Image Processing Lab3

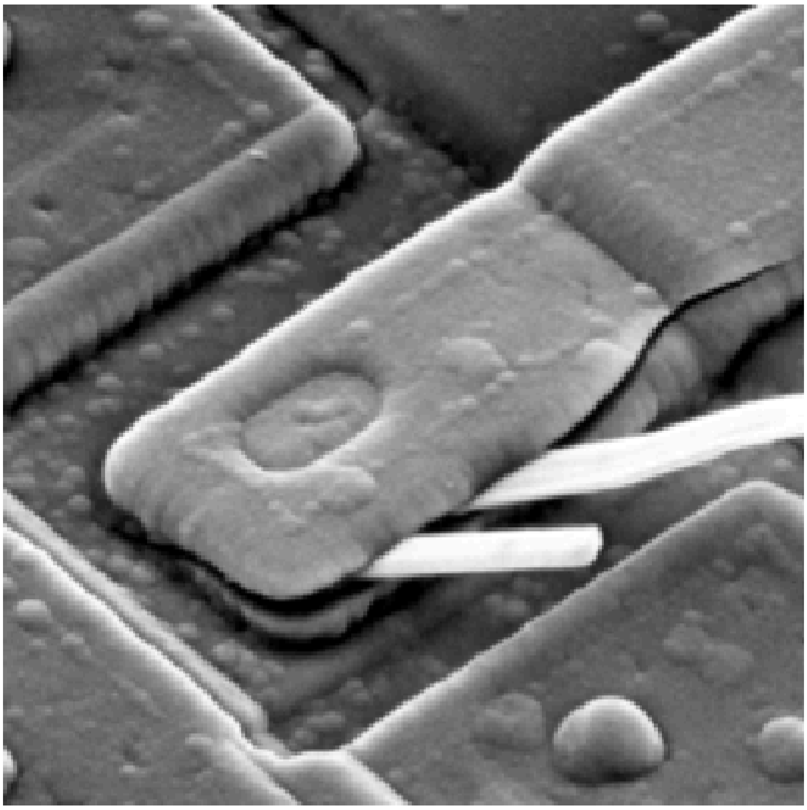
Name: 徐竣霆

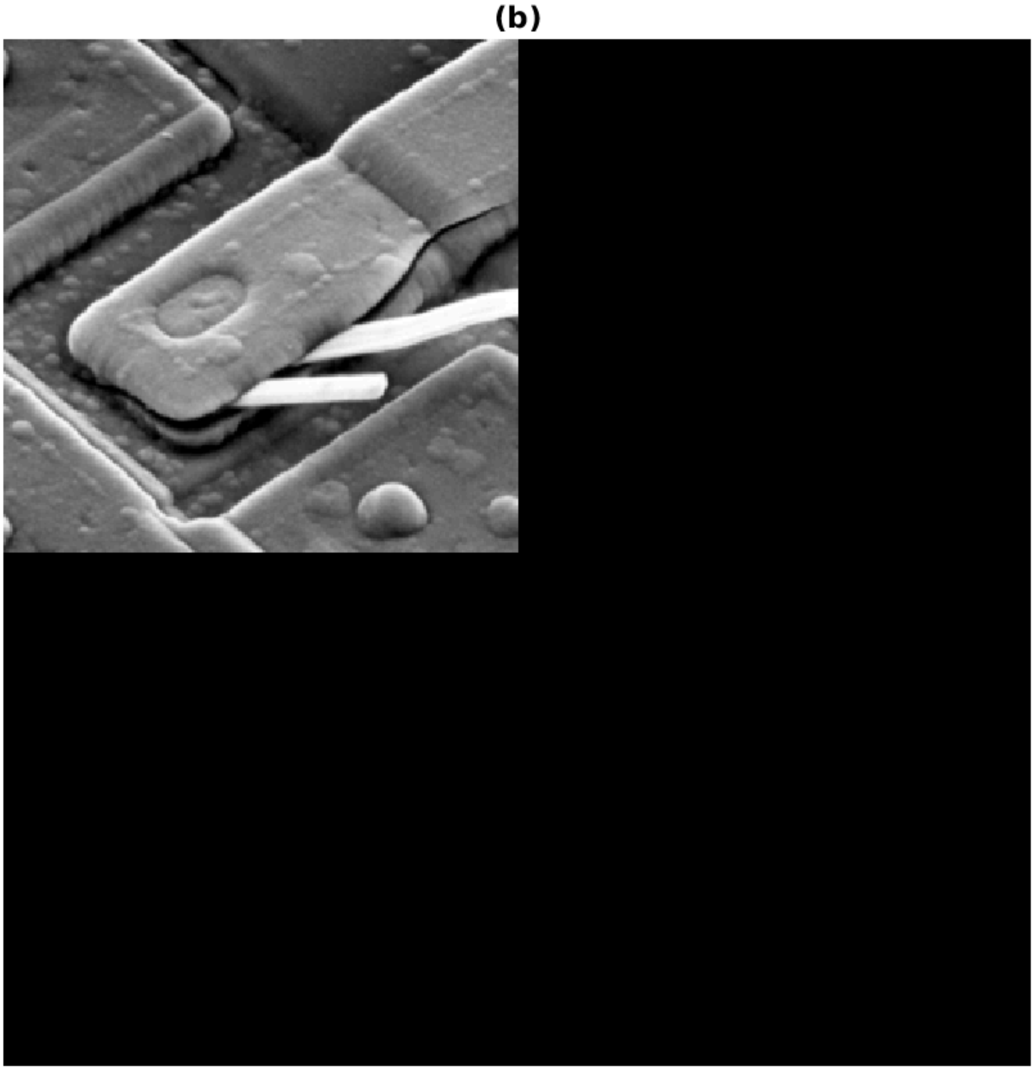
Student ID: 110060012

Proj04-01: Two-Dimensional Fast Fourier Transform

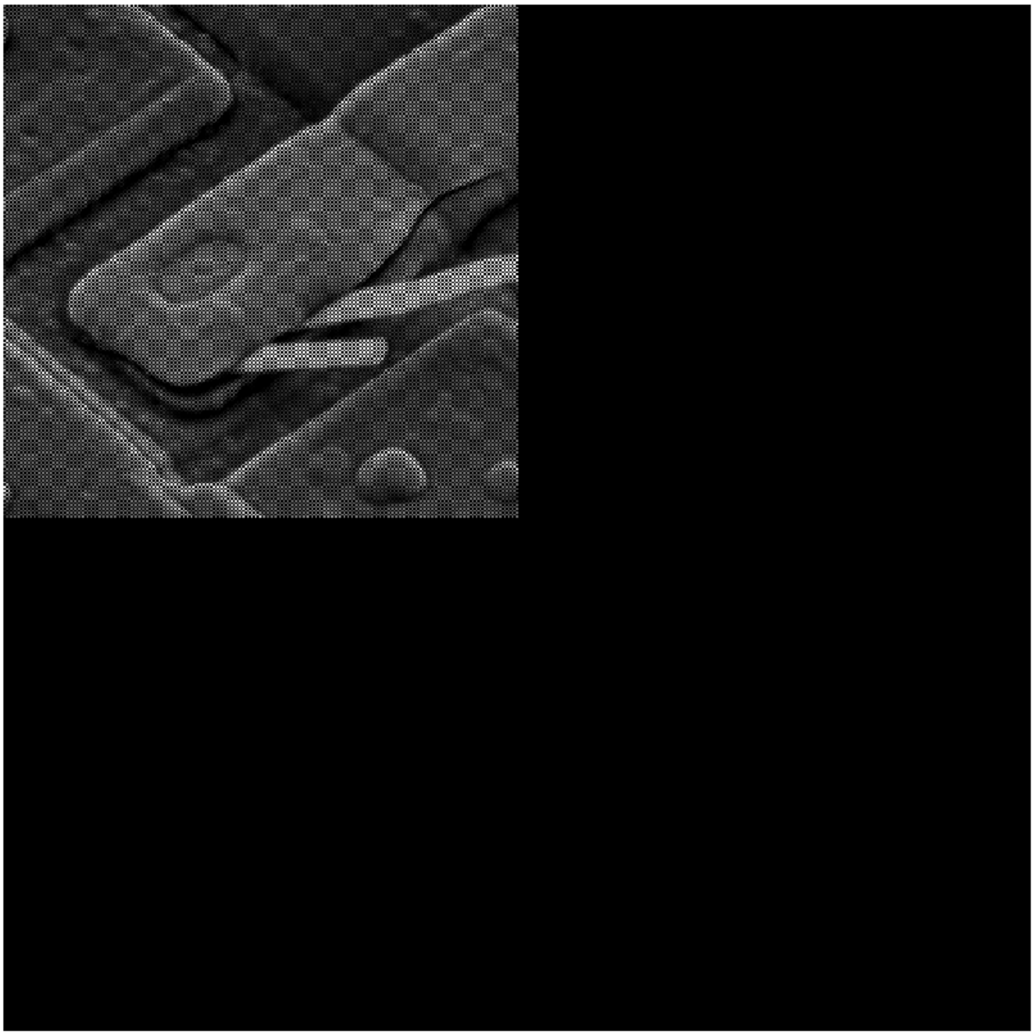
Results

(a)

