

## UE19CS333 - IPCV Lab 3b and Lab 4: Gradient Image + Frequency domain processing

Team No.: 3

### Name and SRN of Members of the Team:

- |                       |                 |
|-----------------------|-----------------|
| 1. Abhishek Aditya BS | - PES1UG19CS019 |
| 2. T Vijay Prashant   | - PES1UG19CS536 |
| 3. Vishal R           | - PES1UG19CS571 |
| 4. Yashas KS          | - PES1UG19CS589 |

\*\*\*\*\*

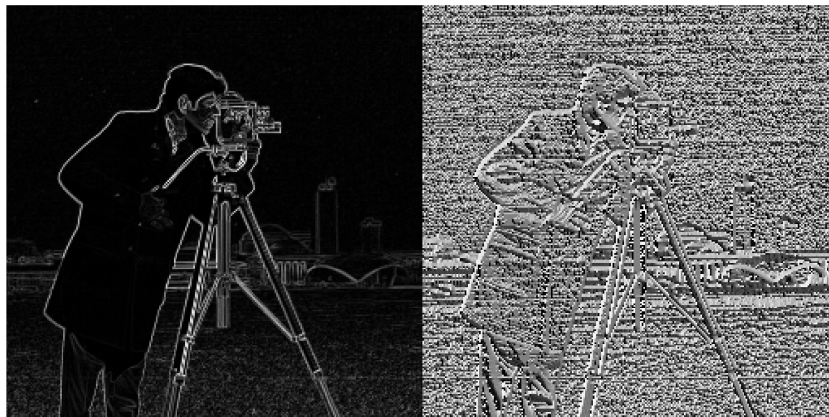
### Lab 3b

1. Read the cameraman image and compute the vertical and horizontal gradient images. Visualize these. Display the Gradient magnitude image and direction of the edges.

```
cm=imread('cameraman.tif');  
cmVG = cm(1:255,:) - cm(2:end, :);  
figure; imshow(cmVG); %we can also use>> imshow((1+log(double(cmVG))),[]);  
cmHG = ?? %TBD by you  
cmGradMag = cmVG./cmHG;  
cmGradDir = atan2(cmVG,cmHG);
```

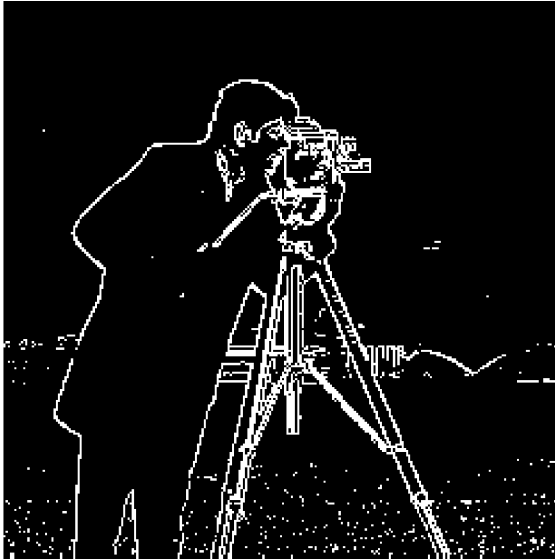
- a. Paste the gradient direction and magnitude images for cameraman

```
>> cmGradDir = atan2(double(cmVG),double(cmHG));  
>> cmVG = cm(1:255,:) - cm(2:end, :);  
>> cmHG = cm(:,1:255) - cm(:,2:end);  
>> cmVG = [zeros(1,256);cmVG];  
>> cmHG = [zeros(256,1),cmHG];  
>> cmGradMag = cmVG./cmHG;  
>> cmGradDir = atan2(double(cmVG),double(cmHG));
```

