

CSE408 Project Phase 2

Members:

- Sebum Lee
- Joseph Milazzo
- Prayag Patel
- Robert Peck
- April Randolph
- Ryan Rockwood

Abstract:

The purpose of this project is to compress an image with different methods and to save this new image in a binary file which can be decoded and displayed.

Keywords:

- Jmagick: Java Api providing access to ImageMagick
- ImageMagick: A library suite of image manipulation functions that can be accessed either from a command line interface or using one of i;
- Colorspace

Pixel - a single point in a raster image, each with there own address and usually composed of three or four component intensities.

Introduction:

The image manipulation library known as ImageMagick is quite a powerful tool by itself. Even more so when paired with Jmagick an API for accessing most of the tools of the ImageMagick library through the Java language. This API allows users to use ImageMagick functions in their own programs.

Description:

QuantizeScheme.java

- Inputs are the number of buckets for each Y, U, & V stream
- The range of values of each stream is divided into evenly spaced buckets
- If the number of buckets is even it uses Midrise formula
 - $stepSize = \frac{|minValue(Type)| + |maxValue(Type)|}{(number\ of\ Buckets)}$
 - Given any Value α in stream type T and new Quantized Value v
 - $v = ceiling(\alpha) - stepSize / 2$
 - $ceiling(X) \equiv$
 - a. $ceil = min$
 - b. $while\ ceil < X : ceil += stepSize$
 - c. $return\ ceil$
- if the number of buckets is odd it uses Midtread formula
 - $stepSize = \frac{|minValue(Type)| + |maxValue(Type)|}{(number\ of\ Buckets)}$
 - Given any Value α in stream type and new Quantized Value v
 - $v = floor(\alpha - stepSize / 2)$
 - $floor(X) \equiv$
 - a. $floor = max$
 - b. $while\ floor > X : floor = floor - stepSize$
 - c. $return\ floor$

PredictiveCodingOdd.java

- Handles Predictive Coding and constructing an image from a YUVSignal.
- Input is a single YUVSignal.
- constructImage takes a YUVSignal and returns a MagickImage.
 - It does this by packing the Y, U, & V bins into an array of bytes and calls constructImage(). constructImage() is a Jmagick function which creates an image row-by-row.
- No Predictive Coding:
 - Sets the YUVSignal's predictiveCodingFlag to NOPC.
- Predictive Coding Option 2
 - Uses function $F_n = floor(f(n-1))$ with error of $E_n = f - F_n$.

- Does this computation for the Y, U, & V bins and stores the new value in the Ynew, Unew, and Vnew arrays in the YUVSignal. The error is stored in the Y-, U-, and V- err arrays respectively.
- Predictive Coding Option 3
 - $$\frac{f(n-1) + f(n-2)}{2}$$

Uses function $F_n = \text{floor}(\frac{f(n-1) + f(n-2)}{2})$ with error of $E_n = f - F_n$.
 - Does this computation for the Y, U, & V bins and stores the new value in the Ynew, Unew, and Vnew arrays in the YUVSignal. The error is stored in the Y-, U-, and V- err arrays respectively.
- Predictive Coding Option 4
 - $$\frac{2 * f(n-1) + f(n-2)}{3}$$

Uses function $F_n = \text{floor}(\frac{2 * f(n-1) + f(n-2)}{3})$ with error of $E_n = f - F_n$.
 - Does this computation for the Y, U, & V bins and stores the new value in the Ynew, Unew, and Vnew arrays in the YUVSignal. The error is stored in the Y-, U-, and V- err arrays respectively.
- Predictive Coding Option 5
 - $$\frac{f(n-1) + 2 * f(n-2)}{3}$$

Uses function $F_n = \text{floor}(\frac{f(n-1) + 2 * f(n-2)}{3})$ with error of $E_n = f - F_n$.
 - Does this computation for the Y, U, & V bins and stores the new value in the Ynew, Unew, and Vnew arrays in the YUVSignal. The error is stored in the Y-, U-, and V- err arrays respectively.
- Predictive Coding Option 6
 - Uses function $F_n = \frac{f(n-1) + f(n-2) + \dots + f(n-10)}{10}$

$\text{floor}(\frac{f(n-1) + f(n-2) + \dots + f(n-10)}{10})$ with error of $E_n = f - F_n$.
 - Does this computation for the Y, U, & V bins and stores the new value in the Ynew, Unew, and Vnew arrays in the YUVSignal. The error is stored in the Y-, U-, and V- err arrays respectively.

