CS543: Computer Vision **Spring 2017**

Problem Set 1

Chongxin Luo (cluo5) *Handed In: February 14, 2017*

1. **Lighting**

   (a) Answer the following regarding the above image.

      i. The direction of dominant light source is from right and above.

      ii. One of the temple tips shows very bright is because the material of the temple tips has the property which contains both diffuse and specular reflection. This specific orientation of surface makes the specular reflection directly into the camera which looks a lot more brighter than the surrounding materials.

      iii. The dark streaks in the wood are caused by the texture of the material. Mainly because those spots has a low albedo and low reflectance, which will show up as darker and less reflective.

      iv. If the table were completely specular, the glasses will not cast a shadow on it. It is because from the camera's view, we will only see the light from the other side of the room bounds off the table surface into the camera, and there will be no influence of the glass blocking the light from the light source.

   (b) Answer the following using the above illustration.

      i. Suppose the surface has a specular component, therefore, the observed intensities contain both specular reflection and diffuse reflection. According to Lambert's cosine law, the diffuse reflection intensity does not depend on viewer angle. Therefore, the change of observed intensities is depending on the specular reflection. As the illustration shown, surface 2 has specular reflection directly point back to the light source, surface 1 has specular reflection towards left and surface 3 has specular reflection towards right. Therefore, while the camera moving around the object, the observed intensities will peak three times, which each time corresponding to the specular reflection of one surface, and rest of the time, the observed intensities stay the same.

      ii. Given the relationship between intensity and surface orientation follows the equation below :

$$I(x) = \rho(x)(S * N(x))$$
$$\text{wehre } \rho = \text{albedo}$$
$$S = \text{directional source}$$
$$N = \text{surface normal}$$
$$I = \text{reflected intensity}$$

      Assume the surfaces has constant albedo and same directional source. The observed intensity from three surfaces can be express as equations below:

$$I_1 = \rho(x)(S * N_1(x)) = \rho(x)(\|S\|\|N\|cos(\theta_1)) = \rho(x)(\|S\|\|N\|cos(\theta_{12}))$$
$$I_2 = \rho(x)(S * N_2(x)) = \rho(x)(\|S\|\|N\|cos(180))$$
$$I_3 = \rho(x)(S * N_3(X)) = \rho(x)(\|S\|\|N\|cos(\theta_3)) = \rho(x)(\|S\|\|N\|cos(\theta_{23}))$$

Given $I_1 = 0.5$, $I_2 = 0.9$, $I_3 = 0.8$, and $\rho(x)$, S, N are constant. Solving the system of equation above, calculated the results as $\theta_{12} = 123.75^o$ and $\theta_{23} = 152.73^o$

2. **Image Pyramids**

(a) A Gaussian and Laplacian pyramid of 5 levels was created in this part. There are two main methods can be used to create the Gaussian pyramid, the first method uses Matlab's build-in function $imgaussfilt$, which creates a Gaussian smooth image. And the second method uses function $fspecial$ to create a Gaussian filter, and use function $imfilter$ to apply the filter onto image. The second method was used in this implementation, which a Gaussian filter of $\sigma = 3$, size $= 4*\sigma+1 = 13$ created using function $fspecial$ and being applied onto each level of image with replicate boundaries method being used. The resulted Gaussian and Laplacian pyramids are shown in the figures below, where the top 5 images are the Gaussian pyramid and bottom 5 images are the corresponding Laplacian pyramid. For the detailed code implementation, please refer to the code attached below.

