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 dynamic range properties of images

1. Write an Auto Exposure Bracketing (AEB) function for Tegra

Using the following code, we can capture a series of pictures with different exposure time settings. The pictures are taken with increasing the exposure time by a factor of 2 from he least exposure time (0s) to longest one (0.2752s).

```
public void captureExposureStack(View v) {

    Range<Long> exposureRange =
characteristics.get(CameraCharacteristics.SENSOR_INFO_EXPOS
URE_TIME_RANGE);
    Long minimumExposure = exposureRange.getLower();
    Long maximumExposure = exposureRange.getUpper();
    Log.e(TAG, "minimumExposure: " + minimumExposure);
```

```
    Log.e(TAG, "maximumExposure: " + maximumExposure);
    Long prevExposure = minimumExposure;
    //SystemClock.sleep(400);
    //check if 2* exposure >maximumExposure
    while (prevExposure + prevExposure < maximumExposure) {
        try {
            //sleep the system for 20ms between each capture.
            SystemClock.sleep(20);
            Log.e(TAG, "exposure: " + prevExposure);

            prevExposure = prevExposure + prevExposure;
            //create capture requester
            //CaptureRequest.Builder requester =
mCameraDevice.createCaptureRequest(mCameraDevice.TEMPLATE_M
ANUAL);
            CaptureRequest.Builder requester =
mCameraDevice.createCaptureRequest(mCameraDevice.TEMPLATE_S
TILL_CAPTURE);
            requester.set(CaptureRequest.CONTROL_AE_MODE,
CaptureRequest.CONTROL_AE_MODE_OFF );


requester.set(CaptureRequest.SENSOR_EXPOSURE_TIME,
prevExposure);
            //requester.setTag("exposureTime_" +
prevExposure);
            //add surface
            requester.addTarget(mCaptureBuffer.getSurface());
            Log.e(TAG, "exposure: " + prevExposure);
            //check capture session and make capture request
            if (mCaptureSession != null)
                exposures.add(prevExposure);
                mCaptureSession.capture(requester.build(),
null, null);
        } catch (CameraAccessException e) {
            Log.e(TAG, "Failed to build actual capture
request", e);
        }
    }

}
```

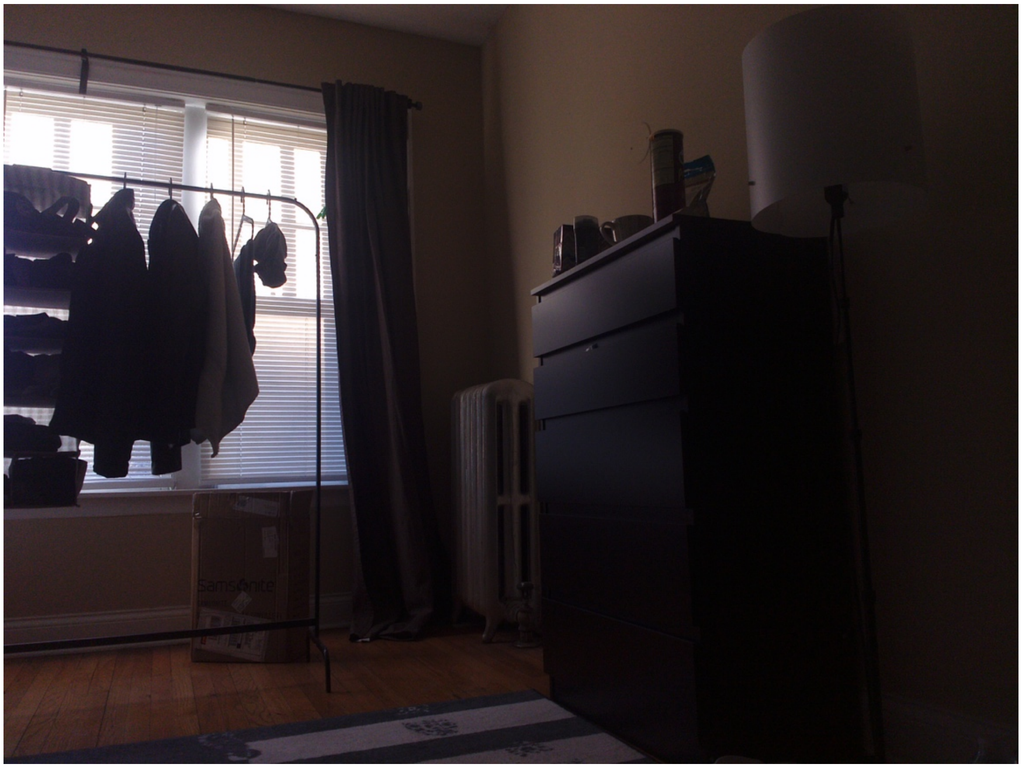

Figure 1: Captured picture with low exposure time

Figure 2: Captured picture with high exposure time

2. Write a program to find the camera response curves for the shield tablet

a) Load the pictures into MATLAB

We can use the following MATLAB code to read pictures into MATLAB. Here, in order to get the exposure time for every picture, the MATLAB build-in function imfinto() is deployed to read the exposure time from picture property.

```
str_Path = '/Users/HKLHK/Desktop/2015 Fall Quarter
(NU)/EECS495 Introduction to Computational
Photography/HW/HW4/Image/';
prevExposure = 67200;
j=1;
while prevExposure <= 275251200
   str_Load = strcat(str_Path, num2str(prevExposure),
'.jpg');
   Image = imread(str_Load);
   Info = imfinfo(str_Load);
   I(:,:,j) = double(reshape(Image, [ ], 3));
   eT(j) = Info.DigitalCamera.ExposureTime;
   prevExposure = prevExposure + prevExposure;
   j = j + 1;
end
```

