HOMEWORK 1

Jiazhi Li

jiazhil@usc.edu

January 22, 2019

1 Bilinear Demosaicing

1.1 Abstract and Motivation

In order to record the beautiful scenes in the real world, human beings design the camera and with the development of technology, the digital camera is intended. Inside the digital camera, there are several image sensors named CMOS. The CMOS can only record intensity of light but they cannot distinguish the color of the light. As we know, in the real world, the light can be divided into three components such as Red, Green and Blue. By this way, we need to add some color filter to block other two color lights and let only one kind of color light absorbed by the CMOS. After that, the gray-level image is obtained. But we are not satisfied with the gray-level image. The real colorful image is what we want. Thus, several different Demosaicing techniques have been fostered. By this way, although we only recorded one kind of color information for each pixel, the other two colors can be re-constructed based on their neighbor pixel values to obtain the full color. Among the techniques of Demosaicing, the bilinear interpolation is the simplest technique.

1.2 Approach and Procedures

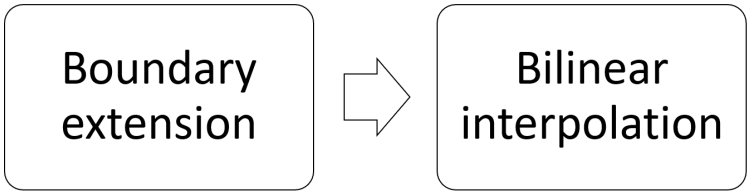
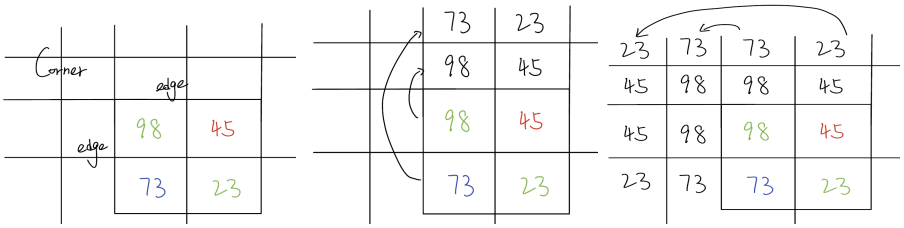


Figure 1: Flow diagram for Bilinear demosaicing

The whole flow diagram is shown in **Figure** 1.



(a) Origin image (b) After row reflection (c) After column reflection

Figure 2: Mirror reflection

Before applying Bilinear interpolation, we need to consider the corner and edge of the image. Thus, there are two method for the same goal in algorithm. The first one is to design different cases for corner, edge and center. The other one is to extend the boundary. Since there are many cases that we need to consider corner and edge, designing different algorithm for each cases is time-consuming. By this way, the function `Imagedata::Boundaryextension(int ex)` is designed for boundary extension. The ex is parameter for the extension level. For example, when we apply ex as 2 and the origin size of image is 390*300, then we get the result image with 394*304. What' more, the method of boundary extension used in the whole project is mirror reflection. The more details about mirror reflection is shown in **Figure** 2.

1.3 Experimental Results

Followed the instructions as mentioned, we get the result image by bilinear demosaicing which shown in **Figure** 3.

(a) CFA (b) Origin image (c) Result image by bilinear demosaicing

Figure 3: Differences between origin image and result image by bilinear demosaicing

1.4 Discussion

Although Bilinear interpolation is fast and simple to implement, which only uses the nearest pixel values in order to find the appropriate color intensity values of that pixel, it tends to produce a greater number of interpolation artifacts such as blurring and edge halos. Having observed the result image obtained in details, we can easily find there are several artifacts including but not limited to:

- Less details
- Edge blurring
- "Zipper effect"
- False color effect

These artifacts might be caused by that when we consider the one single color of each pixel, the other information of other two channels haven't been considered. In the real world, actually the color of one single point is influenced not only by the corresponding channel but also by other two channels. Thus, one way to improve the demosaicing performance is to add the gray-level of two other channels with specific weights to the calculated channel. Followed this thought, Malvar-He-Cutler Demosaicing is designed, which is presented as followed.

What's more, the obvious artifact "zipper effect" in the interpolated image is caused by that bilinear interpolation is done by averaging neighboring pixels indiscriminately. To combat with this artifact, it is natural to derive an algorithm that can detect local spatial features present in the pixel neighborhood and then makes effective choices as to which predictor to use that neighborhood. The result is a reduction or elimination of "zipper-type" artifacts. And algorithms that involve this kind of "intelligent" detection and decision process are referred as adaptive color interpolation algorithms[1].

2 Malvar-He-Cutler(MHC) Demosaicing

2.1 Abstract and Motivation

The above-mentioned method is about bilinear demosaicing which is the simplest method of demosaicing so that there are many artifacts. In order to improve the demosaicing performance by adding the gray-level of two other channels with specific weights to the calculated channel, Malvar-He-Cutler Demosaicing is introduced.

2.2 Approach and Procedures

Since MHC demosaicing is just a improvement method based on bilinear demosaicing, we also need the result of bilinear demosaicing. By this way, the flow diagram is shown in **Figure** 4.

Figure 4: Flow diagram for Malvar-He-Cutler demosaicing

2.3 Experimental Results

Followed the instruction as mentioned, we get the result image by MHC demosaicing. The example in **Figure** 5 shows that Malvar-He-Cutler is visually sharper than bilinear.

(a) Origin image (b) Result image by bilinear demosaicing (c) Result image by Malvar-He-Cutler demosaicing

Figure 5: Differences between result image by bilinear demosaicing and MHC demosaicing

2.4 Discussion

In MHC Demosaicing, the gray-level of the single color doesn't depend only on the gray-level of surrounding pixels in corresponding channel, but also depend on the gray-level of other colors in a broader window. By this way, it effectively improves the performance by weaken the false color effect. Besides, having observed the face of the cat, we can easily find that there are much more details and less color distortion after MHC demosaicing compared to bilinear demosaicing. What' more, the PSNR of MHC demosaicing is better than that of bilinear demosaicing shown in Table 1. Thus, from both PSNR calculation and our sight, MHC demosaicing make some progress in demosaicing.

