Morphing using Delaunay Triangulation

Multimedia Processing and Applications (6th Semester)

Submitted to Dr. Anukriti Bansal

Project Aim

This project required us to implement image morphing between two images using general predefined functions of opencv and python.
In addition a personal aim was to create only those functions from scratch which were not directly related to the implementation of this project.

Project Members

The team members are:

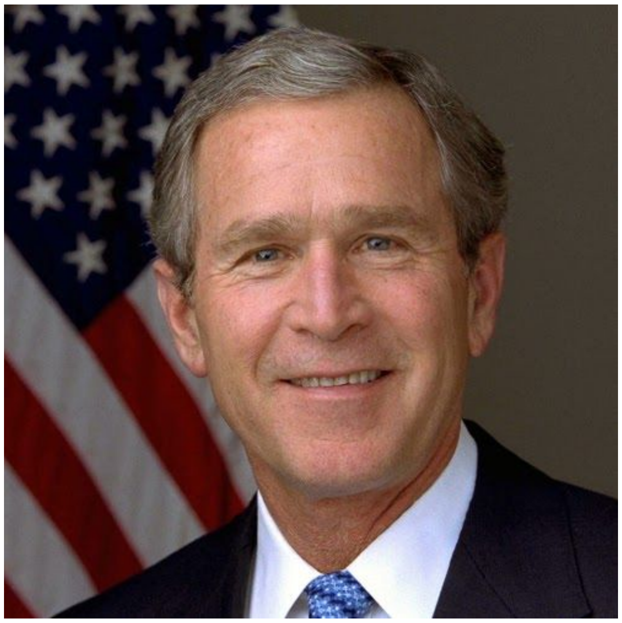
1. Rivanshu Goyal (17UCS132)
2. Daksh Balyan (17UCS049)

Overview

Morphing is an image processing technique which in layman terms can be explained as transformation of one image to another with a smooth transition in between. This smooth transition is achieved by creating a series of in between images which when seen in sequence creates an effect of metamorphosis of one image to another. As the morphing process progresses the initial in between pictures are similar to the starting image, the middle frame is an average of the starting image and final image while the later frames are more similar to the final image.

Applications of image morphing are most often seen in the animation industry and in creating special effects in the entertainment industry. The video games we play or see around us today also use image morphing heavily. Apart from entertainment, image morphing has medical applications too, where the success or failure of a physiological healing process can be easily visualised due to a dynamically morphed video. This enables the doctors to show the process to the patients, other doctors for consultation or for education purposes for young doctors. Machine Design is another field where image morphing has various applications.

Input Images



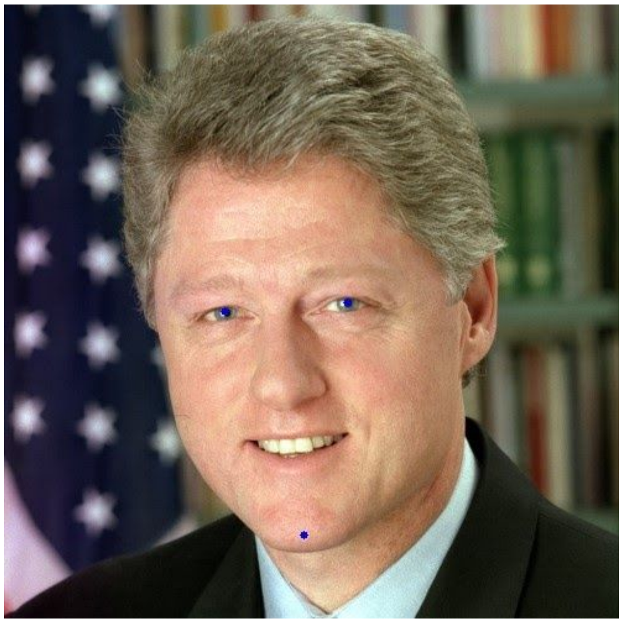
This is the initial image “Clinton.jpg”

