Solution to Problem Set 3

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

For this homework we only give solution to problem 2 and 3.

Problem 2 *Tone Mapping Using the Bilateral Filter*

We now want to reduce the dynamic range of our image to display it on a low-dynamic range device. For this, you will implement a simplified (read: slow) version of Durand and Dorsey's 2002 algorithm. This algorithm only modifies the luminance of an image: it reduces the contrast of the large-scale variation of luminance but preserves local detail. For this, it decomposes the luminance image into a large-scale (a.k.a. base) layer and a detail layer using the bilateral filter. All computation on luminance is performed in the log domain.

The bilateral filter blurs an image except across strong edges. For each pixel, the output is a weighted average of the neighboring pixels where the weight depends on both the spatial distance and the intensity difference:

$$J_s = \frac{1}{k(s)} \sum_{p \in N_s} \exp\{-\frac{(p-s)^2}{2\sigma_d^2}\} \exp\{-\frac{(I_p - I_s)^2}{2\sigma_r^2}\} I_p$$

where $k(s)$ is a normalization term

$$k(s) = \sum_{p \in N_s} \exp\{-\frac{(p-s)^2}{2\sigma_d^2}\} \exp\{-\frac{(I_p - I_s)^2}{2\sigma_r^2}\}.$$

N_s is the neighborhood of s. s and p are both coordinates of image lattice. So there are three parameters for bilateral filtering, the half size of the neighborhood w (the neighborhood is $(2w+1) \times (2w+1)$), spatial standard deviation σ_d and range standard deviation σ_r.

Here is the pseudo code of applying bilateral filtering for tone mapping.

```
input intensity= 1/61*(R*20+G*40+B) r=R/(input intensity), g=G/input
intensity, B=B/input intensity log(base)=Bilateral(log(input
intensity)) log(detail)=log(input intensity)-log(base)
compressfactor=log(output range)/(max(log(base))-min(log(base)));
log_offset=-max(log(base))*compressfactor; log (output
intensity)=log(base)*compressionfactor+log_offset+log(detail) R
output = r*exp(log(output intensity)), etc.
```

The main parameter of this code is output range, which depends the amount of remaining large-scale contrast that you want in the output. A value of 10 to 30 works well.

(a) Write a MATLAB function `Im=imbltflt(im,wsize,sigma_d,sigma_r)` to implement bilateral filtering. Argument `im` is the input image. Parameter `wsize`, `sigma_d` and `sigma_r` correspond to w, σ_d and σ_r in the above equations. Try to avoid writing four loops. Two loops (over image coordinate) are OK. Load image `einstein.jpg`, add AWGN with $\sigma = 0.05$ (as you did in ps2), and apply bilateral filter to denoise the noise contaminated image. Use parameter $w = 5,\ \sigma_d = 2,\ \sigma_r = 0.12$. Display the noisy image and denoised image.

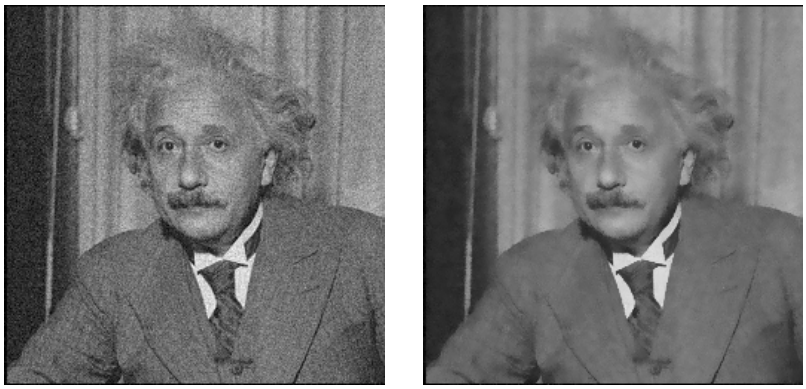
(b) Load HDR image `vinesunset.hdr` using the enclosed MATLAB code `read_rle_rgbe.m` (from `http://www.cis.rit.edu/mcsl/icam/hdr/`). Implement tone mapping using Gaussian filtering, `fspecial('gaussian',21,8)`, instead of bilateral filtering. Set output range to be 30. Display the LDR image. Do you see any artifacts?

