EECS 395/495: Introduction to Computational Photography

Homework 4: High Dynamic Range Imaging and Tone-mapping

Student Name: Haikun Liu

Student Number: 2903021

NetID: hlg483

Objective: Explore the focus properties of images

1. Implement an android function to capture a focal stack

Using the following code, we can capture a series of pictures with different focus settings. The minimum focus distance is read from LENS_INFO_HYPERFOCAL_DISTANCE, which equals to 0.6667; and the maximum focus is obtained from LENS_INFO_MINIMUM_FOCUS_DISTANCE, which equals to 10. The pictures are taken with increasing the focus distance by a factor of 1.5 from 1.5 to 7.59375. There are 5 pictures captured with desired quality to recover the final detailed image. They are shown in the pictures Figure 1 to Figure 5.

```
public void captureFocalStack(View v) {
    //TODO:hw5
```

```
    //getting min and max focusdistance
    float minimumLens =
characteristics.get(CameraCharacteristics.LENS_INFO_HYPERFO
CAL_DISTANCE);
    float maximumLens =
characteristics.get(CameraCharacteristics.LENS_INFO_MINIMUM
_FOCUS_DISTANCE);
    //float maximumLens =
characteristics.get(CameraCharacteristics.LENS_INFO_HYPERFO
CAL_DISTANCE);

    Log.e(TAG, "minimumLens: " + minimumLens);
    Log.e(TAG, "maxmimumLens: " + maximumLens);
    //TODO:hw5
    //setting previous lens to be min or max focus distance.
(guess which one it is!)
    float prev_focus = minimumLens;
    Log.e(TAG, "in captureFocalStack");
    //check if capture session is null
    if (mCaptureSession != null) {
        Log.e(TAG, "prevLens: " + prev_focus);
        //TODO: check if focus distance after changing is in
range
        while (prev_focus * 1.5 < maximumLens) {
            //sleep system clock for 20 ms
            SystemClock.sleep(20);
            Log.e(TAG, "in captureFocalStack while loop");
            try {
                //TODO: set current focus to be 1.5 * previous
focus
                float curr_focus = (float) (prev_focus * 1.5);
                focuses.add(curr_focus);
                //build requester
                //CaptureRequest.Builder requester =
mCameraDevice.createCaptureRequest(mCameraDevice.TEMPLATE_M
ANUAL);
                CaptureRequest.Builder requester =
mCameraDevice.createCaptureRequest(mCameraDevice.TEMPLATE_S
TILL_CAPTURE);
                //TODO: turn off auto focus mode for requester
```

```
requester.set(CaptureRequest.CONTROL_AF_MODE,CaptureRequest
.CONTROL_AF_MODE_OFF);
             //add surface as target in requester

requester.addTarget(mCaptureBuffer.getSurface());
             //TODO: set current focus to requester

requester.set(CaptureRequest.LENS_FOCUS_DISTANCE,
curr_focus);
             //set previous focus = current focus
             prev_focus = curr_focus;
             try {
                 // This handler can be null because we
aren't actually attaching any callback
                 //make capture session

                 mCaptureSession.capture(requester.build(),
/*listener*/null, /*handler*/null);
             } catch (CameraAccessException ex) {
                 Log.e(TAG, "Failed to file actual capture
request", ex);
             }
          } catch (CameraAccessException ex) {
             Log.e(TAG, "Failed to build actual capture
request", ex);
          }
      }
   } else {
      Log.e(TAG, "User attempted to perform a capture
outside the session");
   }

}
```

