Aziza Mankenova

Assignment 3

1. **Problem description**

The given problem involves writing a Java program to make some calculations on 3D pixel arrays of PPM images. The program should work depending on the mode which is given as an input argument. Firstly, we are expected to write the exact 3D array to another ppm file. Secondly, we need to calculate a colour-channel based averagein order to write a black-and-white version of the input image as ppm. Third, we need to perform convolutionoperation using a 2D “filter” array on the input image and save the result as a specific ppm image. Lastly, we need to perform colour quantization by checking the values of the neighbouring pixels to see if they are within a given range and modify these pixels to be equal if they are in the same range. Then we need to write the quantized image to a certain ppm file.

1. **Problem Solution**

In order to write the program 10 static methods were used. Also, to write the code of the program while loops, for loops, recursion, if-else conditionals and switch conditionals were implemented.

The **pixelArray** method is used to write a PPM image file into a 3D array, using one string parameter, the name of the input file. It returns a 3-dimensional integer array. This method is convenient to call in different modes as it is used very frequently.

The **outputFile** method is used to create a new PPM file out of an array, using 3 parameters(name of the input file, 3-dimensional integer array and the name of the output file to be created**).** This method is universal, as it can create any file, out of any array and the name of the output file.

The **modeZero** method is used to read a PPM image file into a 3D array and then write the exact 3D array to another PPM file by calling the methods mentioned before(PixelArray and outputFile). It takes one string inputFile as a parameter.

The **modeOne** method is used to write a black-and-white version of the input image as a PPM image file by calculating a color-channel based average. It takes one string inputFile as a parameter. This method calls three methods(***pixelArray, blackWhite*, *outputFile).***

The **blackWhite** method is used to calculate a color-channel based average, which will give a black-and-white version of the input image**.** This method calculates average color values of each pixel using the three color channels (RGB). By assigning the calculated average value to all three color channels for all the pixels in the image, we obtain the black-and-white version of the image.

The **convolve** method is used to calculate the value of each pixel in the convolved array, using 5 parameters(i,j,k integers that represent the color-channel,row and column of the element of the array, and integer array and filterArray). It returns an integer value after taking the weighted sum of the pixel values.

The **modeTwo** is implemented to perform convolution operation using a 2D array (called “filter”) on the input image and write the result as another PPM image, taking two string parameters. It creates a new smaller array by applying convolution to each color channel of the image separately. And then calls **blackWhite** method in order to make it black-and-white.

The **modeThree** method is used to perform color quantization, taking a string inputFile and integer range parameters. It checks the values of the neighboring pixels to see if they are within a given range and modifies these pixels to be equal if they are in the same range. Then it will write the quantized image to another ppm file.

The **isSafe** method is implemented to check whether there is a neighbor with x,y,z coordinates and whether it was or was not changed before. It returns a boolean value.

The **quantization** method is implemented to check if any of the neighbors of certain pixel, (and also the neighbors of these neighbors, and so on...) are within the range of the pixel value with the element contained in x, y, z coordinate or not. If it is within the range, then all of these elements’ pixel values should be the same as the value in x, y, z. It takes 7 integer values, one integer and one boolean array.

1. **Implementation**

**package** assign3;

**import** java.io.\*;

**import** java.util.\*;

**public** **class** AM2018400387 {

**public** **static** **void** main(String[] args) **throws** FileNotFoundException {

**int** mode = Integer.*parseInt*(args[0]);//read the mode of the program to be executed from the element of the array at 0 index , and store it in the mode integer variable

String inputFile = args[1];//read the name of the input file from the element of the array at index 1

**if**(mode==0) {//if mode is zero

*modeZero*(inputFile);//call the method modeZero

}

**else** **if**(mode == 1){//if mode is one

*modeOne*(inputFile);//call the method modeOne

}

**else** **if**(mode == 2) { //if mode is two

String filter = args[2];//store the name of filter file in the filter variable

*modeTwo*(inputFile,filter);// call the method modeTwo

}

**else** **if**(mode == 3) {//if mode is three

**int** range = Integer.*parseInt*(args[2]);//store the range value in the integer range variable

*modeThree*(inputFile, range);// call the method modeThree

}

}

//this method is used for read a PPM image file into a 3D array, using one parameter, the name of the input file

**public** **static** **int**[][][] pixelArray(String inputFile) **throws** FileNotFoundException {

Scanner input = **new** Scanner(**new** File(inputFile));

input.nextLine();

**int** cols = input.nextInt();//declaring and initializing variable cols, which represents the number of columns

**int** rows = input.nextInt();//declaring and initializing variable rows, which represents the number of rows

input.nextInt();

**int** [][][] array = **new** **int** [rows][cols][3];//declaring a new 3-dimensional array

**int** i = 0;//declaring and initializing variable i, which represents rows

**int** j = 0;//declaring and initializing variable i, which represents columns

**while**(input.hasNextInt()) {//while loop, which executes while there is an integer input

**if**(j == cols) {//if j equals the number of columns we start filling the new row, starting from the first column

j=0;

i++;