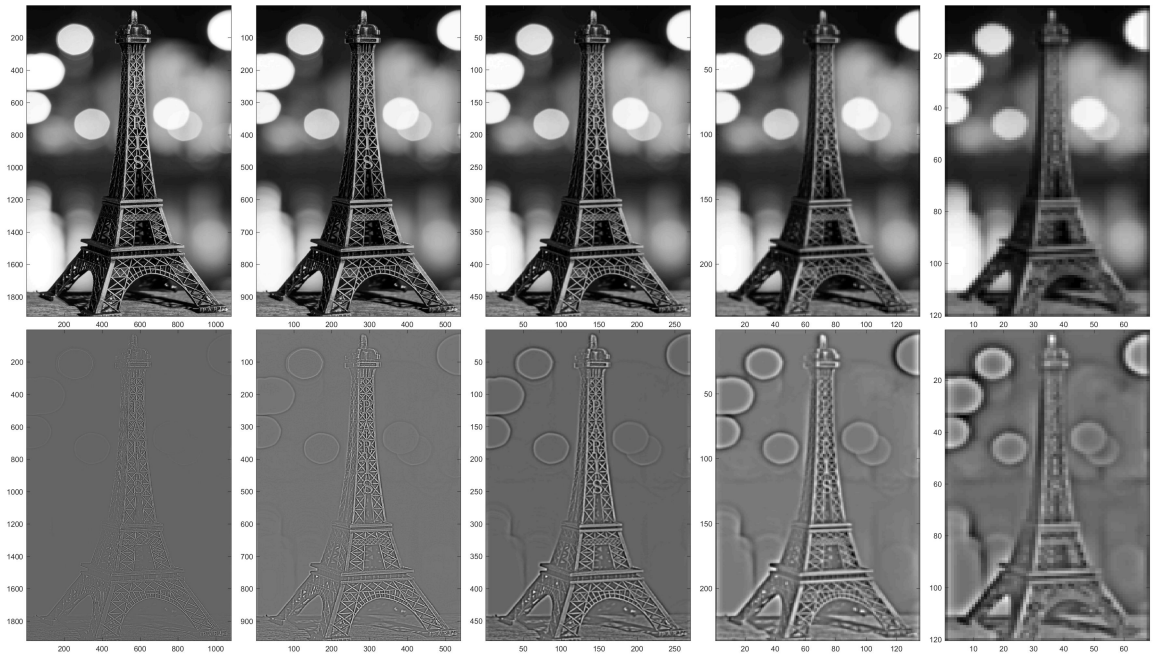

Figure 1: Gaussian/Laplacian Pyramid

```
%loading image
clc; close all; clear all;
levels = 5;
sigma = 3;
hsize = 4*sigma+1;
I = im2double(imread('tower.jpg'));
I = rgb2gray(I);
G_filter = fspecial('gaussian', hsize, sigma);

G_pyramid = cell([1,levels]);
L_pyramid = cell([1,levels]);
G_fft = cell([1,levels]);
L_fft = cell([1,levels]);

%---- Generating both G pyramid and L pyramid -----%
for n = 1:levels
    prev_img = I;
    %post_img = imresize(imgaussfilt(I,sigma),0.5);
    post_img = imresize(imfilter(I,G_filter), 0.5);
    %post_img = impyramid(I, 'reduce');
    up_scale = imresize(post_img, size(prev_img));
    Laplace = prev_img - imgaussfilt(up_scale, sigma);
```

```
    G_fft{n} = log(abs(fftshift(fft2(I)))+1);
    L_fft{n} = log(abs(fftshift(fft2(Laplace)))+1);
    G_pyramid{n} = I;
    L_pyramid{n} = Laplace;
    I = post_img;
end
%---- output both pyramid ----%
figure
ha = tight_subplot(2,levels,[.01 .01],[.1 .1], [.01, .01]);
for n = 1:levels; axes(ha(n)); colormap gray; imagesc(G_pyramid{n}); end
for n = 1:levels; axes(ha(n+levels)); colormap gray; imagesc(L_pyramid{n}); end
set(ha(1:4),'XTickLabel',''); set(ha,'YTickLabel','')

figure
ta = tight_subplot(2,levels,[.01 .01],[.1 .1], [.01, .01]);
for n = 1:levels; axes(ta(n)); colormap jet; imagesc(G_fft{n}, [0,10]); end
for n = 1:levels; axes(ta(n+levels)); colormap jet; imagesc(L_fft{n}, [0,10]); end
set(ta(1:4),'XTickLabel',''); set(ta,'YTickLabel','')
```

(b) The FFT amplitudes of both Gaussian and Laplacian pyramids are calculated and vitalized as shown in the image below. From the FFT amplitudes of the pyramids, we can see that the Gaussian pyramids works as a low-pass filter, which filters out the high frequencies of the image, and makes the image more blurry. The Laplacian pyramids works as a high-pass filter, which filters out the low frequencies and show up the edges of image more clearly.