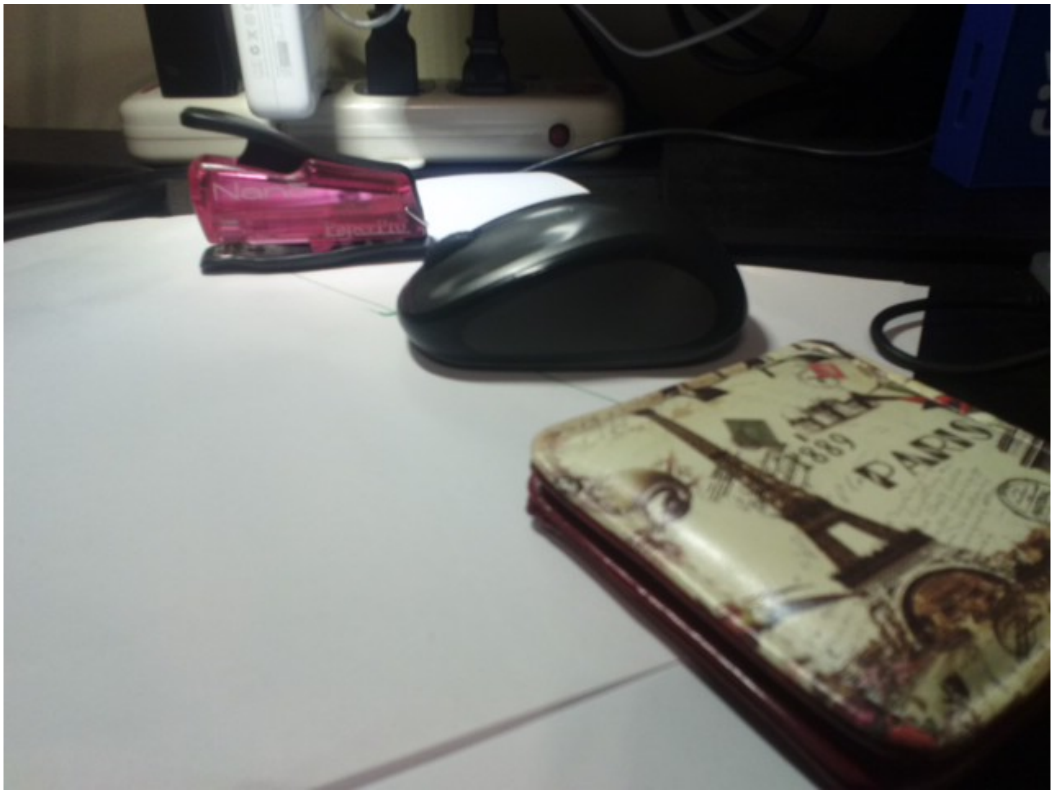


Figure 1: Captured picture with focus distance = 1.5

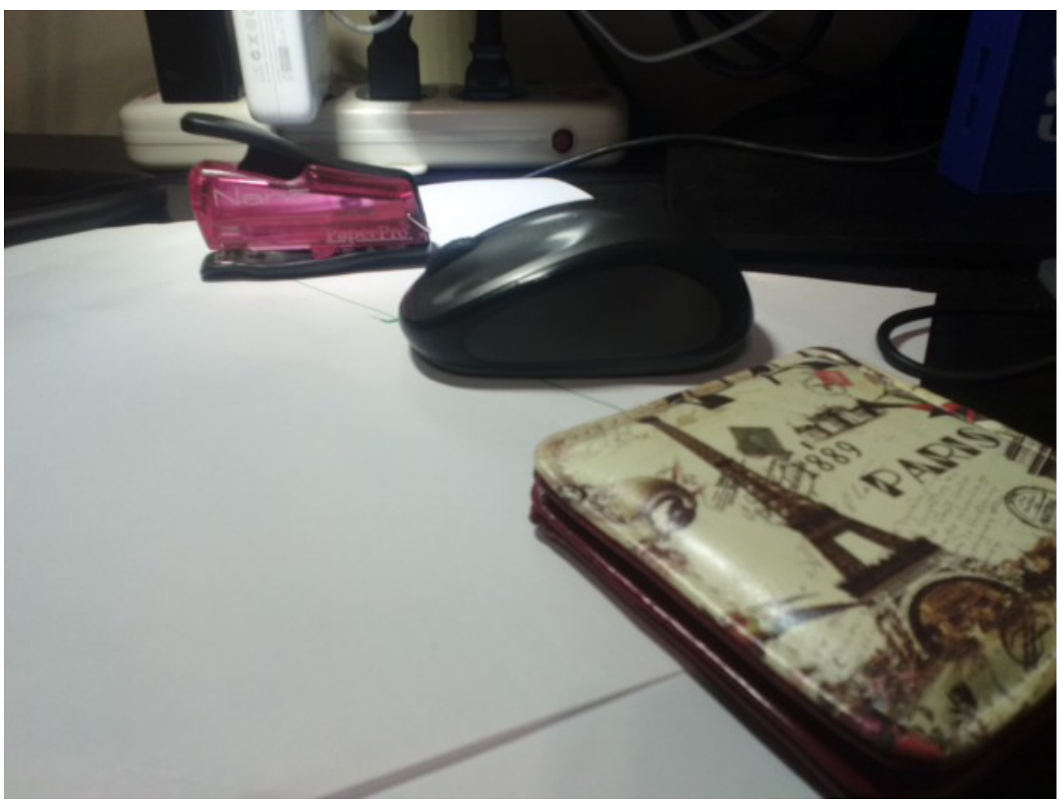


Figure 2: Captured picture with focus distance = 2.25

Figure 3: Captured picture with focus distance = 3.375

Figure 4: Captured picture with focus distance = 5.0625

Figure 5: Captured picture with focus distance = 7.59375

2. Calibrate the focal stack

We begin with reading the pictures into Matlab:

```
clear;
clc;
figureNo = 1;
NumOfIms = 5;
str_Path = '/Users/HKLHK/Desktop/Northwestern University
Q1/EECS495 Introduction to Computational
Photography/HW/HW5/Image/';

for i = 1: NumOfIms
    str_Load = strcat(str_Path, num2str(i), '.jpg');
    Image = imread(str_Load);
    I(:,:,:,i) = Image;
end
```

Notice that there are small changes in magnification between one picture to another. We can compensate the changes by using the following Matlab code:

