Solution to Problem Set 2

Assigned: Feb 23, 2006
Due: March 9, 2006

Problem 1 *Image Histograms*

Load grayscale image `mountrainier.jpg` and display its histogram. Perform histogram equalization. You may use MATLAB functions `imhist` and `histeq`. Use function `rgb2gray` to convert RGB image to grayscale.

Now take image `winterstorm.jpg` and transfer its pixel histogram to image `mountrainier.jpg`.

Please display the original image, transferred image, and their histograms side by side.

Solution. Run the following code to display histogram and do histogram equalization

```
im=im2double(rgb2gray(imread('mountrainier.jpg')));
figure;imshow(im);

% histogram equalization
Im=histeq(im,256);

% display the histograms and image after hist-equalization
figure;imhist(im);
figure;imhist(Im);
figure;imshow(Im);
```

See results in Figure 1. Note that it is desired to use number smaller than 256 in function `histeq`. You may try `Im=histeq(im,64)` to see flatter result.

Run the following code for histogram transfer

```
im1=im2double(rgb2gray(imread('winterstorm.jpg')));
im2=im2double(rgb2gray(imread('mountrainier.jpg')));
figure;imshow(im1);
```

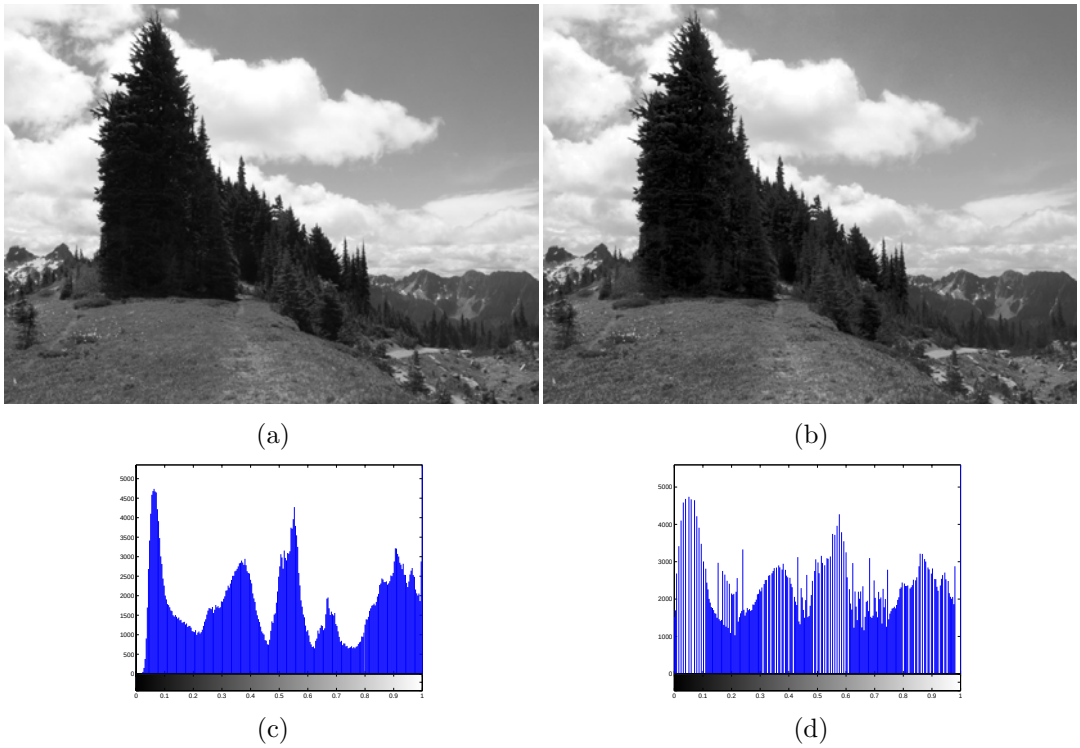
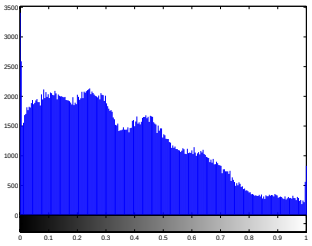


Figure 1: The histogram (c) of the original image (a) is equalized to (d), with corresponding image (b). We are able to see more details of the image after histogram equalization. The histogram in (d) is not completely flat because the intervals are not uniform. You may try `Im=hist(im,64)`.

```
% histogram transfer
counts=imhist(im1)*(prod(size(im2))/prod(size(im1)));
Im=histeq(im2,counts);

% show the histograms
figure;imhist(im1);
figure;imhist(im2);
figure;imhist(Im);

% show the image after histogram transfer
figure;imshow(Im);
```

Please see the results in Figure 2. Certain aspects of the tonescale used by Ansel Adams are transferred to the source image. □

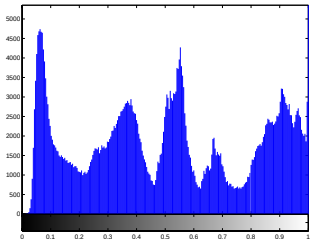
(a)



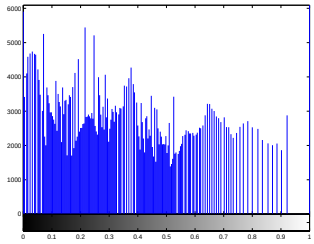
(b)

(c)

(d)



(e)



(f)

Figure 2: Histogram transfer can modify histogram arbitrarily based on a reference image. Here we transfer the histogram of `winterstorm.jpg` to `mountrainier.jpg` and obtain image (d) and the corresponding histogram (f).

Problem 2 *Steerable Pyramids*

2.1 *Warm up*

Download Simoncelli's steerable pyramid code from `http://www.cns.nyu.edu/~eero/steerpyr/`. Please make sure that you have read the readme file and add the path `../matlabPyrTools` and `../matlabPyrTools/MEX` where `..` is the parent directory of the toolbox. Try it on an image with k=3 (four orientations). Load image `einstein.jpg` and set the three highest frequency bands to zero for the horizontal orientation. Reconstruct the image and observe the effect.

