Problem1-Basic Image Manipulation

a. Color Space Transformation

1. Color-to-Grayscale Conversion

Abstract and Motivation

With the development of science and technology, images became an effective medium to express some information. Therefore, many scientists and engineers strive for inventing some useful algorithms to process image. In the lecture, we got some basic concepts of image, such as image height, image width channels and the range of pixel values. Colorful images have multiple channels. The most popular combination of channels is Red, Blue and Green. If an image is black-and-white. This is a grayscale image composed by only one channel. Grayscale image is widely used in remote sensing area and medical area.

This is the first image processing task for us. Because many students have not got access to image processing, we can get familiar with some basic concepts of image by finishing this task, including the way how the pixel information is stored in an image file and how can we do some manipulation of every pixel. There are there methods for converting color images to grayscale image. We can realize them by doing some basic manipulation to pixel

Task: Convert a color image to its grayscale image.

Approaches and Procedures

Theoretical Approach: There are 3 channels in a RGB color image. Every pixel value of every channel ranges 0 to 255, so they can be compressed in a single channel by doing some calculation. The first way to do this is lightness method by finding the maximum value and minimum value of every pixel and average it. The second way is simply average method that is just calculating its average values. Last, it is the luminosity average with a specific formula: 0.21 R + 0.72 G + 0.07 B.

Implementation by C++:

Step1: Read input image "Tiffany.raw" and store the image data in an unsigned char vector (1D). Read key parameters from command line including its height and width and number of channels.

Step2: Run two nested loop for getting access to every pix value. The index for outer loop ranges from 0 to (imageheight-1) and the inner one ranges from 0 to (imagewdith-1).

Step3: For pixel data stored in a 1-D vector, I used the formula: r*(image width)*(number of channels)+c*(number of channels)+depth to get RGB value of every pixel. Then I can do some calculation to get the corresponding pixel value.

Step4:Loops end and output the processed image by a function called fwrite.

Results:

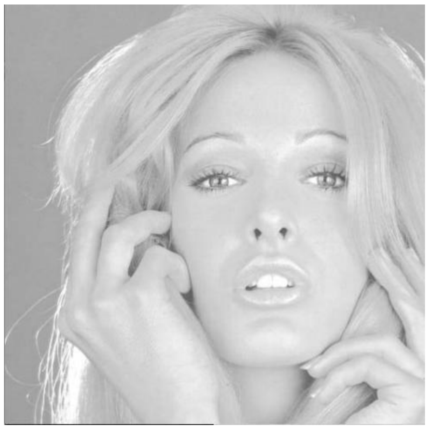


Figure1 lightness method

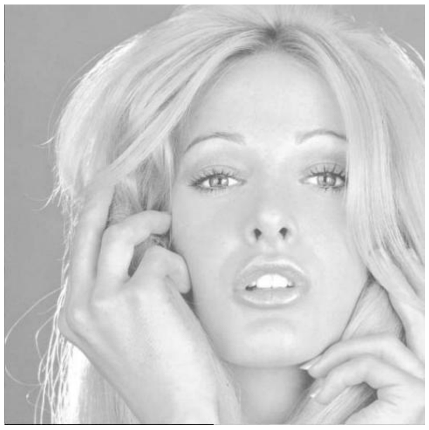


Figure2 average method

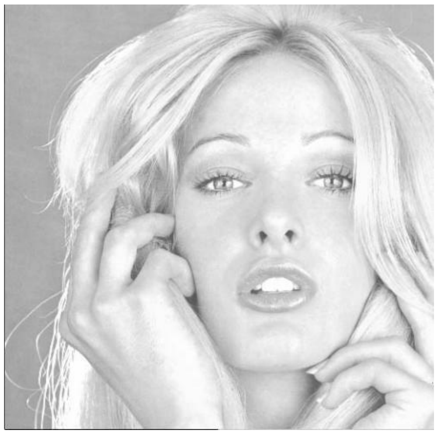


Figure3 luminosity method

Discussion:

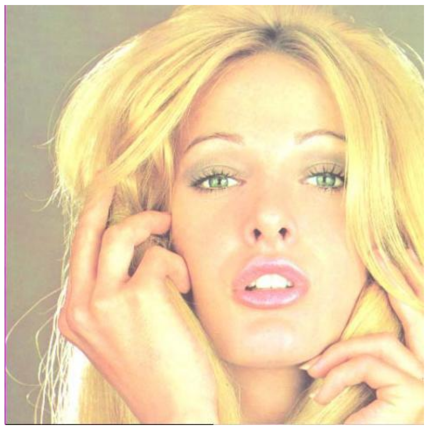


Figure4 original image

We can compare 3 processed images with the original image. We can find that there are many differences between the processed image and original image. For example, the girl's left hand and hair in the original image have different luminance. The front part of her left hand is much brighter than other parts of her left hand. In 3 grayscale images, the luminosity-based one express this detail best.

2. CMY(K) Color Space

Abstract and Motivation

The definition of color space is that all combinations of group values that can represent a color. The number of group values is usually 3 or 4. CMYK is often used in color printing industry, which is quite different from RGB. Elaborately, RGB mode is a luminous color mode and you can see the color on screen even if you are in a dark room. However, CMYK mode depends on reflection. For example, when you are reading a book in a dark room, you cannot see the content of that book because there is no light source and no reflection. Therefore, RGB and CMYK are 2 totally different color spaces. We can use some formula to convert RGB image to CMYK image.

Task: change RGB color image to CMYK image.

Approaches and Procedures

Theoretical Approach:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Ideally, CMY model is simply transposition of RGB Model

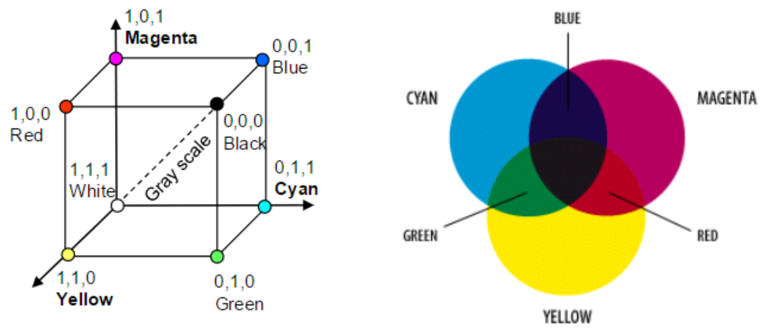


Figure5 RGB TO CMY(Lecture Slide)

Implementation by C++(Take Bear.raw as an example):

Step1: Read input image “Bear.raw” and store the image data in an unsigned char vector (1D). Read key parameters from command line including its height and width and number of channels.

Step2: Run two nested loop for getting access to every pix value. The index for outer loop ranges from 0 to (imageheight-1) and the inner one ranges from 0 to (imagewdith-1).

Step3: For pixel data stored in a 1-D vector, I used the formula: r*(image width)*(number of channels) +c*(number of channels)+depth to get RGB value of every pixel.

Step4: Normalize all RGB value and transfer it to a normalized CMY value according to the formula in Figure5.

Step5: Multiply these normalized pixel values by 255.

Step4: Loops end and output the processed image by a function called fwrite.

Results:

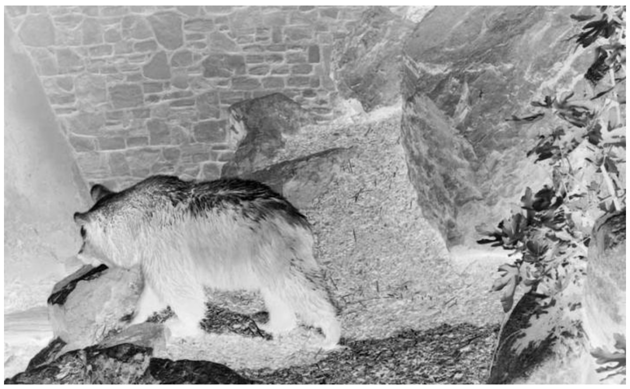


Figure6 Cyan channel of Bear.raw

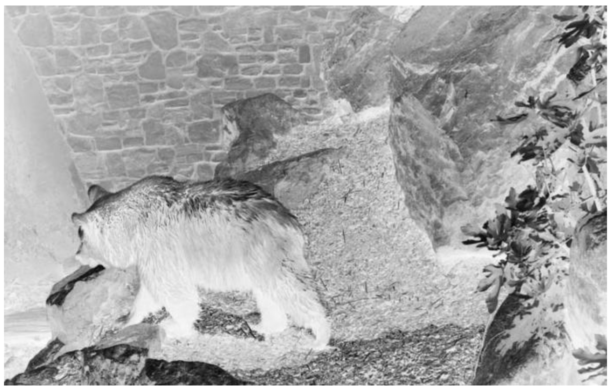


Figure7 Magenta channel of Bear.raw

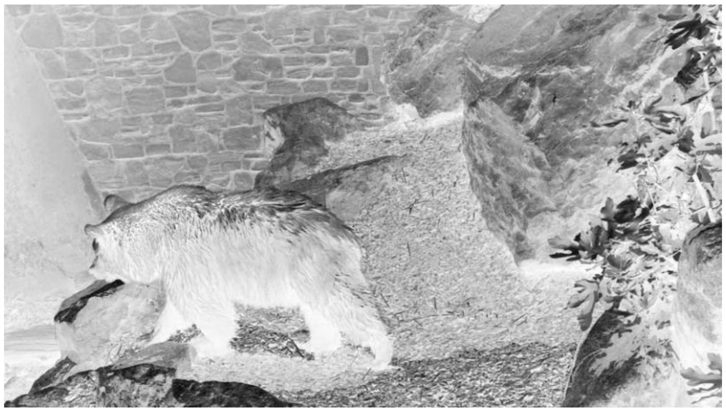


Figure8 Yellow channel of Bear.raw

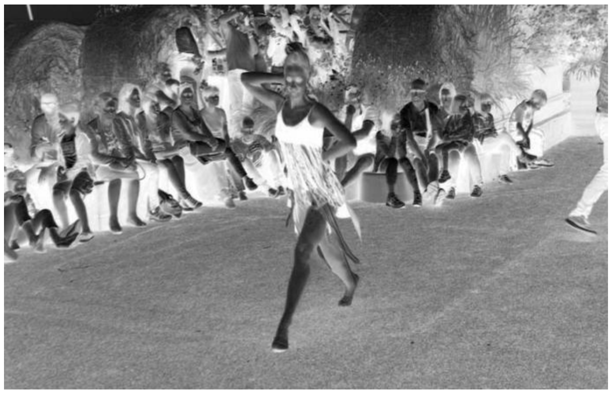


Figure9 Cyan channel of Dance.raw

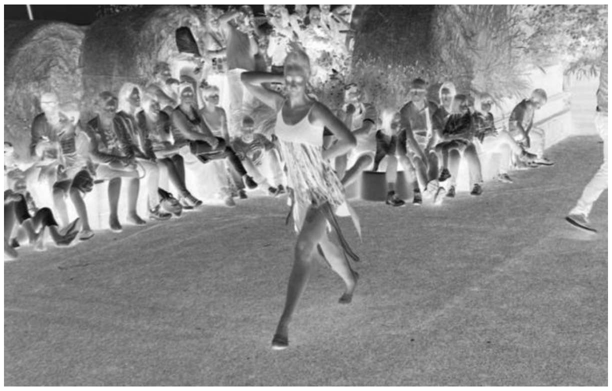


Figure10 Magenta channel of Dance.raw

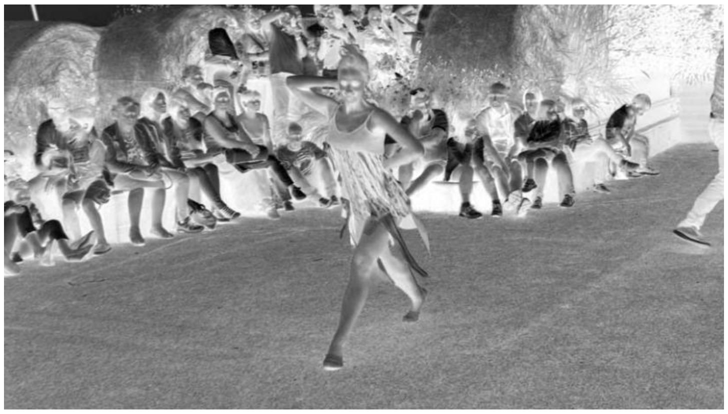


Figure11 Yellow channel of Dance.raw

b. Image Resizing via Bilinear Interpolation

Abstract and Motivation

Image resizing is quite common in our daily life. For example, sometimes we have to adjust the size of image in order to meet particular requirements, including applying visa and finish the report. Also, we can find that it is quite convenient to resize image in Microsoft Word by dragging its edge. With the fundamental manipulation of pixel values, we can easily realize this effect by Bilinear Interpolation.

Task: Resizing image by Bilinear Interpolation.

Approaches and Procedures

Theoretical Approach: The key factor of this task is to find the correct relationship between the pixel of new image and the pixel of old image. This relationship can be expressed in Figure 12:

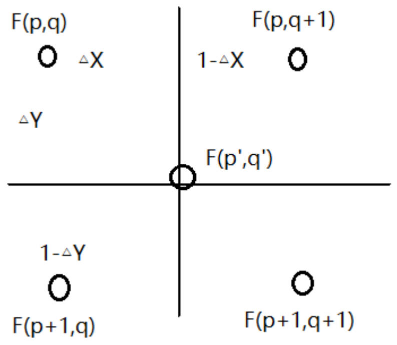


Figure12 Bilinear Interpolation

$F(p',q')$ is a pixel value in new image. First, we must find its corresponding pixel point $F(p,q)$ in the original image. We can get $F(p,q)$ by calculating the corresponding position of the new pixel in the original image. Assume that the size of original image is m*m and we want to get a n*n size image (n>m). The position can be calculated by these formula and finally we got (p,q), Δx and Δy:

$$Coresprow = r*(n/m) \qquad Corespcolumn = c*(n/m)$$

$$p = \lfloor Coresprow \rfloor \qquad q = \lfloor Corespcolumn \rfloor$$

$$\Delta x = Coresprow - p \qquad \Delta y = Corespcolumn - q$$

The last step is calculate $F(p',q')$:

$$F(p',q') = (1-\Delta x)(1-\Delta y)F(p,q) + \Delta x(1-\Delta y)F(p,q+1) + (1-\Delta x)\Delta y F(p+1,q) + \Delta x \Delta y F(p+1,q+1)$$

Implementation by C++:

Step1: Read input image “Airplane.raw” and store the image data in an unsigned char vector (1D). Read key parameters from command line including its height, width, number of channels, newheight and newwidth.

Step2: Create a new vector for output image. Run two nested loop for getting access to every pix value. The index for outer loop ranges from 0 to (newheight-1) and the inner one ranges from 0 to (newwdith-1).

Step3:For every new pixel point in the output image, use previous formula to calculate necessary parameters. Finally, we can get the value for the new pixel point.

Step4:Loops end and output the processed image by a function called fwrite.

