9.35 Sensation And Perception

Spring 2009

Problem set 2

Answer key

Contents

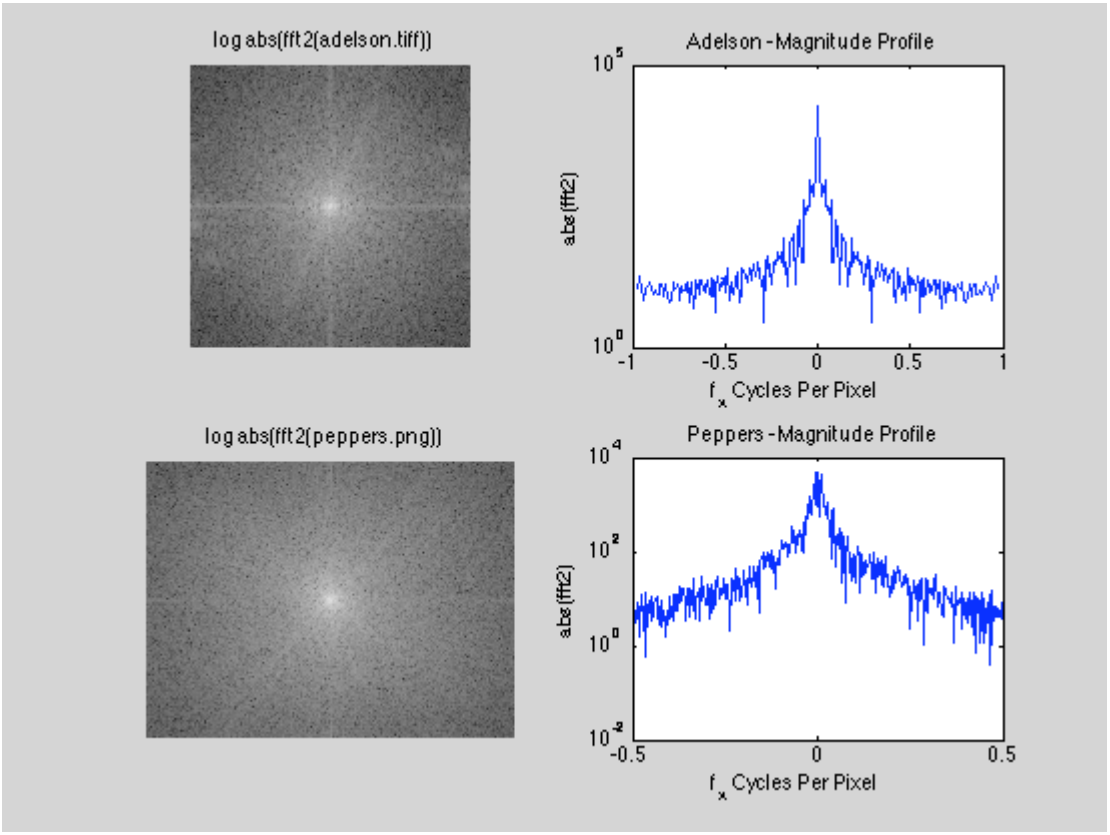
Init

```
clear all
close all
clc
```

Problem 1

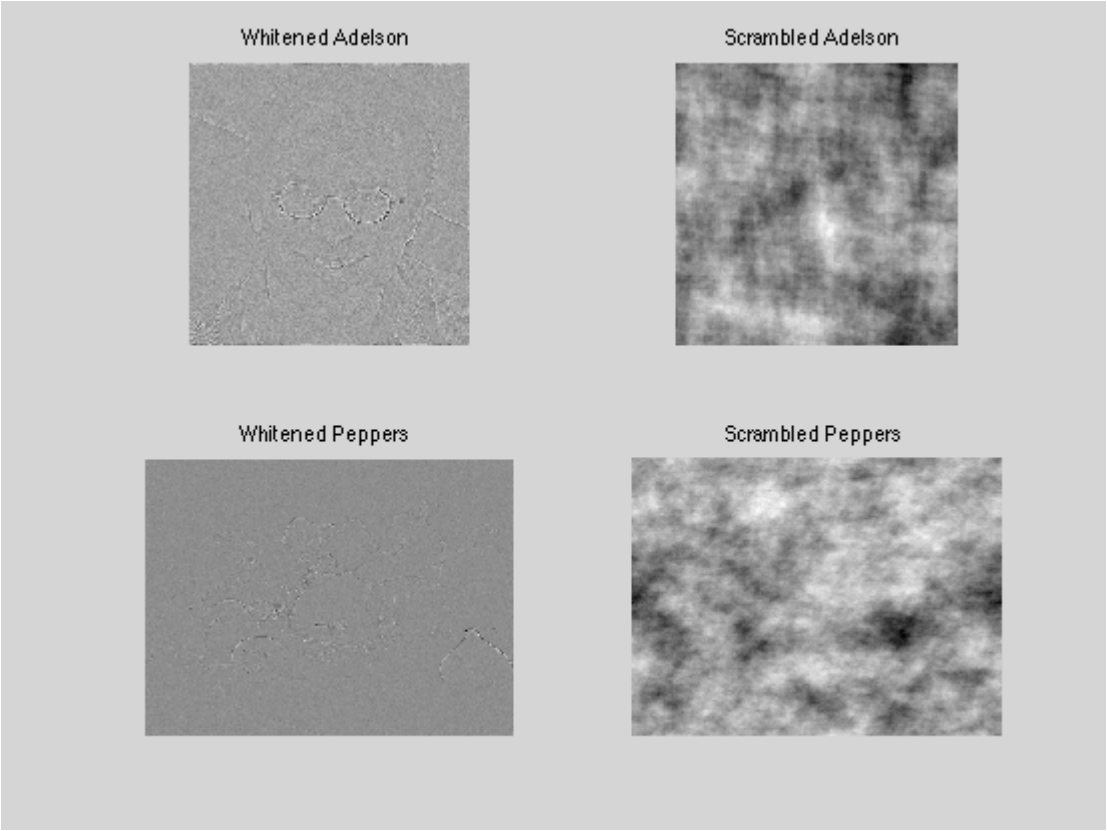
1a natural images have a characteristic 1/f^n power spectrum (where n is somewhere between 1 and 2.5). Meaning that the amplitude of frequencies is inversely proportional to the frequency. This makes sense, since the amplitude of a frequency, roughly, indicates how much the image is changing at that frequency. Natural images have spatial correlations: nearby pixels are more likely to be the same than pixels farther away. This means that hte image will be changing less at high spatial frequencies, and will thus have less power at higher frequencies. Thus, we will have a ~1/f power spectrum. Since peppers and you are natural images, it was safe to assume that you too would have such a power spectrum.

```
im = imread('peppers.png');
im = double(im(:,:,2))./255;

adelson=double(imread('adelson.tiff'))/255;

adelsonFT = fftshift(fft2(adelson));
peppersFT = fftshift(fft2(im));

figure();

subplot(2,2,1);
imagesc(log(abs(adelsonFT))); colormap gray, axis image, axis off
title('log abs(fft2(adelson.tiff))')
subplot(2,2,2);
semilogy((-124:124)/128, abs(adelsonFT(125,:)));
title('Adelson - Magnitude Profile')
xlabel('f_x Cycles Per Pixel')
ylabel('abs(fft2)')

subplot(2,2,3);
imagesc(log(abs(peppersFT))); colormap gray, axis image, axis off
title('log abs(fft2(peppers.png))')
subplot(2,2,4);
semilogy((-191.5:191.5)/383, abs(peppersFT(:, 255)));
xlabel('f_x Cycles Per Pixel')
ylabel('abs(fft2)')
title('Peppers - Magnitude Profile')
```



