Adapting Complex Image Processing Algorithms to Mobile Devices

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

By

Justin Stein

2011

© 2011

Justin Stein

ALL RIGHTS RESERVED

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

12/15/11

Advisor: Dr Andrzej Bieszczad | Date

12/15/11

Dr Richard A Wasniowski | Date

12/15/11

Dr Peter Smith | Date

APPROVED FOR THE UNIVERSITY

12-16-11

Dr. Gary A. Berg | Date

Proving It Can Be Done:

From Desktop Application to Mobile Application with CrazZzy Filterz

By

Justin Stein

Computer Science Program

California State University Channel Islands

Abstract

Mobile devices have come a long way since the earlier times. Cell phones were once only used for making and receiving telephone calls but now are transforming into small personal computers capable of running standalone programs or applications. Consumers are now regarding their mobile device as a portable personal computers. According to the US Smartphone Statistics of Quarter 1 of 2011, Smartphones made up 54% of all mobile phone sales in the US. As of March 2011, the majority of the platforms in which Smartphones run on consist of the Android OS, the Apple iOS, and the RIM BlackBerry OS. The Android OS holds about a 37% share, Apple iOS has about a 27% share and the RIM BlackBerry OS has a 22% share of the market. For this thesis I'm going to demonstrate the evolution of software from a very powerful hardware system such as a desktop computer to be used on this new evolving market of less powerful hardware found on mobile devices through an application I developed for the Android OS called CrazZzy FilterZ.

Even with the advancements of hardware there is still a gap between mobile devices and home computers. Programs that require high computing power to run will not run on mobile devices if it cannot provide the necessary power to do so. Due to this, programs are modified to work with fewer resources that are available to them. These programs end up trading some of their essence, anything from functionality to appearance. The developer has to balance these tradeoffs to a point to keep the end user happy and satisfied.

CrazZzy FilterZ will show some of the different methods to achieve a mobile application version of a desktop program that uses an enormous amount of computing power and is unsuited for use by mobile devices. CrazZzy FilterZ is a program that deals with the manipulating of the bitmap data of images and sharing between members of a community of developers of these image filters.

Acknowledgements

I would like to thank all those who have made this possible for me to achieve my goal. I would also like to give a very special thanks to all of my loved ones who supported me throughout my journey up to now. This is just the end of one chapter with many more to come in this book titled "My Life".

Table of Contents

CHAPTER 1: INTRODUCTION **9**
1.1 INTRODUCTION TO "TRICKS" 9
1.2 INTRODUCTION INTO DIGITAL IMAGES 10
1.3 INTRODUCTION AND EXPECTATIONS OF A MOBILE APPLICATION 10
1.4 INTRODUCTION TO ADOBE® AIR® AND ADOBE® PIXEL BENDER® 10
1.5 INTRODUCTION TO CRAZZZY FILTERZ 11

CHAPTER 2: FIELD OVERVIEW **12**
2.1 IMAGE PROCESSING 12
2.2 IMAGE RESIZING 15
2.3 LOOK-UP TABLES 19

CHAPTER 3: TECHNICAL DETAILS OF THE WORK **22**
3.1 CRAZZZY FILTERZ 22
3.2 LOOK-UP TABLE FILTER 33
3.3 RESIZING IMAGES USING BILINEAR SCALING 37
3.4 CONTROLLING THE ORIENTATION 39
3.5 INTEGRATION 41

CHAPTER 4: EXPERIMENTS **42**
4.1 TESTING ENVIRONMENT 42
4.2 TESTING THE DIFFERENT LANGUAGES 42
4.3 TESTING SPEED AND ACCURACY 46
4.4 USING THE LUT 49

CHAPTER 5: ANALYSIS OF RESULTS **52**
5.1 NATIVE ANDROID IS FASTER THAN ACTIONSCRIPT 3.0 52
5.2 SMALLER IS BETTER 52
5.3 USING THE LOOK-UP TABLE 53

CHAPTER 6: CONCLUSIONS **54**

CHAPTER 7: FUTURE WORK **55**

TABLE OF FIGURES

Figure 1. Digital image's matrix of numbers that represents the value of each pixel. 13

Figure 2. Image filtering Process – Example: Invert Filter 14

Figure 3. Linear Interpolation 15

Figure 4. Bilinear Interpolation for pixels between rows: 20, 21 and column: 15, 16 17

Figure 5. Examples: Resizing by Nearest-Neighbor, Bilinear, and Bicubic Interpolation 18

Figure 6. Graph: Look-up Tables - Normal, Invert, and Brightness 20

Figure 7. Graph: Look-up Tables: Contrast, Curves, and Threshold 21

Figure 8. CrazZzy Filterz, when it is first launched 22

Figure 9. Main Display Screen 23

Figure 10. Opened main menu 24

Figure 11. Selection of an image 25

Figure 12. All Filter Screen 26

Figure 13. Application displaying the Filters Window Menu Options. 27

Figure 14. Loading a new filter 28

Figure 15. Deleting a filter 29

Figure 16. Main displayed image with a filter applied to it 30

Figure 17. Parameter Panel showing the description along with the parameters 31

Figure 18. Pop up window used to modify a parameter of a filter. 32

Figure 19. Change in value is reflected in a small pop-up above the slider 33

Figure 20. The LUT Image 34

Figure 21. The Look-up Table Walkthrough 36

Figure 22. Data Chart: Lapse Times of Invert Filter – Android vs ActionScript 3 44

Figure 23. Graph: Lapsed Times of Invert Filter- Android vs. ActionScript 45

Figure 24. Data Chart: Lapse Times of Invert Filter – Original Sized vs. Resized 47

Figure 25. Graph: Lapse Times of Invert Filter – Original Sized vs. Resized 47

Figure 26. Picture Quality Test: Original Size vs. resized 48

Figure 27. Graph: Histogram Comparsion – Original Size vs. Resized 49

Figure 28. Data Chart: Invert Filter – Bitmap vs. LUT 50

Figure 29. Data Chart: Color Curves Filter – Bitmap vs. LUT 51

Figure 30. Data Graph: Invert and Color Curves Filters – Bitmap vs. LUT 51

Chapter 1: Introduction

1.1 Introduction to “Tricks”

This thesis was about the miniaturizing of hardware and the tuning of the software that must follow. In this age of technology, hardware is becoming smaller, faster, and cheaper for the consumer. The wave of breakthroughs in hardware have allowed for computers of the size of desktops to be the size of a cell phone with the same amount of processing power. The trend today is to compact hardware into much smaller sizes and to continue to improve on its performance. The general rule of thumb is still that computers will have far better performance than cell phones, but now the gap between the two seems to be much smaller.

Developers throughout history have developed “tricks”, to achieve the most out of their hardware, seemingly producing far better performance of an application than it can really manage. An example of this is in a three dimensional game world, where developers will trick the viewer into seeing more details on 3D models than is possible due to the sheer volume of calculations that would be needed to move each and every vertex that these models would contain if every depth on their surfaces was created. To avoid this large amount of calculations which in turn means slowing of the game play of any game using this 3D world, the models are designed with the least possible amount of vertices to define its shape and then are wrapped in a texture or bitmap that contains the required amount of detail instead. This technique gives these models great amount of detail when seen from a far but when viewed from a close the models reveals that their faces have no actual depth to them.

While creating an application, the developer must think of the end users and their experience while using it. While developing a new application, there will be times when the developer must consider what limitations their application has or will have and must weigh the pros and cons of alternative ways that they may want to take to achieve the same result. If we take a look back at the previous example of 3D models wrapped with a texture, the developer must first weigh the pros and cons of the different ways that are available to him to achieve the same result. The pros for adding a texture and limiting the amount of vertices contained in the model will increase the frame rates but the model’s overall detail up-close is dramatically decreased. The decision to go a certain way is then based upon what exactly the developer is trying to accomplish in their application. For a First Person Shooter video game, the end user requires the application to be responsive and most times they are not too concerned with how detailed things looks and don’t have much time to sight-see versus an adventure game in which an end user may want to check out their surroundings more thoroughly. In either case, there is the tradeoff and to create a good application the developer needs to conclude what is acceptable.

1.2 Introduction into Digital Images

Images consist of pixels arranged into columns and rows of a grid. Each pixel is a color and a computer represents each pixel by a number that is equal to that color. A pixel consists of three colors; Red Green and Blue, along with a value for intensity. The values for each color and intensity are represented by that number. When a program does any image processing, it works with that number of each pixel. When manipulating an image, an algorithm is performed on each one of the pixels, for example a 3.2 megabyte image has 1600x2000 pixels that are needed to be computed for any filter to run on to the entire image.

1.3 Introduction and Expectations of a Mobile Application

A mobile device is a device that is portable and independent. It can be use without the limitation of a static power source or structure where it would prevent mobility. A mobile application is any program that can be used on that device.

There are some expectations that users have formed from using mobile applications. The most common expectation is that a mobile application will have are two different viewing layouts. Devices are not normally square but instead are rectangular so, at any given time the display screen of the device will have different sizes for either its height or width. Users expect an application to display a different layout according to the device's orientation.

Another expectation is to have commonalities between mobile applications. When users use a new mobile application they do not want to have to re-learn how a mobile application should react when they perform a certain action. An example of this is when the back button is pressed and users are excepting the application to take a step back to the previous screen or state in which that application was in. Users also expect a menu to appear when the menu button is pressed.

1.4 Introduction to Adobe® AIR® and Adobe® Pixel Bender®

Adobe AIR is a product from Adobe. It is a runtime that enables developers to use HTML, JavaScript, Adobe Flash® Professional software, and ActionScript® to build web applications that run as standalone client applications without the constraints of a browser. (www.adobe.com/products/air/)

There are an abundance of Operating Systems that will run Adobe AIR, one of them is the Google's® Android®. This is only possible in 2.2 or later versions of Android.

Adobe Pixel Bender technology delivers a common image and video processing infrastructure which provides automatic runtime optimization on heterogeneous hardware. (http://www.adobe.com/devnet/pixelbender.html)

Users of Pixel Bender can develop image processing algorithms for use in filters or effects. These filters can be intergraded into other Adobe product including Adobe AIR. These users post their created filters to the Pixel Bender Exchange, http://www.adobe.com/cfusion/exchange/index.cfm?event=productHome&exc=26&loc=en_us, to be shared with any other user in the community.

CrazZzy Filterz uses these two technologies to create an application that is dynamic and seemingly powerful on a mobile device where it was only thought of being available on a desktop computer. CrazZzy Filterz is built on Adobe Air because there is an ActionScript 3.0 SDK which provided the ability to load, use and gain access to the parameters of the Pixel Bender developed filters.

1.5 Introduction to CrazZzy Filterz

CrazZzy Filterz allowed for Adobe Pixel Bender's Filters to manipulate images on a mobile device. With this application the end user was able to load an image, save the image, add new available filters, delete already loaded filters, and change the values of all the parameters of any of the loaded filters.

The idea behind CrazZzy Filterz was to bring this already formed community of Pixel Bender developers to the mobile device community of users. With CrazZzy Filterz the user can load Pixel Bender filters that have already been created by the community or even some created by the user, in turn utilizing the already vast amount of created filters of the Pixel Bender Community.

CrazZzy Filterz has proven to be efficient and feasible as an AIR based application on a mobile device. CrazZzy Filterz has techniques in place to achieve a successful mobile application that can perform the necessary heavy computing required by an image filtering application. It also presented all the expectations that a mobile device user may have formed from previous interactions with other mobile applications.

CrazZzy Filterz reduced the amount of pixels in an image without lowering the image quality and decreased the potential lapse time needed for the user to apply a new filter to the image. CrazZzy Filterz also created and used a Look-up Table when it was displaying an image. It used the Look-up Table to decrease the lapse time that a filter needs to be applied to the image.

CrazZzy Filterz interacted with the mobile device like a native application would. It responded when both the mobile device's hard menu and back buttons were pressed. CrazZzy Filterz also changed its layout according to the orientation of the mobile device.

Chapter 2: Field Overview

There are a lot of different techniques that could have been used to achieve the goal of creating a high resource hog desktop application's counterpart that ran on a mobile device. First is the need to speed up the time that a filter would have to take to run on an image and the best and most effective way to achieve that is to reduce the size of the image. There are a few ways to go about resizing an image. Some of the most used methods are the Nearest-neighbor Interpolation, the Linear or Bilinear Interpolation, and the Bicubic Interpolation.

Another method that is used to speed up the time that a filter is required to run completely upon an image is to use a Look-up Table. There are other perks that are available to a developer when they run a filter upon the Look-up Table of an image such as never losing any of the original image pixel data of the Digital Image.

The last bit of methods are used to create the features of a mobile application that users have come to expect such as the need for the application to have two different layouts, one used when the device is in Landscape mode and one when the device is in Portrait. A few other features that are required by an end user would be the integration of the application with the mobile device and commonality of the application with all the other native application on the mobile device.

CrazZzy Filterz combines all of these methods to achieve an accepted performance level along with the expectations of a mobile device application compared to its desktop counterpart.

2.1 Image Processing

Images consist of pixels arranged into columns and rows of a grid. Each pixel has a color. Digital images have an internal representation of the value of each pixel as a simple matrix of numbers (Figure 1). This matrix of values is the data that is converted in to a visible image on a screen. Any altering of the data in this matrix would directly result in a change in visible appearance of an image. Typically, the matrix is the same size of the amount of individual pixels in the image but through the compression of saving a digital image using different techniques they will vary.