}

array[i][j][0] = input.nextInt();//store the integer input in the Red channel, which is represented by 0

array[i][j][1] = input.nextInt();//store the integer input in the Green channel, which is represented by 1

array[i][j][2] = input.nextInt();//store the integer input in the Blue channel, which is represented by 2

j++;

}

input.close();

**return** array;//return the pixel array

}

//This method is used to create a new PPM file out of an array, using 3 parameters(name of the input file, 3-d array and the name of the output file to be created

**public** **static** **void** outputFile(String inputFile, **int** array[][][], String nameOutputFile) **throws** FileNotFoundException {

Scanner input = **new** Scanner(**new** File(inputFile));

String format = input.nextLine();

/\* cols = \*/input.nextInt();

/\* rows = \*/input.nextInt();

**int** max = input.nextInt();

input.close();

**int** cols = array.length;

**int** rows = array[0].length;

PrintStream output = **new** PrintStream(**new** File(nameOutputFile));//creating a new file named as the given parameter

output.println(format);//print the format

output.println(cols + " " + rows);//print the number of columns and rows

output.println(max);//print the max value

**for**( **int** i = 0; i < rows; i++) {

**for**(**int** j = 0; j < cols; j++) {

output.print(array[i][j][0] + " ");

output.print(array[i][j][1] + " ");

output.print(array[i][j][2] + " \t");

}

output.print("\n");

}

}

//This method is used to read a PPM image file into a 3D array and then write the exact 3D array to another PPM file

**public** **static** **void** modeZero(String inputFile) **throws** FileNotFoundException {

**int** [][][] array = *pixelArray*(inputFile);//reading a PPM image file into a 3D array

*outputFile*(inputFile, array, "output.ppm");//writing the exact 3D array to another PPM file called "output"

}

//This method is used to calculate a color-channel based average and write a black-and-white version of the input image as ppm.

**public** **static** **void** modeOne(String inputFile) **throws** FileNotFoundException {

**int** [][][] array = *pixelArray*(inputFile);//reading a PPM image file into a 3D array

*blackWhite*(array);//calling the blackWhite method

*outputFile*(inputFile, array, "black-and-white.ppm");//writing the 3D array to PPM file called "black-and-white.ppm"

}

//this method is used to calculate a color-channel based average, which will give a black-and-white version of the input image

**public** **static** **void** blackWhite(**int** [][][] array) {

**int** avg;//initializing a variable avg, which represents the color-channel based average

**for**(**int** i = 0; i < array.length; i++) {

**for**(**int** j = 0; j < array[i].length; j++) {

avg = (array[i][j][0] + array[i][j][1] + array[i][j][2]) / 3;//calculating average by summing up the values of each channel and dividing this value by three

array[i][j][0] = avg;//assigning the average value to each of these elements of an array

array[i][j][1] = avg;

array[i][j][2] = avg;

}

}

}

//This method is used to calculate the value of each pixel in the convolved array, using 5 parameters(i,j,k integers that represent the color-channel,row and column of the element of the array, and integer array and filterArray),

**public** **static** **int** convolve(**int** i, **int** j, **int** k, **int** [][][] array, **int** [][] filterArray) {

**int** size = filterArray.length;

**int** sum=0;

**int** startpoint=k;//initializing a temporary variable startpoint in order not to lose the value of k, which represents the column

**for**(**int** a=0;a<size;a++,j++) {

**for**(**int** b=0;b<size;b++,startpoint++) {

sum += filterArray[a][b]\*array[j][startpoint][i];//multiplying the corresponding elements of filterArray and array, and adding this value to the sum variable

}

startpoint = k;//changing startpoint value

}

**return** sum;//returning sum

}

//perform convolution operation using a 2D array (called “filter”) on the input image and write the result as another ppm image

**public** **static** **void** modeTwo(String inputFile, String filter) **throws** FileNotFoundException {

Scanner input = **new** Scanner(**new** File(filter));

String s = input.nextLine();

**int** size = Integer.*parseInt*(s.substring(0,1));//read the size of the filter array into an integer variable size

**int** [][] filterArray = **new** **int** [size][size];//declaring new integer array using the txt file of the filter

**for**(**int** i=0; i < size; i++) {

**for**(**int** j=0; j < size; j++) {

filterArray[i][j] = input.nextInt();//assigning every next integer to each element of the array

}

}

input.close();

**int** [][][] array = *pixelArray*(inputFile);//reading a PPM image file into a 3D array

**int** [][][] convolved = **new** **int** [array.length - size + 1][array[0].length - size + 1][3];//declaring a new integer convolved array of smaller size than the original array

**for**(**int** i=0; i<3; i++){

**for**(**int** j=0; j < convolved.length;j++) {

**for**(**int** k=0; k < convolved[0].length;k++) {

**if**(*convolve*(i,j,k, array, filterArray)<0) {//calling method convolve and checking the value returned. If the value is less than 0...

convolved[j][k][i] = 0;//assigning 0 to the element of convolved array

}

**else** **if**(*convolve*(i,j,k, array, filterArray)>255) {//calling method convolve and checking the value returned. If the value is greater than 255...

