Computer Vision Face Morphing

Ian Sinclair
ENCE 3620-1

March 30, 2022
Dr. Mohammad H. Mahoor

Abstract

Considered is an implementation of a face morphing algorithm that given two images of faces will seamlessly merge one image to the other. Critically, most accurately match the position of the faces during the morphing process this technique requires more than simply overlaying the image intensity values. Rather, landmark or feature points are identified on both images that represent consistencies between the images. Facial feature position and border for example. Then, using these points and central vertices the images are triangularized using Delaunay Triangulation. Next, the related triangles on either image are transformed onto the morphed image to create a seamless transition between photos. Here, the process of matching and transforming already triangularized images is explored. In particular, the morphing of two candidate images is approached.

1 Introduction

Within image morphing, two images depicting similar objects are chosen; then, an algorithm is designed to manipulate the combined intensity value of both images to make it appear as if one object is being morphed into the other. In general because the scale and position of the objects in the images can vary to create a good morphing effect the images should be transformed so that the objects are directly on top of each other and proportional. A method described in [1] indicates that this can be done by first finding points that are consistent or correlated between the two images/ objects.

1.1 Feature/Landmark Selection

A landmark is a point that represents a shared feature between both images. For example, in faces, the outline of the face, eyes, or other facial features make good landmark points because they are likely consistent or present on all pictures of faces. It is important to note that landmark points do not need to share the exact pixel location in both images (And generally they don't.) however, they describe a correlation between objects in the image, a eye landmark points to eyes on both images.

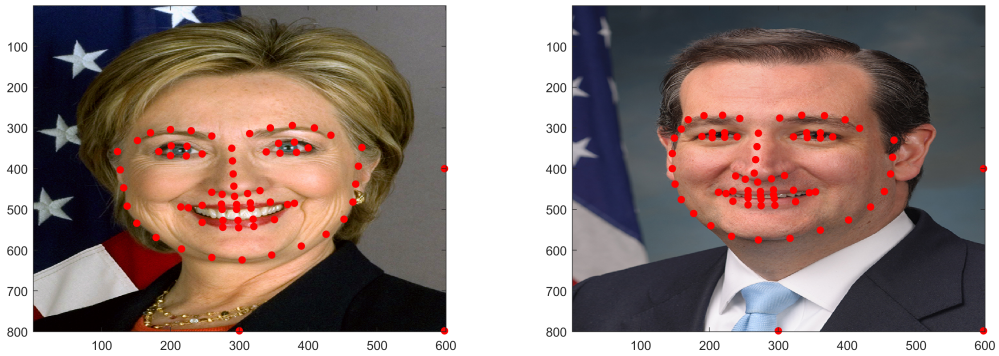


Figure 1: Example of feature selection on facial images.

Note, that a map can then be made from the features of one image to the corresponding features of another. In image morphing; typically, features on both images are mapped to a third morphed image that is somewhere in between the two objects. Or rather a linear parametric model is desired to transition the features of one image slowly towards another.
It is important to note that the selection of features is beyond the scope of this report, and so images with pre-defined landmarks are selected. However, a general analysis is described.
Take images I and J, where coordinate $I(x, y), J(x, y)$ is the intensity value of either image corresponding to coordinate (x, y). Then, let M be a a morphed image in between I and J. There α is the decimal representation of the percent of morphing. Then, the intensity of any point on M corresponds to,

$$M(x, y) = (1 - \alpha)I(x, y) + \alpha J(x, y), \qquad \alpha \in [0, 1] \tag{1}$$

And so note that $\alpha = 1$ corresponds to $M = J$ and for $\alpha = 0$, it follows $M = I$.
Next, because the features of the original images need not be aligned a linear parametric mapping between features is also required. So let (x_I, y_I) and (x_J, y_J) be a feature on image I and J respectively, then that feature maps to feature on M at (x_M, y_M) by,

$$x_M = (1 - \alpha)x_I + \alpha x_J$$
$$y_M = (1 - \alpha)y_I + \alpha y_J$$

Which, given enough features defines a map from images I, J to M for any point in either image. The extension of this map is best implemented using triangulization techniques.

1.2 Delaunay Triangulation

Given a set of feature points on an image, one of the best ways to morph the image while preserving key characteristics is two triangulize different sections on the image and maintain the information in each triangle. A good method described in [1] is Delaunay Triangulation in which the minimum angle of all triangles in the triangulization is maximized. This is done by ensuring the no external feature point is within the circumference of a circle drawn between vertices in each triangle.

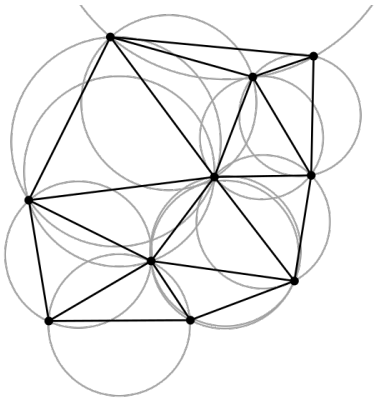


Figure 2: Delaunay Triangulation showing connecting circles.

Then, because the features between images are related, creating a map to the morphed image is the same as finding a geometric transformation from one triangulization to another. Because of the inconsistency of scale and position this is best done using a origin invariant operation like an affine transformation.

