CSC 412 – Operating Systems

Programming Assignment 04, Spring 2021

Monday, February 22nd, 2021

Due date: Tuesday, March 3rd, 11:55pm.

1 What this Assignment is About

1.1 Objectives

The objectives of this assignment are for you to

- Write C code making use of 1D and 2D arrays, and exploiting pointer algebra;
- Work with a small third-party library (here, to load and write TGA images);
- Write some simple image processing applications;
- Use a bash script to search through a directory;
- Start using Doxygen to produce documentation for your code.

This is an individual assignment.

1.2 Handout

Besides this document, the handout for this assignment is a zip archive containing the code of a small library for loading and writing images in the TGA file format, a simple program demonstrating the use of the library on a sample image, and a set of images in the TGA file format.

2 Part I: TGA Image Files and the Code Handout

2.1 The TGA file format

The TGA (Targa file) image format is one of way too many image formats that you may run into. Other formats provide better compression or support for a wider range of colors, so why use this format at all? Because the format for *uncompressed* `.tga` image files is the only truly multi-platform format that is easy to read and write. We could use a more complete library such as `freeimage` or `ImageMagick`, but, as long as your image file stores data uncompressed and does not contain any comments, the code supplied here will do the job.

2.2 The code supplied

The code supplied consists of:

- `rasterImage.h` and `rasterImage.c` are respectively the header and source file that define `ImageStruct` data type and implement a function to create an image of a particular type and dimensions.
- `imageIO_tga.h` and `imageIO_tga.c` are respectively the header file and the source code for the implementation of functions to read and write color and gray-level images in the uncompressed TGA format.
- `main.c` is the code of a small program that shows how to use the library to read and write image files, and to access the pixel data of an image.

2.3 About the `image.h` header file

I want to attract your attention to a few important points regarding this header file.

First, if you are familiar with javadoc style of comments, you probably recognized the characteristic `/**` . For many years, C++ programmers secretly envied their Java-programming colleagues who could produce good-looking code documentation through the javadoc system. They envied them until a brave soul came up with Doxygen, which interprets javadoc-style comments for a multitude of programming languages (C, C++, Python, Perl, etc.). Therefore, coding in C is not a good excuse to have a lousy documentation anymore, and, starting with this assignment, you will be expected to write javadoc-style comments and produce documentation in the form of html files. I will return to this point later in this document.

Second, the two `enum` types, `ImageFileType` and `ImageType`, mention types that we won't encounter in this assignment. We will only deal with `kTGA_COLOR` images that we will store as `RGBA32_RASTER`. I might as well observe here that I love using `enum` types to represent and group constants that belong together. I already alluded to that in the C code samples that I posted on Sakai at the beginning of the semester. Enumerated types work well with switch statements, are easy to expand, and have all the advantages of an integral type (for calculations and array indices) while offering some—limited—protection regarding range.

Next, the `ImageStruct` type stores the 1D raster master pointer as a `void*` pointer. This is a common design decision in C and C++, when the type of the data could be any of multiple possible types (here, besides the 4 bytes of color information that we will be dealing with, we could have a float value or a single byte). Set the pointer to be `void*`, and cast it back to the proper type when the data must be processed. We will encounter this pointer trick several times this semester.

Finally, the `ImageStruct` type contains a field for a 2D array (from the name, but it is also cast to a `void*` pointer). This field is currently not used anywhere in the handout's code, because the corresponding 2D array has not been allocated. This will be a task for you to complete when I propose you later in this assignment to "scaffold" a 2D array on top of an existing 1D array (for extra credit).

2.4 Color images and rasters

Pretty much all libraries dealing with image and video data[1] manipulate image data under the form of a 1D "raster," as shown in Figure 1.

