HDR Imaging

2017 NTU DigiVFX　Project #1

B03901154 電機三 曾耕森
B03901163 電機三 鄭　煦

Project Overview

This project requires us to rebuild high dynamic images from a serial of photos with different exposures. This contains several techniques, including image alignment, radiance map recovering, tone mapping, etc. The procedure of HDR imaging is shown below.

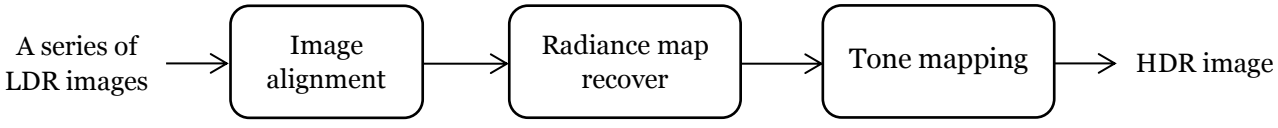


Fig.1　General procedure of high dynamic range imaging

We implemented our HDR processing program in MATLAB. The program contains radiance map rebuilding, which is the project requirement, and two different tone mapping algorithm. Implementation details are shown in section 2. The results and comparison of different algorithms are shown in section 3. Finally, the guide to our program is shown in section 4.

Implementation Details

1. HDR radiance map rebuilding

For radiance map rebuilding, we implemented Debevec's method [1] with different value of λ. The observed value of the i-th pixel in the j-th image is denoted as Z_{ij}.

$$Z_{ij} = f(E_i \times \Delta t_j)$$

where E_i is the radiance at the i-th pixel, Δt_j is the exposure time of the j-th image, and f denotes the camera response function. Let g denote $ln(f^{-1})$ and thus $g(Z_{ij}) = ln(E_i) + ln(\Delta t_j)$. The goal is to find g. The detail of the solution can refer to [1]. Once we obtain g, we can construct the HDR radiance map by

$$ln(E_i) = \frac{\sum_{j=1}^{P} w(Z_{ij})\,(g(Z_{ij}) - ln(\Delta t_j))}{\sum_{j=1}^{P} w(Z_{ij})}$$

The reconstructed camera response curve for RGB channels are shown in figure.

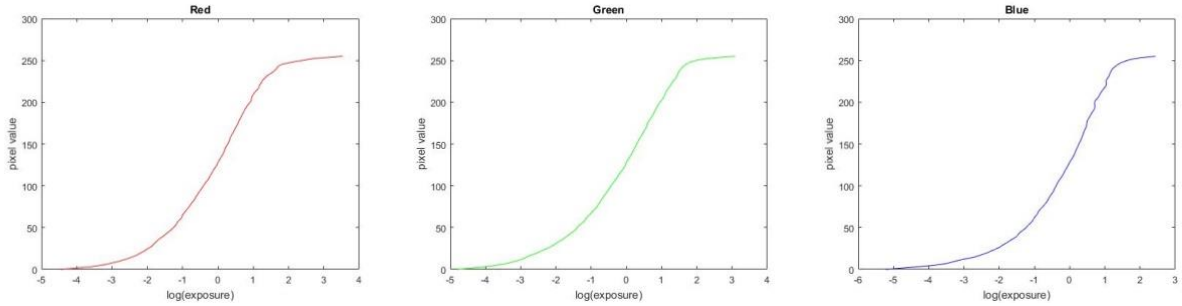


Fig. 2 Reconstructed camera response curve

During the sampling process for *gsolve* function, instead of selecting pixels manually, we choose to select pixels randomly. In fact, the result is also good enough. We choose 400 pixels as sample points, which is big enough to cover most of the level of value. Using more pixels also increases the computation time. There is a trade-off between number of pixels selected and runtime. In addition, choosing different λ significantly affects the camera response curve. We have tried $\lambda = 1, 5, 10, 20$, respectively. The result of setting $\lambda = 1$ wasn't good and jitter can be observed on the curve. We assume that the camera response curve should be smooth. Therefore we choose $\lambda = 10$ in following experiments.