b) Discard the >20% saturation pictures

We keep 5 largest exposure time pictures whose saturations are less than 20%. For the selected pictures, we rearrange their pixels of each color channel into a column vector for easy processing.

```
[numPi color numIm] = size(I);
j=1;
for i=1:numIm
    if j<=5
       if (sum(sum(all(I(:,:,(numIm-
i+1))==255,2)))/numPi)<=0.2
          Im(:,:,5-j+1) =  I(:,:,(numIm-i+1));
          exposureTime(5-j+1) = eT(numIm-i+1);
          j = j+1;
       end
    else
        clear I eT;
        break;
    end
end
```

c) Find the camera response curves

First, we can randomly choose 1000 pixels from the picture of each exposure time setting (there are 5 exposure time settings are used: 0.0172s, 0.0344s, 0.0688s, 0.1376s and 0.2752s). These 1000 pixels can be used to represent the global property of the picture.

```
numPrt = 1000;
id = randperm(numPi,numPrt);
Z_red = zeros(numPrt,length(exposureTime));
Z_green = zeros(numPrt,length(exposureTime));
Z_blue = zeros(numPrt,length(exposureTime));
for i=1:length(exposureTime)
    Z_red(:,i)= Im(id,1,i);
    Z_green(:,i) = Im(id,2,i);
    Z_blue(:,i) = Im(id,3,i);
End
```

Second, we can recover the response curve from the 5000 pixels using gsolve.m. In gsolve.m, we set l = 5 and weighting parameter = 1/256. The outputs of gsolve.m are the response curve g and the recovered log radiance lE for each of the pixels that we input to the algorithm.

```
l = 5; % [.1, 5]

[g_red,lE_red]=gsolve(Z_red,log(exposureTime),l);
[g_green,lE_green]=gsolve(Z_green,log(exposureTime),l);
[g_blue,lE_blue]=gsolve(Z_blue,log(exposureTime),l);
```

we can plot the recovered values g versus the valid range of pixel values for each of the red, green and blue channels. In the same figure, we can also plot of the log exposure for each of the pixels used as input to gsolve.m.

```
for i=1:length(exposureTime)
    X_red(:,i)= lE_red + log(exposureTime(i));
    X_green(:,i) = lE_green + log(exposureTime(i));
    X_blue(:,i) = lE_blue + log(exposureTime(i));
end

figure;
for i=1:length(exposureTime)
    scatter(X_red(:,i),Z_red(:,i),'.','b');
    hold on;
end
plot(g_red,0:255,'r','linewidth',2);
title('Response curve for red channel');
```

```
xlabel('log exposure');
ylabel('pixel value(Z)')
legend('fitted data','response curve')
hold off;

figure;
for i=1:length(exposureTime)
    scatter(X_green(:,i),Z_green(:,i),'.','b');
    hold on;
end
plot(g_green,0:255,'r','linewidth',2);
title('Response curve for green channel');
xlabel('log exposure');
ylabel('pixel value(Z)')
legend('fitted data','response curve')
hold off;

figure;
for i=1:length(exposureTime)
    scatter(X_blue(:,i),Z_blue(:,i),'.','b');
    hold on;
end
plot(g_blue,0:255,'r','linewidth',2);
title('Response curve for blue channel');
xlabel('log exposure');
ylabel('pixel value(Z)')
legend('fitted data','response curve')
hold off;

figure;
plot(g_red,0:255,'r','linewidth',2);
hold on;
plot(g_green,0:255,'g','linewidth',2);
hold on;
plot(g_blue,0:255,'b','linewidth',2);
title('Response curves');
xlabel('log exposure');
ylabel('pixel value(Z)')
legend('Red','Green','Blue')
hold off;
```

