

A Project Report on Morphing

Submitted To:
Dr. Anukriti Bansal

Submitted By:
Mayank Aggarwal (17dcs006)
Harshit Garg (17ucs064)



Credentials

1. NPTEL video on Delaunay Triangulation <https://www.youtube.com/watch?v=tWf1z9i-ORg>
2. WikiPedia for circumscribing triangle and reading text on delaunay

https://en.wikipedia.org/wiki/Circumscribed_circle#Triangles

https://en.wikipedia.org/wiki/Delaunay_triangulation

Contents

- ❑ Aim
- ❑ Overview
- ❑ Input Images
- ❑ Algorithmic Steps
- ❑ Code
 - ❑ Important Python Libraries used
 - ❑ Functions created and their definitions
 - ❑ *circumcenter(p1 , p2 , p3)*
 - ❑ *dist(p1 , p2)*
 - ❑ *area(x1, y1, x2, y2, x3, y3)*
 - ❑ *delaunay(pts , coord)*
 - ❑ *combinations(coord)*
 - ❑ *get_Triangle_List(coord)*
 - ❑ *CallBackFuncForimg1()*
 - ❑ *CallBackFuncForimg2()*
 - ❑ *getcoord()*
 - ❑ *draw_delaunay(img , triangleList , delaunay_color)*
 - ❑ *showTriangulated(img1 , img2)*
 - ❑ *isInsideTriangle(p1,p2,p3,x,y)*
 - ❑ *get_affine_basis(coord)*
 - ❑ *get_intermediate_triangles(srcTri , destTri , k , n)*
 - ❑ *checkRange(sx , sy , dx , dy)*
 - ❑ *morph(no_of_intermed)*
- ❑ Reading of input images and resizing them to same size
- ❑ Getting control points on images using mouse click
- ❑ Triangulating the images and applying affine transformation
- ❑ Results and Conclusion

Aim

Take two images and select control points in both the images and triangulate them after that linearly interpolate the coordinate values of control points and find out the affine coordinates of non-control points in the intermediate frame. Now use these affine coordinates to find out the location of non-control points in the first and second images. Use color interpolation to assign color values to pixels in the intermediate frame then generating and saving all the intermediate frames so that they can be viewed individually and also need saving the source and destination images after triangulation.

Overview

Image morphing can be defined as a technique which constructs the sequence of images depicting the transition between them. The method that is used in this project involves using Delaunay Triangulation and Affine transformation.

Firstly the images are divided into several parts by selecting different points on it. These points on the image are called control points. The control points are used in order to apply the Delaunay triangulation as well as the Affine transformation on the images on them. The details of the methods are explained in the Algorithm section.

Morphing is mainly employed in the field of animations and special effects. In the present day there exist many software like after effects, nuke etc. These software can also be used by people who don't know coding.

Input Images



Clinton Size 500x500



Bush Size 500x500

Algorithmic Steps

Step 1

Reading images and making their size the same if they are of different sizes. In this we have taken two images of size 500x500

Step 2

Getting control points of both the images using mouse clicks in the same order as taken in 1st image. You can take as many control points you want at the output. For simplicity, in this report, we have explained terms by taking a total of 7 points (consisting of 4 corners points 1 left eye , 1 right eye and 1 center of the smile).

Step 3

Triangulating both the images using delaunay triangulation and saving the images. This was done by the following steps:

1. make all the possible combination of the triangles from the control points that is 35 in our case and then rejecting that combinations which are collinear
2. now selecting a combination and getting its circumcenter
3. if any other control points lie inside that circumcircle then reject that triangle otherwise accept
4. repeat step 2 until all the combinations are done.