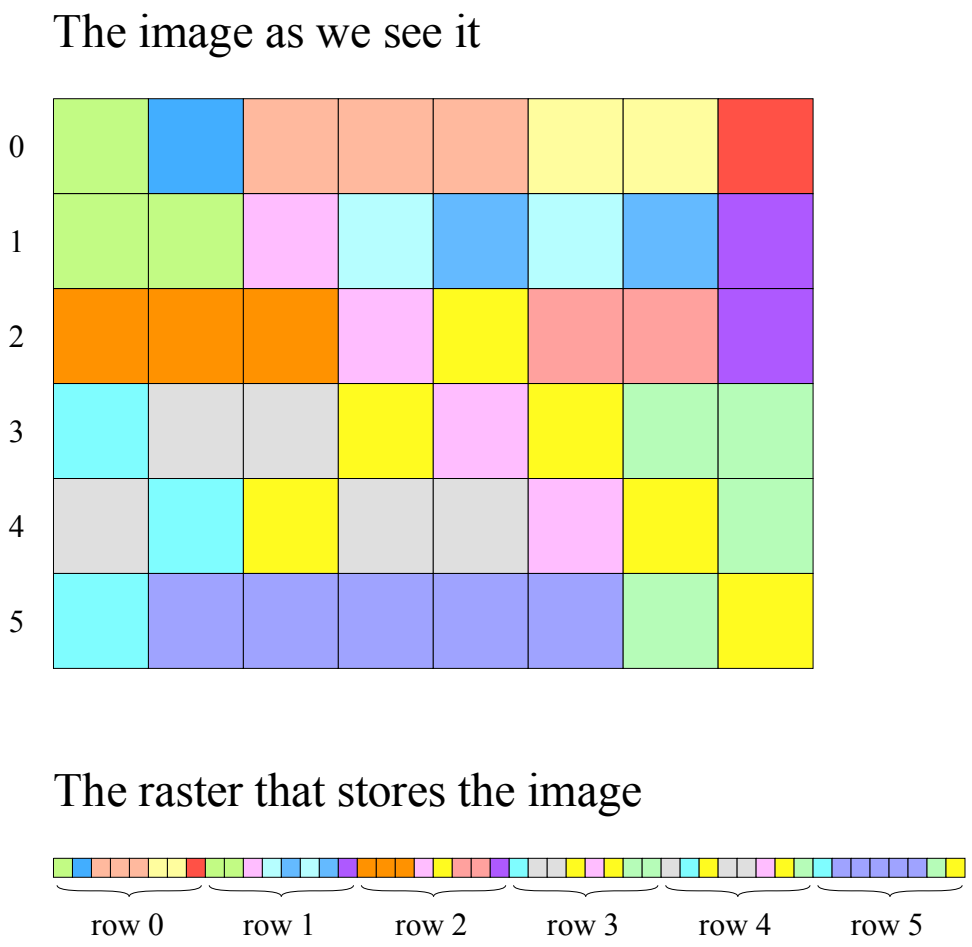

Figure 1: An image stored as a 1D raster.

So, if our image has m rows of n columns and each pixel is stored on k bytes (for a total of $n \times k$ bytes per row and $m \times n \times k$ bytes for the entire image), then the pixel at row i and column j in the image can be addressed at index $i \times \times k + j \times k$ of the 1D raster.

In the uncompressed TGA file format, an image file stores each pixel as 3 unsigned bytes (range 0 to 255) encoding the red, green and blue color channels. When we load an image in memory, however, it is more convenient to store each pixel on an even 4 bytes. Besides allowing image processing and other graphic applications to use this additional byte to encode *transparency* (or the "α channel," as it is called), this also means that a pixel occupies the same space as an `int`, so

[1] An important exception is Apple's old QuickTime library and the `libquicktime` open-source replication of the API for a tiny bit of QuickTime. I mention this here because we may have an encounter with `libquicktime` before the end of the semester.

that we can view our 1D raster either as an `int*` pointing to a pixel or as an `unsigned char*` pointing to a color channel of a pixel.

Note that different applications may store the bytes of the color channel in different orders. The most common orders are `argb` (alpha, red, green, blue) and `rgba`. Throughout this assignment, and future assignments dealing with images, we will use exclusively the `rgba` format.

What this means is that if I want to access the color information of the pixel at row i and column j of the same $m \times n$ image, then I could get a pointer to this pixel's color channels, each seen as an `unsigned char` by writing

```
unsigned char* rgba = (unsigned char*)raster + 4*(i*n + j);
```

Now, I can view this pointer as an array and access directly my color components:

- `rgba[0]`: red channel,
- `rgba[1]`: green channel,
- `rgba[2]`: blue channel,
- `rgba[3]`: alpha channel.

