MINISTRY OF EDUCATION AND RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

Image Processing

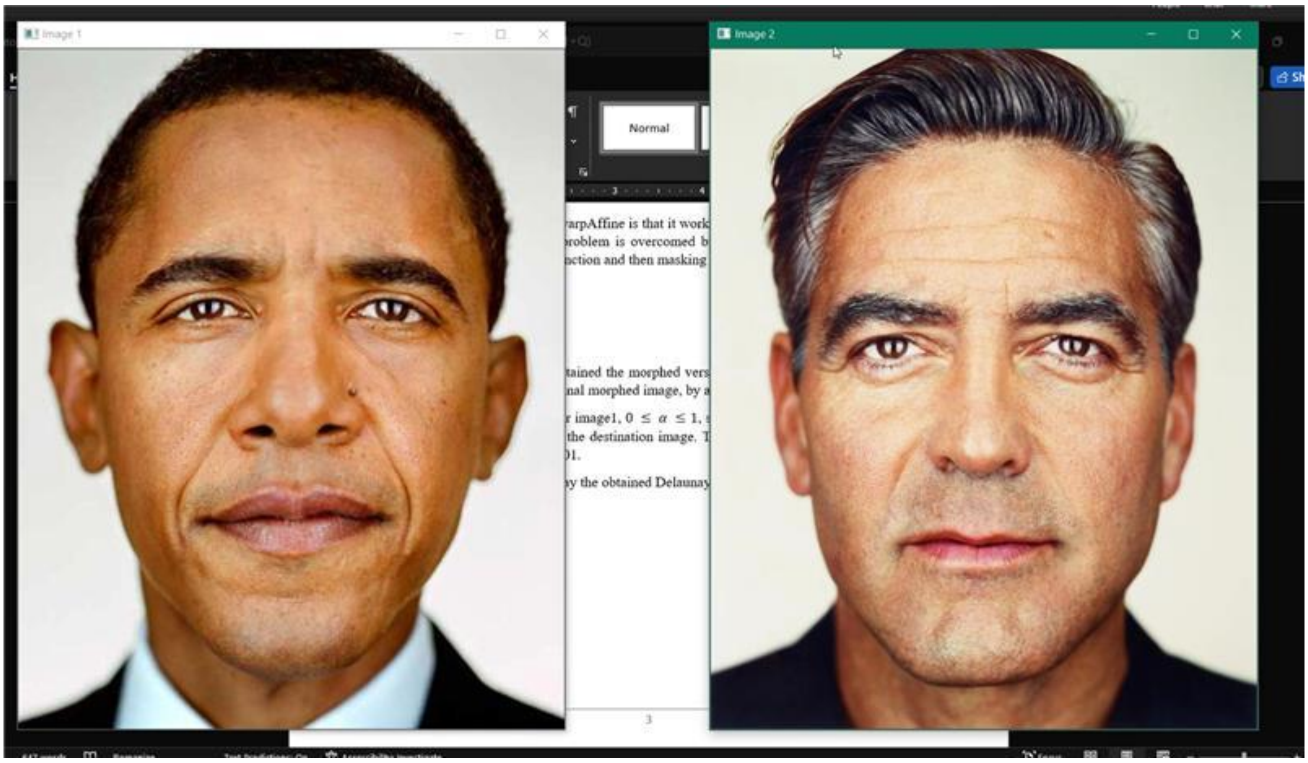
Image Morphing

Students: Ivan Alexandru-Ionuț & Filip Denisa-Mariana
Group: 30432
Semigroup: 1
Teaching Assistant: Oniga Florin
Faculty of Automation and Computer Science, TUCN
2021-2022

Contents

Algorithm Steps

We begin the morphing process by selecting the two portraits we want to blend. In this example, we have chosen an image of Barrack Obama and George Clooney. The two input images must have the same size for the algorithm to work.



1. Find Corresponding Points

For both portraits, we choose a set of identical points, usually key features, according to which the images will be morphed. By identical we mean that the two sets of points (one for each input image) must be relatively in the same positions and must have the same size. We do this by using a mouse callback function and we store the chosen points in two vector data structures.

