EE569: Homework #1

Yifan Wang
wang608@usc.edu #3038184983

January 20, 2019

Assumption and Info

1. All codes are written in Xcode version 10.1(10B61), expect part of shot noise is written in $Matlab$.
2. A submit version uses cmake. All functions can be used if you run $./run$ in bin through CLT and input correct parameters.
3. Figures in this pdf comes raw saved by my codes and exported to $jpeg$ by $PhotoShopCC$ 2017 embedded in $sRGB - IEC61966 - 2.1$ color space for color image and $Dot - Grain$ 15% for gray image.
4. Assume that left top corner of image is Green, others follow bayer pattern in $Homework1$ $Figure1(b)$
5. Red, green, blue are ordered in RGB in raw file my codes saved.
6. Assume height and width of an image are all even number.
7. $readraw.cpp$ provided by professor is modified into to two separate functions which can read and write images at any size. They are named $ReadRaw$ and $WriteRaw$ which locate in $Basic.hpp$.
9. Some figures related to analysis effect of core length or standard deviation are generated by $plot$ function in $Matlab$. They would be captioned. Otherwise, $jpeg$ come from raw files.
10. All details of codes or usage can be found in $readme.pdf$ in code folder which is appended at the end of this file, as well.

1 Problem #1: Image De-mosaicing

With a CFA input, special algorithms are needed to recover color image. This is a key part for cameras to produce color images. A little reeducation on image details would occur than using a monochrome camera which does not have a low pass filter or bayer filter in front of CMOS.

1.1 Methods

1.1.1 Bilinear De-mosaicing

By using definition of bilinear de-mosaicing and separate pixel of a image into 3 conditions, a most accurate form of bilinear de-mosaicing is produced:

1. 4 corner pixels;
2. other pixel on edges;
3. center pixels.

When coding, special care is needed for these 3 different conditions (realized in $fancy.hpp$). Idea from [1] is used to classify color of original pixel. Another function named $BiDemosaicingC$ in $ImgDemHistMupl.hpp$ which uses convolution is provided as well.

(a) CFA sensor input

(b) Original cat image

Figure 1

(a) Considered edges and corners

(b) Zero Padding

(c) Reflection Padding

Figure 2: Bilinear De-mosaicing

(a) MHCDemosaicingC using Zero Padding

(b) Reflection Padding

Figure 3: Malvar-He-Cutler De-mosaicing

Method	Padding	PSNR (dB)
BiDemosaicing	-	25.0043
	Zero	24.2267
	Reflection	23.8998
MHCDemosaicing	Zero	27.1833
	Reflection	27.9533

Table 1: PSNR of Different De-mosaicing Algorithm for $cat.raw$

1.1.2 Malvar-He-Cutler (MHC) De-mosaicing

Formulas to calculate result for MHC de-mosaicing come from lecture note and [2]. Convolution is used in realization. Both zero padding ($Figure3(a)$) and reflection padding ($Figure3(b)$) are available.

1.2 Evaluation and Improvements

$Figure1(b)$ shows original picture. PSNR is calculated in comparison with it. ($Table1$).

Three different versions of bilinear de-mosaicing ($Figure2$) are provided. Among these images, small difference can be observed especially on edges. $Figure2(a)$ provide best result, however it is difficult for coding since every special condition should be written out carefully. While other two approaches are much easier to realize and understand. Compared with original image, image from bilinear de-mosaicing processing has:

1. much less detail;
2. blurred edges;
3. color points or grids affect image quality;
4. color distortions, inconsistency and zipper effect.

These affects might caused by processing each color channel without considering information from others. From real life experience, it is a common sense that value of a color channel in a pixel is related to other channels and pixels around it. These information is thrown away in bilinear de-mosaicing result in less detail, color distortions and inconsistency.

Related color information (other channel's value at this pixel or enlarge range of pixels used in calculation) when calculating a single color channel of a pixel. That is what MHC de-mosaicing improved. It includes more information of other channels, leading to a much sharper image with more details and less color distortions. It can be proven from comparing PSNR or judging by human eyes.

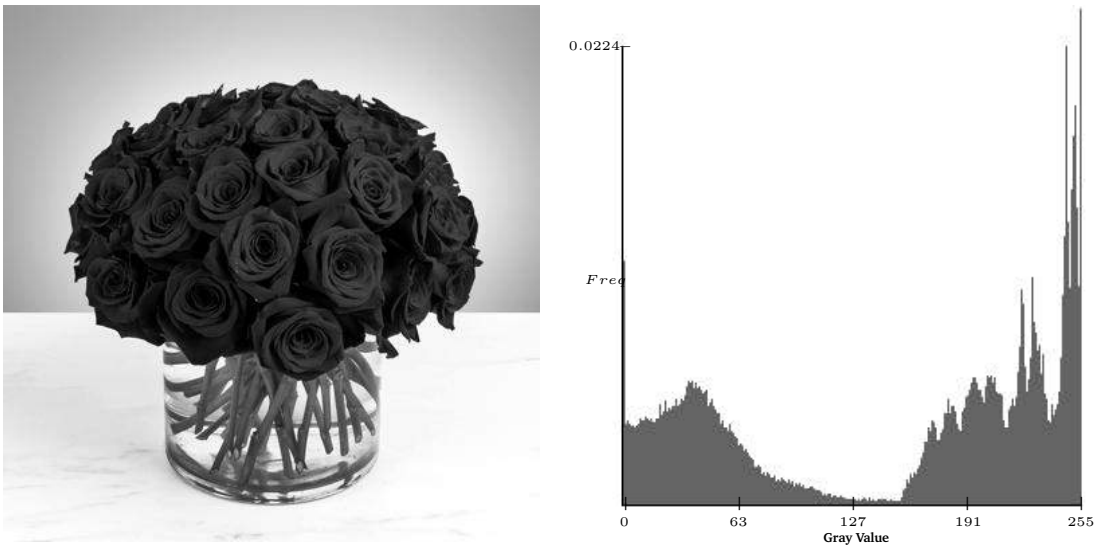
However, in $MHCDemosaicing$, some part of image, especially in edges of chair, more blue color distortion and colored jaggies occur. This may caused by the fact that sampling frequency of red and green in Bayer pattern is much lower than green which would make it much tougher to recover red and blue than green. More complex method or brand new image sensor can be possible solutions. For example $FoveonX3$ sensor from Sigma Cooperation [10], uses stack structure which can get blue, green and red information on a single pixel. In that case, no de-mosaicing is needed any more. It performs well on Sigma's camera.

Different padding methods have slightly different results, but they would never have same accuracy as $BiDemosaicing$ with no padding. While results for zero padding and reflection padding varies from images and de-mosaicing methods.