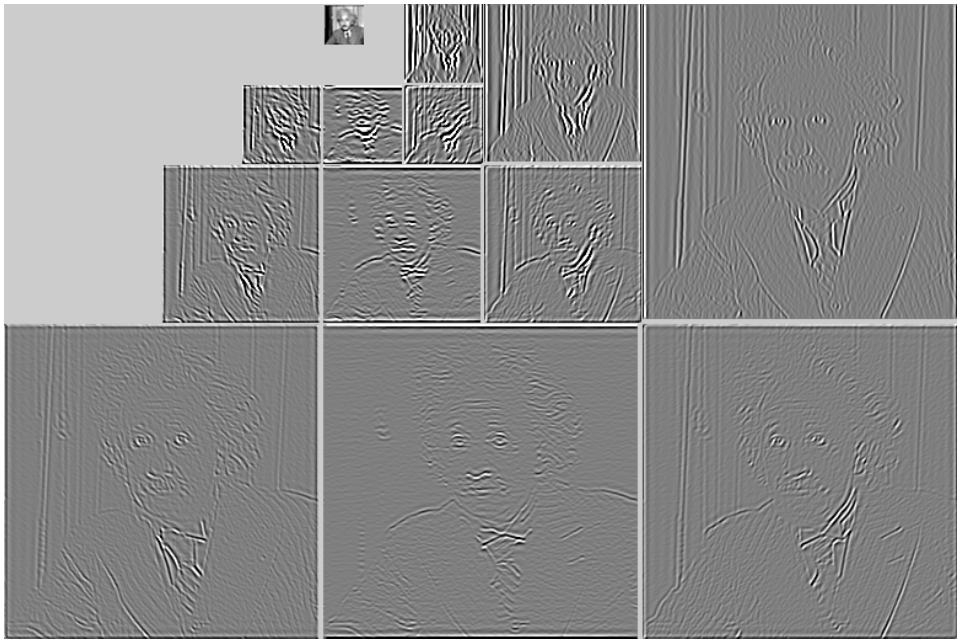
There are two functions you can call from the toolbox, namely `buildSpyr.m` constructed in spatial domain, and `buildSFpyr.m` constructed in frequency domain. The reconstruction of the pyramid obtained by `buildSpyr.m` is not perfect. We shall use `buildSFpyr.m` for the sake of better reconstruction. The following sample code shows pyramid construction, visualization and reconstruction:

```
[pyr,pind] = buildSFpyr(im,3,3); % 3 levels, 4 orientation bands
showSpyr(pyr,pind);
res = reconSFpyr(pyr,pind);
imshow(res);
```

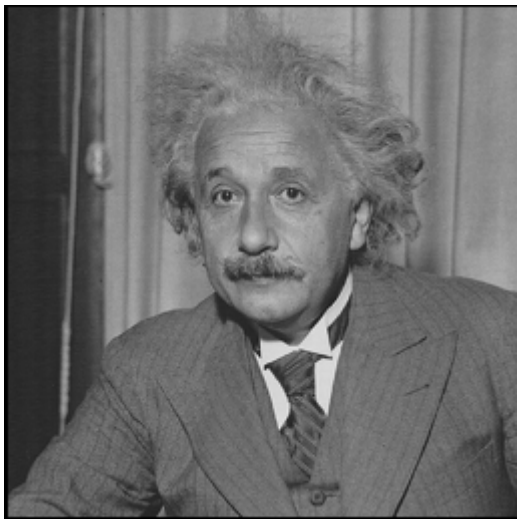
Display the pyramid of the original image. Also display the new pyramid where all the horizontal sub-bands are set to be zero, and the corresponding reconstructed image.

Solution. Run the following code

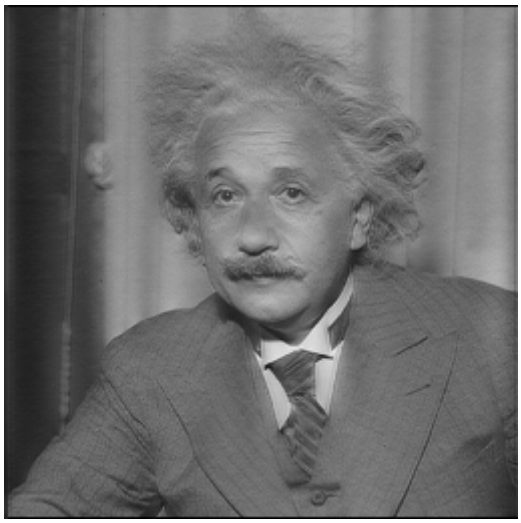
```
im=im2double(imread('einstein.jpg'));
figure;imshow(im);

[pyr,pind] = buildSFpyr(im,3,3); % 3 levels, 4 orientation bands
showSpyr(pyr,pind);
res = reconSFpyr(pyr,pind);

Indices=[2,6,10];

for i=1:3
    res=pyrBand(pyr,pind,Indices(i));
    pyr=setPyrBand(pyr,pind,res*0,Indices(i));
end
```



(a)



(b)



(c)

Figure 3: Steerable pyramid (a) of image (b). The reconstruction of the pyramid (c) where all the horizontal bands (corresponding to vertical features) are set to be zero.

```
res = reconSFpyr(pyr,pind);
figure;imshow(res);
```

