Project 1 - HDR (1)

1. MTB Alignment

Algorithm:

1. Scale down the image to several levels.
2. Convert the images to greyscale and convert to 0,1 using median threshold.
3. For each level, find the best offset minimizing the XOR of two images.
4. The offset of previous level should multiply by 2, and we have final offset.

Our Implementation

The trusted pixel range is +-20 around median in our default setting.

The algorithm sometimes fail on the darkest or lightest images, so I also use OpenCV’s MTB to choose final HDR result one (I think it’s just the parameter’s difference).

Most of time our photos are good enough so that the offset are 0.

2. Robertson’s method

We implemented Robertson’s method:

Mark A Robertson, Sean Borman, and Robert L Stevenson. Dynamic range improvement through multiple exposures. In *Image Processing, 1999. ICIP 99. Proceedings. 1999 International Conference on*, volume 3, pages 159–163. IEEE, 1999.

Target function :

$$\hat{g}, \hat{E}_i = \arg\min_{g, E_i} \sum_{ij} w(Z_{ij}) \left(g(Z_{ij}) - E_i \Delta t_j \right)^2$$

w:weight, g: response curve, E: irradiance, t: exposure time.

Minimize G and E alternatively by solving gradient =0 until converge or reach max iteration.

Weight function

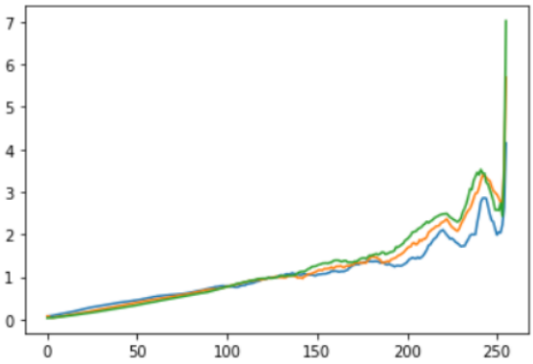
The weight function given by the paper is

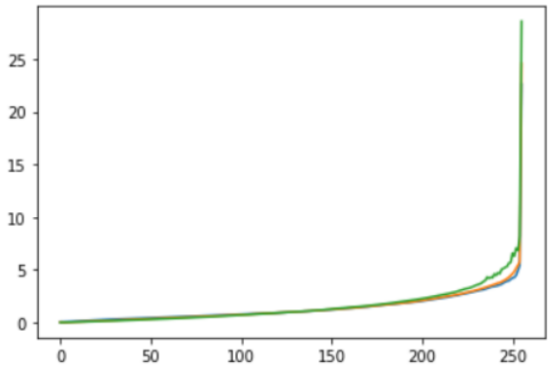
$$w_{ij} = w_{ij}(y_{ij}) = \exp\left(-4 \cdot \frac{(y_{ij} - 127.5)^2}{(127.5)^2} \right)$$

However, we observed that our algorithm acts differently to OpenCV's implementation.

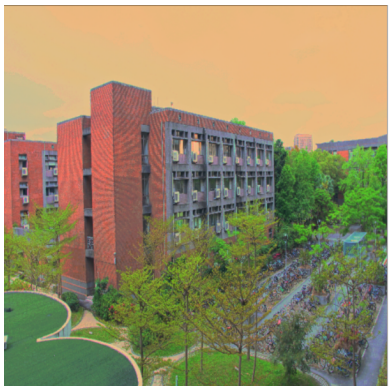
For example, the G (response curve) result is different. Our: LHS, OpenCV: RHS.

The curve should be smooth, like RHS one.



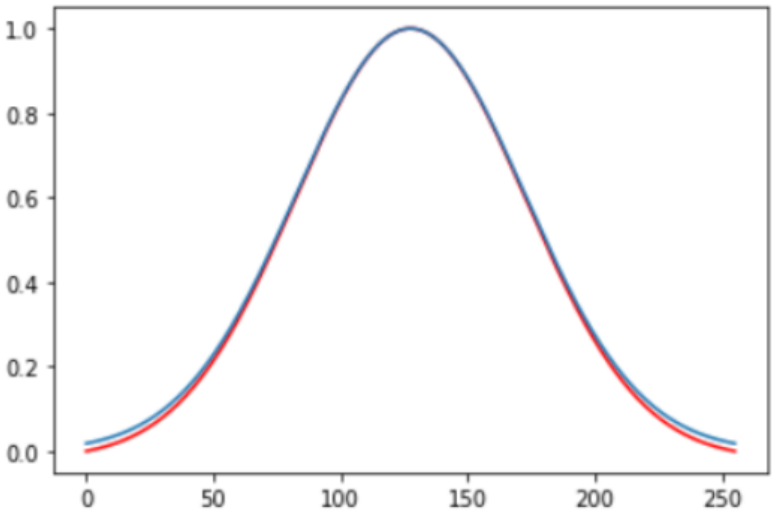


We eventually found that OpenCV is using another weight function. It looks slightly different from the weight function on paper, but make big difference in the final performance.



HDR by Paper's weight.

HDR by OpenCV's weight.



(Blue: paper's weight, Red: OpenCV's)

So we finally decide to use OpenCV's weight to generate our final result.

OpenCV: https://github.com/opencv/opencv/blob/4.x/modules/photo/src/hdr_common.cpp.

Optimization

$$E_i = \frac{\sum_j w(Z_{ij}) g(Z_{ij}) \Delta t_j}{\sum_j w(Z_{ij}) \Delta t_j^2}$$

$$g(m) = \frac{1}{|E_m|} \sum_{ij \in E_m} E_i \Delta t_j$$

These are just closed-form solution. Using for loop to calculate them can be very slow. So we implement them with Numpy (np.add.at can handle this).

Experiment of algorithm

These are some experiment method, not necessarily improve the performance.

1. Smoothing g using sliding window after each iteration.
    a. (G can be smooth enough with better weight function above)
2. Update G,E by $G_{i+1} = tG_i + (1-t)G'_{i+1}, G'_{i+1}$ is the closed form solution.
    a. No big difference.

3. Tone mapping

Radiance map for RAW

Besides Robertson's method, we can also obtain radiance map with RAW image. (our camera .CR2)

We use rawpy to process the RAW image and combine them with average of $E = \frac{X}{t}$ from every RAW.

Algorithms

We Did not Implement tone mapping ourselves.

But we tried several algorithms.

1. Reinhard : In OpenCV, kind of global tone mapping.
2. Mantiuk : In OpenCV, gradient based method.
3. Reinhard with Local operators:

a. Modified from the code on Github

Result

The local tone map method out perform all global method we tried.

Reingard + global

Reinhard + local

Bonus

1. MTB
2. Robertson