1.3 Affine Transformations

An Affine Transformation in $2D$ is capable of translating, rotating, and skewing objects. Critically, lines and parallelisms are preserved however, not necessarily distances or angles. As a result, given an input triangle and a target triangle with different dimensions, it is possible to transform the first to the target. And because lines are preserved the result must still be a triangle. Formally,

an affine transformation is an automorphism for an affine space, and so it is well defined and an objects structure will be preserved, points will be mapped to points, lines to lines, ect. Affine transformations can also be represented as a composition of other linear translations including similarity, and shearing. And as a result, the transformation in homogeneous coordinates for a $2D$ space has 6 degrees of freedom.

$$M_f = \left[\begin{array}{c|c} A & b \\ \hline 0 & 1 \end{array}\right] \tag{2}$$

Where A is a 2×2 matrix and b is a 2×1 column vector.
Now because there are 6 unknown parameters, it is possible to construction a unique solution matrix for the affine transformation between two image spaces using 3 known coordinate points from either image and solving. Which is ideal, because a triangle has three vertices and so mapping the feature points between vertices is all that is required to define the transformation between each triangle in the triangulation from the original to the morphed image.

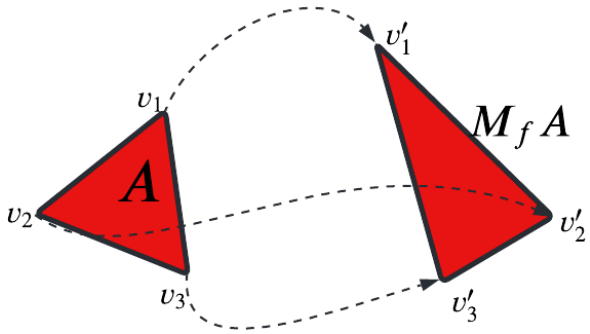


Figure 3: Affine transformation between two triangles.

Within the scope of the report, affine transformations are used to map triangles tightly bounded in by rectangular images cropped from each original picture.

2 Procedure

Considered are two images of the faces of politicians Hillary Clinton and Ted Cruz, with the intention to design an algorithm to smoothly morph the image of Hillary Clinton to that of Ted Cruz. And so consider the raw input images.

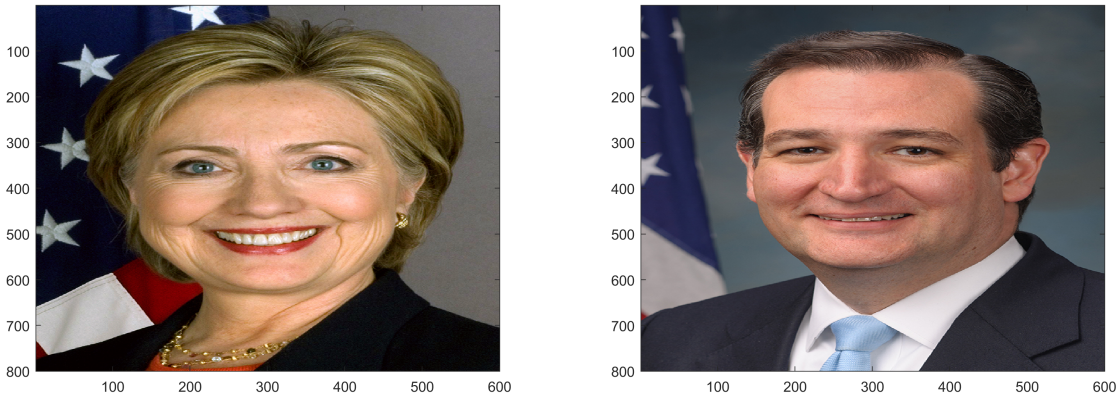


Figure 4: Raw images of Hillary Clinton and Ted Cruz.

Which are uploaded by the MATLAB commans,

```
Raw_Image_Hillary = imread("hillary_clinton.jpg");
Raw_Image_Ted = imread("ted_cruz.jpg");
```

Now, the desired output is a morphed image between the two faces that fluidly transitions between them following the equation,

$$M(x,y) = (1-\alpha)I(x,y) + \alpha J(x,y) \tag{3}$$

For input images I, J and morphed image M. Then, because the scale and position of aspects in the image are inconsistent, a feature matching technique can be better results than simply overlaying the images.
In general consider the overlay with $\alpha = 0.5$, (or a morphed image taking half from each input image.)

Figure 5: Overlay with no smoothing adjustments

```
blankimage = zeros(800,600,3,'uint8');

alpha = 0.5;
for i = 1:length(Raw_Image_Hillary)
    for j = 1:600
        blankimage(i,j,1) = (1-alpha)*Raw_Image_Hillary(i,j,1) + alpha*Raw_Image_Ted(i,j,1);
        blankimage(i,j,2) = (1-alpha)*Raw_Image_Hillary(i,j,2) + alpha*Raw_Image_Ted(i,j,2);
        blankimage(i,j,3) = (1-alpha)*Raw_Image_Hillary(i,j,3) + alpha*Raw_Image_Ted(i,j,3);
    end
end

figure(3)
imagesc(blankimage)
```

Which is not a very good morphing because the facial features don't line up. Or rather, in image morphing, the idea is an object within either image appears to transform into the other, and so matching the position of objects is critical for realism.
As a result, a more technical approach detail in [1] is considered.