See Figure 3 for results. □

2.2 *Coring*

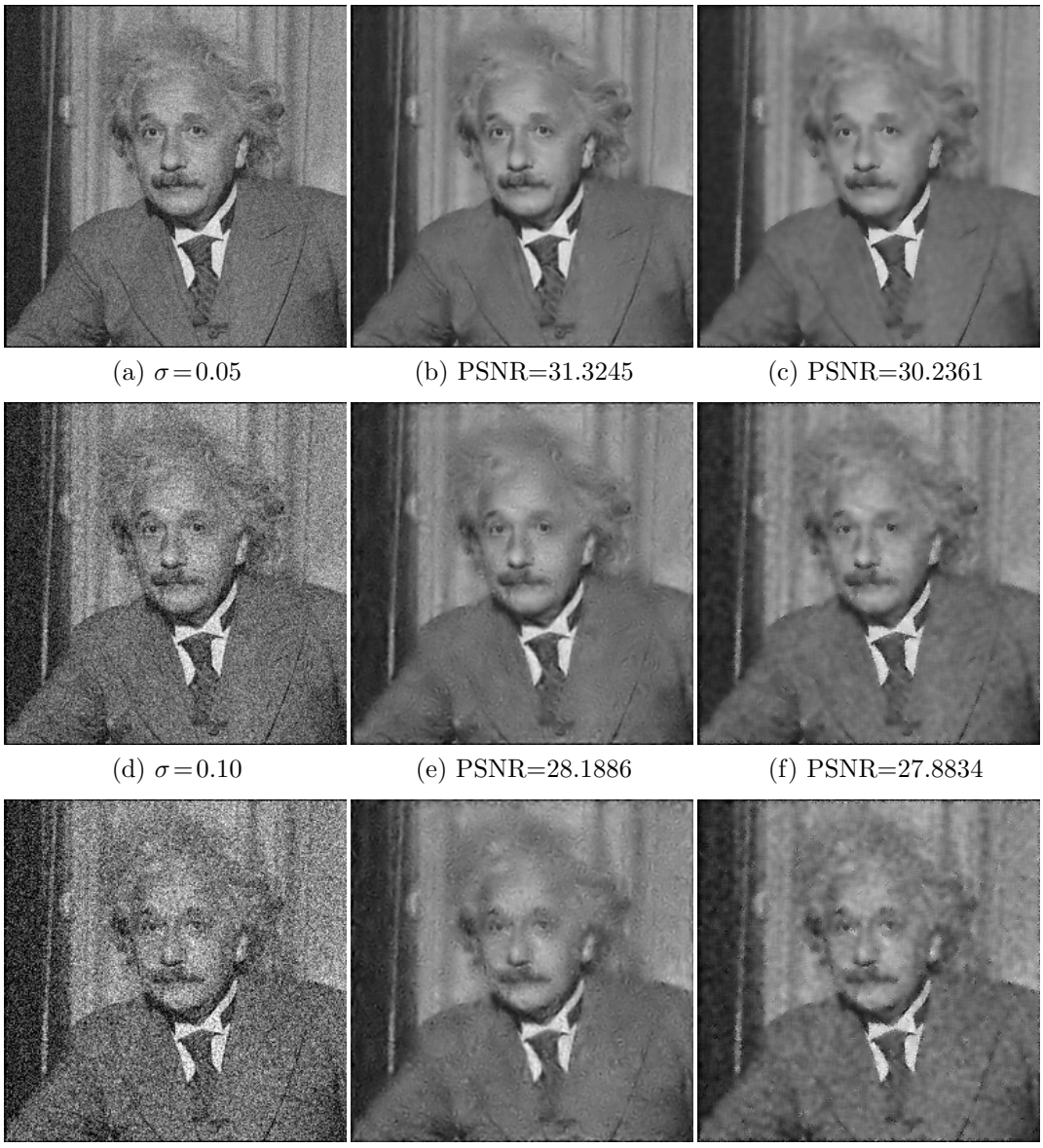
The goal is now to perform denoising using coring. Load `einstein.jpg`. Use MATLAB function `randn` or `imnoise` to synthesize AWGN (additive white Gaussian noise) to the original image. We are going to denoise it by applying a non-linearity to the high-frequency bands. That is, for each coefficient we will apply a function f that lowers the low-amplitude coefficients (that are assumed to predominantly correspond to noise) and preserves the high-amplitude coefficients. The cutoff will depend on the amount of noise, which will be an input to your function. Rather than implementing a method from the literature, we want you to experiment with coring functions. Design a non-linear functions that take the noise level as a parameter, and, given a pyramid coefficient, removes low-amplitude coefficients but keeps high-amplitude ones. That is, design a function $f(\sigma, x)$ and apply it to each coefficient x using the variance of the AWGN as σ. You can also decide to take the scale as a parameter. This exercise will be graded in part on your difference with the ground truth.

Use 3 levels and 3 subbands to build steerable pyramid. Compute PSNR (Peak Signal-to-Noise Ratio) using the provided function `PSNR.m`. Please give the PSNR metric and denoised image for $\sigma = 0.05$, 0.10 and 0.15. Compare your algorithm with Wiener filtering, using MATLAB function `wiener2`. Display the noise image, your denoising result, and the result from `wiener2` side by side. You may also change the number of levels and subbands to see how the performance may change.

Solution. We design a coring function

$$f_i(x) = \left|\tanh^2\left(\frac{x}{\gamma\hat{\sigma}_i}\right)\right| x \tag{1}$$

for the ith subband. A same dimension pyramid is built using a pure noise image and $\hat{\sigma}_i$ is estimated from the ith subband of the noise pyramid. $\gamma = 5$ for $i = 1$ and $\gamma = 3$ otherwise. The shape of the coring function is displayed in Figure 5.



(a) $\sigma = 0.05$ (b) PSNR=31.3245 (c) PSNR=30.2361

(d) $\sigma = 0.10$ (e) PSNR=28.1886 (f) PSNR=27.8834

(g) $\sigma = 0.15$ (h) PSNR=26.3819 (i) PSNR=25.8132

Figure 4: Left column: noise contaminated images. Middle column: denoised results using wavelet coring. Right column: denoised results using Wiener filtering (MATLAB function `wiener2`). Though the wavelet coring algorithm here is not optimized, it does a better job than Wiener filtering both visually and in terms of PSNR value.

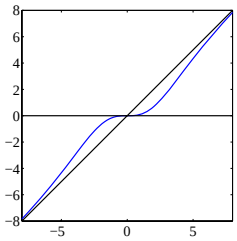


Figure 5: Coring function used in denoising.

Run the sample code

```
im=im2double(imread('einstein.jpg'));
sigma_n=0.10;
noise=randn(size(im))*sigma_n;
imn=im+noise;
figure;imshow(imn);

% 3 levels, 3 orientation bands
[pyr,pind] = buildSFpyr(imn,3,2);
nsubbands=size(pind,1)-1;

% create the pyramid for the noise
[pyrn,pindn] = buildSFpyr(noise,3,2);

for i=1:nsubbands
    res=pyrBand(pyr,pind,i);
    resn=pyrBand(pyrn,pindn,i);
    % estimate the sigma of the noise on the subband
    sigma=std(resn(:))
    if i~=1
        gamma=3;
    else
        gamma=5;
    end
    res=abs(tanh(res/(gamma*sigma)).^2).*res;
    pyr=setPyrBand(pyr,pind,res,i);
end

res = reconSFpyr(pyr,pind);
figure;imshow(res);

h=PSNR(res,im);
```

The denoising results are displayed in Figure 4. Note that PSNR decreases

as the noise level increases.