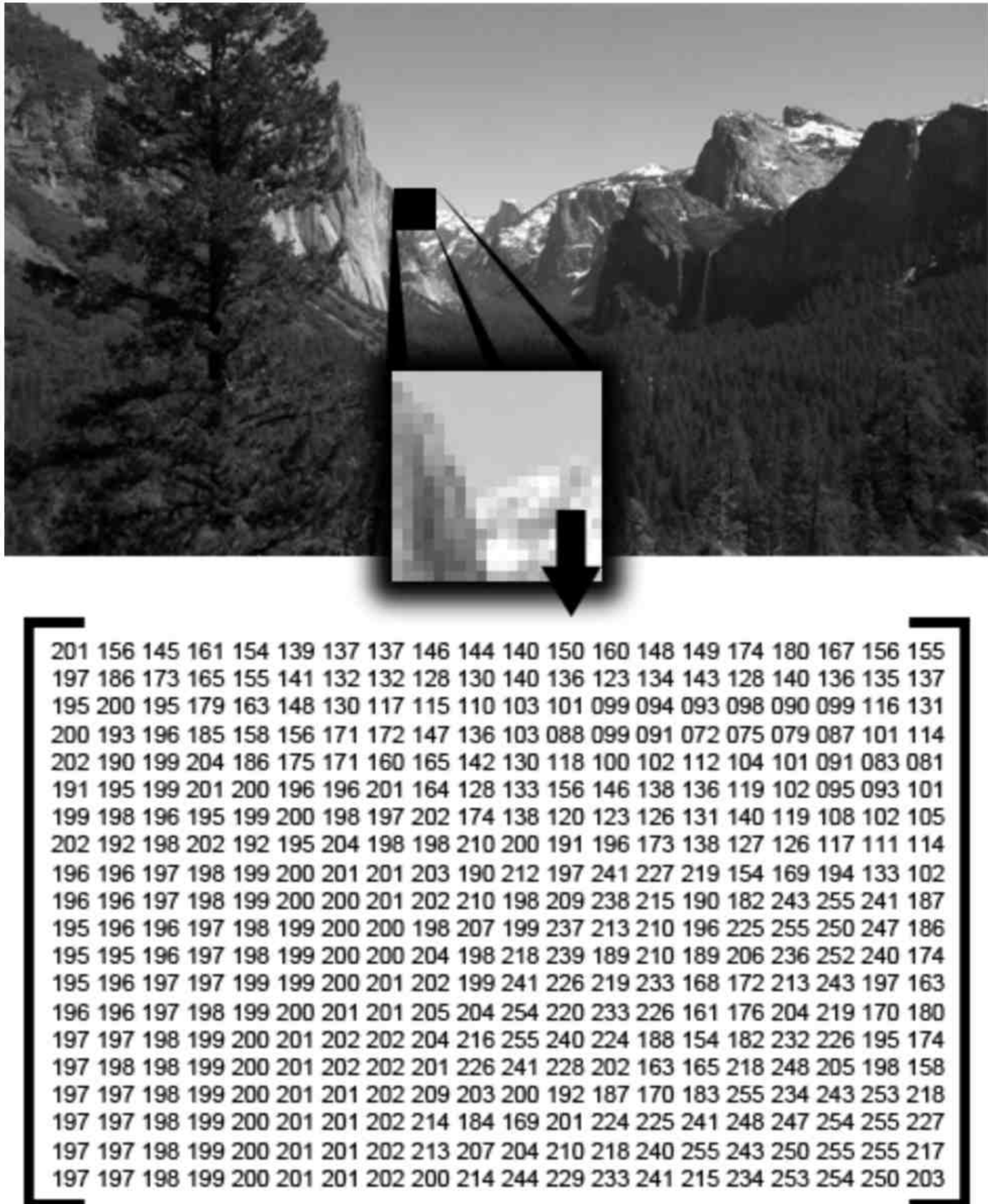

Figure 1. Digital image's matrix of numbers that represents the value of each pixel.

Image Processing refers to the analysis and manipulation of a digital image. This process usually intends for an algorithm to be run on the digital image's data. When an algorithm is run though all the pixels of an image, the algorithm is continually repeated for all the values in the image's data matrix. This algorithm can be used to return a new value in place of the current pixel's value, creating a new image from the data of the old one. This type of manipulation is called Image Filtering and a filter is just an algorithm that is run upon the pixels (Figure 2).

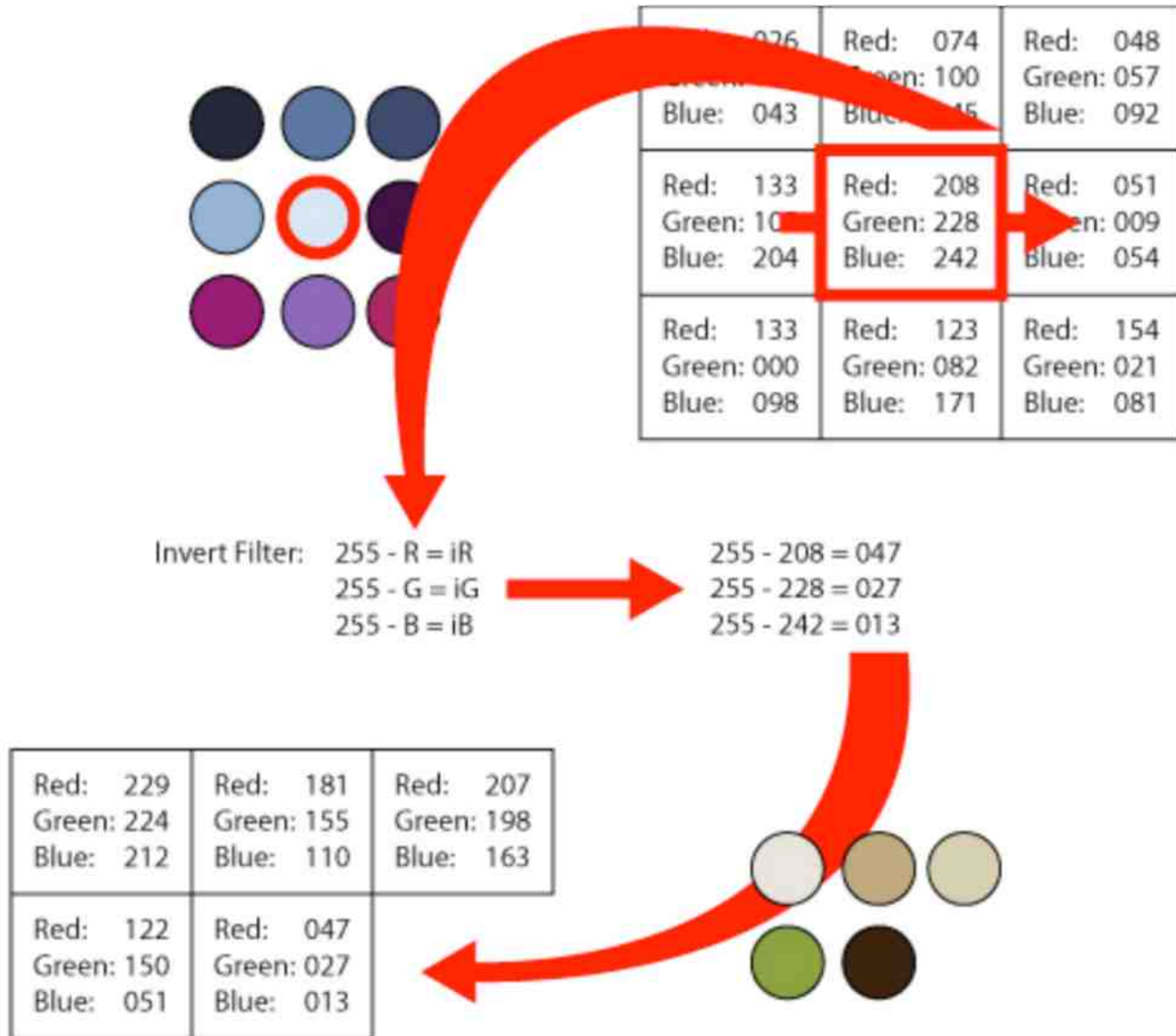

Figure 2. Image filtering Process – Example: Invert Filter

CrazZzy Filterz reduces the amount of values in an image's data matrix and in some cases utilizes the Look-up Table of digital images to lower the total amount of times that an image processing algorithm needs to be run to lower the overall amount of computing time that a filter may require to finish. This helped tremendously with the overall user satisfaction of the mobile application.

2.2 Image resizing

As mentioned before the most effective way to speed up the computing time of a filter is to lower the amount of times that the algorithm must run. To have the filter run on an image, the filter's algorithm must run on each pixel's data of the image hence by lowering the amount of pixels, the developer lowers the amount of times that the algorithm must run and that lowers the lapse time of applying the filter to an image.

When scaling a Digital Image, a new Digital Image is created at a different size. When creating the values of each of the pixels of the new scaled image, developers use Interpolation. Interpolation is a method to approximate new data points within the range of a set of known data points. Interpolation is used to find the value of some non-given point in some space when given the values of the other points around that space. In the case of resizing images the known data points are the pixel's values and their locations according to the original image.

The Nearest-neighbor Interpolation is the easiest way of scaling an image. The Nearest-neighbor Algorithm is simplest and fastest compared to the other algorithms. The algorithm selects the value of the nearest point and does not consider any of the other neighboring points. Due to this, the resulting image becomes pixilated and distinguishable from the original image.

Linear and Bilinear Interpolation are far better at preserving the smoothness of the edges of an image. Linear Interpolation is a form of curve fitting, where the value of an unknown point is calculated by the values of the two nearest known points. The Linear Interpolation Algorithm reveals a straight line between two point's values. The equation to find an unknown point's value between two already known point's values is:

$y = y_0 + \frac{(x - x_0)y_1 - (x - x_0)y_0}{x_1 - x_0}$, where the coordinates of the known points are (x_0, y_0) and (x_1, y_1) (Figure 3).

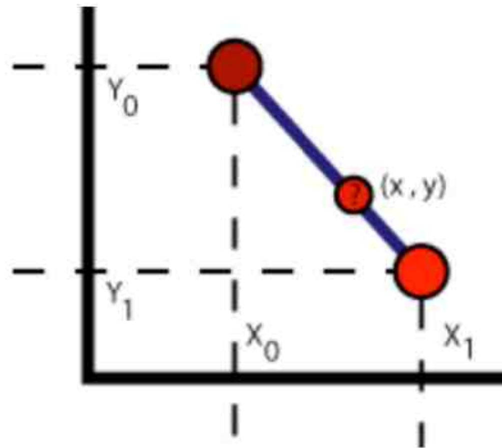

Figure 3. Linear Interpolation

The Bilinear Interpolation Algorithm is basically an extension of the Linear Interpolation Algorithm and is used to find a value of a unknown point upon a 2D plane. Bilinear Interpolation uses the values of the 2x2 neighborhood of known pixel's values surrounding the unknown pixel's location to calculate a weighed value for the unknown pixel's value (Figure 4). The Algorithm runs a Linear Interpolation Algorithm to calculate the values of the points of the two rows above and below of the unknown pixel's position and a last time to calculate the unknown pixel's value by Linear Interpolation between the two previous found values. This Interpolation is used by CrazZzy Filterz because of the quality of the resulting image and the speed of the algorithm.

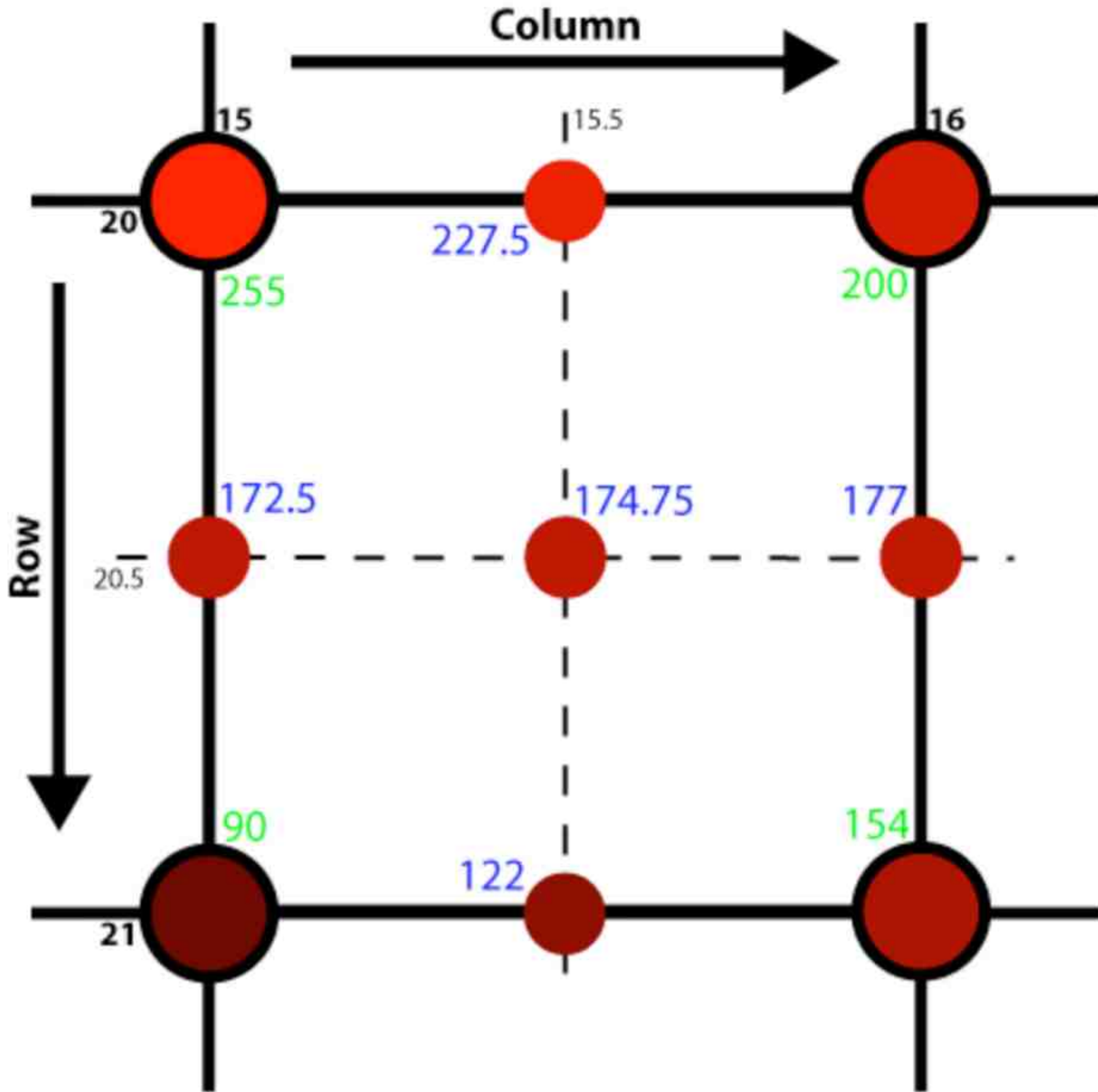

Figure 4. Bilinear Interpolation for pixels between rows: 20, 21 and column: 15, 16

Bicubic Interpolation is the slowest Interpolation Algorithm compared to the other ones that I mentioned but it produces a smoother resulting image with less interpolation artifacts or errors. Bicubic Interpolation is used in resizing an image when speed is not an issue.