2.1 Feature Selection

Within the scope of the experiment, images with pre-defined features have been selected. And so computing the location of the landmarks and displaying them on the image is a task in file management.

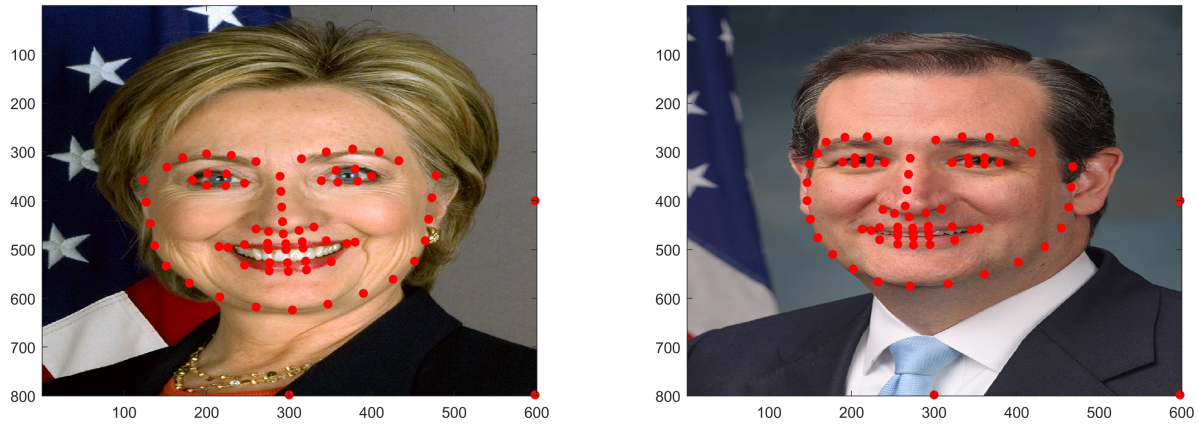


Figure 6: Feature Selection and Location on each image

```
Feature_Points_Hillary_Text = fopen("hillary_clinton.txt");
Feature_Points_Ted_Text = fopen("ted_cruz.txt");

Feature_Points_Hillary = fscanf(Feature_Points_Hillary_Text, '%d', [2 100]);
Feature_Points_Ted = fscanf(Feature_Points_Ted_Text, '%d', [2 76]);

figure(3)
imagesc(Raw_Image_Hillary)
hold on
for feature = Feature_Points_Hillary
    plot(feature(1), feature(2), '.', 'Color', 'red', 'MarkerSize', 20)
end

figure(4)
imagesc(Raw_Image_Ted)
hold on
for feature = Feature_Points_Ted
    plot(feature(1), feature(2), '.', 'Color', 'red', 'MarkerSize', 20)
end
```

Here, the facial features such as samples of the outline of the eyes, mouths and outward faces are taken as landmarks. Essentially, elements that are present (consistent) in both images are selected. Additionally, elements on the boarder of the image are chosen to facilitate changes in the background.
Note, there are exactly the same number of features between the images, this is critical for the desired one-to-one well defined transformation from either image to the morphed image.

2.2 Delaunay Triangulation

After features have been selected, a triangulization of the image can be taken. Where each feature becomes a vertex of at least one triangle. This facilitates the mapping of non-feature points inside the triangles to the morphed image.
As a result, the topology of each triangle has a role in how different parts of the input image are mapped to the morphed image. Generally, larger triangles are preferred, because significant objects in images usually take up space, or rather smaller, skinny triangles are likely to splice larger objects, possible warping the realism of

the final image.
Therefore, Delaunay Triangulation is the suggested method by [1] because it ensures the angle of each triangle is maximal; or admits the triangulation with on average the largest triangles.