Table 1: PSNR of different Demosaicing methods for cat.raw

Method	PSNR(dB)
Bilinear	23.898
MHC	27.956

3 Histogram Manipulation

3.1 Abstract and Motivation

In some cases, the image is a little dark or bright which means that the average of gray-level is too low or too high. Thus, this time, we want to invest some methods to equalize the gray-scale which called histogram equalization. When we apply the histogram equalization to some images with brightness concentrated in low gray scale or high scale, it enable the brightness separate more evenly in whole gray scale.

3.2 Approach and Procedures

What is worth mention first is that all figure in this problem such as histogram, transfer function and cummulative histogram are created by C++ without any toolkit. The method applied to plot these figure is shown in **Figure** 6. All of these figure have the intensity value as the x-axis and the number of pixels as the y-axis.

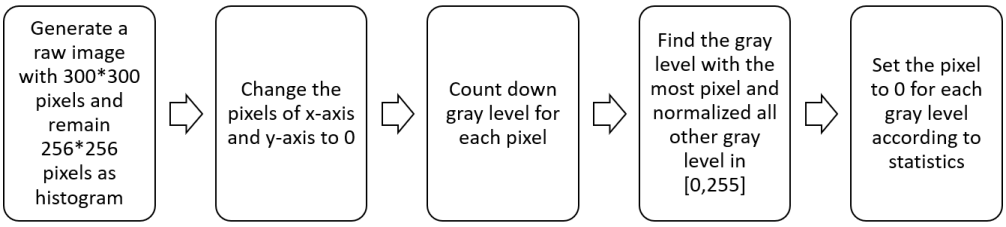


Figure 6: Flow diagram for histogram plotting by C++

The raw image and the histogram of the raw image is shown in **Figure** 7.

(a) Raw image (b) Histogram

Figure 7: Raw image and histogram for rose.raw

There are two method for histogram equalization:

- Method A: Transfer-function-based histogram equalization method
- Method B: Cumulative-probability-based histogram equalization method

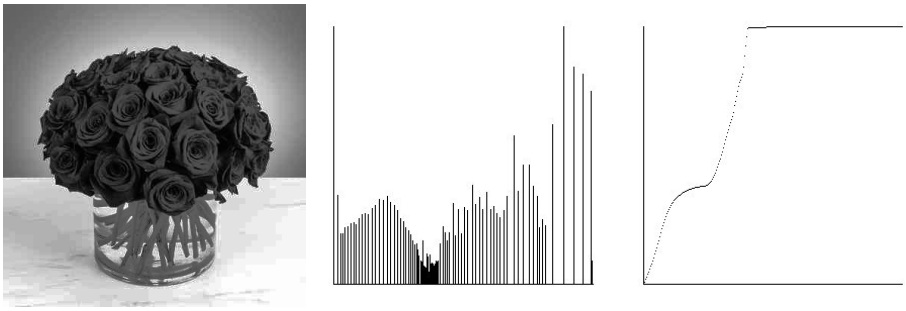
3.3 Experimental Results

The raw image and the histogram of the dark image are shown in **Figure** 8.

(a) Raw image (b) Histogram

Figure 8: Raw image and histogram for rose dark.raw

The result image, the histogram and the transfer function of the dark image after method A are shown in **Figure** 9.



(a) Result image (b) Histogram (c) Transfer function

Figure 9: Result for rose dark.raw by method A

The result image, the histogram and the cummulative histogram of the dark image after method B are shown in **Figure** 10. Something need to be explained. We can find that the histogram is almost a black figure. It also makes sense because after method B the number of pixels in each gray level is same and the number is 625. This time, instead of normalized 625 as 1/256 which is really not good looking in histogram, I just normalized 625 as 1 so that it look like that all values of gray level goes to maximum.

(a) Result image (b) Histogram (c) Cummulative histogram

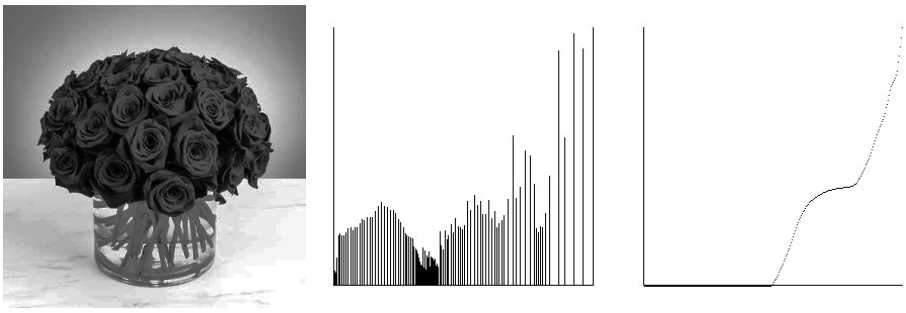
Figure 10: Result for rose dark.raw by method B

The raw image and the histogram of the bright image are shown in **Figure** 11.

(a) Raw image (b) Histogram

Figure 11: Raw image and histogram for rose bright.raw

The result image, the histogram and the transfer function of the bright image after method A are shown in **Figure** 12.



