Logarithm and Power Function To Modify an Image

Spring 2014
CSCI 567: Image Processing
with Applications

Processor: Dr. Nikolay M. Sirakov

Liliya Franke
Wuthiwat Ruangchai
Johnny Esquivel

May 1, 2014

TABLE OF CONTENTS

Abstract

The project focused on building the application that contains logarithm operation and power operation. Logarithm operation and power operation manifests the frequencies of the image by replacing original values of the image with the logarithm or power values of itself. The user's interface is created by using Java swing class. Our group wrote application on java language, because we want the application to be a multi-platform application. We extend function of the application that user can apply the operation on partial part of an image. User can get more information of the image because our application gives before and after applied operation values of the mean and standard deviation. Also user can keep track of how many times that operations were applied to the image and how long that took the application to finish all the operation processes.

Introduction

Nowadays, computers are developed rapidly. Some of applications that were written in the past are not able to run on the new platform anymore. One of the reasons for working on this project is to create an application that is able to run and perform effectively on the new platform. Our group decides to build our project by using Java language because this language is multiple platform language. Java language was build to satisfy object-oriented programming.

Object-oriented programming is a programming paradigm that uses abstraction to create models based on the real world. It uses several techniques from previously established paradigms, including modularity, polymorphism, and encapsulation. Today, many popular programming languages (such as Java, JavaScript, C#, C++, Python, PHP, Ruby and Objective-C) support object-oriented programming (OOP). Object-oriented programming may be seen as the design of software using a collection of cooperating objects, as opposed to a traditional view in which a program may be seen as a collection of functions, or simply as a list of instructions to the computer. In OOP, each object is capable of receiving messages, processing data, and sending messages to other objects. Each object can be viewed as an independent little machine with a distinct role or responsibility. Object-oriented programming is intended to promote greater flexibility and maintainability in programming, and is widely popular in large-scale software engineering. By virtue of its strong emphasis on modularity, object oriented code is intended to be simpler to develop and easier to understand later on, lending itself to more direct analysis, coding, and understanding of complex situations and procedures than less modular programming methods.[1,2,3]

We started by creating a java class. Using a class means to do programming in an OO method. The benefit to this is that couple of people could work on difference parts while others worked on difference part separately. We created a list of functions, which you can see in the document called “Class ImgBlock”. Another benefit to OO is that the functions and variables are encapsulated. What this means is that private variables in the class are not accessible by the program that is running outside of the class. This means that when somebody's programs the GUI it can’t mess up any of the actions going on in the class because it is private [1].

To begin the class we created a constructor. ImgBlock(JLabel jlabel1, JLabel jlabel2, JLabel jlabel3) A constructor is the 1st program that runs when you instantiate the class object. The constructor in the ImgBlock class takes the labels from the graphical user interface and keeps a reference to them inside the class. A label is the box where you put the images. The labels are jlabe1, which is the input image box, jlabel2 which is the transfer curve box and jlabel3 which is the output image box. When the constructor is run, the sizes of the image boxes are measured so that the input image and the output image can be resized to fit the boxes. Also the default function and exponent are set to the log function and at 2.0.

Human eyes can see only specific dynamic range of an image. Logarithm operation is one of the operations in our project that can manifest the frequencies of the image. The image contains components of all frequencies. When the frequency of the image gets higher, their magnitude also gets smaller. The low frequencies have more information of the image than the higher frequencies. The logarithm operation bring up the low frequencies that means it bring up more information of the image. [5] Another

operation in our project is power operation. With the image that is not gamma-encoded, it has too many bits or too much bandwidth to high that humans cannot tell the different or it has less bits or bandwidth of the shadow values than the required bits to keep the same visual quire. We apply the image with the power operation to fix these problems. [2]

Theoretical Background

Image enhancement is generally considered the processes of manipulating an image in the space domain (opposed to the frequency domain). These processes, when applied to an image, make the image better, which is why it is called enhancement. However, whether the image is actually improved is subjective [4]. For instance changing an image from color to grayscale may seen artistic to some people, but could instead seem old fashion to other people. In this work the image enhancement is to transform the gray level of every pixel in the image by using the logarithmic and power transforms. The logarithmic, or log, transform expands the dark values and compresses the light values. The power transform does the opposite, which is to compress the dark values and expand the values of the light pixels [5, 8].

A transform is analogous to a function, in that it converts an input image to an output image which that has different values for all of its pixels. The notation we use is T for the transform as shown below:

$$new\ values = T(\ old\ values\ ) \tag{1}$$

The values that we apply the transform to are the pixel values of the image. Computers are binary machines and therefore use base 2 numbers. Typically the pixel values are described using 8 bits or 1 byte of memory. We call this description grayscale. In grayscale the value can go from $2^0 = 1$ to $2^8 = 256$. However, since we start from 0, the grayscale values will go from 0 to 255. When working with images and applying these transforms we like to work with a float number; in particular we would like to work with values between 0.0 and 1.0. Normalization is the process of converting the grayscale values to the 0.0 to 1.0 range [7]. Once the values are normalized, applying the

log and power transforms is easily done by applying the log and power function to each pixel.