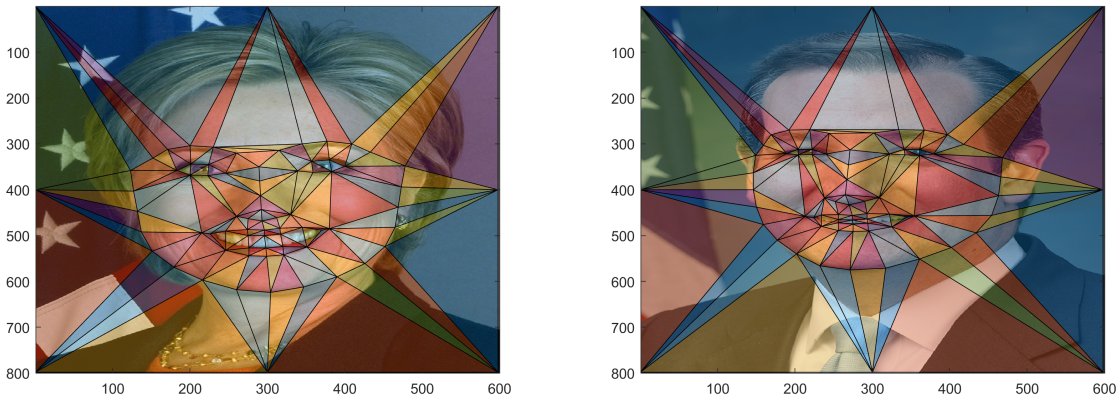


Figure 7: Delaunay Triangulation of the input images based on the selected features

```
Delaunay_Triangulation_File = fopen("tri.txt");

Delaunay_Triangulation = fscanf(Delaunay_Triangulation_File, '%d', [3, 142]);

% Triangles

Hillary_Triangulation = {};

for triangle = Delaunay_Triangulation
    vertex1 = [ Feature_Points_Hillary( 1 , triangle(1)+1 ),
        Feature_Points_Hillary( 1 , triangle(2)+1 ),
        Feature_Points_Hillary( 1 , triangle(3) +1 )];
    vertex2 = [ Feature_Points_Hillary( 2 , triangle(1)+ 1 ),
        Feature_Points_Hillary( 2 , triangle(2) + 1 ),
        Feature_Points_Hillary( 2 , triangle(3) + 1 )
        ];
    Hillary_Triangulation{1,end+1} = vertex1;
    Hillary_Triangulation{2,end} = vertex2;
end


Ted_Triangulation = {};

for triangle = Delaunay_Triangulation
    vertex1 = [ Feature_Points_Ted( 1 , triangle(1)+1 ),
        Feature_Points_Ted( 1 , triangle(2)+1 ),
        Feature_Points_Ted( 1 , triangle(3) +1 )];
    vertex2 = [ Feature_Points_Ted( 2 , triangle(1)+ 1 ),
        Feature_Points_Ted( 2 , triangle(2) + 1 ),
        Feature_Points_Ted( 2 , triangle(3) + 1 )
        ];
    Ted_Triangulation{1,end+1} = vertex1;
    Ted_Triangulation{2,end} = vertex2;
end
```

Note, the above image is color coordinated based on the triangles in either image that map to the same target triangle in the morphed image.
Additionally, the morphed image must be triangulated based on α.

```
Morphed_Triangulation = {};
for i = 1:length(Hillary_Triangulation)
    vertex1 = [ (1-alpha)*Hillary_Triangulation{1,i}(1) + alpha*Ted_Triangulation{1,i}(1),
        (1-alpha)*Hillary_Triangulation{1,i}(2) + alpha*Ted_Triangulation{1,i}(2),
        (1-alpha)*Hillary_Triangulation{1,i}(3) + alpha*Ted_Triangulation{1,i}(3)
        ];
    vertex2 = [ (1-alpha)*Hillary_Triangulation{2,i}(1) + alpha*Ted_Triangulation{2,i}(1),
        (1-alpha)*Hillary_Triangulation{2,i}(2) + alpha*Ted_Triangulation{2,i}(2),
        (1-alpha)*Hillary_Triangulation{2,i}(3) + alpha*Ted_Triangulation{2,i}(3)
        ];
    Morphed_Triangulation{1,end+1} = vertex1;
    Morphed_Triangulation{2,end} = vertex2;
end
```

3 Affine Transformation

Now it is possible to define an affine transformation between the corresponding triangles in both images to the morphed image.

```
    Affine_Hillary = {};
    Affine_Ted = {};

    for i = 1:length(Hillary_Triangulation)
        moving_points_h = [Hillary_Triangulation{1,i}(1) , Hillary_Triangulation{2,i}(1);
            Hillary_Triangulation{1,i}(2) , Hillary_Triangulation{2,i}(2);
            Hillary_Triangulation{1,i}(3) , Hillary_Triangulation{2,i}(3)];

        moving_points_t = [Ted_Triangulation{1,i}(1) , Ted_Triangulation{2,i}(1);
            Ted_Triangulation{1,i}(2) , Ted_Triangulation{2,i}(2);
            Ted_Triangulation{1,i}(3) , Ted_Triangulation{2,i}(3)];

        fixed_points = [Morphed_Triangulation{1,i}(1) , Morphed_Triangulation{2,i}(1);
            Morphed_Triangulation{1,i}(2) , Morphed_Triangulation{2,i}(2);
            Morphed_Triangulation{1,i}(3) , Morphed_Triangulation{2,i}(3)];

        Affine_Hillary{end+1} = fitgeotrans(moving_points_h, fixed_points, 'affine');
        Affine_Ted{end+1} = fitgeotrans(moving_points_t, fixed_points, 'affine');
    end
```

3.1 Inverse Mapping

To prevent pixels in the morphed image from being skipped by the codomain of the affine transformation inverse mapping is preformed starting with each triangle in the morphed image and mapping back to the original images of Hillary and Ted.

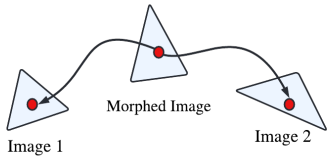


Figure 8: Inverse mapping between triangular regions.