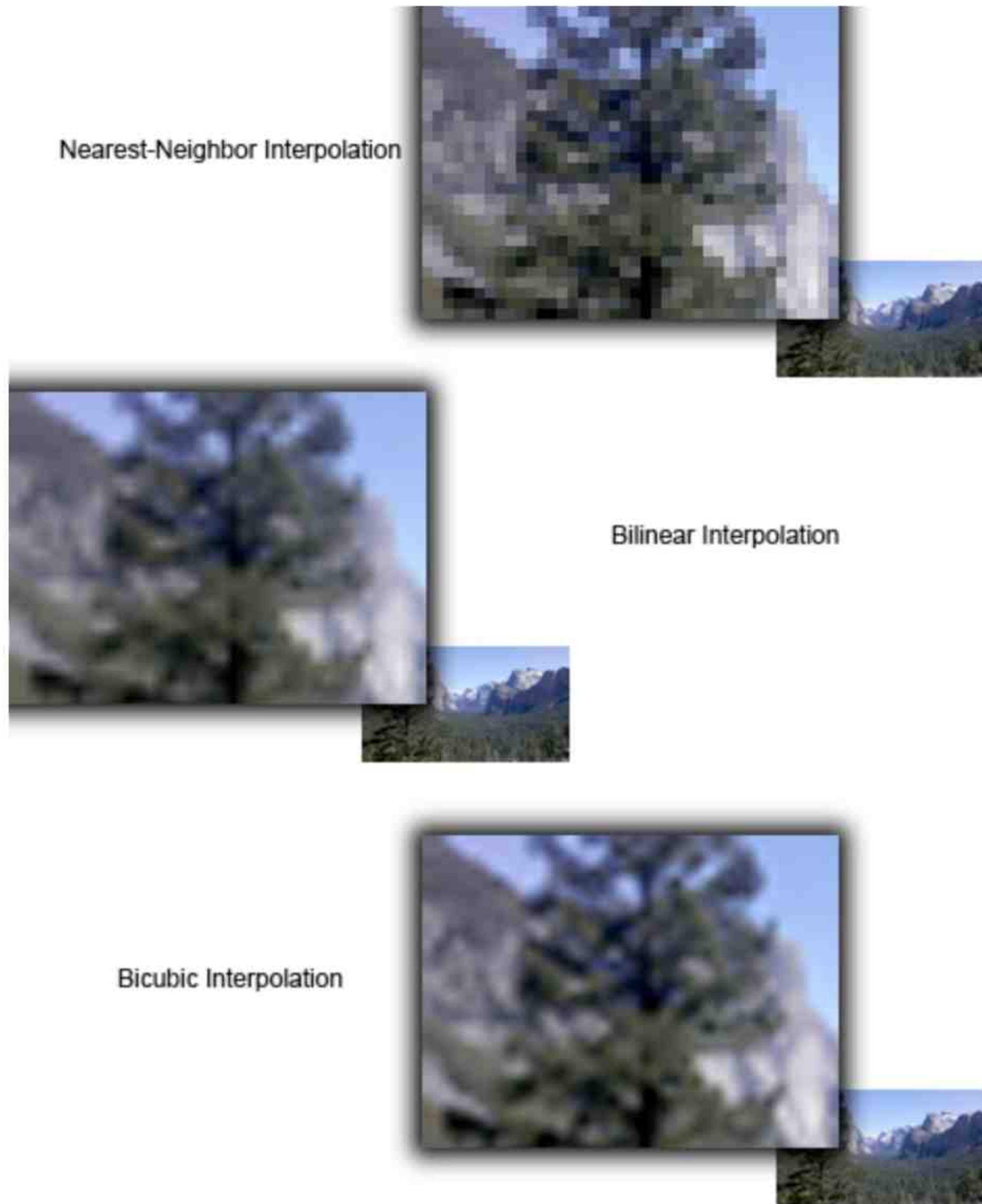

Figure 5. Examples: Resizing by Nearest-Neighbor, Bilinear, and Bicubic Interpolation

2.3 Look-up Tables

A Look-up Table (LUT) is imposed on the data of an image before the image is displayed on to a computer monitor. The LUT maps the image intensity values to brightness values. There are two forms of a LUT that an image can be processed by. The first one is a set of values that run between 0 to 255 and each pixel's intensity values is mapped to a location on the LUT. Each pixel data becomes an index in which to look up a pixel's intensity on the array of value. The second way is to pass each pixel value through an equation on the fly that alters the brightness value of each pixel.

When the LUT is a linear equation with a slope of 1 with a 0 intercept, the displayed image is an exact representation of the underlying image data (Figure 6). A pixel that is with the absolute black color has 0 intensity will be mapped by the LUT as a pixel with 0 brightness. By simply manipulations of the equation of the LUT, one can create many different visual effects from the image's data.

One of the most common examples of using LUT manipulation on an image is to control the brightness and the contrast of each pixel. The equation of the LUT is in the form of a linear equation with different intercepts:

*New intercept = slope * intensity + y-intercept*, intensity is the level of brightness and is between 0 and 1.

When manipulating the brightness of an image, the slope of the equation is a constant 1 but the intercept has changed. When the intercept is less than 0 the displayed image is darken and when the intercept is greater than 0, the displayed image is lighten (Figure 6).

To manipulate the contrast of an image, this equation is changed where the intercept equals:

$$\frac{1}{2}(1 - slope).$$

The contrast is increased as slope increases and the intercept decreases. This is the opposite when the contrast of the displayed image is lowered. The displayed image's pixels' intensities are inverted, black is mapped to white and vice visa, when the slope of this equation is -1 and the intercept is 1 (Figure 7).

When the smallest image's pixel intensity value is mapped to absolute black and the largest intensity value is mapped to absolute white, is called Autoscaling. This is usually the quickest way to create the best contrasting without creating any saturation of white or black. One issue with this is that outlier pixels intensity values will cause a low-contrast image. To counteract this, the max and min values are mapped to the ends of the main body of 98% of the image data.

All of these examples and more (Figures 6, 7) alter the appearance of an image by simply manipulating the Look-up Table. Such manipulation leaves the image content intact so they are completely invertible. A feature of CrazZzy Filterz is to allow a developer to run a filter that only needs to manipulate the Look-up Table of the image to achieve the filter's results, rather than having to running the filter directly on the image content.

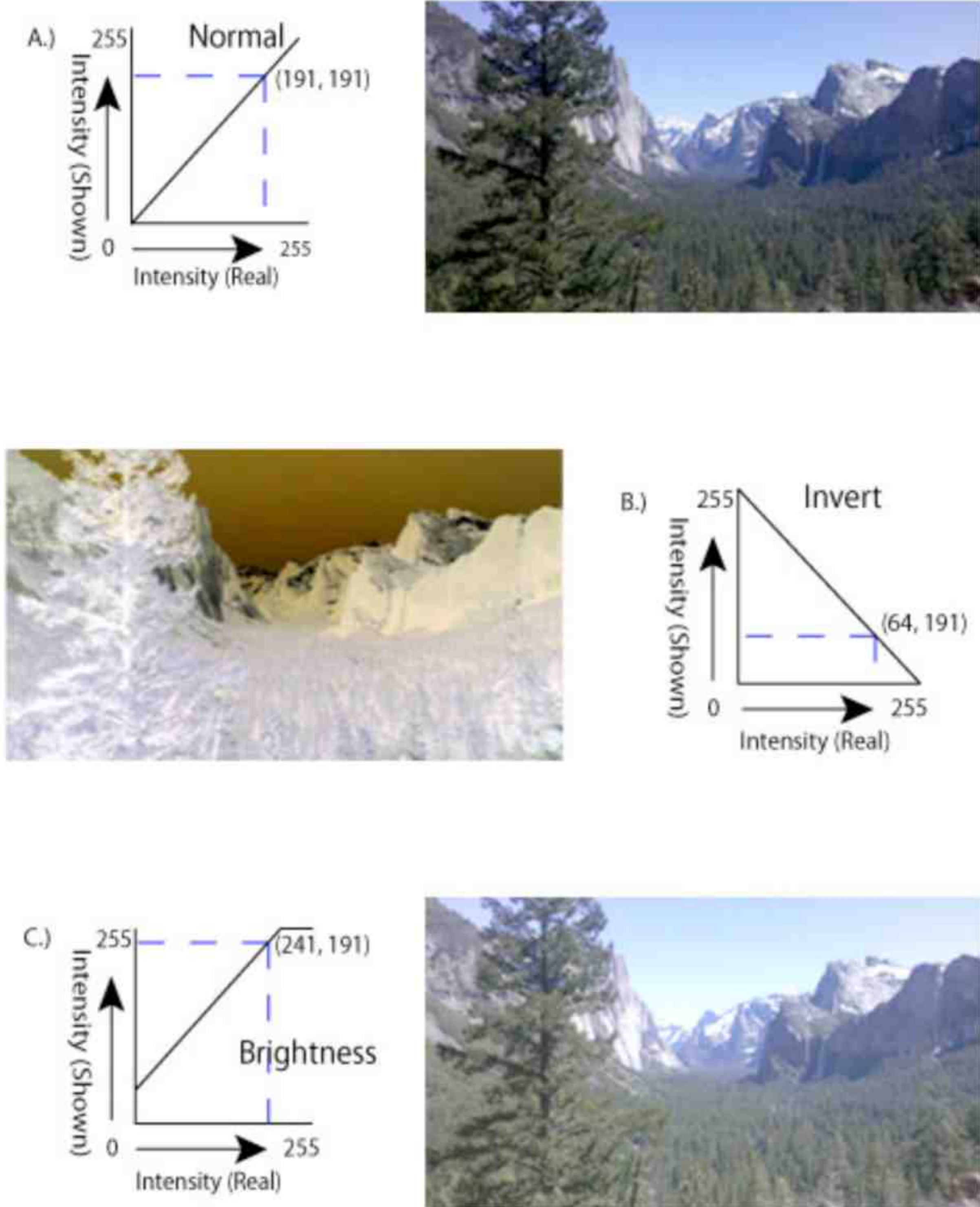

Figure 6. Graph: Look-up Tables - Normal, Invert, and Brightness

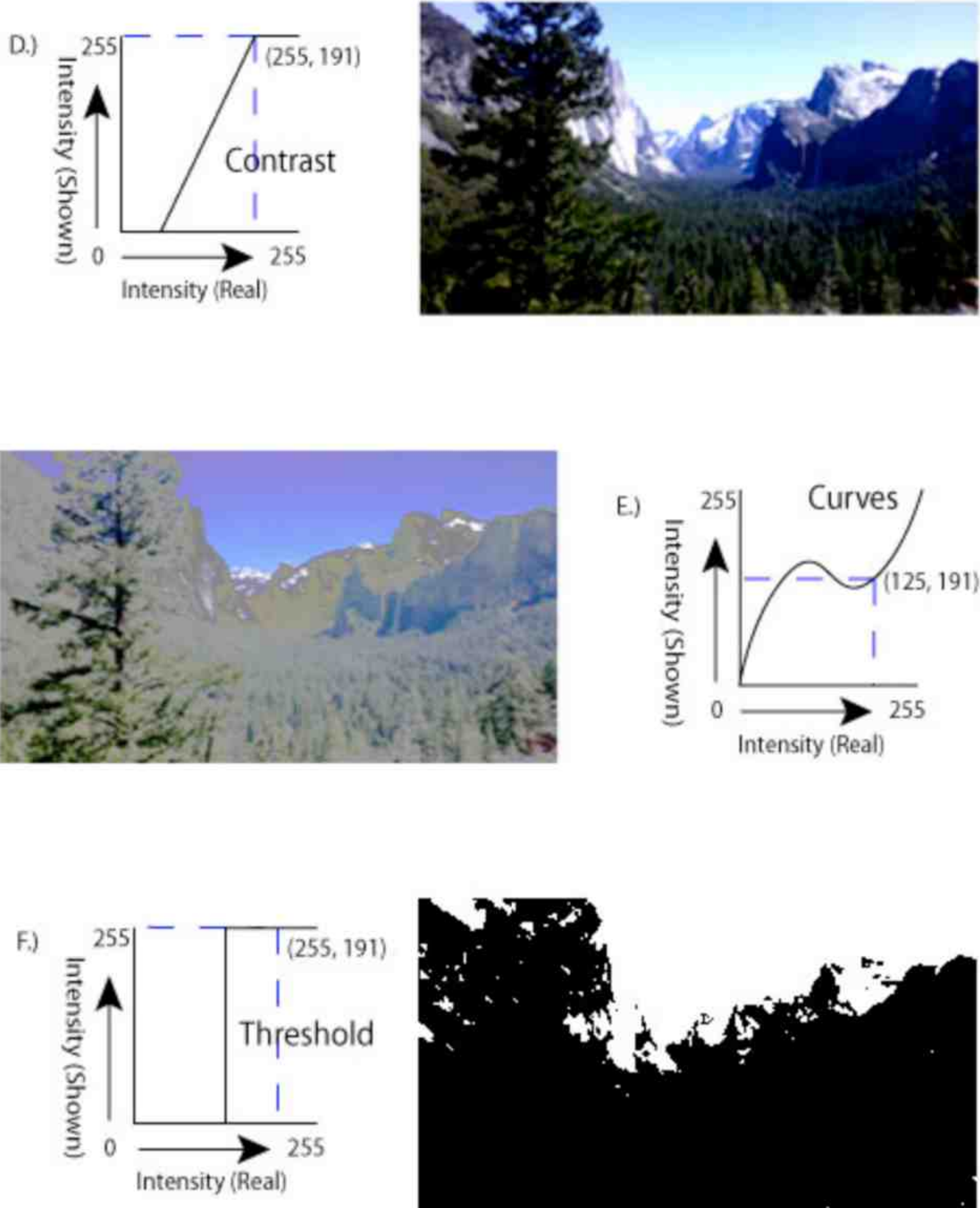

Figure 7. Graph: Look-up Tables: Contrast, Curves, and Threshold

Chapter 3: Technical details of the work

3.1 CrazZzy Filterz

Launching CrazZzy Filterz for that first time presents the user with a little information about it. The title is aligned on top of a red button bar located to the top or to the left depending on the orientation of the device. A transparent black background hides the default application's title art image and the main functionality of the application. The information about CrazZzy Filterz is written on top in white font (Figure 8). This screen slides to the opposite end to the device by pushing on the CrazZzy Filterz title with a finger and sliding it towards the opposite end.