To better aid our understanding of the log and power transforms we will also need to understand the linear or identity transform. The identity function is of the form:

$$y = x \tag{2}$$

As seen above, equation 2 for the identity function simply returns the value that is given. If the identity function were used as a transform on an image, the output image would be exactly the same as the input image [3].

Log Transformations

Logarithm, with a base B, of a number X, returns the exponent that B was raised to, to return the value X [8].

$$\exp = \log_B (X) \tag{3}$$

The plot of the logarithm or log transform, as shown in *Figure **1***, can be used to shrink the light region and enlarge the dark region. Notice that the log function itself is negative

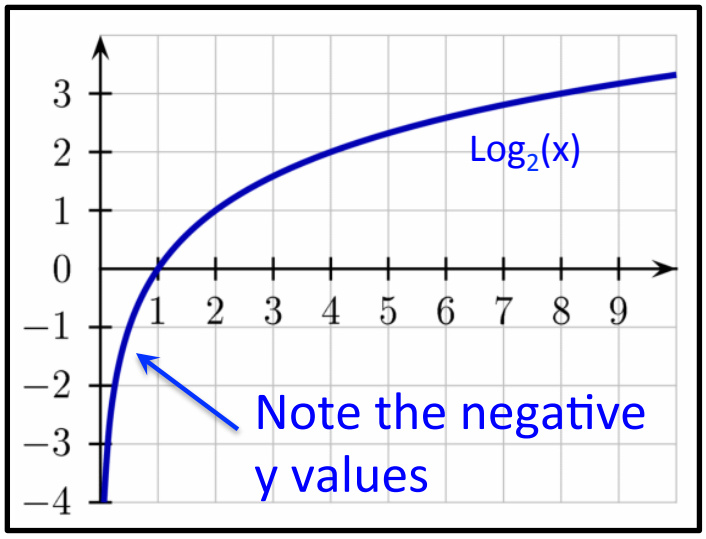

Figure 1. Logarithm Function with a Base = 2.0.

at x values less than 1.0. Therefore, we must add an offset of 1.0 to give us useful y values that are greater or equal to 0.

$$y = log(x + 1.0) \quad (4)$$

Figure ***2*** is an exaggerated illustration of how the log transform changes an image. In this example, suppose that we have a pixel who's original value is 0.5. The output of the linear or identity function return the value 0.5 as the output value. However for the 0.5 value input, the log function returns the value 0.8. Therefore this one pixel has become lighter. As can also be seen in this example, the values from 0.0 to 0.5 get stretched out to 0.0 to 0.8. Therefore we can say that the dark value of the image is

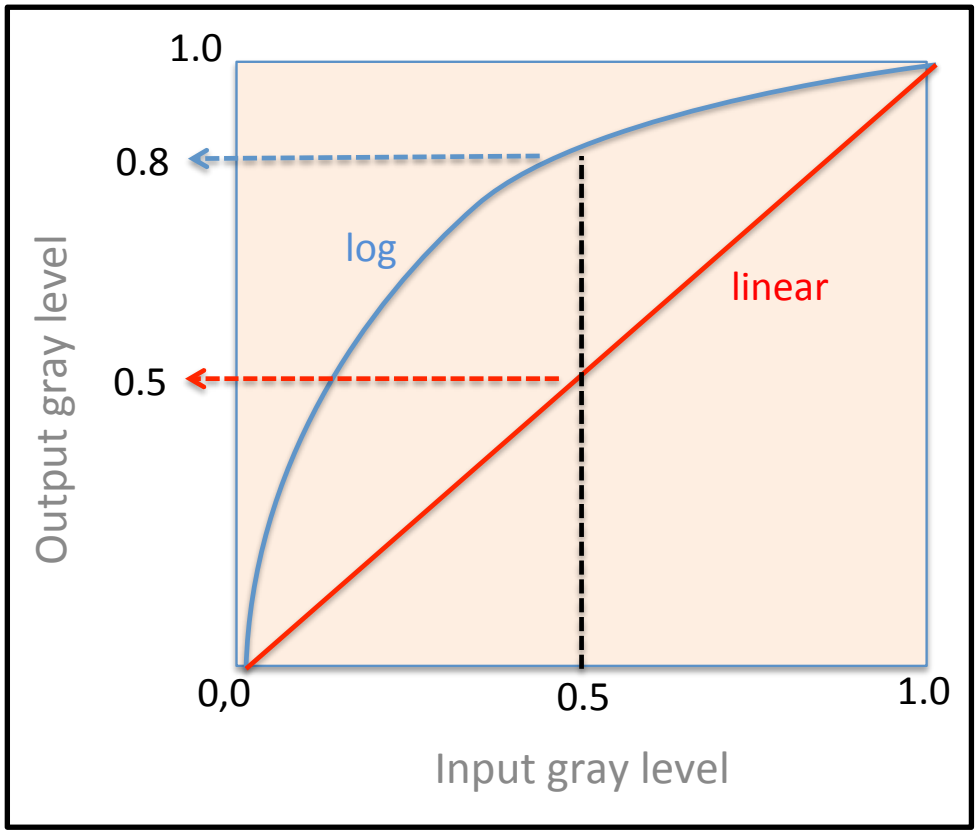

Figure 2. Exaggeration to Show How Log Changes An Image.

expanded. The input values from 0.5 to 1.0 are mapped to the output of 0.8 to 1.0. Therefore we can say that the light values of the image are compressed.

