# HDR Image Generation Assignment

In this Assignment I used python to combine a stack of images captured with different exposures into a single high-dynamic-range image.

**Part1:**

**1. Processing the Raw Images:**

I used the dcraw program to extract auto-white balancing data from the 13th exposure, as it was well-exposed. This provided four multipliers for adjusting the color channels. Then, I converted all raw NEF files to 16-bit TIFF images using these multipliers, with a multiplier of 1 for both green channels, ensuring consistent white balance across all images.

Dcraw command for 13the exposure:

dcraw -i -v exposure13.nef
result:
Filename: **exposure13.nef**
Timestamp: Wed Oct 11 19:33:04 2017
Camera: Nikon D3300
ISO speed: 100
Shutter: 2.0 sec
Aperture: f/4.8
Focal length: 40.0 mm
Embedded ICC profile: no
Number of raw images: 1
Thumb size: 6000 x 4000
Full size: 6016 x 4016
Image size: 6016 x 4016
Output size: 6016 x 4016
Raw colors: 3
Filter pattern: RG/GB
Daylight multipliers: 2.231936 0.932648 1.141543
Camera multipliers: 1.585938 1.000000 2.054688 0.000000

**Convert it to 16 bit linear.tiff file**

# Convert all raw NEF files to linear 16-bit PPM images with specified white balance multipliers
dcraw -4 -T -r 1.585938 1.000000 2.054688 1.000000 *.nef

After this step the final processing has finished and now, we have our processes dataset.

**Part2:**

**2.Merge the LDR images into an HDR image:**

To generate a single HDR image from a set of LDR images, I employed a weighted combination approach. First, I normalized each image by the reciprocal of its exposure time to ensure consistent brightness across all. Then, I utilized a weighting function to average only properly exposed pixels.

For each pixel (i, j) in the HDR image, the formula is:

$$I_{i,j,HDR} = \frac{\sum_k w(I_{k,i,j,LDR}) \frac{I_{k,i,j,LDR}}{t_k}}{\sum_k w(I_{k,i,j,LDR})}$$

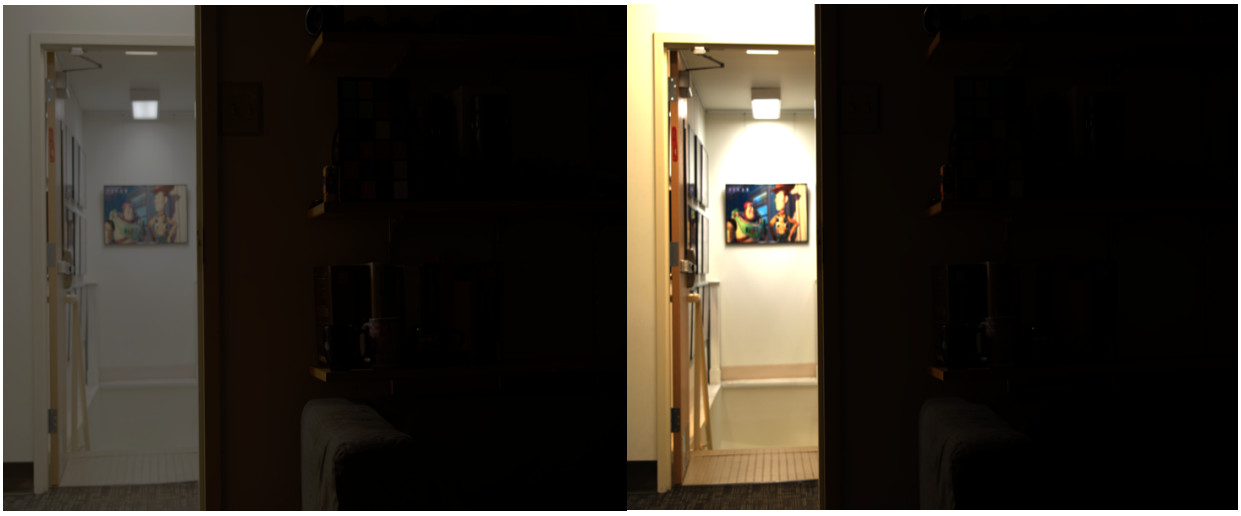
Where Tk is the exposure time of each image

Various weighting schemes can be applied, including:

1. Uniform weighting:
$$w_{uniform}(z) = \begin{cases} 1, & \text{if } Z_{min} \leq z \leq Z_{max}, \\ 0, & \text{otherwise.} \end{cases}$$
2. Tent weighting:
$$w_{tent}(z) = \begin{cases} \min(z, 1-z), & \text{if } Z_{min} \leq z \leq Z_{max}, \\ 0, & \text{otherwise.} \end{cases}$$
3. Gaussian weighting:
$$w_{Gaussian}(z) = \begin{cases} \exp\left(-4\left(\frac{(z-0.5)^2}{0.5^2}\right)\right), & \text{if } Z_{min} \leq z \leq Z_{max}, \\ 0, & \text{otherwise.} \end{cases}$$
4. Photon weighting:
$$w_{photon}(z, t_k) = \begin{cases} t_k, & \text{if } Z_{min} \leq z \leq Z_{max}, \\ 0, & \text{otherwise.} \end{cases}$$

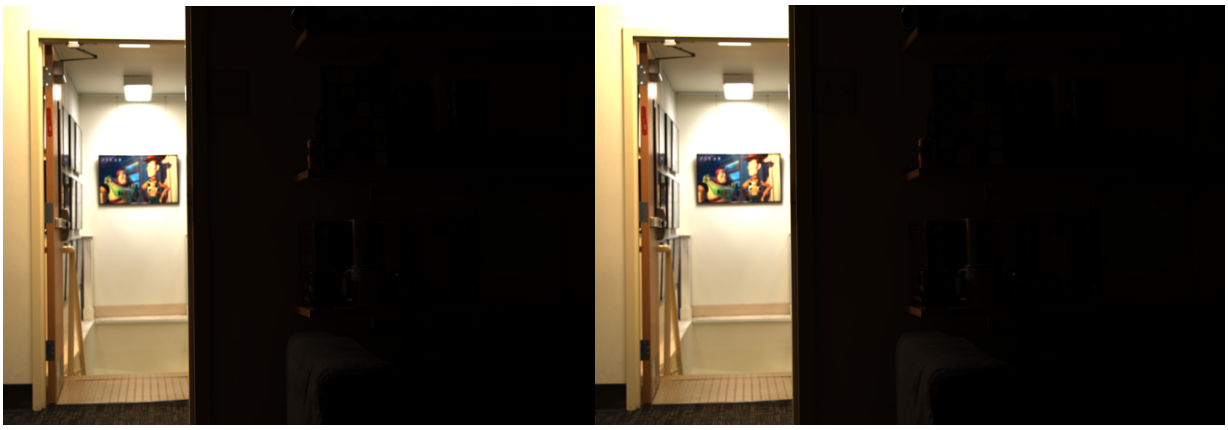
Here, $Z_{min}$ and $Z_{max}$ are tunable parameters controlling the clipping thresholds. For this task, I adopted $Z_{min} = 0.05$ and $Z_{max} = 0.95$.

Here is the result for the generated HDR images by using the (from left to right)Uniform, Tent, Gaussian and Photon weight functions



a) HDR image using Uniform weight function

b) HDR image using Tent weight function.



c) HDR image using Gaussian weight function

d) HDR image using Photon weight function.

**Part3:**

**3. Tone-map the HDR image**

Now we have four HDR images that needs to be tone-mapped for display. We tone-mapped the generated hdr images using the operator proposed by Reinhard et al.

$$I_{i,j,TM} = \frac{\tilde{I}_{i,j,HDR}(1 + \frac{\tilde{I}_{i,j,HDR}}{\tilde{I}^2_{white}})}{1 + \tilde{I}_{i,j,HDR}}, \tag{5}$$

where

$$I_{m,HDR} = \exp\left(\frac{1}{N}\sum_{i,j}\log(I_{i,j,HDR} + \epsilon)\right), \tag{6}$$

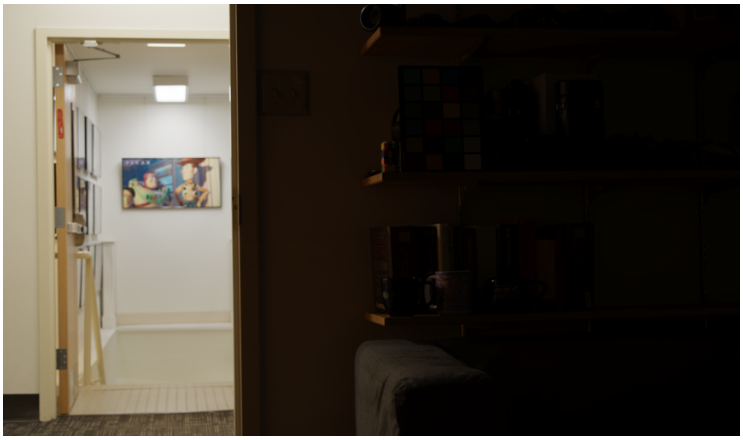
$$\tilde{I}_{i,j,HDR} = \frac{K}{I_{m,HDR}} I_{i,j,HDR}, \tag{7}$$

$$\tilde{I}_{white} = B \max_{i,j}(\tilde{I}_{i,j,HDR}). \tag{8}$$

The parameter K is the key, which determines how bright or dark the resulting tone mapped image is. The parameter B is the burn and can be used to suppress contrast in the result. Finally, N is the number of pixels in the image and ϵ is a small constant to avoid log(0).