(a) Result image (b) Histogram (c) Transfer function

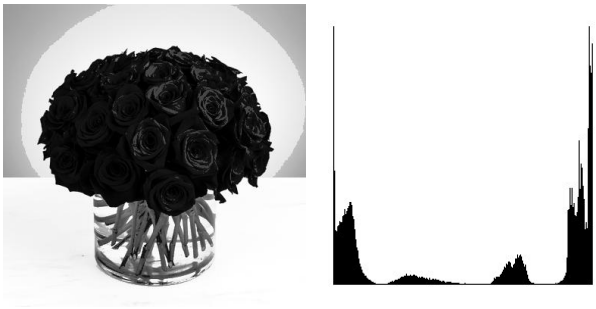
Figure 12: Result for rose bright.raw by method A

The result image, the histogram and the cummulative histogram of the bright image after method B are shown in **Figure** 13.

(a) Result image (b) Histogram (c) Cummulative histogram

Figure 13: Result for rose bright.raw by method B

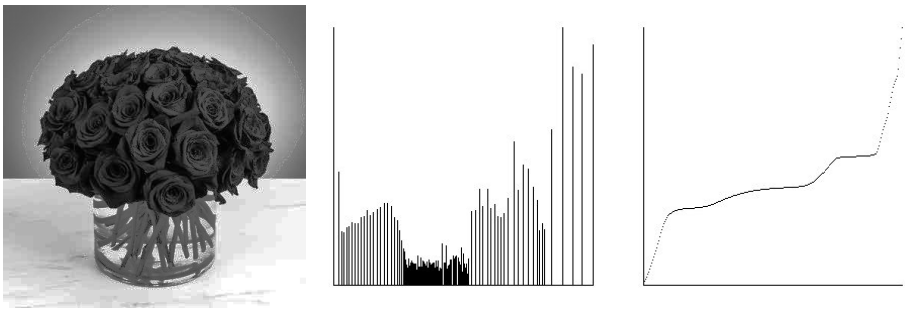
The raw image and the histogram of the mix image are shown in **Figure** 14.



(a) Raw image (b) Histogram

Figure 14: Raw image and histogram for rose mix.raw

The result image, the histogram and the transfer function of the mix image after method A are shown in **Figure** 15.



(a) Result image (b) Histogram (c) Transfer function

Figure 15: Result for rose mix.raw by method A

The result image, the histogram and the cummulative histogram of the mix image after method B are shown in **Figure** 16.

(a) Result image (b) Histogram (c) Cummulative histogram

Figure 16: Result for rose mix.raw by method B

3.4 Discussion

From the PSNR shown in Table 2, the result after two different method is quite same but method A is a little better than method B. However, this advantage may only exist for these three images. Both of them perform a little bit badly in pure black or white background and edges especially stem of rose compared to the origin image. The histogram equalization improves the whole contrast of the image and enlarge domain of gray level. However, both of two method make the lack of some specific gray level, which is the artifacts that less detail about edge and noise generation. By this way, one way to improve the method is to apply the Gaussian filter first to get the low frequency component of the image. At this time , we can only apply the histogram equalization to low frequency component. After that, we add the low frequency component after histogram equalization to remaining high frequency component so that the information about details and edge would not be changed[2].

Table 2: PSNR of different Histogram equalization method

	Method	PSNR(dB)
Dark	-	7.630
	A	16.874
	B	16.637
Bright	-	11.363
	A	16.871
	B	16.638
Mix	-	22.894
	A	16.762
	B	16.567

When the two methods applying to rose mix.raw, the histogram has both extremely dark and extremely bright part, which means it has high contrast. We can get quite similar result as in previous part. And method A perform a litter bit better than method B in stem of flower. However, the table part looks like a little dirty,

which is the part “sacrificing” in histogram equalization.
For method A, in order to make it better, we can consider the bilinear interpolation between pixels to fill in the missing gray level. By this way, the image will look like smoother.
For method B, we can introduce a threshold. For the gray level whose pixel number is less than threshold, it is not necessary to take part in “bucket filling”. They can just preserve the origin gray level which make image have high fidelity. Only for the part with high pixel number greater than threshold, they must move some pixel to other gray level.

4 Gray-level image denoising

4.1 Abstract and Motivation

Due to the influence of sensor material properties, working environment, electronic components and circuit structure, the image acquired by CCD and CMOS will introduce various noises, such as thermal noise caused by resistance, FET Channel thermal noise, photon noise, dark current noise, optical response non-uniform noise[3].
According to the definition of noise in power spectral density, the kinds of noise can be: (Discussed in our project)

- Gaussian noise
- Uniform noise
- Impulse noise
- Shot noise

In the following passage, I will introduce different filters to wipe off these kind of noises.

4.2 Approach and Procedures

At this part, there are five kinds of filter involving:

- Uniform filter
- Gaussoam filter
- Bilateral filter
- Non-local mean filter
- Median filter

The flow diagram is shown in **Figure** 17.