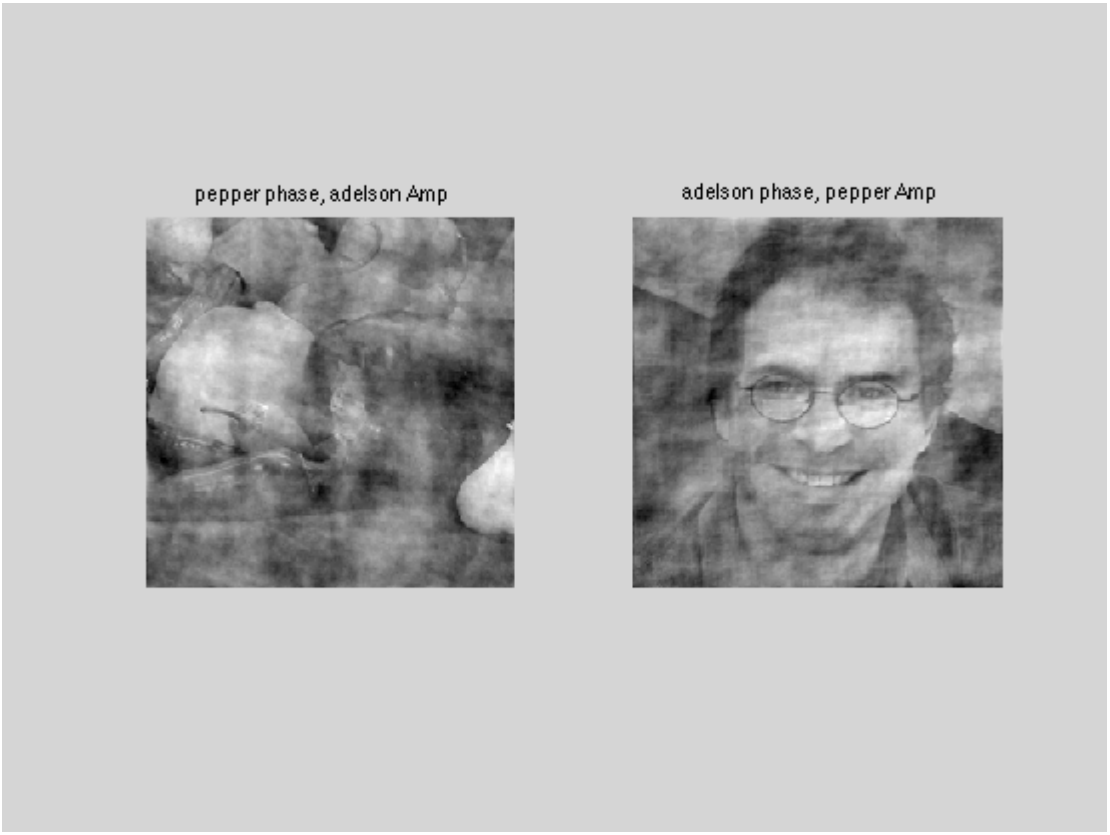
1b whitened images have equal power across the full spectrum, so there is more power at high spatial frequencies than we are used to with a 1/f spectrum.

```
randPeppers = fft2(randn(size(im)));
randAdelson = fft2(randn(size(adelson)));

i = sqrt(-1);
adelsonFT = ifftshift(adelsonFT);
peppersFT = ifftshift(peppersFT);

scrambledAdelson = real(ifft2(abs(adelsonFT).*exp(i*angle(randAdelson))));
scrambledPeppers = real(ifft2(abs(peppersFT).*exp(i*angle(randPeppers))));

whiteAdelson = real(ifft2(abs(randAdelson).*exp(i*angle(adelsonFT))));
whitePeppers = real(ifft2(abs(randPeppers).*exp(i*angle(peppersFT))));

figure();
subplot(2, 2, 1);
imagesc(whiteAdelson); colormap gray, axis image, axis off;
title('Whitened Adelson');
subplot(2, 2, 2);
imagesc(scrambledAdelson); colormap gray, axis image, axis off;
title('Scrambled Adelson');

subplot(2, 2, 3);
imagesc(whitePeppers); colormap gray, axis image, axis off;
title('Whitened Peppers');
subplot(2, 2, 4);
imagesc(scrambledPeppers); colormap gray, axis image, axis off;
title('Scrambled Peppers');
```



1c phase determines the spatial structure of the image, and we care about spatial structure more than the power spectrum. Hence, phase determines what we see in these hybrids.

```
peppersResized = im((1:249)+100, (1:249)+200);
peppersFT = fft2(peppersResized);


figure();
subplot(1,2,1);
modIMG = real(ifft2(abs(adelsonFT).*exp(i*angle(peppersFT))));
imagesc(modIMG); colormap gray, axis image, axis off;
title('pepper phase, adelson Amp');

subplot(1,2,2);
modIMG = real(ifft2(abs(peppersFT).*exp(i*angle(adelsonFT))));
imagesc(modIMG); colormap gray, axis image, axis off
title('adelson phase, pepper Amp');
```



