CS 445 Final Report
Image morphing

Pranav Asthana
pka4@illinois.edu
Department of Computer Science

Darius Nguyen
huy2@illinois.edu
Department of Computer Science

1 Motivation

Image morphing is the process of creating a sequence of images transitioning between two images to give the appearance of one image morphing into the other. The most common approach to generating this morph sequence is to annotate the images with correspondence points that are used to align the images and solve for the weighted average object shape for each step in the sequence. This average shape is generally found by triangulating the correspondence points and interpolating the triangle coordinates between the two triangulations. This process, however, requires high number of correspondence points to produce good results. In [7], an automated method has been discussed to automatically find the morph sequence using a very small number of correspondence points. This method makes use of structural similarity in the two input images and works by representing the two images using a single vector field defined as the half-way domain vector field. A mapping is learnt for each pixel rather than for each triangle. This method is different from methods like optical flow since the images aren't necessarily of the same object and could have very different lighting conditions. We attempt to use both the triangulation method as well as the automated method to generate morph sequences on our own images. In addition to the approach described in [7], we modify the optimization process to use gradient descent instead of the golden-section search ([5]) they have employed.

2 Approach

We used two different approaches to produce morph sequences on image pairs. They have been described below:

2.1 Morphing using a triangular mesh of correspondent points

Warping 2 images into an average image: For our experiment with this method, we first use images of a cat and a lion. The goal is to morph the cat into the lion. This method calls for partitioning the image into triangles, forming a triangular mesh. Thus, we first need to determine a set of correspondences between the animals. The points are first marked manually by hand, then grouped by regions within the image, such as eyes, forehead, chin, etc.
After enough points are set, we derive an average set of points by taking the mean x-y coordinate values between each pair of correspondences. The average set of points is then used to compute a triangular mesh. There are many ways to configure the triangles, but as discussed in the lectures, Delaunay triangulation is most robust, as it ensures that a reasonable set of triangles are drawn, with no triangles having overly sharp corners. This will improve the quality of the morph by reducing the amount of deformation in intermediate frames.
With the calculated triangulation (each being a set of 3 correspondent points), warping is performed individually on each triangle using affine transformation. For a given frame in the morph sequence, we first retrieve the "average" shape by calculating the weighted average of point coordinates. The weights depend on the frame ordinality. For instance, frame 15 out of 60 total frames will have 15/60 weight on the cat's point and 45/60 weight on the lion's point.
We then compute the affine transformation matrix by parameterizing coordinates of the 3 points into a system of 6 equations, then solve this system using least squares approximation. If forward-mapping were used, there would be "holes" in the warped image, since some points would not have a mapping. Therefore, we use inverse mapping to map a pixel in the warped image back to the original image, by multiplying the inverse of the affine matrix with the current pixel coordinates. The result obtained is a set of floating-point numbers, so we leverage bilinear interpolation to compute the pixel intensity from neighboring source pixels.
Cross-dissolve pixel intensity: Following the above warping procedure, we arrive at 2 weighted-average images: an "average cat" and an "average lion". These images have corresponding feature points in the same locations, but we still need to arrive at a combined image for the output frame.
Hence, interpolation is leveraged again to cross-dissolve intensities between matching pixels in the 2 images. The output intensity for a given pixel is the weighted average of the intensity from its source pixel. Using the same weights from the previous part yields good results.

2.2 Automated morphing using structural similarity

This approach tries to improve the way correspondence maps are established between the two images by reducing the dependence on user input correspondences and using structural similarity between the images. The typical approach is

to create a map ϕ between images I_0 and I_1 and a second map back from I_1 and I_0 to deal with occlusions. Here, we define a half-way vector field v and create two maps, $\phi_0 : v \longrightarrow I_0$ and $\phi_1 : v \longrightarrow I_1$ defined as $\phi_0(p) = p - v(p)$ and $\phi_1(p) = p + v(p)$ for every pixel p.
We initialize v with random uniform values in $[-1, 1]$ and perform a coarse-to-fine optimization using gradient descent where the energy(error) function is described as $E = E_{SIM} + \lambda E_{TPS} + \gamma E_{UI}$ (Eq. 1) where λ and γ are constants set to 0.001 and 100 respectively and the terms are described below.
Similarity energy: This is a measure of how dissimilar two image patches are in terms of their edge structure, ignoring differences in lighting or intensity. These image patches are extracted as $N_0 = I_0(\phi_0(N(p)))$ and $N_1 = I_1(\phi_1(N(p)))$ where $N(p)$ is a 5x5 neighbourhood around p in the vector field and I_0 and I_1 are the two images. It is then computed similar to [9] for each point p as shown below:

$$E_{SIM}(p) = 1 - \left(\frac{2.\sigma_{N_0}.\sigma_{N_1} + 58.5}{\sigma_{N_0}^2 + \sigma_{N_1}^2 + 58.5} \right) . \left(\frac{|\sigma_{N_0 N_1}| + 29.3}{\sigma_{N_0}.\sigma_{N_1} + 29.3} \right)$$

Smoothness energy: This term is added to encourage affine transformations, which can be accomplished by minimizing the thin plate spline(TPS) energy[2]. This is computed using image filters on the v_x and v_y components separately and then summed.
$E_{TPS}(p) = TPS(v_x)(p) + TPS(v_y)(p)$ where $TPS(v_x)$ is computed by convolving the filters shown in Fig. 1 over v_x and summed up as $TPS(v_x) = D_{xx} + 2.D_{xy} + D_{yy}$
UI energy: We want to allow users to mark correspondence points manually to help the optimization process and provide artistic control. Consider a pair of corresponding points u_i^0 and u_i^1 in I_0 and I_1. Let $u_i = (u_i^0 + u_i^1)/2$ and $v_{u_i} = (u_i^1 - u_i^0)/2$. Since we know that u_i^0 and u_i^1 correspond to the same point, we want $v(u_i)$ to equal v_{u_i}. The UI energy term for the point u_i is then $||v(u_i) - v_{u_i}||^2$. However, since u_i may not have integer coordinates, we set the UI energy of the 4 points around u_i $(p_i^1, p_i^2, p_i^3, p_i^4)$ according to their bilinear weights, $b(p_i^j, u_i)$, such that $\sum_{j=1}^{4} b(p_i^j, u_i) p_i^j = u_i$. Finally, we have $E_{UI}(p) = \sum_{i=0}^{n} \sum_{j|p_j=p} b(p_j, u) ||v(u_i) - v_{u_i}||^2$ where n is the number of user defined correspondences. By definition, $E_{UI}(p)$ is non-zero in the neighbourhood of the mid-point of corresponding points and zero elsewhere.

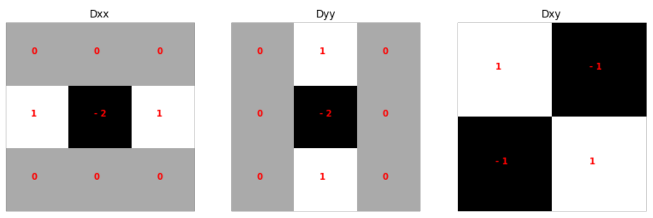

Figure 1. Filters used to compute TPS energy on a grid

Using the energy terms described above, we compute the gradient and perform gradient descent to optimize the vector field in a coarse-to-fine manner where the coarsest level has $\sim$ 60-100 pixels. Each level's optimized v is upsampled and passed to the next level for optimization. The gradient is computed using the finite difference method for both the x and y directions separately as shown below:
$\nabla E(p) = \dfrac{E(v(p) + dv) - E(v(p))}{|dv|}$ where dv is a small value set to (0.1, 0) for calculating the gradient in the x direction and (0, 0.1) for y direction. The updates of the gradient descent were done according to the adam optimizer([6]) with the parameters: $\beta_1 = 0.9, \beta_2 = 0.999, \alpha = 0.005, \epsilon = 10^{-8}$

Motion path from optimized field: Once we have the optimized field that gives us the halfway representation and tells us how each pixel should move in order to align the two images, we can compute the morph sequence. Algorithm details are omitted here due to lack of space but can be found in [7]. In [7], quadratic motion paths have been employed but in our case we were unable to get the optimization to converge to the right value and hence only implemented the linear motion path for testing. Implementing the quadratic motion path would not show any difference since the underlying vector field is problematic. Details of the failure of convergence can be found in 3.2.

