CS 435 - Computational Photography

Final Project - Morphology

YOU MAY WORK WITH A PARTNER IF YOU LIKE!!!
But if you do so, look at the additional information you need to provide in your submission (stated at the end of the document).

Introduction

For our final assignment, we'll attack the problem of morphing from one photo to another. This will require several ideas from this course, including:

- Least Squares Estimate (LSE) for Transformation Matrix Discovery
- Blending

Grading

Scan-filling	10pts
Transforming	10pts
Cross-Dissolve	15pts
Visualizing Points and Triangles	15pts
Morphing	30pts
Additional Tests	20pts
TOTAL	100pts

Table 1: Grading Rubric

The Dataset

For this project you'll need to obtain *two sets of image pairs* that you want to morph together (so, four total images). Each set should be in their own folder with the file names *img1.[ext]*, *img2.[ext]*, where [ext] is the files' extension. The image pairs should be the same size, and the resolution should be reasonable enough for computation time. Crop and/or downsample your images first "offline", if needed.

In addition, for each pair you'll need to (manually) identify a set of matching locations in each. These locations should be placed in a CSV-style plain text file, *correspondences.csv*, in the associated directory. The format of this file is:

```
img1_pt1_x, img1_pt1_y, img2_pt1_x, img2_pt1_y
img1_pt2_x, img1_pt2_y, img2_pt2_x, img2_pt2_y
.....
img1_ptN_x, img1_ptN_y, img2_ptN_x, img2_ptN_y
```

Recall that our image coordinate system starts at the top-left of the image, with the x-axis to the right, and the y-axis down.

1 (10 points) Scan-fill a Triangle

For this project we will need to be able to iterate over all the pixels in a triangle. This is akin to *scan-filling* a triangle a triangle in computer graphics. There are many algorithms out there to do this, with greatly varying efficiency. For this project we'll just use a (relatively) straightforward algorithm.

Pseudocode for this algorithm is as follows:

Given: The three vertices of a triangle.

1. First, determine the max and min y-value of your vertices.
2. For $y = miny : maxy$
   (a) For each line segment making up the triangle
      i. Compute the x-value for the *line* using the current value of y. From the point-slope form of a line this can be computed as

$$x = \frac{y - y_1}{m} + x_1$$

      where m is the slope of the line and (x_1, y_1) are a point on it.
      ii. Check to see if this location is within the bounds of the *line segment* pertaining to this line.
   (b) Grab the min and max valid x-value from the set of valid x computed x values.
   (c) For $x = floor(minx) : ceil(max)$
      i. Fill pixel (x, y).

Write code that, given a triangle with vertices $(20, 30), (100, 50), (50, 80)$, scan-fills this triangle by plotting a point at each location within the triangle on a figure. Include this figure in your report. It should look something like Figure 1