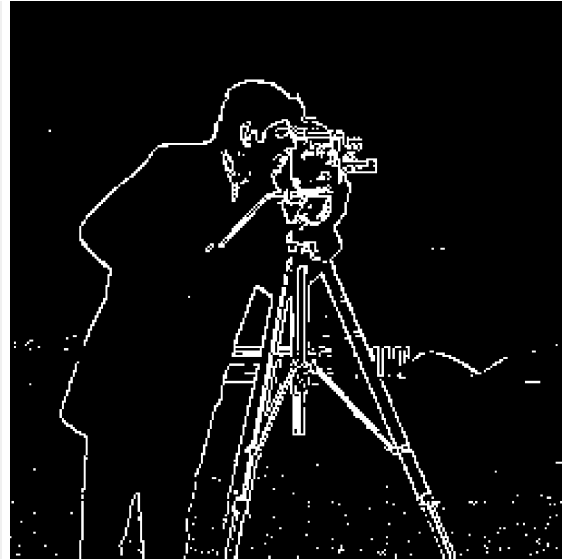


The image above shows the Gradient Magnitude on the right and the Gradient Direction on the left for the cameraman image.

- b. What value of threshold on the Gradient magnitude image (cmGradMag) allows you to visualize the edge map well? (You can try a few values.) Clearly cite the threshold value and paste the relevant lines of code and edge map here.



```
>> cmEdge = cmGradMag > 50;  
>> figure;imshow(cmEdge);
```



```
>> cmEdge = cmGradMag > 60;  
>> figure;imshow(cmEdge);
```

A threshold value between 50 to 60 is ideal for the cameraman image. This threshold is suitable to identify the edges of the cameraman and the tripod but the background building edges are not present.

- c. What are the most prominent edges seen in the Vertical Gradient versus the Horizontal Gradient images?



Fig : Vertical Gradient



Fig: Horizontal Gradient

In the Vertical Gradient image the features of the cameraman's face are more prominently clear and the tripod stand is detected very well and the edges of the smaller buildings are detected in the vertical gradient image. Whereas in the horizontal gradient image the tower is clearly present, and the details of the coat are clearly seen in the horizontal gradient. The camera is also clearly visible in the horizontal gradient.

2. Use conv2 to find vertical and horizontal edges using explicit filters, as follows:  

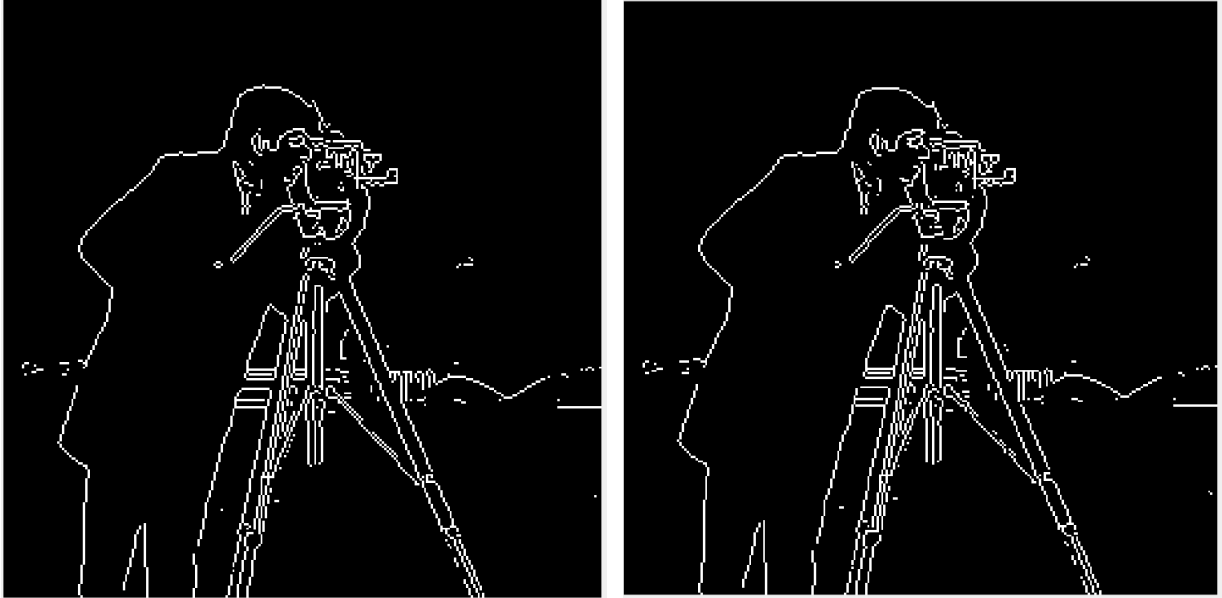
```
cmEdge2 = conv2(cm,[-1 0 1; -1 0 1; -1 0 1], 'same');  
%the parameter 'same' ensures o/p and i/p dimensions are the same  
figure; imshow(uint8(cmEdge2));
```