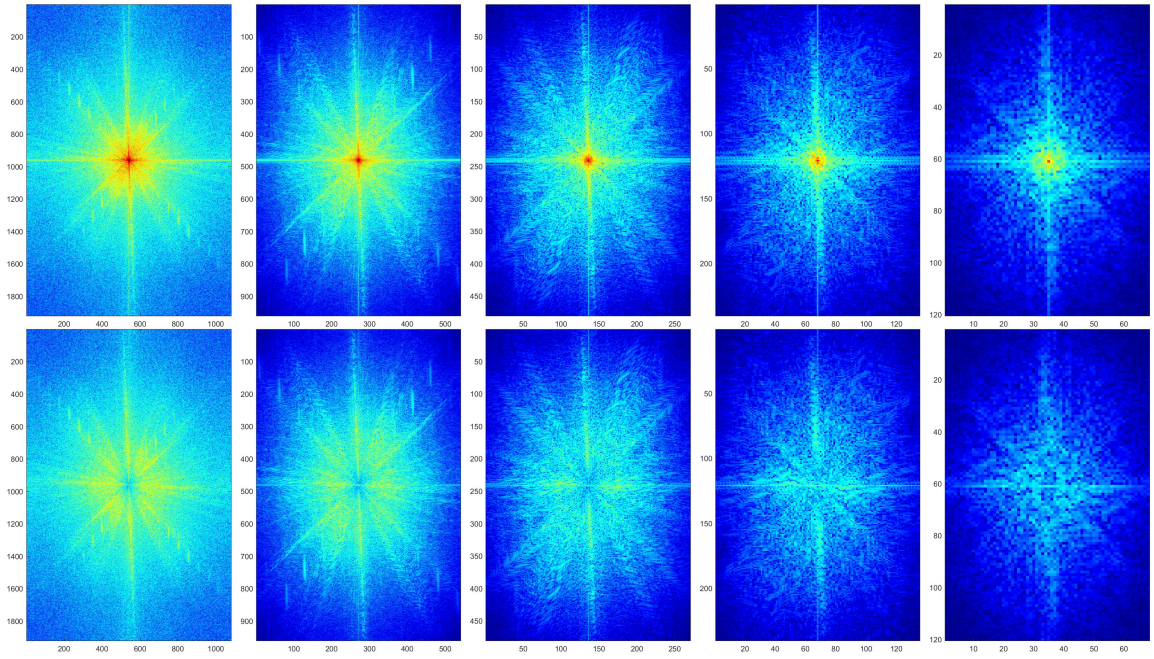

Figure 2: Gaussian/Laplacian FFT

3. **Edge Detection**

   (a) A simple gradient-based edge detector was implemented in this part. A Gaussian filter was created and applied to the input image to produce a smoothed picture. The Gaussian filter that being used has the parameters of $\sigma = 3$, size $= 4 * \sigma + 1 = 13$. Replicate boundary method was being used during the smoothing process. Then, the smoothed image was separated into R,G, and B channels individually, and the gradient and directions of the gradient was calculated for each channel using the Matlab's build-in function *imgradient*. For each pixel, the largest gradient and the corresponding directions amount three color channels was chosen as the final output gradient magnitude and gradient direction for the function *gradientMagnitude*. The result of this simple gradient-based edge detector is shown in figures below.