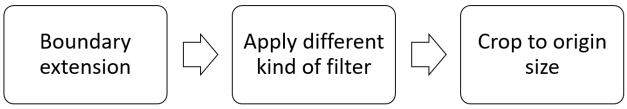


Figure 17: Flow diagram for denoising

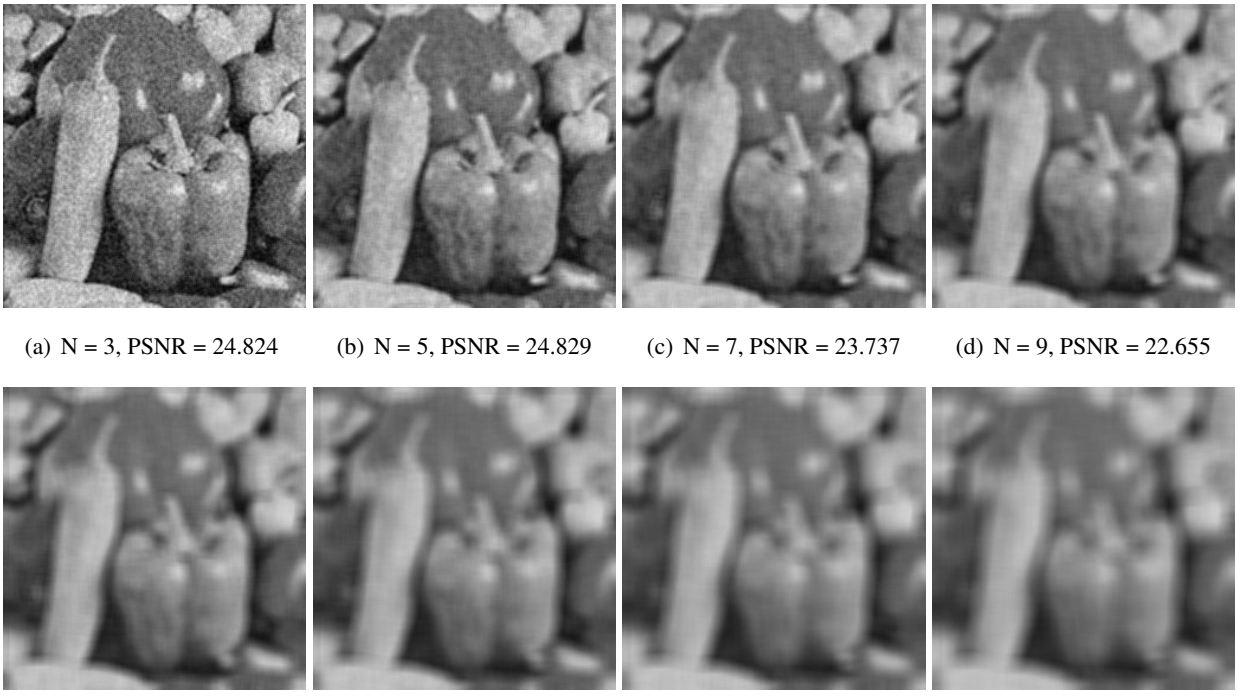
The image without noise and with noise for pepper.raw are shown in **Figure** 18.

(a) Raw image (b) With noise

Figure 18: Raw image and the image with uniform noise

4.3 Experimental Results

The result after uniform filter is shown in **Figure** 19.



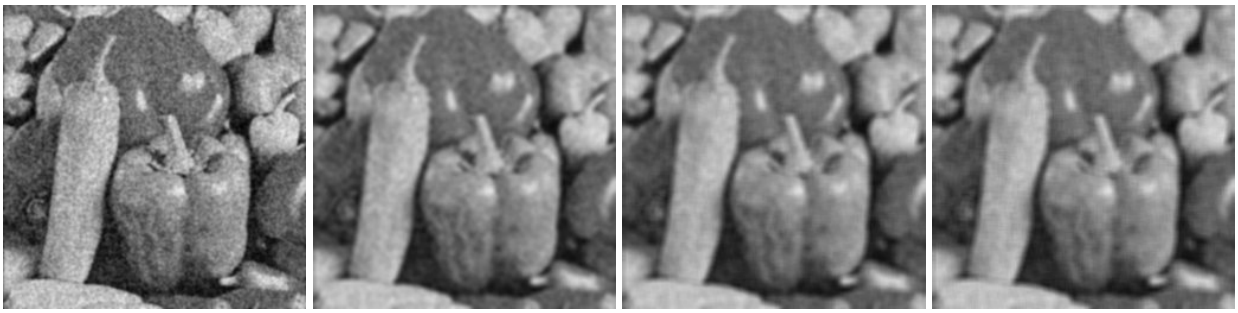
(a) N = 3, PSNR = 24.824 (b) N = 5, PSNR = 24.829 (c) N = 7, PSNR = 23.737 (d) N = 9, PSNR = 22.655

(e) N = 11, PSNR = 21.752 (f) N = 13, PSNR = 21.007 (g) N = 15, PSNR = 20.383 (h) N = 17, PSNR = 19.848

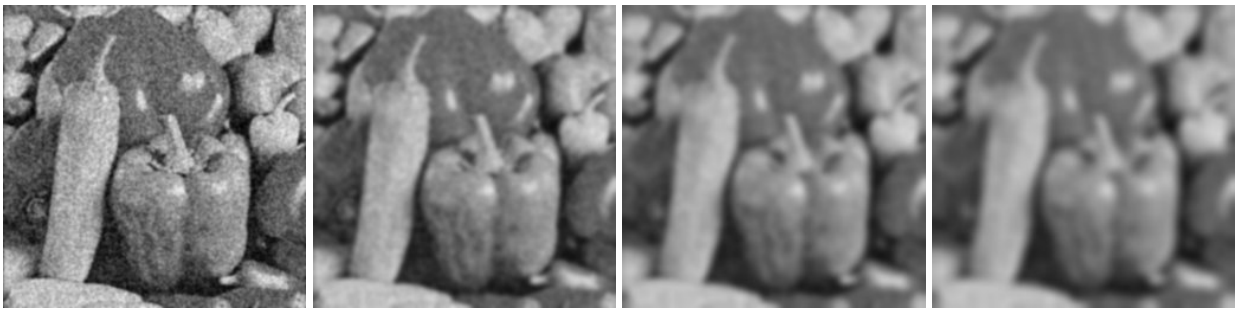
Figure 19: Result image after uniform filter for different window size

The result after Gaussian filter is shown in **Figure** 20. The notation is that (Window size, standard deviation).