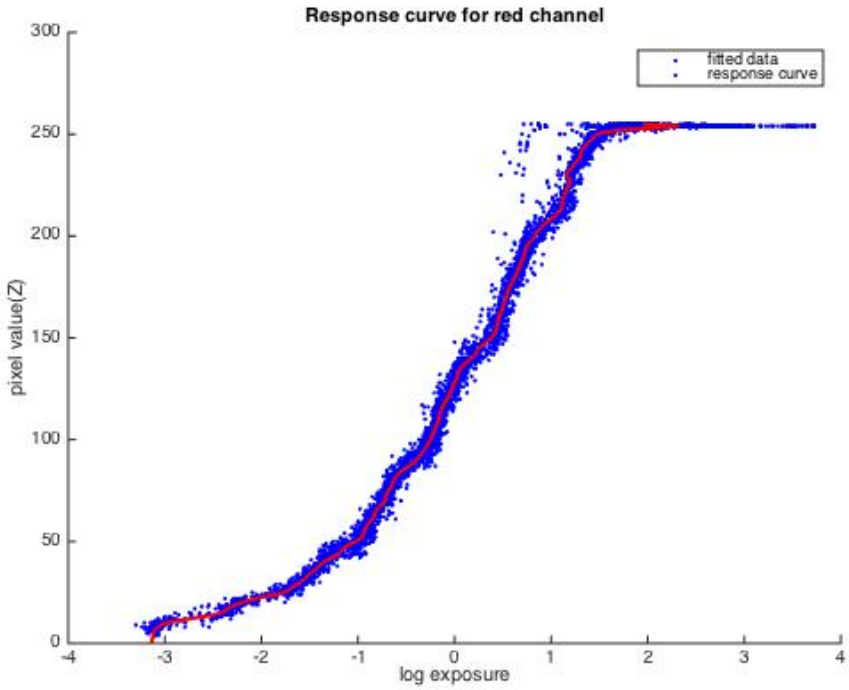

Figure 3: Response curve for red channel

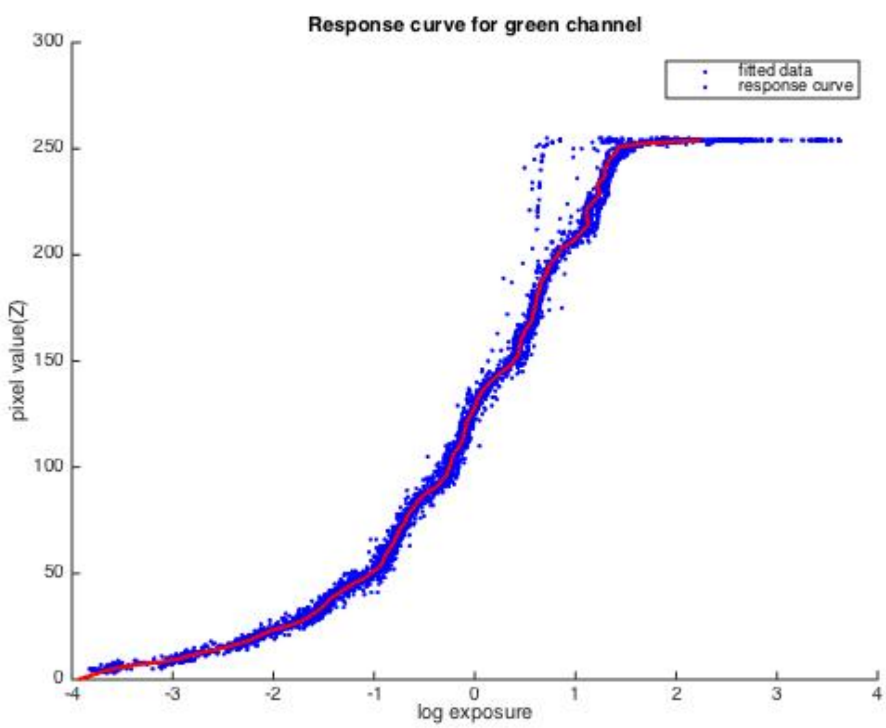

Figure 4: Response curve for green channel

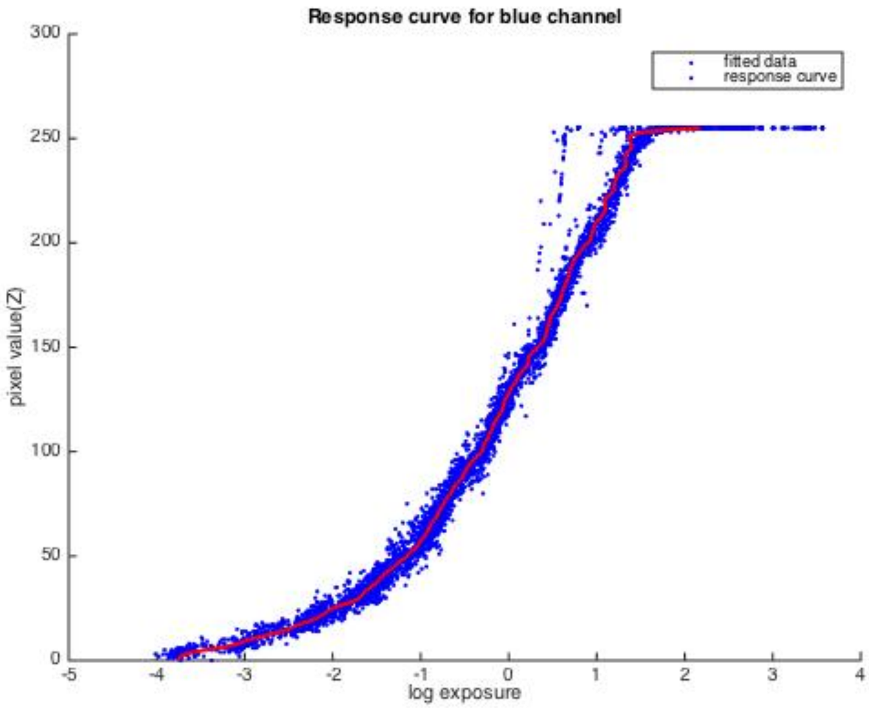

Figure 5: Response curve for blue channel

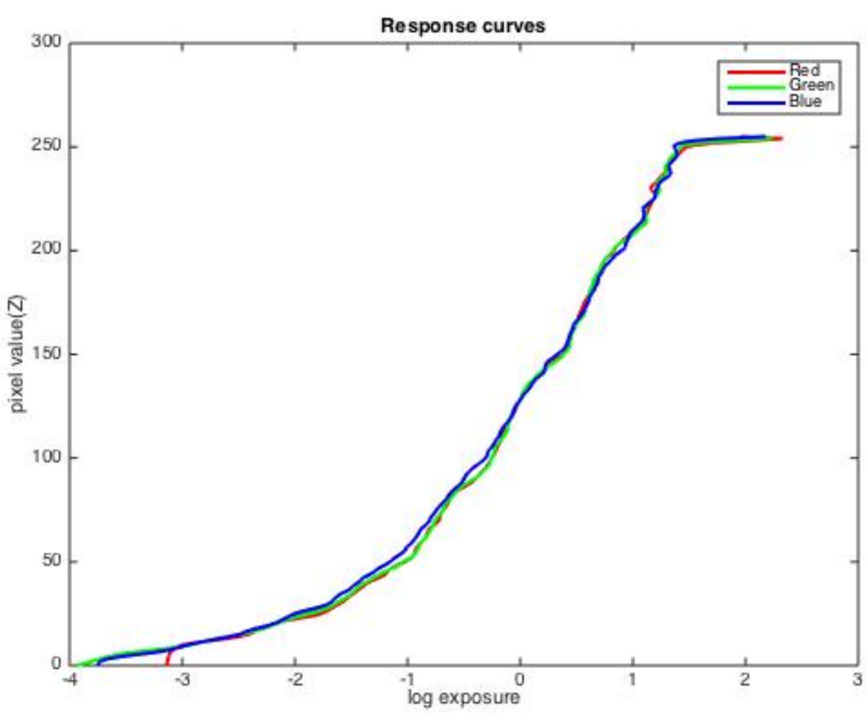

Figure 6: Response curve all 3 channels

3. Recover the HDR radiance map of the scene

We can recover the radiance map using the following equation:

$$\ln(E[i]) = \frac{1}{P}\sum_{j=1}^{P}(g(Z[i,j]) - \ln(B[j]))$$

```
[numPi color numIm] = size(Im);
lnE = zeros(numPi,3);
for i=1:numPi
    g(:,1) = g_red(Im(i,1,:)+1);
    lnE(i,1) = sum(g-log(exposureTime)')/numIm;

    g(:,1) = g_green(Im(i,2,:)+1);
    lnE(i,2) = sum(g-log(exposureTime)')/numIm;

    g(:,1) = g_blue(Im(i,3,:)+1);
    lnE(i,3) = sum(g-log(exposureTime)')/numIm;
end
```

We can plot of the radiance image recovered from the AEB sequence for each color channel. The dynamic range of the scene is nearly 10^5 or 100,000:1.

```
E = 10.^(lnE);
```

```
re_lnE = zeros(1944,2592,3);
for i=1:color
    re_lnE(:,:,i) = reshape(lnE(:,i), [], 2592);
end
[X,Y] = meshgrid(1:2592,1:1944);

figure;
```

```
h=pcolor(X,Y,flip(re_lnE(:,:,1)));
set(h,'edgecolor','none','facecolor','interp');
colorbar;
title('Recovered radiance map: Red (ln scale)');
figure;
h=pcolor(X,Y,flip(re_lnE(:,:,2)));
set(h,'edgecolor','none','facecolor','interp');
colorbar;
title('Recovered radiance map: Green (ln scale)');
figure;
h=pcolor(X,Y,flip(re_lnE(:,:,3)));
set(h,'edgecolor','none','facecolor','interp');
colorbar;
title('Recovered radiance map: Blue (ln scale)')
```