EncodingMain.java

- The inputs for all encoding options are the new integer arrays that might have been created from choosing quantization or predictive coding.
- The Run Length Encoding will count the repetitions of a given element if they are in a continuous order in the array. It counts the amount of repetitions and then prints the element before going to the next one. .
- Shannon-Fano.java is a recursive function that takes the array and uses a created frequency hash table to divide the array by two then assigns

the encoding of '1' or '0' depending on the divide until the entire array has been encoded. This is a top-down approach.

- Huffman.java is also a recursive function that takes the three integer arrays and creates a tree from a bottom-up approach by using the two least frequency nodes and then creating a parent, deleting the children, re-sorting and then re-selecting the next two least nodes again. This, in essence, allows for the highest frequencies to have the lowest encoding.

YUVencoding.java

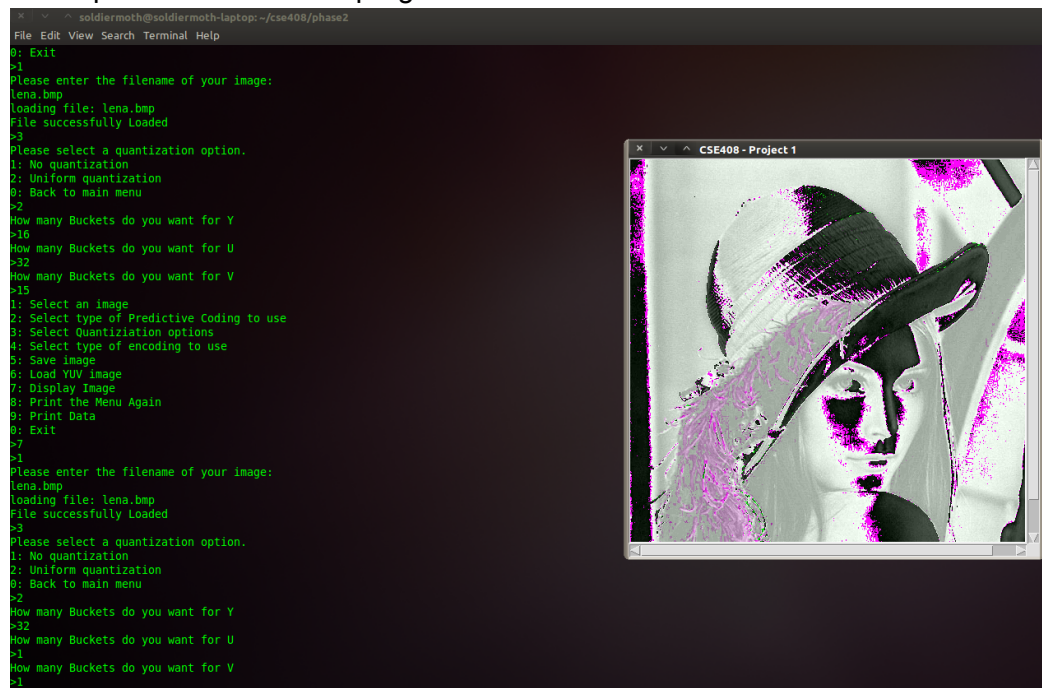
- Has two functions:
 - encodeSignal, which takes a YUVSignal as input and saves it as a binary file via serialization with a YUV extension using the writeObject method.
 - decodeSignal, which prompts the user to enter a file name and decodes the file to a YUVSignal which it returns using the readObject method, after performing any necessary decoding

YUVSignal.java

- A structure that represents the original image and the new image.
- Contains 3 1D arrays (Y, U, & V bins) which hold part of a pixel from original image. Contains 3 1D arrays which hold the image after applying various compression schemes. Contains integer flags to denote the type of schemes used. Also contains a hash table if Huffman or Shannon-Fano coding was used.