In reality the curve is not as drastic as shown in the example. *Figure **3*** shows examples of the log function with different bases greater than 1.0. In this figure you can

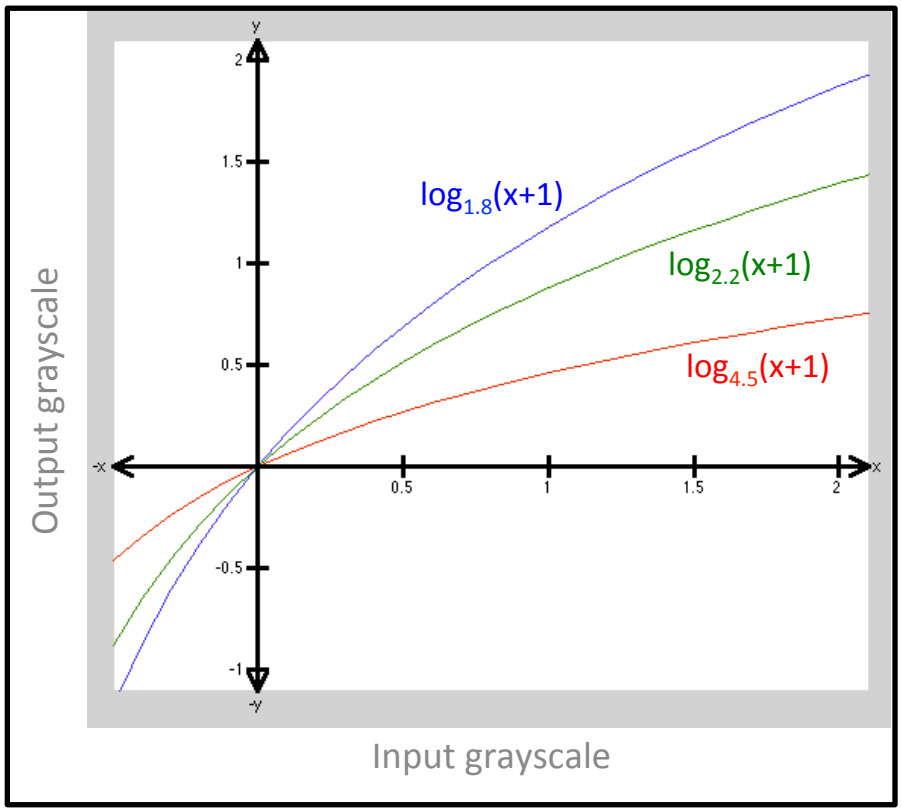

Figure 3. Example of Various Log Functions with Base > 1.0.

see that as the base of the log increases, the transform returns darker images. In fact, with a base of 4.5, the log function output never reaches 1.0. On the other hand, the log function with a base of 1.8 goes beyond 1.0. When this function is implemented in the code, a maximum function will need to be applied to prevent the output from going above 1.0 as shown in equation 5.

$$y = max(log(x + 1.0), 1.0) \quad (5)$$

Lastly, if the base becomes less than 1.0, then the results becomes inverted about the y-axis. Therefore, the base must be greater than 1.0 to prevent the inversion of the function which when inverted provides no practical purpose to our usage of the function.

Power Transformations

The power function raises a number X to an exponent value. The exponent value is the number of times the value X is multiplied by itself. The form of the power function is:

$$Y = X^{EXP} \tag{6}$$

The plot of the power function curve is shown in *Figure **4***, can be used to shrink

