Programming assignment # 4

In this programming assignment, you will implement an image warping transformation by manipulating the pixels that make up a digital image.

1 Requirement Specification

Your program shall take three command-line arguments: a String, representing the name of a file containing a JPEG image, and two integers, representing the \((x, y)\) coordinates of a pixel in the image. Your program will manipulate the pixel array for the image (according to the algorithm described in the next section), and write the result in a file whose name is formed by prefixing the first command-line argument with the following string: “warped.”.

Your program shall consists of a package named assign4 that defines a class called Main, a class called ImageTransform, and possibly other auxiliary classes as you see fit. A sample invocation of your program should look like:

% java assign4.Main checkers.jpg 90 45

If “checkers.jpg” contains the image on the left below, then your program should create a file (named “warped_checkers.jpg”) that contain an image like the one on the right:

To get you started with the program, we provide you with a (functionally complete) implementation of the Main class, and with a partial implementation of the ImageTransform class at:

http://www.cs.stevens.edu/~nicolosi/classes/17fa-cs181/assign4/Main.java
http://www.cs.stevens.edu/~nicolosi/classes/17fa-cs181/assign4/ImageTransform.java

Your main task is to provide the implementation of the following two methods:

```java
public int[] transform();
protected void warpRegion(int[] dstPixels, Rectangle r, Point nw, Point ne, Point sw, Point se);
```

As extra credit 1, additionally re-implement the getPixel(double, double) method to do “color interpolation” (cf. comments for getPixel(double, double)).

As extra credit 2, change your program so that it may also accept five arguments on the command line, where the additional two values are interpreted as the \((x, y)\)-coordinate of a second point that controls the partitioning of the image into quadrants (cf. comments in transform()).
2 The Algorithm: Basic idea

The idea behind image warping is to distort an image by transforming quadrilateral regions into rectangular regions.

To apply the transformation to a given image, the first step is to define the relevant quadrilaterals and the corresponding rectangles. The figure above depicts these. $RTZX$ denotes the pixel region to be warped. (In this assignment, this region will correspond to the entire image, but the transformation also makes sense if $RTZX$ does not cover the whole image area.) The point $V$ determines how $RTZX$ is partitioned into four sub-rectangles, namely $RSVU$, $STWV$, $VWZY$, and $UVYX$. By default, we will assume that the partitioning point $V$ is the midpoint of $RTZX$. (Extra-credit 2 asks you to set the partitioning point based on values from the command line.)

The warping is also controlled by a point $V'$ inside the $RTZX$ rectangle. The control point $V'$ partitions $RTZX$ into four quadrilaterals: $RSV'U$, $STWV'$, $V'WZY$, and $UV'YX$. Intuitively, the essence of warping is to stretch things so that $V'$ gets “moved” onto $V$.

The picture below illustrates how this setup decomposes as four quadrilateral-to-rectangle mappings. The direction of the mappings is represented by the arrow. In detail, these mappings are: $RSV'U \rightarrow RSVU$, $STWV' \rightarrow STWV$, $V'WZY \leftrightarrow VWZY$, and $UV'YX \leftrightarrow UVYX$.

To summarize, to warp the pixel region $RTZX$, we simply warp each of the four sub-regions.

---

1The warping transformation can be generalized so that it makes sense also when the control point $V'$ is outside of $RTZX$, but we will not need that for this assignment.
3 The Algorithm: The details

Your code for the warpRegion(...) method should implement the quadrilateral-to-rectangle transformation. Suppose we want to perform the warping transformation $V'WZY \rightarrow VWZY$ (all other cases are analogous). Then, we must map each pixel of $V'WZY$ into a corresponding pixel of $VWZY$. But how do we iterate over each pixel of $V'WZY$? This seems like a difficult task, since image pixels are typically positioned on a grid, whereas typically two sides of $V'WZY$ will be slanted. Taking the path of least resistance, we will iterate instead over the pixels in $VWZY$, computing for each of them the corresponding pixel in $V'WZY$ and then copying its RGB-value.