(a) (3,1.0) PSNR = 24.761 (b) (3,2.0) PSNR = 24.882 (c) (3,3.0) PSNR = 24.853 (d) (3,4.0) PSNR = 24.840



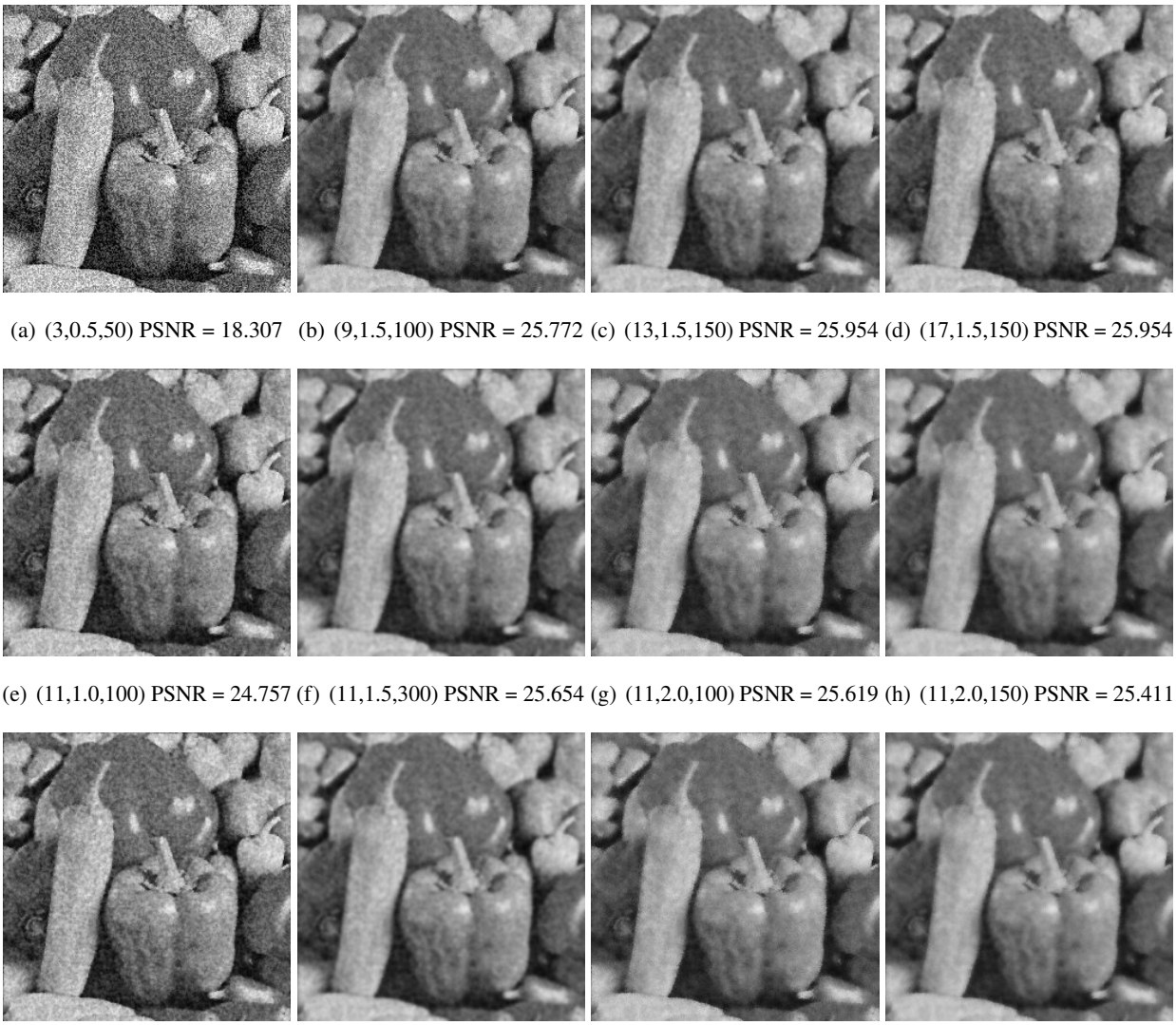
(e) (7,1.0) PSNR = 25.597 (f) (7,2.0) PSNR = 24.959 (g) (7,3.0) PSNR = 24.332 (h) (7,4.0) PSNR = 24.078



(i) (15,1.0) PSNR = 25.598 (j) (15,2.0) PSNR = 24.507 (k) (15,3.0) PSNR = 22.831 (l) (15,4.0) PSNR = 21.872

Figure 20: Result images after Gaussian filter for different window size and standard deviation

The result after Bilateral filter is shown in **Figure** 21. The notation is that (Window size, sigmac, sigmas).



(a) (3,0.5,50) PSNR = 18.307 (b) (9,1.5,100) PSNR = 25.772 (c) (13,1.5,150) PSNR = 25.954 (d) (17,1.5,150) PSNR = 25.954

(e) (11,1.0,100) PSNR = 24.757 (f) (11,1.5,300) PSNR = 25.654 (g) (11,2.0,100) PSNR = 25.619 (h) (11,2.0,150) PSNR = 25.411

(i) (13,1.0,100) PSNR = 24.757 (j) (13,1.5,300) PSNR = 25.654 (k) (13,2.0,100) PSNR = 25.603 (l) (13,2.0,150) PSNR = 25.387

Figure 21: Result images after Bilateral filter for different parameters

The result after non-local mean filter is shown in **Figure** 22. The notation is that (Window size, standard deviation, Search window size, filter parameter h).

(a) (7,0.5,3,3) PSNR = 23.567 (b) (11,1.5,2,2) PSNR = 22.453 (c) (15,1.0,5,5) PSNR = 21.464 (d) (15,1.5,7,7) PSNR = 24.986

Figure 22: Result images after non-local mean filter for different parameters

4.4 Discussion

The type of embedded noise in pepper image is uniform noise. Having count the window size and PSNR in MATLAB, I get the following **Figure** 23 and **Figure** 24.

