Project 1: High Dynamic Range Imaging

Steven R04922170 / Han B01902078

March 29, 2016

1 Project Overview

This project can be simply divided into 4 parts:

- Taking Pictures w/ Different Shutter Speed.
- Perform Image Alignment
- Produce .hdr Image and Output The Response Curve.
- Tone Mapping.

In the following paragraphs, we're going to explain what we have done for each of these 4 parts in detail.

Taking Pictures w/ Different Shutter Speed

Figure 1: Pictures with different exposure

Perform Image Alignment

For image alignment, we implemented the Greg Ward's MTB algorithm (link) in C++ to do image registration. All utility functions related to image alignment are implemented inside the namespace **ImageAlignment**, including a **Bitmap** class implemented for computing MTB. This method is a recursive one, where we subsample the image with a factor of 2 in each dimension and find the best moving offset to return. We first convert the input color images to grey scale, using this equation:

$$grey = (54 \times r + 183 \times g + 19 \times b)/256$$

. And during the phase of determining the best moving offset, we compute the XOR bitmap and then count the sum of that XOR bitmap, since the XOR operation helps us retrieve the distance between 2 threshold bitmaps, and smaller sum represents a more similar pair of bitmaps.

Produce .hdr Image and Output The Response Curve

In this project, we implement the method provided by P. E. Debevec and J. Malik. We recover inverse of the response curve by assuming the g function is continuous and smoothing, solving the objective function to recover it. First, we load in 16 images in the same scenes with different exposure time and randomly sample 50 pixels in each image to be the input of `gSolve` function, and set lambda to 100. The weighted function w() for the objective function has maximum value 1 at intensity value 128 and decrease to 0 at both sides. Second, using the the result of the `gSolve` function to recover each pixel's lnE value for all input images, then use the weighted average to be the result. In this part, we have filtered the outlier for each pixel to calculate the weighted average of lnE by checking the ratio of leave-one-out average and total average. This additional filter can help us remove the moving objects in the scenes. Finally, we turn the float value from the previous step to the RGBE value for each pixel, after this process we will get 32bits/pixel and output our final .hdr result with some file headers.

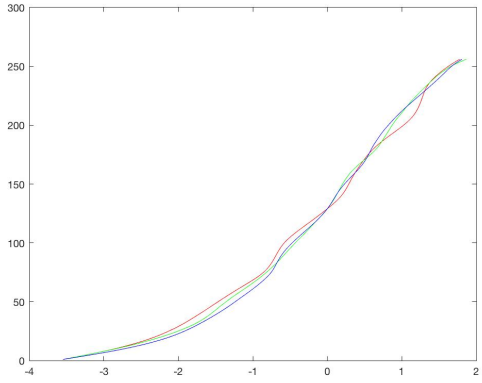

Figure 2: The result of response curve

Tone Mapping

In order to demonstrate the effect, we also use the build-in tone mapping function, `tonemap(hdr,'AdjustLightness',[0.1 1],'AdjustSaturation',2)` to output the RGB image of our HDR result.

2 How to Run Our Program

Compile and Run with our `Makefile`

First, using `make alignAll` will help us align the pictures in `./input/scene{1,2}/` and store the aligned resulted images in `./output/scene{1,2}/aligned/`.

Next, using `matlab` command to run the Matlab code for HDR assembling.

- Step1: go into the `matlab/` directory using `cd matlab/`.
- Step2: execute the matlab code with `matlab -r "hw1('../output/scene2/aligned/', 'SAM_00', '15', '16', '.JPG', '../input/scene2/expose.txt')"`. After the executing of Matlab, you will see the resulted 4 files in the same directory, and they are: `mapping.fig`, `result.hdr`, `result.jpg`, and `result.rgbe`.

Compile and Run Manually

For image alignment, compile the C++ source first, using this command

```
g++ main.cpp `pkg-config --libs --cflags opencv` -o a.out
```

, and then execute `a.out` to align pictures. The usage of `a.out` is:

```
./a.out <input dirname> <detail txt file> <output dirname>
```

, where `<input dirname>` contains the original images and the results will be put inside `<output dirname>` directory. The `<detail txt file>` should contain the file names of all original images as well as the exposure information. For more examples, please take a look at `./input/scene1/detail.txt` and/or `./input/scene2/expose.txt`.