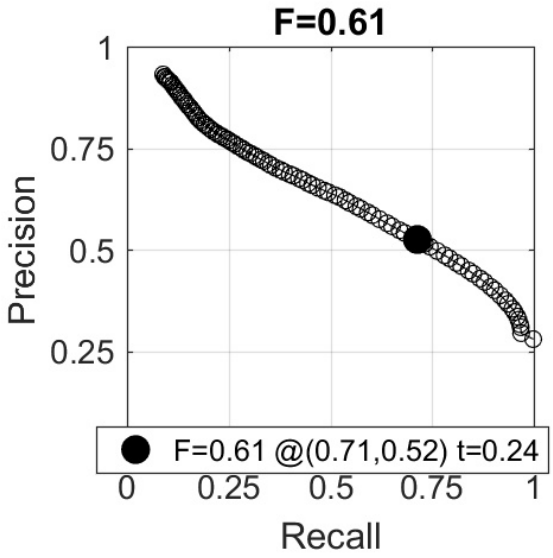

Figure 3: Gradient edge detection precision recall plot

Figure 4: Gradient edge detector sample image

Figure 5: Gradient edge detector sample image soft edge

As shown in the figures above, for the gradient-based edge detector, the overall F-score is 0.605, and the average F-score is 0.641. From the two comparison images, which Figure 4 shows the original image and Figure 5 shows the soft edge image after processing through gradient-based detector. The detector is able to successfully detect most edges, especially when there is a great contract between intensities on the edges, and the detector performs less optimal on the part of the image that has less contracts between edges such as the edges under the shadow.

```
function [mag, theta] = gradientMagnitude(im, sigma)

% step 1 : smooth the image with Gaussian std = sigma
smooth = imgaussfilt(im, sigma);
% step 2 : separating R G B channels for individual operations
[Rmag, Rdir] = imgradient(smooth(:,:,1));
[Gmag, Gdir] = imgradient(smooth(:,:,2));
[Bmag, Bdir] = imgradient(smooth(:,:,3));

imSize = size(Gmag);
mag = zeros(imSize);
theta = zeros(imSize);

row = imSize(1)
colum = imSize(2)

% step 3: Comparing gradient between three color channels, and output the largest gradient
for ii = 1:row
    for jj = 1:colum
        cell = [Rmag(ii,jj), Gmag(ii,jj), Bmag(ii,jj)];
        maxmag = max(cell);
        if maxmag == cell(1)
            theta(ii,jj) = Rdir(ii,jj);
        elseif maxmag == cell(2)
            theta(ii,jj) = Gdir(ii,jj);
        else
            theta(ii,jj) = Bdir(ii,jj);
        end
        mag(ii,jj) = norm(cell);
    end
end
```

(b) In order to improve the results of edge detector, a set of oriented filters were used instead of a simple derivative of Gaussian approach from part (a). The set of oriented filters is a set of elliptical Gaussian filters, with $\sigma_x = 5$, $\sigma_y = 2$, size $= 4 * \sigma_x + 1 = 21$, and orientations range from 0^o to 180^o with 30^o apart in between each filters. There are total of 7 filters being produced for the filter set. The visualized filter set is shown in the figure below.

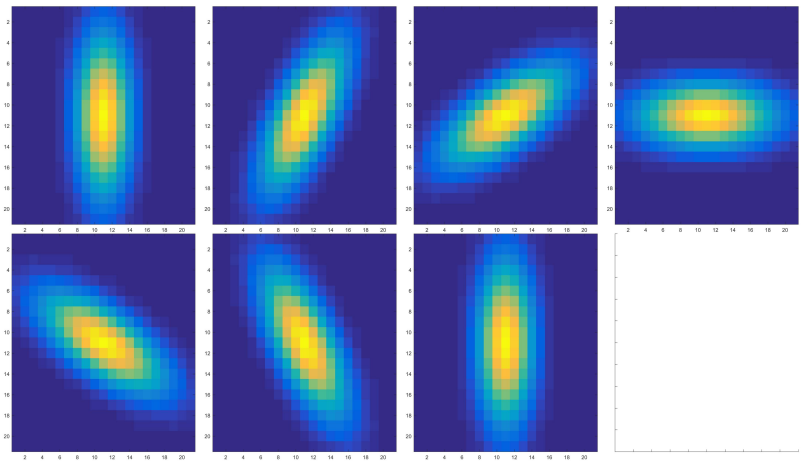

Figure 6: Oriented Filter set visualization

For each input image, it is being separated into R, G, and B color channels, the oriented filter set is applied onto each channel, and the gradient was calculated to each filter output. For each pixel, the gradient magnitude *magnitude* and gradient direction *theta* values for each color channel are chosen as the maximum value amount all 7 filters. And lastly, for each pixel, the maximum magnitude of gradient amount R, G, and B color channels is chosen to be the final output *magnitude*, and the corresponding direction is chosen to be the final *theta*.

The oriented filter edge detector was being tested against a set of input image, and the result gives a total F-score of 56.8, and average F-score of 59.6. The precision-recall graph is shown in the figure below, along with a set of comparison images.

