Muthulakshmi Chandrasekaran

4486-1802-42

muthulac@usc.edu

EE569 Homework #1

February 04, 2018

Problem 1: Basic Image Manipulation

Part A – Color Space Transformation

A.1 Motivation

An image is a two-dimensional function that represents some characteristic viewed in the 3D plane. An image can be defined as $f(x,y)$, where for each point (x,y), $f(x,y)$ represents some characteristic of the image, an example being brightness. A digital image is made of pixels, where each pixel is the smallest unit of the image. A digital image can be represented as an $N * N$ array of elements. Digital Image Processing involves processing the digital image to enhance the features of the image – such as increasing brightness, noise removal etc.

The basic operations are obtaining the input, processing on the image, and presenting the output. This problem introduces these basic operations. The conversion of an image from grayscale to a color space is done, followed by resizing of an image.

There are several types of images – Grayscale images, Color images, Volume Image, Range Image etc. The scope of interest is only on Grayscale images and Color images. A Grayscale Image gives only the brightness information. Each pixel in a grayscale image corresponds to the intensity of light. For example, an 8-bit grayscale image has each pixel value ranging from 0 to 255, '0' representing bright (or white) and '255' representing dark (or black). A Color Image has three values per pixel, each measuring the chrominance and intensity of light. Each pixel can be represented as a vector of three elements. The entire image can be viewed as three layers or planes – where each layer corresponds to a color. The pixel values in each layer represent the brightness of the color in that plane. Accordingly, these color planes are called **color spaces**. Some examples of color spaces are RGB (Red – Green – Blue), CMYK (Cyan – Magenta – Yellow – Black), HSV (Hue – Saturation – Value). RGB Color Space is widely used in the Displays (like computer monitors), since this is an additive spectrum. CMYK color space is widely used in the

printing industry. Converting an image from grayscale to RGB is done by either of the following approaches

- Lightness Method
- Averaging Method
- Luminosity method

Conversion from RGB to CMY follows the formula

$$C = 1 - R;\ M = 1 - G\ ;\ Y = 1 - B$$

A.2 Approach

All the input images are in. raw format. The processing code is done using C++. The input image is read in a three – dimensional array of size $Size1 * Size2 * BytesPerPixel$, where $Size1$ gives the length of the image, $Size2$ gives the width of the image and $BytesPerPixel$ gives the number of bytes needed to store the image. Here in all the given images, $BytesPerPixel$ is 3, which means that each color is represented by a byte.

Here the given color image is to be transformed into Grayscale Image. In the first part, Images on the RGB color space are to be transformed to grayscale images. The three methods used are

- Lightness Method – here the minimum and maximum of each pixel is averaged to get the new grayscale value using the formula

$$New\ pixel\ value = \frac{\min(R, G, B) + \max(R, G, B)}{2}$$

- Averaging Method – here the grayscale value is found by simply taking the mean of the three colors as

$$New\ pixel\ value = \frac{\mathrm{R} + \mathrm{G} + \mathrm{B}}{3}$$

- Luminosity method – here the weighted average of R, G and B pixels are calculated to find the new grayscale value. Here Green is given more weight because green easily perceives human eye.

$$New\ pixel\ value = 0.21 * R + 0.72 * G + 0.07 * B$$

In the second part, RGB Color Space is transformed into CMY(K) color space. Each pixel in the RGB color space is transformed into CMY(K) Color Space using the formula

$$C = 1 - R;\ M = 1 - G\ ;\ Y = 1 - B$$

A.3 Results

In the first part, input image is the Tiffany image, which is in RGB color space, as in Figure 1.1.1. It is converted to grayscale image using the above said three methods. Figure 1.1.2 shows the output of the Tiffany image, converted to grayscale using Lightness method.

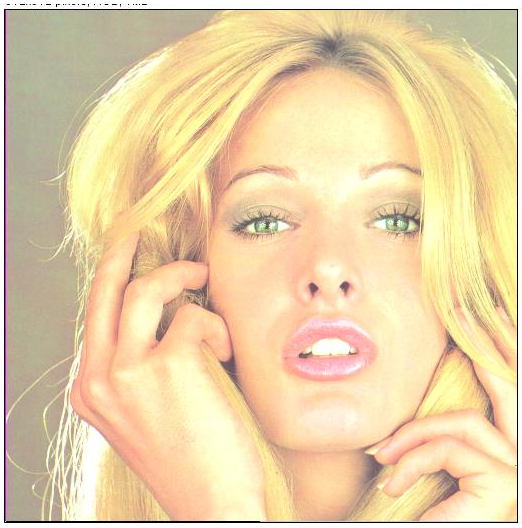


Figure 1.1.1 Original Tiffany. Raw image in RGB Color Space, of size $512 * 512$

Figure 1.1.2 Grayscale Tiffany image using Lightness method, size $512 * 512$

Figure 1.1.3 shows the grayscale Tiffany image, using Averaging Method.

Figure 1.1.3 Grayscale Tiffany.raw image using Averaging method, size $512 * 512$

Figure 1.1.4 Grayscale Tiffany.raw image using Luminosity method, size $512 * 512$

Figure 1.1.4 shows the Tiffany image in grayscale, which is converted from RGB space using the Luminosity method.

In the second part, images in RGB color space are converted to CMY(K) color space. Here the input images are Bear.raw and Dance.raw. Figure 1.1.5 shows the Bear.raw image, of size 854*480 in RGB space.

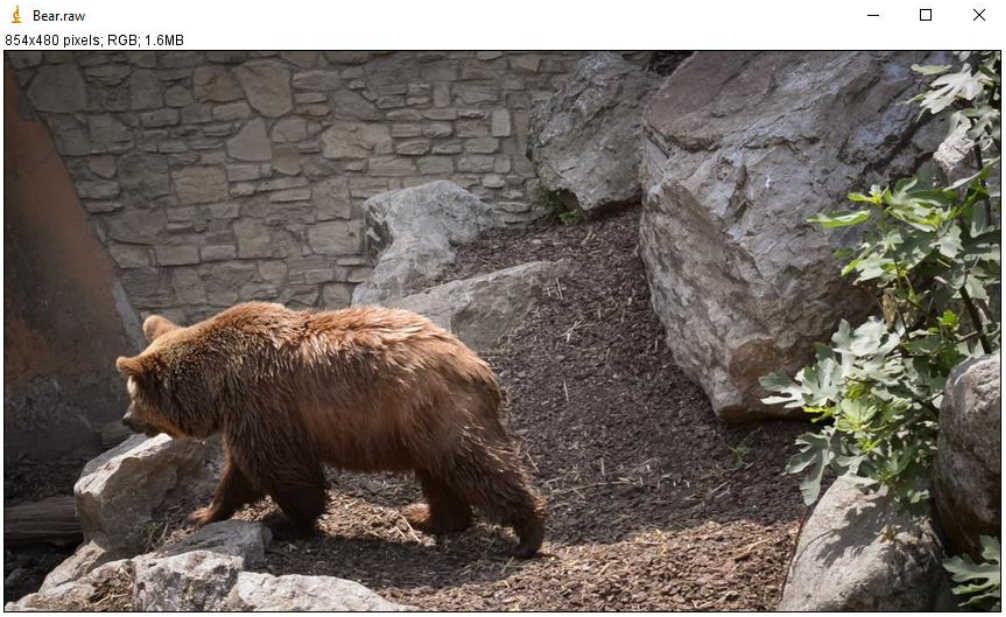


Figure 1.1.5 Original Bear.raw image in RGB Color Space, of size $854 * 480$

Figure 1.1.6 shows the Dance.raw image, of size 854*480 in RGB space

Figure 1.1.6 Original Dance.raw image in RGB Color Space, of size $854 * 480$

The image obtained in CMY(K) color space for the input image (Figure 1.1.5) is shown in Figure 1.1.7.

Figure 1.1.7 Bear.raw image in CMY(K) Color Space, of size $854 * 480$

The individual C, M, Y channel images are shown in Figure 1.1.8(a), Figure 1.1.8(b) and Figure 1.1.8(c) respectively.

Figure 1.1.8(a) Channel C of Bear.raw image

Figure 1.1.8(b) Channel M of Bear.raw image

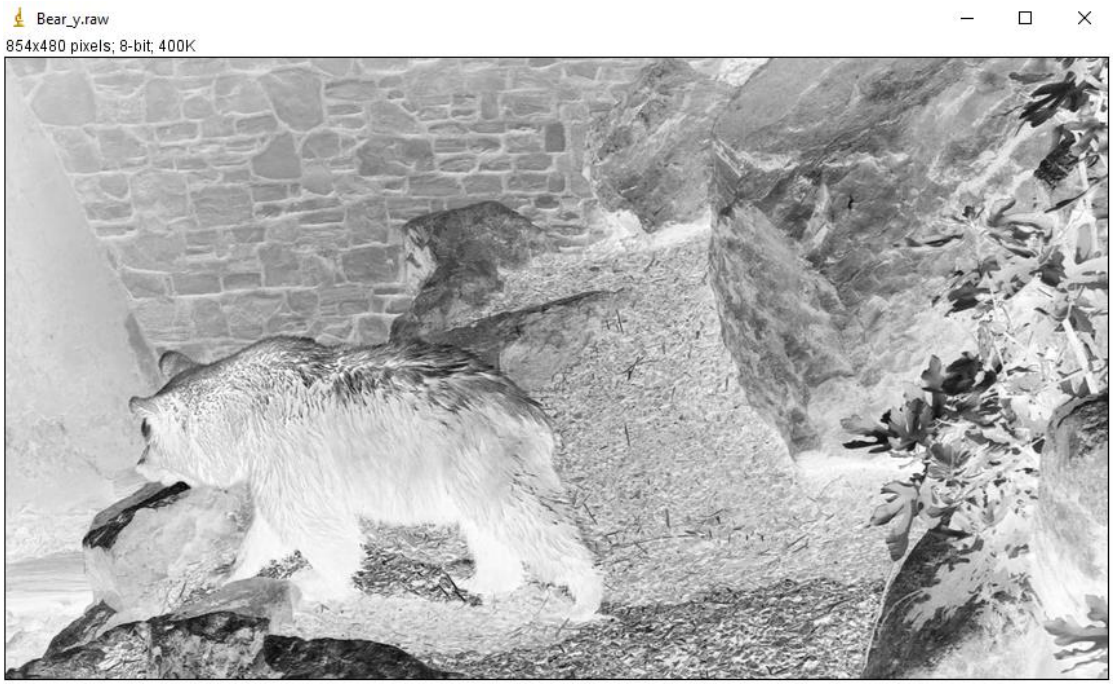


Figure 1.1.8(c) Channel Y of Bear.raw image

The image obtained in CMY(K) color space for input image (Figure 1.1.6) is shown in Figure 1.1.9.

Figure 1.1.9 Dance.raw image in CMY(K) Color Space, of size $854 * 480$

The individual C, M, Y channel images are shown in Figure 1.1.10(a), Figure 1.1.10(b) and Figure 1.1.10(c) respectively.

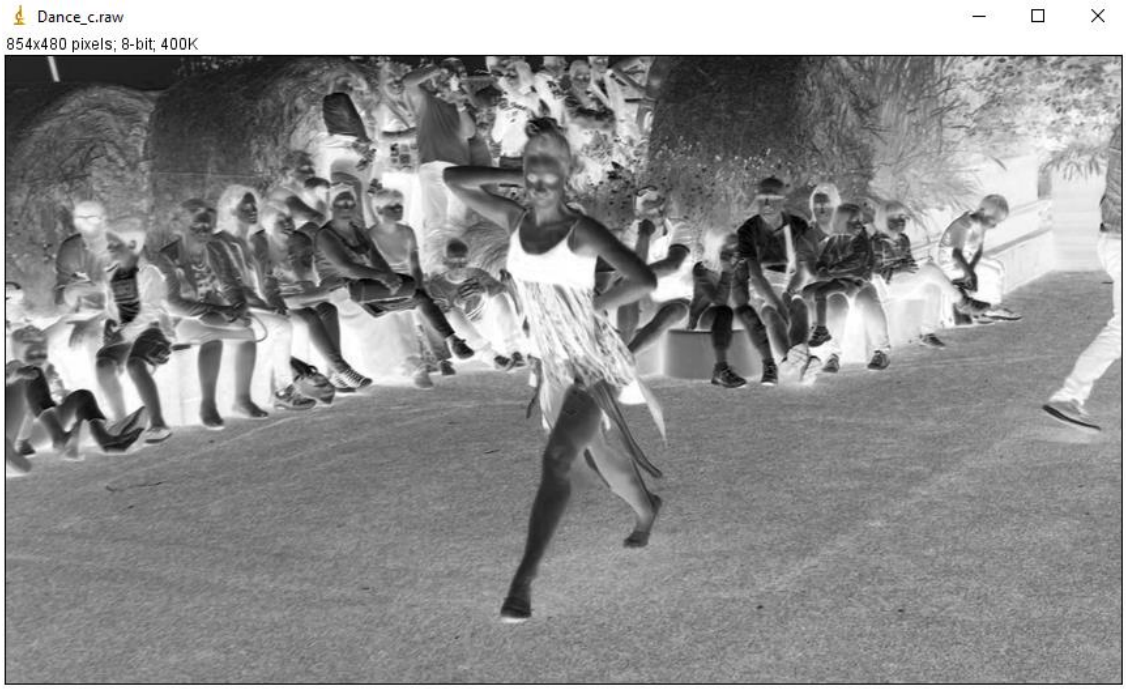


Figure 1.1.10(a) Channel C of Dance.raw image

Figure 1.1.10(b) Channel M of Dance.raw image

Figure 1.1.10(c) Channel Y of Dance.raw image

A.4 Discussion

Basic Color Spaces are RGB for displays and CMY(K) for printing purposes. Basic image processing steps like converting an RGB image into a Grayscale image are analyzed. Conversion to grayscale image can be done in either of the three ways – Averaging, Lightness or Luminosity method. In Averaging method, a pixel is replaced by the average of its values across all the three-color bands. So, even if one channel has extreme values, the image tends to become darker or vice versa. In the lightness method, the greatest and smallest values are averaged. Thus, the grayscale value of every pixel is reduced. This tends to reduce the contrast of the image. In the luminosity method, Green is given more weight compared to Red and Blue. This is because the human eye tends to perceive Green more compared to Red and Blue. So, this method works fine for most of the images, though the best conversion method depends on the image. Inn printing, CMY(K) color space is used, which is the complement of the RGB Color Space. Individual Channels can be extracted from the original CMY image, and thus each channel is a grayscale image. Combination of these channels result in the required colors.

Part B – Image Resizing using Bilinear Interpolation

B.1 Motivation

One vital approach in processing digital images is scaling of an image. Scaling refers to the geometric transformation of an image, in terms of the size of the image. There are two divisions of scaling – Image reduction and Image zooming. Image reduction is done by sub – sampling an image. Image zooming is obtained by interpolation and pixel replication. Scaling, in general, compresses or expands the image along the co – ordinate directions. To convert an image from a size of $M * M$ into a size of $N * N$, $N > M$, interpolation can be used. There are three types of commonly used interpolation methods – Nearest neighbor (only the nearest neighbor pixel), Bilinear (considering the four closest neighbors) and Bicubic (conserving the closest 16 neighbors).

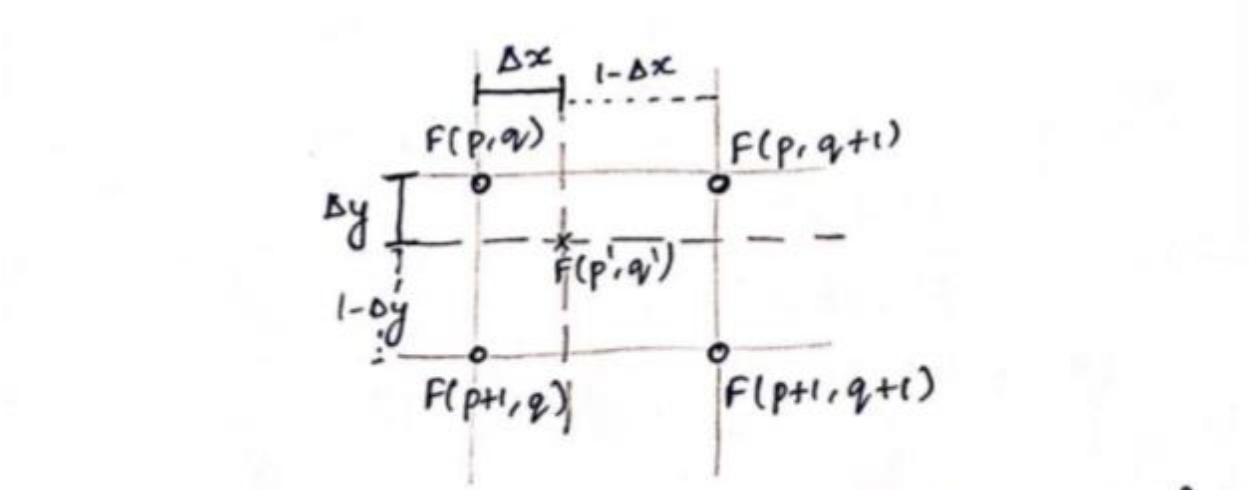
B.2 Approach

Here, the topic of interest is resizing an image using interpolation. The input image is of size $M * M$ pixels. To convert the image into another image of size $N * N$, interpolation is done. For this, pixels need to be calculated in the non – integer positions. Each pixel is obtained by interpolating the values of the neighboring pixels.

Nearest neighbor method considers only the closest pixel. Though this is simple, it tends to magnify noise at the boundaries. Hence this method is not used.

Bilinear interpolation considers the closest four pixels surrounding the unknown pixel's position. Basically, interpolation is done along both the horizontal and vertical axes – hence called bilinear interpolation. The interpolation is not linear – the weighted average of the four pixels is taken.

Consider the image as shown below. The image F is known – The points $F(p,q)$, $F(p,q+1)$, $F(p+1,q)$ and $F(p+1,q+1)$ are known. The pixel value at the point $F(p',q')$ must be calculated. The formula for bilinear interpolation is as follows.



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

The coding approach in C++ is as follows.

i. The input image is obtained from the file using the file commands, and stored in an array.
ii. The output image array is defined.
iii. For every pixel P in the output image, the value is calculated using the bilinear interpolation formula. First, the point P in the Red Band is checked for the nearest neighboring pixel in the known image. Correspondingly Δx and Δy are calculated. Using the four nearest neighbors, the value of P is calculated. This is repeated for every pixel in the R band. The consideration here is that interpolation should be done only in the same color band.
iv. This process is repeated for each unknown pixel in the rest two bands – G and B.

B.3 Results

The actual Airplane.raw image is shown in Figure 1.1.11, which is in RGB color space, of size $512 * 512$.This image is resized to size $650 * 650$, as shown in Figure 1.1.12.

Figure 1.1.11 Original Airplane.raw image of size $512 * 512$

Figure 1.1.12 Resized Airplane.raw image of size $650 * 650$

B.4 Discussion

Bilinear interpolation is used in resizing an image of size $512 * 512$ into an image of size $650 * 650$. This method makes use of the four nearest neighboring pixels of the known image. Here the pixel value is based on the weights determined by the closeness to the known pixels. Bilinear interpolation is not so simple as the nearest neighbor interpolation, but it gives better results in terms of absence of discontinuities. One disadvantage is that this method has a low frequency property, hence the higher frequency regions may appear faded. To improve this, one possible solution is to go for bi cubic interpolation, where the interpolation takes sixteen neighboring pixels into account.

References:

1. William K. Pratt, “Digital Image Processing”, Second Edition, John Wiley & Sons, 2001.
2. Rafael C Gonzalez, Richard E Woods, “Digital Image Processing “, Second Edition, Prentice Hall.
3. Dianyuan Han ,“Comparison of Commonly Used Image Interpolation Methods”, Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)
4. https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

Problem 2: Histogram Equalization

Part A – Basic Histogram Equalization

A.1 Motivation

Image Enhancement refers to the process of improving visual quality of some essential or all parts of the image. Given an input pixel $p(x, y)$, it is transformed to another pixel $q(x, y)$ using some transformation given as

$$q(x, y) = T(p(x, y))$$

where T referes to some transformation applied on the image. Simple transforms could be increasing the brightness of the image, reducing the brightness of the image etc. One useful tool for enhancing the properties of the image is the Histogram.