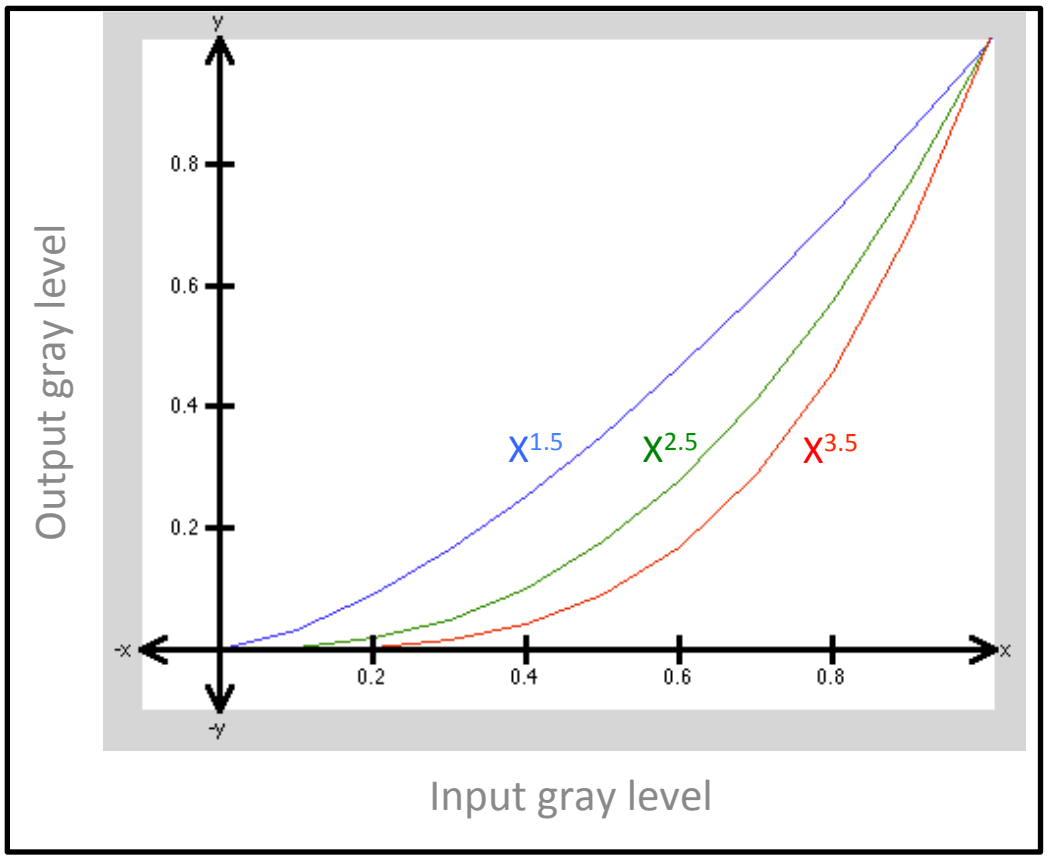

Figure 4. Power Function with various exponents.

the dark regions and enlarge the light region. *Figure **5*** is an exaggerated illustration of how the power transform changes an image. In this example, suppose that we have a pixel who's original value is 0.5. The output of the linear or identity function return the value 0.5 as the output value. However for the 0.5 value input, the power function returns the value 0.2. Therefore this one pixel has become darker. As can also be seen in this example, the values from 0.0 to 0.5 get shrunk out to 0.0 to 0.2. Therefore we can

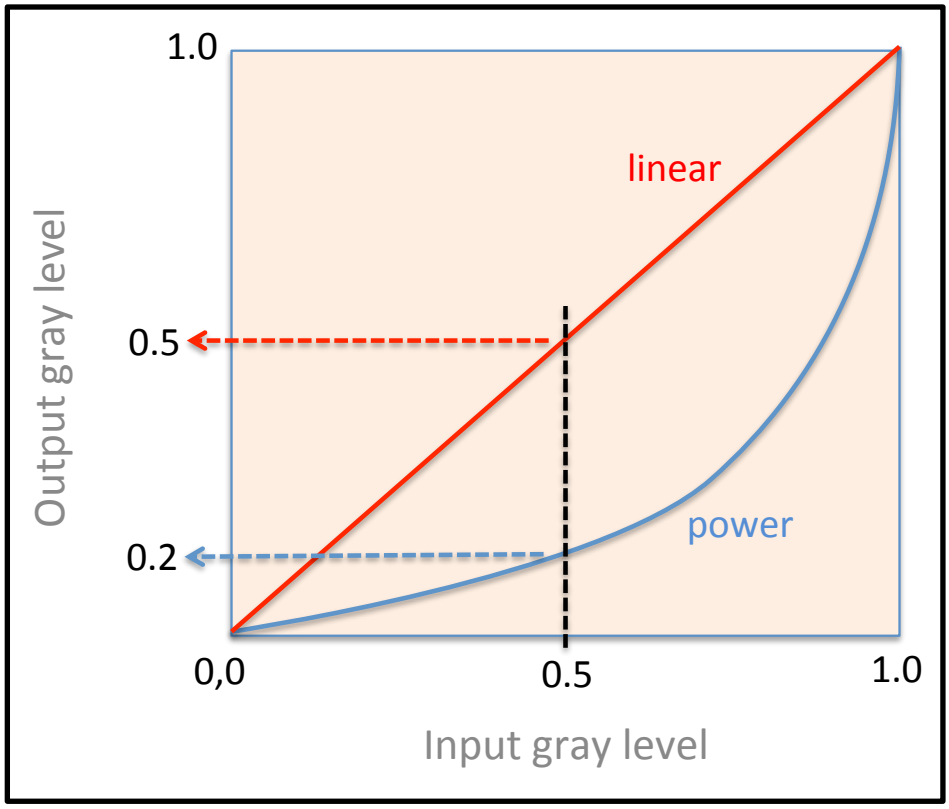

Figure 5. Exaggeration to Show How Log Changes An Image.

say that the dark value of the image is compressed. The input values from 0.5 to 1.0 are mapped to the output of 0.2 to 1.0. Therefore we can say that the light values of the image are expanded.

In reality the curve is not as drastic as shown in the example. *Figure **4*** shows examples of the power function with different exponents. In this figure you can see that as the exponent increases, the curve becomes deeper. This means that as the exponent increase, the dark regions become more compressed and the lighter regions expand more. On the other hand, if the exponent of the power function is reduced to 1.0, then the transfer function becomes the linear identity function. Unlike the log function, the power function does not need to offset the x-axis.

