MPA PROJECT REPORT

VI Semester 2020

Feature Preserved Morphing using Delaunay Triangulation

PREPARED FOR

Dr. Anukriti Bansal

PREPARED BY

Ansh Mittal	-	17ucs028
Anshul Jain	-	17ucs029

Table of Contents

Introduction

Meaning

The word Morphing originated from the Greek word "metamorphosis," which means “to transform”. Now, this refers to the smooth change from one thing to another through a seamless transition.

In our case, morphing means a process of video production by a series of images created by a smooth transition from the initial image to the final image.

Applications

Morphing is widely used to bring special animation effects from one frame to another smoothly to the viewer. It is mainly used in -

- Animations in
    - Movies
    - Animated games
    - Presentations
    - Advertisement
    - Electronic book illustrations
- Computer-based training
- Education purposes
- Medical Purposes
    - Healing of Body
    - Maintaining track of shape change

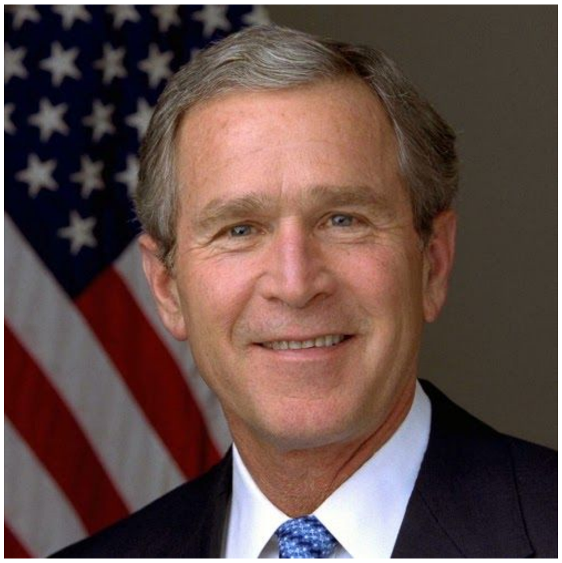
Input Details

We take two images as input and applied morphing to transit from the first image to second. The details of images are described in a table below -

Name	Length	Width	Filetype	Coloured
Bush	500	500	.jpg	✓
Clinton	500	500	.jpg	✓

As we can see all the images are of the same dimension ie. 500*500 and having 3 channels RGB (Red, green, blue) but as we are using Python language, It takes the channels in the order of BGR (Blue, green, red). So the final shape of images becomes (500*500*3).

Clington.jpg



Bush.jpg

Methodology

This part talks about the approach we followed to get the results. The steps we have taken and stepwise output we got.

Libraries Imported

We imported the following libraries-

1. NumPy

Numpy is used for performing fast operations on the array. Numpy operations are faster than normal python for various reasons, primary being its C++ base for calculations. Our previous implementations were taking too much time without it.

2. OpenCV

OpenCV is used to input the image and show the image using functions like imread() and imshow(). It is also used to take input feature points by the left mouse button and to draw lines to show triangulations.

3. OS

OS library is used for interacting with the operating system. We used this library just to create the dump folder for the intermediate frames generated during the morphing process to convert them to make a video.

Algorithm Details

Selecting Control Points

We have read the initial image and final image as img1 and img2.

- Select control-points (non-collinear) from both the images. Note that the number of points should be the same in both img1 and img2 and should be in the same order(relatively). Let's call these sets of points as *init_ctrl_points* and *final_ctrl_points* respectively.
- Append all the corner points of the images in the sets also. So the final length of sets is 4 plus the number of points marked.

Delaunay Triangulation

For Delaunay Triangulation we implemented Bowyer-Watson algorithm. It is an incremental algorithm. It adds one point at a time to a valid mesh of triangles.

We have used connectivity of infected triangles to locate bad triangles which decreased the complexity of the algorithm from O(n^2) to O(n). Further to avoid calculation of circumradius and circumcircle for each triangle we have stored them in a list so that they only get computed once when corresponding triangle is formed. Calculation of circumcenter is done using barycentric coordinates solved by cramer's rule for efficient calculation. The algorithm is explained below.

- The foremost task is initialising mesh with a giant super triangle which can include all the points. We'll delete any triangle from the triangle list if it is found sharing an edge with the super triangle in the end. For the super triangles we chose four points that lie on a circle with default radius = 9999 and default center (0,0) and chose 2 pairs of triangles named in counter-clockwise fashion from this quadrilateral as shown.

