VFX2021: High Dynamic Range Image

Yu-Chu, Yu
r09922104@ntu.edu.tw
Graduate Institute of Computer Science and Information Engerneering
Taiwan

Ching-Yi, Tsai
r09944002@ntu.edu.tw
Graduate Institute of Networking and Multimedia
Taiwan

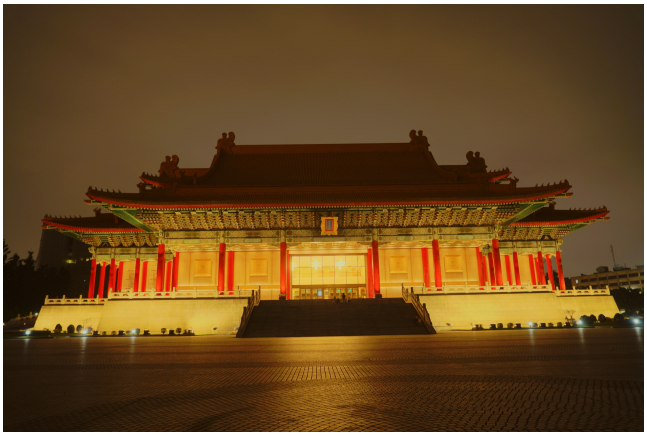

Figure 1: National Theater Hall. An empirically selected final High Dynamic Range(HDR) image.

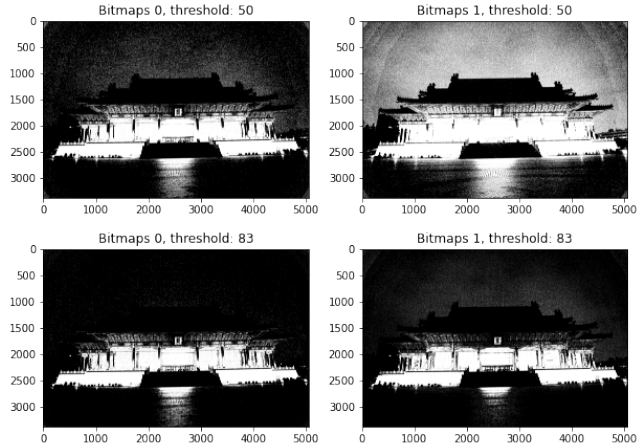

Figure 2: The comparison between different thresholds on a lower exposure pair of images.

ABSTRACT

This report summarizes and explains the implementation details of a take-home assignment which produces High Dynamic Range (HDR) images. We took our sample images of National Theater Hall with various shutter times. We then construct an HDR image through Median Threshold Alignment, Responsive Curves Recovering, Radiance Map Reconstruction, and Global Renihard Tone Mapping to generate an HDR image. The final work is shown as Figure 1.

KEYWORDS

High Dynamic Range Imaging; Image Alignment; Tone Mapping

ACM Reference Format:
Yu-Chu, Yu and Ching-Yi, Tsai. 2021. VFX2021: High Dynamic Range Image. In *Proceedings of NTUCSIE (NTUCSIE VFX'21).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

In the following sections, we introduce our implementation process, showing our experimental results, comparison among different methods, and conclude our findings.

NTUCSIE VFX'21, March 2021, Taipei, Taiwan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 MEDIAN THRESHOLD BITMAP ALIGNMENT

In this section, we explain our implementation of Median Threshold Bitmap (MTB) Alignment algorithm. We follow Greg Ward et al.'s approach [4] and further apply several heuristics observed from experiments which is generalized as principles in the last subsection.

The methods we adopt include determining the optimal offset between two images and aligning a given series of images.

1.1 Two-image Alignment with Multi-Scale Image Pyramid

We construct an image pyramid from each image's greyscale at various lower resolutions. The process starts with the pair of pictures with the lowest resolution and, in turn, the optimal offset on the original two photos.

We create two bitmaps by their median greyscale threshold, respectively, where 0s stand for values less than or equal to the threshold, whereas 1s stand for values greater than the threshold. As asked in class, we create two additional exclusive masks to ignore values closed to the threshold. The two images' error is calculated by the sum of differences between 2 bitmaps, excluding those who lie on the exclusive masks. The optimal offset at each stage can be further determined by the second bitmap shift that causes the slightest error.

