tmicom01, ekang05

Professor Monroe

Comp-40

09/30/2019

Locality

Part I: Implementation plan for Part C

1. function to read in image pixels.
    a. Use Pnm_ppmread to read in values into a 2D array
    b. Testing:
        i. pointer to 2D shall not be null
        ii. Use Pnm_ppmwrite to check if values in 2D array matches the original file

2. Free memory:
    a. Use Pnm_ppmfree
    b. Test:
        i. Run valgrind to check memory leaks
3. Rotate 0:
    a. Doesn't change the original 2D array
    b. Testing:
        i. Print elements of the array before call, and after the call. Thus, compare. It should be the same values in the same order,
4. Rotate 90 (clockwise):
    a. We will use map-row-major to iterate through every element using the geometric calculations given in the Spec (h - j - 1, i).
    b. Testing:
        i. Hard code a rotated 90 image and compare it to the results returned by the function we implemented.

5. Rotate 180 (clockwise):

    a. We will use map-row-major to iterate through every element using the geometric calculations given in the Spec (w - i - 1, h - j - 1).
    b. Testing:

        i. Hard code a rotated 180 image and compare it to the results returned by the function we implemented.

6. Rotate 270 (clockwise)

    a. We will call rotate 90 and rotate 180, respectively. This will add up to 270.

   b. Testing:
      i. Hard code a rotated 270 image and compare it to the results returned by the function we implemented.

7. -time:
   a. We will use cputiming.h implementations to measure how long the CPU took to rotate an image and individual pixel.
      i. CPUTime_Stop() will return the CPU time value we want.
   b. Make sure to CPUTime_Free()
   c. Testing:
      i. Use a small image and compare it to a large image. If implemented correctly, the large image should return a larger time.
      ii. If the same image 90 and 180 should be the same time, 270 will take about twice as much (we call 90 + 180 for 270 rotations).
      iii. If rotation is 0, time must be 0

Extra Functionalities:

8. -flip horizontal:
   a. We will set the new values to (h - i - 1, j). i = column, j = row. We will only be changing columns since we are flipping horizontally.
   b. Testing:
      i. Hard code a horizontally flipped image and compare it to the results returned by the function we implemented.
9. -flip vertical:
   a. We will set the new values to (i, w - j - 1). i = column, j = row. We will only be changing rows since we are flipping vertically.
   b. Testing:
      i. Hard code a vertically flipped image and compare it to the results returned by the function we implemented.
10. -transpose:
   a. We call -flip horizontal and -flip vertical.

b. Testing:

   i. Hard code a vertically + horizontally flipped image and compare it to the results returned by the function we implemented.

Part II:

	row-major access	column-major access	blocked access
90 degree	3	3	1
180 degree	2	3	1

**row-major and column-major are all tied.

**blocked is the fastest

Justification:

We are implementing our UArray2 in a single array instead of an array of arrays. Thus, accessing elements in the array, regardless of distance, should all be constant time.

- Therefore, for <u>small images that fit in the cache</u>, all access types are likely to have the same time.

We predict that blocked access is the fastest since data is stored in blocks in the cache. Instead of storing each pixel/index of the array in the cache, storing blocks of pixels allow us to be more efficient before the cache reaches max capacity.

Row-major and column-major should have same cache hit rate for 90 degree rotations. They both require the same usage of cache to store pixels, but do so in a different order. Assuming the cache will run out of memory as the spec says, both row-major and column-major will be accessing data outside of the cache. Row-major transforms rows into columns; the process will have a high cache hit rate in the original array, but low cache hit in the new array since it have to jump to a new row. Likewise, Column-major has low cache hit rate in accessing element of original image, but a high cache hit in new array. Since it turns columns into rows.

Row-major should be faster than column-major for 180 degree rotations. Row-major is converting rows into rows (first row becomes the last row, etc.), which benefits spatial locality. Therefore, in the cache we only really need to use one row of the old array, and one row of the new array. However, for column-major, we are still jumping to each row, so the cache hit would be higher than row-major 180 degrees.