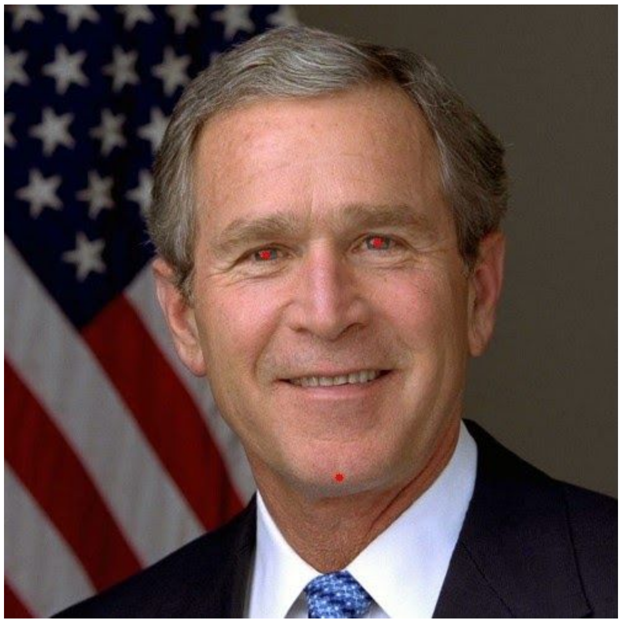
This is the final image “Bush.jpg”

The dimensions of both the images is 500 x 500 px

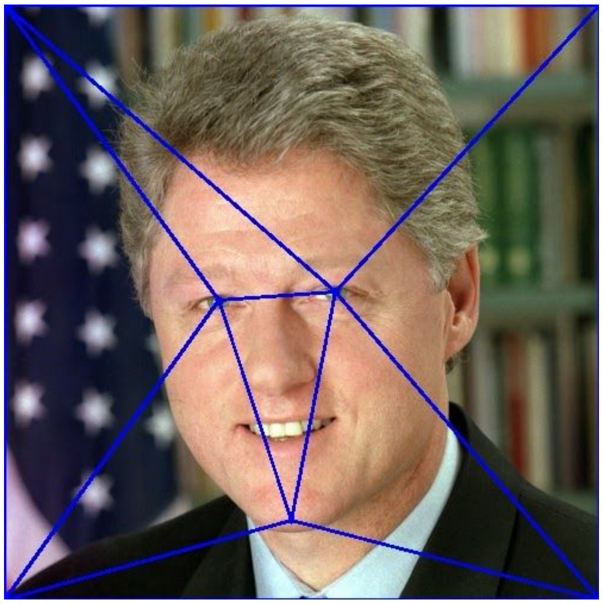
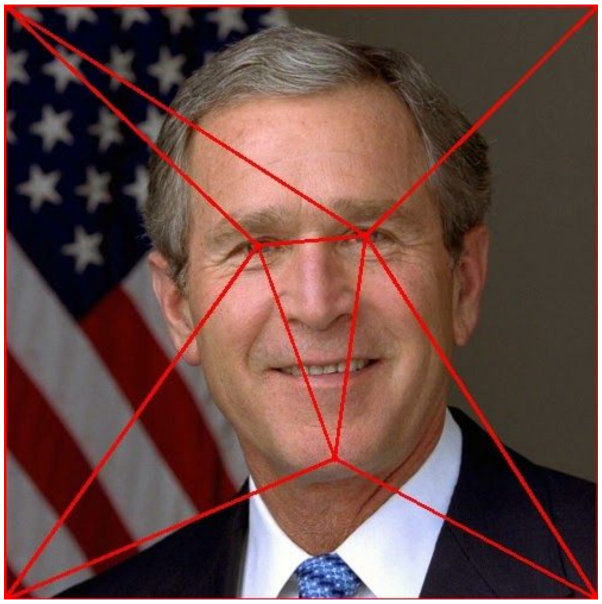
Methodology

1. OpenCV method - ‘cv2.setMouseCallback’ was used to detect and handle mouse events, such as the left mouse click button, which was needed to store the control points of the images as a list of tuples.