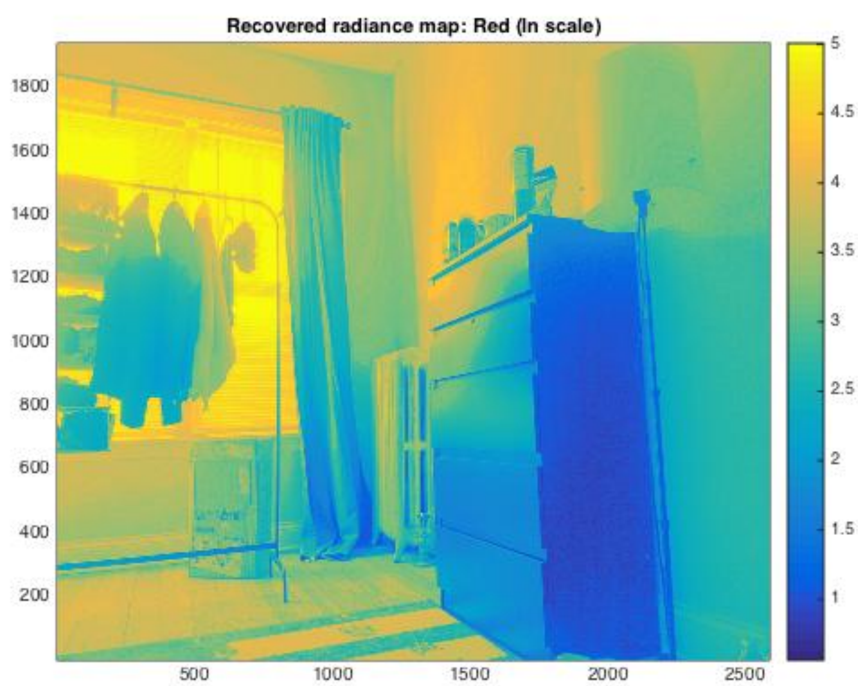

Figure 7: Recovered radiance map for red channel

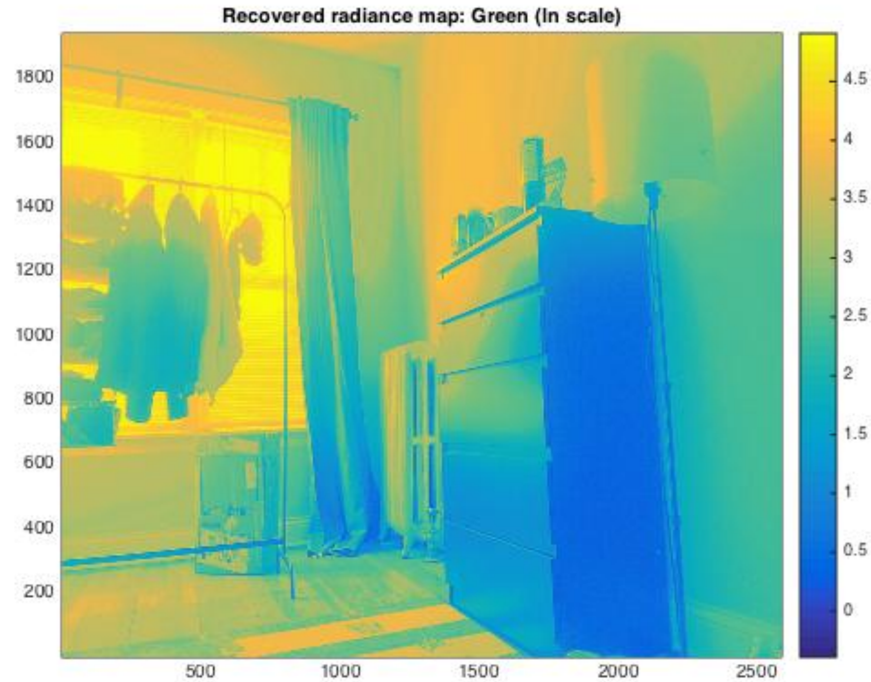

Figure 8: Recovered radiance map for green channel

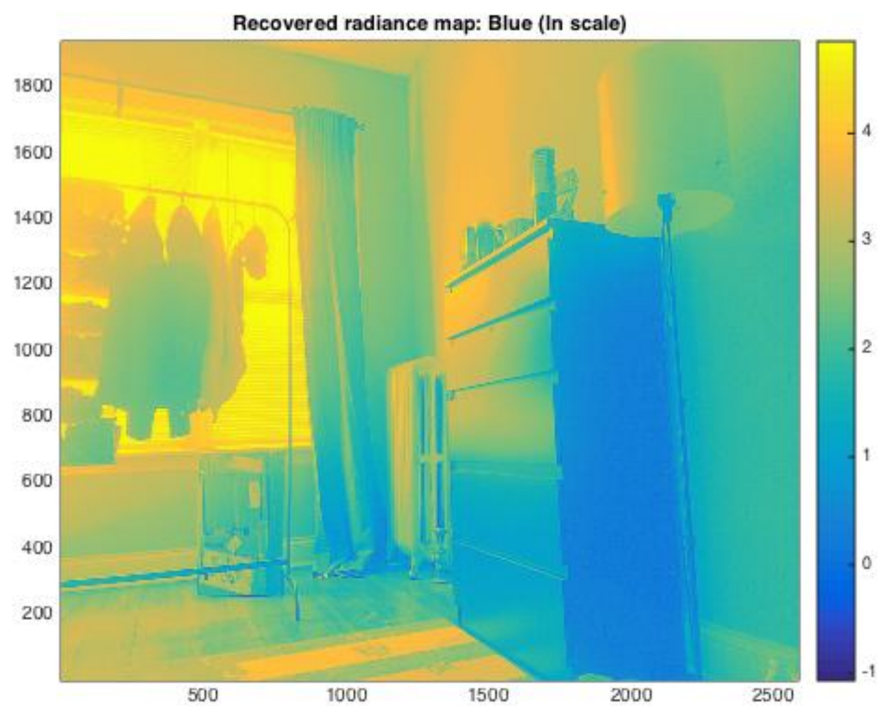

Figure 9: Recovered radiance map for blue channel

4. Implement a tone mapping algorithm to display HDR image

The radiance map recovered has a much larger dynamic range than 0:255. Here, we can apply a simple global tone-mapping algorithm to radiance image.