Algorithms and Procedures

Algorithm implementation in PALPIE were encapsulated in the ImgBlock class. As shown in *Figure **6***, the PALPIE is split into five main classes. The Main GUI class is the code for implementing the user interface and is what the user interacts with. Elements such as buttons and image panels are created in the Main GUI by using the Swing GUI widget class. The GUI calls the ImgBlock class to manipulate the image and get stats about the image. The ImgBlock class calls the ImageIO class when it wants to

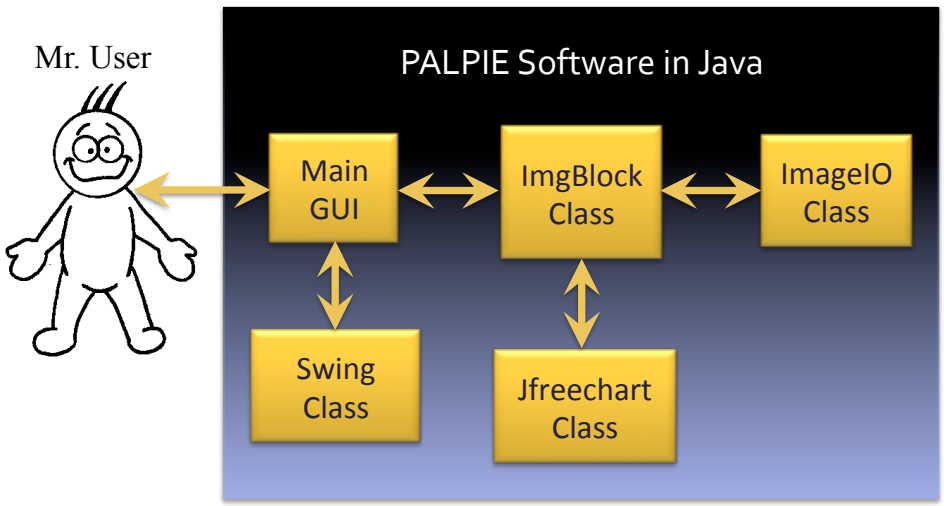

Figure 6. Topology of PALPIE Software.

open an image. It also calls the Jfreechart class when it builds an enhancement curve to send to the GUI.

The primary function of the ImgBlock class is to apply the logarithm and power transform to the image. The procedure to accomplish this is as follows:

1. Open the image requested by the user using ImageIO function.
2. Image values are separated to gray levels for each color.

blueGL = imgIntValue & 0x0000FF (7)

greenGL = (imgIntValue & 0x00FF00) >> 8 (8)

redGL = (imgIntValue & 0x0000FF) >> 16 (9)

3. Each color is normalized from grayscale to floating point number between 0.0 and 1.0.

$$normGL = \sum_{i=0}^{M}\sum_{j=0}^{N} oldGL_{ij} \times \frac{1.0}{255} \quad (10)$$

Where M in the number of pixels in the x-axis, N is the number of pixels in the y-axis, oldGL is the original input gray levels, and normGL is the output normalized gray levels.

4a. If the user selected the log transform [7]:

$$newGL = \sum_{i=0}^{M}\sum_{j=0}^{N} \frac{log_{10}(normGL_{ij})}{log_{10}\,(base)} \quad (11)$$

Where M in the number of pixels in the x-axis, N is the number of pixels in the y-

axis, normGL is the input normalized gray levels, newGL is the output gray levels, and base is the logarithmic base selected by the user.

4b. If the user selected the power transform [7]:

$$newGL = \sum_{i=0}^{M}\sum_{j=0}^{N} normGL_{ij}{}^{exp} \tag{12}$$

Where M in the number of pixels in the x-axis, N is the number of pixels in the y-axis, normGL is the input normalized gray levels, newGL is the output gray levels.

5. The values are then limited using the min and max function as follows:

$$\text{finalGL} = \max(\ 0.0,\ \min(\ 1.0,\ \text{newGL})\ ) \tag{13}$$

6. The finalGL converted from the normalized value to gray level (0 to 255).

$$imgGL = \sum_{i=0}^{M}\sum_{j=0}^{N} finalGL_{ij} \times \frac{255}{1.0} \tag{14}$$

Where M in the number of pixels in the x-axis, N is the number of pixels in y.

