The Sound of Color

Jared Charter

Ryan Brinkley

ECE 438

December 15, 2015

Table of Contents

[Abstract 1](#_Toc437891391)

[Introduction 2](#_Toc437891392)

[Background/Theory 3](#_Toc437891393)

[Experimental Methods 5](#_Toc437891394)

[Program Description 6](#_Toc437891395)

[Analysis/Results 9](#_Toc437891396)

[Summary and Conclusions 12](#_Toc437891397)

[Suggestions for Future Work 13](#_Toc437891398)

[References 14](#_Toc437891399)

[Appendix A 14](#_Toc437891400)

# Abstract

This paper contains a description of the ECE 438 Color of Sound final project. The project was derived from a computer engineering senior design project. The project goal was to implement a system to detect and translate color of an object to sound. Many experimental methods were used to attempt this task until a final method was chosen. This method involved cropping a defined area in the middle of the sample image where the object is located. This cropped image is then filtered and segmented to get the majority color present in the object. A sound frequency associated with this color is calculated and matched in a predefined color table containing various shades of colors. A full description and explanation of the implementation of this method can be found in the Program Description below. The success rate of this implementation is 88.5% on objects of similar color to the training objects and 54.2% on objects not contained in the training set.

# Introduction

The Sound of Color is a final project for ECE 438 at Southern Illinois University Edwardsville. The purpose of the project is to be able to detect the color of an object in the center of an image and convert this color to a sound frequency. The reasoning for this functionality is to be used in a senior design project in which a camera is being used to convert color to sound which is then output to a user through a speaker. In order to accomplish this, the color spectrum had to be linearly mapped onto a defined sound range so that the colors directly corresponded to a specific frequency. In order to detect the color, a small area in the center of the original image is cropped. This cropped image is then filtered for noise and segmented into fewer colors. The majority color of the segmented objects in the image is the color which represents the object’s color and is therefore mapped to the sound frequency through a table of various colors and shades of those colors.

The algorithm for this computer vision application was written and developed in the C Language in a Visual Studio 2010 IDE. Some of the functions used were from software called CVIPtools. “CVIPtools is a collection of computer imaging tools providing services to the users at four layers: the C function layer, the COM interface layer, the CVIPImage layer and the Graphical User Interface (GUI)”. The C/C++ Language libraries contained in this software were used for the functions such as the mean filter and segmentation function. By having the GUI of this software readily available, many of the algorithms for the project were able to be tested in the GUI before being implemented in the program. The algorithm was developed using a training set of images which were taken from a Raspberry Pi 2 PiCamera. These images were taken and used because this is the device which will be taking the images on the final senior design project.

# Background/Theory

The idea for this project is based off of a computer engineering senior design project by one of the members in the project group. The senior design project is a pair of glasses which are made to aid the visually impaired called Blind-Aid glasses, or Blind-Aids for short. One of the functionalities of this project is to provide a visually impaired user with a sense of color through the sense of hearing. This is to be done by detecting the color of an object directly in front of the user using a camera placed on the glasses, then outputting that color as a sound frequency back to the user through a speaker located on the glasses. The colors’ sound frequencies (300Hz – 1000Hz) will increase as they do in their wavelengths on the electromagnetic spectrum which ranges from near ultraviolet (~400nm) on up to near infrared (~700nm), which is otherwise known as the visible light spectrum. This way the user, possibly having no other reference to what colors look like, can have a defined scale to match the sound frequencies to. These glasses are made to be useful for both the blind and color blind, and also provide the user with a sense of distance to an object through haptic feedback with vibration motors. There were no previous algorithms for transposing a visible spectrum onto a linear range such as a sound frequency range which fit this application, so the project for the ECE 438 Computer Vision and Image Analysis class became to design and implement an algorithm which could perform this task. The specifications and requirements of this project were therefore defined by the member of the senior design project.

In order to accomplish this task, we knew that we would need a way to detect the color of the object. Since it was desired that it be the color of the object directly in front of the user, or more specifically that the user is looking at, we came up with the theory to only concentrate on a small area in the middle of the image which would represent where the user is looking. As there are still many different colors in an image which appears to be one color, some form of filtering or segmentation would be needed to reduce the amount of colors in the area and find the truest color of the object. At first it was thought that this could be accomplished simply by averaging the colors in the area, but we realized this would include colors which were not desired and would then alter the actual object color. In order to transpose these colors onto a sound frequency range, we knew the colors would need to be mapped to certain frequencies. Unfortunately, visible light in the form of RGB (red, green, and blue bands) does not increase on a linear scale as it increases in wavelength so a method would need to be devised to represent the visible light spectrum as a linear scale. Some different methods thought up to accomplish this included converting the color space to HSV or HSL, creating a color lookup table of defined colors, or deriving a function to represent the increase and decrease of the three band values. The sound frequency associated with the color was not necessary to be played back by audio for the implementation of the algorithm, but this was easily implemented using the Beep() function in the C language.

# Experimental Methods

There were multiple different methods attempted to complete some of the subtasks for this project before reaching a final solution. The first is determining the object in front of the user and the color of that object. We began by experimenting with a variable sized area of the original image in which to detect the majority color within that area. This area would change by the readings given back from the distance sensors on the glasses and would therefore give the width of the object in front of the user. It was found that in most to all cases though, the majority color when this area was segmented with various amounts of colors would end being the background color in the image. We were then able to determine that we could just focus on one small area directly in the middle of the frame and whatever color was most prevalent in that area would be the color which the user is seeing. The type of segmentation method to use was also an experimental factor which was determined by trying out some of the different median segmentation methods such as PCT and median-cut. We knew we wanted to use a median segmentation as this would allow the colors in the image to be reduced and would apply a median filter. By segmenting a set of images with both methods, it was decided that the median-cut algorithm produced better results in regards to the colors than the PCT algorithm. Both algorithms returned the color count of the image as well so being able to retrieve this count was not a factor.