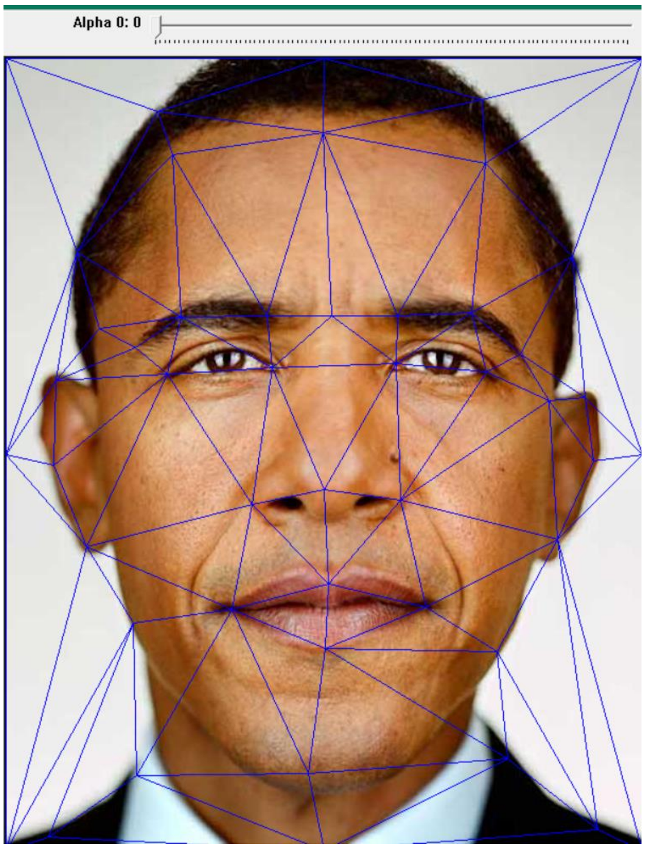
We automatically add 8 points (4 in each corner, and 4 in the middle of the sides), to contain the entire image.