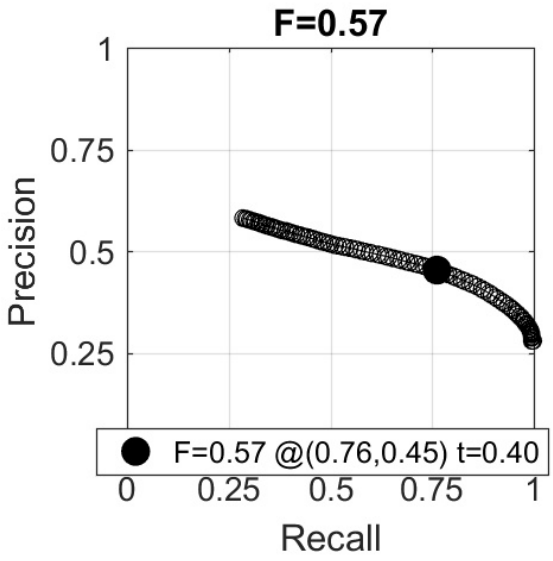

Figure 7: Oriented filter edge detection precision recall plot

Figure 8: Oriented filter edge detection sample image

Figure 9: Oriented filter edge detection sample image soft edge

From the results figures shown above, the oriented filter edge detector has a slightly lower performance comparing with the simple Gaussian filter approach. However, from the soft edge image, we can see that oriented filter detector is very sensitive to edges, which by tuning the parameters of the filter set, we can still improve the performance of this edge detector. The detailed code implementation of the oriented filter edge detector is shown below.

```
function [ mag, theta ] = orientedFilterMagnitude(im)
% Funtion which applys oriented filters to image im, and output a single
% Gradient image

% -- Parameters for the oriented Gaussian filter set ---%
sigx = 10; sigy = 1; width = 4*sigx+1; offset = 0; factor = 1;

orientations = [0:30:180];
fColumn = length(orientations);
grayim = rgb2gray(im);

% -- created normalized oriented Gaussian filter set --%
G_filter = cell([1, fColumn]);
for ii = 1:fColumn
    G_filter{ii} = customgauss([width,width],sigx, sigy,orientations(ii),offset,factor,[0,0]);
    add = sum(sum(G_filter{ii}));
    G_filter{ii} = G_filter{ii}/add;
end

imSize = size(grayim);
imRow = imSize(1); imColumn = imSize(2);

% -- separate image into R G B channels --%
RGB_layers = cell([1,3]);
RGB_layers{1} = im(:,:,1); RGB_layers{2} = im(:,:,2); RGB_layers{3} = im(:,:,3);

mag_array = cell([1,fColumn]);
theta_array = cell([1,fColumn]);

ColorMag_array = cell([1,3]);
ColorTheta_array = cell([1,3]);

mag = zeros(imSize);
theta = zeros(imSize);

for nn = 1:3
    % apply all filter to one color channel
    for ii = 1:fColumn
        [mag_array{ii}, theta_array{ii}] = imgradient( imfilter(RGB_layers{nn}, G_filter{ii}) );
    end
```

```
    % find best mag and theta
    for ii = 1:imRow
        for jj = 1:imColumn
            mag_c = [mag_array{1}(ii,jj), mag_array{2}(ii,jj), mag_array{3}(ii,jj), mag_array{4}(ii,jj),
                mag_array{5}(ii,jj), mag_array{6}(ii,jj), mag_array{7}(ii,jj)];
            theta_c = [theta_array{1}(ii,jj), theta_array{2}(ii,jj), theta_array{3}(ii,jj),
                theta_array{4}(ii,jj), theta_array{5}(ii,jj), theta_array{6}(ii,jj), theta_array{7}(ii,jj)];
            ColorMag_array{nn}(ii,jj) = max(mag_c);
            ColorTheta_array{nn}(ii,jj) = max(theta_c);
        end
    end
end

% -- merge three channels to form final mag and theta output -- %
for ii = 1:imRow
    for jj = 1:imColumn
        c = [ColorMag_array{1}(ii,jj),ColorMag_array{2}(ii,jj),ColorMag_array{3}(ii,jj)];
        maxmag = max(c);
        if maxmag == c(1)
            theta(ii,jj) = ColorTheta_array{1}(ii,jj);
        elseif maxmag == c(2)
            theta(ii,jj) = ColorTheta_array{2}(ii,jj);
        else
            theta(ii,jj) = ColorTheta_array{3}(ii,jj);
        end
        mag(ii,jj) = norm(c);
    end
end
```

(c) For future improvement of the oriented filter edge detector, from the soft edge image output, we can see that this filter is very sensitive to all lines/edges in the image, which result to picking up a lot of details in the image which are not classified as edge. In order to improve the result, we can increase the threshold of the edge detection, which during the process of computing the local maximum along the gradient direction, the area of image of local maximum will be determined larger in order to filter out detailed high frequency changes in the image. By doing so, we can limit out the problem of edge detector picking up small details in the image and classify them as edges.