

# Digital Visual Effects – HDR Imaging

B11902078 張宸瑋, B10902058 胡桓碩

## Summary

Using Python, we implemented **Paul Debevec's Method** to assemble our HDR images. For bonus, we implemented **MTB Algorithm** for alignment, as for tonemapping, we implemented and tested with **Gamma Mapping**, **Photographic Global Mapping**, and **Tonemapping with Bilateral Filtering** as suggested. To refine our approaches and parameters for image processing, we captured 10 different sets of pictures. From these sets, we selected several sets of photos that we deemed satisfactory during our testing process. These sets were included in the homework file for evaluation and reference purposes.

Further detail and elaborations are provided below.

## Photography and Exposure Handling

We use Sony's XQ-CT72's convenient exposure-adjusting feature for photography. This device automatically assigns values such as shutter speeds and aperture sizes, and then calculates these properties into the Exposure Value (EV). As a result, we can effortlessly record the exposure value of each original photo as a power of 2 for further operations. For instance, if a photo has an EV of +2, it implies that its exposure value can be set as  $2^2 = 4$ .

```
# LDR filenames      exposure
20240401_185439.JPG 3.482202
20240401_185443.JPG 2.828427
20240401_185447.JPG 2.462288
20240401_185452.JPG 1.741101
20240401_185455.JPG 1.414213
20240401_185459.JPG 1.148698
20240401_185503.JPG 0.812252
20240401_185508.JPG 0.659753
20240401_185511.JPG 0.574349
20240401_185515.JPG 0.406126
20240401_185518.JPG 0.378929
20240401_185521.JPG 0.307786
20240401_185523.JPG 0.25
```

The left is some text in “data/library3/origin/image\_list.txt,” which documents the operation of our code on the original photos. We sorted the pictures by their exposures and recorded the corresponding values. Additional information regarding the execution of the code can be found in the “README.md.”

## Alignment

First, we use “align.py,” which has the MTB algorithm within to align the original images.

### Implementation Details

First, we write a bitmap conversion function. It will transform the input colored image into grayscale, choose a threshold value, and then convert the gray image to a bitmap. Pixels with value higher than threshold will become white, and those below the threshold will become black. Additionally, pixels with values close to the threshold are ignored in subsequent MTB calculations and are displayed as gray in the bitmap.

Second, we implemented the MTB algorithm as recursive function, which recursively halve the size of colored images, convert them into bitmaps, and try the best shift value in 9 directions with an offset shift value which is double of the return shift value of deeper recursion.

One thing to notice is that in our implementation, user needs to choose a photo, preferably the one with normal exposure, as standard image. All other photos will then attempt to align with this standard photo. The




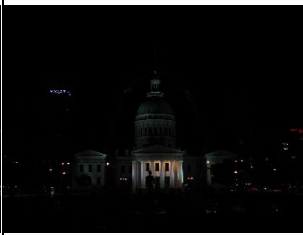








process is that the photo with highest exposure will align to the second-highest exposure one, the second one will align to the third one, and so forth until alignment with standard image is achieved. Similarly, the lowest exposure photo will align to the second-lowest one, and so on.

Regarding the ignoration of the pixels with values close to threshold, this feature is used during the calculation of differences between 2 bitmaps. We calculate how many pixels in the middle  $0.8H \times 0.8W$  area of 2 bitmaps, with size  $H \times W$ , are different and also not gray (need to be ignored). The margin is for the purpose of eliminating the influence of image shifts. Finally,  $H \times W - \text{difference}$  will be the return value which represent how close these 2 bitmaps are (smaller is closer).

### Observations

We found that there will be different effects when choosing mean or median for threshold of bitmap conversion. Theoretically, it’s better to choose median as the threshold. But the table below is an example we discovered, that shows, in the specific photo set, choosing mean for threshold performs better than choosing median.

This difference is likely because when using MTB in photo pairs with large exposure gaps, using median is not robust enough to generate fine result. Thus, to avoid the problem, we tend to take more photos in a set so that we can narrow the exposure gap. We also experimented on the two methods for best outcomes. Additionally, we found that the lower exposure the photo has, the larger the gray (ignored) area in the generated bitmap will be.