2. The number of control points is user dependent but we chose to go with 3 control points.
3. After selecting the control points, triangulation was done only by implementing the delaunay triangulation algorithm.
4. Images depicting triangulation are then shown and saved, using the ‘cv2.imshow’ and ‘cv2.imwrite’ methods of openCV.

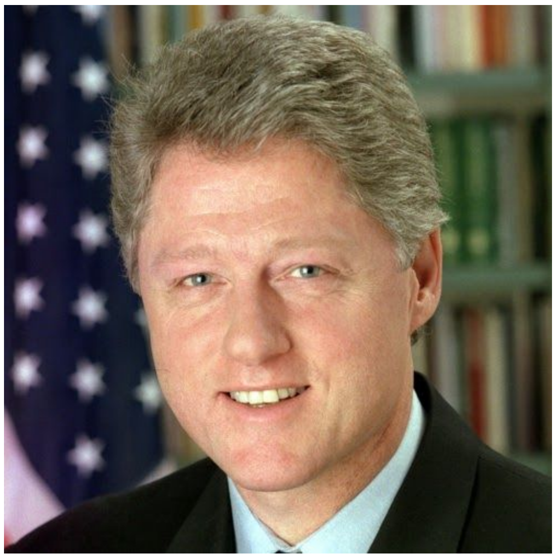
5. The number of intermediary pictures or frames (including the initial and final image), here, were selected to be 100.
6. Then an array with 100 empty frames was created using the **numpy** library.
7. The first and last frame was set as the initial and final image respectively.
8. Now to find out the control points of the intermediate frames we linearly interpolated the control points of the first and last frame.
9. After finding out the control points of the intermediate frames, their corresponding triangle vertices were computed and stored as a list of tuples.
10. Then affine basis was calculated and stored for each triangle of each frame.
11. In each intermediate frame (**i**th) for every pixel ($\mathbf{p}_i$) the following steps were followed:
    11.1. determine which triangle pixel ($\mathbf{p}_i$) belongs to in the **i**th frame
    11.2. Using the affine basis corresponding to the triangle which the pixel ($\mathbf{p}_i$) belonged to, affine coordinates ($\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$) were found out for that pixel ($\mathbf{p}_i$) of the **i**th frame
    11.3. Now by using this triangle, corresponding triangles were selected in the first and last frame (the initial and final image respectively), let us name them as **A** & **B**.

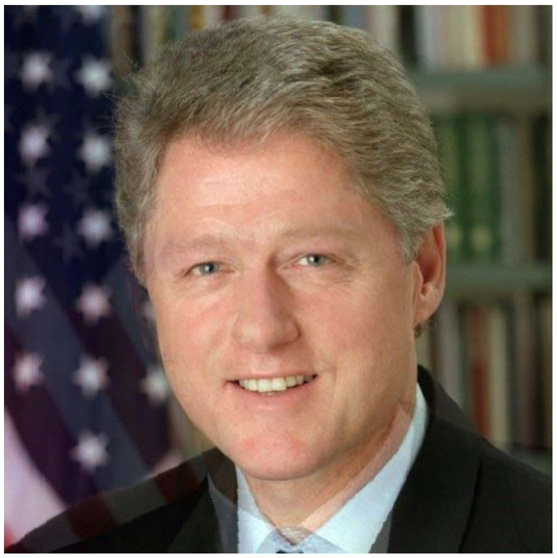
11.4. For triangles **A** & **B**, using the affine coordinates (which are invariant) and affine basis of both triangles, pixels $\mathbf{p}_0$ & $\mathbf{p}_n$ (in the first and last frame) corresponding to pixel $\mathbf{p}_i$, were calculated.