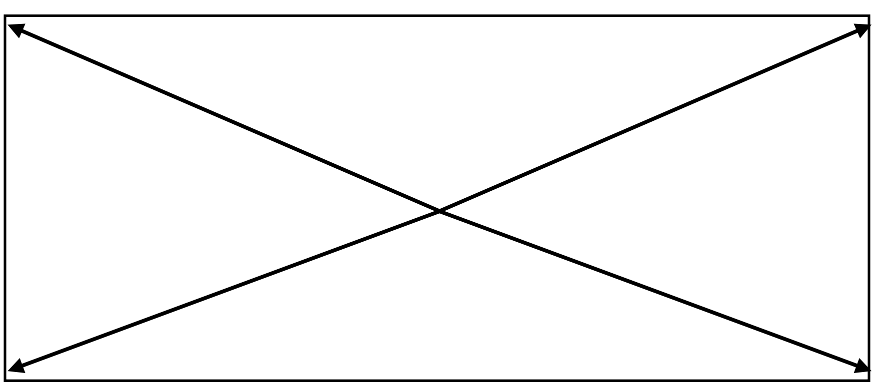
```
f = 2.95/1000;
v=[1.5 2.25 3.375 5.0625 7.59375];
%v=flip([1.5 2.25 3.375 5.0625 7.59375]);
u = 1./(1/f-1./v);

m = u(end)./u;
mr = m;
mi = 2-m;

[row column color numIm] = size(I);

for k=1:numIm
    for i=1:row
        for j = 1:column

            if i<=row/2
                mx = round(i*mi(k));
                if(mx<=0)
                    mx=1;
                end
            else
                mx = round(i*mr(k));
                if(mx>row)
                    mx=row;
                end
            end

            if j<=column/2
                my = round(j*mi(k));
                if(my<=0)
                    my=1;
                end
            else
```

```
                my = round(j*mr(k));
                if(my>column)
                    my=column;
                end
            end
            I_apostrophe(i,j,:,k) = I(mx,my,:,k);
        end
    end
end
```