The method provided here is certainly not the optimal way to design subband coring function. Please refer to Simoncelli and Adelson's ICIP'96 paper "noise removal via Bayesian wavelet coring" for a Bayesian approach of designing optimal coring function. □

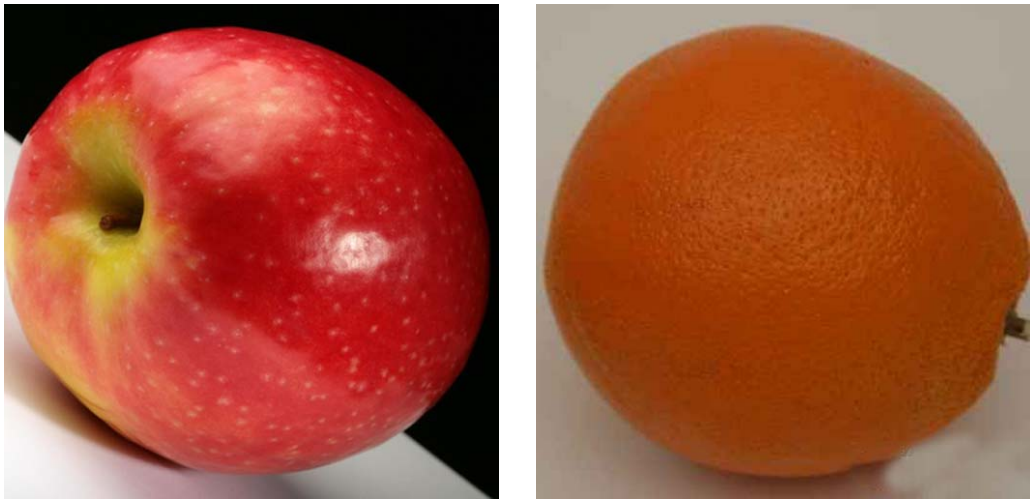
2.3 *Multiscale blending – 6.882 only*

Use the provided `orange.jpg` and `apple.jpg` to generate two "new" fruits: *orange-apple*, where the left side is orange and the right side is apple, and *apple-orange*, where the left side is apple and the right side is orange. Load `mask.bmp`. Use Burt and Adelson's method to blend the two images on Laplacian pyramid using the appropriately smoothed and down-sampled mask. You can imagine that a Gaussian pyramid is built for the mask and the mask at the same level as the Laplacian pyramid is used for blending. You may use MATLAB function `imfilter` and `imresize` for filtering and downsampling. Display the original and "new" fruits side by side. You may use the function `buildLpyr` and `reconLpyr` in the toolbox.

(Extra credit) Create a fun photomontage based on this principle, (e.g. eye in the hand in the original Burt and Adelson paper).

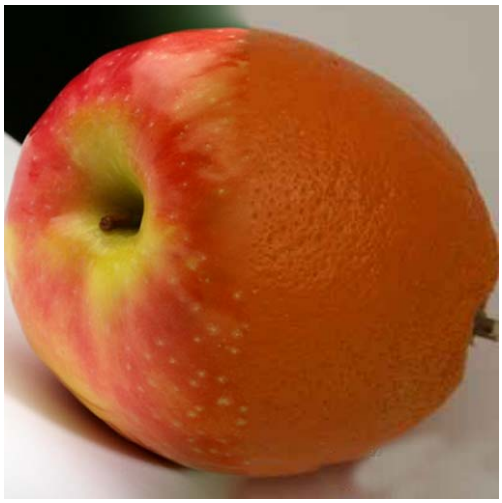
Solution. Run the following code

```
im1=im2double(imread('apple.jpg'));
im2=im2double(imread('orange.jpg'));
figure;imshow(im1);
figure;imshow(im2);

nlevels=7;
mask0=im2double(imread('mask.bmp'));
mask0=imfilter(double(mask0),fspecial('gaussian',5,1),'same','replicate');
% for each color channel
for i=1:3
    [pyr1,pind] = buildLpyr(im1(:,:,i),nlevels);
    [pyr2,pind] = buildLpyr(im2(:,:,i),nlevels);
    pyr=pyr1;
    pyr_r=pyr;

    mask=mask0;
    % for each level
    for j=1:nlevels
```



(a) Apple

(b) Orange



(c) Apple-orange

(d) Orange-apple

Figure 6: Multi-scale blending of apple (a) and orange (b). The blending results (c) and (d) are obtained by blending on Laplacian pyramid.

```
            res1=pyrBand(pyr1,pind,j);
            res2=pyrBand(pyr2,pind,j);
            res=res1.*mask+res2.*(1-mask);
            res_r=res2.*mask+res1.*(1-mask);
            pyr=setPyrBand(pyr,pind,res,j);
            pyr_r=setPyrBand(pyr_r,pind,res_r,j);
            if j~=nlevels
                mask=imfilter(mask,fspecial('gaussian',5,1),'same','replicate');
                mask=imresize(mask,0.5);
            end
        end

        Im(:,:,i)=reconLpyr(pyr,pind);
        Im_r(:,:,i)=reconLpyr(pyr_r,pind);
    end
    figure;imshow(Im);
    figure;imshow(Im_r);
```

See results in Figure 6. □

2.4 *Heeger and Bergen (Extra credit)*

Implement the pyramid texture synthesis by Heeger and Bergen [1995]. Write a MATLAB function to synthesize a new texture by matching the pixel and pyramid coefficient histograms on steerable pyramid for a given input texture. You can try the texture images in `texture.zip` (not necessarily all of them). Because they are grayscale textures you do not need to care about color. Show the input and synthesized images side by side. Show also the histograms from the input and synthesized. Do you find it easy to match histograms? Does the histogram matching depend on the input texture? Feel free to change the number of pyramid levels, number of subbands, and number of bins to see how these parameters may effect the synthesis. You only need to hand in the best synthesized results you obtain.