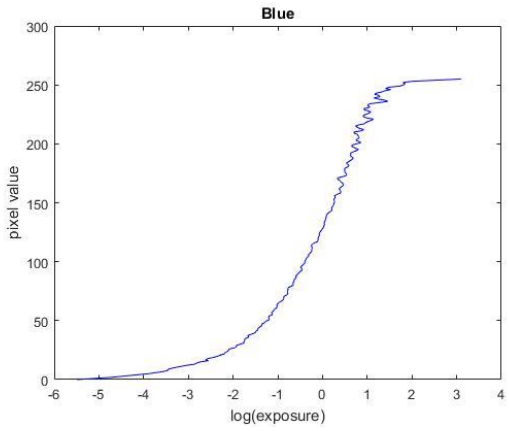


Fig. 3 Reconstructed curve for $\lambda = 1$

2. Tone mapping (Bonus)

We implemented two different tone mapping operator. The algorithms are described below.

2.1 Global operator [2]

This tone mapping algorithm was introduced by Reinhard et al. The idea of this method came from photography, which shared the same problem of dealing with high dynamic range scenes. The algorithm is simple and quick, while producing good results on a variety of images.

The pseudo code is shown as follow:

function $GlobalOperator\ (image\ \boldsymbol{L_w}, key\ a)$

1 $N = NumOfPixel(\boldsymbol{L_w})$
2 $L_{white} = max(\boldsymbol{L})$
3 $\overline{L_w} = exp(sum(log(\boldsymbol{L_w})) / N)$
4 $\boldsymbol{L} = \boldsymbol{L_w} \times a / \overline{L_w}$
5 $\boldsymbol{L_d} = \boldsymbol{L} \times (1 + \boldsymbol{L} / L_{white}^2) \ / \ (1 + \boldsymbol{L})$
6 ***return*** $\boldsymbol{L_d}$

Fig. 4 Pseudo code of global operator algorithm

where $\boldsymbol{L_w}$ is the original luminance matrix of the HDR image, N is the total number of pixels in the image and a is the key value of the image which depends on the brightness of the scene. We first compute the log-average luminance $\overline{L_w}$ as an approximation to the key of the scene and use this value along with a to scale the luminance beforehand. Then we compress the dynamic range and yield $\boldsymbol{L_d}$. L_{white} is a parameter associated to the burn-out effect and we simply set it to the maximum value of luminance in the image. The multiplication and division here are all per-pixel operation.

The results and discussion are shown in section 3.

2.2 Bilateral filtering [3]

Bilateral filtering was developed by Tomasi and Manduchi as an alternative to anisotropic diffusion. In this application, bilateral filtering serves as an edge-preserving filter for large-scale features computation. The process starts with standard Gaussian filtering with kernel f. In addition to f, an extra weight function g in intensity domain is also considered. The function g serves as an edge-stopping measure and decreases weight of pixels with large intensity difference. In our implementation, we use Gaussian function as influence function g. We can thus

compute the output at pixel s by

$$J(s) = \frac{1}{k(s)} \sum_{p \in \Omega} f(p - s) g\big(I(p) - I(s)\big) I(p)$$

Where $k(s) = \sum_{p \in \Omega} f(p - s) g\big(I(p) - I(s)\big)$ is a normalization term.

Durand et al. aimed at reducing contrast while preserving details of images, which can be achieved by decomposing the image into multiplication of two layers. The base layer is computed by applying bilateral filtering to the original image, and the detail layer can be obtained simply by dividing the image by the base layer. Then we reduced the contrast in the base layer while preserving the detail layer. Finally we recompose the two layers and obtain the displayable image.

The major caveat of this algorithm is its slow speed due to per-pixel computation in bilateral filtering. For large size images, the computation time can be intolerably long. To deal with this, Durand et al. suggested some algorithm to speed up the process. They separated the dynamic range into several segments and use the same value of $I(s)$ for the pixels in the same segment. By setting this approximation, we can now compute bilateral filtering in each segment by using convolution, which can be accelerated by using FFT.

```
function PiecewiseBilateral (image I, function f, function g)
1   J = 0
2   num_segments = (max(I) − min(I)) / σ_r
3   for j = 0 to num_segments
4       i_j = min(I) + j × σ_r
5       G_j = g(I − i_j)
6       K_j = G_j ⊗ f
7       H_j = (G_j × I) ⊗ f
8       J_j = H_j / K_j
9       J = J + J_j × Interpolation(I, i_j)
10  return J
```

Fig. 5 Pseudo code of piecewise bilateral filtering algorithm

The operator $\otimes$ denotes convolution, while other operators are all per-pixel computation. σ_r is a parameter of influence function g which, in our case, equals to the standard deviation of g. The function $Interpolation(I, i_j)$ is a mask to J_j matrix which allows only the pixels in current segment to write into J.

