

VFX HW1

“ b05902019 資工三 蔡青邑
b05902040 資工三 宋易軒

Introduction

This is our report for homework1 (NTUCSIE VFX 2019). And the code here shows how we implemented HDR image, along with image adjustment and tone mapping.

At the end, we ran our code on two sets of images (View1 and View2 in testing_images). And finally choose the result of View1 as our final submission.

In this report we'll show we weimplement images.

How to reproduce our code

simply

```
python3 HDR.py
```

You're whelcomed to change the input path of readImages() in our main function to change the images (depends on which images set you'd like to implement on).

MTB alignment

We've implemented the MTB(Median Threshold Bitmap) alignment in function MLT_alignment(images, template) .

- Firstly, compute the grey value of each images by $54 * r + 183 * g + 19 * b$, and binarize the images by the median value(the threshold bitmap).
- Secondly, compute the exclusive bitmap by change the pixel value which is too close (< 4) to threshold to 0.
- Thirddly, compute the errors compared with the template image, and then record the least-error-shift.
- Lastly, by recursively divide the image in half, and repeat the above step,we obtain the fianl aligned images with the shitfed direction.

R	G	B

Grey	Bitmap

Sampling image

The original images consists of 9 images with different shutter times. To access them in a single matrix we have to sample the intensity value on the same position for each value in our intensity range(0~255). That is, we create an array of the size of 256 * 9, and by checking where each value appears on our median image, accordingly search for the value of same position on other images.

Response Curve

- Intensity weighting

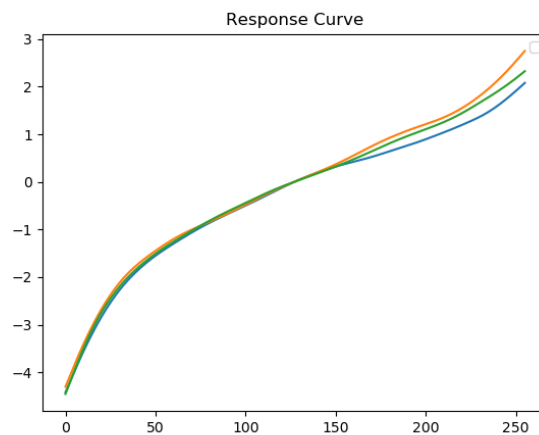
We use a weight function to calculate a weight for each pixel, which can reduce the bias from extreme intensity when calculate the response fuction.

```
def intensity_weighting(pix_value, MAX_intensity):  
    return(min(pix_value, MAX_intensity - pix_value))
```

- Calculation

Minimize the following optimization equation to compute response function.

Sample different pixels with same intensity from resource images, construct the corresponding matrix mentioned in the lecture slide and calculate the inverse matrix. Then we could recover response function.
The figure below shows $\ln E$ with respect to intensity ranged from 0 to 255.



Radiance Map

After we construct the response curve, we combine pixels to reduce noise and obtain a more reliable estimation along with the weight function according to our lecture.

(While implementing, we have to deal with some exception such that the sum of weight may be 0)

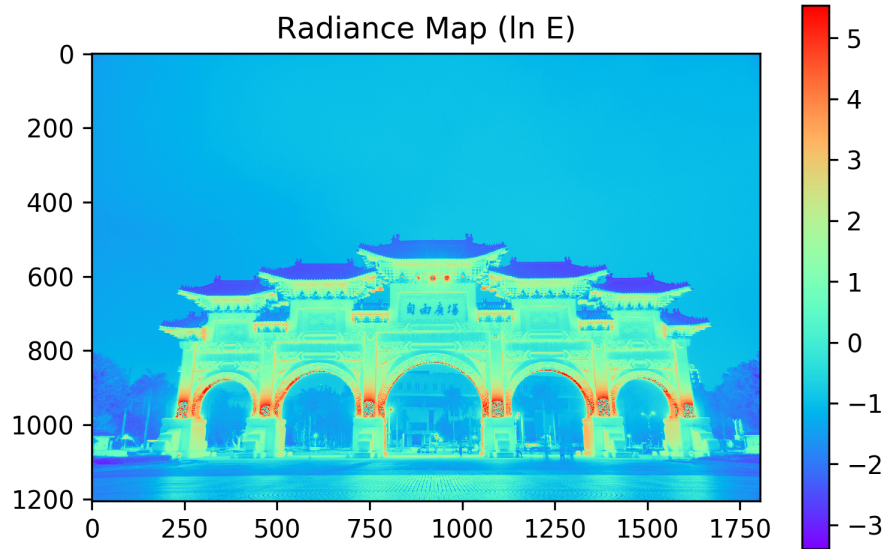
$$\ln E_i = \frac{\sum_{j=1}^P w(Z_{ij})(g(Z_{ij}) - \ln \Delta t_j)}{\sum_{j=1}^P w(Z_{ij})}$$

```

if SumW > 0:
    rad_map[i][j] = np.sum(w * (g - shutter_times) / SumW)
else:
    idx = rd.randrange(len(images))
    rad_map[i][j] = g[idx] - shutter_times[idx]