One of the biggest experimental methods was determining how to make a linear visible light spectrum scale. We first had thought we may be able to convert the image into HSL or HSV and find a way to match the hue of the majority color, but it ended up actually making the colors linear more complicated. We then went with a method of segmenting an image of the visible light spectrum into 32 individual colors and then mapping these to the sound frequency range. This was working okay until we started using training images which contained different shades of these colors, as not every shade of every color is represented on the visible light spectrum chart. It was then that we came across an html color table which contained 12 (plus another for black and white) color columns with 9 rows of shades of each of those colors. This table followed the linear wavelength increase of the light spectrum and contained multiple shades, so we decided to enter these into a table divided by each color and associated a frequency with each of the 14 colors (black and white were added at each end of the range).

Another experimental factor which was determined was the mean filter mask size. We experimented with various mask sizes on different images. After comparing some of the sizes when filtering some sample images, it was found that a larger mask size was better as it got rid of most of the noise in the image. A large mask size for this application was acceptable as we were not concerned with keeping solid edges within the image as it was simply being used to represent a color.

# Program Description

The goal of this project is to determine the color in a defined area in the center of an image and to output a sound corresponding to the visible spectrum below. A color lower on the spectrum would output a lower frequency, and a color on the higher end would output a higher frequency. In this project, black is considered the lowest on the color spectrum, and white is the highest.

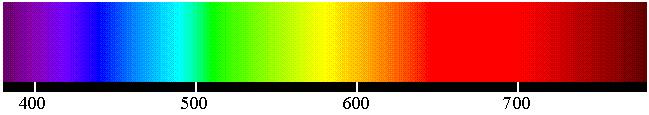


Figure 1: Visible Light Spectrum

Fourteen hues were mapped to frequencies ranging from 300 to 1000 Hz. A color determined to be gray would have a frequency of 0. Each hue was given 9 separate shades ranging from dark to light, as seen in the picture below. The shades were mapped to the RGB color space and placed into a structure for its corresponding hue. The structure also contained that hue’s sound frequency and hue name. These structures were then put into an array that could be iterated upon easily. This structure array allowed for easy access to each color within the program by allowing us to index each item in the structures array to contain its own individual color name, frequency, and structure array of shades which each has its own r, g, and b member. See the appendix for the implementation of these structures and their members.

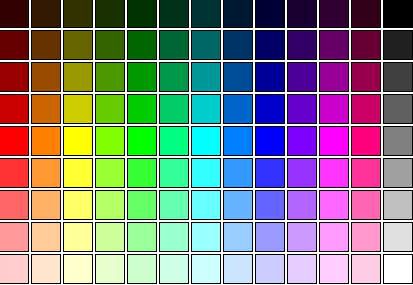


Figure 2: Color table sampled from

First, a 64x64 square was cropped at the center of the image. Pictures with a resolution of 2592x1944 were used. A mean filter with a 9x9 mask was then applied to the image to reduce noise. After this, a median cut was utilized to reduce the total number of colors in the image to 8. The crop, mean\_filter, and median\_cut\_segment functions from CVIPtools were used for these operations.

The next step was to obtain the majority RGB value from the image. The function median\_cut\_segment prints out the most common colors to the console, so the function needed to be modified so that these values could be used in the application. The files median\_cut.c, median\_cut.h, and Colormap.c had to be modified for this step. Rather than modifying these existing files, a custom copy of each of these files was made and then edited to suit the needs of the functions within them.

In the following stage, the closest matching color to the 15 mapped hues was chosen. To do this, each hue’s 9 different shades were compared with the color from the image using the city block distance metric on the three RGB values. The lowest distance found would be the hue chosen.

The final step is to output the frequency associated with the selected hue. Each hue is assigned a frequency from 300 to 1000 Hz with the exception of gray, which is assigned 0. Each hue is linearly spaced by 53.8 Hz. This step is done simply by printing the frequency from the structure associated with that hue.