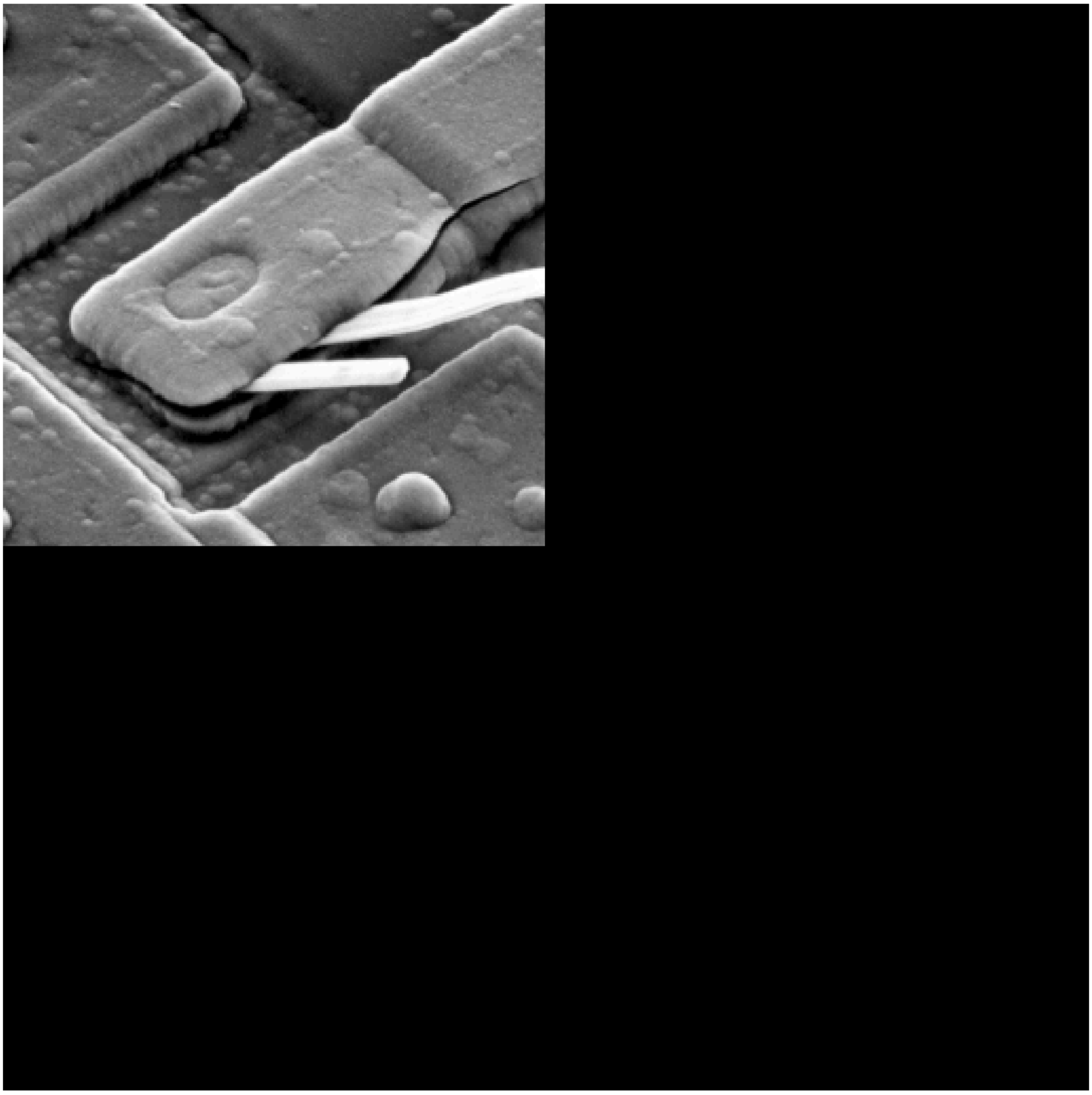



(b)

(c)

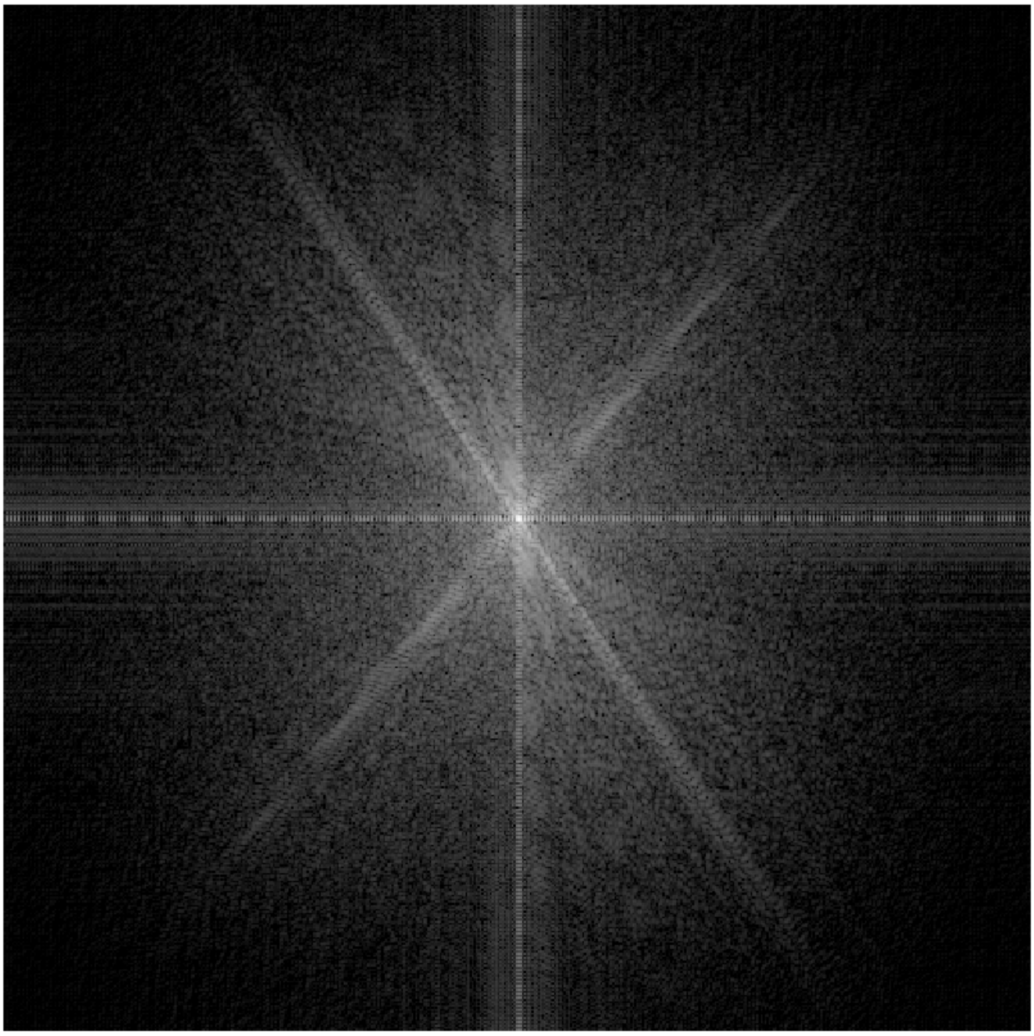


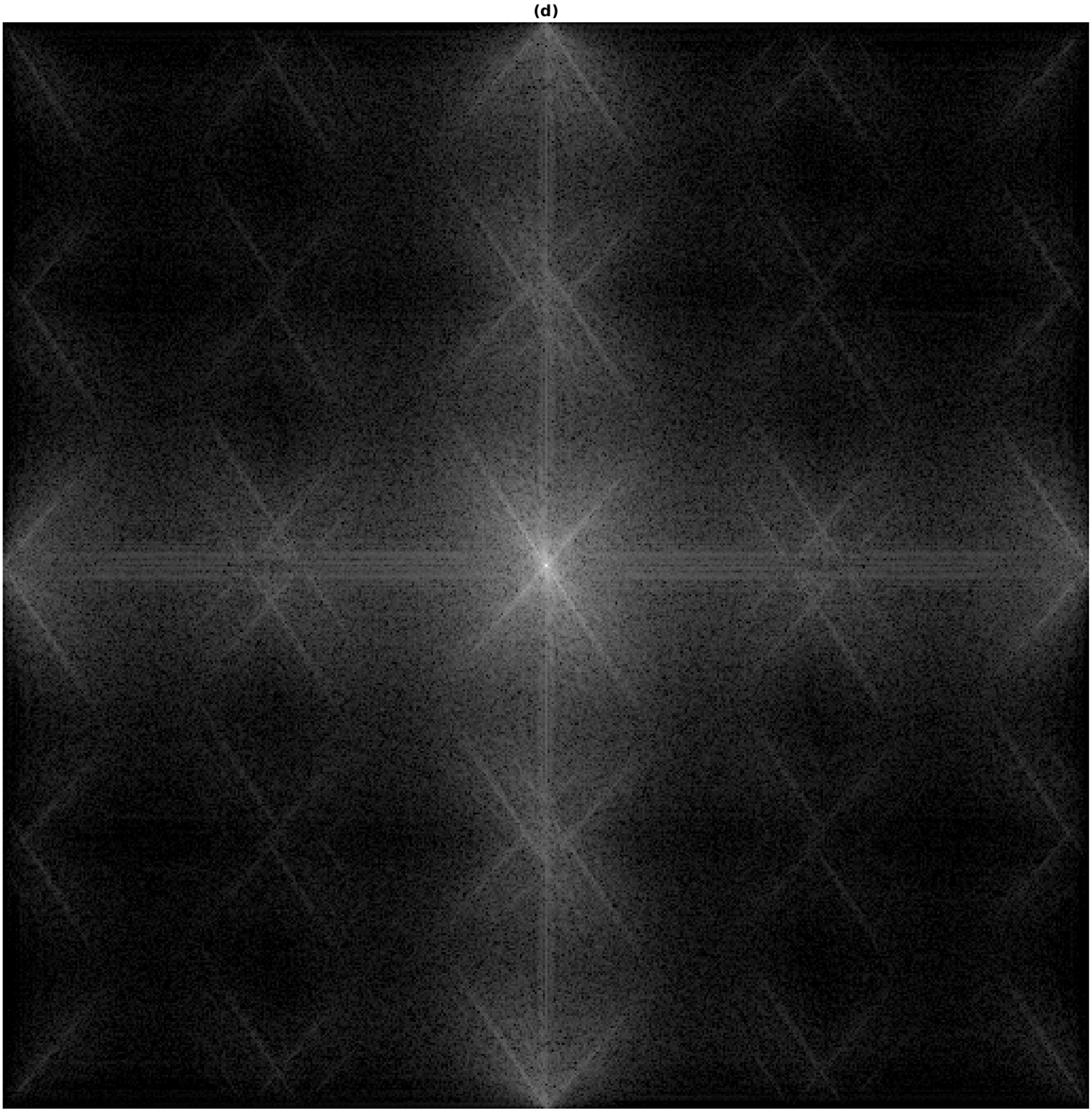
(c)