Bitmap Sets with Different Threshold				
EV	+4.09	+1.51	-1.82	-4.72
Original Photos				
Mean Approach				
Median Approach				

## Generating HDR Image

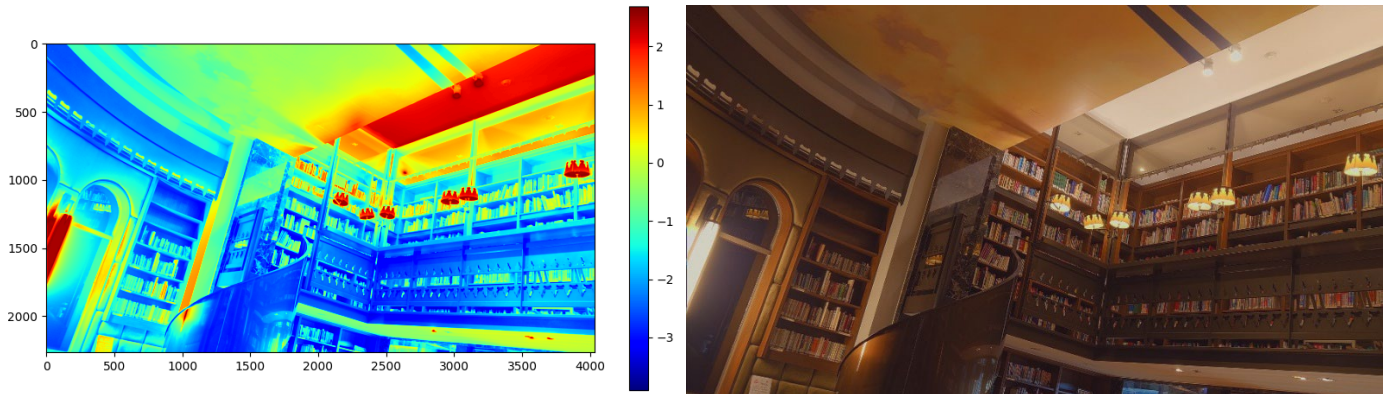
### Implementation Details

Our implementation first picks at most 400 sample pixel locations from the middle  $0.8H \times 0.8W$  area in the size  $H \times W$  of the photos. Similar as alignment, the margin is for avoiding the blank area caused by alignment. Then, it solves the response curve from the values of photos at these 400 pixels locations and the

exposure time of photos using the least square solution function from numpy. Third, it constructs a radiance map from those photos with the response function. This process is applied on 3 color channels separately, and finally the 3 result radiance maps will be combined and output as an HDR image.

Below is our chosen artifact “result.png” from the photo set “data/library3.” We have plotted its radiance map and its response curve.

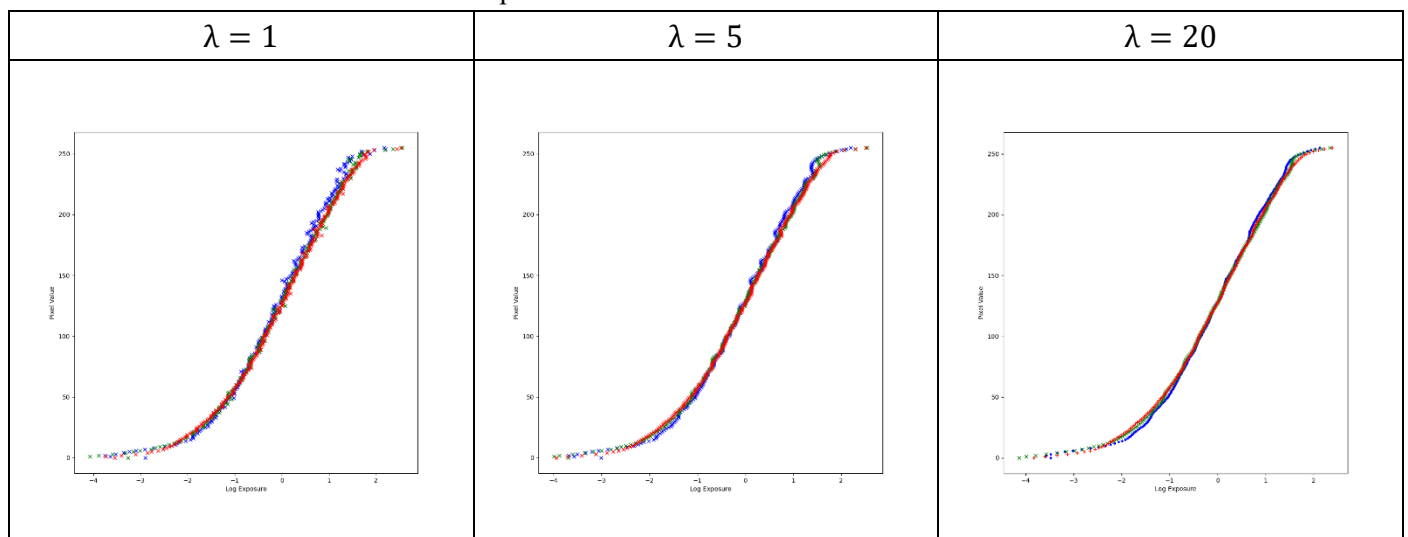
Radiance Map of the HDR Image “data/library3/hdr/hdr.hdr” (left) and the Tonemapping Result (right)



