NTU CSIE 2022 Spring, Digital Visual Effects
Project #1 High Dynamic Range Imaging

Group 33
B07902078 資工四 沈韋辰
B07902072 資工四 陳光裕

Abstract

In this project, we will use the pictures with different exposure times to reconstruct the HDR image, and implement tone mapping on it. Here is a brief description of our routine:

First, we take pictures with different shutter speed values. Then, we use the MTB method to align those pictures to reduce the offset of each sample point among the pictures. As for sampling, we use a downsampling method. That is, we resize the pictures into smaller ones, and use all the pixels in the small image to ensure the sample points we gain are well-distributed. We use Debevec's algorithm to solve the response curve of the camera, and use this curve to reconstruct the HDR image. Here, we use 5 methods to implement tone mapping: Drago's, Mantuik's, Reinhard's, and two our self-written algorithms.

1 Configuration

1.1 Camera

Camera: Canon EOS 450D, with tripod
Lens: TAMRON 18-270mm F3.5 – 6.3 Di II VC PZD

1.2 Develop environment

OS: Linux (Ubuntu 18.04), Windows 10
Language: Python 3.7
Modules: OpenCV, NumPy, matplotlib, Numba
Shutter Speed: 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 1, 2
Resolution: 4272 x 2848

1.3 How to run our code

We use `pipenv` to manage our package. If your don't have `pipenv` , run this:
`$ pip install pipenv`

To install the dependencies, run this: **`$ pipenv install --dev`**. If you have trouble installing them due to the python version, modify `[require] > python_version` in `Pipfile`.

After installation, you can use **`$ pipenv run python main.py`** to run the code.
Enter the id of the album to run the code, it will download the images from our Google Drive. If the download fails, delete the folder `Images/` and run it again, this is an uncertain crash that we still don't know how to fix.

2 The Algorithm

In this section, we will introduce the algorithms we use in this project. Except for part of tone mapping, all the algorithms are written by ourselves.

2.1 Median Threshold Bitmap Alignment

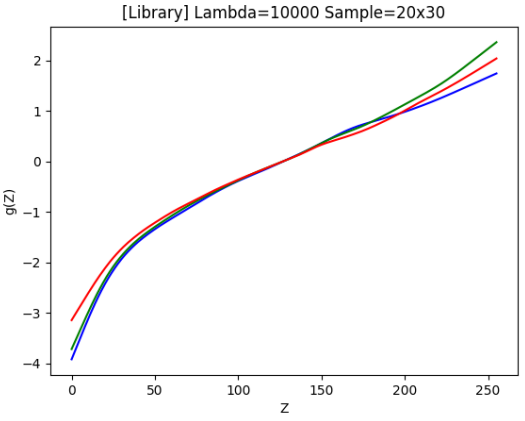
This is the method introduced in the class. In our case, we choose the image with median exposure time as the template, and resize it into 1/64, 1/32, 1/16 … to 1. The template will also build up the mask with the ignorance of 10. For each small image, we shift its MTB to the 9 directions and compute the smallest one. We found that the image darker, the alignment harder. So, this method still has its shortcomings.

2.2 Sampling

Instead of choosing the sample point randomly, we use downsampling because the former may not get a uniform distribution like too many points are in the brighter part or darker part. Our idea is to resize the image into 30 x 20, and use all these 600 pixels to calculate the response curve.

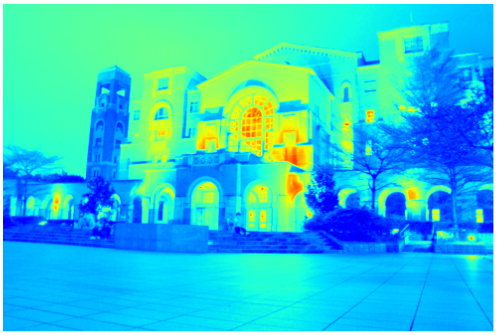
2.3 Response Curve

We use Debevec's method to get the response curve. First we split the image to three channels and build their overdetermined system, solve its least square solution by `np.linalg.lstsq`. Note that there is a constant lambda which can smooth the curve. In our case, we use 10000. In this part, we found the larger contrast will result in these three curves varying more, especially in higher and lower Z-value.



2.4 Radiance Map

After getting the response curve, we could compute all the intensity of each pixel. However, in each case, there are nine 4272 x 2848 pictures. It has a massive time complexity to gain the intensity of all pixels. Thus, we use the module `numba` to speed up the calculation. By the way, we use the same weight function we discussed in the class. As we compute the radiance maps for the three channels, we save it as a `.hdr` file.



2.5 Tone Mapping

As we have a `.hdr` file, we can use a tone mapping algorithm to produce an LDR file. We implement three OpenCV built-in tone mapping algorithms and two our hand-written algorithm which is based on Reinhard algorithm. After tone mapping, we save these pictures as a `.jpg` file.

2.5.1 Drago's Algorithm (OpenCV)

Drago Algorithm use Drago Operator, it display luminance as $L_d = \frac{log(L_w+1)}{log(L_{Max}+1)}$.

2.5.2 Mantiuk's Algorithm (OpenCV)

See R. Mantuik's paper at section 6.

2.5.3 Reinhard's Algorithm (OpenCV)

Reinhard Algorithm uses Photographic tone mapping operator, dodging and burning. We have discussed this method in class.