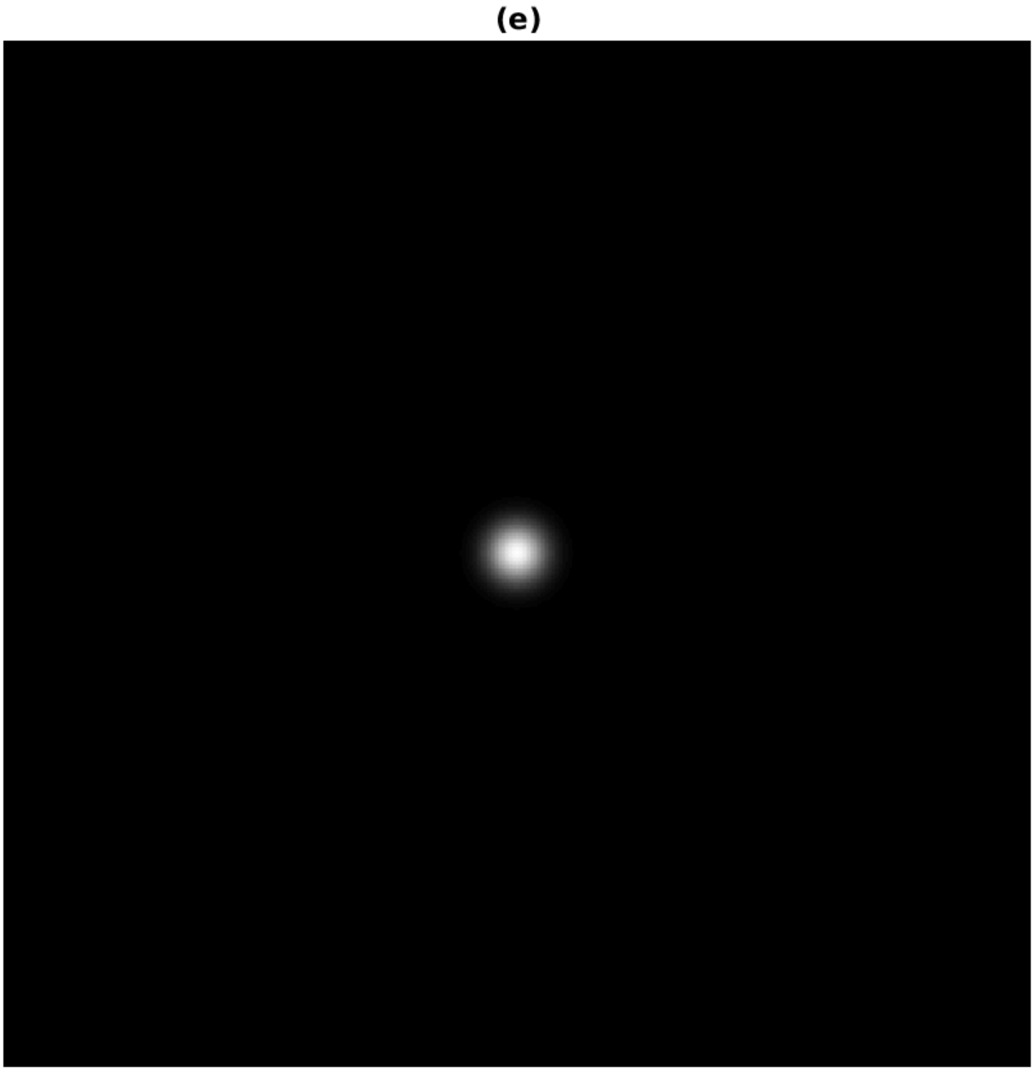
I just attach the result without `input = imresize(input, [256, 256]);` , since otherwise it will be different from Fig.4.35.

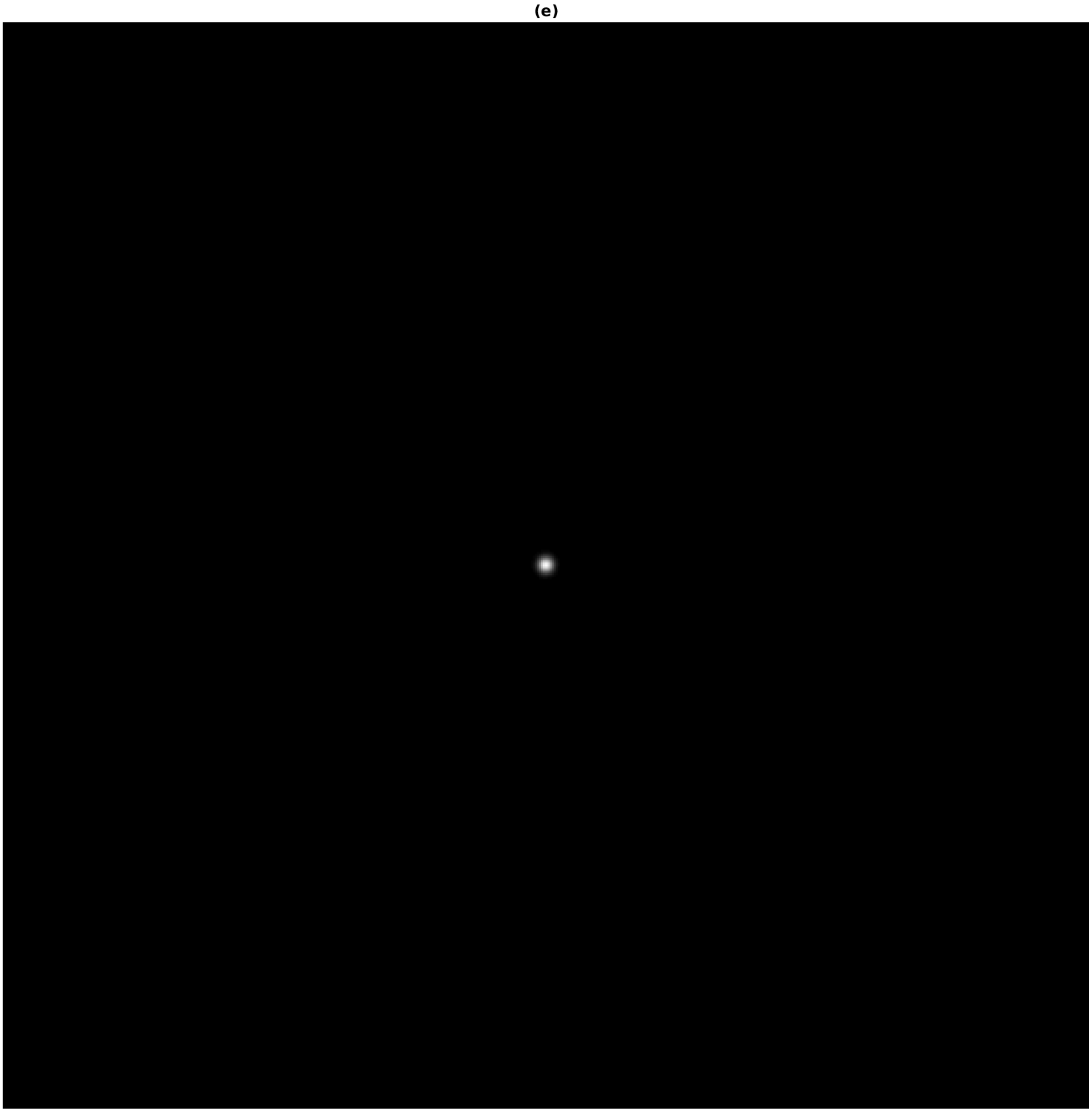
(d)



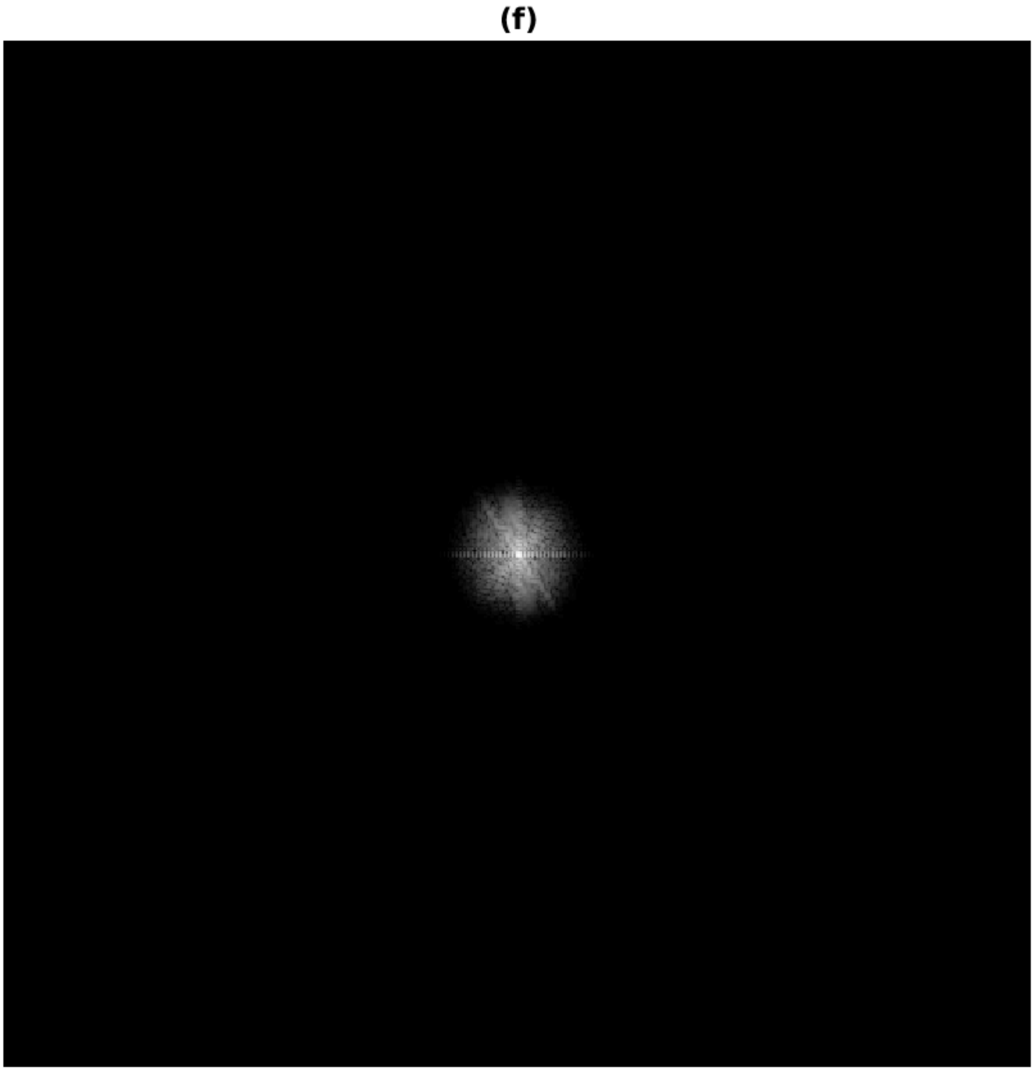


I just attach the result without `input = imresize(input, [256, 256]);` , since otherwise it will be different from Fig.4.35.