To find the optimal offset, we shift the second image $\pm r$, with $r \in \mathbb{N}$ in the x and y position. The process searches for in total

$(2r+1)^2$ offsets. Also, the larger the r, the longer the whole process, and the more precise the optimal offset. Since we take the above process only one time per series of images, to get much more precise results, we set r to be 4, which searches for a total of 9^2 offsets at each stage.

After the optimal offset is determined, we back-propagate this offset to the previous pair of images with higher resolution, multiplying this offset by 2, shifting the second image by the offset, and recursively compute the optimal offset by the process mentioned above.

1.2 Alignment for a Series of Images

We can align any given pair of 2 images by the above algorithm. However, for images with entirely different exposure rates, there may be an unneglectable difference between their bitmaps. Thus, we consider a sequentially pairwise image alignment process, which aligns the adjacent pair of images with the closest exposure rates each time.

After finding the optimal offsets of all the adjacent pairs of images, we set the first image as a reference and align the other photos based on the optimal offsets' cumulative sum.

1.3 Implementation Guideline

After a few empirical experiments, we summarize several implementation guidelines for future reproduction.

1.3.1 Choose an Appropriate Threshold. We have to carefully pick up a specialized threshold value for the pair of images with extremely light or dark exposure instead of directly using the median value. Greg Ward et al. have suggested choosing either the 17th or 83rd percentile as the threshold, respectively. Figure 2 shows our experiments on the comparison between the different thresholds on a lower exposure pair of images. It's more reliable with a higher threshold when processing a lower exposure pair of images.

1.3.2 Avoid Directly Scaling a Bitmap. While implementing a bitmap image pyramid, we came up with two strategies: (1) we generate bitmaps from the original images and directly scaling their bitmap to make a comparison (2) we first scaling the authentic photos and further calculate their bitmaps at each scale. Intuitively, it seems the former is a much more efficient strategy. However, Greg Ward et al. have suggested that never subsampling the bitmaps themselves, or the algorithm might fail.

1.3.3 Determine the Range to Exclude Threshold Noise. Greg Ward et al. suggested that we exclude bits with the pixel value ±4 of the threshold value. We argue that a fixed exclusive interval might exclude too many bits for images with a dense distribution of exposure or exclude too few bits for pictures with a sparse one. To avoid the cases, we use a relative exclusive interval by excluding ±10 percentile of the threshold percentile, making sure that we exclude a fixed number of bits every time.

2 HDR IMAGE RECONSTRUCTION

We follow Debevec's algorithm [2] to reconstruct our HDR image. The whole procedure consists of 3 parts:

(1) sampling robust pixels across images with different exposure rates.

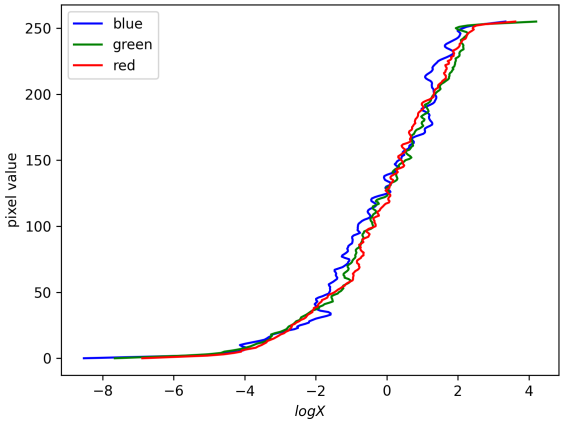

Figure 3: The responsive curves with respect to each channel.

(2) choosing an appropriate weighting function to eliminate the influence from outliers.

(3) solving a linear system to reconstruct responsive curves and the exposure rate of each pixel.

2.1 Sampling Strategies

We noticed that Debevec's algorithm is quite sensitive to the way we sample pixel values. That is, different selection strategies will significantly affect the final responsive curves. Our sampling strategy generally follows two criteria to make our results more robust, which our instructor, Prof. CYY, inspires.

First, we didn't sample points on edge. From an implementation point of view, we apply a Canny edge detector on the original images and further construct 0/1 masks to exclude the pixels that lie on edge.