Figure 8. CrazZzy Filterz, when it is first launched

Once the screen has slid to the opposite end of the device, the default application's art image comes in to focus. This is CrazZzy Filterz main application's screen. The default image sits on top a solid black background along with three buttons. The two outer buttons have circle arrows designed on them, but rotating in opposite directions. These buttons are used to rotate the image clockwise or counter-clockwise. The middle button with a design of a pointing hand over a 2 by 2 grid is used to display all the available filters to the user (Figure 9).

Figure 9. Main Display Screen

The mobile device's hard menu button is used to activate a menu screen which slides open from the top or left of the screen. When this menu is opened, two buttons are revealed. One of the buttons represents an old floppy disk and the other is a magnifying glass over a picture. The two buttons are used to load a new image or save the displayed image (Figure 10).

Figure 10. Opened main menu

When selecting to load a new image, CrazZzy Filterz opens an outside application which is used by the mobile device as its default camera gallery application. This application reveals all the images stored on the device. Once the image is selected, the gallery application returns to the main display screen of CrazZzy Filterz with that particular image modified by the current chosen filter in the place of the previous CrazZzy Filterz's default image (Figure 11).

Figure 11. Selection of an image

When the user selects to save the image from the menu, the image is saved to the same location that the image originally came from. CrazZzy Filterz changes the name of the new image to save the original image from being over-written by the new filtered image.

The All Filters Screen fades in to view when the user presses the middle button on the main screen. This screen is populated with every different filter that has previously been loaded into CrazZzy Filterz. The appearance of each filter is the displayed image modified by that filter using the default values for the filter's parameters. There is also a "Cancel" button located on the bottom or right of screen. This button's icon is a circle with an "X" cut out of it. When this button is pressed, it returns the user back to the display screen without choosing a particular filter, acting much like the mobile device's back button does (Figure 12).

Figure 12. All Filter Screen

The menu is different on the All Filter Screen. There are also two buttons located on the menu when it is activated but are for very different purposes (Figure 13). The first button is labeled "Load a Filter" and is used to load a new filter in to CrazZzy Filterz from the mobile device. That button generates a list of all the filters that are available to the application by searching for any filters that are located in the Image Filters folder on the mobile device (Figure 14).

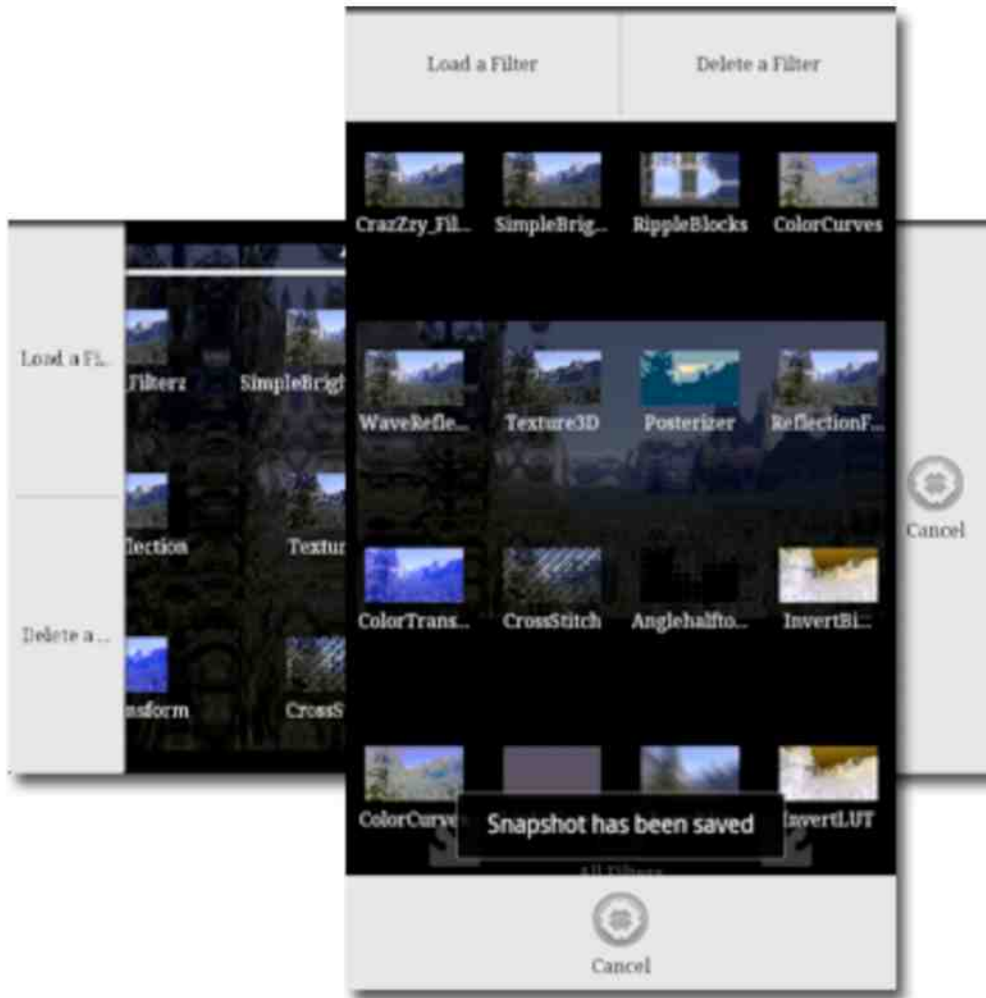

Figure 13. Application displaying the Filters Window Menu Options.

Figure 14. Loading a new filter

The second menu button is label "Delete a Filter" and is used to remove one of the filters that is on the All Filter Screen. Once the button is pressed a small red "X" appears over every one of the filters to indicate the ability to remove that filter from the application (Figure 15). Note: Removing the filter this way does not remove the Filter File from the actual device. To cancel this process is as simple as pressing the cancel button located to the bottom or right of the screen, depending on the orientation of the device.

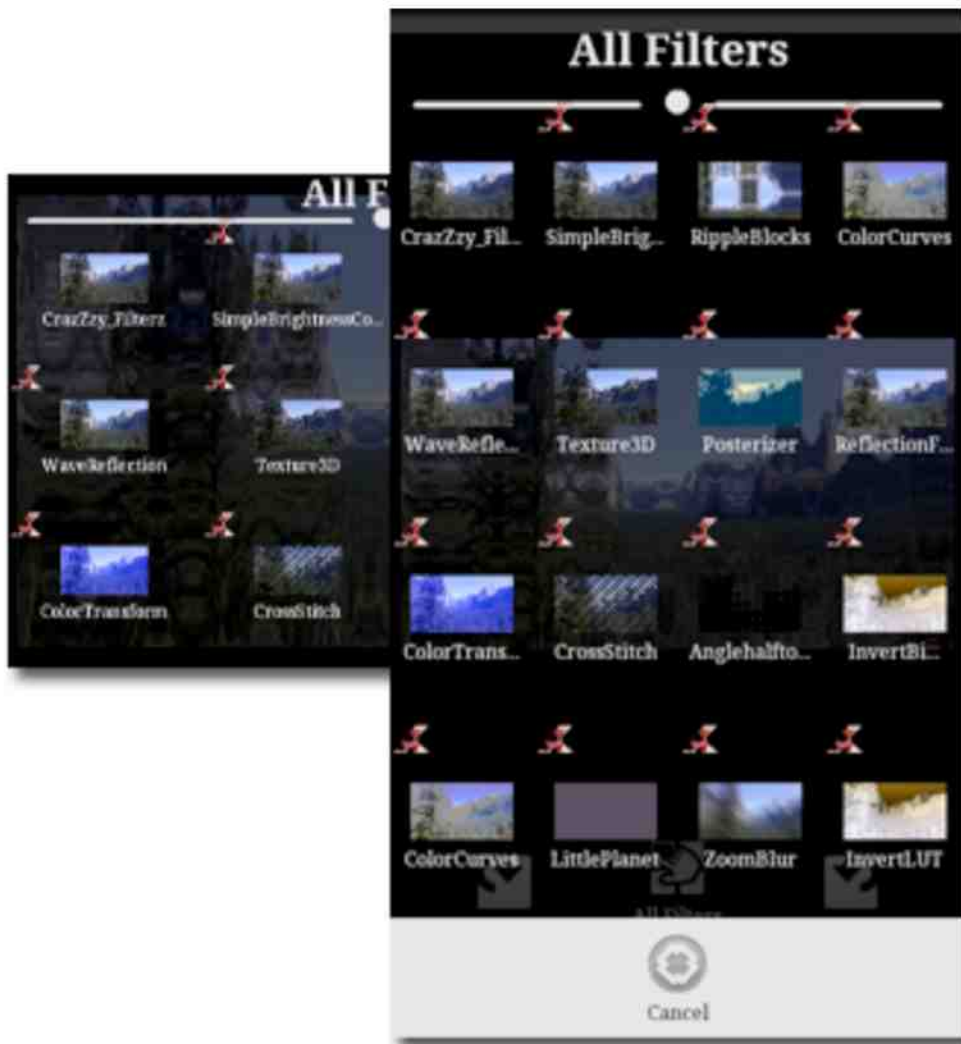

Figure 15. Deleting a filter

Once a filter is chosen, the All Filters Panel fades out to bring the user back to the main application's screen. The main display image has the new selected filter applied to it, and the CrazZzy Filterz title is updated to reflect the applied filter's name (Figure 16). The sliding screen is also updated to reflect the new chosen filter.

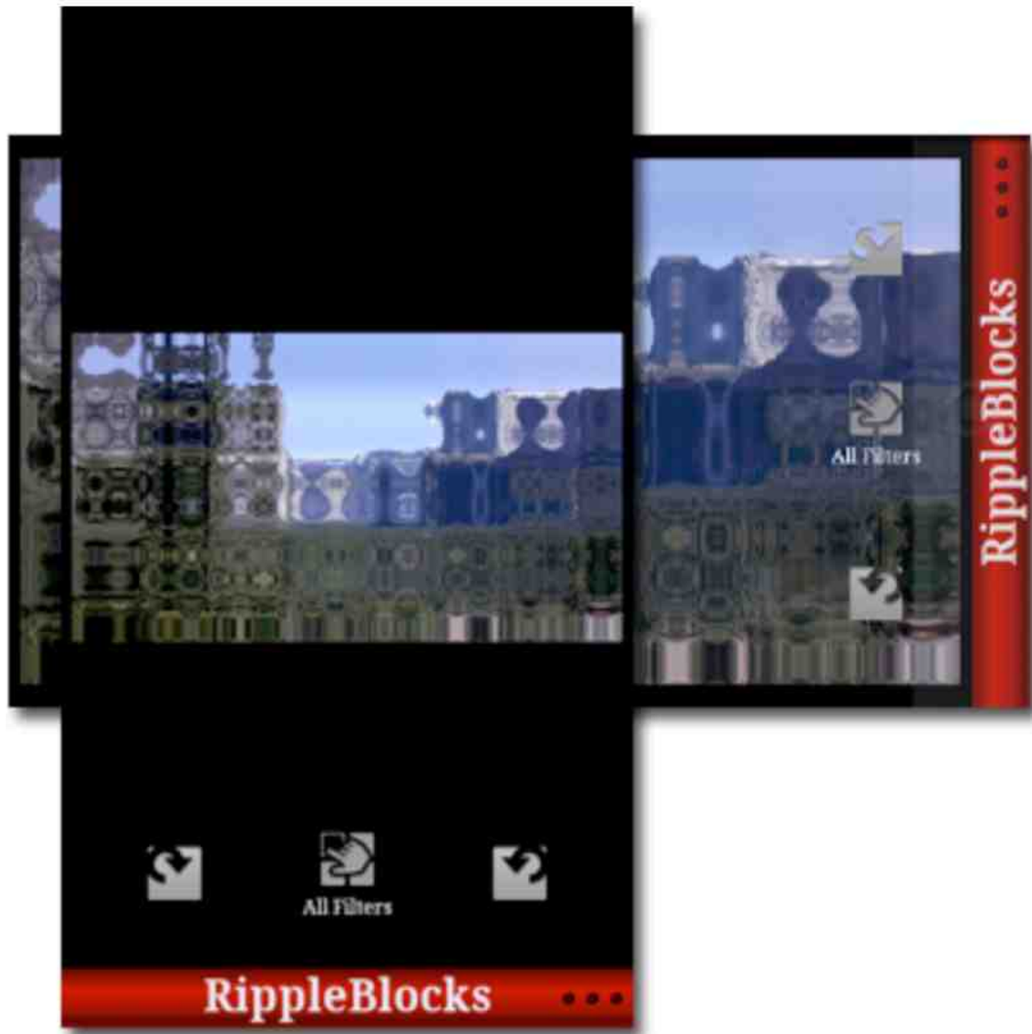

Figure 16. Main displayed image with a filter applied to it

The Parameter Panel is the sliding screen that has been previously mentioned and it slides open and close. The user pushes on the title tab of the panel and drags their finger to the opposite side of the screen causing the panel to slide open or close. The panel contains the basic information of the filter, a description along with the parameters of the filter. The parameters are contained in a scrolling list that scrolls when the user slides his or her finger over it. Each parameter is selectable by tapping it (Figure 17).