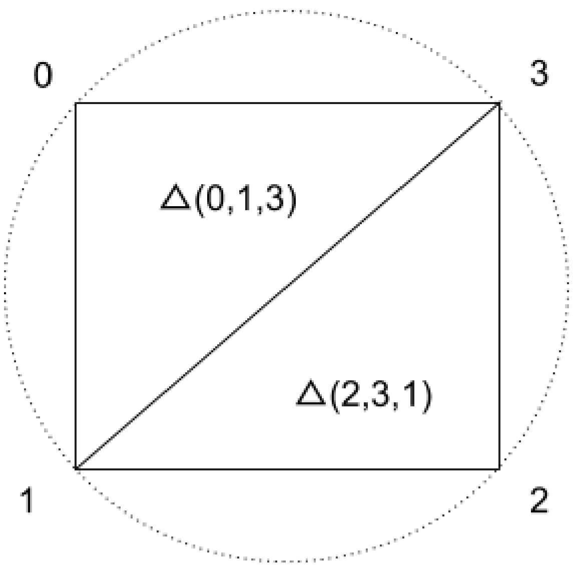


Fig 1. Initial Super Triangles

- Then for each point which is added to this mesh we find all those connected triangles whose circumcircle includes this point. This part has complexity O(n). It iterates on all triangles and checks if the euclidean distance between circumcenter and this new point is greater than circumradius. If it is not, the triangle is added to the list of bad triangles.
- Now calculate the boundary of the infected area. This area is a polygon. For this part complexity is O(n). Start with the first bad triangle and its first edge and repeat until we get new boundaries. At each step check if the triangle on the other side of the edge is a bad triangle or not. If it is one indeed then

switch to that triangle for iteration and make the edge equal to the next counter-clockwise edge in this triangle. In the other case store this edge and opposite triangle in the boundary list.

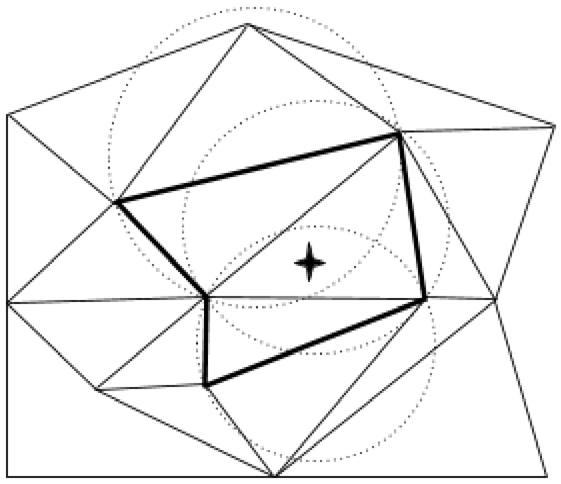


Fig 2. Extraction of Boundary containing bad triangles

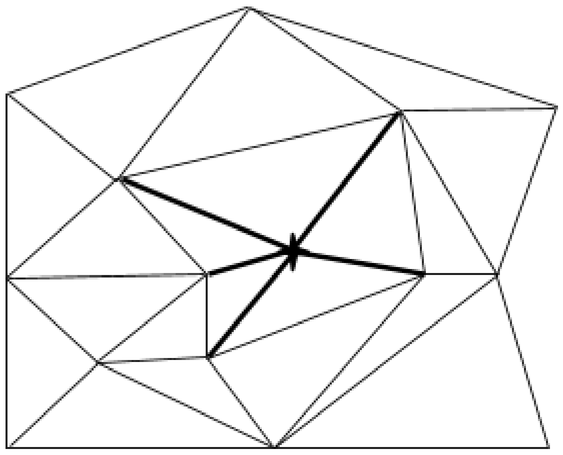


Fig 3. Final Triangulation

- Now the triangles in the bad triangles list are deleted so that the polygon has no edges on its inside. Then a triangle is constructed from each edge of this polygon and the new point (Fig 3). Each point addition gives a net gain of two triangles.

Linear Interpolation

Frames Initialization

- Take a number N from the user which denotes the number of frames in the video.
- Makeg img1 as 0th frame and img2 as the (N-1)th frame, total being N frames.
- Initialize all the other frames as images like img1 but all pixel values zero.

Frames Generation

- Iterate on each frame.
- Calculate delta using the formula given below.

$$\Delta L = (P[N-1] - P[0]) / (N-1) \qquad \ldots(1)$$

Where $P[i]$ denotes the control points in the ith frame and ΔL denotes the average difference of control points.