Step 4

To do affine Transformation from source image to destination image by making some intermediate images in which pixel values are calculated by combination of pixel value in source and destination by reverse warping otherwise we will get holes in the images. The steps which we followed are:

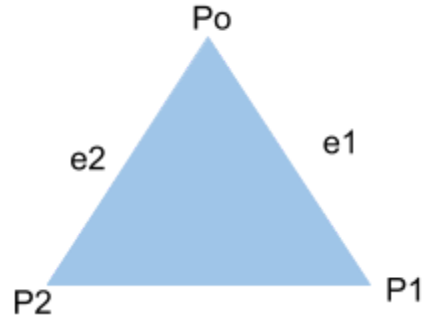
1. Select a triangle in intermediate image and calculate affine basis in source, destination and intermediate images corresponding to that triangle

Affine basis are calculated as follows:

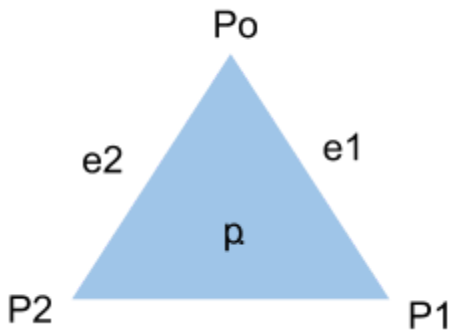
Affine basis are $\overline{e1}$ and $\overline{e2}$

$$\overline{e1} = \overline{P1} - \overline{Po}$$

$$\overline{e2} = \overline{P2} - \overline{Po}$$



2. Calculate affine coordinates alpha and beta using affine basis for each point in that triangle in intermediate image and they are calculated as follows



A point p inside a triangle can be written as

$$\overline{p} = \alpha \overline{e1} + \beta \overline{e2}$$

Where α and β are affine coordinates

3. For all the points lying in that triangle in the intermediate image, find corresponding point in source and destination image by using α and β as

Since affine coordinates doesn't change so

$$\text{For source } \overline{P}_{source} = \alpha \overline{e1}_{source} + \beta \overline{e2}_{source}$$

$$\text{For destination } \overline{P}_{dest} = \alpha \overline{e1}_{dest} + \beta \overline{e2}_{dest}$$

Where \overline{P}_{source} = point corresponding to point \overline{p} (intermediate image) in source image

$\overline{e1}_{source}$ and $\overline{e2}_{source}$ are the affine basis of the corresponding triangle in the source image.

\overline{P}_{dest} = point corresponding to point \overline{p} (intermediate image) in destination image

$\overline{e1}_{dest}$ and $\overline{e2}_{dest}$ are the affine basis of the corresponding triangle in the destination image.

4. Pixel value at \bar{p} in intermediate image is calculated by using the formula mentioned below.

$$\overline{Pix} = \left(\frac{n-k}{n}\right) \overline{Pix1} + \left(\frac{k}{n}\right) \overline{Pix2}$$

Where

\overline{Pix} is calculated pixel value in kth intermediate image at point \bar{p}

$\overline{Pix1}$ is pixel value in Source image at point \bar{p}_{source} corresponding to \bar{p}

$\overline{Pix2}$ is pixel value in Destination image at point \bar{p}_{dest} corresponding to \bar{p}

Code

Important Python Libraries used

CV2, NumPy, math, sys

- CV2 is an openCV library used to read, write, resize, to show images and to draw line and circle on an image only.
- NumPy is a package in python which is used to perform scientific computing. We have used this for making 2d image arrays for intermediate images.
- math is a module in python used to access functions of mathematics. We have used this to find square root only.
- sys allows us to use the functions of the python interpreter we have used sys.get names of the images from the command line argument.

```
import cv2
import numpy as np
import math
import sys
```

Functions Created and their definitions

circumcenter(p1,p2,p3)

Use

To Calculate the circumcenter of the three points given.

Arguments

This function takes 3 arguments as p1, p2 and p3.

return type

coordinate of the circumcenter of the point made by p1, p2 and p3.

```
def circumcenter(p1,p2,p3):

    #using the mathematical formula for circumcenter for three points
    #Source for formula wikipedia

    d = 2 * (p1[0] * (p2[1] - p3[1]) + p2[0] *
              (p3[1] - p1[1]) + p3[0] * (p1[1] - p2[1]))

    det_p1 = p1[0]**2 + p1[1]**2
    det_p2 = p2[0]**2 + p2[1]**2
    det_p3 = p3[0]**2 + p3[1]**2

    cx = (det_p1 * (p2[1] - p3[1])
          + det_p2 * (p3[1] - p1[1])
          + det_p3 * (p1[1] - p2[1])) / d

    cy = (det_p1 * (p3[0] - p2[0])
          + det_p2 * (p1[0] - p3[0])
          + det_p3 * (p2[0] - p1[0])) / d

    return (cx, cy)
```

dist(p1,p2)