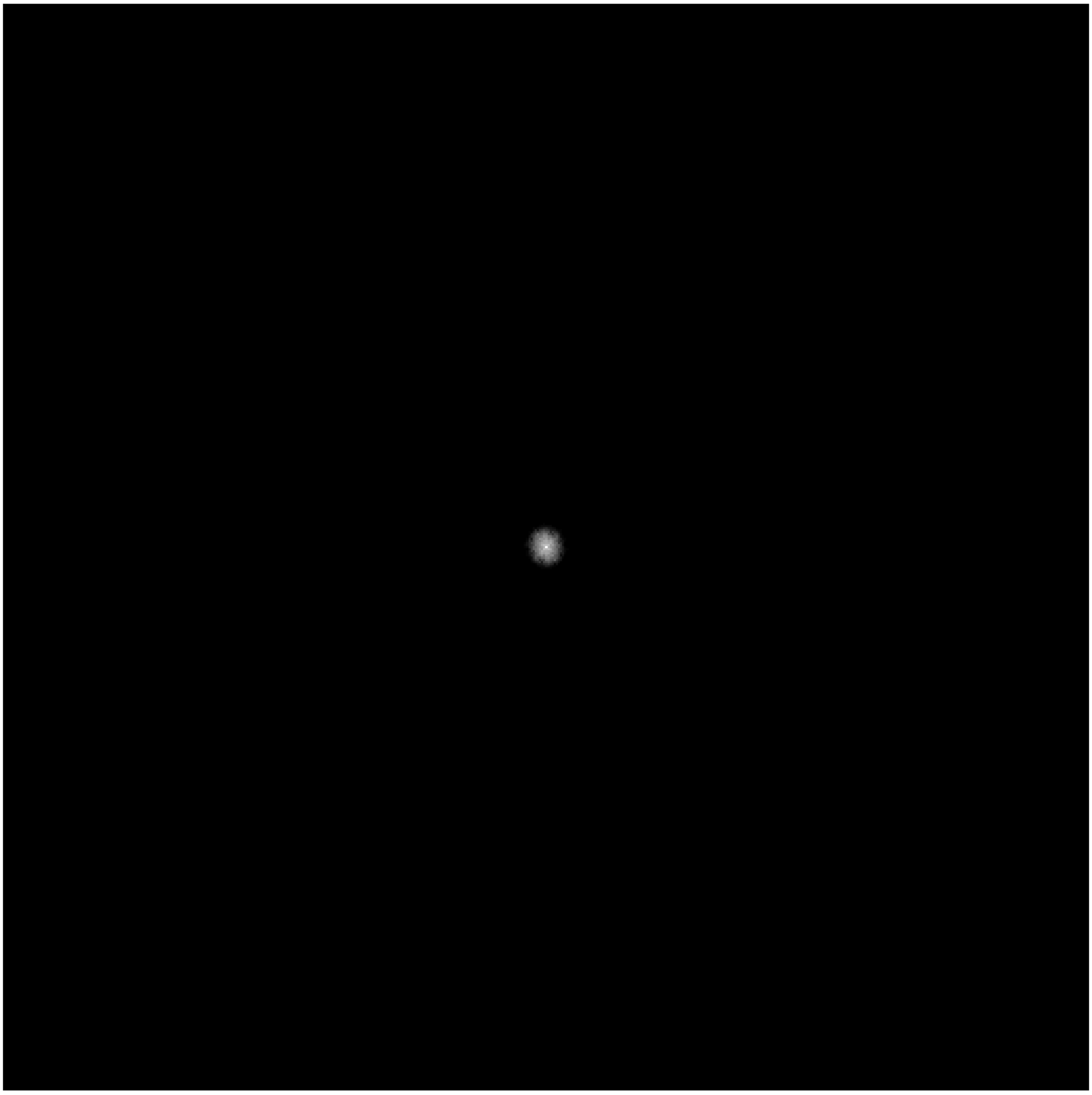

(e)



I just attach the result without `input = imresize(input, [256, 256]);` , since otherwise it will be different from Fig.4.35.

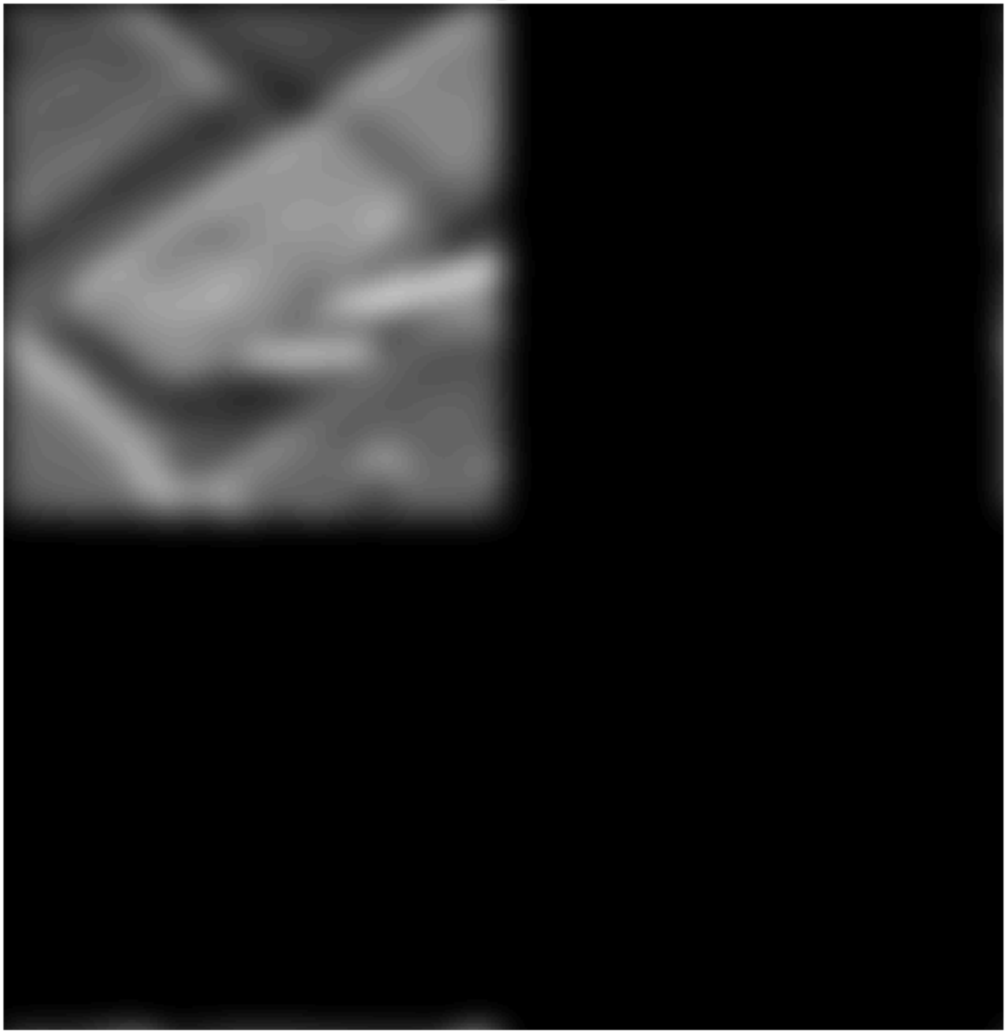

(f)

(f)

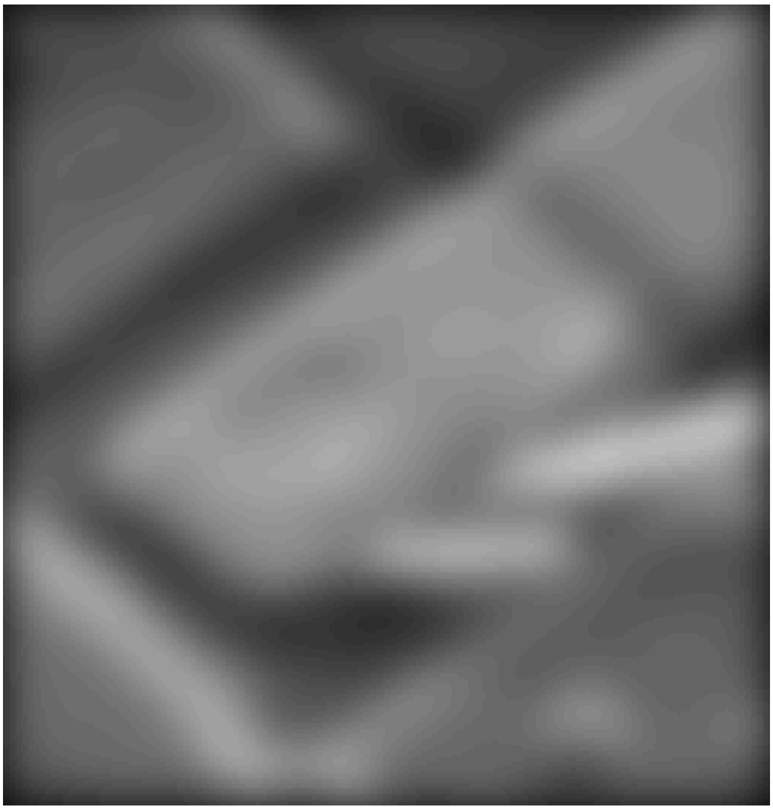


I just attach the result without `input = imresize(input, [256, 256]);` , since otherwise it will be different from Fig.4.35.

(g)



(h)



Implementation

myDFT2

```
function output = myDFT2(input)
    [M, N] = size(input);
    output = zeros(M, N);

    % using matrix multiplication
    u = 0:M - 1;
    v = 0:N - 1;
    W_row = exp(-1i * 2 * pi * (u') * u / M);
    W_col = exp(-1i * 2 * pi * (v') * v / N);
    output = W_row * input * W_col;
end
```

myIDFT2

```
function output = myIDFT2(input)
    [M, N] = size(input);
    output = zeros(M, N);

    % using matrix multiplication
    x = 0:M - 1;
    y = 0:N - 1;
    W_row = exp(1i * 2 * pi * (x') * x / M);
    W_col = exp(1i * 2 * pi * (y') * y / N);
    output = W_row * input * W_col / (M * N);
end
```

myFFT1

```
function output = myFFT1(input)
    n = length(input);

    % Ensure the input length is a power of 2
    if mod(n, 2) ~= 0 && n ~= 1
        error('Input length must be a power of 2');
    end

    % Base case
    if n == 1
        output = input;
        return;
    end

    % Initialize the output array of size n
    output = zeros(1, n);

    % Split into even and odd indexed elements
    X_even = myFFT1(input(1:2:end)); % FFT of even-indexed elements
    X_odd = myFFT1(input(2:2:end)); % FFT of odd-indexed elements

    % Combine the results
    Wn = exp(-2i * pi * (0:(n/2-1)) / n); % Twiddle factors
    output(1:n/2) = X_even + Wn .* X_odd; % First half
    output(n/2+1:n) = X_even - Wn .* X_odd; % Second half
end
```

myIFFT1

```
function output = myIFFT1(input)
    n = length(input);

    % Ensure the input length is a power of 2
    if mod(n, 2) ~= 0 && n ~= 1
        error('Input length must be a power of 2');
    end

    % Base case
    if n == 1
        output = input;
        return;
    end

    % Initialize the output array of size n
    output = zeros(1, n);

    % Split into even and odd indexed elements
    X_even = myIFFT1(input(1:2:end)); % IFFT of even-indexed elements
    X_odd = myIFFT1(input(2:2:end)); % IFFT of odd-indexed elements

    % Combine the results
    Wn = exp(2i * pi * (0:(n/2-1)) / n); % Twiddle factors
    output(1:n/2) = X_even + Wn .* X_odd; % First half
    output(n/2+1:n) = X_even - Wn .* X_odd; % Second half

    % Normalize the result for IFFT
    output = output / 2;
end
```

myFFT2

```
function output = myFFT2(input)

    [M, N] = size(input);

    % Apply 1D FFT to each row
    for row = 1:M
        input(row, :) = myFFT1(input(row, :));
    end

    % Apply 1D FFT to each column
    for col = 1:N
        input(:, col) = myFFT1(input(:, col));
    end

    output = input;
end
```

myIFFT2

```
function output = myIFFT2(input)

    [M, N] = size(input);

    % Apply 1D IFFT to each row
    for row = 1:M
        input(row, :) = myIFFT1(input(row, :));
    end

    % Apply 1D IFFT to each column
    for col = 1:N
        input(:, col) = myIFFT1(input(:, col));
    end

    output = input;
end
```

