Arlen Dumas
Connor Gray

CSC 411 Assignment 3

Acknowledgements:

While waiting for office hours with both Dr. Daniels and Colin, we spoke conceptually with other classmates regarding the control flow of the program and how to best implement the concepts from the class into the assignments.

The Array2 implementation was taken directly from Dr. Daniels

Implementation:

All parts of ppmtrans were correctly implemented with the addition of all extra credit transformations.

The program implements block major storage but does not work correctly.

Design Checklist:

1. Architecture

   The architecture of the program can be broken into two distinct parts; command line argument parsing and transformation functions.

   The argument parsing part of this assignment utilizes clap to match the command line arguments to the possible valid inputs that can be given by the user. These matches are passed to the appropriate transform function as long as they are valid which the clap code ensures or else it throws an error to the user input.

   Because each transformation only needs the original height and weight to identify the space needed for a given output, the transformation part of this assignment utilizes curried functions to pass only the necessary information to our transformation function. By doing this, we're able to condense each transformation effect into one match function that returns a new function with the appropriate row and column transformation equations. Depending on if the accessing method is column major or row major, we can then fill the box with the corresponding pixels.

   The function that performs the transformations accepts a parameter that implements an iterator of a specific type; this specific iterator implementation is supported by array2 for both row and column major and by array2b for block major storage. A vector of default pixels is created that is then mutated to store

the transformed image pixels; this vector is then stored in an RgbImage and written to an output file or standard output.

2. Invariants

The destination indices for a 90 rotation given the original row and column positions.

(new) col_pos = width - row_pos - 1
(new) row_pos = col_pos

The destination indices for a 180 rotation given the original row and column positions.

(new) col_pos = height - col_pos - 1
(new)row_pos = width - row_pos - 1

The row and column indices for block major storage
(The values within the blocks are stored row major order and the blocks themselves are stored column major order)

row_pos = (((i % (blocksize * height)) % blocksize) + ((index / (blocksize * height)) * blocksize)
col_pos = (i % (blocksize * height)) / blocksize

Part D:

Each of the reported times were run on the same 1400 x 1400 image

	row-major	col-major
180 degree	real 0m2.860s user 0m2.597s sys 0m0.051s	real 0m3.193s user 0m2.881s sys 0m0.080s
90 degree	real 0m3.255s user 0m2.989s sys 0m0.021s	real 0m3.472s user 0m3.176s sys 0m0.070s

In general, the results from this test coincide with our predictions from part C of the design document.

Row major access will map 90 degrees into columns, so there will be a cache miss after the first value is mapped. The cache will need to load the next column into memory for writing, then after the first hit there will be another miss and so on.

Column major access will map 90 degrees into rows, so there will be a cache miss after the first value is mapped. The cache will need to load the next column into memory for writing, then after the first hit there will be another miss and so on.

Row major access will map 180 degrees into rows, so there won't be a cache miss until we come across an index that isn't in the cache.

Column major access will map 180 degrees into columns, so there won't be a cache miss until we come across an index that isn't in the cache.

Time:

This project took approximately —-- hours to complete over the following tasks

- Design document: 1 hour
- Block-major: ~10+ hours
- ppmtrans: ~10+ hours
- Help hours: ~2 hours