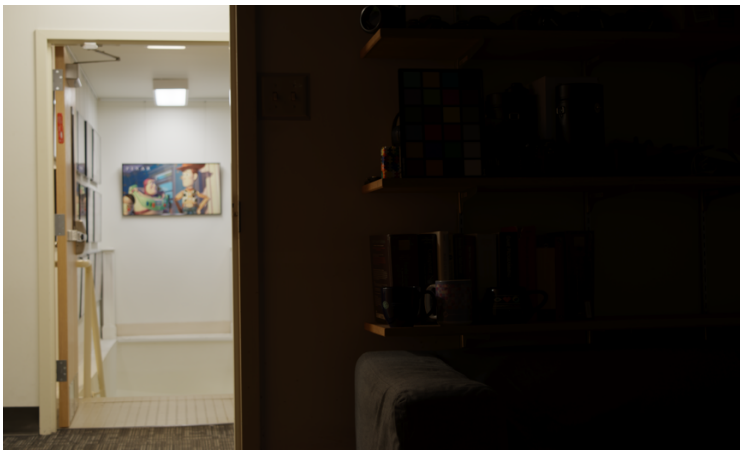
Here is the tone mapped HDR images that we computed with Gaussian: These are the various values of K and B that we used to generate the various tone-mapped images.

1. When k=0.01 and B=0.95



2. When K=0.05 and B=0.85 value

3. When K=0.09 and B=0.1

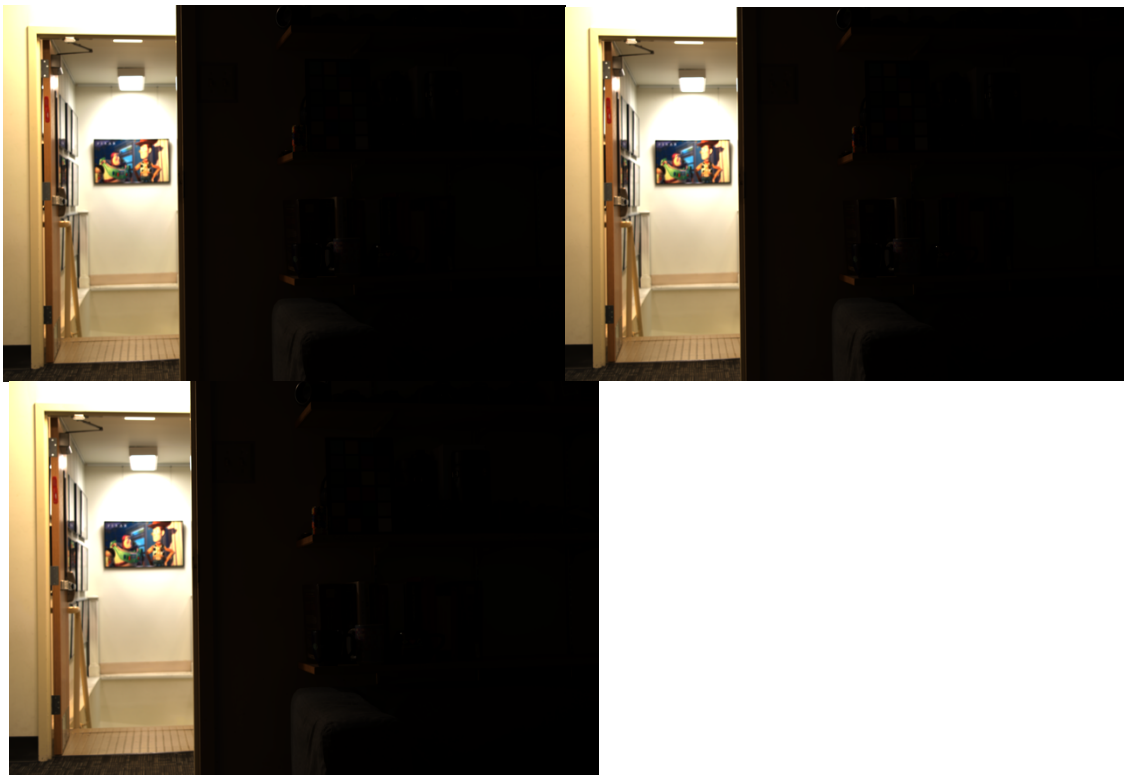


In Reinhard's method, the key value (K) controls the overall brightness of the image. The final key is usually a product of K and this average luminance. So, increasing K will generally make the image brighter, but the effect will be modulated by the average luminance of the image.