Interface Specs:

A sample execution of the program is shown below:



The screenshot shows a Java application window titled "CSE408 - Project 1". The window contains a terminal interface on the left and a grayscale image of a woman's face on the right. The terminal displays the following text:

```
File Edit View Search Terminal Help
0: Exit
>1
Please enter the filename of your image:
lena.bmp
loading file: lena.bmp
File successfully Loaded
>3
Please select a quantization option.
1: No quantization
2: Uniform quantization
0: Back to main menu
>2
How many Buckets do you want for Y
>16
How many Buckets do you want for U
>32
How many Buckets do you want for V
>15
1: Select an image
2: Select type of Predictive Coding to use
3: Select Quantization options
4: Select type of encoding to use
5: Save image
6: Load YUV image
7: Display Image
8: Print the Menu Again
9: Print Data
0: Exit
>7
>1
Please enter the filename of your image:
lena.bmp
loading file: lena.bmp
File successfully Loaded
>3
Please select a quantization option.
1: No quantization
2: Uniform quantization
0: Back to main menu
>2
How many Buckets do you want for Y
>32
How many Buckets do you want for U
>1
How many Buckets do you want for V
>1
```

- Option 1: Select and load an image.
- Option 2: Select type of Predictive Coding to use.
 - Option 1: No Predictive Coding

- Option 2: PC Option 2
- Option 3: PC Option 3
- Option 4: PC Option 4
- Option 5: PC Option 5
- Option 6: PC Option 6
- Option 0: Back to main menu
- Option 3: Select type of Quantization to use.
 - Option 1: No Quantization
 - Option 2: Uniform Quantization
 - Prompts for how many buckets you wish to use for each Y, U, & V signal
 - Option 0: Back to main menu
- Option 4: Select type of encoding to use.
 - Option 1: No encoding
 - Option 2: Rune-length encoding
 - Option 3: Shannon-Fano coding
 - Option 4: Huffman coding
 - Option 0: Back to main menu
- Option 5: Save YUV image.
- Option 6: Load YUV image.
- Option 7: Display currently loaded image.
- Option 8: Print Menu Again.
- Option 9: Print Data.
- Option 0: Exit

System Reqs/Installation/Execution:

- Ubuntu (10.10 Beta)
 - a. Install ImageMagick:
 - version 7:6.6.2.6-1ubuntu1
 - Available in Synaptic Package Manger
 - b. Install Jmagick:
 - version 6.4.0
 - Download source from this link: <http://downloads.jmagick.org/6.4.0/jmagick-6.4.0-src.tar.gz>
 - untar it with 'tar -xzvf jmagick-6.4.0-src.tar.gz'
 - follow instructions in the just downloaded readme file to install
 - c. Obtain project code:
 - Either uncompresss included file with source or check it out using mercurial using the following command:
'hg clone https://cse408.googlecode.com/hg/ cse408'
 - To compile use the following command replacing the curly braced item with paths to the relevant directories/files
'LD_LIBRARY_PATH={Path to folder containing jmagick.so} javac -classpath lib/jmagick.jar:. App.java'
 - To run use following command replacing the curly braced item with paths

to the relevant directories/files

'LD_LIBRARY_PATH={Path to folder containing jmagick.so} java -
classpath lib/jmagick.jar:. App'

- Windows

- a. Install ImageMagick:

- version 6.3.9 Q8

- b. Install Jmagick:

- version 6.4.0
 - Download source from this link: <http://downloads.jmagick.org/6.4.0/jmagick-6.4.0-src.tar.gz> [1]
 - untar it with 'tar -xvzf jmagick-6.4.0-src.tar.gz'
 - follow instructions in the just downloaded readme file to install

- c. Obtain project code in Eclipse:

- Place the project folder in Eclipse workspace.
 - Create a new Java project in Eclipse and give it the same name as the project folder.
 - Add the jmagick.jar file through Add External JARs option. This could be done by going to the Project (cse408) properties -> Java Build Path -> Libraries (tab) -> Add External JARs.
 - Include the jmagick.jar and jmagick.dll files in ImageMagick folder (usually in Program Files) and in Java/bin folders.
 - To run select the Run Option and run as a Java Application.

Conclusions:

By using a suite of compression methods, we can efficiently store images to disk and rebuild the image.

Bibliography:

[1] Jmagick Javadoc [<http>]

Appendix(Roles):

- Sebum Lee
 - Implemented encoding options & decoding
- Joseph Milazzo
 - Create the YUVSignal structure, convert the image into 3 1D Y, U, and V bins, and Predictive Coding options 1 and 3.
- Prayag Patel
 - Predictive Coding options 2, 4, 5 and 6.
- Robert Peck
 - Implemented quantization scheme class
- April Randolph
 - Implemented encoding options & decoding
- Ryan Rockwood
 - Implemented the functions to convert the signal to a binary output file and then convert back. Helped with the color conversions.