LW: More Image Manipulation

For completion credit, you need 20 of 35 points.

# Objectives

Work with

* two-dimensional arrays (writing, reading, iterating)
* Removing rows and columns from two-dimensional arrays
* Remember: you must submit the lab to Mimir **and** attend the lab session to get credit. See the syllabus on how to get credit if you have an excused absence.

# Overview

We frequently want to squeeze an image into a smaller size. In this labwork, we will take images and remove lines to make the image smaller.

If you finish everything and still have time, How would you write the program to get a target width and height and create the image that is the right size using the approach used in this lab?

## Example Photo

* Full-color photo of Reveille.[[1]](#footnote-0)
* Height reduced image of Reveille. Every other line removed..
* Width reduced image of Reveille. Every other line removed..

# 

# Requirements

* Get the start code from Mimir.
  + You do not need to edit:
    - image.cpp
    - functions.h
* Get Image Files you can use[[2]](#footnote-1)
  + From last week's labwork:
    - [Download Zip file](https://drive.google.com/file/d/1BSmDbPoRPDNPIi6AG5WbQo-lqQDAU_OQ/view)
    - [See image files on Google drive](https://drive.google.com/drive/u/1/folders/10KHsZOw6wX2vMg4cZShhMdcMFjtp9Ttq)
* You may only #include
  + <iostream>
  + <fstream>
  + <cstring>
  + <cmath>
  + <limits>
  + “functions.h”
* Read over the starting code.
  + Understand the flow of the program.
    - What are different ways it could have been organized?
  + The struct Pixel is defined in functions.h
  + MAX\_WIDTH and MAX\_HEIGHT are defined as global constants in functions.h.
  + The main program already reads in a filename and the width and height. You can use that to test your program.
  + You will need to use an image viewer to view your PPM file. See the “Viewing ppm files” section below for information on viewing your PPM files.
* Review (calls removeHorizontalLine)  
  int removeHorizontalLines(Pixel image[MAX\_WIDTH][MAX\_HEIGHT], int width, int height, unsigned int frequency);
  + This function will skip frequency rows and then remove the next row. This is done repeatedly until there are no more rows to process.
  + The first parameter is a 2d array of Pixels (structs) that hold a color value
  + The second parameter is the width of the array needed for traversing the array
  + The third parameter is the height of the array needed for traversing the array.
  + The fourth parameter is the number of rows to skip before removing a row.
    - If the frequency is zero, then do not modify the image.
* Implement (is called by removeHorizontalLines)  
  void removeHorizontalLine(unsigned int index, Pixel image[MAX\_WIDTH][MAX\_HEIGHT], int width, int height);
  + This function will remove the row of pixels indexed by index from the image.
  + The first parameter is the index of the row to be removed.
  + The second parameter is a 2d array of Pixels (structs) that hold a color value
  + The third parameter is the width of the array needed for traversing the array
  + The fourth parameter is the height of the array needed for traversing the array.
* Implement (calls removeVerticalLine)  
  int removeVerticalLines(Pixel image[MAX\_WIDTH][MAX\_HEIGHT], int width, int height, unsigned int frequency);
  + Analogous to removeHorizontalLines
* Implement (is called by removeVerticalLines)  
  void removeVerticalLine(unsigned int index, Pixel image[MAX\_WIDTH][MAX\_HEIGHT], int width, int height);
  + Analogous to removeHorizontalLine

# 

# Supporting Information

## Images and RGB Color Model

Images are a two dimensional matrix of pixels where each pixel is a color composed of a red, a green, and a blue value. For example RGB(80, 0, 0) is Aggie maroon.[[3]](#footnote-2)

In image processing, pixel (x,y) refers to the pixel in column x and row y where pixel (0, 0) in the upper left corner of the image and pixel (width-1, height-1) is in the lower right corner.

***Warning:*** This is column-major ordering which is transposed from the row-major ordering that is used for cartesian coordinates where the first index is the row and the second index is the column and (0, 0) is in the lower-left corner. In image files, the width is essentially the number of columns and the height is the number of rows. **So the impact is that you will index with [col][row] instead of with [row][col]**.

Coordinates for a 3X4 (3 columns (i.e. width) by 4 rows (i.e. height) image are shown in the following table.

|  |  |  |
| --- | --- | --- |
| (0, 0) | (1, 0) | (2, 0) |
| (0, 1) | (1, 1) | (2, 1) |
| (0, 2) | (1, 2) | (2, 2) |
| (0, 3) | (1, 3) | (2, 3) |

We will use a Pixel struct (defined in functions.h) that holds a value for red, green and blue. The image will be a 2 dimensional array of Pixels.

### Grayscale

Grayscale images use shades of black. Shades of black have the same value for the red, green, and blue color values. For example RGB(100, 100, 100). To convert a color to grayscale, average the red, green, and blue color values. For example for RGB(75. 30, 95), we calculate (75+30+95)/3 to get 66.6666667. You must **round** to the nearest integer, so the corresponding grayscale color is RGB(67, 67, 67). These RGB values will replace the previous values.