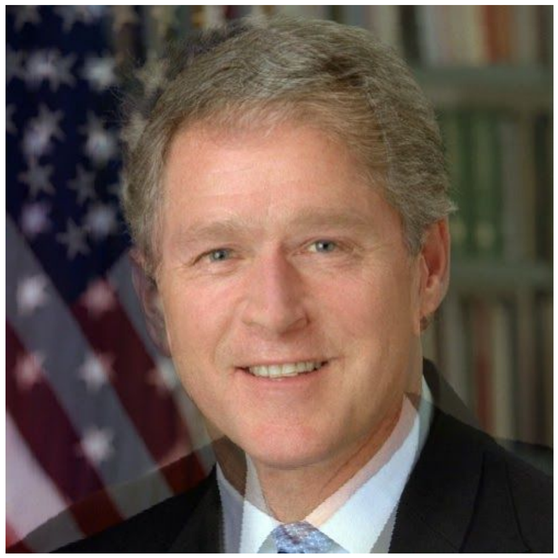
11.5. The color intensities of $\mathbf{p}_0$ & $\mathbf{p}_n$ were then interpolated to $\mathbf{p}_i$

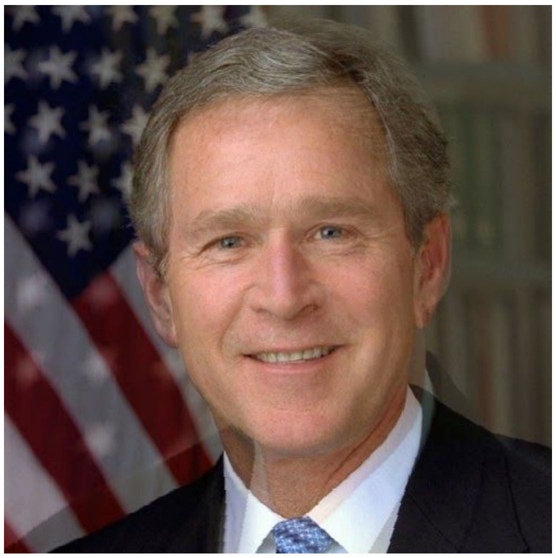
12. Finally all the frames were then saved and a video was made showing the metamorphosis of the initial image to the final image.

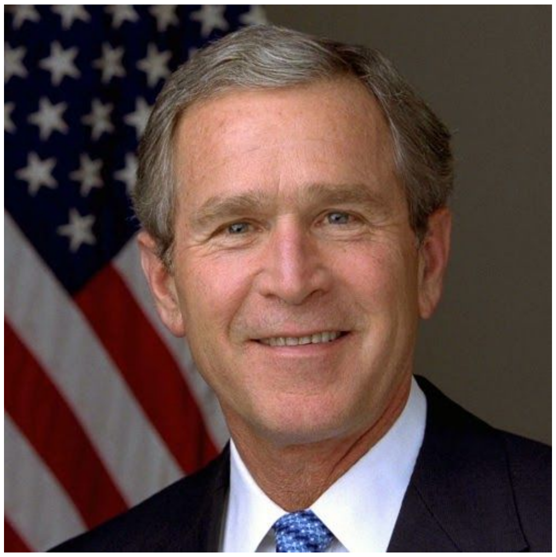
Output Images/ Frames


Initial Image


25th Intermediary Frame/ Image


50th Intermediary Frame/ Image


75th Intermediary Frame/ Image



The final Image/ last frame

Conclusions

Through this project we learned about feature preserving image morphing using affine transformation. We learned about delaunay triangulation and implemented it using the Bowyer–Watson algorithm.

There were also some challenges we faced, which are :

1. We didn’t know at first that the coordinate system in numpy arrays and the one used by openCV are opposite of i.e. (x,y) coordinate in the numpy array is (y,x) in openCV. However a very small problem, it took us quite a while to find out this bug in our code. To solve this we created a function “reverse” which returned the (x,y) tuple in (y,x) format wherever desired.
2. We observed that the calculation of delaunay triangulation should only be performed on one image and then mapped to the other image. The reason behind this conjecture is that individual independent Delaunay triangulation of both the images may lead to unequal no. of triangles in the images which will then in turn bring difficulties in the morphing algorithm.

References

1. http://paulbourke.net/papers/triangulate/
2. https://en.wikipedia.org/wiki/Bowyer%E2%80%93Watson_algorithm
3. Class Notes