Here, the rear camera has an F/2.0 aperture with 2.95mm focal length. We can calculate the lens-to-sensor distance during each exposure u_k using the Gaussian Lens Law: $\frac{1}{u_k} = \frac{1}{f} - \frac{1}{v_k}$. (1). The magnification coefficients can be calculated from: $m_k = {}^{u_N}/_{u_k}$. (3). Then, the calibrated picture can be found using

$$I'(x, y, k) = I(m_k \cdot x, m_k \cdot y, k), \qquad (2)$$

Note that the magnification of the picture should be from the image center and along all directions, which can be regarded as circular magnification. Here, for the sake of simplicity, we can choose 4 magnification directions as shown in the illustration below:



In order to prevent the pixel position overflow, we can add limitation conditions to the magnified pixel, such that every pixel is within the picture range.

3. Compute a depth map from the focal stack

First of all, we need to convert the focal stack into grayscale images:

```
for k = 1: numIm
    I_gray(:,:,k) = rgb2gray(I_apostrophe(:,:,:,k));
end
```

Then, we can use the squared laplacian as a focus measure:

$$M(x,y,k) = \sum_{i=x-K}^{x+K} \sum_{j=y-K}^{y+K} |\nabla^2 I'(i,j,k)|^2, \quad (4)$$

Method 1:

```
Laplacian = [1,4,1;4,-20,4;1,4,1]/6;

for k=1:numIm
    LaplacianImage =
imfilter(I_gray(:,:,k),Laplacian,'replicate');
    M(:,:,k) = LaplacianImage.*LaplacianImage;
end
```

Method 2:

```
kernelSize = 2;

for k=1:numIm
    M(:,:,k) = squaredLaplacian(I_gray(:,:,k),kernelSize);
```

```
end

function [ focusMeasure ] =
squaredLaplacian(Image,kernelSize)
k=kernelSize;
[M,N] = size(Image);
DH = Image;
DV = Image;
DH(1:M-k,:) = diff(Image,k,1);
DV(:,1:N-k) = diff(Image,k,2);
FM = max(DH, DV);
focusMeasure = FM.*FM;
end
```

The depth can then be calculated for each pixel by finding the index into the focal stack $D(x,y)$ where the focus is maximum:

$$D(x,y) = \underset{k}{\operatorname{argmax}}\, M(x,y,k), (5)$$

```
for i=1:row
    for j = 1:column
       Depthmap(i,j)=max(M(i,j,:));
    end
end
figure;
imshow(Depthmap);
```

Using Method 2, we can set different kernel sizes for focus measurement. When kernel size is small, say it equals to 2, the result is vulnerable to the noise, where the kernel emphasizes local characteristics. When the kernel size is lager, say it equals to 10, the result loses the texture details.

Here, for the better quality result, we use Method 1 to obtain depthmap of the focus stack, where the kernel size equals to 3.