* **Round** any value to the nearest whole number.
  + Use round() from <cmath>

### 

### Sepia

Sepia tones are based on aged photos that have a brownish tinge to them. Replacement colors for making a sepia image use a formula to infuse different proportions of the red, green, and blue into replacement values for each color. The following ratios are from [a Java tutorial on creating sepia images](https://www.dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image).

newRed = 0.393\*oldRed + 0.769\*oldGreen + 0.189\*oldBlue

newGreen = 0.349\*oldRed + 0.686\*oldGreen + 0.168\*oldBlue

newBlue = 0.272\*oldRed + 0.534\*oldGreen + 0.131\*oldBlue

* **Round** any value to the nearest whole number.
  + Use round() from <cmath>
* If a calculation results in a value greater than 255, then set the value to 255.

## Image File Format (PPM)

You are probably already familiar with common image formats such as JPEG, PNG, and GIF. However, these formats all use some type of data compression to keep file sizes relatively small. However, we are not ready to tackle these formats in C++.

We are going to use an image format that only requires basic text file I/O.

[The PPM (portable pixel map) format](http://netpbm.sourceforge.net/doc/ppm.html) is a specification for representing images using the RGB color model. PPM is not used widely because it is very inefficient (for example, it does not apply any data compression to reduce the space required to represent an image.) However, PPM is very simple, and there are programs available for Windows, Mac, and Linux that can be used to view ppm images. Even more conveniently, you can use an online tool with your browser to [view your PPM files online](http://paulcuth.me.uk/netpbm-viewer/) or convert it into a widely supported format such as JPG. We will be using the [plain PPM version](http://netpbm.sourceforge.net/doc/ppm.html#plainppm), which stores the data in ASCII (i.e. plain text) rather than in a binary format. Since it is plain text, we will be able to use text file I/O to read and write these image files.

Note that the pixels in a PPM file are given row by row, so is essentially row-major ordering which is transposed from the array image format which is column major.

If you do create your own plain / ASCII PPM files make sure you **remove the comments**, since we are not addressing how to identify and ignore comment lines. Comments are lines that start with the ‘#’ character. I used [the GIMP](https://www.gimp.org/) to create the PPM files provided with the starting code.

### PPM File Specification

* First line: string “P3”
* Second line: width (number of columns) and height (number of rows)
* Third line: max color value (for us, 255)
* Rest of the file: list of RGB values for the image, expressed as a raster of rows, from top to bottom. Each row contains the RGB values (i.e., three values) for each column.

### PPM Example

***Note:*** We have added colors to emphasize that every three numbers represent a single pixel. This version has each row on a separate line.

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0 0 0 0 0 255 0  255 255 255 255 0 255 0 0 0 0 255 0  255 255 0 0 0 255 125 0 255 255 0 125 0 0 255 0 255 255 125 125 125 239 239 239 |

This version is the same as above, but with spaces added to help you visualize the file.

|  |
| --- |
| P3 4 4 255  0 0 0 255 0 0 0 0 0 0 255 0  255 255 255 255 0 255 0 0 0 0 255 0  255 255 0 0 0 255 125 0 255 255 0 125  0 0 255 0 255 255 125 125 125 239 239 239 |

This version has all numbers on a single line.

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0 0 0 0 0 255 0 255 255 255 255 0 255 0 0 0 0 255 0 255 255 0 0 0 255 125 0 255 255 0 125 0 0 255 255 255 0 125 125 125 239 239 239 |

Alternatively, it could be saved with one pixel per line (i.e. 3 numbers per line) or even one number per line (this is what the GIMP did when I used it to create PPM files).

The following also works but makes no sense to a human reading it.

|  |
| --- |
| P3 4 4 255 0 0 0 255 0 0  0 0 0 0 255  0 255 255 255 255  0 255 0 0 0  0 255 0 255 255  0 0 0 255 125  0 255 255 0 125  0 0 255 255 255  0 125 125 125 239  239 239 |

Sample PPM File: [blocks.ppm](https://drive.google.com/file/d/1mQryj1MdhjtWhs6VMsI1viTNlrHTRebk/view) 

### Viewing PPM files

You’ll need to view your PPM files to see the results of your program. Unfortunately, PPM is not supported by many image viewers.

Some options for viewing your files include:

* [Drag files onto this website](http://paulcuth.me.uk/netpbm-viewer/) (http://paulcuth.me.uk/netpbm-viewer/)
  + You don’t have to download any programs!
* [The GIMP](https://www.gimp.org/) is an open-source image editing program similar to Photoshop.
  + ***Warning:*** This is a very large program.

1. [Image taken from TAMU Galleries, Fall 2019](https://tamu.photoshelter.com/galleries) [↑](#footnote-ref-0)
2. [Image taken from TAMU Galleries, Fall 2019](https://tamu.photoshelter.com/galleries) [↑](#footnote-ref-1)
3. [TAMU Web Color Pallette (https://brandguide.tamu.edu/web/web-color-palette.html)](https://brandguide.tamu.edu/web/web-color-palette.html) [↑](#footnote-ref-2)