- a. Paste the filtered image here.



Repeat the exercise using the edge operation with the following operators:

- b. Prewitt and Sobel %>> cmEdge = edge(cm, 'Sobel');  
Paste the result of the edge detection here.



The left image is using the Sobel edge operation whereas the right image is using the Prewitt edge operation.

c. What sort of edges (vertical or horizontal) are these operations yielding?

Both the Prewitt and the Sobel edge operations give vertical edges as can be seen in the above images.

d. How different are the edges computed by Prewitt from Sobel? (Use image difference ( $f1-f2$ ) to compute the difference image and visualize the difference with a suitable transformation; paste the output here.)



The image on the left is the differenced image between the Prewitt and Sobel edge operators. Even after applying different types of transformation functions there isn't any significant difference between the two types of edge operators.

3. Using filters to remove different models of noise:

- a. Adding salt and pepper noise in the spatial domain and removing this using median filter

```
cmSnPNoise = imnoise(cm,'salt & pepper',0.05);  
figure; imshow(cmSnPNoise);  
cmDenoise = medfilt2(cmSnPNoise, [5,5]);  
figure; imshow(cmDenoise);
```



Figure : (left) Salt and Pepper Noise added to cameraman. (right) smoothed image using median filter with 5x5 kernel.

- b. Does 3x3 median filter give a better output? (Compute MSE between the original image and denoised cameraman image for the two median filters, 5x5 and 3x3 and report which one is the better filter for 5% salt and pepper noise on cameraman image based on the MSE.)

```
>> cmDenoise3x3 = medfilt2(cmSnPNoise, [3,3]);  
>> cmDenoise5x5 = medfilt2(cmSnPNoise, [5,5]);  
>> mse_3x3 = immse(cmDenoise3x3, cm);  
>> mse_5x5 = immse(cmDenoise5x5, cm);  
>> fprintf("MSE for Denoised Image using median filter with 3x3 kernel is %0.4f\n",mse_3x3);  
MSE for Denoised Image using median filter with 3x3 kernel is 140.6373  
>> fprintf("MSE for Denoised Image using median filter with 5x5 kernel is %0.4f\n",mse_5x5);  
MSE for Denoised Image using median filter with 5x5 kernel is 282.7434  
>> |
```



Figure : a. Median filtering using 3x3 kernel (left). b. Median filtering using 5x5 kernel (right).

MSE for Denoised Image using median filter with 3x3 kernel is 140.6373

MSE for Denoised Image using median filter with 5x5 kernel is 282.7434

Hence we can say that image denoised with a median filter using a 3x3 kernel produces much better results compared to using a 5x5 kernel. Visually, we can see noticeable differences near the camera region. The denoised image with 3x3 kernel preserves more details in the camera compared to 5x5 kernel median filter.

- c. Add 30% of salt and pepper noise and find the appropriate neighborhood of the median filter to clear this noise. Paste the noisy and denoised images along with the few lines of code to obtain these.