Results:

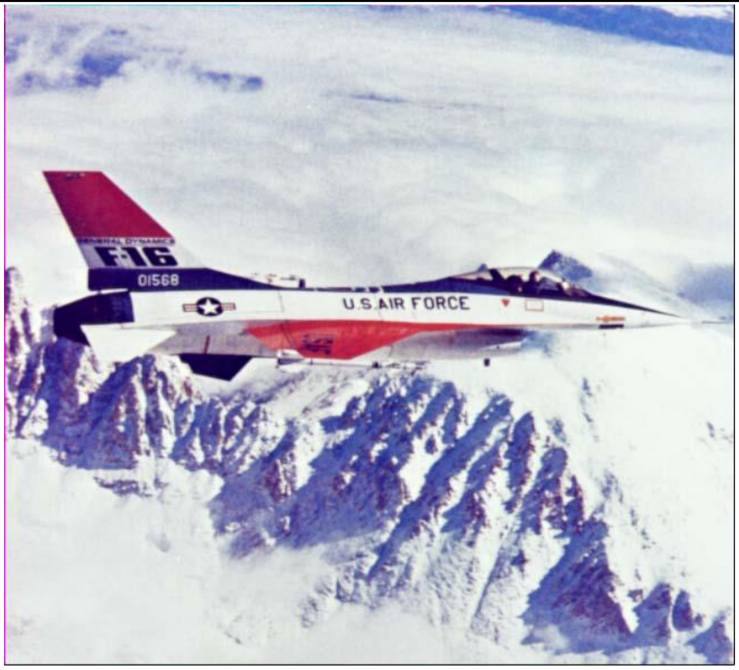


Figure13 Resized Airplane.raw (650*650)

Problem2-Histogram Equalization

a. Histogram Equalization

Abstract and Motivation:

It is quite common that low contrast can bring negative effect to images. Sometimes, due to limited conditions such as insufficient light, bad camera or dark background. Most pixel values distributed in a narrow range. Therefore, some key information might be hidden due to that kind of distribution making the quality of image very poor. From problem1, we gained how to manipulate the pixel values and know the components of an image. Thus, we can enhance the image by 2 method: transfer-function-based method and cumulative-probability-based method.

Task: Do histogram equalization for an image by transfer-function-based method and cumulative-probability-based-method.

Approaches and Procedures

Theoretical Approach: According to the textbook, the best transfer function for histogram equalization is the cdf of the pixel values of the original image. Therefore, we can access the whole image and get some critical statistical information to calculate cdf. Next, in order to get the new pixel value, we assign every “old pixel value” as an input of transfer function(cdf). Finally, the output is equalized pixel value.

The probability-cumulative-based method is that we have to divide total pixels of the original image into 256 parts. Each parts has the same number of pixel.

Implementation by C++(Take transfer function method as an example):

Step1: Read input image “Desk.raw” and store the image data in an unsigned char vector (1D). Read key parameters from command line including its height, width, number of channels.

Step2: Run two nested loop for getting access to every pix value. The index for outer loop ranges from 0 to (height-1) and the inner one ranges from 0 to (width-1). Create 3 1-dimensional arrays called red_pdf, green_pdf and blue_pdf to record the histogram data of each channel. Use fwrite function to output data.

Step3: Create 3 1-D arrays called red_cdf, green_cdf and blue_cdf. Calculate transfer function.

Step4: Use transfer function to get new pixel values.

Step5: Loops end and output the processed image by fwrite function.

Results:

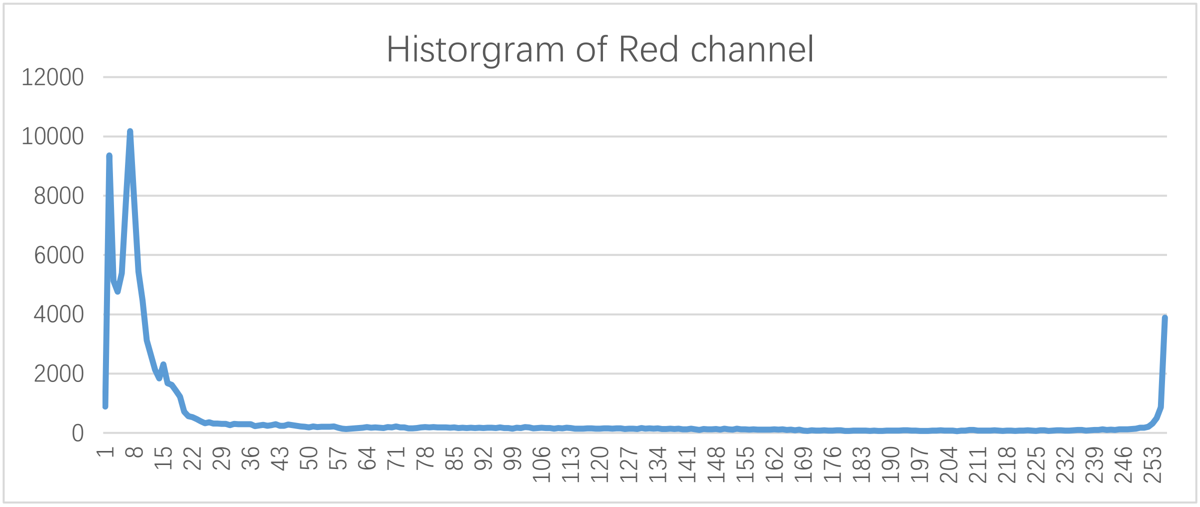


Figure14 Histogram of red channel of the original image

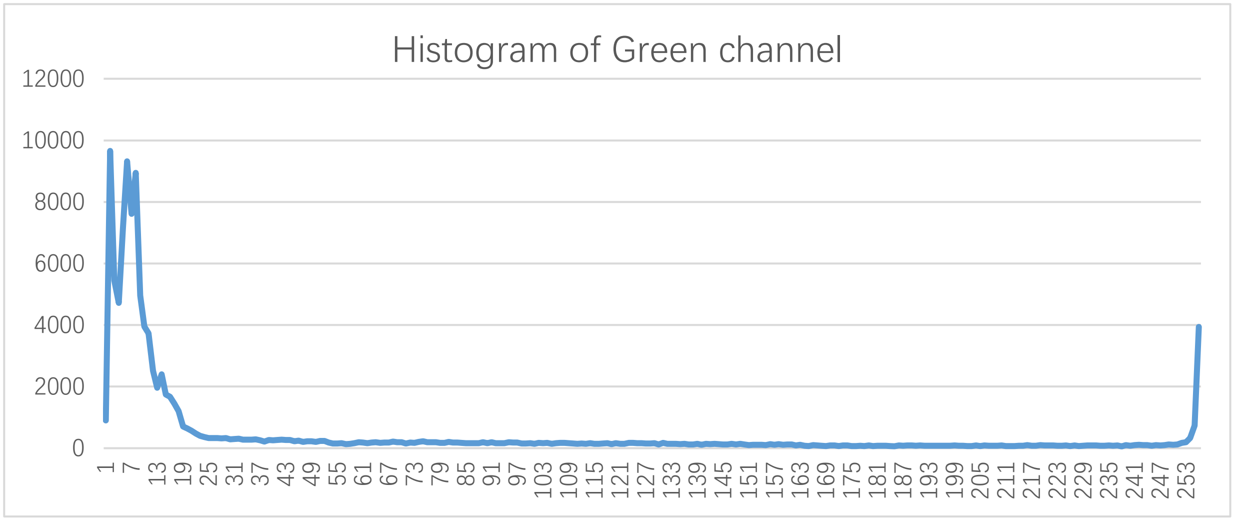


Figure15 Histogram of green channel of the original image

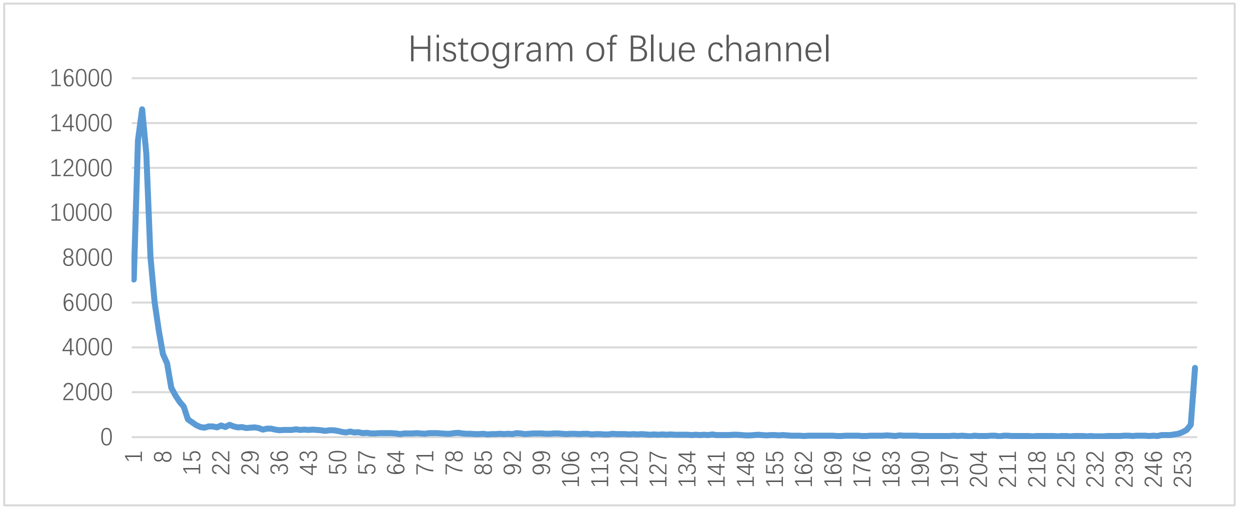


Figure16 Histogram of blue channel of the original image

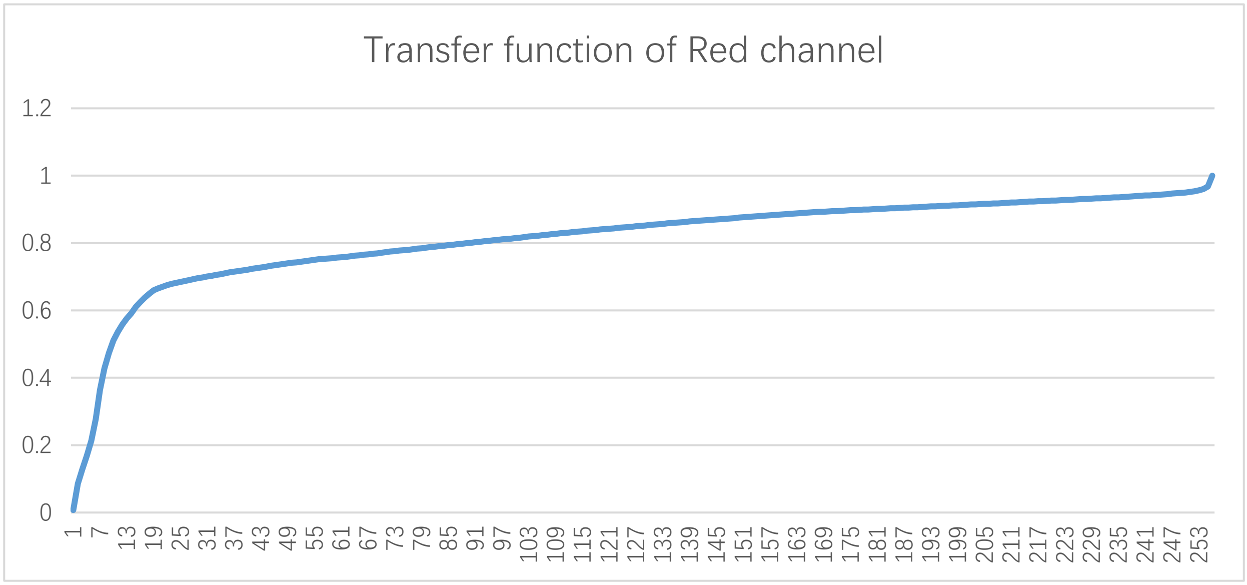


Figure17 Transfer function of red channel

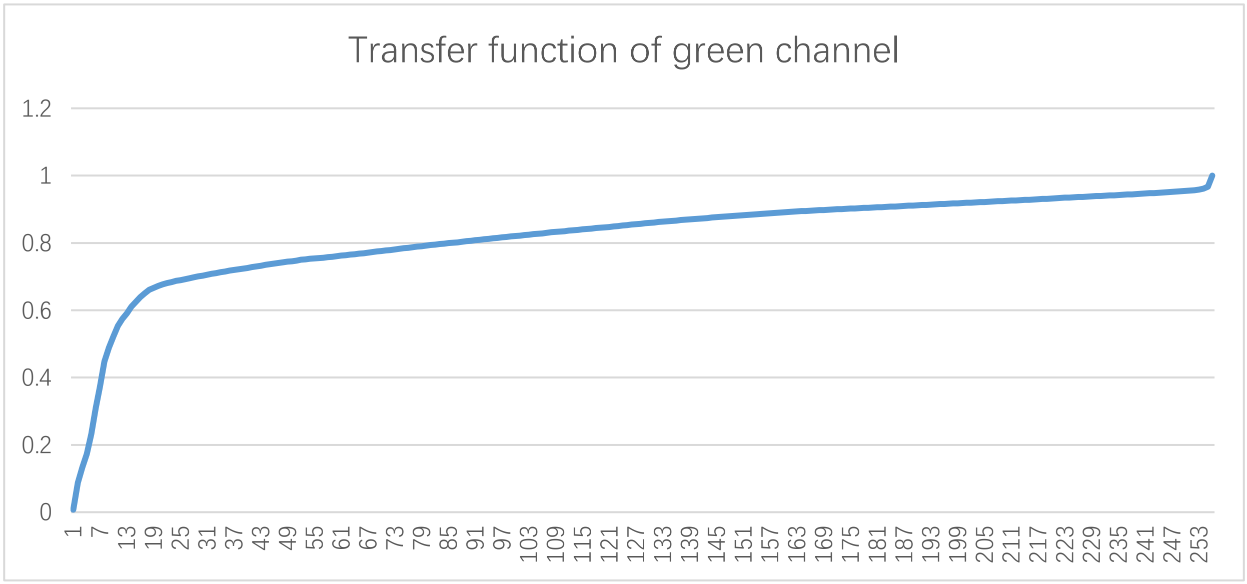


Figure18 Transfer function of green channel

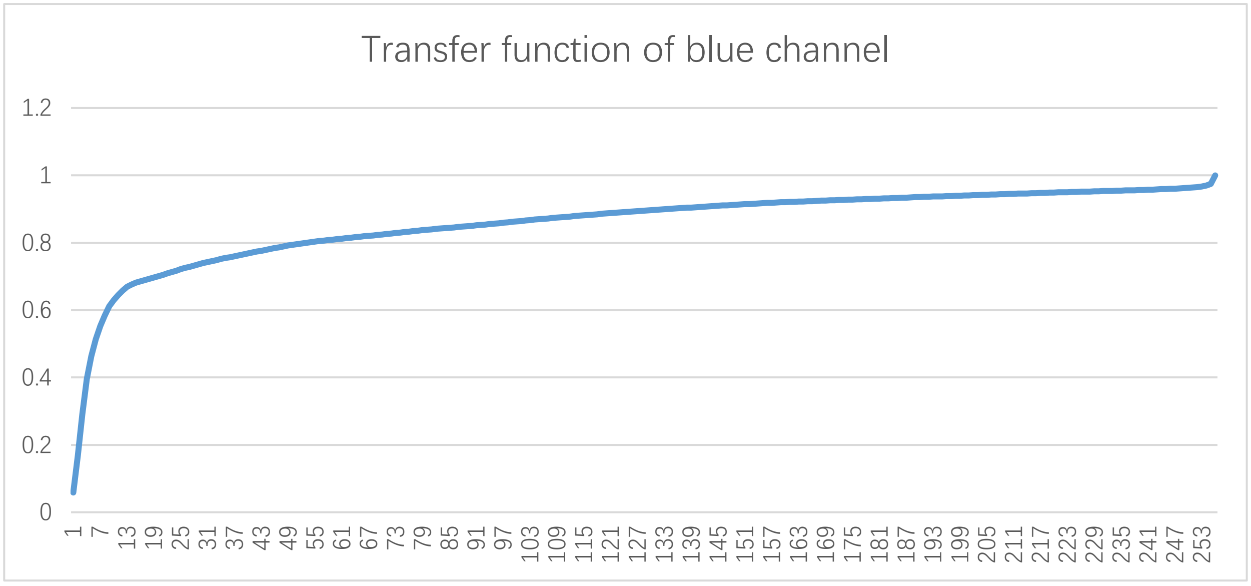


Figure19 Transfer function of blue channel

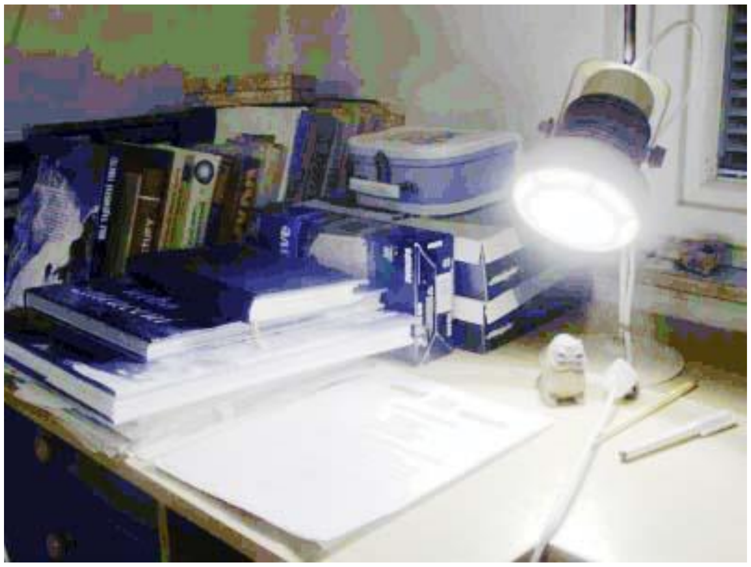


Figure20 Enhanced image by method a.