A histogram of an image plots the number of occurrences of a grayscale image versus the actual grayscale values. Modifications on the histogram of the image effect the contrast and the dynamic range of the image. Histogram equalization refers to spreading out the intensities across all the grayscale values.

A.2 Approach

Histogram Equalization can be done by two methods – using the Transfer Function Approach, and using the Cumulative Probability Based Approach. The motive of Histogram equalization is to evenly distribute the pixels, across the grayscale values 0 ~ 255.

a. <u>Transfer Function Based Approach:</u>

As known, histogram of an image plots the number of occurrences of each pixel of the image against the grayscale value. For a color image, there are three bands – R, G and B. Histograms are obtained for each band.

The following steps are done:

i. For each color band, count the number of pixels in each grayscale value.

Let the count of each pixel p_i, let the number of occurrences be n_i, where $i = 0,1,\ldots,255$.

ii. Normalize the occurrences by the total size – Hence the probability of occurrence of each grayscale value is obtained.

The probability of occurrence of each pixel is

$$prob(p_i) = \frac{n_i}{M * N}$$

if the image is of size $M * N$.

iii. Obtain the Cumulative Distribution Function of the probabilities. Here the CDF acts as the transfer function of the system. The CDF is given by

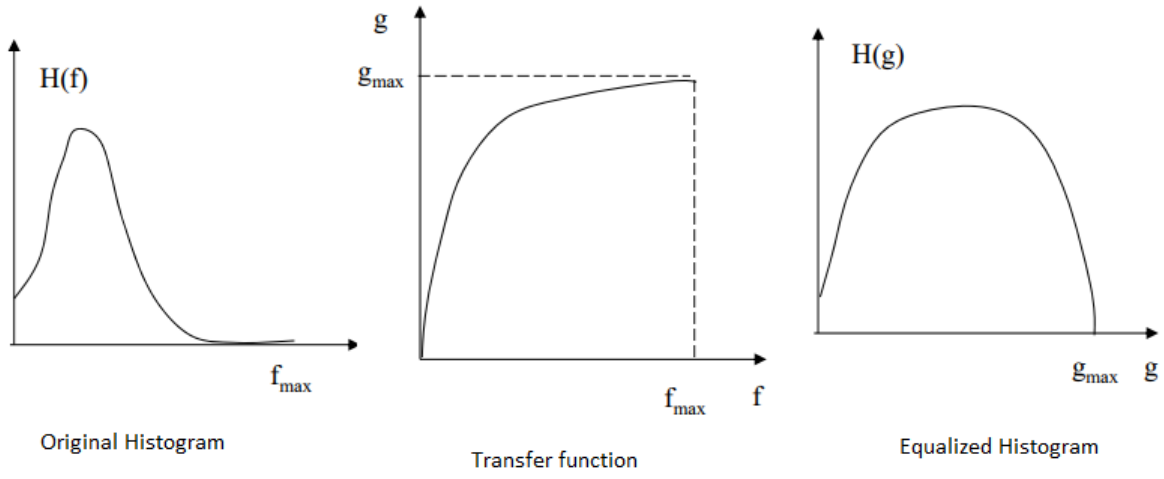
$$cdf(k) = \sum_{i=0}^{i=k} prob(p_i)$$

iv. Based on the values of CDF obtained, the pixels are distributed again across the grayscale values.

v. Now the histogram of the image is equalized – that is pixels are spread out.

vi. This is repeated for all the three bands.

An example of enhancing a too – dark image is shown below.



The coding approach in C++ is as follows.

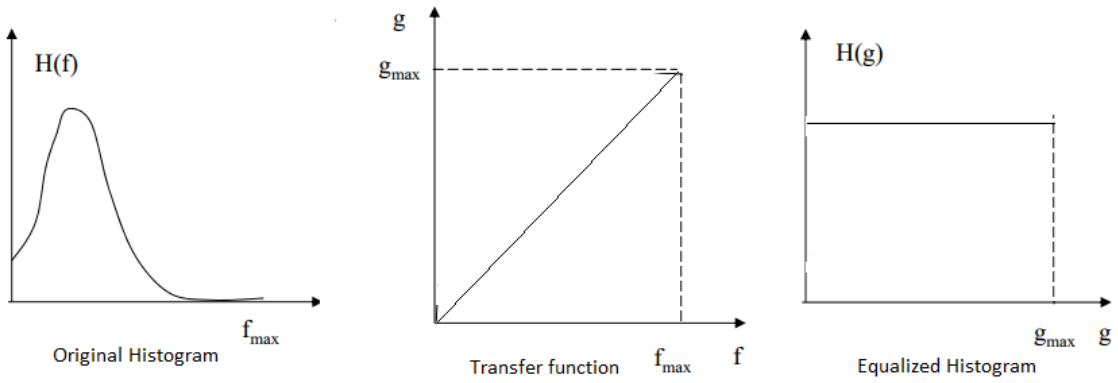
i. The input image is obtained from the file using the file commands, and stored in an array.
ii. The output image array is defined.
iii. The number of occurrences of each grayscale value is calculated, normalized and the CDF is obtained.
iv. Then the output image band is obtained by reallocating pixels according to the transfer function.
v. This process is repeated for each unknown pixel in the rest two bands – G and B.
vi. The final output is written in a separate .raw file. Here the input and output are two separate image files, since the information in the input file should not be modified until finding the perfect distribution.

b. <u>Cumulative Probability based Approach:</u>

This method aims in achieving a flat histogram. The approach is the same as above, except that the transfer function is a linear function, where the output image has a histogram that is uniformly distributed i.e. each pixel has the same number of occurrences. The approach is as follows.

i. For each color band, count the number of pixels in each grayscale value. Let the count of each pixel p_i, let the number of occurrences be n_i, where $i = 0,1,\ldots,255$.
ii. Check if the number occurrences of each pixel p_i is less than the count of $\frac{Total\ Number\ of\ pixels, M*N}{255}$, if the image is of size $M * N$.
iii. If equal assign the value. If less, include some pixels from the next grayscale value to have the value p_i. If more, redistribute them to the next grayscale value.
iv. Thus, the transfer function will be a linear ramp function.

An example of enhancing a too – dark image using this approach is shown below.



The coding approach in C++ is as follows.

i. The input image is obtained from the file using the file commands, and stored in an array.

ii. The output image array is defined.

iii. The number of occurrences of each grayscale value is calculated. It is checked if the total number of values is less than or equal to the value $\frac{Total\ Number\ of\ pixels, M*N}{255}$

iv. A separate variable is used for counting how many pixels go into one grayscale value.

v. By the process of addition and elimination, fixed number of pixels are allocated to each gray scale value.

vi. This process is repeated for the rest two bands – G and B. The final output is written in a separate. raw file. Here the input and output are two separate image files, since the information in the input file should not be modified until finding the perfect distribution.

A.3 Results

Figure 1.2.1 shows the actual Desk. Raw image, on which histogram equalization must performed.

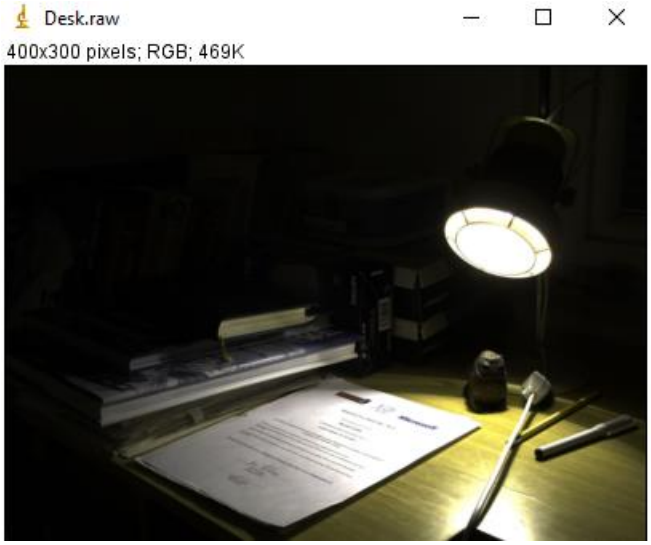


Figure 1.2.1 Original Desk.raw image, of size $400 * 300$

The histogram of each channel R,G and B of the original image Desk.raw is obtained. These values are plotted using MS – Excel as in Figure 1.2.2(a), Figure 1.2.2(b) and Figure 1.2.2(c).

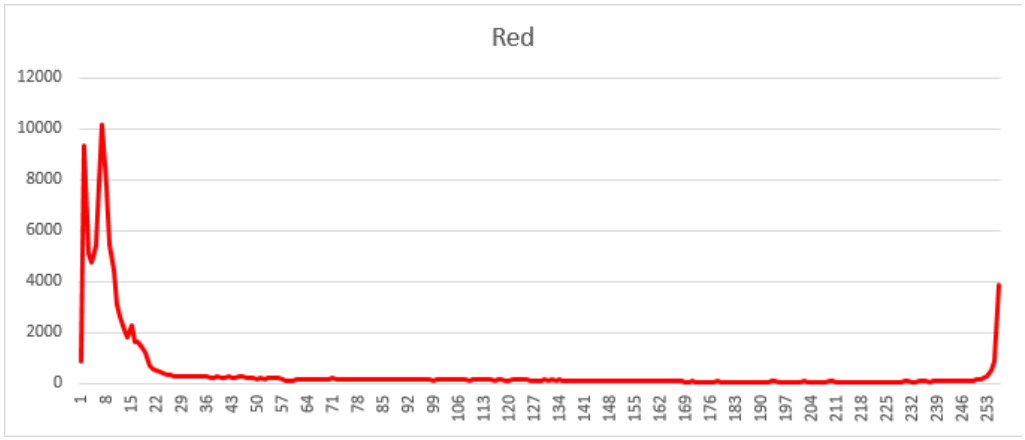


Figure 1.2.2(a) Histogram of R channel of the original Desk.raw image

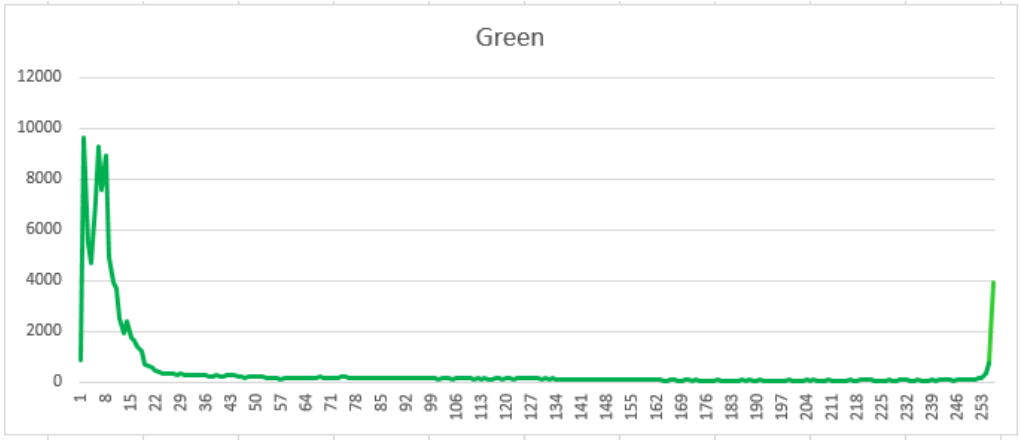


Figure 1.2.2(b) Histogram of G channel of the original Desk.raw image

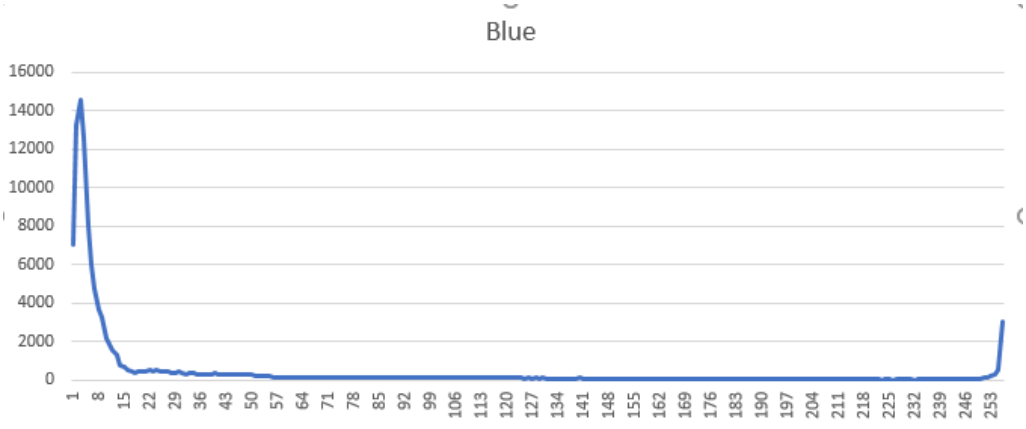


Figure 1.2.2(c) Histogram of B channel of the original Desk.raw image

Applying the Transfer Function based Histogram Equalization method (Method A), the enhanced image is obtained as in Figure 1.2.3.

Figure 1.2.3 Enhanced Desk.raw image after histogram equalization by Method A

The histogram of the actual image and that of the enhanced image is shown in Figure 1.2.4(a) and 1.2.4(b) respectively. The CDF plots of the actual image and enhanced image are shown in Figure 1.2.4(c) and 1.2.4(d) respectively.