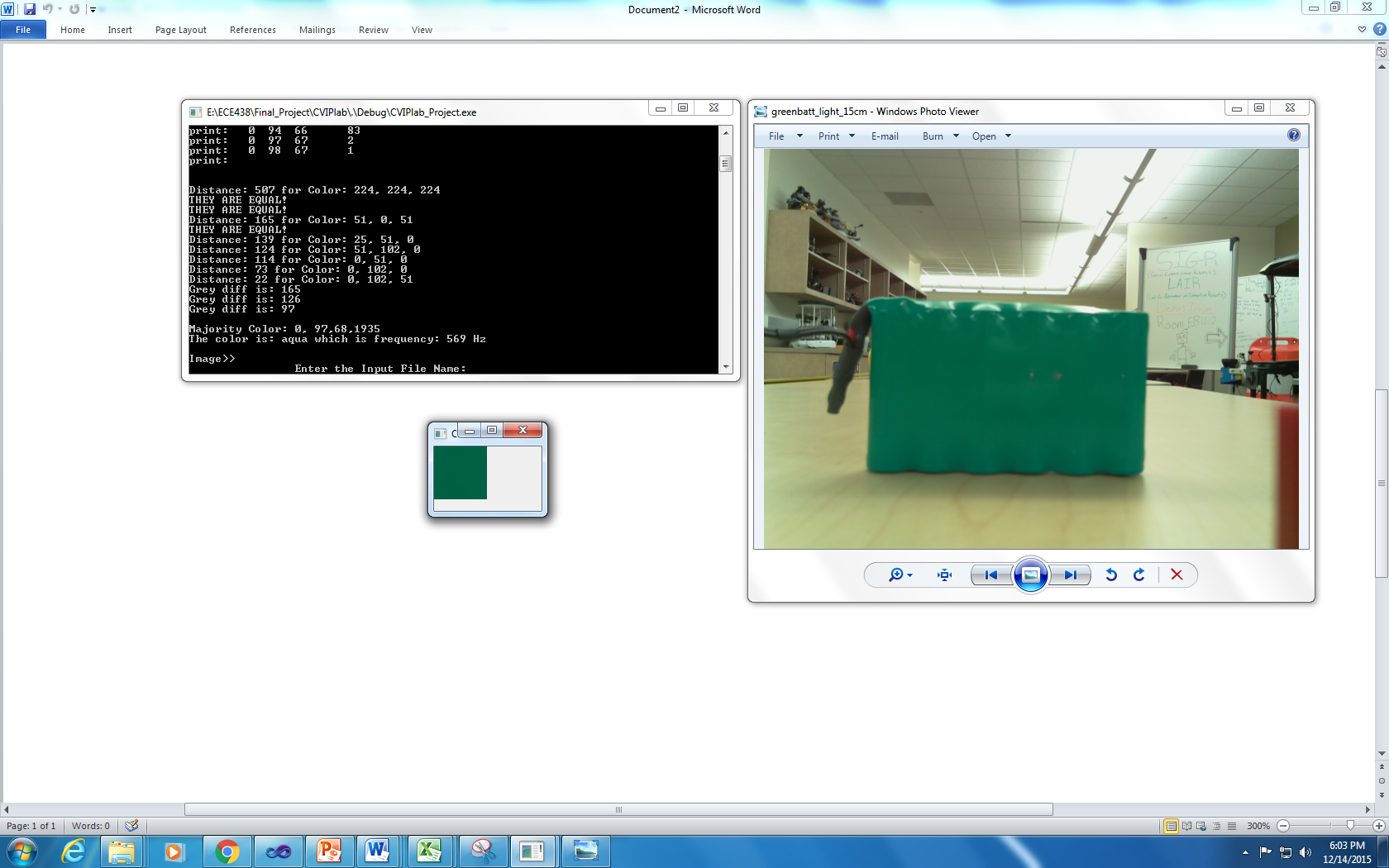
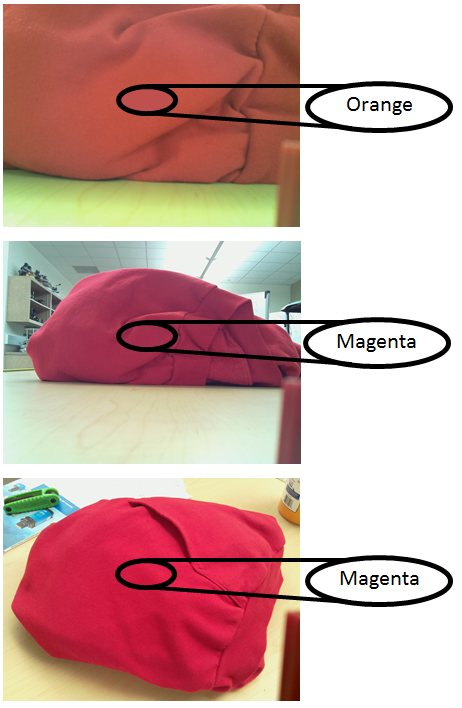
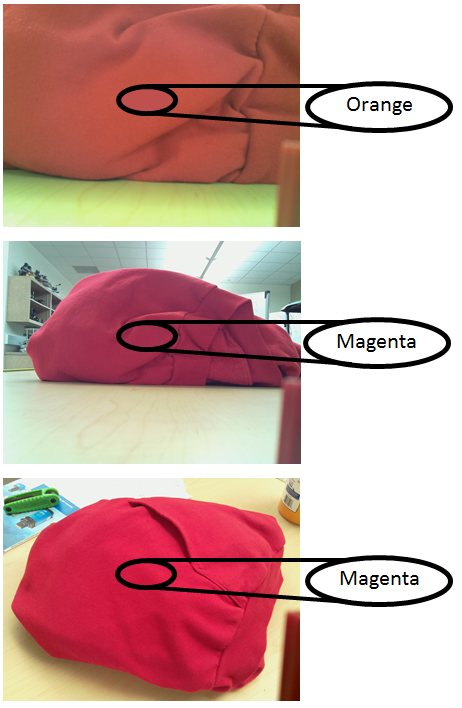


Figure 3: Sample output of the image on the right. The small image is the cropped and segmented area of the image.

# Analysis/Results

Our application was trained on three objects at two distances, 15 centimeters and 45 centimeters away from the lens. The application had trouble initially distinguishing darker hues from gray. This was solved by only choosing gray when the city block distance metric gave a total value less than 30.

These three images show the resulting colors detected for each of the images shown. The object in these images is the redshirt object seen in the tables below and is the same object in each image. The coloration of the object seems to look different in each image, but in only one image was it misclassified as orange instead of magenta.