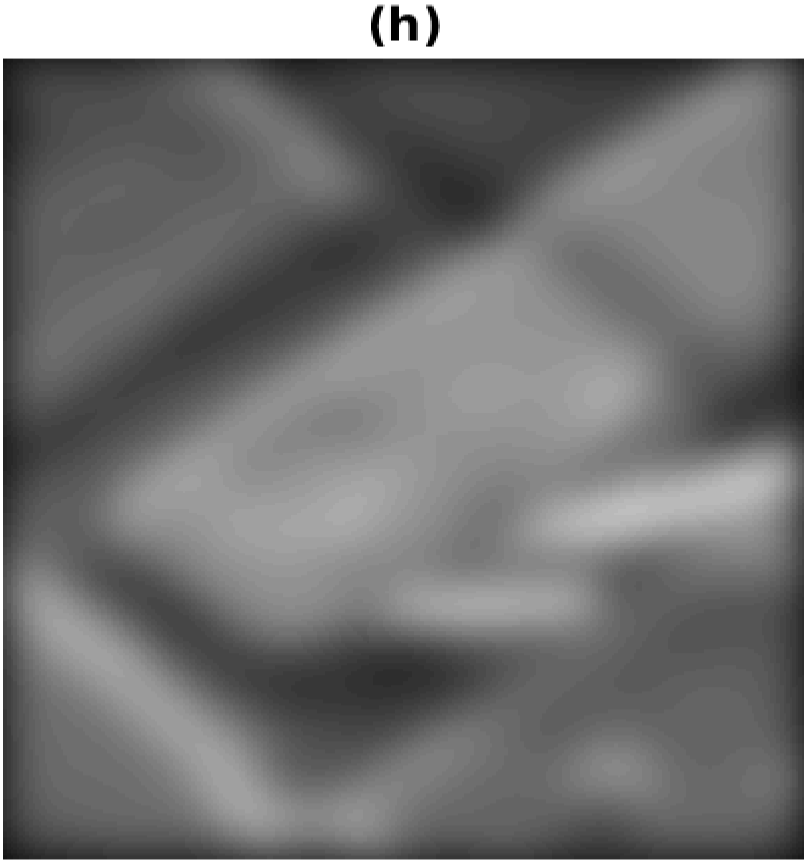
Discussion

Speed Up DFT

I choose Matrix Multiplication over nested for loops since it's significantly faster in MATLAB.(That's really cool)

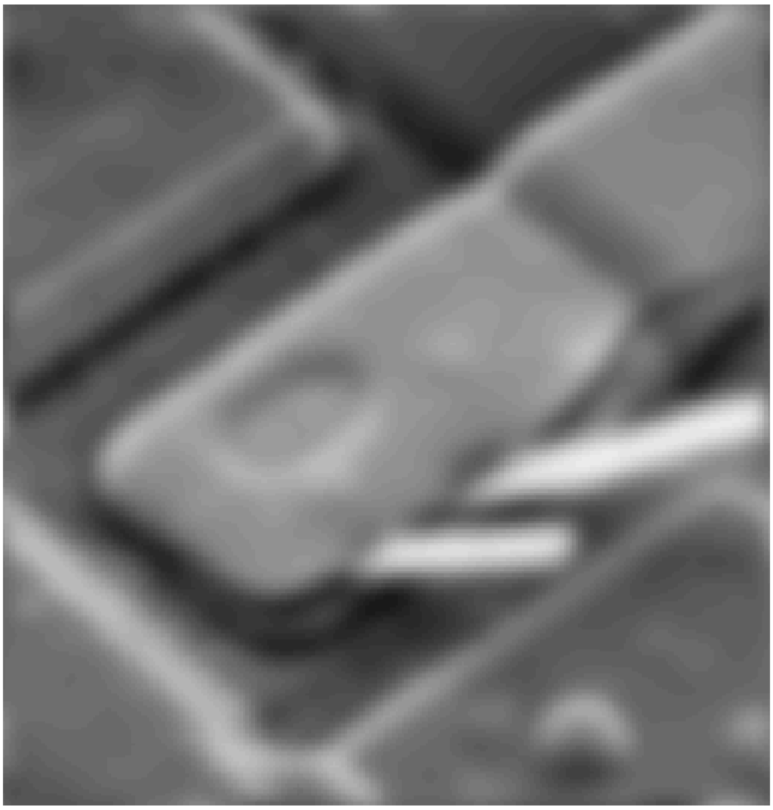
Padding versus No Padding

With padding = Linear convolution



Without padding = Circular convolution

(g)



The main difference is the black borders(Slide Chapter 4 Page 34).
Also, if the size of image is not 2^k, we have to perform padding so that we can utilize Fast Fourier Transform.

FFT

Let's first compare 1D DFT and FFT:
1D DFT:
It takes $O(N^2)$, based on the straightforward convolution operation.
1D FFT:
Due to a well structure of $N = 2^k$, the FFT leverages Divide and Conquer algorithm.
By dividing the term into two parts, namely the odd and even halves, recursively compute the FFT for each of them, then combine the result to get the FFT of the whole part. This contributes to an recursive algorithm with complexity $T(N) = 2T(\frac{N}{2}) + O(N)$, which can be reduced to $O(NlogN)$ by using the Master theorem or a recursion tree.

For 2D:
2D DFT:
We utilize matrix multiplication in 2D DFT, which takes $O(M^2N) + O(MN^2) = O(MN(M+N))$.

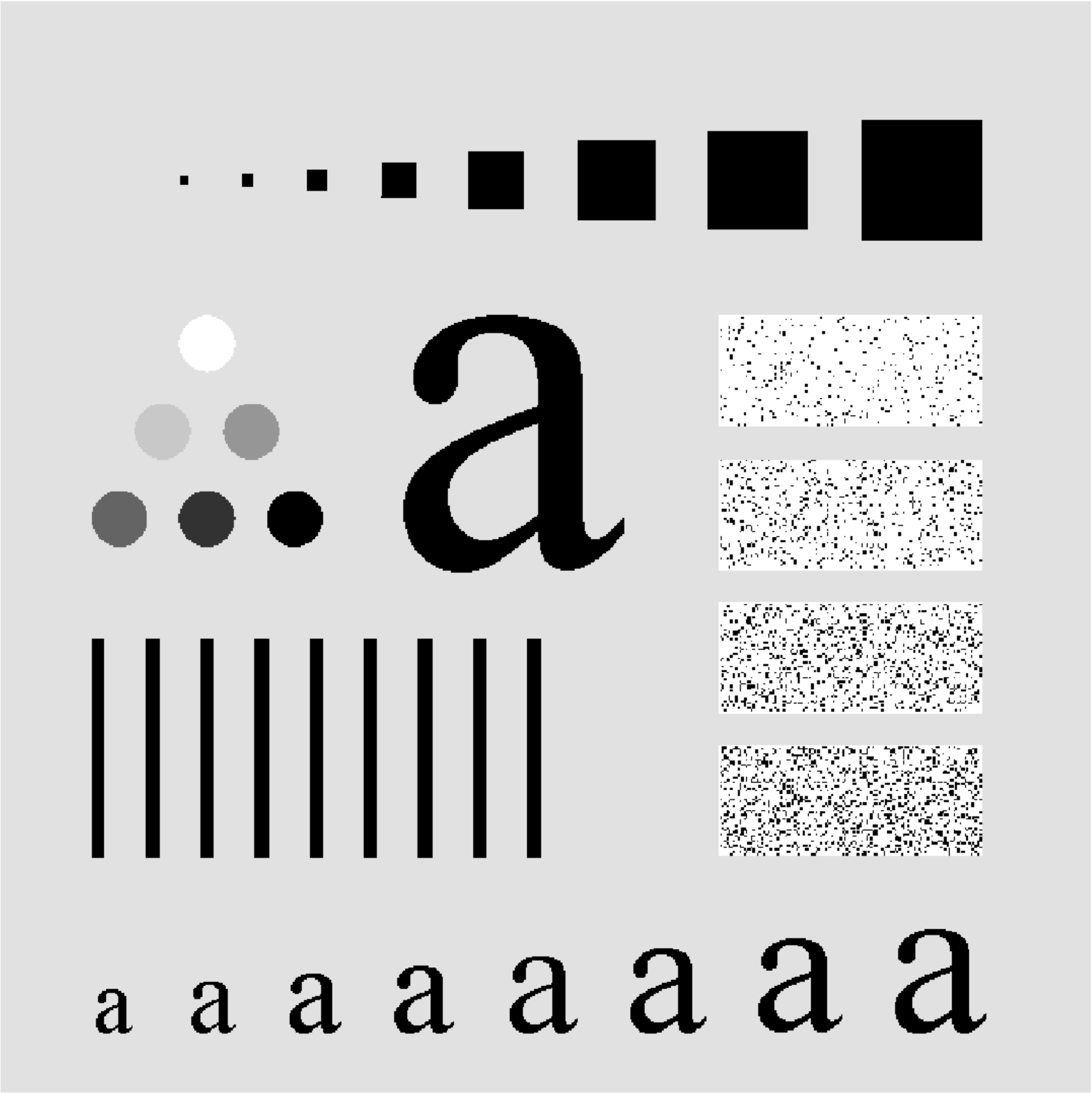
2D FFT:

For 2D FFT, we just apply 1D FFT to each row, and then each column, so the time complexity is $O(MNlogN) + O(NMlogM) = O(MN(logN + logM))$.