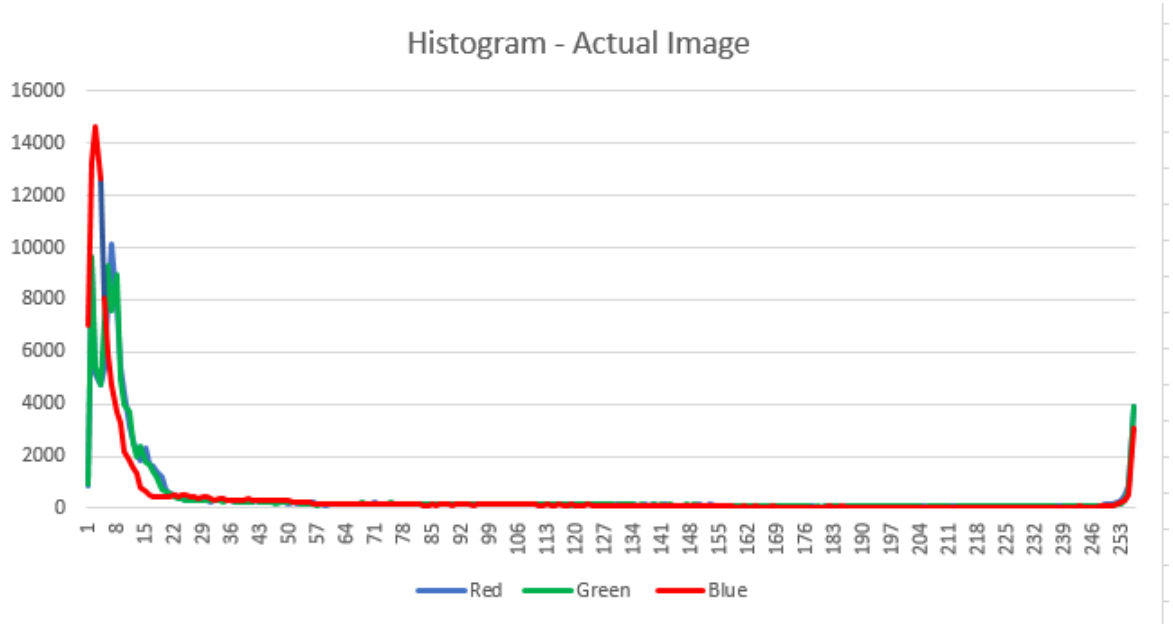


Figure 1.2.4(a) Histogram of the actual Desk.raw Image

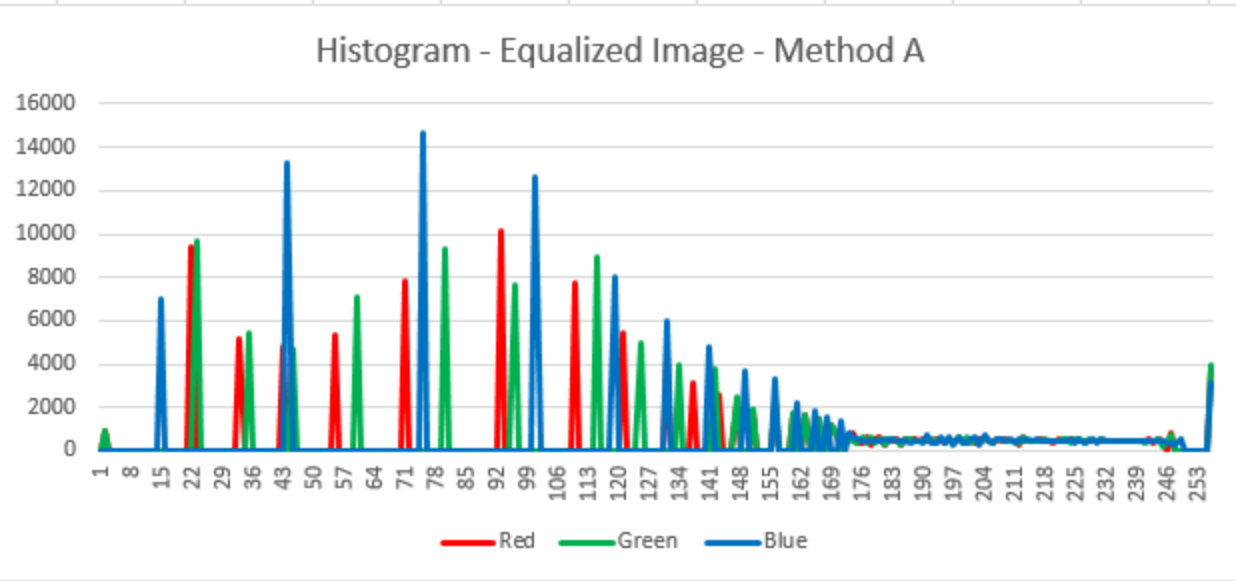


Figure 1.2.4(b) Histogram of the equalized Desk.raw Image

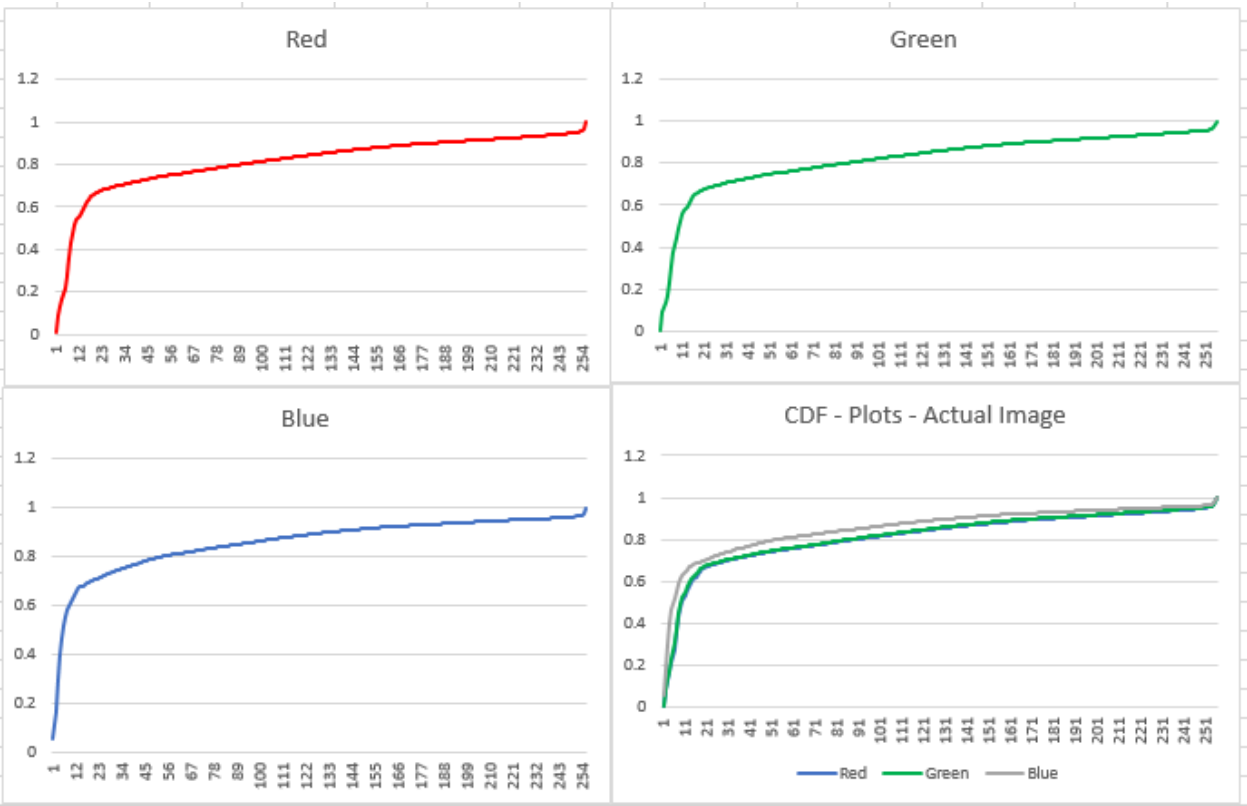


Figure 1.2.4(c) CDF Plot of the actual Desk.raw Image

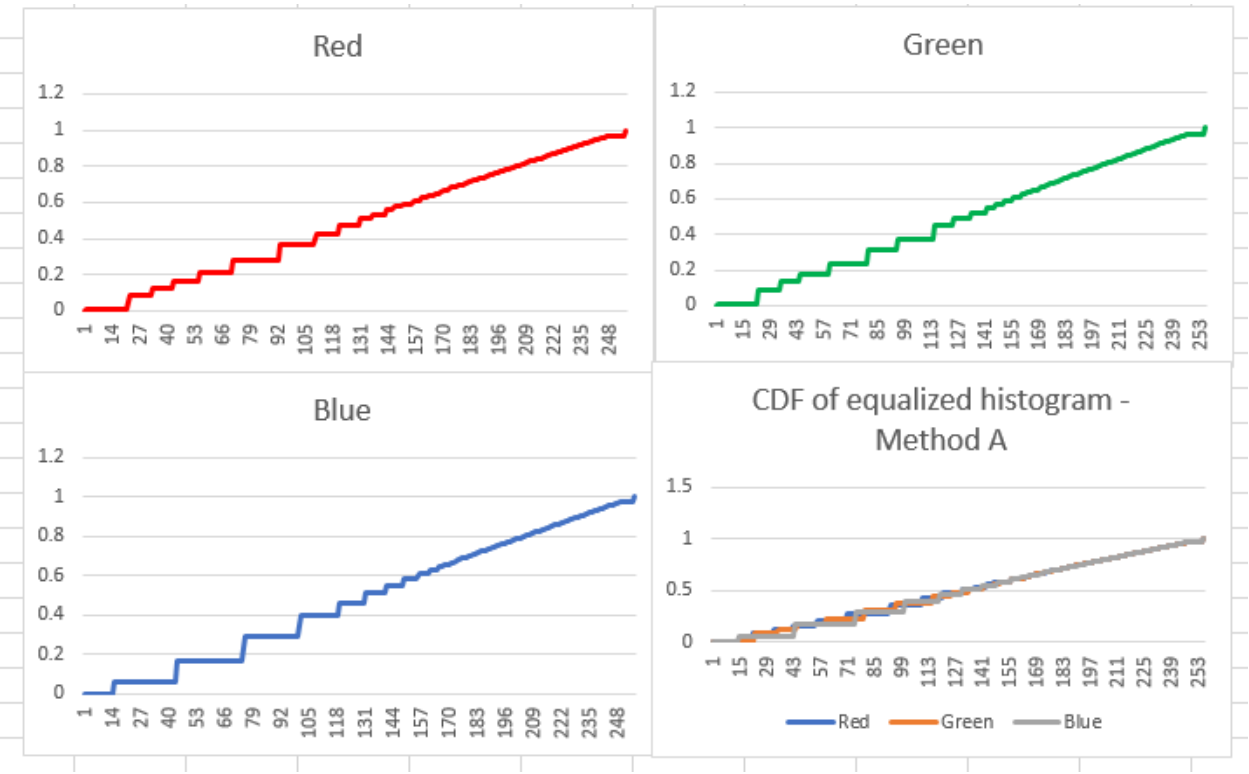


Figure 1.2.4(d) CDF Plot of the equalized Desk.raw Image

Applying the Cumulative Probability based Histogram Equalization method (Method B), the enhanced image is obtained as in Figure 1.2.5.

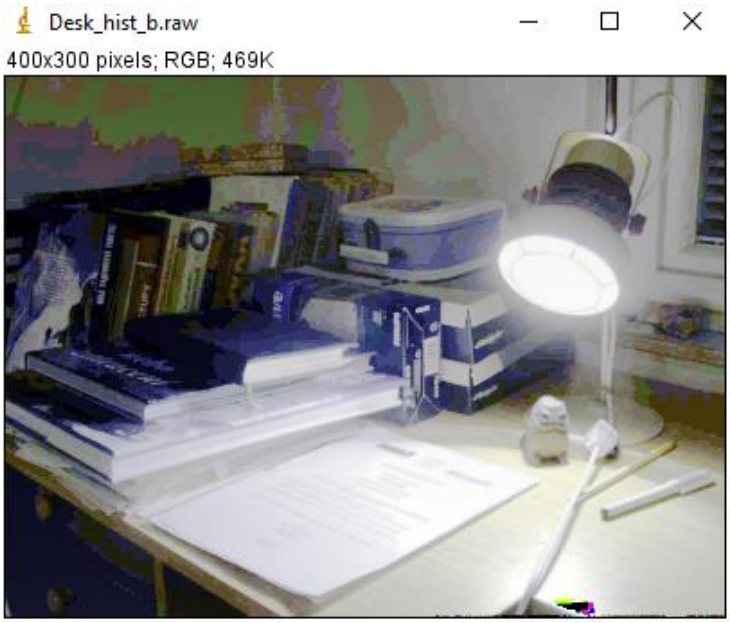


Figure 1.2.5 Enhanced Desk.raw image after histogram equalization by Method B

The histogram of the actual image and that of the enhanced image is shown in Figure 1.2.6(a) and 1.2.6(b) respectively. The CDF plots of the actual image and enhanced image are shown in Figure 1.2.6(c) and 1.2.6(d) respectively.

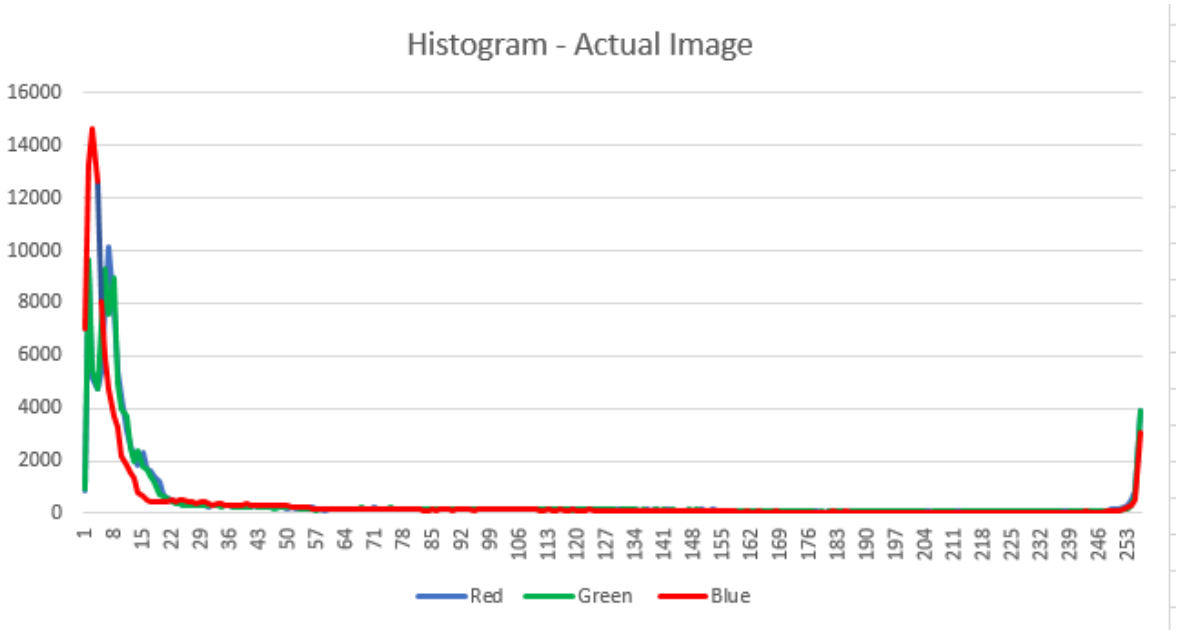


Figure 1.2.6(a) Histogram of the actual Desk.raw Image

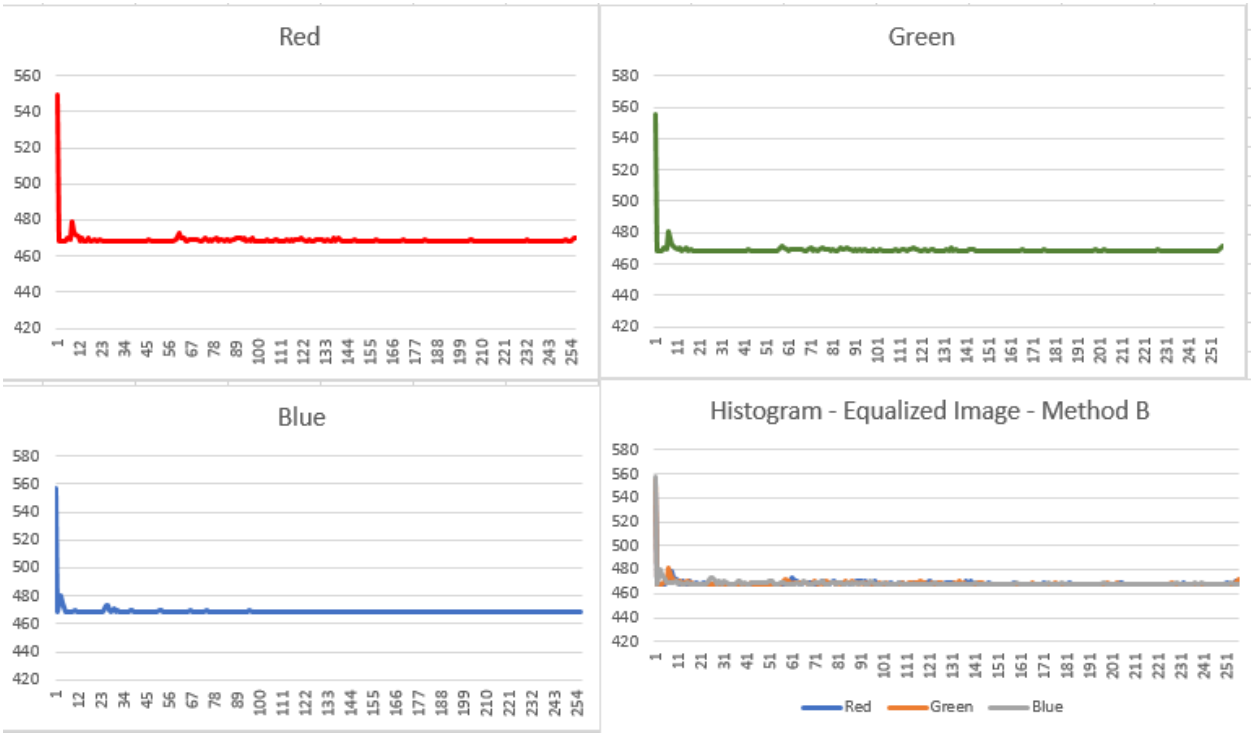


Figure 1.2.6(b) Histogram of the equalized Desk.raw Image

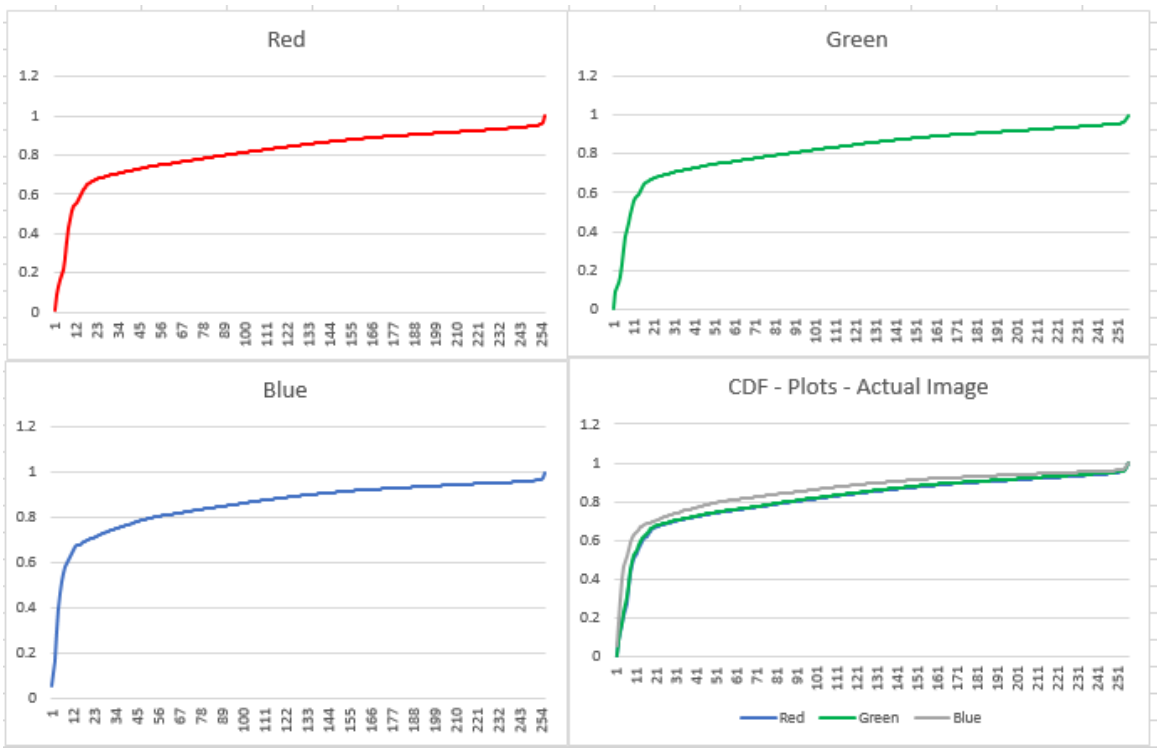


Figure 1.2.6(c) CDF Plot of the actual Desk.raw Image

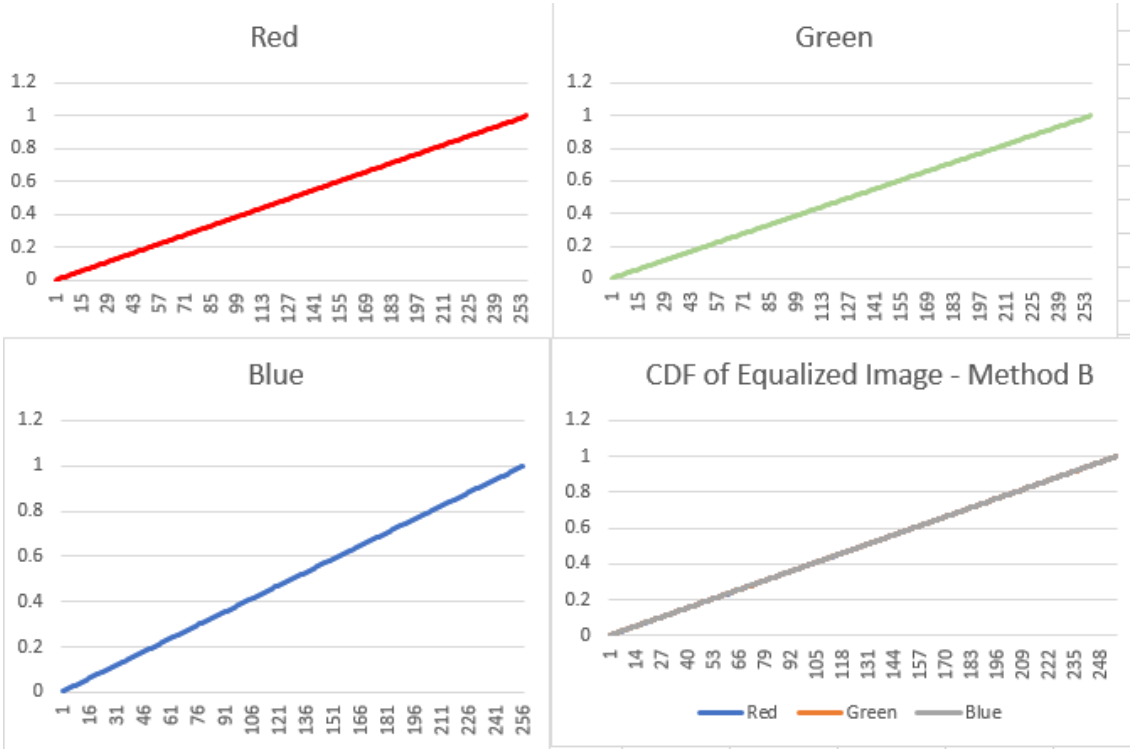


Figure 1.2.6(d) CDF Plot of the equalized Desk.raw Image

A.4 Discussion

(1) The histograms of the R, G and B bands of the original image are plotted in Figure 1.2.2(a), 1.2.2(b) and 1.2.2(c) respectively.
(2) Figure 1.2.3 shows the result of the Histogram equalization using the Transfer function based approach. Figure 1.2.4(a), (b), (c) and (d) show the corresponding plots.
(3) Figure 1.2.5 shows the result of the Histogram equalization using the Cumulative Probability based approach. Figure 1.2.5(a), (b), (c) and (d) show the corresponding plots.
(4) Comparing the input image with the images in Figure 1.2.3 and Figure 1.2.5, it is observed that the cumulative probability based histogram equalization (method B) leaves out some black spots in the corner of the image. This might be possibly due to some pixels being misarranged. But Method B still has an advantage over Method A. In method A pixels can be redistributed only after the highest peak is observed – i.e. pixels can be distributed only towards the right, since it is CDF based. But in Method B, it is a simple bucket filling algorithm, pixels can be redistributed anywhere – either to the left or right, hence a uniform CDF based equalization. To improve the current result, the transfer function can be more clearly defined.

Part B – Image Filtering – Oil Painting Effect

B.1 Motivation

Image Enhancement refers to the process of improving visual quality of some essential or all parts of the image. Out of all the transformation methods available, Histogram is the most useful one. From a histogram, a lot of useful information can be extracted, relating to brightness, contrast, darkness etc. A potential application is the different filters that are used in photo apps, wherein for the same image, different effects are identified.

A proper study of the histogram helps us to identify the areas of low contrast, and enhance them. Every little modification in the histogram gives a new image. The number of input colors can be modified, the range of the histogram could be altered, the edges can be enhanced, different filters can also be applied. Here, the image is enhanced to give an effect like an oil painting.

B.2 Approach

Initially, the histogram of the image is obtained. A histogram of an image plots the number of occurrences of a grayscale image versus the actual grayscale values. Histogram is obtained for each channel. The effect of oil painting using filters can be done in two steps:

i. Quantization of the image: The actual RGB image uses three bytes per pixel, one for each channel. There are 256 grayscale values in each channel. Quantizing the image reduces to 4 grayscale values per channel.

ii. In the quantized image, a filter is applied that replaces each pixel by the most occurring pixel in its $N * N$ neighborhood. The window size is $N * N$. N is chosen to be an odd integer, ranging from 3 to 11. For each position in the output, the new pixel value is calculated based on this $N * N$ filter.

Here too, the input and output images are different, since the $N * N$ filter uses the pixels in the actual image.