2 Problem #1: Histogram Manipulation

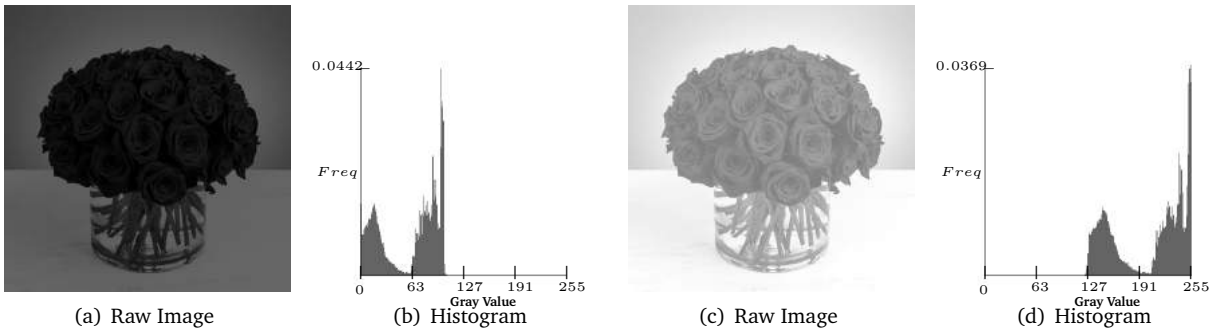
Histogram equalization is generally used to enhance contrast in some part of image while does not affect contrast of whole image. It enables brightness to separate more evenly on histogram. One advantage of this

type of method is reversibility only if function used is known.



(a) Raw Image (b) Histogram

Figure 4: rose_ori.raw



(a) Raw Image (b) Histogram (c) Raw Image (d) Histogram

Figure 5: left: rose_dark.raw, right: rose_bright.raw

2.1 Methods

2.1.1 Method A

A transfer function is used to make histogram become a uniform distribution.

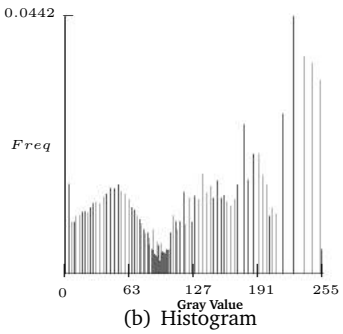
2.1.2 Method B

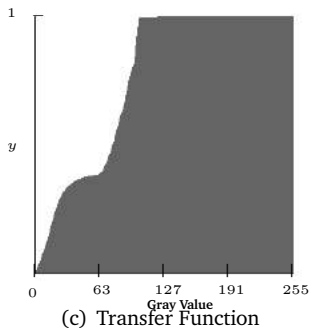
Cumulative histogram is used. $\frac{H*W}{256}$ pixels would be assigned on each $8bit$ gray value.

I use the strategy that line these pixels in a queue according to pixel value $(0-255)$ and location (from left to right, top to bottom), arrange them with new gray value to realize.

2.2 Evaluation and Improvements

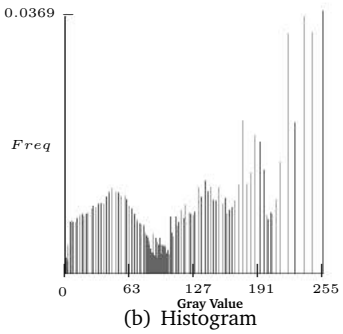
From enhanced images, I observe that both methods would decrease brilliance of white background and increase brilliance of flower. Contrast would be lower when image has pure white or black background. Edges

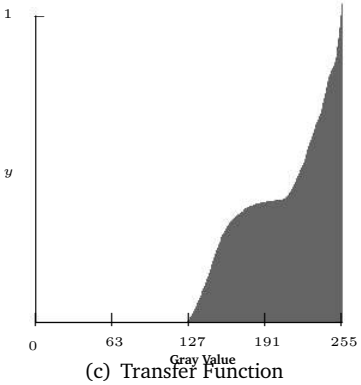




(a) Result (b) Histogram (c) Transfer Function

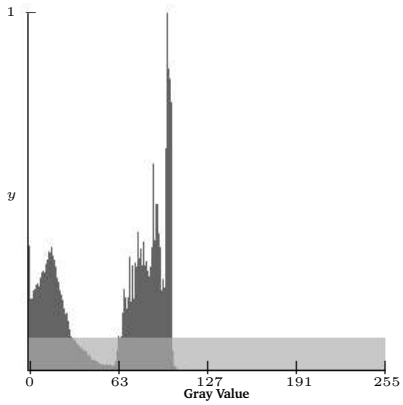
Figure 6: rose_dark.raw using Method A





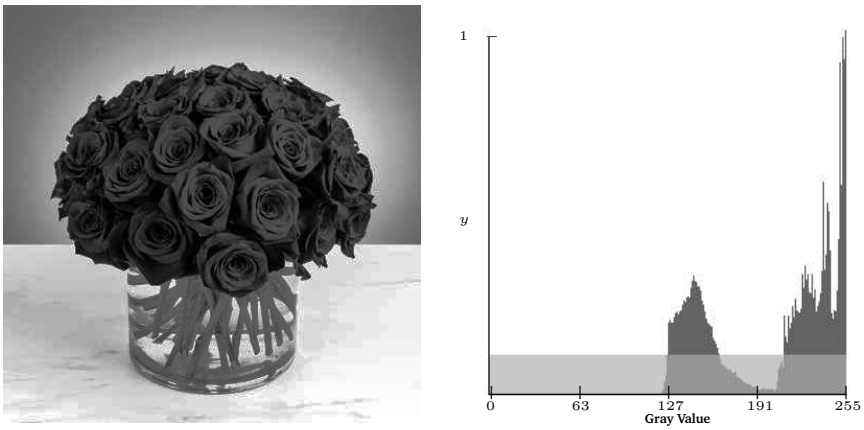
(a) Result (b) Histogram (c) Transfer Function

Figure 7: rose_bright.raw using Method A



(a) Result (b) Histogram before and after (light gray)

Figure 8: rose_dark.raw using Method B



(a) Result (b) Histogram before and after (light gray)

Figure 9: rose_bright.raw using Method B

Image	Method	PSNR (dB)	SSIM
rose_dark.raw	-	7.62971	0.0312068
	A	16.847	0.0190757
	B	16.6367	0.0192734
rose_bright.raw	-	11.3636	0.0347742
	A	16.8127	0.0190492
	B	16.6482	0.0192776
rose_mix.raw	-	22.894	0.0116489
	A	16.7624	0.0189844
	B	16.576	0.0192658

Table 2: PSNR and SSIM of Different Histogram Equalization Method

are not sharp enough. Some details are lost. Especially for stem parts of flowers, they are blurred a lot which make image seems unreal.

Besides, gray value does not change continuously, result in mosaic like patterns occur in image, which is obvious in background. Histogram shows this appearance more clearly that both method would have lots of jumps in gray value while original one looks more continuously (the word continuously means that if some pixels have gray value x, then there are pixels have gray value $x-1$ or $x+1$.