```

Then we draw the radiance map.



Bilateral Tone Mapping

We implement the bilateral tone mapping, using a 5*5 Bilateral Filter with Gaussian function for range and spatial kernel, which can separate detail layer and base layer of the image, and reduce the contrast of base layer. Then, combine them and recover the image. By the method, we could preserve the detail in overexposed and dark area.

Our implementation is in appendix

- Bilateral Filter Definition

- Normalize Term







$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

- Procedure

1. Compute intensity by $0.2126 * R + 0.7152 * G + 0.0722 * B$
2. Divide each color channel by intensity to obtain color layer
3. Apply Bilateral Filter on log intensity to get base layer
4. Subtract intensity by base layer to get detail layer
5. Compress base layer with specific factor
6. Add detail layer to adjusted base layer
7. Recover the log intensity by exponential
8. Multiply each color channel by adjusted intensity

- Experiment

Use different compression factor to adjust the magnitude of detail and contrast of base.

0.5	0.6	0.7
		
0.8	0.9	1.0
		

After experiment, we decide to use 0.85 as our factor.



Gamma tone mapping

Also, we implement the gamma tone mapping, which is rather easy by

```
def gammaToneMapping(image):  
    gamma = 0.8  
    image_corrected = cv2.pow(image/255., 1.0/gamma)  
    return image_corrected
```

Result

The final result is at `./results/Result1.png`.

Before that, we compare different result by applying different tone mapping below, and choose the current best one.

- Gamma + Bilateral
- Gamma
- Bilateral
- Nothing(only normalize)

Gamma + Bilateral	Gamma
Bilateral	Nothing(only normalize)

At the end, we choose Bilateral + Gamma as our final result due to our own aesthetic.

Appendix

- Bilateral Filter

```
def gaussian(x, sigma):  
    return (1.0 / (2 * np.pi * (sigma ** 2))) * np.exp(- (x ** 2) / (2 * sigma ** 2))
```

```

def bilateral_filter(input_image, output_image, x, y, filter_size):
    w_sum = 0
    denoised_intensity = 0
    sigma_r = 100
    sigma_d = 200
    h = input_image.shape[0]
    w = input_image.shape[1]
    for i in range(-filter_size, filter_size + 1):
        for j in range(-filter_size, filter_size + 1):
            if x + i >= h or x + i < 0 or y + j >= w or y + j < 0:
                continue
            fr = gaussian(input_image[x + i][y + j] - input_image[x][y], sigma_r)
            gs = gaussian(np.linalg.norm([i, j]), sigma_d)
            w = fr * gs
            denoised_intensity += w * input_image[x + i][y + j]
            w_sum += w
    denoised_intensity /= w_sum
    return denoised_intensity

def bilateral(input_image):
    filter_size = 2
    output_image = np.zeros(input_image.shape)
    h = input_image.shape[0]
    w = input_image.shape[1]
    for i in range(h):
        for j in range(w):
            output_image[i, j] = bilateral_filter(input_image, output_image, i, j, filter_size)
    return output_image

def bilateral_func(input_image):
    fig, axes = plt.subplots(2, 2, figsize=(15, 15))
    R = input_image[0]
    G = input_image[2]
    B = input_image[1]
    intensity = 0.2126 * R + 0.7152 * G + 0.0722 * B
    r = R / intensity
    g = G / intensity
    b = B / intensity
    log_intensity = np.log(intensity)
    log_base = bilateral(log_intensity)
    log_detail = log_intensity - log_base
    log_abs_scale = np.max(log_base) * compressionfactor
    log_output = log_base * compressionfactor + log_detail - log_abs_scale
    R_out = r * np.exp(log_output)
    G_out = g * np.exp(log_output)
    B_out = b * np.exp(log_output)
    plt.show()
    return [R_out, B_out, G_out]

```