Program 1

Due Oct 7, 2019 by 1:15pm **Points** 20 **Submitting** a file upload **File Types** zip

First:

Post an introduction on the course message board in the “Introductions” thread, if you haven't already

Goals

The goals of this assignment are to familiarize you with the image library that we will be using in this class and to use linear algebra to transform an image according to parameters input from a batch file. Note that you will need to download and use the library and header files from this page in order to use the code I am supplying. You should not modify any of the supplied code.

Background

A general linear transformation of a two-dimensional image has six parameters. Let’s work in the x-y plane (rather than row-column). Each point (p_x, p_y) *could* be transformed according to the parameters (a, b, c, d, e, f) as follows:

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}.$$

In this assignment, we will use more intuitive variables that have the same range of generality. Our parameters will be two scale factors (s_x and s_y), a translation in x and y (t_x and t_y), a rotation angle in degrees (θ), and a shear factor (k). Given these values, we can write a linear transformation as follows:

$$\begin{bmatrix} q_x \\ q_y \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Or, more simply:

$$q = SKRp + t,$$

where:

$$q = \begin{bmatrix} q_x \\ q_y \end{bmatrix}, \; S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}, \; K = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}, \; R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \text{ and } t = \begin{bmatrix} t_x \\ t_y \end{bmatrix}.$$

Program

You are to write a program that takes these six parameters (s_x, s_y, t_x, t_y, θ, and k) as input from the batch file (in that order) and transforms an image according to the parameters. The output image should have the same dimensions as the input image. Any point in the output image that doesn't have a corresponding point in the input image should be black. You should assume that the input image is named "test.gif" and is located in the same directory as your code. You should write your output to "output.gif" and place it in the same directory as your code.

The correct way to accomplish this is compute the inverse transformation for each point in the output image:

$$p = R^{-1}K^{-1}S^{-1}(q - t)$$

This point can then be looked up in the input image to determine the correct pixel color. Note that this point may be outside of the input image, in which case you should leave the output pixel black. It makes the most sense to perform the rotation, scale, and shear with respect to the center of the image rather than the pixel at (0, 0). This can be achieved by subtracting the center of the image from **q** and adding it back in at the end. If **c** is the center of the image (cols / 2, rows / 2), then the following will yield the correct results:

$$p = R^{-1}K^{-1}S^{-1}(q - t - c) + c$$

An additional complication is that the result of the above computation may leave you with a pixel that does not have an integer index (for example, row 101.3 and column 80.7). You could select the nearest neighbor or round down; however, full credit will only be given if you use bilinear interpolation (see below and/or linear algebra notes). After performing bilinear interpolation, round all floating-point color values down (you can use static_cast<byte>).

You can find a test image and sample results given some input parameters below. Your code must compile with Microsoft Visual C++ 2017/2019. Submit your completed code (source and header files only) using Canvas by the start of class on the due date. These must be submitted as a single .zip file. See the **program grading rubric** for notes on how I grade programs.

Important notes

- When you create the project, make it an "Empty project" (don't use precompiled headers, etc.)
- For this assignment, use the column as the x-component and the row as the y-component. (Ignore the change in axis directions that would otherwise occur for the y-axis.)
- Your code should run with the batch file linked below. Some of your code might look like this for example:

```
int main(int argc, char *argv[]) {
	double x_scale;
	sscanf_s(argv[1], "%lf", &x_scale);
```

```
        ...
}
```

- Bilinear interpolation is computed as follows. Let's say we want to interpolate the color at:

$$p = \begin{bmatrix} p_c \\ p_r \end{bmatrix}$$

  Let c = $\lfloor p_c \rfloor$ and r = $\lfloor p_r \rfloor$ (this notation denotes the maximum integer that is not greater than p.

  Also, let $\alpha = p_r - r$ and $\beta = p_c - c$.

  The bilinear interpolation of the image intensities (red, green, blue) at p = (p_c, p_r) are given by the following equation:

$$(1-\alpha)(1-\beta)I(c,r) + \alpha(1-\beta)I(c,r+1) + (1-\alpha)\beta I(c+1,r) + \alpha\beta I(c+1,r+1)$$

- In the bilinear interpolation step, if you need to interpolate (with a weight greater than zero) using any pixel outside of the input image, then set the output pixel to black. We will treat these locations as not computable.
- You do not need to write a matrix or vector class or a matrix inversion method for this program (although you can). The simplest implementation is to compute the inverse of each individual component (S, K, R) and apply them one-by-one (in sequence) to each pixel in the output image to determine the appropriate location in the input image.

Resources

You will need the following code to complete the assignment. The Image.lib will work with VS 2017 or VS 2019. It was compiled for debug mode: **Image.h** (https://canvas.uw.edu/courses/1332532/files/58262691/download?download_frd=1) , **Image.lib** (https://canvas.uw.edu/courses/1332532/files/58262684/download?download_frd=1) . (Place these files in the same directory as your code and add them to the project.) Here is a **test image** (https://canvas.uw.edu/courses/1332532/files/58262709/download?download_frd=1) .

Your code should run with this **batch file** (https://canvas.uw.edu/courses/1332532/files/58262690/download?download_frd=1) in the same directory as your code (it generates the combination image below).

Here are some sample results from my implementation:

- **Rotation** (https://canvas.uw.edu/courses/1332532/files/58262687/download?download_frd=1) by 20 degrees
- **Scale** (https://canvas.uw.edu/courses/1332532/files/58262685/download?download_frd=1) by 1.5 in both x and y

- **Translation** (https://canvas.uw.edu/courses/1332532/files/58262688/download?download_frd=1) by 20 in x and 40 in y
- **Shear** (https://canvas.uw.edu/courses/1332532/files/58262689/download?download_frd=1) by 0.5
- **Combination** (https://canvas.uw.edu/courses/1332532/files/58262686/download?download_frd=1) of all of the above transformations

Here is an alternative version of the **simple program from class** (https://canvas.uw.edu/courses/1332532/files/58617518/download?download_frd=1)

Minimize File Preview

ZOOM

```
            }
        }

        return output;
}


// main method
// Preconditions:  test.gif exists and is a correctly formatted GIF image
// Postconditions: Creates an image output.gif that is the rotation and
photonegative of test.gif
int main()
{
        Image input("test1.gif");
        input = input.photonegative();

        Image output = rotate(input);

        output.writeImage("output.gif");
        return 0;
}
```

to modify an image.