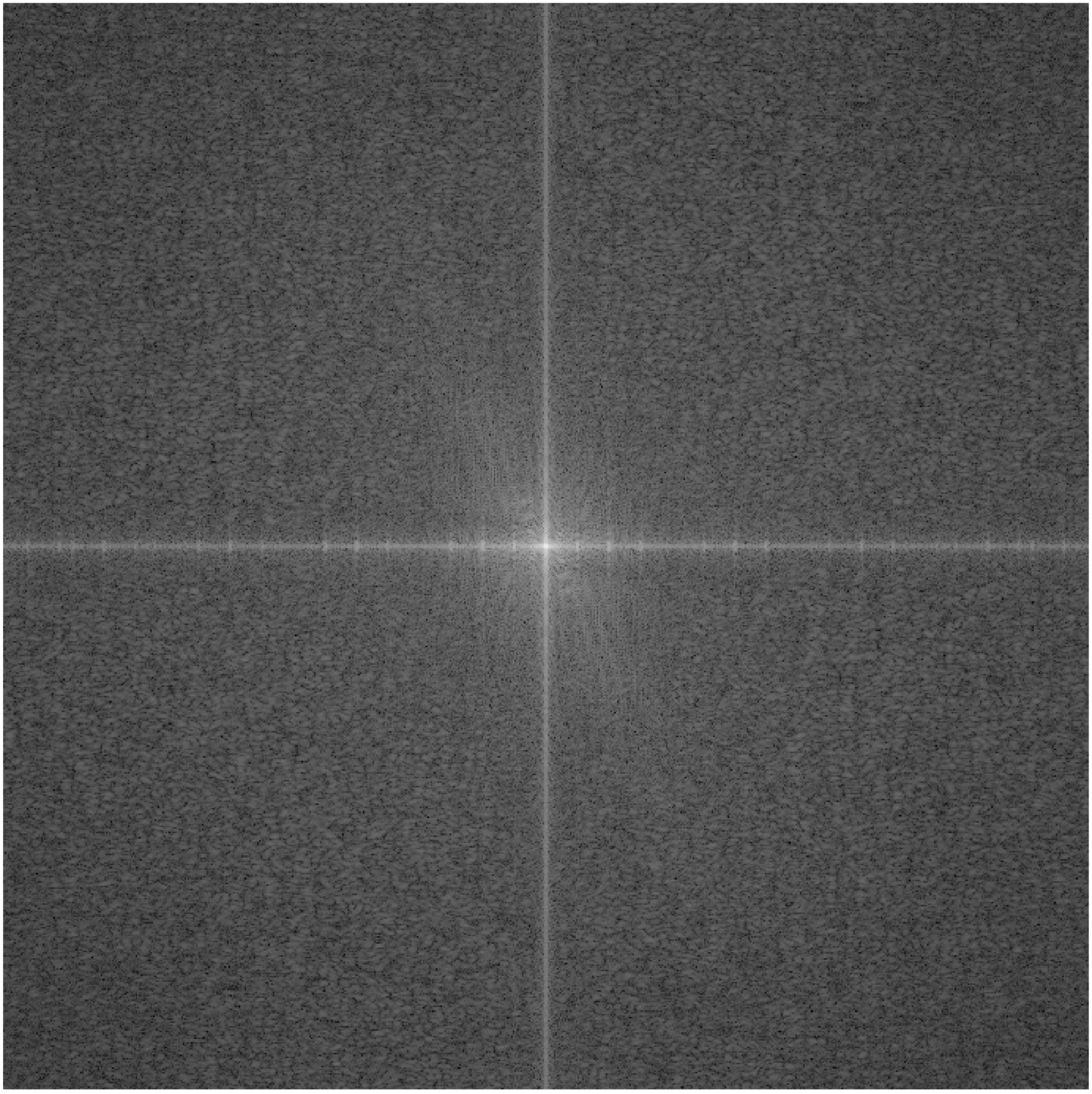
Proj04-02: Fourier Spectrum and Average Value

Results

Original Image



Fourier Spectrum



Implementation

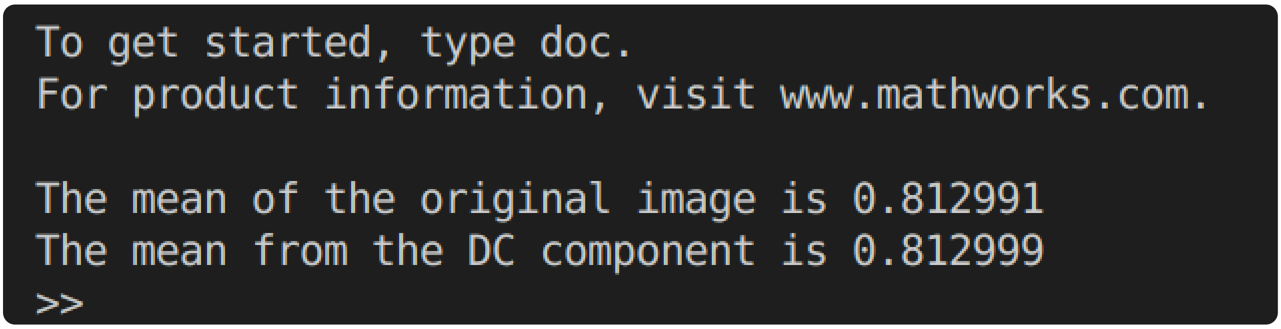
```
image_mean = mean(input(:));
```

I simply use this to get the mean from original image.

```
DC_mean = 4 * output(1 + M / 2, 1 + N / 2) / (M * N);
```

Get the DC Component after I perform fft2() and get get the Fourier spectrum. Then compute the mean.

Discussion



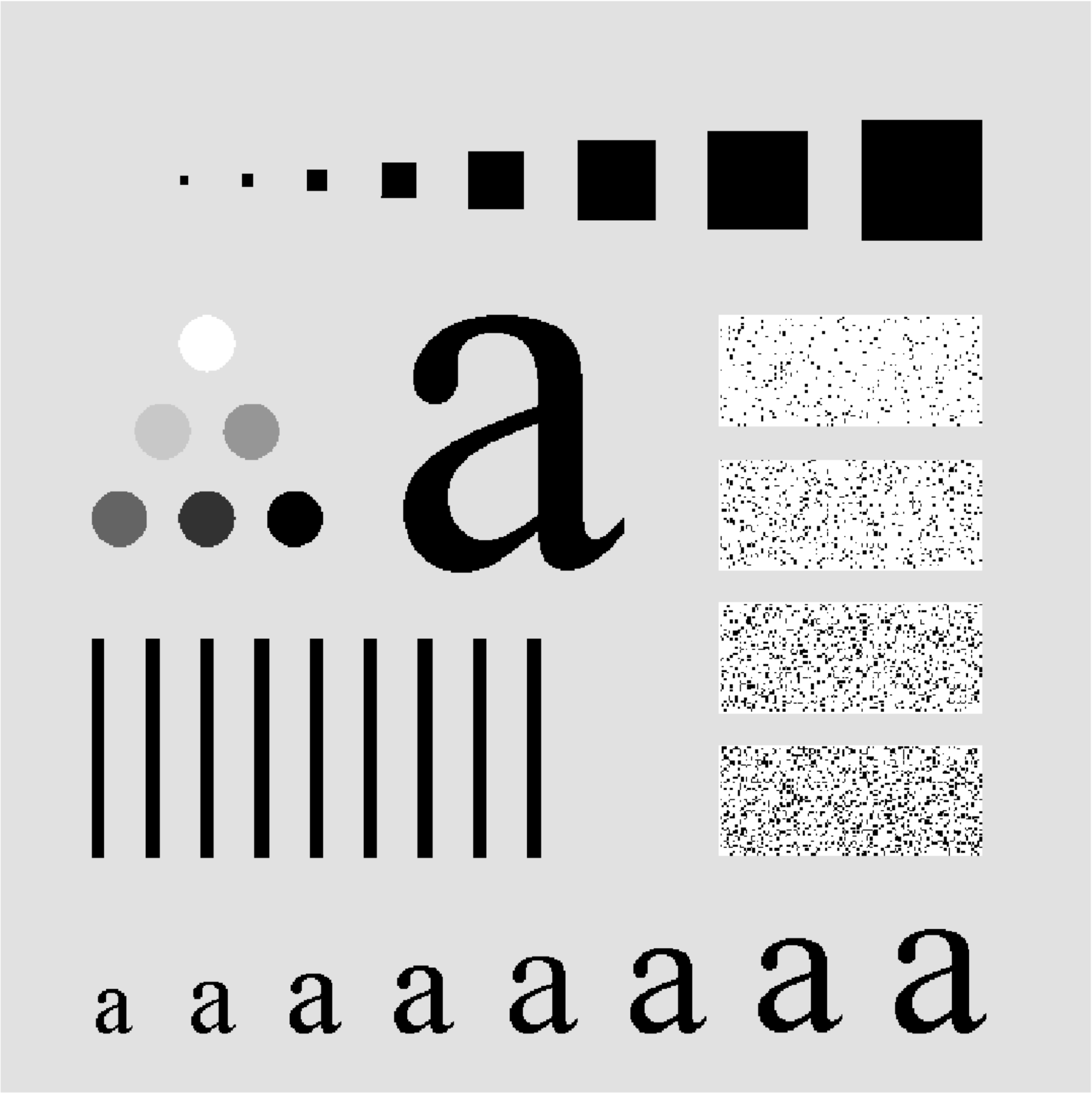
```
To get started, type doc.
For product information, visit www.mathworks.com.

The mean of the original image is 0.812991
The mean from the DC component is 0.812999
>>
```

The means computed by different methods is quite close.

Proj04-03: Lowpass Filtering

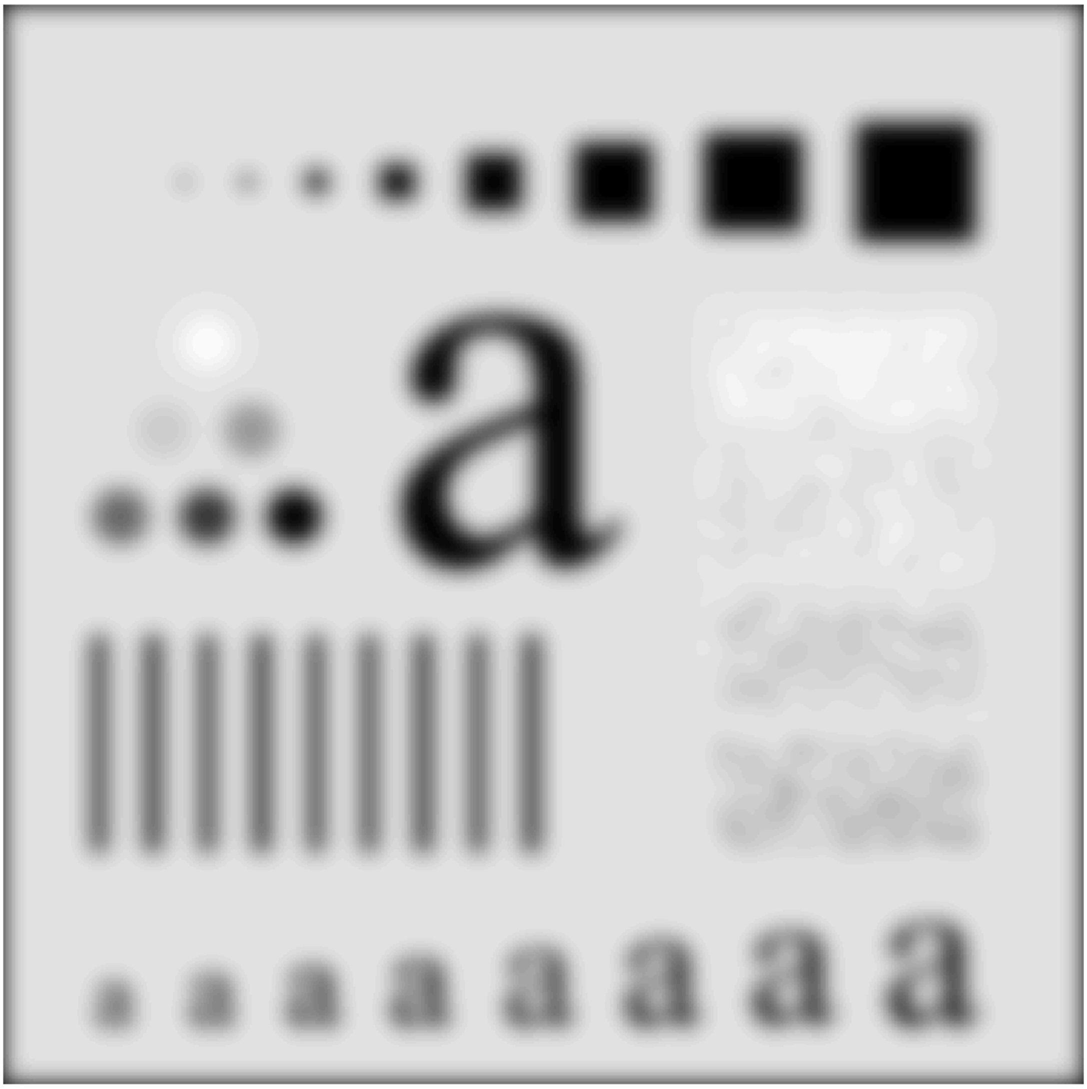
Results

(a)