Solution. . Three MATLAB functions are designed for texture synthesis. `gethist`: to get histograms from the subbands (steerable pyramid) of an image; `heegerbergen`: to synthesize texture image given the histograms; `disphistograms`: to display histograms.
`gethist.m`:

```
%----------------------------------------
```

```
% function to get histograms from an image
%-------------------------------------------
function Histograms=gethist(im,nbins,nlevels,nbands)
% default parameters
if isfloat(im)~=1
    im=im2double(im);
end
if exist('nlevels')~=1
    nlevels=3;
end
if exist('nbands')~=1
    nbands=4;
end
if exist('nbins')~=1
    nbins=256;
end

% build steerable pyramid
[pyr,pind]=buildSFpyr(im,nlevels,nbands-1);

% intensity histogram
Histograms(1).min=0;
Histograms(1).max=1;
Histograms(1).hist=imhist(im,nbins)/prod(size(im));

k=1;
for i=1:nlevels+1
    if i<=nlevels
        n=nbands+(i==1);
    else
        n=1;
    end
    for j=1:n
        res = pyrBand(pyr,pind,k);
        Min=min(res(:));
        Max=max(res(:));
        Histograms(k+1).min=Min;
        Histograms(k+1).max=Max;
        Histograms(k+1).hist=imhist((res-Min)/(Max-Min),nbins)/prod(pind(k,:));
        k=k+1;
    end
    nbins=round(nbins/2);
end
```

heegerbergen.m

```
%-------------------------------------------
% Heeger and Bergen texture synthesis
%-------------------------------------------
function im=heegerbergen(Histograms,synDim,nlevels,nbands,nIterations)
% default values of the parameters
if exist('synDim')~=1
    synDim=[256,256];
end
if exist('nlevels')~=1
    nlevels=3;
end
if exist('nbands')~=1
    nbands=4;
end
if exist('nIterations')~=1
    nIterations=30;
end

% initialize by rand noise
im=rand(synDim);

for i=1:nIterations
    im=min(max(im,0),1);
    im=histeq(im,Histograms(1).hist*prod(synDim));
    % build pyramid
    [pyr,pind]=buildSFpyr(im,nlevels,nbands-1);
    for k=1:size(pind,1)
        HMin=Histograms(k+1).min;
        HMax=Histograms(k+1).max;
        res = pyrBand(pyr,pind,k);
        Min = min(res(:));
        Max = max(res(:));
        res = (res-Min)/(Max-Min); % normalise
        res = histeq(res,Histograms(k+1).hist*prod(pind(k,:)));
        res = res*(HMax-HMin)+HMin;
        pyr = setPyrBand(pyr,pind,res,k);
    end
    % restore the image from the pyramid
    im = reconSFpyr(pyr,pind);
end
```

disphistograms.m

```
%-------------------------------------------------
% function to display histograms
%-------------------------------------------------
```

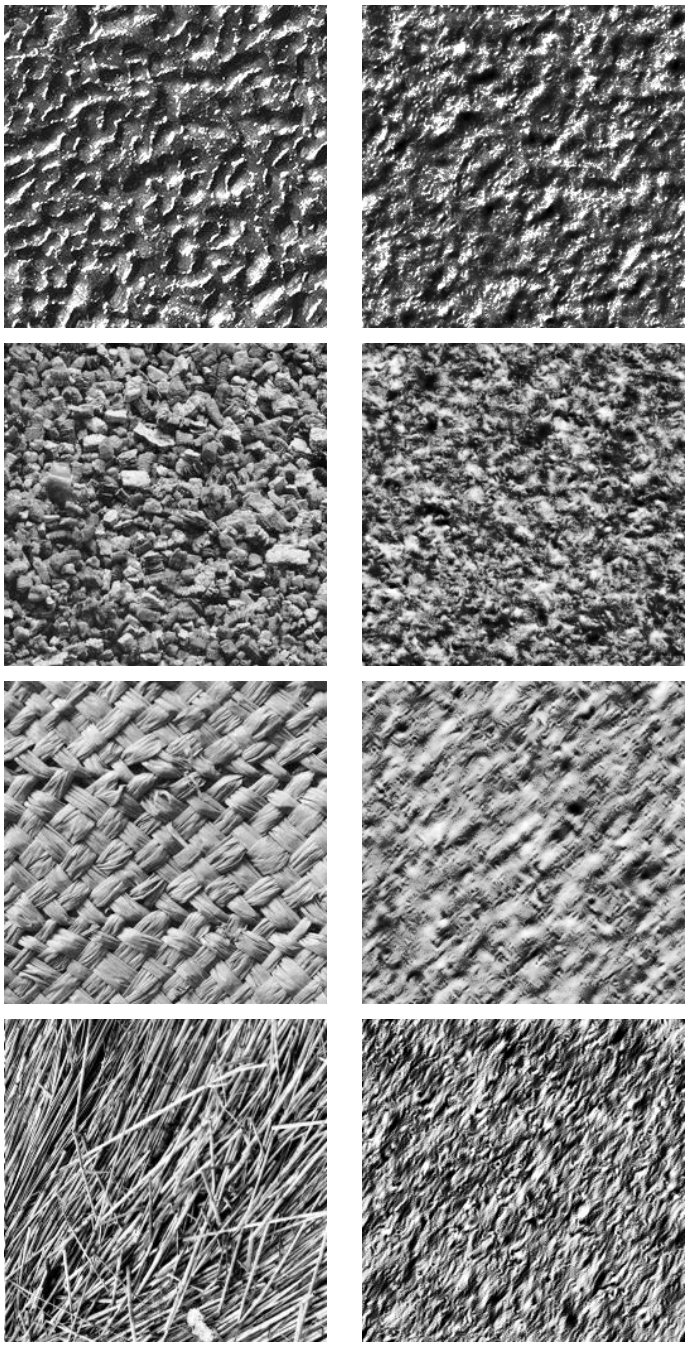


Figure 7: Heeger and Bergen texture synthesis. Left: input texture. Right: synthesized by matching histograms on steerable pyramid.