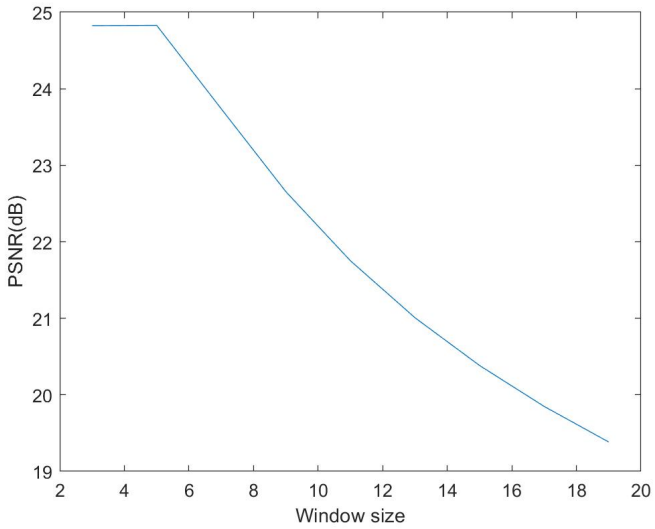


Figure 23: Uniform filter

With these figures and visual quality, we can find that too small or too large number of window size is not good for denoising. The small window cannot wipe off the noise effectively. However, the big window will also blur edge of image.

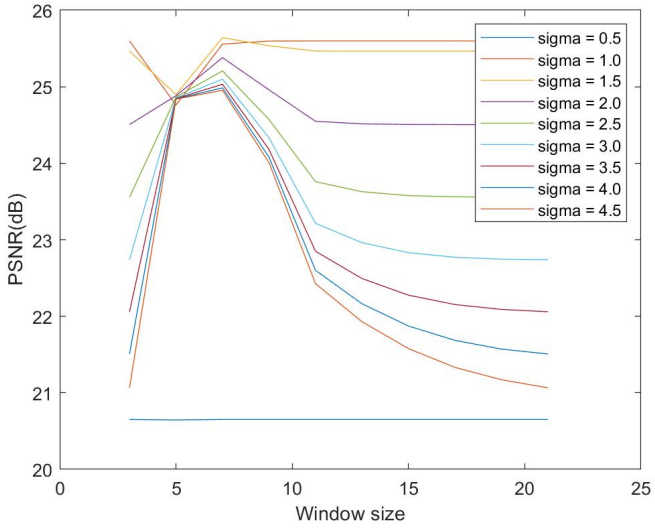


Figure 24: Gaussian filter

Table 3: Step distance for different filter

Method	Window size	Standard deviation	sigmac	sigmas	Search Window size	Filter parameter h
Uniform	3:2:23	-	-	-	-	-
Gaussian	3:2:23	0.5:0.5:5	-	-	-	-
Bilateral	3:2:17	-	0.5:0.5:2.0	50:50:300	-	-
Non-local mean	3:2:15	0.5:0.5:2	-	-	3:2:15	1:1:10

In order to find the best parameter for each filter, I set step distance for each parameter and run the program several time. After comparing PSNR, I find the best parameter for each filter. The step distance is shown in Table 3. The notation is that "start value: step distance : end value". And the best PSNR for each filter is shown in Table 4.

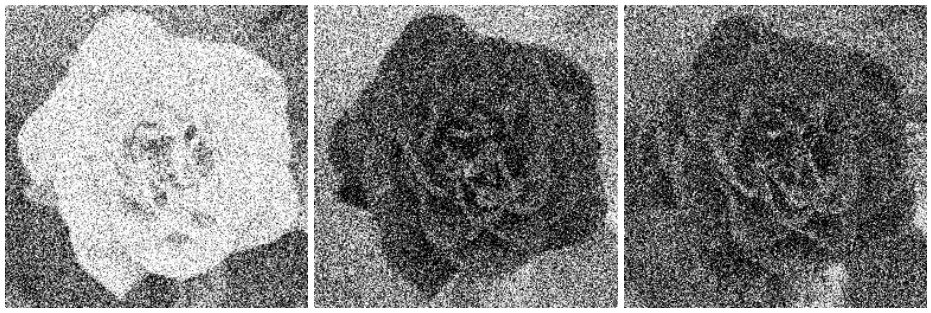
Table 4: PSNR of different filters for cat.raw

Method	Parameter	PSNR(dB)
Uniform	N = 5	24.829
Gaussian	(15,1.0)	25.598
Bilateral	(13,1.5,150)	25.954
Non-local mean	(3,1,31,9)	27.956

5 Color image denoising

5.1 Abstract and Motivation

Similar with gray-level image, the color image might also have some noises in each channel. Thus, we may need to consider each channel separately. The gray-level image of each channel is shown in **Figure** 25.



(a) R channel (b) G channel (c) B channel

Figure 25: Gray-level of three channel of rose noise

5.2 Approach and Procedures

Since the flower is red, the background is green and it is almost dark for Blue channel, we should apply different filter for each channel.

Having tried many combination of filters, I decide to use the combination of Uniform filter, Non-local mean filter and Gaussian filter for red channel. The flow diagram is shown in **Figure** 26. All of these filters are cascaded.

Figure 26: Flow diagram for red channel

For the other two channel, I just use the uniform filter.

5.3 Experimental Results

In this part, I just present the intermediate result after three-level filter in **Figure** 27.

(a) Noise image (b) After Uniform Filter (c) After Non-local mean filter (d) After Gaussian filter

Figure 27: Result for rose color.raw

5.4 Discussion

In this color image, there are impulse noise and uniform noise. Since there are RGB three channel, we can perform filtering on individual channels separately.

Since the several filters have been implemented, I have tried my best to find the best parameter for each filter separately applied to rose color.raw in order to get the highest PSNR. The result is shown in Table 5. This time, I also design median filter which is always the best filter for impulse noise. However, there are impulse noise and uniform noise in this color image. All of these filters don't perform very well in this case.