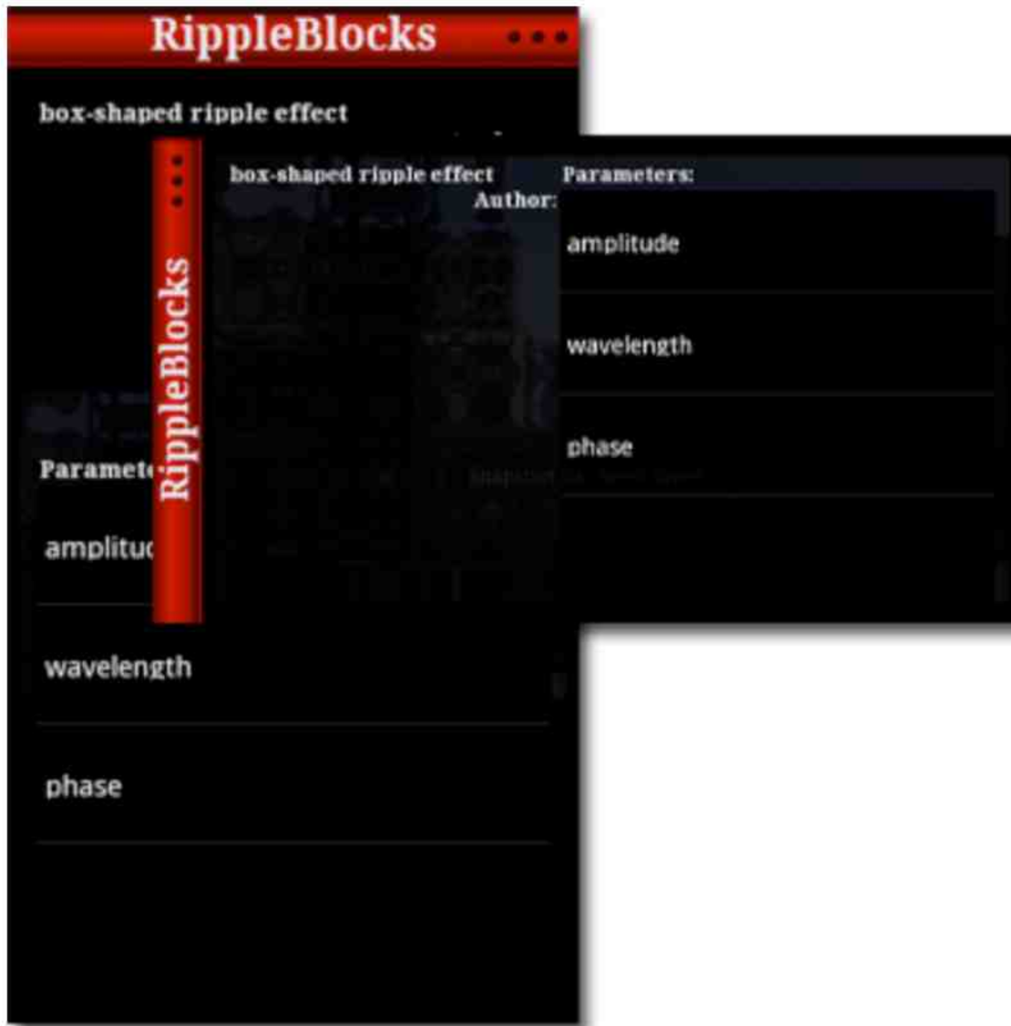

Figure 17. Parameter Panel showing the description along with the parameters

Once one of the parameters is chosen by being pressed, a pop-up screen appears with two buttons arranged at the bottom or right. One of the buttons is labeled “Cancel” and returns the user back to the parameters panel un-doing any changes that were made while the pop-up screen was present. The other button is labeled “Done” and also returns the user back to the to the parameters panel but sets that filter’s parameter with the edited new value. At the top of the pop-up screen reads the name of the filter’s parameter in which the user has chosen. Positioned below the name of the filter are a number of sliders ranging from 1 to 4. Each slider has a double arrowed value indicator revealing the current value of the parameter. Above each slider, its current value and data type are written out (Figure 18).

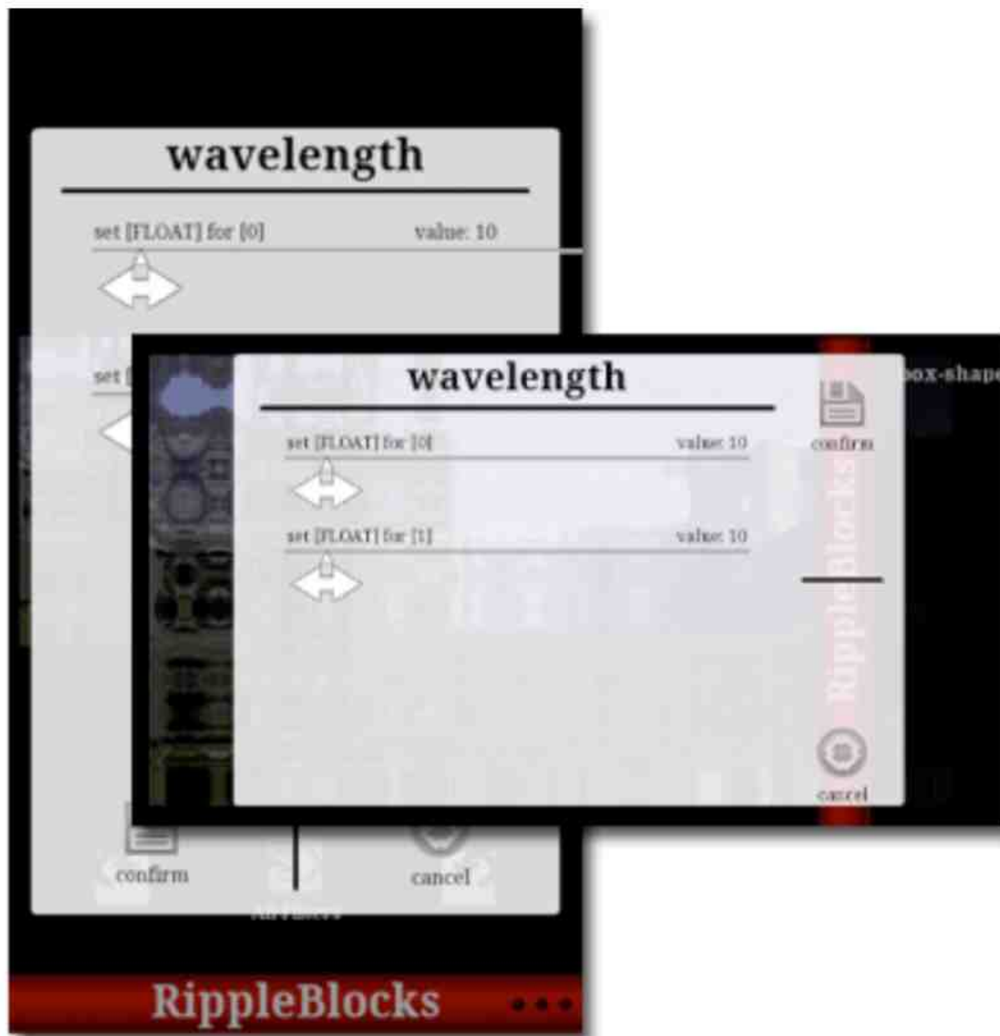

Figure 18. Pop up window used to modify a parameter of a filter.

The value for a specific parameter is set by dragging the double-headed arrow of one of the sliders from one end to the other. As a parameter's value is changed, the filter is updated and reapplied to the image on the display screen, which is only slightly visible due to the slightly transparent background color of the parameter panel. As the value is changing, an up-to-date representation of the displayed image with the filter reapplied using the new value is shown in red pop-up box above the parameter's slider (Figure 19).

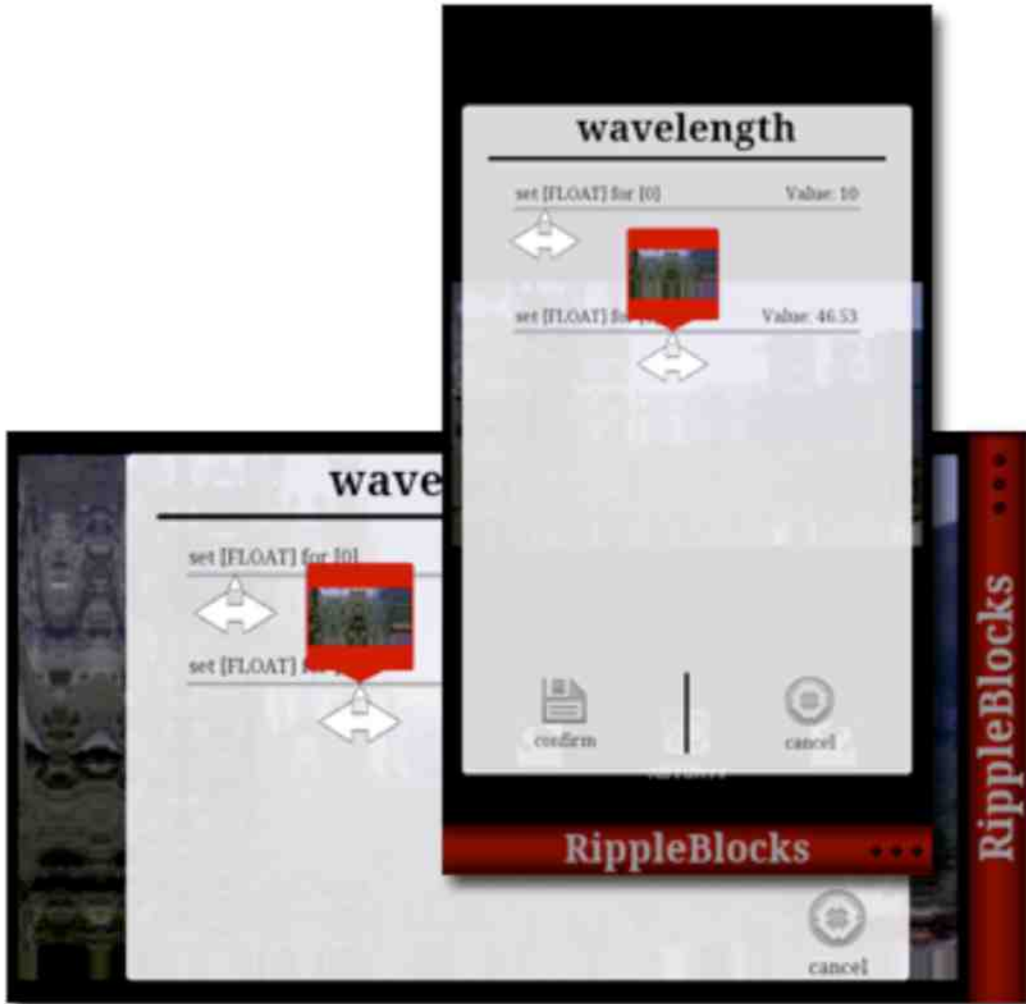

Figure 19. Change in value is reflected in a small pop-up above the slider

The user confirms the desirable values for the parameter and is returned to the application's main screen, where the displayed image reflects the new changes to the parameters of the filter. CrazZzy Filterz achieves all of this with minimal sluggishness and wait times in application's responses thanks to the "tricks" and methods that were used in its development.

3.2 Look-up Table Filter

To allow the developer of a filter the option to run it upon the LUT of the image, CrazZzy Filterz uses the Look-up Table Applying Filter, which is developed as a Pixel Bender Filter. CrazZzy Filterz represents the LUT of the images as an image so that any Pixel Bender Filter could be applied to it (Figure 20). This LUT image is used by the Look-up Table Applying Filter (LUT Applying Filter) when it is applied to the original image.

Figure 20. The LUT Image

The LUT Image has a 256 by 1 pixel dimension. This is used by the LUT Applying Filter as a reference table of the color intensities in which the image's pixels will be mapped too. When there is no change in the LUT image, then the image pixel's intensities are equal to their column location and the displayed image appears unaltered. This LUT image contains all the 256 shades of black and white and the pixels are arranged by intensity shades from darker to lighter.

```
{
   input image4 src;
   input image4 lut;

   output pixel4 result;

void evaluatePixel()
   {
      pixel4 value = sampleNearest(src, outCoord());
      pixel1 alpha = pixel1(sampleNearest(lut, float2(value.a * 255.0, 0.0)).a);
      pixel1 red = pixel1(sampleNearest(lut, float2(value.r * 255.0, 0.0)).r);
      pixel1 green = pixel1(sampleNearest(lut, float2(value.g * 255.0, 0.0)).g);
      pixel1 blue = pixel1(sampleNearest(lut, float2(value.b * 255.0, 0.0)).b);

      result = pixel4(red, green, blue, alpha);
   }
}
```

`LUT Applying Filter`

The LUT Applying Filter (above) requires the LUT Image as input so that it can be used as a database of the color intensities. The filter first takes a base image in which this filter will run on and for each pixel changes the values of the red, green, and blue color intensities along with the pixel's alpha channel to the locations of the values located on the LUT Image. The new intensity values of the displayed image are determined by taking the original images pixels' intensities to be used as the column locaters to retrieve the new intensities from the LUT Image.

To apply the LUT Applying Filter to a selected image, it requires CrazZzy Filterz to store the LUT image and the selected image. The LUT Bitmap Class (below) applies the LUT Applying Filter to the selected image. The LUTBitmap Class extends the Bitmap Class and the _LUTFilter is the LUT Applying Filter Class representation. LUTBitmap requires that it be created with at least the selected image by passing it in as the bitmapData parameter. The bitmapData is passed to the Bitmap Class in which it extends to initialize it. The next parameter could be used for the Look-up Table image and is called LUT. If LUTBitmap is passed the LUT image then it initializes the LUT Applying Filter to be applied. LUTBitmap passes the LUT image and the selected image to the LUT Applying Filter. LUTBitmap uses a ShaderJob Class to run the LUT Applying Filter and sets the image of the LUTBitmap with the returned result.

```
[Embed ( source="res/LUTFilter.pbj", mimeType="application/octet-stream" ) ]
 private var _LUTFilter:Class;

public function LUTBitmap(      bitmapData:BitmapData,
                                LUT:BitmapData = null,
                                pixelSnapping:String ="auto",
                                 smoothing:Boolean = false) {

        super(bitmapData.clone(), pixelSnapping, smoothing);

        if(LUT != null && LUT.width == 256 && LUT.height == 1) {
                this._shader = new Shader(new this._LUTFilter() as ByteArray);
                this._shader.data.lut.input = LUT;
                this._shader.data.src.input = bitmapData;

                // shader jobs
                this._job = new ShaderJob();
                // ShaderJob returns to this object
                this._job.target = this.bitmapData;
                // The Shader assigned to this job
                this._job.shader = _shader;
                this._job.start( true );
}
```

ActionScript 3 class used to ran the Pixel Bender LUT Applying Filter on an image

To help better see the flow of how a filter would be applied to the LUT image and then have that image be applied to the selected image please refer to Figure 21.