(c) Now replace the Gaussian filter with the bilateral filter. Set the parameter $w = 10, \sigma_d = 8, \sigma_r = 0.2$. Display the LDR image. Do the halo artifacts disappear? Compare the result with (b).

(d) (Extra credit) Implement the uncertainty "fix"; implement fast bilateral filtering; implement the trilateral filter

Solution. Here is the implementation of bilateral filtering

```
function Im=imbltflt(im,wsize,sigma_d,sigma_r)
[height,width,n]=size(im);
if n>1
    error('grayscale image only!');
end
```



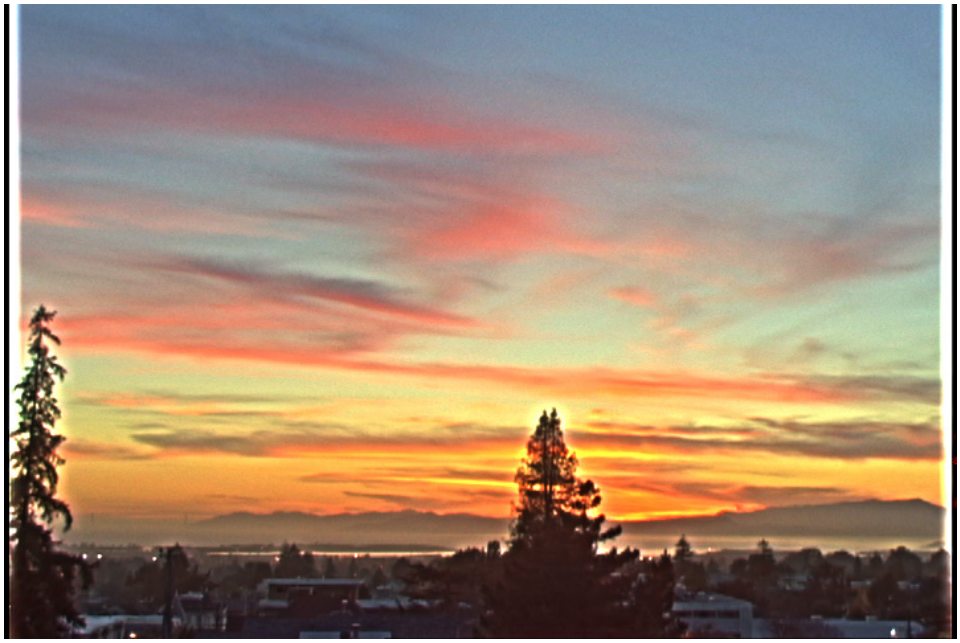
(a) Noise image with $\sigma = 0.05$ (b) Denoised by bilateral filtering

Figure 1: Bilateral filtering can be used for denoising.

```
Gfilter=fspecial('gaussian',wsize*2+1,sigma_d);

for i=1:height
    for j=1:width
        x1=max(1,j-wsize);
        x2=min(width,j+wsize);
        y1=max(1,i-wsize);
        y2=min(height,i+wsize);
        patch=im(y1:y2,x1:x2);
        Gkernel=Gfilter((wsize+1+y1-i):(wsize+1+y2-i),(wsize+1+x1-j):(wsize+1+x2-j));
        fkernel=exp(-(patch-im(i,j)).^2/(2*sigma_r^2)).*Gkernel;
        fkernel=fkernel/sum(fkernel(:));
        Im(i,j)=sum(sum(patch.*fkernel));
    end
end
```

The denoising results of applying bilateral filtering is shown in Figure 1. For part (b) and (c) see the following code.

```
im=read_rle_rgbe('vinesunset.hdr');

isbilateral=1;

% intensity
Im=(im(:,:,1)*20+im(:,:,2)*40+im(:,:,3))/61+0.001;
im_r=im(:,:,1)./Im;
im_g=im(:,:,2)./Im;
im_b=im(:,:,3)./Im;
```



(a) LDR obtained from Gaussian filtering of the base log image.

(b) LDR obtained from Bilateral filtering of the base log image.

