather surprising. The horizontal cosine has its normal, very simple FT. But the rotated cosine seems to have an FT that is much more complicated, with strong diagonal components, and also strong "plus sign" shaped horizontal and vertical components. The question is, where did these horizontal and vertical components come from? The answer is that the FT always treats an image as if it were part of a periodically replicated array of identical images extending horizontally and vertically to infinity. And there are strong edge effects between the neighbors of such a periodic array as can be seen by:

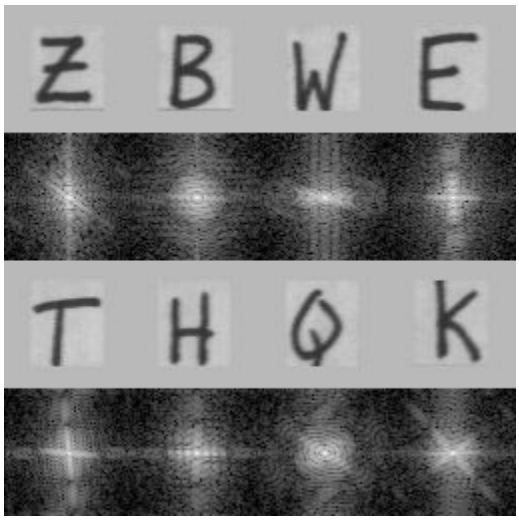
Thus, what we see as the FT in the "slant" image (lower right of the image before last) is actually the combination of the actual FT of the cosine function and that caused by the edge effects of looking at a finite part of the image. These edge effects can be significantly reduced by "windowing" the image with a function that slowly tapers off to a medium gray at the edge. The result can be seen by:

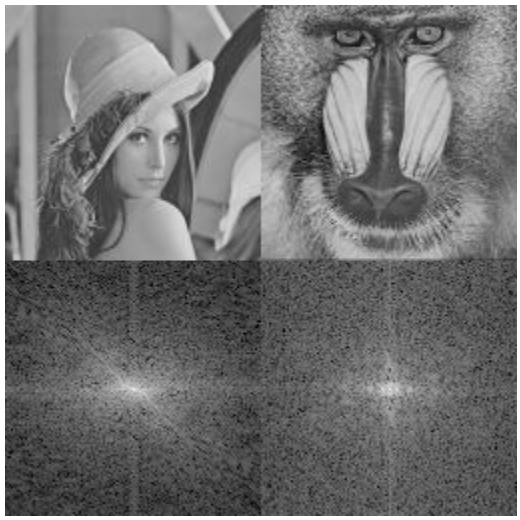


Some example image FFTs:









Frequency Filtering

Frequency filters process an image in the frequency domain. The image is Fourier transformed, multiplied with the filter function and then re-transformed into the spatial domain. Attenuating high frequencies results in a smoother image in the spatial domain, attenuating low frequencies enhances the edges.

Frequency filtering is based on the Fourier Transform. (For the following discussion we assume some knowledge about the Fourier Transform, therefore it is advantageous if you have already read the corresponding worksheet.) The operator usually takes an image and a filter function in the Fourier domain. This image is then multiplied with the filter function in a pixel-by-pixel fashion:

$$G(k,l) = F(k,l)H(k,l)$$

where *F(k,l)* is the input image in the Fourier domain, *H(k,l)* the filter function and *G(k,l)* is the filtered image. To obtain the resulting image in the spatial domain, *G(k,l)* has to be re-transformed using the inverse Fourier Transform.
Since the multiplication in the Fourier space is identical to convolution in the spatial domain, all frequency filters can in theory be implemented as a spatial filter. However, in practice, the Fourier domain filter function can only be approximated by the filtering kernel in spatial domain.

The form of the filter function determines the effects of the operator. There are basically three different kinds of filters: lowpass, highpass and bandpass filters. A low-pass filter attenuates high frequencies and retains low frequencies unchanged. The result in the spatial domain is equivalent to that of a smoothing filter; as the blocked high frequencies correspond to sharp intensity changes, i.e. to the fine-scale details and noise in the spatial domain image.