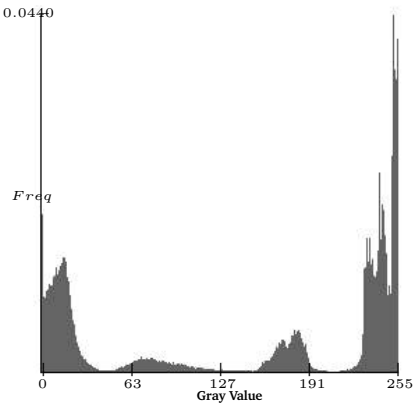
$MethodB$ perform a little worse than $MethodA$ in background (more mosaic like patterns). What's more $MethodB$'s background looks darker. $PSNR$ and $SSIM$ tells difference as well ($Table2$).

When considering $rose\_mix.raw$, its histogram has both extremely dark and extremely bright part, in other word it has super high contrast. Both methods produce almost the same result as previous one with slightly difference on edges. In original image background has sharp change from white to gray which cause light ring around edges. $MethodA$ performs a little better than $MethodB$ in preserving structures and details.

For $MethodA$, to make it performs better, bilinear interpolation between pixels to fill in gaps between two pixel which have large difference in gray value can be used. Might result in a smoother change in gray value.

For $MethodB$, to make it performs better, a threshold can be used. If original gray value at a certain pixel is g, and it should be given new gray value g_{new} according to $MethodB$. If $|g-g_{new}|$ is larger than threshold, another gray value g_{alt} should be used instead of g_{new} to change it (information of it's neighbourhood can be considered as well). It is possible to find a function Number of pixel in gray value $N_x = f(x)$. Rather than distribute pixels evenly to gray value which results a linear cumulative histogram after processing, as well. This can be interrupted as adaptive adjust gray value.

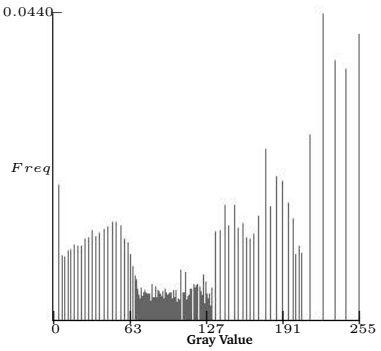
(a) Raw Image



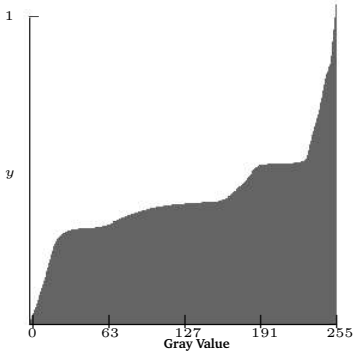
(b) Histogram

Figure 10: rose_mix.raw

(a) Result



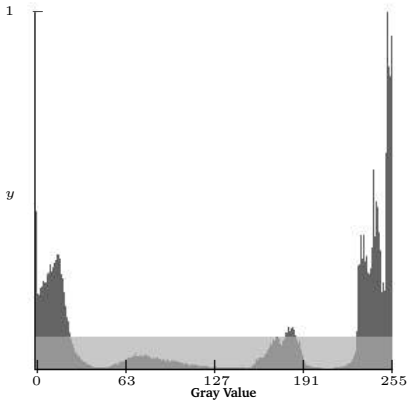
(b) Histogram



(c) Transfer Function

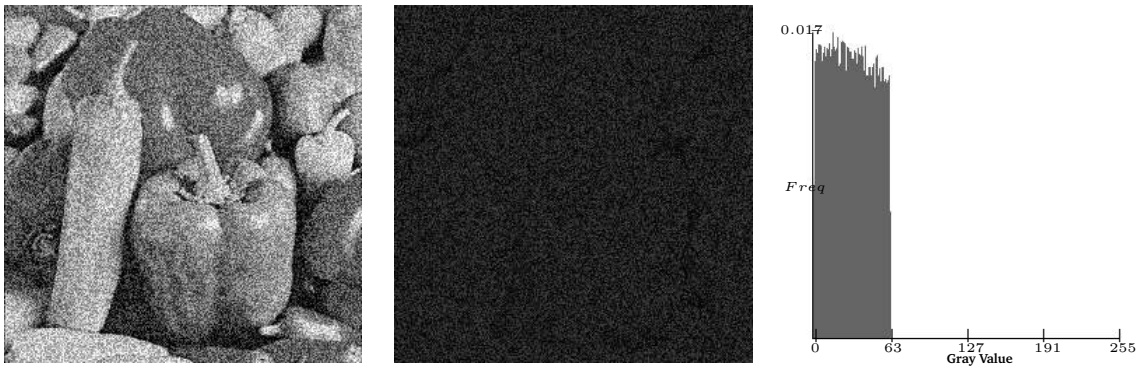
Figure 11: rose_mix.raw using Method A

(a) Result



(b) Histogram before and after (light gray)

Figure 12: rose_mix.raw using Method B



(a) Noise Image (b) Noise (c) Histogram of (b)

Figure 13: pepper_uni.raw
$PSNR = 16.92dB$

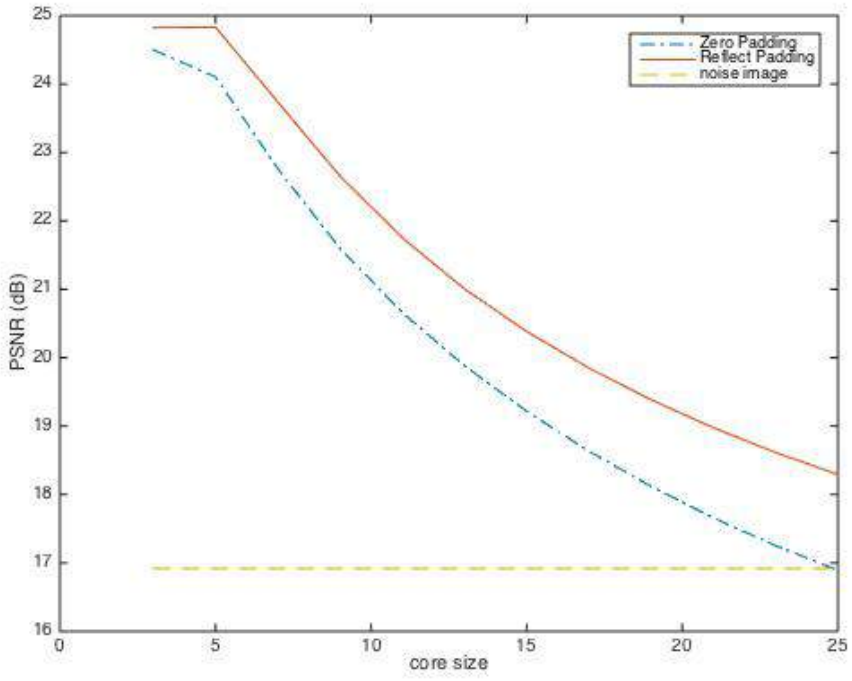


Figure 14: Comparison between Uniform Filter using Zero Padding, Reflect Padding and different core size on pepper_uni.raw
$Matlab$ is used

3 Problem #2: Image De-noising

Many steps in generating an image could cause noise, for example over-heated CMOS sensor, high ISO, noise in signal amplifier, bit error etc. For better image quality, either technologies like Frozen CCD, larger pixel size on sensor etc. are used or de-noising methods are needed. So far most de-noising methods are about averaging, if lots of same image are observed, averaging can be done among these images (a well- known example is stack in PhotoShop, especially useful for sky photography), otherwise only averaging among pixels from one image would blur details.