At this point, we have obtained the base layer by applying bilateral filtering to the image. Next, we compress the contrast in base layer by applying the global operator mentioned above, and then recombine the base layer and the detail layer. The results are shown in section 3.

Results

- **Comparing to Matlab built-in makehdr()**

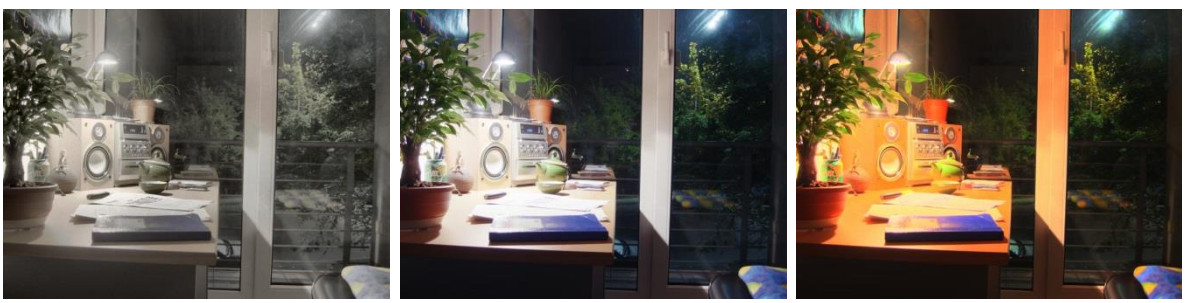
  The pictures shown below are images constructed using Debevec's method and Matlab built-in function `makehdr()`. The tone mapping method used are both Matlab built-in function `tonemap()`.

Our implementation Matlab built-in HDR

Fig. 6 Result images using Debevec's method and Matlab built-in function `makehdr()`. Both images use Matlab built-in `tonemap()` as tone mapping operator.

We can see that Debevec's method has better performance because the former has higher saturation, making it closer to ground truth.

- **Comparing different tone mapping operators**



Matlab tonemap() Global operator Bilateral filtering

Fig. 7 Comparison of different tone mapping algorithms.

We compare our two tone mapping operator to Matlab built-in function. As we can see from the figures, Matlab tonemap() do well on contrast reduction but wash away colors. Our implementation of global operator presents much more vivid colors but does less amount of contrast reduction. Bilateral filtering produces a little bit oversaturated images but performs better than global operator in reducing contrast.