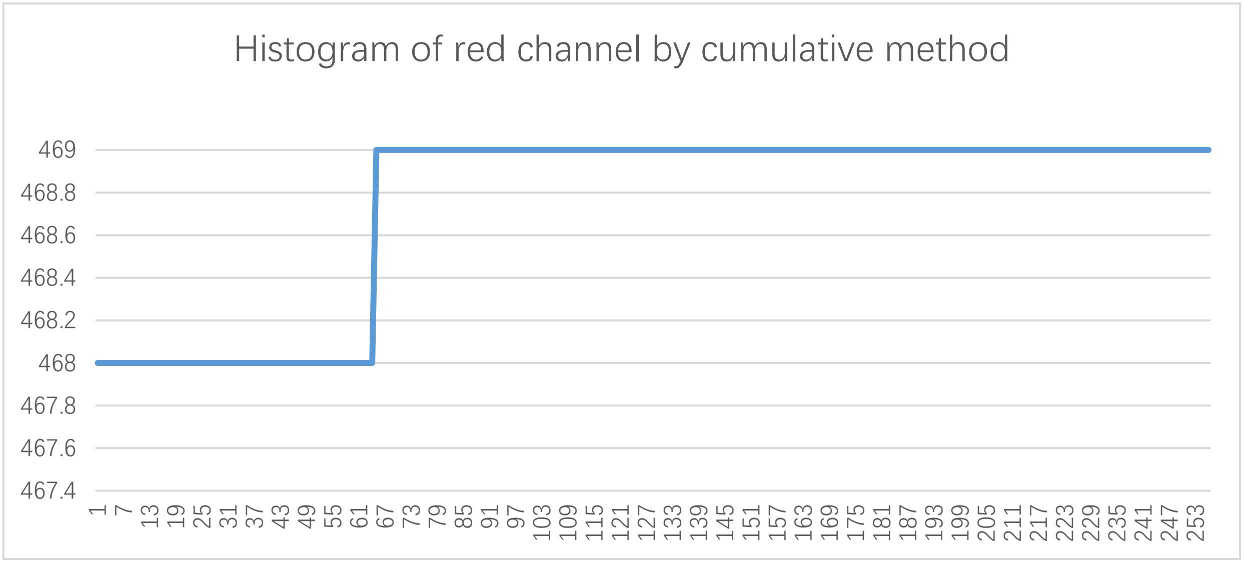


Figure21 Histogram of red channel by cumulative method.

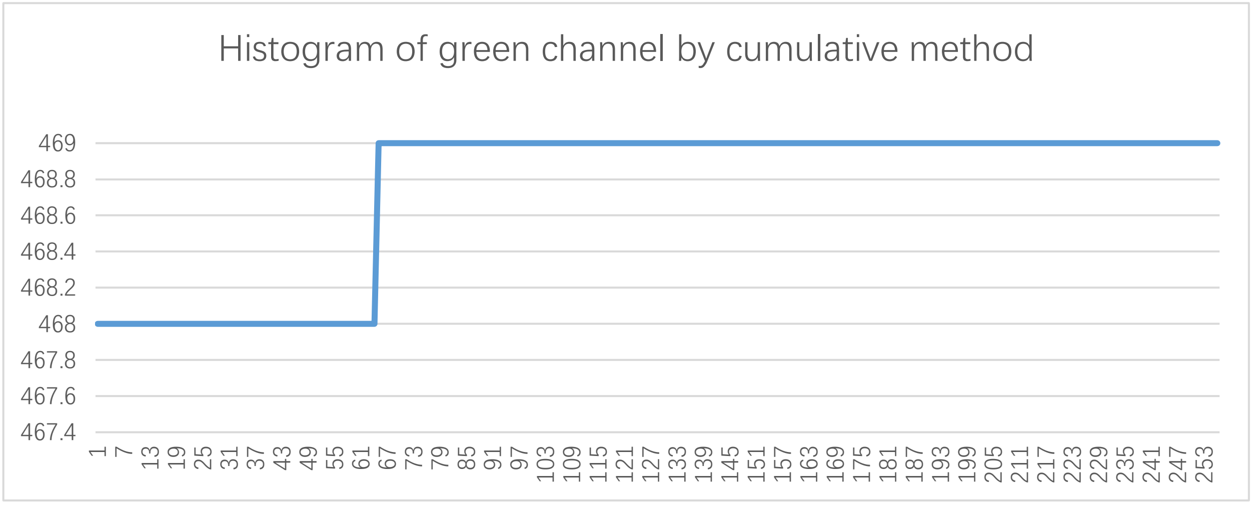


Figure22 Histogram of green channel by cumulative method.

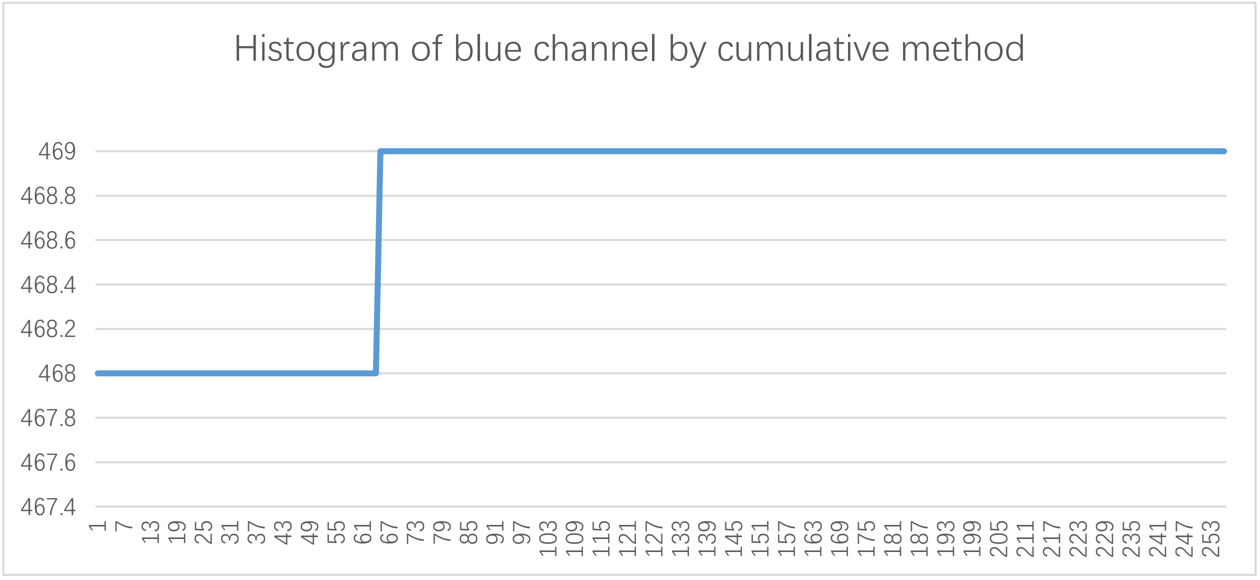


Figure23 Histogram of blue channel by cumulative method.

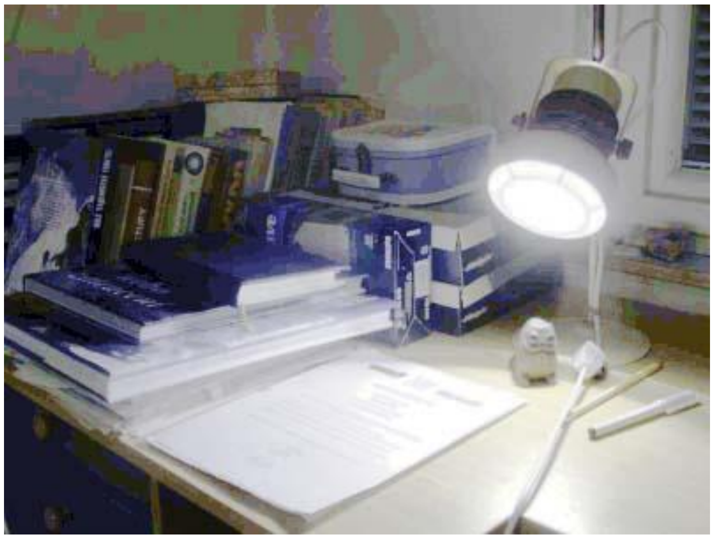


Figure24 Enhanced image by method b.

Discussion:

With the implementation of histogram equalization, we can see some hidden information such as several books and walls in the new image. We can know that the slope of transfer function is very big in first 30 grayscale values. Therefore, there are many pixels with high gray scale values and the equalized image is brighter than the image by method b. However, there are some common disadvantages of these 2 methods. Take the wall as an example, because in the original image, the wall is in the dark and we cannot see anything about it. In the new image, the color of background wall is very strange (upper left area of the image). Maybe we can solve this image by locating the similar area at first. The color of the wall behind the light is white, which should be considered as similarity area of the upper left area. In this way, we can get better quality images.

b. Oil Painting Effect

Abstract and Motivation

After knowing how to do simple histogram equalization, we wonder whether this method can be implemented in order to create some special effect. In our daily life, there are so many apps that can process images and add some special effects to the original image. Therefore, we can use histogram equalization to realize a “oil painting effect”.

Task: Realize an oil painting effect to the original image.

Approaches and Procedures

Theoretical Approach: Quantize all colors of the input color image into an image containing only 64 colors, denoted. In other words, each channel should have only 4 values. We can get 4 values of each channel by dividing total pixels into 4 parts and averaging each part of them. The average value is our desired pixel value. Next, we need to create a neighborhood window for every pixel of reduced color image to find the most frequent value in this window. Finally, assign the most frequent value to the center pixel of this window.

Implementation by C++ (Take Star_Wars.raw as an example):

Step1: Read input image “Star_Wars.raw” and store the image data in an unsigned char vector (1D). Read key parameters from command line including its height and width and number of channels.

Step2: Run two nested loop for getting access to every pix value. The index for outer loop ranges from 0 to (imageheight-1) and the inner one ranges from 0 to (imagewdith-1). Store the pixel info of RGB 3 channels in 3 vectors separately and use the “pair” structure to make pair between pixel value and its corresponding index. Sort all pixel values with respective to red, green and blue channels. Divide the total pixel values into 4 parts and calculate the average value of each part.

Step3: For those pixel in part1, the value of them is equal to the average of part1. So do part2, part3 and part4.

Step4: After step3, we get the 64-color image. Extend the reduced color image according to the neighbor window size N.

Step5: Access to every neighbor window in the extended image and find the most frequent value. Assign it to the center pixel of corresponding neighbor window.

Step6: Loops end and output the processed image by a function called fwrite.

Results:

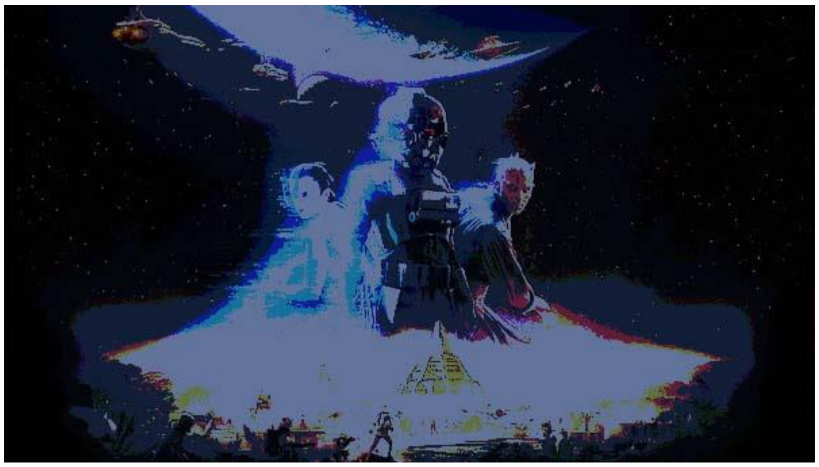


Figure27 64-color version of Star_Wars

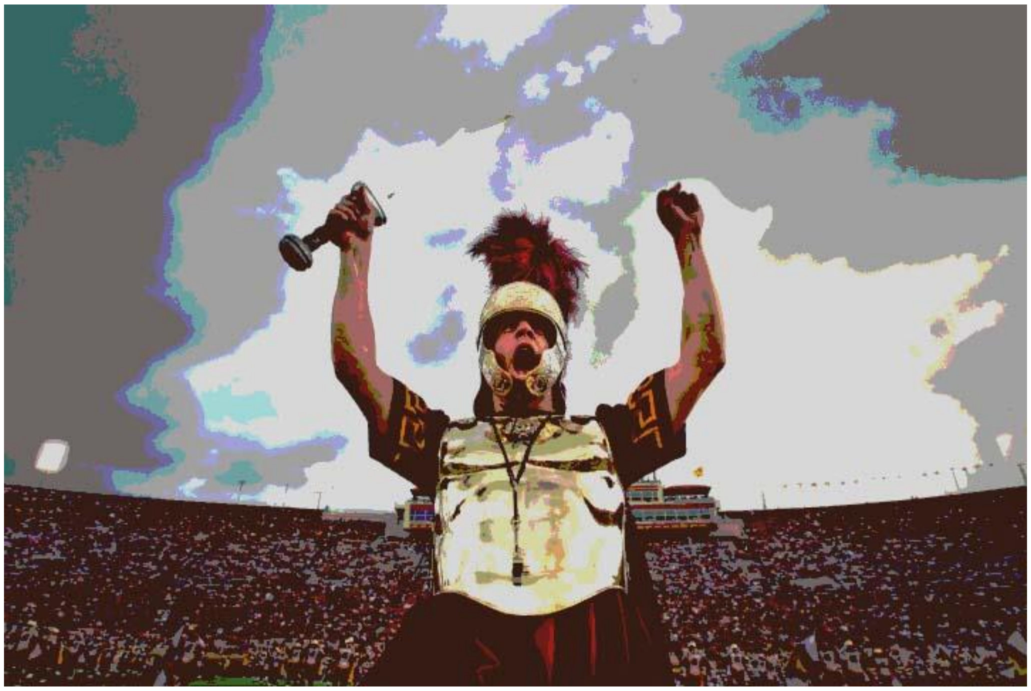


Figure28 64-color version of Trojans

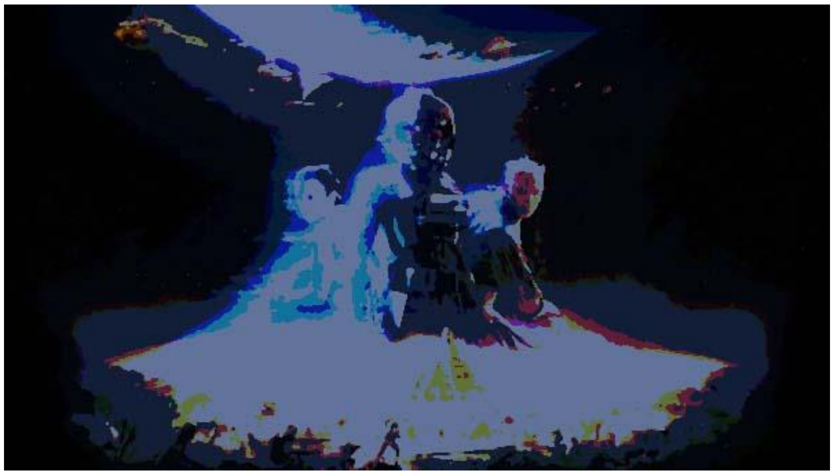


Figure29 N=3 64-color version Star_War

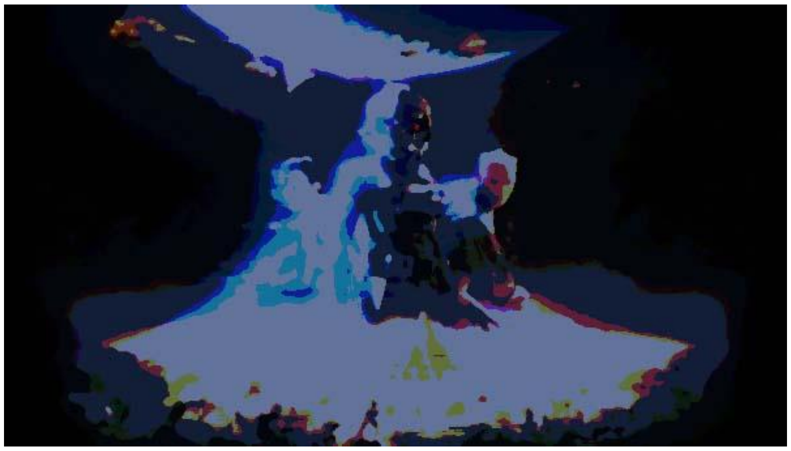


Figure29 N=5 64-color version Star_War

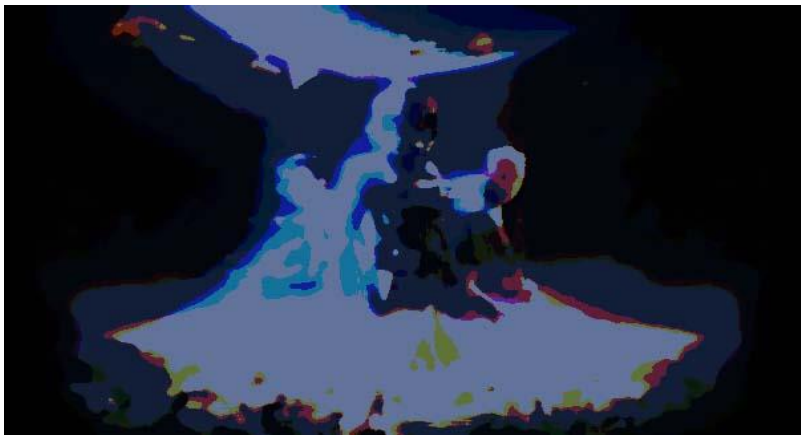


Figure30 N=7 64-color version Star_War

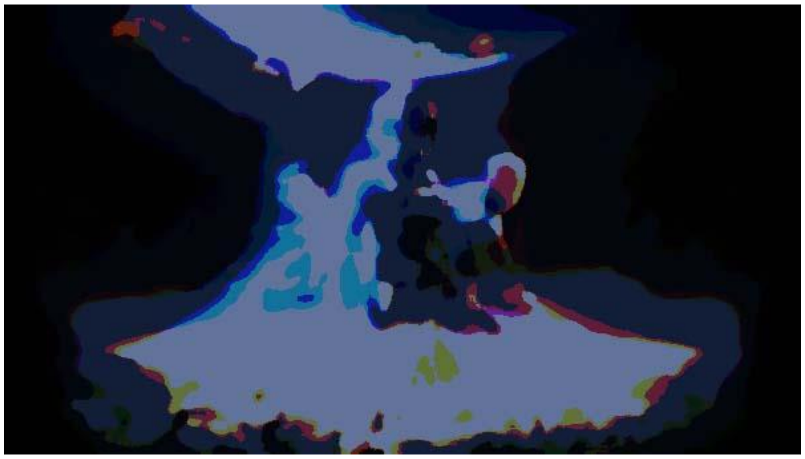


Figure31 N=9 64-color version Star_War

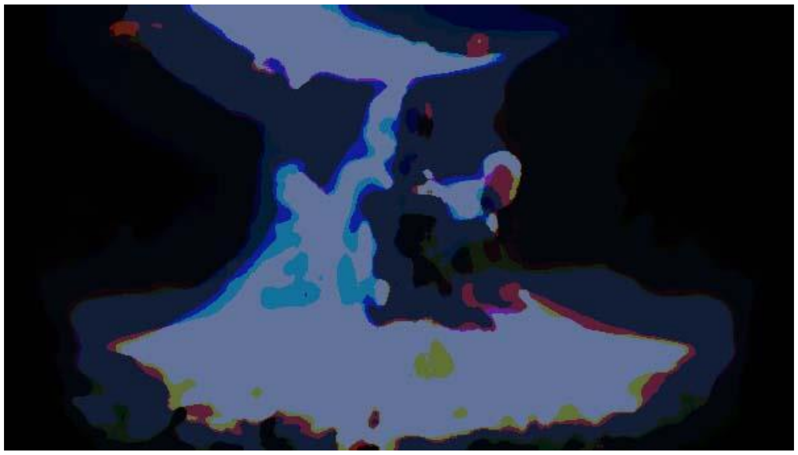


Figure32 N=11 64-color version Star_War

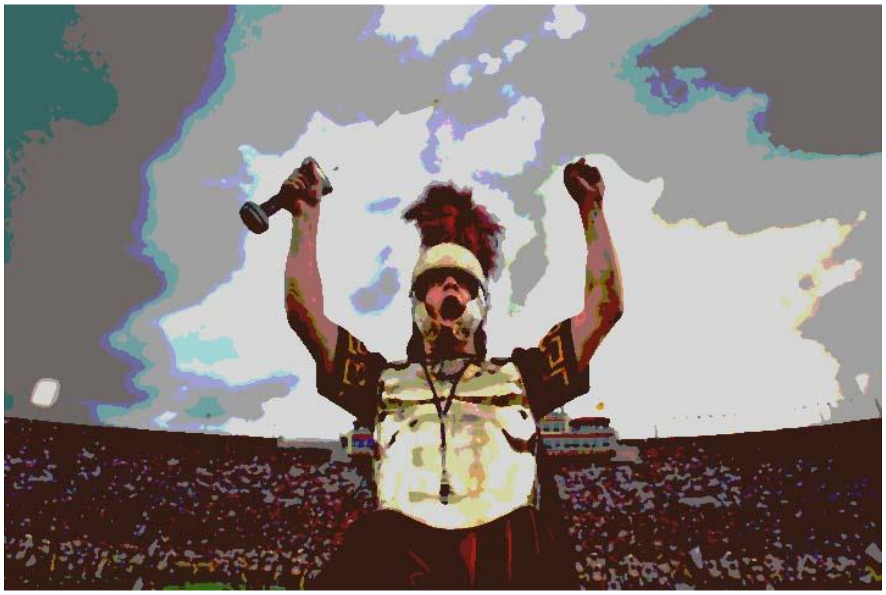


Figure33 N=3 64-color version Trojans

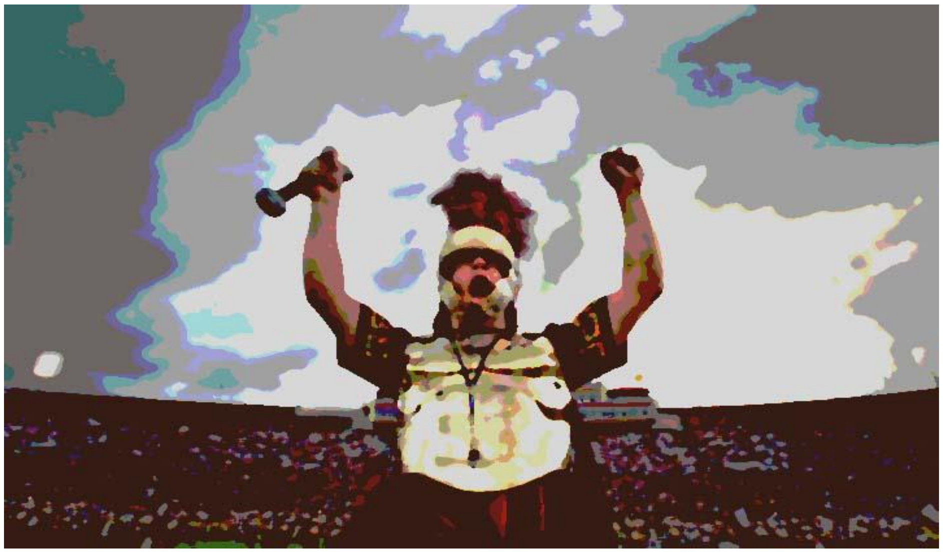


Figure34 N=5 64-color version Trojans

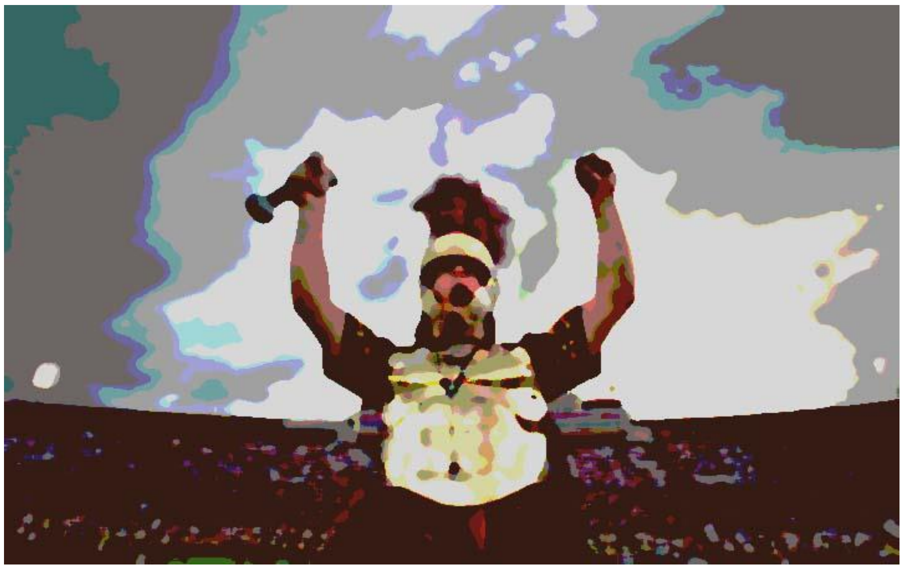


Figure35 N=7 64-color version Trojans

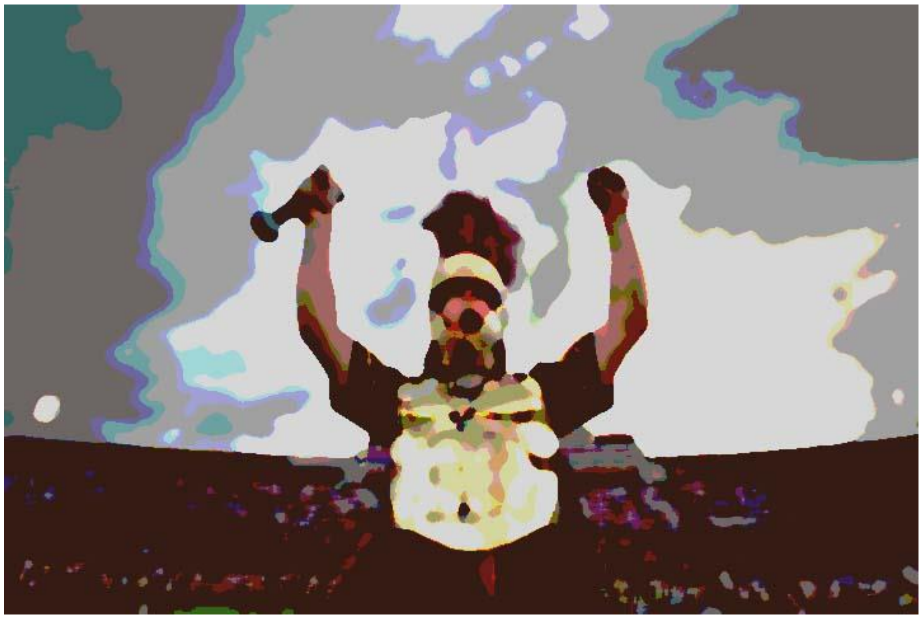


Figure35 N=9 64-color version Trojans

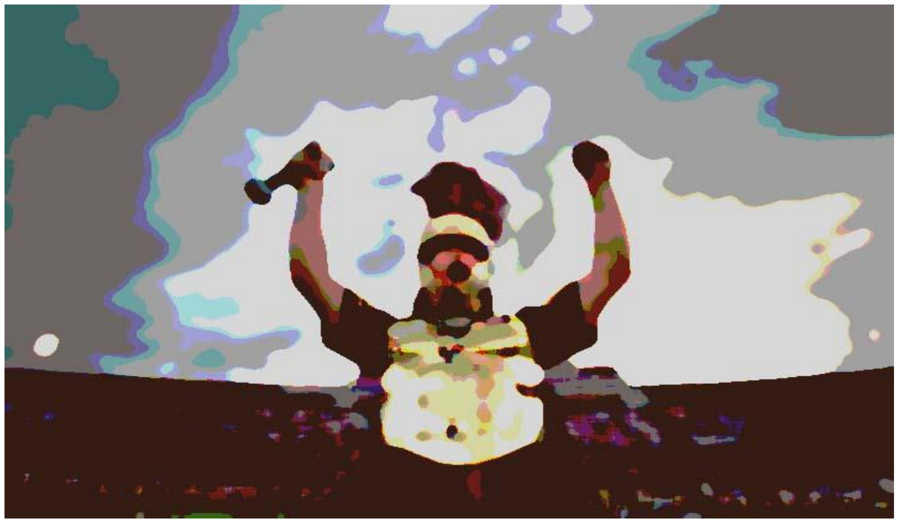


Figure36 N=11 64-color version Trojans

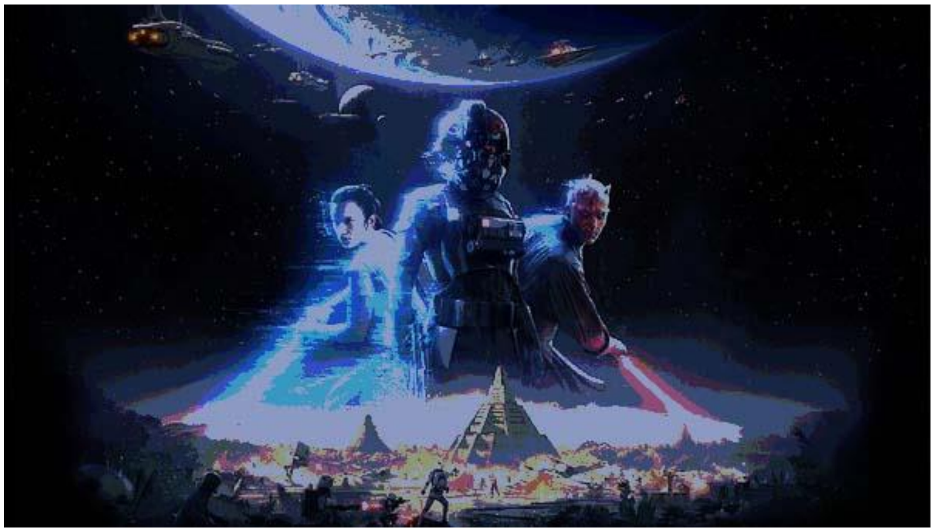


Figure37 512-color version Star_War

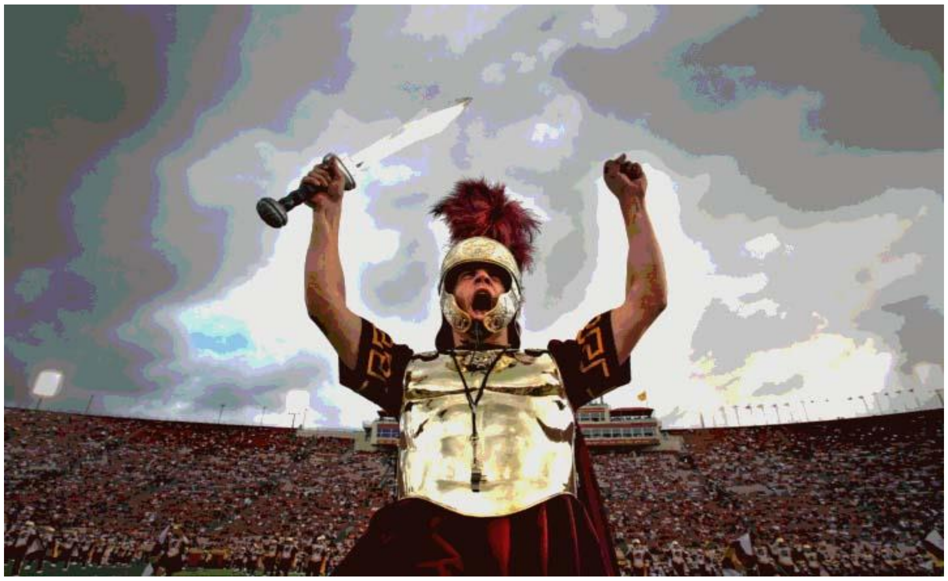


Figure38 512-color version Trojans

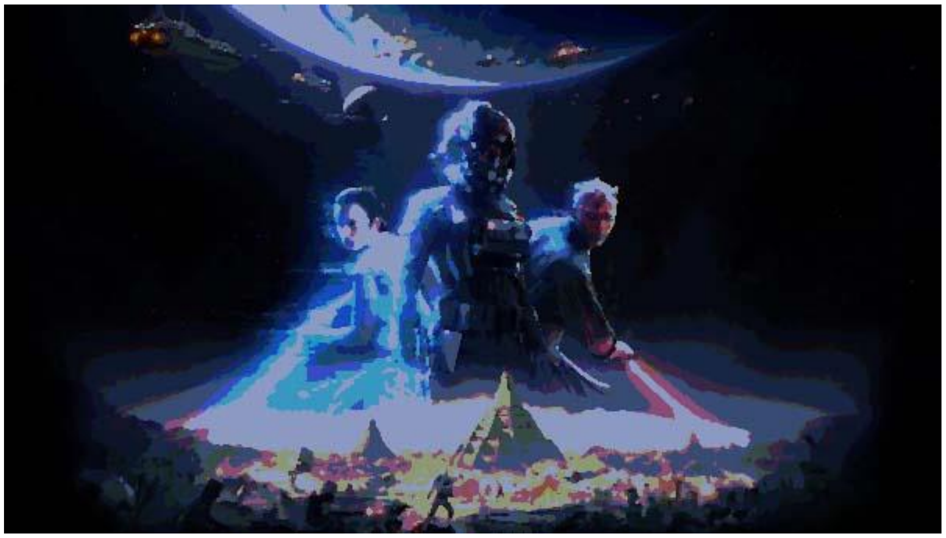


Figure39 N=3 512-color version Star War

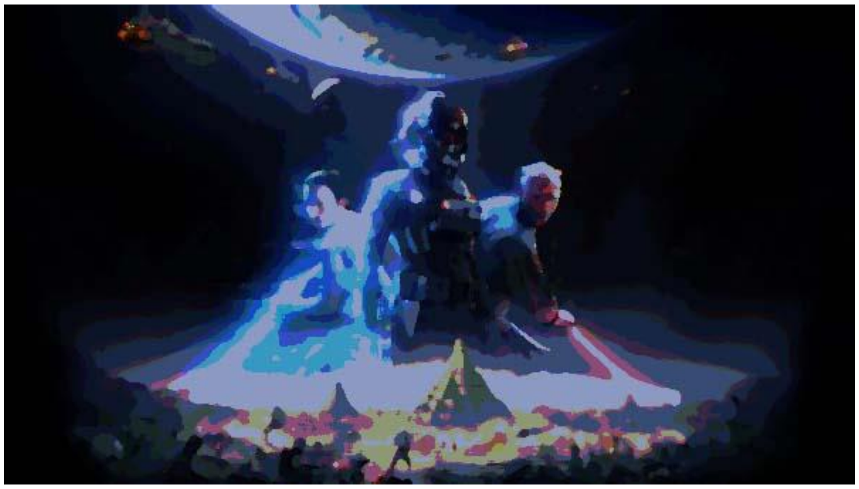


Figure40 N=5 512-color version Star War

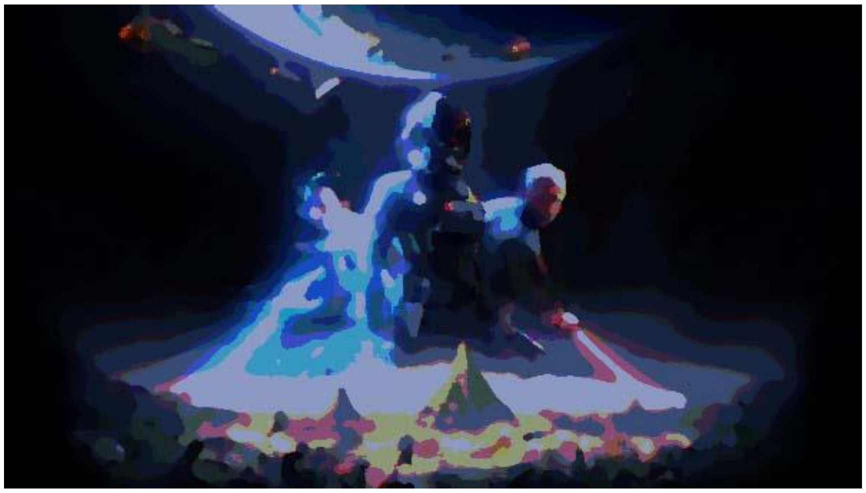


Figure41 N=7 512-color version Star War

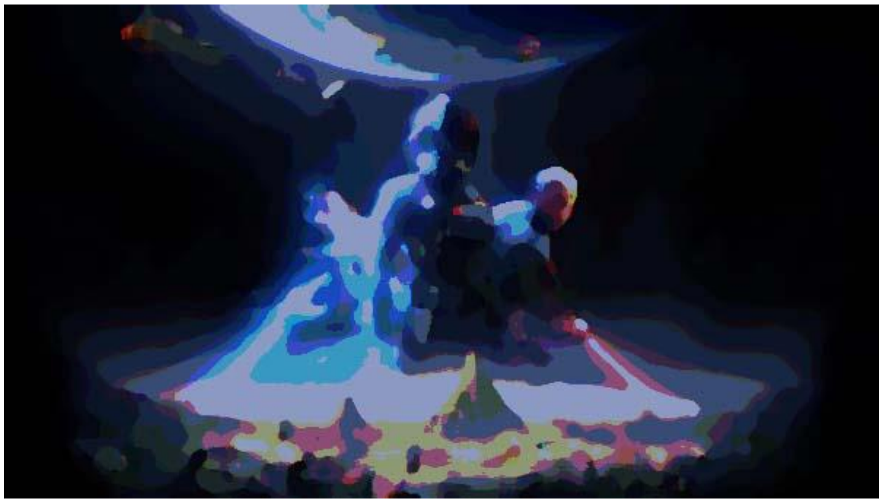


Figure42 N=9 512-color version Star War

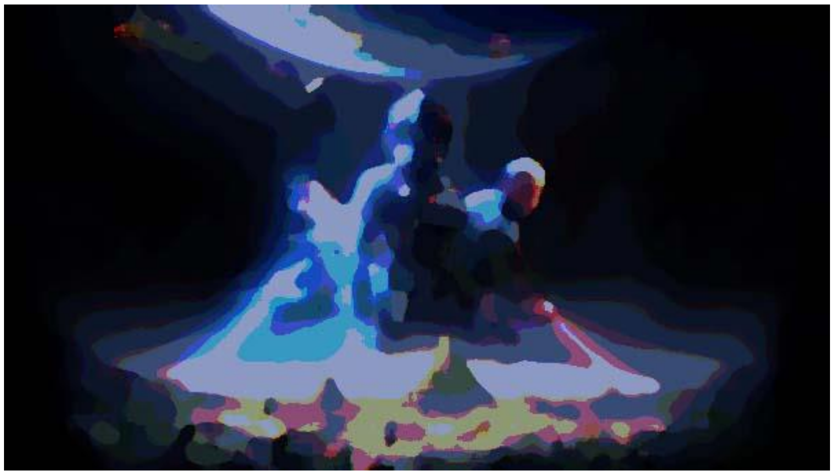


Figure43 N=11 512-color version Star War

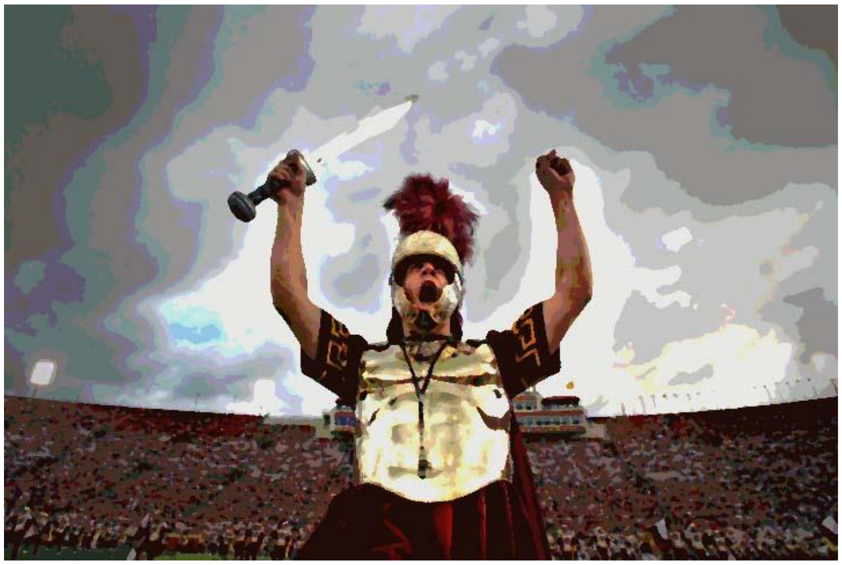


Figure44 N=3 512-color version Trojans

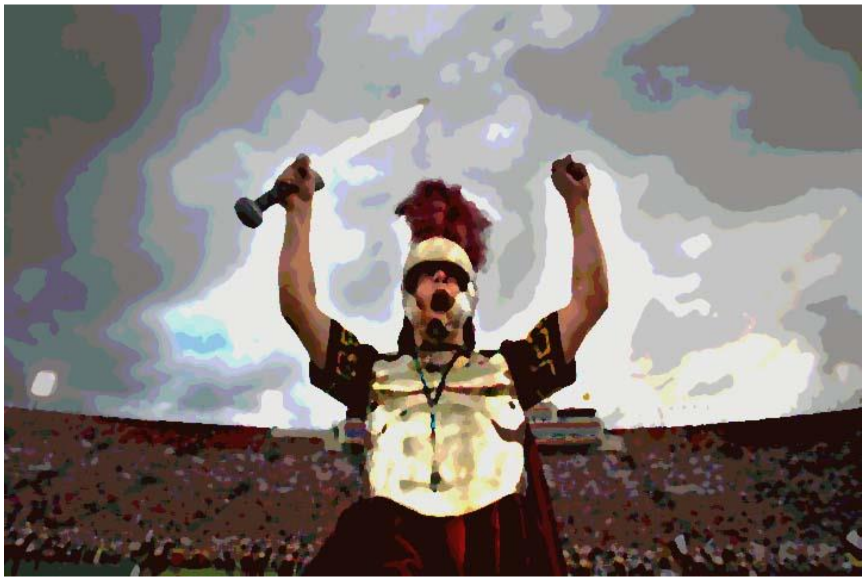


Figure45 N=5 512-color version Trojans

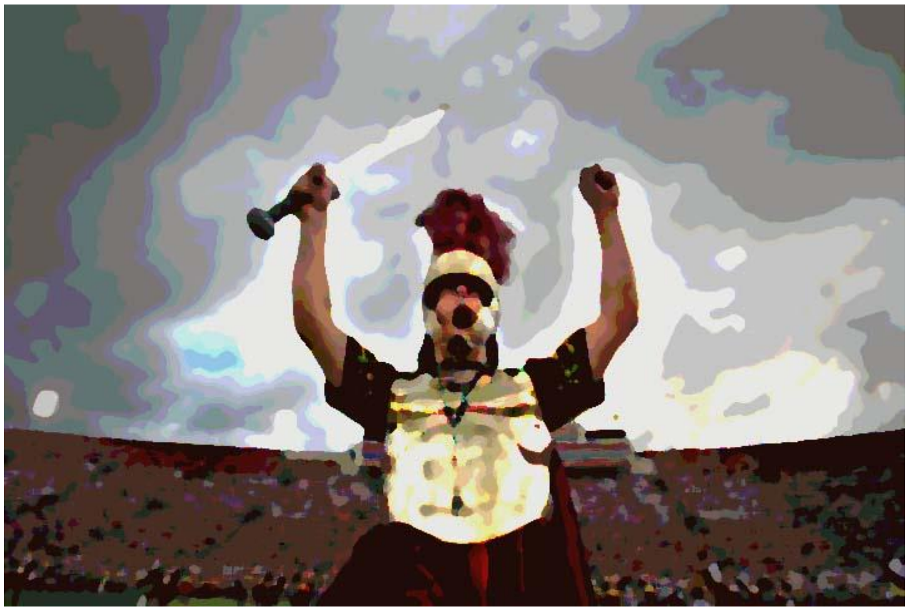


Figure46 N=7 512-color version Trojans

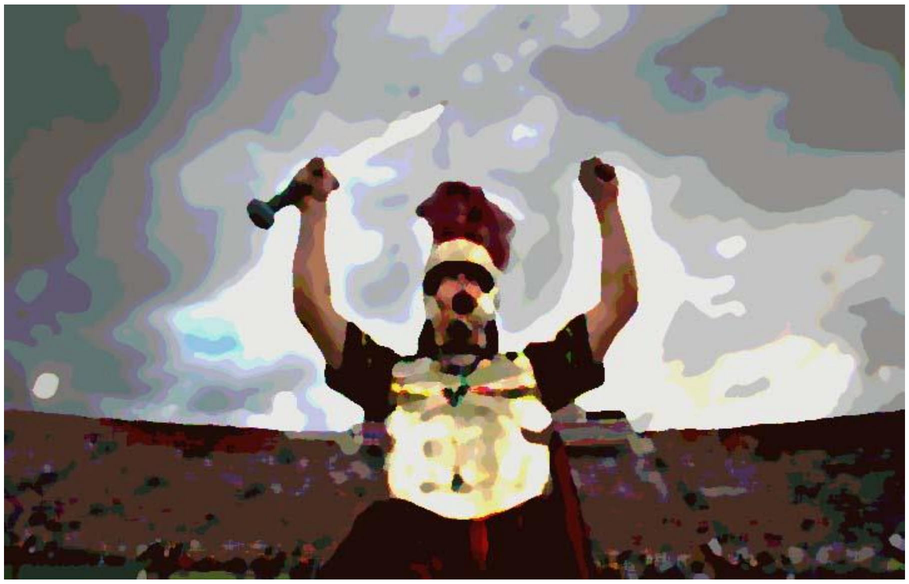


Figure47 N=9 512-color version Trojans

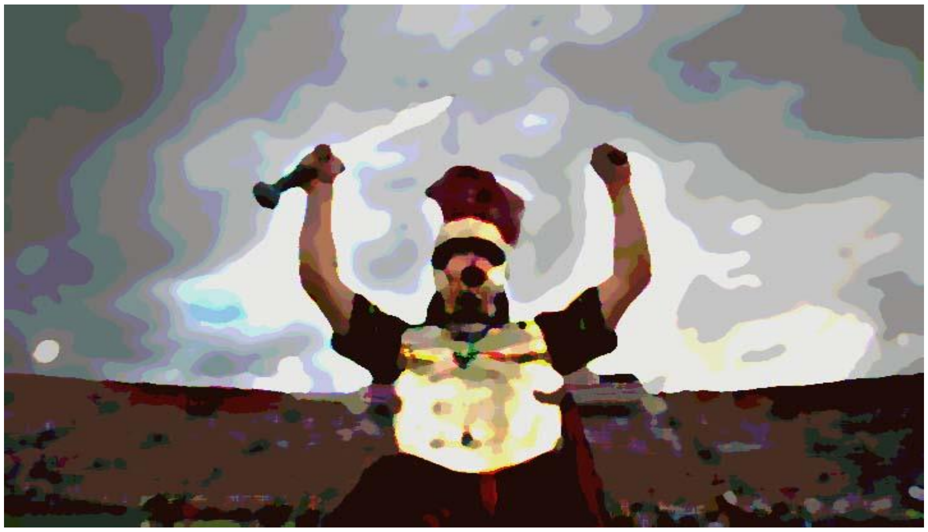


Figure48 N=11 512-color version Trojans

Discussion:

1. The threshold is determined by the order of pixel value. Because I sorted all pixel values of each channel in an ascending order so that if I get the index of the last pixel in each part, I can assign the correct value to the pixel.

2. N=5 is the best. When N gets bigger and bigger, the information and details in the image become less and less. If N is too small, it cannot realize an oil-painting effect. If N is too big, we will lose many details in the image. Therefore, in the range from 3 to 11, N=5 is the best situation from the pictures.

3. When the input image has 512 color, it means each channel has 8 colors. We can see much more details in the 512-color image. Particularly, for images like starwar which background is dark will get better quality if the input has 512 color. So do those pictures like Trojans because its size is very large with a lot of information.

c. Film Effect

Abstract and Motivation