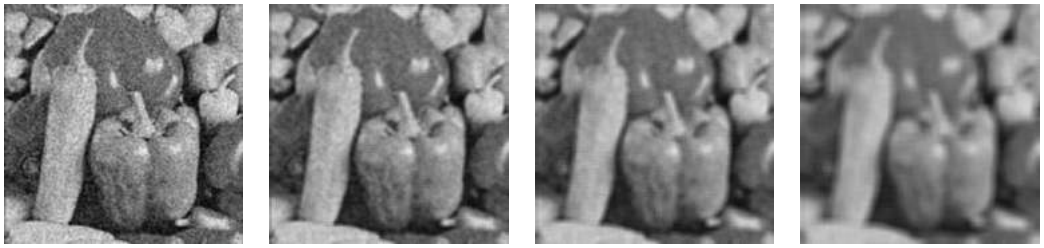
Info:

Default method is reflection padding for Uniform and Gaussian Filter, uncomment code for using zero padding. Other filters using reflection padding method only.

Function for saving histogram is modified for better visualization on color or noise image. Histograms in $Problem\#2$ start to use a new version. Difference is described in $readme.pdf$.

It is uniform noise embedded in $pepper\_uni.raw$. Conclusion can be get from histogram of noise. Since $pepper.raw$ is given as a reference no noise image, I use $|P_1(i,j) - P_2(i,j)|$ to get noise ($Figure13(b)$) and its histogram ($Figure13(c)$). From its histogram, I can conjecture formula to generate a noise is likely to be:

$$64 * U(0,1) \tag{1}$$



(a) $L = 3$, $24.8249dB$ (b) $L = 5$, $24.8298dB$ (c) $L = 7$, $23.738dB$ (d) $L = 11$, $21.7511dB$

Figure 15: Uniform Filtering

3.1 Methods

Several general ded-noising filters are implemented in this part.

3.1.1 Uniform Filtering

For a $N * N$ uniform core, it has all ones inside with a normalization factor $\frac{1}{N*N}$. Comparison is made on effect of different core size and different padding methods ($Figure14$). For a uniform filter, it is better to have a small core. It can be explained that larger core would average more information of edges and corners and produce a blurrer image.

3.1.2 Gaussian Filtering

For a Gaussian Filter, structure is the same as Uniform Filter, expect weights in core follows gaussian distribution with parameter of Euclidean distance between two pixels. Both core length and standard deviation σ would affect result, a general trend is larger core length or larger σ would result in an image with far less details. Some of results are shown in $Figure17$. $Figure16$ shows change of PSNR in different parameters.

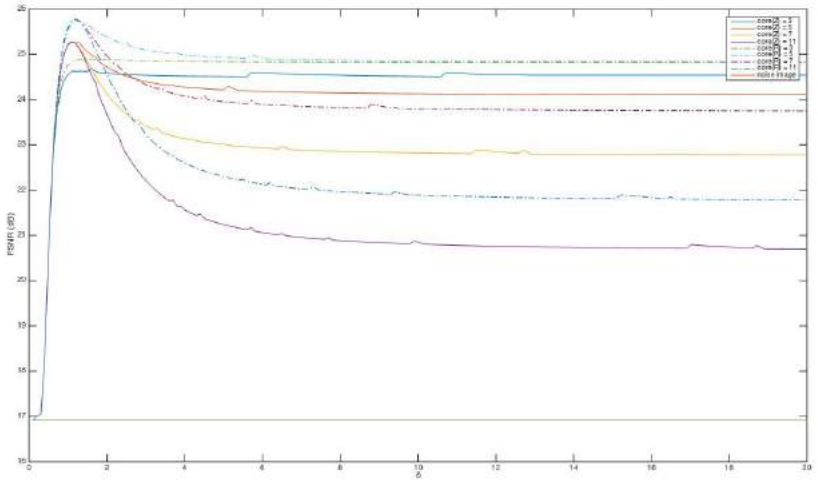
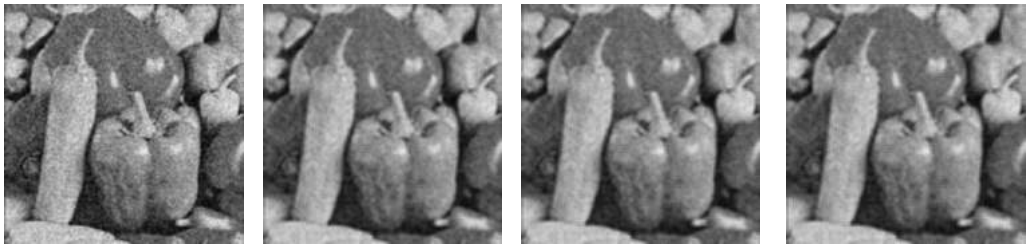
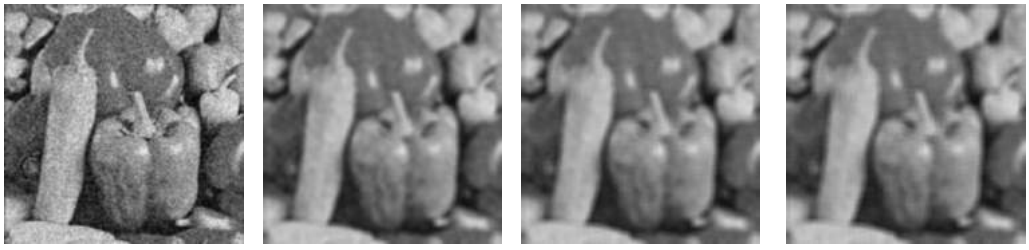


Figure 16: Gaussian Filter on pepper_uni.raw with different core size and σ
coreZ: core size using zero padding; coreR: core size using reflection padding;
$Matlab$ is used

(a) $\sigma = 1$, $24.8346dB$ (b) $\sigma = 3$, $24.8537dB$ (c) $\sigma = 5$, $24.8357dB$ (d) $\sigma = 9$, $24.8274dB$



(e) $\sigma = 1$, $25.5573dB$ (f) $\sigma = 3$, $25.0998dB$ (g) $\sigma = 5$, $24.9311dB$ (h) $\sigma = 9$, $24.8614dB$



(i) $\sigma = 1$, $25.5986dB$ (j) $\sigma = 3$, $23.6568dB$ (k) $\sigma = 5$, $23.0253dB$ (l) $\sigma = 9$, $22.7688dB$

Figure 17: Gaussian Filtering $(a) - (d)$ L = 3, $(e) - (h)$ L = 5, $(i) - (l)$ L = 9