- **Gallery**

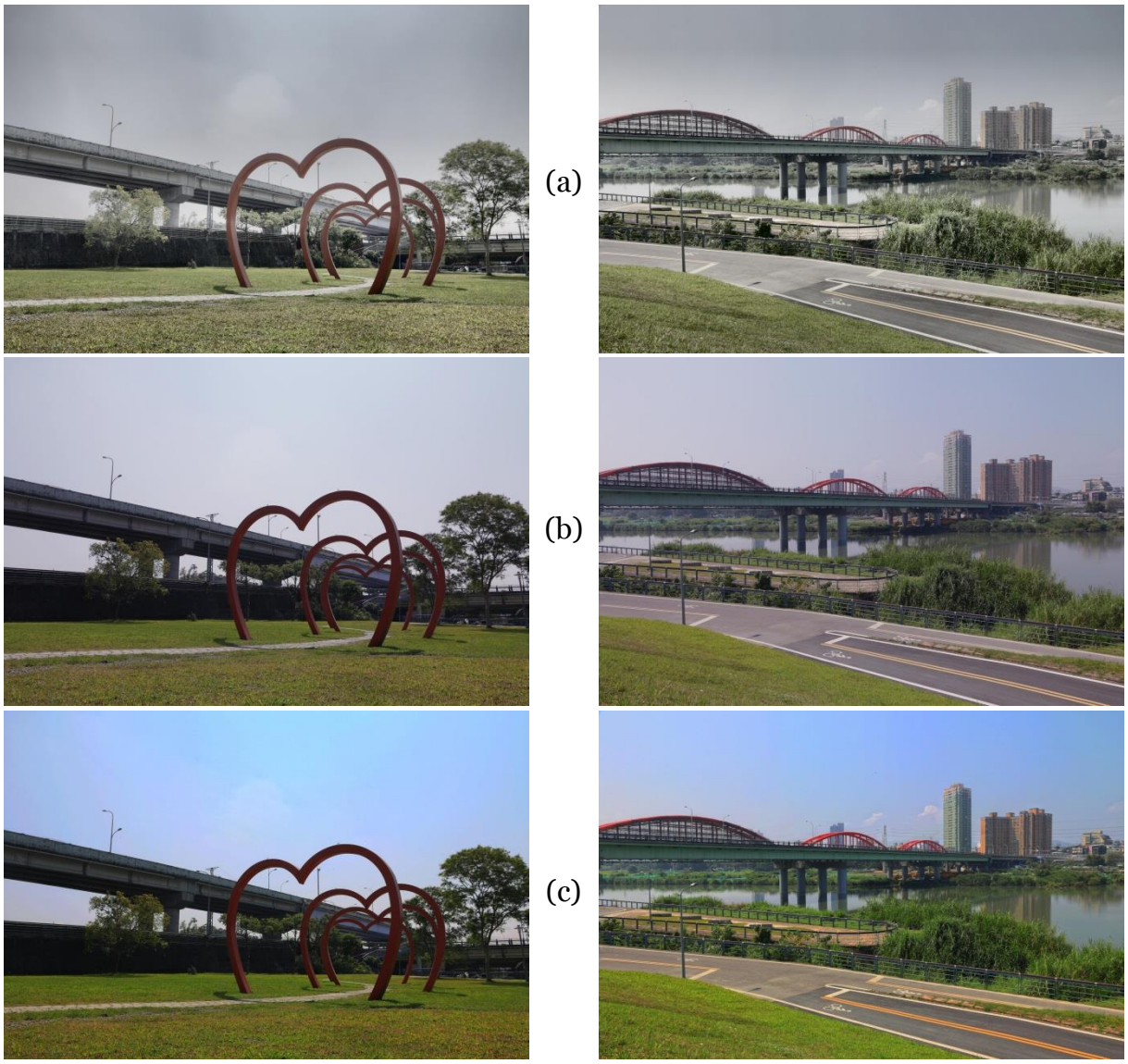


Fig. 8 Result images using Debevec's method as radiance map reproduction algorithm and three different tone mapping operators:
(a) Matlab tonemap() (b) Global operator (c) Bilateral filtering.

