CPSC 4040/6040
Computer Graphics Images

Assigned: Oct 23, 2019
Due: 11:59pm, Sun, Nov 3, 2019

Programming Assignment 5 — Image Warping

(Grading: 0–10 points)

Project Description

This week you will build a warping tool based on affine and projective warps. The tool will provide as input 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 for each of the following transformations: translate, rotate, scale, shear, and perspective warp. 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. In the provided code, you can find routines to perform 3×3 matrix multiplications, multiply a 2D/3D point by a 3×3 matrix, compute the inverse of a 3×3 matrix, and a few others. You may use these routines or write your own if you wish. I have also provided code for processing the sequence of input commands for the transform specification. This is a sample code, your job will be to fill in the code that accumulates the transformation for each of the user's inputs. The best option will be to merge this code 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

Part 2:
- transform the four corners of the image with the forward transform
- make space for a new image exactly big enough to contain the four transformed/warped corners.
- calculate the inverse transform by inverting the final transformation matrix
- warp the image with the inverse matrix (do not forget to normalize), 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 exits.

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

`r`	θ	counter clockwise rotation about image origin, θ in degrees
`s`	*sx sy*	scale (watch out for scale by 0!)
`t`	*dx dy*	translate
`h`	*hx hy*	shear
`f`	fx fy	flip - if fx = 1 flip horizontally, fy = 1 flip vertically
`p`	*px py*	perspective warp
`d`		done

To showcase your program, you should include a *pdf* that shows at least four examples of original and warped images (you can, e.g., use test images from homework 4). As always a README file should be included with instructions, known issues, etc.

Advanced Extension

Implement any of the following:

1. Add a -b (bilinear) switch to the command line that implements a bilinear warp instead of a perspective warp (note some helpful code is in the provided matrix class);

2. Add a -i (interactive) switch to the command line and provide the ability to have the user interactively position the corners of the output warped quadrilateral via the mouse. You are basically performing a projective warp.

3. Support a twirl warp
   `n` *cx cy s* nonlinear twirl warp
   The code should correctly pass as input a center position (cx, cy), and the strength, s, of the warp. Here, tor each pixel, you will apply the inverse map using the twirl functions shown in class.

4. Support a ripple warp
   `m` tx ty *ax ay* nonlinear ripple warp
   The code should correctly pass as input the period length (in pixels) (tx, ty), and the corresponding amplitude values (ax, ay) for the displacement in the x and y direction, respectively. Here, for each pixel, you will apply the inverse map using the ripple functions shown in class.

   In both 3 and 4, you should ignore the matrices specified by all previous warps, and set up only a single warping function to compute the twirl or ripple. Also, you do not need to use the Matrix and Vector classes. Because of the nonlinearity, you can assume that the output image is of the same size as the input image. Depending on parameters, though, some of the (x, y) values will invert to positions (u, v) that may fall outside of the input image.

Submission

Submit using the Canvas system. Along with your code, please include a *pdf page* that shows at least four examples of original and warped images. As always, some basic code documentation is required. If your program has any optional switches or features, make sure to document them properly in the README.