From oil painting effect, we know that some special effects can be created by change the distribution of histogram. In this problem we will implement an algorithm that applies the special effect of the Film image to any original image

Task: Realize a film effect to the original image.

Approaches and Procedures

Theoretical Approach: I realized this effect by a method called "Histogram equalization based on a reference image". First, we have to read 2 images. One is the image you want to process. The other is the reference image. Then, we have to record the histogram of the reference image and calculate its probability mass function for each pixel value by following formula:

$$PMF(N) = \frac{\#(pixelvalue = N)}{\#(Totalpixel)}$$

Next, according to the pmf, we can modify the histogram of the original image. The number of each pixel bin is equal to the product of pmf(n) and total number of pixels in the original image. Finally, we have to sort pixel in an ascending order and assign new pixel value in order to realize this effect.

Implementation by C++:

Step1: Read input image "Girls.raw" and the reference image "film.raw". Store the image data in 2 unsigned char vectors (1D). Read key parameters from command line including its height, width and number of channels.

Step2: Define 3 vectors called red_pmf, green_pmf and blue_pmf. Iterate all pixels in a 2 nested loops and calculate the probability mass function value with respect to each value (0-255).

Step3: This special effect requires us to mirror and reverse the original image. Therefore, we can do these manipulations in a 2 nested loop. Mirror image can be realized by pixel swap. The reverse pixel value is equal to 255-pixel value.

Step4: Define 3 vectors of pair for sorting the pixel values of each channel. The pair data structure can help us to sort the pixel value in an ascending order with its original index in the image.

Step5: Calculate the number of pixel in each grayscale value of each channel by multiplying the pmf by image height and image width. Store them in 3 arrays called r_bins, g_bins and b_bins.

Step6: we can assign the proper grayscale values to sorted pixels in pair vectors according to the number in the r_bins, g_bins and b_bins. If the total number is smaller than the size of the image. Assign 255 to the remaining part pixels.

Step7: we can find every pixels' index in pair data structure and assign the pix value to the corresponding pixel.

Step8: Loops end and output the processed image by a function called fwrite.

Results:

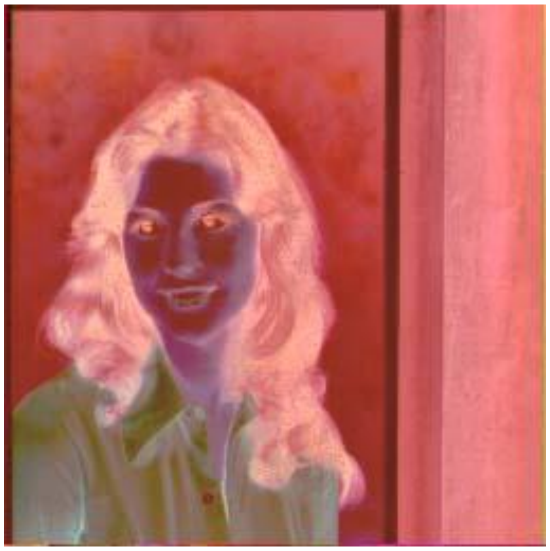


Figure49 Girl_film effect

Problem3-Noise Removal

a. Mix noise in color image.

Abstract and motivation

Noise is very common in our daily life. It will deteriorate the quality of image. In image processing, noise can come from the camera and other electronic devices. Gaussian noise, salt noise and pepper noise belong to common noise. In order to improve the image's quality, noise removal is an important part in image processing. Calculating PSNR is a reliable way to assess the performance of noise removal. Therefore,