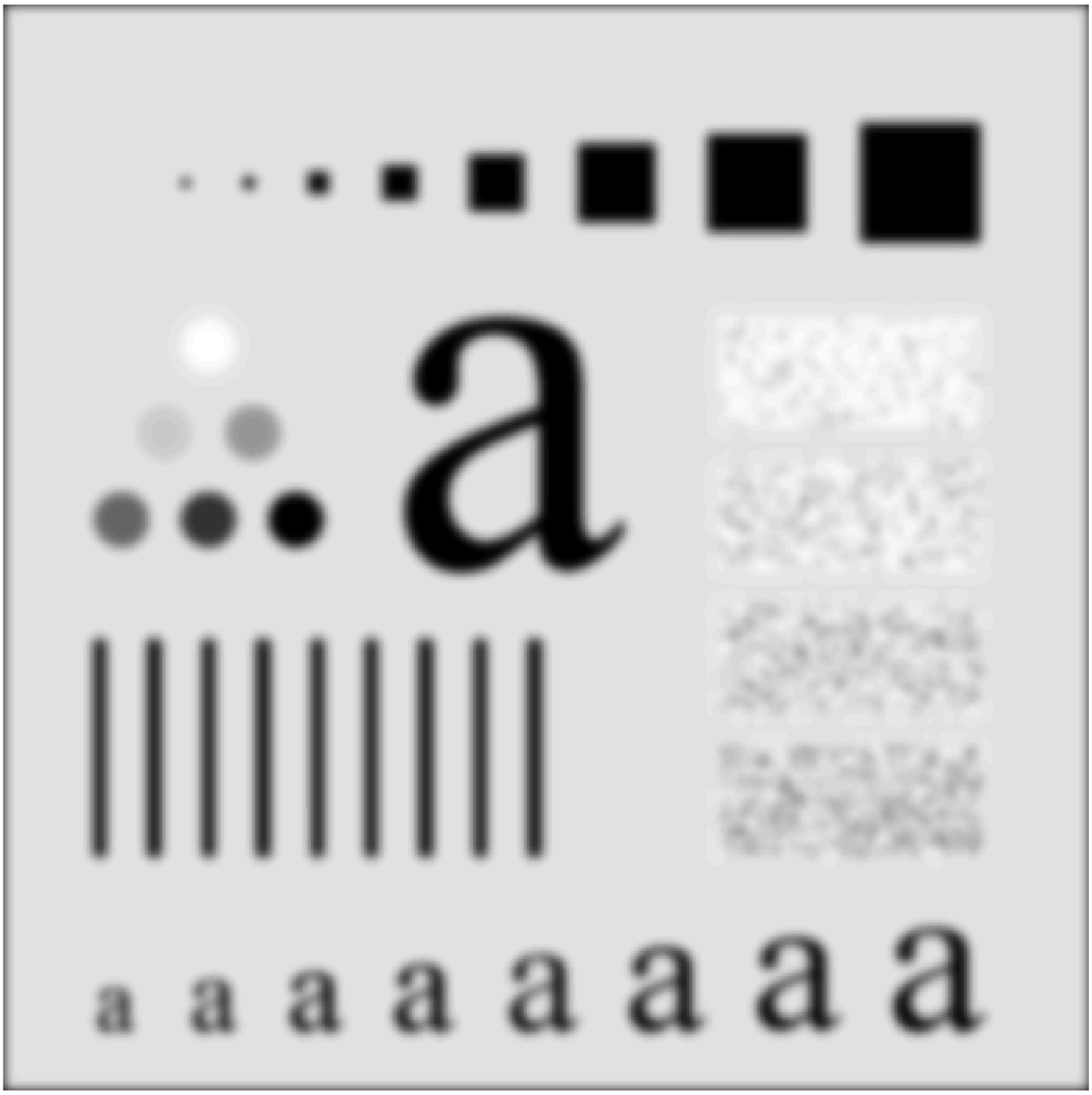


(b)

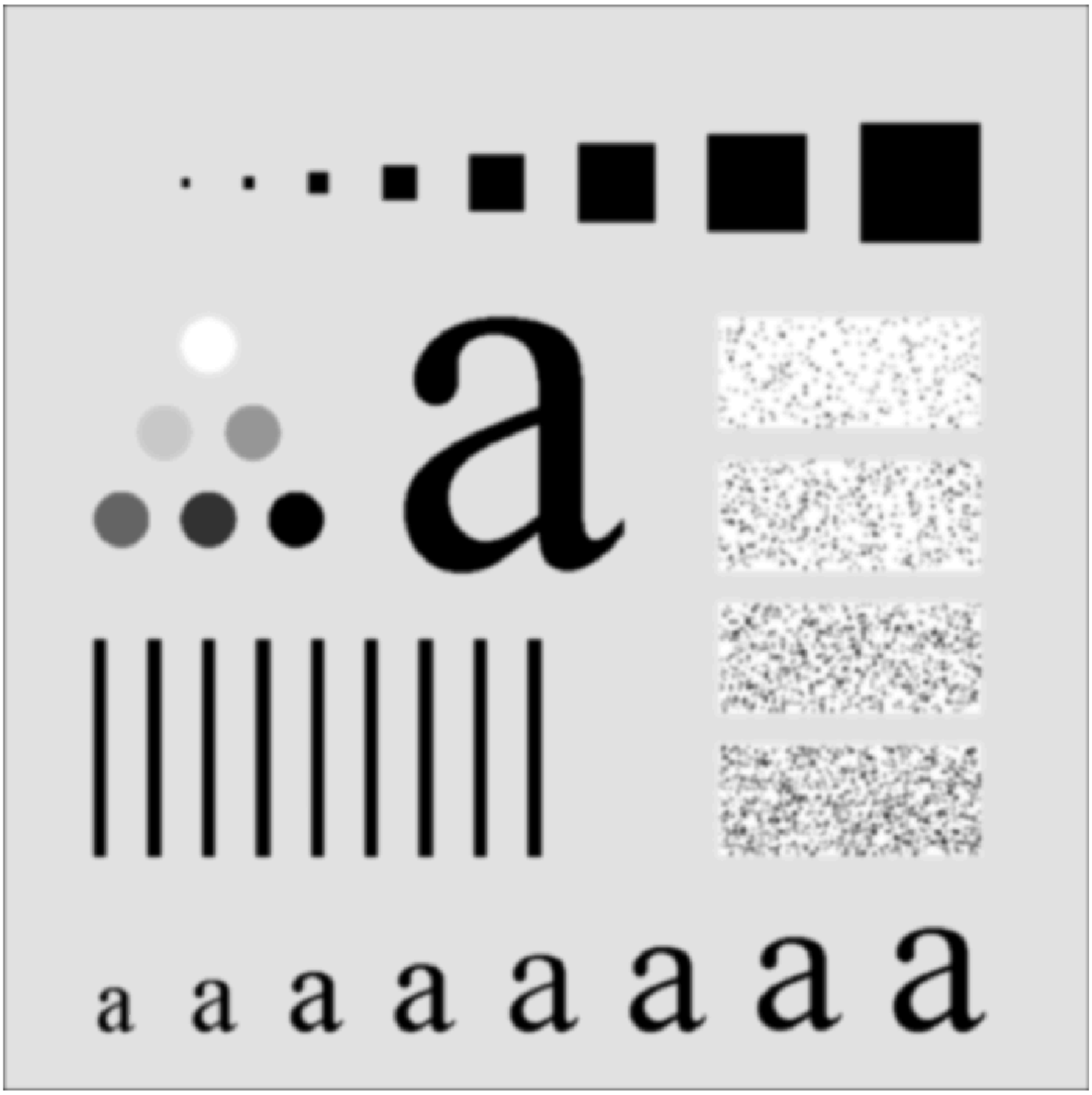
(c)