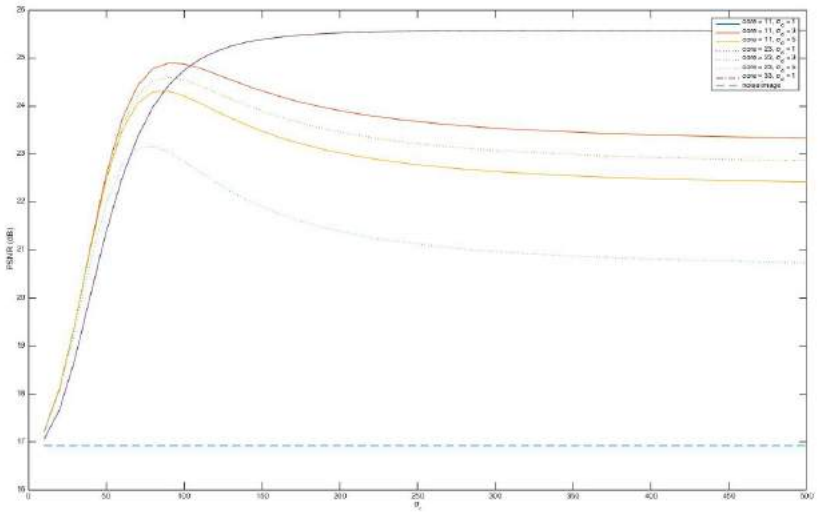
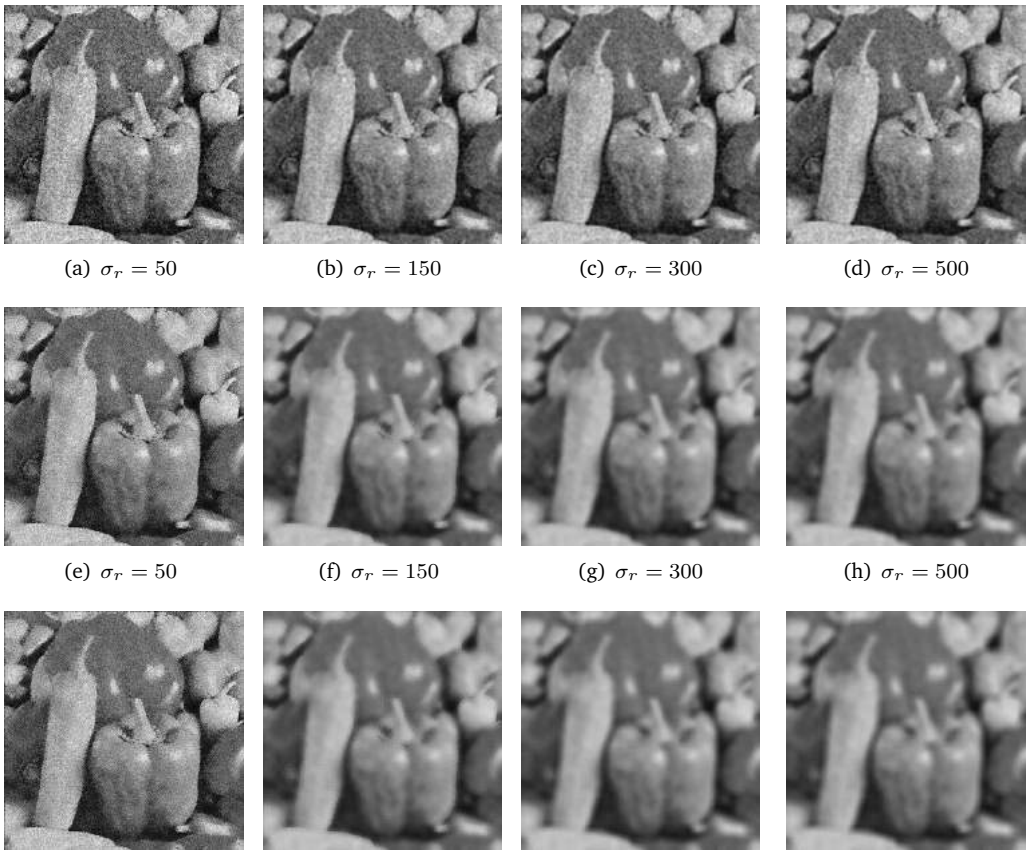


Figure 18: Bilateral Filter on pepper_uni.raw with $core = 11$ and $core = 23$ and different σ_d and σ_r $Matlab$ is used



(a) $\sigma_r = 50$ (b) $\sigma_r = 150$ (c) $\sigma_r = 300$ (d) $\sigma_r = 500$

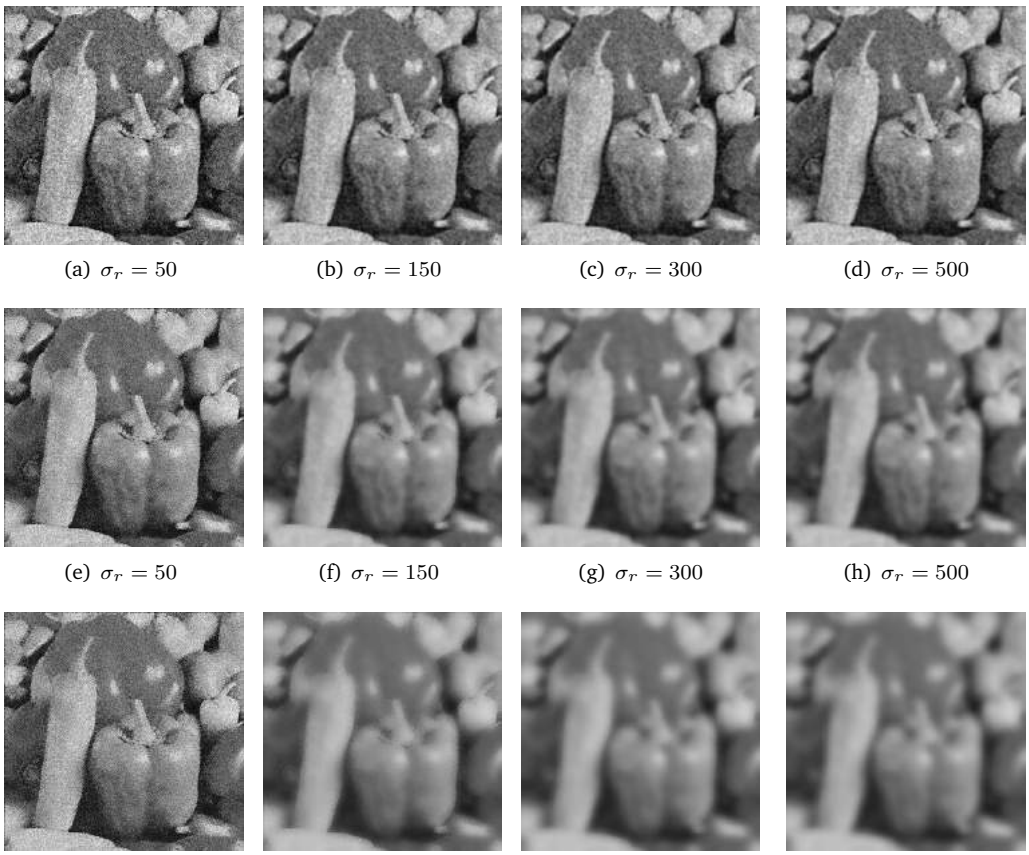
(e) $\sigma_r = 50$ (f) $\sigma_r = 150$ (g) $\sigma_r = 300$ (h) $\sigma_r = 500$

(i) $\sigma_r = 50$ (j) $\sigma_r = 150$ (k) $\sigma_r = 300$ (l) $\sigma_r = 500$

Figure 19: Bilateral Filter $Core = 11$, $(a) - (d)$ $\sigma_d = 1$, $(e) - (h)$ $\sigma_d = 3$, $(i) - (l)$ $\sigma_d = 5$

3.1.3 Bilateral Filtering

This filter comes from homework notes and [3] in which propose to combine both closeness in distance (domain filtering) and similarity in pixel value (range filtering) together to filter image, instead of using domain filtering individually like Gaussian Filter.