Table 5: PSNR of different Demosaicing methods for rose color.raw

Method	PSNR(dB)
Uniform	16.874
Gaussian	16.855
Bilateral	17.946
Non-local mean	18.324
Median	17.453

Thus, by this way, I try to use cascaded filter for these two types of noise. Having tried many combinations of filters, I decide to use the combination of Uniform filter, Non-local mean filter and Gaussian filter. All of these filters are cascaded[4].

It is true that filter may blur the object's edge when smoothing noise. By this way, a successful design of uniform noise filter depends on its ability to distinguish the pattern between the noise and the edges. For example, this rose image can be segmentation as background and flower. In the part of flower, there are many details and edge information so that we can apply Gaussian filter with low window size and low standard deviation for this part to remain the edge. And for the background part, since there are some information that we don't care about compared to flower itself, we use the Gaussian filter with high window size. The flow diagram is shown in **Figure** 28.

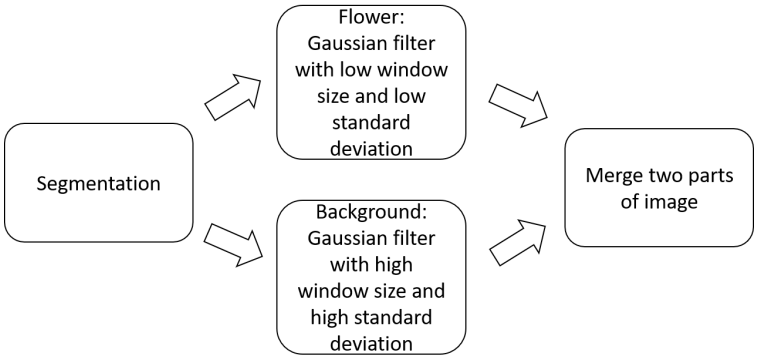


Figure 28: Flow diagram for improvement of filter system

6 Shot noise

6.1 Abstract and Motivation

Shot noise, the time-dependent fluctuations in electrical current caused by the discreteness of the electron charge, is well known to occur in solid-state devices, such as tunnel junctions, Schottky barrier diodes and p-n junctions[5]. Unlike uniform noise, shot noise is Poisson distributed so that we cannot apply the above-mentioned denoising filter directly.

6.2 Approach and Procedures

Before denoising, we need to apply Anscombe root transformation firstly. After that, there are two ways for denoising:

- Method A: Use a Gaussian low pass filter
- Method B: Use the MATLAB code for block-matching and 3-D(BM3D) transform

What's more, the flow diagram is shown in **Figure** 29 and **Figure** 30. And the main function for BM3D by MATLAB is in the folder as main.m.

Figure 29: Flow diagram for Gaussian low pass filter by C++

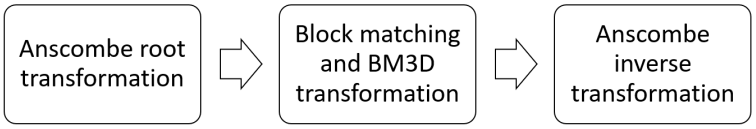
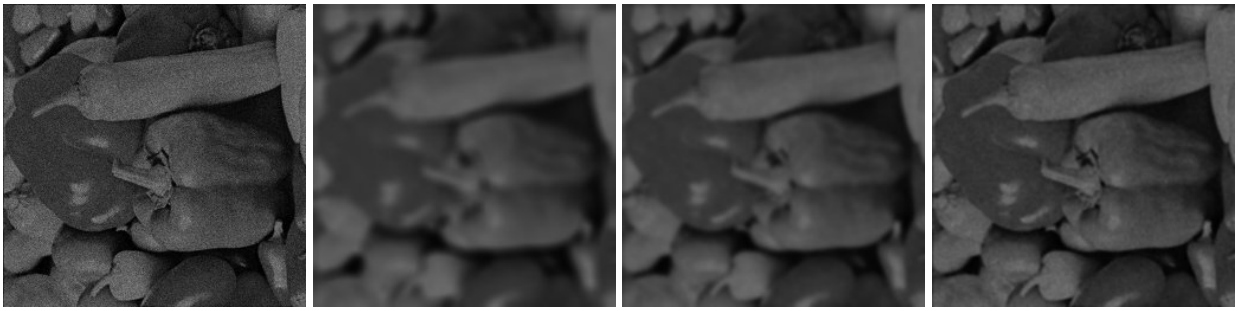


Figure 30: Flow diagram for BM3D transform by MATLAB

6.3 Experimental Results

For method A, I set window size from 3 to 21 and step distance is 2, and standard deviation from 1 to 5 and step distance is 1. For the reason of length, I just present part of results in **Figure** 31. The notation for the following figure is that (window size, standard deviation).



(a) (3,0.1) PSNR = 26.984 (b) (11,4.0) PSNR = 23.875 (c) (15,2.0) PSNR = 25.984 (d) (21,1.0) PSNR =28.874

Figure 31: Result for pepper dark.raw by method A

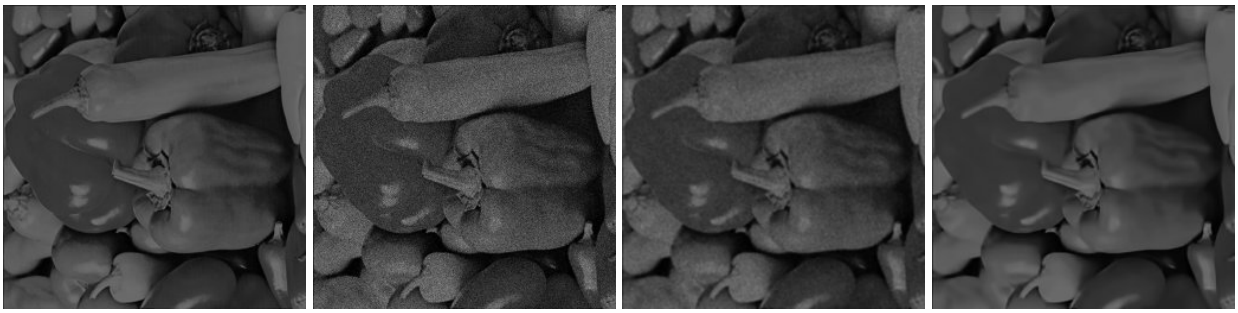
For method B, I set sigma from 1 to 20 and step distance is 1. For the reason of length, I just present part of results in **Figure** 32.