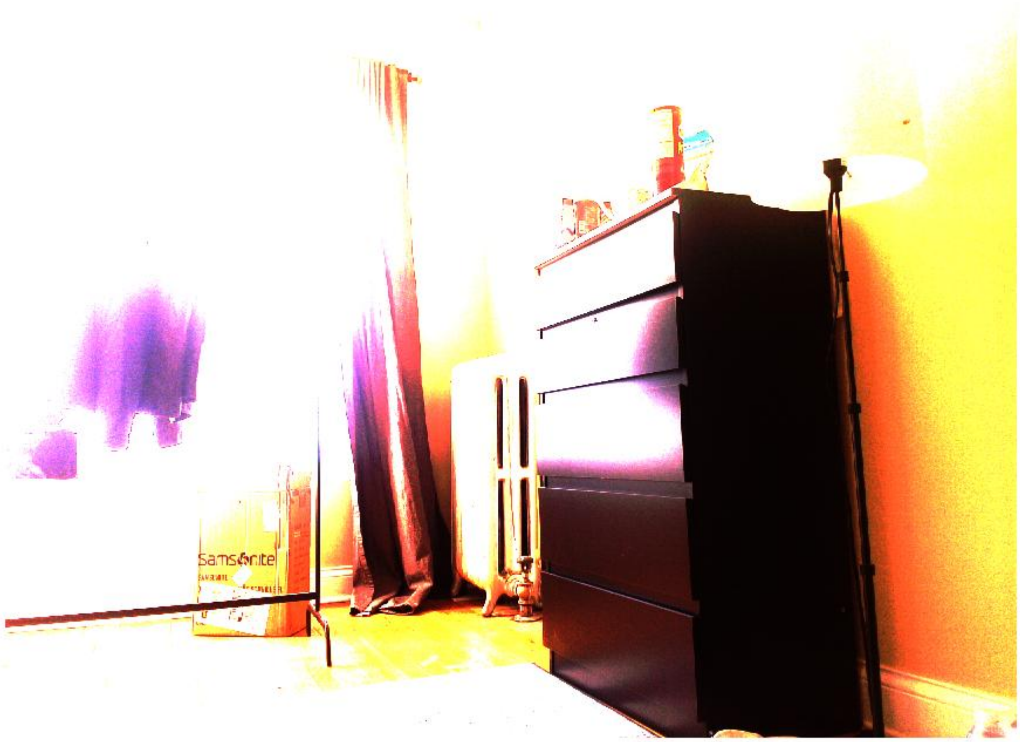

Figure 10: Recovered radiance map

a) We can scale the brightness of each pixel uniformly so that all of the pixels fall in the range [0,1] using the algorithm below:

$$E_{norm}[i] = \frac{E[i]-E_{min}}{E_{max}-E_{min}}$$

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

```
E_norm(:,c) = (E(:,c) - min(E(:,c)))./(max(E(:,c)) -
min(E(:,c)));
end

re_E_norm = zeros(1944,2592,3);
for i=1:color
   re_E_norm(:,:,i) = reshape(256*E_norm(:,i)-1, [], 2592);
end

figure;
subplot(1,2,1);
imshow(uint8(re_E_norm));
subplot(1,2,2);
hist(E_norm);
legend('r','g','b');
```

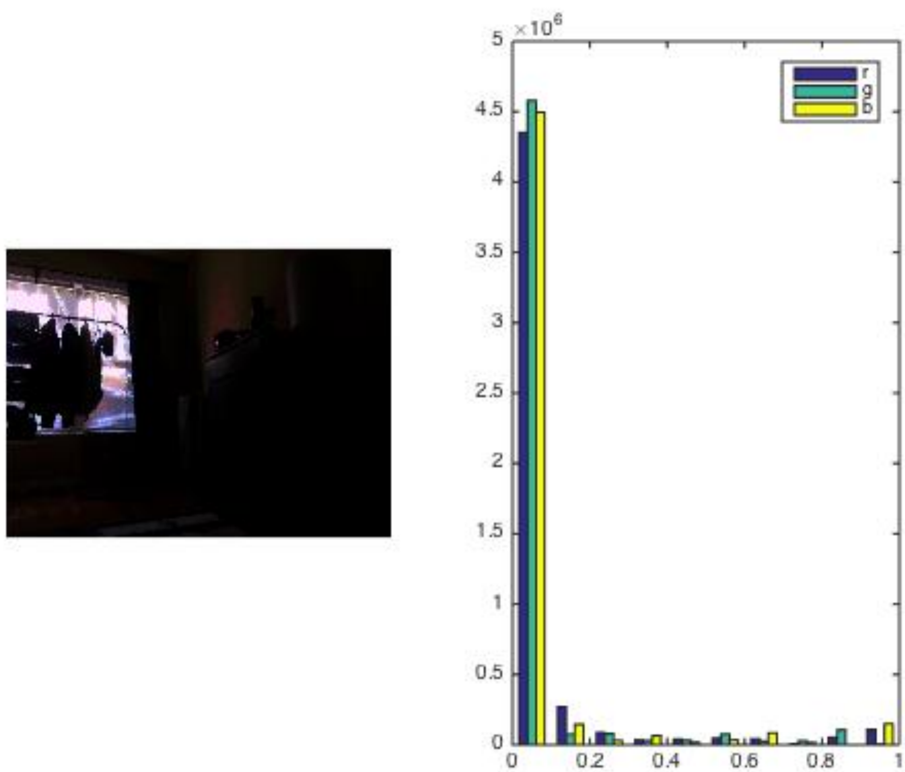

Figure 11: Normalized radiance map with its histogram

Notice that the image on the left panel of Figure 11 looks very dark because most of the display dynamic range will be used up by the pixels with higher radiance values as shown in the histogram.

b) Next, we apply a gamma curve to the image, where gamma = 0.2.

$$E_{gamma}[i] = E_{norm}^{\gamma}[i]$$

```
gamma = 0.2;
E_ga = E_norm.^gamma;

for c=1:color
   E_gamma(:,c) = (E_ga(:,c) -
min(E_ga(:,c)))./(max(E_ga(:,c)) - min(E_ga(:,c)));
end

re_E_gamma = zeros(1944,2592,3);
for i=1:color
   re_E_gamma(:,:,i) = reshape(256*E_gamma(:,i)-1, [],
2592);
end

figure;
subplot(1,2,1);
imshow(uint8(re_E_gamma));
subplot(1,2,2);
hist(E_gamma);
legend('r','g','b');
```