7. All color channels are combined into on integer value.

$$\text{outIntValue} = \text{blueGL} \ \&\ 0\text{xFF} \tag{15}$$

$$\text{outIntValue} = \text{outIntValue} + (\ (\text{greenGL} \ \&\ 0\text{xFF}) << 8\ ) \tag{16}$$

$$\text{outIntValue} = \text{outIntValue} + (\ (\text{redGL} \ \&\ 0\text{xFF}) << 16\ ) \tag{17}$$

8. The gray level value outIntValue is written as an image using the ImageIO class.

Step 1 in the procedure above uses the java library ImageIO to read an image. The ImageIO function returns an integer value from each pixel. The integer value is the combined red, green, and blue values. Each color is an 8-bit value. Therefore the next step (step 2) that is done is to separate the color channels (red, green, and blue) into separate arrays. If the image is grayscale (no color) then only one grayscale array is created. Also each integer value is converted to a double value from 0.0 to 1.0 in step 3. Next, in step 4, the power or log transform is applied to each pixel in the image. Note that there is no log function with different bases therefore we have to use two log functions as shown in step (a). The power and log functions that are used here are from the math library in java. Both the fastMath library from Apache commons and jBlast Matrix Math library were tried to speed up the execution, but they were not any faster. The transformed matrices have a min and max function applied to them to ensure that the results are between 0.0 and 1.0. Next the matrix is changed back to grayscale from the normalized value. If the image is a color image, the three grayscale values are combined into one. Lastly the grayscale matrix is changed back to an image using the ImageIO library.

Region Selection

The "Region selection function" applies the log or power transform to a limited section of the image. The region is given by the GUI, but the coordinates given by the GUI are in relation to the size of the GUI window. The method used to apply the log or power transform to just part of the image is as follows:

1. Receive the window coordinates from the GUI.

2. Convert the window coordinates into image coordinates. The algorithm to do this and the variables used in equations is shown below.

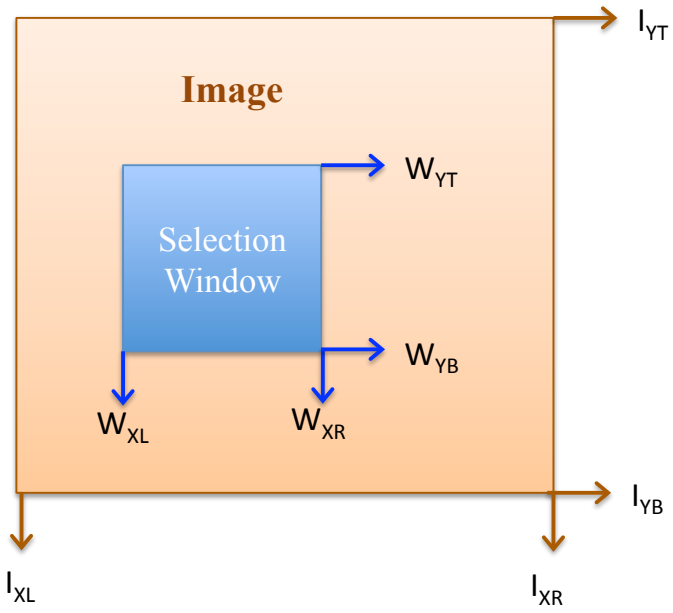

Figure 7. Converting window coordinates into image coordinates.

$$I_{XL} = W_{XL} \times (I_{XS}/W_{XS}) \quad (18)$$

$$I_{XR} = W_{XR} \times (I_{XS}/W_{XS}) \quad (19)$$

$$I_{YT} = W_{YT} \times (I_{YS}/W_{YS}) \quad (20)$$

$$I_{YB} = W_{YB} \times (I_{YS}/W_{YS}) \quad (21)$$

3. Use I_{XL} I_{XR} I_{YT} and I_{YB} as the range for the log or power functions. So for instance the power function algorithm would be:

$$newGL = \sum_{i=I_{XL}}^{I_{XR}} \sum_{j=I_{YB}}^{I_{YT}} oldGL_{ij}{}^{exp}$$

where oldGL is the old normalized gray-level from the pixel. (22)

For the log function

$$newGL = \sum_{I_{XL}}^{I_{XR}} \sum_{I_{YB}}^{I_{YT}} \frac{log_{10}\,(oldGL_{ij})}{log_{10}\,(base)}$$

where oldGL is the old normalized gray-level from the pixel. (23)

Image Mean

The mean of the image returns the average value for all the pixels in the image. The function averages the three color channels (green, blue, and red) together. If the image is grayscale then only the single channel is averages. The mean is applied to the floating point representation of the image therefore the mean will be between 0.0 and 1.0. The higher the mean value the lighter the image is.

The algorithm to calculate the mean is:

$$mean = \frac{1}{MN} \sum_{i=0}^{M} \sum_{j=0}^{N} normalized\_gl\,(i,j)$$

(24)

Where M in the number of pixels in the x-axis, N is the number of pixels in the y-axis, and normalized_gl is the normalized gray levels.

Again this is done for each color and for both the original image and the new image.

Image Standard Deviation

The standard deviation is measure on both the old and new images to let the user how much the contrast of the image has changed. The standard deviation function is applied to the image when it is in the floating point representation. Therefore the standard deviation will be a value between 0.0 and 1.0.

The algorithm to measure the standard deviation [6] is:

$$norm\_std = \sqrt{\frac{1}{MN}\sum_{i=0}^{M}\sum_{j=0}^{N}(mean - \ normalized\_gl(i,j))^2} \tag{25}$$

Where M in the number of pixels in the x-axis, N is the number of pixels in the y-axis, oldGL is the original input gray levels, and normGL is the output normalized gray levels.

Again this is done for each color and for the old and new images.