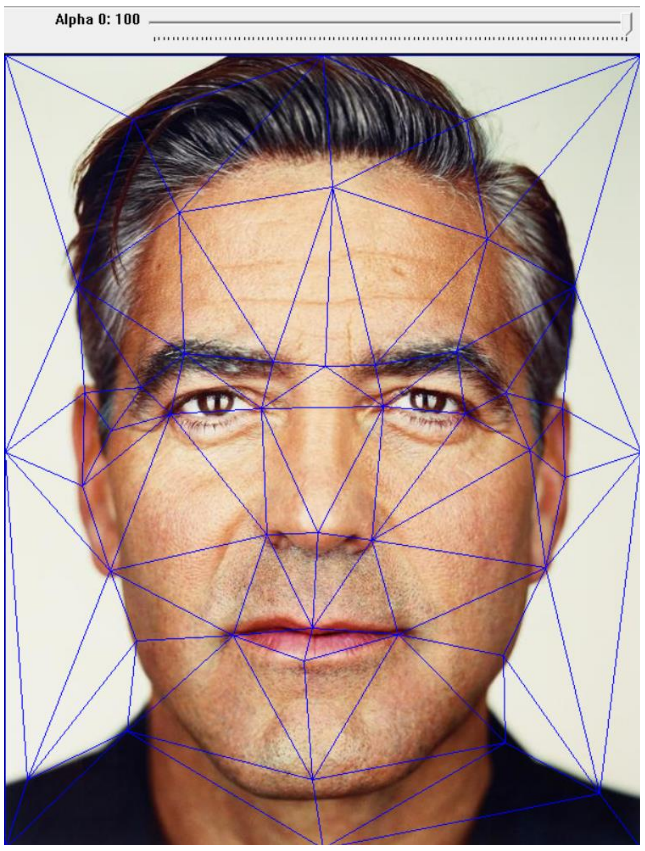
2. Delaunay Triangulation

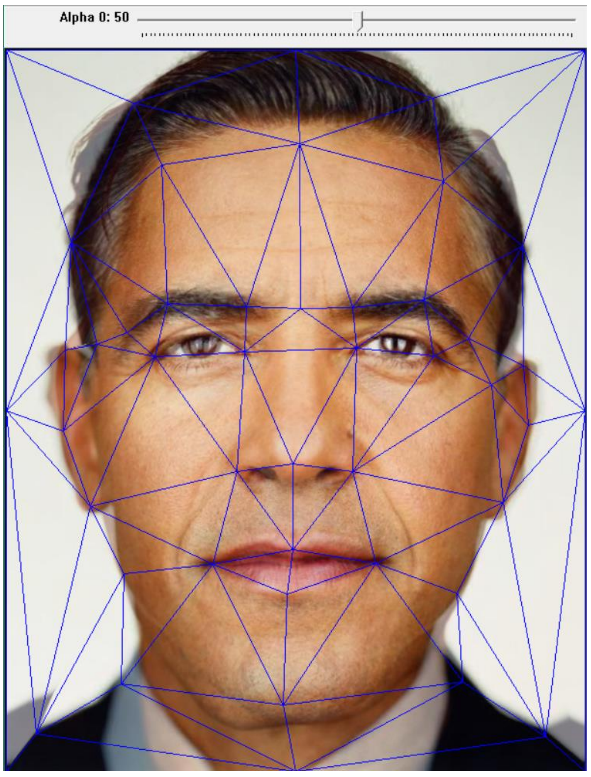
In a Delaunay triangulation, triangles are chosen such that no point is inside the circumcircle of any triangle.

At the previous step, we obtained two sets of X points. We compute the Delaunay triangulation over one set of points, obtaining one set of Y triangles connecting all points.

Moreover, we also need to find a third set of X points for the final morphed image, which is obtained by combining the first two sets of points with an *alpha* value.







3. Affine Transformation

For each of the Y triangle from the first image and the corresponding triangle in the morphed image, we need to compute the affine transform that maps the three corners of the triangle in image 1 to the three corners of the triangle in the morphed image.

At first, we used the predefined getAffineTransform function from the OpenCV library, which takes as input two vectors of points containing the corners of the triangles, and returns a 2x3 matrix containing the affine transform.

The 2x3 affine transform matrix looks like:

$$\begin{matrix} a & b & tx \\ c & d & ty \end{matrix}$$

Where the coefficients a, b, c and d are responsible for the scale, rotation and shear transformations, and the tx and ty coefficients perform the translation of the source point to the destination point.

Afterwards, we went ahead and implemented our own version of the getAffineTransform function. We knew the source points (3 points, one for each corner of the source triangle), denoting the matrix S, and the destination points (3 points, one for each corner of the destination triangle), denoting the matrix D. We needed to obtain the matrix M from the equation S * M = D.

The matrix M is therefore obtained from the equation $M = S^{-1} * D$. From here, we finally obtained the 2x3 affine transformation matrix.

4. Image Warping

Now that we have the affine transform matrix that maps one pixel to another, we need to apply it to every pixel from the source triangle in order to obtain the pixels from the destination triangle.

This is achieved with the function warpAffine, which takes as parameters the source imge, destination image, the affine transform matrix, the size of the destination image and a few flags that can change the way the warping works.