- Calculate the control points of kth frame using the initial image and delta as follow -

$$P[k] = P[0] + K * \Delta L \qquad \dots(2)$$

- For each triangle, take the affine basis of the triangle in kth frame, initial frame and final frame. Lets name them (e_1,e_2), (ei_1,ei_2) and (ef_1,ef_2) respectively.

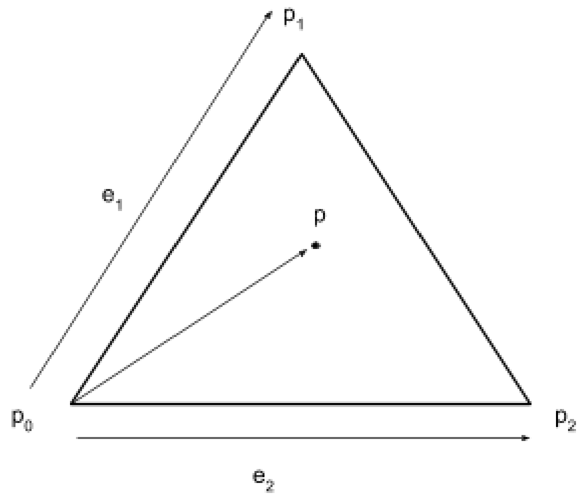


Fig 4. Affine basis e1 and e2 with P_0 as reference point

- Iterate each pixel in triangles by the following steps -
    - Find the bounding box of the triangle.
    - Check if the pixel is inside the triangle. For this we check if the sum of the areas of triangles formed by the point with three vertices is equal to the area of the original triangle.
- Find the affine coordinates α and β by solving the following equation-

$$P[k] - P_0[k] = \alpha * e_1[k] + \beta * e_2[k] \qquad \dots(3)$$

Where, k denotes the kth frame which we are currently on.

$P[k]$ denotes the current pixel coordinate in the kth frame.

$P_0[k]$ denotes the reference point for affine basis vectors of the triangle in kth frame containing the pixel.

$e_1[k]$ and $e_2[k]$ are the affine basis for kth frame as defined earlier

- Use the same affine coordinates earlier and find the coordinates of pixel in initial and final frame as -

$$P_i[k] = P_0[0] + \alpha * e_1[0] + \beta * e_2[0] \qquad ...(4)$$
$$P_f[k] = P_0[N] + \alpha * e_1[N] + \beta * e_2[N] \qquad ...(5)$$

- Assign the weighted sum of colour values of pixel at $P_i[k]$ and $P_f[k]$ in initial and final images respectively to pixel at $P[k]$ with weights as

$$img(P[k]) = \frac{N-1-k}{N-1} * img1(P_i[k]) + \frac{k}{N-1} * img2(P_f[k]) \qquad ...(6)$$

Video Preparation

We store all the intermediate frames in a folder named Morphing_Output and then to generate a video from it, we used the following command -

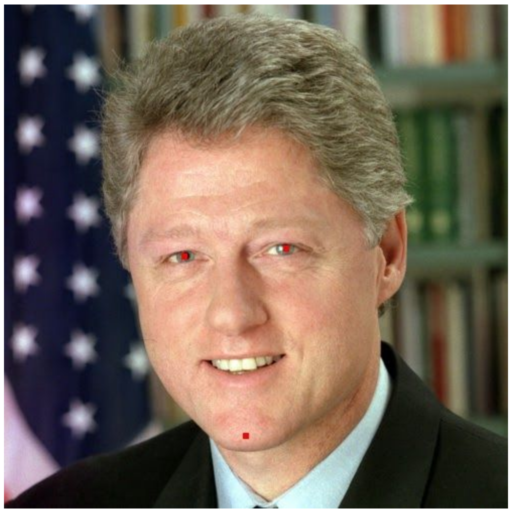
```
ffmpeg -framerate 30 -i ./Morphing_Output/Frame_%d.jpg
Morphing.mp4
```