The approach that is implemented in C++ is as follows:

i. The input image is obtained from the file using the file commands, and stored in an array.

ii. The output image array is defined.

iii. The number of occurrences of each grayscale value is calculated, and stored in an array named $hist$(say). In the input image, the total number of grayscale values will be 256 (i.e. 0 ~255). The total number of pixels in the image $M * N$ is divided into four grayscale values i.e. there are $\frac{M*N}{4}$ pixels in each of the new grayscale value.

iv. The actual grayscale value is obtained by taking the weighted average of the pixels in that range. For example, if the first $\frac{M*N}{4}$ pixels occur in the actual grayscale values $0 \sim j$, then the weighted average is obtained as

$$Weighted\ Average = \frac{(0 * hist[0]) + (1 * hist[1]) + \cdots + (j * hist[j])}{hist[0] + hist[1] + \cdots + hist[j]}$$

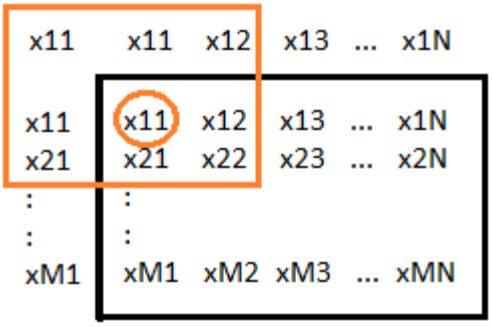
$$Weighted\ Average\ = \sum_{k=0}^{k=j} \frac{k * hist[k]}{hist[k]}$$

Thus, all pixels in the range $0 \sim j$ is replaced by the calculated weighted average. This is repeated to find the other 3 colors of one channel, with the ranges being $(j+1) \sim m, (m+1) \sim n, (n+1) \sim 255$.

Now the image is quantized. The quantized image is stored in a new array.

v. Next step is to use the $N * N$ neighborhood to calculate the most frequent pixel. Each pixel is considered along with its $N * N$ neighbours, and out of the $N * N$ pixels, the most frequently occurring pixel is put in the place of the actual pixel. The window for the pixels

in the boundary are padded by reflection along the boundary as given below (for window of size $3 * 3$, for the pixel $x11$.



Here the pixel x11 is the frequently occurring one of $\{x11, x11, x12, x11, x11, x11, x12, x21, x21, x22\}$. Hence the first pixel value in the output array is stored as $x11$.

vi. This process is repeated for all pixels in the channel, and as well for all the pixels in the G and B channels.

vii. The output array is stored in a separate. raw file, different from the input.

B.3 Results

The actual Star_Wars.raw image is shown in Figure 1.2.7(a), and the quantized image i.e. with a reduced color set (of 64 colors) is shown in Figure 1.2.7(b).

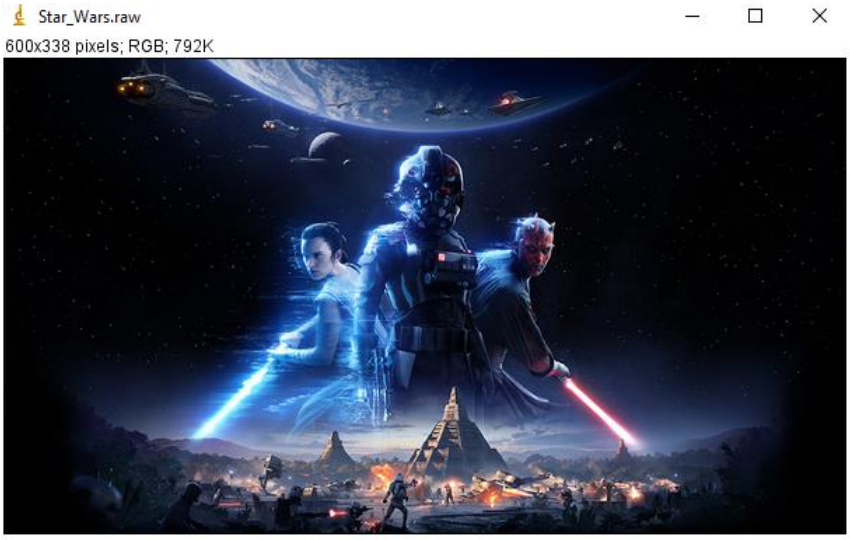


Figure 1.2.7(a) Actual Star_Wars.raw image

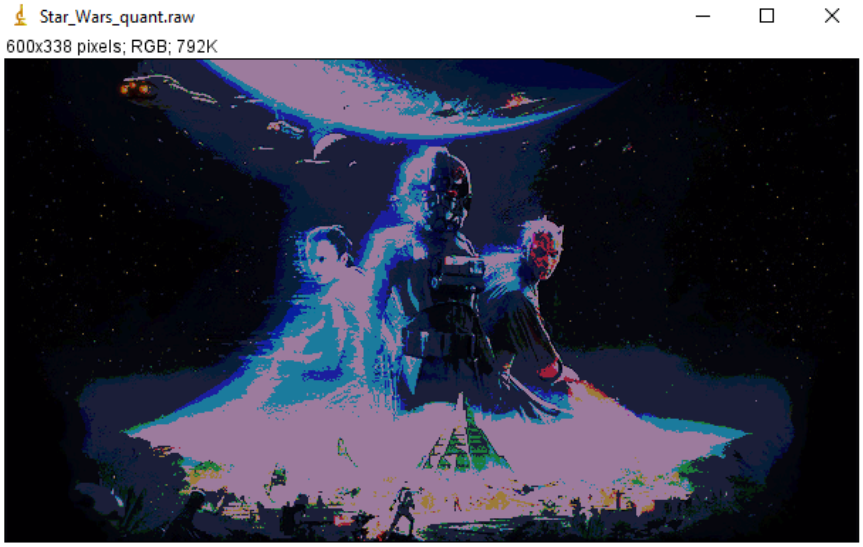


Figure 1.2.7(b) Star_Wars image with a reduced color set of 64 colors, 4 for each channel

The oil painting effect is created by choosing the most frequent pixel in the $N * N$ neighborhood of each pixel. Figure 1.2.7(c) to Figure 1.2.7(g) show results obtained for different values of N.

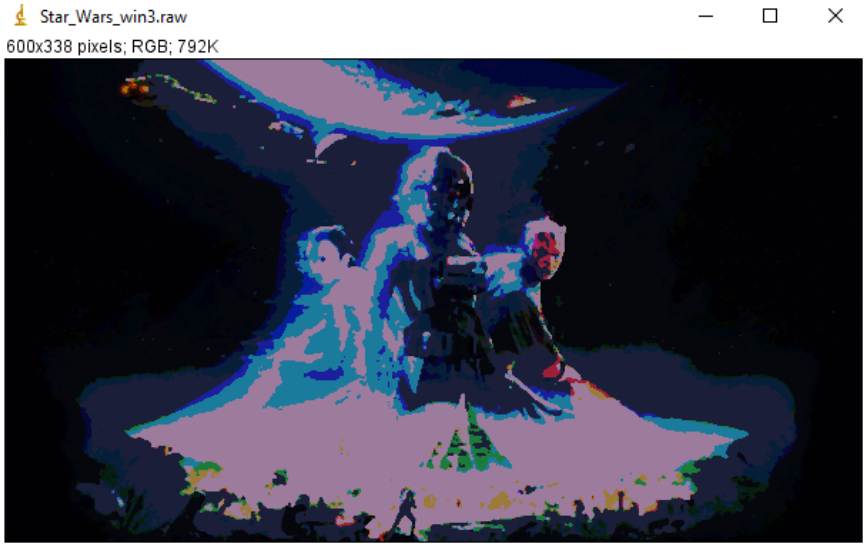


Figure 1.2.7(c) Oil – painting effect for quantized Star_Wars using $N = 3$

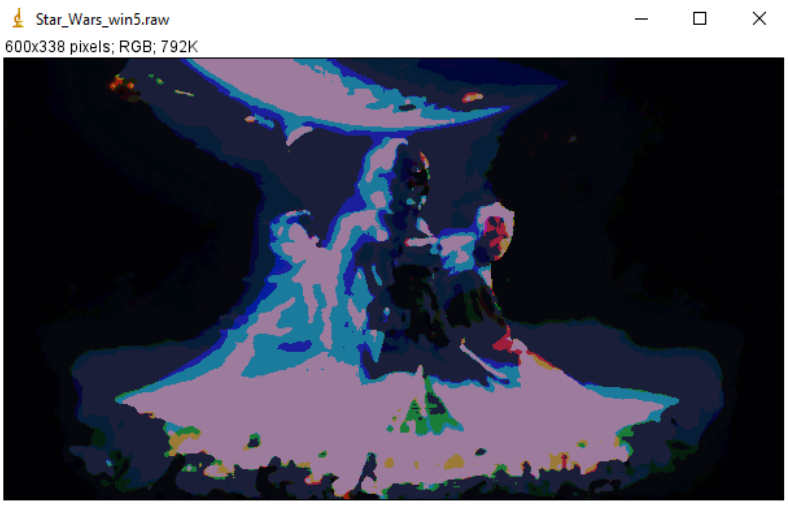


Figure 1.2.7(d) Oil – painting effect for quantized Star_Wars using $N = 5$

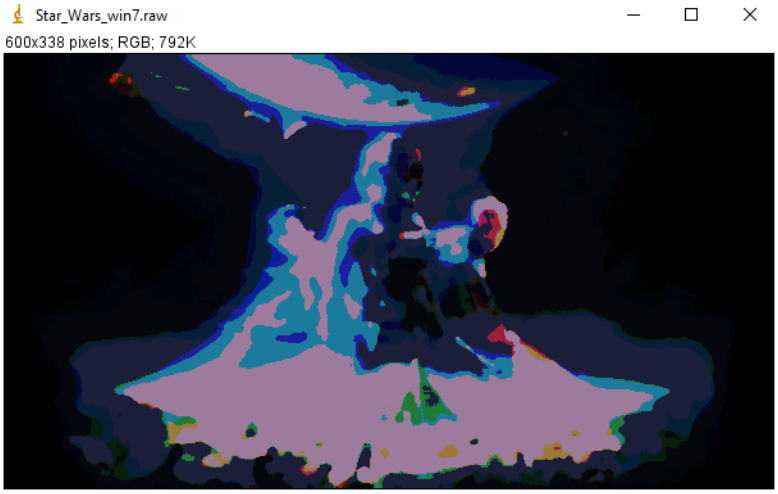


Figure 1.2.7(e) Oil – painting effect for quantized Star_Wars using $N = 7$

Figure 1.2.7(f) Oil – painting effect for quantized Star_Wars using $N = 9$

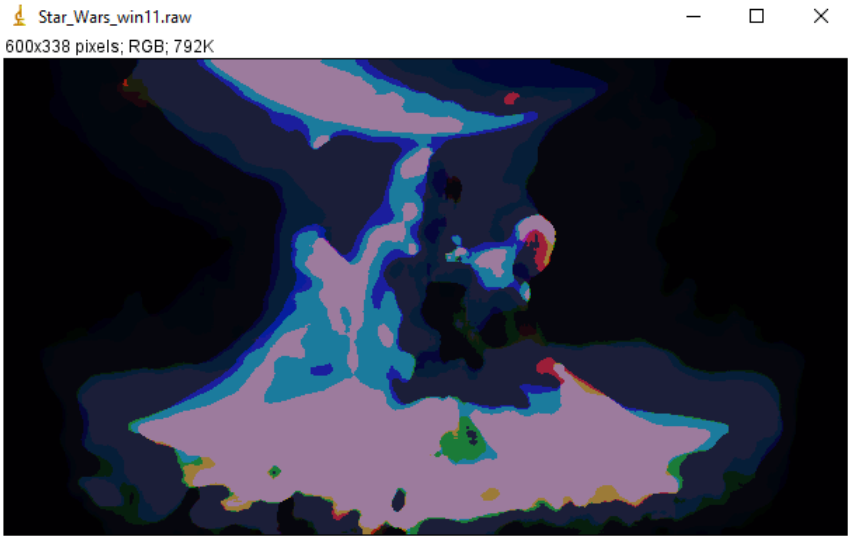


Figure 1.2.7(g) Oil – painting effect for quantized Star_Wars using $N = 11$

The same procedure is repeated for Trojans.raw image, which is shown in Figure 1.2.8(a), and the quantized image i.e. with a reduced color set (of 64 colors) is shown in Figure 1.2.8(b).

Figure 1.2.8(a) Actual Trojans.raw image

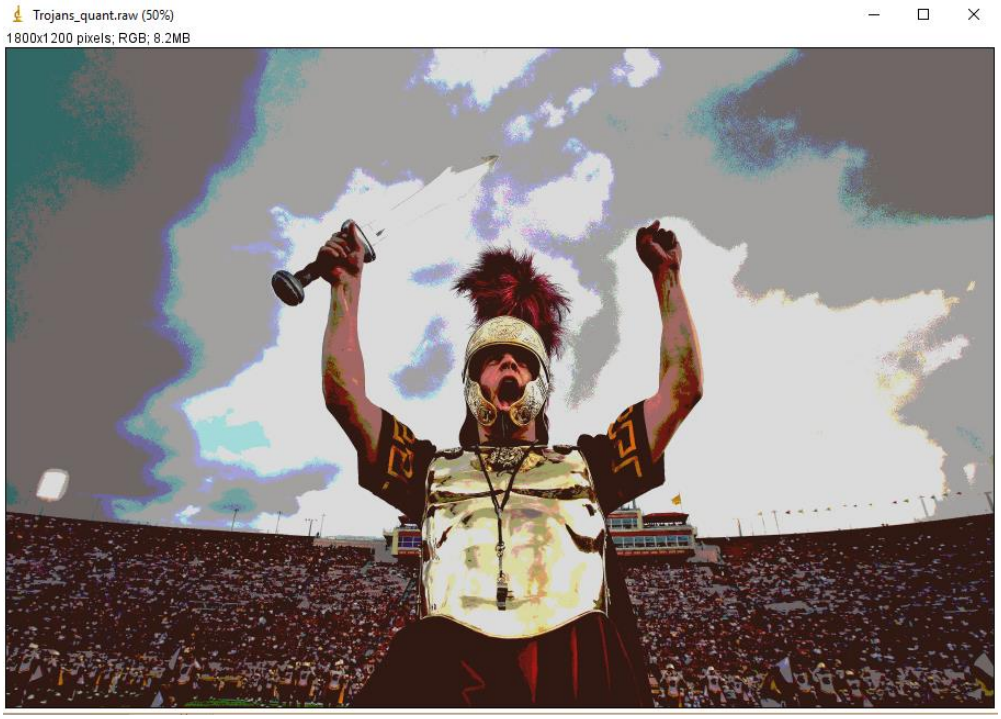


Figure 1.2.8(b) Trojans image with a reduced color set of 64 colors, 4 for each channel

The oil painting effect is created by choosing the most frequent pixel in the $N * N$ neighborhood of each pixel. Figure 1.2.8(c) to Figure 1.2.8(g) show results obtained for different values of N.