3 Results

3.1 Morphing using a triangular mesh of correspondent points

The implementation worked well in this method. From the feature points marked in Fig. 2, the procedure developed the triangular mesh in Fig. 3. The morph function using affine projection produced 60 frames, which were then stitched together in the video linked below.

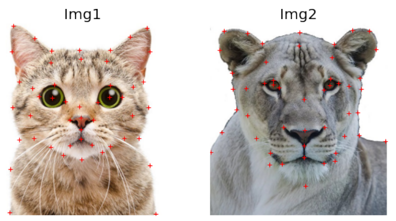

Figure 2. Correspondent points on source images[3][8]

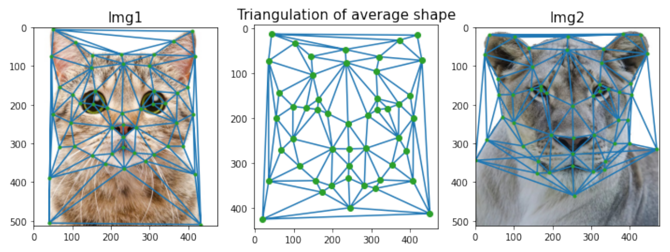

Figure 3. Triangular mesh generated from marked points

Final product: Video of morphing sequence

3.2 Automated morphing

We found that that the gradient descent failed to converge and in fact started diverging after a certain number of iterations. We experimented with reducing the learning rate and tweaked other parameters but failed to converge to the right value. The results shown below are using the values in v just before divergence, which gave the lowest error in our experiments. Note that v is far from the desired representation.

Test 1 - Black arrows on white background

From Fig. 4 (c), we see that v is moving in the right direction, i.e., it has moved from Fig. 4 (b) where v has all random values to keeping the arrow structure valid and slightly tilting all points in the right direction. However, it has been unable to continue beyond this point in the optimization process. Fig. 4 (d) shows the result of the same process but setting $\lambda = 0$ in Eq. 1 in 2.2 since the E_{TPS} term ws causing the divergence.

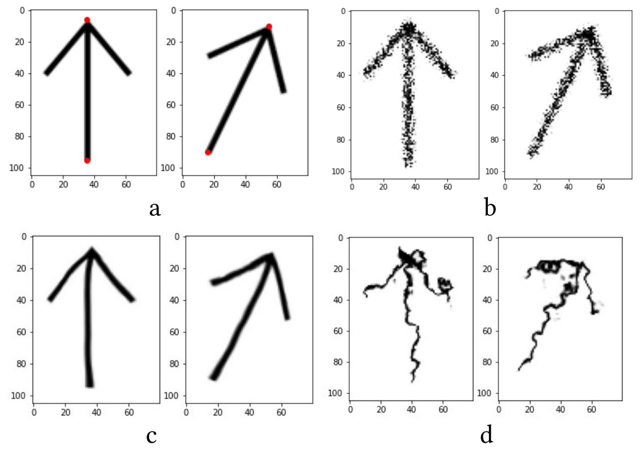

Figure 4. (a) Original pair of images with user correspondences marked in red, (b) Random field applied to images, (c) Field at lowest error in our experiments. Notice that the left arrow is slightly tilting right and vice-versa for the right arrow but it is far from the optimal representation, (d) Field at lowest error in our experiments ignoring E_{TPS} term. Notice that without TPS energy, the transformation is very non-affine leaving straight lines in a wavy mess.

Test 2 - Real images (faces)

In Fig. 5 (b), it is harder to see that the vector field is heading in the right direction, but observe the positions of the eye in both images. The right image's eyes are shifted slightly lower and the left image's eyes are shifted up. However, it fails to maintain the structure of the images. Also notice that the structural similarity term makes the portion to the right of the face similar in both images, since they have a similar edges despite them being different objects.