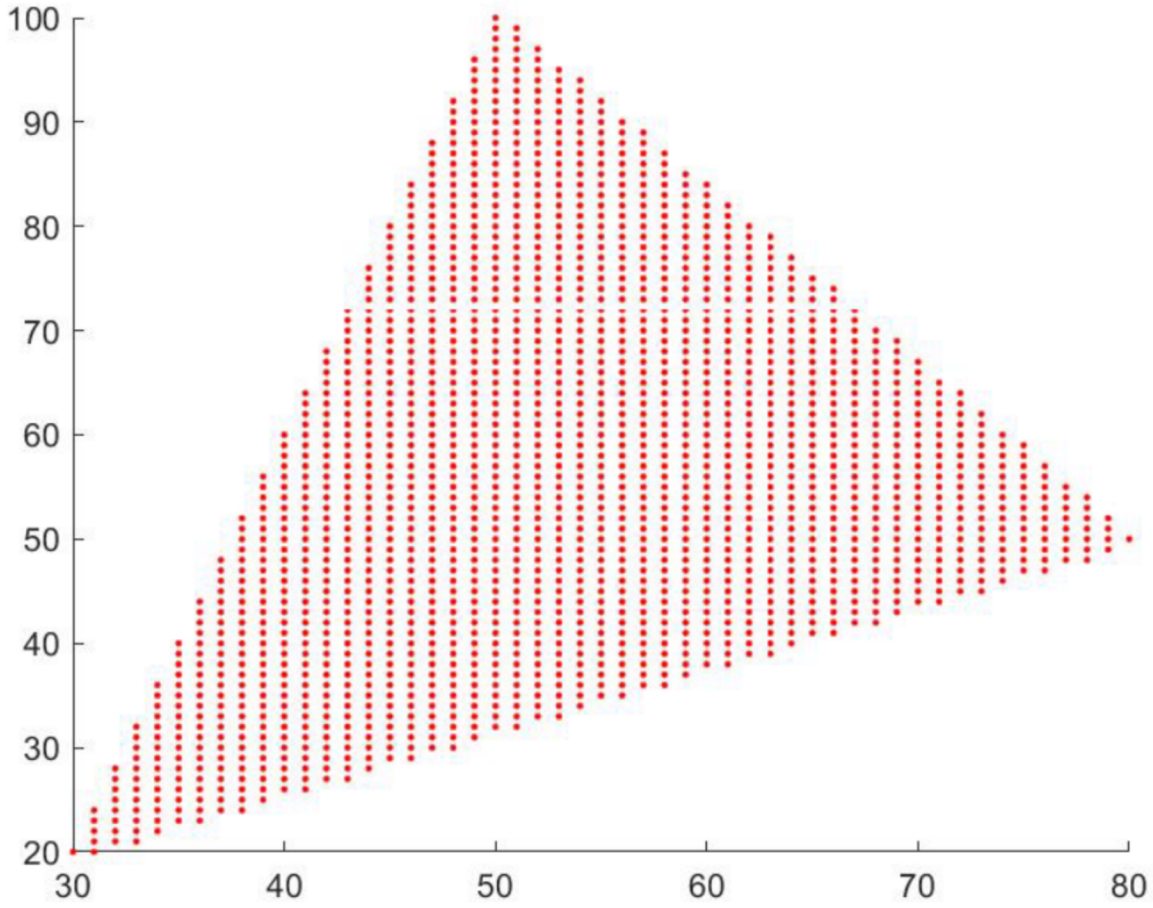

Figure 1: Scan-Filled Triangle

2 (10 points) Finding a Transformation

Before we hop into doing the morphing, let's make sure we can find the transformation between two triangles!

Using the vertices from the prior problem, write a script to do the following:

1. Determine a single transformation matrix that does the following transformation, in this order:
   (a) Rotates by 20 degrees
   (b) Translates by $t_x = 10, t_y = 20$
   (c) Scales uniformly by a factor of 2.
2. Apply your transformation matrix to your new vertices.
3. Uses your new transformed vertices and your original ones, to determine the transformation matrix to take you back to the original locations.
4. Apply this to your transformed vertices to go back to the original locations.

In your report you should provide:

- A plot of your original vertices
- Your transformation matrix
- A plot of your transformed vertices.
- Your learned matrix
- A plot of your recovered vertices.

3 (15 points) Cross Dissolve

As a baseline, let's start off by creating a video that demonstrates cross-dissolve.

Cross-dissolve is just the process of going from one image to another over time, using a global blending factor. You will create a video, 100 frames long, demonstrating the *cross-dissolve* effect. The algorithm is as follows:

1. For $\alpha = 0$ to 1, in increments of 0.01
   (a) Blend your two images using the current value of α
   (b) Save this image as a frame of your movie.

For our purposes, just cross-blend the content that is within the smallest rectangle bounding your correspondence points. All pixels outside of that should be *white*. In addition to creating your two videos (one for each image pair), in your report show the cross-dissolve for each image pair at $\alpha = 0.3$ and $\alpha = 0.7$.

(a) $\alpha = 0.3$

(b) $\alpha = 0.7$

Figure 2: Cross-Dissolve

4 (15 points) Visualize Point and Triangle Correspondences

Now let's generate triangles from your points and visualize them!

Given time, I would have like you to implement your own Delaunay Triangle implementation. However, for this project, just use Matlab's *delaunay* function to get a set of triangles pertaining to the *first* image's points.