Figure : Salt and Pepper Noise added to Cameraman



Figure : (left) Median filter using 7x7 kernel. (middle) Median filter using 5x5 kernel. (right) Median filter using 3x3 kernel.

```
>> cmSnPNoise = imnoise(cm, 'salt & pepper', 0.3);
>> figure; imshow(cmSnPNoise);
>> cmDenoise7x7 = medfilt2(cmSnPNoise, [7,7]);
>> cmDenoise5x5 = medfilt2(cmSnPNoise, [5,5]);
>> cmDenoise3x3 = medfilt2(cmSnPNoise, [3,3]);
>> figure; imshow(cmDenoise7x7); hold on; title('Median Filtering - 7x7 Kernel');
>> figure; imshow(cmDenoise5x5); hold on; title('Median Filtering - 5x5 Kernel');
>> figure; imshow(cmDenoise3x3); hold on; title('Median Filtering - 3x3 Kernel');
>>
```

Figure : Code used to generate the denoised images.

- d. Adding Gaussian (additive white noise) and removing this using linear filtering; first try this:

```
cmGNoise = imnoise(cm,'gaussian',0,0.07);
figure; imshow(uint8(cmGNoise));
```



Figure : Gaussian noise added to image.





Figure : (top-left) Average Filter with 3x3 kernel. (top-right) Average Filter with 5x5 kernel. (bottom-left) Gaussian Filter with 3x3 kernel. (bottom-right) Gaussian Filter with 5x5 kernel.

```
>> cmGNoise = imnoise(cm, 'gaussian', 0, 0.07);
>> figure; imshow(cmGNoise);
>> img_avg3x3 = filter2(fspecial('average',3),cmGNoise)/255;
>> img_avg5x5 = filter2(fspecial('average',5),cmGNoise)/255;
>> img_gau3x3 = filter2(fspecial('gaussian',3,2),cmGNoise)/255;
>> img_gau5x5 = filter2(fspecial('gaussian',5,2),cmGNoise)/255;
>> figure; imshow(img_avg3x3); hold on; title('Average Filter with 3x3 Kernel');
>> figure; imshow(img_avg5x5); hold on; title('Average Filter with 5x5 Kernel');
>> figure; imshow(img_gau3x3); hold on; title('Gaussian Filter with 3x3 Kernel');
>> figure; imshow(img_gau5x5); hold on; title('Gaussian Filter with 5x5 Kernel');
>> |
```

Figure : Code used to Generate the images.

- e. Use an appropriate linear filter (box filter, Gaussian filter, etc.) to smooth the image to denoise this. Report the MSE between Median filtering and smoothing in this case. Try this:

```
g=fspecial('gaussian', size(cm), 0.07);
```



```
cmDenoise2 = conv2(cmGNoise,g,'same');
figure; imshow(uint8(cmDenoise2));
```