For our experiments, we found that the gradient descent moves towards the energy minima but after a point, the TPS energy term blows up and v diverges from the optimal representation. Despite using very small learning rates, this behaviour was observed and we were unable to rectify it.

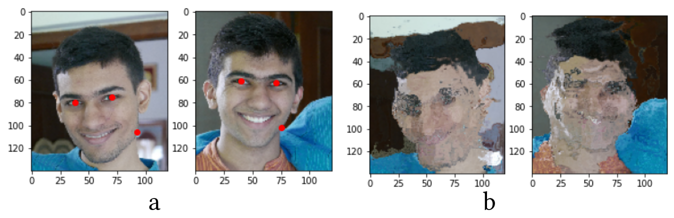

Figure 5. (a) Original pair of images with user correspondences marked in red, (b) Field at lowest error in our experiments applied to both images.

4 Implementation details

The entire codebase for the project is in Python with the following libraries used: opencv, numpy, scipy. For plotting and saving files, matplotlib and pickle were used.
In the triangular mesh method, a function provided by Professor Hoiem is used for marking feature points.[4] The triangle search algorithm checks if a point belong to a triangle using adapted code from Blackpawn[1]. To convert the morph sequence to a video, we use ffmpeg and its Python bindings.

5 Challenge and Innovation

For morphing using a triangular mesh 2.1, although the methodology is presented in the lectures, we needed to figure out the implementation details on our own. It was challenging and time consuming to correctly apply Delaunay triangulation on image pixels, perform affine inverse mapping, and execute interpolation on image shape and intensity. In our first attempt, although we got expected results, the program took a long time to run (30 hours for 60 frames). We then optimized the triangle search algorithm by 1) narrowing the search range and 2) adapting a more efficient method, as described in section 4. The run time reduced drastically to 5 hours for 60 frames.
In [7], the optimization problem is not a trivial one. Implementing the various error terms to run efficiently was moderately difficult but getting the halfway-domain vector field to converge was the most challenging part. Further, some things are not very clearly defined in the paper, for instance how they are performing the golden-section search and their method for computing bounds in the golden section search ([5]). Instead, we opted to change the optimization process to use gradient descent instead and tuning the parameters to improve the optimization was a major challenge which we could not overcome.
Since the paper was quite complex and we changed the optimization step, we were not able to fully implement the method but we were able to show some results with an easier method. We expect 12-15 points.

References

[1] Blackpawn. [n.d.]. *Point in triangle test.* https://blackpawn.com/texts/pointinpoly

[2] Mark Finch, John Snyder, and Hugues Hoppe. 2011. Freeform Vector Graphics with Controlled Thin-Plate Splines. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (Hong Kong, China) *(SA '11)*. Association for Computing Machinery, New York, NY, USA, Article 166, 10 pages. https://doi.org/10.1145/2024156.2024200

[3] Stephen Gallien. [n.d.]. *University of North Alabama's female lion mascot dies after brief illness.* https://abc3340.com/news/local/university-of-north-alabamas-lion-mascot-dies-after-brief-illness

[4] Derek Hoiem. [n.d.]. *Programming Project 3: Gradient-Domain Fusion.* https://courses.engr.illinois.edu/cs445/fa2020/projects/gradient/ComputationalPhotography_ProjectGradient.html

[5] J. Kiefer. 1953. Sequential Minimax Search for a Maximum. *Proc. Amer. Math. Soc.* 4, 3 (1953), 502–506. http://www.jstor.org/stable/2032161

[6] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[7] Jing Liao, Rodolfo S. Lima, Diego Nehab, Hugues Hoppe, Pedro V. Sander, and Jinhui Yu. 2014. Automating Image Morphing Using Structural Similarity on a Halfway Domain. *ACM Trans. Graph.* 33, 5, Article 168 (Sept. 2014), 12 pages. https://doi.org/10.1145/2629494

[8] India Times. [n.d.]. *5 things that scare and stress your cat.* https://timesofindia.indiatimes.com/life-style/relationships/pets/5-things-that-scare-and-stress-your-cat/articleshow/67586673.cms

[9] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612. https://doi.org/10.1109/TIP.2003.819861