Assignment 5 – Color Blindness Simulator

Mateo Garcia

CSE 13S – Spring 2023

Purpose

The purpose of this program is to be able to process an image with all of its colours, and generate a copy of the image that simulates what someone with deuteranopia would see with the same image. The program uses something something called unbuffered file-I/O functions in order to access the original image which is made up of binary, and create a new file. The images are made up of a bunch of pixels each with a set amount of red, green, or blue values, and the more a pixel has of a value, the more it appears like, so 0 red would not have any red, but 200 red would have a lot of it. If you can change the RGB values you can change the how people see the image. This process of accessing and creating files with binary is called "marshaling" or "serialization".

How to Use the Program

This program uses a shell script. The shell script looks into the **bmps** directory which is where all the BMP files are, and runs the command **colorb** on the files whose name ends in **-orig.bmp**. The corresponding output file after the script has been run has the same base but ends in **-colorbp.bmp**. The script is called **cb.sh** and if you go into the **cse13s/asgn5/** directory you can find it there.

First you want to run **./cb.sh**. That will run **colorb** on all of the test pictures. If you want to individually run **colorb**, you should run **./colorb -** [***flag1***] [***flag2***]. There are three flags that can be added after the initial command.

The following are the types of flags you can add:

- -i : Sets the name of the input file. Requires a filename as an argument.
- -o : Sets the name of the output file. Requires a filename as an argument.
- -h : Prints a help message to **stdout**.

Program Design

Data Structures

This assignment will use a lot of Unix file-I/O functions to read and write out files for the images, and have a Buffer data type. The binary of the RGB values will be altered to simulate colour blindness. The marshaling/serialization is used to read/write 8, 16, and 32 byte data and will follow the little-endian byte order. The program wil also work with Windows BMP files so it will have a BMP data type.

Algorithms

The main algorithm we will use to change the colours of the images is adjusting the colour palette of a bitmap image to simulate deuteranopia. Here is the pseudocode for it.

```
int constrain (int x, int a, int b) {
    return x < ? a :
           x > b ? b : x;
}

void bmp_reduce_palette(BMP *bmp){
    for (int i = 0; i < MAX_COLORS; ++i)
    {
        int r = bmp->palette[i].red;
        int g = bmp->palette[i].green;
        int b = bmp->palette[i].blue;

        int new_r, new_g, new_b;

        double SQLE = 0.00999 * r + 0.0664739 * g + 0.7317 * b;
        double SELQ = 0.153384 * r + 0.316624 * g + 0.057134 * b;

        if (SQLE < SELQ) {
            // use 575-nm equations
            new_r = 0.426331 * r + 0.875102 * g + 0.0801271 * b + 0.5;
            new_g = 0.281100 * r + 0.571195 * g + -0.0392627 * b + 0.5;
            new_b = -0.0177052 * r + 0.0270084 * g + 1.00247 * b + 0.5;
        } else {
            // use 475-nm equations
            new_r = 0.758100 * r + 1.45387 * g + -1.48060 * b + 0.5;
            new_g = 0.118532 * r + 0.287595 * g + 0.725501 * b + 0.5;
            new_b = -0.00746579 * r + 0.0448711 * g + 0.954303 * b + 0.5;
        }

        new_r = constrain(new_r, 0, UINT8_MAX);
        new_g = constrain(new_g, 0, UINT8_MAX);
        new_b = constrain(new_b, 0, UINT8_MAX);

        bmp->palette[i].red = new_r;
        bmp->palette[i].green = new_g;
        bmp->palette[i].blue = new_b;
    }
}
```

Function Descriptions

Constructor for Buffer:

```
typedef struct buffer Buffer;

struct buffer {
    int fd;                 // file descriptor from open() or creat()
    int offset;             // offset into buffer a[]
                            //      next valid byte (reading)
                            //      next empty location (writing)
    int num_remaining;      // umber of bytes remaining in buffer (reading)
    uint8_t a[BUFFER_SIZE]; // buffer
}
```

Functions for Buffer:

- **Buffer *read_open(const char *filename);** Open the file **filename** using the open(filename, O_RDONLY) system call.
- **void read_close(Buffer **pbuf);** Call ((*pbuf)-¿fd) to close the file and free the Buffer.
- **Buffer *write_open(const char *filename);** Open the **filename** using the **creat(filename, 0664);** system call.
- **void write_close(Buffer **pbuf);** Write any accumulated bytes that are in the buffer a[] to the file indicated by (*pbuf)-¿fd.

```
void write_close(Buffer **pbuf) {
    if (pbuf != NULL && *pbuf != NULL){
            uint8_t *start = (*pbuf)->a;
            uint32_t num_bytes = (*pbuf)->offset;
            while (num_bytes > 0){
                    ssize_t rc = write((*pbuf)->fd, start, num_bytes);
                    if (rc < 0){
                            fprintf(stderr, "write close error");
                            exit(1);
                    }
                    start += rc;
                    num_bytes -= rc;
                    }
            close((*pbuf)->fd);
            free(*pbuf);
    }
    if (pbuf != NULL) {
        *pbuf = NULL;
    }
}
```

Functions for Marshaling/Serialization:

- **bool read_uint8(Buffer *buf, uint8_t *x);** If the buffer is empty, then refill it from an open file using buf-¿fd. Then store the next byte in the buffer in *x, and update the buffer's internal state, and return true.
- **bool read_uint16(Buffer *buf, uint16_t *x);** To deserialize a uint16_t, we call read_uint8() twice to read two bytes and if either call returns false, this function returns false, because it has reached the EOF. Otherwise, copy te second byte into a new uint16_t variable and shift the new variable to the left by 8 bits.
- **bool read_uint32(Buffer *buf, uint32_t *x);** Same process as read_uint16 but this time you will call uint32_t twice. If don't both return false, then copy the second uint16_t into a new uint32_t variable and shift the new variable to the left by 16 bits.
- **void write_uint8(Buffer *buf, uint8_t x);** If the buffer is full, then call write() as many times as necessary to empty it, then set the next free byte in the buffer to x and increment buf-¿offset.
- **void write_uint16(Buffer *buf, uint16_t x);** To serialize the uint16 x, call write_uint8() twice: first with x, and then with x ¿¿ 8.
- **void write_uint32(Buffer *buf, uint32_t x);** To serialize the uint32 x, call write_uint16() twice: first with x, and then with x ¿¿ 16.

Constructor for BMP and RGB

```
typdef struct color {
    uint8_t red;
    uint8_t green;
    uint8_t blue;
} Color;
```

```
typedef struct bmp {
    uint32_t height;
    uint32_t width;
    Color    palette[MAX_COLORS];
    uint8_t  **a;

} BMP;
```

Functions for BMP:

- **void bmp_write(const BMP *bmp, Buffer *buf)** Write a BMP file.
- **BMP *bmp_create(Buffer *buf)** Create a new BMP struct, read a BMP file into it, and return a pointer to the new struct
- **void bmp_free(BMP **bmp)** Frees the memory and pointers used by the BMP struct passed in
- **void bmp_reduce_palette(BMP *bmp);** Adjust the color palette of a bitmap image to simulate deuteranopia.

Results

All my code works as intended, and the program works just fine. The hardest function to write was write_close.

Error Handling

The main errors I encountered were problems with my IO writing functions, I managed to fix them using proper class member access logic. As for my BMP I spent a lot of time because I missed one of the read_uint8() methods in the pseudocode and it caused me to have lots of problem, so it goes to show to double check your code when translating pseudocode to code.