Second, we only consider the points in which intensity varies from images with different exposure rates. To achieve the goal, we sorted all the pixels concerning their standard deviations across images. By choosing pixels with higher standard deviations with high probability, for instance, selecting the top 20% points with probability 80%, we can still keep the randomness in a way.

2.2 2-Stage Intensity Weighting Function

At first, we tried to apply a triangle window to weigh the pixel value. As shown in Figure 3, this window is good enough to reconstruct the responsive curves. However, when we rebuild the whole image (by calculating weighted $logE_i$ where i is the indices in the entire image), we soon noticed lots of noises in both light and dark places. According to the following equation, which aims to find $logE_i$,

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

However, when applying a triangle window, the pixel value with extraordinarily high and low intensity would get both small weights while calculating $logE_i$. Though we've added a minimal value at

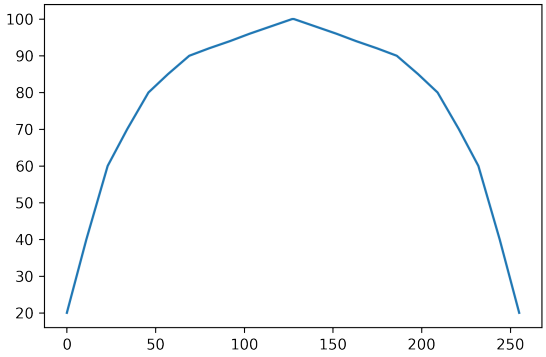

Figure 4: Our customized non-linear weighted function, with range from 20 to 100.

the denominator to avoid divided-by-zero, it still makes $logE_i$ very unstable.

Thus, while calculating the value of $logE_i$, we then applying another non-linear weighted function, with the minimum 20 occurring at $x = 0, 255$, and the maximum 100 occurring at $x = 127, 128$, which is shown in Figure 4. We call this strategy a 2-state weighted function. We utilize a triangle window when reconstructing responsive curves and apply a non-linear window with the minimum larger than 0 when reconstructing the whole image. Figure 5 shows the partial result using only a triangle window, and Figure 1 shows the result after applying such strategy.

2.3 Reconstruction

After determining the process to sample points and weight pixel values, we finally apply Debevec's algorithm to reconstruct an HDR image. There are two steps in total, including reconstructing the responsive curves and reconstructing the whole picture.

2.3.1 Responsive Curves Reconstruction. As discussed in class, our goal is to minimize the following objectives,

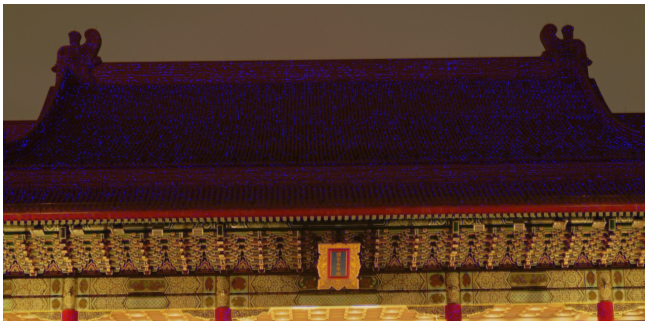

Figure 5: The partial result using only a triangle window. One can easily notice that there exist a huge amount of noises in extremely dark places.

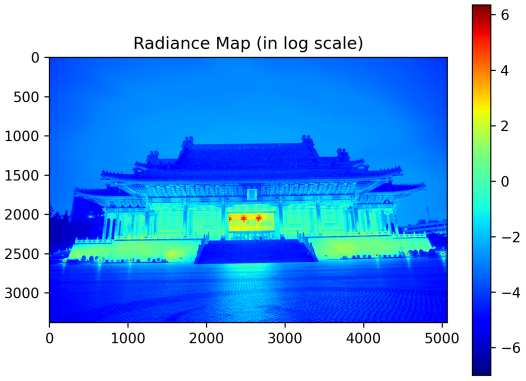

Figure 6: The reconstructed radiance map, shown in a log scale.

$$O = \sum_{i=1}^{N}\sum_{j=1}^{P}\left(w\left(Z_{ij}\right)\left(g\left(Z_{ij}\right) - \log E_i - \log \Delta t_j\right)\right)^2 + \lambda \sum_{z=1}^{254}\left(w\left(z\right)g''\left(z\right)\right)^2 \tag{2}$$

The first term aims to get the correct relationship among all variables according to the equation $g(Z_i j) = \log E_i + \log \Delta t_j$, while the second term aims to get a much smoother curve. In our experiments, we set $\lambda = 5$ to get a more reasonable result. By solving a linear system, we find one of the optimal solution $g^*, E_1^*, \cdots, E_N^* = \arg\min(O)$. After repeating this procedure 3 times regarding each channel (R, G, B channels), we reconstruct all responsive curves as shown in Figure 3.