The next subsection derives this mapping, which aims at expressing a pixel in the destination shape (the rectangle) in terms of a (fractionary) pixel in the origin shape (the quadrilateral). You might want to skip it upon a first reading, and return to this after seeing the following subsection about how to use the mapping in your code.

3.1 Deriving the mapping

For completeness, we consider the general case of a quadrilateral-to-rectangle mapping, where all the corners of the quadrilateral might be distinct from those in the rectangle.

Given a pixel $P = (x, y)$ inside $DCBA$, what should the location of the corresponding pixel inside $D'C'B'A'$ be? Let it be $P' = (x', y')$, and let us compute its coordinate using linear interpolation. First, the easy cases. If the point happens to be $D$, $C$, $B$, or $A$, then the corresponding point is $D'$, $C'$, $B'$, or $A'$, respectively. If the point happens to lie anywhere directly on the $DC$ segment (denoted as $R$ in the diagram), then the corresponding point $R'$ must lie directly on the $D'C'$ segment. Using linear interpolation (e.g., see http://en.wikipedia.org/wiki/Linear_interpolation), we obtain:

$$R' = D' + (C' - D') \frac{|RD|}{|CD|}$$

where $|\cdot|$ denotes the length of the given segment. Note that the above equation is stated in terms of points rather than coordinates. This is just a short-hand notation for a pair of equations—if $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, then the point equation $P_1 = P_2$ is equivalent to the pair of equations $x_1 = x_2$ and $y_1 = y_2$.

Similarly, if the point happens to lie anywhere directly on the segment $AB$ (denoted as $Q$ in the diagram), then the corresponding point $Q'$ can be identified as:

$$Q' = A' + (B' - A') \frac{|QA|}{|BA|}$$
Now for the generic case. If \( P \) does not lie directly on either \( \overline{DC} \) or \( \overline{AB} \), then let \( R \) and \( Q \) be the projections of \( P \) onto \( \overline{DC} \), and \( \overline{AB} \), respectively. Using the above two equations, we first compute the corresponding \( R' \) and \( Q' \) inside \( \overline{D'C'B'A'} \). Next, we perform a third linear interpolation along the \( \overline{R'Q'} \) segment to obtain \( P' \):

\[
P' = R' + \frac{(Q' - R')|PR|}{|QR|}
\]

The latter equation can then be simplified slightly observing that \( |RD| = |QA| \), \( |CD| = |BA| \), and \( |QR| = |CB| \), since \( DCBA \) is a rectangle. Letting \( \alpha = \frac{|RD|}{|CD|} \) and \( \beta = \frac{|PR|}{|CB|} \), the above equations for \( R' \), \( Q' \) and \( P' \) can then be rewritten as:

\[
R' = D' + (C' - D')\alpha \\
Q' = A' + (B' - A')\alpha \\
P' = R' + (Q' - R')\beta
\]

Since \( |RD| \leq |CD| \) and \( |PR| \leq |CB| \), we have that \( \alpha, \beta \in [0, 1] \). Furthermore, \( \alpha \) and \( \beta \) are the only quantities in the above equations that change with \( P \)—everything else is independent of the specific point \( P \) at hand.

### 3.2 Using the mapping

Substituting Equation (1) and Equation (2) into Equation (3), we get:

\[
P' = D' + (C' - D')\alpha + (A' - D')\beta + (D' - C' + B' - A')\alpha\beta
\]

Now to map each pixel from \( \overline{D'C'B'A'} \) into \( DCBA \), iterate over each pixel \( P \) in \( DCBA \), compute \( P' \) and store the color value of the pixel at \( P' \) at the location \( P \). If we let \( c \) run over pixel columns and \( r \) run over pixel rows, then \( \alpha = c/n \), and \( \beta = r/m \), where \( n, m \) are the number of columns and rows, respectively. Therefore, the color of pixel \( P = (D.x + c, D.y + r) \) should be set to the color of pixel \( (P'.x, P'.y) \) as given by the above formula. (Again, recall that an equation over points really just amounts to two equations—one in the \( x \)'s and one in the \( y \)'s.)