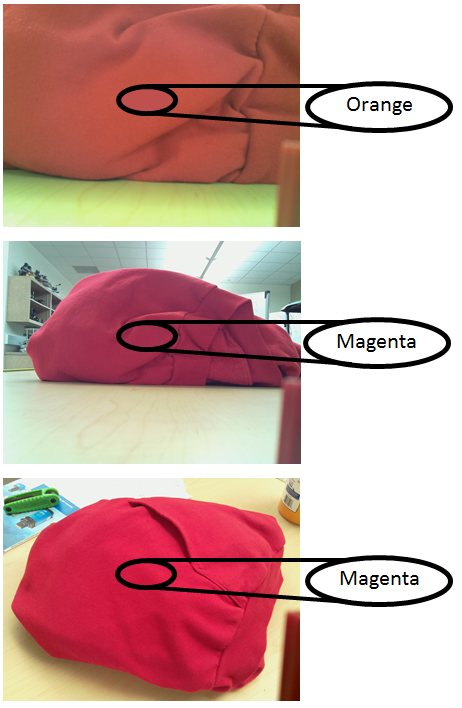


Figure 4: Example of classification of same object in different images

The tables and graph below show the accuracy of our classification of objects of different colors. The first table shows the classification of the test set images of the same three objects used in the training set images. The second table shows the classification of other objects of various colors based off of the color that they appear to be to the naked eye. The chart below shows each of these colors and as their percentages of accuracy in classifying the colors of the objects in the test set images. Colors which were misclassified as being only one color away from the desired color were given an accuracy of 0.5% as the frequency associated with that color is close to the desired frequency and therefore this misclassification would still give the user a correct idea of the color of the object.



Table 1: Test results of test set images of training set objects



Table 2: Test results of test set images of objects not used in training

Figure 5: Correct color classification chart of objects in test images

The three colors used in the training set: magenta, yellow, and aqua, are each at least 75% correct in classification of the test images. Orange, which was untrained, came out to be 100% accurate in the classification of multiple different objects. Black, grey, and white images classified at 50% accuracy and are each very close colors to each other. These colors have been found to be very hard to distinguish as they each are more of a representation of the reflectivity and luminance of an object rather than the hue of the object. Blue colors tend to show up in the camera as very unsaturated and appear to be more of a grey color; therefore our blue object was detected as grey. This same sort of saturation issue caused the pink object in the test set to be detected as a shade of red rather than a shade of pink which it appears to be to the naked eye.

# Summary and Conclusions

The goal of the Color of Sound project was to determine the color in a defined area in the center of an image and output an associated sound frequency to the user. A color on the lower end of the spectrum (shorter wavelengths) will output a lower frequency and colors located on the upper end (longer wavelengths) will output higher frequencies. The purpose of this project was for a computer engineering senior design project, and all of the requirements for the project were met with our implementation. These requirements included detecting the color of an object directly in the middle of the image frame, transposing the visible light spectrum onto a defined 300-700Hz sound frequency range, and outputting the corresponding sound frequency of the object color to the user. The implementation of this project was accomplished with the help of existing libraries from CVIPtools and standard C language libraries. By use of a mean filter and a median-cut segmentation algorithm, the color of the object was able to be detected and extracted. The color table structure made it easy to compare the object color to and match it with the correct color using a city block distance metric. Output of the frequency both by text and audio to the console was added to make the output checkable and easier to understand.

The resulting program’s success rate of 88.5% for the object colors trained with and 54.2% for non-trained object colors is a good accuracy percentage for the purpose of the project. To improve on the other colors which did not classify as well as the trained images, the program could be further modified and trained with a new training set of the desired color of objects. Overall, the success of this implementation met the requirements and provides a solid algorithm for the end goal of the senior design project. The Color of Sound project can officially and successfully convert color to sound.

# Suggestions for Future Work

To improve on this project, the detection and classification of other colors than the colors used in the training set could be enhanced. This could be done either by manually adding in other shades of colors to the color shade arrays, or discovering patterns in which thresholds could be put in place to try and filter out the misclassifications. The black and white colors are currently not easily detected because it is rare to get an image with a dark enough black or light enough white color to not classify it as a different color or a grey shade. The grey tends to take precedence in a lot of objects, especially those which are in lower lit conditions. Perhaps some form of detecting a lightness value of grey saturated colors would allow for more specific color classification or at least better black and white classification. If desired, the program could potentially be made to detect the entire object which the user is looking at (the object in the image frame) and determine the color based off of the entire image, but this may produce undesired results if the wrong object is being detected rather than the one that the user believes he or she is focusing on. Any other improvements beyond these would include code optimization and clean up to improve on the runtime of each detection.

# References

Umbaugh, Scott E., Scott E.*Digital Image Processing And Analysis: Human And Computer Vision*

*Applications With CVIPtools*. Boca Raton, FL : CRC Press, 2011. Print

# Appendix A

CVIPlab.c

**E:\ECE438\finalprogram.c Monday, December 14, 2015 9:07 PM**

/\* =========================================================================

\* Created by: Jared Charter and Ryan Brinkley

\* ECE 438 Final Project: The Sound of Color

\* Last modified: 12/14/2015

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\*\* include header files

\*/

#include "CVIPtoolkit.h"

#include "CVIPconvert.h"

#include "CVIPdef.h"

#include "CVIPimage.h"

#include "CVIPlab.h"

#include "CVIPgeometry.h"

#include "fuzzyc.h"

#include "hist\_thresh.h"

#include <CVIPspfltr.h>

//#include "median\_cut.h" //made own median\_cut\_custom.c file

#include <stdio.h>

#include <string.h>

#include <Windows.h>

#define SPEC\_COLORS 14;

#define COLOR\_SHADES 9;

//Image object for input

Image **\***input**();**

/\*Structure for each color

Members:

-----------

Color name

Color shades (array of color structs)