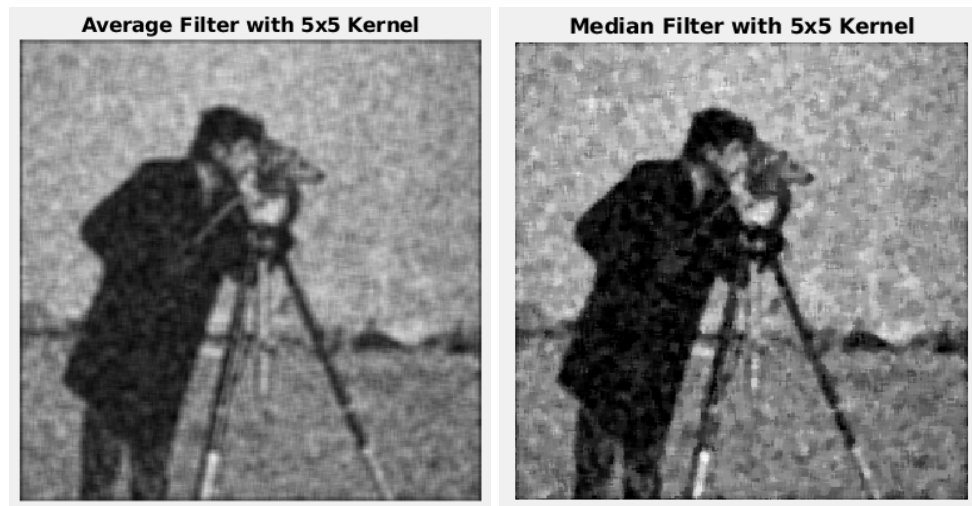


Figure : (left) Average Filter with 5x5 kernel. (right) Median Filter with 5x5 kernel.

```
>> img_avg5x5 = filter2(fspecial('average',5),cmGNoise);
>> img_med5x5 = medfilt2(cmGNoise, [5,5]);
>> mse_avg_med = immse(uint8(img_avg5x5), img_med5x5);
>> mse_avg_cm = immse(uint8(img_avg5x5), cm);
>> mse_med_cm = immse(img_med5x5, cm);
>> figure; imshow(uint8(img_avg5x5)); hold on; title('Average Filter with 5x5 Kernel');
>> figure; imshow(img_med5x5); hold on; title('Median Filter with 5x5 Kernel');
>> fprintf('\nMSE for Denoised Image using Average filter with 5x5 kernel compared with original image is %0.4f\n',mse_avg_cm);

MSE for Denoised Image using Average filter with 5x5 kernel compared with original image is 645.3201
>> fprintf('\nMSE for Denoised Image using Median filter with 5x5 kernel compared with original image is %0.4f\n\n',mse_med_cm);

MSE for Denoised Image using Median filter with 5x5 kernel compared with original image is 648.9920

>> fprintf('\nMSE for Denoised Image using Median filter with 5x5 kernel compared with Denoised Image using Average filter of 5x5 kernel is %0.4f\n\n',mse_avg_med);

MSE for Denoised Image using Median filter with 5x5 kernel compared with Denoised Image using Average filter of 5x5 kernel is 201.9486
```

Figure : Code used to generate the images.

MSE obtained for Denoised Image using average filter with 5x5 kernel compared with original image is 645.3201.

MSE obtained for Denoised Image using Median filter with 5x5 kernel compared with original image is 648.9920.

MSE obtained for Denoised Image using average filter with 5x5 kernel compared with Denoised Image obtained using median filter with 5x5 kernel is 201.9486.

## Lab 4

1. Read the cameraman image and display the Fourier Spectrum

```
cm=imread('cameraman.tif');
CM = fft2(cm);
```

```

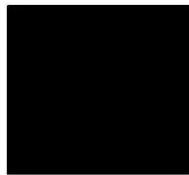
S = abs(CM); %same as (S=sqrt(real(CM).*real(CM))+(imag(CM).*(imag(CM))))
figure; imshow(S,[]);
FshiftCM = fftshift(CM);
S2 = log(1+abs(FshiftCM));
figure; imshow(S2, []);
PhaseAngle = atan2(imag(FshiftCM),real(FshiftCM));
figure; imshow(PhaseAngle,[]);

```

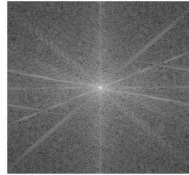
```

>> % Read the cameraman Image and display the fourier spectrum
>> cm=imread('cameraman.tif');
CM = fft2(cm);
S = abs(CM); %same as (S=sqrt(real(CM).*real(CM))+(imag(CM).*(imag(CM))))
figure; imshow(S,[]);
FshiftCM = fftshift(CM);
S2 = log(1+abs(FshiftCM));
figure; imshow(S2, []);
PhaseAngle = atan2(imag(FshiftCM),real(FshiftCM));
figure; imshow(PhaseAngle,[]);
>> |

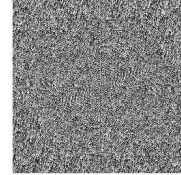
```



Fourier Transform without Shift



Fourier Transform with Shift



Phase Angle Image

- a. Create a rectangle of black and white strips and apply fft2 without the shift operation

```

% black and white strip creation
p=zeros(256,256);
p(1:128,1:end)=255;
figure; imshow(uint8(p));