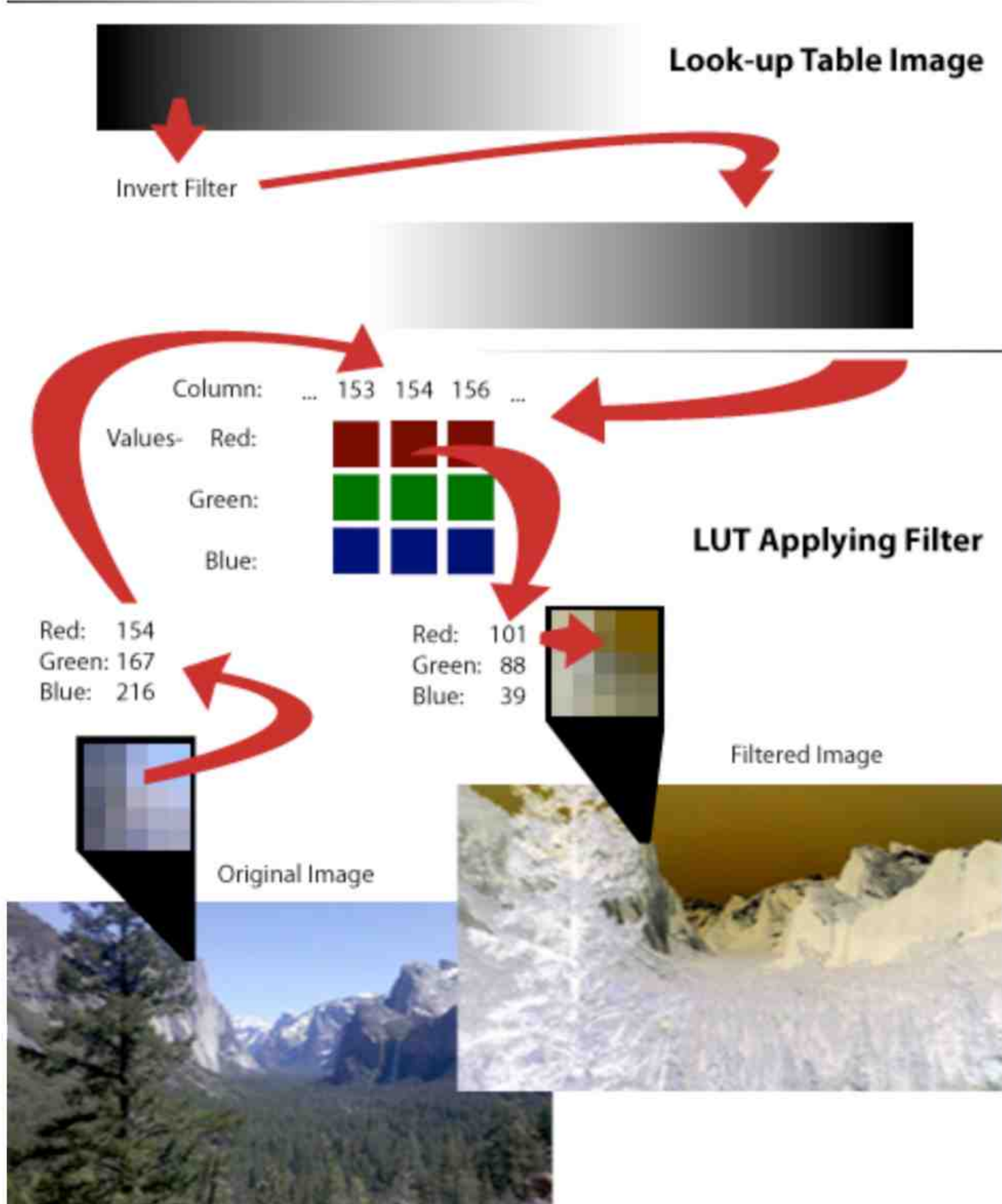

Figure 21. The Look-up Table Walkthrough

3.3 Resizing images using bilinear scaling

CrazZzy Filterz resizes the selected image once it is loaded in from the mobile device. The new size is decided by the dimensions of the devices screen. The selected image has the Bilinear Scaling Filter applied to it (below) with a scale that resizes the image to fit on the devices screen. The Bilinear Scaling Filter calculates each pixels intensity value by taking the linear value of the position of the new pixel.

```
parameter float scale
<
  minValue: 0.0;
  maxValue: 1000.0;
  defaultValue: 1.0;
>;
input image4 src;
output pixel4 dst;

void
evaluatePixel()
{
  // scale should be Math.max( src.width / output.width, src.height / output.height )
  dst = sampleLinear( src, outCoord() * scale ); // bilinear scaling
}
```

The Pixel Filter used for Bilinear Scaling

CrazZzy Filterz resizes the image by calling the public static function resizeByPBJ (below) which returns a new BitmapData with a size where it would fit in between the desiredWidth and desiredHeight without changing the image's dimensions ratio. The aspectRatio is calculated by the width divided by the height of the input image. The factor is calculated by the finding the largest number between the input image's width divided by the desiredWidth and the input image's height divided by the desiredHeight. The new image's size is determined by rather the input image's width is greater than the input image's height. If that is true, then the new image's width will equal the desiredWidth and the image's height will equal desiredWidth / aspectRatio, but if it isn't true, then the new image's width will equal desiredHeight * aspectRatio and the image's height will equal the desiredHeight. Once the new size has been determined for the new image then a ShaderJob is initialized to run the Bilinear Scaling Filter using the factor number and the result is stored in the as the new sized image.

```
[Embed ( source="res/bilinearresample.pbj", mimeType="application/octet-stream" ) ]
private static var BilinearScaling:Class;

//function that resizes a image by bilinear interpolation using pixel bender
public static function resizeByPBJ( input:BitmapData, desiredWidth:int,
                                    desiredHeight:int, cleanup:Boolean = true ):BitmapData {

        var aspectRatio:Number = input.width / input.height;
        var factor:Number = Math.max( input.width / desiredWidth, input.height / desiredHeight );

        // create and configure a Shader object
        var shader:Shader = new Shader();
        // instantiate embedded Pixel Bender bytecode
        shader.byteCode = new EasyBitmapData.BilinearScaling();
        // supply the shader with BitmapData it will manipulate
        shader.data.src.input = input;
        shader.data.scale.value = [factor]; //entered as an array
        var output:BitmapData;// shader will return its data (an image) to this bitmap
        // determine output bitmap dimensions
        ( input.width > input.height )?
                output = new BitmapData( desiredWidth, desiredWidth / aspectRatio ):
                output = new BitmapData( desiredHeight * aspectRatio, desiredHeight );

        var job:ShaderJob = new ShaderJob();
         // ShaderJob returns to this object
        job.target = output;
        // The Shader assigned to this job
        job.shader = shader;
        job.start( true );// runs the job synchronously

        if ( cleanup ) input.dispose();

        return output;
}
```

ActionScript 3 function that resizes an image

3.4 Controlling the Orientation

CrazZzy Filterz controls its layout by calling the doLayout Function (below) every time there is a screen resize. When a user rotates a device to change the orientation of the screen, the screen's width and height values are reversed and that fires off the Resize Event. Once that Event is fired the doLayout method is called and depending on the largest value between the width and height of the screen, objects on the screen are re-positioned.

When the doLayout Function is called the background is set to the stage size and the pop-up screen used to modify the parameters of the filters along with the All Filter screen have their layouts changed according to the orientation of the device. After that CrazZzy Filterz changes the layout of the main screen. The three buttons of the main screen are repositioned. The Parameter Panel and the menu are also repositioned.

```
private function doLayout(e:Event = null):void
{
	this._bg.width = this.stage.stageWidth;
	this._bg.height = this.stage.stageHeight;

	this._allFilters.layout(this.stage.stageWidth, this.stage.stageHeight);
	this._modifier.layout(this.stage.stageWidth, this.stage.stageHeight);

	switch(getOrientation())
	{
		case Orientation.PORTRAIT:
			this._bg.height -= 50;

			this._parameters.position = PanelPosition.BOTTOM;
			this._menu.position = PanelPosition.TOP;

			this._rotateCW.x = this._bg.width * .2 - this._rotateCW.width * .5;
			this._rotateCW.y = this._bg.height * .9 - this._rotateCW.height * .5;
			this._rotateCCW.x = this._bg.width * .8 - this._rotateCCW.width * .5;
			this._rotateCCW.y = this._bg.height * .9 - this._rotateCCW.height * .5;

			this._previewAll.x = this._bg.width * .5 - this._previewAll.width * .5;
			this._previewAll.y = this._bg.height * .9 - this._previewAll.height * .5;
		break;
		case Orientation.LANDSCAPE:
			this._bg.width -= 50;

			this._parameters.position = PanelPosition.RIGHT;
			this._menu.position = PanelPosition.LEFT;

			this._rotateCW.x = this._bg.width * .9 - this._rotateCW.width * .5;
			this._rotateCW.y = this._bg.height * .2 - this._rotateCW.height * .5;

			this._rotateCCW.x = this._bg.width * .9 - this._rotateCCW.width * .5;
			this._rotateCCW.y = this._bg.height * .8 - this._rotateCCW.height * .5;

			this._previewAll.x = this._bg.width * .9 - this._previewAll.width * .5;
			this._previewAll.y = this._bg.height * .5 - this._previewAll.height * .5;
		break;
	}
}
```

ActionScript 3 Function doLayout

3.5 Integration

CrazZzy Filterz uses a database and creates a local file on the mobile device as a place to put new filters that the user will like to add to the application (below). CrazZzy Filterz first checks to see if there is the Image Filters Folder on the device and create one if it doesn't exist. CrazZzy Filterz uses a SQL Database which holds the names of the loaded filters along with their paths to load them into the application. This allows for the user to add any new filter that is located in the folder and automatically loads the already saved filters that the user has already added.

If CrazZzy Filterz is creating the Image Filters Folder then a set of pre-installed filters is copied from the pre-installed folder which was included with the installation of CrazZzy Filterz to the new Image Filters Folder.

```
public function FilterDB()
	{
		this._filterObjs = [];
		var dbFile:File = new File(this.DB_LOC);
		this._sqlConnection = new SQLConnection();
		this._sqlConnection.open(dbFile);
		var externalFile:File = File.documentsDirectory.resolvePath(this.FILE_LOC);
		selectItems(null);
		if (!externalFile.exists)
		createExternalFile(externalFile);
	}

protected function createExternalFile(exFile:File):void
	{
		var preinstalledFile:File = new File("app:/res/pre-installed");
		preinstalledFile.copyTo(exFile);
		var list:Array = exFile.getDirectoryListing();
		for (var i:uint = 0; i < list.length; i++)
		    if(!this.contains(list[i].url))
			addNewFilterItem(String(list[i].name).split(".")[0], list[i].url);
	}
```

Class and Method used to create the database

In addition CrazZzy Filterz is always awaiting certain events to happen. If any of these events are fired, CrazZzy Filterz acts accordingly. When the back button on the device is pressed, CrazZzy Filterz returns to its previous state and when the menu button on the device is pressed, CrazZzy Filterz activates the menu panel.

Chapter 4: Experiments

4.1 Testing Environment

All testing and experimenting during my thesis was conducted on my mobile phone. The phone is a Motorola Droid 2 running the Android 2.2 (Froyo) OS. It is rooted and has a TI OMAP 1GHz processor with dedicated GPU. The processor is over-clocked and running at 1.3GHz along with 512 MB of RAM. The first thing that I tested was to ensure that a Pixel Bender filter would run in ActionScript 3 on the mobile device. The results confirmed that I could continue to develop CrazZzy Filterz in ActionScript 3.

4.2 Testing the different languages

Android vs. ActionScript

After I weighed my pros and cons on if I should attempt to build my application in the native programming language of Android or in ActionScript 3.0, I ran a test to see how much of a speed performance I would be giving up. I created a simple application written in Android that uses a 288 by 216 image and turns it pixels' intensities inverted by a touch of a button. The time that it takes for all the pixels in the 288 by 216 image to have their color intensity to be reversed was recorded and displayed.

```
protected Bitmap invertRGB(Bitmap b){
      int picW = b.getWidth();
      int picH = b.getHeight();

      int[] pix = new int[picW * picH];
      b.getPixels(pix, 0, picW, 0, 0, picW, picH);

      for(int y = 0; y < picH; y++){
            for (int x = 0; x< picW;x++){
              int index = (y * (picW)) + x;
              int red   = android.graphics.Color.red(pix[index]);
              int green  = android.graphics.Color.green(pix[index]);
              int blue   = (pix[index]&0xFF);

              int iRed = 255 - red;
              int iGreen = 255 - green;
              int iBlue = 255 - blue;

            pix[index] = android.graphics.Color.rgb(iRed,iGreen,iBlue);
            }
      }
      return Bitmap.createBitmap(pix, picW, picH, b.getConfig());
}
```

Inverted Filter written in Android

The Inverted Filter (above) that is written in Android first creates a new bitmap by breaking up each pixel data from the passed in bitmap into the three different color channels. Once the channels are isolated, they are then subtracted from the maximum intensity of a color shade, 255. The pixels are then used to populate the new returning bitmap.

I compared this Android Function to one that I created in Pixel Bender.

```
input image4 src;
output pixel4 dst;

 void evaluatePixel()
 {
    dst.rbg = 1. - sampleNearest(src,outCoord()).rbg;
    dst.a = sampleNearest(src,outCoord()).a;
 }
}
```

Inverted Filter written in Pixel Bender

The evaluatePixel Function of the Pixel Bender Filter (above) computes the value of each pixel in a bitmap by substracting the intensity by 1. Pixel Bender already separates the color channels of each pixel and represents each channel's value as a number between 0 and 1.

I ran both filters 30 different times each. The results of the testing below shows the lapsed times in milliseconds that both approaches produced while inverting the colors in the 288 by 216 image.

