Assignment 5 – Color Blindness Simulator

Noah Meisel

CSE 13S – Spring 2023

Purpose

The purpose of this program is to alter standard .bmp formatted images into images simulating deuteranopia. Deuteranopia is a form of colorblindness in which the affected individual lacks the ability to take in green light. To simulate this condition, the program will read in a .bmp file, and through a series of mathematical functions provided in the asgn5 pdf[1], write the adjusted image back into a new .bmp file. Reading the .bmp file itself is done by reading and writing bytes from a buffer file.

How to Use the Program

To use this program the user will need a .bmp file to pass as an argument to the program.

To compile the program, run **make** in the terminal to compile and link all the provided .c files into one executable file "io".

The executable file "io" is run in conjunction with the following three command-line options.

```
-i filename: sets a .bmp file for the program to read from.
-o filename: sets a file for the program to write to.
-h: Prints a help message with these command-line option descriptions to the console
```

The "io" executable can also be run with the shell script **./cb.sh**, which was provided in the asgn5 PDF [1].

Program Design

This program is split across io.c, colorb.c and bmp.c Io.c Implements the main Input/Output functions for this program. The first set of functions are designed to implement a buffer structure to aid in the reading of writing of data from the .bmp files. The rest of the functions in io.c allow for the reading and writing of data into and out of the buffer in increments of 8, 16, and 32 bytes. Bmp.c contains the functions to read and write bmp files as well as convert them into an image that simulates deuteranopia. Colorb.c contains the main() for this program. It takes in the command line options specified above and utilizies the functions in io.c and bmp.c to read and write a .bmp file. The sequence of functions called is described as follows

```
Checks if either file is null.
Creates a buffer file to read the input file to
Creates a bmp structure using that buffer
Closes the buffer
Reduces the bmp pallette.
Writes another buffer file to write to
Writes the new bmp to that buffer before closing the buffer and storing it's contents in the outfile.
```

Data Structures

Buffer

The Buffer data structure control the input and output of our files. The actual buffer is an array of size BUFFER_SIZE, which is defined as 4096 bytes. The buffer also has a file descriptor fd, an offset vlaue that shows the next valid byte to read to, or for writing the next empty location to write. There is also an int value "num_remaining" that keeps track of the number of unused bytes left in the buffer.

BMP

The BMP structure keeps track of the height, width, and the color values of each pixel in .BMP file that was read into a buffer. The structure utilizes another strucutre, Color, that stores the red, green, blue values for each pixel.

Algorithms

Function Descriptions

Buffer *read_open(const char *filename) Takes in a filename and returns a pointer to a new read-buffer. Later in the program this buffer will be used to read bytes from the file.

void read_close(Buffer **pbuf Frees the memory occupied by a specified read-buffer file.

Buffer *write_open(const char *filename) Takes in a filename and returns a pointer to a new write-buffer. Later in the program this buffer will be used to write bytes into a file.

void write_close(Buffer **pbuf Frees the memory occupied by a specified write-buffer file.

bool read_uint8(Buffer *buf,uint8_t *x) Takes in a Buffer file to read from and a pointer to an empty byte x to store the data to be read. Checks if the buffer has at least one byte, and then stores that value in the empty byte x and returns true.

bool read_uint16(Buffer *buf,uint16_t *x) Takes in a buffer file to read from and a pointer to an empty half-word x to store the data to be read. Calls read_uint8t() twice, and combines the read value using a bitwise left-shift by 8 and a bitwise or and then stores those values in x. Returns true.

bool read_uint16(Buffer *buf, uint16_t *x) Takes in a buffer file to read from and a pointer to an empty word x to store the data to be read. Calls read_uint16t() twice, and combines the read value using a bitwise left-shift by 16 and a bitwise or and then stores those values in x. Returns true.

void write_uint8(Buffer *buf, uint8_t x) Takes in a buffer file to write to, and an empty byte x. Checks if the buffer is full, and emptys it if it is, and then writes the value x into the buffer.

void write_uint16(Buffer *buf, uint16_t x) Takes in a buffer file to write to, and an empty half-word x. Calls write_uint8() twice, with x and then x bitwise-right-shifted by 8.

void write_uint32(Buffer *buf, uint32_t x) Takes in a buffer file to write to, and an empty word x. Calls write_uint16() twice, with x and then x bitwise-right-shifted by 16.

void bmp_write(const BMP *bmp, Buffer *buf) Takes in a BMP structure and a buffer file to write to. Writes a BMP file into the buffer. The exact pseudocode for this process can be found in the asgn5 PDF[1]

BMP *bmp_create(Buffer *buf) Takes in a buffer file containing data from a BMP file. Reads this data into a BMP structure. The exact pseudocode for this process can be found in the asgn5 PDF[1]

void bmp_free(BMP **bmp) Takes in a pointer to a BMP structure and frees the memory it is occupying.

void bmp_reduce_palette(BMP *bmp) Takes in a BMP structure and adjusts the values of the image that the structure is holding so that upon being rewritten into a .bmp file the image will be adjusted to simulate deuteranopia. The exact C code for this process can be found in bmp.c as well as the asgn5 PDF where it was originally taken from [1]

