Jack Adkins (jadkin05)
Krish Dhayal (kdhaya01)

Locality Design Doc

Part C: PPMTrans

Step 1: Read in the ppm file to our stucture
- Use pnm.h, take advantage of its internal error checks
- Either col-major or row-major

Step 1 Testing:
- Send in files that are in the wrong format or files that are empty to make sure we are returning errors when that occurs
- Print out the array after the ppm file has been read in. Use row-major mapping function with a simple print

Step 2: Implement the 0, 90, 180, and 270-degree image rotations
- The 270-degree rotation will be the same as the 90-degree just in the other direction, while the 0-degree rotation will just return the given ppm
- Each one is created as its own apply function to be passed into the various mapping functions
- The last pointer to be sent into the apply/mapping functions will be the same type as the array being roasted but with the expected dimensions after the rotation

Step 2 Testing:
- Test for 0 degree: make sure image is unchanged
- Test for 90 & 270, compare the coordinates (x,y) of each pixel and make sure the values of each have changed accordingly based on the rotation
- For all rotations make check the expected new dimensions of the image against the actual ones
- Give the program a variety of different shaped ppm files (squares, rectangles, extremely small, extremely large) to avoid edge cases being left out

Step 3: Implement the time function
- Create a timer that starts right before each image modification and ends as soon as the modification is complete so that we can get an idea of the speeds which each operation is completed at
- Going to use the built-in CPU timing

Step 3 Testing:

- Check the timer on other non-image adjustment functions like our print one to make sure the timer is working properly
- The main testing for this will be making sure that the timer is only timing the duration of the image adjustments(steps 2 & 4), so nothing runs in between the timer being called and the mapping function for the image adjustment being called.

Step 4: (Time Permitting) Implement both flip vertical and horizontal as well as the transpose command

- Each of these will also be in the form of apply functions that can be called through the various mapping functions
- For flip vertical we will be flipping (making negative) the y coordinate of each pixel and for flip horizontal we will be making negative the x coordinate of each pixel
- For transposing an image we will swap the x and y coordinates of each pixel along with the width and height dimensions of the array.
- We will either be using a new array and creating the flipped/transposed version there or just editing the existing array directly by swapping 2 pixels values with each other

Step 4 Testing:

- Since for any two pair pixels for each transformation (ie (x,y) and (y,x) for transposing) check that they have actually swapped after the transformation is complete
- Give the program a variety of different shaped ppm files (squares, rectangles, extremely small, extremely large) to avoid edge cases being left out
- Do similar dimension checks as in step 2 to make sure that they flip during transposing but are untouched during flips.

Part D: Estimations and Justifications

	Row-Major	Column-Major	Block-Major
90-Degree Rotation	1	5	3
180-Degree Rotation	1	5	3

Row Major Justification:

Our UArray2 interface that we designed implements the UArray2 as a UArray of UArrays where each of the inner UArrays is a row in the 2D array and the outer UArray is the first column. Because of this, when using UArray2.h all the elements in a single row are already stored next to each other in memory, meaning there is extremely good spatial locality for row-major access with our UArray2 implementation. It definitely still greatly depends on the block size and cache size, but this dramatically increases the chances of elements in the same row being stored in the same line in the cache resulting in a hit because the elements are already there. As far as comparing the 90-degree rotation to the 180-degree rotation, they both still require a full mapping function of the entire UArray2 in the same manner resulting in the same locality and cache hit rates.

Block Major Justification:

For this access method, we are using the UArray2b interface, which while different, still uses a Uarray2 containing other UArrays for storage. That being said, based on the Uarray2b being read into block by block and then being mapped block by block, there is also a fair amount of spatial locality, therefore similar to the row-major access (although not as much so) there will be a greater chance of neighboring elements in a block being stored in the same line in the cache resulting in that increased hit rate when accessing the UArray2b. While the row-major access and block-major access functions are very similar in their spacial locality, we would expect that more often than not the row-major accessing will have higher hit rates because the block-major access will miss much more often when the blocksize is smaller than the width of the 2D array because the spacial locality would now be greatly decreased. Again, for the different degree rotations they both still require a full mapping function of the entire UArray2b in the same manner resulting in the same locality and cache hit rates.

Column Major Justification:

As we mentioned in the row-major access section we implemented our UArray2 such that each row in the 2D array is stored in its own UArray, therefore when it comes to accessing by column major the hit rates are going to be extremely low. When we access by column major the next element we reach is always going to be in a different UArray than the previous, meaning that there will be little to no spatial locality involved. As a result, the cache itself will have a much higher chance of elements in the same row being stored in the same line than elements in the same column would, leading to may more misses or a low hit rate when accessing. Eviction issues could also arise with large ppms and small block sizes as lines would get filled and need to be evicted before the completion of a column that would also result in slowdowns and further misses. Similarly, for the different degree rotations they both still require a full mapping function of the entire UArray2 in the same manner resulting in the same locality and cache hit rates.