Task: use different kinds of filters to remove the mixed noise and evaluate the performance.

Approaches and Procedures

Theoretical Approach: The Gaussian noise is a random noise with zero mean and sigma square variance. According to weak law of big number, when the number of sample is big enough, the sample mean is zero. Therefore, we can use mean filter or Non-local mean filter to delete Gaussian noise because the new pixel data is determined by the average value of its neighbor pixel. Salt and pepper noise belong to impulse noise because its grayscale value is 0 or 255. Thus, we can use median filter to remove this kind of noise since the new pixel value is the median value of its neighbor.

mean filter:

Step1: Read input image and tore the image data in an unsigned char vector (1D). Read key parameters from command line including its height, width, number of channels and window size.

Step2: Extend image according to the window size N. The size of new image is equal to (height+N-1)*(width+N-1).

Step3: Access to every pixel in the original image. Slide the neighbor window. Calculate the average of N*N pixels for each channel.

Step4: Assign the average value to the center pixel(the pixel in the original image).

Step5: Step8: Loops end and output the processed image by a function called fwrite.

median filter:

Step1: Read input image and tore the image data in an unsigned char vector (1D). Read key parameters from command line including its height, width, number of channels and window size.

Step2: Extend image according to the window size N. The size of new image is equal to (height+N-1)*(width+N-1).

Step3: Access to every pixel in the original image. Slide the neighbor window. Record every pixel value of this window and sort them in an ascending order. Find the median value of this window.

Step4: Assign the median value to the center pixel (the pixel in the original image).

Step5: Step8: Loops end and output the processed image by a function called fwrite

non-local mean filter:

Step1: Read input image and tore the image data in an unsigned char vector (1D). Read key parameters from command line including its height, width, number of channels and search window size, neighbor window size and sigma of Gaussian kernel.

Step2: Extend image according to the search window size N. The size of new image is equal to (height+N-1)*(width+N-1). For each search window, find all neighbor window in it.

Step3: For each neighbor window in the search window, calculate its pixel similarity and location similarity. Then, calculate the weight of the center pixel of neighbor window to the center pixel of search window.

Step4: Aggregation pixels and assign the weighted grayscale value to the pixel in the center of search window.

Results

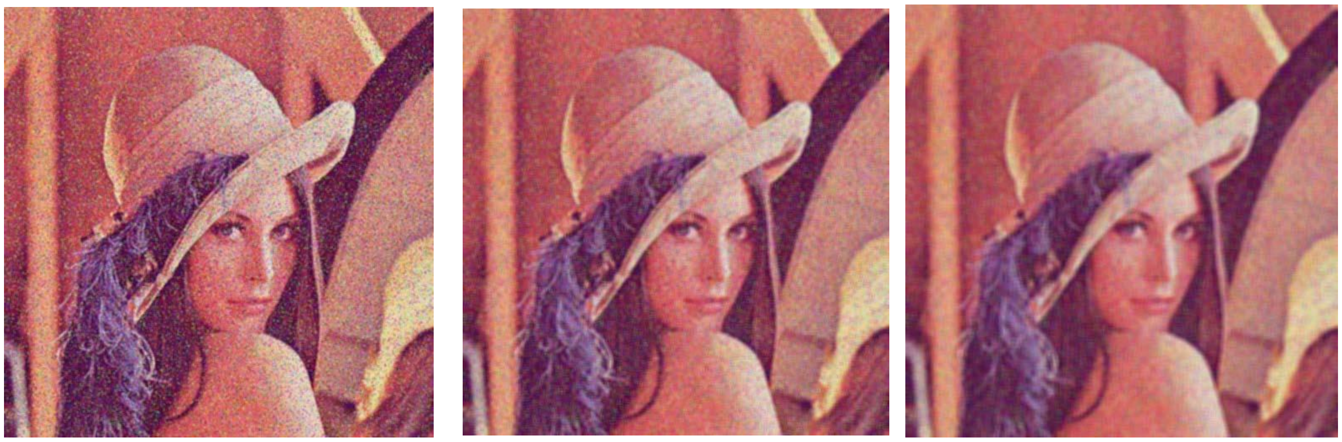


Figure50 Mean filter with window size N= 3,5,7

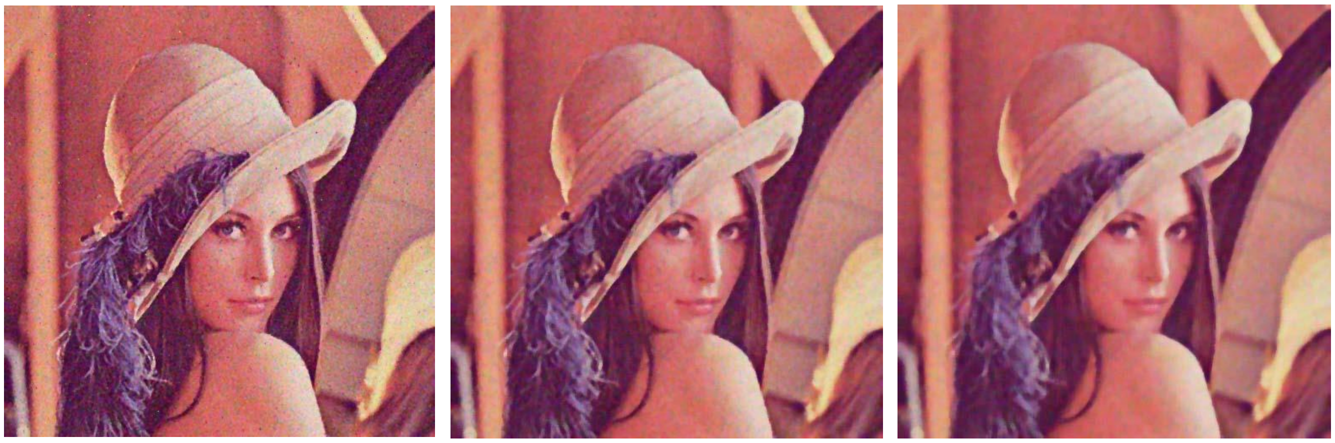


Figure51 Median filter with winodw size N=3,5,7

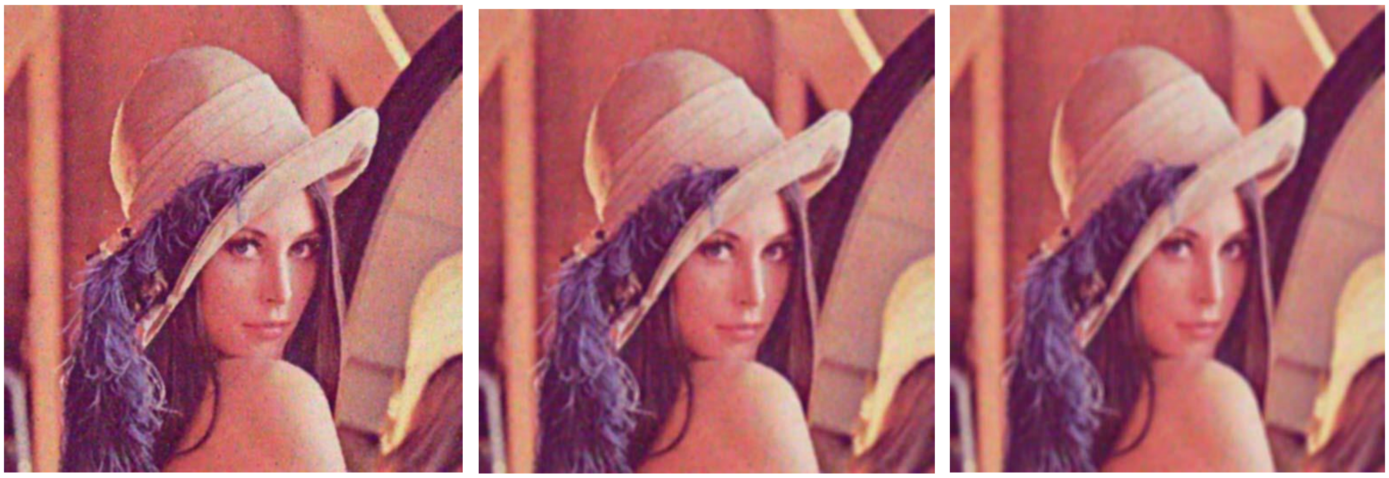


Figure52 Median filter(3*3)+Mean filter(3,5,7)