Color frequency \*/

struct Spectrum**{**

char**\*** name**;**

Color color**[**9**];**

int freq**;**

**};**

// start main funct

void main\_cviplab**(){**

IMAGE\_FORMAT format**;** /\* the input image format \*/

Image **\***cvipImage**;** /\* pointer to the CVIP Image structure \*/

Image **\***cvipImageOut**;** /\* pointer to the CVIP Image structure \*/

char **\***outputfile**;** /\* output file name \*/

CVIP\_BOOLEAN done **=** CVIP\_NO**;**

//Image middle to crop from

int cropRowOffset **=** 980**;**

int cropColOffset **=** 1264**;**

int cropRows **=** 64**;**

int cropCols **=** 64**;**

int i**,**j**,**g**,**b**;** //iterators

//Initilize differences(distances)

int minDiff **=** 1000**;**

int minDiffIndex **=** 0**;**

//Default background for median\_cut

Color bg **= {**0**,** 0**,** 0**};**

//Pointer for storing object color array

int **\***maxColors**;**

//array for storing calculated differences

int diff**[**14**][**9**];**

//array for storing grey differences

int greyDiff**[**3**];**

int isgrey **=** 0**;**//flag for grey color

//Structure of all 14 colors

struct Spectrum spectrum **[**14**] = {**

**{**"white"**,**

**{{** 224 **,** 224 **,** 224 **},**

**{** 224 **,** 224 **,** 224 **},**

**{** 224 **,** 224 **,** 224 **},**

**{** 255 **,** 255 **,** 255 **},**

**{** 255 **,** 255 **,** 255 **},**

**{** 255 **,** 255 **,** 255 **},**

**{** 255 **,** 255 **,** 255 **},**

**{** 255 **,** 255 **,** 255 **},**

**{** 255 **,** 255 **,** 255 **}},** 1000**},**

**{**"pink"**,**

**{{** 51 **,** 0 **,** 51 **},**

**{** 102 **,** 0 **,** 102 **},**

**{** 153 **,** 0 **,** 153 **},**

**{** 204 **,** 0 **,** 204 **},**

**{** 255 **,** 0 **,** 255 **},**

**{** 255 **,** 51 **,** 255 **},**

**{** 255 **,** 102 **,** 255 **},**

**{** 255 **,** 153 **,** 255 **},**

**{** 255 **,** 204 **,** 255 **}},** 946**},**

**{**"magenta"**,**

**{{** 51 **,** 0 **,** 25 **},**

**{** 102 **,** 0 **,** 51 **},**

**{** 153 **,** 0 **,** 76 **},**

**{** 204 **,** 0 **,** 102 **},**

**{** 255 **,** 0 **,** 127 **},**

**{** 255 **,** 51 **,** 153 **},**

**{** 255 **,** 102 **,** 178 **},**

**{** 255 **,** 153 **,** 204 **},**

**{** 255 **,** 204 **,** 229 **}},** 892**},**

**{**"red"**,**

**{{** 51 **,** 0 **,** 0 **},**

**{** 102 **,** 0 **,** 0 **},**

**{** 153 **,** 0 **,** 0 **},**

**{** 204 **,** 0 **,** 0 **},**

**{** 255 **,** 0 **,** 0 **},**

**{** 255 **,** 51 **,** 51 **},**

**{** 255 **,** 102 **,** 102 **},**

**{** 255 **,** 153 **,** 153 **},**

**{** 255 **,** 204 **,** 204 **}},** 838**},**

**{**"orange"**,**

**{{** 51 **,** 25 **,** 0 **},**

**{** 102 **,** 51 **,** 0 **},**

**{** 153 **,** 76 **,** 0 **},**

**{** 204 **,** 102 **,** 0 **},**

**{** 255 **,** 128 **,** 0 **},**

**{** 255 **,** 153 **,** 51 **},**

**{** 255 **,** 178 **,** 102 **},**

**{** 255 **,** 204 **,** 153 **},**

**{** 255 **,** 229 **,** 204 **}},** 785**},**

**{**"yellow"**,**

**{{** 51 **,** 51 **,** 0 **},**

**{** 102 **,** 102 **,** 0 **},**

**{** 153 **,** 153 **,** 0 **},**

**{** 204 **,** 204 **,** 0 **},**

**{** 255 **,** 255 **,** 0 **},**

**{** 255 **,** 255 **,** 51 **},**

**{** 255 **,** 255 **,** 102 **},**

**{** 255 **,** 255 **,** 153 **},**

**{** 255 **,** 255 **,** 204 **}},** 731**},**

**{**"lime"**,**

**{{** 25 **,** 51 **,** 0 **},**

**{** 51 **,** 102 **,** 0 **},**

**{** 76 **,** 153 **,** 0 **},**

**{** 102 **,** 204 **,** 0 **},**

**{** 128 **,** 255 **,** 0 **},**

**{** 153 **,** 255 **,** 51 **},**

**{** 178 **,** 255 **,** 102 **},**

**{** 204 **,** 255 **,** 153 **},**

**{** 229 **,** 255 **,** 204 **}},** 677**},**

**{**"green"**,**

**{{** 0 **,** 51 **,** 0 **},**

**{** 0 **,** 102 **,** 0 **},**

-3-

**{** 0 **,** 153 **,** 0 **},**

**{** 0 **,** 204 **,** 0 **},**

**{** 0 **,** 255 **,** 0 **},**

**{** 51 **,** 255 **,** 51 **},**

**{** 102 **,** 255 **,** 102 **},**

**{** 204 **,** 255 **,** 153 **},**

**{** 229 **,** 255 **,** 204 **}},** 623**},**

**{**"aqua"**,**

**{{** 0 **,** 51 **,** 25 **},**

**{** 0 **,** 102 **,** 51 **},**

**{** 0 **,** 153 **,** 76 **},**

**{** 0 **,** 204 **,** 102 **},**

**{** 0 **,** 255 **,** 128 **},**

**{** 51 **,** 255 **,** 153 **},**

**{** 102 **,** 255 **,** 178 **},**

**{** 153 **,** 255 **,** 204 **},**

**{** 204 **,** 255 **,** 229 **}},** 569**},**

**{**"lightblue"**,**

**{{** 0 **,** 51 **,** 51 **},**

**{** 0 **,** 102 **,** 102 **},**

**{** 0 **,** 153 **,** 153 **},**

**{** 0 **,** 204 **,** 204 **},**

**{** 0 **,** 255 **,** 255 **},**

**{** 51 **,** 255 **,** 255 **},**

**{** 102 **,** 255 **,** 255 **},**

**{** 153 **,** 255 **,** 255 **},**

**{** 204 **,** 255 **,** 255 **}},** 515**},**

**{**"blue"**,**

**{{** 0 **,** 25 **,** 51 **},**

**{** 0 **,** 51 **,** 102 **},**

**{** 0 **,** 76 **,** 153 **},**

**{** 0 **,** 102 **,** 204 **},**

**{** 0 **,** 128 **,** 255 **},**

**{** 51 **,** 153 **,** 255 **},**

**{** 102 **,** 178 **,** 255 **},**

**{** 153 **,** 204 **,** 255 **},**

**{** 204 **,** 229 **,** 255 **}},** 462**},**

**{**"deepblue"**,**

**{{** 0 **,** 0 **,** 51 **},**

**{** 0 **,** 0 **,** 102 **},**

**{** 0 **,** 0 **,** 153 **},**

**{** 0 **,** 0 **,** 204 **},**

**{** 0 **,** 0 **,** 255 **},**

**{** 51 **,** 51 **,** 255 **},**

**{** 102 **,** 102 **,** 255 **},**

**{** 153 **,** 153 **,** 255 **},**

**{** 204 **,** 204 **,** 255 **}},** 408**},**

**{**"purple"**,**

**{{** 25 **,** 0 **,** 51 **},**

**{** 51 **,** 0 **,** 102 **},**

**{** 76 **,** 0 **,** 153 **},**

**{** 102 **,** 0 **,** 204 **},**

**{** 127 **,** 0 **,** 255 **},**

**{** 153 **,** 51 **,** 255 **},**

**{** 178 **,** 102 **,** 255 **},**

**{** 204 **,** 153 **,** 255 **},**

**{** 229 **,** 204 **,** 255 **}},** 354**},**

**{**"black"**,**

**{{** 0 **,** 0 **,** 0 **},**

**{** 32 **,** 32 **,** 32 **},**

**{** 32 **,** 32 **,** 32 **},**

**{** 32 **,** 32 **,** 32 **},**

**{** 32 **,** 32 **,** 32 **},**

**{** 32 **,** 32 **,** 32 **},**

**{** 32 **,** 32 **,** 32 **},**

**{** 32 **,** 32 **,** 32 **},**

**{** 32 **,** 32 **,** 32 **}},** 300**}};**

**while(!**done**) {**

print\_CVIP**(**"\n\nImage>>"**);**

/\*Get the input image \*/

cvipImage **=** input**();**

**if(**cvipImage **== NULL)**

**{**

error\_CVIP**(**"main"**,** "could not read input image"**);**

**break;**

**}**

//for dynamic image size

//cropRowOffset = ((getNoOfRows\_Image(cvipImage))/2)-32;

//cropColOffset = ((getNoOfCols\_Image(cvipImage))/2)-32;

cvipImage **=** crop**(**cvipImage**,**cropRowOffset**,**cropColOffset**,**cropRows**,**cropCols**);** //crop

64x64 image from center

cvipImage **=** mean\_filter**(**cvipImage**,** 9**);** //apply 9x9 mask mean filter

cvipImage **=** median\_cut\_segment**(**cvipImage**,**8**,**CVIP\_NO**,** bg**);** //segment into 8 colors

print\_CVIP**(**"\n"**);**

//retreives the RGB values and count of this color output from the median\_cut

segmentation

maxColors **=** get\_MaxColorValsPlease**();**

/\*This loop calculates the distance to each color and stores it in the diff[][] array.

It first finds the distance to each color's shade using a city-block distance

metric, then

it checks to see if any of these distances are the new smallest distance. If so,

the distance value and

color it is contained in is stored as the minDiff and minDiffIndex. THEY ARE EQUAL!

is printed when a

distance of the same value as the current minDiff is found. Once completed theminDiff will contain the

value of the minimum distance and the minDiffIndex will be the index of the

spectrum structre array which

has the color name and frequency. \*/

minDiff **=** 1000**;**

**for(**i**=**0**;**i**<**14**;**i**++){**

**for(**j**=**0**;**j**<**9**;**j**++){**

diff**[**i**][**j**] =** abs**(**maxColors**[**0**] -** spectrum**[**i**].**color**[**j**].**r**) +** abs**(**maxColors**[**1**] -**

spectrum**[**i**].**color**[**j**].**g**) +** abs**(**maxColors**[**2**] -** spectrum**[**i**].**color**[**j**].**b**);**

**if (**diff**[**i**][**j**] <** minDiff**){**

minDiff **=** diff**[**i**][**j**];**

minDiffIndex **=** i**;**

print\_CVIP**(**"\nDistance: %d for Color: %d, %d, %d"**,** minDiff**,**spectrum**[**i**].**color

**[**j**].**r**,**

spectrum**[**i**].**color**[**j**].**g**,** spectrum**[**i**].**color**[**j**].**b**);**

**}**

**else if(**diff**[**i**][**j**] ==** minDiff**)**

print\_CVIP**(**"\nTHEY ARE EQUAL!"**);**

//For debugging:

//print\_CVIP("\nDistance: %d for Color: %d %d %d", diff[i][j],

spectrum**[**i**].**color**[**j**].**r**,** spectrum**[**i**].**color**[**j**].**g**,** spectrum**[**i**].**color**[**j**].**b**);**

**}**

**}**//endfor

//Each grey distance is calculated individually and stored in the greyDiff array. The

grey RGB values check against

// are the repititions of each of the objects RGB values. For example, (54, 54, 54).

isgrey **=** 0**;**

greyDiff**[**0**] =** abs**(**maxColors**[**0**] -** maxColors**[**0**]) +** abs**(**maxColors**[**1**] -** maxColors**[**0**]) +** abs**(**

maxColors**[**2**] -** maxColors**[**0**]);**

greyDiff**[**1**] =** abs**(**maxColors**[**0**] -** maxColors**[**1**]) +** abs**(**maxColors**[**1**] -** maxColors**[**1**]) +** abs**(**

maxColors**[**2**] -** maxColors**[**1**]);**

greyDiff**[**2**] =** abs**(**maxColors**[**0**] -** maxColors**[**2**]) +** abs**(**maxColors**[**1**] -** maxColors**[**2**]) +** abs**(**

maxColors**[**2**] -** maxColors**[**2**]);**

//This loop checks if the grey values are closer than the current minDiff and if the

color is closer to grey than the

// current selected color.

