Assignment 5 – Color Blindness Simulator

Oleg Mrynskyi

CSE 13S – Spring 2023

Purpose

In this project, the goal is to develop an image-processing program that simulates the experience of color blindness, specifically deuteranopia, which is a severe form of red-green color blindness. The purpose of the program is to simulate what the pictures will look like for people who are colorblind and therefore make informed decisions on certain design choices. The image-processing program will convert normal color images into a simulation of color blindness. By implementing various algorithms and techniques, the program will modify the color values of the input image to mimic the perception of someone with deuteranopia.

By developing such a program, designers can gain insights into the range of colors experienced by individuals with deuteranopia, enabling them to create more inclusive and accessible user interfaces. This project serves as a valuable tool in the field of user interface design, allowing designers to make informed decisions regarding color choices and ensuring their products are accessible to a wider audience.

How to Use the Program

To use the image-processing program for simulating color blindness, follow these steps:

1. First, make sure you have the necessary test images in the ”bmps” directory. These images are provided to you as part of the assignment.

2. Open your terminal or command prompt and navigate to the directory where the program files are located.

3. To manually run the program on each test image, use the following command format:

```
 $ ./colorb -i bmps/[input-file-name] -o bmps/[output-file-name]
```

Replace ‘[input-file-name]‘ with the name of the original image file in the ”bmps” directory, and ‘[output-file-name]‘ with the desired name for the converted image file. This command will apply the color blindness simulation and save the modified image.

4. Repeat the above command for each test image that you want to process.

5. Alternatively, you can use a provided shell script called ”cb.sh” to automate the process. The script will automatically run the program on all BMP files in the ”bmps” directory that have names ending in ”-orig.bmp”. The resulting output files will have the same base name but end in ”-colorb.bmp”. To run the shell script, use the following command:

```
 $ ./cb.sh
```

6. If you want to work with your own image files, convert them to the required BMP format using the ”convert” command from the ImageMagick software suite. Install ImageMagick on your Ubuntu VM using the command:

```
 $ sudo apt install imagemagick
```

7. To convert an image file to BMP format, use the following command format:

```
$ convert [input-file] -colors 256 -alpha off -compress none BMP3:[output-file]
```

Replace ‘[input-file]‘ with the name of your input image file, and ‘[output-file]‘ with the desired name for the converted BMP file. Make sure the output file name ends with ".bmp". The command will convert the image to BMP format with the required specifications.

8. After applying the color blindness simulation, if you want to convert the modified BMP file back to another format (e.g., GIF), use the following command format:

```
$ convert [input-file] [output-file]
```

Replace ‘[input-file]‘ with the name of the color-blindness simulated BMP file and ‘[output-file]‘ with the desired name for the output file in the desired format.

9. If you need any assistance or want to understand the available command-line options, you can use the ‘-h‘ option. This will display a help message with usage instructions on your terminal.

By following these steps, you can use the image-processing program to simulate color blindness and convert images to the desired format, enabling you to assess the impact of color choices on individuals with deuteranopia or other forms of color blindness.

Program Design

`IO.c`

The IO.c primarily focuses on marshaling/serialization and deserialization of data, specifically for reading and writing various data types from and to a buffer. The main data structure used in this module is the Buffer struct, which represents a buffer for reading from or writing to a file. It contains fields such as fd for the file descriptor, offset to track the current position in the buffer, num_remaining to keep track of the remaining bytes in the buffer, and a which is an array representing the buffer itself.

The main algorithms in this module revolve around reading and writing data of different types from and to the buffer. These algorithms include functions such as

```
read_open(), read_close(),
write_open(), write_close(),
read_uint8(), read_uint16(), read_uint32(),
write_uint8(), write_uint16(), and write_uint32().
```

These functions handle the marshaling and unmarshaling process, ensuring that the data is correctly serialized and deserialized in little-endian byte order.

To read data from the buffer, the functions check if the buffer is empty and refill it from an open file if needed. The data is then extracted from the buffer based on the respective data type, updating the buffer's internal state accordingly. Similarly, to write data to the buffer, the functions check if the buffer is full and flush it to the file if necessary. The data is then serialized into the buffer in little-endian byte order.

`BMP.c`

The main data structure in BMP.c is the BMP struct, which represents a BMP image. It contains fields to store information about the image dimensions, color palette, and pixel data. The struct is defined with appropriate data types to accurately represent the BMP file format.

The module includes several key functions to handle BMP files. The `bmp_write` function is responsible for serializing a BMP image and creating a new BMP file. It takes a BMP struct and a Buffer object as input, and writes the necessary data to the file following the BMP file format specifications.

The `bmp_create` function serves as the counterpart to `bmp_write`. It creates a new BMP struct, reads the contents of a BMP file into it, and returns a pointer to the newly created struct. This function takes a Buffer object as input, from which it reads the data according to the BMP file format. It dynamically allocates memory for the BMP struct and the pixel array, ensuring proper storage of the image data.

Another function provided in BMP.c is `bmp_free`, which frees the memory allocated for a BMP struct once it is no longer needed. This function takes a double pointer to a BMP struct, ensuring that the original pointer is set to NULL after the memory is freed, preventing potential memory access issues.

Additionally, the module includes `bmp_reduce_palette`, a function that adjusts the color palette of a BMP image to simulate colorblindess. It applies a set of mathematical equations to modify the RGB values of each color in the palette, producing a visually altered image.