Number of Operations

The last part of the algorithms is the "Number of Operations" function. This function calculates how many the operations were done during the transformation of the last image. Since we don't know how many operation the log function or how many operations the power function use, we simple count how many times the power or log function is called. This works out to basically be equal to the number of pixels in the image which is M x N. Since the number of operations can be pretty large, the function return the value in a long variable type.

Technology and System Requirements

The system requirements to run this program are for any computer system that can install and run the latest Java virtual machine. Generally this includes any Microsoft Windows, Apple OSX, Linux, or Solaris system, however, the Windows operating system must be Windows XP or newer. Of course older computer will take longer to complete the functions than a modern computer system. Designing the GUI to use Java does mean that the user must install Java on the system, unless it is installed by default like on Apple OSX. The tradeoff is that the software developed for Java is very portable. A mouse is also required to click on the buttons and bring the GUI into focus.

The software was designed using the Java JDK 8.0 This JDK has the latest security updates to Java. To design the GUI, the Java Swing library was employed. Java swing is an older GUI widget toolkit. Swing is not thread-safe, which means that there could be memory inconsistences from interference from multiple threads, due to the fact that when it was designed, most computers did not have enough resources to run multiple threads. However we used Netbeans to design the GUI which implements the Event Dispatch Thread to avoid this problem.

Java JFreeChart library was used to create the enhancement curve plot. The library can create any charts and output the chart as an image. Although create a plot using this library proved to be more complicated than expected as the library was build to only create plots in a separate window. Considerable time was invested in manipulating the software to create a chart inside of our GUI.

User Guide

1. The picture what we are going to use should be in .jpg or .bmp format.
2. Open project application.
3. Click “OPEN” button.
4. Browse the address and choose where the image is located in .jpg format.
5. Click “OPEN”.
6. Select the function type. The function type: “Power Function” or “Log Function”. The default function is log function.
7. Adjust “Base of the Function” with “Up” and “Down” buttons. The default values of the function is 2.0.
8. User can select the area of the image that they want to apply the function by click at “Input Image”. Create a rectangular shadow. The function will be applied only in the area of the rectangular shadow.
9. If user wants to cancel all the results of the function, click on the “Undo” button.
10. Saving Image: User shall click on “Save Output File”. Select the address and the name of the output that the user wants to save and click “Open” button.
11. Output:
12. (a) Enhancement Curve: Showing the curve of the function that apply to the image.
13. (b) Output Image: The image after applied function.

Experiments

Applying Logarithm Function to Color Image with Region Selection

The first experiment was using PALPIE to apply the logarithm function to a color image. The original color image, as shown in *Figure **8***, was a Jpeg, size 768 x 1024. The PALPIE software was used to apply a log with a base of 4.0 to the image to a small section of the image by using the region selection function. As is shown in the image after the function is applied, the area becomes lighter.

Figure 8. Color image before and after logarithm with base of 4.0

Applying Power Function with Exponent of 3.2 using Region Selection

This experiment was using PALPIE to apply the power function to a color image. The original color image was a Jpeg, size 702 x 343. The function that was applied was the power function with an exponent of 3.2. The region selection function was also used as seen in *Figure **9***, by a gray rectangle on the input side. On the output side on the right,

the power function is applied to the box making that region darker. The mean changed slightly from 0.15 to 0.14, but the standard deviation did not change. The number of operation to complete this transform was 722,358 and took 190ms. These statistics represent the pixels in the region selection box.