## Observations

We observed that the  $\lambda$  value used in HDR generation affects the color scheme of the image, though it’s difficult to concisely describe its changes, by comparing the plotted response curve generated, we can see that the value affects the smoothness of the curves.

Response Curves with Different  $\lambda$  Values



## Tonemapping

### Normalization

After process of most tonemapping alprithms, the level of pixels may still have values slightly larger than 1. In order to transform the image into LDR format, we need to normalize the levels so that they will all be smaller than or equals to 1. We use two ways to achieve this goal, the first way is directly normalizing the result image linearly, and the second way, which is used in Tonemapping with Bilateral Filtering, is making the result undergo another global mapping first, then normalize the result linearly.

Furthermore, we discovered that another two different ways of normalizing will have different effects. First is normalizing each of the color channels (namely, RGB) individually, and the second way is normalizing all channels simultaneously. When using the former, the output result will often become “green-er”, we guess that this phenomenon may have a resemblance to the “Bayer’s Pattern” in demosaicking mentioned in class, as said pattern is mostly composed of green pixels.

The Difference between the normalizing methods in Gamma Mapping

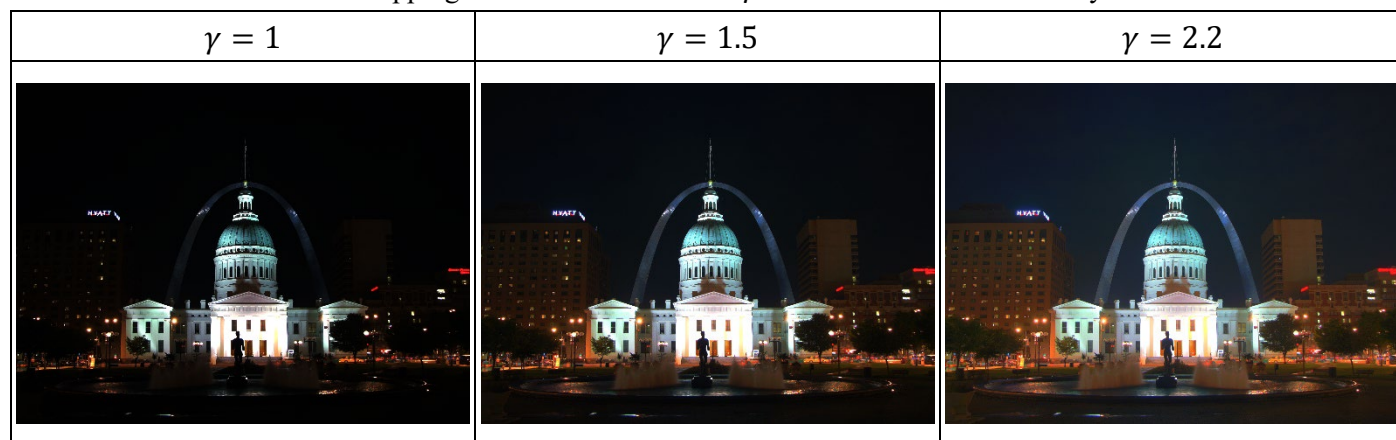


## Gamma Mapping

When implementing Gamma Mapping, we discovered two different algorithms, which leads to distinct effects.