Use

To Calculate distance between two points

Arguments

This function takes 2 arguments as p1 and p2.

return type

float value of distance between p1 and p2 calculated using euclidean distance formula

```
def dist(p1,p2):
    # Euclidean Distance formula
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2 )
```

area(x1,y1,x2,y2,y3)

Use

To Calculate Area of triangle

Arguments

This function takes 3 arguments such that (x1,y1), (x2,y2), (x3,y3) are coordinates of the triangle.

return type

float value of area of triangle

```
def area(x1, y1, x2, y2, x3, y3):  
    # formula to calculate area of triangle  
    return abs((x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)) / 2.0)
```

delaunay(pts , coord)

Use

To check whether the three points passed in pts form a triangle according to delaunay condition or not

Arguments

This function takes 2 arguments as pts and coord.

pts - contains three points that represent the coordinates of the triangle on which the condition is to be checked.

coord - control points coordinate

return

boolean value - whether the passed triangle points fulfill the delaunay condition or not

True - Fulfilled

False - Not Fulfilled

```
def delaunay(pts, coord):  
    # Checking whether the points are collinear or not  
    area_tri =  
    area(pts[0][0], pts[0][1], pts[1][0], pts[1][1], pts[2][0], pts[2][1])  
    if area_tri == 0:  
        return False
```

```

# Finding the circumcenter of the selected points for triangle
c = circumcenter(pts[0],pts[1],pts[2])

#radius of the circumcircle by p1,p2,p3
radius = dist(c,pts[0])

#Getting the distance of remaining points and checking delaunay
condition

for i in range(len(coord)):
    if coord[i] not in pts:
        d = dist(c,coord[i])
        if d < radius:
            return False

return True

```

combinations(coord)

Use

To generate the combinations of points to create triangles.

Arguments

This function takes 1 argument coord the list of control points provided by the user for the Delaunay triangulation.

return type

returns the list of all the combinations of the points.

```

def combinations(coord):
    comb = []
    for i in range(len(coord)):
        for j in range(len(coord)):
            for k in range(len(coord)):
                if(i!=j&j!=k&k!=i):
                    t = (coord[i],coord[j],coord[k])
                    comb.append(t)
    return comb

```

get_Triangle_List(coord)

Use

To check which combinations of points satisfy the delaunay condition.

Arguments

This function takes 1 argument coord the list of control points provided by the user for the Delaunay triangulation.

return type

returns the list of the combinations of the points that satisfy the delaunay condition

```
def get_Triangle_List(coord):
    comb = combinations(coord)
    triangle_list = []
    for i in comb:
        cond = delaunay(i, coord)
        if cond == True:
            triangle_list.append(i)

    return triangle_list
```

CallBackFuncForimg1(event,x,y,flags,param)

To get callback coordinates and draw circle at points clicked in source image which are the control points in source image

```
def CallBackFuncForimg1(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(im1, (x,y), 1, (0, 0, 255), 2)
        coordSrc.append((y,x))
```

CallBackFuncForimg2(event,x,y,flags,param)

To get callback coordinates and draw circle at points clicked in destination image which are the control points in destination image

```
def CallBackFuncForimg2(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(im2, (x,y), 1, (255, 0, 0), 2)
        coordDest.append((y,x))
```

getcoord(window,image)

To get the coordinates from the user using the left mouse button click and storing that value in a list. One can select as many control points along with border points but it should be greater than 3

```
def getcoord(window,image):  
    while (True):  
        cv2.imshow(window, image)  
        if cv2.waitKey(20) == 27:  
            break  
    cv2.destroyAllWindows()
```

draw_delaunay(img,triangleList,delaunay_color)

Use

To display the valid delaunay triangle in the image.