A highpass filter, on the other hand, yields edge enhancement or edge detection in the spatial domain, because edges contain many high frequencies. Areas of rather constant gray-level consist of mainly low frequencies and are therefore suppressed.
A bandpass attenuates very low and very high frequencies, but retains a middle range band of frequencies. Bandpass filtering can be used to enhance edges (suppressing

low frequencies) while reducing the noise at the same time (attenuating high frequencies).

The different filters used are discussed below:

Ideal Filter

An ideal low-pass filter completely eliminates all frequencies above the cutoff frequency while passing those below unchanged; its frequency response is a rectangular function and is a brick-wall filter. The transition region present in practical filters does not exist in an ideal filter. An ideal low-pass filter can be realized mathematically (theoretically) by multiplying a signal by the rectangular function in the frequency domain or, equivalently, convolution with its impulse response, a sinc function, in the time domain.

However, the ideal filter is impossible to realize without also having signals of infinite extent in time, and so generally needs to be approximated for real ongoing signals, because the sinc function's support region extends to all past and future times. Real filters for real-time applications approximate the ideal filter by truncating and windowing the infinite impulse response to make a finite impulse response; applying that filter requires delaying the signal for a moderate period of time, allowing the computation to "see" a little bit into the future. This delay is manifested as phase shift. Greater accuracy in approximation requires a longer delay.

An ideal low-pass filter results in ringing artifacts via the Gibbs phenomenon. These can be reduced or worsened by choice of windowing function, and the design and choice of real filters involves understanding and minimizing these artifacts.

Filter Equations:

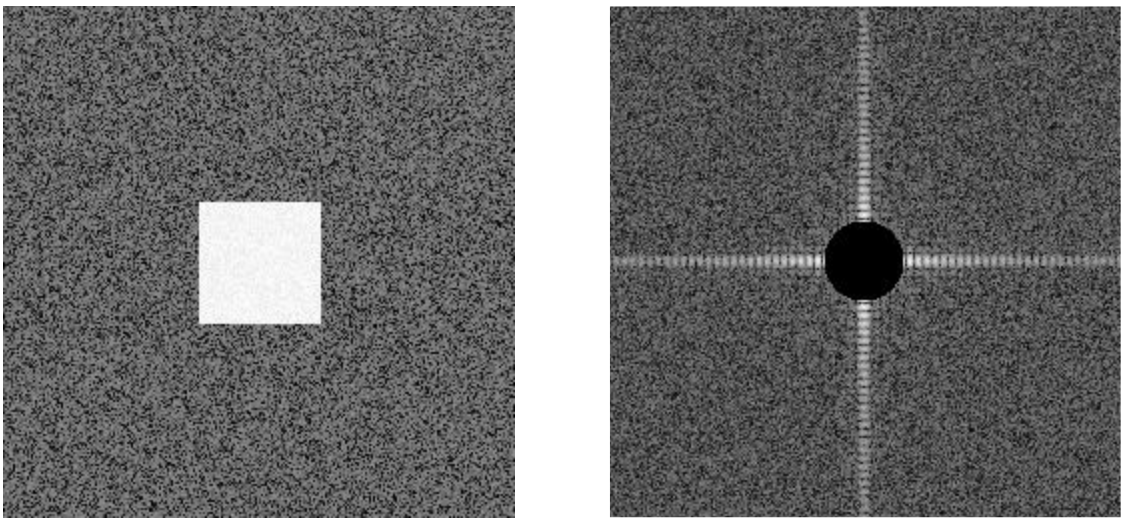
Ideal Low-Pass: $H(u,v) = 0 \quad if\ D(u,v) \leq Do$