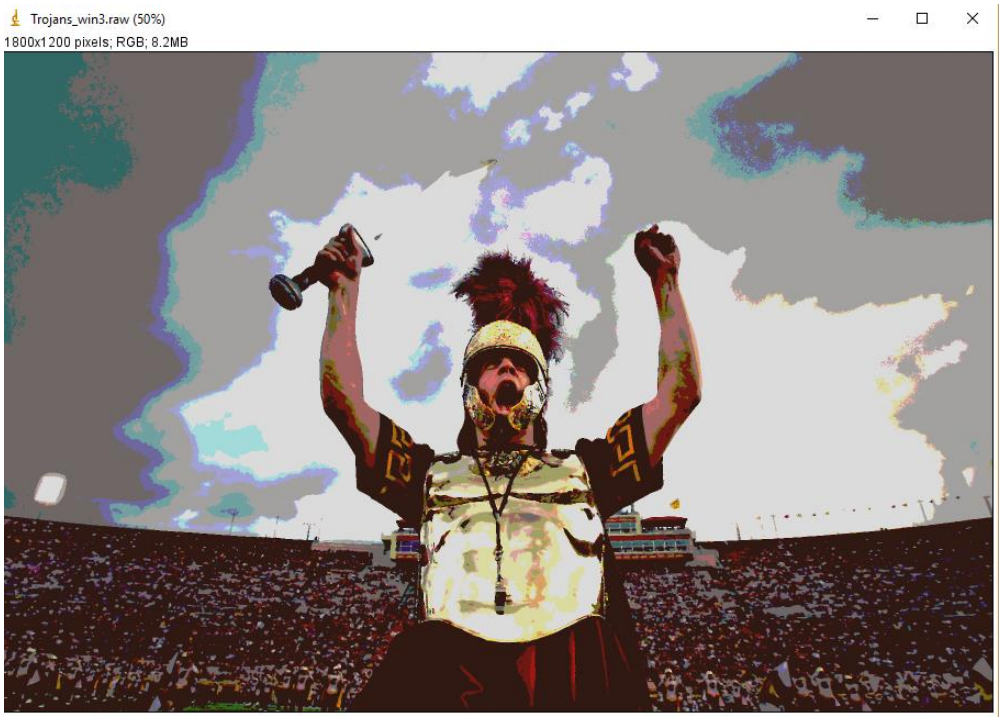


Figure 1.2.8(c) Oil – painting effect for quantized Trojans image using $N = 3$

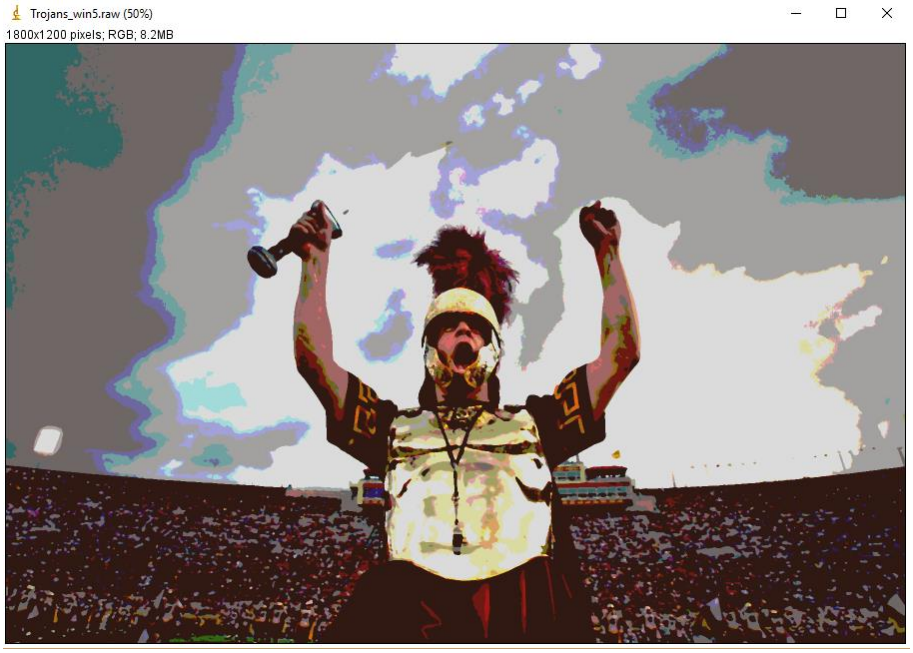


Figure 1.2.8(d) Oil – painting effect for quantized Trojans image using $N = 5$

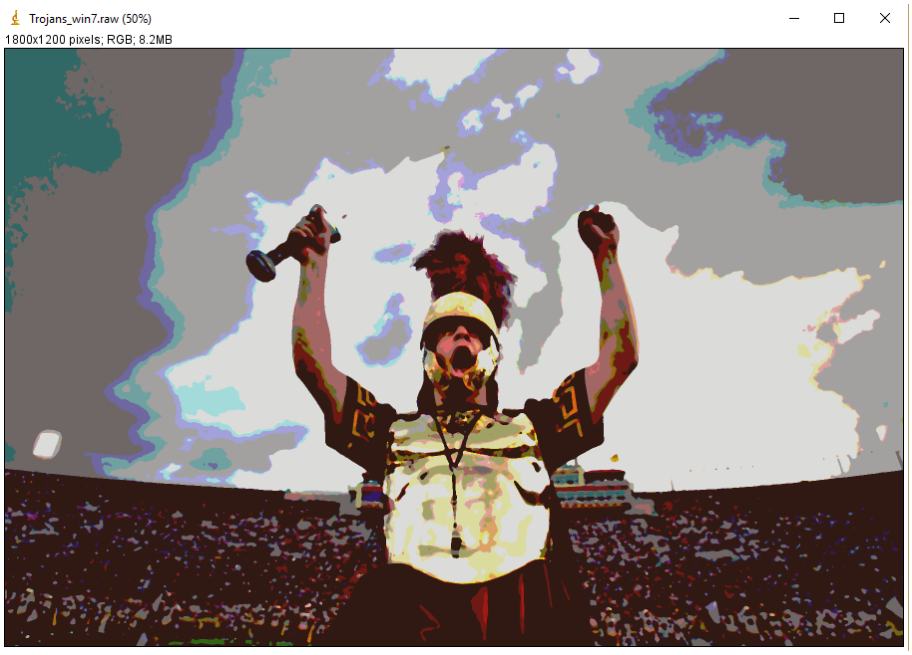


Figure 1.2.8(e) Oil – painting effect for quantized Trojans image using $N = 7$

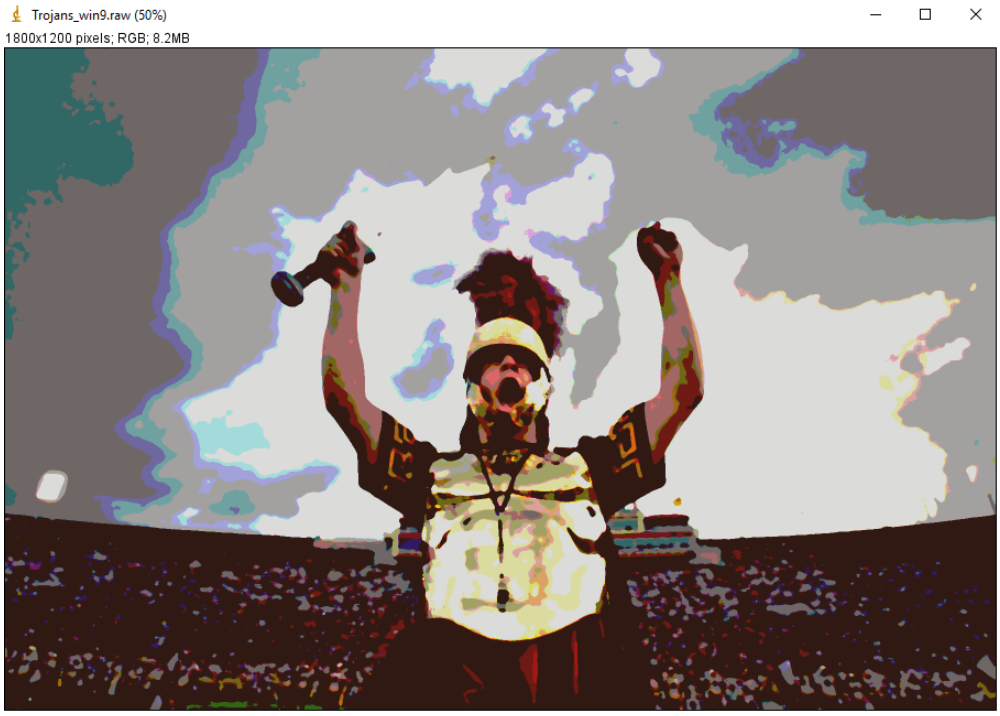


Figure 1.2.8(f) Oil – painting effect for quantized Trojans image using $N = 9$

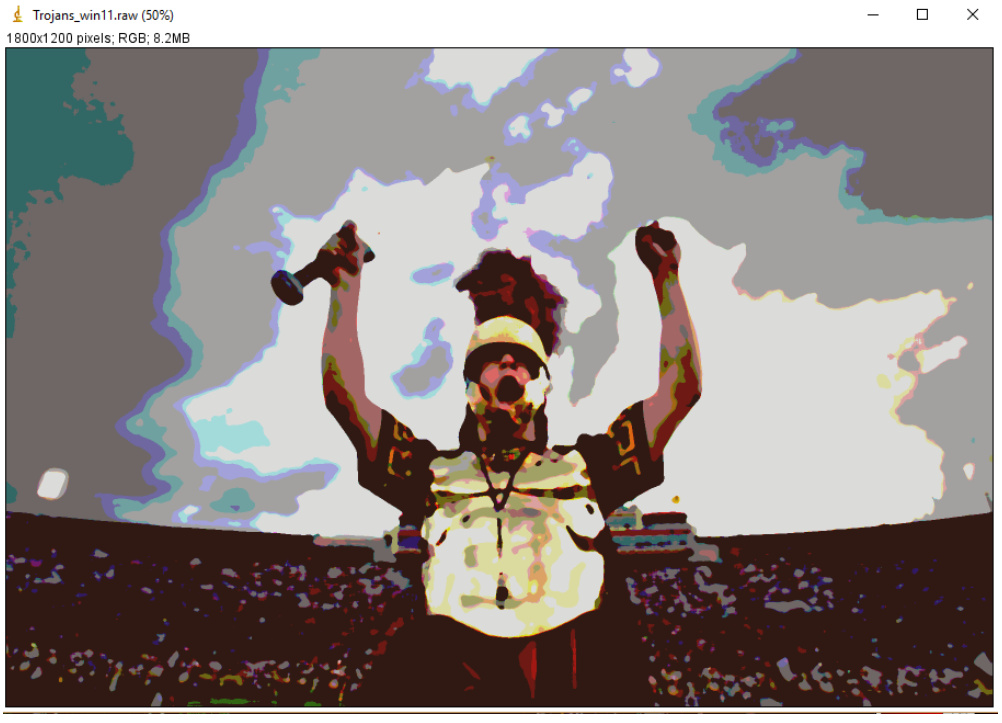


Figure 1.2.8(g) Oil – painting effect for quantized Trojans image using $N = 11$

When the input image has 512 colors instead, the output of Star_Wars.raw and Trojans image are as in Figure 1.2.9 and Figure 1.2.10 respectively.

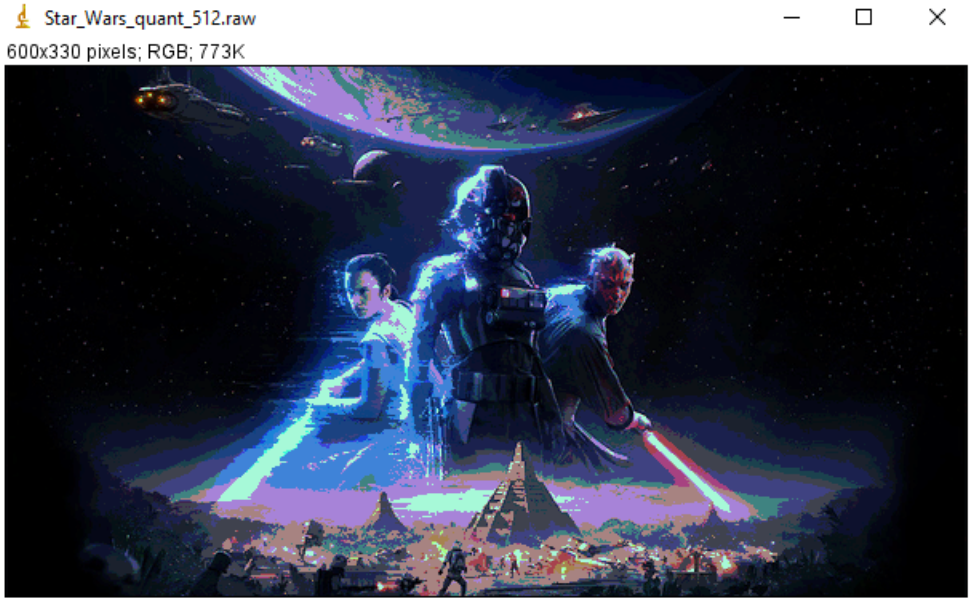


Figure 1.2.9(a) Quantized Output of Star_Wars when the input image has 512 colors

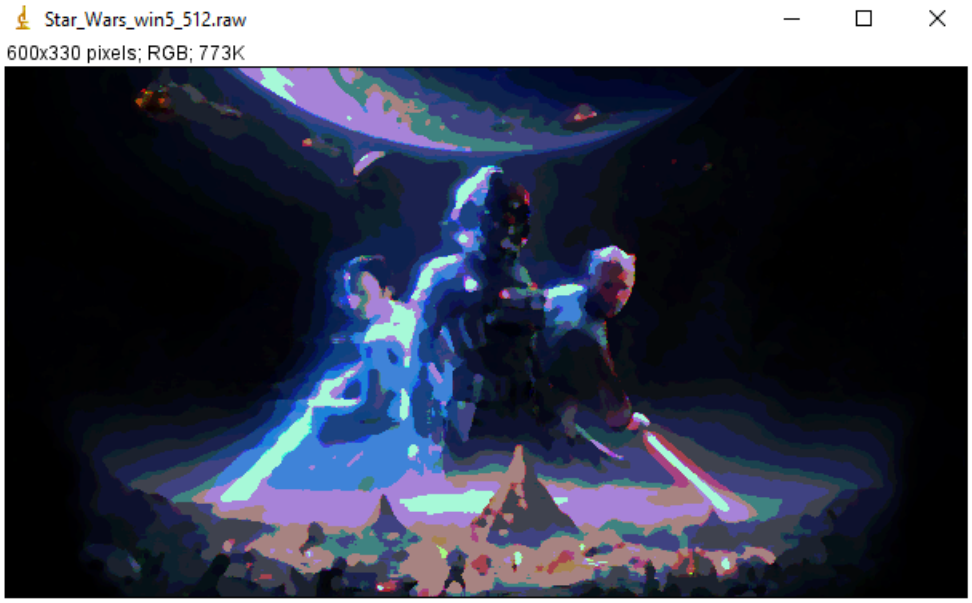


Figure 1.2.9(b) Oil filter effect for N=5 when input has 512 colors

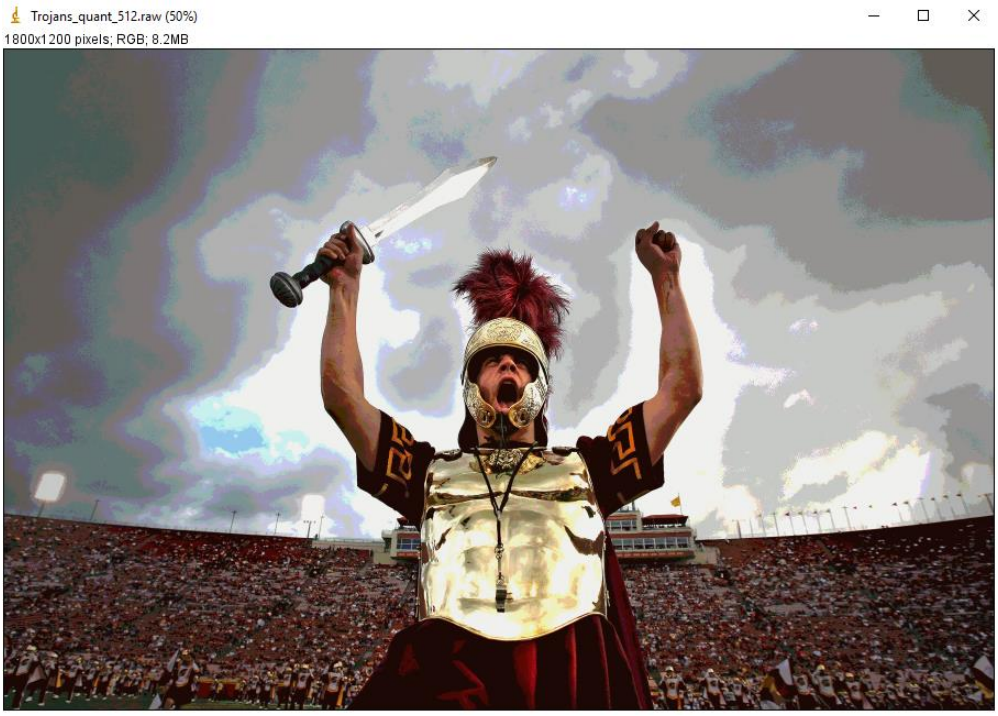


Figure 1.2.10(a) Output of Trojans when the input image has 512 colors

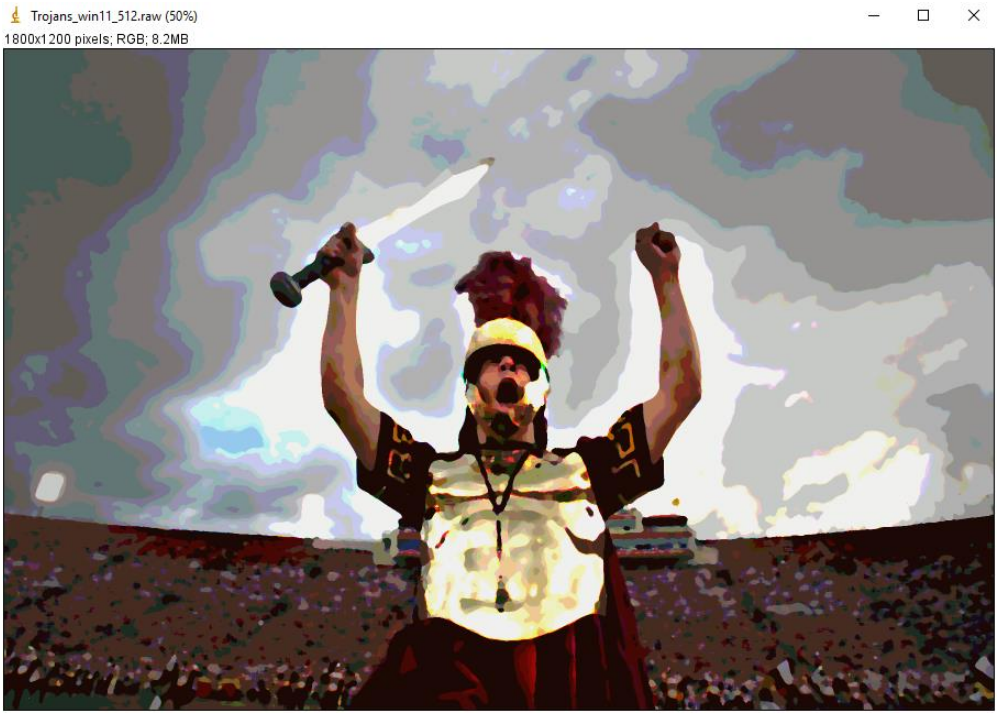


Figure 1.2.10(b) Oil filter effect for N=11 when input has 512 colors

B.4 Results

(1) The reduced color set of both the Star_Wars.raw image and Trojans.raw image is shown in the Figure 1.2.7(b) and 1.2.8(b) respectively. The threshold is chosen as follows:

- Number of ranges are found – In a 64-color set, each color has 4 values.
- Hence of the total $M * N$ values, $\frac{M*N}{4}$ pixels are assigned each value.
- Each grayscale value is chosen as the weighted sum of all the pixels. For example, if the first $\frac{M*N}{4}$ pixels occur in the actual grayscale values $0 \sim j$, then the weighted average is obtained as

$$Weighted\ Average = \frac{(0 * hist[0]) + (1 * hist[1]) + \cdots + (j * hist[j])}{hist[0] + hist[1] + \cdots + hist[j]}$$

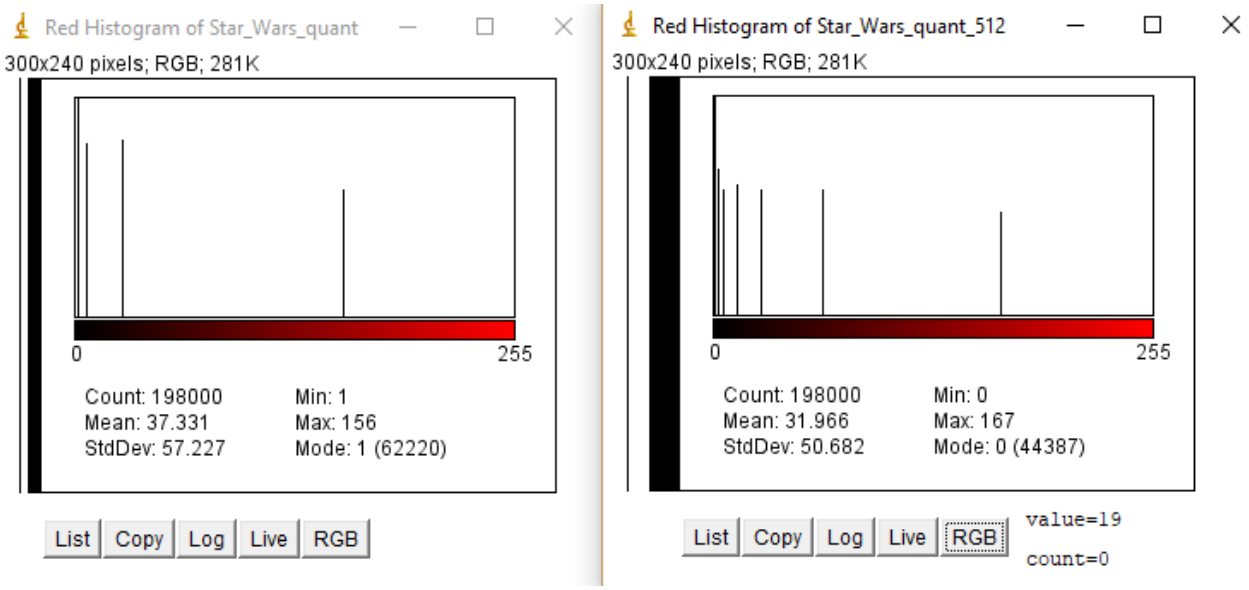
$$Weighted\ Average\ = \sum_{k=0}^{k=j} \frac{k * hist[k]}{hist[k]}$$

- Thus, all pixels in the range $0 \sim j$ is replaced by the calculated weighted average. This is repeated to find the other 3 colors of one channel, with the ranges being $(j+1) \sim m, (m+1) \sim n, (n+1) \sim 255$.

(2) Figure 1.2.7(c) to Figure 1.2.7(g) show the result of oil filter effect on Star_Wars.raw image. It is found that N=5 gives a better effect. This is because the size of the image is small, a proper size of the filter is essential. If the window size is too small, the oil filter effect will not be pronounced at all. If the window size is too big, then there will be over smudging.

Figure 1.2.8(c) to Figure 1.2.8(g) show the result of oil filter effect on Trojans.raw image. It is found that N=11 gives a better oil filter effect. Here the size of the image is big. Hence too small window size gives almost no effect. Only a considerably big window size can bring in oil filter effect.

(3) Figure 1.2.9 and Figure 1.2.10 show the outputs when the color set is 512 colors i.e. 8 colors per channel. The features are more clearly defined when 512 colors are used. This can also be checked by an example of the histogram plots of 64 color Star_Wars image and 512 color Star_wars image. 64 color set has only 4 colors on each channel and 512 color set has 8 channels, as shown below.