For HDR assembling, simply run the `./matlab/hw1.m` will do the trick. The usage of the function `hw1` is:

$$hw1(ImgDir, imagePrefix, startingNumber, imageCount, extension, exposurePath)$$

, where `ImgDir` is the directory in which the images reside, `imagePrefix` is the common prefix of all image filenames, `startingNumber` is the number of the first image after the prefix, `imageCount` is the total number of images, `extension` is the file extension of the images, and `exposurePath` is the path to the plain-text file which holds the detail exposure information as mentioned in the last paragraph.

3 Results

Here we provide our resulted LDR image after tone mapping in Figure 3. And also the comparison between the results of removing / not-removing moving objects can be seen in Figure 4a and Figure 4b.

4 Conclusions

This project taught us about the technology behind a camera.The most interesting thing we have learned from this project is that combining different images with filters to remove outlier can effectively remove the moving objects in the scenes. Also, the Ward's MTB algorithm for image registration is really fast because we use bit-operations. However, this image registration algorithm only take care of the integral moving offsets, regardless of rotational ones, which we should notice when applying Ward's method.

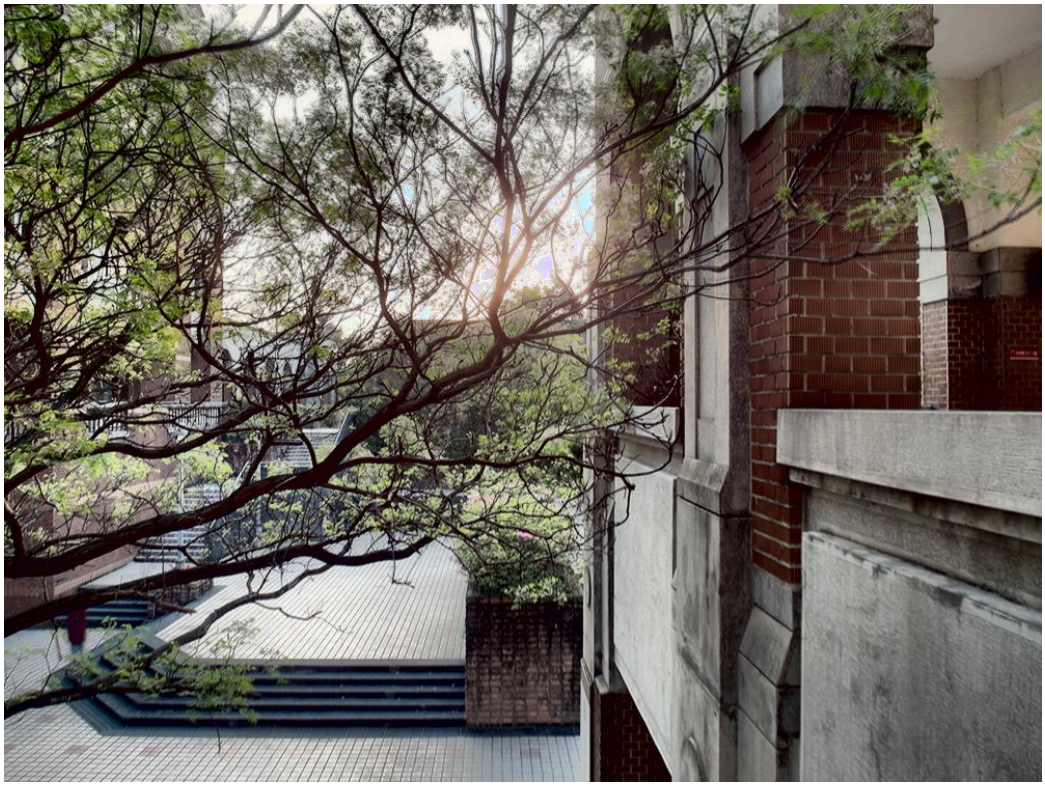

Figure 3: The result of LDR image

(a) Image with moving object (b) Result

Figure 4: The effect of removing moving object