Results

I was able to complete all of the input/output programs and the bmp functions very quickly, but ran into an issue for about an hour where the file I was writing would be created but would always be blank. I traced the bug to my write_uint8() function, where my if statement to check if the buffer was full was encapsulating all of the functions so it was never writing anything to the file. After this, the program worked as intended for most of the images, with the exception of color-chooser. To fix color-chooser not working, I switched all of the instances of width to be rounded width in the program, and this allowed the color chooser to correctly read and write despite not having a divisible by 4 number of pixels. After this fix the program worked as intended. The images all definitely convert to a state that seems to replicate deuteranopia. I am slightly colorblind but even I was able to see pretty massive difference in some of the images. The fruit-loops image specifically looks very washed out and faded compared to the original image.

Example of image conversion

Error Handling

In order to better handle potential errors in this assignment, I implemented conditional checks in many of the prograns to check for unexpected reads/writes. If an error occurs, the program frees any allocated memory and exits the program using exit()

References

[1] Dr. Kerry Veenstra and Ben Grant. *Color Blindness Simulator.* CSE13s, UCSC.

Figure 1: Color-Chooser image before adjustment

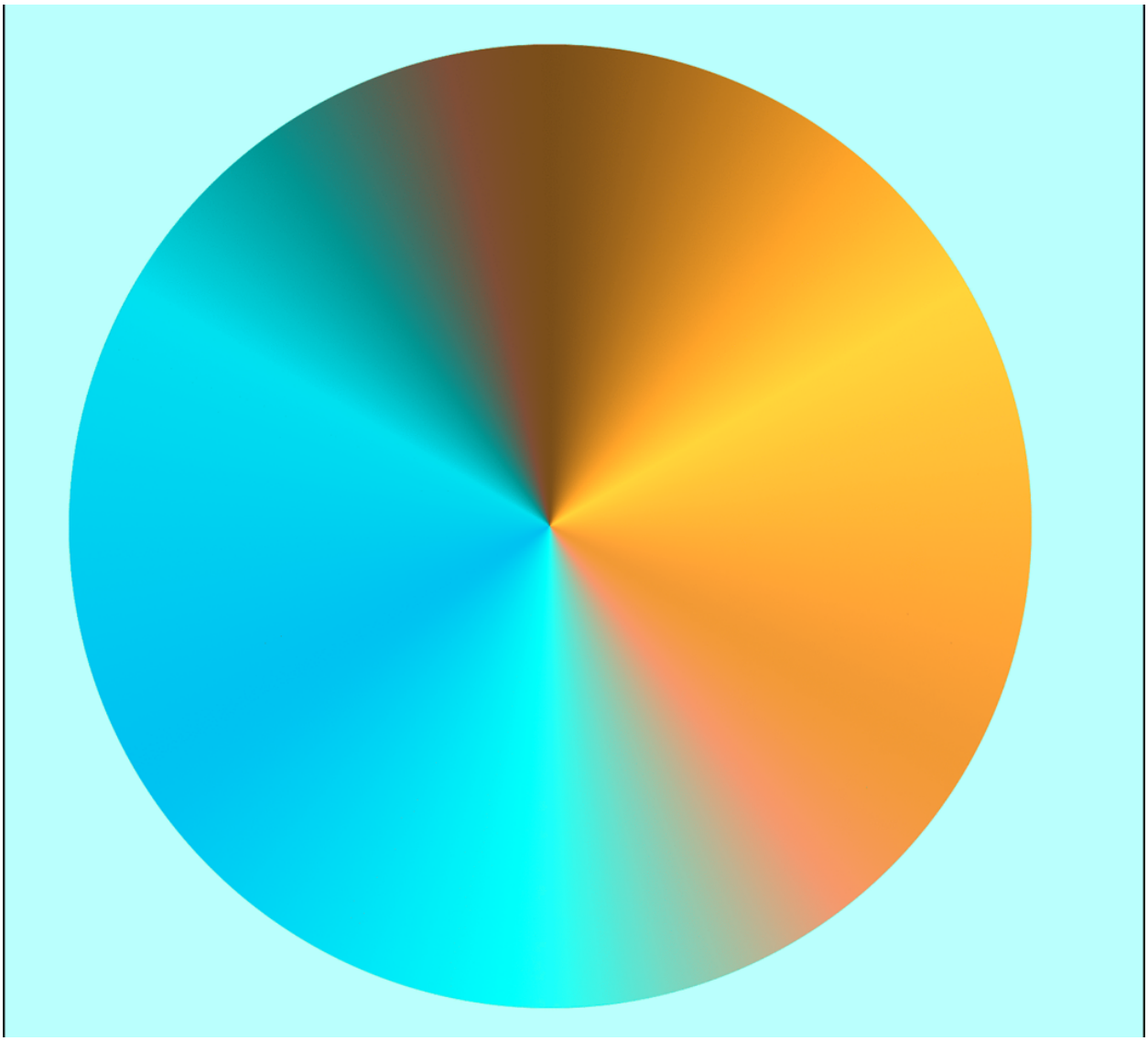

Figure 2: Color-Chooser image adjusted