This is done by creating a bounding box that encloses each triangle in the morphed image. Which, then if a point in the box is within the triangle its the corresponding values from the original images are calculated by the inverse affine transformation. Critically, because the affine transformation is an automorphism that preserves lines, if a point (x', y') is in the the triangle in the morphed image, it is also within the corresponding triangle in the original images.
If a point is within the box and not within a triangle, its value is masked. This can be done because the set of all triangles in the morphed image spans the image space, and so the values will be filled by other triangle iterations.

```
    for i = 1:length(Morphed_Triangulation)
        bbox_Hillary{end+1} = [min(Hillary_Triangulation{1,i}), min(Hillary_Triangulation{2,i
            }), max(Hillary_Triangulation{1,i}), max(Hillary_Triangulation{2,i}) ];
        bbox_Ted{end+1} = [min(Ted_Triangulation{1,i}), min(Ted_Triangulation{2,i}), max(
            Ted_Triangulation{1,i}), max(Ted_Triangulation{2,i}) ];
        bbox_Target{end+1} = [min(Morphed_Triangulation{1,i}), min(Morphed_Triangulation{2,i
            }), max(Morphed_Triangulation{1,i}), max(Morphed_Triangulation{2,i}) ];
    end
```

And lastly, the values are filled by,

```
for i = 1:length(bbox_Target)
        for j = bbox_Target{i}(1):bbox_Target{i}(3)
            for k = bbox_Target{i}(2):bbox_Target{i}(4)
                [IN,ON] = inpolygon(j,k, Morphed_Triangulation{1,i},Morphed_Triangulation{2,i
                    });
                if IN == 1 || ON == 1
                    if j ~= 0 && k ~= 0
                        H_coord = Affine_Hillary{i}.T^-1*[j;k;1];
                        T_coord = Affine_Ted{i}.T^-1*[j;k;1];
                        Out_Image(ceil(k),ceil(j),1) = (1-alpha)*Raw_Image_Hillary(ceil(
                            H_coord(2)), ceil(H_coord(1)),1) + alpha*Raw_Image_Ted(ceil(
                            T_coord(2)), ceil(T_coord(1)),1);
                        Out_Image(ceil(k),ceil(j),2) = (1-alpha)*Raw_Image_Hillary(ceil(
                            H_coord(2)), ceil(H_coord(1)),2) + alpha*Raw_Image_Ted(ceil(
                            T_coord(2)), ceil(T_coord(1)),2);
                        Out_Image(ceil(k),ceil(j),3) = (1-alpha)*Raw_Image_Hillary(ceil(
                            H_coord(2)), ceil(H_coord(1)),3) + alpha*Raw_Image_Ted(ceil(
                            T_coord(2)), ceil(T_coord(1)),3);
                    end
                end
            end
        end
    end
```

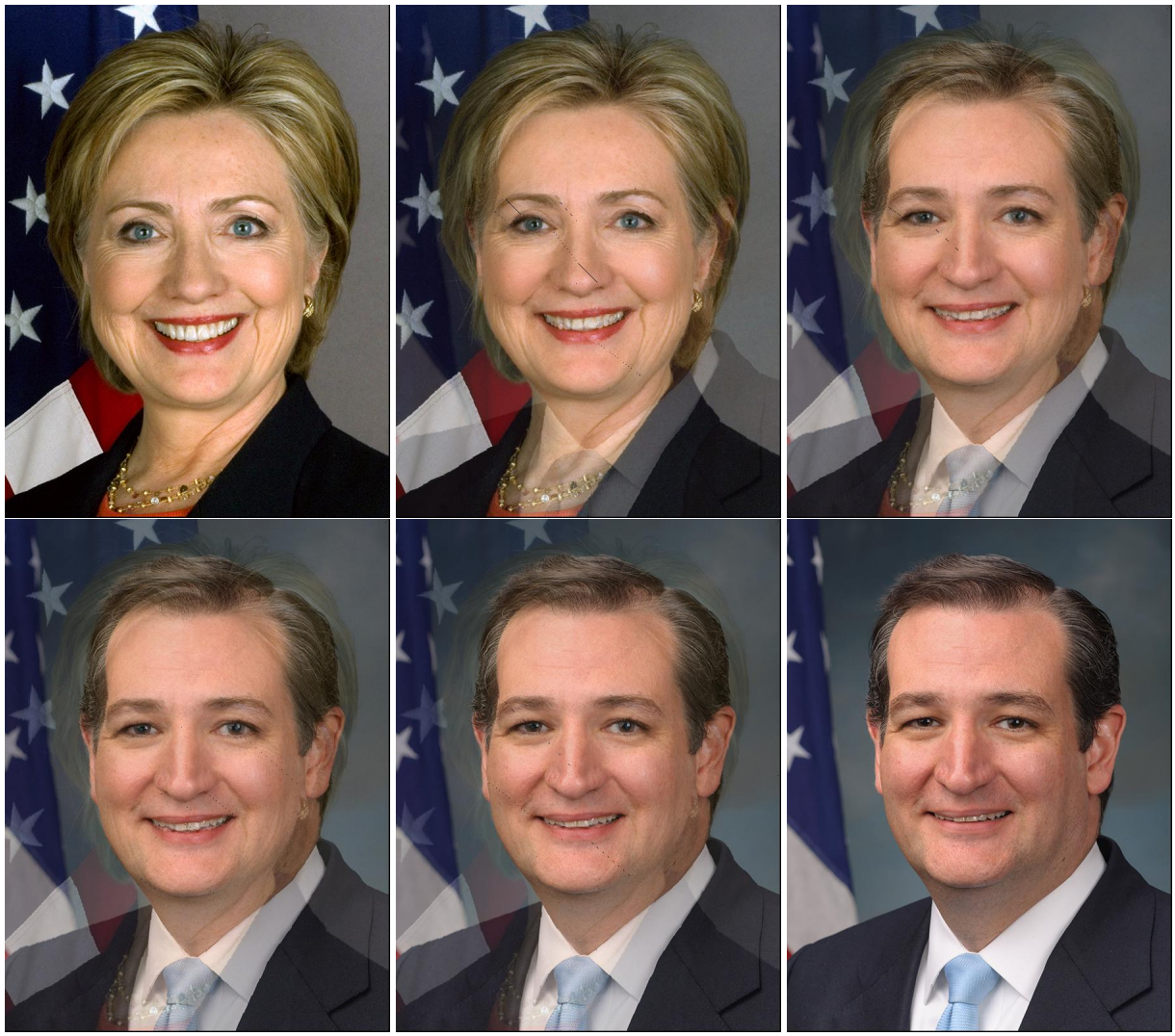
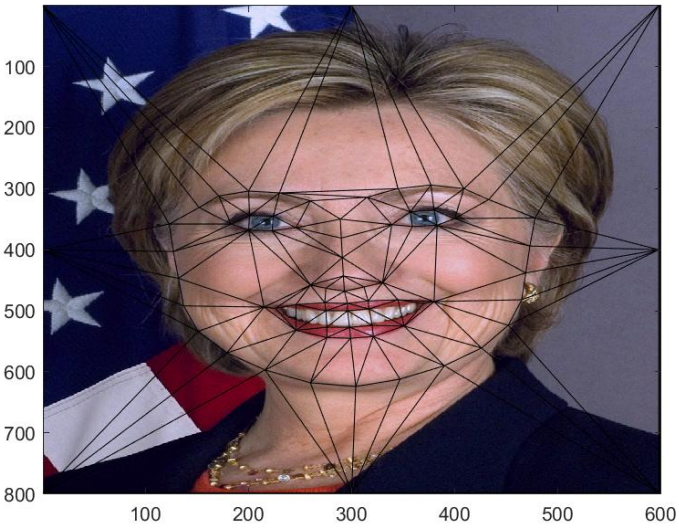