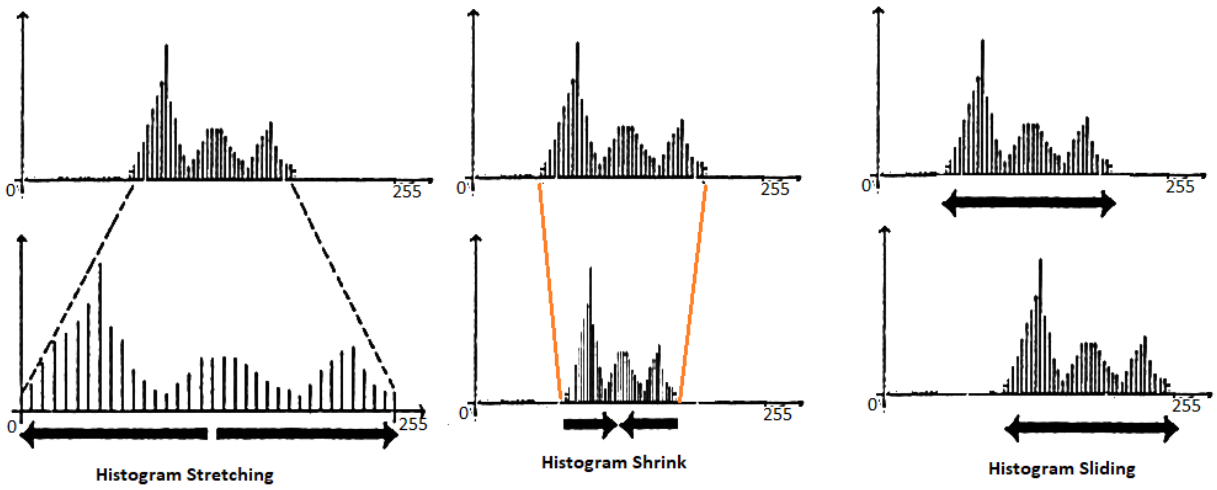


Part C – Image Filtering – Creating Special Film Effect

C.1 Motivation

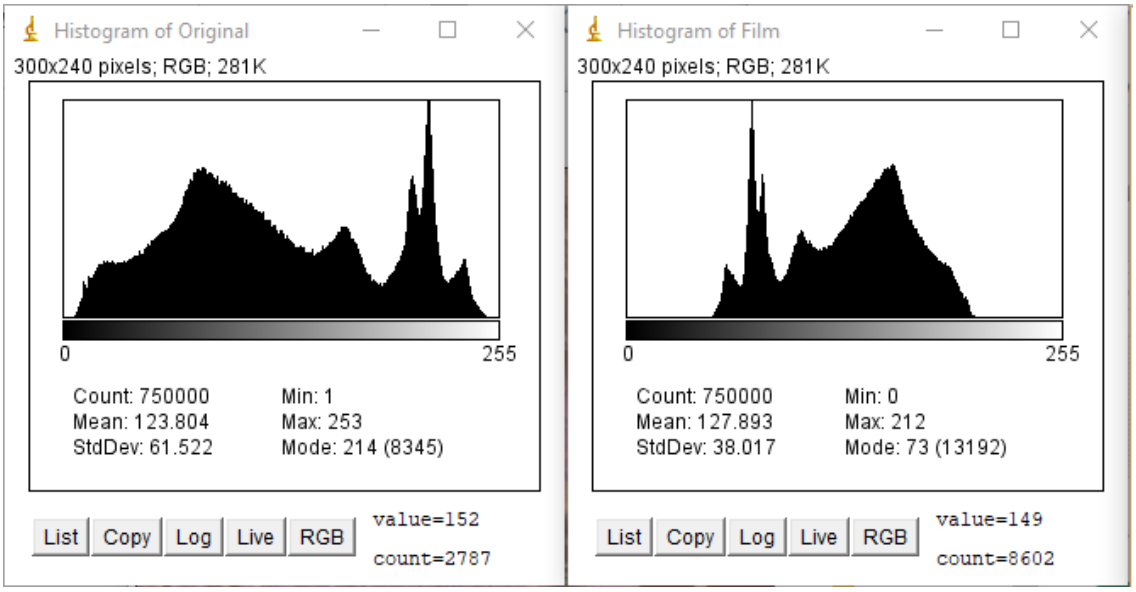
Histograms are one of the most useful tools for enhancing an image and obtaining special effects. Many useful information like contrast, brightness etc. can be obtained from the histogram plots. Contrast enhancement can be done by observing the histogram. Contrast can be defined as the difference between the maximum intensity and the minimum intensity levels of an image. A low contrast image has a narrow histogram, and the image can be made into sharp contrast (to differentiate features) by stretching the histogram. This is called histogram stretching, which is shown below.

The inverse of histogram stretching is the histogram shrink, wherein the actual histogram range is reduced. Hence there is a decrease in contrast. This method is generally not preferred since there is a reduction in visual quality, in most cases. Another operation is the histogram scaling, where the histogram is shifted either towards the brighter or darker side of the grayscale, without affecting the grayscale relationships. The following figure shows these three simple operations. Using these operations, many effects can be achieved.



C.2 Approach

Analyzing the given Original.raw and Film.raw images, the histogram of both the images are compared.



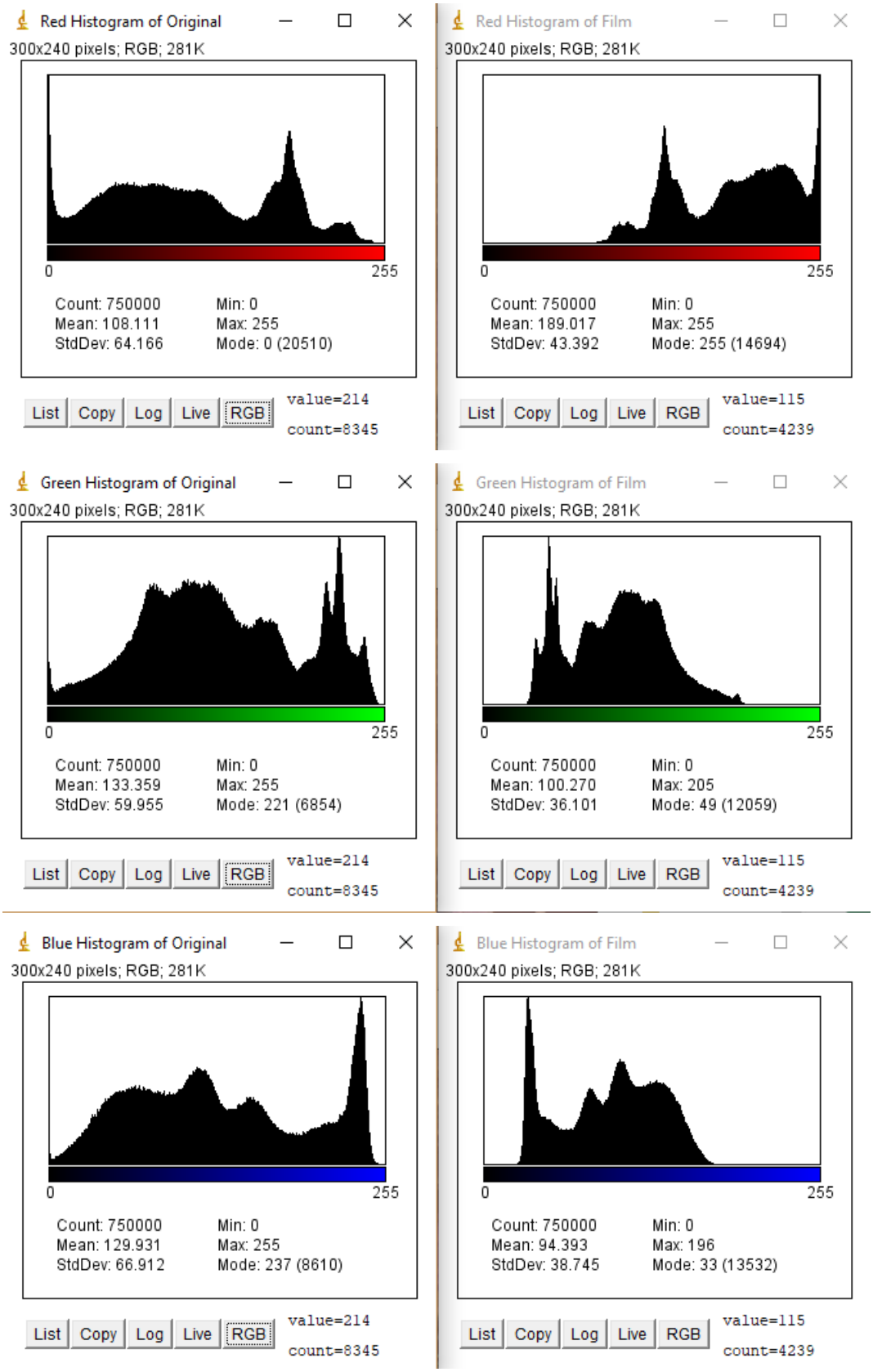


Figure 1.2.11 Histogram comparison of the given Original.raw and Film.raw images

It is observed from Figure 1.2.11 that

- Histogram of the entire image is flipped
- The color space is changed from RGB to CMY(K)
- The Red Channel in the actual image has the grayscale values 0~255, and the film image has values 0~255, but shrunk towards the dark side.
- The Green Channel in the actual image has the grayscale values 0~255, but the film image has values 0~205, plus shrink.
- The Blue Channel in the actual image has the grayscale values 0 ~255, but the film image has values from 0 to 196.

Accordingly, the following approach is done in C++.

i. The input image is obtained from the file using the file commands, and stored in an array $Imagedata$, which is of size $Size1 * Size2 * 3$.

ii. The output image array $ImageOutput$ is defined.

iii. The image is first flipped along the right boundary using the formula

$$ImageOutput[i][j][k] = Imagedata[i][Size2 - j][k]$$

iv. Then the image is converted to CMY(K) Color Space using the formula

$$C = 1 - R\ ;\ M = 1 - G\ ;\ Y = 1 - B$$

v. On the Channel, Histogram Shrink for each pixel $I(x, y)$ in a channel is applied using the formula

$$Shrink\ I(x,y) = Shrink_{min} + [I(x,y) - I(x,y)_{min}] * \left[\frac{Shrink_{max} - Shrink_{min}}{I(x,y)_{max} - I(x,y)_{min}}\right]$$

Where $I(x,y)_{min}$ = minimum gray scale value in the actual image

$I(x,y)_{max}$ = maximum gray scale value in the actual image

$Shrink_{max}$ = maximum gray scale level desired in the Output image

$Shrink_{min}$= minimum gray scale level desired in the Output Image

vi. The same procedure is repeated for the rest of the channels, if necessary.

vii. The output image is stored as a separate .raw file.

C.3 Result

The actual Film.raw image (in Figure 1.2.12(a) is analyzed, and the output is shown in Figure 1.2.12(b).

Figure 1.2.12(a) Actual Original.raw image

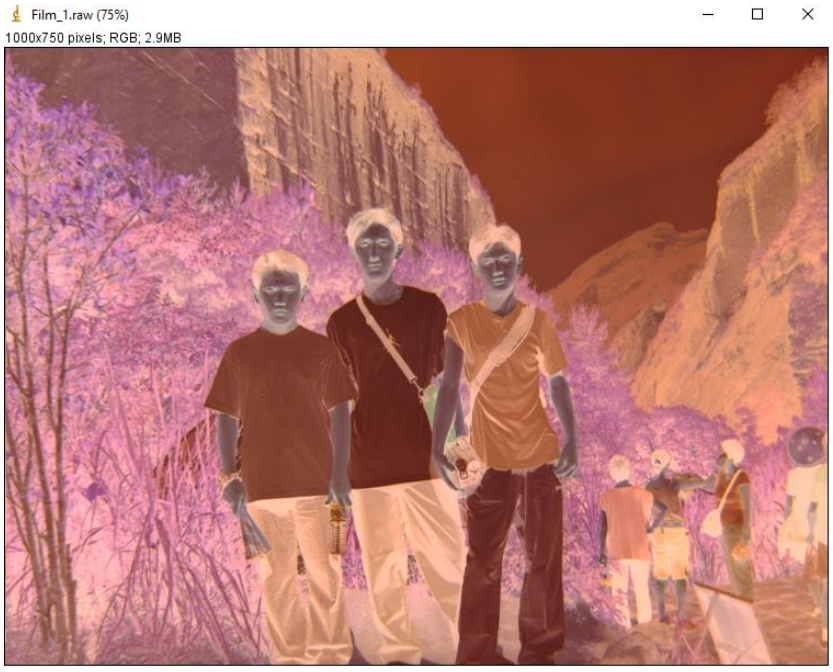


Figure 1.2.12(b) Implemented Film effect on the Original.raw image

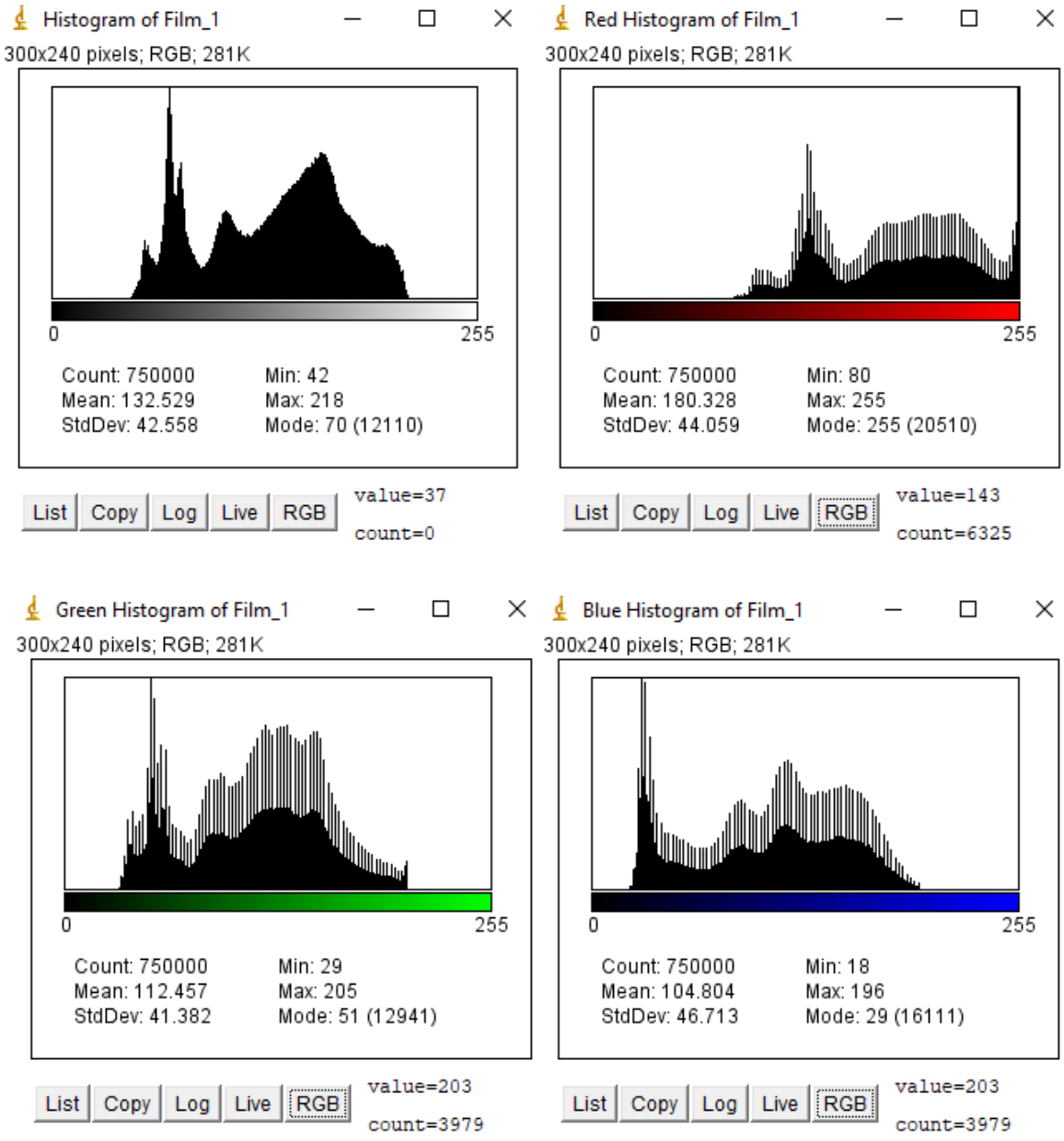
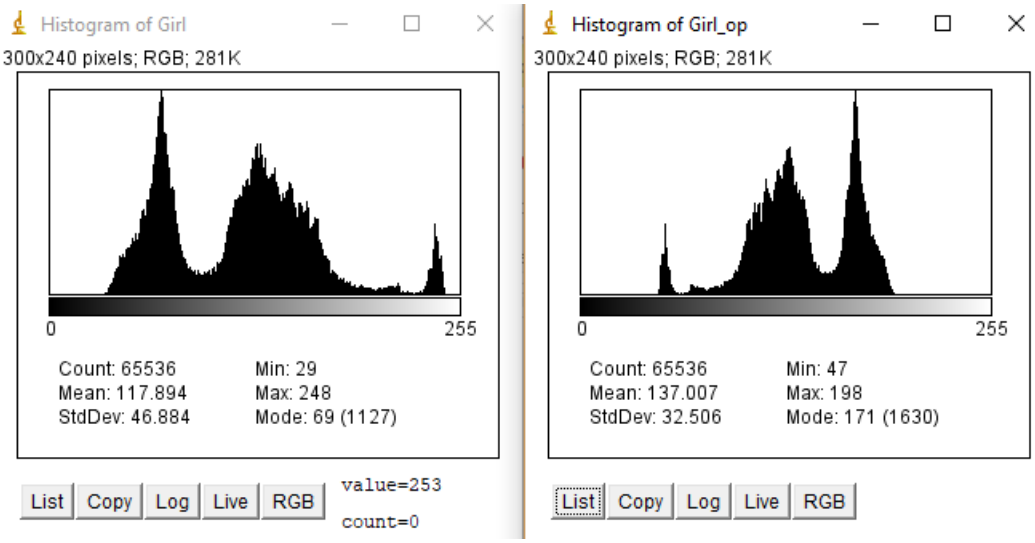


Figure 1.2.13 Histogram plots of the obtained Film effect using the algorithm

The input Girl.raw image is shown in Figure 1.2.14(a). The corresponding special film effect is shown in Figure 1.2.14(b).

Figure 1.2.14(a) Actual Girl.raw image **Figure 1.2.14(b) Film effect on the Girl.raw image**

The corresponding histograms obtained for both the images is shown in Figure 1.2.15.



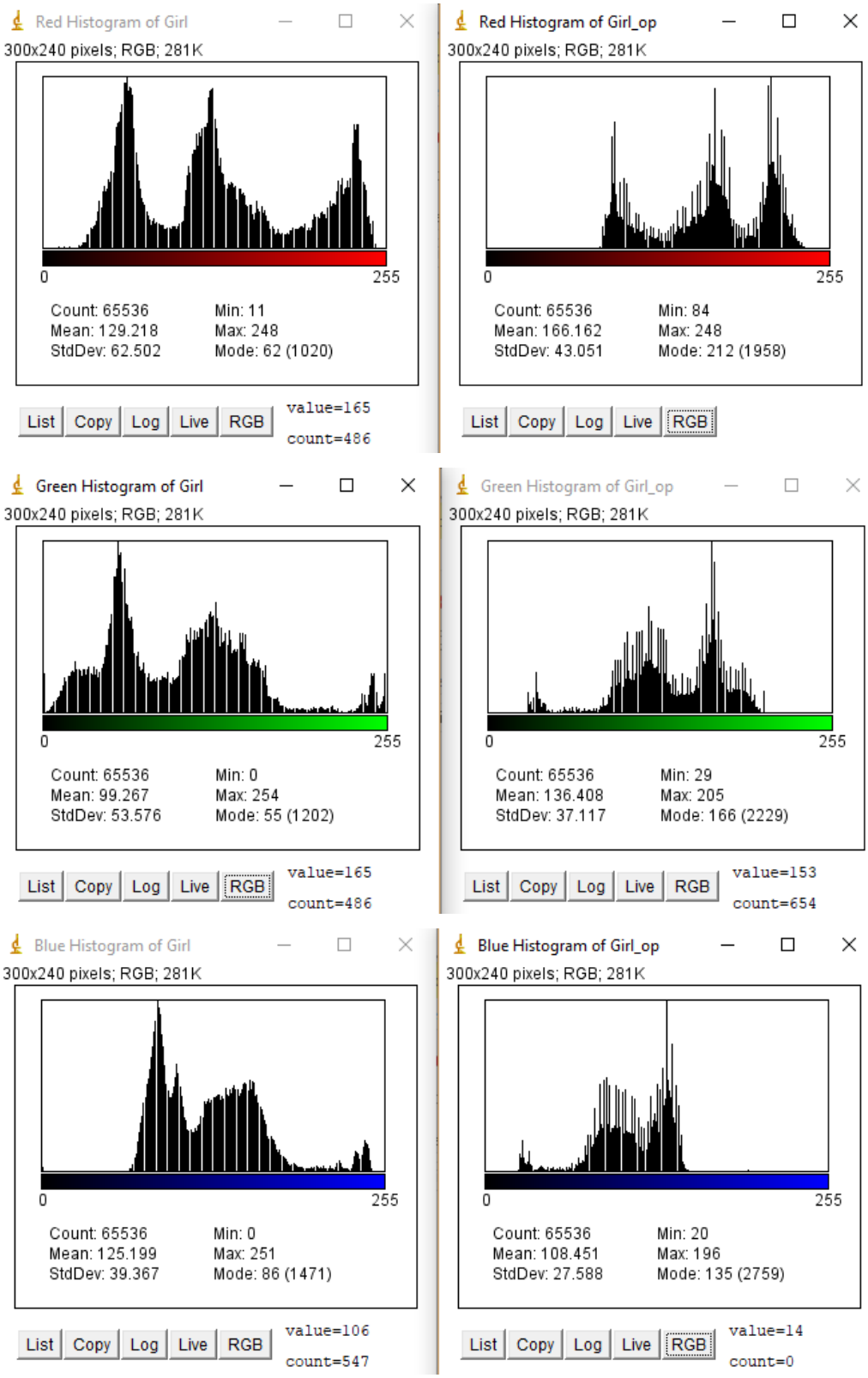


Figure 1.2.15 Comparison of the histogram plots of the actual Girl.raw image and the obtained Girl_op image using the algorithm

C.4 Discussion

The algorithm to achieve the special film effect is as follows:

1. Start
2. Get Input Image
3. Flip the image on its right boundary i.e. flip as if a mirror is on the right margin of the image
4. If the image is in RGB Color Space, convert the image into CMY(K) space
5. Apply the required Histogram Modifications to each individual channel –
6. Output the final image
7. Stop

Using this algorithm, the special film effect can be obtained for any image. In the given Film.raw image, this effect is more prominent because the contrast is high. In the Girl.raw image, the variance of colors is comparatively less, hence the obtained output is of less contrast.