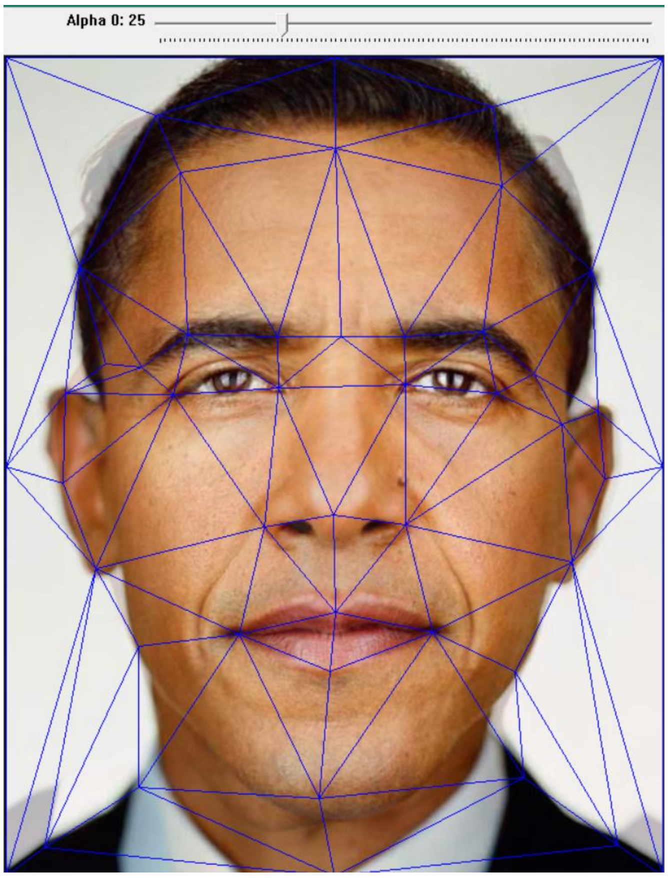
The problem with the function warpAffine is that it works only with rectangular images, not with triangles as we would need. This problem is overcomed by creating a bounding rectangle over a triangle, applying the warpAffine function and then masking all pixels outside the triangle.