Figure 2: LDR display of the HDR image. There are strong halo effect from mere Gaussian filtering.

```
log_Im=log(Im);
if isbilateral
    fprintf('Bilateral filtering...');tic;
    log_base=imbltflt(log_Im,10,8,0.1737);
    toc
else
    log_base=imfilter(log_Im,fspecial('gaussian',21,8),'same','replicate');
end
log_detail=log_Im-log_base;
log_max=max(log_base(:));
log_min=min(log_base(:));
compressfactor=log(30)/(log_max-log_min);
log_offset=-log_max*compressfactor;
log_output=log_base*compressfactor+log_offset+log_detail;

Im_output=exp(log_output);

im_disp(:,:,1)=im_r.*Im_output;
im_disp(:,:,2)=im_g.*Im_output;
im_disp(:,:,3)=im_b.*Im_output;

figure;imshow(im_disp);
```

The method is controlled by a switch variable `isbilateral`. The results are displayed in Figure 2. There are very strong halo effects in the results obtained via Gaussian filtering. These artifacts disappear when bilateral filtering is applied. □

Problem 3 *Poisson Image Editing*

Your goal is to create a photomontage by pasting an image region onto a new background using Poisson image editing. Please read Pérez et al's *SIGGRAPH* 2003 paper "Poisson Image Editing". Focus on Section 2 and 3.1. Implement the algorithm from Equation (7) to (11).

The main task of Poisson image editing is to solve the huge linear system $Ax = b$ in Equation (7). You can explicitly represent A using sparse matrices, and solve it using MATLAB command \. This method should work for the provided examples, but cannot scale up for big masks. We do not recommend this approach, though it is fine for you to use it.

We recommend conjugate gradient (CG) approach. The basic idea of con-

jugate gradient method is that the solution to $Ax = b$ can be represented by a set of vectors, and these vectors are A *orthogonal* between one and another. It is mandatory that matrix A is positive definite. In energy minimization problems A is always semi-positive definite. In our problem matrix A is the second-order derivative matrix which is positive definite.

There are two reasons that (preconditioned) conjugate gradient method is favored in image processing/editing

- For image processing matrix A is huge. The space complexity for LU decomposition is $O(n^3)$, too big for images. But CG only requires matrix multiplication in each iteration, which is essentially filtering. Space complexity is $O(n^2)$
- CG gives good results within a few iterations. Normally the result is good enough if the number of iteration is image width or height, or even less. So CG is efficient. It can be much faster, converging in less than ten iterations, if a good preconditioner is used (not required for the homework).

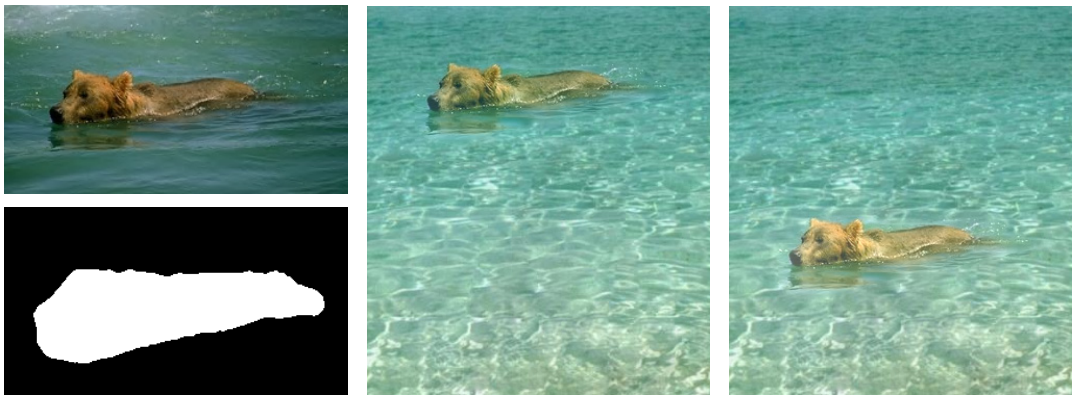
The conjugate gradient method can be summarized in the following pseudo code

Given A, b and an initial guess of x, solve $Ax = b$ iteratively

1. $r = b - Ax$ % residual
2. for $k = 1 : n$
   - $\rho_k = \|r\|^2$
   - If $k = 1$ then $p = r$ else $p = r + \frac{\rho_k}{\rho_{k-1}} p$ % direction
   - $q = Ap$
   - $\alpha = \rho_k / (p^T q)$
   - $x = x + \alpha p$ % new point
   - $r = r - \alpha q$ % residual
3. Output x

Your task is to replace Ax by filtering and other MATLAB operations. For this example you will get good enough result with $n = 100$. To debug you can monitor the norm of r which should be monotonically decreasing.