Figure 9: Image displaying morphing between images.



https://youtu.be/0gpeGIfzTTU

https://youtu.be/cy6gTpXRqVA

4 Conclusion

Considered is an algorithm to fluidly one image into another. This analysis includes understand feature detection, triangulization, the implementation of affine transformations and bounding boxes, and parametric transition models between states (feature locations). In particular, two candidate images were selected that already had selected features and a triangulization. And so, the scope of the report focused on the implementation of affine transformations and image warping techniques. Critically, within the algorithm, the interpolation method on boundaries between triangles is not fully developed and so there may be obstructions or undesired masked portions of the images while morphing.

5 References:

[1] Face Morph Using OpenCV | $C++$ / Python. Satya Mallick. MARCH 11, 2016. Learn OPENCV.

6 Appendix

```
clear

Raw_Image_Hillary = imread("hillary_clinton.jpg");
Raw_Image_Ted = imread("ted_cruz.jpg");



Feature_Points_Hillary_Text = fopen("hillary_clinton.txt");

Feature_Points_Ted_Text = fopen("ted_cruz.txt");

Feature_Points_Hillary = fscanf(Feature_Points_Hillary_Text, '%d', [2 100]);

Feature_Points_Ted = fscanf(Feature_Points_Ted_Text, '%d', [2 76]);


Delaunay_Triangulation_File = fopen("tri.txt");

Delaunay_Triangulation = fscanf(Delaunay_Triangulation_File, '%d', [3, 142]);




Hillary_Triangulation = {};

for triangle = Delaunay_Triangulation
    vertex1 = [ Feature_Points_Hillary( 1 , triangle(1)+1 ),
        Feature_Points_Hillary( 1 , triangle(2)+1 ),
        Feature_Points_Hillary( 1 , triangle(3) +1 )];
    vertex2 = [ Feature_Points_Hillary( 2 , triangle(1)+ 1 ),
        Feature_Points_Hillary( 2 , triangle(2) + 1 ),
        Feature_Points_Hillary( 2 , triangle(3) + 1 )
        ];
    Hillary_Triangulation{1,end+1} = vertex1;
    Hillary_Triangulation{2,end} = vertex2;
end


Ted_Triangulation = {};

for triangle = Delaunay_Triangulation
    vertex1 = [ Feature_Points_Ted( 1 , triangle(1)+1 ),
        Feature_Points_Ted( 1 , triangle(2)+1 ),
        Feature_Points_Ted( 1 , triangle(3) +1 )];
    vertex2 = [ Feature_Points_Ted( 2 , triangle(1)+ 1 ),
        Feature_Points_Ted( 2 , triangle(2) + 1 ),
        Feature_Points_Ted( 2 , triangle(3) + 1 )
        ];
    Ted_Triangulation{1,end+1} = vertex1;
    Ted_Triangulation{2,end} = vertex2;
end

index = 0;
for alpha = 0:0.1:1
```

```
56
57      Morphed_Triangulation = {};
58
59      blendPoint = [];
60      hold on
61      for i = 1:length(Hillary_Triangulation)
62          vertex1 = [ (1-alpha)*Hillary_Triangulation{1,i}(1) + alpha*Ted_Triangulation{1,i}(1),
63              (1-alpha)*Hillary_Triangulation{1,i}(2) + alpha*Ted_Triangulation{1,i}(2),
64              (1-alpha)*Hillary_Triangulation{1,i}(3) + alpha*Ted_Triangulation{1,i}(3)
65              ];
66          vertex2 = [ (1-alpha)*Hillary_Triangulation{2,i}(1) + alpha*Ted_Triangulation{2,i}(1),
67              (1-alpha)*Hillary_Triangulation{2,i}(2) + alpha*Ted_Triangulation{2,i}(2),
68              (1-alpha)*Hillary_Triangulation{2,i}(3) + alpha*Ted_Triangulation{2,i}(3)
69              ];
70          Morphed_Triangulation{1,end+1} = vertex1;
71          Morphed_Triangulation{2,end} = vertex2;
72      end
73
74
75      Affine_Hillary = {};
76      Affine_Ted = {};
77
78      for i = 1:length(Hillary_Triangulation)
79          moving_points_h = [Hillary_Triangulation{1,i}(1) , Hillary_Triangulation{2,i}(1);
80              Hillary_Triangulation{1,i}(2) , Hillary_Triangulation{2,i}(2);
81              Hillary_Triangulation{1,i}(3) , Hillary_Triangulation{2,i}(3)];
82
83          moving_points_t = [Ted_Triangulation{1,i}(1) , Ted_Triangulation{2,i}(1);
84              Ted_Triangulation{1,i}(2) , Ted_Triangulation{2,i}(2);
85              Ted_Triangulation{1,i}(3) , Ted_Triangulation{2,i}(3)];
86
87          fixed_points = [Morphed_Triangulation{1,i}(1) , Morphed_Triangulation{2,i}(1);
88              Morphed_Triangulation{1,i}(2) , Morphed_Triangulation{2,i}(2);