(a) sigma = 5 PSNR = 39.341 (b) sigma = 10 PSNR = 39.764 (c) sigma = 15 PSNR = 39.997 (d) sigma = 20 PSNR = 39.875

Figure 32: Result for pepper dark.raw by method B

6.4 Discussion

For method A, as I make a statistics of PSNR, I just find that parameter for (window size, standard deviation) is (3,0.8) which PSNR can be 35.493405. This is the best PSNR in all my result. The image is shown in **Figure** 33. However, the visual quality is not quite good because of some blurring edges.



(a) Noise free image (b) Noised image (c) Method A (3,0.8) (d) Method B sigma = 15

Figure 33: Conparison of different methods

The method B is quite complicated. The whole BM3D algorithm consists of two steps. They are basic estimate and final estimate. Both of them need Grouping by block-matching, 3D transform, Hard-thresholding, Inverse 3D transform and block-wise estimate. The method B have the excellent performance in denoising from the visual quality. It really wipes off the noise effectively, and remain lots of image details and contain edges of object greatly.

Table 6: PSNR of different Demosaicing methods for cat.raw

Method	PSNR(dB)
Gaussian low pass filter	28.874
BM3D	39.997

Reference

[1] Gao D, Wu X, Shi G, et al. Color demosaicking with an image formation model and adaptive PCA [J]. Journal of Visual Communication and Image Representation, 2012, 23(7):1019-1030.

[2] PIZER, S. M, AMBURN, et al. Adaptive histogram equalization and its variations[J]. Computer Vision Graphics and Image Processing, 1987, 39(3):355-368.

[3] Blouke M M. CCD/CMOS process for integrated image acquisition and early vision signal processing[C]// Charge-coupled Devices and Solid State Optical Sensors. 1990.

[4] Al K N P E. Color Image Processing and Applications[M]. 2000.

[5] De Jong M J M, Beenakker C W J. Shot noise in mesoscopic systems[J]. 1996.

Appendix

In this part, I want to introduce the structure of program in details. There are two classes for the homework, Imagedata and kernel.

Class Imagedata

In class Imagedata, there are five kinds of function such as basic function, Demosaicing, Histogram equalization. Denoise and Shot nosie.

Basic function

```
//basic function
Imagedata(int h, int w, int p); //constructor
~Imagedata(); //destructor
void set_doubledata(); //initialize double data for shot noise
void set_data(int d); //initialize all data with value d
void read(unsigned char* buff); //read data from buff
void load(string path); //load data from path
unsigned char* write(); //write data to buff
void save(string path); //save data to path
Imagedata get_RGB(int i); //get single channel data
int convert(int h, int w, int p);
```

Demosaicing

```
//Demosaicing
Imagedata Boundaryextension(int ex); //Boundary extension
Imagedata Crop(int N); //crop operation
Imagedata Raw2RGB_BL(); //Bilinear demosaicing
Imagedata Raw2RGB_MHC(Imagedata Imagedata_BL); //MHC demosaicing
```

Histogram equalization

```
//Histogram
void Histogram_create(const char* path); //create and save histogram to path
void Histogram_cummulative(const char* path); //create and save cummulative histogram to path
void Transfer_function(const char* path); //create and save transfer function to path
Imagedata Hist_EQ_TR(); //histogram equalization by transfer function
Imagedata Hist_EQ_BF(); //histogram equalization by bucket filling
```

Denoise

```
//Denoise
Imagedata Convolution(Kernel k); //convolution operation with kernel
Imagedata DN_Uniform(int N); //uniform filter
Imagedata DN_Gaussion(int N,double sigma); //Gaussian filter
Imagedata DN_Bilateral(int N, double sigmac, double sigmas); //Bilateral filter
Imagedata DN_Nonlocalmean(int N_neighbor, int N_search, double h, double sigma); //Non-local mean filter
Imagedata DN_Median(int N); //median filter

double PSNR(string path); //calculate PSNR according to the noise-free image in path
void PSNR_save(string path,double PSNR); //save PSNR to path
```

Shot noise

```
//Shot noise
Imagedata Anscombe_forward(); //Anscombe transformation
Imagedata Anscombe_reverse_biased(); //Anscombe inverse transformation(biased)
Imagedata Anscombe_reverse_unbiased(); //Anscombe inverse transformation(unbiased)

Imagedata Boundaryextension_double(int ex); //Boundary extension for double data
Imagedata DN_Gaussion_double(int N, double sigma); // Gaussian filter for double data
Imagedata Convolution_double(Kernel k); // convolution for double data

void double2usignedc(); //change double data to unsigned char data
Imagedata Crop_double(int N); // crop operation for double data
Imagedata Save_double(string path); // save double data to path
double* write_double(); // write double data to buff
```

Class Kernel

```
Kernel(int h, int w); //constructor
~Kernel(); //destructor

void set_data(double d, int i, int j); //set elements(i,j) as value d
double sum(); //get sum
vector<vector<double>> get_wholedata(); //return a two dimension vector as kernel
double get_data(int i, int j); //get the value of element(i,j)
Kernel EQ(); //equalization
```