$= 1 \quad if\ D(u,v) \geq Do$

Ideal High-Pass: $H(u,v) = 0 \quad if\ D(u,v) \geq Do$

$= 1 \quad if\ D(u,v) \leq Do$

where $Do = sqrt(pow(u,2) + pow(v,2))$

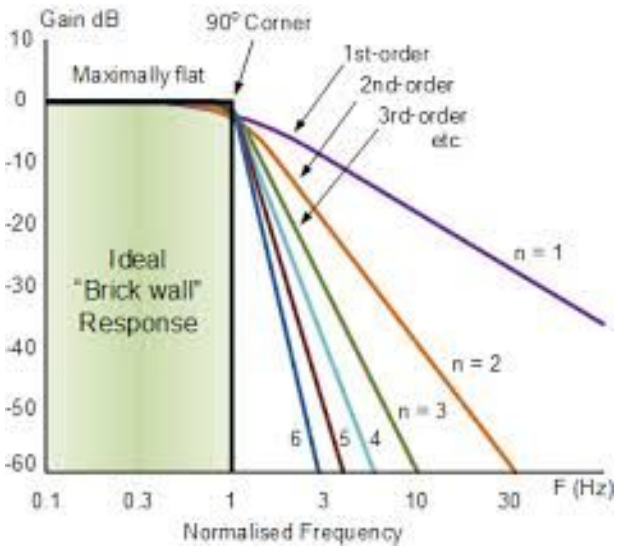


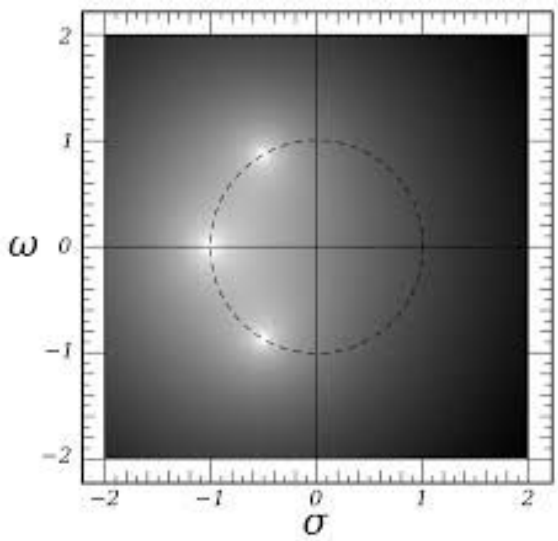
Low pass and High pass Ideal Filters

Butterworth Filter

The Butterworth filter is a type of signal processing filter designed to have as flat a frequency response as possible in the passband. It is also referred to as a maximally flat magnitude filter.

The frequency response of the Butterworth filter is maximally flat (i.e. has no ripples) in the passband and rolls off towards zero in the stopband. When viewed on a logarithmic Bode plot, the response slopes off linearly towards negative infinity.





Filter Equations:

Butterworth Low-Pass: $$H(u,v) = \frac{1}{1+[D(u,v)/D0]^{2n}}$$

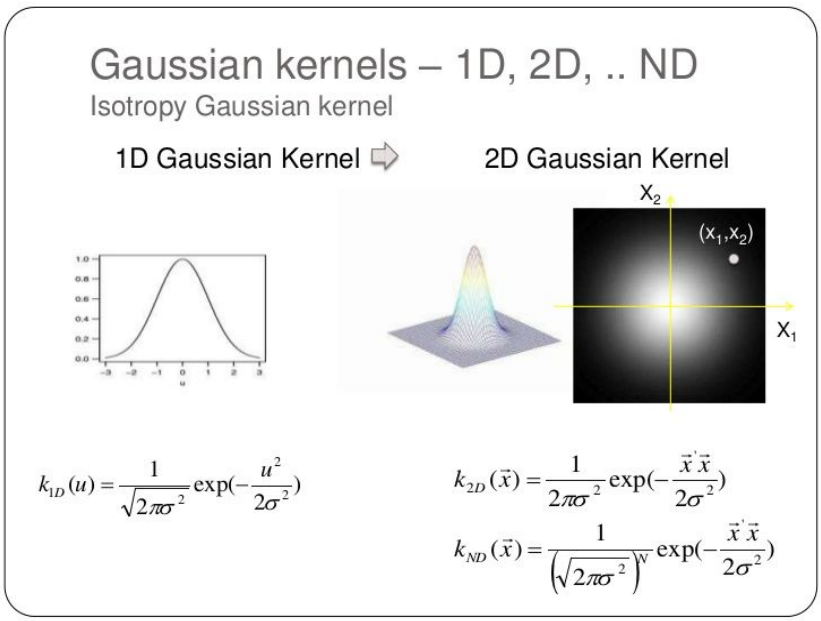
Butterworth High-Pass: $$H(u,v) = \frac{1}{1+[D0/D(u,v)]^{2n}}$$

Gaussian Filter

In electronics and signal processing, a Gaussian filter is a filter whose impulse response is a Gaussian function (or an approximation to it). Gaussian filters have the properties of having no overshoot to a step function input while minimizing the rise and fall time. This behavior is closely connected to the fact that the Gaussian filter has the minimum possible group delay. It is considered the ideal time domain filter, just as the sinc is the ideal frequency domain filter.