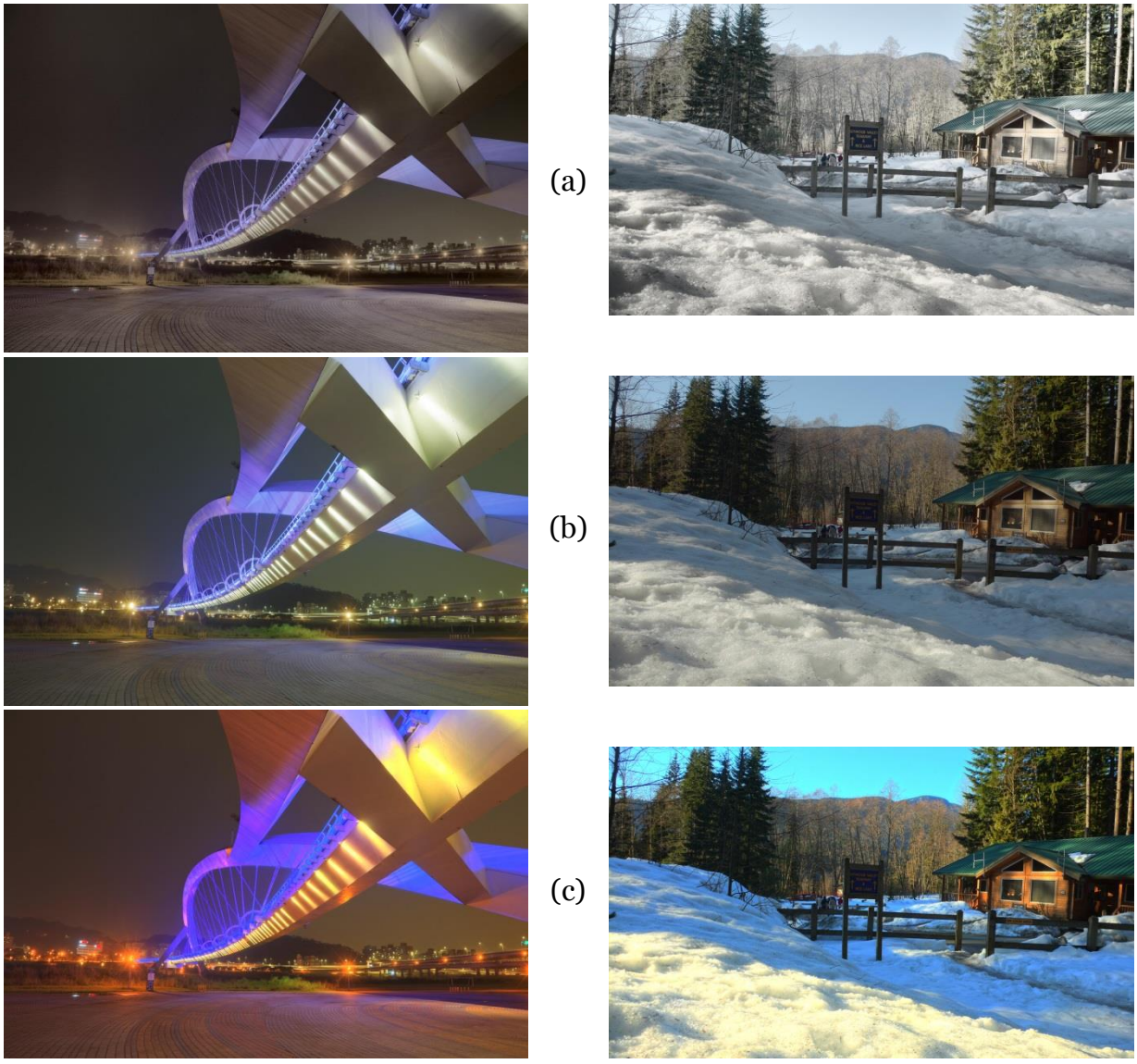


Fig. 9 Result images using Debevec's method as radiance map reproduction algorithm and three different tone mapping operators:
(a) Matlab tonemap() (b) Global operator (c) Bilateral filtering.
*The images on the right hand side is a sample hdr image downloaded from [4]

How to Run Our Program

Our program was written and run on Matlab v.8.6.0 (R2015b) with 64bit Windows 7. To run our program, open Matlab and change to the directory of our code (i.e. Resource/src). We describe the step-by-step procedure as follow:

- **HDR radiance map rebuild**

1. Specify the name of input images and exposure time in "info.txt" in src folder. Format: *image_name exposure_time*
   The images specified will be used to rebuild the radiance map.
2. Execute our HDR script file in Matlab command line.

```
My_hdr
```

3. The program will plot and show the response curves of R, G, and B channel respectively.
4. The result radiance map will be written to "Resource/result" folder and named "result.hdr".

- **Tone mapping**

1. Read in the radiance map in Matlab command line.

```
hdr = hdrread('your_hdr_image.hdr');
```

2. To use the global operator algorithm, type

```
rgb = My_tonemapping_global(hdr, key_value);
```

   The key value is a user-specified parameter and is typically around 0.5 to 1, depends on the brightness of your scene.
3. To use the bilateral filtering algorithm, type

```
rgb = My_tonemapping_ bilateral(hdr, key_value);
```

   Again, key value must be specified. The computation may take a while.
4. You can use `imshow(rgb)` to view the final result. If you want to write the image to file, use

```
imwrite(rgb, 'output_name.jpg');
```

Related Work

In this project, we also tried a method called **exposure fusion** [5] to construct HDR image. This method does not recover the camera response curve. Instead, it computes the best components of an image, and using a weight function to construct a HDR image. In other words, it measures contrast, saturation, and well-exposedness and linearly combines each pixel in the images. Then it applies Laplacian Pyramid and Gaussian Pyramid on each image. Gaussian Pyramid acts as a weight map for Laplacian Pyramid. Finally, we can fuse the modified Laplacian Pyramid as a weight map for each pixel to form a HDR image. Exposure fusion is more intuitive and has more natural look. However, it costs a great amount of memory usage. For example, our laptop is capable of running 16 images during Debevec's method, while it is only capable of running around 8 images using exposure fusion algorithm.