convolved[j][k][i] = 255;//assigning 255 to the element of convolved array

}

**else** {//if the value returned by convolve method is between 0 and 255...

convolved[j][k][i] = *convolve*(i, j, k, array, filterArray);//assigning this value returned to the element of convolved array

}

}

}

}

*blackWhite*(convolved);//calling bLackWhite method to turn the image into the black-and-white image

*outputFile*(inputFile, convolved, "convolution.ppm");////writing the 3D convolved array to PPM file called "convolution.ppm"

}

//This method is used to perform color quantization. It checks the values of the neighboring pixels to see if they are within a given range and

//modifies these pixels to be equal if they are in the same range. Then it will write the quantized image to another ppm file.

**public** **static** **void** modeThree(String inputFile, **int** range) **throws** FileNotFoundException {

**int** [][][] array = *pixelArray*(inputFile);///reading a PPM image file into a 3D array

**boolean** [][][] array2 = **new** **boolean** [array.length][array[0].length][3];//declaring a boolean array2 of the same size as pixel array

**for**(**int** k=0; k<3; k++) {

**for**(**int** i=0;i<array.length; i++) {

**for**(**int** j=0; j<array[0].length; j++) {

*quantization*(i, j, k, array, array2, range, array[i][j][k]);//calling quantization method to check all of the 6 neighbors of each element of the pixel array

}

}

}

*outputFile*(inputFile, array, "quantized.ppm");////writing the 3D array to PPM file called "quantized.ppm"

}

//This method is implemented to check whether there is a neighbor with x,y,z coordinates and whether it was not changed

**public** **static** **boolean** isSafe(**int** x, **int** y, **int** z, **int** [][][] array, **boolean** [][][] array2) {

**int** length = array.length;

**return** (x >= 0 && x < length && y >= 0 && y < length && z >= 0 && z < 3 && array2[x][y][z]==**false**);

}

//this method is implemented to check if any of the neighbors of certain pixel, (and also the neighbors of these neighbors, and so on...) are within the range of the pixel value with

//the element contained in x, y, z coordinate,then all of these elements’ pixel values should be the same as the value in x, y, z.

**public** **static** **void** quantization(**int** x, **int** y, **int** z, **int** [][][] array, **boolean** [][][] array2, **int** range, **int** n) {

**if**(array[x][y][z]<n-range || array[x][y][z]>n+range) {//if the value is out of range it should ignore this neighbor

**return**;

}

**else** {//if the neighbor is in the range

array[x][y][z] = n;//changes the value of the neighbor to the value of initial pixel

array2[x][y][z]=**true**;//change the value of the neighbor in the boolean array, so that it would not be changed later

**if**(*isSafe*(x+1, y, z, array, array2)) {//if it can move down, which means it can increase a row number

*quantization*(x+1, y, z, array, array2, range, array[x][y][z]);//call quantization method again on the neighbor which is below that pixel

}

**if**(*isSafe*(x-1, y, z, array, array2)) {//if it can move up, which means it can decrease a row number

*quantization*(x-1, y, z, array, array2, range, array[x][y][z]);//call quantization method again on the neighbor which is above that pixel

}

**if**(*isSafe*(x, y+1, z, array, array2)) {//if it can move to the right, which means it can increase a column number

*quantization*(x, y+1, z, array, array2, range, array[x][y][z]);//call quantization method again on the neighbor which is on the right of the pixel

}

**if**(*isSafe*(x, y-1, z, array, array2)) {//if it can move to the left, which means it can decrease a column number

*quantization*(x, y-1, z, array, array2, range, array[x][y][z]);//call quantization method again on the neighbor which is on the left of the pixel

}

**if**(*isSafe*(x, y, z+1, array, array2)) {//if it can increase z value, which is a color-channel value

*quantization*(x, y, z+1, array, array2, range, array[x][y][z]);//call quantization method again on the neighbor which is in color channel of greater value

}

**if**(*isSafe*(x, y, z-1, array, array2)) {//if it can decrease z value, which is a color-channel value

*quantization*(x, y, z-1, array, array2, range, array[x][y][z]);//call quantization method again on the neighbor which is in color channel of lesser value

}

}

}

}

1. **Output of the program**
2. Input arguments:0 input.ppm

Изображение выглядит как одежда, женщина, человек, нарядный головной убор

Автоматически созданное описание

1. Input arguments: 1 input.ppm

Изображение выглядит как человек, женщина, одежда, фотография

Автоматически созданное описание

1. Input arguments: 2 input.ppm filter(horizontal).txt

Изображение выглядит как внешний, человек

Автоматически созданное описание

1. Input arguments: 3 input.ppm 25

Изображение выглядит как текст

Автоматически созданное описание

1. **Conclusion**

The program functions and gives the output it was expected to give. The program can perform all the operations that were asked. The program can turn write the PPM image file into another PPM file. It can turn the given image into a black-and white version of it. It can perform convolution operation using a 2D filter. It can perform color quantization. Overall, the given problem was solved.