First, we turn radiance map's RGB into grayscale, then the quantity is computed by:

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

and the display luminance is

$$L_d(x,y) = \frac{L(x,y)(1+\frac{L(x,y)}{L_{white}^2})}{1+L(x,y)}$$

2.5.4 Naive (Hand-written)

Besides the algorithms above, we implement our hand-written algorithm. This easy one is instinctive. Just map L_w from $[0, \infty)$ to $[0, 1]$. The display illuminance is $L_d = \frac{log(L_w * \mu+1)}{log(\mu+1)}$.

2.5.5 Non-Normalized (Hand-written)

This algorithm is based on Reinhard but without normalization and dodging-and-burning. By the way, we use the brightest pixel for L_{white}.

3 Extensions

- Ward's MTB algorithm (Section 2.1)
- Two hand-written tone mapping algorithms (Section 2.5.4 and 2.5.5.)

4 Results

Tone Mapping Algorithm	Indoor	Outdoor (Night time)	Outdoor (Day time)
Original			
Drago			
Mantiuk			
Reinhard			
Naive			
Non-Norm			

5 What we learn and discussion

5.1 How to choose good scene

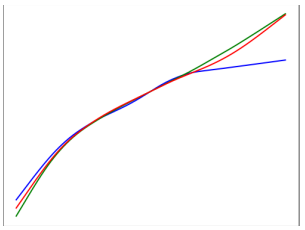
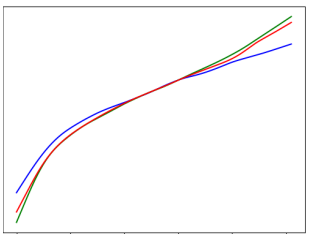
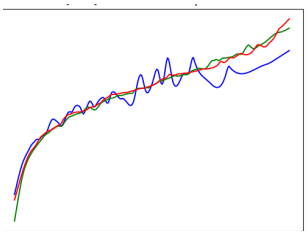
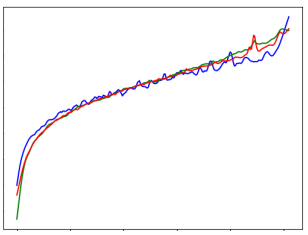
We chose three scenes for this project: NTU main library, Sweetgum Avenue and the room. Library is shooted at night and a little rainy then. Also, there are some people sitting there. Weather and wind may affect the picture a little, but its influence is less at night. Distance to a building is so large that it aligns well when we shoot. On the other hand, The second one is severely affected by moving clouds, people, and leaves on the tree. So, we receive a blur hdr image and get a relatively bad result. The final case is in the room, which is the most stable environment. However, because we are really close to the object. Camera vibration may take its toll on alignment. Also, the light source is too close to the object, which makes part of the object over-exposure.

5.2 Camera stabilization

While taking pictures, wind may cause camera shake. Also, pressing the button also causes shaking. We do our best to avoid it, for example, use delay shoot mode. But it still has some little differences between pictures. Thus, we use the method described in section 2.1.

5.3 λ of response curve and density of sample point

We have talked about our sampling method in section 2.2, we also found that the relations between sampling size and λ**.** Larger sampling size makes the curves closer, and larger λ makes the curves more smooth. But too large of them are not always a good thing. Big sampling size increases the time complexity a lot, and big λ may decrease the preciseness of the response curve. So, we spend a little time finding a suitable value for them.

Lambda \ Sampling size	6x9	20x30
10000		
100		

5.4 Photo alignment

When we do the MTB alignment, we notice that when we try to align a very dark image to a very bright image, this method often fails. So, we came up with an idea, what if we align the

second image to the first one, and the third one aligns to the second one, and so on. To our surprise, the result is even worse, the whole image series shifts to one direction more and more. After some experiment, we decide to use the image which has the median exposure time to be the template of alignment.

5.5 Save `.hdr` file

The radiance map needs to take the exponential value then save as the `.hdr` file. We use radiance map to construct the `.hdr` file at first so we get trouble for days until we remember that it should be the exponential value. Then we use OpenCV to save it instead of `np.save` because `np.save` always saves files as `.npy` files.

5.6 Tone mapping algorithms

After we save the `.hdr` file, we need to do tone mapping, but we don't know how to use an existing tone mapping application. In this case, we decide to write algorithms by ourselves. We tried Reinhard without dodging and burning, a strange algorithm we found on some website, and three OpenCV built-in algorithms. Sadly, we have been fixing our algorithm for already 3 days but still have some problems. During the fixing of algorithms, we try many different parameters of OpenCV built-in algorithms, trying to find the best performance of the pictures. Besides, we use the same tone mapping algorithm to compare different scenes. We have shown the result in section 4.

6 Resources

1. https://ieeexplore.ieee.org/document/6196908
2. https://people.mpi-inf.mpg.de/~mantiuk/papers/mantiuk08mgtmo.pdf
3. http://kkyang.github.io/HDR/
4. http://users.eecs.northwestern.edu/~ollie/eecs395/HW4/HW4.htm
5. http://www.pauldebevec.com/Research/HDR/debevec-siggraph97.pdf
6. https://www.twblogs.net/a/5cc881d6bd9eee1ac30b9ce7
7. https://ssarcandy.tw/2017/04/16/High-Dynamic-Range-Imaging/
8. https://docs.opencv.org/3.4/d2/df0/tutorial_py_hdr.html
9. Course lectures and slide