Reference

[1] William K. Pratt, “Digital Image Processing”, Second Edition, John Wiley & Sons, 2001.

[2] Rafael C Gonzalez, Richard E Woods, “Digital Image Processing “, Second Edition, Prentice Hall.

[3] http://www.uky.edu/~kiernan/eBeo_archives/articles90s/ksk-llc.htm

---

Problem 3 – Noise Removal

Part A – Mix Noise in Color Images

A.1 Motivation

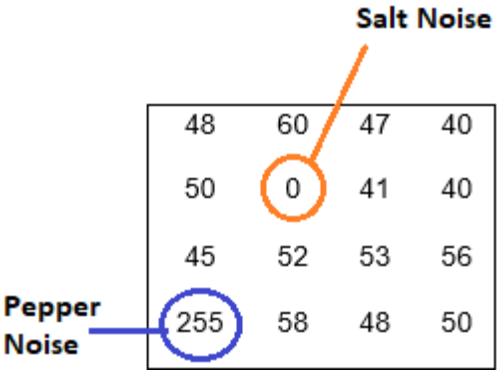
Image Denoising is one of the basic processes in Image Processing. The probability of an image without noise is zero. The sensors pick up data from the real world and convert them to an image. The sensors introduce some type of noise into the data. Noise can be defined as any undesired information that contaminates an image. There could be several types of noise added to the image, like additive noise, impulse noise etc. For each kind of noise, a different kind of filter should be used to achieve maximum retention of information. One measure to check the quality of

denoising is the calculation of Peak Signal to Noise Ratio(PSNR). In short, the goal of denoising is to remove as much noise as possible, retaining all essential pieces of data.

A.2 Approach

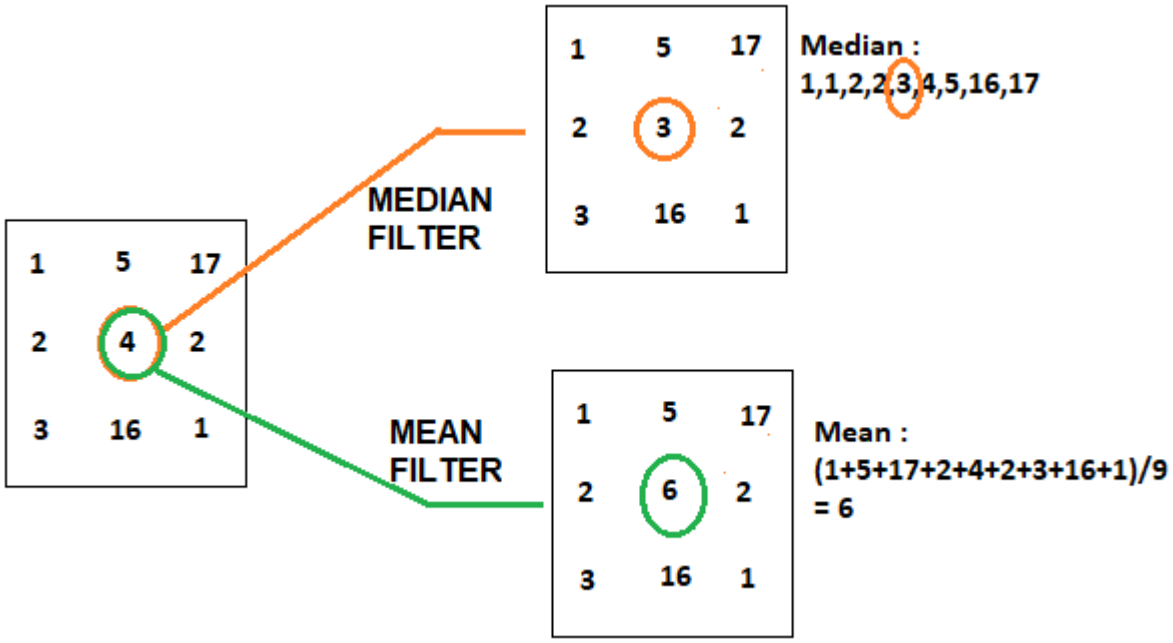
The given image is identified for the type of noise it has. From the histogram of the given image, it is observed that the given image Lena_mixed.raw has two types of noise – Salt and Pepper Noise and Gaussian Noise.

Salt and Pepper noise is an impulsive noise, which is caused due to sudden disturbances on the input signal. Here, noisy pixel value on the grayscale, goes to a value of either 0 or 255. If it goes to 0, it is called a salt noise, since it is white. If it goes to 255, it is called a pepper noise, since it is black.



Another possible noise is the additive white Gaussian Noise caused by poor data acquisition or passing the data through high noise channels. The two major approaches to denoising an image are Spatial domain filtering and frequency domain filtering methods. Spatial filters assume noise as high frequency and do low – pass filtering on the input data. This may create blurring, whereas high pass filters can make the edges sharper. The two types of filters used here are the median filter and the mean filter.

Median filter is a non – linear filter that smoothens an image by using the median of the $N * N$ neighborhood. Every pixel in the N*N neighborhood is replaced by the median in the neighborhood. Mean filter or the averaging filter replaces each pixel by the mean of the pixels in its N*N neighborhood. An example is shown below.



The approach used in C++ is as follows.

i. The input noisy image is obtained from the file using the file commands, and stored in an array $Imagedata$, and the noise free image is obtained in the array $ImageNoiseFree$.

ii. The output image array $ImageOutput$ is defined.

iii. The mean filter and median filter are square operations, since they work on $N * N$ neighborhoods. Each of these filters have a specific window size, and are coded as separate functions.

iv. Then the different combinations of median filter and mean filter for different window sizes are analyzed.

v. For each filter combination, the PSNR Values for the image size $M * N$ are calculated for each channel as follows.

$$PSNR(dB) = 10 * log_{10}\left(\frac{Maximum\ gray\ scale\ value^2\ i.e\ 255^2}{MSE}\right)$$

$$MSE = \frac{1}{M * N}\sum_{i=1}^{N}\sum_{j=1}^{M}((ImageOutput(i,j) - ImageNoiseFree(i,j))^2$$

vi. The combination with the maximum PSNR is a good denoising combination.

A.3 Results

The results are tabulated as follows. Median filter is applied only to R and G channels, where salt and pepper noise is present. Gaussian noise is present in all three channels. Hence mean filter applies to all the three channels.

S. No	Filter Combination	Window Size of Median Filter	Window Size of Mean Filter	Average PSNR (dB)
1	Median	3	-	24.852
2	Median	5	-	23.523
3	Median	7	-	25.82
4	Median	9	-	25.64
5	Mean	-	3	15.71
6	Mean	-	5	13.887
7	Median → Mean	3	3	26.0039
8	Median → Mean	3	5	15.14
9	Median → Mean	5	3	26.443
10	Median → Mean	5	5	25.007
11	Mean → Median	3	3	24.27
12	Mean → Median	3	5	15.23
13	Mean → Median	5	3	16.113
14	Mean → Median	5	5	24.73
15	Median → Mean → Median	3	3	17.52

Figure 1.3.1 shows the noisy Lena_mixed.raw image.

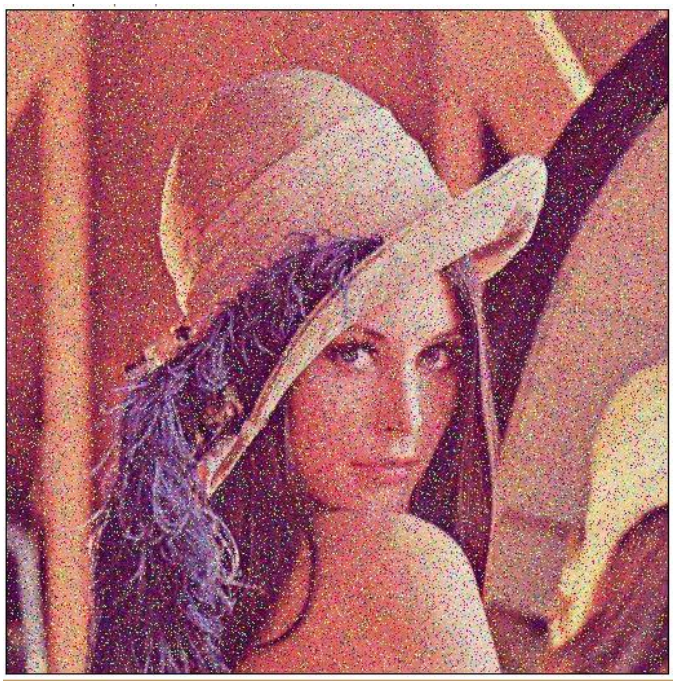


Figure 1.3.1 Lena_mixed.raw – Noisy image

Figure 1.3.2 shows the result of median filtering of the Lena_mixed.raw input image using a window of size 3*3.

Figure 1.3.2 Median filtering only, Window size = 3*3

Figure 1.3.3 shows the result of mean filtering of the Lena_mixed.raw input image using a window of size 3*3.

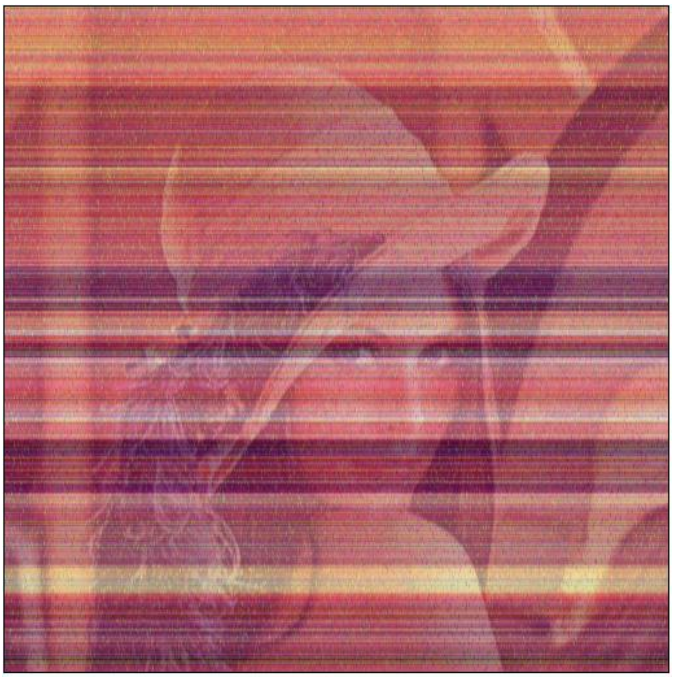


Figure 1.3.3 Mean filtering only, Window size = 3*3

Figure 1.3.4 shows result of using median filter, followed by mean filter, both of window size 3*3.

Figure 1.3.4 Median filter + Mean filter, each of window size 3*3

Figure 1.3.5 shows the output obtained for Median filter followed by the Mean filter combination, where the window size for the median filter is 5 and the mean filter is 3.

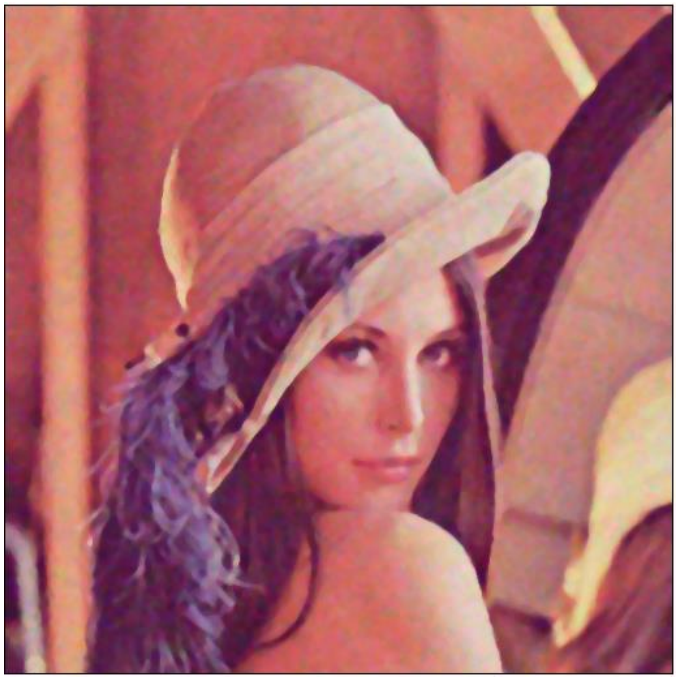


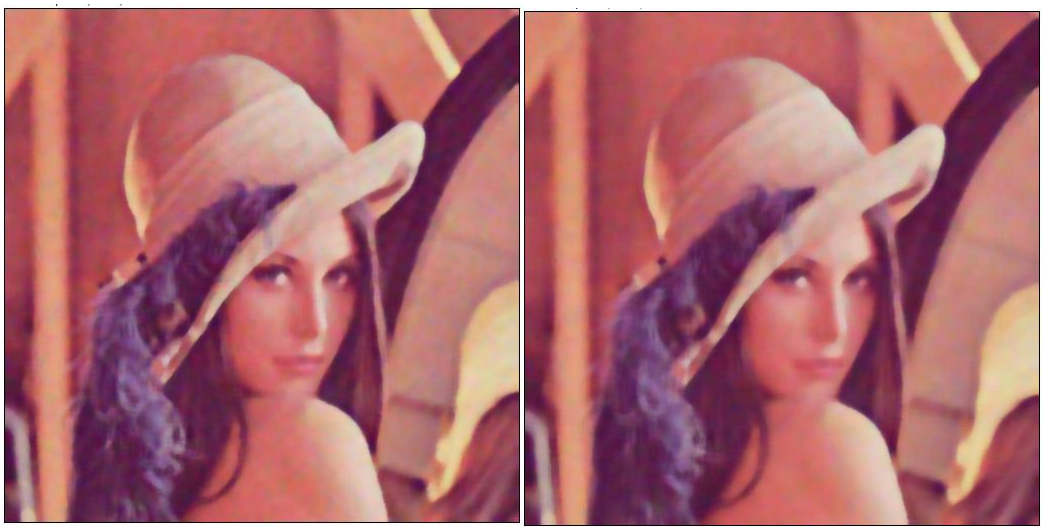
Figure 1.3.5 Median filter of window size 5*5 followed by Mean filter of size 3*3.

Figure 1.3.6 shows an image poor PSNR value, for the combination of Mean filter of window size 3*3, followed by the median filter of size 5*5.

Figure 1.3.6 Mean filter of size 3*3, followed by median filter of size 5*5.

Figure 1.3.7 shows the effect of median filtering for various window sizes.

(a) Median filter of size 5*5 **(b) Median filter of size 7*7**

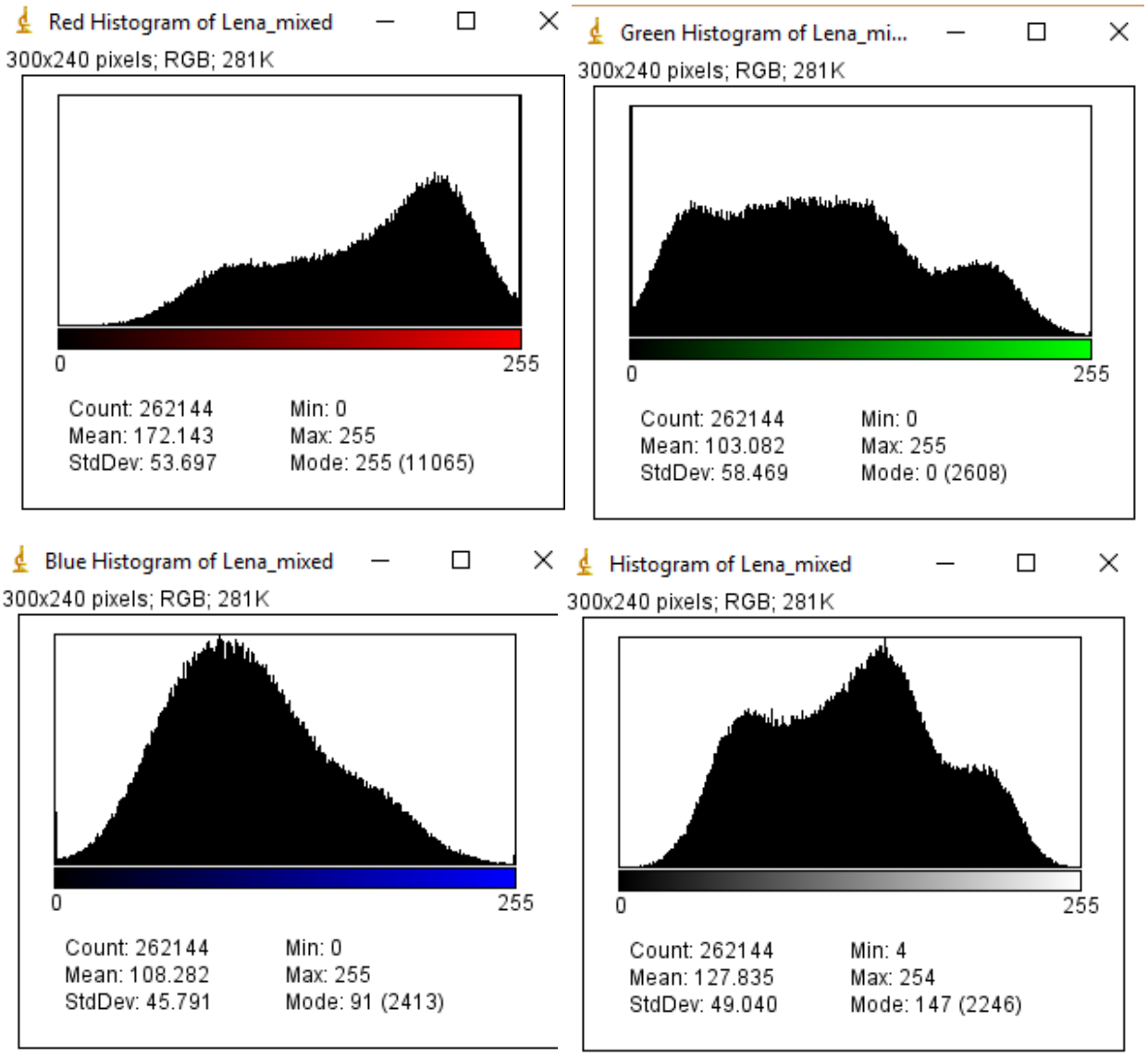


(c) Median filter of size 9*9 **(d) Median filter of size 11*11**

Figure 1.3.7 Median filters for different Window Sizes

A.4 Discussion

(1) In the noisy Lena_mixed.raw image, all the channels do not have the same noise type. Observing the histogram plots of Lena_mixed.raw,



It is observed that Salt and Pepper noise is predominant in the Red and Green channels, which is inferred from the value of the mode (Grayscale value 255 (R) and Grayscale value 0(G)). The blue channel has Gaussian Noise, indicated by the shape of the histogram. Here, I implemented filters for all the three channels together – Median filter for the entire image or mean filter for the entire image. But if individual filtering out of noises for each channel is done, it will give comparatively better results, since the data in the other channels is not affected by smoothening. The mixed noise in the given image is Salt and Pepper Noise, and Gaussian Noise. Salt and Pepper Noise can be removed by Median Filter, which zones out the peak values. Mean filter can be used to reduce the Gaussian Noise.