The Gaussian function is for $x\varepsilon\,(-\infty,\ +\infty)$ and would theoretically require an infinite window length. However, since it decays rapidly, it is often reasonable to truncate the filter window and implement the filter directly for narrow windows, in effect by using a simple rectangular window function. In other cases, the truncation may introduce significant errors. Better results can be achieved by instead using a different window function; see scale space implementation for details.

The Gaussian filter function is: $$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\left(x^2+y^2\right)/\left(2\sigma^2\right)}$$



Algorithm

Depending on the choice in the trackbar system the code executes on the selected image, with the selected filter and cutoff frequency.
The Graphical User Interface is user-friendly and allows real-time changes in any of the three parameters, displaying the outputs automatically.

The code has structures and typedefs used to represent the various data types used.

The functions which carry out the task of filtering are described as below:

FFT::transform: Performs fast fourier transform

Input: Signal array *x*
Output: Array of fourier coefficients *X*

- `X_even = FFT(even indices of X)`
- `X_odd  = FFT(odd indices of X)`
- `for k = 0...N/2:`
    - $X[k] = X\_even[k] + e^{-2i*pi*k/n} * X\_odd[k]$
    - $X[N/2 + k] = X\_even[k] - e^{-2i*pi*k/n} * X\_odd[k]$

FFT::inverseTransform: Performs inverse fourier transform

Input: Array of fourier coefficients *X*
Output: Signal array *x*

- `Interchange real and imaginary part of X`
- `x = FFT::transform(X)`
- `Interchange real and imaginary part of x`
- `return x/N`

FFT::transform2d: Performs fast fourier transform of a 2-D array

Input: Signal matrix *x*
Output: Matrix of fourier coefficients *X*

- `for row i in x:`
    - `X.row(i) = FFT::transform(x.row(i))`
- `for column j in x:`
    - `X.col(j) = FFT::transform(x.col(j))`