**for (**g **=** 0**;** g**<**3**;**g**++){**

print\_CVIP**(**"\nGrey diff is: %d"**,** greyDiff**[**g**]);**

**if ((**greyDiff**[**g**] <=** 30**) && (**greyDiff**[**g**] <** minDiff**)){**

isgrey **=** 1**;**

**}**//endif

**}**//endfor

//Print the color which is selected and its frequency. Or print if grey (no freq)

print\_CVIP**(**"\n\nMajority Color: %d, %d,%d,%d"**,** maxColors**[**0**],**maxColors**[**1**],** maxColors**[**2**],**

maxColors**[**3**]);**

**if(**isgrey **==** 0**){**

print\_CVIP**(**"\nThe color is: %s which is frequency: %d Hz"**,** spectrum**[**minDiffIndex**].**

name**,**spectrum**[**minDiffIndex**].**freq**);**

Beep**(**spectrum**[**minDiffIndex**].**freq**,**2000**);**

**}**

**else if (**isgrey **==** 1**){**print\_CVIP**(**"\nThe color is: grey"**);}**

//play all the color sounds

/\*for(b = 0; b < 14; b++){

printf("\n %s", spectrum[b].name);

Beep(spectrum[b].freq,1500);

}\*/

//display the resultant image

view\_Image**(**cvipImage**,**"Cropped\_Area"**);**

//delete\_Image (cvipImage);

**}**//endwhile

**}**

/\*

\*\*end of the function main

/\*

\*\* The following function reads in the image file specified by the user,

\*\* stores the data and other image info. in a CVIPtools Image structure,

\*\* and displays the image.

\*/

Image**\*** input**(){**

char **\***inputfile**;**

Image **\***cvipImage**;**

/\*

\*\* get the name of the file and stores it in the string 'inputfile '

\*/

print\_CVIP**(**"\n\t\tEnter the Input File Name: "**);**

inputfile **=** getString\_CVIP**();**

/\*

\*\* creates the CVIPtools Image structure from the input file

\*/

cvipImage **=** read\_Image**(**inputfile**,** 1**);**

**if(**cvipImage **== NULL) {**

error\_CVIP**(**"init\_Image"**,** "could not read image file"**);**

free**(**inputfile**);**

**return NULL;**

**}**

/\*

\*\* display the source image

\*/

//view\_Image(cvipImage,inputfile);

/\*

\*\*IMPORTANT: free the dynamic allocated memory when it is not needed

\*/

free**(**inputfile**);**

**return** cvipImage**;**

**}**

**median\_cut\_custom.c**

* Function modified:

Image \*

median\_cut\_segment( Image \*imgP, int newcolors, CVIP\_BOOLEAN is\_bg, Color bg)

{

byte \*\*cvecP, mask;

register byte \*rP, \*gP, \*bP;

unsigned imgsize, no\_pixels = 0;

int rows, cols;

register int i;

register int ind;

Color pixel;

ColorMap colormap;

ColorHistogram \*chP, \*newchP;

ColorHashTable \*chtP;

const char \*fn = "median\_cut\_segment";

chP = new\_ColorHist();

mask = MAXPIXVAL;

rows = getNoOfRows\_Image(imgP);

cols = getNoOfCols\_Image(imgP);

imgsize = rows\*cols;

cvecP = (byte \*\*) malloc(3\*sizeof(byte \*));

getBandVector\_Image(imgP, cvecP);

/\*

\*\* Step 2: attempt to make a histogram of the colors, unclustered.

\*\* If at first we don't succeed, lower maxval to increase color

\*\* coherence and try again. This will eventually terminate, with

\*\* maxval at worst 15, since 32^3 is approximately MAXCOLORS.

\*/

for(;;) {

msg\_CVIP(fn, "making histogram...\n" );

compute\_ColorHist(chP, imgP, MAXCOLORS);

if (chP->histogram != NULL)

break;

mask <<= 1;

msg\_CVIP(fn, "too many colors!\n" );

msg\_CVIP(fn, "reducing pixel sample range to [%u - %u] to improve clustering...\n", ((byte) ~mask)+1, mask);

rP = cvecP[0];

gP = cvecP[1];

bP = cvecP[2];

for(i = 0; i < imgsize; i++, rP++, gP++, bP++ ) {

\*rP &= mask;

\*gP &= mask;

\*bP &= mask;

}

}

msg\_CVIP(fn, "%u colors found\n", chP->no\_of\_colors );

if(is\_bg)

dropColor\_ColorHist(chP, bg, &no\_pixels);

/\*

\*\* Step 3: apply median-cut to histogram, making the new colormap.

\*/

msg\_CVIP(fn, "choosing %d colors...\n", newcolors );

newchP = mediancut( chP, imgsize - no\_pixels, newcolors );

colormap = newchP->histogram;

delete\_ColorHist(chP);

/\*

\*\* Step 4: map the colors in the image to their closest match in the

\*\* new colormap, and write 'em out.