```

```

>> % Create a rectangle of black and white strips
p=zeros(256,256);
p(1:128,1:end) = 255;
figure; imshow(uint8(p));

% Apply fft2 to the image without the shift operation
P = fft2(p);

```



Black and white striped Rectangle

- b. Display the Fourier Spectrum of the strip image 'p'

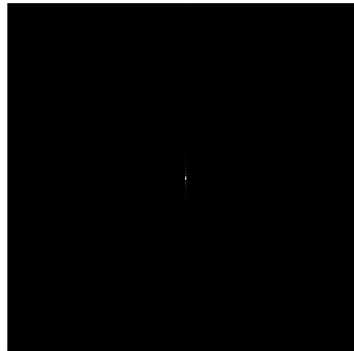
```
>> % Display the Fourier spectrum of the strip image 'p'  
P = abs(P);  
figure; imshow(P,[]);  
>> |
```



Fourier Spectrum of the strip image 'p'

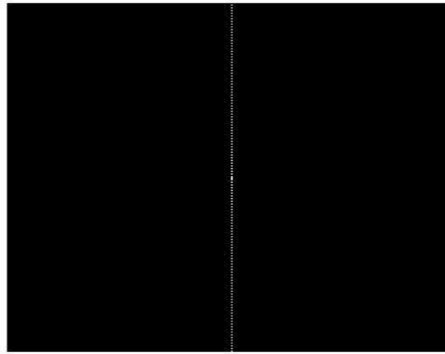
- c. Use fftshift and then display the Fourier Spectrum

```
>> % Use fftshift and then display the Fourier spectrum of the shifted image  
P_shift = fftshift(P);  
figure; imshow(P_shift,[]);  
>>
```



- d. Use fftshift and apply the log transform to display the Fourier Spectrum and be able to see something ☺ (use colormap gray if you use imagesc to display the image)

```
>> % Use fftshift and apply the log transform to display the fourier spectrum  
P_shift_log = log(1+abs(P_shift));  
figure; imshow(P_shift_log,[]);  
>> |
```



- e. Paste the lines of code for 1b-1e and images here.

```
>> % Create a rectangle of black and white strips
p=zeros(256,256);
p(1:128,1:end) = 255;
figure; imshow(uint8(p));

% Apply fft2 to the image without the shift operation
P = fft2(p);

% Display the Fourier spectrum of the strip image 'p'
P = abs(P);
figure; imshow(P,[]);

% Use fftshift and then display the Fourier spectrum of the shifted image
P_shift = fftshift(P);
figure; imshow(P_shift,[]);

% Use fftshift and apply the log transform to display the fourier spectrum
P_shift_log = log(1+abs(P_shift));
figure; imshow(P_shift_log,[]);

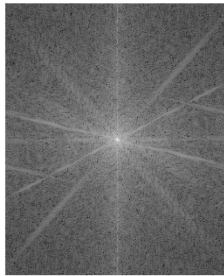
% Display P_shift_log using imagesc and set the colormap to gray
figure; imagesc(P_shift_log); colormap gray;
>>
```

- f. Find the inverse Fourier transform of cameraman after fftshift; obtain the original image using ifftshift. Paste the code and images here.

```

>> % Find Fourier Transform of cameraman image after fftshift
cm = imread('cameraman.tif');
cm_fft = fft2(cm);
cm_fft_shift = fftshift(cm_fft);
figure; imshow(log(1+abs(cm_fft_shift)),[]);
>> % Find the inverse fourier transform of the shifted image
cm_fft_shift_inv = ifftshift(cm_fft_shift);
figure; imshow(abs(cm_fft_shift_inv),[]);
>> % Obtain the original image using ifftshift
cm_fft_shift_inv_ifft = ifft2(cm_fft_shift_inv);
figure; imshow(abs(cm_fft_shift_inv_ifft),[]);
>>

```



Fourier Transform of cameraman



original image using inverse fft.