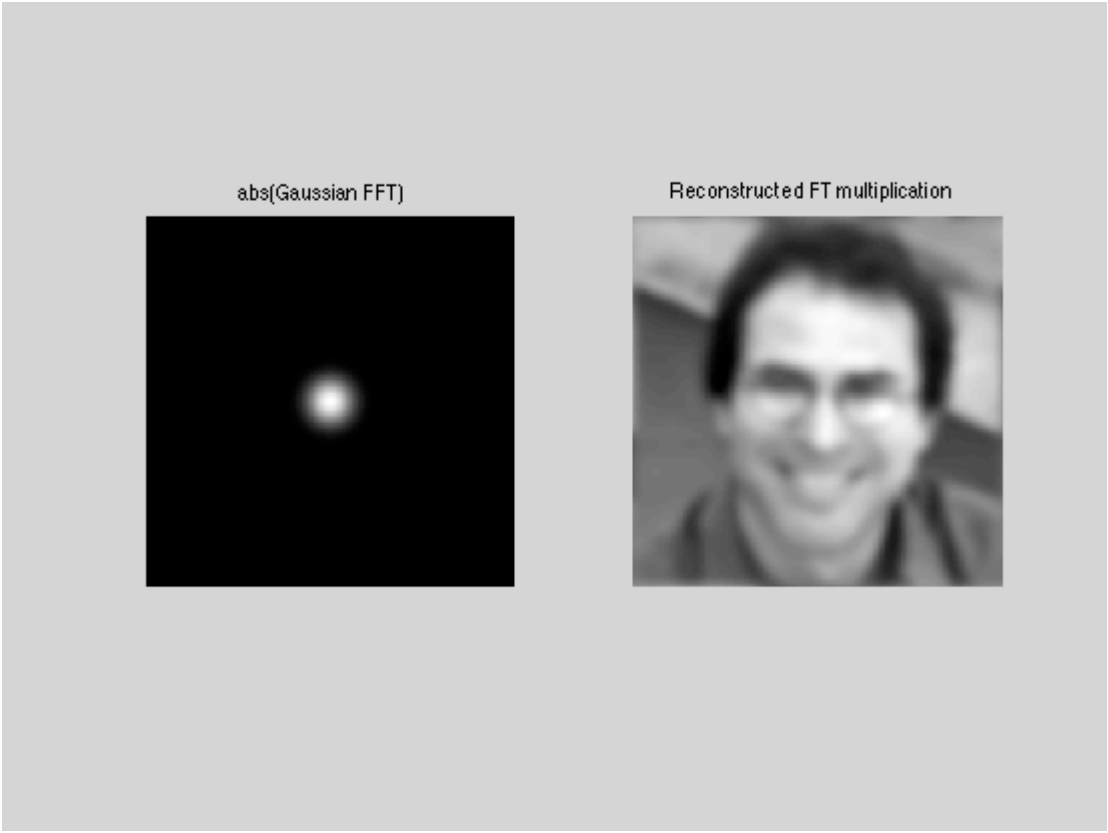
1d Convolving an image with a Gaussian filter blurs the image, because it is a weighted average of intensity over a local area. In other words, we lose high-frequency information. If multiplying the Fourier transforms is the same as convolution, then multiplying with a Gaussian FT should reduce the amplitude of high frequencies. So, a Gaussian FT looks like a radial mask (actually, it is mathematically identical to another Gaussian!). Now, multiplying the Gaussian FT by the FT of an image causes the high-frequencies to dissapear, but preserves the lower frequencies.

