Project 7 (C++): You are to implement the Hough Transform algorithm. You will create two Hough arrays, one uses Cartesian distance formula and the other uses Polar distance formula.

****

Language: C++

Project points: 10pts

Due Date: Soft copy (*.zip) and hard copies (*.pdf):

+1 11/10 early submission: 11/14/2022 Monday before midnight
10/10 on time: 11/17/2022 Thursday before midnight
-1 9/10 for 1 day late: 11/18/2022 Friday before midnight
-2 8/10 for 2 days late: 11/19/2022 Saturday before midnight
-10/10: 11/19/2022 Saturday after midnight
-5/10: does not pass compilation
0/10: program produces no output

*** Name your soft copy and hard copy files using the naming convention as given in the project submission requirement.

*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in **the same email attachments** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.

==

You will be given 5 test data img1, img2, img3, img4, img5: contains 1 point, 2 points, 3 points, three colinear lines and five colinear lines

What to do as follows:

1) Implement your program based on the specs given below.
2) Run and debug your program on img1 until you see 1 sinusoid in both Hough Space.
3) Run and debug your program on img2 until you see 2 sinusoids in both Hough Space.
4) Run and debug your program on img3 until you see 3 sinusoids in both Hough Space.
5) Run your program on img4, you should have multiple sinusoids what intersect at a point (or near-by) in both Hough Space.
6) Run your program on img5, you should have multiple sinusoids what intersect at a point (or near-by) in both Hough Space.

*** Include in your hard copies:

- cover page
- source code
- outFile1 from the results of 2) in the above.
- outFile1 from the results of 3) in the above.
- outFile1 from the results of 4) in the above.
- outFile1 from the results of 5) in the above.
- outFile1 from the results of 6) in the above.

****

I. inFile(argv[1]): a binary image with header

****

II. outFile1 (argv[2]): prettyPrint for both Hough arrays.

****
III. Data structure:
****

- A HoughTransform class
    - (int) numRows
    - (int) numCols
    - (int) minVal
    - (int) maxVal
    - (int) HoughDist // 2 times of the diagonal of the image
    - (int) HoughAngle // 180
    - (int**) imgAry // a 2D int array size of numRows by numCols; needs to dynamically allocate.
    - (int**) CartesianHoughAry //size of HoughDist by HoughAngle; needs to dynamically allocate.
    - (int**) PolarHoughAry //size of HoughDist by HoughAngle; needs to dynamically allocate.
    - (int) angleInDegree
    - (double) angleInRadians
    - (int) offSet // Given in class. See your lecture note.
  - methods:
    - constructor(...)
    - loadImage (…) // load imgAry from inFile
    - buildHoughSpace (...) // See algorithm steps below
    - (double) CartesianDist (…) // use the Cartesian distance formula given in class
    - (double) PolarDist (…) // use the Polar distance formula given in class
    - prettyPrint (…) // As in your previous projects

****
IV. main (…)
****

Step 0: inFile ← open inFile, outFile1 from argv
numRows, numCols, minVal, maxVal ← read from inFile
HoughAngle ← 180
HoughDist ←2 * (the diagonal of the input image)
imgAry ← dynamically allocate
CartesianHoughAry ← dynamically allocate and initialize to zero
PolarHoughAry ← dynamically allocate and initialize to zero
offSet ← // See your lecture note.

Step 1: loadImage (inFile)
prettyPrint (imgAry, outFile1)

Step 2: buildHoughSpace (...)

Step 3: prettyPrint (CartesianHoughAry, outFile1) // with caption indicate it is Cartesian Hough space
prettyPrint (PolarHoughAry, outFile1) // with caption indicate it is Polar Hough space

Step 4: close all files

****
IV. buildHoughSpace (...)
****

Step 1: scan imgAry left to right and top to bottom
Using x for rows and y for column

Step 2: if imgAry [x, y] > 0
computeSinusoid (x, y)

Step 4: repeat step 2 to step 3 until all pixels are processed

****

V. computeSinusoid (x, y)

****

Step 1: angleInDegree ← 0

Step 2: angleInRadians ← angleInDegree / 180.00 * pi

Step 3: dist ← CartesianDist (x, y, angleInRadians)

Step 4: distInt ← (int) dist // cast dist from double to int

Step 5: CartesianHoughAry[distInt][angleInDegree]++

Step 6: dist ← PolarDist (x, y, angleInRadians)

Step 7: distInt ← (int) dist // cast dist from double to int

Step 8: PolarHoughAry[distInt][angleInDegree]++

Step 9: angleInDegree ++

Step 10: repeat step 2 to Step 9 while angleInDegree <= 179

****

VI. CartesianDist (x, y, angleInRadians)

****

// Use the Cartesian distance formula given in class, see your lecture note.

// x & y need to convert to double in computation

// add offSet to the computation.

****

VII. PolarDist (x, y, angleInRadians)

****

// Use the polar distance formula given in class, see your lecture note.

// x & y need to convert to double in computation

// add offSet to the computation.