89              Morphed_Triangulation{1,i}(3) , Morphed_Triangulation{2,i}(3)];
90
91          Affine_Hillary{end+1} = fitgeotrans(moving_points_h, fixed_points, 'affine');
92          Affine_Ted{end+1} = fitgeotrans(moving_points_t, fixed_points, 'affine');
93      end
94
95      Out_Image = zeros(800,600,3,'uint8');
96
97
98
99      bbox_Hillary = {};
100     bbox_Ted = {};
101     bbox_Target = {};
102
103     for i = 1:length(Morphed_Triangulation)
104         bbox_Hillary{end+1} = [min(Hillary_Triangulation{1,i}), min(Hillary_Triangulation{2,i}), max(
                Hillary_Triangulation{1,i}), max(Hillary_Triangulation{2,i})];
105         bbox_Ted{end+1} = [min(Ted_Triangulation{1,i}), min(Ted_Triangulation{2,i}), max(
                Ted_Triangulation{1,i}), max(Ted_Triangulation{2,i})];
106         bbox_Target{end+1} = [min(Morphed_Triangulation{1,i}), min(Morphed_Triangulation{2,i}), max(
                Morphed_Triangulation{1,i}), max(Morphed_Triangulation{2,i})];
107     end
108
109
110     cropped_images_Hillary = {};
```

```
cropped_images_Ted = {};

for bbox = bbox_Hillary
    box_width = bbox{1}(3) - bbox{1}(1);
    box_height = bbox{1}(4) - bbox{1}(2);
    cropped_images_Hillary{end+1} = imcrop(Raw_Image_Hillary, [bbox{1}(1), bbox{1}(2), box_height ,
        box_width]);
end
for bbox = bbox_Ted
    box_width = bbox{1}(3) - bbox{1}(1);
    box_height = bbox{1}(4) - bbox{1}(2);
    cropped_images_Ted{end+1} = imcrop(Raw_Image_Ted, [bbox{1}(1), bbox{1}(2), box_height ,box_width
        ]);
end




warped_images_Hillary = {};
warped_images_Ted = {};
for i = 1:length(cropped_images_Ted)
    warped_images_Hillary{end+1} = imwarp(cropped_images_Hillary{i}, Affine_Hillary{i});
    warped_images_Ted{end+1} = imwarp(cropped_images_Ted{i}, Affine_Ted{i});
end


Out_Image = zeros(800,600,3,'uint8');

for i = 1:length(bbox_Target)
    for j = bbox_Target{i}(1):bbox_Target{i}(3)
        for k = bbox_Target{i}(2):bbox_Target{i}(4)
            [IN,ON] = inpolygon(j,k, Morphed_Triangulation{1,i},Morphed_Triangulation{2,i});
            if IN == 1 || ON == 1
                if j ~= 0 && k ~= 0
                    H_coord = Affine_Hillary{i}.T^-1*[j;k;1];
                    T_coord = Affine_Ted{i}.T^-1*[j;k;1];
                    Out_Image(ceil(k),ceil(j),1) = (1-alpha)*Raw_Image_Hillary(ceil(H_coord(2)),
                        ceil(H_coord(1)),1) + alpha*Raw_Image_Ted(ceil(T_coord(2)), ceil(T_coord(1))
                        ,1);
                    Out_Image(ceil(k),ceil(j),2) = (1-alpha)*Raw_Image_Hillary(ceil(H_coord(2)),
                        ceil(H_coord(1)),2) + alpha*Raw_Image_Ted(ceil(T_coord(2)), ceil(T_coord(1))
                        ,2);
                    Out_Image(ceil(k),ceil(j),3) = (1-alpha)*Raw_Image_Hillary(ceil(H_coord(2)),
                        ceil(H_coord(1)),3) + alpha*Raw_Image_Ted(ceil(T_coord(2)), ceil(T_coord(1))
                        ,3);
                end
            end
            if IN == 1 || ON == 1
                if j >= 1 && k >= 1
                    H_coord = Affine_Hillary{i}.T^-1*[j;k;1];
                    T_coord = Affine_Ted{i}.T^-1*[j;k;1];
                    Out_Image(floor(k),floor(j),1) = (1-alpha)*Raw_Image_Hillary(ceil(H_coord(2)),
                        ceil(H_coord(1)),1) + alpha*Raw_Image_Ted(ceil(T_coord(2)), ceil(T_coord(1))
                        ,1);
                    Out_Image(floor(k),floor(j),2) = (1-alpha)*Raw_Image_Hillary(ceil(H_coord(2)),
                        ceil(H_coord(1)),2) + alpha*Raw_Image_Ted(ceil(T_coord(2)), ceil(T_coord(1))
                        ,2);
```

```
156                         Out_Image(floor(k),floor(j),3) = (1-alpha)*Raw_Image_Hillary(ceil(H_coord(2)),
                                ceil(H_coord(1)),3) + alpha*Raw_Image_Ted(ceil(T_coord(2)), ceil(T_coord(1))
                                ,3);
157                     end
158                 end
159
160                 if ON == 1
161                     if j >= 1 && k >= 1