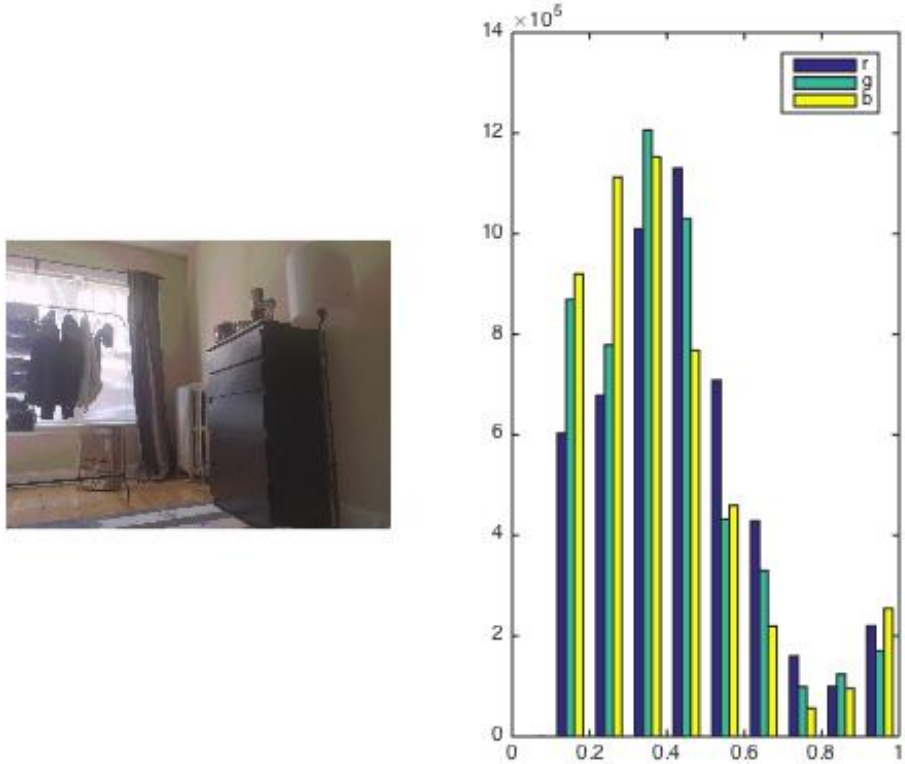

Figure 12: Gamma-corrected radiance map with its histogram

c) Apply global tone mapping operator

i. Convert the radiance image from color to grayscale:

$$L[i] = rgb2gray(E_{norm}[i])$$

ii. Calculate the log average exposure:

$$L_{avg} = \exp\left(\frac{1}{N}\sum_i \ln\left(L[i]\right)\right)$$

iii. Scale the image according to:

$$T[i] = {}^{a}/_{L_{avg}}\, L[i]$$, where a = 0.7

iv. Apply the Reinhard tone-mapping operator:

$$L_{tone}[i] = \frac{T[i]\left(1+{}^{T[i]}/_{T_{max}^2}\right)}{1+T[i]}$$

v. Define the scaling operator

$$M[i] = {}^{L_{tone}[i]}/_{L[i]}$$

vi. calculate the new RGB image according to:

$$R_{new}[i] = M[i] \cdot R[i],$$

$$G_{new}[i] = M[i] \cdot G[i],$$

$$B_{new}[i] = M[i] \cdot B[i]$$

```
a= 0.7;

T = a/L_avg *L;


L_tone = T.*(1+T./((max(max(T)))^2))./(1+T);

M = L_tone./L;

RGB_new = (265*E_gamma-1).*M;

re_RGB_new = zeros(1944,2592,3);
```

```
for i=1:color
    re_RGB_new(:,:,i) = reshape(RGB_new(:,i), [], 2592);
end

re_RGB_new = uint8(re_RGB_new);

figure;
imshow(re_RGB_new);
```

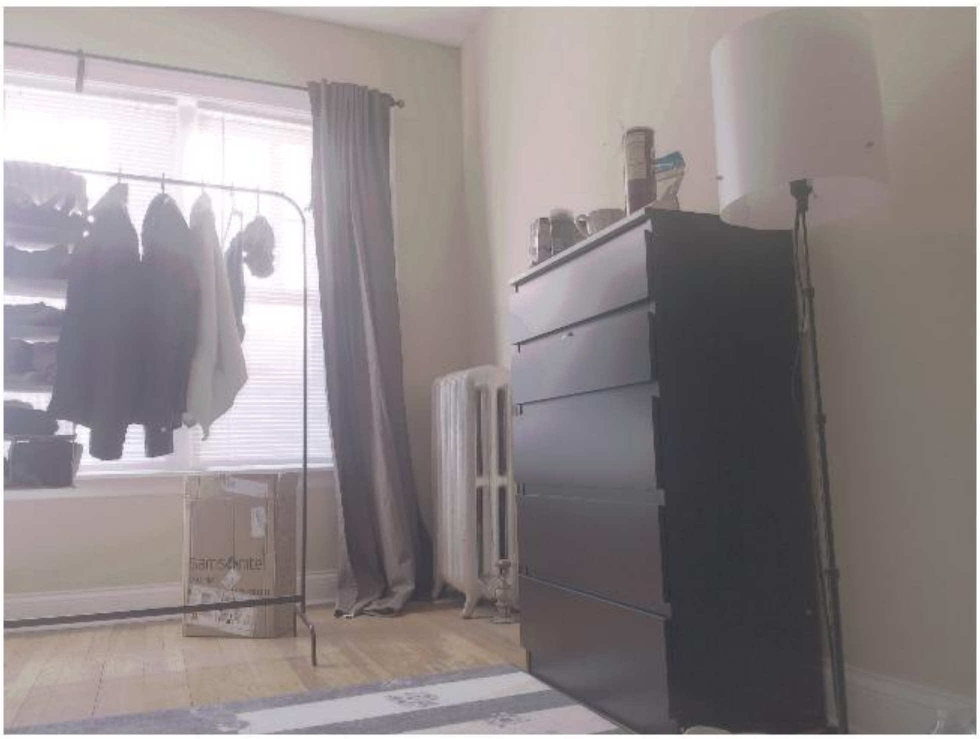

Figure 13: HDR image (a=0.18)