Alternatively, I could see my pixel as a 4-bye `int` that I access through an `int*` pointer by writing

```
int pixel = *((int*)raster + i*n + j);
```

Now I can access my color channels by extracting the different bytes of the `pixel` variables. There is just a small problem: the byte order. Different CPU architectures have a different way to encode `int`, `float`, etc. Or rather, the encoding is the same, but the byte order is different. Intel processors (and their AMD clones) are "small-endian," which means that their least significant byte is stored first, and their most significant byte is stored last.

2.5 The demo program `main.c`

This small programs reads a TGA image into an `ImageStruct`, performs simple operations on the raster data (mirror an image vertically and "kill" one of the color channels) and writes the resulting image as a new TGA file. In other words, it demonstrates all the elementary operations that you will need to combine to complete this assignment. In addition, I give two different versions of either operation. Make sure that you understand well all parts of the code. Post questions on the forums.

2.6 Extra credit (6 points)

Modify the code of `rasterImage.h` , `rasterImage.c`, and `readTGA.c` to implement the 2D raster functionality. Please keep in mind that this 2D raster cannot constitute a duplication of the storage. It has to be a simple scaffolding of a 2D array on top of the 1D raster, giving you a more convenient way to access the data of the 1D raster. Verify that your implementation is correct by using the 2D raster rather than the 1D raster to "kill" that color channel in `main.c`.

3 What to Do, Part II: Image Operators

You are going to implement several small programs that all perform a simple operation on an image. These programs will all use the handout code provided to read and write images.

3.1 Report dimensions

This program named `dimensions` takes as main argument the path to a TGA image file and reports the dimensions of the image. This program may be called with different optional modifiers that affect the output of the program:

- `-h` Output image height only;
- `-w` Output image width only;
- `-v` verbose output.

The modifier `-v` may be combined with `-h` or with `-w`, in either order, but the modifiers `-h` and `-w` are mutually exclusive.

Assuming that the program is executed with the image file `cells.tga` provided with the handout (that image has a width of 640 and a height of 640), possible outputs would be:

- `dimensions ../Images/cells.tga` → `640 480`
- `dimensions -h ../Images/cells.tga` → `480`
- `dimensions -w ../Images/cells.tga` → `640`
- `dimensions -v ../Images/cells.tga` → `width: 640, height: 480`
- `dimensions -vh../Images/cells.tga` → `height: 480`
- `dimensions -wv../Images/cells.tga` → `width: 640`

Error handling:
Your program should report if the image is not found (invalid path) or cannot be read, or if invalid optional modifiers are passed to the program.

3.2 Cropping

This program named `crop` takes as arguments the path to a valid TGA image file, the path to an output directory, the x and y coordinates of a cropping rectangles within the image, and the width and height of that cropping rectangle. It writes in the output folder a new image file that contains the sub-image defined by the crop rectangle.

The name of the output image should be that of the input image, with the `[cropped]` suffix. For example, if the input image is named `myImage.tga`, then the output image should be `myImage [cropped].tga`.

3.3 Image rotation

This program named `rotate` takes as arguments a string specifying the rotation to perform, the path to a valid TGA image file, and the path to an output directory. It writes in the output folder a new image file that contains the rotated image.

The string specifying the rotation can be one of the following: `l` (rotate left by 90 degree), `r` (rotate right by 90 degree), `ll` or `rr` (rotate by 180 degree).

The name of the output image should be that of the input image, with the `[cropped]` suffix. For example, if the input image is named `myImage.tga`, then the output image should be `myImage [cropped].tga`.

Extra credit (up to 3 points)
For 2 points, accept the rotation specification string in uppercase or lowercase (or mixed). For an additional point, take any string made up of `L` or `R` and reduce it to the effective rotation specified. For example, `LLRRRRRLRRLLR`, which amounts to 6 left rotations and 8 right rotations, effectively reduces to `LL` or `RR`.

3.4 Split color channels

This program named `split` takes as arguments the path to a valid TGA image file and the path to an output directory. It outputs in that folder the three image files obtained by separating the red, green, and blue channels of the input image. If the name of the input image was `myImage.tga`, then the output images should be named `myImage_r.tga`, `myImage_g.tga`, and `myImage_b.tga`.