(a) Load image `bear.bmp` and a binary mask `mask.bmp`. Seamlessly paste it onto image `waterpool.bmp` so that the top-left coordinate of the mask



(a) (b) (c)

Figure 3: Poisson image editing. (a) Foreground image with a binary mask. (b) and (c): the foreground is pasted to the background image at top-left location $(20, 1)$ and $(20, 150)$, respectively. Notice how the color of the foreground changes with respect to the ambient colors.

is $(20, 150)$ at the background image. Try also $(20, 1)$ and see how the results are different.

(b) So far we have implemented three methods for seamless pasting, i.e. *multi-scale blending*, *matting* and *Poisson image editing*. Compare the three methods in terms of *advantage*, *limitation*, *speed*, *application scenario* and so on. You may list a table and describe in three paragraphs.

Solution. We give conjugate gradient solver here. Note that the two sides of Equation (7) can be both represented by filtering. The left side is indeed Laplacian filtering with background pixels set to be zero. The Laplacian filter has the form

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}.$$

The first term on the right hand side is filtering to the background pixels with foreground pixels set to be zero, where the filter is

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

From Equation (11) v_{pq} is indeed Laplacian filtering to the foreground image. Run the following code.

```
imfg=im2double(imread('bear.bmp'));
imbg=im2double(imread('waterpool.bmp'));

%x0=20;y0=150;
x0=20;y0=1;

[height_fg,width_fg,n]=size(imfg);
Imbg=imbg(y0:y0+height_fg-1,x0:x0+width_fg-1,:);

fgmask=imread('mask.bmp');
bgmask=1-fgmask;
fgindex=find(fgmask==1);
bgindex=find(bgmask==1);

% Laplacian filter
lapflt=[0 -1 0;-1 4 -1;0 -1 0];
nsumflt=[0 1 0;1 0 1;0 1 0];
nIterations=100;

% for each RGB channel
for l=1:3
    % get the laplacian filtering of the foreground
    fg_lap=imfilter(imfg(:,:,l),lapflt,'same','replicate');

    % initialize the image with the background
    Im=Imbg(:,:,l);
    x=Im(fgindex);

    foo=Im;foo(fgindex)=0;
    b=imfilter(foo,nsumflt,'same','replicate')+fg_lap;

    foo=Im;foo(bgindex)=0;
    r=b-imfilter(foo,lapflt,'same','replicate');
    r=r(fgindex);

    % loop of conjugate gradient method
    for k=1:nIterations
        rou(k)=sum(sum(sum(r.*r)));
        if k==1
            p=r;
        else
            beta=rou(k)/rou(k-1);
```

	multi-scale blending	**matting**	**Poisson editing**
Pros	simple, easy to implement; fast; transition from foreground to background is very smooth	smooth blending at fuzzy boundaries; the interior of the foreground is remained	foreground can adapt to ambient color
Cons	a good alpha map has to be specified	slow; dependent on the trimap;	unexpected foreground details may appear in the pasting
Appl	simple geometry object with sharp boundary	object with fuzzy boundary; difficult to specify alpha map	transparent object; the color of the object needs to be tuned to the background

Table 1: Comparison of three pasting methods.

```
            p=r+beta*p;
        end
        foo=zeros(height_fg,width_fg);
        foo(fgindex)=p;
        foo=imfilter(foo,lapflt,'same','replicate');
        q=foo(fgindex);
        alpha=rou(k)/sum(sum(sum(p.*q)));
        x=x+alpha*p;
        r=r-alpha*q;
        if norm(r(:))<1E-10
            disp(['CG stops at k=' num2str(k)]);
            break;
        end
    end
    Im=Imbg(:,:,l);
    Im(fgindex)=x;
    Im_temp(:,:,l)=Im;
end

Im_comp=imbg;
Im_comp(y0:y0+height_fg-1,x0:x0+width_fg-1,:)=Im_temp;
figure;imshow(Im_comp);
```

See the results in Figure 3. You may see that the color of the foreground adapts to the ambient color from background, depending on where the foreground is pasted.

Part (b). The comparison of the three methods is listed in Table 1. □