The bias (B) is used in a function that maps the high dynamic range values in the image to a displayable range. The bias parameter controls how these values are compressed. A higher bias value will preserve more details in the shadows (darker areas)(noticeable in image 2), while a lower bias value will preserve more details in the highlights

(brighter areas)(noticeable in image 3). So, as B decreases from 0.95 to 0.1, you should see more emphasis on the brighter areas of the image, and less on the darker areas.
Finally, we choose k=0.01 and B=0.95 as the parameters for the rest of images

Here is the tone mapped HDR images for Uniform, Tent and Photon:



**Part4:**

4.Gamma-Correct the LDR image for display:

The tone-mapped images from the previous step still need to be gamma-corrected to present accurately on display, otherwise they will appear quite dark. We apply the Gamma correction and following images has displayed and saved with the following operator, which corresponds to the tone reproduction curve specified in the sRGB stand:

$$C = \begin{cases} 12.92C_{linear}, & C_{linear} \leq 0.0031308 \\ (1+0.055)C_{linear}^{\frac{1}{2.5}} - 0.055, & C_{linear} > 0.0031308. \end{cases}$$

Here Gamma-Correct images:

a) Gamma-correction(Gaussian HDR image)

b) Gamma-correction( Uniform) HDR image

c) Gamma-correction (Tent HDR image)

c) Gamma-correction( Photon HDR image)

**Part5:**
5.Lossy Compression

The result of sweeping over the JPEG quality settings, we have generate couple of JPEG images in order to compare the compression quality :

Code :

```
import cv2
import numpy as np
import os
```

```
def mse(imageA, imageB):
    # Compute Mean Squared Error between two images
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])
    return err


# Load the original image
original =
cv2.imread('/Users/rk/Downloads/HDRProject/nef_images/tiff_images/results/15_images_G/G_K0.01_B0.95/Gau
ssian_hdr_tm_gamma_K_0.01_B_0.95.jpg')

# Initialize variables
min_quality = 100
min_quality_mse = float('inf')

# Sweep over JPEG quality settings
for quality in range(100, 0, -1):
    # Save the image with current quality setting
    filename =
f'/Users/rk/Downloads/HDRProject/nef_images/tiff_images/results/15_images_G/G_K0.01_B0.95/Gaussian_hdr_t
m_gamma_{quality}.jpg'
    cv2.imwrite(filename, original, [int(cv2.IMWRITE_JPEG_QUALITY), quality])

    # Load the compressed image
    compressed = cv2.imread(filename)

    # Calculate MSE between original and compressed image
    current_mse = mse(original, compressed)

    # If the MSE is low enough
    if current_mse < 1:
        # Update minimum quality and its MSE
        min_quality = quality
        min_quality_mse = current_mse
    else:
        # If MSE is above threshold, stop the loop
        break

# Calculate compression ratio
size_png =
os.path.getsize('/Users/rk/Downloads/HDRProject/nef_images/tiff_images/results/15_images_G/G_K0.01_B0.95/
Gaussian_hdr_tm_gamma_K_0.01_B_0.95.jpg')
size_jpg =
os.path.getsize(f'/Users/rk/Downloads/HDRProject/nef_images/tiff_images/results/15_images_G/G_K0.01_B0.95/
Gaussian_hdr_tm_gamma_{min_quality}.jpg')
compression_ratio = size_png / size_jpg

print(f'Minimum quality: {min_quality} (MSE: {min_quality_mse})')
print(f'Compression ratio: {compression_ratio}')
```

**Output:**
Minimum quality: 90 (MSE: 0.5837587151394422)
Compression ratio: 0.9990714537786598

**Discussion:**
The lowest acceptable quality setting was determined by sweeping over the JPEG quality settings from 100 to 1 and saving the image at each quality level. The Mean Squared Error (MSE) between the original and the compressed image was calculated at each step. The quality setting at which the MSE first fell below a threshold of 1 was considered the lowest acceptable quality setting. In our case, that setting was found to be 90.

The compression ratio was calculated as the ratio of the file sizes of the original PNG image and the compressed JPEG image. Specifically, the file size of the original PNG image was divided by the file size of the JPEG image saved at the lowest acceptable quality setting. In this case, the compression ratio was approximately 0.999.

The results demonstrate the trade-off between image quality and file size in image compression. A lower JPEG quality setting results in a smaller file size, but also a lower image quality. In this case, a quality setting of 90 was found to provide a compressed image that was nearly indistinguishable from the original, with a very small reduction in file size (compression ratio close to 1). This suggests that the JPEG compression is very efficient for this particular image, providing high image quality with minimal increase in file size.

a) JPG compression of Gaussian HDR (quality 90)

b) Original Gaussian HDR image