(a) $\sigma_r = 50$ (b) $\sigma_r = 150$ (c) $\sigma_r = 300$ (d) $\sigma_r = 500$

(e) $\sigma_r = 50$ (f) $\sigma_r = 150$ (g) $\sigma_r = 300$ (h) $\sigma_r = 500$

(i) $\sigma_r = 50$ (j) $\sigma_r = 150$ (k) $\sigma_r = 300$ (l) $\sigma_r = 500$

Figure 20: Bilateral Filter $Core = 23$, $(a) - (d)$ $\sigma_d = 1$, $(e) - (h)$ $\sigma_d = 3$, $(i) - (l)$ $\sigma_d = 5$

The formulas I used have slightly different from homework. Domain filtering is a normal Gaussian core ($size = L * L$) related to Euclidean distance between two pixels in location (i, j) and (ii, jj), closeness core G is calculated:

$$G(i, j, ii, jj) = e^{-d^2} \tag{2}$$

where

$$d(i, j, ii, jj)^2 = (i - ii)^2 + (j - jj)^2 \tag{3}$$

Where $2 * \sigma_d^2$ can be omitted, since normalization is needed for the core, σ_r alone is enough. Coefficients on a fixed length core remains the same in on every pixel in image.

Similarly, for similarity core S ($size = L * L$) on a gray scale image for pixels in location (i, j) and (ii, jj), assume gray value range: $(0 - 255)$:

$$S(i, j, ii, jj) = e^{-\frac{sd^2}{\sigma_r^2}} \tag{4}$$

where

$$sd^2 = (P_g(i, j) - P_g(ii, jj))^2 \tag{5}$$

Where σ_r is standard deviation for range filtering. Core would change according different center pixel.

For color image ($8bit$ for each channel), a general idea to calculate Euclidean distance between two pixel value is:

$$sd(i,j,ii,jj)^2 = (P_{red}(i,j) - P_{red}(ii,jj))^2 + (P_{green}(i,j) - P_{green}(ii,jj))^2 + (P_{blue}(i,j) - P_{blue}(ii,jj))^2 \quad (6)$$

While from [4] [5] I found it would be better to use a weighted version:

$$sd(i,j,ii,jj)^2 = (2 + \frac{r}{256}) * sd_{red}^2 + 4 * sd_{green}^2 + (2 + \frac{255 - r}{256}) * sd_{blue}^2 \quad (7)$$

where

$$sd_{red} = (P_{red}(i,j) - P_{red}(ii,jj)) \quad (8)$$

$$sd_{green} = (P_{green}(i,j) - P_{green}(ii,jj)) \quad (9)$$

$$sd_{blue} = (P_{blue}(i,j) - P_{blue}(ii,jj)) \quad (10)$$

$$r = \frac{P_{red}(i,j) + P_{red}(ii,jj)}{2} \quad (11)$$

Weight is designed for human perception. My Bilateral Filter function uses equation (7) rather than (6). After both closeness core G and similarity core S are observed, new pixel value P_{new} at location (i,j) can be calculated:

$$P_{new}(i,j) = \frac{1}{N} * \sum_{P(ii,jj) \in A} C(ii,jj) * S(ii,jj) * P(ii,jj) \quad (12)$$

where A refers to $L * L$ core center in (i,j). N is normalize factor:

$$N = \sum_{P(ii,jj) \in A} G(ii,jj) * S(ii,jj) \quad (13)$$

3.1.4 NL-Means Filtering

(a) pepper_uni.raw (b) rose_color_noise.raw

Figure 21: Result of NL-Means Filter with uniform weight

Basic version of a non-local mean filter use formula provided in homework notes. Only one core size parameter is included and search from entire image. However, time complexity is $O(n^4)$, a long time is required for computation. In that case, I used a $W_s * W_s$ search window and $W_c * W_c$ compare window ([6], [7], [8]) to realize. Compare window has a flavor of uniform filter, that is how it is realized.

(a) pepper_uni.raw (b) rose_color_noise.raw

Figure 22: Result of NL-Means Filter with gaussian weight

Calculation of similarity distance remains the same as Bilateral Filtering in previous section $Equation(7)$. New pixel value at location (i, j) is calculated as follow:

$$P_{new}(i,j) = \frac{1}{N} * \sum_{ii=0}^{W_s} \sum_{jj=0}^{W_s} s(i,j,ii,jj) * g(i,j,ii,jj) * P(ii,jj) \tag{14}$$

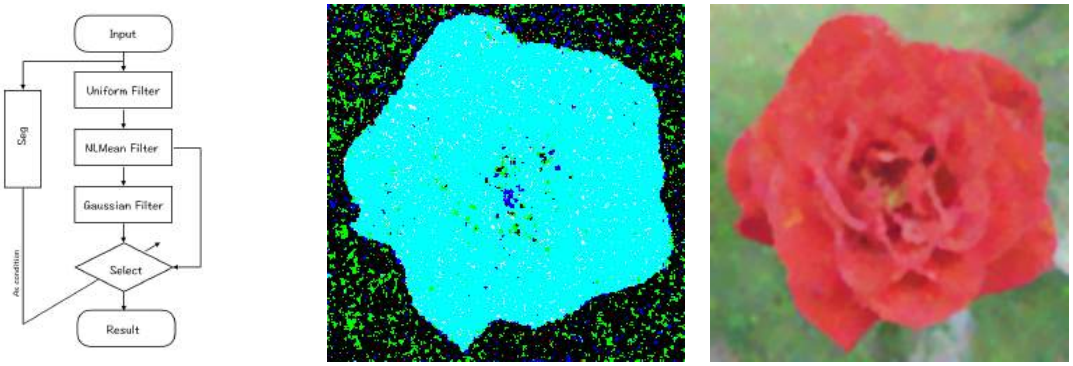
$$N = \sum_{ii=0}^{W_s} \sum_{jj=0}^{W_s} s(i,j,ii,jj) * g(i,j,ii,jj) \tag{15}$$

serves as a normalize factor.

$$s(i,j,ii,jj) = e^{-\frac{(\mu(P(i,j)|(i,j)\in R(i,j)) - \mu(P(ii,jj)|(ii,jj)\in R(ii,jj)))^2}{\sigma^2}} \tag{16}$$

$R(i, j)$ refers $W_c * W_c$ size region center in (i,j);

A small improvement is made on $\mu(P(i, j)|(i, j) \in R(i, j))$ in $Equation16$. In basic version, mean operation is non-weighted, while gaussian weight is used to get a weighted mean for improved version. Result a slightly improvement ($Table3$ NL Mean FilterG).