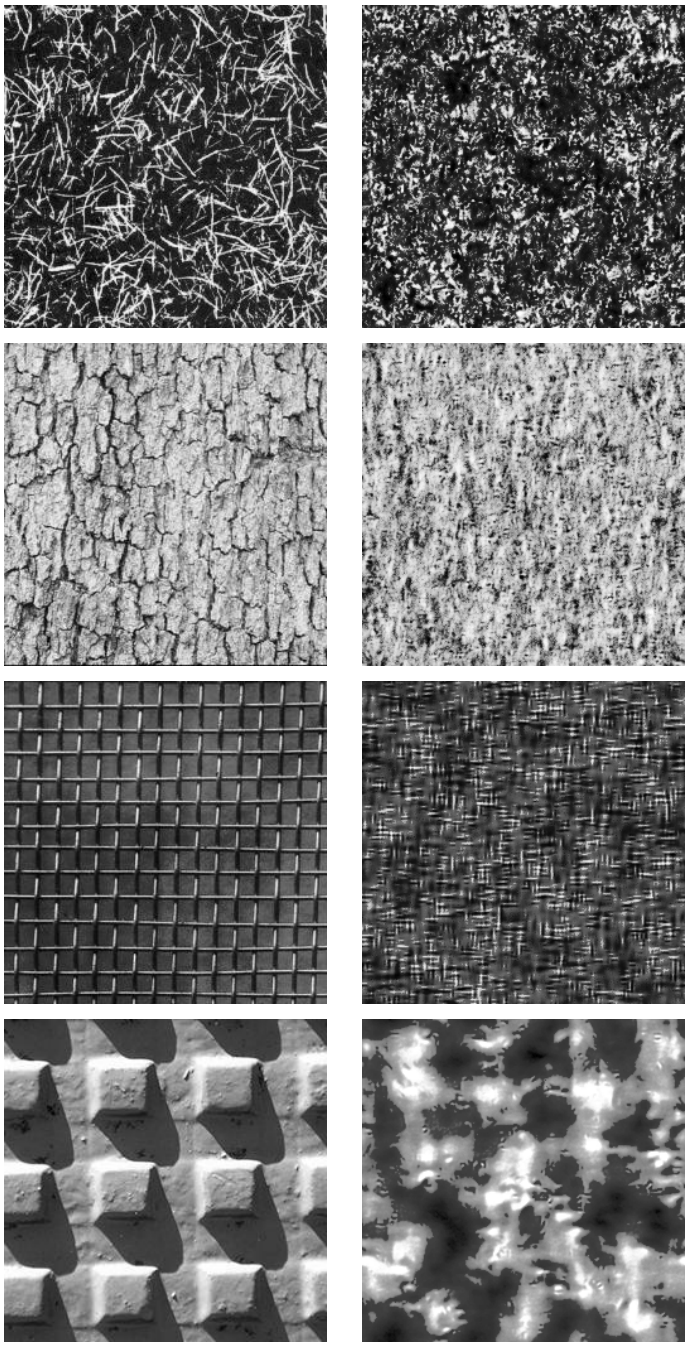


Figure 8: Heeger and Bergen texture synthesis. Left: input texture. Right: synthesized by matching histograms on steerable pyramid.

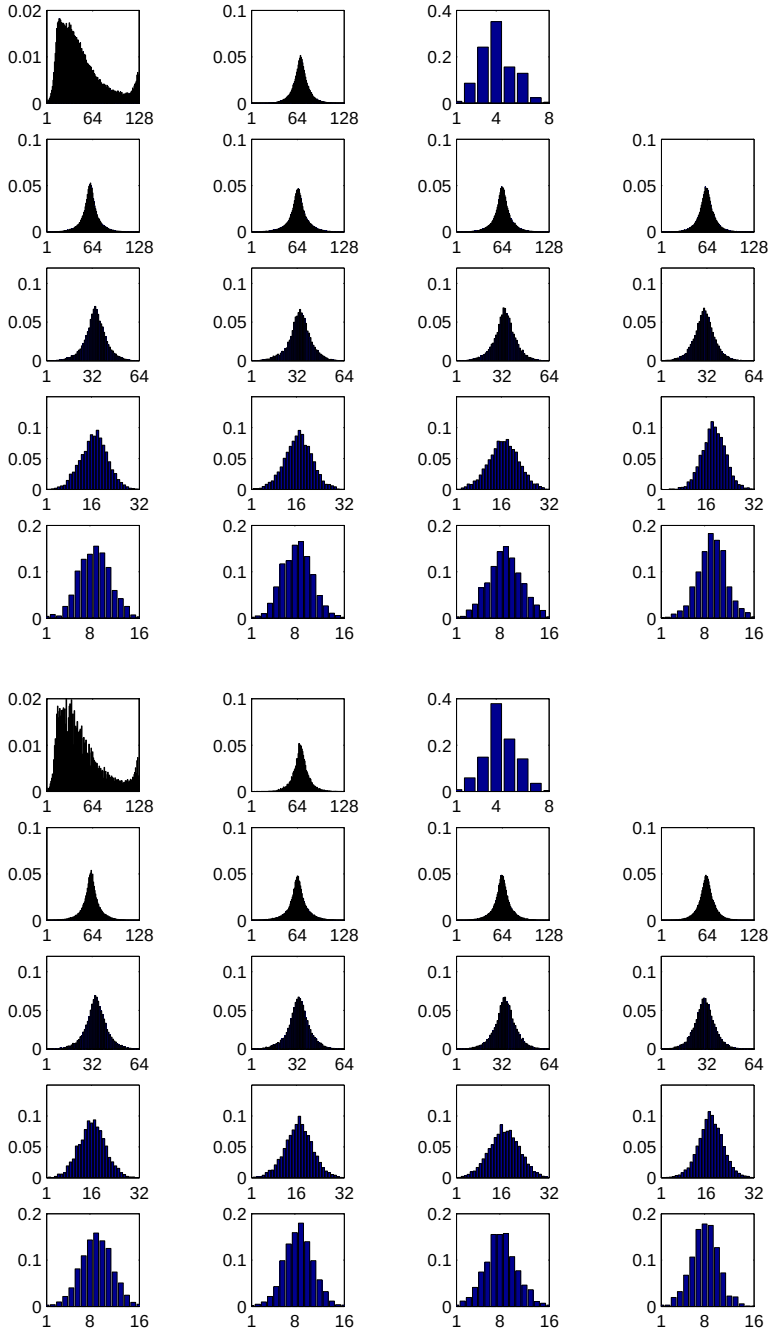
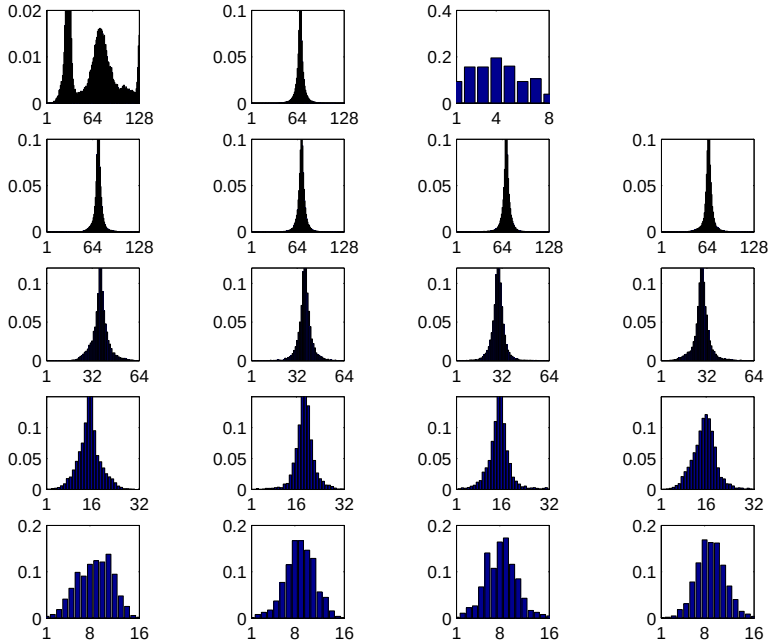
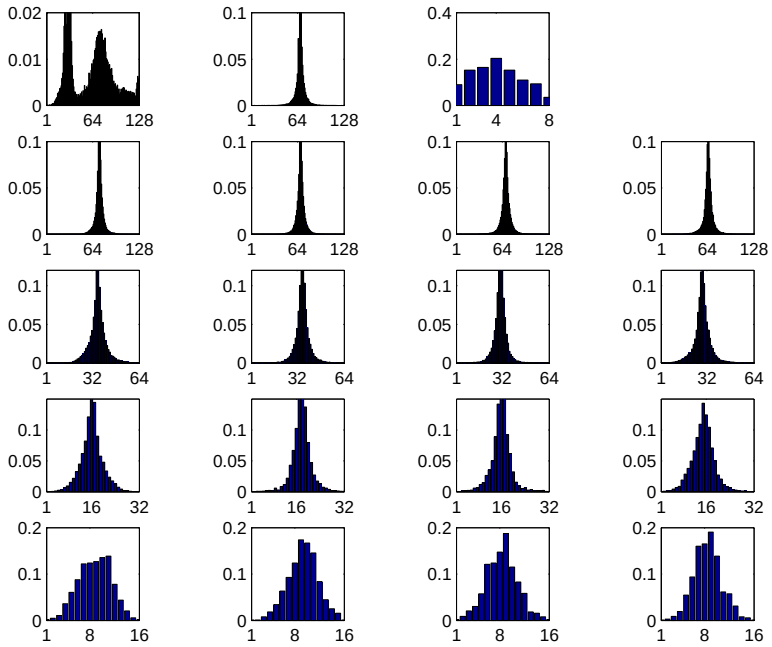