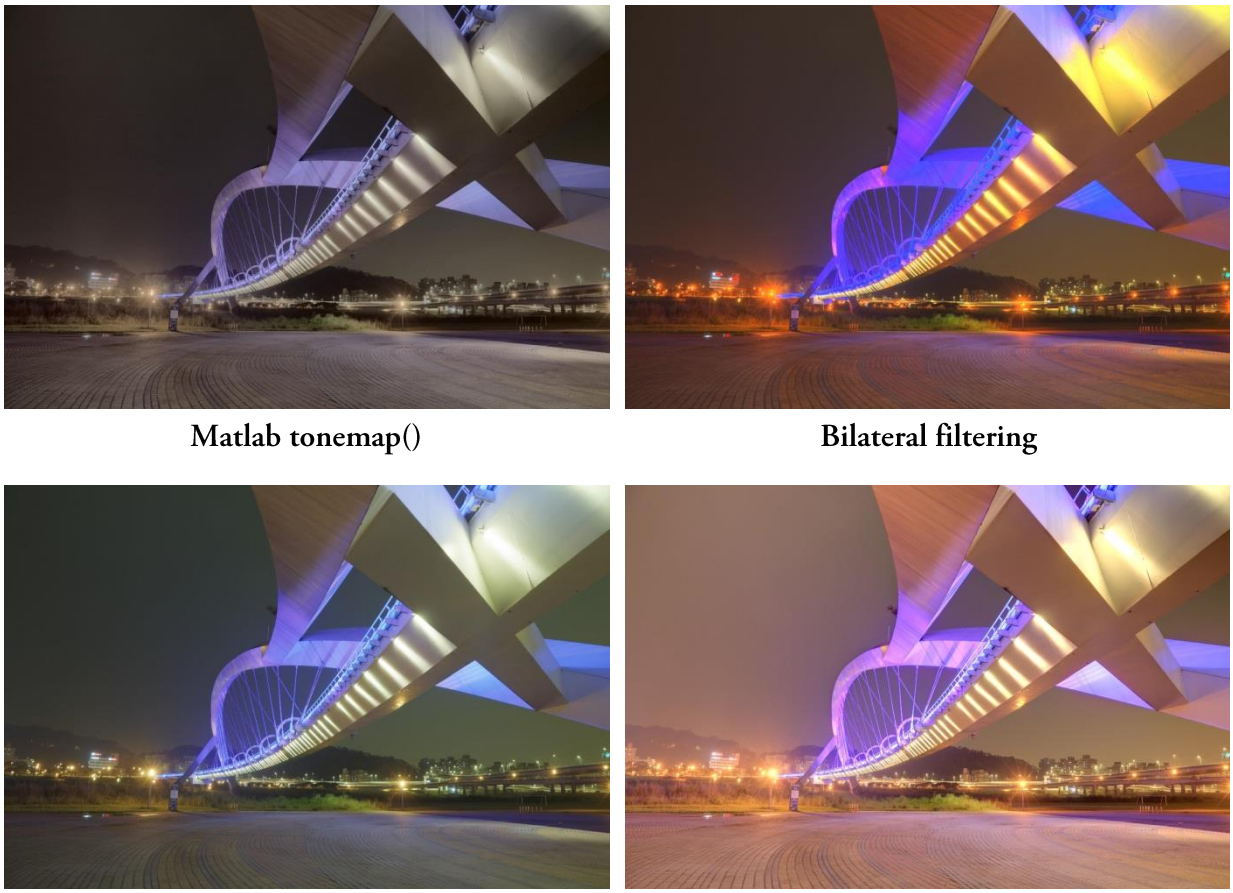


Fig. 10 The comparison between image constructed by exposure fusion method and previously mentioned HDR images.

References

[1] Paul E. Debevec, Jitendra Malik. *Recovering High Dynamic Range Radiance Maps from Photographs*, SIGGRAPH 1997.

[2] Erik Reinhard, Michael Stark, Peter Shirley, Jim Ferwerda, *Photographics Tone Reproduction for Digital Images*, SIGGRAPH 2002.

[3] Fredo Durand, Julie Dorsey, *Fast Bilateral Filtering for the Display of High Dynamic Range Images*, SIGGRAPH 2002.

[4] *HDR Image Gallery*, retrieved from http://pfstools.sourceforge.net/hdr_gallery.html

[5] Mertens, Tom, Jan Kautz, and Frank Van Reeth. *Exposure fusion: A simple and practical alternative to high dynamic range photography*. Computer Graphics Forum. Vol. 28. No. 1. Blackwell Publishing Ltd, 2009.