Data Structures

The BMP module utilizes the following data structures to represent and manipulate BMP images:
1. ‘BMP‘ struct: This structure represents a BMP image and contains fields to store various attributes of the image. The fields include:
- `width`: The width of the image in pixels.
- `height`: The height of the image in pixels.
- `palette`: An array of color palette entries, where each entry represents a color in the image.
- `a`: A two-dimensional array representing the pixel data of the image.
2. ‘Buffer‘ struct: This structure is used for buffering data during file read and write operations. It typically includes fields such as:
- `data`: A pointer to the data buffer.
- `size`: The size of the buffer.
- `position`: The current position within the buffer.
The ‘Buffer‘ struct provides a convenient way to manage and manipulate data during file I/O operations.

Algorithms

Function Descriptions

`BMP FUNCTIONS`

1. `bmp_write(const BMP *bmp, Buffer *buf)`:
   - Input: ‘bmp‘ (const BMP pointer), ‘buf‘ (Buffer pointer)
   - Output: None
   - Purpose: Serialize a BMP image and write it to a buffer.

2. `bmp_create(Buffer *buf)`:
   - Input: ‘buf‘ (Buffer pointer)
   - Output: Pointer to the newly created BMP struct
   - Purpose: Create a new BMP struct, read a BMP file into it, and return a pointer to the new struct.

3. `bmp_free(BMP **bmp)`:
   - Input: Double pointer to a BMP struct
   - Output: None
   - Purpose: Free the memory allocated for the BMP struct and set the pointer to NULL.

4. `bmp_reduce_palette(BMP *bmp)`:
   - Input: Pointer to a BMP struct

- Output: None
- Purpose: Adjust the color palette of a bitmap image to simulate deuteranopia.

`IO FUNCTIONS`

1. `read_open(const char *filename)`:
   - Input: ‘filename‘ (const char pointer)
   - Output: File pointer to the opened file for reading
   - Purpose: Open a file in read mode and return the file pointer.
2. `read_close(FILE *file)`:
   - Input: ‘file‘ (FILE pointer)
   - Output: None
   - Purpose: Close the file opened for reading.
3. `write_open(const char *filename)`:
   - Input: ‘filename‘ (const char pointer)
   - Output: File pointer to the opened file for writing
   - Purpose: Open a file in write mode and return the file pointer.
4. `write_close(FILE *file)`:
   - Input: ‘file‘ (FILE pointer)
   - Output: None
   - Purpose: Close the file opened for writing.
5. `read_uint8(const Buffer *buf)`:
   - Input: ‘buf‘ (const Buffer pointer)
   - Output: Unsigned 8-bit integer value
   - Purpose: Read an unsigned 8-bit integer value from the buffer.
6. `read_uint16(const Buffer *buf)`:
   - Input: ‘buf‘ (const Buffer pointer)
   - Output: Unsigned 16-bit integer value
   - Purpose: Read an unsigned 16-bit integer value from the buffer.
7. `read_uint32(const Buffer *buf)`:
   - Input: ‘buf‘ (const Buffer pointer)
   - Output: Unsigned 32-bit integer value
   - Purpose: Read an unsigned 32-bit integer value from the buffer.
8. `write_uint8(Buffer *buf, uint8_t value)`:
   - Input: ‘buf‘ (Buffer pointer), ‘value‘ (unsigned 8-bit integer)

- Output: None
- Purpose: Write an unsigned 8-bit integer value to the buffer.

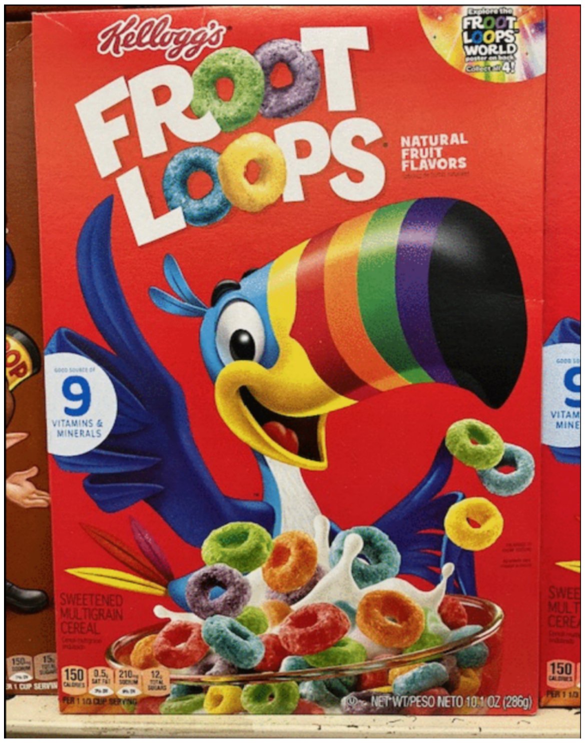
9. `write_uint16(Buffer *buf, uint16_t value)`:
   - Input: ‘buf‘ (Buffer pointer), ‘value‘ (unsigned 16-bit integer)
   - Output: None
   - Purpose: Write an unsigned 16-bit integer value to the buffer.
10. `write_uint32(Buffer *buf, uint32_t value)`:
    - Input: ‘buf‘ (Buffer pointer), ‘value‘ (unsigned 32-bit integer)
    - Output: None
    - Purpose: Write an unsigned 32-bit integer value to the buffer.

Please note that these functions are related to input/output operations and are used to read from or write to a buffer. They are designed to handle specific data types and provide an interface for interacting with the buffer.

Results

When run ./colorb -i bmp/froot-loops-orig.bmp -o froot-loops-map.bmp :

`Before:`



`After:`

Kellogg's
FROOT LOOPS
NATURAL FRUIT FLAVORS
Explore the FROOT LOOPS WORLD poster on back
9 VITAMINS & MINERALS
SWEETENED MULTIGRAIN CEREAL
150 CALORIES
NET WT/PESO NETO 10.1 OZ (286g)

References