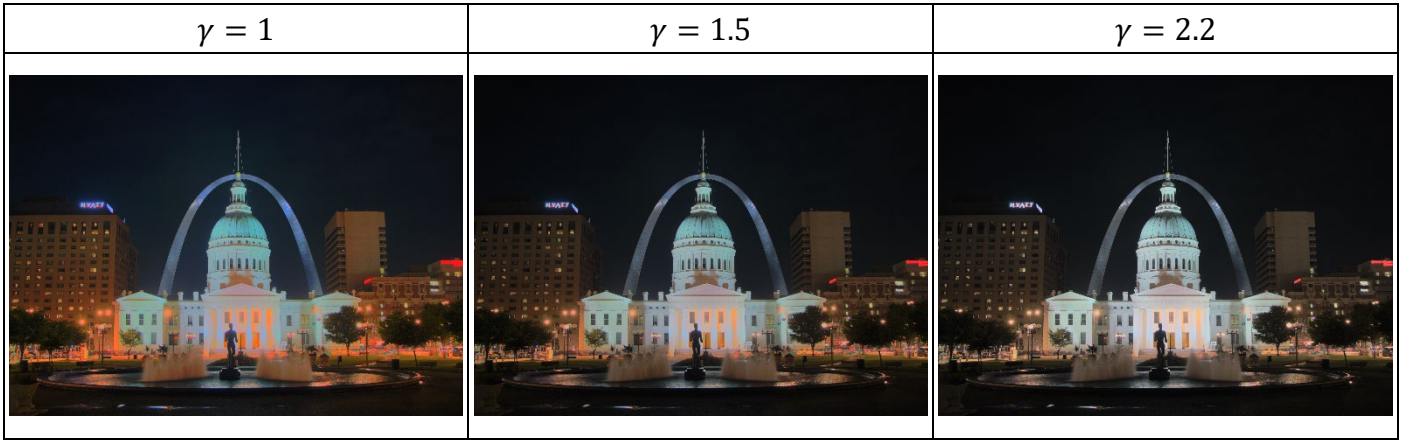
The first approach, we called it “Gamma Intensity”, is applying Gamma correction  $x \rightarrow x^{1/\gamma}$  solely to the luminance first, followed by the restore of color data. This approach exhibits high color saturation when the  $\gamma$  value is high. However, at lower  $\gamma$  values, the result might be high contrast and very dark.

Tonemapping Results with Different  $\gamma$  Values in “Gamma Intensity”



The second approach, we called it “Gamma Color”, directly applies Gamma correction to each color channel. As a result, the outcomes are about opposite to the first approach. At higher  $\gamma$  values, the overall image tends to have lower saturation, while at lower  $\gamma$  values, it has higher saturation.



Tonemapping Results with Different  $\gamma$  Values in “Gamma Color”