FFT::inverseTransform2d: Performs inverse fast fourier transform of a 2-D array

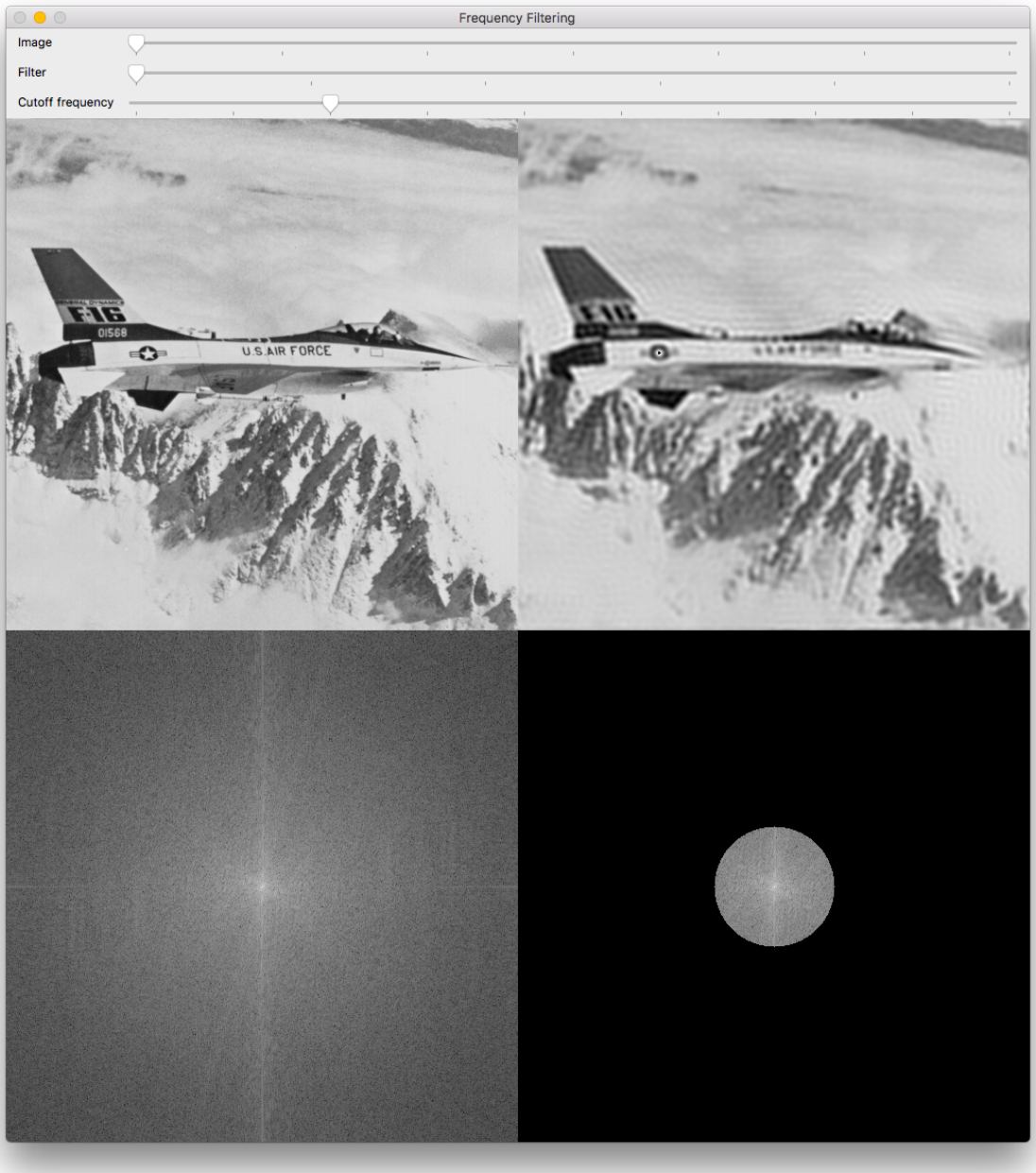
Output: Matrix of fourier coefficients *X*
Input: Signal matrix *x*

- `for row i in X:`
    - `x.row(i) = FFT::inverseTransform(X.row(i))`
- `for column j in X:`
    - `x.col(j) = FFT::inversetransform(X.col(j))`