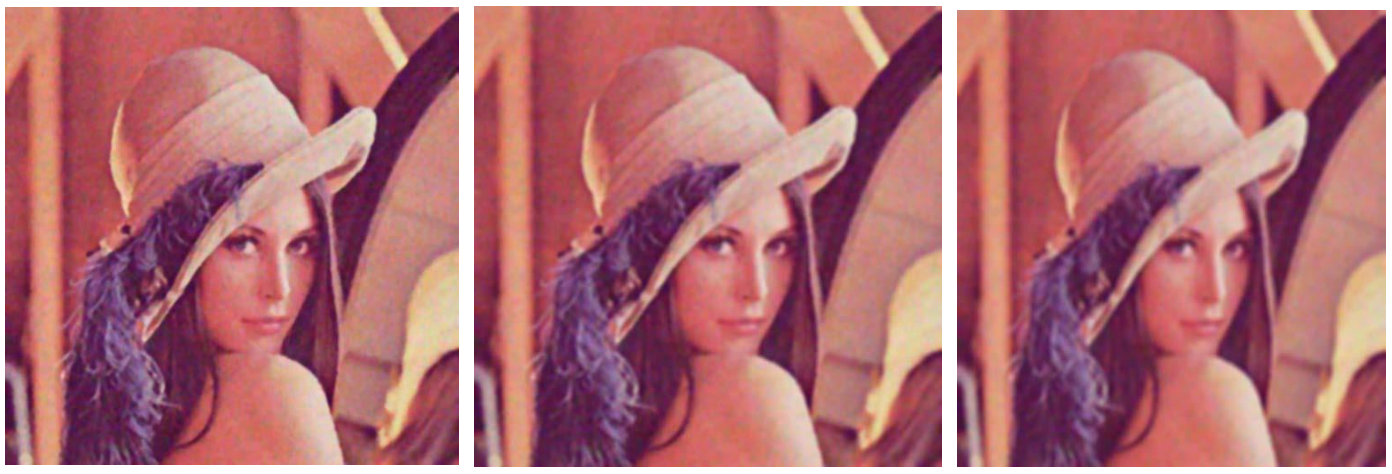


Figure53 Median filter(5*5)+Mean filter(3,5,7)

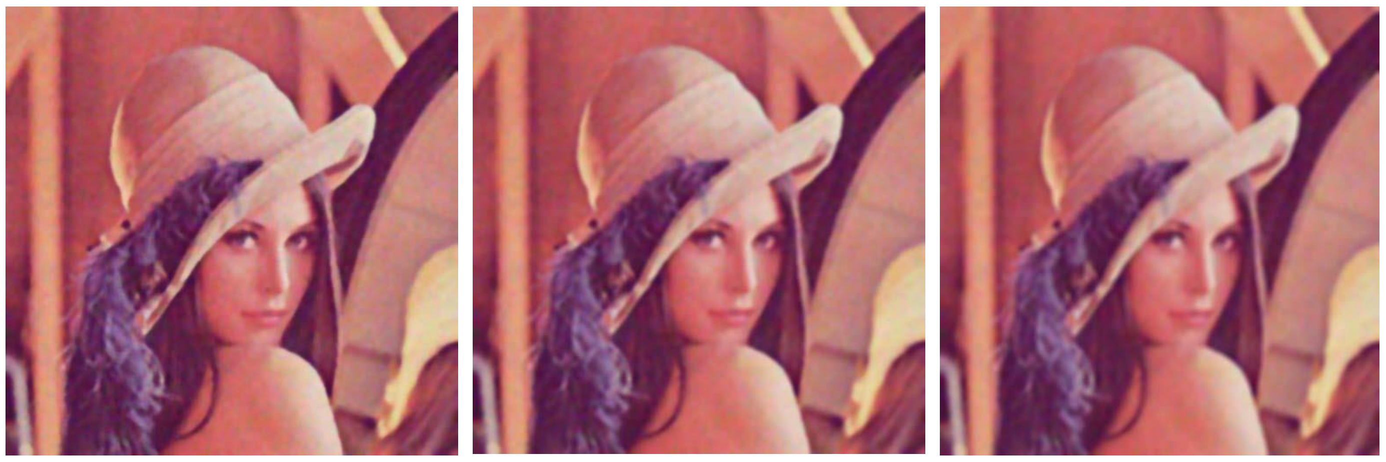


Figure54 Median filter(7*7)+Mean filter(3,5,7)

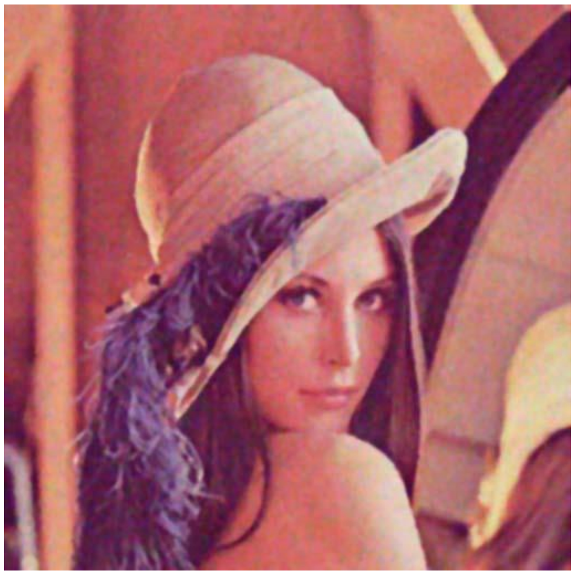


Figure55 median(5*5)+NLM(3,5,25)

Filter	PSNR RED	PSNR GREEN	PSNR BLUE
Mean(3*3)	23.11	22.76	23.88
Mean(5*5)	24.07	23.78	25.31
Mean(7*7)	23.92	23.51	25.41
Median(3*3)	23.94	26.66	23.95
Median(5*5)	25.44	27.11	25.68
Median(7*7)	25.62	26.33	26.09
Median(3*3)+Mean(3*3)	25.23	27.33	25.52
Median(3*3)+Mean(5*5)	25.50	26.45	26.19
Median(3*3)+Mean(7*7)	25.13	25.33	26.13
Median(5*5)+Mean(3*3)	25.80	26.96	26.18
Median(5*5)+Mean(5*5)	25.65	26.18	26.33
Median(5*5)+Mean(7*7)	25.21	25.23	26.15
Median(7*7)+Mean(3*3)	25.71	26.14	26.25
Median(7*7)+Mean(5*5)	25.47	25.61	26.22
Median(7*7)+Mean(7*7)	25.06	24.91	26.03
Mean(5*5)+Median(5*5)	24.21	24.11	25.87
NLM(3,5,25)	22.13	22.46	22.67
Median(5*5)+ NLM(3,5,25)	25.57	27.05	26.23

Table1 Performance of each filter

Discussion

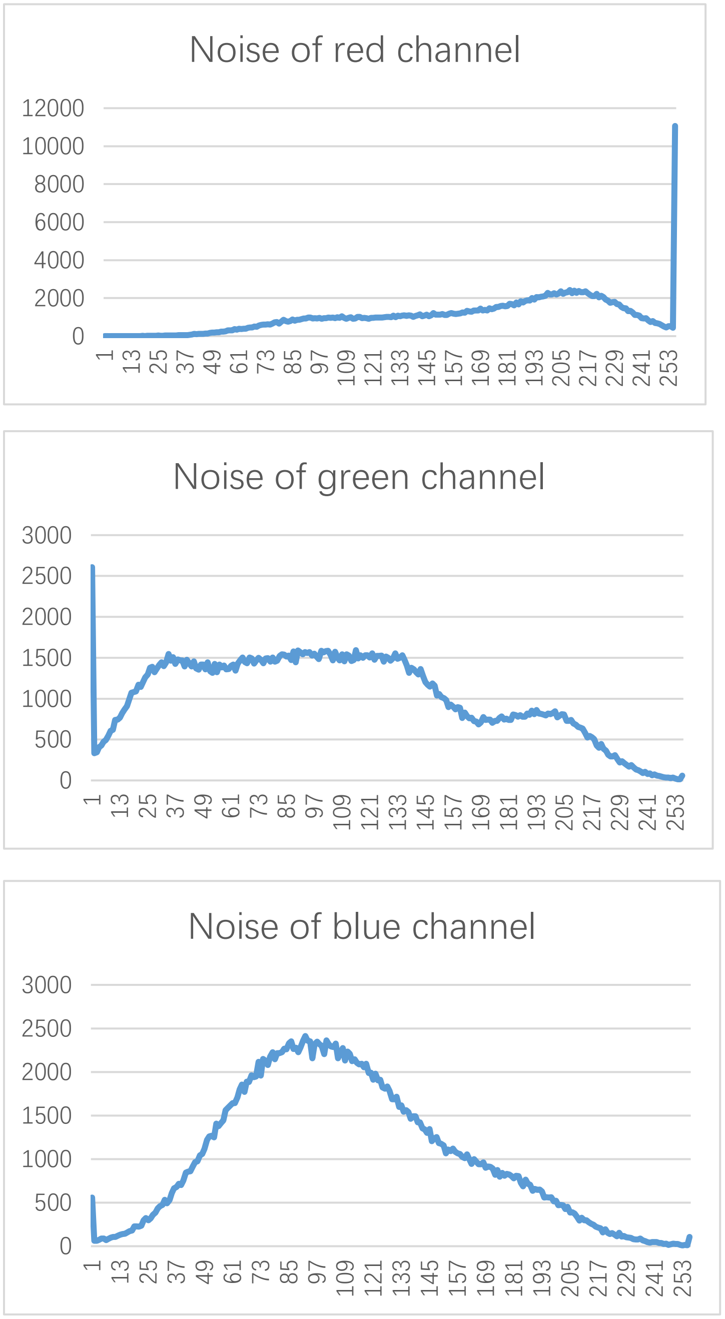


Figure55 histogram of Lena_mixed.raw

(1) According to the histogram of Lena_mixed.raw, the noise distribution of each channel is different. In red channel, the dominant kind of noise is the salt noise because there are over 10000 pixels with 255 grayscale value. Green channel have both Gaussian noise and impulse noise. For blue channel, the major noise is Gaussian noise.

(2) I should perform filtering on individual channels separately.

(3) I utilize median filter to remove impulse noise (salt and pepper noise). The Gaussian noise is removed by mean and non-local mean filter.

(4) The only order I can cascade these filter is that the first one is median filter and the second one is mean or NLM. You can get the evidence from Table1. If I reverse the order, my performance is worse than the performance with a single filter.

(5) With the size of window increasing, the output image becomes more and more blur. Small size of window can keep some details of the original image.

(6) The best combination is the median filter with size=5 and the non-local mean filter with 3*3 neighbor size, 5*5 search window size and 25 sigma. The PSNR for RGB channels are 25.57, 27.05 and 26.23 db.

(7) NLM is a computational complex algorithm because there is a 7-nested loop in it.

(8) In 2010, some scientists invented the fast-NLM. It will faster its speed. Also, we can find some alternatives for the loop in order to get more efficient implementation.

b. Principle component analysis

Abstract and motivation(ANS-1)

Principle Component Analysis is a method to analyze the relation of several features. It can be used in dimensional reduction of data. In the lecture, professor Kuo told us that this process is like a basis change. After a basis change, the new orthogonal basis transfer original data to an uncorrelated column vector. The eigenvalue is the variance of each column vector. The component of a noisy image is the original pixel value part and the noise part. The signal part of an image is dominant and the noise's variance is quite small. Therefore, with PCA, we can distinguish the signal part from the mixed data finishing the denoising task. The components contain signal and noise. In many cases, we choose the components by checking the ratio of signal variance and noise variance. The empirical value of this ratio is from 2.5 to 3.

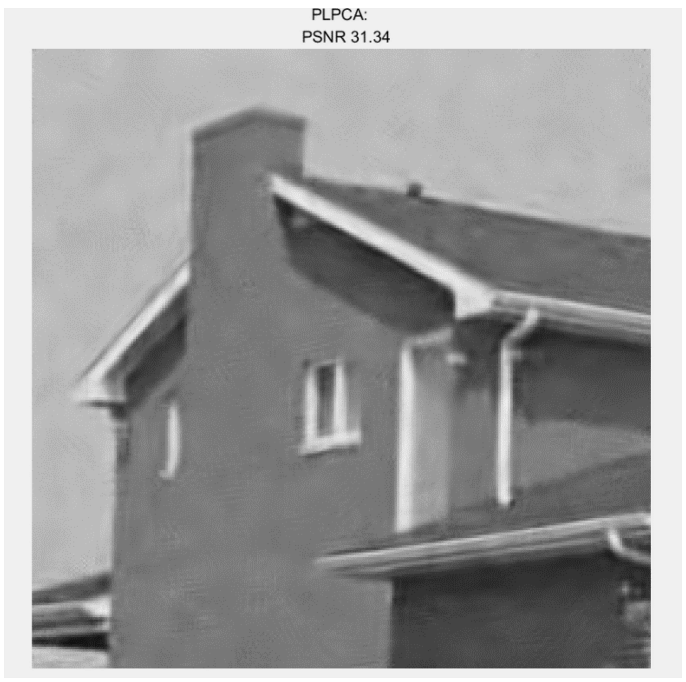
Approaches and Procedures(ANS-2)

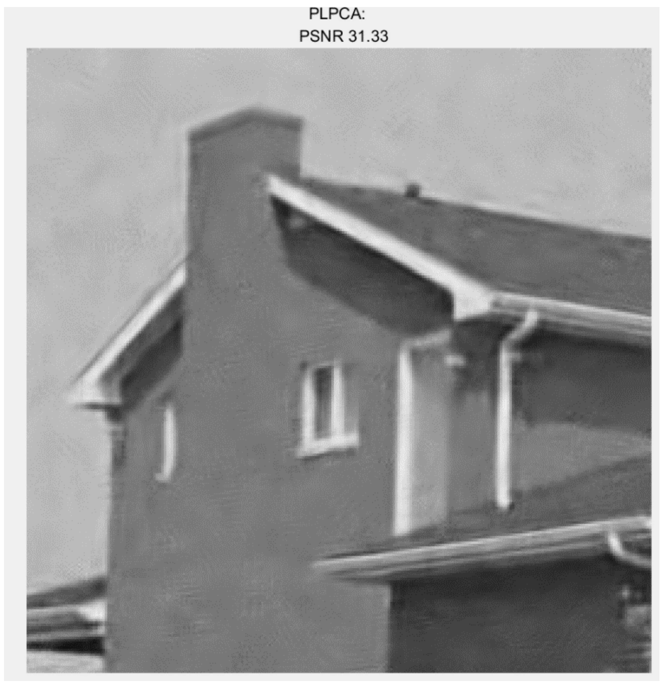
The procedures for PLPCA is not complicated. First, we should decide a proper patch size (Wp) and search window size (Ws) for the local-patch-based PCA. After choosing these two values, we can calculate the number of patches in the original image and record total pixel value in every patch . Next, because our PCA is done in every search window and we should take the overlap into consideration, it is necessary to assign a step for the sliding search window. Therefore, we can get the number of all search window and its position index. Then, in order to proceed PCA, we should stretch the patch pixel data in to a column vector and do PCA in every batch. Next, we should make a summation of each patch in the search window by following formula:

$$\bar{Y} = \frac{1}{M}\sum_{k=1}^{M} Y_k \qquad\qquad Y_{Si} = \frac{1}{M}\sum_{k=1}^{M} Y_k Y_k^{'} - Y\bar{Y}$$

Y_k is the patch elements in original image. M is the search-window size. Then, we can do the PCA and find its corresponding eigenvectors. The basic estimation is the summation of mean vector Y bar and the summation of the projection of each patch vector. Due to different eigenvalue, the weight of projection to each eigenvector is also different so that we can use the hard thresholding to correct the weight. Finally, we have to solve the problem of overlap. Due to the length of step, we cannot avoid overlapping. Therefore, we must create a matrix to record the times of overlap of each pixel. Then, we can aggregate all search window averaging them according to its overlapping times.