Arguments

This function takes 3 arguments as img, triangleList and delaunay_color

img - The image on which we have to draw the triangles

triangleList - The list coordinates of the valid triangles.

delaunay_color - the color of the lines of the triangles.

return type

returns the image having the triangle.

```
def draw_delaunay(img, triangleList,delaunay_color):  
    tri=[]  
  
    for t in triangleList :  
        pt1 = t[0]  
        pt2 = t[1]  
        pt3 = t[2]  
        cv2.line(img, (pt1[1],pt1[0]), (pt2[1],pt2[0]), delaunay_color, 1)  
        cv2.line(img, (pt2[1],pt2[0]), (pt3[1],pt3[0]), delaunay_color, 1)  
        cv2.line(img, (pt3[1],pt3[0]), (pt1[1],pt1[0]), delaunay_color, 1)  
        a=[]  
        a.append(pt1)  
        a.append(pt2)  
        a.append(pt3)
```

```
tri.append(a)
return tri
```

showTriangulated(img1,img2)

Use

To display the valid delaunay triangle in the image.

Arguments

This function takes 2 arguments as img1 and img2

img1 = source image

img2 = destination image

return type

returns the image having the triangle for further process.

```
def showTriangulated(img1,img2):
    size = img1.shape
    r = (0, 0, size[1], size[0])

    coord = coordSrc.copy()

    triangleList = get_Triangle_List(coord)

    tri1 = draw_delaunay(img1,triangleList,(255,0,0))
    # Matching the point p0,..,pn of the source and destination image
    tri2 = []
    for i in range(len(tri1)):
        a = []
        for j in range(len(tri1[i])):
            a.append(coordDest[coordSrc.index(tri1[i][j])])
        tri2.append(a)

    tri2 = draw_delaunay(img2,tri2,(0,255,255))

    cv2.imshow("src",img1)
    cv2.imshow("dest",img2)
    cv2.imwrite("src.jpg",img1)
    cv2.imwrite("dest.jpg",img2)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    return tri1,tri2
```


isInsideTriangle(p1,p2,p3,x,y)

Use

To check if some point lie inside the triangle or not.

Arguments

This function takes 5 arguments such that p1, p2, p3 are coordinates of the triangle and (x,y) is the point which we want to check.

return type

bool value

True - point lies inside the triangle.

False - point doesn't lie inside the triangle.

```
def isInsideTriangle(p1,p2,p3,x,y):  
  
    A = area (p1[0], p1[1], p2[0], p2[1], p3[0], p3[1])  
    A1 = area (x, y, p2[0], p2[1], p3[0], p3[1])  
    A2 = area (p1[0], p1[1], x, y, p3[0], p3[1])  
    A3 = area (p1[0], p1[1], p2[0], p2[1], x, y)  
    if(A == A1 + A2 + A3):  
        return True  
    else:  
        return False
```

get_affine_basis(coord)

Use

To Calculate the affine basis

Arguments

This function takes only 1 argument which contains the coordinates of the triangle

return type

float value of x and y component of both the affine basis of a triangle

```
def get_affine_basis(coord):  
    e1x = coord[1][0]-coord[0][0]  
    e1y = coord[1][1]-coord[0][1]  
    e2x = coord[2][0]-coord[0][0]  
    e2y = coord[2][1]-coord[0][1]  
    return e1x,e1y,e2x,e2y
```

`get_intermediate_triangles(srcTri,destTri,k,n)`

Use

To find the coordinates of the triangle in kth intermediate image corresponding to the triangle in Source and Destination Image.

Arguments

This function take 4 arguments

srcTri - coordinates of triangle in source image

destTri - coordinates of triangle in destination image

k - kth intermediate image

n - k+2

return type

return the coordinates of the triangle in intermediate image

```
def get_intermediate_triangles(srcTri , destTri , k , n):  
    intTri=[]  
    for (st,dt) in zip(srcTri,destTri):  
        a=[]  
        for (coordS,coordD) in zip(st,dt):  
            xi=int(((n-k)/n)*coordS[0]+(k/n)*coordD[0])  
            yi=int(((n-k)/n)*coordS[1]+(k/n)*coordD[1])  
            a.append((xi,yi))  
        intTri.append(a)  
    return intTri
```

`checkRange(sx,sy,dx,dy)`

Use

if sx,sy,dx,dy are out of range i.e if they are negative or greater than the size of image so this function normalize them