Figure 9: Histogram matching for texture (a).



(a) Input



(b) Synthesized

Figure 10: Histogram matching for texture (h).

```
function disphistograms(Histograms,nlevels,nbands)
m=nlevels+1;
n=max(3,nbands);
subplot(m,n,1);plothistogram(Histograms(1),0.02);
subplot(m,n,2);plothistogram(Histograms(2),0.1);%
subplot(m,n,3);plothistogram(Histograms(end),0.4);
k=3;
y_max=[0.1,0.12,0.15,0.2,0.24];
for i=1:nlevels
    for j=1:n
        subplot(m,n,i*n+j);
        plothistogram(Histograms(k),y_max(i));
        k=k+1;
    end
end

function plothistogram(histogram,y_max)
bar(histogram.hist);
n=length(histogram.hist);
xlim([1,n]);
ylim([0,y_max]);
axis square;
set(gca,'XTick',[1,n/2,n]);
```

We set 3 levels of pyramid and 4 subbands in texture synthesis. The number of bins is set to be 128 at the first level, and is divided by 2 as going to a higher level in the pyramid. We list the original and synthesized images of the eight examples in `texture.zip`, as shown in Figure 7 and 8. The visual "successfulness" of synthesis is somewhat descending from (a) to (h). However, when we look at the histogram matching in Figure 9 for (a) and Figure 10 for (h) we can see that the histograms are matched almost equally well. This implies that merely matching marginal distributions (histograms) on band-pass filters are not enough. Please refer to Portilla and Simoncelli's work (IJCV'00) for more statistics included for texture synthesis. □

Problem 3 *Matting*

Given an input image, we want to extract a foreground element and the corresponding alpha matte. We are given a trimap that coarsely classifies pixels as foreground, background, and unknown. You are going to implement a natural image matting algorithm, a simplified version of Chuang et al's

CVPR'01 paper (Section 3). In a nutshell, you must characterize the color distribution of the foreground and background pixels and solve for the values of α, F_g and B_g in a Bayesian framework.

The algorithm will proceed in two steps: 1. gather the color statistics from the foreground and background pixels from the trimap and model them using Gaussians in RGB space. 2. For each pixel in the unknown region, compute the foreground and background color as well as the alpha. For this, we will solve an MAP (maximum a posteriori) problem based on the above Gaussians.

Load `toy.jpg` and `trimap.png`. In the trimap, stored in `trimap.png`, you can regard intensity value $I > 0.95$ as foreground, $I < 0.05$ as background, and the rest pixels are the ambiguous area where an alpha value α, foreground color F_g and background color B_g need to be estimated per pixel. The color distributions will be extracted from all the pixels in the background and foreground regions (once and for all). Compute the mean and covariance for the foreground (resp. background) pixels.

Please see Chuang's paper for the MAP computation of the three unknowns. You need to iterate between solving α and estimating F_g and B_g.

Set $\sigma_C = 0.01$ at first, and try to vary it when the code is debugged. You may need to solve linear equation in this problem. Use MATLAB notation `x=A\b` to solve linear system $Ax = b$. You may loop around all the pixels. If the algorithm takes 10 iterations, it should not take more than one minute. Display the alpha matte.