Results





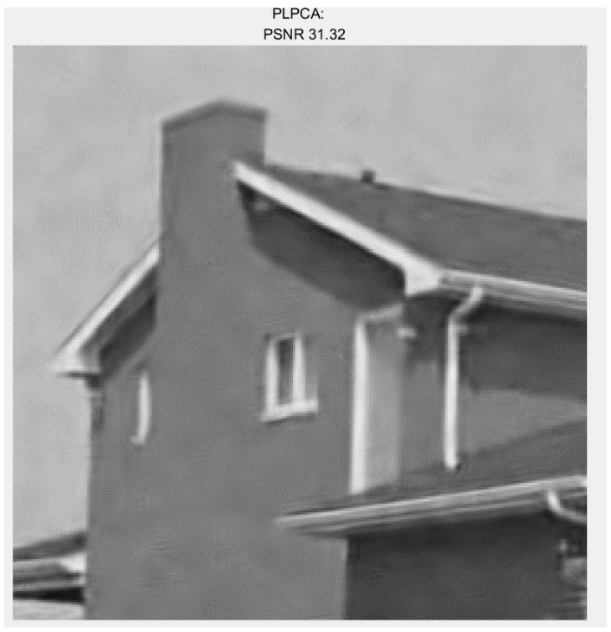
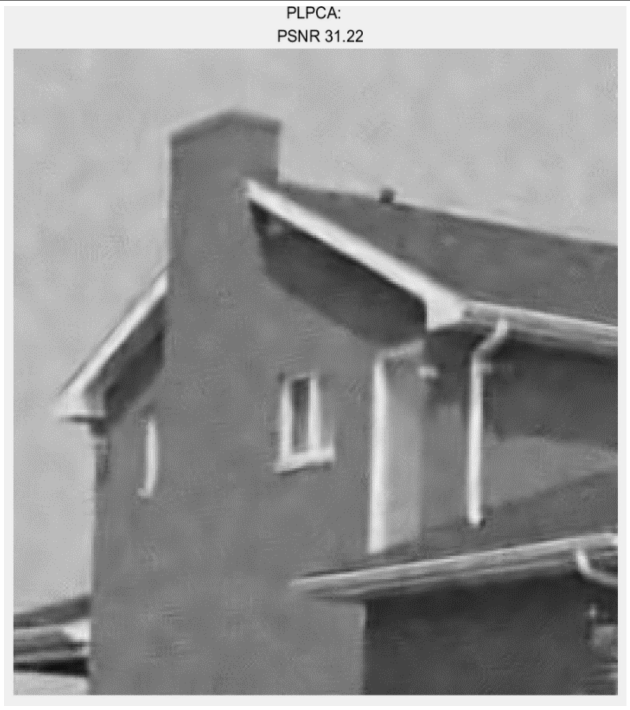


Figure56 Different factor of hard-thresholding



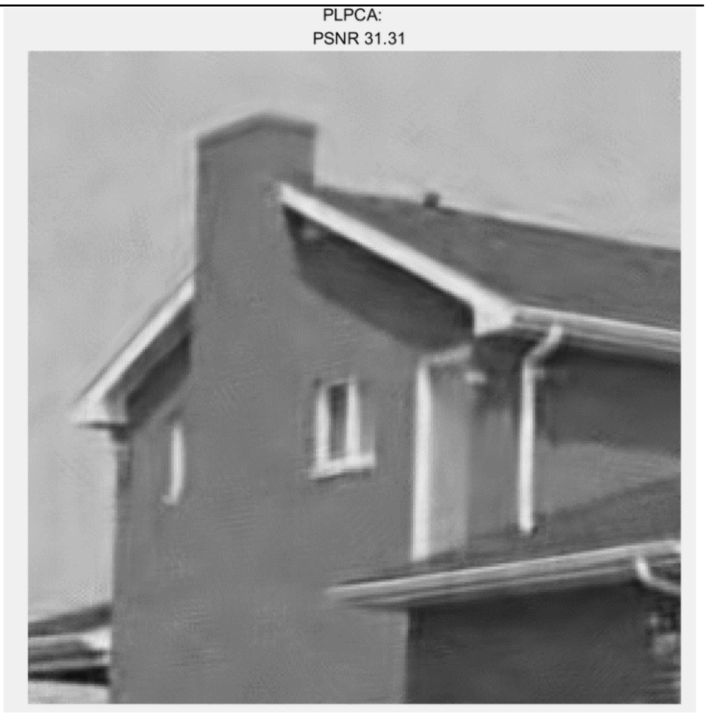
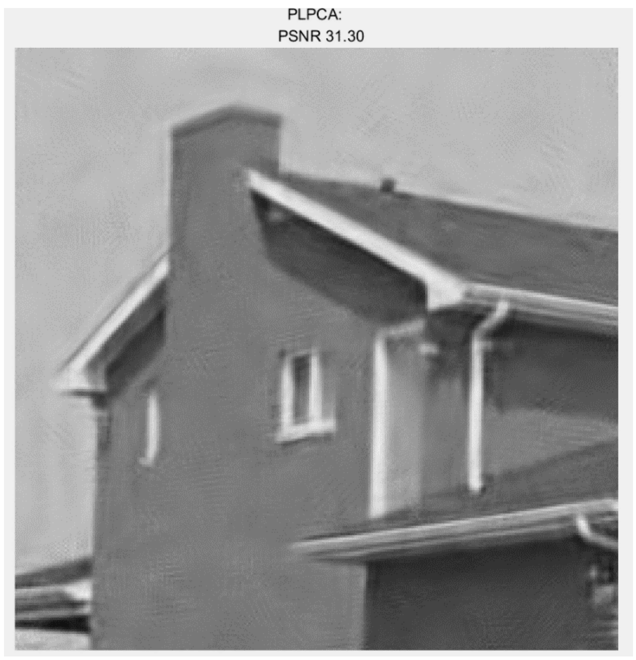


Figure 57 Different patch size (7,11)



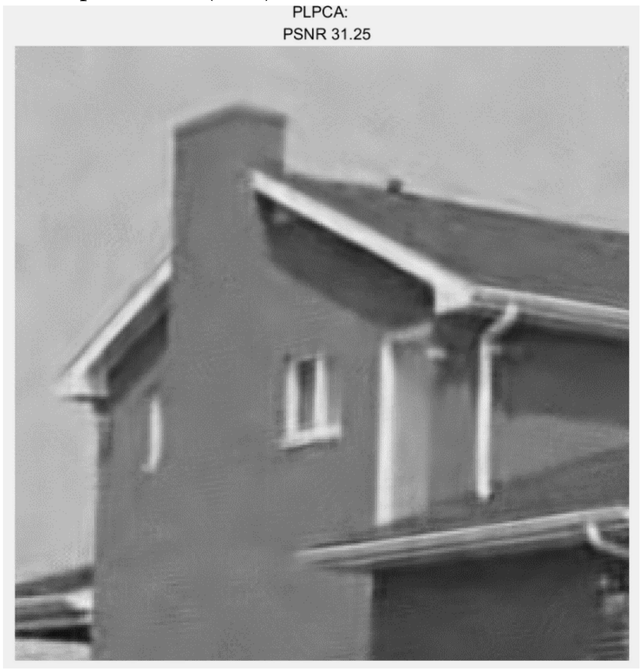


Figure58 Different search window size (15,31)

The results are the output from the program[1].

Discussion:

Parameters	PSNR
Patch Size=9 half search size=23 factor=2.85(best combination)	31.34
Patch Size=9 half search size=23 factor=2.75	31.33
Patch Size=9 half search size=23 factor=2.95	31.32
Patch Size=7 half search size=23 factor=2.85	31.22
Patch Size=11 half search size=23 factor=2.85	31.31
Patch Size=9 half search size=15 factor=2.85	31.30
Patch Size=9 half search size=31 factor=2.85	31.25
Mean filter 5*5	27.23
Non local mean filter(3,5,25)	27.75

Table2 performance of PLPCA

(1) The parameters that we can change include patch size, factor of hard-thresholding and the search window size. If we increase the factor of hard-thresholding, some minor part of signal will be counted since the way we calculate the weight is hard-thresholding. If the search size is too small, the overlap will increase but if it is too big, some principle component would be removed.
(2) The performance of PLPCA is much better than the traditional filter because it is patch based and PCA will help us to get the most variance direction, which represents the signal most. Those traditional filters may remove the signal and keep the noise since the decision rule is very simple and noise is often a small portion.

c. Block matching and 3-D transform filter

Abstract and motivation

[2]The enhancement of the sparsity is achieved by grouping similar 2D image fragments (e.g. blocks) into 3D data arrays which we call "groups". Collaborative filtering is a special procedure developed to deal with these 3D groups. We realize it using the three successive steps: 3D transformation of 3D group, shrinkage of transform spectrum, and inverse 3D transformation. The result is a 3D estimate that consists of the jointly filtered grouped image blocks. By attenuating the noise, the collaborative filtering reveals even the finest details shared by grouped blocks and at the same time it preserves the essential unique features of each individual block. The filtered blocks are then returned to their original positions. Because these blocks are overlapping, for each pixel we obtain many different estimates which need to be combined. Aggregation is a particular averaging procedure which is exploited to take advantage of this redundancy. A significant improvement is obtained by a specially developed collaborative Wiener filtering.

Approaches and Procedures(ANS-1)

This algorithm can be divided into 2 sections: Basic estimate and Final estimate.

Step1: We have to do the grouping. According to the thesis, the concept of grouping is turn the d-dimensional signal into (d+1) dimensional. Image signal is a 2 dimensional signal. Therefore, our purpose is to have 3-dimensional image signals. This algorithms is processing image in blocks so that we have to select similar blocks to the processing one by comparing their Euclidean distance. After having similar blocks, we can stack them together and get a 3-D array.

Step2: Apply some 3-D transform, such as DCT, in frequency domain to the group we got in step1.

Step3: Apply hard-threshoding to the transformed data. From previous PCA problem, we know that half-thresholding is to make the coefficient some small magnitude signal to 0. As we all know, random noise spread widely but its magnitude is quite small so that we can remove most part of the noise. Recording the weights of hard-threshoding and reconstruct the basic estimate by the different weights.

Step4: Apply Inverse 3D transform to get the basic estimate (block).

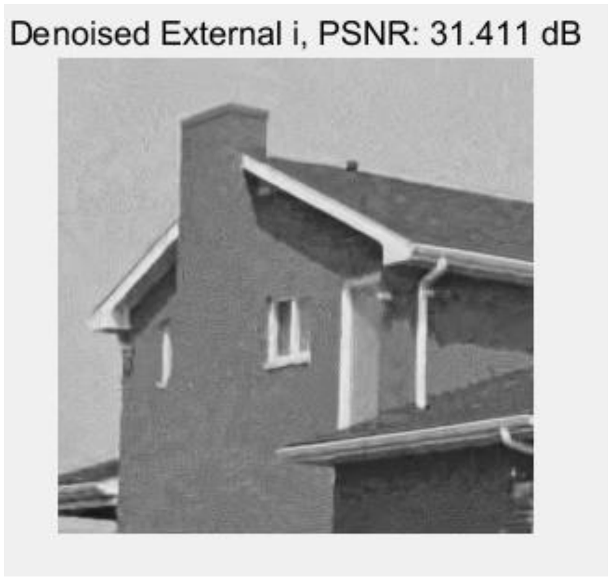
Step5: Now, we move to section2: Final estimate. The input of this section are the original image and the basic estimate. First, it is necessary to find similar block. The basic estimate is the reference image so that for each block we want to process in original image we have to find similar blocks in the reference image instead of finding them in the original image. Therefore, we have 2 groups (the former one in step1 and new one in step5).

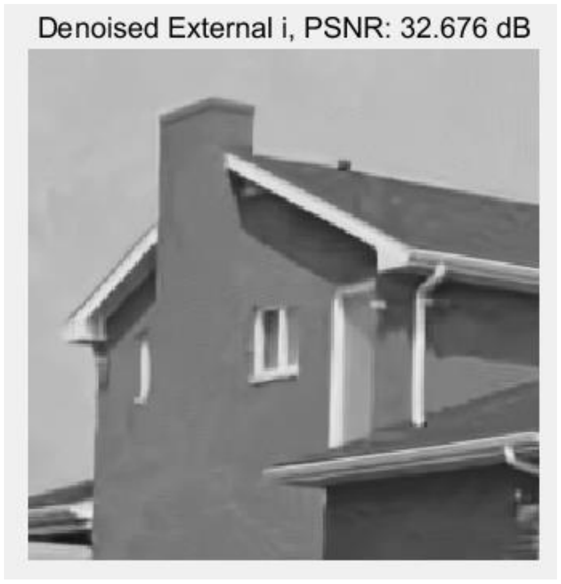
Step6: Collaborative Wiener filtering. Do 3-D transform to 2 blocks and pass a Wiener Filter. Create the Weiner filter initial estimate.

Step7: Calculate the final estimate and reconstruct the image.

Results

The results are the output from the program [2].





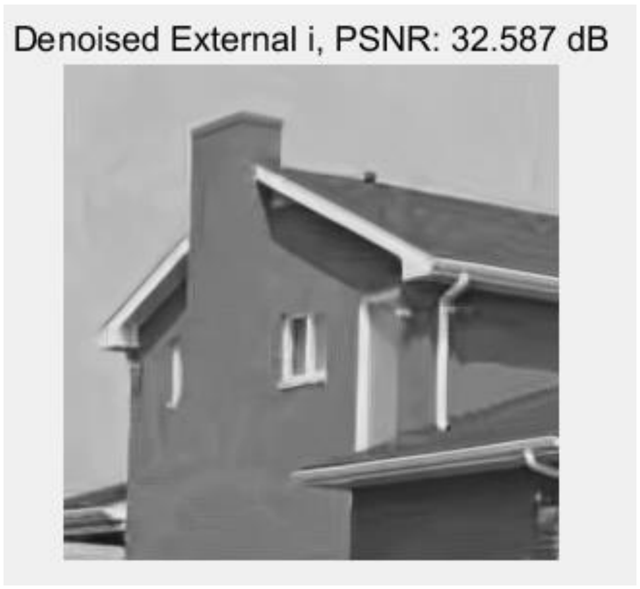


Figure59 BM3D Block size(4,8,16)

Denoised External i, PSNR: 32.641 dB
Denoised External i, PSNR: 32.727 dB

Figure60 BM3D Stepsize(2,4)

Denoised External i, PSNR: 32.625 dB
Denoised External i, PSNR: 32.729 dB

Figure61 BM3D Search Window size(29,49)

Discussion:

Parameters	PSNR(Basic estimate)	PSNR(Final estimate)
Block Size=8 step=3 Search window size =39	32.06	32.68
Block Size=4 step=3 Search window size =39	29.24	31.41
Block Size=16 step=3 Search window size =39	32.18	32.59
Block Size=8 step=4 Search window size =39	31.89	32.64
Block Size=8 step=2 Search window size =39	32.11	32.69
Block Size=8 step=3 Search window size =29	32.05	32.62
Block Size=8 step=3 Search window size =49	32.06	32.73
Mean filter(5*5)	27.23	
Non local mean(3,5,25)	27.75	

Table3 BM3D performance

(1) Because the preparation for this kind of filter is grouping. Block matching is a popular and effective way to find similar blocks in a image and stack them in a 3-D array.

(2) The motivation of Wiener Filtering is that the normal estimator is biased and the error is quite big. In order to remove this error and get more accurate estimate. Also the Wiener filter is more effective. According to

Table3, the Final estimate is better than Basic estimate(without step2). The basic estimate is a result of hard-thresholding. Therefore, step 2 is necessary.

(3) BM3D is a spatial and frequency domain filter. The 3-D transform is done in the frequency domain and the final aggregation is finished in the spatial domain.

(4) The performance of BM3D is better than PLPCA from Table3.

Reference:

[1] http://josephsalmon.eu/code/index_codes.php?page=PatchPCA_denoising

[2] http://www.cs.tut.fi/~foi/GCF-BM3D/