2. Filter this using a Gaussian filter with different sigma; let's start with sig=10.

```

[M,N] = size(p);
P = fft2(double(p));
sig = 10;
H = lpfilter('gaussian',M,N,sig); %hpfiler can be used for high pass filters with 'ideal' or 'btw' or 'gaussian'
figure; imshow(H);
G = H.*P; %filtering in the frequency domain without fftshift to facilitate visual interpretation
g = ifft2(G); %returning to the space domain
g = uint8(g);%this is the filtered image
figure; imshow(g);

```

- a. Apply the Gaussian filter using sig = 30, 60, 100; as the variance of the Gaussian increases, what happens to the black-white interface in p? Paste the lines of code and images here.

```

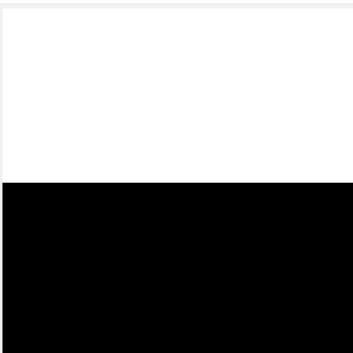
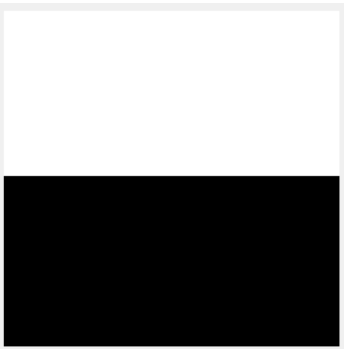
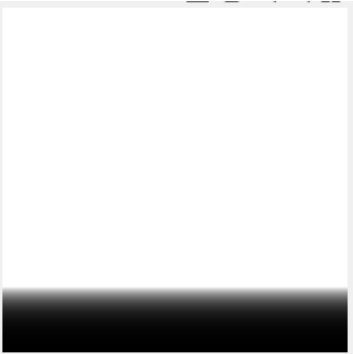
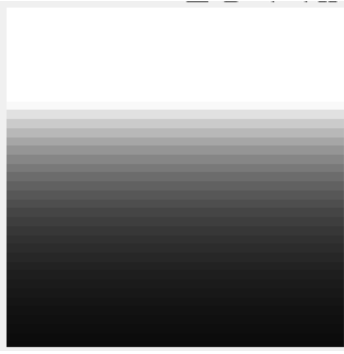
>> p=zeros(256,256);
p(1:128,1:end)=255;
figure; imshow(uint8(p));
[M, N] = size(p);

```

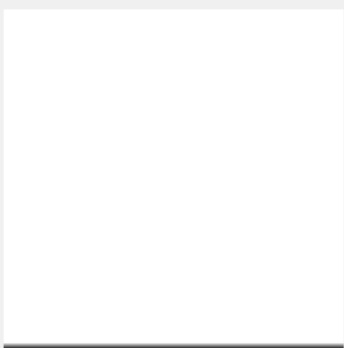

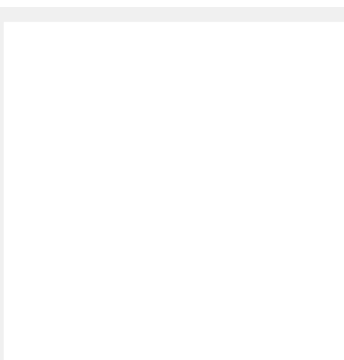

```

>> H = lpfilter('gaussian', M, N, 30);
figure;imshow(H);
g = uint8(iff2(H .* fft2(double(p))));
figure; imshow(g);
H = lpfilter('gaussian', M, N, 60);
figure;imshow(H);
g = uint8(iff2(H .* fft2(double(p))));
figure; imshow(g);
H = lpfilter('gaussian', M, N, 100);
figure;imshow(H);
g = uint8(iff2(H .* fft2(double(p))));
figure; imshow(g);

