

## Project Description

This week you will build a warping tool based on matrix warps, which includes the collection of affinewarps. The tool will provide for input of a sequence of operations, will compose a single transform from the sequence, and finally will apply the transform to the input image via inverse mapping.

You will have to write procedures for constructing the matrices each of the following transformations: translate, rotate, scale, and shear. You will also have to write routines to warp an input image into the new shape, and then do the inverse mapping of the input image to the output image via the inverse of the transformation.

You have been provided routines to do  $3 \times 3$  matrix multiplication, transform of a 2D point by a  $3 \times 3$  homogeneous matrix, and inverse of a  $3 \times 3$  matrix, and a few others. You may use our routines or write your own if you wish. I have also provided routines for processing the sequence of input commands for the transform specification. This is a sample routine, your job will be to fill in the code that accumulates the transformation for each of the user's inputs (it shows one examples with rotate operations). Likely, the best option will be to remove this code from the example and merge it with your code that uses OIIO and OpenGL for image display.

## Basic Requirements

The program you write should be called **warper**, and will consist of 2 parts:

### Part 1:

- read an input image from an image file
- input a sequence of matrix specifications
- multiply these matrices as you go to get a final transform matrix from all of the specified matrices
- transform just the corners of the image with the forward transform
- make space for a new image exactly big enough to contain the four transformed/warped corners.

### Part 2:

- calculate the inverse transform by inverting the final transformation matrix
- transform the image with the inverse matrix, truncating non-integer pixel locations to integers and accounting for if the origin has moved.
- optionally (if a second filename was specified) save the transformed image in an image file.
- display the transformed image

The command line must be of this form:

```
warper inimage.png [outimage.png]
```

If the output filename is missing, the program simply skips saving the output image. If no filenames are present, the program prints an error message and stops.

The specifications for each transform follow. Your program should loop, reading these until the `d` command is input). *Italics* indicate floating point arguments:

<b>r</b>	<i><math>\theta</math></i>	counter clockwise rotation about image origin, $\theta$ in degrees
<b>s</b>	<i>sx sy</i>	scale (watch out for scale by 0!)
<b>t</b>	<i>dx dy</i>	translate
<b>h</b>	<i>hx hy</i>	shear
<b>d</b>		done

For grading, your program will be tested using a script, so please adhere to the standard interface. If your program has any optional switches or features: 1) make sure that they do not interfere with the standard operation of the program (i.e. if the switches are not present the program should run exactly as specified above and below). If you want to give your program a graphical user interface, please turn in two versions of the program, one with the standard interface, and one with the graphical interface, and make sure you include a README file explaining what you have done.

**Code Documentation** Please see Programming Assignment 01 for the expected requirements. Your code will be evaluated based on code structure, commenting, and the inclusion of a README and build script.

### Advanced Extension for 6040 students (worth 3 points)

Modify your executable to also accept a new input type:

<b>n</b>	<i>s cx cy</i>	nonlinear twirl warp
----------	----------------	----------------------

This mode runs your program using nonlinear warping “twirl” functions from Lecture 18. This code should correctly have passed as input a center position (*cx, cy*) and a strength of the warp *s*. In this mode, you ignore the matrices of specified by all previous warps, and set up only a single warping function to compute the twirl.

Note, you will have to extend the input processing function to pass parameters appropriately. These should set a state variable that performs a warp without the matrix. Also, you need not use the Matrix and Vector classes, but you might benefit from using the Vector operations to compute the distance  $r$  and the angle  $\theta$  for each pixel.

Because of the nonlinearity, you can assume that the output image is of the same size as the input image. For each pixel, you will apply the inverse map using the twirl functions. Depending on parameters, some of the  $(x, y)$  values will invert to positions  $(u, v)$  that fall outside of the input image.

### Submission

*(Please read all of these instructions carefully.)*

Please write the program in C or C++, and I recommend using OpenGL and GLUT graphics routines for the display. Use OIIO for image input and output. Please take extra care to make sure that your homework compiles on the School of Computing’s Ubuntu Linux cluster—testing on your own home machine, even if it runs Ubuntu, may not be sufficient. Remember:

both a working build script and README must be provided

Consequently, to receive *any credit whatsoever* this code must compile on the cluster, even if it does not accomplish all of the tasks of the assignment. The grader will give a zero to any code that does not compile.

Submit using the handin procedure outline at <https://handin.cs.clemson.edu/>. You are welcome to use the commandline interface, but the web interface is sufficient. The assignment number is **pa06**.

Finally, since we are using multiple files, please only submit a single file which has aggregated everything. This file should be named `[username]_pa06.tgz` where `[username]` is your Clemson id (please remove the brackets []). For example, mine would be `levinej_pa06.tgz`. Please make sure your build script is at the top level directory.

## Rubric for Grading

4040 students will be graded for the following requirements:

### I. +2 Reading, Writing, and Display

Code correctly reads in input image, displays the warped image after 'd' is input signalling the warp is fully specified, and then writes out the warped image if a second filename is specified.

### II. +3 Warping Basics

Code correctly supports the user interface for entering commands. The warped image dimensions are correctly computed. Code correctly computes the aggregated matrix and makes use of the inverse map to fill in color values for the warped image.

### III. +3 Specific Warping Types

Rotation, scaling, translation, and shearing warps are correctly implemented.

### IV. +2 Clarity

Does the code have good structure? Is the code commented and is a README provided? Is a working build script provided?

6040 students will receive 7 points for completing the above requirements, scaling the above by 70%. To achieve 10 points they must also complete:

### I. Advanced Extensions

Code correctly reads in the three parameters for the swirl warp. Code correctly checks the  $u$  and  $v$  values to ensure they are in the range  $0 < u, v < 1$  (some will fall outside). Using these equations, the code correctly computes a swirl warp and displays the warped image.

Going above and beyond these requirements may result in extra credit at the discretion of the instructor and grader. Please note any extra features you implement in the README.