Numeric results

Here are screenshots of the colors in the pictures and scanbuild results:

```
scan-build make
scan-build: Using '/usr/lib/llvm-14/bin/clang' for static analy
sis
make: Nothing to be done for 'all'.
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2023-05-29-0712
25-19915-1' because it contains no reports.
scan-build: No bugs found.
```

Figure 1: Screenshot of scanbuild results

Figure 2: Screenshot of apples

Figure 3: Screenshot of apples modified

Figure 4: Screenshot of wheel colour

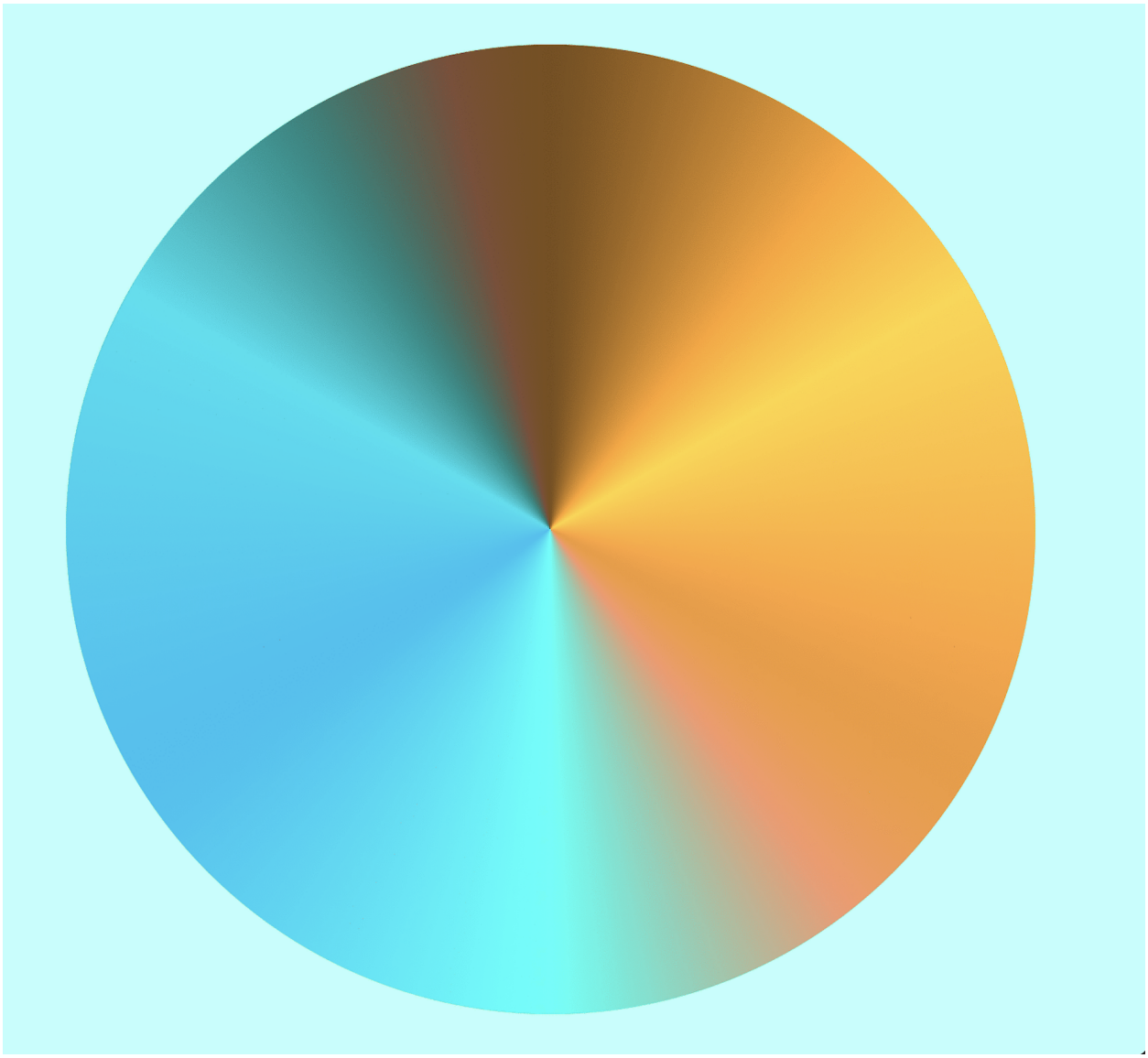

Figure 5: Screenshot of modified wheel colour