```

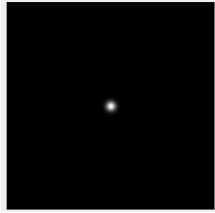
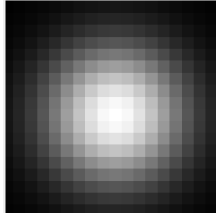
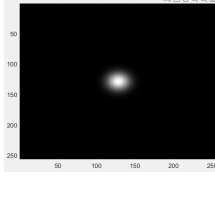
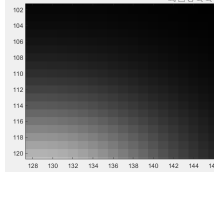
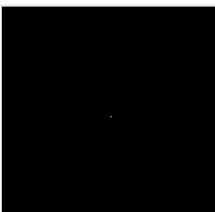
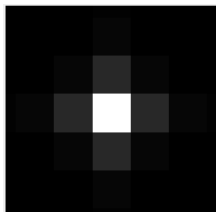
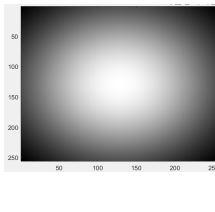
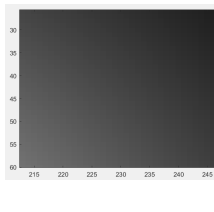
Sigma	Filtered image	Zoomed in for better visualisation
None(original Image)		
30		



60		
100		

It can be seen that the black part of the interface is getting reduced as sigma increases also the no of transition layers reduces as sigma increases.

- b. Use fftshift to visualize the spectrum of the Gaussian sig = 10 versus sig = 100 (with proper centering) and also display the corresponding images in the spatial domain.






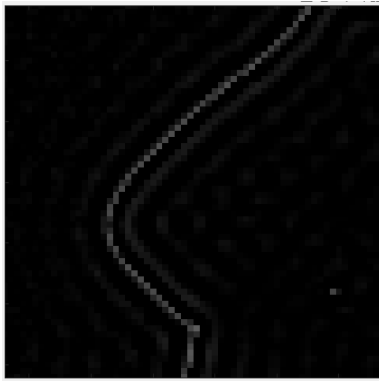
Sigma	Frequency Domain	Frequency Domain Zoomed in	spatial Domain	Spatial Domain zoomed in
10				
100				

- c. Define a low pass (lpfilter) and an ideal high pass filter (hpfilt) and apply these on the cameraman image. Do you see the ringing effect from the 2D sinc function?

```
function H = ilp(M, N , D0)
% Computing meshgrid
u = 0:(M-1);
v = 0:(N-1);
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;
[V, U] = meshgrid(v, u);

D = sqrt(U.^2 + V.^2); % computing Distances
H = double(D <= D0);

>> cam = imread('cameraman.tif');
[M, N] = size(cam);
figure; imshow(uint8(fft2(ilp(M, N, 50)) .* fft2(cam))));
% Highpass Filter
figure; imagesc(uint8(fft2((1 - ilp(M, N, 50)) .* fft2(cam)))); colormap gray;
figure; imshow(cam);
```

Original	Ideal low Pass Filter	Ideal High Pass filter
		
		

Yes the ringing effect can be clearly seen in the low pass filtered image.

- d. Use unsharp masking and/or high boost filtering on the spiral image (called UnsharpMasking\_Spiral.png) to ‘fix’ the blur in the top half of the image.

```
[M, N] = size(spiral);
spiral_mask = uint8(iff2( 1 + (hpfiler('gaussian', M, N, 10) .* fft2(double(spiral)))));
spiral_mask(end/2 - 2 : end, 1 : end) = 0;
subplot 221; imagesc(spiral_mask); colormap gray;
subplot 222; imagesc(spiral_mask + spiral); colormap gray;
subplot 223; imagesc(spiral_mask * 3 + spiral); colormap gray;
trans = spiral_mask * 3 + spiral;
trans = [uint8(1 * (double(trans(1:end / 2 - 3 , 1 : end)) .^ gamma(2))));(trans(end / 2 - 2 : end, 1 : end))];
subplot 224; imagesc(trans); colormap gray;
```