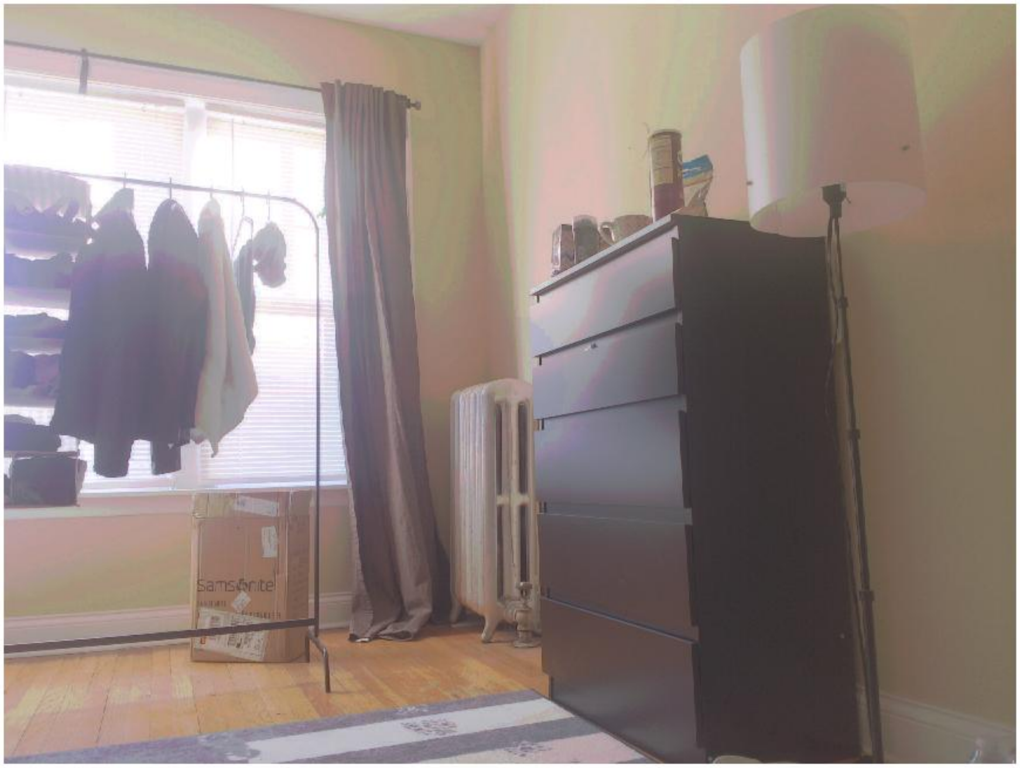

Figure 14: HDR image (a=0.7)

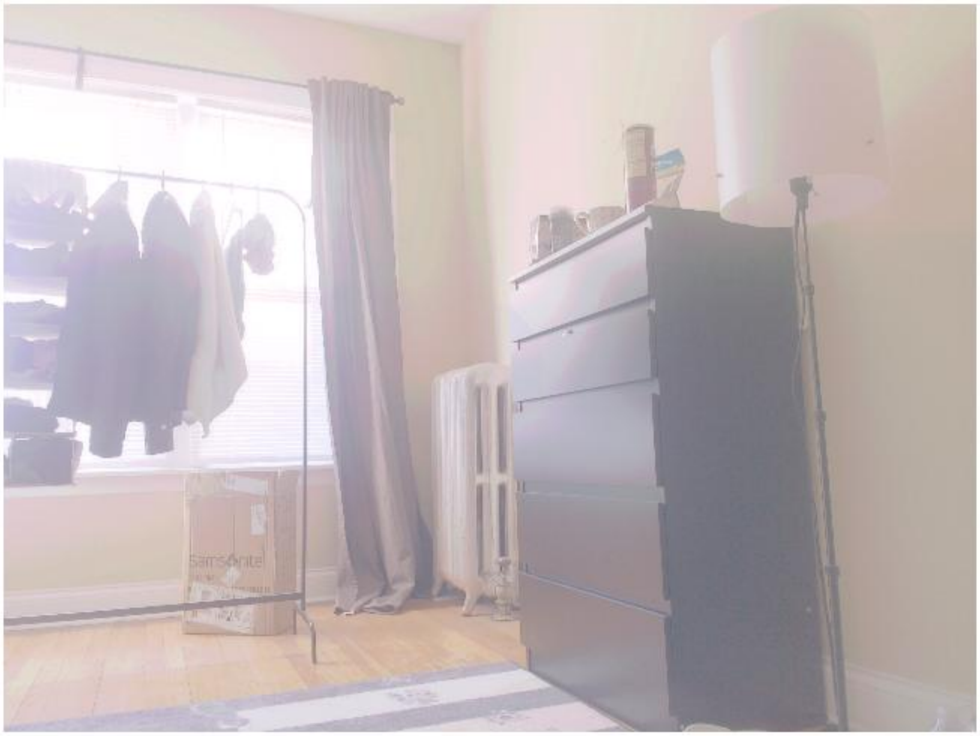

Figure 15: HDR image (a=1.0)