2.3.2 HDR Image Reconstruction. After reconstructing the correct relation between the 8 bit pixel value and the real word exposure rate, we finally reconstruct the original LDR image to a HDR image according to Equation 1. Figure 6 shows the radiance map of our HDR Image (the range is shown in a log scale).

3 TONE MAPPING

In this section, we explain the tone mapping methods used along with implementation details, where we follows Global Reinhard's Tone Mapping [3].

3.1 Global Reinhard Tone Mapping

We implement tone mapping algorithm with Global Reinhard Tone Mapping. Generally, Reinhard's algorithm uses a local operator instead of a global one.
Our goal is to normalize the HDR image, and compress the range from $\mathbb{R}$ to $[0, 1]$, where utilize the following equation:

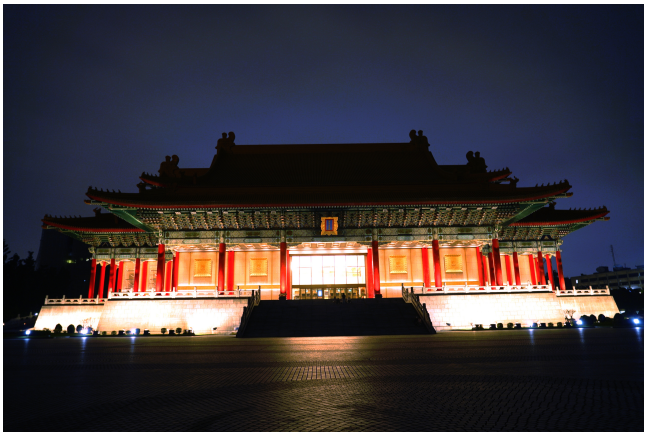

Figure 7: The result after applying Global Reinhard's algorithm.

$$L_d(x,y) = \frac{L_m(x,y)\left(1 + \frac{L_m(x,y)}{L^2_{white}(x,y)}\right)}{1 + L_m(x,y)} \tag{3}$$

Where L_d is the final LDR image, L_m is a normalized HDR image times a key value a related to the illuminance, and L_{white} is the smallest illuminance to be mapped to 1. By setting different L_{white} to each channel, we can control their brightness respectively. Our implementation is shown in Figure 7, with key value $a = 0.18$.

3.2 Final Result

After a few experiments, our tone mapping algorithm consistent with Reinhard's algorithm implemented in OpenCV [1]. The result is shown in Figure 1, with parameters gamma = 1.3, intensity = 0, light_adapt = 0.5, and color_adapt = 0.1.

4 CONCLUSION

We investigate several algorithms to generate an HDR Image in this project, which is much closer to human's actual visual perception. We take several photos with different exposures, implementing the MTB algorithm to align our photos, reproducing Debevec's algorithm to reconstruct responsive curves, and try several different ways to map an HDR image to an LDR image. The final result is significantly better than any one of our original photos and even much better than the result purely generated by OpenCV's well-implemented packages, which shows our success in reproducing such a classic application.

5 REPRODUCE COMMAND

We use python 3.8.2 to code the program. Please make sure that all the related packages have already been installed, and the data path are specified correctly. Typing "`python3 hw1.py`" should reproduce our final result. Notice that we earn our extra bonus credit with the implementation of MTB Alignment and Global Reinhard Tone Mapping.

REFERENCES

[1] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).
[2] Paul E Debevec and Jitendra Malik. 2008. Recovering high dynamic range radiance maps from photographs. In *ACM SIGGRAPH 2008 classes*. 1–10.
[3] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. 2002. Photographic tone reproduction for digital images. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 267–276.
[4] Greg Ward. 2003. Fast, robust image registration for compositing high dynamic range photographs from hand-held exposures. *Journal of graphics tools* 8, 2 (2003), 17–30.