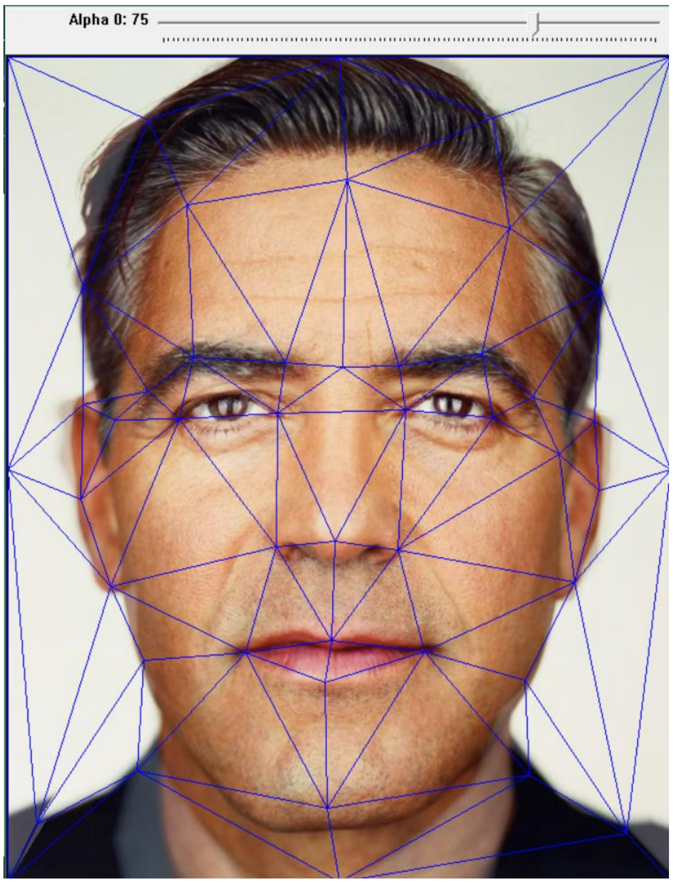
5. Alpha Blending

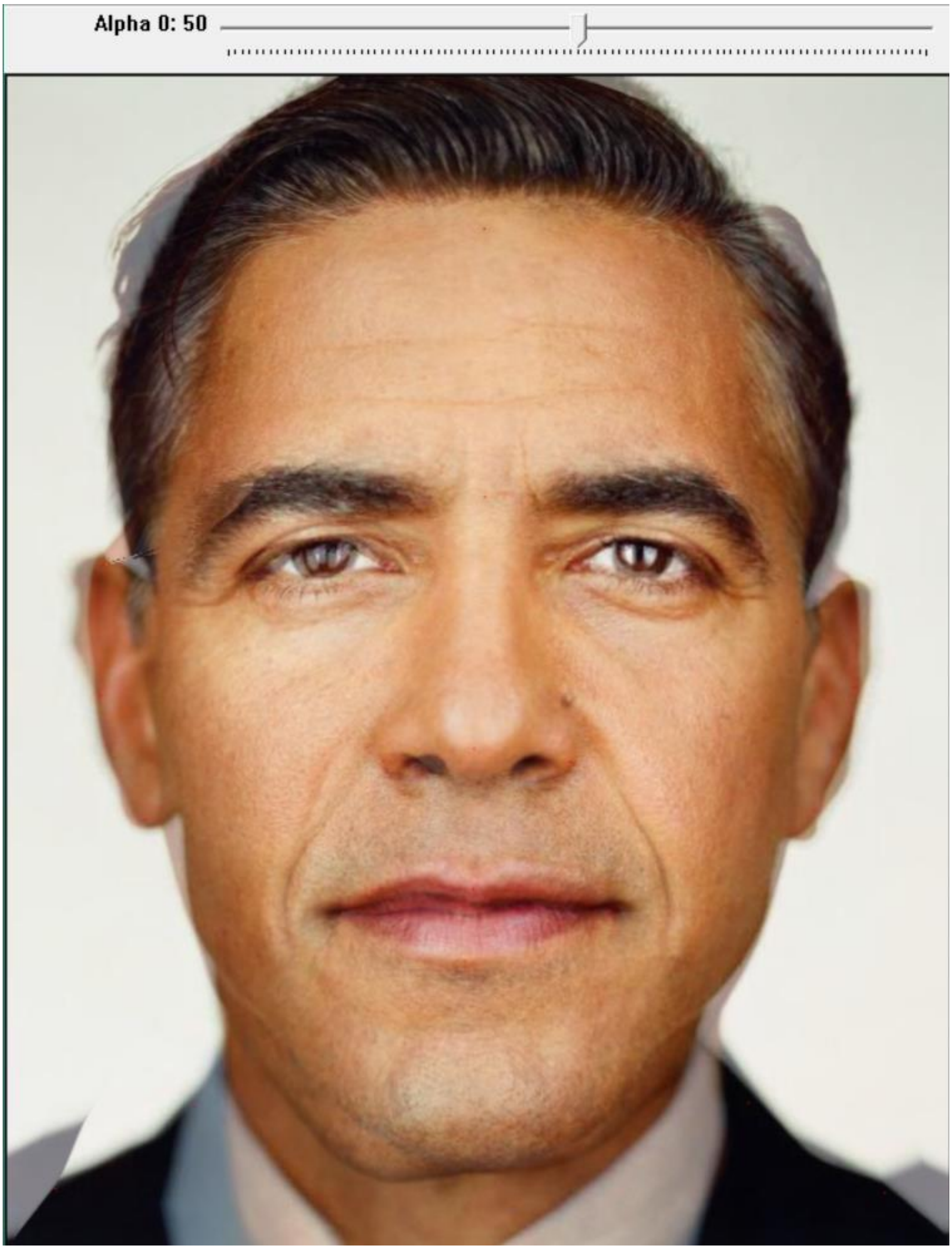
Using the previous steps, we obtained the morphed version of image1 and image2 and the only step left to be done is to obtain the final morphed image, by alpha blending them.

Alpha is the ratio of image2 over image1, $0 \leq \alpha \leq 1$, so that when α is 0 we obtain the source image, and when α is 1, we obtain the destination image. The value of α is obtained with a slider, which increases/decreases α with 0.01.

We also have the ability to display the obtained Delaunay triangles, by toggling their display with a click over the image.






Alpha 0: 50

Problems

If we do not select approximately the same points in the two input images, we obtain something like this, in which the triangles are mixed up and the warping is no longer accurrate.