To get better results, cascading of the filters is important. The better cascading option will be Median filter followed by the Mean filter for improved PSNR values. First, the median filter removes the salt and pepper noise, replacing the pixel value by the median value. Then the mean filter reduces the Gaussian Noise. Considering the other way, mean filter followed by median filter, there will be an increase in darkness or brightness of the image, since the peak values are also averaged in the mean filter. Hence the cascade of median filter, followed by mean filter works fine. The window size of the filter is also a key point in denoising. For smaller window sizes, the filter works fine. Both median and mean filter works fine. As the window size increases, the filter over smoothens the image, making it look blurred, as seen in Figure 1.3.7.

(2) The combination of filters that I tried is listed in Table 1. Of the filter combinations, the one with Median filter followed by the Mean filter gives better results. Improved PSNR is obtained when the median filter is of window size 5*5 and the mean filter is of size 3*3. The shortcoming of this method is that the PSNR value is very close to the other window sizes as well, so a similar filter combination will work fine. Another issue is that the filter is applied to the entire image, not channel by channel. Hence there are chances of data being misinterpreted. To improve its performance, the filter of individual window size can be applied individually to each channel.
Thus, image denoising is implemented.

Part B – Principal Component Analysis(PCA)

B.1 Motivation

Principal Component Analysis(PCA) is a dimensionality Reduction algorithm, yet it is a useful tool for denoising an image. PCA can be employed to reduce Additive White Gaussian noise in the image, but cannot eliminate noise fully. Instead of removing the noise pixel by pixel, some pixels are grouped into a patch, and an image is divided into many patches. Operations are done on the patches. These operations are done within the patch boundaries, wherein hard thresholding is done on the co- efficients. PCA is an orthogonal transformation that is used to find projection of data into dimensions that have the greatest variance. PCA uses orthogonal transformations to denoise an image.

B.2 Approach

PCA involves algorithms that do hard thresholding i.e. the threshold values are hard – coded and not adaptive. Processing a group of pixels poses several advantages like

- Identification of a pattern
- Dimensionality reduction etc.

The major steps in PCA Denoising are

- Identification of an orthogonal basis from the noise image by PCA
- Decomposition of the patch in its basis
- Nullifying the smaller noisy components
- Obtain the denoised patch

PCA can be implemented in three ways of obtaining input patches – Local PCA, Hierarchical PCA and Global PCA. Local PCA chooses patches in a localized boundary i.e. of some window size W*W, and then does PCA. Hierarchical PCA first divides the entire image into four quadrants, then to 16 blocks, until some j partition levels. Then PCA is done. Global PCA takes the entire image as a whole, then PCA is done.

The algorithm in short can be explained as follows

i. Patch based Global PCA : Extract small sub patches , each patch is a vector consisting of n elements. Compute the n*n covariance matrix. Then project n vectors into a smaller dimensionality space.
   PCA actually projects the larger k dimensional set of vectors into a smaller space with m dimensional vectors. While projecting, the most significant set of vectors are retained. Noise is usually in the least significant set of vectors. Hence projection using PCA reduces the noise components.
ii. Patch based Local PCA : The algorithm is the same as above, except that the patches are taken into consideration.

This PCA is implemented in Matlab with reference to reference [6].

B.3 Results

Using Matlab, Patch based PCA is analyzed. The input image is House.raw, as in Figure 1.3.8.

Figure 1.3.8 Input House.raw image

Using Matlab, Additive white Gaussian noise of standard deviation 25 is added. Then PCA algorithm is applied to denoise the image.

Figure 1.3.9 shows the Noisy Image, Figure 1.3.10, Figure 1.3.11 and Figure 1.3.12 show PGPCA, PLPCA and PHPCA implementations respectively.

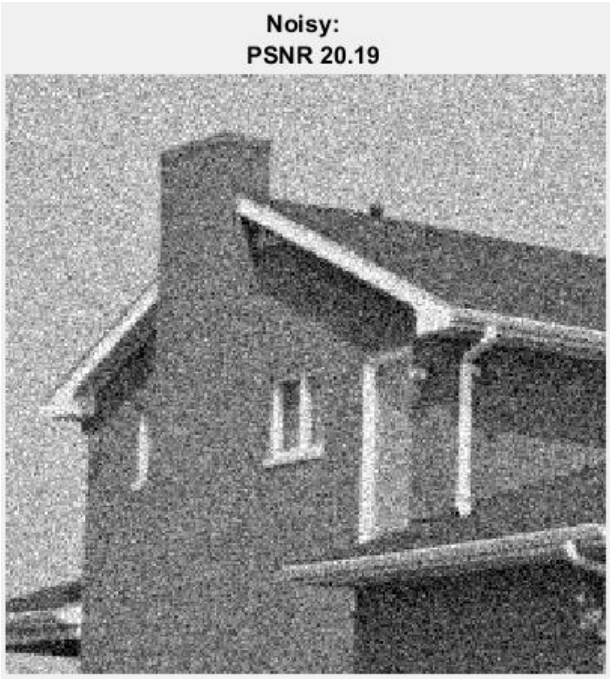


Figure 1.3.9 Noisy Image

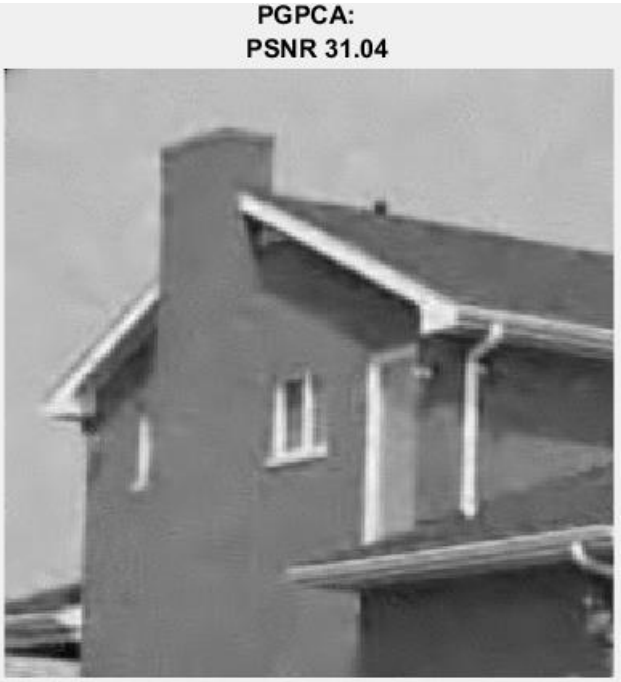
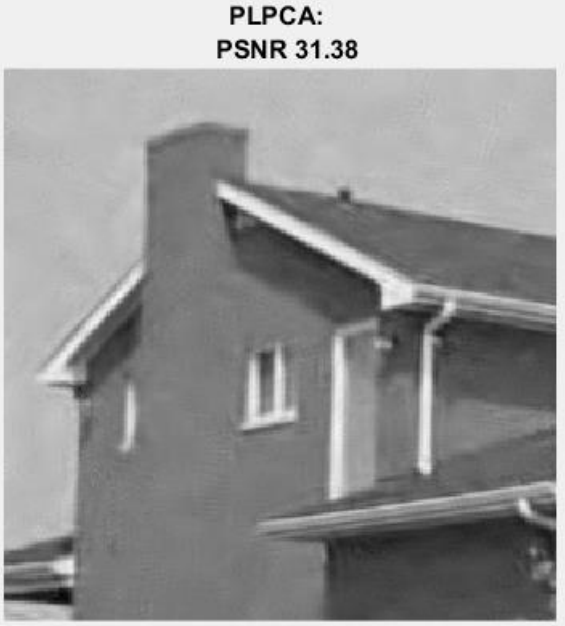


Figure 1.3.10 PGPCA Implementation



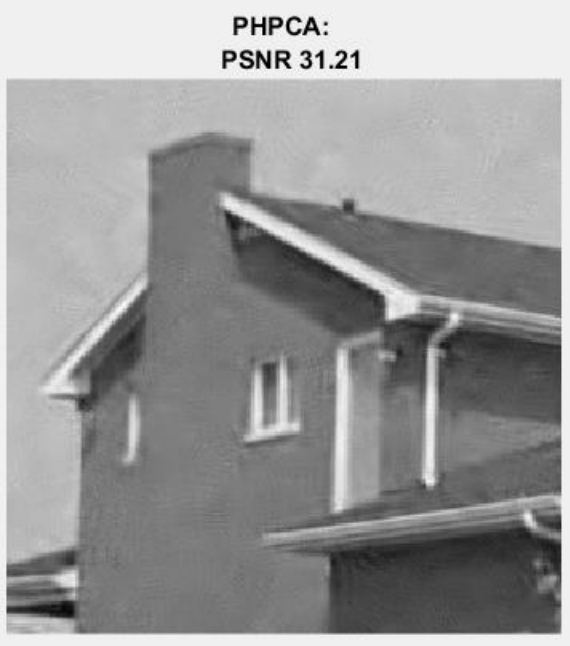


Figure 1.3.11 PLPCA Implementation **Figure 1.3.12 PHPCA Implementation**

B.4 Discussion

(1) PCA is a dimensionality reduction algorithm. PCA actually projects the larger k dimensional set of vectors into a smaller space with m dimensional vectors. While projecting, the most significant set of vectors are retained. Noise is usually in the least significant set of vectors. Hence projection using PCA reduces the noise components. The major components in implementing a PCA are the threshold, window size and the patch size. They should be chosen in such a way that the PSNR is high.

(2) The algorithm is explained as in the Approach to the problem. This is implemented using Reference[6].

(3) House.raw image is implemented for PCA analysis. The threshold is kept constant, and the size of the window and the size of the patch are varied. Different combinations are tried for values of size of patch and size of the window. σ is kept constant at 25, and threshold is kept constant at 2.75. The combination that gives the maximum PSNR is chosen.The results are tabulated as follows.

Half the size of patch Hp	Half the size of window Hw	PSNR(dB)
7	11	15.1
9	20	31.34
10	**23**	**31.38**

10	20	31.35
11	20	31.35
11	21	31.33

Hence the best parameters to obtain the maximum PSNR are patch size = 10 and window size 23.

(4) Applying the Denoising effect using filters to noisy House.raw image, the PSNR is obtained as 23.4 dB, using a median filter combination and a mean filter combination. This is definitely lesser than that obtained using PCA.The advantages of using PCA are

- Simpler approach
- Reduction in dimensions
- More similar patterns
- Better Noise reduction

Part C – Block Matching and 3D Transform Filter (BM3D)

C.1 Motivation

Noise can be introduced at any point in the image processing system. There could be various kinds of noise added to the image, like additive noise, impulse noise etc. For each kind of noise, a different kind of filter denoises efficiently. The most common type of noise added to any image is the Additive White Gaussian Noise(AWGN). One state – of – art technology to denoise images corrupted by AWGN is Block Matching and 3D transform Filters(BM3D). Currently, BM3D is the most efficient filter in denoising the Gaussian Noise.

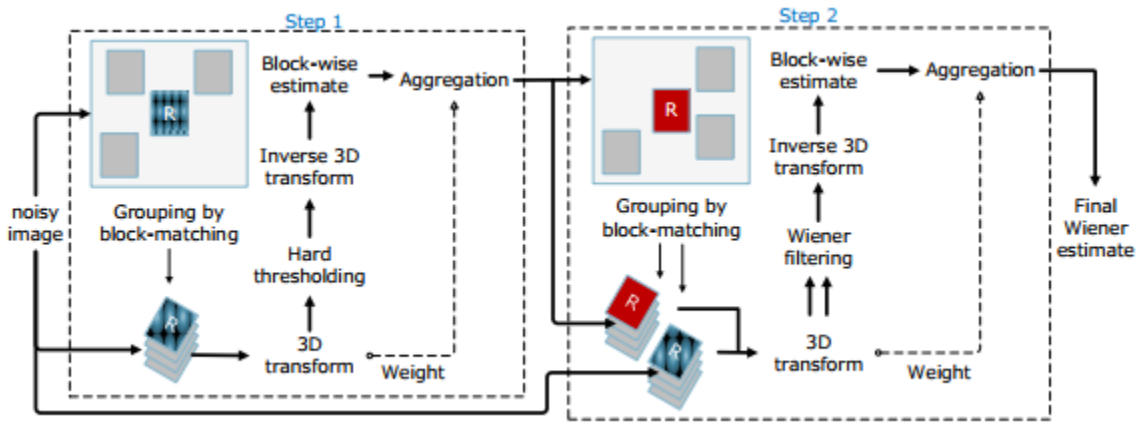
C.2 Approach

BM3D assumes that the image is locally sparse in the frequency domain. There are similar groups or patches that show up similar patterns. BM3D follows collaborative filtering approach, which can be explained as:

i. Each reference patch is scanned for any similar blocks and stacked as a 3D array.
ii. Then spatial domain to frequency domain transformation is applied. The noise components are attenuated.

iii. Inverse 3D transform is applied to attenuate noise. The estimate is based on aggregation of pixels in the actual position.

The architecture of the BM3D algorithm can be explained as follows [2].



There are two stages in denoising using BM3D algorithm.

This algorithm is implemented in Matlab using the source code given by [3].

C.3 Result

BM3D is implemented using the Matlab Code. The input image is the house image added with noise, as shown in Figure 1.3.13.

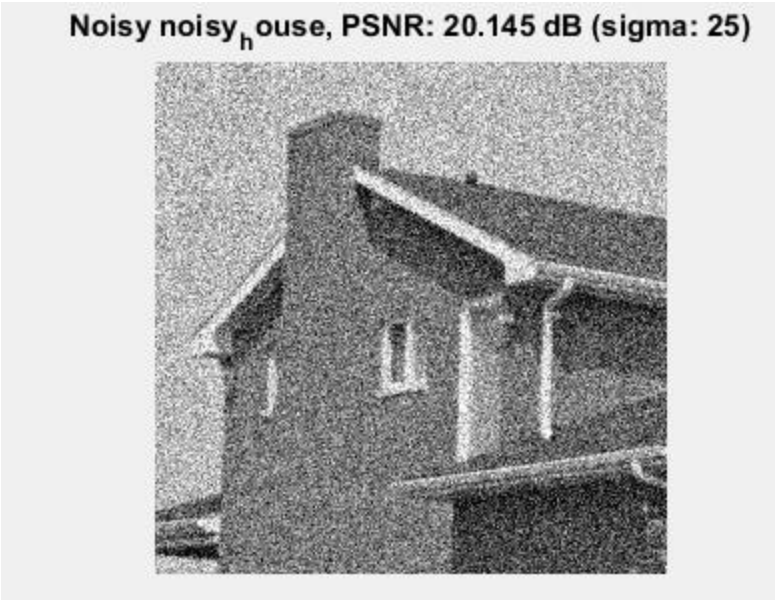


Figure 1.3.13 Noisy_House.jpg

The output of the noisy image denoised using BM3D filter is shown in the Figure 1.3.14.

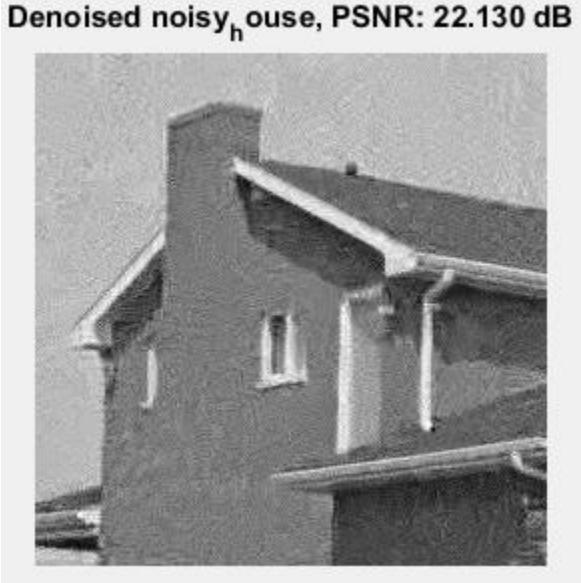


Figure 1.3.14 Denoised Image

C.4 Discussion

(1) BM3D Algorithm:

There are two steps: first using hard thresholding and second using Wiener filter.

i. The input image is divided into many smaller reference blocks. Then for each reference block, other blocks having patterns like the reference blocks are stacked together as a 3D array. Then 3D transform is applied on the blocks that converts into frequency domain. The noise components are removed by hard thresholding. Then inverse transform is applied where in each pixel might be in several different blocks. Aggregation of pixels is done to get an estimate of the pixel by taking the weighted mean of all the values. Now this is the basic estimate, wherein the actual input noisy image is denoised.

ii. The second step is exactly like the first step, except that Wiener filter is used instead of hard thresholding.

This algorithm is explained in the Figure above.

Some tunable parameters that need to be considered is the Wiener filter parameters. Wiener filter requires the use of a variance parameter of the noise that is being added. This variance parameter

can have a large effect on the filter output. Ref [4] gives a detailed explanation of the effect of the tuning parameter on various metrics.

```
%%%%  Specify the std. dev. of the corrupting noise
%%%%
if (exist('sigma') ~= 1),
    sigma               = 25; %% default standard deviation of the AWGN
end
```

The above figure shows the tunable parameter.

Some other examples of tunable parameters include size of patches in the first and second step, maximum number of similar patches, search window size, maximum threshold distance between the patches.

(2) Since BM3D involves a transformation in the 3D domain, there is a need that the input of a 3D transform should be three dimensional. This might be like the Non- Local Means algorithm. Moreover, if the pixels are taken individually, it becomes very difficult to find a pattern that is similar. If two pixels have the same value, that does not mean they are similar. Only if a region is defined, that could be called a block, where all pixels in that block share some similarity. In the first stage, hard thresholding is used, and in second stage, Wiener filtering is used. This step is called hard thresholding because it used a fixed threshold value to attenuate the noise.

Comparing the denoising from the output of hard thresholding, and

(3) BM3D is both a spatial domain and a frequency domain filter. The input noisy image is in spatial domain. The groups are also in spatial domain, then the 3D filter is applied, which converts it into frequency domain.

(4) Comparing the algorithms for PCA and BM3D, BM3D is preferred because a considerable amount of attenuation of noise is achieved. PCA based methods also consume lots of resources.

BM3D algorithm relies on both the local and non – local patches of data, and assumes that the data is highly correlated. If all these characteristics are satisfied, there is a high correlation. This effect of high correlation is evident in the proportionate conversions from the spatial domain to frequency domain and back into the spatial domain. But according to experiments [5], the actual signal can be separated from noise when the compact energy representation of the image is evident in the frequency domain. There are also some disadvantages in the BM3D filtering when

it comes to square patches containing very finite details, sharp edges or curves, from which a sparse representation cannot be derived.

References

[1] William K. Pratt, “Digital Image Processing”, Second Edition, John Wiley & Sons, 2001.

[2]https://ir.lib.uwo.ca/cgi/viewcontent.cgi?referer=https://www.google.com/&httpsredir=1&article=3508&context=etd

[3] Matlab Source Code for BM3D algorithm : http://www.cs.tut.fi/~foi/GCF-BM3D/

[4] http://alumni.soe.ucsc.edu/~xzhu/doc/metricq.html

[5] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian , “BM3D Image Denoising with Shape-Adaptive Principal Component Analysis”, http://www.cs.tut.fi/~foi/GCF-BM3D

[6] Deledalle, Charles-Alban, Joseph Salmon, and Arnak S. Dalalyan. "Image denoising with patch based PCA: local versus global." BMVC. Vol. 81. 2011.