162                         H_coord = Affine_Hillary{i}.T^-1*[j;k;1];
163                         T_coord = Affine_Ted{i}.T^-1*[j;k;1];
164                         if H_coord(1) >= 1 && H_coord(2) >= 1 && T_coord(1) >= 1 && T_coord(2) >= 1
165                             Out_Image(floor(k),floor(j),1) = (1-alpha)*Raw_Image_Hillary(floor(H_coord
                                    (2)), floor(H_coord(1)),1) + alpha*Raw_Image_Ted(floor(T_coord(2)),
                                    floor(T_coord(1)),1);
166                             Out_Image(floor(k),floor(j),2) = (1-alpha)*Raw_Image_Hillary(floor(H_coord
                                    (2)), floor(H_coord(1)),2) + alpha*Raw_Image_Ted(floor(T_coord(2)),
                                    floor(T_coord(1)),2);
167                             Out_Image(floor(k),floor(j),3) = (1-alpha)*Raw_Image_Hillary(floor(H_coord
                                    (2)), floor(H_coord(1)),3) + alpha*Raw_Image_Ted(floor(T_coord(2)),
                                    floor(T_coord(1)),3);
168                             Out_Image(ceil(k),ceil(j),1) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2))
                                    , floor(H_coord(1)),1) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),1);
169                             Out_Image(ceil(k),ceil(j),2) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2))
                                    , floor(H_coord(1)),2) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),2);
170                             Out_Image(ceil(k),ceil(j),3) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2))
                                    , floor(H_coord(1)),3) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),3);
171                             Out_Image(ceil(k),floor(j),1) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2)
                                    ), floor(H_coord(1)),1) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),1);
172                             Out_Image(ceil(k),floor(j),2) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2)
                                    ), floor(H_coord(1)),2) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),2);
173                             Out_Image(ceil(k),floor(j),3) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2)
                                    ), floor(H_coord(1)),3) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),3);
174                             Out_Image(floor(k),ceil(j),1) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2)
                                    ), floor(H_coord(1)),1) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),1);
175                             Out_Image(floor(k),ceil(j),2) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2)
                                    ), floor(H_coord(1)),2) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),2);
176                             Out_Image(floor(k),ceil(j),3) = (1-alpha)*Raw_Image_Hillary(floor(H_coord(2)
                                    ), floor(H_coord(1)),3) + alpha*Raw_Image_Ted(floor(T_coord(2)), floor(
                                    T_coord(1)),3);
177                         end
178                     end
179                 end
180             end
181         end
182     end
183
184
185     figure(5)
186     imagesc(Out_Image)
187     hold on
```

```
188         for triangle = Morphed_Triangulation
189                 pgon = polyshape(triangle{1},triangle{2});
190                 plot(pgon, 'FaceColor', 'blue', 'FaceAlpha',0.1);
191         end
192         savefig('tempfig.fig')
193         figs = openfig('tempfig.fig');
194 
195         saveas(figs, append('C:\Users\IanSi\OneDrive\Desktop\Project 3\AAATriangleMorphingImage',string(
                index),'.jpg'));
196 %       imwrite(Out_Image,append('C:\Users\IanSi\OneDrive\Desktop\Project 3\AAUpdatedMorphingImage',string(
            index),'.jpg'));
197         index = index + 1;
198 
199 end
200 
201 % figure(3)
202 % imagesc(Out_Image)
```