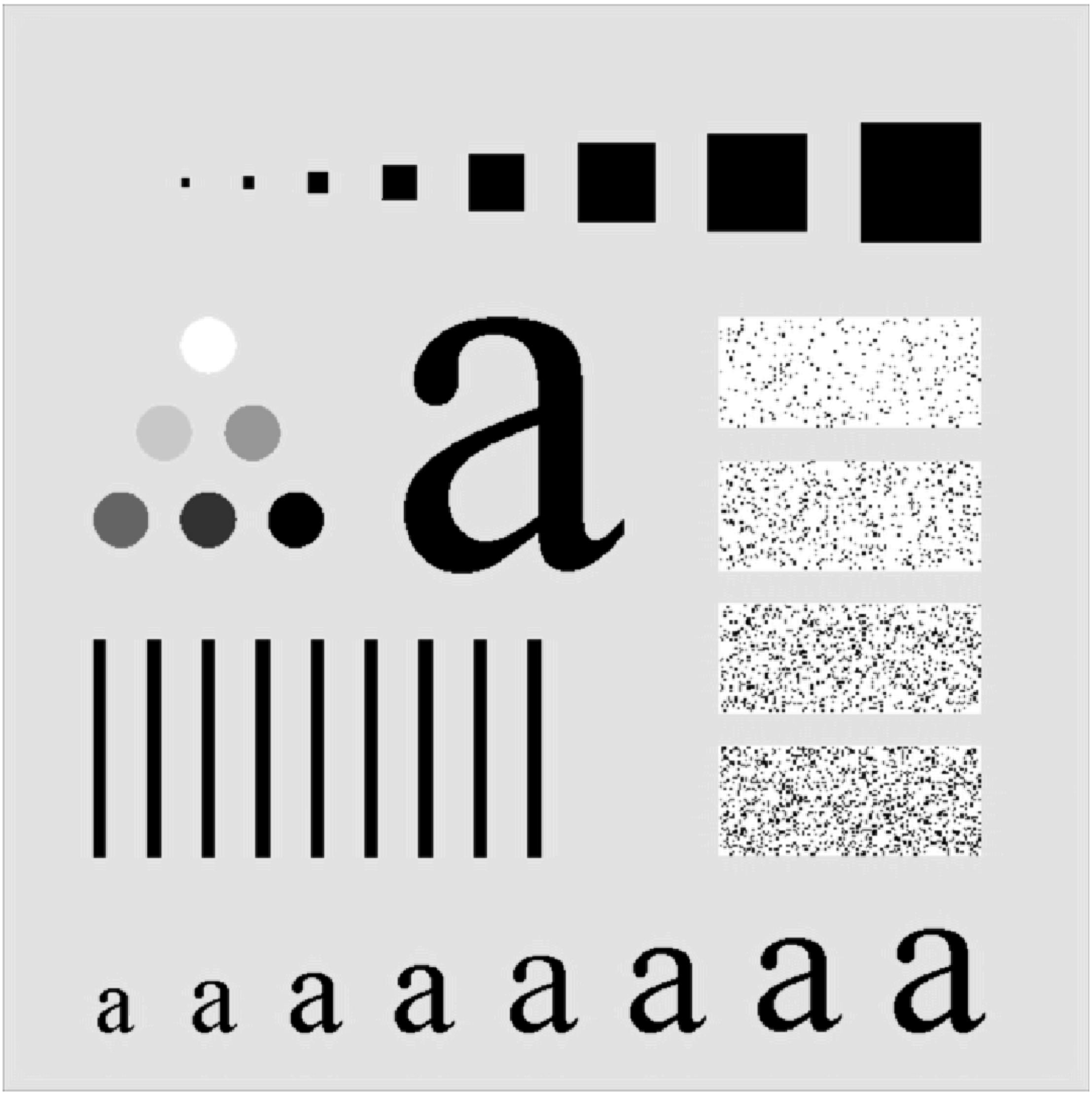
(d)



(e)



(f)



Implementation

myGLPF

```
function output = myGLPF(D0, M, N)
    output = single(zeros(M, N));
    for u = 1:M
        for v = 1:N
            D = sqrt((u - M / 2)^2 + (v - N / 2)^2);
            output(u, v) = exp(-D^2 / (2 * D0^2));
        end
    end
end
```

From Slide - Chapter 4 - Page 51.

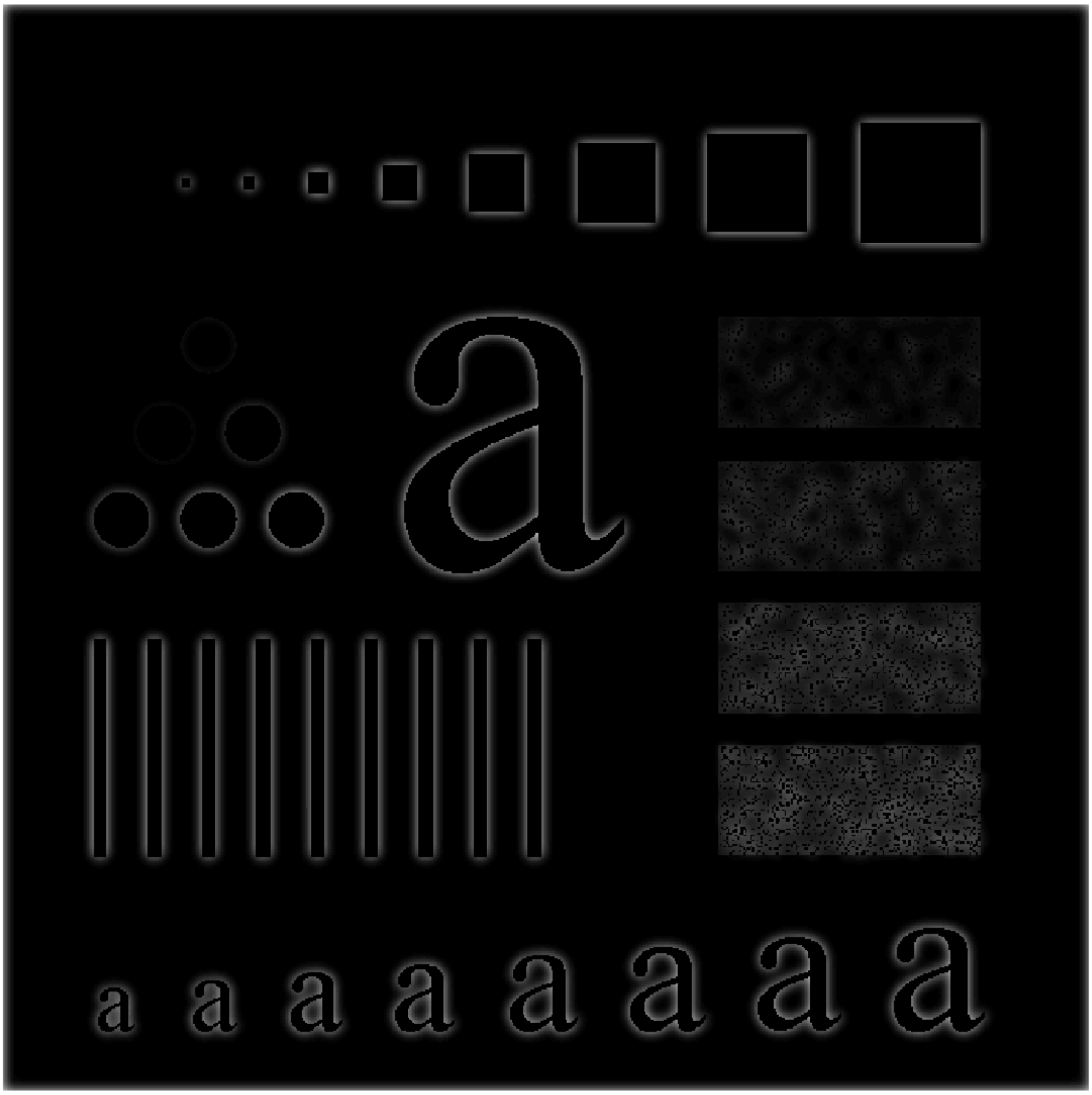
Discussion

It's still quite different from the Fig4.44, I suppose it's because the original image is different? As we can see, as $D0$ get larger, the image become less blurred.

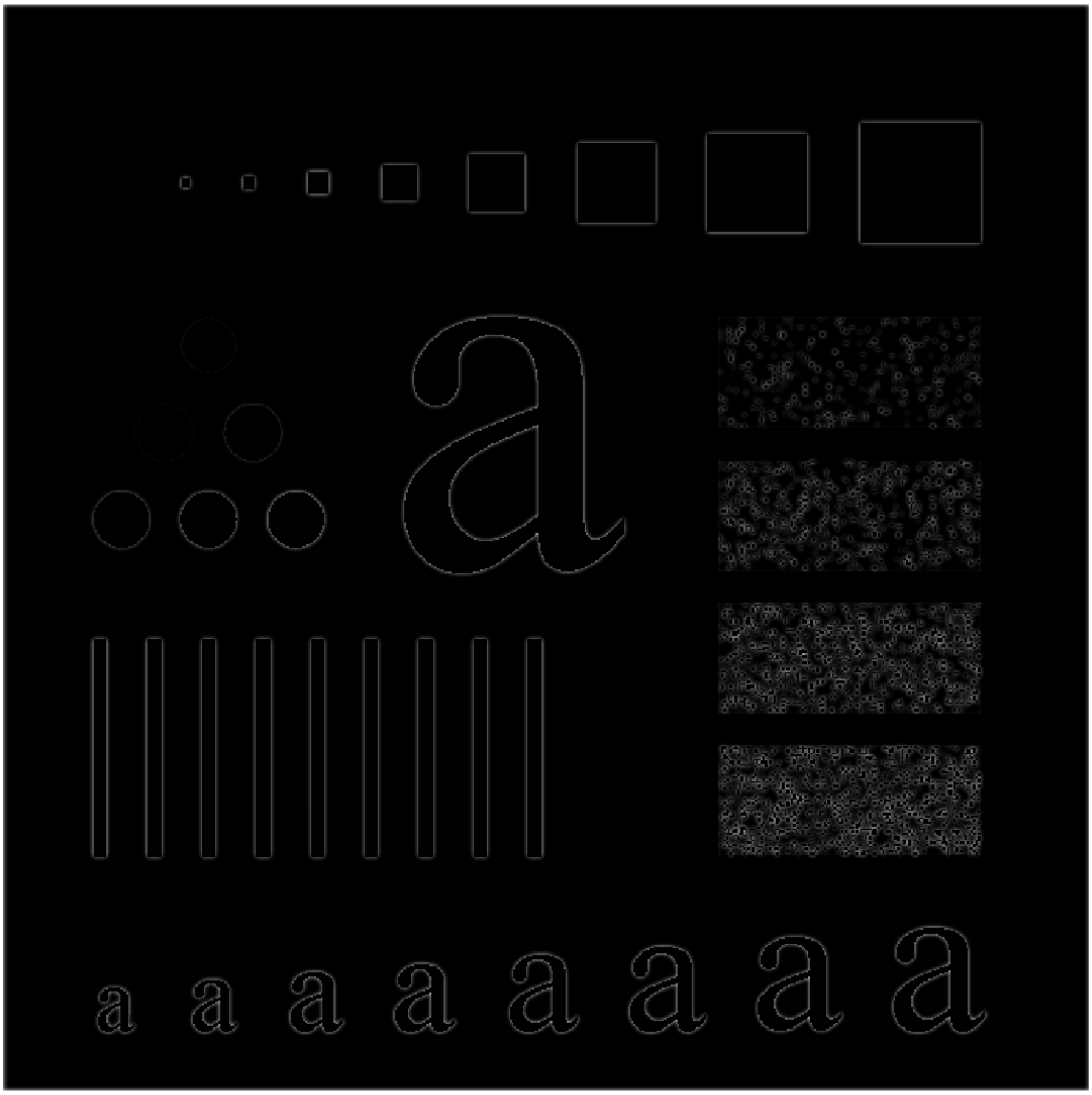
Proj04-04: Highpass Filtering

Results

(b)



(e)



Implementation

myGHPF

```
function output = myGHPF(D0, M, N)
    output = single(zeros(M, N));
    for u = 1:M
        for v = 1:N
            D = sqrt((u - M / 2)^2 + (v - N / 2)^2);
            output(u, v) = 1 - exp(-D^2 / (2 * D0^2));
        end
    end
end
```

From the idea of myGLPF.

Discussion

It's still quite different from the Fig4.53, I suppose it's because the original image is different? As we can see, the background is black, since high frequency usually focus on detailed parts, such as boundaries.