Written language	Lapse Time in milliseconds				
Android	93	93	116	83	96
	100	95	91	94	114
	103	95	92	45	96
	44	114	92	105	94
	92	76	112	97	93
	93	95	98	91	45
ActionScript 3 and PixelBender	186	190	227	145	201
	205	196	176	193	221
	207	198	178	144	201
	143	213	180	211	193
	178	145	212	203	191
	190	199	204	173	143

Figure 22. Data Chart: Lapse Times of Invert Filter – Android vs ActionScript 3

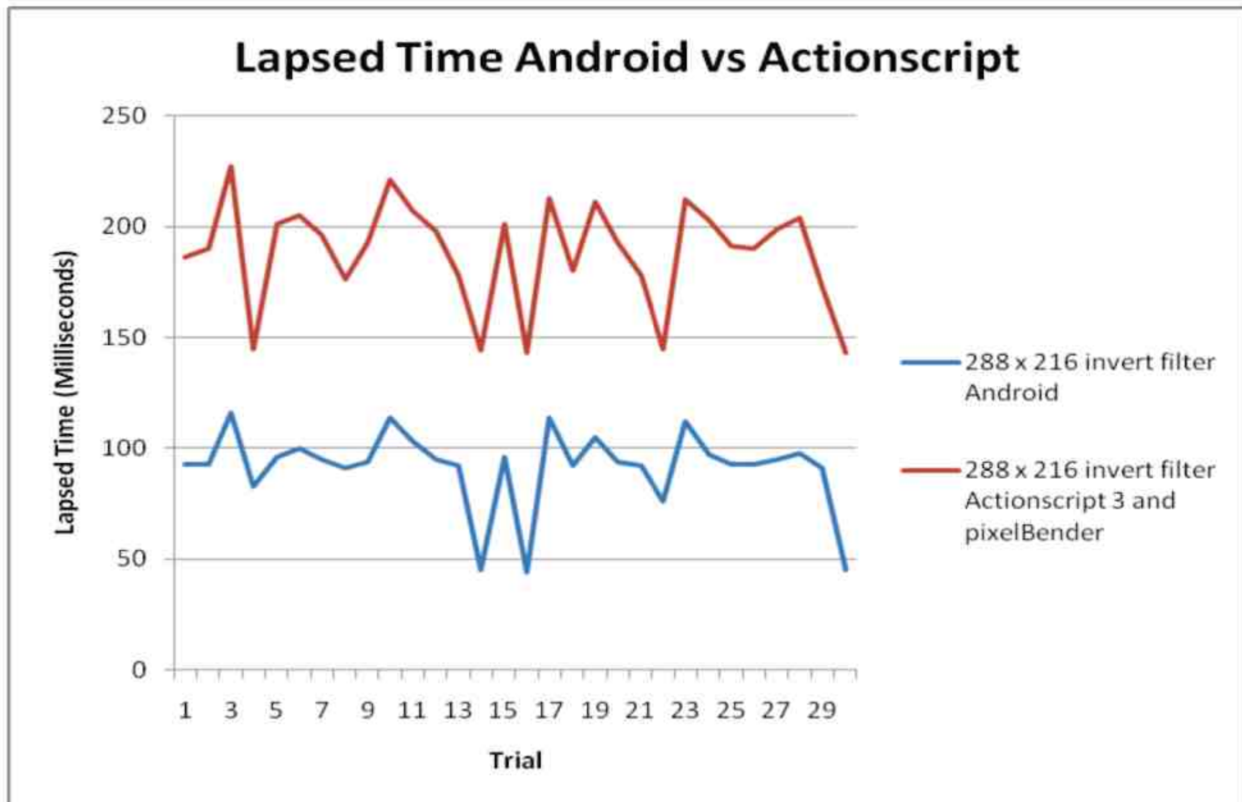

Figure 23. Graph: Lapsed Times of Invert Filter- Android vs ActionScript

Each approach fluctuates within 85 milliseconds between their maximum and minimum times. The average of the Android approach was 91.57 milliseconds and the average of ActionScript 3 approach was 188.2 milliseconds which is double that of the Android approach. This is a very significant difference in value and meant that filters written in Pixel Bender and being used by ActionScript 3 would run about 2 times slower than those filters that are written directly in Android.

4.3 Testing Speed and Accuracy

To demonstrate the improved performance of the filters ran upon an actual full size image stored on a mobile device and a resized image, I ran the Invert Filter 30 times on both forms of the image. I ran the Invert Filter on the original size image 30 times and then I re-ran the Invert Filter 30 more times on the resized dimensions. Images taken on my mobile device using the 5MP Camera creates images with dimensions of 2592 wide and 1456 high but the stage or screen of the mobile device only has a dimension of 480 wide and 854 high. Within CrazZzy Filterz the image is resized to a dimension of either 854 by 479 or 480 by 269 based upon the orientation of the phone and image at the time that the image is loaded into CrazZzy FilterZ.

Image size	Lapse Time in milliseconds				
Original Size 2592 x 1456	8815	8643	8900	8652	8711
	8658	8694	8622	8723	8936
	8715	8653	8670	8645	8694
	9725	8661	8653	8620	8659
	8603	8687	8661	8589	8610
	8672	8628	8600	8610	8588
Resized to fit screen 480 x 269	340	365	380	359	369
	345	340	348	403	348
	336	376	341	340	363
	360	361	344	338	349
	358	344	359	387	340
	350	385	339	346	361

Figure 24. Data Chart: Lapse Times of Invert Filter – Original Sized vs Resized

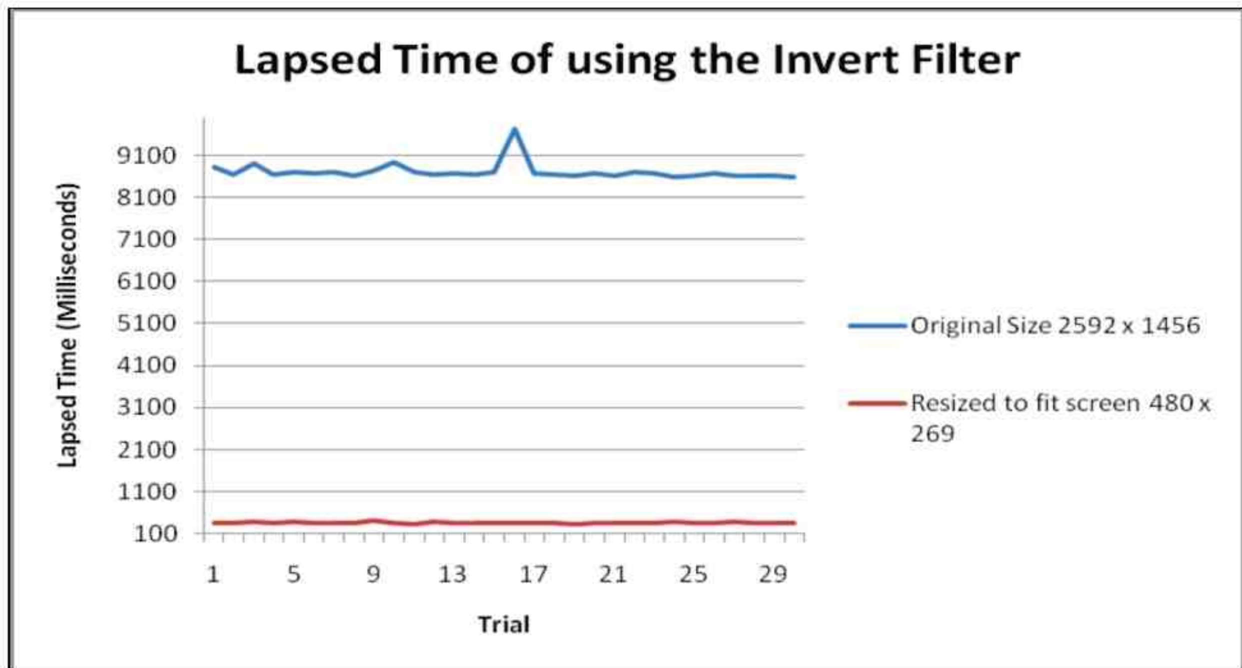

Figure 25. Graph: Lapse Times of Invert Filter – Original Sized vs Resized

When the Invert filter ran on the full-sized image, the computing time that lapsed was an average of 8709.9 milliseconds with a maximum lapsed time of 9725 milliseconds and a minimum lapsed time of 8588 milliseconds. The data shows that when the filter ran on the resized image, that it resulted in an average lapsed time of only 355.8 milliseconds with a maximum lapsed time of 403 milliseconds and a minimum lapsed time of 336 milliseconds. This data also shows that the lapsed time of the original size has a fluctuation of 1137 milliseconds but on the resized image the fluctuation is only 67 milliseconds.

To reveal the difference in image quality that the resized image has over the full size image, I compared the two histograms of the same fixed area from both the images once they were displayed onto the screen of the mobile device. A histogram is a graph of the total count of pixels at each color intensity level. To compare both of the histograms, screenshots were taken of the image displayed on my device, one when it was resized and one of the image left in its original resolution. I continue by opening these images in to Adobe Photoshop and extracted the histogram data from the same selected area of each image (Figure 26).

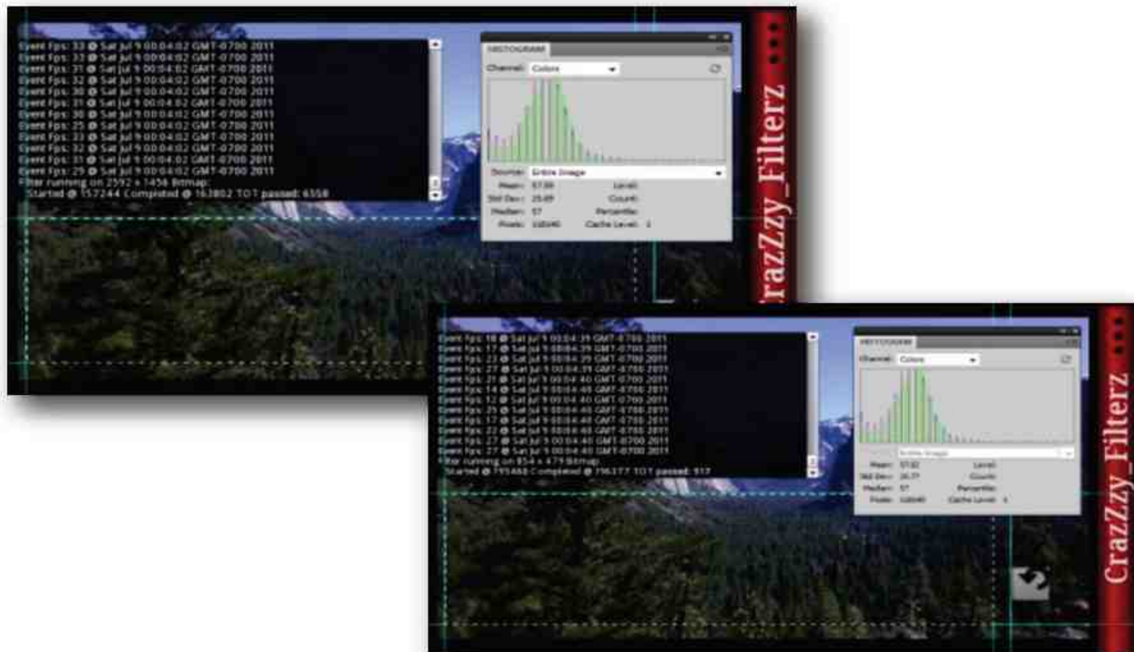

Figure 26. Picture Quality Test: Original Size vs resized

The histogram from 118140 pixels of the full-sized image revealed a mean of 57.59, a median of 57 along with a standard deviation of 25.69. The histogram from the same fixed area of pixels as the resized image revealed a mean of 57.02, median of 57 along with a standard deviation of 25.77. The graph below shows the histograms of the two images up to the 100 intensity mark to show how similar they are.

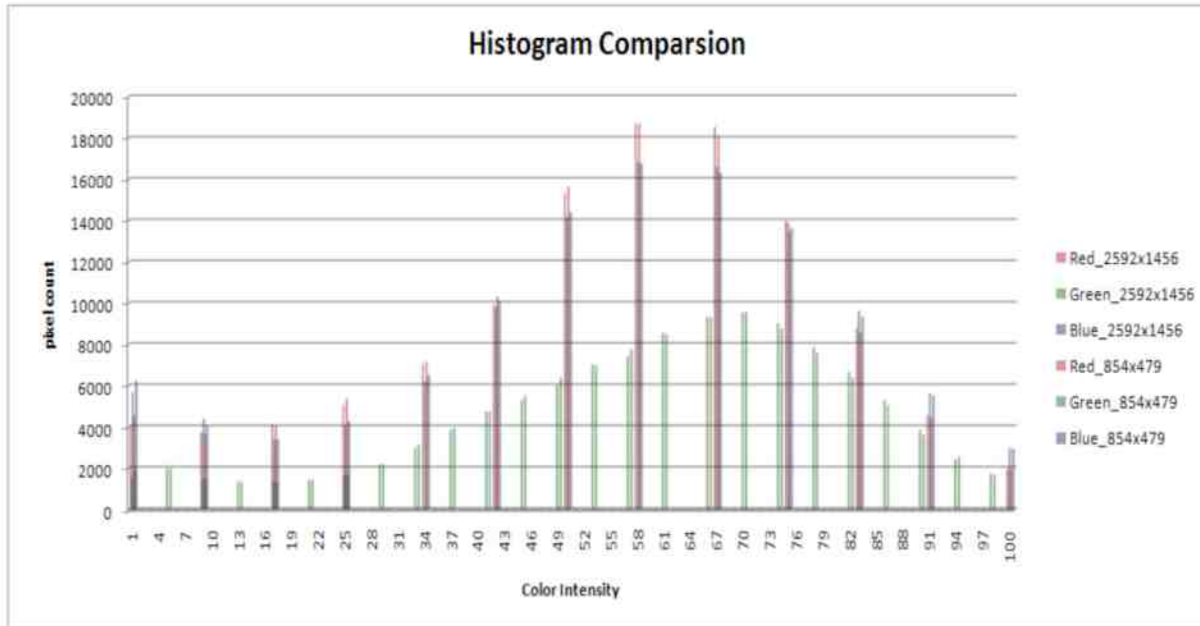

Figure 27. Graph: Histogram Comparsion – Original Size vs Resized

4.4 Using the LUT

To test the benefit of applying a filter upon the LUT of an image rather than the conventional way of applying it directly upon the image content, I applied two very different filters upon both the image and then again on its LUT and recorded the results. The two filters that were applied were the Invert Filter which consists of only a two line algorithm and a much more complicated Color Channel Curve Filter where the algorithm changes the intensity values of each color channel according to a Cubic Spline Interpolation of four different points. Each filter was applied 30 times upon a 480 by 269 image and then repeated but applied this time to the LUT of the image.

When the Invert Filter ran directly on the bitmap of the image, the data shows a lapsed time that has a range between a maximum of 403 and a minimum of 336 milliseconds, with an average of 355.8 milliseconds. When the Invert Filter ran on the LUT of the image, data shows an increase in lapsed time. The lapsed times have a range between a maximum of 600 and a minimum of 524 milliseconds, with an average of 539.7 milliseconds.

