# Warping and Morphing

## CS 4475: Project 5

## 1 - Objective

For this project, you will write the code for an image warp, based on point correspondences, using radial basis functions. Your code will use bilinear interpolation to create a high quality result. In addition, you will take a photograph and warp it using this code. Finally, you will produce a morph between a photo of your face and a photo of one of your classmates.

## 2 - Deadline

**Your project should be submitted on T-Square by 11:55PM on Monday, June 23, 2014.**

## 3 - Process

### 3.1 Download the base source

Download and unzip both of the provided zip files for this project. The warping code that you write and turn in will be based on warp_template. The face morphing will be performed using the code in the "morphing" folder. You will not be modifying the morphing code, just using it.

### 3.2 Project description

The most important part of this project is to add your warping code to the provided code template for warping. The code you write should be based on the pseudo-code for warping that Dr. Turk presented in class on Monday, June 16. All of your warping code should be placed in the "perform_warp()" function in the warp.pde file. The skeleton code in perform_warp() currently does two things. First, it prints out the x,y -> x,y point correspondences that the user has created, and this is to show you how to access those point pairs (which you will use for the warp). Second, there is dummy code for going through all of the pixels of the target image and copying the corresponding pixels from the source image. You will replace these lines with code that actually performs a 2D warp. As mentioned in class, you will also use bilinear interpolation instead of nearest neighbor lookup when you get the color out of the source image.

The second part of the assignment is for you to take a photograph and warp it using your warp code. If you want to you a picture of a person, make sure that you get permission to use their picture for this project. You will turn in the original photo and the warped version.

The third part of the assignmenet is to morph your face (using the provided morphing code) into the face of a classmate. Both of these photos will be given to you. Since we are going to "chain" these morphs together, you need to stick to your assigned target classmate. You will turn in the polys.txt file that specifies the line segment correspondences for your morph.

### 3.3 Suggested Approach

First, familiarize yourself with the use of the template code for warping. Learn how to place points, move points, and delete points. There is a README.txt file along with the code that explains the GUI for this program.

Next, look at the dummy version of perform_warp() in warp.pde. Notice how you can access the corresponding points, as shown in the first loop. Begin to translate the warping pseudo-code into working code in this routine. Do **not** write the bilinear interpolation code yet – just grab the color of the closest pixel from your ps value in the source image. Test out your code first with just a **single pair of points**, one in the source and one in the target. When warping with just a single pair of points, the result should be a translation of the image, without warping. If

the left point "bi" is on the tip of the nose in the source image, then the target image should have the tip of the nose at the position of the point "ci". You should be able to print out the value of your displacement vector (bi – ci), and see if the numbers make sense. You should also be able to print out the weight value (w), the sum of the weights (wsum), and the sum of the displacements times the weights (vsum). When you divide vsum by wsum, the answer (when using a single point pair) should be the same as your original displacement vector. When printing out such values, do **not** print them out for each pixel – this is too much information. Just print them out for a single pixel such as (i = 100, j = 100) using an if-statement. Once you have debugged warping with a single pair of points, try out multiple point correspondences.

Once your warping is working properly, begin working on bilinear interpolation. You should feel free to use the built-in processing function lerpColor() for this. Debug this by stretching out a portion of an image, by having four points in a small square in the source, and moving those four points farther apart in the target. When using nearest neighbor, this will show blocky pixels, but when using bilinear interpolation this should look more smooth.

When your warping code is fully functional, modify the code to read in an image that you took, instead of Patrick Stewart. Warp your image, and keep the result to turn in.

Finally, create a morph between a photo of you and the assigned photo of a classmate. To do this, you will need to learn how to use the provided morphing program. The README.txt file provided with this code will explain what you need to do to get started. It is important that you read this so that you know how to modify the window sizes to fit your laptop screen. (Do **not** modify the sizes of the face photos or crop them – these need to be kept at their original size in order to make a chain of morphs through everyone in the class.) The GUI is a little bit more involved for morphing based on line segments, so the README.txt file will help here. Familiarizing yourself with all of the mouse and keyboard options will help you in making the morph. When you have a morph that you like, be sure to press the "w" key before you quit the program. This saves the line segment correspondences in a file called "polys.txt". If you want to make several variations of the morph, you should make a copy of polys.txt, so you don't overwrite your previous polys.txt.

### 3.4 Source code

You should modify the source code in any way you see fit and comment your code (include your name in the header). The source code is written in Processing. Visit "Processing.org/reference/" for more information on built in functions and structure. You are allowed to use any built in processing/openGL functions in this project to accomplish your goal.

### 3.5 Authorship Rules

The code that you turn in entirely your own. You are allowed to talk to other members of the class and to the Professor and the TA about general implementation of the assignment. It is also fine to seek the help of others for general Processing/Java programming questions. You may not, however, use code that anyone other than yourself has written. Code that is explicitly not allowed includes code taken from the Web, from books, from previous assignments or from any source other than yourself. You should not show your code to other students. Feel free to seek the help of the Professor and the TA's for suggestions about debugging your code.

### 3.6 Submission

In order to run the source code, it must be in a folder named after the main file. When submitting any assignment, leave it in this folder, compress it and submit via T-square. There are three items that you will turn in for this assignment: 1) a working version of your warping code (based on the provided template), 2) an image pair that consists of a photograph that you took, along with a warped version of that same photograph, and 3) the polys.txt file that specifies the line segment correspondences for morphing between your face and your classmate's face.