Now show your two images with the points and the triangles superimposed. An example can be found in Figure 4

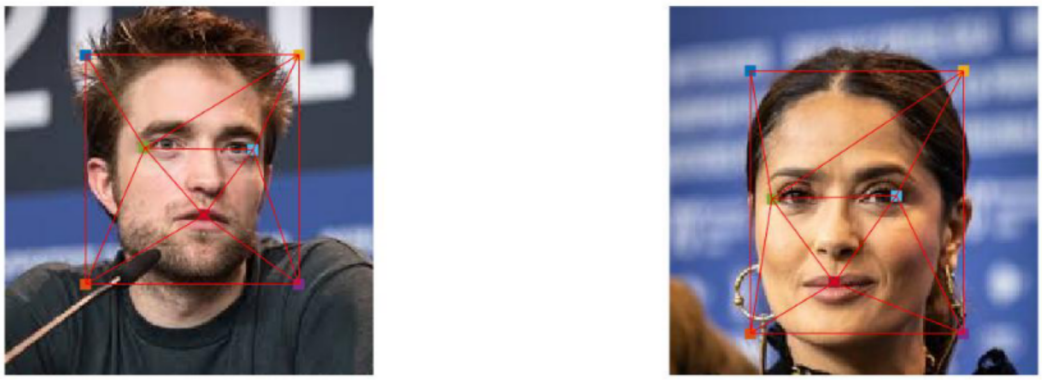

Figure 3: Triangulations

5 (30 points) Morphing

Now let's morph!

We'll once again vary our α from 0 to 1 in increments of 0.01 and create a video 100 frames long. However, now at each step we'll do morphing/blending based on the triangles extracted in the prior part.

Pseudocode for the morphing process was provided in the lecture slides, but here it is, in a bit more detail:

1. For $\alpha = 0$ to 1, in increments of 0.01
   (a) Set the background of the image to white.
   (b) For each triangle
      i. Compute the vertex location of the new destination triangle by linearly interpolating the vertex locations of your two source triangles according to α.
      ii. Compute the affine transformation matrices needed to go from your source triangles to the destination triangle.
      iii. For each pixel in the *destination* triangle (you should be able to re-use your scan-filling code to help you here)
         A. Use the inverse of the transformation matrices to find the corresponding locations in the source triangles.
         B. Blend the values from the sources triangles according to α, and assign that value to the current pixel at the destination triangle.

As you did with cross-dissolving, in addition to creating your two videos (one for each image pair), in your report show the morphed image for each image pair at $\alpha = 0.3$ and $\alpha = 0.7$.

(a) $\alpha = 0.3$

(b) $\alpha = 0.7$

Figure 4: Morphed Images

6 (20 points) Additional Tests

In your report you will provide sample images of your two test cases, and your submission will include the generated videos. To test the robustness of your implementation, we will run a few additional tests on your code.

Submission

NOTE: that 8 points of your grade is based on being able to run your code easily.

IN ADDITION: With your your submission, if you worked with someone else, let me know how evenly the work was split. If each contributed evenly it would be 50/50. I will use this information to adjust grades for pairs where one partner did more of the work.

For your submission, upload to Blackboard a single zip file containing:

1. PDF writeup that includes:
   (a) For Part 1: Your plot
   (b) For Part 2:
      - A plot of your original vertices.
      - Your single transformation matrix.
      - A plot of your transformed vertices.
      - Your "learned" transformation matrix.
      - A plot of your recovered vertices.
   (c) For Part 3: Images for $\alpha = 0.3$ and $\alpha = 0.7$ for both your image pairs.
   (d) For Part 4: Images showing the point correspondences and triangulations.
   (e) For Part 5: Images for $\alpha = 0.3$ and $\alpha = 0.7$ for both your image pairs.
2. A README text file (not Word or PDF) that explains
   - Features of your program
   - Name of your entry-point script
   - Any useful instructions to run your script.
3. Your source files
4. Your 4 images (two image correpondences) and two point correspondence files, organized as specified in the Dataset section.
5. Your 4 videos. One per image pair for Part 3 and Part 5.