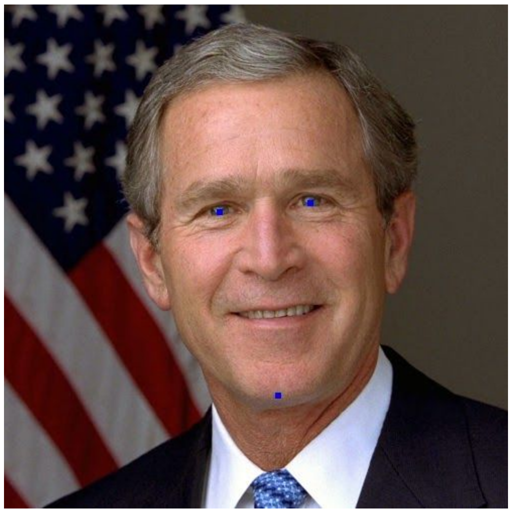
In which -

1. ./Morphing_Output/Frame_%d.jpg - denotes path + name + file type of the image.
2. 30 - denotes the number of frames per second
3. Morphing.mp4 represents the name of the resulting video.

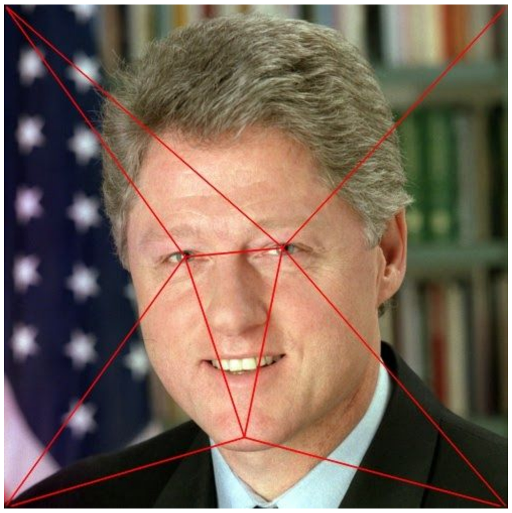
Intermediate Images

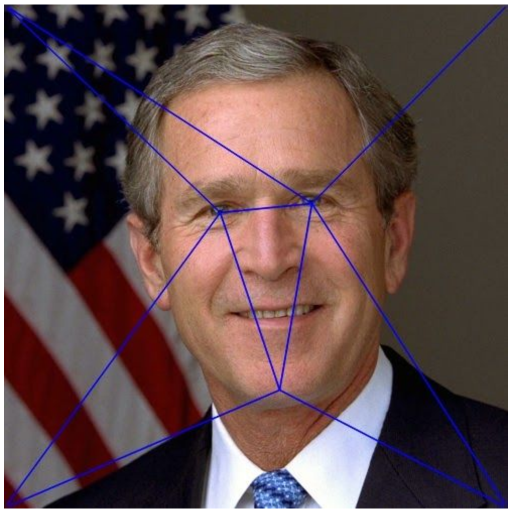
Images with Control Points



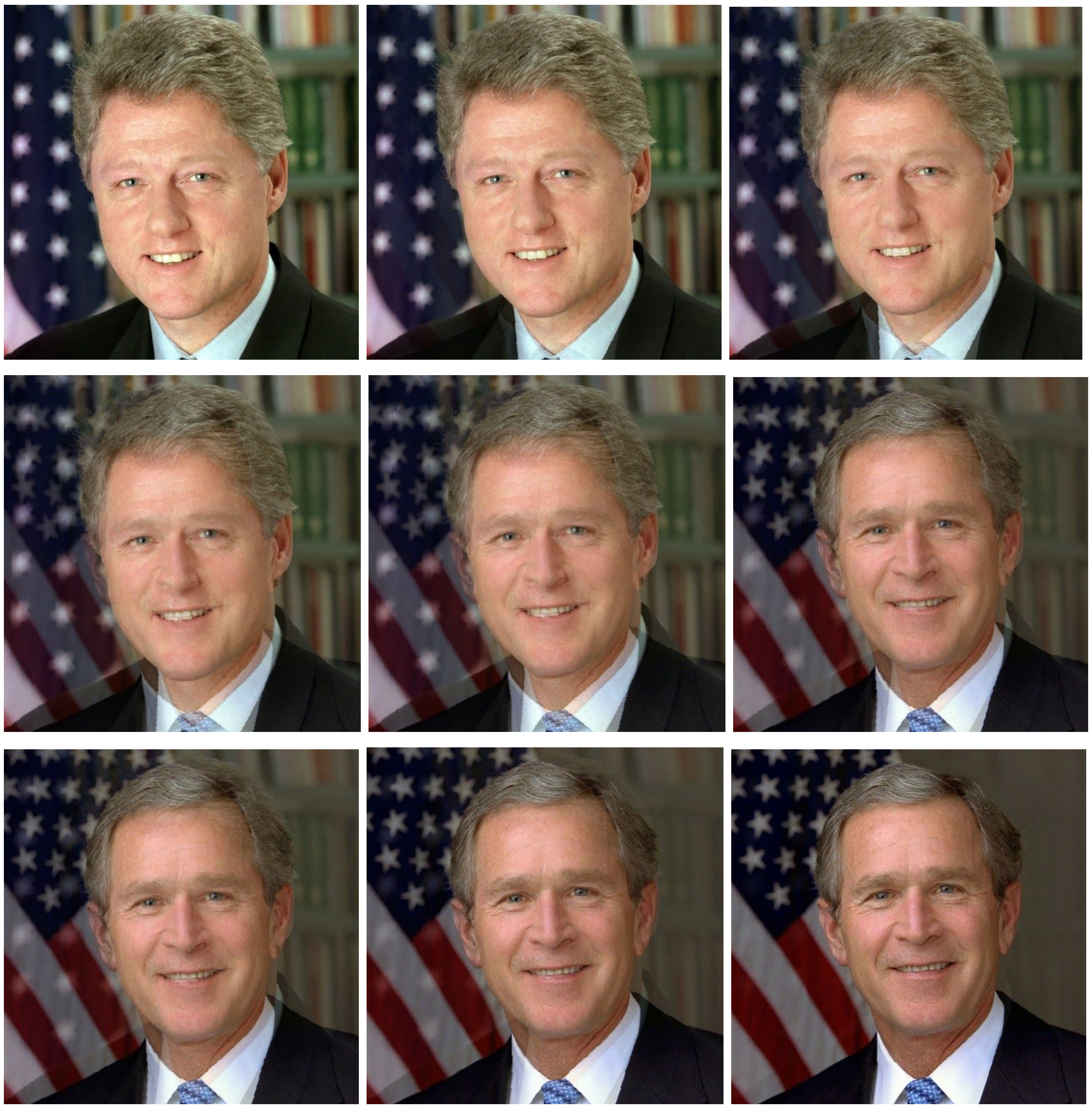


Images with Triangulation





Output



Conclusion

Overcoming failed steps

We encountered some problems while implementing but did a workaround to solve them as described below.

- We initially tried to do all code in clean python without importing any additional library overhead but doing so increase execution time heavily. So we then imported the Numpy library for performing all mathematical operations. As Numpy performs them on C++ background it reduces 50% execution time.
- Then we encountered another problem of accessing elements of the image array. The Numpy or Python access image array by first its row then by column, unlike OpenCV. This although confusing is needed for their background operations to work efficiently. We handled it by keeping notation in the way Numpy (or Python) access them as we would be needing them the most. We changed them only when we performed on OpenCV functions.
- Lastly we faced the problem of deciding colour of control points to show when picking them by mouse in monochrome mode. If we chose markers to be white colour they might not be visible when picking over a lighter region similarly if we chose black colour then it would become hard to see the pointer in darker regions. So we made the marker with black border containing white pixels. This also looks good visually.

Scope of Improvements