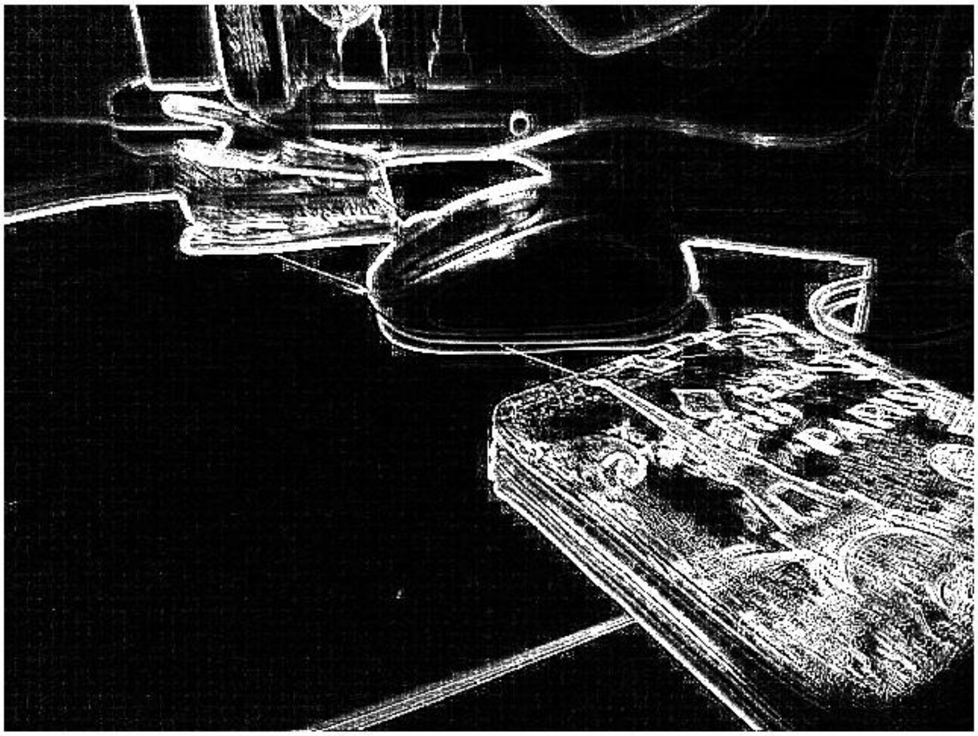


Figure 7: Calculated depthmap of the focus stack

We can create a pixel map that tells us which image in the stack a given pixel is in focus:

```
for i=1:row
    for j = 1:column
       index = find(M(i,j,:)==max(M(i,j,:)));
       if length(index)>1
           D(i,j) = index(1);
       else
           D(i,j) = index;
       end
    end

end
```

A depth index map can be built using the following Matlab code:

```
DepthIndexMap = uint8(D.*(255/max(max(D))));
figure;
imshow(DepthIndexMap);
```