3.5 Extra credit (up to 6 points)

Before writing the output image, verify if a file with the same name already exists in the output folder (2 points). If it does, add an index to the new output image (3 points). For example, when writing a cropped version of `myImage.tga` , if the file `myImage [cropped].tga` already exists, name the new file `myImage [cropped 2].tga`, unless `myImage [cropped 2].tga` already exists, in which case name it `myImage [cropped 3].tga`, etc. For an additional point, retroactively modify the suffix of the first output image `[cropped 1]`

4 What to Do, Part III: Scripting

4.1 Input parameters

Your script should be named `script04.sh` and take in three arguments:

- The path to a directory where to find images in the uncompressed TGA file format,
- The path to folder containing your the source files (and the handout code, as specified later);
- The path to an output folder.

4.2 Tasks to perform

Your script, named `script04.sh`, should first build the executable for each of the image processing programs, and then, for each TGA file found in the input folder, use the appropriate image operator to produce in the output folder:

- Three images storing the separate color channels of the input image;
- The result of a left rotation of the input image.

4.3 Extra credit: Break an image into quadrants (6 points)

This enhanced version of the script, named `script04EC1.sh`, should produce, in addition to what the basic `script04.sh` already delivers,

- The images storing the division of the input image into four quadrants.

5 What to submit

5.1 The pieces

A complete submission will include:

- a separate source file for each of the image processing, operations;
- the version of `rasterImage.h`, `rasterImage.c`, `imageIO_tga.h`, and `imageIO_tga.c` that you used to build your image operators;
- your `bash` script,
- the html documentation produced by Doxygen for your programs,
- a report.

Do **not** submit the image files and `main.c` program of the handout.

5.2 Organization

You should submit a zip archive of a folder named `Prog04` containing

- your report,
- a directory named `Documentation` containing the html documentation produced by Doxygen,
- a directory named `Programs` containing
  - a source file for each of the image operators,

  - a directory named `Code Handout` containing the version of `rasterImage.h`, `rasterImage.c`, `imageIO_tga.h`, and `imageIO_tga.c` that you used to build your image operators.

- A folder named `Scripts` which contains the `script04.sh` script file and possibly `script04EC1.sh` if you implemented it.

5.3 The report

The report should discuss your design decisions. If you had to interpret some of the specifications of the assignment then state clearly what you did (definitely so if you decided to only implement a subset of the requirements).

Identify any limitations of your programs (besides the obvious, e.g. the fact that you can only handle TGA image files). Are there combinations of arguments that cause your program to crash? Did you decide to only implement a subset of the requirements?

Finally, if you implemented any of the "extra credit" sections, make sure to point it out, so that the graders know what to look for.

5.3.1 Execution

We are going to test your programs and script with different input files and lists of arguments. Your grade for this section will reflect to what extent you produce the desired output for all the test cases. So, make sure that you test your program and script before you submit.

5.3.2 Code quality

For this part of the grade we look at things such as:

- Proper indentation;
- Good comments. This part of the grade does not cover the javadoc-style comments (they are evaluated in the Documentation section) but comments in the code, explaining important blocks of code. Comments should not be simple translations of the code in plain language (e.g. `x +=2;` being commented as "add 2 to x").
- Good choice of identifiers (chances are that `a`, `b`, and `c` are not good identifiers, particularly if the first two are strings and the third one an integer. In any case, identifiers should clearly indicate the purpose of a variable or function.
- Consistent choice of identifiers. For example, if you have a variable `int beanCount` that does what it claims to do, then it would be a bad idea to have another variable `int num_of_boxes`.
- Good implementation decision: avoid super long functions (at your current skill level, if your function occupies more than one page of your screen, it's probably too long); select proper data types; create your own when appropriate.

Your grade for this section will reflect how much you actually implemented. A superbly commented and indented "Hello World!" program won't bring you many "code quality" points.

5.3.3 No syntax error

Code with syntax errors (compiler errors for the C program) will get a grade of 0 for execution and quality. Comment out the part with syntax errors and explain in the comments what you were trying to do.

5.4 Grading

- Execution: 45%
  - C program: 30%
    - `dimensions`: 7%,
    - `split`: 6%,
    - `rotate`: 8%,
    - `crop`: 9%.
  - `bash` script: 15%
- Code quality: 25%
  - C program: 15%
  - `bash` script: 10%
- Documentation produced by Doxygen: 10%
- Report: 10%
- Folder organization: 10%