- A significant improvement in Delaunay Triangulation is to use better structures that allow spatial search(like QuadTree) and presort the to be added points by one coordinate, the one having the largest range should be chosen(say *x axis* is sorted in ascending order). Then if a point is at a distance greater than circumradius from a circumcenter of a triangle, then that triangle will be never needed again as further points would never be in the interior of that triangle's circumcircle. Then this can reduce the complexity further. We didn't make these changes as it was already sufficiently fast for us as most of the time was consumed by the frame calculation part so we have to focus on that more.
- In cases when many points(>37) were selected for Delaunay triangulation we didn't get results like what was given by SciPy's Delaunay function for the

same input. We were unable to figure it out so we state here that our Delaunay function fails if more than 37 points are given to it.
- If someone wrongly marks any control point then he has no option other than rerunning the whole script. This can be handled by providing the user an option in the terminal itself to remark the control points. Since only we and our instructor are only using it for now, we can save this part in future work which can be done when making this project public.

References

- Class Notes
- OpenCV Documentation
- Numpy Documentation
- OS Documentation
- Argument Parser in Python
- Circumcircle of a triangle using Barycentric Coordinate System
- Find if a point lies inside a triangle
- Area of a triangle using ShoeLace Formula
- Bowyer-Watson Algorithm
- Just for comprehending Delaunay Triangulation(Video, Papers, Implementation)
- For comparing Delaunay