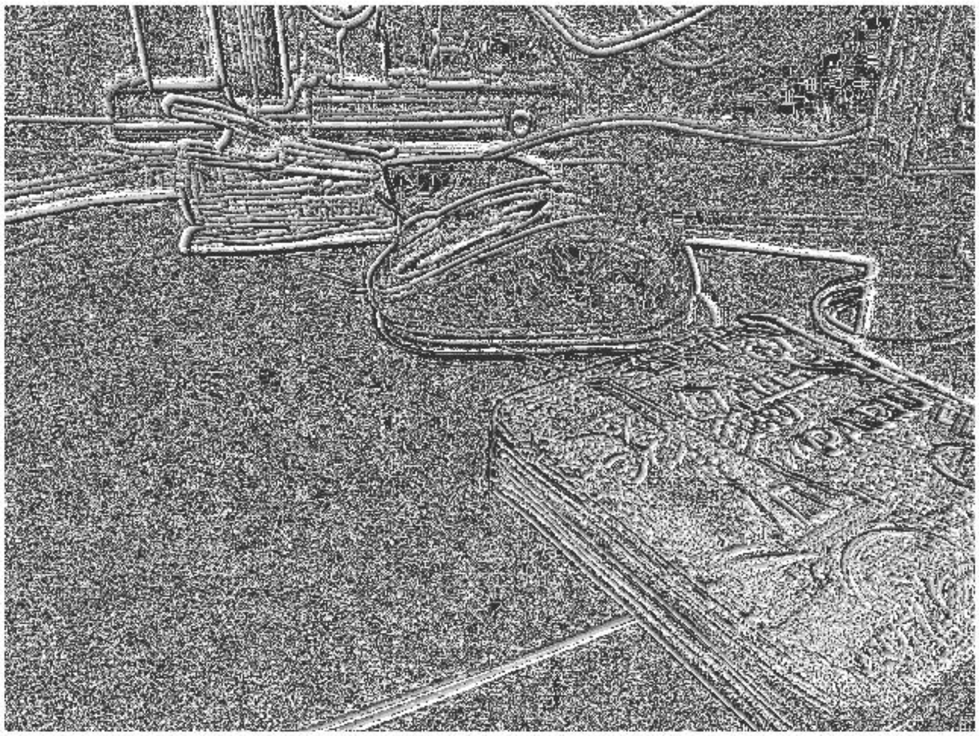


Figure 8: A depth index map computed from the focal stack

4. Recover an all-focus image of the scene

Once we have computed a depth map of the scene, we can use it to recover an all-focus image of the scene:

$$A(x, y) = I'(x, y, D(x, y)). \quad (8)$$

```
for c = 1:color
```

```
    for i=1:row
        for j = 1:column
            A(i,j,c) = I_apostrophe(i,j,c,D(i,j));
        end
    end
end
figure;
imshow(A)
```

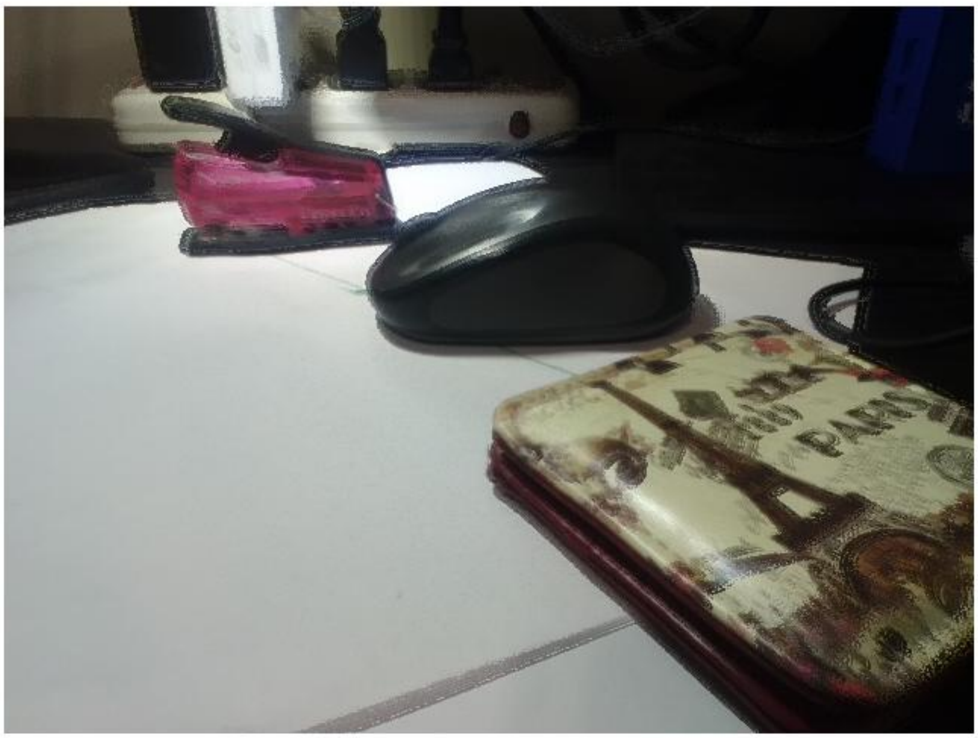


Figure 9: An all-focus image computed from the focal stack

Note that, because the calibration of the focal stack is not precise, as we can see in the Figure 9, there are lot of noise in the rendered picture. The final result is not clear enough as an all-focus image. If we can use feature matching to calibrate the focal stack, although that may introduce computational complexity, the final all-focus image can be supposed as a good one.