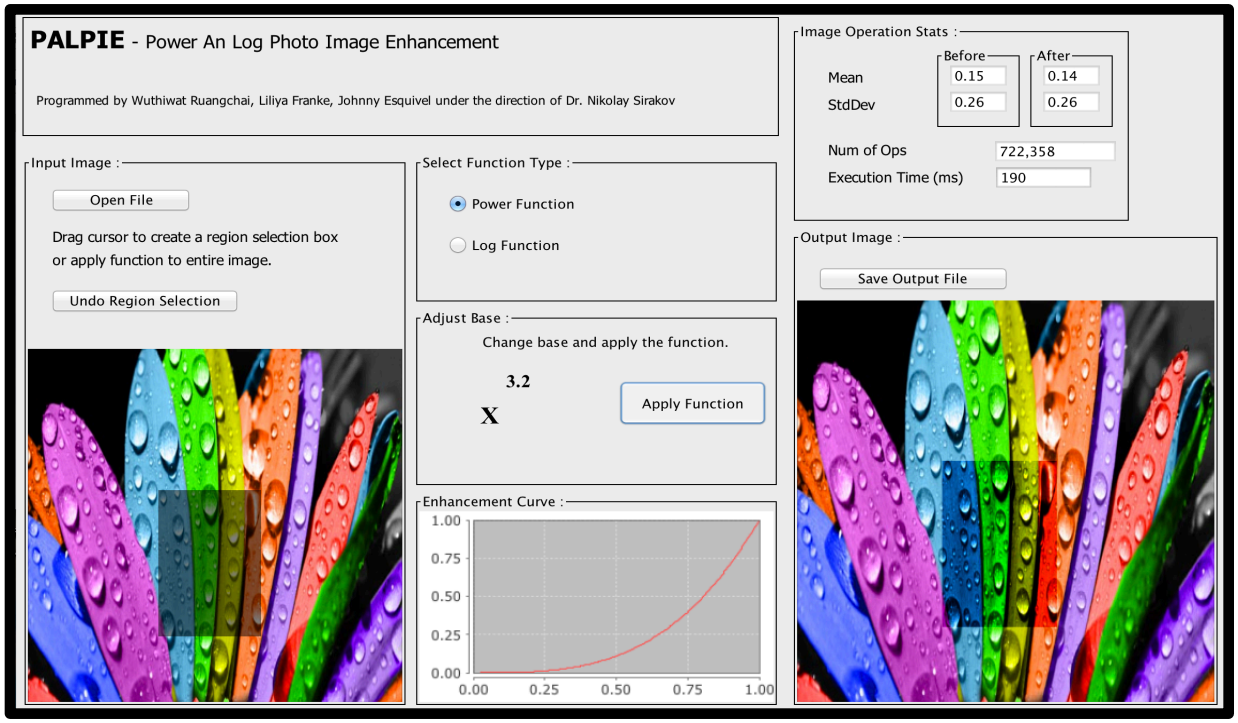

Figure 9. Color image before and after region selection applying power to the 3.2.

Applying Logarithm Function with Base of 1.6 to a Grayscale Image.

This experiment was using PALPIE to apply the logarithm function to a grayscale image. The original grayscale image was a Jpeg, size 512 x 512, as shown in *Figure **10***. The function that was applied was the logarithm function with a base of 1.6, as shown in *Figure **11***. The mean changed slightly from 0.21 to 0.30, meaning the image got lighter. The standard deviation changed from 0.23 to 0.36 meaning that the contrast increased. The number of operation to complete this transform was 786,432 and took 223ms.

Figure 10. Color image before selection applying power to the 3.2.

Figure 11. Grayscale image after power function with exponent of 1.6

Reveal Hidden Feature using Power Function with Exponent of 0.2

This experiment demonstrated how PALPIE could be used to reveal hidden features using the power function. The original color image was a Jpeg, size 620 x 509, as shown in *Figure **12***. The function that was applied was the power function with an exponent of 0.2, as shown in *Figure **13***. The mean changed slightly from 0.03 to 0.17 meaning that the image got lighter. The standard deviation when from 0.07 to 0.24 meaning that there became more contrast in the image. The number of operation to complete this transform was 946,740 and took 287ms.

Figure 12. Color image before and after region selection applying power to the 3.2.

Figure 13. Grayscale image before and after power function with exponent of 1.6

Conclusion

In this project, the user interface was created by using Java Swing Class. We successfully created the user interface. We then passed the value of the function based. We then pass the region of the image that the user wants to apply to the function to. Finally we save the image after the function has been applied. Java makes it easy to import the image for the project. The standard functions in Java can then be used to apply the logarithm and power function to each pixel. With Java we could use the mathematical formulas on graphics and pictures.

From our experiments we can see that the program was written successfully and is fully functional. The user GUI is working as we can see when we open any of the pictures. Experiments show that we can change color from light to dark and back again. If the picture too light than we can use the power function to make the picture darker. If the picture is too dark we use the logarithm function to add lightning to the pictures. During our experiments we could see that when we used the original dark picture and applied the logarithm function to lighten the picture we got the desired result. The result was we could recognize an animal emerge from the dark picture. The program that we created is easy to use and user friendly.

References

[1] *Object-oriented Programming*. Retrieved Apr 18, 2014 from Wikipedia website, http://en.wikipedia.org/wiki/Object-...ed_programming

[2] *Image Processing*. Retrieved Apr 20, 2014 from Wolfram Alpha website, https://reference.wolfram.com/mathematica/tutorial/ImageProcessing.html

[3] Casado, C. O., & Popova, A., (2010*). Image Contrast Enhancement Methods*. Published online via Academia-e. Retrieved Apr 20, 2014 from Academica-e website, http://academica-e.unavarra.es/bitstream/handle/2454/2368/577327.pdf?sequence=1

[4] Gonzalez C. R., Woods E. R., (2000) *Digital Image Processing*, 2nd Ed., 2000. Prentice Hall, New Jersey.

[5] Jain, A., (1989) Fundamentals of Digital Processing, Princeton Hall 1989, New Jersey

[6] Image Processing: Brightness, Contrast, Gamma, and Exponential/Logarithic Settings in ProAnalyst (2010). Retrieved Apr 21, 2014 from XciteX Inc. Website, http://www.xcitex.com/Resource%20Center/ProAnalyst/Application%20Notes/App%20Note%20151%20-%20Image%20Processing%20Brightness,%20Contrast,%20Gamma%20and%20Exponential.pdf

[7] Kang, S. K., Min, J.H., Paik, J.K., (2001). *Segmentation based spatially adaptive motion blur removal and its application to surveillance systems*. Proc. Int. Conf. Image Process. , vol. 1, pp. 245–248. 2001.

[8] Sirakov, N., (2014) CSCI 567: Image Processing with Applications Lecture Notes. Annotated during the spring 2014 semester.