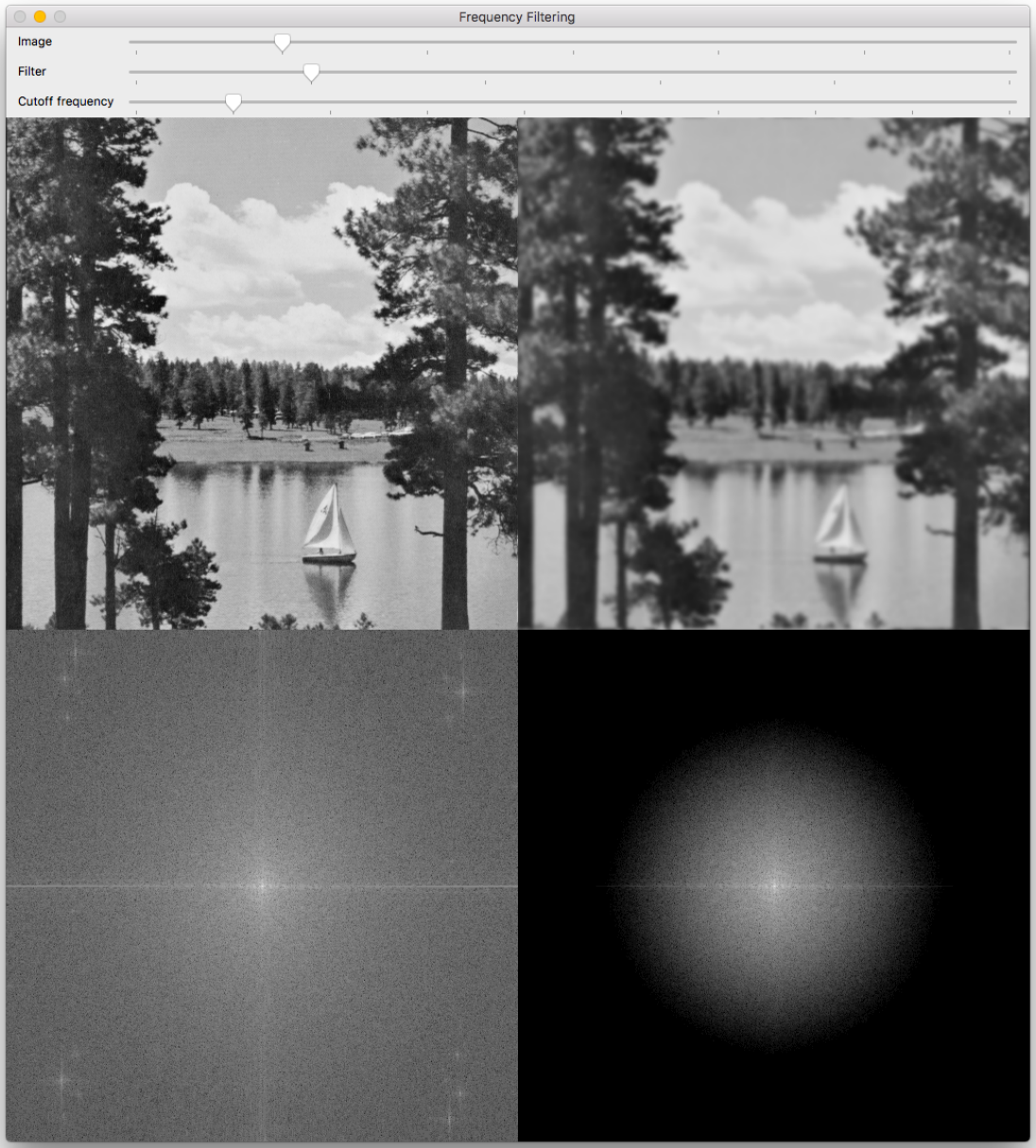
Results

Low Pass Filters:

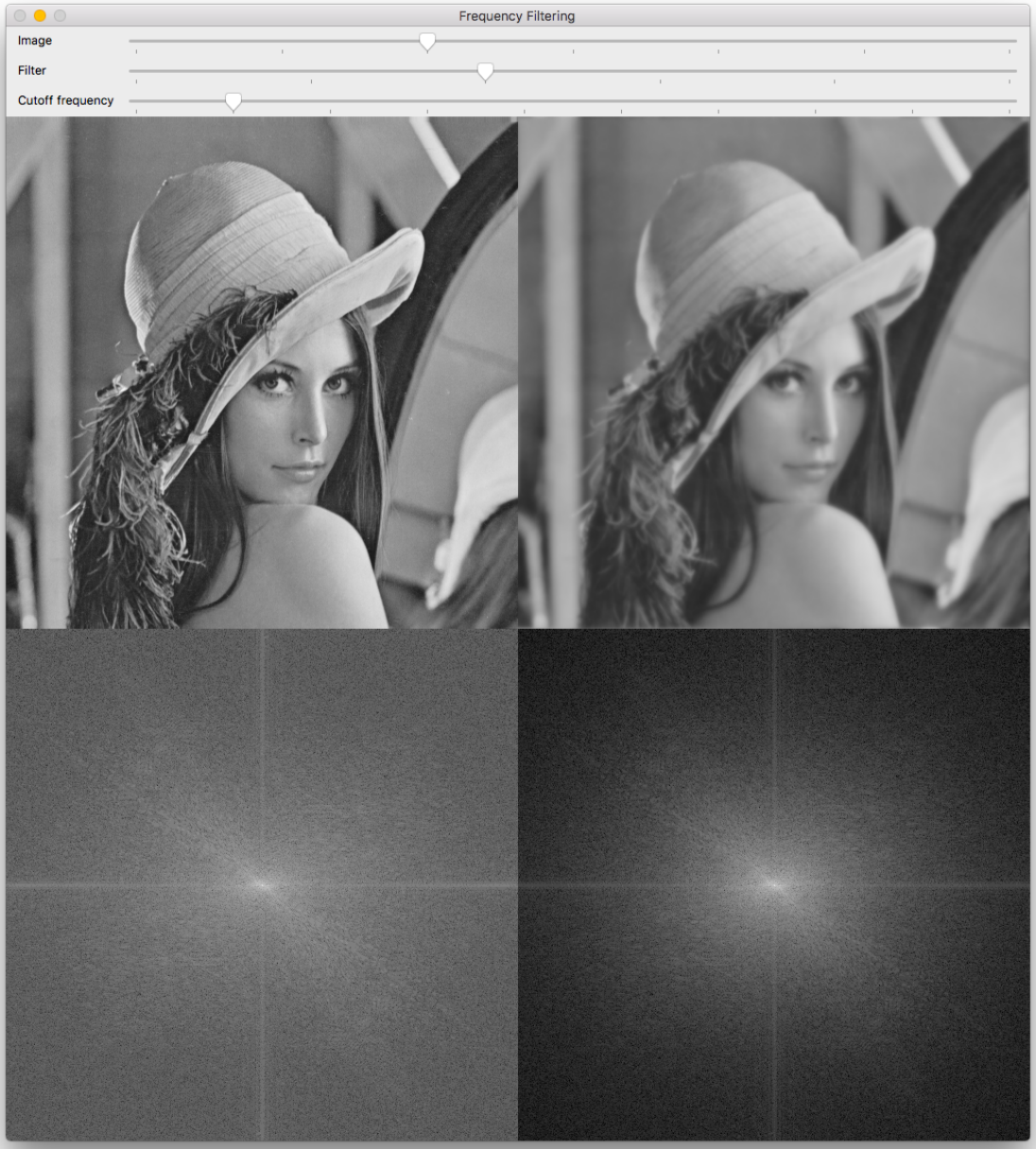
Ideal



Gaussian



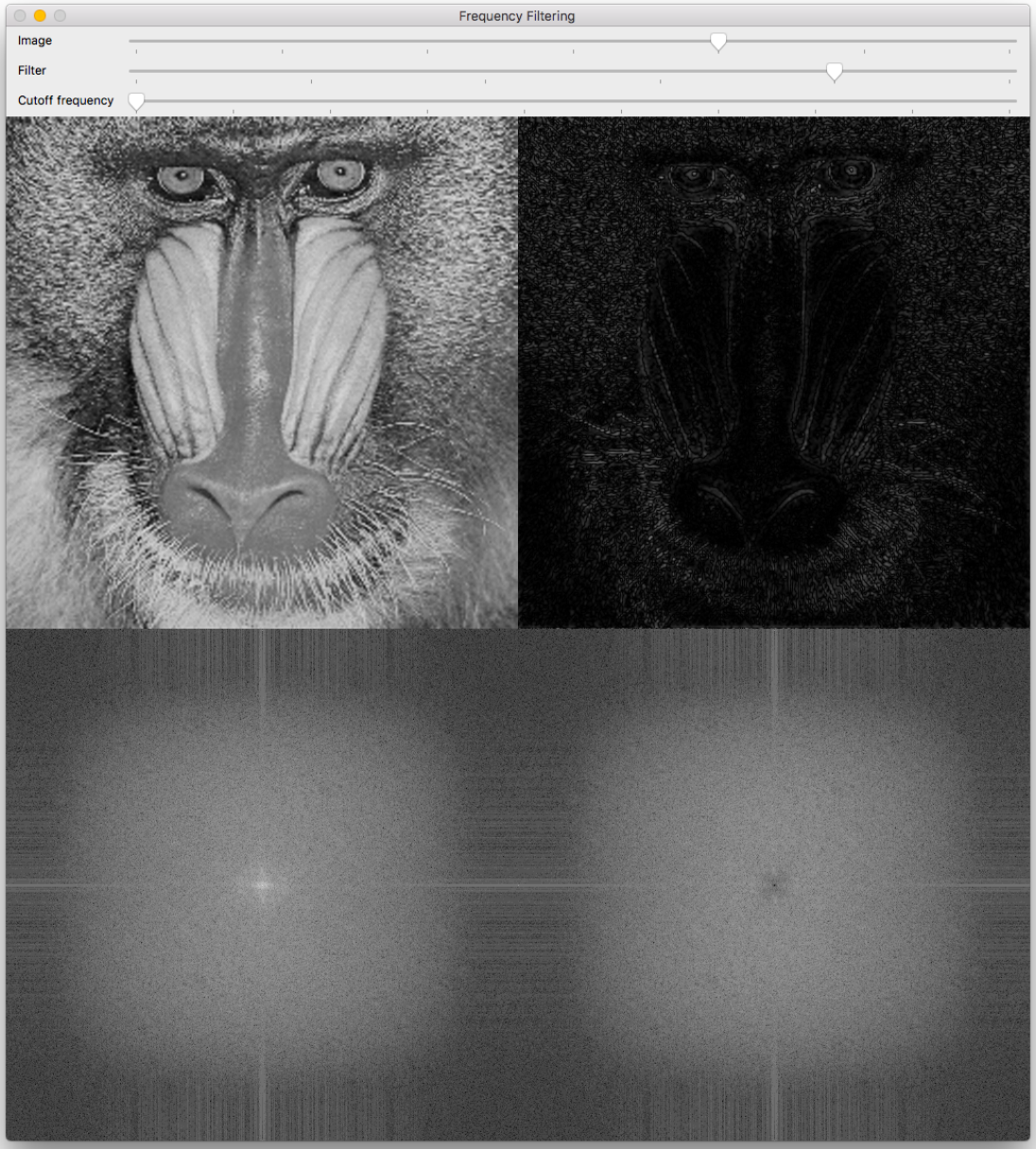
Butterworth



High Pass Filters:

Ideal

Gaussian



Butterworth

Analysis

Aditya Narayan [13EC35010]

1. Different low-pass and high-pass filters were realized and their performance were compared over an array of input images.

2. The ideal filters suffered from artifacts arising from the ringing effect. The ringing effect arises from the sharp cutoff of the ideal filter. Gaussian and butterworth filters do not suffer from the ringing effect.

3. The gaussian filter allows for smoother transition from the passband to the stopband. However, the passband transfer function is not constant. Thus, we incur a passband attenuation while using a Gaussian low-pass filter and a stopband leakage while using a gaussian high-pass filter.

4. The butterworth filter was designed to have a flat passband transfer function and a constant transition from passband to stopband. As the order of the butterworth filter increases, it resembles an ideal filter with a sharp cutoff. It also does not suffer from passband attenuation/leakage. Hence, it performs better edge detection than a gaussian filter.

5. The FFT implementation that was realized was the radix-2 Cooley-Tukey algorithm. It achieves an O(N logN) performance over an array of length N and $O(N^2 \log N)$ over a matrix of size N row * N columns. We can achieve a faster implementation by preferring an in-place variant of FFT over a recursive implementation.

Analysis

Adarsh Kumar Kosta

[13EC35008]

1. Frequency filtering is advantageous compared to spatial filtering in the sense that even large spatial kernels can be handled with the same complexity. The convolution operator is simplified to the multiplication operator which is easy to handle and implement, thereby reducing the time complexity in some cases.

2. The Ideal Low-pass and High-pass filters are not practically realizable due to hardware limitations. There is a ringing effect observed in case of these filters because of the sharp cutoff which produces artifacts in the image. This is referred to as Gibbs phenomenon and can be avoided by proper choice of windowing function.

3. The Butterworth low-pass filter has a maximally flat filter response in the low frequency region. The slope at which the filter function magnitude dies out can be controlled by the order. A 2nd order Butterworth filter is implemented in this experiment.

4. The Gaussian filter follows the Gaussian curve for filtering, the curve extending from $(-\infty, \infty)$ and the cutoff defined in terms of standard deviation form the central frequency (0).

5. Both Butterworth and Gaussian High-pass filters are used for edge detection in images. The butterworth filter produces better results compared to gaussian. The edge detected image can be used for image sharpening application by adding contrast to the edges.