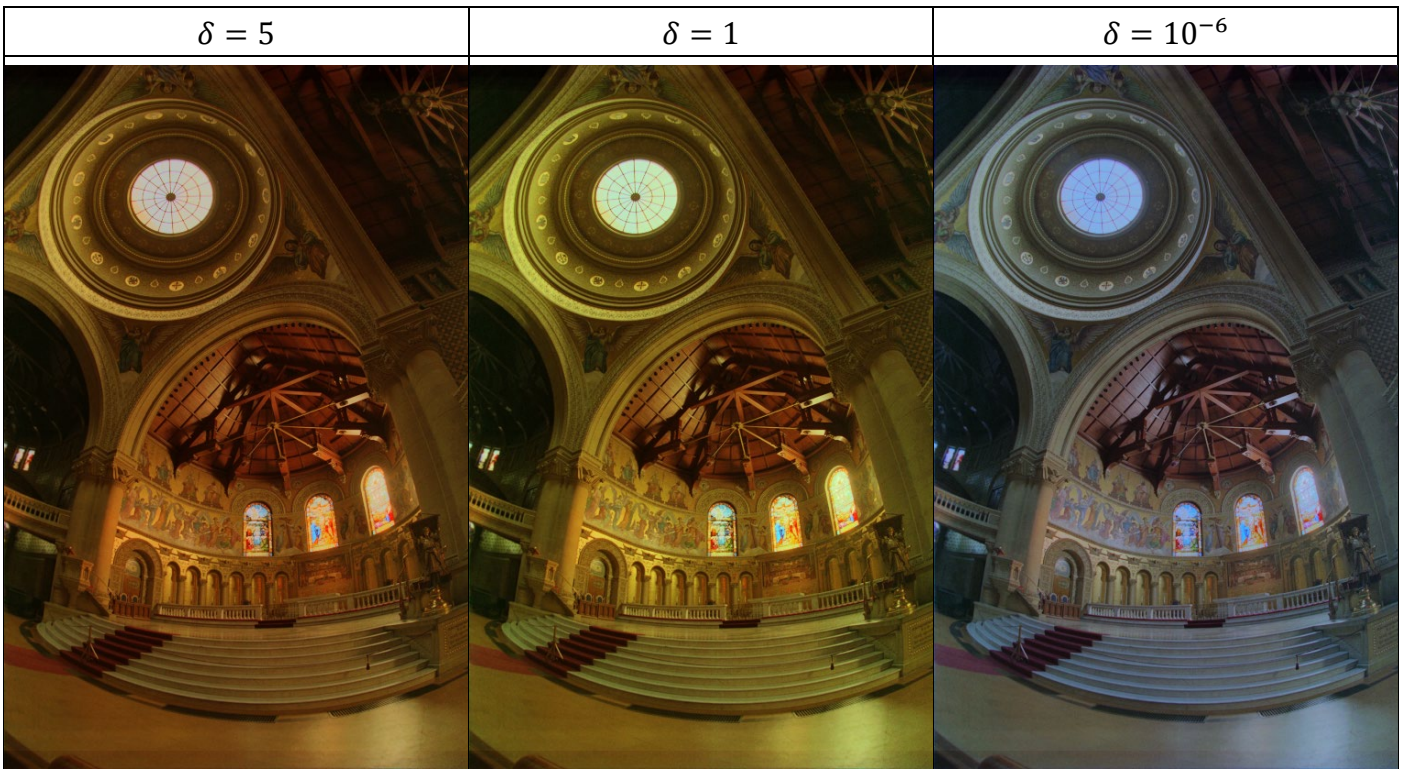
### Global Mapping

We write a function that apply Global Mapping on single channel. In the Global Mapping on a whole HDR image, we first apply single-channel Global Mapping on its grayscale (luminance), and restore colors back. But we found that after restoring colors, some pixels still have values slight larger than 1 as we mentioned above when talking about normalization. Thus, we apply single-channel Global Mapping on 3 channels again and linearly normalize them as the final step. This normalization method is also used in the Tonemapping with Bilateral Filtering.

We discovered another point that worth noticing is the value of delta, which is the  $\delta$  in the following equation (also mentioned in “README.md”),

$$\overline{L_w} = \exp \left( \frac{1}{N} \sum_{x,y} \log(\delta + L_w(x,y)) \right)$$

would affect the output images. You can compare the examples below.

Global Mapping Results with Different  $\delta$  Values

## Tonemapping with Bilateral Filtering

Our implementation first uses bilateral filter to filter out the low-frequency (base) part of the luminance of HDR image, and gets the high-frequency (detail) part by simple subtraction. Then it compresses the low-frequency part and combines them together. Finally, after restoring color, we apply the single-channel Global Mapping on the 3 color channels and linearly normalize them again to get a better result.

High-frequency Part (left) and Low-frequency Part (right) of the HDR Image “data/library3/hdr/hdr.hdr”



## About our Artifact

We finally chose the generated tonemapping result of the “library3” photo sets as the “result.png” artifact. Here’s an overview of our process in creating this image.

Initially, we experimented with Gamma Mapping, Global Mapping, and Tonemapping with Bilateral Filtering as well as the high (detail) and low (base) frequency images generated from the latter. We also compared the results with those generated by using Drago Tonemapping from OpenCV. Eventually, we found that the result of Bilateral Mapping looked the best for this particular photo set.

Subsequently, we began adjusting the parameters for Tonemapping with Bilateral Filtering.

At the End, the parameters for the finished artifact are (1, 3.5, 13, 40, 5, ALL), which can also be found within the “data/library3/hdr/tonemap.txt.”

Last but not least, we had fun when collecting photos and ventured into places that we didn’t know existed in NTU campus. And by experimenting with the pictures, we discovered the potency of image-processing that for example, by simply changing a picture’s tone, can have a result whose characteristic is vastly different from its original. Overall, this is a challenging yet rewarding project, and thanks to the course’s team that provided guidance during the project.

## Appendix

We worked on and experimented with some other parameters in this project, their descriptions are noted in the “README.md.” Reading the file will also help you understanding how to execute our code.

The folders with name starting with “test” holds the pictures that we collected from the internet (links are listed below) prior to our photo collecting, which have helped us with establishing the project’s structure and how to set certain parameters for desired results.

1. Images in “data/test1/origin” are from <https://www.pauldebevec.com/Research/HDR/>
2. Images in “data/test{2,3}/origin” are from [https://en.wikipedia.org/wiki/Multi-exposure\\_HDR\\_capture](https://en.wikipedia.org/wiki/Multi-exposure_HDR_capture)
3. Image in “data/test4/hdr” is from <https://filesamples.com/formats/hdr>