\*/

msg\_CVIP(fn, "mapping image to new colors...\n" );

chtP = new\_ColorHT();

rP = cvecP[0];

gP = cvecP[1];

bP = cvecP[2];

for (i = 0; i < imgsize; i++, rP++, gP++, bP++ ) {

/\*

\* Check hash table to see if what we have already matched this

\* color.

\*/

assign\_Color(pixel,\*rP,\*gP,\*bP);

if(is\_bg && equal\_Color(pixel,bg)) continue;

ind = lookUpColor\_ColorHT( chtP, pixel );

if ( ind == -1 ) { /\* search colormap for closest match. \*/

register int j, r, g, b;

register long dist, newdist;

dist = 2000000000;

for (j = 0; j < newcolors; j++) {

r = getRed\_Color( colormap[j].pixel );

g = getGrn\_Color( colormap[j].pixel );

b = getBlu\_Color( colormap[j].pixel );

newdist = ( \*rP - r ) \* ( \*rP - r ) +

( \*gP - g ) \* ( \*gP - g ) +

( \*bP - b ) \* ( \*bP - b );

if ( newdist < dist ) {

ind = j;

dist = newdist;

}

}

addObject\_ColorHT(chtP, pixel, ind);

}

\*rP = getRed\_Color(colormap[ind].pixel);

\*gP = getGrn\_Color(colormap[ind].pixel);

\*bP = getBlu\_Color(colormap[ind].pixel);

colormap[ind].value++;

}

print\_CVIP("IN MEDIAN SEGMENT! WOO!");

//print\_ColorHist(newchP);

colorTemp = print\_ColorHist\_MaxColor(newchP);

colorArray[0] = colorTemp[0];

colorArray[1] = colorTemp[1];

colorArray[2] = colorTemp[2];

colorArray[3] = colorTemp[3];

delete\_ColorHT(chtP);

delete\_ColorHist(newchP);

//print\_CVIP("\nHERE THEY STAY: %d,%d,%d,%d", colorArray[0],colorArray[1],colorArray[2],colorArray[3]);

free(cvecP);

return imgP;

}

* Function added:

int\* get\_MaxColorValsPlease(){

//print\_CVIP("\nHERE THEY STAY STILL PLEASE: %d,%d,%d,%d", colorArray[0],colorArray[1],colorArray[2],colorArray[3]);

return colorArray;

}

**colormap\_custom.c**

* Function added:

int\* print\_ColorHist\_MaxColor( ColorHistogram \*chP )

{

register int i;

const char \*fn = "print";

int redVal,greenVal,blueVal,totalVal;

int valArray[4];

if(!chP->histogram) return;

/\* Sort by count. \*/

qsort( chP->histogram, chP->no\_of\_colors, sizeof(ColorHistObject), valueCompare );

msg\_CVIP(fn, "\nTOTAL NUMBER OF UNIQUE COLORS = %u.\n\n",chP->no\_of\_colors);

msg\_CVIP(fn, "RED GRN BLU\tCOUNT\n" );

msg\_CVIP(fn, "--- --- ---\t-----\n" );

valArray[0] = getRed\_Color(chP->histogram[0].pixel);

valArray[1] = getGrn\_Color(chP->histogram[0].pixel);

valArray[2] = getBlu\_Color(chP->histogram[0].pixel);

valArray[3] = chP->histogram[0].value;

for ( i = 0; i < chP->no\_of\_colors; i++ )

msg\_CVIP(fn, "%3d %3d %3d\t%d\n", getRed\_Color(chP->histogram[i].pixel), getGrn\_Color(chP->histogram[i].pixel), getBlu\_Color(chP->histogram[i].pixel), chP->histogram[i].value );

msg\_CVIP(fn,"\n" );

//print\_CVIP("HERE THEY START: %d,%d,%d,%d", valArray[0],valArray[1],valArray[2],valArray[3]);

return valArray;

}

**CVIPlab.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ======================================================================

\* Computer Vision/Image Processing Tool Project - Dr. Scott Umbaugh SIUE

\* ======================================================================

\*

\* File Name: CVIPlab.h

\* Description: contains function prototypes, constants and types

\* defined in CVIPlab functions.

\* Related Files: \*.c

\* Initial Coding Date: Tue Apr 27 21:49:42 CDT 1993

\* Portability: Standard (ANSI) C

\* Credit(s): Gregory Hance and Jared Charter and Ryan Brinkley

\* Southern Illinois University @ Edwardsville

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef CVIP\_LAB\_\_H

#define CVIP\_LAB\_\_H

extern Image\* median\_cut\_segment( Image \*imgP, int newcolors, CVIP\_BOOLEAN is\_bg, Color bg);

extern int\* get\_MaxColorValsPlease();

#endif

**median\_cut.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ======================================================================

\* Computer Vision/Image Processing Tool Project - Dr. Scott Umbaugh SIUE

\* ======================================================================

\* File Name: median\_cut.h

\* Description:

\* Related Files:

\* Initial Coding Date: Thu Oct 7 15:00:03 CDT 1993

\* Portability: Standard (ANSI) C

\* References:

\* Author(s): Gregory Hance and Jared Charter and Ryan Brinkley

\* Southern Illinois University @ Edwardsville

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#if !defined(CVIP\_MEDIAN\_CUT\_\_INCLUDED\_)

#define CVIP\_MEDIAN\_CUT\_\_INCLUDED\_

#ifdef \_\_cplusplus

extern "C"{

#endif

extern Image \*median\_cut\_segment(Image \*imgP, int newcolors,

CVIP\_BOOLEAN is\_bg, Color bg);

extern Image \*median\_cut\_setup(Image \*imgP);

extern int\* get\_MaxColorValsPlease(void);

#ifdef \_\_cplusplus

}

#endif

#endif