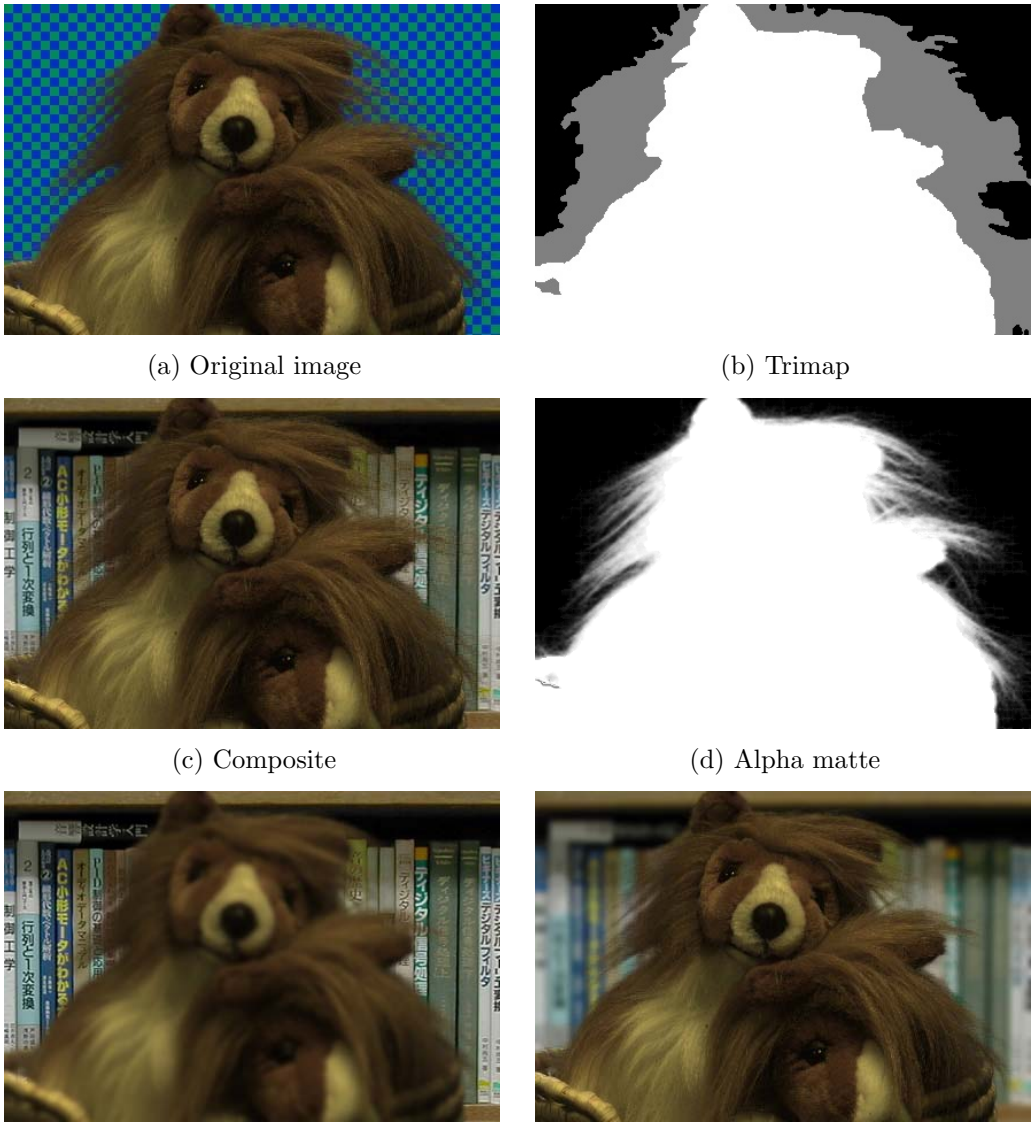
Load another image `bookshelf.jpg`. Compose the foreground to this image with the estimated α and F_g's. Does the composition look natural to you? Can you see any artifacts?

Hand in the alpha matte and composite image.

(Extra credit) You can blur the background or foreground in composition to obtain effects of focus and field of depth. Play with blurring kernels and see if they look natural.

Solution. Run the following code

```
im=im2double(imread('toy.jpg'));
mask=im2double(imread('mask.png'));
figure;imshow(im);
figure;imshow(mask);

% parameters
```



(a) Original image (b) Trimap

(c) Composite (d) Alpha matte

(e) Blurred foreground (f) Blurred background

Figure 11: Bayesian matting. After estimating alpha matte (d) from the image (a) and trimap (b), we can compose the foreground to other background and obtain (c). By blurring foreground (e) and background (f) in the composition, we may obtain effects of focus and depth.

```
sigma_C=0.01;

% the index of the foreground, background and alpha area
fgIndex=find(mask>=0.95);
bgIndex=find(mask<=0.05);
alphaIndex=find(mask<0.95 & mask>0.05);

% concatenate the color to matrices
[height,width,n]=size(im);
X=reshape(im,[height*width,n]);

X_fg=X(fgIndex,:);
X_bg=X(bgIndex,:);
C=X(alphaIndex,:);

% estimate color distribution (mean and covariance)
mu_fg=mean(X_fg,1)';
invcov_fg=inv(cov(X_fg));

mu_bg=mean(X_bg,1)';
invcov_bg=inv(cov(X_bg));

%-----------------------------------------------------------
% Matting algorithm
%-----------------------------------------------------------

% initialize alpha map
Im=zeros(height,width);
Im(fgIndex)=1;
Im=imfilter(Im,fspecial('gaussian',7,1),'same','replicate');
alpha=Im(alphaIndex);

% iterative algorithm
nPixels=length(alphaIndex);

for k=1:20
    % solve the linear system
    for i=1:nPixels
        A(1:3,1:3)=invcov_fg+alpha(i).^2/sigma_C^2*eye(3);
        A(1:3,4:6)=alpha(i)*(1-alpha(i))/sigma_C^2*eye(3);
        A(4:6,1:3)=A(1:3,4:6);
        A(4:6,4:6)=invcov_bg+eye(3)*(1-alpha(i))^2/sigma_C^2;
        b(1:3,1)=invcov_fg*mu_fg+C(i,:)'*alpha(i)/sigma_C^2;
        b(4:6,1)=invcov_bg*mu_bg+C(i,:)'*(1-alpha(i))/sigma_C^2;
        x(:,i)=A\b;
```

```
        end
        F=x(1:3,:)';
        B=x(4:6,:)';

        % estimate alpha
        alpha=sum((C-B).*(F-B),2)./sum((F-B).^2,2);
        alpha=min(max(alpha,0),1);
    end

    % merge to get the complete alpha map
    alphamap=zeros(height,width);
    alphamap(fgIndex)=1;
    alphamap(alphaIndex)=alpha;
    figure;imshow(alphamap);

    Im=im2double(imread('bookshelf.jpg'));
    % blur the background to increase depth
    Im=imfilter(Im,fspecial('gaussian',3,.5),'same','replicate');
    cAlphaMap=repmat(alphamap(:),[1,3]);
    imc=X_fg.*cAlphaMap+reshape(Im,[height*width,3]).*(1-cAlphaMap);
    imc=reshape(imc,[height,width,3]);
    figure;imshow(imc);
```

See the results in Figure 11. □