(a) Filter combinations used (b) Segmentation result (c) De-noising result $PSNR = 22.7512$

Figure 23: Filter Combination

Method	pepper_uni.raw	rose_color_noise.raw
-	16.92	11.7303
Uniform Filter	24.8298	21.9711
Gaussian Filter	25.7762	22.1612
Bilateral Filter	25.9862	22.2806
NL Mean Filter	27.1214	22.6845
NL Mean FilterG	27.3396	22.6854

Table 3: PSNR of Different De-nosing Algorithm

3.2 Result

3.2.1 Gray Image

From [11], I notice that uniform noise is named as quantization noise. It is generally a result from converting analog data into digital data and follows uniform distribution.

$Table3$ shows different result on pepper image with uniform noise. More complex filters can generate better de-noising result.

Both impulse and uniform noise are embedding in rose image, so that it would be difficult to deal with in a traditional way. For example, median filtering (realized in $fancy.hpp$) generally performs well against impulse noise, However with the existence of uniform noise, result is not pleased. So that a pipeline of filters to get best de-noising performance is proposed.

It is better to filter on all channels rather than filter each channel independently. Generally, color has correlations with others which is the same as de-mosaicing. While noise on each channel might not be related. With this difference, better de-noising result might be achieved.

Due to the fact when removing uniform noise, most filters tends to average image and blur edges. To preserve edges, I suggest following attempts:

1. Combine some edge detection methods (sobel, prewit etc., or even using methods have a flavor of ORB or SIFT to detect corners and edges), after filtering, the parts where edges are detected can be enhanced;
2. Or filtering happens only on none edge parts, and do nothing or filter less on edges parts.
3. Anti noise method is another possible way. If a good loss function can be designed, an anti noise can be generated which can cancel noise in image (a flavor of active noise control (ANC) in acoustic). De-noising process can be regarded as an optimization problem in this view.
4. Saliency detection [12] or image segmentation methods can be used on noisy image to separate background and foreground, or even separate image into more parts. Appling different filters with various strength on each part of image. Finally, join these image parts together. This method is useful especially when image has large simple backgrounds with huge noises or they are already blurred because of shallow depth of focus when taking the photos.

I used used a pipeline of Uniform Filter, NLMeans Filter and Gaussian Filter to de-noise color rose image, with a fancy structure in $Figure23(a)$

3.2.2 Color Image

Instead of just using a single pipeline, some fancy ideas from image segmentation are used. Foreground and background can be separated using some methods ($fancy.hpp$). I use this image as a reference to select which pixel to choose.

While in main pipeline, a uniform filter with a small core is used to average noise especially for averaging impulse noise, the reason why not use a large core is I want to preserve more details. A NL Means Filter is used and result would be saved (let's call it $ImgB$) before it was filter by another uniform filter and get $ImgU$.

Select method is used in this step to decide whether pixel at location (i, j) belongs to foreground or background. If it belongs to foreground, use color information from $ImgB(i, j)$. Otherwise, use $ImgU(i, j)$ for result. Segmentation result and de-noising result are show in $Figure23(b), (c)$. A better de-noising result is achieved for $PSNR = 22.7512$.

3.3 Shot Noise

$BM3D.m$ in $Matlab$ from [13] for de-noising is used in this part, my function is saved in $DeShotNoise.m$. With anscombe transformation from [14] to transfer image to follow gaussian distribution and using BM3D function in $BM3D.m$ to de-noising. Transfer it back by using unbiased anscombe transformation ($Figure24(d)$). To maintain consistence of evaluation, de-noising result is saved in raw and evaluated by my $C++$ code as well.

A similar function using gaussian filter to de-noising is implemented in $C++$ using gaussian filter realized by myself ($Figure24(c)$).

It is obvious $BM3D$ achieves a far better de-noiseing performance. But to notice that for using $BM3D$ a further linear transformation is needed to the result of anscombe transformation to $(0, 1)$.

(a) (b) $PSNR = 30.2103$, $SSIM = 0.074872$

(c) $PSNR = 35.5763$, $SSIM = 0.0835131$ (d) $PSNR = 39.2808$, $SSIM = 0.0827526$

Figure 24: (a): noise free image, (b): noise image, (c): using Gaussian Filter, (d): using BM3D

More Ideas

For NL Means filter and BM3D filter, when considering similar fragments, normal way is to compute the L_P norm among the pixel value. I wander if more structure information can be utilized when the two area have different brilliance but similar pattern, for example if there is an image of a fence which are partly shaded by trees. In that case, even though the structure of fence is the same in both bright and shaded area, they would be regard as dissimilar. A demean method can be applied to get fragments with similar structure regardless of their large difference in luminance.

To make it further, for each fragment a direction descriptor should be computed to group the similar patterns with rotation difference, a favor similar to BRIEF descriptor in SIFT or ORB corner detection.

What's more, non-maximum suppression can be applied when there are several fragments overlapping. By using this choose one fragment with greatest similarity, and discard the rest.

References

[1] Zhen R , Stevenson R L . Image Demosaicing[J]. 2015. pp. 14–15.

[2] Getreuer P . Malvar-He-Cutler Linear Image Demosaicking[J]. Image Processing on Line, 2011, 1.

[3] R. Manduchi and C. Tomasi, "Bilateral Filtering for Gray and Color Images," Computer Vision, IEEE International Conference on(ICCV), Bombay, India, 1998, pp. 839. doi:10.1109/ICCV.1998.710815

[4] [Online]`https://en.wikipedia.org/wiki/Color_difference`

[5] [Online]`https://www.compuphase.com/cmetric.htm`

[6] Buades A , Coll B , Morel J M . A non-local algorithm for image denoising[C] Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. IEEE, 2005.

[7] Dabov K , Foi A , Katkovnik V , et al. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering[J]. IEEE Transactions on Image Processing, 2007, 16(8):2080-2095.

[8] Vignesh R , Oh B T , Kuo C C J . Fast Non-Local Means (NLM) Computation With Probabilistic Early Termination[J]. IEEE Signal Processing Letters, 2010, 17(3):277-280.

[9] M.MahmoudiandG.Sapiro,"Fastimageandvideodenoisingvianon- local means of similar neighborhoods," IEEE Signal Process. Lett., vol. 12, no. 12, pp. 839–842, Dec/ 2008.

[10] [Online]`https://en.wikipedia.org/wiki/Foveon_X3_sensor`

[11] Boyat A K , Joshi B K . A Review Paper : Noise Models in Digital Image Processing[J]. Signal & Image Processing An International Journal, 2015, 6(2):63-75.

[12] Hou X , Zhang L . Saliency Detection: A Spectral Residual Approach[C] 2007 IEEE Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2007.