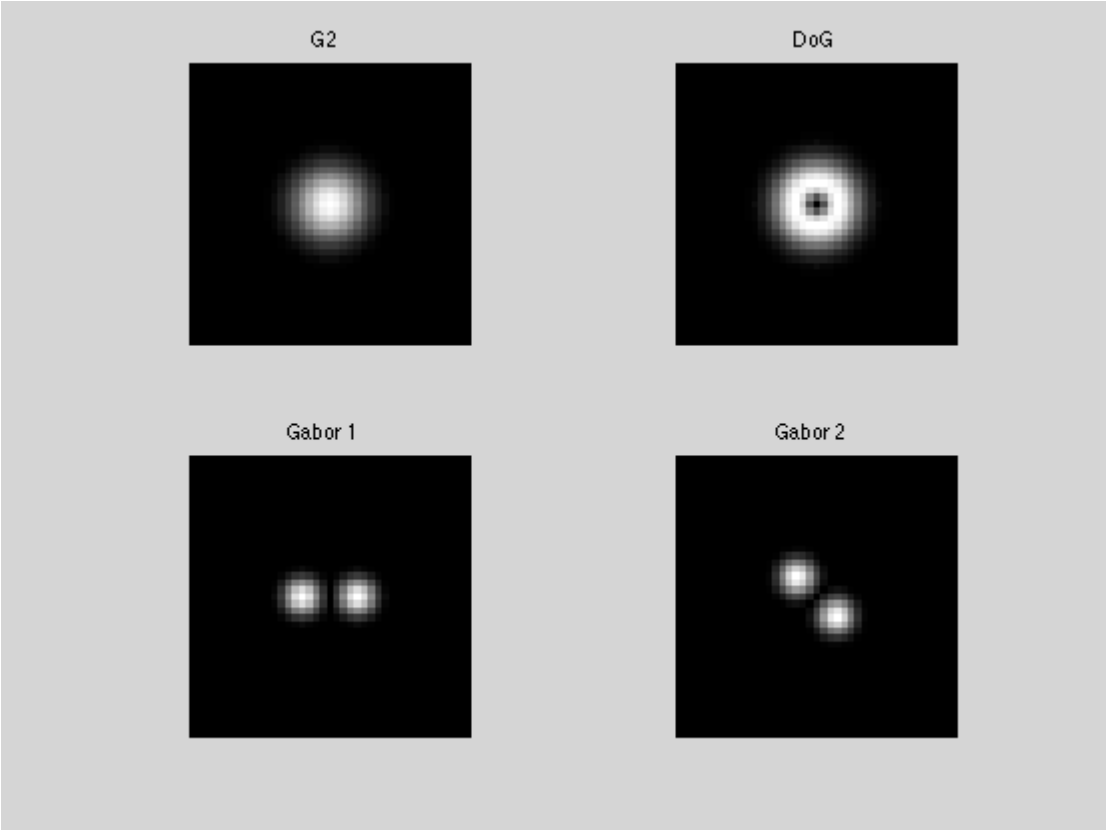
For more complex filters, we can understand the FT by looking at the components. A DoG filter selects out certain frequencies but responds poorly to ambient (low frequencies) or very high frequencies. Thus, it is a 'band-pass' filter, and the FT will look like a ring. Sine waves are single points in an FT (because sines are the basis function), and multiplying them by a Gaussian will simply introduce some frequency spill-off around those points. Different orientations of sine waves of the same frequency will simply have different components in f_x and f_y, so the points will be rotated about the origin.

It is important to note the inverse relationship between image distance and frequency. For instance, the wider Gaussian has a sharper FT, because it has lower frequencies, while a narrower (low STD) Gaussian has more high-frequencies and will have a broader FT. Correspondingly, high-frequency sine waves will be points further from the origin.

```
[X Y] = meshgrid(-20:20);

% Anonymous functions are a shortcut for repeated calculations like this
Gaussian = @(s) exp(-.5*(X.^2+Y.^2)/(s^2));

G1 = Gaussian(4);
G2 = Gaussian(2);

% Normalize (integrate to 1)
G1n = G1 ./ sum(G1(:));
G2n = G2 ./ sum(G2(:));

% make diff. of. gauss
DoG = G2n-G1n;

% make gabors
f = 4/size(X,1);
phi = (0:3)*pi/4;
for i = 1:4
   I{i} = G1.*sin(2*pi*f*(X*cos(phi(i))+Y*sin(phi(i))));
end


G1p = padarray(G1, [104 104]);
g1FT = fftshift(fft2(G1p));

figure();
subplot(1,2,1);
imagesc(abs(g1FT)); colormap gray, axis image, axis off
title('abs(Gaussian FFT)');

subplot(1,2,2);
newFT = ifftshift(g1FT).*adelsonFT;
```

```
modIMG = ifftshift(real(ifft2(newFT))); % For some reason, the phase shifts by pi here, requires ifftshift
imagesc(modIMG); colormap gray, axis image, axis off
title('Reconstructed FT multiplication')

% Anonymous function for simplicity
dispFT = @(im) imagesc(fftshift(abs(fft2(im))));

figure
subplot(2, 2, 1)
dispFT(G2); colormap gray, axis image, axis off;
title('G2')
subplot(2, 2, 2)
dispFT(DoG); colormap gray, axis image, axis off;
title('DoG')
subplot(2, 2, 3)
dispFT(I{1}); colormap gray, axis image, axis off;
title('Gabor 1')
subplot(2, 2, 4)
dispFT(I{2}); colormap gray, axis image, axis off;
title('Gabor 2')
```