Arguments

This function take 4 arguments

(sx,sy) - coordinate in source image

(dx,dy) - coordinate in destination image

return type

return the normalized coordinates

```
def checkRange(sx , sy , dx , dy):
    if sx<0:
        sx=0
    if dx<0:
        dx=0
    if sy<0:
        sy=0
    if dy<0:
        dy=0
    if sx>img1.shape[0]-1:
        sx=img1.shape[0]-1
    if dx>img2.shape[0]-1:
        dx=img2.shape[0]-1
    if sy>img1.shape[1]-1:
        sy=img1.shape[1]-1
    if dy>img2.shape[1]-1:
        dy=img2.shape[1]-1
    return sx,sy,dx,dy
```

morph(no_of_intermed)

Use

To do affine Transformation from source image to destination image by making some intermediate images in which pixel value are calculated by combination of pixel value in source and destination image

Arguments

This function takes only 1 argument which is the number of intermediate images we want to make.

```
def morph(no_of_intermed):
    n=no_of_intermed+2

    for k in range(1,no_of_intermed+1):

        print(str(k)+" intermediate is generating it may take some time
Please Wait...")
        inter=np.zeros_like(img1,dtype=np.uint8)
        row,col,channel=inter.shape
```

```

intTri=get_intermediate_triangles(tri1,tri2,k,n)

for ( s_tri , i_tri , d_tri ) in zip( tri1 , intTri , tri2 ):

    src_e1x , src_e1y , src_e2x , src_e2y = get_affine_basis(s_tri)
    int_e1x , int_e1y , int_e2x , int_e2y = get_affine_basis(i_tri)
    dest_e1x , dest_e1y , dest_e2x , dest_e2y =
get_affine_basis(d_tri)

    for r in range(row):
        for c in range(col):
            if isInsideTriangle(i_tri[0],i_tri[1],i_tri[2],r,c):

                X = r-i_tri[0][0]
                Y = c-i_tri[0][1]

alpha=((int_e2y*X)-(Y*int_e2x))/((int_e1x*int_e2y)-(int_e2x*int_e1y))

beta=((int_e1y*X)-(Y*int_e1x))/((int_e1y*int_e2x)-(int_e2y*int_e1x))

dest_x=int(alpha*dest_e1x+beta*dest_e2x+d_tri[0][0])

dest_y=int(alpha*dest_e1y+beta*dest_e2y+d_tri[0][1])

src_x=int(alpha*src_e1x+beta*src_e2x+s_tri[0][0])
src_y=int(alpha*src_e1y+beta*src_e2y+s_tri[0][1])

src_x,src_y,dest_x,dest_y=checkRange(src_x,src_y,dest_x,dest_y)

inter[r][c][0]=int(((n-k)/n)*img1[src_x][src_y][0]
                    +(k/n)*img2[dest_x][dest_y][0])
inter[r][c][1]=int(((n-k)/n)*img1[src_x][src_y][1]
                    +(k/n)*img2[dest_x][dest_y][1])
inter[r][c][2]=int(((n-k)/n)*img1[src_x][src_y][2]
                    +(k/n)*img2[dest_x][dest_y][2])

#         cv2.imshow("inter"+str(k),inter)
name="inter"+str(k)+".jpg"

```

```
cv2.imwrite(name, inter)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Reading of input images and resizing them to same size

```
img1=cv2.imread("bush.jpg")img2=cv2.imread("clinton.jpg")

""" below commented two lines are used when we have pass name of images as
command line argument"""
# img1=cv2.imread(str(sys.argv[1]))
# img2=cv2.imread(str(sys.argv[2]))
img2 = cv2.resize(img2,(img1.shape[1],img1.shape[0]))
im1=np.copy(img1)
im2=np.copy(img2)
window1 = 'image1'
window2= 'image2'
coordSrc=[]
coordDest=[]
```

Getting control points on images using mouse click