[13] [Online]`http://www.cs.tut.fi/~foi/GCF-BM3D`

[14] Markku Mäkitalo, Foi A . Optimal Inversion of the Anscombe Transformation in Low-Count Poisson Image Denoising[J]. IEEE Transactions on Image Processing, 2011, 20(1):99-109.

Appendices

Environment:
Xcode version 10.1(10B61) command line tool (C++). Edit your working directory in: $Run \rightarrow Options \rightarrow WorkingDirectory$ where your raw images is stored.
Cmake 2.8. A wrapping is made for using. Put you image under $/bin$ and $./run$ in command line tool.

Data Organization:
Images are stored in a $3D\ vector[channel][height][width]$, $dtype = uint8\_t$. If $channel = 3$, colors are saved in R, G, B in 0, 1, 2.
All functions can process both color and gray image if there is no specific notice.

Basic Organization

$Basic.hpp$: reading, write raw functions and data type change functions.
$Matrix.hpp$: operation on matrix, ex: convolution, addition etc.
$ImgBasic.hpp$: general function for image processing used in homework, like generate gaussian core etc.
$Evaluation.hpp$: evaluate the result, SNR, PSNR, SSIM etc.
$ImgDemHistMupl.hpp$: realization for $Homework1\ Problem1$.
$ImageDenoising.hpp$: realization for $Homework1\ Problem2$.

A Basic.hpp

1. Functions for read and write the raw image;
2. basic functions called by image processing functions;
3. center pixels.

A.1 $vector < vector < vector < uint8\_t >>>$ ReadRaw($const\ char*$ Path, int BytesPerPixel, $vector < int >$ Size)

Read raw image (8-bit) from file;

1. Path: path to the input file;
2. BytesPerPixel: number of channel;
3. Size: 1D vector: height, width;
4. return a 3D vector with image data.

A.2 $void$ WriteRaw($const\ char*$ Path, $vector < vector < vector < uint8\_t >>>$ Res, int BytesPerPixel, $vector < int >$ Size)

Write image (8-bit) to file;

1. Path: path to the output file;

2. Res: image data to be saved;
3. BytesPerPixel: number of channel;
4. Size: 1D vector: height, width.

Warnings:
Other functions in $Basic.hpp$ would not be introduced, they are deep layers for homework problems.

B ImgDemHistMupl.hpp

Functions for $Problem1$ in $Homework1$;

B.1 BiDemosaicingC($vector < vector < vector < uint8\_t >>>$ ImageData)

Bilinear Demosaicing using convolution method. The one that consider all edges carefully is in $fancy.hpp$.
Default using reflect padding. Zero padding is available as well.

1. ImageData: Input CFA image (if channel > 1 only used first channel);
2. return a 3D vector saved the color image result.

B.2 $vector < vector < vector < uint8\_t >>>$ MHCDemosaicing($vector < vector < vector < uint8\_t >>>$ ImageData)

Malvar-He-Cutler (MHC) Demosaicing using explicit convolution method.
Default using reflect padding. Zero padding is available as well.

1. ImageData: Input CFA image (if channel > 1 only used first channel);
2. return a 3D vector saved the color image result.

B.3 $void$ saveHistogram($vector < vector < vector < uint8\_t >>>$ ImageData, $const\ char*$ name)

Save histogram to a 300*300 image.
The frequency of histogram is normalized to $(0 - 255)$. The width of histogram is 256 pixel, indicate pixel value of raw image $0 - 256$.
Modified on 2019/01/10:
To get better visualized histogram result on noised image, especially the color noise image, I ignored the pixel value = $0/255$ on each channel, while the previous one do not.

1. ImageData: Input image;
2. name: name for the save histogram (.raw).

B.4 $vector < vector < vector < uint8\_t >>>$ HistEqMethodA($vector < vector < vector < uint8\_t >>>$ ImageData)

Method A which use a transfer function, it would be saved.

1. ImageData: Input image.

B.5 $vector < vector < vector < uint8\_t >>>$ HistEqMethodB($vector < vector < vector < uint8\_t >>>$ ImageData)

Method A which use a cumulative histogram, it would be saved.

1. ImageData: Input image.

C ImageDenoising.hpp:

Functions for $Problem2$ in $Homework1$.

C.1 $vector < vector < vector < uint8\_t >>>$ getNoise($vector < vector < vector < uint8\_t >>>$ ImageData, $vector < vector < vector < uint8\_t >>>$ NoiseImageData)

Get the noise in reference of noise free image.

1. ImageData: image free of noise;
2. NoiseImageData: image with noise.

C.2 $vector < vector < vector < uint8\_t >>>$ UniformFilter($vector < vector < vector < uint8\_t >>>$ ImageData, int corelen)

Uniform Filter. Default using reflect padding. Zero padding is available as well.

1. ImageData: Input image;
2. corelen: length for a square core.

C.3 $vector < vector < vector < uint8\_t >>>$ GaussianFilter($vector < vector < vector < uint8\_t >>>$ ImageData, int corelen, $double$ sigma)

Gaussian Filter. Default using reflect padding. Zero padding is available as well.

1. ImageData: Input image;
2. corelen: length for a square core;
3. sigma: standard deviation for Gaussian Core.

C.4 $vector < vector < vector < uint8\_t >>>$ BilateralFilter($vector < vector < vector < uint8\_t >>>$ ImageData, int corelen, $double$ sigmaD, $double$ sigmaR)

Bilateral Filter. Default using reflect padding.

1. ImageData: Input image;
2. corelen: length for a square core;
3. sigmaD: standard deviation for Gaussian Core;
4. sigmaR: standard deviation for similarity Core.

C.5 $vector < vector < vector < uint8\_t >>>$ NLMeansFilter($vector < vector < vector < uint8\_t >>>$ ImageData, $double$ sigma, int searchW, int compW)

None Local Means Filter. Default using reflect padding.

1. ImageData: Input image;
2. corelen: length for a square core;
3. sigma: standard deviation;
4. searchW: size of search window;
5. compW: size of compare/average window.

C.6 $vector < vector < vector < uint8\_t >>>$ ShotNoise($vector < vector < vector < uint8\_t >>>$ ImageData, int corelen, $double$ sigma)

Denoising the shot noise using anscombe transformation, and general Gaussian Filter. A version using BM3D is realized in Matlab $DeShotNoise.m$.

1. ImageData: Input image;
2. corelen: length for a square Gaussian Core;
3. sigma: standard deviation for Gaussian Core.

C.7 $vector < vector < vector < uint8\_t >>>$ NLMeansFilterG($vector < vector < vector < uint8\_t >>>$ ImageData, $double$ sigma, $double$ sigmaD, int searchW, int compW)

None Local Means Filter using gaussian core to get the mean. Default using reflect padding.

1. ImageData: Input image;
2. corelen: length for a square core;
3. sigma: standard deviation;
4. sigmaD: standard deviation for gaussian core;
5. searchW: size of search window;
6. compW: size of compare/average window.

More:

A function name $Fancy1()$ is the demo for the best de-noising result on $rose\_color\_noise.raw$ using the structure I proposed.

It would write the processed image to $fancy1\_result.raw$, and return it.