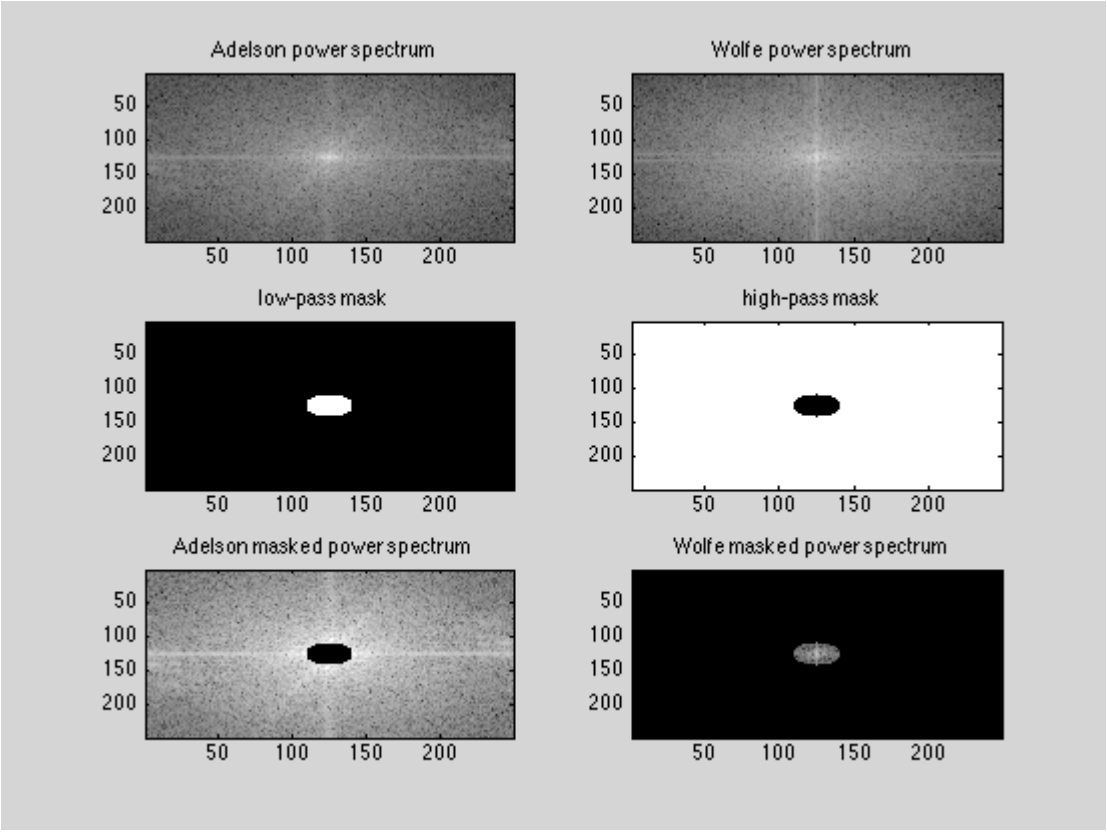


Problem 2

Hybrid images look like one image at close distances and the other image at longer distances because of your contrast sensitivity function. There is more contrast "energy" (contrast weighted by your sensitivity to it) in the high spatial frequency image when you are close, and more of this energy in the low spatial frequency image when you are far. This works because we have a U shaped contrast sensitivity function that drops off for high and low spatial frequencies alike.

```
[X Y] = meshgrid([-124:124], [-124:124]);
[Th, R] = cart2pol(X,Y);
mask = (R<=16);


adelsonFT = fft2(adelson);

wolfe = double(imread('wolfe.tif'))/255;
wolfeFT = fft2(wolfe);

figure();
subplot(3,2,1);
imagesc(log(abs(fftshift(adelsonFT))));
colormap gray
title('Adelson power spectrum')

subplot(3,2,2);
imagesc(log(abs(fftshift(wolfeFT))));
colormap gray
title('Wolfe power spectrum')

subplot(3,2,3);
imagesc(mask);
colormap gray
title('low-pass mask');

subplot(3,2,4);
imagesc((~mask));
colormap gray
title('high-pass mask');

subplot(3,2,5);
imagesc(log(abs(fftshift(adelsonFT)).*(~mask)));
colormap gray
title('Adelson masked power spectrum')


subplot(3,2,6);
imagesc(log(abs(fftshift(wolfeFT)).*mask));
colormap gray
```

```
title('Wolfe masked power spectrum')

scaleIm = @(im) (im-min(im(:)))/(max(im(:))-min(im(:)));

wolfeLowPass = scaleIm(real(ifft2(fftshift(mask).*wolfeFT)));
adelsonHighPass = scaleIm(real(ifft2(fftshift(~mask).*adelsonFT)));

hybrid = wolfeLowPass + adelsonHighPass;

figure();
imagesc(hybrid); colormap gray, axis image, axis off
```



Published with MATLAB® 7.7