```
cv2.namedWindow(window1)cv2.setMouseCallback(window1, CallBackFuncForimg1)
getcoord(window1,im1)

cv2.namedWindow(window2)
cv2.setMouseCallback(window2, CallBackFuncForimg2)
getcoord(window2,im2)

r1,c1,ch1= img1.shape
r2,c2,ch2 = img2.shape
```

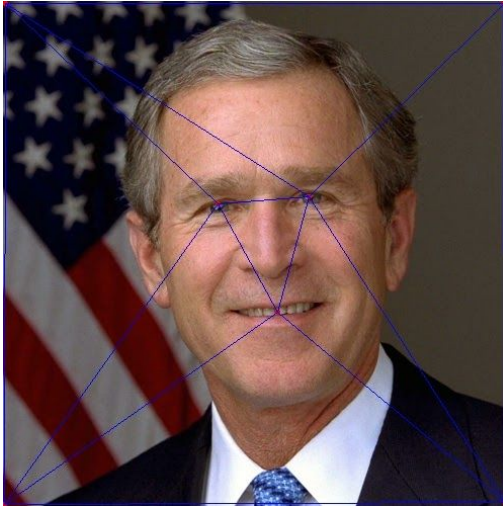
Triangulating the images and applying affine transformation

```
tri1,tri2 = showTriangulated(im1,im2)
morph(int(input("Enter number of intermediate you want ")))
```

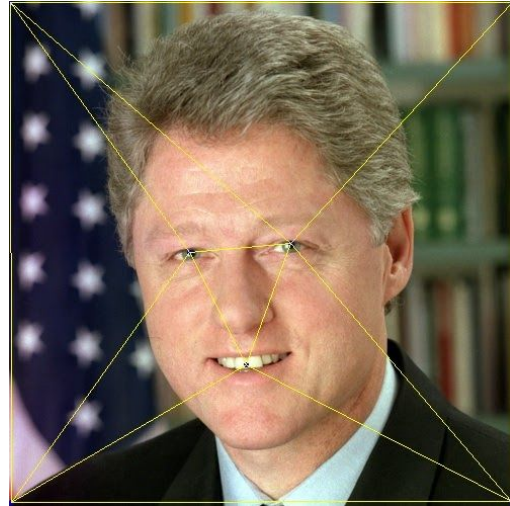
Results and Conclusion

Triangulated Images and Intermediate images

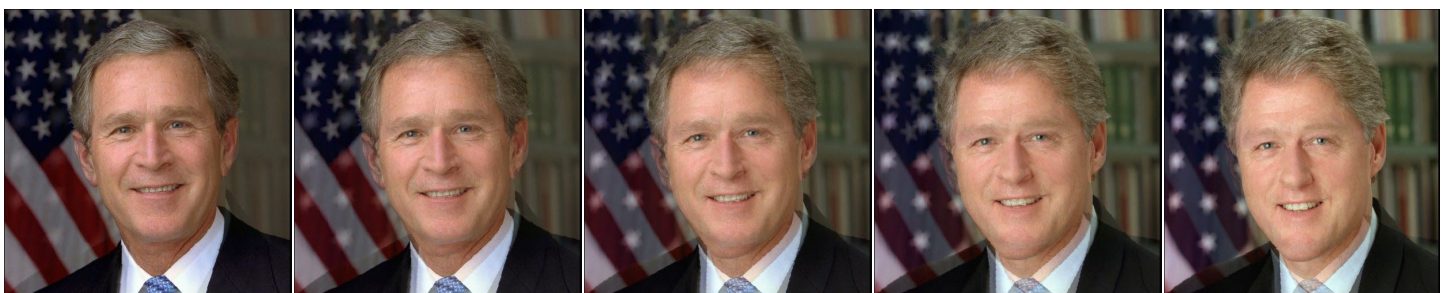
Source Image




Destination Image



5 Intermediate Images



[Video link](#) (created with 50 frames at 30fps)



We were able to apply the Delaunay triangulation and Affine transformation without using the inbuilt functions of the OpenCV library. This gave an insight into how a library functions are built. Along with it, we were able to apply various mathematical formulas for the various steps that were required for the computation of the above-mentioned algorithms. There were two major problems that we encountered implementing the algorithm -

1) The first was that we were obtaining holes while performing the affine transformation as we were doing forward mapping. This was resolved by doing reverse mapping on the image.

2) Second was that of maintaining the list of the corresponding control points of the two images as they were getting changed after applying the certain functions. This was handled by changing the list of the second image according to the arrangement of the points in the list of first image.