The data taken when the Color Channel Curves Filter was applied shows two very different data sets. When this filter ran directly upon the bitmap, the lapsed times have a range between a maximum of 4315 and a minimum of 3826 milliseconds. The average lapsed time of this data set is 3911.3 milliseconds. The data collected when this filter was run on the LUT, revealed lapsed times that have a range between a maximum of 599 and a minimum of 531 milliseconds. The average lapsed time of that data set is 548.53 milliseconds.

Filter Applied On	Lapse Time in milliseconds				
Invert Filter Bitmap	340	365	380	359	369
	345	340	348	403	348
	336	376	341	340	363
	360	361	344	338	349
	358	344	359	387	340
	350	385	339	346	361
Invert Filter LUT	544	552	529	527	528
	548	526	550	529	563
	528	527	527	529	528
	550	558	526	600	532
	584	527	529	528	526
	530	583	524	528	531

Figure 28. Data Chart: Invert Filter – Bitmap vs LUT

Filter Applied On	Lapse Time in milliseconds				
Color Curves Filter Bitmap	3851	3961	3876	3925	3896
	3974	3932	3857	3924	3948
	3892	4315	3865	3896	3852
	3871	3859	3992	3959	3859
	3916	3826	3885	3829	3994
	3872	3874	3892	3843	3904
Color Curves Filter LUT	562	553	533	537	532
	560	531	553	534	533
	554	538	599	534	568
	534	535	556	533	547
	537	550	547	533	546
	550	584	555	594	534

Figure 29. Data Chart: Color Curves Filter – Bitmap vs LUT

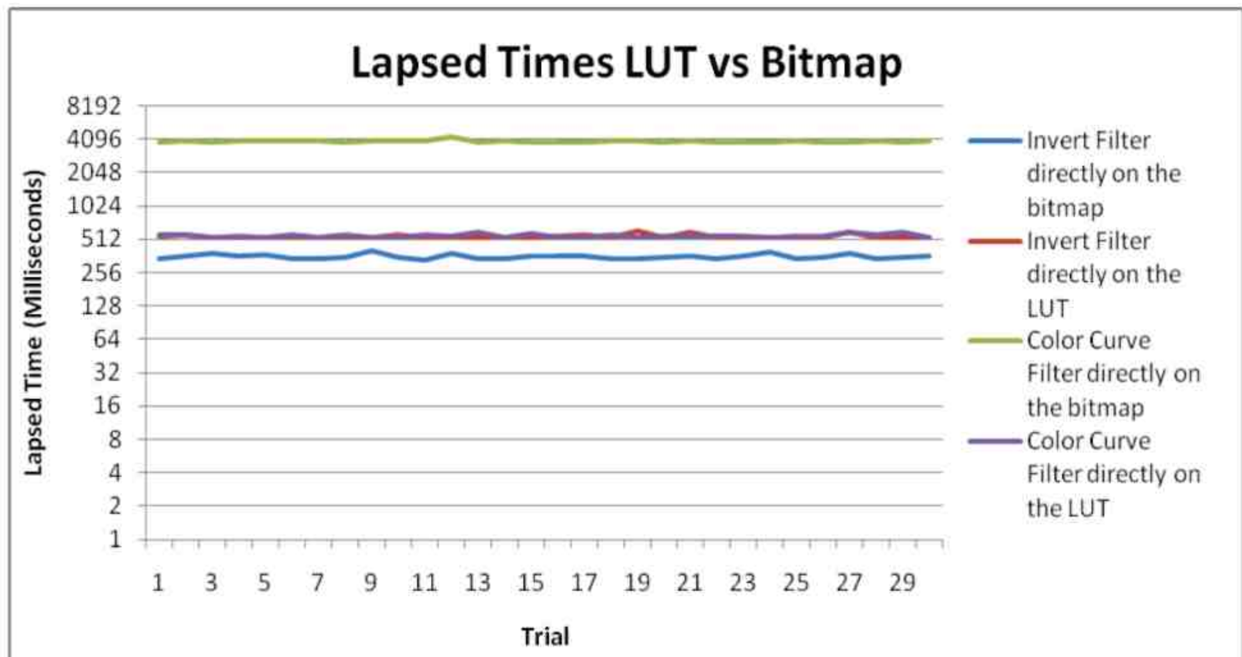

Figure 30. Data Graph: Invert and Color Curves Filters – Bitmap vs LUT

Chapter 5: Analysis of Results

5.1 Native Android is faster than ActionScript 3.0

After performing the Android vs. ActionScript 3 Performance Test back in section 4.2, it was apparently clear that it was going to be an uphill battle to create CrazZzy Filterz using ActionScript 3. The data reveals that if CrazZzy FilterZ was developed directly in Android, that it could possibly perform 2.055 times faster than being developed on Adobe AIR according to the average lapsed time of 91.57 milliseconds recorded from the Android Application and the average lapsed time of 188.2 milliseconds recorded from the ActionScript 3 written application. Reviewing the data (Figure 22) shows that the Android Application ran the fastest lapsed time of only 44 milliseconds while the Adobe AIR Application's closes lapsed time is 143 milliseconds. The Android Application demonstrates that it runs about 3.25 times faster than the Adobe AIR Application when comparing both of the applications' fastest lapsed times.

The range between the maximum and minimum lapsed times of the two approaches were similar with the Android Application having a range difference of 99 milliseconds and the Adobe AIR Application having a range difference of 111 milliseconds. This indicates that both approaches will return consistent and predictable results.

5.2 Smaller is better

The data collected from the testing in section 4.3, Testing Speed and Accuracy, revealed that when the filter was run on the full-sized image with dimensions of 2592 by 1456, that the average lapsed time is 8709.9 milliseconds (Figure 24). A 2592 by 1456 image consists of 3773952 pixels and each pixel requires the filter's algorithm to modify its intensity, so the algorithm average lapsed time was .0023 milliseconds per pixel.

Looking at the lapsed times of the filter that ran on the resized 480 by 269 image (Figure 24), shows an average lapsed time of 355.8 milliseconds. The averaged lapse time of performance on the resized image is 24.48 times faster than compared to the full-size image. The resized image consists of 129120 pixels which is 29.23 times smaller than that of the full-sized image. The filter's algorithm ran an average of .0028 milliseconds per pixel.

As expected the computing time per operation upon a pixel was roughly the same and the lapsed times created by each image was in proportional to its size and number of pixels needing to be modified.

The Quality Test of the full-size and resized image displayed on the mobile device provided an expected conclusion also. Any and all devices that display an image to a screen of different resolution must resize the image anyway to fit. The resized image should appear the same as a full-size image that is resized by the device. When the two histograms are compared, they reveal that they are nearly identical to each other (Figure 27).

A histogram is basically a bar graph representation of the total count of pixels with each different intensity value. The histograms of the two images were created from the same section of the images using the same amount of pixels (Figure 26). They showed that their means had a difference of only .57. They also have a difference in the standard deviation of only .08. These differences that are presented to the end user are only minor and nearly undetectable alternations between the two images.

5.3 Using the Look-up Table

The data collected from the LUT Tests from section 4.4 revealed some very interesting findings. When a filter is used on the LUT of an image, that filter runs only on an image with the dimensions of 256 by 1. The LUT contains the 256 levels of black and white, which is used by the LUT Applying filter to be applied directly on to the image. The lapsed times of the Invert Filter running directly on the image indicates that it is quicker than when it is applied to the LUT and then ran through the LUT Applying Filter (Figure 28).

The average lapsed time that the Invert Filter took to run directly on the image was 355.8 milliseconds but it took an average lapsed time of 539.7 milliseconds to run the filter upon the LUT and then have the LUT applied to the image (Figure 28). When we analysis the amount of time that the Invert Filter takes to ran per pixel, we see that the lapsed time of the operation performed on each pixel is $\frac{355.8}{460*269}$ which is about .0029 milliseconds. If the Invert Filter takes a lapse time of .0029 milliseconds to run on a pixel then it only took about 256 * .0029 = .7424 milliseconds to run the Invert Filter on the LUT. The LUT Applying Filter must take 539.7 - .7424 = 538.9576 milliseconds, which is about $\frac{538.9576}{460*269}$ = .0044 milliseconds per pixel.

The Color Curves Filter produced an average lapsed time of 3911.3 milliseconds to run on the image (Figure 29). The filter's algorithm takes about $\frac{3911.3}{460*269}$ = .0316 milliseconds to modify a pixel. With that in mind, when that same filter is ran on the LUT of image with far less pixels, the total lapsed time to run the Color Curve Filter will take .0316 * 256 = 8.0896 milliseconds. Now add that to the average lapsed time of the LUT Applying Filter when applied to a 460 by 269 pixel image, 538.9576 milliseconds, the overall lapsed time to run the Color Curves Filter becomes 538.9576 + 8.0896 = 547.0472 milliseconds.

In the case of the Color Curves Filter, it runs about 3911.3 – 547.0472 = 3364.2528 milliseconds longer, when it is applied directly on the image verses using the image's LUT. It is the opposite case with the Invert Filter because it runs faster directly on the image rather than the LUT of the image.

Chapter 6: Conclusions

The first conclusion that was formed from the building of this application was the fact that if CrazZzy FilterZ was developed in Android rather than ActionScript 3 that it would ran a whole lot faster. After such a conclusion is formed, one may be puzzled in the reasoning behind continuing the development of the application in the slower language. The reasoning behind this decision was due to the time restraints that were present during the development of the application. To make use of the pre-populated library and community of Pixel Bender Filters and developers, I would have had to re-develop that functionality in Android rather than use the already developed functionality in Adobe AIR.

After the completion of CrazZzy Filterz I have concluded that by adding the ability to run any given filter upon the LUT of an image and by resizing the images to maintainable sizes without lowering the quality of the image when it is displayed upon the mobile device's screen, has proven that CrazZzy Filterz have achieved its goals that it were given to it at the start.

CrazZzy Filterz set out to become a counterpart mobile application of an application that is currently used on a desktop that requires enormous amount of processing power to execute and run. CrazZzy Filterz lowers the processing power that it requires to an acceptable level where it can succeed on a mobile device and still accomplish it goals. When the application was tested by a small group of end users, the feedback was overall pleasant with no mention of unacceptable wait times to run any of the pre-installed filters or any unacceptable pixelizetion of the chosen image in which to run any of the filters on.

Overall CrazZzy Filterz proves that a mobile device could be used as a platform for applications that were previously thought not suitable such as a full functioning image filtering application.

Chapter 7: Future Work

In future work, I would like to develop this application as a native application written for both of Android OS and Apples iOS. This will present great improvement in computing time when the filter is ran. I believe that I could rewrite this application using the same tested methods that I presented in this thesis and achieve a great improvement on the overall performance of the application. Also by rewriting my application, I will open up the user group to the majority of mobile users and remove the need for the AIR application to be installed on the device.

In Addition to the rewriting of the application, I will add more functionality. An additional future functionality of CrazZzy Filterz will be the addition of layering the filters upon each other. As of now the image can only have one filter applied to it at a time and when the user changes the applied filter to a different one, the previous filter is removed from the image and the new filter is than applied to the image.

Another enhancement that I will add to a future release will allow for the user to select the pixels of the image that they would like the filter to be applied to rather than having the filter applied to the whole image.

In a future release of CrazZzy Filterz, it will allow the users to sync each other's filter libraries. CrazZzy Filterz will discover itself on another's mobile device and when instructed to by both parties will save any filters that the other user has that they do not have already.

A final addiction to CrazZzy Filterz would be to have the decision to either apply the filter directly to the image or the image's LUT left up to CrazZzy Filterz. As of now, the developers are in charge of deciding if it would be better to apply their filter to the LUT or not. I would like to have CrazZzy Filterz test a new filter to found out if it is beneficial to apply the filter to LUT. The test would take place on small sample image where the filter is applied first to the image and then on the LUT. CrazZzy Filterz should then be able to conclude which approach is best for that filter. One potential problem that CrazZzy Filterz would have to look out for is that to use the method of applying the filter to the LUT requires that the filter must to be free of needing any other pixel's information, so there would have to be some build-in fail safes.

References

1. Rogers, Yvonne and Preece, Jennifer. John Wilsey & Sons. (2007) *Interaction Design: beyond human-computer interaction*. 2nd Edition. West Sussex, England
2. Moock, Colin. O'Reilly Media. (2007) *Essential ActionScript 3.0: ActionScript 3.0 Programming Fundamentals*. 1st Edition. Sebastopol, CA
3. Dougherty, Geoff. Cambridge. (2009) Digital *Image Processing for Medical Applications*. United Kingdom
4. Thanksmister, "AS3 Scrolling List for Android and iOS devices" (Oct 14 2010). Retrieved 2011, from http://www.thanksmister.com/index.php/archive/android-as3-scrolling-list/
5. Peter J. Ackla, "MATLAB array manipulation tips and tricks" (Oct 18 2003). Retrieved 2011, from http://home.online.no/~pjacklam/matlab/doc/mtt/doc/mtt.pdf
6. "Adobe AIR 3" (2011). Retrieved 2011, from http://www.adobe.com/products/air/
7. "Pixel Bender basics for Flash" (2011). Retrieved 2011, from http://www.adobe.com/devnet/flash/articles/pixel_bender_basics.html
8. "Pixel Bender Technology Center" (2011). Retrieved 2011, from http://www.adobe.com/devnet/pixelbender.html
9. "Fundamentals of Image Processing" by hany.farid@dartmouth.edu http://www.cs.dartmouth.edu/~farid
10. Anusha Sethuraman, "US Smartphone Statistics – Q1 2011 Overview" (May 6, 2011). Retrieved 2011, from http://www.smartonline.com/mobile-2/us-smartphone-statistics-q1-2011-overview/
11. "Nearest-neighbor interpolation" (Oct 9, 2011). Retrieved 2011, from http://en.wikipedia.org/wiki/Nearest-neighbor_interpolation
12. "Interpolation" (Oct 16, 2011). Retrieved 2011, from http://en.wikipedia.org/wiki/Interpolation
13. "Bilinear interpolation" (AUG 10, 2011). Retrieved 2011, from http://en.wikipedia.org/wiki/Bilinear_interpolation
14. "Bicubic interpolation" (June 19, 2011). Retrieved 2011, from http://en.wikipedia.org/wiki/Bicubic_interpolation
15. "Lookup table" (Sep 21, 2011). Retrieved 2011, from http://en.wikipedia.org/wiki/Lookup_table